# Social Ball: An immersive research paradigm to study social ostracism

Author: Erdem Meral & Hannes Rosenbusch

15/07/2022

## Contents

### Social Ball

The social ball game is an updated version of the classic cyberball game used in many scientific studies on social exclusion. Next to improved graphics, controls, and avatar personalization, it offers many new opportunities for experimental designs (see main manuscript).

This file demonstrates how to load, clean, and explore datasets generated with the social ball software. For questions, please contact erdemozanmeral@gmail.com. To download the software, please refer to the supplementary materials of the main manuscript.

We use the following libraries in this demo script:

```
library(psych)
library(plyr)
library(dplyr)
library(ggplot2)
library(pander)
library(scales)
library(RJSONIO)
library(jsonlite)
library(ltm)
```

### Loading the data

Data generated from social ball studies comes in a tabular format. To export the data you can click on the "export" button from the main admin or user interface. Admins can export data from each scenario and users can only export the data for scenarios which they have created a lobby for. The data can be loaded into R with a variety of functions.

```
study_data <- read.csv2("socialball_CM_20210707.csv")
```

## Data dimensions

An individual row in the social ball export describes all the data from a single participant in the study. In the example dataset, we have 4415 participants and 10 variables (columns) that were measured per participants.

```
dim(study_data)
```

```
## [1] 4415   10
```

## Exported variables

Next, we look at the names of the variables (columns) included in our export.

```
names(study_data)
```

```
##  [1] "scenario.id"           "scenario"              "date"
##  [4] "user.id"               "age"                   "sex"
##  [7] "name"                  "avatar"                "play.events"
## [10] "questionnaire.answers"
```

We will explore these variables individually, starting with *scenario.id* and *scenario*.

### Variables: *scenario.id* & *scenario*

*scenario.id* is a numeric indicator variable describing the game mode in which the social ball experiment was played, whereas *scenario* provides the name of the game mode. In the example dataset, participants played a custom 6-person social exclusion scenario with the lobby feature in which the participants only received the ball a few times while the other players tossed it around among themselves. This mode was called "Overgooien > 6 personen" as the study was conducted in the Netherlands. The table below shows that the full dataset also includes runs with other scenarios.

```
study_data %>%
  distinct(scenario.id, scenario) %>%
  pandoc.table()
```

```
##
## ---------------------------------------
##  scenario.id          scenario
## ------------- -------------------------
##       1               Two balls
##
##       2               One ball
##
##       3        Overgooien > 6 personen
##
##       4                 Test
## ---------------------------------------
```

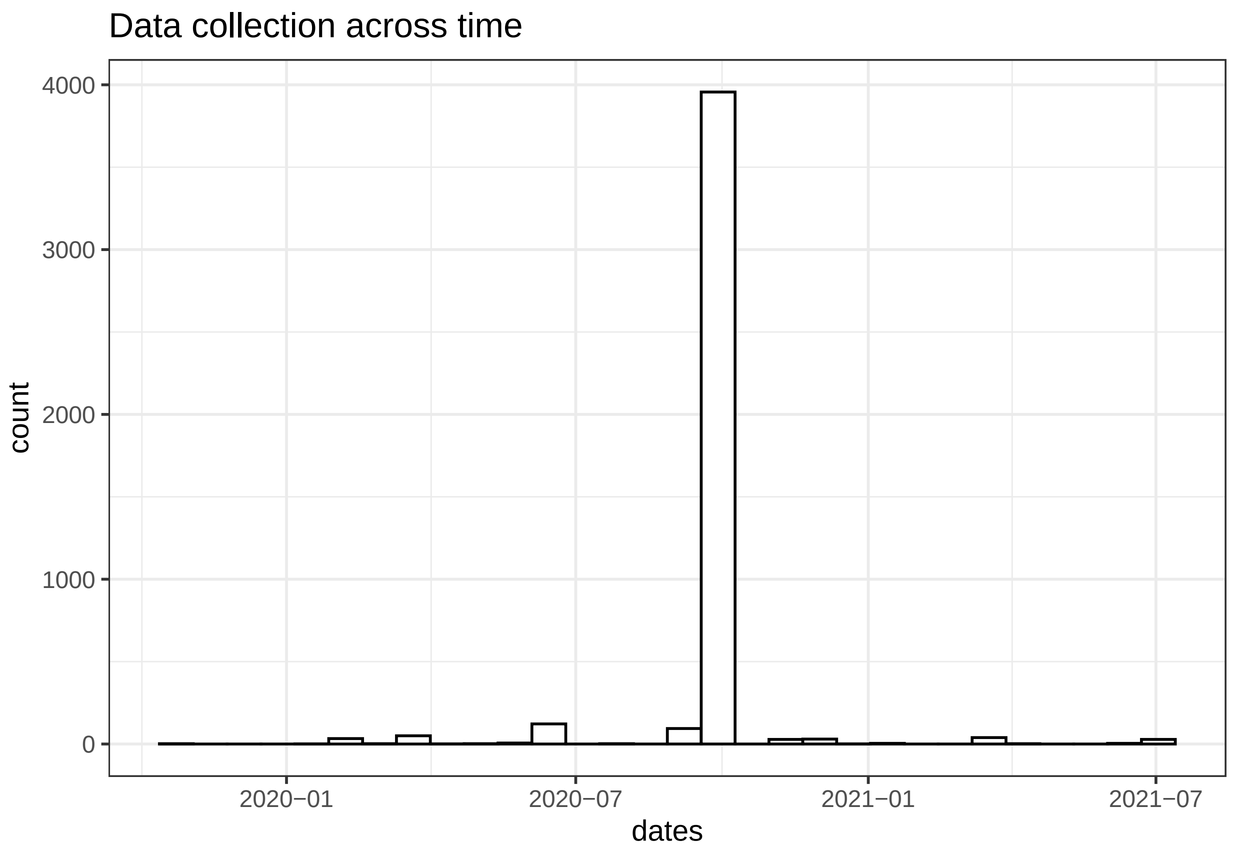
We restrict the data to the target scenario for the current analyses:

```
study_data <- study_data %>%
  filter(scenario == "Overgooien > 6 personen")
```

