transitiveInference

Petra Borovska

21 June 2020

General Set Up

clean environment and console

```
cat("\014")
```

```
rm(list = ls())
```

Installing all necessary packages for data preparation

Data Preparation

Data preparation, loding data set, basic adjustments

get current directory

```
here()
```

```
## [1] "C:/Users/ibm/Documents/Results/R"
```

list the files in the desired direcotry

```
here("dataSets")
```

```
## [1] "C:/Users/ibm/Documents/Results/R/dataSets"
```

```
list.files(here("dataSets"))  ## just double check if it's correctly saved
```

```
## [1] "pilotA_sleep_tr.csv"  "pilotA_sleep_tr.xlsx" "pilotA_sleep_ts.csv"
## [4] "pilotA_sleep_ts.xlsx" "pilotA_wake_tr.csv"   "pilotA_wake_tr.xlsx"
## [7] "pilotA_wake_ts.csv"   "pilotA_wake_ts.xlsx"
```

load the data for each data set

```
tr_sleep <-
        read.csv(
                here(
                        "dataSets",
                        "pilotA_sleep_tr.csv"
                )
        )

tr_wake <-
        read.csv(
                here(
                        "dataSets",
                        "pilotA_wake_tr.csv"
                )
        )


test_sleep <-
        read.csv(
                here(
                        "dataSets",
                        "pilotA_sleep_ts.csv"
                )
        )

test_wake <-
        read.csv(
                here(
                        "dataSets",
                        "pilotA_wake_ts.csv"
                )
        )
```

look at the data

```
## show the data
# tr_sleep
# tr_wake
# test_sleep
# test_wake
```

clean data, delete not useful rows from each data set Drop na only for testing, where the length of tibble is the same, not do for training, it varies and it will drop also useful rows

```
test_sleep = test_sleep %>% drop_na()
test_wake = test_wake %>% drop_na()
```

```
ts_sleep_R = test_sleep
ts_wake_R = test_wake
```

test_wake with the most data, I guess use that need to get average from each pair then I need to switch participants columns with rows

my columns are always:
pairType _condition_token letterPos1 _condition_token letterPos2 _condition_token key_resp_test.corr _condition_token <- that might be different name with training key_resp_test.rt _condition_token <- also might be different for training

to calculate frequencies - total scores: premise 50 1deg 20 2deg 10 anchor 10

```
premTot = 50
oneDegTot = 20
twoDegTot = 10
anchorTot = 10
```

```
colName_w_ts =
        ts_wake_R %>%
        colnames(.)
colNum_w_ts =
        ts_wake_R %>%
        ncol()
```

```
# ts_wake_6_select =
#         ts_wake_R %>%
#         select(
#                 c(1:5)
#         )
#
#
# ts_wake_6_multiple =
#         ts_wake_6_select %>%
#         group_by(
#                 pairType_W_6
#         ) %>%
#         summarize(
#                 tibble(
#                 total_W_6 = sum(key_resp_test.corr_W_6),
#                 rt_mean_W_6 = mean(key_resp_test.rt_W_6)
#                 )
#         ) %>%
#         add_column(max_W_6 = c(10, 20, 50, 10))
```

Functions

write a funtion which gives me summary dfs, containing total number per pair type, mean rt per pair type and total number of pair types per group

parameters that needs to be inserted: type of data set which columns from the data set, 5 columns for each participant - maybe create a range variables for each participant

does not have to be insert as parameter, but need to be set up
vary pair type - it's always first column of new data set, so replace the column name it does not have to vary proportion column, again replace by number - it does not have to vary mean column - again replace by number - it does not have to vary

think about that when training - na s must be drop during the function call - when selection going on - so between step

For the function is good to make copy

```
summary_subj_ts_f <- function(dataSet, s, e){

        s = as.integer(s)
        e = as.integer(e)

        dataSet_select = dataSet[, c(s:e)]
        dataSet_select = as_tibble(dataSet_select)


        origColNames = colnames(dataSet_select)[c(1, 4, 5)]

        colnames(dataSet_select) <- c("pairType", "letterPos1", "letterPos2",
                                      "key_resp_test.corr", "key_resp_test.rt")

        dataSet_multiple =
                dataSet_select %>%
                group_by(
                        pairType
                ) %>%
                summarize(
                        tibble(
                        total_corr = sum(key_resp_test.corr),
                        mean_rt = mean(key_resp_test.rt)
                        )
                )

        colnames(dataSet_multiple) <- origColNames


        dataSet_multiple_t =
                dataSet_multiple %>%
                gather(key = corr_cond_subj, value = value, 2:ncol(dataSet_multiple)) %>%
                spread_(key = names(dataSet_multiple)[1],value = 'value')


        dataSet_multiple_t_head = slice_head(dataSet_multiple_t)
        dataSet_multiple_t_tail = slice_tail(dataSet_multiple_t)
        dataSet_multiple_t_less =
                dataSet_multiple_t_tail %>%
                select(-1) %>%
                rename(anchor_rt = anchor, oneDegree_rt = oneDegree,
                       premise_rt = premise, twoDegree_rt = twoDegree)

        oneRowDf = cbind(dataSet_multiple_t_head, dataSet_multiple_t_less)

        return(oneRowDf)
}
```

