Snacks & Science: an introduction to R

Bas Baccarne

Contents

Part 1: The basics

Why R?

- Open source
  - free!
- Reproducible research
  - transparent!
- Packages
  - flexible!
- Data collection and transformations
  - multi-featured!

The R interface

- Lower left: console (commands and responses, this is R, the rest is R Studio)
  - write commands here (try `1+1`)
  - if it works, add to script
  - clear console with `CTRL + L`
- Upper left: scripts (text files)
  - use `#` for comments
  - use `CTRL + ENTER` to run a single line or selection
  - save as .R files
- Upper right: environment (data & functions)
  - includes brief summary
  - data.frames can be clicked
  - clear with broom icon
- Lower right: help
  - type `?searchterm` in console for the help file of a function or package
  - type `??searchterm` in console for non-exact matching searches

Data objects in R

The basic element in R is a single variable (stored with `<-` or `=`)

```r
# store value "the dude" in a variable called name
name <- "the dude"
# store the number 43 in a variable called age
age <- 43
# store the logical value TRUE in a variable called male
male <- TRUE
# store the current time in a variable called date
date <- Sys.time()
# note that character data is always between " "
```

Each variable has a class (think of it as a data type)

```r
class(name)
```

```
## [1] "character"
```

```
class(age)
```

```
## [1] "numeric"
```

```
class(male)
```

```
## [1] "logical"
```

```
class(date)
```

```
## [1] "POSIXct" "POSIXt"
```

You can play with these variables

```
age / 2
```

```
## [1] 21.5
```

```
paste(name, "is cool")
```

```
## [1] "the dude is cool"
```

```
paste("the answer to life, the universe and everything is", age - 1)
```

```
## [1] "the answer to life, the universe and everything is 42"
```

```
weekdays(date)
```

```
## [1] "maandag"
```

Variables are often combined in arrays (e.g.)

```
ninjaturtles <- c("Michelangelo", "Leonardo", "Donatello", "Raphael")
ninjaturtles
```

```
## [1] "Michelangelo" "Leonardo"     "Donatello"    "Raphael"
```

The most common data object is a data.frame (similar to Excel or SPSS)

```
df <- data.frame(
    number_id = c("the first number", "the second number", "the third number"),
    number = c(1,2,3),
    even_number = c(FALSE, TRUE, FALSE)
)
df
```

```
##           number_id number even_number
## 1  the first number      1       FALSE
## 2 the second number      2        TRUE
## 3  the third number      3       FALSE
```

Functions in R

Besides data, R also uses functions that allow more complicated processing. Functions are formatted as `function(arguments)`. The arguments between brackets provide the values that are processed by the function. The `rnorm()` function, for example, requires an amount of random numbers you want (n), and the mean and standard deviation of the distribution from which the random numers are drawn (mean and sd) as arguments:

```
rnorm(n = 2, mean = 10, sd = 1)
```

```
## [1] 10.18920 11.94368
```

Most functions come with some documentation that explains the arguments and output value.

```
?rnorm
```

You can also create your own function like this:

```
make_awesome <- function(name){
    paste(name, "is super awesome!")
}
```

And then use it like this (it is now also visible in your environment)

```
make_awesome("R")
```

```
## [1] "R is super awesome!"
```

Packages in R

Most functions in R are bundled into packages. R comes with a large set of preinstalled packages (see lower-right panel > packages). But you can also download new packages (e.g. to get Twitter data). There are tons of packages that allow you to do the craziest things. You can install them like this:

```
install.packages("ggplot2")
```

Once installed, you can load packages like this

```
library("ggplot2")
```

Most errors in R come from typo's or packages that are not installed or loaded. Learn how to interpret errors! Remember that there are packages for just about everything. Google is your friend.

Part 2: Getting data

Loading data in R

You can get data from several sources, both local and online. As long as it has a certain *structure*, R can *parse* it (JSON, HTML, CSV, SAV, log files as TXT, . . .).

In this example we will use sample data from the mobileDNA project, stored as a CSV file (6 respondents, one week of data).

```
logs <- read.csv2("sampledata.csv", as.is = TRUE)
```

You can see in the environment that we have a new data object (a data.frame) with 5699 rows and 17 variables. We can now start exploring these data.

