Big Data Analytics

Lecture 6: Aggregation and Visualization I

Prof. Dr. Ulrich Matter
01/04/2021

Updates

Examination Part I: Timeline of take-home exercises

- Examination handed out via GitHub (Classroom): 7 May 2020
- Deadline to hand in results: **8 June 2020 (16:00)**

Format of take-home exercises

- GitHub classroom group assignment.
- Basic starter code handed out as repository.
- A data analytics project based on a large data set, including the entire data pipeline.
- Tasks
    - Instructions in README
    - Improve efficiency of given code
    - Extend code: complete specific tasks
    - Explain/document procedure (conceptual understanding)
- ‘Product’: the repository, including R code, and a report in R markdown.

Examination Part II: Group Projects/Presentations

- Groups formed decentrally (same groups as for take-home exercises).
- Own research question, find a data set, think of approach/strategy, implement in R, presentation of results as Rmd/R-presentation recorded in a ‘screencast’.
- Hand in screencast via Canvas/Studynet (assignment is already open), commit code/rmd to GitHub-classroom (initial group formation assignment).

Register in GitHub Classroom

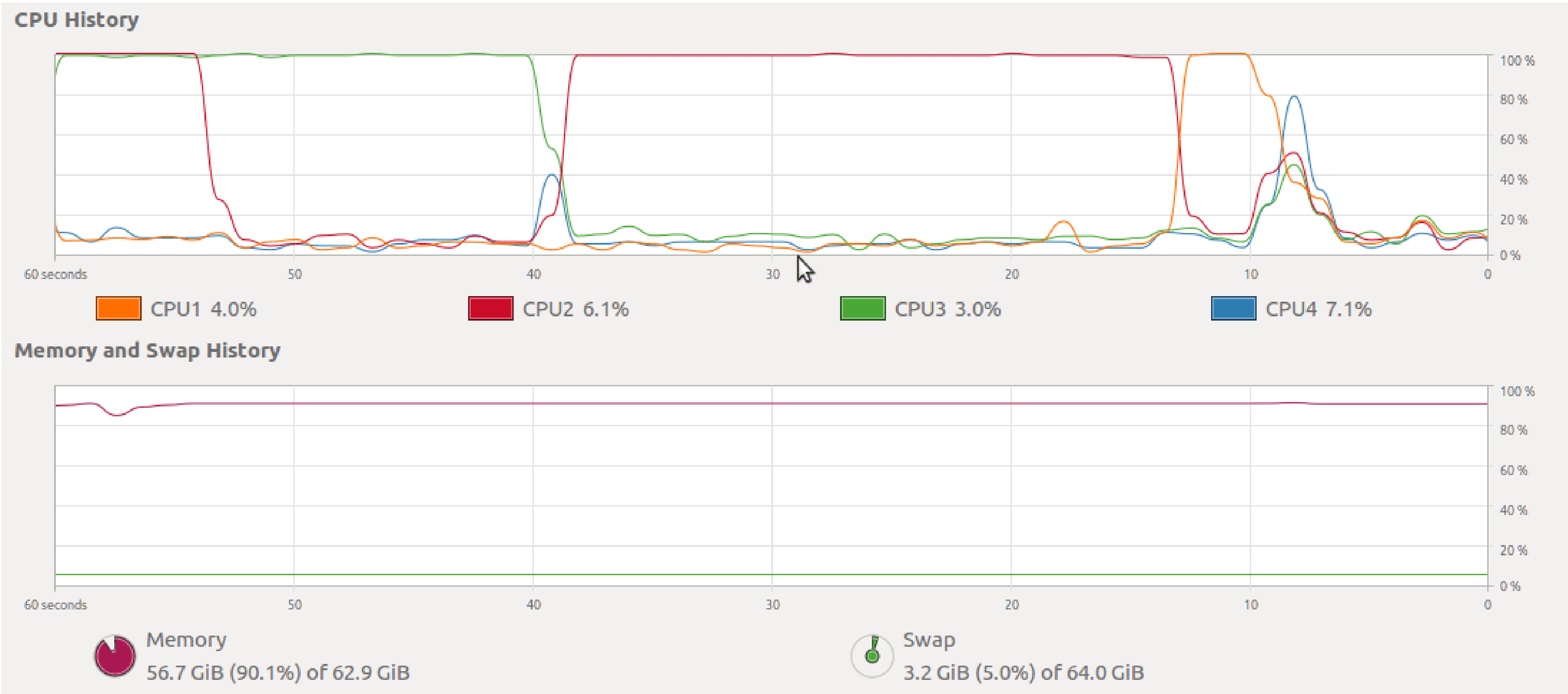
- **By the end of the month, teams must be set!**
- Please register, if you have not done so yet and join your team in GitHub Classroom!
- Still problems finding a team? Use the **Q&A Section in Canvas**! In case of emergencies, email me: ulrich.matter@unisg.ch

