IRE final report

Associated git repository

https://github.com/PulakIIIT/IRE-major-project

Introduction

Our project was to collect data related to computer software and process it to obtain high-quality tabular data. Since computer software is a huge domain we have collected data of the various packages present in the Mac, Windows, and Linux systems.

Sources for data

The first step in the process of data collection was to search for appropriate sources for the dataset. Since the domain computer software is huge we tried to look for many types of sources which we could find on the internet. Here we tried this list of sources

- Category: Lists of software: The major challenge here was that the structure of data was different across different pages and hence difficult to scrape. Also, the fields were not such which could be easily merged for data collected from different pages. Also, these were not necessarily computer software hence they would have filled wrong data in our tables with mobile software as well
- Computer Hope's Free Computer Help: This website was good in terms of effort to scrape but again it was difficult to get 30+ features from this source alone for computer software
- Find Open Datasets and Machine Learning Projects: We searched for datasets on Kaggle but most of the computer software related datasets are very sparse and small and hence not useful to us

Finally, after looking for some more sources we landed on collecting data from package managers. More about this in the next section.

Package Manager

For collecting package information for the Mac windows and Linux systems we have collected the data from their respective package managers. Basically, package managers are tools that automate the process of installing, upgrading, configuring and removing software from computer systems hence it made sense to collect packages data directly through them.

Below is the list of package managers we have collected data from.

- Mac

    - Homebrew
    - MacPorts
- Windows
    - Chocolatey
- Linux
    - APT

Detailed information about the Tables

We finally provide you with the following tables:

- Merged Table for Mac Packages
- Individual table for MacPorts (Sparse and Dense)
- Individual table for Brew (Sparse and Dense)
- Table for Chocolatey Packages (Sparse and Dense)
- Table for APT Packages (Sparse and Dense)

Mac Packages

For mac packages, we collected data from MacPorts and Homebrew package manager. After collecting we merged the two tables to get one final table. We have provided you with the merged as well as the individual tables for MacPorts and Homebrew.

Below is the general information about the combined table

- Number of rows = 2538
- Columns statistics
    - Total number of columns = 93
    - Total number of columns that have less than 20% null values = 44
    - Total number of columns that have less than 50% null values = 52

Below I would give you the details about the information one could get from this table

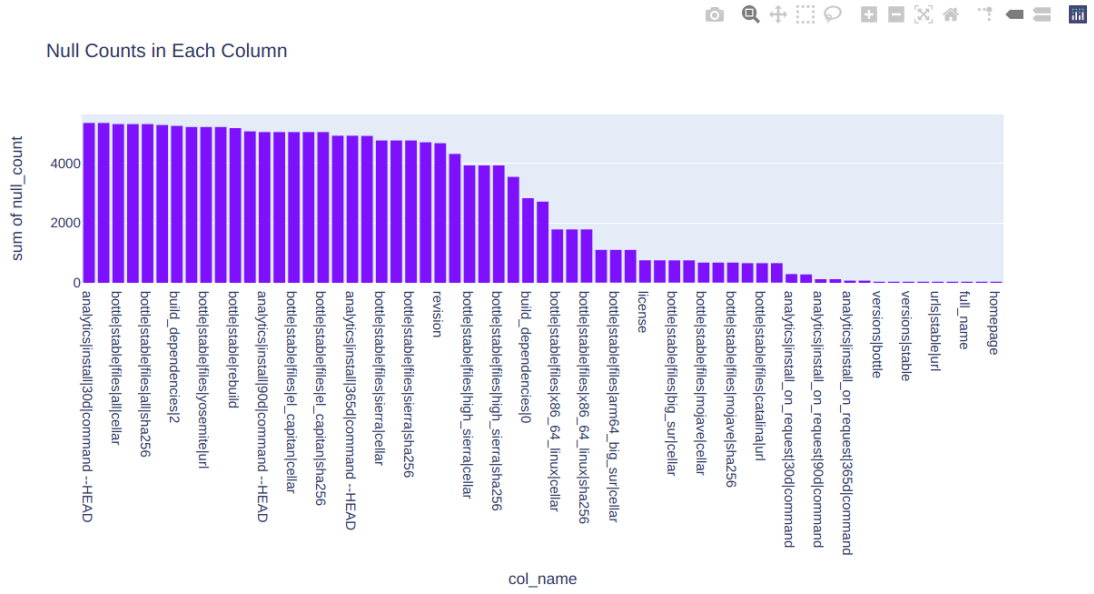
Information contained	Example data item
The name and description of the packages both according to Homebrew and MacPorts	Name/Full Name = Wget Description = Internet file retriever also long description is provided
The stable URL of the packages	Stable URL
The stable version of the packages	Homebrew = 1.21.2 MacPorts = 1.21.2
Homepage	Homepage MacPorts Homepage
Installation analytics of the packages for the last 30days/90days/365days	1876669 number of times installed in the last 365 days (and 12 more such statistics)
The stable URLs / cellar / sha256 of these packages in various MacOS versions like Catalina, Mojave etc.	Cellar = /usr/local/Cellar SHA256 for mojave = e0d4b68c9e5abeaa6395241c43307c4bbd26133cd63d1363219745357 = https://ghcr.io/v2/homebrew/core/wget/blobs/sha256:e0d4b68c9e5abeaa6395241c43307c4bbd26133cd63d13632197 (30 such columns)
The dependencies of these packages according to both	libidn2 etc. (18 such columns which tell about running and build dependency for both MacPorts and Homebrew)

MacPorts and Homebrew	
The license information	Brew License = GPL-3.0-or-later MacPorts = GPL-3+
The Platform of the packages	Darwin free BSD
Revision history	possible values in this column can be 1,2,3,...12... etc.
MacPorts category	net/www
MacPorts Maintainer information	Count = 1396 Maintainer first name = Ryan design
MacPorts Variant information	gnutls/ssl

Other than this we also provide the individual table from Homebrew and MacPorts

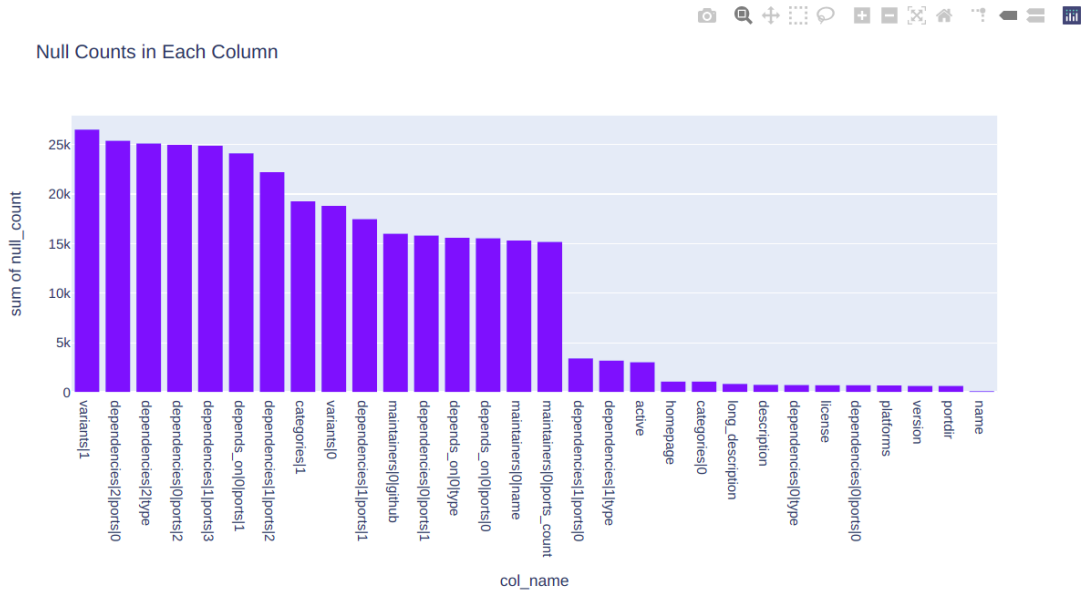
- Homebrew Table information
  - Number of rows = 5862
  - Columns statistics
    - Column Details = All the Brew specific information from the above table
    - Null Count Plot:

```
Number of rows: 5862
Number of columns: 64
Total Entries in the Data: 375168

Total Null Entries in the Data: 168304
```



- MacPorts Table information
  - Number of rows = 30546
  - Columns
    - Column Details = All the MacPorts specific information from the above table
    - Null Count Plot:

```
Number of rows: 30546
Number of columns: 30
Total Entries in the Data: 916380


Total Null Entries in the Data: 339907
```



