Package 'gghdx'

September 23, 2024

Title HDX Theme, Scales, and Other Conveniences for 'ggplot2'

Version 0.1.4

Description A Humanitarian Data Exchange (HDX) theme, color palettes, and scales for 'ggplot2' to allow users to easily follow the HDX visual design guide, including convenience functions for for loading and using the Source Sans 3 font.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, ggplot2, ggthemes, lifecycle, magrittr, purrr, rlang, showtext, sysfonts, tibble

Depends R (>= 2.10)

LazyData true

Suggests covr, here, knitr, rmarkdown, scales, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://github.com/OCHA-DAP/gghdx

BugReports https://github.com/OCHA-DAP/gghdx/issues

Config/testthat/edition 3

NeedsCompilation no

Author Seth Caldwell [aut, cre, cph]

Maintainer Seth Caldwell <caldwellst@gmail.com>

Repository CRAN

Date/Publication 2024-09-23 13:50:23 UTC

Contents

`df_covid` *Example COVID-19 dataset*

Description

COVID-19 dataset derived from global WHO data. Used to provide simple graph matching the example graphs on the HDX visual guide.

Usage

```
df_covid
```

Format

A data frame with 27 rows and 3 variables:

date Date

cases_monthly Confirmed COVID-19 cases in the past 30 days

flag Flag for that date

Source

https://data.humdata.org/dataviz-guide/visual-identity/#/visual-identity/colours

geom_text_hdx *Text*

Description

Text geoms are useful for labeling plots. They can be used by themselves as scatterplots or in combination with other geoms, for example, for labeling points or for annotating the height of bars. `geom_text_hdx()` adds only text to the plot. `geom_label_hdx()` draws a rectangle behind the text, making it easier to read. The only difference with the base `geom_text()` is that the default font family is Source Sans 3. `geom_label_hdx()` also incorporates a default dark gray background, white text, and no borders.

Usage

```
geom_text_hdx(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_label_hdx(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  fill = hdx_hex("gray-dark"),
  color = "white",
  fontface = "bold",
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = unit(0.25, "lines"),
  label.r = unit(0.15, "lines"),
  label.size = 0,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
```

)

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. Cannot be jointy specified with nudge_x or nudge_y. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). • A string nameing the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`parse` If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.

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

`check_overlap` If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`fill` Fill color for label box. Defaults to dark gray.

`color` Font color. Defaults to white.

`fontface` Font emphasis. Defaults to bold.

`label.padding` Amount of padding around label. Defaults to 0.25 lines.

`label.r` Radius of rounded corners. Defaults to 0.15 lines.

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

Details

Note that when you resize a plot, text labels stay the same size, even though the size of the plot area changes. This happens because the "width" and "height" of a text element are 0. Obviously, text labels do have height and width, but they are physical units, not data units. For the same reason, stacking and dodging text will not work by default, and axis limits are not automatically expanded to include all text.

`geom_text()` and `geom_label()` add labels for each row in the data, even if coordinates x, y are set to single values in the call to `geom_label()` or `geom_text()`. To add labels at specified points use `annotate()` with `annotate(geom = "text", ...)` or `annotate(geom = "label", ...)`.

To automatically position non-overlapping text labels see the ggrepel package.

Value

A ggplot2 layer that can be added to a `ggplot2::ggplot()` plot.

Examples

```
library(ggplot2)
load_source_sans_3()

p <- ggplot(
  data = mtcars,
  mapping = aes(
    x = mpg,
    y = mpg,
    label = rownames(mtcars)
  )
)

p + geom_text_hdx()
p + geom_label_hdx()
```

---

gghdx *Set HDX theme and aesthetics*

Description

`gghdx()` gives you the convenience of `theme_hdx()` without having to explicitly call it for each plot. It also allows for setting the default continuous and discrete scales to follow the HDX color scheme, including default line and point colors and area fills. `gghdx_reset()` returns all of these values back to the defaults.

Usage

```
gghdx(
  showtext = TRUE,
  base_size = 10,
  base_family = "Source Sans 3",
  horizontal = TRUE
)

gghdx_reset()
```

Arguments

showtext	logical If TRUE, uses the showtext package to add the Source Sans 3 font and runs showtext_auto() so all future plots in this session will use the font.
base_size	base font size, given in pts.
base_family	base font family
horizontal	logical Horizontal axis lines?

