What software does social science run on?

Measuring validated use in replication code

Gaurav Sood

Abstract

Software that research depends on earns its authors very little. A metric that could change this is a signal, and a signal is informative in proportion to what it costs to produce. The available ones are cheap. A download is one fetch of a file, which a build server or a more frequent release schedule will supply in quantity, and a prose mention is one sentence in a methods section. Code deposited with a published paper is costly by comparison: economics and political science journals now collect it and a growing number run it before publication, so adding one to a package's count takes a published paper that loads the package. I parse every R, Python and Stata file in 8,349 such deposits from 69 journal collections and count what each of 3,223 packages is loaded by. Three results follow. The most loaded package is `estout`, a Stata command that formats regression output into a table, and 3 of the four most loaded also present results rather than estimate them. Stata appears in 1.8 times as many deposits as R and is absent from the literature that counts research software, because that literature relies on public registries such as CRAN and PyPI to say which project a library name belongs to and Stata has no such registry; I reconstruct one from the Statistical Software Components archive and release it. A long tail of heavily used Stata software belongs to no registry at all, which caps what a credit system built on registry lookups can observe.

Contents

The most widely used piece of third-party software in social science replication code is a table formatter. `estout`, a Stata command whose job is turning regression output into a formatted table, appears in 39% of the Stata deposits in this corpus, ahead of every estimator.

Software that research depends on earns its authors very little (Howison and Herbsleb 2011; Smith et al. 2016). Someone choosing between writing a paper and writing a package is choosing between something a hiring committee knows how to value and something it does not, and that calculation is one reason a good deal of research software never gets written (Sood 2021).

A better count would change that calculation, which makes the question what the available counts actually count. A download is one fetch of a file. A continuous-integration job that installs a package on every commit produces hundreds of them, a mirror produces more, and a maintainer who cuts more frequent releases collects more again (Sood 2021). A prose mention is one sentence in a methods section (Schindler et al. 2022), recording what an author thought to write down.

A metric used to allocate credit is a signal, and a signal is informative in proportion to what it costs to produce (Spence 1973). Both of these are close to free. An author who wants a larger download count can change their release schedule, so the committee reading that number learns as much about release practice as about use, and a metric that cheap invites the behaviour it is supposed to reward.

Deposited code is not cheap. Economics and political science journals now ask authors to deposit the code behind an accepted paper, and some journals run that code before publication (Stodden et al. 2018; Vilhuber 2020). Adding one to a package's count there takes a paper accepted at such a journal, with the package loaded in the code behind it. I call a package's appearance in that code **validated use**.

I measure validated use across R, Python and Stata in 8,349 deposits from 69 journal collections. Three results follow. Packages that format output sit at the top of the table, ahead of every estimator. Stata appears in 1.8 times as many deposits as R and is missing from the studies that count research software, because those studies rely on public package registries and Stata has none. And a long tail of the software in daily use belongs to no registry either, which caps

what any registry-based credit system can see.

1 What validated use means

The estimand:

> Among replication deposits in collection J containing at least one analyzable script in language L, the share that statically reference package P.

The unit is the deposit, not the mention. A deposit calling `ggplot2` two hundred times is one user of `ggplot2`, and a mention-weighted count would rank whichever deposits happen to be largest.

Static reference is not execution. A deposit may load a package in a superseded script, inside a dead branch, or in setup code that never runs, and code kept out of the deposit is invisible to us. A replication package is a curated artifact rather than a forensic copy of an environment (Trisovic et al. 2022). Reachability from master scripts is reported as a sensitivity analysis, which bounds dead code without pretending to observe execution.

Reference also means direct reference. A deposit loading `fixest` also runs `Rcpp` and `sandwich`, and neither is counted. That is the right choice for a credit metric: what an author chose and what their choice pulled in are different quantities, and crediting the closure would hand the top of the table to whatever the popular packages happen to depend on.

Expanding the closure is a separate exercise, and possible in two of the three languages. R declares dependencies in `DESCRIPTION`, Python in a distribution's `requires_dist`, and both can be walked mechanically. An SSC `.pkg` manifest records them, when it records them at all, as prose in a free-text field. The `reghdfe` manifest reads `Requires: Stata version 11.2 and ftools from SSC (q.v.)`, so extracting a graph from it means parsing English.

The closest prior work executed R code from over 2,000 Harvard Dataverse deposits to study reproducibility (Trisovic et al. 2022). This paper takes a similar corpus and asks which libraries the code loads, across three languages instead of one.

