NYC Taxi EDA - Update: The fast & the curious

Code

2017-08-05

1 Introduction

This is a comprehensive Exploratory Data Analysis for the New York City Taxi Trip Duration (https://www.kaggle.com/c/nyc-taxi-trip-duration) competition with tidy R (http://tidyverse.org/) and ggplot2 (http://ggplot2.tidyverse.org/).

The goal of this playground challenge is to predict the *duration of taxi rides in NYC* based on features like trip coordinates or pickup date and time. The data (https://www.kaggle.com/c/nyc-taxi-trip-duration/data) comes in the shape of 1.5 million training observations (`../input/train.csv`) and 63k test observation (`../input/test.csv`). Each row contains one taxi trip.

In this notebook, we will first study and visualise the original data, engineer new features, and examine potential outliers. Then we add two **external data sets** on the NYC weather (https://www.kaggle.com/mathijs/weather-data-in-new-york-city-2016) and on the theoretically fastest routes (https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm). We visualise and analyse the new features within these data sets and their impact on the target *trip_duration* values. Finally, we will make a brief excursion into viewing this challenge as a **classification problem** and finish this notebook with a **simple XGBoost model** that provides a basic prediction (final part under construction).

I hope that this notebook will help you in getting started with this challenge. There is lots of room for you to develop your own ideas for new features and visualisations. In particular the classification approach and the final model are only a basic starting point for you to improve and optimise them for better performance. As always, any feedback, questions, or constructive criticism are much appreciated.

A **note on hidden figures:** Due to the memory/run-time limitations of the Kaggle kernels I had to disable a few auxilliary visualisations as the notebook grew towards its final stage. I tried to only remove those plots that didn't affect the flow of the analysis too much. However,

all of the corresponding code is still contained in this notebook and you can easily recover these hidden plots when running the script locally (or forking it and only running a part of it in the kernel) by removing the `eval=FALSE` option in the header of the code block. I will point out hidden plots in the text. (Also, the movie hidden figures (http://www.imdb.com/title/tt4846340/) is a pretty awesome piece of history for maths (and space) geeks like us ;-))

Finally, I would like to **thank everyone of you** for viewing, upvoting, or commenting on this kernel! I'm still a beginner here on Kaggle and your support means a lot to me :-) . Also, thanks to NockedDown (https://www.kaggle.com/nockeddown) for suggesting the new update title pun in the comments! ;-)

Enough talk. Let's get started:

1.1 Load libraries and helper functions

Hide

```
library('ggplot2') # visualisation
library('scales') # visualisation
library('grid') # visualisation
library('RColorBrewer') # visualisation
library('corrplot') # visualisation
library('alluvial') # visualisation
library('dplyr') # data manipulation
library('readr') # input/output
library('data.table') # data manipulation
library('tibble') # data wrangling
library('tidyr') # data wrangling
library('stringr') # string manipulation
library('forcats') # factor manipulation
library('lubridate') # date and time
library('geosphere') # geospatial locations
library('leaflet') # maps
library('leaflet.extras') # maps
library('maps') # maps
library('xgboost') # modelling
library('caret') # modelling
```

We use the *multiplot* function, courtesy of R Cookbooks (http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/) to create multi-panel plots.

Hide

```
# Define multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot obj
ects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))
```

```
      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

1.2 Load data

We use *data.table's* fread function to speed up reading in the data:

Hide

```
train <- as.tibble(fread('../input/nyc-taxi-trip-duration/train.csv'))
test <- as.tibble(fread('../input/nyc-taxi-trip-duration/test.csv'))
sample_submit <- as.tibble(fread('../input/nyc-taxi-trip-duration/sample_submi
ssion.csv'))
```

1.3 File structure and content

Let's have an overview of the data sets using the *summary* and *glimpse* tools. First the training data:

Hide

```
summary(train)
```

```
##       id              vendor_id     pickup_datetime    dropoff_datetime
##  Length:1458644     Min.   :1.000   Length:1458644     Length:1458644
##  Class :character   1st Qu.:1.000   Class :character   Class :character
##  Mode  :character   Median :2.000   Mode  :character   Mode  :character
##                     Mean   :1.535
##                     3rd Qu.:2.000
##                     Max.   :2.000
##  passenger_count pickup_longitude  pickup_latitude dropoff_longitude
##  Min.   :0.000   Min.   :-121.93   Min.   :34.36   Min.   :-121.93
##  1st Qu.:1.000   1st Qu.: -73.99   1st Qu.:40.74   1st Qu.: -73.99
##  Median :1.000   Median : -73.98   Median :40.75   Median : -73.98
##  Mean   :1.665   Mean   : -73.97   Mean   :40.75   Mean   : -73.97
##  3rd Qu.:2.000   3rd Qu.: -73.97   3rd Qu.:40.77   3rd Qu.: -73.96
##  Max.   :9.000   Max.   : -61.34   Max.   :51.88   Max.   : -61.34
##  dropoff_latitude store_and_fwd_flag trip_duration
##  Min.   :32.18    Length:1458644     Min.   :      1
##  1st Qu.:40.74    Class :character   1st Qu.:    397
##  Median :40.75    Mode  :character   Median :    662
##  Mean   :40.75                       Mean   :    959
##  3rd Qu.:40.77                       3rd Qu.:   1075
##  Max.   :43.92                       Max.   :3526282
```

Hide

```
glimpse(train)
```

```
## Observations: 1,458,644
## Variables: 11
## $ id                <chr> "id2875421", "id2377394", "id3858529", "id3...
## $ vendor_id         <int> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2...
## $ pickup_datetime   <chr> "2016-03-14 17:24:55", "2016-06-12 00:43:35...
## $ dropoff_datetime  <chr> "2016-03-14 17:32:30", "2016-06-12 00:54:38...
## $ passenger_count   <int> 1, 1, 1, 1, 1, 6, 4, 1, 1, 1, 1, 4, 2, 1, 1...
## $ pickup_longitude  <dbl> -73.98215, -73.98042, -73.97903, -74.01004,...
## $ pickup_latitude   <dbl> 40.76794, 40.73856, 40.76394, 40.71997, 40....
## $ dropoff_longitude <dbl> -73.96463, -73.99948, -74.00533, -74.01227,...
## $ dropoff_latitude  <dbl> 40.76560, 40.73115, 40.71009, 40.70672, 40....
## $ store_and_fwd_flag <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"...
## $ trip_duration     <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
```

And then the testing data:

Hide

```
summary(test)
```

```
##       id               vendor_id     pickup_datetime    passenger_count
##  Length:625134      Min.   :1.000   Length:625134      Min.   :0.000
##  Class :character   1st Qu.:1.000   Class :character   1st Qu.:1.000
##  Mode  :character   Median :2.000   Mode  :character   Median :1.000
##                     Mean   :1.535                      Mean   :1.662
##                     3rd Qu.:2.000                      3rd Qu.:2.000
##                     Max.   :2.000                      Max.   :9.000
##  pickup_longitude  pickup_latitude dropoff_longitude dropoff_latitude
##  Min.   :-121.93   Min.   :37.39   Min.   :-121.93   Min.   :36.60
##  1st Qu.: -73.99   1st Qu.:40.74   1st Qu.: -73.99   1st Qu.:40.74
##  Median : -73.98   Median :40.75   Median : -73.98   Median :40.75
##  Mean   : -73.97   Mean   :40.75   Mean   : -73.97   Mean   :40.75
##  3rd Qu.: -73.97   3rd Qu.:40.77   3rd Qu.: -73.96   3rd Qu.:40.77
##  Max.   : -69.25   Max.   :42.81   Max.   : -67.50   Max.   :48.86
##  store_and_fwd_flag
##  Length:625134
##  Class :character
##  Mode  :character
##
##
##
```

Hide

```
glimpse(test)
```

```
## Observations: 625,134
## Variables: 9
## $ id                <chr> "id3004672", "id3505355", "id1217141", "id2...
## $ vendor_id         <int> 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1...
## $ pickup_datetime   <chr> "2016-06-30 23:59:58", "2016-06-30 23:59:53...
## $ passenger_count   <int> 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 4, 1, 1, 1, 1...
## $ pickup_longitude  <dbl> -73.98813, -73.96420, -73.99744, -73.95607,...
## $ pickup_latitude   <dbl> 40.73203, 40.67999, 40.73758, 40.77190, 40....
## $ dropoff_longitude <dbl> -73.99017, -73.95981, -73.98616, -73.98643,...
## $ dropoff_latitude  <dbl> 40.75668, 40.65540, 40.72952, 40.73047, 40....
## $ store_and_fwd_flag <chr> "N", "N", "N", "N", "N", "N", "N", "N", "N"...
```

We find:

- *vendor_id* only takes the values 1 or 2, presumably to differentiate two taxi companies
- *pickup_datetime* and (in the training set) *dropoff_datetime* are combinations of date and time that we will have to re-format into a more useful shape
- *passenger_count* takes a median of 1 and a maximum of 9 in both data sets
- The *pickup/dropoff_longitute/latitute* describes the geographical coordinates where the meter was activate/deactivated.
- *store_and_fwd_flag* is a flag that indicates whether the trip data was sent immediately to the vendor ("N") or held in the memory of the taxi because there was no connection to the server ("Y"). Maybe there could be a correlation with certain geographical areas with bad reception?
- *trip_duration:* our target feature in the training data is measured in seconds.

1.4 Missing values

Knowing about missing values is important because they indicate how much we don't know about our data. Making inferences based on just a few cases is often unwise. In addition, many modelling procedures break down when missing values are involved and the corresponding rows will either have to be removed completely or the values need to be estimated somehow.

Here, we are in the fortunate position that our data is complete and there are no missing values:

Hide

```
sum(is.na(train))
```

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

Hide

```
sum(is.na(test))
```

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

1.5 Combining train and test

In preparation for our eventual modelling analysis we combine the *train* and *test* data sets into a single one. I find it generally best not to examine the *test* data too closely, since this bears the risk of overfitting your analysis to this data. However, a few simple consistency checks between the two data sets can be of advantage.

Hide

```
combine <- bind_rows(train %>% mutate(dset = "train"),
                     test %>% mutate(dset = "test",
                                     dropoff_datetime = NA,
                                     trip_duration = NA))
combine <- combine %>% mutate(dset = factor(dset))
```

1.6 Reformating features

For our following analysis, we will turn the data and time from characters into *date* objects. We also recode *vendor_id* as a factor. This makes it easier to visualise relationships that involve these features.

Hide

```
train <- train %>%
  mutate(pickup_datetime = ymd_hms(pickup_datetime),
         dropoff_datetime = ymd_hms(dropoff_datetime),
         vendor_id = factor(vendor_id),
         passenger_count = factor(passenger_count))
```

1.7 Consistency check

It is worth checking whether the *trip_durations* are consistent with the intervals between the *pickup_datetime* and *dropoff_datetime*. Presumably the former were directly computed from the latter, but you never know. Below, the *check* variable shows “TRUE” if the two intervals are not consistent:

Hide

```
train %>%
  mutate(check = abs(int_length(interval(dropoff_datetime,pickup_datetime)) +
trip_duration) > 0) %>%
  select(check, pickup_datetime, dropoff_datetime, trip_duration) %>%
  group_by(check) %>%
  count()
```

```
## # A tibble: 1 x 2
## # Groups:   check [1]
##   check       n
##   <lgl>   <int>
## 1 FALSE 1458644
```

And we find that everything fits perfectly.

2 Individual feature visualisations

Visualisations of feature distributions and their relations are key to understanding a data set, and they often open up new lines of inquiry. I always recommend to examine the data from as many different perspectives as possible to notice even subtle trends and correlations.

In this section we will begin by having a look at the distributions of the individual data features.

We start with a map of NYC and overlay a managable number of pickup coordinates to get a general overview of the locations and distances in question. For this visualisation we use the leaflet (https://rstudio.github.io/leaflet/) package, which includes a variety of cool tools for interactive maps. In this map you can zoom and pan through the pickup locations:

Hide

```
set.seed(1234)
foo <- sample_n(train, 8e3)

leaflet(data = foo) %>% addProviderTiles("Esri.NatGeoWorldMap") %>%
  addCircleMarkers(~ pickup_longitude, ~pickup_latitude, radius = 1,
                   color = "blue", fillOpacity = 0.3)
```

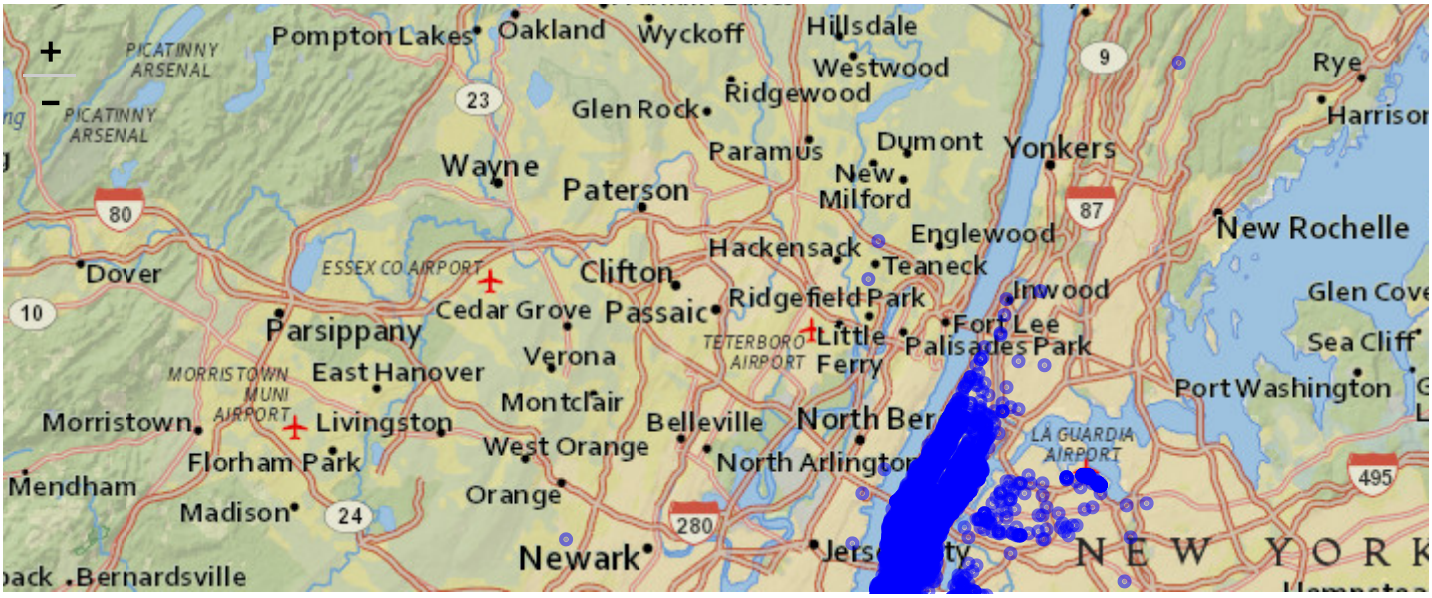

Fig. 1

In turns out that almost all of our trips were in fact taking place in Manhattan only. Another notable hot-spot is JFK airport towards the south-east of the city.

The map gives us an idea what some of the our distributions could look like. Let's start with plotting the target feature *trip_duration*:

Hide

```
train %>%
  ggplot(aes(trip_duration)) +
  geom_histogram(fill = "red", bins = 150) +
  scale_x_log10() +
  scale_y_sqrt()
```

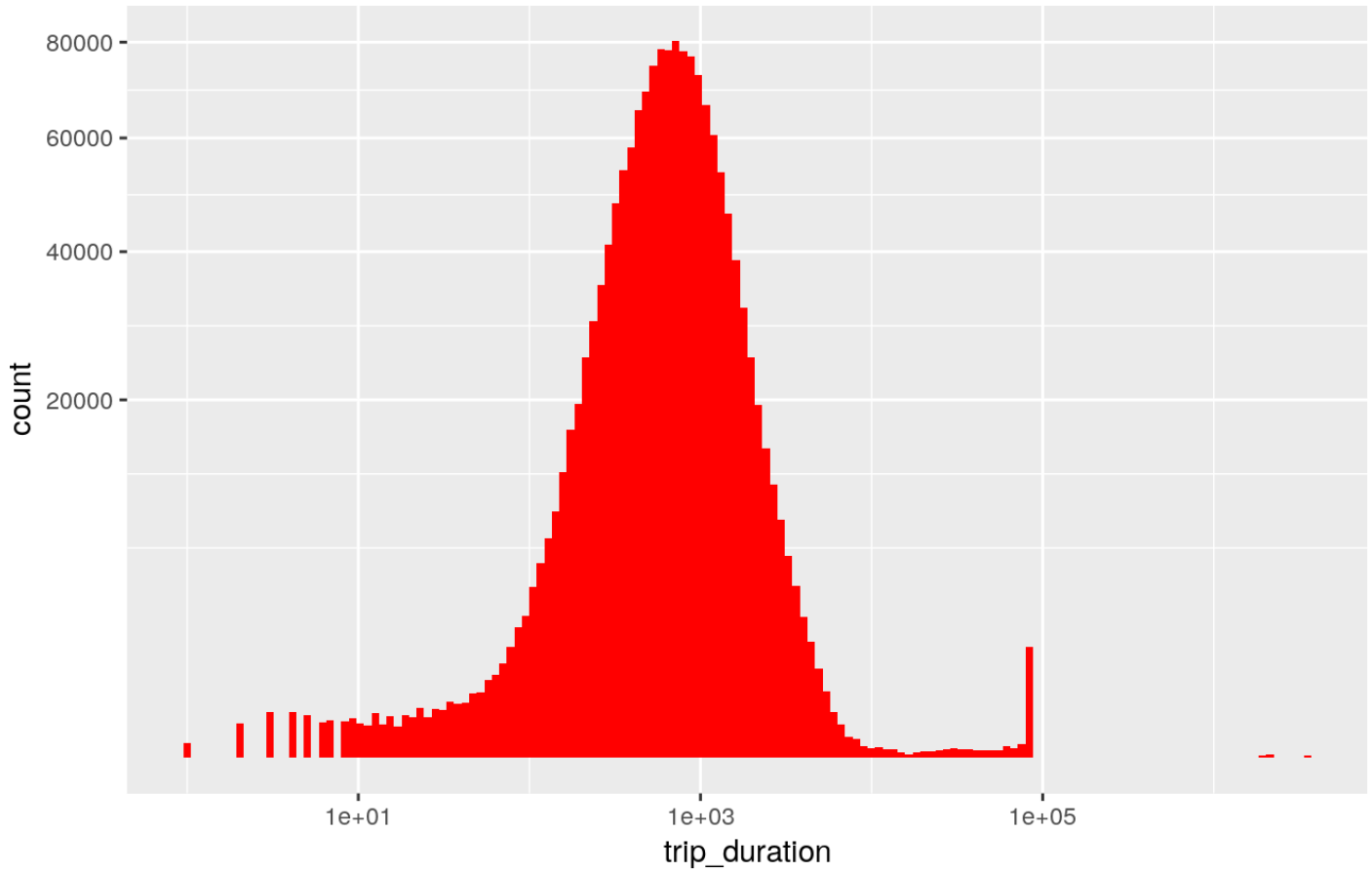

Fig. 2

Note the logarithmic x-axis and square-root y-axis.

We find:

- the majority of rides follow a rather smooth distribution that looks almost log-normal with a peak just short of 1000 seconds, i.e. about 27 minutes.
- There are several suspiciously short rides with less than 10 seconds duration.
- Additionally, there is a strange delta-shaped peak of *trip_duration* just before the 1e5 seconds mark and even a few way above it:

Hide

```
train %>%
  arrange(desc(trip_duration)) %>%
  select(trip_duration, pickup_datetime, dropoff_datetime, everything()) %>%
  head(10)
```

```
## # A tibble: 10 x 11
##    trip_duration     pickup_datetime    dropoff_datetime        id
##            <int>              <dttm>              <dttm>     <chr>
##  1       3526282 2016-02-13 22:46:52 2016-03-25 18:18:14 id0053347
##  2       2227612 2016-01-05 06:14:15 2016-01-31 01:01:07 id1325766
##  3       2049578 2016-02-13 22:38:00 2016-03-08 15:57:38 id0369307
##  4       1939736 2016-01-05 00:19:42 2016-01-27 11:08:38 id1864733
##  5         86392 2016-02-15 23:18:06 2016-02-16 23:17:58 id1942836
##  6         86391 2016-05-31 13:00:39 2016-06-01 13:00:30 id0593332
##  7         86390 2016-05-06 00:00:10 2016-05-07 00:00:00 id0953667
##  8         86387 2016-06-30 16:37:52 2016-07-01 16:37:39 id2837671
##  9         86385 2016-06-23 16:01:45 2016-06-24 16:01:30 id1358458
## 10         86379 2016-05-17 22:22:56 2016-05-18 22:22:35 id2589925
## # ... with 7 more variables: vendor_id <fctr>, passenger_count <fctr>,
## #   pickup_longitude <dbl>, pickup_latitude <dbl>,
## #   dropoff_longitude <dbl>, dropoff_latitude <dbl>,
## #   store_and_fwd_flag <chr>
```

Those records would correspond to 24-hour trips and beyond, with a maximum of almost 12 days. I know that rush hour can be bad, but those values are a little unbelievable.

Over the year, the distributions of *pickup_datetime* and *dropoff_datetime* look like this:

Hide

```
p1 <- train %>%
  ggplot(aes(pickup_datetime)) +
  geom_histogram(fill = "red", bins = 120) +
  labs(x = "Pickup dates")

p2 <- train %>%
  ggplot(aes(dropoff_datetime)) +
  geom_histogram(fill = "blue", bins = 120) +
  labs(x = "Dropoff dates")

layout <- matrix(c(1,2),2,1,byrow=FALSE)
multiplot(p1, p2, layout=layout)
```

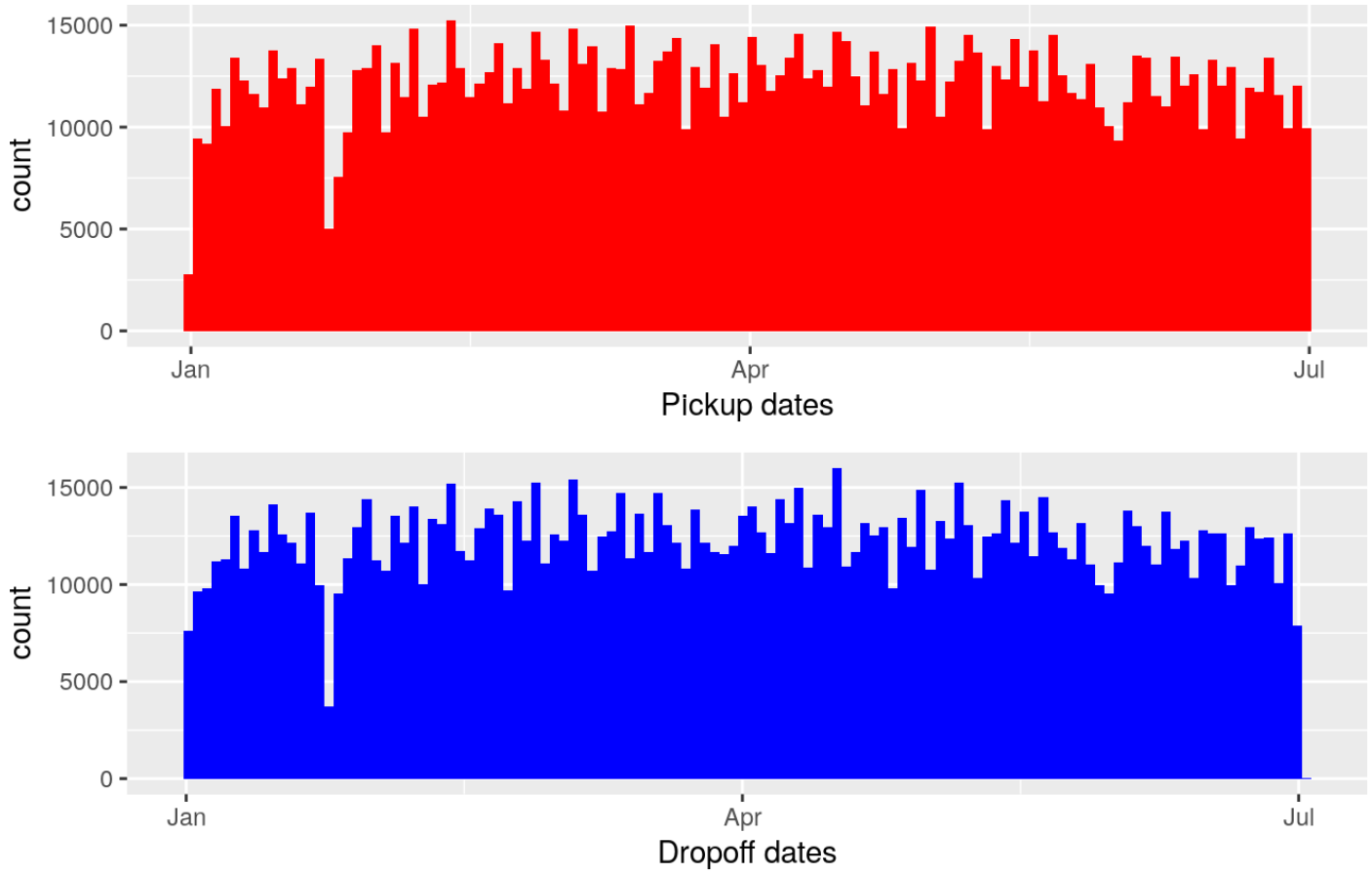

Fig. 3

Hide

```
p1 <- 1; p2 <- 1
```

Fairly homogeneous, covering half a year between January and July 2016. There is an interesting drop around late January early February (hidden plot zoom in).

Hide

```
train %>%
  filter(pickup_datetime > ymd("2016-01-20") & pickup_datetime < ymd("2016-02-
10")) %>%
  ggplot(aes(pickup_datetime)) +
  geom_histogram(fill = "red", bins = 120)
```

That's winter in NYC, so maybe snow storms or other heavy weather? Events like this should be taken into account, maybe through some handy external data set?

In the plot above we can already see some daily and weekly modulations in the number of trips. Let's investigate these variations together with the distributions of *passenger_count* and *vendor_id* by creating a multi-plot panel with different components:

Hide

```
p1 <- train %>%
  group_by(passenger_count) %>%
  count() %>%
  ggplot(aes(passenger_count, n, fill = passenger_count)) +
  geom_col() +
  scale_y_sqrt() +
  theme(legend.position = "none")

p2 <- train %>%
  ggplot(aes(vendor_id, fill = vendor_id)) +
  geom_bar() +
  theme(legend.position = "none")

p3 <- train %>%
  ggplot(aes(store_and_fwd_flag)) +
  geom_bar() +
  theme(legend.position = "none") +
  scale_y_log10()

p4 <- train %>%
  mutate(wday = wday(pickup_datetime, label = TRUE)) %>%
  group_by(wday, vendor_id) %>%
  count() %>%
  ggplot(aes(wday, n, colour = vendor_id)) +
  geom_point(size = 4) +
  labs(x = "Day of the week", y = "Total number of pickups") +
  theme(legend.position = "none")

p5 <- train %>%
  mutate(hpick = hour(pickup_datetime)) %>%
  group_by(hpick, vendor_id) %>%
  count() %>%
  ggplot(aes(hpick, n, color = vendor_id)) +
  geom_point(size = 4) +
  labs(x = "Hour of the day", y = "Total number of pickups") +
  theme(legend.position = "none")

layout <- matrix(c(1,2,3,4,5,5),3,2,byrow=TRUE)
multiplot(p1, p2, p3, p4, p5, layout=layout)
```

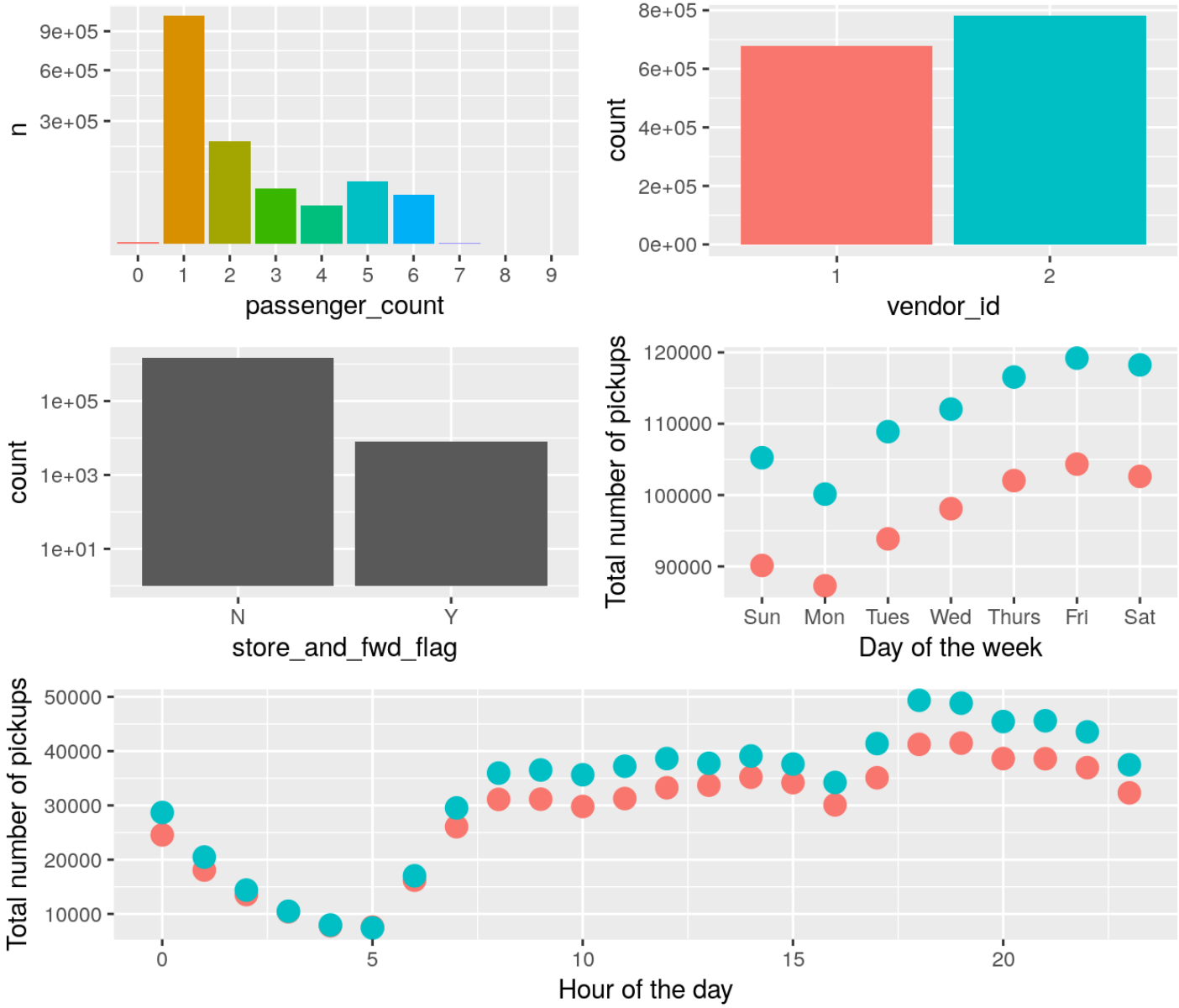

Fig. 4

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1; p5 <- 1
```

We find:

- There are a few trips with zero, or seven to nine passengers but they are a rare exception:

Hide

```
train %>%
  group_by(passenger_count) %>%
  count()
```

```
## # A tibble: 10 x 2
## # Groups:   passenger_count [10]
##    passenger_count       n
##             <fctr>   <int>
##  1               0      60
##  2               1 1033540
##  3               2  210318
##  4               3   59896
##  5               4   28404
##  6               5   78088
##  7               6   48333
##  8               7       3
##  9               8       1
## 10               9       1
```

- The vast majority of rides had only a single passenger, with two passengers being the (distant) second most popular option.
- Towards larger passenger numbers we are seeing a smooth decline through 3 to 4, until the larger crowds (and larger cars) give us another peak at 5 to 6 passengers.
- Vendor 2 has significantly more trips in this data set than vendor 1 (note the logarithmic y-axis). This is true for every day of the week.
- We find an interesting pattern with Monday being the quietest day and Friday very busy. This is the same for the two different vendors, with *vendor_id == 2* showing significantly higher trip numbers.
- As one would intuitively expect, there is a strong dip during the early morning hours. There we also see not much difference between the two vendors. We find another dip around 4pm and then the numbers increase towards the evening.
- The *store_and_fwd_flag* values, indicating whether the trip data was sent immediately to the vendor (“N”) or held in the memory of the taxi because there was no connection to the server (“Y”), show that there was almost no storing taking place (note again the logarithmic y-axis):

Hide

```
train %>%
  group_by(store_and_fwd_flag) %>%
  count()
```

```
## # A tibble: 2 x 2
## # Groups:   store_and_fwd_flag [2]
##   store_and_fwd_flag       n
##                <chr>   <int>
## 1                  N 1450599
## 2                  Y    8045
```

These numbers are equivalent to about half a percent of trips not being transmitted immediately.

