Social Proof is in the Pudding: The (Non)-Impact of Social Proof on Software Downloads[*]

Lucas Shen[†] Gaurav Sood[‡]

August 19, 2024

Abstract

Open-source software is widely used in commercial applications. Pair that with the fact that when choosing open-source software for a new problem, developers often use social proof as a cue. These two facts raise concern that bad actors can game social proof metrics to induce the use of malign software. We study the question using two field experiments. On the largest developer platform, GitHub, we buy 'stars' for a random set of GitHub repositories of new Python packages and estimate their impact on package downloads. We find no discernible impact. In another field experiment, we manipulate the number of human downloads for Python packages. Again, we find little effect.

Keywords: Field Experiment; RCT; GitHub; Security; Malware; Open-Source Software; Social Proof

[*]You can download the replication materials from https://github.com/soodoku/social_proof_stars/.

[†]Lucas is a Senior Scientist at the Institute for Human Development and Potential, Agency for Science, Technology and Research, and can be reached at lucas@lucasshen.com

[‡]Gaurav can be reached at gsood07@gmail.com

1 Introduction

Most commercial software relies on open-source software. But vetting the quality of software with respect to security is hard. Hence, most software developers use cheap heuristics like social proof to choose between multiple open-source software that all purport to solve the same problem. This raises the concern that bad actors can game social proof metrics to induce developers to use malign software. In this paper, we explore this possibility using two field experiments.

Our first field experiment was conducted on GitHub, the world's largest developer platform. On GitHub, we manipulated social proof of a random set of GitHub repositories associated with new Python packages by buying 'stars' for them. For a random subset of the treated repositories, we further increase the 'stars' by getting people in our network to 'star' the repositories. In all, our manipulation raises the median number of stars from 0 to 20–65 stars, depending on the treatment group. Such a dramatic increase in the number of stars, however, has little impact on the number of package downloads. Three months after the treatment, we cannot reject the null that the manipulation had no effect.

In a second field experiment, we manipulate social proof of Python packages by manipulating the download count in the official downloads registry. In particular, we use a script to download a random set of Python packages multiple times. In all, the median number of downloads more than triples from about 20 to 100 because of the treatment. But there is little impact of the treatment on the number of downloads as many as three months later.[1]

Our study allays but doesn't dispel some of the concerns about the consequences of

[1] In the SI, we check if bot and human downloads Granger cause human downloads. Data suggest human downloads cause future human downloads. But given it is inconsistent with the experimental results, we are wary of reading too much into it.

inducing malware by gaming social proof. For one, a meaningful treatment effect may be malign software being used by 1–2 organizations. And our experiments are underpowered to detect such subtle effects. (In fact, it is nearly impossible to design experiments to detect such effects.) For two, it is entirely plausible that a more intense treatment would do the trick.

2 Social Proof

Others' choices, especially those of similar others or those we admire, can be informative (Cialdini, 2003; Messing and Westwood, 2014; Rao, Greve and Davis, 2001; Salganik, Dodds and Watts, 2006; Venema et al., 2020; Amblee and Bui, 2011). They can tell us what is useful or desirable. For instance, if someone is in the market for a car, they may pay attention to the cars in the office parking lot. Companies understand the value of social proof and regularly use it to try to influence customers. For instance, many companies prominently show lists of customers who have bought their products on their websites. News media companies and retailers like Amazon use social proof to aid consumer choice. For instance, Amazon, for many product categories, allows customers to sort by best sellers. It also shows how many customers bought a product from a particular seller over the last month or year. News media companies show lists like 'most read articles' to aid consumers.

Social proof can be especially influential when people need to make a quick decision or when acquiring information about the quality of a product is hard. Software is one such case. It is hard for most developers to establish how safe software is quickly. Inspecting the code takes time and skill. Hence, when choosing between multiple software programs that purport to provide the same functionality, people often rely on social proof. To shed further light on how manipulable developer's choices are, we study whether certain kinds of social proof affect the decision to download software.

In the first field experiment, we manipulate the number of "stars" on the world's largest software development platform, GitHub, and assess its impact on the number of downloads of the associate package. In the second field experiment, we manipulate the number of downloads of Python packages to assess its impact on future downloads.

3 GitHub Experiment

GitHub is the most popular platform for creating, storing, managing, and sharing code. Over 100 million developers use GitHub. The platform hosts over 30M public repositories. (A repository can be thought of as a folder. It generally contains code for a single project.)

GitHub provides various ways for a user to interact with a repository. Users can fork or clone (copy) a repository, open an issue (to raise a concern about the code), and *star* a repository. But users can't use social proof metrics common to other social media, e.g., how often a repository is visited, etc., to judge which repository to interact with. GitHub, however, prominently shows four other social proof signals: how often a repository has been forked, the number of open issues that a repository has, the number of users watching a repository, and the number of users who have starred it. The first two actions are uncommon and hence irrelevant for most repositories. 'Watching' a repository is more common, but it is still fairly infrequent because it is an expensive thing to do; 'watching' a repository means that you are alerted about all changes in the repository. Stars, an analog of "likes," are the most widely used form of social proof on GitHub.

Users star repositories for various reasons. First, some use it as a convenient bookmarking tool. Users can browse and search the repositories they have starred at any point. (Recently, GitHub introduced a way to organize the repositories you 'star,' in line with the curatorial purpose of 'stars.') Second, GitHub customizes a user's news feed based on the stars, e.g., recommending other repositories, showing news about consequential changes to the repository, etc. Third, some users 'star' a repository to disseminate information about useful repositories to their followers. All the users who follow a user (and have enabled such messages to be shown) see the repositories 'starred' by the users they follow in their news feed. Lastly, users 'star' a repository to show support for a project or a user or to add to the social proof.

Repository owners try hard to get people to 'star' their repositories to increase visibility (and hence usage). As we noted above, one of the ways starring a repository increases its visibility is by exposing followers of those who star a repository to it. The second way it helps increase visibility is by increasing the odds of the repository becoming a trending repository.[2] These trending repositories get additional attention, including from the media. The third potential path of impact is via greater social proof. A person looking for software may end up looking at the repository and may be persuaded by the number of 'stars' to download the software.

Overall, the number of stars a repository is widely considered the primary signal of its popularity. For this reason, it is the metric we intervene on. Our experiment's effects can be thought to come primarily from the path of greater social proof for two reasons. First, our manipulation is small enough to not make a repository trend. Second, the repositories on which we manipulate 'stars' are generated in a way so as not to map to the interests of the users 'starring' the repository which means stars have limited informational value for the followers.

[2]While GitHub now uses a combination of factors to determine trending repositories, stars are generally suspected as a key factor. See for example https://github.com/orgs/community/discussions/3083.

3.1 Sample and Randomization

Our population of interest is repositories associated with new Python packages. We focus on new Python packages because we conjecture that quality is the least certain when a repository is new. Hence, for a new repository, social proof provides the strongest signal. Our choice of studying repositories associated with Python packages stems from the fact that we can get reliable data on the number of downloads for a Python package. Our sample includes new packages listed between 24 and 30 April 2023. (We identified new PyPI packages by taking a set difference of the PyPI index on April 30 and April 24.) Of these packages, we only keep Python packages with a public GitHub repository. (We used the GitHub source URL from the package's setup configuration file to link the package to a GitHub repository.) In all, we identified 622 new packages with a public GitHub repository. Of the 622 public repositories, we assigned 100 to the treatment group.

Table SI 1.1 (column (1)) reports (pre-treatment) descriptive statistics of the packages. Most new packages were created in 2022 but were released in 2023. 96% of the packages had an open issue on GitHub. On average, a repository had been forked 20 times, and the user under whom the repository was listed had, on average, 4 subscribers (Section 3.4).

3.2 Treatment Conditions

Our treatment includes stars from two sources: market-bought and network-based clicks. We asked users in our network to 'star' the 100 repositories (Table SI 1.8). This group serves as our 'low dosage' treatment group. The median number of followers of users in our network who starred the treatment repositories was 9 (the mean was 64, Table SI 1.7). We expect the benefits of these stars to also come primarily from greater social proof because the repositories that the users starred did not focus on the interests and specializations of the users. We triggered the friend requests to 'star' the repositories on May 12, 2023. Our friends took about 10 days to 'star' all the 25 repositories (see Figure 1).

Of the 100 packages, we randomly selected 25 packages and bought 50 stars for each from Baddhi Shop on May 12, 2023. The stars we bought from the vendor were assigned over a few days, plausibly to avoid triggering GitHub's anomaly detection algorithms. One interesting feature of the stars bought on the market is that they were all from users whose accounts were created around April 20, 2023. We expect some of these stars to be taken down by GitHub integrity teams, so these stars likely only have a short-run impact. For the stars we bought, we have no reason to expect the users who starred the repository or their followers to be authentic. So, we only expect the benefits of these stars to accrue from people who look at the GitHub repository before downloading a package.

3.3 Attrition and Analytic Strategy

About one week after the start of the intervention, on May 20, 2023, we took a snapshot of the GitHub repository. By that time, we could only retrieve 582 repositories (Section 3.4). Thirty-seven repositories in the control group and three from the treatment group were lost because the GitHub repository was deleted or moved to private status. It is plausible that the somewhat lower attrition in the treatment group is because the repository owners were buoyed by seeing more stars.

Our primary estimand is Intent-To-Treat (ITT) effects on Python package downloads. We use the entire dataset of 622 Python packages for that. In the SI, we also estimate the Local Average Treatment Effect (LATE) that subsets on the compliers: Github repositories for which we see a net increase of at least 20 stars during the treatment period. (Note that some of the stars in the treatment period could be organic.)

3.4 Balance Tests

To confirm that the randomization was done correctly, we compare the means of various attributes across the treatment and control groups. Looking at the primary outcome variable, downloads, in the pre-treatment period, we find little difference between treatment and control packages (see Table SI 1.3).

We supplemented this check with other balance tests that rely on GitHub data. We did not take a snapshot of Github attributes before applying treatment. We only took a snapshot of GitHub user characteristics linked to the PyPI packages on May 20 (Section 3.3). So our balance tests that rely on GitHub data use post-treatment data, though one right at the end of the treatment period. As we noted above, there was attrition between the start and end of treatment.

Table SI 1.1 reports statistical summaries of various attributes of the Python packages associated with treated and control repositories. Column (2) and column (3) report the means and standard deviations (in parentheses) for packages in the control and treatment groups, respectively. The last column reports the standardized mean difference. We compare (a) repository size (in megabytes), (b) whether the repository is forked from a previously existing repository, (c) the year of repository creation, (d) the number of subscribers, (e) whether issues exist (or is enabled), (f) the number of forks into other repositories, (g) the number of open issues, (h) the number of topics listed,[3] and (i) whether the primary detected language is Python (not all Python packages have Python as the primary source code language). As the table shows, except for two characteristics (the number of subscribers and the number of forks), the treatment and control are statistically indistinguishable.

