RIn

Statistics and Data Exploratory Tools

Complete Reference Manual — Version 1.2.8

A free, portable, open-source data-management, estimation, charting, and causal-inference tool with familiar econometric syntax. Offline-first NLP. Full diff-diff 3.x causal suite. 109 commands. No license server.

@akirawisnu

University of Göttingen

Contents

Part I — Why Rln exists

1. Motivation

Rln exists because empirical social-science research increasingly needs three things at once that no single existing tool delivers well: compact econometric syntax, modern causal-inference estimators, and the ability to handle administrative-scale datasets and free-response text data — all without leaving one analysis environment.

The compact-syntax tradition — gen, replace, if, by, xtreg, didregress — is the lingua franca of applied microeconometrics for good reason. Six tokens express what would take thirty in pandas + statsmodels. Graduate students learn this syntax in their first econometrics course and reach for it for the rest of their careers. But the leading commercial implementation is per-seat licensed, machine-locked, and many current-generation causal estimators are available only as third-party user-contributed packages with inconsistent interfaces.

R and Python have every estimator anyone could want, but using them means context-switching between two languages with different data-frame conventions, and giving up the dense input syntax that makes small data transformations fast to write. "Regress income on education with clustered standard errors" is one line in compact econometric syntax and roughly thirty in Python + statsmodels.

Rln tries to be the middle option: familiar input syntax in front, the Python ecosystem behind. The 130 commands documented in this manual all parse a compact econometric grammar and dispatch to pandas, statsmodels, scipy, linearmodels, diff-diff, polars, matplotlib, or transformers — whichever library is right for the task. Users write familiar syntax and get Python internals for free, without having to learn the Python ecosystem itself.

2. Design principles

- **Free and open-source.** MIT licensed. No per-seat cost, no registration, no telemetry. Source is readable Python; users can inspect any estimator and see exactly what it computes.
- **Portability over install convenience.** The rln/ folder is self-contained. Everything a session needs — NLP models, argos translation pairs, example data — lives inside rln/. Copy the folder onto a thumb drive, move it to a different OS, and it runs. No registration, no machine-specific license, no separate cache directory in the user home.
- **Offline-first where possible.** The neural NLP backend (hf) is powerful but requires gigabytes of weights. For translate and summarize, Rln bundles offline alternatives — nlp translate via argos-translate, nlp summarize via sumy. Once installed, both work on machines without internet.
- **Modern econometrics, built in.** The 2021–2024 wave of staggered-DiD papers (Callaway-Sant'Anna, Sun-Abraham, Borusyak-Jaravel-Spiess, Gardner, Wing et al., Goodman-Bacon, Rambachan-Roth) is implemented through the diff-diff library. All eleven estimators are exposed through a single didregress command with a method() option, not eleven different third-party packages with conflicting conventions.

• **Larger-than-RAM out of the box.** Every core command has a pandas-backed implementation that loads data into memory. The lrtm subsystem adds a parallel polars-backed implementation for use, describe, summarize, tabulate, generate, keep, merge, fuzzmerge, collapse, contract, and save. This lets Rln process multi-gigabyte parquet files with minimal RAM.

• **Scripting fidelity.** script files, local/global macros, foreach/forvalues loops, by/bysort prefixes, quietly, capture, preserve/restore, and r()/e() return registers all behave the way researchers expect from compact econometric syntax. Existing scripts with familiar syntax often run with zero edits.

• **Reasonable defaults, escape hatch to Python.** When Rln's abstractions aren't enough, `python: expr` and `python { ... }` blocks drop into Python with the current dataset bound as `df`. Users never get stuck — they always have the full Python ecosystem one keyword away.

3. Relationship to R and Python

Rln is a thin syntax-and-dispatch layer. The numerical work is done by the same libraries Python users already know — what Rln adds is the unified compact-syntax frontend and the integration glue between them. The table below shows what a typical applied workflow looks like in each environment.

What you want	R	Python	Rln
Load .dta / .csv / .xlsx / .parquet	haven + readr + arrow	pandas + pyreadr + pyarrow	use / import
Regress Y on X with robust SE	lm + sandwich	statsmodels OLS + HC1	regress, robust
Panel FE regression	plm	linearmodels.PanelOLS	xtreg, fe
Callaway-Sant'Anna DiD	did package	differences package	didregress, method(cs)
Honest DiD sensitivity	HonestDiD package	— manual	didregress, method(honest)
Offline translate	argosTranslate	argostranslate	nlp translate
10GB parquet file	arrow::open_dataset	polars	lrtm use
Fuzzy merge	fuzzyjoin	polyfuzz	fuzzmerge
Cost	Free	Free	Free
Source available	Yes	Yes	Yes

Rln is not a clone of any single tool. It does not implement a matrix programming language, a generic ml maximum-likelihood command, a survey-estimation suite, or the thousands of user-contributed packages that exist in adjacent ecosystems. It implements the core ~130-command subset that covers most applied research workflows, and where it departs from convention it does so in favor of modern best-practice alternatives — for example, the diff-diff 3.x DiD estimators instead of older user-contributed implementations.

Part II — Installation and environment

4. Installation

Rln is distributed as a regular Python package folder. There is no installer.

```
# 1. Unzip rln/ wherever you want it to live
#    (the folder is fully self-contained)
cd rln/

# 2. Install dependencies
pip install -r requirements.txt

# 3. Run
python main.py

# Or build a single-file Windows executable
pyinstaller rln.spec
```

Python 3.10 or newer is required. Tested on Python 3.10, 3.11, 3.12, and 3.13 across Windows 10/11, macOS, and Ubuntu 22.04+. Minimum system RAM: 2 GB for interactive use, 8 GB if you want to use the neural NLP backend.

5. Portable model folders

Rln roots every model cache inside the project folder. This is the key portability guarantee: copy rln/, get identical behavior on the destination machine.

Folder	Used by	Contents
rln/hf_models/	hf / nlp (neural)	HuggingFace weights (transformers)
rln/argos_models/	nlp translate	argos-translate .argosmodel packages
rln/examples/	All showcase files	Sample CSVs and script files

6. Running Rln

Three entry points:

Interactive REPL: `python main.py`

Script file batch: `python main.py do script.do`

Single expression: `python main.py -c "use data.dta, clear; describe"`

Part III — Package and dependency inventory

Rln is a thin compact-syntax layer over a carefully-chosen set of Python packages. Knowing which package backs which command helps both when debugging edge cases and when writing a python{} block that has to match Rln's behavior.

7. Core runtime

Package	Version pin	Role in Rln
pandas	>=2.0	Primary in-memory dataframe. Every command that isn't in lrtm mode uses a pandas DataFrame.
numpy	>=1.24	Underpins pandas; exposed as np in python{} blocks.
rich	>=13	Terminal output: tables, colored text, progress bars.
prompt_toolkit	>=3.0	REPL line editing, history, and autocomplete.
textual	>=0.40	TUI for browse (interactive data grid) and doedit (script file editor).

8. Statistics and econometrics

Package	Role
statsmodels	OLS (regress), TWFE fallback for didregress method(twfe), xtreg entity-dummies fallback, t-tests, predict/residuals/test.
scipy	Statistical distributions for p-value computation (t, F, normal, chi-sq). Used wherever Rln reports significance.
linearmodels	Preferred backend for xtreg (PanelOLS with entity and time effects) when installed. Falls back to statsmodels LSDV otherwise.
matplotlib	All charts. histogram, kdensity, scatter, twoway, graph bar/box/line, marginsplot.

9. Causal inference

Package	Role
diff-diff (>=3.0)	Powers didregress method(did|cs|sa|bjs|gardner|stacked|sdid|eventstudy|bacon|honest). A single consistent API across 2021-2024 staggered-DiD estimators plus Rambachan-Roth (2023) sensitivity.

10. NLP

Package	Role
transformers	HuggingFace backend. Powers hf classify, sentiment, summarize, translate, ner, embed.
sentence-transformers	Recommended for hf embed.
torch	Required by transformers for model inference.
argostranslate	Offline translation engine. Powers nlp translate.
sumy	Offline extractive summarization (7 algorithms). Powers nlp summarize.
nltk	Tokenizer and stopword data for sumy. Rln has an offline fallback when NLTK data is missing.
polyfuzz	Fuzzy-string matching backend for fuzzmerge.

11. Large data

Package	Role
polars (>=0.20)	Lazy/streaming dataframe engine. Backs all lrtm subcommands. Reads parquet without loading into RAM.
pyarrow	Parquet and Arrow I/O used by both pandas and polars.

pyreadr	Read/write .dta and .RData files.
dbfread	Read legacy .dbf (dBase) files. Used by import and lrtm convert.
openpyxl	Excel .xlsx read/write. Used by import excel, export excel.

Part IV — Command reference

Every Rln command is documented below with its exact syntax, relevant options, behavior, and at least one worked example. Commands are organized topically. The order within each group roughly follows the dependency graph of a typical analysis: load data, inspect it, transform it, estimate models, visualize, and report.

Notation used below: square brackets [ ] mark optional tokens; italics mark user-supplied names (like varname or filepath); pipe | separates alternatives; ... means "any number of".

12. Data I/O

Rln reads and writes every tabular format a researcher typically encounters. Format is detected by file extension.

use

Syntax:

```
use "filepath" [, clear]
```

Load a dataset into the working DataFrame. Auto-detects the format from the file extension (.dta, .csv, .tsv, .xlsx, .xls, .parquet, .feather, .dbf, .rdata, .rds, .sav, .sas7bdat, .json). Bare names also resolve against the bundled examples/ folder and your most recent project folder, so use "demographics.csv" opens the shipped example without a full path.

