

Package ‘ggstratify’

September 25, 2026

Title Fast Stratified Descriptive Figures with a Point-and-Click GUI

Version 0.2.0

Description A point-and-click 'shiny' interface for the descriptive analysis that comes before any model is chosen. Pass a data frame, pick the variable to describe, and add the layers you want to see it within: a second variable becomes the panels of a 'ggplot2' `facet_wrap()`, and further variables become separate figures, one file each, taken either one variable at a time or crossed. Every stratum is reported with the number of observations behind it, on the figure and on each of its panels; strata that contain none are listed rather than dropped, and rows with a missing value in a layer variable are excluded and counted. A continuous variable can be categorized into quantile groups, equal-width bins or user-supplied cut points, a variable of any type can be turned into whether it is missing or observed, so that the rows a layer would exclude become a stratum of their own, and a date or date-time variable can be read at a chosen resolution, either as a calendar period or as a position in the yearly cycle such as the month or the season; any of them can then be used as a layer. The figure types follow those offered by the 'ggplotgui' package and add the line plot for change over time, an optional LOWESS smoother, and the Kaplan-Meier curve estimated by 'survival', with an optional number-at-risk table. An error bar can show a standard error or a confidence interval, the latter from the t distribution for a mean and from the Clopper-Pearson or Wilson method for a proportion, and the points can be joined by a line computed from the same summary, which is how a trend over time is read. A survey weight can be set: the figure is then drawn from the weighted data, every count is reported both as rows and as the sum of the weights, and error bars and confidence bands are design-based estimates from the 'survey' package. Columns are described as they are typed, so convert each to the type you mean first. Figures are written as PNG or SVG, and the application prints the 'ggplot2' code behind the figure on screen, so that a description can be repeated, shared or accounted for later. Everything runs locally, with no network access and no AI involved.

License GPL-3

URL <https://github.com/AkiShiroshita/ggstratify>,
<https://akishiroshita.github.io/ggstratify/>

BugReports <https://github.com/AkiShiroshita/ggstratify/issues>

Encoding UTF-8

Language en-US

Depends R (>= 4.1)

Imports bslib (>= 0.5.0), checkmate (>= 2.1.0), data.table (>= 1.14.0), ggplot2 (>= 3.4.0), grDevices, patchwork (>= 1.1.0), ragg, shiny (>= 1.7.0), stats, survey (>= 4.5), survival (>= 3.2.0), utils

Suggests knitr, rmarkdown, svglite (>= 2.1.0), testthat (>= 3.0.0), tibble

VignetteBuilder knitr

LazyData true

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Akihiro Shiroshita [aut, cre, cph],
Yuki Kataoka [aut]

Maintainer Akihiro Shiroshita <akihirokun8@gmail.com>

Repository CRAN

Date/Publication 2026-09-25 07:10:02 UTC

Contents

epi_cohort	2
ggstratify	4
Index	7

epi_cohort	<i>A synthetic clinical cohort</i>
------------	------------------------------------

Description

Six hundred simulated patients, shaped like the descriptive tables that motivate this package: a few continuous measurements crossed with several categorical variables worth stratifying on.

Usage

epi_cohort

Format

A data frame with 600 rows and 13 columns:

id Patient identifier, "P0001" to "P0600".

age Age in years.

sex Factor: "Male", "Female".

site Factor: "Site A", "Site B", "Site C", "Site D". "Site D" has no observations.

treatment Factor: "Control", "Low dose", "High dose".

severity Factor: "Mild", "Moderate", "Severe".

bmi Body mass index, kg/m².

crp C-reactive protein, mg/L. NA for 122 patients who were not tested, more often the milder ones.

los_days Length of stay in days.

fu_days Days of follow-up, to death or to censoring. Censoring is by dropout or by the end of the study at 365 days.

death 1 if the patient died during follow-up, 0 if censored.

admit_date Date of admission, over 2021 to 2023. Moderate and severe cases cluster in the winter, mild ones in the summer.

svy_weight Survey weight: the number of admissions each patient stands for, larger for the under-sampled mild presentations. Sums to about 50,000.

