

# 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 easily 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
```

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

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

## Matrix 2\*2 Needs to be of same type

```
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"
```

```
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

```
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

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

# Create a factor object.
```

```

factor_apple <- factor(apple_colors)

# Print the factor.
print(factor_apple)

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

print(nlevels(factor_apple))

## [1] 3

# 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 ++

```

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>

print(df$age)

## [1] 23 22 24 25

```

Missing values are “NOT ASSIGNED” or “NA”

```

print(df$subject)

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

print(is.na(df$subject))

## [1] FALSE FALSE FALSE TRUE

```

I dont want a factor, I want characters !

Sometimes you have to 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

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

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

```
a <- 12
```

```
print((a>20) || (a==12))
```

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

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

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

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

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

## Element wise logic

When a vector is tested against a vector of same length

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

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

```
print(a&b)
```

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

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

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

## Others

```
a <- 1:8
```

```
print(a)
```

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

```
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 the statement 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 = 1) {
  # 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(1,2,3,NA, na.rm = TRUE)
```

```
## [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
```



```

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 = getRandomId(650)

# letters and LETTERS are variables declared by default in R containing the minuscule and capital letters
print(c(0:9, letters, LETTERS))

## [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9" "a" "b" "c" "d" "e" "f" "g"
## [18] "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x"
## [35] "y" "z" "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
## [52] "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

# Add the ids to the data
newDataFrame = data.frame(id=ids, sertARien=1:650)

```

## 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  $a \times b = \text{product}(a, b)$
- \* A function that adds a prefix to a string, such that  $\text{prefixedString} = \text{prefix}(\text{prefixString}, \text{string})$
- \* 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

```

install.packages("ggplot2")
install.packages("psych")

```

### Load libraries

```

library(ggplot2)
library(psych)

```

```

##
## Attaching package: 'psych'

```

```
## The following objects are masked from 'package:ggplot2':
##
##    %+%, alpha
help(package= "ggplot2")
#vignette("ggplot2-specs", package = "ggplot2")
```

## Explore your data

### Load data

Usually you will load three type of data: \* Excel files: .xlsx \* 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")
pasdenom = read.csv("~/Google Drive/Master Students/courses/introduction_a_r/data/raw/data.csv")
class(pasdenom)

## [1] "data.frame"

#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
## age          1 650   41.06  14.42   41.00   41.01  19.27  18.00
## isOlder*      2 650    NaN    NA     NA     NaN    NA    Inf
## IQ            3 650   99.62  19.65   97.49   98.64  20.02  42.75
## responseTime  4 650 1951.79 411.37 1882.00 1918.58 412.16 1103.00
## performance   5 650    0.49   0.18    0.50    0.50   0.17   0.00
```

```
## id*          6 650      NaN      NA      NA      NaN      NA      Inf
## university   7 650      0.63     0.48     1.00     0.67     0.00     0.00
##              max      range skew kurtosis      se
## age          65.00    47.00  0.01     -1.30  0.57
## isOlder*     -Inf     -Inf      NA      NA      NA
## IQ           167.33   124.58  0.43      0.15  0.77
## responseTime 3376.00 2273.00 0.78      0.50 16.14
## performance   1.00     1.00 -0.01     -0.26  0.01
## id*          -Inf     -Inf      NA      NA      NA
## university    1.00     1.00 -0.55     -1.70  0.02
```

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

```
## $`FALSE`
##              vars   n    mean      sd median trimmed      mad      min
## age              1 576   38.24   12.84   39.00   38.18   17.79   18.00
## isOlder*         2 576    NaN      NA      NA      NaN      NA      Inf
## IQ               3 576  101.83   19.39  100.25  100.96   19.11   42.75
## responseTime     4 576 1867.61  334.68 1826.00 1848.75  345.45 1103.00
## performance      5 576    0.51    0.18    0.52    0.51    0.17    0.00
## id*              6 576    NaN      NA      NA      NaN      NA      Inf
## university       7 576    0.64    0.48    1.00    0.67    0.00    0.00
##              max      range skew kurtosis      se
## age              60.00    42.00  0.04     -1.31  0.54
## isOlder*         -Inf     -Inf      NA      NA      NA
## IQ              167.33   124.58  0.36      0.20  0.81
## responseTime    2927.00 1824.00 0.49     -0.23 13.94
## performance      1.00     1.00 -0.02     -0.28  0.01
## id*             -Inf     -Inf      NA      NA      NA
## university       1.00     1.00 -0.56     -1.69  0.02
##
## $`TRUE`
##              vars   n    mean      sd median trimmed      mad      min      max
## age              1  74   62.95    1.38   63.00   62.93    1.48   61.00   65.00
## isOlder*         2  74    NaN      NA      NA      NaN      NA      Inf     -Inf
## IQ               3  74   82.44   11.45   81.73   81.99    9.98   58.26  111.36
## responseTime     4  74 2607.04  361.98 2533.00 2583.37  394.37 1969.00 3376.00
## performance      5  74    0.40    0.15    0.42    0.41    0.15    0.06    0.72
## id*              6  74    NaN      NA      NA      NaN      NA      Inf     -Inf
## university       7  74    0.62    0.49    1.00    0.65    0.00    0.00    1.00
##              range skew kurtosis      se
## age              4.00  0.00     -1.39  0.16
## isOlder*         -Inf      NA      NA      NA
## IQ              53.10  0.33     -0.20  1.33
## responseTime    1407.00  0.52     -0.67 42.08
## performance      0.65 -0.25     -0.54  0.02
## id*             -Inf      NA      NA      NA
## university       1.00 -0.49     -1.78  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 numerical
```

```
##      variable # n.obs type      H1      H2      H3
## age          1   650   1        57        20        41
## isOlder*     2   650   2        FALSE      FALSE      FALSE
## IQ           3   650   1      82.47043    83.22762    80.90943
## responseTime 4   650   1        2176      1682      2157
## performance  5   650   1    0.5669042    0.5501323    0.4339533
## id*          6   650   3 846TVIi7vy6m 8TGgFovZhpwc M461gZCo2WCP
## university   7   650   1          1          1          1
##      H4      T1      T2      T3
## age      42      60      65      24
## isOlder* FALSE    FALSE    TRUE    FALSE
## IQ      104.19482  94.63783  71.00707  117.35119
## responseTime 2122    2574    2211    1430
## performance 0.5703210 0.2779218 0.3950018 0.4777132
## id*      DeWYFMnTHzLv TRCgXxyJdnB9 fHTuBCMEv6XG xLWjZTfxYEit
## university 1          0          1          1
##      T4
## age      28
## isOlder* FALSE
## IQ      110.41197
## responseTime 1567
## performance 0.6450491
## id*      VVQZl08mdk4k
## university 1
```

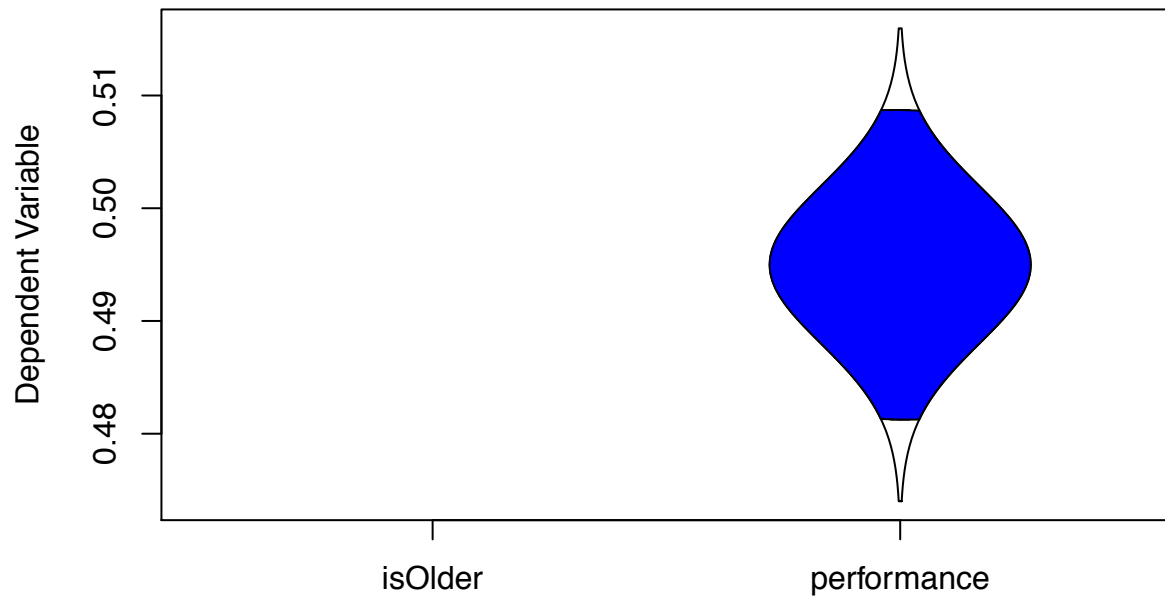
```
head(data)
```

```
##   age isOlder      IQ responseTime performance      id university
## 1  57   FALSE  82.47043        2176    0.5669042 846TVIi7vy6m      1
## 2  20   FALSE  83.22762        1682    0.5501323 8TGgFovZhpwc      1
## 3  41   FALSE  80.90943        2157    0.4339533 M461gZCo2WCP      1
## 4  42   FALSE 104.19482        2122    0.5703210 DeWYFMnTHzLv      1
## 5  44   FALSE  92.82097        2300    0.5969814 QZAzwKbsEKVl      1
## 6  21   FALSE 119.71190        1886    0.4573477 Wz61IDoR5Vud      0
```

```
# Some quick plots
```

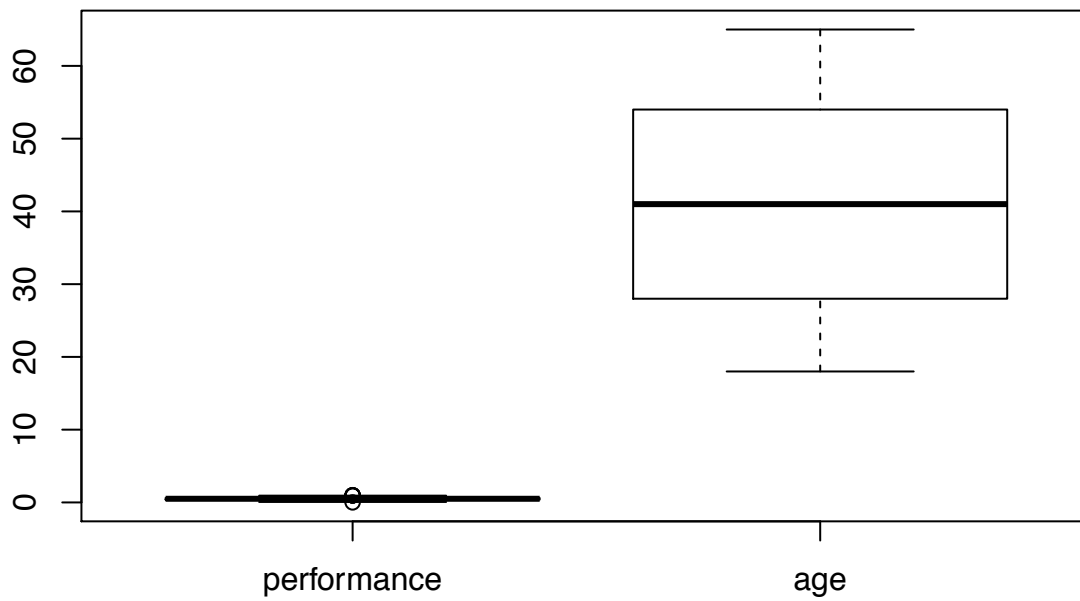
```
error.bars(data[, c("isOlder", "performance")])
```

