Package ‘ruler’

March 2, 2020

Title Tidy Data Validation Reports

Version 0.2.2

Description Tools for creating data validation pipelines and tidy reports. This package offers a framework for exploring and validating data frame like objects using 'dplyr' grammar of data manipulation.

License MIT + file LICENSE


BugReports https://github.com/echasnovski/ruler/issues

Depends R (>= 3.4.0)

Imports dplyr (>= 0.7.0), keyholder, rlang, tibble, tidyr (>= 0.7.0)

Suggests covr, knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

NeedsCompilation no

Author Evgeni Chasnovski [aut, cre]

Maintainer Evgeni Chasnovski <evgeni.chasnovski@gmail.com>

Repository CRAN

Date/Publication 2020-03-02 21:20:03 UTC

R topics documented:

ruler-package .................................................. 2
act_after_exposure ........................................... 3
any_breaker .................................................... 4
assert_any_breaker ........................................... 4
ruler-package

Description

ruler offers a set of tools for creating tidy data validation reports using dplyr grammar of data manipulation. It is designed to be flexible and extendable in terms of creating rules and using their output.

Details

The common workflow is:

- Define dplyr-style packs of rules for basic data units (data, group, column, row, cell) to obey.
- Expose some data to those rules. The result is the same data with possibly created exposure attribute. Exposure contains information about applied packs and tidy data validation report.
- Use data and exposure to perform some actions: assert about rule breakers, impute data, remove outliers and so on.

To learn more about ruler browse vignettes with browseVignettes(package = "ruler"). The preferred order is:

1. Design process and exposure format.
2. Rule packs.
3. Validation

Author(s)

Maintainer: Evgeni Chasnovski <evgeni.chasnovski@gmail.com>
**act_after_exposure**

**See Also**

Useful links:
- [https://echasnovski.github.io/ruler/](https://echasnovski.github.io/ruler/)
- [https://github.com/echasnovski/ruler](https://github.com/echasnovski/ruler)
- Report bugs at [https://github.com/echasnovski/ruler/issues](https://github.com/echasnovski/ruler/issues)

---

**act_after_exposure  Act after exposure**

**Description**

A wrapper for consistent application of some actions based on the data after exposure.

**Usage**

```r
act_after_exposure(.tbl, .trigger, .actor)
```

**Arguments**

- `.tbl` Result of `exposure`, i.e. data frame with `exposure` attribute.
- `.trigger` Function which takes `.tbl` as argument and returns `TRUE` if some action needs to be performed.
- `.actor` Function which takes `.tbl` as argument and performs the action.

**Details**

Basically `act_after_exposure()` is doing the following:

- Check that `.tbl` has a proper `exposure` attribute.
- Compute whether to perform intended action by computing `.trigger(.tbl)`.
- If trigger results in `TRUE` then `.actor(.tbl)` is returned. In other case `.tbl` is returned.

It is a good idea that `.actor` should be doing one of two things:

- Making side effects. For example throwing an error (if condition in `.trigger` is met), printing some information and so on. In this case it should return `.tbl` to be used properly inside a pipe.
- Changing `.tbl` based on exposure information. In this case it should return the imputed version of `.tbl`.

**See Also**

`any_breaker` for trigger which returns `TRUE` in case any rule breaker is found in exposure.
`assert_any_breaker` for usage of `act_after_exposure()` in building data validation pipelines.
Examples

```r
exposure_printer <- function(.tbl) {print(get_exposure(.tbl)); .tbl}
mtcars_exposed <- mtcars %>%
  expose(data_packs(. %>% dplyr::summarise(nrow_low = nrow(.) > 50))) %>%
  act_after_exposure(any_breaker, exposure_printer)
```

---

### any_breaker

**Is there any breaker in exposure?**

**Description**

Function designed to be used as trigger in `act_after_exposure()`. Returns TRUE if exposure attribute of `.tbl` has any information about data units not obeying the rules, i.e. rule breakers.

**Usage**

```r
any_breaker(.tbl)
```

**Arguments**

`.tbl` Result of `exposure`, i.e. data frame with `exposure` attribute.

