Package: dataverifyr (via r-universe)

November 6, 2026

Type Package

Title A Lightweight, Flexible, and Fast Data Validation Package that Can Handle All Sizes of Data

Version 0.1.11

Description Allows you to define rules which can be used to verify a given dataset. The package acts as a thin wrapper around more powerful data packages such as 'dplyr', 'data.table', 'arrow', and 'DBI' ('SQL'), which do the heavy lifting.

License MIT + file LICENSE

URL https://github.com/DavZim/dataverifyr, https://davzim.github.io/dataverifyr/

BugReports https://github.com/DavZim/dataverifyr/issues

Imports yaml

Suggests arrow, data.table, DBI, dplyr, dbplyr, duckdb (>= 1.5.1), RSQLite, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://nmrgroup.r-universe.dev

Date/Publication 2026-04-10 18:04:42 UTC

RemoteUrl https://github.com/DavZim/dataverifyr

RemoteRef HEAD

RemoteSha 420f9c064c64c466f31556c399d5957b5d92dabc

Contents

bind_rules . . . 2
check_data . . . 3
data_column . . . 4
dataverifyr_plus . . . 5
describe . . . 6
detect_backend . . . 7
filter_fails . . . 8
plot_res . . . 9
reference_rule . . . 10
rule . . . 11
ruleset . . . 12
sample_data . . . 14
write_rules . . . 14

Index **16**

`bind_rules` *Programatically Combine a List of Rules and Rulesets into a Single Ruleset*

Description

Programatically Combine a List of Rules and Rulesets into a Single Ruleset

Usage

```
bind_rules(rule_ruleset_list)
```

Arguments

`rule_ruleset_list`
: a list of rules and rulesets you whish to combine into a single list

Value

a ruleset which consolidates all the inputs

check_data *Checks if a dataset confirms to a given set of rules*

Description

Checks if a dataset confirms to a given set of rules

Usage

```
check_data(
  x,
  rules,
  xname = deparse(substitute(x)),
  stop_on_fail = FALSE,
  stop_on_warn = FALSE,
  stop_on_error = FALSE,
  stop_on_schema_fail = FALSE,
  extra_columns = c("ignore", "warn", "fail")
)
```

Arguments

x	a dataset, either a data.frame, dplyr::tibble, data.table::data.table, arrow::arrow_table, arrow::open_dataset, or dplyr::tbl (SQL connection). Can also be a named list of datasets when using reference rules.
rules	a list of rules
xname	optional, a name for the x variable (only used for errors)
stop_on_fail	when any of the rules fail, throw an error with stop
stop_on_warn	when a warning is found in the code execution, throw an error with stop
stop_on_error	when an error is found in the code execution, throw an error with stop
stop_on_schema_fail	when any schema checks fail, throw an error with stop
extra_columns	how to treat columns in x that are not declared in optional data_columns attached to a ruleset. One of "ignore" (default), "warn", or "fail".

Value

a data.frame-like object with one row for each rule and its results

See Also

detect_backend()

Examples

```
rs <- ruleset(
  rule(mpg > 10),
  rule(cyl %in% c(4, 6)), # missing 8
  rule(qsec >= 14.5 & qsec <= 22.9)
)
rs

check_data(mtcars, rs)

# schema + relation checks in one output
orders <- data.frame(order_id = 1:3, customer_id = c(10, 99, NA), amount = c(10, -5, 20))
customers <- data.frame(customer_id = c(10, 11))

rs2 <- ruleset(
  rule(amount >= 0, name = "amount non-negative"),
  reference_rule(
    local_col = "customer_id",
    ref_dataset = "customers",
    ref_col = "customer_id",
    allow_na = TRUE
  ),
  data_columns = list(
    data_column("order_id", type = "int", optional = FALSE),
    data_column("customer_id", type = "double", optional = FALSE),
    data_column("amount", type = "double", optional = FALSE)
  ),
  data_name = "orders"
)

check_data(list(orders = orders, customers = customers), rs2)
```

`data_column` *Define a Column Specification for Schema Checks*

Description

Creates a single column declaration used in `ruleset(..., data_columns = ...)`. Column declarations are schema checks (column existence, optionality, and declared type), whereas `rule()` is for row-wise value checks.

