Package: plot2 (via r-universe)

October 11, 2026

Title A Plotting Assistant for Fast 'ggplot2' Visualisations

Date 2026-04-16

Version 1.99.0.9031

Description A streamlined extension of 'ggplot2' designed to simplify and accelerate the creation of data visualisations. 'plot2' automates common tasks such as axis handling, plot type selection, and data transformation, allowing users to create complex, publication-ready plots with minimal code. It integrates seamlessly with the tidyverse and retains full compatibility with 'ggplot2', while offering additional conveniences like enhanced sorting, faceting, and custom theming.

URL https://msberends.github.io/plot2, https://github.com/msberends/plot2

Depends R (>= 4.1.0)

Imports broom (>= 1.0.0), cleaner (>= 1.5.1), crayon (>= 1.5.0), dplyr (>= 1.1.0), forcats (>= 1.0.0), ggplot2 (>= 4.0.0), showtext (>= 0.9.0), showtextdb (>= 3.0.0), sysfonts (>= 0.8.0), tidyselect (>= 1.2.0), tidyr (>= 1.3.0)

Suggests AMR (>= 3.0.1), arrow, bslib, cli, clipr, DT, fansi, ggbeeswarm (>= 0.7.0), ggforce (>= 0.5.0), glue, ggplot2movies, grDevices, grid (>= 4.5.0), haven, jsonlite, nanoparquet, readODS, readxl (>= 1.3.0), rmarkdown (>= 2.11), knitr (>= 1.30), lubridate, patchwork, pillar, plotly (>= 4.8.0), rio, rlang, rstudioapi (>= 0.10.0), scales, sf (>= 1.0.0), shiny (>= 1.10.0), shinyjs, stringr (>= 1.5.0), testthat, vctrs, viridisLite, xml2, yaml

License GPL-2

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 2

VignetteBuilder knitr

Config/pak/sysreqs libfreetype6-dev libicu-dev libpng-dev zlib1g-dev

Repository https://msberends.r-universe.dev

Date/Publication 2026-07-12 18:14:34 UTC

RemoteUrl https://github.com/msberends/plot2

RemoteRef HEAD

RemoteSha d09e04df4c8c6f883e1e7bb4b44cdfc5b1e43c70

Contents

add_mapping . . . 2
add_type . . . 3
admitted_patients . . . 9
as_plotly . . . 9
coord_spider . . . 10
create_interactively . . . 13
dec_mark . . . 14
get_colour . . . 15
get_plot_title . . . 19
labellers . . . 20
md_to_expression . . . 20
move_layer . . . 22
netherlands . . . 22
plot2 . . . 23
plot2-methods . . . 38
plot2-options . . . 66
theme_minimal2 . . . 67

Index **69**

`add_mapping` *Add Additional Mapping*

Description

This function can be used to adjust the mapping of a plot.

Usage

```
add_mapping(plot, ...)
```

Arguments

`plot`	A `ggplot2` plot.
`...`	Arguments passed on to `ggplot2::aes()`.

Examples

```
p <- iris |> plot2(Sepal.Length, Sepal.Width, Species, zoom = TRUE)
p

p |> add_mapping(shape = Species)
```

---

add_type *Add Plot Element*

Description

Quickly and conveniently add a new 'geom' to an existing plot2/ggplot model. Like plot2(), these functions support tidy evaluation, meaning that variables can be unquoted. Better yet, they can contain any function with any output length, or any vector. They can be added using the pipe (new base R's |> or tidyverse's %>%).

Usage

```
add_type(plot, type = NULL, mapping = aes(), ..., data = NULL, move = 0)

add_line(
  plot,
  y = NULL,
  x = NULL,
  colour = getOption("plot2.colour", "ggplot2"),
  linetype,
  linewidth,
  ...,
  inherit.aes = NULL,
  move = 0,
  legend.value = NULL
)

add_point(
  plot,
  y = NULL,
  x = NULL,
  colour = getOption("plot2.colour", "ggplot2"),
  size,
  shape,
  ...,
  inherit.aes = NULL,
  move = 0,
  legend.value = NULL
)

add_col(
```

```
  plot,
  y = NULL,
  x = NULL,
  colour = getOption("plot2.colour", "ggplot2"),
  colour_fill,
  width,
  ...,
  inherit.aes = NULL,
  move = 0,
  legend.value = NULL
)

add_errorbar(
  plot,
  min,
  max,
  colour = getOption("plot2.colour", "ggplot2"),
  width = 0.5,
  ...,
  inherit.aes = NULL,
  move = 0
)

add_smooth(
  plot,
  y = NULL,
  x = NULL,
  colour = getOption("plot2.colour", "ggplot2"),
  linetype,
  linewidth,
  formula,
  method,
  se,
  ...,
  inherit.aes = NULL,
  move = 0,
  legend.value = NULL
)

add_sf(
  plot,
  sf_data,
  colour = getOption("plot2.colour_sf", "grey50"),
 colour_fill = getOption("plot2.colour_sf_fill", getOption("plot2.colour", "ggplot2")),
  size = 2,
  linewidth = 0.1,
  datalabels = NULL,
  datalabels.colour = "black",
```

```
  datalabels.size = 3,
  datalabels.angle = 0,
  datalabels.font = getOption("plot2.font"),
  datalabels.nudge_y = 2500,
  ...,
  inherit.aes = FALSE
)
```

Arguments

plot	A ggplot2 plot.
type	A ggplot2 geom name, all geoms are supported. Full function names can be used (e.g., "geom_line"), but they can also be abbreviated (e.g., "l", "line"). These geoms can be abbreviated by their first character: area ("a"), boxplot ("b"), column ("c"), histogram ("h"), jitter ("j"), line ("l"), point ("p"), ribbon ("r"), violin ("v").
mapping	A mapping created with aes() to pass on to the geom.
data	Data to use in mapping.
move	Number of layers to move the newly added geom down, e.g., move = 1 will place the newly added geom down 1 layer, thus directly under the highest layer.
x, y	aesthetic arguments
colour, colour_fill	colour of the line or column, will be evaluated with get_colour(). If colour_fill is missing but colour is given, colour_fill will inherit the colour set with colour.
linetype, linewidth, shape, size, width, ...	Arguments passed on to the geom.
inherit.aes	A logical to indicate whether the default aesthetics should be inherited, rather than combining with them.
legend.value	Text to show in an additional legend that will be created. Since ggplot2 does not actually support this, it may give some false-positive warnings or messages, such as "Removed 1 row containing missing values or values outside the scale range".
min, max	minimum (lower) and maximum (upper) values of the error bars
formula	Formula to use in smoothing function, eg. y ~ x, y ~ poly(x, 2), y ~ log(x). NULL by default, in which case method = NULL implies formula = y ~ x when there are fewer than 1,000 observations and formula = y ~ s(x, bs = "cs") otherwise.
method	Smoothing method (function) to use, accepts either NULL or a character vector, e.g. "lm", "glm", "gam", "loess" or a function, e.g. MASS::rlm or mgcv::gam, stats::lm, or stats::loess. "auto" is also accepted for backwards compatibility. It is equivalent to NULL. For method = NULL the smoothing method is chosen based on the size of the largest group (across all panels). stats::loess() is used for less than 1,000 observations; otherwise mgcv::gam() is used with formula = y ~ s(x, bs = "cs")

with `method = "REML"`. Somewhat anecdotally, `loess` gives a better appearance, but is $O(N^2)$ in memory, so does not work for larger datasets.

If you have fewer than 1,000 observations but want to use the same `gam()` model that `method = NULL` would use, then set `method = "gam", formula = y ~ s(x, bs = "cs")`.

`se` Display confidence band around smooth? (`TRUE` by default, see `level` to control.)

`sf_data` an 'sf' data.frame

`datalabels` a column of `sf_data` to add as label below the points

`datalabels.colour`, `datalabels.size`, `datalabels.angle`, `datalabels.font` properties of `datalabels`

`datalabels.nudge_y` is `datalabels` is not `NULL`, the amount of vertical adjustment of the datalabels (positive value: more to the North, negative value: more to the South)

Details

The function `add_line()` will add:

- `geom_hline()` if only `y` is provided and it contains one unique value;
- `geom_vline()` if only `x` is provided and it contains one unique value;
- `geom_line()` in all other cases.

The function `add_errorbar()` only adds error bars to the `y` values, see *Examples*.

Value

a `ggplot` object

Examples

```
head(iris)

p <- iris |>
  plot2(x = Sepal.Length,
        y = Sepal.Width,
        category = Species,
        zoom = TRUE)
p

# if not specifying x or y, current plot data are taken
p |> add_line()

p |> add_smooth()

# single values for add_line() will plot 'hline' or 'vline'
# even considering the `category` if set
p |>
  add_line(y = mean(Sepal.Width))
```

```
# set `colour` to ignore existing colours
# and use `legend.value` to add a legend
p |>
  add_line(y = mean(Sepal.Width),
           colour = "red",
           legend.value = "Average")

# also works with facets
iris |>
  plot2(x = Sepal.Length,
        y = Sepal.Width,
        facet = Species) |>
  add_line(y = mean(Sepal.Width),
           legend.value = "Average")

iris |>
  plot2(x = Sepal.Length,
        y = Sepal.Width,
        facet = Species) |>
  add_line(y = range(Sepal.Width),
           legend.value = "Range")


p |>
  add_line(x = mean(Sepal.Length)) |>
  add_line(y = mean(Sepal.Width))

p |>
  add_point(x = median(Sepal.Length),
            y = median(Sepal.Width),
            shape = 13,
            size = 25,
            show.legend = FALSE)

# multiple values will just plot multiple lines
p |>
  add_line(y = fivenum(Sepal.Width),
           colour = "blue",
           legend.value = "Tukey's Numbers")

p |>
  add_line(y = quantile(Sepal.Width, c(0.25, 0.5, 0.75)),
           colour = c("red", "black", "red"),
           linewidth = 1)

# use move to move the new layer down
p |>
  add_point(size = 5,
            colour = "lightgrey",
            move = -1)

# providing x and y will just plot the points as new data,
p |>
```

```
  add_point(y = 2:4,
            x = 5:7,
            colour = "red",
            size = 5)
# even with expanded grid if x and y are not of the same length
p |>
  add_point(y = 2:4,
            x = 5:8,
            colour = "red",
            size = 5)

# any mathematical transformation of current values is supported
df <- data.frame(var_1 = c(1:100),
                 var_2 = rnorm(100, 100, 25),
                 var_3 = rep(LETTERS[1:5], 5))
df |>
  plot2(var_1, var_2) |>
  add_line(y = mean(var_2),
           linetype = 3,
           legend.value = "Average") |>
  add_col(y = var_2 / 5,
          width = 0.25,
          colour = "blue",
          legend.value = "This *is* **some** symbol: $beta$")

# plotting error bars was never easier
library("dplyr", warn.conflicts = FALSE)
df2 <- df |>
  as_tibble() |>
  slice(1:25) |>
  filter(var_1 <= 50) |>
  mutate(error1 = var_2 * 0.9,
         error2 = var_2 * 1.1)

df2

df2 |>
 plot2(var_1, var_2, var_3, type = "col", datalabels = FALSE, alpha = 0.25, width = 0.75) |>
  # adding error bars was never easier - just reference the lower and upper values
  add_errorbar(error1, error2)

# adding sf objects is just as convenient as all else
netherlands |>
  plot2()
netherlands |>
  plot2(colour_fill = "viridis", colour_opacity = 0.75)
netherlands |>
  plot2(colour_fill = "viridis", colour_opacity = 0.75) |>
  # add the same sf object, but now with empty fill and other options
  add_sf(netherlands,
         colour = "black",
         colour_fill = NA,
         linetype = 2,
```

```
            linewidth = 0.5)
```

`admitted_patients` *Example Data Set with Admitted Patients*

Description

An auto-generated data set containing fictitious patients admitted to hospitals.

Usage

```
admitted_patients
```

Format

A tibble/data.frame with 250 observations and 7 variables:

- `date`
  date of hospital admission
- `patient_id`
  ID of the patient (fictitious)
- `gender`
  gender of the patient
- `age`
  age of the patient
- `age_group`
  age group of the age of the patient, generated with `AMR::age_groups()`
- `hospital`
  ID of the hospital, from A to D
- `ward`
  type of ward, either ICU or Non-ICU

`as_plotly` *Create Interactive Plotly*

Description

Transform a `ggplot2`/`plot2` object to an interactive plot using the Plotly R Open Source Graphing Library.

Usage

```
as_plotly(plot, ...)

plotly_style(plot, ...)
```

Arguments

plot	A ggplot2 plot.
...	In case of as_plotly(): arguments to pass on to layout() to change the Plotly layout object In case of plotly_style(): arguments to pass on to style() to change the Plotly style object

Examples

```
mtcars |>
  plot2(mpg, hp) |>
  as_plotly()

mtcars |>
  plot2(mpg, hp) |>
  as_plotly(dragmode = "pan") |>
  plotly_style(marker.line.color = "red",
               hoverinfo = "y")
```

---

coord_spider *Spider (Radar) Coordinate System for ggplot2*

Description

coord_spider() implements a new spider / radar coordinate system using a linear (non-munched) polar-like transform. It is intended for spider charts where connected paths are drawn as *straight chords* between axes (not circular arcs), and where background grid rings are polygons (not circles). Compared to ggplot2::coord_radial(), this coordinate system:

- uses is_linear() = TRUE to prevent path "munching" into arcs;
- supports a polygonal background grid (rings + spokes);
- optionally draws axis labels (theta) and radial (r) labels using grid grobs in the panel background.

