Getting and Cleaning Data

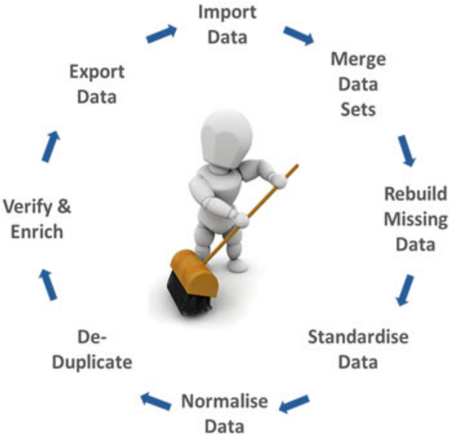
Ciprian Alexandru & Nicoleta Caragea

r-project.ro | @alexcipro

ICEA 2016 | Bucharest | ROMANIA | November 5th, 2016

R-omania Team

Cleaning Data - Why?



How to clean the data?

"This is not what I meant when I said 'we need better data cleansing!'"

Topics

- Intro
- About R
  - R environment
  - R installation
  - GUI (RStudio and R Commander)
  - Packages
- Import and Export Data
- Data Manipulation
- Validate package
- Simputation package

R-omania Team

Intro

Importing and cleaning data are the most important processes in data analysis. R is an efficient environment for detecting, diagnosing and finding data abnormalities. Along with the basic functions in R packages, there are packages dedicated to these processes. The `validate` package, a contribution of Mark van der Loo and Edwin de Jonge, help the data analysts to data validation process by checking data expectations about the data set. The `simputation` package aims to simplify missing value imputation using different methods like models and donor imputation. The models included in package are linear regression, robust linear regression, CART models and Random forest, respective the donor imputation methods k-nearest neigbour (based on gower's distance), sequential hotdeck (LOCF, NOCB), random hotdeck, predictive mean matching.

About R

What is R?

- R is a programming language and software environment for statistical computing and graphics
- The key point is the environment

From where R comes?

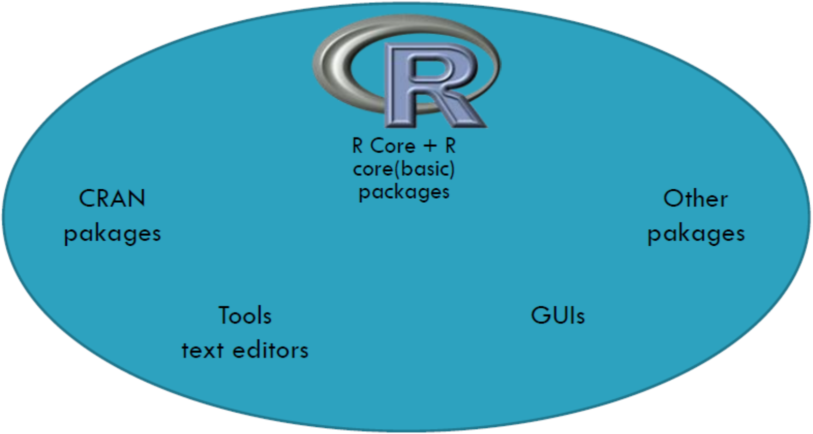
- 1997 - Ross Ihaka and Robert Gentleman, professors of statistical at the Auckland University from New Zeeland, starts to build a new software for statistical analysis and data graphical visualizations
- R is a dialect of S language (S was built by AT&T Bell Laboratories as a software for data analysis, statistical modeling, simulation and graphics)

Why R?

- R is supported by *academia*
- R is an *open source* initiative, similar with the Linux operating system or LaTeX markup language
- R is not just a statistics package, it's a *statistical programming language*
- R is designed to *overcome* the data scientist *problems*
- R is both *flexible*, powerful and endless

R-omania Team

R Environment



R Core + R core(basic) packages

CRAN pakages

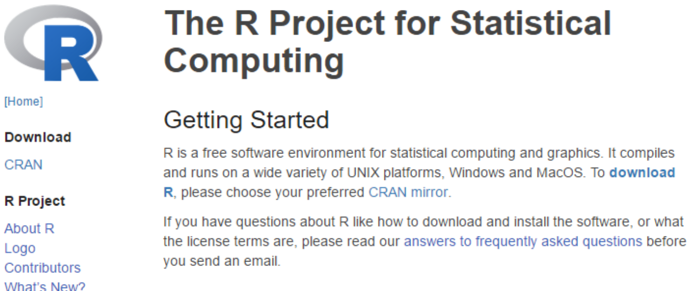
Other pakages

Tools text editors

GUIs

R Quick installation

- Install R for UNIX platforms, Windows and MacOS from `https://www.r-project.org/`
- The Windows users just clicks, other users know better than others
- R version 3.3.2 (Sincere Pumpkin Patch) has been released on Monday 2016-10-31



The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

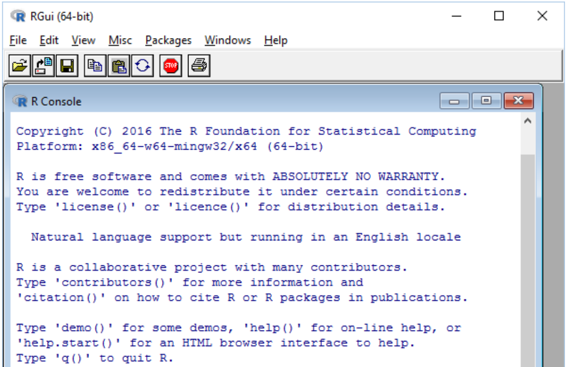
If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

GUI: R Console, R Studio, R Commander

- R Console: default GUI, include: the default multipledocumentinterface (MDI) and the single-document interface (SDI)
- R Studio: probably most complex GUI or integrated development environment (IDE) for R [https://www.rstudio.com/products/rstudio/features/, 2015-09-20]
- R Commander: contributed package Rcmdr: basic statistics GUI

R Console

- default GUI of R environment, included in R core
- > is the command prompt followed by a flashing cursor, meaning that R is waiting your reaction
- instructions/commands interpreted as functions



