# Package 'tidyterra'

November 9, 2022

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

**Version** 0.3.1

**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), crayon, data.table, dplyr (>= 1.0.0), ggplot2 (>= 3.1.0), magrittr, rlang, scales, sf (>= 1.0.0), terra (>= 1.5-12), tibble (>= 3.0.0), tidyr (>= 1.0.0)

**Suggests** isoband, knitr, lifecycle, maptiles, rmarkdown, s2, testthat (>= 3.0.0), tidyverse, vdiffr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.1

**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** 2022-11-09 12:40:02 UTC

# R topics documented:



---

`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_spatraster(), 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_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 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(). See **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

**[Questioning]** 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 Also

pull_crs()

Coercing objects: as_coordinates(), 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_tibble.Spat` *Coerce a Spat* object to data frames*

## Description

`as_tibble()` method for SpatRaster and SpatVector.

## 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, ..., .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:<br>• `"minimal"`: No name repair or checks, beyond basic existence,<br>• `"unique"`: Make sure names are unique and not empty,<br>• `"check_unique"`: (default value), no name repair, but check they are `unique`,<br>• `"universal"`: Make the names `unique` and syntactic<br>• a function: apply custom name repair (e.g., `.name_repair = make.names` for names in the style of base R).<br>• A purrr-style anonymous function, see `rlang::as_function()` |

## Value

A tibble.

## terra equivalent

`terra::as.data.frame()`

## Methods

Implementation of the **generic** `tibble::as_tibble()` function.

**SpatRaster:**

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

### About layer names

When coercing SpatRaster objects to data frames, `x` and `y` names are reserved for geographic coordinates of each cell of the raster. 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.

### See Also

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

Coercing objects: `as_coordinates()`, `as_spatraster()`, `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, 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. |
| 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().

## 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_rgb(), geom_spatraster(), 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 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)
```

---

`compare_spatrasters` *Compare attributes of two SpatRasters*

---

## Description

Two SpatRasters are compatible (in terms of combining layers) if the crs, extent and resolution are similar. In those cases you can combine the SpatRasters 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 SpatRasters are compatible, plus an informative message flagging the issues found (if any).

## 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

Other helpers: `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"))
```

---

# 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**,**g**,**b**: Value of the red, green and 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: `hypsometric_tints_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"
  )
```

---

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

---

## Description

`drop_na()` method drops geometries where any attribute specified by `...` contains a missing value.

## Usage

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

## Arguments

| | |
|---|---|
| `data` | A SpatVector created with `terra::vect()`. |
| `...` | `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**.

## 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

## See Also

`tidyr::drop_na()`. **[Questioning]** A method for SpatRaster is also available, see `drop_na.SpatRaster()`.

Other tidyr.methods: `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")
```

---

`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()`.

## 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 internall for filtering. See **Examples**.

### SpatVector:

This method relies on the implementation of `dplyr::filter()` method on the sf package. The result is a SpatVector with all the geometries that produce a value of `TRUE` for all conditions.

## See Also

`dplyr::filter()`

Other dplyr methods: `mutate.Spat`, `pull.Spat`, `relocate.Spat`, `rename.Spat`, `select.Spat`, `slice.Spat`

Other single table verbs: `mutate.Spat`, `rename.Spat`, `select.Spat`, `slice.Spat`

## 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 SpatRasters and SpatVectors to data frames for compatibility with `ggplot2::ggplot()`.

## Usage

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

## S3 method for class 'SpatVector'
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 to methods |
| `.name_repair` | Treatment of problematic column names: |

- `"minimal"`: No name repair or checks, beyond basic existence,
- `"unique"`: Make sure names are unique and not empty,
- `"check_unique"`: (default value), no name repair, but check they are `unique`,
- `"universal"`: Make the names `unique` and syntactic
- 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.