Exploring the data

```
# general summaries (for an extended summary: use summary(logs))
str(logs)
```

```
## 'data.frame':    5699 obs. of  17 variables:
##  $ X              : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ endTimeMillis  : num  1.52e+12 1.52e+12 1.52e+12 1.52e+12 1.52e+12 ...
##  $ id             : chr  "5344c7e2-4fe1-463a-8e61-58b66e697a7c" "937ad0f6-70ee-4f7c-99ea-34852b74b86
##  $ model          : chr  "SM-A310F" "MotoG3" "MotoG3" "SM-G955F" ...
##  $ session        : int  1522562164 1522591129 1522592153 1522601484 1522601484 1522601578 152260157
##  $ startTimeMillis: num  1.52e+12 1.52e+12 1.52e+12 1.52e+12 1.52e+12 ...
##  $ startTime      : chr  "2018-04-01T07:56:09.615" "2018-04-01T16:05:14.288" "2018-04-01T16:22:27.91
##  $ endTime        : chr  "2018-04-01T07:57:02.470" "2018-04-01T16:08:23.523" "2018-04-01T16:22:41.31
##  $ notification   : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ application    : chr  "com.whatsapp" "com.android.chrome" "com.Relmtech.Remote" "com.whatsapp" ..
##  $ battery        : int  97 33 42 19 19 19 19 17 29 26 ...
##  $ latitude       : num  0 51 51 51 51 ...
##  $ longitude      : num  0 3.7 3.7 3.67 3.67 ...
##  $ data_version   : chr  "1.4" "1.4" "1.4" "1.4" ...
##  $ social_app     : logi  TRUE FALSE FALSE TRUE FALSE FALSE ...
##  $ duration_sec   : num  52.9 189.2 13.4 27.9 30.3 ...
##  $ startTijdstip  : chr  "07:56" "16:05" "16:22" "18:51" ...
```

```
# variable names
names(logs)
```

```
##  [1] "X"               "endTimeMillis"   "id"
##  [4] "model"           "session"         "startTimeMillis"
##  [7] "startTime"       "endTime"         "notification"
## [10] "application"     "battery"         "latitude"
## [13] "longitude"       "data_version"    "social_app"
## [16] "duration_sec"    "startTijdstip"
```

```
# count the rows
nrow(logs)
```

```
## [1] 5699
```

```
# first rows
head(logs)
```

```
##   X endTimeMillis                                   id    model    session
## 1 1  1.522562e+12 5344c7e2-4fe1-463a-8e61-58b66e697a7c SM-A310F 1522562164
## 2 2  1.522592e+12 937ad0f6-70ee-4f7c-99ea-34852b74b861   MotoG3 1522591129
## 3 3  1.522593e+12 937ad0f6-70ee-4f7c-99ea-34852b74b861   MotoG3 1522592153
## 4 4  1.522602e+12 573ab0a9-55c2-4532-934a-ea8f0f5d78cd SM-G955F 1522601484
## 5 5  1.522602e+12 573ab0a9-55c2-4532-934a-ea8f0f5d78cd SM-G955F 1522601484
## 6 6  1.522602e+12 573ab0a9-55c2-4532-934a-ea8f0f5d78cd SM-G955F 1522601578
##   startTimeMillis               startTime                 endTime
## 1    1.522562e+12 2018-04-01T07:56:09.615 2018-04-01T07:57:02.470
## 2    1.522592e+12 2018-04-01T16:05:14.288 2018-04-01T16:08:23.523
## 3    1.522593e+12 2018-04-01T16:22:27.911 2018-04-01T16:22:41.319
## 4    1.522601e+12 2018-04-01T18:51:38.247 2018-04-01T18:52:06.130
## 5    1.522602e+12 2018-04-01T18:52:13.050 2018-04-01T18:52:43.351
## 6    1.522602e+12 2018-04-01T18:53:02.243 2018-04-01T18:53:17.296
##   notification                   application battery latitude longitude
## 1        FALSE                  com.whatsapp      97  0.00000  0.000000
## 2        FALSE            com.android.chrome      33 51.04971  3.696372
## 3        FALSE           com.Relmtech.Remote      42 51.04971  3.696372
## 4        FALSE                  com.whatsapp      19 51.04946  3.665288
## 5        FALSE  com.samsung.android.contacts      19 51.04946  3.665288
## 6        FALSE com.samsung.android.messaging      19 51.04946  3.665288
##   data_version social_app duration_sec startTijdstip
## 1          1.4       TRUE       52.855         07:56
## 2          1.4      FALSE      189.235         16:05
## 3          1.4      FALSE       13.408         16:22
## 4          1.4       TRUE       27.883         18:51
## 5          1.4      FALSE       30.301         18:52
## 6          1.4      FALSE       15.053         18:53
```

Subsetting data

These statistics are all at the level of the total data.frame. However, most of the time we're interested in a subset of the data. Subsetting data can be done in several ways

You can both subset rows and columns:

```
# subsetting the second row of the third variable (id) > square brackets =  index based
logs[2,3]
```

```
## [1] "937ad0f6-70ee-4f7c-99ea-34852b74b861"
```

```
# subsetting the same variable, through 'name subsetting' > dollar sign = name based
logs$id[2]
```

```
## [1] "937ad0f6-70ee-4f7c-99ea-34852b74b861"
```

```
# which unique IDs are in the data
unique(logs$id)
```

```
## [1] "5344c7e2-4fe1-463a-8e61-58b66e697a7c"
## [2] "937ad0f6-70ee-4f7c-99ea-34852b74b861"
## [3] "573ab0a9-55c2-4532-934a-ea8f0f5d78cd"
## [4] "442be784-049d-4add-9000-7f8605c53e1c"
## [5] "ca9ff4af-1e04-4de2-b858-7b0b4d61322e"
## [6] "35a9137c-72d6-4830-9237-2afdc39d5536"
```

```r
# give the first timestamps
head(logs$startTime)
```

```
## [1] "2018-04-01T07:56:09.615" "2018-04-01T16:05:14.288"
## [3] "2018-04-01T16:22:27.911" "2018-04-01T18:51:38.247"
## [5] "2018-04-01T18:52:13.050" "2018-04-01T18:53:02.243"
```

```r
# what is the class of the variable duration_sec?
class(logs$duration_sec)
```

```
## [1] "numeric"
```

```r
# crosstabs of id and notifications
table(logs$id, logs$notification)
```

```
##
##                                         FALSE TRUE
##   35a9137c-72d6-4830-9237-2afdc39d5536   381   22
##   442be784-049d-4add-9000-7f8605c53e1c   698   47
##   5344c7e2-4fe1-463a-8e61-58b66e697a7c  1305  121
##   573ab0a9-55c2-4532-934a-ea8f0f5d78cd  1337  135
##   937ad0f6-70ee-4f7c-99ea-34852b74b861   665   40
##   ca9ff4af-1e04-4de2-b858-7b0b4d61322e   874   74
```