### Variables: *date*

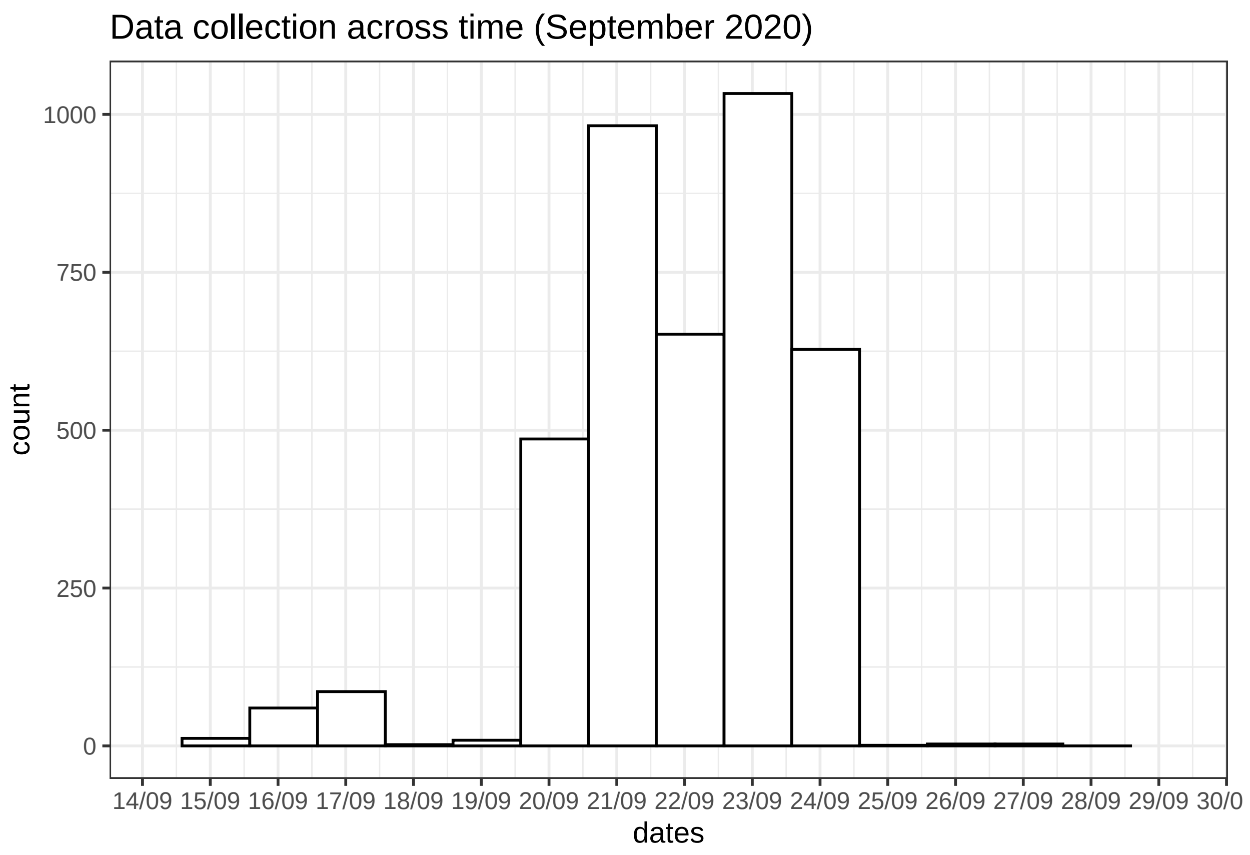
The next varible is *date* and non-surprisingly indicates the date and time when the according participant entered the social ball environment. The following code chunk can be re-used to quickly inspect the dates and times recorded for a study. This simple operation can alert you to anomalies in the data collection process or dropout in longitudinal studies.

```r
study_data$dates <- as.POSIXct(study_data$date,format="%d/%m/%Y %H:%M")
ggplot(study_data) +
  geom_histogram(aes(dates), color = 1, fill = "white") +
  theme_bw() +
  ggtitle("Data collection across time")
```



As can be seen in the plot, the large majority of the observations was collected during a short period of time in the second half of 2020. One can zoom into a specific period (below: second half of September) with the following code.

```r
start <- as.POSIXct("15/09/2020 06:22", format="%d/%m/%Y %H:%M")
end   <- as.POSIXct("30/09/2020 06:22", format="%d/%m/%Y %H:%M")

ggplot(study_data) +
  geom_histogram(aes(dates), binwidth = 1*3600*24, color = 1, fill = "white") +
  ggtitle("Data collection across time (September 2020)") +
  theme_bw() +
  scale_x_datetime(breaks="1 day", labels=date_format("%d/%m"), limits =  c(start, end))
```



Naturally, the date variable can also be used to reduce the total dataset to a specific time period. For instance, if we knew that the analysis relevant data was collected between the 20th and 24th of September 2020 we could select the data with the following code chunk:

```
start <- as.POSIXct("20/09/2020 00:00",format="%d/%m/%Y %H:%M")
end   <- as.POSIXct("24/09/2020 23:59",format="%d/%m/%Y %H:%M")

study_data <- study_data %>%
  filter(between(dates, start, end))
paste(nrow(study_data), "paticipants left")
```

```
## [1] "2578 paticipants left"
```

**Variables: *user.id***

The next variable included in the export is *user.id*. This unique integer value is incrementally assigned to identify each new participant entering the study.

```
glimpse(study_data$user.id)
```

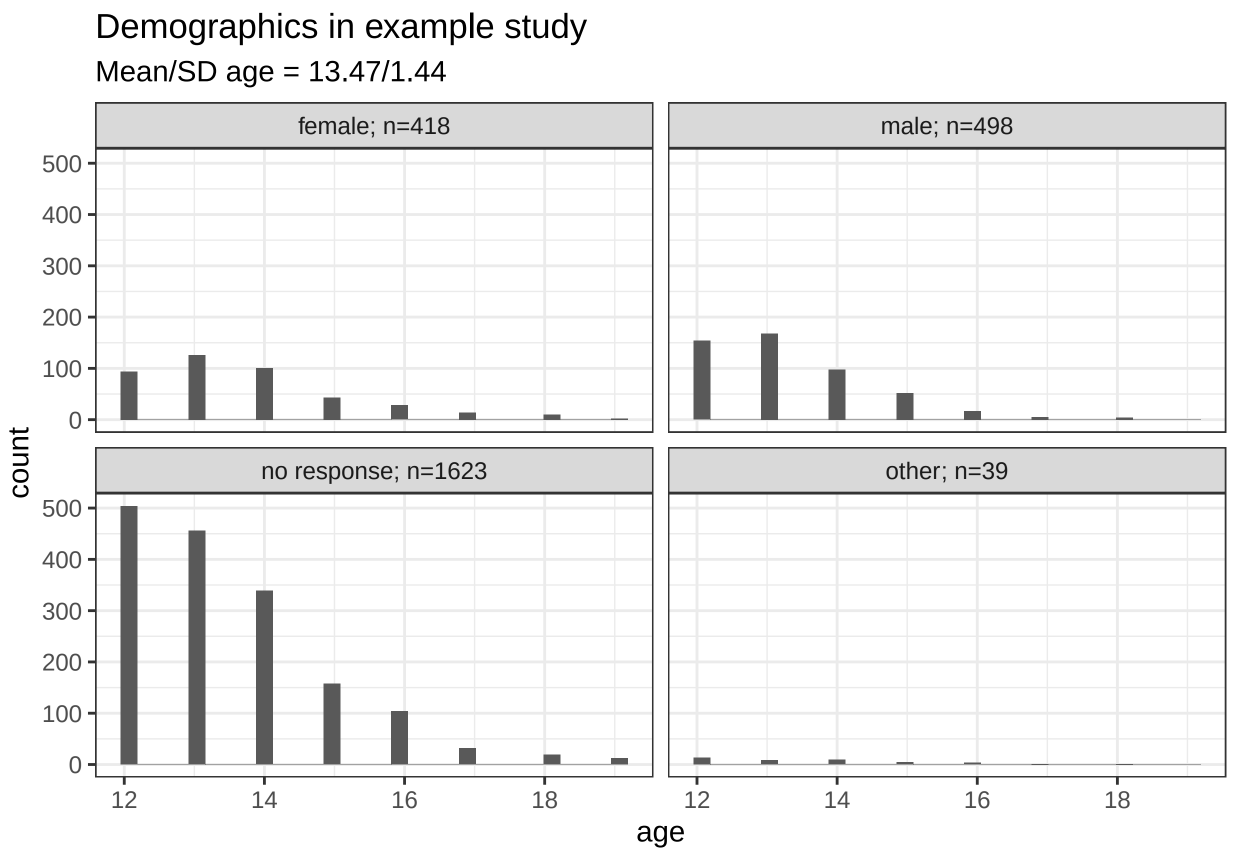
```
##  int [1:2578] 409 412 413 414 415 416 417 418 421 424 ...
```

