Package ‘tidyselect’

February 21, 2022

Title  Select from a Set of Strings

Version  1.1.2

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 (>= 1.0.1),
          vctrs (>= 0.3.0)

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

License  MIT + file LICENSE

Encoding  UTF-8

ByteCompile  true

RoxygenNote  7.1.2


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

VignetteBuilder  knitr

Config/testthat/edition  3

NeedsCompilation  yes

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

Maintainer  Lionel Henry <lionel@rstudio.com>

Repository  CRAN

Date/Publication  2022-02-21 16:00:02 UTC
**R topics documented:**

- all_of
- eval_rename
- everything
- faq-external-vector
- faq-selection-context
- language
- peek_vars
- starts_with
- 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**

```
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:

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

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

```r
vars <- c("Sepal.Length", "Sepal.Width")
```

```r
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
#> # ... with 146 more rows
```

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

```r
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
#> # ... with 146 more rows
```

```r
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
#> # ... with 296 more rows
```

If any of the variable is missing from the data frame, that’s an error:

```r
starwars %>% select(all_of(vars))
```
## Error:
## ! Can’t subset columns that don’t exist.
## x Columns ‘Sepal.Length’ and ‘Sepal.Width’ don’t exist.

Use `any_of()` to allow missing variables:

```r
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:

```r
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
#> # ... with 146 more rows
```

```r
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
#> # ... with 146 more rows
```

### See Also

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

---

**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.
eval_rename

Usage

eval_rename(
  expr,
  data,
  env = caller_env(),
  ..., 
  strict = TRUE,
  name_spec = NULL,
  error_call = caller_env()
)

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


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 `expr(c(...))`:
my_function <- function(.x, ...) {
  eval_select(expr(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(expr(c({{ .expr }}, ...))), .x)
}```
# Note that the trick above works because `expr({{ 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()`:

```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)

---

### 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**

```r
everything(vars = NULL)
```

```r
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 \( n \)th 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:

```r
library(tidyverse)

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

Use everything() to select all variables:

```r
ing %>% select(everything())
```  
```r
#> # 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
#> # ... with 146 more rows
```  
```r
mtcars %>% pivot_longer(everything())
```  
```r
#> # A tibble: 352 x 2
#> name value
#> <chr> <dbl>
#> 1 mpg 21
#> 2 cyl 6
#> 3 disp 160
#> 4 hp 110
#> # ... with 348 more rows
```  
Use last_col() to select the last variable:

```r
iris %>% select(last_col())
```  
```r
#> # A tibble: 150 x 1
#> Species
#> <fct>
#> 1 setosa
#> 2 setosa
#> 3 setosa
#> 4 setosa
#> # ... with 146 more rows
```  
```r
mtcars %>% pivot_longer(last_col())
```  
```r
#> # 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> <dbl> <chr> <dbl>
```
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)
#> # 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
```

mtcars %>% select(mpg:disp)
#> # A tibble: 32 x 3
#> mpg cyl disp
#> <dbl> <dbl> <dbl>
#> 1 21 6 160
#> 2 21 6 160
```

See Also

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

> Note: Using an external vector in selections is ambiguous.
>
> i Use `all_of(vars)` instead of `vars` to silence this message.
>
>
> This message is displayed once per session.

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[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`:

```r
vars
```

> [1] "cyl" "am" "vs"

```r
some_df$vars
```

> [1] 1 2 3 4

In a selection context however, the column wins:

```r
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:

```r
some_df %>% select(all_of(vars))
```

> # A tibble: 4 x 3
> cyl am vs
> <dbl> <dbl> <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.

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>.

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

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).

---

**Selection language**
Description

Overview of selection features::

Tidyverse selections implement a dialect of R where operators make it easy to select variables:

• : for selecting a range of consecutive variables.
• ! for taking the complement of a set of variables.
• & and | for selecting the intersection or the union of two sets of variables.
• c() for combining selections.

In addition, you can use selection helpers. Some helpers select specific columns:

• everything(): Matches all variables.
• last_col(): Select last variable, possibly with an offset.

These helpers select variables by matching patterns in 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.