2 Results

Every count pools both repositories: 1,272 deposits of economics from Zenodo and 7,077 of political science from Harvard Dataverse, across 69 collections. The released table carries the split beside every pooled total, because the two halves are very different sizes and a pooled number with no breakdown asks a reader to take the composition on trust.

Every count is a share of the deposits **at risk**, meaning those that contain analyzable code in that language. A share against any other denominator is not comparable to anything.

Literate documents make that denominator harder to compute than it looks. A Jupyter notebook's file type is `ipynb` and the code inside it is Python, so counting file types alone puts a deposit whose only Python lives in a notebook into the numerator of every Python package it loads and into the denominator of none. A deposit therefore counts as at risk for a language when it holds an analyzable file in that language *or* when a reference in that language came

Table 1: Most used third-party packages, by language

language	package	deposits	of	share
stata	estout	2440	6212	39%
stata	outreg2	1039	6212	17%
stata	coefplot	986	6212	16%
stata	reghdfe	800	6212	13%
stata	ivreg2	333	6212	5%
stata	binscatter	227	6212	4%
stata	spmap	195	6212	3%
stata	parmest	180	6212	3%
stata	unique	172	6212	3%
stata	clarify	171	6212	3%
r	ggplot2	1650	3518	47%
r	dplyr	1384	3518	39%
r	foreign	1162	3518	33%
r	stargazer	1091	3518	31%
r	tidyverse	977	3518	28%
r	xtable	761	3518	22%
r	plyr	619	3518	18%
r	lmtest	583	3518	17%
r	haven	566	3518	16%
r	scales	534	3518	15%
python	pandas	312	406	77%
python	numpy	291	406	72%
python	matplotlib	192	406	47%
python	scipy	156	406	38%
python	scikit-learn	89	406	22%
python	statsmodels	75	406	18%
python	seaborn	60	406	15%
python	nltk	44	406	11%
python	tqdm	31	406	8%
python	requests	28	406	7%

Note: The ten packages appearing in the most deposits in each language. *deposits* counts deposits referencing the package; *of* is the number of deposits containing analyzable code in that language, which is the only denominator these shares are comparable against. A deposit counts once however many times it calls the package.

out of it. The correction is exact for R, Python and Stata: every deposit with chunks in one of them yields at least one reference in it.

2.1 The most-used research software formats tables

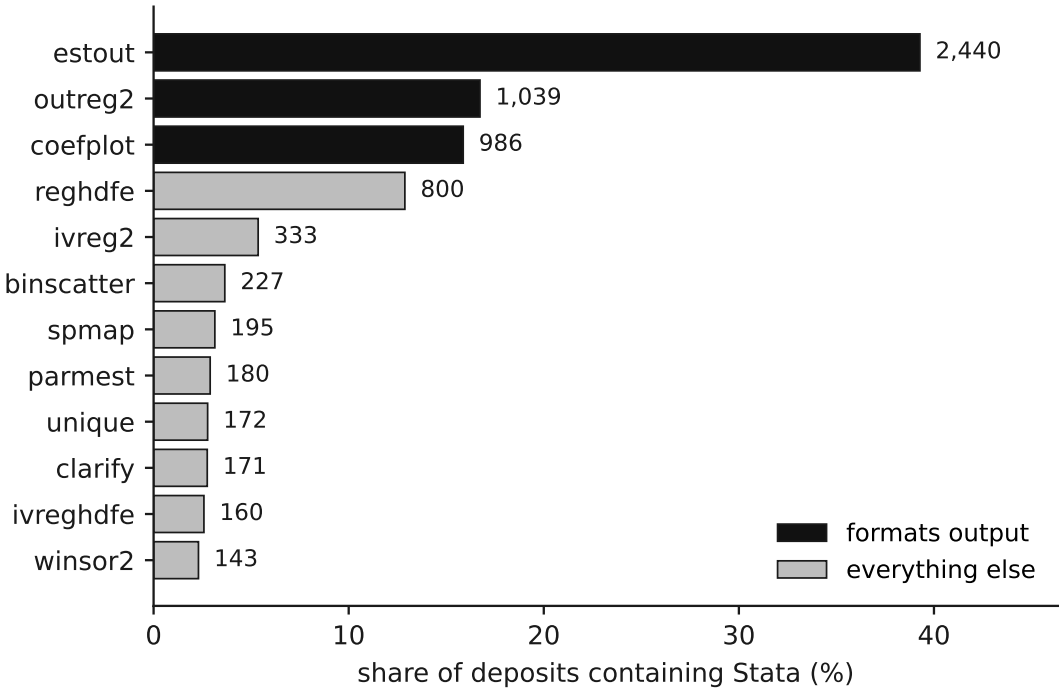

Figure 1: The twelve most used Stata packages, by share of deposits

Note: Packages whose purpose is presenting results rather than estimating anything are filled dark. Counts to the right of each bar are deposits. Shares are of deposits containing analyzable Stata.

