Downloads are cheap

Validated use as a measure of research software

Gaurav Sood

Abstract

Counts of research software use allocate credit, and the available ones are cheap to produce: a download is one fetch of a file. I propose validated use, a package loaded in replication code deposited at a journal that checks it, as a costly signal, and measure it in 13,245 deposits from 75 economics and political science journal collections, which load 4,565 R, Python and Stata packages. Among packages the corpus uses, validated use and downloads agree loosely, and least where automatic installation is commonest: the rank correlation is 0.61 for Stata, 0.53 for R and 0.36 for Python. Downloads track a package's position in the dependency graph more closely than its use. The most used software prepares exhibits: `estout` is in 43% of Stata deposits. Stata appears in 1.4 times as many deposits as R and is missing from studies of research software because its packages are scattered across SSC, two journals' archives and authors' sites with no index across them. I build that index, 10,147 commands in 5,581 packages, and release it with the mention-level data.

Contents

Research software earns its authors little credit (Howison and Herbsleb 2011; Smith et al. 2016). A hiring committee knows what a paper is worth and not what a package is worth, so useful software goes unwritten (Sood 2019). Part of the problem is the count. The usual measure of how much a package is used is how often it is downloaded, and a download costs almost nothing to produce. This paper proposes a count that is expensive to move, whether a package is loaded in the replication code a journal requires and checks, and compares it with downloads in 13,245 deposits from economics and political science journals. The two disagree most about the packages that matter. Among R packages used in at least twenty deposits, the rank correlation is 0.31, and `stargazer`, the 4th most used, is the 388th most downloaded.

A count used to give credit is a signal, and a signal is worth what it costs to produce (Spence 1973). A build server that installs a package on every commit produces hundreds of downloads, a mirror produces more, and a maintainer who releases more often collects more again (Sood 2019). PyPI's own operators found in 2026 that metadata requests were a large share of the downloads of widely installed packages, and that much of the rest came from misconfigured proxies and runaway build systems (Python Package Index 2026). A mention in prose is one sentence in a methods section (Schindler et al. 2022), and often the sentence is missing: in biology, between a third and a half of software mentions are formal citations, and the version is usually lost (Howison and Bullard 2016).

Deposited code is not cheap. Economics and political science journals ask authors to deposit the code behind an accepted paper, and some run it before publication (Stodden et al. 2018; Vilhuber 2020; Herbert et al. 2024). To add one to a package's count there, someone has to get a paper accepted at such a journal with the package loaded in its code. I call a package's appearance in that code **validated use**.

The cost is what separates this count from two I built before it. In 2018 I counted how often public GitHub repositories import each Python package (Sood and Tran 2018), and in 2025 I

estimated the same from a random sample of repositories (Sood 2025). Both count use rather than fetches, and both share a weaker form of the defect in downloads: anyone can push a repository. Nobody can move a count over verified deposits without publishing a paper.[1]

I measure validated use in R, Python and Stata across 13,245 deposits from 75 journal collections, and ask three questions:

1. **RQ1 (Validity of usage metrics):** How far do validated use and download counts agree, and which software does each overstate?
2. **RQ2 (Registries as instruments):** What does counting through a package registry leave out, in each language, and where does the missing software live?
3. **RQ3 (Field-specific dependence):** Which software does each field depend on?

The paper adds two things to the counts. One is the measurement: a sampling frame, a validated extractor for three languages, and the mention-level record, released so the counts can be recomputed. The other is an index from Stata commands to the packages that provide them, built from the distribution manifests of SSC, the *Stata Journal*, the *Stata Technical Bulletin* and the authors' sites that replication code installs from. Sergio Correia compiled such a list for SSC alone, for the AEA Data Editor's package scanner (Reiner and Vilhuber 2021). Nothing covered the other three archives, and `grc1leg`, one of the most used Stata packages in this corpus, lives in none of the lists an SSC index can hold.

1 Related work

1.1 How the use of research software has been measured

Most of what is known comes from reading papers. In a hand-coded sample of ninety biology articles, most software mentions were informal, a minority were formal citations, and the software mentioned was often impossible to find or to pin to a version (Howison and Bullard 2016). Later work automated the reading. Nearly five thousand PDFs in biomedicine and economics were annotated by hand to train extractors (Du et al. 2021); one such extractor was run over PubMed Central and its output built into a knowledge graph (Schindler et al. 2022); the mentions in the biomedical literature were released at scale (Istrate et al. 2022). The same pipeline shows citation practice improving slowly in the coronavirus literature (Du et al. 2022) and shows retracted articles using less open-source software than matched controls (Schindler et al. 2023).

Citation counts of software are the second instrument, and the studies that use them mostly document why they fail. Software is rarely cited in the databases built to index research outputs (Park and Wolfram 2019). One widely used R package is cited under many inconsistent forms (Li et al. 2019), and the citation formats R packages ask for change over a package's life (Wang and Li 2024). Institutional repositories hold very little software as a recognized output (Carlin et al. 2023).

[1] I proposed counting package use in code instead of downloads in March 2019 (Sood 2019). The first version of this collector, which read imports out of *Journal of Politics* Dataverse deposits, was committed in July 2023 (Sood 2023). It lacked a sampling frame, validation, an index for Stata and a released corpus. This paper supplies them.

All of these read what authors wrote about software. This paper reads the software they ran, which removes the step at which a use goes unrecorded. Three efforts have read code before. One executed the R files in over 2,000 Harvard Dataverse deposits to ask whether they ran (Trisovic et al. 2022). One had students inspect the replication files of more than ten thousand economics, political science and statistics papers and record which *language* each used (Upton et al. 2026), and another classified economics replication packages by file extension to the same end (Kranz 2023). One resolved SSC package names out of economics replication code, to validate a Stata command that pins package versions, and found that the hundred most downloaded packages are 71% of SSC downloads but 93% of the packages publications load (Correia and Seay 2024). None publishes per-package counts across languages with the denominator each share divides by. Two unpublished efforts come close: `renv` run over R scripts in Dataverse deposits (Arel-Bundock 2022), and the AEA Data Editor's scan of do-files for SSC packages during review (AEA Data Editor and Reiner 2021; Reiner and Vilhuber 2021). Downloads remain the standing answer to how much a Stata command is used (Choodari-Oskooei and Morris 2016).

1.2 What a count is for

The case that software work is under-rewarded is older than these measurements. Scientists write software inside a reputation system that pays in publications and citations, which shapes what gets written and whether it is maintained (Howison and Herbsleb 2011), and most learn to write it from peers rather than from training (Hannay et al. 2009). A review of what funders, maintainers and scientists would need to measure, from funding through use to scientific impact, found few measures of use (Howison et al. 2015). The software citation principles are the normative answer: cite software as one cites a paper (Smith et al. 2016). A recent study paired articles with their code repositories and found that a large share have code contributors who are not authors (Brown et al. 2026). It measures who wrote the code behind a paper; this paper measures whose code the paper's analysis ran on.

