Data Rt

Syntax, tips and tricks to work in R

Joachim Waterschoot*

Latest update - 08 December 2022

Contents

*Department of Developmental, Personality and Social Psychology, Ghent University, joachim.waterschoot@ugent.be

1 Get R and youR data Ready

1.1 Get ready

To work with R, you need packages, including useful functions. By **installing** these packages on your computer, you can make use of these functions. Indeed, they are already written out for you and you install the package so you can make use of them.

At each start of a new R session (e.g., after shutting down your computer), you have to load the packages you want to use. Instead of using `library(X)` for each package, you can make a list so they can be loaded all at once.

Following packages are required for (1), (2) and (3):

```
# my list of packages
x<-c('readxl','haven','psych','sjPlot','writexl','foreign')

# do this only once
install.packages(x)

# do this at the start of each session
lapply(x, require, character.only = TRUE)

# use this for only one package
library(PACKAGE_NAME)
```

Also important is that R knowns in which folder you want to work. By following code, you set your work directory in which R can find the required datasets and also can save the files you want to save.

```
setwd("/Users/joachimwaterschoot/Downloads/Analyses")
```

1.2 Welcome, data!

Datasets can be loaded in every format. Beware, these functions also include more options (e.g., a specific tab in an excel file). More information can be found on the page of the packages (search for the function in Google)

```
df <- readRDS('FILE.rds')
df <- read_xlsx('FILE.xlsx')
df <- read.spss("FILE.sav",
                use.value.labels = FALSE,
                to.data.frame=TRUE)
```

After data is loaded, you can check whether it occurred correctly. Do this by checking the column names and the dimension (number of rows X number of columns).

```
names(df)
dim(df)
head(df,10)
View(df)
View(head(dffull,10)) # check first 10 rows
```

2 WoRk with the dataset

2.1 Preparatory work

To reverse a numeric value or to change specific values to another value in a column, you can use the recode function:

```
library(dplyr)
df$item_r <- as.factor(recode(df$item, '1'='5', '2'='4','3'='3','4'='2','5'='1'))
```

When it is only a numeric columns, a trick to be used is to subtract that column from the highest value + 1. Here, a 1-5 scale is reversed by subtracting it from 6. By doing this, 5 becomes 1, 4 becomes 2, etc.

```
df$item_r <- 6-as.numeric(df$item)
```

When you want to add information to a dataset (e.g., df), based on another dataset (e.g., df_other), you can use the `match()` function to first match values of the same variable in the same datasets (e.g., participation number).

```
df$VARIABLE1 <- df_other$VARIABLE2[match(df$ID,df_other$df)]
```

Rename only one column

```
names(df)[names(df) == 'old.var.name'] <- 'new.var.name'
```

2.2 Putting variables in the right format

Importantly, you want to have variables in the right format. This is important before starting analyses.

```
# formatting into numeric variable
df$VARIABLE1 <- as.numeric(df$VARIABLE1)

# formatting into a categorical variable
df$VARIABLE2 <- as.factor(df$VARIABLE2)
# check the levels
levels(df$VARIABLE2)
# provide different labels to the levels
levels(df$VARIABLE2) <- c('LEVEL1','LEVEL2')
```

2.3 Make variables

In the `keys.list`, you make an overview of which items belong to which variable. Do this for only those containing at least 2 items or more.

```
keys.list <- list(
  VARIABLE1 = c("ITEM1","ITEM2"),
  VARIABLE2 = c("ITEM1","ITEM2")
)
```

This list will be used in the following code to calculate your variables and to add them in the described order to the dataset. You only have to run this.

! Beware: `df` refers to the name of your dataset

```
columns <- unlist(keys.list, use.names=FALSE)
scaleitems <- df[,columns]
scaleitems <- sapply(scaleitems, as.numeric)
df <- df[, ! names(df) %in% columns, drop = F]
```

```
keys <- make.keys(scaleitems,keys.list)
scores <- psych::scoreFast(keys, scaleitems, impute="none")

means <- as.data.frame(scores)
colnames(means) <- sub("-A.*", "", colnames(means))

df <- cbind(df, scaleitems,means)
```

The `sjt.itemanalysis` function provide a nice overview of the internal consistencies of your variables.

- overview of items
- percentage of missing values
- standard deviation
- skewness: the higher, the skewer
- item difficulty: should range between .20 and .80. Ideal value is `p+(1-p)/2` (mostly between .50 and .80)
- item discrimination: acceptable cut-off of .20. The closer to 1, the better.
- Cronbach's Alpha if item was removed from scale
- mean (or average) inter-item correlation: acceptable between .20 and .40
- Cronbach's Alpha: acceptable cut-off of .70

```
sjt.itemanalysis(df[,c(keys.list$VARIABLE1)])
sjt.itemanalysis(df[,c(keys.list$VARIABLE2)])
```

3 Save a dataset

```
saveRDS(df,'NAME.rds') # R format
write_csv(df,'NAME.csv') # csv format
write_xlsx(df,'NAME.xlsx') # excel format
write_sav(df,'NAME.sav') # SPSS format
```

4 Power

A very nice online tool to use is https://webpower.psychstat.org/wiki/models/index

5 Correlations (num x num)

5.1 Cross-sectional

The following packages are required:

```
library(devtools)
# install_github('watjoa/CaviR') # in case you have not installed the package yet
library(CaviR)
library(qgraph)
```

As no packages were available providing a clean output of a correlation table, including descriptive statistics, I function was constructed in the `CaviR` package. By using this, a clean table will appear which can be copied wright away in excel, word, etc.

```
# make a subset of the dataframe 'cordf' including the preferred variables
cordf <- df[,c('VARIABLE1','VARIABLE2')]
# run the correlation table
coRtable(cordf)
```

Make a useful netwerk plot based on the variables in the correlation table.

```
corMat <- cor(cordf, use = "pairwise.complete.obs") # Correlate data
Graph_lasso <- qgraph(corMat, graph = "glasso",
                      layout = "spring", tuning = 0.25,
                      labels = colnames(corMat),
                      sampleSize = nrow(cordf))
```

