Package: schmear (via r-universe)

May 29, 2026

Title Build Structured Data Frame Subtypes

Version 0.1.0

Description Provides developer-focused helper functions and S3 classes to ease the creation of structured subtypes of data frames. Developers can require certain columns and types to be present, and can enforce crossing and nesting relationships between values in different columns. Type-specific metadata and attributes are preserved through common data frame manipulations.

Imports rlang, cli, vctrs

Suggests dplyr, pillar, testthat (>= 3.0.0)

License MIT + file LICENSE

Encoding UTF-8

URL https://corymccartan.com/schmear/, https://github.com/CoryMcCartan/schmear

BugReports https://github.com/CoryMcCartan/schmear/issues

Config/testthat/edition 3

Config/roxygen/version 8.0.0

Config/roxygen/markdown TRUE

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Cory McCartan [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-6251-669X>)

Maintainer Cory McCartan <mccartan@psu.edu>

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

Date/Publication 2026-05-29 12:31:25 UTC

RemoteUrl https://github.com/cran/schmear

RemoteRef HEAD

RemoteSha 3166ea5831f7e2db686ff6c06799116bdd02b5c4

Contents

`dplyr-sch_df` *dplyr integration for sch_df*

Description

These methods hook into dplyr's three extension generics ([dplyr::dplyr_row_slice()], [dplyr::dplyr_col_modify()], [dplyr::dplyr_reconstruct()]) plus the base-R [names()] replacement and 1-d '[' to enforce schema constraints across dplyr operations with near-zero overhead.

Details

Each method calls 'NextMethod()' first (performing the actual dplyr operation) and then re-validates the result against the schema, running only the checks that can plausibly be violated by that type of operation:

| Operation | Checks run | |—|—| | 'dplyr_row_slice' | none by default; '"relationships"' behind a flag | | 'dplyr_col_modify' | '"names"', '"types"', '"distinct"' | | 'dplyr_reconstruct' | '"names"', '"types"' | | '[' (1-d column subsetting) | '"names"' | | 'names<-' | errors if any *schema* column name is changed |

Row slicing ('arrange', 'filter', 'slice', semi/anti joins)

Row operations cannot introduce new name or type violations, so no validation is run by default. Relationship constraints (crossing completeness, primary-key uniqueness) *can* be broken by removing rows. Pass '.check_relationships = TRUE' in '...' to opt in to that check.

Column modification ('mutate')

A column modification can delete required columns (via 'NULL' assignment), assign the wrong type, or introduce duplicate values into a 'distinct = TRUE' column, so all three cheap checks are run.

Reconstruction (joins)

'dplyr_reconstruct()' is called after joins. Only names and types are checked: distinct and relationship checks are omitted because joins can intentionally produce non-distinct rows or incomplete crossings.

Column subsetting (‘select‘, ‘relocate‘)

A 1-d ‘[‘ call selects or reorders columns. Reordering is always safe; but selecting a column subset could drop required columns, so a names check is run.

Renaming (‘rename‘, ‘rename_with‘, ‘select‘ with rename)

Renaming a column that belongs to the schema (either directly or as a member of a [sch_multiple()] group) is never valid without also updating the schema definition, so an error is raised immediately.

`sch_coerce` *Coerce a data frame to conform to a schema*

Description

Attempts to coerce each column of ‘data‘ to the type expected by ‘schema‘, using the coercion method defined for each column type. After coercion, optionally validates the result with [sch_validate()].

Usage

```
sch_coerce(schema, data, validate = TRUE, call = rlang::caller_env())
```

Arguments

`schema`	A schema object created by [sch_schema()].
`data`	A data frame to coerce.
`validate`	If ‘TRUE‘ (default), [sch_validate()] is called on the coerced data after all columns have been processed. Set to ‘FALSE‘ to skip validation, which can be useful when you want to inspect the coerced result before checking constraints.
`call`	The environment or call used for error reporting, passed to [rlang::abort()]. Useful when wrapping ‘sch_coerce()‘ inside another function so that errors point to the right place.

Details

Coercion is applied column-by-column using the ‘coerce‘ function registered for each type in the internal ‘type_fns‘ registry. For example, a column specified as ‘sch_integer()‘ will be coerced with [as.integer()]. Nested schemas (created with [sch_nest()]) are handled by recursing into each element data frame, and grouped columns (created with [sch_multiple()]) have each member column coerced individually.

