

Validatetools

Edwin de Jonge, Statistics Netherlands

@edwindjonge | github.com/edwindj



Who am I?

- ▶ Data scientist / Methodologist at Statistics Netherlands (aka CBS).
- ▶ Author of several R-packages, including *whisker*, *validate*, *errorlocate*, *docopt*, *tableplot*, *chunked*, *ffbase*,...
- ▶ Co-author of *Statistical Data Cleaning with applications in R (2018)* (sorry for the plug, but relevant for this talk...)



CAUTION: BAD DATA



**BAD DATA QUALITY
MAY RESULT IN
FRUSTRATION AND
LEAD TO DROP
KICKING YOUR
COMPUTER**

Data cleaning...

A large part of your and our job is spent in data-cleaning:

- ▶ getting your data in the right shape (e.g. tidyverse)
- ▶ checking validity (e.g. validate, dataMaid, errorlocate)
- ▶ impute values for missing or erroneous data (e.g. VIM, simulation)
- ▶ see data changes, improvements (e.g. daff, lumberjack)

Desirable data cleaning properties:

- ▶ Reproducible data checks
- ▶ Automate repeated data checking (e.g. monthly/quarterly stats)
- ▶ Monitor data improvements / changes.
- ▶ But **How?**



Data Cleaning philosophy

- ▶ “**Explicit is better than implicit**”.
- ▶ Make data knowledge as **explicit** as possible.
- ▶ Store these as **validation rules**.

Advantages:

- ▶ Easy checking of rules
- ▶ Data quality statistics: how often is each rule violated?
- ▶ Allows for reasoning on rules: which variables are involved in errors?
- ▶ Simplifies rule changes and additions.

Rules

- ▶ Data rules are solidified domain knowledge.
- ▶ In statistical production: much domain knowledge available.
- ▶ Many rules, many diverse subpopulations.
- ▶ Real world knowledge e.g. :
 - age is not negative.
 - human age is less then 150 years.
- ▶ Expert knowledge, e.g:
 - IF profit > 0 THEN turnover > 0
 - IF married THEN age > 16

R package validate

With package validate you formulate explicit rules that data must conform to:

```
library(validate)
check_that( data.frame(age=160, job = "no", income = 3000),
  age >= 0,
  age < 150,
  job %in% c("yes", "no"),
  if (job == "yes") age >= 16,
  if (income > 0) job == "yes"
)
```


Rules (2)

A lot of datacleaning packages are using validate rules to facilitate their work.

- ▶ validate: validation **checks** and data **quality stats** on data.
- ▶ errorlocate to find **errors** in variables (in stead of records)
- ▶ rspa data **correction** under data constraints
- ▶ deductive deductive **correction**
- ▶ dcmodify deterministic **correction** and **imputation**.

Why-o-why validate tools?

- ▶ We have package validate, what is the need?

In “real life”

- ▶ We have large data analysis processes with many steps.
- ▶ Often many rules (> 100) on many variables (> 20) and many observations ($> 0.5M$) with diverse subpopulations.
- ▶ Results often in redundant or (partially) contradictive rule sets. . .

Because we'd like to...

- ▶ clean up rule sets (kind of meta-cleaning. . .).
- ▶ detect and resolve problems with rules:
 - Detect **conflicting** rules.
 - Remove **redundant** rules.
 - **Substitute** values and **simplify** rules.
 - Detect unintended rule **interactions**.
- ▶ check the rule set using formal logic (without any data!).
- ▶ solve these kind of fun problems :-)

Formal logic

Rule set S

A validation rule set S is a conjunction of rules r_i , which applied on record \mathbf{x} returns TRUE (valid) or FALSE (invalid)

$$S(\mathbf{x}) = r_1(\mathbf{x}) \wedge \cdots \wedge r_n(\mathbf{x})$$

Note

- ▶ a record has to comply to each rule r_i .
- ▶ it is thinkable that two or more r_i are in conflict, making each record invalid.