Options:

`clear` — Drop any currently loaded data without warning. Required when data is already in memory.

Example:

```
use "examples/demographics.csv", clear
use "survey.dta", clear
use "panel.parquet", clear
```

import

Syntax:

```
import delimited "file" [, delim(,) rowrange(N) varnames(N) encoding(utf-8) clear]
import excel "file" [, sheet("name") cellrange(A1:Z999) firstrow clear]
import html "url_or_file" [, table(N) clear]
```

Read a file with non-default options. Use for CSVs with unusual delimiters, Excel files with sheets, or HTML tables on web pages.

Options:

`delim(c)` — Column delimiter (default: comma for .csv, tab for .tsv).

`sheet(name)` — Excel sheet to read (default: first sheet).

`cellrange(A1:Z999)` — Excel cell range (default: full sheet).

`firstrow` — Use the first row as variable names (Excel).

`table(N)` — Which HTML <table> element to read, 1-indexed.

`encoding(e)` — Character encoding for text files (default: utf-8).

`clear` — Discard currently loaded data.

Example:

```
import delimited "legacy.tsv", delim(tab) clear
import excel "report.xlsx", sheet("Q3") firstrow clear
import html "https://en.wikipedia.org/wiki/List_of_countries_by_GDP", table(2) clear
```

save

Syntax:

```
save "filepath" [, replace]
```

Write the current DataFrame to disk. Format is inferred from the extension. Supported: .dta, .csv, .parquet, .feather, .xlsx, .rdata, .json.

Options:

`replace` — Overwrite without asking if the file exists.

Example:

```
save "clean_panel.dta", replace
save "output/results.parquet", replace
```

export

Syntax:

```
export delimited "file" [, delim(,) replace noquote]
export excel "file" [, sheet("name") replace firstrow(varlabels)]
```

Write to CSV/TSV or Excel with explicit options. For vanilla saves prefer `save`; use `export` when you need non-default options.

Options:

`replace` — Overwrite existing file.

`delim(c)` — Column delimiter.

`noquote` — Do not quote string fields.

`firstrow(varlabels)` — Write variable labels as the first row instead of names.

Example:

```
export delimited "share_me.csv", replace
export excel "report.xlsx", sheet("Sample") replace
```

log

Syntax:

```
log using "file" [, replace append text]
log close
```

Mirror all REPL output to a file. `log close` ends the log.

Options:

`replace` — Overwrite existing log.

`append` — Append to existing log.

`text` — Plain text format (default). The 'smcl' option is accepted but written as plain text.

Example:

```
log using "analysis.log", replace
describe
summarize income
log close
```

13. Exploration

browse

Syntax:

```
browse [varlist] [if condition] [in range]
```

Launch an interactive full-screen TUI data browser (built on textual). Navigate with arrow keys, PageUp/Down, Home/End. Press q to exit. Shows types, labels, and supports sorting by column. Cells are colour-coded by type — numbers in blue, text in orange, missing values muted — using one shared scheme across the desktop GUI, the terminal browser, and the Android app. You can also explore a file directly without loading it: browse "data.parquet" opens a read-only preview (capped for responsiveness) and leaves the in-memory dataset untouched.

Example:

```
browse
browse income age if gender == "Female"

browse "eu_firms_panel.parquet"
```

describe

Syntax:

`describe [varlist]`

Show variable names, storage types, display formats, and labels, plus dataset source and observation count. Without a varlist, describes every variable.

Example:

```
describe
describe income age gender
```

codebook

Syntax:

`codebook [varlist]`

Detailed per-variable summary: type, unique values, missing count, mean/sd/min/max for numeric, top-5 frequencies for strings.

Example:

```
codebook income
codebook
```

list

Syntax:

`list [varlist] [if condition] [in range] [, noobs separator(N)]`

Print observations as a formatted table.

Options:

noobs — Suppress the observation-number column.

separator(N) — Print a horizontal rule every N rows (default 5).

Example:

```
list in 1/10
list name income if income > 50000 in 1/20
list, separator(0)
```

tabulate (alias: tab)

Syntax:

`tabulate var1 [var2] [if condition] [in range] [weight] [, missing sort nolabel]`

Frequency table. One variable: one-way distribution. Two variables: cross-tabulation with row/column totals. Under a weight clause, cell values are sums of weights rather than raw counts.

Options:

`missing` — Include missing values as a category (default: excluded).

`sort` — Sort by frequency descending (default: by value).

`nolabel` — Show codes, not labels, for encoded variables.

Example:

```
tab gender
tab gender marital_status
tab city if region == "West", sort
```

summarize (alias: sum)

Syntax:

```
summarize [varlist] [if condition] [in range] [weight] [, detail]
```

Descriptive statistics (N, mean, sd, min, max) for numeric variables. Under a weight clause, Obs becomes Sum(W) and the mean, standard deviation, and percentiles are weighted using the formula appropriate to the weight type (see section 16.1).

Options:

`detail` — Add percentiles (1, 5, 10, 25, 50, 75, 90, 95, 99), skewness, kurtosis, and four smallest/largest values.

Example:

```
summarize
sum income age if gender == "Female", detail
sum wage [fweight=pop]                          // weighted
sum wage [fweight=round(pop_frac)], detail    // round() in clause
```

tabstat

Syntax:

```
tabstat varlist [if condition] [in range] [weight] [, by(g) stats(mean sd ...)
format(%fmt)]
```

Compact summary statistics — one row per stat, one column per variable. Default stats are n, mean, sd, min, max. Under weights, every stat is weight-aware including all percentiles. With by(g), stats are computed per group.

Options:

`by(g)` — Compute statistics per group of variable g.

`stats(...)` — Space-separated list. Available: n, count, mean, sd, var, min, max, sum, median, range, iqr, and any pN where N is a percentile (p1, p5, p25, p50, p75, p90, p95, p99).

`format(%f)` — Print format for the values (default %.4f).

Example:

```
tabstat income educ
tabstat income educ, by(region) stats(n mean p50 iqr)
tabstat wage [aweight=w], by(region) stats(mean p25 p50 p75 p90)
```

count

Syntax:
`count [if condition] [in range]`

Report the number of observations matching the condition. Without a condition, reports total observation count. Also sets r(N).

Example:

```
count
count if missing(income)
count if inrange(age, 25, 65)
```

14. Variables and transformations

generate (alias: gen)

Syntax:
`generate [type] newvar = expression [if condition]`

Create a new variable. `type` is optional (byte, int, long, float, double, str) — Rln infers dtype from the expression. Expressions may use any built-in or extension function (see Part V).

Example:

```
gen log_income = ln(income)
gen senior = (age >= 65)
gen bigcity = inlist(city, "NYC", "LA", "Chicago")
gen fullname = first_name + " " + last_name
```

replace

Syntax:
`replace var = expression [if condition]`

Modify an existing variable. If no condition is given, every row is updated. Use `replace` (not `gen`) for edits — `gen` errors on existing variables to prevent accidental overwrites.

Example:

```
replace income = income / 1000
replace gender = "F" if gender == "Female"
replace age = . if age > 120
```

rename

Syntax:

```
rename old_name new_name
```

Rename a variable. Both names must be valid identifiers.

Example:

```
rename Q_01 satisfaction
rename inc_usd income
```

drop

Syntax:

```
drop varlist          (drop variables)
drop if condition     (drop observations matching condition)
```

Remove variables or rows. Be explicit which you mean.

Example:

```
drop _fillin _merge
drop if missing(income) | age < 18
drop if year < 2010
```

keep

Syntax:

```
keep varlist          (keep only listed variables)
keep if condition     (keep only matching observations)
```

Inverse of drop. Keep only the variables or rows that match.

Example:

```
keep id age income education
keep if inrange(year, 2018, 2022)
```

label

Syntax:

```
label variable varname "text"
label define lblname 1 "Male" 2 "Female"
label values varname lblname
label list [lblname]
```

Variable labels (displayed by describe) and value labels (a mapping from integer codes to display text).

Example:

```
label variable income "Annual income (USD)"
label define genderlbl 1 "Male" 2 "Female" 3 "Nonbinary"
label values gender genderlbl
label list
```

destring

Syntax:

```
destring varlist, replace [force]
```

Convert string variables to numeric. Unparseable values become NaN.

Options:

`replace` — Replace the original variables in place.

`force` — Convert all numeric-looking values; coerce the rest to NaN without error.

Example:

```
destring price, replace force
```

tostring

Syntax:

```
tostring varlist, replace [format(fmt)]
```

Convert numeric variables to string.

Options:

`format(fmt)` — Printf-style format string (e.g. %.2f, %05d).

Example:

```
tostring id, replace
tostring rate, replace format(%.4f)
```

encode

Syntax:

```
encode stringvar, generate(newvar) [label(lblname)]
```

Encode a string variable into an integer code with an attached value label. Useful for bringing categorical text into an estimation command.

Example:

```
encode country, generate(country_id)
```

order

Syntax:

```
order varlist [, first last after(var) before(var) alphabetical]
```

Reorder columns.

Options:

`first` — Move varlist to the beginning (default).

`last` — Move varlist to the end.

`after(var)` — Position varlist immediately after the named variable.

`before(var)` — Position varlist immediately before the named variable.

`alphabetical` — Alphabetize all columns (ignores varlist).

