Open Source Software Network Analysis

# Team Members

Derek Banks (dmb3ey), Camille Leonard (cvl7qu), Shilpa Narayan (smn7ba) and Nick Thompson (nat3fa).

# Problem Statement and Background

**Abstract:** Open Source Software (OSS) is computer software with its source code shared with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Examples include Linux operating system, Apache server software, and R statistical programming software. Despite its extensive use, reliable measures of the scope and impact of OSS are scarce. The creation and use of OSS highlight an aspect of technology diffusion and flow that is not captured in science and technology indicators. Supported by the National Science Foundation (NSF) and building on research conducted over the last couple of years, we aim to measure the production, impact, and diffusion of OSS in specific sectors, institutions and geographic areas using data scraped from multiple hosting platforms (e.g., GitHub, GitLab, SourceForge). We will generate and analyze networks of contributors (through collaborations between software developers) and networks of OSS projects (through reuses across projects and shared contributors), and will identify key/influential players in this ecosystem.

# Objective

For the capstone project, our group will be focusing on conducting network modeling of OSS licensed python packages. Main areas of interest include: network modeling of package dependencies and network modeling of contributors. Specifically, we will create a network of package contributors where commits data will be used as edge weight. We will also create a network of packages which include linkages for dependencies. The measure of impact for packages will be number of downloads.

# Data Sources

Data will be collected from the PYPI API’s available through Google Big Query, Github API, and previous work conducted by the project sponsor.

Data for this project was limited to downloads occurring in the last year, defined as 01/01/2020 to 01/01/2021.

## Big Query Data Collection

The data available through [the API](https://warehouse.pypa.io/api-reference/bigquery-datasets.html) contains two tables of interest. The first data is the Metadata Table. This table contains information about each package such as package name, version, author, maintainer, dependencies, etc. The second table is the Downloads Table. It contains information about each package download. Data dictionaries can be found for each table in Appendix A.

All the data was collected from the Metadata Table. The Metadata was reduced to include only packages that are being developed on GitHub. This was accomplished by selected records which had a home page record pointing to a GitHub repository. The data was limited to GitHub only because our project objective focuses on creating a contributor network and the sponsor had a database of GitHub contributor data through 2019.

Download data was limited to downloads that occurred between 01/01/2020 and 01/01/2021.

## GitHub API - Contributor Information

GitHub has an API that can be used to access commit and contributor data. Based on the past work conducted by the sponsor’s group, we do not, at present, need to use the API.

We joined our PyPI Metadata dataset to the GitHub data which contains contributor data. The join adds contributor identifying information, commits data, ... for each package.

\*\*Details on joins\*\*



Metadata Assembly Steps

* Download metadata
  + Confirm packages aren’t missing from list
  + We found at one point in the process that packages that did not have any dependencies had been dropped from our table. We fixed that and any remaining errors were added back into the table.
  + Had to use slurm script (add more details)
* Create slug names from home\_page column
* Upload metadata table to pgadmin database
* Join metadata table to commits table and contributors table
  + Completed using the rivanna shell, couldn’t complete with python
  + CREATE MATERIALIZED VIEW gh.pypi\_slug\_contrib\_commits AS (
  + SELECT \*
  + FROM gh.temp\_meta\_1 as a
  + LEFT JOIN gh.commits\_dd\_nmrc\_jbsc as b ON b.slug = a.slug
  + LEFT JOIN gh\_cost.sectored\_fractioned\_103121 as c ON b.login = c.login
  + );
  + This view has 234,362,819 rows

## Citation as an Impact Factor

* [GitHub Statistics as a Measure of the Impact of Open-Source Bioinformatics Software](https://www.frontiersin.org/articles/10.3389/fbioe.2018.00198/full)
* [Challenges of measuring software impact through citations: An examination of the lme4 R package](https://www.sciencedirect.com/science/article/abs/pii/S1751157718304796)
  + We acquired citation data for all the citable objects identified from the previous step from three
  + Sources:
    - Web of Science Cited Reference Search
    - Web of Science Data Citation Index (DCI)
    - Google Scholar
  + This article is not encouraging towards pursuing citations as an impact measure.
* FORCE11 Software Citation Working Group
* Smith AM, Katz DS, Niemeyer KE, FORCE11 Software Citation Working Group. 2016. Software citation principles. PeerJ Computer Science 2:e86 <https://doi.org/10.7717/peerj-cs.86>
* Our sponsor told us that citations are not a good / reliable source for impact factor. Research conducted on this method supports that conclusion. We will not be pursuing citations as an impact factor. Instead, we will focus on number of downloads as the impact factor.

Research how to deal with missing data / have a preliminary idea of how that affects our project

* Missing data can be imputed using various procedures such as mean, mode, most frequent and constant and so on.
* There are also other sophisticated imputing techniques such as [iterative imputer](https://scikit-learn.org/stable/modules/impute.html) which results in higher accuracy.
* Python’s scikit-learn package offers a simple imputer and iterative imputer.
* For network analysis, there might be other procedures to impute the data as suggested in the papers below:
  + <https://www.cmu.edu/joss/content/articles/volume10/huisman.pdf>
  + <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8480427/>

Have a list of what we will be including in our graphs

* Nodes
  + Python package name
  + Contributors
  + Organizations of contributors
  + Country code of downloaders
  + Organizations of users/downloaders
  + Dependency package
  + Organizations and packages
  + Organizations and contributors
  + Countries and contributors
  + Countries and packages
  + Countries and organizations?
* Edges
  + Contributors contributing to packages
  + Dependencies between packages
  + Organizations creating packages
  + Contributors or organizations from countries
  + Contributors from organizations or countries
* Impact measures
  + # of downloads
  + # of organizations?

