Package 'tidyterra'

November 6, 2026

Title 'tidyverse' Methods and 'ggplot2' Helpers for 'terra' Objects

Version 1.3.1

Description Provides methods from 'tidyverse' packages for 'SpatRaster' and 'SpatVector' objects created with 'terra', plus 'ggplot2' 'geoms' and scales for plotting those objects. It is designed to let users manipulate spatial data with familiar 'dplyr' and 'tidyr' verbs before visualizing results with 'ggplot2'.

License MIT + file LICENSE

URL https://dieghernan.github.io/tidyterra/, https://github.com/dieghernan/tidyterra

BugReports https://github.com/dieghernan/tidyterra/issues

Depends R (>= 4.1.0)

Imports cli (>= 3.0.0), data.table, dplyr (>= 1.2.0), generics, ggplot2 (>= 4.0.0), grDevices, isoband, lifecycle, magrittr, rlang (>= 1.2.0), scales, sf (>= 1.0.0), terra (>= 1.8-10), tibble (>= 3.0.0), tidyr (>= 1.0.0), tools, utils, vctrs

Suggests dtplyr, hexbin, knitr, maptiles, quarto, s2, stringi, testthat (>= 3.0.0), withr

VignetteBuilder quarto

Config/Needs/coverage covr

Config/Needs/website geodata, dieghernan/gitdevr, ragg, styler, metR, ggspatial, cpp11, remotes, gganimate, gifski, tidyverse, bibtex, ggblend

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

X-schema.org-keywords r, terra, ggplot-extension, r-spatial, rspatial, cran, cran-r, r-package, rstats, rstats-package

NeedsCompilation no

Author Diego Hernangómez [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-8457-4658>), Dewey Dunnington [ctb] (ORCID: <https://orcid.org/0000-0002-9415-4582>, for ggspatial code), ggplot2 authors [cph] (for contour code), Andrea Manica [ctb] (ORCID: <https://orcid.org/0000-0003-1895-450X>), Michael Sumner [ctb] (ORCID: <https://orcid.org/0000-0002-2471-7511>)

Maintainer Diego Hernangómez <diego.hernangomezherrero@gmail.com>

Repository CRAN

Date/Publication 2026-11-06 17:50:02 UTC

Contents

arrange.SpatVector . . . 3
as_coordinates . . . 5
as_sf . . . 6
as_spatraster . . . 7
as_spatvector . . . 8
as_tibble.Spat . . . 10
autoplot.Spat . . . 13
bind_cols.SpatVector . . . 15
bind_rows.SpatVector . . . 16
compare_spatrasters . . . 18
complete.SpatVector . . . 19
count.SpatVector . . . 21
cross_blended_hypsometric_tints_db . . . 23
cross_join.SpatVector . . . 24
distinct.SpatVector . . . 26
drop_na.Spat . . . 27
expand.SpatVector . . . 30
fill.SpatVector . . . 31
filter-joins.SpatVector . . . 32
filter.Spat . . . 35
fortify.Spat . . . 37
geom_spatraster . . . 40
geom_spatraster_rgb . . . 45
geom_spat_contour . . . 48
ggspatvector . . . 54
glance.Spat . . . 57
glimpse.Spat . . . 58
grass_db . . . 59
group_by.SpatVector . . . 62
hypsometric_tints_db . . . 64

is_regular_grid . . . 66
mutate-joins.SpatVector . . . 67
mutate.Spat . . . 70
nest.SpatVector . . . 72
nest_join.SpatVector . . . 74
pivot_longer.SpatVector . . . 76
pivot_wider.SpatVector . . . 79
princess_db . . . 82
pull.Spat . . . 83
pull_crs . . . 85
reframe.SpatVector . . . 87
relocate.Spat . . . 88
rename.Spat . . . 89
replace_na.Spat . . . 91
required_pkgs.Spat . . . 92
rows.SpatVector . . . 93
rowwise.SpatVector . . . 96
scale_coltab . . . 98
scale_cross_blended . . . 100
scale_grass . . . 108
scale_hypso . . . 115
scale_princess . . . 123
scale_terrain . . . 127
scale_whitebox . . . 131
scale_wiki . . . 135
select.Spat . . . 139
slice.Spat . . . 141
summarise.SpatVector . . . 146
tidy.Spat . . . 148
uncount.SpatVector . . . 151
unite.Spat . . . 152
volcano2 . . . 154

Index **156**

`arrange.SpatVector` *Order a* `SpatVector` *using column values*

Description

`arrange.SpatVector()` orders the geometries of a `SpatVector` by the values of selected columns.

Usage

```
## S3 method for class 'SpatVector'
arrange(.data, ..., .by_group = FALSE, .locale = NULL)
```

Arguments

.data	A SpatVector created with terra::vect().
...	<data-masking> Variables, or functions of variables. Use desc() to sort a variable in descending order.
.by_group	If TRUE, sort first by grouping variable. This applies to grouped SpatVector objects only.
.locale	The locale to sort character vectors in. • If NULL, the default, uses the "C" locale unless the deprecated dplyr.legacy_locale global option escape hatch is active. See the dplyr-locale help page for more details. • If a single string from stringi::stri_locale_list() is supplied, then this will be used as the locale to sort with. For example, "en" will sort with the American English locale. This requires the stringi package. • If "C" is supplied, then character vectors will always be sorted in the C locale. This does not require stringi and is often much faster than supplying a locale identifier. The C locale is not the same as English locales, such as "en", particularly when it comes to data containing a mix of upper and lower case letters. This is explained in more detail on the locale help page under the Default locale section.

Value

A SpatVector object.

terra equivalent

terra::sort().

Methods

Implementation of the **generic** dplyr::arrange() method for SpatVector objects.

See Also

dplyr::arrange().

Other **dplyr** verbs that operate on rows: distinct.SpatVector(), filter.Spat, rows.SpatVector, slice.Spat

Examples

```
library(terra)
library(dplyr)

v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

# Single variable
```

```
v |>
  arrange(desc(iso2))

# Two variables
v |>
  mutate(even = as.double(cpro) %% 2 == 0) |>
  arrange(desc(even), desc(iso2))

# With new variables
v |>
  mutate(area_geom = terra::expanse(v)) |>
  arrange(area_geom)
```

`as_coordinates` *Get cell number, row and column from a* `SpatRaster`

Description

`as_coordinates()` returns the position of each cell in the `SpatRaster` matrix.

Usage

```
as_coordinates(x, as.raster = FALSE)
```

Arguments

`x`	A `SpatRaster` object.
`as.raster`	If `TRUE`, the result is a `SpatRaster` object with three layers indicating the position of each cell (cell number, row and column).

Value

A tibble or a `SpatRaster` (if `as.raster = TRUE`) with one row (or cell) for each cell in `x`.

When `as.raster = TRUE`, the resulting `SpatRaster` has the same CRS, extent and resolution as `x`.

See Also

`slice.SpatRaster()`.

Coercing objects: `as_sf()`, `as_spatraster()`, `as_spatvector()`, `as_tibble.Spat`, `fortify.Spat`, `tidy.Spat`

Examples

```
library(terra)

f <- system.file("extdata/cyl_temp.tif", package = "tidyterra")

r <- rast(f)

as_coordinates(r)
as_coordinates(r, as.raster = TRUE)

as_coordinates(r, as.raster = TRUE) |> plot()
```

---

as_sf *Coerce a SpatVector to an sf object*

Description

`as_sf()` coerces a `SpatVector` into an `sf` object. It wraps `sf::st_as_sf()` and preserves groups created with `group_by.SpatVector()`.

Usage

```
as_sf(x, ...)
```

Arguments

`x`	A `SpatVector` created with `terra::vect()`.
`...`	Additional arguments passed to `sf::st_as_sf()`.

Value

An `sf` object with an additional `tbl_df` class for pretty printing.

See Also

Coercing objects: `as_coordinates()`, `as_spatraster()`, `as_spatvector()`, `as_tibble.Spat`, `fortify.Spat`, `tidy.Spat`

Examples

```
library(terra)

f <- system.file("extdata/cyl.gpkg", package = "tidyterra")
v <- terra::vect(f)

# The input is ungrouped.
v
```

```
is_grouped_spatvector(v)

# Get ungrouped data.
a_sf <- as_sf(v)

dplyr::is_grouped_df(a_sf)

# Grouped

v$gr <- c("C", "A", "A", "B", "A", "B", "B")
v$gr2 <- rep(c("F", "G", "F"), 3)

gr_v <- group_by(v, gr, gr2)

gr_v
is_grouped_spatvector(gr_v)

group_data(gr_v)

# An sf object.

a_gr_sf <- as_sf(gr_v)

dplyr::is_grouped_df(a_gr_sf)

group_data(a_gr_sf)
```

`as_spatraster` *Coerce a data frame to* `SpatRaster`

Description

`as_spatraster()` converts a data frame or tibble into a `SpatRaster`. It wraps the `terra::rast()` S4 method for signature `data.frame`.

Usage

```
as_spatraster(x, ..., xycols = 1:2, crs = "", digits = 6)
```

Arguments

`x`	A tibble or data frame.
`...`	Additional arguments passed to `terra::rast()`.
`xycols`	A vector of integers of length 2 determining the position of the columns that hold the x and y coordinates.
`crs`	A CRS in several formats (PROJ.4, WKT, EPSG code, etc.) or a spatial object from **sf** or **terra** that includes the target coordinate reference system. See `pull_crs()` and **Details**.

digits — Integer to set the precision for detecting whether points are on a regular grid (a low number of digits is a low precision).

Details

If no crs is provided and the tibble was created with as_tibble.SpatRaster(), the crs is inferred from attr(x, "crs").

Value

A SpatRaster object.

terra equivalent

terra::rast() (see S4 method for signature data.frame).

See Also

is_regular_grid() for checking whether coordinates form a regular grid. pull_crs() retrieves CRS information, as do sf::st_crs() and terra::crs().

Coercing objects: as_coordinates(), as_sf(), as_spatvector(), as_tibble.Spat, fortify.Spat, tidy.Spat

Examples

```
library(terra)

r <- rast(matrix(1:90, ncol = 3), crs = "EPSG:3857")

r

# Create tibble
as_tbl <- as_tibble(r, xy = TRUE)

as_tbl

# From tibble
newrast <- as_spatraster(as_tbl, crs = "EPSG:3857")
newrast
```

---

as_spatvector *Coerce objects to SpatVector*

Description

as_spatvector() turns an existing object into a SpatVector. It wraps the terra::vect() S4 method for the data.frame signature.

Usage

```
as_spatvector(x, ...)

## S3 method for class 'data.frame'
as_spatvector(x, ..., geom = c("lon", "lat"), crs = "")

## S3 method for class 'sf'
as_spatvector(x, ...)

## S3 method for class 'sfc'
as_spatvector(x, ...)

## S3 method for class 'SpatVector'
as_spatvector(x, ...)
```

Arguments

x	A tibble, data frame or **sf** object of class `sf` or `sfc`.
...	Additional arguments passed to `terra::vect()`.
geom	Character vector naming the fields that contain the geometry data. Use two names for point coordinates (`x` and `y`) or one name for a column with WKT geometries.
crs	A CRS in several formats (PROJ.4, WKT, EPSG code, etc.) or a spatial object from **sf** or **terra** that includes the target coordinate reference system. See `pull_crs()` and **Details**.

Details

This function differs from `terra::vect()` in the following ways:

- Rows with geometry values `NA` or `""` are removed before conversion.
- If `x` is a grouped data frame (see `dplyr::group_by()`), the grouping variables are transferred and a grouped `SpatVector` is created (see `group_by.SpatVector()`).
- If no `crs` is provided and the tibble was created with `as_tibble.SpatVector()`, the `crs` is inferred from `attr(x, "crs")`.
- It handles the conversion of `EMPTY` geometries between **sf** and **terra**.

Value

A `SpatVector` object.

terra equivalent

`terra::vect()`.

See Also

pull_crs() for retrieving CRS and the corresponding utilities sf::st_crs() and terra::crs().

Coercing objects: as_coordinates(), as_sf(), as_spatraster(), as_tibble.Spat, fortify.Spat, tidy.Spat

Examples

```
library(terra)

v <- vect(matrix(1:80, ncol = 2), crs = "EPSG:3857")

v$cat <- sample(LETTERS[1:4], size = nrow(v), replace = TRUE)

v

# Create tibble
as_tbl <- as_tibble(v, geom = "WKT")

as_tbl

# From tibble
newvect <- as_spatvector(as_tbl, geom = "geometry", crs = "EPSG:3857")
newvect
```

---

as_tibble.Spat *Coerce SpatRaster and SpatVector objects to tibbles*

Description

as_tibble() methods for SpatRaster and SpatVector objects.

Usage

```
## S3 method for class 'SpatRaster'
as_tibble(
  x,
  ...,
  xy = FALSE,
  na.rm = FALSE,
  .name_repair = c("unique", "check_unique", "universal", "minimal", "unique_quiet",
    "universal_quiet")
)

## S3 method for class 'SpatVector'
as_tibble(
  x,
  ...,
```

```
  geom = NULL,
  .name_repair = c("unique", "check_unique", "universal", "minimal", "unique_quiet",
    "universal_quiet")
)
```

Arguments

x	A SpatRaster created with terra::rast() or a SpatVector created with terra::vect().
...	Arguments passed to terra::as.data.frame().
xy	logical. If TRUE, the coordinates of each raster cell are included
na.rm	logical. If TRUE, cells that have a NA value in at least one layer are removed. If the argument is set to NA only cells that have NA values in all layers are removed
.name_repair	Treatment of problematic column names: • "minimal": No name repair or checks, beyond basic existence, • "unique": Make sure names are unique and not empty, • "check_unique": (default value), no name repair, but check they are unique, • "universal": Make the names unique and syntactic • "unique_quiet": Same as "unique", but "quiet" • "universal_quiet": Same as "universal", but "quiet" • a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R). • A purrr-style anonymous function, see rlang::as_function() This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.
geom	character or NULL. If not NULL, either "WKT" or "HEX", to get the geometry included in Well-Known-Text or hexadecimal notation. If x has point geometry, it can also be "XY" to add the coordinates of each point

Value

A tibble.

terra equivalent

terra::as.data.frame().

Methods

Implementation of the **generic** tibble::as_tibble() methods for Spat* objects.

SpatRaster and SpatVector:

The returned tibble includes the CRS of the original object as an attribute in WKT format (see pull_crs()).

Layer and column names

When coercing SpatRaster objects to data frames, x and y are reserved names for the geographic coordinates of each cell. **terra** also allows layers with duplicated names.

When coercing a SpatRaster to a tibble, **tidyterra** may rename its layers to avoid these issues. Specifically, layers may be renamed in the following cases:

- Layers with duplicated names.
- When coercing to a tibble, if xy = TRUE, layers named x or y are renamed.
- When working with methods from tidyverse packages, for example filter.SpatRaster(), the same renaming happens.

tidyterra displays a message describing the renamed layers.

The same issue affects SpatVector objects with reserved names such as geometry (when geom = c("WKT", "HEX")) and x, y (when geom = "XY"). These names represent geometry columns in terra::as.data.frame(). If geom is not NULL, the same renaming logic described for SpatRaster also applies to SpatVector columns.

See Also

tibble::as_tibble(), terra::as.data.frame().

Coercing objects: as_coordinates(), as_sf(), as_spatraster(), as_spatvector(), fortify.Spat, tidy.Spat

Examples

```
library(terra)
# SpatRaster
f <- system.file("extdata/cyl_temp.tif", package = "tidyterra")
r <- rast(f)

as_tibble(r, na.rm = TRUE)

as_tibble(r, xy = TRUE)

# SpatVector

f <- system.file("extdata/cyl.gpkg", package = "tidyterra")
v <- vect(f)

as_tibble(v)
```

`autoplot.Spat` *Create a complete ggplot for* `Spat*` *objects*

Description

`autoplot()` uses **ggplot2** to draw plots like those produced by `terra::plot()`/`terra::plotRGB()` in a single command.

Usage

```
## S3 method for class 'SpatRaster'
autoplot(
  object,
  ...,
  rgb = NULL,
  use_coltab = NULL,
  facets = NULL,
  nrow = NULL,
  ncol = 2
)

## S3 method for class 'SpatVector'
autoplot(object, ...)

## S3 method for class 'SpatGraticule'
autoplot(object, ...)

## S3 method for class 'SpatExtent'
autoplot(object, ...)
```

Arguments

`object`	A `SpatRaster` created with `terra::rast()`, a `SpatVector` created with `terra::vect()`, a `SpatGraticule` (see `terra::graticule()`) or a `SpatExtent` (see `terra::ext()`).
`...`	Other arguments passed to `geom_spatraster()`, `geom_spatraster_rgb()` or `geom_spatvector()`.
`rgb`	Logical. If `TRUE`, plot as an RGB image. If `NULL` (the default), `autoplot.SpatRaster()` tries to guess.
`use_coltab`	Logical. If `TRUE`, plot with the corresponding color table from `terra::coltab()`. If `NULL` (the default), `autoplot.SpatRaster()` tries to guess. See also `scale_fill_coltab()`.
`facets`	Logical. If `TRUE`, display facets. If `NULL` (the default), `autoplot.SpatRaster()` tries to guess.
`nrow`, `ncol`	Number of rows and columns in the facet.

Value

A **ggplot2** plot.

Methods

Implementation of the **generic** ggplot2::autoplot() methods for Spat* objects.

SpatRaster**:**

Uses geom_spatraster() or geom_spatraster_rgb().

SpatVector**,** SpatGraticule **and** SpatExtent**:**

Uses geom_spatvector(). Labels can be placed with geom_spatvector_text() or geom_spatvector_label().

See Also

ggplot2::autoplot().

Other **ggplot2** helpers: fortify.Spat, geom_spat_contour, geom_spatraster(), geom_spatraster_rgb(), ggspatvector, stat_spat_coordinates()

Other **ggplot2** methods: fortify.Spat

Examples

```
file_path <- system.file("extdata/cyl_temp.tif", package = "tidyterra")

library(terra)
temp <- rast(file_path)

library(ggplot2)
autoplot(temp)

# With a tile

tile <- system.file("extdata/cyl_tile.tif", package = "tidyterra") |>
  rast()

autoplot(tile)

# With color tables

ctab <- system.file("extdata/cyl_era.tif", package = "tidyterra") |>
  rast()

autoplot(ctab)

# With vectors
v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
autoplot(v)

v |> autoplot(aes(fill = cpro)) +
  geom_spatvector_text(aes(label = iso2)) +
  coord_sf(crs = 25829)
```

`bind_cols.SpatVector` *Bind multiple* `SpatVector`*,* `sf` *and data frame objects by column*

Description

Bind any number of `SpatVector` objects, data frames and `sf` objects by column, making a wider result. This is similar to `do.call(cbind, data_frames)`.

Where possible, prefer using a join to combine `SpatVector` and data frame objects. `bind_spat_cols()` binds rows in the order in which they appear, so it is easy to create meaningless results without realizing it.

Usage

```
bind_spat_cols(
  ...,
  .name_repair = c("unique", "universal", "check_unique", "minimal")
)
```

Arguments

`...`	Objects to combine. The first argument must be a `SpatVector`. Each subsequent argument can be a `SpatVector`, `sf` object or data frame. Inputs are recycled to the same length, then matched by position.
`.name_repair`	One of `"unique"`, `"universal"`, or `"check_unique"`. See `vctrs::vec_as_names()` for the meaning of these options.

Value

A `SpatVector` object with the combined columns. Its geometry and CRS come from the first `SpatVector` in `...`.

terra equivalent

`cbind()` method.

Methods

Implementation of the `dplyr::bind_cols()` function for `SpatVector` objects. For the second and subsequent arguments in `...`, the geometry is not cbinded and only the data frame-like columns are kept.

See Also

`dplyr::bind_cols()`.

Other **dplyr** verbs that operate on pairs of `SpatVector` and data frame objects: `bind_rows.SpatVector`, `cross_join.SpatVector()`, `filter-joins.SpatVector`, `mutate-joins.SpatVector`, `nest_join.SpatVector()`, `rows.SpatVector`

Examples

```
library(terra)
sv <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
df2 <- data.frame(letters = letters[seq_len(nrow(sv))])

# Data frame
bind_spat_cols(sv, df2)

# Another SpatVector
bind_spat_cols(sv[1:2, ], sv[3:4, ])

# sf objects
sfobj <- sf::read_sf(system.file("shape/nc.shp", package = "sf"))

bind_spat_cols(sv[1:9, ], sfobj[1:9, ])

# Mixed

end <- bind_spat_cols(sv, sfobj[seq_len(nrow(sv)), 1:2], df2)

end
glimpse(end)

# Row sizes must be compatible when column-binding.
try(bind_spat_cols(sv, sfobj))
```

---

bind_rows.SpatVector *Bind multiple* SpatVector, sf, sfc *and data frame objects by row*

Description

Bind any number of SpatVector objects, data frames, sf and sfc objects by row, making a longer result. This is similar to do.call(rbind, data_frames), but the output will contain all columns that appear in any of the inputs.

Usage

```
bind_spat_rows(..., .id = NULL)
```

Arguments

...	Objects to combine. The first argument must be a SpatVector. Each subsequent argument can be a SpatVector, sf or sfc object or a data frame. Columns are matched by name and any missing columns are filled with NA.
.id	The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.

Value

A SpatVector object of the same type as the first element of ....

terra equivalent

rbind() method.

Methods

Implementation of the dplyr::bind_rows() function for SpatVector objects.

The first argument should be a SpatVector. Each subsequent argument can be a SpatVector, an sf or sfc object or a data frame:

- If subsequent spatial objects have a different CRS from the first element, they are reprojected to the CRS of the first element with a message.
- If any element of ... is a tibble/data frame, the rows are column-bound with empty geometries with a message.

To create an input in another CRS, use terra::project() to transform its coordinates. Assigning terra::crs(x) <- "EPSG:4326" changes only the CRS metadata. It does not transform coordinates and must only be used when that CRS is known to describe the existing coordinates.

See Also

dplyr::bind_rows().

Other dplyr verbs that operate on pairs of SpatVector and data frame objects: bind_cols.SpatVector, cross_join.SpatVector(), filter-joins.SpatVector, mutate-joins.SpatVector, nest_join.SpatVector(), rows.SpatVector

Examples

```
library(terra)
v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

v1 <- v[1, "cpro"]
v2 <- v[3:5, c("name", "iso2")]

# You can supply individual SpatVector objects as arguments.
bind_spat_rows(v1, v2)

# When you supply a column name with the `.id` argument, a new column is
# created to link each row to its original data frame.
bind_spat_rows(v1, v2, .id = "id")


# Use with sf objects.
sfobj <- sf::st_as_sf(v2[1, ])

sfobj
```

```
bind_spat_rows(v1, sfobj)

# Reproject with a message when the CRS differs.
sfobj_3857 <- as_spatvector(sfobj) |> project("EPSG:3857")

bind_spat_rows(v1, sfobj_3857)

# Bind data frames with a message.
data("mtcars")
bind_spat_rows(v1, sfobj, mtcars, .id = "id2")

# Use lists
bind_spat_rows(list(v1[1, ], sfobj[1:2, ]))

# Use a named list with `.id`.
bind_spat_rows(list(
  SpatVector = v1[1, ], sf = sfobj[1, ],
  mtcars = mtcars[1, ]
), .id = "source")
```

`compare_spatrasters` *Compare attributes of two* `SpatRaster` *objects*

Description

Two `SpatRaster` objects are compatible (in terms of combining layers) if the CRS, extent and resolution are similar. In those cases you can combine the objects simply as `c(x, y)`.

This function compares those attributes and reports the results. See **Resolving differences** for minimal guidance.

Usage

```
compare_spatrasters(x, y, digits = 6)
```

Arguments

`x, y`	`SpatRaster` objects.
`digits`	Integer to set the precision for comparing the extent and the resolution.

Value

An invisible logical value indicating whether the `SpatRaster` objects are compatible, plus an informative message flagging any issues found.

terra equivalent

`terra::compareGeom()`.

Resolving differences

- For a **different CRS**, try `terra::project()`.
- For a **different extent**, try `terra::resample()`.
- For a **different resolution**, try `terra::resample()`, `terra::aggregate()` or `terra::disagg()`.

See Also

`terra::identical()`.

Examples

```
library(terra)

x <- rast(matrix(1:90, ncol = 3), crs = "EPSG:3857")

# Nothing
compare_spatrasters(x, x)

# Different CRS
y_nocrs <- x
crs(y_nocrs) <- NA

compare_spatrasters(x, y_nocrs)

# Different extent
compare_spatrasters(x, x[1:10, , drop = FALSE])

# Different resolution
y_newres <- x

res(y_newres) <- res(x) / 2
compare_spatrasters(x, y_newres)

# Everything

compare_spatrasters(x, project(x, "epsg:3035"))
```

---

complete.SpatVector *Complete missing combinations in a SpatVector*

Description

`complete()` turns implicit missing combinations in a `SpatVector` into explicit rows while preserving geometry and spatial metadata.

Usage

```
## S3 method for class 'SpatVector'
complete(data, ..., fill = list(), explicit = TRUE)
```

Arguments

data	A SpatVector.
...	<data-masking> Specification of columns to expand or complete. Columns can be atomic vectors or lists. • To find all unique combinations of x, y and z, including those not present in the data, supply each variable as a separate argument: expand(df, x, y, z) or complete(df, x, y, z). • To find only the combinations that occur in the data, use nesting: expand(df, nesting(x, y, z)). • You can combine the two forms. For example, expand(df, nesting(school_id, student_id), date) would produce a row for each present school-student combination for all possible dates. When used with factors, expand() and complete() use the full set of levels, not just those that appear in the data. If you want to use only the values seen in the data, use forcats::fct_drop(). When used with continuous variables, you may need to fill in values that do not appear in the data: to do so use expressions like year = 2010:2020 or year = full_seq(year,1).
fill	A named list that for each variable supplies a single value to use instead of NA for missing combinations.
explicit	Should both implicit (newly created) and explicit (pre-existing) missing values be filled by fill? By default, this is TRUE, but if set to FALSE this will limit the fill to only implicit missing values.