### 95% confidence limits

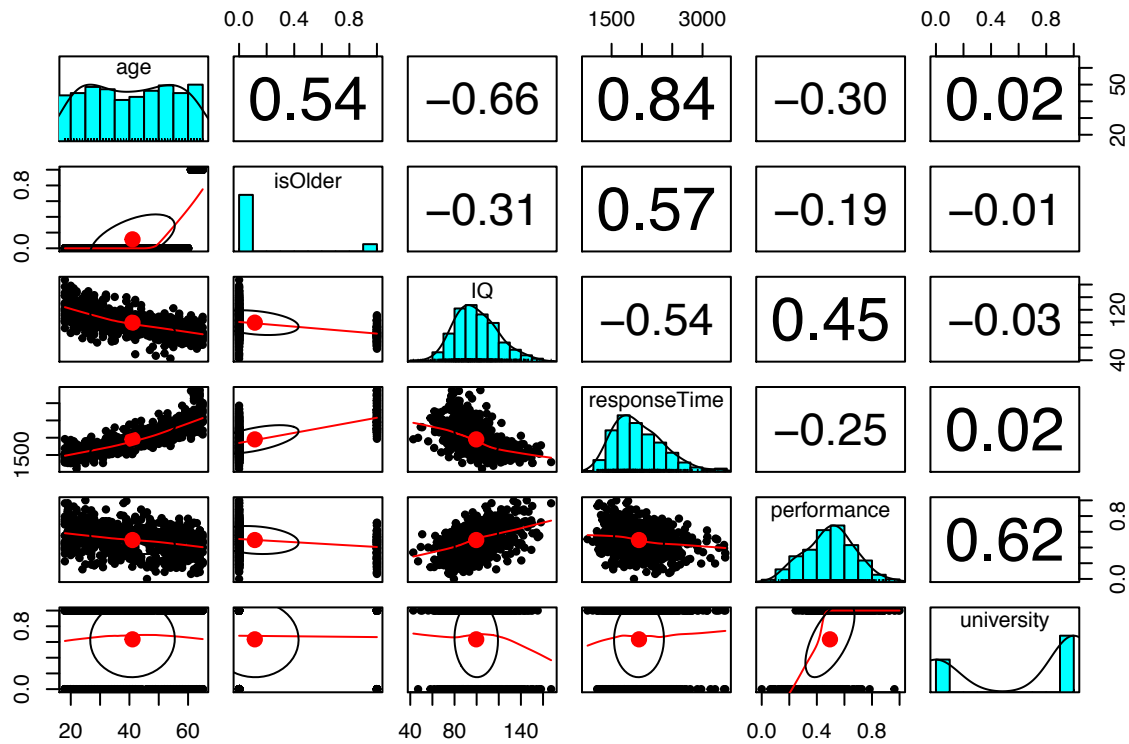


### Independent Variable

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

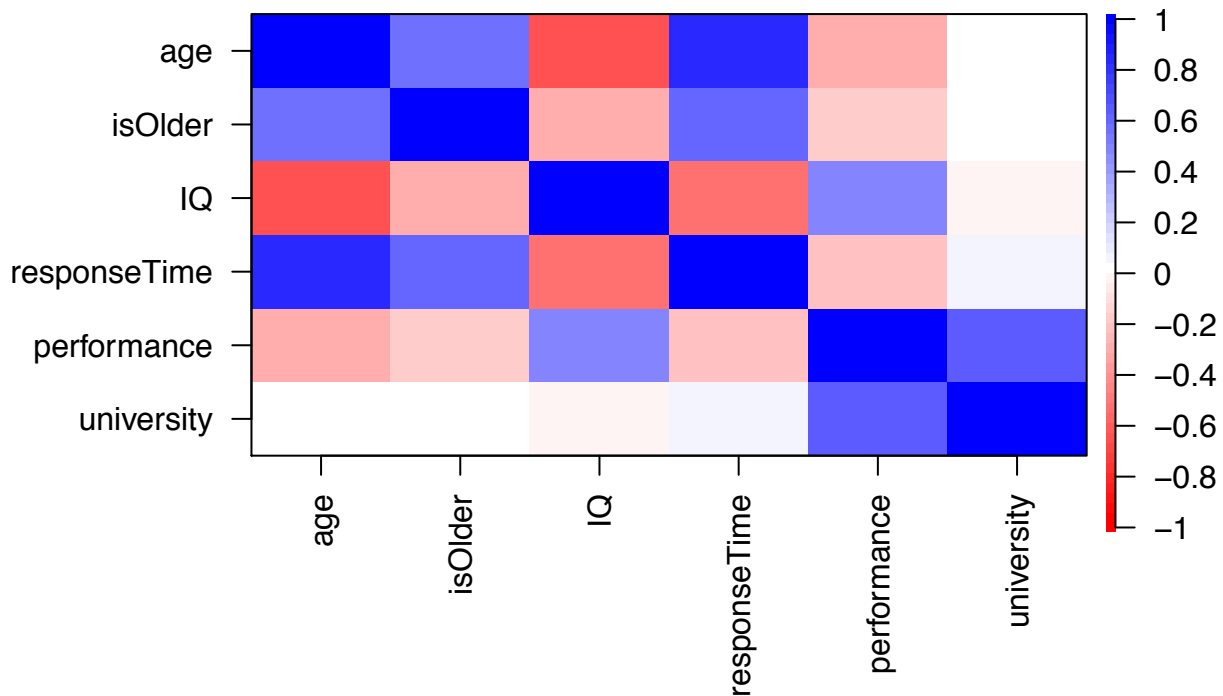


```
pairs.panels(data[-6])
```



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

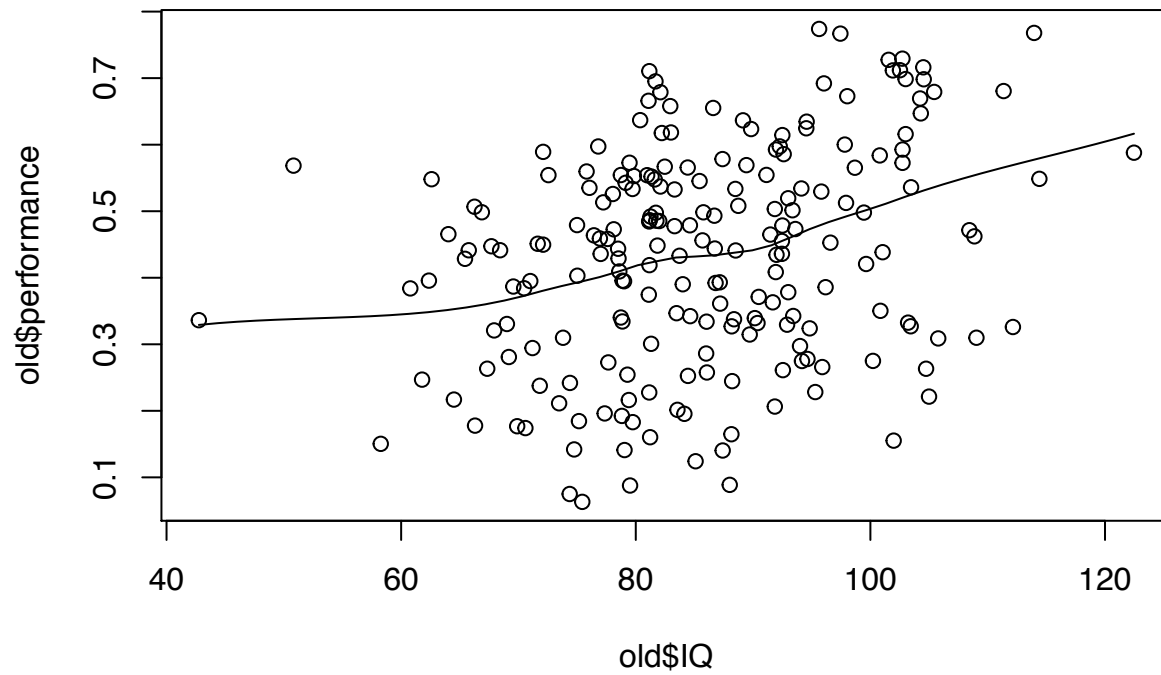
Correlation plot



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

# Filter your data
```

```
dataScrubbed = scrub(data,3:4,min=c(88,1300), max=c(115, 1600), newvalue=NA)
old = subset(data,data$age>50)
scatter.smooth(old$IQ, old$performance)
```

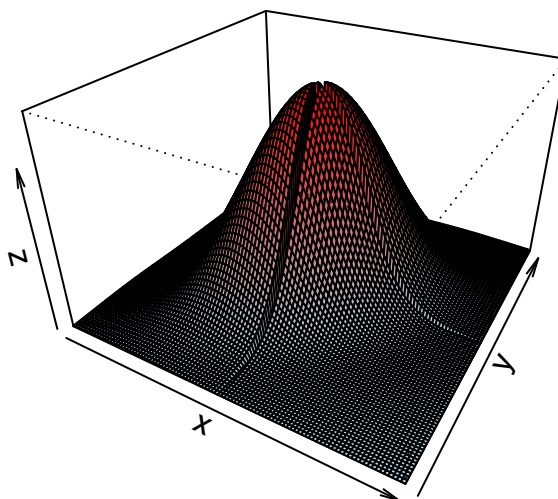
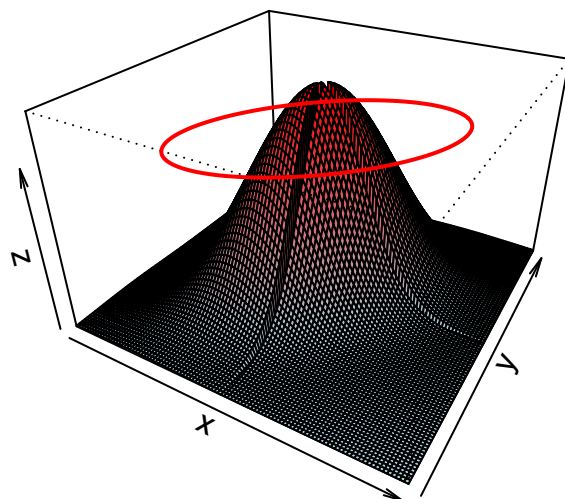


## Correlations

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

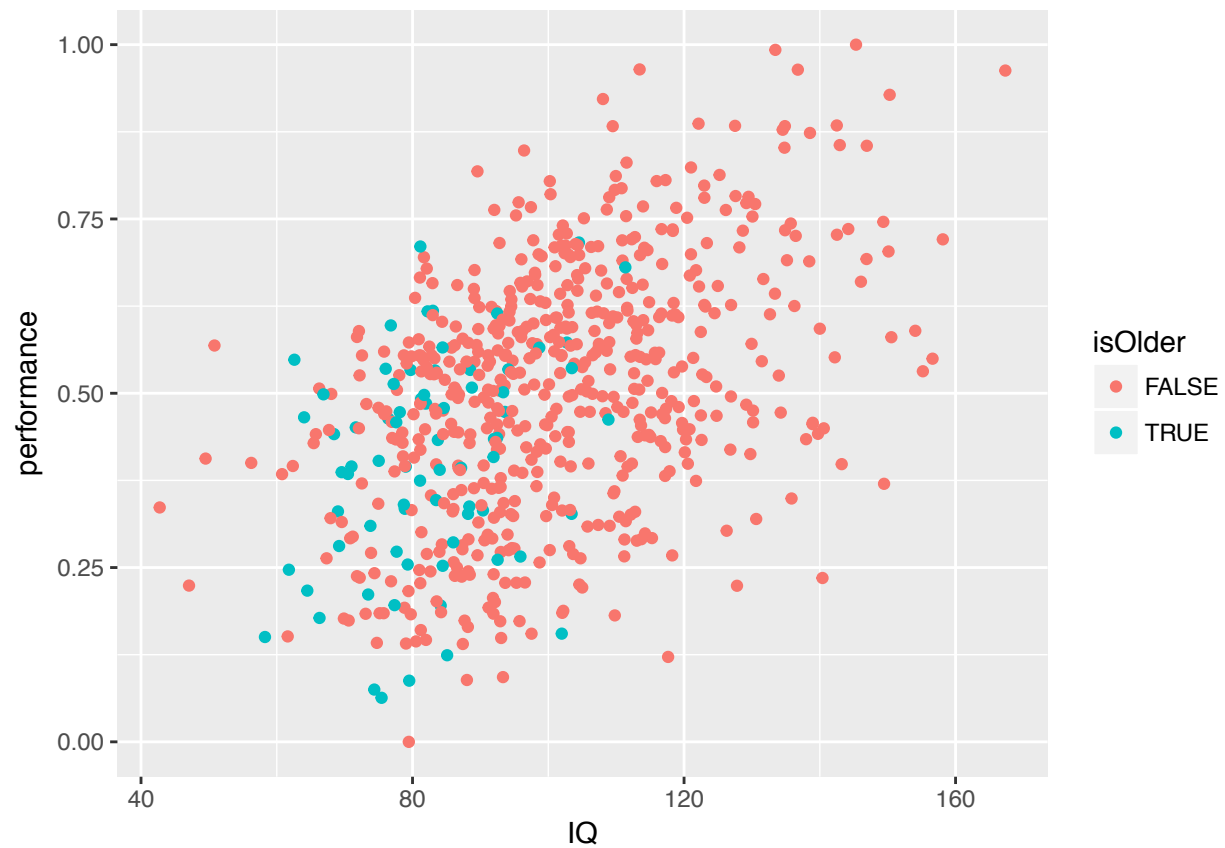
**Bivariate density rho = 0.57**

**Bivariate density rho = 0.57**



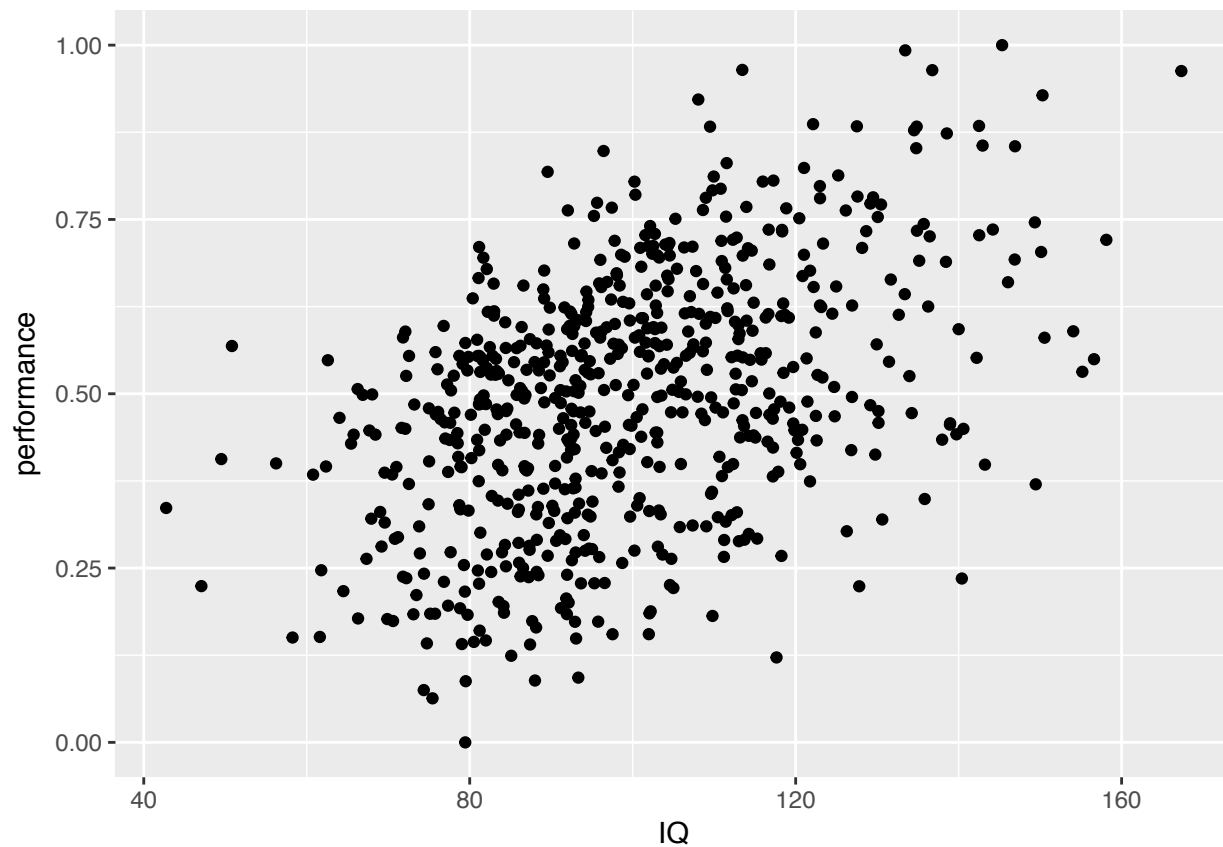
Plot Performance as a function of IQ. Performance =  $f(\text{IQ})$ .