Details

Three features are deliberate. `site` declares a fourth level, "Site D", that recruited nobody, so stratifying by `site` produces a stratum with $n = 0$; the app lists it and draws no figure. `severity` has a small "Severe" group – 20 patients against 381 and 199 – which is what the minimum-n control is there to be tried on: it is drawn at the default minimum of 10, and disappears from the figures, with a reason, as soon as the minimum is raised past 20. `crp` is the only column with missing values, and they are not missing at random: the milder the presentation, the more often the measurement was left undone, so 105 of the 381 mild patients have no CRP against none of the 20 severe ones. Deriving *missing vs observed* from `crp` and describing another variable within it is what that pattern is there for.

The data are simulated. They describe no real patients and support no clinical conclusion.

`admit_date` is the fourth of these. It spans three calendar years and has several hundred distinct values, so it cannot be stratified on as it stands – that is the point of it. Read at a resolution first, by season or by month of the year, it becomes a variable with four or twelve levels that can be used as a layer like any other. The seasonality is in the case mix rather than in the number of admissions: moderate and severe presentations cluster in the winter and mild ones spread into the summer, so a season carries a real difference in `crp` and in `los_days` rather than noise.

`svy_weight` is the fifth. It reads the cohort as a sample of some 50,000 admissions in which severe presentations were over-sampled and mild ones under-sampled, and gives each patient the number of admissions they stand for. Set it as the survey weight and the weighted cohort is mostly mild, as the admissions were: a weighted mean of `crp` and a weighted proportion of death come out below the unweighted ones. `site` can serve as the sampling strata to try those controls with it.

`fu_days` and `death` make the data usable for a Kaplan-Meier curve, `age` for the methods that cut a continuous variable into groups, `crp` for the one that splits a variable by whether it has a value, and `admit_date` for the one that reads a date at a chosen resolution.

Source

Simulated by data-raw/epi_cohort.R.

Examples

```
str(epi_cohort)

# The empty stratum that the app reports with n = 0.
table(epi_cohort$site)

# Who was not tested, by how ill they were.
table(epi_cohort$severity, is.na(epi_cohort$crp))

# Too many distinct dates to stratify on, until it is read at a resolution.
length(unique(epi_cohort$admit_date))
table(factor(month.abb[data.table::month(epi_cohort$admit_date)],
          levels = month.abb))
```

ggstratify

Describe a data set, one layer at a time

Description

Launches a point-and-click Shiny interface for descriptive analysis: pick the variable you want to describe, then add the layers you want to see it within. The figure, the number of observations behind it and the R code that reproduces it all appear together.

Usage

```
ggstratify(dataset, launch.browser = TRUE, ...)
```

Arguments

dataset	The data to describe, and the one thing the function needs: a data frame – including a tibble or a data.table – or a matrix, already read into your session and already of the types you mean. A file path is not accepted; see <i>Before you start</i> . Whatever it is given becomes a plain data.table, so a grouped tibble is described as its rows, not as its groups. The object is copied, never modified in place, and its name is what the generated code refers to it by – so pass a named object, rather than an expression: ggstratify(head(cohort)) is described perfectly well, but the code it prints says d <- mydata, because head(cohort) is not a name.
launch.browser	Passed to shiny::runApp() . TRUE opens the system browser.
...	Further arguments passed to shiny::runApp() .

Value

Invisibly NULL, after the app is closed. Called for its side effect of running a Shiny application.

The layers

A description has a variable being described and, around it, the variables you want to see it within. ggstratify calls those the layers, and asks you to place each one:

One layer The variable on its own. A plain ggplot2 figure.

Two layers The second variable becomes the panels of a `facet_wrap()`, so the whole comparison is one figure.

Three or more Two layers is what a single figure holds. Beyond that, you choose: one variable stays as the `facet_wrap()` panels, and the rest become separate figures – one file each. With columns A, B, C and D you might describe D, panel it by C, and get one figure per level of A and of B.

The variables that make separate figures can be treated separately, which is the default – ticking sex and treatment gives the figures `sex_M`, `sex_F`, `treatment_A`, ... – or crossed, giving one figure per observed combination. Separately is usually the right question for descriptive work: crossing two five-level variables gives twenty-five mostly empty figures.