Value

A SpatVector object.

Methods

Implementation of the **generic** tidyr::complete() method for SpatVector objects.

complete() preserves the geometry column while expanding missing combinations. New combinations receive empty geometries.

See Also

tidyr::complete().

Other **tidyr** verbs for handling missing values: drop_na.Spat, expand.SpatVector(), fill.SpatVector(), replace_na.Spat

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
v <- dplyr::mutate(v, grp = ifelse(iso2 %in% c("ES-AV", "ES-BU"), "a", "b"))

complete(v, grp, tidyr::nesting(iso2, name)) |>
  glimpse()
```

count.SpatVector *Count the observations in each* SpatVector *group*

Description

count() lets you quickly count the unique values of one or more variables: df |> count(a, b) is roughly equivalent to df |> group_by(a, b) |> summarise(n = n()). count() is paired with tally(), a lower-level helper that is equivalent to df |> summarise(n = n()). Supply wt to perform weighted counts, switching the summary from n = n() to n = sum(wt).

add_count() is equivalent to count() but uses mutate() instead of summarise() so that it adds a new column with group-wise counts.

Usage

```
## S3 method for class 'SpatVector'
count(
  x,
  ...,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  .drop = deprecated(),
  .dissolve = TRUE
)

## S3 method for class 'SpatVector'
tally(x, wt = NULL, sort = FALSE, name = NULL)

## S3 method for class 'SpatVector'
add_count(x, ..., wt = NULL, sort = FALSE, name = NULL, .drop = deprecated())
```

Arguments

x	A SpatVector created with terra::vect().
...	<data-masking> Variables to group by.
wt	<data-masking> Frequency weights. Can be NULL or a variable: • If NULL (the default), counts the number of rows in each group. • If a variable, computes sum(wt) for each group.
sort	If TRUE, will show the largest groups at the top.
name	The name of the new column in the output. If omitted, it will default to n. If there's already a column called n, it will use nn. If there's a column called n and nn, it'll use nnn, and so on, adding ns until it gets a new name.
.drop	**[Deprecated]** Argument no longer supported, empty groups are always removed (see dplyr::count(), .drop = TRUE argument).
.dissolve	Logical. If TRUE, dissolve borders between aggregated geometries.

Value

A SpatVector object with updated grouping metadata.

terra equivalent

terra::aggregate().

Methods

Implementation of the **generic** dplyr::count() method family for SpatVector objects.

tally() will always return a disaggregated geometry while count() can handle this. See also summarise.SpatVector().

See Also

dplyr::count(), dplyr::tally().

Other **dplyr** verbs that operate on groups of rows: group_by.SpatVector(), reframe.SpatVector(), rowwise.SpatVector(), summarise.SpatVector()

Examples

```
library(terra)
f <- system.file("ex/lux.shp", package = "terra")
p <- vect(f)

p |> count(NAME_1, sort = TRUE)

p |> count(pop = ifelse(POP < 20000, "A", "B"))

# tally() is a lower-level function that assumes grouping is already done.
p |> tally()

p |>
  group_by(NAME_1) |>
  tally()

# Dissolve geometries by default

library(ggplot2)
p |>
  count(NAME_1) |>
  ggplot() +
  geom_spatvector(aes(fill = n))

# Opt out
p |>
  count(NAME_1, .dissolve = FALSE, sort = TRUE) |>
  ggplot() +
  geom_spatvector(aes(fill = n))
```

`cross_blended_hypsometric_tints_db` *Cross-blended hypsometric tints*

Description

A tibble including the color map of 4 gradient palettes. All palettes also include a definition of color limits in terms of elevation (meters) that can be used with `ggplot2::scale_fill_gradientn()`.

Format

A tibble of 41 rows and 6 columns with the following fields:

pal Name of the palette.

limit Recommended elevation limit (in meters) for each color.

r Value of the red channel (RGB color mode).

g Value of the green channel (RGB color mode).

b Value of the blue channel (RGB color mode).

hex Hex code of the color.

Details

From Patterson & Jenny (2011):

More recently, the role and design of hypsometric tints have come under scrutiny. One reason for this is the concern that people misread elevation colors as climate or vegetation information. Cross-blended hypsometric tints, introduced in 2009, are a partial solution to this problem. They use variable lowland colors customized to match the differing natural environments of world regions, which merge into one another.

Source

Derived from:

- Patterson, T., & Jenny, B. (2011). The Development and Rationale of Cross-blended Hypsometric Tints. *Cartographic Perspectives,* (69), 31-46. doi:10.14714/CP69.20.

See Also

`scale_fill_cross_blended_c()`.

Other datasets: `grass_db`, `hypsometric_tints_db`, `princess_db`, `volcano2`

Examples

```
data("cross_blended_hypsometric_tints_db")

cross_blended_hypsometric_tints_db

# Select a palette
warm <- cross_blended_hypsometric_tints_db |>
  filter(pal == "warm_humid")

f <- system.file("extdata/asia.tif", package = "tidyterra")
r <- terra::rast(f)

library(ggplot2)

p <- ggplot() +
  geom_spatraster(data = r) +
  labs(fill = "elevation")

p +
  scale_fill_gradientn(colors = warm$hex)

# Use with limits
p +
  scale_fill_gradientn(
    colors = warm$hex,
    values = scales::rescale(warm$limit),
    limit = range(warm$limit),
    na.value = "lightblue"
  )
```

---

cross_join.SpatVector *Cross joins for* SpatVector *objects*

Description

[Experimental]

Cross joins match each row in `x` to every row in `y`.

See `dplyr::cross_join()` for details.

Usage

```
## S3 method for class 'SpatVector'
cross_join(x, y, ..., copy = FALSE, suffix = c(".x", ".y"))
```

Arguments

x	A SpatVector created with terra::vect().
y	A data frame or other object coercible to a data frame. If a SpatVector or sf object is provided, this method returns an error.
...	Additional arguments passed to sf::st_as_sf().
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.

Value

A SpatVector object.

Methods

Implementation of the **generic** dplyr::cross_join() method for SpatVector objects.

The geometry column has sticky behavior. The result repeats each geometry in x once for every row in y.

If y has a column named geometry, it is treated as a regular attribute and receives the suffix from suffix.

See Also

dplyr::cross_join().

Other **dplyr** verbs that operate on pairs of SpatVector and data frame objects: bind_cols.SpatVector, bind_rows.SpatVector, filter-joins.SpatVector, mutate-joins.SpatVector, nest_join.SpatVector(), rows.SpatVector

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

labels <- data.frame(period = c("past", "present"))

cross_join(v, labels)
```

`distinct.SpatVector` *Keep distinct/unique rows and geometries of* `SpatVector` *objects*

Description

Keep only unique/distinct rows and geometries from a `SpatVector`.

Usage

```
## S3 method for class 'SpatVector'
distinct(.data, ..., .keep_all = FALSE)
```

Arguments

`.data`	A `SpatVector` created with `terra::vect()`.
`...`	<`data-masking`> Optional variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, all variables in the data frame are used. There is a reserved variable name, `geometry`, that removes duplicate geometries. See **Methods**.
`.keep_all`	If `TRUE`, keep all variables in `.data`. If a combination of `...` is not distinct, this keeps the first row of values.

Value

A `SpatVector` object.

terra equivalent

`terra::unique()`.

Methods

Implementation of the **generic** `dplyr::distinct()` method for `SpatVector` objects.

You can remove duplicate geometries by passing the reserved name `geometry` to `...`. See **Examples**.

See Also

`dplyr::distinct()`, `terra::unique()`.

Other **dplyr** verbs that operate on rows: `arrange.SpatVector()`, `filter.Spat`, `rows.SpatVector`, `slice.Spat`

Examples

```
library(terra)

v <- vect(system.file("ex/lux.shp", package = "terra"))

# Create a vector with duplicates.
v <- v[sample(seq_len(nrow(v)), 100, replace = TRUE), ]
v$gr <- sample(LETTERS[1:3], 100, replace = TRUE)

# All duplicates
ex1 <- distinct(v)
ex1

nrow(ex1)

# Duplicates by gr
ex2 <- distinct(v, gr)
ex2
nrow(ex2)

# Keep all columns.
ex2b <- distinct(v, gr, .keep_all = TRUE)
ex2b
nrow(ex2b)

# Unique geometries
ex3 <- distinct(v, geometry)

ex3
nrow(ex3)
# Same as terra::unique()
terra::unique(ex3)

# Keep information for unique values.
distinct(v, geometry, .keep_all = TRUE)
```

`drop_na.Spat` *Drop attributes of* `Spat*` *objects containing missing values*

Description

- SpatVector: `drop_na()` drops geometries when any attribute specified by `...` contains a missing value.
- SpatRaster: `drop_na()` drops cells when any layer specified by `...` contains a missing value.

Usage

```
## S3 method for class 'SpatVector'
```

```
drop_na(data, ...)

## S3 method for class 'SpatRaster'
drop_na(data, ...)
```

Arguments

data	A SpatVector created with terra::vect() or a SpatRaster created with terra::rast().
...	<tidy-select> Attributes to inspect for missing values. If empty, all attributes are used.

Value

A Spat* object of the same class as data. See **Methods**.

terra equivalent

terra::trim().

Methods

Implementation of the **generic** tidyr::drop_na() methods for Spat* objects.

SpatVector**:**

This method operates on attributes, meaning that NA values are assessed in the attributes (columns) of each geometry (row). The result is a SpatVector with potentially fewer geometries than the input.

SpatRaster**:**

[Questioning]

The implementation of drop_na.SpatRaster() can be understood as a masking method based on the values of the layers (see terra::mask()).

SpatRaster layers are treated as columns and SpatRaster cells as rows, so rows (cells) with any NA value on any layer become NA. You can also mask the cells (rows) based on the values of specific layers (columns). Interior cells remain on the regular raster grid with missing values. Masking does not remove individual cells from that grid.

drop_na() effectively removes outer cells that are NA (see terra::trim()), so the extent of the resulting object may differ from the extent of the input (see terra::resample() for more information).

Check the **Examples** to have a better understanding of this method.

Feedback needed!:
Visit https://github.com/dieghernan/tidyterra/issues. The implementation of this method for SpatRaster may change in the future.

See Also

tidyr::drop_na().

Other **tidyr** verbs for handling missing values: complete.SpatVector(), expand.SpatVector(), fill.SpatVector(), replace_na.Spat

Examples

```
library(terra)

f <- system.file("extdata/cyl.gpkg", package = "tidyterra")

v <- terra::vect(f)

# Add missing values.
v <- v |> mutate(iso2 = ifelse(cpro <= "09", NA, cpro))

# Initial plot.
plot(v, col = "red")

# Drop geometries with missing values in iso2.
v |>
  drop_na(iso2) |>
  plot(col = "red")
# SpatRaster method


r <- rast(
  crs = "EPSG:3857",
  extent = c(0, 10, 0, 10),
  nlyr = 3,
  resolution = c(2.5, 2.5)
)
terra::values(r) <- seq_len(ncell(r) * nlyr(r))

# Add missing values.
r[r > 13 & r < 22 | r > 31 & r < 45] <- NA

# Initial plot.
plot(r, nc = 3)

# Mask with lyr.1.
r |>
  drop_na(lyr.1) |>
  plot(nc = 3)

# Mask with lyr.2.
r |>
  drop_na(lyr.2) |>
  plot(nc = 3)

# Mask with lyr.3.
r |>
  drop_na(lyr.3) |>
  plot(nc = 3)

# Mask all layers.
r |>
  drop_na() |>
```

```
plot(nc = 3)
```

`expand.SpatVector` *Expand* `SpatVector` *attribute combinations*

Description

`expand()` returns a tibble with all combinations of selected attributes. It does not return a `SpatVector` because newly created combinations do not have a well-defined geometry. Use `complete.SpatVector()` when empty geometries should be added explicitly.

Usage

```
## S3 method for class 'SpatVector'
expand(data, ..., .name_repair = "check_unique")
```

Arguments

`data`	A `SpatVector`.
`...`	<data-masking> Specification of columns to expand or complete. Columns can be atomic vectors or lists. • To find all unique combinations of `x`, `y` and `z`, including those not present in the data, supply each variable as a separate argument: `expand(df, x, y, z)` or `complete(df, x, y, z)`. • To find only the combinations that occur in the data, use `nesting`: `expand(df, nesting(x, y, z))`. • You can combine the two forms. For example, `expand(df, nesting(school_id, student_id), date)` would produce a row for each present school-student combination for all possible dates. When used with factors, `expand()` and `complete()` use the full set of levels, not just those that appear in the data. If you want to use only the values seen in the data, use `forcats::fct_drop()`. When used with continuous variables, you may need to fill in values that do not appear in the data: to do so use expressions like `year = 2010:2020` or `year = full_seq(year,1)`.
`.name_repair`	One of `"check_unique"`, `"unique"`, `"universal"`, `"minimal"`, `"unique_quiet"`, or `"universal_quiet"`. See `vec_as_names()` for the meaning of these options.

Value

A tibble.

Methods

Implementation of the **generic** tidyr::expand() method for SpatVector objects.

The output is a tibble with attribute combinations. Geometry is not preserved because new combinations do not have a well-defined geometry.

See Also

tidyr::expand(), complete.SpatVector().

Other **tidyr** verbs for handling missing values: complete.SpatVector(), drop_na.Spat, fill.SpatVector(), replace_na.Spat

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
v$grp <- rep(c("A", "B"), length.out = nrow(v))

expand(v, grp, cpro)
```

---

fill.SpatVector *Fill in missing values with previous or next value on a SpatVector*

Description

Fills missing values in selected columns using the next or previous entry. This is useful in the common output format where values are not repeated and are only recorded when they change.

Usage

```
## S3 method for class 'SpatVector'
fill(data, ..., .by = NULL, .direction = c("down", "up", "downup", "updown"))
```

Arguments

data	A SpatVector.
...	<tidy-select> Columns to fill.
.by	**[Experimental]** <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.
.direction	Direction in which to fill missing values. Currently either "down" (the default), "up", "downup" (i.e. first down and then up) or "updown" (first up and then down).

Value

A SpatVector object.

Methods

Implementation of the **generic** `tidyr::fill()` method for `SpatVector` objects.

Grouped `SpatVector`

With grouped `SpatVector` objects created by `group_by.SpatVector()`, `fill()` is applied *within* each group and does not fill across group boundaries.

See Also

`tidyr::fill()`.

Other **tidyr** verbs for handling missing values: `complete.SpatVector()`, `drop_na.Spat`, `expand.SpatVector()`, `replace_na.Spat`

Examples

```
library(dplyr)

lux <- terra::vect(system.file("ex/lux.shp", package = "terra"))

# Leave some blanks for demo purposes

lux_blnk <- lux |>
  mutate(NAME_1 = if_else(NAME_1 != NAME_2, NA, NAME_2))

as_tibble(lux_blnk)

# `fill()` defaults to replacing missing data from top to bottom
lux_blnk |>
  fill(NAME_1) |>
  as_tibble()

# direction = "up"
lux_blnk |>
  fill(NAME_1, .direction = "up") |>
  as_tibble()

# Grouping and downup - will restore the initial state
lux_blnk |>
  group_by(ID_1) |>
  fill(NAME_1, .direction = "downup") |>
  as_tibble()
```

`filter-joins.SpatVector`

Filtering joins for `SpatVector` *objects*

Description

Filtering joins filter rows from x based on the presence or absence of matches in y:

- semi_join() returns all rows from x with a match in y.
- anti_join() returns all rows from x without a match in y.

See dplyr::semi_join() for details.

Usage

```
## S3 method for class 'SpatVector'
semi_join(x, y, by = NULL, copy = FALSE, ...)

## S3 method for class 'SpatVector'
anti_join(x, y, by = NULL, copy = FALSE, ...)
```

Arguments

x	A SpatVector created with terra::vect().
y	A data frame or other object coercible to a data frame. If a SpatVector or sf object is provided, this method returns an error. See terra::intersect() for spatial joins.
by	A join specification created with join_by(), or a character vector of variables to join by. If NULL, the default, *_join() will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying by explicitly. To join on different variables between x and y, use a join_by() specification. For example, join_by(a == b) will match x$a to y$b. To join by multiple variables, use a join_by() specification with multiple expressions. For example, join_by(a == b, c == d) will match x$a to y$b and x$c to y$d. If the column names are the same between x and y, you can shorten this by listing only the variable names, like join_by(a, c). join_by() can also be used to perform inequality, rolling, and overlap joins. See the documentation at ?join_by for details on these types of joins. For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, by = c("a", "b") joins x$a to y$a and x$b to y$b. If variable names differ between x and y, use a named character vector like by = c("x_a" = "y_a", "x_b" = "y_b"). To perform a cross-join, generating all combinations of x and y, see cross_join().
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
...	Other parameters passed onto methods.

Value

A SpatVector object.

terra equivalent

`terra::merge()`.

Methods

Implementation of the **generic** `dplyr::semi_join()` method family for `SpatVector` objects.

The geometry column has sticky behavior. This means that the result always has the geometry of `x` for the records that match the join conditions.

See Also

`dplyr::semi_join()`, `dplyr::anti_join()`, `terra::merge()`.

Other **dplyr** verbs that operate on pairs of `SpatVector` and data frame objects: `bind_cols.SpatVector`, `bind_rows.SpatVector`, `cross_join.SpatVector()`, `mutate-joins.SpatVector`, `nest_join.SpatVector()`, `rows.SpatVector`

Examples

```
library(terra)
library(ggplot2)

# Vector
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

# A data frame
df <- data.frame(
  cpro = sprintf("%02d", 1:10),
  x = runif(10),
  y = runif(10),
  letter = rep_len(LETTERS[1:3], length.out = 10)
)

v

# Semi join
semi <- v |> semi_join(df)

semi

autoplot(semi, aes(fill = iso2)) + labs(title = "Semi Join")

# Anti join

anti <- v |> anti_join(df)

anti

autoplot(anti, aes(fill = iso2)) + labs(title = "Anti Join")
```

`filter.Spat` *Subset cells/geometries of* `Spat*` *objects*

Description

These functions subset a `Spat*` object by applying the expressions in `...` to determine which rows should be kept (for `filter()`) or dropped (for `filter_out()`).

Multiple conditions can be supplied separated by a comma. These will be combined with the `&` operator. To combine comma-separated conditions using `|` instead, wrap them in `dplyr::when_any()`.

Both `filter()` and `filter_out()` treat `NA` like `FALSE`. This subtle behavior can affect how you write your conditions when missing values are involved. See `dplyr::filter()`.

You can filter a `SpatRaster` by its geographic coordinates. Use `filter(.data, x > 42)`. The names `x` and `y` are reserved in **terra** because they refer to the geographic coordinates of the layer.

See **Examples** and section **Layer and column names** on `as_tibble.Spat()`.

Usage

```
## S3 method for class 'SpatRaster'
filter(.data, ..., .preserve = FALSE, .keep_extent = TRUE)

## S3 method for class 'SpatVector'
filter(.data, ..., .by = NULL, .preserve = FALSE)

## S3 method for class 'SpatVector'
filter_out(.data, ..., .by = NULL, .preserve = FALSE)
```

Arguments

`.data`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`...`	<`data-masking`> Expressions that return a logical value and are defined in terms of the layers/attributes in `.data`. If multiple expressions are included, they are combined with the `&` operator. Only cells/geometries for which all conditions evaluate to `TRUE` are kept. See **Methods**.
`.preserve`	Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
`.keep_extent`	Logical. If `TRUE`, keep the extent of the resulting `SpatRaster`. If `FALSE`, `terra::trim()` is called so the extent may differ from the extent of the input. See also `drop_na.SpatRaster()`.
`.by`	<`tidy-select`> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.

Value

A `Spat*` object of the same class as `.data`. See **Methods**.

Methods

Implementation of the **generic** dplyr::filter() methods for Spat* objects.

SpatRaster**:**

Cells that do not meet the conditions in ... are returned as NA. On a multi-layer SpatRaster, NA is propagated across all layers.

If .keep_extent = TRUE, the returned SpatRaster has the same CRS, extent, resolution and number of cells as .data. If .keep_extent = FALSE, the outer NA cells are trimmed with terra::trim(), so the extent and number of cells may differ. The output still has the same CRS and resolution as .data.

The x and y coordinates of the SpatRaster are also available internally for filtering. See **Examples**.

SpatVector**:**

The result is a SpatVector containing the geometries whose attributes satisfy all conditions.

See Also

dplyr::filter().

Other **dplyr** verbs that operate on rows: arrange.SpatVector(), distinct.SpatVector(), rows.SpatVector, slice.Spat

Examples

```
library(tidyterra)

points <- terra::vect(
  data.frame(x = c(1, 2), y = c(3, 4), value = c(5, 15)),
  geom = c("x", "y"), crs = "EPSG:3857"
)
filtered_points <- filter(points, value > 10)
filtered_points
nrow(filtered_points)

library(terra)
f <- system.file("extdata/cyl_temp.tif", package = "tidyterra")

r <- rast(f) |> select(tavg_04)

plot(r)

# Filter temps
r_f <- r |> filter(tavg_04 > 11.5)

# Extent is kept
plot(r_f)

# Filter temps and extent
r_f2 <- r |> filter(tavg_04 > 11.5, .keep_extent = FALSE)
```

```
# Extent has changed
plot(r_f2)

# Filter by geographic coordinates
r2 <- project(r, "epsg:4326")

r2 |> plot()

r2 |>
  filter(
    x > -4,
    x < -2,
    y > 42
  ) |>
  plot()
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
glimpse(v)
v |> filter(cpro < 10)

# Same as
v |> filter_out(cpro >= 10)
```

`fortify.Spat` *Fortify* `Spat*` *objects*

Description

Fortify `SpatRaster` and `SpatVector` objects to data frames. This provides native compatibility with `ggplot2::ggplot()`.

These methods are now implemented as wrappers around `tidy.Spat` methods.

Usage

```
## S3 method for class 'SpatRaster'
fortify(
  model,
  data,
  ...,
  .name_repair = c("unique", "check_unique", "universal", "minimal", "unique_quiet",
    "universal_quiet"),
  maxcell = terra::ncell(model) * 1.1,
  pivot = FALSE
)

## S3 method for class 'SpatVector'
fortify(model, data, ...)

## S3 method for class 'SpatGraticule'
```

```
fortify(model, data, ...)

## S3 method for class 'SpatExtent'
fortify(model, data, ..., crs = "")
```

Arguments

model	A SpatRaster created with terra::rast(), a SpatVector created with terra::vect(), a SpatGraticule (see terra::graticule()) or a SpatExtent (see terra::ext()).
data	Not used by this method.
...	Ignored by these methods.
.name_repair	Treatment of problematic column names: • "minimal": No name repair or checks, beyond basic existence, • "unique": Make sure names are unique and not empty, • "check_unique": (default value), no name repair, but check they are unique, • "universal": Make the names unique and syntactic • "unique_quiet": Same as "unique", but "quiet" • "universal_quiet": Same as "universal", but "quiet" • a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R). • A purrr-style anonymous function, see rlang::as_function() This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.
maxcell	Positive integer or Inf. Maximum number of raster cells to use. Use Inf to remove this sampling limit. Reprojection for the plot can still change the grid, so Inf does not guarantee native plot resolution. The input raster is not modified.
pivot	Logical. When TRUE, a SpatRaster is returned in long format. When FALSE (the default), it is returned as a data frame with one column per layer. See **Details**.
crs	Input that includes or represents a CRS. It can be an sf or sfc object, a SpatRaster or SpatVector object, a crs object from sf::st_crs(), a character string (for example a PROJ string) or an integer representing an EPSG code.

Value

fortify.SpatVector(), fortify.SpatGraticule() and fortify.SpatExtent() return an sf object.

fortify.SpatRaster() returns a tibble. See **Methods**.

Methods

Implementation of the **generic** ggplot2::fortify() methods for Spat* objects.

SpatRaster**:**

Returns a tibble that can be used with ggplot2::geom_*, such as ggplot2::geom_point() and ggplot2::geom_raster().

The resulting tibble includes coordinates in the x and y columns. The values of each layer are added as extra columns using the layer names from the SpatRaster.

The CRS of the SpatRaster can be retrieved with `attr(fortifiedSpatRaster, "crs")`.

You can convert the fortified object back to a SpatRaster with as_spatraster().

When pivot = TRUE, the SpatRaster is fortified in long format (see tidyr::pivot_longer()). The fortified object has the following columns:

- x, y: Coordinates of the cell center in the corresponding CRS.
- lyr: Name of the SpatRaster layer associated with value.
- value: Cell value for the corresponding lyr.

This option can be useful when combining several geom_* layers or when faceting.

SpatVector, SpatGraticule and SpatExtent:

Returns an sf object that can be used with ggplot2::geom_sf(). For SpatVector objects, Z and M coordinates are dropped for plotting. Use as_sf() to preserve these coordinates when converting to sf.

See Also

tidy.Spat, sf::st_as_sf().

Other **ggplot2** helpers: autoplot.Spat, geom_spat_contour, geom_spatraster(), geom_spatraster_rgb(), ggspatvector, stat_spat_coordinates()

Other **ggplot2** methods: autoplot.Spat

Coercing objects: as_coordinates(), as_sf(), as_spatraster(), as_spatvector(), as_tibble.Spat, tidy.Spat

Examples

```
# Demonstrate use with ggplot2.
library(ggplot2)

# Get a SpatRaster.
r <- system.file("extdata/volcano2.tif", package = "tidyterra") |>
  terra::rast() |>
  terra::project("EPSG:4326")

# You can now use a SpatRaster with any geom.
ggplot(r, maxcell = 50) +
  geom_histogram(aes(x = elevation),
    bins = 20, fill = "lightblue",
    color = "black"
  )

# For SpatVector, SpatGraticule and SpatExtent, use geom_sf().

# Create a SpatVector.
extfile <- system.file("extdata/cyl.gpkg", package = "tidyterra")
cyl <- terra::vect(extfile)
```

```
class(cyl)

ggplot(cyl) +
  geom_sf()

# SpatGraticule
g <- terra::graticule(60, 30, crs = "+proj=robin")

class(g)

ggplot(g) +
  geom_sf()

# SpatExtent
ex <- terra::ext(cyl)

class(ex)

ggplot(ex, crs = cyl) +
  geom_sf(fill = "red", alpha = 0.3) +
  geom_sf(data = cyl, fill = NA)
```

`geom_spatraster` *Plot* `SpatRaster` *objects*

Description

This geom plots `SpatRaster` objects (see `terra::rast()`). It is designed to plot the object by layers, as `terra::plot()` does.

For plotting `SpatRaster` objects as map tiles, such as RGB `SpatRaster` objects, use `geom_spatraster_rgb()`. To plot one panel per raster layer in two columns, add `ggplot2::facet_wrap(~lyr, ncol = 2)`. The default `maxcell = 500000` limits the cells sampled for plotting.

The underlying implementation is based on `ggplot2::geom_raster()`.

`stat_spatraster()` complements `geom_spatraster()` when you need to change the `geom`.

Usage

```
geom_spatraster(
  mapping = aes(),
  data,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = FALSE,
  interpolate = FALSE,
  maxcell = 5e+05,
```

```
  use_coltab = TRUE,
  mask_projection = FALSE,
  ...
)

stat_spatraster(
  mapping = aes(),
  data,
  geom = "raster",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = FALSE,
  maxcell = 5e+05,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). See **Aesthetics**, especially the use of the fill and alpha aesthetics with SpatRaster layers.
data	A SpatRaster object.
na.rm	If TRUE, the default, missing values are silently removed. If FALSE, missing values are removed with a warning.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, override the default aesthetics rather than combining with them.
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
maxcell	Positive integer or Inf. Maximum number of raster cells to use. Use Inf to remove this sampling limit. Reprojection for the plot can still change the grid, so Inf does not guarantee native plot resolution. The input raster is not modified.
use_coltab	Logical. Only applicable to SpatRaster objects that have an associated color table from terra::coltab(). If TRUE, use that color table on the plot. See also scale_fill_coltab().
mask_projection	Logical, defaults to FALSE. If TRUE, mask out areas outside the input extent. For example, to avoid data wrapping around the dateline in equal-area projections. This argument is passed to terra::project() when reprojecting the SpatRaster.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`geom` Geom used to display the data. Recommended values for `SpatRaster` are `"raster"` (the default), `"point"`, `"text"` and `"label"`.

