Big Data Analytics

Lecture 6: Aggregation Visualization I

Prof. Dr. Ulrich Matter
(University of St. Gallen)

01/04/2021

1 Data

In this tutorial we explore the world of New York's famous Yellow Caps. The NYC Taxi & Limousine Commission (TLC) provides detailed data on all trip records including pick-up and drop-off times/locations. When combining all available trip records (2009-2018), we get a rather large data set of over 200GB. The code examples below illustrate how to collect and compile the entire data set. In order to avoid long computing times, the code examples shown below are based on a small sub-set of the actual raw data (however, all examples involving virtual memory, are in theory scalable to the extent of the entire raw data set).

1.1 Gathering and Compilation of all the raw data

The raw data consists of several monthly CSV-files and can be downloaded via the TLC's website. The following short R-script automates the downloading of all available trip-record files. *NOTE*: Downloading all files can take several hours and will occupy over 200GB!

```
##################################
# Fetch all TLC trip recrods
# Data source:
# https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
# Input: Monthly csv files from urls
# Output: one large csv file
# UM, St. Gallen, January 2019
##################################

# SET UP -----------------

# load packages
library(data.table)
library(rvest)
library(httr)

# fix vars
BASE_URL <- "https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2018-01.csv"
OUTPUT_PATH <- "../data/tlc_trips.csv"
START_DATE <- as.Date("2009-01-01")
END_DATE <- as.Date("2018-06-01")


# BUILD URLS -----------
```

```
# parse base url
base_url <- gsub("2018-01.csv", "", BASE_URL)
# build urls
dates <- seq(from= START_DATE,
                  to = END_DATE,
                  by = "month")
year_months <- gsub("-01$", "", as.character(dates))
data_urls <- paste0(base_url, year_months, ".csv")

# FETCH AND STACK CSVS ----------------

# download, parse all files, write them to one csv
for (url in data_urls) {

     # download to temporary file
     tmpfile <- tempfile()
     download.file(url, destfile = tmpfile)

     # parse downloaded file, write to output csv, remove tempfile
     csv_parsed <- fread(tmpfile)
     fwrite(csv_parsed,
            file =  OUTPUT_PATH,
            append = TRUE)
     unlink(tmpfile)

}
```

2 Data aggregation with chunked data files

In this first part of the data aggregation tutorial, we will focus on the `ff`-based approach to employ parts of the hard disk as 'virtual memory'. This means, all of the examples are easily scalable without risking too much memory pressure. Given the size of the entire TLC database (over 200GB), we will only use one million taxi trips records of January 2009.[1]

2.1 Data aggregation: The 'split-apply-combine' strategy

The 'split-apply-combine' strategy plays an important role in many data analysis tasks, ranging from data preparation to summary statistics and model-fitting.[2] The strategy can be defined as "break up a problem into manageable pieces, operate on each piece independently and then put all the pieces back together." (Wickham 2011, 1)

Many R users are familiar with the basic concept of split-apply-combine implemented in the `plyr`-package intended for the usual in-memory operations (data set fits into RAM). Here, we explore the options for split-apply-combine approaches to large data sets that do not fit into RAM.

2.1.1 Data import and cleaning

First, we read the raw taxi trips records into R with the `ff`-package.

[1] Note that the code examples below could also be run based on the entire TLC database (provided that there is enough hard-disk space available). But, creating the `ff` chunked file structure for a 200GB CSV would take hours or even days.

[2] Moreover, 'split-apply-combine' is closely related to a core strategy of Big Data analytics with distributed systems: Map/Reduce - more on this in the lecture on distributed systems.

```
# load packages
library(ff)
library(ffbase)

# set up the ff directory (for data file chunks)
if (!dir.exists("fftaxi")){
     system("mkdir fftaxi")
}
options(fftempdir = "fftaxi")

# import a few lines of the data, setting the column classes explicitly
col_classes <- c(V1 = "factor",
                 V2 = "POSIXct",
                 V3 = "POSIXct",
                 V4 = "integer",
                 V5 = "numeric",
                 V6 = "numeric",
                 V7 = "numeric",
                 V8 = "numeric",
                 V9 = "numeric",
                 V10 = "numeric",
                 V11 = "numeric",
                 V12 = "factor",
                 V13 = "numeric",
                 V14 = "numeric",
                 V15 = "factor",
                 V16 = "numeric",
                 V17 = "numeric",
                 V18 = "numeric")

# import the first one million observations
taxi <- read.table.ffdf(file = "../data/tlc_trips.csv",
                        sep = ",",
                        header = TRUE,
                        next.rows = 100000,
                        colClasses= col_classes,
                        nrows = 1000000
                        )
```

Following the data documentation provided by TLC, we give the columns of our data set more meaningful names and remove the empty columns (some covariates are only collected in later years).

```
# first, we remove the empty vars V8 and V9
taxi$V8 <- NULL
taxi$V9 <- NULL


# set covariate names according to the data dictionary
# see https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf
# note instead of taxizonne ids, long/lat are provided

varnames <- c("vendor_id",
              "pickup_time",
              "dropoff_time",
```

```
              "passenger_count",
              "trip_distance",
              "start_lat",
              "start_long",
              "dest_lat",
              "dest_long",
              "payment_type",
              "fare_amount",
              "extra",
              "mta_tax",
              "tip_amount",
              "tolls_amount",
              "total_amount")
names(taxi) <- varnames
```

When inspecting the factor variables of the data set, we notice that some of the values are not standardized/normalized and the resulting factor levels are, therefore, somewhat ambiguous. We better clean this before getting into data aggregation tasks. Note the `ff`-specific syntax needed to recode the factor.

```
# inspect the factor levels
levels(taxi$payment_type)
```

```
## [1] "Cash"      "CASH"      "Credit"    "CREDIT"    "Dispute"   "No Charge"
```

```
# recode them
levels(taxi$payment_type) <- tolower(levels(taxi$payment_type))
taxi$payment_type <- ff(taxi$payment_type,
                        levels = unique(levels(taxi$payment_type)),
                        ramclass = "factor")
# check result
levels(taxi$payment_type)
```

```
## [1] "cash"      "credit"    "dispute"   "no charge"
```

2.1.2 Aggregation with split-apply-combine

First, we have a look at whether trips paid with credit card tend to involve lower tip amounts than trips paid by cash. In order to do so, we create a table that shows the average amount of tip paid for each payment-type category.