Columns present in ‘data‘ but not named in ‘schema‘ (i.e., those covered by [sch_others()]) are left untouched.

Value

'data' with columns coerced to their schema types, invisibly, if coercion succeeds. If any column cannot be coerced, an error of class 'sch_coercion_error' is raised with a summary of all failures. When 'validate = TRUE', a subsequent [sch_validate()] call may also raise a 'sch_validation_error' if the coerced data still violates schema constraints (e.g., out-of-bounds values or uniqueness violations).

Examples

```
schema <- sch_schema(
    id = sch_integer(distinct = TRUE),
    name = sch_character(missing = FALSE),
    score = sch_numeric()
)

# Coerce a data frame with character columns
df <- data.frame(id = c("1", "2", "3"), name = c("Alice", "Bob", "Carol"), score = 1:3)
str(sch_coerce(schema, df))

# Nested schema coercion
nested_schema <- sch_schema(
    group = sch_factor(),
    info = sch_nest(x = sch_numeric(), y = sch_integer())
)
nested_df <- data.frame(group = "A")
nested_df$info <- list(data.frame(x = "1.5", y = "2"))
str(sch_coerce(nested_schema, nested_df))
```

---

sch_df *Construct and validate a schema-aware data frame*

Description

Bare-bones constructor for a data frame with an attached schema. This function should be called by package developers writing their own internal constructors. The only checks are for the types of 'data' and 'schema'. The 'validate_sch_df()' is a lightweight wrapper around [sch_validate()] that also returns the input.

Usage

```
new_sch_df(data, schema, groups = NULL, class = NULL, use_tbl = TRUE)

validate_sch_df(x)
```

Arguments

data	A data frame.
schema	A 'sch_schema' object.

groups	A named list of character vectors of column names, for use with [sch_multiple()].
class	Additional classes to add to the object, in addition to "sch_df" and tibble/data.frame classes.
use_tbl	If ‘TRUE‘, the returned object will have tibble classes ‘tbl_df‘ and ‘tbl‘ in addition to ‘data.frame‘. As a reminder, ‘tbl_df‘ affects the behavior of the object (slicing, row names, etc.), while ‘tbl‘ affects printing only.
x	A ‘sch_df‘ object.

Value

A data frame with class ‘c(class, "sch_df", ...)‘ and attributes ‘sch_schema‘ and ‘sch_groups‘.

The input, if validation is successful.

Examples

```
schema = sch_schema(
    .desc = "MCMC draws",
    .relationships = ~ chain * draw * parameter,
    chain = sch_integer("Chain number"),
    draw = sch_integer("Draw number", bounds = c(1, Inf), closed = c(TRUE, FALSE)),
    parameter = sch_character("Parameter name"),
    value = sch_numeric("Parameter value")
)
d_raw = data.frame(chain = 1L, draw = 1:4, parameter = "mu", value = rnorm(4))
d = new_sch_df(d_raw, schema, class="mcmc_draws")
validate_sch_df(d)
str(d)
```

---

sch_schema *Define a structured data type*

Description

Defines the structure of a single 'observation' for a structured data frame. Each column has type restrictions and may be required or optional. Schemas support nesting relationships.

Usage

```
sch_schema(..., .desc = NULL, .relationships = NULL)

sch_others()

sch_any(desc = NULL, missing = TRUE, required = TRUE, distinct = FALSE)

sch_multiple(
  name,
  type,
```

```
  desc = NULL,
  required = TRUE,
  check = NULL,
  msg = NULL,
  coerce = NULL
)

sch_nest(..., .desc = NULL)

sch_numeric(
  desc = NULL,
  bounds = c(-Inf, Inf),
  closed = c(TRUE, TRUE),
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)

sch_integer(
  desc = NULL,
  bounds = c(-Inf, Inf),
  closed = c(TRUE, TRUE),
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)

sch_logical(desc = NULL, missing = TRUE, required = TRUE, distinct = FALSE)

sch_character(desc = NULL, missing = TRUE, required = TRUE, distinct = FALSE)

sch_factor(
  desc = NULL,
  levels = NULL,
  strict = TRUE,
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)

sch_date(
  desc = NULL,
  bounds = c(as.Date(-Inf), as.Date(Inf)),
  closed = c(FALSE, FALSE),
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)
```

```
sch_datetime(
  desc = NULL,
  bounds = c(as.POSIXct(-Inf), as.POSIXct(Inf)),
  closed = c(FALSE, FALSE),
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)

sch_inherits(
  desc = NULL,
  class,
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)

sch_list_of(
  desc = NULL,
  class,
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)

sch_custom(
  name,
  desc = NULL,
  check,
  msg,
  coerce,
  ...,
  missing = TRUE,
  required = TRUE,
  distinct = FALSE
)
```

