Package ‘tidyselect’

January 27, 2020

Title  Select from a Set of Strings

Version  1.0.0

Description  A backend for the selecting functions of the 'tidyverse'.
             It makes it easy to implement select-like functions in your own
             packages in a way that is consistent with other 'tidyverse'
             interfaces for selection.

Depends  R (>= 3.2)

Imports  ellipsis, glue (>= 1.3.0), purrr (>= 0.3.2), rlang (>= 0.4.3),
         vctrs (>= 0.2.2)

Suggests  covr, crayon, dplyr, knitr, magrittr, rmarkdown, testthat (>=
          2.3.0), tibble (>= 2.1.3), withr

License  GPL-3

Encoding  UTF-8

LazyData  true

ByteCompile  true

RoxygenNote  7.0.2


BugReports  https://github.com/r-lib/tidyselect/issues

VignetteBuilder  knitr

NeedsCompilation  yes

Author  Lionel Henry [aut, cre],
        Hadley Wickham [aut],
        RStudio [cph]

Maintainer  Lionel Henry <lionel@rstudio.com>

Repository  CRAN

Date/Publication  2020-01-27 20:20:02 UTC
R topics documented:

- eval_rename ............................... 2
-faq-external-vector .................... 4
-faq-selection-context ................... 6
-peek-vars ................................. 6
-select Helpers ............................ 7

Description

eval_select() and eval_rename() evaluate defused R code (i.e. quoted expressions) according to the special rules of the tidyselect syntax. They power functions like dplyr::select(), dplyr::rename(), or tidyr::pivot_longer().

See the Get started vignette to learn how to use eval_select() and eval_rename() in your packages.

Usage

eval_rename(
  expr,
  data,
  env = caller_env(),
  ..., strict = TRUE,
  name_spec = NULL
)
eval_select(
  expr,
  data,
  env = caller_env(),
  ..., strict = TRUE,
  include = NULL,
  exclude = NULL,
  name_spec = NULL
)

Arguments

expr  Defused R code describing a selection according to the tidyselect syntax.
data  A named list, data frame, or atomic vector. Technically, data can be any vector with names() and "[[" implementations.
The environment in which to evaluate `expr`. Discarded if `expr` is a quosure.

These dots are for future extensions and must be empty.

If `TRUE`, out-of-bounds errors are thrown if `expr` attempts to select or rename a variable that doesn’t exist. If `FALSE`, failed selections or renamings are ignored.

A name specification describing how to combine or propagate names. This is used only in case nested `c()` expressions like `c(foo = c(bar = starts_with("foo")))`. See the `name_spec` argument of `vctrs::vec_c()` for a description of valid name specs.

Character vector of column names to always include or exclude from the selection.

The select and rename variants take the same types of inputs and have the same type of return value. However `eval_rename()` has a few extra constraints. It requires named inputs, and will fail if a data frame column is renamed to another existing column name. See the selecting versus renaming section in the syntax vignette for a description of the differences.

A named vector of numeric locations, one for each of the selected elements.

The names are normally the same as in the input data, except when the user supplied named selections with `c()`. In the latter case, the names reflect the new names chosen by the user.

A given element may be selected multiple times under different names, in which case the vector might contain duplicate locations.


```
library(rlang)

# Interpret defused code as selection:
x <- expr(mpg:cyl)
eval_select(x, mtcars)

# Interpret defused code as a renaming selection. All inputs must # be named within `c()`: try(eval_rename(expr(mpg), mtcars))
eval_rename(expr(c(foo = mpg)), mtcars)

# The evaluators return a named vector of locations. Here are # examples of using these location vectors to implement `select()` # and `rename()`:
```
faq-external-vector

```r
select <- function(.x, ...) {
  pos <- eval_select(expr(c(...)), .x)
  set_names(.x[pos], names(pos))
}
rename <- function(.x, ...) {
  pos <- eval_rename(expr(c(...)), .x)
  names(.x)[pos] <- names(pos)
  .x
}

select(mtcars, mpg:cyl)
rename(mtcars, foo = mpg)
```

faq-external-vector

**FAQ - Note: Using an external vector in selections is ambiguous**

---

**Description**

**Ambiguity between columns and external variables:**

With selecting functions like `dplyr::select()` or `tidyr::pivot_longer()`, you can refer to variables by name:

```r
mtcars %>% select(cyl, am, vs)
```

```r
# A tibble: 32 x 3
#> cyl am vs
#> <dbl> <dbl> <dbl>
#> 1 6 1 0
#> 2 6 1 0
#> 3 4 1 1
#> 4 6 0 1
#> ... with 28 more rows
```

```r
mtcars %>% select(mpg:disp)
```

```r
# A tibble: 32 x 3
#> mpg cyl disp
#> <dbl> <dbl> <dbl>
#> 1 21 6 160
#> 2 21 6 160
#> 3 22.8 4 108
#> 4 21.4 6 258
#> ... with 28 more rows
```