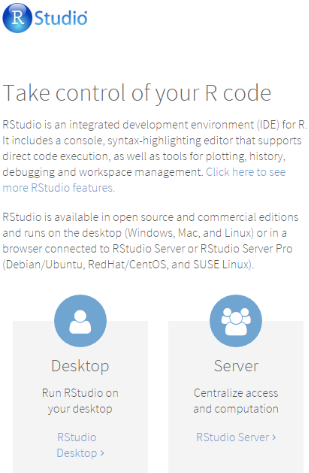
```
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

Always useful. . .

- R is case sensitive: variable ais different from A
- Keywords: if, else, repeat, while, function, for, in, next, break, TRUE, FALSE, NULL, Inf, NaN, NA, NA_integer_, NA_real_, NA_complex_, and finally, NA_character_
- navigation commands executed: arrow Up and Down
- Ctrl+L clear the console content

R Studio

https://www.rstudio.com/



RStudio

Take control of your R code

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. Click here to see more RStudio features.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux).

Desktop

Run RStudio on your desktop

RStudio Desktop ›

Server

Centralize access and computation

RStudio Server ›

Announcing RStudio v1.0!

Today we're very pleased to announce the availability of RStudio Version 1.0! Version 1.0 is our 10th major release since the initial launch in February 2011 (see the full release history below), and our biggest ever! Highlights include:

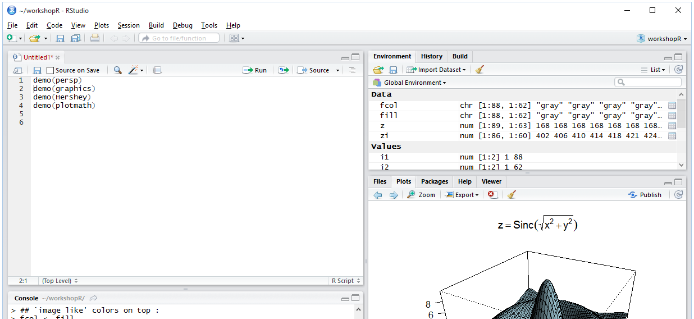
- Authoring tools for R Notebooks.
- Integrated support for the sparklyr package (R interface to Spark).
- Performance profiling via integration with the profvis package.
- Enhanced data import tools based on the readr, readxl and haven packages.
- Authoring tools for R Markdown websites and the bookdown package.
- Many other miscellaneous enhancements and bug fixes.

Source:

R Studio IDE

4 working area:

- Text/Commands/Script editor
- Console
- Environment, History, Build
- Files, Plots, Packages, Help, Viewer



```
demo(persp)
demo(graphics)
demo(Hershey)
demo(plotmath)
```

```
Data
  fcol    chr [1:88, 1:62] "gray" "gray" "gray" "gray"…
  fill    chr [1:88, 1:62] "gray" "gray" "gray" "gray"…
  z       num [1:89, 1:63] 168 168 168 168 168 168 168…
  zi      num [1:86, 1:60] 402 406 410 414 418 421 424…
Values
  i1      num [1:2] 1 88
  i2      num [1:2] 1 62
```

```
> ## `image like' colors on top :
> fcol <- fill
```

R Studio - features 1

- open source and commercial editions integrates the tools you use with R into a single environment
- available for Windows, Mac and Linux
- running on desktop, web browser and server
- efficient navigation to files and functions
- structure your work into projects
- integrated support for Gitand subversion
- authoring HTML, PDF, Word Documents, and slide shows
- supports interactive graphics with Shiny and ggvis

R Studio - features 2

Integrated Development Environment (IDE):

- syntax highlighting
- code completion
- smart indentation
- execute R code directly from the source editor
- quickly jump to function definitions

Bring your workflow together:

- Integrated R help and documentation
- Easily manage multiple working directories using projects
- Workspace browser and data viewer

Authoring & Debugging:

- Interactive debugger to diagnose and fix errors quickly

R Commander

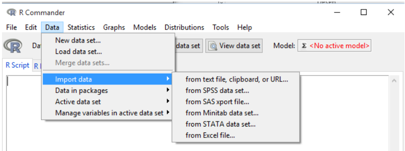
- `install.packages("Rcmdr")`
- `library(Rcmdr)`

Features:

- Data manipulation
- Statistics -basic statistical analyses
- Graphs - simple statistical graphs
- Models - numerical summaries, confidence intervals, hypothesis tests, diagnostics, and graphs for a statistical model, and for adding diagnostic quantities (eg, residuals) to the data set
- Distributions - probabilities, quantiles, and graphs of standard statistical distributions

R Commander (2)

- Script/editor window shows the command generates by user interactions with the menus.
- Edit/View data sets (reasonable small data sets)
- Data import from plain-text, Minitab, SPSS, or STATA
- Output window
- Messages window
- Graphics Device windows (appear separately)
- Save Graphs to different file type: bitmap, PDF, Postscript, EPS.
- Submit or Ctrl+rwill run your commands



R - Packages

Installed and loaded initialy:

- stats; graphics; grDevices; utils; datasets; methods; base

Installed but not loaded:

- base; boot; class; cluster; codetools; compiler; datasets; foreign; graphics; grDevices; grid; KernSmooth; lattice; MASS; matrix; methods; mgcv; nlme; nnet; parallel; rpart; spatial; splines; stats; stats4; survival; tcltk; tools; utils

Contributed CRAN packages:

- ggplot2; zoo; ggmap; 7.168+

Other packages:

- not included in CRAN

Very used functions

- `ls()`
- `rm()`
- `install.packages("zoo")`
- `remove.packages("zoo")`
- `library(zoo)`

Operators & functions

- standard arithmetic operators: +, -, *, and /
- mathematical functions: sqrt, exp, and log
- relational operators <=, <, ==, >, >= and !=
- logical operators: | for OR and & for AND
- assignment operators: <- or = and ->

```
# Variable x gets value 2:
x <- 2
# Value 2 goes to variable x:
2 -> x
```

Operator syntax

- $component extraction
- [[[indexing
- :sequence operator

```
x <- c(1:10)
x[(x < 5) | (x > 8)]
```

```
## [1]  1  2  3  4  9 10
```

Operator syntax (2)

```
1:5
```

```
## [1] 1 2 3 4 5
```

```
(a <-data.frame(name = c("Ion", "Maria"), income = c(1800,
```

```
##    name income
## 1   Ion   1800
## 2 Maria   2500
```

```
a$name
```

```
## [1] Ion   Maria
## Levels: Ion Maria
```

Operator syntax (3)

```
a[1]
```

```
##    name
## 1   Ion
## 2 Maria
```

```
a[2]
```

```
##   income
## 1   1800
## 2   2500
```

```
a[[1]]
```

```
## [1] Ion   Maria
```

Special values +Inf, -Inf, NaN

- R is properly infinite numerical values
- NaN-Not a Number
- Complex number:

```r
sqrt(as.complex(-2))
```

```
## [1] 0+1.414214i
```

```r
sqrt(-2+0i)
```

```
## [1] 0+1.414214i
```

Special values (2)

```
(a <- 2/0)
```

```
## [1] Inf
```

```
class(a)
```

```
## [1] "numeric"
```

```
exp(a)
```

```
## [1] Inf
```

```
exp(-a)
```

```
## [1] 0
```

Special values (3)

```
a - a
```

```
## [1] NaN
```

```
sqrt(a)
```

```
## [1] Inf
```

R objects

Five “atomic” classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

R objects (2)

```
(x <- "a") # character
```

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

```
class(x)
```

```
## [1] "character"
```

```
(x <- 1) # numeric
```

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

```
class(x)
```

```
## [1] "numeric"
```

R objects (3)

```
(x <- 1:5) # integer
```

```
## [1] 1 2 3 4 5
```

```
class(x)
```

```
## [1] "integer"
```

```
(x <- 2+3i) # complex
```

```
## [1] 2+3i
```

```
class(x)
```

```
## [1] "complex"
```

R objects (4)

```
(x <-TRUE) # logical
```

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

```
class(x)
```

```
## [1] "logical"
```

R objects (5)

```r
a <- 1
b <- as.integer(1)
a == b
```

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

```r
identical(a, b)
```

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

R objects (6)

Near equality

```
(a <- 0.2 + 0.2 + 0.2)
```

```
## [1] 0.6
```

```
(b <- 0.6)
```

```
## [1] 0.6
```

```
a == b
```

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

R-omania Team

```
all.equal(a, b)
```

R objects (7)

```r
a <- 1
class(a)
```

```
## [1] "numeric"
```

```r
typeof(a)
```

```
## [1] "double"
```

```r
b <- 1:2
class(b)
```

```
## [1] "integer"
```

R objects (8)