Example:

```
order id year
order gender, after(age)
order, alphabetical
```

recode

Syntax:

```
recode var (old1 = new1) (old2 = new2) ... [, generate(newvar)]
recode var (1/5 = 1) (6/10 = 2) (else = 3) [, generate(newvar)]
```

Map old values to new values. Ranges (lo/hi) and the special keywords 'else', 'missing', 'nonmissing' are supported.

Options:

`generate(v)` — Store the recoded values in a new variable instead of replacing the original.

Example:

```
recode age (0/17 = 1) (18/64 = 2) (65/max = 3), generate(age_cat)
recode satisfaction (1/2 = 0) (3 = .) (4/5 = 1), generate(happy)
```

reshape

Syntax:

```
reshape long stubnames, i(id_var) j(time_var)
reshape wide stubnames, i(id_var) j(time_var)
```

Pivot between wide and long format. stubnames is a list of variable prefixes (wide) or single variables (long).

Example:

```
* Wide: id income2018 income2019 income2020
* becomes Long: id year income
reshape long income, i(id) j(year)
```

clonevar

Syntax:

```
clonevar newvar = existingvar
```

Exact copy of a variable, including its label and value labels. Equivalent to `gen newvar = existingvar` followed by label-copying.

Example:

```
clonevar income_backup = income
```

split

Syntax:

```
split varname [, parse(sep) generate(stub) limit(N)]
```

Split a string variable on a separator into multiple new variables.

Options:

`parse(c)` — Separator character or string (default: space).

`generate(s)` — Prefix for the new variables (default: original name + 1, 2, ...).

`limit(N)` — Maximum number of splits (default: unlimited).

Example:

```
split fullname, parse(" ") generate(name) limit(2)
split csvfield, parse(",")
```

15. Data operations

sort

Syntax:

```
sort varlist
```

Sort data in ascending order on the given variables (primary, secondary, ...).

Example:

```
sort year month
sort id year
```

gsort

Syntax:

```
gsort [+-]var1 [+-]var2 ...
```

Sort with direction. Prefix `-` for descending.

Example:

```
gsort -income age
gsort -year +state
```

duplicates

Syntax:

```
duplicates report [varlist]
duplicates drop [varlist] [, force]
duplicates tag [varlist], generate(newvar)
duplicates list [varlist]
```

Find, drop, tag, or list duplicate observations (on all variables or a specified subset).

Options:

- **`force`** — Skip the confirmation prompt on drop.
- **`generate(v)`** — Name of the tag indicator (0/1) for duplicates tag.

Example:

```
duplicates report id
duplicates drop id year, force
duplicates tag id, generate(dup)
```

append

Syntax:

```
append using "filename" [, force generate(newvar)]
```

Stack another dataset below the current one. Columns with the same name are concatenated; extra columns in either file become NaN in the other.

Options:

- **`force`** — Proceed even when column types differ; coerces to common type.
- **`generate(v)`** — Create an indicator variable identifying source file (0=original, 1=appended).

Example:

```
append using "wave2.dta", generate(wave)
```

merge

Syntax:

```
merge 1:1 varlist using "file" [, keep(match|master|using|both) generate(_merge)
nogenerate]
merge m:1 varlist using "file"
merge 1:m varlist using "file"
```

Key-based merge. The first argument specifies cardinality of the master:using relationship.

Options:

- **`keep(...)`** — Which observations to keep after merge.
- **`generate(v)`** — Name of merge-status indicator (default _merge).
- **`nogenerate`** — Suppress creation of the merge-status indicator.

Example:

```
merge 1:1 id using "extra_vars.dta", keep(match)
merge m:1 country using "country_gdp.csv"
```

fuzzmerge

Syntax:

```
fuzzmerge varname using "filename" [, threshold(0.8) method(tfidf) generate(newvar)]
```

Approximate-string merge via the best available fuzzy-matching backend. Useful when join keys differ by spelling ("IBM" vs "I.B.M." vs "International Business Machines"). The backend degrades automatically — PolyFuzz, then RapidFuzz, then a pure-Python matcher — so fuzzmerge runs on every edition (the lite desktop has no PolyFuzz; Android has neither PolyFuzz nor RapidFuzz). It reports which backend was used.

Options:

`threshold(x)` — Minimum similarity score in [0, 1] to consider a match (default 0.8).

`method(m)` — Matching algorithm: tfidf (default), rapidfuzz, editdistance, embeddings.

`generate(v)` — Variable to store match score (default: _match_score).

Example:

```
fuzzmerge company_name using "ref_companies.csv", threshold(0.85) method(tfidf)
```

collapse

Syntax:

```
collapse (stat) varlist [if] [in] [weight] [, by(groupvars)]
collapse (stat1) newvar1=var1 (stat2) newvar2=var2 [weight] [, by(groupvars)]
```

Aggregate to a summary dataset. Each (stat) clause specifies the function applied to the following variables. Under a weight clause, every aggregator is computed using the weighted formula appropriate to the stat (weighted mean, weighted sum, weighted SD, weighted percentiles, etc.).

Options:

`by(varlist)` — Compute within groups defined by varlist instead of over the whole dataset.

Example:

```
collapse (mean) income age, by(country year)
collapse (mean) avg_inc=income (sum) total_pop=population (sd) sd_inc=income, by(region)
collapse (mean) wage [fweight=pop], by(industry)        // weighted
```

Notes: *Available stats: mean, median, sum, count, min, max, sd, first, last, p1, p5, p10, p25, p50, p75, p90, p95, p99.*

fillin

Syntax:

```
fillin varlist
```

Fill in all combinations of varlist — if some (year, state) pairs are missing from a panel, fillin adds the missing rows (with NaN elsewhere) and creates _fillin = 1 to flag them.

Example:

```
fillin id year
```

cross

Syntax:

```
cross using "filename"
```

Cartesian product of the current dataset and a using dataset. Every row of master is paired with every row of using.

Example:

```
cross using "quarters.csv"
```

sample

Syntax:

```
sample N [, count]
```

Keep a random sample. N is a percentage (0-100) by default, or an exact count with the `count` option.

Options:

`count` — Interpret N as an absolute row count.

Example:

```
sample 10
sample 500, count
```

16. Estimation

Every estimation command in Rln accepts an `if` condition, an `in` row range, and an optional weight clause. Coefficient and variance vectors are stashed in e() for use by predict, test, lincom, margins, and coefplot.

16.1 Weight clauses

Every command in this section accepts a weight clause in square brackets, anywhere in the command line. The clause has four interchangeable forms by weight type:

Form	Meaning
[fweight=expr]	Frequency weights (integer): the row represents N observations.
[aweight=expr]	Analytic (inverse-variance) weights.
[pweight=expr]	Sampling (probability) weights. Forces robust standard errors automatically.
[iweight=expr]	Generic importance weights, no statistical semantics.

The right-hand side of a weight clause is a full expression — not just a variable name. Any function available to gen/replace works inside the clause, including round() to convert fractional weights to integers required by fweight:

```
summarize wage [fweight=pop]
summarize wage [fweight=round(pop_frac)]    // round() inside the clause
regress y x [aweight=w1 + w2]               // arithmetic
```

```
tabulate region gender [pweight=sqrt(survey_wt)]
```

Under weights, summarize reports Sum(W) instead of Obs, tabulate cells become sums of weights, tabstat percentiles use weighted quantiles, collapse aggregators are weighted, and regression commands route through WLS (for OLS) or statsmodels' freq_weights kwarg (for GLMs). The coefficient point estimates from regress y x [fweight=w] match row-duplicated OLS to floating-point precision — this is verified by the test suite.

16.2 Linear regression

regress (alias: reg)

Syntax:

```
regress depvar indepvars [if] [in] [weight] [, robust cluster(var) noconstant]
```

Ordinary least-squares regression. Prefix variables with `i.` to create dummy-variable sets for categorical inputs. Results stored in e() registers: e(b), e(V), e(N), e(r2), e(rmse), e(F).

Options:

- **`robust`** — HC1 (heteroskedasticity-robust) standard errors.
- **`cluster(v)`** — Cluster-robust standard errors on the given variable.
- **`noconstant`** — Omit the intercept.

Example:

```
reg log_wage education experience i.industry, robust
reg income education, cluster(state)
reg wage educ exper [pweight=surveyweight]
```

ivregress

Syntax:

```
ivregress 2sls depvar exogvars (endog = instruments) [if] [in] [weight] [, robust]
```

Two-stage least-squares instrumental-variable regression. The endogenous variable and its instruments are written in parentheses; exogenous controls go before the parentheses.

Options:

- **`robust`** — HC1 standard errors.
- **`cluster(v)`** — Cluster-robust standard errors.

Example:

```
ivregress 2sls log_wage age (educ = mom_educ dad_educ), robust
```

predict

Syntax:

```
predict newvar [, xb residuals]
```

After an estimation command, write fitted values or residuals to a new column.

Options:

`xb` — Linear prediction (default).

`residuals` — Residuals (observed minus fitted).

Example:

```
reg income education
predict yhat
predict r, residuals
```

test

Syntax:

```
test var1 [var2 ...]
test var1 = 0
test var1 = var2
```

Wald test of linear hypotheses after an estimation command. Reports F (or chi-squared) and p-value.

Example:

```
test education = experience
test i.industry
```

lincom

Syntax:

```
lincom expression
```

Linear combination of estimated coefficients with standard error and confidence interval. Operates on the most recent estimation.

