Analyzing the Polyglot Notebooks Project

# Project Overview

Any software development project has intrinsic data associated with it. The aim of this research effort is to examine ways of getting that data out of a software development project and visualized in an intuitive manner that can help others easily understand the project in question.

Previously, I have successfully visualized code using Python code in a Jupyter Notebook environment to build interactive charts for others to consume (Eland, Visualizing Code, 2022). However, this approach took a lot of manual effort among multiple files, was very undocumented, and was not accessible to those without a programming background.

Specifically, I want to prove that I can run a script to extract historical information about git repositories into CSV files and then analyze those files in Tableau to answer critical questions about the project and its trajectory in a way that can tell a compelling story to the community.

*Note: if you are unfamiliar with git repositories or source files in general, the Data Sources section below outlines their usage.*

As a demonstration of this, this report uses that methodology to analyze and describe the open-source “Polyglot Notebooks”. The results of this analysis tell the story of a vibrant but stable project led by several different key contributors and a large number of occasional contributors.

*Note: my project proposal selected the ML .NET repository instead of Polyglot Notebooks. However, in analysis I discovered the ML .NET repository was slightly corrupt and missing some historical data and would not make a good target for analysis, so I switched to the Polyglot Notebooks codebase instead.*

## Polyglot Notebooks Project Overview

Polyglot Notebooks, formerly known as .NET Interactive, is an open-source project initiated by Microsoft in an attempt to bring the power of dotnet languages such as C# and F# to the Jupyter Notebooks environment by creating a custom kernel that can run code written in a variety of languages including C#, F#, JavaScript, Powershell, and SQL. This is accomplished via an extension to Visual Studio Code (Regio, 2022).

Jupyter Notebooks, on the other hand, is an open-source way of mixing together Python or R and markdown documentation into notebook files that can execute cells of code and display the results of their operations. Effectively, Polyglot Notebooks aims to expand Jupyter Notebooks to serve additional programming languages.

Polyglot Notebooks is still in preview under active development by Microsoft and members of the community and had its first public release in April of 2021. Its source code is publicly available at <https://GitHub.com/dotnet/interactive> (dotnet).

*Full disclosure: I have not worked on the Polyglot Notebooks code, but I do interact with their team at Microsoft, have written on the topic before professionally* (Eland, Interactive C# with Polyglot Notebooks, 2022)*, and am giving a conference talk on the project in early May.*

# Datasets

This analysis task does not use any pre-made dataset. Instead, this project uses custom Python scripts to treat the git version control system and the files on disk as data sources. These scripts extract data from these sources into multiple CSV files.

This analysis work is accomplished by a set of Python scripts I developed for this project that plug into a Jupyter Notebook I created for this project. This code is not the focus of the project, but it can be found at <https://github.com/IntegerMan/DATA605FinalProject>.

For details of how to run this project, see the readme.md file for documentation and the Gather.ipynb Jupyter Notebook to execute the code. Note that the code requires that you have cloned a repository of interest onto your hard drive already.

## Getting data from git Repositories

Any git repository consists of one or more commits. Each commit has an author, a date and time, a unique hash code, a message, and modifies one or more files. These modifications can either be insertions or deletions and often are a combination of the two.

My theory was that I could access the git repository information from any git repository using pre-made libraries and then structure that data in such a way that I could export it to a CSV file for further analysis (Eland, Extracting git repository data with PyDriller, 2023).

### Commit Data

To do this, I used the open-source PyDriller library to read commit information from a local repository (Davide, n.d.). I then transformed the data into an appropriate shape using Python and the Pandas library and saved it to CSV as **Commits.csv**.

Table

Description automatically generated

Because software projects take a long time and people tend to change computers periodically during these projects, sometimes commits come in from the same author, but with a slightly different name or E-Mail address.

I handled this by manually coding a list of aliases for authors I observed had multiple entries. My code then interpreted commits from these aliases as commits from the base author. This allowed me to fully attribute code to these individuals.

I also looked up each author and coded in their country, state / city, and region if I could determine it by looking at their GitHub profile, LinkedIn profile, or Twitter profile. This gave me a rough idea as to where the team was geographically and let me include it in the dataset.

### File Commit Data

Once I had a list of commits, I was able to look at the individual files in that commit and make separate records in a new **FileCommits.csv** file that contained one row per commit per file. This allowed me to get more granular analysis later on of the frequency of modifications to individual files.