The trip volume per hour of the day depends somewhat on the month and strongly on the day of the week:

Hide

```
p1 <- train %>%
  mutate(hpick = hour(pickup_datetime),
         Month = factor(month(pickup_datetime, label = TRUE))) %>%
  group_by(hpick, Month) %>%
  count() %>%
  ggplot(aes(hpick, n, color = Month)) +
  geom_line(size = 1.5) +
  labs(x = "Hour of the day", y = "count")

p2 <- train %>%
  mutate(hpick = hour(pickup_datetime),
         wday = factor(wday(pickup_datetime, label = TRUE))) %>%
  group_by(hpick, wday) %>%
  count() %>%
  ggplot(aes(hpick, n, color = wday)) +
  geom_line(size = 1.5) +
  labs(x = "Hour of the day", y = "count")

layout <- matrix(c(1,2),2,1,byrow=FALSE)
multiplot(p1, p2, layout=layout)
```

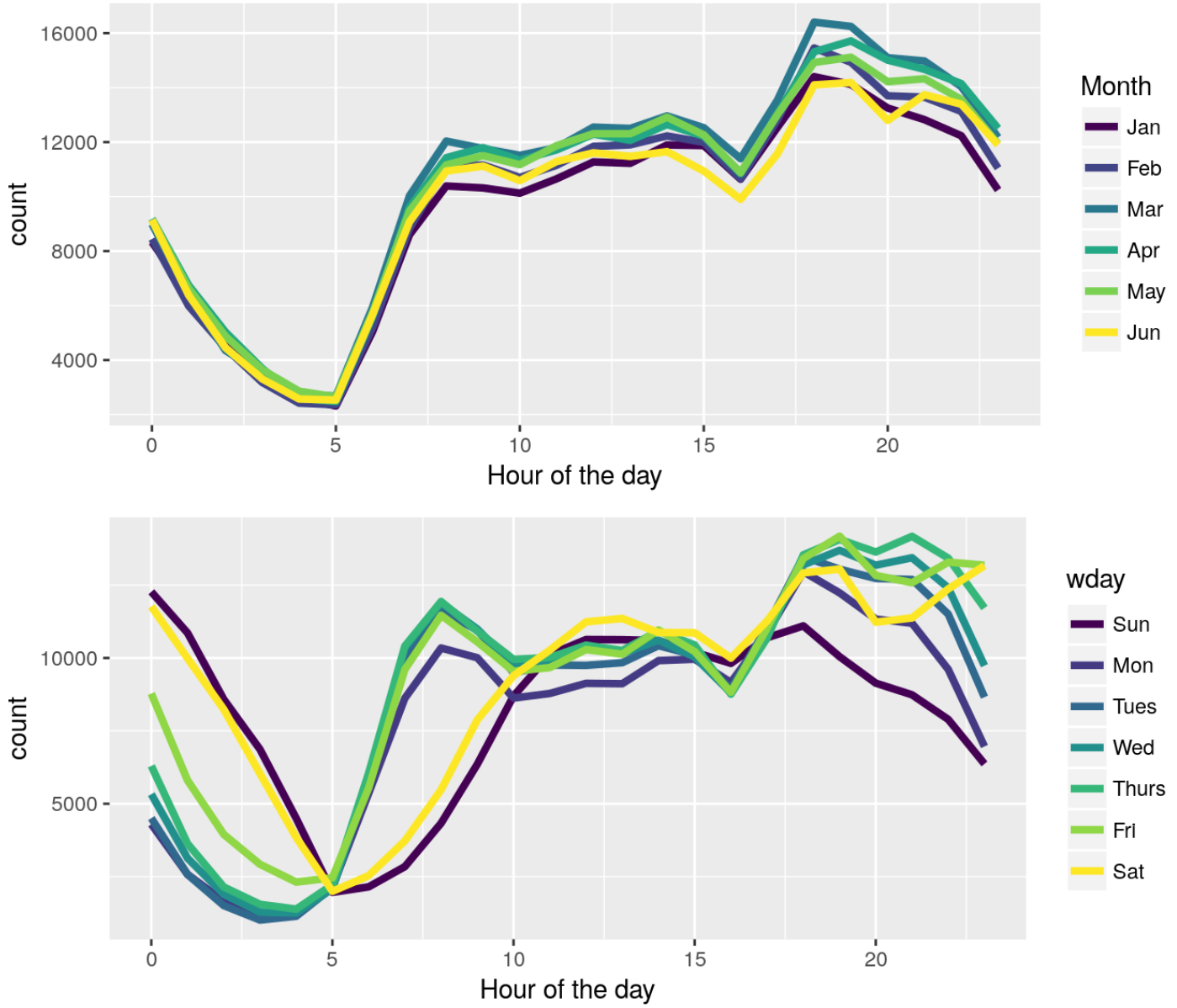

Fig. 5

Hide

```
p1 <- 1; p2 <- 1
```

We find:

- January and June have fewer trips, whereas March and April are busier months. This tendency is observed for both *vendor_ids*.
- The weekend (Sat and Sun, plus Fri to an extend) have higher trip numbers during the early morning ours but lower ones in the morning between 5 and 10, which can most likely be attributed to the contrast between NYC business days and weekend night life. In addition, trip numbers drop on a Sunday evening/night.

Finally, we will look at a simple overview visualisation of the *pickup/dropoff* latitudes and longitudes:

Hide

```
p1 <- train %>%
  filter(pickup_longitude > -74.05 & pickup_longitude < -73.7) %>%
  ggplot(aes(pickup_longitude)) +
  geom_histogram(fill = "red", bins = 40)

p2 <- train %>%
  filter(dropoff_longitude > -74.05 & dropoff_longitude < -73.7) %>%
  ggplot(aes(dropoff_longitude)) +
  geom_histogram(fill = "blue", bins = 40)

p3 <- train %>%
  filter(pickup_latitude > 40.6 & pickup_latitude < 40.9) %>%
  ggplot(aes(pickup_latitude)) +
  geom_histogram(fill = "red", bins = 40)

p4 <- train %>%
  filter(dropoff_latitude > 40.6 & dropoff_latitude < 40.9) %>%
  ggplot(aes(dropoff_latitude)) +
  geom_histogram(fill = "blue", bins = 40)

layout <- matrix(c(1,2,3,4),2,2,byrow=FALSE)
multiplot(p1, p2, p3, p4, layout=layout)
```

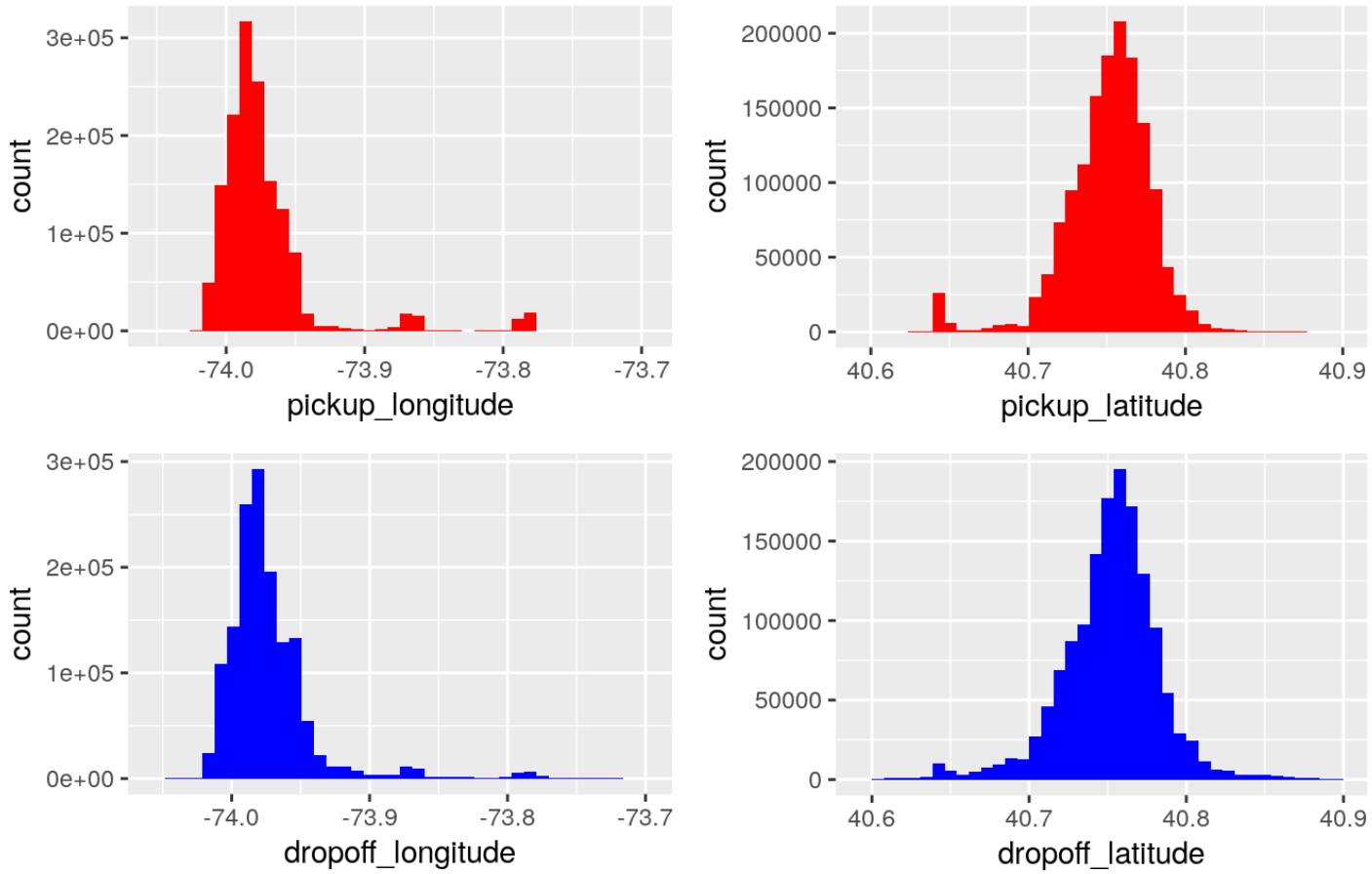

Fig. 6

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1
```

Here we had constrain the range of latitude and longitude values, because there are a few cases which are way outside the NYC boundaries. The resulting distributions are consistent with the focus on Manhattan that we had already seen on the map. These are the most extreme values from the *pickup_latitude* feature:

Hide

```
train %>%
  arrange(pickup_latitude) %>%
  select(pickup_latitude, pickup_longitude) %>%
  head(5)
```

```
## # A tibble: 5 x 2
##   pickup_latitude pickup_longitude
##             <dbl>            <dbl>
## 1        34.35970        -65.84839
## 2        34.71223        -75.35433
## 3        35.08153        -71.79990
## 4        35.31031        -72.07433
## 5        36.02930        -77.44075
```

Hide

```
train %>%
  arrange(desc(pickup_latitude)) %>%
  select(pickup_latitude, pickup_longitude) %>%
  head(5)
```

```
## # A tibble: 5 x 2
##   pickup_latitude pickup_longitude
##             <dbl>            <dbl>
## 1        51.88108        -72.80967
## 2        44.37194        -66.97216
## 3        43.91176        -71.88165
## 4        43.48689        -74.19514
## 5        43.13965        -72.59102
```

We need to keep the existence of these (rather astonishing) values in mind so that they don't bias our analysis.

3 Feature relations

While the previous section looked primarily at the distributions of the individual features, here we will examine in more detail how those features are related to each other and to our target *trip_duration*.

3.1 Pickup date/time vs *trip_duration*

How does the variation in trip numbers throughout the day and the week affect the average trip duration? Do quieter days and hours lead to faster trips? Here we include the *vendor_id* as an additional feature. Furthermore, for the hours of the day we add a smoothing layer to indicate the extent of the variation and its uncertainties:

Hide

```
p1 <- train %>%
  mutate(wday = wday(pickup_datetime, label = TRUE)) %>%
  group_by(wday, vendor_id) %>%
  summarise(median_duration = median(trip_duration)/60) %>%
  ggplot(aes(wday, median_duration, color = vendor_id)) +
  geom_point(size = 4) +
  labs(x = "Day of the week", y = "Median trip duration [min]")

p2 <- train %>%
  mutate(hpick = hour(pickup_datetime)) %>%
  group_by(hpick, vendor_id) %>%
  summarise(median_duration = median(trip_duration)/60) %>%
  ggplot(aes(hpick, median_duration, color = vendor_id)) +
  geom_smooth(method = "loess", span = 1/2) +
  geom_point(size = 4) +
  labs(x = "Hour of the day", y = "Median trip duration [min]") +
  theme(legend.position = "none")

layout <- matrix(c(1,2),2,1,byrow=FALSE)
multiplot(p1, p2, layout=layout)
```

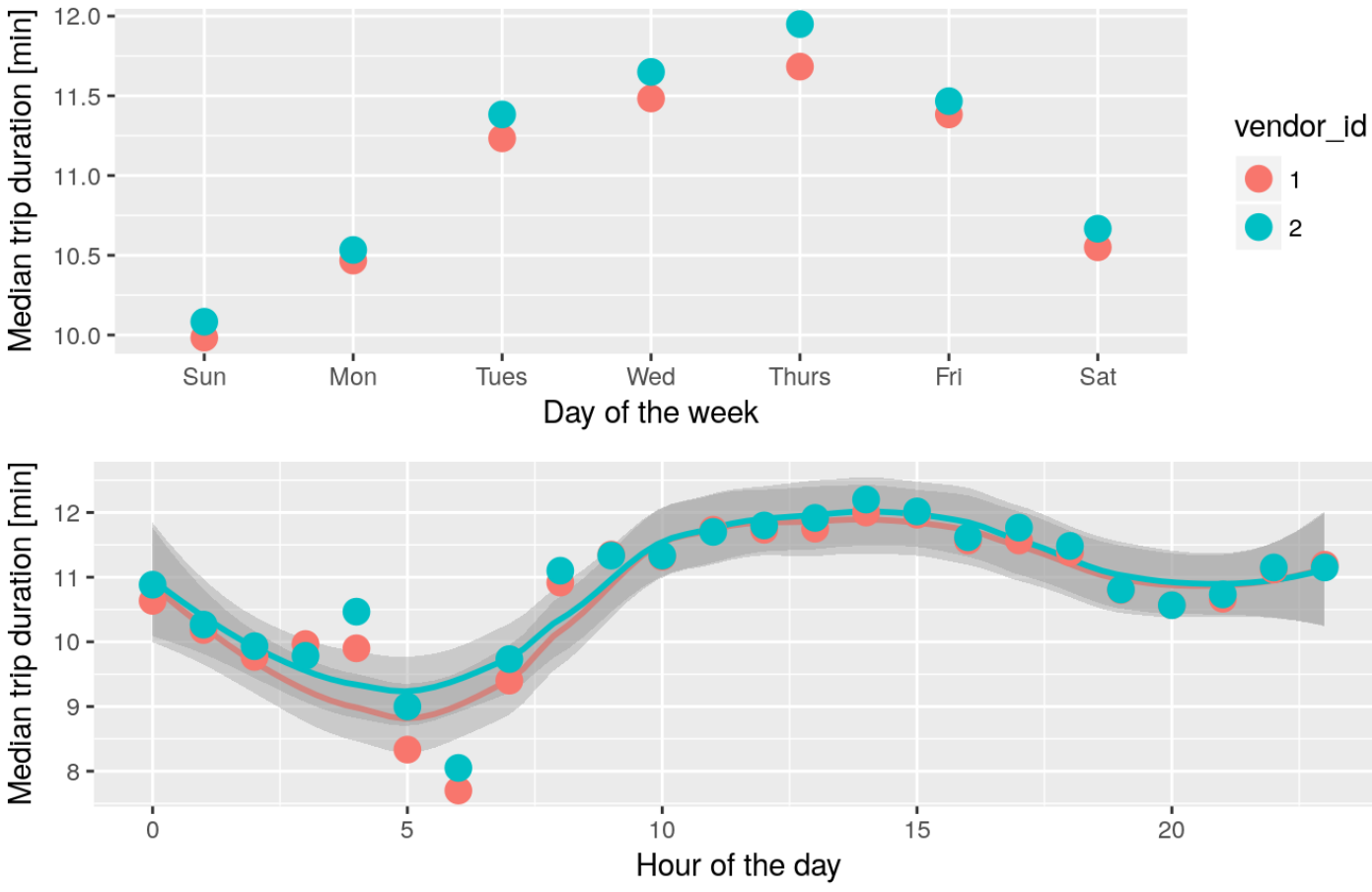

Fig. 7

Hide

```
p1 <- 1; p2 <- 1
```

We find:

- There is indeed a similar pattern as for the business of the day of the week. Vendor 2, the one with the more frequent trips, also has consistently higher trip durations than vendor 1. **It will be worth adding the *vendor_id* feature to a model to test its predictive importance.**
- Over the course of a typical day we find a peak in the early afternoon and dips around 5-6am and 8pm. **The weekday and hour of a trip appear to be important features for predicting its duration and should be included in a successful model.**

3.2 Passenger count and Vendor vs *trip_duration*

The next question we are asking is whether different numbers of passengers and/or the different vendors are correlated with the duration of the trip. We choose to examine this issue using a series of boxplots for the *passenger_counts* together with a *facet wrap* which contrasts the two *vendor_ids*:

Hide

```
train %>%
  ggplot(aes(passenger_count, trip_duration, color = passenger_count)) +
  geom_boxplot() +
  scale_y_log10() +
  theme(legend.position = "none") +
  facet_wrap(~ vendor_id) +
  labs(y = "Trip duration [s]", x = "Number of passengers")
```

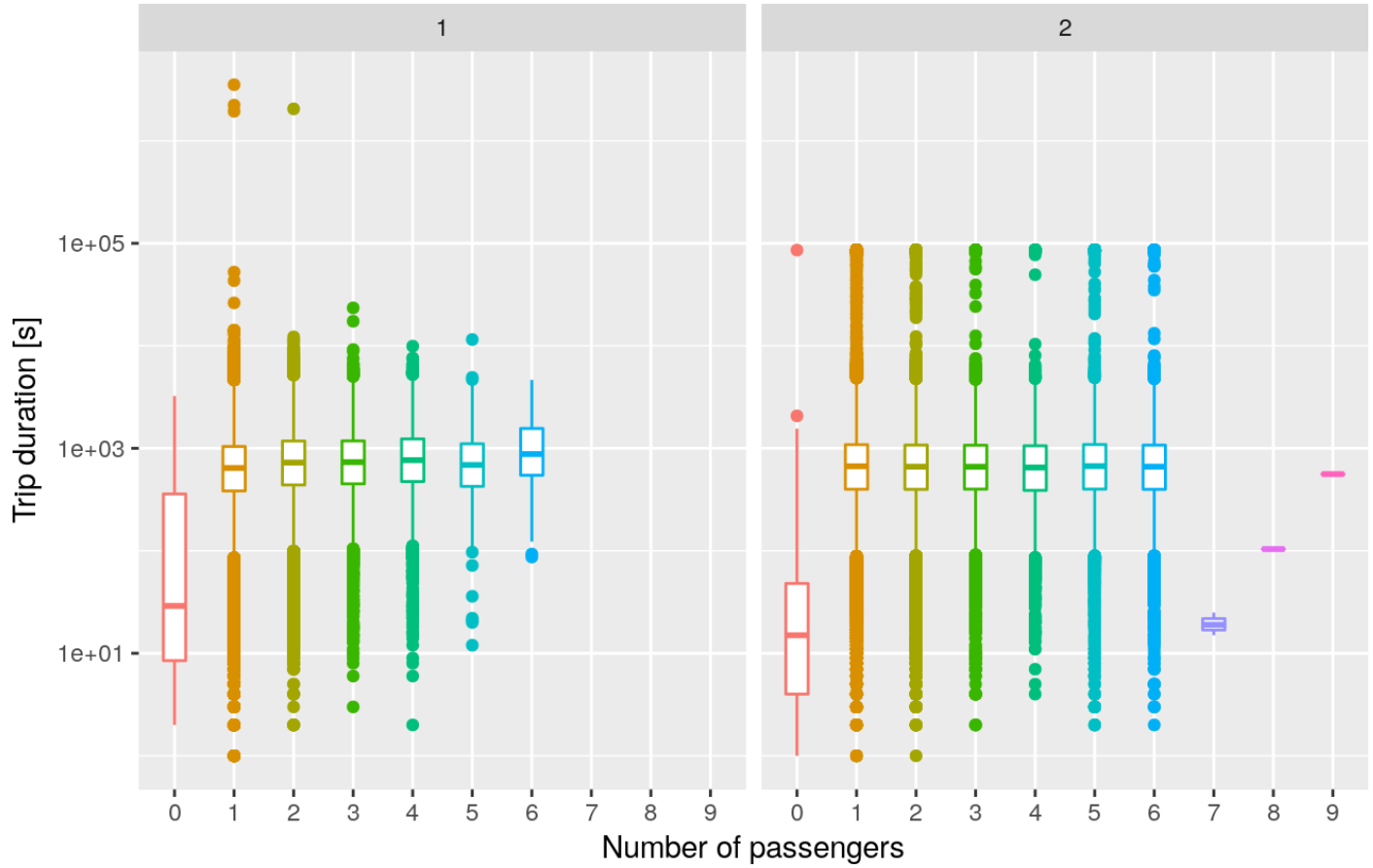

Fig. 8

We find:

- Both vendors have short trips without any passengers.
- Vendor 1 has all of the trips beyond 24 hours, whereas vendor 2 has all of the (five) trips with more than six passengers and many more trips that approach the 24-hour limit.
- Between 1 and 6 passengers the median trip durations are remarkably similar, in particular for vendor 2. There might be differences for vendor 1, but they are small (hidden plot; note the logarithmic y-axis).

Hide

```
train %>%
  ggplot(aes(trip_duration, fill = vendor_id)) +
  geom_density(position = "stack") +
  scale_x_log10()
```

Comparing the densities of the *trip_duration* distribution for the two vendors we find that the medians are very similar, whereas the means are likely skewed by vendor 2 containing most of the long-duration outliers:

Hide

```
train %>%
  group_by(vendor_id) %>%
  summarise(mean_duration = mean(trip_duration),
            median_duration = median(trip_duration))
```

```
## # A tibble: 2 x 3
##   vendor_id mean_duration median_duration
##      <fctr>         <dbl>           <dbl>
## 1         1      845.4382             658
## 2         2     1058.6432             666
```

3.3 Store and Forward vs *trip_duration*

The temporary storing of the trip data only occured for Vendor 1:

Hide

```
train %>%
  group_by(vendor_id, store_and_fwd_flag) %>%
  count()
```

```
## # A tibble: 3 x 3
## # Groups:   vendor_id, store_and_fwd_flag [3]
##   vendor_id store_and_fwd_flag      n
##      <fctr>              <chr>  <int>
## 1         1                  N 670297
## 2         1                  Y   8045
## 3         2                  N 780302
```

Hide

```
train %>%
  filter(vendor_id == 1) %>%
  ggplot(aes(passenger_count, trip_duration, color = passenger_count)) +
  geom_boxplot() +
  scale_y_log10() +
  facet_wrap(~ store_and_fwd_flag) +
  theme(legend.position = "none") +
  labs(y = "Trip duration [s]", x = "Number of passengers") +
  ggtitle("Store_and_fwd_flag impact")
```

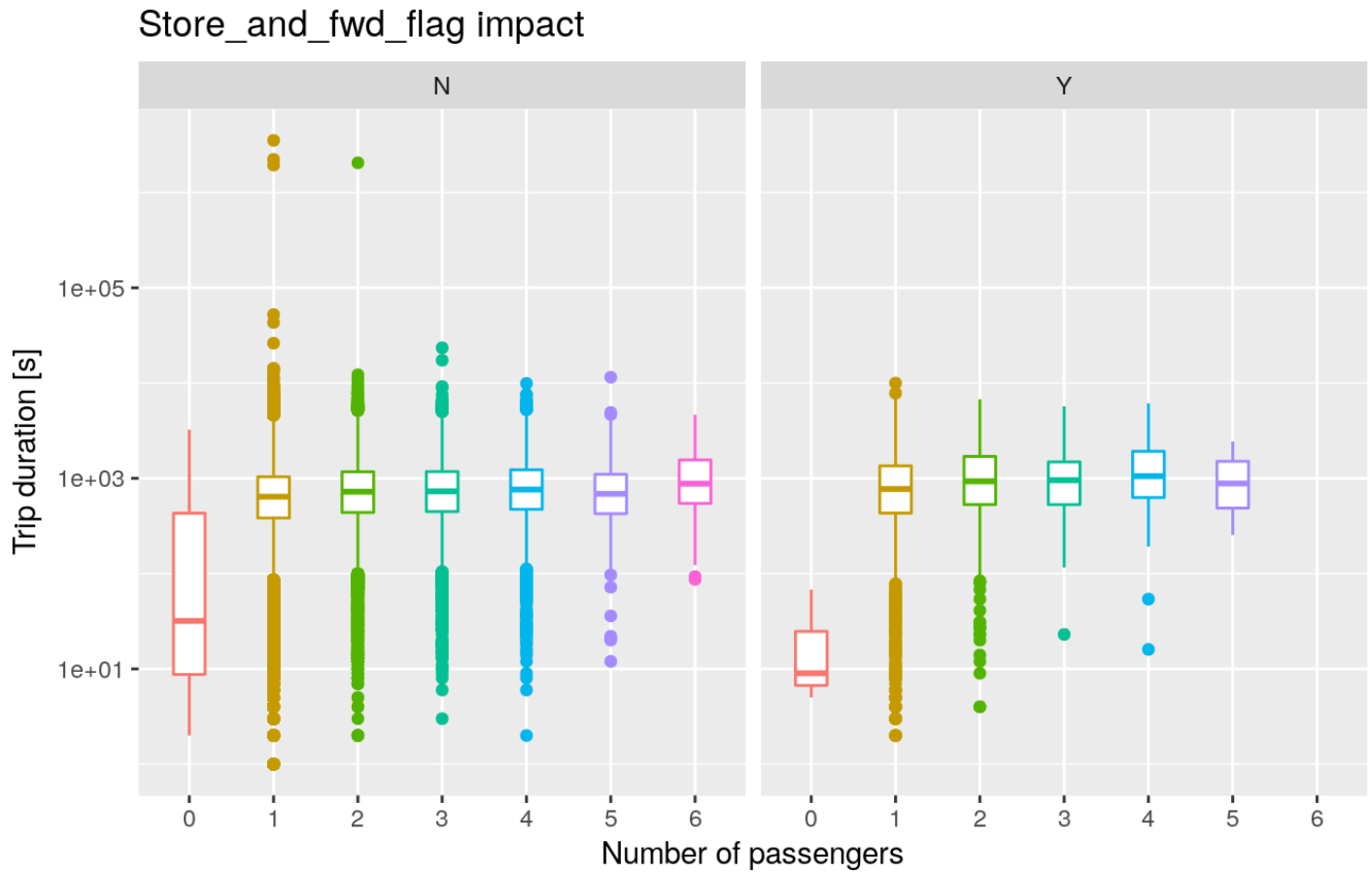

Fig. 9

We find that there is no overwhelming differences between the stored and non-stored trips. The stored ones might be slightly longer, though, and don't include any of the suspiciously long trips.

4 Feature engineering

In this section we build new features from the existing ones, trying to find better predictors for our target variable. I prefer to define all these new features in a single code block below and then study them in the following subsections. This does not correspond to a linear analysis but it makes the notebook more readable and ensures that we don't miss any stray feature definitions.

The new temporal features (date, month, wday, hour) are derived from the *pickup_datetime*. We got the JFK and La Guardia airport coordinates from Wikipedia. The *blizzard* feature is based on the external weather data.

Hide

```
jfk_coord <- tibble(lon = -73.778889, lat = 40.639722)
la_guardia_coord <- tibble(lon = -73.872611, lat = 40.77725)

pick_coord <- train %>%
  select(pickup_longitude, pickup_latitude)
drop_coord <- train %>%
  select(dropoff_longitude, dropoff_latitude)
train$dist <- distCosine(pick_coord, drop_coord)
train$bearing = bearing(pick_coord, drop_coord)

train$jfk_dist_pick <- distCosine(pick_coord, jfk_coord)
train$jfk_dist_drop <- distCosine(drop_coord, jfk_coord)
train$lg_dist_pick <- distCosine(pick_coord, la_guardia_coord)
train$lg_dist_drop <- distCosine(drop_coord, la_guardia_coord)

train <- train %>%
  mutate(speed = dist/trip_duration*3.6,
         date = date(pickup_datetime),
         month = month(pickup_datetime, label = TRUE),
         wday = wday(pickup_datetime, label = TRUE),
         wday = fct_relevel(wday, c("Mon", "Tues", "Wed", "Thurs", "Fri", "Sa
t", "Sun")),
         hour = hour(pickup_datetime),
         work = (hour %in% seq(8,18)) & (wday %in% c("Mon","Tues","Wed","Thur
s","Fri")),
         jfk_trip = (jfk_dist_pick < 2e3) | (jfk_dist_drop < 2e3),
         lg_trip = (lg_dist_pick < 2e3) | (lg_dist_drop < 2e3),
         blizzard = !( (date < ymd("2016-01-22") | (date > ymd("2016-01-29")))
 )
         )
```

4.1 Direct distance of the trip

From the coordinates of the pickup and dropoff points we can calculate the direct *distance* (as the crow flies) between the two points, and compare it to our *trip_durations*. Since taxis aren't crows (in most practical scenarios), these values correspond to the minimum possible travel distance.

To compute these distances we are using the *distCosine* function of the geosphere (https://cran.r-project.org/web/packages/geosphere/index.html) package for spherical trigonometry. This method gives us the shortest distance between two points on a spherical earth. For the purpose of this localised analysis we choose to ignore ellipsoidal distortion of the earth's shape. Here are the raw values of distance vs duration (based on a down-sized sample to speed up the kernel).

Hide