A count that gives credit will be gamed if it can be. Citation counts are inflated through reference metadata that never appears in the article (Besançon et al. 2024), and researchers choosing what to read respond to whichever indicators are put in front of them (Lemke et al. 2021). So the question to ask of any count is what it costs to move. This paper asks it of downloads and of its own measure.

2 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 that calls `ggplot2` two hundred times is one user of `ggplot2`. A count weighted by mentions would rank the largest deposits.

A static reference is not a run. A deposit may load a package in an old script, in a branch that never executes, or in setup code that never runs, and code left out of the deposit cannot be seen at all. A replication package is a curated artifact, not a copy of the environment that

produced the paper (Trisovic et al. 2022). Where a deposit has a master script, the counts can be recomputed over the files it reaches, and the ranking holds.[2]

A reference also means a direct one. A deposit that loads `fixest` also runs `Rcpp` and `sandwich`, and neither is counted. What an author chose and what the choice pulled in are different quantities, and crediting the closure would put whatever popular packages depend on at the top of the table.

The closure could be added for two of the three languages. R declares dependencies in `DESCRIPTION` and Python in a distribution's `requires_dist`, and both can be walked by machine. An SSC `.pkg` manifest records them, when it records them at all, as prose: the `reghdfe` manifest reads `Requires: Stata version 11.2 and ftools from SSC (q.v.)`.

3 Methods

3.1 Sampling frame

The frame is the list of journals whose data-and-code policy the Social Science Data Editors *actively verify*, from their Data and Code Availability Standard. The frame spans two repositories because the two hold different disciplines: Zenodo's verified collections are economics journals, and Harvard Dataverse's journal collections are mostly political science. The first appendix lists the collections.

Each Dataverse collection's deposits were listed in full through the repository's API, paging where a collection exceeded one page, so the counts are enumerated and not taken from the repository's own figures. One collection present in a 2024 snapshot, `regionalstatistics`, no longer exists. Six collections returned nothing on a first pass; five of them answered minutes later and were retried, and the sixth is the one that is gone.

10 journals on the list are recorded as **unlocated**: eight AEA titles deposit to openICPSR, which this corpus does not cover, and two could not be placed. Every Zenodo community slug was checked before use, because Zenodo ignores an unknown `communities=` filter and returns all 7.1 million records.

Of 1,642 Zenodo deposits attempted, 1,583 (96%) yielded files. Table 7, in the first appendix, gives the reason for each of the rest: an archive over the size cap, a multi-part archive that was not reassembled, or a download that failed.

3.2 Measurement

Strings and comments are excluded by structure. In a syntax tree, `cat("run library(dplyr)")` is a call whose argument is a string node, so a walk over call nodes never sees the text inside it. Regular expressions give no such guarantee, and the appendix measures the difference.

Stata needed an index. R has CRAN and Python has PyPI, so an import can be looked up. Stata has no registry that maps a command to its package, so the mapping was rebuilt

[2] In the 1,965 deposits whose master script's calls can all be followed, the median deposit has 86% of its files on that path, and recomputing over reachable files moves no package among the twenty most used by more than 4 places. The appendix has the details.

from distribution manifests: 5,581 packages and 10,147 commands across four kinds of archive, released with this paper.

One such list already existed for SSC. Sergio Correia compiled a command-to-package list for the AEA Data Editor's package scanner (Reiner and Vilhuber 2021), by a route that shares no code with the crawl here. The two lists hold 6,549 commands in common and name the same package for 6,546 of them. The crawl holds 2,012 SSC commands that list lacks and lacks 257 it holds, most of them internal subroutines.

A manifest lists the files a package ships, which is not the list of commands it provides. Two rules close the gap. A file named by Stata's convention for internal subroutines, such as `_eststo` or `reghdfe_p`, is a helper: a call to it counts as a use of its package, and it is not reported as a command of its own. A journal package is credited with a command only when it also ships that command's help file, because an article's package bundles whatever its example needs; a *Stata Journal* package for calibrating Cox models ships a copy of `grc1leg.ado`. A command found in more than one archive is credited to the first of SSC, the *Journal*, the *Bulletin* and the authors' sites to list it, since journal authors mirror their software to SSC.

One kind of package has no command to find. `egen n = nvals(x)` calls official `egen`, which runs a file named `_gnvals.ado`, and a package such as `egenmore` is a collection of such files. Each `egen` call is therefore read twice, as the command and as the file its function lives in, and the file resolves to official Stata where Stata ships it and to its package where a package does.

The download counts are existing data. SSC's are the monthly hits behind Stata's `ssc hot`, served by RePEc. CRAN's come from one mirror's logs through `cranlogs` (Csárdi 2015). PyPI's are a monthly dump of its public download table for the fifteen thousand most downloaded projects (Kemenade et al. 2026). Each is pinned by date and digest beside the registries.

Shares are not comparable across languages. `pandas` appears in 81% of Python deposits because Python in a replication package almost always means a dataframe pipeline. Stata's most used package reaches 43% over a much longer tail, because Stata does natively what a Python deposit imports a library to do. The two are shares of different kinds of deposit.

4 Results

Every count pools both repositories: 1,358 deposits of economics from Zenodo and 11,887 of political science from Harvard Dataverse, across 75 collections. The released table carries the split beside every pooled total, because the two halves differ in size.

Every count is a share of the deposits **at risk**: those that contain analyzable code in the language. A deposit counts as at risk for a language when it holds an analyzable file in that language or when a reference in that language came out of it. The second clause covers notebooks and literate documents, whose file type is `ipynb` or `Rmd` while the code inside is Python or R; without it a deposit whose only Python lives in a notebook would enter the numerator of every Python package it loads and the denominator of none.

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

language	package	deposits	of	share
stata	estout	3835	8942	43%
stata	coefplot	1671	8942	19%
stata	outreg2	1553	8942	17%
stata	reghdfe	1535	8942	17%
stata	ivreg2	494	8942	6%
stata	grc1leg	490	8942	5%
stata	binscatter	434	8942	5%
stata	egenmore	396	8942	4%
stata	unique	332	8942	4%
stata	spmap	321	8942	4%
r	ggplot2	3252	6471	50%
r	dplyr	3134	6471	48%
r	tidyverse	2398	6471	37%
r	stargazer	2035	6471	31%
r	foreign	1724	6471	27%
r	haven	1427	6471	22%
r	xtable	1408	6471	22%
r	scales	1274	6471	20%
r	stringr	1148	6471	18%
r	tidyr	1090	6471	17%
python	pandas	732	909	81%
python	numpy	684	909	75%
python	matplotlib	475	909	52%
python	scipy	335	909	37%
python	scikit-learn	271	909	30%
python	seaborn	203	909	22%
python	statsmodels	176	909	19%
python	tqdm	124	909	14%
python	nltk	104	909	11%
python	requests	79	909	9%

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, and the share divides by it. A deposit counts once however many times it calls the package.