Cleaning data

Data is often dirty. Although the level of dirtiness can vary, some cleaning is almost always required. For example `logs$startTime` is a character vector and no timestamp vector.

```r
logs$startTime[1]
```

```
## [1] "2018-04-01T07:56:09.615"
```

```r
class(logs$startTime)
```

```
## [1] "character"
```

To change this:

```r
# how to transform characters to dates? Provide the structure of the data (see ?strptime)
strptime(logs$startTime[1],"%Y-%m-%dT%H:%M:%OS")
```

```
## [1] "2018-04-01 07:56:09 CEST"
```

```r
# ok, that works
# now overwrite the existing startTime variable with a clean one
logs$startTime <- as.POSIXct(strptime(logs$startTime,"%Y-%m-%dT%H:%M:%OS"))
```

Part 3: Processing & analyzing data

New variables

In the data cleaning part, we showed how we can override variables. We can also create new variables like this

```
# add a new variable with the value TRUE for all rows
logs$new <- TRUE
head(logs$new)
```

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

Of course that's not very usefull. Most of the time, we'll want to calculate new variables based on other variables. For example, the duration variable was calculated by subtracting the startime in milliseconds from the endtime in milliseconds:

```
logs$new_duration <- logs$endTimeMillis - logs$startTimeMillis
head(logs$new_duration)
```

```
## [1]  52855 189235  13408  27883  30301  15053
```

```
# oh ow, this is still in milliseconds, which is hard to interpret.
# Let's change this to seconds:
logs$new_duration <- logs$new_duration/1000
head(logs$new_duration)
```

```
## [1]  52.855 189.235  13.408  27.883  30.301  15.053
```

```
# now this looks the same as this, hurray!
head(logs$duration)
```

```
## [1]  52.855 189.235  13.408  27.883  30.301  15.053
```

Another example: let's extract the hour and the weekday of the startTime variable:

```
# new variable: weekday
logs$weekday <- weekdays(logs$startTime)
head(logs$weekday)
```

```
## [1] "zondag" "zondag" "zondag" "zondag" "zondag" "zondag"
```

```
# new variable: time in hours and minutes
logs$time <- format(logs$startTime, format = "%H:%M")
head(logs$time)
```

```
## [1] "07:56" "16:05" "16:22" "18:51" "18:52" "18:53"
```

This this event happen during the weekend?

```
logs$weekend <- ifelse(logs$weekday=="zaterdag" | logs$weekday=="zondag", TRUE, FALSE)
head(logs$weekend)
```

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

```
table(logs$weekend, logs$weekday)
```

```
##
##         dinsdag donderdag maandag vrijdag woensdag zaterdag zondag
##   FALSE     885       875     905     712      747        0      0
##   TRUE        0         0       0       0        0      608    967
```

Which other variables could we calculate? Do you have any ideas of your own?

Aggregations

The examples above focused on extra variables in within the same data format (each row is an 'app event'). Another benefit of R as a research tool is its flexibility to play with different data formats. For example: we could create a data.frame with 'class hours' for every respondent and compare that to the timestamps of the app usage to add a new variable 'during class hours', TRUE/FALSE to our app event data.frame.

Another way to play with different data formats is to generate aggregated data.frames. The 'logs' data.frame is a dataframe in which each row is an **app event**, we can also easily generate data.frames in which each row is a **session**, a **day**, a **respondent**, etc. The most common way to do this is with a combination of `group_by()` and `aggregate()` (part of the 'dplyr' package). Let me show you how:

```r
library(dplyr)
```

