Package 'tidyterra'

January 8, 2025

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

Version 0.6.2

Description Extension of the 'tidyverse' for 'SpatRaster' and 'SpatVector' objects of the 'terra' package. It includes also new 'geom_' functions that provide a convenient way of visualizing 'terra' objects 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 (>= 3.6.0)

Imports cli (>= 3.0.0), data.table, dplyr (>= 1.0.0), ggplot2 (>= 3.5.0), magrittr, rlang, scales, sf (>= 1.0.0), terra (>= 1.5-12), tibble (>= 3.0.0), tidyr (>= 1.0.0)

Suggests hexbin, isoband, knitr, lifecycle, maptiles, metR, rmarkdown, s2, testthat (>= 3.0.0), vctrs, vdiffr

VignetteBuilder knitr

Config/Needs/coverage covr

Config/Needs/website geodata, dieghernan/gitdevr, ragg, styler, ggspatial, cpp11, remotes, gganimate, gifski

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

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]
(<https://orcid.org/0000-0001-8457-4658>),
Dewey Dunnington [ctb] (<https://orcid.org/0000-0002-9415-4582>, for
ggspatial code),
ggplot2 authors [cph] (for contour code)

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

Repository CRAN

Date/Publication 2025-01-08 09:20:02 UTC

Contents

`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)
```

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`, will sort first by grouping variable. Applies to grouped `SpatVector` only (see `group_by.SpatVector()`).

Value

A `SpatVector` object.

terra equivalent

`terra::sort()`

Methods

Implementation of the **generic** `dplyr::arrange()` function for `SpatVector` class.

See Also

dplyr::arrange()

Other single table verbs: filter.Spat, mutate.Spat, rename.Spat, select.Spat, slice.Spat, summarise.SpatVector()

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

Other **dplyr** methods: bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

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() can be used to obtain the position of each cell on 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 the same number of rows (or cells) than the number of cells in x.

When as.raster = TRUE the resulting SpatRaster has the same crs, extension and resolution than x

See Also

slice.SpatRaster()

Coercing objects: as_sf(), as_spatraster(), as_spatvector(), as_tibble.Spat, fortify.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 a* sf *object*

Description

as_sf() turns a SpatVector to sf object. This is a wrapper of sf::st_as_sf() with the particularity that the groups created with group_by.SpatVector() are preserved.

Usage

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

Arguments

x	A SpatVector.
...	additional arguments passed on to sf::st_as_sf().

Value

A sf object object with an additional tbl_df class, for pretty printing method.

See Also

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

Examples

```
library(terra)

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

# This is ungrouped
v
is_grouped_spatvector(v)

# Get an 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)

# A sf

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()` turns an existing data frame or `tibble` into a `SpatRaster`. This is a wrapper of `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 on 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 on several formats (PROJ.4, WKT, EPSG code, ..) or and 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 has been created with the method `as_tibble.SpatRaster()`, the `crs` is inferred from `attr(x, "crs")`.

Value

A `SpatRaster`.

terra equivalent

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

See Also

`pull_crs()` for retrieving crs, and the corresponding utils `sf::st_crs()` and `terra::crs()`.

Coercing objects: `as_coordinates()`, `as_sf()`, `as_spatvector()`, `as_tibble.Spat`, `fortify.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` *Method for coercing objects to* `SpatVector`

Description

`as_spatvector()` turns an existing object into a `SpatVector`. This is a wrapper of `terra::vect()` S4 method for signature `data.frame`.

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 on to `terra::vect()`.
`geom`	character. The field name(s) with the geometry data. Either two names for x and y coordinates of points, or a single name for a single column with WKT geometries.
`crs`	A crs on several formats (PROJ.4, WKT, EPSG code, ..) or and 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()` on the following:

- geometries with `NA` or `""` values are removed prior to conversion

- If x is a grouped data frame (see dplyr::group_by()) the grouping vars are transferred and a "grouped" SpatVector is created (see group_by.SpatVector()).
- If no crs is provided and the tibble has been created with the method as_tibble.SpatVector(), the crs is inferred from attr(x, "crs").
- Handles correctly the conversion of EMPTY geometries between **sf** and **terra**.

Value

A SpatVector.

terra equivalent

terra::vect()

See Also

pull_crs() for retrieving crs, and the corresponding utils sf::st_crs() and terra::crs().

Coercing objects: as_coordinates(), as_sf(), as_spatraster(), as_tibble.Spat, fortify.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 a* SpatVector *or* SpatRaster *object to data frames*

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 = "unique")

## S3 method for class 'SpatVector'
as_tibble(x, ..., geom = NULL, .name_repair = "unique")
```

Arguments

x	A SpatRaster created with terra::rast() or a SpatVector created with terra::vect().
...	Arguments passed on 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. • 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().
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() function.

SpatRaster **and** SpatVector**:**

The tibble is returned with an attribute including the crs of the initial object in WKT format (see pull_crs()).

About layer/column names

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

In the process of coercing a SpatRaster to a tibble, **tidyterra** may rename the layers of your SpatRaster for overcoming this issue. Specifically, layers may be renamed on the following cases:

- Layers with duplicated names.
- When coercing to a tibble, if xy = TRUE, layers named x or y would be renamed.
- When working with tidyverse methods (i.e. filter.SpatRaster()), the latter would happen as well.

tidyterra would display a message informing of the changes on the names of the layer.

The same issue happens for SpatVector with names geometry (when geom = c("WKT", "HEX")) and x, y (when geom = "XY"). These are reserved names representing the geometry of the SpatVector (see terra::as.data.frame()). If geom is not NULL then the logic described for SpatRaster would apply as well for the columns of the SpatVector.

See Also

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

Coercing objects: as_coordinates(), as_sf(), as_spatraster(), as_spatvector(), fortify.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 as the ones 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, ...)
```

Arguments

`object`	A `SpatRaster` created with `terra::rast()` or a `SpatVector` created with `terra::vect()`.
`...`	other arguments passed to `geom_spatraster()`, `geom_spatraster_rgb()` or `geom_spatvector()`.
`rgb`	Logical. Should be plotted as a RGB image? If `NULL` (the default) `autoplot.SpatRaster()` would try to guess.
`use_coltab`	Logical. Should be plotted with the corresponding `terra::coltab()`? If `NULL` (the default) `autoplot.SpatRaster()` would try to guess. See also `scale_fill_coltab()`.
`facets`	Logical. Should facets be displayed? If `NULL` (the default) `autoplot.SpatRaster()` would try to guess.
`nrow`, `ncol`	Number of rows and columns on the facet.

