Validate observations stored in a DB

Edwin de Jonge @edwindjonge

Statistics Netherlands Research & Development
@edwindjonge

useR! 2021, July 6 2021

validatedb

(Virtual) Take home message

- Validate your database records, before statistical analysis.
- Build rules with `validate`, execute them with `validatedb`
- Built on `dbplyr`, so works on DBI supported database.

Data validation

- Is a first step in data-cleaning: is your data valid? (see Van der Loo and De Jonge 2018)

Contradicts domain knowledge?

- Does the data not contradict "real world"/domain knowledge?

Examples of domain knowledge demands

- age is non-negative
- turnover − costs = profit
- profit not larger then 60% of turnover
- children do not have an income.

Why validation?

- Communicate data quality without ambiguities
- Make knowledge explicit and organize it
- Create repeatable and comparable data quality reports
- Reuse ruleset for data cleaning purposes

`validate`: data validation rules

R package `validate` (ntbcw `validatedb`) , is a domain specific language to express demands in R on `data.frames`. (Loo and Jonge 2021)

- Each validation rule is a R expression.
- Data can be checked if it conforms to the rules.
- Allows for external storage/maintenance of rules.
- Allows to calculate statistics on data errors.
- Allows to reason over rules (e.g. contradictions) (e.g. `validatetools`).
- Allows for guided imputation/data correction, (e.g. `dcmodify`, `errorlocate`, `rspa`).

validate

Check if data is meets up to the domain knowledge

```
rules <- validator(age > 0, income >= 0)
cf <- confront(data, rules)

summary(cf)
```

However `validate` works on a `data.frame`

Large tables?

- `data.frame` and `data.table` are great, but require all data to be in memory.
- A common solution is to use a database table, and use selections or aggregations to for the analysis.
- Still need to do quality control on the data, aren't we?

Enter `validatedb`

- `validatedb` can be used as a drop-in replacement for `validate` on a data.frame'
- uses the machinery of `dbplyr` to translate the R checks into `SQL`
- Checks are `lazy`, can be stored in the database, or in a sparse format.

Let's look at an example

Suppose we have a database table `tbl_income`:

id	age	salary
a	12	5000
b	35	NA

```
print(tbl_income)
```

```
## # Source:   table<income> [?? x 3]
## # Database: sqlite 3.35.5 [:memory:]
##   id      age salary
##   <chr> <dbl>  <dbl>
## 1 a        12   5000
## 2 b        35     NA
```

Let's check its data quality

```
# Let's define a rule set and confront the table with it:
rules <- validator( is_working = if (salary > 0) age >= 15
                  , income     = salary >= 0
                  , mean_age   = mean(age,na.rm=TRUE) > 12
                  )

# and confront!
confronted <- confront(tbl_income, rules)
```

```
summary(confronted)
```

name	items	npass	nfail	nNA	warning	error	expression
is_working	2	0	1	1	FALSE	FALSE	if (salary > 0) age >= 15
income	2	1	0	1	FALSE	FALSE	(salary - 0) >= -1e-08
mean_age	1	1	0	0	FALSE	FALSE	mean(age, na.rm = TRUE) > 12

Result is lazy:

```
print(confronted)
```

```
## Object of class 'tbl_validation'
## Call:
##     confront.tbl_sql(tbl = dat, x = x, ref = ref, key = key, sparse = sparse)
##
## Confrontations: 3
## Tbl            : income (:memory:)
## Sparse         : FALSE
## Fails          : [??] (see `values`, `summary`)
## Errors         : 0
```

Result of each check/rule is a column:

```
values(confronted, type="tbl")
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.35.5 [:memory:]
##   is_working income mean_age
##        <int>  <int>    <int>
## 1          0      1        1
## 2         NA     NA        1
```

Automatic sql translation:

```
rules
```

```
## Object of class 'validator' with 3 elements:
##  is_working: !(salary > 0) | (age >= 15)
##  income    : salary >= 0
##  mean_age  : mean(age, na.rm = TRUE) > 12
```

translates to:

```
show_query(confronted)
```

```
## SELECT CASE WHEN (`salary` > 0.0) THEN (`age` >= 15.0) END AS `is_working`,
##   (`salary` - 0.0) >= -1e-08 AS `income`,
##   AVG(`age`) OVER () > 12.0 AS `mean_age`
## FROM `income`
```

Sparse representation

- A validation / confrontation results in a (lazy) table representing per record the results of the validation checks.
- Can be used to materialize or aggregate
- But also possible to store the result sparsely: only the invalid checks.

Sparse representation (2)

```
confronted_sparse <- confront( tbl_income
                             , rules
                             , sparse=TRUE
                             )
```

Results in only checks that were invalid:

```
values(confronted_sparse, type="tbl")
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.35.5 [:memory:]
##     row rule        fail
##   <int> <chr>      <int>
## 1     1 is_working     1
## 2     2 is_working    NA
## 3     2 income        NA
```

The end

Thank you for your attention!

Questions?

Curious?

```
install.packages("validatedb")
```

Feedback and suggestions?

https://github.com/data-cleaning/validatedb/issues

References

Loo, Mark P. J. van der, and Edwin de Jonge. 2021. "Data Validation Infrastructure for r." *Journal of Statistical Software, Articles* 97 (10): 1–31. https://doi.org/10.18637/jss.v097.i10.

Van der Loo, Mark, and Edwin De Jonge. 2018. *Statistical Data Cleaning with Applications in r*. Wiley Online Library.