Details

gghdx() changes global settings for this R session. This includes updating the ggplot2 default geometries using ggplot2::update_geom_defaults() and setting global options to scale color and fill for ggplot2:

- options("ggplot2.discrete.fill")
- options("ggplot2.discrete.colour")
- options("ggplot2.continuous.fill")
- options("ggplot2.continous.colour")

The default discrete scale is scale_..._hdx() for both fill and color. For continuous scales, the default is scale_fill_gradient_hdx_mint() for fill and scale_color_gradient_hdx_sapphire() for color.

Once gghdx() is run, the easiest way to return to the default ggplot2 settings is to run gghdx_reset(). This will make changes by running:

- ggplot2::reset_theme_settings(): resets the global theme to default.
- For all of the options listed above, run options("option") <- NULL.
- showtext::showtext_end() to stop using the showtext library if it was activated.
- Runs ggplot2::update_geom_defaults() for all geometries in ggplot2_geom_defaults().

You can also simply restart your R session to return to the defaults.

Value

No return value, run for the side effects described in Details.

See Also

gghdx() relies on the following functions:

- theme_hdx() as the default theme.
- load_source_sans_3() to load the font and activate showtext.
- hdx_geom_defaults() as the default geometries to set with ggplot2::update_geom_defaults().
- scale_color_hdx_discrete() and other family of functions to set standard fill and color scales.

Examples

```
library(ggplot2)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  ) +
  labs(
    x = "Miles per gallon",
    y = "Horsepower",
    title = "Horsepower relative to miles per gallon"
  )

# see the plot using base aesthetics
p

# automatically use the gghdx theme and visuals
gghdx()
p

# get rid of the changes of gghdx
gghdx_reset()
p
```

---

ggplot2_geom_defaults *Default ggplot2 geometry aesthetics*

Description

Default geometry aesthetics from the ggplot2 library. All of the aesthetics are the standard ggplot2 defaults for those changed in `gghdx()` based on `hdx_geom_defaults()`. Used in `gghdx_reset()` to return the plotting defaults back to normal.

Usage

```
ggplot2_geom_defaults()
```

Details

These aesthetics were manually pulled from ggplot2 from the geometries' `default_aes` information, such as `ggplot2::GeomPoint$default_aes`. Since the `default_aes` is changed after `gghdx()` is run, the default geometries in this function are hardcoded.

Value

A list of geometry defaults.

Examples

```
library(purrr)
library(ggplot2)

# updating geom defaults (like default color of a point or fill for bar)
purrr::walk(
  hdx_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .),
)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  )

# see the points are automatically in HDX sapphire
p

# need to reset back to the default geometries
purrr::walk(
  ggplot2_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .)
)

# now the points are back to default black
p
```

`hdx_colors` *Hex values for HDX colors*

Description

`hdx_colors()` conveniently returns a vector of hex values for specified color ramps. Full values can be found in hdx_color_list. If you know the name of the color you want, such as "sapphire-hdx", you can use `hdx_hex(c("sapphire-hdx"))` to directly access the hex code.

Usage

```
hdx_colors(colors = c("sapphire", "mint", "tomato", "gray"))

hdx_colours(colors = c("sapphire", "mint", "tomato", "gray"))

hdx_hex(color_names)

hdx_color_names()
```

hdx_colour_names()

Arguments

colors	Specified color ramps to return. Some set of "sapphire", "mint", "tomato", and "gray. By default returns all colors.
color_names	Vector of color names. Valid values are all available using hdx_colors

Details

All valid color names are in the named vector returned by hdx_colors() or accessible in the convenient hdx_color_names().

Value

- hdx_colors() returns a named vector of hex values.
- hdx_color_names() returns a character vector of color names.

See Also

Other color hdx: hdx_color_list, hdx_pal_discrete()

Examples

```
# get hex values
hdx_colors()
hdx_colors("sapphire")

# get color names
hdx_color_names()
```

---

hdx_color_list *HDX color ramps*

Description

List of color ramps from the HDX visual identity. These are mint, sapphire, tomato, and grays.