For historical reasons, it is also possible to refer an external vector of variable names. You get the correct result, but with a note informing you that selecting with an external variable is ambiguous because it is not clear whether you want a data frame column or an external object.

```r
vars <- c("cyl", "am", "vs")
result <- mtcars %>% select(vars)
```

```r
# Note: Using an external vector in selections is ambiguous.
```
This note will become a warning in the future, and then an error. We have decided to deprecate this particular approach to using external vectors because they introduce ambiguity. Imagine that the data frame contains a column with the same name as your external variable.

```r
some_df <- mtcars
some_df$vars <- 1:nrow(mtcars)
```

These are very different objects but it isn’t a problem if the context forces you to be specific about where to find `vars`:

```r
vars
#> [1] "cyl" "am" "vs"
```

```r
some_df$vars
#> [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
#> [29] 29 30 31 32
```

In a selection context however, the column wins:

```r
some_df %>% select(vars)
#> # A tibble: 32 x 1
#> vars
#> <int>
#> 1  1
#> 2  2
#> 3  3
#> 4  4
#> # ... with 28 more rows
```

**Fixing the ambiguity:**

To make your selection code more robust and silence the message, use `all_of()` to force the external vector:

```r
some_df %>% select(all_of(vars))
#> # A tibble: 32 x 3
#>    cyl  am  vs
#>  <dbl> <dbl> <dbl>
#> 1    6   1   0
#> 2    6   1   0
#> 3    4   1   1
#> 4    6   0   1
#> # ... with 28 more rows
```

For more information or if you have comments about this, please see the [Github issue](https://github.com/tidyverse/tidyselect/issues) tracking the deprecation process.
**faq-selection-context**  FAQ - Error: Must be used within a selecting function

**Description**

Functions like `starts_with()`, `contains()` or `matches()` are **selection helpers** that only work in a selection context.

Examples of valid selection contexts are:

- Inside `dplyr::select()`.
- The `cols` argument of `tidyr::pivot_longer()`.

Using a selection helper anywhere else results in an error:

```r
starts_with("foo")
#> Error: `starts_with()` must be used within a *selecting* function.
#> i See <https://tidyselect.r-lib.org/reference/faq-selection-context.html>.
```

```r
mtcars[contains("foo")]
#> Error: `contains()` must be used within a *selecting* function.
#> i See <https://tidyselect.r-lib.org/reference/faq-selection-context.html>.
```

```r
subset(mtcars, select = matches("foo"))
#> Error: `matches()` must be used within a *selecting* function.
#> i See <https://tidyselect.r-lib.org/reference/faq-selection-context.html>.
```

If you see this error, you’ve probably used a selection helper in the wrong place, possibly as the result of a typo (e.g. misplaced comma or wrong argument name).

---

**peek_vars**  Peek at variables in the selection context

**Description**

- `peek_vars()` returns the vector of names of the variables currently available for selection.
- `peek_data()` returns the whole input vector (only available with `eval_select()`).

Read the **Get started** for examples of how to create selection helpers with `peek_vars()`.

The variable names in a selection context are registered automatically by `eval_select()` and `eval_rename()` for the duration of the evaluation. `peek_vars()` is the glue that connects selection helpers to the current selection context.

**Usage**

```r
peek_vars(..., fn = NULL)
peek_data(..., fn = NULL)
```
select_helpers

Arguments

... These dots are for future extensions and must be empty.

fn The name of the function to use in error messages when the helper is used in the wrong context. If not supplied, a generic error message is used instead.

Description

These functions allow you to select variables based on their names.

- `starts_with()`: Starts with a prefix.
- `ends_with()`: Ends with a suffix.
- `contains()`: Contains a literal string.
- `matches()`: Matches a regular expression.
- `num_range()`: Matches a numerical range like x01, x02, x03.
- `all_of()`: Matches variable names in a character vector. All names must be present, otherwise an out-of-bounds error is thrown.
- `any_of()`: Same as `all_of()`, except that no error is thrown for names that don’t exist.
- `everything()`: Matches all variables.
- `last_col()`: Select last variable, possibly with an offset.