**Variables: *age* & *sex***

The following two variables are *age* (in years) and *sex*. These variables are alwas collected on the startup screen of the social ball study. In the example (sub-) dataset, these variables are distributed as follows:

```
m_age = round(mean(study_data$age, na.rm = T), 2)
sd_age = round(sd(study_data$age, na.rm = T), 2)
```

```
study_data %>%
  mutate(sex = replace(sex, sex == "", "no response")) %>%
  group_by(sex) %>%
  mutate(group_count = n()) %>%
  ungroup() %>%
  mutate(sex_count = paste0(sex, "; n=", group_count)) %>%
  ggplot() +
    geom_histogram(aes(age)) +
    facet_wrap(~sex_count) +
    theme_bw() +
    ggtitle("Demographics in example study", subtitle = paste0("Mean/SD age = ", m_age,"/", sd_age))
```



As can be seen, the participants were quite young, which is not surprising as the example study took place in a school environment. Also, many participants did not fill out the answer probing for sex. If values are required by the researcher it is of course possible to use non-skippable (or better) questions.

**Variables: *name* & *avatar***

The next value, *name*, is provided by participants at the start of the game and determines their displayed name during the social ball game.

A participant can select the look of their in-game avatar and their choice is recorded in the variable *avatar*. Missing values on this variable indicate cases where participants did not start the game. The eight avatar choices available in the example study are depicted in the screenshot below.

```
study_data %>%
  mutate(avatar = replace(avatar, avatar == "", "missing")) %>%
  ggplot() +
```

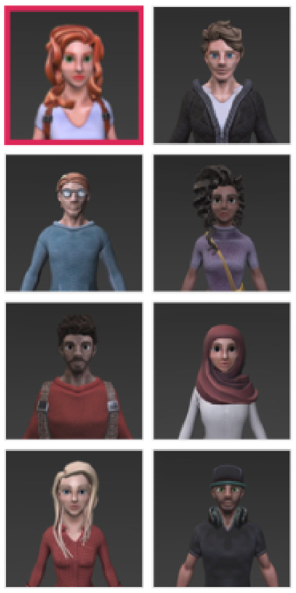
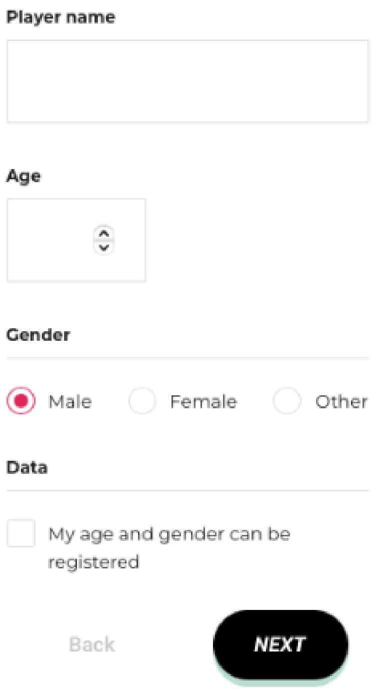


Figure 1: fig1



Player name

Age

Gender

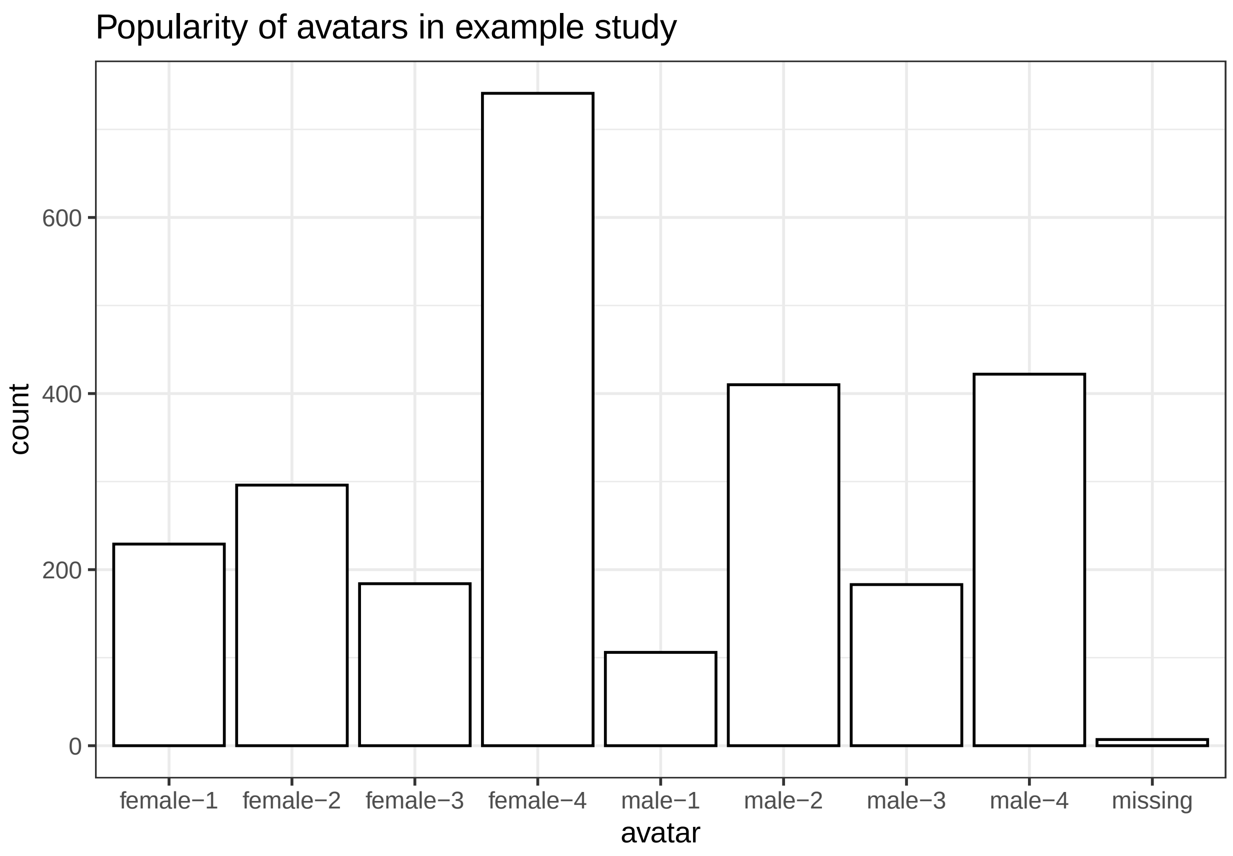
[✓ Male] Female Other

Data

☐ My age and gender can be registered

Figure 2: avatars

```r
  geom_bar(aes(avatar), color = 1, fill = "white") +
  theme_bw() +
  ggtitle("Popularity of avatars in example study")
```