```
set.seed(4321)
train %>%
  sample_n(5e4) %>%
  ggplot(aes(dist, trip_duration)) +
  geom_point() +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Direct distance [m]", y = "Trip duration [s]")
```

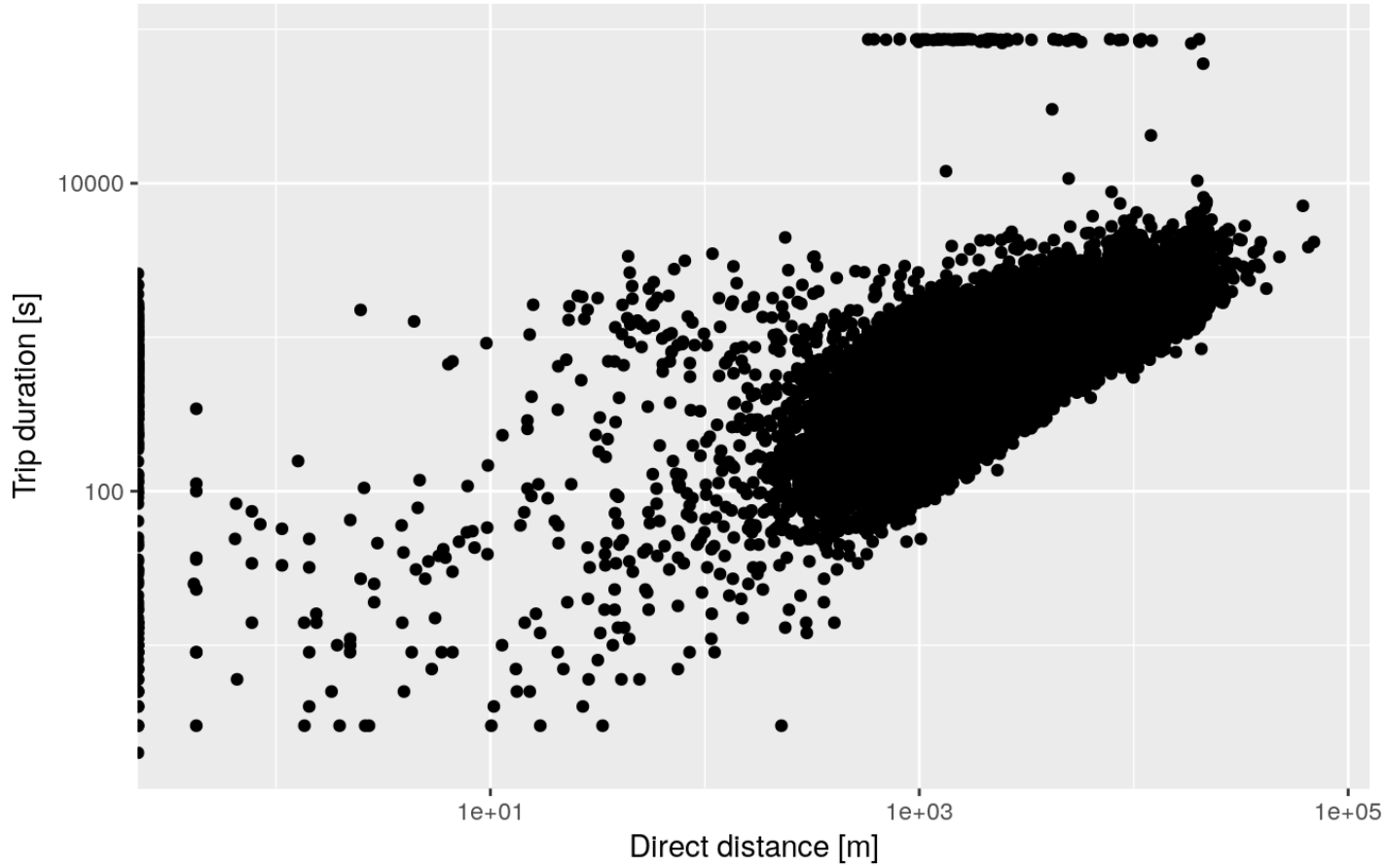

Fig. 10

We find:

- The distance generally increases with increasing *trip_duration*

- Here, the 24-hour trips look even more suspicious and are even more likely to be artefacts in the data.
- In addition, there are number of trips with very short distances, down to 1 metre, but with a large range of apparent *trip_durations*

Let's filter the data a little bit to remove the extreme (and the extremely suspicious) data points, and bin the data into a 2-d histogram (hidden plot). This plot shows that in log-log space the *trip_duration* is increasing slower than linear for larger *dist*ance values.

Hide

```
train %>%
  filter(trip_duration < 3600 & trip_duration > 120) %>%
  filter(dist > 100 & dist < 100e3) %>%
  ggplot(aes(dist, trip_duration)) +
  geom_bin2d(bins = c(500,500)) +
  scale_x_log10() +
  scale_y_log10() +
  labs(x = "Direct distance [m]", y = "Trip duration [s]")
```

4.2 Travel speed

Distance over time is of course velocity, and by computing the average apparent velocity of our taxis we will have another diagnostic to remove bogus values. Of course, we won't be able to use *speed* as a predictor for our model, since it requires knowing the travel time, but it can still be helpful in cleaning up our training data and finding other features with predictive power. This is the *speed* distribution:

Hide

```
train %>%
  filter(speed > 2 & speed < 1e2) %>%
  ggplot(aes(speed)) +
  geom_histogram(fill = "red", bins = 50) +
  labs(x = "Average speed [km/h] (direct distance)")
```

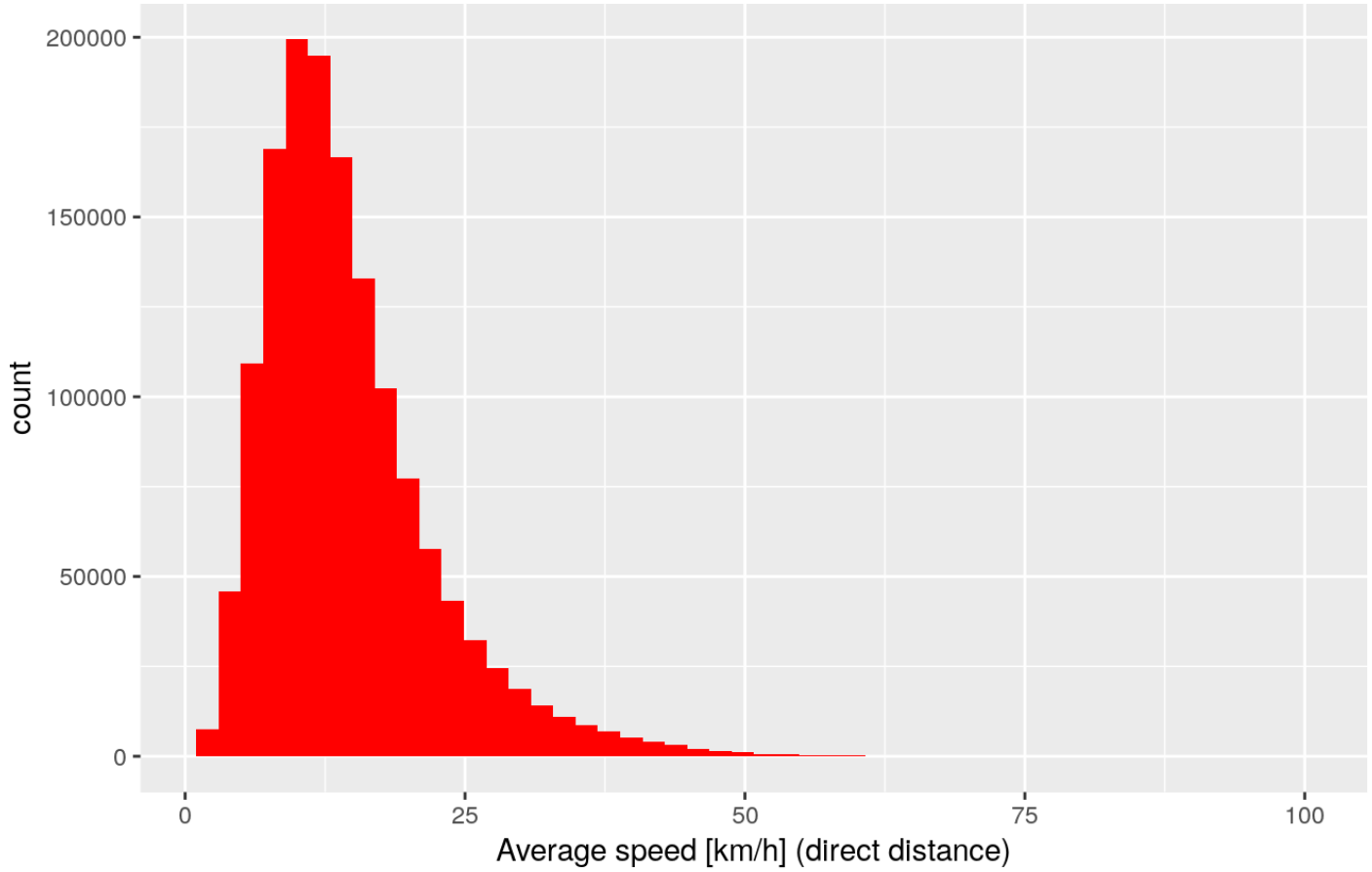

Fig. 11

Well, after removing the most extreme values this looks way better than I would have expected. An average speed of around 15 km/h sounds probably reasonable for NYC. Everything above 50 km/h certainly requires magical cars (or highway travel). Also keep in mind that this refers to the direct distance and that the real velocity would have been always higher.

In a similar way as the average duration per day and hour we can also investigate the average speed for these time bins:

Hide

```
p1 <- train %>%
  group_by(wday, vendor_id) %>%
  summarise(median_speed = median(speed)) %>%
  ggplot(aes(wday, median_speed, color = vendor_id)) +
  geom_point(size = 4) +
  labs(x = "Day of the week", y = "Median speed [km/h]")

p2 <- train %>%
  group_by(hour, vendor_id) %>%
  summarise(median_speed = median(speed)) %>%
  ggplot(aes(hour, median_speed, color = vendor_id)) +
  geom_smooth(method = "loess", span = 1/2) +
  geom_point(size = 4) +
  labs(x = "Hour of the day", y = "Median speed [km/h]") +
  theme(legend.position = "none")

p3 <- train %>%
  group_by(wday, hour) %>%
  summarise(median_speed = median(speed)) %>%
  ggplot(aes(hour, wday, fill = median_speed)) +
  geom_tile() +
  labs(x = "Hour of the day", y = "Day of the week") +
  scale_fill_distiller(palette = "Spectral")

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)
```

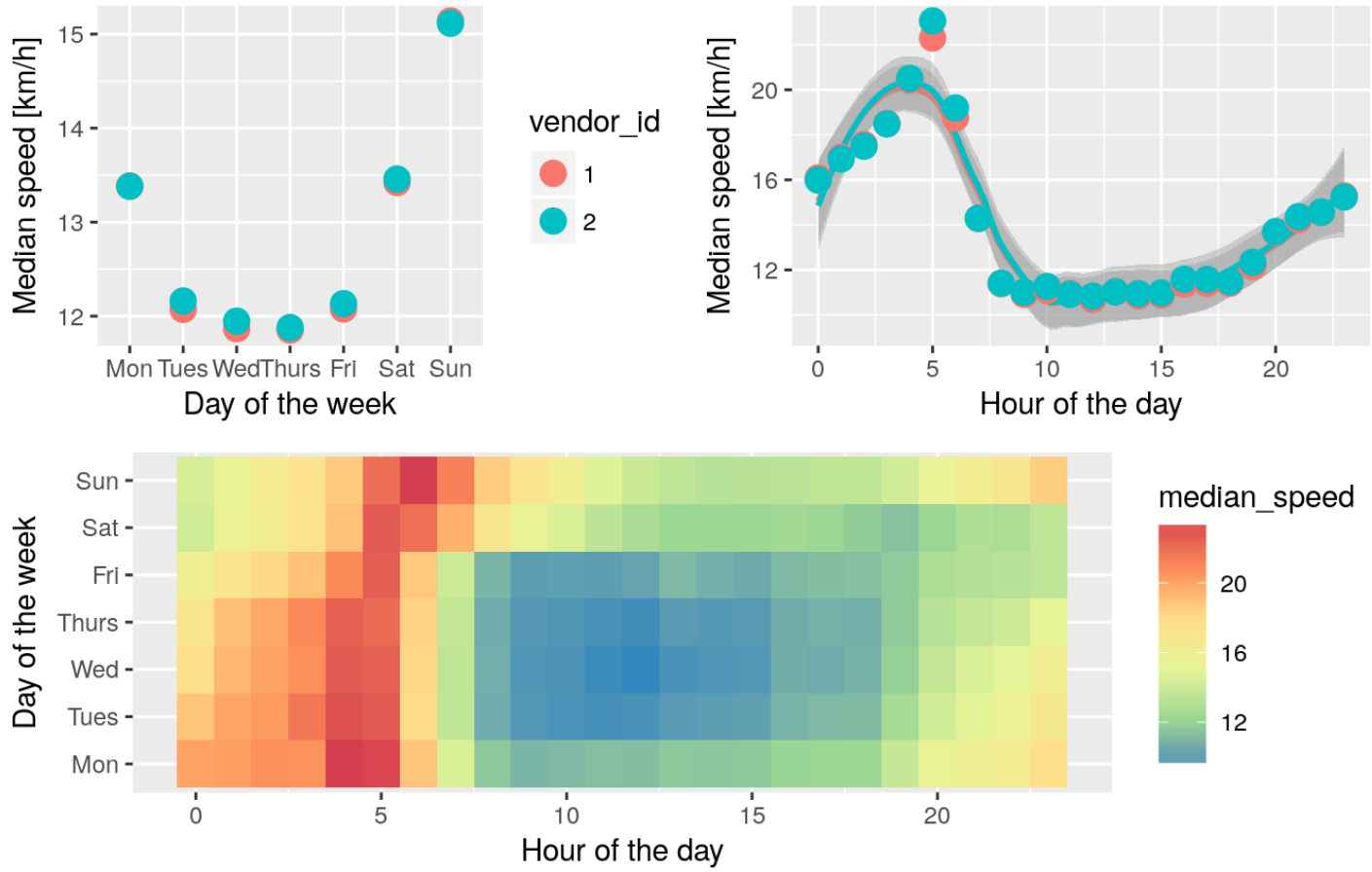

Fig. 12

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1
```

We find:

- Our taxis appear to be travelling faster on the weekend and on a Monday than during the rest of the week.
- The early morning hours allow for a speedier trip, with everything from 8am to 6pm being similarly slow.
- There are almost no differences between the two vendors.
- The heatmap in the lower panel visualises how these trends combine to create a "low-speed-zone" in the middle of the day and week. **Based on this, we create a new feature *work*, which we define as 8am-6pm on Mon-Fri.**

4.3 Bearing direction

If the direct *distance* is the magnitude of the trip vector then the *bearing* is its (initial) direction. Easily estimated through the *geosphere* package, it tells us whether our trip started out for instance in the direction of North-West or South-East. Here we visualise the *bearing*

distribution and its relation to *trip_duration*, direct *distance*, and *speed*:

Hide

```
p1 <- train %>%
  filter(dist < 1e5) %>%
  ggplot(aes(bearing)) +
  geom_histogram(fill = "red", bins = 75) +
  scale_x_continuous(breaks = seq(-180, 180, by = 45)) +
  labs(x = "Bearing")

p2 <- train %>%
  filter(dist < 1e5) %>%
  ggplot(aes(bearing, dist)) +
  geom_bin2d(bins = c(100,100)) +
  labs(x = "Bearing", y = "Direct distance") +
  scale_y_log10() +
  theme(legend.position = "none") +
  coord_polar() +
  scale_x_continuous(breaks = seq(-180, 180, by = 45))

p3 <- train %>%
  filter(trip_duration < 3600*22) %>%
  filter(dist < 1e5) %>%
  ggplot(aes(bearing, trip_duration)) +
  geom_bin2d(bins = c(100,100)) +
  scale_y_log10() +
  labs(x = "Bearing", y = "Trip duration") +
  coord_polar() +
  scale_x_continuous(breaks = seq(-180, 180, by = 45))

p4 <- train %>%
  filter(speed < 75 & dist < 1e5) %>%
  ggplot(aes(bearing, speed)) +
  geom_bin2d(bins = c(100,100)) +
  labs(x = "Bearing", y = "Speed") +
  coord_polar() +
  scale_x_continuous(breaks = seq(-180, 180, by = 45))

layout <- matrix(c(1,2,3,4),2,2,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```

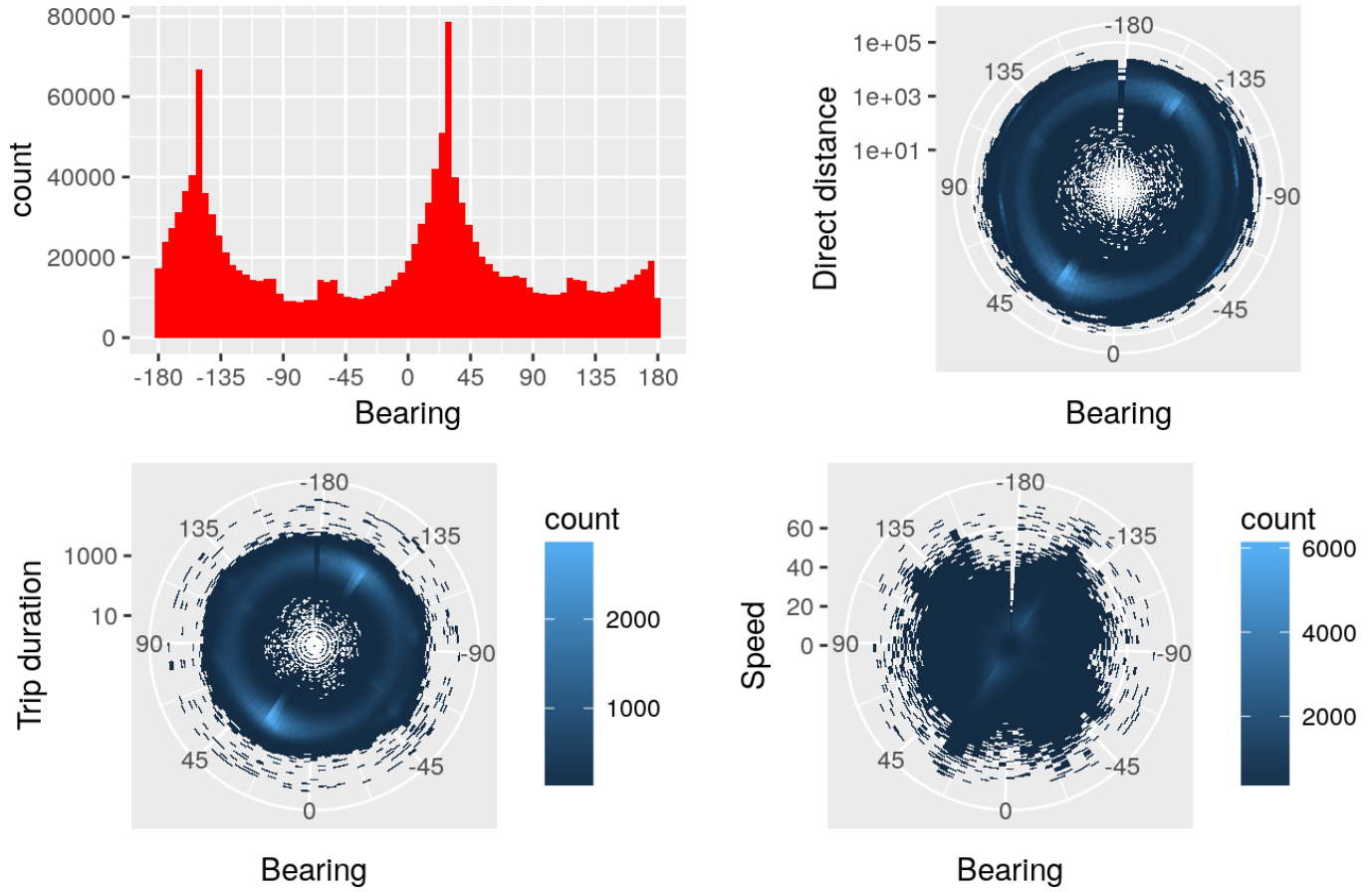

Fig. 13

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1
```

This is one of the few cases where polar coordinates might be actually helpful. (Ignore the small discontinuities caused by the binning.)

We find:

- The *bearing* direction has two prominent peaks around 30 and -150 degrees. Intuitively, those might relate to the orientation of Manhattan. Three or four smaller, less sharp peaks are visible in between those larger peaks.
- The 2-d histograms of *distance* and *trip_duration* don't reveal much structure. There are indications for shorter durations and distances at the two peaks, but those might simply be caused by the larger number of observations in these peaks. Faint clusters towards higher *trip_durations* might be visible near -60 and 125 degrees *bearing*.
- The bright "rings" we see in the *distance* and *trip_duration* reflect the familiar distribution peaks on the logarithmic scale.

- *Speed*, on the other hand, shows an interestingly clustered structure along the *bearing* directions. Zones of higher speed can be identified at the aforementioned *bearing* peaks as well roughly perpendicular to them (most prominently around 125 degrees). It is possible that we are seeing the Manhattan street grid in this data. Note, that here the radial scale is not logarithmic.

4.4 Airport distance

In our maps (above) and trip paths (below) we noticed that a number of trips began or ended at either of the two NYC airports: JFK and La Guardia. Since airports are usually not in the city centre it is reasonable to assume that the pickup/dropoff distance from the airport could be a useful predictor for longer *trip_durations*. Above, we defined the coordinates of the two airports and compute the corresponding distances. (Their distributions are included as a hidden plot.)

Hide

```
p1 <- train %>%
  ggplot(aes(jfk_dist_pick)) +
  geom_histogram(bins = 30, fill = "red") +
  scale_x_log10() +
  scale_y_sqrt() +
  geom_vline(xintercept = 2e3) +
  labs(x = "JFK pickup distance")

p2 <- train %>%
  ggplot(aes(jfk_dist_drop)) +
  geom_histogram(bins = 30, fill = "blue") +
  scale_x_log10() +
  scale_y_sqrt() +
  geom_vline(xintercept = 2e3) +
  labs(x = "JFK dropoff distance")

p3 <- train %>%
  ggplot(aes(lg_dist_pick)) +
  geom_histogram(bins = 30, fill = "red") +
  scale_x_log10() +
  scale_y_sqrt() +
  geom_vline(xintercept = 2e3) +
  labs(x = "La Guardia pickup distance")

p4 <- train %>%
  ggplot(aes(lg_dist_drop)) +
  geom_histogram(bins = 30, fill = "blue") +
  scale_x_log10() +
  scale_y_sqrt() +
  geom_vline(xintercept = 2e3) +
  labs(x = "La Guardia dropoff distance")

layout <- matrix(c(1,2,3,4),2,2,byrow=FALSE)
multiplot(p1, p2, p3, p4, layout=layout)
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1; p5 <- 1
```

Based on these numbers, we can define a JFK/La Guardia trip as having a pickup or dropoff distance of less than 2 km from the corresponding airport.

What are the *trip_durations* of these journeys?

Hide

```
p1 <- train %>%
  filter(trip_duration < 23*3600) %>%
  ggplot(aes(jfk_trip, trip_duration, color = jfk_trip)) +
  geom_boxplot() +
  scale_y_log10() +
  theme(legend.position = "none") +
  labs(x = "JFK trip")

p2 <- train %>%
  filter(trip_duration < 23*3600) %>%
  ggplot(aes(lg_trip, trip_duration, color = lg_trip)) +
  geom_boxplot() +
  scale_y_log10() +
  theme(legend.position = "none") +
  labs(x = "La Guardia trip")

layout <- matrix(c(1,2),1,2,byrow=FALSE)
multiplot(p1, p2, layout=layout)
```

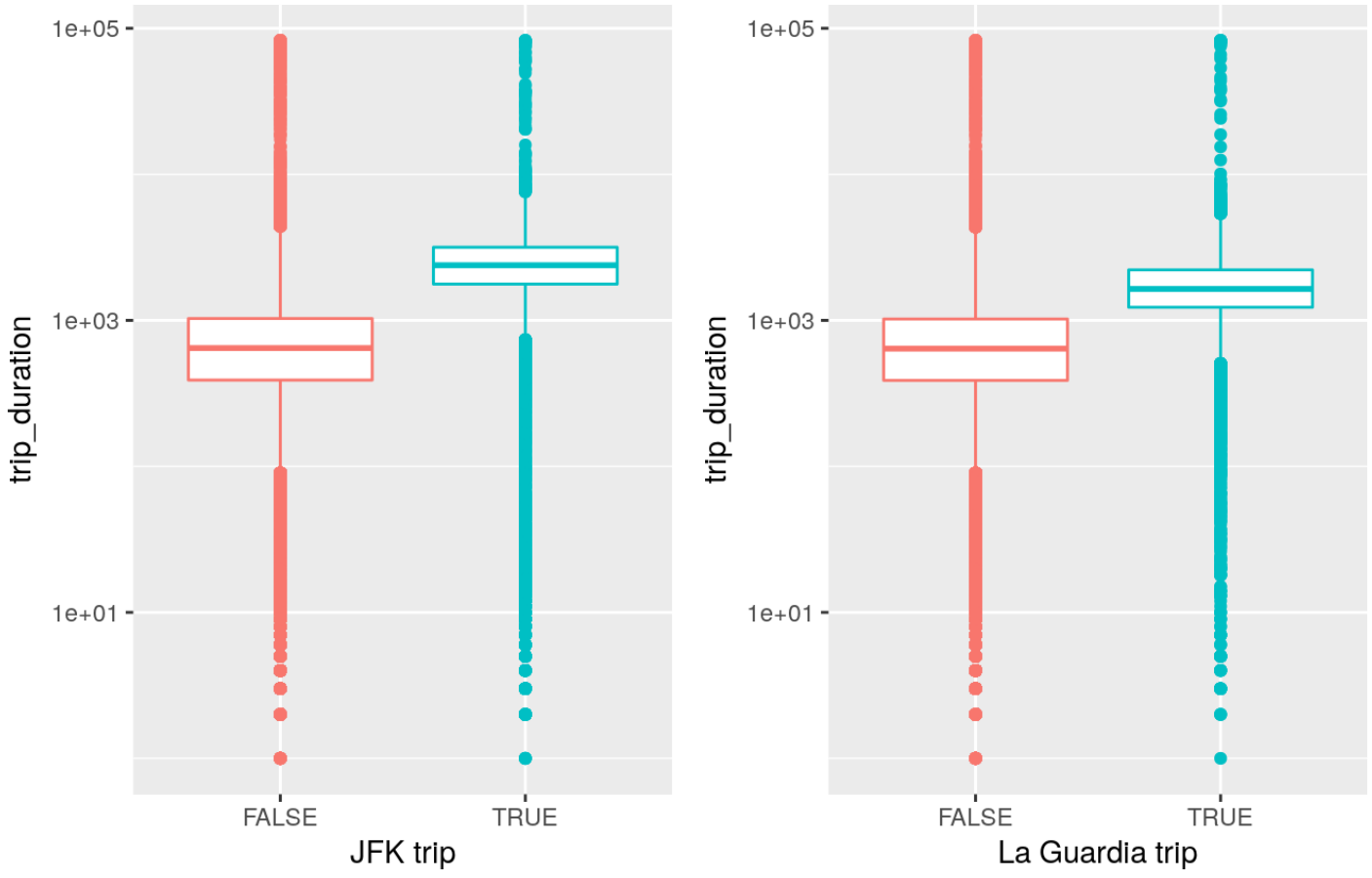

Fig. 14

Hide

```
p1 <- 1; p2 <- 1
```

We find that our hypothesis was correct and that trips to the airport, in particular the more distant JFK, have significantly longer average *trip_durations*. **These two features should definitely be part of our model.**

5 Data cleaning

Before we turn to the modelling it is time to clean up our training data. We have waited to do this until now to have a more complete overview of the problematic observations. The aim here is to remove trips that have improbable features, such as extreme trip durations or very low average speed.

While there might also be a number of bogus trip durations in the test data we shouldn't be able to predict them in any case (unless there were some real correlations). By removing these training data values we will make our model more robust and more likely to generalise to unseen data, which is always our primary goal in machine learning.

5.1 Extreme trip durations

Let's visualise the distances of the trips that took a day or longer. Unless someone took a taxi from NYC to LA it is unlikely that those values are accurate. Here we make use of the *maps* package to draw an outline of Manhattan, where most of the trips begin or end. We then overlay the pickup coordinates in red, and the dropoff coordinates in blue.

To further add the trip connections (direct distance) we use another handy tool from the *geosphere* package: *gcIntermediate* allows us to interpolate the path between two sets of coordinates. I've seen this tool first in action in this truly outstanding kernel (https://www.kaggle.com/jonathanbouchet/u-s-commercial-flights-tracker-map/) by Jonathan Bouchet (https://www.kaggle.com/jonathanbouchet). (If you haven't seen his kernel, then I strongly recommend you check it out; right after reading this one, of course ;-)).

5.1.1 Longer than a day

We start with the few trips that pretend to have taken several days to complete:

Hide

```
day_plus_trips <- train %>%
  filter(trip_duration > 24*3600)

day_plus_trips %>% select(pickup_datetime, dropoff_datetime, speed)
```

```
## # A tibble: 4 x 3
##       pickup_datetime    dropoff_datetime       speed
##                <dttm>              <dttm>       <dbl>
## 1 2016-01-05 00:19:42 2016-01-27 11:08:38 0.037436252
## 2 2016-02-13 22:38:00 2016-03-08 15:57:38 0.010519765
## 3 2016-01-05 06:14:15 2016-01-31 01:01:07 0.002645458
## 4 2016-02-13 22:46:52 2016-03-25 18:18:14 0.020339450
```

Hide

```
ny_map <- as.tibble(map_data("state", region = "new york:manhattan"))

tpick <- day_plus_trips %>%
  select(lon = pickup_longitude, lat = pickup_latitude)
tdrop <- day_plus_trips %>%
  select(lon = dropoff_longitude, lat = dropoff_latitude)

p1 <- ggplot() +
  geom_polygon(data=ny_map, aes(x=long, y=lat), fill = "grey60") +
  geom_point(data=tpick,aes(x=lon,y=lat),size=1,color='red',alpha=1) +
  geom_point(data=tdrop,aes(x=lon,y=lat),size=1,color='blue',alpha=1)

for (i in seq(1,nrow(tpick))){
  inter <- as.tibble(gcIntermediate(tpick[i,],  tdrop[i,], n=30, addStartEnd=T
RUE))
  p1 <- p1 +  geom_line(data=inter,aes(x=lon,y=lat),color='blue',alpha=.75)
}

p1 + ggtitle("Longer than a day trips in relation to Manhattan")
```

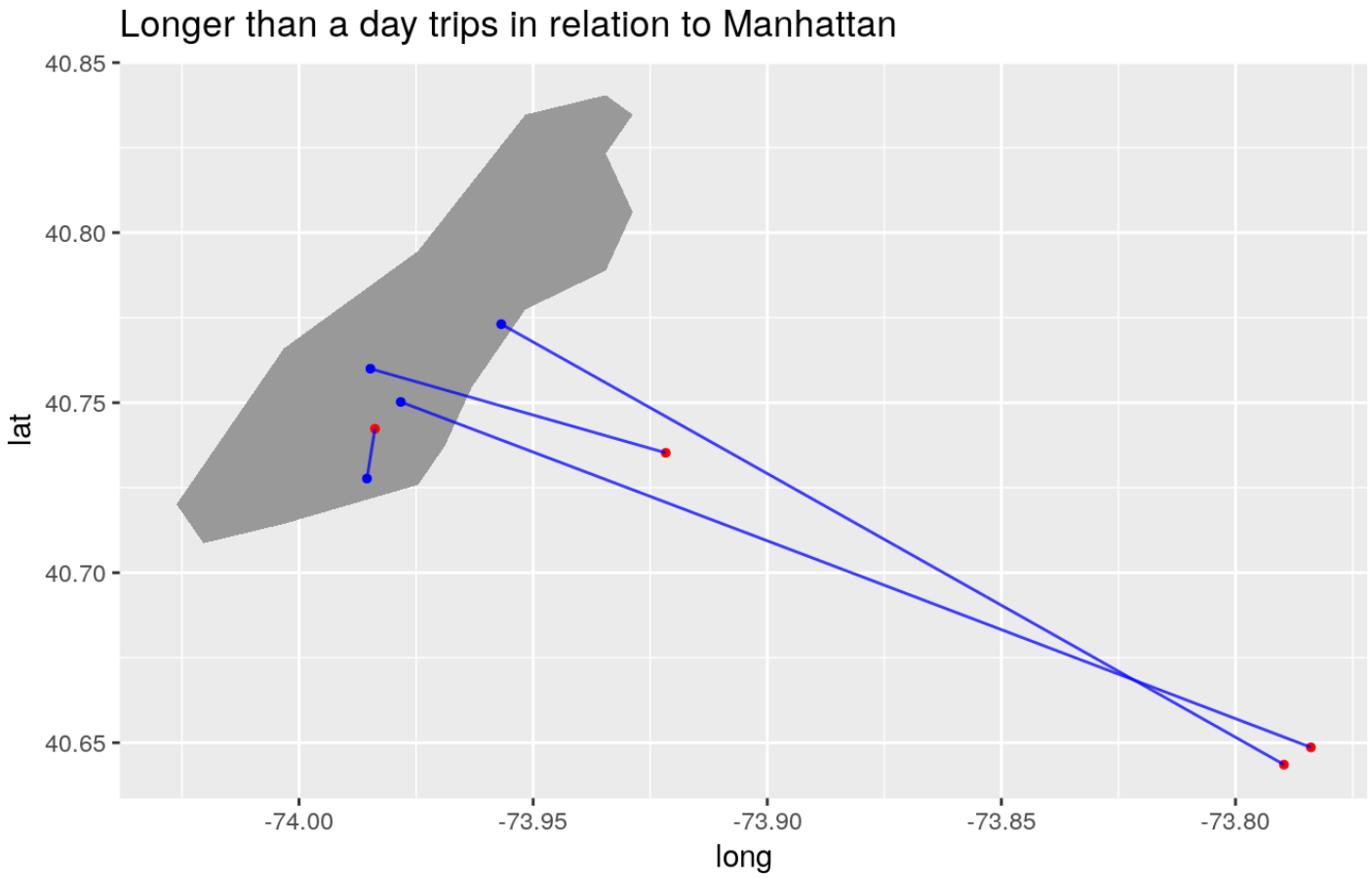

Fig. 15

Hide

```
p1 <- 1
```

We find nothing out of the ordinary here. While a trip to JFK can seem like an eternity if your flight is boarding soon, it is unlikely to take this long in real time. The average taxi speeds don't look very likely either.