```r
typeof(b)
```

```
## [1] "integer"
```

```r
is.numeric(a)
```

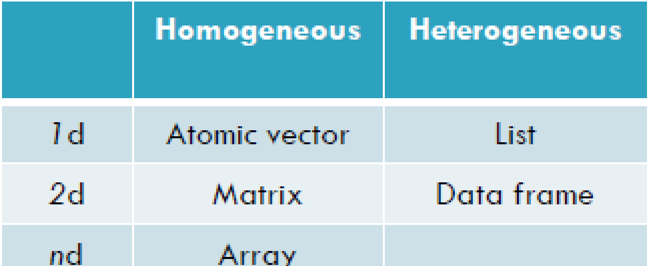
```
## [1] TRUE
```

```r
is.numeric(b)
```

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

R - data structures

- factors
- atomic vector
- matrix
- array
- data frame
- list
- table



	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

R - factor object

Factors - categorical data (unordered or ordered)

```
y <- c("yes", "no", "yes", "yes", "yes", "no")
x <- c("yes", "no", "yes", "yes", "yes", "no")
y <- as.factor(x)
x
```

```
## [1] "yes" "no"  "yes" "yes" "yes" "no"
```

```
y
```

```
## [1] yes no  yes yes yes no
## Levels: no yes
```

R - factor object (2)

```
str(x)
```

```
##  chr [1:6] "yes" "no" "yes" "yes" "yes" "no"
```

```
str(y)
```

```
##  Factor w/ 2 levels "no","yes": 2 1 2 2 2 1
```

R - factor object (3)

```
table(y)
```

```
## y
##  no yes
##   2   4
```

```
y
```

```
## [1] yes no  yes yes yes no
## Levels: no yes
```

```
levels(y)
```

```
## [1] "no"  "yes"
```

R - factor object (3)

```r
x <- factor(c("yes", "no", "yes", "yes", "no"), levels = c(
x
```

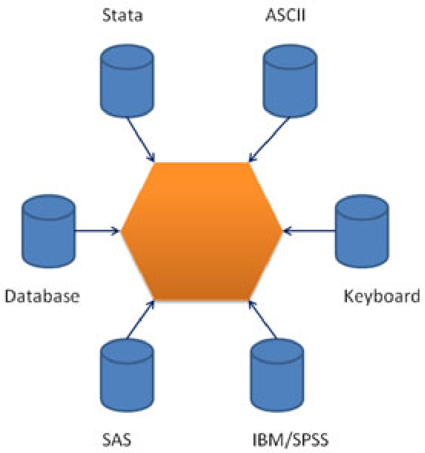
```
## [1] yes no  yes yes no
## Levels: yes no
```

R - data structures - data frame

- specific for data analysis/statisticians
- fundamental data structure by most of R's modeling software
- 2D vector (matrix), Heterogeneous data type
- list of vectors of equal length
- data.frame(., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringsAsFactors())
- is.data.frame(x)
- dim(x)
- ncol(x)
- nrow(x)
- x[row, col] or x[observation, variable]

R - Import and Export Data

The most flexible environment for data import



Data import (TXT)

read.table(file, header = FALSE, sep = “”, quote = “"”’, dec = “.”, nrows = -1, skip = 0, colClasses = NA, . . .)

```
date_txt <- read.table("pop_2015.txt", header=TRUE, sep=","
head(date_txt)
```

```
##   varsta persoane
## 1      0   191867
## 2      1   193175
## 3      2   180820
## 4      3   185018
## 5      4   206322
## 6      5   214428
```

Data import (TXT) - 1

```
#fisier0 <- "https://raw.githubusercontent.com/alexcipro/i
#date_txt_internet <- read.table(fisier0, header = TRUE, s
#head(date_txt_internet)
```

Data import (CSV)

read.csv(file, header = TRUE, sep = “,”, quote = “"”, dec = “.”, fill = TRUE, comment.char = “”, . . .)

```
# read.csv
mydata <- read.csv("pop_2015.csv")
# similar cu read.table
mydata <- read.table("pop_2015.csv", head = TRUE, sep = ","
```

Data import (Excel v1) - 1

- read first worksheet from mydata.xlsx
- first row contains variable names

```r
#install.packages("xlsx")
#install.packages("rJava")

# work only in R-32 bit version
library(rJava)
library(xlsx)
```

```
## Loading required package: xlsxjars
```

Data import (Excel v1) - 2

```
mydata1 <- read.xlsx("mydata.xlsx", 1)
head(mydata1)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
# read data from the worksheet: Sheet1
mydata2 <- read.xlsx("mydata.xlsx", sheetName = "Sheet1")
head(mydata2)
```

Data import (Excel v2)

```r
library(readxl)
mydata3 <- readxl::read_excel("mydata.xlsx")
head(mydata3)
```

```
## # A tibble: 6 × 6
##   Ozone Solar.R  Wind  Temp Month   Day
##   <chr>   <chr> <dbl> <dbl> <dbl> <dbl>
## 1    41     190   7.4    67     5     1
## 2    36     118   8.0    72     5     2
## 3    12     149  12.6    74     5     3
## 4    18     313  11.5    62     5     4
## 5    NA      NA  14.3    56     5     5
## 6    28      NA  14.9    66     5     6
```

Data import (DBF)

```
require(foreign)
```

```
## Loading required package: foreign
```

```
mydata <- read.dbf("mydata.dbf")
head(mydata)
```

```
##   OZONE SOLAR.R WIND TEMP MONTH DAY
## 1    41     190    7   67     5   1
## 2    36     118    8   72     5   2
## 3    12     149   13   74     5   3
## 4    18     313   12   62     5   4
## 5    NA      NA   14   56     5   5
## 6    28      NA   15   66     5   6
```

Data import (SPSS)

```
library(foreign) # using the arguments
use.value.labels don't convert value labels to factor
levels mydata <- read.spss(file.choose(),
use.value.labels = FALSE)
```

Data import (SAS)

```
# install.packages("Hmisc")
# library(Hmisc)
# mydata <- sasxport.get("d:/mydata.xpt")
```

Data import (stata)

```
# library(foreign)
# mydata <- read.dta("mydata.dta")
```

Data export (TXT)

```
write.table(mydata2, "mydataw.txt", sep="\t")
```

Data export (Excel)

```r
library(xlsx)
write.xlsx(mydata2, "mydataw.xlsx")
```

Data export (DBF)

```r
library(foreign)
write.dbf(mydata2, "mydataw.dbf")
```

SQL in R

```r
#install.packages("sqldf")
require(sqldf)
```

```
## Loading required package: sqldf

## Loading required package: gsubfn

## Loading required package: proto

## Loading required package: RSQLite

## Loading required package: DBI
```

```r
myCO2 <- CO2
head(CO2, 3)
```

SQL - select all variables

```
# SQL
s02 <- sqldf("select * from myCO2")
```

```
## Loading required package: tcltk
```

```
# R
r02 <- myCO2[ , ]
```

SQL - select only one variable

```
# SQL
s03 <- sqldf("select Type from myCO2")
# R
r03 <- myCO2[ , "Type"]
```

SQL - subset of variables

```r
# SQL
s01 <- sqldf("select Type, conc from myCO2")
# R
r01 <- myCO2[, c("Type", "conc")]
# testing s01 vs. r01
all.equal(s01, r01)
```

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

SQL - case sensitivity

