Team_id: c_uragiw

Aditya Govil, Fahmid Fahid, Cameron Nelson, Gautam Worah
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{agovil, ffahid, cenelso3, gworah}@ncsu.edu

***Abstract*—Visualization is a key part of our modern day lives. We need visualizations to understand everything from social media growth to stock market fluctuations. In Python, Matplotlib [1] is arguably the most popular visualization library for over a decade. But there are other alternatives. Plotly [2], for example, besides offering visualization, has many other relevant features. According to our understanding, there is no study done to compare these two libraries. So, we did some rigorous experiments to understand the code-base, the community and the performance of these two libraries and found that although the new functionalities of Plotly are relevant today, they come at a cost of resource usage and complexity. On the other hand, Matplotlib is stable, faster and simpler. We recommend that if there are no restrictions in resources, low risk factors and greater flexibility for learning, then one should use Plotly. Otherwise, Matplotlib is the ideal choice.**

I. Overview

Matplotlib [1] is an object oriented Python library that generates basic, fine-grained 2D visualizations. It is considered one of the oldest and most widely used Python libraries due to its accuracy and quickness. However, to produce modern day graphics, an extensive amount of code is needed. To remedy this disadvantage, we looked into an alternative library called Plotly.

Plotly [2] is an interactive Python library that has similar functionalties to Matplotlib, but with additional features such as 3D visualizations, easy browser integration, and high interactivity. It is also cross-language compatible. Having all these, it is likely that Plotly is more complex and resource heavy.

Due to these reasons, a developer or firm may face a dilemma while selecting a tool for visualization in Python. As per our knowledge, no study has been done to compare these two libraries. For example, we do not know which library has better render time or which library has better open source community support. Comparing these libraries might give us better insight into their specific usage for developers as well as large scale companies.

With these ideas in mind, we tried to ask the following questions in our project:

- **RQ1: Static Analysis** Can we compare the source code structure and compare the reliability of these two libraries?
- **RQ2: Performance** Can we compare the performance, both in build time and in render time and understand the resource usage?
- **RQ3: Community Interaction** Can we compare the open-source community behind them and understand the responsiveness and stability?

To answer these questions, we used several open-source tools and home-grown scripts. The tools we used and their outputs are listed in Table I and our results are on GitHub respectively.

II. Approach

A. RQ1: Static Analysis

Understanding the complexity and documentation practices of the two libraries give us insight into the readability of the libraries. These, in turn, show how reliable a library can be towards future projects and for a developer. To understand this, we used Pylint [3] on Matplotlib and Plotly's GitHub repositories to collect data on each library's project size, source code documentation, bug detections and code smells. We then reported the results in Table III. Code Breakdown and Documentation lists the project size of each library by the number of modules, classes, methods and functions they have as well as a calculated documented percentage of each of them. Bug/Code Smells lists a series of warnings Pylint detects for each library. These series of warnings include programming standard violations (Convention), bad code smells (Refactor), Python specific problems (Warning) and probable bugs in the source code (Error).

B. RQ2: Performance

By comparing the render time and build time of the two libraries, we can imply the CPU usage in general. These, in turn, shows the cost of using these libraries and can give a better insight into their usability for large scale projects. To understand the render time of each library, we have implemented our home-grown scripts. We tried to draw two different types of graph, namely a **Sine Curve** and **6 Bar Charts**. A Sine Curve takes higher CPU resource to calculate the curve as well as to visualize them. On the other hand, Bar Charts are simpler to produce and render. We tried different screen size and did 10 iteration for each type of graph and calculated the run-time. Then we measured the median value to find out the average run-time for each type of graphsRender Time Source Code Links.

To understand the build time of each library, we have used an open source continuous integration tool called Travis CI [4]. To get the build time of the two libraries, we integrated Travis

CI tool with them. Since it needs permission from the owner of these repositories, we created dummy repositoriesBuild Time Dummy Repository Links with the code base of these libraries. Then we measured the mean value of build time for each library.

C. RQ3: Community Interaction

To determine how the community is interacting with each of the libraries, we plotted the time series graph of code commits. We also analyzed the median of issue resolution time and code contribution percentage per member. Libraries which are heavily dependent on a few members of the community are less likely to be stable because in case those members shift away from the project, those libraries will stagnate and their growth will stall. An analysis of the amount of code added per year is also done to determine the growth pattern of a library over time. Sharp inclines or edges signify that the library has been split or it has been merged with some other repository.