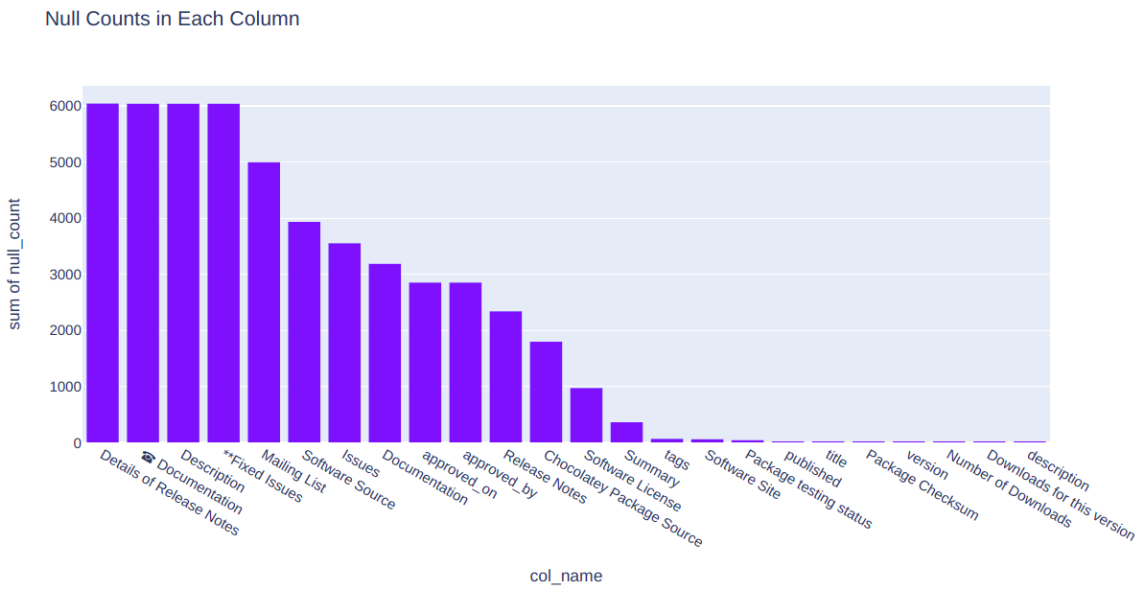
Windows Packages

For the windows packages, we collected data from the Chocolatey package manager.

Below is the general information about the combined table

- Number of rows = 6043
- Columns statistics
  - Total number of columns = 25
  - Null Count Plot:

```
Number of rows: 6043
Number of columns: 25
Total Entries in the Data: 151075


Total Null Entries in the Data: 51191
```



Below I would give you the details about the information one could get from this table

Column Name	Column Description	Example Entry
package_name	Name of the package	GoogleChrome.Dev
Number of Downloads	Number of times the package has been downloaded	6277
Software Site	URL of the software site	Google Chrome URL
Software Source	Source of software URL	Source for MyPaint
Summary	Some related information about the development	Google Chrome Dev is released once or twice a week and is generally several major releases ahead of the stable channel.
Downloads for this version	Downloads for this version	5677
Package Checksum	Checksum of the package	
approved_on	Date Time on which the package was approved	Apr 27 2017 14:53:34
Software License	License	Google chrome License
approved_by	Package approved by what Entity	AdmiringWorm
Issues	The URL for tracking issues	Example URL for MyPaint
Mailing List	The Mailing list for discussion and help	Example Mailing list for AppVeyor
published	The date published	03-04-2013
Documentation	The documentation of the package	Example documentation of SpecFlow
description	The description of the package	Example description for visual c++ "These tools allow you to build C++ libraries and applications targeting Windows desktop"
Chocolatey Package Source	The chocolatey package source	Example URL for paint
Package testing status	Testing status of the package	Passing on Jan 01 0001 00:00:00.
title	The title of the package	Google Chrome Dev Web Browser
Release Notes	The notes of the latest stable release	Example Note of Visual c++
tags	The tags associated with the packge	['google', 'chrome', 'web', 'browser', 'html5']
version	Latest stable version of the package	27.0.1453.12

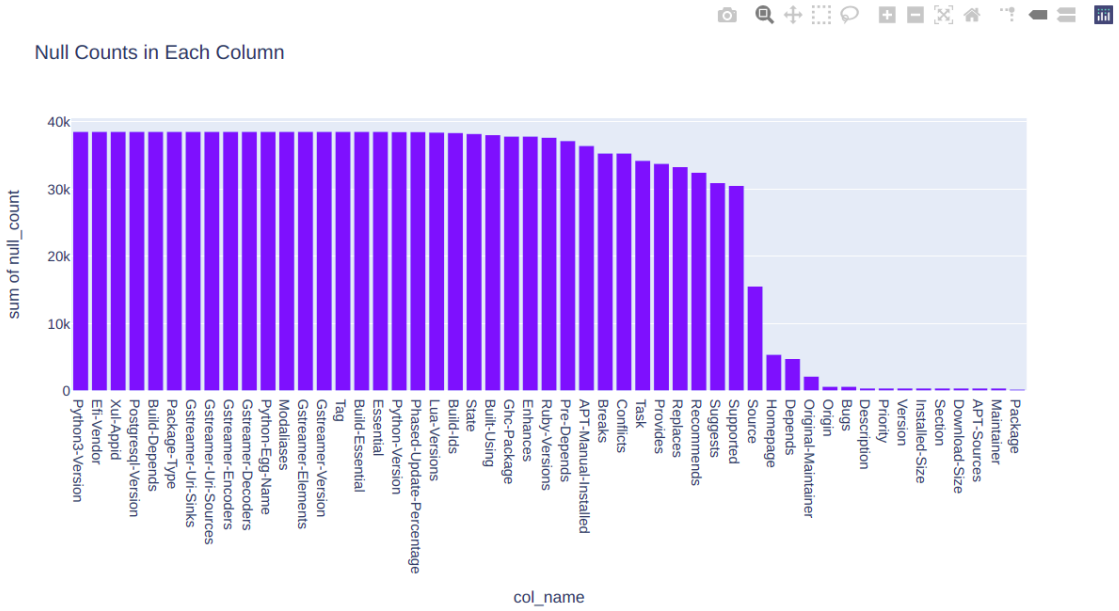
Linux Packages

For the Linux packages, we collected data from the APT package manager.

Below is the general information about the combined table

- Number of rows = 38504
- Columns statistics
  - Total number of columns = 51
  - Null Count Plot:

```
Number of rows: 38504
Number of columns: 51
Total Entries in the Data: 1963704


Total Null Entries in the Data: 1368015
```



Below I would give you the details about the information one could get from this table

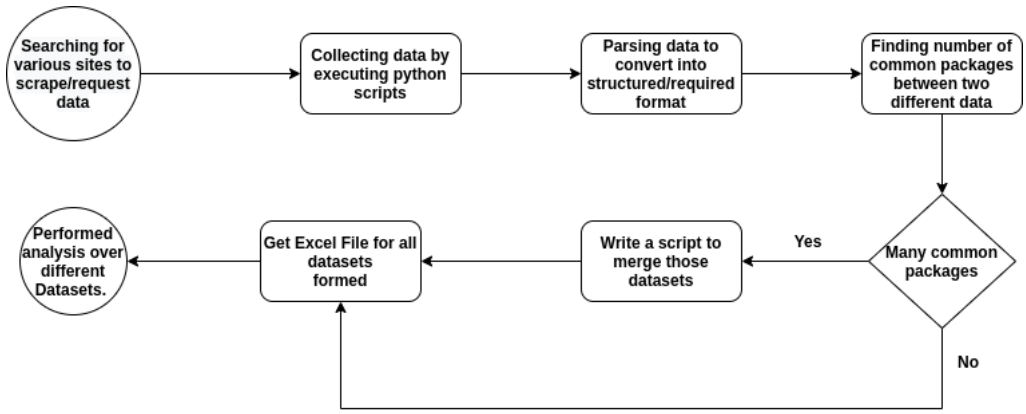
Column Name	Column Description	Example Entry
Package	Name of the package	wget
Maintainer	Maintainer of the package	Ubuntu Developers ubuntu-devel-discuss@lists.ubuntu.com
Homepage	Homepage URL	Wget Homepage URL
Description	Description of the package	retrieves files from the web Wget is a network utility to retrieve files from the web using HTTP(S) and FTP, the two most widely used internet protocols
Source	source package of the current package	eg for ipython is ipython itself
Conflicts	Conflicts of the current package	wget-ssl
Bugs	URL where bugs information is stored	Bugs information of wget
Installed-Size	Size of the package after it is installed	905 kB
APT-Sources	Source of the package	source URL
Recommends		ca-certificates
Replaces	If the software replaces any previous software	eg Tesseract-ocr replatces tesseract-ocr-data (<< 2)
Version	Version of the package	1.17.1-1ubuntu1.5
Download-Size	Download size of the package	299 kB
Suggests	Suggestions related to this package	Example for IPython package = ipython3-notebook, ipython3-qtconsole, python3-zmq (>= 2.1.11)
Original-Maintainer	The original maintainer of the package	Noël Köthe noel@debian.org
Task	Tasks related to the package	standard, mythbuntu-backend-slave, mythbuntu-backend-master, ubuntu-touch-core, ubuntu-touch, ubuntu-sdk
Supported	time for which the package is supported	5y
APT-Manual-Installed	variable to signify whether the manual is installed with the package or not	yes
Origin	The origin of the package	ubuntu
Breaks	Situation in which the current package	For example dmraid break on mdadm (<= 3.2.5-5ubuntu2) is installed on machine

	breaks	
Section	The section to which the package belongs to	web
Depends	The list of package on which the software depends	libc6 (>= 2.17), libidn11 (>= 1.13), libpcre3, libssl1.0.0 (>= 1.0.1), libuuid1 (>= 2.16), zlib1g (>= 1:1.1.4)
Priority	The priority of the package	standard