Aggregate per session

```r
sessions <- logs %>%
        # group by user id & session id
        group_by(id, session) %>%
        # define summary variables
        summarize(
            # what was the date of this session?
            datum = as.Date(min(startTime)),
            # how many apps were used in this session?
            appcount = length(application),
            # does this session contain only social apps?
            all_social_apps = !any(social_app == FALSE),
            # was this session triggered by a social app?
            social_trigger = social_app[1],
            # what was the duration of this session
            duration_sec = sum(duration_sec),
            # at which hour did this session start?
            startTijdstip = startTijdstip[1],
            # which apps were used during this session?
            apps = paste(application,collapse=" -> "),
            # which weekday?
            weekday = weekday[1],
            # what time?
            time = time[1],
            # which hour?
            hour = format(startTime, format = "%H")[1]
            )

head(sessions)
```

```
## # A tibble: 6 x 12
## # Groups:   id [1]
##                                      id    session      datum appcount
##                                   <chr>      <int>     <date>    <int>
## 1 35a9137c-72d6-4830-9237-2afdc39d5536 1522566419 2018-04-01        1
## 2 35a9137c-72d6-4830-9237-2afdc39d5536 1522567013 2018-04-01        1
## 3 35a9137c-72d6-4830-9237-2afdc39d5536 1522569023 2018-04-01        1
## 4 35a9137c-72d6-4830-9237-2afdc39d5536 1522571910 2018-04-01        1
## 5 35a9137c-72d6-4830-9237-2afdc39d5536 1522573084 2018-04-01        1
## 6 35a9137c-72d6-4830-9237-2afdc39d5536 1522573137 2018-04-01        1
## # ... with 8 more variables: all_social_apps <lgl>, social_trigger <lgl>,
## #   duration_sec <dbl>, startTijdstip <chr>, apps <chr>, weekday <chr>,
## #   time <chr>, hour <chr>
```

Aggregate per user

```
users <- sessions %>%
        # group by user id & session id
        group_by(id) %>%
        # define summary variables
        summarize(
            # how many apps did this user use?
            appcount = sum(appcount),
            # how many session took place during this period?
            sessions = length(session),
            # how many social only session took place
            social_only_n = sum(all_social_apps),
            # which proportion of sessions were 'social only' sessions
            social_only_p = sum(all_social_apps)/length(session),
            # how many sessions were triggered by a social app?
            social_trigger_n = sum(social_trigger),
            # which proportion of sessions were triggered by social apps?
            social_trigger_p = sum(social_trigger)/length(session)
            )

head(users)
```

```
## # A tibble: 6 x 7
##                                      id appcount sessions social_only_n
##                                   <chr>    <int>    <int>         <int>
## 1 35a9137c-72d6-4830-9237-2afdc39d5536      403      148             8
## 2 442be784-049d-4add-9000-7f8605c53e1c      745      207            23
## 3 5344c7e2-4fe1-463a-8e61-58b66e697a7c     1426      466           143
## 4 573ab0a9-55c2-4532-934a-ea8f0f5d78cd     1472      318           129
## 5 937ad0f6-70ee-4f7c-99ea-34852b74b861      705      219            13
## 6 ca9ff4af-1e04-4de2-b858-7b0b4d61322e      948      284            45
## # ... with 3 more variables: social_only_p <dbl>, social_trigger_n <int>,
## #   social_trigger_p <dbl>
```

Which other variables could we calculate? Do you have any ideas of your own?

Pro tip: always define how your data should look like before you start processing. Know what kind of data you need, and in which format it should be.

Statistics

Once you're done with preprocessing your data, R is also an excellent tool for data analysis. Most statistical analyses are available in the base package. For SEM, use the **lavaan** package.

T-tests

Do sessions that are ***triggered by a social app*** have a longer ***duration***?

```r
# what is the mean duration for social triggered apps versus non-social triggered apps?
tapply(logs$duration_sec, logs$social_app, mean)
```

```
##    FALSE     TRUE 
## 55.22984 54.61688
```

```r
# perform t-test
t <- t.test(duration_sec ~ social_trigger, data = sessions)
t$statistic
```

```
##        t 
## 1.418361
```

```r
t$parameter
```

```
##       df 
## 1599.388
```

```r
t$p.value
```

```
## [1] 0.1562801
```

```r
# no significant difference
```

Correlations

Is there a correlation between ***battery level*** and the ***duration*** of the app usage?

```r
r <- cor.test(logs$battery, logs$duration_sec)
r
```

```
## 
##  Pearson's product-moment correlation
## 
## data:  logs$battery and logs$duration_sec
## t = 0.84787, df = 5697, p-value = 0.3965
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.01473539  0.03718533
## sample estimates:
##        cor 
## 0.01123254
```

```r
# no significant correlation
```

Usa a similar approach for regression, chi square, SEM, ...

Part 4: Visualizing data

This is the fun part! At least for me :) The most powerful dataviz engine in R is ***ggplot***, a very handy package which makes use of the principles of ***grammar of graphics*** . Grammar of graphics builds datavisualizations based on the following principles:

- Data is visualized in a ***sphere*** or canvas that must be defined. Most of the time, this is a two dimensional space with an X and a Y coordinate.
- Next, objects or ***geoms*** can be positioned within this sphere. Each row in the dataset is represented by such a geom, and the position of this geom can reflect one or two variables.
- Finally, these geoms can vary in their ***aesthetics*** (color, shape, border, size, alpha, thickness, . . .), which can all reflect another variable in the dataset
- Additionally, you can split up the sphere in different spheres based on once another variable, called ***facets***

This way, you can visualize up to 10 variables in a single datavisualization (although this is often a bit of a data-interpretation overdose). Time for the action!

```r
library("ggplot2")
```