5.2 Multilevel

The same `CaviR` package can be used for multilevel correlations (up until **2 levels**). The upper diagonal will present the correlations within levels of the grouping variable. The lower diagonal will do this across groups.

```
dataset <- df[,c('VARIABLE1','VARIABLE2', 'NummerP')]
group <- 'NummerP'

multicoR(dataset, group=group)
```

6 Frequencies (cat x cat)

Following packages are required:

```
x<-c('sjPlot')
lapply(x, require, character.only = TRUE)
```

When you want to check whether, for instance, more male are present in a specific group, you can get a clean contingency table with a chi-square test by following code. Of course, you can work with all the different options as you wish:

```
tab_xtab(df$VAR1,df$VAR2,
         show.cell.prc = FALSE,
         show.row.prc = TRUE,
         show.col.prc = FALSE,
         show.legend = TRUE,
         show.na = FALSE,
         show.summary=TRUE)
```

7 MANOVA (num x cat)

Following packages are required:

```
x<-c('stats','effectsize','emmeans','psych','multcomp')
lapply(x, require, character.only = TRUE)
```

You need to list all outcomes you want to have involved in your MANOVA (`OUTCOME1, OUTCOME2`), together with the group variables (`VAR1`). Also, covariates can be added by doing `VAR1 + COVAR1`. Here, we specified to have the Wilk's Lambda value in our MANOVA. This can be changed (see specific options of the function online). The `eta_squared()` function provides effect sizes.

```
root.manova <- manova(cbind(OUTCOME1,OUTCOME2) ~ VAR1, data = df)
summary(root.manova,test="Wilks")
summary.aov(root.manova)
eta_squared(root.manova)
```

When having significant effects, you probably want to check the descriptive differences. The `describeBy` function can do this for multiple variables `OUTCOME1 + OUTCOME2` and even multiple grouping variables `VAR1 + VAR2`. The function contains more options (see online information).

```
describeBy(OUTCOME1 + OUTCOME2 ~ VAR1 + VAR2,
           mat=TRUE,type=3,digits=2,
           data = df)
```

When having more than 2 levels in your grouping variable, you probably want to perform multiple comparison post-hoc analyses to check how all levels differ from each other in terms of significance. The `emmeans` function is really useful and also provides the comparison based on letters.

```
cld(emmeans(lm(OUTCOME1~VAR1,data=df),
            list(pairwise ~ VAR1), adjust = "tukey"),
    alpha=0.05,Letters=letters,adjust="tukey")
```

8 Regression modelling

Following packages are required:

```
x<-c('sjPlot','lme4','lmerTest','performance','effectsize',
     # for visualizations
     'reghelper','interactions','ggeffects','ggplot2')
lapply(x, require, character.only = TRUE)
```

Before building our model, we need to make sure that all our input is ok.
For instance, we want to have dummy or effect codings instead of factor variables. Here, we work with the example of a factor with 3 levels, resulting in 2 dummy codings:

```
df$dummy1 <- as.factor(df$VAR1)
levels(df$dummy1) <- c(0,1,0)
df$dummy1 <- as.numeric(df$dummy1)

df$dummy2 <- as.factor(df$VAR1)
levels(df$dummy2) <- c(0,0,1)
df$dummy2 <- as.numeric(df$dummy2)
```

or you want to use effect codings.

To my opinion, this is the most useful one to do as things are centered in our interpretation. When using dummy, you need to interpret the main effect of a particular variable 'when all other coefficients are kept stable'. In the case of a dummy coding, this means that this refers to the main effect in presence of the reference value of the dummy coding. When doing effect coding, this is not the case and you have a more clear interpretation of the main effect.

```
df$dummy1 <- ifelse(df$VAR1 == "group2", 1, 0)
df$dummy2 <- ifelse(df$VAR1 == "group3", 1, 0)
df$dummy3 <- ifelse(df$VAR1 == "group4", 1, 0)
```

Also important is to center numeric variables. When centering, you keep the standard deviation. When standardizing, this is forced to 1. This is a choice, but I, personally, like it when the initial variance is kept in terms of interpretating the results.

```
df$VAR1.c <- as.numeric(scale(df$VAR1,scale=FALSE,center=TRUE))
```

Time to build your model (e.g., 4 main effects and 1 two-way interaction):

```
model <- lm(OUTCOME~ VAR1.c + VAR2.c + VAR3.c*VAR4.c, data=df)
```

Check the output of the model:

```
summary(model)
```

Check standardized coefficients for the sake of interpretation and reporting:

```
standardize_parameters(model, method = "basic")
```

Check effect sizes, which always should be done next to *p*-values:

```
eta_squared(model)
```

There are also function to have a clear html output of your model:

```
tab_model(model)
```

Important is to check to what your model does satisfy all models. A nice way to do this is using the `check_model` function, however this might take a while (especially in more complex models). You can check

all diagnostics separately (which may be faster):

Model assumptions:

1. **Linearity**: is my model linear?
   - *Check?*: (1) is the point cloud at random and (2) is the blue line in plot 2 similar to the horizontal line in *plot 2*?
   - *Violation?*: consider another relationship (e.g. cubric, curvilinear)
2. **Normality**: is the distribution of my parameters / residuals normal?
   - *Check?*: do I have a Q-Q plot in *plot 3* where all datapoints are as close too the diagonal? Is the distribution as similar as possible to the normal distribution in *plot 8*?
   - *Violation?*: consider transformations of your parameters or check which variable is necessary to add to the model
3. **Homoscedasticity**: is the spread of my data across levels of my predictor the same?
   - *Check?*: (1) is the point cloud at random and (2) is the blue line in plot 2 similar to the horizontal line in *plot 2*? (3) Is there a pattern in *plot 4*?
   - *Violation?*: in case of heteroscedasticity, you will have inconsistency in calculation of standard errors and parameter estimation in the model. This results in biased confidence intervals and significance tests.
4. **Independence**: are the errors in my model related to each other?
5. **Influential outliers**: are there outliers influential to my model?
   - *Check?*: is the blue line in *plot 7* curved?
   - *Violation?*: this could be problematic for estimating parameters (e.g. mean) and sum of squared and biased results.

```
check_model(model) #might take a while

check_normality(model)
check_outliers(model)
check_heteroscedasticity(model)
multicollinearity(model)
```