Example:

```
reg log_wage educ exper
lincom educ + 2*exper
lincom educ - exper
```

margins

Syntax:

```
margins [, dydx(*) dydx(varlist) at(var=val) atmeans]
```

Marginal effects or predictive margins after an estimation. With dydx, reports partial derivatives; with at(), reports predictions at user-specified covariate values.

Options:

`dydx(*)` — Average marginal effects of every regressor.

`dydx(varlist)` — Marginal effects of the listed regressors only.

`at(var=val)` — Predict at this value of var (others at sample mean).

`atmeans` — Predict at sample means.

Example:

```
logit employed age educ i.region
margins, dydx(*)
margins, at(educ=16) atmeans
```

16.3 Generalized linear models

logit

Syntax:

`logit depvar indepvars [if] [in] [weight] [, robust cluster(var) or noconstant]`

Binary logistic regression. The outcome must be 0/1.

Options:

`robust` — HC1 standard errors.

`cluster(v)` — Cluster-robust standard errors.

`or` — Print odds ratios instead of raw coefficients.

`noconstant` — Omit the intercept.

Example:

```
logit employed age educ i.region, robust
logit passed study_hours [pweight=sw]
```

probit

Syntax:

`probit depvar indepvars [if] [in] [weight] [, robust cluster(var) noconstant]`

Binary probit regression. Same setup as logit; outcome must be 0/1.

Example:

```
probit defaulted age income credit_score, robust
```

poisson

Syntax:

```
poisson depvar indepvars [if] [in] [weight] [, robust cluster(var) exposure(var) offset(var) irr]
```

Poisson regression for count data. Use exposure() for rate regressions where the natural log of the exposure variable enters as an offset. The irr option prints incidence-rate ratios.

Options:

- **`robust`** — HC1 standard errors.
- **`cluster(v)`** — Cluster-robust standard errors.
- **`exposure(v)`** — Add ln(v) as an offset (rate-of-occurrence model).
- **`offset(v)`** — Add v directly as an offset (assumed already in log scale).
- **`irr`** — Print incidence-rate ratios (exp(coef)).

Example:

```
poisson visits age income, exposure(years) irr
poisson crashes speed_limit i.weather [aweight=traffic], cluster(state)
```

nbreg

Syntax:

```
nbreg depvar indepvars [if] [in] [weight] [, robust cluster(var) exposure(var) offset(var) irr]
```

Negative binomial regression for over-dispersed count data. Same options as poisson; nbreg additionally estimates the dispersion parameter alpha.

Example:

```
nbreg crashes speed_limit i.weather, cluster(state)
```

tobit

Syntax:

```
tobit depvar indepvars [if] [in] [, ll(value) ul(value) robust cluster(var)]
```

Censored regression via custom maximum likelihood. At least one of ll() (lower limit) or ul() (upper limit) is required. Reports sigma (the latent error standard deviation).

Options:

`ll(v)` — Lower censoring limit.

`ul(v)` — Upper censoring limit.

`robust` — Robust standard errors.

Example:

```
tobit hours age educ children, ll(0)         // bottom-coded at zero
tobit score age educ, ll(0) ul(100)            // bounded outcome
```

16.4 Correlations and simple tests

correlate

Syntax:
`correlate varlist [if condition]`

Pearson correlation matrix over listwise-complete observations.

Example:

```
correlate income education age experience
```

pwcorr

Syntax:
`pwcorr varlist [if condition] [, sig star(alpha)]`

Pairwise correlations (each pair uses its own complete-case observations) with optional significance markers.

Options:

`sig` — Print p-values under each correlation.

`star(a)` — Append * to correlations with p < a (default 0.05).

Example:

```
pwcorr income education age, sig star(0.01)
```

ttest

Syntax:

```
ttest var == value
ttest var1 == var2
ttest var, by(groupvar)
```

One-sample, paired, or two-sample (by group) t-test.

Example:

```
ttest income == 50000
ttest pre_test == post_test
ttest income, by(treatment)
```

16.5 Quantile family

Four commands for percentile-based data exploration and outlier treatment. All four are weight-aware: with a weight clause, every percentile is computed from the weighted distribution rather than the unweighted one.

pctile

Syntax:

```
pctile newvar = expr [if] [in] [weight] [, nquantiles(N) percentiles(p1 p2 ...)
genp(name)]
```

Create a new variable whose first k rows hold the percentile cut points of the expression. Default is the median (one cut).

Options:

`nquantiles(N)` — Cuts at N-1 equally-spaced percentiles. nquantiles(4) = quartile cuts.

`percentiles(p*)` — Explicit percentile list, values 0..100.

`genp(name)` — Also create a companion variable holding the p values.

Example:

```
pctile cut = income, nquantiles(4)
pctile cut = wage [fweight=pop], percentiles(10 25 50 75 90) genp(p)
```

xtile

Syntax:

```
xtile newvar = expr [if] [in] [weight] [, nquantiles(N) cutpoints(var) altdef]
```

Assign each row to one of N quantile bins (1..N) based on the expression. Default is quartiles (4 bins).

Options:

`nquantiles(N)` — Number of bins; default 4.

`cutpoints(var)` — Use the first k non-missing values of var as cut points.

`altdef` — Use strict-greater-than tie-breaking (default: >=).

Example:

```
xtile inc_quartile = income, nquantiles(4)
xtile inc_decile = income [pweight=w], nquantiles(10)
```

centile

Syntax:

```
centile [varlist] [if] [in] [weight] [, centile(p1 p2 ...) level(N)]
```

Report specified percentiles with binomial-based confidence intervals. Default is the median. Under weights, point estimates are computed but CIs are suppressed (bootstrap CIs are deferred to a future release).

Options:

`centile(p*)` — Percentiles to report, values 0..100. Default 50.

`level(N)` — Confidence level for the CI (default 95).

Example:

```
centile wage
centile wage educ, centile(10 25 50 75 90) level(99)
```

winsor2 (alias: winsorize)

Syntax:

```
winsor2 varlist [if] [in] [weight] [, cuts(lo hi) suffix(_w) replace trim by(g)]
```

Cap (or drop) extreme values at given percentiles. Default cuts are at the 1st and 99th percentiles. By default, creates new variables with the suffix _w. Use replace to overwrite in place.

Options:

`cuts(lo hi)` — Lower and upper percentiles (default 1 99).

`suffix(str)` — Suffix for the new variables (default _w).

`replace` — Overwrite the original columns.

`trim` — DROP outliers instead of capping them.

`by(group)` — Compute cuts per group.

Example:

```
winsor2 wage                                  // wage_w at p1/p99
winsor2 wage income, cuts(2 98) by(industry)
winsor2 wage [aweight=w], cuts(5 95) replace
winsor2 wage, cuts(1 99) trim                 // drop, do not cap
```

16.6 Post-estimation diagnostics

Diagnostic tests that operate on the most recent estimation result. These read from e() and stash their own output in r() for programmatic access.

vif

Syntax:

`vif`

Variance inflation factors after a regress. Reports VIF and tolerance (1/VIF) for every regressor. A VIF above 10 (some authors prefer 5) is a common warning threshold for multicollinearity. Stores results in r().

Example:

```
reg log_wage educ exper exper_sq age age_sq
vif
```

estat

Syntax:

```
estat <subtest>
  estat hettest         // Breusch-Pagan test for heteroskedasticity
  estat bgodfrey        // Breusch-Godfrey test for serial correlation
  estat imtest          // White's test for heteroskedasticity
  estat ovtest          // Ramsey RESET test for omitted variables
  estat dwatson         // Durbin-Watson autocorrelation statistic
  estat summarize       // summary of the estimation sample
```

Umbrella for post-estimation specification tests after regress. Each subtest computes its own statistic, reports test value and p-value (where applicable), and stores results in r().

Example:

```
reg log_wage educ exper i.region
estat hettest                  // is variance constant?
estat ovtest                   // are we missing covariates?
estat dwatson                  // is there autocorrelation?
```

dwstat

Syntax:

`dwstat`

Durbin-Watson statistic from the residuals of the most recent regress. DW close to 2 indicates no first-order serial correlation; below 1.5 or above 2.5 suggests autocorrelation worth investigating. Equivalent to `estat dwatson` and provided for convenience.

Example:

```
reg log_wage educ exper
dwstat
```

xtserial

Syntax:

`xtserial depvar indepvars [if]`

Wooldridge (2002) test for first-order autocorrelation in the idiosyncratic errors of a panel. Requires xtset to have been run first. Reports F and p-value; rejection means within-unit residuals are serially correlated and panel-robust SE may be needed.

Example:

```
xtset firm_id year
xtserial log_wage educ exper
```

17. Panel and causal inference

xtset

Syntax:

`xtset panelvar [timevar]`

Declare panel structure. panelvar identifies units; timevar (if given) identifies periods. Required before xtreg and used as defaults by didregress.

Example:

```
xtset firm_id year
xtset state year
```

xtreg

Syntax:

`xtreg depvar indepvars [if] [, fe re robust cluster(var)]`

Panel regression. Uses linearmodels.PanelOLS when installed, otherwise statsmodels LSDV with entity dummies.

Options:

- **`fe`** — Entity fixed effects (within estimator).
- **`re`** — Random effects GLS (default).
- **`robust`** — HC-robust standard errors.
- **`cluster(v)`** — Cluster-robust standard errors.

