Package 'compstatslib'

August 8, 2026

Type Package

Title Interactive 2D and 3D Visualization of Data and Statistical Concepts

Version 0.8.0

Date 2026-08-02

Description Interactive gadgets and plotting functions for visualizing data sets and statistical concepts in two and three dimensions. Explore a data frame as a 3D point cloud you can rotate, fit a moderated (interaction) regression and view its surface as a 3D wireframe, or plot principal components and regression fits in two dimensions. Each interactive gadget returns the call that reproduces its final view, including the viewing angle, so an exploratory session can be pasted into a script or report. Also provides simulation-based demonstrations of sampling distributions, confidence intervals, t-tests, and matrix inversion for teaching and self-study.

Imports lattice, miniUI, plotly, shiny

Depends R (>= 4.1.0)

License MIT + file LICENSE

URL https://github.com/compstatslib/compstatslib

BugReports https://github.com/compstatslib/compstatslib/issues

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

LazyData true

NeedsCompilation no

Author Soumya Ray [aut, cre], Daniele Melotti [aut]

Maintainer Soumya Ray <soumya.ray@gmail.com>

Repository CRAN

Date/Publication 2026-08-08 11:50:13 UTC

Contents

`compstatslib-reproduce`

Reproduction calls returned by the interactive gadgets

Description

Every `interactive_*()` gadget hands its final state back to the console in one of two shapes, both of which print as the `plot_*()` call that reproduces what was on screen:

Usage

```
## S3 method for class 'compstatslib_args'
print(x, ...)

## S3 method for class 'compstatslib_points'
print(x, ...)
```

Arguments

x	An object of one of the two classes.
...	Ignored, for consistency with print.

Details

`compstatslib_args` A named list of arguments for the paired plot function. Still an ordinary list, so `do.call(plot_fn, result)` works.

`compstatslib_points` A dataframe of points, with any further arguments carried alongside. Still an ordinary dataframe, so `nrow()`, `[` and passing it straight back to the plot function all behave as before.

Derived or accumulated state that a user would not retype (a `prcomp` result, an accumulated sampling cache) rides along as an attribute rather than appearing in the printed call.

Value

x, invisibly. Called for the printed reproduction call.

Examples

```
if (interactive()) {
  # A settings-style gadget returns compstatslib_args
  result <- interactive_t_test()
  result                              # prints plot_t_test(...)
  do.call(plot_t_test, result)        # still a plain list

  # A points-style gadget returns compstatslib_points
  pts <- interactive_regression()
  pts                                 # prints the call, then the points
  nrow(pts)                           # still a plain dataframe
  plot_regression(pts)

  # Derived state rides along as an attribute
  pca_pts <- interactive_pca()
  attr(pca_pts, "pca")
}
```

---

interactive_logit *compstatslib interactive_logit() function*

Description

Interactive visualization function that lets you point-and-click to add data points, while it automatically plots and updates a logistic regression curve and associated statistics.

Usage

```
interactive_logit(
  points = data.frame(),
  formula = y ~ x,
  min_x = 0,
  max_x = 50,
  ...
)
```

Arguments

points	An optional dataframe of x and y points to plot and estimate the regression. If no points are provided, the user is free to click and create points on the plot area.
formula	A formula to use in estimating logit (e.g., y ~ x).
min_x	Minimum x value for the plot range.
max_x	Maximum x value for the plot range.
...	Further arguments passed to the plot_logit() function that produces the plot.

Details

Click on the plotting area to add points and see a corresponding logistic regression curve. Click "Done" to return the points to the console.

Value

On "Done", a compstatslib_points object: the dataframe of point coordinates, carrying formula, min_x and max_x alongside. It prints the plot_logit call that reproduces the plot and then the points themselves, and is still an ordinary dataframe. On "Cancel", NULL. See compstatslib-reproduce.

See Also

plot_logit

Examples

```
if (interactive()) {
  # Selecting coordinates on the plot area, storing them in 'pts'
  pts <- interactive_logit()

  # Reproduce the curve non-interactively
  plot_logit(pts, formula = y ~ x)

  # Or replot them in the gadget and keep clicking
  interactive_logit(pts)
}
```

`interactive_matrix_inverse` *compstatslib interactive_matrix_inverse() function*

Description

Interactive function that allows one to visualize a matrix inversion using sliders to adjust matrix parameters.