Usage

hdx_color_list

Format

A list with 4 data frames:

Source

https://data.humdata.org/dataviz-guide/visual-identity/#/visual-identity/colours/

See Also

Other color hdx: hdx_colors(), hdx_pal_discrete()

---

hdx_display_pal *Display HDX palette*

Description

Displays the HDX color palettes. By default, shows all values for all palettes. You can change the number of values for each palette or only show a subset of the available palettes (from hdx_pal_...()).

Usage

```
hdx_display_pal(
  n = NULL,
  palette = c("discrete", "gray", "mint", "sapphire", "tomato")
)
```

Arguments

n	Number of colors for each palette to show.
palette	Character vector of palettes to show.

Value

Plot of HDX color palettes.

Examples

```
hdx_display_pal()
hdx_display_pal(n = 3)
```

hdx_geom_defaults *Default HDX geometry aesthetics*

Description

Default geometry aesthetics fitting the HDX design guide. Used in `gghdx()` to set default fill, color, size, and point geometry defaults, which is not possible using just `theme_hdx()`.

Usage

```
hdx_geom_defaults()
```

Details

Derived from the ggthemr methods.

Value

A list of geometry defaults.

See Also

- `gghdx()` for automatically setting default geometries, along with other styling.
- `ggplot2_geom_defaults()` for the ggplot2 default aesthetics.

Examples

```
library(purrr)
library(ggplot2)

# updating geom defaults (like default color of a point or fill for bar)
purrr::walk(
  hdx_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .),
)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  )

# see the points are automatically in HDX sapphire
p

# need to reset back to the default geometries
purrr::walk(
```

```
  ggplot2_geom_defaults(),
  ~ do.call(what = ggplot2::update_geom_defaults, args = .)
)

# now the points are back to default black
p
```

`hdx_pal_discrete` *HDX color palette (discrete)*

Description

The hues in the HDX palette are sapphire, mint, and tomato.

Usage

```
hdx_pal_discrete()

hdx_pal_sapphire()

hdx_pal_tomato()

hdx_pal_mint()

hdx_pal_gray()
```

Details

`hdx_pal_discrete()` utilizes all hues for up to a 12 element discrete scale.

`hdx_pal_mint()`, `hdx_pal_tomato()`, and `hdx_pal_sapphire()` allow for a 4 element discrete scale using only the specified color. These are color ramps with a range from dark, normal (HDX standard), light, and ultra light.

Value

A palette function.

See Also

Other color hdx: `hdx_color_list`, `hdx_colors()`

Examples

```
hist(mtcars$mpg, col = hdx_pal_discrete()(5))
```

`label_number_hdx` *Format and labels numbers in HDX key figures style*

Description

Use `format_number_hdx()` to directly format numeric vectors, and use `label_number_hdx()` in the same way as the `scales::` family of label functions. The return value of `label_number_hdx()` is a function, based on the `additional_prefix`. So you should pass it in to `scales_...()` `labels` parameter in the same way as `scales_...()`

Usage

```
label_number_hdx(additional_prefix = "")

format_number_hdx(x, additional_prefix = "")
```

Arguments

`additional_prefix`	Additional prefix to add to string, that will come between `sign_prefix` and the number. For example, `"$"` could produce a return value of `-$1.1K`.
`x`	Numeric vector to format

Details

Numeric vectors are formatted in the HDX style for key figures, which abbreviates numbers 1,000 and above to X.YK, 10,000 and above to XYK, 100,000 and above to XYZK, and the same for 1,000,000 and above, replacing the K with an M, and the same for B. Details of the data viz style can be found in the visualization guidelines

Just for continuity, values are labeled with T for trillion, and that is the maximum formatting available, anything above the trillions will continue to be truncated to report in the trillions.

Deals with negative values in case those ever need to be formatted in similar manners. Also ensures that rounding is performed so numbers look correct. Not to be used for percents, which should just use `scales::label_percent()`.

Value

`label_number_hdx()`: "labelling" function, in the same way as `scales::label_...()` functions work, i.e. a function that takes `x` and returns a labelled character vector of `length(x)`.