Architectural Design

We started off by browsing the websites which could provide us with the relevant information about the given domain. We gathered data from across multiple websites by applying techniques such as web scraping, requesting data through APIs, etc. The data was then parsed to get the information about the column names(features) of the data. The data collected was then converted to a format required by us for further processing by executing some python scripts. As mentioned previously, the data we collected was about packages from various package managers. So, if there were sufficient common packages between data collected from 2 different package managers, the data were merged by concatenating features that were different and by merging those which served a similar purpose(information-wise) by renaming them to some common column name. The data wasn't merged if the number of common packages was very less. Finally, an Excel sheet was made for every data for easy visualization purposes and different analyses were done to get meaningful insights about each of the datasets.

Below is the visual representation of the same.

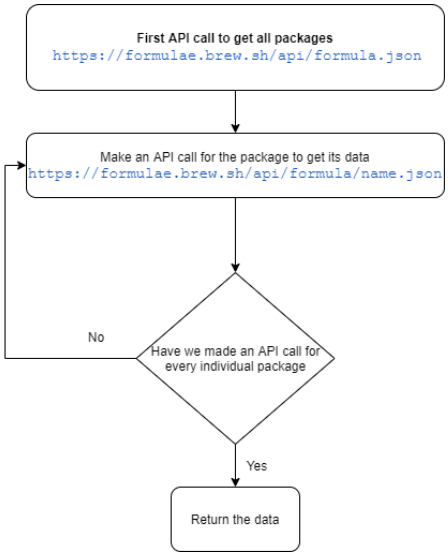


Data Collection

API Calls

Homebrew API

It is a very rich API that provides us with various ways to get the data. We have focused on a way that takes a little bit longer but at the same time gives us the ability to collect the largest amount of data i.e. the maximum number of features for each and every individual row.



Macports API

Macports has a website that lists all the software packages that are available `https://ports.macports.org/`. Since Macports is an open-source project we could look at the source code of the website. When opened the source code on Github we noticed that the MacPorts website has a well-documented publicly accessible API that powered the website. The API has a root at `https://ports.macports.org/api/v1/`. Through each request to this API, we could get data on about 50 software packages. We made almost 600 requests to get metadata of all the software packages available on the website. This was similar to the homebrew data we collected.

Web scraping

For data collection of the Chocolatey package manager, there was no publicly accessible API which we could use to collect data. On their official website `https://community.chocolatey.org/packages` there is a list of all the packages available. We attempted to scrape the website first in a static manner by using `beautifulsoup` library of python. However, we noticed some information on the page was dynamically generated (like the virus scan results) so we could not collect that information by using static web scraping. After this, we attempted dynamic web scraping using `Selenium`. The problem with this approach was that it seemed like some data was not loading properly after we collected the initial 1000 packages. This may be because the website had noticed unusual traffic from our IP address and started to not answer requests from it. We tried deploying our own `proxy` servers on `google cloud platform` to try to dodge the IP banning. This also had two pitfalls: First is the ethical concern of bypassing blocking and overloading the server with requests. This is unacceptable for a research project. The second is that introducing proxies slows down the process of sending requests and loading web pages significantly. Considering both of these aspects we dropped the idea of web scraping and looked into alternative methods.

Data collection from the command line

APT

Collection for `apt-get` packages was kicked off by getting the list of packages that apt-get provides, using command `apt list` for the same. The information about each of the packages was then consolidated by writing a python script that would communicate to the terminal and get the required details by the command `apt show PACKAGE_NAME`

Choco

After the failed web scraping attempt we almost gave up on `choco` as we had sunk a lot of time into it with no good results. As our last attempt, we tried getting all the list of packages by using `choco list` command and even get more detailed information by using `choco list --detailed` command. The information was not as detailed as the website but we considered this a good middle ground. The information was in the form of a plain text file of unstructured data so we had to now convert it into structured information which we cover in the next section.

Data Parsing and Cleaning

APT

The data we received was as shown below. This data was then converted into a dictionary with key as the feature and value as the description corresponding to that.

Further, all the unique keys across all packages were consolidated by loading the previous file. Then a .tsv file was made by iterating over all packages and appending the value corresponding to the key in the list if it existed for that package and with an empty string otherwise. This .tsv file was then loaded into excel to get the .xlsx file over which various analyses were performed.

```
Package: wget
Version: 1.20.3-1ubuntu1
Priority: standard
Section: web
Origin: Ubuntu
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Noël Köthe <noel@debian.org>
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Installed-Size: 1,016 kB
Depends: libc6 (>= 2.17), libidn2-0 (>= 0.6), libpcre2-8-0 (>= 10.32), libpsl5
Recommends: ca-certificates
Conflicts: wget-ssl
Homepage: https://www.gnu.org/software/wget/
Task: standard
Download-Size: 349 kB
APT-Manual-Installed: no
APT-Sources: http://in.archive.ubuntu.com/ubuntu focal/main amd64 Packages
Description: retrieves files from the web
 Wget is a network utility to retrieve files from the web
 using HTTP(S) and FTP, the two most widely used internet
 protocols. It works non-interactively, so it will work in
 the background, after having logged off. The program supports
 recursive retrieval of web-authoring pages as well as FTP
 sites -- you can use Wget to make mirrors of archives and
 home pages or to travel the web like a WWW robot.
 .
 Wget works particularly well with slow or unstable connections
 by continuing to retrieve a document until the document is fully
 downloaded. Re-getting files from where it left off works on
 servers (both HTTP and FTP) that support it. Both HTTP and FTP
 retrievals can be time stamped, so Wget can see if the remote
 file has changed since the last retrieval and automatically
 retrieve the new version if it has.
 .
```

```
{'APT-Sources': 'http://in.archive.ubuntu.com/ubuntu xenial/universe amd64 '
                'Packages',
 'Bugs': 'https://bugs.launchpad.net/ubuntu/+filebug',
 'Depends': '0ad-data (>= 0.0.20), 0ad-data (<= 0.0.20-1), 0ad-data-common (>= '
            '0.0.20), 0ad-data-common (<= 0.0.20-1), '
            'libboost-filesystem1.58.0, libc6 (>= 2.15), libcurl3-gnutls (>= '
            '7.16.2), libenet7, libgcc1 (>= 1:3.4), libgl1-mesa-glx | libgl1, '
            'libglooxl3v5, libicu55 (>= 55.1-1~), libminiupnpc10 (>= '
            '1.9.20140610), libnspr4 (>= 2:4.9.2), libnvtt2, libopenal1 (>= '
            '1.14), libpng12-0 (>= 1.2.13-4), libsdl2-2.0-0 (>= 2.0.4), '
            'libstdc++6 (>= 5.2), libvorbisfile3 (>= 1.1.2), libwxbase3.0-0v5 '
            '(>= 3.0.2+dfsg), libwxgtk3.0-0v5 (>= 3.0.2+dfsg), libx11-6, '
            'libxcursor1 (>> 1.1.2), libxml2 (>= 2.9.0), zlib1g (>= 1:1.2.0)',
 'Description': 'Real-time strategy game of ancient warfare 0 A.D. (pronounced '
                '"zero ey-dee") is a free, open-source, cross-platform '
                'real-time strategy (RTS) game of ancient warfare. In short, '
                'it is a historically-based war/economy game that allows '
                'players to relive or rewrite the history of Western '
                'civilizations, focusing on the years between 500 B.C. and 500 '
                'A.D. The project is highly ambitious, involving '
                'state-of-the-art 3D graphics, detailed artwork, sound, and a '
                'flexible and powerful custom-built game engine.',
 'Download-Size': '5,072 kB',
 'Homepage': 'http://play0ad.com/',
 'Installed-Size': '18.3 MB',
 'Maintainer': 'Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>',
 'Origin': 'Ubuntu',
 'Original-Maintainer': 'Debian Games Team '
                        '<pkg-games-devel@lists.alioth.debian.org>',
 'Package': '0ad',
 'Pre-Depends': 'dpkg (>= 1.15.6~)',
 'Priority': 'optional',
 'Section': 'universe/games',
 'Version': '0.0.20-1'}
```

Homebrew and Macports

The data was collected in `JSON` format. So we flattened the multi-level `json` dictionaries to a single level so that we can export the data as CSV and Excel files. This data was mostly cleaned and required very little to no processing.