Usage

```
interactive_matrix_inverse(x1 = 1, y1 = 2, x2 = 2, y2 = 1)
```

Arguments

`x1`	The first row (or column) vector of matrix A. This parameter is set to 1 by default.
`y1`	The second row (or column) vector of matrix A. This parameter is set to 2 by default. The area of the parallelogram resulting from these two vectors is the determinant of matrix A.
`x2`	The first row (or column) vector of the inverse matrix A^(-1). This parameter is set to 2 by default.
`y2`	The second row (or column) vector of the inverse matrix A^(-1). This parameter is set to 1 by default. The area of the parallelogram resulting from these two vectors is the determinant of the inverse matrix A^(-1).

Details

Inspired by: https://math.stackexchange.com/questions/295250/geometric-interpretations-of-matrix-inver
1922830#1922830

Use the sliders in the viewer to adjust the matrix parameters and see the resulting transformation. Click "Done" to close.

Value

On "Done", a `compstatslib_args` object: a named list of `x1`, `y1`, `x2` and `y2` at their final slider positions, which prints the `plot_matrix_inverse` call that reproduces the plot. It is still an ordinary list, so `do.call(plot_matrix_inverse, result)` works. On "Cancel", `NULL`. See compstatslib-reproduce.

See Also

`plot_matrix_inverse`

Examples

```
if (interactive()) {
  # Move the sliders, then Done
  result <- interactive_matrix_inverse()

  # Reproduce the plot non-interactively
  do.call(plot_matrix_inverse, result)

  # Or resume the gadget where it was left
  do.call(interactive_matrix_inverse, result)
}
```

---

interactive_moderation_3d

compstatslib interactive_moderation_3d() function

Description

Interactive 3D moderation visualization. Shiny gadget wrapping `plot_moderation_3d` with rotation sliders for live viewing of the wireframe surface from any angle. Useful in classroom demos to show how the interaction-induced surface twist becomes visible (or vanishes) under different viewing angles.

Usage

```
interactive_moderation_3d(
  formula = y ~ x * z,
  data = moderation_data,
  iv = NULL,
  mod = NULL,
  ...
)
```

Arguments

`formula`	A model formula. With exactly two predictors the first is taken as the IV and the second as the moderator. With three or more predictors, you must supply `iv` and `mod`. Defaults to `y ~ x * z`, matching the bundled `moderation_data`.
`data`	A data frame containing the variables named in `formula`. Defaults to the bundled `moderation_data`.
`iv`	Optional character. Name of the predictor to place on the first horizontal axis. Required when `formula` has more than two predictors.
`mod`	Optional character. Name of the predictor to place on the second horizontal axis. Required when `formula` has more than two predictors.
`...`	Further arguments forwarded to `plot_moderation_3d` (and from there to `wireframe`).

Details

The gadget exposes two rotation sliders:

- Z rotation (0–360, default 40) — rotation about the vertical axis. Try 0 to align the IV slope plane with the screen, or 270 to align the moderator slope plane.
- X rotation (-90 to -70, default -70) — tilt above/below the surface.

The *vertical* axis of the rendered plot depicts the outcome (DV) regardless of variable naming. With the bundled data the moderator is named `z`, but `z` appears on a horizontal plot axis, not the vertical "z-axis" of the wireframe.

This function only works in IDEs that support Shiny gadgets (RStudio, Positron, etc.).

Value

On "Done", a `compstatslib_args` object: a named list of `formula`, `data`, `iv`, `mod` and — crucially — the `z_rot` / `x_rot` viewing angles the gadget was left at, plus anything passed through `...`. It prints the `plot_moderation_3d` call that reproduces the view, and is still an ordinary list, so `do.call(plot_moderation_3d, result)` works. On "Cancel", `NULL`. See compstatslib-reproduce.

The printed call shows the expression you passed for `data` rather than its values, so pass a named variable if you want a pasteable call.

See Also

`plot_moderation_3d` for the non-interactive version; `moderation_data` for the bundled teaching dataset.

Examples

```
if (interactive()) {
  # Default synthetic example
  interactive_moderation_3d()

  # User's own data and naming
  set.seed(1)
  my_df <- data.frame(
    score = rnorm(100),
    time  = rnorm(100),
    dose  = rnorm(100)
  )
  interactive_moderation_3d(score ~ time * dose, my_df)

  # Multi-predictor model — `w` is bundled noise, included as a
  # control. Pick which two predictors go on the plot axes; the
  # rest are held at typical values (here, `mean(w)`).
  interactive_moderation_3d(y ~ x + z + w + x:z, moderation_data,
                            iv = "x", mod = "z")

  # Rotate to an angle worth keeping, click Done, then freeze that
  # exact view into a script or report
```

```
  result <- interactive_moderation_3d()
  do.call(plot_moderation_3d, result)
}
```

`interactive_pca` *compstatslib interactive_pca() function*

Description

Interactive visualization function that lets you point-and-click to add data points, while it automatically plots and updates principal component vectors.

