

parqit

A grammar of data manipulation for Stata,
backed by Parquet and an embedded DuckDB engine

Out-of-core computation in the verbs you already know — manipulate before you load

Miguel Portela

NIPE, School of Economics, Management and Political Science
Universidade do Minho

Also affiliated with BPLIM (Banco de Portugal) and the IZA@LISER Network

2026 UK Stata Conference
London, 3–4 September 2026

Companion to *Augmenting Stata with artificial intelligence*, Thursday 3 September, 13:00.

PART 1

Manipulate before you load

What parqit is

One sentence

parqit turns Stata's implicit current dataset into a **lazy view over columnar files**. You write ordinary Stata verbs; they build a plan; an embedded DuckDB executes it out-of-core over Parquet.

- Public since June 2026, MIT; on SSC since 28 August (`ssc install parqit, all`), with GitHub a few releases ahead (v0.1.33, 3 September). Stata 16 or newer, MP recommended; four platform builds — Linux, Windows, Intel and Apple-Silicon macOS; the plugin embeds DuckDB 1.5.3, so there is nothing else to install.
- Typed commands, or the **User** > **parqit** menu and its ten dialogs, which write the same commands.
- Co-authored with Rute Costa, Paulo Guimarães and Marta Silva (BPLIM, Banco de Portugal); designed by the speaker, implemented by two AI coding agents under his direction and review.

Manipulate before you load.

The dataset is a plan, not a block of memory

Stata today

- One dataset, loaded whole, in memory.
- RAM is the ceiling: the file must fit before you can even look at it.
- Verbs run as you type them, over memory.
- More tables mean frames — each loaded whole.
- Large data: subset it elsewhere first, or wait.

With parqit

- A *view*: a plan over Parquet files on disk.
- Disk is the ceiling: DuckDB streams and spills; RAM is for the *result*.
- Verbs build the plan; one query runs it — columnar, parallel, out of core.
- More tables mean named views, joined to each other on disk.
- Large data: open in seconds; explore, filter, aggregate; collect only the result.

Labels, value labels, notes, formats and characteristics ride inside the Parquet file. Frames keep their job for what is already in memory; for everything else, a view does it without the memory.

Think in views and plans, not in what fits: memory becomes the last step, not the first.

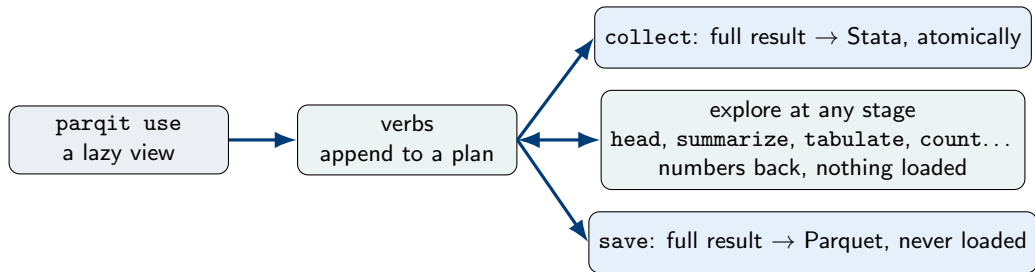
Not another Parquet reader

Reading and writing Parquet in Stata is already served — by `pq`, by the legacy `stata-parquet`, and by StataNow's own `import parquet`. `parqit` is the whole loop — open, explore, transform, combine, materialise — in Stata's own verbs; reading and writing are its two ends, not its point:

- a full grammar of single-table and two-table verbs — `keep`, `gen`, `collapse`, `merge`, `reshape`, `pivot` — that compiles to *one* DuckDB query;
- exploration that runs on disk: `describe`, `summarize`, `tabulate`, `codebook`, `misstable`, `count if`;
- SQL if you want it (`parqit show` prints the query it generated), and nobody has to learn it.

The product is the manipulation layer, not the transfer.

One lazy view: look whenever you like, two destinations for the full result



- Mutation verbs *do not* materialise when you type them: they append a step to a plan.
- You never need `collect` just to look: `head`, `summarize`, `tabulate`, `count...` run over the plan as it stands — after a `keep`, a `gen`, a `collapse` — and return numbers or a few rows.
- Two destinations for the *full* result: `collect` replaces the current dataset atomically, once the new data is complete and valid; `save` writes Parquet directly, never loading it.

