Package ‘GenericValidationRules’

June 2, 2020

Type Package

Title Generic Validation Rules for the European Statistical System

Version 0.5.0

Date 2019-07-12

Author Olav ten Bosch, Dick Windmeijer, Mark van der Loo

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

Description Implements convenience functions for the 20 most often occurring types of data validation rules within the European Statistical System. This package can be used standalone but it also integrates with the 'validate' package.

License MIT + file LICENSE

LazyData no

Imports utils, stats

Suggests tinytest, validate

RoxygenNote 7.1.0

Encoding UTF-8

R topics documented:

COC — *Code consistency*

Description

Check that the codes used in fields are consistent with other codes used in another field of the same record, the same field in different records of the same file or in different datasets from the same country.

Note

The interface proposed in the original document contains redundancies, and it is easier to express this rule type directly in **validate** syntax as shown in the examples below.

References

Main types of validation rules for ESS data: COC

See Also

Other validation-functions: `COV()`, `FDL()`, `FDM()`, `FDT()`, `RNR()`, `RTS()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
# First example: consistency of TABLE and FREW
library(validate)
data(COCdat)
rules <- validator(
   if ( TABLE == "T01" ) FREQ == "A"
 , if ( TABLE == "T02" ) FREQ == "Q")

result <- confront(COCdat, rules)
summary(result)
values(result)
as.data.frame(result)

# Second example: consistency of TABLE and FREQ
data(COC2dat)
result <- confront(COC2dat, rules)
summary(result)
values(result)
as.data.frame(result)

# Third example: country must be EL. Envelope data can be passed
# as a single-row data frame
data(COC3dat)
rules <- validator(REPORTING == envelope$Country)
env <- data.frame(Country="EL", stringsAsFactors=FALSE)
result <- confront(COC3dat, rules, ref=list(envelope=env))
summary(result)
values(result)
as.data.frame(result)

# Fourth example: REPORTING country and PARTNER country cannot be the same
data(COC4dat)
# we convert to character as in the original data, these variables are
# different types of 'factor' (categorical) variables.
rules <- validator(as.character(REPORTING) != as.character(PARTNER) )
result <- confront(COC3dat, rules)
summary(result)
values(result)
as.data.frame(result)
```

`COV` *COV: Codes are Valid*

Description

COV: Codes are Valid

Usage

```
COV(d, codelistTable = NULL, codelist = NULL)
```

Arguments

`d`	When used in a validation rule, a bare (unquoted) name of a variable. Otherwise a vector of class `character`. Coerced to character as necessary.
`codelistTable`	A character scalar: name of a csv file (US convention) with at least a column `Codes`, containing the valid codes.
`codelist`	character vector with valid codes. When used in a validation rule, this must be a literal character vector (i.e. not a variable defined elswhere in your script).

References

Main types of validation rules for ESS data: COV

See Also

Other validation-functions: `COC`, `FDL()`, `FDM()`, `FDT()`, `RNR()`, `RTS()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
data(COVdat)

# Using COV with 'validate'
library(validate)
rule <- validator(COV(DIRECTION, codelist=c("IN", "OUT")) == TRUE)
cf <- confront(COVdat, rule)
summary(cf)
as.data.frame(cf)

# Using COV directly
COV(COVdat$DIRECTION, codelist=c("IN", "OUT"))

# Using 'validate' directly
rule <- validator( DIRECTION %in% c("IN", "OUT") )
```

`COVdat` *Example data sets from the Eurostat Generic Validation Rules*

Description

These are the datasets from the Eurostat document where the generic validaiton rules are defined. They can be loaded with `data(name)`, for example `data(FDLdat)`. The following datasets are available:

- `COVdat`
- `COCdat`
- `COC2dat`
- `COC3dat`
- `COC4dat`
- `COC5dat`
- `FDLdat`
- `FDL2dat`
- `FDMdat`
- `REPdat`
- `RNRdat`
- `RWDdat`
- `FDLdat`
- `FDTdat`
- `RETdat`
- `RTSdat`

- `VIRdat`
- `VCOdat`
- `VADdat`
- `HT_AGE_GROUPSdat`
- `VSAdat`

Format

csv files.