**See Also**

`assert_any_breaker` for implicit usage of `any_breaker()`.

**Examples**

```r
mtcars %>%
  expose(data_packs(. %>% dplyr::summarise(nrow_low = nrow(.) > 50))) %>%
  any_breaker()
```

---

### assert_any_breaker

**Assert presence of rule breaker**

**Description**

Function to assert if `exposure` resulted in detecting some rule breakers.

**Usage**

```r
assert_any_breaker(.tbl, .type = "error", .silent = FALSE, ...)
```
bind_exposures

Arguments

.tbl
Result of `exposure`, i.e. data frame with `exposure` attribute.
.type
The type of assertion. Can be only one of "error", "warning" or "message".
.silent
If TRUE no printing of rule breaker information is done.
...
Arguments for printing rule breaker information.

Details

In case breaker presence this function does the following:

- In case .silent is FALSE print rows from exposure `report` corresponding to rule breakers.
- Make assertion of the chosen .type about breaker presence in exposure.
- Return .tbl (for using inside a pipe).

If there are no breakers only .tbl is returned.

See Also

- `any_breaker` for checking of breaker presence in exposure result.
- `act_after_exposure` for making general actions based in exposure result.

Examples

```r
## Not run:
mtcars %>%
  expose(data_packs(. %>% dplyr::summarise(nrow_low = nrow(.) > 50))) %>%
  assert_any_breaker()
## End(Not run)
```

bindung_exposures  Bind exposures

Description

Function to bind several exposures into one.

Usage

`bind_exposures(..., .validate_output = TRUE)`

Arguments

...
Exposures to bind.
.validate_output
Whether to validate with `is_exposure()` if the output is exposure.
Details

Note that the output doesn’t have names in list-column fun in packs info.

Examples

```r
my_data_packs <- data_packs(
  data_dims = . %>% dplyr::summarise(nrow_low = nrow(.) < 10),
  data_sum = . %>% dplyr::summarise(sum = sum(.) < 1000)
)

ref_exposure <- mtcars %>% expose(my_data_packs) %>% get_exposure()

exposure_1 <- mtcars %>% expose(my_data_packs[1]) %>% get_exposure()
exposure_2 <- mtcars %>% expose(my_data_packs[2]) %>% get_exposure()
exposure_binded <- bind_exposures(exposure_1, exposure_2)

exposure_pipe <- mtcars %>%
  expose(my_data_packs[1]) %>%
  expose(my_data_packs[2]) %>%
  get_exposure()

identical(exposure_binded, ref_exposure)
identical(exposure_pipe, ref_exposure)
```

---

**cell-pack**

**Cell rule pack**

Description

Cell rule pack is a rule pack which defines a set of rules for cells, i.e. functions which convert cells of interest to logical values. It should return a data frame with the following properties:

- Number of rows equals to number of rows for checked cells.
- Column names should be treated as concatenation of 'column name of check cell' + 'separator' + 'rule name'
- Values indicate whether the cell follows the rule.

Details

This format is inspired by scoped variants of transmute().

The most common way to define cell pack is by creating a functional sequence containing one of:

- `transmute_all(.funs = rules(...))`
- `transmute_if(.predicate,.funs = rules(...))`
- `transmute_at(.vars,.funs = rules(...))`
**Note** that (as of dplyr version 0.7.4) when only one column is transmuted, names of the output don’t have a necessary structure. The ‘column name of check cell’ is missing which results (after exposure) into empty string in var column of validation report. The current way of dealing with this is to name the input column (see examples).

**Using rules()**

Using `rules()` to create list of functions for scoped dplyr "mutating" verbs (such as `summarise_all()` and `transmute_all()`) is recommended because:

- It is a convenient way to ensure consistent naming of rules without manual name.
- It adds a common prefix to all rule names. This helps in defining separator as prefix surrounded by any number of non-alphanumeric values.

**Note about rearranging rows**

**Note** that during exposure packs are applied to keyed object with id key. So they can rearrange rows as long as it is done with functions supported by keyholder. Rows will be tracked and recognized as in the original data frame of interest.

**See Also**

Data pack, group pack, column pack, row pack.

