Package ‘tidyselect’

March 11, 2024

Title  Select from a Set of Strings

Version  1.2.1

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.

License  MIT + file LICENSE


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

Depends  R (>= 3.4)

Imports  cli (>= 3.3.0), glue (>= 1.3.0), lifecycle (>= 1.0.3), rlang (>= 1.0.4), vctrs (>= 0.5.2), withr

Suggests  covr, crayon, dplyr, knitr, magrittr, rmarkdown, stringr, testthat (>= 3.1.1), tibble (>= 2.1.3)

VignetteBuilder  knitr

ByteCompile  true

Config/testthat/edition  3

Config/Needs/website  tidyverse/tidytemplate

Encoding  UTF-8

RoxygenNote  7.3.0.9000

NeedsCompilation  yes

Author  Lionel Henry [aut, cre], Hadley Wickham [aut], Posit Software, PBC [cph, fnd]

Maintainer  Lionel Henry <lionel@posit.co>

Repository  CRAN

Date/Publication  2024-03-11 14:10:02 UTC
all_of

R topics documented:

- all_of
- eval_relocate
- eval_rename
- everything
- faq-external-vector
- faq-selection-context
- language
- peek_vars
- starts_with
- tidyselect_data_proxy
- where

Index

all_of

Select variables from character vectors

Description

These selection helpers select variables contained in a character vector. They are especially useful for programming with selecting functions.

- `all_of()` is for strict selection. If any of the variables in the character vector is missing, an error is thrown.
- `any_of()` doesn't check for missing variables. It is especially useful with negative selections, when you would like to make sure a variable is removed.

The order of selected columns is determined by the order in the vector.

Usage

```r
all_of(x)

any_of(x, ..., vars = NULL)
```

Arguments

- `x`: A vector of character names or numeric locations.
- `...`: These dots are for future extensions and must be empty.
- `vars`: A character vector of variable names. If not supplied, the variables are taken from the current selection context (as established by functions like `select()` or `pivot_longer()`).
Examples

Selection helpers can be used in functions like `dplyr::select()` or `tidyr::pivot_longer()`. Let’s first attach the tidyverse:

```
library(tidyverse)

# For better printing
iris <- as_tibble(iris)
```

It is a common to have a names of variables in a vector.

```
vars <- c("Sepal.Length", "Sepal.Width")
iris[, vars]
```

```
# A tibble: 150 x 2
#> Sepal.Length Sepal.Width
#> <dbl> <dbl>
#> 1 5.1 3.5
#> 2 4.9 3
#> 3 4.7 3.2
#> 4 4.6 3.1
#> # i 146 more rows
```

To refer to these variables in selecting function, use `all_of()`:

```
iris %>% select(all_of(vars))
```

```
# A tibble: 150 x 2
#> Sepal.Length Sepal.Width
#> <dbl> <dbl>
#> 1 5.1 3.5
#> 2 4.9 3
#> 3 4.7 3.2
#> 4 4.6 3.1
#> # i 146 more rows
```

```
iris %>% pivot_longer(all_of(vars))
```

```
# A tibble: 300 x 5
#> Petal.Length Petal.Width Species name value
#> <dbl> <dbl> <fct> <chr> <dbl>
#> 1 1.4 0.2 setosa Sepal.Length 5.1
#> 2 1.4 0.2 setosa Sepal.Width 3.5
#> 3 1.4 0.2 setosa Sepal.Length 4.9
#> 4 1.4 0.2 setosa Sepal.Width 3
#> # i 296 more rows
```

If any of the variable is missing from the data frame, that’s an error:
starwars %>% select(all_of(vars))
#> Error:
#> i In argument: `all_of(vars)`.
#> Caused by error in `all_of()` at rlang/R/eval-tidy.R:121:3:
#> ! Can't subset elements that don't exist.
#> x Elements `Sepal.Length` and `Sepal.Width` don't exist.