Value

A **ggplot2** layer.

terra equivalent

`terra::plot()`.

Coordinates

When the `SpatRaster` does not have a CRS, that is, `terra::crs(rast) == ""`, the geom does not make any assumption about the scales.

On `SpatRaster` objects that have a CRS, the geom uses `ggplot2::coord_sf()` to adjust the scales. This means that the `SpatRaster` **may be reprojected**.

Aesthetics

`geom_spatraster()` understands the following aesthetics:

- `fill`
- `alpha`

If `fill` is not provided, `geom_spatraster()` creates a **ggplot2** layer with all the layers of the `SpatRaster` object. Use `facet_wrap(~lyr)` to display its layers. layers.

If `fill` is used, it should contain the name of one layer that is present on the `SpatRaster` (for example, `geom_spatraster(data = rast, aes(fill = <name_of_lyr>))`). Layer names can be retrieved using `names(rast)`.

Using `geom_spatraster(..., mapping = aes(fill = NULL))` or `geom_spatraster(..., fill = <color value(s)>)` creates a layer with no mapped `fill` aesthetic.

`fill` can use computed variables.

For `alpha`, use a computed variable or the name of one layer that is present on the `SpatRaster`. When `alpha` is a layer name, it can refer to the same layer as `fill` or to another layer in the `SpatRaster`. See section **Computed variables**.

`stat_spatraster()`:

`stat_spatraster()` understands the same aesthetics as `geom_spatraster()` when `geom = "raster"` (the default):

- `fill`
- `alpha`

When `geom = "raster"`, the `fill` and `alpha` arguments behave as in `geom_spatraster()`, so they can be mapped to `SpatRaster` layer names. If another `geom` is used, `stat_spatraster()` understands the aesthetics required by that `geom`, so `aes(fill = <name_of_lyr>)` and `aes(alpha = <name_of_lyr>)` are not applicable.

The `x` and `y` aesthetics are mapped by default, so you do not need to add them in `aes()`. In every case, aesthetics should be mapped with computed variables. See **Computed variables** and **Examples**.

Facets

You can use `facet_wrap(~lyr)` to create a faceted plot for each layer of the `SpatRaster` object. See `ggplot2::facet_wrap()` for details.

Computed variables

This geom computes variables internally that are available for use as aesthetics, using (for example) `aes(alpha = after_stat(value))` (see `ggplot2::after_stat()`).

- `after_stat(value)`: Cell values of the `SpatRaster`.
- `after_stat(alpha)`: Cell values of the layer mapped to `alpha`, when `alpha` is the name of a layer.
- `after_stat(lyr)`: Name of the layer.

Source

Based on the `layer_spatial()` implementation in **ggspatial**. Thanks to Dewey Dunnington and ggspatial contributors.

See Also

`ggplot2::geom_raster()`, `ggplot2::coord_sf()`, `ggplot2::facet_wrap()`.

Recommended geoms:

- `ggplot2::geom_point()`.
- `ggplot2::geom_label()`.

- ggplot2::geom_text().

Other ggplot2 helpers: autoplot.Spat, fortify.Spat, geom_spat_contour, geom_spatraster_rgb(), ggspatvector, stat_spat_coordinates()

Examples

```
# Average spring temperature in Castile and Leon (Spain)
file_path <- system.file("extdata/cyl_temp.tif", package = "tidyterra")

library(terra)
temp_rast <- rast(file_path)

library(ggplot2)

# Display a single layer.
names(temp_rast)

ggplot() +
  geom_spatraster(data = temp_rast, aes(fill = tavg_04)) +
  # You can use coord_sf().
  coord_sf(crs = 3857) +
  scale_fill_grass_c(palette = "celsius")

# Display facets.
ggplot() +
  geom_spatraster(data = temp_rast) +
  facet_wrap(~lyr, ncol = 2) +
  scale_fill_grass_b(palette = "celsius", breaks = seq(0, 20, 2.5))

# Non-spatial rasters.

no_crs <- rast(crs = NA, extent = c(0, 100, 0, 100), nlyr = 1)
values(no_crs) <- seq_len(ncell(no_crs))

ggplot() +
  geom_spatraster(data = no_crs)

# Downsample.

ggplot() +
  geom_spatraster(data = no_crs, maxcell = 25)



# Using stat_spatraster
# Default
ggplot() +
  stat_spatraster(data = temp_rast) +
  facet_wrap(~lyr)

# Using points
ggplot() +
```

```
  stat_spatraster(
    data = temp_rast,
    aes(color = after_stat(value)),
    geom = "point", maxcell = 250
  ) +
  scale_colour_viridis_c(na.value = "transparent") +
  facet_wrap(~lyr)

# Using points and labels

r_single <- temp_rast |> select(1)

ggplot() +
  stat_spatraster(
    data = r_single,
    aes(color = after_stat(value)),
    geom = "point",
    maxcell = 2000
  ) +
  stat_spatraster(
    data = r_single,
    aes(label = after_stat(round(value, 2))),
    geom = "label",
    alpha = 0.85,
    maxcell = 20
  ) +
  scale_colour_viridis_c(na.value = "transparent")
```

`geom_spatraster_rgb` *Plot* `SpatRaster` *objects as images*

Description

This geom plots `SpatRaster` objects (see `terra::rast()`) as RGB images. The layers are combined so they represent the red, green and blue channels. Use `geom_spatraster_rgb(data = tile, r = 1, g = 2, b = 3)` to map layers 1, 2 and 3 to those channels. The package includes `extdata/cyl_tile.tif` as an example RGB raster.

For plotting `SpatRaster` objects by layer values use `geom_spatraster()`.

The underlying implementation is based on `ggplot2::geom_raster()`.

Usage

```
geom_spatraster_rgb(
  mapping = aes(),
  data,
  interpolate = TRUE,
  r = 1,
  g = 2,
```

```
  b = 3,
  alpha = 1,
  maxcell = 5e+05,
  max_col_value = 255,
  ...,
  stretch = NULL,
  zlim = NULL,
  mask_projection = FALSE
)
```

Arguments

mapping	Ignored.
data	A SpatRaster object.
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
r, g, b	Integer giving the layer number in data used for the red (r), green (g) and blue (b) channel.
alpha	The alpha transparency, a number in [0,1], see argument alpha in hsv.
maxcell	Positive integer or Inf. Maximum number of raster cells to use. Use Inf to remove this sampling limit. Reprojection for the plot can still change the grid, so Inf does not guarantee native plot resolution. The input raster is not modified.
max_col_value	Number giving the upper bound of the color value range. When this is 255 (the default), the result is computed most efficiently. See grDevices::rgb().
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept. • The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

stretch — character. Option to stretch the values to increase contrast: "lin" (linear) or "hist" (histogram). The linear stretch uses `stretch` with arguments `minq=0.02` and `maxq=0.98`

zlim — numeric vector of length 2. Range of values to plot (optional). If this is set, and `stretch="lin"` is used, then the values are stretched within the range of `zlim`. This allows creating consistent coloring between SpatRasters with different cell-value ranges, even when stretching the colors for improved contrast

mask_projection — Logical, defaults to `FALSE`. If `TRUE`, mask out areas outside the input extent. For example, to avoid data wrapping around the dateline in equal-area projections. This argument is passed to `terra::project()` when reprojecting the `SpatRaster`.

Value

A **ggplot2** layer.

terra equivalent

`terra::plotRGB()`.

Aesthetics

No `aes()` is required. In fact, `aes()` will be ignored.

Coordinates

When the `SpatRaster` does not have a CRS, that is, `terra::crs(rast) == ""`, the geom does not make any assumption about the scales.

On `SpatRaster` objects that have a CRS, the geom uses `ggplot2::coord_sf()` to adjust the scales. This means that the `SpatRaster` **may be reprojected**.

Source

Based on the `layer_spatial()` implementation in the **ggspatial** package. Thanks to Dewey Dunnington and to ggspatial contributors.

See Also

`ggplot2::geom_raster()`, `ggplot2::coord_sf()`, `grDevices::rgb()`.

You can also get RGB tiles from the **maptiles** package. See `maptiles::get_tiles()`.

Other **ggplot2** helpers: `autoplot.Spat`, `fortify.Spat`, `geom_spat_contour`, `geom_spatraster()`, `ggspatvector`, `stat_spat_coordinates()`

Examples

```
library(ggplot2)
library(tidyterra)

rgb <- terra::rast(nrows = 2, ncols = 2, nlyrs = 3, vals = 1:12)
ggplot() +
  geom_spatraster_rgb(data = rgb, r = 1, g = 2, b = 3)



# Tile of Castile and Leon (Spain) from OpenStreetMap
file_path <- system.file("extdata/cyl_tile.tif", package = "tidyterra")

library(terra)
tile <- rast(file_path)

library(ggplot2)

ggplot() +
  geom_spatraster_rgb(data = tile) +
  # You can use coord_sf
  coord_sf(crs = 3035)

# Combine with sf objects
vect_path <- system.file("extdata/cyl.gpkg", package = "tidyterra")

cyl_sf <- sf::st_read(vect_path)

ggplot(cyl_sf) +
  geom_spatraster_rgb(data = tile) +
  geom_sf(aes(fill = iso2)) +
  coord_sf(crs = 3857) +
  scale_fill_viridis_d(alpha = 0.7)
```

---

geom_spat_contour *Plot* SpatRaster *contours*

Description

These geoms create contours from SpatRaster objects. To create contours from a specific layer, map that layer to the z aesthetic with `aes(z = layer_name)`. Otherwise, all layers are used to create contours. See also the **Facets** section.

The underlying implementation is based on `ggplot2::geom_contour()`.

`geom_spatraster_contour_text()` creates labeled contours and it is implemented on top of `isoband::isolines_grob()`.

Usage

```
geom_spatraster_contour(
  mapping = NULL,
  data,
  ...,
  maxcell = 5e+05,
  bins = NULL,
  binwidth = NULL,
  breaks = NULL,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  mask_projection = FALSE
)

geom_spatraster_contour_text(
  mapping = NULL,
  data,
  ...,
  maxcell = 5e+05,
  bins = NULL,
  binwidth = NULL,
  breaks = NULL,
  size.unit = "mm",
  label_format = scales::label_number(),
  label_placer = isoband::label_placer_minmax(),
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  mask_projection = FALSE
)

geom_spatraster_contour_filled(
  mapping = NULL,
  data,
  ...,
  maxcell = 5e+05,
  bins = NULL,
  binwidth = NULL,
  breaks = NULL,
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  mask_projection = FALSE
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). See **Aesthetics** for supported contour aesthetics.
data	A SpatRaster object.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept. • The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
maxcell	Positive integer or Inf. Maximum number of raster cells to use. Use Inf to remove this sampling limit. Reprojection for the plot can still change the grid, so Inf does not guarantee native plot resolution. The input raster is not modified.
bins	Number of contour bins. Overridden by breaks.
binwidth	The width of the contour bins. Overridden by bins.
breaks	One of: • Numeric vector to set the contour breaks • A function that takes the range of the data and binwidth as input and returns breaks as output. A function can be created from a formula (e.g. ~ fullseq(.x, .y)). Overrides binwidth and bins. By default, this is a vector of length ten with pretty() breaks.
na.rm	If TRUE, the default, missing values are silently removed. If FALSE, missing values are removed with a warning.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, override the default aesthetics rather than combining with them.

`mask_projection`
Logical, defaults to `FALSE`. If `TRUE`, mask out areas outside the input extent. For example, to avoid data wrapping around the dateline in equal-area projections. This argument is passed to `terra::project()` when reprojecting the `SpatRaster`.

`size.unit` How the `size` aesthetic is interpreted: as millimetres (`"mm"`, default), points (`"pt"`), centimetres (`"cm"`), inches (`"in"`), or picas (`"pc"`).

`label_format` One of:

- `NULL` for no labels. This produces the same result as `geom_spatraster_contour()`.
- A character vector giving labels (must have the same length as the breaks produced by `bins`, `binwidth` or `breaks`).
- A function that takes the breaks as input and returns labels as output, as in the default setup (`scales::label_number()`).

`label_placer` Function that controls how labels are placed along the isolines. Uses `label_placer_minmax()` by default.

Value

A **ggplot2** layer.

terra equivalent

`terra::contour()`.

Aesthetics

`geom_spatraster_contour()` and `geom_spatraster_contour_text()` understand the following aesthetics:

- `alpha`
- `colour`
- `group`
- `linetype`
- `linewidth`
- `weight`

`geom_spatraster_contour_text()` also understands:

- `size`
- `label`
- `family`
- `fontface`

In addition to the aesthetics listed above, `geom_spatraster_contour_filled()` understands:

- `fill`

- subgroup

Check ggplot2::geom_contour() for more information on contours and vignette("ggplot2-specs", package = "ggplot2") for an overview of the aesthetics.

Computed variables

These geoms compute variables internally that are available for use as aesthetics, using (for example) aes(color = after_stat(<computed>)) (see ggplot2::after_stat()).

- after_stat(lyr): Name of the layer.
- after_stat(level): Height of contour. For contour lines, this is a numeric vector that represents bin boundaries. For contour bands, this is an ordered factor that represents bin ranges.
- after_stat(nlevel): Height of contour, scaled to maximum of 1.
- after_stat(level_low), after_stat(level_high) and after_stat(level_mid): For contour bands only, the lower and upper bin boundaries and the midpoint between them.

Dropped variables

- z: After contouring, the z values of individual data points are no longer available.

Coordinates

When the SpatRaster does not have a CRS, that is, terra::crs(rast) == "", the geom does not make any assumption about the scales.

On SpatRaster objects that have a CRS, the geom uses ggplot2::coord_sf() to adjust the scales. This means that the SpatRaster **may be reprojected**.

Facets

You can use facet_wrap(~lyr) to create a faceted plot for each layer of the SpatRaster object. See ggplot2::facet_wrap() for details.

See Also

ggplot2::geom_contour().

The **metR** package also provides a set of alternative functions:

- metR::geom_contour2().
- metR::geom_text_contour() and metR::geom_label_contour().
- metR::geom_contour_tanaka().

Other **ggplot2** helpers: autoplot.Spat, fortify.Spat, geom_spatraster(), geom_spatraster_rgb(), ggspatvector, stat_spat_coordinates()

Examples

```
library(terra)

# Raster
f <- system.file("extdata/volcano2.tif", package = "tidyterra")
r <- rast(f)

library(ggplot2)

ggplot() +
  geom_spatraster_contour(data = r)

# Labeled
ggplot() +
  geom_spatraster_contour_text(
    data = r, breaks = c(110, 130, 160, 190),
    color = "grey10", family = "serif"
  )

ggplot() +
  geom_spatraster_contour(
    data = r, aes(color = after_stat(level)),
    binwidth = 1,
    linewidth = 0.4
  ) +
  scale_color_gradientn(
    colours = hcl.colors(20, "Inferno"),
    guide = guide_coloursteps()
  ) +
  theme_minimal()

# Filled with breaks
ggplot() +
  geom_spatraster_contour_filled(data = r, breaks = seq(80, 200, 10)) +
  scale_fill_hypso_d()

# Both lines and contours
ggplot() +
  geom_spatraster_contour_filled(
    data = r, breaks = seq(80, 200, 10),
    alpha = 0.7
  ) +
  geom_spatraster_contour(
    data = r, breaks = seq(80, 200, 2.5),
    color = "grey30",
    linewidth = 0.1
  ) +
  scale_fill_hypso_d()
```

ggspatvector *Plot* SpatVector *objects*

Description

Wrappers of the ggplot2::geom_sf() family used to plot SpatVector objects (see terra::vect()).

Usage

```
geom_spatvector(
  mapping = aes(),
  data = NULL,
  na.rm = FALSE,
  show.legend = NA,
  ...
)

geom_spatvector_label(
  mapping = aes(),
  data = NULL,
  na.rm = FALSE,
  show.legend = NA,
  ...,
  inherit.aes = TRUE
)

geom_spatvector_text(
  mapping = aes(),
  data = NULL,
  na.rm = FALSE,
  show.legend = NA,
  ...,
  check_overlap = FALSE,
  inherit.aes = TRUE
)

stat_spatvector(
  mapping = NULL,
  data = NULL,
  geom = "rect",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	A SpatVector object, see terra::vect().
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. You can also set this to one of "polygon", "line", and "point" to override the default legend.
...	Other arguments passed to ggplot2::geom_sf() functions. These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or linewidth = 3.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().
check_overlap	If TRUE, text that overlaps previous text in the same layer will not be plotted. check_overlap happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling geom_text(). Note that this argument is not supported by geom_label().
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.

Details

These functions are wrappers of `ggplot2::geom_sf()` functions. Since a `fortify.SpatVector()` method is provided, **ggplot2** treats a `SpatVector` in the same way as an `sf` object. A side effect is that you can use `ggplot2::geom_sf()` directly with `SpatVector` objects.

See `ggplot2::geom_sf()` for details on aesthetics, etc.

Value

A **ggplot2** layer.

terra equivalent

`terra::plot()`.

See Also

`ggplot2::geom_sf()`.

Other **ggplot2** helpers: `autoplot.Spat`, `fortify.Spat`, `geom_spat_contour`, `geom_spatraster()`, `geom_spatraster_rgb()`, `stat_spat_coordinates()`

Examples

```
# Create a SpatVector
extfile <- system.file("extdata/cyl.gpkg", package = "tidyterra")

cyl <- terra::vect(extfile)
class(cyl)

library(ggplot2)

ggplot(cyl) +
  geom_spatvector()

# With params

ggplot(cyl) +
  geom_spatvector(aes(fill = name), color = NA) +
  scale_fill_viridis_d() +
  coord_sf(crs = 3857)

# Add labels
ggplot(cyl) +
  geom_spatvector(aes(fill = name), color = NA) +
  geom_spatvector_text(aes(label = iso2),
    fontface = "bold",
    color = "red"
  ) +
  scale_fill_viridis_d(alpha = 0.4) +
  coord_sf(crs = 3857)

# You can now use geom_sf() with SpatVector objects.
```

```
ggplot(cyl) +
  geom_sf() +
  labs(
    title = paste("cyl is", as.character(class(cyl))),
    subtitle = "With geom_sf()"
  )
```

`glance.Spat` *Glance at a* `Spat*` *object*

Description

`glance()` accepts a `Spat*` object and returns a tibble with exactly one row. The summaries typically contain geographic information.

Usage

```
## S3 method for class 'SpatRaster'
glance(x, ...)

## S3 method for class 'SpatVector'
glance(x, ...)
```

Arguments

`x`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`...`	Ignored by this method.

Value

A tibble with one row. See **Methods**.

Methods

Implementation of the **generic** `generics::glance()` methods for `Spat*` objects.

See Also

`glimpse.Spat`, `generics::glance()`.

Other **generics** methods: `required_pkgs.Spat`, `tidy.Spat`

Examples

```
library(terra)

# SpatVector
v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

glance(v)

# SpatRaster
r <- rast(system.file("extdata/cyl_elev.tif", package = "tidyterra"))

glance(r)
```

`glimpse.Spat` *Preview* `Spat*` *objects*

Description

`glimpse()` is like a transposed version of `print()`: layers/columns run down the page and data runs across. This makes it possible to see every layer/column in a `Spat*` object.

Usage

```
## S3 method for class 'SpatRaster'
glimpse(x, width = NULL, ..., n = 10, max_extra_cols = 20)

## S3 method for class 'SpatVector'
glimpse(x, width = NULL, ..., n = 10, max_extra_cols = 20)
```

Arguments

`x`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`width`	Width of output: defaults to the setting of the `width` option (if finite) or the width of the console.
`...`	Arguments passed to `as_tibble.SpatRaster()` or `as_tibble.Spat()` methods.
`n`	Maximum number of rows to show.
`max_extra_cols`	Number of extra columns or layers to print abbreviated information for, if `n` is too small for the `Spat*` object.

Value

The input object `x`, returned invisibly so `glimpse()` can be used within a data pipeline.

terra equivalent

`print()`.

Methods

Implementation of the **generic** dplyr::glimpse() methods for Spat* objects.

See Also

tibble::print.tbl_df()

Other **dplyr** verbs that operate on columns: mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, select.Spat

Examples

```
library(terra)

# SpatVector
v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

v |> glimpse(n = 2)

# Use in a pipeline.
v |>
  glimpse() |>
  mutate(a = 30) |>
  # With options.
  glimpse(geom = "WKT")

# SpatRaster
r <- rast(system.file("extdata/cyl_elev.tif", package = "tidyterra"))

r |> glimpse()

# Use in a pipeline.
r |>
  glimpse() |>
  mutate(b = elevation_m / 100) |>
  # With options
  glimpse(xy = TRUE)
```

---

grass_db *GRASS color tables*

Description

A tibble including the color map of 51 gradient palettes. Some palettes also include a definition of color limits that can be used with ggplot2::scale_fill_gradientn().

Format

A tibble of 2920 rows and 6 columns with the following fields:

pal Name of the palette.

limit (Optional) limit for each color.

r Value of the red channel (RGB color mode).

g Value of the green channel (RGB color mode).

b Value of the blue channel (RGB color mode).

hex Hex code of the color.

Details

Summary of palettes provided, description and recommended use:

palette	use	description	range
aspect	General	Aspect-oriented gray colors	
aspectcolr	General	Aspect-oriented rainbow colors	0 to 36
bcyr	General	Blue through cyan and yellow to red	
bgyr	General	Blue through green and yellow to red	
blues	General	White to blue	
byg	General	Blue through yellow to green	
byr	General	Blue through yellow to red	
celsius	General	Blue to red for Celsius temperatures	-80 to
corine	Land cover	EU Corine land cover colors	111 to
curvature	General	Terrain curvature colors	-0.1 to
differences	General	Difference-oriented colors	
elevation	Topography	Relative raster values mapped to elevation colors	
etopo2	Topography	ETOPO2 worldwide bathymetry and topography colors	-11000
evi	Natural	Enhanced Vegetation Index colors	-1 to 1
fahrenheit	Temperature	Blue to red for Fahrenheit temperatures	-112 to
forest_cover	Natural	Percentage of forest cover	0 to 1
gdd	Natural	Accumulated growing degree days	0 to 60
grass	General	Perceptually uniform GRASS GIS green	
greens	General	White to green	
grey	General	Grayscale	
gyr	General	Green through yellow to red	
haxby	Topography	Relative colors for bathymetry or topography	
inferno	General	Inferno perceptually uniform sequential color table	
kelvin	Temperature	Blue to red for temperatures in Kelvin	193.15
magma	General	Magma perceptually uniform sequential color table	
ndvi	Natural	Normalized Difference Vegetation Index colors	-1 to 1
ndwi	Natural	Normalized Difference Water Index colors	-200 to
nlcd	Land cover	US National Land Cover Dataset colors	0 to 95
oranges	General	White to orange	
plasma	General	Plasma perceptually uniform sequential color table	
population	Human	Human population classification breaks	0 to 10
population_dens	Human	Human population density classification breaks	0 to 10

precipitation	Climate	Precipitation color table, 0 to 2000 mm	0 to 70
precipitation_daily	Climate	Daily precipitation color table, 0 to 1000 mm	0 to 10
precipitation_monthly	Climate	Monthly precipitation color table, 0 to 1000 mm	0 to 10
rainbow	General	Rainbow color table	
ramp	General	Color ramp	
reds	General	White to red	
roygbiv	General		
rstcurv	General	Terrain curvature from r.resamp.rst	-0.1 to
ryb	General	Red through yellow to blue	
ryg	General	Red through yellow to green	
sepia	General	Yellowish-brown to white	
slope	General	`r.slope.aspect`-style slope colors for raster values from 0 to 90	0 to 90
soilmoisture	Natural	Soil moisture color table, 0.0 to 1.0	0 to 1
srtm	Topography	Shuttle Radar Topography Mission elevation colors	-11000
srtm_plus	Topography	Shuttle Radar Topography Mission elevation colors with seafloor colors	-11000
terrain	Topography	Global elevation color table from -11000 to +8850 m	-11000
viridis	General	Viridis perceptually uniform sequential color table	
water	Natural	Water depth	
wave	General	Color wave	

terra equivalent

`terra::map.pal()`.