Example:

```
xtreg log_wage education experience i.year, fe cluster(firm_id)
```

didregress

Syntax:

```
didregress (depvar) (treatment) [, group(var) time(var) method(...)
                                   first_treat(var) post_periods(v1 v2 ...)
                                   reference_period(v) aggregate(...)
                                   covariates(v1 v2 ...) M(value) robust]
```

Difference-in-differences estimation, dispatching to one of eleven estimators via the method() option. Per-period coefficient vectors from event-study and dynamic methods are stored in e()['coefficients'] for use by coefplot. See the estimator table below.

Example:

```
xtset state_id year
didregress (gdp_growth) (treated), group(state_id) time(year) ///
    method(cs) first_treat(first_treat_year) aggregate(event_study)
coefplot, title("CS event study")

* Sensitivity analysis (must follow a method(eventstudy) run):
didregress (gdp_growth) (treated), method(eventstudy) post_periods(2018 2019 2020 2021)
didregress (gdp_growth) (treated), method(honest) m(1.0)
```

The eleven didregress methods:

method(...)	Estimator	Backend class
twfe	Two-way fixed effects (baseline)	statsmodels OLS + dummies
did	Basic 2×2 DiD	diff_diff.DifferenceInDifferences
cs	Callaway-Sant'Anna (2021) staggered	diff_diff.CallawaySantAnna
sa	Sun-Abraham (2021) interaction-weighted	diff_diff.SunAbraham
bjs (aliases: imputation, did_imputation, borusyak)	Borusyak-Jaravel-Spiess (2024) imputation	diff_diff.ImputationDiD
gardner	Gardner (2022) two-stage DiD	diff_diff.TwoStageDiD
stacked	Stacked DiD (Wing, Freedman, Hollingsworth 2024)	diff_diff.StackedDiD
sdid	Synthetic DiD (block treatment)	diff_diff.SyntheticDiD
eventstudy	Full event study, pre+post dynamics	diff_diff.MultiPeriodDiD
bacon	Goodman-Bacon decomposition (diagnostic)	diff_diff.BaconDecomposition
honest	Rambachan-Roth (2023) sensitivity on eventstudy	diff_diff.HonestDiD

Rln auto-derives missing columns where possible: if you pass only per-row treated (0/1) but method(cs) needs first_treat, Rln computes it. If method(did) gets a year column instead of 0/1, a binary post indicator is derived from post_periods, reference_period, or a median split (with a warning). Synthetic

DiD and eventstudy need a block (absorbing) treatment indicator; Rln builds ever_treated from per-row treated automatically when the treatment varies within units.

After didregress: Use coefplot (section 18) to visualize results. For event-study and dynamic methods (eventstudy, cs/gardner/stacked with aggregate(event_study)) it produces a forest plot of per-period effects with confidence intervals, a horizontal zero line, and a vertical reference-period marker. For scalar-ATT methods (twfe/did/cs default/sa/bjs/sdid) it produces a single point with CI whisker labeled by the method name.

On trajectory balancing (tjbal): The Hazlett-Xu (2018) trajectory-balancing method is not implemented in Rln. There is no Python implementation of the kernel-balancing algorithm in any major package; porting it would require substantial verification work against the original replication datasets. Users who need trajectory balancing should fall back to the R implementation until a numerically-verified Python version exists.

18. Charts

histogram (alias: hist)

Syntax:

```
histogram var [if] [, bins(N) normal frequency density title("t") color(c)
name(file)]
```

Histogram with optional normal-density overlay.

Options:

- **`bins(N)`** — Number of bins (default 20).
- **`normal`** — Overlay a normal density with matching mean/sd.
- **`frequency`** — Y-axis is counts (default: density).
- **`density`** — Y-axis is density (default).
- **`title(t)`** — Plot title.
- **`color(c)`** — Bar color (name or hex).
- **`name(f)`** — Save to file instead of displaying (e.g. `name(plot.png)`).

Example:

```
histogram income, bins(30) normal title("Income distribution")
```

kdensity

Syntax:

```
kdensity var [if] [, bwidth(N) normal title("t") name(file)]
```

Kernel density plot.

Options:

- **`bwidth(x)`** — Bandwidth (default: Silverman's rule).
- **`normal`** — Overlay a normal density with matching mean/sd.

Example:

```
kdensity log_income, normal title("Log-income density")
```

scatter

Syntax:

```
scatter yvar xvar [if] [, by(var) lfit mcolor(c) msize(N) title("t") mlabel(v)
name(f)]
```

Scatter plot.

Options:

- **`by(v)`** — Facet into subplots by group.
- **`lfit`** — Overlay an OLS fitted line.
- **`mcolor(c)`** — Marker color.
- **`msize(n)`** — Marker size.
- **`mlabel(v)`** — Label each point with values of variable v.

Example:

```
scatter income education, lfit title("Income vs. years of schooling")
```

twoway (alias: tw)

Syntax:

```
twoway (plotspec1) (plotspec2) ... [, title("t") legend name(f)]
```

Multi-layer plot. Each (plotspec) is one layer. Supported plot types: scatter, line, connected, lfit, lfitci, qfit, area.

Example:

```
twoway (scatter income education) (lfit income education)
twoway (line gdp year, sort) (scatter gdp year)
```

marginsplot

Syntax:

```
marginsplot [, title("t") name(f)]
```

Plot the last `margins` result with 95% confidence bars.

Example:

```
reg income education age
margins, at(education=(8 12 16))
marginsplot
```

coefplot

Syntax:

```
coefplot [, level(N) title("...") xlabel("...") ylabel("...") ylim(lo hi)
name("file.png") zero_line(off) ref_line(off) connect drop_cons(off)]
```

Forest plot of the most recent estimation's coefficients with confidence intervals. The plot type is auto-detected from what the last estimation stored in e():

• Event study / dynamic DiD (didregress method(eventstudy), or cs/gardner/stacked with aggregate(event_study)) — forest plot of per-period effects against period number, with horizontal zero line and a vertical reference-period line. Pre-period points render in grey, post-period in blue. A connecting line is drawn between adjacent points for readability.
• Scalar-ATT DiD (twfe, did, cs default, sa, bjs, sdid) — single point with horizontal CI whisker labeled with the method name. When the estimator returns SE=0 (e.g. SDID without enough control units for placebo variance), the CI is suppressed and a red "SE not estimated" annotation is added rather than drawing zero-width whiskers that would silently mislead.
• Linear / GLM (regress, logit, probit, poisson, nbreg, tobit, ivregress) — horizontal forest plot of every coefficient with CI whiskers. The constant is dropped by default; use drop_cons(off) to include it.

Options:

`level(N)` — Confidence level (default 95).

`title("...")` — Plot title.

`xlabel/ylabel` — Axis labels.

`ylim(lo hi)` — Force y-axis range.

`zero_line(off)` — Suppress the zero reference line.

`ref_line(off)` — Suppress the vertical reference-period line (event study only).

`connect` — Draw a connecting line between adjacent points (default ON for event studies).

`drop_cons(off)` — Include the constant in generic plots.

`name("f.png")` — Save to file (PNG, PDF, or SVG by extension) instead of opening.

Example:

```
// Event study with publication-quality forest plot:
didregress (gdp_growth) (treated), method(eventstudy) ///
    post_periods(2018 2019 2020 2021 2022) reference_period(2017)
coefplot, title("GDP-growth event study") name("gdp_es.png")

// Compare DiD methods side by side:
didregress (y) (treated), method(cs)
coefplot, name("cs.png")
didregress (y) (treated), method(bjs)
coefplot, name("bjs.png")

// After regress / logit / etc.:
reg log_wage educ exper i.region, robust
coefplot, level(99)
```

graph

Syntax:

```
graph bar (stat) var [, over(var) horizontal title("t")]
graph box var [, over(var) title("t")]
graph line yvar xvar [, by(var) sort title("t")]
graph export "file" [, replace width(N) height(N) dpi(N)]
graph close
```

Umbrella for bar, box, and line charts, plus export/close utilities.

Example:

```
graph bar (mean) income, over(city)
graph box income, over(education_level)
graph line gdp year, sort by(country)
graph export "fig1.pdf", replace width(1200)
```

19. Scripting and control flow

local

Syntax:

```
local macname = expression
local macname "text"
local macname val1 val2 val3
```

Define a local macro. Reference later with `macname' (backtick + apostrophe). Scoped to the current script file or block.

Example:

```
local yvars income wage earnings
local y_unit "USD"
foreach y of local yvars {
    summarize `y'
}
```

global

Syntax:

```
global macname = expression
global macname "text"
```

Define a global macro. Reference with $macname or ${macname}. Persists across script files (unlike local).

Example:

```
global datadir "/projects/paper1/data"
use "$datadir/survey.dta", clear
```

foreach

Syntax:

```
foreach macname in list { commands }
foreach macname of varlist varlist { commands }
foreach macname of numlist numlist { commands }
foreach macname of local mname { commands }
```

Loop over a list. `of varlist` iterates over column names; `of numlist` iterates over number ranges (e.g. `1/10`, `2 4 6`).

Example:

```
foreach y in income wage earnings {
    summarize `y'
}
foreach x of varlist x* {
    replace `x' = 0 if missing(`x')
}
```

forvalues

Syntax:

```
forvalues i = start/end { commands }
forvalues i = start(step)end { commands }
```

Loop over a numeric range.

Example:

```
forvalues y = 2010/2020 {
    summarize income if year == `y'
}
forvalues t = 0(0.1)1 {
    display `t'
}
```

by / bysort

Syntax:

```
by varlist: command
bysort varlist: command
```

Run a command once per group defined by varlist. bysort sorts first.

