Package: jstats (via r-universe)

July 1, 2026

Title Simplified Statistical Analysis Tools for Social Science

Version 0.9.96

Description Provides simplified tools for common statistical analyses used in social science research and teaching, with output formatted in the style of SPSS and Stata. Analysis functions include descriptive statistics, frequency tables, t-tests, ANOVA, correlations, chi-square tests, cross-tabulations, linear regression, logistic regression, and Cronbach's alpha, with built-in model diagnostics and publication-oriented plotting. Supporting tools handle data management (recoding, labelling, filtering, dummy coding, scale construction), automatic detection and handling of coded missing values, and unified data import and export for SPSS, Stata, SAS, Excel, CSV, and R formats. Functions accept unquoted variable names and formula syntax for ease of use, handle haven-labelled variables automatically, and return results invisibly while printing user-friendly tables. The package stays close enough to base R conventions that users learn transferable skills rather than a private dialect, supporting a smooth transition from SPSS, Stata, or SAS into the broader R ecosystem.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.2.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports ggplot2, haven (>= 2.3.0), labelled, rlang, readxl, writexl, callr

Suggests generics

LazyData true

URL https://jma61.github.io/jstats/

Config/pak/sysreqs make libicu-dev libx11-dev zlib1g-dev

Repository https://jma61.r-universe.dev

Date/Publication 2026-07-01 08:52:43 UTC

RemoteUrl https://github.com/JMA61/jstats

RemoteRef HEAD

RemoteSha 108257e7ffb54795bbe47c92ee7968e15ca4a1d4

Contents

`clinic` *Clinic intervention example dataset*

Description

A small synthetic mental-health and wellbeing intervention sample, used as the package's messy-data companion to `community`. Where `community` is the clean default, `clinic` deliberately carries undeclared missing-value codes, a column whose value labels were stripped on import, and an imperfect scale item, so the declare-and-clean workflow has realistic material to work on. The 70 clients and 16 variables echo the teaching structure of `community` – an interaction, a null variable, non-overlapping missingness, a recode dichotomy, a clean logistic outcome, a multi-category variable, and a Likert battery – in a psychology setting. The data are synthetic, but the relationships among the variables are realistic.

Usage

```
clinic
```

Format

A data frame with 70 rows and 16 variables:

ClientID Client ID, character ("C001", "C002", ...).

Stress Perceived stress (integer, 0-40). Carries SPSS-style missing values (-99 Refused, -98 Don't know).

SocialSupport Perceived social support (integer, 0-24); the buffering partner in the Stress-by-SocialSupport interaction on Flourishing.

SleepHours Average nightly sleep in hours. Carries SPSS-style missing values (-99 Refused, -98 Don't know), placed on cases that do not overlap the Stress codes, so a model using both predictors drops more cases than either alone.

Flourishing Flourishing score (integer, 0-100); built with a Stress-by-SocialSupport interaction (the buffering hypothesis).

ScreenTime Daily screen time in hours; deliberately near-independent of the other variables.

PriorTherapy Received therapy before the study, dichotomy coded 1/2 (1 Yes, 2 No); recode to 0/1 before use as a logistic-regression outcome.

SoughtHelp Sought professional help during the study, dichotomy coded 0/1 (0 No, 1 Yes). A clean logistic-regression outcome.

Medication Currently taking medication, dichotomy coded 0/1 (0 No, 1 Yes). Carries an SPSS-style missing value (-99 Refused).

Condition Treatment condition, 4 categories (1 Control, 2 CBT, 3 Mindfulness, 4 Support group); has a modest effect on Flourishing.

MoodRating Mood rating (integer, 1-10). Arrives "dirty": literal -99 (Refused) and -98 (Don't know) codes are present in the data with NO missing-value declaration, the state of play after a CSV or Excel import. The package's jdeclare_udm() demonstration variable: summary statistics are poisoned until the codes are declared.

Anxiety1 "I felt calm and relaxed." 5-point Likert (1 Not at all to 5 Extremely); reverse-keyed (the variable label ends in " R"). Reverse-code before scale scoring.

Anxiety2 "I worried about many different things." 5-point Likert. Arrives with literal -99/-98 codes present in the data and NO missing-value declaration – the undeclared contrast to Anxiety4.

Anxiety3 "I felt afraid for no clear reason." 5-point Likert; arrives with its value labels stripped (a plain numeric column, as after a CSV import that dropped the labels).

Anxiety4 "I had trouble controlling my worry." 5-point Likert. Carries properly declared SPSS-style missing values (-99 Refused, -98 Don't know) – the declared contrast to Anxiety2.

Anxiety5 "I felt restless or on edge." 5-point Likert; weakly loaded (a Cronbach's-alpha drop candidate when scoring the scale).

Details

The five Anxiety items form a single scale, with one deliberate problem per item: Anxiety1 is reverse-keyed; Anxiety2 carries literal -99/-98 codes that are not declared as missing; Anxiety3 arrives with its value labels stripped; Anxiety4 carries the same -99/-98 codes but properly declared (the clean contrast to Anxiety2); and Anxiety5 is the weak item that scale-reliability output flags for dropping. Stress and SleepHours carry SPSS-style missing values on non-overlapping cases, so listwise deletion across both reduces the analysis sample below the per-variable counts. MoodRating and Anxiety2 are the two columns whose -99/-98 codes arrive undeclared, awaiting `jdeclare_udm()`. The Stress-by-SocialSupport interaction on Flourishing is the buffering hypothesis (higher social support weakens the negative association between stress and flourishing), and treatment Condition has a modest effect on Flourishing.

Source

Synthetic data generated by `data-raw/clinic_data_generator.R` (random seed 20260614).

See Also

`community`, the clean default example dataset.

`community` *Community survey example dataset*

Description

A small synthetic survey dataset used throughout the package as a runnable example. It backs the function examples, serves as a teaching dataset for new users, and demonstrates cross-platform save and load behavior. The 100 respondents and 15 variables are chosen to exercise the kinds of data social-science users actually have: Likert scales, dichotomies, a multi-category variable, continuous measures, and SPSS-style user-defined missing values. The data are synthetic, but the relationships among the variables are realistic.

Usage

```
community
```

Format

A data frame with 100 rows and 15 variables:

RespondentID Respondent ID, character ("R001", "R002", ...).

Income Annual income (USD). Carries SPSS-style missing values (-99 Refused, -98 Don't know).

Education Highest education level, 5 categories (1 Some high school, 2 High school graduate, 3 Some college, 4 Bachelor's degree, 5 Graduate degree). Carries SPSS-style missing values (-99 Refused, -98 Don't know).

Age Age in years (integer, 18-80).

WellbeingScore Flourishing score (integer, 0-100); built with an Income-by-Age interaction.

Volunteer Volunteered in past year, dichotomy coded 0/1 (0 No, 1 Yes).

OwnsHome Owns home, dichotomy coded 1/2 (1 Yes, 2 No); recode to 0/1 before use as a logistic-regression outcome.

Smoker Current smoker, dichotomy coded 0/1 (0 No, 1 Yes). Carries an SPSS-style missing value (-99 Refused).

CommuteTime Daily commute time in minutes (integer); deliberately near-independent of the other variables.

Region Region of residence, 4 categories (1 North, 2 South, 3 East, 4 West).

Environment1 "Climate change is a serious threat." 5-point Likert (1 Strongly Disagree to 5 Strongly Agree). Carries SPSS-style missing values (-99 Refused, -98 Don't know).

Environment2 "Concern about the environment is exaggerated." 5-point Likert; reverse-keyed (the variable label ends in " R"). Reverse-code before scale scoring.

Environment3 "Government should do more for the environment." 5-point Likert. Carries SPSS-style missing values (-99 Refused, -98 Don't know).

Environment4 "I would pay more for environmentally friendly products." 5-point Likert.

Environment5 "Pollution is a major cause of public health problems." 5-point Likert; weakly loaded (a Cronbach's-alpha drop candidate when scoring the scale).

Details

`community` is the clean default example dataset. For a companion dataset that deliberately carries undeclared missing-value codes, stripped value labels, and an imperfect scale – the material the data-cleaning workflow operates on – see `clinic`.

The five Environment items form a single attitude scale: item 2 is reverse-keyed, and item 5 is the weak item that scale-reliability output flags for dropping. Income, Education, Smoker, Environment1, and Environment3 carry SPSS-style missing values, with the codes placed on partly non-overlapping cases so that listwise deletion visibly reduces the analysis sample below the per-variable counts. All of community's missing-value codes are properly declared; for a dataset with undeclared codes and other deliberate data-cleaning problems, see `clinic`.

Source

Synthetic data generated by `data-raw/community_data_generator.R` (random seed 20260605).

See Also

clinic, the messy-data companion dataset.

---

jalpha *Cronbach's Alpha Reliability Analysis*

Description

Computes Cronbach's alpha and prints SPSS-style reliability output including a case processing summary, overall alpha, item statistics, and item-total statistics with alpha-if-item-deleted. Built from scratch with no external package dependencies beyond base R. Handles haven-labelled variables automatically. Detects potentially reverse-coded or misfit items.

Usage

```
jalpha(
  data,
  ...,
  subset = NULL,
  variable.id = NULL,
  value.id = NULL,
  case.processing.detail = NULL,
  digits = NULL
)
```

Arguments

data	A data frame.
...	Unquoted variable names (scale items) within data. Use colon notation (e.g. Item1:Item6) to select a range of consecutive columns.
subset	An optional unquoted logical expression (e.g. Group == 1) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
variable.id	Character or NULL. Variable label display mode: one of "both", "names", "labels", "legend", or "legend.bottom". "names" shows variable names only; "both" shows "name: label"; "labels" shows each item's label in the Item column of the Item Statistics and Item-Total Statistics tables (best for short labels; the returned tables and the reverse-coding diagnostic keep variable names); "legend"/"legend.bottom" keep names and print a label legend after the final table. NULL (default) defers to joutput()'s variable.id setting. Not a logical.
value.id	Not supported by jalpha(). The function does not display value labels, so passing this argument is an error. It exists only to return a clear message rather than misreporting the token as a missing variable. Leave at NULL (default).

`case.processing.detail`
: Per-call override of the Case Processing Summary detail tier: one of `"none"`, `"totals"`, or `"per_code"`. `NULL` (default) uses the active `joutput()` level default.

`digits`
: Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

Details

A red "Reliability Analysis" title is printed first, followed by the case processing summary, overall alpha, item statistics, and item-total statistics.

Value

Invisibly returns a list of class `jst_alpha` containing: `alpha` (Cronbach's alpha), `n_items`, `n_used`, `n_excluded`, `item_statistics`, `item_total_statistics`, and `sample_info` (pipeline and missing data counts). item statistics data frame, and item-total statistics data frame.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jalpha(community, Environment1, Environment2, Environment3,
       Environment4, Environment5)

# Using juse() default
juse(community)
jalpha(Environment1, Environment2, Environment3, Environment4,
       Environment5)
```

---

jaov *One-way ANOVA (traditional or Welch method)*

Description

Runs a one-way ANOVA and prints a formatted group descriptives table followed by an ANOVA table. By default, runs the traditional ANOVA assuming equal variances. Optional parameters provide post-hoc tests, effect size, Levene's test, and confidence intervals. Set welch = TRUE for the Welch correction when equal variances cannot be assumed. Handles haven-labelled, numeric, and factor grouping variables. For haven-labelled variables, numeric codes are displayed alongside labels in the group descriptives table.

Usage

```
jaov(
  formula,
  data,
  welch = FALSE,
  posthoc = NULL,
  effect.size = NULL,
  levene = NULL,
  ci = NULL,
  subset = NULL,
  variable.id = NULL,
  value.id = NULL,
  case.processing.detail = NULL,
  full = FALSE,
  digits = NULL
)
```

Arguments

formula	A formula of the form DV ~ Group.
data	A data frame containing variables referenced in formula.
welch	Logical. If FALSE (default), runs traditional ANOVA. If TRUE, runs Welch's ANOVA (does not assume equal variances).
posthoc	Logical or NULL. If TRUE, prints Tukey HSD pairwise comparisons. Not available when welch = TRUE. If NULL (default), defers to joutput().
effect.size	Logical or NULL. If TRUE, prints eta-squared. If NULL (default), defers to joutput().
levene	Logical or NULL. If TRUE, prints Levene's test for homogeneity of variance. If NULL (default), defers to joutput().
ci	Logical or NULL. If TRUE, adds 95% confidence intervals to the group descriptives table. If NULL (default), defers to joutput().
subset	An optional unquoted logical expression (e.g. Group == 1) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
variable.id	Character or NULL. Variable label display mode: one of "both", "names", "labels", "legend", or "legend.bottom". "names" shows variable names only; "both" shows "name: label"; "labels" shows the DV and grouping-variable labels wherever the variable name appears (table captions and the ANOVA Source row; group levels follow the value.id mode) – best for short labels; "legend"/"legend.bottom" keep names and print a label legend after the output. NULL (default) defers to joutput()'s variable.id setting. Not a logical.
value.id	Character or NULL. Value-label display mode for the group descriptives rows: "both" ("code: label"), "values" (bare code), or "labels" (the label, degrading to the bare code where a code has none). "legend" and "legend.bottom" keep the bare code in the table and print a value-label legend after it ("legend"

per-table, "legend.bottom" consolidated where multiple tables are produced). A no-op for grouping variables with no value labels. NULL (default) defers to joutput()'s value.id setting. Not a logical.

case.processing.detail
: Per-call override of the Case Processing Summary detail tier: one of "none", "totals", or "per_code". NULL (default) uses the active joutput() level default.

full
: Logical. If TRUE, turns on posthoc, effect.size, levene, and ci all at once. Does not override explicit FALSE values.

digits
: Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; digits = 0 prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to joutput()'s digits setting (default 3).

Details

A red title identifying the test type is printed first, followed by variable labels (if present), then the results tables.

Value

Invisibly returns a list of class jst_anova containing: model (the aov or oneway.test object), model_frame (the analysis data frame used for plotting), test_type, formula, descriptives, f, df1, df2, p, eta_squared, n, and sample_info (pipeline and missing data counts).

See Also

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jaov(WellbeingScore ~ Region, data = community)
jaov(WellbeingScore ~ Region, data = community, welch = TRUE)
jaov(WellbeingScore ~ Region, data = community, full = TRUE)

# Using juse() default
juse(community)
jaov(WellbeingScore ~ Region)
jaov(WellbeingScore ~ Region, full = TRUE)
```

javg *Compute a row-wise mean across multiple variables*

Description

`javg()` computes the mean of values across multiple variables for each case (row) in the data frame. This is typically used to create scale means from a set of related items.

By default, cases with any missing values receive `NA`. Use the `min.valid` argument to allow partial means — for example, `min.valid = 1` computes the mean of available values as long as at least one item is non-missing.

By default, the denominator is the number of non-missing values for each case. Use `fixed = TRUE` to always divide by the total number of variables regardless of missing values.

Variables can be listed individually or using colon notation to select a range of consecutive columns (e.g. `Attitude1:Attitude6`).

Usage

```
javg(data, ..., min.valid = NULL, fixed = FALSE, var.label = NULL)
```

Arguments

`data`	A data frame, or omit to use the `juse()` default.
`...`	Unquoted variable names. Use colon notation (e.g. `Attitude1:Attitude6`) to select a range of consecutive columns.
`min.valid`	Integer (optional). The minimum number of non-missing values required to compute a mean. If a case has fewer non-missing values, the result is `NA`. If omitted, all values must be non-missing (equivalent to setting min.valid to the number of variables).
`fixed`	Logical. If `FALSE` (default), the denominator for each case is the number of non-missing values (i.e. the mean adjusts for missing data). If `TRUE`, the denominator is always the total number of variables (i.e. missing values effectively count as zero).
`var.label`	Character string (optional). A variable label to attach to the result. If omitted, an auto-generated label is used.