`format_number_hdx()`: Formatted character vector of number strings

Examples

```
# label_number_hdx()

library(ggplot2)
```

```
# discrete scaling
p <- ggplot(txhousing) +
  geom_point(
    aes(
      x = median,
      y = volume
    )
  )

p

p +
  scale_x_continuous(
    labels = label_number_hdx("$")
  ) +
  scale_y_continuous(
    labels = label_number_hdx()
  )

# number_hdx()

x <- c(1234, 7654321)
format_number_hdx(x)
format_number_hdx(x, "$")
```

`load_source_sans_3` *Load and use Source Sans 3*

Description

Simple wrapper for `sysfonts::font_add_google()` and `showtext::showtext_auto()` to load the Source Sans 3 font and specify all plots to automatically use showtext. Use to load the default font family for `geom_text_hdx()` and `geom_label_hdx()`.

Usage

```
load_source_sans_3(family = NULL, regular = NULL)
```

Arguments

`family`	Character string for the Source Sans 3 family. If `NULL`, defaults to `"Source Sans 3"`, the standard family name. See "Details" in the `sysfonts::font_add()` documentation for further explanation. Used only when no internet connection is available to directly load from Google.
`regular`	Path to the font file for the regular font face. If `NULL`, defaults to `"SourceSans3-Regular.ttf"`, the standard file name downloaded from Source Sans 3. Used only when no internet connection is available to directly load from Google.

Details

By default, the font is loaded from Google using `sysfonts::font_add_google()`. If an internet connection is unavailable, then attempts to use a locally installed version of the font using `sysfonts::font_add(family, regular)`. If you have the font installed but still receive an error from this function, check the `family` and `regular` arguments match your installed font.

Value

Nothing, run for side effect of loading the font and activating showtext.

See Also

`gghdx()` for automatically running `load_source_sans_3()`, along with other styling.

Examples

```
library(ggplot2)
p <- ggplot(
  data = mtcars,
  mapping = aes(
    x = mpg,
    y = mpg,
    label = rownames(mtcars)
  )
)

# font not loaded so error will be generated
try(p + geom_label_hdx())

load_source_sans_3()

p + geom_label_hdx()
```

`scale_color_hdx_discrete` *HDX color scales*

Description

Color scales using the HDX palette. For discrete color scales, the `scale_color_hdx_...()` and `scale_fill_hdx_...()` family of functions are available. For gradient scales, use `scale_color_gradient_hdx()` and `scale_fill_gradient_hdx()` functions for a single color scale or `scale_..._gradient2...()` alternative.

Usage

```
scale_color_hdx_discrete(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_discrete(na.value = hdx_hex("gray-light"), ...)

scale_color_hdx_gray(na.value = hdx_hex("tomato-hdx"), ...)

scale_colour_hdx_gray(na.value = hdx_hex("tomato-hdx"), ...)

scale_colour_hdx_grey(na.value = hdx_hex("tomato-hdx"), ...)

scale_color_hdx_grey(na.value = hdx_hex("tomato-hdx"), ...)

scale_color_hdx_mint(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_mint(na.value = hdx_hex("gray-light"), ...)

scale_color_hdx_sapphire(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_sapphire(na.value = hdx_hex("gray-light"), ...)

scale_color_hdx_tomato(na.value = hdx_hex("gray-light"), ...)

scale_colour_hdx_tomato(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_discrete(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_gray(na.value = hdx_hex("tomato-hdx"), ...)

scale_fill_hdx_grey(na.value = hdx_hex("tomato-hdx"), ...)

scale_fill_hdx_mint(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_sapphire(na.value = hdx_hex("gray-light"), ...)

scale_fill_hdx_tomato(na.value = hdx_hex("gray-light"), ...)

scale_fill_gradient_hdx(na.value = "transparent", ...)

scale_fill_gradient_hdx_sapphire(na.value = "transparent", ...)

scale_fill_gradient_hdx_mint(na.value = "transparent", ...)

scale_fill_gradient_hdx_tomato(na.value = "transparent", ...)

scale_color_gradient_hdx(na.value = "transparent", ...)

scale_colour_gradient_hdx(na.value = "transparent", ...)
```

```
scale_color_gradient_hdx_sapphire(na.value = "transparent", ...)

scale_colour_gradient_hdx_sapphire(na.value = "transparent", ...)

scale_color_gradient_hdx_mint(na.value = "transparent", ...)

scale_colour_gradient_hdx_mint(na.value = "transparent", ...)

scale_color_gradient_hdx_tomato(na.value = "transparent", ...)

scale_colour_gradient_hdx_tomato(na.value = "transparent", ...)

scale_color_gradient2_hdx(na.value = "transparent", ...)

scale_colour_gradient2_hdx(na.value = "transparent", ...)

scale_fill_gradient2_hdx(na.value = "transparent", ...)
```

Arguments

`na.value`	Colour to use for missing values
`...`	Arguments passed on to `discrete_scale`

`palette` A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take (e.g., `scales::pal_hue()`).