Typical usage is to map a discrete variable (categories) to x (theta) and a continuous variable (scores) to y (radius), then draw a polygon with geom_polygon().

plot2() uses this novel coordinate system to plot spider plots, created using plot2(..., type = "spider").

Usage

```
coord_spider(
  theta = "x",
  start = 0,
  clip = "off",
  inner.radius = 0,
  outer.radius = 0.8,
```

```
  axis_labels = TRUE,
  axis_label_size = 9,
  axis_label_pad_mm = 1,
  r_labels = TRUE,
  r_label_size = 8,
  r_label_pad_mm = 0
)

CoordSpider
```

Arguments

`theta` Character scalar specifying which position aesthetic is interpreted as the angular coordinate (theta). Must be `"x"` or `"y"`. In the common spider-chart case, use `theta = "x"` with a discrete `x` scale (categories around the circle) and a continuous `y` scale (radius). Setting `theta = "y"` swaps these roles.

`start` Numeric scalar giving the angular offset in **radians** for the first axis. Angles increase according to the `direction` field stored in the coord (this implementation fixes `direction = 1` internally). With `start = 0`, the first axis is at 12 o'clock (positive vertical). Use `start = pi/2` to rotate the first axis to 3 o'clock, etc.

`clip` Character scalar passed through to the coordinate system to control clipping of grobs to the panel. Use `"on"` to clip all drawing (including axis labels drawn in `render_bg()`) to the panel region, and `"off"` to allow grobs to extend outside the panel. When `clip = "on"`, labels that lie beyond the panel boundary will be cut off, so in that case it is recommended to set `outer.radius < 1` to reserve headroom inside the panel.

`inner.radius` Numeric scalar in [0, 1) specifying the inner radius of the spider coordinate system as a fraction of the panel radius. Values greater than 0 create a "donut" / hollow center. The data radius is linearly rescaled onto [`inner.radius`, `outer.radius`].

`outer.radius` Numeric scalar in (0, 1] specifying the maximum radius used for drawing the spider geometry as a fraction of the panel radius. Setting `outer.radius < 1` shrinks the spider grid and data polygon inward, thereby reserving whitespace inside the panel for labels while keeping `clip = "on"`. This is the preferred way to create consistent headroom independent of plot margins and device size.

`axis_labels` Logical; if `TRUE`, draw the theta-axis labels (typically the discrete category labels) at the ends of the spokes using `grid::textGrob` in the panel background. Labels are taken from the trained discrete limits of the theta scale.

`axis_label_size`
Numeric scalar giving the font size (in points) for theta-axis labels drawn when `axis_labels = TRUE`.

`axis_label_pad_mm`
Numeric scalar giving the outward padding (in mm) applied to theta-axis labels beyond the spoke endpoints. This padding is applied in npc coordinates via `grid::unit()`. Increase this to separate labels from the grid. When `clip = "on"`, excessive padding may push labels outside the panel and cause clipping; prefer reducing padding or decreasing outer.radius‘ in that case.

r_labels — Logical; if TRUE, draw radial (r / y-scale) labels along the "upper" spoke (the spoke at angle start, i.e. 12 o'clock when start = 0). Breaks and labels are taken from the trained radial scale (typically the y scale when theta = "x"). This means user-supplied scale_y_continuous(breaks = ...) will control both the grid rings and the printed radial labels.

r_label_size — Numeric scalar giving the font size (in points) for radial labels drawn when r_labels = TRUE.

r_label_pad_mm — Numeric scalar giving the horizontal padding (in mm) applied to radial labels from the upper spoke. Use positive values to move labels to the right of the spoke. With clip = "on", large padding can cause clipping; consider smaller padding and/or smaller outer.radius.

Value

A ggproto object that inherits from ggplot2::Coord. This object is used by ggplot2 to transform data coordinates and to render the panel background (grid rings/spokes and optional axis labels).

Examples

```
# Example data: one observation across five metrics.
# 'metric' is the discrete theta axis, 'value' is the radial axis.
library(ggplot2)

df <- data.frame(
  metric = c("Final exam grade",
             "Midterm grade",
             "Assignments completed",
             "Hours studying",
             "Classes attended"),
  value = c(70, 60, 10, 15, 25))
df

ggplot(df, aes(metric, value, group = 1)) +
    # Spider/Radar plots only require geom_polygon() with coord_spider():
    geom_polygon(colour = "red", fill = NA, linewidth = 1) +
    coord_spider()

ggplot(df, aes(metric, value, group = 1)) +
    geom_polygon(colour = "red", fill = NA, linewidth = 1) +
    geom_point(colour = "red", size = 3) +
    coord_spider(inner.radius = 0.25,
                 r_label_pad_mm = 2) +
    scale_y_continuous(limits = c(0, 100),
                       breaks = c(0, 25, 50, 75, 100)) +
    theme(panel.grid.major.y = element_line(colour = "blue"))

df |>
  plot2(type = "spider")
```

create_interactively *Interactively Create a* plot2

Description

This Shiny app allows for interactive creation of a plot2.

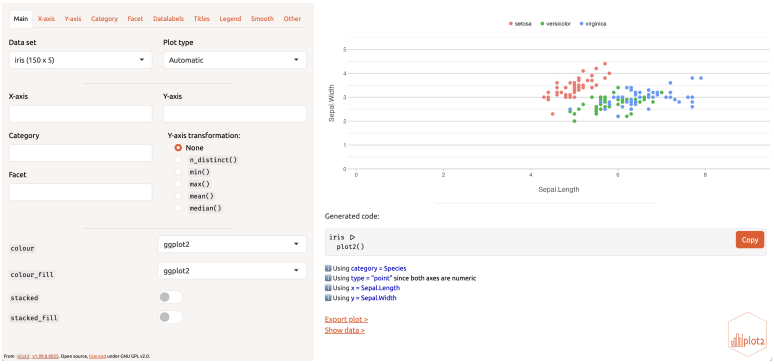
Usage

```
create_interactively(
  data = NULL,
  css_code = NULL,
  logo_path = system.file("logo.svg", package = "plot2"),
  pretty_labels = FALSE,
  hide_generated_code = FALSE,
  hide_export_buttons = NULL,
  upload_tab = FALSE
)
```

Arguments

data	A data set to load. Not strictly required, since all data sets in the global environment will be shown.
css_code	Additional CSS code to load in the app.
logo_path	Path to the logo shown in the app. Default to the plot2 logo. Use NULL to not use a logo.
pretty_labels	A logical to switch to pretty, readable labels, instead of argument names in code style.
hide_generated_code	A logical to hide generated code.
hide_export_buttons	A logical to hide export buttons and functionality. TRUE will hide the elements completely, NULL will show a clickable text to expand buttons (default), FALSE will show the expanded buttons.
upload_tab	A logical to show a dedicated **Upload** tab in the sidebar. When TRUE, a full-featured data-import tab is added with format-specific options for many file types (CSV, TSV, Excel, SPSS, Stata, SAS, RDS, JSON, Parquet, Feather, YAML, XML, and more). The "Upload data set..." dropdown item will redirect to this tab instead of opening a modal. Requires the readxl package in addition to the packages checked for upload_tab = FALSE. Administrators can ensure all dependencies are present by running plot2:::install_shiny_deps() before launching the app.

Details



Examples

```
## Not run:

create_interactively()

iris |> create_interactively()

# With the upload tab for importing external data files:
create_interactively(upload_tab = TRUE)

## End(Not run)
```

`dec_mark` *Use Decimal Comma?*

Description

These functions determine which characters the decimal mark and big mark should be that are used in plotting. They base the determination on `getOption("OutDec")`, which is also what `base::format()` uses.

Usage

```
dec_mark()

big_mark()
```

Details

If the option `"dec_mark"` is set, that value will be used for `dec_mark()` if it is either a comma or a full stop.

At default, `big_mark()` returns a full stop if `dec_mark()` returns a comma, and a space otherwise. If the option `"big_mark"` is set, that value will be used if it is either a comma (`","`) or a full stop (`"."`) or a space (`" "`) or an empty character (`""`).

Examples

```
# at default, this follows `getOption("OutDec")`:
dec_mark()
# and big_mark() returns a space if dec_mark() returns ".":
big_mark()

# you you can set options to alter behaviour:
options(dec_mark = ",", big_mark = ".")
dec_mark()
big_mark()

options(dec_mark = ",", big_mark = NULL)
dec_mark()
big_mark()

options(big_mark = ",")
dec_mark()
big_mark()

# clean up
options(dec_mark = NULL, big_mark = NULL)
```

`get_colour` *Get (and Register) Colours from* R*, Viridis and More*

Description

Retrieves, expands, and prints colours or palettes from R, `viridisLite`, or user-defined sources, with support for registering and unregistering custom colour sets for (automatic) reuse in plots.

Usage

```
get_colour(x, length = 1, opacity = 0)

register_colour(...)

unregister_colour(regex)

add_white(x, white)
```

Arguments

`x` Colour or colour palette name. Input can be:

- One of the colourblind-safe `viridisLite` palettes:
  - `"viridis"`
  - `"magma"`
  - `"inferno"`

  - "plasma"
  - "cividis"
  - "rocket"
  - "mako"
  - "turbo"
- One of the built-in R palettes (these are from R 4.5.1):
  - "Accent"
  - "Alphabet"
  - "Classic Tableau"
  - "Dark 2"
  - "Okabe-Ito"
  - "Paired"
  - "Pastel 1"
  - "Pastel 2"
  - "Polychrome 36"
  - "R3"
  - "R4"
  - "Set 1"
  - "Set 2"
  - "Set 3"
  - "Tableau 10"
  - "ggplot2"
  - "grayscale"
  - "greyscale"
  - "heatmap"
  - "rainbow"
  - "terrain"
  - "topo"
- One of the 657 built-in colours() in R (even case-insensitive), such as "bisque", "blue1", "chartreuse1", "cornsilk4", "lightcoral"
- One of the pre-registered colours using register_colour()

length	Size of the vector to be returned.
opacity	Amount of opacity (0 = solid, 1 = transparent).
...	Named vectors with known, valid colours. They must be coercible with get_colour().
regex	A regular expression to unregister colours.
white	A number between [0, 1] to add white to x.

Details

Limitless Palette Length:

A palette from R will be expanded where needed, so even get_colour("R4", length = 20) will work, despite "R4" only supporting a maximum of eight colours.

Registering Colours In Another Package:

To register/unregister colours in another package, add something like this to a package file `R/zzz.R`:

```
#' @importFrom plot2 register_colour
.onLoad <- function(...) {
  register_colour(
    my_colour1  = "#007A8A",
    my_colour2  = "#2E8540",
    my_colour3  = "#5B3FA8",
    my_colours  = c("my_colour1", "my_colour2", "my_colour3"))

  # set as default upon package load
  options(plot2.colour = "my_colours",
          plot2.colour_font_secondary = "my_colour1")
}

#' @importFrom plot2 unregister_colour
.onUnload <- function(...) {
  unregister_colour("^my_colour") # this is a regular expression
  options(plot2.colour = NULL,
          plot2.colour_font_secondary = NULL)
}
```

Do not forget to add `plot2` to `Imports:` in the `DESCRIPTION` file of your package.

Value

character vector in HTML format (i.e., `"#AABBCC"`) with new class `colour`

Examples

```
get_colour(c("red", "tan1", "#ffa", "FFAA00"))

par(mar = c(0.5, 2.5, 1.5, 0)) # set plot margins for below plots

# all colourblind-safe colour palettes from the famous viridisLite package
barplot(1:7,
        col = get_colour("viridis", 7))
barplot(1:7,
        col = get_colour("magma", 7))

barplot(8:1,
        col = get_colour("R4", 8),
        main = "Some palettes have only 8 colours...")
barplot(20:1,
        col = get_colour("R4", 20),
        main = "Not anymore!")


# Registering Colours ---------------------------------------------------
```

```
# to register colours, use named input - the values will be evaluated
# with get_colour()
get_colour("red123")
register_colour(red123 = "red", red456 = "#ff0000", red789 = "f00")
get_colour("red123")
get_colour("red456")
get_colour("red789")

# you can also register a group name
register_colour(red_group = c("red123", "ff4400", "red3", "red4"))
get_colour("red_group")
get_colour("red_group", 3)

# Registering colours is ideal for your (organisational) style in plots.
# Let's say these are your style:
register_colour(navy_blue = "#1F3A93",
                burnt_orange = "#D35400",
                forest_green = "#2C6F47",
                goldenrod_yellow = "#DAA520",
                slate_grey = "#708090",
                plum_purple = "#8E4585")

# Then register the whole colour list too:
register_colour(my_organisation = c("navy_blue", "burnt_orange",
                                    "forest_green", "goldenrod_yellow",
                                    "slate_grey", "plum_purple"))
# Check that it works:
get_colour("my_organisation", length = 6)

# Now use it in plots as you like:
iris |>
  plot2(x = Species, y = where(is.double), colour = "my_organisation")

# Or even set the option to use it in any future plot:
options(plot2.colour = "my_organisation")

iris |>
  plot2(x = Species, y = where(is.double))

# reset option again
options(plot2.colour = NULL)


# Use add_white() to add white to existing colours:
colours <- get_colour("R4", 6)
colours
add_white(colours, 0.25)
add_white(colours, 0.5)
add_white(colours, 0.75)

add_white("red", 0)     # 100% red
add_white("red", 1/256)
add_white("red", 1/128)
```

```
add_white("red", 1/64)
add_white("red", 1/32)
add_white("red", 1/16)
add_white("red", 1/8)
add_white("red", 1/4)
add_white("red", 1/2)
add_white("red", 1)     # 100% white
```

---

get_plot_title *Get Plot Title*

Description

Get the title of the plot, or a default value. If the title is not set in a plot, this function tries to generate one from the plot mapping.