In simple words, this means we first split the data set into subsets, each of which containing all observations belonging to a distinct payment type. Then, we compute the arithmetic mean of the tip-column of each of these subsets. Finally, we combine all of these results in one table (i.e., the split-apply-combine strategy). When working with `ff`, the `ffdfply()`-function provides a user-friendly implementation of split-apply-combine type of tasks.

```
# load packages
library(doBy)

# split-apply-combine procedure on data file chunks
tip_pcategory <- ffdfdply(taxi,
                          split = taxi$payment_type,
                          BATCHBYTES = 100000000,
                          FUN = function(x) {
                               summaryBy(tip_amount~payment_type,
                                         data = x,
```

```
                                        FUN = mean,
                                        na.rm = TRUE)})
```

```
## 2021-03-03 22:08:35, calculating split sizes
## 2021-03-03 22:08:35, building up split locations
## 2021-03-03 22:08:36, working on split 1/2, extracting data in RAM of 1 split elements, totalling, 0.0
## 2021-03-03 22:08:36, ... applying FUN to selected data
## 2021-03-03 22:08:36, ... appending result to the output ffdf
## 2021-03-03 22:08:36, working on split 2/2, extracting data in RAM of 3 split elements, totalling, 0.0
## 2021-03-03 22:08:36, ... applying FUN to selected data
## 2021-03-03 22:08:36, ... appending result to the output ffdf
```

Note how the output describes the procedure step by step. Now we can have a look at the resulting summary statistic in the form of a `data.frame()`.

```
as.data.frame(tip_pcategory)
```

```
##   payment_type tip_amount.mean
## 1         cash    0.0008161834
## 2       credit    2.1619737355
## 3      dispute    0.0035074627
## 4    no charge    0.0041056466
```

The result would go against our initial hypothesis. However, the comparison is a little flawed. If trips paid by credit card also tend to be longer, the result is not too surprising. We should thus look at the share of tip (or percentage), given the overall amount paid for the trip.

We add an additional variable `percent_tip` and then repeat the aggregation exercise for this variable.

```
# add additional column with the share of tip
taxi$percent_tip <- (taxi$tip_amount/taxi$total_amount)*100

# recompute the aggregate stats
tip_pcategory <- ffdfdply(taxi,
                          split = taxi$payment_type,
                          BATCHBYTES = 100000000,
                          FUN = function(x) {
                               summaryBy(percent_tip~payment_type, # note the difference here
                                         data = x,
                                         FUN = mean,
                                         na.rm = TRUE)})
```

```
## 2021-03-03 22:08:36, calculating split sizes
## 2021-03-03 22:08:36, building up split locations
## 2021-03-03 22:08:37, working on split 1/2, extracting data in RAM of 1 split elements, totalling, 0.0
## 2021-03-03 22:08:37, ... applying FUN to selected data
## 2021-03-03 22:08:37, ... appending result to the output ffdf
## 2021-03-03 22:08:37, working on split 2/2, extracting data in RAM of 3 split elements, totalling, 0.0
## 2021-03-03 22:08:37, ... applying FUN to selected data
## 2021-03-03 22:08:38, ... appending result to the output ffdf
```

```
# show result as data frame
as.data.frame(tip_pcategory)
```

```
##   payment_type percent_tip.mean
## 1         cash      0.005978433
## 2       credit     16.004172819
## 3      dispute      0.045659709
## 4    no charge      0.040432542
```

2.1.3 Cross-tabulation of ff vectors

Also in relative terms, trips paid by credit card tend to be tipped more. However, are there actually many trips paid by credit card? In order to figure this out, we count the number of trips per payment type by applying the `table.ff`-function provided in `ffbase`.

```
table.ff(taxi$payment_type)
```

```
##
##      cash    credit   dispute no charge
##    781295    215424       536      2745
```

Incidentally, trips paid in cash are way more frequent than trips paid by credit card. Again using the `table.ff`-function, we investigate what factors might be correlated with payment types. First, we have a look at whether payment type is associated with the number of passengers in a trip.

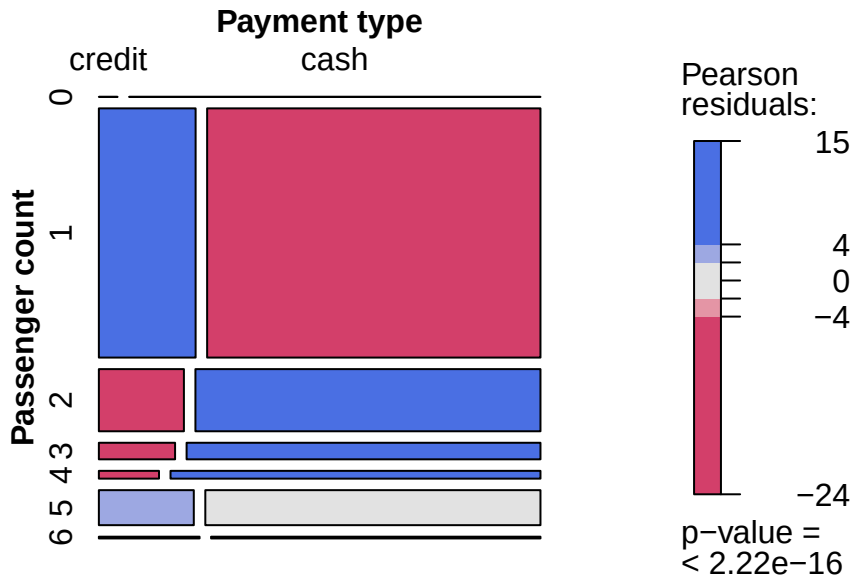
```
# select the subset of observations only containing trips paid by credit card or cash
taxi_sub <- subset.ffdf(taxi, payment_type=="credit" | payment_type == "cash")
taxi_sub$payment_type <- ff(taxi_sub$payment_type,
                        levels = c("credit", "cash"),
                        ramclass = "factor")

# compute the cross tabulation
crosstab <- table.ff(taxi_sub$passenger_count,
                     taxi_sub$payment_type
                     )
# add names to the margins
names(dimnames(crosstab)) <- c("Passenger count", "Payment type")
# show result
crosstab
```

```
##                Payment type
## Passenger count credit   cash
##               0      2     44
##               1 149990 516828
##               2  32891 133468
##               3   7847  36439
##               4   2909  17901
##               5  20688  73027
##               6   1097   3588
```

From the raw numbers it is hard to see whether there are significant differences between the categories cash and credit. We therefore use a visualization technique called 'mosaic plot' to visualize the cross-tabulation.

```
# install.packages(vcd)
# load package for mosaic plot
library(vcd)
```

```
# generate a mosaic plot
mosaic(crosstab, shade = TRUE)
```



The plot suggests that trips involving more than one passenger tend to be paid rather by cash than by credit card.

3 High-speed in-memory data aggregation with `data.table`

For large data sets that still fit into RAM, the `data.table`-package provides very fast and elegant functions to compute aggregate statistics.

3.1 Data import

We use the already familiar `fread()` to import the same first million observations from the January 2009 taxi trips records.

```
# load packages
library(data.table)

# import data into RAM (needs around 200MB)
taxi <- fread("../data/tlc_trips.csv",
              nrows = 1000000)
```

3.2 Data preparation

We prepare/clean the data as in the `ff`-approach above.

```
# first, we remove the empty vars V8 and V9
taxi$V8 <- NULL
taxi$V9 <- NULL


# set covariate names according to the data dictionary
# see https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf
# note instead of taxizonne ids, long/lat are provided
```

```
varnames <- c("vendor_id",
              "pickup_time",
              "dropoff_time",
              "passenger_count",
              "trip_distance",
              "start_lat",
              "start_long",
              "dest_lat",
              "dest_long",
              "payment_type",
              "fare_amount",
              "extra",
              "mta_tax",
              "tip_amount",
              "tolls_amount",
              "total_amount")
names(taxi) <- varnames

# clean the factor levels
taxi$payment_type <- tolower(taxi$payment_type)
taxi$payment_type <- factor(taxi$payment_type, levels = unique(taxi$payment_type))
```

Note the simpler syntax of essentially doing the same thing, but all in-memory.