Decision: These values should be removed from the training data set for continued exploration and modelling.

5.1.2 Close to 24 hours

Call me crazy, but I don't think it is inconceivable that someone takes a taxi for a trip that lasts almost a day (with breaks, of course). In very rare occasions this might happen; provided, of course, that the distance travelled was sufficiently long.

Here we define day-long trips as taking between 22 and 24 hours, which covers a small peak in our raw *trip_duration* distribution. Those are the top 5 direct distances (in m) among the day-long trips:

Hide

```
day_trips <- train %>%
  filter(trip_duration < 24*3600 & trip_duration > 22*3600)

day_trips %>%
  arrange(desc(dist)) %>%
  select(dist, pickup_datetime, dropoff_datetime, speed) %>%
  head(5)
```

```
## # A tibble: 5 x 4
##       dist     pickup_datetime    dropoff_datetime     speed
##      <dbl>              <dttm>              <dttm>     <dbl>
## 1 60666.07 2016-06-04 13:54:29 2016-06-05 13:40:30 2.5525397
## 2 42401.12 2016-01-28 21:43:02 2016-01-29 21:33:30 1.7784875
## 3 28115.94 2016-03-14 22:46:05 2016-03-15 22:24:27 1.1893653
## 4 22807.88 2016-06-18 17:41:47 2016-06-19 16:30:41 0.9996878
## 5 22758.58 2016-06-01 19:52:42 2016-06-02 19:35:09 0.9599740
```

The top one is about 60 km (about 37 miles), which is not particularly far. The average speed wouldn't suggest a generous tip, either. What do these trips look like on the map?

Hide

```
ny_map <- as.tibble(map_data("state", region = "new york:manhattan"))

set.seed(2017)
day_trips <- day_trips %>%
  sample_n(200)

tpick <- day_trips %>%
  select(lon = pickup_longitude, lat = pickup_latitude)
tdrop <- day_trips %>%
  select(lon = dropoff_longitude, lat = dropoff_latitude)

p1 <- ggplot() +
  geom_polygon(data=ny_map, aes(x=long, y=lat), fill = "grey60") +
  geom_point(data=tpick,aes(x=lon,y=lat),size=1,color='red',alpha=1) +
  geom_point(data=tdrop,aes(x=lon,y=lat),size=1,color='blue',alpha=1)

for (i in seq(1,nrow(tpick))){
  inter <- as.tibble(gcIntermediate(tpick[i,],  tdrop[i,], n=30, addStartEnd=T
RUE))
  p1 <- p1 +  geom_line(data=inter,aes(x=lon,y=lat),color='blue',alpha=.25)
}

p1 + ggtitle("Day-long trips in relation to Manhattan")
```

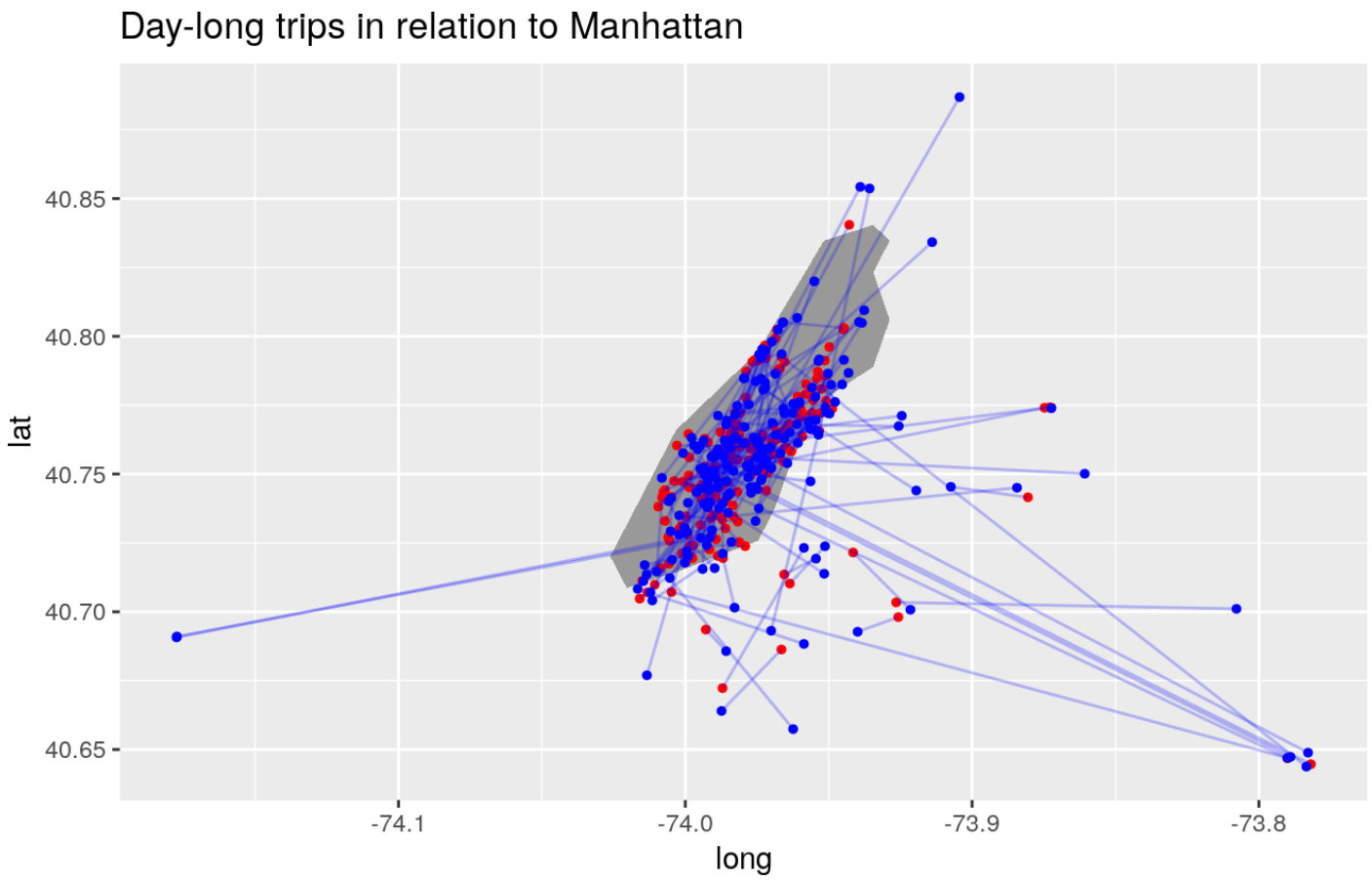

Fig. 16

Hide

```
p1 <- 1
```

Here we are plotting only 200 of the about 1800 connections to keep the map reasonably readable and the script fast. Pickup points are red and dropoff points are blue.

We find:

- A few longer distances stand out, but they are exceptions. The two major group of trips are those within Manhattan and those between Manhattan and the airports.
- There is little to suggest that these extreme *trip_durations* were real.
- There is another insight here which is rather intuitive: trips to or from any of the airports (most prominently JFK) are unlikely to be very short. **Thus, the a close distance of either pickup or dropoff to the airport could be a valuable predictor for longer *trip_duration*.** This is something that we took from here to the feature engineering.

Decision: We will remove *trip_durations* longer than 22 hours from the exploration and possibly from the modelling.

5.1.3 Shorter than a few minutes

On the other side of the *trip_duration* distribution we have those rides that appear to only have lasted for a couple of minutes. While such short trips are entirely possible, let's check their durations and speeds to make sure that they are realistic.

Hide

```
min_trips <- train %>%
  filter(trip_duration < 5*60)

min_trips %>%
  arrange(dist) %>%
  select(dist, pickup_datetime, dropoff_datetime, speed) %>%
  head(5)
```

```
## # A tibble: 5 x 4
##    dist     pickup_datetime    dropoff_datetime speed
##   <dbl>              <dttm>              <dttm> <dbl>
## 1     0 2016-02-29 18:39:12 2016-02-29 18:42:59     0
## 2     0 2016-01-27 22:29:31 2016-01-27 22:29:58     0
## 3     0 2016-01-22 16:13:01 2016-01-22 16:13:20     0
## 4     0 2016-01-18 15:24:43 2016-01-18 15:28:57     0
## 5     0 2016-05-04 22:28:43 2016-05-04 22:32:51     0
```

5.1.3.1 Zero-distance trips

In doing so, we notice that there are a relatively large number of zero-distance trips:

Hide

```
zero_dist <- train %>%
  filter(near(dist,0))
nrow(zero_dist)
```

```
## [1] 5897
```

What are their nominal top durations?

Hide

```
zero_dist %>%
  arrange(desc(trip_duration)) %>%
  select(trip_duration, pickup_datetime, dropoff_datetime, vendor_id) %>%
  head(5)
```

```
## # A tibble: 5 x 4
##   trip_duration     pickup_datetime    dropoff_datetime vendor_id
##           <int>              <dttm>              <dttm>    <fctr>
## 1         86352 2016-06-05 01:09:39 2016-06-06 01:08:51         2
## 2         85333 2016-01-01 15:27:28 2016-01-02 15:09:41         2
## 3         78288 2016-05-18 13:40:45 2016-05-19 11:25:33         2
## 4          5929 2016-06-08 16:47:44 2016-06-08 18:26:33         2
## 5          4683 2016-05-25 17:36:49 2016-05-25 18:54:52         2
```

There really are a few taxis where the data wants to tell us that they have not moved at all for about a day. While carrying a passenger. We choose not to believe the data in this case.

Once we remove the extreme cases, this is what the distribution looks like:

Hide

```
zero_dist %>%
  filter(trip_duration < 6000) %>%
  ggplot(aes(trip_duration, fill = vendor_id)) +
  geom_histogram(bins = 50) +
  scale_x_log10()
```

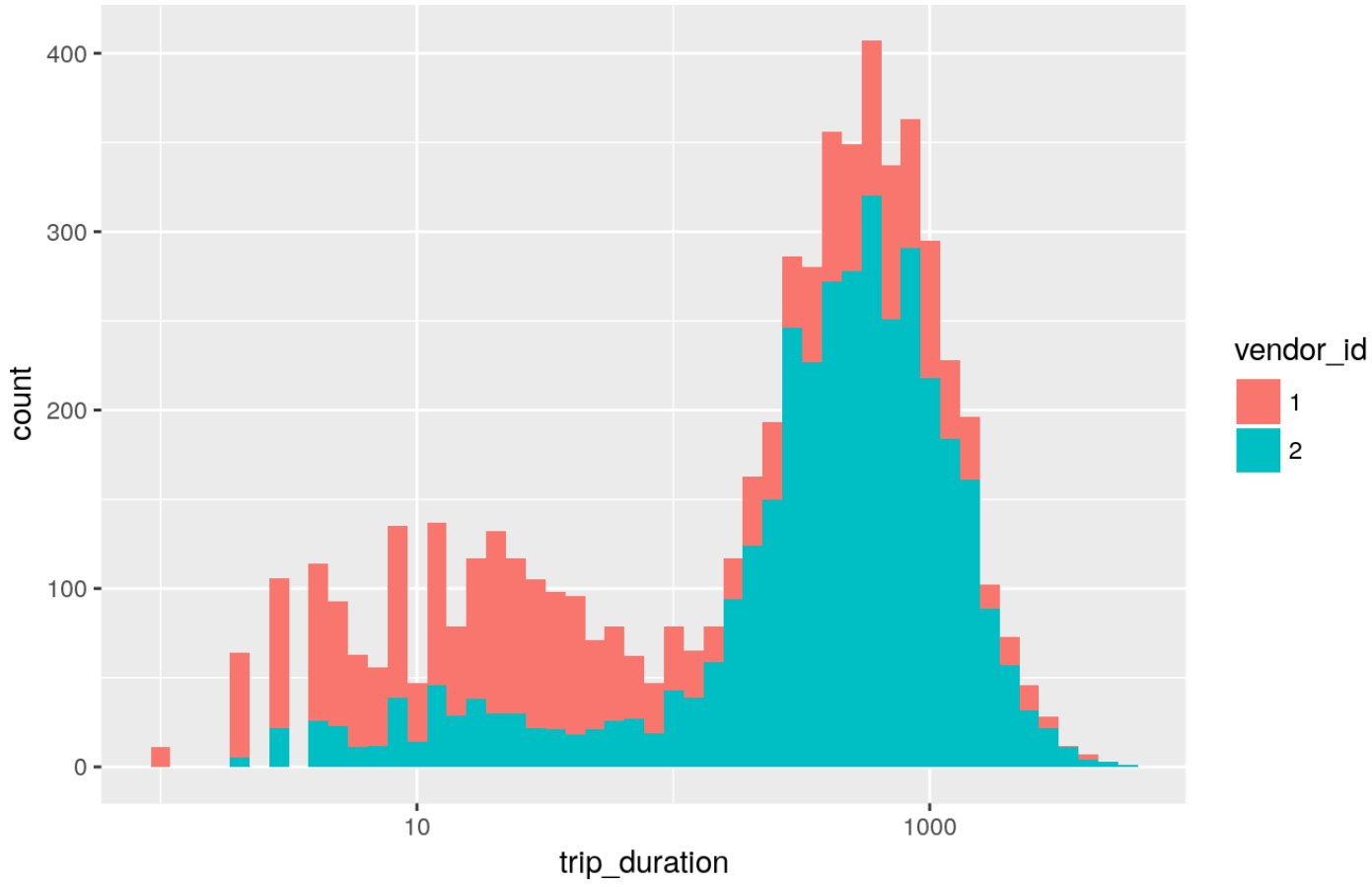

Fig. 17

We find:

- *trip_durations* of about a minute might still be somehow possible, assuming that someone got into a taxi but then changed their mind before the taxi could move. Whether that should count as a “trip” is a different question. But trip durations of about 15 minutes (900 s) without any distance covered seem hardly possible. Unless they involve traffic jams, but who get’s into a taxi that’s stuck in a traffic jam?
- It is also noteworthy that most trips in the less-than-a-minute-group were from vendor 1, whereas the 10-minute-group predominantly consists of vendor 2 taxis.

Decision: We will remove those zero-distance trips that took more than a minute for our continued analysis. Removing them from the modelling might be detrimental if there are similar trips in the test sample.

5.1.3.2 Short trips with above zero distance

After removing the zero-distance trips, those are the short rides with the highest average speed:

Hide

```
min_trips <- train %>%
  filter(trip_duration < 5*60 & dist > 0)

min_trips %>%
  arrange(desc(speed)) %>%
  select(trip_duration, dist, pickup_datetime, speed) %>%
  head(10)
```

```
## # A tibble: 10 x 4
##    trip_duration        dist     pickup_datetime     speed
##            <int>       <dbl>              <dttm>     <dbl>
##  1             7  18054.6075 2016-02-13 20:28:30 9285.2267
##  2           282 320484.3902 2016-04-02 20:33:19 4091.2901
##  3             2    784.2246 2016-06-12 06:35:13 1411.6043
##  4            51  19970.4984 2016-06-19 20:18:35 1409.6822
##  5           279 104877.3262 2016-03-20 21:07:56 1353.2558
##  6             2    703.8524 2016-06-23 13:36:48 1266.9344
##  7            20   6919.2843 2016-05-28 15:14:19 1245.4712
##  8             3    913.9078 2016-05-30 17:12:12 1096.6894
##  9             4   1030.8929 2016-01-17 03:11:56  927.8036
## 10             3    761.8285 2016-06-13 16:34:30  914.1942
```

Clearly, the top speed values are impossible. (We include a map plot of the general distribution of distances within the sample as a hidden figure.)

Hide

```
ny_map <- as.tibble(map_data("state", region = "new york:manhattan"))

set.seed(1234)
foo <- min_trips %>%
  sample_n(600)

tpick <- foo %>%
  select(lon = pickup_longitude, lat = pickup_latitude)
tdrop <- foo %>%
  select(lon = dropoff_longitude, lat = dropoff_latitude)

p1 <- ggplot() +
  geom_polygon(data=ny_map, aes(x=long, y=lat), fill = "grey60") +
  geom_point(data=tpick,aes(x=lon,y=lat),size=1,color='red',alpha=1) +
  geom_point(data=tdrop,aes(x=lon,y=lat),size=1,color='blue',alpha=1)

for (i in seq(1,nrow(tpick))){
  inter <- as.tibble(gcIntermediate(tpick[i,],  tdrop[i,], n=30, addStartEnd=T
RUE))
  p1 <- p1 +  geom_line(data=inter,aes(x=lon,y=lat),color='blue',alpha=.25)
}

p1 + ggtitle("Minute-long trips in relation to Manhattan")
p1 <- 1
```

We find (from the hidden plot):

- Most distances are in fact short, which means that combined with setting an average speed limit we should be able to remove those values that are way beyond being realistic. This should also get rid of many trips that appear to have durations of seconds only.

Decision: We impose a lower *trip_duration* limit of 10 seconds and a (very conservative) speed limit of 100 km/s (62 mph). (Remember that this refers to the direct distance.)

5.2 Intermission - The best spurious trips

Every data set has a few entries that are just flat out ridiculous. Here are the best ones from this one, with pickup or dropoff locations more than 300 km away from NYC (JFK airport)

Hide

```
long_dist <- train %>%
  filter( (jfk_dist_pick > 3e5) | (jfk_dist_drop > 3e5) )
long_dist_coord <- long_dist %>%
  select(lon = pickup_longitude, lat = pickup_latitude)

long_dist %>%
  select(id, jfk_dist_pick, jfk_dist_drop, dist, trip_duration, speed) %>%
  arrange(desc(jfk_dist_pick))
```

```
## # A tibble: 31 x 6
##           id jfk_dist_pick jfk_dist_drop         dist trip_duration
##        <chr>         <dbl>         <dbl>        <dbl>         <int>
##  1 id2854272     4128726.8    4128718.56 1.482711e+01           499
##  2 id3777240     4128721.2    4128726.17 2.182484e+01          1105
##  3 id2306955     1253573.5      21483.58 1.242299e+06           792
##  4 id1974018     1115645.8    1115645.83 0.000000e+00           369
##  5 id0267429      988747.9     988747.87 0.000000e+00           961
##  6 id0838705      695766.6     481140.64 2.154677e+05          1131
##  7 id0205460      684132.6     684132.55 0.000000e+00           329
##  8 id0978162      674251.0     941618.26 3.151168e+05           875
##  9 id3525158      665876.7     665876.68 0.000000e+00           385
## 10 id1510552      642663.3     472019.96 8.922125e+05           611
## # ... with 21 more rows, and 1 more variables: speed <dbl>
```

Many zero-distance trips with more than a minute duration, which we would remove anyway. But just out of curiousity, where did they happen? (We will again use the amazing *leaflet* package, this time with individual markers that give us *id* and direct *distance* information for mouse-over and click actions.)

Hide

```
leaflet(long_dist_coord) %>%
  addTiles() %>%
  setView(-92.00, 41.0, zoom = 4) %>%
  addProviderTiles("CartoDB.Positron") %>%
  addMarkers(popup = ~as.character(long_dist$dist), label = ~as.character(long
_dist$id))
```

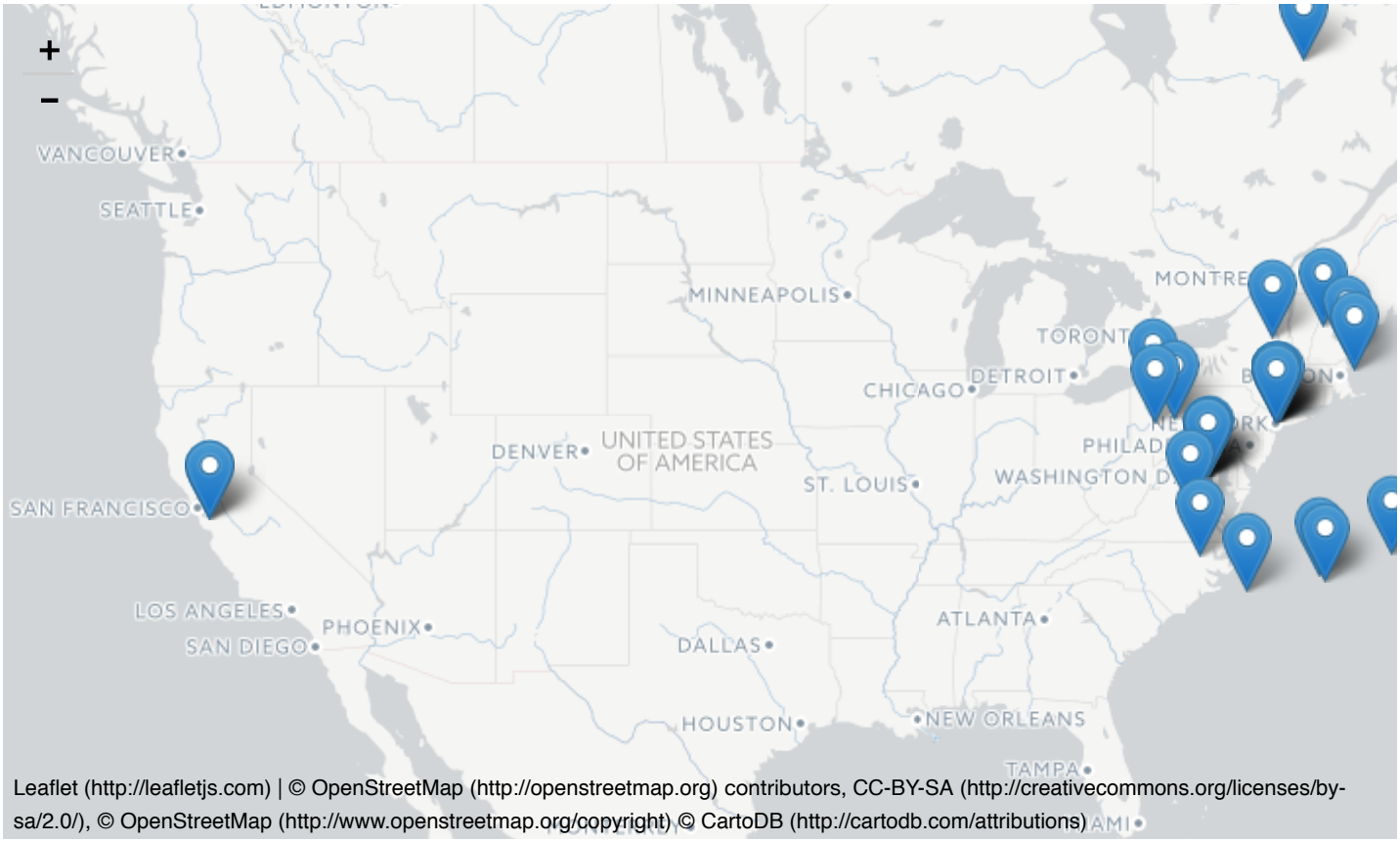

Fig. 18

See what I mean?

Not only are there two(!) lone NYC taxis near San Francisco, but 9 others are actually ocean-going taxis, who knew ;-) . Most of the others will be removed by our previous cleaning limits.

These long-distance locations represent outliers that should be removed to improve the robustness of predictive models.

5.3 Final cleaning

Here we apply the cleaning filters that are discussed above. This code block is likely to expand as the analysis progresses.

Hide

```
train <- train %>%
  filter(trip_duration < 22*3600,
         dist > 0 | (near(dist, 0) & trip_duration < 60),
         jfk_dist_pick < 3e5 & jfk_dist_drop < 3e5,
         trip_duration > 10,
         speed < 100)
```

6 External data

In this playground competition we have been encouraged to supplement our analysis with additional data sources. These are published as Kaggle data set and are being collected in this discussion thread (https://www.kaggle.com/c/nyc-taxi-trip-duration/discussion/36699).

In fact, there are monetary prizes (https://www.kaggle.com/c/nyc-taxi-trip-duration#Prizes) for publishing the top 4 data sets, which together with the Kernel awards gives this competition a fresh and interesting spin.

If you want to know more about how to include multiple data sources in your Kaggle kernel then check out this article (https://www.kaggle.com/product-feedback/32423).

6.1 Weather reports

We start by incorporating a list of *NYC weather data* provided by Mathijs Waegemakers (https://www.kaggle.com/mathijs) right here (https://www.kaggle.com/mathijs/weather-data-in-new-york-city-2016). In the next paragraph I copy the data description verbatim:

> Weather data collected from the National Weather Service. It contains the first six months of 2016, for a weather station in central park. It contains for each day the minimum temperature, maximum temperature, average temperature, precipitation, new snow fall, and current snow depth. The temperature is measured in Fahrenheit and the depth is measured in inches. T means that there is a trace of precipitation.

Of particular interest here will be the rain and snow fall statistics, which I'm curious to compare to the dip in trip numbers in late January. Check also this kernel (https://www.kaggle.com/naveenjafer/weather-and-avg-speed-correlation/) which presents an independent analysis of the same data set.

Note, that this data set has been updated on July 23rd to correct a merging mistake. If your analysis uses a local copy then you should update it.

6.1.1 Data import, overview, formatting, joining

The data can be found in the data set sub-directory of the `../input` directory and it looks like this:

Hide

```
weather <- as.tibble(fread("../input/weather-data-in-new-york-city-2016/weather_data_nyc_centralpark_2016.csv"))
```

Hide

```
glimpse(weather)
```

```
## Observations: 366
## Variables: 7
## $ date                <chr> "1-1-2016", "2-1-2016", "3-1-2016", "4-1-2...
## $ maximum temerature  <int> 42, 40, 45, 36, 29, 41, 46, 46, 47, 59, 40...
## $ minimum temperature <int> 34, 32, 35, 14, 11, 25, 31, 31, 40, 40, 26...
## $ average temperature <dbl> 38.0, 36.0, 40.0, 25.0, 20.0, 33.0, 38.5, ...
## $ precipitation       <chr> "0.00", "0.00", "0.00", "0.00", "0.00", "0...
## $ snow fall           <chr> "0.0", "0.0", "0.0", "0.0", "0.0", "0.0", ...
## $ snow depth          <chr> "0", "0", "0", "0", "0", "0", "0", "0", "0...
```

We turn the *date* into a *lubridate* object and convert the traces ("T") of rain and snow into small numeric amounts. We also save the maximum and minimum temperature (beware the typo!) in a shorter form:

Hide

```
weather <- weather %>%
  mutate(date = dmy(date),
         rain = as.numeric(ifelse(precipitation == "T", "0.01", precipitatio
n)),
         s_fall = as.numeric(ifelse(`snow fall` == "T", "0.01", `snow fall`)),
         s_depth = as.numeric(ifelse(`snow depth` == "T", "0.01", `snow
depth`)),
         all_precip = s_fall + rain,
         has_snow = (s_fall > 0) | (s_depth > 0),
         has_rain = rain > 0,
         max_temp = `maximum temerature`,
         min_temp = `minimum temperature`)
```

Then we join this information to our training data using the common *data* column:

Hide

```
foo <- weather %>%
  select(date, rain, s_fall, all_precip, has_snow, has_rain, s_depth, max_tem
p, min_temp)

train <- left_join(train, foo, by = "date")
```

6.1.2 Visualisation and impact on *trip_duration*

Let's compare the snow fall statistics to our trip numbers per day:

Hide

```
p1 <- train %>%
  group_by(date) %>%
  count() %>%
  ggplot(aes(date,n/1e3)) +
  geom_line(size = 1.5, color = "red") +
  labs(x = "", y = "Kilo trips per day")

p2 <- train %>%
  group_by(date) %>%
  summarise(trips = n(),
            snow_fall = mean(s_fall),
            rain_fall = mean(rain),
            all_precip = mean(all_precip)) %>%
  ggplot(aes(date, snow_fall)) +
  geom_line(color = "blue", size = 1.5) +
  labs(x = "", y = "Snowfall") +
  scale_y_sqrt() +
  scale_x_date(limits = ymd(c("2015-12-28", "2016-06-30")))

p3 <- train %>%
  group_by(date) %>%
  summarise(trips = n(),
            snow_depth = mean(s_depth)) %>%
  ggplot(aes(date, snow_depth)) +
  geom_line(color = "purple", size = 1.5) +
  labs(x = "", y = "Snow depth") +
  scale_y_sqrt() +
  scale_x_date(limits = ymd(c("2015-12-29", "2016-06-30")))

p4 <- train %>%
  group_by(date) %>%
  summarise(median_speed = median(speed)) %>%
  ggplot(aes(date, median_speed)) +
  geom_line(color = "orange", size = 1.5) +
  labs(x = "Date", y = "Median speed")

layout <- matrix(c(1,2,3,4),4,1,byrow=FALSE)
multiplot(p1, p2, p3, p4, layout=layout)
```

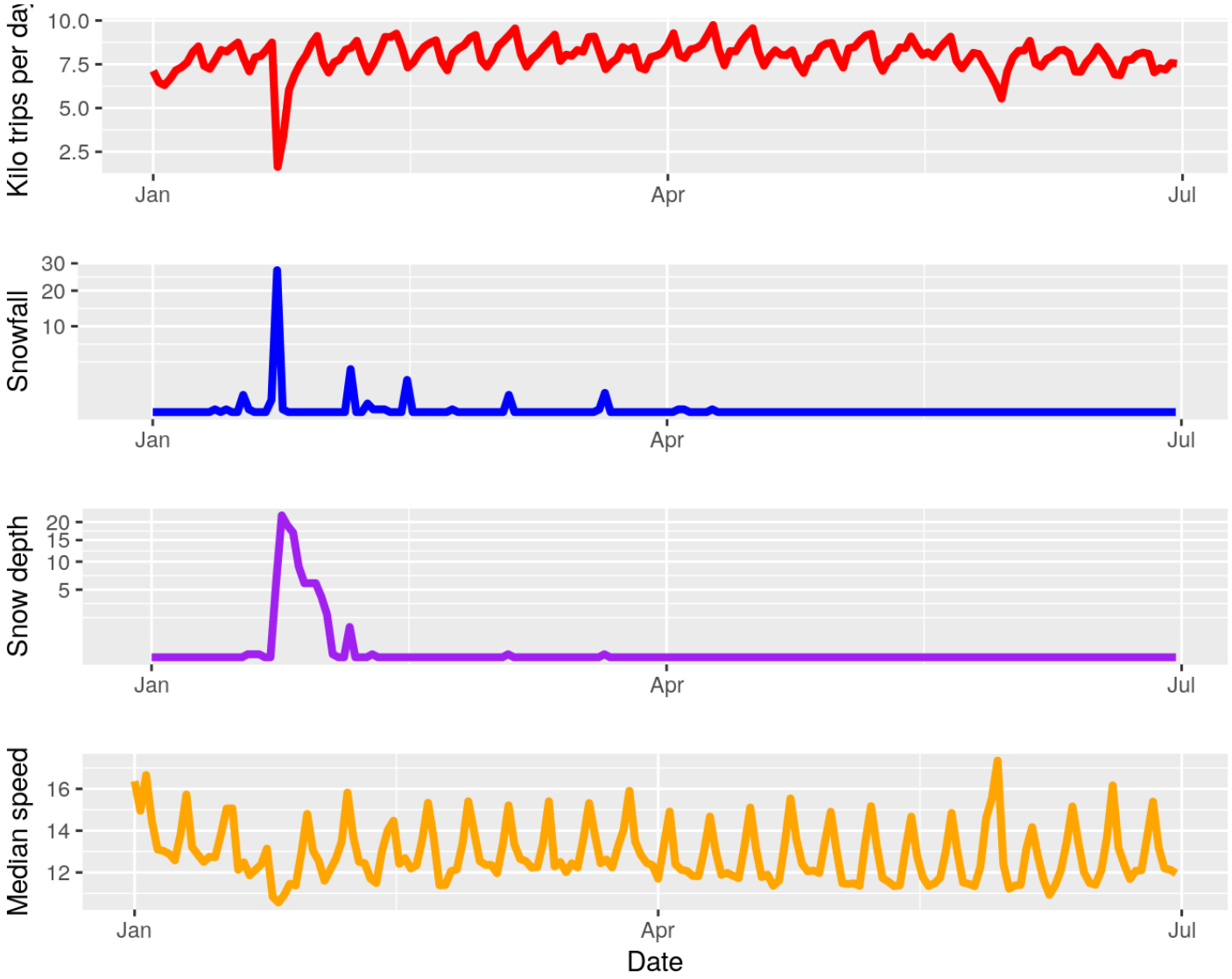

Fig. 19

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1
```

I knew it! The dip in trip volume corresponds to the largest (and first?) snow fall of the winter in NYC (Jan 23rd). In fact, NYC was hit by a blizzard and experienced record-breaking snowfall (https://www.weather.gov/okx/Blizzard_Jan2016). The impact on the traffic patterns was enormous, and the median speed slowed down notably. (Note the square-root y-axis in the two middle plots.)

After this large spike, we clearly see that the following periods of snow fall, albeit signficantly less heavy, have nowhere near as large an effect on the number of taxi trips or their velocities. A possible exception might have been the last snow in mid March, which due to the fact that it hadn't snowed in a while might have caught people by surprise.

Now, do lower numbers of trips in snowy weather also lead to longer journeys? To answer this question we look at a scatter plot between the average trip duration and the total precipitation (rain + snow) for all days in our sample:

Hide