Usage

```
get_plot_title(plot, valid_filename = TRUE, default = NULL)
```

Arguments

plot	A ggplot2 plot.
valid_filename	A logical to indicate whether the returned value should be a valid filename, defaults to TRUE.
default	The default value, if a plot title is absent.

Examples

```
without_title <- plot2(mtcars)
with_title <- plot2(mtcars, title = "Plotting **mpg** vs. **cyl**!")

# default is a guess:
get_plot_title(without_title)
get_plot_title(without_title, valid_filename = FALSE)
get_plot_title(with_title)
get_plot_title(with_title, valid_filename = FALSE)

# unless 'default' is set (only affects plots without title):
get_plot_title(without_title, default = "title")
get_plot_title(with_title, default = "title")
```

labellers — *Label Euro currencies*

Description

Format numbers as currency, rounding values to dollars or cents using a convenient heuristic.

Usage

```
euros(x, big.mark = big_mark(), decimal.mark = dec_mark(), ...)

dollars(x, big.mark = big_mark(), decimal.mark = dec_mark(), ...)
```

Arguments

x	Values.
big.mark	Thousands separator, defaults to big_mark().
decimal.mark	Decimal mark, defaults to dec_mark().
...	Any argument to give to the geom. This will override automatically-set settings for the geom.

Examples

```
## Not run:
profit <- data.frame(group = LETTERS[1:4],
                     profit = runif(4, 10000, 25000))

profit |>
  plot2(y.labels = euros,
        datalabels = FALSE)

profit |>
  plot2(y.labels = euros,
        datalabels.format = euros)

## End(Not run)
```

md_to_expression — *Convert Markdown to Plotmath Expression*

Description

This function converts common markdown language to an R plotmath expression. plot2() uses this function internally to convert plot titles and axis titles.

Usage

```
md_to_expression(x)
```

Arguments

x	Text to convert, only the first value will be evaluated.

Details

This function only supports common markdown (italic, bold, bold-italic, subscript, superscript), but also supports some additional functionalities for more advanced expressions using R plotmath. Please see *Examples*.

In `plot2()`, this function can be also set to argument `category.labels` to print the data values as expressions:

- `plot2(..., category.labels = md_to_expression)`

Value

An expression if x is length 1, or a list of expressions otherwise

Examples

```
# use '*' for italics, not '_', to prevent conflicts with variable naming
md_to_expression("this is *italic* text, this is _not italic_ text")

md_to_expression("this is **bold** text")

md_to_expression("this is ***bold and italic*** text")

# subscript and superscript can be done in HTML or markdown with curly brackets:
md_to_expression("this is some<sub>subscripted text</sub>, this is also_{subscripted} text")
md_to_expression("this is some<sup>superscripted text</sup>, this is also^{superscripted} text")

# use $...$ to use any plotmath expression as-is (see ?plotmath):
md_to_expression("text $omega$ text, $a[x]$")

mtcars |>
  plot2(mpg, hp,
        title = "*These are* the **Greek** lower $omega$ and upper $Omega$",
        x.title = "x_{mpg}",
        y.title = "y_{hp}")

mtcars |>
  plot2(mpg, hp,
      title = "$f[X](x)==frac(1, sigma*sqrt(2*pi))*plain(e)^{frac(-(x-mu)^2, 2*sigma^2)}$",
        subtitle = "Some insane $widehat(plotmath)$ title")
```

move_layer — *Move a* `ggplot` *Layer*

Description

Use this function to move a certain plot layer up or down. This function returns a `ggplot` object.

Usage

```
move_layer(plot, move = -1, layer = length(plot$layers))
```

Arguments

`plot`	A `ggplot` object.
`move`	Number of layers to move `layer` up or down.
`layer`	The layer to affect, defaults to top layer.

netherlands — *Example Geography Data Set: the Netherlands*

Description

A data set containing the geometies of the twelve provinces of the Netherlands, according to Statistics Netherlands (2021).

Usage

```
netherlands
```

Format

A data.frame with 12 observations and 3 variables:

- `province`
  name of the Dutch province
- `area_km2`
  area in square kilometres
- `geometry`
  geometry of the province, of class sfc_MULTIPOLYGON/sfc

plot2 *Conveniently Create a New* ggplot

Description

The plot2() function is a convenient wrapper around many ggplot2 functions. By design, the ggplot2 package requires users to use a lot of functions and manual settings, while the plot2() function does all the heavy lifting automatically and only requires users to define some arguments in one single function, greatly increases convenience.

Moreover, plot2() allows for in-place calculation of y, all axes, and all axis labels, often preventing the need to use group_by(), count(), mutate(), or summarise().

See plot2-methods for all implemented methods for different object classes.

Usage

```
plot2(
  .data,
  x = NULL,
  y = NULL,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = TRUE,
  y.title = TRUE,
  category.title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
  facet.fill = NULL,
  facet.bold = TRUE,
  facet.italic = FALSE,
  facet.size = 10,
  facet.margin = 8,
  facet.repeat_lbls_x = TRUE,
  facet.repeat_lbls_y = NULL,
  facet.fixed_y = NULL,
  facet.fixed_x = NULL,
  facet.drop = FALSE,
```

```
facet.nrow = NULL,
facet.relative = FALSE,
x.date_breaks = NULL,
x.date_labels = NULL,
x.date_remove_years = NULL,
category.focus = NULL,
colour = getOption("plot2.colour", "ggplot2"),
colour_fill = NULL,
colour_opacity = 0,
x.lbl_angle = 0,
x.lbl_align = NULL,
x.lbl_italic = FALSE,
x.lbl_taxonomy = FALSE,
x.remove = FALSE,
x.position = "bottom",
x.max_items = Inf,
x.max_txt = "(rest, x%n)",
category.max_items = Inf,
category.max_txt = "(rest, x%n)",
facet.max_items = Inf,
facet.max_txt = "(rest, x%n)",
x.breaks = NULL,
x.n_breaks = NULL,
x.transform = "identity",
x.expand = NULL,
x.limits = NULL,
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = NULL,
y.transform = "identity",
y.position = "left",
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
```

```
y_secondary.colour = colour,
y_secondary.colour_fill = colour_fill,
y_secondary.scientific = NULL,
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
category.character = NULL,
x.sort = NULL,
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = "%n",
datalabels.colour = "grey25",
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
smooth.method = NULL,
smooth.formula = NULL,
smooth.se = TRUE,
smooth.level = 0.95,
smooth.alpha = NULL,
smooth.linewidth = NULL,
smooth.linetype = NULL,
smooth.colour = NULL,
size = NULL,
linetype = 1,
```

```
  linewidth = NULL,
  binwidth = NULL,
  width = NULL,
  jitter_seed = NA,
  violin_scale = "count",
  legend.position = NULL,
  legend.title = NULL,
  legend.reverse = FALSE,
  legend.nrow = NULL,
  legend.barheight = 6,
  legend.barwidth = 1.5,
  legend.nbin = 300,
  legend.italic = FALSE,
  sankey.node_width = 0.15,
  sankey.node_whitespace = 0.03,
  sankey.alpha = 0.5,
  sankey.remove_axes = NULL,
  zoom = FALSE,
  sep = " / ",
  print = FALSE,
  text_factor = getOption("plot2.text_factor", 1),
  font = getOption("plot2.font"),
  theme = getOption("plot2.theme", "theme_minimal2"),
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  data = NULL,
  ...
)
```

Arguments

`.data` Data to plot.

`x` Plotting 'direction' for the X-axis. This can be:

- A single variable from `.data`, such as `x = column1`
- A function to calculate over one or more variables from `.data`, such as `x = format(column1, "%Y")`, or `x = ifelse(column1 == "A", "Group A", "Other")`
- Multiple variables from `.data`, such as `x = c(column1, column2, column2)`, or using selection helpers such as `x = where(is.character)` or `x = starts_with("var_")` *(only allowed and required for Sankey plots using* `type = "sankey"`*)*

`y` Values to use for plotting along the Y-axis. This can be:

- A single variable from `.data`, such as `y = column1`
- Multiple variables from `.data`, such as `y = c(column1, column2)` or `y = c(name1 = column1, "name 2" = column2)`, or using selection helpers such as `y = where(is.double)` or `y = starts_with("var_")` *(multiple variables only allowed if* `category` *is not set)*

- A function to calculate over `.data` returning a single value, such as `y = n()` for the row count, or based on other variables such as `y = n_distinct(person_id)`, `y = max(column1)`, or `y = median(column2) / column3`
- A function to calculate over `.data` returning multiple values, such as `y = quantile(column1, c(0.25, 0.75))` or `y = range(age)` *(multiple values only allowed if* `category` *is not set)*

`category, facet` Plotting 'direction' (`category` is called 'fill' and 'colour' in `ggplot2`). This can be:

- A single variable from `.data`, such as `category = column1`
- A function to calculate over one or more variables from `.data`, such as `category = median(column2) / column3`, or `facet = ifelse(column1 == "A", "Group A", "Other")`
- Multiple variables from `.data`, such as `facet = c(column1, column2)` (use `sep` to control the separator character)
- One or more variables from `.data` using selection helpers, such as `category = where(is.double)` or `facet = starts_with("var_")`

The `category` can also be a date or date/time (class `Date` or `POSIXt`).

`type` Type of visualisation to use. This can be:

- A `ggplot2` geom name or their abbreviation such as `"col"` and `"point"`. All geoms are supported (including `geom_blank()`).
  Full function names can be used (e.g., `"geom_histogram"`), but they can also be abbreviated (e.g., `"h"`, `"hist"`). The following geoms can be abbreviated by their first character: area (`"a"`), boxplot (`"b"`), beeswarm (`"bs"`), column (`"c"`), histogram (`"h"`), jitter (`"j"`), line (`"l"`), point (`"p"`), ribbon (`"r"`), and violin (`"v"`).
  Please note: in `ggplot2`, 'bars' and 'columns' are equal, while it is common to many people that 'bars' are oriented horizontally and 'columns' are oriented vertically since Microsoft Excel has been using these terms this way for many years. For this reason, `type = "bar"` will set `type = "col"` and `horizontal = TRUE`.
- One of these additional types:
  - `"barpercent"` (short: `"bp"`), which is effectively a shortcut to set `type = "col"` and `horizontal = TRUE` and `x.max_items = 10` and `x.sort = "freq-desc"` and `datalabels.format = "%n (%p)"`.
  - `"dumbbell"` (short: `"d"`), which sets `type = "point"` and `horizontal = TRUE`, and adds a line between the points (using `geom_segment()`). The line colour cannot be changed. This plot type is only possible when the `category` has two distinct values.
  - `"linedot"` (short: `"ld"`), which sets `type = "line"` and adds two point geoms using `add_point()`; one with large white dots and one with smaller dots using the colours set in `colour`. This is essentially equal to base R `plot(..., type = "b")` but with closed shapes.
  - `"back-to-back"` (short: `"b2b"`) creates a back-to-back plot, sometimes called a Tornado diagram, a Butterfly plot, or a Population Pyramid. It uses `facet` to distinquish the left and right plots. Therefore, `facet` must be set to (a column containing) two unique values.

- "upset" or "UpSet" (short: "u") creates an UpSet plot, which requires x to contain multiple variables from .data that contain 0/1 or FALSE/TRUE values. It is also possible to provide y, on which then summarise_function will be applied.
- "sankey" (short: "s") creates a Sankey plot using category for the flows and requires x to contain multiple variables from .data. At default, it also sets x.expand = c(0.05, 0.05) and y.limits = c(NA, NA) and y.expand = c(0.01, 0.01). The so-called 'nodes' (the 'blocks' with text) are considered the datalabels, so you can set the text size and colour of the nodes using datalabels.size, datalabels.colour, and datalabels.colour_fill. The transparency of the flows can be set using sankey.alpha, and the width of the nodes can be set using sankey.node_width. Sankey plots can also be flipped using horizontal = TRUE, and the label angles can be set with datalabels.angle.
- "spider" or "radar" (short: "sp") creates a Spider plot using our novel CoordSpider coordinate system.

- Left blank. In this case, the type will be determined automatically: "boxplot" if there is no X-axis or if the length of unique values per X-axis item is at least 3, "point" if both the y and x axes are numeric, and the option "plot2.default_type" otherwise (which defaults to "col"). Use type = "blank" or type = "geom_blank" to *not* add a geom.

title, subtitle, caption, tag, x.title, y.title, category.title, legend.title, y_secondary.title

A title to use. This can be:

- A character, which supports markdown by using md_to_expression() internally if markdown = TRUE (which is the default)
- A function to calculate over .data, such as title = paste("Based on n =", n_distinct(person_id), "individuals") or subtitle = paste("Total rows:", n()), see *Examples*
- An expression, e.g. using parse(text = "...")

