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, recipes)
▸ checking validity (e.g. validate, dataMaid, errorlocate)
▸ impute values for missing or erroneous data (e.g. VIM, simputation, recipes)
▸ see data changes, improvements (e.g. daff, diffobj, lumberjack)

Desirable data cleaning properties:

▸ Reproducible data checks.
▸ Automate repetitive data checking (e.g. monthly/quarterly).
▸ Monitor data improvements / changes.
▸ How do this systematically?
Data Cleaning philosophy

▶ “Explicit is better than implicit”.
▶ Data rules are solidified domain knowledge.
▶ Store these as validation rules and apply these when necessary.

Advantages:

▶ Easy checking of rules: data validation.
▶ Data quality statistics: how often is each rule violated?
▶ Allows for reasoning on rules: which variables are involved in errors? How do errors affect the resulting statistic?
▶ Simplifies rule changes and additions.
R package validate

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

```r
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"
)
```
A lot of data cleaning 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?

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 :-)

---

**Note:** The text is a humorous way of explaining the necessity of validate tools, with a focus on cleaning and resolving rule sets, detecting conflicts, removing redundancies, substituting values, and simplifying rules. The emphasis is on the playful and problem-solving aspects of the toolset.
Problem: infeasibility

Problem

One or more rules in conflict: all data incorrect! (and yes that happens when rule sets are large . . .)

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

```r
## [1] TRUE
```
KEEP CALM
AND
RESOLVE CONFLICT
Conflict, and now?

```r
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:

```r
rules <- validator( r1 = if (income > 0) age >= 16 
                     , r2 = age < 12 
                 )

# age > 16 is always FALSE so r1 can be simplified
simplify_conditional(rules)
```

```r
## 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

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

simplify_rules(rules, job = "yes")
```

## Object of class 'validator' with 3 elements:
## r2 : income > 0
## r3 : age >= 16
## .const_job: job == "yes"
How does it work?

validatetools:

- reformulates rules into formal logic form.
- translates them into a mixed integer program for each of the problems.

Rule types

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

If statement is Modus ponens:

\[
\text{if } P \text{ then } Q \\
\iff P \implies Q \\
\iff \neg P \lor Q
\]
Example

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

\[
r_{\text{example}}(x) = \text{job} \not\in \text{"yes"} \lor \text{income} > 0
\]

```r
print(rules)
```

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

SDCR

validatetools

▶ Available on CRAN

More theory?
← See book

Thank you for your attention! / Köszönöm a figyelmet!
Addendum
Formal logic

Rule set $S$

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

$$S(x) = r_1(x) \land \cdots \land r_n(x)$$

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:

$$r_i(x) = \bigvee_j C^j_i(x)$$

with:

$$C^j_i(x) = \begin{cases} 
  a^T x \leq b \\
  a^T x = b \\
  x_j \in F_{ij} \text{ with } F_{ij} \subseteq D_j \\
  x_j \not\in F_{ij} \text{ with } F_{ij} \subseteq D_j 
\end{cases}$$
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.

\[
\text{Minimize } f(x) = 0; \\
\text{s.t. } Rx \leq d
\]

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