Usage

```
data_column(
  col,
  type = NA_character_,
  optional = FALSE,
  description = NA_character_
)
```

Arguments

col	column name.
type	optional declared type (for example "int", "double", "str", "logical"). Use NA_character_ for no type declaration.
optional	logical; if FALSE, the column is required.
description	optional free-text description.

Value

A data_column object (list) that can be passed in ruleset(..., data_columns = list(...)).

Examples

```
rs <- ruleset(
  rule(price >= 0),
  data_columns = list(
    data_column("price", type = "double", optional = FALSE),
    data_column("note", type = "str", optional = TRUE)
  )
)
rs

# combined with row rules and strict schema stopping
order_rules <- ruleset(
  rule(price >= 0, allow_na = FALSE),
  data_columns = list(
    data_column("order_id", type = "int", optional = FALSE),
    data_column("price", type = "double", optional = FALSE),
    data_column("note", type = "str", optional = TRUE)
  )
)

check_data(
  data.frame(order_id = 1:3, price = c(10, 20, 30), note = c("ok", NA, "ok")),
  order_rules,
  stop_on_schema_fail = TRUE
)
```

---

dataverifyr_plus *Add Rules and Rulesets Together*

Description

- allows you to add rules and rulesets into larger rulesets. This can be useful if you want to create a ruleset for a dataset out of checks for other datasets.

Usage

```
datavarifyr_plus(a, b)

## S3 method for class 'ruleset'
a + b

## S3 method for class 'rule'
a + b
```

Arguments

a	the first ruleset you wish to add
b	the second ruleset you wish to add

---

describe *Describes a dataset*

Description

Note that the current version is in the beta stadium at best, that means the R-native formats (data.frame, dplyr/tibble, or data.table) are a lot faster than arrow or SQL-based datasets.

Usage

```
describe(x, skip_ones = TRUE, digits = 4, top_n = 3, fast = FALSE)
```

Arguments

x	a dataset, either a data.frame, dplyr::tibble, data.table::data.table, arrow::arrow_table, arrow::open_dataset, or dplyr::tbl (SQL connection)
skip_ones	logical, whether values that occur exactly once should be omitted from most_frequent
digits	integer, number of digits to round numeric values in most_frequent
top_n	integer, number of most frequent values to include in most_frequent; set to 0 to skip the most_frequent computation
fast	logical, when TRUE skip expensive fields (n_distinct, median) by returning NA for them

Details

Numeric values in most_frequent are rounded to digits (default: 4). If a variable has at most 1 distinct value, most_frequent is left empty. By default, values with count 1 are omitted from most_frequent.

Value

a `data.frame`, `dplyr::tibble`, or `data.table::data.table` containing a summary of the dataset given

See Also

Similar to skimr::skim(), summarytools::dfSummary(), and gtExtras::gt_plt_summary()

Examples

```
describe(mtcars)
```

`detect_backend` *Detects the backend which will be used for checking the rules*

Description

The detection will be made based on the class of the object as well as the packages installed. For example, if a `data.frame` is used, it will look if `data.table` or `dplyr` are installed on the system, as they provide more speed. Note the main functions will revert the

Usage

```
detect_backend(x)
```

Arguments

`x`	The data object, ie a data.frame, tibble, data.table, arrow, or DBI object

Value

a single character element with the name of the backend to use. One of `base-r`, `data.table`, `dplyr`, `collectibles` (for arrow or DBI objects)

See Also

`check_data()`

Examples

```
data <- mtcars
detect_backend(data)
```

`filter_fails` *Filters a result dataset for the values that failed the verification*

Description

Filters a result dataset for the values that failed the verification

Usage

```
filter_fails(res, x, per_rule = FALSE)
```

Arguments

`res`	a result data.frame as outputted from `check_data()` or a ruleset
`x`	a dataset that was used in `check_data()`
`per_rule`	if set to TRUE, a list of filtered data is returned, one for each failed verification rule. If set to FALSE, a data.frame is returned of the values that fail any rule.