```
# SQL is not case-sensitive
# s04 <- sqldf("select type, coNC from myCO2")
# R is case-sensitive
# r04 <- myCO2[, c("type", "coNC")]
```

SQL - variable selection through number

```
head(myCO2[, c(1, 3, 5)], 3)
```

```
##   Plant  Treatment uptake
## 1   Qn1 nonchilled   16.0
## 2   Qn1 nonchilled   30.4
## 3   Qn1 nonchilled   34.8
```

```
# the order of variables is important
head(myCO2[, c(5, 2)], 3)
```

```
##   uptake   Type
## 1   16.0 Quebec
## 2   30.4 Quebec
## 3   34.8 Quebec
```

SQL - variable selection through logic values

```
# selectionf or variables/columns/fields by logic variables
head(myCO2[, c(TRUE, FALSE, FALSE, TRUE, FALSE)], 3)
```

```
##   Plant conc
## 1   Qn1   95
## 2   Qn1  175
## 3   Qn1  250
```

```
# or
head(myCO2[, colnames(myCO2) > "d"], 3)
```

```
##   Plant   Type  Treatment uptake
## 1   Qn1 Quebec nonchilled   16.0
## 2   Qn1 Quebec nonchilled   30.4
## 3   Qn1 Quebec nonchilled   34.8
```

SQL - selections by criteria (1)

```
# SQL
s05 <- sqldf("select * from myCO2 where uptake < 20")
# R
r05 <- myCO2[ myCO2[, "uptake"] < 20, ]
# or using with function
r05w <- with(myCO2, myCO2[uptake < 20, ]) # identical with
```

SQL - selections by criteria (2)

```r
# SQL
s06 <- sqldf("select * from myCO2 where uptake < 20 and Typ
# R
r06 <- with(myCO2, myCO2[uptake < 20 & Type == 'Quebec', ])
```

SQL - first n observations

```
# SQL
s07 <- sqldf("select * from myCO2 limit 6")
# R
r07 <- head(myCO2, 6)
```

SQL - NULL

```
r08 <- r06
r08[2:4, 1] <- NA
r08[5, 4] <- NA
r08
```

```
##     Plant   Type  Treatment conc uptake
## 1     Qn1 Quebec nonchilled   95   16.0
## 8    <NA> Quebec nonchilled   95   13.6
## 15   <NA> Quebec nonchilled   95   16.2
## 22   <NA> Quebec    chilled   95   14.2
## 29    Qc2 Quebec    chilled   NA    9.3
## 36    Qc3 Quebec    chilled   95   15.1
```

SQL - Not NULL

```
# SQL
s09 <- sqldf("select * from r08 where plant is not null")
# R
r09 <- with(r08, r08[!is.na(Plant), ])
```

SQL - is NULL

```
# SQL
s10 <- sqldf("select * from r08 where plant is null")
# R
r10 <- with(r08, r08[is.na(Plant), ])
```

SQL - without missing values

```
# R
na.omit(r08)
```

```
##    Plant   Type  Treatment conc uptake
## 1    Qn1 Quebec nonchilled   95   16.0
## 36   Qc3 Quebec    chilled   95   15.1
```

R - Data manipulation

Data selection and manipulation (1)

- `which.max(x)`, `which.min(x)` - returns the index of the greatest/smallest element of x
- `rev(x)` - reverses the elements of x
- `sort(x)` - sorts the elements of x in increasing order; to sort in decreasing order: `rev(sort(x))`
- `cut(x,breaks)` - divides x into intervals (factors); breaks is the number of cut intervals or a vector of cut points
- `match(x,y)` returns a vector of the same length as x with the elements of x that are in y (NA otherwise)
- `which(x==a)` returns a vector of the indices of x if the comparison operation is true (TRUE), in this example the values of ifor which `x[i] == a` (the argument of this function must be a variable of mode logical)

Data selection and manipulation (2)

- `choose(n,k)` computes the combinations of k events among n repetitions = n!/[(n ???k)!k!]
- `na.omit(x)` suppresses the observations with missing data (NA)
- `na.fail(x)` returns an error message if x contains at least one NA `complete.cases(x)` returns only observations (rows) with no NA
- `unique(x)` if x is a vector or a data frame, returns a similar object but with the duplicates suppressed
- `table(x)` returns a table with the numbers of the different values of x (typically for integers or factors)
- `split(x,f)` divides vector x into the groups based on f

R-omania Team

Data selection and manipulation (3)

- `subset(x,...)` returns a selection of x with respect to criteria (..., typically comparisons: x$V1 $<$ 10); if x is a data frame, the option select gives variables to be kept (or dropped, using a minus)
- `na.fail(x)` returns an error message if x contains at least one NA
- `complete.cases(x)` returns only observations (rows) with no NA

Data reshaping (1)

- `merge(a,b)` merge two data frames by common col or row names
- `stack(x,...)` transform data available as separate cols in a data frame or list into a single col
- `unstack(x,...)` inverse of `stack()`
- `rbind(...)`, `cbind(...)` combines supplied matrices, data frames, etc. by rows or cols
- `melt(data,id.vars,measure.vars)` changes an object into a suitable form for easy casting, (`reshape2` package)

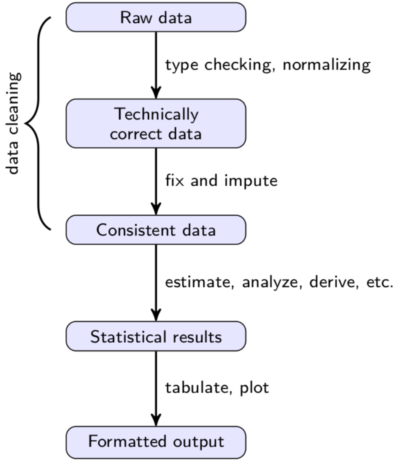
Data reshaping (2)

- `cast(data,formula,fun)` applies fun to melted data using formula (`reshape2` package)
- `recast(data,formula)` melts and casts in a single step (`reshape2` package)
- `reshape(x,direction...)` reshapes data frame between ‘wide’ (repeated measurements in separate cols) and ‘long’ (repeated measurements in separate rows) format based on direction
- `aggregate(x,by,fun)` input df; output df; applies fun to subsets of x, as grouped based on index.

Statistical analysis steps

R-omania Team

Concepts



data cleaning

Raw data

type checking, normalizing

Technically correct data

fix and impute

Consistent data

estimate, analyze, derive, etc.

Statistical results

tabulate, plot

Formatted output

(Edwin de Jonge, Mark van der Loo, *An introduction to data*

Technically correct data

- Well-defined format (data structure)
- Well-defined types (numbers, date/time, string, categorical...)
- Statistical units can be identified (persons, transactions, phone calls...)
- Variables can be identified as properties of statistical units.
- Note: tidy data $\subset$ technically correct data

Consistent data

- Data satisfies demands from domain knowledge (validation process)

Dirty tabular data

read.table vs. readr::read_csv

- `read.table`: R's swiss army knife
    - fairly strict (no sniffing)
    - very flexible
    - interface could be cleaner
- `readr::read_csv`
    - easy to switch between strict/lenient parsing
    - compact control over column types
    - fast
    - clear reports of parsing failure

reading with read.table (1)

```
dat <- read.table(file = "table/unnamed.csv",
  header = FALSE,
  col.names = c("age","height"),
  stringsAsFactors = FALSE,
  sep = ",")
dat
```