Of the ten most widely used Stata packages (Table 1), 3 format output instead of estimating anything: `estout`, `outreg2`, `coefplot`. Fig. 1 shows the same ordering with the formatters marked. The single most used is `estout`, in 39% of Stata deposits, ahead of every regression command.

For a credit metric this has an awkward implication. A large share of what research depends on is the machinery for turning estimates into a publishable exhibit, and whoever writes that machinery has produced something the field uses constantly and cites almost never.

2.2 Most deposits contain Stata, which studies of research software omit

6,219 deposits contain analyzable Stata against 3,510 with R and 388 with Python, roughly 1.8 Stata deposits for every R one, and Fig. 2 shows the split is disciplinary rather than incidental.

Studies of research software cover R and Python, the two languages with public package registries. The omission is not random with respect to what it leaves out. It drops the language most of this work is written in, and it drops it because counting Stata is harder, so

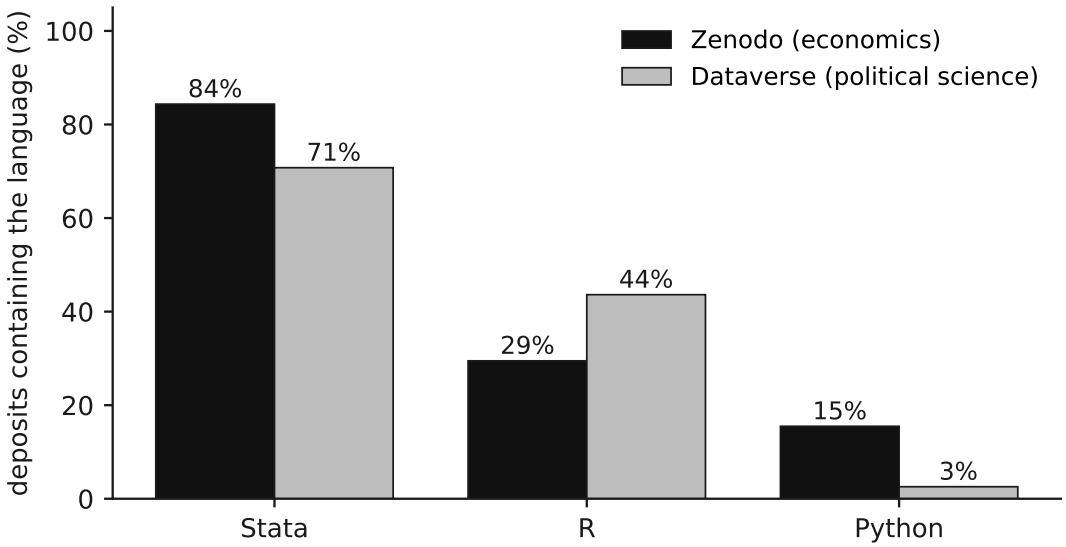

Figure 2: Languages by repository, and so by discipline

Note: Share of deposits containing at least one analyzable file in each language. Zenodo's verified collections are economics journals; the Harvard Dataverse collections are mostly political science. A deposit may contain more than one language, so the bars within a repository do not sum to one hundred.

the discipline that uses Stata most disappears from the record for a reason that has nothing to do with how much software it uses.

2.3 The formatter leads every journal; second place follows the field

`estout` is the most loaded Stata package in 12 of the 12 largest collections (Table 2), in economics and political science alike.

Second place is not the same everywhere, and it separates the two fields. Across all 29 collections holding at least 50 Stata deposits, `reghdfe`, a fixed-effects estimator, outranks `coefplot` in 8 of the 9 economics collections, and `coefplot`, which draws coefficient plots, outranks `reghdfe` in 20 of the 20 political science ones.

Repository is the obvious confound, since the frame's two halves are nearly one discipline each, and a 2024 scrape differs from a 2026 one in more than subject. It survives the check. 5 of those economics collections deposit to Harvard Dataverse, so for them the repository, the vintage and the collection method are all held fixed against the political science journals beside them, and the split still appears.

Pooling is what makes this visible. A single ranking over the whole corpus puts the formatters on top and conceals that the tool underneath them differs by field.