The category.title defaults to TRUE if the legend items are numeric.

title.linelength

Maximum number of characters per line in the title, before a linebreak occurs.

title.colour Text colour of the title.

subtitle.linelength

Maximum number of characters per line in the subtitle, before a linebreak occurs.

subtitle.colour

Text colour of the subtitle.

na.replace Character to put in place of NA values if na.rm = FALSE.

na.rm Remove NA values from showing in the plot.

facet.position, facet.fill, facet.bold, facet.italic, facet.size, facet.margin, facet.repeat_lbls_x, facet.repeat_lbls_y, facet.drop, facet.nrow, facet.relative

Additional settings for the plotting direction facet.

`facet.fixed_y` A logical to indicate whether all y scales should have the same limits. Defaults to `TRUE` only if the coefficient of variation (standard deviation divided by mean) of the maximum values of y is less than 25%.

`facet.fixed_x` A logical to indicate whether all x scales should have the same breaks. This acts like the inverse of `x.drop`.

`x.date_breaks` Breaks to use when the X-axis contains dates, will be determined automatically if left blank. This accepts values such as `"1 day"` and `"2 years"`.

`x.date_labels` Labels to use when the X-axis contains dates, will be determined automatically if left blank. This accepts 'Excel' date-language such as `"d mmmm yyyy"`.

`x.date_remove_years`
A logical to indicate whether the years of all `x` values must be unified. This will set the years of all `x` values to 1970 if the data does not contain a leap year, and to 1972 otherwise. This allows to plot years on the `category` while maintaining a date range on `x`. The default is `FALSE`, unless `category` contains all years present in `x`.

`category.focus` A value of `category` that should be highlighted, meaning that all other values in `category` will be greyed out. This can also be a numeric value between 1 and the length of unique values of `category`, e.g. `category.focus = 2` to focus on the second legend item.

`colour` Get_colour(s) to set, will be evaluated with `get_colour()` if set. This can also be one of the viridis colours with automatic implementation for any plot: `"viridis"`, `"magma"`, `"inferno"`, `"plasma"`, `"cividis"`, `"rocket"`, `"mako"` or `"turbo"`. Also, this can be a named vector to match values of `category`, see *Examples*. Using a named vector can be used to manually sort the values of `category`.

`colour_fill` Get_colour(s) to be used for filling, will be determined automatically if left blank and will be evaluated with `get_colour()`.

`colour_opacity` Amount of opacity for `colour`/`colour_fill` (0 = solid, 1 = transparent).

`x.lbl_angle` Angle to use for the X-axis in a counter-clockwise direction (i.e., a value of `90` will orient the axis labels from bottom to top, a value of `270` will orient the axis labels from top to bottom).

`x.lbl_align` Alignment for the X-axis between `0` (left aligned) and `1` (right aligned).

`x.lbl_italic` logical to indicate whether the x labels should in in *italics*.

`x.lbl_taxonomy` A logical to transform all words of the `x` labels into italics that are in the microorganisms data set of the `AMR` package. This uses `md_to_expression()` internally and will set `x.labels` to parse expressions.

`x.remove`, `y.remove`
A logical to indicate whether the axis labels and title should be removed.

`x.position`, `y.position`
Position of the axis.

`x.max_items`, `category.max_items`, `facet.max_items`
Number of maximum items to use, defaults to infinite. All other values will be grouped and summarised using the `summarise_function` function. **Please note:** the sorting will be applied first, allowing to e.g. plot the top *n* most frequent values of the X-axis by combining `x.sort = "freq-desc"` with `x.max_items =` *n*.

`x.max_txt`, `category.max_txt`, `facet.max_txt`
: The text to use of values not included number of `*.max_items`. The placeholder `%n` will be replaced with the outcome of the `summarise_function` function, the placeholder `%p` will be replaced with the percentage.

`x.breaks`, `y.breaks`
: A breaks function or numeric vector to use for the axis.

`x.n_breaks`, `y.n_breaks`
: Number of breaks, only useful if `x.breaks` cq. `y.breaks` is `NULL`.

`x.transform`, `y.transform`, `category.transform`
: A transformation function to use, e.g. `"log2"`. This can be: `"asinh"`, `"asn"`, `"atanh"`, `"boxcox"`, `"compose"`, `"date"`, `"exp"`, `"hms"`, `"identity"`, `"log"`, `"log10"`, `"log1p"`, `"log2"`, `"logit"`, `"modulus"`, `"probability"`, `"probit"`, `"pseudo_log"`, `"reciprocal"`, `"reverse"`, `"sqrt"`, `"time"`, `"timespan"`, `"yj"`.

`x.expand`, `y.expand`
: expansion to use for the axis, can be length 1 or 2. `x.expand` defaults to 0.5 and `y.expand` defaults to `0.25`, except for sf objects (then both default to 0).

`x.limits`, `y.limits`
: Limits to use for the axis, can be length 1 or 2. Use `NA` for the highest or lowest value in the data, e.g. `y.limits = c(0, NA)` to have the y scale start at zero.

`x.labels`, `y.labels`, `y_secondary.labels`
: A labels function or character vector to use for the axis.

`x.character`
: A logical to indicate whether the values of the X-axis should be forced to character. The default is `FALSE`, except for years (values between 2000 and 2050) and months (values from 1 to 12).

`x.drop`
: logical to indicate whether factor levels should be dropped.

`x.mic`
: logical to indicate whether the X-axis should be formatted as MIC values, by dropping all factor levels and adding missing factors of 2.

`x.zoom`, `y.zoom`
: A logical to indicate if the axis should be zoomed on the data, by setting `x.limits = c(NA, NA)` and `x.expand = 0` for the X-axis, or `y.limits = c(NA, NA)` and `y.expand = 0` for the Y-axis.

`y.24h`
: a logical to indicate whether the y labels and breaks should be formatted as 24-hour sequences

`y.age`
: A logical to indicate whether the y labels and breaks should be formatted as ages in years.

`y.scientific`, `y_secondary.scientific`
: A logical to indicate whether the y labels should be formatted in scientific notation. Defaults to `TRUE` only if the range of the y values spans more than `10e5`.

`y.percent`, `y_secondary.percent`
: A logical to indicate whether the y labels should be formatted as percentages.

`y.percent_break`
: A value on which the Y-axis should have breaks.

`y_secondary`
: Values to use for plotting along the secondary Y-axis. This functionality is poorly supported by `ggplot2` and might give unexpected results. Setting the secondary Y-axis will set the colour to the axis titles.

y_secondary.type
: see type

y_secondary.colour, y_secondary.colour_fill
: Colours to set for the secondary Y-axis, will be evaluated with get_colour().

category.type
: Type of the category, one or more of: "colour" (default), "shape", "size", "linetype", "linewidth", "alpha". There is no need to set "fill", as plot2 handles colour-setting internally and determines automatically whether the colour or fill aesthetic must be used.

category.labels, category.percent, category.breaks, category.expand, category.midpoint
: Settings for the plotting direction category.

category.limits
: Limits to use for a numeric category, can be length 1 or 2. Use NA for the highest or lowest value in the data, e.g. category.limits = c(0, NA) to have the scale start at zero.

category.date_breaks
: Breaks to use when the category contains dates, will be determined automatically if left blank. This will be passed on to seq.Date(by = ...) and thus can be: a number, taken to be in days, or a character string containing one of "day", "week", "month", "quarter" or "year" (optionally preceded by an integer and a space, and/or followed by "s").

category.date_labels
: Labels to use when the category contains dates, will be determined automatically if left blank. This accepts 'Excel' date-language such as "d mmmm yyyy".

category.character
: A logical to indicate whether the values of the category should be forced to character. The default is FALSE, except for years (values between 2000 and 2050) and months (values from 1 to 12).

x.sort, category.sort, facet.sort
: Sorting of the plotting direction, defaults to TRUE, except for continuous values on the X-axis (such as dates and numbers). Applying one of the sorting methods will transform the values to an ordered factor, which ggplot2 uses to orient the data. Valid values are:

- A manual vector of values
- TRUE: sort factors on their levels, otherwise sort ascending on alphabet, while maintaining numbers in the text (*numeric* sort)
- FALSE: sort according to the order in the data
- NULL: do not sort/transform at all
- "asc" or "alpha": sort as TRUE
- "desc": sort factors on their reversed levels, otherwise sort descending on alphabet, while maintaining numbers in the text (*numeric* sort)
- "order" or "inorder": sort as FALSE
- "freq" or "freq-desc": sort descending according to the frequencies of y computed by summarise_function (highest value first)
- "freq-asc": sort ascending according to the frequencies of y computed by summarise_function (lowest value first)

`x.complete`, `category.complete`, `facet.complete`

A value to complete the data. This makes use of `tidyr::full_seq()` and `tidyr::complete()`. For example, using `x.complete = 0` will apply `data |> complete(full_seq(x, ...), fill = list(x = 0))`. Using value `TRUE` (e.g., `x.complete = TRUE`) is identical to using value `0`.

`datalabels`

Values to show as datalabels, see also `datalabels.format`. This can be:

- Left blank. This will default to the values of `y` in column-type plots, or when plotting spatial 'sf' data, the values of the first column. It will print a maximum of 25 labels unless `datalabels = TRUE`.
- `TRUE` or `FALSE` to force or remove datalabels
- A function to calculate over `.data`, such as `datalabels = paste(round(column1), "\n", column2)`

`datalabels.round`

Number of digits to round the datalabels, applies to both `"%n"` and `"%p"` for replacement (see `datalabels.format`).

`datalabels.format`

Format to use for datalabels. This can be a function (such as `euros()`) or a text. For the text, `"%n"` will be replaced by the count number, and `"%p"` will be replaced by the percentage of the total count. Use `datalabels.format = NULL` to *not* transform the datalabels.

`datalabels.colour`, `datalabels.colour_fill`, `datalabels.size`, `datalabels.angle`, `datalabels.lineheight`

Settings for the datalabels.

`decimal.mark`

Decimal mark, defaults to `dec_mark()`.

`big.mark`

Thousands separator, defaults to `big_mark()`.

`summarise_function`

A function to use if the data has to be summarised, see *Examples*. This can also be `NULL`, which will be converted to `function(x) x`.

`stacked`

A logical to indicate that values must be stacked.

`stacked_fill`

A logical to indicate that values must be filled (i.e., stacked to 100%).

`horizontal`

A logical to turn the plot 90 degrees using `coord_flip()`. This option also updates some theme options, so that e.g., `x.lbl_italic` will still apply to the original X-axis.

`reverse`

A logical to reverse the *values* of `category`. Use `legend.reverse` to reverse the *legend* of `category`.

`smooth`

A logical to add a smooth. In histograms, this will add the density count as an overlaying line (default: `TRUE`). In all other cases, a smooth will be added using `geom_smooth()` (default: `FALSE`).

`smooth.method`, `smooth.formula`, `smooth.se`, `smooth.level`, `smooth.alpha`, `smooth.linewidth`, `smooth.linetype`, `smooth.colour`

Settings for `smooth`.

`size`

Size of the geom. Defaults to `2` for geoms point, jitter, and beeswarm; `4` for a UpSet plots (using `type = "upset"`); `5` for a dumbbell plots (using `type = "dumbbell"`); `0.75` otherwise.

`linetype` Linetype of the geom, only suitable for geoms that draw lines. Possible options are:

1. `"solid"` (default)
2. `"dashed"`
3. `"dotted"`
4. `"dotdash"`
5. `"longdash"`
6. `"twodash"`

Both numbers and text are supported, e.g., `linetype = 2` and `linetype = "dashed"` yield the same results.

`linewidth` Linewidth of the geom, only suitable for geoms that draw lines. Defaults to:

- `0.10` for sf
- `0.25` for geoms that are continous and have fills (such as area), except for geoms boxplot and violin
- `1.0` for dumbbell plots (using `type = "dumbbell"`) and UpSet plots (using `type = "upset"`)
- `0.5` otherwise (such as geoms line, boxplot, violin, and histogram)

`binwidth` Width of bins (only useful for `geom = "histogram"`), can be specified as a numeric value or as a function that calculates width from `x`, see `geom_histogram()`. It defaults to approx. `diff(range(x))` divided by 12 to 22 based on the data.

`width` Width of the geom. Defaults to `0.66` for geoms boxplot, violin and jitter, and to `0.5` otherwise.

`jitter_seed` Seed (randomisation factor) to be set when using `type = "jitter"`.

`violin_scale` Scale to be set when using `type = "violin"`, can also be set to `"area"`.

`legend.position`
Position of the legend, must be `"top"`, `"right"`, `"bottom"`, `"left"` or `"none"` (or `NA` or `NULL`), can be abbreviated. Defaults to `"right"` for numeric `category` values and 'sf' plots, and `"top"` otherwise.

`legend.reverse`, `legend.barheight`, `legend.barwidth`, `legend.nbin`, `legend.italic`, `legend.nrow`
Other settings for the legend.

`sankey.node_width`
Width of the vertical nodes in a Sankey plot.

`sankey.node_whitespace`
Whitespace between the nodes in a Sankey plot.

`sankey.alpha` Alpha of the flows in a Sankey plot.

`sankey.remove_axes`
Logical to indicate whether all axes must be removed in a Sankey plot.

`zoom` A logical to indicate if the plot should be scaled to the data, i.e., not having the x and y axes to start at 0. This will set `x.zoom = TRUE` and `y.zoom = TRUE`.