`breaks` One of:

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

`limits` One of:

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

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`labels` One of:

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

guide A function used to create a guide or its name. See guides() for more information.

call The call used to construct the scale for reporting messages.

super The super class to use for the constructed scale

Value

Relevant ggplot2 scale object to add to a ggplot2::ggplot() plot, either ggplot2::ScaleDiscrete or ggplot2::ScaleContinuous.

See Also

gghdx() for setting default fill and color scaling, along with other styling.

Examples

```
library(ggplot2)

# discrete scaling
p1 <- ggplot(iris) +
  geom_point(
    aes(
      x = Sepal.Length,
      y = Petal.Width,
      color = Species
    )
  )

p1

p1 + scale_color_hdx_discrete()
p1 + scale_color_hdx_mint()

# use gradient scaling
p2 <- ggplot(iris) +
  geom_point(
    aes(
      x = Sepal.Length,
      y = Petal.Width,
      color = Petal.Length
    )
  )
```

```
p2

p2 + scale_color_gradient_hdx_mint()
p2 + scale_color_gradient_hdx_tomato()
```

`scale_y_continuous_hdx` *Position scales for continuous y data*

Description

`scale_y_continuous_hdx()` and the three variants with different `trans` arguments are defaults scales for the y axis that ensures the distance from data to the y-axis is reduced to 0, as is common throughout the HDX data visualization guidelines. This is done by setting `expand = c(0, 0)`.

Usage

```
scale_y_continuous_hdx(...)

scale_y_log10_hdx(...)

scale_y_reverse_hdx(...)

scale_y_sqrt_hdx(...)
```

Arguments

`...`	Other arguments pass on to `ggplot2::scale_y_continuous()`.

Details

For simple manipulation of labels and limits, you may wish to use `labs()` and `lims()` instead.

Value

`ggplot2::ScaleContinuousPosition` object to scale a `ggplot2::ggplot()` plot.

Examples

```
library(ggplot2)

p <- ggplot(df_covid) +
  geom_line(
    aes(
      x = date,
      y = cases_monthly
    )
  )
```

```
p

# start y axis at 0
p + scale_y_continuous_hdx()
p + scale_y_log10_hdx()
```

---

theme_hdx *ggplot color theme based on HDX visual design guide*

Description

A theme that approximates the style of the *Humanitarian Data Exchange (HDX)*.

Usage

```
theme_hdx(base_size = 10, base_family = "Source Sans 3", horizontal = TRUE)
```

Arguments

base_size	base font size, given in pts.
base_family	base font family
horizontal	logical Horizontal axis lines?

Details

`theme_hdx()` implements a chart that follows the general visual guide of the HDX platform, as defined in the dataviz-guide.

Use `scale_color_hdx_discrete()` with this theme.

HDX uses two fonts in its official typography, with the free Google font Source Sans 3 being easily available in R. Use the **sysfonts** package to add the Google font easily.

Value

A `theme()` to stylize a `ggplot2::ggplot()` plot.

References

- Humanitarian Data Exchange
- Google Fonts, Source Sans 3
- HDX Dataviz Guide

See Also

`gghdx()` for automatically applying the theme to all plots in this current R session, along with other styling.

Examples

```
library(ggplot2)

p <- ggplot(mtcars) +
  geom_point(
    aes(
      x = mpg,
      y = hp
    )
  ) +
  labs(
    x = "Miles per gallon",
    y = "Horsepower",
    title = "Horsepower relative to miles per gallon"
  )

# the default font is source sans 3
# an error will occur if not loaded before using theme_hdx()
try(p + theme_hdx())

# you can change the base family
p + theme_hdx(base_family = "sans")

# or load Source Sans 3 using gghdx() or load_source_sans_3()
load_source_sans_3()
p + theme_hdx()

# we can change the axis line direction depending on the plot
p + theme_hdx(horizontal = FALSE)
```

Index