introduction

Albert Buchard

12/20/2016

How to keep your code clean

Coding convention

- Pick a naming convention and stick to it
- camelCase = “this is a nice style”
- snake_case = “this is ok too”
- Comment your code
- Look at the google style book to make sure your code is easilly readable by anyone
- https://google.github.io/styleguide/Rguide.xml
- they advice to use only “<-” and not “=” but i personally think it is pointless

Storage

Keep a README.md file at the root of your folder explaining where everything is, helping someone that knows nothing about your data to navigate your work. Keeping your work in the cloud, through services like dropbox, icloud, or google drive. The best would be github but it is not easy in the begining.

Folders

Keep your folder clean, with clear names in minuscules separated by “_" :

- data
  - raw
  - preprocessed
  - analysis
    - analysis_one …
- scripts
  - preprocessing: scripts that transforms the raw data in processed data
  - analysis: scripts that use preprocessed data and performs analysis on it
  - markdown: your markdown files
  - r_files: other R files, like utility functions
- media: here should go any ressources, presentations, images you produced or needed etc…
  - presentations
  - graphics
  - text
  - notes
- backups: you might need a backup folder when in doubt
  - data
  - script
  - media

Variables

R infer on its own the type of the variable that you want to create based on the input you give. All variables are at minimum a vector.

Atomic vector data type

```
# Character
a <- "This is a character vector"

# Numeric (integer)
a = 12
a <- 12

# Numeric (Float)
a <- 12.2

# Logical
a <- TRUE
a <- FALSE

print(a)
```

```
## [1] FALSE
```

Combine atomic elements

Vectors needs to be of same type

```
a <- c(1,2,3)
print(a)
```

```
## [1] 1 2 3
```

```
a <- c(1,"a")
print(a)
```

```
## [1] "1" "a"
```

List can mix types

```
a <- list(1,2,3)
print(a)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```r
a <- list(1,"a")
print(a)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
```

Matrix 2*2 Needs to be of same type

```r
a <- matrix( c('a','a','b','c','b',2), nrow = 2, ncol = 3, byrow = T)
print(a)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "b"
## [2,] "c"  "b"  "2"
```

```r
a <- matrix( c('a','a','b','c','b',2), nrow = 2, ncol = 3, byrow = F)
print(a)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "b"  "b"
## [2,] "a"  "c"  "2"
```

Array N*N Needs to be of same type

```r
a <- array(c('green','yellow'),dim = c(3,3,2))
print(a)
```

```
## , , 1
##
##      [,1]     [,2]     [,3]
## [1,] "green"  "yellow" "green"
## [2,] "yellow" "green"  "yellow"
## [3,] "green"  "yellow" "green"
##
## , , 2
##
##      [,1]     [,2]     [,3]
## [1,] "yellow" "green"  "yellow"
## [2,] "green"  "yellow" "green"
## [3,] "yellow" "green"  "yellow"
```

Factor

For categorical variables

```r
# Create a vector.
apple_colors <- c('green','green','yellow','red','red','red','green')

# Create a factor object.
```

```r
factor_apple <- factor(apple_colors)

# Print the factor.
print(factor_apple)
```

```
## [1] green  green  yellow red    red    red    green
## Levels: green red yellow
```

```r
print(nlevels(factor_apple))
```

```
## [1] 3
```

```r
# Change names of the factors
levels(factor_apple) <- c("Kindof Green", "Kindof Red", "Kindof Yellow")
print(factor_apple)
```

```
## [1] Kindof Green  Kindof Green  Kindof Yellow Kindof Red    Kindof Red
## [6] Kindof Red    Kindof Green
## Levels: Kindof Green Kindof Red Kindof Yellow
```

Dataframe ++

```r
first_names = c("Melissa", "Sibylle", "Zoe", "Maria")
ages = c(23, 22, 24, 25)

df <- data.frame(first_name = first_names,
                 age = ages,
                 subject = as.character(c("Activity", "Motivation", "Fluid Intelligence", NA)))



print(df)
```

```
##   first_name age            subject
## 1    Melissa  23           Activity
## 2    Sibylle  22         Motivation
## 3        Zoe  24 Fluid Intelligence
## 4      Maria  25               <NA>
```

```r
print(df$age)
```

```
## [1] 23 22 24 25
```

Missing values are "NOT ASSIGNED" or "NA"

```r
print(df$subject)
```

```
## [1] Activity           Motivation         Fluid Intelligence
## [4] <NA>
## Levels: Activity Fluid Intelligence Motivation
```

```r
print(is.na(df$subject))
```

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

I dont want a factor, I want characters !

Sometimes you have ot use function such as apply or sapply, that performs simple loops on your data.

```
print(sapply(df[, 3], as.character))
```

```
## [1] "Activity"            "Motivation"         "Fluid Intelligence"
## [4] NA
```

```
df[, 3] <- sapply(df[, 3], as.character)

print(df)
```

```
##   first_name age            subject
## 1    Melissa  23           Activity
## 2    Sibylle  22         Motivation
## 3        Zoe  24 Fluid Intelligence
## 4      Maria  25               <NA>
```

Operators

Relational operators

```
print(12>23)
```

```
## [1] FALSE
```

```
a = 12
print(a == 12)
```

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

```
print(a != 32)
```

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

```
print(a >= 11)
```

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

```
a = 32
```

Tests

```
if (a == 432) {
  print("a est egale a 12 !")
} else {
  print("pas egale a 12")
}
```

```
## [1] "pas egale a 12"
```

Logical operators

```r
print((12>23)&&(12<23))
```

```
## [1] FALSE
```

```r
a <- 12
print((a>20)||(a==12))
```

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

```r
print(!(a == 32))
```

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

```r
!is.na(a)
```

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

Element wise logic

When a vector is tested against a vector of same length

```r
a <- c(F, T, T)
b <- c(T, F, T)
print(a&b)
```

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

```r
print(a|b)
```

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

Others

```r
a <- 1:8
print(a)
```

```
## [1] 1 2 3 4 5 6 7 8
```

```r
a <- rep("ce qui est repete", 4)
print(a)
```

```
## [1] "ce qui est repete" "ce qui est repete" "ce qui est repete"
## [4] "ce qui est repete"
```

Flow control statements

Flow controls statements are all thestatement of a language that will redirect the flow of execution of a program.

Conditional control

Sometime you want to execute something only if a condition is true. The most used is the “if/else if/else” statement.

```
a = c(F,F,F,T)

if (a[1]) {
  print("first")
} else if (a[2]) {
  print("second")
}  else if (a[3]) {
  print("third")
}  else if (a[4]) {
  print("quatrieme")
} else {
  print("invalid")
}
```

```
## [1] "quatrieme"
```

```
b <- c(F,F,F,T)

# TODO: write a statement that checks if b has any of its value equal to TRUE.
# If it does return all the indices of the TRUE values
# If not, say that you did not find a True value in any of the %SIZE% elements of b
# (hint:: ?any and ?which)
```

When you only want to check the value of *ONE* variable. Another way is to use the *switch* statement. It test the value of a variable against several possibilities, like so:

```
strangeName <- "Grabulas"
switch(strangeName,
       "BJ Gabbour" = {
         print("It was Gabbour all along !")
       },
       "Hortiche" = {
         print("She's just everywhere")
       },
       "Grabulas" = {
         print("Run you fools !")
       })
```

```
## [1] "Run you fools !"
```

Loops

You often need to repeat some statement. That’s what *for* and *while* are here for !

```
for (i in 1:NROW(df)) {
  person = df[i,]

  print(paste0(person$first_name, person$age))
}
```

```
## [1] "Melissa23"
## [1] "Sibylle22"
## [1] "Zoe24"
## [1] "Maria25"
```

```
i = 0
while(i<10) {
  print(i)
  i = i + 1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

Functions

Functions are the central concept to programming. You can think of it as a box containing a series of instructions. Usually they take input, perform change on it, and returns a value. Note that they do not always take input or return a value and act only a series on instructions that do not takes input and do not return anything, but for example will read or write some information on the disk, or setting some parameter.

A simple sum

```
sommeFunction = function  (x, y, z) {
  # Do some action on the parameters
  sum = x+y+z

  # Return to sender the result of your computation
  return(sum)
}

somme = sommeFunction(1,2,3)
```

But of course R has a better function already built in !

```
sum(c(1,2,3))
```

```
## [1] 6
```

A function that generates participant ids

What if you want to set some default parameters ? Here is a more complex function that plays with strings to create random IDs for your subjects.

```
getRandomId = function(numberOfIds = 1, lenght=12, allowedCharacters = c(0:9, letters, LETTERS))
{
    # initialize vector
```

```r
    randomStrings = c(1:numberOfIds)

    # start the generation loop
    for (i in 1:numberOfIds)
    {
        randomStrings[i] <- paste(sample(allowedCharacters, lenght, replace=TRUE),
                                  collapse="")
    }

    # return the strings
    return(randomStrings)
}

# TODO Now generate 650 ids !
# ids = ...

# Add the ids to the data
```

Create your own function

Choose one between those three possible function, and create them: * A function that returns the product of two numbers such that aTimesb = product(a,b) * A function that adds a prefix to a string, such that prefixedString = prefix(prefixString, string) * A function that takes out the mean of each column of a data frame, and divides by the standard deviation (process called normalization)

Libraries

Install libraries

```r
install.packages("ggplot")
install.packages("psych")
```

Load libraries

```r
library(ggplot2)
library(psych)
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
#help(package= "psych")
#vignette("ggplot2-specs", package = "ggplot2")
```

Explore your data

Load data

Usually you will load three type of data: * Excel files: .xslx * Comma separated values: .csv * R data file: .RData

You load them differently

```
# Load a RData
setwd("~/Google Drive/Master Students/courses/introduction_a_r")
load("data/raw/data.RData")
#or if you want to rename your data
renamedData = get(load("data/raw/data.RData"))
```

When in doubt, google it !

```
# But.. how to load EXCEL FILES ?
# TODO Check stackoverflow / Google and load excel and csv file
# "load xlsx file R"
# "load csv file in R"
# can you copy paste ?? read.clipboard
```

Explore your data

Looking at your data before starting asking question is important to detect errors you might have made, wrong IDs, numbers of NA, wacky values… https://cran.r-project.org/web/packages/psych/vignettes/overview.pdf

```
# View(data)

# Psych has a lot of tools for exploratory analysis
library(psych)
describe(data)
```

```
##              vars   n    mean     sd  median trimmed    mad    min     max
## age             1 650   41.91  13.82   42.00   41.95  17.79   18.0   65.00
## isOlder*        2 650     NaN     NA      NA     NaN     NA    Inf    -Inf
## IQ              3 650   97.84  18.85   95.80   96.77  18.24   49.8  172.64
## responseTime    4 650 1979.54 401.66 1898.50 1951.40 412.90 1241.0 3723.00
## performance     5 650    0.45   0.20    0.46    0.45   0.20    0.0    1.00
## id*             6 650     NaN     NA      NA     NaN     NA    Inf    -Inf
## university      7 650    0.61   0.49    1.00    0.64   0.00    0.0    1.00
##                range  skew kurtosis    se
## age            47.00 -0.01    -1.19  0.54
## isOlder*        -Inf    NA       NA    NA
## IQ            122.84  0.59     0.51  0.74
## responseTime 2482.00  0.69     0.24 15.75
## performance     1.00  0.02    -0.44  0.01
## id*             -Inf    NA       NA    NA
## university      1.00 -0.47    -1.79  0.02
```