8.1 Significant interactions

When working with interaction effects, you want to check standardized simple slopes (can be done for 2- and 3-way effects). The John-Newman plot is interesting to check for which values of the moderator your interaction is significant (nice for interpretation):

```
sim_slopes(model,
           pred = VAR3.c,
           modx=VAR4.c,
           #  modx.values=c(-1.17,-0.42,0.34,1.10),
                    # if you want to have values for specific levels
           # mod2 = VAR5.c, # in case of 3-way interaction
           # mod2.values = VAR5.c,
           centered="all",
           jnplot = TRUE)
```

Following code can be used for **a fast check** of how you interaction looks like.
Beware, the x-axis variable needs to be numeric!

```
graph_model(model,
            y=OUTCOME,
            x=VAR3.c,
            lines=VAR4.c,
            #split=VAR5.c # inc ase of 3-way interaction
            errorbars = 'SE',
            bargraph=FALSE)+
  coord_cartesian(ylim=c(1,5))+
  theme_classic()+
  labs(
    title = 'TITLE',
    subtitle = 'SUBTITLE',
    x = "X AXIS",
    y = "Y AXIS",
    color="MODERATOR"
  )
```

Following code can be used for **a complete check** of how you interaction looks like, including the presentation of simple slopes on your figure.

First, we get a dataframe including predicted values based on our model. In the `ggeffect` function, you can specify the numeric levels you want to have predicted values.

```
predictions <- data.frame(ggeffect(model,
                                   c('VAR3.c[-1,0,1]',
                                     'VAR4.c[meansd]') # you can specify values here
))
```

Here, we make sure that our x-axis is numeric and our moderator levels are categorical.

```
predictions$x <- as.numeric(predictions$x)
predictions$group <- as.factor(predictions$group)
```

Here, we have an extended code for a nice two-way interaction figure. As the figure will show up in the *Plots* tab in Rstudio, you can save it as png or as pdf.

```
ggplot(predictions,
       aes(x, predicted, color=group, linetype=group)) +
  geom_line(size=0.8)+
  geom_point(size=1, shape=19)+

  # set up different linetypes for black/white printing
  scale_linetype_manual('MODERATOR', #give name to moderator
                        labels=c('-1 SD','Mean','+1 SD'), # as an example
                        values = c("dotted","dashed","solid"))+ # as an example

  # set up different colors
  scale_color_manual('MODERATOR',
                     labels=c('-1 SD','Mean','+1 SD'),
                     values = c("#18a1cd","#1d81a2","#15607a"))+ #check google for hex code

  # provide textual info on simple slopes in figure
  #x is the position on the x-axis
  #y is the position on the y-axis (check predictions dataset for highest values)
```

```
#label is the simple slope coefficient
annotate("text", x=1, y=4.37, label=".59 ***",fontface =1, color = "black",angle=0)+
annotate("text", x=1, y=3.91, label=".49 ***",fontface =1, color = "black",angle=0)+
annotate("text", x=1, y=3.45, label=".38 ***",fontface =1, color = "black",angle=0)+

# settings for a theme
theme(
  legend.position = 'top',
  legend.title = element_blank(),
  axis.title.x = element_blank(),
  plot.caption = element_text(color = "black"  ),
  text=element_text(   family="Helvetica"),
  strip.placement = "outside")+

# info regarding y axis
scale_y_continuous(limits = c(1,5),  # range
                   breaks = seq(1,5, by = 1))+ # 1 to 5 by steps of 1

# info regarding x axis
scale_x_continuous(breaks=c(-1,0,1), # range
                   labels=c('-1','0','1'), # labels
                   expand = c(0.2, 0))+ # space on the left and right side of the figure

# ggplot theme
theme_bw() +

# info regarding legend
theme(legend.key.size = unit(1, 'cm'),
      legend.box = 'vertical')+

# labels for the axes
labs(
  title = 'TITLE',
  subtitle = 'SUBTITLE',
  x = 'X AXIS',
  y = 'Y AXIS')
```

The three-way interaction figure is a little bit more complicated. This will be added soon.

9 Long and wide

9.1 Wide to long

First, we make a list of those variables we want to restructure.

```
longlist <- list(
  VAR1=c('VAR1_1','VAR1_2','VAR1_3','VAR1_4'),
  VAR2=c('VAR2_1','VAR2_2','VAR2_3','VAR2_4')
)
```

The list is used in the code below:

```
dflong <- reshape(data = df, # name of the dataset
                  idvar = "ID", # group variable including dependent variance
                  varying = longlist,
                  direction="long",
                  v.names = names(longlist),
                  sep="_")
```

9.2 Long to wide

Complete the following function:

```
dfwide <- reshape(dflong, # name of the dataset
                  idvar = "ID",  # name of the grouping variable including dependent variance
                  timevar = "time", # time variable
                  direction = "wide")

head(dfwide) # check whether the function did what you expected
```

10 Multilevel Modeling

Following packages are required:

```
x<-c('sjPlot','lme4','lmerTest','performance','effectsize',
     # for visualizations
     'reghelper','interactions','ggeffects','ggplot2')
lapply(x, require, character.only = TRUE)
```

Before building our model, we need to make sure that all our input is ok.
For instance, we want to have dummy or effect codings instead of factor variables. Here, we work with the example of a factor with 3 levels, resulting in 2 dummy codings:

```
df$dummy1 <- as.factor(df$VAR1)
levels(df$dummy1) <- c(0,1,0)
df$dummy1 <- as.numeric(df$dummy1)

df$dummy2 <- as.factor(df$VAR1)
levels(df$dummy2) <- c(0,0,1)
df$dummy2 <- as.numeric(df$dummy2)
```

or you want to use effect codings.