**Examples**

```r
cell_outlier_rules <- . %>% dplyr::transmute_at(
  c("disp", "qsec"),
  rules(z_score = abs(. - mean(.)) / sd(.) > 1)
)

cell_packs(outlier = cell_outlier_rules)

# Dealing with one column edge case
improper_pack <- . %>% dplyr::transmute_at(
  dplyr::vars(vs),
  rules(improper_is_neg = . < 0)
)

proper_pack <- . %>% dplyr::transmute_at(
  dplyr::vars(vs = vs),
  rules(proper_is_neg = . < 0)
)

mtcars[1:2, ] %>%
expose(cell_packs(improper_pack, proper_pack)) %>%
get_report()
```
Description

Column rule pack is a rule pack which defines a set of rules for columns as a whole, i.e. functions which convert columns of interest to logical values. It should return a data frame with the following properties:

- Number of rows equals to one.
- Column names should be treated as concatenation of 'check column name' + 'separator' + 'rule name'.
- Values indicate whether the column as a whole follows the rule.

Details

This format is inspired by dplyr’s scoped variants of summarise() applied to non-grouped data.

The most common way to define column pack is by creating a functional sequence with no grouping and ending with one of:

- `summarise_all(.funs = rules(...)).`
- `summarise_if(.predicate,.funs = rules(...)).`
- `summarise_at(.vars,.funs = rules(...)).`

Note that (as of dplyr version 0.7.4) when only one column is summarised, names of the output don’t have a necessary structure. The 'check column name' is missing which results (after exposure) into empty string in var column of validation report. The current way of dealing with this is to name the input column (see examples).

Using rules()

Using `rules()` to create list of functions for scoped dplyr "mutating" verbs (such as `summarise_all()` and `transmute_all()`) is recommended because:

- It is a convenient way to ensure consistent naming of rules without manual name.
- It adds a common prefix to all rule names. This helps in defining separator as prefix surrounded by any number of non-alphanumeric values.

See Also

Data pack, group pack, row pack, cell pack.
Examples

# Validating present columns
numeric_column_rules <- . %>% dplyr::summarise_if(
  is.numeric,
  rules(mean(.) > 5, sd(.) < 10)
)
character_column_rules <- . %>% dplyr::summarise_if(
  is.character,
  rules(. %in% letters[1:4])
)

col_packs(
  num_col = numeric_column_rules,
  chr_col = character_column_rules
)

# Dealing with one column edge case
improper_pack <- . %>% dplyr::summarise_at(
  dplyr::vars(vs),
  rules(improper_is_chr = is.character)
)

proper_pack <- . %>% dplyr::summarise_at(
  dplyr::vars(vs = vs),
  rules(proper_is_chr = is.character)
)

mtcars %>%
  expose(col_packs(improper_pack, proper_pack)) %>%
  get_report()

---

Data rule pack

Description

Data rule pack is a rule pack which defines a set of rules for data as a whole, i.e. functions which convert data to logical values. It should return a data frame with the following properties:

- Number of rows equals to one.
- Column names should be treated as rule names.
- Values indicate whether the data as a whole follows the rule.

Details

This format is inspired by dplyr’s summarise() applied to non-grouped data.

The most common way to define data pack is by creating a functional sequence with no grouping and ending with summarise(...).
See Also

Group pack, Column pack, row pack, cell pack.

Examples

data_dims_rules <- .>%
  dplyr::summarise(nrow_low = nrow(.) > 10,
                   nrow_up = nrow(.) < 20,
                   ncol_low = ncol(.) > 5,
                   ncol_up = ncol(.) < 10)
data_na_rules <- .>%
  dplyr::summarise(all_not_na = Negate(anyNA)(.))
data_packs(
  data_nrow = data_dims_rules,
  data_na = data_na_rules)

expose

Expose data to rule packs

Description

Function for applying rule packs to data.

Usage

expose(.tbl, ..., .rule_sep = inside_punct("\_\_.\_."),
       .remove_obeyers = TRUE, .guess = TRUE)

Arguments

.tbl Data frame of interest.

... Rule packs. They can be in pure form or inside a list (at any depth).