```
##   age height
## 1  21    6.0
## 2  42    5.9
## 3  18   5.7*
## 4  21   <NA>
```

reading with read.table (2)

```
class(dat$height)
```

```
## [1] "character"
```

```
dat$height <- as.numeric(dat$height)
```

```
## Warning: NAs introduced by coercion
```

```
dat
```

```
##   age height
## 1  21    6.0
## 2  42    5.9
## 3  18     NA
## 4  21     NA
```

define colClasses

- this will generate an error

```
#dat <- read.table(
#  file = "table/unnamed.csv",
#  header = FALSE,
#  col.names = c("age","height"),
#  colClasses = c("numeric","numeric"),
#  stringsAsFactors = FALSE,
#  sep = ",")
```

reading with the readr package (1)

- parse columns as ‘number’ (flexible)

```r
readr::read_csv("table/unnamed.csv",
    col_names=c("age","height"),
    col_types="nn")
```

```
## # A tibble: 4 × 2
##       age height
##     <dbl>  <dbl>
## 1     21    6.0
## 2     42    5.9
## 3     18    5.7
## 4     21     NA
```

reading with the readr package (2)

- parse columns as ‘double’ (strict)

```r
readr::read_csv("table/unnamed.csv",
    col_names=c("age","height"),
    col_types="dd")
```

```
## Warning: 1 parsing failure.
## row    col              expected actual
##   3 height no trailing characters      *

## # A tibble: 4 × 2
##     age height
##   <dbl>  <dbl>
## 1    21    6.0
## 2    42    5.9
```

R-omania Team

Real dirty data (1)

```
source("parse/parse_outfile.R")
to <- read_tof("parse/skylark-1d.out")
to
```

```
##   TRIM 3.61 :  TRend analysis and Indices for Monitoring
##   STATISTICS NETHERLANDS
##
##   Date/Time: 4-7-2016 15:08:28
##
##   Title :   skylark-1d
##
##   Comment: Example 1; using linear trend model
##
##   The following  5 variables have been read from file:
##   F:\TRIM\TRIM manual demo\skylark.dat
```

Real dirty data (2)

```
get_n_site(to)
```

```
## [1] 55
```

```
get_n_site_stringr(to)
```

```
## [1] 55
```

```
get_time_indices(to)
```

```
##   Time  Model std.err. Imputed std.err..1
## 1    1 1.0000   1.0000      NA         NA
## 2    2 1.0496   0.0149  0.8948     0.0410
## 3    3 1.1017   0.0312  0.9777     0.0601
## 4    4 1.1563   0.0491  0.9790     0.0678
```

Lessons learned

- (base) R has great text processing tools.
- Need to work with regular expressions[1]
- Write many small functions extracting single data elements.
- Don't overgeneralize: adapt functions as you meet new input.
- Smart use of existing tools (`read.table(text=)`)

[1] Mastering Regular Expressions (2006) by Jeffrey Friedl is a great resource

Packages for standard format parsing

- **`jsonlite`**: parse JSON files
- **`yaml`**: parse yaml files
- **`xml2`**: parse XML files
- **`rvest`**: scrape and parse HTML files

String normalization

Bring a text string in a standard format, e.g.

- Standardize upper/lower case (casefolding)
  - `stringr`: `str_to_lower`, `str_to_upper`, `str_to_title`
  - `base R`: `tolower`, `toupper`
- Remove accents (transliteration)
  - `stringi`: `stri_trans_general`
  - `base R`: `iconv`
- Re-encoding
  - `stringi`: `stri_encode`
  - `base R`: `iconv`
- Uniformize encoding (unicode normalization)
  - `stringi`: `stri_trans_nfkc` (and more)

Approximate text matching: edit-based distances

	Allowed operation			
Distance	substitution	deletion	insertion	transposition
Hamming	✔	✖	✖	✖
LCS	✖	✔	✔	✖
Levenshtein	✔	✔	✔	✖
OSA	✔	✔	✔	✔*
Damerau-Levenshtein	✔	✔	✔	✔

*Substrings may be edited only once.

"leela" → "leea" → "leia"

```
stringdist::stringdist("leela","leia",method="dl")
```

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

R-omania Team

Some pointers for approximate matching

- Normalisation and approximate matching are complementary
- See Mark Van Der Loo useR2014 talk or paper on stringdist for more distances
- The fuzzyjoin package allows fuzzy joining of datasets

R-omania Team

Other good stuff

- lubridate: extract dates from strings

```
lubridate::dmy("17 December 2015")
```

```
## [1] "2015-12-17"
```

- tidyr: many data cleaning operations to make your life easier
- readr: Parse numbers from text strings

```
readr::parse_number(c("2%","6%","0.3%"))
```

```
## [1] 2.0 6.0 0.3
```

Validation and Imputation

The `validate` package, in summary

- Make data validation rules explicit
- Treat them as objects of computation
  - store to / read from file
  - manipulate
  - annotate
- Confront data with rules
- Analyze/visualize the results

Use rules to correct data

Main idea

Rules restrict the data. Sometimes this is enough to derive a correct value uniquely.

Examples

- Correct typos in values under linear restrictions
  - $123 + 45 \neq 177$, but $123 + \underline{54} = 177$.
- Derive imputations from values under linear restrictions
  - $123 + \texttt{NA} = 177$, compute $177 - 123 = 54$.

Both can be generalized to systems $\mathbf{Ax} \leq \mathbf{b}$.

Validate

```
library(magrittr)
library(validate)
data(retailers)
head(retailers, 3)
```

```
##   size incl.prob staff turnover other.rev total.rev staf
## 1  sc0      0.02    75       NA        NA      1130
## 2  sc3      0.14     9     1607        NA      1607
## 3  sc3      0.14    NA     6886       -33      6919
##   total.costs profit vat
## 1       18915  20045  NA
## 2        1544     63  NA
## 3        6493    426  NA
```

A first glance

```
retailers %>%
  check_that(other.rev > 0, profit < turnover) %>%
  summary()
```

```
##   rule items passes fails nNA error warning        expre
## 1   V1    60     23     1  36 FALSE   FALSE     other.re
## 2   V2    60     48     4   8 FALSE   FALSE profit < tu
```

Define rules for reuse (1)

```
v <- validator(staff >= 0,
  turnover >= 0,
  other.rev >= 0,
  total.rev >= 0,
  turnover + other.rev == total.rev,
  if (staff > 0) staff.costs > 0
)
```

Define rules for reuse (2)

```
v
```

```
## Object of class 'validator' with 6 elements:
##  V1: staff >= 0
##  V2: turnover >= 0
##  V3: other.rev >= 0
##  V4: total.rev >= 0
##  V5: turnover + other.rev == total.rev
##  V6: !(staff > 0) | staff.costs > 0
```

```
summary(v)
```

```
##   block nvar rules linear
## 1     1    2     2      1
## 2     2    3     4      4
```

getters and setters for rule metadata (1)

```
created(v)
```

```
## [1] "2016-11-05 12:58:44 EET" "2016-11-05 12:58:44 EET"
## [3] "2016-11-05 12:58:44 EET" "2016-11-05 12:58:44 EET"
## [5] "2016-11-05 12:58:44 EET" "2016-11-05 12:58:44 EET"
```

```
origin(v)
```

```
## [1] "command-line" "command-line" "command-line" "comman
## [5] "command-line" "command-line"
```