## Value

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

## Methods

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

### 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()`.

### SpatVector:

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_rgb()`, `geom_spatraster()`, `ggspatvector`, `stat_spat_coordinates()`

Other ggplot2 methods: `autoplot.Spat`

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

## Examples

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

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 raster, contours, etc.
library(ggplot2)

# Use here the raster with resample
ggplot(r, maxcell = 10000) +
  # Need the aes parameters
  geom_raster(aes(x, y, fill = elevation)) +
  # Adjust the coords
  coord_equal()

# Or any other geom
ggplot(r) +
  geom_histogram(aes(x = elevation), bins = 20, fill = "lightblue", color = "black")

# 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()

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,
  ...
)

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() or 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. |
| ... | Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat. |
| 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()`).

`value` Values of the SpatRaster.

`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()`

Alternative 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_hypso_c()

# Display facets
ggplot() +
  geom_spatraster(data = temp_rast) +
  facet_wrap(~lyr, ncol = 2) +
  scale_fill_hypso_b()


# 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 = NA) +
  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 = NA)
```

---

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,
  ...
)
```

## 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()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |

## Value

A ggplot2 layer

## terra equivalent

`terra::plotRGB()`

### 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

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

### 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()`.

## 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_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()` or `ggplot2::aes_()`. See **Aesthetics** specially in the use of `fill` aesthetic. |
| `data` | A SpatRaster object. |
| `...` | Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat. |

| | |
|---|---|
| maxcell | positive integer. Maximum number of cells to use for the plot. |
| bins | Number of contour bins. Overridden by binwidth. |
| binwidth | The width of the contour bins. Overridden by breaks. |
| breaks | One of:<br>• Numeric vector to set the contour breaks<br>• 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)).<br>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. |

## Value

A ggplot2 layer

## terra equivalent

terra::contour()

## Aesthetics

geom_spatraster_contour() understands the following aesthetics:

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

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.

### Computed variables

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

`nlevel` Height of contour, scaled to maximum of 1.

`lyr` Name of the layer.

`level_low`, `level_high`, `level_mid` (contour bands only) Lower and upper bin boundaries for each band, as well the mid point between the boundaries.

### 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()`

Other ggplot2 utils: `autoplot.Spat`, `fortify.Spat`, `geom_spatraster_rgb()`, `geom_spatraster()`, `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)


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.<br>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, either as a ggproto Geom subclass or as a string naming the geom stripped of the geom_ prefix (e.g. "point" rather than "geom_point")

position
: Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

## 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 SpatVectors.

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_rgb(), geom_spatraster(), 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()"
  )
```

---

`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**,**g**,**b**: Value of the red, green and blue channel (RGB color mode).
- **hex**: Hex code of the color.

## Source

cpt-city: http://soliton.vm.bytemark.co.uk/pub/cpt-city/.

## See Also

scale_fill_hypso_c()

Other datasets: cross_blended_hypsometric_tints_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()`, `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.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. See `dplyr::mutate()`. |

## 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 **generics** `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:**
This method relies on the implementation of `dplyr::mutate()` method on the sf package. 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()`

Other dplyr methods: `filter.Spat`, `pull.Spat`, `relocate.Spat`, `rename.Spat`, `select.Spat`, `slice.Spat`

Other single table verbs: `filter.Spat`, `rename.Spat`, `select.Spat`, `slice.Spat`

## Examples

```
library(terra)

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

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

mod
plot(mod)

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

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

---

# 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()`.

## 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:<br>• a literal layer/attribute name<br>• a positive integer, giving the position counting from the left<br>• a negative integer, giving the position counting from the right.<br>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()` |

## Value

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

On SpatRasters, 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.

**SpatRaster:**
When using option `na.rm = TRUE`, only cells with a value distinct to `NA` are extracted. See `terra::as.data.frame()`.
If `xy = TRUE` option is used, 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:**
See `terra::as.data.frame()` options.