One plan; look at it whenever you like; two destinations for the whole result.

The workflow, in the verbs you already use

```
parqit use using /data/qp_*.parquet    // lazy view: schema probed, no rows loaded

parqit head                            // look around first: the current
parqit summarize wage                  // dataset stays untouched

parqit keep if year >= 2010             // each verb appends to the plan
parqit gen double lwage = log(wage)
parqit collapse (mean) lwage (count) n = wage, by(firm year)
parqit summarize lwage                 // collapsed result; nothing loaded yet

parqit show                            // the SQL it generated, if you are curious
parqit save firm_year_panel.parquet    // ...or collect, to bring it into Stata
```

Nothing here is a new language. The one new idea is that the verbs are lazy.

The vocabulary is already Stata; the novelty is when the work happens.

Explore first, load last

The whole exploration family runs as push-down queries over the view *as it stands* — before or after any keep, gen, merge or collapse: the engine computes, only bounded summary output reaches Stata, and the current dataset never changes.

- describe reads only the Parquet footer — no data scan at all.
- codebook, misstable patterns, tabulate, summarize, detail, histogram scan engine-side and return numbers, not rows.
- So you can profile a file that would never fit in RAM, and check a collapse or a merge, *before* deciding what — if anything — to load.

The interesting work happens before the current dataset is replaced.

2.67 billion rows: collect was the wrong question

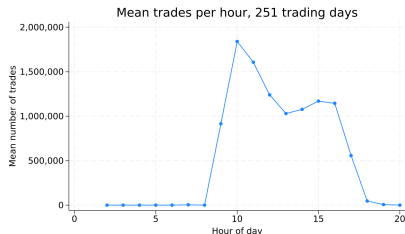
- A real glob of 2,672,287,500 rows was opened as a view.
- Stata's plugin interface cannot receive a result beyond $2^{31} - 1$ observations, so collect now **refuses with a real message** instead of dying in the parser.
- The remedy was not a bigger buffer. It was collapse on the lazy plan, which reproduced the exact total — and the refusal message now names that route.

Out-of-core matters when the result itself, not only the source, stays out of Stata.

The same source, live: eight blocks, seconds each

251 daily Parquet files, 30 GB, 2,672,287,500 rows $\times$ 15 columns — one year of exchange trades. The crash-course do-file, one idea per block, measured on a 48-core server:

use, describe, count: 2.67 bn rows, from the footers	0.1 s
count if on one date: pushdown, 9.0 M rows	< 1 s
summarize two columns over every row	5–6 s
one day: keep if, gen, collapse, show, head	1 s
save that result to Parquet; collect it into Stata	1 s
year leaderboard: collapse over every row, 122,195 tickers	6 s
two-stage collapse to a 19-row hour profile; collect	5 s
two named views merged on disk: the day's largest trades	1 s
labels written into the footer, back after use	—



the 19 collected rows

Stata's memory stayed empty until the last 19 rows — and every step took seconds, not hours.

A field test at BPLIM, Banco de Portugal — 2 September

Rute Costa (BPLIM) timed `parqit` use against `pq` use on three databases she works with every month — the same eager read into memory, results checked with descriptive statistics:

Database	Parquet	Rows	Vars	pq	parqit
Prices (SDP)	126 MB	58.4 M	15	≈ 30 s	19 s
Wages, Social Security (GR)	5.61 GB	1.64 bn	11	≈ 1 h	3 min
Panel, Social Security (FLOWS)	22 GB	664 M	50	> 1 h 30, or a crash	6 min

- Labels came back by themselves; with `pq` she re-applied the metadata with a separate tool.
- Her two requests — update one partition of a Hive tree in place, and a simpler help — shipped as `v0.1.31` the same day: `partitions(replace|append)`, `help parqit` + `help parqit_technical`.
- In-house benchmark (`parqit` 0.1.29 vs `pq` 3.0.7, synthetic $10\text{ M} \times 13$): use 1.3 s vs 6.6 s; merge + collect 2.3 s vs 9.8 s. One machine, one date.