Value

A numeric vector the same length as `nrow(data)`, suitable for assigning to a new column: `MyData$ScaleMean <- javg(Var1, Var2, Var3)`.

See Also

`jsum` for computing row-wise sums.

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# Set the default data frame (so you can omit it in function calls)
juse(community)

# Mean of three variables (all must be non-missing)
community$EnvAvg <- javg(Environment1, Environment3, Environment4)

# Mean with partial data allowed (at least 2 non-missing)
community$EnvAvg <- javg(Environment1, Environment3, Environment4,
                         min.valid = 2)

# Mean using colon range for consecutive columns
community$ScaleMean <- javg(Environment1:Environment5)

# Mix colon ranges and explicit names (e.g. after reverse-coding an item)
community$Environment2R <- jrecode(community, Environment2,
                                   map = "1=5; 2=4; 3=3; 4=2; 5=1")
community$ScaleMean <- javg(Environment1, Environment2R,
                            Environment3:Environment5)

# Fixed denominator (always divide by total number of variables)
community$EnvAvg <- javg(Environment1, Environment3, Environment4,
                         min.valid = 2, fixed = TRUE)

# With a custom variable label
community$ScaleMean <- javg(Environment1:Environment5,
                            var.label = "Environment Scale Mean")

# With an explicit data frame (instead of using juse default)
community$EnvAvg <- javg(community, Environment1, Environment3,
                         Environment4)

# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
juse(NULL)
```

`jcomplete` *Set a listwise complete-case filter for matching N across analyses*

Description

`jcomplete()` registers a set of variables and activates a listwise deletion filter that excludes any case with a missing value on any of the registered variables. This ensures that all subsequent analyses use the same set of complete cases, which is essential when preliminary analyses need to match the N of a final regression model.

The setting is stored per dataset, so switching `juse()` between datasets preserves each dataset's setting independently.

The jcomplete filter applies whenever the matching dataset is used, regardless of whether it was supplied via `juse()` or specified explicitly in a function call. To bypass temporarily without losing the setting, use `jcomplete(off)` before the analysis and `jcomplete(on)` afterward. This matches the SPSS USE ALL / FILTER convention.

Usage

```
jcomplete(data, ..., preview = FALSE, console = FALSE, non.deletes = FALSE)
```

Arguments

`data`	A data frame. If omitted, uses the default set by `juse()`. Pass `NULL` to clear the filter entirely. Pass the bare word `off` to deactivate, or `on` to reactivate. Call with no arguments to check the current status.
`...`	Unquoted variable names to include in the listwise check.
`preview`	Logical. If `TRUE`, open a viewer (RStudio data tab) showing the rows the listwise filter will drop, with a leading `Row` column (original data position) and a trailing `DeletionCheck` flag (1 = the row will be dropped). May be used on its own to preview the already-set filter without re-listing the variables (`jcomplete(preview = TRUE)`). Default `FALSE`.
`console`	Logical or numeric. Print the dropped rows to the console. `TRUE` prints the first 10; a number prints that many. Independent of `preview`: on its own it prints to the console without opening the viewer; combine with `preview = TRUE` to get both. The console listing is always limited to the dropped rows so it cannot flood the console. Default `FALSE`.
`non.deletes`	Logical. If `TRUE`, the viewer shows every case (with `DeletionCheck` marking which will drop) rather than only the dropped rows. Affects the viewer only; the console listing stays deleted-rows-only. Default `FALSE`.

Value

Invisibly returns `NULL`. When a preview is requested, invisibly returns the previewed data frame instead, so it can be captured (e.g. `jcomplete_rows <- jcomplete(preview = TRUE)`).

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
juse(community)
jcomplete(Income, Education, Age)
jdesc(Age)                    # Uses only complete cases on those 3 vars
jcomplete(Income, Education, Age, preview = TRUE)  # Set and preview together
jcomplete(preview = TRUE)      # Preview the already-set filter (viewer)
jcomplete(preview = TRUE, non.deletes = TRUE)  # Viewer shows all cases
jcomplete(console = 10)        # Console only -- first 10 dropped rows
jcomplete(preview = TRUE, console = 25)        # Viewer and console
jcomplete(off)                 # Deactivate
```

```
jcomplete(on)                 # Reactivate
jcomplete()                   # Check status
jcomplete(NULL)               # Clear entirely
# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
juse(NULL)
```

`jconvert` *Convert user-defined missing value (UDM) declarations between formats*

Description

`jconvert()` provides a single entry point for changing how user- defined missing values (UDMs) are represented on the columns of a data frame already in memory. Three target formats are supported: SPSS-style (`na_values` on `haven_labelled_spss`), Stata-style (`tagged_na` on `haven_labelled`), and base R (declarations stripped, declared cells converted to plain `NA`). Replaces `jstrip_udm()` (retired in v0.9.5); the base R target is the strip behavior.

Usage

```
jconvert(data, to = NULL, ..., vars = NULL, udm.notice = TRUE)
```

Arguments

`data`	A data frame, or omitted to use the `juse()` default.
`to`	One of `"baseR"`, `"spss"`, or `"stata"` (case-sensitive). When `NULL` (the default), `jconvert()` reads `joptions("missing.convention")`: if the slot is set to `"spss"` or `"stata"`, `to` resolves to that value; if the slot is at its `"none"` default, `jconvert()` errors with guidance naming the three concrete options. The destructive `"baseR"` target is never auto-resolved – it must always be passed explicitly.
`...`	Optional unquoted variable names. When supplied, only the listed variables are scanned. Mutually exclusive with `vars`.
`vars`	Alternative scope-by-vector path: a character vector of variable names. Mutually exclusive with `...`. When both `...` and `vars` are empty, `jconvert()` operates on the whole data frame.
`udm.notice`	Logical; `TRUE` (default) prints a notification summarizing what was converted (and what was skipped) along with an assignment-syntax reminder. `FALSE` suppresses the message. Always-on by default; does not consult `joutput()` because the function reports an action it just performed rather than explaining system behavior.

Details

The three target formats:

`to = "baseR"` Strip all UDM declarations and convert declared cells to plain `NA`. For SPSS-form columns (`na_values` / `na_range` on `haven_labelled_spss`), masks declared codes to `NA` and removes the attributes; value labels are preserved so the column can still round-trip through `jsave()` with original labeling. For columns carrying Stata-style missing values (`tagged_na` markers), uses `haven::zap_missing()` to convert them to plain `NA`s.

`to = "spss"` Convert Stata-style or SAS-style missing values to SPSS-form numeric codes. Letter tags map to numeric codes via `joptions("udm.convention.codes")` (default `-99`, `-98`, `-97`): `.a` -> `codes[1]`, `.b` -> `codes[2]`, and so on. SAS-style (uppercase) tags are case-corrected to Stata-style (lowercase) before the numeric mapping – for round-trip purposes the package treats `.A` and `.a` as the same conceptual marker, and mixed-case columns collapse to a single lowercase marker (SPSS has no parallel uppercase convention). The notification's per-column display shows the original (pre-correction) tag for SAS-corrected columns – e.g. `.A "Refused" -> -99` – so the user-visible mapping reflects what was actually in the data on input. Letter tags beyond `.d` (after case correction) are refused with guidance to use `jrecode()` for manual mapping.

`to = "stata"` Convert SPSS-form numeric codes to Stata-style missing values. Letter tags are assigned by ordering rather than by convention: each column's own declared `na_values` codes are sorted by absolute value descending (ties broken with more-negative-first), then mapped `.a, .b, .c` in that order. Convention codes are NOT consulted for this direction; they only govern the reverse (Stata to SPSS) mapping. Round-trip conversions are not guaranteed to preserve the original numeric codes (e.g. SPSS `c(-1, 9)` -> Stata `.a, .b` -> SPSS `c(-99, -98)` loses the original numbers), but the value labels survive intact and the missingness semantics are preserved. Range-based SPSS missings (`na_range`) are out of cross-format scope; columns with `na_range` are refused with guidance to enumerate the range in SPSS first. Columns with more than 4 distinct `na_values` codes are also refused (matches the 4-code cap on Stata letter-tag mapping).

Pre-flight checks for `to = "spss"` include a collision check: if a column's target numeric code (e.g. `-99` for `.a`) is present as genuine data in the column, the call errors before any data is touched. The error message lists every colliding column and presents three resolution paths: change the convention codes via `joptions(udm.convention.codes = ...)`, scope the call via `vars = c(...)` to exclude affected columns, or recode the real- data values via `jrecode()` first. Atomicity applies to every error mode – the entire `jconvert()` call either succeeds or errors before mutating the data frame.

Pattern A – value labels suggest missingness but no formal declaration. When a column has no formal UDM declaration but carries value labels matching the package's missing-label wordlist (e.g. `"Refused"`, `"Don't know"`, `"Not applicable"`), `jconvert()` skips the column and surfaces it in the notification with the affected value/label pairs. To formalise these as UDMs use `jdeclare_udm()`; to leave them as ordinary data, no action is needed.

Value

The data frame with the requested conversions applied, returned invisibly. As with `jrelabel()` and `jrecode()`, the user must assign the return value back to retain the changes.

See Also

jload for the load-time strip alternative (preserve.udm = FALSE); joptions for setting the default convention and convention codes session-wide.

Examples

```
# community ships with SPSS-form UDMs (Income, Education, Smoker,
# Environment1, Environment3), so the conversions run on it directly.

# Strip UDMs from every applicable variable:
df <- jconvert(community, to = "baseR")

# Convert SPSS-form UDMs to Stata-style missing values:
df <- jconvert(community, to = "stata")

# Scope by unquoted names:
df <- jconvert(community, to = "baseR", Income, Education)

# Scope by character vector (alternative form):
df <- jconvert(community, to = "baseR", vars = c("Income", "Education"))

# Suppress the notification (e.g. inside a script):
df <- jconvert(community, to = "baseR", udm.notice = FALSE)

## Not run:
# Convert with target inferred from joptions:
joptions(missing.convention = "spss")
df <- jconvert(df)   # converts any Stata-form columns to SPSS

## End(Not run)
```

---

jcopy *Copy a data frame, carrying its classification registrations*

Description

Copies a data frame to a new name AND clones any classification registrations (jnumeric / jcount / jdummy) attached to it, so the copy behaves the same as the original under later analysis calls. A plain assignment (newdata <- mydata) copies the data but not the registrations, because registrations live in a name-keyed session notebook rather than on the data object; jcopy() is the verb that keeps the two together across a rename or copy.

Usage

```
jcopy(data, name, overwrite = FALSE, quiet = FALSE)
```

Arguments

`data`	The source data frame (unquoted). May be omitted when a juse() default is set, in which case the default frame is the source.
`name`	The destination name (unquoted) the copy is assigned to. When a single name is given it is read as the destination, not the source.
`overwrite`	Logical; if FALSE (the default) and the destination name already exists in your environment, an interactive session asks before overwriting.
`quiet`	Logical; if TRUE, suppress the confirmation message.

Details

Like jload(), jcopy() cannot see the name on the left of an assignment, so the new name is supplied as an argument. The destination name is unquoted, and a single name is always taken as the destination, with the source coming from the juse() default:

- `jcopy(mydata, newdata)` – copy `mydata` to `newdata`.
- `jcopy(newdata)` – copy the juse() default frame to `newdata`.

Registrations travel only when the source frame carries them; copying an unregistered frame just copies the data. The copy is independent of the original.

Value

Invisibly NULL. Called for its side effect: the copy is assigned into the calling environment under `name`, and its registrations are cloned onto that name.

See Also

`jload`, `jsave`, `juse`

Examples

```
## Not run:
  jdummy(community, Region)        # register a classification on community
  jcopy(community, survey)         # survey carries Region's registration

  juse(community)
  jcopy(survey2)                   # copy the default (community) to survey2

## End(Not run)
```

jcorr *Bivariate correlation matrix with p values and pairwise N*

Description

Computes pairwise correlations and prints a formatted lower-triangle correlation matrix showing r, p values, and pairwise N for each pair. Supports Pearson (default), Spearman, and Kendall methods. Handles haven-labelled and factor variables with numeric levels. Warns when variables may be categorical rather than continuous.

Usage

```
jcorr(
  data,
  ...,
  method = "pearson",
  subset = NULL,
  variable.id = NULL,
  numeric = NULL,
  categorical = NULL,
  count = NULL,
  value.id = NULL,
  layout = NULL,
  case.processing.detail = NULL,
  digits = NULL
)
```

Arguments

data	A data frame.
...	Unquoted variable names within data.
method	Character. Correlation method: "pearson" (default), "spearman", or "kendall".
subset	An optional unquoted logical expression (e.g. Group == 1) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
variable.id	Character or NULL. Variable label display mode: one of "both", "names", "labels", "legend", or "legend.bottom". "names" shows variable names only; "both" shows "name: label"; "labels" shows variable labels as the matrix row/column headers (honored even if the matrix grows wide – best for short labels; rerun with a legend mode otherwise); "legend"/"legend.bottom" keep names and print a label legend after the table. NULL (default) defers to joutput()'s variable.id setting. Not a logical.
numeric	Optional character vector of variable names to treat as continuous for this call (the per-call counterpart of jnumeric()). Its only effect in jcorr() is to suppress the structural "seems categorical" caution for those variables; correlations

are computed the same way regardless (labelled variables are coerced to numeric either way).

`categorical` Not supported by `jcorr()` yet. Correlation requires numeric variables; supplying `categorical` raises an error pointing to `jcrosstab()` for association between categorical variables. (How `jcorr()` should handle an asserted-categorical variable is a parked design decision.)

`count` Optional character vector of variable names to treat as counts for this call (the per-call counterpart of `jcount()`). A count is numeric-like here, so it behaves like `numeric`: it suppresses the "seems categorical" caution for those variables.

`value.id` Not supported by `jcorr()`. The function does not display value labels, so passing this argument is an error. It exists only to return a clear message rather than misreporting the token as a missing variable. Leave at NULL (default).

`layout` Character or NULL. How each correlation cell is laid out when three or more variables are given: `"wide"` (default) puts r and its p-value on one line with N on a second line beneath; `"stacked"` places r, p, and N on three separate lines, giving a narrower table that fits more variables before wrapping. Ignored for a single pair (two variables), which always prints a one-line summary. NULL (default) defers to the `corr.layout` setting in `joptions()` (itself defaulting to "wide").

`case.processing.detail`
Per-call override of the Case Processing Summary detail tier: one of `"none"`, `"totals"`, or `"per_code"`. `NULL` (default) uses the active `joutput()` level default.

`digits` Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

Details

A red title identifying the correlation method is printed first, followed by variable labels (if present), then the matrix.

Value

Invisibly returns a list of class `jst_corr` containing: `r` (correlation matrix), `p` (p-value matrix), `n` (pairwise N matrix), `method`, `model_frame` (the analysis data frame used for plotting), and `sample_info` (pipeline and missing data counts).