Recap Week 5

Beyond memory

- RAM is not sufficient to handle the amount of data to be analyzed…
- **What to do?**
- Scale up by using parts of the available Mass Storage (hard-disk) as *virtual memory.

Virtual memory



Out-of-memory strategies

- Chunked data files on disk
- Memory-mapped files and shared memory

Out-of-memory strategies

- Chunked data files on disk: `ff`-package
- Memory-mapped files and shared memory: `bigmemory`-package

Aggregation and Visualization

Setting: NYC yellow caps

- Data source: NYC Taxi & Limousine Commission (TLC)
- Data on all trip records including pick-up and drop-off times/locations.
  - (2009-2018)
  - Trip-level observations
  - Amount of fare paid
  - Amount of tip paid, etc.
- All raw data: over 200GB
  - **Here: First 1 million observations (in January 2009)**

Gathering and Compilation of all the raw data

```
##################################
# 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") # set to "2009-01-01" for the first file only


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

# parse base url
base_url <- gsub("2018-01.csv", "", BASE_URL)
```

Data aggregation with chunked data files

Data aggregation: The ‘split-apply-combine’ strategy

- Background: Compute a statistic for specific groups (e.g. women vs men, etc.)

1. Split the data into subsamples (e.g. one for women, one for men)
2. Compute the statistic for each of the subsamples.
3. Combine all results in one table.

Preparation: Data import and cleaning

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

```
# 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",
```

Preparation: Data import and cleaning

Following the data documentation provided by TLC, we give the columns of our data set more meaningful names.

```
# 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",
```

Preparation: Data cleaning

```
# 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"
```

Aggregation with split-apply-combine

- Goal: a table that shows the average amount of tip paid for each payment-type category.
- Approach: `ffdfply()` and `summaryBy()`

Aggregation with split-apply-combine

```r
# 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)})
```

```
## 2021-02-19 15:25:40, calculating split sizes
```

```
## 2021-02-19 15:25:40, building up split locations
```

```
## 2021-02-19 15:25:40, working on split 1/2, extracting data in RAM of 1 split elements, totalling,
```

```
## 2021-02-19 15:25:40, ... applying FUN to selected data
```

```
## 2021-02-19 15:25:41, ... appending result to the output ffdf
```

Aggregation with split-apply-combine

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
```

Aggregation with split-apply-combine

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)})
```

```
## 2021-02-19 15:25:41, calculating split sizes

## 2021-02-19 15:25:41, building up split locations

## 2021-02-19 15:25:41, working on split 1/2, extracting data in RAM of 1 split elements, totalling,

## 2021-02-19 15:25:42, ... applying FUN to selected data
```

Cross-tabulation of `ff` vectors

Goal: Get number of observations by covariate-values Approach: Cross-tabulatoni with `table.ff()` (`ffbase`-package)

Cross-tabulation of `ff` vectors

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

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

Cross-tabulation of `ff` vectors

- What factors are correlated with payment types?
- Is payment type associated with the number of passengers in a trip?

Cross-tabulation of `ff` vectors

```
# 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
```

Visualization of cross-tabulations

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

```
## Loading required package: grid
```

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

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

Necessary condition for `data.table`

- Data still fit into RAM
- Possible with our subsample of 1 million rows (on most modern computers).
- Unlikely to work well with the full data set (200GB)

Data import

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

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

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

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")
```

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

- With `[]` 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 'subsetting' the frame.

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

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

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

And we can do the same with 'splitting' the rows first **by** specific groups and apply the function to each batch of observations.