Details

Implementation of `ggplot2::autoplot()` method.

Value

A **ggplot2** layer

Methods

Implementation of the **generic** `ggplot2::autoplot()` function.

`SpatRaster`**:**
Uses `geom_spatraster()` or `geom_spatraster_rgb()`.

`SpatVector`**:**
Uses `geom_spatvector()`. Labels can be placed with `geom_spatvector_text()` or `geom_spatvector_label()`.

See Also

`ggplot2::autoplot()`

Other **ggplot2** utils: `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 coltabs

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 frames objects by column*

Description

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

Where possible prefer using a join to combine `SpatVector` and data frames objects. `bind_spat_cols()` binds the rows in 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 should be a `SpatVector` and each of the subsequent arguments can either be a `SpatVector`, a `sf` object or a data frame. Inputs are recycled to the same length, then matched by position.
`.name_repair`	One of `"unique"`, `"universal"`, or `"check_unique"`. See `dplyr::bind_cols()` for **Details**.

Value

A `SpatVector` with the corresponding columns. The geometry and CRS would correspond to the the first `SpatVector` of `...`.

terra equivalent

`cbind()` method

Methods

Implementation of the `dplyr::bind_rows()` function for `SpatVector` objects. Note that for the second and subsequent arguments on `...` the geometry would not be cbinded, and only the data frame (-ish) columns would be kept.

See Also

`dplyr::bind_cols()`

Other **dplyr** verbs that operate on pairs `Spat*`/data.frame: `bind_rows.SpatVector`, `filter-joins.SpatVector`, `mutate-joins.SpatVector`

Other **dplyr** methods: arrange.SpatVector(), bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.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 frames objects by row*

Description

Bind any number of SpatVector, data frames and sf/sfc objects by row, making a longer result. This is similar to do.call(rbind, dfs), 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 should be a SpatVector and each of the subsequent arguments can either be a SpatVector, a sf/sfc object or a data frame. Columns are matched by name, and any missing columns will be 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 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 element of ... should be a SpatVector. Subsequent elements may be SpatVector, sf/sfc objects or data frames:

- If subsequent SpatVector/sf/sfc objects present a different CRS than the first element, those elements would be reprojected to the CRS of the first element with a message.
- If any element of ... is a tibble/data frame the rows would be cbinded with empty geometries with a message.

See Also

dplyr::bind_rows()

Other **dplyr** verbs that operate on pairs Spat*/data.frame: bind_cols.SpatVector, filter-joins.SpatVector, mutate-joins.SpatVector

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.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 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
sfobj <- sf::st_as_sf(v2[1, ])

sfobj

bind_spat_rows(v1, sfobj)

# Would reproject with a message on different CRS
sfobj_3857 <- as_spatvector(sfobj) %>% project("EPSG:3857")

bind_spat_rows(v1, sfobj_3857)

# And with 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, ]))

# Or named list combined 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 informing of the results. See **Solving issues** section 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

A invisible logical TRUE/FALSE indicating if the SpatRaster objects are compatible, plus an informative message flagging the issues found (if any).

terra equivalent

terra::identical()

Solving issues

- On **non-equal crs**, try terra::project().
- On **non-equal extent** try terra::resample().
- On **non-equal resolution** you can try terra::resample(), terra::aggregate() or terra::disagg().

See Also

terra::identical()

Other helpers: is_grouped_spatvector(), is_regular_grid(), pull_crs()

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"))
```

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()).

Usage

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

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

Arguments

x	A SpatVector.
...	<data-masking> Variables to group by.
wt	Not implemented on this method
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	Handling of factor levels that don't appear in the data, passed on to group_by(). For count(): if FALSE will include counts for empty groups (i.e. for levels of factors that don't exist in the data). **[Deprecated]** For add_count(): deprecated since it can't actually affect the output.
.dissolve	logical. Should borders between aggregated geometries be dissolved?

Value

A SpatVector object with an additional attribute.

terra equivalent

terra::aggregate()

Methods

Implementation of the **generic** dplyr::count() family functions 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 group of rows: group-by.SpatVector, rowwise.SpatVector(), summarise.SpatVector()

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

Examples

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


p %>% count(NAME_1, sort = TRUE)

p %>% count(NAME_1, sort = TRUE)

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

# tally() is a lower-level function that assumes you've done the grouping
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 the palettes includes also a definition of colors 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"
  )
```

---

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, will use all variables in the data frame. There is a reserved variable name, geometry, that would remove 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() function.

SpatVector**:**

It is possible to remove duplicate geometries including the geometry variable explicitly in the ... call. See **Examples**.

See Also

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

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

Examples

```
library(terra)

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

# Create a vector with dups
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 NAME_1
ex2 <- distinct(v, gr)
ex2
nrow(ex2)

# Same but keeping all cols
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)

# Unique keeping info
distinct(v, geometry, .keep_all = TRUE)
```

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

Description

- `SpatVector`: `drop_na()` method drops geometries where any attribute specified by `...` contains a missing value.
- `SpatRaster`: `drop_na()` method drops cells where 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` `terra::rast()`.
`...`	<tidy-select> Attributes to inspect for missing values. If empty, all attributes are used.

Value

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

terra equivalent

`terra::trim()`

Methods

Implementation of the **generic** `tidyr::drop_na()` function.

`SpatVector`**:**

The implementation of this method is performed on a `by-attribute` basis, meaning that `NAs` are assessed on the attributes (columns) of each vector (rows). The result is a `SpatVector` with potentially less geometries than the input.

`SpatRaster`**:**

[Questioning]

Actual 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 considered as columns and `SpatRaster` cells as rows, so rows (cells) with any `NA` value on any layer would get a `NA` value. It is possible also to mask the cells (rows) based on the values of specific layers (columns).

`drop_na()` would effectively remove outer cells that are `NA` (see `terra::trim()`), so the extent of the resulting object may differ of the extent of the input (see `terra::resample()` for more info).