```
names(v)
```

```
## [1] "V1" "V2" "V3" "V4" "V5" "V6"
```

getters and setters for rule metadata (2)

```
description(v)
```

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

```
cf <- confront(retailers, v)
cf
```

```
## Object of class 'validation'
## Call:
##     confront(x = retailers, dat = v)
##
## Confrontations: 6
## With fails    : 2
## Warnings      : 0
## Errors        : 0
```

getters and setters for rule metadata (3)

```r
summary(cf)
```

```
##   rule items passes fails nNA error warning
## 1   V1    60     54     0   6 FALSE   FALSE
## 2   V2    60     56     0   4 FALSE   FALSE
## 3   V3    60     23     1  36 FALSE   FALSE
## 4   V4    60     58     0   2 FALSE   FALSE
## 5   V5    60     19     4  37 FALSE   FALSE
## 6   V6    60     50     0  10 FALSE   FALSE
##                                       expression
## 1                                     staff >= 0
## 2                                  turnover >= 0
## 3                                 other.rev >= 0
## 4                                 total.rev >= 0
## 5 abs(turnover + other.rev - total.rev) < 1e-08
```

getters and setters for rule metadata (4)

```
aggregate(cf, by="record") %>% head(3)
```

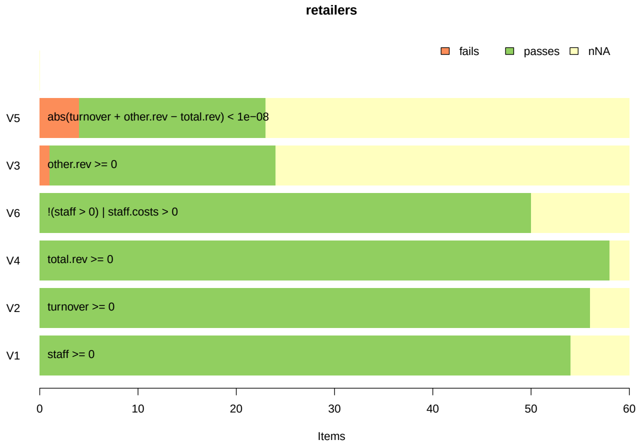
```
##   npass nfail nNA  rel.pass  rel.fail    rel.NA
## 1     2     0   4 0.3333333 0.0000000 0.6666667
## 2     4     0   2 0.6666667 0.0000000 0.3333333
## 3     3     2   1 0.5000000 0.3333333 0.1666667
```

```
sort(cf, by="rule") %>% head(3)
```

```
##    npass nfail nNA  rel.pass   rel.fail    rel.NA
## V5    19     4  37 0.3166667 0.06666667 0.6166667
## V3    23     1  36 0.3833333 0.01666667 0.6000000
## V6    50     0  10 0.8333333 0.00000000 0.1666667
```

getters and setters for rule metadata (5)

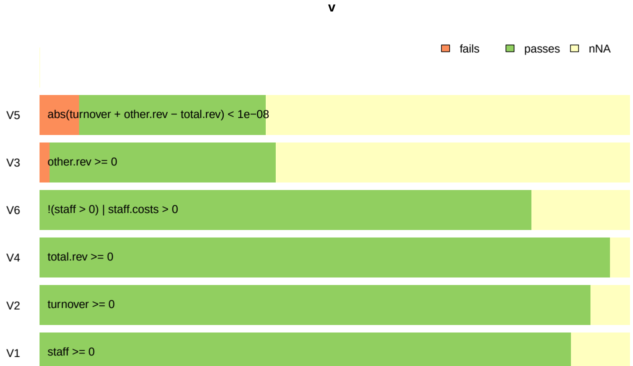
```
barplot(cf, main="retailers")
```



R-omania Team

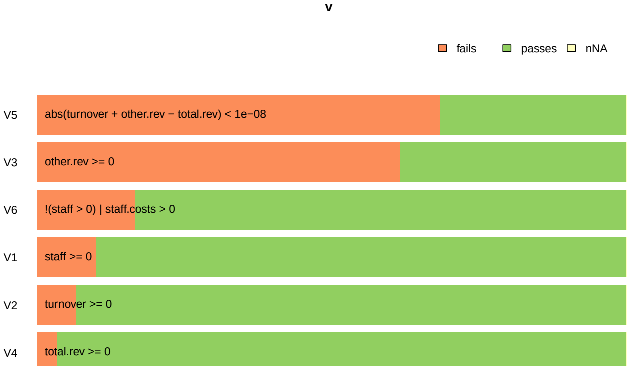
setting options (1)

```
retailers %>%
  confront(v, lin.eq.eps=1e-8) %>%
  barplot()
```



setting options (2)

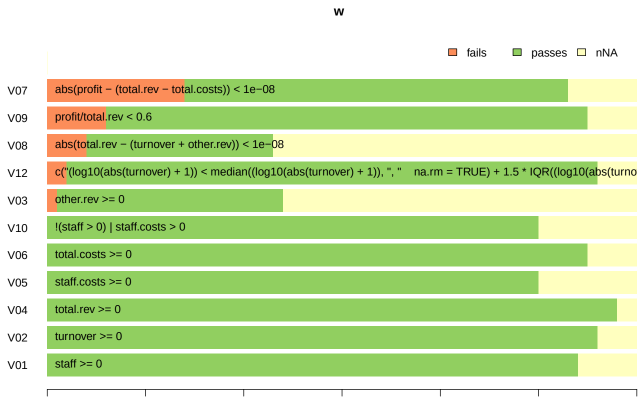
```
retailers %>%
  confront(v, na.value=FALSE) %>%
  barplot()
```



R-omania Team

Reading from file

```
w <- validator(.file="validate/rules.R")
confront(retailers,w) %>% barplot()
```



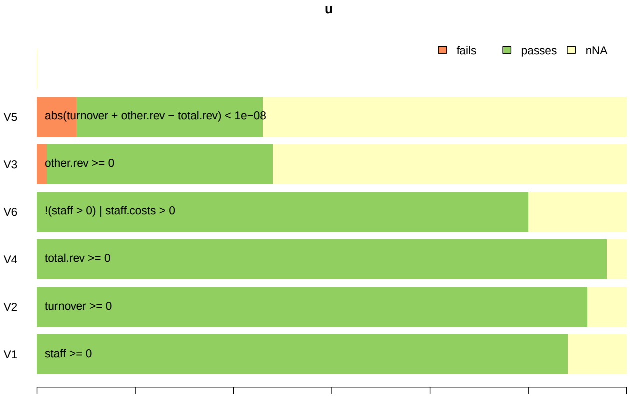
Rich metadata: yaml files (1)

```
u <- validator(.file="validate/rules.yaml")
u
```

```
## Object of class 'validator' with 6 elements:
##  V1 [nonnegative staff]               : staff >= 0
##  V2 [nonnegative turnover]            : turnover >= 0
##  V3 [nonnegative other rev]           : other.rev >= 0
##  V4 [nonnegative total rev]           : total.rev >= 0
##  V5 [Revenue balance]                 : turnover + other
##  V6 [positive staff => positive costs]: !(staff > 0) | s
```

Rich metadata: yaml files (2)