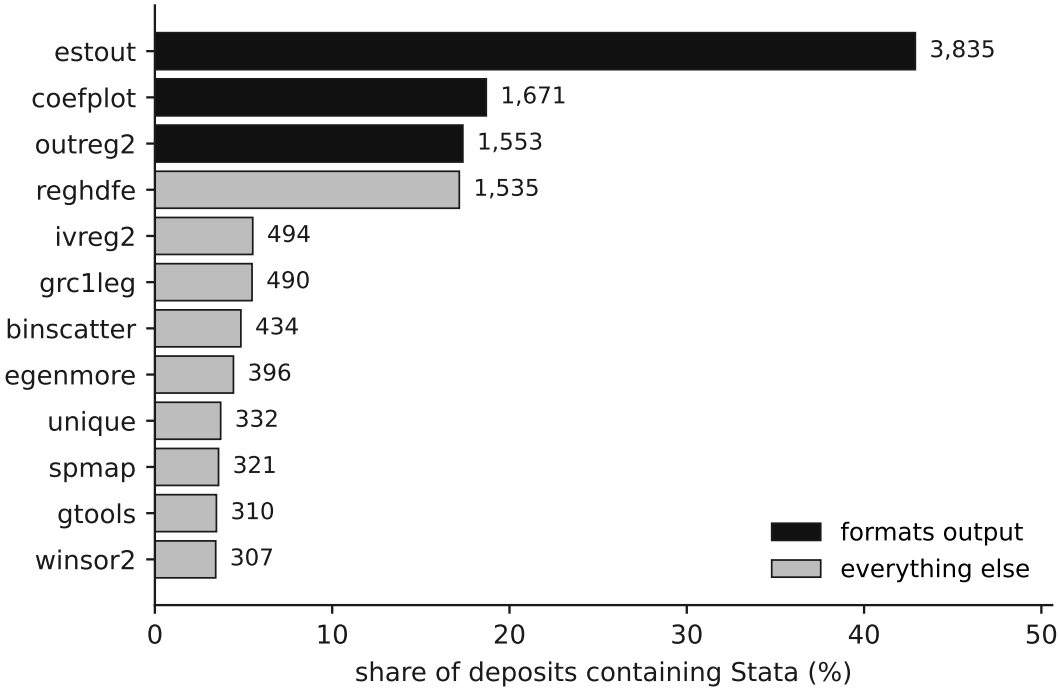

Figure 1: The twelve most used Stata packages, by share of deposits

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.

4.1 What published code loads

Of the ten most used Stata packages (Table 1), 3 prepare exhibits and estimate nothing: `estout`, `coefplot`, `outreg2`. Fig. 1 shows the same ordering with those packages marked. The most used of all is `estout`, in 43% of Stata deposits, ahead of every regression command.

Much of what research depends on, then, is the machinery that turns estimates into a publishable table or figure, and the field uses that machinery constantly and cites it almost never.

4.2 RQ1: Validated use and download counts

Table 2: Agreement between validated use and download counts, from both sides

registry	used	rho, used	in 20 or more	counted	ever used	rho, counted	top 100 used
Stata (SSC)	865	0.61	0.54	3,906	22%	0.47	85
R (CRAN)	2,300	0.53	0.31	24,719	9%	0.39	76
Python (PyPI)	563	0.36	0.26	15,000	3%	0.18	39

Note: *used* is packages the corpus uses that the registry has a download count for, and the two columns after it are Spearman rank correlations between deposits and downloads among them, and among those in at least twenty deposits. *counted* is every package the registry has a count for; *ever used* is the share of those in any deposit, the correlation after it counts an unused package as zero, and the last column is how many of the hundred most downloaded are used at all. SSC counts hits in one month, CRAN downloads over a year from one mirror, and PyPI thirty days for its fifteen thousand most downloaded projects, so its *counted* is that list and not PyPI. Every comparison is of ranks within one registry.

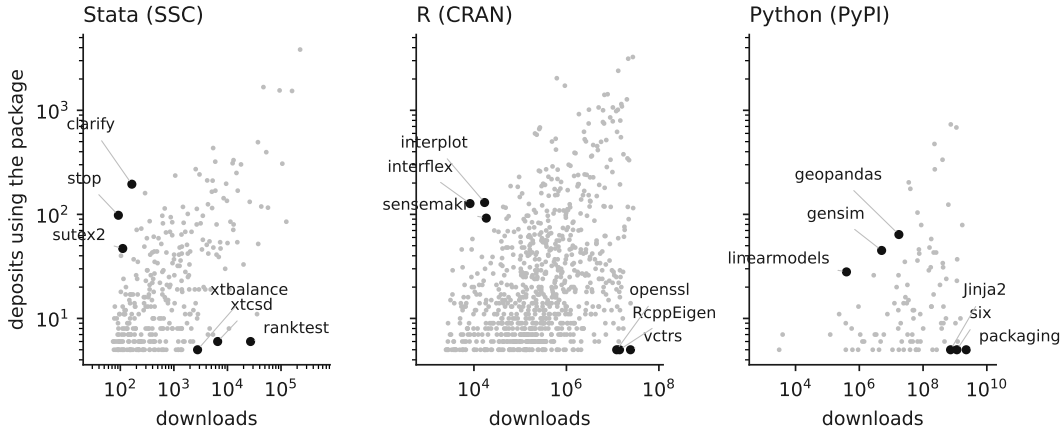

Figure 2: Deposits using a package against its downloads

Note: Packages used in at least five deposits, on logarithmic axes. Labeled are the packages ranked furthest above and furthest below their download rank, chosen by rank difference.