```
ggplot(data, aes(x=IQ, y=performance, color=isOlder)) + geom_point()
```



```
ggplot(data, aes(x=IQ, y=performance)) + geom_point()
```

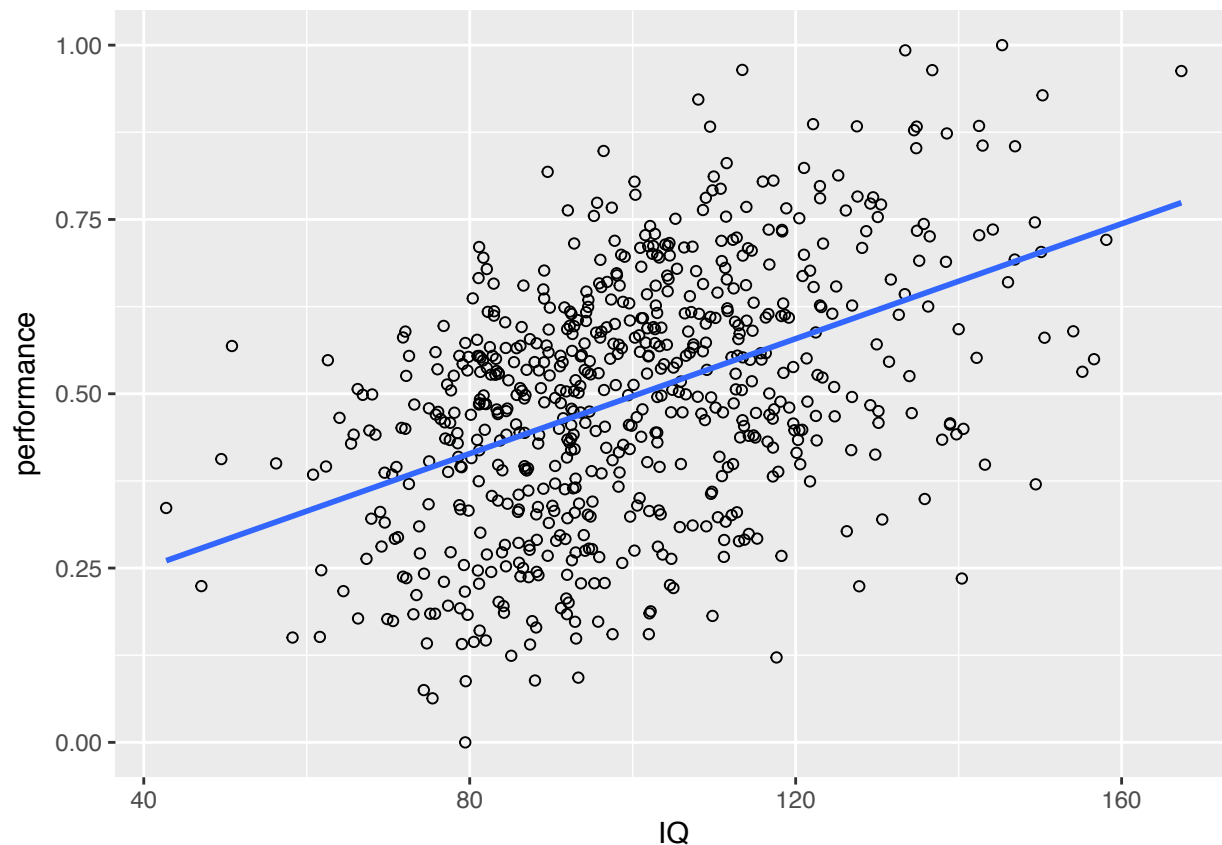




$\text{performance} = \text{beta} \cdot \text{IQ} + \text{intercept} + \text{bruit}$

Now, what kind of relation do you think exist between the two?

```
ggplot(data, aes(x=IQ, y=performance)) +  
  geom_point(shape=1) +      # Use hollow circles  
  geom_smooth(method=lm,    # Add linear regression line  
              se=FALSE)
```



Usually you want to know the interaction between multiple variables. For example the performance might be linked to the IQ, the Age, and/or the university !

performance =  $\beta IQ + age + university + IQage + \text{intercept} + \text{bruit}$

```
# regression
```

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

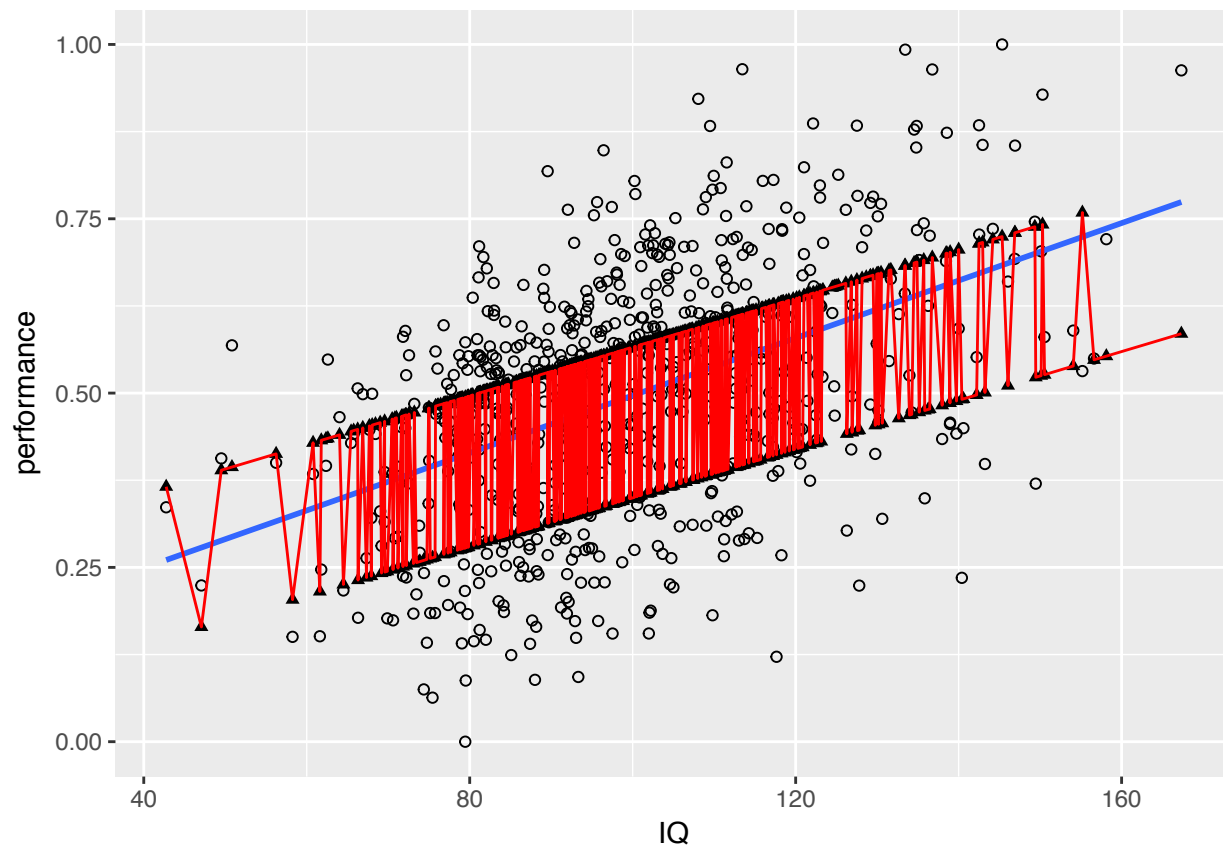
```
summary(regression)
```