Chocolatey

Choco data collected was very unstructured, as shown in the below figure. We structured the data using a rule-based approach, code given on Github.

Unstructured data:

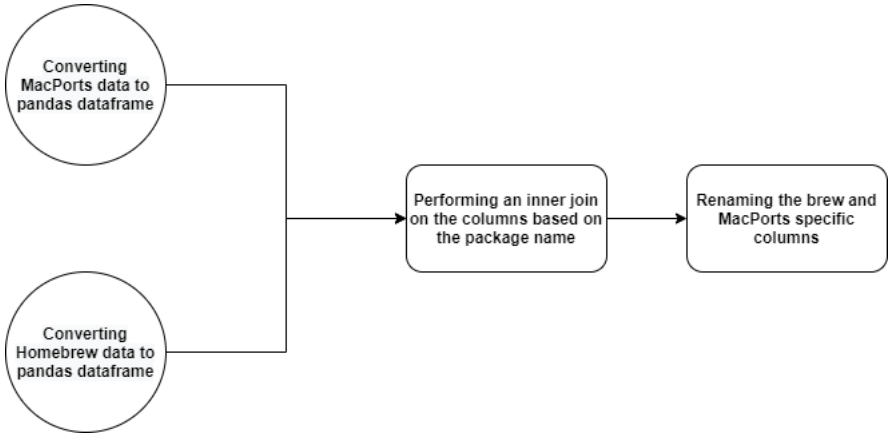
```
visualcpp-build-tools 15.0.26228.20170424 [Approved]
 Title: Visual C++ Build Tools 2017 | Published: 26-04-2017
 Package approved by AdmiringWorm on Apr 27 2017 14:53:34.
 Package testing status: Exempted on Apr 26 2017 14:16:42.
 Number of Downloads: 47184 | Downloads for this version: 32745
 Package url
 Chocolatey Package Source: n/a
 Package Checksum: '1MZ0d8T7M6xBTzzPz5R6VhR6v+74Ojyru4fF8MaNTOYRPlfJQrjS0fACQIuXZz/StmX8xmUKFkZBsl9ge9PlDg==' (SHA512)
 Tags: Microsoft Visual C++ Build Tools 2015 Compiler admin
 Software Site: https://blogs.msdn.microsoft.com/vcblog/2016/11/16/introducing-the-visual-studio-build-tools/
 Software License: https://msdn.microsoft.com/en-US/cc300389.aspx
 Documentation: https://docs.microsoft.com/en-us/cpp/
 Issues: https://connect.microsoft.com/VisualStudio
 Summary: Standalone compiler, libraries and scripts.
 Description: These tools allow you to build C++ libraries and applications targeting Windows desktop.
  They are the same tools that you find in Visual Studio 2017 in a scriptable standalone installer.
  Now you only need to download the tools you need to build C++ projects.
  ### Note
  This package installs MSBuild and Visual C++ tools. For MSBuild only see [microsoft-build-tools]
(https://chocolatey.org/packages/microsoft-build-tools) package.
  ### Package Specific

  #### Package Parameters
  The following package parameters can be set:

   * `/IncludeRequired` - install the MSBuild tools and required VC++ tools
   * `/IncludeRecommended` - install the MSBuild tools and recommended (default) VC++ tools
   * `/IncludeOptional` - install the MSBuild tools and all of the optional VC++ tools

  These parameters can be passed to the installer with the use of `--params`.
  For example: `--params "'/IncludeOptional'"`.
 Release Notes: https://www.visualstudio.com/en-us/news/releasenotes/vs2017-relnotes
```

Structured result:


```
[
  {
    "package_name": "windowsxappremover",
    "version": "1.02",
    "title": "Windows X App Remover",
    "published": "28-04-2017",
    "approved_by": "flcdrg",
    "approved_on": "Apr 29 2017 05:56:46.",
    "Package testing status": "Passing on Apr 29 2017 05:15:28.",
    "Number of Downloads": 1036,
    "Downloads for this version": 1036,
    "tags": [
      "windows",
      "privacy",
      "utility"
    ],
    "Chocolatey Package Source": "https://github.com/TakataSanshiro/Chocolatey-Packages
    "Package Checksum": "'x9IE7MK5DB9CPaziRCX0s4H/0Tr69bHKqy8b/6D1fbFf6p7hpSRT+I+1gFV1h
    "Software Site": "https://sourceforge.net/projects/windows8appremover/",
    "Software License": "https://www.gnu.org/licenses/gpl-3.0.en.html",
    "Software Source": "https://sourceforge.net/p/windows8appremover/code/ref/master/",
    "Documentation": "https://sourceforge.net/p/windows8appremover/wiki/Home/",
    "Issues": "https://sourceforge.net/p/windows8appremover/tickets/",
    "Summary": "Remove Modern UI Apps from Windows Installations or Images.",
    "Release Notes": "https://sourceforge.net/p/windows8appremover/wiki/Changelog/",
    "description": "Remove Modern UI (Metro) Apps from Windows! With the Windows X App
  },
```


Merging Data from different Tables

Since we collected data from four different package managers we were looking to combine them to make a much more rich and extensive table. After looking at all the 6 pairings of these four tables only the merge between MacPorts and Homebrew data makes sense since the number of common packages between them exceeds 2500 which is greater than 50% of the total number of packages we got from the Brew Package manager. Combinations between other pairs had very few packages in common and we have provided them as separate tables only.



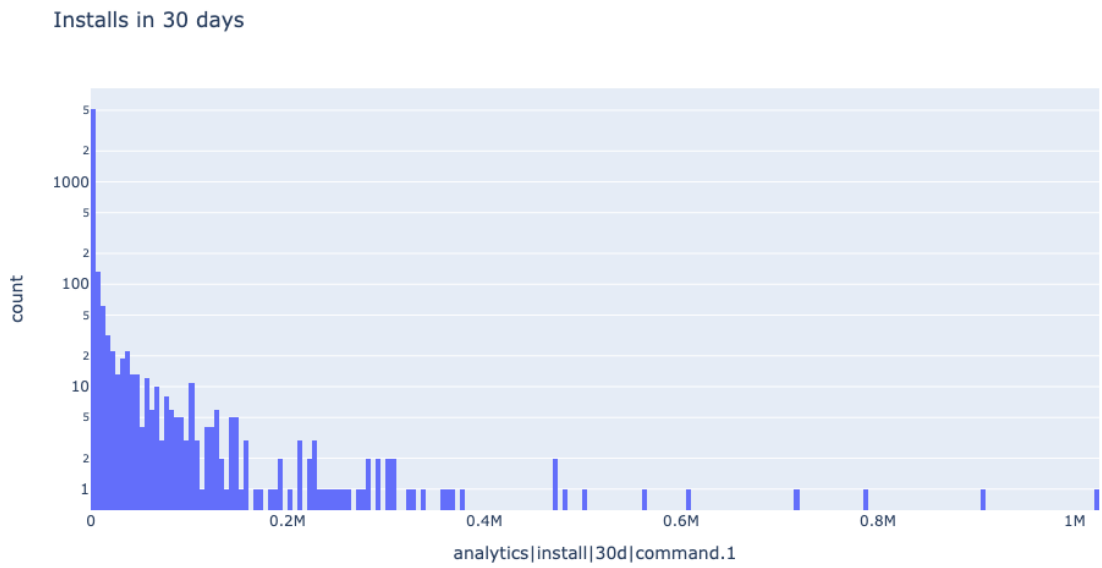
Analysis

For homebrew, we have divided the analysis into two sections, CLI and GUI software. Overall CLI software has much more installs as GUI software generally depends on CLI software but for a layman, GUI software is much more relevant as that is what most people use.