2.4 Some of the most-used software is in no registry at all

`grc1leg` is the most widespread Stata command the resolver cannot attribute to any package, and it appears in 315 deposits. StataCorp distributes it from their own website and no registry

Table 2: Most loaded Stata packages in the twelve largest journal collections

collection	field	deposits	#1	#2	#3
restat	economics	775	estout (281)	outreg2 (189)	reghdfe (130)
jop	political science	492	estout (186)	coefplot (82)	outreg2 (66)
ej-replication-repository	economics	455	estout (296)	reghdfe (164)	outreg2 (127)
isq	political science	364	estout (77)	clarify (40)	outreg2 (32)
ajps	political science	354	estout (116)	coefplot (53)	outreg2 (52)
restud-replication	economics	353	estout (209)	reghdfe (126)	outreg2 (76)
BJPolS	political science	285	estout (123)	coefplot (66)	outreg2 (31)
PSRM	political science	253	estout (79)	outreg2 (39)	coefplot (31)
the_review	political science	244	estout (109)	coefplot (57)	outreg2 (39)
polbehavior	political science	221	estout (52)	coefplot (42)	outreg2 (22)
internationalinteractions	political science	213	estout (45)	outreg2 (26)	clarify (23)
researchandpolitics	political science	197	estout (54)	coefplot (34)	outreg2 (25)

Note: Collections ranked by deposits containing analyzable Stata. *deposits* is that count; each package column gives the package and the deposits loading it. Field is assigned from a named list in the source rather than from the repository, because several economics journals deposit to Harvard Dataverse alongside the political science ones.

indexes it. The same pattern holds below it: `dcdensity` from its author's page, `btscs` from a replication archive, `prvalue` and `listcoef` from the SPost suite distributed with a textbook.

A credit system built on registries cannot see any of them, and indexing SSC more carefully will not help, because these were never on SSC. That caps what anyone can currently count about Stata, and the cap is a property of the ecosystem rather than of the resolver. R and Python have nothing comparable: a package outside CRAN or PyPI is rare.

3 What better counts are for

These counts have two uses, and the second is the one that motivated collecting them at this scale.

The first is credit itself. A package in 39% of the Stata deposits at policy-verified journals gives a software author something a hiring committee can check, which a download count does not. Every package's number is published rather than summarised, so the claim can be verified by whoever is evaluating it.

The second is deciding what to test. Work on software reliability has to choose which packages to examine, and choosing by reputation reproduces whatever the field already believes. The sibling project `kasauti` runs probes against every release of a package to find where results changed without anyone saying so, and it picks targets partly by how often a function appears in real code. Validated-use counts answer that question directly, and they point at the packages where a silent change would touch the most published work.

4 Sampling frame

The inclusion rule comes from outside this paper. It is the journal list maintained by the **Social Science Data Editors** in the Data and Code Availability Standard: journals whose data-and-code policy the editors *actively verify*. That is a criterion a reader can check, and it avoids the trap of a frame assembled from whatever a keyword search happened to return.

The frame spans two repositories, which is not a convenience: they hold different disciplines. Zenodo's verified collections are economics; Harvard Dataverse's journal collections are mostly political science, and a study of only one would describe a field while claiming to describe social science.

Table 3: The sampling frame, by repository

repository	collections	deposits
Zenodo	6	1601
Harvard Dataverse	74	14674
total	80	16275

Note: Journals whose data-and-code policy the Social Science Data Editors record as actively verified. Zenodo deposits are the counts the communities declare; Dataverse deposits are enumerated collection by collection.

The Dataverse figure is enumerated rather than declared. One request returns a journal collection's entire dataset list, so these are counted deposits and not a figure supplied by the repository, and a collection too large for one listing was paged instead.

The enumeration turned up two things that belong in the record, because a frame that only reports its successes is not a frame. One journal collection present in a 2024 snapshot, `regionalstatistics`, no longer exists: Harvard answers "Can't find dataverse with identifier". And six collections returned nothing on one pass, of which five succeeded minutes later, transient, and distinguished from the one that is genuinely gone, because retrying the first kind works and retrying the second never will.

10 DCAS journals are recorded as **unlocated** rather than omitted: eight AEA titles live on openICPSR, and two could not be placed. A coverage gap belongs in the frame as a row, not in a silence.

Every collection is verified before use. Zenodo does not reject an unknown `communities=` filter, it ignores it and returns all 7.1 million records, so an unverified slug would substitute the entire archive for one journal, an error shaped like abundance.

Of 1,601 Zenodo deposits attempted, 1,494 (93%) yielded files. Table 4 itemises the rest rather than summarising them, because the reasons are not interchangeable. A size cap is a deliberate choice, an unreassembled multi-part archive is a format the pipeline declined to handle, and a failure is a bug or an outage.