Half life of code is another useful metric that is used to calculate the percentage of code that is replaced or decayed every year (in this case 50%). By calculating the amount of code being replaced per year we can estimate the code quality and structure. A repository that has a gentle graph (for example the Linux kernel or the Git language repository) is stable and has fewer API changes. This means that older systems are less likely to break and the overall repository is well organized.

III. Results and Analysis

RQ1: Can we compare the source code structure and find reliability of these libraries?

Based on the results of Pylint, Matplotlib had 120 modules with an average documentation of 71% while Plotly had 937 modules with an average documentation of 50%. This means that Matplotlib is favorably relied on towards developers since its source code structure is less complex and heavily documented. However, Pylint detected $2,862$ Warnings and 341 Errors in Matplotlib compared to 641 Warnings and 95 Errors in Plotly. Despite Matplotlib being well documented, it contains a large number of bugs and code smells that could affect Matplotlib later on throughout development if they are not resolved soon. Detail results can be **found here** and **here**.

Judging from the Static Analysis perspective, Plotly is more stable than Matplotlib as it has less code smells. On the other hand, Matplotlib has less module and well documented. A developer might find it easy to work with Matplotlib while Plotly might give more reliability to the end product.

RQ2: Can we compare the source code structure and compare the reliability of these two libraries?

According to our study from our home-grown script, for each screen size and graph type, Plotly takes significantly more time to render than Matplotlib. For example, to render a sine curve on a $1920x1440$ screen, Matplotlib takes about 0.0189 seconds while Plotly takes 0.7333 seconds. Detailed results can be **found here**. From the results that we collected from Travis CI, it took 24 minutes and 39 seconds to build Matplotlib and 23 minutes and 23 seconds to build Plotly. In short, both the libraries took almost similar time to build for different versions of python and on different operating system environments.

So, if resource scarcity is something a developer is concerned with, one should choose Matplotlib over Plotly.

RQ3: Can we compare the open-source community behind them and understand the responsiveness and stability?

The code repository analysis from Figure 1 shows that Matplotlib has a stable growth curve after the year 2010. A large percentage of code that was added in the year 2005 is still a part of the codebase. On the other hand, Plotly has added about 80% new code in the year 2018 alone, as visible from Figure 2. This shows that Matplotlib is stable and robust in terms of code structure as the original code was well designed to accommodate future changes. Plotly on the other hand shows a sharp rise in the number of lines of code added. These kind of changes are not suitable for an enterprise company using this library in its products.

An analysis of the contributor activityContributor Activity Github Link of the libraries shows that Matplotlib has about 6 core contributors and Plotly has 2 core contributors. The small number of core contributors to Plotly make it extremely fragile and risk prone. It is also observed from Table II that the issue resolution time for Matplotlib (17 days) is significantly more than that of Plotly (3 days). Matplotlib has significantly higher issue resolution time because it has a large number of older issues ($>$ 5 years) that increase its median.

On generating the code decay plotsMatplotlib and Plotly Code Decay graphs for both the libraries it is visible that Matplotlib has a much more stable curve than that of Plotly. More than 80% code of Plotly was replaced within a span of 6 months whereas it took more than 3 years for 50% of the code to be replaced in Matplotlib.

IV. Conclusions

According to our results, Matplotlib is more stable, faster and simpler than Plotly. Moreover, the open-source community of Matplotlib is more responsive and from our half-life study, we can say that the aging of Matplotlib is gradual. Despite all these factors, Plotly offers greater control for complex visualizations and is highly customizable for web integrations as compared to Matplotlib.

So, if the development has lesser risk factors, has enough time for learning and is not bounded by resource usage, Plotly seems like a better choice, because of its richer features. But if the development has high risk factors, or there are time restrictions or resource scarcities, Matplotlib is still a safe and viable choice.

V. Appendix

TABLE I
LIST OF ALL THE TOOLS THAT WERE USED IN THIS PROJECT

Name	Description	Usage
Pylint	Static Analysis Tool for Python	Understand the source code structure, documentation, project size and code smells
Travis CI	Integration Tool	Understand the integration and the build time of the libraries
Homegrown Script	-	Understand the rendering runtime for different screen sizes and shapes
Gitlab	Github API	Understand community feedback, issue resolve time, library half life etc.