Use any_of() to allow missing variables:

starwars %>% select(any_of(vars))
#> # A tibble: 87 x 0

any_of() is especially useful to remove variables from a data frame because calling it again does not cause an error:

iris %>% select(-any_of(vars))
#> # A tibble: 150 x 3
#> Petal.Length Petal.Width Species
#> <dbl> <dbl> <fct>
#> 1 1.4 0.2 setosa
#> 2 1.4 0.2 setosa
#> 3 1.3 0.2 setosa
#> 4 1.5 0.2 setosa
#> # i 146 more rows

iris %>% select(-any_of(vars)) %>% select(-any_of(vars))
#> # A tibble: 150 x 3
#> Petal.Length Petal.Width Species
#> <dbl> <dbl> <fct>
#> 1 1.4 0.2 setosa
#> 2 1.4 0.2 setosa
#> 3 1.3 0.2 setosa
#> 4 1.5 0.2 setosa
#> # i 146 more rows

See Also

The selection language page, which includes links to other selection helpers.

---

**eval_relocate**

*Evaluate an expression to relocate variables*

**Description**

eval_relocate() is a variant of eval_select() that moves a selection to a new location. Either before or after can be provided to specify where to move the selection to. This powers dplyr::relocate().
Usage

eval_relocate(
    expr,
    data,
    ...,  
    before = NULL,
    after = NULL,
    strict = TRUE,
    name_spec = NULL,
    allow_rename = TRUE,
    allow_empty = TRUE,
    allow_predicates = TRUE,
    before_arg = "before",
    after_arg = "after",
    env = caller_env(),
    error_call = caller_env()
)

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.
...       These dots are for future extensions and must be empty.
before, after  Defused R code describing a selection according to the tidyselect syntax. The
              selection represents the destination of the selection provided through expr. Sup-
              plying neither of these will move the selection to the left-hand side. Supplying
              both of these is an error.
strict     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.
name_spec  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.
allow_rename     If TRUE (the default), the renaming syntax c(foo = bar) is allowed. If FALSE, it
            causes an error. This is useful to implement purely selective behaviour.
allow_empty     If TRUE (the default), it is ok for expr to result in an empty selection. If FALSE, it
            will error if expr yields an empty selection.
allow_predicates If TRUE (the default), it is ok for expr to use predicates (i.e. in where()). If
            FALSE, will error if expr uses a predicate. Will automatically be set to FALSE if
            data does not support predicates (as determined by tidyselect_data_has_predicates()).
before_arg, after_arg  Argument names for before and after. These are used in error messages.
env       The environment in which to evaluate expr. Discarded if expr is a quosure.
error_call  The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the call argument of `abort()` for more information.

Value

A named vector of numeric locations with length equal to `length(data)`. Each position in `data` will be represented exactly once.

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.

Examples

```r
library(rlang)

# Interpret defused code as a request to relocate
x <- expr(c(mpg, disp))
after <- expr(wt)
eval_relocate(x, mtcars, after = after)

# Supplying neither 'before' nor 'after' will move the selection to the left-hand side
eval_relocate(x, mtcars)

# Within a function, use 'enquo()' to defuse a single argument.
# Note that 'before' and 'after' must also be defused with 'enquo()'.
my_relocator <- function(x, expr, before = NULL, after = NULL) {
  eval_relocate(enquo(expr), x, before = enquo(before), after = enquo(after))
}
my_relocator(mtcars, vs, before = hp)

# Here is an example of using 'eval_relocate()' to implement 'relocate()'.
# Note that the dots are passed on as a defused call to 'c(...)'.
relocate <- function(.x, ..., .before = NULL, .after = NULL) {
  pos <- eval_relocate(
    expr(c(...)),
    .x,
    before = enquo(.before),
    after = enquo(.after)
  )
  set_names(.x[pos], names(pos))
}
relocate(mtcars, vs, .before = hp)
relocate(mtcars, starts_with("d"), .after = last_col())
```
eval_rename

Evaluate an expression with tidyselect semantics

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,
  allow_predicates = TRUE,
  error_call = caller_env()
)

eval_select(
  expr,
  data,
  env = caller_env(),
  ..., 
  include = NULL,
  exclude = NULL,
  strict = TRUE,
  name_spec = NULL,
  allow_rename = TRUE,
  allow_empty = TRUE,
  allow_predicates = TRUE,
  error_call = caller_env()
)

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.
env The environment in which to evaluate expr. Discarded if expr is a quosure.
... These dots are for future extensions and must be empty.
strict  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.

name_spec  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.

allow_predicates  If TRUE (the default), it is ok for expr to use predicates (i.e. in where()). If FALSE, will error if expr uses a predicate. Will automatically be set to FALSE if data does not support predicates (as determined by `tidyselect_data_has_predicates()`).

error_call  The execution environment of a currently running function, e.g. `caller_env()`.