```
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
```

data.table-syntax for cross-tabulations

Similarly we can use `data.table`'s `dcast()` for crosstabulation-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:               2 133468  32891
## 4:               3  36439   7847
## 5:               4  17901   2909
## 6:               5  73027  20688
## 7:               6   3588   1097
```

(Big) Data Visualization

ggplot2

- ‘Grammar of Graphics’
- Build plots layer-by-layer
- **Here**: Usefull tool for explorative visualization
- In-memory operations
  - Works well with 1 million obs.

Exploration: what determines tip amounts?

Set up the canvas…

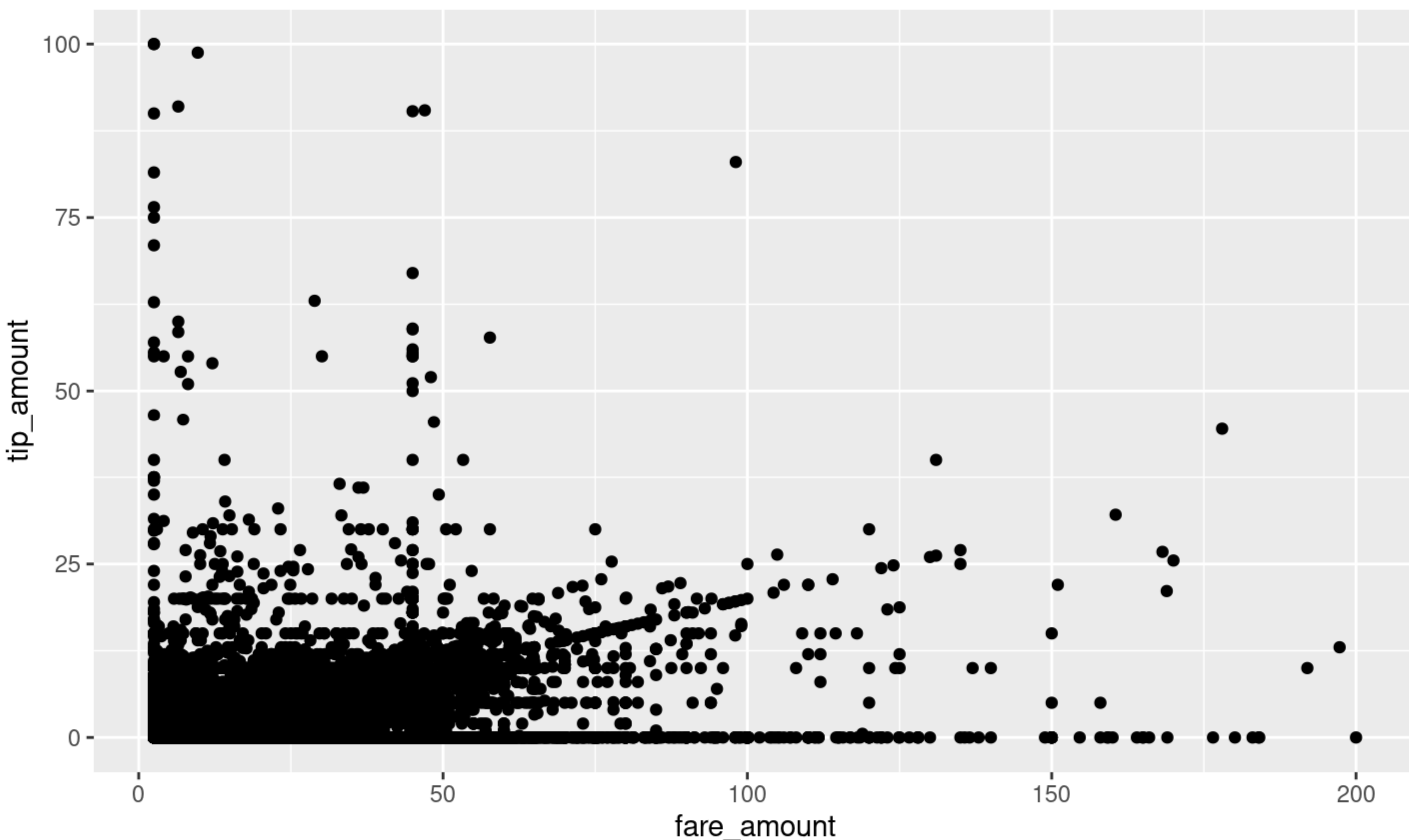
```
# load packages
library(ggplot2)

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

Exploration: what determines tip amounts?