## Getting data from Source Code

When source code is cloned to disk, it lives as individual files and directories on your hard drive. While it is technically possible to try to interpret the source code and extract meaning from it by coding an interpreter per programming language supported, I wanted my process to be more general and focus just on lines of code per file. This approach allowed my analysis process to support any programming language that saved a file to disk.

Gathering this data was fairly trivial and involved writing a small Python program that recursively looked at all files in a directory and then examined each directory inside of that directory (Eland, Building a File Analysis Dataset with Python, 2023). As it went it looked only at source files and read the count of lines in that file. The results were then saved to a **FileSizes.csv** file and included relative path information and the names of the first and second directories that the tool entered called Project and Area respectively.

Table

Description automatically generated

## Aggregating Data

Once I had the commit data, file commit data, and file size data, I decided to aggregate these data sources into two de-normalized files. While Tableau and tools like it do allow you to join between data sources, I found that I was iterating at a rate that fast enough to cause bugs in Tableau’s data sources and having flattened or de-normalized files gave me some additional debugging insight into places my scripts were failing while also preventing me from having to redo work down the line in Tableau. This has the added benefit of allowing me to provide clearer usage instructions to others who want to replicate my work later.

I generated a **MergedFileData.csv** file that provides additional information about each file commit by joining together the FileCommits.csv data source and the FileSizes.csv data source. This join operation occurred in Python code using Pandas as part of the data extraction process I mentioned above.

The resulting dataset looks as follows:

Table

Description automatically generated

Next, I decided to aggregate commit activity at a per-file basis by joining FileSizes.csv and FileCommits.csv using Python and Pandas to have a flattened view of metrics per file. This data was then saved into **FileData.csv**

Table

Description automatically generated

All told, my primary data sources for working with Tableau were FileData.csv and MergedFileData.csv as these provided the most context during reporting.

## Limitations of this Approach

While I am proud of my data extraction process, it does have some limitations, starting with its performance.

Unfortunately, the whole process is fairly slow, taking anywhere from 0.2 to 0.8 seconds per commit to analyze. This resulted in it taking roughly 17 minutes to generate the required files from the Polyglot Notebooks repository.

This process was also used on other git repositories to validate its portability and a similar performance level was measured for each one. The primary cause of slowness was the PyDriller library and not any custom analysis work I was doing, so further analysis work after this course will attempt to move away from PyDriller in an attempt to increase analysis performance.

Secondly, the process doesn’t handle file renames yet. If a large number of commits occurred on a file with a specific name and then another developer came in and moved or renamed that file, my data aggregation process currently only counts commits against the final name of the file for file-level aggregation statistics. This is a limitation of my current approach with scripting and something I want to address as I work to rewrite my aggregation process later to improve performance.

Thirdly, this code is very new and largely untried and may have some bugs in its analysis approach that I did not catch during testing or preparing this report.

Finally, this approach will not work with projects that do not use the git version control system or are simply too big to fit all commit data into memory at once.

That being said, I view this approach as a viable prototype for further analysis of any git-based project.

# Findings

Let’s transition from talking about the high-level approach and move on to discussing the Polyglot Notebooks codebase.

## How active has the project been over time?

The project began in September of 2017 with a trickle of commits every month, but it did not pick up in pace until May of 2019. In May of 2019 we saw 74 commits with a net addition of almost 175,000 lines of code.

Chart, line chart

Description automatically generated

The project’s net lines of code (insertions – deletions) then stabilized briefly before a great purge of 200,000 lines of code in January of 2020 before returning back to more stable growth areas.

In terms of peak numbers of individual commits, the project peaked in May of 2020 with 278 commits in that month alone.

That said, the project is still very active with most months having over 50 commits each day, while the net lines of code remains in safer areas indicating that drastic changes to the repository are no longer happening.

## How many people have been working on the project at once?

Open-source projects are often maintained by two or three key contributors. These projects can be vulnerable if a key individual moves off of the project for whatever reason.

Polyglot Notebooks has a healthy mix of active developers each month that appears to be climbing in recent months.

Chart, line chart

Description automatically generated

This corresponds well with the additional press I’ve observed this project receiving since November of 2022 and a growing importance I’ve noticed Microsoft starting to put on the project.