Arguments

...	Column specifications, in the form of 'col_name = col_type' pairs, where 'col_type' is a call to a column type constructor listed here, such as 'sch_numeric()'. Every type must be a kind of vector, i.e., [vctrs::obj_is_vector()] must return 'TRUE'. All columns must be named, except for 'sch_others()', as described below, and 'sch_multiple()', which describes a group of columns sharing the same type. A named 'sch_nest()' describes columns stored as a nested data frame. The special function 'sch_others()' indicates the preferred location of other columns not explicitly mentioned in the schema. If no 'sch_others()' appears, then other

columns are not allowed. Trailing commas are permitted.

`.relationships` An optional one-sided formula describing the structural relationships between values in different columns. Formulas can only involve named arguments to '...'. Use '*' to signify crossed levels, which will verify all combinations exist, '/' to signify nested levels, and '+' to create compound keys (bundling columns into a single identifier). See the examples below.

`desc, .desc` A description of the column for consumers of the schema. The type contraints will be described separately and do not need to be included in the description. For example for "age", the description might be "Age of the patient in years", not "Non-negative integer representing the age of the patient in years". For the overall 'sch_schema', the 'desc' will be printed as part of the header for data frames implementing the schema, by default.

`missing` If 'TRUE', the column may be contain missing values. Otherwise, any missing values result in an error.

`required` If 'TRUE' (default), the group entry in 'sch_groups' must contain at least one column name. If 'FALSE', an empty character vector for that entry is also accepted.

`distinct` If 'TRUE', the column must contain no duplicate values (after accounting for nesting structure).

`name` A name for the custom type.

`type` A column type constructor (e.g. [sch_numeric()]) specifying the expected type of every column in the group.

`check` A two-argument function that checks whether an object satisfies the type. The first argument is the object to check, and the second is the full type specification.

`msg` A one-argument function that generates a descriptive message about the type when passed the type object itself. Should not end with a period.

`coerce` A two-argument function that attempts to coerce an object to the type. The first argument is the object to coerce, and the second is the full type specification.

`bounds` Length-two vector 'c(min, max)' specifying the allowed range of values.

`closed` Length-two logical vector specifying whether the bounds are closed (inclusive) or open (exclusive).

`levels` A character vector of factor levels, or NULL not enforce specific levels.

`strict` If 'TRUE', only factors with the specified levels are accepted. If 'FALSE', character vectors with the specified levels are also accepted.

`class` A character vector of class names.

Value

An object of class 'sch_schema',

Functions

- `sch_others()`: A placeholder for other non-required columns in a schema.
- `sch_any()`: A column of any type. No type checking is performed.

- `sch_multiple()`: A group of multiple columns sharing the same type. The group is identified by ‘name‘, which must appear as an entry in the ‘sch_groups‘ attribute of the data frame being validated. That entry is a character vector of column names that belong to this group.

  Optionally accepts cross-column ‘check‘, ‘msg‘, and ‘coerce‘ functions that are applied to the entire group after per-column type checks pass. These must all be provided together or not at all.

  ‘sch_multiple()‘ must be unnamed in an ‘sch_schema()‘ call. Per-column constraints such as ‘missing‘ and ‘distinct‘ are set on the inner ‘type‘ argument.
- `sch_nest()`: A set of columns stored as a nested list-column of data frames. Must be given a name in the outer ‘sch_schema()‘.
- `sch_numeric()`: A numeric vector that is optionally constrained to be within a certain range.
- `sch_integer()`: An integer vector that is optionally constrained to be within a certain range.
- `sch_logical()`: A logical vector.
- `sch_character()`: A character vector.
- `sch_factor()`: A factor with specified levels.
- `sch_date()`: A Date vector that is optionally constrained to be within a certain range.
- `sch_datetime()`: A POSIXct vector that is optionally constrained to be within a certain range.
- `sch_inherits()`: A list-column whose elements satisfy ‘inherits(_, class)‘.
- `sch_list_of()`: A vector satisfying ‘inherits(_, class)‘.
- `sch_custom()`: A custom type defined by user-provided check, type message, and coercion functions. Additional named values to be stored along with the type specification may be passed via ‘...‘ and will be available to the check, message, and coercion function as elements of the ‘type‘ argument.

