

Assignment 3: data checking

Mark van der Loo and Edwin de Jonge

useR!2021

Exercise 1, variable checks

Variable checks are checks that can be performed on a field-by-field basis. For example checking if **Age** is non-negative, or of integer type. Variable checks are among the simplest checks.

We will use the **SBS2000** data set, that is included with **validate**.

Load the **SBS2000** data set.

```
library(validate)
data(SBS2000)
```

a) Check the variable type

The following **validator** defines a rule that column **turnover** should be a **numeric**. When confronted with a data set (e.g. **SBS2000**) we see that this is the case.

```
rules <- validator(
  is.numeric(turnover)
)
out <- confront(SBS2000, rules)
summary(out)
```

```
##   name items passes fails nNA error warning      expression
## 1   V1      1       1     0  0 FALSE   FALSE is.numeric(turnover)
```

Adjust the code, so it checks that **size** is a text variable and **staff** is a **integer**. Explain why the size rule fails.

b) Create a rules object using **is.na()** to check missing items in **turnover** and **profit**. **confront** the rules with **SBS2000** and summarize the results.

c) Create a rules object using **field_length** (or **nchar**) to check that

- **size** has at least 2 characters.
- **id** codes have exactly 5 characters.
- **size** has at least 2 and at most 3 characters.

confront the rules set with these rules.

d) Check with the function **number_format** which of the following records has two digits. Use **values** to print the “faulty” records.

e) As d) but now with at least two digits.

f) As d) but now with a check that the number should start with “2.” and two digits

Exercise 2

The functions `is_complete()` and `all_complete()` are convenience functions that test for missing values or combinations thereof in records. Have a look at their help files.

- a)
 - Create a rule set where you test ether `id` is complete.
 - Create a rules set where you test ether the combination `id` and `turnover` is complete.
 - **confront** the data with these rules.
- b) Balance restrictions occur for example in economic micro data, where financial balances must be met. Implement the following rules and check them on the data:
 - profit is total revue minus total costs.
 - total revenue is turnover plus other revenue
 - profit is at most 60% of total revenue.
- c) `validate` also accepts conditional rules: `if {rule_p} rule_q`; Execute the following:

Note in the summary that the rule is rewritten: it turns the check into a vectorized check (which an `if` statement isn't), so it can be checked efficiently in a `data.frame`.

Exercise 3

It is a good idea to store the validation rules apart from the data handling. This make it more easy to reuse a set of validation rules in different parts of the process and even share the rules to others to communicate which quality checks have been done on the data, and a user can test if the data complies.

- a) Put the following rules in a `rules.R` file

```
turnover >= 0

staff >= 0

# profits can be negative (not for long...)
profit == total.rev - total.costs
```

and create a validator with `validator(.file="rules.R")`.

- b) Rules can have metadata, this can be seen when turning a rule set into a `data.frame`
- c) Use `names(rules)` to set the `id` of the rules to `BR01`, `BR02` and `BR03`. You can do this in the same way as for renaming elements of a vector. Use the `label` function to set the label of the third rule to “profit def”. Use `print(rules)` to see the differences.
- d) Export the rule set to “`br.yml`” in `yaml` format using `export_yaml`. Open “`br.yml`” in RStudio (or another text editor) and copy the comments of the rules in **a)** into the descriptions of the `yaml` file. Read the `br.yml` file into a validator object.

Exercise 4

`validate` checks are normally executed in a `data.frame`. When your data is big, it is an option to execute the validation checks on a database using the pkg `validatedb`.

These are the same checks as `validate` but translated into `SQL` and executed on a `data.base`.

- a) Execute the following code:

```
library(validatedb)
# we are using a sqlite database in this demo
library(RSQLite)
```

```
# connect with the database file
con <- DBI::dbConnect(SQLite(), "SBS2000.db")

# retrieve a handle to the table "enterprise" in this database file.
enterprise <- tbl(con, "enterprise")

print(enterprise)
```

- b) Use `confront` to execute the rules on the database table and `print` and do a `summary`.
- c) Use `values` on the result of `confront` see the contents of the checks.
- d) Use `show_query` to print the SQL code that is executed on the database.