Overall we notice power-law distribution in the histogram of installs vs a number of packages. This confirms our beliefs that a very small number of the packages make up most of the download

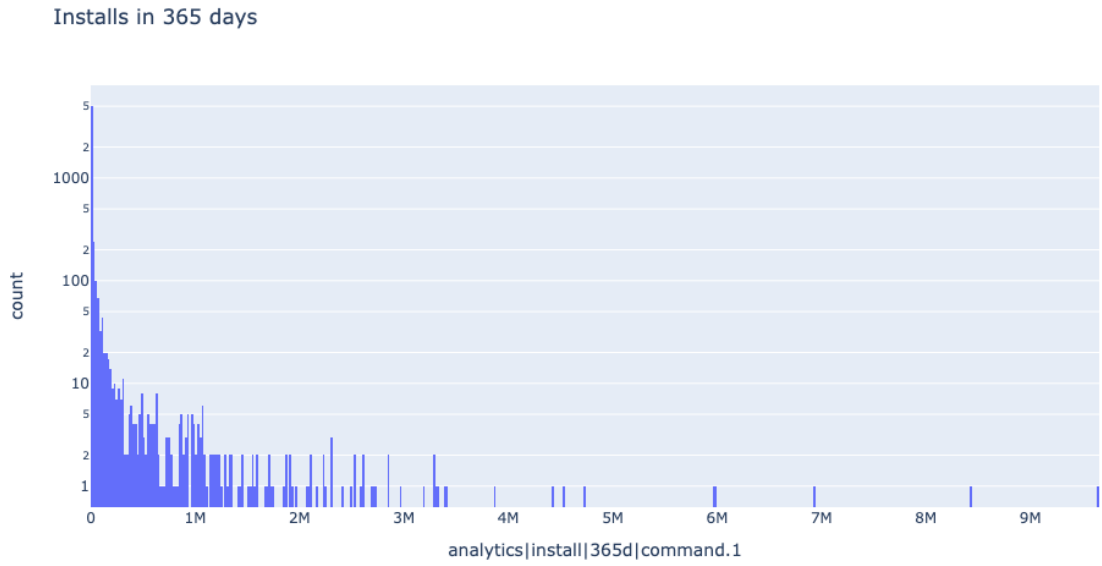
CLI software

The installs of different software have a very skewed distribution and seem to follow the power law. The maximum number of installs in 30 days was “openssl@1.1” with 1023040 installs. We notice python3.9 is the most downloaded programming language. On the other hand, 5000+ software had less than 5000 installs in the same time frame.

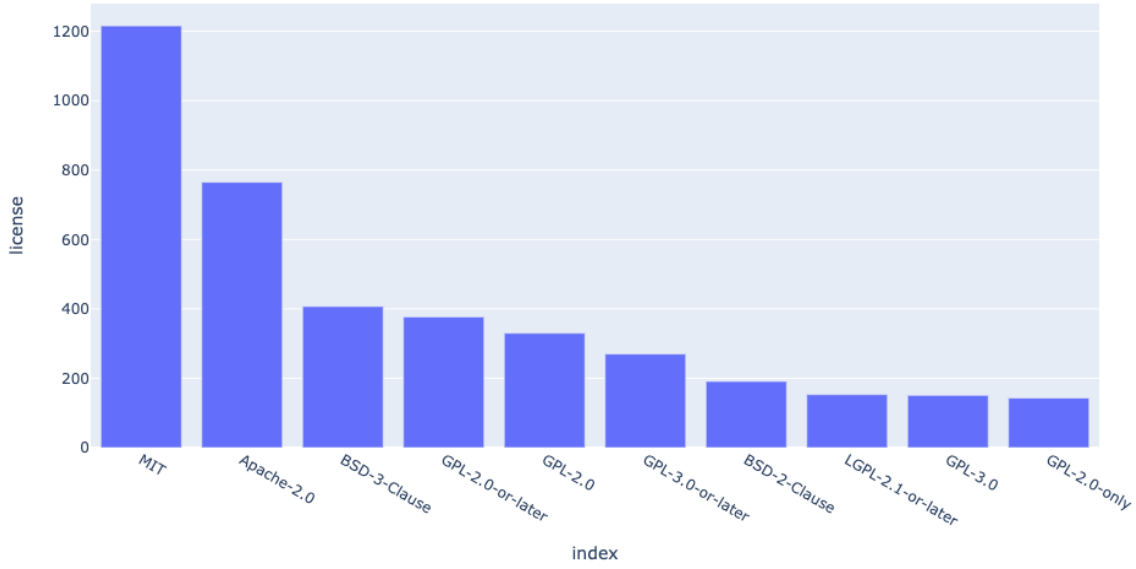


	name	analytics|install|30d|command.1
5052	openssl@1.1	1023040.0
2150	gdbm	909762.0
6041	readline	786567.0
5870	python@3.9	717857.0
695	ca-certificates	608457.0

A similar trend was noticed in installs of software over the past 1 year. The maximum number of installs in 1 year was again “openssl@1.1” with 9659770 installs. Now we also have the famous database management software “sqlite” on the top 5 list. The top list does not change that much with openssl, python, readline and gdbm still showing up:



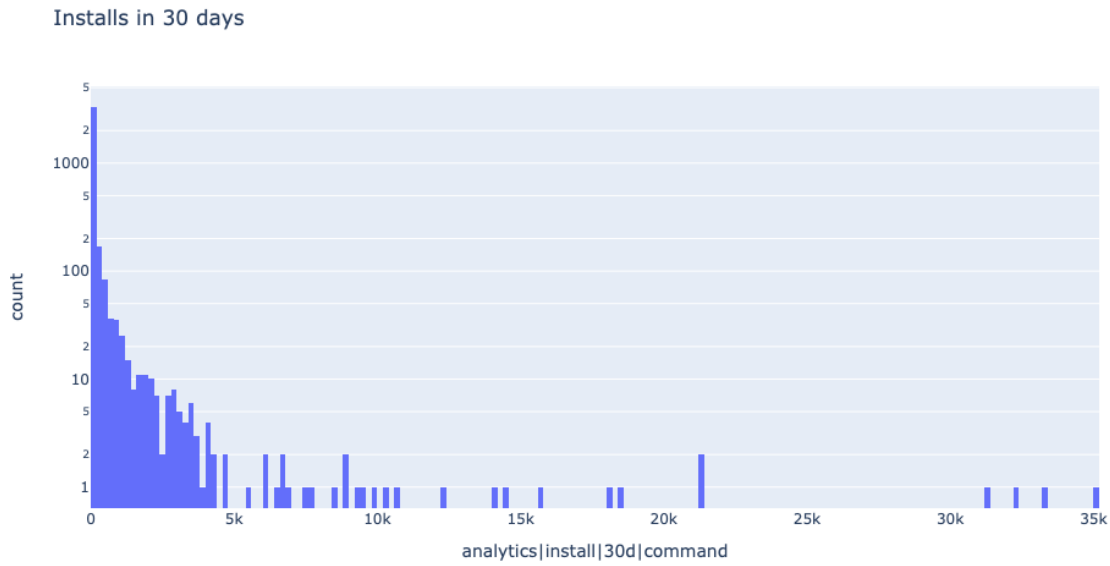
	name	analytics|install|90d|command.1
5052	openssl@1.1	2734837.0
5870	python@3.9	1988212.0
2150	gdbm	1962645.0
6679	sqlite	1542628.0
6041	readline	1520552.0



Software licenses play a crucial role for any software, empowering users to modify or re-distribute the software if the license allows. We encountered over 200 unique licenses that different software had in our data collection. The top 10 software licenses from our database are shown in the bar plot below. The most common license was the MIT license which was present in over 1200 software.

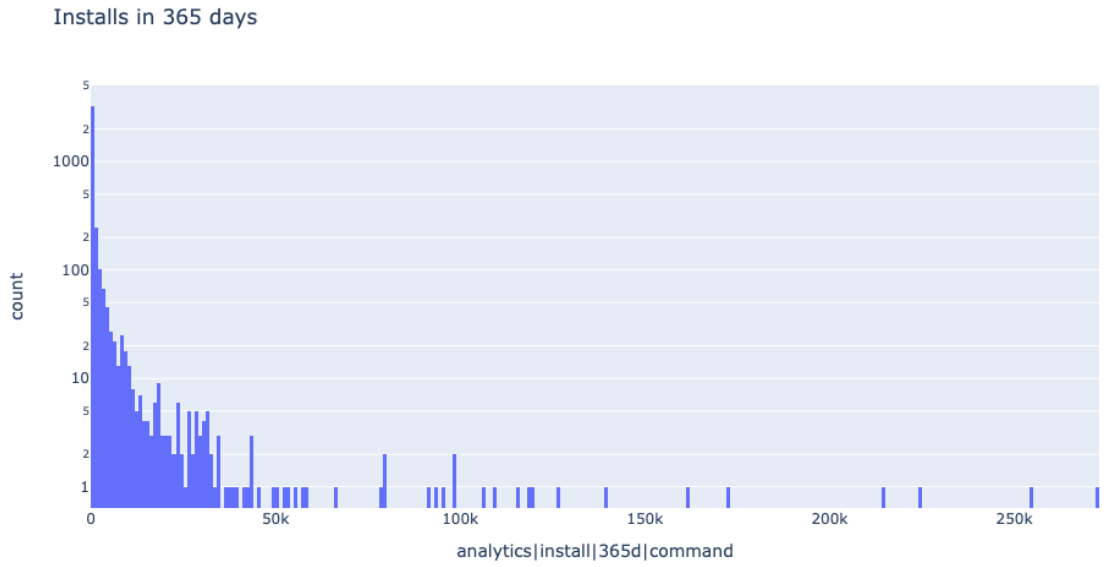
GUI software (Casks)

In the last 30 days for GUI software, we notice `Iterm2`, `Visual-studio-code` and `docker` appearing. This shows that the skewness of our dataset is towards a developer population as package managers tend to be used most by them. We also see general-purpose software like `google-chrome` and `microsoft-edge` which are web browsers - one of the most important software on a computer.



	full_token	analytics|install|30d|command
1531	iterm2	35125.0
2014	microsoft-edge	33305.0
3676	visual-studio-code	32379.0
1275	google-chrome	31324.0
817	docker	21311.0

In the last 1 year, we don't observe many changes to the list compared to the last 30-day result. `Android-platform-tools` come in at number 5 which increases the developer tools dominance even more.