.rule_sep Regular expression used as separator between column and rule names in col
   packs and cell packs.

.remove_obeyers Whether to remove elements which obey rules from report.

.guess Whether to guess type of unsupported rule pack type (see Details).
Details

expose() applies all supplied rule packs to data, creates an exposure object based on results and stores it to attribute 'exposure'. It is guaranteed that .tbl is not modified in any other way in order to use expose() inside a pipe.

It is a good idea to name all rule packs: explicitly in ... (if they are supplied not inside list) or during creation with respective rule pack function. In case of missing name it is imputed based on possibly existing exposure attribute in .tbl and supplied rule packs. Imputation is similar to one in rules() but applied to every pack type separately.

Default value for .rule_sep is the regular expression characters ._. surrounded by non alphanumerical characters. It is picked to be used smoothly with dplyr’s scoped verbs and rules() instead of funs(). In most cases it shouldn’t be changed but if needed it should align with .prefix in rules().

Value

A .tbl with possibly added 'exposure' attribute containing the resulting exposure. If .tbl already contains 'exposure' attribute then the result is binded with it.

Guessing

To work properly in some edge cases one should specify pack types with appropriate function. However with .guess equals to TRUE expose will guess the pack type based on its output after applying to .tbl. It uses the following features:

- Presence of non-logical columns: if present then the guess is group pack. Grouping columns are guessed as all non-logical. This works incorrectly if some grouping column is logical: it will be guessed as result of applying the rule. Note that on most occasions this edge case will produce error about grouping columns define non-unique levels.

- Combination of whether number of rows equals 1 (n_rows_one) and presence of .rule_sep in all column names (all_contain_sep). Guesses are:
  - Data pack if n_rows_one == TRUE and all_contain_sep == FALSE.
  - Column pack if n_rows_one == TRUE and all_contain_sep == TRUE.
  - Row pack if n_rows_one == FALSE and all_contain_sep == FALSE. This works incorrectly if output has one row which is checked. In this case it will be guessed as data pack.
  - Cell pack if n_rows_one == FALSE and all_contain_sep == TRUE. This works incorrectly if output has one row in which cells are checked. In this case it will be guessed as column pack.

Examples

my_rule_pack <- . %>% dplyr::summarise(nrow_neg = nrow(.) < 0)
my_data_packs <- data_packs(my_data_pack_1 = my_rule_pack)

# These pipes give identical results
mtcars %>% expose(my_data_packs) %>% get_report()
mtcars %>% expose(my_data_pack_1 = my_rule_pack) %>% get_report()
# This throws an error because no pack type is specified for my_rule_pack
## Not run:
  mtcars %>% expose(my_data_pack_1 = my_rule_pack, .guess = FALSE)
## End(Not run)

# Edge cases against using 'guess = TRUE' for robust code
# Not run

group_rule_pack <- . %>% dplyr::mutate(vs_one = vs == 1) %>%
  dplyr::group_by(vs_one, am) %>%
  dplyr::summarise(n_low = dplyr::n() > 10)
group_rule_pack_dummy <- . %>% dplyr::mutate(vs_one = vs == 1) %>%
  dplyr::group_by(mpg, vs_one, wt) %>%
  dplyr::summarise(n_low = dplyr::n() > 10)
row_rule_pack <- . %>% dplyr::transmute(neg_row_sum = rowSums(.) < 0)
cell_rule_pack <- . %>% dplyr::transmute_all(rules(neg_value = . < 0))

# Only column 'am' is guessed as grouping which defines non-unique levels.
## Not run:
  mtcars %>%
    expose(group_rule_pack, .remove_obeyers = FALSE, .guess = TRUE) %>%
    get_report()
## End(Not run)

# Values in 'var' should contain combination of three grouping columns but
# column 'vs_one' is guessed as rule. No error is thrown because the guessed
# grouping column define unique levels.
mtcars %>%
  expose(group_rule_pack_dummy, .remove_obeyers = FALSE, .guess = TRUE) %>%
  get_report()

# Results should have in column 'id' value 1 and not 0.
mtcars %>% dplyr::slice(1) %>%
  expose(row_rule_pack) %>%
  get_report()

mtcars %>% dplyr::slice(1) %>%
  expose(cell_rule_pack) %>%
  get_report()

---

**exposure**

**Exposure**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exposure is a result of exposing data to rules. It is implemented with S3 class exposure which is a list of the following structure: packs_info - a packs_info object; report - tidy data validation report.</td>
</tr>
</tbody>
</table>

**Usage**

`is_exposure(.x)`
get_exposure(.object)

remove_exposure(.object)

**Arguments**

- `.x` Object to test.
- `.object` Object to get or remove `exposure` attribute from.