```
train %>%
  group_by(date, has_snow) %>%
  summarise(duration = mean(trip_duration),
            all_precip = mean(all_precip)) %>%
  ggplot(aes(all_precip, duration, color = has_snow)) +
  geom_jitter(width = 0.04, size = 2) +
  scale_x_sqrt() +
  scale_y_log10() +
  labs(x = "Amount of total precipitation", y = "Average trip duration")
```

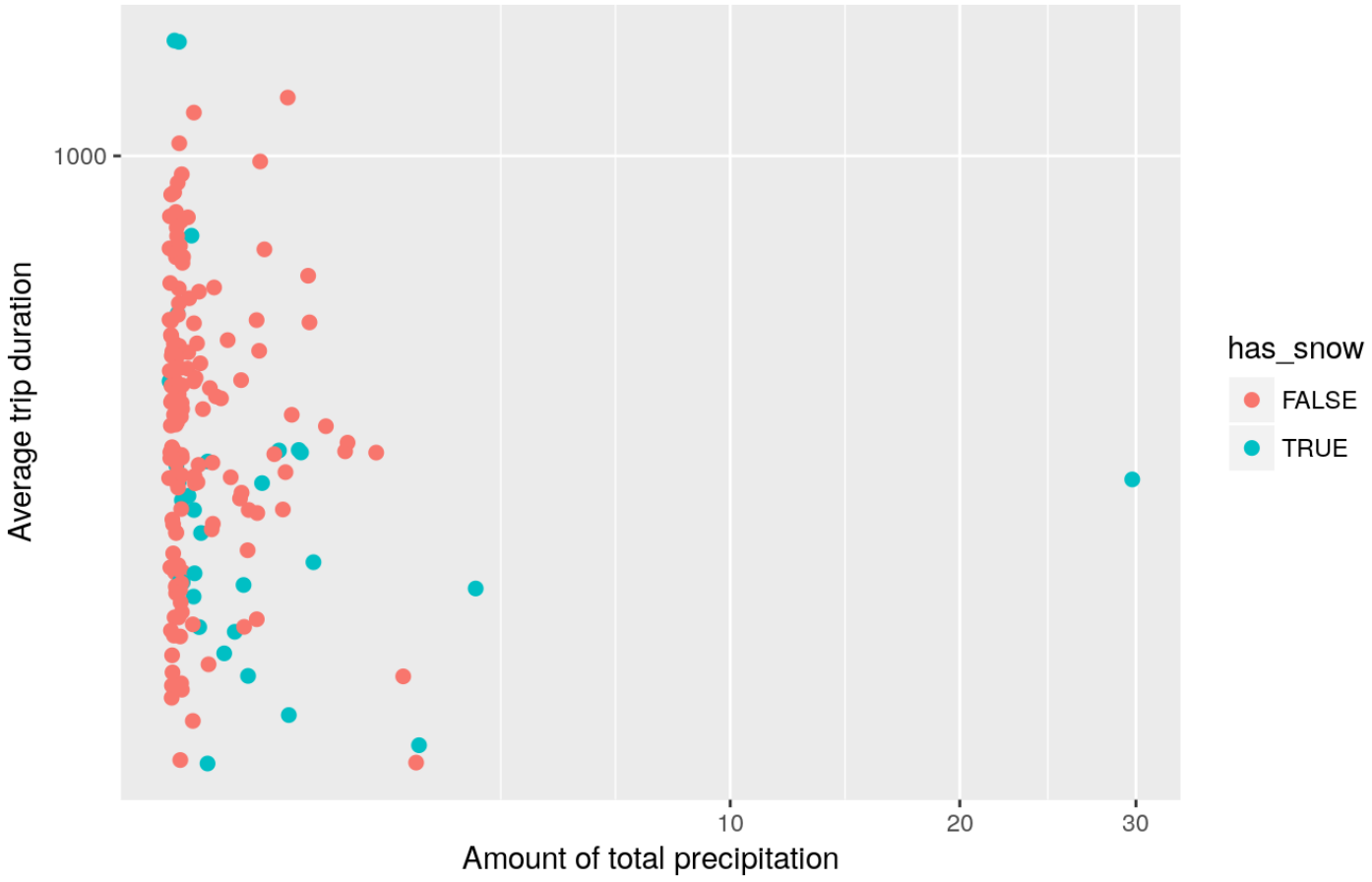

Fig. 20

We find that except for a snow day with long-duration trips it seems more like snow would lead to shorter trips. However, this could simply mean that passengers were more likely to travel shorter distances, so how does the average speed compare?

Hide

```
p1 <- train %>%
  filter(speed < 50) %>%
  ggplot(aes(has_snow, speed, color = has_snow)) +
  geom_boxplot() +
  theme(legend.position = "none") +
  labs(x = "Snowfall")

p2 <- train %>%
  filter(speed < 50) %>%
  ggplot(aes(has_rain, speed, color = has_rain)) +
  geom_boxplot() +
  theme(legend.position = "none") +
  labs(x = "Rainfall")

layout <- matrix(c(1,2),1,2,byrow=FALSE)
multiplot(p1, p2, layout=layout)
```

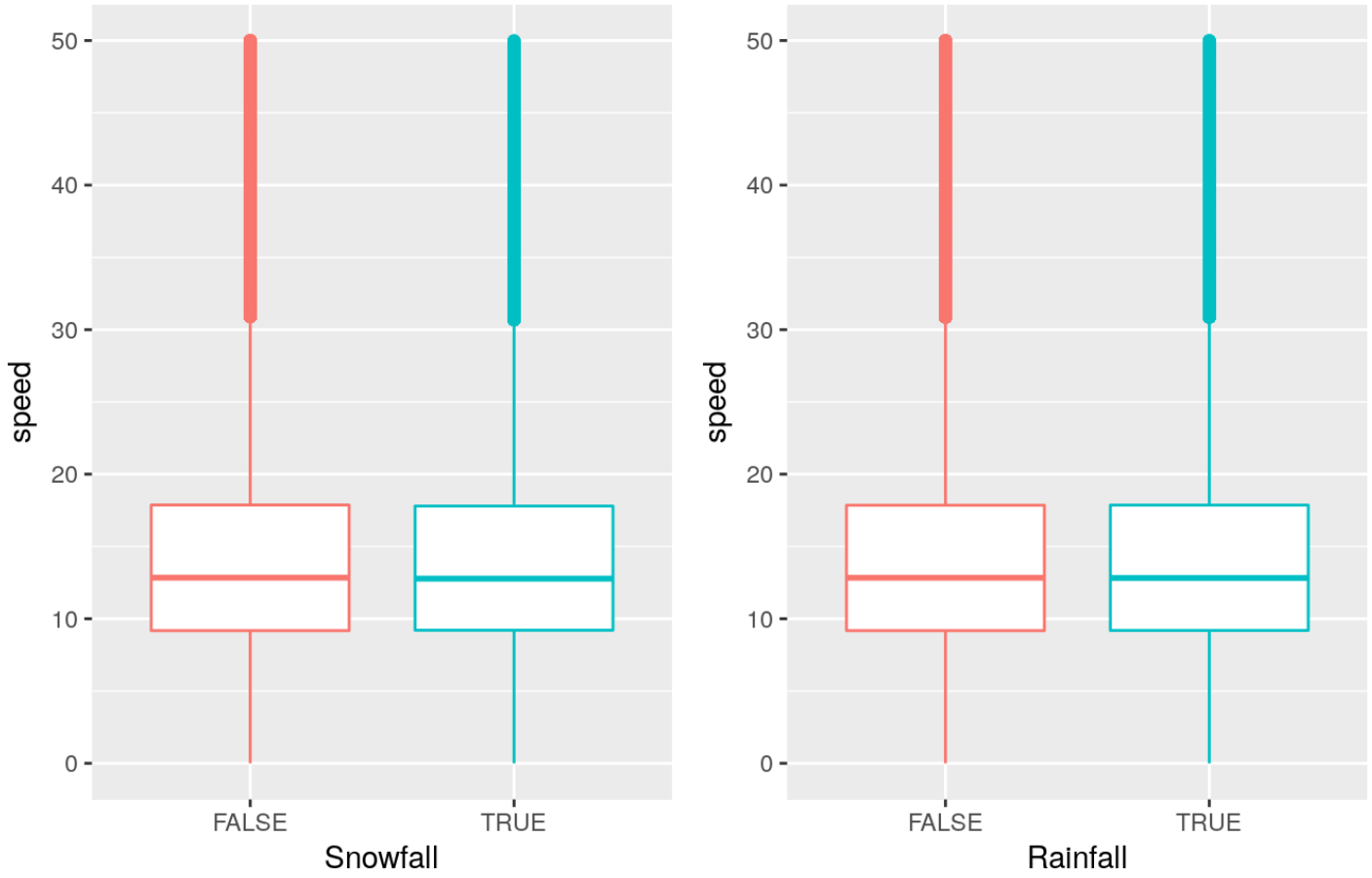

Fig. 21

Hide

```
p1 <- 1; p2 <- 1
```

We find that there is no significant difference here. Moreover, Jan 23rd (blizzard) is not at the top of the list of snow days with the longest median *trip_duration* (it's in the top 10, though).

Hide

```
train %>%
  filter(has_snow == TRUE) %>%
  group_by(date) %>%
  summarise(med_duration = median(trip_duration),
            med_speed = median(speed)) %>%
  arrange(desc(med_duration)) %>%
  head(10)
```

```
## # A tibble: 10 x 3
##          date med_duration med_speed
##        <date>        <dbl>     <dbl>
##  1 2016-01-26          824  10.56575
##  2 2016-01-25          805  10.83328
##  3 2016-01-27          752  10.92293
##  4 2016-01-28          732  11.46148
##  5 2016-01-29          705  11.38243
##  6 2016-02-11          670  11.70530
##  7 2016-03-04          663  11.96139
##  8 2016-02-23          662  11.38501
##  9 2016-01-23          660  12.38501
## 10 2016-01-19          656  12.13287
```

We can spot a different interesting pattern in this table, though, when looking at the top 5 slowest snow days. Interestingly, the overall lowest median speeds were observed *two days after* the blizzard: from Jan 25th until 27th (although note that the day immediately after the blizzard, Jan 24th, was a Sunday):

Hide

```
train %>%
  group_by(date) %>%
  summarise(med_duration = median(trip_duration),
            med_speed = median(speed)) %>%
  arrange(med_speed) %>%
  head(5)
```

```
## # A tibble: 5 x 3
##         date med_duration med_speed
##       <date>        <dbl>     <dbl>
## 1 2016-01-26          824  10.56575
## 2 2016-01-25          805  10.83328
## 3 2016-01-27          752  10.92293
## 4 2016-06-08          768  10.93106
## 5 2016-06-01          761  11.22508
```

Conclusion: From an exploratory point of view this doesn't look too promising. Still, I'm inclined to include the rain and snow fall occurence in a tentative model to see whether it shows any interaction terms that we can't see at the moment. The "blizzard effect" on the (working) week after the heavy snow fall is probably worth taking into account separately. **Thus, we create a *blizzard* feature in our engineering code block.**

6.2 Fastest Routes

Another interesting additional data set is an estimate of the *fastest routes for each trip* provided by oscarleo (https://www.kaggle.com/oscarleo) using the the Open Source Routing Machine, OSRM (http://project-osrm.org/). The data can be found here (https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm) and includes the pickup/dropoff streets and total distance/duration between these two points together with a sequence of travels steps such as turns or entering a highway.

Note, that according to discussion items (https://www.kaggle.com/c/nyc-taxi-trip-duration/discussion/37033) those really are the "fastest", not the shortest, routes between the two points. This will most likely not account for traffic volume, and the fastest route on one day might not be the fastest on another day of the week. Still, here we have an actual driving distance and duration measure that should be valuable for our prediction goal.

6.2.1 Data import and overview

The data comes in three separate files, split into the *train* (2 files) vs *test* trip IDs. Here, we will load and examine the data corresponding to the training observations. Note, that this data set is more than twice as large as our original training data.

Hide

```
foo <- as.tibble(fread("../input/new-york-city-taxi-with-osrm/fastest_routes_train_part_1.csv"))
bar <- as.tibble(fread("../input/new-york-city-taxi-with-osrm/fastest_routes_train_part_2.csv"))
foobar <- as.tibble(fread("../input/new-york-city-taxi-with-osrm/fastest_routes_test.csv"))

fastest_route <- bind_rows(foo, bar, foobar)
```

Hide

```
glimpse(fastest_route)
```

```
## Observations: 2,083,777
## Variables: 12
## $ id                   <chr> "id2875421", "id2377394", "id3504673", "i...
## $ starting_street      <chr> "columbus circle", "2nd avenue", "greenwi...
## $ end_street           <chr> "east 65th street", "washington square we...
## $ total_distance       <dbl> 2009.1, 2513.2, 1779.4, 1614.9, 1393.5, 1...
## $ total_travel_time    <dbl> 164.9, 332.0, 235.8, 140.1, 189.4, 138.8,...
## $ number_of_steps      <int> 5, 6, 4, 5, 5, 5, 2, 6, 9, 4, 4, 5, 4, 11...
## $ street_for_each_step <chr> "columbus circle|central park west|65th s...
## $ distance_per_step    <chr> "0|576.4|885.6|547.1|0", "877.3|836.5|496...
## $ travel_time_per_step <chr> "0|61.1|60.1|43.7|0", "111.7|109|69.9|25....
## $ step_maneuvers       <chr> "depart|rotary|turn|new name|arrive", "de...
## $ step_direction       <chr> "left|straight|right|straight|arrive", "n...
## $ step_location_list   <chr> "-73.982316,40.767869|-73.981997,40.76768...
```

The glimpse view shows us that indeed for each step the streets are named (*streets_for_each_step*) alongside a detailed account of the *distance_per_step*, *travel_time_per_step*, and the *step_maneuvers* and *step_direction* describing the journey. The *step_maneuvers* are named in a pretty self-explanatory way and are also described in the data set overview (https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm). *Depart* and *arrive* are individual steps. The direction of a *step_maneuver* "turn" is listed in the the *step_directions*. The total distance is measured in *meters* and the duration in *seconds*, like in the original data.

6.2.2 New data visualisations

Before joining the "fastest routes" data to our training set, let's first visualise the distributions of the numerical parameters:

Hide

```
p1 <- fastest_route %>%
  ggplot(aes(total_travel_time)) +
  geom_histogram(bins = 150, fill = "red") +
  scale_x_log10() +
  scale_y_sqrt()

p2 <- fastest_route %>%
  ggplot(aes(total_distance)) +
  geom_histogram(bins = 150, fill = "red") +
  scale_x_log10() +
  scale_y_sqrt()

p3 <- fastest_route %>%
  ggplot(aes(number_of_steps)) +
  geom_bar(fill = "red")

p4 <- fastest_route %>%
  ggplot(aes(total_distance, total_travel_time)) +
  geom_bin2d(bins = c(80,80))

layout <- matrix(c(1,2,3,4,4,4),2,3,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```

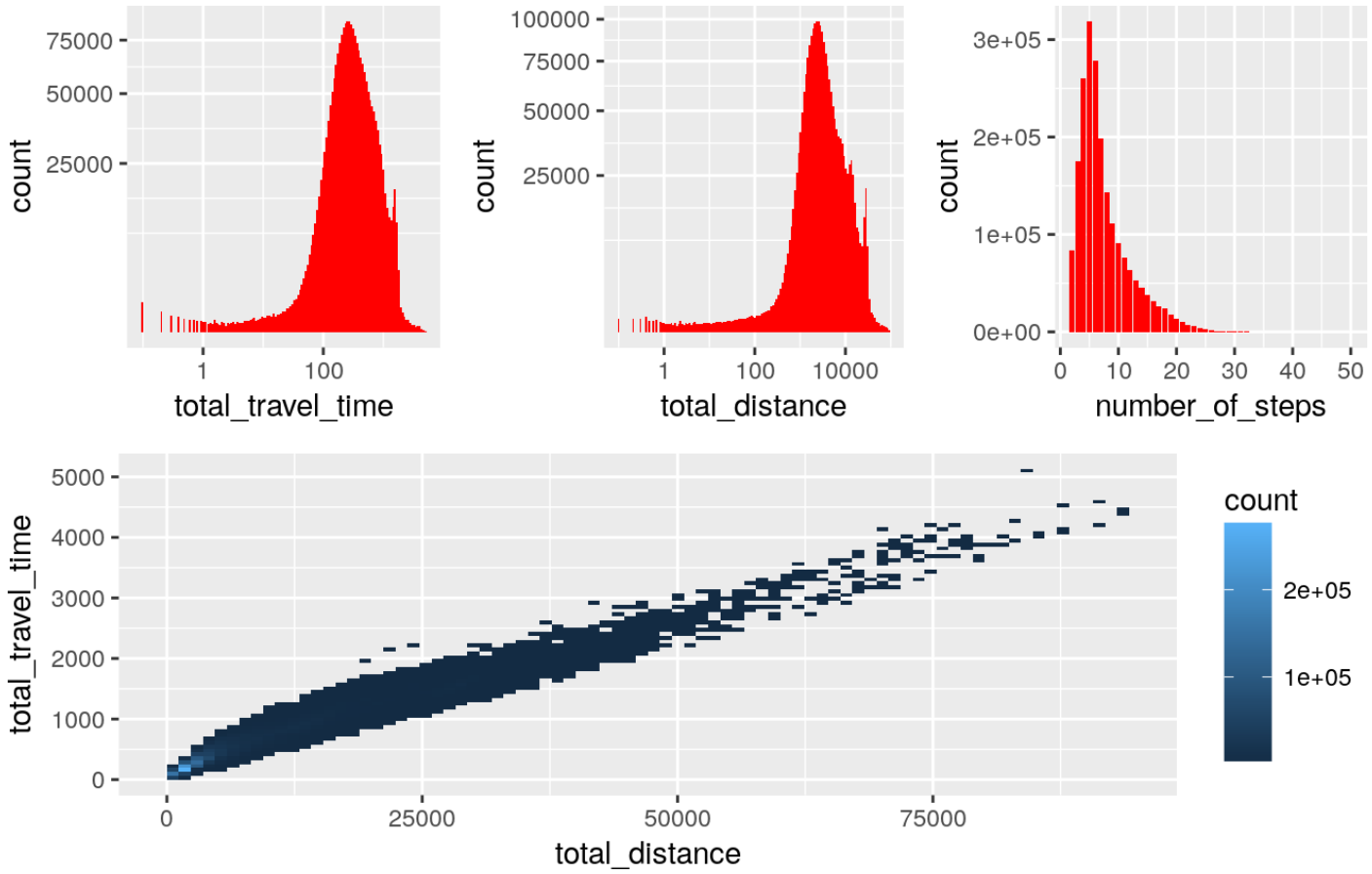

Fig. 22

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1
```

We find:

- The *total_travel_time* and *total_distance* peak at values of around 300 s (5 min) and 2800 m, respectively. These are the median values. Really high values are rare in both features, however there is are indications for secondary peaks towards longer times and distances. Very short distances (few meters) and travel times (few seconds) are present in the data.
- The single most frequent *number_of_steps* is 5. Since “depart” and “arrive” are always present, there are no trips with less than 2 steps. There are almost 60k trips that only contain these two steps, which would mean that they would have at most travelled in a straight line, as far as I can see:

Hide

```
fastest_route %>% filter(number_of_steps == 2) %>% nrow()
```

```
## [1] 83340
```

- Larger number of steps are present, but only about 20% are above 10 and less than 2% are above 20 steps.
- *total_distance* and *total_travel_time* are strongly correlated in a pretty linear way; except for possibly the very short distances. There is a certain amout of scatter around this linear dependency and it looks rather homogeneous over the entire range of distances and times.

Hide

```
fastest_route %>%
  ggplot(aes(factor(number_of_steps), total_travel_time, color = factor(number
_of_steps))) +
  geom_boxplot() +
  labs(x = "number_of_steps") +
  theme(legend.position = "none")
```

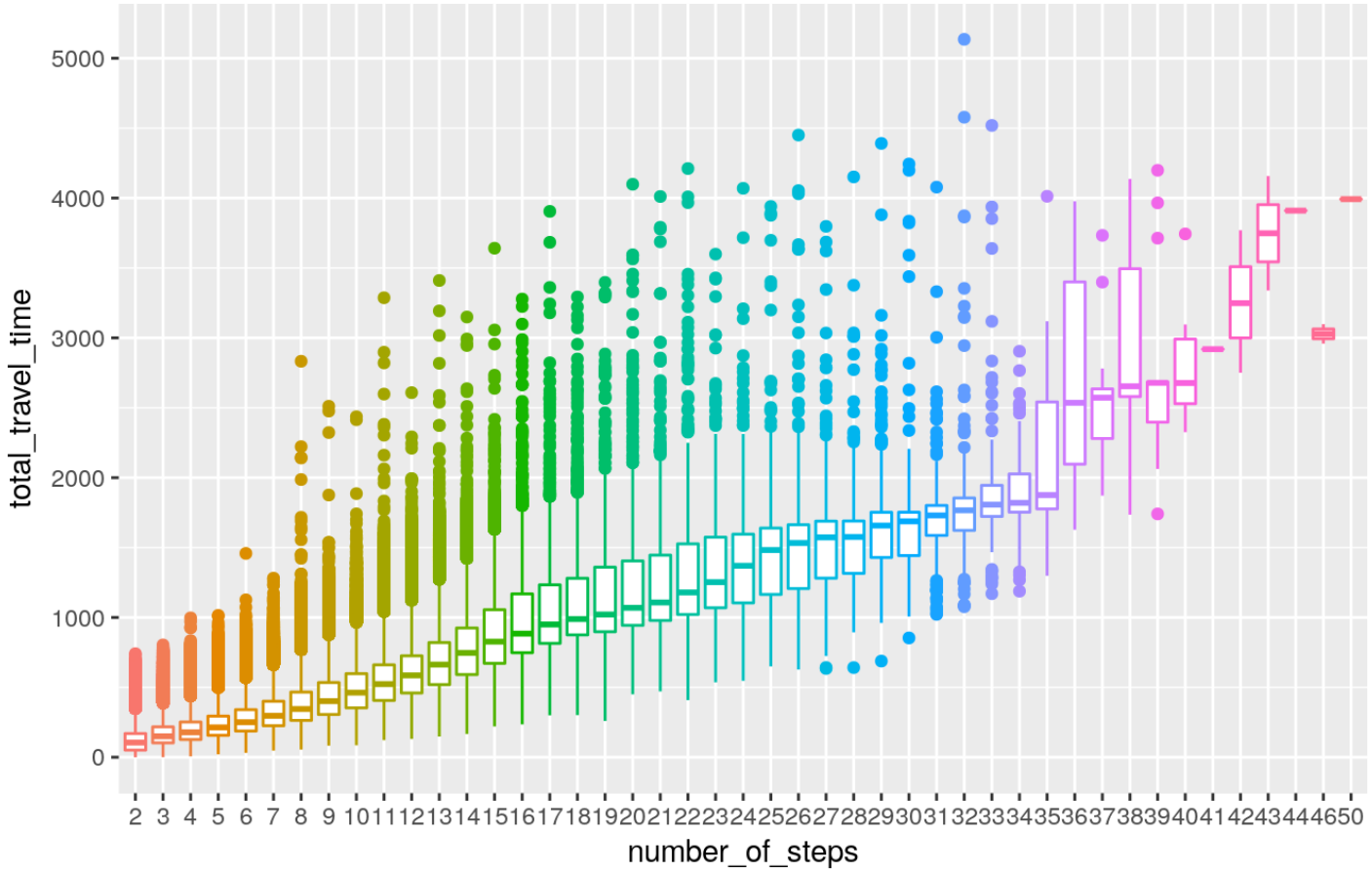

Fig. 23

We also find that the median travel time correlates well with the *number_of_steps*, but that there is also much overlap between neighbouring step counts, along with numerous outliers predominantly towards longer durations.

The distributions of *total_distance* per *number_of_steps* look very similar to the plot above.

6.2.3 Joining and derived features

In a first look at the joined data, we will mainly focus on the *total_distance* and *total_travel_time*, when combined with our original training data set:

Hide

```
foo <- fastest_route %>%
  select(id, total_distance, total_travel_time, number_of_steps,
         step_direction, step_maneuvers) %>%
  mutate(fastest_speed = total_distance/total_travel_time*3.6,
         left_turns = str_count(step_direction, "left"),
         right_turns = str_count(step_direction, "right"),
         turns = str_count(step_maneuvers, "turn")
         ) %>%
  select(-step_direction, -step_maneuvers)

train <- left_join(train, foo, by = "id") %>%
  mutate(fast_speed_trip = total_distance/trip_duration*3.6)
```

In addition, we also create the following new features:

- *fastest_speed* is the average speed of the fastest route based on the OSRM values, and *fast_speed_trip* is the speed assuming the fastest route and the actual trip duration. Both features are in units of km/h.
- *left_turns*, *right_turns*, and *turns* are the total number of those maneuvers per route. The *step_directions* “left” and “right” might not always be associated to the *step_maneuver* “turn”, but in most cases they are so we are using them as a first estimate of the number of left and right turns. Especially left turns might slow down travel.

Beyond that, for instance the detailed list of streets used and their respective *travel_time_per_step* might allow us to take into account known areas of slow traffic.

6.2.4 Visualisation and impact on *trip_duration*

Hide

```
p1 <- train %>%
  ggplot(aes(trip_duration)) +
  geom_density(fill = "red", alpha = 0.5) +
  geom_density(aes(total_travel_time), fill = "blue", alpha = 0.5) +
  scale_x_log10() +
  coord_cartesian(xlim = c(5e1, 8e3))

p2 <- train %>%
  ggplot(aes(dist)) +
  geom_density(fill = "red", alpha = 0.5) +
  geom_density(aes(total_distance), fill = "blue", alpha = 0.5) +
  scale_x_log10() +
  coord_cartesian(xlim = c(2e2, 5e4))

p3 <- train %>%
  ggplot(aes(trip_duration, total_travel_time)) +
  geom_bin2d(bins = c(120,120)) +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  theme(legend.position = "none")

p4 <- train %>%
  ggplot(aes(dist, total_distance)) +
  geom_bin2d(bins = c(70,70)) +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  theme(legend.position = "none")

layout <- matrix(c(1,2,3,4),2,2,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```

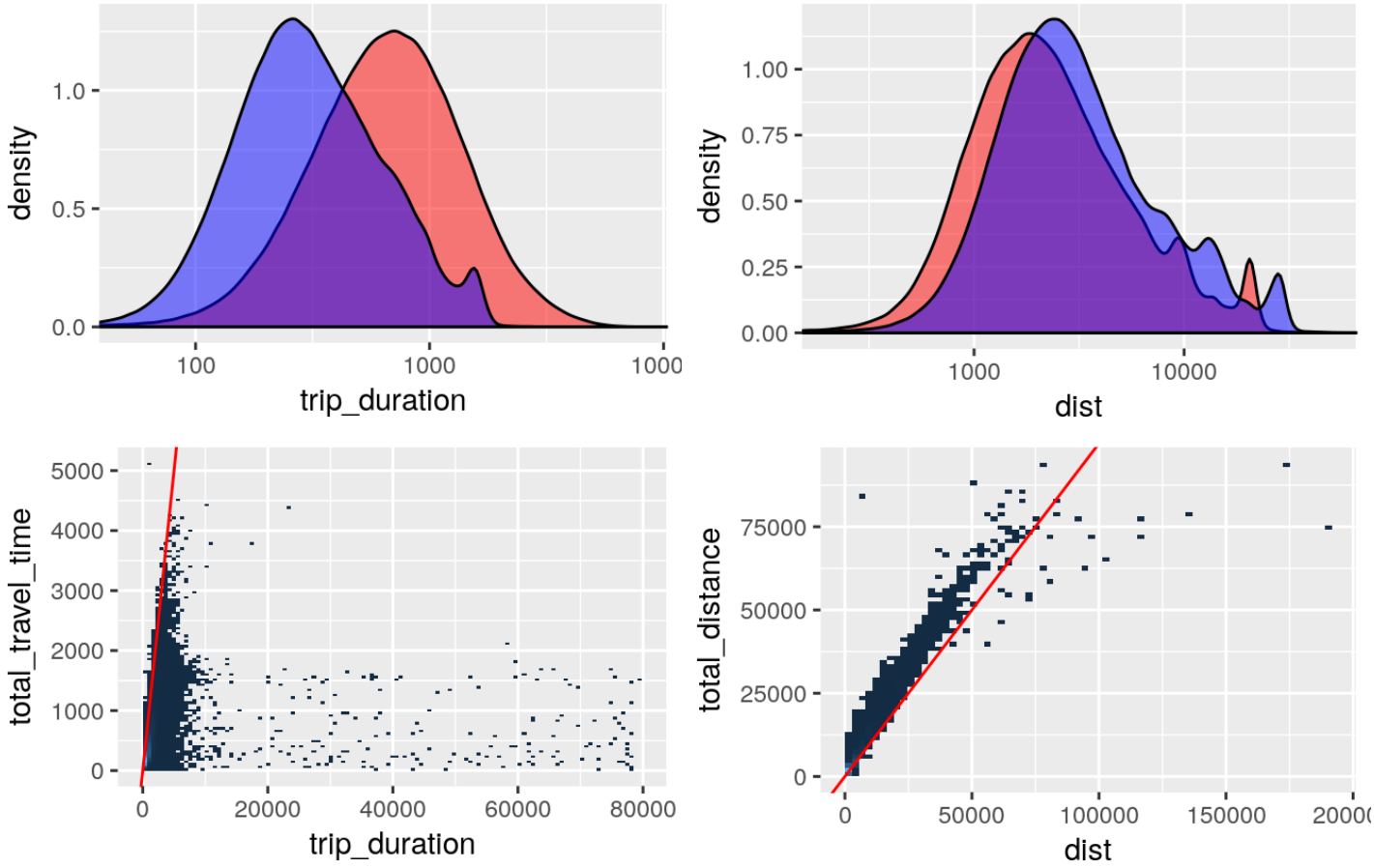

Fig. 24

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1
```

Here the *fastest* features are in blue and the original *trip_duration* and direct *dist*ance in red. In the binned scatter plots the red line has slope 1.

We find:

- The actual *trip_durations* are on average significantly longer than those for the fastest route. The shape of the distributions is also subtly different, with the *fastest* ones having more of a right skew than a left skew in log space. The widths of the distributions, however, are broadly similar.
- The binned scatter plot of *trip_duration* vs *total_travel_time* (lower left corner) reveals that there are numerous ouliers towards *trip_durations*. Interestingly, almost all of those appear below 2000 s (about 0.55 h) of *total_travel_time*. We will study those trips in more detail below.
- The direct distances (as the crow flies) are longer than the *fastest* distances, but not by that much. By and large, the direct *dist*ance is a pretty good proxy for the actual, *fastest* distance according to the binned scatter plot. There are (rare) outliers towards longer direct *dist*ances, which indicates potentially problematic data entries.

What about the *speed*? Comparing our original *speed* estimate to the *fastest_speed* from the OSRM data should give us an idea how realistic the values of the former feature are.

(For this hidden figure, in the kernel-density plot the *fastest* values are in blue and the red line in the binned scatter plot has slope equal one.)

Hide