## Where are the authors?

The authors of the Polyglot Notebooks project are primarily Microsoft employees around Redmond and Seattle with a few Microsoft and non-Microsoft individuals interspersed throughout the United States. There are a few contributors in the UK, Mexico, and Kazakhstan, but development is primarily occurring in the United States and more specifically in Washington.

Map

Description automatically generated

It should be noted, however, that a location could not be determined for every committer involved in the project and this set of countries is likely incomplete.

The git commit data does include timezone information, but I observed this data to be largely unreliable so it is not represented in this report. As an anecdotal example, I write this report from Ohio in the United States and the time zone information I’ve observed from my own commits indicated I’m writing from Abu Dhabi.

On an interesting note, if you aggregate the net lines of code by country, the United Kingdom has actually contributed a negative amount of lines of code to the project due to some high volume deletes coming from authors in the UK.

Map

Description automatically generated

This just goes to show that net lines of code is an awful metric to use to gauge the productivity of your developers.

## Who worked the most on this project?

If we look at volume of commits and only include authors with five total commits or more, there are definitely a few key contributors on this project.

Chart, waterfall chart

Description automatically generated

Most notably, Diego Colombo has been incredibly active on the project over the years, though his rate of commits may be slowing in 2023 as of mid-March.

However, Jon Sequeira and Brett V. Forsgren are also incredibly active on the project and newer authors Osvaldo Calles and Shyam Namboodiripad are making their presence known.

Looking at the lines of code authored, we see that Brett V. Forsgren has clearly authored the most lines of code, but our key contributors have also made significant contributions.

Chart

Description automatically generated

If we plot all contributors on the project based on their weekly contributions, some clear trends emerge.

Chart, scatter chart

Description automatically generated

First of all, there are a large number of small commits occurring regularly by a variety of contributors.

However, the high-profile contributors such as Diego Colombo are committing regularly at extremely high volumes, though the volume of commits is decreasing as the project grows more and more stable.

This trend is clarified if we exclude all but the top five authors:

Chart, scatter chart

Description automatically generated

## How is the project structured?

While Tableau tree maps leave something to be desired compared to tree maps on other platforms, they can tell us a decent amount of information.

Here is the breakdown of the structure of the project in terms of its directory structure, the size of each file, and highlighting the programming language used per file.

Chart, treemap chart

Description automatically generated

As you might expect for a project centered around dotnet development, C# is the predominant language. However, there are entire regions of code dedicated to other languages such as F#, shell scripts, markdown files for documentation, and Jupyter notebooks for sample code.

## Are there any large files?

Most projects have a few files that are simply too big to be easily maintained without introducing bugs.

Chart, treemap chart

Description automatically generated

Polyglot Notebooks largely dodges this trend, though it does have a Formatting-outputs.ipynb file that is 2,638 lines long. However this is a Jupyter Notebook and the raw contents of those files is not visible to a human, so this is actually fairly safe.

Beyond this file a number of C# and F# files are in the 1,600 lines of code range which can be problematic, but nothing I would say needs to be urgently fixed.

All-in-all, this is one of the best codebases I’ve ever seen in terms of file sizes.

## What files have changed the most?

In many software projects you have a few key files that change very frequently. This is often an indication that those files are trying to do too much and have too many different reasons to change.

Polyglot Notebooks does have some code that changes frequently.

Chart, treemap chart

Description automatically generated

Specifically the README.md file seems to change with every commit. As a documentation file, this is likely being changed as part of an automated or mandated process for a changelog and is probably fine to ignore.

However, CommandLineParser.cs and KernelExtensions.cs both have changed over 100 times over the course of the project. This makes me suspect that these files are a central focus and may need additional scrutiny, particularly given their larger size. It might be possible that these files can be reduced into a number of additional smaller files that have fewer reasons to change or are less likely to break.

## How long has it been since each file was modified?

Usually codebases have a large number of files that haven’t been modified in awhile and a smaller set that have. This is actually the reverse of what we see in Polyglot Notebooks.

Chart, treemap chart

Description automatically generated

Here we see that most files have changed recently and only a few isolated files have remained unchanged for years.

To me, this indicates that a number of system-wide changes have occurred over the years that necessitated modifying most files in the system. It is unclear why those changes did not impact the older files as well, but this is certainly an interesting anomaly in the data.