**Variables:** ***play.events***

The two remaining variables in the data export are: *play.events* and *questionnaire.answers*. These variables often include the data, which are of primary interest in social ball studies. The former records the participants' behavior during the social ball game, including ball/object throws/catches, waving to other players, and chat messages. Further, timestamps (in seconds since the game start) are added to the individual behaviors. The format of a single participant's *play.events* data is a long character string, with behaviors delimited by curly braces. We show a truncated snippet of one participant's data below.

```r
TRUNCATE <- 328
first_entry <- paste0(substr(study_data$play.events[300], 1, TRUNCATE), "]")
print(first_entry)
```

```
## [1] "[{\"type\":\"throw\",\"from\":\"0\",\"to\":\"2\",\"ball\":\"Object A: ball\",\"timestamp\":\"18
```

Given that the string has a regular (json) structure, it can be printed out in a more readable format like so:

```r
prettify(first_entry)
```

```
## [
##     {
##         "type": "throw",
##         "from": "0",
##         "to": "2",
##         "ball": "Object A: ball",
##         "timestamp": "18.943"
##     },
##     {
##         "type": "throw",
##         "from": "2",
##         "to": "3",
##         "ball": "Object A: ball",
##         "timestamp": "21.735"
##     },
##     {
##         "type": "throw",
##         "from": "3",
##         "to": "4",
##         "ball": "Object A: ball",
##         "timestamp": "30.991"
##     },
##     {
##         "type": "throw",
##         "from": "4",
##         "to": "5",
##         "ball": "Object A: ball",
##         "timestamp": "34.451"
##     }
## ]
##
```

One can see that the first actions consisted of different players throwing the ball toeach other. The human player is always labelled player 0. If we wanted to, for instance, count the number of times that the human player received the ball or waved, we have to extract that player-specific information from the *play.events* variable. The following-code chunk demonstrates one way of achieving that.

```
all_data_participant1 <- fromJSON(study_data$play.events[3]) %>% as.data.frame
tail(all_data_participant1)
```

```
##     type player timestamp from   to           ball
## 65  wave      1     92.91 <NA> <NA>           <NA>
## 66 throw   <NA>    94.568    5    1 Object A: ball
## 67  wave      3    97.034 <NA> <NA>           <NA>
## 68 throw   <NA>    97.644    1    3 Object A: ball
## 69 throw   <NA>   100.682    3    4 Object A: ball
## 70  wave      1   104.713 <NA> <NA>           <NA>
```

We can also see how many times each of the players threw or received the ball by looking at the columns "from" and "to".

```
cat("Number thows")
pandoc.table((table(all_data_participant1$from)))

cat("\nNumber receptions")
pandoc.table(table(all_data_participant1$to))
```

```
## Number thows
## -----------------------
##  0   1   2   3   4   5
## --- --- --- --- --- ---
##  2   5   5   5   7   6
## -----------------------
##
##
## Number receptions
## -----------------------
##  0   1   2   3   4   5
## --- --- --- --- --- ---
##  1   5   5   5   8   6
## -----------------------
```

The human player was the first one to throw the ball after the game started, explaining why they had two throws in total, despite only receiving it once. Specifically, 23 seconds into the game, player three threw them the ball:

```
all_data_participant1 %>%
  filter(to == "0") %>%
  pandoc.table()
```

```
##
## ----------------------------------------------------------
##  type    player   timestamp   from   to        ball
## ------- -------- ----------- ------ ---- ----------------
##  throw     NA      23.277      3     0    Object A: ball
## ----------------------------------------------------------
```

Further, the column ‘player’ denotes the names of the players engaging in other activities. For the current example, we see that the participant waved to the other players in the game six times within a 20 second window, after not receiving the ball for a while.

```
all_data_participant1 %>%
  filter(player == "0") %>%
  pandoc.table()
```

```
##
## ----------------------------------------------
##  type   player   timestamp   from   to   ball
## ------ -------- ----------- ------ ---- ------
##  wave     0       59.604      NA    NA    NA
##
##  wave     0        61.29      NA    NA    NA
##
##  wave     0       61.526      NA    NA    NA
##
##  wave     0       65.296      NA    NA    NA
##
##  wave     0       69.523      NA    NA    NA
##
##  wave     0       80.509      NA    NA    NA
## ----------------------------------------------
```

Naturally, we can add these in-game statistics as variables to our original dataframe *study_data* in which each row stores the data from one participant. One way to add custom variables is by defining custom functions and applying them over the rows of the dataframe. The following code chunk demonstrates how to do that for the custom variable *waves* indicating how often the participant waved in the game.

```
#custom function for counting waves
count_waves_human <- function(play.events.cell){
  #cannot count waves if no game was played
  if(play.events.cell == "null" | play.events.cell == ""){
    return(NA)
  }

  #transform cell to df
  cell_contents_df <- fromJSON(play.events.cell) %>% as.data.frame

  #if no player waves, the 'player' column will be omitted
  if(! "player" %in% colnames(cell_contents_df)){
    return(0)
  }

  #otherwise simply count the occurrence of the target behavior
  sum(cell_contents_df$type == "wave" & cell_contents_df$player == "0")
}

#count waves from all participants
study_data$waves <- unlist(lapply(study_data$play.events, count_waves_human),
                           use.names = FALSE)

#inspect new waves variable
summary(study_data$waves)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00    4.00   40.00   71.52  108.00  319.00     541
```

We see that there is quite a wide spread in the amount of times participants waved in the game.