## See Also

`dplyr::pull()`

Other dplyr methods: `filter.Spat`, `mutate.Spat`, `relocate.Spat`, `rename.Spat`, `select.Spat`, `slice.Spat`

## 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 slightly differs. For example, a sf user could do:

`sf::st_transform(x, 25830)`

While a terra user needs to:

`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()`

Other helpers: `compare_spatrasters()`, `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()` 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:**
This method relies on the implementation of `dplyr::relocate()` method on the sf package. The result is a SpatVector with the attributes on a different order.

## See Also

dplyr::relocate()

Other dplyr methods: filter.Spat, mutate.Spat, pull.Spat, rename.Spat, select.Spat, slice.Spat

## Examples

```
library(terra)


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

names(spatrast)


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

---

# rename.Spat *Rename layers/attributes*

---

## Description

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

## Usage

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

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

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

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

## Arguments

| | |
|---|---|
| .data | A SpatRaster created with terra::rast() or a SpatVector created with terra::vect(). |

| | |
|---|---|
| `...` | For `rename()`: `tidy-select` Use `new_name = old_name` to rename selected variables.<br>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:**
This method relies on the implementation of `dplyr::rename()` method on the sf package. The result is a SpatVector with the renamed attributes on the function call.

## See Also

`dplyr::rename()`

Other dplyr methods: `filter.Spat`, `mutate.Spat`, `pull.Spat`, `relocate.Spat`, `select.Spat`, `slice.Spat`

Other single table verbs: `filter.Spat`, `mutate.Spat`, `select.Spat`, `slice.Spat`

## Examples

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

spatrast

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

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

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

## Description

Replace NAs 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.methods: `drop_na.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()
```

`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**.

## Usage

```
scale_fill_cross_blended_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1
)

scale_colour_cross_blended_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1
)

scale_fill_cross_blended_c(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
```

```
)

scale_colour_cross_blended_c(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_fill_cross_blended_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_cross_blended_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  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
)

scale_colour_cross_blended_tint_d(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1
)

scale_fill_cross_blended_tint_c(
  palette = "cold_humid",
  ...,
```

```
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  guide = "colourbar"
)

scale_colour_cross_blended_tint_c(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  guide = "colourbar"
)

scale_fill_cross_blended_tint_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_cross_blended_tint_b(
  palette = "cold_humid",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  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". |

| | |
|---|---|
| ... | Other arguments passed on to discrete_scale(), continuous_scale(), or binned_scale() to control name, limits, breaks, labels and so forth. |
| 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.value | Missing values will be replaced with this value. |
| guide | A function used to create a guide or its name. See guides() for more information. |
| n | the number of colors ($\geq 1$) 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:<br>• NULL to use the default scale range<br>• A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum<br>• 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, raster of small parts of the globe (and with a limited range of elevations) may not be well represented. As an example, a raster with a range of values on [100, 200] would appear almost as an uniform color.

This could be adjusted using the limits/values provided by **ggplot2**.

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. http://www.shadedrelief.com/hypso/hypso.html

## See Also

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

Other gradient scales and palettes for hypsometry: `scale_hypso`, `scale_terrain`, `scale_whitebox`, `scale_wiki`

## Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = 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(
      direction = "horizontal",
      title.position = "top",
      barwidth = 25
    )
  ) +
  labs(fill = "elevation (m)") +
  theme_minimal() +
  theme(legend.position = "bottom")


# 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_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**.

## Usage

```
scale_fill_hypso_d(palette = "etopo1_hypso", ..., alpha = 1, direction = 1)

scale_colour_hypso_d(palette = "etopo1_hypso", ..., alpha = 1, direction = 1)

scale_fill_hypso_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_colour_hypso_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_fill_hypso_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
```

```
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_hypso_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

hypso.colors(n, palette = "etopo1_hypso", alpha = 1, rev = FALSE)

scale_fill_hypso_tint_d(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1
)

scale_colour_hypso_tint_d(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1
)

scale_fill_hypso_tint_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  guide = "colourbar"
)

scale_colour_hypso_tint_c(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
```