To my opinion, this is the most useful one to do as things are centered in our interpretation. When using dummy, you need to interpret the main effect of a particular variable 'when all other coefficients are kept stable'. In the case of a dummy coding, this means that this refers to the main effect in presence of the reference value of the dummy coding. When doing effect coding, this is not the case and you have a more clear interpretation of the main effect.

```
df$effect1 <- as.factor(df$VAR1)
levels(df$effect1) <- c(-1,1,0)
df$effect1 <- as.numeric(df$effect1)

df$effect2 <- as.factor(df$VAR1)
levels(effect2) <- c(-1,0,1)
df$effect2 <- as.numeric(df$effect2)
```

Also important is to center numeric variables. When centering, you keep the standard deviation. When standardizing, this is forced to 1. This is a choice, but I, personally, like it when the initial variance is kept in terms of interpretating the results.

```
dflong$VAR1.c <- as.numeric(scale(dflong$VAR1,scale=FALSE,center=TRUE))
```

Time to build your model (e.g., 4 main effects and 1 two-way interaction):

```
ML_model <- lmer(OUTCOME~ VAR1.c + VAR2.c + VAR3.c*VAR4.c + (1|ID), data=dflong) # remark the random ef
```

Check the output of the model:

```
summary(model)
```

Check standardized coefficients for the sake of interpretation and reporting:

```
standardize_parameters(model, method = "basic")
```

Check effect sizes, which always should be done next to *p*-values:

```
eta_squared(model)
```

There are also function to have a clear html output of your model:

```
tab_model(model)
```

Important is to check to what your model does satisfy all models. A nice way to do this is using the `check_model` function, however this might take a while (especially in more complex models). You can check all diagnostics separately (which may be faster):

Model assumptions:

1. **Linearity**: is my model linear?
   - *Check?*: (1) is the point cloud at random and (2) is the blue line in plot 2 similar to the horizontal line in *plot 2*?
   - *Violation?*: consider another relationship (e.g. cubric, curvilinear)
2. **Normality**: is the distribution of my parameters / residuals normal?
   - *Check?*: do I have a Q-Q plot in *plot 3* where all datapoints are as close too the diagonal? Is the distribution as similar as possible to the normal distribution in *plot 8*?
   - *Violation?*: consider transformations of your parameters or check which variable is necessary to add to the model
3. **Homoscedasticity**: is the spread of my data across levels of my predictor the same?
   - *Check?*: (1) is the point cloud at random and (2) is the blue line in plot 2 similar to the horizontal line in *plot 2*? (3) Is there a pattern in *plot 4*?
   - *Violation?*: in case of heteroscedasticity, you will have inconsistency in calculation of standard errors and parameter estimation in the model. This results in biased confidence intervals and significance tests.
4. **Independence**: are the errors in my model related to each other?
5. **Influential outliers**: are there outliers influential to my model?
   - *Check?*: is the blue line in *plot 7* curved?
   - *Violation?*: this could be problematic for estimating parameters (e.g. mean) and sum of squared and biased results.

```
check_model(model) #might take a while

check_normality(model)
check_outliers(model)
check_heteroscedasticity(model)
multicollinearity(model)
```

10.1 Significant interactions

When working with interaction effects, you want to check standardized simple slopes (can be done for 2- and 3-way effects). The John-Newman plot is interesting to check for which values of the moderator your interaction is significant (nice for interpretation):

```
sim_slopes(model,
           pred = VAR3.c,
           modx=VAR4.c,
           #  modx.values=c(-1.17,-0.42,0.34,1.10),
                  # if you want to have values for specific levels
```

```
            # mod2 = VAR5.c, # in case of 3-way interaction
            # mod2.values = VAR5.c,
            centered="all",
            jnplot = TRUE)
```

Following code can be used for **a fast check** of how you interaction looks like.
Beware, the x-axis variable needs to be numeric!

```
graph_model(model,
             y=OUTCOME,
             x=VAR3.c,
             lines=VAR4.c,
             #split=VAR5.c # inc ase of 3-way interaction
             errorbars = 'SE',
             bargraph=FALSE)+
  coord_cartesian(ylim=c(1,5))+
  theme_classic()+
  labs(
    title = 'TITLE',
    subtitle = 'SUBTITLE',
    x = "X AXIS",
    y = "Y AXIS",
    color="MODERATOR"
  )
```

Following code can be used for **a complete check** of how you interaction looks like, including the presentation of simple slopes on your figure.

First, we get a dataframe including predicted values based on our model. In the `ggeffect` function, you can specify the numeric levels you want to have predicted values.

```
predictions <- data.frame(ggeffect(model,
                                   c('VAR3.c[-1,0,1]',
                                     'VAR4.c[meansd]') # you can specify values here
))
```

Here, we make sure that our x-axis is numeric and our moderator levels are categorical.

```
predictions$x <- as.numeric(predictions$x)
predictions$group <- as.factor(predictions$group)
```

Here, we have an extended code for a nice two-way interaction figure. As the figure will show up in the *Plots* tab in Rstudio, you can save it as png or as pdf.

```
ggplot(predictions,
       aes(x, predicted, color=group, linetype=group)) +
  geom_line(size=0.8)+
  geom_point(size=1, shape=19)+

  # set up different linetypes for black/white printing
  scale_linetype_manual('MODERATOR', #give name to moderator
                        labels=c('-1 SD','Mean','+1 SD'), # as an example
                        values = c("dotted","dashed","solid"))+ # as an example

  # set up different colors
  scale_color_manual('MODERATOR',
                     labels=c('-1 SD','Mean','+1 SD'),
```

```
                    values = c("#18a1cd","#1d81a2","#15607a"))+ #check google for hex code

# provide textual info on simple slopes in figure
#x is the position on the x-axis
#y is the position on the y-axis (check predictions dataset for highest values)
#label is the simple slope coefficient
annotate("text", x=1, y=4.37, label=".59 ***",fontface =1, color = "black",angle=0)+
annotate("text", x=1, y=3.91, label=".49 ***",fontface =1, color = "black",angle=0)+
annotate("text", x=1, y=3.45, label=".38 ***",fontface =1, color = "black",angle=0)+

# settings for a theme
theme(
  legend.position = 'top',
  legend.title = element_blank(),
  axis.title.x = element_blank(),
  plot.caption = element_text(color = "black"  ),
  text=element_text(   family="Helvetica"),
  strip.placement = "outside")+

# info regarding y axis
scale_y_continuous(limits = c(1,5),  # range
                   breaks = seq(1,5, by = 1))+ # 1 to 5 by steps of 1

# info regarding x axis
scale_x_continuous(breaks=c(-1,0,1), # range
                   labels=c('-1','0','1'), # labels
                   expand = c(0.2, 0))+ # space on the left and right side of the figure

# ggplot theme
theme_bw() +

# info regarding legend
theme(legend.key.size = unit(1, 'cm'),
      legend.box = 'vertical')+

# labels for the axes
labs(
  title = 'TITLE',
  subtitle = 'SUBTITLE',
  x = 'X AXIS',
  y = 'Y AXIS')
```