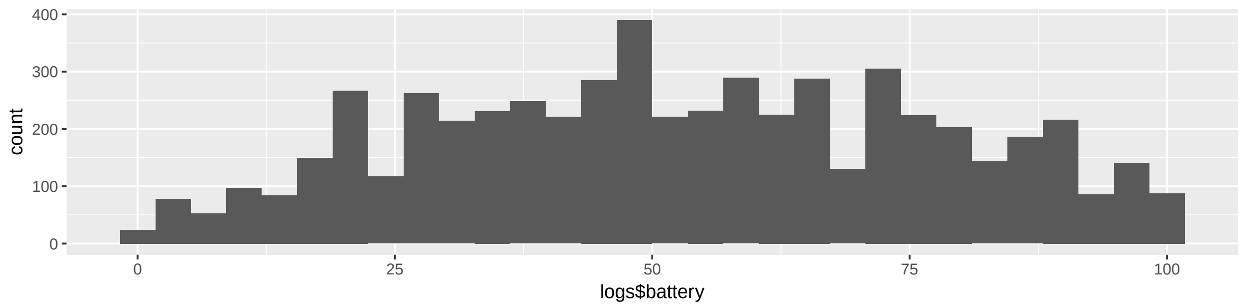
The gg object

Similar to everything else in R (data, functions, results of statistical analyses, . . .), a ggplot viz is also stored as a ggplot object in the R environment. We must first define the sphere, we can then add layer after layer.

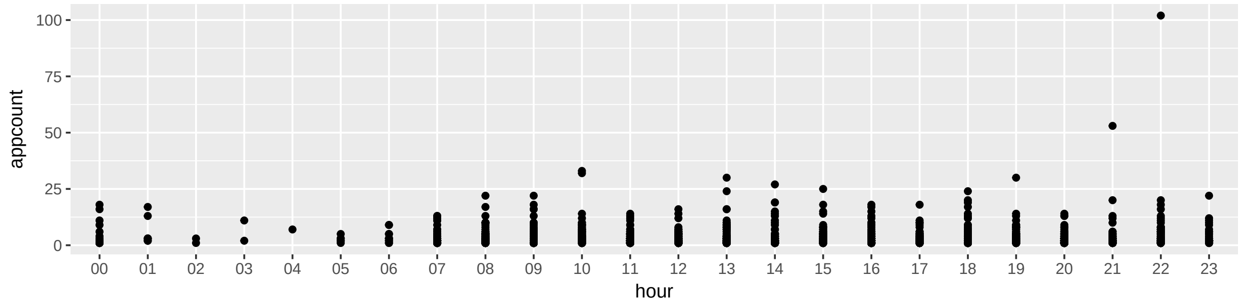
Lazy quick plots: Qplot!

If you quickly want to check something, you can skip this and simply use qplot:

```r
qplot(logs$battery)
```



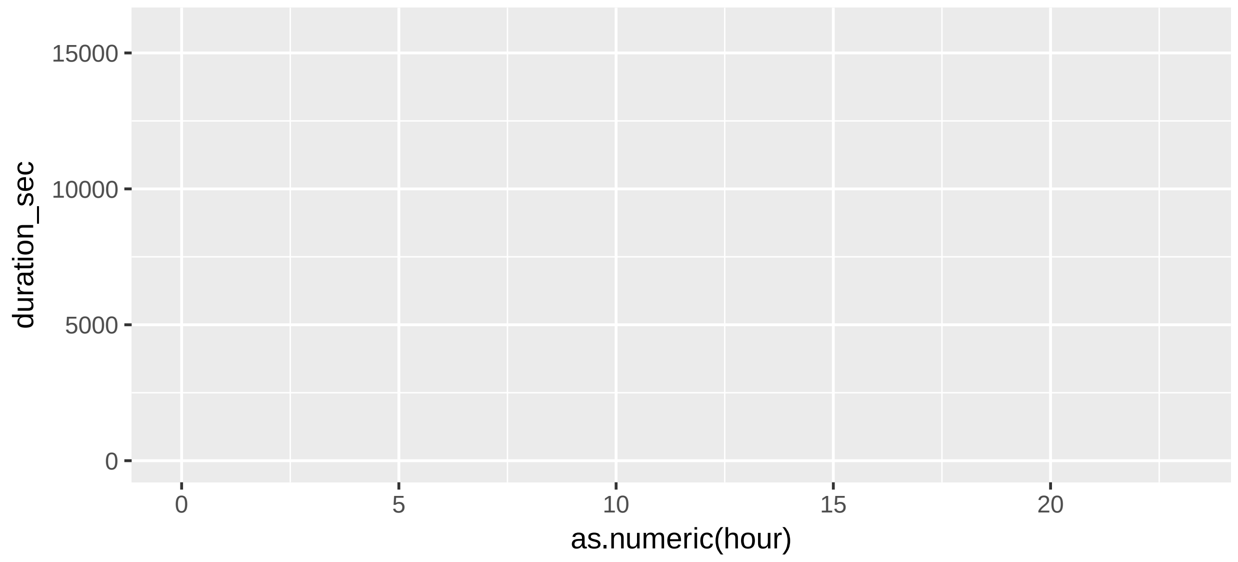
```r
qplot(hour, appcount, data = sessions)
```



The real deal: ggplot()