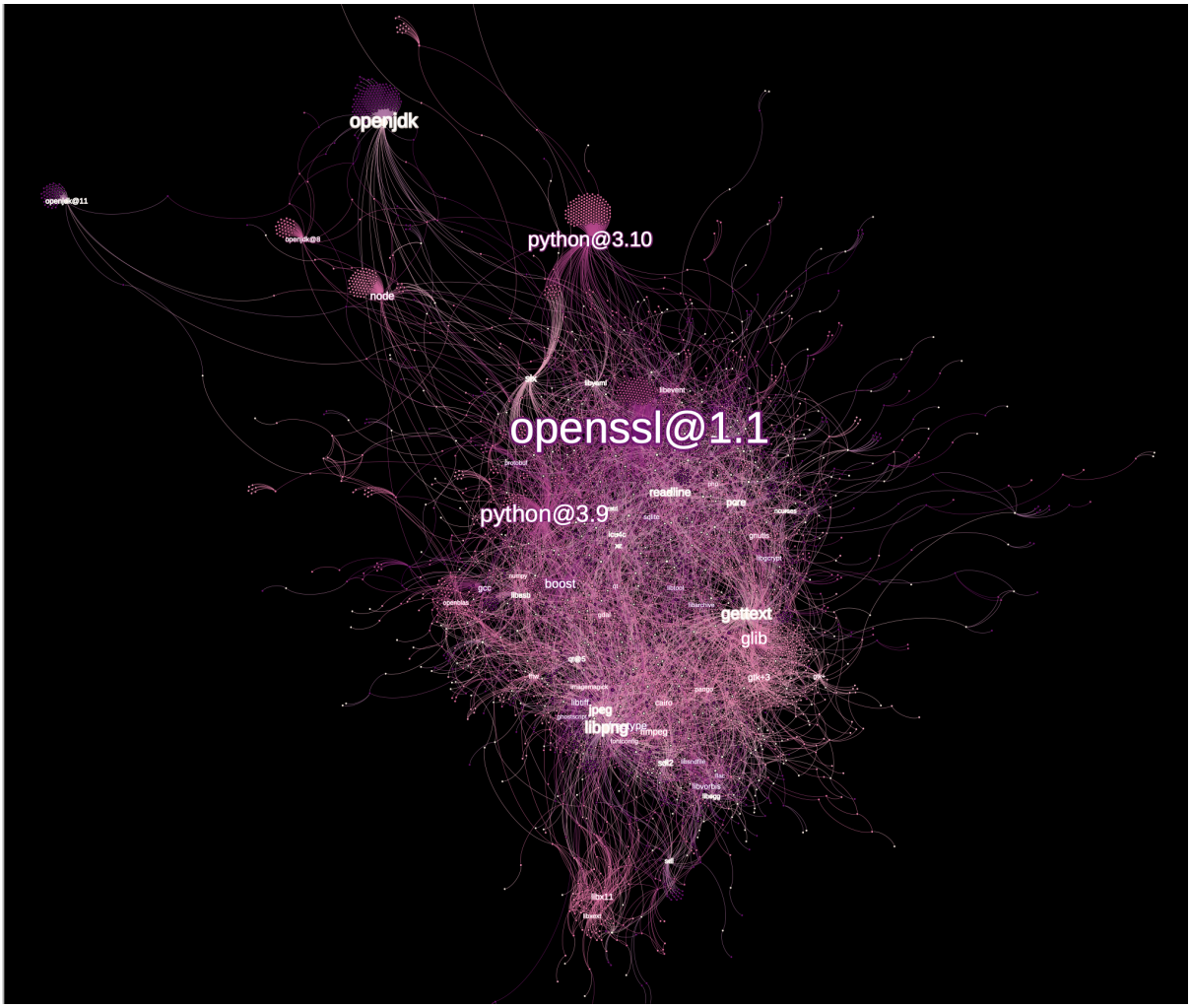
	full_token	analytics|install|365d|command
1531	iterm2	272411
3676	visual-studio-code	254939
817	docker	224279
1275	google-chrome	214342
133	android-platform-tools	172914

Dependency Graph for software on Homebrew.

We constructed a directed graph that has an edge from `a` to `b` if software `a` depends on software `b`. Then we plotted the largest weakly connected component of the graph using the open-source software `Gephi.`

Graph properties:

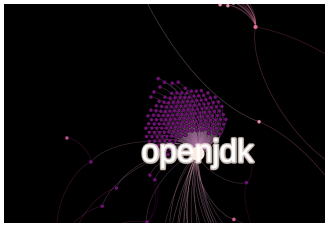
- Node size is proportional to the degree
- Label size is proportional to the degree
- Color is proportional to Closeness Centrality
- The clustering algorithm is `force atlas`
- Node labels are shown for nodes with `degree > 40`

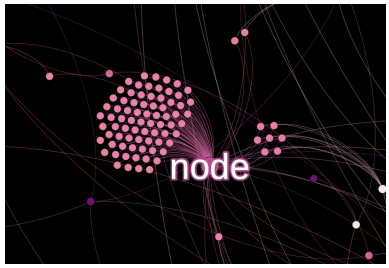


These are some of the properties of the entire graph:

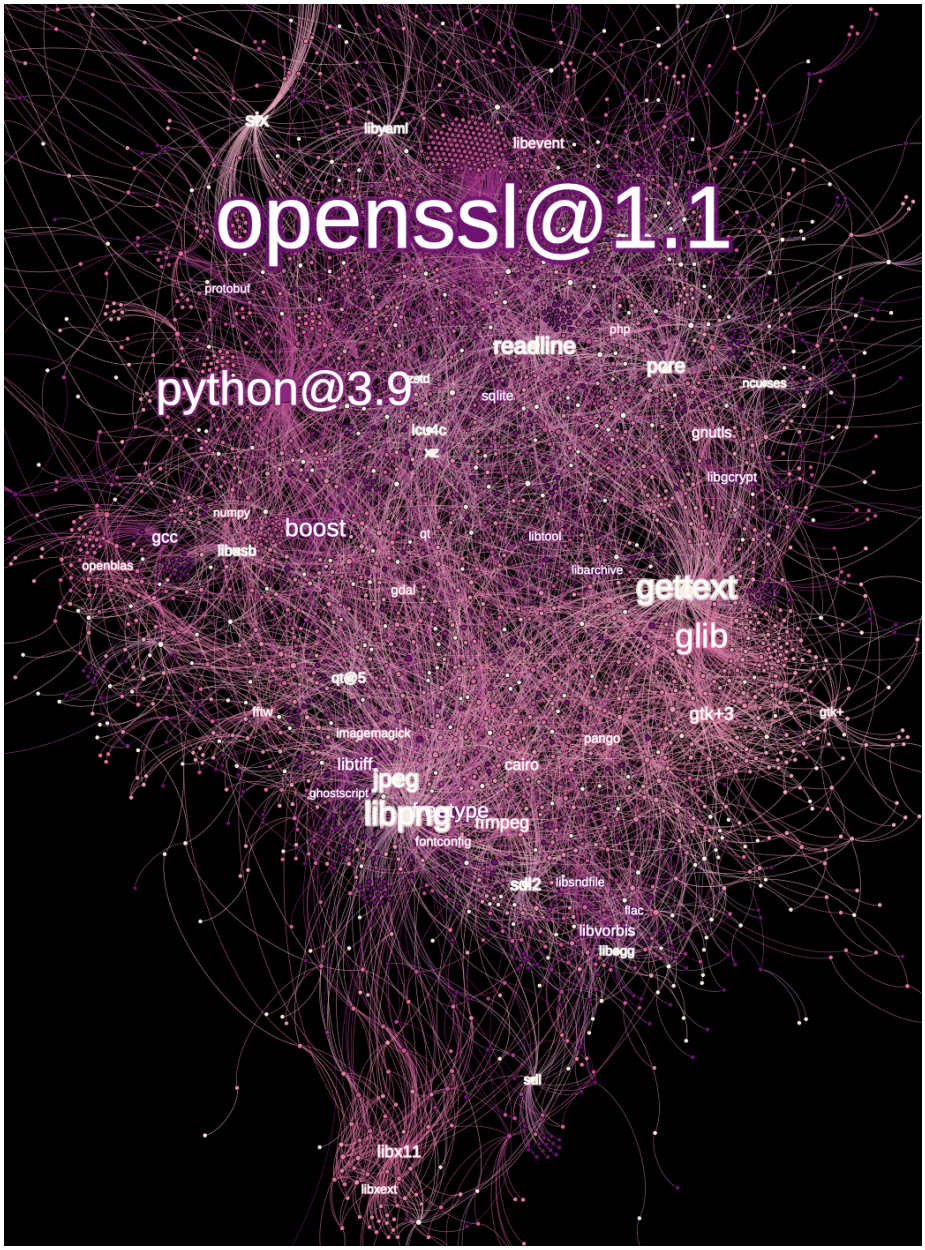
Type: Directed Graph
Number of nodes: 5862
Number of edges: 7901
Average in degree: 1.3478
Average out degree: 1.3478
Network density: 0.00022996647396750992
2321 weakly connected components
5861 strongly connected components