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: `fill.SpatVector()`, `replace_na.Spat`

Other **tidyr** methods: `fill.SpatVector()`, `pivot_longer.SpatVector()`, `pivot_wider.SpatVector()`, `replace_na.Spat`

Examples

```
library(terra)

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

v <- terra::vect(f)

# Add NAs
v <- v %>% mutate(iso2 = ifelse(cpro <= "09", NA, cpro))
```

```
# Init
plot(v, col = "red")

# Mask with lyr.1
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 NAs
r[r > 13 & r < 22 | r > 31 & r < 45] <- NA

# Init
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)

# Auto-mask all layers
r %>%
  drop_na() %>%
  plot(nc = 3)
```

`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, ..., .direction = c("down", "up", "downup", "updown"))
```

Arguments

data	A SpatVector.
...	<tidy-select> Columns to fill.
.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() function for SpatVector.

Grouped SpatVector

With grouped SpatVector created by group_by.SpatVector(), fill() will be applied *within* each group, meaning that it won't fill across group boundaries.

See Also

tidyr::fill()

Other **tidyr** verbs for handling missing values: drop_na.Spat, replace_na.Spat

Other **tidyr** methods: drop_na.Spat, pivot_longer.SpatVector(), pivot_wider.SpatVector(), replace_na.Spat

Examples

```
library(dplyr)

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

# Leave some blanks for demo purporses

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()` return all rows from `x` with a match in `y`.
- `anti_join()` return 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` **of** `sf` **object** is provided it would return an error (see `terra::intersect()` for performing 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() family

SpatVector**:**

The geometry column has a sticky behavior. This means that the result would have always the geometry of x for the records that matches the join conditions.

See Also

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

Other **dplyr** verbs that operate on pairs Spat*/data.frame: bind_cols.SpatVector, bind_rows.SpatVector, mutate-joins.SpatVector

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.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)) + ggtitle("Semi Join")


# Anti join

anti <- v %>% anti_join(df)

anti

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

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

Description

The `filter()` function is used to subset `Spat*` objects, retaining all cells/geometries that satisfy your conditions. To be retained, the cell/geometry must produce a value of `TRUE` for all conditions.

It is possible to filter a `SpatRaster` **by its geographic coordinates**. You need to use `filter(.data, x > 42)`. Note that `x` and `y` are reserved names on **terra**, since they refer to the geographic coordinates of the layer.

See **Examples** and section **About layer 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, ..., .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	Ignored for Spat* objects.
.keep_extent	Should the extent of the resulting SpatRaster be kept? On FALSE, terra::trim() is called so the extent of the result may be different of the extent of the output. See also drop_na.SpatRaster().

Value

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

Methods

Implementation of the **generic** dplyr::filter() function.

SpatRaster**:**

Cells that do not fulfill the conditions on ... are returned with value NA. On a multi-layer SpatRaster the NA is propagated across all the layers.

If .keep_extent = TRUE the returning SpatRaster has the same crs, extent, resolution and hence the same number of cells than .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 would present in any case the same crs and resolution than .data.

x and y variables (i.e. the longitude and latitude of the SpatRaster) are also available internally for filtering. See **Examples**.

SpatVector**:**

The result is a SpatVector with all the geometries that produce a value of TRUE for all conditions.

See Also

dplyr::filter()

Other single table verbs: arrange.SpatVector(), mutate.Spat, rename.Spat, select.Spat, slice.Spat, summarise.SpatVector()

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

Examples

```
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()
```

---

fortify.Spat *Fortify* Spat* *Objects*

Description

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

Usage

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

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

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

Arguments

model	A SpatRaster created with terra::rast() or a SpatVector created with terra::vect().
data	Not used by this method.
...	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.
.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.
- 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().

maxcell — positive integer. Maximum number of cells to use for the plot.

pivot — Logical. When TRUE the SpatRaster would be fortified on long format. When FALSE (the default) it would be fortified as a data frame with a column for each layer. See **Details**.

Value

fortify.SpatVector() returns a sf object and fortify.SpatRaster() returns a tibble. See **Methods**.

Methods

Implementation of the **generic** ggplot2::fortify() method.

SpatRaster**:**

Return a tibble than can be used with ggplot2::geom_* like ggplot2::geom_point(), ggplot2::geom_raster(), etc.

The resulting tibble includes the coordinates on the columns x, y. The values of each layer are included as additional columns named as per the name of the layer on the SpatRaster.

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

It is possible to convert the fortified object onto a SpatRaster again with as_spatraster().

When pivot = TRUE the SpatRaster is fortified in a "long" format (see tidyr::pivot_longer()). The fortified object would have the following columns:

- x,y: Coordinates (center) of the cell on the corresponding CRS.
- lyr: Indicating the name of the SpatRaster layer of value.
- value: The value of the SpatRaster in the corresponding lyr.

This option may be useful when using several geom_* and for faceting, see **Examples**.

SpatVector **and** SpatGraticule**:**

Return a sf object than can be used with ggplot2::geom_sf().

See Also

sf::st_as_sf(), as_tibble.Spat, as_spatraster(), ggplot2::fortify().

Other **ggplot2** utils: 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

Examples

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

fortified <- ggplot2::fortify(r)

fortified

# The crs is an attribute of the fortified SpatRaster

attr(fortified, "crs")

# Back to a SpatRaster with
as_spatraster(fortified)

# You can now use a SpatRaster with any geom
library(ggplot2)

ggplot(r) +
  geom_histogram(aes(x = elevation),
    bins = 20, fill = "lightblue",
    color = "black"
  )

# ... and other packages
# Use metR with facets
library(metR)
temp <- terra::rast(system.file("extdata/cyl_temp.tif",
  package = "tidyterra"
))
brks <- seq(0, 21, 3) # Fix breaks!

# Pivot option for faceting
ggplot(temp, aes(x, y), pivot = TRUE) +
  # tidyterra, don't inherit aes
  geom_spatraster_contour_filled(
    data = temp, inherit.aes = FALSE,
    breaks = brks
  ) +
  # metR
  geom_contour_tanaka(aes(z = value), breaks = brks) +
  facet_wrap(~lyr, nrow = 1) +
  scale_fill_whitebox_d(palette = "muted") +
  theme_minimal() +
  labs(
    title = "tidyterra + metR", subtitle = "Facets",
    fill = "temp (°C)", x = "", y = ""
  )
```

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

cyl

# To sf
ggplot2::fortify(cyl)

# Now you can use geom_sf() straight away thanks to fortify::SpatVector()

library(ggplot2)

ggplot(cyl) +
  geom_sf()
```

`geom_spatraster` *Visualise* `SpatRaster` *objects*

Description

This geom is used to visualise `SpatRaster` objects (see `terra::rast()`). The geom is designed for visualise the object by layers, as `terra::plot()` does.

For plotting `SpatRaster` objects as map tiles (i.e. RGB `SpatRaster`), use `geom_spatraster_rgb()`.

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

`stat_spatraster()` is provided as a complementary function, so the `geom` can be modified.

Usage

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

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** specially in the use of fill aesthetic.
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.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them.
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
maxcell	positive integer. Maximum number of cells to use for the plot.
use_coltab	Logical. Only applicable to SpatRaster objects that have an associated coltab. Should the coltab be used on the plot? See also scale_fill_coltab().
...	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	The geometric object to use display the data. Recommended geom for SpatRaster are "raster" (the default), "point","text" and "label".