Small file: a bit faster. Real data: twenty times faster — and a 22 GB panel that used to bring Stata down now opens in six minutes.

Transform, aggregate, reshape: one line each

```
parqit keep if inrange(age, 25, 45) & !missing(ln_wage, hours)    // rows
parqit keep idcode year age hours ln_wage union race industry    // columns
parqit gen double wage = exp(ln_wage)
parqit egen double person_mean = mean(ln_wage), by(idcode)
parqit replace hourly_wage = . if hours <= 0
parqit sort idcode year
parqit sample 10, seed(20260825)                                // engine-side, reproducible
parqit collapse (mean) m=ln_wage (sd) s=ln_wage (p50) med=ln_wage ///
              (count) n=ln_wage, by(year union)                // percentiles out of core
parqit contract year race, freq(workers)
parqit pivot (mean) m=ln_wage (count) n=ln_wage, rows(year) cols(union)
parqit reshape wide ln_wage, i(idcode) j(year)
parqit reshape long ln_wage, i(idcode) j(year)
```

Every line is lifted from the crash courses that ship with the package (`parqit_basics.do`, `parqit_tour.do`); `help parqit` is the manual, `help parqit_technical` the contract.

Each verb appends one stage; the whole chain is one query, however large the file.

Combine: files, views, and the dataset in memory

```
parqit use using workers_*.parquet          // many files, or a Hive tree: one table
parqit merge m:1 industry using industries.parquet, ///
    keep(match) keepusing(sector productivity) nogenerate
parqit merge m:1 industry using view:industry_stats, /// a view joined to a view
    keep(match) keepusing(industry_mean industry_n) nogenerate
parqit append using workers_late.parquet     // sources may be files or views
parqit joinby industry using events.parquet // all pairs within industry

* data already in memory, lookup on disk: Stata's own merge/append, no round trip
parqit mergein m:1 industry using industries.parquet, keep(match) nogenerate
parqit appendin using workers_late.parquet
```

A lazy merge fills the common variables from the using side on using-only rows, exactly as native merge does — one of the four wrong answers in Part 2 was precisely that.

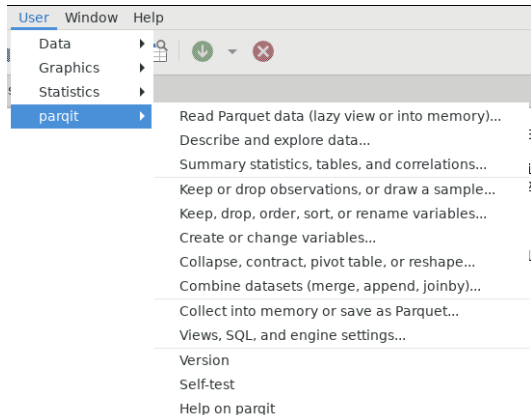
The using side never has to be loaded: a file, a glob, another view, or the data already in memory.

Explore, one line each: numbers back, nothing loaded

```
parqit describe          parqit ds          parqit lookfor wage experience
parqit head 6            parqit count if missing(ln_wage)
parqit codebook ln_wage hours union race
parqit misstable patterns ln_wage hours union
parqit levelsof industry          parqit distinct idcode year, joint
parqit duplicates report idcode year
parqit summarize ln_wage, detail          // Stata-exact moments and percentiles
parqit tabulate union collgrad, row col
parqit tabstat ln_wage, statistics(n mean sd p50) by(race)
parqit correlate ln_wage age tenure          parqit pwcorr ln_wage age, obs sig
parqit histogram ln_wage, bins(20)          // bins on disk, drawn with twoway
parqit set statamissing on          // missing largest of all, as in Stata
parqit query "qualify row_number() over (partition by idcode order by year) = 1"
parqit sql "select industry, avg(ln_wage) from read_parquet('w.parquet') group by 1"
```

Every one of these ran in the engine; only the numbers came back — and two escape hatches wait for the day the verbs run out.

Point and click: the **User** > **parqit** menu



- **Ten dialogs, one per task**, rewritten in v0.1.29 against StataCorp's guidelines ([P] *Dialog programming*, Appendix C).
- **Every dialog builds an ordinary parqit command** and echoes it to the Results and Review windows: a click is reproducible as a do-file line.
- **Populate** fills the pickers from the view, from memory, or from the Parquet footer of the named file; **Create...** opens the expression builder; an existing output file asks first.
- **Stata's own menus are never altered**: one submenu, kept by one line in `profile.do`.