Value

the dataset with the entries that did not match the given rules

Examples

```
rules <- ruleset(
  rule(mpg > 10 & mpg < 30), # mpg goes up to 34
  rule(cyl %in% c(4, 8)), # missing 6 cyl
  rule(vs %in% c(0, 1), allow_na = TRUE)
)

res <- check_data(mtcars, rules)

filter_fails(res, mtcars)
filter_fails(res, mtcars, per_rule = TRUE)

# alternatively, the first argument can also be a ruleset
filter_fails(rules, mtcars)
filter_fails(rules, mtcars, per_rule = TRUE)
```

plot_res *Visualize the results of a data validation*

Description

Visualize the results of a data validation

Usage

```
plot_res(
  res,
  main = "Verification Results per Rule",
  colors = c(pass = "#308344", fail = "#E66820"),
  labels = TRUE,
  table = TRUE
)
```

Arguments

res	a data.frame as returned by check_data()
main	the title of the plot
colors	a named list of colors, with the names pass and fail
labels	whether the values should be displayed on the barplot
table	show a table in the legend with the values

Value

a base r plot

Examples

```
rs <- ruleset(
  rule(Ozone > 0 & Ozone < 120, allow_na = TRUE), # some mising values and > 120
  rule(Solar.R > 0, allow_na = TRUE),
  rule(Solar.R < 200, allow_na = TRUE),
  rule(Wind > 10),
  rule(Temp < 100)
)

res <- check_data(airquality, rs)
plot_res(res)
```

`reference_rule` *Define a Relational Reference Rule*

Description

Creates a rule that checks whether values in a local column exist in a column of a referenced dataset. Use with `check_data()` by supplying `x` as a named list of datasets and setting `data_name` in `ruleset()` (or by ordering the list so the first entry is the primary dataset).

Usage

```
reference_rule(
  local_col,
  ref_dataset,
  ref_col,
  name = NA,
  allow_na = FALSE,
  negate = FALSE,
  ...
)
```

Arguments

`local_col`	column name in the primary dataset.
`ref_dataset`	name of the referenced dataset in the `x` list.
`ref_col`	column name in the referenced dataset.
`name`	optional display name for the rule.
`allow_na`	logical; if `TRUE`, missing values in `local_col` are treated as passing.
`negate`	logical; if `TRUE`, inverts the rule (values must *not* be in the referenced column).
`...`	additional fields attached to the rule object.

Value

A `reference_rule` object that can be included in `ruleset()`.

Examples

```
flights <- data.frame(carrier = c("AA", "BB", NA_character_))
carriers <- data.frame(carrier_id = c("AA"))

rs <- ruleset(
  reference_rule(
    local_col = "carrier",
    ref_dataset = "carriers",
    ref_col = "carrier_id",
    allow_na = TRUE
```

```
  ),
  data_name = "flights"
)

check_data(list(flights = flights, carriers = carriers), rs)

# negated relation: value must NOT exist in blacklist
blacklist <- data.frame(carrier_id = c("XX", "YY"))
rs_neg <- ruleset(
  reference_rule(
    local_col = "carrier",
    ref_dataset = "blacklist",
    ref_col = "carrier_id",
    negate = TRUE,
    allow_na = TRUE
  ),
  data_name = "flights"
)

check_data(list(flights = flights, blacklist = blacklist), rs_neg)
```

`rule` *Creates a single data rule*

Description

Creates a single data rule

Usage

```
rule(expr, name = NA, allow_na = FALSE, negate = FALSE, ...)

## S3 method for class 'rule'
print(x, ...)
```

Arguments

`expr`	an expression which dictates which determines when a rule is good. Note that the expression is evaluated in `check_data()`, within the given framework. That means, for example if a the data given to `check_data()` is an `arrow` dataset, the expression must be mappable from `arrow` (see also arrow documentation). The expression can be given as a string as well.
`name`	an optional name for the rule for reference
`allow_na`	does the rule allow for NA values in the data? default value is FALSE. Note that when NAs are introduced in the expression, `allow_na` has no effect. Eg when the rule `as.numeric(vs) %in% c(0, 1)` finds the values of `vs` as `c("1", "A")`, the rule will throw a fail regardless of the value of `allow_na` as the NA is introduced in the expression and is not found in the original data. However, when the values of `vs` are `c("1", NA)`, `allow_na` will have an effect.

negate	is the rule negated, only applies to the expression not allow_na, that is, if expr = mpg > 10, allow_na = TRUE, and negate = TRUE, it would match all mpg <= 10 as well as NAs.
...	additional arguments that are carried along for your documentation, but are not used. Could be for example date, person, contact, comment, etc
x	a rule to print

Value

The rule values as a list

Methods (by generic)