However, for more advanced visualizations we want to work more granular. First, we define the sphere with an X and a Y axis:

```r
g <- ggplot(data = sessions, aes( x = as.numeric(hour), y = duration_sec))
g
```

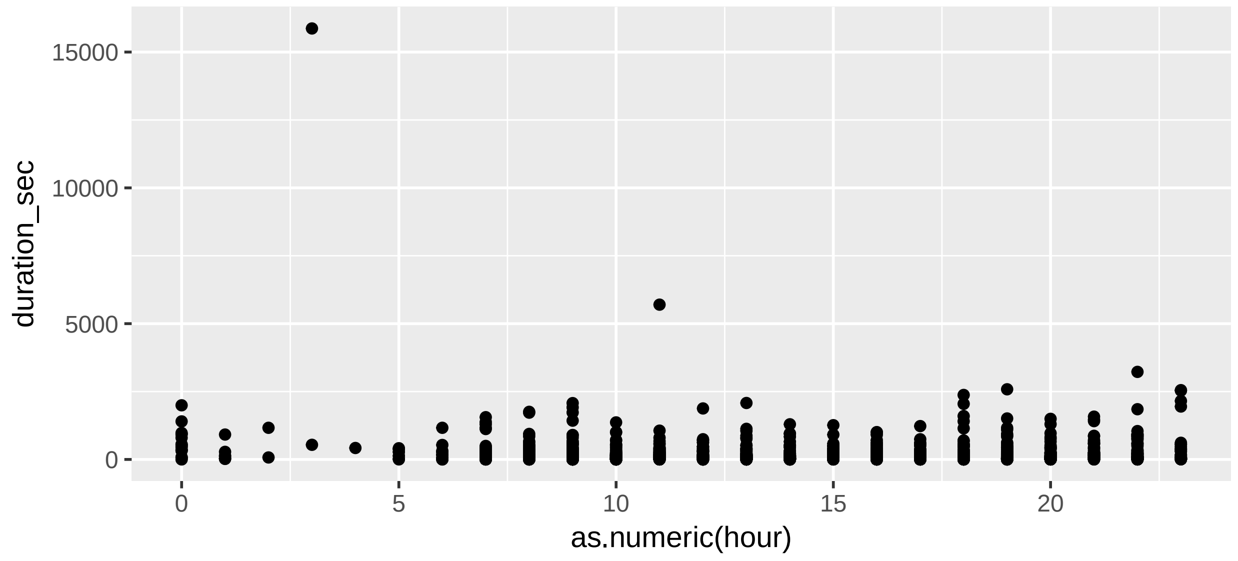


```r
# an empty canvas, ready for creativity!
```

Adding geoms

To this sphere, we can add ***geoms***. The most common geom is a point (cfr. scatterplot)

```r
# this adds points to the sphere, for each row in the dataset
g + geom_point()
```



Other often used geoms are:

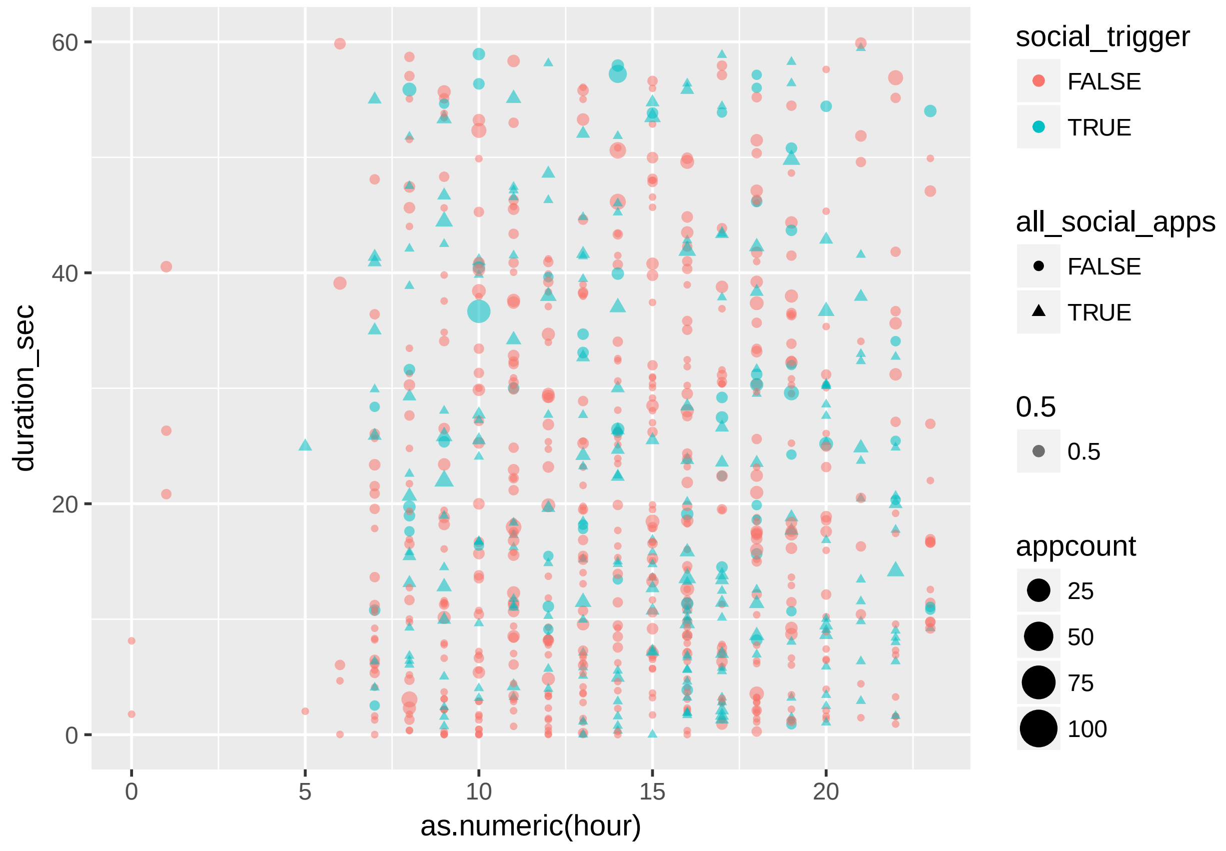
- geom_histogram()
- geom_bar()
- geom_smooth()
- geom_boxplot()
- geom_dotplot()
- geom_violin()
- geom_line()
- geom_errorbar()
- ...