Table 4: Why a deposit yielded no code

reason	deposits
archive over the size cap	66
deposit contains no code files	39
multi-part archive, not reassembled	2

Note: Zenodo deposits that produced no analyzable file, by reason. The size cap and the unreassembled multi-part archives are deliberate choices and are recoverable; the single failure is a zip using a compression method Python does not implement.

Table 5: How each language is read

language	mechanism
R	tree-sitter syntax walk
Python	stdlib ast, tokenize fallback
Stata	purpose-built lexer, SSC command-to-package index
.Rmd / .qmd / .Rnw	knitr chunk splitter, routed per engine
.ipynb	per cell, kernel language from the metadata

Note: Literate formats are split first and each chunk routed to the extractor for its own language, so a Python chunk in a knitr document resolves against PyPI rather than against CRAN.

5 Measurement

Two choices carry most of the weight. The first is that strings and comments are excluded structurally instead of being stripped out beforehand. In a syntax tree, `cat("run library(dplyr)")` is a call whose argument is a string node; those bytes never form a call, so a walk over call nodes cannot see them. Regex detection offers no such guarantee, and the validation section measures the difference.

The second is that Stata needed an artifact that did not exist. R has CRAN and Python has PyPI, so an import can be looked up. Stata has no machine-readable registry, so a command→package index had to be reconstructed from SSC distribution manifests, 3,967 packages and 7,468 commands, and is released alongside this paper. Without it Stata cannot be measured at all, which is the mechanical reason prior work has left out the language social science uses most.

Shares are not comparable across languages, and the denominators are why. `pandas` appears in 77% of Python deposits, because Python in a replication package almost always means a dataframe pipeline. Stata's most common package reaches 39% over a much longer tail, because Stata does natively what a Python deposit imports a library to do. Reading the gap as "Python users are more consistent" would misread it: these are shares of different kinds of deposit, and the two languages are not substitutes for each other in this corpus.

Table 6: Mentions that could name a package, and those left unresolved

language	candidate mentions	unresolved	rate
stata	517793	39402	7.6%
r	150462	2103	1.4%
python	92026	6414	7.0%

Note: Restricted to mentions that could name a third-party package, so Stata builtins, standard libraries and programs a deposit defines itself are excluded from both columns. An unresolved mention is one the resolver cannot attribute, and not evidence that no package exists.

6 Validation

Each check below establishes one thing, and the text says which.

Registry resolution measures coverage, not precision: a local module can share a name with a real distribution, and nothing here would notice. 6.3% of third-party-candidate mentions resolve to no registry, and every such name is enumerated in the released tables.

The denominator does more work than the rate. Against *all* mentions the same figure is 0.7%, a flattering way to describe identical data: it divides by 6,200,926 Stata builtin calls that were never candidates for registry resolution and so could never have been unresolved. The rate reported here uses the mentions that could name a package, which is 9 times larger and is the number a reader should use to discount the coverage claim.

So does the unit. Per deposit, which is the unit used everywhere else in this paper, the distribution sits nowhere near the pooled figure:

Table 7: Unresolved share, pooled and per deposit

statistic	unresolved share
pooled over mentions	6.3%
median deposit	0.0%
90th-percentile deposit	22.7%
deposits fully resolved	70%

Note: Restricted to mentions that could name a third-party package. The pooled figure and the median deposit answer different questions, and the gap between them is the point: the pooled rate describes a minority of deposits.

Every candidate mention resolves in 70% of deposits. The pooled figure is accurate and describes a minority: read it as an upper bound on how much of a typical deposit goes unnamed.

Language explains most of what is left. Stata carries 82% of all unresolved mentions, and the bulk of that traces to the registry gap described above, where `grc1leg` and the SPost commands are used widely and indexed nowhere. No amount of indexing SSC reaches them. Any measurement of Stata software use, this one or another, is bounded by that gap rather than by how carefully the indexing was done. What remains after it is lexer error, enumerated in the limitations below and not netted out.

An unresolved mention means no package could be named for it, which is a different thing from no package existing. Two constructs make that distinction explicit instead of letting it inflate the rate: 1,398 Python relative imports (`from .models import Constants` names a module in the deposit, not a distribution) and 508 names built at run time (`library(pkg, character.only = TRUE)`, where the package is a variable and no static analysis can know it). Both are reported in their own categories. An earlier version of this pipeline folded both into `unknown`, which overstated the unresolved rate and credited some of them to real packages that happen to share a name: `models`, `results` and `log` are distributions on PyPI as well as things people call their own files.

Stata's builtin list is measured rather than curated. It is the one place where an omission does damage: a command that official Stata provides but the list lacks falls through to the SSC index, and if some package ships a file of that name, the mention is credited there. So every command in the corpus was checked against StataCorp's public help server, which is the vendor's own answer to whether a command is official.