- print(rule): Prints a rule

Examples

```
r <- rule(mpg > 10)
r

r2 <- rule(mpg > 10, name = "check that mpg is reasonable", allow_na = TRUE,
           negate = FALSE, author = "me", date = Sys.Date())
r2

check_data(mtcars, r)

rs <- ruleset(
  rule(mpg > 10),
  rule(cyl %in% c(4, 6)), # missing 8
  rule(qsec >= 14.5 & qsec <= 22.9)
)
rs
check_data(mtcars, rs)
```

---

ruleset *Creates a set of rules*

Description

Creates a set of rules

Usage

```
ruleset(..., data_columns = NULL, meta = NULL, data_name = NULL)

## S3 method for class 'ruleset'
print(x, n = 3, ...)
```

Arguments

...	a list of rules
data_columns	optional list of schema declarations created with internal data_column() helper.
meta	optional metadata list for v1 YAML workflows.
data_name	optional name of the primary dataset when check_data() receives a named list of datasets.
x	a ruleset to print
n	a maximum number of rules to print

Value

the list of rules as a ruleset

Methods (by generic)

- print(ruleset): Prints a ruleset

Examples

```
r1 <- rule(mpg > 10)
r2 <- rule(mpg < 20)
rs <- ruleset(r1, r2)
rs

rs <- ruleset(
  rule(cyl %in% c(4, 6, 8)),
  rule(is.numeric(disp))
)
rs

# combine row, schema, and relational checks
orders <- data.frame(order_id = 1:4, customer_id = c(10, 11, 99, NA), amount = c(10, 20, -5, 30))
customers <- data.frame(customer_id = c(10, 11, 12))

rs2 <- ruleset(
  rule(amount >= 0, name = "amount must be non-negative"),
  reference_rule(
    local_col = "customer_id",
    ref_dataset = "customers",
    ref_col = "customer_id",
    allow_na = TRUE
  ),
  data_columns = list(
    data_column("order_id", type = "int", optional = FALSE),
    data_column("customer_id", type = "int", optional = FALSE),
    data_column("amount", type = "double", optional = FALSE)
  ),
  data_name = "orders"
)
```

```
check_data(list(orders = orders, customers = customers), rs2)
```

`sample_data` *Sample Orders Dataset for Examples and Tests*

Description

A small, human-readable dataset with mixed column types, missing values, and one datetime column. It is designed for documentation examples and unit tests.

Usage

```
sample_data
```

Format

A data frame with 8 rows and 6 variables:

order_id Integer order identifier.

customer_tier Character tier (`"bronze"`, `"silver"`, `"gold"`, etc), includes one `NA`.

amount Numeric order amount, includes one negative value and one `NA`.

paid Logical payment flag, includes one `NA`.

payment_method Character payment method, includes one `NA`.

order_time `POSIXct` order timestamp in UTC, includes one `NA`.

Examples

```
sample_data
```

`write_rules` *Read and write rules to a yaml file*

Description

Read and write rules to a yaml file

Usage

```
write_rules(x, file, format = c("v1", "pre_v1"))

read_rules(file)
```

Arguments

x	a list of rules
file	a filename
format	output format. "v1" writes structured YAML with meta, data-columns, and data-rules. "pre_v1" keeps the pre package version 1.0 flat-list structure.

Value

the filename invisibly

Functions

- read_rules(): reads a ruleset back in

Examples

```
rr <- ruleset(
  rule(mpg > 10),
  rule(cyl %in% c(4, 6, 8))
)
file <- tempfile(fileext = ".yml")
write_rules(rr, file)
```

Index

∗ **datasets**
sample_data, 14
+.rule *(*dataverifyr_plus*)*, 5
+.ruleset *(*dataverifyr_plus*)*, 5

arrow::arrow_table, *3*, *6*
arrow::open_dataset, *3*, *6*

bind_rules, 2

check_data, 3
check_data(), *7–9*

data.frame, *3*, *6*
data.table::data.table, *3*, *6*
data_column, 4
datavarifyr_plus *(*dataverifyr_plus*)*, 5
dataverifyr_plus, 5
describe, 6
detect_backend, 7
detect_backend(), *3*
dplyr::tbl, *3*, *6*
dplyr::tibble, *3*, *6*

filter_fails, 8

plot_res, 9
print.rule *(*rule*)*, 11
print.ruleset *(*ruleset*)*, 12

read_rules *(*write_rules*)*, 14
reference_rule, 10
rule, *3*, 11
rule(), *4*
ruleset, 12

sample_data, 14

write_rules, 14