GUI Stata on Linux, v0.1.29, after `parqit` menu;
unchanged through v0.1.33; Results pane blanked.

The same verbs, clicked: every dialog leaves a typed command behind.

The read dialog, as Stata draws it

The screenshot shows the 'Operation' dialog box in Stata. It has three radio buttons under the 'Operation' section: 'Open a lazy view over a source, or read the source into memory (parqit use)' (selected), 'Open the data in memory as a lazy view (parqit open _data)', and 'Resolve a path and report whether it exists (parqit path)'. Below this is a text field for the filename, containing 'qp_2008_2013.parquet'. To the right of the text field are three buttons: 'Describe', 'Browse...', and 'Populate'. Below the filename field is a section for 'Variables to read (leave empty for all variables):' with a list box containing 'id', 'firm', 'year', and 'wage'. At the bottom of the dialog, there is a checkbox for 'Union files with different schemas by column name (relaxed)' and a note: 'A lazy view probes the schema and loads no result rows; materialise it later with parqit collect or parqit save.' The bottom of the dialog has three buttons: '?', a refresh icon, and a save icon, followed by 'Submit', 'Cancel', and 'OK' buttons.

... and what **OK** left in the Results and Review windows:

```
. parqit use using qp_2008_2013.parquet  
(lazy view default opened over qp_2008_2013.parquet: 4 columns; schema probed, n  
> 0 rows loaded – use parqit collect or parqit save)
```

Populate read the variable list from the Parquet footer; OK typed the command for you.

PART 2

Where plausible answers go wrong

The translation boundary *is* the product

Between Stata and Parquet/SQL sit five things that do not line up by themselves:

- **Missing.** Stata's . is larger than everything; SQL's NULL is not a value at all.
- **Types.** Stata storage types versus Parquet physical types — and the integers that survive neither trip intact.
- **Names.** Case, duplicates, illegal characters, and names that only differ after sanitising.
- **Order.** A plan does not preserve physical row order unless it is told to.
- **Metadata.** Labels, value labels, notes, formats and characteristics are *data*, and they can be lost silently.

Familiar syntax does not guarantee familiar semantics.

Four wrong answers a user would have believed

When	What happened	Closed by
12 June	A named view returned <i>another view's</i> data: open first, open second, return to first, collect — and get the second one's rows, with success.	one bridge per promotion, plus a regression test
14 July	Two exact integers, 2^{53} and $2^{53}+1$, were converted to double too early. <code>contract</code> returned <i>one</i> key with <code>freq=2</code> . No warning.	wide integers stay exact in the lazy plan
22 August	<code>save</code> after <code>sort</code> could write the <i>original file</i> instead of memory: the fast path trusted <code>c(changed)</code> , which Stata does not set after <code>sort</code> , <code>gsort</code> or Mata edits.	general path by default; the copy became opt-in
22 August	<code>merge</code> left the common variables <i>empty</i> on using-only rows: native gave <code>c=444, sc="ud"</code> ; <code>parqit</code> gave <code>c=.</code> , <code>sc=""</code> .	projection fixed, variants covered

Every one of these returned plausible data with `rc 0`.

A severity scale built for data, not for crashes

S0	silent data-integrity hazard: <code>rc 0</code> , no note or a misleading one
S1	loud-but-wrong: visibly bad output, still <code>rc 0</code>
S2	structural or contract gaps: atomicity, lifecycle, portability, documentation (the full reports add S3 for tests and packaging)

For a package that moves data, a crash is the *good* outcome. The dangerous class is the one that returns a number and lets you build on it.

Rank findings by what a user would fail to notice.

When a green test protected the wrong answer

- `strpos(s, "")` returned **0**. Native Stata returns **1**, and so does DuckDB.
- The translator contained an explicit rule forcing the wrong value — *and a unit test asserting it*.
- The test was green, the comment was confident, and the premise was wrong. Native Stata adjudicated.

A test written from the same misunderstanding as the code does not detect the misunderstanding. It preserves it.

A test is only as good as its oracle.

