Statistical Data Cleaning with R

Mark van der Loo, Statistics Netherlands

@markvdloo | github.com/markvanderloo

use of R in official Statistics
Bucharest, 2017

# starring:
library(validate)
library(validatetools)
library(dcmodify)
library(errorlocate)
library(deductive)
library(VIM)
library(simputation)
library(rspa)
library(lumberjack)
These slides and code

Available via

http://www.github.com/markvanderloo/uRos2017
Who am I?

- PhD: theoretical chemistry (2008)
- Official Statistician since 2007
- Working with R (first S+) since 2007
- First package (extremevalues) on CRAN 2010

Van der Loo and De Jonge (2012, 2017)
Agenda

- Statistical value chain
- When is data ‘clean’?
- Where are the errors?
- Handling missing data
- Tracking changes in data
Statistical value chain
Statistical Value Chain

- Raw data: Extract and coerce
- Input data: Impute and adjust
- Valid: Analyze and infer
- Statistics: Format
- Output
Why the statistical value chain?

It is a way to

▶ Organize thoughts
▶ Separate production systems
▶ Store data in well-defined states
  – allows monitoring
  – allows for reuse and data sharing across the office

It is not

▶ A prescription for linear data processing.
▶ Forcing you to use these exact five steps.
Tip, or how I organize my analyses projects

- Each step is in a numbered folder (i.e. 01Raw)
- The folder contains R (Rmd) files that
  - Pull in the data from the previous folder, process it, and write the output.
  - There’s also a `readme.md` describing the output in that folder.
Data validation
Data validation

Definition

An activity, checking whether a combination of values comes from a predefined set of allowed value combinations.

ESS Handbook on methodology of data validation (2015)

Examples

► Is profit stored as a number (not text)?
► Is age >= 0?
► When age < 15, is job_status == "no job"?
► Is the mean turnover of companies in an economic sector positive?
An example Input dataset

```r
library(validate)
data(retailers)head(retailers[3:7],3)
```

```r
## staff turnover other.rev total.rev staff.costs
## 1 75 NA NA 1130 NA
## 2 9 1607 NA 1607 131
## 3 NA 6886 -33 6919 324

# we add a ID column for later use...
retailers$rec_id <- sprintf("%03d",1:nrow(retailers))
```

Van der Loo and De Jonge (2017b)
Data validation with the `validate` package

```r
# define validation rules
rules <- validator(
  turnover + other.rev == total.rev,
  turnover >= 0,
  other.rev >= 0,
  total.rev >= 0,
  if (staff > 0) staff.costs > 0
)

# confront data with ruleset
cf <- confront(retailers, rules, key="rec_id")
```
Data validation with the validate package

```
summary(cf)
```

## name items passes fails nNA error warning
## 1 V1 60 19 4 37 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 50 0 10 FALSE FALSE

## expression
## 1 abs(turnover + other.rev - total.rev) < 1e-08
## 2 (turnover - 0) >= -1e-08
## 3 (other.rev - 0) >= -1e-08
## 4 (total.rev - 0) >= -1e-08
## 5 !(staff > 0) | (staff.costs > 0)
Data validation with the \texttt{validate} package

\begin{verbatim}
barplot(cf, main="retailers")
\end{verbatim}
Data validation with the validate package

```r
as.data.frame(cf) %>% head()
```

```r
## rec_id name value expression
## 1 001 V1 NA abs(turnover + other.rev - total.rev) < 1e-08
## 2 002 V1 NA abs(turnover + other.rev - total.rev) < 1e-08
## 3 003 V1 FALSE abs(turnover + other.rev - total.rev) < 1e-08
## 4 004 V1 TRUE abs(turnover + other.rev - total.rev) < 1e-08
## 5 005 V1 NA abs(turnover + other.rev - total.rev) < 1e-08
## 6 006 V1 NA abs(turnover + other.rev - total.rev) < 1e-08
```
Overview

rules
validator

confront

results
validation
data

data.frame
Overview

Define $K = \{0, 1, \ldots, N\}$ a finite set of keys identifying objects and variables, and $D$ the union of all the variable domains.