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

allow_rename  If TRUE (the default), the renaming syntax c(foo = bar) is allowed. If FALSE, it causes an error. This is useful to implement purely selective behaviour.

allow_empty  If TRUE (the default), it is ok for expr to result in an empty selection. If FALSE, will error if expr yields an empty selection.

Details  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.

Value  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.

See Also  `https://tidyselect.r-lib.org/articles/syntax.html` or `vignette("syntax", package = "tidyselect")` for a technical description of the rules of evaluation.

Examples  

```r
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)

# Within a function, use `enquo()` to defuse one argument:
my_function <- function(x, expr) {
  eval_select(enquo(expr), x)
}

# If your function takes dots, evaluate a defused call to `c(...)`
# with `enquo(c(...))`:
my_function <- function(.x, ...) {
  eval_select(enquo(c(...)), .x)
}

# If your function takes dots and a named argument, use `{{ }}`
# inside the defused expression to tunnel it inside the tidyselect DSL:
my_function <- function(.x, .expr, ...) {
  eval_select(enquo(c({{ .expr }}, ...)), .x)
}

# Note that the trick above works because `enquo({{ arg }})` is the
# same as `enquo(arg)`.

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

---

everything Select all variables or the last variable

**Description**

These functions are selection helpers.
• `everything()` selects all variable. It is also useful in combination with other tidyselect operators.
• `last_col()` selects the last variable.

Usage

everything(vars = NULL)

last_col(offset = 0L, vars = NULL)

Arguments

vars A character vector of variable names. If not supplied, the variables are taken from the current selection context (as established by functions like `select()` or `pivot_longer()`).

offset Set it to n to select the nth var from the end.

Examples

Selection helpers can be used in functions like `dplyr::select()` or `tidyr::pivot_longer()`. Let’s first attach the tidyverse:

library(tidyverse)

# For better printing
iris <- as_tibble(iris)
mtcars <- as_tibble(mtcars)

Use `everything()` to select all variables:

iris %>% select(everything())
#> # A tibble: 150 x 5
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>          <dbl>      <dbl>       <dbl>       <dbl> <fct>
#> 1          5.1        3.5        1.4          0.2 setosa
#> 2          4.9        3           1.4          0.2 setosa
#> 3          4.7        3.2         1.3          0.2 setosa
#> 4          4.6        3.1         1.5          0.2 setosa
#> # i 146 more rows

mtcars %>% pivot_longer(everything())
#> # A tibble: 352 x 2
#>   name value
#>   <chr>  <dbl>
#> 1 mpg    21
#> 2 cyl    6
#> 3 disp   160
#> 4 hp     110
#> # i 348 more rows
Use `last_col()` to select the last variable:

```r
iris %>% select(last_col())
#> # A tibble: 150 x 1
#>   Species
#> <fct>
#> 1 setosa
#> 2 setosa
#> 3 setosa
#> 4 setosa
#> # i 146 more rows
```

```r
mtcars %>% pivot_longer(last_col())
#> # A tibble: 32 x 12
#>   mpg cyl disp hp drat wt qsec vs am gear name  value
#>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <dbl>
#> 1  21   6  160  110  3.9  2.62  16.5   0   1   4   carb     4
#> 2  21   6  160  110  3.9  2.88  17.0   0   1   4   carb     4
#> 3  22.8  4  108   93  3.85  2.32  18.6   1   1   4   carb     1
#> 4  21.4  6  258  110  3.08  3.22  19.4   1   0   3   carb     1
#> # i 28 more rows
```

Supply an offset `n` to select a variable located `n` positions from the end:

```r
mtcars %>% select(1:last_col(5))
#> # A tibble: 32 x 6
#>   mpg cyl disp hp drat wt
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1   21   6  160  110   3.9  2.62
#> 2   21   6  160  110   3.9  2.88
#> 3  22.8  4  108   93   3.85  2.32
#> 4  21.4  6  258  110   3.08  3.22
#> # i 28 more rows
```

See Also

The selection language page, which includes links to other selection helpers.

Description

**Ambiguity between columns and external variables:**

With selecting functions like `dplyr::select()` or `tidyr::pivot_longer()`, you can refer to variables by name:
mtcars %>% select(cyl, am, vs)
#> # 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
#> # i 28 more rows

mtcars %>% select(mpg:disp)
#> # 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
#> # i 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 warning 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.

vars <- c("cyl", "am", "vs")
result <- mtcars %>% select(vars)
#> Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
#> # i Please use `all_of()` or `any_of()` instead.
#> # Was:
#> data %>% select(vars)
#> # Now:
#> data %>% select(all_of(vars))
#> See
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.

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.

some_df <- mtcars[1:4, ]
some_df$vars <- 1:nrow(some_df)

These are very different objects but it isn’t a problem if the context forces you to be specific about where to find vars:
vars
#> [1] "cyl" "am" "vs"

some_df$vars
#> [1] 1 2 3 4

In a selection context however, the column wins:

some_df %>% select(vars)
#> # A tibble: 4 x 1
#> vars
#> <int>
#> 1 1
#> 2 2
#> 3 3
#> 4 4

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

some_df %>% select(all_of(vars))
#> # A tibble: 4 x 3
#> cyl  am  vs
#> <dbl> <int> <dbl>
#> 1 6 1 0
#> 2 6 1 0
#> 3 4 1 1
#> 4 6 0 1

For more information or if you have comments about this, please see the Github issue tracking the deprecation process.

Description
Functions like starts_with(), contains() or matches() are selection helpers that only work in a selection context, e.g. dplyr::select() or the cols argument of tidyr::pivot_longer().

Using a selection helper anywhere else results in an error:

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

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>
#> for details.

If you see this error, you may have used a selection helper in the wrong place, possibly as the result of a typo (e.g. misplaced comma or wrong argument name). Alternatively, you may be deliberately trying to reduce duplication in your code by extracting out a selection into a variable:

my_vars <- c(name, species, ends_with("color"))
#> Error in eval(expr, envir, enclos): object 'name' not found

To make this work you’ll need to do two things:

- Wrap the whole thing in a function
- Use any_of() or all_of() instead of bare variable names

my_vars <- function()
  c(any_of(c("name", "species")), ends_with("color"))
}
dplyr::select(starwars, my_vars())
#> # A tibble: 87 x 5
#> name species hair_color skin_color eye_color
#> <chr> <chr> <chr> <chr> <chr>
#> 1 Luke Skywalker Human blond fair blue
#> 2 C-3PO Droid <NA> gold yellow
#> 3 R2-D2 Droid <NA> white, blue red
#> 4 Darth Vader Human none white yellow
#> # i 83 more rows

table(language, Selection language)
Description

Overview of selection features:

`tidyselect` implements a DSL for selecting variables. It provides helpers for selecting variables:

- **`var1:var10`**: variables lying between `var1` on the left and `var10` on the right.
- **`starts_with("a")`**: names that start with "a".
- **`ends_with("z")`**: names that end with "z".
- **`contains("b")`**: names that contain "b".
- **`matches("x\cdot y")`**: names that match regular expression `x\cdot y`.
- **`num_range(x, 1:4)`**: names following the pattern, `x1`, `x2`, ..., `x4`.
- **`all_of(vars)/any_of(vars)`**: matches names stored in the character vector `vars`. `all_of(vars)` will error if the variables aren’t present; `any_of(var)` will match just the variables that exist.
- **`everything()`**: all variables.
- **`last_col()`**: furthest column on the right.
- **`where(is.numeric)`**: all variables where `is.numeric()` returns TRUE.

As well as operators for combining those selections:

- **`!selection`**: only variables that don’t match selection.
- **`selection1 & selection2`**: only variables included in both `selection1` and `selection2`.
- **`selection1 | selection2`**: all variables that match either `selection1` or `selection2`.

When writing code inside packages you can substitute "var" for `var` to avoid R CMD check notes.

Simple examples

Here we show the usage for the basic selection operators. See the specific help pages to learn about helpers like `starts_with()`.

The selection language can be used in functions like `dplyr::select()` or `tidyr::pivot_longer()`.

Let’s first attach the tidyverse:

```r
library(tidyverse)
# For better printing
iris <- as_tibble(iris)
```

Select variables by name:

```r
starwars %>%% select(height)
#> # A tibble: 87 x 1
#> height
#> <int>
#>  1 172
#>  2 167
#>  3  96
#>  4 202
#> # i 83 more rows
```
iris %>% pivot_longer(Sepal.Length)

#> # A tibble: 150 x 6
#> Sepal.Width Petal.Length Petal.Width Species name value
#>       <dbl>        <dbl>     <dbl> <fct>     <chr>  <dbl>
#> 1       3.5         1.4      0.2 setosa Sepal.Length 5.1
#> 2       3           1.4      0.2 setosa Sepal.Length 4.9
#> 3       3.2         1.3      0.2 setosa Sepal.Length 4.7
#> 4       3.1         1.5      0.2 setosa Sepal.Length 4.6
#> i 146 more rows

Select multiple variables by separating them with commas. Note how the order of columns is determined by the order of inputs:

starwars %>% select(homeworld, height, mass)

#> # A tibble: 87 x 3
#> homeworld height mass
#> <chr>     <int>  <dbl>
#> 1 Tatooine 172    77
#> 2 Tatooine 167    75
#> 3 Naboo     96     32
#> 4 Tatooine 202    136
#> i 83 more rows

Functions like tidyrr::pivot_longer() don't take variables with dots. In this case use c() to select multiple variables:

iris %>% pivot_longer(c(Sepal.Length, Petal.Length))

#> # A tibble: 300 x 5
#> Sepal.Width Petal.Width Species name value
#>       <dbl>     <dbl> <fct>     <chr>  <dbl>
#> 1       3.5      0.2 setosa Sepal.Length 5.1
#> 2       3.5      0.2 setosa Petal.Length 1.4
#> 3       3        0.2 setosa Sepal.Length 4.9
#> 4       3        0.2 setosa Petal.Length 1.4
#> i 296 more rows

Operators:
The : operator selects a range of consecutive variables:

starwars %>% select(name:mass)

#> # A tibble: 87 x 3
#> name      height mass
#> <chr>     <int>  <dbl>
#> 1 Luke Skywalker 172   77
#> 2 C-3PO      167   75
#> 3 R2-D2      96    32
#> 4 Darth Vader 202  136
#> i 83 more rows
The ! operator negates a selection:

```r
starwars %>% select(!(name:mass))
#> # A tibble: 87 x 11
#> hair_color skin_color eye_color birth_year sex gender homeworld species
#> <chr>     <chr>     <chr>     <dbl>   <chr>   <chr>     <chr>     <chr>
#> 1 blond    fair      blue      19     male   masculine Tatooine Human
#> 2 <NA>     gold      yellow    112 none   masculine Tatooine Droid
#> 3 <NA>     white,    blue red   33     none   masculine Naboo Droid
#> 4 none     white      yellow    41.9    male masculine Tatooine Human
#> # i 83 more rows
#> # i 3 more variables: films <list>, vehicles <list>, starships <list>
```

```r
iris %>% select(!c(Sepal.Length, Petal.Length))
#> # A tibble: 150 x 3
#> Sepal.Width Petal.Width Species
#> <dbl>    <dbl>    <fct>
#> 1 3.5       0.2    setosa
#> 2 3         0.2    setosa
#> 3 3.2       0.2    setosa
#> 4 3.1       0.2    setosa
#> # i 146 more rows
```