**Value**

- `get_exposure()` returns object if it is exposure and its attribute 'exposure' otherwise.
- `remove_exposure()` returns object with removed attributed 'exposure'.

**Examples**

```r
my_col_packs <- col_packs(
  col_sum_props = . %>% dplyr::summarise_all(
    rules(
      col_sum_low = sum(.) > 100,
      col_sum_high = sum(.) < 1000
    )
  )
)
mtcars_exposed <- mtcars %>% expose(my_col_packs)
mtcars_exposure <- mtcars_exposed %>% get_exposure()

is_exposure(mtcars_exposure)

identical(remove_exposure(mtcars_exposed), mtcars)

identical(get_exposure(mtcars_exposure), mtcars_exposure)
```

**group-pack**  

*Group rule pack*

**Description**

Group rule pack is a rule pack which defines a set of rules for groups of rows as a whole, i.e. functions which convert groups of interest to logical values. It should return a data frame with the following properties:

- There should be present some columns which combined values uniquely describe group. They should be defined during creation with `group_packs()`.
- Number of rows equals to number of checked groups.
- Names of non-grouping columns should be treated as rule names.
- Values indicate whether the group as a whole follows the rule.
Details

This format is inspired by dplyr’s summarise() applied to grouped data.
The most common way to define data pack is by creating a functional sequence with grouping and
ending with summarise(...).

Interpretation

Group pack output is interpreted in the following way:

- All grouping columns are united with delimiter .group_sep (which is an argument of group_packs()).
- Levels of the resulting column are treated as names of some new variables which should be
  exposed as a whole. Names of non-grouping columns are treated as rule names. They are
  transformed in column pack format and interpreted accordingly.

Exposure result of group pack is different from others in a way that column var in exposure report
doesn’t represent the actual column in data.

See Also

Data pack, Column pack, row pack, cell pack.

Examples

vs_am_rules <- . %>%
dplyr::group_by(vs, am) %>%
dplyr::summarise(nrow_low = n(.) > 10,
                 nrow_up = n(.) < 20,
                 rowmeans_low = rowMeans(.) > 19)

group_packs(vs_am = vs_am_rules, .group_vars = c("vs", "am"))

inside_punct

Inside punctuation regular expression

Description

Function to construct regular expression of form: ‘non alpha-numeric characters’ + ‘some charac-
ters’ + ‘non alpha-numeric characters’.

Usage

inside_punct(.x = "\\._\\.")

Arguments

.x     Middle characters to be put between non alpha-numeric characters.
**Description**

An S3 class `packs_info` to represent information about packs in `exposure`. It is a tibble with the following structure:

- **name** <chr>: Name of the pack.
- **type** <chr>: Pack type.
- **fun** <list>: List (preferably unnamed) of rule pack functions.
- **remove_obeyers** <lgl>: value of `.remove_obeyers` argument of `expose()` with which pack was applied.

**Usage**

```r
is_packs_info(.x, .skip_class = FALSE)
get_packs_info(.object)
```

**Arguments**

- `.x` Object to test.
- `.skip_class` Whether to skip checking inheritance from `packs_info`.
- `.object` Object to get `packs_info` value from `exposure` attribute.

**Details**

To avoid possible confusion it is preferred (but not required) that list-column `fun` doesn’t have names. Names of packs are stored in `name` column. During `exposure` `fun` is always created without names.