The answer depends on which packages you ask about, so Table 2 asks from both sides. Start from every package a registry counts. Downloads then say a good deal about *whether* this field uses a package. 9% of CRAN appears in any deposit; 52% of its most downloaded tenth does, against 0% of its least downloaded, and 76 of the hundred most downloaded packages appear

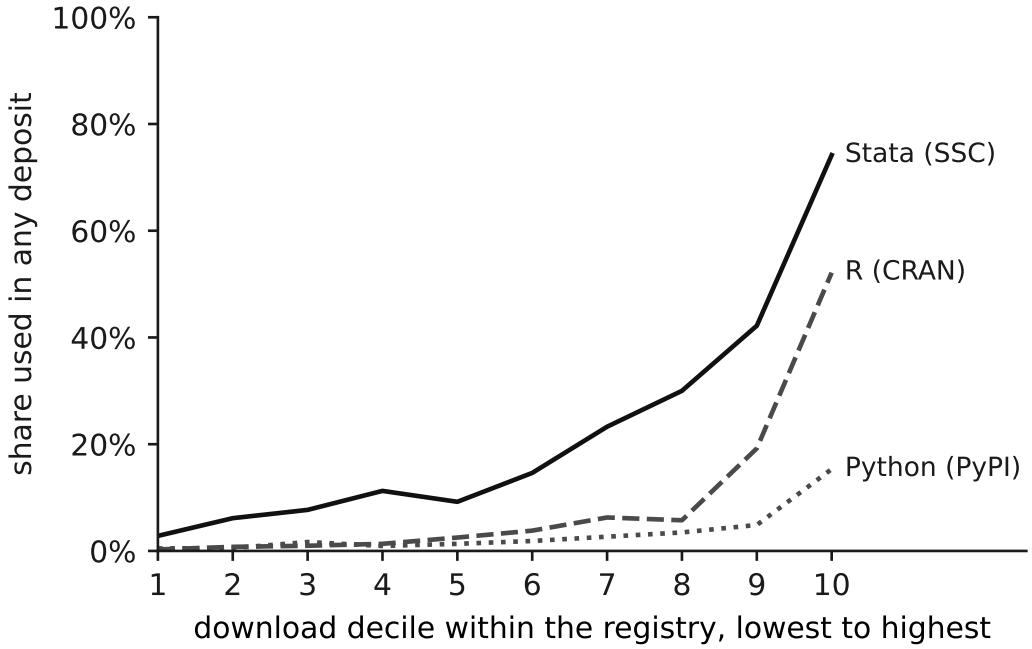

Figure 3: Share of a registry's packages used in any deposit, by download decile

Note: Packages the registry has a download count for, split into tenths by that count. For PyPI the tenths are of its fifteen thousand most downloaded projects.

somewhere (Fig. 3). Now start from the packages the field uses. Downloads say much less about *how much*. Among R packages in at least twenty deposits, the ones a committee would ask about, the rank correlation is 0.31. `stargazer` is 4th among R packages by validated use and 388th by downloads.

The two counts disagree for two reasons, and only one bears on this paper's argument. The first is that they count different people. Downloads pool everyone who uses R or Python; this corpus is economics and political science. `packaging`, `six`, `Jinja2` and `Pygments` rank far higher by downloads than by use here partly because most Python is not written by social scientists. Holding the audience fixed shows how much of the gap that explains. Among the 164 R packages the corpus uses that CRAN's *Econometrics* and *Causal Inference* task views list, which is software written for this field, the correlation rises from 0.53 to 0.65.

The second reason is that a download is cheap, and it shows in what downloads track. Across the R packages the corpus uses, downloads correlate at 0.87 with the number of other packages that install the package as a dependency; validated use correlates with that number at 0.45. Downloads come close to measuring a package's place in the dependency graph. The R packages ranked furthest below their download rank, `vctrs`, `RcppEigen`, `openssl` and `systemfonts` (Fig. 2), are all installed by other software, on every machine and build server that installs the other software. Given a package's reverse dependencies, downloads add a partial rank correlation of 0.31 with use.

One prediction failed. I expected agreement to rise once heavily depended-on packages were set aside, at two cut-offs fixed in advance. It fell, to 0.44 among packages with at most a hundred reverse dependencies and 0.34 with at most ten. The packages set aside are the head of both lists, `ggplot2` and `dplyr`, where the two counts agree, and among packages nothing depends on, downloads carry little signal of any kind. I computed the partial correlation above after seeing that result, so it describes and does not test.

Agreement also falls as automatic installation rises across registries. It is highest for Stata, 0.61, where `estout` leads both lists and a package is installed only when a person types `ssc install`. It is lower for R, whose packages declare dependencies that install with them, and lowest for Python, 0.36, where continuous integration reinstalls an environment on every commit. The costly-signal argument predicts this order. Three registries are three observations.

Concentration differs too. The hundred most downloaded CRAN packages take 50% of CRAN's downloads and 30% of the package uses in this corpus. For SSC the two shares are 76% and 71%, close to each other and well below the 93% share of package loads that Correia and Seay (2024) report for economics code, on a different corpus in an earlier month.

A download count exists only where a registry keeps one. Software the corpus finds in the *Stata Journal*, the *Bulletin*, an author's site or a GitHub repository has none. 214 of the Python packages the corpus uses fall below the 15,000 projects PyPI's counts cover, which tells us one thing about them: each has fewer downloads than every project on that list. Ranked as tied at the bottom, they move the Python correlation from 0.36 to 0.39.

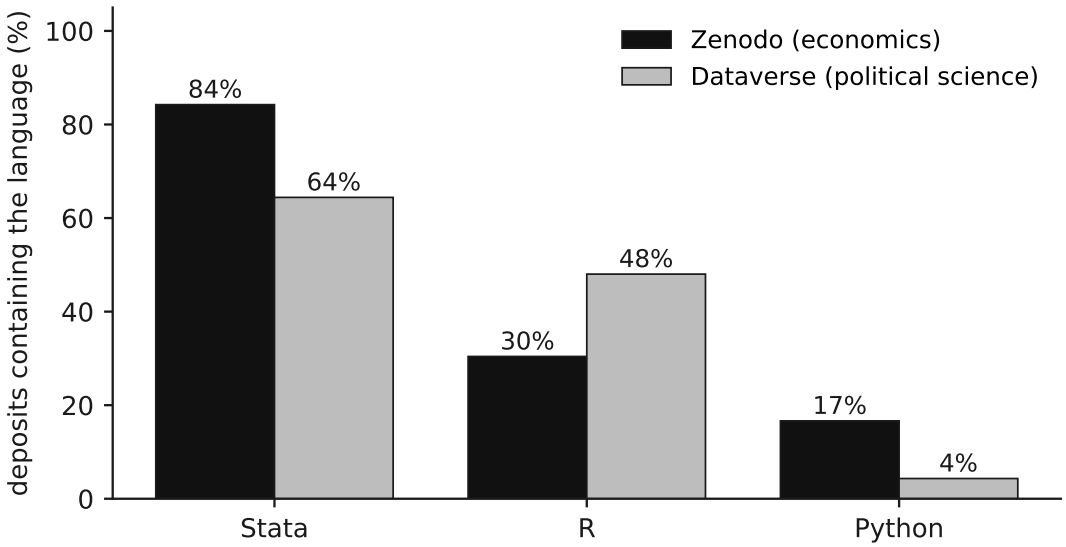

Figure 4: Languages by repository, and so by discipline

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.