TABLE II
DETAILS OF REPOSITORY ISSUE ANALYSIS

	Median Issue Resolution Time	**Percentage of Open Issues**
Matplotlib	17 days	23%
Plotly	3 days	12%

TABLE III
PYLINT ANALYSIS

	Code breakdown and Documentation			Bugs/Code smells	
	type	number	%documented	type	number
Matplotlib	module	120	63.33	convention	8357
	class	701	73.75	refactor	1475
	method	4440	87.68	warning	2862
	function	801	61.30	error	341
Plotly	module	937	4.27	convention	21966
	class	703	9.39	refactor	3905
	method	14232	99.50	warning	641
	function	368	86.68	error	95

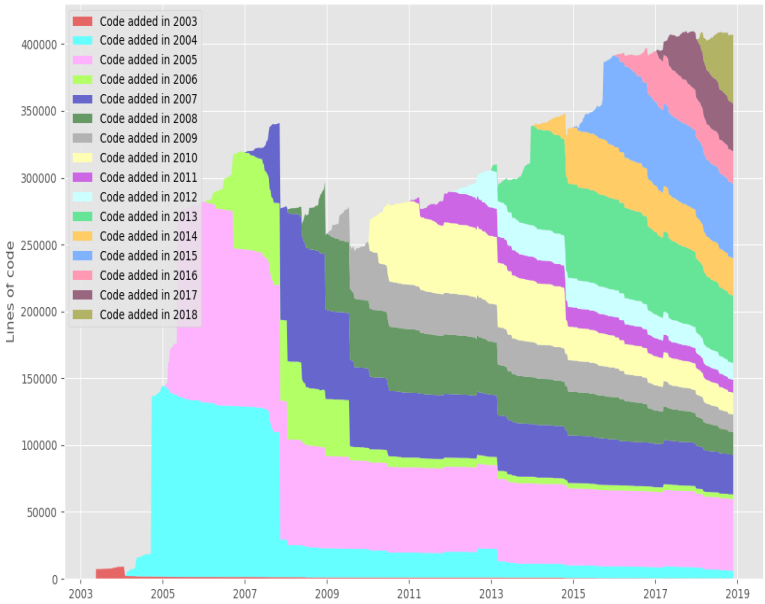

Fig. 1. Code growth for Matplotlib

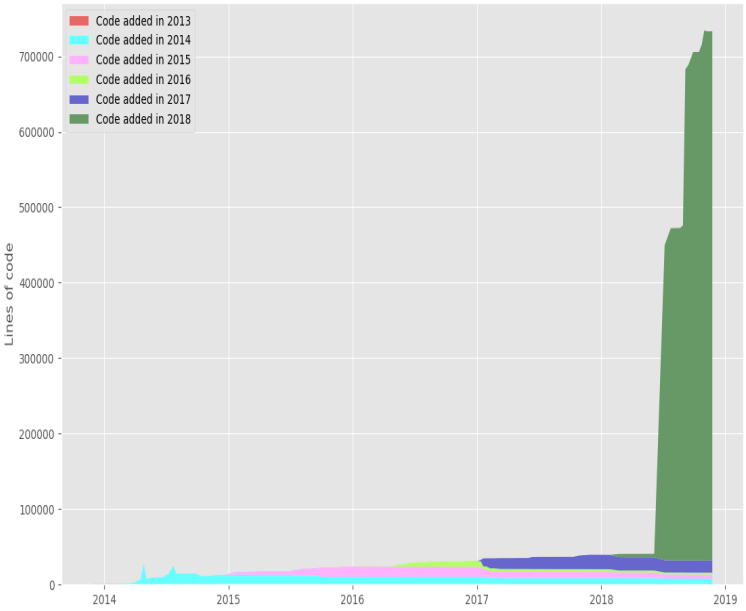

Fig. 2. Code growth for Plotly

References

[1] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[2] C. Sievert, C. Parmer, T. Hocking, S. Chamberlain, K. Ram, M. Corvellec, and P. Despouy, "plotly: Create interactive web graphics via plotlys javascript graphing library [software]," 2016.

[3] S. Thenault *et al.*, "Pylintcode analysis for python," *URL https://www. pylint. org*, 2001.

[4] Travis CI, GMBH. Travis CI. [Online]. Available: https://www.travis-ci. com