That check changed more than expected. It moved 358 commands out of "resolves to nothing", including `ds`, `pca` and `testparm`, and it removed every case where a command claimed by both official Stata and an SSC package had been credited to the package. With the curated list alone the unresolved rate on this same corpus is 12.3% against 6.3%, and 1,648 mentions would have gone to an SSC package that merely ships a file of the same name. Much of what looked like a coverage gap was ignorance of Stata on this end, and the method was never the limit.

Reading the check takes care. Every query returns HTTP 200, including for names that do not exist, so existence is read from the body text; and the server documents concepts and functions as well as commands, so pages from the User's Guide and function manuals are excluded. Without that second filter the system variables `_n` and `_b` and the function `e()` are reported as official Stata commands.

Agreement with a second tool gives a different kind of evidence. `renv::dependencies()` (1.2.3) over 3,336 of the corpus's 3,510 R deposits gives a mean per-deposit Jaccard of 0.96 and a pooled Jaccard of 0.95; 74% of deposits agree exactly. Both are reported because they answer different questions: the mean weights every deposit equally, the pooled figure every package-deposit pair, so a headline cannot be flattered by deposits with one obvious package in them.

A further 63 deposits are excluded, because renv returned nothing at all for them. It runs with `errors = "ignored"` and drops a file it cannot decode in silence, so those are files one side never read, and scoring them would measure a decoding difference as a disagreement. This is the treatment the oracle already gives files R cannot parse.

Where the two do disagree, the direction is one-sided: across the worst cases this extractor found 341 packages renv did not, against 96 the other way. The pattern behind them is the same decoding problem, partially masked: in the worst Zenodo deposit renv reports a single package, and every one of its twenty-three R files carries bytes outside ASCII. Read that way, the figures above bound agreement from below.

This is convergent validity, and it is weaker than accuracy. `renv` performs the same kind of static scan over the same files and shares the same blind spots: a package named only in a

string, or loaded through a variable, is invisible to both. Agreement therefore shows that two implementations of one idea agree, and says nothing about whether either is right. What it can catch is an implementation error on this side large enough to show against an independent codebase, which is why the per-deposit disagreements are written out and not reduced to the headline.

Syntactic accuracy has a true oracle. For "does this file contain a call to `library` whose argument is `x`", R's own parser defines the answer, so disagreement is an error on this side and the error rate is exact. Running R's parser over 35,470 `.R` files across both repositories and recursing its expression trees gives precision 1.0000 and recall 1.0000 over 94,764 name-file pairs.

Both repositories, because for a while it was one. The check globbed the Zenodo directory and its result was reported as though it described the corpus, which left 15,560 Dataverse files vouched for by a measurement that had never read them. Extending it found a disagreement immediately, and the disagreement was the oracle's.

A further 1,433 files R itself could not parse. Those are excluded from the score instead of counted as clean, since a file nobody can parse may well have dependencies, and scoring it as empty would award credit for finding nothing in it.

How that figure was reached matters as much as its value, because a broken comparison also produces perfect agreement. Six disagreements were adjudicated by running R and watching what it did, and five were the oracle's fault. It unboxed a one-element result into a bare string, so a lone `LearnBayes` was read as eight packages named after its letters. It skipped every `character.only = TRUE` call, though a string literal there is still a literal. It could not see through `base::library(c)`, because such a call's head is itself a call. It never descended into a function's formals, hiding every `pkg::fun()` used as a default argument. And it tested the flag against the literal `TRUE` while the corpus writes `character.only = T`, which is a symbol, so `library(pkg, character.only = T)` inside a loop was read as a dependency named `pkg`. The sixth was an error here: `"m"::baz()` is legal R and the extractor dropped it.

Where the two disagreed about `requireNamespace(pkg)`, running R settled it: `library(pkg)` with `pkg <- "stats"` errors, so the symbol is the name, while `requireNamespace(pkg)` returns `TRUE`, so the symbol is a variable.

A score tuned against the data it grades proves little, so the extractor is also checked against ground truth written by someone else: `renv`'s own test fixture, which enumerates thirteen loader spellings and states in its comments which ones must be found. That check failed where the oracle had passed, because two implementations sharing a blind spot look exactly like two implementations agreeing. It found `xfun::pkg_attach(c("g", "h"))` and `pkg_attach2("i", "j")` unhandled. Both are handled now, and the fixture runs as a test.

The oracle bounds one thing exactly, whether the tree-sitter walk sees what R sees. It says nothing about Stata, nothing about resolution against CRAN, and nothing about whether a `library()` call means the analysis used the package.