3.3 `data.table`-syntax for 'split-apply-combine' operations

With the `[]`-syntax we index/subset usual `data.frame` objects in R. When working with `data.table`s, much more can be done in the step of 'sub-setting' the frame.[3]

For example, we can directly compute on columns.

```
taxi[, mean(tip_amount/total_amount)]
```

```
## [1] 0.03452489
```

Moreover, in the same step, we can 'split' the rows *by* specific groups and apply the function to each subset.

```
taxi[, .(percent_tip = mean((tip_amount/total_amount)*100)), by = payment_type]
```

```
##    payment_type  percent_tip
## 1:         cash  0.005978433
## 2:       credit 16.004172819
## 3:    no charge  0.040432542
## 4:      dispute  0.045659709
```

Similarly, we can use `data.table`'s `dcast()` for cross-tabulation-like operations.

```
dcast(taxi[payment_type %in% c("credit", "cash")],
      passenger_count~payment_type,
      fun.aggregate = length,
      value.var = "vendor_id")
```

```
##    passenger_count   cash credit
## 1:               0     44      2
## 2:               1 516828 149990
```

[3] See https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html for a detailed introduction to the syntax.

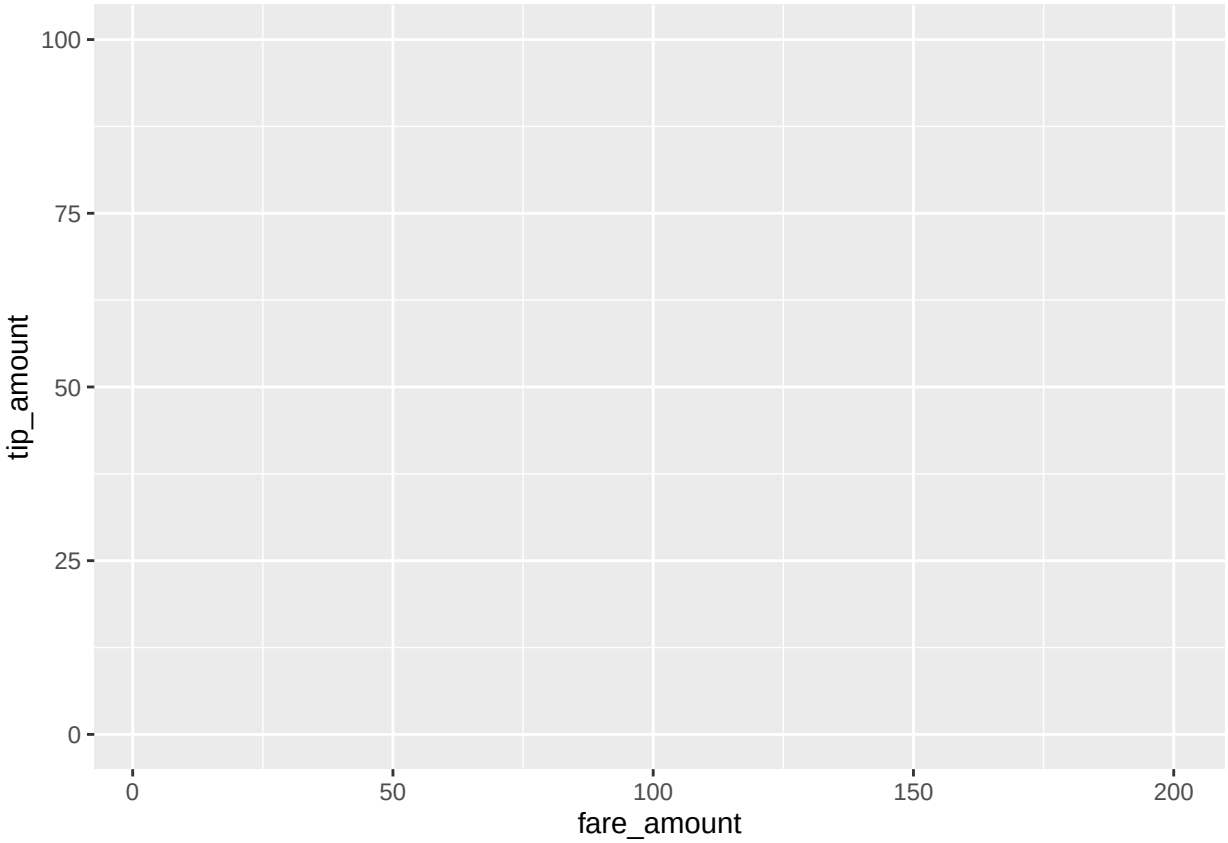
```
## 3:               2 133468  32891
## 4:               3  36439   7847
## 5:               4  17901   2909
## 6:               5  73027  20688
## 7:               6   3588   1097
```

4 (Big) Data Visualization

In order to better understand the large data set at hand (particularly regarding the determinants of tips paid) we use `ggplot2` to visualize some key aspects of the data.

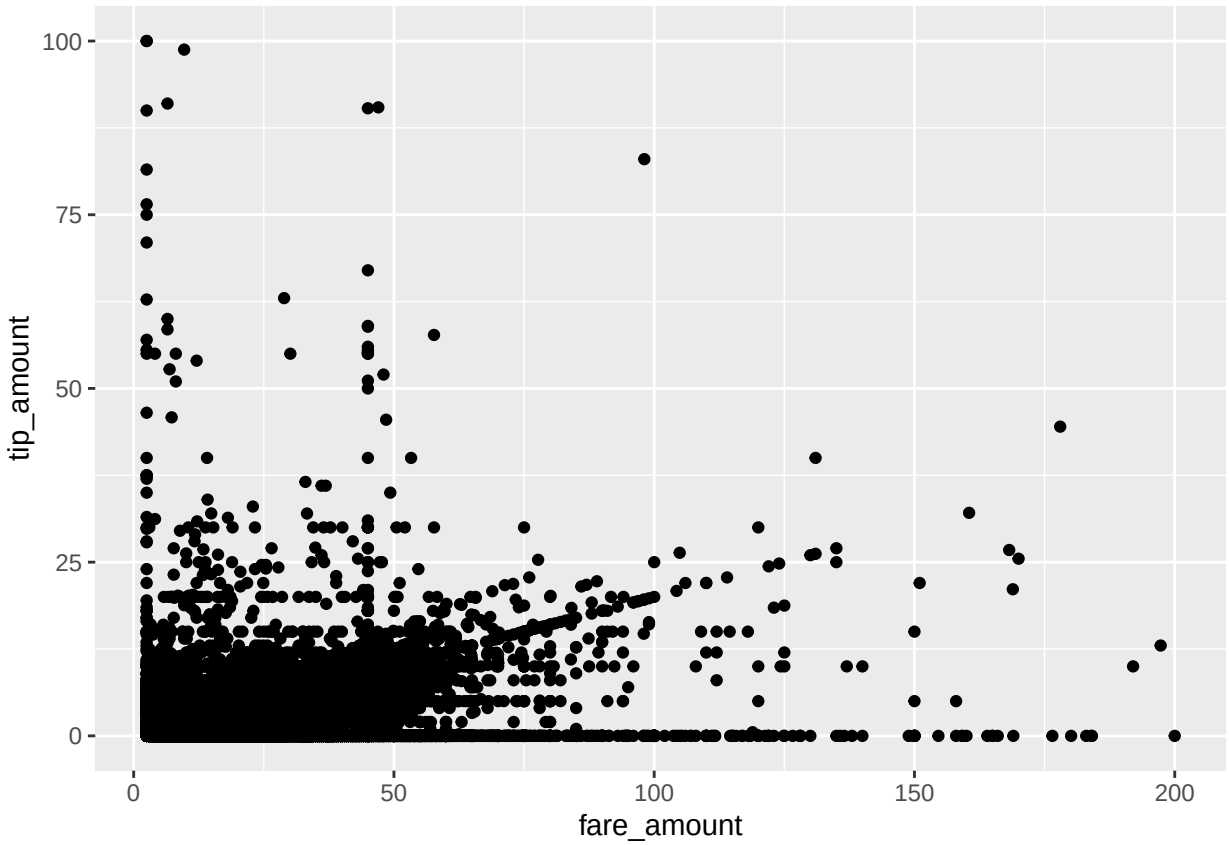
First, let's look at the raw relationship between fare paid and the tip paid. We set up the canvas with `ggplot`.

```
# load packages
library(ggplot2)

# set up the canvas
taxiplot <- ggplot(taxi, aes(y=tip_amount, x= fare_amount))
taxiplot
```