Tables SI 1.2 to SI 1.4 repeat the balance tests, making an additional split by dosage,

[3]Examples of topics from TensorFlow include "machine-learning," "deep-neural-networks," "deep-learning," "neural-network," and "distributed."

reaching similar conclusions.[4]

Another way to look at balance is in terms of user characteristics of repository owners. There are 545 users behind the 582 repositories. Eight users appear in more than one group. Table SI 1.5 reports whether users releasing treatment and control packages differ in how they decorate their user profile by listing their company, email, personal webpage, and a brief biography of themselves. Table SI 1.5 reports the difference in characteristics between these developers of the treated and control packages.) In each instance, we detect no difference between publishers of treatment and control.

Lastly, on May 20, we also took a snapshot of the PyPI Python package metadata: the number of dependencies and package description length. The former was captured through setup configuration files and the latter from the "readme" documentation or equivalent. While these attributes were collected post-intervention, these constitute balance tests to the extent that these characteristics are slow-moving. Tables SI 1.1 to SI 1.3 report differences between treated and control groups.

3.5 Manipulation Check

While historical download logs are immutable, users can rescind stars. Anticipating that bought stars might be flagged by GitHub and removed shortly after adding, we took a snapshot of the stars of each package at the end of every day. This allows us to more

[4]High-dosage repositories and high-dosage users are different from the control group on two aspects: high-dosage users have more gists (blogs/short code snippets) and are likelier to list the company they work for at the 5 percent level (Table SI 1.6). Given that our groups are randomly assigned, we ascribe these two statistically significant but substantively small differences to chance.

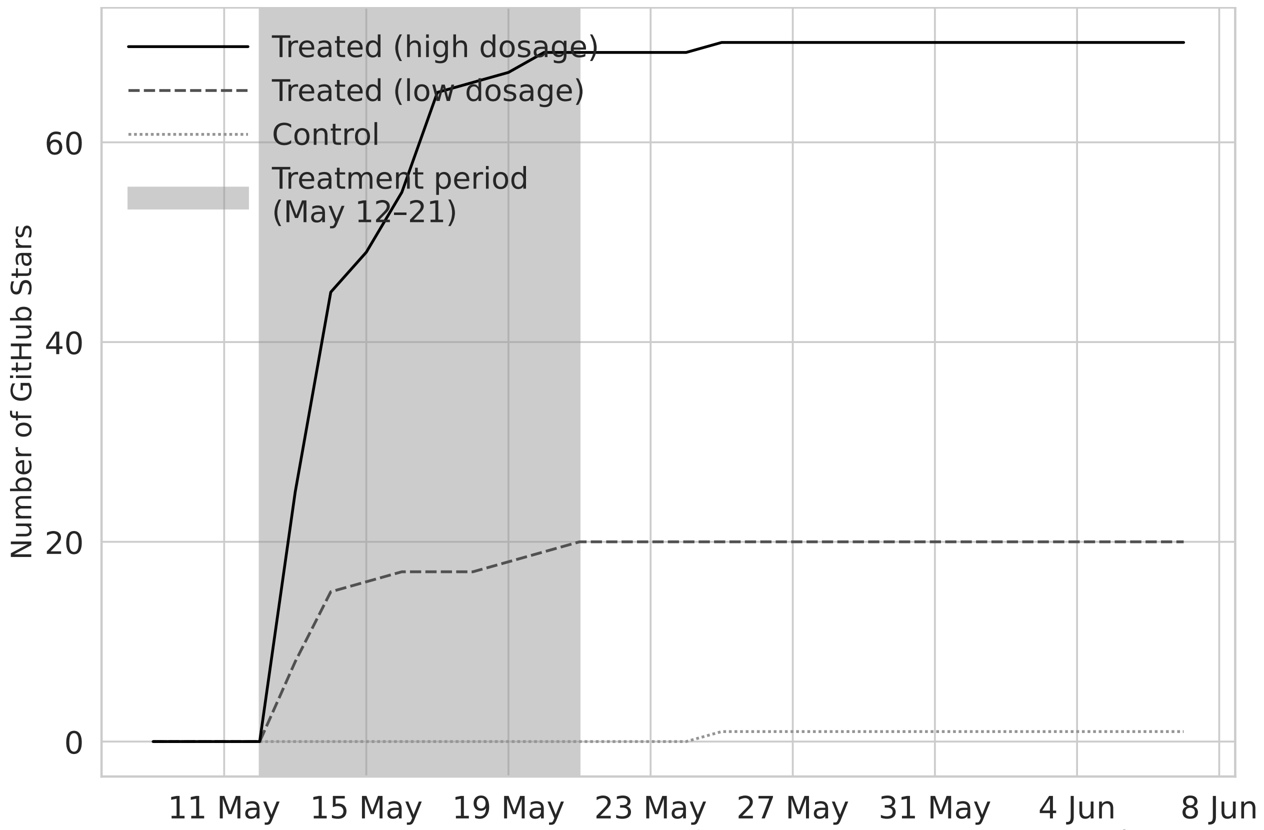


Figure 1. Manipulation Check: GitHub Stars for Treated vs. Control. The figure plots the median number of stars on a particular day for the three groups: high-dosage treatment (market and network stars), low-dosage treatment (network stars), and control (no stars). In all, we plot data for 585 packages and 17,550 package days. The shaded vertical bar indicates the period during which the treatment was applied. Table SI 1.9 presents formal estimates of this manipulation check. See Figure SI 1.1 for the figure that shows the means.

accurately capture the change in stars during our experiment period.[5]

Figure 1 shows the time trend of the median number of stars across the three groups: high-dosage treatment (market and network stars; $n = 25$), low-dosage treatment (market stars; $n = 75$), and control (no stars; $n = 485$). By the end of the intervention (May 21), relative to the control group, the median number of additional stars in the low-dosage group was 19 ($p < .001$), while the median number of additional stars in the high-dosage group was 69 ($p < .001$, Table SI 1.9).

[5]Historical stars with timestamps are available from the API but do not retain records of removed stars.

3.6 Outcome Measure: PyPI Downloads

The primary outcome for both the field experiments, the GitHub experiment, and the PyPI experiment (Section 4) is Python package downloads. These Python package download metrics come from the centralized Python Package Index (PyPI) repository. It hosts open-source Python packages uploaded to it by package developers. Users download packages as needed from PyPI.

To log package downloads from PyPI, the Linehaul project implements a daemon that listens for download events and logs them. Specifically, it tracks details like the package name, date and time the package was downloaded, package version, and the type of installer software. Linehaul then feeds the download logs to the publicly available Google BigQuery. Some installers are known bots (Table SI 1.13). These bots tend to be caching mirrors for distributional and security purposes. For instance, Bandersnatch is the official mirroring service of PyPI to help improve accessibility and download speed for users of various geographical regions. We restrict the analysis to human downloads (Table SI 1.13). The main analyses focus on differences in medians, given the huge volatility in means induced by a few extreme outliers (Figures SI 1.2 to SI 1.4).

3.7 Results

Figure 2 traces the median number of PyPI downloads over time. We expect to see the trajectory of the downloads change after the treatment but don't see any evidence for it. The ITT estimates from Table SI 1.10 confirm this observation. One month after intervention (on June 21), the median number of downloads in the low-dosage and high-dosage groups were statistically indistinguishable from the control group. We estimated differences in medians at four more time points (over the next four months) but never found any significant difference between the groups. We also estimated a model that allows treatment heterogeneity over time

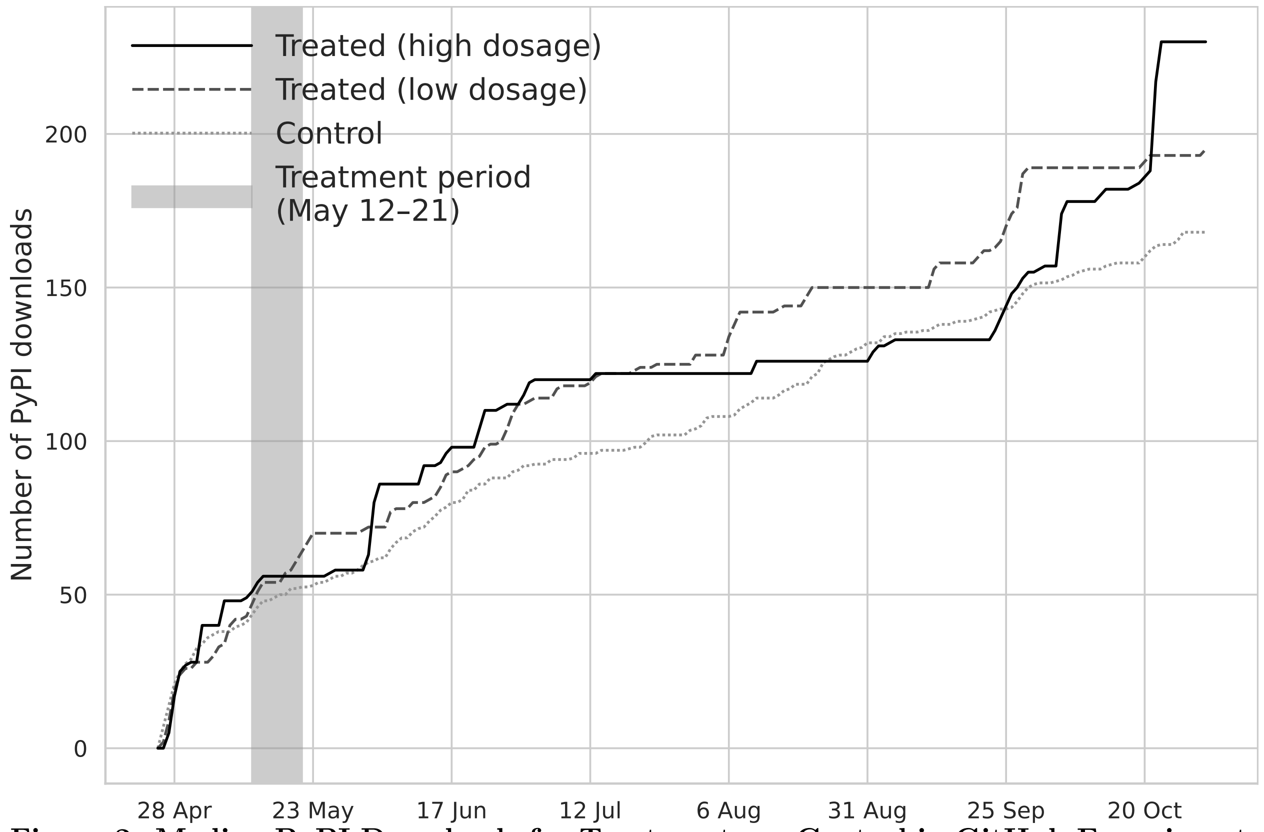


Figure 2. Median PyPI Downloads for Treatment vs. Control in GitHub Experiment. The figure plots the median of cumulative downloads for each of the three groups. Each point is a day averaged within the group for 622 packages and 118,180 package days. Treatment is distinguished by low and high dosage (see Section 3.1). The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table SI 1.13). See also Figures SI 1.3 to SI 1.4 for the time series of individual packages. Table SI 1.10 reports estimates of differences in medians. Figure SI 1.2 plots the means.

via a linear trend. Neither the low-dosage nor high-dosage group has a trend different from that of the control group (see Table SI 1.10). In Appendix SI 1.1.7, we examine differences in means and come to similar conclusions. We also estimate the LATE using the treatment assignment as an instrument for compliance (receiving at least 20 stars, see Figure 1). The mean differences in downloads for compliers are larger than the ITT estimates, as anticipated, but are non-significant (Table SI 1.12).[6] [7]

4 PyPI Experiment

We experimentally manipulate human downloads and test the hypotheses that higher human downloads lead to yet higher future human downloads.[8]

4.1 Design

We randomly sampled 50,000 packages from the PyPI repository and filtered to those with at least five human downloads (Section 3.6). This left us with 23,916 packages. We then randomly assigned 20% of the packages to the treatment group ($n = 4,814$) and the remaining 80% to the control group ($n = 19,102$). For packages in the treatment group, we wrote a script to download the same package 100 times. We downloaded the package in a way that each download shows up in the official Linehaul numbers (Section 3.6).[9]. We treated the

[6]Subsetting downloads to just those installed by *pip* (Table SI 1.13), the most common human installer yields similar answers.