Usage

```
interactive_pca(meancenter = TRUE)
```

Arguments

`meancenter`	A logical parameter that will mean center the points if set to TRUE, while it will not mean center the points if set to FALSE. By default it is set to TRUE.

Details

Click on the plotting area to add points and see corresponding principal components. Click "Done" to return results to the console.

Value

On "Done", a `compstatslib_points` object: the dataframe of point coordinates, carrying `meancenter` alongside. It prints the `plot_pca` call that reproduces the plot and then the points themselves, and is still an ordinary dataframe. The `prcomp` result (or `NULL` if fewer than 3 points were added) rides along as `attr(result, "pca")` rather than appearing in the printed call. On "Cancel", `NULL`. See compstatslib-reproduce.

See Also

`plot_pca`

Examples

```
if (interactive()) {
  # Click 3 or more points, then Done
  pts <- interactive_pca()

  # Reproduce the plot non-interactively
  plot_pca(pts)

  # The prcomp result is carried along as an attribute
```

```
  attr(pts, "pca")

  # Start without mean-centering
  interactive_pca(meancenter = FALSE)
}
```

`interactive_regression` *compstatslib interactive_regression() function*

Description

Interactive visualization function that lets you point-and-click to add data points, while it automatically plots and updates a regression line and associated statistics.

Usage

```
interactive_regression(points = data.frame(), ...)
```

Arguments

`points`	An optional `dataframe` of *x* and *y* points to plot and estimate the regression. If no `points` are provided, the user is free to click and create points on the plot area.
`...`	Further arguments passed to the `plot_regression()` function that produces the plot.

Details

Click on the plot area to add points and see a corresponding regression line. Click "Done" to return the points to the console.

Value

On "Done", a `compstatslib_points` object: the dataframe of point coordinates, which prints the `plot_regression` call that reproduces the plot and then the points themselves. It is still an ordinary dataframe, so `nrow()`, `[` and `plot_regression(result)` all work. On "Cancel", `NULL`. See compstatslib-reproduce.

The plot area additionally displays:

`Raw intercept`	The y-coordinate at which the regression line crosses the y-axis.
`Raw slope`	The value of the slope parameter.
`Correlation`	The strength of the linear relationship.
`SSR`	The sum of squares regression.
`SSE`	The sum of squares error.
`SST`	The sum of squares total.
`R-squared`	The multiple coefficient of determination.

See Also

plot_regression

Examples

```
if (interactive()) {
  # Selecting coordinates on the plot area, storing them in 'pts'
  pts <- interactive_regression()

  # Reproduce the plot non-interactively from what came back
  plot_regression(pts)

  # Or replot them in the gadget and keep clicking
  interactive_regression(pts)

  # Providing coordinates beforehand
  points <- data.frame(x = c(1, 4, 7), y = c(2, 5, 8))
  interactive_regression(points)
}
```

---

interactive_sampling *compstatslib interactive_sampling() function*

Description

Interactive visualization to sample from a population and see how a given sampling statistic (theta) is distributed.

Usage

```
interactive_sampling(population, sample_size = 10, theta = mean)
```

Arguments

population	A vector of values following any population distribution you wish to simulate.
sample_size	The size of each sample to draw from the population
theta	The function that computes the statistic of interest from each sample (e.g., mean or median).

Details

Use the controls in the viewer to draw more samples or change simulation parameters. Click "Done" to return results to the console.

The printed call shows the expressions you passed for population and theta rather than their values, so capture the population in a variable first if you want a pasteable call that does not re-simulate it.

Value

On "Done", a compstatslib_args object: a named list of population, sample_size, theta and reps as last set, which prints the plot_sampling call that reproduces the plot. It is still an ordinary list, so do.call(plot_sampling, result) works. The accumulated sampling cache rides along as attr(result, "vars") rather than appearing in the printed call — pass it as plot_sampling(vars =) to keep building on the same sampling distribution instead of starting fresh. On "Cancel", NULL. See compstatslib-reproduce.