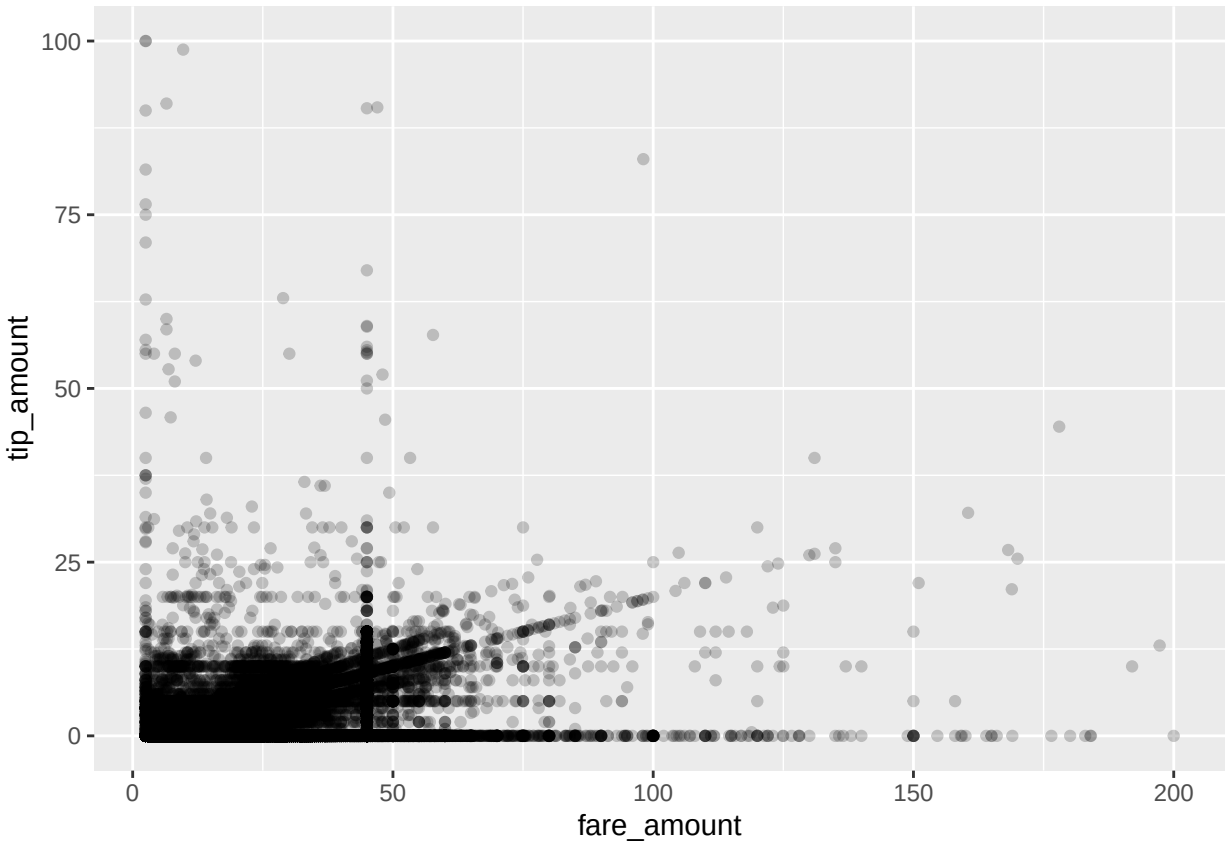
Now we visualize the co-distribution of the two variables with a simple scatter-plot.

```
# simple x/y plot
taxiplot +
     geom_point()
```



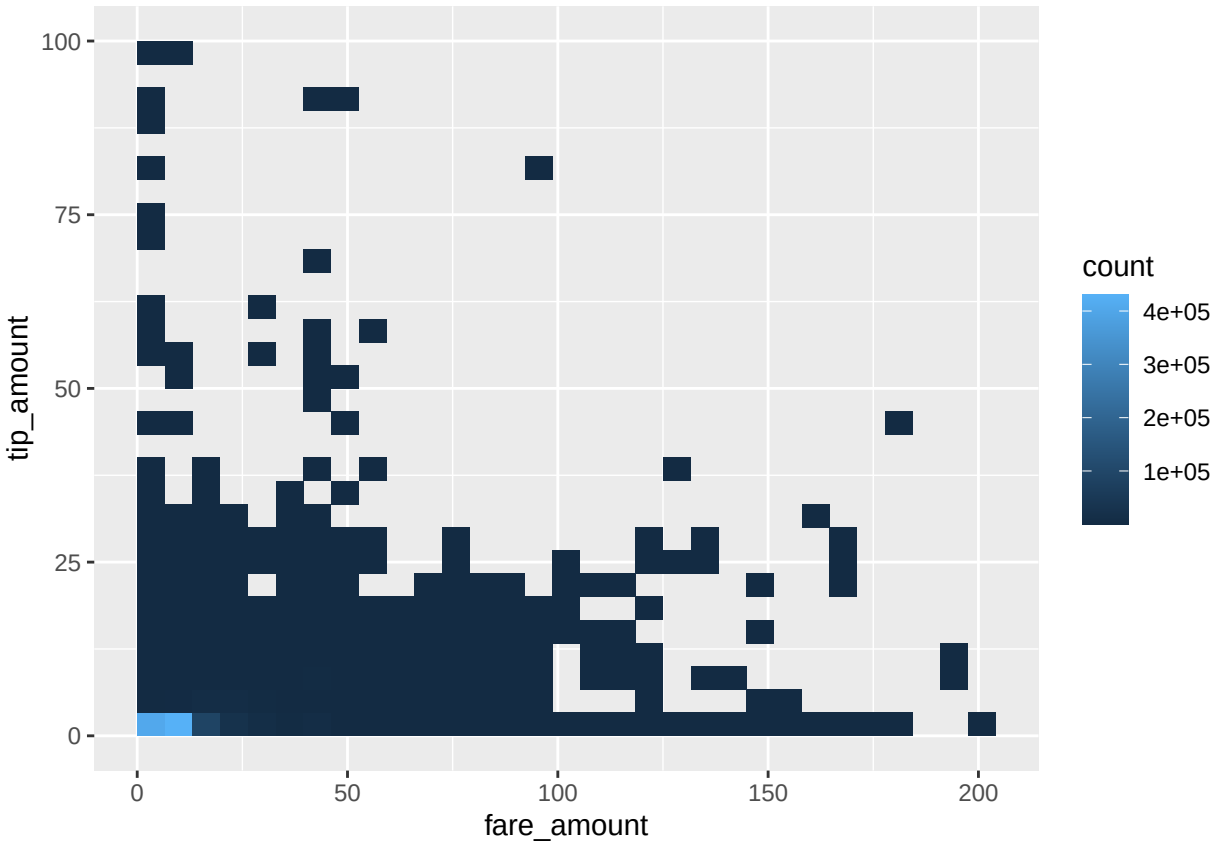
Note that this took quite a while, as R had to literally plot one million dots on the canvas. Moreover many dots fall within the same area, making it impossible to recognize how much mass there actually is. This is typical for visualization exercises with large data sets. One way to improve this, is by making the dots more transparent by setting the `alpha` parameter.

```
# simple x/y plot
taxiplot +
     geom_point(alpha=0.2)
```