Usage

```r
starts_with(match, ignore.case = TRUE, vars = peek_vars(fn = "starts_with"))

ends_with(match, ignore.case = TRUE, vars = peek_vars(fn = "ends_with"))

contains(match, ignore.case = TRUE, vars = peek_vars(fn = "contains"))

matches(
  match,
  ignore.case = TRUE,
  perl = FALSE,
  vars = peek_vars(fn = "matches"
)

num_range(prefix, range, width = NULL, vars = peek_vars(fn = "num_range"))

all_of(x)

any_of(x, ..., vars = peek_vars(fn = "any_of"))

everything(vars = peek_vars(fn = "everything"))

last_col(offset = 0L, vars = peek_vars(fn = "last_col"))
```
Arguments

- `match` A character vector. If length > 1, the union of the matches is taken.
- `ignore.case` If TRUE, the default, ignores case when matching names.
- `vars` A character vector of variable names. When called from inside selecting functions like `dplyr::select()` these are automatically set to the names of the table.
- `perl` Should Perl-compatible regexps be used?
- `prefix` A prefix that starts the numeric range.
- `range` A sequence of integers, like `1:5`.
- `width` Optionally, the "width" of the numeric range. For example, a range of 2 gives "01", a range of three "001", etc.
- `x` An index vector of names or locations.
- `...` These dots are for future extensions and must be empty.
- `offset` Set it to n to select the nth var from the end.

Details

In selection context you can also use these operators:

- "/" for taking the difference between two sets of variables.
- ":" for selecting a range of consecutive variables.
- "c" for selecting the union of sets of variables.

The boolean operators were more recently overloaded to operate on selections:

- "!" for taking the complement of a set of variables.
- "&" and "|" for selecting the intersection or the union of two sets of variables.

The order of selected columns is determined by the inputs.

- `one_of(c("foo","bar"))` selects "foo" first.
- `c(starts_with("c"),starts_with("d"))` selects all columns starting with "c" first, then all columns starting with "d".

Value

An integer vector giving the position of the matched variables.

Examples

```r
nms <- names(iris)
vars_select(nms, starts_with("Petal"))
vars_select(nms, ends_with("Width"))
vars_select(nms, contains("etal"))
vars_select(nms, matches(".t."))
vars_select(nms, Petal.Length, Petal.Width)
vars_select(nms, everything())
```
vars_select(nms, last_col())
vars_select(nms, last_col(offset = 2))

# With multiple matchers, the union of the matches is selected:
vars_select(nms, starts_with(c("Petal", "Sepal")))

# `!` negates a selection:
vars_select(nms, !ends_with("Width"))

# `&` and `|` take the intersection or the union of two selections:
vars_select(nms, starts_with("Petal") & ends_with("Width"))
vars_select(nms, starts_with("Petal") | ends_with("Width"))

# `/` takes the difference of two selections
vars_select(nms, starts_with("Petal") / ends_with("Width"))

# `all_of()` selects the variables in a character vector:
vars <- c("Petal.Length", "Petal.Width")
vars_select(nms, all_of(vars))

# Whereas `all_of()` is strict, `any_of()` allows missing
# variables.
try(vars_select(nms, all_of(c("Species", "Genres"))))
vars_select(nms, any_of(c("Species", "Genres")))

# The lax variant is especially useful to make sure a variable is
# selected out:
vars_select(nms, -any_of(c("Species", "Genres")))

# The order of selected columns is determined from the inputs
vars_select(names(mtcars), starts_with("c"), starts_with("d"))
vars_select(names(mtcars), one_of(c("carb", "mpg")))
all_of(select_helpers), 7
any_of(select_helpers), 7
contains(select_helpers), 7
dplyr::select(), 8
ends_with(select_helpers), 7
eval_rename, 2
eval_rename(), 6
eval_select(eval_rename), 2
eval_select(), 6
everything(select_helpers), 7
faq-external-vector, 4
faq-selection-context, 6
last_col(select_helpers), 7
matches(select_helpers), 7
num_range(select_helpers), 7
peek_data(peek_vars), 6
peek_vars, 6
quosure, 3
select_helpers, 7
selection helpers, 6
starts_with(select_helpers), 7
vctrs::vec_c(), 3