4.3 RQ2: What a registry leaves out

8,954 deposits contain analyzable Stata against 6,194 with R and 773 with Python, roughly 1.4 Stata deposits for every R one. Fig. 4 shows that the split follows discipline: economics deposits are Stata, and political science deposits are mixed.

Studies of research software cover R and Python, the two languages with public package registries. That leaves out the language most of this code is written in, and it leaves it out because Stata is harder to count, so the discipline that uses Stata most drops from the record for a reason unrelated to how much software it uses. The next subsection measures what the registries miss.

4.4 Registries: what they cover and what they miss

A package can be counted only if the name in the code can be looked up somewhere. For R and Python one registry does almost all of that work. For Stata no single registry does, and Table 3 shows where the rest of its software lives.

Stata's software is scattered, not informal. SSC holds 4,092 packages. The *Stata Journal* and its predecessor, the *Stata Technical Bulletin*, publish the software behind their articles: 913 and 479 packages. Authors serve more from their own sites; `grc1leg`, which combines graphs under one legend, comes from a StataCorp developer's page and from nowhere else. All of these use the format `net install` reads, a `stata.toc` listing packages and a `.pkg` manifest listing each package's files, and none was indexed by command. I indexed all four. 1,296 deposits use at least one of the 176 packages that an index of SSC alone reports as belonging to nothing, among them `renvars`, published in the *Bulletin*.

Table 3: Where the software a deposit uses lives

language	deposits	primary registry	secondary archive	code host	author's site	unresolved
Stata	7,199	96.7%	8.3%	0.0%	11.1%	14.6%
R	6,311	99.8%	22.0%	2.2%	0.0%	9.3%
Python	872	97.7%	0.0%	0.0%	0.0%	18.0%

Note: Share of deposits using at least one package found in each place, over deposits with any use that could name a third-party package; a deposit can appear in several columns. The primary registry is SSC, CRAN or PyPI. Secondary archives are the *Stata Journal* and the *Stata Technical Bulletin* for Stata, and Bioconductor and the CRAN archive for R. A code host is credited only when the deposit itself installs the package from one. *Unresolved* is a deposit using at least one name that none of these accounts for.

Authors' sites have no list, so they are found from the code. Every `net install ..., from(URL)` line in the corpus names one, and each site named is then crawled whole from its own `stata.toc`, because `net install` is typed once by hand and most deposits that use a command carry no install line for it. Looking up the names still unplaced led to three more sites, one of them the new home of the SPost suite. Together that is 480 packages on 26 hosts, 20% of them on GitHub, and 27 addresses that no longer serve a manifest. A site no deposit installs from is not found. That bounds this tier by the corpus in a way the other three are not.

Registries lose software as well as lacking it. 22% of R deposits load a package that CRAN has since archived. A lookup against today's CRAN calls those unknown, and they are nearly all of R's secondary column.

Off-registry software is not peculiar to Stata. In R it lives on GitHub. 2.2% of R deposits load a package that CRAN and Bioconductor lack and that the deposit itself installs with `install_github` or a relative. A name is credited to a code host only when the same deposit installs it from one. Pooling the evidence would let one deposit's `install_github("someone/utils")` vouch for every unresolved `utils` in the corpus. Relaxing the rule to names installed from a code host anywhere in the corpus would account for 190 of the 587 R deposits with an unresolved name.

Table 4: Names no archive accounts for, labeled by hand

category	names	most used
formal archive not indexed	2	glmmadmb, qgis
code host	23	dcdensity, vdemdata, starpolishr
author site	21	btscs, ols_spatial_hac, grinter
commercial	3	arcpy, gurobi, arcgisscripting
official, undocumented	2	beep, _regress
local program	1	pstar
not software	25	the, this, jobid

Note: Every name used in at least 5 deposits that resolves to nothing, in all three languages, looked up one at a time. The labeled file is released with the data.

What is left is short enough to read. 77 names appear in at least 5 deposits and resolve to nothing. I looked each one up (Table 4). 49 are software. The other 28 are not: commands official Stata ships without documenting, programs a deposit defines for itself, and text the lexer read as code. Below that threshold sit 1,844 Stata names that each appear in one deposit: typos, macro fragments and one author's helpers.

Stata is hard to count because its software has no single registry. An index removes that difficulty.

4.5 RQ3: Field-specific software dependence

Table 5: Most loaded Stata packages in the twelve largest journal collections

collection	field	deposits	#1	#2	#3
restat	economics	1183	estout (486)	outreg2 (293)	reghdfe (263)
jop	political science	655	estout (266)	coefplot (136)	outreg2 (93)
ej-replication-repository	economics	485	estout (321)	reghdfe (184)	outreg2 (135)
isq	political science	464	estout (120)	coefplot (58)	outreg2 (49)
ajps	political science	431	estout (151)	coefplot (79)	outreg2 (69)
BJPolS	political science	393	estout (174)	coefplot (104)	outreg2 (41)
restud-replication	economics	379	estout (220)	reghdfe (138)	outreg2 (80)
the_review	political science	355	estout (170)	coefplot (91)	outreg2 (61)
PSRM	political science	352	estout (124)	coefplot (55)	outreg2 (52)
qje	economics	328	estout (191)	reghdfe (156)	binscatter (72)
polbehavior	political science	289	estout (76)	coefplot (65)	outreg2 (29)
researchandpolitics	political science	248	estout (84)	coefplot (53)	outreg2 (28)

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.

`estout` is the most loaded Stata package in 12 of the 12 largest collections (Table 5), in economics and political science alike.

Second place differs by field. Across the 35 collections with 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 23 of the 26 political science collections.

Repository could explain the split, since each half of the frame is nearly one discipline and a 2024 scrape differs from a 2026 one in more than subject. It does not. 5 of the economics collections deposit to Harvard Dataverse beside the political science journals, with the same repository, vintage and collection method, and the split appears among them too.

A single ranking over the whole corpus puts the exhibit tools on top and hides that the estimator beneath them differs by field.

5 Discussion and Recommendations

5.1 What better counts are for

These counts have two uses. The first is credit. A package in 43% of the Stata deposits at verified journals gives its author a number that anyone can look up in the released tables.

The second is choosing what to test. Work on software reliability has to pick packages to examine, and picking by reputation reproduces what the field already believes. The companion project `kasauti` probes every release of a package for results that changed without notice, and it picks its targets partly by how often a function appears in real code. Validated-use counts answer that question, and they point at the packages where a silent change would touch the most published work.

5.2 Limitations

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 unit cost is the cost of a paper, not of a package, so the measure resists the cheap manipulation that download counts invite and not a determined one.