### Variables: ***questionnaire.answers***

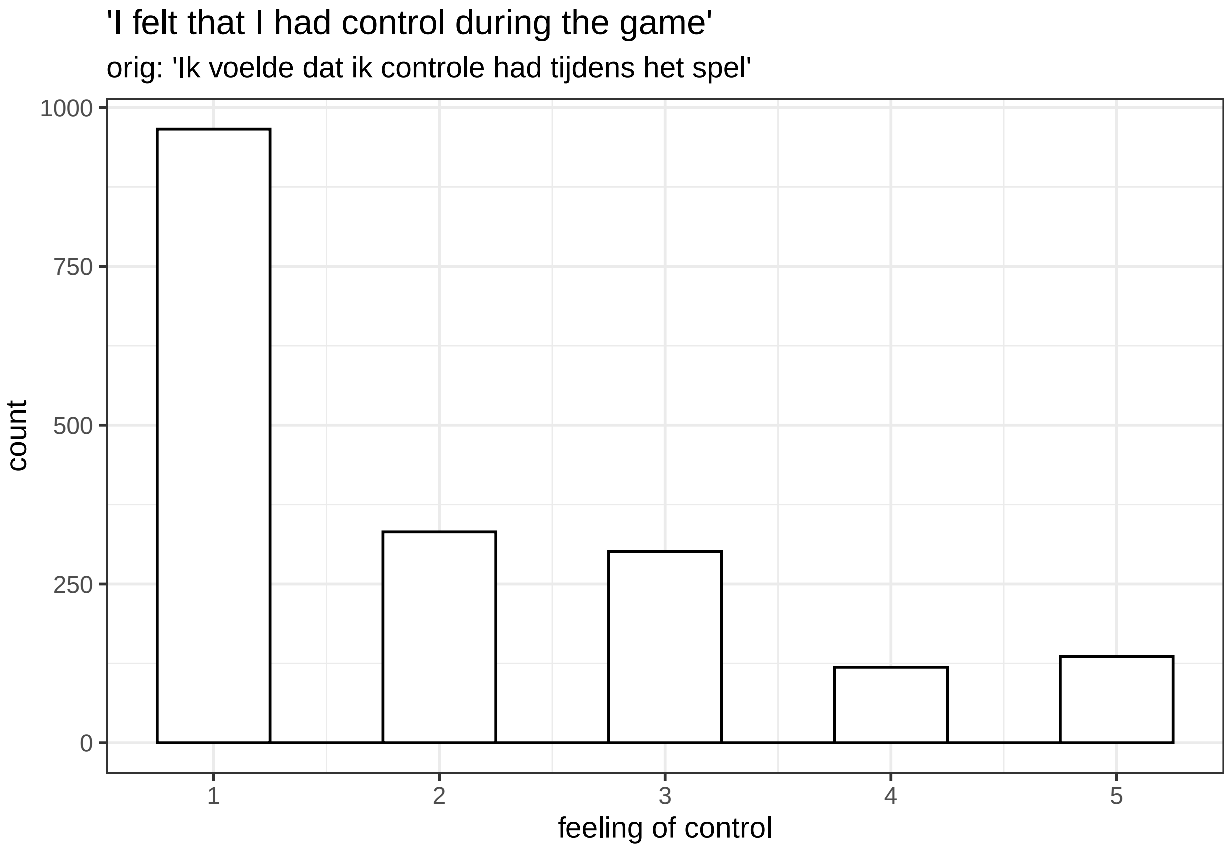
The last variable *questionnaire.answers* also consists of json strings which include all the information from pre- or post- social ball questions, stimuli, or experimental manipulations. Thus, it is very often a key variable in a social ball study. The data can be extracted in the same way as for the *play.events* variable. Below we show one way to extract all the data from all the players and bind them together in a dataframe with the same number of rows as *study_data*.

```
survey_data <- list()

#explicitly iterate over rows
for(i in 1:nrow(study_data)){

  #check if questionnaire data is available for current row
  valid <- isValidJSON(study_data$questionnaire.answers[i], TRUE)

  #add data to list
  if(valid){
    participant_answers <- fromJSON(study_data$questionnaire.answers[i])
    survey_data[[i]] <- as.data.frame(participant_answers)

  #or note missingness
  }else{
    survey_data[[i]] <-data.frame("missing_survey" = TRUE)
  }
```

```r
}

#combine all questionnaire answers into one dataframe (rows = participants; columns = questions)
survey_data <- rbind.fill(survey_data)
```

We could go ahead and combine the social ball data with the questionnaire data by binding the columns together. Below we show how to do so and plot one of the questionnaire variables. Given that the study was run in the Netherlands the original question was asked in Dutch.

```r
all_data <- cbind(study_data, survey_data)
felt_in_control <- as.numeric(all_data$Ik_voelde_dat_ik_controle_had_tijdens_het_spel_)
ggplot() +
  geom_histogram(aes(felt_in_control), binwidth = 0.5, color = 1, fill = "white") +
  theme_bw() +
  labs(x = "feeling of control") +
  ggtitle("'I felt that I had control during the game'",
          subtitle = "orig: 'Ik voelde dat ik controle had tijdens het spel'")
```



The plot shows that the participants largelty felt that they had little control, which is not surprising given that they were largely excluded by the other five players in the game.

## Creating variables

### Need satisfaction score

Many social ball / cyber ball studies (and many social scientific studies in general) assess latent constructs with a multitude of measurements that are combined into a single scale score. In the manuscript, the need satisfaction score is described consisting of perceived control ("Ik voelde dat ik controle had tijdens het spel"), belonging (“Ik voelde dat ik bij de groep hoorde tijdens het spel”), se;f-esteem (“Ik had het gevoel dat de andere spelers mij aardig vonden”), and meaningful existence (“Ik voelde mij zichtbaar tijdens het spel”). Below we compute the cronbach’s alpha for the scale and compute the individual scale scores.

```
items = c("Ik_voelde_mij_zichtbaar_tijdens_het_spel_", "Ik_voelde_dat_ik_controle_had_tijdens_het_spel_

all_data %>%
  dplyr::select(all_of(items)) %>%
  mutate_all(as.numeric) %>%
  cronbach.alpha(., na.rm = T)
```

```
##
## Cronbach's alpha for the '.' data-set
##
## Items: 4
## Sample units: 2578
## alpha: 0.834
```

```
all_data$need_satisfaction = all_data %>%
  dplyr::select(all_of(items)) %>%
  mutate_all(as.numeric) %>%
  rowMeans(., na.rm = T)
```