```
##
## Call:
## lm(formula = performance ~ IQ + age + university + IQ:age, data = data,
##     na.action = na.omit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.30911 -0.06891 -0.00063  0.07283  0.32474
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.582e-02  7.511e-02  -1.143   0.254
## IQ           4.330e-03  6.731e-04   6.433 2.43e-10 ***
## age          5.046e-05  1.695e-03   0.030   0.976
## university   2.357e-01  9.092e-03  25.920 < 2e-16 ***
## IQ:age       -5.195e-07  1.705e-05  -0.030   0.976
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

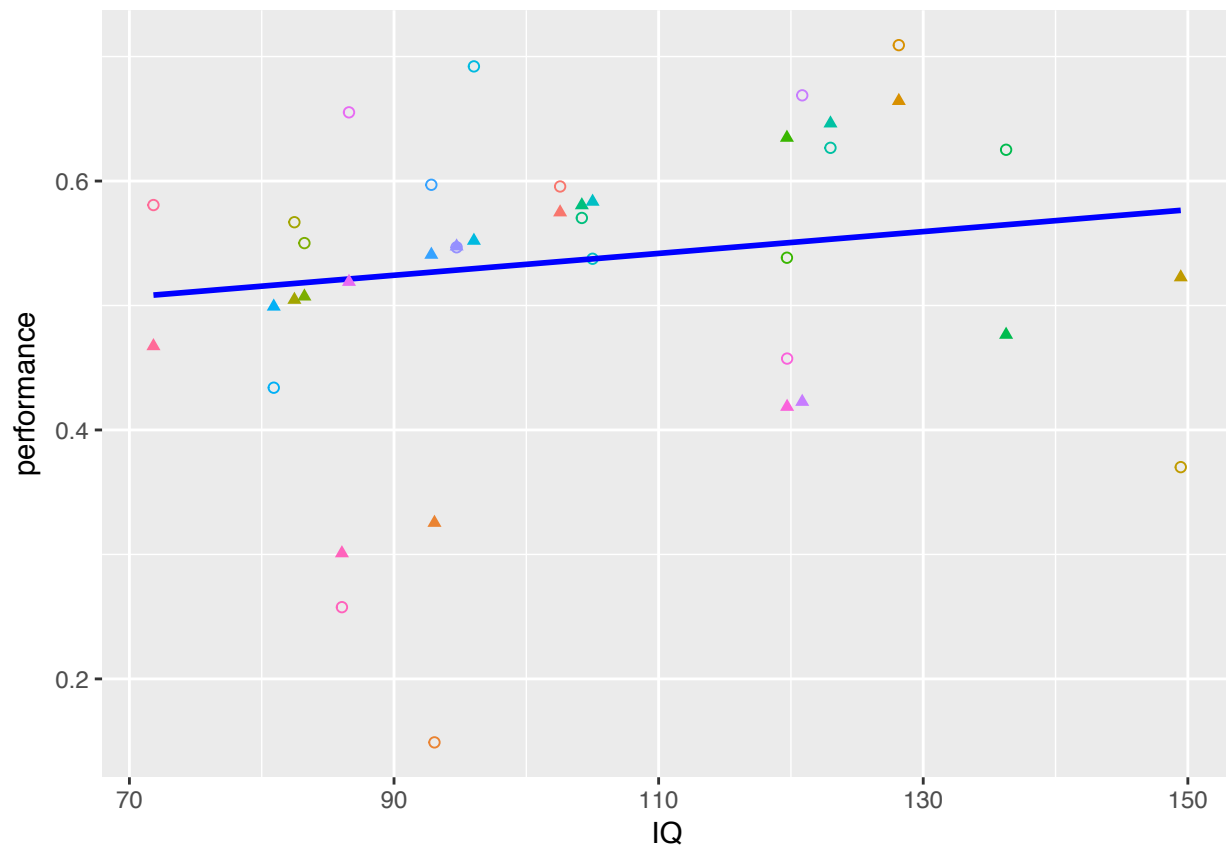
```
## Residual standard error: 0.1113 on 645 degrees of freedom
## Multiple R-squared:  0.6122, Adjusted R-squared:  0.6098
## F-statistic: 254.5 on 4 and 645 DF,  p-value: < 2.2e-16

data$prediction = 3.497e-03 * data$IQ + 2.162e-01 * data$university
```

```
ggplot(data, aes(x=IQ, y=performance)) +
  geom_point(shape=1) +      # Use hollow circles
  geom_smooth(method=lm,     # Add linear regression line
              se=FALSE)+
  geom_point(aes(x=IQ, y=prediction), shape=17)+
  geom_line(aes(x=IQ, y=prediction), color="red")
```



```
ggplot(data[1:20,], aes(x=IQ, y=performance, color=id)) +
  geom_point(shape=1)+      # Use hollow circles
  geom_smooth(method=lm,     # Add linear regression line
              se=FALSE, color="blue")+
  geom_point(aes(x=IQ, y=prediction, color=id), shape=17)+ theme(legend.position="none")
```



*# But is this model the good one ?*

```
regression2 = lm(formula = performance ~ IQ , data = data, na.action = na.omit)
summary(regression2)
```

```
##
## Call:
## lm(formula = performance ~ IQ, data = data, na.action = na.omit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44747 -0.12092  0.02391  0.11823  0.41245
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.0844036  0.0322395   2.618  0.00905 **
## IQ          0.0041213  0.0003175  12.980 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1589 on 648 degrees of freedom
## Multiple R-squared:  0.2064, Adjusted R-squared:  0.2051
## F-statistic: 168.5 on 1 and 648 DF, p-value: < 2.2e-16
```

*# Anova(model1,model2,test="Chisq") allows to compare models*

```
anova(regression,regression2,test="Chisq")
```

```
## Analysis of Variance Table
```

```
##
```

```

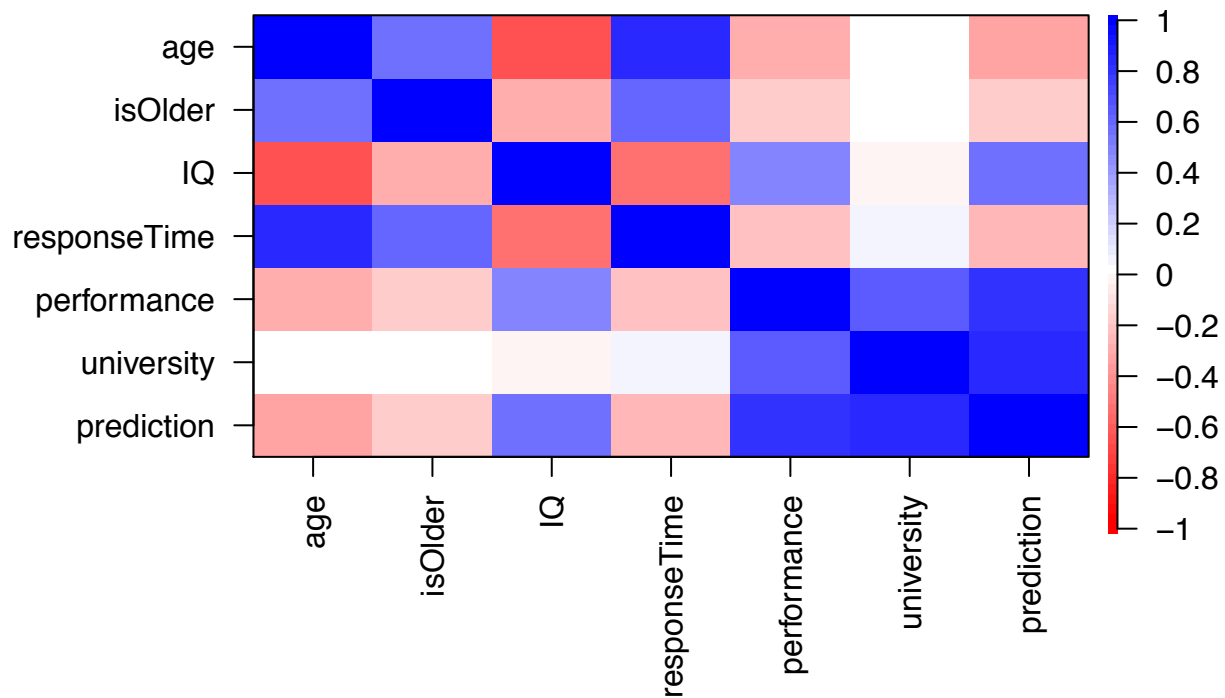
## Model 1: performance ~ IQ + age + university + IQ:age
## Model 2: performance ~ IQ
##   Res.Df    RSS Df Sum of Sq  Pr(>Chi)
## 1      645  7.9968
## 2      648 16.3644 -3    -8.3676 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# 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    0.54 -0.66          0.84         -0.30      0.02
## isOlder        0.54    1.00 -0.31          0.57         -0.19     -0.01
## IQ             -0.66   -0.31  1.00         -0.54          0.45     -0.03
## responseTime   0.84    0.57 -0.54          1.00         -0.25      0.02
## performance   -0.30   -0.19  0.45         -0.25          1.00      0.62
## university     0.02   -0.01 -0.03          0.02          0.62      1.00
## prediction    -0.35   -0.18  0.53         -0.28          0.78      0.83
##           prediction
## age           -0.35
## isOlder        -0.18
## IQ              0.53
## responseTime   -0.28
## performance     0.78
## university      0.83
## prediction      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     0.66    0.82 0.4          0.56          0          0
## prediction     0.00    0.00 0.0          0.00          0          0
##           prediction
## age           0
## isOlder        0
## IQ              0
## responseTime   0
## performance     0
## university      0
## prediction      0
##
## To see confidence intervals of the correlations, print with the short=FALSE option
corPlot(data.matrix(data[-6]))

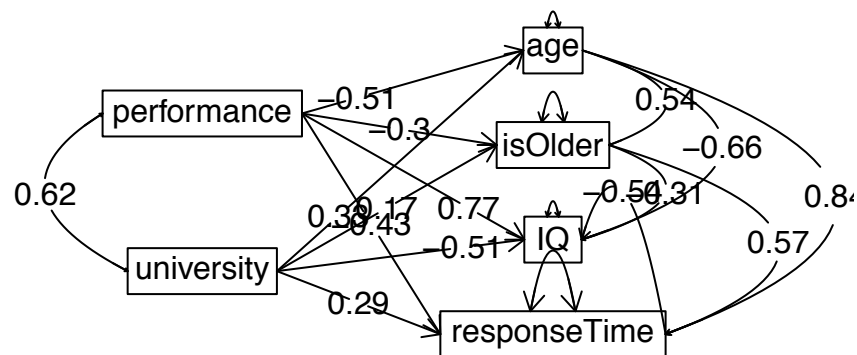
```