The counts 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. That answers "how many pieces of research chose this" and not "how much research depends on this".

The corpus comes from two collections with different instruments behind them. The Harvard Dataverse material is a January 2024 scrape that kept only `.do`, `.r` and `.py` files; the Zenodo material is whole deposits, including notebooks, knitr documents and `.ado` files. A package used mainly inside a literate document is under-counted on the Dataverse side, and the two collections are two years apart. Recomputing every ranking on the file types both collections share shows what that costs. For Stata and R all twenty top places hold. For Python, `ipython` and `spacy` drop out of the top twenty and four other packages move, none above rank fourteen; Python is the language whose code most often lives in a notebook. The top ten in every language, the part the tables print, does not change.

Deposits whose archives exceed the size cap yield no code, and they are code-heavy: the median archive in the smallest quartile holds 14 code files and the median in the largest holds 46. 2 deposits split a package across `.z01`, `.7z.002` or `.partN.rar` segments, which are not reassembled; only the last segment carries the archive's directory, so the download completes and the first read fails.

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

Two false positives remain: SSC packages whose only command is a single common token attract stray matches from Mata-style assignments outside a `mata:` block. At present they are `fastcd` (command `c`) in 6 deposits, `wordy` (command `word`) in 1 deposits.

The lexer still reads some text as code. A do-file can hold prose an author never commented

out, log output pasted back into the script, and fragments of Mata or R outside any block the lexer recognizes, and their first words land in command position. 28 of the 77 names in Table 4 are of this kind, `the` and `this` among them, and Stata keywords account for 56 more mentions across 13 deposits. They inflate the unresolved rate and nothing else: a word of English cannot collide with a package name.

The index of authors' sites is bounded by the corpus. SSC and the two journals are crawled whole, but a site is found only if some deposit installs from it or if looking up an unplaced name led to it. Software served from a site no deposit names stays unresolved. Sites also die: 27 of the addresses deposits install from no longer serve a manifest, and for some commands, `btscs` among them, the only copies left are the ones authors shipped inside their deposits.

Julia files are collected and not parsed, so Julia is reported as language presence only, as are MATLAB, SAS and SPSS. MATLAB has no import statement and no registry to resolve against.

Which version ran is mostly not recorded. Code names the packages it loads and almost never the release, so these counts pool a package across its history. Where a file does say, the release records it: a Stata `version` statement in 1,194 of the 8,983 deposits holding a do-file, an `renv.lock` or a bundled `DESCRIPTION` in 183 of 6,628 deposits with R code, and a notebook's own record of the interpreter that ran it. At 13% coverage for the best of these, the environment table supports checking particular deposits and not estimating a distribution.

5.3 Recommendations

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, verified replication code loads, over 13,245 deposits and 4,565 packages.

The software social science leans on hardest, `estout` first, prepares exhibits, and a download count does not show that. The language most of this code is written in was left out of studies of research software for want of an index, and the index is now public. Registries miss something in every language: in Stata, two journals' archives and authors' sites; in R, GitHub.

A committee cannot reward what it cannot count (Sood 2019). Evaluation committees and the designers of credit systems should use validated use where they now use downloads or citations to software papers. Moving it by one costs a published, verified paper, a build server cannot inflate it, and every package's count is published.

6 Data and code availability

The Stata command-to-package index is deposited at 10.5281/zenodo.21926099 under CC0, with 10,147 commands across 5,581 packages, as a record of its own.

The mention-level record and the aggregates are deposited at 10.5281/zenodo.21943908 under CC0, covering 4,565 packages, with counts by year and by journal. Each mention carries its package, the function where the source names one, the file, the line and a snippet, so the counts can be recomputed without re-parsing the corpus. The deposited code itself, 447,289 files, stays at the archives it came from. The tables are also served at https://recite.github.io/softver

se/data/, and each package can be looked up at https://recite.github.io/softverse/lookup/. Both DOIs resolve to the latest version of their record.

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

7 Reproducibility

The corpus, the registry snapshots, the download counts and the extractor are pinned, and the first appendix lists each pin. Every mention carries the extractor version and the registry lock that produced it.

8 Acknowledgements

Daniel Weitzel contributed to an earlier version of this project.

9 Competing interests

The author has no competing financial interests. The argument that better counts would increase the production of research software is the author's own (Sood 2019), and the paper releases the index it says the field lacks.

10 Funding

This research received no external funding.

11 Author contributions

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

12 References

AEA Data Editor, and Lydia Reiner. 2021. *Identifying Stata Packages Used in Replication Packages*. https://aeadataeditor.github.io/projects/project4.

Arel-Bundock, Vincent. 2022. *Which R Packages Do Scientists Use?* https://arelbundock.com/posts/dataverse_r/.

Besançon, Lonni, Guillaume Cabanac, Cyril Labbé, and Alexander Magazinov. 2024. "Sneaked References: Fabricated Reference Metadata Distort Citation Counts." *Journal of the Association for Information Science and Technology* 75 (12): 1368–79. https://doi.org/10.1002/asi.24896.

Brown, Eva Maxfield, Isaac Slaughter, and Nicholas Weber. 2026. "Code Contribution and Credit in Science." *Quantitative Science Studies* 7: 543–63. https://doi.org/10.1162/qss.a.465.

Carlin, Domhnall, Austen Rainer, and David Wilson. 2023. "Where Is All the Research Software? An Analysis of Software in UK Academic Repositories." *PeerJ Computer Science* 9: e1546. https://doi.org/10.7717/peerj-cs.1546.

Choodari-Oskooei, Babak, and Tim P. Morris. 2016. "Quantifying the Uptake of User-Written Commands over Time." *The Stata Journal* 16 (1): 88–95. https://doi.org/10.1177/1536867X1601600110.

Correia, Sergio, and Matthew P. Seay. 2024. "Require: Package Dependencies for Reproducible Research." *The Stata Journal* 24 (4): 599–613. https://doi.org/10.1177/1536867X241297915.

Csárdi, Gábor. 2015. *Cranlogs: Download Logs from the 'RStudio' 'CRAN' Mirror*. The R Foundation. https://doi.org/10.32614/CRAN.package.cranlogs.

Du, Caifan, Johanna Cohoon, Patrice Lopez, and James Howison. 2021. "Softcite Dataset: A Dataset of Software Mentions in Biomedical and Economic Research Publications." *Journal of the Association for Information Science and Technology* 72 (7): 870–84. https://doi.org/10.1002/asi.24454.

Du, Caifan, Johanna Cohoon, Patrice Lopez, and James Howison. 2022. "Understanding Progress in Software Citation: A Study of Software Citation in the CORD-19 Corpus." *PeerJ Computer Science* 8: e1022. https://doi.org/10.7717/peerj-cs.1022.