Source

Derived from https://github.com/OSGeo/grass/tree/main/lib/gis/colors. See also r.color - GRASS GIS Manual.

References

GRASS Development Team (2024). *Geographic Resources Analysis Support System (GRASS) Software, Version 8.3.2*. Open Source Geospatial Foundation, USA. https://grass.osgeo.org.

See Also

`scale_fill_grass_c()`.

Other datasets: `cross_blended_hypsometric_tints_db`, `hypsometric_tints_db`, `princess_db`, `volcano2`

Examples

```
data("grass_db")

grass_db
# Select a palette

srtm_plus <- grass_db |>
  filter(pal == "srtm_plus")
```

```
f <- system.file("extdata/asia.tif", package = "tidyterra")
r <- terra::rast(f)

library(ggplot2)

p <- ggplot() +
  geom_spatraster(data = r) +
  labs(fill = "elevation")

p +
  scale_fill_gradientn(colors = srtm_plus$hex)

# Use with limits
p +
  scale_fill_gradientn(
    colors = srtm_plus$hex,
    values = scales::rescale(srtm_plus$limit),
    limit = range(srtm_plus$limit),
    na.value = "lightblue"
  )
```

`group_by.SpatVector` *Group a* `SpatVector` *by one or more variables*

Description

Most data operations are done on groups defined by variables. `group_by.SpatVector()` adds new attributes to an existing `SpatVector` indicating the corresponding groups. See **Methods**.

Usage

```
## S3 method for class 'SpatVector'
group_by(.data, ..., .add = FALSE, .drop = group_by_drop_default(.data))

## S3 method for class 'SpatVector'
ungroup(x, ...)
```

Arguments

`.data, x`	A `SpatVector` object. See **Methods**.
`...`	<`data-masking`> In `group_by()`, variables or computations to group by. Computations are always done on the ungrouped data frame. To perform computations on the grouped data, you need to use a separate `mutate()` step before the `group_by()`. Computations are not allowed in `nest_by()`. In `ungroup()`, variables to remove from the grouping.
`.add`	When `FALSE`, the default, `group_by()` will override existing groups. To add to the existing groups, use `.add = TRUE`.

`.drop` Drop groups formed by factor levels that don't appear in the data? The default is `TRUE` except when `.data` has been previously grouped with `.drop = FALSE`. See `group_by_drop_default()` for details.

Details

See **Details** on `dplyr::group_by()`.

Value

A `SpatVector` object with updated grouping metadata.

Methods

Implementation of the **generic** `dplyr::group_by()` method family for `SpatVector` objects.

Grouping metadata

Mixing **terra** and **dplyr** syntax on a grouped or row-wise `SpatVector`, for example by subsetting with `v[1:3, 1:2]`, can corrupt its grouping metadata. **tidyterra** attempts to restore this metadata the next time you use a **dplyr** verb on the object.

Some operations, such as `terra::spatSample()`, create a new `SpatVector` without preserving grouping metadata. Call `group_by.SpatVector()` or `rowwise.SpatVector()` again, as appropriate.

See Also

`dplyr::group_by()`, `dplyr::ungroup()`.

Other **dplyr** verbs that operate on groups of rows: `count.SpatVector()`, `reframe.SpatVector()`, `rowwise.SpatVector()`, `summarise.SpatVector()`

Other **dplyr** grouping methods: `group_data.SpatVector()`, `group_map.SpatVector()`, `group_nest.SpatVector()`, `group_split.SpatVector()`, `group_trim.SpatVector()`

Examples

```
library(terra)
f <- system.file("ex/lux.shp", package = "terra")
p <- vect(f)

by_name1 <- p |> group_by(NAME_1)

# Grouping does not change how the SpatVector looks.
by_name1

# But it adds metadata for grouping. See the coercion to tibble.

# Not grouped.
p_tbl <- as_tibble(p)
class(p_tbl)
```

```
head(p_tbl, 3)

# Grouped.
by_name1_tbl <- as_tibble(by_name1)
class(by_name1_tbl)
head(by_name1_tbl, 3)

# It changes how it acts with the other dplyr verbs:
by_name1 |> summarise(
  pop = mean(POP),
  area = sum(AREA)
)

# Each call to summarise() removes a layer of grouping.
by_name2_name1 <- p |> group_by(NAME_2, NAME_1)

by_name2_name1
group_data(by_name2_name1)

by_name2 <- by_name2_name1 |> summarise(n = dplyr::n())
by_name2
group_data(by_name2)

# To remove grouping, use ungroup().
by_name2 |>
  ungroup() |>
  summarise(n = sum(n))

# By default, group_by() overrides existing grouping.
by_name2_name1 |>
  group_by(ID_1, ID_2) |>
  group_vars()

# Use `.add = TRUE` to append instead.
by_name2_name1 |>
  group_by(ID_1, ID_2, .add = TRUE) |>
  group_vars()

# You can group by expressions. This is shorthand for a mutate() followed
# by a group_by().
p |>
  group_by(ID_COMB = ID_1 * 100 / ID_2) |>
  relocate(ID_COMB, .before = 1)
```

---

hypsometric_tints_db *Hypsometric palettes database*

Description

A tibble including the color map of 34 gradient palettes. Each palette also includes a definition of color limits in terms of elevation (meters) that can be used with `ggplot2::scale_fill_gradientn()`.

Format

A tibble of 1223 rows and 6 columns with the following fields:

pal Name of the palette.

limit Recommended elevation limit (in meters) for each color.

r Value of the red channel (RGB color mode).

g Value of the green channel (RGB color mode).

b Value of the blue channel (RGB color mode).

hex Hex code of the color.

Source

cpt-city: https://phillips.shef.ac.uk/pub/cpt-city/.

See Also

`scale_fill_hypso_c()`.

Other datasets: `cross_blended_hypsometric_tints_db`, `grass_db`, `princess_db`, `volcano2`

Examples

```
data("hypsometric_tints_db")

hypsometric_tints_db

# Select a palette
wikicols <- hypsometric_tints_db |>
  filter(pal == "wiki-2.0")

f <- system.file("extdata/asia.tif", package = "tidyterra")
r <- terra::rast(f)

library(ggplot2)

p <- ggplot() +
  geom_spatraster(data = r) +
  labs(fill = "elevation")

p +
  scale_fill_gradientn(colors = wikicols$hex)

# Use with limits
p +
  scale_fill_gradientn(
```

```
    colors = wikicols$hex,
    values = scales::rescale(wikicols$limit),
    limit = range(wikicols$limit)
  )
```

`is_regular_grid` *Check whether* `x` *and* `y` *positions form a regular grid*

Description

Assess whether the `x` and `y` coordinates of an object form a regular grid. This function is called for its side effects.

This function is called internally by `as_spatraster()`.

Usage

```
is_regular_grid(xy, digits = 6)
```

Arguments

`xy`	A matrix, data frame or tibble with at least two columns representing `x` and `y` coordinates.
`digits`	Integer to set the precision for detecting whether points are on a regular grid (a low number of digits is a low precision).

Value

`NULL`, returned invisibly if the coordinates form a regular grid. Otherwise, the function raises an error.

See Also

`as_spatraster()`.

Examples

```
p <- matrix(1:90, nrow = 45, ncol = 2)

is_regular_grid(p)

# Jitter locations.
set.seed(1234)
jitter <- runif(length(p)) / 10e4
p_jitter <- p + jitter

# Adjust digits.
is_regular_grid(p_jitter, digits = 4)
```

mutate-joins.SpatVector

Mutating joins for SpatVector *objects*

Description

Mutating joins add columns from y to x, matching observations based on the keys. The four mutating joins are: inner join, left join, right join and full join.

See dplyr::inner_join() for details.

Usage

```
## S3 method for class 'SpatVector'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)

## S3 method for class 'SpatVector'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)

## S3 method for class 'SpatVector'
right_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)

## S3 method for class 'SpatVector'
```

```
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)
```

Arguments

x	A SpatVector created with terra::vect().
y	A data frame or other object coercible to a data frame. If a SpatVector or sf object is provided, this method returns an error. See terra::intersect() for spatial joins.
by	A join specification created with join_by(), or a character vector of variables to join by. If NULL, the default, *_join() will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying by explicitly. To join on different variables between x and y, use a join_by() specification. For example, join_by(a == b) will match x$a to y$b. To join by multiple variables, use a join_by() specification with multiple expressions. For example, join_by(a == b, c == d) will match x$a to y$b and x$c to y$d. If the column names are the same between x and y, you can shorten this by listing only the variable names, like join_by(a, c). join_by() can also be used to perform inequality, rolling, and overlap joins. See the documentation at ?join_by for details on these types of joins. For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, by = c("a", "b") joins x$a to y$a and x$b to y$b. If variable names differ between x and y, use a named character vector like by = c("x_a" = "y_a", "x_b" = "y_b"). To perform a cross-join, generating all combinations of x and y, see cross_join().
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
...	Other parameters passed onto methods.
keep	Should the join keys from both x and y be preserved in the output? • If NULL, the default, joins on equality retain only the keys from x, while joins on inequality retain the keys from both inputs. • If TRUE, all keys from both inputs are retained. • If FALSE, only keys from x are retained. For right and full joins, the data in key columns corresponding to rows that only exist in y are merged into the key columns from x. Can't be used when joining on inequality conditions.

Value

A SpatVector object.

terra equivalent

terra::merge().

Methods

Implementation of the **generic** dplyr::inner_join() method family for SpatVector objects.

The geometry column has sticky behavior. This means that the result always has the geometry of x for the records that match the join conditions.

For right_join() and full_join(), empty geometries may be returned (since y is expected to be a data frame with no geometries). Although these join operations are not common in spatial workflows, the function may crash because handling of EMPTY geometries differs between **terra** and **sf**.

See Also

dplyr::inner_join(), dplyr::left_join(), dplyr::right_join(), dplyr::full_join(), terra::merge().

Other **dplyr** verbs that operate on pairs of SpatVector and data frame objects: bind_cols.SpatVector, bind_rows.SpatVector, cross_join.SpatVector(), filter-joins.SpatVector, nest_join.SpatVector(), rows.SpatVector

Examples

```
library(terra)
library(ggplot2)
# Vector
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

# A data frame
df <- data.frame(
  cpro = sprintf("%02d", 1:10),
  x = runif(10),
  y = runif(10),
  letter = rep_len(LETTERS[1:3], length.out = 10)
)

# Inner join
inner <- v |> inner_join(df)

nrow(inner)
autoplot(inner, aes(fill = letter)) + labs(title = "Inner Join")

# Left join

left <- v |> left_join(df)
nrow(left)
```

```
autoplot(left, aes(fill = letter)) + labs(title = "Left Join")


# Right join
right <- v |> right_join(df)
nrow(right)

autoplot(right, aes(fill = letter)) + labs(title = "Right Join")

# Check empty geometries against data from the data frame.
ggplot(right, aes(x, y)) +
  geom_point(aes(color = letter))

# Full join
full <- v |> full_join(df)
nrow(full)

autoplot(full, aes(fill = letter)) + labs(title = "Full Join")

# Check against data from the data frame.
ggplot(full, aes(x, y)) +
  geom_point(aes(color = letter))
```

`mutate.Spat` *Create, modify and delete cell values/layers/attributes of* `Spat*` *objects*

Description

`mutate()` adds new layers/attributes and preserves existing ones on a `Spat*` object.

Usage

```
## S3 method for class 'SpatRaster'
mutate(
  .data,
  ...,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)

## S3 method for class 'SpatVector'
mutate(
  .data,
  ...,
  .by = NULL,
  .keep = c("all", "used", "unused", "none"),
```

```
  .before = NULL,
  .after = NULL
)
```

Arguments

.data	A SpatRaster created with terra::rast() or a SpatVector created with terra::vect().
...	<data-masking> Name-value pairs. The name gives the name of the column in the output. The value can be: • A vector of length 1, which will be recycled to the correct length. • A vector the same length as the current group (or the whole data frame if ungrouped). • NULL, to remove the column. • A data frame or tibble, to create multiple columns in the output.
.keep	Control which columns from .data are retained in the output. Grouping columns and columns created by ... are always kept. • "all" retains all columns from .data. This is the default. • "used" retains only the columns used in ... to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side. • "unused" retains only the columns *not* used in ... to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them. • "none" doesn't retain any extra columns from .data. Only the grouping variables and columns created by ... are kept.
.before, .after	<tidy-select> Optionally, control where new columns should appear (the default is to add to the right hand side). See relocate() for more details.
.by	<tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.

Value

A Spat* object of the same class as .data. See **Methods**.

terra equivalent

Some **terra** methods for modifying cell values: terra::ifel(), terra::classify(), terra::clamp(), terra::app(), terra::lapp() and terra::tapp().

Methods

Implementation of the **generic** dplyr::mutate() methods for Spat* objects.

SpatRaster**:**
Adds new layers and preserves existing ones. The result is a SpatRaster with the same extent, resolution and CRS as .data. Only the values and possibly the number of layers are modified.

SpatVector:

The result is a SpatVector with the attributes modified and possibly renamed as specified in the function call.

See Also

dplyr::mutate() methods.

terra provides several ways to modify Spat* objects:

- terra::ifel().
- terra::classify().
- terra::clamp().
- terra::app(), terra::lapp(), terra::tapp().

Other **dplyr** verbs that operate on columns: glimpse.Spat, pull.Spat, relocate.Spat, rename.Spat, select.Spat

Examples

```
library(terra)

# SpatRaster method
f <- system.file("extdata/cyl_temp.tif", package = "tidyterra")
spatrast <- rast(f)

mod <- spatrast |>
  mutate(exp_lyr1 = exp(tavg_04 / 10)) |>
  select(tavg_04, exp_lyr1)

mod
plot(mod)

# SpatVector method
f <- system.file("extdata/cyl.gpkg", package = "tidyterra")
v <- vect(f)

v |>
  mutate(cpro2 = paste0(cpro, "-CyL")) |>
  select(cpro, cpro2)
```

---

nest.SpatVector *Nest* SpatVector *rows*

Description

[Experimental]

nest() creates list-columns of SpatVector objects.

Usage

```
## S3 method for class 'SpatVector'
nest(.data, ..., .by = NULL, .key = NULL, .names_sep = NULL)
```

Arguments

.data	A SpatVector.
...	<tidy-select> Columns to nest; these will appear in the inner data frames. Specified using name-variable pairs of the form new_col = c(col1, col2, col3). The right hand side can be any valid tidyselect expression. If not supplied, then ... is derived as all columns *not* selected by .by, and will use the column name from .key. **[Deprecated]**: previously you could write df |> nest(x, y, z). Convert to df |> nest(data = c(x, y, z)).
.by	<tidy-select> Columns to nest *by*; these will remain in the outer data frame. .by can be used in place of or in conjunction with columns supplied through .... If not supplied, then .by is derived as all columns *not* selected by ....
.key	The name of the resulting nested column. Only applicable when ... isn't specified, i.e. in the case of df |> nest(.by = x). If NULL, then "data" will be used by default.
.names_sep	If NULL, the default, the inner names will come from the former outer names. If a string, the new inner names will use the outer names with names_sep automatically stripped. This makes names_sep roughly symmetric between nesting and unnesting.

Value

A tibble with one or more list-columns of SpatVector objects.

terra equivalent

terra::svc().

Methods

Implementation of the **generic** tidyr::nest() method for SpatVector objects.

The geometry column must be nested with the other attributes that form each nested SpatVector. These nested list-columns contain SpatVector objects and cannot be passed directly to tidyr::unnest().

See Also

tidyr::nest(), terra::svc().

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

v |>
  group_by(cpro) |>
  nest()

# Convert to a named SpatVectorCollection.
nested <- nest(v, .by = cpro)

sv <- pull(nested, data)
names(sv) <- pull(nested, cpro)

terra::svc(sv)
```

---

nest_join.SpatVector *Nest join* SpatVector *objects*

Description

[Experimental]

`nest_join()` returns a tibble with the attributes and geometry of `x`, plus a list-column containing matching rows from `y`.

Usage

```
## S3 method for class 'SpatVector'
nest_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  keep = NULL,
  name = NULL,
  ...,
  na_matches = c("na", "never")
)
```

Arguments

`x`	A `SpatVector`.
`y`	A data frame. Spatial `y` inputs are not supported, use spatial joins from **terra** for that workflow.
`by`	A join specification created with `join_by()`, or a character vector of variables to join by.

If NULL, the default, *_join() will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying by explicitly.

To join on different variables between x and y, use a join_by() specification. For example, join_by(a == b) will match x$a to y$b.

To join by multiple variables, use a join_by() specification with multiple expressions. For example, join_by(a == b, c == d) will match x$a to y$b and x$c to y$d. If the column names are the same between x and y, you can shorten this by listing only the variable names, like join_by(a, c).

join_by() can also be used to perform inequality, rolling, and overlap joins. See the documentation at ?join_by for details on these types of joins.

For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, by = c("a", "b") joins x$a to y$a and x$b to y$b. If variable names differ between x and y, use a named character vector like by = c("x_a" = "y_a", "x_b" = "y_b").

To perform a cross-join, generating all combinations of x and y, see cross_join().

copy — If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

keep — Should the new list-column contain join keys? The default will preserve the join keys for inequality joins.

name — The name of the list-column created by the join. If NULL, the default, the name of y is used.

... — Other parameters passed onto methods.

na_matches — Should two NA or two NaN values match?

- "na", the default, treats two NA or two NaN values as equal, like %in%, match(), and merge().
- "never" treats two NA or two NaN values as different, and will never match them together or to any other values. This is similar to joins for database sources and to base::merge(incomparables = NA).

Value

A tibble.

Methods

Implementation of the **generic** dplyr::nest_join() method for SpatVector objects.

The output is a tibble with the attributes and WKT geometry of x, plus a list-column with matching rows from y.

See Also

dplyr::nest_join().

Other **dplyr** verbs that operate on pairs of SpatVector and data frame objects: bind_cols.SpatVector, bind_rows.SpatVector, cross_join.SpatVector(), filter-joins.SpatVector, mutate-joins.SpatVector, rows.SpatVector

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
extra <- tibble::tibble(cpro = c("05", "09"), value = c(1, 2))

nest_join(v, extra, by = "cpro")
```

---

pivot_longer.SpatVector

Pivot SpatVector *from wide to long*

Description

pivot_longer() "lengthens" a SpatVector, increasing the number of rows and decreasing the number of columns. The inverse transformation is pivot_wider.SpatVector().

Learn more in tidyr::pivot_longer().

Usage

```
## S3 method for class 'SpatVector'
pivot_longer(
  data,
  cols,
  ...,
  cols_vary = "fastest",
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = NULL,
  names_transform = NULL,
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = NULL,
  values_transform = NULL
)
```

Arguments

data	A SpatVector to pivot.
cols	<tidy-select> Columns to pivot into longer format.
...	Additional arguments passed on to methods.
cols_vary	When pivoting cols into longer format, how should the output rows be arranged relative to their original row number?

- `"fastest"`, the default, keeps individual rows from `cols` close together in the output. This often produces intuitively ordered output when you have at least one key column from `data` that is not involved in the pivoting process.
- `"slowest"` keeps individual columns from `cols` close together in the output. This often produces intuitively ordered output when you utilize all of the columns from `data` in the pivoting process.

`names_to` A character vector specifying the new column or columns to create from the information stored in the column names of `data` specified by `cols`.

- If length 0, or if `NULL` is supplied, no columns will be created.
- If length 1, a single column will be created which will contain the column names specified by `cols`.
- If length >1, multiple columns will be created. In this case, one of `names_sep` or `names_pattern` must be supplied to specify how the column names should be split. There are also two additional character values you can take advantage of:
  - `NA` will discard the corresponding component of the column name.
  - `".value"` indicates that the corresponding component of the column name defines the name of the output column containing the cell values, overriding `values_to` entirely.

`names_prefix` A regular expression used to remove matching text from the start of each variable name.

`names_sep, names_pattern`

If `names_to` contains multiple values, these arguments control how the column name is broken up.

`names_sep` takes the same specification as `separate()`, and can either be a numeric vector (specifying positions to break on), or a single string (specifying a regular expression to split on).

`names_pattern` takes the same specification as `extract()`, a regular expression containing matching groups (`()`).

If these arguments do not give you enough control, use `pivot_longer_spec()` to create a spec object and process manually as needed.

`names_ptypes, values_ptypes`

Optionally, a list of column name-prototype pairs. Alternatively, a single empty prototype can be supplied, which will be applied to all columns. A prototype (or ptype for short) is a zero-length vector (like `integer()` or `numeric()`) that defines the type, class, and attributes of a vector. Use these arguments if you want to confirm that the created columns are the types that you expect. Note that if you want to change (instead of confirm) the types of specific columns, you should use `names_transform` or `values_transform` instead.

`names_transform, values_transform`

Optionally, a list of column name-function pairs. Alternatively, a single function can be supplied, which will be applied to all columns. Use these arguments if you need to change the types of specific columns. For example, `names_transform = list(week = as.integer)` would convert a character variable called `week` to an integer.

If not specified, the type of the columns generated from `names_to` will be character, and the type of the variables generated from `values_to` will be the common type of the input columns used to generate them.

`names_repair` What happens if the output has invalid column names? The default, `"check_unique"` is to error if the columns are duplicated. Use `"minimal"` to allow duplicates in the output, or `"unique"` to de-duplicated by adding numeric suffixes. See `vctrs::vec_as_names()` for more options.

`values_to` A string specifying the name of the column to create from the data stored in cell values. If `names_to` is a character containing the special `.value` sentinel, this value will be ignored, and the name of the value column will be derived from part of the existing column names.

`values_drop_na` If `TRUE`, will drop rows that contain only `NA`s in the `values_to` column. This effectively converts explicit missing values to implicit missing values, and should generally be used only when missing values in `data` were created by its structure.

Value

A `SpatVector` object.

Methods

Implementation of the **generic** `tidyr::pivot_longer()` method for `SpatVector` objects.

The geometry column has sticky behavior. This means that the result always has the geometry of `data`.

See Also

`tidyr::pivot_longer()`.

Other **tidyr** verbs for pivoting: `pivot_wider.SpatVector()`

Examples

```
library(dplyr)
library(tidyr)
library(ggplot2)
library(terra)

temp <- rast((system.file("extdata/cyl_temp.tif", package = "tidyterra")))
cyl <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra")) |>
  project(temp)

# Add average temperature.

temps <- terra::extract(temp, cyl, fun = "mean", na.rm = TRUE, xy = TRUE)
cyl_temp <- cbind(cyl, temps) |>
  glimpse()
```

```
# Pivot to long format for plotting.
cyl_temp |>
  pivot_longer(
    cols = tavg_04:tavg_06,
    names_to = "label",
    values_to = "temp"
  ) |>
  ggplot() +
  geom_spatvector(aes(fill = temp)) +
  facet_wrap(~label, ncol = 1) +
  scale_fill_whitebox_c(palette = "muted")
```

`pivot_wider.SpatVector`

Pivot `SpatVector` *from long to wide*

Description

`pivot_wider()` "widens" a `SpatVector`, increasing the number of columns and decreasing the number of rows. The inverse transformation is `pivot_longer.SpatVector()`.

Usage

```
## S3 method for class 'SpatVector'
pivot_wider(
  data,
  ...,
  id_cols = NULL,
  id_expand = FALSE,
  names_from = "name",
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_vary = "fastest",
  names_expand = FALSE,
  names_repair = "check_unique",
  values_from = "value",
  values_fill = NULL,
  values_fn = NULL,
  unused_fn = NULL
)
```

Arguments

`data`	A `SpatVector` to pivot.
`...`	Additional arguments passed on to methods.

id_cols
: <tidy-select> A set of columns that uniquely identify each observation. Typically used when you have redundant variables, that is, variables whose values are perfectly correlated with existing variables.

  Defaults to all columns in data except for the columns specified through names_from and values_from. If a tidyselect expression is supplied, it will be evaluated on data after removing the columns specified through names_from and values_from.

  Because "geometry" columns are sticky, they are removed from names_from and values_from.

id_expand
: Should the values in the id_cols columns be expanded by expand() before pivoting? This results in more rows, the output will contain a complete expansion of all possible values in id_cols. Implicit factor levels that aren't represented in the data will become explicit. Additionally, the row values corresponding to the expanded id_cols will be sorted.

names_from, values_from
: <tidy-select> A pair of arguments describing which column (or columns) to get the name of the output column (names_from), and which column (or columns) to get the cell values from (values_from).

  If values_from contains multiple values, the value will be added to the front of the output column.

names_prefix
: A regular expression used to remove matching text from the start of each variable name.

names_sep
: If names_from or values_from contains multiple variables, this will be used to join their values together into a single string to use as a column name.

names_glue
: Instead of names_sep and names_prefix, you can supply a glue specification that uses the names_from columns (and special .value) to create custom column names.

names_sort
: Should the column names be sorted? If FALSE, the default, column names are ordered by first appearance.

names_vary
: When names_from identifies a column (or columns) with multiple unique values, and multiple values_from columns are provided, in what order should the resulting column names be combined?