```
p1 <- train %>%
  ggplot(aes(speed)) +
  geom_density(fill = "red", alpha = 0.5) +
  geom_density(aes(fastest_speed), fill = "blue", alpha = 0.5) +
  coord_cartesian(xlim = c(0, 80))

p2 <- train %>%
  ggplot(aes(speed, fastest_speed)) +
  geom_bin2d(bins = c(50,50)) +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  theme(legend.position = "none")

layout <- matrix(c(1,2),1,2,byrow=TRUE)
multiplot(p1, p2, layout=layout)
p1 <- 1; p2 <- 1
```

Here we find several effects:

- By not accounting for traffic, our *fastest* speeds are always above 20 km/h (12 mph), a value that is not frequently reached in our actual data.
- The theoretical speed disribution shows several peaks, most prominently a very sharp one between 25 and 30 km/h (15 - 18 mph) and a broader one around 40 km/h (25 mph). Without knowing very much about the traffic management in NYC I would assume that they correspond to zones of different speed limits and their linear combinations.
- There is no correlation between *speed* and *fastest_speed*. This reflects the large number of delays in the scatter plot of *trip_duration* vs *total_travel_time* which we will study in a separate step.

At face value, the impact of the number of *turns* on the *total_travel_time* is very similar for *left_turns* and *right_turns*. A main reason behind this will be that the *total_travel_time* increases with increasing *number_of_steps* of any kind and that the number of *turns* is of

course correlated with the *number_of_steps*. We will learn more from the percentage of turns per trip, which we will investigate in a moment. From these hidden plots we find the following results:

Hide

```
p1 <- train %>%
  ggplot(aes(factor(left_turns), total_travel_time, color = factor(left_turn
s))) +
  geom_boxplot() +
  labs(x = "Number of left turns") +
  theme(legend.position = "none")

p2 <- train %>%
  ggplot(aes(factor(right_turns), total_travel_time, color = factor(right_turn
s))) +
  geom_boxplot() +
  labs(x = "Number of right turns") +
  theme(legend.position = "none")

p3 <- train %>%
  ggplot(aes(factor(turns), total_travel_time, color = factor(turns))) +
  geom_boxplot() +
  labs(x = "Total number of turns") +
  theme(legend.position = "none")

layout <- matrix(c(1,2,3,3),2,2,byrow=TRUE)
multiplot(p1, p2, p3, layout=layout)

p1 <- 1; p2 <- 1; p3 <- 1
```

- There is a “NA” entry here which is caused by a single observation (“id3008062”) which is missing from the *fastest_routes* data. This is a very minor issue without any practical impact on our predictions, but it’s good to be aware of this “NA” entry. It turns out (https://www.kaggle.com/oscarleo/new-york-city-taxi-with-osrm/discussion/37337) that OSRM does not provide a route for this data point.
- Right turns and slightly more frequent than left turns (up to 21 vs 19).
- For the total number of *turns* there is a first maximum in the median *total_travel_time* at 13 turns, after which the median declines somewhat.

Now let’s move on to the turn percentages:

Hide

```
foo <- train %>%
  filter(!is.na(left_turns)) %>%
  mutate(left_frac = left_turns/number_of_steps,
         right_frac = right_turns/number_of_steps)

p1 <- foo %>%
  ggplot(aes(left_frac)) +
  geom_histogram(bins = 20, fill = "red") +
  labs(x = "Left turns / all steps")

p2 <- foo %>%
  ggplot(aes(right_frac)) +
  geom_histogram(bins = 20, fill = "red") +
  labs(x = "Right turns / all steps")

p3 <- foo %>%
  ggplot(aes(left_frac, total_travel_time)) +
  geom_bin2d(bins = c(40,40)) +
  theme(legend.position = "none") +
  labs(x = "Left turns / all steps")

p4 <- foo %>%
  ggplot(aes(right_frac, total_travel_time)) +
  geom_bin2d(bins = c(40,40)) +
  theme(legend.position = "none") +
  labs(x = "Right turns / all steps")

p5 <- foo %>%
  ggplot(aes(left_frac, fastest_speed)) +
  geom_bin2d(bins = c(40,40)) +
  theme(legend.position = "none") +
  labs(x = "Left turns / all steps")

p6 <- foo %>%
  ggplot(aes(right_frac, fastest_speed)) +
  geom_bin2d(bins = c(40,40)) +
  theme(legend.position = "none") +
  labs(x = "Right turns / all steps")
```

```
layout <- matrix(c(1,2,3,4,5,6),3,2,byrow=TRUE)
multiplot(p1, p2, p3, p4, p5, p6, layout=layout)
```

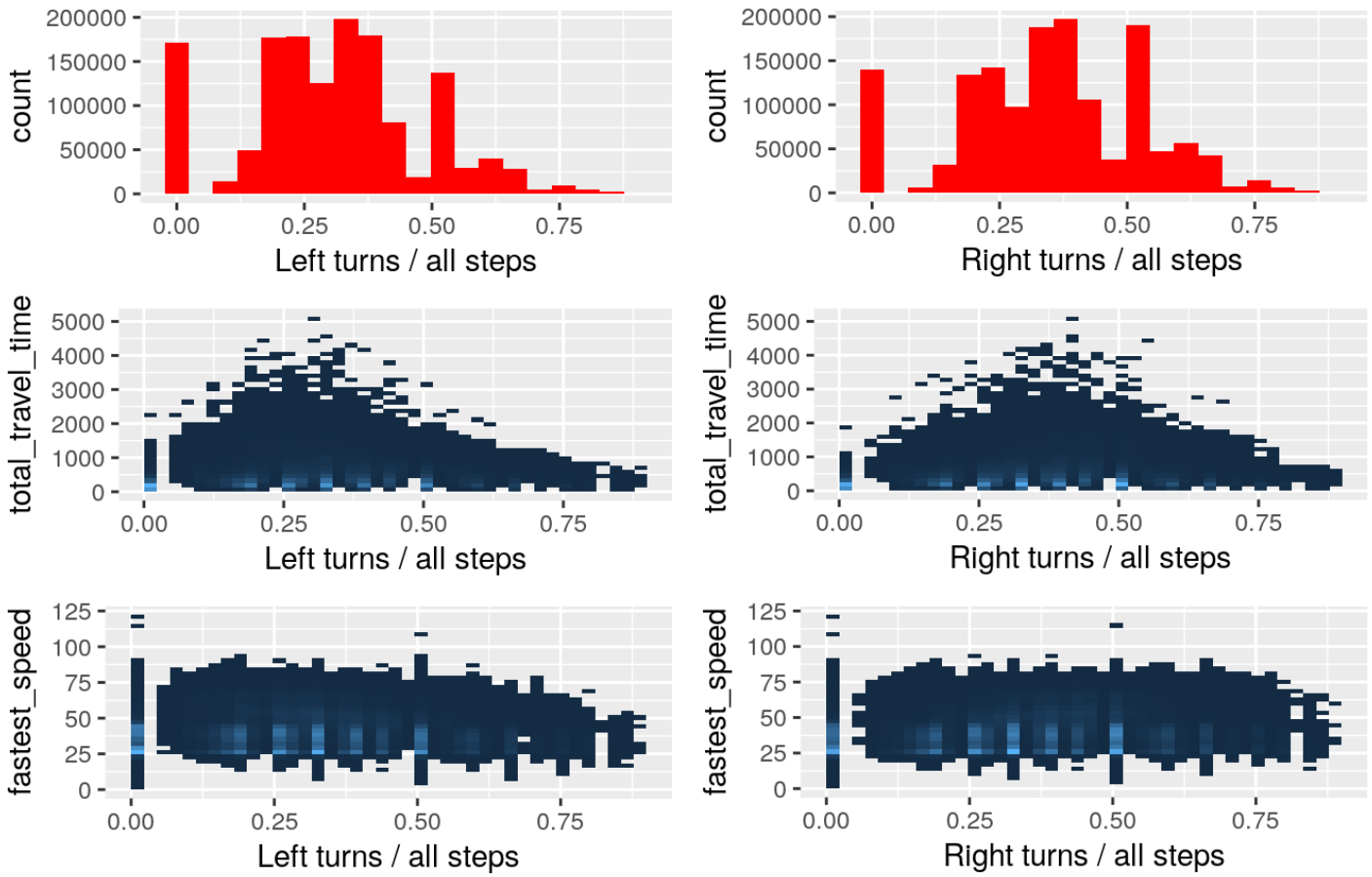

Fig. 25

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1; p5 <- 1; p6 <- 1; foo <- 1
```

We find:

- Right turns are indeed somewhat more frequent than left turns. Both kind of turns make up mostly less than 50% of all travel maneuvers but can reach 80%. There is a significant number of trips without any turns at all.
- A higher fraction of turns does *not* correlate with higher *total_travel_times*. Instead, the longest times are reached at around 1/3 of maneuvers being turns.
- On the other hand, the *fastest_speed* (i.e. *total_distance* / *total_travel_time*) shows indications for lower average values for larger number of *left_turns* for the percentage range of 50% to 80%. Notably, there is no such obvious impact for the number of *right_turns* up until 75%. Thus, excessive numbers of *left* turns really appear to be slowing down our travel speed.

6.2.5 *trip_duration* vs *total_travel_time* - a look at curious delays

In this section we will compare the OSRM *total_travel_time* to the taxi *trip_duration* in our original data. We will specifically focus on understanding those curious cases that suggest significant delays in our data.

Here we plot the a scatter plot of the two timing features colour-coded by the percentage of left turns in the the OSRM route (the solid red line has again a slope of 1):

Hide

```
train %>%
  ggplot(aes(trip_duration, total_travel_time, color = left_turns / number_of_
steps)) +
  geom_point(alpha = 0.3) +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  scale_colour_gradient(low = "blue", high = "red") +
  scale_x_log10() +
  scale_y_log10() +
  geom_hline(yintercept = c(50,2000), linetype = 2) +
  labs(color = "Left turns [%]")
```

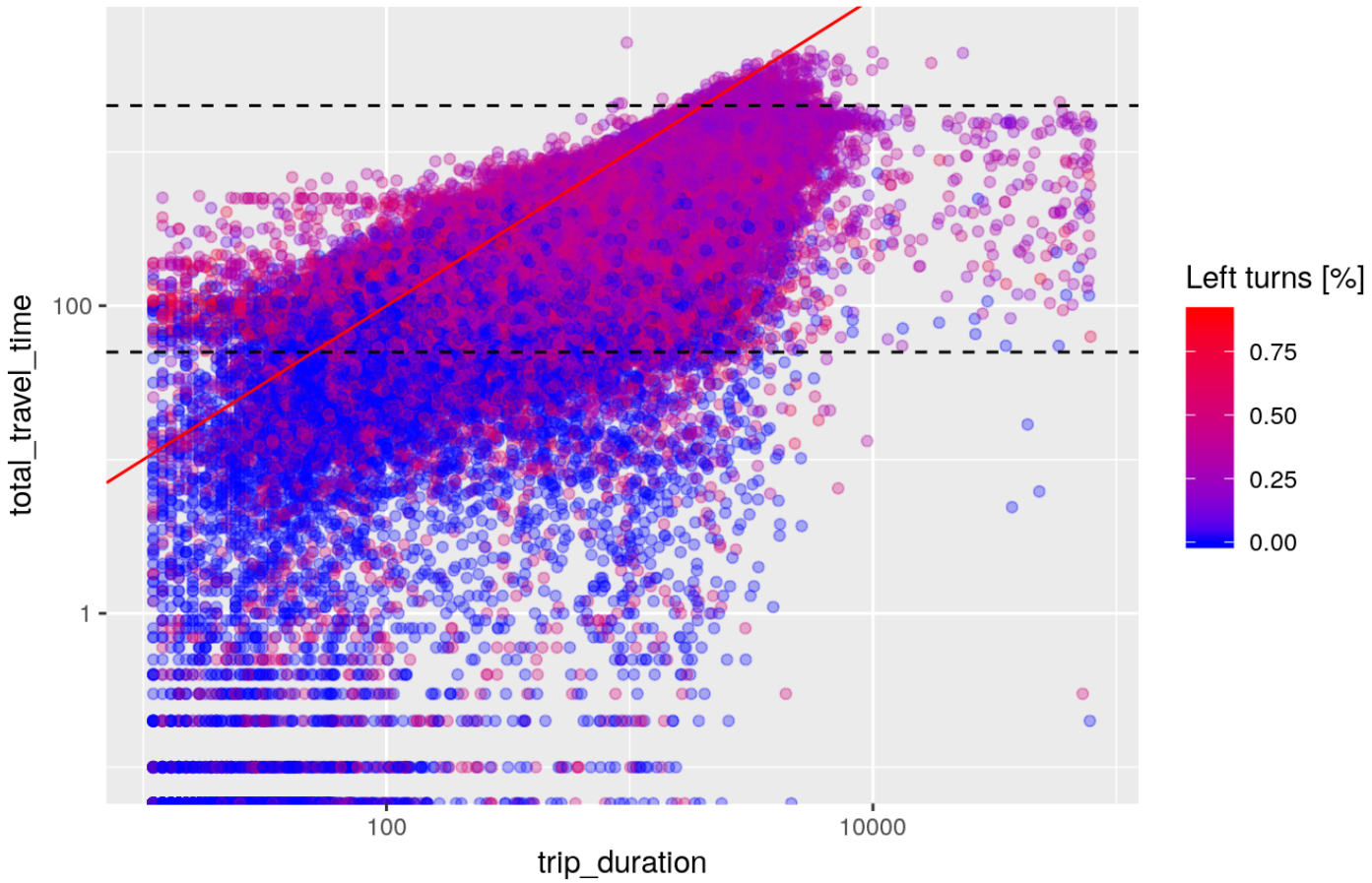

Fig. 26

Note the double-log axes.

We find:

- There is a population of data point where the *trip_duration* (the actual timing of the taxi ride) is lower than the *total_travel_time* based on the fastest OSRM route. These differences can be significant for the trips that take less than 1000 s (i.e. about 17 min) on the fastest route but become less severe for rides longer than that.
- There are no “fastest” routes with more than 10 ks *total_travel_time*. The maximum in the cleaned sample is at about 5000 s or 1.5 h. Almost all of the significant long-delay outliers in *trip_duration* (indicated by the horizontal dashed lines) lie in the range of 100 - 2000 s *total_travel_time* and extend beyond 10 ks *trip_duration*.
- There appears to be a tendency for longer trips (especially in *total_travel_time*) to contain a higher fraction of *left_turns*. However, this apparent effect might be due to the shortest rides containing no turns and longer rides (above 1 min) including an average number of turns. In any case, within the long-delay outliers there is no obvious impact of the percentage of *left_turns*.

The most prominent outliers only make up a very small percentage of the overall rides:

Hide

```
train %>%
  filter(total_travel_time > 50 & total_travel_time < 2000) %>%
  mutate(delay = trip_duration > 1e4) %>%
  group_by(delay) %>%
  count()
```

```
## # A tibble: 2 x 2
## # Groups:   delay [2]
##   delay       n
##   <lgl>   <int>
## 1 FALSE 1434779
## 2  TRUE     207
```

We will briefly examine those data points using a comparison of relative frequencies in bar plots and density overlays:

Hide

```
foo <- train %>%
  filter(total_travel_time > 50 & total_travel_time < 2000) %>%
  mutate(delay = trip_duration > 1e4)

p1 <- foo %>%
  ggplot(aes(vendor_id, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p2 <- foo %>%
  ggplot(aes(jfk_trip, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p3 <- foo %>%
  ggplot(aes(lg_trip, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p4 <- foo %>%
  ggplot(aes(work, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p5 <- foo %>%
  ggplot(aes(total_distance, fill = delay)) +
  geom_density(alpha = 0.5)
```

```
layout <- matrix(c(1,2,3,4,5,5),2,3,byrow=TRUE)
multiplot(p1, p2, p3, p4, p5, layout=layout)
```

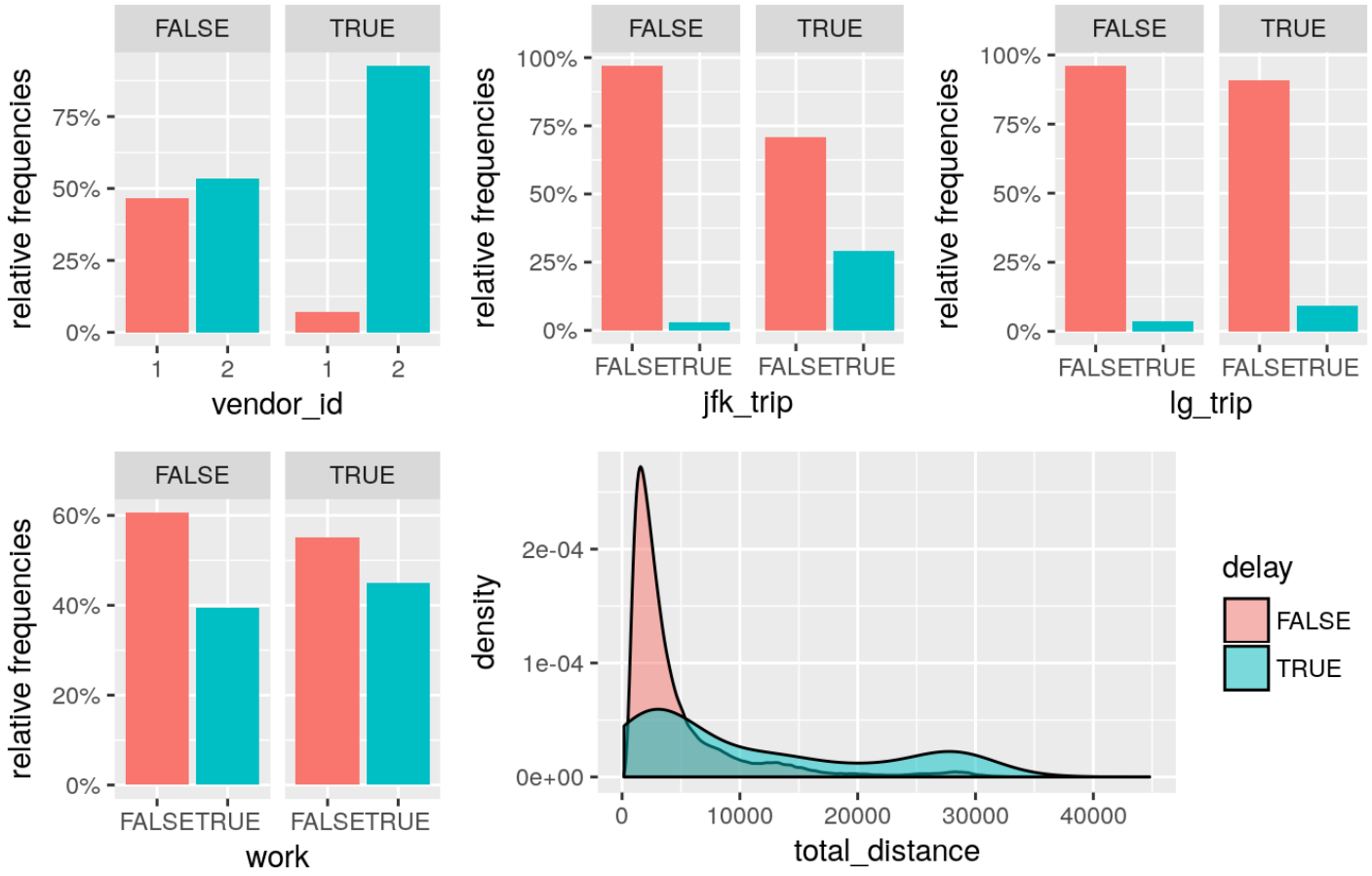

Fig. 27

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1; p5 <- 1; p6 <- 1
```

Here the facets (true vs false) refer to the observations being part of the long-delay group or not.

We find:

- Almost all long delays happen for the taxis of vendor 2, even though vendor 2 has only a slightly higher percentage of rides overall. This discrepancy is definitely significant enough to explain most of the effect.
- Another interesting fact is that in the long-delay sample the fraction of JFK trips is significantly higher, and the fraction of La Guardia trips is slightly higher, than in the general population. We could hypothesise that for an airport journey the *trip_duration* might include the waiting time while queuing for passengers. JFK trips make up almost 30% of the long-delay trips (60 out of 207).

- The percentage of delays during *work* hours is slightly higher but not with practical significance.
- Furthermore, in the long-delay distribution the long distances are clearly favoured over shorter ones. Remember that here we are only comparing trips within the same bracket of *total_travel_time* (50 - 2000 s).
- In addition, not shown here is the observation that long delays appear to be slightly more frequent on Tuesdays or in January and March; compared to Wednesday and Thursdays or February. However, considering the number of categories and that the deviations at most reach 5 percentage points these effect might be random.

Another curious group of outliers can be found in the lower middle portion of the *trip_duration* vs *total_travel_time* plot (hidden figure; where we encode the colour of the trips by their *vendor_id*, to further highlight the impact of vendor 2 on the long-duration outliers on the top right and the few ones on the bottom right).

Hide

```
train %>%
  ggplot(aes(trip_duration, total_travel_time, color = vendor_id)) +
  geom_point(alpha = 0.3) +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  scale_x_log10() +
  scale_y_log10() +
  geom_hline(yintercept = c(10), linetype = 2) +
  geom_vline(xintercept = c(600,5000), linetype = 2)
```

The second group of outliers lies at *total_travel_time < 10* and *trip_duration = 600 - 4000*; units in seconds as usual. See the hidden figure or fig 26. Therefore, those are trips over very short distances that should have only taken a few seconds, but in our data they took between 10 minutes and 1.4 hours. In a way, there is a continuum of data points in the lower left to middle part of the plot, but the ones over 10 minutes show another peak in density:

Hide

```
train %>%
  filter(total_travel_time < 10 & trip_duration < 10000) %>%
  ggplot(aes(trip_duration)) +
  geom_density(fill = "red") +
  geom_vline(xintercept = c(600,5000), linetype = 2) +
  scale_x_log10()
```

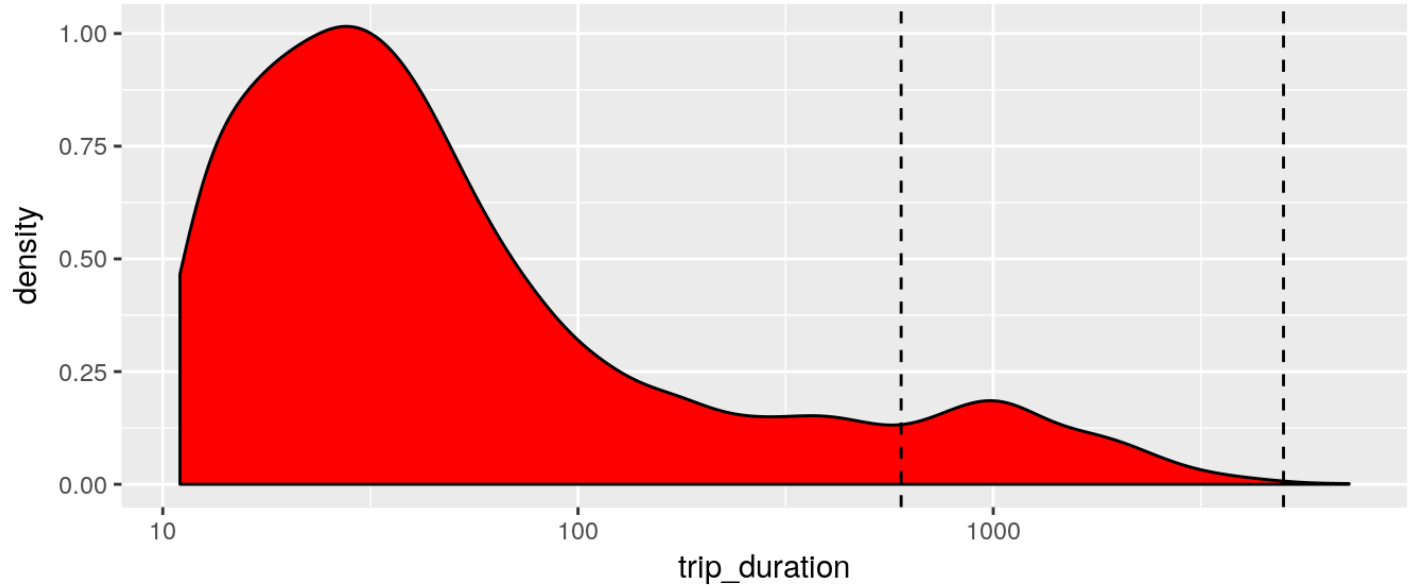

Fig. 28

Hide

```
foo <- train %>%
  filter(total_travel_time < 10) %>%
  mutate(delay = trip_duration > 600)

p1 <- foo %>%
  ggplot(aes(vendor_id, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p2 <- foo %>%
  ggplot(aes(wday, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p3 <- foo %>%
  ggplot(aes(passenger_count, group = delay)) +
  geom_bar(aes(y = ..prop.., fill = factor(..x..)), stat="count") +
  scale_y_continuous(labels=scales::percent) +
  ylab("relative frequencies") +
  facet_grid(~delay) +
  theme(legend.position = "none")

p4 <- foo %>%
  ggplot(aes(total_distance, fill = delay)) +
  geom_density(alpha = 0.5) +
  coord_cartesian(xlim = c(0,100))

layout <- matrix(c(1,1,2,2,2,2,3,3,3,4,4,4),2,6,byrow=TRUE)
multiplot(p1, p2, p3, p4, layout=layout)
```

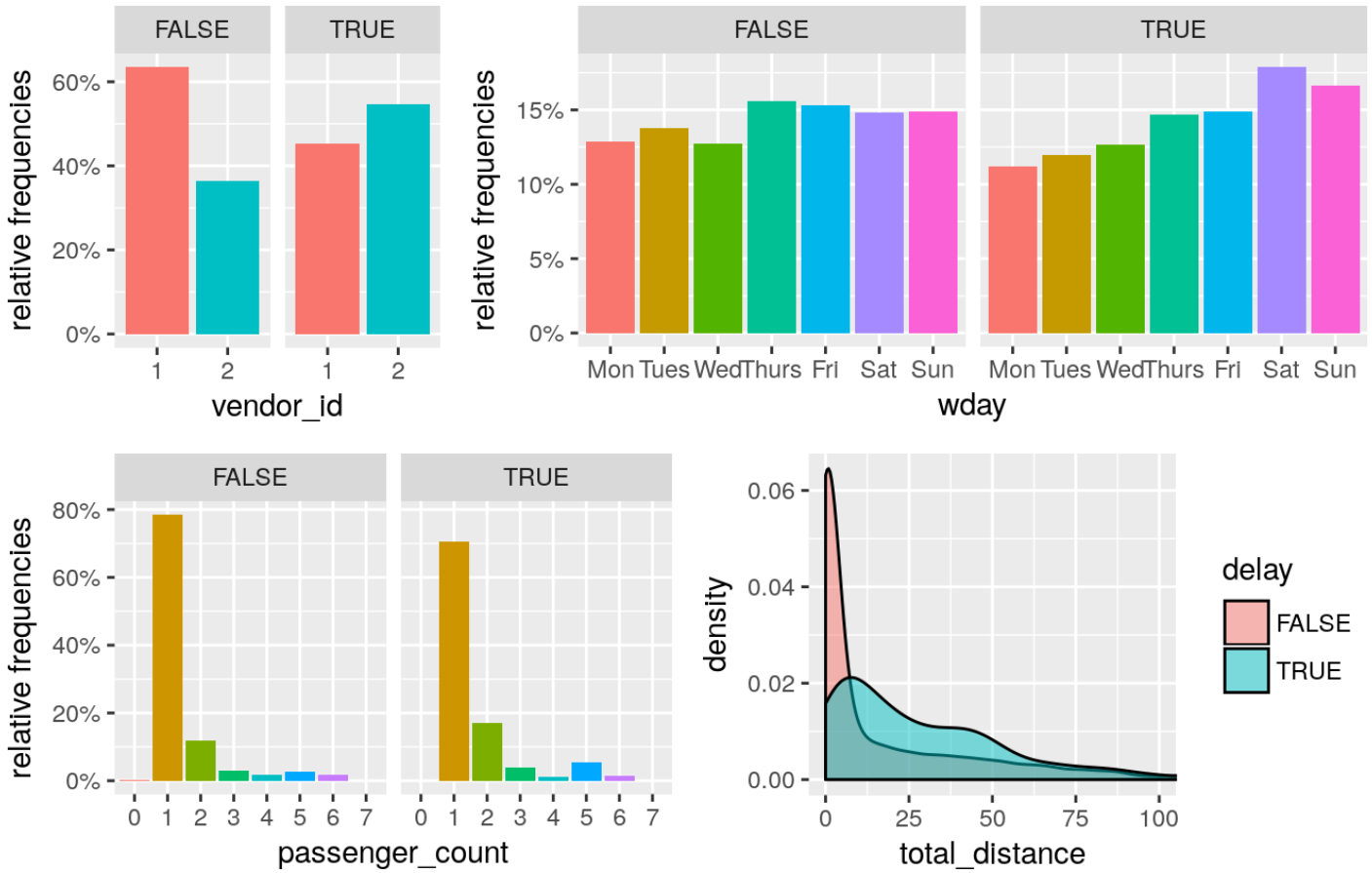

Fig. 29

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1; p5 <- 1; p6 <- 1
```

We find:

- Again, vendor 2 has systematically longer *trip_durations* than vendor 1.
- Delays of this second kind appear to be more frequent on weekends, compared to the overall distribution.
- The number of passengers is higher too for these delays, so maybe we are counting the time it takes them (and their luggage) to get in and out of the taxi?
- The distances for the delays are again longer, but notice the different scale here. Distances of less than 100 m should not take more than 10 min.
- Not shown: *jfk_trip* and *lg_trip* have no impact here because the overall distances are so short.

In summary, we find that the *vendor_id* has distinguishing power for the two outlier groups, with airport trips very likely to contribute to the long-delay outliers. Other factors like *wday* or *passenger_count* might contribute additional signal.

7 Correlations overview

After engineering new features and before starting the modelling, we will visualise the relations between our parameters using a *correlation matrix*. For this, we need to change all the input features into a numerical format. The visualisation uses the *corrplot* function from the eponymous package. *Corrplot* gives us great flexibility in manipulating the style of our plot.

What we see below, are the colour-coded *correlation coefficients* for each combination of two features. In simplest terms: this shows whether two features are connected so that one changes with a predictable trend if you change the other. The closer this coefficient is to zero the weaker is the correlation. Both 1 and -1 are the ideal cases of perfect correlation and anti-correlation (dark blue and dark red in the plots below).

Here, we are of course interested if and how strongly our correlate with the *trip_duration*, the prediction of which is the ultimate goal of this challenge. But we also want to know whether our potential predictors are *correlated among each other*, so that we can reduce the collinearity in our data set and improve the robustness of our prediction. (The full correlation plot is included as a hidden figure.)