The three-way interaction figure is a little bit more complicated. This will be added soon.

11 Structural Modelling

For this part, we need the `lavaan` package.

As we cannot formulate interaction effects right away in the model, we calculate them first, outside the model. Of course, this is only relevant when you want to include an interaction effect.

```
df$VAR1VAR2 <- df$VAR1.c * df$VAR2.c # variables are centered!

# write the model

SEM <- '
OUTCOME1 ~ VAR1 + VAR2 + VAR1VAR2
OUTCOME2 ~ VAR1 + VAR2 + VAR1VAR2
'

# run the model
fit <- sem(model = SEM, data = df)

options(max.print=1000000) #useful when output is large

# check output of the model
summary(fit,
        fit.measures = TRUE,
        standardize = TRUE,
        rsquare = TRUE)

# useful when fit of model is not good:
modificationIndices(fit, sort.=TRUE, minimum.value=3)
```

A useful online tool to visualize structural equation model ishttps://app.diagrams.net/

12 Mediation

Here, you can find a simple example of a code to construct a mediation model, including one predictor (PRED), a mediator (MED) and an outcome (OUTCOME). Also, we included the calculation of the indirect and the total effect.

```
model <-
'
# predictor -> mediator
MED ~ a*PRED

# predictor + mediator --> outcome
OUTCOME ~ c*PRED + b*MED

# calculation of an indirect effect
ab := a*b

# calculating total effect
total := c + (a*b)
'

fit <- sem(model, data = df)
summary(fit,
        fit.measures = TRUE,
        standardized = TRUE,
        rsquare = TRUE)
modificationIndices(fit, sort.=TRUE, minimum.value=3)
```

13 Multilevel SEM

Only up to 2 levels in R (currently)

```
dflong$VAR1VAR2 <- dflong$VAR1.c * dflong$VAR2.c # variables are centered!

# write the model
ML_SEM <- '
level:1
OUTCOME1 ~ VAR1 + VAR2 + VAR1VAR2
OUTCOME2 ~ VAR1 + VAR2 + VAR1VAR2

level:2
OUTCOME1 ~ VAR1 + VAR2 + VAR1VAR2
OUTCOME2 ~ VAR1 + VAR2 + VAR1VAR2
'

# run the model
fit <- sem(model = ML_SEM,
           data = dflong,
           cluster = "ID", # group variable including dependent variance
           optim.method = "em")

options(max.print=1000000) #useful when output is large

# check the output
summary(fit,
        fit.measures = TRUE,
        standardize = TRUE,
        rsquare = TRUE)

# useful when fit of model is not good:
modificationIndices(fit, sort.=TRUE, minimum.value=3)
```

14 Cross-lagged panel model

```
model <-
'
VAR1_T2 + VAR2_T2 ~ VAR1_T1 + VAR2_T1
'

fit <- sem(model, data = df)
summary(fit,
        fit.measures = TRUE,
        standardized = TRUE,
        rsquare = TRUE)
modificationIndices(fit, sort.=TRUE, minimum.value=3)
```

15 RI-CLPM

Flournoy wrote an amazingly useful package to generate a syntax for a RI-CLPM in the `riclpmr` package.

```
library(devtools)
# install_github('jflournoy/riclpmr') # in case you have not installed the package yet
library(riclpmr)
library(lavaan)
```

Select the variables from your **wide** dataset.

```
data_riclpm <- dfwide[,c("ID", # also important to include the grouping avriable
                         'VARIABLEX_1','VARIABLEX_2','VARIABLEX_3','VARIABLEX_4',
                         'VARIABLEY_1','VARIABLEY_2','VARIABLEY_3','VARIABLEY_4')]

# give different column names to make the output of the model more readable
colnames(data_riclpm) <- c("id",
                           'x1','x2','x3','x4',
                           'y1','y2','y3','y4')

data_riclpm <- data_riclpm[ , -c(1)] #remove ID

# refer which columns belong to a specific variable
var_groups <- list(
  x=c('x1','x2','x3','x4'),
  y=c('y1','y2','y3','y4')
)
```

Just run the following code. Herein, a constrained and an unconstrained model are performed and compared (via ANOVA). Based on which model provides the best fit of the data, you can check the output of the model.

```
# construct contraint model
model_text <- riclpmr::riclpm_text(var_groups,
                                   constrain_over_waves = TRUE,
                                   constrain_ints = "free")

fit_constraints <- riclpmr::lavriclpm(riclpmModel = model_text,
                                      data = data_riclpm,
                                      missing = 'fiml',
                                      meanstructure = T,
```

```
                                        int.ov.free = T)
# construct unconstraint model
model_text <- riclpmr::riclpm_text(var_groups,
                                   constrain_over_waves = FALSE,
                                   constrain_ints = "free")

fit_noconstraints <- riclpmr::lavriclpm(riclpmModel = model_text,
                                        data = data_riclpm,
                                        missing = 'fiml',
                                        meanstructure = T,
                                        int.ov.free = T)

# run this to compare constraint and unconstraint model in terms of data fit
anova(fit_constraints,fit_noconstraints)

# Check the output of the chosenmodel
summary(fit_constraints,
        fit.measures = TRUE,
        standardize = TRUE,
        rsquare = TRUE)
```