[7]We also tested whether the manufactured stars in the treatment groups led to yet more stars (above and beyond what we added) and find no evidence of this (untabulated).

[8]We also analyzed if human downloads Granger cause humans downloads (Appendix SI 1.3). We find that they do. But as we noted above, we are wary to read too much into the results given they conflict with the experimental results.

[9]When installing packages, most user systems usually cache the source download files (e.g., `.tar.gz`, etc.) on the local drive to save on bandwidth resources. Cognizant of this, we wrote the script so that it installs packages without using the cached download directory. This option forces the process to always download the package from PyPI instead of using the cached source files.

packages between June 3, 2023 and June 8, 2023.

4.2 Results

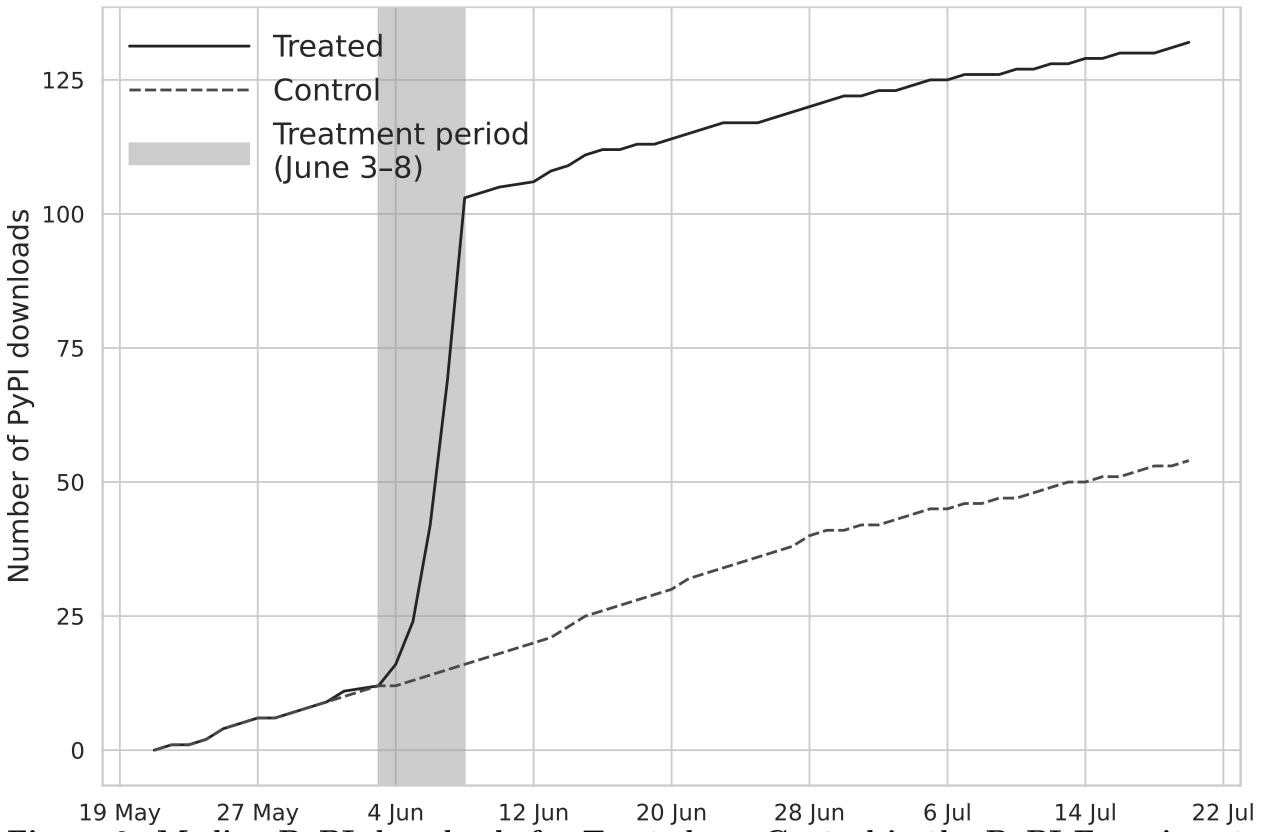


Figure 3. Median PyPI downloads for Treated vs. Control in the PyPI Experiment. The figure shows trends in median daily downloads for the treated packages (n = 4,814) and control group packages (n = 19,102) for 1,458,876 package-day observations. The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table SI 1.13). See Figure SI 1.5 for the same figure of mean downloads. Table SI 1.14 reports the estimates in the differences in medians.

Figure 3 visualizes our results for the PyPI experiment. Downloads are extremely volatile. To mute the effects of extreme outliers, we focus on the medians (as with Section 3). Figure 3 plots medians for the treatment and control groups. Pre-treatment, little separates the treatment and control groups. The treatment causes the daily median series for the treatment and control group to diverge over the treatment period (June 3–8). On June 8, the median treatment package has 83 more downloads relative to the control group ($p < .001$, column (1) of Table SI 1.14). However, after the treatment application period, the difference between the series is roughly constant. Treated packages, if anything, have a less sharp slope than the control group; the median difference is .2 fewer downloads per day ($p < .001$, column (2) of Table SI 1.14). The LATE estimates for compliers are no different (Table SI 1.15). Overall, providing social proof doesn't appear to increase downloads of the treated packages.

5 Discussion

Astroturfing using commercially bought stars on GitHub doesn't help, plausibly, because the users who like the repository have fake followers. We also see no evidence that additional official Python downloads increase the number of Python packages downloaded. But we are mindful of Carl Sagan's famous aphorism: "The absence of evidence is not evidence of absence." It is plausible that the effect was too small to detect or that if the treatment had been even more intense, it would have persuaded more people. The threat remains as long as social proof metrics remain manipulable and people are insufficiently aware of their manipulability.

References

Amblee, Naveen and Tung Bui. 2011. “Harnessing the influence of social proof in online shopping: The effect of electronic word of mouth on sales of digital microproducts.” *International journal of electronic commerce* 16(2):91–114.

Cialdini, Robert B. 2003. *Influence.* Influence At Work.

Messing, Solomon and Sean J Westwood. 2014. “Selective exposure in the age of social media: Endorsements trump partisan source affiliation when selecting news online.” *Communication research* 41(8):1042–1063.

Rao, Hayagreeva, Henrich R Greve and Gerald F Davis. 2001. “Fool’s gold: Social proof in the initiation and abandonment of coverage by Wall Street analysts.” *Administrative science quarterly* 46(3):502–526.

Salganik, Matthew J, Peter Sheridan Dodds and Duncan J Watts. 2006. “Experimental study of inequality and unpredictability in an artificial cultural market.” *science* 311(5762):854–856.

Seabold, Skipper and Josef Perktold. 2010. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference.*

Venema, Tina AG, Floor M Kroese, Jeroen S Benjamins and Denise TD De Ridder. 2020. “When in doubt, follow the crowd? Responsiveness to social proof nudges in the absence of clear preferences.” *Frontiers in psychology* 11:499433.

SI 1 Supporting Information

SI 1.1 GitHub Experiment

SI 1.1.1 Balance tests: GitHub repositories characteristics

Table SI 1.1. Balance tests: GitHub repositories characteristics

	(1)		(2)		(3)		(3)-(2)	
	Full sample		Control packages		Treated packages		Pairwise t-test	
Variable	N	Mean/(SE)	N	Mean/(SE)	N	Mean/(SE)	N	Normalized difference
Repository size	582	7.85 (1.22)	485	8.44 (1.43)	97	4.89 (1.63)	582	-0.14
Forked = 1	582	0.04 (0.01)	485	0.04 (0.01)	97	0.05 (0.02)	582	0.07
Year created	582	2022.31 (0.08)	485	2022.32 (0.08)	97	2022.28 (0.20)	582	-0.02
Subscribers	582	4.33 (0.54)	485	3.84 (0.50)	97	6.75 (2.02)	582	0.18**
Has issues = 1	582	0.96 (0.01)	485	0.96 (0.01)	97	0.95 (0.02)	582	-0.05
Number of forks	582	19.87 (4.78)	485	15.22 (3.92)	97	43.12 (20.89)	582	0.18**
Number of open issues	582	4.57 (1.14)	485	4.19 (1.26)	97	6.46 (2.67)	582	0.08
Number of topics listed	582	1.76 (0.14)	485	1.76 (0.15)	97	1.76 (0.35)	582	0.00
Python = 1	582	0.86 (0.01)	485	0.87 (0.02)	97	0.82 (0.04)	582	-0.13

Notes—Table reports the balance in GitHub repository characteristics between the treated and control Python packages. Column (1) reports the mean of repository characteristics. Column (2) reports the mean of the control packages. Column (3) reports the mean of the treated packages. Standard errors of the mean are in parentheses. The last column reports the standardized mean difference. The repository size is in megabytes. Forked indicates whether the repository was forked from another existing repository. Topics are optional labels for a GitHub repository (e.g., web application, encryption, Python). Python = 1 indicates Python is the primary detected language. See Table SI 1.2 for the same table with low and high dosage treatments. See Table SI 1.3 for the same table reporting balance for package description and dependency balance. Significance levels: $^{***}p < .01; ^{**}p < .05; ^{*}p < .1$.