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jcorr(community, Income, Age, WellbeingScore)
```

```
jcorr(community, Income, Age, WellbeingScore, method = "spearman")

# Using juse() default
juse(community)
jcorr(Income, Age, WellbeingScore)
```

---

jcount *Register variables as counts for analysis*

Description

`jcount()` tells jstats to treat one or more variables as count variables (non-negative whole-number tallies). A count is numeric-like – it passes wherever a numeric variable does and shows mean/median in `jscreen` – and additionally carries count semantics: it is the asserted signal behind the count-regression caveat in `jlm` and the routing target for future count-model functions. Unlike the structural guess, jcount accepts counts of any range, including those outside the automatic small-range detection (e.g. a 0-30 victimization count).

Usage

```
jcount(data, ..., remove = FALSE, clear.all = FALSE)
```

Arguments

`data`	A data frame, or omitted to use the `juse` default. `jcount(NULL)` clears the count registrations on the `juse` default frame (or, with no default set, the only frame that carries them; if several do, it asks rather than wiping all). `jcount(data, NULL)` clears that one frame's count registrations. Called with no arguments, `jcount()` lists the session's numeric and count registrations.
`...`	One or more unquoted variable names to register.
`remove`	Logical; if `TRUE`, remove the count registration for the named variables instead of adding it.
`clear.all`	Logical; if `TRUE`, clear count registrations on every data frame that carries them.

Details

A variable carries exactly one registered intent at a time, so registering it as a count clears any prior dummy or numeric registration. Registration changes no data and assigns nothing. It is stored for the session, keyed by the data frame's name; save the data frame in R format (.rds) to keep it across sessions.

Value

`invisible(NULL)`. Called for its side effect on the session registration notebook.

See Also

jnumeric, jdummy

Examples

```
df <- data.frame(arrests = c(0, 1, 2, 0, 3, 1, 0, 12),
                 age     = c(21, 34, 45, 29, 51, 38, 26, 60))
jcount(df, arrests)                # treat as a count (here 0-12)
jcount(df, arrests, remove = TRUE)
jcount()                           # list all registrations
jcount(df, NULL)                   # clear df's count registrations
jcount(clear.all = TRUE)           # clear every frame's count registrations
```

---

jcrosstab *Cross-tabulation with optional chi-square test of independence*

Description

Produces a cross-tabulation of two categorical variables, showing observed frequencies and row percentages by default. Column percentages, expected frequencies, adjusted standardized residuals, and a chi-square test of independence are available via arguments. Handles haven-labelled, numeric, factor, and character variables. For haven-labelled variables, numeric codes are displayed alongside labels.

Usage

```
jcrosstab(
  formula,
  data,
  chisq = FALSE,
  expected = FALSE,
  row.pct = TRUE,
  col.pct = FALSE,
  residuals = "none",
  subset = NULL,
  variable.id = NULL,
  value.id = NULL,
  case.processing.detail = NULL,
  digits = NULL
)
```

Arguments

formula	A formula of the form Row ~ Column.
data	A data frame containing variables referenced in formula.
chisq	Logical. If TRUE, prints the chi-square test of independence below the cross-tabulation. Default is FALSE.

`expected` Logical. If TRUE, prints expected frequencies alongside observed. Default is FALSE.

`row.pct` Logical. If TRUE (default), shows row percentages.

`col.pct` Logical. If TRUE, shows column percentages. Default is FALSE.

`residuals` Character. Cell residuals to display: `"none"` (default) or `"adjusted"`. `"adjusted"` adds an `(Adj.Res.)` line to each cell showing the adjusted standardized (Haberman) residual: (observed - expected) divided by its standard error. Under independence these are approximately standard normal, so a value beyond +/-1.96 flags a cell whose count departs from expected at the .05 level. This localizes a significant chi-square to individual cells, and matches the "Adjusted standardized" residual in SPSS CROSSTABS. At `joutput("full")` the residual cells are flagged (`*` past +/-1.96, `**` past the Bonferroni cutoff) and an interpretation note is printed below the table naming both thresholds. Not a logical.

`subset` An optional unquoted logical expression (e.g. `Group == 1`) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.

`variable.id` Character or NULL. Variable label display mode: one of `"both"`, `"names"`, `"labels"`, `"legend"`, or `"legend.bottom"`. `"names"` shows variable names only; `"both"` shows `"name: label"`; `"labels"` shows the row/column variable labels (table header and caption; cell value levels follow the value.id mode) – best for short labels; `"legend"`/`"legend.bottom"` keep names and print a label legend after the table. NULL (default) defers to `joutput()`'s `variable.id` setting. Not a logical.

`value.id` Character or NULL. Value-label display mode for both table axes: `"both"` (`"code: label"`), `"values"` (bare code), or `"labels"` (the label, degrading to the bare code where a code has none). `"legend"` and `"legend.bottom"` keep the bare code in the table and print a value-label legend after it (`"legend"` per-table, `"legend.bottom"` consolidated where multiple tables are produced). A no-op for axis variables with no value labels. NULL (default) defers to `joutput()`'s `value.id` setting. Not a logical.

`case.processing.detail`
Per-call override of the Case Processing Summary detail tier: one of `"none"`, `"totals"`, or `"per_code"`. `NULL` (default) uses the active `joutput()` level default.

`digits` Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

Details

A red "Cross-Tabulation" title is printed first, followed by variable labels (if present), then the table and optional test results.

Value

Invisibly returns a list of class `jst_crosstab` containing: `observed` (observed frequency table), `expected` (expected frequency table), `adjusted_residuals` (matrix of adjusted standardized residuals), `n` (total N), `model_frame` (the analysis data frame used for plotting), `sample_info` (pipeline and missing data counts), and if `chisq = TRUE`: `chi_square`, `df`, and `p`.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# Cross-tabulation only
jcrosstab(Education ~ Volunteer, data = community)

# With chi-square test
jcrosstab(Education ~ Volunteer, data = community, chisq = TRUE)

# With expected frequencies and column percentages
jcrosstab(Education ~ Volunteer, data = community,
          expected = TRUE, col.pct = TRUE)

# With adjusted standardized residuals (interpretation note at full output)
jcrosstab(Education ~ Volunteer, data = community, residuals = "adjusted")

# Using juse() default
juse(community)
jcrosstab(Education ~ Volunteer)
jcrosstab(Education ~ Volunteer, chisq = TRUE)
```

---

jdata_dir *Return the configured data folder*

Description

Read-side companion to `joptions(data.dir = ...)`: returns the currently configured data folder as a string, for use in scripts that need the path itself (building a file path, checking existence, cleaning up test files) without reaching into package-internal option names.

Usage

```
jdata_dir(default = ".")
```

Arguments

`default`	Value returned when no data folder is configured. Defaults to `"."` (the working directory).

Details

joptions() prints the folder but returns invisible(NULL); jdata_dir() returns it as a value. When no folder is configured, the default is returned (".", the working directory, by default), so the result drops straight into file.path. Pass default = NULL to detect the unconfigured state explicitly.

Value

A length-one character string (the configured folder, or default); or default unchanged when it is NULL.

See Also

joptions to set the folder; jload and jsave, which resolve files against it.

Examples

```
## Not run:
joptions(data.dir = "Data")
jdata_dir()                                  # "Data"
f <- file.path(jdata_dir(), "community.rds") # build a path in that folder
if (file.exists(f)) file.remove(f)

jdata_dir(default = NULL)                    # NULL if nothing configured

## End(Not run)
```

---

jdeclare_udm *Declare user-defined missing values on a variable*

Description

jdeclare_udm() declares one or more user-defined missing values (UDMs) on a variable. UDMs are specific data values – typically negative codes such as -99 or Stata-style tagged markers such as .a – that indicate *why* a value is missing (refused, don't know, not applicable, etc.) rather than simply that it is missing. Once declared, UDM cells are automatically excluded from analyses but remain visible in the data for diagnostic purposes (see jfreq()).

The function operates in declarative mode: each call states the column's complete UDM set. A second call to jdeclare_udm() on the same column replaces, not augments, the prior declaration. This matches SPSS's MISSING VALUES and Stata's mvdecode semantics. When prior UDMs are dropped, a note lists them so the destructive aspect of the replacement is not silent.

Usage

```
jdeclare_udm(
  data,
  var,
  codes = NULL,
  labels = NULL,
  convention = NULL,
  udm.notice = TRUE
)
```

Arguments

data	A data frame containing the variable.
var	The variable to declare UDMs on (unquoted, e.g. Income).
codes	Numeric vector of code values to declare as UDMs. Accepts two forms: **Option A (separate codes and labels)** Unnamed numeric vector; labels supplied via the labels argument. E.g. codes = c(-99, -98), labels = "-99=Refused; -98=Don't know". **Option C (haven-style named vector)** Named numeric vector; names are the labels. E.g. codes = c(Refused = -99, `Don't know` = -98). Under Stata convention, code values may be Stata-style missing-value markers created with haven::tagged_na(), e.g. codes = c(Refused = tagged_na("a")).
labels	Optional. A quoted string in the form "value=label; value=label" pairing labels with codes (Option A only). Must be NULL when codes is named (Option C).
convention	Optional. One of "spss" or "stata"; overrides the convention resolution for this call. When NULL (the default), the convention is resolved from the column's existing UDM declaration (if any), then from joptions("missing.convention"), then from the SPSS-form default.
udm.notice	Logical. When TRUE (the default), the function prints a notification summarizing what was declared. Set FALSE to suppress.

Value

The data frame, with the specified variable updated to carry the declared UDMs.

Missing-Values Convention

Under SPSS convention, codes are declared as numeric values via the column's na_values attribute (haven's representation of SPSS-form UDMs). The data cells themselves are unchanged; only the metadata that flags certain values as missing is added.

Under Stata convention with Stata-style missing-value input, the function attaches value labels to existing Stata-style missing-value cells on the column.

Under Stata convention with numeric input, the function converts matching cells to Stata-style missing-value markers (Session 30 design lock). The mapping is ordering-based: codes sorted by absolute value descending, more-negative-first as tie-breaker, then assigned .a, .b, .c, .d in that

order. The assignment proceeds independently of `joptions("udm.convention.codes")` (which only governs the reverse Stata-to-SPSS direction). A conversion note in the standard/full `joutput` tier shows the Stata-style equivalent for future calls.

Mixed conventions and file export

A single data frame may carry both SPSS-form and Stata-form UDM columns. In-memory analysis and display tolerate the mix without issue (each column renders in its native form). The constraint shows up at file-export time: `.sav` cannot represent Stata-style missing values; `.dta` cannot represent SPSS-form `na_values` declarations; `.xpt` can represent neither form. `jsave()` pre-flights the DF against the destination format and errors with a pointer to `jconvert()` when the mix is incompatible. The post-declaration mismatch notice emitted at the bottom of this function's output exists to alert you early if a single-column declaration ends up out of step with the rest of its DF.

See Also

jrecode, jconvert, joptions, jstats

Examples

```
# clinic$MoodRating arrives "dirty": -99/-98 sit in the data as
# ordinary numbers (the state after a CSV or Excel import), so summary
# statistics are poisoned until the codes are declared missing.
df <- clinic
jdesc(df, MoodRating)        # mean dragged far down by -99/-98

# SPSS form: declare -99 and -98 as UDMs with labels
df <- jdeclare_udm(df, MoodRating,
                   codes  = c(-99, -98),
                   labels = "-99=Refused; -98=Don't know")
jdesc(df, MoodRating)        # codes now excluded as missing

# Equivalent using named codes (one step instead of codes + labels)
df2 <- jdeclare_udm(clinic, MoodRating,
                    codes = c("Refused" = -99, "Don't know" = -98))

# Stata-style: label Stata-style missing-value cells. The jrecode() call
# turns the literal codes into tagged cells; jdeclare_udm() labels them.
df3 <- clinic
df3$Mood2 <- jrecode(df3, MoodRating,
                     map = "-99=.a; -98=.b; else=copy",
                     convention = "stata")
df3 <- jdeclare_udm(df3, Mood2,
                    codes = c("Refused"    = haven::tagged_na("a"),
                              "Don't know" = haven::tagged_na("b")))
```

jdesc *Descriptive statistics for one or more variables*

Description

Computes basic descriptive statistics (N, non-missing, min, max, mean, SD) for one or more variables in a data frame. Prints a formatted table and invisibly returns the underlying results as a data frame.

Usage

```
jdesc(
  data,
  ...,
  by = NULL,
  subset = NULL,
  variable.id = NULL,
  numeric = NULL,
  categorical = NULL,
  count = NULL,
  value.id = NULL,
  case.processing.detail = NULL,
  digits = NULL
)
```

Arguments

data	A data frame, or a numeric vector.
...	Unquoted variable names within data (ignored if data is a vector).
by	An optional unquoted grouping variable name. When provided, descriptives are computed separately for each group, with a separate titled table per dependent variable.
subset	An optional unquoted logical expression (e.g. Group == 1) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
variable.id	Character or NULL. Variable label display mode: one of "both", "names", "labels", "legend", or "legend.bottom". "names" shows variable names only; "both" shows "name: label"; "labels" shows each variable's label in place of its name (in the descriptives table; for grouped output, as the per-variable caption and the grouping-variable column header) – best for short labels; "legend" and "legend.bottom" keep names and print a label legend after the table. NULL (default) defers to joutput()'s variable.id setting. Not a logical.
numeric	Optional character vector of variable names to treat as continuous for this call (the per-call counterpart of jnumeric()). Its only effect in jdesc() is to suppress the structural "seems categorical" descriptive caution for those variables; the descriptives themselves are computed the same way regardless.

`categorical` Not supported by `jdesc()` yet. `jdesc()` always computes numeric descriptives; supplying `categorical` raises an error pointing to `jfreq()` for a categorical summary. (How `jdesc()` should handle an asserted-categorical variable is a parked design decision.)

`count` Optional character vector of variable names to treat as counts for this call (the per-call counterpart of `jcount()`). A count is numeric-like here, so it behaves like `numeric`: it suppresses the "seems categorical" caution for those variables.

`value.id` Character or NULL. Value-label display mode for the grouped descriptive headers (the by-group rows): `"both"` (`"code: label"`), `"values"` (bare code), or `"labels"` (the label, degrading to the bare code where a code has none). `"legend"` and `"legend.bottom"` keep the bare code in the table and print a value-label legend after it (`"legend"` per-table, `"legend.bottom"` consolidated where multiple tables are produced). A no-op for grouping variables with no value labels, and for ungrouped calls. NULL (default) defers to `joutput()`'s `value.id` setting. Not a logical.

`case.processing.detail` Per-call override of the Case Processing Summary detail tier: one of `"none"`, `"totals"`, or `"per_code"`. `NULL` (default) uses the active `joutput()` level default.

`digits` Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

Details

Output is structured consistently with `jfreq()`: a red title is printed first, followed by a block showing the type and variable label (or "None" if no label is present) for each variable, then a single blank line before the table. For multiple variables, one type/label entry is printed per variable before the shared table.

Summarizes numeric, haven-labelled, logical, numeric-coded factor, and numeric-looking character variables. Variables that cannot be summarized — text factors, text character variables, and date/time variables — are skipped with a warning directing the user to `jfreq()` (date/time variables are not supported here). When every requested variable is unsummarizable, jdesc() stops with an error. Also accepts a simple numeric vector. Supports grouped descriptives via the by parameter.

Haven-labelled variables are reported as `haven_labelled (Categorical)` in the type line; the uninformative `vctrs_vctr` class is suppressed.