  - "fastest" varies names_from values fastest, resulting in a column naming scheme of the form: value1_name1, value1_name2, value2_name1, value2_name2. This is the default.
  - "slowest" varies names_from values slowest, resulting in a column naming scheme of the form: value1_name1, value2_name1, value1_name2, value2_name2.

names_expand
: Should the values in the names_from columns be expanded by expand() before pivoting? This results in more columns, the output will contain column names corresponding to a complete expansion of all possible values in names_from. Implicit factor levels that aren't represented in the data will become explicit. Additionally, the column names will be sorted, identical to what names_sort would produce.

names_repair
: What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates

in the output, or "unique" to de-duplicated by adding numeric suffixes. See vctrs::vec_as_names() for more options.

values_fill — Optionally, a (scalar) value that specifies what each value should be filled in with when missing.

This can be a named list if you want to apply different fill values to different value columns.

values_fn — Optionally, a function applied to the value in each cell in the output. You will typically use this when the combination of id_cols and names_from columns does not uniquely identify an observation.

This can be a named list if you want to apply different aggregations to different values_from columns.

unused_fn — Optionally, a function applied to summarize the values from the unused columns (i.e. columns not identified by id_cols, names_from, or values_from).

The default drops all unused columns from the result.

This can be a named list if you want to apply different aggregations to different unused columns.

id_cols must be supplied for unused_fn to be useful, since otherwise all unspecified columns will be considered id_cols.

This is similar to grouping by the id_cols then summarizing the unused columns using unused_fn.

Value

A SpatVector object.

Methods

Implementation of the **generic** tidyr::pivot_wider() method for SpatVector objects.

The geometry column has sticky behavior. This means that the result always has the geometry of data.

See Also

tidyr::pivot_wider().

Other **tidyr** verbs for pivoting: pivot_longer.SpatVector()

Examples

```
library(dplyr)
library(tidyr)
library(ggplot2)

cyl <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

# Add an extra row with information.
xtra <- cyl |>
  slice(c(2, 3)) |>
  mutate(
```

```
    label = "extra",
    value = TRUE
  ) |>
  rbind(cyl) |>
  glimpse()

# Pivot by geometry.
xtra |>
  pivot_wider(
    id_cols = iso2:name, values_from = value,
    names_from = label
  )
```

`princess_db` *Princess palettes database*

Description

A tibble including the color map of 15 gradient palettes.

Format

A tibble of 75 rows and 5 columns with the following fields:

pal Name of the palette.

r Value of the red channel (RGB color mode).

g Value of the green channel (RGB color mode).

b Value of the blue channel (RGB color mode).

hex Hex code of the color.

Source

https://leahsmyth.github.io/Princess-Colour-Schemes/index.html.

See Also

`scale_fill_princess_c()`.

Other datasets: `cross_blended_hypsometric_tints_db`, `grass_db`, `hypsometric_tints_db`, `volcano2`

Examples

```
data("princess_db")

princess_db

# Select a palette
maori <- princess_db |>
  filter(pal == "maori")

f <- system.file("extdata/volcano2.tif", package = "tidyterra")
r <- terra::rast(f)

library(ggplot2)

p <- ggplot() +
  geom_spatraster(data = r) +
  labs(fill = "elevation")

p +
  scale_fill_gradientn(colors = maori$hex)
```

`pull.Spat` *Extract a single layer/attribute*

Description

`pull()` is similar to `$` on a data frame. It is mostly useful because it looks nicer in pipes and can optionally name the output.

You can extract the geographic coordinates of a `SpatRaster`. Use `pull(.data, x, xy = TRUE)`. `x` and `y` are reserved names in **terra**, since they refer to cell coordinates.

See **Examples** and section **Layer and column names** on `as_tibble.Spat()`.

Usage

```
## S3 method for class 'SpatRaster'
pull(.data, var = -1, name = NULL, ...)

## S3 method for class 'SpatVector'
pull(.data, var = -1, name = NULL, ...)
```

Arguments

`.data`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`var`	A variable specified as: • a literal layer/attribute name. • a positive integer, giving the position counting from the left.

- a negative integer, giving the position counting from the right.

The default returns the last layer/attribute (on the assumption that's the column you've created most recently).

This argument is taken by expression and supports quasiquotation (you can unquote column names and column locations).

`name` An optional parameter that specifies the column to be used as names for a named vector. Specified in a similar manner as `var`.

`...` Arguments passed to `as_tibble.SpatRaster()` or `as_tibble.Spat()` methods.

Value

A vector with one element per cell or geometry in `.data`. For `SpatRaster` objects, the default (`na.rm = FALSE`) keeps missing values. Passing `na.rm = TRUE` to `...` removes cells with missing values and can shorten the result. See `terra::as.data.frame()`.

terra equivalent

`terra::values()`.

Methods

Implementation of the **generic** `dplyr::pull()` methods. Each method first coerces the `Spat*` object to a tibble (see as_tibble.Spat) and then applies `dplyr::pull()` to the tibble.

`SpatRaster`**:**

When passing `na.rm = TRUE` to `...`, only cells with a value other than `NA` are extracted. See `terra::as.data.frame()`.

If `xy = TRUE` is passed to `...`, two columns named `x` and `y` (corresponding to the geographic coordinates of each cell) are available in positions `1` and `2`. Therefore, `pull(.data, 1)` and `pull(.data, 1, xy = TRUE)` return different results.

`SpatVector`**:**

When passing `geom = "WKT"` or `geom = "HEX"` to `...`, the geometry of the `SpatVector` can be extracted with `var = geometry`. Similarly, when using `geom = "XY"`, the coordinates can be extracted with `var = x` or `var = y`. See the options in `terra::as.data.frame()`.

See Also

`dplyr::pull()`.

Other **dplyr** verbs that operate on columns: `glimpse.Spat`, `mutate.Spat`, `relocate.Spat`, `rename.Spat`, `select.Spat`

Examples

```
library(terra)
f <- system.file("extdata/cyl_tile.tif", package = "tidyterra")
r <- rast(f)
```

```
# Extract the second layer.
r |>
  pull(2) |>
  head()

# With `xy`, the first two columns are `x` (longitude) and `y` (latitude).

r |>
  pull(2, xy = TRUE) |>
  head()

# With renaming

r |>
  mutate(cat = cut(cyl_tile_3, c(0, 100, 300))) |>
  pull(cyl_tile_3, name = cat) |>
  head()
```

`pull_crs` *Extract CRS in WKT format*

Description

Extract the WKT version of the CRS associated with a string, number, `sf` object or `Spat*` object. Well-known text (WKT) is a character string representation of coordinate reference systems (CRS). It identifies the parameters of each CRS precisely and is the standard used by **sf** and **terra**.

Usage

```
pull_crs(.data, ...)
```

Arguments

`.data`	Input potentially including or representing a CRS. It could be an `sf` or `sfc` object, a `SpatRaster` or `SpatVector` object, a `crs` object from `sf::st_crs()`, a character string (for example a PROJ string) or an integer (representing an EPSG code).
`...`	Ignored.

Details

Although the WKT representation is the same, the **sf** and **terra** APIs differ slightly. For example, **sf** can do:

```
sf::st_transform(x, 25830)
```

While the **terra** equivalent is:

```
terra::project(bb, "epsg:25830")
```

Knowing the WKT helps smooth workflows when working with different packages and object types. Assigning a CRS records how existing coordinates should be interpreted. Establish the source CRS from the data provider before assigning it. Reprojecting with `terra::project()` or `sf::st_transform()` transforms coordinates from a known source CRS to a target CRS.

Value

A character string containing the CRS in WKT format, or `NA` if no CRS can be determined.

Internals

A thin wrapper around `sf::st_crs()` and `terra::crs()`.

See Also

`terra::crs()` and `sf::st_crs()` to learn how these packages handle CRS definitions.

Examples

```
# sf objects.

sfobj <- sf::st_as_sfc("MULTIPOINT ((0 0), (1 1))", crs = 4326)

fromsf1 <- pull_crs(sfobj)
fromsf2 <- pull_crs(sf::st_crs(sfobj))

# terra objects.

v <- terra::vect(sfobj)
r <- terra::rast(v)

fromterra1 <- pull_crs(v)
fromterra2 <- pull_crs(r)

# Integers.
fromint <- pull_crs(4326)

# Characters.
fromchar <- pull_crs("epsg:4326")

all(
  fromsf1 == fromsf2,
  fromsf2 == fromterra1,
  fromterra1 == fromterra2,
  fromterra2 == fromint,
  fromint == fromchar
)

cat(fromsf1)
```

`reframe.SpatVector` *Reframe each group of a* `SpatVector`

Description

[Experimental]

`reframe()` can return any number of rows per group. The geometry of each group is aggregated and repeated for each row created for that group.

Usage

```
## S3 method for class 'SpatVector'
reframe(.data, ..., .by = NULL, .dissolve = TRUE)
```

Arguments

`.data`	A `SpatVector` created with `terra::vect()`.
`...`	<`data-masking`> Name-value pairs of summary functions. The name will be the name of the variable in the result. The value can be: • A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`. • A data frame with 1 row, to add multiple columns from a single expression.
`.by`	<`tidy-select`> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.
`.dissolve`	Logical. If `TRUE`, dissolve borders between aggregated geometries.

Value

A `SpatVector` object.

Methods

Implementation of the **generic** `dplyr::reframe()` method for `SpatVector` objects.

For grouped inputs and calls using `.by`, geometries are aggregated per group. If a group produces more than one row, the aggregated group geometry is repeated for each output row.

See Also

`dplyr::reframe()`.

Other **dplyr** verbs that operate on groups of rows: `count.SpatVector()`, `group_by.SpatVector()`, `rowwise.SpatVector()`, `summarise.SpatVector()`

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
v$grp <- rep(c("A", "B"), length.out = nrow(v))

v |>
  reframe(value = c(min(as.double(cpro)), max(as.double(cpro))), .by = grp)

v |>
  rowwise() |>
  reframe(value = 1:2)
```

---

relocate.Spat *Change layer/attribute order*

Description

Use `relocate()` to change layer/attribute positions, using the same syntax as `select.Spat` to make it easy to move blocks of layers/attributes at once.

Usage

```
## S3 method for class 'SpatRaster'
relocate(.data, ..., .before = NULL, .after = NULL)

## S3 method for class 'SpatVector'
relocate(.data, ..., .before = NULL, .after = NULL)
```

Arguments

`.data`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`...`	<tidy-select> layers/attributes to move.
`.before, .after`	<tidy-select> Destination of layers/attributes selected by `...`. Supplying neither will move layers/attributes to the left-hand side. Specifying both is an error.

Value

A `Spat*` object of the same class as `.data`. See **Methods**.

terra equivalent

`terra::subset(data, c("name_layer", "name_other_layer"))`.

Methods

Implementation of the **generic** dplyr::relocate() methods for Spat* objects.

SpatRaster**:**
Relocates layers of a SpatRaster.

SpatVector**:**
The result is a SpatVector with the attributes in a different order.

See Also

dplyr::relocate().

Other **dplyr** verbs that operate on columns: glimpse.Spat, mutate.Spat, pull.Spat, rename.Spat, select.Spat

Examples

```
library(terra)

f <- system.file("extdata/cyl_tile.tif", package = "tidyterra")
spatrast <- rast(f) |> mutate(aa = 1, bb = 2, cc = 3)

names(spatrast)

spatrast |>
  relocate(bb, .before = cyl_tile_3) |>
  relocate(cyl_tile_1, .after = last_col())
```

---

rename.Spat *Rename layers/attributes*

Description

rename() changes the names of individual layers/attributes using new_name = old_name syntax. rename_with() renames layers/attributes using a function.

Usage

```
## S3 method for class 'SpatRaster'
rename(.data, ...)

## S3 method for class 'SpatRaster'
rename_with(.data, .fn, .cols = everything(), ...)

## S3 method for class 'SpatVector'
rename(.data, ...)

## S3 method for class 'SpatVector'
rename_with(.data, .fn, .cols = everything(), ...)
```

Arguments

.data	A SpatRaster created with terra::rast() or a SpatVector created with terra::vect().
...	For rename.Spat*(): <tidy-select> Use new_name = old_name to rename selected variables. For rename_with.Spat*(): additional arguments passed onto .fn.
.fn	A function used to transform the selected .cols. Should return a character vector the same length as the input.
.cols	<tidy-select> Columns to rename; defaults to all columns.

Value

A Spat* object of the same class as .data. See **Methods**.

terra equivalent

names(Spat*) <- c("a", "b", "c").

Methods

Implementation of the **generic** dplyr::rename() methods for Spat* objects.

SpatRaster**:**
Renames layers of a SpatRaster.

SpatVector**:**
The result is a SpatVector with the attributes renamed as specified in the function call.

See Also

dplyr::rename().

Other **dplyr** verbs that operate on columns: glimpse.Spat, mutate.Spat, pull.Spat, relocate.Spat, select.Spat

Examples

```
library(terra)
f <- system.file("extdata/cyl_tile.tif", package = "tidyterra")
spatrast <- rast(f) |> mutate(aa = 1, bb = 2, cc = 3)

spatrast

spatrast |> rename(
  this_first = cyl_tile_1,
  this_second = cyl_tile_2
)

spatrast |> rename_with(
  toupper,
  .cols = starts_with("c")
)
```

`replace_na.Spat` *Replace missing values with specified values*

Description

Replace `NA` values in layers/attributes with specified values.

Usage

```
## S3 method for class 'SpatRaster'
replace_na(data, replace = list(), ...)

## S3 method for class 'SpatVector'
replace_na(data, replace, ...)
```

Arguments

`data`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`replace`	A named list of values, with one value for each layer/attribute that has missing values to be replaced. Each value in `replace` will be cast to the type of the column in `data` where it is used as a replacement.
`...`	Additional arguments for methods. Currently unused.

Value

A `Spat*` object of the same class as `data`. See **Methods**.

terra equivalent

Use `r[is.na(r)] <- <replacement>`.

See Also

`tidyr::replace_na()`.

Other **tidyr** verbs for handling missing values: `complete.SpatVector()`, `drop_na.Spat`, `expand.SpatVector()`, `fill.SpatVector()`

Examples

```
library(terra)

f <- system.file("extdata/cyl_temp.tif", package = "tidyterra")
r <- rast(f)

r |> plot()

r |>
  replace_na(list(tavg_04 = 6, tavg_06 = 20)) |>
```

```
plot()
```

`required_pkgs.Spat` *Determine packages required by* `Spat*` *objects*

Description

Determine packages required by `Spat*` objects.

Usage

```
## S3 method for class 'SpatRaster'
required_pkgs(x, ...)

## S3 method for class 'SpatVector'
required_pkgs(x, ...)

## S3 method for class 'SpatGraticule'
required_pkgs(x, ...)

## S3 method for class 'SpatExtent'
required_pkgs(x, ...)
```

Arguments

`x`	A `SpatRaster` created with `terra::rast()`, a `SpatVector` created with `terra::vect()`, a `SpatGraticule` (see `terra::graticule()`) or a `SpatExtent` (see `terra::ext()`).
`...`	Ignored by these methods.

Value

A character vector of required package names.

Methods

Implementation of the **generic** `generics::required_pkgs()` method for `Spat*` objects.

See Also

`generics::required_pkgs()`.

Other **generics** methods: `glance.Spat`, `tidy.Spat`

Examples

```
file_path <- system.file("extdata/cyl_temp.tif", package = "tidyterra")

library(terra)

r <- rast(file_path)

# With rasters
r
required_pkgs(r)

# With vectors
v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
v
required_pkgs(v)
```

---

rows.SpatVector *Row operations for* SpatVector *objects*

Description

[Experimental]

Methods for the dplyr::rows_insert() family on SpatVector objects.

Usage

```
## S3 method for class 'SpatVector'
rows_insert(
  x,
  y,
  by = NULL,
  ...,
  conflict = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)

## S3 method for class 'SpatVector'
rows_append(x, y, ..., copy = FALSE, in_place = FALSE)

## S3 method for class 'SpatVector'
rows_update(
  x,
  y,
  by = NULL,
  ...,
```

```
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)

## S3 method for class 'SpatVector'
rows_patch(
  x,
  y,
  by = NULL,
  ...,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)

## S3 method for class 'SpatVector'
rows_upsert(x, y, by = NULL, ..., copy = FALSE, in_place = FALSE)

## S3 method for class 'SpatVector'
rows_delete(
  x,
  y,
  by = NULL,
  ...,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x	A SpatVector.
y	A data frame, sf object or SpatVector.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
conflict	For rows_insert(), how should keys in y that conflict with keys in x be handled? A conflict arises if there is a key in y that already exists in x. One of: • "error", the default, will error if there are any keys in y that conflict with keys in x. • "ignore" will ignore rows in y with keys that conflict with keys in x.

copy
: If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

in_place
: Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables).
When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

unmatched
: For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled?
One of:
- "error", the default, will error if there are any keys in y that are unmatched by the keys in x.
- "ignore" will ignore rows in y with keys that are unmatched by the keys in x.

Value

A SpatVector object.

Methods

Implementation of the **generic** dplyr::rows_insert() method family for SpatVector objects.

Row operations update attributes while preserving the geometry column. When inserting data frame rows without geometry, the output contains empty geometries for the new rows.

See Also

dplyr::rows_insert().

Other **dplyr** verbs that operate on rows: arrange.SpatVector(), distinct.SpatVector(), filter.Spat, slice.Spat

Other **dplyr** verbs that operate on pairs of SpatVector and data frame objects: bind_cols.SpatVector, bind_rows.SpatVector, cross_join.SpatVector(), filter-joins.SpatVector, mutate-joins.SpatVector, nest_join.SpatVector()

Examples

```
v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

rows_update(
  v,
  tibble::tibble(cpro = "05", name = "New name"),
  by = "cpro"
)

rows_insert(
  v,
  tibble::tibble(cpro = "99", name = "New province"),
  by = "cpro"
)
```

rowwise.SpatVector *Group* SpatVector *objects by rows*

Description

rowwise() lets you compute on a SpatVector one row at a time. This is most useful when a vectorized function does not exist.

Most **dplyr** verb implementations in **tidyterra** preserve row-wise grouping. The exception is summarise.SpatVector(), which returns a grouped SpatVector. You can explicitly ungroup with ungroup.SpatVector() or as_tibble(), or convert to a grouped SpatVector with group_by.SpatVector().

Usage

```
## S3 method for class 'SpatVector'
rowwise(data, ...)
```

Arguments

data	A SpatVector object. See **Methods**.
...	<tidy-select> Variables to be preserved when calling summarise.SpatVector(). This is typically a set of variables whose combination uniquely identifies each row. See dplyr::rowwise(). Unlike group_by.SpatVector(), you cannot create new variables here. Instead, you can select multiple variables, for example with everything().

Details

See **Details** on dplyr::rowwise().

Value

A SpatVector object with updated grouping metadata.

Methods

Implementation of the **generic** dplyr::rowwise() method for SpatVector objects.

Grouping metadata

Mixing **terra** and **dplyr** syntax on a grouped or row-wise SpatVector, for example by subsetting with v[1:3, 1:2], can corrupt its grouping metadata. **tidyterra** attempts to restore this metadata the next time you use a **dplyr** verb on the object.

Some operations, such as terra::spatSample(), create a new SpatVector without preserving grouping metadata. Call group_by.SpatVector() or rowwise.SpatVector() again, as appropriate.

See Also

dplyr::rowwise().

Other **dplyr** verbs that operate on groups of rows: count.SpatVector(), group_by.SpatVector(), reframe.SpatVector(), summarise.SpatVector()

Examples

```
library(terra)
library(dplyr)

v <- terra::vect(system.file("shape/nc.shp", package = "sf"))

# Select new births
nb <- v |>
  select(starts_with("NWBIR")) |>
  glimpse()

# Compute the mean of NWBIR for each geometry.
nb |>
  rowwise() |>
  mutate(nb_mean = mean(c(NWBIR74, NWBIR79)))

# Additional examples

# Use c_across() to select many variables more easily.
nb |>
  rowwise() |>
  mutate(m = mean(c_across(NWBIR74:NWBIR79)))

# Compute the minimum of x and y in each row

nb |>
  rowwise() |>
  mutate(min = min(c_across(NWBIR74:NWBIR79)))

# Summarize.
v |>
  rowwise() |>
  summarise(mean_bir = mean(BIR74, BIR79)) |>
  glimpse() |>
  autoplot(aes(fill = mean_bir))

# Supply a variable to be kept
v |>
  mutate(id2 = as.integer(CNTY_ID / 100)) |>
  rowwise(id2) |>
  summarise(mean_bir = mean(BIR74, BIR79)) |>
  glimpse() |>
  autoplot(aes(fill = as.factor(id2)))
```

scale_coltab *Discrete scales based on* SpatRaster *color tables*

Description

Some categorical SpatRaster objects may have an associated color table. These functions generate scales and color vectors based on the color table from terra::coltab() associated with a SpatRaster.

You can also get a vector of colors named with the corresponding factor with get_coltab_pal().

Additional arguments ... are passed to ggplot2::discrete_scale().

tidyterra documents only a selection of these additional arguments, so check ggplot2::discrete_scale() to see the full range of arguments accepted.

Usage

```
scale_fill_coltab(
  data,
  ...,
  alpha = NA,
  na.translate = FALSE,
  na.value = "transparent",
  drop = TRUE
)

scale_colour_coltab(
  data,
  ...,
  alpha = NA,
  na.translate = FALSE,
  na.value = "transparent",
  drop = TRUE
)

get_coltab_pal(x)
```

Arguments

data, x	A SpatRaster with one or more color tables. See terra::has.colors().
...	Arguments passed on to ggplot2::discrete_scale
	breaks One of:

- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`alpha` The alpha transparency: could be `NA` or a number in [0,1]. See argument `alpha` in `scale_fill_terrain_d()`.

`na.translate` Logical. If `TRUE`, remove `NA` values from the legend. The default is `FALSE`.

`na.value` Missing values will be replaced with this value. By default, **tidyterra** uses `na.value = "transparent"` so cells with `NA` are not filled. See also #120.

`drop` Logical. If `TRUE`, omit unused factor levels from the scale. The default (`TRUE`) removes unused factors.

Value

A **ggplot2** scale for the `fill` or `colour` aesthetic. If no color table is available, the scale functions return a blank layer. `get_coltab_pal()` returns a named character vector of colors, or `NULL` if `x` is not a `SpatRaster` object with a color table.

See Also

terra::coltab(), ggplot2::discrete_scale(), ggplot2::scale_fill_manual().

Other color scales, palettes and hypsometric tints: scale_cross_blended, scale_grass, scale_hypso, scale_princess, scale_terrain, scale_whitebox, scale_wiki

Examples

```
library(terra)
# Geological Eras
# Spanish Geological Survey (IGME)

r <- rast(system.file("extdata/cyl_era.tif", package = "tidyterra"))

plot(r)

# Get the color table palette.
coltab_pal <- get_coltab_pal(r)

coltab_pal


# With ggplot2 + tidyterra
library(ggplot2)

gg <- ggplot() +
  geom_spatraster(data = r)

# Default plot
gg

# With color tables
gg +
  scale_fill_coltab(data = r)
```

---

scale_cross_blended *Cross-blended hypsometric tint scales*

Description

Implementation of the cross-blended hypsometric gradients presented in doi:10.14714/CP69.20. The following fill scales and palettes are provided:

- scale_*_cross_blended_d(): For discrete values.
- scale_*_cross_blended_c(): For continuous values.
- scale_*_cross_blended_b(): For binning continuous values.
- cross_blended.colors(): A gradient color palette. See also grDevices::terrain.colors() for details.

An additional set of scales is provided. These scales can act as hypsometric (or bathymetric) tints.

- `scale_*_cross_blended_tint_d()`: For discrete values.
- `scale_*_cross_blended_tint_c()`: For continuous values.
- `scale_*_cross_blended_tint_b()`: For binning continuous values.
- `cross_blended.colors2()`: A gradient color palette. See also `grDevices::terrain.colors()` for details.

See **Details**.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a selection of these additional arguments, so check the **ggplot2** functions listed above to see the full range of arguments accepted by these scales.

Usage

```
scale_fill_cross_blended_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_cross_blended_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_cross_blended_c(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_cross_blended_c(
  palette = "cold_humid",
```

```
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_cross_blended_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_cross_blended_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

cross_blended.colors(n, palette = "cold_humid", alpha = 1, rev = FALSE)

scale_fill_cross_blended_tint_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_cross_blended_tint_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_cross_blended_tint_c(
  palette = "cold_humid",
  ...,
```

```
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_cross_blended_tint_c(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_cross_blended_tint_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_cross_blended_tint_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "coloursteps"
)

cross_blended.colors2(n, palette = "cold_humid", alpha = 1, rev = FALSE)
```

Arguments

palette	A valid palette name. The name is matched to the list of available palettes, ignoring upper vs. lower case. See cross_blended_hypsometric_tints_db for more information. The available values are listed below. "arid", "cold_humid", "polar", "warm_humid".

`...` Arguments passed on to `ggplot2::discrete_scale`, `ggplot2::continuous_scale`, `ggplot2::binned_scale`

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`n.breaks` An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation.

`nice.breaks` Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If `TRUE` (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

`alpha` The alpha transparency, a number in [0,1], see argument alpha in `hsv`.

`direction` Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.

`na.translate` Logical. If `TRUE`, remove `NA` values from the legend. The default is `FALSE`.

drop — Logical. If TRUE, omit unused factor levels from the scale. The default (TRUE) removes unused factors.

na.value — Missing values will be replaced with this value. By default, **tidyterra** uses na.value = "transparent" so cells with NA are not filled. See also #120.

guide — A function used to create a guide or its name. See guides() for more information.

n — the number of colors (≥ 1) to be in the palette.

rev — logical indicating whether the ordering of the colors should be reversed.

values — if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See rescale() for a convenience function to map an arbitrary range to between 0 and 1.

limits — One of:

- NULL to use the default scale range
- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see coord_cartesian()).