```
  na.value = NA,
  guide = "colourbar"
)

scale_fill_hypso_tint_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_hypso_tint_b(
  palette = "etopo1_hypso",
  ...,
  alpha = 1,
  direction = 1,
  values = NULL,
  limits = NULL,
  na.value = NA,
  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". |
| ... | Other arguments passed on to discrete_scale(), continuous_scale(), or binned_scale() to control name, limits, breaks, labels and so forth. |
| 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.value | Missing values will be replaced with this value. |
| guide | A function used to create a guide or its name. See guides() for more information. |

`n` the number of colors ($\geq 1$) 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, raster of small parts of the globe (and with a limited range of elevations) may not be well represented. As an example, a raster with a range of values on `[100, 200]` would appear almost as an uniform color.

This could be adjusted using the `limits`/`values` provided by **ggplot2**.

`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://soliton.vm.bytemark.co.uk/pub/cpt-city/.

## See Also

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

Other gradient scales and palettes for hypsometry: scale_cross_blended, scale_terrain, scale_whitebox, scale_wiki

## Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = 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(
```

```
      direction = "horizontal",
      title.position = "top",
      barwidth = 25
    )
  ) +
  labs(fill = "elevation (m)") +
  theme_minimal() +
  theme(legend.position = "bottom")


# 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_terrain` *Terrain colour scales from grDevices*

---

## Description

Implementation of the classic color palettes used by default by the terra package (see `terra::plot()`):

- `scale_*_terrain_d()`: For discrete values.
- `scale_*_terrain_c()`: For continuous values.
- `scale_*_terrain_b()`: For binning continuous values.

## Usage

```
scale_fill_terrain_d(..., alpha = 1, direction = 1)

scale_colour_terrain_d(..., alpha = 1, direction = 1)

scale_fill_terrain_c(
  ...,
```

```
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_colour_terrain_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_fill_terrain_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_terrain_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)
```

## Arguments

| | |
|---|---|
| `...` | Other arguments passed on to `discrete_scale()`, `continuous_scale()`, or `binned_scale()` to control name, limits, breaks, labels and so forth. |
| `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.value` | Missing values will be replaced with this value. |
| `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/color` aesthetics.

## See Also

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

Other gradient scales and palettes for hypsometry: scale_cross_blended, scale_hypso, scale_whitebox, scale_wiki

## Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_terrain_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_terrain_b(breaks = seq(70, 200, 10))

# With discrete values
factor <- volcano2_rast %>% mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))


ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_terrain_d(na.value = "gray10")
```

---

# scale_whitebox *Gradient scales from WhiteboxTools color schemes*

---

## Description

Implementation of the gradient palettes provided by WhiteboxTools. Three scales are provided:

- `scale_*_whitebox_d()`: For discrete values.
- `scale_*_whitebox_c()`: For continuous values.
- `scale_*_whitebox_b()`: For binning continuous values.

Additionally, a color palette `whitebox.colors()` is provided. See also `grDevices::terrain.colors()` for details.

## Usage

```
scale_fill_whitebox_d(palette = "high_relief", ..., alpha = 1, direction = 1)

scale_colour_whitebox_d(palette = "high_relief", ..., alpha = 1, direction = 1)

scale_fill_whitebox_c(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_colour_whitebox_c(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_fill_whitebox_b(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_whitebox_b(
  palette = "high_relief",
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  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". |
| ... | Other arguments passed on to discrete_scale(), continuous_scale(), or binned_scale() to control name, limits, breaks, labels and so forth. |
| 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.value | Missing values will be replaced with this value. |
| guide | A function used to create a guide or its name. See guides() for more information. |
| n | the number of colors ($\geq 1$) 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()

Other gradient scales and palettes for hypsometry: scale_cross_blended, scale_hypso, scale_terrain, scale_wiki