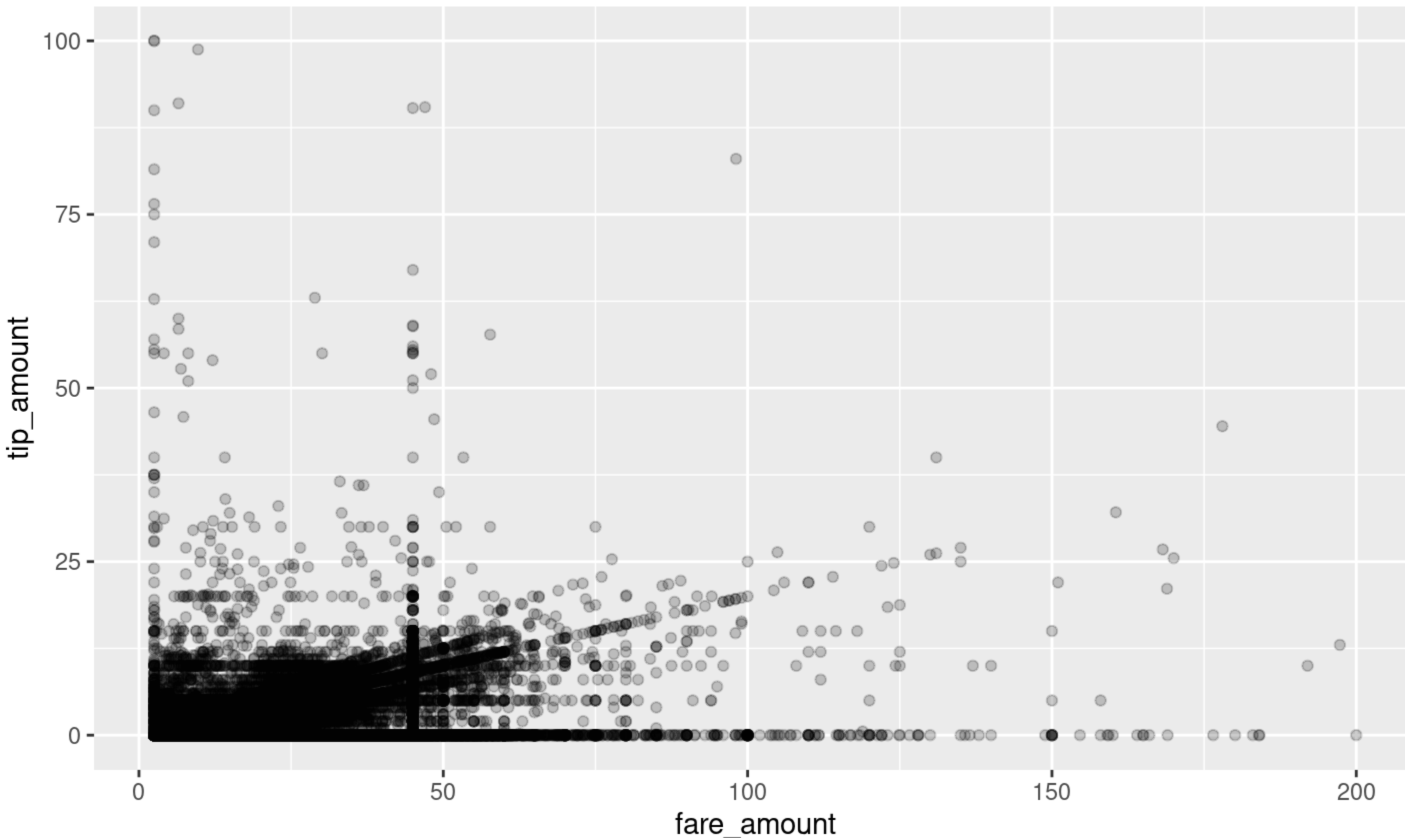
Visualize the co-distribution of the two variables with a simple scatter-plot.

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



Problem: too many points

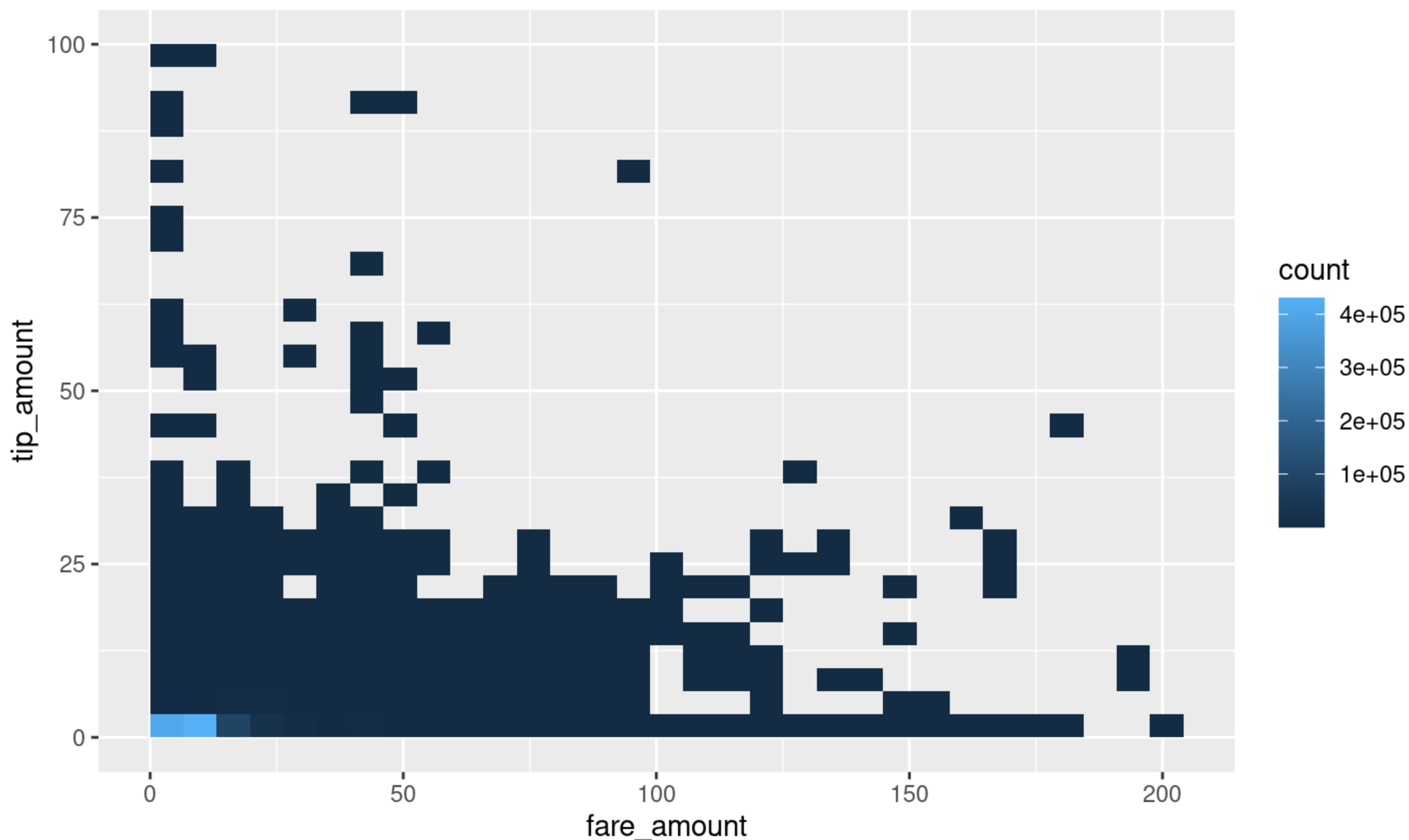
```
# simple x/y plot
taxiplot +
    geom_point(alpha=0.2)
