 Introduction to validate

Data Validation is an activity verifying whether or not a combination of values is a member of a set of acceptable combinations

The validate package is intended to make checking your data easy, maintainable, and reproducible. It does this by allowing you to

* test data against a reusable set of data validation rules:
* investigate, summarise, and visualise data validation results;
* import and export rule sets from and to various formats;
* filter, select and otherwise manipulate data validation rules’;
* investigate, summarise, and visualise rule sets.

For advanced rule manipulation there is the [validatetools](https://cran.r-project.org/package=validatetools) package.

1.1 A quick example

Here’s an example demonstrating the typical workflow. We’ll use the built-in cars data set, which contains 50 cases of speed and stopping distances of cars.

data(cars)

head(cars, 3)

## speed dist

## 1 4 2

## 2 4 10

## 3 7 4

Validating data is all about checking whether a data set meets presumptions or expectations you have about it, and the validate package makes it easy for you to define those expectations. Let’s do a quick check on variables in the cars data set. We first load the package, and create a list of data quality demands with the validator() function.

**library**(validate)

*rules* <- validator(speed >= 0

, dist >= 0

, speed/dist <= 1.5

, cor(speed, dist)>=0.2)

Here, the first three rules are record-wise checks: each record will yield one answer. In the last rule we check whether speed and distance are positively correlated this will yield a single TRUE or FALSE for the whole data set. We now confront the data with those rules and save the output into a variable called out.

out <- confront(cars, rules)

The easiest way to check the results is with summary().

summary(out)

## name items passes fails nNA error warning expression

## 1 V1 50 50 0 0 FALSE FALSE speed - 0 >= -1e-08

## 2 V2 50 50 0 0 FALSE FALSE dist - 0 >= -1e-08

## 3 V3 50 48 2 0 FALSE FALSE speed/dist <= 1.5

## 4 V4 1 1 0 0 FALSE FALSE cor(speed, dist) >= 0.2

This returns a data frame with one line of information for each rule V1, V2, V3 and V4. To be precise:

* How many data items were checked against each rule.
* How many items passed, failed or resulted in NA.
* Whether the check resulted in an error (could not be performed) or gave an warning.
* The expression that was actually evaluated to perform the check.

The same information can be summarized graphically as follows .

plot(out)

In this plot each horizontal bar indicates the percentage of Failing, Passing, and Missing cases. The table in the legend lists the total number of Fails, Passes and Missings, summed over all checks. Here, we have 4 rules. The first three rules yield 50 results each, while the last rule yields a single result. Hence there are 151 validation results in total.

Using the function violating we can select the records that violate one or more rules. We select only the first three results because the last rule can not be interpreted record by record.

violating(cars, out[1:3])

## speed dist

## 1 4 2

## 3 7 4

We can extract all individual results using for example as.data.frame.

df\_out <- as.data.frame(out)

head(df\_out, 3)

## name value expression

## 1 V1 TRUE speed - 0 >= -1e-08

## 2 V1 TRUE speed - 0 >= -1e-08

## 3 V1 TRUE speed - 0 >= -1e-08

We see that in record 1, rule V1, was satisfied (the result is TRUE), and that validate left a bit of slack when executing the rule, to avoid false negatives caused machine rounding issues.

Summarizing, the basic workflow in validate is to create a rule set, confront a data set with the rules in the rule set, and than analyze or use the results further. To understand which checks you can perform with validate you only need to remember the following.

Any R expression that results in a logical is accepted by validate as a validation rule.

2 Variable checks

Variable checks are checks that can be performed on a field-by-field basis. An example is checking that a variable called Age is nonnegative, or of integer type. Variable checks are among the simplest checks.

**Data**

In this section we will use the SBS2000 dataset, that is included with validate.

**library**(validate)

data(SBS2000)

head(SBS2000, 3)

## id size incl.prob staff turnover other.rev total.rev staff.costs

## 1 RET01 sc0 0.02 75 NA NA 1130 NA

## 2 RET02 sc3 0.14 9 1607 NA 1607 131

## 3 RET03 sc3 0.14 NA 6886 -33 6919 324

## total.costs profit vat

## 1 18915 20045 NA

## 2 1544 63 NA

## 3 6493 426 NA

2.1 Variable type

In R, one can test the type of a variable using built-in functions such as is.numeric or is.character.

is.character("hihi")

## [1] TRUE

is.character(3)

## [1] FALSE

In validate, any function starting with is. (‘is’ followed by a dot) is considered a validation function.

rules <- validator(

is.character(size)

, is.numeric(turnover)

)

out <- confront(SBS2000, rules)

summary(out)

## name items passes fails nNA error warning expression

## 1 V1 1 0 1 0 FALSE FALSE is.character(size)

## 2 V2 1 1 0 0 FALSE FALSE is.numeric(turnover)

We see that each rule checks a single item, namely one column of data. The first rule is violated (it is in fact a factor variable). The second rule is satisfied.