Hannay, Jo Erskine, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. 2009. "How Do Scientists Develop and Use Scientific Software?" *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, 1–8. https://doi.org/10.1109/secse.2009.5069155.

Herbert, Sylvérie, Hautahi Kingi, Flavio Stanchi, and Lars Vilhuber. 2024. "Reproduce to Validate: A Comprehensive Study on the Reproducibility of Economics Research." *Canadian Journal of Economics/Revue Canadienne d'économique* 57 (3): 961–88. https://doi.org/10.1111/caje.12728.

Howison, James, and Julia Bullard. 2016. "Software in the Scientific Literature: Problems with Seeing, Finding, and Using Software Mentioned in the Biology Literature." *Journal of the Association for Information Science and Technology* 67 (9): 2137–55. https://doi.org/10.1002/asi.23538.

Howison, James, Ewa Deelman, Michael J. McLennan, Rafael Ferreira da Silva, and James D. Herbsleb. 2015. "Understanding the Scientific Software Ecosystem and Its Impact: Current and Future Measures." *Research Evaluation* 24 (4): 454–70. https://doi.org/10.1093/rese

val/rvv014.

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.

Istrate, Ana-Maria, Donghui Li, Dario Taraborelli, Michaela Torkar, Boris Veytsman, and Ivana Williams. 2022. *A Large Dataset of Software Mentions in the Biomedical Literature*. https://doi.org/10.48550/arXiv.2209.00693.

Kemenade, Hugo van, Cal Paterson, Agriya Khetarpal, et al. 2026. *Hugovk/Top-Pypi-Packages: Release 2026.09*. Zenodo. https://doi.org/10.5281/zenodo.22225583.

Kranz, Sebastian. 2023. *Usage Shares of Programming Languages in Economics Research*. https://www.r-bloggers.com/2023/12/usage-shares-of-programming-languages-in-economics-research/.

Lemke, Steffen, Athanasios Mazarakis, and Isabella Peters. 2021. "Conjoint Analysis of Researchers' Hidden Preferences for Bibliometrics, Altmetrics, and Usage Metrics." *Journal of the Association for Information Science and Technology* 72 (6): 777–92. https://doi.org/10.1002/asi.24445.

Li, Kai, Pei-Ying Chen, and Erjia Yan. 2019. "Challenges of Measuring Software Impact Through Citations: An Examination of the Lme4 r Package." *Journal of Informetrics* 13 (1): 449–61. https://doi.org/10.1016/j.joi.2019.02.007.

Park, Hyoungjoo, and Dietmar Wolfram. 2019. "Research Software Citation in the Data Citation Index: Current Practices and Implications for Research Software Sharing and Reuse." *Journal of Informetrics* 13 (2): 574–82. https://doi.org/10.1016/j.joi.2019.03.005.

Python Package Index. 2026. *Metadata Requests No Longer Tracked in PyPI Download Counts*. https://blog.pypi.org/posts/2026-08-31-download-counts/.

Reiner, Lydia, and Lars Vilhuber. 2021. *Statapackagesearch: Search for (Missing) Packages in Stata Code*. Software repository, first commit April 2021. https://github.com/labordynamicsinstitute/Statapackagesearch.

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.

Schindler, David, Erjia Yan, Sascha Spors, and Frank Krüger. 2023. "Retracted Articles Use Less Free and Open-Source Software and Cite It Worse." *Quantitative Science Studies* 4 (4): 820–38. https://doi.org/10.1162/qss_a_00275.

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. 2019. *Countpy: Incentivizing More and Better Software*. https://www.gojiberries.io/countpy-incentivizing-more-and-better-software/.

Sood, Gaurav. 2023. *Softverse: Auto-Compute Citations to Software from Replication Files*. Software repository, first commit 12 July 2023. https://github.com/recite/softverse.

Sood, Gaurav. 2025. *User: Count How Often a Python Package Has Been Included on Public Repos. In GitHub*. Software repository, first commit 12 March 2025. https://github.com/recite/user.

Sood, Gaurav, and Khanh Tran. 2018. *Countpy: Count the Total Number of Times a Python Package Has Been Imported in Public Repos. On Github*. Software repository, first commit 4 August 2018. https://github.com/recite/countpy.

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.

Upton, Elizabeth, Zhuoer Cai, Pamela Jakiela, Owen Ozier, and Sruthi Raman. 2026. "The Software Behind the Stats: A Student Exploration of Software Trends in Economics, Political Science, and Statistics." *Journal of Statistics and Data Science Education*, ahead of print. https://doi.org/10.1080/26939169.2026.2725967.

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

Wang, Yuzhuo, and Kai Li. 2024. "How Do Official Software Citation Formats Evolve over Time? A Longitudinal Analysis of r Programming Language Packages." *Scientometrics* 129 (7): 3997–4019. https://doi.org/10.1007/s11192-024-05064-6.

Appendix: Sampling frame, coverage and provenance

Appendix: Validation of static parsing

12.1 Coverage

Resolution measures coverage, not precision: a deposit's own module can share a name with a real distribution and still resolve. 1.8% of the mentions that could name a third-party package

Table 6: 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.

Table 7: Why a deposit yielded no code

reason	deposits
deposit contains no code files	38
download or extraction failed	12
archive over the size cap	7
multi-part archive, not reassembled	2

Note: Zenodo deposits that produced no analyzable file, by reason. Deposits over the size cap and multi-part archives could be recovered; the single failure is a zip using a compression method Python does not implement.

Table 8: How each language is read

language	mechanism
R	tree-sitter syntax walk
Python	stdlib ast, tokenize fallback
Stata	purpose-built lexer, 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.

Table 9: Pinned registry snapshots

registry	snapshot	sha256
bioconductor	2026-08-11	3ffed30246d706ae
cran	2026-08-11	aefeb12f742cad24
cran_archive	2026-08-11	5e267904e40ed060
cran_dependencies	2026-09-17	a7d61d884dd42c9b
cran_task_views	2026-09-17	5f66bd8ecf1cc547
downloads_cran	2026-09-17	e008acfbd4f85bc9
downloads_pypi	2026-09-17	922a1717f7700c51
downloads_ssc	2026-09-17	268c874044f39650
julia_general	2026-08-11	9cf5a9b5f71da392
pypi	2026-08-11	751e02220c921dea
pypi_import_map	2026-09-17	bc02385c473b7ccf
stata_base_ado	2026-09-17	1af4034964152f60
stata_index	2026-09-17	5414adfa0ad29683
stata_official	2026-09-17	805974d631db724e

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. `stata_index` is the command-to-package index, `stata_official` the list of official Stata commands checked against StataCorp's help server, `stata_base_ado` the file names in Stata's base directory, and `pypi_import_map` the table from Python import names to distributions.