## Examples

```
filepath <- system.file("extdata/volcano2.tif", package = "tidyterra")

library(terra)
volcano2_rast <- rast(filepath)

# Palette
plot(volcano2_rast, col = whitebox.colors(100))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_whitebox_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_whitebox_b(breaks = seq(70, 200, 10), palette = "atlas")
```

```
# With discrete values
factor <- volcano2_rast %>% mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))


ggplot() +
  geom_spatraster(data = factor, aes(fill = cats)) +
  scale_fill_whitebox_d(na.value = "gray10", palette = "soft")


# Display all the whitebox palettes

pals <- c(
  "atlas", "high_relief", "arid", "soft", "muted", "purple",
  "viridi", "gn_yl", "pi_y_g", "bl_yl_rd", "deep"
)

# Helper 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)
```

---

`scale_wiki` *Gradient scales from Wikipedia color schemes*

---

## Description

Implementation based on the Wikipedia Colorimetric conventions for topographic maps. Three scales are provided:

- `scale_*_wiki_d()`: For discrete values.
- `scale_*_wiki_c()`: For continuous values.
- `scale_*_wiki_b()`: For binning continuous values.

Additionally, a color palette `wiki.colors()` is provided. See also `grDevices::terrain.colors()` for details.

## Usage

```
scale_fill_wiki_d(..., alpha = 1, direction = 1)

scale_colour_wiki_d(..., alpha = 1, direction = 1)

scale_fill_wiki_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_colour_wiki_c(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "colourbar"
)

scale_fill_wiki_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

scale_colour_wiki_b(
  ...,
  alpha = 1,
  direction = 1,
  na.value = NA,
  guide = "coloursteps"
)

wiki.colors(n, alpha = 1, rev = FALSE)
```

## Arguments

| | |
|---|---|
| `...` | Other arguments passed on to `discrete_scale()`, `continuous_scale()`, or `binned_scale()` to control name, limits, breaks, labels and so forth. |
| `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.value | Missing values will be replaced with this value. |
| guide | A function used to create a guide or its name. See guides() for more information. |
| n | the number of colors ($\geq 1$) 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()

Other gradient scales and palettes for hypsometry: scale_cross_blended, 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 = wiki.colors(100))

library(ggplot2)
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_wiki_c()

# Binned
ggplot() +
  geom_spatraster(data = volcano2_rast) +
  scale_fill_wiki_b(breaks = seq(70, 200, 10))

# With discrete values
factor <- volcano2_rast %>% mutate(cats = cut(elevation,
  breaks = c(100, 120, 130, 150, 170, 200),
  labels = c(
    "Very Low", "Low", "Average", "High",
    "Very High"
  )
))


ggplot() +
```

```
geom_spatraster(data = factor, aes(fill = cats)) +
scale_fill_wiki_d(na.value = "gray10")
```

---

`select.Spat` *Subset layers/attributes of Spat* objects*

---

## Description

Select (and optionally rename) attributes/layers in Spat* objects, using a concise mini-language. See **Methods**.

## 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:**

This method relies on the implementation of `dplyr::select()` method on the sf package. The result is a SpatVector with the selected (and possibly renamed) attributes on the function call.

## See Also

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

Other dplyr methods: filter.Spat, mutate.Spat, pull.Spat, relocate.Spat, rename.Spat, slice.Spat

Other single table verbs: filter.Spat, mutate.Spat, rename.Spat, slice.Spat

## 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()` 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 raster 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)

## 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)

## 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.<br>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.<br>If a negative value of n or prop is provided, the specified number or proportion of rows will be removed.<br>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. If the proportion of a group size does not yield an integer number of rows, the absolute value of prop*nrow(.data) is rounded down. |
| order_by | Variable or function of variables to order by. |
| 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. |
| weight_by | 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:**
This method relies on the implementation of `dplyr::slice()` method on the sf package. The result is a SpatVector where the attributes of the selected geometries are preserved.

## 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 dplyr methods: `filter.Spat`, `mutate.Spat`, `pull.Spat`, `relocate.Spat`, `rename.Spat`, `select.Spat`

Other single table verbs: `filter.Spat`, `mutate.Spat`, `rename.Spat`, `select.Spat`

## 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()
```

`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 raster 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`, `hypsometric_tints_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