Open Source Software Impact Analysis Report

Derek Banks - dmb3ey@virginia.edu
Camille Leonard – cvl7qu@virginia.edu
Shilpa Narayan – smn7ba@virginia.edu
Nick Thompson – nat3fa@virginia.edu

1. Background and Motivation

The National Science Foundation (NSF) recently announced their usage of $21 million to fund open source development through a new program: Pathways to Enable Open-Source Ecosystems (PEOSE). Existing NSF-funded research projects already result in publicly accessible, modifiable, and distributable open source software, data platforms, and even open hardware that catalyzes further innovation. The NSF wants to follow the best examples of open source development where the product is widely adopted and forms the foundation of a self-sustaining open source ecosystem (OSE). A distributed community of developers and a broad base of users across academia, industry, and government make up these OSEs. Despite its extensive use, reliable measures of the scope and impact of Open Source Software (OSS) are scarce. The creation and use of OSS highlights an aspect of technology diffusion and flow that is not captured in science and technology indicators

2. Project Summary

Our goal was to measure the impact of OSS using network modeling and related measurements like centrality, number of package downloads, and calculating development costs leveraged through package dependencies. Network analysis approach can be applied to open source software networks to measure influence of a package similar to a node in a network using centrality measures. The dependency network is represented by a directed acyclic graph and the graph centrality measures that we used to measure influence of packages are degree, in-degree, out-degree, and eigenvector centrality. We also gathered the data and computed the number of downloads a package had, as well as the development cost for both the package itself and its dependencies. We analyzed the correlation between these statistics to determine the influential packages and find if the number of downloads, development cost, or dependency cost was related to package influence.

Additionally, we were able to associate each package with the county it was developed in based on the country of the top GitHub contributor. We used breath first search to create a package dependency tree to properly associate all of the countries a package leverages. We then aggregated cost of all of the packages on a country level to show the flow of development costs leveraged between countries. This helps to visualize the diffusion of innovation across countries, and is a framework that could be applied to organizations or other Open Source Languages as well.

3. Data

Data Investigation:

Our project began with an exploration of different code hosting platforms such as GitHub, GitLab, SourceForge, etc. Each site was reviewed to determine whether public repositories were available and how many public repositories there were, if there was an API, how many users it had, if contributor information was available, if repositories had categories assigned. The results of this investigation can be found under the data exploration and investigation folder. The OSS_Ecosystem.xlsx file contains a summary of our research. Github was chosen as the hosting platform of interest due to the large number of users and publicly available repositories.

A webscraper was written to collect SourceForge data. The code and data can be found under the Sourceforge folder in data exploration and investigation.

Due to our data science background, we were interested in characterizing open source python packages. A list of packages was collected from the Python Package Index (PyPI).

The data collected and analyzed in this project can be found on ICPSR with repo ids: 158827 and 16848. The data is broken down as follows:

Google Big Query:

- Python Packages – This is the distribution metadata table (referred to as distribution_metadata on GCB). This table contains information about each package such as package name, version, author, maintainer, dependencies, etc. This data source is described in detail in the Appendix.
- Python Package Downloads – This is the file downloads table (referred to as file_downloads on GCB). It contains information about each package download for the subject time period. Our analysis aims to quantify the impact of OSS software packages. Therefore, we were interested in the number of downloads completed during the subject time period not information on each individual download. Data downloaded from the file_downloads table was aggregated by package name, version, and download country code to produce a download count. The downloads table was queried to aggregate downloads by package name, version, and country between 01/01/2020 to 01/01/2021. This data source is described in detail in the Appendix.

Scraped GitHub Data (sponsor):

- Commits and Sector Information – This is publicly available data on development activity of individual GitHub hosted projects and their contributors. This data was collected and aggregated by previous work conducted by Korkmaz et al. To begin, the researchers used the Ruby Gem Licensee to classify the LICENSE file in each repository. This enabled them to select only packages with OSS approved licenses. Then the GHOST.jl package was used to collect and track GitHub user attributes and scrape all the commit history. This data was filtered to exclude any forked, mirrored, or archived repositories spanning GitHub's creation in 2008 through the end of

2019. Finally, the data was de-duplicated and filtered to exclude known bot accounts and commits merged from previous repositories. The produced commits activity data set totaled 3,260,612 distinct contributors and 7,628,101 distinct repositories. This process also gathered sector information on the GitHub contributors, which contained information such as the contributor country, organization, email, etc.