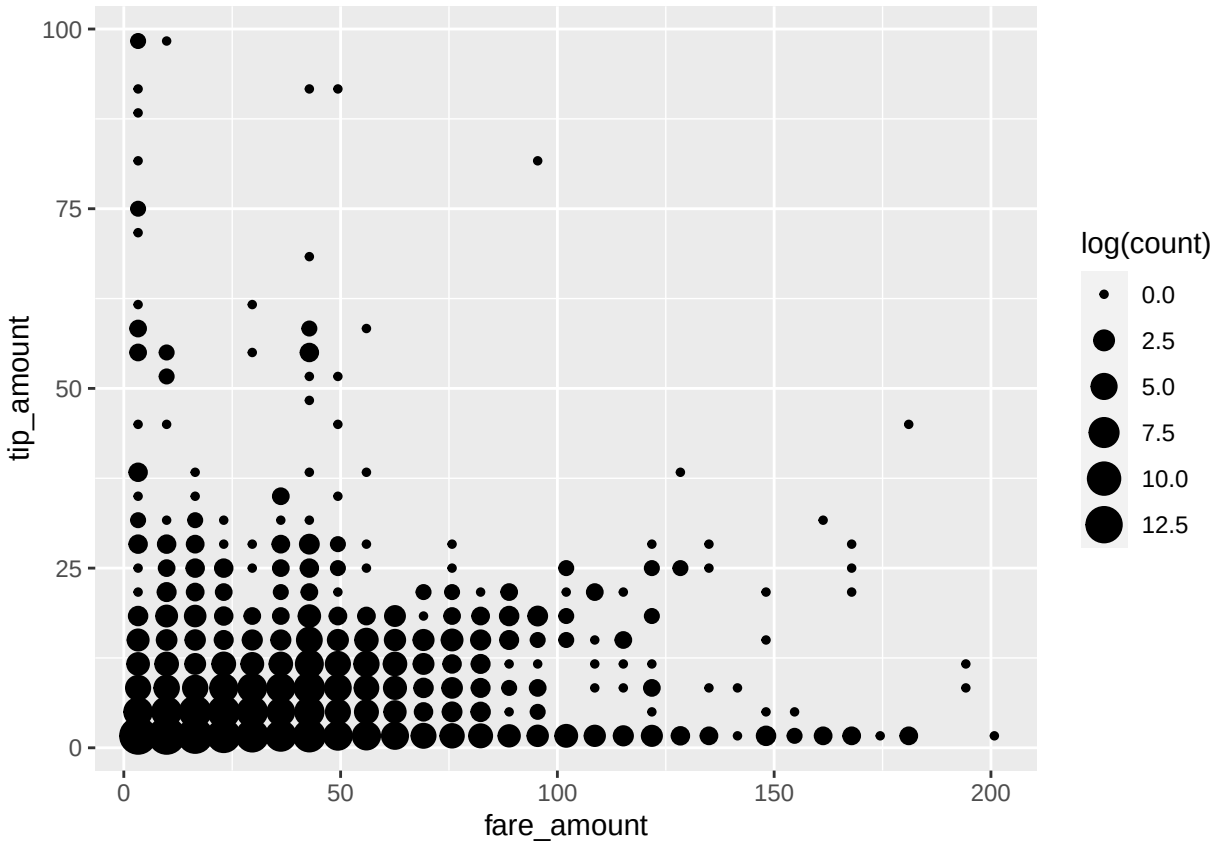
Alternatively, we can compute two-dimensional bins. Thereby, the canvas is split into rectangles and the number of observations falling into each respective rectangle is computed. The visualization is based on plotting the rectangles with counts greater than 0 and the shading of the rectangles indicates the count values.

```
# 2-dimensional bins
taxiplot +
     geom_bin2d()
```



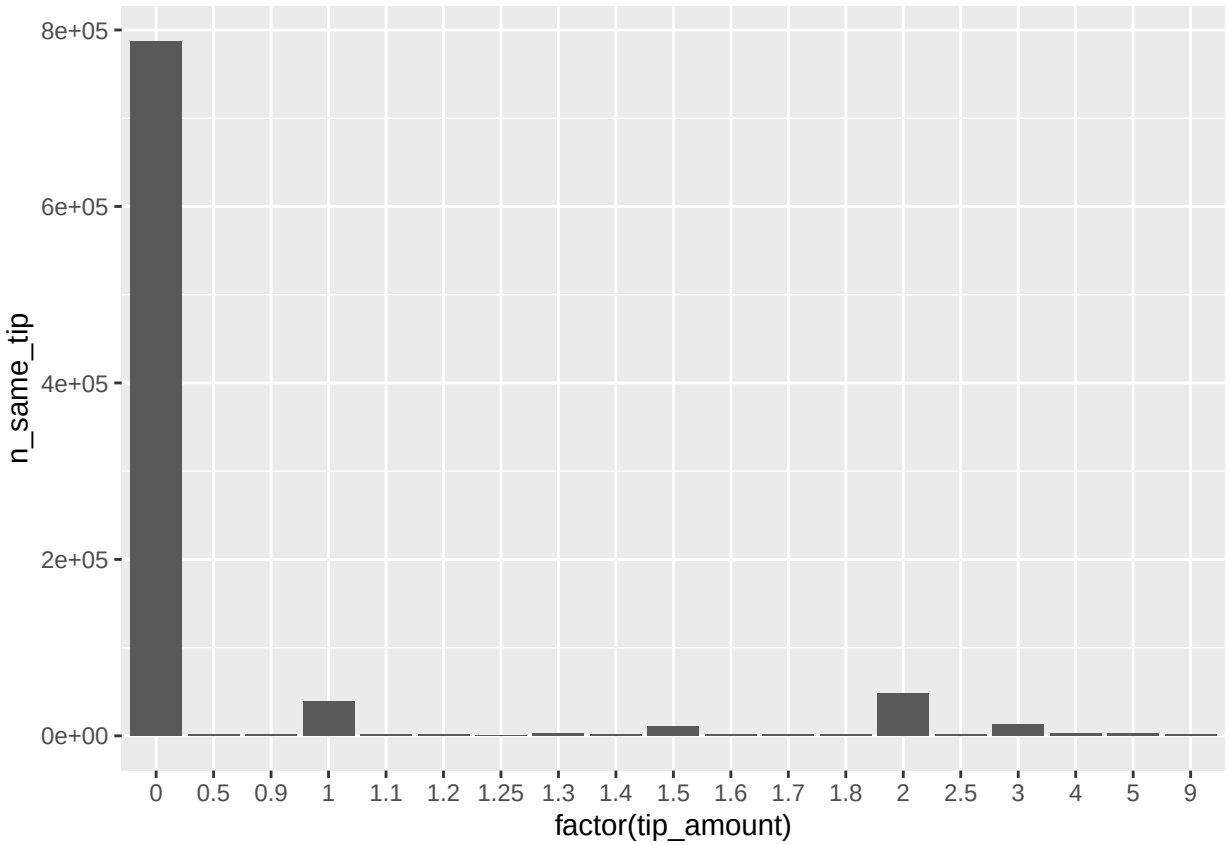
A large part of the tip/fare observations seem to be in the very lower-left corner of the pane, while most other trips seem to be evenly distributed. However, we fail to see slighter differences in this visualization. In order to reduce the dominance of the 2d-bins with very high counts, we display the natural logarithm of counts and display the bins as points.

```
# 2-dimensional bins
taxiplot +
     stat_bin_2d(geom="point",
                 mapping= aes(size = log(..count..))) +
     guides(fill = FALSE)
```



