# Package ‘ggincerta’

November 11, 2025

**Title** Extend 'ggplot2' with Layers and Scales for Spatial Uncertainty Visualization

**Version** 0.1.0

**Description** Provide specialized 'ggplot2' layers and scales for spatial uncertainty visualization, including bivariate choropleth maps, pixel maps, glyph maps, and exceedance probability maps.

**Imports** dplyr, grDevices, grid, gtable, rlang, scales, sf, stats, withr

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Suggests** testthat (>= 3.0.0), vdiffr

**Config/testthat/edition** 3

**Depends** ggplot2 (>= 3.5.0), R (>= 4.1.0)

**LazyData** true

**URL** https://github.com/maggiexma/ggincerta

**BugReports** https://github.com/maggiexma/ggincerta/issues

**NeedsCompilation** no

**Author** Xueqi Ma [aut, cre, cph],
Emi Tanaka [aut, ths] (ORCID: <https://orcid.org/0000-0002-1455-259X>),
Weihao Li [ths],
Quan Vu [ths],
Francis Hui [ths]

**Maintainer** Xueqi Ma <maggiexma07@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-11-11 22:00:02 UTC

## Contents

---

`duo` *Format input and assign the "map" class*

---

### Description

`duo()` and `duo_exceed()` create paired mapping objects that combine two variables, record their names, and assign the bivariate/exceed class as an attribute for use in aesthetic mappings.

### Usage

```
duo(v1, v2)

duo_exceed(estimate, error)
```

### Arguments

| | |
|---|---|
| `v1, v2` | Input variables for `duo()`. |
| `estimate, error` | Input variables for `duo_exceed()` representing the point estimate and its uncertainty. |

### Value

A list-like object containing pairs of values from the two variables, with attributes storing the variable names and the class.

### Examples

```
value <- nc$value
sd <- nc$sd
res <- duo(value, sd)
res_exceed <- duo_exceed(value, sd)
class(res); class(res_exceed)
attr(res, "vars"); attr(res_exceed, "vars")
```

---

`nc` *North Carolina SIDS data*

---

## Description

The dataset `nc` is derived from the North Carolina shapefile (`nc.shp`) included in the **sf** package. Two random variables, `value` and `sd`, have been added for demonstration purposes.

Further details about the original data can be found in the spdep package vignette.

## Usage

```
nc
```

## Format

A `sf` object.

## Examples

```
head(nc)

plot(sf::st_geometry(nc))
```

---

`ScaleBivariate` *Bivariate color scales*

---

## Description

`scale_*_bivariate` creates a bivariate palette by mixing two colour ramps, then implements the mapping by binning the variables `v1`, `v2` and assigning each bin combination to a colour.

## Usage

```
ScaleBivariate

scale_fill_bivariate(
  name1 = NULL,
  name2 = NULL,
  colors = c("gold", "red4"),
  n_breaks = 4,
  breaks = c("quantile", "equal"),
  flip = c("none", "vertical", "horizontal", "both"),
  guide_size = 1.5,
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "fill",
  ...
)

scale_color_bivariate(
  name1 = NULL,
  name2 = NULL,
  colors = c("gold", "red4"),
  n_breaks = 4,
  breaks = c("quantile", "equal"),
  flip = c("none", "vertical", "horizontal", "both"),
  guide_size = 1.5,
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "colour",
  ...
)

scale_colour_bivariate(
  name1 = NULL,
  name2 = NULL,
  colors = c("gold", "red4"),
  n_breaks = 4,
  breaks = c("quantile", "equal"),
  flip = c("none", "vertical", "horizontal", "both"),
  guide_size = 1.5,
  na.value = NA,
  na.translate = TRUE,
  aesthetics = "colour",
  ...
)
```

## Arguments

| | |
|---|---|
| `name1, name2` | Optional names for `v1` and `v2`. Used as axis titles in the legend. If `NULL`, the default, the names are taken from the mapping. |
| `colors` | A character vector of length two specifying the colors for the bivariate palette. |
| `n_breaks` | An integer guiding the number of bins for each variable. |
| `breaks` | Method used to bin the variables: `"quantile"` (the default) or `"equal"`. |
| `flip` | Method used to flip the legend: `"none"` (the default), `"vertical"`, `"horizontal"`, or `"both"`. |
| `guide_size` | A numeric value controlling the size of the legend graphic, in centimeters. |
| `na.value` | If `na.translate = TRUE`, what aesthetic value should the missing values be displayed as? Does not apply to position scales where `NA` is always placed at the far right. |
| `na.translate` | Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`. |

| | |
|---|---|
| `aesthetics` | The names of the aesthetics that this scale works with. |
| `...` | Other arguments passed to `ggplot2::discrete_scale()`. |

## Format

An object of class `ScaleBivariate` (inherits from `ScaleDiscrete`, `Scale`, `ggproto`, `gg`) of length 5.

## Value

A `ScaleBivariate` ggproto object.

## See Also

ggplot2::Scale for the base `ggproto` class that all scale objects inherit from.

## Examples

```
# Create a bivariate fill scale
sc <- scale_fill_bivariate()
class(sc)
sc$palette(9)

# Basic bivariate map
p <- ggplot(nc) + geom_sf(aes(fill = duo(value, sd)))

# Customize axis labels
p + scale_fill_bivariate(name1 = 'var1', name2 = 'var2')
```

---

`ScaleExceed` *Exceedance probability colour scales*

---

## Description

`scale_*_exceed` computes exceedance probabilities from a specified distribution and maps onto a continuous gradient colour scale.