Table 10: Mentions that could name a package, and those left unresolved

language	candidate mentions	unresolved	rate
stata	884968	19793	2.2%
r	347973	2847	0.8%
python	306806	4572	1.5%

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.

resolve to no registry, and the released tables list every such name.

The denominator matters. Against all mentions the same rate is 0.2%, because that denominator includes 13,034,612 calls to official Stata commands that could never be unresolved. The rate above, over mentions that could name a package, is 10 times larger.

Per deposit, the unresolved share is far below the pooled rate (Table 11).

Table 11: Unresolved share, pooled and per deposit

statistic	unresolved share
pooled over mentions	1.8%
median deposit	0.0%
90th-percentile deposit	2.2%
deposits fully resolved	86%

Note: Restricted to mentions that could name a third-party package. A minority of deposits hold most of the unresolved mentions.

Every candidate mention resolves in 86% of deposits.

Stata holds 73% of all unresolved mentions, spread over 2,131 names, of which 1,844 appear in a single deposit. The most widespread, `dcdensity`, is in 65 deposits. Table 4 accounts for every name above the labeling threshold; below it are lexer errors and authors' own programs.

Two kinds of name are neither resolved nor unresolved. 21,345 are Python imports of a module in the deposit itself, relative (`from .models import Constants`) or absolute when a file of that name sits in the deposit. 1,050 are names built at run time, as in `library(pkg, character.only = TRUE)`, where the package is a variable. Both are reported in their own categories, since `models`, `results` and `log` are PyPI distributions as well as names people give their own files.

12.2 Official Stata commands

An omission from the list of official Stata commands does damage: a command Stata provides that the list lacks falls through to the SSC index, and if a package ships a file of that name, the use is credited to the package. Every command name in the corpus was therefore checked against StataCorp's public help server. The server answers only for commands with a help page, and Stata ships utilities without one, `_dots` and `tset` among them, so the list also takes every `.ado` file name in the base directory of a Stata installation, read from a public container build: 3,346 names.

The check moved 428 commands, including `ds`, `pca` and `testparm`, from unresolved to official, 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 corpus is 23.8% against 1.8%, and 3,389 mentions would have gone to an SSC package that ships a file of the same name. The help server returns HTTP 200 for every query, so existence is read from the page text, and pages from the User's Guide and the function manual are excluded, since they document `_n`, `_b` and `e()`, which are not commands.

12.3 Agreement with `renv`

`renv::dependencies()` (1.1.8), run over 6,209 of the corpus's 6,194 R deposits, gives a mean per-deposit Jaccard of 0.99 and a pooled Jaccard of 0.98, and 93% of deposits agree exactly. The mean weights every deposit equally and the pooled figure every package-deposit pair. 72 deposits are excluded because renv returned nothing for them: it runs with `errors = "ignored"` and skips a file it cannot decode.

The comparison sets aside three kinds of name that renv reports and this extractor leaves out: 1,879 base R packages such as `grid` and `stats`; 546 cases of `rmarkdown`, which renv adds for any `.Rmd` file; and 4,454 names found only in files outside the analysis set, such as a package library shipped inside a deposit. Without setting these aside, the mean Jaccard is 0.95 and the pooled Jaccard 0.91. After it, renv names 206 packages this extractor does not, and this extractor names 1,039 that renv does not.

This is agreement between two static scans of the same files, and it is weaker than accuracy: a package named only in a string, or loaded through a variable, is invisible to both.

12.4 Agreement with R's parser

Whether a file contains a call to `library` with argument `x` has an exact answer, given by R's own parser. Running R's parser over 112,092 `.R` files from both repositories and walking its expression trees gives precision 1.0000 and recall 1.0000 over 257,817 name-file pairs, 85,308 of the files from Dataverse. 2,161 files that R could not parse are excluded. The oracle's own walk had to handle a one-element result, a `character.only = T` written with the symbol `T`, calls such as `base::library(c)`, and `pkg::fun()` used as a default argument; the one disagreement that was the extractor's error, `"m"::baz()` with a string on the left, is fixed.

The extractor also runs, as a test, against `renv`'s own test fixture, which lists thirteen ways of loading a package and states which must be found.

This check covers one thing exactly: whether the syntax walk sees what R sees. It says nothing about Stata, about resolution against CRAN, or about whether a `library()` call means the analysis used the package. No sample has been labeled by hand for use, so there is no precision or recall against research use.

12.5 Code that nothing runs

A deposit can hold scripts nothing runs: an old draft, a helper that was replaced, a figure cut in review. Where a deposit has a master script, the files it reaches can be told from the files it does not, and the counts can be recomputed over the reachable ones.

A master is a script that runs other scripts and is run by none, through `source()`, `do`, `run`, `include` or a local import. 2,661 of 13,180 deposits have one, and in 1,965 of those every edge could be followed. The rest mostly build a path from a macro, as in `do "$root/clean.do"`, which names a file only at run time; 9,835 edges are written that way, against 50,515 that could be followed. Counting a macro edge as unreachable would report an idiom as dead code, so those deposits are left out.

In the deposits with a complete graph, 69% of analyzable files are reachable from the master,

and the median deposit has 86% of its files on that path. Recomputed over reachable files alone, the twenty most used packages keep a median 80% of their Stata deposits, 77% of their R deposits and 71% of their Python deposits. The package hit hardest keeps 40%, and no package moves more than 4 places. The counts shrink and the order holds.

Table 12: Counts over every analyzable file, and over files a master script reaches

Language	Package	Deposits	Reached by a master	Kept	Rank
R	dplyr	684	534	78%	1 to 1
R	ggplot2	641	470	73%	2 to 2
R	tidyverse	486	391	80%	3 to 3
R	xtable	380	301	79%	4 to 4
R	stargazer	363	291	80%	5 to 5
Stata	estout	512	405	79%	1 to 1
Stata	reghdfe	247	198	80%	2 to 2
Stata	outreg2	185	139	75%	3 to 3
Stata	coefplot	159	134	84%	4 to 4
Stata	ivreg2	96	68	71%	5 to 5
Python	pandas	231	163	71%	1 to 2
Python	numpy	222	179	81%	2 to 1
Python	matplotlib	147	110	75%	3 to 4
Python	scipy	142	121	85%	4 to 3
Python	scikit-learn	101	76	75%	5 to 5

Note: The five most loaded packages per language. Restricted to deposits with a master script and a graph containing no edge that could not be followed. *Kept* is the share of the deposits that survive the restriction; *Rank* gives the package's position before and after it.

This check has two limits. It says nothing about the 10,519 deposits with no master script, which are most of them. And a file a master reaches can still load a package inside a branch that never runs, which no static reader can see.