We note that there are many cases with very low fare amounts, many cases with no or hardly any tip, and quite a lot of cases with very high tip amounts (in relation to the rather low fare amount). In the following, we dissect this picture by having a closer look at 'typical' tip amounts and whether they differ by type of payment.

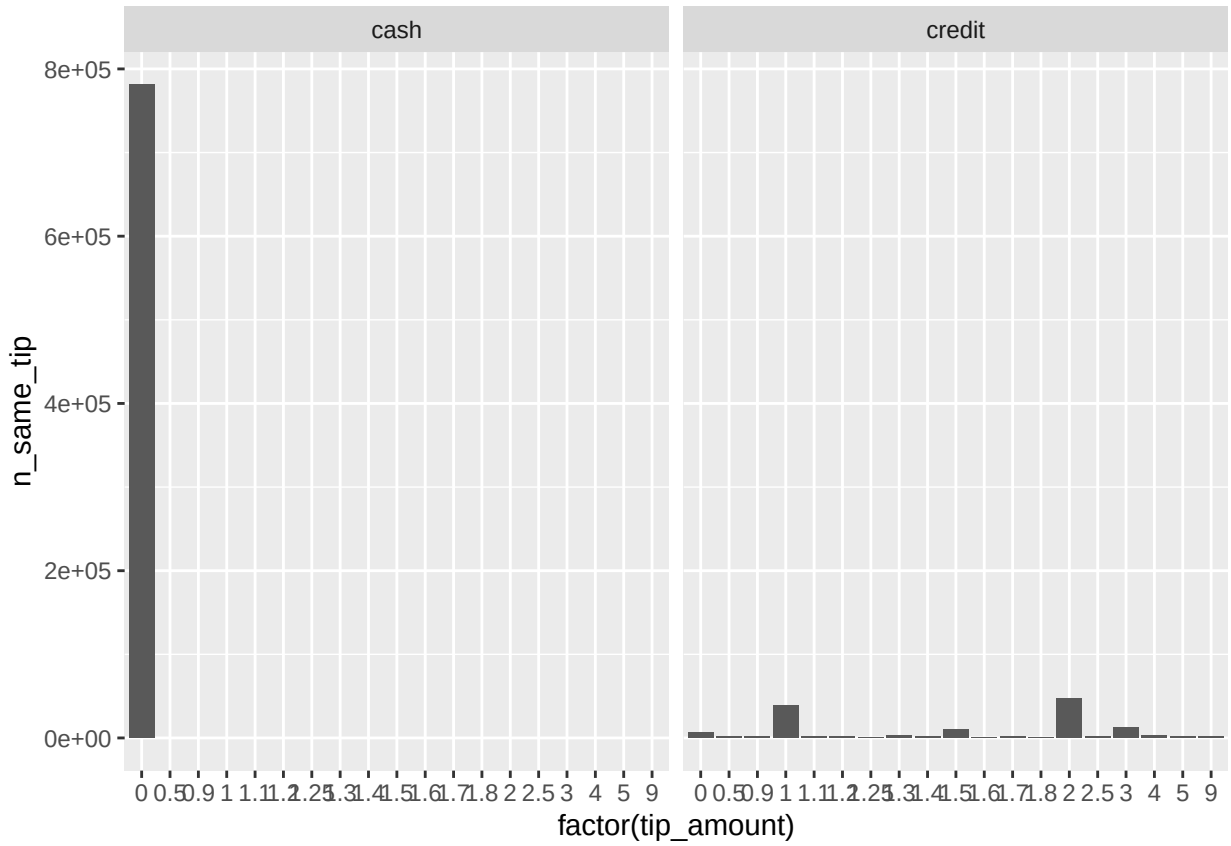
```
# compute frequency of per tip amount and payment method
taxi[, n_same_tip:= .N, by= c("tip_amount", "payment_type")]
frequencies <- unique(taxi[payment_type %in% c("credit", "cash"),
                           c("n_same_tip", "tip_amount", "payment_type")][order(n_same_tip, decreasing =


# plot top 20 frequent tip amounts
fare <- ggplot(data = frequencies[1:20], aes(x = factor(tip_amount), y = n_same_tip))
fare + geom_bar(stat = "identity")
```



Indeed, paying no tip at all is quite frequent, overall.[4] The bar plot also indicates that there seem to be some 'focal points' in the amount of tips paid. Clearly, paying one USD or two USD is more common than paying fractions. However, fractions of dollars might be more likely if tips are paid in cash and customers simply add some loose change to the fare amount paid.

```
fare + geom_bar(stat = "identity") +
     facet_wrap("payment_type")
```