```
describeBy(data, group = "isOlder")
```

```
## $`FALSE`
##              vars   n    mean     sd  median trimmed    mad    min     max
## age             1 584   39.49  12.44   39.00   39.53  16.31   18.0   60.00
## isOlder*        2 584     NaN     NA      NA     NaN     NA    Inf    -Inf
## IQ              3 584   99.36  18.95   97.71   98.39  17.92   49.8  172.64
## responseTime    4 584 1911.34 352.71 1842.50 1886.99 345.45 1241.0 3433.00
## performance     5 584    0.46   0.20    0.47    0.46   0.21    0.0    1.00
## id*             6 584     NaN     NA      NA     NaN     NA    Inf    -Inf
## university      7 584    0.62   0.49    1.00    0.65   0.00    0.0    1.00
##                range  skew kurtosis    se
## age            42.00 -0.01    -1.18  0.51
## isOlder*        -Inf    NA       NA    NA
## IQ            122.84  0.52     0.45  0.78
## responseTime 2192.00  0.70     0.29 14.60
## performance     1.00 -0.03    -0.46  0.01
## id*             -Inf    NA       NA    NA
## university      1.00 -0.48    -1.77  0.02
##
## $`TRUE`
##              vars  n    mean     sd  median trimmed    mad     min     max
## age             1 66   63.30   1.39   63.50   63.37   2.22   61.00   65.00
## isOlder*        2 66     NaN     NA      NA     NaN     NA     Inf    -Inf
## IQ              3 66   84.39  10.99   84.02   84.28  14.04   66.41  105.98
## responseTime    4 66 2582.98 288.90 2539.50 2551.02 253.52 2177.00 3723.00
## performance     5 66    0.38   0.14    0.39    0.37   0.17    0.12    0.71
## id*             6 66     NaN     NA      NA     NaN     NA     Inf    -Inf
## university      7 66    0.59   0.50    1.00    0.61   0.00    0.00    1.00
##                range  skew kurtosis    se
## age             4.00 -0.37    -1.12  0.17
## isOlder*        -Inf    NA       NA    NA
## IQ             39.57  0.03    -1.21  1.35
## responseTime 1546.00  1.26     2.34 35.56
## performance     0.59  0.07    -0.80  0.02
## id*             -Inf    NA       NA    NA
## university      1.00 -0.36    -1.90  0.06
##
## attr(,"call")
## by.data.frame(data = x, INDICES = group, FUN = describe, type = type)
```

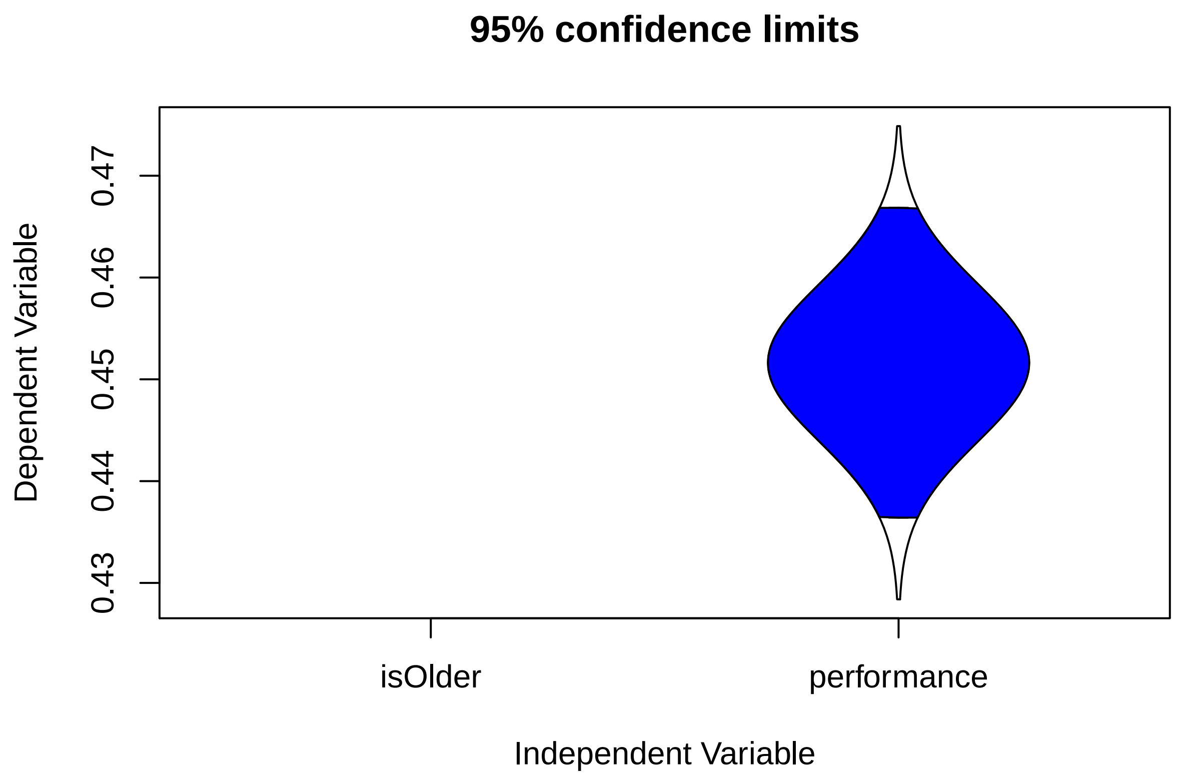
```
describeData(data)
```

```
## n.obs =  650 of which  650   are complete cases.   Number of variables =  7  of which all are numeri
##              variable # n.obs type          H1           H2           H3
## age                   1   650    1          63           62           32
## isOlder*              2   650    2        TRUE         TRUE        FALSE
## IQ                    3   650    1    74.44752     76.34852    112.18540
## responseTime          4   650    1        2222         2457         1916
## performance           5   650    1   0.3344507    0.5295560    0.4320815
## id*                   6   650    3 bV3jEslC8Tlv N4aDqBoVT5m3 78201FfgBn4l
## university            7   650    1           1            1            0
##                        H4           T1           T2           T3
## age                    44           46           59           37
## isOlder*            FALSE        FALSE        FALSE        FALSE
## IQ              121.94757     79.52374     82.16423     94.31278
## responseTime         1953         1853         2234         1691
## performance     0.6621028    0.5593342    0.4643382    0.2361071
## id*          bYh4gaq3Gqmx rO3GmcXNLca8 aIWNM14JYReD frU9oHEQakAW
## university              0            1            1            0
##                        T4
## age                    25
## isOlder*            FALSE
## IQ               89.51989
## responseTime         1593
## performance     0.5130609
## id*          J43nhoqij4Ij
## university              1
```

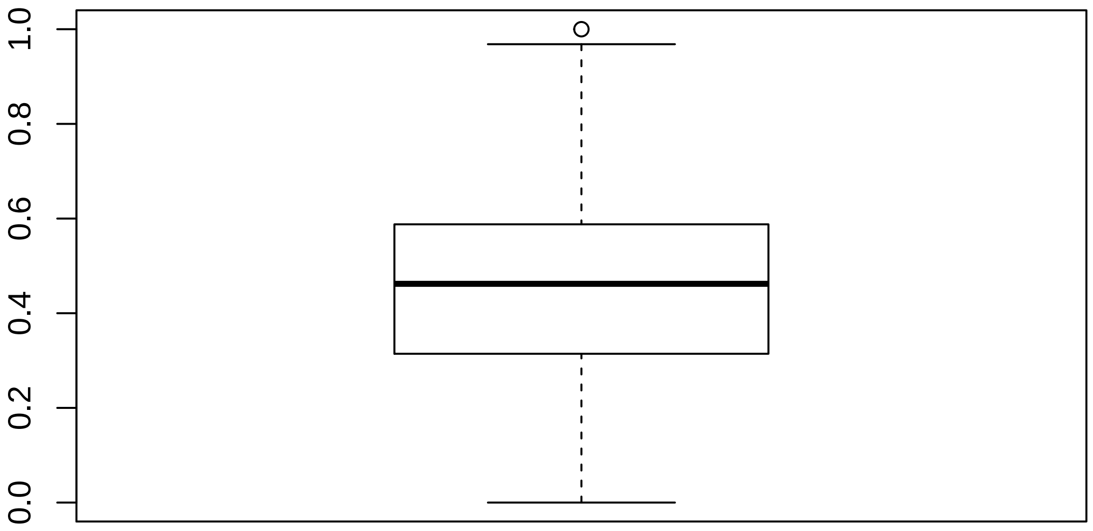
```r
head(data)
```