4. Notebooks and Code

1. SourceForge WebScraper.ipynb – This code is the webscrapper written to collect SourceForge data.
2. Gitea API Exploration .ipynb – This code connects to Gitea and makes preliminary requests to the API.
3. gitlab_api_data.ipynb – This code pulls repo and contributor information from the GitLab API.
4. oss_2021_bigquery_pypi_meta_downloads.ipynb – This code is what we used to download the pypi data from Google BigQuery.
5. master_joins_tables_nb.ipynb – This notebook contains all of the joins and queries used to aggregate our data and create the necessary datasets.
6. pypi_dep_centrality_statistics.ipynb - This notebook contains our work to create and evaluate the centrality measures as well as find the correlations between our statistics.
7. pypi_country_diffusion.ipynb – This notebook contains our diffusion cost analysis between the countries where a package was developed and the countries that are dependent upon that package.

5. Future Work and Challenges

Our preliminary diffusion analysis can be extended to other organizations, sectors, or other Open Source programming languages such as R. The major challenges that we ran into was missing data that prevented us from showing the full picture on our country diffusion analysis. This missing data was twofold; we lacked commit data from some of the largest packages (pandas) as they are not hosted on GitHub, which is where we sourced our commit data. We also lacked the country information on some of the largest contributors, which made it difficult to correctly attribute a package to its proper country, and as such, some packages needed to be dropped. Our current diffusion analysis accounts for only 7% of the downloaded packages between 2020 and 2021.

6. Appendix

Table 1 - Distribution Metadata

The description for each field in the PyPi metadata package and additional information can be found at the following URIs:

- https://packaging.python.org/specifications/core-metadata/
- https://www.python.org/dev/peps/pep-0621/

Field name	Type	Mode	Description	Potential Use
metadata_version	STRING	NULLABLE	Version of the file format; legal values are “1.0”, “1.1”, “1.2”, “2.1” and “2.2”. Automated tools consuming metadata SHOULD warn if metadata_version is greater than the highest version they support, and MUST fail if metadata_version has a greater major version than the highest version they support (as described in PEP 440, the major version is the value before the first dot). For broader compatibility, build tools MAY choose to produce distribution metadata using the lowest metadata version that includes all of the needed fields.	
name	STRING	REQUIRED	Name of package	
version	STRING	REQUIRED	Version of package	
summary	STRING	NULLABLE	Short description of package	
description	STRING	NULLABLE	Long description of package	
description_content_type	STRING	NULLABLE	A string stating the markup syntax (if any) used in the distribution’s description, so that tools can intelligently render the description.	
author	STRING	NULLABLE		Contributor network

author_email	STRING	NULLABLE		Contributor network
maintainer	STRING	NULLABLE	Empty when author is current maintainer, only used when someone different than author is maintainer	Contributor network
maintainer_email	STRING	NULLABLE		Contributor network
license	STRING	NULLABLE	Text indicating the license covering the distribution where the license is not a selection from the “License” Trove classifiers. See “Classifier” below. This field may also be used to specify a particular version of a license which is named via the Classifier field, or to indicate a variation or exception to such a license.	Select OSI licenses
keywords	STRING	NULLABLE	A list of additional keywords, separated by commas, to be used to assist searching for the distribution in a larger catalog.	Categories
classifiers	STRING	REPEATED	Indicates development status, can be used to find stable releases of software (List of classifiers Development Status :: 1 - Planning Development Status :: 2 - Pre-Alpha Development Status :: 3 - Alpha Development Status :: 4 - Beta Development Status :: 5 - Production/Stable Development Status :: 6 - Mature Development Status :: 7 - Inactive	Categories
platform	STRING	REPEATED	Operating systems?	
home_page	STRING	NULLABLE	A string containing the URL for the distribution’s home page.	Getting more contributor information
download_url	STRING	NULLABLE	A string containing the URL from which this version of the distribution can be downloaded.	

requires_python	STRING	NULLABLE	This field specifies the Python version(s) that the distribution is guaranteed to be compatible with. Installation tools may look at this when picking which version of a project to install.	
requires	STRING	REPEATED	Dependencies uses, PEP 508 https://www.python.org/dev/peps/pep-0508/	Dependency network
provides	STRING	REPEATED		
obsoletes	STRING	REPEATED		
requires_dist	STRING	REPEATED	Dependencies uses, PEP 508 https://www.python.org/dev/peps/pep-0508/ Each entry contains a string naming some other distutils project required by this distribution. The format of a requirement string contains from one to four parts: A project name, in the same format as the Name: field. The only mandatory part. A comma-separated list of 'extra' names. These are defined by the required project, referring to specific features which may need extra dependencies. A version specifier. Tools parsing the format should accept optional parentheses around this, but tools generating it should not use parentheses. An environment marker after a	