## Usage

```
ScaleExceed

scale_fill_exceed(
  name = NULL,
  palette = "Oranges",
  type = "seq",
  direction = 1,
  dist_fun = NULL,
  threshold = 1.64,
  limits = c(0, 1),
  na.value = NA,
  guide = "colourbar",
  aesthetics = "fill",
  ...
)

scale_colour_exceed(
  name = NULL,
  palette = "Oranges",
  type = "seq",
  direction = 1,
  dist_fun = NULL,
  threshold = 1.64,
  limits = c(0, 1),
  na.value = NA,
  guide = "colourbar",
  aesthetics = "colour",
  ...
)
```

## Arguments

| | |
|---|---|
| `name` | The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted. |
| `palette` | A palette function that when called with a numeric vector with values between 0 and 1 returns the corresponding output values (e.g., `scales::pal_area()`). |
| `type` | One of "seq" (sequential), "div" (diverging) or "qual" (qualitative) |
| `direction` | Sets the order of colours in the scale. If 1, the default, colours are as output by `RColorBrewer::brewer.pal()`. If -1, the order of colours is reversed. |
| `dist_fun` | A function used to compute the exceedance probability. If `NULL` (the default), a normal distribution with `stats::pnorm()` is used. |
| `threshold` | A numeric value specifying the threshold `q` in the exceedance probability expression `P(X > q)`. |
| `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()`). |
| `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. |
| `aesthetics` | The names of the aesthetics that this scale works with. |
| `...` | Other arguments passed to `ggplot2::continuous_scale()`. |

## Format

An object of class `ScaleExceed` (inherits from `ScaleContinuous`, `Scale`, `ggproto`, `gg`) of length 2.

## Value

A `ScaleExceed` ggproto object.

## Examples

```
# Create an exceedance probability scale
sc <- scale_fill_exceed()
class(sc)

# Basic bivariate map
p <- ggplot(nc) + geom_sf(aes(fill = duo_exceed(value, sd)))
```

---

`StatGlyph` *Generate glyph maps on sf objects*

---

## Description

`geom_sf_glyph()` adds a glyph map layer based on simple feature (sf) objects to a ggplot. A glyph map is essentially a centroid-based map, where each region is represented by a rotated glyph, and the rotation angle indicates the value of `v2` specified in the mapping.

## Usage

```
StatGlyph

geom_sf_glyph(
  mapping = NULL,
  data = NULL,
  size = 70,
  style = "icone",
  max_v2 = NULL,
  position = "identity",
  show.legend = TRUE,
  inherit.aes = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `ggplot2::aes()`. `v1` and `v2` are required, which are the variables used for glyph fill and rotation, respectively. |
| `data` | The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`). |
| `size` | A positive numeric scaling factor controlling glyph size. Larger values produce smaller glyphs. |
| `style` | Either `"icone"` or `"semi"`. Controls the glyph shape. |
| `max_v2` | Numeric value setting the upper limit for `v2`. |
| `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. |
| `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. |
| `inherit.aes` | If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`. |
| `...` | 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.

## Format

An object of class `StatGlyph` (inherits from `StatSf`, `Stat`, `ggproto`, `gg`) of length 3.

## Value

A list of ggplot2 layer objects.

## Glyph map layer contents

The layer returned by `geom_sf_glyph()` actually contains two scales, corresponding to the two variables specified in the mapping. Therefore, modifying the scale for `v1` will trigger a warning indicating that the scale for `fill` is being replaced.

## Examples

```
# Basic glyph map
p <- ggplot(nc) + geom_sf_glyph(mapping = aes(v1 = value, v2 = sd))
p1 <- ggplot(nc) + geom_sf_glyph(mapping = aes(v1 = value, v2 = sd), style = "semi")

# Customize labels and theme
p + labs(title = "glyph map on nc") + theme(legend.position = "left", legend.box = "horizontal")

# Replacing the internal fill scale triggers a message
# ("Scale for fill is already present. Adding another scale for fill...")
p + scale_fill_distiller(palette = "Blues")
```

---

`StatPixel` *Generate pixel maps on sf objects*

---

## Description

`geom_sf_pixel()` adds a pixel map layer based on simple feature (sf) objects to a ggplot. In a pixel map, each region is divided into small pixels, with colours mapped from values sampled from distribution specified.

## Usage

```
StatPixel

geom_sf_pixel(
  mapping = NULL,
  data = NULL,
  n = 40,
  distribution = "uniform",
  seed = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `ggplot2::aes()`. `v1` and `v2` are required, which are the variables used as the parameters in the sampling distribution. |
| `data` | The data to be displayed in this layer. There are three options:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.<br>A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.<br>A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`). |
| `n` | integer of length 1 or 2, number of grid cells in x and y direction (columns, rows) |
| `distribution` | Sampling distribution: `"uniform"`(the default) or, `"normal"`.<br>• `"uniform"` treats `v1` as the centre and uniformly samples within the range `(v1 - v2, v1 + v2)`.<br>• `"normal"` treats `v1` as the mean and `v2` as the standard deviation. |
| `seed` | Random seed used to ensure reproducibility of the sampling process. |
| `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. |
| `inherit.aes` | If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`. |

`...` 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.

## Format

An object of class `StatPixel` (inherits from `StatSf`, `Stat`, `ggproto`, `gg`) of length 3.

## Value

A list of ggplot2 layer objects.

## Pixel map layer contents

The layer returned by `geom_sf_pixel()` internally includes a scale object created by `scale_fill_distiller()`. Therefore, modifying the scale will trigger a message indicating that the scale for `fill` is being replaced.

## Examples

```
# Basic pixel map
p <- ggplot(nc) + geom_sf_pixel(mapping = aes(v1 = value, v2 = sd), n = 20)

# Replacing the internal fill scale triggers a message
# ("Scale for fill is already present. Adding another scale for fill...")
p + scale_fill_distiller(palette = "Blues")
```

# Index