```
##   age isOlder        IQ responseTime performance           id university
## 1  63    TRUE  74.44752         2222   0.3344507 bV3jEslC8Tlv          1
## 2  62    TRUE  76.34852         2457   0.5295560 N4aDqBoVT5m3          1
## 3  32   FALSE 112.18540         1916   0.4320815 78201FfgBn4l          0
## 4  44   FALSE 121.94757         1953   0.6621028 bYh4gaq3Gqmx          0
## 5  31   FALSE 128.14781         1751   0.3728825 WlKPPvN7EosE          0
## 6  29   FALSE 116.20584         1785   0.8645725 RPuMuh65ISC3          1
```

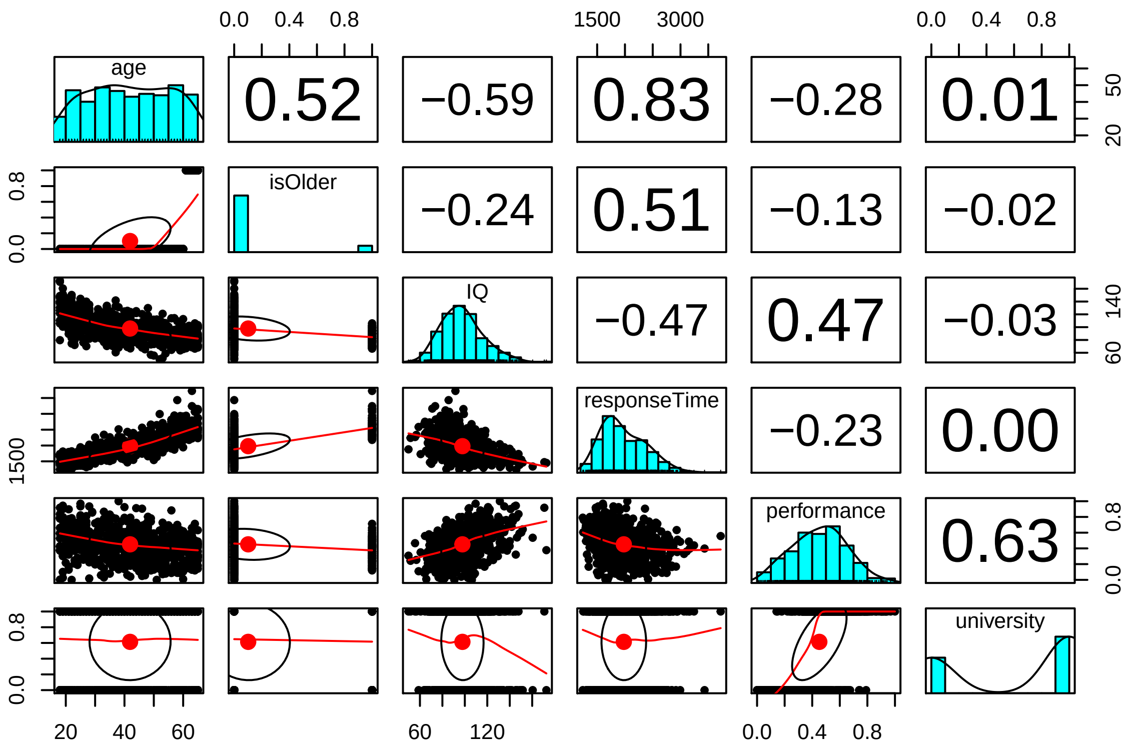
```r
# Some quick plots
error.bars(data[, c("isOlder", "performance")])
```



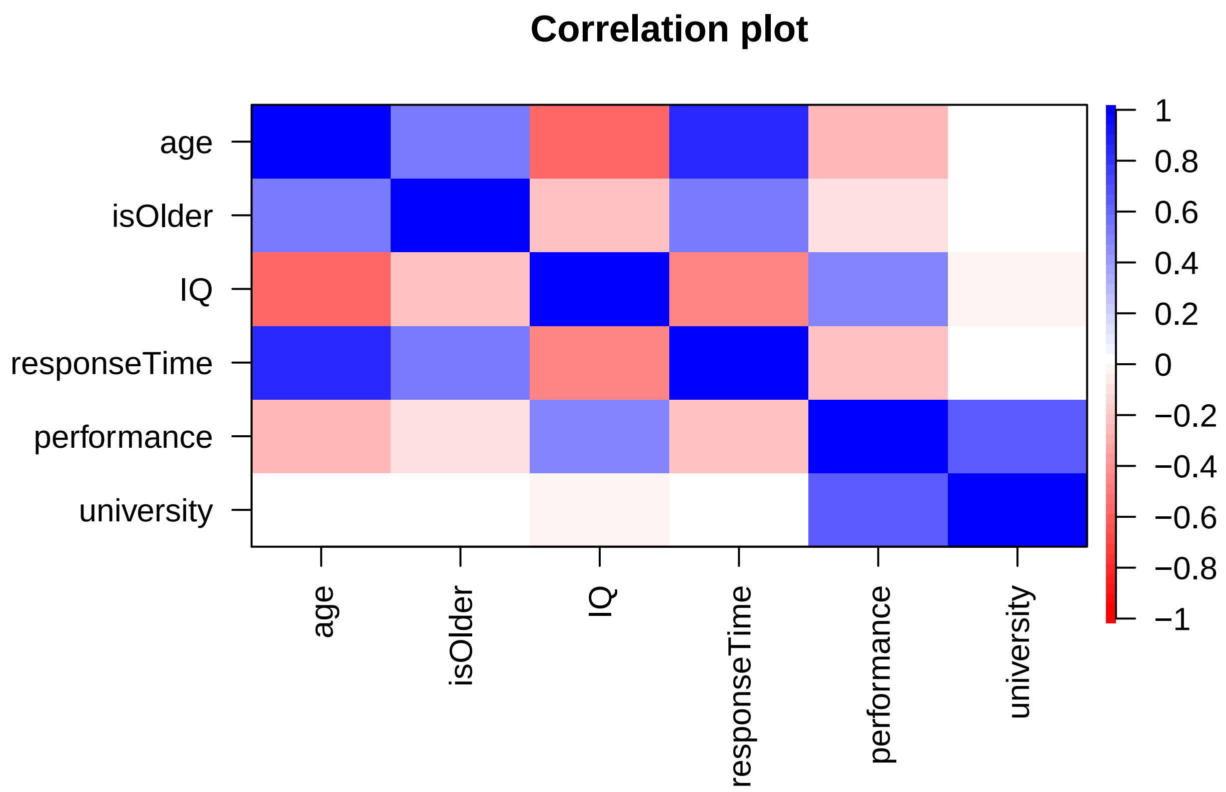
```r
boxplot(data[, c("performance")])
```



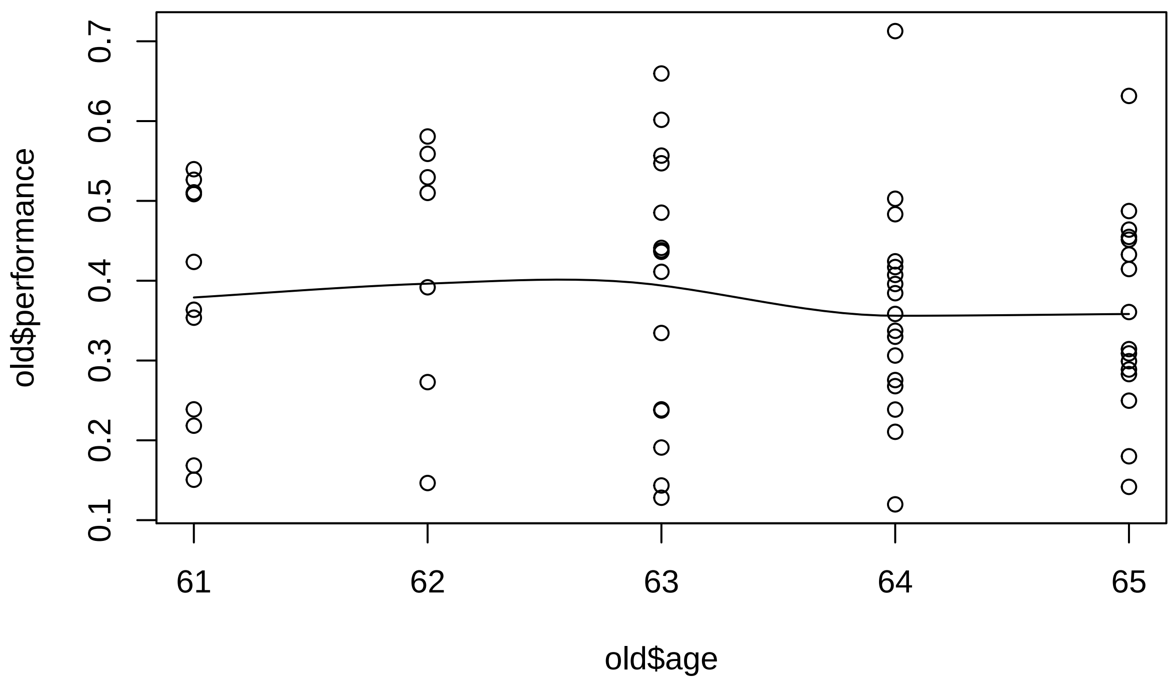
```r
pairs.panels(data[-6])
```



```r
corPlot(data.matrix(data[-6]))
```

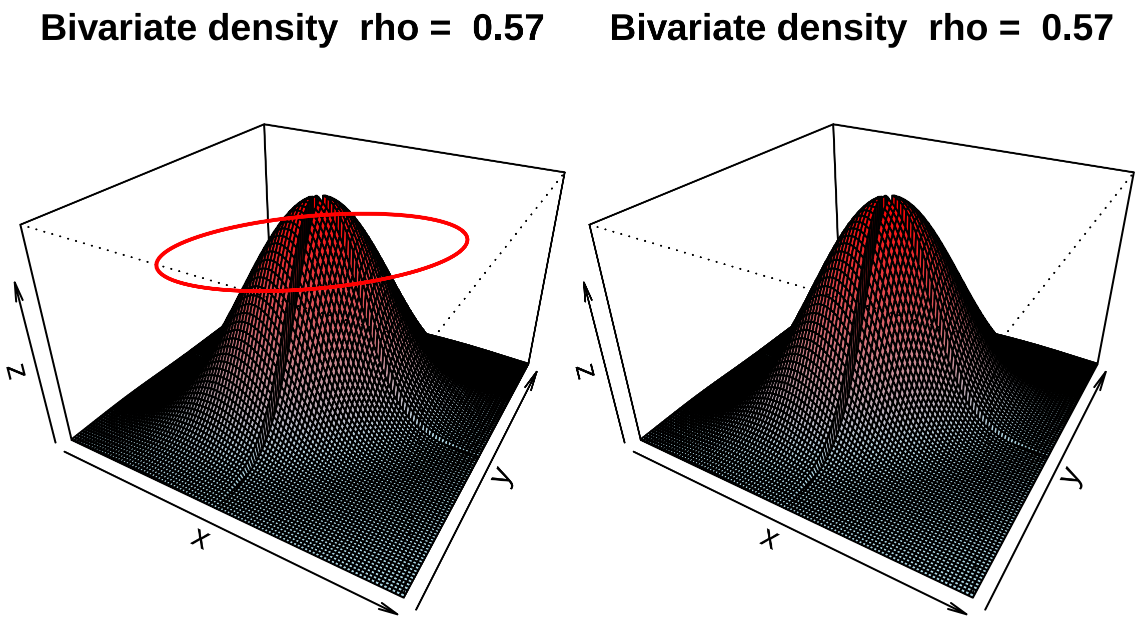


```r
# Look for outliers
# outlier(data[, c("performance", "IQ", "responseTime")],cex=.8)

# Filter your data
dataScrubbed = scrub(data,2:3,min=c(88,1300), max=c(115, 1600), newvalue=NA)
old = subset(data,data$isOlder==1)
scatter.smooth(old$age, old$performance)
```



Correlations

```
# What is a correlation ?
draw.cor(expand=20,cuts=c(0,0),r = 0.57)
```



```
# regression
regression = lm(formula = performance ~ IQ + age + university + IQ:age, data = data, na.action = na.omi
summary(regression)
```

```
##
## Call:
## lm(formula = performance ~ IQ + age + university + IQ:age, data = data,
##     na.action = na.omit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.43909 -0.07859  0.00311  0.07959  0.35653
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.555e-01  7.974e-02  -3.204  0.00142 **
## IQ           5.597e-03  7.357e-04   7.608 9.91e-14 ***
## age          1.445e-03  1.840e-03   0.785  0.43251
## university   2.599e-01  9.781e-03  26.568  < 2e-16 ***
## IQ:age      -1.534e-05  1.882e-05  -0.815  0.41556
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1211 on 645 degrees of freedom
## Multiple R-squared:  0.6263, Adjusted R-squared:  0.624
## F-statistic: 270.3 on 4 and 645 DF,  p-value: < 2.2e-16
```

```
# But is this model the good one ?
regression2 = lm(formula = performance ~ IQ * age , data = data, na.action = na.omit)
summary(regression2)
```