Table SI 1.2. Balance tests: GitHub repositories characteristics (low-dosage and high-dosage treatment groups vs. control)

	(1)		(2)		(3)		(2)-(1)		(3)-(1)	
	Control packages		Treated (low dose)		Treated (high dose)		Pairwise t-test			
Variable	N	Mean/(SE)	N	Mean/(SE)	N	Mean/(SE)	N	Normalized difference	N	Normalized difference
Repository size	485	8.44	73	4.43	24	6.28	558	-0.16	509	-0.09
		(1.43)		(1.98)		(2.76)				
Forked = 1	485	0.04	73	0.04	24	0.08	558	0.02	509	0.19
		(0.01)		(0.02)		(0.06)				
Year created	485	2022.32	73	2022.49	24	2021.63	558	0.10	509	-0.30*
		(0.08)		(0.19)		(0.55)				
Subscribers	485	3.84	73	4.45	24	13.75	558	0.05	509	0.42***
		(0.50)		(1.63)		(6.37)				
Has issues = 1	485	0.96	73	0.96	24	0.92	558	0.00	509	-0.17
		(0.01)		(0.02)		(0.06)				
Number of forks	485	15.22	73	17.05	24	122.42	558	0.02	509	0.40***
		(3.92)		(11.25)		(76.12)				
Number of open issues	485	4.19	73	3.64	24	15.04	558	-0.02	509	0.29*
		(1.26)		(1.70)		(9.39)				
Number of topics listed	485	1.76	73	1.84	24	1.54	558	0.02	509	-0.08
		(0.15)		(0.44)		(0.49)				
Python = 1	485	0.87	73	0.82	24	0.83	558	-0.13	509	-0.10
		(0.02)		(0.05)		(0.08)				

Notes— Same as Table SI 1.1, except the treatment group is distinguished by low and high dosage treatment packages (see Section 3.2). *** Significant at the 1 percent level. ** Significant at the 5 percent level. * Significant at the 10 percent level.

SI 1.1.2 Balance tests: PyPI package characteristics

Table SI 1.3. Balance tests: PyPI package characteristics

	(1)		(2)		(3)		(3)-(2)	
	Full sample		Control packages		Treated packages		Pairwise t-test	
Variable	N	Mean/(SE)	N	Mean/(SE)	N	Mean/(SE)	N	Normalized difference
Package description length	622	2458.61 (143.49)	522	2516.53 (161.85)	100	2156.24 (287.41)	622	-0.11
Package description length (cleaned)	622	2363.36 (138.58)	522	2416.14 (156.14)	100	2087.89 (280.47)	622	-0.10
Number of dependencies	622	3.73 (0.31)	522	3.60 (0.34)	100	4.36 (0.66)	622	0.10

Notes—Table reports the balance in PyPI package characteristics between the treated and control Python packages. Column (1) reports the mean of repository characteristics. Column (2) reports the mean of the control packages. Column (3) reports the mean of the treated packages. Standard errors of the mean are in parentheses. The last column reports the standardized mean differences. Package description length is the length of the (optional) package description, which can include instructions on installation and usage, etc. These usually come from a `Readme` file. The description may have been marked up to be rendered in HTML, so we do some basic cleaning of the raw text to convert HTML to text and report it in the table. Number of dependencies is based on the number of requirements the package relies on. See also Table SI 1.1 for balance tests in GitHub repository characteristics. See Table SI 1.4 for the same table with low and high-dosage treatments. Significance levels: $^{***}p < .01; ^{**}p < .05; ^{*}p < .1$.

Table SI 1.4. Balance tests: PyPI package characteristics (low-dosage and high-dosage treatment groups vs. control)

	(1)		(2)		(3)		(2)-(1)		(3)-(1)	
	Control		Treated (low)		Treated (high)		Pairwise t-test			
Variable	N	Mean/(SE)	N	Mean/(SE)	N	Mean/(SE)	N	Normalized difference	N	Normalized difference
Package description length	522	2516.53 (161.85)	75	2145.51 (351.09)	25	2188.44 (471.91)	597	-0.11	547	-0.11
Package description length (cleaned)	522	2416.14 (156.14)	75	2077.33 (342.40)	25	2119.56 (462.04)	597	-0.10	547	-0.10
Number of dependencies	522	3.60 (0.34)	75	4.55 (0.77)	25	3.80 (1.35)	597	0.13	547	0.03

Notes— Same as Table SI 1.3, except the treatment group is distinguished by low and high dosage treatment packages (see Section 3.2). See also Table SI 1.1 and Table SI 1.2. Significance levels: $^{***}p < .01; ^{**}p < .05; ^{*}p < .1$.

SI 1.1.3 Balance tests: GitHub user characteristics

Table SI 1.5. Balance tests: GitHub user characteristics

	(1)		(2)		(3)			(3)-(2)
	Full sample		Control packages		Treated packages			Pairwise t-test
Variable	N	Mean/(SE)	N	Mean/(SE)	N	Mean/(SE)	N	Normalized difference
Number of repositories	545	57.57 (12.88)	453	51.80 (14.06)	92	85.96 (32.05)	545	0.11
Number of gists	545	3.28 (0.56)	453	3.03 (0.57)	92	4.51 (1.72)	545	0.10
Number of followers	545	189.85 (78.36)	453	194.56 (93.34)	92	166.66 (66.05)	545	-0.02
Number of people followed	545	11.40 (1.44)	453	10.81 (1.58)	92	14.32 (3.45)	545	0.11
Year created	545	2017.33 (0.16)	453	2017.43 (0.18)	92	2016.86 (0.36)	545	-0.16
Year updated	545	2017.33 (0.16)	453	2017.43 (0.18)	92	2016.86 (0.36)	545	-0.16
Organization = 1	545	0.22 (0.02)	453	0.22 (0.02)	92	0.25 (0.05)	545	0.07
List company = 1	545	0.27 (0.02)	453	0.26 (0.02)	92	0.34 (0.05)	545	0.17
List email = 1	545	0.32 (0.02)	453	0.32 (0.02)	92	0.34 (0.05)	545	0.04
List personal site = 1	545	0.46 (0.02)	453	0.45 (0.02)	92	0.51 (0.05)	545	0.12
List brief bio = 1	545	0.48 (0.02)	453	0.47 (0.02)	92	0.51 (0.05)	545	0.08
Brief bio length	545	28.44 (1.75)	453	28.94 (1.95)	92	25.98 (3.84)	545	-0.08

Notes—Table reports the balance in GitHub user characteristics between the treated and control Python packages. Column (1) reports the mean of user characteristics. Column (2) reports the mean of users of the control packages. Column (3) reports the mean of users of the treated packages. Standard errors of the mean are in parentheses. The last column reports the standardized mean difference. The year created and year updated indicate when the GitHub user account was created and updated, respectively. The organization indicates whether the package is hosted in an organization account. Brief bio length is the number of characters in the (optional) biography. See Table SI 1.6 for the same table with low and high-dosage treatments. Significance levels: $^{***}p < .01$;$^{**} p < .05$;$^{*} p < .1$.

Table SI 1.6. Balance tests: GitHub user characteristics (low-dosage and high-dosage treatment groups vs. control)

		(1)		(2)		(3)		(2)-(1)		(3)-(1)
		Control		Treated (low)		Treated (high)		Pairwise t-test		
Variable	N	Mean/(SE)	N	Mean/(SE)	N	Mean/(SE)	N	Normalized difference	N	Normalized difference
Number of repositories	453	51.80 (14.06)	70	93.61 (41.71)	22	61.59 (19.73)	523	0.13	475	0.04
Number of gists	453	3.03 (0.57)	70	3.00 (1.09)	22	9.32 (6.32)	523	-0.00	475	0.28**
Number of followers	453	194.56 (93.34)	70	103.07 (40.03)	22	369.00 (244.44)	523	-0.06	475	0.11
Number of people followed	453	10.81 (1.58)	70	16.80 (4.45)	22	6.41 (2.21)	523	0.17	475	-0.18
Year created	453	2017.43 (0.18)	70	2017.06 (0.40)	22	2016.23 (0.79)	523	-0.10	475	-0.32
Year updated	453	2017.43 (0.18)	70	2017.06 (0.40)	22	2016.23 (0.79)	523	-0.10	475	-0.32
Organization	453	0.22 (0.02)	70	0.26 (0.05)	22	0.23 (0.09)	523	0.09	475	0.02
List company	453	0.26 (0.02)	70	0.29 (0.05)	22	0.50 (0.11)	523	0.06	475	0.51**
List email	453	0.32 (0.02)	70	0.34 (0.06)	22	0.32 (0.10)	523	0.05	475	-0.00
List personal site	453	0.45 (0.02)	70	0.47 (0.06)	22	0.64 (0.10)	523	0.04	475	0.37*
List brief bio	453	0.47 (0.02)	70	0.51 (0.06)	22	0.50 (0.11)	523	0.08	475	0.05
Brief bio length	453	28.94 (1.95)	70	23.84 (4.25)	22	32.77 (8.69)	523	-0.13	475	0.09

Notes—Same as Table SI 1.5, except the treatment group is distinguished by low and high dosage treatment packages (see Section 3.1). Significance levels: $^{***}p < .01;^{**}\,p < .05;^{*}\,p < .1$.

SI 1.1.4 Organic GitHub Starrers

Table SI 1.7. Summary statistics of organic starrers' GitHub characteristics

	(1)	(2)
	Mean (s.d.)	Median [IQR]
Public repositories	35.0 (49.2)	12.0 [6.0,61.0]
Public gists	9.4 (29.4)	0.0 [0.0,4.0]
Followers	63.6 (160.4)	9.0 [1.0,17.0]
Following	34.7 (90.5)	7.0 [0.0,20.0]
Account age (years)	8.1 (3.5)	8.5 [6.1,10.4]
Lists name	0.9 (0.4)	—
Lists company	0.4 (0.5)	—
Lists blog/website	0.6 (0.5)	—
Lists geographical location	0.5 (0.5)	—
Lists brief biography	0.5 (0.5)	—
Lists Twitter handle	0.3 (0.5)	—

Notes—Table shows summary statistics of the organic GitHub star givers listed in Table SI 1.8 (corresponding to the high dosage treatment condition in the GitHub experiment, see Section 3.2). The last five rows are indicator variables.

Table SI 1.8. List of GitHub organic starrers

	(1) Created	(2) Blog/website	(3) Brief profile biography	(4) Repos	(5) Gists	(6) Followers	(7) Following
basharnaji	2014-05-14	—	—	9	0	9	10
bmwhetter	2016-09-29	—	My name is Brian Whetter and I graduated with a...	3	0	1	1
BonaventureR	2017-09-20	bonaventureraj.me	—	6	0	3	3
c-s-ale	2016-06-02	ox.work	Lover of Machine Learning, D&D, and Physics!	28	0	13	2
chris-alexiuk	2022-09-26	—	I work on Ox and FourthBrain, and a bunch of co...	19	0	10	0
danweitzel	2013-01-15	http://danweitzel.net	—	11	5	13	25
deepankermishra	2014-10-17	—	—	5	0	5	9
dhingratul	2013-06-21	https://www.linkedin.com/in/dhingratul/	Machine Learning Engineer— Experienced research...	82	0	95	64
dhruvarora-db	2021-07-20	—	—	0	0	0	0
dwillis	2008-03-04	http://thescoop.org/	I teach data journalism at the University of Ma...	208	134	705	130
humanpranav	2020-07-15	—	Hello!	8	0	0	0
khurramnasser	2012-07-09	—	—	0	0	0	0
LSYS	2015-01-13	https://www.lucasshen.com	Applied econs — Data science — Quantitative soc...	13	4	17	407
Madhu009	2016-06-07	https://www.linkedin.com/in/madhusanjeeviai/	AI, Blockchain, Mobile Dev, Web Dev,	61	0	186	7
NoahFinberg	2014-08-27	domo.com	Data App Innovation @ Domo. Former Co-founder &...	12	4	10	20
pvbhanuteja	2017-10-30	bhanu.cyou	—	61	12	1	0
rajashekar	2011-10-20	https://rajashekar.dev/	Software Engineer, Learner, Developer, Prokopton	67	9	23	33
sharmaamits	2017-12-19	—	—	1	0	0	0
sjhangiani12	2015-05-25	sharanjhangiani.com	—	41	0	5	8
soodoku	2011-04-11	http://www.gsood.com	—	93	29	239	10

Notes—Table lists the 20 organic GitHub star givers (corresponding to the high dosage treatment condition in the GitHub experiment, see Section 3.2). See Table SI 1.7 for summary statistics of these starrers.