Example:

```
bysort country year: egen avg_gdp = mean(gdp)
by region: summarize income
```

quietly (alias: qui)

Syntax:

```
quietly command
```

Run a command with all output suppressed. Useful in loops that would otherwise flood the console.

Example:

```
qui foreach y of varlist y* {
    regress `y' x1 x2
    display "`y' R^2 = " e(r2)
}
```

capture

Syntax:

```
capture command
capture noisily command
```

Run a command but swallow any error. Sets _rc to 0 on success or the error code on failure. With `noisily`, prints output normally but still captures the error. Combines with `if _rc == 0 { ... }` to branch. The `_rc` macro is also available inside expression contexts (generate, replace, assert) — e.g. `assert _rc != 0` after `capture` will succeed.

Example:

```
capture confirm variable income
if _rc != 0 {
    display "income column not present"
}

* _rc is also visible in expressions:
capture summarize bad_var
gen had_error = _rc != 0
```

return

Syntax:

`return list`

Show r() results stored by the most recent command. Typical r() keys: r(N), r(mean), r(sd), r(min), r(max).

Example:

```
summarize income
return list
display r(mean)
```

ereturn

Syntax:

`ereturn list`

Show e() results stored by the most recent estimation command. Typical e() keys: e(N), e(r2), e(rmse), e(b), e(V), e(method). After didregress with method(eventstudy), e() also includes `coefficients` (the per-period vector that coefplot reads) and `reference_period`.

Example:

```
regress income education
ereturn list
```

20. Programming and diagnostics

assert

Syntax:

`assert condition [if cond] [in range]`

Halt with an error (rc=9) if the condition is false for any row. Used for sanity checks in pipelines. Honors `if` and `in` clauses — the assertion is only evaluated on the matching subset. The `_rc` macro is visible inside the expression, so `assert _rc != 0` works after a `capture`.

Example:

```
assert age >= 0 & age <= 120
assert !missing(id)
```

```
assert income > 0 if age >= 18         // subset only
capture confirm variable bad
assert _rc != 0                        // verify capture worked
```

preserve / restore

Syntax:

```
preserve
... (any commands) ...
restore
```

Snapshot the current dataset and later revert to the snapshot. Useful when you want to experiment without losing your state.

Example:

```
preserve
    keep if year == 2020
    collapse (mean) income, by(region)
    save "snapshot.dta", replace
restore
```

egen

Syntax:

```
egen newvar = function(args) [if condition] [, by(groupvars)]
```

"Extended" generate — group-level and cross-row computations that plain gen can't do in one line.

Example:

```
egen avg_income = mean(income), by(country year)
egen rich = tag(country), by(country)
egen z_income = std(income)
egen missing_count = rowmiss(q1 q2 q3 q4 q5)
egen firm_id = group(company industry)
```

Notes: *Available functions: mean, median, sum, count, min, max, sd, total (alias sum), rowtotal, rowmean, rowmin, rowmax, rowmiss, group, rank, tag, seq, concat, std.*

notes

Syntax:

```
notes                    (show all notes)
notes : "text"          (add a note)
notes drop N             (drop note number N)
notes drop _all          (drop all notes)
```

Attach and manage dataset-level free-text notes. Notes travel with the dataset when saved.

Example:

```
notes : "Dropped obs with income < 0 (n=14) on 2024-08-01"
notes
```

display (alias: di)

Syntax:

```
display expression
display "string"
```

Like a calculator: evaluate and print an expression. Supports scalar r() results (e.g. `display r(mean)`).

Example:

```
display 2 + 3 * sqrt(16)
display "Hello, " + "world"
summarize income
display r(mean)
```

isid

Syntax:

```
isid varlist [, sort]
```

Verify that varlist uniquely identifies observations. Errors if duplicates exist.

Options:

- **`sort`** — Sort by varlist in the process.

Example:

```
isid country year
```

levelsof

Syntax:

```
levelsof varname [if] [, local(macname) clean separate(sep)]
```

List unique values of a variable. With local(), stores them in a local macro for use in subsequent loops.

Example:

```
levelsof country, local(ctys)
foreach c of local ctys {
    display "Country: `c'"
}
```

distinct

Syntax:

```
distinct [varlist] [if]
```

Count the number of distinct combinations of varlist (or the full dataset).

Example:

```
distinct country year
```

compress

Syntax:

```
compress [varlist]
```

Shrink storage by downcasting numeric types (int64 → int32/16/8 where values fit). Commonly reclaims 30-60% of memory on datasets loaded from CSV.

Example:

```
compress
```

21. NLP — neural backend (hf / nlp)

The `hf` command (kept as an alias; `nlp` also dispatches the neural subcommands) runs transformer models locally via HuggingFace's transformers library. Models are cached to rln/hf_models/ so the folder stays portable. First use of a model triggers a download; subsequent uses are fully offline.

hf classify (also: nlp classify)

Syntax:

```
hf classify var, labels("label1 label2 label3") [generate(newvar) model(name) multi
threshold(0.5) batch(N) if cond]
```

Zero-shot text classification via NLI. Returns the most probable label per row (or all labels above threshold with `multi`).

Options:

`labels(...)` — Candidate labels (space-separated, quoted).

`generate(v)` — Output variable (default: <var>_label).

`model(m)` — HuggingFace model ID. Default: facebook/bart-large-mnli.

`multi` — Return all labels above threshold, pipe-joined.

`threshold(x)` — Score threshold for `multi` (default 0.5).

`batch(N)` — Batch size (default 8).

Example:

```
hf classify review, labels("positive negative neutral") generate(sent)
```

hf sentiment (also: nlp sentiment)

Syntax:

```
hf sentiment var [, generate(newvar) model(name) batch(N)]
```

Sentiment analysis. Default model is multilingual 1-5 star rating.

Options:

`model(m)` — Default: nlptown/bert-base-multilingual-uncased-sentiment. Alternative: distilbert-base-uncased-finetuned-sst-2-english (English only).

Example:

```
hf sentiment review, generate(stars)
```

hf summarize (also: nlp summarize when neural backend chosen)

Syntax:

```
hf summarize var [, generate(newvar) maxlen(150) minlen(30) model(name) batch(N)]
```

Neural abstractive summarization (generates new sentences). Contrast with `nlp summarize` which is extractive and fully offline.

Options:

`maxlen(N)` — Max output tokens.

`minlen(N)` — Min output tokens.

`model(m)` — Default facebook/bart-large-cnn. Multilingual: csebuetnlp/mT5_multilingual_XLSum.

Example:

```
hf summarize article, generate(summary) maxlen(80)
```

hf translate (also: nlp translate when neural backend chosen)

Syntax:

```
hf translate var, from(src) to(tgt) [generate(newvar) model(name) batch(N)]
```

Neural machine translation. Default: Helsinki-NLP/opus-mt-{src}-{tgt} if available, else NLLB-200 for the long tail of language pairs. Contrast with the offline `nlp translate` backend.

Example:

```
hf translate review, from(id) to(en) generate(review_en)
```

hf ner (also: nlp ner)

Syntax:

```
hf ner var [, generate(newvar) model(name) batch(N)]
```

Named entity recognition. Returns a pipe-joined list of TYPE:text pairs.

Options:

`model(m)` — Default: dslim/bert-base-NER. Multilingual: Davlan/xlm-roberta-large-ner-hrl.

Example:

```
hf ner document, generate(entities)
```

hf embed (also: nlp embed)

Syntax:

```
hf embed var [, generate(stub) model(name) dims(N) batch(N)]
```

Compute sentence embeddings and expand into `stub_1, stub_2, ..., stub_N` columns.

Options:

`dims(N)` — Truncate/project to N dimensions.

`model(m)` — Default: sentence-transformers/all-MiniLM-L6-v2 (384-d). Multilingual: sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2.

Example:

```
hf embed review, generate(emb) dims(64)
```

hf models

Syntax:

```
hf models [task]
```

List recommended models for each NLP task. Useful when you need ideas for non-default models.

Example:

```
hf models
hf models translate
```

hf download

Syntax:

```
hf download model_name
```

Pre-download a model to rln/hf_models/ for offline or portable use. Run once online; thereafter the folder is portable.

Example:

```
hf download facebook/bart-large-mnli
hf download Helsinki-NLP/opus-mt-id-en
```

hf cache

Syntax:
`hf cache`

Show HuggingFace model cache info (sizes, paths). Useful for checking what's been downloaded and how much disk it uses.

Example:

```
hf cache
```

22. NLP — offline backend (nlp)

The offline NLP dispatcher `nlp` handles translate and summarize without requiring neural models. Every other subcommand (classify, sentiment, ner, embed, models, download, cache) is delegated to the neural backend above.

nlp translate

Syntax:
`nlp translate var, from(src) to(tgt) [generate(newvar) pivot(en) if cond]`

Offline machine translation via argos-translate. Models (≈100 MB per language pair) live in rln/argos_models/ and travel with the folder. If the direct pair isn't installed, auto-downloads or pivots through English when both src→en and en→tgt exist.

Options:

- **`pivot(lang)`** — Pivot language when direct pair not installed (default: en).
- **`generate(v)`** — Output variable.

Example:

```
nlp download translate id en
nlp download translate en de
nlp translate text, from(id) to(de) generate(text_de)   ; pivots through English
```

Notes: *Set RLN_NLP_NO_AUTODOWNLOAD=1 to disable the automatic one-shot download.*

nlp summarize

Syntax:

```
nlp summarize var [, generate(newvar) sentences(3)
method(lsa|lexrank|textrank|luhn|kl|sumbasic|edmundson) language(english) if cond]
```

Offline extractive summarization via sumy. Picks the N most informative sentences from the input rather than generating new ones. No model downloads required; runs in pure Python. Legacy hf-style options `maxlen(N)` and `minlen(N)` are accepted as aliases — maxlen becomes approximately `sentences = max(1, N / 40)`.