```
##
## Call:
## lm(formula = performance ~ IQ * age, data = data, na.action = na.omit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.49483 -0.12879  0.01883  0.12809  0.45783
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.383e-02  1.142e-01   0.384    0.701
## IQ           4.239e-03  1.061e-03   3.994 7.24e-05 ***
## age         -1.703e-03  2.656e-03  -0.641    0.522
## IQ:age       1.633e-05  2.717e-05   0.601    0.548
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1751 on 646 degrees of freedom
## Multiple R-squared:  0.2174, Adjusted R-squared:  0.2138
## F-statistic: 59.83 on 3 and 646 DF,  p-value: < 2.2e-16
```

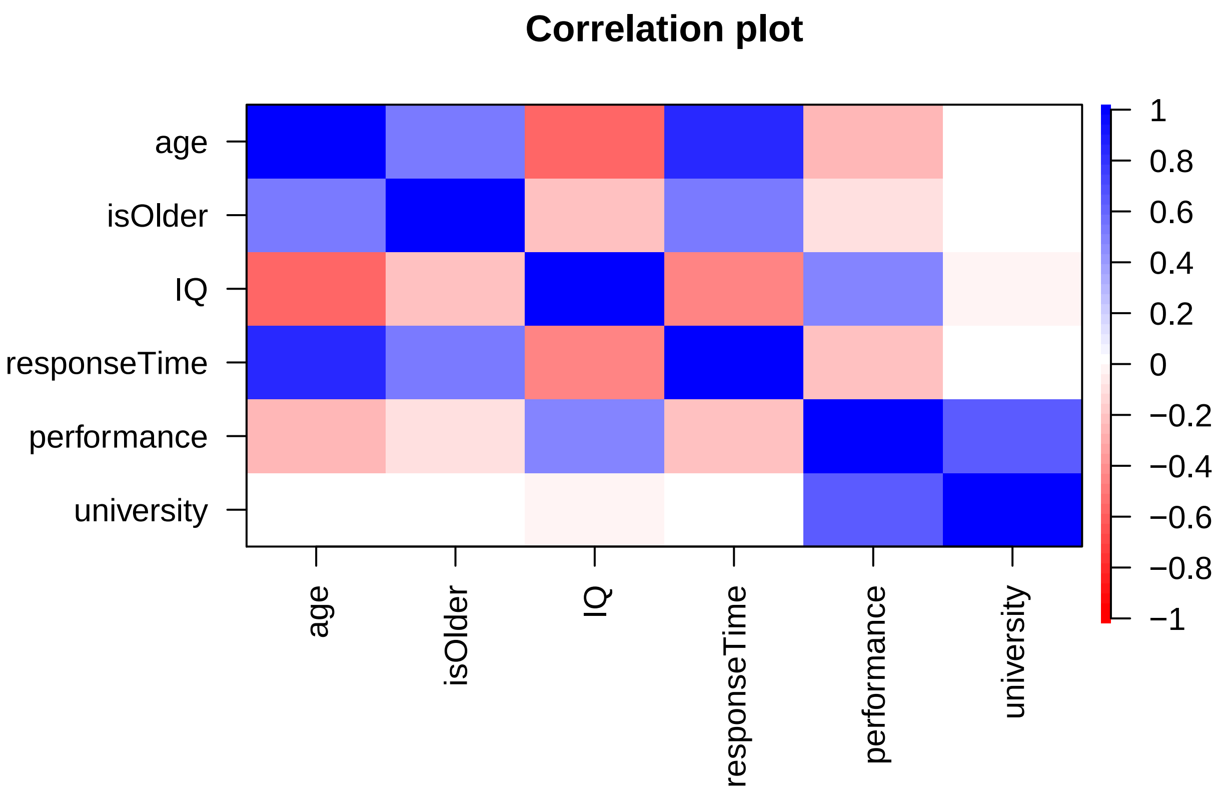
```
# Anova(model1,model2) allows to compare models
anova(regression,regression2)
```

```
## Analysis of Variance Table
##
## Model 1: performance ~ IQ + age + university + IQ:age
## Model 2: performance ~ IQ * age
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1    645  9.4604
## 2    646 19.8137 -1   -10.353 705.87 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

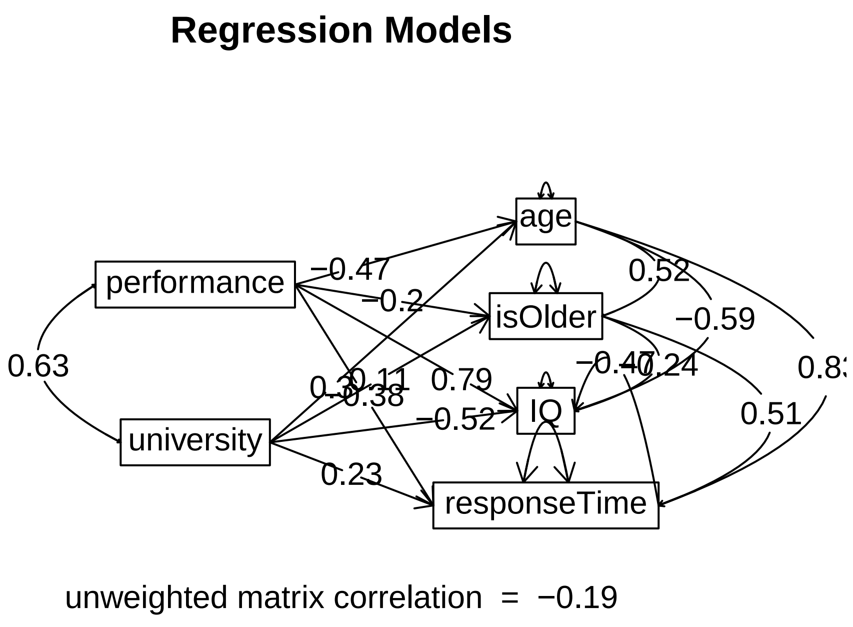
```
# Other way to analyses correlation (usually through covariance matrix)
# correlation and analysis
corr.test(data.matrix(data[-6]))
```

```
## Call:corr.test(x = data.matrix(data[-6]))
## Correlation matrix
##               age isOlder    IQ responseTime performance university
## age          1.00    0.52 -0.59         0.83       -0.28       0.01
## isOlder      0.52    1.00 -0.24         0.51       -0.13      -0.02
## IQ          -0.59   -0.24  1.00        -0.47        0.47      -0.03
## responseTime  0.83    0.51 -0.47         1.00       -0.23       0.00
## performance -0.28   -0.13  0.47        -0.23        1.00       0.63
## university   0.01   -0.02 -0.03         0.00        0.63       1.00
## Sample Size
## [1] 650
## Probability values (Entries above the diagonal are adjusted for multiple tests.)
##               age isOlder  IQ responseTime performance university
## age          0.00    0.00 0.0         0.00           0          1
## isOlder      0.00    0.00 0.0         0.00           0          1
## IQ           0.00    0.00 0.0         0.00           0          1
## responseTime 0.00    0.00 0.0         0.00           0          1
## performance  0.00    0.00 0.0         0.00          0          0
## university   0.88    0.69 0.5         0.92          0          0
##
##  To see confidence intervals of the correlations, print with the short=FALSE option
```

```
corPlot(data.matrix(data[-6]))
```



```
# Multivariate correlation predict y columns with x
setCor(y = 1:4,x=c(5,7),data=data)
```

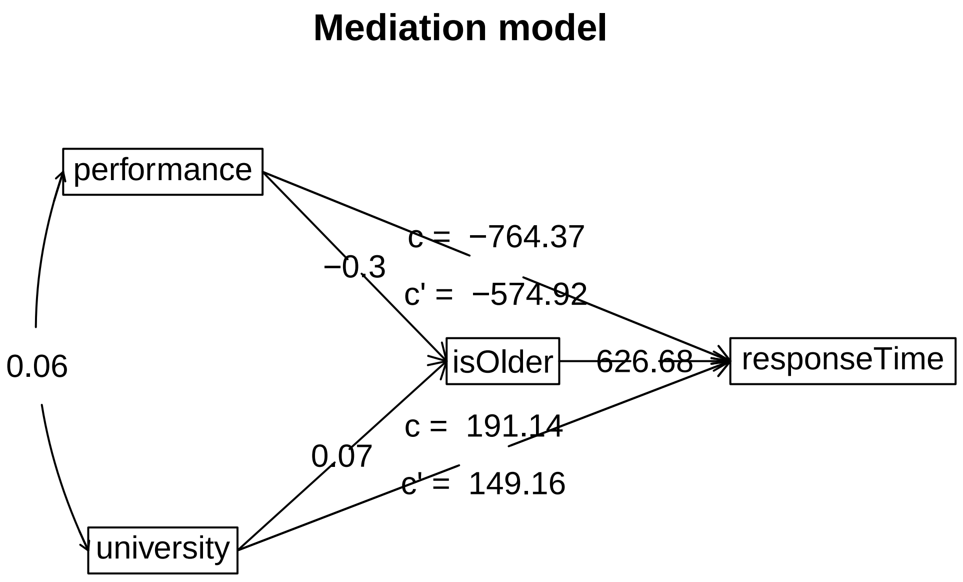


```
## Call: setCor(y = 1:4, x = c(5, 7), data = data)
```

```
##
## Multiple Regression from raw data
##
## Beta weights
##               age isOlder    IQ responseTime
## performance -0.47   -0.20  0.79        -0.38
## university   0.30    0.11 -0.52         0.23
##
## Multiple R
##          age      isOlder           IQ responseTime
##         0.37         0.15         0.62         0.29
## multiple R2
##          age      isOlder           IQ responseTime
##        0.134        0.024        0.384        0.086
##
##  Unweighted multiple R
##          age      isOlder           IQ responseTime
##         0.16         0.08         0.27         0.13
##  Unweighted multiple R2
##          age      isOlder           IQ responseTime
##         0.03         0.01         0.07         0.02
##
##  SE of Beta weights
##              age isOlder   IQ responseTime
## performance 0.05    0.05 0.04         0.05
## university  0.05    0.05 0.04         0.05
##
##  t of Beta Weights
##               age isOlder     IQ responseTime
## performance -10.0   -3.96  20.07        -7.79
## university    6.4    2.17 -13.26         4.81
##
## Probability of t <
##                 age isOlder IQ responseTime
## performance 0.0e+00 8.2e-05  0      2.7e-14
## university  3.1e-10 3.1e-02  0      1.9e-06
##
##  Shrunken R2
##          age      isOlder           IQ responseTime
##        0.131        0.021        0.382        0.083
##
## Standard Error of R2
##          age      isOlder           IQ responseTime
##        0.025        0.012        0.030        0.021
##
## F
##          age      isOlder           IQ responseTime
##        50.00         7.93       201.84        30.35
##
## Probability of F <
##          age      isOlder           IQ responseTime
##     0.00e+00     3.95e-04     0.00e+00     2.52e-13
##
##  degrees of freedom of regression
## [1]   2 647
##
## Various estimates of between set correlations
## Squared Canonical Correlations
## [1] 0.38424 0.00067
## Chisq of canonical correlations
## [1] 313.01   0.43
##
##  Average squared canonical correlation =  0.19
##  Cohen's Set Correlation R2 =  0.38
##  Shrunken Set Correlation R2 =  0.38
##  F and df of Cohen's Set Correlation  43.97 8 1280
## Unweighted correlation between the two sets =  -0.19
```