Value

A **ggplot2** layer

terra equivalent

`terra::plot()`

Coords

When the `SpatRaster` does not present a crs (i.e., `terra::crs(rast) == ""`) the geom does not make any assumption on the scales.

On `SpatRaster` that have a crs, the geom uses `ggplot2::coord_sf()` to adjust the scales. That means that also 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 properly the `SpatRaster` layers.

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

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

`fill` can use computed variables.

For `alpha` use computed variable. See section **Computed variables**.

`stat_spatraster()`**:**

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

- `fill`
- `alpha`

When `geom = "raster"` the `fill` parameter would behave as in `geom_spatraster()`. If another `geom` is used `stat_spatraster()` would understand the aesthetics of the required `geom` and `aes(fill = <name_of_lyr>)` would not be applicable.

Note also that mapping of aesthetics `x` and `y` is provided by default, so the user does not need to add those aesthetics on `aes()`. In all the cases the aesthetics should be mapped by using computed variables. See section **Computed variables** and **Examples**.

Facets

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

Computed variables

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

- `after_stat(value)`: Values of the `SpatRaster`.
- `after_stat(lyr)`: Name of the layer.

Source

Based on the `layer_spatial()` implementation on **ggspatial** package. 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** utils: `autoplot.Spat`, `fortify.Spat`, `geom_spat_contour`, `geom_spatraster_rgb()`, `ggspatvector`, `stat_spat_coordinates()`

Examples

```
# Avg temperature on spring in Castille 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 *Visualise* SpatRaster *objects as images*

Description

This geom is used to visualise SpatRaster objects (see terra::rast()) as RGB images. The layers are combined such that they represent the red, green and blue channel.

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
)
```

Arguments

mapping	Ignored.
data	A SpatRaster object.
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
r, g, b	Integer representing the number of layer of data to be considered as 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. Maximum number of cells to use for the plot.
max_col_value	Number giving the maximum of the color values 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

Value

A **ggplot2** layer

terra equivalent

`terra::plotRGB()`

Aesthetics

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

Coords

When the `SpatRaster` does not present a crs (i.e., `terra::crs(rast) == ""`) the geom does not make any assumption on the scales.

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

Source

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

See Also

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

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

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

Examples

```
# Tile of Castille 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 of SpatRaster objects. To specify a valid surface, you should specify the layer on aes(z = layer_name), otherwise all the layers would be consider for creating contours. See also **Facets** section.

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

[Experimental] 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
)

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
)

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
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `ggplot2::aes()`. See **Aesthetics** specially in the use of `fill` aesthetic.
`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. Maximum number of cells to use for the plot.

`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.

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

`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 produced the same result than `geom_spatraster_contour()`.
- A character vector giving labels (must be 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 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() / geom_spatraster_contour_text() understands the following aesthetics:

- alpha
- colour
- group
- linetype
- linewidth geom_spatraster_contour_text() understands also:
- size
- label
- family
- fontface

Additionally, geom_spatraster_contour_filled() understands also the following aesthetics, as well as the ones listed above:

- fill
- subgroup

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

Computed variables

These geom computes internally some variables 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 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), after_stat(level_mid): (contour bands only) Lower and upper bin boundaries for each band, as well the mid point between the boundaries.

Dropped variables

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

Coords

When the SpatRaster does not present a crs (i.e., terra::crs(rast) == "") the geom does not make any assumption on the scales.

On SpatRaster that have a crs, the geom uses ggplot2::coord_sf() to adjust the scales. That means that also the SpatRaster **may be reprojected**.

Facets

You can use facet_wrap(~lyr) for creating a faceted plot by 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** utils: 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)


# Labelled
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 = .7
  ) +
  geom_spatraster_contour(
    data = r, breaks = seq(80, 200, 2.5),
    color = "grey30",
    linewidth = 0.1
  ) +
  scale_fill_hypso_d()
```

---

ggspatvector *Visualise* SpatVector *objects*

Description

Wrappers of ggplot2::geom_sf() family used to visualise 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,
  ...,
  nudge_x = 0,
  nudge_y = 0,
  label.size = 0.25,
  inherit.aes = TRUE
)

geom_spatvector_text(
  mapping = aes(),
  data = NULL,
  na.rm = FALSE,
  show.legend = NA,
  ...,
  nudge_x = 0,
  nudge_y = 0,
  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 on to `ggplot2::geom_sf()` functions. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `linewidth = 3`.

`nudge_x, nudge_y` Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with `position`.

`label.size` Size of label border, in mm.

`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. `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** treat a `SpatVector` in the same way that a `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 utils: 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 use now geom_sf with SpatVectors!



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

`glimpse.Spat` *Get a nice glimpse of your* `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. See `dplyr::glimpse()`.
`...`	Arguments passed on to `as_tibble()` methods for `SpatRaster` and `SpatVector`. See `as_tibble.SpatRaster()`.
`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

original `x` is (invisibly) returned, allowing `glimpse()` to be used within a data pipeline.

terra equivalent

`print()`

Methods

Implementation of the **generic** `dplyr::glimpse()` function 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`

Other **dplyr** methods: `arrange.SpatVector()`, `bind_cols.SpatVector`, `bind_rows.SpatVector`, `count.SpatVector()`, `distinct.SpatVector()`, `filter-joins.SpatVector`, `filter.Spat`, `group-by.SpatVector`, `mutate-joins.SpatVector`, `mutate.Spat`, `pull.Spat`, `relocate.Spat`, `rename.Spat`, `rowwise.SpatVector()`, `select.Spat`, `slice.Spat`, `summarise.SpatVector()`