Formal definition

A validation rule is a surjective function

$$v : D^K \rightarrow \{0, 1, \text{NA}\}$$

That takes a data set and outputs a logical or NA.

Implementation

In validate any statement that evaluates to a logical is considered a validation rule.
Rule import/export

Read/write validation rules from/to

- Commandline (shown before)
- Files (free-form, yaml)
- data.frame

```r
# Contents of rules.txt

# sanity of the 'turnover' variable
mean(turnover) >= 0
mean(turnover) > mean(profit)

# balance checks
turnover + other.rev == total.rev

rules <- validator(.file="rules.txt")
```
## Using validation results

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>summary</td>
<td>Short by-rule summary</td>
</tr>
<tr>
<td>aggregate</td>
<td>Aggregate by rule, or by record</td>
</tr>
<tr>
<td>sort</td>
<td>Aggregate and sort</td>
</tr>
<tr>
<td>values</td>
<td>Get raw output values</td>
</tr>
<tr>
<td>as.data.frame</td>
<td>Cast results into data frame</td>
</tr>
<tr>
<td>compare</td>
<td>Compare datasets wrt a rule set</td>
</tr>
<tr>
<td>cells</td>
<td>Compare cell values.</td>
</tr>
</tbody>
</table>
Rule management
Managing rule sets

Problem
Rule sets often grow organically and are hardly pruned. As a result, they may contain redundancies or contradictions rendering the rule set inefficient and hard to understand.

Idea
Weed out redundancies and contradictions automatically (as much as possible)
Rule management with `validatetools`

Recal our rule set

```
rules

## Object of class 'validator' with 5 elements:
## V1: turnover + other.rev == total.rev
## V2: turnover >= 0
## V3: other.rev >= 0
## V4: total.rev >= 0
## V5: !(staff > 0) | (staff.costs > 0)
```
Rule management with validatetools

library(validatetools)
rules <- simplify_rules(rules)

## No fixed values found.

rules

## Object of class 'validator' with 4 elements:
## V1: turnover + other.rev == total.rev
## V2: turnover >= 0
## V3: other.rev >= 0
## V5: staff <= 0 | staff.costs > 0
Changing data based on domain knowledge
Observation

Up to about 50% of changes in data may be done manually (or scripted) based on direct input of domain experts.

Question

Can we support use of external domain knowledge in the data cleaning process while separating it from the code?
Data modification with dcmodify

```r
library(dcmodify)
m <- modifier(
  if (other.rev < 0) other.rev <- -1 * other.rev
)
modified <- modify(retailers, m)

head(modified[3:7], 3)
```

<table>
<thead>
<tr>
<th></th>
<th>staff</th>
<th>turnover</th>
<th>other.rev</th>
<th>total.rev</th>
<th>staff.costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
<td>NA</td>
<td>NA</td>
<td>1130</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1607</td>
<td>NA</td>
<td>1607</td>
<td>131</td>
</tr>
<tr>
<td>3</td>
<td>NA</td>
<td>6886</td>
<td>33</td>
<td>6919</td>
<td>324</td>
</tr>
</tbody>
</table>
Data modification with dcmodify

Main idea

Separate domain knowledge from the main program flow.

Main features

- Define modifying rules on command line or separate text files
- Add metadata to modifying rules
- Read, inspect, manipulate, and apply rules to data
Error localization
Error localization

Question

Knowing that a record violates a number of rules, what fields do I need to change so I can fix things?

Answer

Find the smallest (weighted) number of fields whose values can be replaced so that all rules can be satisfied.