Details

On scale_*_cross_blended_tint_* palettes, the position of the gradients and the limits of the palette are redefined. Instead of treating the color palette as a continuous gradient, they are rescaled to act as a hypsometric tint. These tints roughly correspond to the following values:

- Blue colors: Negative values.
- Green colors: Values from 0 to 1,000.
- Browns: Values from 1,000 to 4,000.
- Whites: Values above 4,000.

These ranges vary depending on the palette definition (see cross_blended_hypsometric_tints_db for an example of how this can be achieved).

The palette setup may not always be suitable for your specific data. For example, a SpatRaster of small parts of the globe (and with a limited range of elevations) may not be well represented. As an example, a SpatRaster with values in the range [100, 200] appears almost as a uniform color. This can be adjusted using the limits/values arguments.

When passing the limits argument to scale_*_cross_blended_tint_*, the colors are restricted to those specified by this argument, keeping the distribution of the tint. You can combine this with oob, for example oob = scales::oob_squish, to avoid blank pixels in the plot.

cross_blended.colors2() provides a gradient color palette where the distance between colors differs depending on the type of color. In contrast, cross_blended.colors() provides a uniform gradient across colors. See **Examples**.

Value

A **ggplot2** scale for the `fill` or `colour` aesthetic. `cross_blended.colors()` and `cross_blended.colors2()` return a character vector of colors.

Source

- Patterson, T., & Jenny, B. (2011). The Development and Rationale of Cross-blended Hypsometric Tints. *Cartographic Perspectives,* (69), 31-46. doi:10.14714/CP69.20.
- Patterson, T. (2004). *Using Cross-blended Hypsometric Tints for Generalized Environmental Mapping.* Online, Accessed June 10, 2022.

See Also

cross_blended_hypsometric_tints_db, `terra::plot()`, `terra::minmax()`, `ggplot2::scale_fill_viridis_c()`.

See also **ggplot2** documentation on additional ... arguments.

Other color scales, palettes and hypsometric tints: `scale_coltab`, `scale_grass`, `scale_hypso`, `scale_princess`, `scale_terrain`, `scale_whitebox`, `scale_wiki`

Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = cross_blended.colors(100, palette = "arid"))

# Palette with uneven colors
plot(volcano2_rast, col = cross_blended.colors2(100, palette = "arid"))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_cross_blended_c(palette = "cold_humid")

# Full map with true tints

f_asia <- system.file("extdata/asia.tif", package = "tidyterra")
asia <- rast(f_asia)

ggplot() +
  geom_spatraster(data = asia) +
  scale_fill_cross_blended_tint_c(
    palette = "warm_humid",
    labels = scales::label_number(),
    breaks = c(-10000, 0, 5000, 8000),
    guide = guide_colorbar(reverse = TRUE)
  ) +
  labs(fill = "elevation (m)") +
```

```
  theme(
    legend.position = "bottom",
    legend.title.position = "top",
    legend.key.width = rel(3),
    legend.ticks = element_line(colour = "black", linewidth = 0.3),
    legend.direction = "horizontal"
  )

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_cross_blended_b(breaks = seq(70, 200, 25), palette = "arid")

# With breaks
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_cross_blended_b(
    breaks = seq(75, 200, 25),
    palette = "arid"
  )

# With discrete values
factor <- volcano2_rast |>
  mutate(cats = cut(elevation,
    breaks = c(100, 120, 130, 150, 170, 200),
    labels = c(
      "Very Low", "Low", "Average", "High",
      "Very High"
    )
  ))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_cross_blended_d(na.value = "gray10", palette = "cold_humid")

# Tint version
ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_cross_blended_tint_d(
    na.value = "gray10",
    palette = "cold_humid"
  )

# Display all the cross-blended palettes

pals <- unique(cross_blended_hypsometric_tints_db$pal)

# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
```

```
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = cross_blended.colors(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
# Display all version 2 cross-blended palettes.

pals <- unique(cross_blended_hypsometric_tints_db$pal)

# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = cross_blended.colors2(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
```

`scale_grass` *GRASS scales*

Description

Implementation of GRASS color tables. The following fill scales and palettes are provided:

- `scale_*_grass_d()`: For discrete values.
- `scale_*_grass_c()`: For continuous values.
- `scale_*_grass_b()`: For binning continuous values.
- `grass.colors()`: Gradient color palette. See also `grDevices::terrain.colors()` for details.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a subset of these additional arguments, so see the **ggplot2** functions listed above for the full range.

These palettes implement `terra::map.pal()`, the default color palettes used by `terra::plot()` in **terra** versions above 1.7.78.

Usage

```
scale_fill_grass_d(
  palette = "viridis",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_grass_d(
  palette = "viridis",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_grass_c(
  palette = "viridis",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  use_grass_range = TRUE,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_grass_c(
  palette = "viridis",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  use_grass_range = TRUE,
  na.value = "transparent",
  guide = "colourbar"
)
```

```
scale_fill_grass_b(
  palette = "viridis",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  use_grass_range = TRUE,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_grass_b(
  palette = "viridis",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  use_grass_range = TRUE,
  na.value = "transparent",
  guide = "coloursteps"
)

grass.colors(n, palette = "viridis", alpha = 1, rev = FALSE)
```

Arguments

palette	A valid palette name. The name is matched to the list of available palettes, ignoring upper vs. lower case. See grass_db for more information.
...	Arguments passed on to ggplot2::discrete_scale, ggplot2::continuous_scale, ggplot2::binned_scale

breaks One of:

- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

minor_breaks One of:

- NULL for no minor breaks
- waiver() for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

labels One of the options below. Please note that when labels is a vector, it is highly recommended to also set the breaks argument as a vector to protect against unintended mismatches.

- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

expand For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function expansion() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

n.breaks An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

alpha The alpha transparency, a number in [0,1], see argument alpha in hsv.

direction Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.

na.translate Logical. If TRUE, remove NA values from the legend. The default is FALSE.

drop Logical. If TRUE, omit unused factor levels from the scale. The default (TRUE) removes unused factors.

values if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the colours vector. See rescale() for a convenience function to map an arbitrary range to between 0 and 1.

limits One of:

- NULL to use the default scale range
- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see coord_cartesian()).

use_grass_range
: Logical. If TRUE, use the suggested range when plotting. See **Details**.

na.value
: Missing values will be replaced with this value. By default, **tidyterra** uses na.value = "transparent" so cells with NA are not filled. See also #120.

guide
: A function used to create a guide or its name. See guides() for more information.

n
: the number of colors (≥ 1) to be in the palette.

rev
: logical indicating whether the ordering of the colors should be reversed.

Details

Some palettes are mapped by default to a specific range of values (see grass_db). Set use_grass_range = FALSE to map the color scales to the range of values of the fill/colour aesthetics. See **Examples**.

When passing the limits argument, the colors are restricted to those specified by this argument, keeping the distribution of the palette. You can combine this with oob, for example oob = scales::oob_squish, to avoid blank pixels in the plot.

Value

A **ggplot2** scale for the fill or colour aesthetic. grass.colors() returns a character vector of colors.

terra equivalent

terra::map.pal().

Source

Derived from https://github.com/OSGeo/grass/tree/main/lib/gis/colors. See also r.color - GRASS GIS Manual.

References

GRASS Development Team (2024). *Geographic Resources Analysis Support System (GRASS) Software, Version 8.3.2*. Open Source Geospatial Foundation, USA. https://grass.osgeo.org.

See Also

grass_db, terra::plot(), terra::minmax(), ggplot2::scale_fill_viridis_c().

See also **ggplot2** documentation on additional ... arguments.

Other color scales, palettes and hypsometric tints: scale_coltab, scale_cross_blended, scale_hypso, scale_princess, scale_terrain, scale_whitebox, scale_wiki

Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = grass.colors(100, palette = "haxby"))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_grass_c(palette = "terrain")

# Use without default limits.
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_grass_c(palette = "terrain", use_grass_range = FALSE)

# Full map with true tints

f_asia <- system.file("extdata/asia.tif", package = "tidyterra")
asia <- rast(f_asia)

ggplot() +
  geom_spatraster(data = asia) +
  scale_fill_grass_c(
    palette = "srtm_plus",
    labels = scales::label_number(),
    breaks = c(-10000, 0, 5000, 8000),
    guide = guide_colorbar(reverse = FALSE)
  ) +
  labs(fill = "elevation (m)") +
  theme(
    legend.position = "bottom",
    legend.title.position = "top",
    legend.key.width = rel(3),
    legend.ticks = element_line(colour = "black", linewidth = 0.3),
    legend.direction = "horizontal"
  )

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_grass_b(breaks = seq(70, 200, 25), palette = "sepia")

# With discrete values
factor <- volcano2_rast |>
  mutate(cats = cut(elevation,
    breaks = c(100, 120, 130, 150, 170, 200),
    labels = c(
      "Very Low", "Low", "Average", "High",
```

```
      "Very High"
    )
  ))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_grass_d(palette = "soilmoisture")


# Display all the GRASS palettes
data("grass_db")

pals_all <- unique(grass_db$pal)

# In batches
pals <- pals_all[c(1:25)]
# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = grass.colors(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)

# Second batch
pals <- pals_all[-c(1:25)]

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = grass.colors(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
```

scale_hypso *Gradient scales for hypsometric and bathymetric tints*

Description

Implementation of a selection of gradient palettes available in cpt-city.

The following scales and palettes are provided:

- `scale_*_hypso_d()`: For discrete values.
- `scale_*_hypso_c()`: For continuous values.
- `scale_*_hypso_b()`: For binning continuous values.
- `hypso.colors()`: A gradient color palette. See also `grDevices::terrain.colors()` for details.

An additional set of scales is provided. These scales can act as hypsometric (or bathymetric) tints.

- `scale_*_hypso_tint_d()`: For discrete values.
- `scale_*_hypso_tint_c()`: For continuous values.
- `scale_*_hypso_tint_b()`: For binning continuous values.
- `hypso.colors2()`: A gradient color palette. See also `grDevices::terrain.colors()` for details.

See **Details**. Use `scale_fill_hypso_tint_c(palette = "gmt_globe")` for continuous hypsometric tints. Use `scale_fill_hypso_c()` for a continuous gradient.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a selection of these additional arguments, so check the **ggplot2** functions listed above to see the full range of arguments accepted by these scales.

Usage

```
scale_fill_hypso_d(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_hypso_d(
  palette = "etopo1_hypso",
  ...,
```

```
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_hypso_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_hypso_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_hypso_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_hypso_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

hypso.colors(n, palette = "etopo1_hypso", alpha = 1, rev = FALSE)

scale_fill_hypso_tint_d(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
```

```
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_hypso_tint_d(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_hypso_tint_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_hypso_tint_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_hypso_tint_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_hypso_tint_b(
```

```
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = "transparent",
  guide = "coloursteps"
)

hypso.colors2(n, palette = "etopo1_hypso", alpha = 1, rev = FALSE)
```

Arguments

palette	A valid palette name. The name is matched to the list of available palettes, ignoring upper vs. lower case. See hypsometric_tints_db for more information. The available values are listed below. "arctic", "arctic_bathy", "arctic_hypso", "c3t1", "colombia", "colombia_bathy", "colombia_hypso", "dem_poster", "dem_print", "dem_screen", "etopo1", "etopo1_bathy", "etopo1_hypso", "gmt_globe", "gmt_globe_bathy", "gmt_globe_hypso", "meyers", "meyers_bathy", "meyers_hypso", "moon", "moon_bathy", "moon_hypso", "nordisk-familjebok", "nordisk-familjebok_bathy", "nordisk-familjebok_hypso", "pakistan", "spain", "usgs-gswa2", "utah_1", "wiki-2.0", "wiki-2.0_bathy", "wiki-2.0_hypso", "wiki-schwarzwald-cont", "xkcd-painbow".
...	Arguments passed on to ggplot2::discrete_scale, ggplot2::continuous_scale, ggplot2::binned_scale breaks One of: • NULL for no breaks • waiver() for the default breaks (the scale limits) • A character vector of breaks • A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation. minor_breaks One of: • NULL for no minor breaks • waiver() for the default breaks (none for discrete, one minor break between each major break for continuous) • A numeric vector of positions • A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions. labels One of the options below. Please note that when labels is a vector, it is highly recommended to also set the breaks argument as a vector to protect against unintended mismatches. • NULL for no labels • waiver() for the default labels computed by the transformation object • A character vector giving labels (must be same length as breaks)

- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`n.breaks` An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation.

`nice.breaks` Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If `TRUE` (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

`alpha` The alpha transparency, a number in [0,1], see argument alpha in `hsv`.

`direction` Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.

`na.translate` Logical. If `TRUE`, remove `NA` values from the legend. The default is `FALSE`.

`drop` Logical. If `TRUE`, omit unused factor levels from the scale. The default (`TRUE`) removes unused factors.

`na.value` Missing values will be replaced with this value. By default, **tidyterra** uses `na.value = "transparent"` so cells with `NA` are not filled. See also #120.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`n` the number of colors (≥ 1) to be in the palette.

`rev` logical indicating whether the ordering of the colors should be reversed.

`values` if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See `rescale()` for a convenience function to map an arbitrary range to between 0 and 1.

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

Details

On `scale_*_hypso_tint_*` palettes, the position of the gradients and the limits of the palette are redefined. Instead of treating the color palette as a continuous gradient, they are rescaled to act as a hypsometric tint. These tints roughly correspond to the following values:

- Blue colors: Negative values.
- Green colors: Values from 0 to 1,000.
- Browns: Values from 1,000 to 4,000.
- Whites: Values above 4,000.

These ranges vary depending on the palette definition (see hypsometric_tints_db for an example of how this can be achieved).

The palette setup may not always be suitable for your specific data. For example, a `SpatRaster` of small parts of the globe (and with a limited range of elevations) may not be well represented. As an example, a `SpatRaster` with values in the range `[100, 200]` appears almost as a uniform color. This can be adjusted using the `limits`/`values` arguments.

When passing the `limits` argument to `scale_*_hypso_tint_*`, the colors are restricted to those specified by this argument, keeping the distribution of the tint. You can combine this with `oob`, for example `oob = scales::oob_squish`, to avoid blank pixels in the plot.

`hypso.colors2()` provides a gradient color palette where the distance between colors differs depending on the type of color. In contrast, `hypso.colors()` provides a uniform gradient across colors. See **Examples**.

Value

A **ggplot2** scale for the `fill` or `colour` aesthetic. `hypso.colors()` and `hypso.colors2()` return a character vector of colors.

Source

cpt-city: https://phillips.shef.ac.uk/pub/cpt-city/.

See Also

hypsometric_tints_db, `terra::plot()`, `terra::minmax()`, `ggplot2::scale_fill_viridis_c()`

See also **ggplot2** documentation on additional `...` arguments.

Other color scales, palettes and hypsometric tints: `scale_coltab`, `scale_cross_blended`, `scale_grass`, `scale_princess`, `scale_terrain`, `scale_whitebox`, `scale_wiki`

Examples

```
library(ggplot2)
library(tidyterra)

elevation <- terra::rast(nrows = 2, ncols = 2, vals = c(-100, 0, 500, 1000))
ggplot() +
  geom_spatraster(data = elevation) +
  scale_fill_hypso_tint_c(palette = "gmt_globe")
```

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = hypso.colors(100, palette = "wiki-2.0_hypso"))

# Palette with uneven colors
plot(volcano2_rast, col = hypso.colors2(100, palette = "wiki-2.0_hypso"))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_hypso_c(palette = "colombia_hypso")

# Full map with true tints

f_asia <- system.file("extdata/asia.tif", package = "tidyterra")
asia <- rast(f_asia)

ggplot() +
  geom_spatraster(data = asia) +
  scale_fill_hypso_tint_c(
    palette = "etopo1",
    labels = scales::label_number(),
    breaks = c(-10000, 0, 5000, 8000),
    guide = guide_colorbar(reverse = TRUE)
  ) +
  labs(fill = "elevation (m)") +
  theme(
    legend.position = "bottom",
    legend.title.position = "top",
    legend.key.width = rel(3),
    legend.ticks = element_line(colour = "black", linewidth = 0.3),
    legend.direction = "horizontal"
  )

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_hypso_b(breaks = seq(70, 200, 25), palette = "wiki-2.0_hypso")

# With breaks
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_hypso_b(
    breaks = seq(75, 200, 25),
    palette = "wiki-2.0_hypso"
  )
```

```
# With discrete values
factor <- volcano2_rast |> mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_hypso_d(na.value = "gray10", palette = "dem_poster")

# Tint version
ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_hypso_tint_d(na.value = "gray10", palette = "dem_poster")

# Display all the cpt-city palettes

pals <- unique(hypsometric_tints_db$pal)

# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = hypso.colors(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
# Display all version 2 cpt-city palettes.

pals <- unique(hypsometric_tints_db$pal)

# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
```

```
    col = hypso.colors2(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
```

`scale_princess` *Gradient scales from princess color schemes*

Description

Implementation of the gradient palettes presented in https://leahsmyth.github.io/Princess-Colour-Schemes/index.html. Three scales are provided:

- `scale_*_princess_d()`: For discrete values.
- `scale_*_princess_c()`: For continuous values.
- `scale_*_princess_b()`: For binning continuous values.

Additionally, a color palette `princess.colors()` is provided. See also `grDevices::terrain.colors()` for details.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a selection of these additional arguments, so check the **ggplot2** functions listed above to see the full range of arguments accepted by these scales.

Usage

```
scale_fill_princess_d(
  palette = "snow",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_princess_d(
  palette = "snow",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)
```

```
scale_fill_princess_c(
  palette = "snow",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_princess_c(
  palette = "snow",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_princess_b(
  palette = "snow",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_princess_b(
  palette = "snow",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

princess.colors(n, palette = "snow", alpha = 1, rev = FALSE)
```

Arguments

palette	A valid palette name. The name is matched to the list of available palettes, ignoring upper vs. lower case. The available values are listed below. "snow", "ella", "bell", "aura", "denmark", "france", "arabia", "america", "asia", "neworleans", "punz", "scotland", "cold", "norge", "maori".
...	Arguments passed on to ggplot2::discrete_scale, ggplot2::continuous_scale, ggplot2::binned_scale
	breaks One of:

- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

minor_breaks One of:

- NULL for no minor breaks
- waiver() for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

labels One of the options below. Please note that when labels is a vector, it is highly recommended to also set the breaks argument as a vector to protect against unintended mismatches.

- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

limits One of:

- NULL to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

expand For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function expansion() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

n.breaks An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

alpha The alpha transparency, a number in [0,1], see argument alpha in hsv.

direction	Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.
na.translate	Logical. If TRUE, remove NA values from the legend. The default is FALSE.
drop	Logical. If TRUE, omit unused factor levels from the scale. The default (TRUE) removes unused factors.
na.value	Missing values will be replaced with this value. By default, **tidyterra** uses na.value = "transparent" so cells with NA are not filled. See also #120.
guide	A function used to create a guide or its name. See guides() for more information.
n	the number of colors (≥ 1) to be in the palette.
rev	logical indicating whether the ordering of the colors should be reversed.

Value

A **ggplot2** scale for the fill or colour aesthetic. princess.colors() returns a character vector of colors.

Source

https://github.com/LeahSmyth/Princess-Colour-Schemes.

See Also

terra::plot(), ggplot2::scale_fill_viridis_c().

See also **ggplot2** documentation on additional ... arguments.

Other color scales, palettes and hypsometric tints: scale_coltab, scale_cross_blended, scale_grass, scale_hypso, scale_terrain, scale_whitebox, scale_wiki

Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = princess.colors(100))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_princess_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_princess_b(breaks = seq(70, 200, 10), palette = "denmark")
```

```
# With discrete values
factor <- volcano2_rast |> mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_princess_d(na.value = "gray10", palette = "maori")


# Display all the princess palettes

pals <- unique(princess_db$pal)

# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = princess.colors(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
```

`scale_terrain` *Terrain color scales from* **grDevices**

Description

Implementation of the classic color palette `terrain.colors()`:

- `scale_*_terrain_d()`: For discrete values.
- `scale_*_terrain_c()`: For continuous values.
- `scale_*_terrain_b()`: For binning continuous values.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.

- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a selection of these additional arguments, so check the **ggplot2** functions listed above to see the full range of arguments accepted by these scales.

Usage

```
scale_fill_terrain_d(
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_terrain_d(
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_terrain_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_terrain_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_terrain_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_terrain_b(
  ...,
```

```
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)
```

Arguments

`...` Arguments passed on to `ggplot2::discrete_scale`, `ggplot2::continuous_scale`, `ggplot2::binned_scale`

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

n.breaks An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

alpha	The alpha transparency, a number in [0,1], see argument alpha in hsv.
direction	Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.
na.translate	Logical. If TRUE, remove NA values from the legend. The default is FALSE.
drop	Logical. If TRUE, omit unused factor levels from the scale. The default (TRUE) removes unused factors.
na.value	Missing values will be replaced with this value. By default, **tidyterra** uses na.value = "transparent" so cells with NA are not filled. See also #120.
guide	A function used to create a guide or its name. See guides() for more information.

Value

A **ggplot2** scale for the fill or colour aesthetic.

See Also

terra::plot(), ggplot2::scale_fill_viridis_c() and **ggplot2** documentation on additional ... arguments.

Other color scales, palettes and hypsometric tints: scale_coltab, scale_cross_blended, scale_grass, scale_hypso, scale_princess, scale_whitebox, scale_wiki

Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_terrain_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_terrain_b(breaks = seq(70, 200, 10))
```

```
# With discrete values
factor <- volcano2_rast |> mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_terrain_d(na.value = "gray10")
```

`scale_whitebox` *Gradient scales from* **WhiteboxTools** *color schemes*

Description

Implementation of the gradient palettes provided by WhiteboxTools. Three scales are provided:

- `scale_*_whitebox_d()`: For discrete values.
- `scale_*_whitebox_c()`: For continuous values.
- `scale_*_whitebox_b()`: For binning continuous values.

Additionally, a color palette `whitebox.colors()` is provided. See also `grDevices::terrain.colors()` for details.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a selection of these additional arguments, so check the **ggplot2** functions listed above to see the full range of arguments accepted by these scales.

Usage

```
scale_fill_whitebox_d(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_whitebox_d(
```

```
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_whitebox_c(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_colour_whitebox_c(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_whitebox_b(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_whitebox_b(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

whitebox.colors(n, palette = "high_relief", alpha = 1, rev = FALSE)
```

Arguments

`palette` A valid palette name. The name is matched to the list of available palettes, ignoring upper vs. lower case. The available values are listed below. `"atlas"`, `"high_relief"`, `"arid"`, `"soft"`, `"muted"`, `"purple"`, `"viridi"`, `"gn_yl"`, `"pi_y_g"`, `"bl_yl_rd"`, `"deep"`.

`...` Arguments passed on to `ggplot2::discrete_scale`, `ggplot2::continuous_scale`, `ggplot2::binned_scale`

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

n.breaks An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

alpha	The alpha transparency, a number in [0,1], see argument alpha in hsv.
direction	Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.
na.translate	Logical. If TRUE, remove NA values from the legend. The default is FALSE.
drop	Logical. If TRUE, omit unused factor levels from the scale. The default (TRUE) removes unused factors.
na.value	Missing values will be replaced with this value. By default, **tidyterra** uses na.value = "transparent" so cells with NA are not filled. See also #120.
guide	A function used to create a guide or its name. See guides() for more information.
n	the number of colors (≥ 1) to be in the palette.
rev	logical indicating whether the ordering of the colors should be reversed.

Value

A **ggplot2** scale for the fill or colour aesthetic. whitebox.colors() returns a character vector of colors.

Source

https://github.com/jblindsay/whitebox-tools, under MIT License. Copyright (c) 2017-2021 John Lindsay.

See Also

terra::plot(), ggplot2::scale_fill_viridis_c().

See also **ggplot2** documentation on additional ... arguments.

Other color scales, palettes and hypsometric tints: scale_coltab, scale_cross_blended, scale_grass, scale_hypso, scale_princess, scale_terrain, scale_wiki

Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
```

```
plot(volcano2_rast, col = whitebox.colors(100))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_whitebox_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_whitebox_b(breaks = seq(70, 200, 10), palette = "atlas")

# With discrete values
factor <- volcano2_rast |> mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_whitebox_d(na.value = "gray10", palette = "soft")


# Display all the whitebox palettes

pals <- c(
  "atlas", "high_relief", "arid", "soft", "muted", "purple",
  "viridi", "gn_yl", "pi_y_g", "bl_yl_rd", "deep"
)

# Helper function for plotting

ncols <- 128
rowcol <- grDevices::n2mfrow(length(pals))

opar <- par(no.readonly = TRUE)
par(mfrow = rowcol, mar = rep(1, 4))

for (i in pals) {
  image(
    x = seq(1, ncols), y = 1, z = as.matrix(seq(1, ncols)),
    col = whitebox.colors(ncols, i), main = i,
    ylab = "", xaxt = "n", yaxt = "n", bty = "n"
  )
}
par(opar)
```

`scale_wiki` *Gradient scales from* **Wikipedia** *color schemes*

Description

Implementation based on the Wikipedia Colorimetric conventions for topographic maps.

Three scales are provided:

- `scale_*_wiki_d()`: For discrete values.
- `scale_*_wiki_c()`: For continuous values.
- `scale_*_wiki_b()`: For binning continuous values.

Additionally, a color palette `wiki.colors()` is provided. See also `grDevices::terrain.colors()` for details.

Additional arguments `...` are passed to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

tidyterra documents only a selection of these additional arguments, so check the **ggplot2** functions listed above to see the full range of arguments accepted by these scales.