Four oracles, four different questions

Native Stata	Is the <i>semantics</i> right? What does <code>merge</code> , <code>collapse</code> or <code>strpos</code> actually do here?
PyArrow	Are the <i>bytes and the schema</i> on disk what we claim?
DuckDB CLI	Is the <i>plan</i> the one we meant to generate?
Filesystem and binary tools	Who <i>owns</i> the file, did the write land atomically, and is the shipped artefact the one we tested?

These are not redundant checks of the same thing. Each can pass while another fails.

No single oracle can certify this boundary.

Six auditors, one certifier — 22 August

- Six parallel auditors, each owning a different way of being wrong: types and values, names and metadata, verb semantics, atomicity and I/O, documentation contract, performance.
- Roughly 1,300 targeted experiments against the three data oracles: native Stata, PyArrow, the DuckDB CLI.
- 38 triaged items implemented in severity order by a *separate* agent, then eight residual findings from an independent verifier.
- Final gate: 98 C++ cases / 3,653 assertions, and 97 Stata verdicts with zero failures — with performance against the previous release inside the noise.

Parallel agents pay off when each owns a different falsifier.

From a finding to a structural gate

A fixed bug is not closed until its *class* is harder to reintroduce:

- ① a minimal reproduction, kept;
- ② a standalone verify test with an external oracle;
- ③ a runner that refuses empty filters, a FAIL masked by a later PASS, or an aborted run;
- ④ two real Stata sessions exercising bridges and file ownership concurrently;
- ⑤ a release lint that pins version, help, dialogs, package selectors and the public surface;
- ⑥ and, where possible, a design guard that makes the mistake unrepresentable.

Mutate an assertion and the runner must fail — otherwise the harness proves nothing.

The audit trail, June to September

June	first adversarial passes: a named view returning the wrong data; reshape fabricating a plausible result; invalid UTF-8 producing an unreadable file with <code>rc 0</code>
July	a reliability campaign on v0.1.21 found 20 findings — five of them S0 — while the whole test suite was green , and returned NO-GO. v0.1.22 received a <i>scoped</i> GO
August	<code>strpos</code> and its wrong test; the six-auditor campaign and v0.1.28; then v0.1.29 rebuilt the point-and-click layer on StataCorp's own interface guidelines, and SSC published it on the 28th
September	a holistic audit of v0.1.29 on the 1st: a <i>conditional</i> GO with two S1 semantic findings — a string partition key reading the literal "NULL" back as missing; <code>float</code> columns compared with decimal literals in single precision. v0.1.30, released that night, fixes all sixteen, each pinned by a test; v0.1.31 ships BPLIM's two requests the next day; v0.1.33 adds the co-authors

Thirty-four versions in twelve weeks record learning, not automatic maturity.

PART 3

What can honestly be claimed

Out-of-core, precisely stated

- Parquet and delimited text are scanned out of core and can exceed memory; DuckDB spills to a temporary directory.
- **Opening is not free:** it probes schema and metadata, and a delimited source may be sampled.
- .dta and Excel are *not* engine-scannable — they are imported through a package-owned Parquet bridge.
- collect is bounded by the plugin interface's 32-bit observation index, $2^{31} - 1$ — Stata/MP itself goes further. save is the large-result path.

Out-of-core is a data path, not a claim of infinite scale.

Metadata, precisely stated

- Variable labels, value labels, notes, display formats and characteristics survive a save / use cycle, stored in standard Parquet key-value metadata.
- The file stays plain Parquet, readable by pandas, polars, R and Spark.
- **One documented loss:** extended missings .a-.z collapse to a single null — the format has one missing concept. Their *labels* survive; their identities do not.
- Third-party readers see codes and raw values; they do not rehydrate Stata labels automatically.
- In the field: at BPLIM, a read through pq was followed by a separate metadata step (`metaxl` apply); with `parqit`, in the user's words, nothing is needed.

Metadata fidelity is part of data fidelity — and its limits belong on the slide.