Fellegi and Holt (1976)
Error localization

# recall our rule set
rules

## Object of class 'validator' with 4 elements:
## V1: turnover + other.rev == total.rev
## V2: turnover >= 0
## V3: other.rev >= 0
## V5: staff <= 0 | staff.costs > 0
Locating errors with errorlocate

```r
library(errorlocate)
error_locations <- locate_errors(modified, rules)
values(error_locations)[30:37, 3:7]
```

<table>
<thead>
<tr>
<th></th>
<th>staff</th>
<th>turnover</th>
<th>other.rev</th>
<th>total.rev</th>
<th>staff.costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[2,]</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[3,]</td>
<td>FALSE</td>
<td>FALSE</td>
<td>NA</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[4,]</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[5,]</td>
<td>FALSE</td>
<td>FALSE</td>
<td>NA</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[6,]</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[7,]</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>[8,]</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

De Jonge and Van der Loo (2016)
Locating errors with `errorlocate`

```r
summary(error_locations)
```

## Variable:

### name errors missing

<table>
<thead>
<tr>
<th></th>
<th>turnover</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>total.rev</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>incl.prob</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>staff</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>other.rev</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>staff.costs</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>total.costs</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>profit</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>vat</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>rec_id</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

## Errors per record:

### errors records

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Replacing erroneous values

```r
# Replace erroneous values with NA (default)
fixable_data <- replace_errors(retailers, rules)

# check nr of missings
sum(is.na(retailers))

## [1] 80

sum(is.na(fixable_data))

## [1] 85
```
Deductive data cleaning
Deductive imputation

**impute_lr**

Derive unique imputations (where possible) based on linear restrictions.

**Example**

```
# ruleset
turnover + other.rev == total.rev
turnover  >=  0
other.rev >=  0
```

If `total.rev = 0`, then `turnover` and `other.rev` must equal 0.
Deductive imputation with deductive

```r
library(deductive)
lr_imputed <- impute_lr(fixable_data, rules)
# check nr of imputations using validate::cells
cells(start=retailers, fixable=fixable_data, impute_lr=lr_imputed, compare='sequential')
```

```r
## Object of class cellComparison:
##
## cells(start = retailers, fixable = fixable_data, impute_lr = lr_imputed, compare = 'sequential')
##
##   start  fixable  impute_lr
## cells    660     660     660
## available  580    580     575
## missing     80     80     85
## still_available  580    575     575
## unadapted   580    575     575
## adapted      0      0      0
## imputed      0      5      0
## new_missing   0      0     36
## still_missing 80     80     85
```

Van der Loo and De Jonge (2017c)
Deductive correction: typos in numbers

correct_typos

Check whether linear balance restrictions can be fixed by assuming a typographic error in one of the numbers.

Example

```
# ruleset
turnover + other.rev == total.rev
turnover  >= 0
other.rev >= 0
```

If turnover = 100, other.rev = 50 and total.rev = 105, swapping the last two digits in total.rev fixes the error.
Deductive correction with deductive

typos_corrected <- correct_typos(lr_imputed, rules[1:3])
Deductive correction with deductive

```r
# Compare progress on rule violation using validate::compare
compare(rules, lr_imputed, typos_corrected)
```

```r

<table>
<thead>
<tr>
<th>Status</th>
<th>Version</th>
<th>D0001</th>
<th>D0002</th>
</tr>
</thead>
<tbody>
<tr>
<td>validations</td>
<td></td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>verifiable</td>
<td></td>
<td>217</td>
<td>217</td>
</tr>
<tr>
<td>unverifiable</td>
<td></td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>still_unverifiable</td>
<td></td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>new_unverifiable</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>satisfied</td>
<td></td>
<td>217</td>
<td>217</td>
</tr>
<tr>
<td>still_satisfied</td>
<td></td>
<td>217</td>
<td>217</td>
</tr>
<tr>
<td>new_satisfied</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>violated</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>still_violated</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>new_violated</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Imputation
Visualization of missing values with VIM

VIM::aggr(typos_corrected[3:7])

Kowarik and Templ (2016)
Inspection of the missing data mechanism

VIM::pbox(typos_corrected[3:7], pos=1, las=2)
Imputing of missing values in R

Specialized packages

- Many available (VIM, mice, Amelia, mi, . . .)
- Interfaces vary (a lot)

DIY with model/predict