`FDL` *Check that the length of the data in the field is acceptable*

Description

Check that the length of the data in the field is acceptable

Usage

```
FDL(d, x = NULL, min = NULL, max = NULL, minDec = NULL, maxDec = NULL)
```

Arguments

`d`	When used in a validation rule, a bare (unquoted) name of a variable. Otherwise a vector of class `character`. Coerced to character as necessary.
`x`	Number of code points required.
`min`	Mimimum number of code points
`max`	Maximum number of code points
`minDec`	minimum number of decimal positions
`maxDec`	maximal number of decimal positions

Details

The number of code points (string length in terms of human-readable characters) may depend on current locale settings or encoding issues including those caused by inconsistent choices of `UTF` normalization.

References

Main types of validation rules for ESS data: FDL

See Also

Other validation-functions: `COC`, `COV()`, `FDM()`, `FDT()`, `RNR()`, `RTS()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
data(FDLdat)

# Using 'validate' directly
library(validate)
## Minimum nr of characters
rule <- validator( nchar(PARTNER) >= 2 )
cf <- confront(FDLdat, rule)
summary(cf)
as.data.frame(cf)


## Minimum and Maximum
rule <- validator( nchar(PARTNER) >= 2, nchar(PARTNER) <= 4 )
cf <- confront(FDLdat, rule)
summary(cf)
as.data.frame(cf)

# Using FDL with 'validate'
rule <- validator(FDL(PARTNER, x=2) == TRUE)
cf <- confront(FDLdat, rule)
summary(cf)
as.data.frame(cf)

rule <- validator(FDL(PARTNER, min=2, max=4) == TRUE)
cf <- confront(FDLdat, rule)
summary(cf)
as.data.frame(cf)

# Using FDL directly
FDL(FDLdat$PARTNER, x=2)
FDL(FDLdat$PARTNER, min=2, max=4)
```

FDM *Field is Mandatory or empty*

Description

Field is Mandatory or empty

Usage

```
FDM(d, mandatoryLevel = c("Mandatory", "Empty"))
```

Arguments

d	When used in a validation rule, a bare (unquoted) name of a variable. Otherwise a vector of class `character`. Coerced to character as necessary.
mandatoryLevel	character scalar indicating whether a variable must be filled or must be empty.

References

Main types of validation rules for ESS data: FDM

See Also

Other validation-functions: `COC`, `COV()`, `FDL()`, `FDT()`, `RNR()`, `RTS()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
data(FDMdat)

# Example using 'validate' directly
library(validate)

## OBS_VALUE is not an empty string and not NA ("Mandatory")
rules <- validator(OBS_STATUS != "" & !is.na(OBS_VALUE) )
summary( confront(FDMdat, rules) )

## OBS_VALUE is empty
rules <- validator(OBS_STATUS == "" | is.na(OBS_VALUE) )
summary( confront(FDMdat, rules) )

# Example using FDM with 'validate'
## OBS_STATUS must be empty
rule <- validator(FDM(OBS_STATUS, mandatoryLevel="Empty") == TRUE)
cf <- confront(FDMdat, rule)
summary(cf)
as.data.frame(cf)

## OBS_STATUS is mandatory
rule <- validator(FDM(OBS_STATUS, mandatoryLevel="Mandatory") == TRUE)
cf <- confront(FDMdat, rule)
summary(cf)
as.data.frame(cf)

# Example using FDM directly
FDM(FDMdat$OBS_STATUS, mandatoryLevel="Mandatory")
FDM(FDMdat$OBS_STATUS, mandatoryLevel="Empty")
```

FDT *Field type*

Description

Test whether a variable is of the required 'field type'.

Usage

```
FDT(
  d,
  ft = c("Alphabetic", "Alphanumeric", "Numeric", "NumericWithDecimals"),
  exceptions = "NA"
)
```

Arguments

`d`	When used in a validation rule, a bare (unquoted) name of a variable. Otherwise a vector of class `character`. Coerced to character as necessary.
`ft`	`[character]` Field type.
`exceptions`	`[character]` vector of acceptable values, beyond `"Numeric"` or `"NumericWithDecimals"`.