SI 1.1.5 Manipulation check

Table SI 1.9. GitHub Experiment: Manipulation Check.

	(1)	(2)	(3)	(4)
	Outcome variable is: GitHub stars			
	Difference in medians		Difference in means	
	On May 12	On May 21	On May 12	On May 21
Treatment (low dosage)	0.00	20.00***	−0.27	18.43***
	(0.27)	(0.27)	(2.76)	(2.55)
	[−0.52 to 0.52]	[19.48 to 20.52]	[−5.69 to 5.15]	[13.41 to 23.44]
	$< p = 1.000 >$	$< p = 0.000 >$	$< p = 0.923 >$	$< p = 0.000 >$
Treatment (high dosage)	0.00	69.00***	6.87	62.15***
	(0.87)	(0.87)	(6.81)	(4.57)
	[−1.70 to 1.70]	[67.30 to 70.70]	[−6.52 to 20.25]	[53.16 to 71.13]
	$< p = 1.000 >$	$< p = 0.000 >$	$< p = 0.314 >$	$< p = 0.000 >$
Constant	0.00	0.00	8.41***	8.69***
	(0.10)	(0.10)	(1.07)	(1.08)
	[−0.19 to 0.19]	[−0.19 to 0.19]	[6.31 to 10.52]	[6.58 to 10.81]
	$< p = 1.000 >$	$< p = 1.000 >$	$< p = 0.000 >$	$< p = 0.000 >$
Median/Mean of outcome	0.0	1.0	8.7	13.7
Package observations	585	585	585	585
Day observations	1	1	1	1
Package-day observations	585	585	585	585

Note: The table presents manipulation checks for the treatment variable: GitHub stars by presenting differences in medians and means for the treatment and control groups. Column (1) reports differences in medians at the start of treatment (May 12). Column (2) reports differences in medians at the end of treatment (May 21). Columns (1)–(2) correspond to Figure 1. Columns (3)–(4) report differences in means, corresponding to Figure SI 1.1. Section 3.2 describes the high- and low-dosage treatment conditions. Standard errors are clustered by packages. Parentheses: standard errors. Square brackets: 95% confidence intervals. Angle brackets: p-values. Significance levels: $^{+}$ 0.1 * 0.05 ** 0.01 *** 0.001.

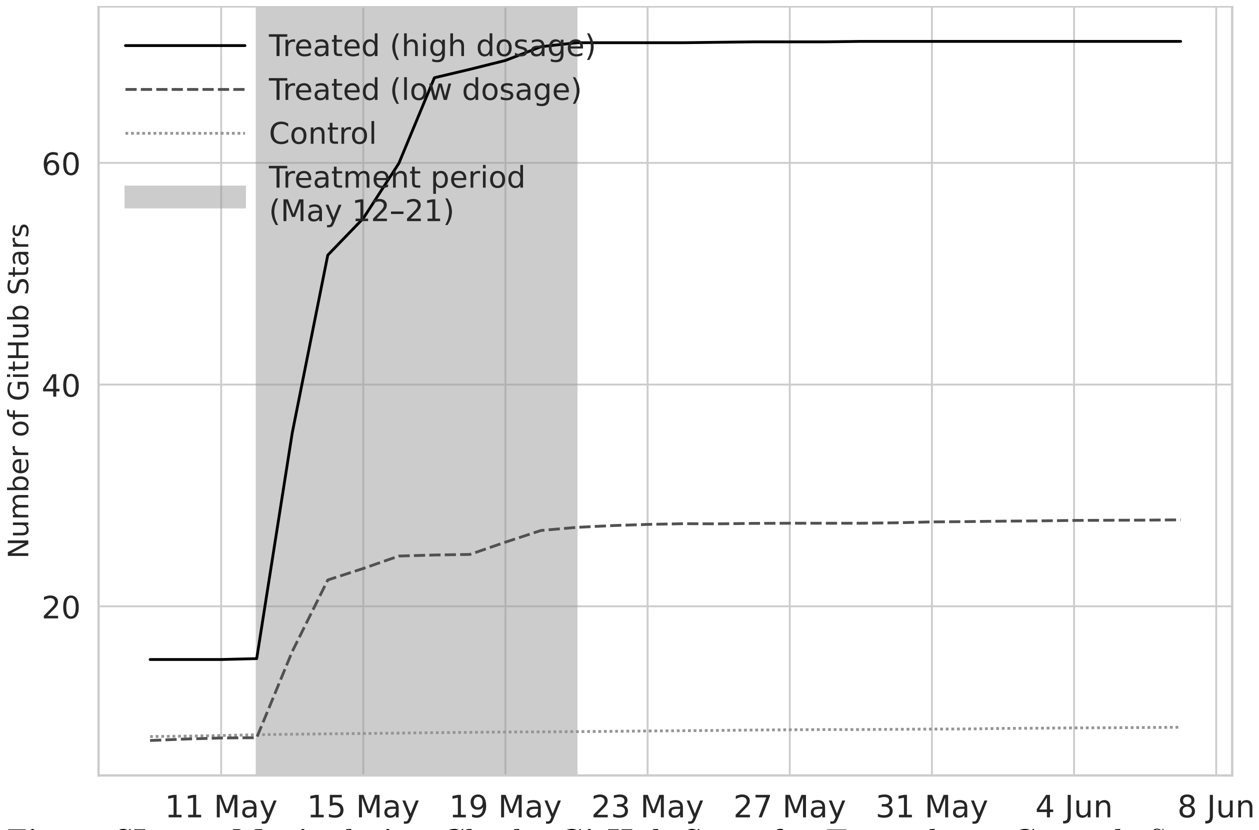


Figure SI 1.1. Manipulation Check: GitHub Stars for Treated vs. Control. Same as Figure 1, except in means. The figure reports the means version of Figure 1. The figure plots the mean number of stars on a particular day for the three groups: high-dosage treatment (market and network stars), low-dosage treatment (market stars), and control (no stars). In all, we plot data for 585 packages and 17,550 package days. The shaded vertical bar indicates the period during which the treatment was being applied. Table SI 1.9 presents estimates of the impact of treatment on downloads.

SI 1.1.6 Differences in medians

Table SI 1.10. GitHub Experiment Results - ITT estimates for medians.

	(1)	(2)	(3)	(4)	(5)	(6)
	Outcome variable is: PyPI downloads					
	On Jun 21	On Jul 21	On Aug 21	On Sep 21	On Oct 21	Full post period
Treatment (low dosage)	8.0	25.0	28.0	20.0	20.0	10.3
	(17.4)	(24.7)	(28.6)	(32.5)	(32.5)	(7.7)
	[-26.2 to 42.2]	[-23.4 to 73.4]	[-28.1 to 84.1]	[-43.9 to 83.9]	[-43.9 to 83.9]	[-4.8 to 25.3]
	$< p = 0.646 >$	$< p = 0.311 >$	$< p = 0.328 >$	$< p = 0.539 >$	$< p = 0.539 >$	$< p = 0.183 >$
Treatment (high dosage)	14.0	24.0	7.0	-7.0	-7.0	20.0
	(50.7)	(50.0)	(50.7)	(49.3)	(49.3)	(16.3)
	[-85.7 to 113.7]	[-74.3 to 122.3]	[-92.6 to 106.6]	[-103.9 to 89.9]	[-103.9 to 89.9]	[-11.8 to 51.9]
	$< p = 0.783 >$	$< p = 0.632 >$	$< p = 0.890 >$	$< p = 0.887 >$	$< p = 0.887 >$	$< p = 0.218 >$
Linear trend						0.7^{***}
						(0.0)
						[0.6 to 0.8]
						$< p = 0.000 >$
Treatment (low dosage) × Linear trend						0.2
						(0.4)
						[-0.6 to 0.9]
						$< p = 0.695 >$
Treatment (high dosage) × Linear trend						-0.1
						(0.7)
						[-1.5 to 1.3]
						$< p = 0.910 >$
Constant	84.0^{***}	98.0^{***}	119.0^{***}	140.0^{***}	140.0^{***}	56.8^{***}
	(4.8)	(6.3)	(8.2)	(10.0)	(10.0)	(4.1)
	[74.6 to 93.4]	[85.5 to 110.5]	[103.0 to 135.0]	[120.4 to 159.6]	[120.4 to 159.6]	[48.6 to 64.9]
	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.000 >$
Median of outcome	86.0	103.5	125.0	141.0	141.0	112.0
Package observations	622	622	622	622	622	622
Day observations	1	1	1	1	1	165
Package-day observations	622	622	622	622	622	102,630

Note: The table presents Intention-to-Treat (ITT) estimates for median downloads in the GitHub experiment. Corresponds to Figure 2. Columns (1)–(5) report snapshots of the difference in medians at various dates. Column (6) reports post-treatment differences in medians over the full post-treatment period of 165 days, allowing for heterogeneous treatment effects through a linear time trend. Parentheses: standard errors. Square brackets: 95% confidence intervals. Angle brackets: p-values. Significance levels: $^{***}p < .001;^{***} p < .01;^{**} p < .05;^{+} p < .1$.

SI 1.1.7 Differences in Means

While there appears to be a break in trend in Figure SI 1.2, we note three additional time series behaviors. First, the within-group variance is large, as will be evident in the estimates (Table SI 1.11). Second, this, at least in part, can be explained by a few extreme outliers (Figures SI 1.3 to SI 1.4). Third, and relatedly, we do not observe a similar trend break for the medians (Figure 2).

Table SI 1.14 reports the ITT estimates. Approximately one month after intervention, on June 21, the low-dosage group's mean download tally is no higher than the control group ($p = .051$), while the high-dosage group's download tally is higher but statistically indistinguishable from the control group ($p = .495$, Table SI 1.11). Three months after intervention (on August 21), the low-dosage group's mean download tally is 1,180 lower than the control group ($p = .069$), while the high-dosage group's download tally is 7k higher than the control group ($p = .388$), with the standard errors of the estimates always in the same order of magnitude as the estimates (Table SI 1.11). We also estimate a model that allows the treatment effect to vary over time (column (6) of Table SI 1.11). Neither the low-dosage nor high-dosage group has a trend different from the control group. If anything, the low-dosage treatment group has a weaker trend than the control group ($p = .066$).

The figure plots the median of cumulative downloads for each of the three groups. Each point is a day averaged within the group for 622 packages and 118,180 package days. Treatment is distinguished by low and high dosages (see Section 3.1). The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table SI 1.13). See also Figures SI 1.3 to SI 1.4 for the time series of individual packages. Table SI 1.10 reports estimates of differences in medians. Figure SI 1.2 plots differences in means of cumulative downloads by groups.