```r
iris %>% select(!ends_with("Width"))
#> # A tibble: 150 x 3
#> Sepal.Length Petal.Length Species
#> <dbl>    <dbl>    <fct>
#> 1 5.1      1.4    setosa
#> 2 4.9      1.4    setosa
#> 3 4.7      1.3    setosa
#> 4 4.6      1.5    setosa
#> # i 146 more rows
```

& and | take the intersection or the union of two selections:

```r
iris %>% select(starts_with("Petal") & ends_with("Width"))
#> # A tibble: 150 x 1
#> Petal.Width
#> <dbl>
#> 1 0.2
#> 2 0.2
#> 3 0.2
#> 4 0.2
#> # i 146 more rows
```

```r
iris %>% select(starts_with("Petal") | ends_with("Width"))
#> # A tibble: 150 x 3
#> Petal.Length Petal.Width Sepal.Width
#> <dbl>    <dbl>    <dbl>
#> 1 1.4      0.2      3.5
```
To take the difference between two selections, combine the & and ! operators:

```r
iris %>% select(starts_with("Petal") & !ends_with("Width"))
```

# A tibble: 150 x 1

```r
#> Petal.Length
#> <dbl>
#> 1 1.4
#> 2 1.4
#> 3 1.3
#> 4 1.5
#> # i 146 more rows
```

## Details

The order of selected columns is determined by the inputs.

- `all_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".

### 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)
```

### 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 selection helpers match variables according to a given pattern.

- `starts_with()`: Starts with an exact prefix.
- `ends_with()`: Ends with an exact suffix.
- `contains()`: Contains a literal string.
- `matches()`: Matches a regular expression.
- `num_range()`: Matches a numerical range like x01, x02, x03.

Usage

```
starts_with(match, ignore.case = TRUE, vars = NULL)
ends_with(match, ignore.case = TRUE, vars = NULL)
contains(match, ignore.case = TRUE, vars = NULL)
matches(match, ignore.case = TRUE, perl = FALSE, vars = NULL)
um_range(prefix, range, suffix = "", width = NULL, vars = NULL)
```

Arguments

- `match`: A character vector. If length > 1, the union of the matches is taken. For `starts_with()`, `ends_with()`, and `contains()` this is an exact match. For `matches()` this is a regular expression, and can be a stringr pattern.
- `ignore.case`: If TRUE, the default, ignores case when matching names.
- `vars`: A character vector of variable names. If not supplied, the variables are taken from the current selection context (as established by functions like `select()` or `pivot_longer()`).
- `perl`: Should Perl-compatible regexps be used?
- `prefix, suffix`: A prefix/suffix added before/after 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.
Examples

Selection helpers can be used in functions like `dplyr::select()` or `tidyr::pivot_longer()`. Let’s first attach the tidyverse:

```r
library(tidyverse)

# For better printing
iris <- as_tibble(iris)

starts_with() selects all variables matching a prefix and `ends_with()` matches a suffix:

```r
iris %>% select(starts_with("Sepal"))
#> # A tibble: 150 x 2
#>   Sepal.Length Sepal.Width
#>       <dbl>       <dbl>
#> 1       5.1       3.5
#> 2       4.9        3
#> 3       4.7        3.2
#> 4       4.6        3.1
#> # i 146 more rows
```

```r
iris %>% select(ends_with("Width"))
#> # A tibble: 150 x 2
#>   Sepal.Width Petal.Width
#>        <dbl>       <dbl>
#> 1       3.5       0.2
#> 2        3       0.2
#> 3       3.2       0.2
#> 4       3.1       0.2
#> # i 146 more rows
```

You can supply multiple prefixes or suffixes. Note how the order of variables depends on the order of the suffixes and prefixes:

```r
iris %>% select(starts_with(c("Petal", "Sepal")))
#> # A tibble: 150 x 4
#>   Petal.Length Petal.Width Sepal.Length Sepal.Width
#>        <dbl>       <dbl>       <dbl>       <dbl>
#> 1       1.4       0.2         5.1       3.5
#> 2       1.4       0.2         4.9        3
#> 3       1.3       0.2         4.7       3.2
#> 4       1.5       0.2         4.6       3.1
#> # i 146 more rows
```

