R data validation course
Part 2

by Statistics Iceland, for/with our colleagues from:
Armenia, Azerbaijan, Belarus, Georgia, Kazakhstan, Kyrgyzstan,
Moldova, Tajikistan, Turkmenistan, Ukraine, Uzbekistan

March 2021

Organization

- Lectures (on 22, **24**, 26 of March 2021, from 9:00 GMT to 12:00 GMT, via interprefy platform)
- Practical sessions (23 and 25 of March 2021, from 9:00 GMT to 11:00 GMT, via TEAMS)
- Useful sharing at: https://github.com/violetacln/learnRval

R-packages frequently used:

```
# install.packages("tidyverse",
#                  "validate", "validatetools",
#                  "errorlocate", "dcmodify", "simputation",
#                  "DataExplorer", "funModeling",
#                   "sm", "LaplacesDemon", "DDoutlier", ## new
#                  "caret", "tidymodels",  # optional
#                  dependencies=TRUE)
```

Previously

Part 1: Reproducible, reviewed, improved

data processing and analysis with validation, as stages of GSBPM, using R

Part 2:

Workflow for *error detection*

based on:

expert rules and statistical analysis

2.1. Error detection with expert rules:

- structure of validation rules
- use of validation rules
- examples

2.2. Advanced validation with statistical analysis:

- univariate and multivariate distributions
- outliers
- data variability
- testing assumptions about input and output data
- note: multiple-source data

2.1. Error detection with expert rules

2.1.1. structure of validation rules: build, import/export, documentation

How to write the rules?

- All "R-statements that evaluate to a logical" <=>
  Primary validation of the validation rules :)
- Basic operators
  - binary, comparisons and logical $<, ==, \leq, \%in\%, \&, \&\&, |, ||$
  - logical, unary $!, all(), any()$
  - logical implications $if(C_1)\ C_2$
  - types *is.sometype*
- Pattern matching *grepl*()

Examples

```
library(validate)
df <- ggplot2::diamonds  # used in first example
v1  <- validator(depth >= 43)
# how did we know? min(ggplot2::diamonds$depth)
v2 <- validator( !all(depth <0) )
v3 <- validator(if(clarity=="IF") cut %in% c("Fair", "Good",
                              "Very Good", "Premium", "Ideal"))
v4 <- validator(is.numeric(price))
summary(confront(df, v1+v2+v3v4)) # can also do as.data.frame()
data("SBS2000")           # used in second example
v5sbs <- validator(
          grepl("^sc[0-9]$", size)
        , field_format(id, "^RET\\d{2}$" , type="regex") )
# and check
summary(confront(SBS2000, v5sbs))
```

(continue: how to write the rules)

- Functional dependency
  $X_1 \to X_2$ written as $X_1 \sim X_2$
- Operations which contain group-wise functions
  do_by(), *sum_by*(), ...
- Functions for simplifying particular checks:
  exists_any (one), (all) *is_unique* (complete), *field_format*, *contains_exactly* (at least, at most)

Examples

```
# same postcode implies same city
d <- data.frame(postcode =c("170", "600", "101"),
                city = c("Seltjarnarnes", "Akureyri", "Reykjavik"))
vfd <- validator(postcode ~ city)
confront(d, vfd)
# is there exactly one city called Seltjaranrnes for each postcode?
vv <- validator(exists_one(city=="Seltjarnarnes", by=list(postcode)))
summary(confront(d, vv))
# calculate median of turnover-variable for each size-variable
summary(confront(SBS2000,
        validator(do_by(turnover, by=size, fun=median, na.rm=TRUE)>300)))
```

Where to write/read the rules?

- R-objects:
  - validator-objects (class=validator)
    - created by the validator() function
    - similar to lists

```
library(validate)
data("iris")
c(validator(Sepal.Length>=0) , validator(Sepal.Width>=0))
```

- Data frames:
- saving rules *into* df: df <- as.data.frame(validator())
- reading rules *from* df: valobj <- validator(.data=df)

- Files: *text* and *yaml*
  - text file, for example *textFile.R*

```
# condition 1
C(X1, X2, ...)

# example
Sepal.Length>=0
```

From text file into validator object:

```
validation_object <- validator(.file="textFile.R")
```

- yaml file, for example *yamlFile.yaml*

```
---
include:
  - my_rules.yaml
  - your_rules.yaml

options:
  raise: errors
  lin.eq.eps: 0
  lin.ineq.eps: 0
---

C(X1, X2, ...)

Sepal.Length>=0
```

- Data bases: directly in/from data frames

ValidatorObject $\leftrightarrow$ *Dataframe* $\leftrightarrow$ *DataBase*

How/where to include documentation (metadata) about the rules?