See Also

plot_sampling

Examples

```
if (interactive()) {
  my_pop <- rnorm(100000)
  result <- interactive_sampling(my_pop)

  # Reproduce the plot non-interactively (draws fresh samples)
  do.call(plot_sampling, result)

  # Or continue from the samples already accumulated in the gadget
  do.call(plot_sampling, c(result, list(vars = attr(result, "vars"))))

  bimodal <- c(rnorm(100000, mean = 4), rnorm(100000, mean = -4))
  interactive_sampling(bimodal, theta = median)
}
```

---

interactive_scatter3d *compstatslib interactive_scatter3d() function*

Description

Interactive 3D scatterplot. Shiny gadget wrapping plot_scatter3d with column pickers (x / y / z / color) and display-control sliders (aspect ratio per axis, marker opacity, marker size). The plot re-renders live as inputs change. On Done, the equivalent plot_scatter3d() call is printed to the console (copy-pasteable into a script) and the chosen arguments are returned invisibly so that do.call(plot_scatter3d, result) (or do.call(interactive_scatter3d, result) to resume a session) reproduces the same plot. On Cancel, returns NULL.

Usage

```
interactive_scatter3d(
  data = moderation_data,
  x = NULL,
  y = NULL,
  z = NULL,
  color = NULL,
```

```
  aspect = c(1, 1, 1),
  opacity = 0.8,
  size = 5,
  camera = NULL,
  titles = NULL,
  ...
)
```

Arguments

`data`	A data frame containing at least three numeric columns. Defaults to the bundled `moderation_data`.
`x`, `y`, `z`	Optional character. Names of numeric columns of `data` to pre-select in the axis pickers. When `NULL` (default), the first three numeric columns are used.
`color`	Optional character. Name of any column of `data` to pre-select in the color picker, or `NULL` (default) for uniform color.
`aspect`, `opacity`, `size`	Initial values for the aspect-ratio, opacity, and marker-size sliders. Same defaults and validation as `plot_scatter3d`.
`camera`	Optional list giving the initial 3D camera position. Useful for resuming a session at a previously captured view. `NULL` (default) uses plotly's default view.
`titles`	Optional named list / character vector with custom axis titles (recognized names: `x`, `y`, `z`). Threaded through every render and included in the Done output.
`...`	Further arguments forwarded to `plot_scatter3d` on every render. Not echoed in the printed Done call.

Details

Every `plot_scatter3d()` argument is accepted here as a starting value — column choices, aspect ratio, opacity, marker size, camera position, and axis titles. Pickers and sliders are pre-positioned accordingly; the user can change them mid-session (the title arg has no in-gadget UI but threads through every render and into the Done output).

Only numeric columns of `data` appear in the x / y / z pickers (matching `plot_scatter3d`'s numeric-axis contract); the color picker offers all columns plus a `"(none)"` sentinel. Errors before launching if `data` has fewer than three numeric columns, or if any supplied initial `x`/`y`/`z` is not a numeric column, or if initial `color` is not in `names(data)`.

User-driven rotation and zoom are preserved across slider / picker re-renders within the gadget. The current camera state is captured via the `plotly_relayout` event and re-passed into each render so structural changes (e.g. toggling color) don't reset the view. If the user rotated or zoomed (or a starting `camera` was supplied), the camera is included in the Done output so the reproduced `plot_scatter3d()` call opens at the same view — useful for nailing down a specific angle for an Rmd / report.

The printed Done call uses `deparse(substitute(data))` captured at gadget entry, so it shows the user's variable name (e.g. `plot_scatter3d(data = my_df, ...)`). Args still at their `plot_scatter3d()` defaults are omitted from the printed call. Caveat: inline expressions are reproduced literally — calling `interactive_scatter3d(read.csv("x.csv"))` prints `plot_scatter3d(data = read.csv("x.csv"), ...)` which re-runs the read when pasted; capture the data in a variable first to avoid that.

This function only works in IDEs that support Shiny gadgets (RStudio, Positron, etc.).

Value

On "Done", a `compstatslib_args` object: a named list of `data`, `x`, `y`, `z`, `color`, `aspect`, `opacity`, `size`, `camera` and `titles`. It prints the `plot_scatter3d` call that reproduces the plot, and is still an ordinary list, so `do.call(plot_scatter3d, result)` works. `camera` is `NULL` unless the user rotated or zoomed during the session (or supplied an initial camera). On "Cancel", `NULL`. See compstatslib-reproduce.