Options:

`sentences(N)` — Number of sentences to extract (default 3).

`method(m)` — Algorithm: lsa (default), lexrank, textrank, luhn, kl, sumbasic, edmundson.

`language(l)` — Stemmer/stopword language (default english).

Example:

```
nlp summarize article, generate(sum1) sentences(2) method(textrank)
nlp summarize article_de, generate(sum2) language(german) method(lexrank)
```

nlp download

Syntax:

```
nlp download translate src tgt     — install an argos-translate pair
nlp download summary               — verify sumy is working
nlp download hf_model_name         — delegate to hf download
```

Download and install an argos-translate language pair (or delegate to the neural downloader). In v1.1.1, if the direct src→tgt pair is unavailable but pivot legs exist, both legs are installed automatically.

Example:

```
nlp download translate id en
nlp download translate en id
nlp download translate id de   ; auto-installs id->en and en->de
```

23. Larger-than-RAM mode (lrtm)

`lrtm` is a parallel command set backed by polars' lazy engine. It operates on on-disk parquet files without loading them into RAM. Use `lrtm collect` at the end to materialize the filtered/transformed result to the regular pandas working DataFrame for downstream commands.

lrtm use

Syntax:

```
lrtm use "filepath"
```

Lazy-load a parquet (or other supported) file. No data is read yet — polars builds a query plan. Subsequent lrtm commands add filters, projections, and aggregations to the plan.

Example:

```
lrtm use "datasets/firm_panel_10GB.parquet"
```

lrtm convert

Syntax:

`lrtm convert "input" [, output("file.parquet") chunk(500000)]`

Convert CSV, Excel, DBF, DTA, or RData files to parquet for fast subsequent lrtm access. CSV is read in chunks to avoid RAM spikes; other formats are read eagerly then streamed out.

Example:

```
lrtm convert "huge_export.csv"
lrtm convert "legacy.dbf", output("legacy.parquet")
```

lrtm describe

Syntax:

`lrtm describe`

Show schema, row count estimate, and file metadata without scanning data.

Example:

```
lrtm describe
```

lrtm summarize

Syntax:

`lrtm summarize [varlist]`

Streaming summary statistics (min, max, mean, count, null count) on the lazy frame.

Example:

```
lrtm summarize revenue employees year
```

lrtm count

Syntax:

`lrtm count [if condition]`

Count rows in the lazy frame, optionally under a filter. Streaming — no full scan of RAM is needed.

Example:

```
lrtm count
lrtm count if revenue > 1000000 & year >= 2020
```

lrtm tabulate

Syntax:

```
lrtm tabulate var1 [var2]
```

Fast frequency table (one-way or two-way) via polars group_by. Streaming; reads only the required columns.

Example:

```
lrtm tabulate sector
lrtm tabulate country sector
```

lrtm generate

Syntax:

```
lrtm generate newvar = expression
```

Lazy transform — adds a computed column to the query plan without executing.

Example:

```
lrtm generate log_rev = ln(revenue)
lrtm generate bigfirm = (employees > 500)
```

lrtm filter (alias: lrtm keep if)

Syntax:

```
lrtm filter condition
lrtm keep if condition
lrtm drop if condition
```

Apply a lazy row filter. Internally equivalent to keep/drop if.

Example:

```
lrtm keep if year >= 2020 & country == "DE"
lrtm drop if missing(revenue)
```

lrtm sort

Syntax:

```
lrtm sort varlist
```

Add an ORDER BY to the query plan.

Example:

```
lrtm sort year -revenue
```

lrtm merge

Syntax:

`lrtm merge using "file" [, on(varlist) how(inner|left|right|outer)]`

Streaming join against another on-disk file. Hash-join when small, sort-merge otherwise. Supports all standard join types.

Example:

```
lrtm merge using "refdata.parquet", on(country_code) how(left)
```

lrtm fuzzmerge

Syntax:

`lrtm fuzzmerge varname using "file" [, threshold(0.8) workers(4)]`

Streaming fuzzy join. Workers controls parallel matching threads.

Example:

```
lrtm fuzzmerge company_name using "sec_companies.parquet", threshold(0.85) workers(8)
```

lrtm collapse

Syntax:

`lrtm collapse (stat) varlist [, by(groupvars)]`

Streaming groupby aggregation. Same stats as plain `collapse`, but never materializes the full dataset.

Example:

```
lrtm collapse (mean) revenue (sum) employees, by(country sector)
```

lrtm save

Syntax:

`lrtm save "file.parquet"`

Execute the query plan and write the result to parquet. The intermediate dataset never enters RAM if polars can stream it.

Example:

```
lrtm save "filtered_firms.parquet"
```

lrtm head

Syntax:

`lrtm head [N]`

Preview the first N rows of the current plan (default 10). Forces partial execution.

Example:

```
lrtm head 20
```

lrtm collect

Syntax:

`lrtm collect`

Materialize the lazy result into the pandas working DataFrame. After this, all regular Rln commands work on it.

Example:

```
lrtm use "big.parquet"
lrtm keep if revenue > 1e7
lrtm collect
regress log_rev employees i.sector
```

lrtm status

Syntax:

`lrtm status`

Show the current query plan (filters, transformations) and estimated row count.

Example:

```
lrtm status
```

lrtm clear

Syntax:

`lrtm clear`

Drop the lazy frame and free its memory. The regular pandas DataFrame is untouched.

Example:

```
lrtm clear
```

24. System and utility

help

Syntax:
`help [topic]`

Show command-level help. Without a topic, lists every command grouped by category. Supports every command name and every expression function (e.g. `help inlist`, `help regexm`, `help didregress`).

Example:

```
help
help regress
help inlist
```

clear

Syntax:
`clear`

Drop all data from memory — both the pandas DataFrame and any lrtm lazy frame. Scalar macros (local/global) are preserved.

Example:

```
clear
```

pwd

Syntax:
`pwd`

Print the current working directory.

Example:

```
pwd
```

cd

Syntax:
`cd "directory"`

Change the working directory. Affects subsequent relative paths in use/import/save/export/log.

Example:

```
cd "/projects/paper1"
cd ..
```

dir

Syntax:

`dir [pattern]`

List files in the current (or pattern-matched) directory.

Example:

```
dir
dir *.dta
dir data/*.csv
```

set

Syntax:

`set [setting [value]]`

Show or change settings. Settings include: more (pagination), maxvar (max variables), type (default numeric type), seed (random seed). Use set workdir "<path>" to pin the default folder that file pickers and bare use/do names resolve against (it otherwise follows your latest project, starting from the examples folder). The choice persists between sessions.

Example:

```
set
set seed 42
set more off

set workdir "C:/my/project"
```

memory

Syntax:

`memory`

Report current memory usage: DataFrame size, per-column breakdown, Python process RSS.

Example:

```
memory
```

do

Syntax:

`do "filename.do"`

Execute a script file. Supports `*` and `//` comments, `/* ... */` block comments, `///` line continuation, nested script files, and all scripting constructs (local, global, foreach, forvalues, quietly, capture).

Example:

```
do "analysis/step1_clean.do"
do "build_panel.do"
```

doedit

Syntax:

`doedit ["filename.do"]`

Open a terminal-based script file editor with syntax highlighting (textual). Without a filename, opens an untitled buffer. Ctrl+S saves; Ctrl+R runs.

Example:

```
doedit
doedit "analysis/main.do"
```

python

Syntax:

```
python: expr
python { ... multi-line code ... }
```

Drop into Python. Single-line form evaluates an expression; block form runs arbitrary Python code. The current DataFrame is available as `df`, pandas as `pd`, and numpy as `np`. Any assignment to `df` is reflected back into Rln — including initial assignment, so you can bootstrap a dataset from inside a python block when none was loaded previously.

Inside script files, python blocks correctly preserve indentation. Multi-line `if:`, `for:`, `try:` etc. all work as expected — uniform leading whitespace is stripped before exec, so you can indent the block body for readability inside the script file without breaking Python's parser.

Example:

```
python: df.shape

* Multi-line block — indentation preserved:
python {
    import scipy.stats as st
    df["z"] = st.zscore(df["income"])
    if df["z"].abs().max() > 5:
        print("Outlier present")
    for col in df.select_dtypes(include="number").columns:
        print(f"{col}: {df[col].mean():.2f}")
}

* Bootstrap a dataset from scratch:
python {
    df = pd.read_parquet("/data/big.parquet")
    df = df.query("year >= 2015")
}
```

```
describe   // df is now visible to subsequent commands
```

25. Packages

ssc

Syntax:

```
ssc install pkg1 [pkg2 ...]
ssc remove pkg
ssc list
ssc search keyword
ssc update pkg
```

compact wrapper around pip. Useful for installing optional backends (diff-diff, argostranslate, sumy, polyfuzz, linearmodels) that Rln commands check for at runtime. On a source install this wraps pip normally. Packaged builds (the lite desktop executables and the Android app) are self-contained and cannot pip-install new packages; there ssc install reports this clearly and points you to the source install or a larger build tier rather than failing obscurely.