`sep` Separator character to use if multiple columns are given to either of the three directions: `x`, `category` and `facet`, e.g. `facet = c(column1, column2)`.

`print` A logical to indicate if the result should be printed instead of just returned.

text_factor	Text factor to use, which will apply to all texts shown in the plot.
font	Font (family) to use, can be set with `options(plot2.font = "...")`. Can be any installed system font or any of the > 1400 font names from Google Fonts. When using custom fonts in R Markdown, be sure to set the chunk option `fig.showtext = TRUE`, otherwise an informative error will be generated.
theme	A valid `ggplot2` theme to apply, or `NULL` to use the default `theme_grey()`. This argument accepts themes (e.g., `theme_bw()`), functions (e.g., `theme_bw`) and characters themes (e.g., `"theme_bw"`). The default is `theme_minimal2()`, but can be set with `options(plot2.theme = "...")`.
background	The background colour of the entire plot, can also be `NA` to remove it. Will be evaluated with `get_colour()`. Only applies when `theme` is not `NULL`.
markdown	A logical to turn all labels and titles into plotmath expressions, by converting common markdown language using the `md_to_expression()` function (defaults to `TRUE`).
data	Substitute for `.data`, used in formula notation, e.g., `plot2(hp ~ mpg, data = mtcars)`.
...	Any argument to give to the geom. This will override automatically-set settings for the geom.

Details

The `plot2()` function is a convenient wrapper around many `ggplot2` functions such as `ggplot()`, `aes()`, `geom_col()`, `facet_wrap()`, `labs()`, etc., and provides:

- Writing as few lines of codes as possible
- Easy plotting in three 'directions': `x` (the regular X-axis), `category` (replaces 'fill' and 'colour') and `facet`
- Automatic setting of these 'directions' based on the input data
- Setting in-place calculations for all plotting directions and even `y`
- Easy way for sorting data in many ways (such as on alphabet, numeric value, frequency, original data order), by setting a single argument for the 'direction': `x.sort`, `category.sort` and `facet.sort`
- Easy limiting values, e.g. by setting `x.max_items = 5` or `category.max_items = 5`
- Markdown support for any title text, with any theme
- Integrated support for any Google Font and any installed system font
- An extra clean, minimalistic theme with a lot of whitespace (but without unnecessary margins) that is ideal for printing: `theme_minimal2()`
- Some conveniences from Microsoft Excel:
  - The Y-axis starts at 0 if possible
  - The y scale expands at the top to be better able to interpret all data points
  - Date breaks can be written in a human-readable format (such as "d mmm yyyy")
  - Labels with data values can easily be printed and are automatically determined
- Support for any `ggplot2` extension based on `ggplot2::fortify()`

The ggplot2 package in conjunction with the tidyr, forcats and cleaner packages can provide above functionalities, but the goal of the plot2() function is to generalise this into one function. The generic plot2() function currently has 152 arguments, all with a default value. **Less typing, faster coding.**

Value

a ggplot object

Examples

```
options(plot2.colour = NULL, plot2.colour_sf_fill = NULL)

head(iris)

# no variables determined, so plot2() will try for itself -
# the type will be points since the first two variables are numeric
iris |>
  plot2()

# if x and y are set, no additional mapping will be set:
iris |>
  plot2(Sepal.Width, Sepal.Length)
iris |>
  plot2(Species, Sepal.Length)

# the arguments are in this order: x, y, category, facet
iris |>
  plot2(Sepal.Length, Sepal.Width, Petal.Length, Species)

iris |>
  plot2(Sepal.Length, Sepal.Width, Petal.Length, Species,
        colour = "viridis") # set the viridis colours

# set your own colours
iris |>
  plot2(Sepal.Length, Sepal.Width, Petal.Length, Species,
        colour = c("white", "red", "black"))

# y can also be multiple (named) columns
iris |>
  plot2(x = Sepal.Length,
        y = c(Length = Petal.Length, Width = Petal.Width),
        category.title = "Petal property")
iris |>
  # with included selection helpers such as where(), starts_with(), etc.:
  plot2(x = Species, y = where(is.double))

# the category type can be one or more aesthetics
iris |>
  plot2(zoom = TRUE,
        category.type = c("colour", "shape"),
```

```
        size = 3)
iris |>
  plot2(zoom = TRUE,
        category = Petal.Length,
        category.type = c("colour", "size"),
        colour = "viridis")

# easily add a smooth
iris |>
  plot2(zoom = TRUE,
        smooth = TRUE)
iris |>
  plot2(zoom = TRUE,
        smooth = TRUE,
        smooth.method = "lm")

# support for secondary Y-axis
mtcars |>
  plot2(x = mpg,
        y = hp,
        y_secondary = disp ^ 2,
        y_secondary.scientific = TRUE,
        title = "Secondary Y-axis sets colour to the axis titles")


admitted_patients

# the arguments are in this order: x, y, category, facet
admitted_patients |>
  plot2(hospital, age)

admitted_patients |>
  plot2(hospital, age, gender)

admitted_patients |>
  plot2(hospital, age, gender, ward)

# or use any function for y
admitted_patients |>
  plot2(hospital, median(age), gender, ward)
admitted_patients |>
  plot2(hospital, n(), gender, ward)

admitted_patients |>
  plot2(x = hospital,
        y = age,
        category = gender,
        colour = c("F" = "#3F681C", "M" = "#375E97"),
        colour_fill = "#FFBB00AA",
        linewidth = 1.25,
        y.age = TRUE)

admitted_patients |>
```

```
  plot2(age, type = "hist")

# even titles support calculations, including support for {glue}
admitted_patients |>
  plot2(age, type = "hist",
        title = paste("Based on n =", n_distinct(patient_id), "patients"),
        subtitle = paste("Total rows:", n()),
        caption = glue::glue("From {n_distinct(hospital)} hospitals"),
        x.title = paste("Age ranging from", paste(range(age), collapse = " to ")))

# the default type is column, datalabels are automatically
# set in non-continuous types:
admitted_patients |>
  plot2(hospital, n(), gender)

admitted_patients |>
  plot2(hospital, n(), gender,
        stacked = TRUE)

admitted_patients |>
  plot2(hospital, n(), gender,
        stacked_fill = TRUE)

# two categories might benefit from a dumbbell plot:
admitted_patients |>
  plot2(hospital, median(age), gender, type = "dumbbell")

# sort on any direction:
admitted_patients |>
  plot2(hospital, n(), gender,
        x.sort = "freq-asc",
        stacked = TRUE)

admitted_patients |>
  plot2(hospital, n(), gender,
        x.sort = c("B", "D", "A"), # missing values ("C") will be added
        category.sort = "alpha-desc",
        stacked = TRUE)

# support for Sankey plots
Titanic |> # a table from base R
  plot2(x = c(Age, Class, Survived),
        category = Sex,
        type = "sankey")

# support for spider plots
ggplot2::diamonds |>
   plot2(x = cut,
         y = mean(price),
         category = color,
         type = "spider",
         y.labels = dollars)
```

```
# matrix support, such as for cor()
correlation_matrix <- cor(mtcars)
class(correlation_matrix)
head(correlation_matrix)
mtcars |>
  cor() |>
  plot2()

mtcars |>
  cor() |>
  plot2(colour = c("blue3", "white", "red3"),
        datalabels = TRUE,
        category.title = "*r*-value",
        title =  "Correlation matrix")


# plot2() supports all S3 extensions available through
# ggplot2::fortify() and broom::augment(), such as regression models:
lm(mpg ~ hp, data = mtcars) |>
  plot2(x = mpg ^ -3,
        y = hp ^ 2,
        smooth = TRUE,
        smooth.method = "lm",
        smooth.formula = "y ~ log(x)",
        title = "Titles/captions *support* **markdown**",
        subtitle = "Axis titles contain the square notation: x^2")

# sf objects (geographic plots, 'simple features') are also supported
netherlands |>
  plot2()
netherlands |>
  plot2(colour_fill = "viridis", colour_opacity = 0.75) |>
  add_sf(netherlands, colour = "black", colour_fill = NA)

# support for any system or Google font
mtcars |>
  plot2(mpg, hp,
        font = "Rock Salt",
        text_factor = 1.25,
        title = "This plot uses a Google Font")
```

---

plot2-methods *Methods for* plot2()

Description

These are the implemented methods for different S3 classes to be used in plot2(). Since they have an extensive list of arguments, they are placed here on a separate manual page.