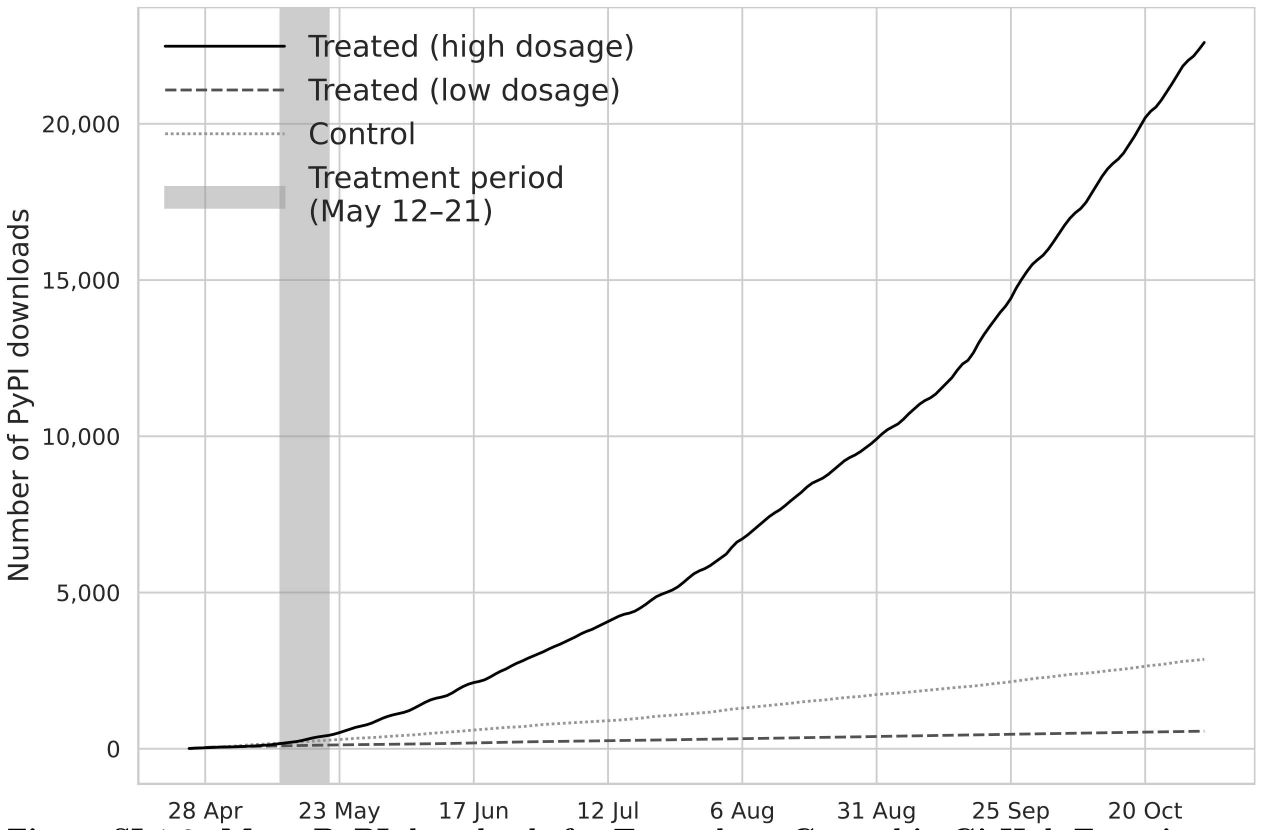


Figure SI 1.2. Mean PyPI downloads for Treated vs. Control in GitHub Experiment. Same as Figure 2, except for means. The figure plots the mean of cumulative downloads for each of the three groups. Each point is a day averaged within the group for 622 packages and 118,180 package days. Treatment is distinguished by low and high dosages (see Section 3.1). The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table SI 1.13). See also Figures SI 1.3 to SI 1.4 for the time series of individual packages. Table SI 1.11 reports estimates of differences in medians.

Table SI 1.11. GitHub Experiment Results - ITT estimates for means.

	(1)	(2)	(3)	(4)	(5)	(6)
			Outcome variable is: PyPI downloads			
	On Jun 21	On Jul 21	On Aug 21	On Sep 21	On Oct 21	Full post period
Treatment (low dosage)	-260.1^{+}	-743.3^{+}	$-1,180.3^{+}$	$-1,595.5^{+}$	$-2,101.2^{+}$	-12.0
	(132.8)	(424.7)	(648.3)	(855.0)	(1,135.2)	(76.4)
	[−520.8 to 0.6]	[−1,577.4 to 90.8]	[−2,453.5 to 92.9]	[−3,274.6 to 83.5]	[−4,330.4 to 128.1]	[−162.0 to 138.0]
	$< p = 0.051 >$	$< p = 0.081 >$	$< p = 0.069 >$	$< p = 0.062 >$	$< p = 0.065 >$	$< p = 0.875 >$
Treatment (high dosage)	679.6	3,727.3	7,035.0	11,213.5	17,550.9	$-2,303.5$
	(995.8)	(4,496.9)	(8,150.4)	(12,796.6)	(19,813.3)	(2,202.8)
	[−1,276.0 to 2,635.2]	[−5,103.8 to 12,558.3]	[−8,970.8 to 23,040.8]	[−13,916.4 to 36,343.4]	[−21,358.5 to 56,460.3]	[−6,629.4 to 2,022.4]
	$< p = 0.495 >$	$< p = 0.408 >$	$< p = 0.388 >$	$< p = 0.381 >$	$< p = 0.376 >$	$< p = 0.296 >$
Linear trend						15.9^{*}
						(7.1)
						[2.0 to 29.9]
						$< p = 0.025 >$
Treatment (low dosage) × Linear trend						-13.2^{+}
						(7.1)
						[−27.2 to 0.9]
						$< p = 0.066 >$
Treatment (high dosage) × Linear trend						116.0
						(124.3)
						[−128.2 to 360.1]
						$< p = 0.351 >$
Constant	406.7^{**}	$1,022.0^{*}$	$1,543.0^{*}$	$2,043.8^{*}$	$2,643.2^{*}$	123.6^{+}
	(127.3)	(419.1)	(641.5)	(846.6)	(1,126.5)	(70.6)
	[156.8 to 656.6]	[198.9 to 1,845.1]	[283.1 to 2,802.9]	[381.2 to 3,706.3]	[431.0 to 4,855.4]	[−15.0 to 262.2]
	$< p = 0.001 >$	$< p = 0.015 >$	$< p = 0.016 >$	$< p = 0.016 >$	$< p = 0.019 >$	$< p = 0.080 >$
R^2	0.00333	0.00648	0.00880	0.0113	0.0137	0.0140
Mean of outcome	402.7	1,082.2	1,683.4	2,302.1	3,095.3	1,586.5
Package observations	622	622	622	622	622	622
Day observations	1	1	1	1	1	165
Package-day observations	622	622	622	622	622	102,630

Note: Similar to Table SI 1.10, but for differences in means. The table presents Intention-to-Treat (ITT) estimates for mean downloads in the GitHub experiment. Corresponds to Figure SI 1.2. Columns (1)–(5) report snapshots of the difference in mean at various dates. Column (6) reports post-treatment differences in means over the full post-treatment period of 165 days, allowing for heterogeneous treatment effects through a linear time trend. Parentheses: standard errors. Square brackets: 95% confidence intervals. Angle brackets: p-values. Significance levels: $^{***}p < .001;^{***} p < .01;^{**} p < .05;^{+} p < .1$.

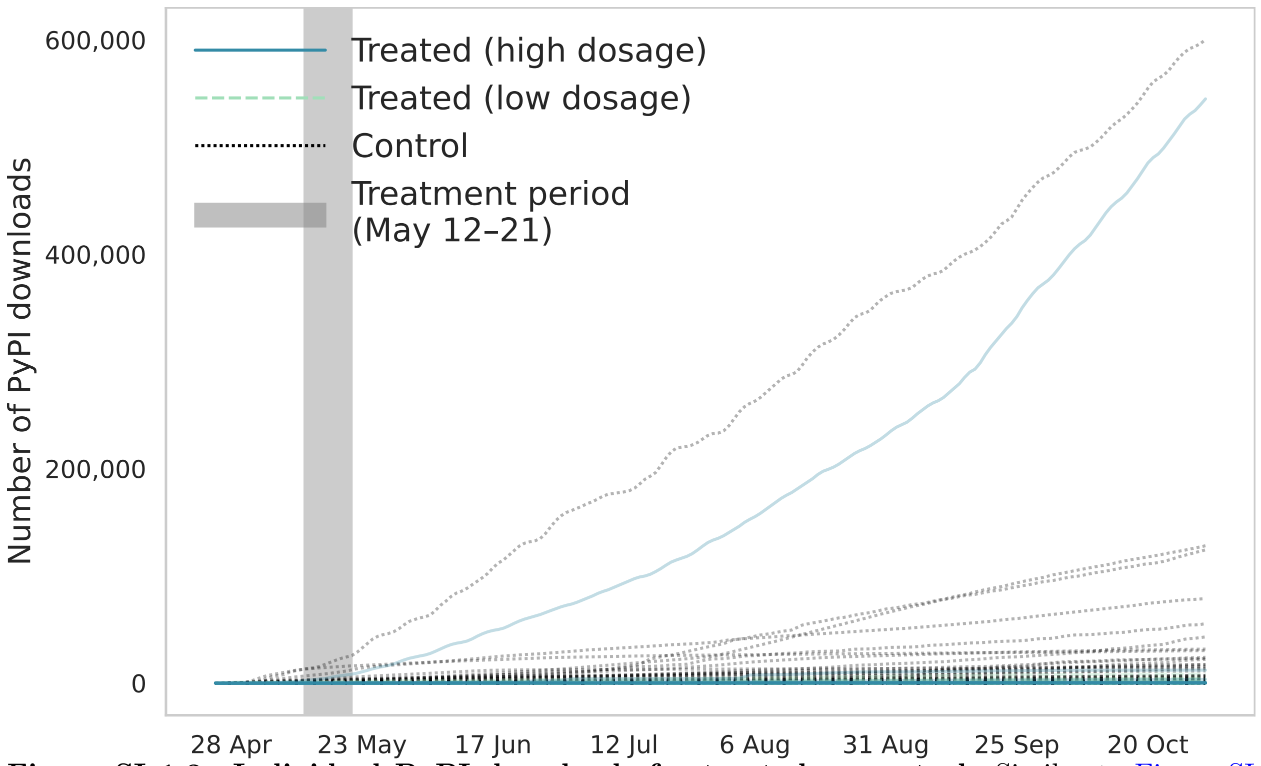


Figure SI 1.3. Individual PyPI downloads for treated vs control. Similar to Figure SI 1.2, but with the time series of the individual packages.