- set / extract information about the rules with functions:
  - origin(), names(), created(), label(), description(), meta()
- simple reading of information:
  - $v[[n]]$ for n-th rule of validator v or
  - meta(v) for the whole data frame of rules
- properties of validators: summary(), length(), variables()

Examples

```
vdoc <- validator(positive_speed = speed >= 0,
                  ratio = speed/dist <= 1.5)
vdoc
names(vdoc)
names(vdoc)[1]
#meta(vdoc[1],"foo") <- 1
vdoc[[1]]
meta(vdoc)
summary(vdoc)
variables(vdoc, as="matrix")
```

Break?

Film about the automatic teacher with R-packages!
https://seankross.com/2017/09/25/Create-Videos-from-R-Markdown-Documents-with-Ari.html

2.1.2. use of validation rules: confront, manipulate, validate*!

Confront

```
cf <- confront(data_example, validator_object)
```

- Options
  - raise (all, errors), while default: everything is caught
  - lin.eq.eps (default 10^{-8})
- Reference data
  - compare data with other versions
  - validate domain of a variable by using a reference list of codes
- Functions
  - aggregate(cf), aggregate(cf, by='record')
  - sort(cf)
  - summary(cf), summary(cf[c(1,2)])

Example

```
d <- data.frame(postcode =c("170", "600", "101"),
                city = c("Seltjarnarnes", "Akureyri", "Reykjavik"))
vfd <- validator(city %in% c(city_reference$city))
d1 <-d$city
summary(confront(d, vfd, ref=list(city_refrence=d1)))
```

Manipulate rules (and sets of rules)

$c(v1, v2)$ or $v1 + v2$

$v[[2]]$

Try this with your own validator-objects!

Validate the validation rules!

(see https://github.com/data-cleaning/validatetools)

- Feasibility: is_infeasible(), make_feasible(), detect_infeasible_rules()
- Simplifying: simplify_rules()
- Removing redundancies: detect_redundancy(), remove_redundancy()
- Finding fixed values: simplify_fixed_values()

2.1.3. More examples about validation of validation rules :)

as seen at: validatetools - R - package page:
https://github.com/data-cleaning/validatetools

```
vrules <- validator( rule1 = price > 100
                   , rule2 = price < 100
                   )
validatetools::is_infeasible(vrules)
validatetools::detect_infeasible_rules(vrules)
validatetools::make_feasible(vrules)
validatetools::is_contradicted_by(vrules, "rule1")
```

```
vrules <- validator( if (age < 16) income == 0
                   , job %in% c("yes", "no")
                   , if (job == "yes") income > 0
                   )
validatetools::simplify_rules(vrules, age = 13)
```

https://github.com/data-cleaning/validatetools

```
vrules <- validator( price >= 100, price <=100)
validatetools::detect_fixed_variables(vrules)
#> $price
#> [1] 100
validatetools::simplify_fixed_variables(vrules)
```

```
vrules <- validator( r1 = if (income > 0) age >= 16
                   , r2 = age < 12
                   )
# age > 16 is always FALSE so r1 can be simplified
validatetools::simplify_conditional(vrules)
```

from https://github.com/data-cleaning/validatetools

```
vrules <- validator( rule1 = price > 1200
                   , rule2 = price > 1800
                   )

validatetools::detect_redundancy(vrules)
# rule1 is superfluous
validatetools::remove_redundancy(vrules)
```

2.2. Advanced validation with statistical analysis

2.2.1. univariate and multivariate distributions

```
# chose your favorite data set; df <- ggplot2::diamonds
# names of variables which are discrete and continuous
 dnames <-names(DataExplorer::split_columns(df)$discrete)
 cnames <- names(DataExplorer::split_columns(df)$continuous)
 # marginal distributions: see Part 1.
 # cumulative distribution functions of numerical variables
 plots_cumulative <- lapply(cnames, FUN=function(var) {
   ggplot2::ggplot(df, ggplot2::aes(df[[var]])) +
     ggplot2::stat_ecdf(geom = "point") +
     ggplot2::xlab(var) +
     ggplot2::ylab("cumulative prob")
 }
 )
 plots_cumulative
```

2.2.2. outliers

```
# any data set; example: df <- datasets::iris
# names of variables which are discrete and continuous
dnames <-names(DataExplorer::split_columns(df)$discrete)
cnames <- names(DataExplorer::split_columns(df)$continuous)

# qqplots of continuous variables
outliers_cont <- DataExplorer::plot_qq(df)

# boxplots by each discrete
outliers_by_Discretes <- lapply(dnames, FUN=function(varr) {
    DataExplorer::plot_boxplot( df, by=varr ,
                    geom_boxplot_args = list("outlier.color"="red"))
    }
    )
#outliers_by_Discretes
```

```r
# continue #
## univariate limits, using Tukey (inter-quartiles)
outliers_table_Tukey <-
  knitr::kable(
  lapply(cnames, FUN=function(x0) {
  c(
    x0,
    funModeling::tukey_outlier(as.data.frame(df)[[x0]])
  )
}
)
#, format="markdown"
, col.names = " ", caption="Interquartiles based: Tukey method"
)
outliers_table_Tukey
```

```r
## univariate limits, using Hampel (median based)
outliers_table_Hampel <-
   knitr::kable(
   lapply(cnames, FUN=function(x0) {

    c( x0, funModeling::hampel_outlier(df[[x0]]) )
  }
  )
  #, format="markdown"
  , col.names = " ", caption="Median based: Hampel"
)
outliers_table_Hampel
```