16 Basic visualizations

Run this code to make a descriptive bar plot

```
aggregate <- as.data.frame(aggregate(df$OUTCOME,by=list(df$GROUP),
                                     mean, na.rm=TRUE))
colnames(aggregate) <- c('GROUP','OUTCOME')

ggplot(aggregate,
       aes(x = GROUP, y = OUTCOME)) +
  geom_bar(stat="identity",width = 0.7,
           position=position_dodge())+
  ylim(1,5)+
  theme(
    legend.position = 'right',
    legend.title = element_blank(),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black"  ),
    text=element_text(   family="Helvetica",size=7)
  )+
  theme_classic(base_size=10)+ # fontsize in figure
  labs(
    y="Y AXIS",
    x="X AXIS"
  )
```

Run this code to make a descriptive line plot

```
aggregate <- as.data.frame(aggregate(df$OUTCOME,by=list(df$GROUP),
                                     mean, na.rm=TRUE))
colnames(aggregate) <- c('GROUP','OUTCOME')

ggplot(aggregate,
       aes(x = GROUP, y = OUTCOME)) +
  geom_point(colour = "#007f7f")+
  geom_line(colour = "#007f7f", size = 1) +
  ylim(1,5)+
  theme(
    legend.position = 'right',
    legend.title = element_blank(),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black"  ),
    text=element_text(   family="Helvetica",size=7)
  )+
  theme_classic(base_size=10)+ # fontsize in figure
  labs(
    y="Y AXIS",
    x="X AXIS"
  )
```

17 Two-step cluster analysis

Following packages are required:

```
x<-c('factoextra','cowplot','cluster','NbClust','questionr','tidyr','vcd')
lapply(x, require, character.only = TRUE)
```

17.1 Preparatory steps

First, we standardize all cluster variables make a subset of our dataframe, only and only including the cluster variables. This dataset will be used to perform some validation techniques in order to select the most accurate number of clusters. In this example, we use three cluster variables.

```
# standardization
df$VAR1.sc <- as.numeric(scale(df$VAR1,scale=TRUE))
df$VAR2.sc <- as.numeric(scale(df$VAR2,scale=TRUE))
df$VAR3.sc <- as.numeric(scale(df$VAR3,scale=TRUE))

# make a subset (only including complete cases here)
df_clust <- na.omit(df[,c("VAR1.sc","VAR2.sc","VAR3.sc")])
```

17.2 Validation before clustering

Before we actually want to cluster, we already can have an indication to what extent 'our data is clusterable'. This is done using the Hopkin statistic (Lawson & Jurs, 1990), which can be interpreted as a 'chance of finding meaningful clusters'. When this is 0.50, this means there is an equally high chance of observing meaningful clusters as finding meaningless clusters. The higher, the better.

In our experience, we noticed that, when having a Hopkin statistic of, for instance, 0.52 or 0.56, the validation techniques of finding the optimal number of clusters indeed were inconclusive.

```
# original dataset
res.o <- get_clust_tendency(df_clust,
                            n = nrow(df_clust)-1,
                            graph=FALSE)
H_dataset <- round(res.o$hopkins_stat,3)

# random dataset
random_dffull <- as.data.frame(apply(df_clust,2,
                                     function(x){runif(length(x),
                                                       min(x),
                                                       max(x))}
))

# testing tendency
res.r <- get_clust_tendency(random_dffull,
                            n = nrow(random_dffull)-1,
                            graph=FALSE)

H_random <- round(res.r$hopkins_stat,3)

# make table for output
table_H <- cbind(H_dataset,H_random)
colnames(table_H) <- c('df_clust','random dataset')
```

```
rownames(table_H) <- 'Hopkin statistic'
table_H
```

17.3 Validation during clustering

The following set of code will end up in four figures showing a particular type of validation for a number of clusters. Indeed, we want to have all four of them, as we want to make a considered decision on how many clusters are in our dataset. This does not mean that all four types of validations will point towards the same number of clusters (sometimes it does, indicating strong evidence for a particular number). Therefore, you need to consider all types and explain in your reporting why you choose for a particular number of clusters.

The validation techniques are:

1. *Elbow method*: the number of clusters with both a minimum of within-cluster variation and a maximum of between-cluster variation

2. *the Average Silhouette method*: the number of clusters with the highest average silhouette, indicating the best quality of clustering (Kaufman & Rousseeuw, 1990)

3. *the Gap statistic method*: the number of clusters with the highest Gap-statistic (Tibshirani et al., 2001)

4. *Majority rule*: a summary of 30 indices reporting the most optimal number of clusters using the 'NbClust' function (Charrad et al.,2014), including the CH index (Calinski and Harabasz, 1974)