Mediation analysis

```r
mediate(y = 4, x = c(5,7), m = 2, data = data)
```

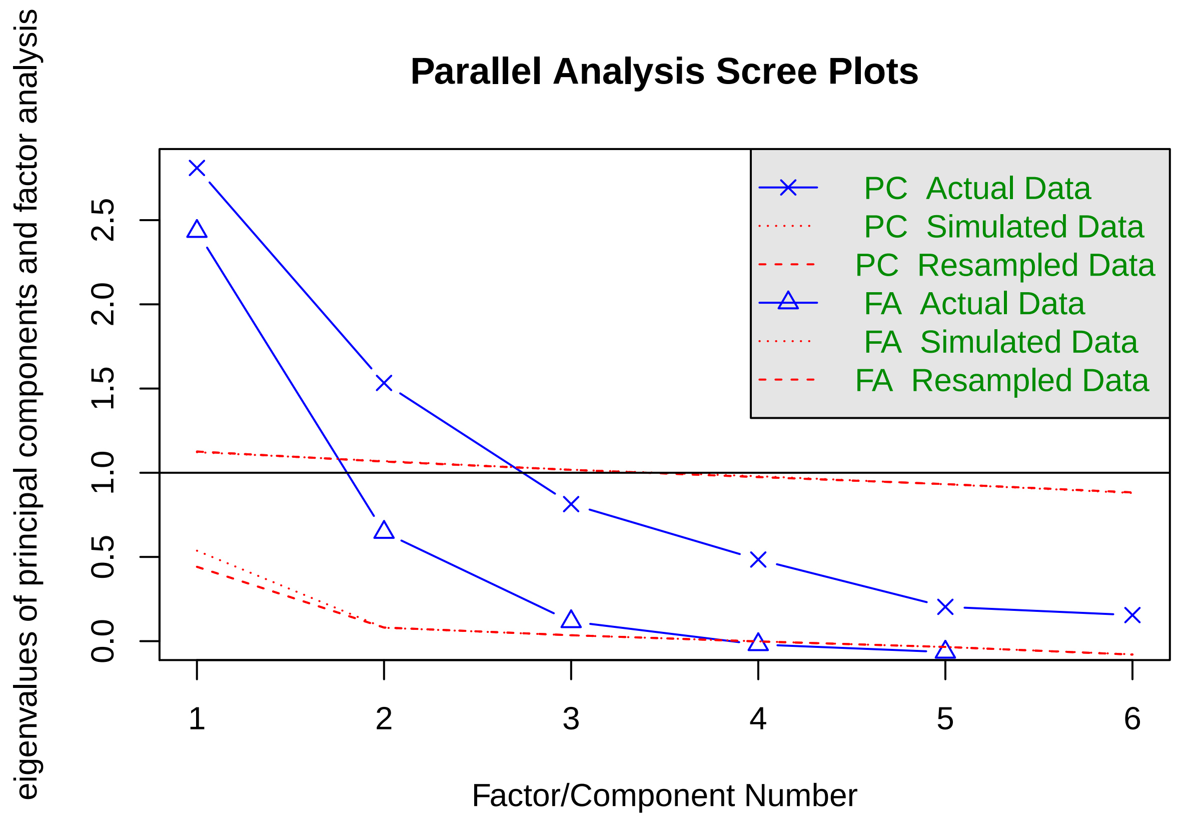


```
## Call: mediate(y = 4, x = c(5, 7), m = 2, data = data)
##
## The DV (Y) was  responseTime . The IV (X) was  performance university . The mediating variable(s) =
##
## Total Direct effect(c) of  performance  on  responseTime  =  -764.37   S.E. =  98.12  t direct =  -7
## Direct effect (c') of  performance  on  responseTime  removing  isOlder  =  -574.92   S.E. =  86.78
## Indirect effect (ab) of  performance  on  responseTime  through  isOlder   =  -189.45
## Mean bootstrapped indirect effect =  -189.62  with standard error =  40.62  Lower CI =  -272.72    U
##
## Total Direct effect(c) of  university  on  responseTime  =  191.14   S.E. =  39.77  t direct =  4.81
## Direct effect (c') of  university  on  NA  removing  isOlder  =  149.16   S.E. =  34.89  t direct =
## Indirect effect (ab) of  university  on  responseTime  through  isOlder   =  41.99
## Mean bootstrapped indirect effect =  -189.62  with standard error =  40.62  Lower CI =  4.79    Uppe
## R2 of model =  0.3
##  To see the longer output, specify short = FALSE in the print statement
##
```

```
##  Full output
##
##  Total effect estimates (c)
##              responseTime    se     t     Prob
## performance      -764.37 98.12 -7.79 2.66e-14
## university        191.14 39.77  4.81 1.92e-06
##
## Direct effect estimates     (c')
##              responseTime    se     t     Prob
## performance      -574.92 86.78 -6.63 7.32e-11
## university        149.16 34.89  4.28 2.19e-05
##
##  'a'  effect estimates
##              isOlder   se     t     Prob
## performance   -0.30 0.08 -3.96 8.25e-05
## university     0.07 0.03  2.17 3.06e-02
##
##  'b'  effect estimates
##          responseTime    se     t Prob
## isOlder       626.68 44.18 14.18    0
##
##  'ab'  effect estimates
##              responseTime    boot    sd   lower   upper
## performance      -189.45 -189.62 40.62 -272.72 -113.22
## university         41.99   42.52 19.68    4.79   81.74
```

Dimensionality reduction

```
# How many components should you expect ?
dataWithoutId = data[-6]
fa.parallel(dataWithoutId)
```



```
## Parallel analysis suggests that the number of factors =  3  and the number of components =  2
```

```
# Data loads on variable
principal(dataWithoutId, nfactors = 2)
```

```
## Principal Components Analysis
## Call: principal(r = dataWithoutId, nfactors = 2)
## Standardized loadings (pattern matrix) based upon correlation matrix
##                RC1   RC2   h2   u2 com
## age           0.93 -0.06 0.86 0.14 1.0
## isOlder       0.68  0.04 0.47 0.53 1.0
## IQ           -0.69  0.25 0.54 0.46 1.3
## responseTime  0.89 -0.03 0.79 0.21 1.0
## performance  -0.28  0.89 0.87 0.13 1.2
## university    0.11  0.89 0.81 0.19 1.0
##
##                        RC1  RC2
## SS loadings           2.68 1.66
## Proportion Var        0.45 0.28
## Cumulative Var        0.45 0.72
## Proportion Explained  0.62 0.38
## Cumulative Proportion 0.62 1.00
##
## Mean item complexity =  1.1
## Test of the hypothesis that 2 components are sufficient.
##
## The root mean square of the residuals (RMSR) is  0.11
##  with the empirical chi square  230.19  with prob <  1.2e-48
##
```

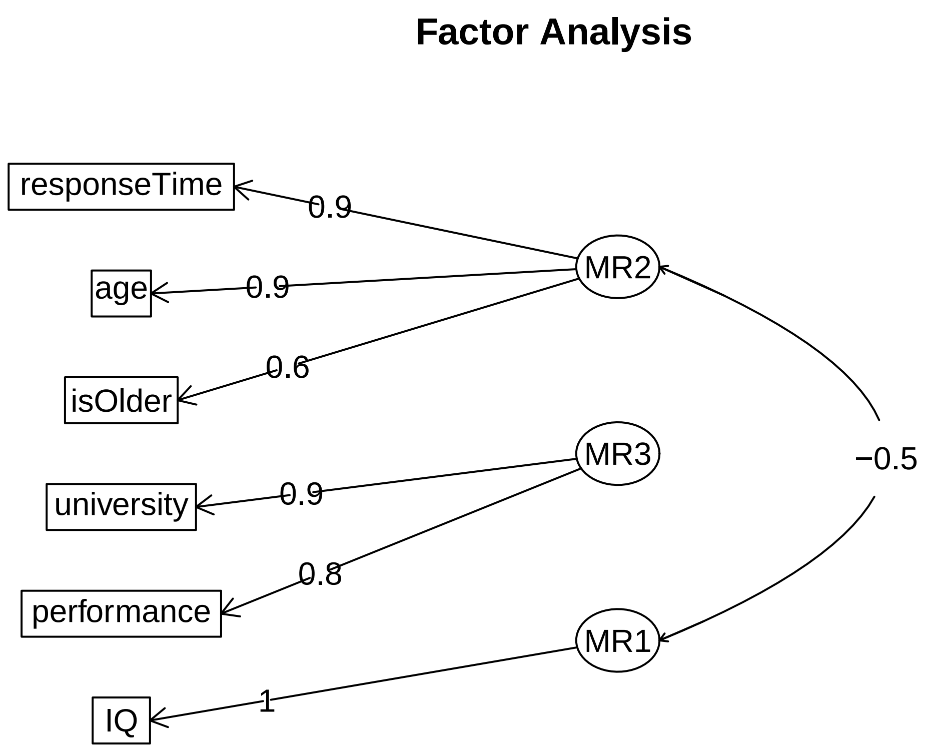
```
## Fit based upon off diagonal values = 0.93
```

```r
# Latent variable loads on data (age, IQ, university)
factanal(dataWithoutId, factors = 3)
```

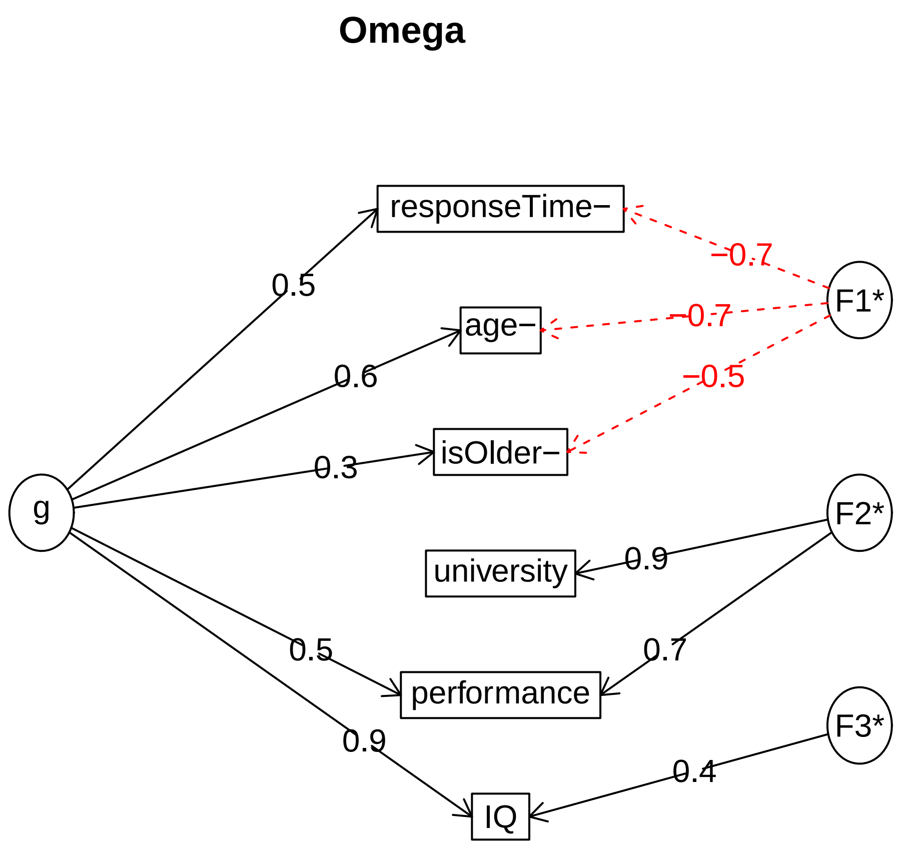
```
##
## Call:
## factanal(x = dataWithoutId, factors = 3)
##
## Uniquenesses:
##          age      isOlder           IQ responseTime  performance
##        0.118        0.671        0.059        0.208        0.005
##   university
##        0.462
##
## Loadings:
##              Factor1 Factor2 Factor3
## age           0.880          -0.326
## isOlder       0.570
## IQ           -0.329           0.908
## responseTime  0.865          -0.205
## performance  -0.136   0.915   0.374
## university            0.726
##
##                Factor1 Factor2 Factor3
## SS loadings      1.973   1.377   1.126
## Proportion Var   0.329   0.230   0.188
## Cumulative Var   0.329   0.558   0.746
##
## The degrees of freedom for the model is 0 and the fit was 0
```

```r
fa.diagram(fa(dataWithoutId, nfactors = 3))
```

```
## Loading required namespace: GPArotation
```