```r
iris %>% select(ends_with(c("Width", "Length")))
#> # A tibble: 150 x 4
#>   Sepal.Width Petal.Width Sepal.Length Petal.Length
#>        <dbl>       <dbl>       <dbl>       <dbl>
#> 1       3.5       0.2         5.1       3.5
#> 2        3       0.2         4.9        3
#> 3       3.2       0.2         4.7       3.2
#> 4       3.1       0.2         4.6       3.1
#> # i 146 more rows
```
starts_with

```r
#> 1  3.5  0.2  5.1  1.4
#> 2  3   0.2  4.9  1.4
#> 3  3.2  0.2  4.7  1.3
#> 4  3.1  0.2  4.6  1.5
#> # i 146 more rows

contains() selects columns whose names contain a word:

```r
iris %>% select(contains("al"))
#> # A tibble: 150 x 4
#> Sepal.Length Sepal.Width Petal.Length Petal.Width
#> <dbl> <dbl>   <dbl>     <dbl>
#> 1  5.1  3.5   1.4       0.2
#> 2  4.9  3     1.4       0.2
#> 3  4.7  3.2   1.3       0.2
#> 4  4.6  3.1   1.5       0.2
#> # i 146 more rows
```r

starts_with(), ends_with(), and contains() do not use regular expressions. To select with a regexp use matches():

```r
# [pt] is matched literally:
iris %>% select(contains("al"))
#> # A tibble: 150 x 0
# [pt] is interpreted as a regular expression
iris %>% select(matches("[pt]al"))
#> # A tibble: 150 x 4
#> Sepal.Length Sepal.Width Petal.Length Petal.Width
#> <dbl> <dbl>   <dbl>     <dbl>
#> 1  5.1  3.5   1.4       0.2
#> 2  4.9  3     1.4       0.2
#> 3  4.7  3.2   1.3       0.2
#> 4  4.6  3.1   1.5       0.2
#> # i 146 more rows
```r

starts_with() selects all variables starting with a prefix. To select a range, use num_range(). Compare:

```r
billboard %>% select(starts_with("wk"))
#> # A tibble: 317 x 76
#> wk1 wk2 wk3 wk4 wk5 wk6 wk7 wk8 wk9 wk10 wk11 wk12 wk13
#> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1  87  82  72  77  87  94  99  NA  NA  NA  NA  NA  NA
#> 2  91  87  92  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
#> 3  81  70  68  67  66  57  54  53  51  51  51  47  47
#> 4  76  76  72  69  67  65  55  59  62  61  61  59  61
#> # i 313 more rows
```
# tidyselect_data_proxy

```
#> # i 63 more variables: wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>,
#> # wk18 <dbl>, wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, ...

billboard %>% select(num_range("wk", 10:15))
#> # A tibble: 317 x 6
#> wk10  wk11  wk12  wk13  wk14  wk15
#> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 NA    NA    NA    NA    NA    NA
#> 2 NA    NA    NA    NA    NA    NA
#> 3 51    51    51    47    44    38
#> 4 61    61    59    61    66    72
#> # i 313 more rows
```

See Also

The selection language page, which includes links to other selection helpers.

---

**Description**

- `tidyselect_data_proxy()` returns a data frame.
- `tidyselect_data_has_predicates()` returns TRUE or FALSE

If your doesn’t support predicate functions, return a 0-row data frame from `tidyselect_data_proxy()` and FALSE from `tidyselect_data_has_predicates()`.

**Usage**

```
tidyselect_data_proxy(x)

tidyselect_data_has_predicates(x)
```

**Arguments**

- `x` A data-frame like object passed to `eval_select()`, `eval_rename()`, and friends.
where

Select variables with a function

Description

This selection helper selects the variables for which a function returns TRUE.

Usage

where(fn)

Arguments

fn A function that returns TRUE or FALSE (technically, a predicate function). Can also be a purrr-like formula.