Usage

```
scale_fill_wiki_d(
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_colour_wiki_d(
  ...,
  alpha = 1,
  direction = 1,
  na.translate = FALSE,
  drop = TRUE
)

scale_fill_wiki_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)
```

```
scale_colour_wiki_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "colourbar"
)

scale_fill_wiki_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

scale_colour_wiki_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = "transparent",
  guide = "coloursteps"
)

wiki.colors(n, alpha = 1, rev = FALSE)
```

Arguments

`...` Arguments passed on to `ggplot2::discrete_scale`, `ggplot2::continuous_scale`, `ggplot2::binned_scale`

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- NULL for no labels
- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

limits One of:

- NULL to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

expand For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function expansion() to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

n.breaks An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if breaks = waiver(). Use NULL to use the default number of breaks given by the transformation.

nice.breaks Logical. Should breaks be attempted placed at nice values instead of exactly evenly spaced between the limits. If TRUE (default) the scale will ask the transformation object to create breaks, and this may result in a different number of breaks than requested. Ignored if breaks are given explicitly.

alpha The alpha transparency, a number in [0,1], see argument alpha in hsv.

direction Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.

na.translate Logical. If TRUE, remove NA values from the legend. The default is FALSE.

drop Logical. If TRUE, omit unused factor levels from the scale. The default (TRUE) removes unused factors.

na.value Missing values will be replaced with this value. By default, **tidyterra** uses na.value = "transparent" so cells with NA are not filled. See also #120.

guide A function used to create a guide or its name. See guides() for more information.

n the number of colors (≥ 1) to be in the palette.

rev logical indicating whether the ordering of the colors should be reversed.

Value

A **ggplot2** scale for the fill or colour aesthetic. wiki.colors() returns a character vector of colors.

See Also

terra::plot(), ggplot2::scale_fill_viridis_c().

See also **ggplot2** documentation on additional ... arguments.

Other color scales, palettes and hypsometric tints: scale_coltab, scale_cross_blended, scale_grass, scale_hypso, scale_princess, scale_terrain, scale_whitebox

Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = wiki.colors(100))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_wiki_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_wiki_b(breaks = seq(70, 200, 10))

# With discrete values
factor <- volcano2_rast |> mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))

ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_wiki_d(na.value = "gray10")
```

---

select.Spat *Subset layers/attributes of* Spat* *objects*

Description

Select (and optionally rename) attributes/layers in Spat* objects, using a concise mini-language. See **Methods**. For a SpatRaster, select(r, rain = rainfall) keeps the rainfall layer and names it rain. To mask cells based on their values, use filter.Spat() instead.

Usage

```
## S3 method for class 'SpatRaster'
select(.data, ...)

## S3 method for class 'SpatVector'
select(.data, ...)
```

Arguments

`.data`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`...`	<`tidy-select`> One or more unquoted expressions separated by commas. Layer/attribute names can be used as if they were positions in the `Spat*` object, so expressions like `x:y` can be used to select a range of layers/attributes.

Value

A `Spat*` object of the same class as `.data`. See **Methods**.

terra equivalent

`terra::subset()`.

Methods

Implementation of the **generic** `dplyr::select()` methods for `Spat*` objects.

`SpatRaster`**:**

Selects and renames layers of a `SpatRaster`. The result is a `SpatRaster` with the same extent, resolution and CRS as `.data`. Only the number and possibly the names of layers are modified.

`SpatVector`**:**

The result is a `SpatVector` with the attributes selected and possibly renamed as specified in the function call.

See Also

`dplyr::select()`, `terra::subset()`.

Other **dplyr** verbs that operate on columns: `glimpse.Spat`, `mutate.Spat`, `pull.Spat`, `relocate.Spat`, `rename.Spat`

Examples

```
library(tidyterra)

weather <- terra::rast(
  nrows = 2, ncols = 2, nlyrs = 2,
  vals = c(20, 25, 30, 35, 10, 15, 20, 25)
)
names(weather) <- c("temperature", "rainfall")
```

```
rain <- select(weather, rain = rainfall)
terra::nlyr(rain)
names(rain)

library(terra)

# SpatRaster method

spatrast <- rast(
  crs = "EPSG:3857",
  nrows = 10,
  ncols = 10,
  extent = c(100, 200, 100, 200),
  nlyr = 6,
  vals = seq_len(10 * 10 * 6)
)

spatrast |> select(1)

# By name
spatrast |> select(lyr.1:lyr.4)

# Rename
spatrast |> select(a = lyr.1, c = lyr.6)

# SpatVector method

f <- system.file("extdata/cyl.gpkg", package = "tidyterra")

v <- vect(f)

v

v |> select(1, 3)

v |> select(iso2, name2 = cpro)
```

`slice.Spat` *Subset cells/rows/columns/geometries using their positions*

Description

`slice()` methods let you index cells/rows/columns/geometries by their (integer) locations. They allow you to select, remove or duplicate those dimensions of a `Spat*` object.

If you want to slice by geographic coordinates, use `filter.SpatRaster()`.

It includes helpers for common use cases:

- `slice_head()` and `slice_tail()` select the first or last cells/geometries.
- `slice_sample()` randomly selects cells/geometries.

- `slice_rows()` and `slice_cols()` subset entire rows or columns of a `SpatRaster`.
- `slice_colrows()` subsets regions of the `SpatRaster` by row and column position of a `SpatRaster`.

You can get a skeleton of your `SpatRaster` with the cell, column and row index with `as_coordinates()`.

See **Methods** for details.

Usage

```
## S3 method for class 'SpatRaster'
slice(.data, ..., .preserve = FALSE, .keep_extent = FALSE)

## S3 method for class 'SpatVector'
slice(.data, ..., .by = NULL, .preserve = FALSE)

## S3 method for class 'SpatRaster'
slice_head(.data, ..., n, prop, .keep_extent = FALSE)

## S3 method for class 'SpatVector'
slice_head(.data, ..., n, prop, by = NULL)

## S3 method for class 'SpatRaster'
slice_tail(.data, ..., n, prop, .keep_extent = FALSE)

## S3 method for class 'SpatVector'
slice_tail(.data, ..., n, prop, by = NULL)

## S3 method for class 'SpatRaster'
slice_min(
  .data,
  order_by,
  ...,
  n,
  prop,
  with_ties = TRUE,
  .keep_extent = FALSE,
  na.rm = TRUE
)

## S3 method for class 'SpatVector'
slice_min(
  .data,
  order_by,
  ...,
  n,
  prop,
  by = NULL,
  with_ties = TRUE,
  na_rm = FALSE
)
```

```
## S3 method for class 'SpatRaster'
slice_max(
  .data,
  order_by,
  ...,
  n,
  prop,
  with_ties = TRUE,
  .keep_extent = FALSE,
  na.rm = TRUE
)

## S3 method for class 'SpatVector'
slice_max(
  .data,
  order_by,
  ...,
  n,
  prop,
  by = NULL,
  with_ties = TRUE,
  na_rm = FALSE
)

## S3 method for class 'SpatRaster'
slice_sample(
  .data,
  ...,
  n,
  prop,
  weight_by = NULL,
  replace = FALSE,
  .keep_extent = FALSE
)

## S3 method for class 'SpatVector'
slice_sample(.data, ..., n, prop, by = NULL, weight_by = NULL, replace = FALSE)

slice_rows(.data, ...)

## S3 method for class 'SpatRaster'
slice_rows(.data, ..., .keep_extent = FALSE)

slice_cols(.data, ...)

## S3 method for class 'SpatRaster'
slice_cols(.data, ..., .keep_extent = FALSE)
```

```
slice_colrows(.data, ...)

## S3 method for class 'SpatRaster'
slice_colrows(.data, ..., cols, rows, .keep_extent = FALSE, inverse = FALSE)
```

Arguments

.data	A SpatRaster created with terra::rast() or a SpatVector created with terra::vect().
...	<data-masking> Integer row values. Provide either positive values to keep or negative values to drop. The values provided must be either all positive or all negative. Indices beyond the number of rows in the input are silently ignored. See **Methods**.
.preserve	Ignored for Spat* objects.
.keep_extent	Logical. If TRUE, keep the extent of the resulting SpatRaster. See also terra::trim(), terra::extend().
.by, by	<tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.
n, prop	Provide either n, the number of rows, or prop, the proportion of rows to select. If neither are supplied, n = 1 will be used. If n is greater than the number of rows in the group (or prop > 1), the result will be silently truncated to the group size. prop will be rounded towards zero to generate an integer number of rows. A negative value of n or prop will be subtracted from the group size. For example, n = -2 with a group of 5 rows will select 5 - 2 = 3 rows; prop = -0.25 with 8 rows will select 8 * (1 - 0.25) = 6 rows.
order_by	<data-masking> Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
with_ties	Should ties be kept together? The default, TRUE, may return more rows than you request. Use FALSE to ignore ties, and return the first n rows.
na.rm	Logical. If TRUE, remove cells with NA values when computing slice_min()/slice_max(). The default is TRUE.
na_rm	Should missing values in order_by be removed from the result? If FALSE, NA values are sorted to the end (like in arrange()), so they will only be included if there are insufficient non-missing values to reach n/prop.
weight_by	<data-masking> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the Details section for more technical details regarding these weights.
replace	Should sampling be performed with (TRUE) or without (FALSE, the default) replacement.
cols, rows	Integer column and row values of the SpatRaster.
inverse	If TRUE, .data is inverse-masked to the given selection. See terra::mask().

Value

A `Spat*` object of the same class as `.data`. See **Methods**.

terra equivalent

`terra::subset()` and `terra::spatSample()`.

Methods

Implementation of the **generic** `dplyr::slice()` methods for `Spat*` objects.

`SpatRaster`**:**

The result is a `SpatRaster` with the CRS and resolution of the input and the values of the selected cells, columns or rows.

Use `.keep_extent = TRUE` to preserve the extent of `.data` on the output. The non-selected cells have a value of `NA`.

`SpatVector`**:**

The result is a `SpatVector` where the attributes of the selected geometries are preserved. If `.data` is a grouped `SpatVector`, the operation will be performed on each group, so that (e.g.) `slice_head(df, n = 5)` will select the first five rows in each group.

See Also

`dplyr::slice()`, `terra::spatSample()`, `as_coordinates()`, `filter.SpatRaster()`.

Other **dplyr** verbs that operate on rows: `arrange.SpatVector()`, `distinct.SpatVector()`, `filter.Spat`, `rows.SpatVector`

Examples

```
library(terra)

f <- system.file("extdata/cyl_temp.tif", package = "tidyterra")
r <- rast(f)

# Slice first 100 cells
r |>
  slice(1:100) |>
  plot()

# Rows
r |>
  slice_rows(1:30) |>
  plot()

# Cols
r |>
  slice_cols(-(20:50)) |>
  plot()
```

```
# Spatial sample
r |>
  slice_sample(prop = 0.2) |>
  plot()

# Slice regions
r |>
  slice_colrows(
    cols = c(20:40, 60:80),
    rows = -c(1:20, 30:50)
  ) |>
  plot()

# Group-wise operation with SpatVector objects
v <- terra::vect(system.file("ex/lux.shp", package = "terra"))


glimpse(v) |> autoplot(aes(fill = NAME_1))

gv <- v |> group_by(NAME_1)
# All slice helpers operate per group, silently truncating to the group
# size.
gv |>
  slice_head(n = 1) |>
  glimpse() |>
  autoplot(aes(fill = NAME_1))
gv |>
  slice_tail(n = 1) |>
  glimpse() |>
  autoplot(aes(fill = NAME_1))
gv |>
  slice_min(AREA, n = 1) |>
  glimpse() |>
  autoplot(aes(fill = NAME_1))
gv |>
  slice_max(AREA, n = 1) |>
  glimpse() |>
  autoplot(aes(fill = NAME_1))
```

`summarise.SpatVector` *Summarize each group of a* `SpatVector` *down to one geometry*

Description

`summarise()` creates a new `SpatVector`. It returns one geometry for each combination of grouping variables. If there are no grouping variables, the output will have a single geometry summarizing all observations in the input and combining all the geometries of the `SpatVector`. It will contain one column for each grouping variable and one column for each of the summary statistics that you have specified.

`summarise.SpatVector()` and `summarize.SpatVector()` are synonyms.

Usage

```
## S3 method for class 'SpatVector'
summarise(.data, ..., .by = NULL, .groups = NULL, .dissolve = TRUE)

## S3 method for class 'SpatVector'
summarize(.data, ..., .by = NULL, .groups = NULL, .dissolve = TRUE)
```

Arguments

.data	A SpatVector created with terra::vect().
...	<data-masking> Name-value pairs of summary functions. The name will be the name of the variable in the result. The value can be: • A vector of length 1, e.g. min(x), n(), or sum(is.na(y)). • A data frame with 1 row, to add multiple columns from a single expression.
.by	<tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.
.groups	**[Experimental]** Grouping structure of the result. • "drop_last": drops the last level of grouping. This was the only supported option before version 1.0.0. • "drop": All levels of grouping are dropped. • "keep": Same grouping structure as .data. • "rowwise": Each row is its own group. When .groups is not specified, it is set to "drop_last" for a grouped data frame, and "keep" for a rowwise data frame. In addition, a message informs you of how the result will be grouped unless the result is ungrouped, the option "dplyr.summarise.inform" is set to FALSE, or when summarise() is called from a function in a package.
.dissolve	Logical. If TRUE, dissolve borders between aggregated geometries.

Value

A SpatVector object.

terra equivalent

terra::aggregate().

Methods

Implementation of the **generic** dplyr::summarise() method for SpatVector objects.

As in the **sf** implementation, this function can dissolve geometries with .dissolve = TRUE or create MULTI geometries with .dissolve = FALSE. See **Examples**. Summary expressions operate on the attributes of the input geometries. Changing .dissolve changes how their geometries are combined, while attribute summaries such as counts, means and sums use the same input rows.

See Also

dplyr::summarise(), terra::aggregate().

Other **dplyr** verbs that operate on groups of rows: count.SpatVector(), group_by.SpatVector(), reframe.SpatVector(), rowwise.SpatVector()

Examples

```
library(tidyterra)

provinces <- terra::vect(
  system.file("extdata/cyl.gpkg", package = "tidyterra")
)
provinces |>
  group_by(iso2) |>
  summarise(n = dplyr::n())

library(terra)
library(ggplot2)

v <- vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

# Grouped
gr_v <- v |>
  mutate(start_with_s = startsWith(name, "S")) |>
  group_by(start_with_s)

# Dissolve geometries.
diss <- gr_v |>
  summarise(n = dplyr::n(), mean = mean(as.double(cpro)))

diss

autoplot(diss, aes(fill = start_with_s)) +
  ggplot2::labs(title = "Dissolved")

# Keep geometries separate.
no_diss <- gr_v |>
  summarise(n = dplyr::n(), mean = mean(as.double(cpro)), .dissolve = FALSE)

# Same statistic.
no_diss

autoplot(no_diss, aes(fill = start_with_s)) +
  ggplot2::labs(title = "Not Dissolved")
```

---

tidy.Spat *Tidy* Spat* *objects for plotting*

Description

tidy() methods for SpatRaster, SpatVector, SpatGraticule and SpatExtent objects. These methods return a tibble for SpatRaster objects and sf objects for vector-based inputs. This interface is similar to fortify.Spat and is provided in case the ggplot2::fortify() method is deprecated in the future.

Usage

```
## S3 method for class 'SpatRaster'
tidy(
  x,
  ...,
  .name_repair = c("unique", "check_unique", "universal", "minimal", "unique_quiet",
    "universal_quiet"),
  maxcell = terra::ncell(x) * 1.1,
  pivot = FALSE
)

## S3 method for class 'SpatVector'
tidy(x, ...)

## S3 method for class 'SpatGraticule'
tidy(x, ...)

## S3 method for class 'SpatExtent'
tidy(x, ..., crs = "")
```

Arguments

x	A SpatRaster created with terra::rast(), a SpatVector created with terra::vect(), a SpatGraticule (see terra::graticule()) or a SpatExtent (see terra::ext()).
...	Ignored by these methods.
.name_repair	Treatment of problematic column names: • "minimal": No name repair or checks, beyond basic existence, • "unique": Make sure names are unique and not empty, • "check_unique": (default value), no name repair, but check they are unique, • "universal": Make the names unique and syntactic • "unique_quiet": Same as "unique", but "quiet" • "universal_quiet": Same as "universal", but "quiet" • a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R). • A purrr-style anonymous function, see rlang::as_function() This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.
maxcell	Positive integer or Inf. Maximum number of raster cells to use. Use Inf to remove this sampling limit. Reprojection for the plot can still change the grid, so Inf does not guarantee native plot resolution. The input raster is not modified.

pivot — Logical. When TRUE, a SpatRaster is returned in long format. When FALSE (the default), it is returned as a data frame with one column per layer. See **Details**.

crs — Input that includes or represents a CRS. It can be an sf or sfc object, a SpatRaster or SpatVector object, a crs object from sf::st_crs(), a character string (for example a PROJ string) or an integer representing an EPSG code.

Value

tidy.SpatVector(), tidy.SpatGraticule() and tidy.SpatExtent() return an sf object.

tidy.SpatRaster() returns a tibble. See **Methods**.

Methods

Implementation of the **generic** generics::tidy() methods for Spat* objects.

SpatRaster**:**

Returns a tibble that can be used with ggplot2::geom_*, such as ggplot2::geom_point() and ggplot2::geom_raster().

The resulting tibble includes coordinates in the x and y columns. The values of each layer are added as extra columns using the layer names from the SpatRaster.

The CRS of the SpatRaster can be retrieved with attr(tidySpatRaster, "crs").

You can convert the tidy object back to a SpatRaster with as_spatraster().

When pivot = TRUE, the SpatRaster is returned in long format (see tidyr::pivot_longer()). The tidy object has the following columns:

- x, y: Coordinates of the cell center in the corresponding CRS.
- lyr: Name of the SpatRaster layer associated with value.
- value: Cell value for the corresponding lyr.

This option can be useful when combining several geom_* layers or when faceting.

SpatVector**,** SpatGraticule **and** SpatExtent**:**

Returns an sf object that can be used with ggplot2::geom_sf().

See Also

sf::st_as_sf(), generics::tidy().

Other **generics** methods: glance.Spat, required_pkgs.Spat

Coercing objects: as_coordinates(), as_sf(), as_spatraster(), as_spatvector(), as_tibble.Spat, fortify.Spat

Examples

```
# Get a SpatRaster
r <- system.file("extdata/volcano2.tif", package = "tidyterra") |>
  terra::rast() |>
  terra::project("EPSG:4326")
```

```
r_tidy <- tidy(r)

r_tidy

# Convert back to a `SpatRaster`.
as_spatraster(r_tidy)

# SpatVector
cyl <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

cyl

tidy(cyl)

# SpatExtent
ex <- cyl |> terra::ext()

ex

tidy(ex)

# With a CRS
tidy(ex, crs = pull_crs(cyl))

# SpatGraticule
grat <- terra::graticule(60, 30, crs = "+proj=robin")

grat
tidy(grat)
```

`uncount.SpatVector` *Duplicate* `SpatVector` *rows*

Description

`uncount()` duplicates rows according to a weighting variable.

Usage

```
## S3 method for class 'SpatVector'
uncount(data, weights, ..., .remove = TRUE, .id = NULL)
```

Arguments

`data`	A `SpatVector`.
`weights`	A vector of weights. Evaluated in the context of `data`; supports quasiquotation.
`...`	Additional arguments passed on to methods.

`.remove`	If `TRUE`, and `weights` is the name of a column in `data`, then this column is removed.
`.id`	Supply a string to create a new variable which gives a unique identifier for each created row.

Value

A `SpatVector` object.

Methods

Implementation of the **generic** `tidyr::uncount()` method for `SpatVector` objects.

Each duplicated row keeps the input geometry.

See Also

`tidyr::uncount()`.

Examples

```
library(tidyr)

v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))
v$copies <- rep_len(1:2, nrow(v))

uncount(v, copies)
```

---

unite.Spat *Unite* `Spat*` *layers or attributes*

Description

`unite()` combines multiple layers or attributes by pasting their values together.

Usage

```
## S3 method for class 'SpatRaster'
unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)

## S3 method for class 'SpatVector'
unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)
```

Arguments

data	A SpatRaster or SpatVector.
col	The name of the new column, as a string or symbol. This argument is passed by expression and supports quasiquotation (you can unquote strings and symbols). The name is captured from the expression with rlang::ensym() (note that this kind of interface where symbols do not represent actual objects is now discouraged in the tidyverse; we support it here for backward compatibility).
...	<tidy-select> Columns to unite
sep	Separator to use between values.
remove	If TRUE, remove input columns from output data frame.
na.rm	If TRUE, missing values will be removed prior to uniting each value.

Value

A SpatRaster or SpatVector object.

Methods

Implementation of the **generic** tidyr::unite() methods for Spat* objects.

SpatRaster**:**

The selected layers are united cell by cell. The new layer is categorical because tidyr::unite() returns a character vector.

SpatVector**:**

The geometry column has sticky behavior and is never united with attributes.

See Also

tidyr::unite().

Examples

```
library(tidyr)

v <- terra::vect(system.file("extdata/cyl.gpkg", package = "tidyterra"))

unite(v, "label", iso2, cpro, sep = "-")

r <- terra::rast(system.file("extdata/cyl_temp.tif", package = "tidyterra"))

unite(r, "label", tavg_04, tavg_05, sep = "-", remove = FALSE)
```

volcano2 *Updated topographic information on Auckland's Maungawhau volcano*

Description

You may already know the volcano dataset. This dataset provides updated information for Maungawhau (Mt. Eden) from Toitu Te Whenua Land Information New Zealand, the government agency that provides free online access to New Zealand's most up-to-date land and seabed data.

Format

A matrix of 174 rows and 122 columns. Each value is the corresponding altitude in meters.

Note

Information needed for regenerating the original `SpatRaster` file:

- resolution: `c(5, 5)`
- extent: `1756969, 1757579, 5917003, 5917873` (xmin, xmax, ymin, ymax)
- Coordinate reference system: NZGD2000 / New Zealand Transverse Mercator 2000 (`EPSG:2193`)

Source

Auckland LiDAR 1m DEM (2013).

DEM for LiDAR data from the Auckland region captured in 2013. The original data has been downsampled to a resolution of 5 m due to disk space constraints.

Data license: CC BY 4.0.

See Also

volcano.

Other datasets: `cross_blended_hypsometric_tints_db`, `grass_db`, `hypsometric_tints_db`, `princess_db`

Examples

```
data("volcano2")
filled.contour(volcano2, color.palette = hypso.colors, asp = 1)
title(main = "volcano2 data: filled contour map")

# Geo-tag
# Empty raster

volcano2_raster <- terra::rast(volcano2)
terra::crs(volcano2_raster) <- pull_crs(2193)
terra::ext(volcano2_raster) <- c(1756968, 1757576, 5917000, 5917872)
```