Example:

```
ssc install diff-diff
ssc install argostranslate sumy
ssc list
ssc update polars
```

Part V — Expression language reference

Rln's expression language is evaluated by every command that accepts an `= expression` or `if condition`: gen, replace, drop/keep if, count if, summarize if, and the if-clause of every other command. Expressions can reference variables by name, use parentheses freely, and call any of the functions below.

26. Built-in string functions

Function	Meaning	Example
substr(s, start, length)	Substring, 1-indexed (standard convention).	substr(name, 1, 3)
strlen(s)	Character count.	strlen(notes)
length(s)	Alias for strlen.	length(text)
upper(s)	Uppercase.	upper(name)
lower(s)	Lowercase.	lower(email)
trim(s)	Strip leading/trailing whitespace.	trim(response)
ltrim(s) / rtrim(s)	Strip leading / trailing whitespace.	ltrim(name)
strmatch(s, "pattern")	Glob-style match (* and ?) — returns 1/0.	strmatch(code, "A*")
strpos(s, "sub")	1-indexed position of sub within s (0 if not found).	strpos(email, "@")
subinstr(s, "old", "new", .)	Replace all occurrences of old with new.	subinstr(name, " ", " ", .)
word(s, n)	n-th whitespace-separated token.	word(fullname, 2)
real(s)	Convert string → numeric (NaN if unparseable).	real(price_str)
string(x)	Convert numeric → string.	string(year)

27. Built-in numeric functions

Function	Meaning	Example
abs(x)	Absolute value.	abs(x - mean_x)
ceil(x)	Smallest integer ≥ x. ceil(3.2)=4, ceil(-3.7)=-3.	ceil(score)
floor(x)	Largest integer ≤ x. floor(3.7)=3, floor(-3.7)=-4.	floor(age / 10)
round(x)	Round to nearest integer (banker's rounding at exact halves).	round(wage)
round(x, N)	Round to N decimal places.	round(rate, 4)
trunc(x)	Truncate toward zero. trunc(3.7)=3, trunc(-3.7)=-3.	trunc(income / 1000)
int(x)	Alias for trunc(x).	int(miles)
ln(x)	Natural log.	ln(income + 1)
log(x)	Alias for ln.	log(x)
log10(x)	Base-10 log.	log10(population)
exp(x)	e^x.	exp(beta_hat)
sqrt(x)	Square root.	sqrt(variance)
min(a, b, ...)	Minimum (elementwise) of two or more args.	min(age, 65)
max(a, b, ...)	Maximum (elementwise) of two or more args.	max(income, 0)
mod(a, b)	Modulo (remainder).	mod(year, 4)
cond(c, t, f)	If-else (vectorized).	cond(income > 50000, 1, 0)
missing(x)	TRUE if x is NA / NaN / blank.	missing(income)
x^n (or xn)**	Exponentiation.	age^2

Rounding inside weight clauses: round() (and any other expression function) can be used inside a weight clause when raw weights are fractional but fweight needs integers:

```
summarize wage [fweight=round(pop_frac)]    // round() before fweight
regress y x [aweight=w1 + w2]               // arithmetic
tabulate region [pweight=sqrt(survey_wt)]   // any expression
```

28. Extension functions

These extension functions match standard semantics found in compact-syntax econometric tools. They accept expressions (not just bare variables) as their first argument.

inlist()

Syntax: `inlist(var, v1, v2, ...)`

TRUE where var equals any of the listed values. Works on strings and numbers.

```
gen bigcity = inlist(city, "NYC", "LA", "Chicago")
keep if inlist(age, 18, 21, 30, 45, 65)
```

inrange()

Syntax: `inrange(expr, lo, hi)`

TRUE where lo ≤ expr ≤ hi. First argument can be any expression.

```
gen workingage = inrange(age, 18, 65)
gen mid_income = inrange(income/1000, 30, 80)
```

regexm()

Syntax: `regexm(s, "pattern")`

TRUE where the PCRE-style regex matches anywhere in s. NA-safe.

```
gen has_digit = regexm(name, "[0-9]")
keep if regexm(email, "@gmail\\.com$")
```

regexr()

Syntax: `regexr(s, "pattern", "replacement")`

Replace every regex match. Supports backreferences \1, \2, etc.

```
gen clean = regexr(phone, "[^0-9]", "")
gen masked = regexr(notes, "\\d{3}-\\d{4}", "XXX-XXXX")
```

regexs()

Syntax: `regexs(s, "pattern_with_groups", n)`

Extract capture group n (1-based). Non-matching rows return NA.

```
gen area = regexs(phone, "\\(([0-9]+)\\)", 1)
gen domain = regexs(email, "@(.+)$", 1)
```

29. Logical, comparison, and special symbols

Symbol	Meaning
== / != / ~=	Equality / inequality (the `~=` accepted).
> >= < <=	Numeric and string comparison.
& | !	Logical AND, OR, NOT.
+ - * /	Arithmetic.
^ or **	Exponentiation (Rln translates ^ to ** outside string literals).
_n	Current row number (1-indexed, standard convention).
_N	Total rows in dataset.
"text" or `"text"'	String literal. Compound-quote syntax accepted.
. or missing()	Missing value. `x == .` and `missing(x)` are equivalent.

Part VI — Appendices

A. Sample script files shipped with Rln

File (under examples/)	Demonstrates
sample.do	Quickstart — basic load, describe, regress.
extension_functions_showcase.do	Every extension function (inlist/inrange/regexm/regexr/regexs/real/string/length) inside gen/replace/if.
did_panel_showcase.do	All eleven didregress methods on a staggered panel.
nlp_showcase.do	Neural (hf) pipeline: classify, sentiment, translate, summarize, NER, embed.
nlp_extend_showcase.do	Offline (nlp) pipeline with side-by-side comparison to the neural backend.
lrtm_showcase.do	Larger-than-RAM mode via polars.

B. Environment variables

Variable	Effect
RLN_NLP_NO_AUTODOWNLOAD=1	Disable the one-shot auto-download in `nlp translate` when a language pair isn't installed.
ARGOS_PACKAGES_DIR=<path>	Override the argos-translate model directory (default: rln/argos_models).
HF_HOME=<path>	Override the HuggingFace model cache directory (default: rln/hf_models).
TRANSFORMERS_OFFLINE=1	Prevent HuggingFace from attempting online model lookups at all.

C. Troubleshooting

Q: "No installed argos-translate path for xx -> yy."

A: Rln v1.1.1+ will auto-download the direct pair or both pivot legs on first use. If auto-download fails (no internet, firewall, RLN_NLP_NO_AUTODOWNLOAD=1), run `nlp download translate <src> <tgt>`. If no direct pair exists, install both legs: `nlp download translate <src> en` and `nlp download translate en <tgt>`.

Q: "sumy backend failed" / NLTK data missing.

A: Rln ships a regex-based fallback tokenizer that works without any NLTK corpora. If you want better tokenization quality, run once online: `python -c "import nltk; nltk.download('punkt_tab'); nltk.download('stopwords')"`.

Q: "diff-diff method 'xxx' failed: ... falling back to TWFE."

A: The backend couldn't fit the requested model. Common causes: the panel doesn't have enough periods, first_treat() is missing for a staggered estimator, or the treatment indicator isn't absorbing for sdid/eventstudy. Rln auto-derives most of these but prints a warning when it does.

Q: method(honest) with no prior eventstudy.

A: Rambachan-Roth sensitivity requires event-study results. Run method(eventstudy) first; Rln caches the result on state and reuses it for honest.

Q: PyInstaller build misses a module.

A: Check rln.spec — v1.1.1 includes `commands.nlp_extend` and hiddenimports for argos, sumy, and diff_diff. If you added optional dependencies, add them to the hiddenimports list.

Q: "Cannot evaluate expression ...: name 'X' is not defined."

A: Variable X doesn't exist in the current DataFrame. Run `describe` to see available columns. Common cause: loading the wrong CSV, or using a column name with a typo.

Q: Regex pattern with ^ anchor works differently than expected.

A: Fixed in v1.1.1: the `^` → `**` exponentiation rewrite now skips string literals, so `"^abc"` inside a regex pattern is preserved as a start-of-string anchor.

D. License and credits

Rln stands on the shoulders of every package listed in Part III. In particular, the following projects are core dependencies and are the reason Rln can exist as a small wrapper rather than a large rewrite of every algorithm:

- **pandas** — BSD-3-Clause (https://pandas.pydata.org)
- **statsmodels** — BSD-3-Clause (https://www.statsmodels.org)
- **scipy** — BSD-3-Clause (https://scipy.org)
- **matplotlib** — PSF-based (https://matplotlib.org)
- **rich** — MIT (https://github.com/Textualize/rich)
- **textual** — MIT (https://github.com/Textualize/textual)
- **prompt_toolkit** — BSD-3-Clause (https://github.com/prompt-toolkit/python-prompt-toolkit)
- **diff-diff** — MIT (https://pypi.org/project/diff-diff)
- **argostranslate** — MIT / CC0 (https://github.com/argosopentech/argos-translate)
- **sumy** — Apache-2.0 (https://github.com/miso-belica/sumy)
- **transformers** — Apache-2.0 (https://github.com/huggingface/transformers)
- **sentence-transformers** — Apache-2.0 (https://www.sbert.net)
- **polars** — MIT (https://www.pola.rs)

linearmodels — NCSA (https://github.com/bashtage/linearmodels)

polyfuzz — MIT (https://github.com/MaartenGr/PolyFuzz)

pyreadr — GPL-3.0 (https://github.com/ofajardo/pyreadr)

dbfread — MIT (https://github.com/olemb/dbfread)