Examples

```
library(terra)

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

v %>% glimpse(n = 2)

# Use on 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 on 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 includes also a definition of colors 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
aspect	General	aspect oriented grey colors
aspectcolr	General	aspect oriented rainbow colors
bcyr	General	blue through cyan through yellow to red
bgyr	General	blue through green through 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 degree Celsius temperature
corine	Land Cover	EU Corine land cover colors
curvature	General	for terrain curvatures
differences	General	differences oriented colors
elevation	Topography	maps relative ranges of raster values to elevation color ramp
etopo2	Topography	colors for ETOPO2 worldwide bathymetry/topography
evi	Natural	enhanced vegetative index colors
fahrenheit	Temperature	blue to red for Fahrenheit temperature
forest_cover	Natural	percentage of forest cover
gdd	Natural	accumulated growing degree days
grass	General	GRASS GIS green (perceptually uniform)
greens	General	white to green
grey	General	grey scale
gyr	General	green through yellow to red
haxby	Topography	relative colors for bathymetry or topography
inferno	General	perceptually uniform sequential color table inferno
kelvin	Temperature	blue to red for temperature in Kelvin scale
magma	General	perceptually uniform sequential color table magma
ndvi	Natural	Normalized Difference Vegetation Index colors
ndwi	Natural	Normalized Difference Water Index colors
nlcd	Land Cover	US National Land Cover Dataset colors
oranges	General	white to orange
plasma	General	perceptually uniform sequential color table plasma
population	Human	color table covering human population classification breaks
population_dens	Human	color table covering human population density classification breaks
precipitation	Climate	precipitation color table (0..2000mm)
precipitation_daily	Climate	precipitation color table (0..1000mm)
precipitation_monthly	Climate	precipitation color table (0..1000mm)
rainbow	General	rainbow color table
ramp	General	color ramp
reds	General	white to red
roygbiv	General	
rstcurv	General	terrain curvature (from r.resamp.rst)
ryb	General	red through yellow to blue
ryg	General	red through yellow to green
sepia	General	yellowish-brown through to white
slope	General	r.slope.aspect-type slope colors for raster values 0-90

soilmoisture	Natural	soil moisture color table (0.0-1.0)
srtm	Topography	color palette for Shuttle Radar Topography Mission elevation
srtm_plus	Topography	color palette for Shuttle Radar Topography Mission elevation (with seafloor color
terrain	Topography	global elevation color table covering -11000 to +8850m
viridis	General	perceptually uniform sequential color table viridis
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**.
...	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. This argument was previously called add, but that prevented creating a new grouping variable called add, and conflicts with our naming conventions.
.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 an additional attribute.

Methods

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

When mixing terra and dplyr syntax on a grouped SpatVector (i.e, subsetting a SpatVector like v[1:3,1:2]) the groups attribute can be corrupted. **tidyterra** would try to re-group the SpatVector. This would be triggered the next time you use a **dplyr** verb on your SpatVector.

Note also that some operations (as terra::spatSample()) would create a new SpatVector. In these cases, the result won't preserve the groups attribute. Use group_by() to re-group.

See Also

dplyr::group_by(), dplyr::ungroup()

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

Examples

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


by_name1 <- p %>% group_by(NAME_1)

# grouping doesn't change how the SpatVector looks
by_name1

# But add 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 removing 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 instead append
by_name2_name1 %>%
  group_by(ID_1, ID_2, .add = TRUE) %>%
  group_vars()

# You can group by expressions: this is a short-hand
# 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 33 gradient palettes. All the palettes includes also a definition of colors limits in terms of elevation (meters), that can be used with `ggplot2::scale_fill_gradientn()`.

Format

A `tibble` of 1102 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: http://seaviewsensing.com/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 if x and y positions conforms a regular grid*

Description

Assess if the coordinates x,y of an object conforms a regular grid. This function is called by its side effects.

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

Usage

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

Arguments

`xy`	A matrix, data frame or tibble of 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

`invisible()` if is regular or an error message otherwise

See Also

`as_spatraster()`

Other helpers: `compare_spatrasters()`, `is_grouped_spatvector()`, `pull_crs()`

Examples

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

is_regular_grid(p)


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

# Need to 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. There are four mutating joins: the inner join, and the three outer joins.

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 of sf object** is provided it would return an error (see terra::intersect() for performing 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() family

SpatVector**:**

The geometry column has a sticky behavior. This means that the result would have always the geometry of x for the records that matches the join conditions.

Note that for right_join() and full_join() it is possible to return empty geometries (since y is expected to be a data frame with no geometries). Although this kind of joining operations may not be common on spatial manipulation, it is possible that the function crashes, since handling of EMPTY geometries differs on **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 Spat*/data.frame: bind_cols.SpatVector, bind_rows.SpatVector, filter-joins.SpatVector

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.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)) + ggtitle("Inner Join")
```

```
# Left join

left <- v %>% left_join(df)
nrow(left)

autoplot(left, aes(fill = letter)) + ggtitle("Left Join")


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

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

# There are empty geometries, check with data from df
ggplot(right, aes(x, y)) +
  geom_point(aes(color = letter))


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

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

# Check with data from df
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. `transmute()` adds new layers/attributes and drops existing ones. New variables overwrite existing variables of the same name. Variables can be removed by setting their value to `NULL`.

Usage

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

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

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

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

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 layer/attribute in the output.

Value

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

terra equivalent

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

Methods

Implementation of the **generic** `dplyr::mutate()`, `dplyr::transmute()` functions.

`SpatRaster`**:**

Add new layers and preserves existing ones. The result is a `SpatRaster` with the same extent, resolution and crs than `.data`. Only the values (and possibly the number) of layers is modified.

`transmute()` would keep only the layers created with `....`

`SpatVector`**:**

The result is a `SpatVector` with the modified (and possibly renamed) attributes on the function call.

`transmute()` would keep only the attributes created with `....`

See Also

`dplyr::mutate()`, `dplyr::transmute()` methods.

terra provides several ways to modify `Spat*` objects:

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

Other single table verbs: `arrange.SpatVector()`, `filter.Spat`, `rename.Spat`, `select.Spat`, `slice.Spat`, `summarise.SpatVector()`

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

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)
```

---

pivot_longer.SpatVector

Pivot SpatVector *from wide to long*

Description

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

Learn more in tidyr::pivot_wider().

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 NAs in the value_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() function.

SpatVector**:**

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

See Also

tidyr::pivot_longer()

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

Other **tidyr** methods: drop_na.Spat, fill.SpatVector(), pivot_wider.SpatVector(), replace_na.Spat

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 temp

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

# And pivot long for plot
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, i.e. 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. Note that "geometry" columns is sticky, hence it would be 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() function.

SpatVector**:**

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

See Also

tidyr::pivot_wider()

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

Other **tidyr** methods: drop_na.Spat, fill.SpatVector(), pivot_longer.SpatVector(), replace_na.Spat

Examples

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

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

# Add extra row with info
xtra <- cyl %>%
  slice(c(2, 3)) %>%
  mutate(
    label = "extra",
    value = TRUE
  ) %>%
  rbind(cyl, .) %>%
  glimpse()

# Pivot by geom
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's mostly useful because it looks a little nicer in pipes and it can optionally name the output.

It is possible to extract the geographic coordinates of a `SpatRaster`. You need to use `pull(.data, x, xy = TRUE)`. `x` and `y` are reserved names on terra, since they refer to the geographic coordinates of the layer.