Value

Invisibly returns a list of class `jst_desc` containing: `descriptives` (data frame of statistics, or NULL for grouped output), and `sample_info` (pipeline and missing data counts). Also prints a formatted table to the console.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jdesc(community, Age)
jdesc(community, Income, Age, WellbeingScore)
jdesc(community, WellbeingScore, by = Volunteer)

# Using juse() default
juse(community)
jdesc(Age)
jdesc(Income, Age, WellbeingScore)
jdesc(WellbeingScore, by = Volunteer)

# With a vector directly
jdesc(community$Age)
```

---

jdummy *Register categorical variables for dummy coding in regression*

Description

`jdummy()` registers a categorical variable so that `jlm()` automatically expands it into dummy (indicator) variables when it appears in a regression formula. The original data frame is never modified. Several variables can be registered in one call; the `ref` setting then applies to each of them.

Registrations are stored per dataset, so switching `juse()` between datasets preserves each dataset's registrations independently.

Usage

```
jdummy(
  data,
  ...,
  ref = "first",
  show = FALSE,
  remove = FALSE,
  clear.all = FALSE,
  max.categories = 20L
)
```

Arguments

`data`	A data frame, or omit to use the `juse()` default. `jdummy(NULL)` clears the dummy registrations on the `juse()` default data frame (or, with no default set, the only frame that carries them; if several do, it asks rather than wiping all).
`...`	One or more unquoted variable names to register. Omit (along with data) to display all current registrations. A lone `NULL` in the variable slot – `jdummy(data, NULL)` – clears that frame's dummy registrations.

ref	The reference category (excluded from the regression model). Can be a numeric code, a quoted label name, or `first` (default) or `last`. Applied to every variable named in the call; to use different reference categories, register the variables in separate calls.
show	Logical. If `TRUE`, prints the dummy coding scheme table showing the pattern of 0s and 1s. Default is `FALSE`.
remove	Logical. If `TRUE`, removes the registration for the specified variable(s). Default is `FALSE`.
clear.all	Logical. If `TRUE`, clears dummy registrations on every data frame that carries them. Default is `FALSE`.
max.categories	Integer. Maximum number of categories a variable may have to be dummy-coded; a variable with more raises an error. Raise it to dummy-code a higher-cardinality variable. Default `20L`.

Value

Invisibly returns `NULL`. Called for its side effect.

See Also

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
juse(community)
jdummy(Region)                      # Register, first category as reference
jdummy(Region, Education)           # Register several at once
jdummy(Region, ref = "last")        # Last category as reference
jdummy(Region, ref = 4)             # Reference by numeric code
jdummy(Region, ref = "East")        # Reference by value label
jdummy(Region, show = TRUE)         # Show coding scheme
jdummy(Region, show = "all")        # Full scheme (for many categories)
jdummy()                            # Show all registrations
jdummy(Region, remove = TRUE)       # Remove one registration
jdummy(community, NULL)             # Clear community's dummy registrations
jdummy(NULL)                        # Clear the default frame's (or ask)
jdummy(clear.all = TRUE)            # Clear every frame's dummy registrations
# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
juse(NULL)
```

---

jfreq *SPSS-like frequency tables for categorical variables*

Description

Prints an SPSS-style frequency table (Freq, Total %, Valid %, Cum. %) for each variable supplied. Designed for use with unquoted variable names, and also accepts a plain vector.

Usage

```
jfreq(
  data,
  ...,
  subset = NULL,
  variable.id = NULL,
  value.id = NULL,
  case.processing.detail = NULL
)
```

Arguments

data	A data frame, or a vector.
...	Unquoted variable name(s) within data (ignored if data is a vector).
subset	An optional unquoted logical expression (e.g. Group == 1) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
variable.id	Character or NULL. Variable label display mode: one of "both", "names", "labels", "legend", or "legend.bottom". "names" shows variable names only; "both" shows "name: label"; "labels" uses each variable's label as its table caption (best for short labels); "legend" prints a label legend under each variable's own table; "legend.bottom" prints one consolidated legend after all tables. NULL (default) defers to joutput()'s variable.id setting. Not a logical. (Replaces the former inline Type/label block.)
value.id	Character or NULL. Value-label display mode for the frequency-table valid rows: "both" ("code: label"), "values" (bare code), or "labels" (the label, degrading to the bare code where a code has none). "legend" and "legend.bottom" keep the bare code in the table and print a value-label legend after it ("legend" per-table, "legend.bottom" consolidated where multiple tables are produced). A no-op for variables with no value labels. NULL (default) defers to joutput()'s value.id setting. Not a logical.
case.processing.detail	Accepted for API symmetry. jfreq's Case Processing Summary is top-table only (no missing-data breakdown), so this argument has no effect; per-variable code detail already appears in each variable's frequency table.

Details

Output is structured consistently with jdesc(): a single red "Frequencies" title is printed first, followed by the default-data note (if a juse() default was used), any pipeline messages, and the Case Processing Summary (when at least one pipeline stage was active for this call). Each variable then gets its own block consisting of the variable name on its own line, indented Type and Variable label lines (suppressed when joutput()'s variable.id toggle is off), a blank line, and the frequency table. The frequency table ends with a Total row showing the post-pipeline N.

For haven-labelled variables, value labels and numeric codes are combined in the frequency table rows (e.g. 1: Strongly Oppose). The type line reports haven_labelled (Categorical) and sup-

presses the uninformative `vctrs_vctr` class. Variable labels are shown for all variable types, not only haven-labelled ones.

Value

Invisibly returns a list of class `jst_freq` containing: `frequencies` (named list of data frames, one per variable) and `sample_info` (pipeline and missing data counts).

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jfreq(community, Region)
jfreq(community, Region, Education)

# Using juse() default
juse(community)
jfreq(Region)
jfreq(Region, Education)

# With a vector directly
jfreq(community$Region)
```

---

jlikert *Register variables as Likert (ordered response) items*

Description

`jlikert()` declares one or more value-labelled variables as Likert items – ordered response scales (for example 1 = Strongly disagree through 5 = Strongly agree). It is the ordered-scale counterpart to `jdummy` (categorical), `jnumeric` (continuous), and `jcount` (count): a variable carries exactly one registered intent at a time, so registering it as Likert clears any prior numeric, count, or dummy registration on it.

Usage

```
jlikert(data, ..., remove = FALSE, clear.all = FALSE)
```

Arguments

`data`	A data frame, or omitted to use the `juse` default.
`...`	One or more unquoted variable names to register, or a single `NULL` to clear this frame's Likert registrations (see Details).
`remove`	Logical; if TRUE, remove the named variables' Likert registrations instead of adding them.
`clear.all`	Logical; if TRUE, clear Likert registrations on every data frame.

Details

Scope – display only. The Likert intent refines reporting, not analysis. It sets the variable's subclass to "Likert" in `jscreen`'s Variable Types table, marking it as an ordered scale rather than a generic N-category variable. It does NOT change how any analysis treats the variable (there is no order-aware modelling), and it does not by itself change `jplot` output – a value-labelled small-range variable already plots as an ordered, labelled bar regardless of this registration.

Like the other registration verbs, registrations are session-scoped and keyed by data-frame name; save the frame in R format (.rds) with `jsave` to keep them across sessions.

Clearing mirrors the other registration verbs:

- `jlikert(data, NULL)` – clear this frame's Likert registrations.
- `jlikert(NULL)` – clear the `juse()` default frame (or the sole frame carrying Likert registrations; if several do, it asks rather than clearing them all).
- `jlikert(clear.all = TRUE)` – clear every frame.

`jlikert()` with no arguments prints the current registration status.

Value

Invisibly NULL. Called for its side effect on the session registry.

See Also

`jnumeric`, `jcount`, `jdummy`, `jscreen`

Examples

```
jlikert(community, Environment1, Environment2)  # declare two Likert items
jscreen(community)                              # Sub-class shows "Likert"
jlikert(community, Environment1, remove = TRUE) # undo one
jlikert(community, NULL)                        # clear the registrations
```

`jlm` *SPSS-like linear regression output with standardized coefficients*

Description

Fits a linear model using `stats::lm()` and prints SPSS-style output, including unstandardized coefficients, standard errors, t values, p values, and standardized coefficients (beta). Standardized coefficients are left blank for the intercept and for dummy-coded categorical terms.

Usage

```
jlm(
  formula,
  data,
  subset = NULL,
  variable.id = NULL,
  numeric = NULL,
  categorical = NULL,
  count = NULL,
  ci = NULL,
  std = "regular",
  diagnostics = NULL,
  ref.categories = NULL,
  full = FALSE,
  case.processing.detail = NULL,
  digits = NULL,
  ...,
  value.id = NULL
)
```

Arguments

formula	A model formula, e.g. y ~ x1 + x2.
data	A data frame containing variables referenced in formula.
subset	An optional unquoted logical expression (e.g. Group == 1) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
variable.id	Character or NULL. Variable label display mode: one of "both", "names", "labels", "legend", or "legend.bottom". "names" shows variable names only; "both" shows "name: label"; "labels" replaces each coefficient's variable name with its label in the Coefficients table (factor level decoration is preserved) – best for short labels; "legend" prints a label legend between the Coefficients table and the R-squared/fit block; "legend.bottom" prints it at the very end. NULL (default) defers to joutput()'s variable.id setting. Not a logical.
numeric	Optional character vector of variable names that should be treated as continuous (numeric) even if they have value labels. For example, numeric = "Age" or numeric = c("Age", "Education").
categorical	Optional character vector of variable names that should be treated as categorical even if they lack value labels. For example, categorical = "Program" or categorical = c("Program", "Region"). The first sorted unique value becomes the reference category. Use jdummy() for control over the reference category.
count	Optional character vector of variable names to treat as counts for this call (the per-call counterpart of jcount()). On the dependent variable it speaks the count-regression caveat definitively rather than as a hedge, and applies even

when the variable sits outside the structural 0-6 band. On an independent variable it behaves like `numeric` (a count predictor enters the model as numeric). A variable cannot be listed in both `count` and `categorical`.

`ci` Logical or NULL. If TRUE, appends a 95% confidence interval for each unstandardized coefficient (b) at the right of the coefficient table. If NULL (default), defers to `joutput()`'s regression.ci setting (off at minimal and standard, on at full). Computed as the closed form b +/- t(.975, residual df) * SE.

`std` Character. Controls the standardized-coefficient column. One of `"regular"` (default) – standardized betas with the prevalence-scaled betas of dummy and dichotomous predictors suppressed, since a fully standardized beta on a 0/1 indicator is not comparable to the continuous betas; `"all"` – the same standardized betas with nothing suppressed; `"gelman"` – Gelman (2008) scaling, where continuous predictors are placed on a divide-by-two-standard-deviations scale and binary predictors keep their raw 0/1 contrast (shown for all predictors, and headed "Gelman beta"); or `"none"` – omit the column. The returned object always carries both the full regular betas (`beta`) and the full Gelman betas (`beta_gelman`) regardless of this display choice.

`diagnostics` Logical, character vector, or NULL. If TRUE, prints VIF table and diagnostic plots. If a character vector, specifies which diagnostics to show: `vif`, `residuals`, `qq`, `scale`, `cooks`, `leverage`. If NULL (default), defers to `joutput()` session setting.

`ref.categories` Logical or NULL. Per-call override for showing the reference-categories block (the baseline level dropped from each set of dummy variables). `NULL` (default) defers to `joutput()`'s `ref.categories` setting. Applies to `jlm()` and `jlogistic()` only, since they are the functions that produce dummy-coded coefficient tables.

`full` Logical. If TRUE, turns on the coefficient confidence interval and diagnostics. Does not override explicit FALSE values.

`case.processing.detail`
Per-call override of the Case Processing Summary detail tier: one of `"none"`, `"totals"`, or `"per_code"`. `NULL` (default) uses the active `joutput()` level default.

`digits` Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

`...` Reserved for argument-name checking. Passing `which`, `plots`, or `show` will produce a helpful error suggesting `diagnostics` instead.

`value.id` Character or NULL. Value-label display mode for the dummy category rows in the Coefficients table: one of `"both"` (`"code: label"`, degrading to a bare code where a code has no label), `"values"` (the bare code), or `"labels"` (the value label, degrading to the bare code where none exists). The reference category folded into each grouped variable's header follows the same mode. `"legend"` and `"legend.bottom"` are not supported here: a coefficient table already pairs each value label with its row, so a separate legend block would only duplicate

it. Passing either explicitly is an error; a `joutput()` default of `"legend"` or `"legend.bottom"` is tolerated and rendered as `"both"`, so it does not break a bare call. Variables with no value labels render identically under all supported modes. NULL (default) defers to `joutput()`'s `value.id` setting. Applies only to multi-category dummy predictors; continuous and single-contrast (dichotomous) predictors are unaffected. Not a logical.

Details

Also prints key model summary information (R-squared, adjusted R-squared, residual standard error, F-test, sums of squares, and N). If any coefficients are dropped due to perfect collinearity, a warning message is printed.

A red "Linear Regression" title is printed first, followed by variable labels (if present), then the coefficient table and model fit statistics.

Handling of variables:

- Variables registered with `jdummy()` are expanded into dummy variables using the registered reference category.
- Unregistered haven-labelled variables with value labels are automatically treated as categorical (converted to factors). The first category is used as the reference, and an informational message suggests using `jdummy()` for control over the reference category.
- Haven-labelled variables without value labels are treated as continuous (converted to numeric).
- The `numeric` argument overrides auto-detection for variables that have value labels but should be treated as continuous (e.g. Age with labels like "18 years", "19 years").
- The `categorical` argument forces variables without value labels (or plain numeric variables) to be treated as categorical (e.g. a numeric Program variable coded 1–4 from a CSV file).
- The dependent variable is always modelled as numeric. Naming it in `numeric` or `count` does not change that; it only asserts the DV's role so the count / categorical-like note is silenced (`numeric`) or stated definitively (`count`).

Value

Invisibly returns a list of class `jst_lm` containing:

model The fitted `lm` object.

model_type Character string `linear`.

model_frame The model frame used to fit the model.

formula_used The formula after dummy expansion.

coefficients Formatted coefficient table (data frame); includes 95% CI Lower / Upper columns when `ci` is on.

coefficients_raw Flat data frame of raw, full-precision coefficient statistics (one row per coefficient): `term` (machine key), `b`, `SE`, `t`, `df`, `p`, `beta`, and `ci_lower` / `ci_upper` bounds (present regardless of the `ci` display toggle). Carries `beta_standardization` and `outcome` attributes.

fit_raw List of raw, full-precision fit statistics (R-squared, adjusted R-squared, residual SE, F with its dfs and p-value, residual df, and N).

r_squared R-squared value.

adj_r_squared Adjusted R-squared value.

residual_se Residual standard error.

f_statistic Named numeric vector with F value, df1, df2, and p.

sums_of_squares Named numeric vector (regression, residual, total).

n Number of observations used in the model.

dummy_coef_names Names of dummy variable columns created by `jdummy()` registrations.

ref_cats Reference category descriptions for all categorical variables in the model.

vif Named numeric vector of VIF values, or NULL for bivariate.

sample_info Pipeline and missing data counts.

See Also

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame (named argument)
jlm(WellbeingScore ~ Income + Age, data = community)

# With explicit data frame (positional argument)
jlm(WellbeingScore ~ Income + Age, community)

# Using juse() default
juse(community)
jlm(WellbeingScore ~ Income + Age)

# CATEGORICAL PREDICTORS
#
# Per-call: categorical = ... applies for one call only and does not
# persist. Useful for a quick one-off analysis.
jlm(WellbeingScore ~ Region + Age, categorical = "Region")

# The recommended approach for repeated analyses: register the variable
# with jdummy() before running jlm(). This sets the categorical
# treatment persistently, so subsequent jlm() calls (and other
# analyses) use the same coding without re-specifying.
jdummy(community, Region)
jlm(WellbeingScore ~ Region + Age)

# To choose a non-default reference category:
jdummy(community, Region, ref = "West")
jlm(WellbeingScore ~ Region + Age)

# FORCING NUMERIC TREATMENT
#
# Use numeric = ... when a variable has value labels (haven_labelled)
# but you want it treated as a continuous score (e.g., a Likert
# scale you want the slope-per-unit interpretation for).
```

```
jlm(WellbeingScore ~ Age + Education, numeric = "Education")

# Multiple overrides at once
jlm(WellbeingScore ~ Education + Environment4 + Smoker,
    numeric = c("Education", "Environment4"), categorical = "Smoker")

# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
jdummy(community, NULL)
juse(NULL)
```

`jload` *Load a data file into R*

Description

`jload()` reads a data file and assigns it as a data frame in your environment. Supports SPSS (`.sav`), Stata (`.dta`), SAS (`.sas7bdat`, `.xpt`), Excel (`.xlsx`, `.xls`), CSV (`.csv`), and R's native `.rds` format.