Additionally, we also calculate the summary statistics for each sub need item separately (table 2 in the manuscript).

```
items = c("Ik_voelde_mij_zichtbaar_tijdens_het_spel_", "Ik_voelde_dat_ik_controle_had_tijdens_het_spel_

all_data %>%
  dplyr::select(all_of(items)) %>%
  mutate_all(as.numeric) %>%
  cronbach.alpha(., na.rm = T)
```

```
##
## Cronbach's alpha for the '.' data-set
##
## Items: 4
## Sample units: 2578
## alpha: 0.834
```

```
all_data$need_satisfaction = all_data %>%
  dplyr::select(all_of(items)) %>%
  mutate_all(as.numeric) %>%
  rowMeans(., na.rm = T)

all_data %>%
  dplyr::select(all_of(items)) %>%
  mutate_all(as.numeric) %>%
  describe(., na.rm = T)
```

```
##                                                              vars    n mean   sd
## Ik_voelde_mij_zichtbaar_tijdens_het_spel_                       1 1854 1.83 1.21
## Ik_voelde_dat_ik_controle_had_tijdens_het_spel_                 2 1854 1.99 1.27
## Ik_voelde_dat_ik_bij_de_groep_hoorde_tijdens_het_spel_          3 1854 1.88 1.17
## Ik_had_het_gevoel_dat_de_andere_spelers_mij_aardig_vonden_      4 1854 2.24 1.33
##                                                              median trimmed  mad
## Ik_voelde_mij_zichtbaar_tijdens_het_spel_                         1    1.59 0.00
## Ik_voelde_dat_ik_controle_had_tijdens_het_spel_                1    1.77 0.00
## Ik_voelde_dat_ik_bij_de_groep_hoorde_tijdens_het_spel_          1    1.67 0.00
## Ik_had_het_gevoel_dat_de_andere_spelers_mij_aardig_vonden_      2    2.07 1.48
##                                                             min max range skew
## Ik_voelde_mij_zichtbaar_tijdens_het_spel_                     1   5     4 1.35
## Ik_voelde_dat_ik_controle_had_tijdens_het_spel_               1   5     4 1.08
## Ik_voelde_dat_ik_bij_de_groep_hoorde_tijdens_het_spel_        1   5     4 1.17
## Ik_had_het_gevoel_dat_de_andere_spelers_mij_aardig_vonden_    1   5     4 0.70
##                                                             kurtosis   se
## Ik_voelde_mij_zichtbaar_tijdens_het_spel_                         0.70 0.03
## Ik_voelde_dat_ik_controle_had_tijdens_het_spel_                   0.02 0.03
## Ik_voelde_dat_ik_bij_de_groep_hoorde_tijdens_het_spel_            0.30 0.03
## Ik_had_het_gevoel_dat_de_andere_spelers_mij_aardig_vonden_       -0.73 0.03
```

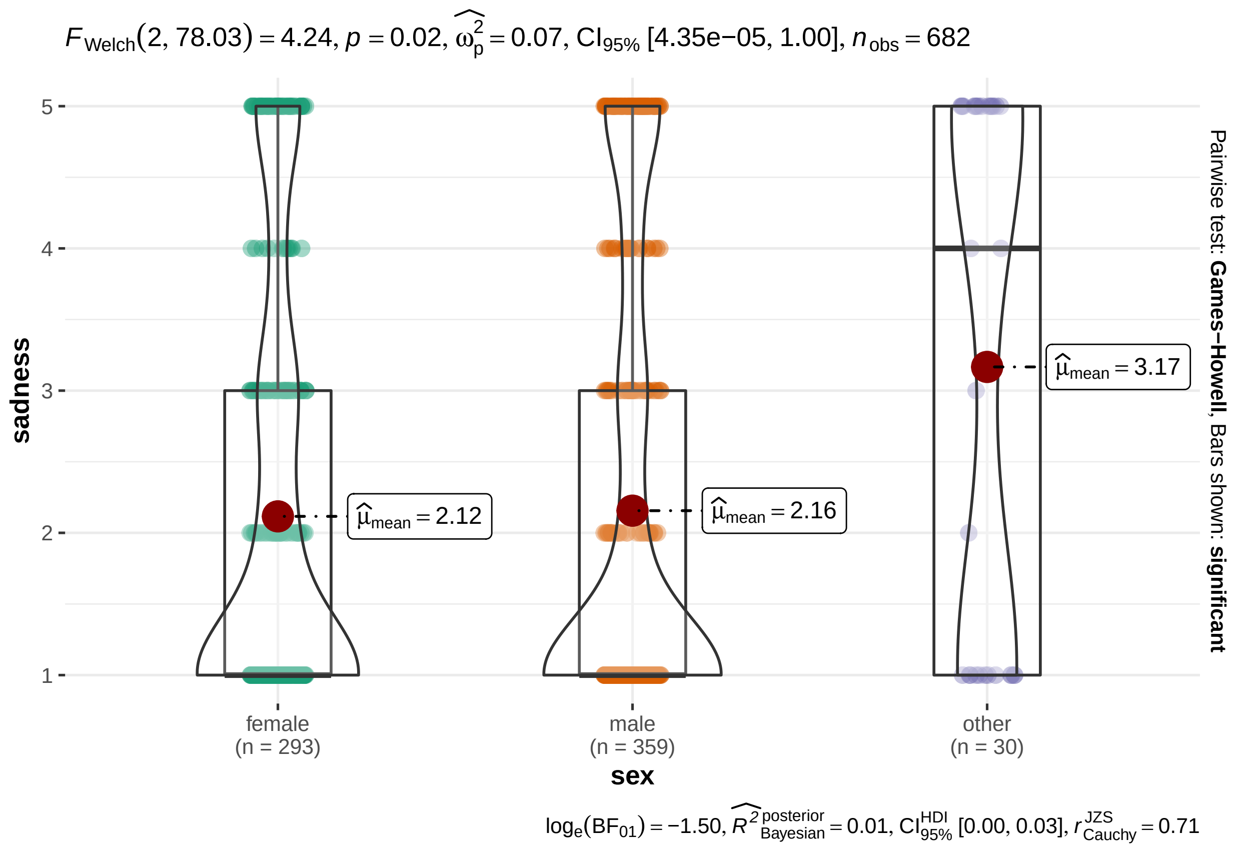
## Example analyses and plots

Below are analyses presentd in the manuscript, as well as examples of common analyses from cyberball/social ball studies.

### Do different participant groups feel differently after the game?

Common analyses in classic Cyberball studies involve the comparison of groups on post-game measurements. A typical example is the comparison of experimental groups regarding their cognitions and emotions. Below we conduct a comparable analysis of male and female participants regarding their reported level of sadness during the exclusion experience.

```
library(ggstatsplot)

plot_data <- all_data %>%
  rename(sadness = Ik_voelde_mij_verdrietig_tijdens_het_spel_) %>%
  mutate(sadness = as.numeric(sadness)) %>%
  dplyr::select(sex, sadness) %>%
  filter(sex != "")

ggbetweenstats(plot_data, sex, sadness)
```



As we can see, male and female participants did not differ on the level of sadness experienced during the game. People identifying with another gender experienced significantly more sadness than the other two groups. Notice however, that the current analysis does not determine the cause of this difference. For this, pre- and post-measurements would be needed which are possible in social ball, but were not conducted for the example study.

### How does participant behavior vary throughout the game?

One advantage of using the social ball software is being able to examine differences between participants in respect to their in-game behavior. For instance, we could visually investigate how participants differ in their waving behavior across time.

First, let's bring the timestamps of the participants' in-game waves into a nice format.