See **Examples** and section **About layer 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).
`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 on to `as_tibble.Spat()`

Value

A vector the same number of cells/geometries as `.data`.

On `SpatRaster` objects, note that the default (`na.rm = FALSE`) would remove empty cells, so you may need to pass (`na.rm = FALSE`) to `...`. See `terra::as.data.frame()`.

terra equivalent

`terra::values()`

Methods

Implementation of the **generic** dplyr::pull() function. This is done by coercing the Spat* object to a tibble first (see as_tibble.Spat) and then using dplyr::pull() method over the tibble.

SpatRaster**:**

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

If xy = TRUE option is passed to ..., two columns names x and y (corresponding to the geographic coordinates of each cell) are available in position 1 and 2. Hence, pull(.data, 1) and pull(.data, 1, xy = TRUE) return different result.

SpatVector**:**

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

See Also

dplyr::pull()

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

Examples

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

# Extract second layer
r %>%
  pull(2) %>%
  head()

# With xy the first two cols 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 on WKT format*

Description

Extract the WKT version of the CRS associated to a string, number of sf/Spat* object.

The Well-known text (WKT) representation of coordinate reference systems (CRS) is a character string that identifies precisely the parameters of each CRS. This is the current standard used on **sf** and **terra** packages.

Usage

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

Arguments

`.data`	Input potentially including or representing a CRS. It could be a `sf/sfc` object, a `SpatRaster/SpatVector` object, a `crs` object from `sf::st_crs()`, a character (for example a proj4 string) or a integer (representing an EPSG code).
`...`	ignored

Details

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

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

While **sf** equivalent is:

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

Knowing the WKT would help to smooth workflows when working with different packages and object types.

Value

A WKT representation of the corresponding CRS.

Internals

This is a thin wrapper of `sf::st_crs()` and `terra::crs()`.

See Also

`terra::crs()`, `sf::st_crs()` for knowing how these packages handle CRS definitions.

Other helpers: `compare_spatrasters()`, `is_grouped_spatvector()`, `is_regular_grid()`

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

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)
```

---

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 than `.data`. See **Methods**.

terra equivalent

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

Methods

Implementation of the **generic** `dplyr::relocate()` function.

`SpatRaster`**:**
Relocate layers of a `SpatRaster`.

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

See Also

`dplyr::relocate()`

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

Other **dplyr** methods: `arrange.SpatVector()`, `bind_cols.SpatVector`, `bind_rows.SpatVector`, `count.SpatVector()`, `distinct.SpatVector()`, `filter-joins.SpatVector`, `filter.Spat`, `glimpse.Spat`, `group-by.SpatVector`, `mutate-joins.SpatVector`, `mutate.Spat`, `pull.Spat`, `rename.Spat`, `rowwise.SpatVector()`, `select.Spat`, `slice.Spat`, `summarise.SpatVector()`

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()`.
`...`	Depending on the function: • For `rename.Spat*()`: <`tidy-select`>. Use `new_name = old_name` to rename selected variables. • For `rename_with()`: 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 than `.data`. See **Methods**.

terra equivalent

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

Methods

Implementation of the **generic** `dplyr::rename()` function.

`SpatRaster`**:**

Rename layers of a `SpatRaster`.

SpatVector:

The result is a SpatVector with the renamed attributes on the function call.

See Also

dplyr::rename()

Other single table verbs: arrange.SpatVector(), filter.Spat, mutate.Spat, select.Spat, slice.Spat, summarise.SpatVector()

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

Other dplyr methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rowwise.SpatVector(), select.Spat, slice.Spat, summarise.SpatVector()

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* NA*s with specified values*

Description

Replace NA values on 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	list of values, with one value for each layer/attribute that has NA values to be replaced.
...	Ignored

Value

A Spat* object of the same class than data. Geometries and spatial attributes are preserved.

terra equivalent

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

See Also

tidyr::replace_na()

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

Other **tidyr** methods: drop_na.Spat, fill.SpatVector(), pivot_longer.SpatVector(), pivot_wider.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()
```

---

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

Description

rowwise() allows you to compute on a SpatVector a row-at-a-time. This is most useful when a vectorised function doesn't exist.

Most **dplyr** verbs implementation in **tidyterra** preserve row-wise grouping, with the exception of summarise.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 identify each row. See dplyr::rowwise().

Details

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

Value

The same SpatVector object with an additional attribute.

Methods

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

When mixing terra and dplyr syntax on a row-wise SpatVector (i.e, subsetting a SpatVector like v[1:3,1:2]) the groups attribute can be corrupted. **tidyterra** would try to re-generate the SpatVector. This would be triggered the next time you use a **dplyr** verb on your SpatVector.

Note also that some operations (as terra::spatSample()) would create a new SpatVector. In these cases, the result won't preserve the groups attribute. Use rowwise.SpatVector() to re-group.

See Also

dplyr::rowwise()

Other **dplyr** verbs that operate on group of rows: count.SpatVector(), group-by.SpatVector, summarise.SpatVector()

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, select.Spat, slice.Spat, 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 on each geometry
nb %>%
  rowwise() %>%
  mutate(nb_mean = mean(c(NWBIR74, NWBIR79)))

# Additional examples

# use c_across() to more easily select many variables
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)))

# Summarising
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_color_coltab` *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 parameters ... would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters 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.

`labels` One of:

- `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.

`minor_breaks` One of:

- `NULL` for no minor breaks

- `waiver()` for the default breaks (one minor break between each major break)
- 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 breaks.

`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`.
`na.translate`	Should `NA` values be removed from the legend? Default is `TRUE`.
`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`	Should unused factor levels be omitted from the scale? The default (`TRUE`) removes unused factors.
`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.
`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

The corresponding **ggplot2** layer with the values applied to the `fill/colour` aesthetics.

See Also

`terra::plot()`, `ggplot2::scale_fill_viridis_c()`

See also **ggplot2** docs on additional ... parameters.