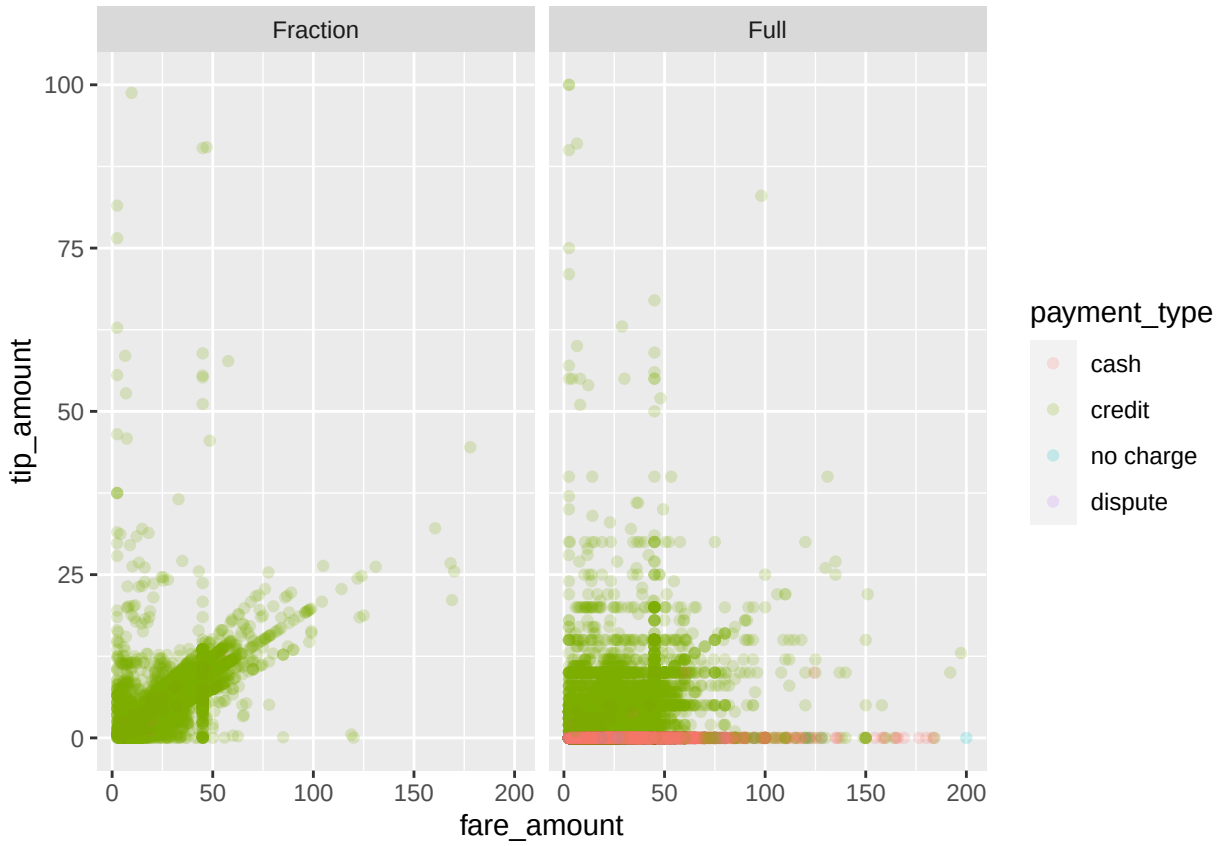
[4] Or, could there be another explanation for this pattern in the data?



Clearly, it looks like trips paid in cash tend not to be tipped (at least in this subsample).

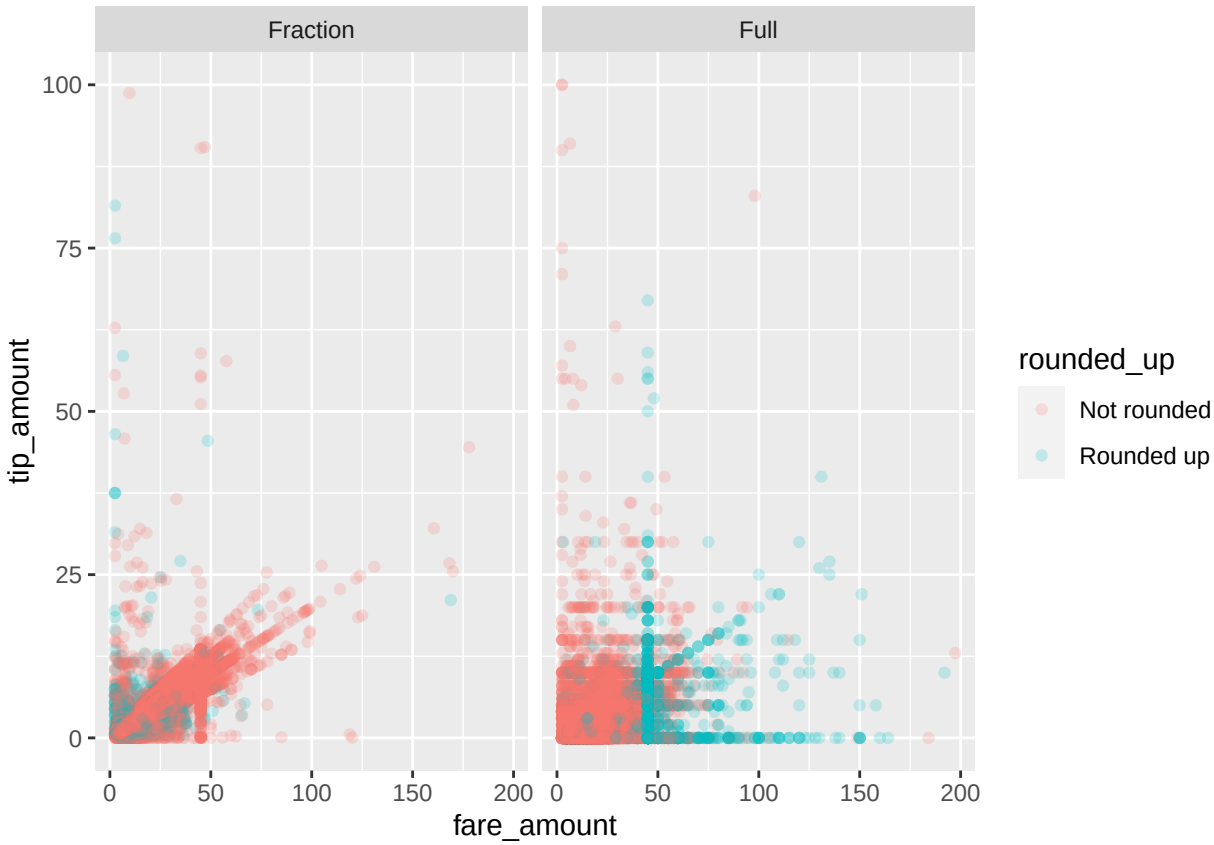
Let's try to tease this information out of the initial points plot. Trips paid in cash are often not tipped, we thus should indicate the payment method. Moreover, tips paid in full dollar amounts might indicate a habit.

```
# indicate natural numbers
taxi[, dollar_paid := ifelse(tip_amount == round(tip_amount,0), "Full", "Fraction"),]


# extended x/y plot
taxiplot +
     geom_point(alpha=0.2, aes(color=payment_type)) +
     facet_wrap("dollar_paid")
```



Now the picture is getting clearer. Paying tip seems to follow certain rules of thumb. Certain fixed amounts tend to be paid independent of the fare amount (visible in the straight lines of dots on the right-hand panel). At the same time, the pattern in the left panel indicates another habit: computing the amount of tip as a linear function of the total fare amount ('pay 10% tip'). A third habit might be to determine the amount of tip by 'rounding up' the total amount paid. In the following, we try to tease the latter out, only focusing on credit card payments.