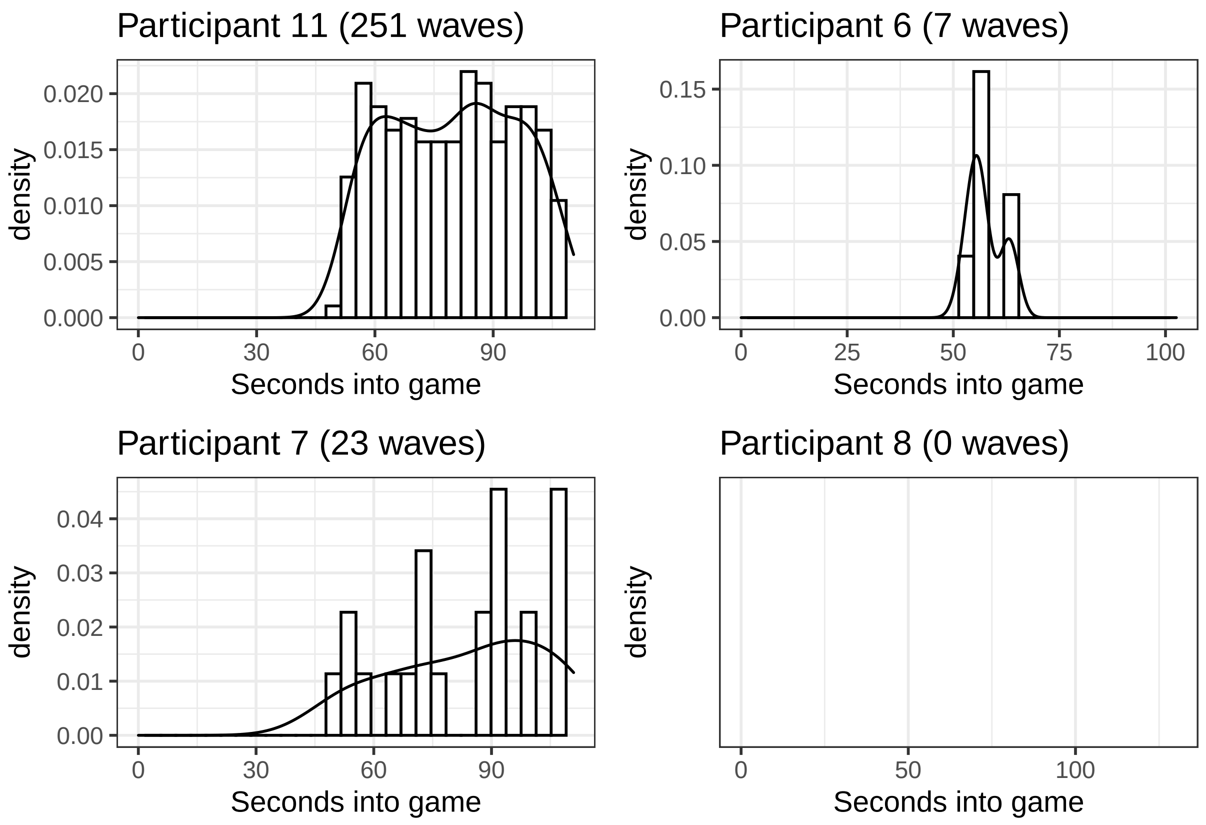
```r
#custom function for extracting timepoints
get_wave_timepoints <- function(play.events.cell){

  #cannot get timepoints if no game was played
  if(play.events.cell == "null" | play.events.cell == ""){
    return(NA)
  }

  #transform cell to df
  cell_contents <- fromJSON(play.events.cell) %>% as.data.frame

  #if no player waves, the 'player' column will be omitted
  if(! "player" %in% colnames(cell_contents)){
    return(0)
  }

  #otherwise simply count the occurrence of the target behavior
  timepoints <- cell_contents$timestamp[cell_contents$type == "wave" &
                                          cell_contents$player == "0"]
  as.numeric(timepoints)
}

all_timepoints <- lapply(study_data$play.events, get_wave_timepoints)
```

Now we have a list, *all_timepoints*, with 2578 elements. Each element is a vector of timepoints and every participant has one vector associated with them. It is also useful to record the endpoint of each game marked by the last activity.

```r
get_endpoints <- function(play.events.cell){
  if(play.events.cell == "null" | play.events.cell == ""){
    return(NA)}
  cell_contents <- fromJSON(play.events.cell) %>% as.data.frame
  endpoint <- cell_contents$timestamp[nrow(cell_contents)]
  as.numeric(endpoint)
}
all_endpoints <- lapply(study_data$play.events, get_endpoints)
```

Now that we have the timepoints of the start (0 seconds), the end, and the waves, we can look at snapshots of some individuals' waving behavior like so:

```r
library(lattice)
library(patchwork)
make_plot<- function(index){
  my_plot <- ggplot(NULL, aes(x = all_timepoints[[index]], y = ..density..)) +
          xlim(c(0, all_endpoints[[index]])) +
          geom_histogram(colour = 1, fill = "white") +
          geom_density() +
          labs(x = "Seconds into game") +
          theme_bw() +
          ggtitle(paste0("Participant ", index, " (", length(all_timepoints[[index]]), " waves)"))
return(my_plot)
}
selected_participants <- c(11,6,7,8)
my_plots = lapply(selected_participants, make_plot)
wrap_plots(my_plots)
```