- *Optional* exercise

```
#testing DDoutlier-package
# https://cran.r-project.org/web/packages/DDoutlier/index.html

# chose a data set, then
X <- df
# connectivity based outlier detection, by setting an optional k
outlier_score <- DDoutlier::COF(dataset=X, k=8)
# sort and find index for most outlying observations
names(outlier_score) <- 1:nrow(X)
sort(outlier_score, decreasing = TRUE)
```

```r
#continue#

# inspect the distribution of outlier scores
DD_hist <- hist(outlier_score)
# classify observations (outlier detection based on distance calculations)
# so only numerical variables; if df <- datasets::iris, then only first 4
Distance_Based_classif <- cls_observations <-
      DDoutlier::DB(dataset=X[,1:4], d=1, fraction=0.05)$classification
## remove outliers from dataset if you really want to
XX <- X[cls_observations=='Inlier',]
```

- could use the non-parametric bootstrap procedure based on the bootlier test
- for *ts*, use anomaly detection methods

2.2.3. data variability

```
dnames <- names(DataExplorer::split_columns(df)$discrete)
cnames <- names(DataExplorer::split_columns(df)$continuous)
#continuous variables ---
print("mean, sd, skewness, standardised kurtosis and
        standardised 5th and 6th cumulants are
        calculated for continuous variables,
        by using the package SimMultiCorrData ")
c_res <- c("var1","", "", "", "", "", "")
var1 <- character()
for (var1 in cnames){
    c_res <- cbind(c_res, c(var1,
              SimMultiCorrData::calc_moments(df[[var1]])))
}
c_res
# or use bootstrap!
```

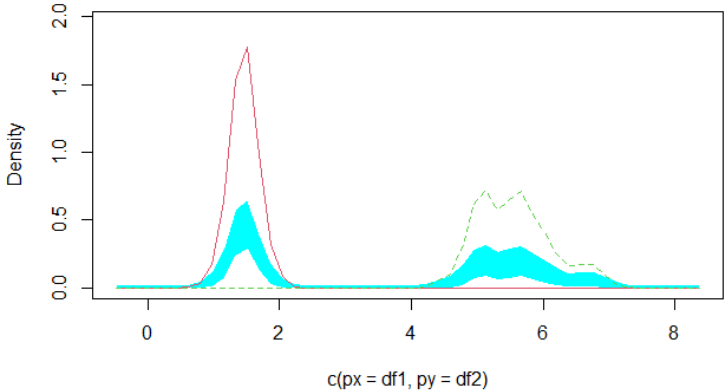
```
#continue: discrete variables ---
d_res <- c("var1","var2","","","", "")
var1 <- character()
var2 <- character()
for (var1 in dnames){
  for (var2 in dnames)
    {
    d_res <- cbind(d_res,
        c(var1, var2,
          funModeling::infor_magic(input =
          df[[var1]],target =df[[var2]])))
    }
}
d_res
```

en, *mi*, *ig*, *gr* are: maximum total entropy, mutual information/entropy, information gain between input and target, information gain ratio between input and target

2.2.4. testing assumptions about input and output data

```
print("the assumption about data to be checked: distributional difference")
# check if df1 and df2 come from different distributions:
# use resampling methods and/or KL measure
# example
df1 = iris[1:50,]$Petal.Length
df2= iris[101:150,]$Petal.Length
#continuous
  library(LaplacesDemon)
  kld <- LaplacesDemon::KLD(px=df1,py=df2)
  kld
#or resampling based tests, using sm::density.compare
  group.index <- rep( 1:2, c(length(df1), length(df2)) )
  sm::sm.density.compare(c(px=df1,py=df2),
                         group = group.index, model = "equal")

## note that a plot is generated automatically by this function
```


Density
0.0
0.5
1.0
1.5
2.0
0
2
4
6
8
c(px = df1, py = df2)

To follow

Part 3: Workflow for *analysis and dissemination of results*
with validation, as stages of GSBPM, using *R*

Assignment 2 (to be done mostly during class):

Download the Rmarkdown file *RvalPractice2.Rmd*.

Run, for a data set of your choice, each chunk of R-code included in the file.

Knit the whole file for producing a report in your favorite format: Word, html, pdf.

Thank you!

Discussion