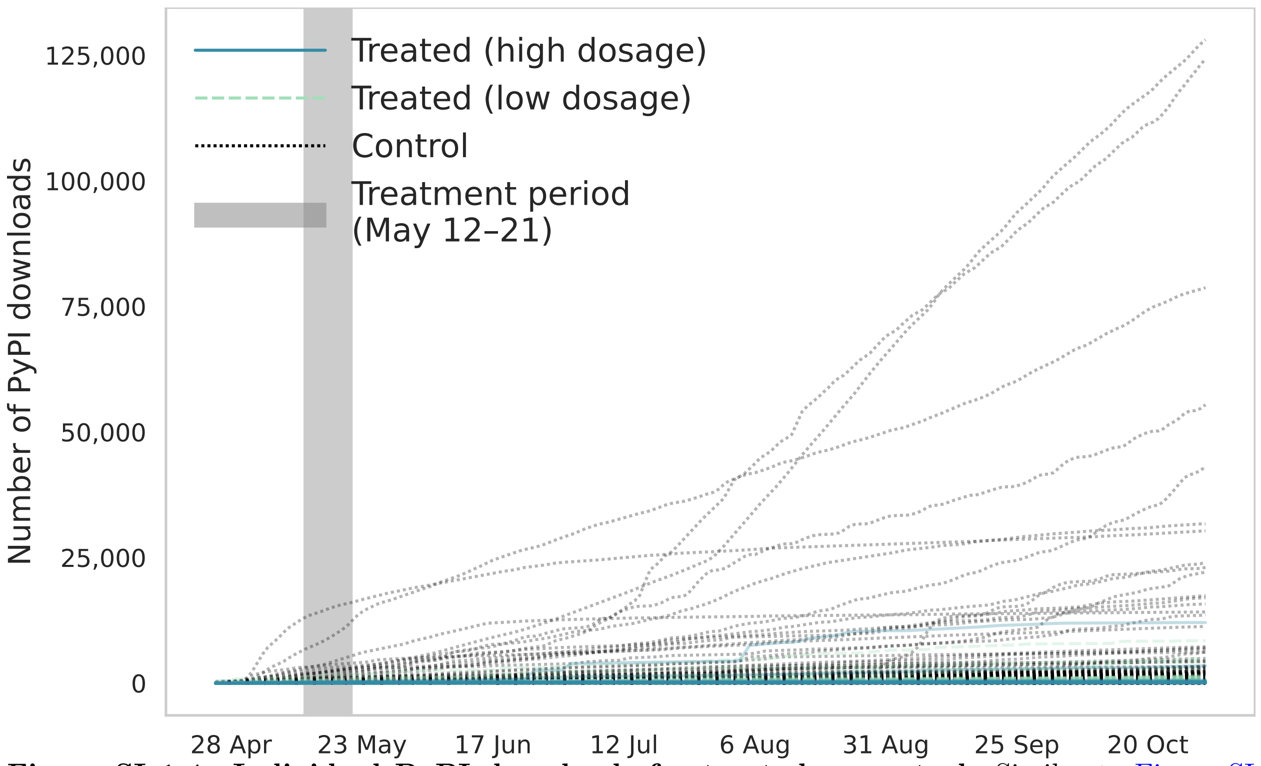


Figure SI 1.4. Individual PyPI downloads for treated vs control. Similar to Figure SI 1.3, but without the top two extreme time series.

Table SI 1.12. GitHub Experiment Results - LATE estimates for means.

	(1)	(2)	(3)	(4)	(5)	(6)
	Outcome variable is: PyPI downloads					
	On Jun 21	On Jul 21	On Aug 21	On Sep 21	On Oct 21	Full post period
Received treatment	213.9	859.3	$1,963.1$	$3,660.5$	$6,317.4$	$-1,299.7$
	$(1,347.3)$	$(2,662.2)$	$(4,660.9)$	$(7,275.5)$	$(11,067.7)$	$(1,245.9)$
	$[-2,426.8$ to $2,854.7]$	$[-4,358.5$ to $6,077.1]$	$[-7,172.1$ to $11,098.3]$	$[-10,599.3$ to $17,920.2]$	$[-15,375.0$ to $28,009.7]$	$[-3,741.6$ to $1,142.2]$
	$< p = 0.874 >$	$< p = 0.747 >$	$< p = 0.674 >$	$< p = 0.615 >$	$< p = 0.568 >$	$< p = 0.297 >$
Linear trend						15.9^{*}
						(7.1)
						[2.0 to 29.8]
						$< p = 0.025 >$
Received treatment × Linear trend						42.5
						(71.4)
						$[-97.5$ to $182.5]$
						$< p = 0.552 >$
Constant	655.7^{**}	$1,041.0^{*}$	$1,556.9^{*}$	$2,066.6^{*}$	$2,664.7^{*}$	123.6^{+}
	(249.4)	(429.5)	(645.6)	(856.4)	$(1,135.8)$	(70.5)
	$[166.9$ to $1,144.4]$	$[199.2$ to $1,882.8]$	$[291.5$ to $2,822.2]$	$[388.0$ to $3,745.2]$	$[438.6$ to $4,890.9]$	$[-14.6$ to $261.8]$
	$< p = 0.009 >$	$< p = 0.015 >$	$< p = 0.016 >$	$< p = 0.016 >$	$< p = 0.019 >$	$< p = 0.080 >$
Mean of outcome	671.2	$1,103.2$	$1,698.9$	$2,331.4$	$3,121.8$	$1,586.5$
Package observations	622	622	622	622	622	622
Day observations	1	1	1	1	1	165
Package-day observations	622	622	622	622	622	$102,630$

Note: The table presents Local Average Treatment Effect (LATE) estimates for mean downloads in the GitHub experiment. Compliers (those who "Received treatment") are defined as those receiving at least 20 stars at the end of the treatment window. LATE estimates are from instrumental variable regressions, instrumenting compliers with the random treatment assignment. Corresponds to Figure SI 1.2. Columns (1)–(5) report snapshots of the difference in mean at various dates. Column (6) reports post-treatment differences in means over the full post-treatment period of 165 days, allowing for heterogeneous treatment effects through a linear time trend. Parentheses: standard errors. Square brackets: 95% confidence intervals. Angle brackets: p-values. Significance levels: $^{***}p < .001;^{***}p < .01;^{**}p < .05;^{+}p < .1$.

SI 1.2 PyPI Experiment

Table SI 1.13. Classification of human vs bot downloads by package installer.

Installer Name	Type
pip	Human
Browser	Bot
Bandersnatch	Bot
setuptools	Human
Nexus	Human
requests	Bot
devpi	Bot
pdm	Human
Homebrew	Human
Artifactory	Human
OS	Human
Bazel	Human
pex	Human
conda	Human
chaquopy	Human

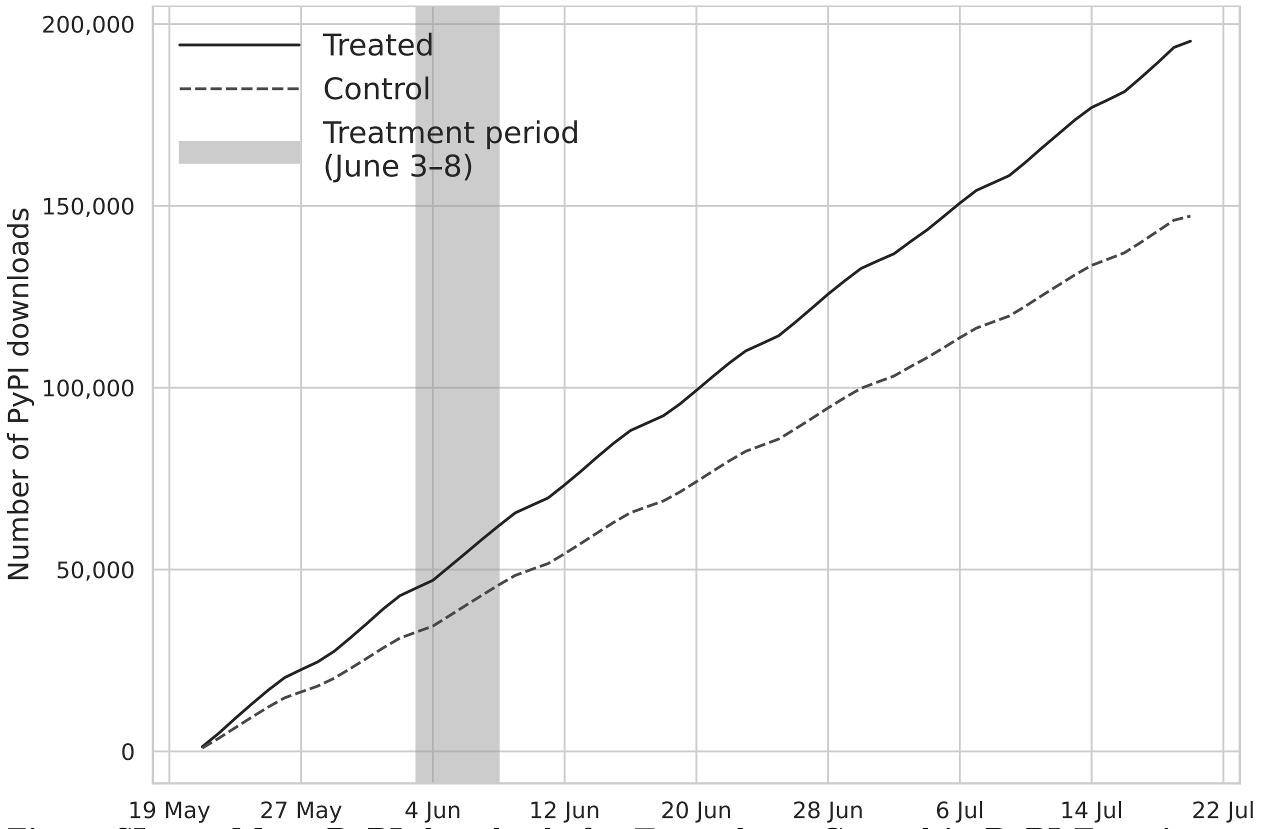


Figure SI 1.5. Mean PyPI downloads for Treated vs. Control in PyPI Experiment. Same as Figure 3, except in means. The figure shows trends in median daily downloads for the treated packages ($n = 4,814$) and control group packages ($n = 19,102$) for 1,458,876 package-day observations. The shaded vertical bar indicates the treatment period. Downloads include only human downloads (Table SI 1.13). Table SI 1.14 reports the estimates in the differences in means.

Table SI 1.14. PyPI Experiment Results - ITT estimates for medians and means.