See Also

`plot_scatter3d` for the non-interactive version; `moderation_data` for the bundled teaching dataset.

Examples

```
if (interactive()) {
  # Default synthetic example
  interactive_scatter3d()

  # User's own data
  set.seed(1)
  my_df <- data.frame(
    score = rnorm(100),
    time  = rnorm(100),
    dose  = rnorm(100),
    group = factor(sample(c("ctrl", "treat"), 100, replace = TRUE))
  )
  interactive_scatter3d(my_df)

  # Pre-position columns and starting display
  interactive_scatter3d(my_df,
                        x = "score", y = "time", z = "dose",
                        color = "group",
                        aspect = c(1, 1.5, 1),
                        titles = list(x = "Score", y = "Time",
                                      z = "Dose"))

  # Resume a previous session from its returned state
  result <- interactive_scatter3d(my_df)
  do.call(interactive_scatter3d, result)

  # Or finalize the captured view as a non-interactive plot
  do.call(plot_scatter3d, result)
}
```

`interactive_t_test` *compstatslib interactive_t_test() function*

Description

Interactive visualization that allows one to adjust the parameters affecting hypothesis testing in order to see how their variation influences the null t and alternative t distributions, and statistical power.

Usage

```
interactive_t_test(
  diff = 0.5,
  sd = 4,
  n = 100,
  alpha = 0.05,
  error_matrix = FALSE
)
```

Arguments

`diff`	The initial difference between the null and alternative means.
`sd`	The initial standard deviation of the population.
`n`	The initial sample size.
`alpha`	The initial significance level.
`error_matrix`	Logical; whether to show the error matrix initially.

Details

Use the sliders in the viewer to adjust parameters. The movement of the alternative t-statistics distribution with respect to the null distribution will be visible, as well as the consequent change in statistical power. Click "Done" to close.

All five arguments match `plot_t_test`'s, so a gadget session can be resumed from a previous result.

Value

On "Done", a `compstatslib_args` object: a named list of `diff`, `sd`, `n`, `alpha` and `error_matrix` at their final slider positions, which prints the `plot_t_test` call that reproduces the plot. It is still an ordinary list, so `do.call(plot_t_test, result)` works. On "Cancel", `NULL`. See compstatslib-reproduce.

See Also

`plot_t_test`

Examples

```
if (interactive()) {
  # Move the sliders, then Done
  result <- interactive_t_test()

  # Reproduce the plot non-interactively
  do.call(plot_t_test, result)

  # Or launch pre-configured for a small, noisy study
  interactive_t_test(diff = 0.2, sd = 5, n = 30)
}
```

`machine_precision` *compstatslib machine_precision() function*

Description

Code function that reports the smallest possible number on the user's machine such that 1 + x != 1.

Usage

```
machine_precision()
```

Value

A numeric value representing the smallest possible number that the user's computer can effectively represent.

Examples

```
machine_precision()

# Anything smaller is swallowed by rounding
1 + machine_precision() != 1
1 + machine_precision() / 2 != 1
```

`moderation_data` *Synthetic moderation dataset*

Description

A bundled synthetic dataset calibrated to show a clear interaction (moderation) effect when visualized as a 3D wireframe surface. Used as the default `data` argument for `plot_moderation_3d` and `interactive_moderation_3d`, so that calling either function with no arguments produces a working classroom demonstration.

Usage

```
moderation_data
```

Format

A data frame with 200 rows and 4 numeric columns:

y Outcome (dependent variable).

x Independent variable.

z Moderator.

w Unrelated noise predictor — drawn independently of `x`, `z`, and `y`. Any correlation with `y` is spurious. Useful for demonstrating that irrelevant predictors do not change the IV/mod surface and for exercising the multi-predictor `iv`/`mod` arguments of `plot_moderation_3d`.

Details

Generated as `y = 0.5*x + 0.3*z + 0.8*(x*z) + N(0, 1)` with `x, z, w ~ N(0, 2)` drawn independently, `n = 200`, and `set.seed(42)`.

Variable names follow the moderation-literature convention (Aiken & West) where `y` is the outcome, `x` is the independent variable, and `z` is the moderator. Note that in 3D wireframe plots the *vertical* plot axis depicts the outcome `y` regardless of variable naming.

Source

Generated by `data-raw/moderation_data.R`. Purely synthetic.