We observe some interesting things from the clusters in the graph:



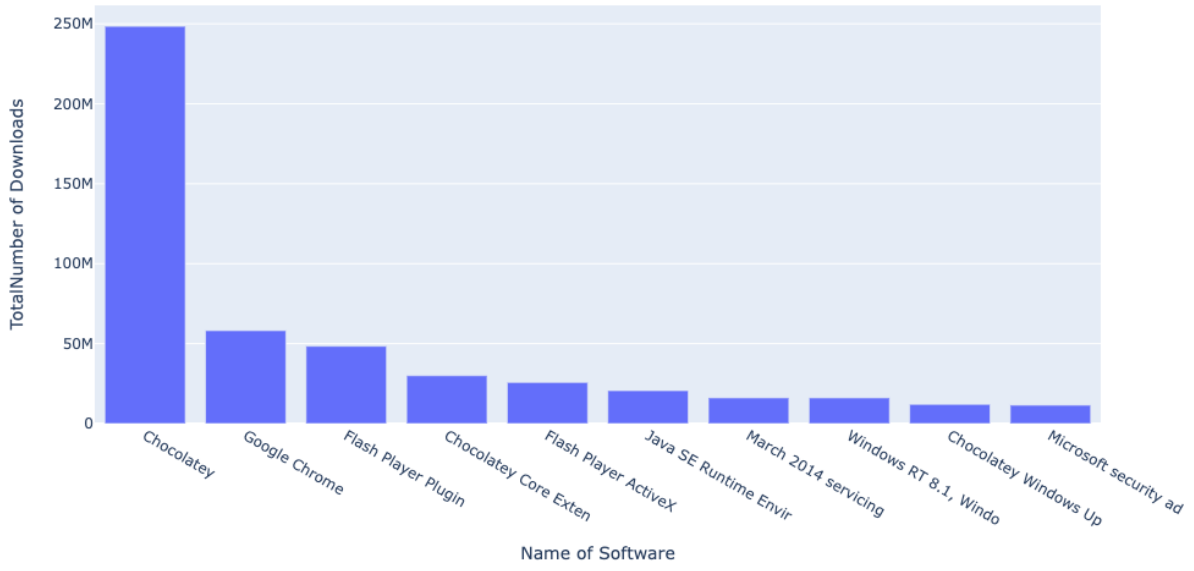


- Clusters around
  - `OpenJDK`, a Java framework, shows the relevance of Java today.
  - `Python3.10` is the latest version of python, a programming language every programmer should know.
  - `Node`, a javascript framework that allows JS to be run server-side, has overtaken server-side development in the last decade.
- The central island of the graph has its biggest node as openssl followed by python3.9. `Openssl` and `Python` is also the most downloaded software of Homebrew so it makes sense that a lot of software depends on it. The central island has tons of low-level software libraries mostly written in C/C++ like `libpng`, `glib`, `boost`, etc. These form the backbone of any modern-day software stack.

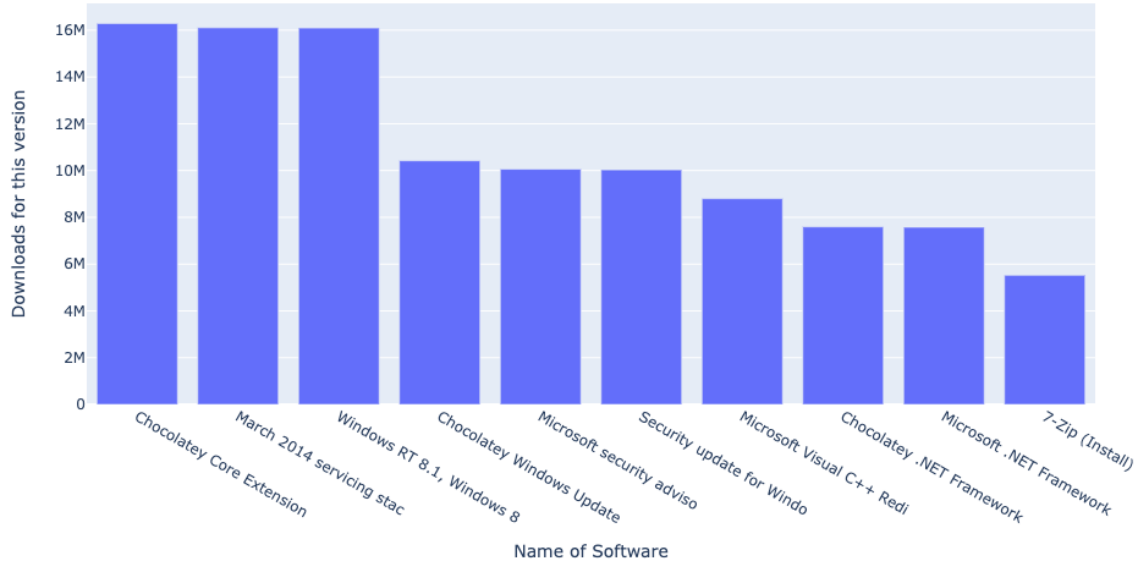


Windows is the most popular OS in the world and chocolatey is the most popular package manager on windows. We have done some analysis on the software present on chocolatey.

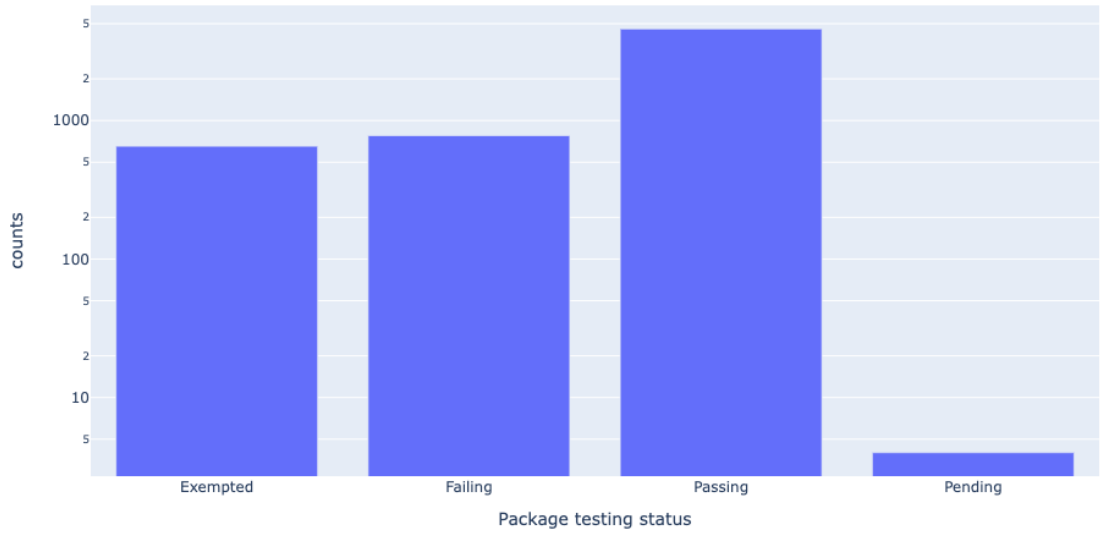
First, the most downloaded software is `Chocolatey` itself, which is the GUI version of `choco` the command-line tool. The next most common software is `google-chrome` with over 50 M installs and no other browser is on the top 10 list. This shows the dominant role of `google-chrome` in the web browser world. Next, we see other popular software like `Flash`, and `Java runtime.`