## Correlation plot



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

## Regression Models



unweighted matrix correlation = -0.19

```
## Call: setCor(y = 1:4, x = c(5, 7), data = data)
##
## Multiple Regression from raw data
##
## Beta weights
##           age isOlder   IQ responseTime
## performance -0.51   -0.30  0.77         -0.43
```

```

## university 0.33 0.17 -0.51 0.29
##
## Multiple R
## age isOlder IQ responseTime
## 0.40 0.23 0.61 0.34
## multiple R2
## age isOlder IQ responseTime
## 0.158 0.054 0.368 0.116
##
## Unweighted multiple R
## age isOlder IQ responseTime
## 0.18 0.11 0.27 0.15
## 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 -11.02 -6.05 19.40 -9.19
## university 7.23 3.58 -12.89 6.20
##
## Probability of t <
## age isOlder IQ responseTime
## performance 0.0e+00 2.4e-09 0 0e+00
## university 1.4e-12 3.7e-04 0 1e-09
##
## Shrunk R2
## age isOlder IQ responseTime
## 0.156 0.051 0.367 0.113
##
## Standard Error of R2
## age isOlder IQ responseTime
## 0.026 0.017 0.030 0.023
##
## F
## age isOlder IQ responseTime
## 60.85 18.35 188.74 42.44
##
## Probability of F <
## age isOlder IQ responseTime
## 0.00e+00 1.77e-08 0.00e+00 0.00e+00
##
## degrees of freedom of regression
## [1] 2 647
##
## Various estimates of between set correlations
## Squared Canonical Correlations
## [1] 0.371 0.001
## Chisq of canonical correlations

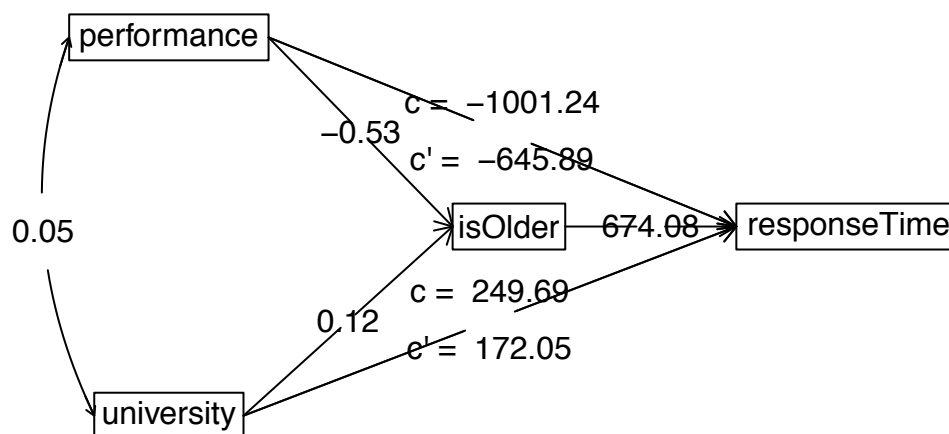
```

```
## [1] 299.24  0.66
##
## Average squared canonical correlation = 0.19
## Cohen's Set Correlation R2 = 0.37
## Shrunk Set Correlation R2 = 0.36
## F and df of Cohen's Set Correlation 41.84 8 1280
## Unweighted correlation between the two sets = -0.19
```

## Mediation analysis

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

### Mediation model



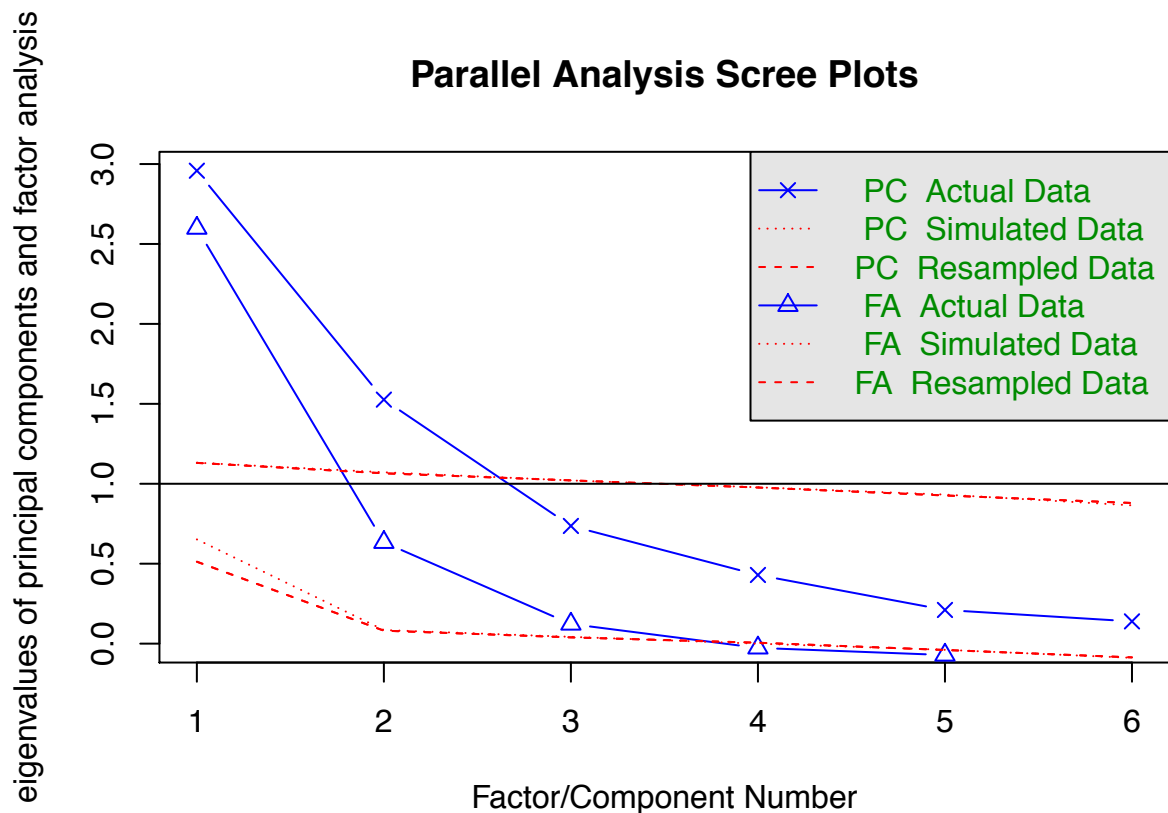
```
## Call: mediate(y = 4, x = c(5, 7), m = 2, data = data)
##
## The DV (Y) was responseTime . The IV (X) was performance university . The mediating variable(s) =
##
## Total Direct effect(c) of performance on responseTime = -1001.24 S.E. = 108.92 t direct =
## Direct effect (c') of performance on responseTime removing isOlder = -645.89 S.E. = 94.39
## Indirect effect (ab) of performance on responseTime through isOlder = -355.35
## Mean bootstrapped indirect effect = -354.54 with standard error = 58.72 Lower CI = -476.11 Upper CI =
##
## Total Direct effect(c) of university on responseTime = 249.69 S.E. = 40.27 t direct = 6.2
## Direct effect (c') of university on responseTime removing isOlder = 172.05 S.E. = 34.28 t direct =
## Indirect effect (ab) of university on responseTime through isOlder = 77.64
## Mean bootstrapped indirect effect = 77.64 with standard error = 36.8 Lower CI = 36.8 Upper CI =
## R2 of model = 0.37
## To see the longer output, specify short = FALSE in the print statement
##
## Full output
##
## Total effect estimates (c)
##      responseTime      se      t Prob
## performance    -1001.24 108.92 -9.19 0e+00
## university      249.69  40.27  6.20 1e-09
```



```
##
## Direct effect estimates      (c')
##           responseTime      se      t      Prob
## performance      -645.89  94.39  -6.84  1.80e-11
## university        172.05  34.28   5.02  6.73e-07
##
## 'a' effect estimates
##           isOlder      se      t      Prob
## performance      -0.53  0.09  -6.05  2.40e-09
## university        0.12  0.03   3.58  3.73e-04
##
## 'b' effect estimates
##           responseTime      se      t      Prob
## isOlder          674.08  41.45  16.26    0
##
## 'ab' effect estimates
##           responseTime      boot      sd      lower      upper
## performance      -355.35  -354.54  58.72  -476.11  -244.53
## university        77.64   77.81  21.54   36.80   122.48
```

## Dimensionality reduction

```
# How many components should you expect ?
dataWithoutId = data[-c(6,8)]
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       0.71 -0.01 0.50 0.50 1.0
## IQ           -0.75  0.20 0.60 0.40 1.1
## responseTime  0.90 -0.02 0.81 0.19 1.0
## performance -0.31  0.88 0.86 0.14 1.2
## university   0.12  0.91 0.84 0.16 1.0
```

```
##
##           RC1  RC2
## SS loadings      2.84 1.64
## Proportion Var    0.47 0.27
## Cumulative Var    0.47 0.75
## Proportion Explained 0.63 0.37
## Cumulative Proportion 0.63 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 0.1
## with the empirical chi square 186.72 with prob < 2.7e-39
##
## Fit based upon off diagonal values = 0.95
```

```
# Latent variable loads on data (age, IQ, university)
```

```
factanal(dataWithoutId, factors = 3)
```

```
##
```

```
## Call:
```

```
## factanal(x = dataWithoutId, factors = 3)
```

```
##
```

```
## Uniquenesses:
```

```
##           age          isOlder          IQ responseTime  performance
##           0.158          0.621          0.005          0.129          0.005
##  university
##           0.484
```

```
##
```

```
## Loadings:
```

```
##           Factor1 Factor2 Factor3
## age           0.842          -0.364
## isOlder        0.607
## IQ            -0.378          0.919
## responseTime  0.909          -0.210
## performance -0.171  0.924  0.335
## university    0.712
```

```
##
```

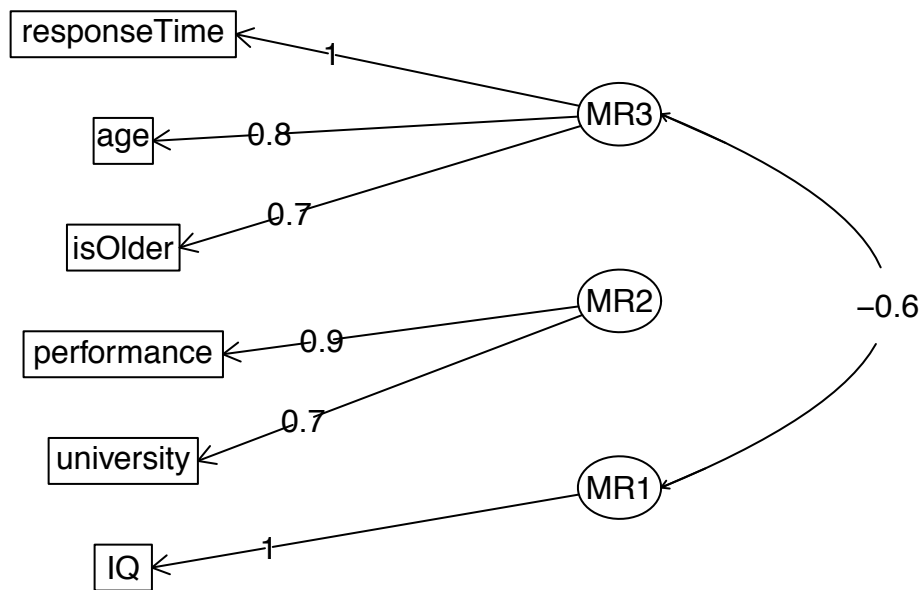
```
##           Factor1 Factor2 Factor3
## SS loadings    2.076  1.373  1.149
```

```
## Proportion Var    0.346    0.229    0.192
## Cumulative Var    0.346    0.575    0.766
##
## The degrees of freedom for the model is 0 and the fit was 0.0016
```