Usage

```
## Default S3 method:
plot2(
  .data,
  x = NULL,
  y = NULL,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = TRUE,
  y.title = TRUE,
  category.title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
  facet.fill = NULL,
  facet.bold = TRUE,
  facet.italic = FALSE,
  facet.size = 10,
  facet.margin = 8,
  facet.repeat_lbls_x = TRUE,
  facet.repeat_lbls_y = NULL,
  facet.fixed_y = NULL,
  facet.fixed_x = NULL,
  facet.drop = FALSE,
  facet.nrow = NULL,
  facet.relative = FALSE,
  x.date_breaks = NULL,
  x.date_labels = NULL,
  x.date_remove_years = NULL,
  category.focus = NULL,
  colour = getOption("plot2.colour", "ggplot2"),
  colour_fill = NULL,
  colour_opacity = 0,
  x.lbl_angle = 0,
  x.lbl_align = NULL,
  x.lbl_italic = FALSE,
  x.lbl_taxonomy = FALSE,
  x.remove = FALSE,
  x.position = "bottom",
```

```
x.max_items = Inf,
x.max_txt = "(rest, x%n)",
category.max_items = Inf,
category.max_txt = "(rest, x%n)",
facet.max_items = Inf,
facet.max_txt = "(rest, x%n)",
x.breaks = NULL,
x.n_breaks = NULL,
x.transform = "identity",
x.expand = NULL,
x.limits = NULL,
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = NULL,
y.transform = "identity",
y.position = "left",
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
y_secondary.colour = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.colour_fill = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.scientific = NULL,
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
```

```
category.character = NULL,
x.sort = NULL,
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = "%n",
datalabels.colour = "grey25",
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
smooth.method = NULL,
smooth.formula = NULL,
smooth.se = TRUE,
smooth.level = 0.95,
smooth.alpha = NULL,
smooth.linewidth = NULL,
smooth.linetype = NULL,
smooth.colour = NULL,
size = NULL,
linetype = 1,
linewidth = NULL,
binwidth = NULL,
width = NULL,
jitter_seed = NA,
violin_scale = "count",
legend.position = NULL,
legend.title = NULL,
legend.reverse = FALSE,
legend.nrow = NULL,
legend.barheight = 6,
legend.barwidth = 1.5,
legend.nbin = 300,
legend.italic = FALSE,
sankey.node_width = 0.15,
sankey.node_whitespace = 0.03,
```

```
  sankey.alpha = 0.5,
  sankey.remove_axes = NULL,
  zoom = FALSE,
  sep = " / ",
  print = FALSE,
  text_factor = getOption("plot2.text_factor", 1),
  font = getOption("plot2.font"),
  theme = getOption("plot2.theme", "theme_minimal2"),
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  ...
)

## S3 method for class 'formula'
plot2(
  .data = NULL,
  x = NULL,
  y = NULL,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = TRUE,
  y.title = TRUE,
  category.title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
  facet.fill = NULL,
  facet.bold = TRUE,
  facet.italic = FALSE,
  facet.size = 10,
  facet.margin = 8,
  facet.repeat_lbls_x = TRUE,
  facet.repeat_lbls_y = NULL,
  facet.fixed_y = NULL,
  facet.fixed_x = NULL,
  facet.drop = FALSE,
  facet.nrow = NULL,
  facet.relative = FALSE,
  x.date_breaks = NULL,
```

```
x.date_labels = NULL,
x.date_remove_years = NULL,
category.focus = NULL,
colour = getOption("plot2.colour", "ggplot2"),
colour_fill = NULL,
colour_opacity = 0,
x.lbl_angle = 0,
x.lbl_align = NULL,
x.lbl_italic = FALSE,
x.lbl_taxonomy = FALSE,
x.remove = FALSE,
x.position = "bottom",
x.max_items = Inf,
x.max_txt = "(rest, x%n)",
category.max_items = Inf,
category.max_txt = "(rest, x%n)",
facet.max_items = Inf,
facet.max_txt = "(rest, x%n)",
x.breaks = NULL,
x.n_breaks = NULL,
x.transform = "identity",
x.expand = NULL,
x.limits = NULL,
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = NULL,
y.transform = "identity",
y.position = "left",
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
y_secondary.colour = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.colour_fill = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.scientific = NULL,
```

```
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
category.character = NULL,
x.sort = NULL,
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = "%n",
datalabels.colour = "grey25",
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
smooth.method = NULL,
smooth.formula = NULL,
smooth.se = TRUE,
smooth.level = 0.95,
smooth.alpha = NULL,
smooth.linewidth = NULL,
smooth.linetype = NULL,
smooth.colour = NULL,
size = NULL,
linetype = 1,
linewidth = NULL,
binwidth = NULL,
width = NULL,
```

```
  jitter_seed = NA,
  violin_scale = "count",
  legend.position = NULL,
  legend.title = NULL,
  legend.reverse = FALSE,
  legend.nrow = NULL,
  legend.barheight = 6,
  legend.barwidth = 1.5,
  legend.nbin = 300,
  legend.italic = FALSE,
  sankey.node_width = 0.15,
  sankey.node_whitespace = 0.03,
  sankey.alpha = 0.5,
  sankey.remove_axes = NULL,
  zoom = FALSE,
  sep = " / ",
  print = FALSE,
  text_factor = getOption("plot2.text_factor", 1),
  font = getOption("plot2.font"),
  theme = getOption("plot2.theme", "theme_minimal2"),
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  data = NULL,
  ...
)

## S3 method for class 'freq'
plot2(
  .data,
  x = .data$item,
  y = .data$count,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = "Item",
  y.title = "Count",
  category.title = TRUE,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
```

```
facet.fill = NULL,
facet.bold = TRUE,
facet.italic = FALSE,
facet.size = 10,
facet.margin = 8,
facet.repeat_lbls_x = TRUE,
facet.repeat_lbls_y = NULL,
facet.fixed_y = NULL,
facet.fixed_x = NULL,
facet.drop = FALSE,
facet.nrow = NULL,
facet.relative = FALSE,
x.date_breaks = NULL,
x.date_labels = NULL,
x.date_remove_years = NULL,
category.focus = NULL,
colour = getOption("plot2.colour", "ggplot2"),
colour_fill = NULL,
colour_opacity = 0,
x.lbl_angle = 0,
x.lbl_align = NULL,
x.lbl_italic = FALSE,
x.lbl_taxonomy = FALSE,
x.remove = FALSE,
x.position = "bottom",
x.max_items = Inf,
x.max_txt = "(rest, x%n)",
category.max_items = Inf,
category.max_txt = "(rest, x%n)",
facet.max_items = Inf,
facet.max_txt = "(rest, x%n)",
x.breaks = NULL,
x.n_breaks = NULL,
x.transform = "identity",
x.expand = NULL,
x.limits = NULL,
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
```

```
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = NULL,
y.transform = "identity",
y.position = "left",
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
y_secondary.colour = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.colour_fill = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.scientific = NULL,
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
category.character = NULL,
x.sort = "freq-desc",
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = "%n",
datalabels.colour = "grey25",
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
```

```
  smooth.method = NULL,
  smooth.formula = NULL,
  smooth.se = TRUE,
  smooth.level = 0.95,
  smooth.alpha = NULL,
  smooth.linewidth = NULL,
  smooth.linetype = NULL,
  smooth.colour = NULL,
  size = NULL,
  linetype = 1,
  linewidth = NULL,
  binwidth = NULL,
  width = NULL,
  jitter_seed = NA,
  violin_scale = "count",
  legend.position = NULL,
  legend.title = NULL,
  legend.reverse = FALSE,
  legend.nrow = NULL,
  legend.barheight = 6,
  legend.barwidth = 1.5,
  legend.nbin = 300,
  legend.italic = FALSE,
  sankey.node_width = 0.15,
  sankey.node_whitespace = 0.03,
  sankey.alpha = 0.5,
  sankey.remove_axes = NULL,
  zoom = FALSE,
  sep = " / ",
  print = FALSE,
  text_factor = getOption("plot2.text_factor", 1),
  font = getOption("plot2.font"),
  theme = getOption("plot2.theme", "theme_minimal2"),
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  ...
)

## S3 method for class 'sf'
plot2(
  .data,
  x = NULL,
  y = NULL,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = FALSE,
  y.title = FALSE,
```

```
  category.title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
  facet.fill = NULL,
  facet.bold = TRUE,
  facet.italic = FALSE,
  facet.size = 10,
  facet.margin = 8,
  facet.repeat_lbls_x = TRUE,
  facet.repeat_lbls_y = NULL,
  facet.fixed_y = NULL,
  facet.fixed_x = NULL,
  facet.drop = FALSE,
  facet.nrow = NULL,
  facet.relative = FALSE,
  x.date_breaks = NULL,
  x.date_labels = NULL,
  x.date_remove_years = NULL,
  category.focus = NULL,
  colour = getOption("plot2.colour_sf", "grey50"),
 colour_fill = getOption("plot2.colour_sf_fill", getOption("plot2.colour", "ggplot2")),
  colour_opacity = 0,
  x.lbl_angle = 0,
  x.lbl_align = NULL,
  x.lbl_italic = FALSE,
  x.lbl_taxonomy = FALSE,
  x.remove = FALSE,
  x.position = "bottom",
  x.max_items = Inf,
  x.max_txt = "(rest, x%n)",
  category.max_items = Inf,
  category.max_txt = "(rest, x%n)",
  facet.max_items = Inf,
  facet.max_txt = "(rest, x%n)",
  x.breaks = NULL,
  x.n_breaks = NULL,
  x.transform = "identity",
  x.expand = 0,
  x.limits = NULL,
```

```
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = 0,
y.transform = "identity",
y.position = "left",
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
y_secondary.colour = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.colour_fill = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.scientific = NULL,
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
category.character = NULL,
x.sort = NULL,
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = NULL,
datalabels.colour = "black",
```

```
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
smooth.method = NULL,
smooth.formula = NULL,
smooth.se = TRUE,
smooth.level = 0.95,
smooth.alpha = NULL,
smooth.linewidth = NULL,
smooth.linetype = NULL,
smooth.colour = NULL,
size = NULL,
linetype = 1,
linewidth = NULL,
binwidth = NULL,
width = NULL,
jitter_seed = NA,
violin_scale = "count",
legend.position = "right",
legend.title = NULL,
legend.reverse = FALSE,
legend.nrow = NULL,
legend.barheight = 6,
legend.barwidth = 1.5,
legend.nbin = 300,
legend.italic = FALSE,
sankey.node_width = 0.15,
sankey.node_whitespace = 0.03,
sankey.alpha = 0.5,
sankey.remove_axes = NULL,
zoom = FALSE,
sep = " / ",
print = FALSE,
text_factor = getOption("plot2.text_factor", 1),
font = getOption("plot2.font"),
theme = theme_minimal2(panel.grid.major = element_blank(), panel.grid.minor =
 element_blank(), panel.border = element_blank(), plot.margin = unit(c(5, 5, 0, 0),
 units = "pt"), axis.title = element_blank(), axis.text = element_blank(), axis.line =
  element_blank(), axis.ticks = element_blank()),
```

```
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  data = NULL,
  crs = NULL,
  datalabels.centroid = NULL,
  ...
)

## S3 method for class 'data.frame'
plot2(
  .data,
  x = NULL,
  y = NULL,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = TRUE,
  y.title = TRUE,
  category.title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
  facet.fill = NULL,
  facet.bold = TRUE,
  facet.italic = FALSE,
  facet.size = 10,
  facet.margin = 8,
  facet.repeat_lbls_x = TRUE,
  facet.repeat_lbls_y = NULL,
  facet.fixed_y = NULL,
  facet.fixed_x = NULL,
  facet.drop = FALSE,
  facet.nrow = NULL,
  facet.relative = FALSE,
  x.date_breaks = NULL,
  x.date_labels = NULL,
  x.date_remove_years = NULL,
  category.focus = NULL,
  colour = getOption("plot2.colour", "ggplot2"),
  colour_fill = NULL,
```

```
colour_opacity = 0,
x.lbl_angle = 0,
x.lbl_align = NULL,
x.lbl_italic = FALSE,
x.lbl_taxonomy = FALSE,
x.remove = FALSE,
x.position = "bottom",
x.max_items = Inf,
x.max_txt = "(rest, x%n)",
category.max_items = Inf,
category.max_txt = "(rest, x%n)",
facet.max_items = Inf,
facet.max_txt = "(rest, x%n)",
x.breaks = NULL,
x.n_breaks = NULL,
x.transform = "identity",
x.expand = NULL,
x.limits = NULL,
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = NULL,
y.transform = "identity",
y.position = "left",
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
y_secondary.colour = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.colour_fill = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.scientific = NULL,
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
```

```
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
category.character = NULL,
x.sort = NULL,
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = "%n",
datalabels.colour = "grey25",
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
smooth.method = NULL,
smooth.formula = NULL,
smooth.se = TRUE,
smooth.level = 0.95,
smooth.alpha = NULL,
smooth.linewidth = NULL,
smooth.linetype = NULL,
smooth.colour = NULL,
size = NULL,
linetype = 1,
linewidth = NULL,
binwidth = NULL,
width = NULL,
jitter_seed = NA,
violin_scale = "count",
legend.position = NULL,
legend.title = NULL,
legend.reverse = FALSE,
```

```
  legend.nrow = NULL,
  legend.barheight = 6,
  legend.barwidth = 1.5,
  legend.nbin = 300,
  legend.italic = FALSE,
  sankey.node_width = 0.15,
  sankey.node_whitespace = 0.03,
  sankey.alpha = 0.5,
  sankey.remove_axes = NULL,
  zoom = FALSE,
  sep = " / ",
  print = FALSE,
  text_factor = getOption("plot2.text_factor", 1),
  font = getOption("plot2.font"),
  theme = getOption("plot2.theme", "theme_minimal2"),
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  ...
)

## S3 method for class 'matrix'
plot2(
  .data,
  x = NULL,
  y = NULL,
  category = NULL,
  facet = NULL,
  type = NULL,
  x.title = FALSE,
  y.title = FALSE,
  category.title = NULL,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  title.linelength = 60,
  title.colour = getOption("plot2.colour_font_primary", "black"),
  subtitle.linelength = 60,
  subtitle.colour = getOption("plot2.colour_font_secondary", "grey35"),
  na.replace = "",
  na.rm = FALSE,
  facet.position = "top",
  facet.fill = NULL,
  facet.bold = TRUE,
  facet.italic = FALSE,
  facet.size = 10,
  facet.margin = 8,
  facet.repeat_lbls_x = TRUE,
```

```
facet.repeat_lbls_y = NULL,
facet.fixed_y = NULL,
facet.fixed_x = NULL,
facet.drop = FALSE,
facet.nrow = NULL,
facet.relative = FALSE,
x.date_breaks = NULL,
x.date_labels = NULL,
x.date_remove_years = NULL,
category.focus = NULL,
colour = getOption("plot2.colour", "ggplot2"),
colour_fill = NULL,
colour_opacity = 0,
x.lbl_angle = 0,
x.lbl_align = NULL,
x.lbl_italic = FALSE,
x.lbl_taxonomy = FALSE,
x.remove = FALSE,
x.position = "bottom",
x.max_items = Inf,
x.max_txt = "(rest, x%n)",
category.max_items = Inf,
category.max_txt = "(rest, x%n)",
facet.max_items = Inf,
facet.max_txt = "(rest, x%n)",
x.breaks = NULL,
x.n_breaks = NULL,
x.transform = "identity",
x.expand = NULL,
x.limits = NULL,
x.labels = NULL,
x.character = NULL,
x.drop = FALSE,
x.mic = FALSE,
x.zoom = FALSE,
y.remove = FALSE,
y.24h = FALSE,
y.age = FALSE,
y.scientific = NULL,
y.percent = FALSE,
y.percent_break = 0.1,
y.breaks = NULL,
y.n_breaks = NULL,
y.limits = NULL,
y.labels = NULL,
y.expand = NULL,
y.transform = "identity",
y.position = "left",
```

```
y.zoom = FALSE,
y_secondary = NULL,
y_secondary.type = type,
y_secondary.title = TRUE,
y_secondary.colour = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.colour_fill = get_colour(getOption("plot2.colour", "ggplot2"), 2),
y_secondary.scientific = NULL,
y_secondary.percent = FALSE,
y_secondary.labels = NULL,
category.type = "colour",
category.labels = NULL,
category.percent = FALSE,
category.breaks = NULL,
category.limits = NULL,
category.expand = 0,
category.midpoint = NULL,
category.transform = "identity",
category.date_breaks = NULL,
category.date_labels = NULL,
category.character = NULL,
x.sort = NULL,
category.sort = TRUE,
facet.sort = TRUE,
x.complete = NULL,
category.complete = NULL,
facet.complete = NULL,
datalabels = TRUE,
datalabels.round = ifelse(y.percent, 2, 1),
datalabels.format = "%n",
datalabels.colour = "grey25",
datalabels.colour_fill = NULL,
datalabels.size = (3 * text_factor),
datalabels.angle = 0,
datalabels.lineheight = 1,
decimal.mark = dec_mark(),
big.mark = big_mark(),
summarise_function = base::sum,
stacked = FALSE,
stacked_fill = FALSE,
horizontal = FALSE,
reverse = horizontal,
smooth = NULL,
smooth.method = NULL,
smooth.formula = NULL,
smooth.se = TRUE,
smooth.level = 0.95,
smooth.alpha = NULL,
smooth.linewidth = NULL,
```

```
  smooth.linetype = NULL,
  smooth.colour = NULL,
  size = NULL,
  linetype = 1,
  linewidth = NULL,
  binwidth = NULL,
  width = NULL,
  jitter_seed = NA,
  violin_scale = "count",
  legend.position = NULL,
  legend.title = NULL,
  legend.reverse = FALSE,
  legend.nrow = NULL,
  legend.barheight = 6,
  legend.barwidth = 1.5,
  legend.nbin = 300,
  legend.italic = FALSE,
  sankey.node_width = 0.15,
  sankey.node_whitespace = 0.03,
  sankey.alpha = 0.5,
  sankey.remove_axes = NULL,
  zoom = FALSE,
  sep = " / ",
  print = FALSE,
  text_factor = getOption("plot2.text_factor", 1),
  font = getOption("plot2.font"),
  theme = getOption("plot2.theme", "theme_minimal2"),
  background = getOption("plot2.colour_background", "white"),
  markdown = TRUE,
  ...
)
```

Arguments

`.data`, `data` Data to plot.