```
confront(retailers,u) %>%
  barplot()
```



R-omania Team

Error localization

Notes on `errorlocate`

- For in-record rules
- Support for
  - linear (in)equality rules
  - Conditionals on categorical variables (if male then not pregnant)
  - Mixed conditionals (has job then age $>=$ 15)
  - Conditionals w/linear predicates (staff $>$ 0 then staff cost $>$ 0)
- Optimization is mapped to MIP problem.

Missing values

Mechanisms (Rubin):

- **MCAR**: missing completely at random
- **MAR**: $P(Y = \texttt{NA})$ depends on value of X
- **MNAR**: $P(Y = \texttt{NA})$ depends on value of Y

Imputation

Purpose of imputation vs prediction

- Prediction: estimate a single value (often for a single use)
- Imputation: estimate values such that the completed data set allows for valid inference[a]

[a] This is very difficult!

Imputation methods

- Deductive imputation
- Imputation based on predictive models
- Donor imputation (knn, pmm, sequential/random hot deck)

Predictive model-based imputation

$$\hat{y} = \hat{f}(\boldsymbol{x}) + \epsilon$$

e.g.Linear regression

$$\hat{y} = \alpha + \boldsymbol{x}^T \hat{\beta} + \epsilon$$

- Residual:
  - $\epsilon = 0$ Impute expected value
  - ϵ drawn from observed residuals e
  - $\epsilon \sim N(0, \sigma)$ parametric residual, $\hat{\sigma}^2 = \mathrm{var}(e)$
- Multiple imputation (Bayesian bootstrap)
  - Draw β from parametric distribution, impute multiple times.

R-omania Team

Donor imputation (hot deck)

Method variants:

- **Random hot deck:** copy value from random record.
- **Sequential hot deck:** copy value from previous record.
- *k*-**nearest neighbours:** draw donor from *k* neares neigbours
- **Predictive mean matching:** copy value closest to prediction

Donor pool variants:

- per variable
- per missing data pattern
- per record

Note on multivariate donor imputation

Many multivariate methods seem relatively *ad hoc*, and more theoretical and empirical comparisons with alternative approaches would be of interest.

Andridge and Little (2010) *A Review of Hot Deck Imputation for Survey Non-response*. Int. Stat. Rev. **78**(1) 40-64

Methods supported by `simputation`

- Model based (optionally add [non-]parametric random residual)
  - linear regression
  - robust linear regression
  - CART models
  - Random forest
- Donor imputation (including various donor pool specifications)
  - k-nearest neigbour (based on gower's distance)
  - sequential hotdeck (LOCF, NOCB)
  - random hotdeck
  - Predictive mean matching
- Other
  - (groupwise) median imputation (optional random residual)
  - Proxy imputation (copy from other variable)

Simputation package

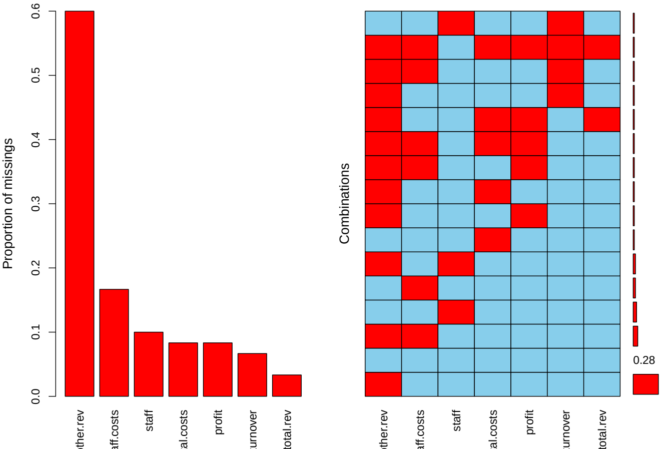
Investigate missing data patterns

```
library(VIM)
```

```
## Loading required package: colorspace

## Loading required package: grid

## Loading required package: data.table

## VIM is ready to use.
##  Since version 4.0.0 the GUI is in its own package VIMGU
##
##              Please use the package to use the new (and olc

## Suggestions and bug-reports can be submitted at: https:/

##
```

plot missing data patterns (1)

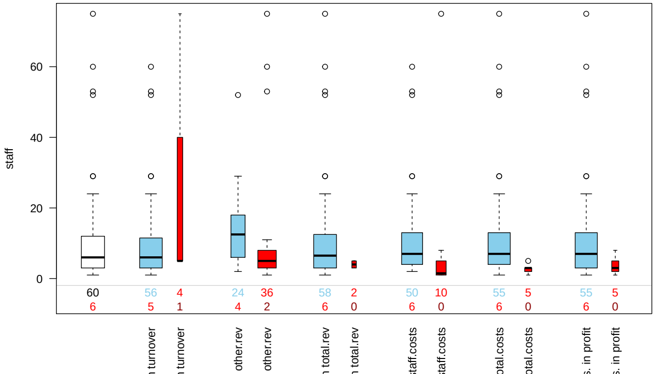
```
a <- VIM::aggr(retailers[3:9], sortComb=TRUE, sortVar=TRUE
```



R-omania Team

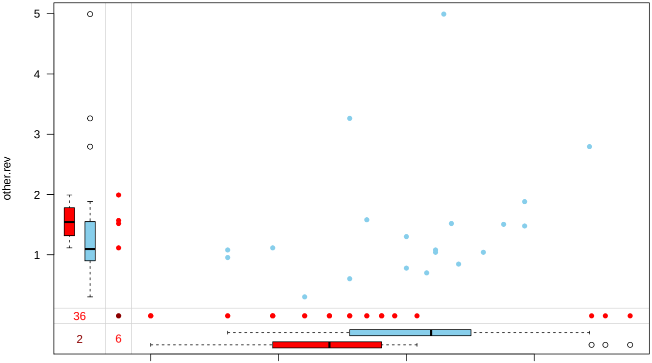
plot missing data patterns (2)

```
#
VIM::pbox(retailers[3:9],las=2)
```



plot missing data patterns (3)

```
dat <- log10(abs(retailers[c(3,5)]))
VIM::marginplot(dat, las=1, pch=16)
```



plot missing data patterns (4)

```
# testing means
t.test(log(staff) ~ is.na(other.rev), data=retailers)
```

```
##
##  Welch Two Sample t-test
##
## data:  log(staff) by is.na(other.rev)
## t = 2.7464, df = 46.014, p-value = 0.008572
## alternative hypothesis: true difference in means is not
## 95 percent confidence interval:
##  0.1985149 1.2880867
## sample estimates:
## mean in group FALSE  mean in group TRUE
##            2.329996            1.586695
```

Impute values using simputation

Linear model to impute three variables:

```
d1 <- impute_lm(retailers,   turnover ~ staff)
validate::cells(retailers, d1)
```

```
## Object of class cellComparison:
##
##    validate::cells(retailers, d1)
##
##                  D0001 D0002
## cells              600   600
## available          520   523
## missing             80    77
## still_available    520   520
## unadapted          520   520
## adapted              0     0
## imputed              0     3
```

Use staff as predictor for multiple variables

```
d2 <- impute_lm(retailers, turnover + other.rev + total.rev
validate::cells(retailers, d2)
```