Formal logic (2)

Rule $r_i(x)$

A rule a disjunction of atomic clauses:

$$r_i(x) = \bigvee_j C_i^j(x)$$

with:

$$C_i^j(x) = \begin{cases} \mathbf{a}^T \mathbf{x} \leq b \\ \mathbf{a}^T \mathbf{x} = b \\ x_j \in F_{ij} \text{ with } F_{ij} \subseteq D_j \\ x_j \notin F_{ij} \text{ with } F_{ij} \subseteq D_j \end{cases}$$

Rule types

- ▶ *linear* restrictions
- ▶ *categorical* restrictions
- ▶ *if* statements with linear and categorical restrictions

If statement is Modus ponens:

$$\begin{aligned} & \text{if } P \text{ then } Q \\ \Leftrightarrow & P \implies Q \\ \Leftrightarrow & \neg P \vee Q \end{aligned}$$

Example

```
rules <- validator(  
  example = if (job == "yes") income > 0  
)
```

$$r_{\text{example}}(x) = \text{job} \notin \text{"yes"} \vee \text{income} > 0$$

```
print(rules)
```

```
## Object of class 'validator' with 1 elements:  
##  example: !(job == "yes") | (income > 0)
```

Mixed Integer Programming

Each rule set problem can be translated into a mip problem, which can be readily solved using a mip solver.

`validatetools` uses `lpSolveApi`.

$$\begin{aligned} &\text{Minimize } f(\mathbf{x}) = 0; \\ &\text{s.t. } \mathbf{R}\mathbf{x} \leq \mathbf{d} \end{aligned}$$

with \mathbf{R} and \mathbf{d} the rule definitions and $f(\mathbf{x})$ is the specific problem that is solved.

Problem: infeasibility

Problem

One or more rules in conflict:

- ▶ $S(x)$ always is 'FALSE, **no data correct**
- ▶ *happens more often than you think*

```
library(validatetools)
rules <- validator( is_adult = age >=21
                    , is_child = age < 18
                    )
is_infeasible(rules)
```

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



KEEP CALM

AND

**RESOLVE
CONFLICT**

Conflict, and now?

```
rules <- validator( is_adult = age >=21
                    , is_child = age < 18
                    )
# Find out which rule would remove the conflict
detect_infeasible_rules(rules)
```

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

```
# And its conflicting rule(s)
is_contradicted_by(rules, "is_adult")
```

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

- ▶ One of these rules needs to be removed
- ▶ Which one? Depends on human assessment. . .

Detecting and removing redundant rules

Rule r_1 may imply r_2 , so r_2 can be removed.

```
rules <- validator( r1 = age >= 18
                    , r2 = age >= 12
                    )
detect_redundancy(rules)
```

```
##      r1      r2
## FALSE  TRUE
```

```
remove_redundancy(rules)
```

```
## Object of class 'validator' with 1 elements:
##  r1: age >= 18
```



Value substitution

```
rules <- validator( r1 = if (gender == "male") weight > 50
                    , r2 = gender %in% c("male", "female")
                    )

substitute_values(rules, gender = "male")
```

```
## Object of class 'validator' with 2 elements:
##   r1           : weight > 50
##   .const_gender: gender == "male"
```

Conditional statement

A bit more complex reasoning, but still classical logic:

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

```
## Object of class 'validator' with 2 elements:  
##  r1: income <= 0  
##  r2: age < 12
```

All together now!

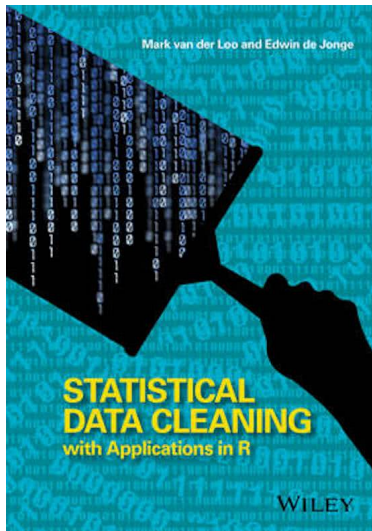
simplify_rules applies all simplification methods to the rule set

```
rules <- validator( r1 = if (age < 16) income == 0
                    , r2 = job %in% c("yes", "no")
                    , r3 = if (job == "yes") income > 0
                    )
simplify_rules(rules, job = "yes")
```

```
## Object of class 'validator' with 3 elements:
##   r1           : age >= 16
##   r3           : income > 0
##   .const_job: job == "yes"
```



Interested?



SDCR

M. van der Loo and E. de Jonge
(2018) *Statistical Data Cleaning
with applications in R* Wiley, Inc.

[validatetools](#)

► Available on [CRAN](#)

More theory?

← See book

Thank you for your attention! / Köszönöm a figyelmet!