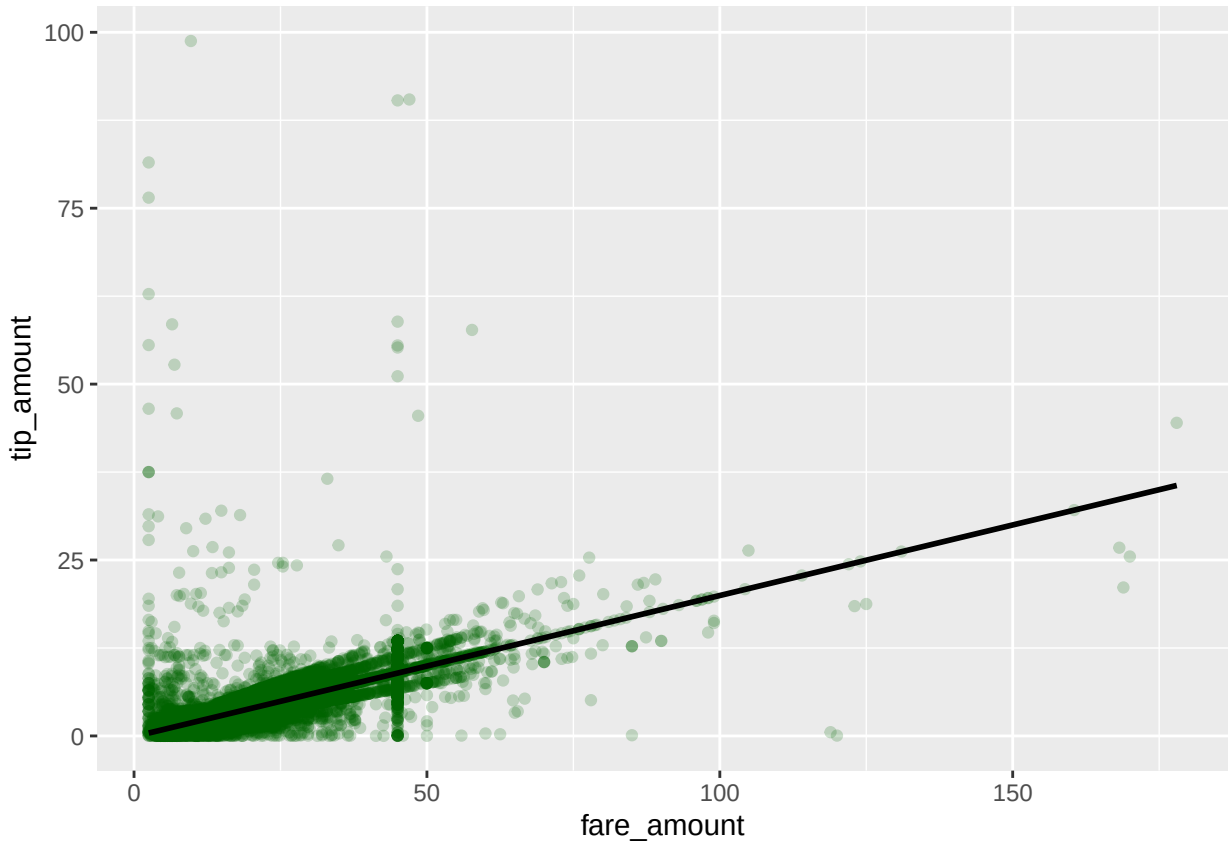
```
taxi[, rounded_up := ifelse(fare_amount + tip_amount == round(fare_amount + tip_amount, 0),
                            "Rounded up",
                            "Not rounded")]
# extended x/y plot
taxiplot +
     geom_point(data= taxi[payment_type == "credit"],
                alpha=0.2, aes(color=rounded_up)) +
     facet_wrap("dollar_paid")
```



Finally, we can start modelling. A reasonable first shot is to model the tip amount as a linear function of the fare amount, conditional on the no-zero tip amounts paid as fractions of a dollar.

```
modelplot <- ggplot(data= taxi[payment_type == "credit" & dollar_paid == "Fraction" & 0 < tip_amount],
                    aes(x = fare_amount, y = tip_amount))
modelplot +
     geom_point(alpha=0.2, colour="darkgreen") +
     geom_smooth(method = "lm", colour = "black")
```

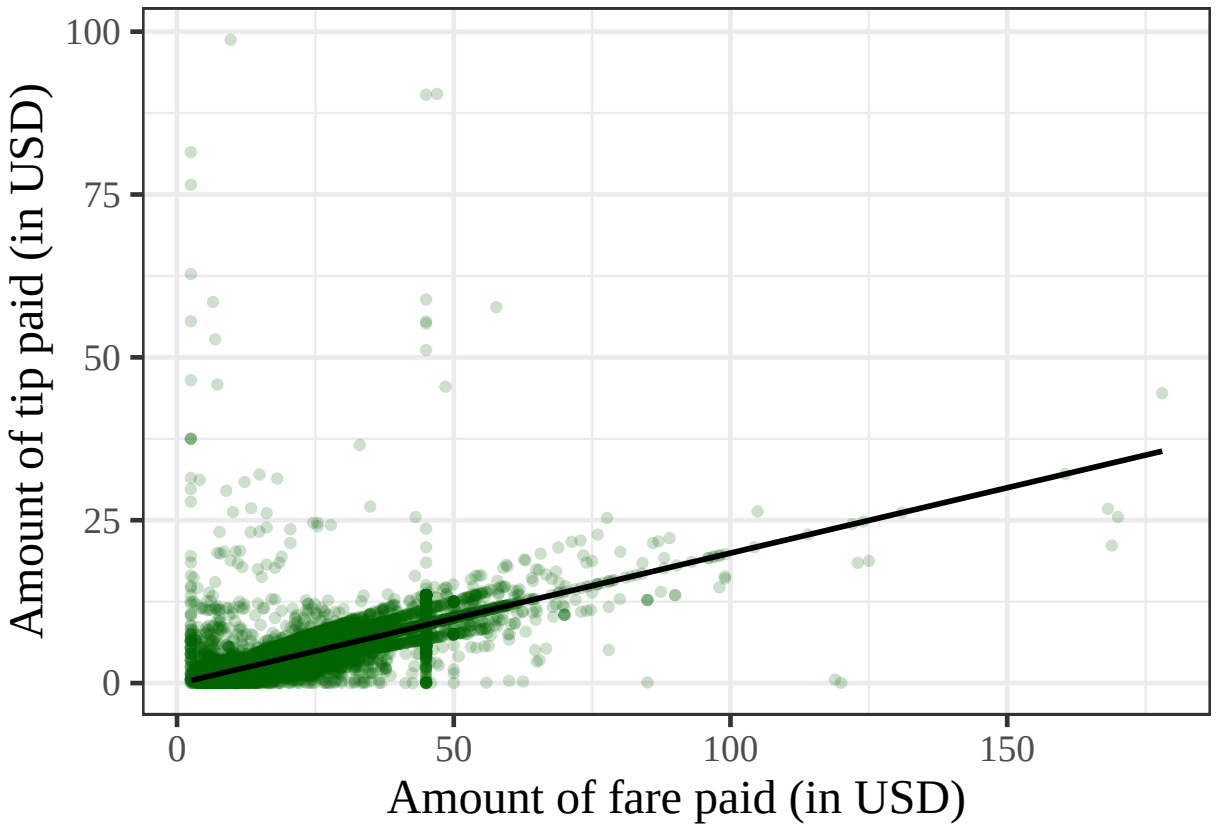
```
## `geom_smooth()` using formula 'y ~ x'
```



Finally, we prepare the plot for reporting.

```
modelplot <- ggplot(data= taxi[payment_type == "credit" & dollar_paid == "Fraction" & 0 < tip_amount],
                    aes(x = fare_amount, y = tip_amount))
modelplot +
     geom_point(alpha=0.2, colour="darkgreen") +
     geom_smooth(method = "lm", colour = "black") +
     ylab("Amount of tip paid (in USD)") +
     xlab("Amount of fare paid (in USD)") +
     theme_bw(base_size = 18, base_family = "serif")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



References

Wickham, Hadley. 2011. "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software* 40 (1).