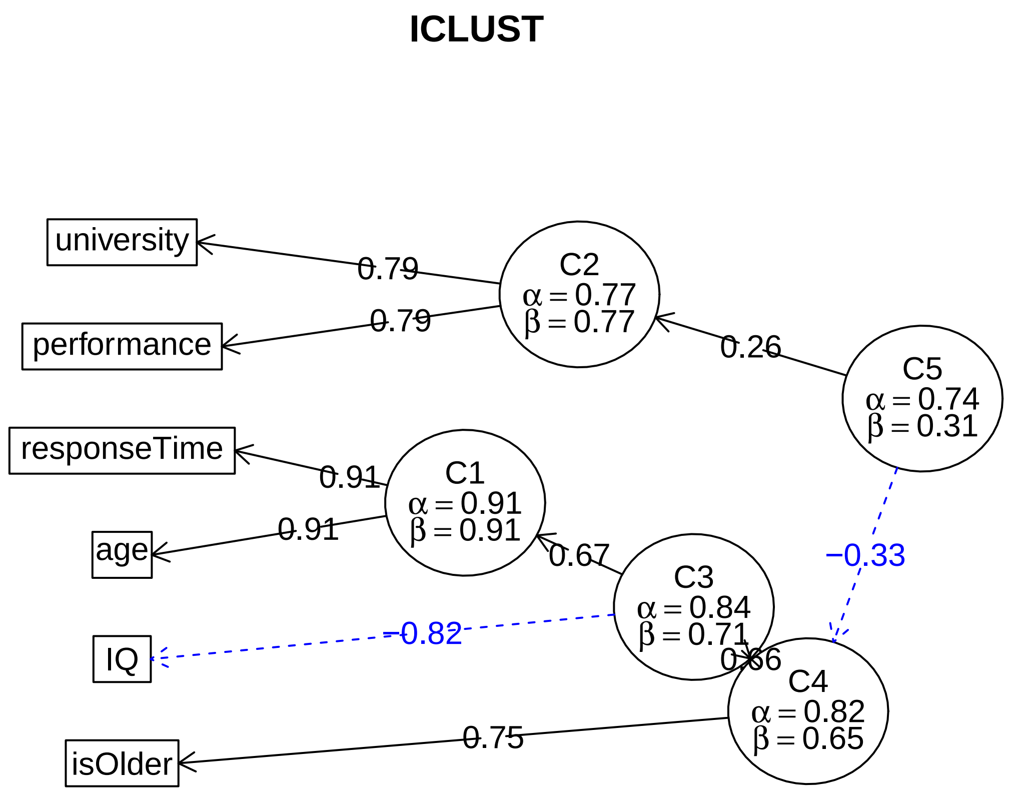


```
# Hierarchical
#install.packages("GPArotation")
library(GPArotation)
omega(dataWithoutId)
```



```
## Omega
## Call: omega(m = dataWithoutId)
## Alpha:                 0.74
## G.6:                   0.84
## Omega Hierarchical:    0.46
## Omega H asymptotic:    0.51
## Omega Total            0.9
##
## Schmid Leiman Factor loadings greater than  0.2
##                    g   F1*   F2*   F3*   h2   u2   p2
## age-            0.59 -0.73             0.88 0.12 0.40
## isOlder-        0.25 -0.51             0.33 0.67 0.20
## IQ              0.91              0.37 0.98 0.02 0.86
## responseTime-   0.49 -0.75             0.79 0.21 0.30
## performance     0.46        0.74       0.78 0.22 0.27
## university                  0.86       0.74 0.26 0.00
##
## With eigenvalues of:
##    g  F1*  F2*  F3*
## 1.70 1.35 1.28 0.16
##
## general/max  1.26   max/min =   8.24
## mean percent general =  0.34    with sd =  0.29 and cv of  0.85
## Explained Common Variance of the general factor =  0.38
##
## The degrees of freedom are 0  and the fit is  0
## The number of observations was  650  with Chi Square =  0.02  with prob <  NA
## The root mean square of the residuals is  0
## The df corrected root mean square of the residuals is  NA
##
## Compare this with the adequacy of just a general factor and no group factors
## The degrees of freedom for just the general factor are 9  and the fit is  1.83
## The number of observations was  650  with Chi Square =  1179.88  with prob <  2.7e-248
## The root mean square of the residuals is  0.26
## The df corrected root mean square of the residuals is  0.33
##
## RMSEA index =  0.2  and the 90 % confidence intervals are  0.2 0.469
## BIC =  1121.59
##
## Measures of factor score adequacy
##                                                  g  F1*  F2*   F3*
## Correlation of scores with factors            0.92 0.90 0.91  0.45
## Multiple R square of scores with factors      0.84 0.81 0.84  0.20
## Minimum correlation of factor score estimates 0.68 0.63 0.67 -0.60
##
##  Total, General and Subset omega for each subset
##                                                  g  F1*  F2*  F3*
## Omega total for total scores and subscales    0.90 0.85 0.85 0.97
## Omega general for total scores and subscales  0.46 0.26 0.06 0.84
## Omega group for total scores and subscales    0.42 0.59 0.79 0.14
```

```
#Clusters
iclust(dataWithoutId)
```



```
## ICLUST (Item Cluster Analysis)
## Call: iclust(r.mat = dataWithoutId)
##
## Purified Alpha:
## [1] 0.74
##
## G6* reliability:
## [1] 0.62
##
## Original Beta:
## [1] 0.31
##
## Cluster size:
## [1] 6
##
## Item by Cluster Structure matrix:
##              [,1]
## age          -0.82
## isOlder      -0.47
## IQ            0.63
## responseTime -0.75
## performance   0.65
## university    0.31
##
## With eigenvalues of:
## [1] 2.4
##
## Purified scale intercorrelations
##  reliabilities on diagonal
##  correlations corrected for attenuation above diagonal:
##      [,1]
## [1,] 0.74
##
## Cluster fit =  0.67   Pattern fit =  0.89  RMSR =  0.2
```

```
# structural equatiom modelin
sem = esem(r = cor(dataWithoutId), varsX = c(5,6), varsY = 1:4, nfX = 2, nfY = 1,
n.obs = 650, plot = FALSE)
```

```
## The estimated weights for the factor scores are probably incorrect.  Try a different factor extracti
```

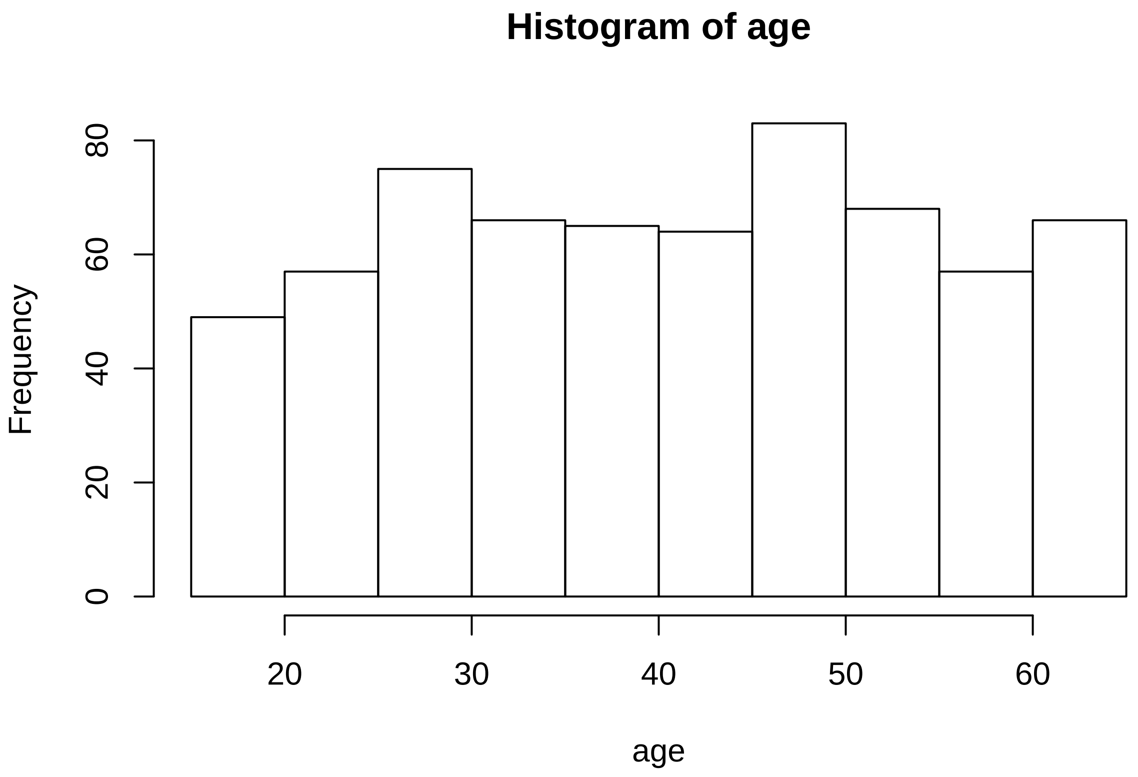
```
print(sem)
```

```
## Exploratory Structural Equation Modeling  Analysis using method =  minres
## Call: esem(r = cor(dataWithoutId), varsX = c(5, 6), varsY = 1:4, nfX = 2,
##     nfY = 1, n.obs = 650, plot = FALSE)
##
## For the 'X' set:
##                 MR1  MR2
## IQ            -0.84 0.47
## responseTime  0.84 0.47
##
## For the 'Y' set:
##                 MR1
## performance  1.00
## university   0.63
## age         -0.28
## isOlder     -0.13
##
## Correlations between the X and Y sets.
##       X1   X2    Y1
## X1  1.00 0.00 -0.46
## X2  0.00 1.00  0.12
## Y1 -0.46 0.12  1.00
##
## The degrees of freedom for the null model are  30  and the empirical chi square  function was  3385.
## The degrees of freedom for the model are 0  and the empirical chi square function was  25.56
##   with prob <  NA
##
## The root mean square of the residuals (RMSR) is  0.04
## The df corrected root mean square of the residuals is  NA
##  with the empirical chi square  25.56  with prob <  NA
## The total number of observations was  650  with fitted Chi Square =  541.31  with prob <  NA
##
## Empirical BIC =  NA
## ESABIC =  NA
## Fit based upon off diagonal values = 0.99
## To see the item loadings for the X and Y sets combined, and the associated fa output, print with  sh
```