`x` Plotting 'direction' for the X-axis. This can be:

- A single variable from `.data`, such as `x = column1`
- A function to calculate over one or more variables from `.data`, such as `x = format(column1, "%Y")`, or `x = ifelse(column1 == "A", "Group A", "Other")`
- Multiple variables from `.data`, such as `x = c(column1, column2, column2)`, or using selection helpers such as `x = where(is.character)` or `x = starts_with("var_")` *(only allowed and required for Sankey plots using* `type = "sankey"`*)*

`y` Values to use for plotting along the Y-axis. This can be:

- A single variable from `.data`, such as `y = column1`
- Multiple variables from `.data`, such as `y = c(column1, column2)` or `y = c(name1 = column1, "name 2" = column2)`, or using selection helpers such

as y = where(is.double) or y = starts_with("var_") *(multiple variables only allowed if* category *is not set)*

- A function to calculate over .data returning a single value, such as y = n() for the row count, or based on other variables such as y = n_distinct(person_id), y = max(column1), or y = median(column2) / column3
- A function to calculate over .data returning multiple values, such as y = quantile(column1, c(0.25, 0.75)) or y = range(age) *(multiple values only allowed if* category *is not set)*

category, facet

Plotting 'direction' (category is called 'fill' and 'colour' in ggplot2). This can be:

- A single variable from .data, such as category = column1
- A function to calculate over one or more variables from .data, such as category = median(column2) / column3, or facet = ifelse(column1 == "A", "Group A", "Other")
- Multiple variables from .data, such as facet = c(column1, column2) (use sep to control the separator character)
- One or more variables from .data using selection helpers, such as category = where(is.double) or facet = starts_with("var_")

The category can also be a date or date/time (class Date or POSIXt).

type

Type of visualisation to use. This can be:

- A ggplot2 geom name or their abbreviation such as "col" and "point". All geoms are supported (including geom_blank()).
  Full function names can be used (e.g., "geom_histogram"), but they can also be abbreviated (e.g., "h", "hist"). The following geoms can be abbreviated by their first character: area ("a"), boxplot ("b"), beeswarm ("bs"), column ("c"), histogram ("h"), jitter ("j"), line ("l"), point ("p"), ribbon ("r"), and violin ("v").
  Please note: in ggplot2, 'bars' and 'columns' are equal, while it is common to many people that 'bars' are oriented horizontally and 'columns' are oriented vertically since Microsoft Excel has been using these terms this way for many years. For this reason, type = "bar" will set type = "col" and horizontal = TRUE.
- One of these additional types:
  - "barpercent" (short: "bp"), which is effectively a shortcut to set type = "col" and horizontal = TRUE and x.max_items = 10 and x.sort = "freq-desc" and datalabels.format = "%n (%p)".
  - "dumbbell" (short: "d"), which sets type = "point" and horizontal = TRUE, and adds a line between the points (using geom_segment()). The line colour cannot be changed. This plot type is only possible when the category has two distinct values.
  - "linedot" (short: "ld"), which sets type = "line" and adds two point geoms using add_point(); one with large white dots and one with smaller dots using the colours set in colour. This is essentially equal to base R plot(..., type = "b") but with closed shapes.

- "back-to-back" (short: "b2b") creates a back-to-back plot, sometimes called a Tornado diagram, a Butterfly plot, or a Population Pyramid. It uses facet to distinquish the left and right plots. Therefore, facet must be set to (a column containing) two unique values.
- "upset" or "UpSet" (short: "u") creates an UpSet plot, which requires x to contain multiple variables from .data that contain 0/1 or FALSE/TRUE values. It is also possible to provide y, on which then summarise_function will be applied.
- "sankey" (short: "s") creates a Sankey plot using category for the flows and requires x to contain multiple variables from .data. At default, it also sets x.expand = c(0.05, 0.05) and y.limits = c(NA, NA) and y.expand = c(0.01, 0.01). The so-called 'nodes' (the 'blocks' with text) are considered the datalabels, so you can set the text size and colour of the nodes using datalabels.size, datalabels.colour, and datalabels.colour_fill. The transparency of the flows can be set using sankey.alpha, and the width of the nodes can be set using sankey.node_width. Sankey plots can also be flipped using horizontal = TRUE, and the label angles can be set with datalabels.angle.
- "spider" or "radar" (short: "sp") creates a Spider plot using our novel CoordSpider coordinate system.

- Left blank. In this case, the type will be determined automatically: "boxplot" if there is no X-axis or if the length of unique values per X-axis item is at least 3, "point" if both the y and x axes are numeric, and the option "plot2.default_type" otherwise (which defaults to "col"). Use type = "blank" or type = "geom_blank" to *not* add a geom.

title, subtitle, caption, tag, x.title, y.title, category.title, legend.title, y_secondary.title

A title to use. This can be:

- A character, which supports markdown by using md_to_expression() internally if markdown = TRUE (which is the default)
- A function to calculate over .data, such as title = paste("Based on n =", n_distinct(person_id), "individuals") or subtitle = paste("Total rows:", n()), see *Examples*
- An expression, e.g. using parse(text = "...")

The category.title defaults to TRUE if the legend items are numeric.

title.linelength

Maximum number of characters per line in the title, before a linebreak occurs.

title.colour Text colour of the title.

subtitle.linelength

Maximum number of characters per line in the subtitle, before a linebreak occurs.

subtitle.colour

Text colour of the subtitle.

na.replace Character to put in place of NA values if na.rm = FALSE.

na.rm Remove NA values from showing in the plot.

`facet.position`, `facet.fill`, `facet.bold`, `facet.italic`, `facet.size`, `facet.margin`, `facet.repeat_lbls_x`, `facet.repeat_lbls_y`, `facet.drop`, `facet.nrow`, `facet.relative`
: Additional settings for the plotting direction `facet`.

`facet.fixed_y`
: A logical to indicate whether all y scales should have the same limits. Defaults to `TRUE` only if the coefficient of variation (standard deviation divided by mean) of the maximum values of y is less than 25%.

`facet.fixed_x`
: A logical to indicate whether all x scales should have the same breaks. This acts like the inverse of `x.drop`.

`x.date_breaks`
: Breaks to use when the X-axis contains dates, will be determined automatically if left blank. This accepts values such as `"1 day"` and `"2 years"`.

`x.date_labels`
: Labels to use when the X-axis contains dates, will be determined automatically if left blank. This accepts 'Excel' date-language such as `"d mmmm yyyy"`.

`x.date_remove_years`
: A logical to indicate whether the years of all `x` values must be unified. This will set the years of all `x` values to 1970 if the data does not contain a leap year, and to 1972 otherwise. This allows to plot years on the `category` while maintaining a date range on `x`. The default is `FALSE`, unless `category` contains all years present in `x`.

`category.focus`
: A value of `category` that should be highlighted, meaning that all other values in `category` will be greyed out. This can also be a numeric value between 1 and the length of unique values of `category`, e.g. `category.focus = 2` to focus on the second legend item.

`colour`
: Get_colour(s) to set, will be evaluated with `get_colour()` if set. This can also be one of the viridis colours with automatic implementation for any plot: `"viridis"`, `"magma"`, `"inferno"`, `"plasma"`, `"cividis"`, `"rocket"`, `"mako"` or `"turbo"`. Also, this can be a named vector to match values of `category`, see *Examples*. Using a named vector can be used to manually sort the values of `category`.

`colour_fill`
: Get_colour(s) to be used for filling, will be determined automatically if left blank and will be evaluated with `get_colour()`.

`colour_opacity`
: Amount of opacity for `colour`/`colour_fill` (0 = solid, 1 = transparent).

`x.lbl_angle`
: Angle to use for the X-axis in a counter-clockwise direction (i.e., a value of `90` will orient the axis labels from bottom to top, a value of `270` will orient the axis labels from top to bottom).

`x.lbl_align`
: Alignment for the X-axis between `0` (left aligned) and `1` (right aligned).

`x.lbl_italic`
: logical to indicate whether the x labels should in in *italics*.

`x.lbl_taxonomy`
: A logical to transform all words of the `x` labels into italics that are in the microorganisms data set of the `AMR` package. This uses `md_to_expression()` internally and will set `x.labels` to parse expressions.

`x.remove`, `y.remove`
: A logical to indicate whether the axis labels and title should be removed.

`x.position`, `y.position`
: Position of the axis.

`x.max_items`, `category.max_items`, `facet.max_items`
: Number of maximum items to use, defaults to infinite. All other values will be grouped and summarised using the `summarise_function` function. **Please note:** the sorting will be applied first, allowing to e.g. plot the top *n* most frequent values of the X-axis by combining `x.sort = "freq-desc"` with `x.max_items =` *n*.

`x.max_txt`, `category.max_txt`, `facet.max_txt`
: The text to use of values not included number of `*.max_items`. The placeholder `%n` will be replaced with the outcome of the `summarise_function` function, the placeholder `%p` will be replaced with the percentage.

`x.breaks`, `y.breaks`
: A breaks function or numeric vector to use for the axis.

`x.n_breaks`, `y.n_breaks`
: Number of breaks, only useful if `x.breaks` cq. `y.breaks` is `NULL`.

`x.transform`, `y.transform`, `category.transform`
: A transformation function to use, e.g. `"log2"`. This can be: `"asinh"`, `"asn"`, `"atanh"`, `"boxcox"`, `"compose"`, `"date"`, `"exp"`, `"hms"`, `"identity"`, `"log"`, `"log10"`, `"log1p"`, `"log2"`, `"logit"`, `"modulus"`, `"probability"`, `"probit"`, `"pseudo_log"`, `"reciprocal"`, `"reverse"`, `"sqrt"`, `"time"`, `"timespan"`, `"yj"`.

`x.expand`, `y.expand`
: expansion to use for the axis, can be length 1 or 2. `x.expand` defaults to 0.5 and `y.expand` defaults to `0.25`, except for sf objects (then both default to 0).

`x.limits`, `y.limits`
: Limits to use for the axis, can be length 1 or 2. Use `NA` for the highest or lowest value in the data, e.g. `y.limits = c(0, NA)` to have the y scale start at zero.

`x.labels`, `y.labels`, `y_secondary.labels`
: A labels function or character vector to use for the axis.

`x.character`
: A logical to indicate whether the values of the X-axis should be forced to character. The default is `FALSE`, except for years (values between 2000 and 2050) and months (values from 1 to 12).

`x.drop`
: logical to indicate whether factor levels should be dropped.

`x.mic`
: logical to indicate whether the X-axis should be formatted as MIC values, by dropping all factor levels and adding missing factors of 2.

`x.zoom`, `y.zoom`
: A logical to indicate if the axis should be zoomed on the data, by setting `x.limits = c(NA, NA)` and `x.expand = 0` for the X-axis, or `y.limits = c(NA, NA)` and `y.expand = 0` for the Y-axis.

`y.24h`
: a logical to indicate whether the y labels and breaks should be formatted as 24-hour sequences

`y.age`
: A logical to indicate whether the y labels and breaks should be formatted as ages in years.

`y.scientific`, `y_secondary.scientific`
: A logical to indicate whether the y labels should be formatted in scientific notation. Defaults to `TRUE` only if the range of the y values spans more than `10e5`.

`y.percent`, `y_secondary.percent`
: A logical to indicate whether the y labels should be formatted as percentages.

`y.percent_break`
: A value on which the Y-axis should have breaks.

`y_secondary`
: Values to use for plotting along the secondary Y-axis. This functionality is poorly supported by `ggplot2` and might give unexpected results. Setting the secondary Y-axis will set the colour to the axis titles.

`y_secondary.type`
: see `type`

`y_secondary.colour`, `y_secondary.colour_fill`
: Colours to set for the secondary Y-axis, will be evaluated with `get_colour()`.

`category.type`
: Type of the `category`, one or more of: `"colour"` (default), `"shape"`, `"size"`, `"linetype"`, `"linewidth"`, `"alpha"`. There is no need to set `"fill"`, as `plot2` handles colour-setting internally and determines automatically whether the `colour` or `fill` aesthetic must be used.

`category.labels`, `category.percent`, `category.breaks`, `category.expand`, `category.midpoint`
: Settings for the plotting direction `category`.

`category.limits`
: Limits to use for a numeric category, can be length 1 or 2. Use `NA` for the highest or lowest value in the data, e.g. `category.limits = c(0, NA)` to have the scale start at zero.

`category.date_breaks`
: Breaks to use when the category contains dates, will be determined automatically if left blank. This will be passed on to `seq.Date(by = ...)` and thus can be: a number, taken to be in days, or a character string containing one of "day", "week", "month", "quarter" or "year" (optionally preceded by an integer and a space, and/or followed by "s").

`category.date_labels`
: Labels to use when the category contains dates, will be determined automatically if left blank. This accepts 'Excel' date-language such as `"d mmmm yyyy"`.

`category.character`
: A logical to indicate whether the values of the category should be forced to character. The default is `FALSE`, except for years (values between 2000 and 2050) and months (values from 1 to 12).