You can also add multiple geoms to the sphere if you want (just use a + between them)

Aesthetics

It becomes even more interesting if we start playing with the ***aesthetics***. For example:

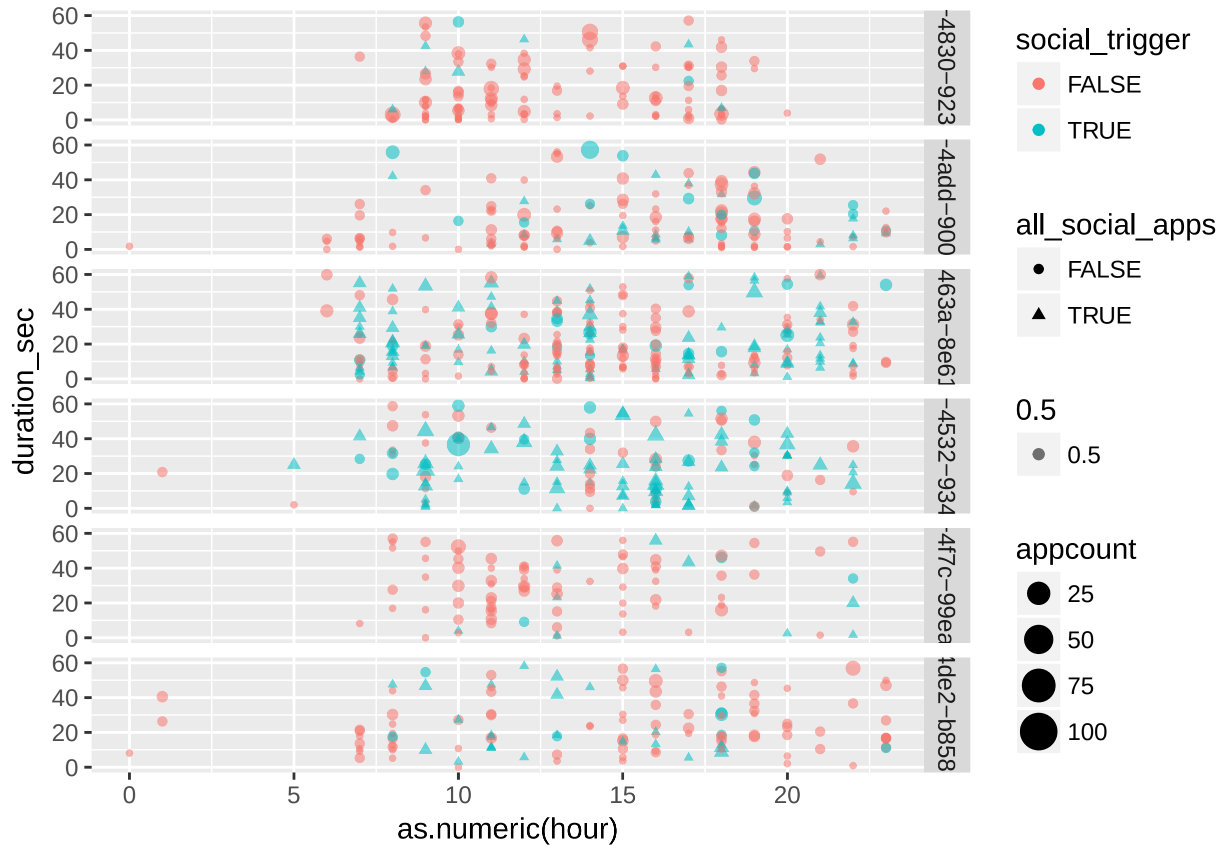
```
g + geom_point(aes(color = social_trigger,
                   size = appcount,
                   alpha = 0.5,
                   shape = all_social_apps)) +
    # remove outliers for the visualization
    scale_y_continuous(limits = c(0, 60))
```