The file format is determined entirely by the file extension — `jload()` reads the extension (e.g. `.sav`, `.dta`, `.xlsx`) and uses the appropriate reader automatically.

By default, `jload()` looks for the file in the working directory. If a data folder is configured with `joptions(data.dir = ...)`, that folder is searched first. If a full file path is provided, it is used directly.

The data frame is automatically named after the file (without the extension). Use the `name` argument to specify a different name.

Usage

```
jload(
  file,
  name = NULL,
  use = FALSE,
  overwrite = FALSE,
  package = FALSE,
  check.missing = TRUE,
  sheet = NULL,
  preserve.udm = TRUE,
  udm.notice = NULL,
  quiet = FALSE
)
```

Arguments

`file`	Character string. The filename (e.g. `"mydata.sav"`) or a full file path (e.g. `"C:/Projects/mydata.sav"`). Use forward slashes in file paths. If the extension is omitted, `jload()` searches for common data file types automatically.

name	Character string (optional). The name to assign the data frame in your environment. If omitted, the name is derived from the filename.
use	Logical. If TRUE, automatically calls juse() on the loaded data frame to set it as the default for jstats functions. Default is FALSE.
overwrite	Logical. If TRUE, overwrites an existing object with the same name without prompting. If FALSE (default), prompts for confirmation in interactive sessions. In non-interactive sessions, overwrites with a warning message.
package	Logical. If TRUE, loads a jstats example dataset shipped in the package (e.g. community, clinic) by bare name, bypassing the disk search. Use this when a same-named file in the working directory or data folder would otherwise shadow the shipped dataset. If FALSE (default), a matching disk file takes precedence and the shipped dataset is used only when no file matches. file must be a bare name with no path or extension when package = TRUE.
check.missing	Logical. If TRUE (default), scans numeric variables for values that look like coded missing values (e.g. -99, 999) and reports them. Set to FALSE to skip this check.
sheet	For Excel files only. The sheet to read — either a sheet name (character) or sheet number (integer). Defaults to the first sheet. If the file has multiple sheets and sheet is not specified, a message lists the available sheets.
preserve.udm	Logical. If TRUE (default), user-defined missing values arriving with the file are preserved: SPSS-style codes such as -99 keep their original numeric values in the data frame, with metadata attached so the package's analysis functions still treat them as missing, and Stata-style tagged values (.a, .b, ...) are kept as read. If FALSE, both forms are converted to plain NA on import and the metadata is stripped. Applies to any loaded file whose columns carry missing-value declarations — typically .sav, .dta, and .sas7bdat files, and .rds files saved from such data. For .sav files, TRUE corresponds to haven's user_na = TRUE.
udm.notice	Per-call override for the user-defined missing value (UDM) notification frequency. NULL (default) defers to the global setting from joutput(). TRUE prints the notification on every load; FALSE suppresses it; NULL at the global level shows once per session. See ?joutput for the full toggle behavior.
quiet	Logical; default FALSE. When TRUE, suppresses jload()'s informational messages (the directory-resolution note, file found, load summary, default-data note, and the UDM narrative, overriding udm.notice). Errors, warnings, the multi-sheet advisory, and the overwrite prompt are still shown.

Details

File paths: Use forward slashes (/) in file paths. If you copy a path from Windows File Explorer, replace the backslashes with forward slashes. R does not accept single backslashes in file paths.

File search order:

1. If the path contains a directory separator (/), the path is used directly.
2. If the path is a bare filename, jload() checks: (a) the folder named by joptions("data.dir") if it is set and exists; (b) the working directory.

Auto-naming: The data frame name is derived from the filename by stripping the extension. If the resulting name starts with a digit (which R does not allow as a variable name), you must supply the `name` argument.

Excel files: Excel files (`.xlsx`, `.xls`) do not contain variable or value labels. The data will be loaded as plain numeric, character, or logical columns. Use `jrelabel()` to add labels after loading if needed.

Coded missing values: When `check.missing = TRUE`, the function scans numeric variables for values that appear to be coded missing values. Only whole-number values are considered (coded missing values are always integers like -99, 999, etc.). Two detection methods are used:

- For SPSS files, user-defined missing values stored in the file metadata are reported with high confidence.
- A heuristic scan detects negative values among otherwise positive data and extreme outlier values (5x the range of other values).

Detected values are reported but not changed. Use `jrecode` to convert them to `NA` if needed.

Value

Invisibly returns the loaded data frame. The primary effect is assigning the data frame in the calling environment.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
## Not run:
# SPSS
jload("community.sav")
jload("community.sav", use = TRUE)
jload("community.sav", name = "MySurvey")

# Stata
jload("community.dta")

# SAS
jload("community.sas7bdat")
jload("community.xpt")

# Excel
jload("community.xlsx")
jload("community.xlsx", sheet = "Wave2")
jload("community.xlsx", sheet = 2)

# CSV and R native
jload("community.csv")
jload("community.rds")

# Extension omitted -- jload searches for a matching file automatically
```

```
jload("community")

# Full file path
jload("C:/Projects/Data/community.dta")

# Quiet load (e.g. in a .Rprofile or startup script): suppresses the
# informational messages while still loading. Errors and warnings still show.
jload("community.rds", name = "MyData", quiet = TRUE)

## End(Not run)
```

`jlogistic` *Logistic regression with SPSS-style output*

Description

Fits a binary logistic regression using `stats::glm()` with `family = binomial` and prints formatted output including an omnibus model test, model summary statistics, and a coefficients table with odds ratios (Exp(B)).

Usage

```
jlogistic(
  formula,
  data,
  subset = NULL,
  variable.id = NULL,
  numeric = NULL,
  categorical = NULL,
  count = NULL,
  ci = NULL,
  classification = FALSE,
  diagnostics = NULL,
  ref.categories = NULL,
  full = FALSE,
  case.processing.detail = NULL,
  digits = NULL,
  ...,
  value.id = NULL
)
```

Arguments

`formula`	A model formula, e.g. `DV ~ IV1 + IV2`. The DV must be a binary variable coded 0/1.
`data`	A data frame containing variables referenced in `formula`.

`subset` An optional unquoted logical expression (e.g. `Group == 1`) to subset cases for this call only.

`variable.id` Character or NULL. Variable label display mode: one of `"both"`, `"names"`, `"labels"`, `"legend"`, or `"legend.bottom"`. `"names"` shows variable names only; `"both"` shows `"name: label"`; `"labels"` replaces each coefficient's variable name with its label in the Coefficients table (factor level decoration is preserved) – best for short labels; `"legend"` prints a label legend just below the Coefficients table (at the coefficients/fit seam); `"legend.bottom"` prints it at the very end. NULL (default) defers to `joutput()`'s `variable.id` setting. Not a logical.

`numeric` Optional character vector of variable names to treat as continuous even if they have value labels.

`categorical` Optional character vector of variable names to treat as categorical even if they lack value labels.

`count` Optional character vector of independent-variable names to treat as counts for this call (the per-call counterpart of `jcount()`). A count predictor is numeric-like, so it enters the model exactly as `numeric` would; the argument is provided for symmetry with the other analysis functions. The binary dependent variable is fixed, so naming it here has no effect. A variable cannot be listed in both `count` and `categorical`.

`ci` Logical or NULL. If TRUE, adds 95% confidence intervals for Exp(B). If NULL (default), defers to `joutput()`.

`classification` Logical. If TRUE, prints a classification table showing predicted vs observed outcomes. Default is FALSE.

`diagnostics` Logical, character vector, or NULL. If TRUE, prints VIF table. If a character vector, `vif` is currently the only supported option. If NULL (default), defers to `joutput()`.

`ref.categories` Logical or NULL. Per-call override for showing the reference-categories block (the baseline level dropped from each set of dummy variables). `NULL` (default) defers to `joutput()`'s `ref.categories` setting. Applies to `jlm()` and `jlogistic()` only, since they are the functions that produce dummy-coded coefficient tables.

`full` Logical. If TRUE, turns on ci, classification, and diagnostics. Does not override explicit FALSE values.

`case.processing.detail`
Per-call override of the Case Processing Summary detail tier: one of `"none"`, `"totals"`, or `"per_code"`. `NULL` (default) uses the active `joutput()` level default.

`digits` Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

`...` Reserved for argument-name checking. Passing `which`, `plots`, or `show` will produce a helpful error suggesting `diagnostics` instead.

`value.id` Character or NULL. Value-label display mode for the dummy category rows in the Coefficients table: one of `"both"` (`"code: label"`, degrading to a bare code where a code has no label), `"values"` (the bare code), or `"labels"` (the value label, degrading to the bare code where none exists). The reference category folded into each grouped variable's header follows the same mode. `"legend"` and `"legend.bottom"` are not supported here: a coefficient table already pairs each value label with its row, so a separate legend block would only duplicate it. Passing either explicitly is an error; a `joutput()` default of `"legend"` or `"legend.bottom"` is tolerated and rendered as `"both"`, so it does not break a bare call. Variables with no value labels render identically under all supported modes. NULL (default) defers to `joutput()`'s `value.id` setting. Applies only to multi-category dummy predictors; continuous and single-contrast (dichotomous) predictors are unaffected. Not a logical.

Details

The dependent variable must be coded 0/1. If it is not, the function stops with a clear error message and suggests the appropriate `jrecode()` command.

Handles haven-labelled variables, registered dummy variables via `jdummy()`, and the `numeric`/`categorical` overrides in the same way as `jlm()`.

Value

Invisibly returns a list of class `jst_logistic` containing:

model The fitted `glm` object.

model_type Character string `logistic`.

model_frame The model frame used to fit the model.

formula_used The formula after dummy expansion.

coefficients Formatted coefficient table (data frame).

coefficients_raw Flat data frame of raw, full-precision coefficient statistics (one row per coefficient): `term` (machine key, shared with jlm), `b`, `SE`, `Wald`, `df`, `p`, `exp_b`, and `exp_ci_lower` / `exp_ci_upper` odds-ratio CI bounds (present regardless of the `ci` display toggle). Carries an `outcome` attribute.

fit_raw List of raw, full-precision model-level fit statistics: `ll_model`, `ll_null`, `deviance`, `null_deviance`, the omnibus likelihood-ratio test (`chi_sq`, `omnibus_df`, `omnibus_p`), Cox & Snell and Nagelkerke pseudo R-squared (`cox_snell_r2`, `nagelkerke_r2`), `aic`, and `n`.

nagelkerke_r2 Nagelkerke pseudo R-squared.

cox_snell_r2 Cox & Snell pseudo R-squared.

neg2ll -2 Log Likelihood.

aic Akaike Information Criterion.

omnibus Named vector: chi_square, df, p.

n Number of observations.

predicts Character string describing what the model predicts.

dummy_coef_names Names of dummy variable columns.

ref_cats Reference category descriptions.

vif Named numeric vector of VIF values, or NULL.

sample_info Pipeline and missing data counts.

See Also

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame -- Volunteer is already coded 0/1
jlogistic(Volunteer ~ Income + Age, data = community)

# A 1/2-coded dichotomy (Yes = 1, No = 2) must be recoded to 0/1 first
df <- community
df$OwnsHome01 <- jrecode(df, OwnsHome,
                         map = "1=1; 2=0", labels = "0=No; 1=Yes")
jlogistic(OwnsHome01 ~ Income + Age, data = df)

# Using juse() default
juse(community)
jlogistic(Volunteer ~ Income + Age)

# CATEGORICAL PREDICTORS
#
# Per-call: categorical = ... applies for one call only and does not
# persist.
jlogistic(Volunteer ~ Region + Age, categorical = "Region")

# The recommended approach for repeated analyses: register the variable
# with jdummy() before running jlogistic(). This sets categorical
# treatment persistently across subsequent analyses.
jdummy(community, Region)
jlogistic(Volunteer ~ Region + Age)

# To choose a non-default reference category:
jdummy(community, Region, ref = "West")
jlogistic(Volunteer ~ Region + Age)

# FORCING NUMERIC TREATMENT
#
# Use numeric = ... when a labelled variable should enter as a score.
jlogistic(Volunteer ~ Age + Education, numeric = "Education")

# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
jdummy(community, NULL)
juse(NULL)
```

jnumeric *Register variables as numeric for analysis*

Description

`jnumeric()` tells jstats to treat one or more variables as numeric (continuous) wherever their analysis class matters, overriding the package's automatic structural guess. It is the counterpart to `jdummy` (categorical) and `jcount` (count): a variable carries exactly one registered intent at a time, so registering it as numeric clears any prior dummy or count registration. Registration changes no data and assigns nothing – you do not write `df <- jnumeric(...)`. It is stored for the session, keyed by the data frame's name; save the data frame in R format (.rds) to keep it across sessions.

Usage

```
jnumeric(data, ..., remove = FALSE, clear.all = FALSE)
```

Arguments

`data`	A data frame, or omitted to use the `juse` default. `jnumeric(NULL)` clears the numeric registrations on the `juse` default frame (or, with no default set, the only frame that carries them; if several do, it asks rather than wiping all). `jnumeric(data, NULL)` clears that one frame's numeric registrations. Called with no arguments, `jnumeric()` lists the session's numeric and count registrations.
`...`	One or more unquoted variable names to register.
`remove`	Logical; if `TRUE`, remove the numeric registration for the named variables instead of adding it.
`clear.all`	Logical; if `TRUE`, clear numeric registrations on every data frame that carries them.

Details

The typical use is a small-range whole number that the structural classifier would treat as categorical (e.g. a 0-6 attitude item) but that you want analyzed as a continuous score.

Value

`invisible(NULL)`. Called for its side effect on the session registration notebook.

See Also

`jdummy`, `jcount`

Examples

```
# Treat a labelled Likert item as a continuous score (slope-per-unit)
jnumeric(community, Environment2)             # one labelled 1-5 item
jnumeric(community, Environment2, Environment4)  # several at once
jnumeric(community, Environment2, remove = TRUE) # unregister one
jnumeric()                           # list all registrations
jnumeric(community, NULL)            # clear community's numeric registrations
jnumeric(clear.all = TRUE)           # clear every frame's numeric registrations
```

`joptions` *Set or display session-level package options*

Description

Controls session-wide settings that affect how the package handles missing-value information and related conventions. `joptions` complements `joutput`: joutput governs output verbosity and tiering, while joptions holds session-wide conventions plus a small number of per-function display defaults (currently the `jcorr()` cell layout). Settings are read fresh on each function call: changing a setting after data has been loaded does not retroactively transform data already in memory. `jconvert` is the explicit transform path for data already in the workspace.

Usage

```
joptions(
  missing.convention = NULL,
  udm.convention.codes = NULL,
  data.dir = NULL,
  corr.layout = NULL,
  quiet = FALSE
)
```

Arguments

`missing.convention`	One of `"none"`, `"spss"`, or `"stata"`. See Slots.
`udm.convention.codes`	Numeric vector, length 1 to 3. See Slots.
`data.dir`	Character string (length 1), or `NULL`. See Slots.
`corr.layout`	One of `"wide"` or `"stacked"`, or `NULL`. See Slots.
`quiet`	Logical; default FALSE. When TRUE, joptions() applies the change silently, suppressing both the status panel and the convention nudge. A bare joptions() status query always prints regardless of quiet.