`x.sort`, `category.sort`, `facet.sort`
: Sorting of the plotting direction, defaults to `TRUE`, except for continuous values on the X-axis (such as dates and numbers). Applying one of the sorting methods will transform the values to an ordered factor, which `ggplot2` uses to orient the data. Valid values are:

- A manual vector of values
- `TRUE`: sort factors on their levels, otherwise sort ascending on alphabet, while maintaining numbers in the text (*numeric* sort)
- `FALSE`: sort according to the order in the data
- `NULL`: do not sort/transform at all
- `"asc"` or `"alpha"`: sort as `TRUE`
- `"desc"`: sort factors on their reversed levels, otherwise sort descending on alphabet, while maintaining numbers in the text (*numeric* sort)

- "order" or "inorder": sort as FALSE
- "freq" or "freq-desc": sort descending according to the frequencies of y computed by summarise_function (highest value first)
- "freq-asc": sort ascending according to the frequencies of y computed by summarise_function (lowest value first)

x.complete, category.complete, facet.complete
: A value to complete the data. This makes use of tidyr::full_seq() and tidyr::complete(). For example, using x.complete = 0 will apply data |> complete(full_seq(x, ...), fill = list(x = 0)). Using value TRUE (e.g., x.complete = TRUE) is identical to using value 0.

datalabels
: Values to show as datalabels, see also datalabels.format. This can be:

- Left blank. This will default to the values of y in column-type plots, or when plotting spatial 'sf' data, the values of the first column. It will print a maximum of 25 labels unless datalabels = TRUE.
- TRUE or FALSE to force or remove datalabels
- A function to calculate over .data, such as datalabels = paste(round(column1), "\n", column2)

datalabels.round
: Number of digits to round the datalabels, applies to both "%n" and "%p" for replacement (see datalabels.format).

datalabels.format
: Format to use for datalabels. This can be a function (such as euros()) or a text. For the text, "%n" will be replaced by the count number, and "%p" will be replaced by the percentage of the total count. Use datalabels.format = NULL to *not* transform the datalabels.

datalabels.colour, datalabels.colour_fill, datalabels.size, datalabels.angle, datalabels.lineheight
: Settings for the datalabels.

decimal.mark
: Decimal mark, defaults to dec_mark().

big.mark
: Thousands separator, defaults to big_mark().

summarise_function
: A function to use if the data has to be summarised, see *Examples*. This can also be NULL, which will be converted to function(x) x.

stacked
: A logical to indicate that values must be stacked.

stacked_fill
: A logical to indicate that values must be filled (i.e., stacked to 100%).

horizontal
: A logical to turn the plot 90 degrees using coord_flip(). This option also updates some theme options, so that e.g., x.lbl_italic will still apply to the original X-axis.

reverse
: A logical to reverse the *values* of category. Use legend.reverse to reverse the *legend* of category.

smooth
: A logical to add a smooth. In histograms, this will add the density count as an overlaying line (default: TRUE). In all other cases, a smooth will be added using geom_smooth() (default: FALSE).

`smooth.method`, `smooth.formula`, `smooth.se`, `smooth.level`, `smooth.alpha`, `smooth.linewidth`, `smooth.linetype`, `smooth.colour`

Settings for `smooth`.

`size` Size of the geom. Defaults to `2` for geoms point, jitter, and beeswarm; `4` for a UpSet plots (using `type = "upset"`); `5` for a dumbbell plots (using `type = "dumbbell"`); `0.75` otherwise.

`linetype` Linetype of the geom, only suitable for geoms that draw lines. Possible options are:

1. `"solid"` (default)
2. `"dashed"`
3. `"dotted"`
4. `"dotdash"`
5. `"longdash"`
6. `"twodash"`

Both numbers and text are supported, e.g., `linetype = 2` and `linetype = "dashed"` yield the same results.

`linewidth` Linewidth of the geom, only suitable for geoms that draw lines. Defaults to:

- `0.10` for sf
- `0.25` for geoms that are continous and have fills (such as area), except for geoms boxplot and violin
- `1.0` for dumbbell plots (using `type = "dumbbell"`) and UpSet plots (using `type = "upset"`)
- `0.5` otherwise (such as geoms line, boxplot, violin, and histogram)

`binwidth` Width of bins (only useful for `geom = "histogram"`), can be specified as a numeric value or as a function that calculates width from `x`, see `geom_histogram()`. It defaults to approx. `diff(range(x))` divided by 12 to 22 based on the data.

`width` Width of the geom. Defaults to `0.66` for geoms boxplot, violin and jitter, and to `0.5` otherwise.

`jitter_seed` Seed (randomisation factor) to be set when using `type = "jitter"`.

`violin_scale` Scale to be set when using `type = "violin"`, can also be set to `"area"`.

`legend.position`

Position of the legend, must be `"top"`, `"right"`, `"bottom"`, `"left"` or `"none"` (or `NA` or `NULL`), can be abbreviated. Defaults to `"right"` for numeric `category` values and 'sf' plots, and `"top"` otherwise.

`legend.reverse`, `legend.barheight`, `legend.barwidth`, `legend.nbin`, `legend.italic`, `legend.nrow`

Other settings for the legend.

`sankey.node_width`

Width of the vertical nodes in a Sankey plot.

`sankey.node_whitespace`

Whitespace between the nodes in a Sankey plot.

`sankey.alpha` Alpha of the flows in a Sankey plot.

`sankey.remove_axes`

Logical to indicate whether all axes must be removed in a Sankey plot.

zoom
: A logical to indicate if the plot should be scaled to the data, i.e., not having the x and y axes to start at 0. This will set `x.zoom = TRUE` and `y.zoom = TRUE`.

sep
: Separator character to use if multiple columns are given to either of the three directions: `x`, `category` and `facet`, e.g. `facet = c(column1, column2)`.

print
: A logical to indicate if the result should be printed instead of just returned.

text_factor
: Text factor to use, which will apply to all texts shown in the plot.

font
: Font (family) to use, can be set with `options(plot2.font = "...")`. Can be any installed system font or any of the > 1400 font names from Google Fonts. When using custom fonts in R Markdown, be sure to set the chunk option `fig.showtext = TRUE`, otherwise an informative error will be generated.

theme
: A valid `ggplot2` theme to apply, or `NULL` to use the default `theme_grey()`. This argument accepts themes (e.g., `theme_bw()`), functions (e.g., `theme_bw`) and characters themes (e.g., `"theme_bw"`). The default is `theme_minimal2()`, but can be set with `options(plot2.theme = "...")`.

background
: The background colour of the entire plot, can also be `NA` to remove it. Will be evaluated with `get_colour()`. Only applies when `theme` is not `NULL`.

markdown
: A logical to turn all labels and titles into plotmath expressions, by converting common markdown language using the `md_to_expression()` function (defaults to `TRUE`).

...
: Any argument to give to the geom. This will override automatically-set settings for the geom.

crs
: The coordinate reference system (CRS) to use. If this is not left blank, `sf::st_transform()` will be used to transform the geometric data to the new CRS.

datalabels.centroid
: A logical to indicate whether datalabels must be centred on the polygon (using `sf::st_centroid()`, the default), or be placed on the 'best' spot on the surface (using `sf::st_point_on_surface()`).

Details

For geographic information system (GIS) analysis, use the `sf` package with a data set containing geometries. The result can be used as input for `plot2()`.

`plot2-options` `plot2()` *Options*

Description

These are the package options for usage in `plot2()`. These can all be set using `options()`, and their values can be retrieved using `getOption()`.

Package Options

- big_mark
  Default character for a thousands separator
- dec_mark
  Default character for a decimal separator
- plot2.colour
  Default colour to use in plots
- plot2.colour_background
  Default background colour to use in plots
- plot2.colour_font_axis
  Default colour of the axis text (axis.title.x and axis.title.x)
- plot2.colour_font_primary
  Default colour of the plot title and plot tag
- plot2.colour_font_secondary
  Default colour of the plot subtitle
- plot2.colour_sf
  Default colour of the borders in sf plots
- plot2.colour_sf_fill
  Default colour of the area fills in sf plots
- plot2.default_type
  Default plot type
- plot2.font
  Default plot font family
- plot2.silent
  Default value to show console messages when plotting
- plot2.text_factor
  Default text factor to use in plots
- plot2.theme
  Default plot theme

---

theme_minimal2 *An Even More Minimal Theme*

Description

This ggplot2 theme provides even more white area and less clutter than theme_minimal().

Usage

```
theme_minimal2(
  ...,
  colour_font_primary = getOption("plot2.colour_font_primary", "black"),
  colour_font_secondary = getOption("plot2.colour_font_secondary", "grey35"),
  colour_font_axis = getOption("plot2.colour_font_axis", "grey25"),
  colour_background = getOption("plot2.colour_background", "white")
)
```

Arguments

`...` Arguments passed on to `ggplot2::theme()`.

`colour_font_primary` Colour to set for the plot title and tag.

`colour_font_secondary` Colour to set for the plot subtitle and caption.

`colour_font_axis` Colour to set for the axis titles on both x and y.

`colour_background` Colour to set for the background.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(hp, mpg)) +
  geom_point()

ggplot(mtcars, aes(hp, mpg)) +
  geom_point() +
  theme_minimal2()

# in plot2(), the 'theme' argument defaults to theme_minimal2():
mtcars |>
  plot2(hp, mpg)

# set to NULL to use the ggplot2 default:
mtcars |>
  plot2(hp, mpg, theme = NULL)
```

Index

∗ **datasets**
admitted_patients, 9
coord_spider, 10
netherlands, 22

add_col *(*add_type*)*, 3
add_errorbar *(*add_type*)*, 3
add_errorbar(), *6*
add_line *(*add_type*)*, 3
add_line(), *6*
add_mapping, 2
add_point *(*add_type*)*, 3
add_point(), *27*, *59*
add_sf *(*add_type*)*, 3
add_smooth *(*add_type*)*, 3
add_type, 3
add_white *(*get_colour*)*, 15
admitted_patients, 9
aes(), *5*, *34*
AMR::age_groups(), *9*
area, *33*, *65*
as_plotly, 9
as_plotly(), *10*

base::format(), *14*
beeswarm, *32*, *65*
big_mark *(*dec_mark*)*, 14
big_mark(), *14*, *20*, *32*, *64*
boxplot, *33*, *65*

character, *17*, *28*, *30*, *31*, *60*, *62*, *63*
colours(), *16*
coord_flip(), *32*, *64*
coord_spider, 10
CoordSpider, *28*, *60*
CoordSpider *(*coord_spider*)*, 10
count(), *23*
create_interactively, 13

data.frame, *6*, *9*, *22*
dec_mark, 14
dec_mark(), *14*, *20*, *32*, *64*
dollars *(*labellers*)*, 20

euros *(*labellers*)*, 20
euros(), *32*, *64*
expansion, *30*, *62*
expression, *21*, *28*, *60*

facet_wrap(), *34*
factor, *31*, *63*
filled, *32*, *64*
function, *26–28*, *32*, *58–60*, *64*

geom_blank(), *27*, *59*
geom_col(), *34*
geom_histogram(), *33*, *65*
geom_hline(), *6*
geom_line(), *6*
geom_segment(), *27*, *59*
geom_smooth(), *32*, *64*
geom_vline(), *6*
get_colour, 15
get_colour(), *5*, *16*, *29*, *31*, *34*, *61*, *63*, *66*
get_plot_title, 19
getOption(), *66*
ggplot(), *34*
ggplot2, *23*, *34*
ggplot2::aes(), *2*
ggplot2::Coord, *12*
ggplot2::fortify(), *34*
ggplot2::theme(), *68*
group_by(), *23*

histogram, *33*, *65*

jitter, *32*, *33*, *65*

labellers, 20
labs(), *34*
layout(), *10*

line, *33*, *65*
list, *21*
logical, *5*, *19*, *29–34*, *61–64*, *66*

md_to_expression, 20
md_to_expression(), *28*, *29*, *34*, *60*, *61*, *66*
mgcv::gam(), *5*
MIC values, *30*, *62*
microorganisms, *29*, *61*
move_layer, 22
mutate(), *23*

netherlands, 22

option, *14*, *28*, *60*
options(), *66*

plot2, 23
plot2(), *3*, *10*, *20*, *21*, *23*, *34*, *35*, *38*, *66*
plot2-methods, *23*, 38
plot2-options, 66
plot2.data.frame (plot2-methods), 38
plot2.default (plot2-methods), 38
plot2.formula (plot2-methods), 38
plot2.freq (plot2-methods), 38
plot2.matrix (plot2-methods), 38
plot2.sf (plot2-methods), 38
plotly_style (as_plotly), 9
plotly_style(), *10*
plotmath, *20*, *21*, *34*, *66*
point, *32*, *65*
printed, *33*, *66*

register_colour (get_colour), 15
register_colour(), *16*
regular expression, *16*
reversed, *31*, *63*

selection helpers, *26*, *27*, *58*, *59*
sf, *33*, *65*
sf::st_centroid(), *66*
sf::st_point_on_surface(), *66*
sf::st_transform(), *66*
stacked, *32*, *64*
stats::loess(), *5*
style(), *10*
summarise(), *23*

theme, *34*, *66*
theme_grey(), *34*, *66*
theme_minimal(), *67*
theme_minimal2, 67
theme_minimal2(), *34*, *66*
tibble, *9*
tidyr::complete(), *32*, *64*
tidyr::full_seq(), *32*, *64*

unregister_colour (get_colour), 15

violin, *33*, *65*