Examples

```
head(moderation_data)
summary(lm(y ~ x * z, data = moderation_data))
summary(lm(y ~ w, data = moderation_data))  # w is noise — no real effect
```

pca_degenerate *Degenerate PCA demo dataset*

Description

A small two-column dataset used to demonstrate a degenerate PCA case where the variance is concentrated almost entirely along one axis, producing a near-singular principal component structure.

Usage

```
pca_degenerate
```

Format

A data frame with 16 rows and 2 numeric columns:

x Horizontal coordinate.

y Vertical coordinate.

Examples

```
head(pca_degenerate)
```

plot_logit *compstatslib plot_logit() function*

Description

Non-interactive visualization function that plots given points, logistic regression line and relevant statistics.

Usage

```
plot_logit(
  points,
  formula = y ~ x,
  regression = TRUE,
  stats = TRUE,
  min_x = 0,
  max_x = 1,
  legend_loc = "topleft"
)
```

Arguments

points	A dataframe of *x* and *y* coordinates to plot.
formula	A formula to use in estimating logit (e.g., y ~ x).
regression	Logical parameter of whether to plot a regression line (TRUE by default)
stats	Logical parameter of whether to display relevant statistics on the plot area (TRUE by default). The following parameters are provided on the plot area: **Intercept** The intercept of the logistic regression. **Coefficient** The coefficient of the independent variable. **AIC** An information theoretic fit criterion.
min_x	Minimum x value for the plot range.
max_x	Maximum x value for the plot range.
legend_loc	Position of the statistics legend on the plot (e.g., "topleft", "bottomright").

Value

A dataframe containing the points coordinates.

See Also

interactive_regression

Examples

```
mydata <- data.frame(iv = c(-6, -3, 1, 3, 5, 8), dv = c(0, 0, 0, 1, 1 ,1))
plot_logit(mydata, formula = dv ~ iv)
```

---

plot_matrix_inverse *compstatslib plot_matrix_inverse() function*

Description

Non-interactive plotting function that helps visualize an inverse.

Usage

```
plot_matrix_inverse(x1, y1, x2, y2)
```

Arguments

x1	The first row (or column) vector of matrix A.
y1	The second row (or column) vector of matrix A.
x2	The first row (or column) vector of the inverse matrix A^(-1).
y2	The second row (or column) vector of the inverse matrix A^(-1).

Details

The user can choose the magnitude of the vectors, which will be graphically represented by the function.

Value

No return value, called for side effects (plots to the active device).

See Also

interactive_matrix_inverse

Examples

```
# A matrix and its inverse: the two parallelograms have reciprocal areas
plot_matrix_inverse(1, 2, 2, 1)

# A near-singular matrix stretches its inverse dramatically
plot_matrix_inverse(1, 0.9, 0.9, 1)
```

---

plot_moderation_3d *compstatslib plot_moderation_3d() function*

Description

Non-interactive visualization of a moderated regression. Fits an `lm()` from the supplied formula and data, predicts the outcome over a regular grid of the IV and moderator (any other predictors held at typical values), and renders the fitted surface as a 3D wireframe with a height-mapped color gradient.

Usage

```
plot_moderation_3d(
  formula = y ~ x * z,
  data = moderation_data,
  iv = NULL,
  mod = NULL,
  z_rot = 40,
  x_rot = -70,
  zlim = NULL,
  ...
)
```

Arguments

formula	A model formula. The first variable on the right-hand side is treated as the IV and the second as the moderator when the formula has exactly two predictors. With three or more predictors, you must specify iv and mod explicitly. Any formula structure R supports works (x * z, x + z + x:z, x:z, transformations, etc.).
data	A data frame containing every variable named in formula. Defaults to the bundled moderation_data.
iv	Optional character. Name of the predictor to place on the first horizontal axis. Required when formula has more than two predictors.
mod	Optional character. Name of the predictor to place on the second horizontal axis. Required when formula has more than two predictors.
z_rot	Numeric. Rotation about the vertical axis, in degrees (lattice's screen$z). Default 40. Try 0 to align the IV slope plane with the screen, or 270 for the moderator slope plane.
x_rot	Numeric. Rotation about the horizontal axis, in degrees (lattice's screen$x). Default -70.
zlim	Numeric length-2 vector. Vertical-axis range. Default NULL, which uses the union of the actual outcome's range in data and the fitted-surface range over the prediction grid.
...	Further arguments forwarded to wireframe.