Just run this. There is no need for any adaptation from your side. Later on, I will put all of this into a single function, but now you can just copy and run this list:

```
wss <- function(k) {
  hkmeans(dffull, k,
          hc.metric="euclidian",
          hc.method='ward.D2',
          iter.max = 10,
          km.algorithm = "Hartigan-Wong")$tot.withinss
}
k.values <- 1:10
wss_values <- map_dbl(k.values, wss)
elbowmethod <- as.data.frame(t(rbind(k.values,wss_values)))

avg_sil <- function(k) {
  km.res <- hkmeans(dffull, k,
                    hc.metric="euclidian",
                    hc.method='ward.D2',
                    iter.max = 10,
                    km.algorithm = "Hartigan-Wong")
  ss <- silhouette(km.res$cluster, dist(dffull, method = "euclidean"))
  mean(ss[, 3])
}


hkm.res <- hkmeans(dffull, k=3,
                   hc.metric="euclidian",
                   hc.method='ward.D2',
                   iter.max = 10,
                   km.algorithm = "Hartigan-Wong")
```

```
ss <- silhouette(hkm.res$cluster, dist(dffull, method = "euclidean"))
ss <- cbind(ss[,1],ss[,2],ss[,3])
ss <- as.data.frame(ss)
colnames(ss) <- c('cluster','neighbor','silhouette')
ss <- with(ss, ss[order(cluster, silhouette,decreasing = TRUE),])
ss$cluster <- as.factor(ss$cluster)
ss$ppnr <- 1:dim(ss)[1]
ss <- ss[order(match(ss$cluster, c("1", "2", "3"))),]

averagesilhplotplot <- ggplot(data=ss,
                              aes(x=ppnr, y=silhouette,fill=cluster, order=cluster)) +
  geom_bar(stat="identity",width = 0.7,
           position=position_dodge())+
  scale_fill_manual("Cluster",
                    values = c("1"="#2f3c4d","2"="#ad131b","3"="#cc6a0e"))+
  theme(
    legend.position = 'right',
    legend.spacing.x = unit(1, 'mm'),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black")
  )+
  theme_minimal()+
  ylab("Silhouette width")+
  labs(subtitle = "Silhouette coefficients by cases" ) +
  xlab("Cases")

minnumber <- 1
maxnumber <- 10
dataset <- dffull

uncertacompytable<-c()
for (i in minnumber:maxnumber) {
  km.res <- hkmeans(dataset, i,
                    hc.metric="euclidian",
                    hc.method='ward.D2',
                    iter.max = 10,
                    km.algorithm = "Hartigan-Wong")
  sum_i <- km.res$betweenss + km.res$tot.withinss
  number_i <- i
  ss_i <- cbind(km.res$tot.withinss, km.res$betweenss,sum_i,number_i)
  ss_i <- as.data.frame(ss_i)
  ss_i <- round(ss_i,2)
  colnames(ss_i) <- c('within','between','sum',"clusters")
  uncertacompytable <- rbind(uncertacompytable,ss_i)
}

uncertacompytable$within <- (uncertacompytable[,1]/uncertacompytable[,3])*100
uncertacompytable$between <- (uncertacompytable[,2]/uncertacompytable[,3])*100
uncertacompytablee <- uncertacompytable[,c("clusters",'within','between')]
uncertacompytablee <- round(uncertacompytablee,2)
data_longtable <- gather(uncertacompytablee, type, variance, c('within','between'),
                         factor_key=TRUE)
```

```
varianceplot <-ggplot(data=data_longtable, aes(x=clusters, y=variance, fill=type)) +
  geom_bar(stat="identity",width = 0.7,

           position=position_dodge(),color='black')+
  scale_fill_manual("Variance",
                    values = c("within" = "#2f3c4d",
                               "between" = "#ad131b"))+
  theme(
    legend.spacing.x = unit(1, 'mm'),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black")
  )+
  scale_y_continuous(labels = function(x) paste0(x, "%"))+
  scale_x_continuous(name="K clusters", breaks=c(1:15))+
  theme_minimal()+
  theme(legend.position = 'top')+
  labs(subtitle = "Between- and Within variance")

k.values <- 2:10

avg_sil_values <- map_dbl(k.values, avg_sil)

averagesilh <- as.data.frame(t(rbind(k.values,avg_sil_values)))

averagesilhplot <- ggplot(data=averagesilh, aes(x=k.values, y=avg_sil_values)) +
  geom_line(stat="identity",width = 0.7,
            position=position_dodge(),color='black')+
  geom_point()+
  theme(
    legend.position = 'right',
    legend.spacing.x = unit(1, 'mm'),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black")
  )+
  scale_x_continuous(name="K clusters", breaks=c(1:10))+
  theme_minimal()+
  ylab("Average Silhouettes")+
  labs(subtitle = "Average silhouette method")

gapstatisticplot <- fviz_nbclust(dffull,
                                 hkmeans,
                                 k.max = 10,
                                 linecolor = "black",
                                 method="gap_stat", nboot=50) +
  labs(title=NULL, subtitle = "Gap statistic", xlab='K clusters') +theme_minimal()

otherind <- NbClust(dffull, distance = "euclidean",
                    min.nc = 3, max.nc = 7,
                    method = "ward.D2", index ="all")


bestnc <- as.data.frame(t(otherind$Best.nc))
bestnc$Number_clusters <- as.factor(bestnc$Number_clusters)
```

```
summ.bestnc <- as.data.frame(summary(bestnc$Number_clusters))
clusters <- rownames(summ.bestnc)
summ.bestnc <- cbind(summ.bestnc,clusters)
colnames(summ.bestnc) <- c('Frequency',"clusters")
summ.bestnc$clusters <- as.numeric(summ.bestnc$clusters)
summaryplot <- ggplot(data=summ.bestnc, aes(x=clusters, y=Frequency)) +
  geom_bar(stat="identity",width = 0.7,
           position=position_dodge(),fill="#2f3c4d")+
  theme(
    legend.position = 'right',
    legend.spacing.x = unit(1, 'mm'),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black")
  )+
  theme_minimal()+
  ylab("Frequency of indices")+
  scale_y_continuous(breaks = seq(0, max(summ.bestnc$Frequency), by = 1))+
  scale_x_continuous(name="Clusters", breaks=c(0:10))+
  xlab("Clusters")+
  ggtitle('Summary frequency 30 indices')

# the figures into one figure
plot_grid(varianceplot,averagesilhplot,
          gapstatisticplot,summaryplot,nrow = 2)
```

17.4 After having decided the number of clusters

After all validation techniques, you can choose the number of clusters and perform the `hkmeans` function.

```
# function for two-step clustering procedure
hkm <- hkmeans(df[,c("VAR1.sc","VAR2.sc","VAR3.sc")],
               k = 3, # chosen number of clusters
               hc.metric="euclidian",
               hc.method='ward.D2',
               iter.max = 10,
               km.algorithm = "Hartigan-Wong")

# assign cluster variable to your dataset
df$clusters <- as.factor(hkm$cluster)

# check proportions of participants into the clusters
freq(df1$clusters)
```