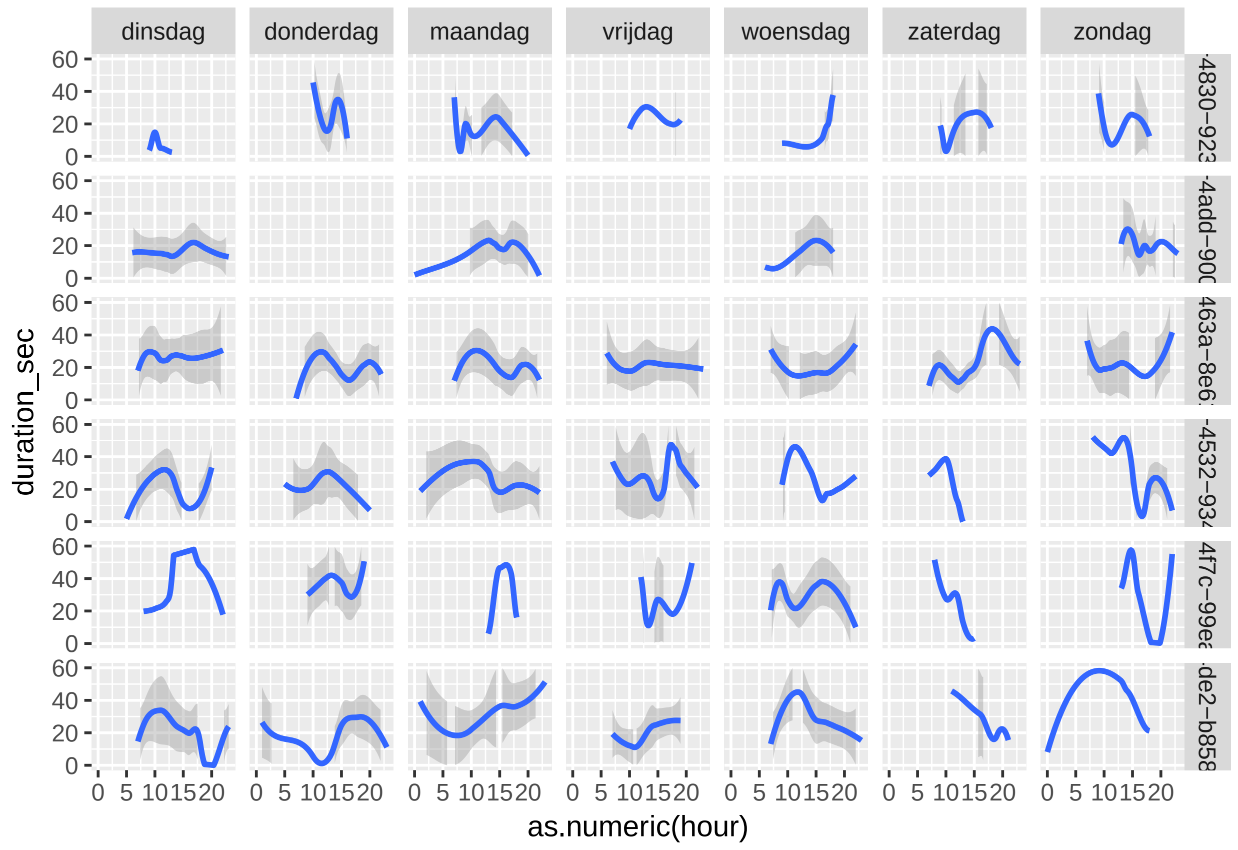
Facets

As you can see, when you add more and more variables, it becomes quite messy. Therefore, it can sometimes be handy to split the graph in subgraphs. This is called ***facetting***. You can use up to two variables to split you viz.

```
# one variable
g + geom_point(aes(color = social_trigger,
                   size = appcount,
                   alpha = 0.5,
                   shape = all_social_apps)) +
    facet_grid(id ~ .) +
    scale_y_continuous(limits = c(0, 60))
```



```r
# two variables
g + geom_smooth() +
    facet_grid(id ~ weekday) +
    scale_y_continuous(limits = c(0, 60))
```



Titles et al.

You can also change the main theme, the title, the axis labels, axis ranges, . . . A good resource is:

- https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Now what?

- ***But can you do this?***
  - Any question on R possibilities and additional processing?
- ***How do you want to proceed***?
  - Extra courses?
  - Recurring meet-ups?
  - User groups?
- ***How can you proceed on your own***?
  - Play & Google (stackoverflow)
  - Coursera: https://www.coursera.org/specializations/jhu-data-science
  - https://www.r-bloggers.com/
- ***What we didn't cover***?
  - 'for' loops
  - 'if' / 'else' structures
  - Merging data
  - Pattern detection
  - Exporting data outside R (csv, images, . . .)
  - Advanced functions
  - APIs for data acquisition
  - Scraping & parsing
  - SQL queries
  - APIs for data annotation (e.g. Microsofts project Oxford)
  - Elasticsearch
  - Advanced statistics
  - Text mining & tidy data
  - NLP & sentiment analysis
  - Network analysis
  - Clean programming
  - Github & collaborative coding
  - Shiny apps
  - Building your own packages
  - Machine learning
  - Qualtrics integration
  - Reading xlsx & SPSS datafiles
  - Integrating R & Python
  - . . .