```
names(volcano2_raster) <- "volcano2"

library(ggplot2)

ggplot() +
  geom_spatraster(data = volcano2_raster) +
  scale_fill_hypso_c() +
  labs(
    title = "volcano2 SpatRaster",
    subtitle = "Georeferenced",
    fill = "Elevation (m)"
  )
```

Index

∗ **coerce**
as_coordinates, 5
as_sf, 6
as_spatraster, 7
as_spatvector, 8
as_tibble.Spat, 10
fortify.Spat, 37
tidy.Spat, 148
∗ **datasets**
cross_blended_hypsometric_tints_db, 23
grass_db, 59
hypsometric_tints_db, 64
princess_db, 82
volcano2, 154
∗ **dplyr.cols**
glimpse.Spat, 58
mutate.Spat, 70
pull.Spat, 83
relocate.Spat, 88
rename.Spat, 89
select.Spat, 139
∗ **dplyr.group_functions**
group_by.SpatVector, 62
∗ **dplyr.groups**
count.SpatVector, 21
group_by.SpatVector, 62
reframe.SpatVector, 87
rowwise.SpatVector, 96
summarise.SpatVector, 146
∗ **dplyr.pairs**
bind_cols.SpatVector, 15
bind_rows.SpatVector, 16
cross_join.SpatVector, 24
filter-joins.SpatVector, 32
mutate-joins.SpatVector, 67
nest_join.SpatVector, 74
rows.SpatVector, 93
∗ **dplyr.rows**
arrange.SpatVector, 3
distinct.SpatVector, 26
filter.Spat, 35
rows.SpatVector, 93
slice.Spat, 141
∗ **generics.methods**
glance.Spat, 57
required_pkgs.Spat, 92
tidy.Spat, 148
∗ **ggplot2.methods**
autoplot.Spat, 13
fortify.Spat, 37
∗ **ggplot2.utils**
autoplot.Spat, 13
fortify.Spat, 37
geom_spat_contour, 48
geom_spatraster, 40
geom_spatraster_rgb, 45
ggspatvector, 54
∗ **gradients**
scale_coltab, 98
scale_cross_blended, 100
scale_grass, 108
scale_hypso, 115
scale_princess, 123
scale_terrain, 127
scale_whitebox, 131
scale_wiki, 136
∗ **helpers**
compare_spatrasters, 18
is_regular_grid, 66
pull_crs, 85
∗ **tibble.methods**
as_tibble.Spat, 10
∗ **tidyr.character**
unite.Spat, 152
∗ **tidyr.missing**
complete.SpatVector, 19
drop_na.Spat, 27

expand.SpatVector, 30
fill.SpatVector, 31
replace_na.Spat, 91
∗ **tidyr.nest**
nest.SpatVector, 72
∗ **tidyr.pivot**
pivot_longer.SpatVector, 76
pivot_wider.SpatVector, 79
∗ **tidyr.rows**
uncount.SpatVector, 151
?dplyr_by, *31*, *35*, *71*, *87*, *144*, *147*
?join_by, *33*, *68*, *75*

add_count.SpatVector *(*count.SpatVector*)*, 21
aes(), *55*
alpha, *42, 43*, *51*
annotation_borders(), *55*
anti_join(), *33*
anti_join.SpatVector *(*filter-joins.SpatVector*)*, 32
arrange(), *144*
arrange.SpatVector, 3
arrange.SpatVector(), *26*, *36*, *95*, *145*
as_coordinates, 5
as_coordinates(), *6*, *8*, *10*, *12*, *39*, *142*, *145*, *150*
as_sf, 6
as_sf(), *5, 6*, *8*, *10*, *12*, *39*, *150*
as_spatraster, 7
as_spatraster(), *5, 6*, *10*, *12*, *39*, *66*, *150*
as_spatvector, 8
as_spatvector(), *5, 6*, *8*, *12*, *39*, *150*
as_tibble(), *10*, *96*
as_tibble.Spat, *5, 6*, *8*, *10*, 10, *39*, *84*, *150*
as_tibble.Spat(), *35*, *58*, *83, 84*
as_tibble.SpatRaster *(*as_tibble.Spat*)*, 10
as_tibble.SpatRaster(), *8*, *58*, *84*
as_tibble.SpatVector *(*as_tibble.Spat*)*, 10
as_tibble.SpatVector(), *9*
autoplot.Spat, 13, *39*, *44*, *47*, *52*, *56*
autoplot.SpatExtent *(*autoplot.Spat*)*, 13
autoplot.SpatGraticule *(*autoplot.Spat*)*, 13
autoplot.SpatRaster *(*autoplot.Spat*)*, 13
autoplot.SpatRaster(), *13*
autoplot.SpatVector *(*autoplot.Spat*)*, 13

bind.Spat *(*bind_rows.SpatVector*)*, 16
bind_cols.SpatVector, 15, *17*, *25*, *34*, *69*, *75*, *95*
bind_rows.SpatVector, *15*, 16, *25*, *34*, *69*, *75*, *95*
bind_spat_cols *(*bind_cols.SpatVector*)*, 15
bind_spat_rows *(*bind_rows.SpatVector*)*, 16

colour, *51*
compare_spatrasters, 18
complete(), *20*, *30*
complete.SpatVector, 19
complete.SpatVector(), *28*, *30–32*, *91*
coord_cartesian(), *105*, *111*, *119*
count(), *22*
count.SpatVector, 21
count.SpatVector(), *63*, *87*, *97*, *148*
cross_blended.colors *(*scale_cross_blended*)*, 100
cross_blended.colors2 *(*scale_cross_blended*)*, 100
cross_blended_hypsometric_tints_db, 23, *61*, *65*, *82*, *103*, *105, 106*, *154*
cross_join(), *33*, *68*, *75*
cross_join.SpatVector, 24
cross_join.SpatVector(), *15*, *17*, *34*, *69*, *75*, *95*

desc(), *4*
distinct.SpatVector, 26
distinct.SpatVector(), *4*, *36*, *95*, *145*
dplyr-locale, *4*
dplyr::anti_join(), *34*
dplyr::arrange(), *4*
dplyr::bind_cols(), *15*
dplyr::bind_rows(), *17*
dplyr::count(), *21, 22*
dplyr::cross_join(), *24, 25*
dplyr::distinct(), *26*
dplyr::filter(), *35, 36*
dplyr::full_join(), *69*
dplyr::glimpse(), *59*
dplyr::group_by(), *9*, *63*
dplyr::inner_join(), *67*, *69*
dplyr::left_join(), *69*
dplyr::mutate(), *71, 72*
dplyr::nest_join(), *75*

dplyr::pull(), *84*
dplyr::reframe(), *87*
dplyr::relocate(), *89*
dplyr::rename(), *90*
dplyr::right_join(), *69*
dplyr::rows_insert(), *93*, *95*
dplyr::rowwise(), *96*, *97*
dplyr::select(), *140*
dplyr::semi_join(), *33*, *34*
dplyr::slice(), *145*
dplyr::summarise(), *147*, *148*
dplyr::tally(), *22*
dplyr::ungroup(), *63*
dplyr::when_any(), *35*
drop_na.Spat, *20*, 27, *31*, *32*, *91*
drop_na.SpatRaster *(*drop_na.Spat*)*, 27
drop_na.SpatRaster(), *28*, *35*
drop_na.SpatVector *(*drop_na.Spat*)*, 27

everything(), *96*
expand(), *20*, *30*, *80*
expand.SpatVector, 30
expand.SpatVector(), *20*, *28*, *32*, *91*
expansion(), *99*, *104*, *111*, *119*, *125*, *129*, *133*, *138*
extract(), *77*

fill, *42*, *43*, *51*
fill.SpatVector, 31
fill.SpatVector(), *20*, *28*, *31*, *91*
filter-joins.SpatVector, 32
filter.Spat, *4*, *26*, 35, *95*, *145*
filter.Spat(), *139*
filter.SpatRaster *(*filter.Spat*)*, 35
filter.SpatRaster(), *12*, *141*, *145*
filter.SpatVector *(*filter.Spat*)*, 35
filter_out.SpatVector *(*filter.Spat*)*, 35
fortify.Spat, *5*, *6*, *8*, *10*, *12*, *14*, 37, *44*, *47*, *52*, *56*, *149*, *150*
fortify.SpatExtent *(*fortify.Spat*)*, 37
fortify.SpatExtent(), *38*
fortify.SpatGraticule *(*fortify.Spat*)*, 37
fortify.SpatGraticule(), *38*
fortify.SpatRaster *(*fortify.Spat*)*, 37
fortify.SpatRaster(), *38*
fortify.SpatVector *(*fortify.Spat*)*, 37
fortify.SpatVector(), *38*, *56*
full_join(), *69*
full_join.SpatVector *(*mutate-joins.SpatVector*)*, 67

generics::glance(), *57*
generics::required_pkgs(), *92*
generics::tidy(), *150*
geom_spat_contour, *14*, *39*, *44*, *47*, 48, *56*
geom_spatraster, 40
geom_spatraster(), *13*, *14*, *39*, *40*, *45*, *47*, *52*, *56*
geom_spatraster_contour *(*geom_spat_contour*)*, 48
geom_spatraster_contour_filled *(*geom_spat_contour*)*, 48
geom_spatraster_contour_text *(*geom_spat_contour*)*, 48
geom_spatraster_rgb, 45
geom_spatraster_rgb(), *13*, *14*, *39*, *40*, *44*, *52*, *56*
geom_spatvector *(*ggspatvector*)*, 54
geom_spatvector(), *13*, *14*
geom_spatvector_label *(*ggspatvector*)*, 54
geom_spatvector_label(), *14*
geom_spatvector_text *(*ggspatvector*)*, 54
geom_spatvector_text(), *14*
get_coltab_pal *(*scale_coltab*)*, 98
get_coltab_pal(), *98*
ggplot2::aes(), *41*, *50*
ggplot2::after_stat(), *43*, *52*
ggplot2::autoplot(), *14*
ggplot2::binned_scale, *104*, *110*, *118*, *124*, *129*, *133*, *137*
ggplot2::binned_scale(), *101*, *108*, *115*, *123*, *128*, *131*, *136*
ggplot2::continuous_scale, *104*, *110*, *118*, *124*, *129*, *133*, *137*
ggplot2::continuous_scale(), *101*, *108*, *115*, *123*, *127*, *131*, *136*
ggplot2::coord_sf(), *42*, *43*, *47*, *52*
ggplot2::discrete_scale, *98*, *104*, *110*, *118*, *124*, *129*, *133*, *137*
ggplot2::discrete_scale(), *98*, *100*, *101*, *108*, *115*, *123*, *127*, *131*, *136*
ggplot2::facet_wrap(), *43*, *52*
ggplot2::fortify(), *38*, *149*
ggplot2::geom_contour(), *48*, *52*
ggplot2::geom_label(), *43*
ggplot2::geom_point(), *38*, *43*, *150*

ggplot2::geom_raster(), *38*, *40*, *43*, *45*, *47*, *150*
ggplot2::geom_sf(), *39*, *54–56*, *150*
ggplot2::geom_text(), *44*
ggplot2::ggplot(), *37*
ggplot2::scale_fill_gradientn(), *23*, *59*, *65*
ggplot2::scale_fill_manual(), *100*
ggplot2::scale_fill_viridis_c(), *106*, *112*, *120*, *126*, *130*, *134*, *139*
ggspatvector, *14*, *39*, *44*, *47*, *52*, 54
glance.Spat, 57, *92*, *150*
glance.SpatRaster *(*glance.Spat*)*, 57
glance.SpatVector *(*glance.Spat*)*, 57
glimpse.Spat, *57*, 58, *72*, *84*, *89*, *90*, *140*
glimpse.SpatRaster *(*glimpse.Spat*)*, 58
glimpse.SpatVector *(*glimpse.Spat*)*, 58
grass.colors *(*scale_grass*)*, 108
grass_db, *23*, 59, *65*, *82*, *110*, *112*, *154*
grDevices::rgb(), *46*, *47*
grDevices::terrain.colors(), *100*, *101*, *108*, *115*, *123*, *131*, *136*
group, *51*
group_by(), *31*, *35*, *71*, *87*, *144*, *147*
group_by.SpatVector, 62
group_by.SpatVector(), *6*, *9*, *22*, *32*, *62*, *63*, *87*, *96*, *97*, *148*
group_by_drop_default(), *63*
group_data.SpatVector(), *63*
group_map.SpatVector(), *63*
group_nest.SpatVector(), *63*
group_split.SpatVector(), *63*
group_trim.SpatVector(), *63*
grouped, *145*
guides(), *105*, *112*, *119*, *126*, *130*, *134*, *138*

hsv, *46*, *104*, *111*, *119*, *125*, *130*, *134*, *138*
hypso.colors *(*scale_hypso*)*, 115
hypso.colors2 *(*scale_hypso*)*, 115
hypsometric_tints_db, *23*, *61*, 64, *82*, *118*, *120*, *154*

inner_join.SpatVector *(*mutate-joins.SpatVector*)*, 67
is_regular_grid, 66
is_regular_grid(), *8*
isoband::isolines_grob(), *48*

join, *15*
join_by(), *33*, *68*, *74*, *75*

key glyphs, *42*, *46*, *50*

label_placer_minmax(), *51*
lambda, *98*, *99*, *104*, *105*, *110*, *111*, *118*, *119*, *125*, *129*, *133*, *137*, *138*
layer geom, *55*
layer position, *55*
layer(), *41*, *42*, *46*, *50*
left_join.SpatVector *(*mutate-joins.SpatVector*)*, 67
linetype, *51*
linewidth, *51*
locale, *4*
long format, *38*, *150*

maptiles::get_tiles(), *47*
match(), *75*
merge(), *75*
mutate(), *21*
mutate-joins.SpatVector, 67
mutate.Spat, *59*, 70, *84*, *89*, *90*, *140*
mutate.SpatRaster *(*mutate.Spat*)*, 70
mutate.SpatVector *(*mutate.Spat*)*, 70

nest.SpatVector, 72
nest_join.SpatVector, 74
nest_join.SpatVector(), *15*, *17*, *25*, *34*, *69*, *95*

option, *58*

pivot_longer.SpatVector, 76
pivot_longer.SpatVector(), *79*, *81*
pivot_wider(), *79*
pivot_wider.SpatVector, 79
pivot_wider.SpatVector(), *76*, *78*
pretty(), *50*
princess.colors *(*scale_princess*)*, 123
princess_db, *23*, *61*, *65*, 82, *154*
print(), *58*
pull.Spat, *59*, *72*, 83, *89*, *90*, *140*
pull.SpatRaster *(*pull.Spat*)*, 83
pull.SpatVector *(*pull.Spat*)*, 83
pull_crs, 85
pull_crs(), *7–11*

quasiquotation, *84*, *153*

recycled, *15*
reframe.SpatVector, 87
reframe.SpatVector(), *22*, *63*, *97*, *148*
relocate(), *71*
relocate.Spat, *59*, *72*, *84*, 88, *90*, *140*
relocate.SpatRaster *(*relocate.Spat*)*, 88
relocate.SpatVector *(*relocate.Spat*)*, 88
rename.Spat, *59*, *72*, *84*, *89*, 89, *140*
rename.SpatRaster *(*rename.Spat*)*, 89
rename.SpatVector *(*rename.Spat*)*, 89
rename_with.SpatRaster *(*rename.Spat*)*, 89
rename_with.SpatVector *(*rename.Spat*)*, 89
replace_na.Spat, *20*, *28*, *31*, *32*, 91
replace_na.SpatRaster *(*replace_na.Spat*)*, 91
replace_na.SpatVector *(*replace_na.Spat*)*, 91
required_pkgs.Spat, *57*, 92, *150*
required_pkgs.SpatExtent *(*required_pkgs.Spat*)*, 92
required_pkgs.SpatGraticule *(*required_pkgs.Spat*)*, 92
required_pkgs.SpatRaster *(*required_pkgs.Spat*)*, 92
required_pkgs.SpatVector *(*required_pkgs.Spat*)*, 92
rescale(), *105*, *111*, *119*
right_join(), *69*
right_join.SpatVector *(*mutate-joins.SpatVector*)*, 67
rlang::as_function(), *11*, *38*, *149*
rlang::ensym(), *153*
rows.SpatVector, *4*, *15*, *17*, *25*, *26*, *34*, *36*, *69*, *75*, 93, *145*
rows_append.SpatVector *(*rows.SpatVector*)*, 93
rows_delete.SpatVector *(*rows.SpatVector*)*, 93
rows_insert.SpatVector *(*rows.SpatVector*)*, 93
rows_patch.SpatVector *(*rows.SpatVector*)*, 93
rows_update.SpatVector *(*rows.SpatVector*)*, 93
rows_upsert.SpatVector *(*rows.SpatVector*)*, 93
rowwise.SpatVector, 96
rowwise.SpatVector(), *22*, *63*, *87*, *96*, *148*
scale_color_coltab *(*scale_coltab*)*, 98
scale_color_cross_blended_b *(*scale_cross_blended*)*, 100
scale_color_cross_blended_c *(*scale_cross_blended*)*, 100
scale_color_cross_blended_d *(*scale_cross_blended*)*, 100
scale_color_cross_blended_tint_b *(*scale_cross_blended*)*, 100
scale_color_cross_blended_tint_c *(*scale_cross_blended*)*, 100
scale_color_cross_blended_tint_d *(*scale_cross_blended*)*, 100
scale_color_grass_b *(*scale_grass*)*, 108
scale_color_grass_c *(*scale_grass*)*, 108
scale_color_grass_d *(*scale_grass*)*, 108
scale_color_hypso_b *(*scale_hypso*)*, 115
scale_color_hypso_c *(*scale_hypso*)*, 115
scale_color_hypso_d *(*scale_hypso*)*, 115
scale_color_hypso_tint_b *(*scale_hypso*)*, 115
scale_color_hypso_tint_c *(*scale_hypso*)*, 115
scale_color_hypso_tint_d *(*scale_hypso*)*, 115
scale_color_princess_b *(*scale_princess*)*, 123
scale_color_princess_c *(*scale_princess*)*, 123
scale_color_princess_d *(*scale_princess*)*, 123
scale_color_terrain_b *(*scale_terrain*)*, 127
scale_color_terrain_c *(*scale_terrain*)*, 127
scale_color_terrain_d *(*scale_terrain*)*, 127
scale_color_whitebox_b *(*scale_whitebox*)*, 131
scale_color_whitebox_c *(*scale_whitebox*)*, 131
scale_color_whitebox_d *(*scale_whitebox*)*, 131
scale_color_wiki_b *(*scale_wiki*)*, 136
scale_color_wiki_c *(*scale_wiki*)*, 136
scale_color_wiki_d *(*scale_wiki*)*, 136
scale_colour_coltab *(*scale_coltab*)*, 98
scale_colour_cross_blended_b

(scale_cross_blended), 100
scale_colour_cross_blended_c (scale_cross_blended), 100
scale_colour_cross_blended_d (scale_cross_blended), 100
scale_colour_cross_blended_tint_b (scale_cross_blended), 100
scale_colour_cross_blended_tint_c (scale_cross_blended), 100
scale_colour_cross_blended_tint_d (scale_cross_blended), 100
scale_colour_grass_b (scale_grass), 108
scale_colour_grass_c (scale_grass), 108
scale_colour_grass_d (scale_grass), 108
scale_colour_hypso_b (scale_hypso), 115
scale_colour_hypso_c (scale_hypso), 115
scale_colour_hypso_d (scale_hypso), 115
scale_colour_hypso_tint_b (scale_hypso), 115
scale_colour_hypso_tint_c (scale_hypso), 115
scale_colour_hypso_tint_d (scale_hypso), 115
scale_colour_princess_b (scale_princess), 123
scale_colour_princess_c (scale_princess), 123
scale_colour_princess_d (scale_princess), 123
scale_colour_terrain_b (scale_terrain), 127
scale_colour_terrain_c (scale_terrain), 127
scale_colour_terrain_d (scale_terrain), 127
scale_colour_whitebox_b (scale_whitebox), 131
scale_colour_whitebox_c (scale_whitebox), 131
scale_colour_whitebox_d (scale_whitebox), 131
scale_colour_wiki_b (scale_wiki), 136
scale_colour_wiki_c (scale_wiki), 136
scale_colour_wiki_d (scale_wiki), 136
scale_coltab, 98, *106*, *112*, *120*, *126*, *130*, *134*, *139*
scale_cross_blended, *100*, 100, *112*, *120*, *126*, *130*, *134*, *139*
scale_fill_coltab (scale_coltab), 98
scale_fill_coltab(), *13*, *41*
scale_fill_cross_blended_b (scale_cross_blended), 100
scale_fill_cross_blended_c (scale_cross_blended), 100
scale_fill_cross_blended_c(), *23*
scale_fill_cross_blended_d (scale_cross_blended), 100
scale_fill_cross_blended_tint_b (scale_cross_blended), 100
scale_fill_cross_blended_tint_c (scale_cross_blended), 100
scale_fill_cross_blended_tint_d (scale_cross_blended), 100
scale_fill_grass_b (scale_grass), 108
scale_fill_grass_c (scale_grass), 108
scale_fill_grass_c(), *61*
scale_fill_grass_d (scale_grass), 108
scale_fill_hypso_b (scale_hypso), 115
scale_fill_hypso_c (scale_hypso), 115
scale_fill_hypso_c(), *65*
scale_fill_hypso_d (scale_hypso), 115
scale_fill_hypso_tint_b (scale_hypso), 115
scale_fill_hypso_tint_c (scale_hypso), 115
scale_fill_hypso_tint_d (scale_hypso), 115
scale_fill_princess_b (scale_princess), 123
scale_fill_princess_c (scale_princess), 123
scale_fill_princess_c(), *82*
scale_fill_princess_d (scale_princess), 123
scale_fill_terrain_b (scale_terrain), 127
scale_fill_terrain_c (scale_terrain), 127
scale_fill_terrain_d (scale_terrain), 127
scale_fill_terrain_d(), *99*
scale_fill_whitebox_b (scale_whitebox), 131
scale_fill_whitebox_c (scale_whitebox), 131
scale_fill_whitebox_d (scale_whitebox),

131
scale_fill_wiki_b *(scale_wiki)*, 136
scale_fill_wiki_c *(scale_wiki)*, 136
scale_fill_wiki_d *(scale_wiki)*, 136
scale_grass, *100*, *106*, 108, *120*, *126*, *130*, *134*, *139*
scale_hypso, *100*, *106*, *112*, 115, *126*, *130*, *134*, *139*
scale_princess, *100*, *106*, *112*, *120*, 123, *130*, *134*, *139*
scale_terrain, *100*, *106*, *112*, *120*, *126*, 127, *134*, *139*
scale_whitebox, *100*, *106*, *112*, *120*, *126*, *130*, 131, *139*
scale_wiki, *100*, *106*, *112*, *120*, *126*, *130*, *134*, 135
scales::label_number(), *51*
select.Spat, *59*, *72*, *84*, *88–90*, 139
select.SpatRaster *(select.Spat)*, 139
select.SpatVector *(select.Spat)*, 139
semi_join(), *33*
semi_join.SpatVector *(filter-joins.SpatVector)*, 32
separate(), *77*
sf, *6*, *15–17*, *25*, *33*, *38*, *39*, *56*, *68*, *85*, *149*, *150*
sf::st_as_sf(), *6*, *25*, *39*, *150*
sf::st_crs(), *8*, *10*, *38*, *85*, *86*, *150*
sf::st_transform(), *86*
sfc, *9*
size, *51*
slice.Spat, *4*, *26*, *36*, *95*, 141
slice.SpatRaster *(slice.Spat)*, 141
slice.SpatRaster(), *5*
slice.SpatVector *(slice.Spat)*, 141
slice_colrows *(slice.Spat)*, 141
slice_cols *(slice.Spat)*, 141
slice_head.SpatRaster *(slice.Spat)*, 141
slice_head.SpatVector *(slice.Spat)*, 141
slice_max.SpatRaster *(slice.Spat)*, 141
slice_max.SpatVector *(slice.Spat)*, 141
slice_min.SpatRaster *(slice.Spat)*, 141
slice_min.SpatVector *(slice.Spat)*, 141
slice_rows *(slice.Spat)*, 141
slice_sample.SpatRaster *(slice.Spat)*, 141
slice_sample.SpatVector *(slice.Spat)*, 141
slice_tail.SpatRaster *(slice.Spat)*, 141
slice_tail.SpatVector *(slice.Spat)*, 141
SpatRaster, *5*, *7*, *8*, *10–12*, *14*, *18*, *27*, *28*, *35–43*, *45*, *47*, *48*, *51*, *52*, *71*, *83–85*, *89*, *90*, *98*, *99*, *105*, *120*, *139*, *140*, *142*, *144*, *145*, *149*, *150*, *153*, *154*
SpatVector, *3*, *4*, *6*, *8–12*, *14–17*, *19–22*, *24–28*, *30–34*, *36–39*, *54*, *56*, *62*, *63*, *67–69*, *72–76*, *78*, *79*, *81*, *84*, *85*, *87*, *89*, *90*, *93*, *95*, *96*, *140*, *145–147*, *149–153*
stat_spat_coordinates(), *14*, *39*, *44*, *47*, *52*, *56*
stat_spatraster *(geom_spatraster)*, 40
stat_spatvector *(ggspatvector)*, 54
stretch, *47*
stringi::stri_locale_list(), *4*
summarise(), *21*
summarise.SpatVector, 146
summarise.SpatVector(), *22*, *63*, *87*, *96*, *97*
summarize.SpatVector *(summarise.SpatVector)*, 146

tally(), *22*
tally.SpatVector *(count.SpatVector)*, 21
terra::aggregate(), *19*, *22*, *147*, *148*
terra::app(), *71*, *72*
terra::as.data.frame(), *11*, *12*, *84*
terra::clamp(), *71*, *72*
terra::classify(), *71*, *72*
terra::coltab(), *13*, *41*, *98*, *100*
terra::compareGeom(), *18*
terra::contour(), *51*
terra::crs(), *8*, *10*, *86*
terra::disagg(), *19*
terra::ext(), *13*, *38*, *92*, *149*
terra::extend(), *144*
terra::graticule(), *13*, *38*, *92*, *149*
terra::has.colors(), *98*
terra::identical(), *19*
terra::ifel(), *71*, *72*
terra::intersect(), *33*, *68*
terra::lapp(), *71*, *72*
terra::map.pal(), *61*, *109*, *112*
terra::mask(), *28*, *144*
terra::merge(), *34*, *69*
terra::minmax(), *106*, *112*, *120*
terra::plot(), *13*, *40*, *42*, *56*, *106*, *109*, *112*, *120*, *126*, *130*, *134*, *139*

terra::plotRGB(), *13*, *47*
terra::project(), *17*, *19*, *41*, *47*, *51*, *86*
terra::rast(), *7, 8*, *11*, *13*, *28*, *35*, *38*, *40*, *45*, *57, 58*, *71*, *83*, *88*, *90–92*, *140*, *144*, *149*
terra::resample(), *19*, *28*
terra::sort(), *4*
terra::spatSample(), *63*, *96*, *145*
terra::subset(), *140*, *145*
terra::svc(), *73*
terra::tapp(), *71, 72*
terra::trim(), *28*, *35, 36*, *144*
terra::unique(), *26*
terra::values(), *84*
terra::vect(), *4*, *6*, *8, 9*, *11*, *13*, *21*, *25, 26*, *28*, *33*, *35*, *38*, *54, 55*, *57, 58*, *68*, *71*, *83*, *87, 88*, *90–92*, *140*, *144*, *147*, *149*
terrain.colors(), *127*
tibble, *5*, *7–12*, *17*, *23*, *30, 31*, *38, 39*, *57*, *59, 60*, *65*, *73–75*, *82*, *84*, *149, 150*
tibble::as_tibble(), *11, 12*
tibble::print.tbl_df(), *59*
tidy.Spat, *5, 6*, *8*, *10*, *12*, *37*, *39*, *57*, *92*, 148
tidy.SpatExtent *(*tidy.Spat*)*, 148
tidy.SpatExtent(), *150*
tidy.SpatGraticule *(*tidy.Spat*)*, 148
tidy.SpatGraticule(), *150*
tidy.SpatRaster *(*tidy.Spat*)*, 148
tidy.SpatRaster(), *150*
tidy.SpatVector *(*tidy.Spat*)*, 148
tidy.SpatVector(), *150*
tidyr::complete(), *20*
tidyr::drop_na(), *28*
tidyr::expand(), *31*
tidyr::fill(), *32*
tidyr::nest(), *73*
tidyr::pivot_longer(), *39*, *76*, *78*, *150*
tidyr::pivot_wider(), *81*
tidyr::replace_na(), *91*
tidyr::uncount(), *152*
tidyr::unite(), *153*
tidyr::unnest(), *73*
tidyselect, *80*

uncount.SpatVector, 151
ungroup.SpatVector *(*group_by.SpatVector*)*, 62
ungroup.SpatVector(), *96*
unite.Spat, 152
unite.SpatRaster *(*unite.Spat*)*, 152
unite.SpatVector *(*unite.Spat*)*, 152

vctrs::vec_as_names(), *11*, *15*, *38*, *78*, *81*, *149*
vec_as_names(), *30*
volcano, *154*
volcano2, *23*, *61*, *65*, *82*, 154

whitebox.colors *(*scale_whitebox*)*, 131
wiki.colors *(*scale_wiki*)*, 136