Value

Invisibly returns `NULL`. Called for the side effect of updating session options and printing the status panel.

Slots

missing.convention Character, length 1. One of `"none"`, `"spss"`, or `"stata"`. Default: `"none"`. `"none"` preserves loaded data as-is (no automatic conversion between user-defined missing value (UDM) representations at load time). `"spss"` or `"stata"` opts into load-time auto-conversion via `jload`, and also supplies the target convention for fresh UDM declarations on columns with no existing convention.

udm.convention.codes Numeric vector, length 1 to 3, whole numbers, no duplicates. Sign unconstrained. Default: `c(-99, -98, -97)`. The recommended UDM code set used by `jconvert` when translating Stata-style missing values (`.a`, `.b`, `.c`, `.d`) into SPSS-form numeric codes, and by the load-time diagnostic for convention-matched detection.

data.dir Character string (length 1), or `NULL`. Default: `NULL`. When `NULL`, `jsave` writes bare-filename saves to the working directory and `jload` searches the working directory. When set, names a folder (relative to the working directory) used as both the save target for bare-filename saves and as the first directory searched on bare-filename loads. The folder is auto-created on first save if it doesn't already exist (nested paths are created in full). To clear a previously-set folder back to this default, pass `data.dir = ""` (an empty string); passing `data.dir = NULL` leaves the current setting unchanged (see Call patterns). Filenames containing a directory separator (`/`) bypass this setting and are taken literally.

corr.layout Character, length 1. One of `"wide"` or `"stacked"`. Default: `"wide"`. The default cell layout for `jcorr` when three or more variables are correlated: `"wide"` puts r and p on one line with N beneath; `"stacked"` stacks r, p, and N on three lines for a narrower table that fits more variables. A per-call `layout` argument to `jcorr()` overrides this. It lives here rather than in `joutput` because it is specific to one function's output, not a tiered analysis-content toggle.

Call patterns

`joptions()` Print the current settings panel.

`joptions(NULL)` Reset all slots to defaults, then print the panel.

`joptions(slot = value, ...)` Set one or more slots, then print the panel. Passing `slot = NULL` as a named argument leaves that slot at its current value – useful for setting one slot without touching another. To reset a single slot to its default, pass the default value explicitly (e.g. `joptions(missing.convention = "none")`). Because `data.dir`'s default is `NULL` – which already means "leave alone" – it is cleared instead with `data.dir = ""`.

Environment-scan notice

Setting `missing.convention` to `"spss"` or `"stata"` triggers a one-time scan of `globalenv()` for data frames whose predominant UDM convention differs from the newly-set value. When mismatches exist, a one-line notice lists the affected data frames and suggests `jconvert` for alignment. The notice is informational; nothing is changed. Plain data frames with no UDM-bearing columns – including the course datasets in their standard form – do not trigger the notice.

See Also

`joutput` for output-verbosity settings; `jstats` for the package overview.

Examples

```
joptions()                                        # show current settings
joptions(missing.convention = "spss")             # set, panel, nudge
joptions(udm.convention.codes = c(-99, -98))      # set, panel, no nudge
joptions(data.dir = "Data")                       # set save/load folder
joptions(missing.convention = "stata",
         udm.convention.codes = c(-99, -98, -97)) # set both
joptions(missing.convention = "spss",
         udm.convention.codes = NULL)             # set mc, leave codes
joptions(NULL)                                    # reset all to defaults
```

---

joutput *Set session-level output verbosity*

Description

Controls what analysis functions display by default. Three preset levels are available, and individual toggles can override specific settings within any level. Per-call arguments on analysis functions always take precedence over joutput() settings.

Usage

```
joutput(
  level,
  effect.size = NULL,
  regression.ci = NULL,
  means.ci = NULL,
  levene = NULL,
  posthoc = NULL,
  diagnostics = NULL,
  case.processing = NULL,
  case.processing.detail = NULL,
  variable.id = NULL,
  value.id = NULL,
  ref.categories = NULL,
  udm.notice = NULL,
  digits = NULL,
  quiet = FALSE
)
```

Arguments

`level` Character. One of `minimal`, `standard` (default), or `full`. If omitted, prints the current settings. If `NULL`, resets to defaults (standard with no toggle overrides).

minimal Stripped-down output for power users. Core results only – no Case Processing Summary, no variable labels, no reference categories, no effect sizes, no CIs.

standard Default. Suitable for teaching and routine use. Includes Case Processing Summary, reference categories, effect sizes, and confidence intervals for means and mean differences (`jt`, `jaov`); regression coefficient CIs (`jlm`, `jlogistic`) are reserved for full. Variable labels are off by default (`variable.id = "names"`); request a label legend or in-table labels per call or via the `variable.id` toggle.

full Everything in standard plus a variable label legend (`variable.id = "legend"`), regression coefficient confidence intervals, assumption checks (Levene's test), post-hoc tests, regression diagnostics, and the most detailed Case Processing Summary (per-code missing breakdown).

`effect.size` Logical or NULL. Override the level's default for effect size display.

`regression.ci` Logical or NULL. Override the level's default for confidence intervals on regression coefficients (`jlm`, `jlogistic`). Off at minimal and standard, on at full.

`means.ci` Logical or NULL. Override the level's default for confidence intervals on means and mean differences (`jt`, `jaov`). Off at minimal, on at standard and full.

`levene` Logical or NULL. Override the level's default for Levene's test display.

`posthoc` Logical or NULL. Override the level's default for post-hoc test display (jaov only).

`diagnostics` Logical or NULL. Override the level's default for regression diagnostic output (jlm only).

`case.processing`
Three-state toggle. `TRUE` forces the Case Processing Summary to print on every call. `FALSE` suppresses it on every call. `NULL` (the auto-suppress default at the standard tier) prints only when the call had something to report – pipeline state was active (`jsubset`, `jcomplete`, or per-call `subset`), listwise deletion excluded at least one case (in listwise functions like `jlm`, `jt`), or a per-variable discrepancy notification fires (in `jdesc`/`jfreq`). The minimal tier sets this to `FALSE`; the full tier sets it to `TRUE`; the standard tier sets it to `NULL`.

`case.processing.detail`
Detail tier for the Case Processing Summary's missing-data breakdown: `"none"` (no bottom table), `"totals"` (one summed missing row per variable), or `"per_code"` (per user-defined missing value code plus system-missing). The minimal tier defaults to `"none"`, standard to `"totals"`, full to `"per_code"`.

`variable.id` Character or NULL. Variable label display mode, one of `"both"`, `"names"`, `"labels"`, `"legend"`, or `"legend.bottom"`. `"names"` shows variable names only; `"both"` shows `"name: label"`, with no labels block. `"labels"` replaces variable names with their labels in the analysis output itself (table rows, captions, crosstab dimnames, or `jplot` axis/legend titles) – best when labels are short. `"legend"` keeps names in place and prints a label legend at the function's mid position (for `jlm`/`jlogistic` between the coefficients and fit blocks; for `jfreq` under each variable's own table; elsewhere directly after the single table). `"legend.bottom"` keeps names in place and prints one consolidated legend at the very end of the output. The minimal and standard tiers default to `"none"`; the full tier defaults to `"legend"`. Not a logical – `TRUE`/`FALSE` are not accepted.

`value.id` Character or NULL. Value-label display mode for the categorical levels that appear in `jfreq` valid rows, the `jt`/`jaov` group descriptives, the `jcrosstab` axes, and the grouped `jdesc` headers. One of `"both"` (`"code: label"`, degrading to a bare code where a code has no label), `"values"` (the bare stored code), or `"labels"` (the value label, degrading to the bare code per code where none exists). `"legend"` and `"legend.bottom"` keep the bare code in the table and print a value-label legend after it (`"legend"` per-table, `"legend.bottom"` consolidated where multiple tables are produced). Variables with no value labels render identically under all three modes, so this is a no-op for plain numeric data. The minimal tier defaults to `"values"`; the standard and full tiers default to `"both"`. Distinct from `variable.id`, which governs the one-per-variable descriptive label. Not a logical.

`ref.categories` Logical or NULL. Override the level's default for the reference categories block (registered dummies).

`udm.notice` Three-state toggle controlling the user-defined missing-value (UDM) notification emitted by `jload()` for `.sav` files. `TRUE` prints the notification on every load that involves UDM-bearing variables; `FALSE` suppresses it entirely; `NULL` ("auto") prints it only the first time in a session, then suppresses it. Standard level uses `NULL` (auto), minimal uses `FALSE`, full uses `TRUE`.

`digits` Integer or NULL. Number of decimal places shown for continuous statistics in the analysis-function output tables (range 0-7; `digits = 0` prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. All three preset levels default to 3.

`quiet` Logical; default FALSE. When TRUE, joutput() applies the level/toggle change silently (the status panel is not printed). A bare joutput() status query always prints regardless of quiet.

Value

Invisibly returns NULL. Called for its side effect of setting session options.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
joutput("standard")                      # effect sizes + means/diff CIs (jt, jaov)
joutput("standard", regression.ci = TRUE) # also show jlm/jlogistic coefficient CIs
joutput("full")                          # everything
joutput()                                # show current settings
joutput(NULL)                            # reset to defaults
```

jplot *Visualise jst_* result objects or plot variables directly from a data frame*

Description

Unified plotting function. Can be called in three ways:

Usage

```
jplot(x, which = "core", ...)

## Default S3 method:
jplot(
  x,
  ...,
  by = NULL,
  type = NULL,
  line = FALSE,
  equation = TRUE,
  r2 = TRUE,
  band = "ci",
  subset = NULL,
  labels = NULL,
  numeric = NULL,
  categorical = NULL,
  count = NULL
)

## S3 method for class 'jst_lm'
jplot(
  x,
  which = "core",
  focal = NULL,
  at = "zero",
  equation = TRUE,
  r2 = TRUE,
  ...
)

## S3 method for class 'jst_logistic'
jplot(x, which = "core", focal = NULL, at = "zero", ...)

## S3 method for class 'jst_ttest'
jplot(x, which = "core", ...)

## S3 method for class 'jst_anova'
```

```
jplot(x, which = "core", ...)

## S3 method for class 'jst_corr'
jplot(x, which = "core", ...)

## S3 method for class 'jst_crosstab'
jplot(x, which = "core", ...)

## S3 method for class 'jst_desc'
jplot(x, which = "core", ...)

## S3 method for class 'jst_freq'
jplot(x, which = "core", ...)
```

Arguments

x	A result object from one of the package's analysis functions (result-object form), or a data frame (data-first form).
which	Character vector. core (default), all, or one or more specific plot names valid for the object's class. (Result-object form only.)
...	Additional arguments: for the result-object form these are passed to class-specific methods; for the data-first form these are unquoted variable names (1 or 2).
by	Unquoted variable name for group-coloring (data-first form).
type	Character. Plot type override for the data-first form. One of histogram, bar, scatter, box, grouped_bar. If NULL (default), auto-detected from variable types.
line	Controls a line overlay on data-first scatter plots. One of FALSE (default; no line), TRUE (alias for lm), lm, loess, connect.
equation	Logical. If TRUE (default), displays the equation in the subtitle for line = "lm" scatter plots (data-first form) or jst_lm fit plots (result-object form).
r2	Logical. If TRUE (default), displays R-squared in the subtitle alongside the equation.
band	Character. Uncertainty band type for line = "lm" scatter plots. One of ci (default; 95% confidence band for the mean, flares at the ends), pi (95% prediction interval for individual observations), see (constant-width band at +/- t*SEE; useful for teaching homoskedasticity), none.
subset	Optional unquoted logical expression to filter cases for this call only (data-first form).
labels	Character or NULL. Variable label display mode (data-first and formula forms): one of "both", "names", "labels", "legend", or "legend.bottom". "names" uses variable names as axis/legend titles; "labels" uses each variable's label as its axis/legend title instead (falling back to the name when unlabelled) and prints no console legend; "legend" and "legend.bottom" keep names on the axes and print a console label legend. "both" is accepted but currently renders as "names" on plots (the "name: label" form for plot titles is deferred to a later phase). NULL (default) defers to joutput()'s variable.id setting. Not a logical.

numeric	Optional character vector of plotted-variable names to treat as continuous for this call (the per-call counterpart of jnumeric()). In jplot() a variable's class chooses the geometry, so this forces numeric handling (histogram for a single variable; scatter / numeric axis in the formula and two-variable forms). Applies to the plotted variables only, not the by grouping variable.
categorical	Optional character vector of plotted-variable names to treat as categorical for this call (the per-call counterpart of jdummy() for plotting purposes). Forces categorical geometry (bar for a single variable; box / categorical axis in the formula form). A variable cannot be listed in both categorical and numeric/count.
count	Optional character vector of plotted-variable names to treat as counts for this call (the per-call counterpart of jcount()). A count is numeric-like for plotting, so it draws the same as numeric; it is provided for symmetry with the other analysis functions.
focal	Unquoted name of the independent variable to place on the x-axis for jst_lm / jst_logistic fit and probability plots. Defaults to the first IV in the model.
at	Character string or named list specifying where non-focal independent variables are held when drawing the fitted line in jst_lm / jst_logistic methods. One of zero (default), mean, mixed (categorical at 0, interval at mean), or a named list list(Var1 = value, ...).

Details

Result-object form: Pass a result object returned by one of the package's analysis functions. Produces appropriate plots for each class of result (see valid plot names below).

Formula form (for plots that distinguish DV from IV): Pass a formula as the first argument, followed optionally by a data frame. Used for scatterplots and boxplots, consistent with the formula syntax of jlm(), jaov(), and jt(). The DV on the left of ~ goes on the y-axis; the IV on the right goes on the x-axis. Only single-IV formulas are supported here; for multi-IV models, fit with jlm() and pass the result to jplot().

Variable-list form (for distributions and counts): Pass a data frame followed by one or two unquoted variable names. Used for histograms (1 numeric), bar charts (1 categorical), and grouped bar charts (2 categorical). Calls that would otherwise auto-detect to a scatter or boxplot produce a helpful error directing you to the formula form.

Supports pipeline integration (jsubset, jcomplete, per-call subset), grouping via by = , and regression lines with equation/R-squared/band annotations.

Valid plot names by class (for the result-object form):

- jst_lm: fit, predicted, effects, coef, vif, residuals, qq, scale, cooks, leverage
- jst_logistic: probability, roc, calibration, binned, cooks, leverage, coef, vif
- jst_ttest, jst_anova: box
- jst_corr: heatmap, scatter (scatter requires exactly 2 variables in the correlation)
- jst_crosstab: bar

The shortcut keyword core (default) produces a curated default set for the class; all produces every plot the class supports.

Valid plot types for the data-first form: `histogram`, `bar`, `scatter`, `box`, `grouped_bar`.

Valid `line` values: `FALSE` (default), `TRUE` (alias for `lm`), `lm`, `loess`, `connect`.

Valid `band` values: `ci` (default confidence band around the regression line, flares at the ends), `pi` (prediction interval for individual observations, wider), `see` (constant-width +/- t*SEE band illustrating the homoskedasticity assumption), `none` (no band).

Value

Invisibly, a single `ggplot` object if one plot is produced, or a named list of `ggplot` objects if multiple are produced (result-object form). Invisibly returns the `ggplot` object for the data-first form.

Methods (by class)

- `jplot(default)`: the default method: a scatter or box plot from a formula (`DV ~ IV`), or a histogram or bar chart from a data frame and one or more variables.
- `jplot(jst_lm)`: diagnostic, coefficient (forest), and fitted-effect plots for a `jlm()` linear-regression result.
- `jplot(jst_logistic)`: predicted-probability (S-curve) and coefficient plots for a `jlogistic()` result.
- `jplot(jst_ttest)`: a group-comparison box plot for a `jt()` result, with the group means marked.
- `jplot(jst_anova)`: a group-comparison box plot for a `jaov()` result, with the group means marked.
- `jplot(jst_corr)`: a heat-map of the correlation matrix for a `jcorr()` result, or a scatter plot for a single pair.
- `jplot(jst_crosstab)`: a grouped bar chart of cell counts for a `jcrosstab()` result.
- `jplot(jst_desc)`: (planned) direct plotting of a `jdesc()` result is not yet available; this method points you to the data-first form, for example `jplot(data, Variable)`.
- `jplot(jst_freq)`: (planned) direct plotting of a `jfreq()` result is not yet available; this method points you to the data-first form, for example `jplot(data, Variable)`.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# Result-object form
m <- jlm(WellbeingScore ~ Income + Age, community)
jplot(m)                            # core diagnostics + fit plot
jplot(m, which = "coef")            # coefficient forest plot
jplot(m, which = "fit", focal = Age, at = "mean")

# Formula form (scatter and box)
jplot(WellbeingScore ~ Income, community)              # scatter
jplot(WellbeingScore ~ Income, community, line = "lm")  # + regression line
jplot(WellbeingScore ~ Income, community, line = "lm", band = "see")
```

```
jplot(WellbeingScore ~ Income, community, by = Volunteer, line = "lm")

# Boxplot: assert the grouping variable as categorical (labelled
# variables otherwise enter numerically; jdummy() registration also works)
jplot(WellbeingScore ~ Region, community, categorical = "Region")

# Variable-list form (distributions and counts)
jplot(community, Age)                     # histogram
jplot(community, Region)                  # bar chart
jplot(community, Region, Volunteer,       # grouped bar chart
      categorical = c("Region", "Volunteer"))

# Using juse() default (formula form; omit the data frame)
juse(community)
jplot(WellbeingScore ~ Income)              # scatter
jplot(WellbeingScore ~ Income, line = "lm")  # + regression line
```