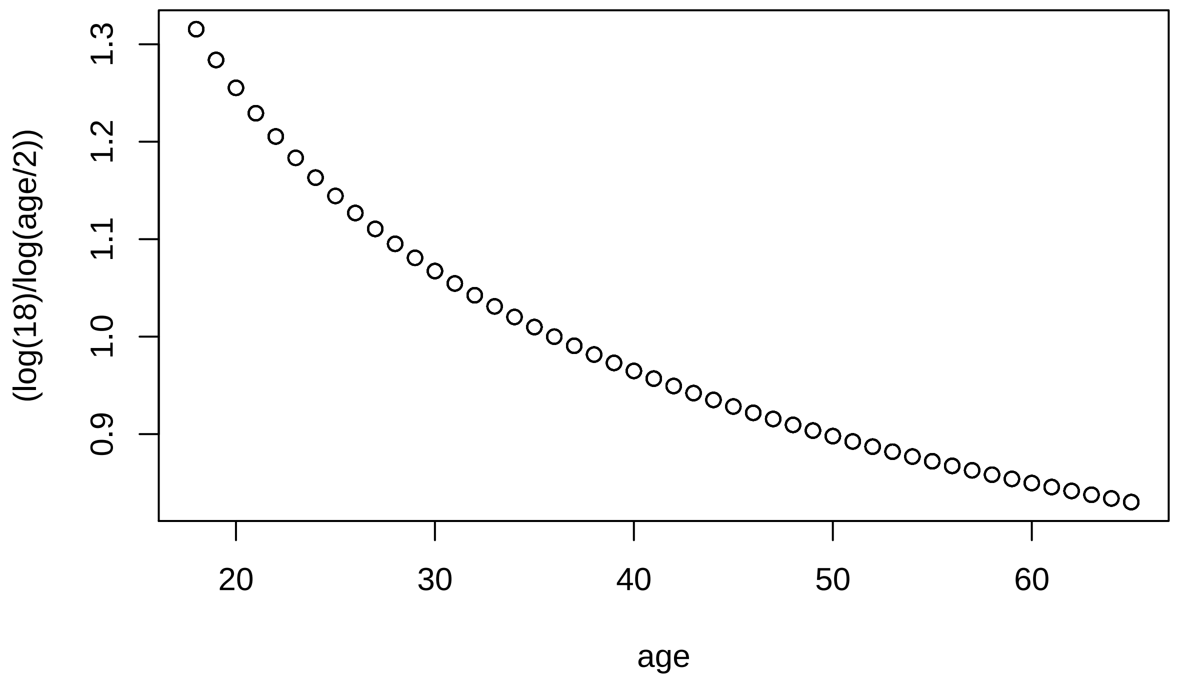
```
esem.diagram(sem)
```

How to generate fake data

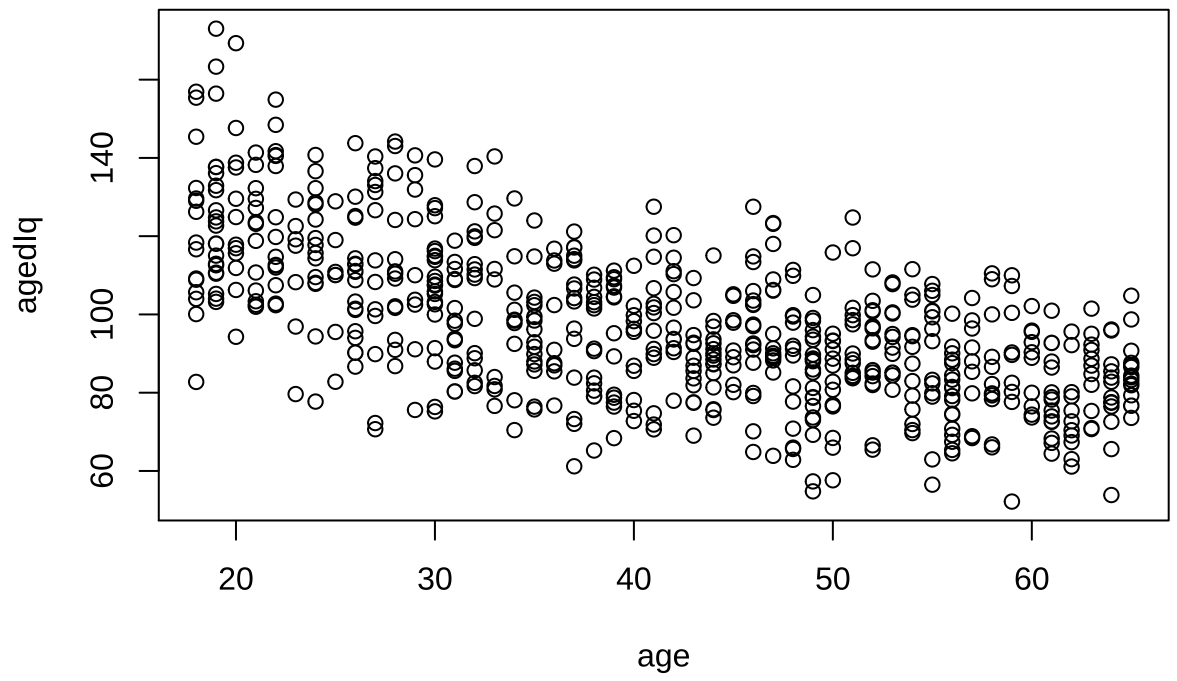
```r
# Number of subjects
N = 650

# Sample from a list
age = sample(18:65, N, replace = T)

hist(age)
```



```r
# Sample from gaussian distribution to generate IQ
baseIq = rnorm(N, 100, 15)
agedIq = baseIq * (log(18)/log(age/2))
data = data.frame(age = age, isOlder = age>60, IQ = agedIq)

plot(age, (log(18)/log(age/2)))
```

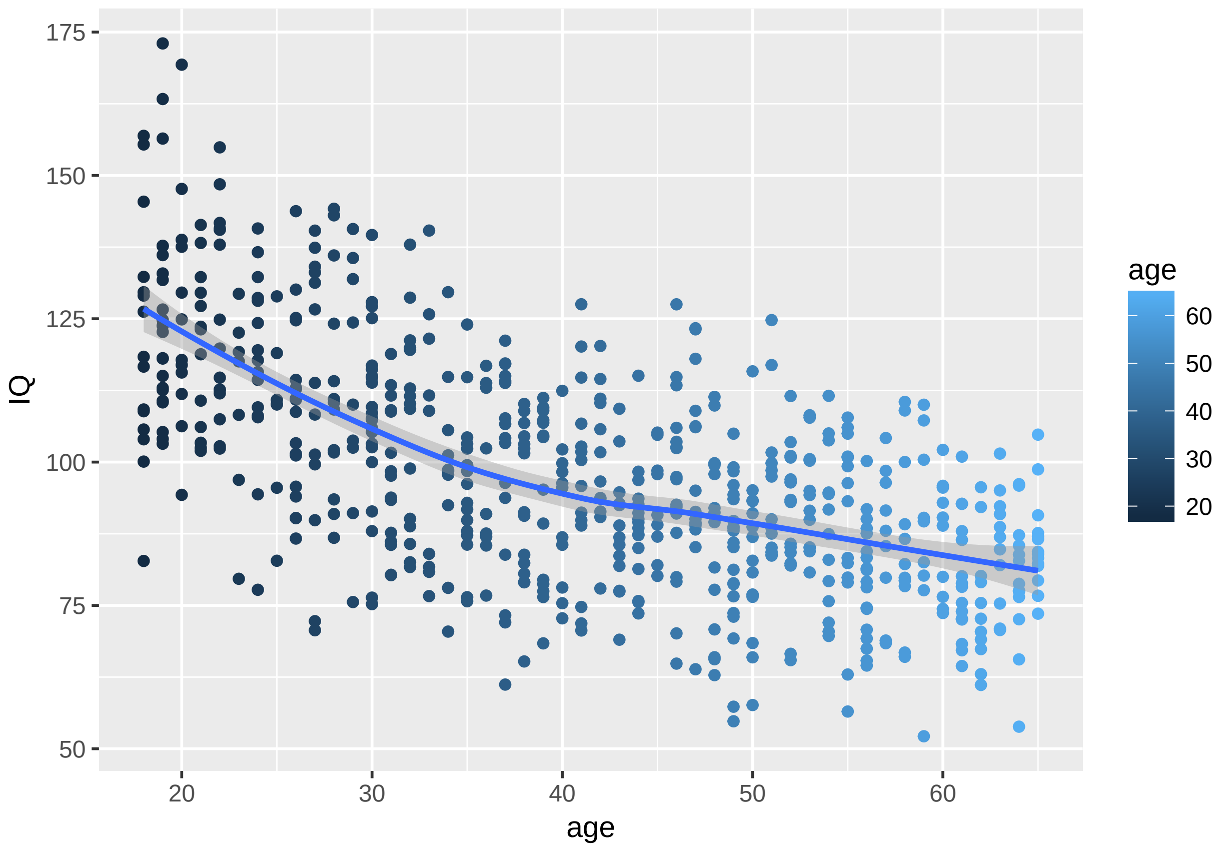


```r
plot(age,agedIq)
```

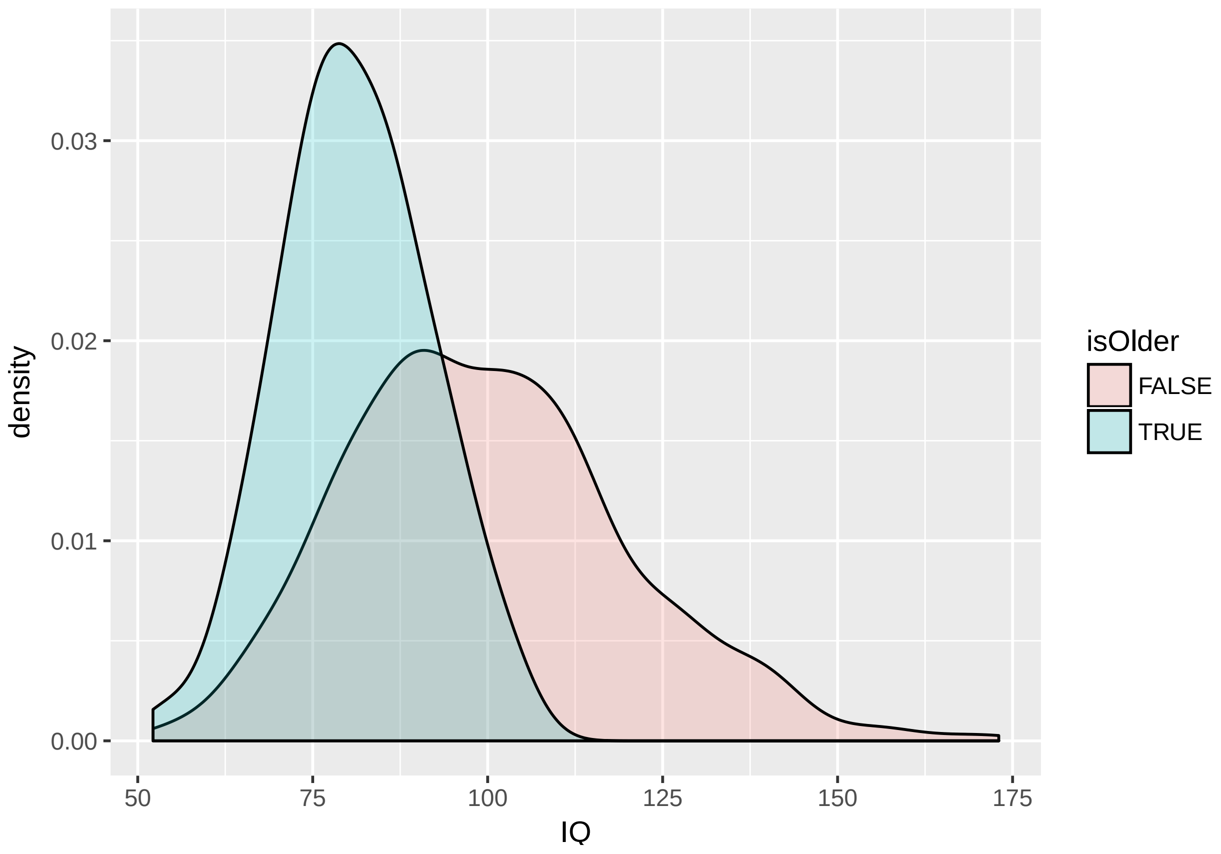


```r
# A slightly more beatifull way of plotting
# install.packages("ggplot")
library(ggplot2)

# Line plot + prediction error
ggplot(data, aes(x = age, y = IQ, color=age)) + geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```