## Who has the most ownership of each area?

If we look at the most frequent committer to any file, we can usually identify the individual driving development or possessing the most domain knowledge in that code.

Chart, treemap chart

Description automatically generated

Key individuals who have the majority of commits in individual areas can become knowledge silos that may leave the project and take their knowledge with them, posing a risk to the team’s success in their absence.

In this project we see that Diego Colombo has been the most frequent committer to roughly half of the files with Brett V. Forsgren and Jon Sequeira keeping pace with the other half of the files.

We also see a few key contributors with ownership of key areas, such as Justin Chen’s involvement with Powershell and Charles Gagnon’s involvement in SQL language features.

Pivoting things to look at the individual area of people’s involvement, we see that most of our frequent contributors touch most areas of the system.

Chart, bar chart

Description automatically generated

This is somewhat of an anomaly with most systems having clear specialists in one area versus another.

This added diversity of authors in each area helps the team spread knowledge around and reduces the risks present in a key individual leaving the project.

This diversity can further be visualized in a tree view of the number of unique authors modifying each file:

Chart, treemap chart

Description automatically generated

Here we see that it is somewhat rare for files to have fewer than 3 authors (red files), but these files with few authors tend to be the smaller files that are less frequently changed and typically less critical.

We also note that the file with the most unique authors is the readme.md file, further indicating that this documentation is mandatory to change every time a commit is made.

# Conclusion and Next Steps

All told, I believe that Polyglot Notebooks is an incredibly stable project with a core team that is very motivated in expanding the codebase, a stable net lines of code, and a growing number of contributors.

Graphical user interface, timeline

Description automatically generated

While key contributors change over time, the project benefits from a diverse set of developers involved in every area of the project and sharing knowledge with each other and the community.

Additionally, the raw code metrics behind the project suggest that the code is very healthy and should be stable enough to handle continued development. Of course, these metrics are based purely on change frequency and line count and do not involve language-specific insights such as cyclomatic complexity, % tests, % comment, or other measurements.

# Chart, treemap chart Description automatically generated

## Next Steps for the Analysis Workflow

This visualization experiment indicates to me that the prototype scripts I’ve developed can be applied to most git-based software projects. After this course ends I plan to rewrite my data aggregation process in dotnet and incorporate it into a Polyglot Notebook that others can use to analyze their code. During this step I will look to improve analysis performance and handle renamed files.

I also want to document the analysis process in Tableau, expand the data visualization efforts into Power BI or even potentially Excel workbooks, and provide a Polyglot or Jupyter Notebook with inline visuals directly in the notebook and instructions for others to use this code to analyze their own repositories.

In its most polished form, I could build a WinUI desktop or mobile application that could perform the same analysis for users and provide rich charts directly in that application. Alternatively, this product could integrate into Visual Studio and/or Visual Studio Code as an extension to the editor.

Ultimately, I believe this project to have merit to many software development teams out there and I believe the analysis I’ve provided has shown some merit, even though there are a few outstanding script performance and accuracy issues.

# References

Davide, S. (n.d.). *PyDriller*. Retrieved from GitHub: https://github.com/ishepard/pydriller

dotnet. (n.d.). *.NET Interactive.* Retrieved from GitHub: http://github.com/dotnet/interactive

Eland, M. (2022, December 14). *Interactive C# with Polyglot Notebooks.* Retrieved from The New Dev's Guide: https://newdevsguide.com/2022/12/14/polyglot-notebooks-csharp/

Eland, M. (2022, January). *Visualizing Code*. Retrieved from MattEland.dev: https://matteland.dev/visualizingcode

Eland, M. (2023, February 26). *Building a File Analysis Dataset with Python.* Retrieved from Accessible AI: https://accessibleai.dev/post/building\_file\_analysis\_data/

Eland, M. (2023, February 26). *Extracting git repository data with PyDriller.* Retrieved from Accessible AI: https://accessibleai.dev/post/extracting-git-data-pydriller/

Regio, C. (2022, November 3). *.NET Interactive Notebooks is now Polyglot Notebooks!* Retrieved from Microsoft .NET Blogs: https://devblogs.microsoft.com/dotnet/dotnet-interactive-notebooks-is-now-polyglot-notebooks/