```



2-D bins

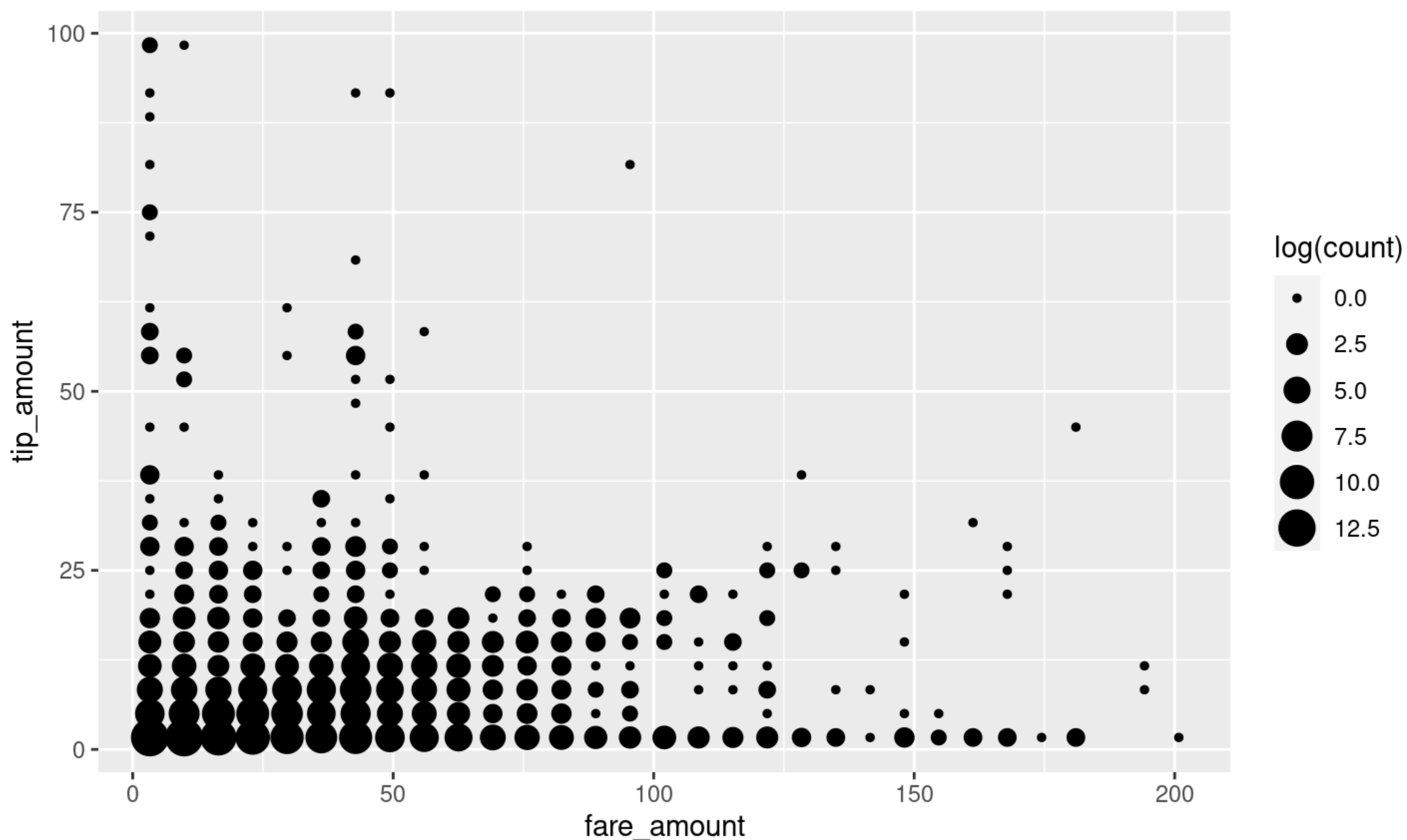
Where are most observations located?

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



2-D bins: ln of count

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

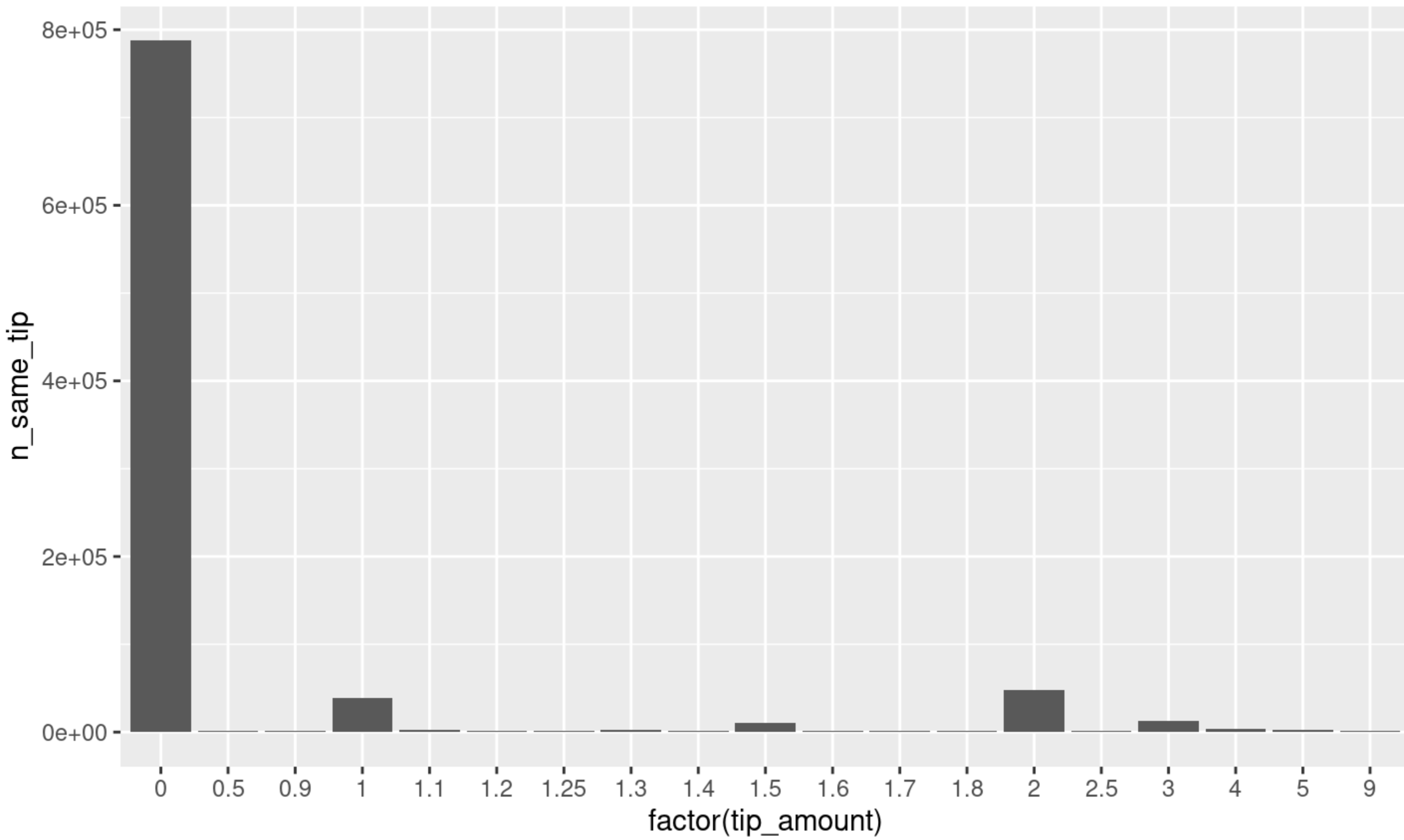


Frequencies