2.2 Missingness

Use R’s standard is.na() to check missing items in individual variables. Negate it to check that values are available.

rule <- validator(

!is.na(turnover)

, !is.na(other.rev)

, !is.na(profit)

)

out <- confront(SBS2000, rule)

summary(out)

## name items passes fails nNA error warning expression

## 1 V1 60 56 4 0 FALSE FALSE !is.na(turnover)

## 2 V2 60 24 36 0 FALSE FALSE !is.na(other.rev)

## 3 V3 60 55 5 0 FALSE FALSE !is.na(profit)

We see that in 4 cases the variable turnover is missing, while other.rev and profit are missing respectively in 36 and 5 occasions.

To demand that all items must be present or absent for a certain variable, use R’s quantifiers: any() or all(), possibly negated.

rules <- validator(

!any(is.na(incl.prob))

, all(is.na(vat)) )

out <- confront(SBS2000, rules)

summary(out)

## name items passes fails nNA error warning expression

## 1 V1 1 1 0 0 FALSE FALSE !any(is.na(incl.prob))

## 2 V2 1 0 1 0 FALSE FALSE all(is.na(vat))

2.3 Field length

The number of characters in text fields can be tested using either R’s standard nchar() function, or with the convenience function field\_length.

rules <- validator(

nchar(as.character(size)) >= 2

, field\_length(id, n=5)

, field\_length(size, min=2, max=3)

)

out <- confront(SBS2000, rules)

summary(out)

## 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 60 0 0 FALSE FALSE

## expression

## 1 nchar(as.character(size)) >= 2

## 2 field\_length(id, n = 5)

## 3 field\_length(size, min = 2, max = 3)

One advantage of check\_field\_length is that its argument is converted to character (recall that size is a factor variable). The function field\_length can be used to either test for exact field lengths or to check whether the number of characters is within a certain range.

The field length is measured as the number of [code points](https://en.wikipedia.org/wiki/Code_point). Use type="width" to measure the printed width (nr of columns) or type="bytes" to count the number of bytes.

2.4 Format of numeric fields

For numbers that are stored in character type, there is a convenience function called number\_format() that accepts a variable name and a format specification.

dat <- data.frame(x = c("2.54","2.66","8.142","23.53"))

To check that the numbers are formatted with one figure before, and two figures after the decimal point, we perform the following check.

rule <- validator( number\_format(x, format="d.dd"))

values(confront(dat, rule))

## V1

## [1,] TRUE

## [2,] TRUE

## [3,] FALSE

## [4,] FALSE

Here, the specification format="d.dd" describes the allowed numeric formats. In this specification the "d" stands for a digit, any other character except the asterisk (\*) stands for itself. The asterisk is interpreted as ‘zero or more digits’. Here are some examples of how to define number formats.

| **format** | **match** | **non-match** |
| --- | --- | --- |
| 0.dddd | "0.4321" | "0.123","1.4563" |
| d.ddEdd | "3.14E00" | "31.14E00" |
| d.\*Edd | "0.314E01","3.1415297E00" | "3.1415230" |
| d.dd\* | "1.23", "1.234",…… | "1.2" |

The last example shows how to check for a minimal number of digits behind the decimal point.

There are special arguments to check the number of decimal figures after the decimal separator.

x <- c("12.123","123.12345")

number\_format(x, min\_dig=4)

## [1] FALSE TRUE

number\_format(x, max\_dig=3)

## [1] TRUE FALSE

number\_format(x, min\_dig=2, max\_dig=4)

## [1] TRUE FALSE

number\_format(x, min\_dig=2, max\_dig=10)

## [1] TRUE TRUE

*# specify the decimal separator.*

number\_format("12,123", min\_dig=2, dec=",")

## [1] TRUE

The arguments min\_dig, max\_dig and dec are ignored when format is specified.

This function is convenient only for fairly simple number formats. Generic pattern matching in strings is discussed in the next section.

2.5 General field format

A simple way to check for more general format is to use [globbing patterns](https://en.wikipedia.org/wiki/Glob_(programming)). In such patterns, the asterisk wildcard character (\*) is interpreted as ‘zero or more characters’ and the question mark (?) is interpreted as ‘any character’.

For example, to check that the id variable in SBS2000 starts with "RET", and that the size variable has consists of "sc" followed by precisely one character, we can do the following.

rule <- validator(field\_format(id, "RET\*")

, field\_format(size, "sc?" ))

out <- confront(SBS2000, rule)

summary(out)

## name items passes fails nNA error warning expression

## 1 V1 60 60 0 0 FALSE FALSE field\_format(id, "RET\*")

## 2 V2 60 60 0 0 FALSE FALSE field\_format(size, "sc?")

Here, the globbing pattern "RET\*" is understood as ’a string starting with "RET", followed by zero or more characters. The pattern "sc?" means ’a string starting with "sc", followed by a single character.