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

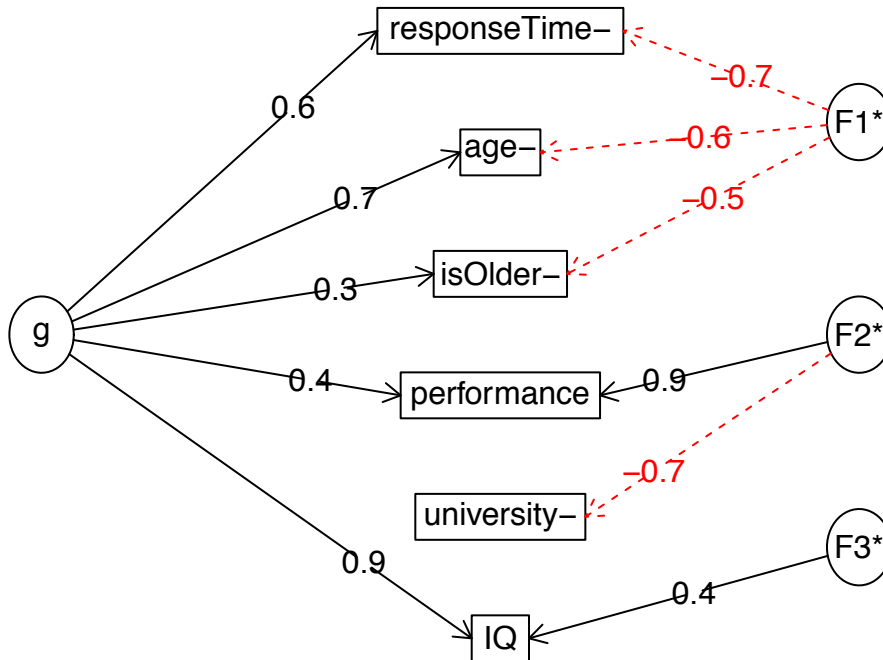
```
## Loading required namespace: GPArotation
```

## Factor Analysis



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

## Omega



```

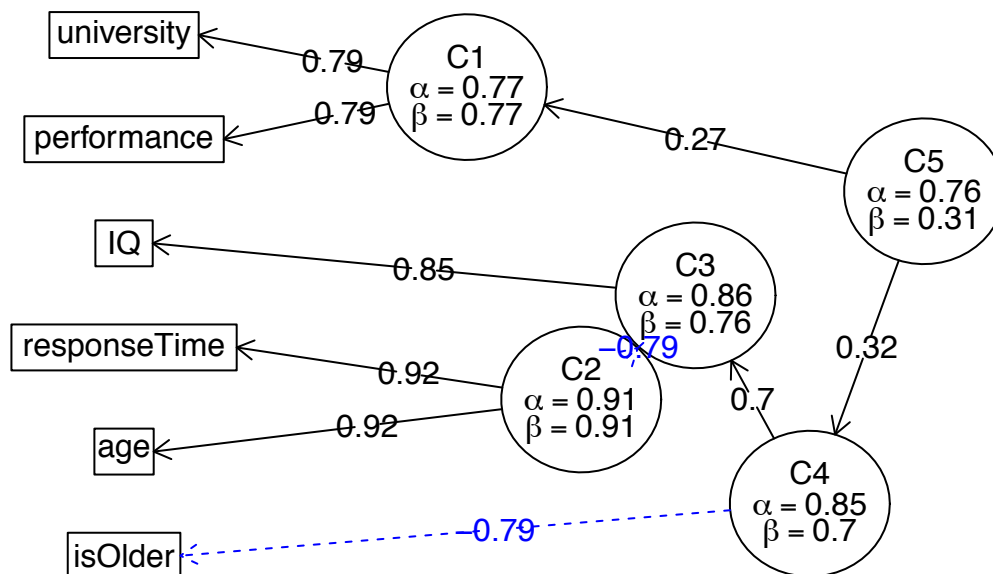
## Omega
## Call: omega(m = dataWithoutId)
## Alpha:          0.69
## G.6:            0.83
## Omega Hierarchical: 0.61
## Omega H asymptotic: 0.68
## Omega Total      0.9
##
## Schmid Leiman Factor loadings greater than 0.2
##      g    F1*   F2*   F3*   h2   u2   p2
## age-    0.67 -0.63                0.84 0.16 0.53
## isOlder- 0.34 -0.51                0.38 0.62 0.31
## IQ       0.91          0.41 1.00 0.00 0.83
## responseTime- 0.57 -0.74                0.87 0.13 0.38
## performance 0.43          0.90 1.00 0.00 0.19
## university-          -0.71 0.52 0.48 0.00
##
## With eigenvalues of:
##      g    F1*   F2*   F3*
## 1.91 1.19 1.31 0.19
##
## general/max 1.46   max/min = 6.86
## mean percent general = 0.37   with sd = 0.29 and cv of 0.77
## Explained Common Variance of the general factor = 0.41
##
## The degrees of freedom are 0 and the fit is 0
## The number of observations was 650 with Chi Square = 1.05 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 1.75
## The number of observations was 650 with Chi Square = 1128.6 with prob < 3.1e-237
## The root mean square of the residuals is 0.24
## The df corrected root mean square of the residuals is 0.31
##
## RMSEA index = 0.192 and the 90 % confidence intervals are 0.192 0.459
## BIC = 1070.31
##
## Measures of factor score adequacy
##
## Correlation of scores with factors      g  F1*  F2*  F3*
## Multiple R square of scores with factors      0.91 0.88 0.99 0.52
## Minimum correlation of factor score estimates 0.67 0.56 0.95 -0.45
##
## Total, General and Subset omega for each subset
##
## Omega total for total scores and subscales      g  F1*  F2*  F3*
## Omega general for total scores and subscales      0.90 0.87 0.31 0.99
## Omega group for total scores and subscales      0.61 0.36 0.27 0.82
## Omega group for total scores and subscales      0.26 0.51 0.05 0.17

#Clusters
iclust(dataWithoutId)
```

## ICLUST



```
## ICLUST (Item Cluster Analysis)
## Call: iclust(r.mat = dataWithoutId)
##
```

```

## Purified Alpha:
## [1] 0.76
##
## G6* reliability:
## [1] 0.63
##
## Original Beta:
## [1] 0.31
##
## Cluster size:
## [1] 6
##
## Item by Cluster Structure matrix:
##           [,1]
## age        -0.83
## isOlder    -0.53
## IQ          0.67
## responseTime -0.78
## performance  0.65
## university  0.29
##
## With eigenvalues of:
## [1] 2.5
##
## Purified scale intercorrelations
## reliabilities on diagonal
## correlations corrected for attenuation above diagonal:
##           [,1]
## [1,] 0.76
##
## Cluster fit = 0.71   Pattern fit = 0.9   RMSR = 0.2
# structural equation 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 extraction
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          -0.88 0.48
## responseTime  0.88 0.48
##
## For the 'Y' set:
##           MR1
## performance  1.00
## university   0.62
## age          -0.30
## isOlder      -0.19

```

```
##
## Correlations between the X and Y sets.
##      X1  X2  Y1
## X1  1.00 0.00 -0.49
## X2  0.00 1.00  0.07
## Y1 -0.49 0.07  1.00
##
## The degrees of freedom for the null model are 30 and the empirical chi square function was 3816.0
## The degrees of freedom for the model are 0 and the empirical chi square function was 42.29
## with prob < NA
##
## The root mean square of the residuals (RMSR) is 0.05
## The df corrected root mean square of the residuals is NA
## with the empirical chi square 42.29 with prob < NA
## The total number of observations was 650 with fitted Chi Square = 498.78 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)
```

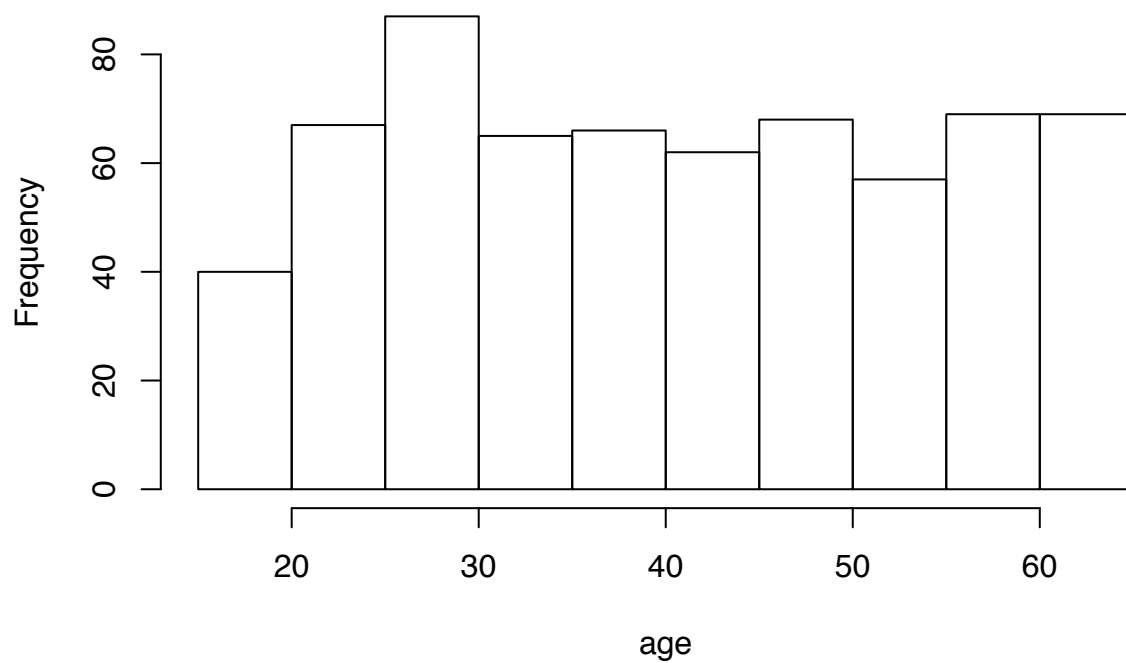
## How to generate fake data

```
# Number of subjects
N = 650

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

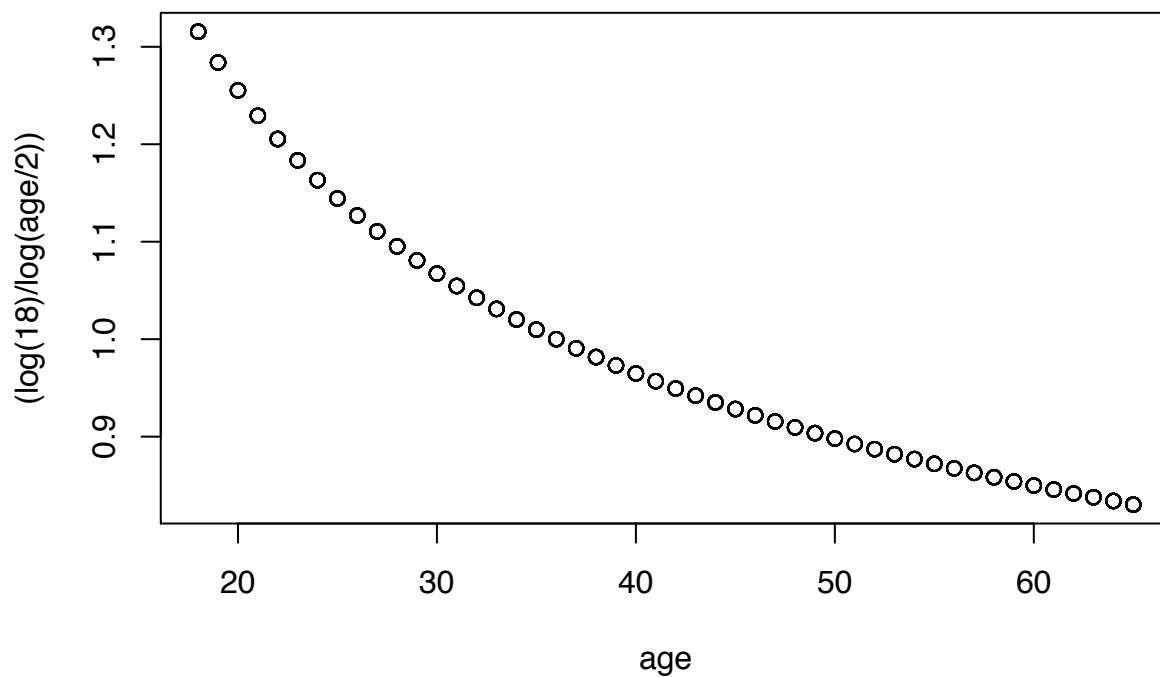
hist(age)
```