Other gradient scales and palettes for hypsometry: `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")
```

`scale_coltab` *Discrete scales based in the color table of a* `SpatRaster`

Description

Some categorical `SpatRaster` objects may have an associated color table. This function extract those values. These functions generates scales and vector of colors based on the color table `terra::coltab()` associated to a `SpatRaster`.

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

Additional parameters `...` would be passed on to `ggplot2::discrete_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check `ggplot2::discrete_scale()` to see the full range of parameters accepted.

Usage

```
scale_fill_coltab(
  data,
  ...,
  alpha = 1,
  na.translate = FALSE,
```

```
  na.value = "transparent",
  drop = TRUE
)

scale_colour_coltab(
  data,
  ...,
  alpha = 1,
  na.translate = FALSE,
  na.value = "transparent",
  drop = TRUE
)

get_coltab_pal(x)
```

Arguments

data, x	A SpatRaster with one or several 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.
	labels One of: • 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, a number in [0,1], see argument alpha in hsv.

na.translate	Should NA values be removed from the legend? Default is TRUE.
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	Should unused factor levels be omitted from the scale? The default (TRUE) removes unused factors.

Value

The corresponding **ggplot2** layer with the values applied to the fill/colour aesthetics.

See Also

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

Examples

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

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

plot(r)

# Get coltab
coltab_pal <- get_coltab_pal(r)

coltab_pal


# With ggplot2 + tidyterra
library(ggplot2)

gg <- ggplot() +
  geom_spatraster(data = r)


# Default plot
gg

# With coltabs
gg +
  scale_fill_coltab(data = r)
```

`scale_cross_blended` *Cross blended hypsometric tints scales*

Description

Implementation of the cross blended hypsometric gradients presented on 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 parameters `...` would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters 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 info. Values available are: "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.

labels One of:

- 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.

minor_breaks One of:

- NULL for no minor breaks
- waiver() for the default breaks (one minor break between each major break)
- 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 breaks.

`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` Should `NA` values be removed from the legend? Default is `TRUE`.

`drop` Should unused factor levels be omitted 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. A rough description of these tints are:

- Blue colors: Negative values.
- Green colors: 0 to 1.000 values.
- Browns: 1000 to 4.000 values.

- Whites: Values higher than 4.000.

The following orientation would vary depending on the palette definition (see cross_blended_hypsometric_tints_db for an example on how this could be achieved).

Note that the setup of the palette may not be always 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 a range of values on `[100, 200]` would appear almost as an uniform color. This could be adjusted using the `limits`/`values` parameters.

`cross_blended.colors2()` provides a gradient color palette where the distance between colors is different depending of the type of color. In contrast, `cross_blended.colors()` provides an uniform gradient across colors. See **Examples**.

Value

The corresponding **ggplot2** layer with the values applied to the `fill`/`colour` aesthetics.

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.* Accessed June 10, 2022. https://www.shadedrelief.com/hypso/hypso.html

See Also

cross_blended_hypsometric_tints_db, `terra::plot()`, `terra::minmax()`, `ggplot2::scale_fill_viridis_c()`.

See also **ggplot2** docs on additional `...` parameters.

Other gradient scales and palettes for hypsometry: `scale_color_coltab()`, `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 = 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")

# Use hypsometric  tint version...
```

```
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_cross_blended_tint_c(palette = "cold_humid")

# ...but not suitable for the range of the raster: adjust
my_lims <- minmax(volcano2_rast) %>% as.integer() + c(-2, 2)

ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_cross_blended_tint_c(
    palette = "cold_humid",
    limits = my_lims
  )

# 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 limits and breaks
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_cross_blended_tint_b(
    breaks = seq(75, 200, 25),
    palette = "arid",
    limits = my_lims
  )

# 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 fun 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 the cross-blended palettes on version 2

pals <- unique(cross_blended_hypsometric_tints_db$pal)

# Helper fun 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 parameters `...` would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters accepted by these scales.

These palettes are an implementation of `terra::map.pal()`, that is the default color palettes provided by `terra::plot()` (**terra** > `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 info.
...	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.

`labels` One of:

- `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.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- 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 breaks.

`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 Should NA values be removed from the legend? Default is TRUE.

drop Should unused factor levels be omitted 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. Should the scale 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). However, it is possible to modify this behaviour with the use_grass_range argument, When FALSE the color scales would be mapped to the range of values of the color/fill aesthethics, See **Examples**.

Value

The corresponding **ggplot2** layer with the values applied to the fill/colour aes().

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** docs on additional ... parameters:

Other gradient scales and palettes for hypsometry: scale_color_coltab(), scale_cross_blended, 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 = grass.colors(100, palette = "haxby"))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_grass_c(palette = "terrain")

# Use with no 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 fun 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 representing hypsometry and bathymetry*

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**.

Additional parameters `...` would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters 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 info. Values available are: "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".
...	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. labels One of: • 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.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- 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 breaks.

`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` Should `NA` values be removed from the legend? Default is `TRUE`.

`drop` Should unused factor levels be omitted 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. A rough description of these tints are:

- Blue colors: Negative values.
- Green colors: 0 to 1.000 values.
- Browns: 1000 to 4.000 values.
- Whites: Values higher than 4.000.

The following orientation would vary depending on the palette definition (see hypsometric_tints_db for an example on how this could be achieved).

Note that the setup of the palette may not be always 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 a range of values on [100, 200] would appear almost as an uniform color. This could be adjusted using the limits/values parameters.

hypso.colors2() provides a gradient color palette where the distance between colors is different depending of the type of color. In contrast, hypso.colors() provides an uniform gradient across colors. See **Examples**.

Value

The corresponding **ggplot2** layer with the values applied to the fill/colour aesthetics.

Source

cpt-city: http://seaviewsensing.com/pub/cpt-city/.

See Also

hypsometric_tints_db, terra::plot(), terra::minmax(), ggplot2::scale_fill_viridis_c()

See also **ggplot2** docs on additional ... parameters.

Other gradient scales and palettes for hypsometry: scale_color_coltab(), scale_cross_blended, scale_grass, 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 = 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")

# Use hypsometric  tint version...
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_hypso_tint_c(palette = "colombia_hypso")

# ...but not suitable for the range of the raster: adjust
my_lims <- minmax(volcano2_rast) %>% as.integer() + c(-2, 2)

ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_hypso_tint_c(
    palette = "colombia_hypso",
    limits = my_lims
  )

# 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 limits and breaks
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_hypso_tint_b(
    breaks = seq(75, 200, 25),
    palette = "wiki-2.0_hypso",
    limits = my_lims
  )

# 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 cpl_city palettes

pals <- unique(hypsometric_tints_db$pal)

# Helper fun 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 the cpl_city palettes on version 2

pals <- unique(hypsometric_tints_db$pal)

# Helper fun 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 parameters `...` would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters 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. Values available are: "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.

labels One of:

- 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.

minor_breaks One of:

- NULL for no minor breaks

- `waiver()` for the default breaks (one minor break between each major break)
- 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 breaks.

`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`	Should `NA` values be removed from the legend? Default is `TRUE`.
`drop`	Should unused factor levels be omitted 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

The corresponding **ggplot2** layer with the values applied to the `fill/colour` aesthetics.

Source

`https://github.com/LeahSmyth/Princess-Colour-Schemes`.