	(1)	(2)	(3)	(4)
	Outcome variable is: PyPI downloads			
	Diff. in medians		Diff. means	
	Jun 22	Full post period	Jun 22	Full post period
Treatment group	83.0^{***}	86.7^{***}	$26,888.8$	$16,127.6$
	(0.9)	(0.4)	(52,957.6)	(30,751.4)
	[81.3 to 84.7]	[85.8 to 87.5]	[$-76,911.6$ to 130,689.1]	[$-44,147.1$ to 76,402.3]
	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.612 >$	$< p = 0.600 >$
Linear trend		0.9^{***}		$2,436.4^{***}$
		(0.0)		(551.9)
		[0.9 to 1.0]		[1,354.7 to 3,518.0]
		$< p = 0.000 >$		$< p = 0.000 >$
Treatment group × Linear trend		-0.2^{***}		748.0
		(0.0)		(1,558.3)
		[-0.3 to -0.2]		[$-2,306.3$ to 3,802.3]
		$< p = 0.000 >$		$< p = 0.631 >$
Constant	33.0^{***}	18.1^{***}	$79,904.2^{***}$	$45,253.6^{***}$
	(0.5)	(0.3)	(18,079.3)	(10,235.3)
	[32.1 to 33.9]	[17.5 to 18.7]	[44,467.6 to 115,340.8]	[25,191.7 to 65,315.5]
	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.000 >$	$< p = 0.000 >$
Median/Mean of outcome	43	49	85,317	104,119
Package observations	23,916	23,916	23,916	23,916
Day observations	1	42	1	42
Package-day observations	23,916	1,004,472	23,916	1,004,472

Note: The table presents pre-to-post treatment changes in PyPI downloads. Column (1) reports differences in means 14 days after intervention occurred (22 Jun 2023). Column (2) reports post-treatment differences in means over the 42 days, allowing for heterogeneous treatment effects through a linear time trend. Columns (1)–(2) correspond to Figure 3. Columns (3)–(4) do the same for differences in medians and correspond to Figure SI 1.5. Standard errors are clustered by packages. Parentheses: standard errors. Square brackets: 95% confidence intervals. Angle brackets: p-values. Significance levels: $^{***}p < .001;^{***} p < .01;^{**} p < .05;^{+} p < .1$.

Table SI 1.15. PyPI Experiment Results - LATE estimates for means.

	(1)	(2)
	Outcome variable is: PyPI downloads	
	Jun 22	Full post period
Received treatment	$52,427.1$	$31,445.3$
	$(103,217.8)$	$(59,946.3)$
	$[-149876.1$ to $254,730.3]$	$[-86,047.3$ to $148,937.9]$
	$< p = 0.612 >$	$< p = 0.600 >$
Linear trend		$2,436.4^{***}$
		(551.8)
		$[1,354.8$ to $3,518.0]$
		$< p = 0.000 >$
Received treatment $\times$ Linear trend		$1,458.4$
		$(3,037.7)$
		$[-4,495.4$ to $7,412.2]$
		$< p = 0.631 >$
Constant	$79,904.2^{***}$	$45,253.6^{***}$
	$(18,078.4)$	$(10,235.1)$
	$[44,471.3$ to $115,337.1]$	$[25,193.2$ to $65,314.0]$
	$< p = 0.000 >$	$< p = 0.000 >$
Mean of outcome	$85,317$	$104,119$
Package observations	$23,916$	$23,916$
Day observations	1	42
Package-day observations	$23,916$	$1,004,472$

Note: The table presents LATE estimates for mean downloads in the PyPI experiment. Compliers (those who "Received treatment") are defined as those receiving at least 50 downloads at the end of the treatment window. LATE estimates are from instrumental variable regressions, instrumenting compliers with the random treatment assignment. Column (1) reports differences in means 14 days after intervention occurred (22 Jun 2023). Column (2) reports post-treatment differences in means over the 42 days, allowing for heterogeneous treatment effects through a linear time trend. Columns (1)–(2) correspond to Figure 3. Columns (3)–(4) do the same for differences in medians and correspond to Figure SI 1.5. Standard errors are clustered by packages. Parentheses: standard errors. Square brackets: 95% confidence intervals. Angle brackets: p-values. Significance levels: $^{***}p < .001;^{***} p < .01;^{**} p < .05;^{+} p < .1$.

SI 1.3 PyPI Observational Analysis

In this section, we present observational evidence using historical download data to understand the impact of human and bot downloads on human downloads. Using vector auto-regressive (VAR) models, we find that past human downloads predict future human downloads.

SI 1.3.1 Sample and Data

We started by retrieving the full enumeration of all Python packages available from the PyPI repository. The index is extensive, with 458,274 packages at our retrieval time. We randomly sampled fifty thousand packages to keep querying costs and data processing tractable. We then queried daily downloads over a 3-month period (going back 91 days from our query date) from BigQuery and successfully retrieved n= 40,565.[10] We have the daily downloads split by package installer for each package. We classify installers into bot versus human downloads (Section 3.6). We filter our dataset to packages with at least 50 human downloads over the three months, which gives us 13,481 packages and 1,226,771 package-day observations. So, our final sample is of Python packages that have been downloaded somewhat consistently over the last three months.

SI 1.3.2 Research Design

To estimate how past downloads predict future downloads, we estimate a VAR model for each package to understand the non-experimental correlational evidence. Specifically, we want to understand the extent to which past downloads can explain future downloads.

The VAR model of each package has two equations: one where human downloads at

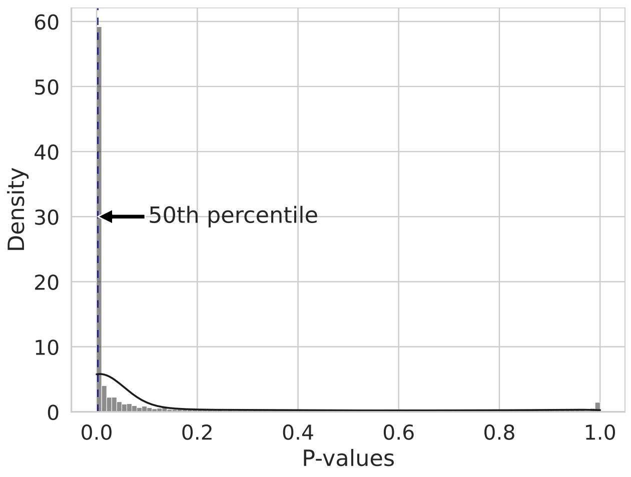
[10]`https://warehouse.pypa.io/api-reference/bigquery-datasets.html`.

time t is the outcome, and the other where bot downloads at time t is the outcome. Both outcomes are modeled as a function of their past values (e.g., human downloads) and the past values of the other (e.g., bot downloads) up to a select number of temporal lag $(t - p)$. The maximum number of lags is three weeks (21 days) to allow downloads up to three weeks old to affect present downloads. The Akaike Information Criterion then determines the number of lags (p) for each package. We implement this using *statsmodels* (Seabold and Perktold, 2010). Some packages do not have a fitted model because of numerical instability or unit roots. For packages that successfully converge with an optimal lag of at least a day (n = 6630), we further do a Granger causality test where the null is that past values of a time series do not collectively predict future values. From this, we get n = 6620 p-values from the Granger test (10 models fail to compute from numerical issues, Seabold and Perktold (2010)). Having a p-value that rejects the null at conventional levels does not mean that past downloads cause future downloads.

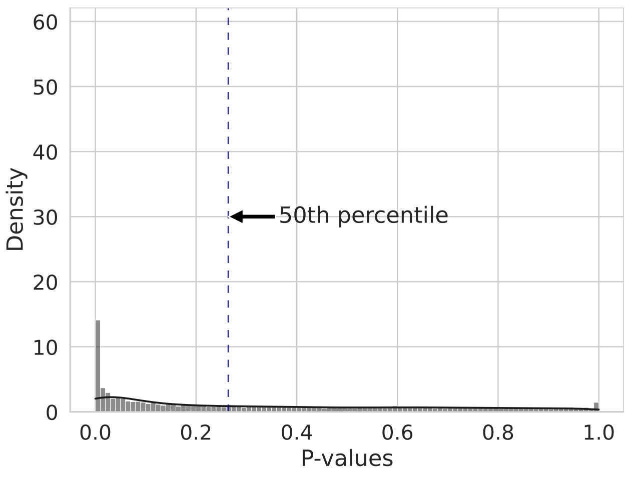
SI 1.3.3 Results

Figure SI 1.6 reports the collection of p-values from all the successful VAR runs for the four versions of the Granger tests. The vertical dashed line indicates the 50th percentile in p-values. Overall, we observe that only past human downloads can predict human downloads (Figure SI 1.6a). Its 50th percentile in p-values is approximately .002, implying more than 50% of the packages' VAR model have p-values small enough to reject the null hypothesis at the 1 percent level that past human downloads do not predict human downloads. Specifically, the percentile value for a p-value of .05 is the 69th percentile.

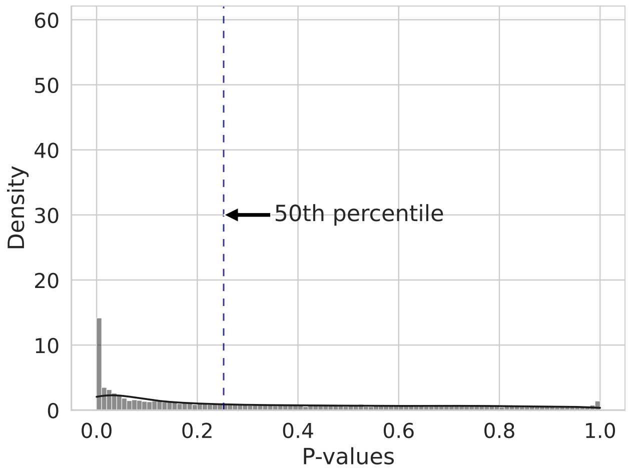
This same observation is not true for the three other versions (Figures SI 1.6a to SI 1.6d). Some packages have Granger tests where the p-values are small enough to reject the null at conventional levels. However, they are in the minority, with the 50th percentiles in p-values that much higher. The percentile values for these three groups, with a p-value of



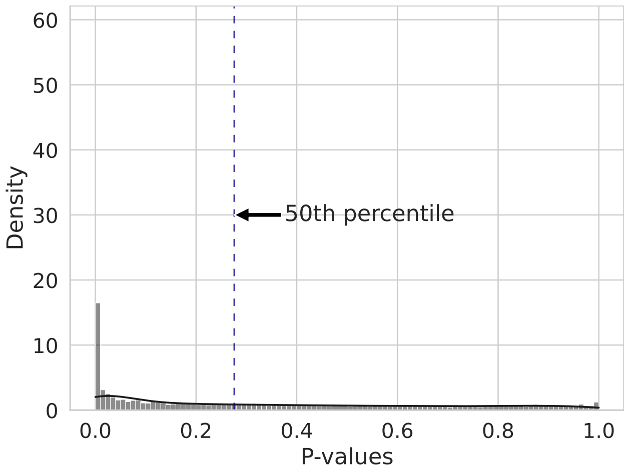
(a) Human → human



(b) Human → bot



(c) Bot → human



(d) Bot → bot

Figure SI 1.6. Distribution of P-values From Granger Causality Tests. The figure plots the distribution of p-values for each of the four versions (by panels) of the Granger tests for 6,620 packages and 602,420 package days. All plots have the same scale. Each package's downloads over three months are first estimated in a VAR model, with human downloads and bot downloads as the two outcomes. Estimated coefficients are then used to implement the Granger causality tests. P-values are binned into 100 equal-width bins, each with a range of .01. The vertical dashed line indicates the 50th percentile value in the p-values.

.05, are the 25th, 26th, and 26th percentiles.

In all, we conclude that past human downloads do not predict bot downloads, and past bot downloads predict neither human nor bot downloads.