Histogram of age



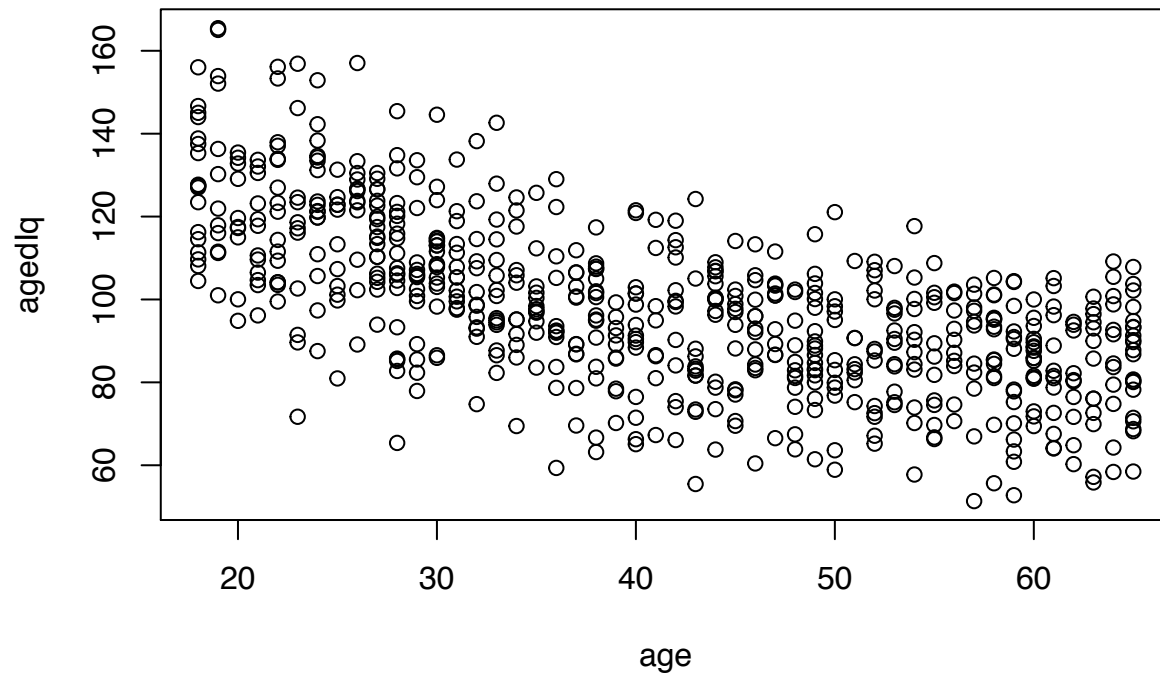
```
# Sample from gaussian distribution to generate IQ
baseIq = rnorm(N, 100, 15)
agedIq = baseIq * (log(18)/log(age/2))
data = data.frame(age = age, isOlder = age>60, IQ = agedIq)

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





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



```
# A slightly more beautiful way of plotting
```

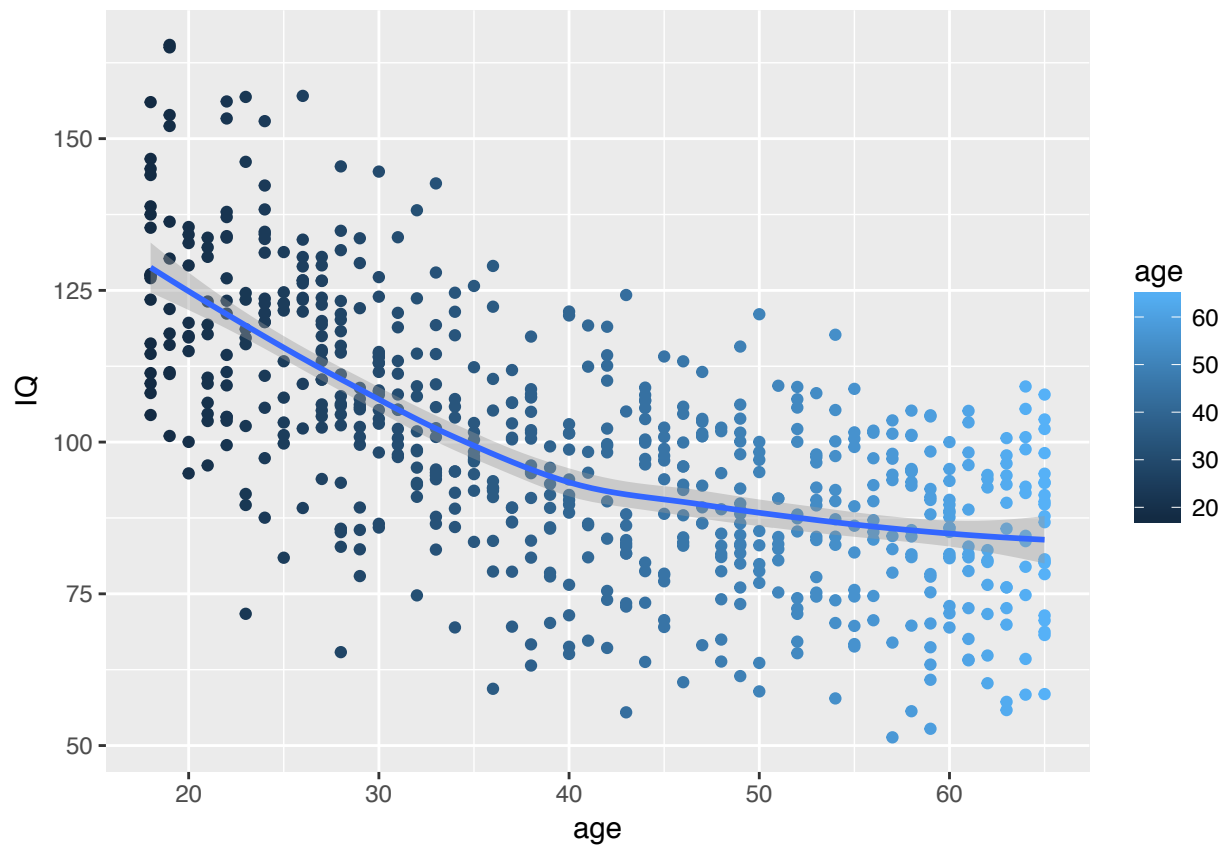
```
# install.packages("ggplot")
```

```
library(ggplot2)
```

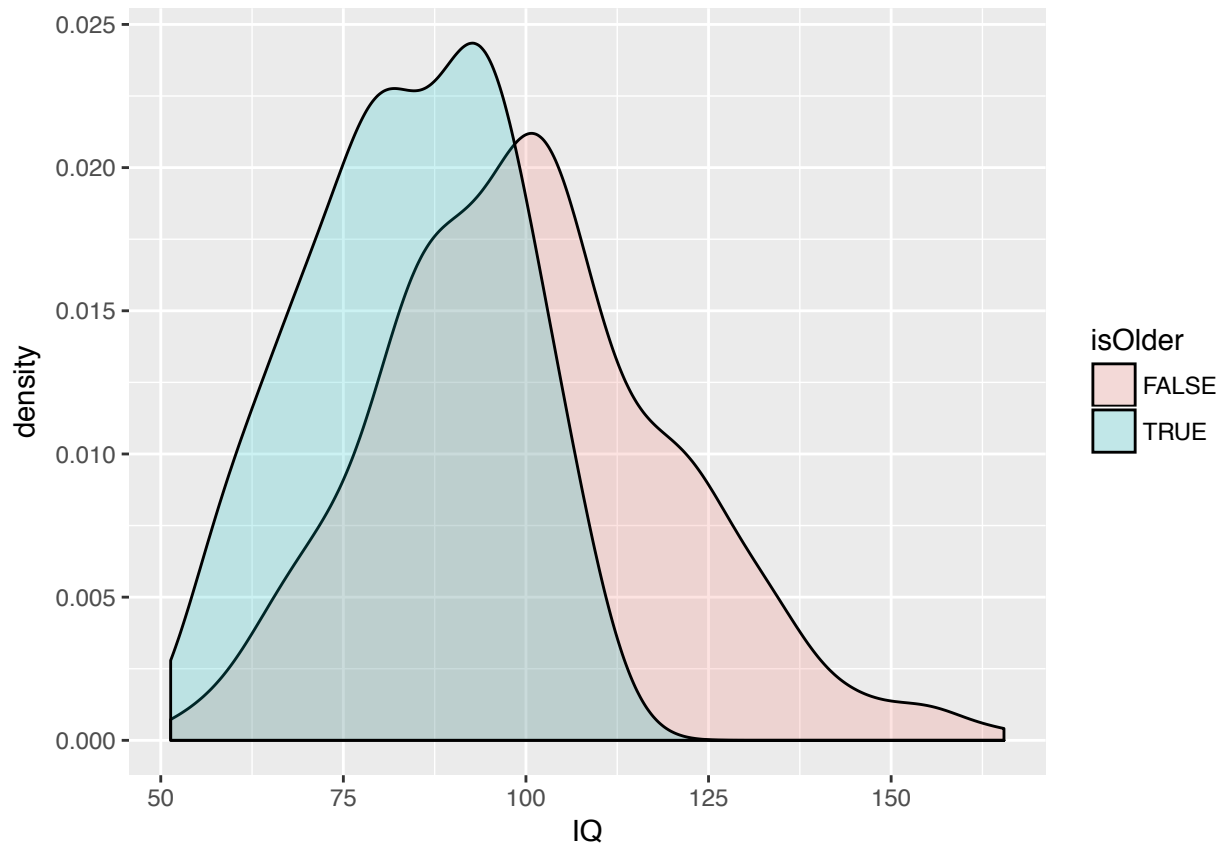
```
# Line plot + prediction error
```