A certification is a perimeter, not a seal

- The July opinion on v0.1.22 names its perimeter: full Stata runtime on Linux x86-64; macOS and Windows only as *structural* build gates. Out of scope, listed: power loss, SIGKILL, a genuinely full disk, NFS/SMB, antivirus interference.
- It was written to be **self-revoking** — any failure of the tag pipeline or of the published artefact cancels it.
- August's certification of v0.1.28 is a different object: audit, remediation, and the release gates quoted earlier.
- v0.1.29 did not change the certified engine at all — only the .ado dispatcher and the ten dialogs. A 1 September audit of that same engine still found two S1 semantic findings outside every earlier perimeter.
- v0.1.30, released that night, fixes them and pins them with new tests; three more releases followed in two days — and by the same rule they are a new perimeter, not the old certificate. That is what a perimeter means.

A PASS names a version, an artefact, a platform and a perimeter.

The provenance gap we cannot close

- The reachable Git history begins on 23 June, with a single commit of 75,895 lines that already contains v0.1.7, the tests, the benchmarks and the early audits.
- The first eleven days — and the commits the June audit reports cite — are **no longer in the object database**.
- Those reports and the changelog still describe that period, but it is not a reproducible chain.

If a project is going to claim an audit trail, the trail has to survive its own housekeeping.

Three tools, three jobs

<code>import parquet</code>	StataNow's native, eager import into memory
<code>pq</code>	made high-performance Parquet I/O in Stata a practical reality, and supplied the packaging and plugin prior art
<code>parqit</code>	adds a lazy manipulation and exploration layer, and a second data path that never loads the result

Building `parqit` meant first cataloguing how a Stata–columnar bridge can go wrong: column manifests keyed by position, names lost to sanitising, missing-value round-trips. Those hazard classes are **engine-independent** — they threaten any such bridge, including this one — and each became an invariant test here.

`pq` made the bridge practical; `parqit` turns that bridge into a manipulation layer.

Where it is, and where it is not

- Installable today **from GitHub**, in one line, for Linux, Windows and both macOS architectures — or from a downloaded zip on an air-gapped cluster.
- **On SSC**: the candidate went to the archive maintainer in late August; he installed it on an ARM Mac, asked for the example do-files, and published the revised one on 28 August — `ssc install parqit`, all installs v0.1.29 with the two crash-course do-files. GitHub is ahead (v0.1.33); an SSC update will follow.
- Performance evidence is *dated and versioned*: one user's field test on 2 September and an in-house benchmark from June, both on an earlier slide. No universal ranking is claimed against any other package.

Shipping somewhere is not publication everywhere.

What this case adds to the AI story

The companion talk on `xhdfc` covers the protocol: the write lock, the contract, the oracle hierarchy, the GO/NO-GO. `parqit` used the same evidence discipline — what changes is the *object under test*.

- An estimator can be checked against an exact solution. A **translation between two languages** cannot: the oracle has to be the other system itself.
- The failures were not crashes or slow convergence. They were **plausible tables** — a merge, a frequency count, a saved file.
- And an AI-written test can make a misunderstanding permanent, which is why the oracle must live outside the system under test.

Here the lesson is semantic translation, not another workflow diagram.

Takeaways

- 1 **Think in views and plans; explore first, load last.** The verbs are the ones you already know; what is new is that they are lazy, that the ceiling is disk rather than RAM, and that the result may never need to enter Stata.
- 2 **The dangerous bugs at a language boundary return plausible data with `rc 0`** — so rank findings by what a user would fail to notice.
- 3 **Verify outside the system under test.** Four oracles, four questions — and a test written from the same misunderstanding as the code proves nothing.
- 4 **Say the perimeter out loud:** which version, which artefact, which platform, which scope.

Thank you

Resources

- Repository, releases and documentation: <https://github.com/reisportela/parqit>
- Install: `ssc install parqit, all` — or the latest release, `net install parqit, from("https://github.com/reisportela/parqit/releases/latest/download")` replace
- Then: `parqit version, parqit selftest, help parqit`

parqit is co-authored with **Rute Costa**, **Paulo Guimarães** and **Marta Silva** (BPLIM, Banco de Portugal). We thank **Jon Rothbaum**, whose pq is the prior art and whose care for correctness shaped this package's test suite; **Kit Baum**, who tested and published the SSC candidate; and the DuckDB project.

parqit is not affiliated with StataCorp, and is provided as is.

miguel.portela@eeg.uminho.pt