Details

The user's formula is fitted as-is via lm(), so any structure R's formula language supports works. predict() then evaluates the fitted model at every point of an expand.grid() over the IV and moderator ranges (15 steps each). Predictors other than the IV and moderator are held at typical values: mean() for numeric variables, the first level for factors, the lexicographically first value for character variables, and FALSE for logicals. This is a pedagogical simplification; if you need other hold-out values, fit lm() yourself and slice the surface manually.

The *vertical* axis of the rendered plot depicts the outcome (DV) regardless of variable naming. With the bundled moderation_data the moderator is named z, but z appears on a horizontal plot axis, not the vertical "z-axis" of the wireframe.

When the formula contains predictors beyond iv and mod, a one-line message() is emitted noting that other predictors are held at their typical values. Suppress with suppressMessages().

iv and mod must be numeric. Categorical (factor or character) variables on the plot axes are not supported and will raise an error.

Value

A trellis object (lattice surface). Print or assign to render.

See Also

interactive_moderation_3d for the rotatable Shiny gadget version; moderation_data for the bundled teaching dataset.

Examples

```
# Default synthetic example
plot_moderation_3d()

# Additive model — flat surface, no twist
plot_moderation_3d(y ~ x + z, moderation_data)

# User's own data and naming
set.seed(1)
my_df <- data.frame(
  score = rnorm(100),
  time  = rnorm(100),
  dose  = rnorm(100)
)
plot_moderation_3d(score ~ time * dose, my_df)

# Multi-predictor model — `w` is bundled noise, included as a
# control. Pick which two predictors go on the plot axes; the rest
# are held at typical values (here, `mean(w)`).
plot_moderation_3d(y ~ x + z + w + x:z, moderation_data,
                   iv = "x", mod = "z")
```

---

plot_pca *compstatslib plot_pca() function*

Description

Plots data points and their principal component vectors.

Usage

```
plot_pca(
  points = data.frame(),
  meancenter = TRUE,
  xlim = c(-50, 50),
  ylim = c(-50, 50)
)
```

Arguments

points	A dataframe with x and y columns containing the point coordinates.
meancenter	Logical; if TRUE (default), points are mean-centered before computing PCA.
xlim	The x-axis limits for the plot.
ylim	The y-axis limits for the plot.

Value

If 3 or more points are provided, invisibly returns the prcomp result. Otherwise returns NULL invisibly.

See Also

interactive_pca

Examples

```
points <- data.frame(x = c(-20, -5, 5, 15, 30), y = c(-15, -10, 5, 10, 25))
plot_pca(points)

# Without mean-centering, the components are anchored at the origin
plot_pca(points, meancenter = FALSE)
```

---

plot_regression *compstatslib plot_regression() function*

Description

Non-interactive visualization function that plots given points, regression line and relevant statistics.

Usage

```
plot_regression(points, regression = TRUE, stats = TRUE)
```

Arguments

points	A dataframe of *x* and *y* coordinates to plot.
regression	A logical parameter that plots a regression line when set to TRUE and hides it when set to FALSE. It is set to TRUE by default.
stats	A logical parameter that displays the relevant statistics on the plot area when set to TRUE; it hides them when set to FALSE. It is set to TRUE by default.

Value

A dataframe containing the points coordinates. Additionally, the following parameters are provided on the plot area:

Raw intercept	The y-coordinate at which the regression line crosses the y-axis.
Raw slope	The value of the slope parameter.
Correlation	The strength of the linear relationship.
SSR	The sum of squares regression.
SSE	The sum of squares error.
SST	The sum of squares total.
R-squared	The multiple coefficient of determination.

See Also

interactive_regression

Examples

```
points <- data.frame(x = c(1, 3, 5, 8), y = c(2, 4, 6 ,8))

plot_regression(points)
```

---

plot_sample_ci *compstatslib plot_sample_ci() function*

Description

Visualize the confidence intervals of samples drawn from a population.

Usage

```
plot_sample_ci(
  num_samples = 100,
  sample_size = 100,
  pop_size = 10000,
  distr_func = rnorm,
  ...
)
```

Arguments

num_samples	Number of the samples to draw from the population.
sample_size	Size of each sample drawn from the population.
pop_size	Size of the overall population to simulate.
distr_func	Random data generation function to simulate the population data (e.g., rnorm or runif).
...	Parameters to pass to the distr_func function specified.

Value

No return value, called for side effects (plots to the active device).