Details

The sets of 'Alphabetic' and 'Alphanumeric' characters are determined by the POSIX named ranges `"[:alpha:]"` respectively `"[:alnum:]"`. The interpretation of these character ranges depends on the current `locale`, see `regex`. Numeric values are those that can be coerced to integer or are in the list of `exceptions`. Acceptable NumericWithDecimals are numbers that have at least a single decimal after the decimal separator '`.`' (it is not required to have a number before it).

References

- Main types of validation rules for ESS data: FDT
- POSIX regular expressions. The Open Group base specifications Issue 6.

See Also

Other validation-functions: `COC`, `COV()`, `FDL()`, `FDM()`, `RNR()`, `RTS()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
data(FDTdat)

# Using FDT with 'validate'
library(validate)
rules <- validator(FDT(OBS_VALUE, ft="Numeric", exceptions="NA")==TRUE)
```

```
cf <- confront(FDTdat, rules)
summary(cf)
as.data.frame(cf)

# Using FDT directly
FDT(FDTdat$OBS_VALUE, ft="Numeric", exceptions="NA")
```

`period_to_int` *Turn a period into an integer*

Description

Annual periods are turned in to the integer year. Quarterly and Monthly periods are turned in to the month number, counted from the year zero, so quarters and months have consecutive numbers accross years.

Usage

```
period_to_int(x, from = c("annual", "quarterly", "monthly"))
```

Arguments

`x`	a `character` vector.
`from`	`character` scalar, indicating the period format (see `RTS` for supported formats).

See Also

Other utilities: `year_from_period()`

Examples

```
periods <- c("2018-Q4","2019-Q1")
period_to_int(periods, from="quarterly")
```

REP *Records expected are provided.*

Description

Records expected are provided.

Usage

```
REP(coverage = c("Min", "Max", "Only", "Excl"), keyTable, ...)
```

Arguments

`coverage`	character scalar indicating the type of coverage: • `Min`: The combinations represent the minimum coverage of the records to be provided (More combinations are acceptable) • `Max`: The combinations represent the maximum coverage of the records to be provided (Less combinations are acceptable) • `Only`: The combinations are represented in all the records to be provided (not less, not more combinations can be accepted) • `Excl`: The combinations should not be provided in records
`keyTable`	When used directly, a data frame containig (keys or key combinations) that must be present in the data. When used in a validation rule, the bare (unqouted) name of the data frame when passed as a reference data with `validate::confront` (see example).
`...`	When used in a validation rule, a comma separated list of bare (unquoted) column names. Otherwise a named, comma separated list of `character` vectors.

Value

A `logical` vector with length the number of records. It is `FALSE` for any record when the check fails on the coverage of the records provided: Expected codes or Periods for a specific field or combination of fields is present (check that no combination has been missed out)

References

Main types of validation rules for ESS data: REP

Examples

```
data(REPdat)

# Using REP in 'validate' (NOTE: keyTable = ref)
library(validate)
rule <- validator(REP(coverage="Only", keyTable=ref
   ,TABLE, FREQ, TIME_PERIOD, REPORTING, PARTNER, DIRECTION, AGE, ADJUST) == TRUE)
cf <- confront(REPdat, rule, ref = data.frame(ADJUST=c("S", "N")))
```

```
summary(cf)
as.data.frame(cf)

# Using REP directly (NOTE: named columns)
REP(coverage="Only",keyTable=data.frame(ADJUST=c("S", "N"))
    , TABLE       = REPdat$TABLE
    , FREQ        = REPdat$FREQ
    , TIME_PERIOD = REPdat$TIME_PERIOD
    , REPORTING   = REPdat$REPORTING
    , PARTNER     = REPdat$PARTNER
    , DIRECTION   = REPdat$DIRECTION
    , AGE         = REPdat$AGE
    , ADJUST      = REPdat$ADJUST )
```

`RNR` *Check number of records.*

Description

Check that the number of records in a file is higher or equal to a minimum and (optionally) is lower or equal to a maximum.

Usage