Data Statistics:

Pypi Distribution Meta Data:

|  |  |
| --- | --- |
| **Description** | **Row count** |
| Number of rows | ~ 5 Million |
| Number of rows with Licenses and homepage in github and released | ~2.8 Million |
| Number of rows with Licenses and homepage in github and released with dependencies | ~1.2 Million |

Downloads (one year):

|  |  |
| --- | --- |
| **Description** | **Row count** |
| Number of rows | ~ 5 Million |
|  |  |
|  |  |

# Roadblocks

Need to narrow the scope of what data we want to work with.

* Downloads for the last year? - yes
* Downloads for only packages compatible with certain python versions? - no
  + Python 3

How to deal with missing values - can we use general imputation techniques or would it be different for network analysis. - sponsor told us to worry about this later

How to deal with citations as a measure of impact. - we won’t be doing this

How are we defining contributors?

Just author, and or maintainer, and or contributors according to development repo?

Shilpa to add a description of the challenges we faced accessing the data

* The dependency data type in pypi distribution meta data table in Array[string] format. It follows PEP808 convention. To clean the data we had to explode that field into multiple rows per package and version. While doing this we used big query sql syntax to unnest the field and join it back with the data on the package name. However, we lost some packages which do not have dependencies when we did this. To overcome this, we manually queried the missing packages from the table and appended it to the dataset.
  + Cross join vs Left join?
* The home\_page link in the data doesn’t’ have any specific format. Therefore, cleaning that to get slugs was challenging and we could do our best. This is an area for improvement to see if we can enhance ways to clean it to get max possible matches of slugs from the github database which will improve contributor data
* While joining the slugs with commits and user data we faced challenges with memory while running through python over Rivanna (we are given only 256 GB memory). To overcome this, we optimized the queries to get aggregates of additions, deletions and users data and used the Rivanna shell to connect to oss postgres sql server to directly run the queries. We indexed the tables ad created materialized views to create joins.
* The networks generated are massive and it is hard to visualize through networkx. We are able to generate networks with samples of the data. Not sure if we should use pyspark to overcome this or use javascript or other technologies.

# Additional Work Done

## SourceForge Web Scraping

As part of our initial investigation phase of project planning, we spent some time analyzing open source repositories as an optional data source for our project. Our sponsor was very interested in SourceForge as an option for us to look into, so we dug deeper and created a web scraper to help gather an initial dataset. We were able to scrape

* Title
* Description
* Downloads
* URLs for each project
* Project rating (0-5 stars)
* Number of reviews
* Last updated
* Categories the project falls under
* Contributors

Unfortunately, when it came to contributors, we were unable to find great data, as SourceForge seems to only be listing the Top 5 contributors to a project. After searching through the commit history in the repository, we found that there was not sufficient contributor information to be scraped from there either.

# Projected Work for Next Semester

# Notes

[Add a working table for what we’re working on and who’s doing what]

For next week:

* Refine objective
* Get data to share with Abbas
  + Put examples of data in this document
* Make sure Abbas has access to the onedrive – confirmed he got access
* Consolidate google drive and one drive – Abbas now has access.
* Check to see if we all have access to rivanna - yes
* How to speed up our query to get our dataset - done
  + Can’t use pyarrow because it can’t process array strings which is what our dataset is

11-4-21

Expectation for the end of the semester

* Complete this document
* Add pieces of data
* Add struggles and how they were dealt with
* Create an outline of what we will be doing next semester
* Have a complete dataset
* EDA
  + How many nodes, user, number of interactions (simple operations on data and network)
* What do we want to do with this data next semester beyond creating networks
  + Centrality measures

Non-Trivial Result

* Looking at the code
  + How many variables, packages imported, functions defined as a metric of impact
* If i hide some nodes in teh graph can you predict weight. What percent of graph needs to be defined to predict the weight of the missing nodes (measure of sparcity).

Send Abbas a summary / overview of data on Wed and we won’t have meeting.

List of 10 things to find out about the graph this semester (# contributors, etc.)

11/11/21

We discovered that there are a large number of packages that do not depend on another python package and therefore were dropped in the cross join that was done to get the metadata table dependency column exploded. This was fixed by a left join. There still appeared about 10,000 packages that were missing from the package column but listed in the dependency column.

Our plan of action is to take the list of 10,000 missing packages that are dependencies and add them to the metadata table as new rows.

We’ve not completed the joins yet as we need to rerun the new metadata through a formatting code. Then we need to add this table to the database to conduct the joins with two other tables. This will result in the metadata table with contributors and commits.

11/18/21

List of 10 things to find out about the graph this semester (# contributors, etc.)

* Density
* Transitivity
* Distance
* Modularity
* Centrality measures
  + Degree centrality
  + Betweenness centrality (network could be too large for this)
  + Page rank
  + In and out degree centrality (for dependency network)

Update Sunday the 21st and second update either before thanksgiving or the 28th

# Appendix A – Google Big Query PyPI API

Table 1 – Distribution Metadata

For a description of each field [go here](https://packaging.python.org/specifications/core-metadata/).

<https://packaging.python.org/specifications/declaring-project-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](https://pypi.org/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](https://packaging.python.org/guides/analyzing-pypi-package-downloads/)]

## 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 |  |  |