Examples

```
plot_sample_ci(sample_size=300, distr_func=rnorm, mean=50, sd=10)
plot_sample_ci(sample_size=300, distr_func=runif, min=17, max=35)
```

plot_sampling *compstatslib plot_sampling() function*

Description

Plot distribution of a population, samples drawn from the population, given sampling statistic.

Usage

```
plot_sampling(
  population,
  sample_size,
  theta,
  reps = 1,
  vars = NULL,
  replot_population = TRUE
)
```

Arguments

population	A vector of values following any population distribution you wish to simulate.
sample_size	The size of each sample to draw from the population
theta	The function that computes the statistic of interest from each sample (e.g., mean or median).
reps	Number of repeated samples to draw on each call.
vars	Optional cached state from a previous call, used by interactive_sampling() to accumulate samples across draws.
replot_population	Logical; whether to redraw the population panel.

Value

A list containing the population data, sample statistics, and other information of interest

Examples

```
plot_sampling(rnorm(100000), sample_size = 100, reps = 50, theta = median)
```

plot_scatter3d *compstatslib plot_scatter3d() function*

Description

Non-interactive 3D scatterplot. Renders three numeric columns of a data frame as a rotatable point cloud, with an optional fourth column mapped to marker color.

Usage

```
plot_scatter3d(
  data = moderation_data,
  x = NULL,
  y = NULL,
  z = NULL,
  color = NULL,
  aspect = c(1, 1, 1),
  opacity = 0.8,
  size = 5,
  camera = NULL,
  titles = NULL,
  ...
)
```

Arguments

data	A data frame. Defaults to the bundled moderation_data.
x, y, z	Optional character. Names of numeric columns of data to place on each axis. When all three are NULL, the first three numeric columns of data are used; if data has more than three numeric columns, a one-line message() reports the chosen and skipped columns.
color	Optional character. Name of any column of data to map to marker color. Numeric columns yield a continuous color scale; factor / character / logical columns yield a discrete palette with a legend. NULL (default) draws a uniform color.
aspect	Length-3 positive numeric vector giving the x / y / z aspect ratio (forwarded to plotly's scene$aspectratio). Default c(1, 1, 1).
opacity	Single numeric in (0, 1]. Marker opacity. Default 0.8.
size	Single positive numeric. Marker size. Default 5.
camera	Optional list giving the initial 3D camera position (plotly's scene$camera). Typically a list with eye, center, and/or up, each itself a list with numeric x, y, z components. Useful for reproducing a specific rotation / zoom captured from interactive_scatter3d. NULL (default) uses plotly's default view. The plot also sets a constant uirevision so user-driven rotation / zoom is preserved across re-renders.

titles	Optional named character vector / list giving custom axis titles. Recognized names are x, y, z. Unsupplied axes fall back to the source column name.
...	Further arguments forwarded to plot_ly.

Details

Axis types (x / y / z) must be numeric. Factor, character, logical, and Date / POSIXct columns are rejected with a clear error pointing to color for categorical separation.

Value

A plotly htmlwidget. Print or assign to render.

See Also

interactive_scatter3d for a Shiny gadget that wraps this function with live column pickers and display sliders.

Examples

```
# Default synthetic example — `moderation_data` has 4 numeric cols,
# so a one-line message notes which 3 were picked.
plot_scatter3d()

# Explicit columns, no color
plot_scatter3d(moderation_data, x = "x", y = "z", z = "y")

# Map a fourth column to color
plot_scatter3d(moderation_data, x = "x", y = "z", z = "y", color = "w")

# Stretched aspect, thinner larger markers, custom titles
plot_scatter3d(moderation_data,
               x = "x", y = "z", z = "y",
               aspect = c(1, 2, 4),
               opacity = 0.3, size = 10,
               titles = list(x = "IV", y = "Moderator", z = "Outcome"))
```

---

plot_t_test *compstatslib plot_t_test() function*

Description

Non-interactive visualization function that plots null and alternative t distributions of a t-test. Shows rejection zone and power as areas under the curves.

Usage

```
plot_t_test(diff = 0.5, sd = 4, n = 100, alpha = 0.05, error_matrix = FALSE)
```

Arguments

diff	The test difference (defaults to 0.5).
sd	Population standard deviation (defaults to 4).
n	Sample size (defaults to 100).
alpha	Significance level (defaults to 0.05).
error_matrix	Whether to display type I/II error matrix (defaults to FALSE).

Value

No return value, called for side effects (plots to the active device).

See Also

interactive_t_test

Examples

```
plot_t_test()
plot_t_test(diff=-0.1, sd=3)
```

Index