Further Data Processing

Appliying the function. First getting number of columns for each data set, so I know how many participants we have - total / 5.

Also getting name of the columns, so I know which tokens used = subject number.

```
noCol_ts_W = ncol(ts_wake_R)
noCol_ts_S = ncol(ts_sleep_R)

noCol_ts_W    ## 20 columns / 5 = number of participants
```

```
## [1] 20
```

```
noCol_ts_S    ## 10 columns / 5 = number of participants
```

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

```
namesCol_ts_W = colnames(ts_wake_R)
namesCol_ts_S = colnames(ts_sleep_R)

namesCol_ts_W
```

```
##  [1] "pairType_W_6"            "letterPos1_W_6"
##  [3] "letterPos2_W_6"          "key_resp_test.corr_W_6"
##  [5] "key_resp_test.rt_W_6"    "pairType_W_9"
##  [7] "letterPos1_W_9"          "letterPos2_W_9"
##  [9] "key_resp_test.corr_W_9"  "key_resp_test.rt_W_9"
## [11] "pairType_W_11"           "letterPos1_W_11"
## [13] "letterPos2_W_11"         "key_resp_test.corr_W_11"
## [15] "key_resp_test.rt_W_11"   "pairType_W_15"
## [17] "letterPos1_W_15"         "letterPos2_W_15"
## [19] "key_resp_test.corr_W_15" "key_resp_test.rt_W_15"
```

```
#  [1] "pairType_W_6"            "letterPos1_W_6"          "letterPos2_W_6"
#  [4] "key_resp_test.corr_W_6"  "key_resp_test.rt_W_6"    "pairType_W_9"
#  [7] "letterPos1_W_9"          "letterPos2_W_9"          "key_resp_test.corr_W_9"
# [10] "key_resp_test.rt_W_9"    "pairType_W_11"           "letterPos1_W_11"
# [13] "letterPos2_W_11"         "key_resp_test.corr_W_11" "key_resp_test.rt_W_11"
# [16] "pairType_W_15"           "letterPos1_W_15"         "letterPos2_W_15"
# [19] "key_resp_test.corr_W_15" "key_resp_test.rt_W_15"
#

# tokens_W = 6, 9, 11, 15

namesCol_ts_S
```

```
##  [1] "pairType_S_7"           "letterPos1_S_7"         "letterPos2_S_7"
##  [4] "key_resp_test.corr_S_7" "key_resp_test.rt_S_7"   "pairType_S_8"
##  [7] "letterPos1_S_8"         "letterPos2_S_8"         "key_resp_test.corr_S_8"
## [10] "key_resp_test.rt_S_8"
```

```
#  [1] "pairType_S_7"           "letterPos1_S_7"         "letterPos2_S_7"
#  [4] "key_resp_test.corr_S_7" "key_resp_test.rt_S_7"   "pairType_S_8"
#  [7] "letterPos1_S_8"         "letterPos2_S_8"         "key_resp_test.corr_S_8"
# [10] "key_resp_test.rt_S_8"

# tokens_S = 7, 8
```

Creating a df out of the function. Gives me total of corr responses per pair type and average rt per pair type. Then make a proportion out of it and combine those df, into one bigger df. Consider to write another function.

```
# ## wake testing
# subj_6 = summary_subj_ts_f(ts_wake_R, 1, 5)
# subj_9 = summary_subj_ts_f(ts_wake_R, 6, 10)
# subj_11 = summary_subj_ts_f(ts_wake_R, 11, 15)
# subj_15 = summary_subj_ts_f(ts_wake_R, 16, 20)
#
# ## sleep testing
# subj_7 = summary_subj_ts_f(ts_sleep_R, 1, 5)
# subj_8 = summary_subj_ts_f(ts_sleep_R, 6, 10)
```

Now create a tibble out of those single outputs. First transpose rows and columns. <- that's already in the function Then add together. <- also in the function

```
# df_ts = rbind(subj_6, subj_9, subj_11, subj_15, subj_7, subj_8)
```