Hide

```
train %>%
  select(-id, -pickup_datetime, -dropoff_datetime, -jfk_dist_pick,
         -jfk_dist_drop, -lg_dist_pick, -lg_dist_drop, -date) %>%
  mutate(passenger_count = as.integer(passenger_count),
         vendor_id = as.integer(vendor_id),
         store_and_fwd_flag = as.integer(as.factor(store_and_fwd_flag)),
         jfk_trip = as.integer(jfk_trip),
         wday = as.integer(wday),
         month = as.integer(month),
         work = as.integer(work),
         lg_trip = as.integer(lg_trip),
         blizzard = as.integer(blizzard),
         has_snow = as.integer(has_snow),
         has_rain = as.integer(has_rain)) %>%
  select(trip_duration, speed, everything()) %>%
  cor(use="complete.obs", method = "spearman") %>%
  corrplot(type="lower", method="circle", diag=FALSE)
```

If we remove the features with little to no significant impact on other features that are not directly related then our (effective) correlation matrix looks like this. In this plot (and in the hidden figure) the fading serves the purpose that everything that would be hard to see is not

worth seeing:

Hide

```
foo <- train %>%
  select(-id, -pickup_datetime, -dropoff_datetime, -jfk_dist_pick,
         -jfk_dist_drop, -lg_dist_pick, -lg_dist_drop, -date,
         -store_and_fwd_flag, -hour, -rain, -s_fall, -all_precip,
         -has_rain, -s_depth, -min_temp, -max_temp,
         -wday, -right_turns, -turns, -fast_speed_trip) %>%
  mutate(passenger_count = as.integer(passenger_count),
         vendor_id = as.integer(vendor_id),
         jfk_trip = as.integer(jfk_trip),
         lg_trip = as.integer(lg_trip),
         month = as.integer(month),
         work = as.integer(work),
         blizzard = as.integer(blizzard),
         has_snow = as.integer(has_snow)) %>%
  select(trip_duration, speed, everything())

foo %>%
  cor(use="complete.obs", method = "spearman") %>%
  corrplot(type="lower", method="circle", diag=FALSE)
```

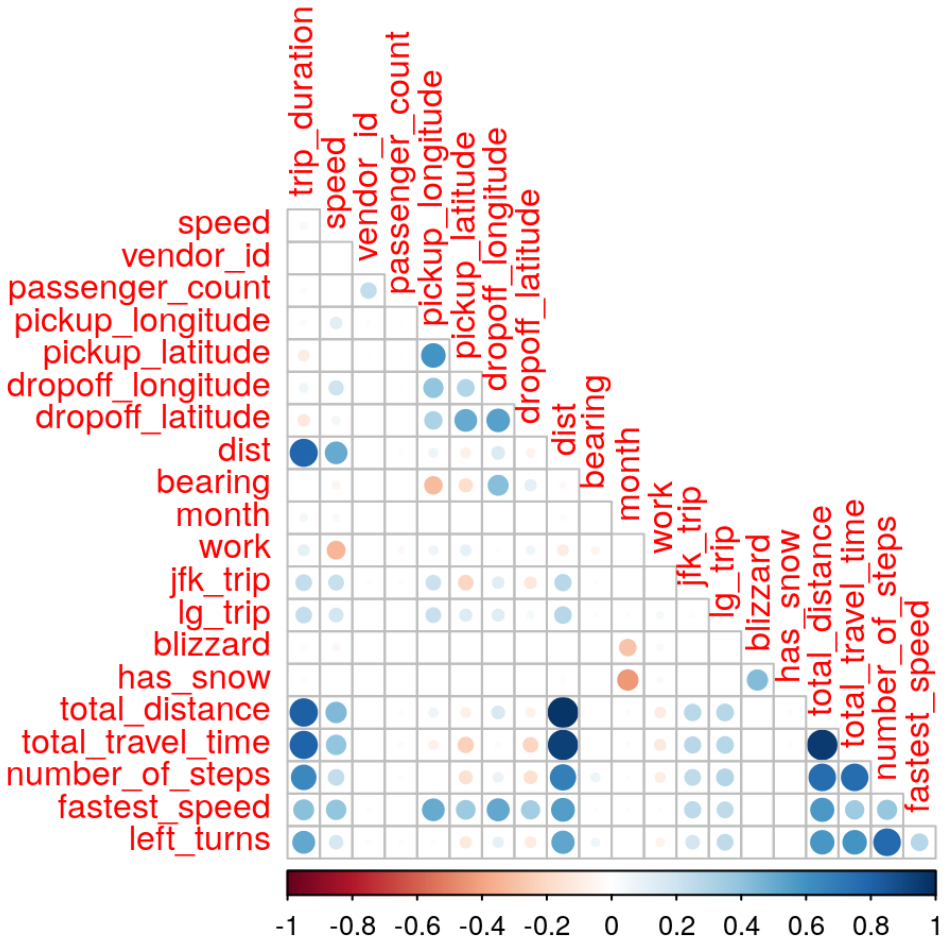

Fig. 30

Hide

```
foo <- 1
```

We find:

- The strongest correlations with the *trip_duration* are seen for the direct *dist*ance as well as the *total_distance* and *total_travel_time* derived from the OSRM data. As we have already seen above, the distances are highly correlated with one another and with the *total_travel_time*. Also the number of turns, presumably mostly via the *number_of_steps*, are having an impact on the *trip_duration*.
- Another effect on the *trip_duration* can bee seen for our engineered features *jfk_trip* and *lg_trip*; indicating journeys to either airport. A similar statement is true for the average *speed* and airport travel.
- The pickup and dropoff coordinates are correlated, which is a bit puzzling but might be partly explained by the shape of Manhattan stretching from south-west to north-east. Another part of the explanation might be short trips within lower or upper Manhattan only.

- *vendor_id* is correlated with *passenger_count* because of vendor 2 having all the (five) trips with more than 6 passengers.
- Among the *weather* data (mostly seen in the full hidden plot), the minimum and maximum temperatures show a strong correlation with each other and the *month* of the year, which is intuitively understandable. Also the different ways of measuring precipitation are naturally correlated with each other and, in case of *snow*, with the *month*.

We should be able to see these relations in our model.

8 An excursion into classification

Our challenge is by nature a *regression problem:* we want to predict a continous variable, *trip_duration* from a set of features. In this kernel we will continue to treat it as a regression problem, but in this section I want to present a brief excursion into the possibility of turning it into a *classification problem*.

In a classification context our target variable can only take a (small) number of discrete values. Often, we are faced with a binary outcome, for instance “Survived vs Not Survived” in the popular Titanic challenge (https://www.kaggle.com/c/titanic). Here we want to examine which factors contribute to *shorter or longer trip_durations.*

Recall the distribution of *trip_durations* from the beginning of this notebook, now cleaned with the more spurious trips removed. Here we separate it into three parts: short, middle, and long duration:

Hide

```
train_group <- train %>%
  mutate(tgroup = case_when(trip_duration < 3e2 ~ "fast",
                            trip_duration >= 3e2 & trip_duration <= 1.6e3 ~ "m
id",
                            trip_duration > 1.6e3 ~ "slow"))

train_group %>%
  ggplot(aes(trip_duration, fill = tgroup)) +
  geom_histogram(bins = 300) +
  scale_x_log10() +
  scale_y_sqrt()
```

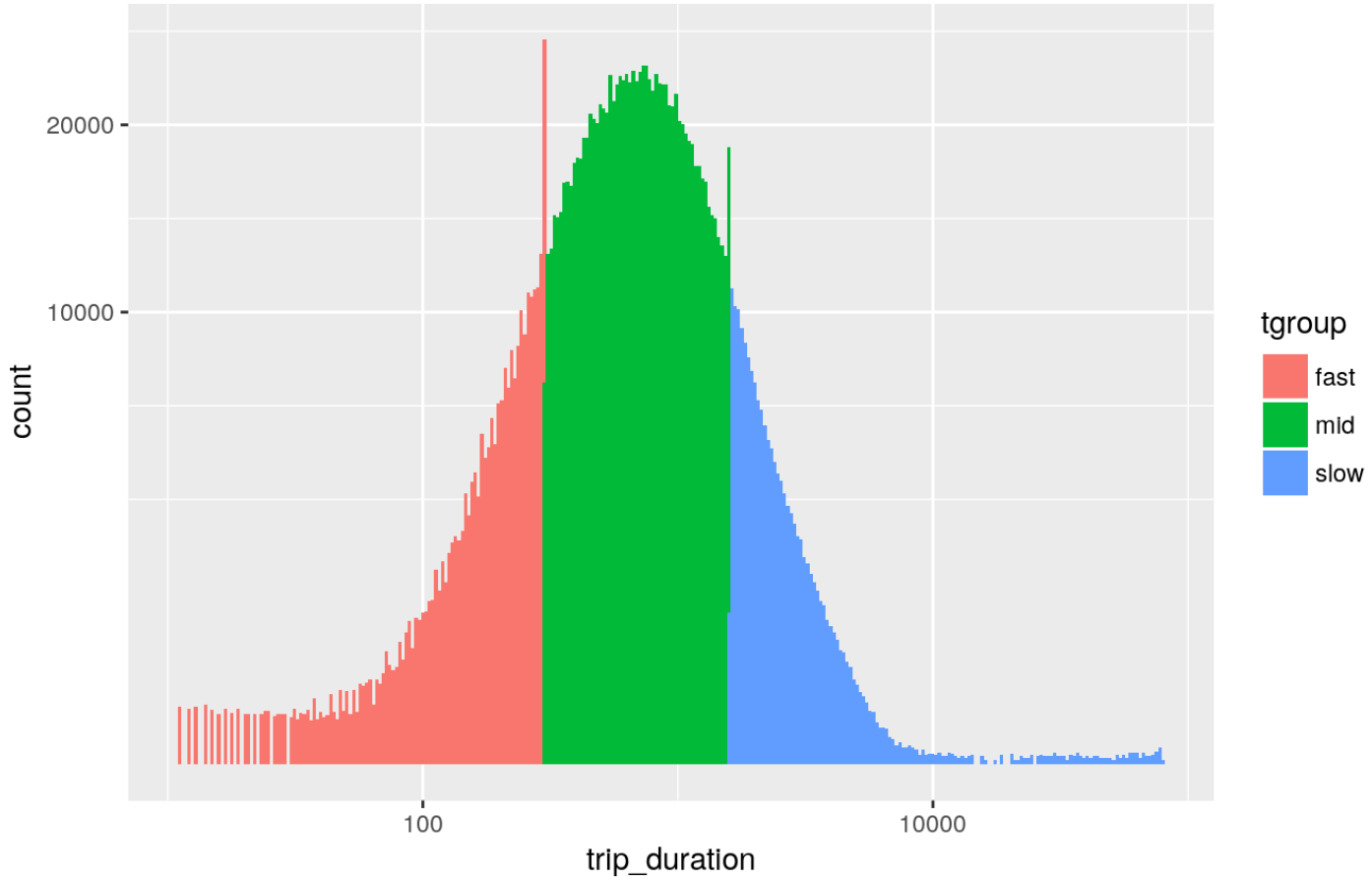

Fig. 31

(Ignore the apparent spikes caused by the binning resolution.)

For the purpose of this excercise, we are not interested in the middle bit. Our aim is to compare the properties of the fast vs slow trips.

Hide

```
train_group <- train_group %>%
  filter(tgroup != "mid")

p1 <- train_group %>%
  ggplot(aes(wday, fill = tgroup)) +
  geom_bar(position = "fill") +
  theme(legend.position = "none")

p2 <- train_group %>%
  ggplot(aes(month, fill = tgroup)) +
  geom_bar(position = "fill") +
  theme(legend.position = "none")

p3 <- train_group %>%
  ggplot(aes(hour, fill = tgroup)) +
  geom_bar(position = "fill")

p4 <- train_group %>%
  ggplot(aes(jfk_trip, fill = tgroup)) +
  geom_bar(position = "fill") +
  theme(legend.position = "none")

p5 <- train_group %>%
  ggplot(aes(lg_trip, fill = tgroup)) +
  geom_bar(position = "fill") +
  theme(legend.position = "none")

p6 <- train_group %>%
  ggplot(aes(blizzard, fill = tgroup)) +
  geom_bar(position = "fill") +
  theme(legend.position = "none")

p7 <- train_group %>%
  ggplot(aes(work, fill = tgroup)) +
  geom_bar(position = "fill") +
  theme(legend.position = "none")

layout <- matrix(c(1,1,2,2,3,3,3,3,4,5,6,7),3,4,byrow=TRUE)
multiplot(p1, p2, p3, p4, p5, p6, p7, layout=layout)
```

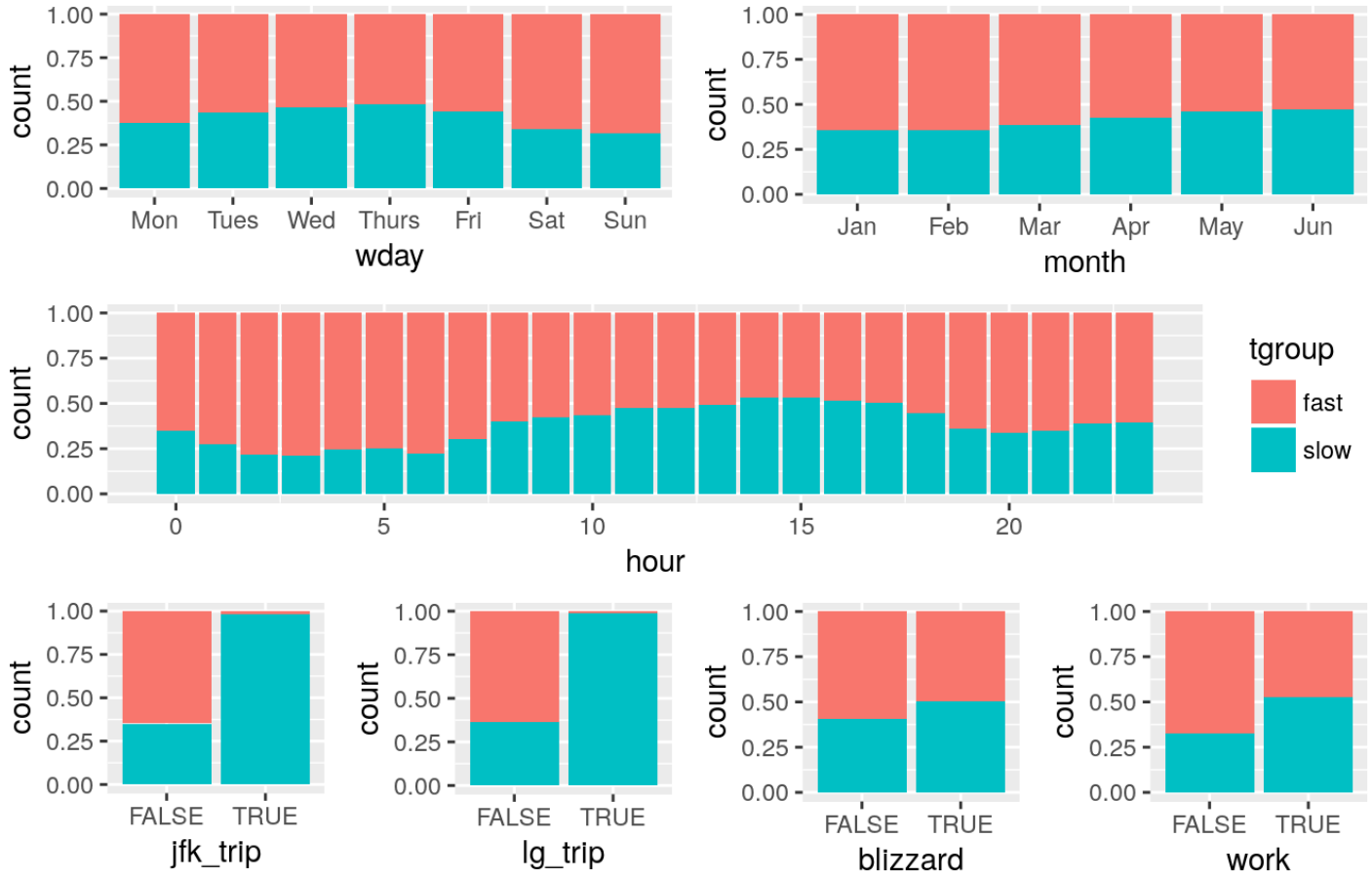

Fig. 32

Hide

```
p1 <- 1; p2 <- 1; p3 <- 1; p4 <- 1; p5 <- 1; p6 <- 1; p7 <- 1
```

We find:

- Our airport features, *jfk_trip* and *lg_trip*, behave as expected and are predominantly slow trips. Also the *blizzard* feature shows a promising increase in slow trips.
- Throughout the week, there is a notable difference between the weekend and the weekdays. It might be worth to encode the *wday* feature according to this trend. We see that the *monthly* trend already shows a gradually increasing “slow” proportion.
- During the day, we find again the difference between night and (working) day that we had seen in previously in the scatter plot plus heatmap and which is partly resolved in our engineered *work* feature.

For a visualisation of the *pickup* coordinates for the *fast* vs *slow* rides we build another map using the great *leaflet* package. Those maps don’t require external API calls and run fully contained in a Kaggle kernel. We visualise our two groups of trips using a *heatmap* which we add through tools in the *leaflet.extras* package.

In this *leaflet* visualisation, you can toggle and overlay the *fast* vs *slow* rides independently, and also choose a dark vs light background map (in addition to a free zoom and pan). These layers are easy to define within a leaflet object; have a look at the code to see how concise the *addLayersControl* element is. Zoom in to see street-level resolution:

Hide

```
pick_fast <- train_group %>%
  filter(tgroup == "fast") %>%
  select(long = pickup_longitude, lat = pickup_latitude)

pick_slow <- train_group %>%
  filter(tgroup == "slow") %>%
  select(long = pickup_longitude, lat = pickup_latitude)

leaflet(pick_fast) %>%
  addProviderTiles(providers$CartoDB.DarkMatter, group = "Dark map") %>%
  addProviderTiles(providers$CartoDB.Positron, group = "Light map") %>%
  addScaleBar() %>%
  setView(-73.95, 40.74, zoom = 11) %>%
  addHeatmap(max = 140, gradient = "Reds", radius = 12,
             minOpacity = 0.15, blur = 15, group = "Fast trips") %>%
  addHeatmap(lng = pick_slow$long, lat = pick_slow$lat,
             max = 100, gradient = "Blues", radius = 12,
             minOpacity = 0.25, blur = 15, group = "Slow trips") %>%
  addLayersControl(
    baseGroups = c("Dark Map", "Light map"),
    overlayGroups = c("Fast trips", "Slow trips"),
    options = layersControlOptions(collapsed = FALSE)
  )
```

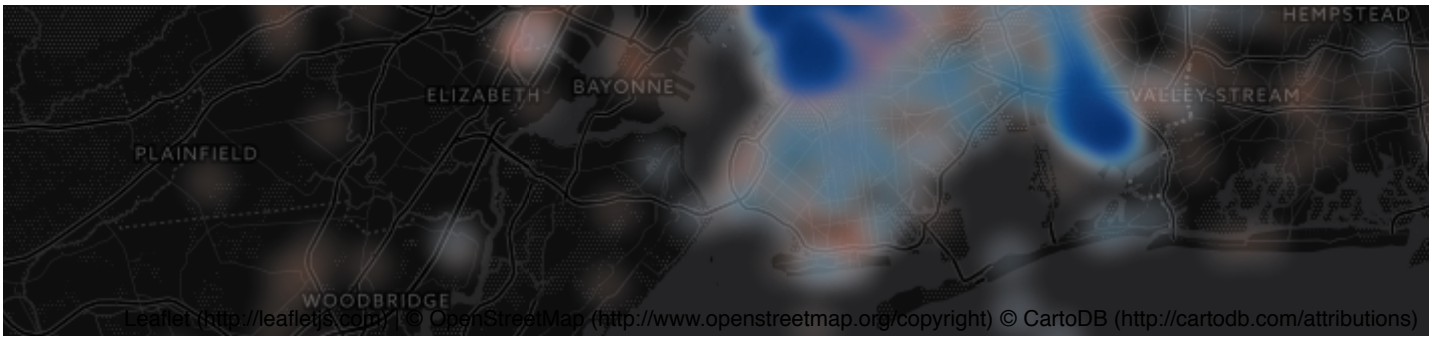

Fig. 33

We find:

- As expected, the airports stand out as origins of *slow* rides. Interestingly, a similar observations can be made for the areas surrounding the airports.
- There is a number of *fast* trip pickups just accross the Hudson river and near Newark airport in Jersey.
- Interestingly, except for the airport locations the *fast* trip pickups appear to be more spatially extended than the slow ones.

For a different kind of visualisation of your classification problem I would like to illustrate an *alluvial plot*. Made available through the *alluvial* package (), those plots are a kind of mix between a flow chart and a bar plot and they show how the target categories relate to various discrete features.

I would like to thank retrospectprospect (https://www.kaggle.com/retrospectprospect) for introducing me to alluvial plots in their very elegant EDA kernel (https://www.kaggle.com/retrospectprospect/titanic-machine-learning-from-eda-to-xgb) for the Titanic (https://www.kaggle.com/c/titanic) challenge! The same user also has shared an informative and concise kernel (https://www.kaggle.com/retrospectprospect/nyc-taxi-trip-duration-eda-to-xgb/) in this competition, and if you haven't seen it yet then I definitely recommend you to check it out.

Without further ado, here is the plot:

Hide

```
allu_train <- train_group %>%
  group_by(tgroup, work, wday, jfk_trip) %>%
  count() %>%
  ungroup

alluvial(allu_train %>% select(-n),
         freq=allu_train$n, border=NA,
         col=ifelse(allu_train$tgroup == "fast", "red", "blue"),
         cex=0.75,
         hide = allu_train$n < 150,
         ordering = list(
           order(allu_train$tgroup=="fast"),
           NULL,
           NULL,
           NULL))
```

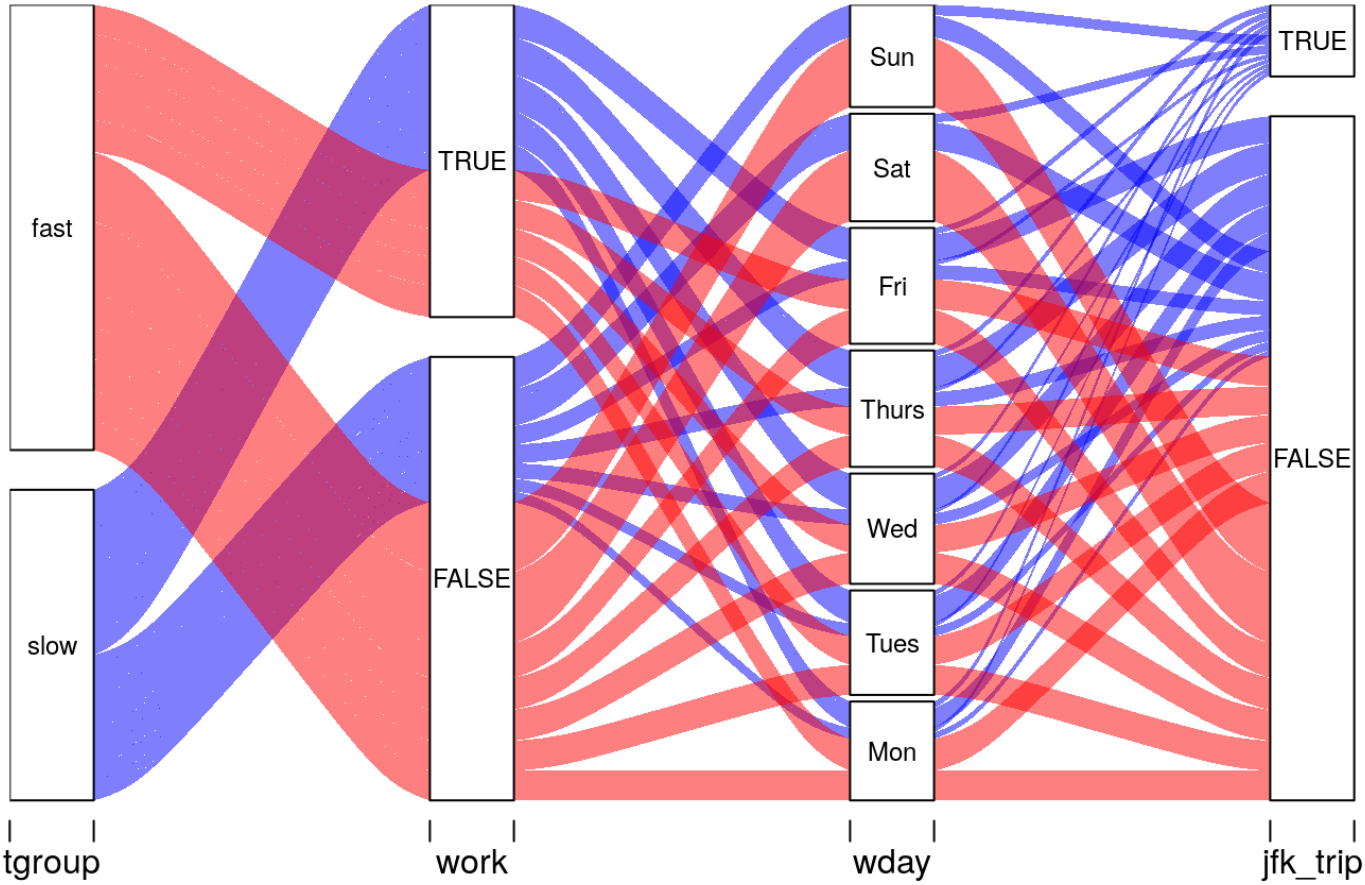

Fig. 34

In this plot, the vertical sizes of the blocks and the widths of the stripes (called “alluvia”) are proportional to the frequency. We decided to *hide* the alluvia with the lowest frequencies using the parameter of the same name. Within a category block you can re-*order* the vertical

layering of the alluvia to emphasise certain aspects of your data set. Here, we see nicely how the *fast* vs *slow* groups fan out into the different categories.

We find:

- Outside of work hours (*work == FALSE*) the majority of trips are *fast*, which we can see from the larger proportion of red alluvia.
- In terms of *wday*, Sat and Sun contain a larger proportion of *fast* trips than the weekdays. In addition, Sat and Sun have of course no contribution to *work == TRUE*, since no alluvia connect the corresponding boxes. This is a useful consistency check.
- Airport trips are almost entirely *slow* ones. We removed the few thin alluvias (using the *hide* parameter) because they were hard to see in the first place.

In summary: We find that looking at this problem from the point of view of classifying *fast* vs *slow* trips helps us to see certains effects clearer and to notice useful details about for instance the spatial distributions or the impact of work hours. Furthermore, we derive helpful hints on how to encode otherwise randomly ordered categorical variables like *wday* (day of the week) or *month* (randomly with respect to trip duration).

This concludes our brief excursion into classification. **I encourage any of you who are interested to take this classification approach and see how far you can take it.**

9 A simple model and prediction

In a brief final step we will feed our exploratory and engineering insights into a simple model to predict the target *trip_duration* for the *test* data set. However, this is primarily an EDA kernel and not a modelling one. This section leaves plenty of room for improvement for anyone who wants to optimise it. We will be using *XGBoost*, but you can try out any other algorithm.

I'm expecting that as time progresses in this competition the focus will shift towards the intricacies of model selection and tuning. I can think of a few more masters and grandmasters that I would like to see demonstrating their skills here, so that I can learn from them. For instance, this kernel won't touch on the subject of model stacking at all.

9.1 Preparations

9.1.1 *Train* vs *test* overlap

In order to make sure that we are really training on features that are relevant to our *test* data set we will now briefly compare the temporal and spatial properties of the *train* and *test* data. This is another consistency check. We could have done this before the exploration, but my

personal preference is to examine the training data first before looking at the *test* data set so that my analysis is as unbiased as possible. Here are the two relevant comparison plots:

Hide

```
foo <- combine %>%
  mutate(date = date(ymd_hms(pickup_datetime))) %>%
  group_by(date, dset) %>%
  count()
foo %>%
  ggplot(aes(date, n/1e3, color = dset)) +
  geom_line(size = 1.5) +
  labs(x = "", y = "Kilo trips per day")
```

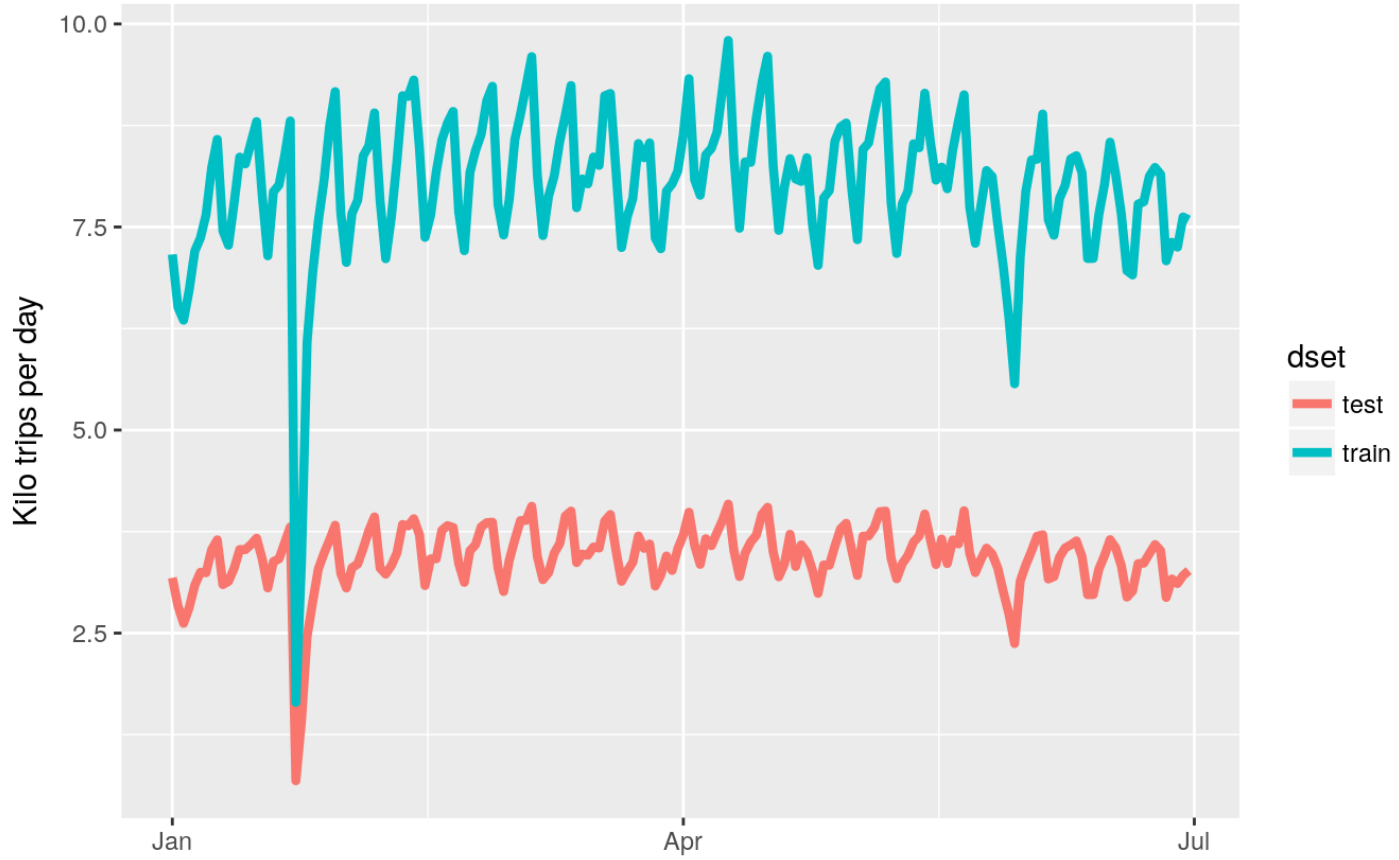

Fig. 35

Hide

```
pick_good <- combine %>%
  filter(pickup_longitude > -75 & pickup_longitude < -73) %>%
  filter(pickup_latitude > 40 & pickup_latitude < 42)
pick_good <- sample_n(pick_good, 5e3)

pick_good %>%
  ggplot(aes(pickup_longitude, pickup_latitude, color = dset)) +
  geom_point(size=0.1, alpha = 0.5) +
  coord_cartesian(xlim = c(-74.02,-73.77), ylim = c(40.63,40.84)) +
  facet_wrap(~ dset) +
  #guides(color = guide_legend(override.aes = list(alpha = 1, size = 4))) +
  theme(legend.position = "none")
```

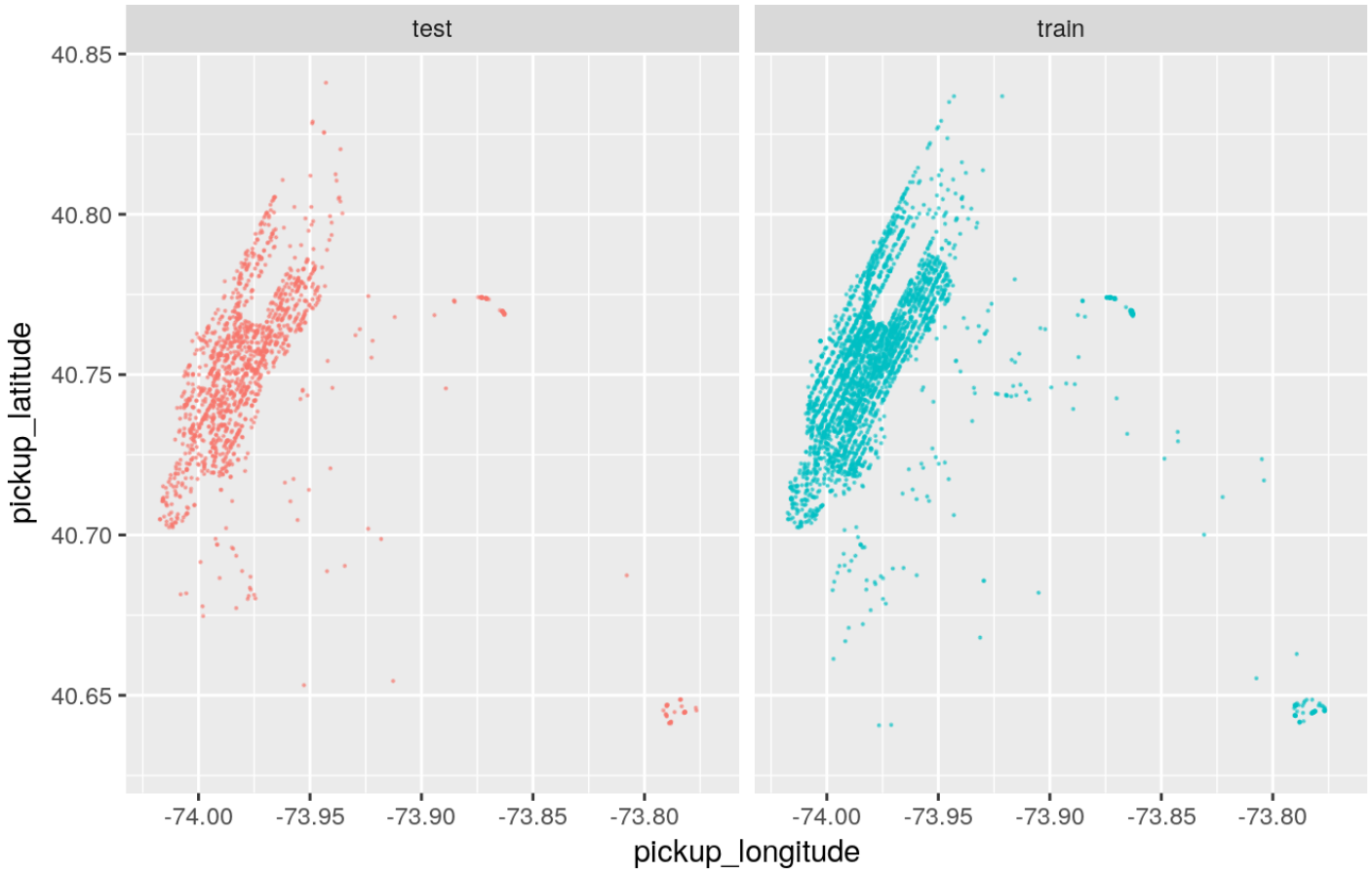

Fig. 36

We find that our *train* and *test* data sets do indeed cover the same time range and geographical area.