**Value**

`get_packs_info()` returns `packs_info` attribute of `object` if it is `exposure` and of its ‘`exposure`’ attribute otherwise.
Examples

```r
my_row_packs <- row_packs(
  row_mean Props = . %>% dplyr::transmute(row_mean = rowMeans(.)) %>%
  dplyr::transmute(
    row_mean_low = row_mean > 20,
    row_mean_high = row_mean < 60
  ),
  row_outlier = . %>% dplyr::transmute(row_sum = rowSums(.)) %>%
  dplyr::transmute(
    not_row_outlier = abs(row_sum - mean(row_sum)) / sd(row_sum) < 1.5
  )
)
my_data_packs <- data_packs(
  data_dims = . %>% dplyr::summarise(nrow = nrow(.) == 32,
                                   ncol = ncol(.) == 5)
)
mtcars_exposed <- mtcars %>%
  expose(my_data_packs, .remove_obeyers = FALSE) %>%
  expose(my_row_packs)
mtcars_exposed %>% get_packs_info()
mtcars_exposed %>% get_packs_info() %>% is_packs_info()
```

---

**row-pack**

**Row rule pack**

**Description**

Row rule pack is a rule pack which defines a set of rules for rows as a whole, i.e. functions which convert rows of interest to logical values. It should return a data frame with the following properties:

- Number of rows equals to **number of checked rows**.
- Column names should be treated as **rule names**.
- Values indicate whether the **row as a whole** follows the rule.

**Details**

This format is inspired by dplyr’s `transmute()`.

The most common way to define row pack is by creating a functional sequence containing `transmute(...)`.

**Note about rearranging rows**

*Note* that during exposure packs are applied to keyed object with id key. So they can rearrange rows as long as it is done with functions supported by keyholder. Rows will be tracked and recognized as in the original data frame of interest.
rule-packs

See Also
Data pack, group pack, column pack, cell pack.

Examples

```r
some_row_mean_rules <- . %>% dplyr::slice(1:3) %>%
  dplyr::mutate(row_mean = rowMeans(.)) %>%
  dplyr::transmute(
    row_mean_low = row_mean > 10,
    row_mean_up = row_mean < 20
  )

all_row_sum_rules <- . %>% dplyr::mutate(row_sum = rowSums(.)) %>%
  dplyr::transmute(row_sum_low = row_sum > 30)

row_packs(
  some_row_mean_rules,
  all_row_sum_rules
)
```

rule-packs Create rule packs

Description
Functions for creating different kinds of rule packs. Rule is a function which converts data unit of interest (data, group, column, row, cell) to logical value indicating whether this object satisfies certain condition. Rule pack is a function which combines several rules into one functional block. It takes a data frame of interest and returns a data frame with certain structure and column naming scheme. Types of packs differ in interpretation of their output.

Usage
data_packs(...)

group_packs(..., .group_vars, .group_sep = ".")

col_packs(...)

row_packs(...)

cell_packs(...)

Arguments

... Functions which define packs. They can be in pure form or inside a list (at any depth).

.group_vars Character vector of names of grouping variables.
.group_sep String to be used as separator when uniting grouping levels for var column in exposure report.

Details

These functions convert . . . to list, apply rlang’s squash() and add appropriate classes (group_packs() also adds necessary attributes). Also they are only constructors and do not check for validity of certain pack. Note that it is allowed for elements of . . . to not have names: they will be computed during exposure. However it is a good idea to manually name packs.

Value

data_packs() returns a list of what should be data rule packs, group_packs() - group rule packs, col_packs() - column rule packs, row_packs() - row rule packs, cell_packs() - cell rule packs.

---

ruler-report  Tidy data validation report

Description

A tibble representing the data validation result of certain data units in tidy way:

- pack <chr>: Name of rule pack from column 'name' of corresponding packs_info object.
- rule <chr>: Name of the rule defined in rule pack.
- var <chr>: Name of the variable which validation result is reported. Value '.all' is reserved and interpreted as 'all columns as a whole'. Note that var doesn’t always represent the actual column in data frame (see group packs).
- id <int>: Index of the row in tested data frame which validation result is reported. Value 0 is reserved and interpreted as 'all rows as a whole'.
- value <lgl>: Whether the described data unit obeys the rule.