```
## Object of class cellComparison:
##
##    validate::cells(retailers, d2)
##
##                  D0001 D0002
## cells              600   600
## available          520   559
## missing             80    41
## still_available    520   520
## unadapted          520   520
## adapted              0     0
## imputed              0    39
```

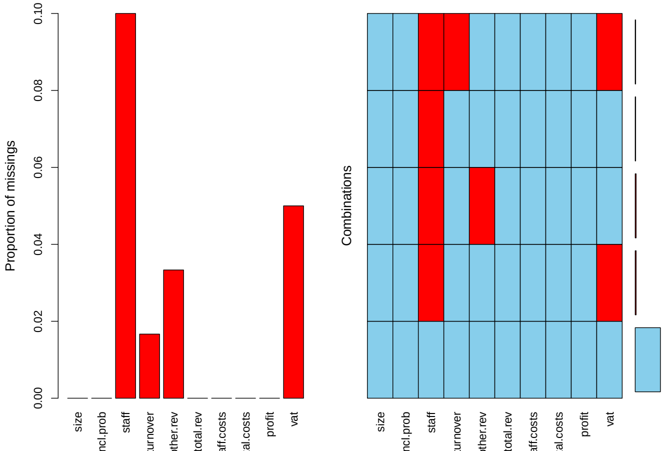
Impute everything excepts staff as a function of staff (1)

```
d3 <- impute_lm(retailers, . - staff ~ staff)
validate::cells(retailers, d3)
```

```
## Object of class cellComparison:
##
##    validate::cells(retailers, d3)
##
##                   D0001 D0002
## cells               600   600
## available           520   588
## missing              80    12
## still_available     520   520
## unadapted           520   520
## adapted               0     0
## imputed               0    68
```

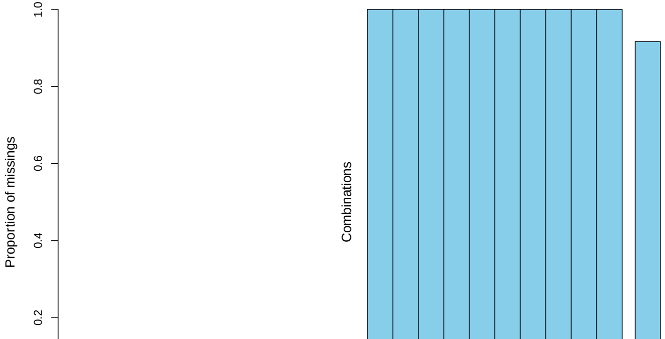
Impute everything excepts staff as a function of staff (2)

```r
aggr(d3)
```



Chain methods

```
d4 <- retailers %>%
  impute_lm(.-staff ~ staff) %>% # linear model
  impute_median(. ~ size)        # group median
aggr(d4)
```



R-omania Team

Some other methods

copy value from proxy

```
d5 <- impute_proxy(retailers, total.rev ~ vat)
validate::cells(retailers,d5)
```

```
## Object of class cellComparison:
##
##    validate::cells(retailers, d5)
##
##                  D0001 D0002
## cells              600   600
## available          520   521
## missing             80    79
## still_available    520   520
## unadapted          520   520
## adapted              0     0
## imputed              0     1
```

CART model imputation

```
d6 <- impute_cart(retailers, total.rev ~ .)
```

robust linear model with parametric residuals added

```
d7 <- impute_rlm(retailers,
        total.rev ~  staff,
        add_residual = "normal")
```

variance of estimation, including imputation by bootstrap (1)

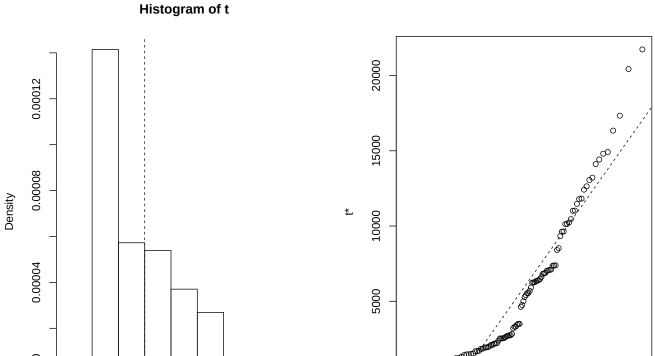
```
stat <- function(dat,i){
  dat <- dat[i,,drop=FALSE]
  dat %<>% impute_lm(staff.costs ~ staff + total.rev) %>%
    impute_lm(staff.costs ~ staff) %>%
    impute_median(staff.costs ~ size) %>%
    impute_const(staff.costs ~ 0)
  mean(dat[,"staff.costs"],na.rm=TRUE)
}
stat(retailers,seq_len(nrow(retailers)))
```

```
## [1] 6398.378
```

```
library(boot)
```

variance of estimation, including imputation by bootstrap (2)

```
b = boot(data=retailers, statistic = stat,R=100)
plot(b)
```



Credits

- `deductive` Mark van der Loo, Edwin de Jonge
- `errorlocate` Edwin de Jonge, Mark van der Loo
- `gower` Mark van der Loo
- `jsonlite` Jeroen Ooms, Duncan Temple Lang, Lloyd Hilaiel
- `magrittr` Stefan Milton Bache, Hadley Wickham
- `rex` Kevin Ushey Jim Hester, Robert Krzyzanowski
- `simputation` Mark van der Loo
- `stringdist` Mark van der Loo, Jan van der Laan, R Core, Nick Logan
- `stringi` Marek Gagolewski, Bartek Tartanus
- `stringr` Hadley Wickham, RStudio
- `tidyr` Hadley Wickham, RStudio
- `validate` Mark van der Loo, Edwin de Jonge
- `VIM` Matthias Templ, Andreas Alfons, Alexander Kowarik, Bernd Prantner
- `xml2` Hadley Wickham, Jim Hester, Jeroen Ooms, RStudio, R foundation

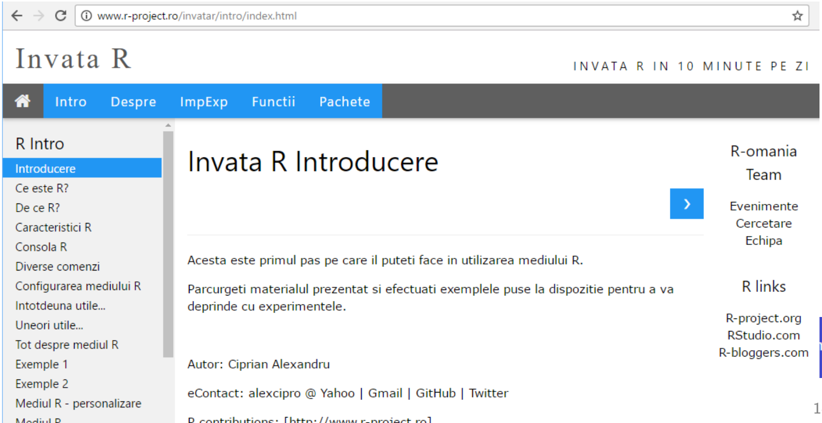
Thank you for your kind attention!

eContact: alexcipro @ Yahoo / Gmail / GitHub / Twitter

R-omania Team

Learn R - Invata R

Link:
[http://www.r-project.ro/invatar/intro/index.html]
Release date: 05.nov.2016



Invata R Introducere

Acesta este primul pas pe care il puteti face in utilizarea mediului R.

Parcurgeti materialul prezentat si efectuati exemplele puse la dispozitie pentru a va deprinde cu experimentele.

Autor: Ciprian Alexandru

eContact: alexcipro @ Yahoo | Gmail | GitHub | Twitter

R contributions: [http://www.r-project.ro]