Worked examples

Three larger end-to-end example scripts are bundled with the package and show how to build a schema, attach group metadata, validate a compliant data frame, and exercise a range of corruption cases. List them with:

“‘r list.files(system.file("examples", package = "schmear"), full.names = TRUE) “‘

The included scripts are ‘mcmc_draws.R‘ (fully-crossed MCMC posterior draws), ‘ei_spec.R‘ (an ecological-inference specification using [sch_multiple()] groups with a row-sum cross-column check), and ‘election_data.R‘ (tidy election returns with a compound primary key and nested/crossed relationship structure).

Examples

```
sch_schema(
    .desc = "MCMC draws",
    .relationships = ~ chain * draw * parameter,
    chain = sch_integer("Chain number"),
    draw = sch_integer("Draw number", bounds = c(1, Inf), closed = c(TRUE, FALSE)),
    parameter = sch_factor("Parameter name", levels = c("mu", "sigma", "log_lik")),
    value = sch_numeric("Parameter value")
```

```
)

sch_schema(
    .desc = "Student data",
    .relationships = ~ (grade + teacher) / table_group,
    birthday = sch_date("Date of birth", required = FALSE),
    height = sch_numeric(
        "Height in inches",
        bounds = c(0, 108),
        closed = c(FALSE, TRUE)
    ),
    grade = sch_factor(strict = FALSE, levels = c("Kindergarten", "1st", "2nd")),
    teacher = sch_nest(
        first = sch_character("First name"),
        last = sch_character("Last name")
    ),
    table_group = sch_integer(bounds=c(1, 6)),
    enrolled = sch_logical(missing = FALSE),
    sch_others()
)

sch_schema(
    .desc = "Causal inference data",
    treatment = sch_factor(levels = c("control", "treatment"), missing = FALSE),
    outcome = sch_numeric(missing = FALSE),
    sch_multiple("covariates", type = sch_any(missing = FALSE), required = FALSE),
    sch_others()
)

sch_custom(
   name = "even",
   check = function(x, type) is.integer(x) && all(x %% 2 == 0),
   msg = function(type) "vector of even integers",
   coerce = function(x, type) (as.integer(x) %/% 2) * 2
)
```

`sch_validate` *Validate a data frame against a schema*

Description

Checks that a data frame conforms to a schema, validating column presence, types, missing values, uniqueness constraints, and nesting structure.

Usage

```
sch_validate(
  schema,
  data,
  check = c("names", "types", "distinct", "relationships"),
```

```
  call = rlang::caller_env()
)
```

Arguments

schema	A schema object created by [sch_schema()].
data	A data frame to validate.
check	A character vector specifying which checks to perform. The default runs all checks. Possible values: - ‘"names"‘: check for missing required columns and unexpected extra columns. - ‘"types"‘: check column types and missing-value (‘NA‘) constraints. - ‘"distinct"‘: check uniqueness constraints for columns marked ‘distinct = TRUE‘. Relatively expensive. - ‘"relationships"‘: validate relationship formulas (primary-key uniqueness and crossing/nesting completeness). Relatively expensive.
call	The environment or call used for error reporting, passed to [rlang::abort()]. Useful when wrapping ‘sch_validate()‘ inside another function so that the error points to the right place.

Value

‘data‘, invisibly, if validation succeeds. Otherwise, an error of class ‘sch_validation_error‘ is raised with a formatted summary of all issues found.

Examples

```
# Basic validation: valid data passes silently
schema <- sch_schema(
    id = sch_integer(distinct = TRUE),
    name = sch_character(missing = FALSE),
    age = sch_numeric(required = FALSE)
)
df <- data.frame(id = 1:3, name = c("Alice", "Bob", "Carol"), age = c(25, NA, 30))
sch_validate(schema, df)

# Invalid data throws validation errors; wrap in try() so examples can run
# missing required columns
try(sch_validate(schema, data.frame(id = 1:2)))

# type constraints not satisfied
try(sch_validate(schema, data.frame(id = c(1L, 1L), name = c("Alice", NA))))
```

Index