`jrecode` *Recode a variable with explicit value mapping and optional labels*

Description

`jrecode()` recodes a variable using a simple map string that specifies how old values should be converted to new values. It is designed for situations where you need to collapse categories, change numeric codes, or recode dichotomies. Variable and value labels are handled automatically.

Map and labels rules can also produce missing values: plain system NA via the `NA` / `System` / `SYSMIS` aliases, or Stata-style tagged missing values (`.a` through `.z`) when the active convention is Stata. See *Missing values in the map* below for the canonical patterns under each convention.

Usage

```
jrecode(data, orig.var, map, labels = NULL, convention = NULL)
```

Arguments

`data`	A data frame containing the original variable.
`orig.var`	The variable to recode (unquoted, e.g. `AgeGroup`).
`map`	A quoted string specifying the recode rules, using the format `"old=new"` with rules separated by semicolons. Multiple old values mapping to the same new value are separated by commas on the left side. An optional `else` clause controls what happens to values not covered by the map: • No else clause: the function stops with a message if any values are left unmapped, so you can fix the map before proceeding. • `else=NA` (also `else=System` or `else=SYSMIS`): unmapped values are deliberately set to system NA.

- `else=copy`: unmapped values are carried across unchanged.
- `else=.a` (or any Stata-style missing-value token, Stata convention only): unmapped values are set to that Stata-style missing value.

Individual values can also be mapped to system NA using the same aliases: `"-5=NA"`, `"-5=System"`, or `"-5=SYSMIS"`.

Under Stata convention, values can be mapped to Stata-style missing-value tokens: `"-99=.a; -98=.b"`.

Examples:

- `"1=1; 2=0"`
- `"1=1; 2,3=2; 4,5=3; else=NA"`
- `"1=1; 2=0; else=copy"`
- `"-5=System; else=copy"`
- `"3=1; 4=2; else=.a"` (Stata convention only)

`labels` Optional. A quoted string specifying value labels for the new variable, using the format `"code=Label Text"` with rules separated by semicolons. If supplied, these labels are used as-is.

The left side of each rule may be a numeric code or, under Stata convention, a Stata-style missing-value token (`.a` through `.z`). Tagged-NA labels are stored on the tag itself, not on a numeric code.

If omitted, the function attempts to transfer value labels automatically from the original variable. This works when the original variable has value labels and the mapping is one-to-one (no categories are collapsed). When categories are collapsed, labels cannot be transferred automatically and a note is printed.

Example: `"1=Male; 0=Female"` or `".a=Refused; .b=Don't know"`.

`convention` Optional. One of `"spss"`, `"stata"`, or `NULL` (default). Controls whether Stata-style missing-value tokens (`.a` through `.z`) are accepted in the map and labels arguments. Inert when no Stata-style missing-value tokens appear in either argument.

When `NULL`, the convention is resolved from `joptions("missing.convention")`; if that is also unset, the default is SPSS. Most users set the convention once at the top of a session via `joptions()` (or in their `.Rprofile`) rather than supplying this argument on every call. See `?joptions` for details.

Details

The function accepts haven-labelled, plain numeric, and factor variables.

The variable label from the original variable is carried across automatically with "(recoded)" appended. If the original variable has no variable label, the variable name is used instead.

Value labels are handled in three ways, in order of priority:

1. If `labels` is supplied, those labels are used as-is.
2. If `labels` is omitted and the original variable has value labels, they are automatically transferred to the new codes — provided the mapping is one-to-one (no collapsing). For example, recoding 1/2 to 1/0 will carry "Yes" and "No" across to the new codes automatically.

3. If categories are collapsed (multiple old values map to one new value), automatic transfer is not possible and a note is printed directing you to supply labels manually.

NA values in the original variable are always set to NA in the new variable, regardless of the `else` setting.

Values that appear to be coded missing values (e.g. -99, -9, 999) from SPSS or another package are automatically detected and set to NA, even when `else=copy` is used. A note is printed when this occurs.

If the map does not include an `else` clause and there are unmapped values in the variable, the function stops with a message listing the unmapped values so you can fix the map before proceeding.

If the map specifies values that do not exist in the original variable, a warning is issued (but the function continues). This helps catch typos in the map string.

Missing values in the map. The package supports two conventions for representing user-defined missing values (UDMs), and the syntax for producing UDMs from `jrecode()` depends on which one is active:

Under **SPSS convention** (the default), UDMs are real numeric codes carrying metadata that flags them as missing. The two-step canonical pattern is:

```
df$EducR <- jrecode(df, Education,
                    map    = "1,2=1; 3=2; 4,5=3; -99,-98=-99",
                    labels = "1=High school or less; 2=Some college; 3=Degree")
df <- jdeclare_udm(df, EducR, codes = c(Refused = -99))
```

The `jrecode()` call assigns the numeric sentinel `-99`; the subsequent `jdeclare_udm()` call attaches the label and flags `-99` as missing. Labeling `-99` inside the `labels` argument is unnecessary — `jdeclare_udm()` owns that label.

Under **Stata convention**, UDMs are typed missing cells marked with Stata-style tags (`.a` through `.z`). The single-call canonical pattern is:

```
df$EducR <- jrecode(df, Education,
                    map    = "1,2=1; 3=2; 4,5=3; else=.a",
              labels = "1=High school or less; 2=Some college; 3=Degree; .a=Refused")
```

Under Stata convention, `jdeclare_udm()` is not needed for this pattern — `jrecode()` handles both the value recoding and the Stata-style missing-value labeling in one call.

Writing Stata-style missing-value tokens while the active convention is SPSS raises an informative error that echoes the user's call rewritten in SPSS-style syntax. Switching the convention session-wide is one line: `joptions(missing.convention = "stata")`.

Value

A `haven_labelled` vector with the recoded values, variable label, and (if supplied or auto-transferred) value labels applied. Assign this to a new column in your data frame: `MyData$AgeGroupR <- jrecode(MyData, AgeGroup, map = "...")`

See Also

jdeclare_udm for declaring user-defined missing values on a column after a recode (the SPSS-style canonical pattern).

jrelabel for applying labels to an existing variable after a recode.

joptions for the session-level missing.convention setting.

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
# Recode with explicit labels (a 1/2 dichotomy to 0/1)
df <- community
df$OwnsHome01 <- jrecode(df, OwnsHome,
                         map    = "1=1; 2=0",
                         labels = "0=No; 1=Yes")

# Collapse categories (must supply labels)
df$RegionR <- jrecode(df, Region,
                      map    = "1,2=1; 3,4=2",
                      labels = "1=North or South; 2=East or West")

# Use else=copy to carry unspecified values across unchanged
df$EducR <- jrecode(df, Education,
                    map    = "5=4; else=copy",
                    labels = "4=Bachelor's degree or higher")

# Use else=NA to deliberately drop unspecified values to system NA
df$EducR2 <- jrecode(df, Education,
                     map    = "4=1; 5=1; else=NA",
                     labels = "1=College degree")

# Convert a specific coded missing value to system NA
df$EducR3 <- jrecode(df, Education, map = "-99=System; else=copy")

# Stata convention: Stata-style missing-value tokens in map and labels
# (single call; convention = "stata" scopes the choice to this call only)
df$EducR4 <- jrecode(df, Education,
                     map    = "1,2=1; 3,4,5=2; else=.a",
                     labels = "1=No college; 2=College; .a=Refused",
                     convention = "stata")

# Using juse() default
juse(df)
df$RegionR2 <- jrecode(Region, map = "1,2=1; 3,4=2",
                       labels = "1=North or South; 2=East or West")
```

jrelabel *Apply variable and value labels to a variable*

Description

jrelabel() attaches a variable label and/or value labels to any variable in a data frame. It is designed as a simple label applicator — it does not recode values or compare variables. Use it to add labels after a recode, to fix missing labels, or to label any variable that needs them.

The function accepts haven-labelled, plain numeric, factor, and character variables. The output is always a haven_labelled vector, which is compatible with all jstats functions.

Both the labels and var.label arguments are optional. If neither is supplied, the function returns the variable unchanged as a haven_labelled vector.

If the variable already has labels, they are silently overwritten when new labels are provided.

Usage

```
jrelabel(data, var, labels = NULL, var.label = NULL)
```

Arguments

data	A data frame containing the variable.
var	The variable to label (unquoted, e.g. StatusR).
labels	Optional. A quoted string specifying value labels using the format "code=Label Text" with rules separated by semicolons. Examples: • "1=Yes; 0=No" • "1=Employed; 2=Unemployed; 3=Student; 4=Retired"
var.label	Optional. A quoted string to use as the variable label (the description shown by jdesc(), jfreq(), etc.). If omitted, any existing variable label is preserved. If the variable has no existing label, no variable label is set.

Value

A haven_labelled vector with the requested labels applied. Assign this back to a column in your data frame: MyData$VarName <- jrelabel(MyData, VarName, ...)

See Also

jrecode for recoding values with optional labels in a single step.

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
# Add value labels after a recode
df <- data.frame(Status = c(1, 2, 1, 2, 1, 2))
df$StatusR <- ifelse(df$Status == 1, 1, 0)
df$StatusR <- jrelabel(df, StatusR, labels = "1=Yes; 0=No",
                       var.label = "Status (recoded)")

# Add just a variable label
df$StatusR <- jrelabel(df, StatusR, var.label = "Employment Status")

# Add just value labels
df$StatusR <- jrelabel(df, StatusR, labels = "1=Yes; 0=No")

# Using juse() default
juse(df)
df$StatusR <- jrelabel(StatusR, labels = "1=Active; 0=Inactive")
```

---

jsave *Save a data frame to a file*

Description

`jsave()` writes a data frame to a file. Supports SPSS (`.sav`), Stata (`.dta`), SAS interchange (`.xpt`), Excel (`.xlsx`), CSV (`.csv`), and R's native `.rds` format.

The file format is determined entirely by the file extension you provide — for example, `"mydata.sav"` saves as SPSS, `"mydata.dta"` saves as Stata, and `"mydata.xlsx"` saves as Excel. Changing the extension changes the format.

By default, `jsave()` writes bare-filename saves to the working directory, matching base R's `saveRDS()` and `write.csv()`. To save into a subfolder, set `joptions(data.dir = "...")` once per session (or in `.Rprofile`). Filenames containing a directory separator (`/`) bypass this setting and are taken literally.

If the `data` argument is omitted, the default data frame set by `juse()` is used.

Usage

```
jsave(data, file, overwrite = FALSE, preserve.udm = TRUE)
```

Arguments

`data`	A data frame (unquoted). If omitted, uses the default set by `juse()`.
`file`	Character string. The filename with extension (e.g. `"mydata.sav"`) or a full file path. Use forward slashes in file paths.
`overwrite`	Logical. If `TRUE`, overwrites an existing file without prompting. If `FALSE` (default), prompts for confirmation in interactive sessions. In non-interactive sessions, stops with an error.

`preserve.udm` Logical. If `TRUE` (the default), missing-value declarations are written as they stand; formats that cannot store them (notably Excel and CSV) drop the metadata, and SPSS-style codes such as -99 then read back as ordinary numbers. If `FALSE`, those codes are blanked to plain NA before writing, so they become empty cells. Mirrors the `preserve.udm` argument of `jload`. The pre-flight checks for the .sav, .dta, and .xpt formats run before this step, so a missing-value form a target format cannot represent is still reported and blocked rather than silently dropped.

Details

File paths: Use forward slashes (`/`) in file paths. If you copy a path from Windows File Explorer, replace the backslashes with forward slashes. R does not accept single backslashes in file paths.

File location:

- If the path contains a directory separator, the file is saved to that exact location.
- If the path is a bare filename and `joptions("data.dir")` is set, the file is saved to that folder (auto-created if it doesn't yet exist).
- If the path is a bare filename and `joptions("data.dir")` is unset (the default), the file is saved to the working directory.

Format notes:

- SPSS (`.sav`) and Stata (`.dta`) preserve variable labels and value labels.
- Excel (`.xlsx`) and CSV (`.csv`) do not preserve variable or value labels.
- R native (`.rds`) preserves the data frame exactly as it exists in R, including all attributes.
- Stata files are written as version 14 format.
- Legacy Excel format (`.xls`) is not supported for saving. Use `.xlsx` instead.