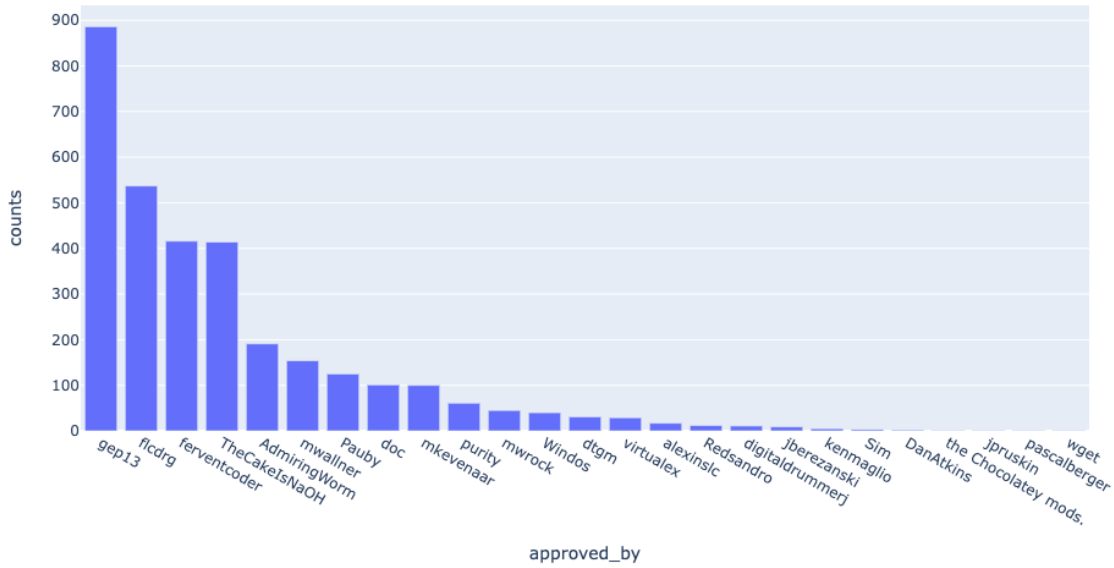
We also analyzed installs for the current version. In this, we notice very different results as the downloads depend on when the version was released. Windows update scores high as their version never changes.



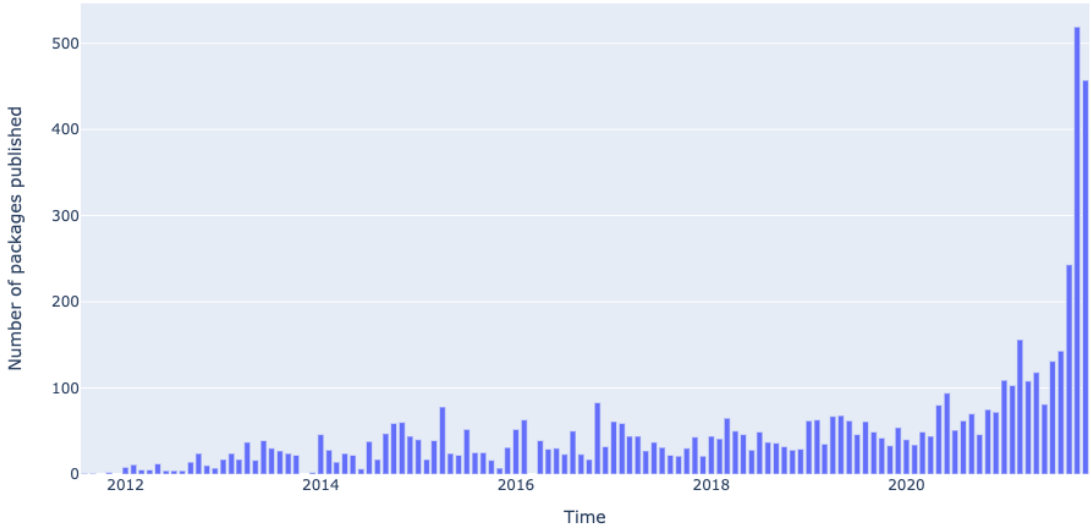
Next, we analyzed the status of the different software. We use a log scale as 5000+ software were passing. But again it was unexpected to see almost 1000 software failing their tests, and another 1000 Exempted. But very few softwares have pending status which goes on to suggest that the testing cycle is very fast and efficient.



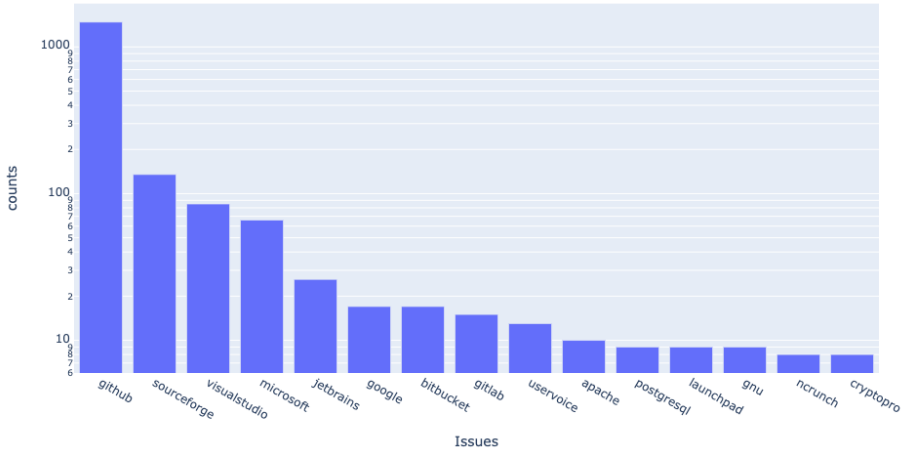
Next, we analyze the chocolatey admins who approve the packages (and make sure they are not malicious). The most active admin is `gep13` who has approved over 800+ packages followed by `flcdrg`.



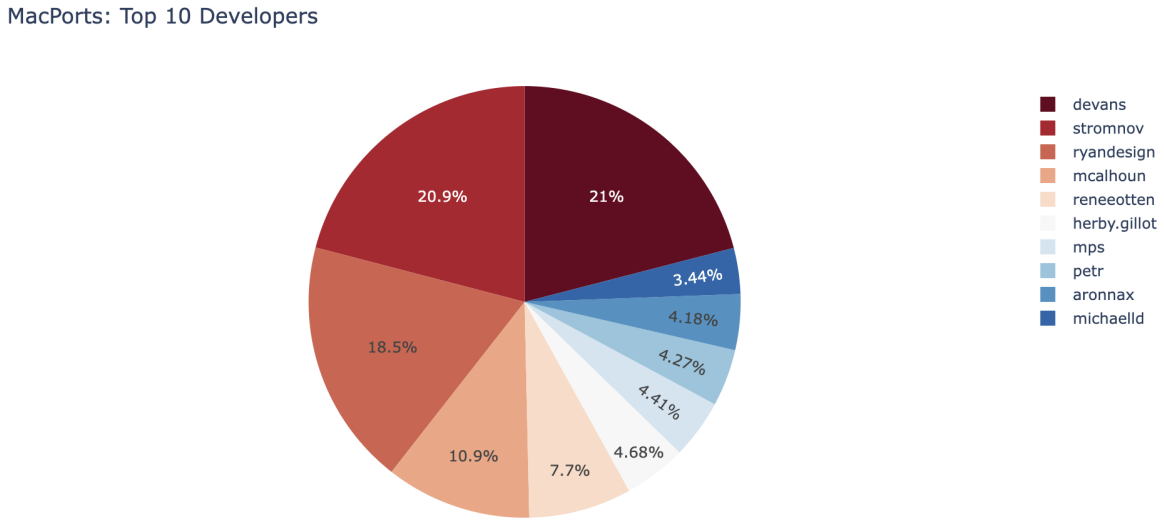
Next, we analyze when the packages were published on the choco ecosystem. In this plot, each bar represents a month. We notice chocolatey has exploded in popularity over the last 3 months with over 500 packages being published but it had a very modest start in 2012 with less than 20 packages being published every month.



We explored where issues for a given package are tracked. We again use log scale as `Github` has an order of magnitude higher packages than the next place which is `sourceforge`. This confirms that `Github` has captured a great chunk of the developer market for the maintenance of software packages.



Macports has a field called maintainers. These are the people responsible for developing the software and making sure the package is bug-free. Out of the top 10 maintainers `devans` maintained over 20% of the packages.



We also analyzed the description of the packages. First, we cleaned the data to get rid of punctuations and stopwords. Here is the word cloud for MacPorts

We see `Python`, `PHP`, `C` , `Perl` stand out which are popular languages. Also, words like `framework`, `client` and `extension` which are commonly used in the software world are present.

References

- For Data collection:
  - Homebrew API Documentation: Documentation to understand how to scrape data from the homebrew API

    - Chocolatey Documentation : To understand what this particular package manager is and what all information can it provide to us
    - Launchpad API : We considered using launchpad API to collect data for various computer software
- For data cleaning:
    - Data cleaning techniques - We looked at various techniques present in this blog for data cleaning
    - https://github.com/kjam/data-cleaning-101 - The repository contains several notebooks which have python code for data cleaning. We looked at it for references.
- For analysis:
    - Python tools for big data analytics - We looked at some of the analysis tools provided in this paper.
    - Closeness centrality - For coloring the dependency graph