```r
m <- lm(Y ~ X, data=mydata)
ina <- is.na(mydata$Y)
mydata[ina, "Y"] <- predict(m, newdata = mydata[ina,])
```

- Code duplication, doesn’t always work
Idea of the simputation package

Provide

- a uniform interface,
- with consistent behaviour,
- across commonly used methodologies

To facilitate

- experimentation
- configuration for production
- Integration with other process steps
The simputation interface

`impute_<model>(data, <imputed vars> ~ <predictor vars>)`

Example: linear model imputation

```r
library(simputation)
typos_corrected[3:7] %>%
  impute_lm(other.rev ~ turnover) %>%
  head(3)
```

```r
## staff turnover other.rev total.rev staff.costs
## 1 75 NA NA 1130 NA
## 2 9 1607 0 1607 131
## 3 NA 6886 33 6919 324
```
Example: chaining imputations

typos_corrected[3:7] %>%
  impute_lm(other.rev ~ turnover + staff) %>%
  impute_lm(other.rev ~ staff) %>%
  head(3)

# staff turnover other.rev total.rev staff.costs
# 1 75 NA 4564.514 1130 NA
# 2 9 1607 0.000 1607 131
# 3 NA 6886 33.000 6919 324
Example: robust imputation ($M$-estimation)

typos_corrected[3:7] %>%
impute_rlm(other.rev ~ turnover + staff) %>%
impute_rlm(other.rev ~ staff) %>%
head(3)

##
### staff turnover other.rev total.rev staff.costs
### 1 75 NA 62.10186 1130 NA
### 2 9 1607 0.00000 1607 131
### 3 NA 6886 33.00000 6919 324
Example: Multiple variables, same predictors

```r
typos_corrected %>%
impute_rlm(other.rev + total.rev ~ turnover)
```

```r
typos_corrected %>%
impute_rlm( . ~ turnover ~ turnover)
```
Example: grouping

typos_corrected %>%
  impute_rlm(total.rev ~ turnover | size)

# or, using dplyr::group_by
typos_corrected %>%
  group_by(size) %>%
  impute_rlm(total.rev ~ turnover)
Example: add random residual

typos_corrected %>%
  \texttt{impute_rlm}(total.rev \sim turnover \mid size,
    add_residual=\texttt{"observed"})

typos_corrected %>%
  \texttt{impute_rlm}(total.rev \sim turnover \mid size,
    add_residual=\texttt{"normal"})
Example: train on A, apply to B

```r
m <- MASS::rlm(other.rev ~ turnover + staff
                , data=typos_corrected)
impute(retailers, other.rev ~ m)
```
Currently available methods in simputation

- Model based (optional random residual):
  - standard/$M$/elasticnet regression
  - CART models and Random forest

- Multivariate
  - EM-based imputation
  - missForest (=iterative random forest)

- Donor imputation (including various donor pool specifications)
  - k-nearest neigbour (based on gower’s distance)
  - sequential, random hotdeck
  - Predictive mean matching

- Other
  - (groupwise) median imputation (optional random residual)
  - Proxy imputation: copy another variable or use a simple transformation to compute imputed values.
Adjusting values to match restrictions
Value adjustment

Problem

Most (model-based) imputation methods do not take validation rules into account. Imputed records typically violate e.g. balance restrictions.

Idea

Minimally adjust the imputed numbers such that rules are satisfied:

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \| \mathbf{x} - \mathbf{x}^0 \|_{w}, \text{ s.t. } \mathbf{A} \mathbf{x} \leq \mathbf{b}$$

Pannekoek and Zhang (2012), Van der Loo (2017b)
Adjusting values with the `rspa` package

Step 1: Create an imputed dataset, remembering where the missings went.