Value

Invisibly returns `NULL`. Called for its side effect of writing a file to disk.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# A runnable save into R's session temporary folder
jsave(community, file.path(tempdir(), "community.sav"), overwrite = TRUE)

## Not run:
# The file extension determines the format ---
# the same data frame can be saved in any supported format
jsave(community, "community.sav")         # SPSS
jsave(community, "community.xlsx")        # Excel
jsave(community, "community.csv")         # CSV
jsave(community, "community.rds")         # R native
```

```
# Stata and SAS formats cannot carry community's SPSS-form missing-value
# declarations -- convert first (jsave() pre-flights this and says so)
jsave(jconvert(community, to = "stata"), "community.dta")   # Stata
jsave(jconvert(community, to = "baseR"), "community.xpt")   # SAS interchange

# Using juse() default
jsave(, "community.sav")

# Full file path
jsave(community, "C:/Output/community.sav")

## End(Not run)
```

`jscreen` *Data screening overview*

Description

Provides a quick overview of a data frame for screening. A red "Data Screening" title is printed first, then a short header block (case and variable counts, cases with missing data, variables with outliers), followed by up to three tables: a Variable Types table (Base R storage type, the jstats analysis-role class, an optional sub-class, an optional classification source, distinct-value counts, and optional central- tendency columns), a Missing Data & Outliers table, and – when variable labels are shown – a Variable Labels table last. Handles haven-labelled and date/time variables gracefully.

Usage

```
jscreen(
  data,
  ...,
  outlier.sd = 3,
  subset = NULL,
  variable.id = NULL,
  value.id = NULL,
  types = TRUE,
  issues = TRUE,
  r.type = FALSE,
  stats = FALSE,
  digits = NULL
)
```

Arguments

`data`	A data frame.
`...`	Optional unquoted variable names to screen. If omitted, all variables in the data frame are screened.

`outlier.sd` Numeric. Number of standard deviations from the mean to flag as potential outliers (Numeric-class variables only). Default is 3.

`subset` An optional unquoted logical expression (e.g. `Group == 1`) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.

`variable.id` Character or NULL. Variable label display mode: one of `"both"`, `"names"`, `"labels"`, `"legend"`, or `"legend.bottom"`. `"names"` shows variable names only; `"both"` shows `"name: label"`; `"labels"` shows labels in each table's Variable column (best for short labels); `"legend"` and `"legend.bottom"` keep names and print a label legend after the tables. NULL (default) defers to `joutput()`'s `variable.id` setting. Not a logical.

`value.id` Not supported by `jscreen()`. The function does not display value labels, so passing this argument is an error. It exists only to return a clear message rather than misreporting the token as a missing variable. Leave at NULL (default).

`types` Logical. If TRUE (default), prints the Variable Types table. Set FALSE to suppress it.

`issues` Logical. If TRUE (default), prints the Missing Data & Outliers table, which lists only the variables that actually have missing data or flagged outliers (clean variables are omitted). Set FALSE to suppress the table entirely. Suppressing `types`, `issues`, and `labels` together leaves only the header block.

`r.type` Logical. If TRUE, adds a "Base R Type" column (numeric / haven_labelled / factor / character / date-time) to the Variable Types table, showing each variable's storage type alongside its jstats class. Default is FALSE: the storage type is expert detail (its main signal is "this variable carries value labels / came from SPSS or Stata"), so it is opt-in rather than shown by default. The returned data frame always includes it regardless of this setting.

`stats` Logical or character. Adds central-tendency columns to the Variable Types table for numeric-like variables. FALSE (default) shows none; TRUE shows both Mean and Median; `"mean"` or `"median"` shows only that one. Numeric and Count variables show both; a numeric dichotomy shows its raw mean and a blank median; N-category and other non-numeric variables are blank. The returned data frame always includes Mean and Median regardless of this setting.

`digits` Integer or NULL. Number of decimal places for the Mean and Median columns. NULL (default) defers to `joutput()`'s `digits` setting (default 3).

Details

The jstats Class column reports how the package treats each variable in analyses (Numeric, Categorical, Numbers-as-text, Date-time, Unsupported), in contrast to the Base R Type column's storage view; the same classification gates outlier screening, so only Numeric-class variables are SD-screened and the Outliers cell is left blank for the rest. Zero counts are shown blank so only affected variables carry numbers; a column (or the whole Missing/Outliers table) is omitted entirely when nothing is flagged, and the header count lines explain the omission.

When at least one variable's class comes from a registration (jnumeric, jcount, or jdummy) rather than the structural guess, a Source column appears. It reads as an exception-marker: "registered" is shown against the registered variables and the structurally classified rows are left blank, so the

registrations stand out at a glance. (The returned data frame still records the literal tier for every row.) Set `stats = TRUE` (or "mean" / "median") to add central-tendency columns for the numeric-like variables: Numeric and Count variables show Mean and Median, while a numeric dichotomy shows the raw mean of its stored codes and a blank median. A numeric dichotomy coded other than 0/1 (e.g. the 1/2 Group-4 coding) is flagged with a "*" on its sub-class cell, since its raw mean is not a proportion; the marker shows even when `stats` is off, surfacing the recode need.

When variable names are supplied, only those variables are screened. When omitted, all variables in the data frame are screened. If a `subset` expression references variables not already in the screening list, they are included automatically.

Value

Invisibly returns a data frame of the screening results, with one row per variable and columns including the Base R type, the jstats `Class` and `SubClass`, the classification `Source` ("registered" or "structural"), distinct-value count, missing count and percentage, the outlier count (NA for non-Numeric variables), and the `Mean` and `Median` (NA where not meaningful: Median is NA for dichotomies, and both are NA for non-numeric-like variables). The returned values are the raw counts; only the printed tables blank zeros and omit clean rows.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jscreen(community)
jscreen(community, outlier.sd = 2.5)

# Show the Base R storage type column
jscreen(community, r.type = TRUE)

# Add Mean and Median columns for numeric-like variables
jscreen(community, stats = TRUE)

# Suppress tables (header block only)
jscreen(community, types = FALSE, issues = FALSE)

# Using juse() default
juse(community)
jscreen()
jscreen(Income, Age, WellbeingScore)
jscreen(Income, Age, WellbeingScore, subset = Volunteer == 1)
```

`jsubset` *Set, activate, deactivate, or clear a per-dataset case-selection expression*

Description

`jsubset()` sets a persistent case-selection expression that is applied automatically by jstats analysis functions when the default data frame (set by `juse()`) is in use. This is analogous to the SPSS FILTER command.

The expression is stored per dataset, so switching `juse()` between datasets preserves each dataset's setting independently.

The expression applies whenever the matching dataset is used, regardless of whether it was supplied via `juse()` or specified explicitly in a function call. To bypass it temporarily without losing it, use `jsubset(off)` before the analysis and `jsubset(on)` afterward. This matches the SPSS FILTER / USE ALL convention.

Expressions use standard R logical operators: `==`, `!=`, `<`, `<=`, `>`, `>=`, `&` (AND), `|` (OR), `!` (NOT), `xor()` (XOR), and `%in%`. Using `=` for equality or the SPSS-style keywords `AND`/`OR`/`NOT` will produce a helpful error suggesting the correct R syntax.

Usage

```
jsubset(data, expr)
```

Arguments

`data`	Optional data frame. If supplied, the expression is stored on that dataset specifically. If omitted, the dataset set by `juse()` is used.
`expr`	A logical expression (e.g. `Age < 40 & Gender == 1`), or one of the following special values: `off` Deactivate the setting but remember the expression. `on` Reactivate a previously deactivated setting. `NULL` Clear the setting entirely (forget the expression). If `expr` and `data` are both omitted, prints the current jsubset status.

Value

Invisibly returns `NULL`. Called for its side effect.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
use(community)
jsubset(Age < 40)                        # Set using juse default
jsubset(community, Age < 40)             # Explicit dataset
jsubset(Age < 40 & WellbeingScore > 50)  # Compound condition
jsubset(off)                             # Deactivate
jsubset(on)                              # Reactivate
jsubset()                                # Check status
jsubset(NULL)                            # Clear entirely
# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
use(NULL)
```

`jsum` *Compute a row-wise sum across multiple variables*

Description

`jsum()` computes the sum of values across multiple variables for each case (row) in the data frame. This is typically used to create composite scores from a set of related items (e.g. summing 6 survey items into a total scale score).

By default, cases with any missing values receive `NA`. Use the `min.valid` argument to allow partial sums — for example, `min.valid = 1` returns the sum of available values as long as at least one item is non-missing.

Variables can be listed individually or using colon notation to select a range of consecutive columns (e.g. `Attitude1:Attitude6`).

Usage

```
jsum(data, ..., min.valid = NULL, var.label = NULL)
```

Arguments

`data`	A data frame, or omit to use the `juse()` default.
`...`	Unquoted variable names. Use colon notation (e.g. `Attitude1:Attitude6`) to select a range of consecutive columns.
`min.valid`	Integer (optional). The minimum number of non-missing values required to compute a sum. If a case has fewer non-missing values, the result is `NA`. If omitted, all values must be non-missing (equivalent to setting min.valid to the number of variables).
`var.label`	Character string (optional). A variable label to attach to the result. If omitted, an auto-generated label is used.

Value

A numeric vector the same length as `nrow(data)`, suitable for assigning to a new column: `MyData$Total <- jsum(Var1, Var2, Var3)`.

See Also

`javg` for computing row-wise means.

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
# Set the default data frame (so you can omit it in function calls)
juse(community)

# Sum three variables (all must be non-missing)
community$EnvTotal <- jsum(Environment1, Environment3, Environment4)

# Sum with partial data allowed (at least 2 non-missing)
community$EnvTotal <- jsum(Environment1, Environment3, Environment4,
                           min.valid = 2)

# Sum using colon range for consecutive columns
community$EnvTotal <- jsum(Environment1:Environment5)

# Mix colon ranges and explicit names (e.g. after reverse-coding an item)
community$Environment2R <- jrecode(community, Environment2,
                                   map = "1=5; 2=4; 3=3; 4=2; 5=1")
community$ScaleTotal <- jsum(Environment1, Environment2R,
                             Environment3:Environment5)

# With a custom variable label
community$ScaleTotal <- jsum(Environment1:Environment5,
                             var.label = "Environment Scale Total")

# With an explicit data frame (instead of using juse default)
community$EnvTotal <- jsum(community, Environment1, Environment3,
                           Environment4)

# Not normally needed. You'd clear a default or registration only to
# undo a mistake, or -- as in this example -- to reset state for testing.
juse(NULL)
```

`jt` *Independent samples or paired samples t-test*

Description

Runs a t-test and prints formatted group descriptives and test results. By default, runs the traditional Student's independent samples t-test assuming equal variances. Optional parameters provide Welch's correction, paired samples, effect size (Cohen's d), Levene's test, and confidence interval for the mean difference. Handles haven-labelled, numeric, and factor grouping variables. For haven-labelled variables, numeric codes are displayed alongside labels in the group descriptives table.

Usage

```
jt(
  formula,
  data,
  paired = FALSE,
  welch = FALSE,
  effect.size = NULL,
  levene = NULL,
  ci = NULL,
  subset = NULL,
  variable.id = NULL,
  value.id = NULL,
  case.processing.detail = NULL,
  full = FALSE,
  digits = NULL
)
```

Arguments

`formula`	A formula of the form `DV ~ Group`.
`data`	A data frame containing variables referenced in `formula`.
`paired`	Logical. If TRUE, runs a paired samples t-test. The two groups must have equal sample sizes. Default is FALSE.
`welch`	Logical. If FALSE (default), runs Student's t-test (equal variances assumed). If TRUE, runs Welch's t-test. Ignored when paired = TRUE.
`effect.size`	Logical or NULL. If TRUE, prints Cohen's d. If NULL (default), defers to `joutput()` session setting.
`levene`	Logical or NULL. If TRUE, prints Levene's test for homogeneity of variance. Ignored when paired = TRUE. If NULL (default), defers to `joutput()`.
`ci`	Logical or NULL. If TRUE, adds 95% confidence interval for the mean difference. If NULL (default), defers to `joutput()`.
`subset`	An optional unquoted logical expression (e.g. `Group == 1`) to subset cases for this call only. Applied after jcomplete and jsubset. Does not affect other function calls.
`variable.id`	Character or NULL. Variable label display mode: one of `"both"`, `"names"`, `"labels"`, `"legend"`, or `"legend.bottom"`. `"names"` shows variable names

only; "both" shows "name: label"; "labels" shows the DV and grouping-variable labels in the table captions (group levels follow the value.id mode) – best for short labels; "legend"/"legend.bottom" keep names and print a label legend after the output. NULL (default) defers to joutput()'s variable.id setting. Not a logical.

value.id — Character or NULL. Value-label display mode for the group descriptives rows: "both" ("code: label"), "values" (bare code), or "labels" (the label, degrading to the bare code where a code has none). "legend" and "legend.bottom" keep the bare code in the table and print a value-label legend after it ("legend" per-table, "legend.bottom" consolidated where multiple tables are produced). A no-op for grouping variables with no value labels. NULL (default) defers to joutput()'s value.id setting. Not a logical.

case.processing.detail — Per-call override of the Case Processing Summary detail tier: one of "none", "totals", or "per_code". NULL (default) uses the active joutput() level default.

full — Logical. If TRUE, turns on effect.size, levene, and ci all at once. Does not override explicit FALSE values.

digits — Integer or NULL. Number of decimal places for continuous statistics in the output tables (range 0-7; digits = 0 prints whole numbers with no trailing decimal point). Does not affect p-values, percentages, or integer quantities (counts, N, degrees of freedom), which keep their own fixed conventions. NULL (default) defers to joutput()'s digits setting (default 3).

Details

A red title identifying the test type is printed first, followed by variable labels (if present), then the results tables.

Value

Invisibly returns a list of class jst_ttest containing: model (the t.test result), model_frame (the analysis data frame used for plotting), test_type, formula, descriptives, t, df, p, mean_difference, ci (95% CI), cohens_d, d_label, n, and sample_info (pipeline and missing data counts).

See Also

jstats for the package overview, workflow conventions, and complete function listing.

Examples

```
# With explicit data frame
jt(WellbeingScore ~ Volunteer, data = community)
jt(WellbeingScore ~ Volunteer, data = community, welch = TRUE)
jt(WellbeingScore ~ Volunteer, data = community, full = TRUE)

# Using juse() default
juse(community)
jt(WellbeingScore ~ Volunteer)
```

```
jt(WellbeingScore ~ Volunteer, full = TRUE)
```

`jupdate` *Update jstats to the latest version*

Description

`jupdate()` installs the most recent version of jstats. While jstats is in its pre-release phase this downloads and installs the latest pre-built version; once jstats reaches CRAN, the same command will update it the ordinary way. Either way, you run one command instead of having to remember an install line. It is safe to call from the console, a script, or a Quarto document.

Usage

```
jupdate(ask = FALSE)
```

Arguments

`ask`	Logical. When `TRUE` and the session is interactive, jupdate() shows the available and installed versions and asks for confirmation before installing. Defaults to `FALSE` (update without prompting), which is also what happens in any non-interactive session, such as a Quarto render.

Details

The function checks for an internet connection first; if jstats is already up to date it says so and stops. The install runs in a separate R process so the copy of jstats loaded in your session does not lock its own files during the install (the usual cause of a failed update on Windows). After a successful update you restart R once to load the new version.

Value

Invisibly `NULL`. Called for its side effect of installing the update, and for the messages it prints.

Examples

```
## Not run:
jupdate()            # update without prompting
jupdate(ask = TRUE)  # confirm before updating

## End(Not run)
```

`juse` *Set or display the default data frame for jstats functions*

Description

`juse()` sets a default data frame that will be used automatically by all jstats functions when the `data` argument is omitted. This reduces typing and makes interactive use more convenient.

The function stores the *name* of the data frame, not a copy of the data. This means any changes you make to the data frame (adding columns, recoding variables, etc.) are automatically reflected in subsequent function calls.

Usage

```
juse(data)
```

Arguments

`data`	A data frame (unquoted). If omitted, prints the current default. Use `juse(NULL)` to clear the default.

Value

Invisibly returns `NULL`. Called for its side effect of setting, displaying, or clearing the default data frame.

Note

`juse()` stores the *name* of the data frame, not a copy of the data. This means any changes you make to the data frame (adding columns, recoding variables, etc.) are automatically reflected in subsequent function calls. This differs from base R's `attach()`, which creates a snapshot that can become stale after modifications. `juse()` is the recommended approach for this package.

See Also

`jstats` for the package overview, workflow conventions, and complete function listing.

Examples

```
juse(community)             # Set community as the default
juse()                      # Display current default
jdesc(Age, WellbeingScore)  # Uses community automatically
juse(NULL)                  # Clear the default
```

Index