One claim is missing from all of this. No probability sample has been hand-labelled, so there is no human-validated precision or recall against research *use*. The labelling frame is designed and can be run; it has not been.

7 Limitations

The gap between reference and execution, set out in the definition above, is bounded by the reachability analysis and not closed by it.

Costly to inflate is not impossible to inflate. An author can load a package they did not need, and a package bundled into a template that circulates in a subfield collects counts without anyone choosing it. The cost of a unit is the cost of the paper, not of the package, so what this measure resists is the cheap manipulation that download counts invite, and not a determined one.

The counts also weight every deposit alike. A package used once in each of fifty short papers and one used throughout a single large project score fifty to one, which is the right answer for "how many pieces of research chose this" and the wrong one for "how much research depends on this".

The corpus is two collections with different instruments behind them. The Harvard Dataverse material is a January 2024 scrape that retained only `.do`, `.r` and `.py` files; the Zenodo material is whole deposits, including notebooks, knitr documents and `.ado`. A package used mainly inside a literate document is therefore under-counted on the Dataverse side, and the two collections are two years apart.

Recomputing every ranking on the file types both collections share measures what that costs, and the cost differs by language. For Stata and R all twenty top places hold. For Python they do not: `ipython` and `spacy` drop out of the top twenty and four other packages move, none of them above rank fourteen. Python is the language whose code most often lives in a notebook, and notebooks are what the 2024 scrape did not collect, so this is where the gap should appear. The top ten in every language, which is what the tables in this paper print, does not change.

Deposits whose only content exceeds the archive size cap yield no code, and the tail this excludes is not data without code. Code files scale with archive size: a median of 13 code files in the smallest quartile of archives against 46 in the largest. The gap therefore runs against large, code-heavy projects rather than against large datasets.

Multi-part archives are not reassembled. 2 deposits split a package across `.z01`, `.7z.002` or `.partN.rar` segments. Only the tail segment carries the central directory, so it downloads at exactly the size the repository states and lists every member before failing on the first read, which presents as a corrupt download when nothing was corrupted. These are counted, not retried, since no number of retries assembles one segment.

Stata resolution reflects today's status, not the status when the code was written. A command that was user-written in 2010 and later absorbed into official Stata resolves against what it is now, which can manufacture a trend for anyone reading these counts over time.

Two false positives survive, and are named here instead of suppressed. SSC packages whose only command is a single common token attract stray matches from Mata-style bare assignments outside a `mata:` block, which the lexer cannot recognise as Mata. Currently `fastcd` (command `c`) in 4 deposits, `wordy` (command `word`) in 1 deposits. A rule against single-letter commands would hide the next instance, and since the count is computed, the list shrinks on its own as causes are fixed.

Three lexer gaps remain open and countable. Stata keywords reach command position in contexts the lexer does not model: `in` and `using` after an old-style `/* */` line continuation, and `str` inside an `infile dictionary` block, where a line such as `str FACT_ID 5-12` declares a storage type rather than calling anything. Together that is 2,477 mentions across 56 deposits. They inflate the unresolved rate and do nothing else, since a keyword cannot collide with an SSC package name and so nothing is misattributed. A blocklist would remove the symptom and leave the contexts unmodelled, so the number is reported instead.

MATLAB, SAS and SPSS are reported as language presence only. MATLAB has no import statement and no resolvable registry, so attributing toolboxes would be unbounded error.

Which version ran is mostly not recorded. Code names the packages it loads and almost never the release it loaded them at, so these counts pool a package across its whole history. Where a file does say, this release now records it: a Stata `version` statement in 777 of the 6,100 deposits holding a do-file, an `renv.lock` or a bundled `DESCRIPTION` in 28 of 3,529 with R code, and a notebook's own note of the interpreter that executed it. At 13% coverage for the best-covered of these, the environment table supports checking particular deposits and not estimating a distribution, which is why the denominator ships beside it. A reader who wants the share of economics papers still on an unsupported Stata cannot have it from this corpus. A reader who wants to know what the deposits that do say were running can.

8 Conclusion

Research software is credited badly because the numbers available to credit it with are cheap to produce. This paper offers one that is not: a count of the packages that published, policy-verified replication code loads, over 8,349 deposits and 3,223 packages.

What the counts show is not what a download ranking would suggest. The software social science leans on hardest is the software that prepares its exhibits, and `estout` leads every one of the largest journal collections in both disciplines. Underneath that, the tools differ by field rather than by repository. The language carrying most of this work, Stata, is missing from the literature on research software for a mechanical reason: it has no registry mapping a command to a package, so this paper built one. And a long tail of software that researchers use constantly sits outside every registry, which is a ceiling on what any credit system built on registry lookups can reach. That ceiling belongs to the ecosystem, and no amount of care in the method moves it.