See Also

`terra::plot()`, `ggplot2::scale_fill_viridis_c()`

See also **ggplot2** docs on additional ... parameters.

Other gradient scales and palettes for hypsometry: `scale_color_coltab()`, `scale_cross_blended`, `scale_grass`, `scale_hypso`, `scale_terrain`, `scale_whitebox`

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 fun 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 colour 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 parameters `...` would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters 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.

`labels` One of:

- `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.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (one minor break between each major break)
- 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 breaks.

`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` Should `NA` values be removed from the legend? Default is `TRUE`.

`drop` Should unused factor levels be omitted 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

The corresponding **ggplot2** layer with the values applied to the `fill/colour` aesthetics.

See Also

`terra::plot()`, `ggplot2::scale_fill_viridis_c()` and **ggplot2** docs on additional `...` parameters.

Other gradient scales and palettes for hypsometry: `scale_color_coltab()`, `scale_cross_blended`, `scale_grass`, `scale_hypso`, `scale_princess`, `scale_whitebox`

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 parameters `...` would be passed on to:

- Discrete values: `ggplot2::discrete_scale()`.
- Continuous values: `ggplot2::continuous_scale()`.
- Binned continuous values: `ggplot2::binned_scale()`.

Note that tidyterra just documents a selection of these additional parameters, check the **ggplot2** functions listed above to see the full range of parameters 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. Values available are: "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.

labels One of:

- 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.

minor_breaks One of:

- NULL for no minor breaks
- waiver() for the default breaks (one minor break between each major break)
- 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 breaks.

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	Should NA values be removed from the legend? Default is TRUE.
drop	Should unused factor levels be omitted 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

The corresponding **ggplot2** layer with the values applied to the fill/colour aesthetics.

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** docs on additional ... parameters.

Other gradient scales and palettes for hypsometry: scale_color_coltab(), scale_cross_blended, scale_grass, scale_hypso, scale_princess, scale_terrain

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 fun 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)
```

`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**.

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 than `.data`. See **Methods**.

terra equivalent

`terra::subset()`

Methods

Implementation of the **generic** `dplyr::select()` function.

`SpatRaster`**:**

Select (and rename) layers of a `SpatRaster`. The result is a `SpatRaster` with the same extent, resolution and crs than `.data`. Only the number (and possibly the name) of layers is modified.

`SpatVector`**:**

The result is a `SpatVector` with the selected (and possibly renamed) attributes on the function call.

See Also

dplyr::select(), terra::subset()

Other single table verbs: arrange.SpatVector(), filter.Spat, mutate.Spat, rename.Spat, slice.Spat, summarise.SpatVector()

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), slice.Spat, summarise.SpatVector()

Examples

```
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 lets you index cells/rows/columns/geometries by their (integer) locations. It allows you to select, remove, and duplicate those dimensions of a `Spat*` object.

If you want to slice your `SpatRaster` **by geographic coordinates** use `filter.SpatRaster()` method.

It is accompanied by a number of 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()` allow to 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, ..., .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)

## S3 method for class 'SpatRaster'
slice_tail(.data, ..., n, prop, .keep_extent = FALSE)

## S3 method for class 'SpatVector'
slice_tail(.data, ..., n, prop)

## 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, 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, 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, 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`	Should the extent of the resulting `SpatRaster` be kept? See also `terra::trim()`, `terra::extend()`.
`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, should cells that present a value of `NA` removed 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.
`replace`	Should sampling be performed with (`TRUE`) or without (`FALSE`, the default) replacement.
`cols, rows`	Integer col/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 than `.data`. See **Methods**.

terra equivalent

`terra::subset()`, `terra::spatSample()`

Methods

Implementation of the **generic** dplyr::slice() function.

SpatRaster**:**

The result is a SpatRaster with the crs and resolution of the input and where cell values of the selected cells/columns/rows are preserved.

Use .keep_extent = TRUE to preserve the extent of .data on the output. The non-selected cells would present 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().

You can get a skeleton of your SpatRaster with the cell, column and row index with as_coordinates().

If you want to slice by geographic coordinates use filter.SpatRaster().

Other single table verbs: arrange.SpatVector(), filter.Spat, mutate.Spat, rename.Spat, select.Spat, summarise.SpatVector()

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

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, summarise.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 SpatVectors--------------------------------------
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` *Summarise 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 summarising 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`.
`...`	<`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, to add multiple columns from a single expression. **[Deprecated]** Returning values with size 0 or >1 was deprecated as of 1.1.0. Please use `reframe()` for this instead.
`.by`	Ignored on this method (**[Experimental]** on **dplyr**).
`.groups`	See `dplyr::summarise()`
`.dissolve`	logical. Should borders between aggregated geometries be dissolved?

Value

A `SpatVector`.

terra equivalent

`terra::aggregate()`

Methods

Implementation of the **generic** `dplyr::summarise()` function.

`SpatVector`**:**

Similarly to the implementation on **sf** this function can be used to dissolve geometries (with `.dissolve = TRUE`) or create `MULTI` versions of geometries (with `.dissolve = FALSE`). See **Examples**.

See Also

dplyr::summarise(), terra::aggregate()

Other single table verbs: arrange.SpatVector(), filter.Spat, mutate.Spat, rename.Spat, select.Spat, slice.Spat

Other **dplyr** verbs that operate on group of rows: count.SpatVector(), group-by.SpatVector, rowwise.SpatVector()

Other **dplyr** methods: arrange.SpatVector(), bind_cols.SpatVector, bind_rows.SpatVector, count.SpatVector(), distinct.SpatVector(), filter-joins.SpatVector, filter.Spat, glimpse.Spat, group-by.SpatVector, mutate-joins.SpatVector, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, rowwise.SpatVector(), select.Spat, slice.Spat

Examples

```
library(terra)
library(ggplot2)

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

# Grouped
gr_v <- v %>%
  mutate(start_with_s = substr(name, 1, 1) == "S") %>%
  group_by(start_with_s)


# Dissolving
diss <- gr_v %>%
  summarise(n = dplyr::n(), mean = mean(as.double(cpro)))

diss

autoplot(diss, aes(fill = start_with_s)) + ggplot2::ggtitle("Dissolved")

# Not dissolving
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::ggtitle("Not Dissolved")
```

---

volcano2 *Updated topographic information on Auckland's Maungawhau volcano*

Description

Probably you already know the volcano dataset. This dataset provides updated information of Maungawhau (Mt. Eden) from Toitu Te Whenua Land Information New Zealand, the Government's 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)
- coord. ref. : 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 5m due to disk space constrains.

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