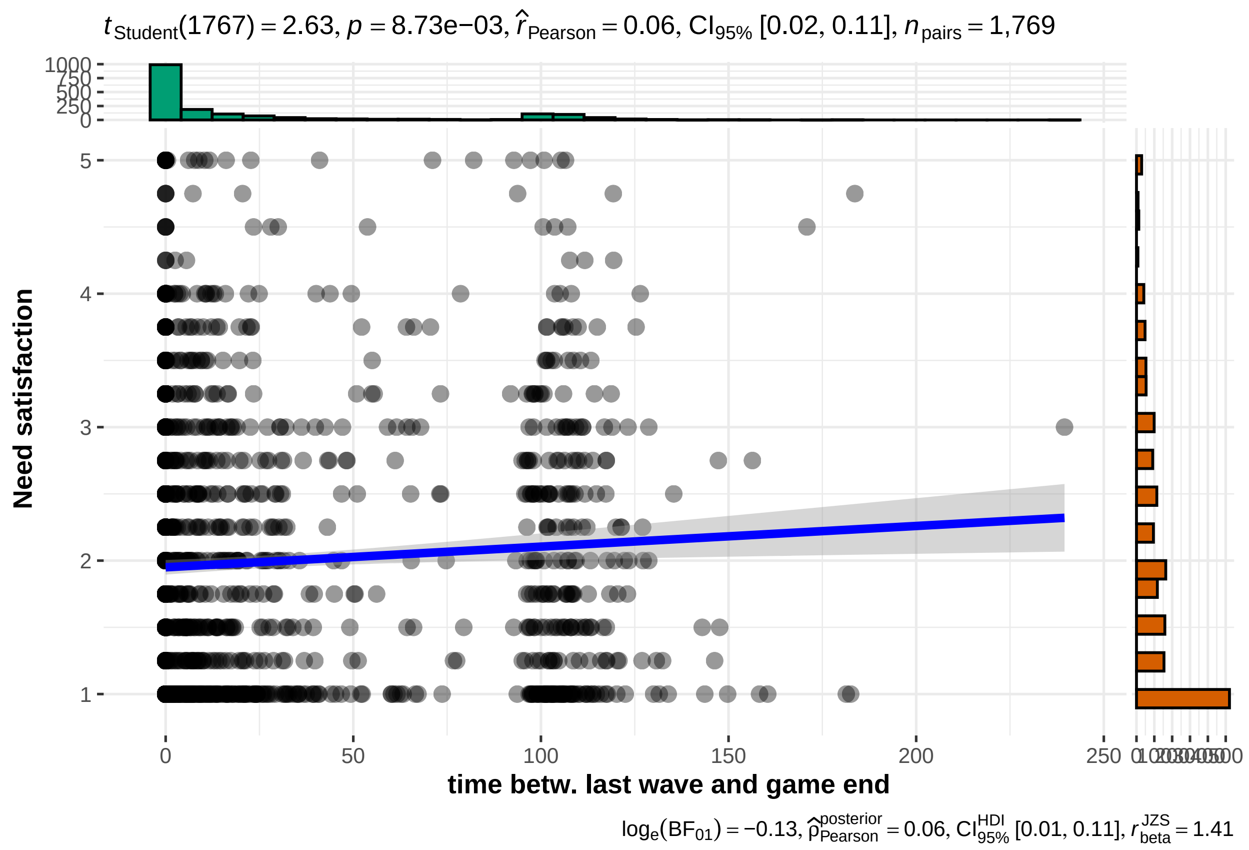


We see that participants showed very different patterns of waving behavior over the trajectory of the game.

**Does in-game "giving-up" predict post-game emotions?**

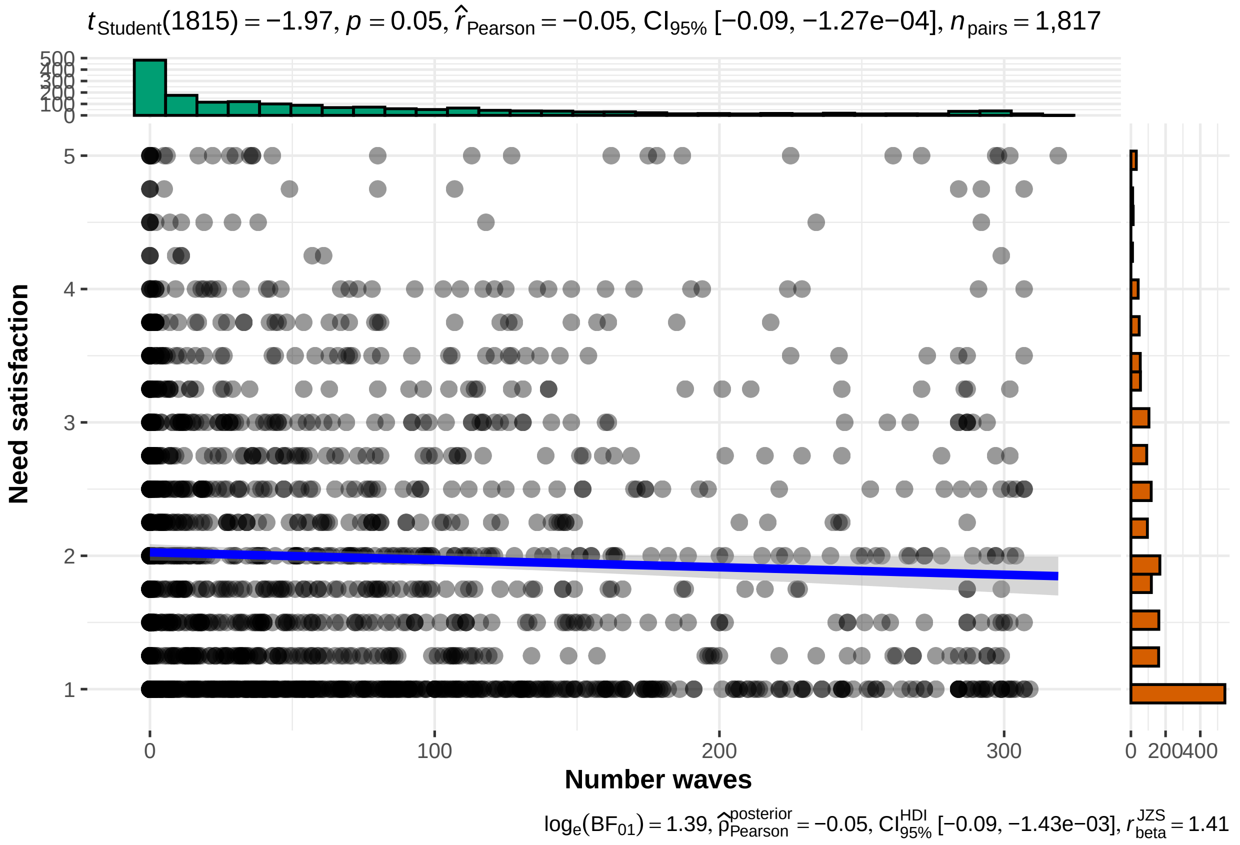
We can now extract information from each individuals' behavior and relate it to post-game measurements. For instance, one could hypothesize that stopping to wave before the end of the game could predict how the participant subsequently scores on the need satisfaction scale.

```
last_waves <- sapply(all_timepoints, max)
all_data$times_btw_wave_and_end <- unlist(all_endpoints) - last_waves
plot_data <- all_data %>% filter(!is.na(need_satisfaction) &
                                 !is.infinite(need_satisfaction) &
                                 !is.na(times_btw_wave_and_end) &
                                 !is.infinite(times_btw_wave_and_end))
ggscatterstats(plot_data, times_btw_wave_and_end, need_satisfaction,
               ggplot.component  = ggplot2::labs(x = "time betw. last wave and game end",
                                                 y = "Need satisfaction"))
```



Both the frequentist and the Bayesian interval locate the likely correlation coefficient somewhere between 0.02 and 0.11. This is actually contrary to my personal a priori hypothesis that people who stopped waving long before the end, would score lower on need satisfaction. Altogether, it seems like the two measures only correlate to a very small degree. Stopping to wave should certainly not be used as a behavioral indicator of people's need satisfaction. Similarly, there does not seem to be a strong association between the number of waves during the game and need satisfaction:

```
plot_data <- all_data %>%
  filter(!is.na(need_satisfaction) &
         !is.infinite(need_satisfaction) &
         !is.na(waves) &
         !is.infinite(waves))
ggscatterstats(plot_data, waves, need_satisfaction,
               ggplot.component  = ggplot2::labs(x = "Number waves",
                                                 y = "Need satisfaction"))
```



Notice, that the analyses here have been conducted primarily for tutorial purposes. If there are questions, please reach us under the contact information provided toward the top of this file :)