```
# Histograms
ggplot(data, aes(IQ, ..density.., fill = isOlder)) +
geom_density(alpha=0.2)
```



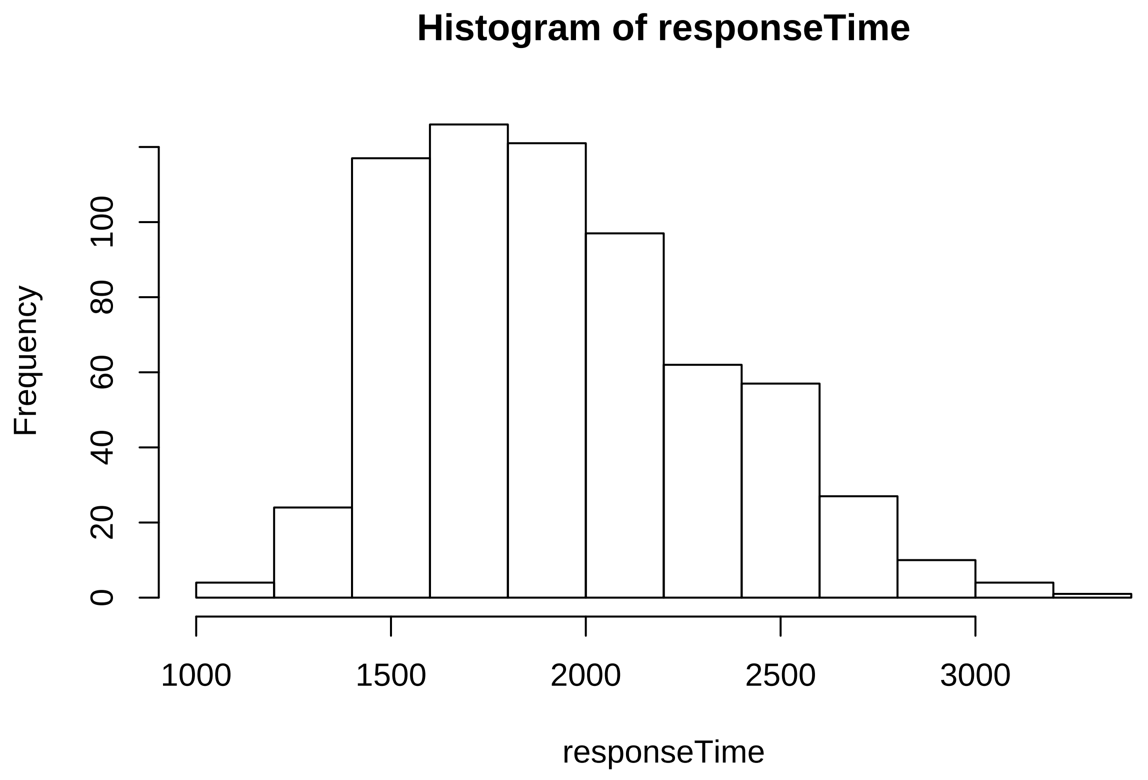
```r
# Sample from a custom distribution
# For example the Exponential
# P(x) = lambda * exp (-lambda*x)
# lambda = 1 / mean
# CDF = 1 - exp(-lambda*x) ==> x = - ln (1-CDF) / lambda
mean = 1500
lambda = 1 / mean
CDF = sample(0:100, N, replace = T) / 100
x = -log(1-CDF) / lambda
```

Complex distribution

Sample from from complex distribution such as Diffusion Model for Response Time Lets image you are slower with age !

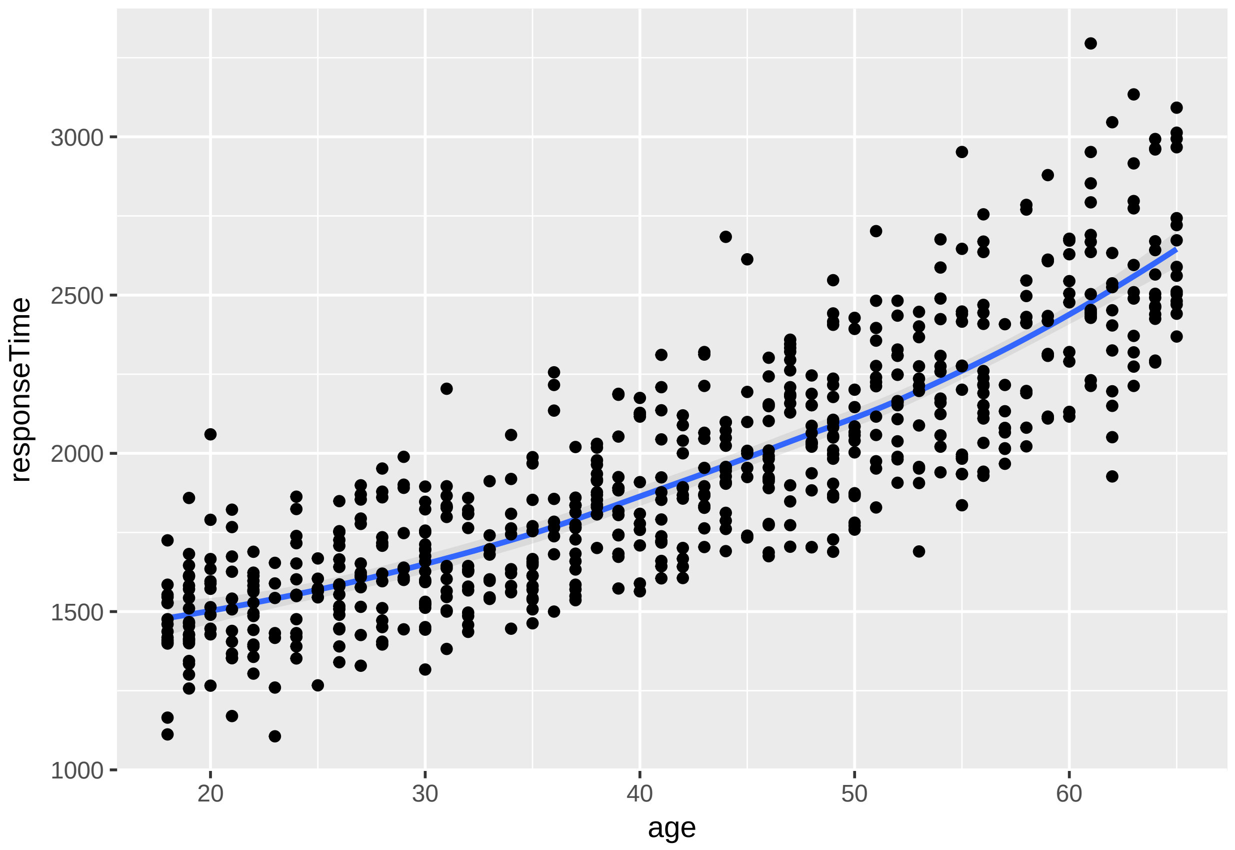
```r
# Here bias depends on age
responseTime = rep(NA, N)
currentAccumulators = rep(0,N)
bias = sample(c(-1,1), N, T)+rnorm(N,0,0.1)
threshold = 1000
for(i in 1:5000) {
  currentAccumulators = currentAccumulators + bias * abs(rnorm(N, 0,1-(0.008*age)))
  indices = which(abs(currentAccumulators) > threshold)
  newIndices = indices[!indices%in%which(!is.na(responseTime))]
  responseTime[newIndices] = i
}
```

```r
hist(responseTime)
```



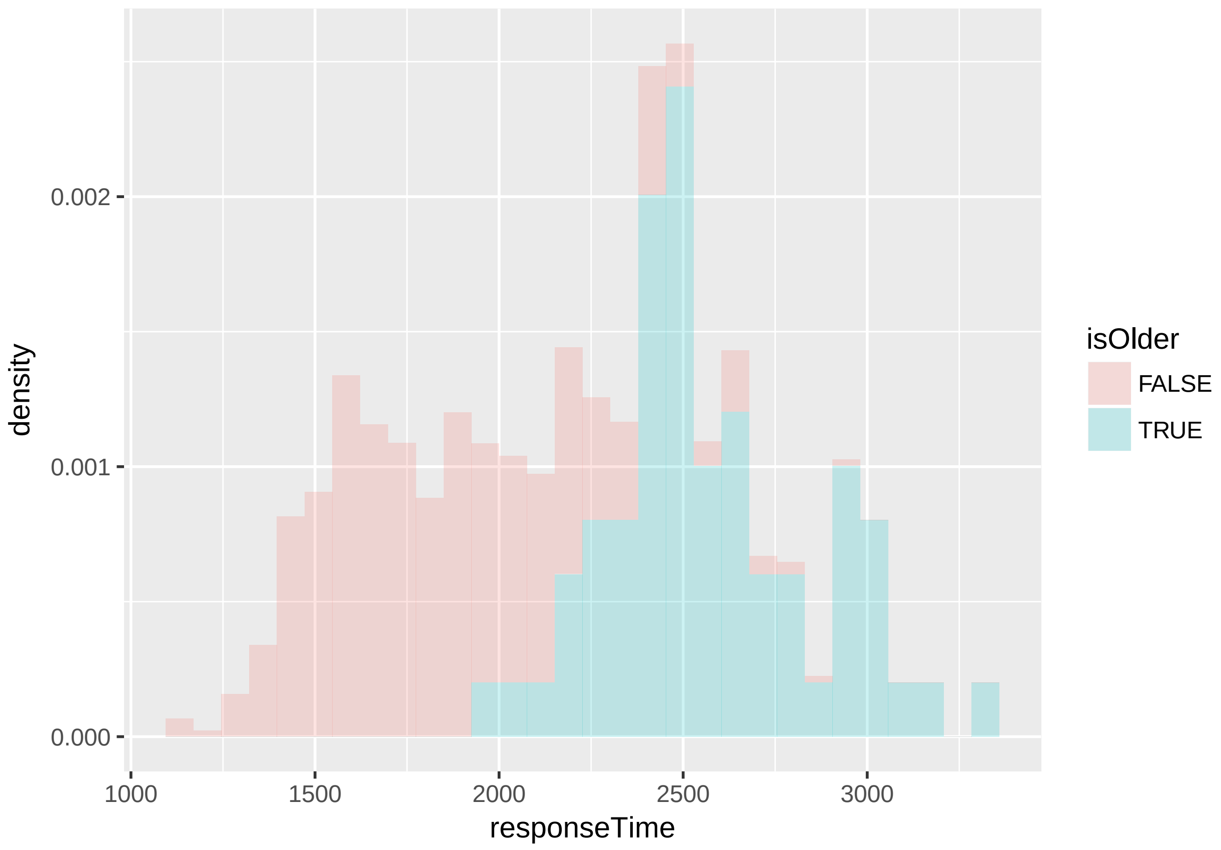
```r
data$responseTime = responseTime

ggplot(data, aes(age, responseTime)) + geom_smooth(alpha=0.2) + geom_point()
```

```
## `geom_smooth()` using method = 'loess'
```



```
ggplot(data, aes(responseTime, ..density.., fill=isOlder)) + geom_histogram(alpha=0.2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Create fake correlation in the data

For example imagine IQ is linked to performance and wether or not they went to university

```
# University
university = rep(0,N)
university[which(rnorm(N)>-0.25)] = 1

# Generate fake performance linked to IQ
performance = rnorm(N,60,15)*agedIq/100 + 30*university

# Scale between 0 and 1
performance = performance - min(performance)
performance = performance / max(performance)

data$performance = performance
data$id = getRandomId(N)
data$age = age
data$university =university
```

Save your data

```
# Set your working directory - your reference for the file system
setwd("~/Google Drive/Master Students/courses/introduction_a_r")
```

```r
# Save into your data/raw
save(data, file = "data/raw/data.RData")

# CSV format
write.csv(data, file = "data/raw/data.csv", row.names = FALSE)

# XLS - needs a library
library(WriteXLS)
WriteXLS(data, "data/raw/data.xls")
```

Example: Scaling function

```r
scale_by <- function (data, by = "minmax") {
        copied_data = data.frame(data)
        columns = names(data)

        switch(by,
               meanvar = {
                       center <- mean
                       spread <- sd
               },
               medianvar = {
                       center <- median
                       spread <- sd
               },
               minmax = {
                       center <- min
                       spread <- max
               })

        for (column in columns) {
                center_by = center(copied_data[[column]], na.rm = T)
                reduced_by = spread(copied_data[[column]], na.rm = T)
                copied_data[[column]] = (copied_data[[column]] - center_by) / reduced_by
        }
        return(copied_data)
}
```