```
# 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, decreasi
```

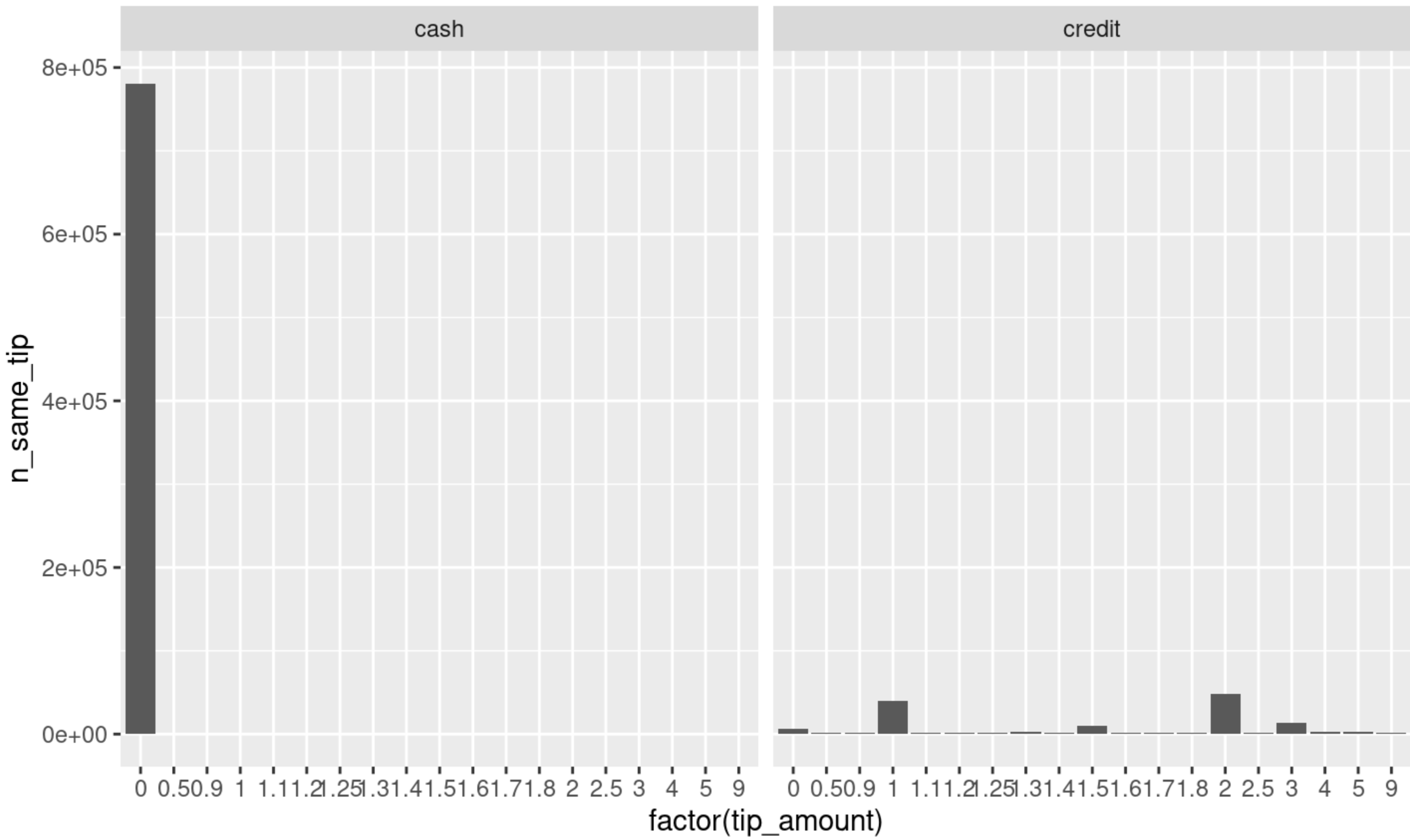
Frequencies

```
# 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")
```



Split by payment type

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



Split by payment type

Let's have a closer look at non-zero tip amounts.

```
fare + geom_bar(data = frequencies[2:40],
                stat = "identity") +
     facet_wrap("payment_type")
```

Payment habits?

Fractions of dollars due to loose change as tip?

```
# 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")
```

Payment habits?

Rounding up?

```
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")
```

Modelling of payment habits

‘X% tip rule’?

```
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'
```

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