9.1.2 Data formatting

Here we will format the selected features to turn them into integer columns, since many classifiers cannot deal with categorical values. For the encoding we make use of our exploratory knowledge, including the insights gained in our classification excursion. This is necessary since most classifiers would assume a natural ordering for integer features (i.e. 1 < 2 < 3 with respect to impact on the target). An alternative would be to use one-hot encoding.

In order for the encoding to be homogeneous between the *train* and *test* data sets it is a useful habit to perform it on the combined data, in case there are differences in some of the factor levels. For instance, if the *passenger_count* were a genuine factor feature then here the *train* and *test* data would contain different levels (since *test* is missing 7 and 8 passenger trips).

This formatting stage is practically a repeat of what we did at the beginning of the *Feature Engineering* section. In principle we could have already worked with the *combine* data set back then, but for the purpose of keeping it simple we used only the *train* sample back then. You are very welcome to take this kernel and improve it for better efficiency.

Nothing magical here in our engineering/formatting code block:

Hide

```
# airport coordinates again, just to be sure
jfk_coord <- tibble(lon = -73.778889, lat = 40.639722)
la_guardia_coord <- tibble(lon = -73.872611, lat = 40.77725)

# derive distances
pick_coord <- combine %>%
  select(pickup_longitude, pickup_latitude)
drop_coord <- combine %>%
  select(dropoff_longitude, dropoff_latitude)
combine$dist <- distCosine(pick_coord, drop_coord)
combine$bearing = bearing(pick_coord, drop_coord)

combine$jfk_dist_pick <- distCosine(pick_coord, jfk_coord)
combine$jfk_dist_drop <- distCosine(drop_coord, jfk_coord)
combine$lg_dist_pick <- distCosine(pick_coord, la_guardia_coord)
combine$lg_dist_drop <- distCosine(drop_coord, la_guardia_coord)

# add dates
combine <- combine %>%
  mutate(pickup_datetime = ymd_hms(pickup_datetime),
         dropoff_datetime = ymd_hms(dropoff_datetime),
         date = date(pickup_datetime)
  )

# add weather info
foo <- weather %>%
  select(date, rain, s_fall, all_precip, has_snow, has_rain, s_depth, max_tem
p, min_temp)
combine <- left_join(combine, foo, by = "date")

# add fast routes
foo <- fastest_route %>%
  select(id, total_distance, total_travel_time, number_of_steps,
         step_direction, step_maneuvers) %>%
  mutate(fastest_speed = total_distance/total_travel_time*3.6,
         left_turns = str_count(step_direction, "left"),
         right_turns = str_count(step_direction, "right"),
         turns = str_count(step_maneuvers, "turn")
         ) %>%
  select(-step_direction, -step_maneuvers)
```

```
combine <- left_join(combine, foo, by = "id")

# reformat to numerical and recode levels
combine <- combine %>%
  mutate(store_and_fwd_flag = as.integer(factor(store_and_fwd_flag)),
         vendor_id = as.integer(vendor_id),
         month = as.integer(month(pickup_datetime)),
         wday = wday(pickup_datetime, label = TRUE),
         wday = as.integer(fct_relevel(wday, c("Sun", "Sat", "Mon", "Tues", "W
ed", "Thurs", "Fri"))),
         hour = hour(pickup_datetime),
         work = as.integer( (hour %in% seq(8,18)) & (wday %in%
c("Mon","Tues","Fri","Wed","Thurs")) ),
         jfk_trip = as.integer( (jfk_dist_pick < 2e3) | (jfk_dist_drop < 2e3)
 ),
         lg_trip = as.integer( (lg_dist_pick < 2e3) | (lg_dist_drop < 2e3) ),
         has_rain = as.integer(has_rain),
         has_snow = as.integer(has_snow),
         blizzard = as.integer( !( (date < ymd("2016-01-22") | (date > ymd("20
16-01-29")))) )
         )
```

You might have noticed that we now recode the weekdays (*wday*) according to the results from the classification section. Conveniently, the *months* are already in a sequence of increasing fraction of slow trips. (Anybody got an idea why this could be?)

Consistency check:

Hide

```
glimpse(combine)
```

```
## Observations: 2,083,778
## Variables: 41
## $ id                 <chr> "id2875421", "id2377394", "id3858529", "id3...
## $ vendor_id          <int> 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2...
## $ pickup_datetime    <dttm> 2016-03-14 17:24:55, 2016-06-12 00:43:35, ...
## $ dropoff_datetime   <dttm> 2016-03-14 17:32:30, 2016-06-12 00:54:38, ...
## $ passenger_count    <int> 1, 1, 1, 1, 1, 6, 4, 1, 1, 1, 1, 4, 2, 1, 1...
## $ pickup_longitude   <dbl> -73.98215, -73.98042, -73.97903, -74.01004,...
## $ pickup_latitude    <dbl> 40.76794, 40.73856, 40.76394, 40.71997, 40....
## $ dropoff_longitude  <dbl> -73.96463, -73.99948, -74.00533, -74.01227,...
## $ dropoff_latitude   <dbl> 40.76560, 40.73115, 40.71009, 40.70672, 40....
## $ store_and_fwd_flag <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ trip_duration      <int> 455, 663, 2124, 429, 435, 443, 341, 1551, 2...
## $ dset               <fctr> train, train, train, train, train, train, ...
## $ dist               <dbl> 1500.1995, 1807.5298, 6392.2513, 1487.1625,...
## $ bearing            <dbl> 99.932546, -117.063997, -159.608029, -172.7...
## $ jfk_dist_pick      <dbl> 22315.02, 20258.98, 21828.54, 21461.51, 236...
## $ jfk_dist_drop      <dbl> 21025.4008, 21220.9775, 20660.3899, 21068.1...
## $ lg_dist_pick       <dbl> 9292.897, 10058.778, 9092.997, 13228.107, 8...
## $ lg_dist_drop       <dbl> 7865.248, 11865.578, 13461.012, 14155.920, ...
## $ date               <date> 2016-03-14, 2016-06-12, 2016-01-19, 2016-0...
## $ rain               <dbl> 0.29, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ s_fall             <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ all_precip         <dbl> 0.29, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0...
## $ has_snow           <int> 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ has_rain           <int> 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0...
## $ s_depth            <dbl> 0.00, 0.00, 0.01, 0.00, 0.00, 6.00, 0.00, 0...
## $ max_temp           <int> 51, 83, 28, 48, 55, 39, 78, 66, 87, 79, 63,...
## $ min_temp           <int> 40, 62, 16, 30, 38, 28, 63, 54, 73, 63, 50,...
## $ total_distance     <dbl> 2009.1, 2513.2, 11060.8, 1779.4, 1614.9, 13...
## $ total_travel_time  <dbl> 164.9, 332.0, 767.6, 235.8, 140.1, 189.4, 1...
## $ number_of_steps    <int> 5, 6, 16, 4, 5, 5, 5, 17, 2, 13, 6, 9, 4, 4...
## $ fastest_speed      <dbl> 43.86149, 27.25157, 51.87452, 27.16641, 41....
## $ left_turns         <int> 1, 2, 5, 2, 2, 1, 1, 4, 0, 7, 2, 6, 1, 1, 4...
## $ right_turns        <int> 1, 2, 7, 1, 2, 3, 3, 9, 1, 5, 2, 2, 1, 1, 3...
## $ turns              <int> 1, 2, 9, 1, 3, 3, 2, 6, 0, 9, 3, 5, 2, 2, 5...
## $ month              <int> 3, 6, 1, 4, 3, 1, 6, 5, 5, 3, 5, 5, 2, 6, 5...
## $ wday               <int> 3, 1, 4, 5, 2, 2, 7, 2, 7, 6, 4, 1, 7, 5, 7...
## $ hour               <int> 17, 0, 11, 19, 13, 22, 22, 7, 23, 21, 22, 1...
## $ work               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

```
## $ jfk_trip           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ lg_trip            <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ blizzard           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

Looks good. The only non-numerical features are *id*, *pickup_datetime*, *dropoff_datetime*, and *date*, which will remove in any case, together with *dset* which we will use now to separate the *train* vs *test* again.

9.1.3 Feature selection, metric adjustment, validation split, and careful cleaning

Not all features in our data set will be useful. Here we only include meaningful variables and remove for instance the *id* feature and most of the weather features.

In principle, we could include all features and leave the selection of the useful ones to our modelling algorithm. However, in our case a pre-selection could be useful because we have (a) engineered a couple of features from existing ones (such as *work* or *blizzard*), and (b) imported additional features with strong correlation (such as *total_distance* and *total_travel_time*). Both strategies can cause significant *collinearity* within our training feature set, which will make it more difficult to interpret the result of our model in terms of the impact of individual features. Furthermore, from a pragmatic point of view any strongly correlated features don't add much new information and should (in my opinion) be removed for reasons of statistical parsimony.

Feature selection is normally an iterative process where we run an initial model with either few or many features and then step-by-step add or remove some features based on the results of the previous run. For the sake of kernel run time here we only include one model fitting step. You are very welcome to use it as a starting point for your own analysis and combine it with insights from other kernels. We will base our feature selection on the results of the exploratory analysis, in particular the final correlation matrix.

In this code block, the feature selection is structured with a little more detail than necessary, in order to make it easier to generalise it to a new problem:

Hide

```
# Specific definitions:
#---------------------------------
# predictor features
train_cols <- c("total_travel_time", "total_distance", "hour", "dist",
                "vendor_id", "jfk_trip", "lg_trip", "wday", "month",
                "pickup_longitude", "pickup_latitude", "bearing", "lg_dist_dro
p")
# target feature
y_col <- c("trip_duration")
# identification feature
id_col <- c("id")
# auxilliary features
aux_cols <- c("dset")
# cleaning features
clean_cols <- c("jfk_dist_drop", "jfk_dist_pick")
#---------------------------------

# General extraction
#---------------------------------
# extract test id column
test_id <- combine %>%
  filter(dset == "test") %>%
  select_(.dots = id_col)

# all relevant columns for train/test
cols <- c(train_cols, y_col, aux_cols, clean_cols)
combine <- combine %>%
  select_(.dots = cols)

# split train/test
train <- combine %>%
  filter(dset == "train") %>%
  select_(.dots = str_c("-",c(aux_cols)))
test <- combine %>%
  filter(dset == "test") %>%
  select_(.dots = str_c("-",c(aux_cols, clean_cols, y_col)))
#---------------------------------
```

For our taxi challenge, the evaluation metric is RMSLE (https://www.kaggle.com/c/nyc-taxi-trip-duration#evaluation), the Root Mean Squared Logarithmic Error (https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError). Essentially, this means that we are optimising the prediction vs data deviations in log space. This has the advantage that large individual outliers don't get as much weight as they would in a linear metric.

In order to easily simulate the evaluation metric in our model fitting we replace the *trip_duration* with its logarithm. (The `+ 1` is added to avoid an undefined `log(0)` and we need to remember to remove this 1 second for the prediction file, although it shouldn't make a huge difference if we didn't.)

Hide

```
train <- train %>%
  mutate(trip_duration = log(trip_duration + 1))
```

In order to assess how well our model generalises we will perform a cross-validation step and also split our training data into a *train* vs *validation* data set. Thereby, the model performance can be evaluated on a sample that the algorithm has not seen. We split our data into 80/20 fractions using a tool from the caret package (https://cran.r-project.org/web/packages/caret/index.html). We wouldn't really need this package here for such a simple task, but I'm including it to give it a mention. *Caret* is a multi-purpose ML package that is certainly worth checking out.

Hide

```
set.seed(4321)
trainIndex <- createDataPartition(train$trip_duration, p = 0.8, list = FALSE,
times = 1)

train <- train[trainIndex,]
valid <- train[-trainIndex,]
```

Finally, we are performing careful cleaning steps on our *train* sample only. By removing observations from the training data we always run the risk of overfitting our model to a more "idealised" version of reality. In particular for the current data set, where lots of values don't make obvious sense, this idealised model might not generalise well to unseen data, which could have similar "quirks". For this reason, we are keeping the validation sample unchanged, so that any noticeable deviations between *train* and *valid* model score can alert us to overfitting.

Why clean at all? Well, in my opinion obvious outliers should be removed to make the model robust. It's true that these outliers might also exist in the *test* data, but if there is a way to predict them then this suggests that they might not be outliers after all. And if we can't

predict them there's no way to take them into account anyway. Lastly, the answer also depends on the *goal* of your analysis (and your evaluation metric) which might not always aim at fitting a given *test* set as good as possible but sometimes have a more pragmatic target that allows you to ignore extreme values.

Here we only remove the few *trip_duration* that are longer than a day and the couple of data points far, far away from NYC:

Hide

```
valid <- valid %>%
  select_(.dots = str_c("-",c(clean_cols)))

train <- train %>%
  filter(trip_duration < 24*3600,
         jfk_dist_pick < 3e5 & jfk_dist_drop < 3e5
         ) %>%
  select_(.dots = str_c("-",c(clean_cols)))
```

9.2 XGBoost parameters and fitting

For the model fitting we will use everybody's favourite *XGBoost - eXtreme Gradient Boosting*. This is a decision-tree gradient boosting algorithm with a high degree of popularity here on Kaggle. In this kernel we will implement a fairly straight-forward fitting strategy. Feel free to optimise the parameters and get some inspiration from other kernels (https://www.kaggle.com/gaborfodor/from-eda-to-the-top-lb-0-367/) that are more focussed on model fitting. Also try out other classifiers, such as Megan's baseline Random Forest (https://www.kaggle.com/mrisdal/last-place-laura-benchmark/) to see how they treat our features.

A few brief words about *gradient boosting*, as far as I understand it: Boosting is what we call the step-by-step improvement of a weak learner (like a relatively shallow decision tree of *max_depth* levels) by successively applying it to the results of the previous learning step (for *nrounds* times in total). *Gradient Boosting* focusses on minimising the Loss Function (according to our evaluation metric) by training the algorithm on the gradient of this function. The method of *Gradient Decent* iteratively moves into the direction of the greatest decent (i.e. most negative first derivative) of the loss function. The step sizes can vary from iteration to iteration but has a multiplicative *shrinkage factor eta in (0,1]* associated with it for additional tuning. Smaller values of eta result in a slower decent and require higher *nrounds*.

In order for *XGBoost* to properly ingest our data samples we need to re-format them slightly:

Hide

```
#convert to XGB matrix
foo <- train %>% select(-trip_duration)
bar <- valid %>% select(-trip_duration)

dtrain <- xgb.DMatrix(as.matrix(foo),label = train$trip_duration)
dvalid <- xgb.DMatrix(as.matrix(bar),label = valid$trip_duration)
dtest <- xgb.DMatrix(as.matrix(test))
```

Now we define the meta-parameters that govern how *XGBoost* operates. See here (https://github.com/dmlc/xgboost/blob/master/doc/parameter.md) for more details. The *watchlist* parameter tells the algorithm to keep an eye on both the *training* and *validation* sample metrics. Those parameters are not optimised for performance, so feel free to play with them:

Hide

```
xgb_params <- list(colsample_bytree = 0.7, #variables per tree
                   subsample = 0.7, #data subset per tree
                   booster = "gbtree",
                   max_depth = 5, #tree levels
                   eta = 0.3, #shrinkage
                   eval_metric = "rmse",
                   objective = "reg:linear",
                   seed = 4321
                   )

watchlist <- list(train=dtrain, valid=dvalid)
```

And here we *train* our classifier, i.e. fit it to the *training* data. To ensure reproducability we set an R seed here. To make this model run in the kernel environment, given our extensive EDA run time, we will restrict the learning to 60 sample rounds.

Hide

```
set.seed(4321)
gb_dt <- xgb.train(params = xgb_params,
                   data = dtrain,
                   print_every_n = 5,
                   watchlist = watchlist,
                   nrounds = 60)
```

```
## [1]  train-rmse:4.227815 valid-rmse:4.226777
## [6]  train-rmse:0.836443 valid-rmse:0.835214
## [11] train-rmse:0.445397 valid-rmse:0.443782
## [16] train-rmse:0.420891 valid-rmse:0.419287
## [21] train-rmse:0.412767 valid-rmse:0.411220
## [26] train-rmse:0.410120 valid-rmse:0.408587
## [31] train-rmse:0.407339 valid-rmse:0.405596
## [36] train-rmse:0.405224 valid-rmse:0.403590
## [41] train-rmse:0.403177 valid-rmse:0.401740
## [46] train-rmse:0.401562 valid-rmse:0.400126
## [51] train-rmse:0.400236 valid-rmse:0.398776
## [56] train-rmse:0.399145 valid-rmse:0.397616
## [60] train-rmse:0.398196 valid-rmse:0.396632
```

After the fitting we are running a 5-fold cross-validation (CV) to estimate our model's performance. Also this stage would exceed the Kaggle run-time limit for a larger number of rounds, therefore I'm limiting it here to 15 sample rounds to demonstrate the principle. You should use at least a few 100 in your analysis, depending on your XGBoost parameters. The early-stopping parameter will make sure that the CV fitting is stopped once the model can't be improved through additional steps.

Hide

```
xgb_cv <- xgb.cv(xgb_params,dtrain,early_stopping_rounds = 10, nfold = 5, nrou
nds=15)
```

```
## [1]  train-rmse:4.227837+0.000287    test-rmse:4.227776+0.000829
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 10 rounds.
##
## [2]  train-rmse:2.978744+0.000522    test-rmse:2.978735+0.001204
## [3]  train-rmse:2.110670+0.000484    test-rmse:2.110653+0.001094
## [4]  train-rmse:1.514938+0.003098    test-rmse:1.515065+0.003327
## [5]  train-rmse:1.107978+0.002013    test-rmse:1.108181+0.002499
## [6]  train-rmse:0.839584+0.001026    test-rmse:0.839878+0.002332
## [7]  train-rmse:0.666502+0.001613    test-rmse:0.666953+0.002008
## [8]  train-rmse:0.560638+0.000965    test-rmse:0.561226+0.001965
## [9]  train-rmse:0.499904+0.001153    test-rmse:0.500613+0.001727
## [10] train-rmse:0.466107+0.001236    test-rmse:0.467003+0.001629
## [11] train-rmse:0.447140+0.000624    test-rmse:0.448114+0.002100
## [12] train-rmse:0.436292+0.000844    test-rmse:0.437406+0.002450
## [13] train-rmse:0.429372+0.000389    test-rmse:0.430577+0.002594
## [14] train-rmse:0.425331+0.000575    test-rmse:0.426619+0.002645
## [15] train-rmse:0.422833+0.000681    test-rmse:0.424229+0.002608
```

9.3 Feature importance

After training we will check which features are the most important for our model. This can provide the starting point for an iterative process where we identify, step by step, the significant features for a model. Here we will simply visualise these features:

Hide

```
imp_matrix <- as.tibble(xgb.importance(feature_names = colnames(train %>% sele
ct(-trip_duration)), model = gb_dt))

imp_matrix %>%
  ggplot(aes(reorder(Feature, Gain, FUN = max), Gain, fill = Feature)) +
  geom_col() +
  coord_flip() +
  theme(legend.position = "none") +
  labs(x = "Features", y = "Importance")
```

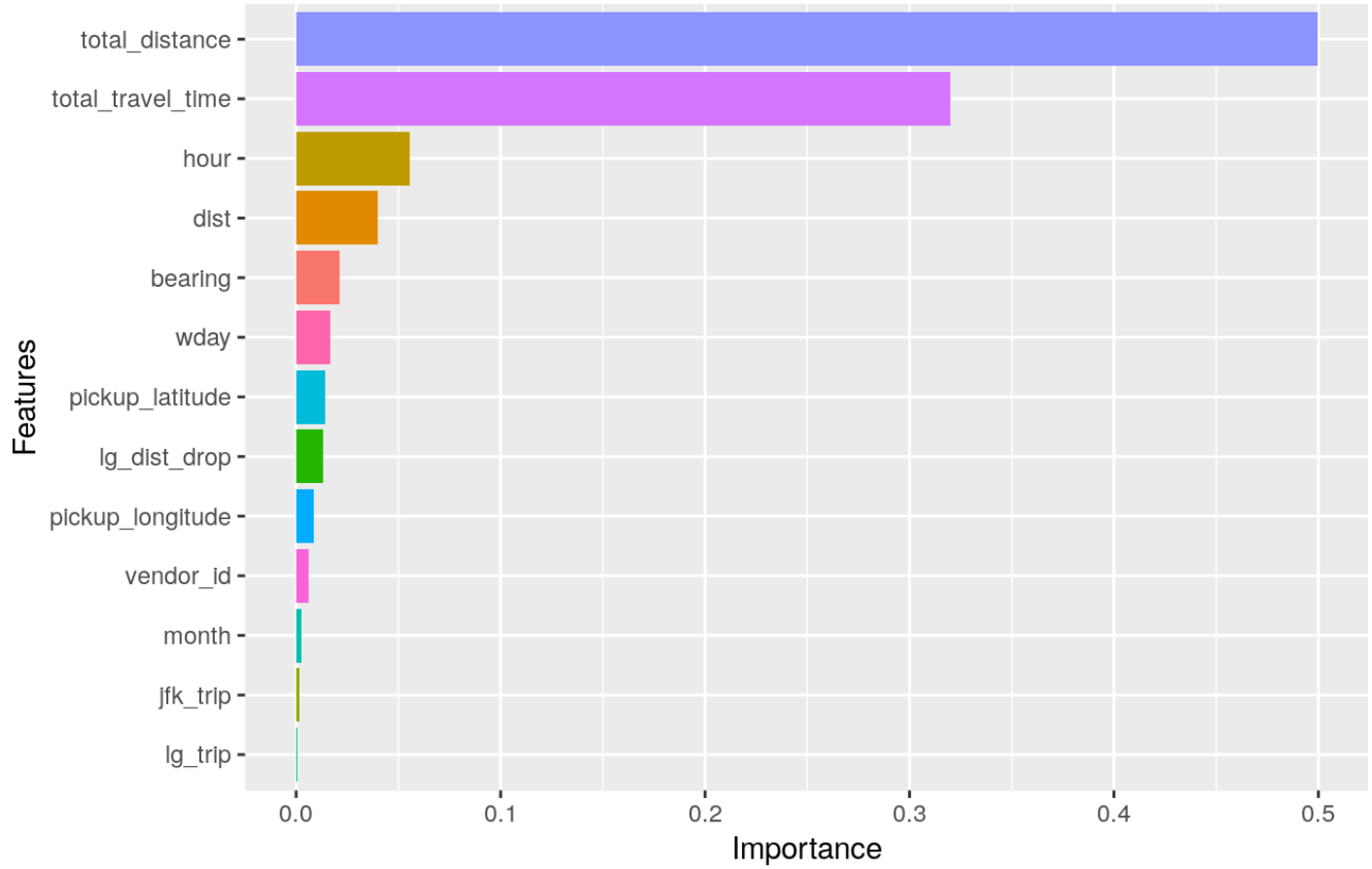

Fig. 34

We find that, unsurprisingly, the OSRM features like *total_distance* and *total_travel_time* are by far the most important predictors for the *trip_duration*. Beyond that, our engineered features like *wday* or *bearing* are not doing bad either.

9.4 Prediction and submission file

Once we are reasonably happy with our CV score we use the corresponding model to make a prediction for the test data, write the submission file, and submit it to the Kaggle leaderboard. This gives a performance score based on an *independent* data set. For this very reason, it is advisable to aim to keep the number of leaderboard submission low. Otherwise you run the risk to have your model influenced too much by this public leaderboard data which will result in overfitting. It's better to trust your local CV. Just ask anyone who has just completed the Mercedes challenge ;-)

Practically, this pre-ulimate code block will apply the *XGBoost* model to the testing data and write a submission file according to the competition specification. You will then find the file `submit.csv` in the "Output" tab of the Kaggle kernel or, if you have run the script locally, in the source directory.

Hide

```
test_preds <- predict(gb_dt,dtest)
pred <- test_id %>%
  mutate(trip_duration = exp(test_preds) - 1)

pred %>% write_csv('submit.csv')
```

All that's left to do is to double-check this submission file to make sure that we are not wasting a (potentially) valuable submission with a mal-formatted output file. We are comparing its shape and content with the *sample submission file* that we read in at the very beginning, and also plot the distribution of the predicted values compared to the training values for an approximate comparison:

Hide

```
identical(dim(sample_submit),dim(pred))
```

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

Hide

```
glimpse(sample_submit)
```

```
## Observations: 625,134
## Variables: 2
## $ id            <chr> "id3004672", "id3505355", "id1217141", "id215012...
## $ trip_duration <int> 959, 959, 959, 959, 959, 959, 959, 959, 959, 959...
```

Hide

```
glimpse(pred)
```

```
## Observations: 625,134
## Variables: 2
## $ id            <chr> "id3004672", "id3505355", "id1217141", "id215012...
## $ trip_duration <dbl> 809.41115, 408.47892, 369.20089, 987.35398, 305....
```

Hide

```
bind_cols(sample_submit, pred) %>% head(5)
```

```
## # A tibble: 5 x 4
##          id trip_duration       id1 trip_duration1
##       <chr>         <int>     <chr>          <dbl>
## 1 id3004672           959 id3004672       809.4112
## 2 id3505355           959 id3505355       408.4789
## 3 id1217141           959 id1217141       369.2009
## 4 id2150126           959 id2150126       987.3540
## 5 id1598245           959 id1598245       305.8023
```

Hide

```
foo <- train %>%
  select(trip_duration) %>%
  mutate(dset = "train",
         trip_duration = exp(trip_duration) - 1)
bar <- pred %>%
  mutate(dset = "predict")

bind_rows(foo, bar) %>%
  ggplot(aes(trip_duration, fill = dset)) +
  geom_density(alpha = 0.5) +
  scale_x_log10()
```

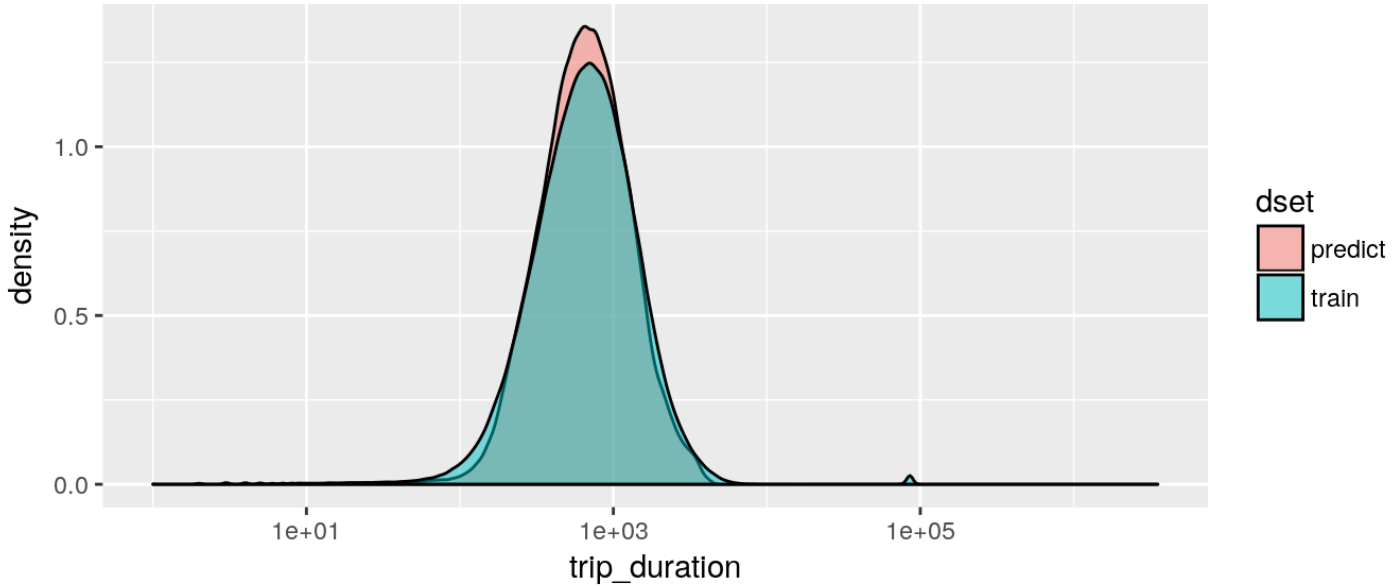

Fig. 35

Prediction file format and *trip_duration* range successfully verified.

In its present state, this model will give you a score of 0.410 on the public leaderboard which, at the time of writing, will put you in the top 40% of submissions. Clearly this is just a starting point for you to explore and improve the modelling process. Here are a few suggestions:

- First of all, run the training and cross-validation for a larger number of rounds. This alone will immediately improve your score.
- Add further features to the model and explore the impact of more or less cleaning.
- Try out different XGBoost parameters and see how they affect the CV score.

The rest is up to you. I hope that this extensive EDA gave you useful insights into the data as well as ideas to get you started with your own analysis. I wish you the best of success.

Many thanks for reading this kernel! Have fun!