```
RNR(data, min = 1, max = NULL)

RNR2(data, min = 1, max = NULL)
```

Arguments

`data`	A bare (unquoted) '`.`' when used in a validation rule, otherwise a data frame.
`min`	nonnegative integer
`max`	nonnegative integer, not smaller then `min`, or `NULL`. If `max` is set to `NULL`, only the minimum number of records is checked.

Value

For `RNR` A `logical` vector of length `nrow(data)`. All `FALSE` when the number of records satisfies the bounds, otherwise all `TRUE`.

For `RNR2`, `TRUE` or `FALSE`.

Note

These checks can easily be performed directly with **validate** directly (see examples). `RNR` and `RNR2` are provided for consistency.

References

Main types of validation rules for ESS data: RNR

See Also

Other validation-functions: COC, COV(), FDL(), FDM(), FDT(), RTS(), RWD(), VAD(), VCO(), VIR(), VSA()

Examples

```
data(RNRdat)

# Using 'validate' directly
library(validate)
rules <- validator( nrow(.) >= 4 )
cf <- confront(RNRdat, rules)
summary(cf)
as.data.frame(cf)


# Using RNR directly
RNR(RNRdat, min=4)

# Using RNR with 'validate' (NOTE: data= . )
rule <- validator(RNR(data=., min=4) == TRUE)
cf <- confront(RNRdat, rule)
out <- as.data.frame(cf)
```

RTS *Check that records are present for time series*

Description

A record set is split by a set of *dimensions*. For each split, the variable indicating the time period must be both gapless and within bounds.

Usage

```
RTS(timevar, ftp, ltp, ...)
```

Arguments

`timevar`	When used in a validation rule, a bare (unquoted) name of the variable representing a time period (e.g. `TIME_PERIOD`). Otherwise a vector of class `character`. Coerced to character as necessary.
`ftp`	First time period in one of the supported notations (see Details)

`ltp`	Last time period in one of the supported notations (see Details)
`...`	Comma-separated list of bare (unquoted) dimensions. Time series must be gapless from `ftp` to `ltp` for each combination of values in these dimensions.

Details

The following notations for time periods are supported:

- `YYYY`: annual data, e.g. `"2016"`
- `YYYY-?QN`: quarterly data, e.g. `"2016Q1"` or `"2016-Q1"`
- `YYYYMNN`: monthly data, e.g. `"2016M01"`, `"2016M10"`

References

Main types of validation rules for ESS data: RTS

See Also

Other validation-functions: `COC`, `COV()`, `FDL()`, `FDM()`, `FDT()`, `RNR()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
# RTS examples
data(RTSdat)

# Example using RTS with 'validate'
library(validate)
rules <- validator(
 RTS(TIME_PERIOD, ftp = "2008", ltp = "2010"
   , TABLE, FREQ, REPORTING, PARTNER, DIRECTION, AGE, ADJUST) == TRUE
)
cf <- confront(RTSdat, rules)
summary(cf)
out <- as.data.frame(cf)

# Example using RTS directly
RTS(RTSdat$TIME_PERIOD, ftp = "2008", ltp = "2010"
   , RTSdat$TABLE, RTSdat$FREQ, RTSdat$REPORTING
   , RTSdat$PARTNER, RTSdat$DIRECTION, RTSdat$AGE, RTSdat$ADJUST)

# Or, using the 'with' function from base R
with(RTSdat
   , RTS( TIME_PERIOD, ftp = "2008", ltp = "2010"
        , TABLE, FREQ, REPORTING, PARTNER, DIRECTION, AGE, ADJUST)
)
```

RWD *Records are without duplicate ID keys.*

Description

Records are without duplicate ID keys.

Usage

```
RWD(...)

RWD2(...)
```

Arguments

`...`	When used in a validation rule, a comma separated list of bare (unquoted) column names. Otherwise a comma separated list of `character` vectors.