Usage

is_report(.x, .skip_class = FALSE)

get_report(.object)

Arguments

.x Object to test.

.skip_class Whether to skip checking inheritance from ruler_report.

.object Object to get report value from exposure attribute.
Details

There are four basic combinations of var and id values which define five basic data units:

- var == `.all` and id == 0: Data as a whole.
- var != `.all` and id == 0: Group (var shouldn’t be an actual column name) or column (var should be an actual column name) as a whole.
- var == `.all` and id != 0: Row as a whole.
- var != `.all` and id != 0: Described cell.

Value

get_report() returns report element of object if it is exposure and of its ‘exposure’ attribute otherwise.

Examples

```r
my_row_packs <- row_packs(
  row_mean_props = . %>% dplyr::transmute(row_mean = rowMeans(.)) %>%
    dplyr::transmute(
      row_mean_low = row_mean > 20,
      row_mean_high = row_mean < 60
    )
  row_outlier = . %>% dplyr::transmute(row_sum = rowSums(.)) %>%
    dplyr::transmute(
      not_row_outlier = abs(row_sum - mean(row_sum)) / sd(row_sum) < 1.5
    )
)
my_data_packs <- data_packs(
  data_dims = . %>% dplyr::summarise(nrow = nrow(.) == 32,
    ncol = ncol(.) == 5)
)
mtcars_exposed <- mtcars %>%
  expose(my_data_packs, .remove_obeyers = FALSE) %>%
  expose(my_row_packs)
mtcars_exposed %>% get_report()
mtcars_exposed %>% get_report() %>% is_report()
```

rules

Create a list of rules

Description

rules() is a function designed to create input for .funs argument of scoped dplyr "mutating" verbs (such as summarise_all() and transmute_all()). For version of dplyr less than 0.8.0 it is a direct wrapper for funs() which does custom name repair (see 'Details'). For newer versions it converts bare expressions with . as input into formulas and repairs names of the output.
Usage

```r
rules(..., .args = list(), .prefix = "_._")
```

Arguments

... Element(s) suitable as .funs argument (in scoped "mutating" verbs) for current version of dplyr. It can also be a bare expression with . as input even if dplyr version is 0.8.0 or newer.

`.args` A named list of additional arguments to be added to all function calls (as in dplyr::funs()). **Note** that this argument isn’t used if installed version of dplyr is 0.8.0 or newer. Use other methods to supply arguments: ... argument in scoped verbs or make own explicit functions.

`.prefix` Prefix to be added to function names.

Details

rules() repairs names by the following algorithm:

- Absent names are replaced with the 'rule__\ind\' where \ind\ is the index of function position in the ... .
- `.prefix` is added at the beginning of all names. The default is _. . It is picked for its symbolism (it is the Morse code of letter 'R') and rare occurrence in names. In those rare cases it can be manually changed but this will not be tracked further. **Note** that it is a good idea for `.prefix` to be syntactic, as newer versions of dplyr (>= 0.8.0) will force tibble names to be syntactic. To check if string is "good", use it as input to make.names(): if output equals that string than it is a "good" choice.

Examples

```r
if (utils::packageVersion("dplyr") < "0.8.0") {
  rules_1 <- rules(mean, sd, .args = list(na.rm = TRUE))
  rules_1_ref <- dplyr::funs('._rule__1' = mean, '._rule__2' = sd, .args = list(na.rm = TRUE))
  identical(rules_1, rules_1_ref)

  rules_2 <- rules(mean, sd = sd, "var")
  rules_2_ref <- dplyr::funs( '._rule__1' = mean,
                             '._sd' = sd,
                             '._rule__3' = "var"
  )
  identical(rules_2, rules_2_ref)

  rules_3 <- rules(mean, .prefix = "a_a_")
  rules_3_ref <- dplyr::funs('a_a_rule__1' = mean)
  identical(rules_3, rules_3_ref)
}
```

if (utils::packageVersion("dplyr") >= "0.8.0") {
  # `rules()` also accepts bare expression calls with `.` as input, which is
# not possible with advised `list()` approach of `dplyr`
dplyr::summarise_all(mtcars[, 1:2], rules(sd, "sd", sd(.), ~ sd(.)))

dplyr::summarise_all(mtcars[, 1:2], rules(sd, .prefix = "a_a_"))

# Use `...' in `summarise_all()` to supply extra arguments
dplyr::summarise_all(data.frame(x = c(1:2, NA)), rules(sd), na.rm = TRUE)

spread_groups

Spread grouping columns

Description

Function that is used during interpretation of group pack output. It converts grouped summary into column pack format.