The counts are static reference and not execution, they weight every deposit alike, and they cover two disciplines and not the whole field. Each of those is measured above. What they support is modest and concrete: a software author can point at a number a committee can check, and work on software reliability can choose its targets by how much published research would be touched.

9 Data and code availability

The Stata command-to-package index is deposited at 10.5281/zenodo.21926099 under CC0, with 7,468 commands across 3,967 packages. It is released separately from this paper because

its usefulness does not depend on this corpus: anyone counting Stata needs the same mapping.

The pipeline makes four things, and they are not equally public. The **corpus** is the deposited code itself, 217,573 files, too large to redistribute and in any case already public at the archives it came from. The **atomic record** is one row per mention, carrying the package, the function where the source names one, the file, the line and a snippet. The **aggregates** are sums of that record, by package, year, journal and function. The **registry** is the instrument rather than a result: the CRAN, PyPI and SSC name tables that turn a token into a package, pinned so a count can be reproduced rather than merely repeated.

The atomic record and the aggregates are deposited at 10.5281/zenodo.21943909 under CC0, covering 3,223 packages, with the counts by year and by journal alongside them. Releasing the mention rows and not only their sums is what lets a reader disagree with a decision made here and recount without re-parsing the corpus. The same tables are served at https://recite.github.io/softverse/data/, and every package can be looked up by name at https://recite.github.io/softverse/lookup/, so a software author or an evaluating committee can check a number instead of taking the top ten in Table 1 on trust. The archived version is the one to cite: a repository serves whatever it holds today.

The extraction pipeline, the resolver and the frame are at https://github.com/recite/softverse.

10 Reproducibility

The corpus, the registry snapshots and the extractor are all pinned. Every mention carries the extractor version, grammar version and registry lock that produced it, plus its source line and a snippet, so any number here can be traced to the text it came from without re-running anything.

Table 8: Pinned registry snapshots

registry	sha256
cran	aefeb12f742cad24
cran_archive	5e267904e40ed060
bioconductor	3ffed30246d706ae
pypi	751e02220c921dea
julia_general	9cf5a9b5f71da392

Note: Every mention carries the lock identifier of the registry snapshot it was resolved against, so a count can be reproduced against the registry as it stood rather than as it stands.

11 Acknowledgements

Daniel Weitzel contributed to an earlier version of this project.

12 Competing interests

The author has no competing financial interests. Two intellectual ones are worth naming. The argument that better counts would increase the production of research software is taken from the author's own earlier writing (Sood 2021), and the paper releases the artefact it argues the field lacks, so it both proposes a metric and supplies it.

13 Funding

This research received no external funding.

14 Author contributions

Sole author. Conceptualisation, methodology, software, data curation, formal analysis, and writing.

15 References

Howison, James, and James D. Herbsleb. 2011. "Scientific Software Production." *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, ahead of print. https://doi.org/10.1145/1958824.1958904.

Schindler, David, Felix Bensmann, Stefan Dietze, and Frank Krüger. 2022. "The Role of Software in Science: A Knowledge Graph-Based Analysis of Software Mentions in PubMed Central." *PeerJ Computer Science* 8: e835. https://doi.org/10.7717/peerj-cs.835.

Smith, Arfon M., Daniel S. Katz, and Kyle E. Niemeyer. 2016. "Software Citation Principles." *PeerJ Computer Science* 2: e86. https://doi.org/10.7717/peerj-cs.86.

Sood, Gaurav. 2021. *Countpy: Incentivizing More and Better Software*. https://www.gojiberries.io/countpy-incentivizing-more-and-better-software/.

Spence, Michael. 1973. "Job Market Signaling." *The Quarterly Journal of Economics* 87 (3): 355–74. https://doi.org/10.2307/1882010.

Stodden, Victoria, Jennifer Seiler, and Zhaokun Ma. 2018. "An Empirical Analysis of Journal Policy Effectiveness for Computational Reproducibility." *Proceedings of the National Academy of Sciences* 115 (11): 2584–89. https://doi.org/10.1073/pnas.1708290115.

Trisovic, Ana, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. 2022. "A Large-Scale Study on Research Code Quality and Execution." *Scientific Data* 9 (1): 60. https://doi.org/10.1038/s41597-022-01143-6.

Vilhuber, Lars. 2020. "Reproducibility and Replicability in Economics." *Harvard Data Science Review* 2 (4). https://doi.org/10.1162/99608f92.4f6b9e67.