These helpers select variables from a character vector:

• 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.

This helper selects variables with a function:

• where(): Applies a function to all variables and selects those for which the function returns TRUE.

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
#> # ... with 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
#> # ... with 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
#> # ... with 83 more rows

Functions like `tidyr::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 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
#> # ... with 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
#> # ... with 83 more rows

The ! operator negates a selection:

```r
starwars %>% select(!(name:mass))
```

```r
#> 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
#> # ... with 83 more rows, and 3 more variables: films <list>, vehicles <list>,
#> # starships <list>
```

```r
iris %>% select(!c(Sepal.Length, Petal.Length))
```

```r
#> 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
#> # ... with 146 more rows
```

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

```r
#> 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
#> # ... with 146 more rows
```

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

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

```r
#> A tibble: 150 x 1
#> Petal.Width
#> <dbl>
#> 1 0.2
#> 2 0.2
#> 3 0.2
#> 4 0.2
#> # ... with 146 more rows
```

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

```r
#> A tibble: 150 x 3
```
To take the difference between two selections, combine the & and ! operators:

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

### 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".

---

**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**

```
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 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`.

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, width = NULL, vars = NULL)
```

Arguments

- `match` A character vector. If length > 1, the union of the matches is taken.
- `ignore.case` 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()`).
- `vars` A prefix that starts the numeric range.
- `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.
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
ggplot(iris %>% select(starts_with("Sepal")))
```n
ggplot(iris %>% select(ends_with("Width")))

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

```r
ggplot(iris %>% select(starts_with(c("Petal", "Sepal"))))
ggplot(iris %>% select(ends_with(c("Width", "Length"))))
```n
#> 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
#> # ... with 146 more rows

contains() selects columns whose names contain a word:

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
#> # ... with 146 more rows

These helpers do not use regular expressions. To select with a regexp use matches():

# [pt] is matched literally:
iris %>% select(contains("[pt]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
#> # ... with 146 more rows

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

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 NA NA NA
#> 2 91 87 92 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
#> 3 81 70 68 67 66 57 54 53 51 51 51 51 47 47 47 47 47 47
#> 4 76 76 72 69 67 65 55 59 62 61 61 59 61 61 61 61 61 61
#> # ... with 313 more rows, and 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
#> # ... with 313 more rows

See Also

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

---

**where**  
*Select variables with a function*

**Description**

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

**Usage**

```r
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:

```
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
#> # ... with 146 more rows

is.numeric(iris[[4]])
#> [1] TRUE

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. 1.4 0.2
#> 3 4.7 3.2 1.3 0.2
#> 4 4.6 3.1 1.5 0.2
#> # ... with 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:

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. 1.4 0.2
#> 3 4.7 3.2 1.3 0.2
#> 4 4.6 3.1 1.5 0.2
#> # ... with 146 more rows

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. 1.4 0.2
#> 3 4.7 3.2 1.3 0.2
#> 4 4.6 3.1 1.5 0.2
#> # ... with 146 more rows
where

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         1.4       0.2
#> 3 4.7     3.2       1.3       0.2
#> 4 4.6     3.1       1.5       0.2
#> # ... with 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
#> # ... with 146 more rows
Index

abort(), 5
all_of(), 2
all_of(), 2, 12
any_of (all_of), 2
any_of(), 2, 12
contains (starts_with), 16
contains(), 12, 16
ends_with (starts_with), 16
ends_with(), 12, 16
eval_rename(), 4
eval_rename(), 15
eval_select (eval_rename), 4
eval_select(), 15
everything(), 7
everything(), 7, 12
faq-external-vector, 9
faq-selection-context, 11
language, 11
last_col (everything), 7
last_col(), 7, 12
matches (starts_with), 16
matches(), 12, 16
num_range (starts_with), 16
num_range(), 12, 16
peek_data (peek_vars), 15
peek_vars, 15
quosure, 5
select_helpers (language), 11
selection helper, 19
selection helpers, 2, 7, 15, 16
selection language, 4, 9, 19
starts_with, 16
starts_with(), 12, 16
vctrs::vec_c(), 5
where, 19
where(), 12