Examples

Selection helpers can be used in functions like `dplyr::select()` or `tidyr::pivot_longer()`. Let’s first attach the tidyverse:

```r
library(tidyverse)
# For better printing
iris <- as_tibble(iris)

where() takes a function and returns all variables for which the function returns TRUE:

```r
is.factor(iris[[4]])
#> [1] FALSE

is.factor(iris[[5]])
#> [1] TRUE

iris %>% select(where(is.factor))
#> # A tibble: 150 x 1
#> Species
#> <fct>
#> 1 setosa
#> 2 setosa
#> 3 setosa
#> 4 setosa
#> # i 146 more rows

is.numeric(iris[[4]])
#> [1] TRUE
```r
is.numeric(iris[[5]])
#> [1] FALSE

iris %>% select(where(is.numeric))
#> # A tibble: 150 x 4
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>       <dbl>     <dbl>       <dbl>       <dbl>
#> 1       5.1       3.5         1.4         0.2
#> 2       4.9       3.0         1.4         0.2
#> 3       4.7       3.2         1.3         0.2
#> 4       4.6       3.1         1.5         0.2
#> # i 146 more rows
```

**The formula shorthand:**

You can use purrr-like formulas as a shortcut for creating a function on the spot. These expressions are equivalent:

```r
iris %>% select(where(is.numeric))
#> # A tibble: 150 x 4
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>       <dbl>     <dbl>       <dbl>       <dbl>
#> 1       5.1       3.5         1.4         0.2
#> 2       4.9       3.0         1.4         0.2
#> 3       4.7       3.2         1.3         0.2
#> 4       4.6       3.1         1.5         0.2
#> # i 146 more rows
```

```r
iris %>% select(where(function(x) is.numeric(x)))
#> # A tibble: 150 x 4
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>       <dbl>     <dbl>       <dbl>       <dbl>
#> 1       5.1       3.5         1.4         0.2
#> 2       4.9       3.0         1.4         0.2
#> 3       4.7       3.2         1.3         0.2
#> 4       4.6       3.1         1.5         0.2
#> # i 146 more rows
```

```r
iris %>% select(where(~ is.numeric(.x)))
#> # A tibble: 150 x 4
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#>       <dbl>     <dbl>       <dbl>       <dbl>
#> 1       5.1       3.5         1.4         0.2
#> 2       4.9       3.0         1.4         0.2
#> 3       4.7       3.2         1.3         0.2
#> 4       4.6       3.1         1.5         0.2
#> # i 146 more rows
```

The shorthand is useful for adding logic inline. Here we select all numeric variables whose mean is greater than 3.5:
iris %>% select(where(~ is.numeric(.x) & mean(.x) > 3.5))
#> # A tibble: 150 x 2
#> Sepal.Length Petal.Length
#> <dbl> <dbl>
#> 1 5.1 1.4
#> 2 4.9 1.4
#> 3 4.7 1.3
#> 4 4.6 1.5
#> # i 146 more rows
Index

abort(), 6, 8
all_of, 2
all_of(), 2
all_of(vars), 15
any_of(all_of), 2
any_of(), 2
any_of(vars), 15

contains(starts_with), 19
contains(), 19
contains(b), 15

ends_with(starts_with), 19
ends_with(), 19
ends_with(z), 15
eval_relocate, 4
eval_rename, 7
eval_rename(), 18, 22
eval_select(eval_rename), 7
eval_select(), 4, 18, 22
everything, 9
everything(), 10, 15

faq-external-vector, 11
faq-selection-context, 13

language, 14
last_col(everything), 9
last_col(), 10, 15

matches(starts_with), 19
matches(), 19
matches(x.y), 15

num_range(starts_with), 19
num_range(), 19

peek_data(peek_vars), 18
peek_vars, 18

quoseure, 5, 7

select_helpers(language), 14
selection helper, 23
selection helpers, 2, 9, 18, 19
selection language, 4, 11, 22
starts_with, 19
starts_with(), 15, 19
starts_with(a), 15

tidyselect_data_has_predicates
(tidyselect_data_proxy), 22
tidyselect_data_has_predicates(), 5, 8
tidyselect_data_proxy, 22

vctrs::vec_c(), 5, 8

where, 23
where(is.numeric), 15