			semicolon. This means that the requirement is only needed in the specified conditions. See PEP 508 for full details of the allowed format. The project names should correspond to names as found on the Python Package Index. Version specifiers must follow the rules described in Version specifiers.	
provides_dist	STRING	REPEATED	Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools. Each entry contains a string naming a Distutils project which is contained within this distribution. This field must include the project identified in the Name field, followed by the version : Name (Version). A distribution may provide additional names, e.g. to indicate that multiple projects have been bundled together. For instance, source distributions of the ZODB project have historically included the transaction project, which is now available as a separate distribution. Installing such a source distribution satisfies requirements for both ZODB and transaction. A distribution may also provide a “virtual” project name, which does not correspond to any separately-	

			distributed project: such a name might be used to indicate an abstract capability which could be supplied by one of multiple projects. E.g., multiple projects might supply RDBMS bindings for use by a given ORM: each project might declare that it provides ORM-bindings, allowing other projects to depend only on having at most one of them installed. A version declaration may be supplied and must follow the rules described in Version specifiers. The distribution's version number will be implied if none is specified. This field may be followed by an environment marker after a semicolon.	
obsoletes_dist	STRING	REPEATED	Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools. Each entry contains a string describing a distutils project's distribution which this distribution renders obsolete, meaning that the two projects should not be installed at the same time. Version declarations can be supplied. Version numbers must be in the format specified in Version specifiers. This field may be followed by an environment marker after a semicolon. The most common use of this field will	

			be in case a project name changes, e.g. Gorgon 2.3 gets subsumed into Torqued Python 1.0. When you install Torqued Python, the Gorgon distribution should be removed.	
requires_external	STRING	REPEATED	Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools. Each entry contains a string describing some dependency in the system that the distribution is to be used. This field is intended to serve as a hint to downstream project maintainers, and has no semantics which are meaningful to the distutils distribution. The format of a requirement string is a name of an external dependency, optionally followed by a version declaration within parentheses. This field may be followed by an environment marker after a semicolon. Because they refer to non-Python software releases, version numbers for this field are not required to conform to the format specified in PEP 440: they should correspond to the version scheme used by the external dependency. Notice that there is no particular rule on the strings to be used.	
project_urls	STRING	REPEATED	Links to various things like documentation, repo, probably not	

			useful to us	
uploaded_via	STRING	NULLABLE		
upload_time	TIMESTAMP	NULLABLE		
filename	STRING	NULLABLE		
size	INTEGER	NULLABLE		
path	STRING	NULLABLE		
python_version	STRING	NULLABLE	Python version, can be used to filter data within scope, format of this column varies (cp27, cp36, source, etc)	Filtering data
packagetype	STRING	NULLABLE	Options [bdist_egg, bdist_dumb, bdist_wininst, bdist_msi, sdist, bdist_wheel, bdist_rpm, bdist_dmg]	
comment_text	STRING	NULLABLE	Mostly null	
has_signature	BOOLEAN	NULLABLE	Don't know what this represents	
md5_digest	STRING	REQUIRED	Hash	
sha256_digest	STRING	NULLABLE	Hash	
blake2_256_digest	STRING	NULLABLE	Hash	

Caveats

In addition to the caveats listed in the background above, Linehaul suffered from a bug which caused it to significantly under-report download statistics prior to July 26, 2018. Downloads before this date are proportionally accurate (e.g. the percentage of Python 2 vs. Python 3 downloads) but total numbers are lower than actual by an order of magnitude. [Source]

Table 2 - File Downloads

Name indented according to nesting. I believe this data is in JSON format.

name	Type	Mode	Description	Potential Use
timestamp	TIMESTAMP	REQUIRED		
country_code	STRING	NULLABLE		

url	STRING	REQUIRED		
project	STRING	REQUIRED	Project name	
file	RECORD	REQUIRED		
filename	STRING	NULLABLE		
project	STRING	NULLABLE		
version	STRING	NULLABLE		
type	STRING	NULLABLE		
details	RECORD	NULLABLE		
installer	RECORD	NULLABLE		
name	STRING	NULLABLE	Name of installer	
version	STRING	NULLABLE		
python	STRING	NULLABLE		
implementation	RECORD	NULLABLE		
name	STRING	NULLABLE		
version	STRING	NULLABLE		
distro	RECORD	NULLABLE		
name	STRING	NULLABLE		

version	STRING	NULLABLE		
id	STRING	NULLABLE		
libc	RECORD	NULLABLE		
lib	STRING	NULLABLE		
version	STRING	NULLABLE		
system	RECORD	NULLABLE		
name	STRING	NULLABLE		
release	STRING	NULLABLE		
cpu	STRING	NULLABLE		
openssl_version	STRING	NULLABLE		
setuptools_version	STRING	NULLABLE		
tls_protocol	STRING	NULLABLE		
tls_cipher	STRING	NULLAB		