Value

For `RWD` a `logical` vector with lenght the number of records. It is `FALSE` for any record that is the duplicate of another record (with respect to the variables in the argument).

For `RDW2`: `TRUE` when there are no records that have duplicate values for the variables in the argument, else `FALSE`.

References

Main types of validation rules for ESS data: RWD

See Also

Other validation-functions: `COC`, `COV()`, `FDL()`, `FDM()`, `FDT()`, `RNR()`, `RTS()`, `VAD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
data(RWDdat)
library(validate)
# Using RWD with 'validate'
rules <- validator(
  RWD(TABLE, FREQ, TIME_PERIOD, REPORTING, PARTNER, DIRECTION, AGE, ADJUST) == TRUE )
cf <- confront(RWDdat, rules)
summary(cf)
as.data.frame(cf)

# Using RWD directly

RWD(RWDdat$TABLE, RWDdat$FREQ, RWDdat$TIME_PERIOD, RWDdat$REPORTING
, RWDdat$PARTNER, RWDdat$DIRECTION, RWDdat$AGE, RWDdat$ADJUST)
```

VAD — *Values for Aggregates are consistent with Details*

Description

Check that values for aggregates are consistent with the sum of values for detailed data.

Usage

```
VAD(data, field, aggregate_code, operator, tolerance, refdata)
```

Arguments

`data`	a data frame when called directly. When used in a validation rule, `data=.` to reference the data set passed to `confront`.
`field`	a character scalar containing the codes to be used for this check.
`aggregate_code`	a charactar scalar containing the code for the aggregate.
`operator`	a charactar scalar containing a relational Operator between aggregates and the sum of detailed data
`tolerance`	(acceptable margin) expressed in a absolute number
`refdata`	When called directly, a data frame. When used in a validation rule, the name of the reference variable passed to `confront`.

Value

A `logical` vector with length the number of records. Each element of this vector contains the result of the check

References

Main types of validation rules for ESS data: VAD

See Also

Other validation-functions: `COC`, `COV()`, `FDL()`, `FDM()`, `FDT()`, `RNR()`, `RTS()`, `RWD()`, `VCO()`, `VIR()`, `VSA()`

Examples

```
data(VADdat)
data(HT_AGE_GROUPSdat)

# Example in par 4.3.6 of ESTAT doc, using 'validate'
library(validate)
rule <- validator(VAD(data=., field='AGE', aggregate_code='TOTAL', operator='='
  , tolerance='0.01', ref=refdata) == TRUE)
cf <- confront(VADdat, rule, ref=list(refdata=HT_AGE_GROUPSdat))
summary(cf)
as.data.frame(cf)

# example using VAD directly

VAD(data=VADdat, field='AGE', aggregate_code='TOTAL', operator='='
  , tolerance='0.01', ref=HT_AGE_GROUPSdat)
```

VCO *Values are Consistent*

Description

Pivot a table in long format around a variable, execute a validation rule after pivoting, transform the results to the unpivoted table.

Usage

```
VCO(data, pivot, idvars, rule)
```

Arguments

`data`	A bare (unquoted) '`.`' when used in a validation rule, otherwise a data frame.
`pivot`	Pivoting variable: the values `VAL1`, `VAL2`, . . .,`VALN` of this variabe become columns after pivoting.
`idvars`	Identifying variables: these variables stay in rows after pivoting. All variables that are not pivot or identifying are considered observations. A variable named `OBS` will be split over columns `OBS.VAL1`, `OBS.VAL2`, ... When `VCO` is used in a validation rule with **validate**, `idvars` must be a literal character vector. That is, its value can not be a variable defined earlier in the script.
`rule`	A bare expression in terms of the `OBS.VARi` variables.

References

Main types of validation rules for ESS data: VCO

See Also

Other validation-functions: COC, COV(), FDL(), FDM(), FDT(), RNR(), RTS(), RWD(), VAD(), VIR(), VSA()

Examples