Using the following code, we can have a barplot. Here, we work with the standardized cluster variables, although it might be interesting to have a figure with the raw values as the standardized variables make relative differences. Then, we need to make sure that we also interpret the absolute values (lower values do not mean low values!):

```
# calculate means of variables by cluster levels
hkmdata <- as.data.frame((aggregate(df[,c("VAR1.sc","VAR2.sc","VAR3.sc")],
                                    by=list(cluster=df$clusters),
                                    mean)))

# restructure into long format
```

```
data_long <- gather(hkmdata, type, measurement,
                    names(df1[,c("VAR1.sc","VAR2.sc","VAR3.sc")]),
                    factor_key=TRUE)
# change column names
colnames(data_long) <- c('Clusters','Type','Value')

# make sure they have the right format
data_long$Clusters <- as.numeric(data_long$Clusters)
data_long$Type <- as.factor(data_long$Type)

# if you want to change the order of the clusters, this can be done here:
data_long$Clusters <- as.character(data_long$Clusters)
data_long$Clusters <- as.factor(data_long$Clusters)
levels(data_long$Clusters) <- c("3","4","1","2","5") # preferred order
data_long$Clusters <- as.character(data_long$Clusters)
data_long$Clusters <- as.numeric(data_long$Clusters)

# figure
ggplot(data=data_long,
       aes(x=Clusters, y=Value, fill=Type, linetype=Type)) +
  geom_bar(stat="identity",width = 0.7,
           position=position_dodge(),
           color='black')+

  # y axis
  scale_y_continuous(limits = c(-2.2,2.2))+

  # provide color to bars
  scale_fill_manual("Type of motvation", # name of cluster variables

                    # labels for cluster variables
                    labels=c('Autonomous','Controlled',
                             'Distrust','Effort-based'),

                    # colors for cluster variables
                    values=c('#2e2e2e','#858585',
                             '#cecece','#ffffff') )+

  # provide different linetype to bars (useful for black/white printing)
  scale_linetype_manual("Type of Motvation", # name of cluster variables

                        # labels for cluster variables
                        labels=c('Autonomous','Controlled',
                                 'Distrust','Effort-based'),

                        # linetypes for cluster variables
                        values=c('dotted','dotdash',
                                 'dashed','solid') )+

  # settings for x-axis
   scale_x_continuous(name="Cluster",  # name for x axis
                      breaks=c(1:3), # 'breaks' = number of clusters
```

```
                      # labels for clusters.
                      # Here you can add the proportions
                      # underneath the name, using \n
                      labels= c( "Cluster 1\n20%",
                                 "Cluster 2\n20%",
                                 "Cluster 3\n20%"))+

  # settings for layout
  theme(
    legend.position = 'right',
    legend.spacing.x = unit(1, 'mm'),
    axis.title.x = element_blank(),
    plot.caption = element_text(color = "black")
  )+

  # ggplot theme
  theme_bw()
```

Based on the output of the figure - which can help in interpreting the content of a cluster - now you can provide labels to the clusters in the dataset itself. This variable can be used in between-cluster analyses (can be done using frequency tables or (M)ANOVA)

```
levels(df$clusters) <- c('NAME_clus1','NAME_clus2','NAME_clus3',
                         'NAME_clus4','NAME_clus5')
```

17.5 Validation after clustering

Even after we chose the number of clusters and even labelled them, we can perform a validation technique to check the 'stability' of the clusters. Basically, it performed the clustering analyses on one part of the dataset, and uses the K-mean values as the input of a clustering analysis on the other part of the dataset:

```
# divide dataset into two parts: subset A and B
set.seed(7)
ss <- sample(1:2,size=nrow(df),replace=TRUE,prob=c(0.5,0.5))
subsetA <- df[ss==1,]
subsetB <- df[ss==2,]

# perform clustering analysis in both parts
hkmA <- hkmeans(subsetA[,c("VAR1.sc","VAR2.sc","VAR3.sc")],
                k = 3,
                hc.metric="euclidian",
                hc.method='ward.D2',
                iter.max = 10,
                km.algorithm = "Hartigan-Wong")

hkmB <- hkmeans(subsetB[,c("VAR1.sc","VAR2.sc","VAR3.sc")],
                k = 3,
                hc.metric="euclidian",
                hc.method='ward.D2',
                iter.max = 10,
                km.algorithm = "Hartigan-Wong")

# use the output of the cluster analysis in each subset as the initial starting points
# for a clustering analysis in the other subset
```

```
kmeanAB <- kmeans(subsetA[,c("VAR1.sc","VAR2.sc","VAR3.sc")],
                  # here we refer to the values of the cluster analyses
                  # in the other subset
                  hkmB$centers,
                  iter.max=10,
                  nstart=1,
                  algorithm = "Hartigan-Wong")
subsetA$AB <- as.factor(kmeanAB$cluster) # assign it to the subset

kmeanBA <- kmeans(subsetB[,c("VAR1.sc","VAR2.sc","VAR3.sc")],
                  hkmA$centers,
                  iter.max=10,
                  nstart=1,
                  algorithm = "Hartigan-Wong")
subsetB$BA <- as.factor(kmeanBA$cluster)
```

So, when this is done, we want to have an indication of how good the clustering in a different part of the dataset results in a good clustering in another part of the dataset. The stability is checked with a Cohen's Kappa-index k testing the correspondence between the subsample-clustering results and the clustering results forming from the original clustering procedure. An acceptable cluster stability is assumed when k is .60 or higher (Asendorpf et al., 2001). The final results of the clustering procedure will be presented in a barplot with the standardized cluster variables as a function of the cluster classification.

Beware, we might need to restructure the table to make sure we have the same clusters being compared.

```
# check in subset A
mytableAB <- with(subsetA, table(clusters,AB))
mytableAB # table. When ok, do nothing.

# When not oke, change order of columns:
colnames(mytableAB) <- c("1","2","4","3")
mytableAB <- mytableAB[ , c("1","2","3","4")]

# computer Kappa
Kappa(mytableAB)

# repeat for subset B:
mytableBA <- with(subsetB, table(clusters,BA))
mytableBA# table. When ok, do nothing.

# When not oke, change order of columns:
colnames(mytableBA) <- c("1","3","4","2")
mytableBA <- mytableBA[ , c("1","2","3","4")]

Kappa(mytableBA)
```