What it tells you

Every level is reported with the number of observations it contains, and that number is written into the figure title and onto each panel strip inside it: a strip reads `site: Site A (N = 303)` rather than `Site A`, so that a level which does not explain itself – the bare range a categorized variable produces – still says which variable it is a level of. A level with no observations at all (an unused factor level) is listed with `N = 0` and produces no figure. Rows with a missing value in any layer variable are excluded – a row that does not say which panel it belongs to cannot be drawn in one – and the number excluded is reported on the Strata tab and by the generated code.

Three things descriptive work keeps needing

Under **Derive a variable**, a new categorical variable is made from one you already have, and can then be used as a layer like any other categorical variable. A continuous variable becomes quantile groups, equal-width bins or your own cut points. Any variable, of any type, becomes *missing vs observed*: the rows where it has a value and the rows where it does not.

That last one is the layer for the question of who is not in the data. Rows with a missing value in a layer variable are excluded, but a missing-vs-observed variable never has one – `is.na()` answers for every row – so the rows that some other layer would have dropped are exactly the ones it puts in front of you. Describe another variable within it to see how the people whose CRP was never measured differ from the people whose CRP was. Describing the variable itself within its own missingness leaves the Missing figure with nothing to draw, and the app says so under **Layers** rather than drawing an empty figure without comment.

Under **Type of graph**, *Kaplan-Meier curve* draws survival curves from a time variable and an event indicator, fitted with `survival::survfit()` within each group, panel and figure, optionally with a confidence band, censoring marks and a number-at-risk table under the curve. *Line* draws change over time: put time on the X axis and the measurement on the Y axis, and name the subject identifier under **One line per** to get one line per subject.

A *Line* or *Scatter* figure can carry a LOWESS smoother – `stats::loess()` through `ggplot2::geom_smooth()` – with its span under your control and, when a grouping variable is set, one fit per group.

Survey weights

Under **Describe**, **Survey weight** takes a numeric column that is never negative. Each row then counts for as many people as its weight: a histogram, density, boxplot or violin is drawn through `ggplot2`'s `weight` aesthetic, and a smoother is fitted with the weights. Every count is reported twice, as the rows a figure is drawn from and as the sum of their weights. The bar on a *Dot + Error* figure and the band on a *Kaplan-Meier curve* are design-based estimates from `survey::svymean()`, `survey::svyciprop()` and `survey::svykm()`. **Sampling strata** and **Clusters** complete the design, with a cluster ID read within its stratum; the design is built over the whole sample, and every panel and figure is a subpopulation of it. A dotplot cannot be weighted, and rows with no weight, stratum or cluster are excluded and counted. `svy_weight` in `epi_cohort` is a weight to try this on.

Before you start

Convert each column to the type you mean it to have – numeric for measurements, factor for groups, with the levels in the order you want them read – before passing the data. `ggstratify` describes what it is given; it does not guess what you meant. A grouping variable stored as 1, 2, 3 will be described as a number.

This is why the data must be an object you already have in your session. There is no file to choose from inside the application, and no file path to hand it: a data set that arrives by being read from disk arrives with types that a reader guessed, and it is then described on those guessed types without anyone having looked at them. Read the file yourself, run `str()` or `summary()` over the result, fix the types that are wrong, and pass that object:

```
cohort <- read.csv("cohort.csv")
cohort$treatment <- factor(cohort$treatment,
                           levels = c("Control", "Low dose", "High dose"))

str(cohort)
ggstratify(cohort)
```

The figure types, themes and palettes otherwise match those offered by the **ggplotgui** package. All data handling uses **data.table**. Figures are written as PNG (through **ragg**) or as SVG, and the "R-code" tab shows the self-contained **ggplot2** code for the figure on screen – just the figure, since writing the files is what the export button is for. The preview, the export button and that code are all produced by the same code generator, so the code you are shown is the code that made the figure.

Examples

```
if (interactive()) {
  ggstratify(iris)
  ggstratify(epi_cohort)
}
```

Index

* **datasets**

epi_cohort, [2](#)

epi_cohort, [2](#), [6](#)

ggplot2::geom_smooth(), [5](#)

ggstratify, [4](#)

shiny::runApp(), [4](#)

str(), [6](#)

summary(), [6](#)

survey::svyciprop(), [6](#)

survey::svykm(), [6](#)

survey::svymean(), [6](#)

survival::survfit(), [5](#)