```
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)
```



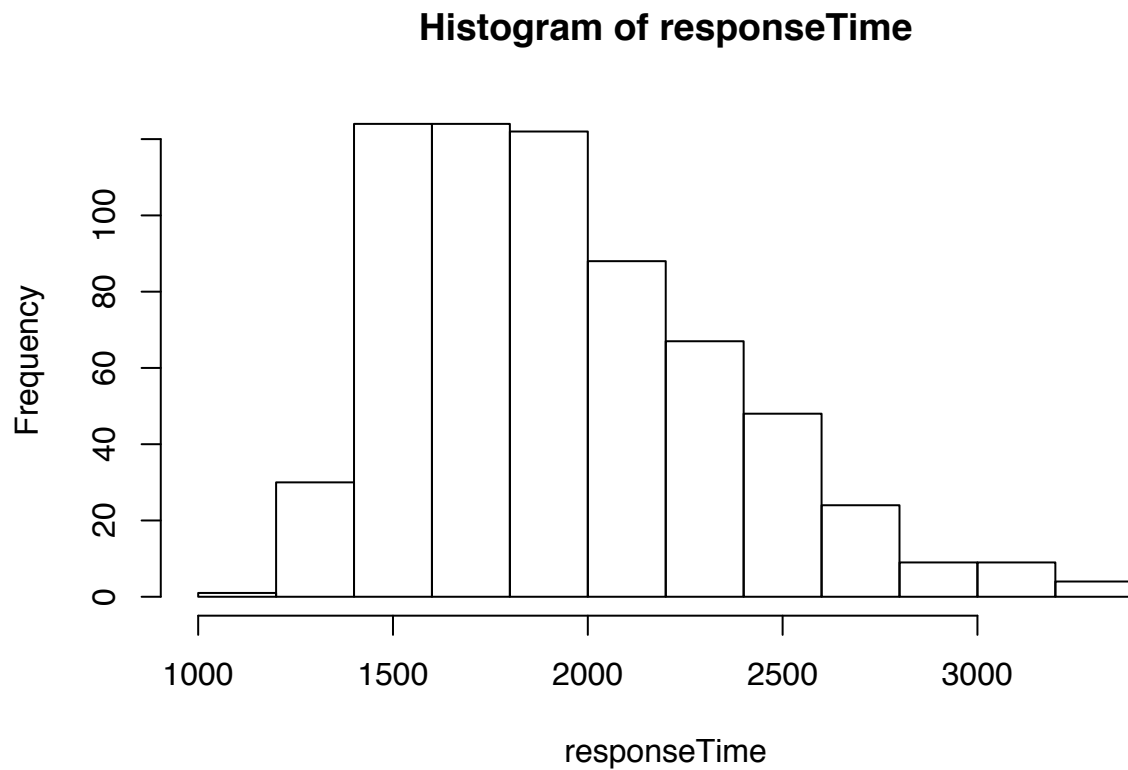
```
# Sample from a custom distribution
# For example the Exponential
#  $P(x) = \lambda * \exp(-\lambda * x)$ 
#  $\lambda = 1 / \text{mean}$ 
#  $\text{CDF} = 1 - \exp(-\lambda * x) \implies x = -\ln(1 - \text{CDF}) / \lambda$ 
mean = 1500
lambda = 1 / mean
CDF = sample(0:100, N, replace = T) / 100
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 !

```
# 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
}
```

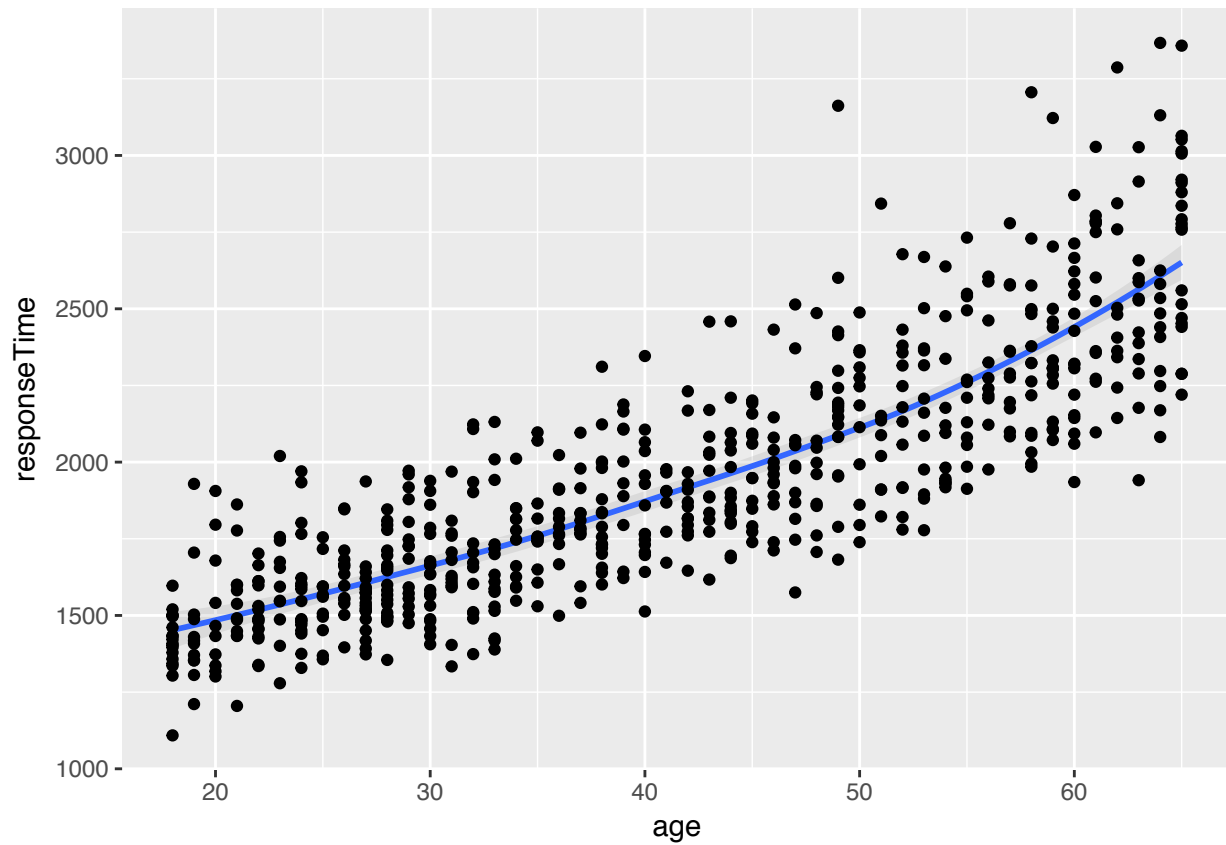
```
hist(responseTime)
```



```
data$responseTime = responseTime
```

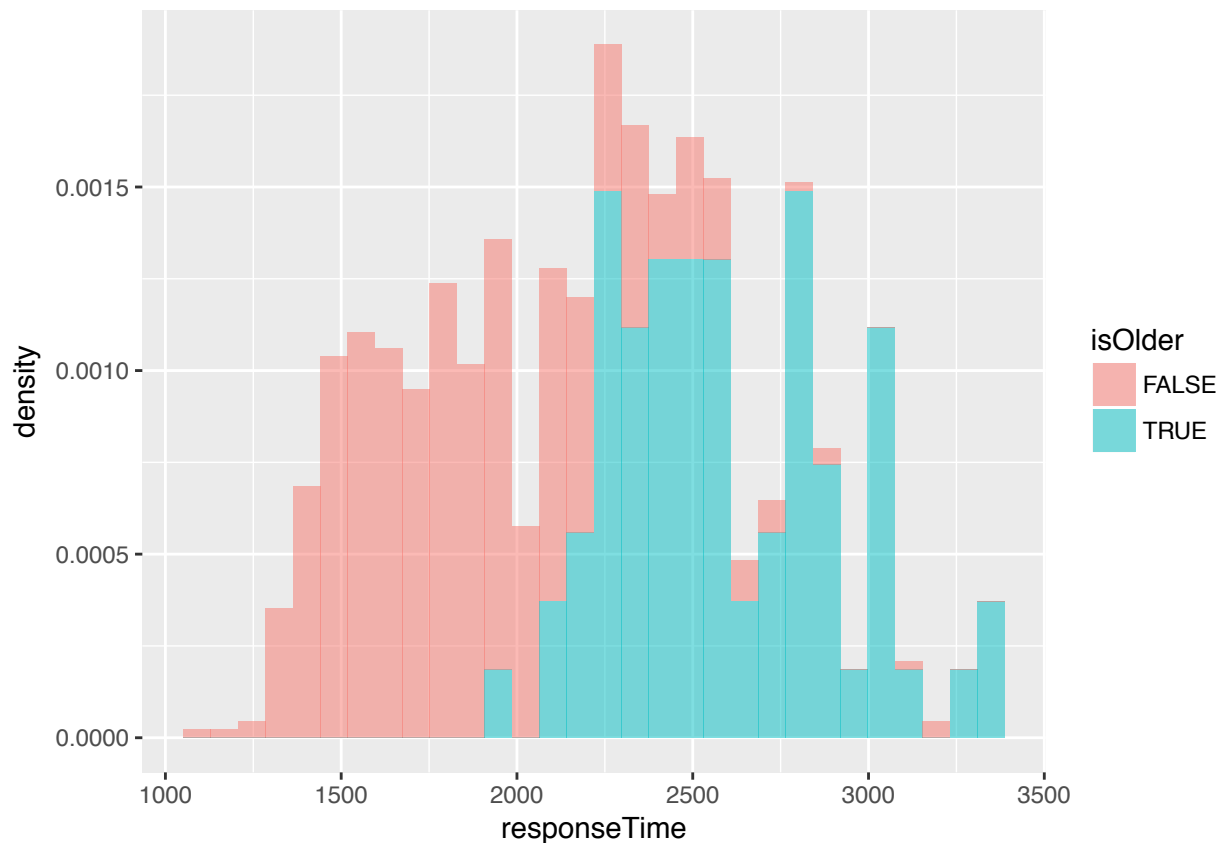
```
ggplot(data, aes(age, responseTime)) + geom_smooth(alpha=0.2) + geom_point()
```

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



```
ggplot(data, aes(responseTime, ..density.., fill=isOlder)) + geom_histogram(alpha=0.5)
```

```
## `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 whether 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")
```

```

# 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")

```

Generate fake questionnaire data

```

questionnaire_data = data.frame(like_sushi=sample(c(1:5),650, TRUE), like_risoto=sample(c(1:5), 650, TRUE),
WriteXLS(questionnaire_data, "data/raw/questionnaire_data.xls")

```

## The Perl script 'WriteXLS.pl' failed to run successfully.

## Example: Scaling function

```

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)
}

```

## Debugging

A good rule of thumb is that you will introduce a bug in your code every ten new lines. Besides real design issue in your code, bugs are usually due to overlooking certain extreme “use case” that you did not plan for, or because you did not pay attention to the values of your variables and content of your data.

Quick fix bugs: - Wrong variable name - Forgot a comma or parenthesis

Harder bugs: - Wrong format + Using a factor as a string or numeric + Some NA in the data + Some misformatted strings or outlier values - Wrong data size - Silent bugs - return incorrect values (+/- design issues)

Three steps to correct a bug: - Reproduce - Isolate : harder part - Correct

```
# Break on error
# browser()
# breakpoints
# print
```