```
data(VCOdat)
# Example using VCO directly on a data set.
VCO(VCOdat, pivot = "AGE"
 , idvars = c("TABLE","FREQ", "TIME_PERIOD","REPORTING","PARTNER","DIRECTION")
 , rule = OBS_VALUE.Y0_18/OBS_VALUE.TOTAL<0.5)

# Example using VCO in a validation rule with the 'validate' package (NOTE: data = . )
library(validate)
rules <- validator(
  VCO( data = . , pivot = "AGE"
   , idvars = c("TABLE","FREQ", "TIME_PERIOD","REPORTING","PARTNER","DIRECTION")
   , rule = OBS_VALUE.Y0_18/OBS_VALUE.TOTAL<0.5) == TRUE)
cf <- confront(VCOdat, rules)
summary(cf)
as.data.frame(cf)
```

`VIR` *Check that values are within a range*

Description

Check that values are within a range

Usage

```
VIR(d, Min = NULL, Max = NULL)
```

Arguments

`d`	When used in a validation rule, a bare (unquoted) name of a variable. Otherwise a vector of class `character`. Coerced to character as necessary.
`Min`	smallest allowed value
`Max`	largest allowed value

Value

A `logical` with the length of `d`.

Note

Checking ranges can be doen with `VIR` but is even easier to do directly with **validate**. See the examples.

References

Main types of validation rules for ESS data: VIR

See Also

Other validation-functions: COC, COV(), FDL(), FDM(), FDT(), RNR(), RTS(), RWD(), VAD(), VCO(), VSA()

Examples

```
data(VIRdat)
library(validate)
# Using 'validate' directly:
rules <- validator(OBS_VALUE >= 1)
cf <- confront(VIRdat, rules)
summary(cf)
as.data.frame(cf)

# Using VIR directly
VIR(VIRdat$OBS_VALUE, Min = 1)

# Using VIR in a validation rule

rules <- validator(VIR(OBS_VALUE, Min=1) == TRUE)
cf <- confront(VIRdat, rules)
summary(cf)
as.data.frame(cf)
```

VSA *Check that the length of the data in the field is acceptable*

Description

Check that the length of the data in the field is acceptable

Usage

```
VSA(
  data,
  tolerance = 0.01,
  value = "OBS_VALUE",
  adjust = "ADJUST",
  nsa = "N",
  sa = "S",
  idvars
)
```

Arguments

`data`	When used in a validation rule, a `.` to reference the data set passed to `confront`. A data frame when called directly.
`tolerance`	allowed relative difference (unadjusted data is the reference).
`value`	character scalar: name of the variable with values (e.g. `OBS_VALUE`) column with observed values. Otherwise a numeric vector.
`adjust`	character scalar: name of the variable in `data` indicating whether data is seasonally adjusted or not.
`nsa`	A character scalar. The code used to indicate non-seasonally adjusted data.
`sa`	A character scalar. The code used to indicate seasonally adjusted data.
`idvars`	character vector with names of all dimensions (identifying variables), except for the time period and the adjustment status.

References

Main types of validation rules for ESS data: VSA

See Also

Other validation-functions: `COC`, `COV()`, `FDL()`, `FDM()`, `FDT()`, `RNR()`, `RTS()`, `RWD()`, `VAD()`, `VCO()`, `VIR()`

Examples

```
data(VSAdat)

# Using VSA directly
VSA(data= VSAdat
  , idvars = c("TABLE","FREQ","REPORTING","PARTNER","DIRECTION","AGE"))



# Using VSA with 'validate' (NOTE: data=.)
library(validate)
rules <- validator(
  VSA(data=.
    , idvars = c("TABLE","FREQ","REPORTING","PARTNER","DIRECTION","AGE")) == TRUE
)
cf <- confront(VSAdat, rules,raise="all")
summary(cf)
as.data.frame(cf)
```

`year_from_period` *Integer year from period string*

Description

Extracts first four characters of each element of `x` and converts to integer.

Usage

```
year_from_period(x)
```

Arguments

`x`	a `character` vector.

See Also

Other utilities: `period_to_int()`

Examples

```
periods <- c("2018-Q4","2019-Q1")
year_from_period(periods)
```

Index