Usage

spread_groups(.tbl, ..., .group_sep = ".", .col_sep = "._.")

Arguments

.tbl Data frame with result of grouped summary.
... A selection of grouping columns (as in tidyr::unite()).
.group_sep A string to be used as separator of grouping levels.
.col_sep A string to be used as separator in column pack.

Details

Multiple grouping variables are converted to one with tidyr::unite() and separator .group_sep. New values are then treated as variable names which should be validated and which represent the group data as a whole.

Value

A data frame in column pack format.

Examples

mtcars_grouped_summary <- mtcars %>%
dplyr::group_by(vs, am) %>%
dplyr::summarise(n_low = dplyr::n() > 6, n_high = dplyr::n() < 10)

spread_groups(mtcars_grouped_summary, vs, am)

spread_groups(mtcars_grouped_summary, vs, am, .group_sep = "__")
spread_groups(mtcars_grouped_summary, vs, am, .col_sep = "__")
Index

about applied packs, 2  get_exposure (exposure), 12
act after exposure, 3, 5  get_packs_info (packs_info), 15
act after exposure(), 4  get_report (ruler-report), 18
actions, 2  Group pack, 10
going breaker, 3, 4, 5  group pack, 7, 8, 11, 17, 21
appropriate function, 11  group packs, 18
gassert about rule breakers, 2  group rule packs, 18
gassert_any_breaker, 3, 4  group-pack, 13
gbind_exposures, 5  group_packs (rule-packs), 17
gCell pack, 11  group_packs (), 13
gcell pack, 8, 10, 14, 17  id key, 7, 16
gcell packs, 10  inside_punct, 14
gcell rule packs, 18  is_exposure (exposure), 12
gcell_packs (rule-packs), 17  is_exposure(), 5

gcell_packs (rule-packs), 17  is_packs_info (packs_info), 15
gcol packs, 10  is_report (ruler-report), 18

gcol_packs (rule-packs), 17  keyed object, 7, 16

gCol pack, 10, 11, 14  Pack type, 15
gcolumn pack, 7, 14, 17, 21  packs, 2

gcolumn rule packs, 18  packs (rule-packs), 17

gcolumn pack, 8  packs_info, 6

gcolumn-pack, 13  packs_info, 12, 15, 18

gcol_packs (), 17  pipe, 3, 5, 11

gdetecting, 4  remove_exposure (exposure), 12

gdata_packs (rule-packs), 17  report, 5

gdata pack, 7, 8, 11, 14, 17  Row pack, 11

gdata rule packs, 18  row pack, 7, 8, 10, 14

gdata-pack, 9  row rule packs, 18

gdata_packs (rule-packs), 17  row-pack, 16

gdetecting, 4  row_packs (rule-packs), 17

Expose, 2  rule pack, 6, 8, 9, 13, 16

gexpose, 10  rule-packs, 17

gexpose(), 15  ruler (ruler-package), 2

gexposing, 12  ruler-package, 2

gexposure, 2–5, 7, 8, 11, 12, 15  ruler-report, 18

gexposure report, 14, 18  rules, 19

gfunctional sequence, 6, 8, 9, 14, 16

functions supported by keyholder, 7, 16

funs(), 11, 19
rules(), 7, 8, 11
scoped variants of summarise(), 8
scoped variants of transmute(), 6
scoped verbs, 11, 20
spread_groups, 21
squash(), 18
summarise(), 9, 14
summarise_all(), 7, 8, 19
summary, 21
syntactic, 20
tidy data validation report, 2, 12
tidyr::unite(), 21
transmute(), 16
transmute_all(), 7, 8, 19
united, 14
validation report, 7, 8