```r
library(rspa)
# Create imputed set:
imputed <- typos_corrected %>%
  tag_missing() %>% # remember missings
  impute_median(staff ~ size) %>% # group-wise median
  impute_rlm( # robust regression
    turnover + other.rev + total.rev + staff.costs ~ staff)

# Check: is everything imputed?
sum(is.na(imputed[3:7]))
```

## [1] 0
Adjusting values with the \texttt{rspa} package

Step 2: adjust imputed values.

```r
valid <- imputed %>% match_restrictions(rules)

# Check: do we satisfy all rules?
valid %>% confront(rules, lin.eq.eps=0.01) %>% summary()
```

```r
## name items passes fails nNA error warning
## 1 V1 60 60 0 0 FALSE FALSE
## 2 V2 60 60 0 0 FALSE FALSE
## 3 V3 60 59 1 0 FALSE FALSE
## 4 V5 60 60 0 0 FALSE FALSE

## expression
## 1 abs(turnover + other.rev - total.rev) < 0.01
## 2 turnover >= 0
## 3 other.rev >= 0
## 4 staff <= 0 | staff.costs > 0
```
Logging changes in data
Logging changes in data

Question

I would like to know which operation had what influence on my data values, statistics, validation results.

Idea

- All data flows through the pipe `%>%`. It sees input and output.
- Construct a special pipe operator that measures and stores differences between in- and output.
The lumberjack operator %>>%

```r
library(lumberjack)
imputed <- typos_corrected %>>%
  start_log(log = validate:::lbj_cells()) %>>%
  impute_lm(turnover ~ staff) %>>%
  impute_median(other.rev + turnover ~ size) %>>%
  dump_log()

read.csv("cells.csv") %>% head()
```

<table>
<thead>
<tr>
<th></th>
<th>step</th>
<th>time</th>
<th>expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2017-11-09 10:14:39</td>
<td>impute_lm(turnover ~ staff)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2017-11-09 10:14:39</td>
<td>impute_median(other.rev + turnover ~ size)</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2017-11-09 10:14:39</td>
<td>cells available missing still_available unadapted adapted imputed</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>660</td>
<td>611</td>
<td>49</td>
<td>611</td>
<td>611</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>660</td>
<td>614</td>
<td>46</td>
<td>611</td>
<td>611</td>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>660</td>
<td>619</td>
<td>41</td>
<td>614</td>
<td>614</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|   |   |   |   |
|---|---|---|
| 1 | 0 | 49 |
| 2 | 0 | 46 |
| 3 | 0 | 41 |
The lumberjack package

Allows you to

▶ replace the magrittr pipe `%>%` with the lumberjack operator `%>>%`
▶ record what happens to your data as it flows through the `%>>%` pipe using loggers
▶ use a logger exported by lumberjack or validate, or define your own logger.

Van der Loo (2017c)
Conclusions

Main takeaway

► Domain knowledge should be separated from main programme flow
  ─ validation rules
  ─ modifying rules

► Formalizing domain knowledge in rules means you can compute on them!
► Many building blocks that integrate easily are available freely from R

Also

Please do not hesitate to send us questions, remarks, bug reports, on our packages, preferably through our github repo’s.
References

- ESS (2015) Methodology of data validation ESS handbook delivered by the ESSnet on validation.
- Fellegi, IP and D. Holt (1976) A systematic approach to automatic edit and imputation JASA
- Pannekoek, J. and Zhang, L.-C. Optimal adjustments for inconsistency in imputed data. Discussion paper 201219 Statistics Netherlands.
- Van der Loo and De Jonge (2017a) dcmodify: Modify Data Using Externally Defined Modification Rules. R package version 0.1.1
- Van der Loo, M. and De Jonge (2017b) validate: Data Validation Infrastructure. R package version 0.2.0.
- Van der Loo, M. and De Jonge (2017c) deductive: Data Correction and Imputation Using Deductive Methods R package version 0.1.3.
- Van der Loo, M. (2017) simputation: Simple Imputation R package version 0.2.2.
- Van der Loo, M. (2017c) lumberjack: Track Changes in Data R package version 0.2.0.