INflow Labs

Using Git Versioning Data to Analyze Development Lifecycles

by Ryan Malani, Shaun Laranjeira March 15, 2023 4 min read

Recently, we heard from a customer that it is difficult to track and understand the progress being made on a software project, and it is especially difficult when they aren't a part of the active agile scrum meetings. Well, that got us thinking that there could be a way to use the versioning data (from git, a DoD DevSecOps approved tool) of a code repository to understand a little bit more about the progression of the project and from there, developing a product that serves to infer the lifecycle of a similar project in the future (and we have some ideas for that too, so stay tuned!).

Getting Started

There are a few python libraries that we used for this short project. This project was also inspired by this write up on notebook community[1]. For reference, we are using M1/M2 Macs at HQ, so we set up our development environment from miniforge[2]. Similarly to the previous project regarding document keyword frequencies, we started with creating a conda environment:

```
conda create env --name YOUR_NAME -y
```

Once you have your new conda environment activate it using:

```
conda activate YOUR_NAME
```

For those using Windows, creating a new virtual environment[3] should act to serve the same purpose. To specify the python version used, you can use the command option –-python.

With the environment active, it's time to start installing the requirements that we'll be using for this project with pip. This single line should install all of the desired packages.

```
pip install python-dateutil pandas git random plotly matplotlib numpy
mplcursors
```

The packages, dateutil[4], pandas[5], GitPython[6], plotly[7], matplotlib[8], numpy[9], mplcursors[10] have far greater capability than we explore here - we strongly recommend exploring the references. Lastly, pull down the repository that you're looking to analyze and clone it locally using:

```
git clone <COPIED SSH ADDRESS OF REPO>
```

Pulling in the Repo Data to be Analyzed

Using the packages you installed, we'll read in the repo commit data using the following few lines of code:

```
GIT_REPO_PATH = r'../repositories/big-bang'

repo = git.Repo(GIT_REPO_PATH, odbt=git.GitCmdObjectDB)

repo

commits = pd.DataFrame(repo.iter_commits('master'), columns=['raw'])
commits.head()
```

This will display the top 5 commits in the DataFrame. Next, we'll want to pull the last commit to the repository to identify the different data points that each commit holds to further build out our dataframe using lambda functions.

```
last_commit = commits.iloc[0]

last_commit['raw'].__slots__

commits['sha'] = commits['raw'].apply(lambda x: str(x))
commits['author'] = commits['raw'].apply(lambda x: x.author.name)
commits['committed_date'] = commits['raw'].apply(lambda x:
pd.to_datetime(x.committed_datetime, utc=True))
commits['message'] = commits['raw'].apply(lambda x: x.message)
commits['parents'] = commits['raw'].apply(lambda x: x.parents)
commits['committer'] = commits['raw'].apply(lambda x: x.committer)

commits.head()
```

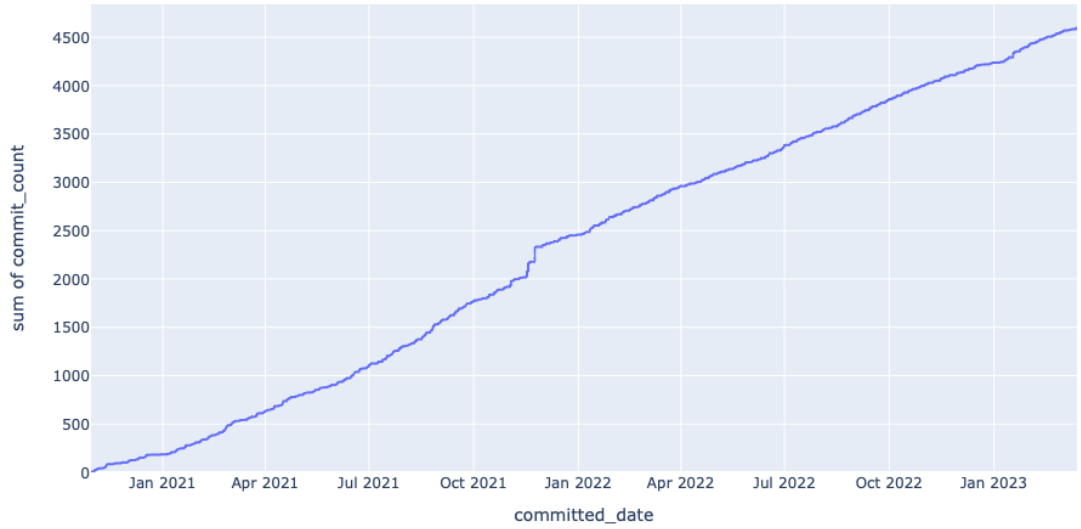
Out[7]:

	raw	sha	author	committed_date	message	
0	4c2a531567007909429bbe6840f69556c1b83772	4c2a531567007909429bbe6840f69556c1b83772	Ryan Garcia	2023-03-14 21:49:40+00:00	Merge branch 'enable-mtls-for-neuvector-metric...	(4989e8adf01ecd40016362e42
1	488f89f27d38579be9fe1f6be677ba3e96b95f73	488f89f27d38579be9fe1f6be677ba3e96b95f73	Brett Charrier	2023-03-14 21:49:40+00:00	Enable mTLS for Neuvector metrics\n	(b7c7731260cb9993c8b43710
2	4989e8adf01ecd40016362e424b279c5e9b704d4	4989e8adf01ecd40016362e424b279c5e9b704d4	Micah Nagel	2023-03-14 17:52:55+00:00	Merge branch 'update-kyverno-policies-tag-1.1....	(11b3b19854f664968dcf58aac
3	d8d05afdb8f43af607a30cf55475a35bbe9b65f3	d8d05afdb8f43af607a30cf55475a35bbe9b65f3	mr-bot	2023-03-14 17:52:55+00:00	kyverno-policies update to 1.1.0-bb.3\n	(ddd6e3eb73cec0bddee1ee224
4	11b3b19854f664968dcf58aac56079d123533a96	11b3b19854f664968dcf58aac56079d123533a96	Micah Nagel	2023-03-14 17:52:32+00:00	Merge branch 'update-kyverno-tag-2.6.5-bb.3' i...	(15c82732e41ae61799da95da

Visualizing the Commit Data

Now the DataFrame should have the columns for sha id, author, commit date, commit message, parent commits, and committer. The first visualization we'll want to see is the commits per day compounded over the development period. We can do so by creating a new DataFrame with commits grouped by commited date and then using an empiral cumulative distribution function (ECDF) plot:

```
# grouping the commits by date since there are multiple for each date
typically
commits_by_date =
commits.groupby('committed_date').size().reset_index(name='commit_cou
nt')

commits_by_date

# using the dataframe created in the previous cell with the count of
commits per day, we'll create an interactive plot
fig = px.ecdf(commits_by_date, x='committed_date', y='commit_count',
ecdfnorm=None)
fig.show(renderer='notebook')
```



Gaining a Deeper Understanding of the Commit Data

Now, managing the delivery of software projects boils down to more than just commits, as we can understand in more depth the magnitude of changes by incorporating the data associated with additions and deletions as parts of commits. This will help us understand if the team is making bigger changes to the code at different points in the development process or different parts of the year. Now this is a little more challenging, so I'll share the code block to join the lines changed data to the commits DataFrame and then explain how it works afterwards.

```
# understanding stats associated with a random commit
rand_commit = commits.iloc[random.randint(0, commits.shape[0])]
['raw']

rand_commit.stats.files

# in order to make a stacked dataframe to later merge, we'll wrap a
dataframe around a pandas series of the stats and call the stack()
method
rand_commit_df =
pd.DataFrame(pd.Series(rand_commit.stats.files)).stack()

stats = pd.DataFrame(commits['raw'].apply(lambda x:
pd.Series(x.stats.files, dtype=object)).stack()).reset_index(level=1)
stats = stats.rename(columns={ 'level_1' : 'filename', 0 :
'stats_modifications'})
stats_modifications = stats['stats_modifications'].apply(lambda x:
pd.Series(x))
stats = stats.join(stats_modifications)
del(stats['stats_modifications'])

commits = commits.join(stats)
del(commits['raw'])

commits.head()
```

Out[14]:

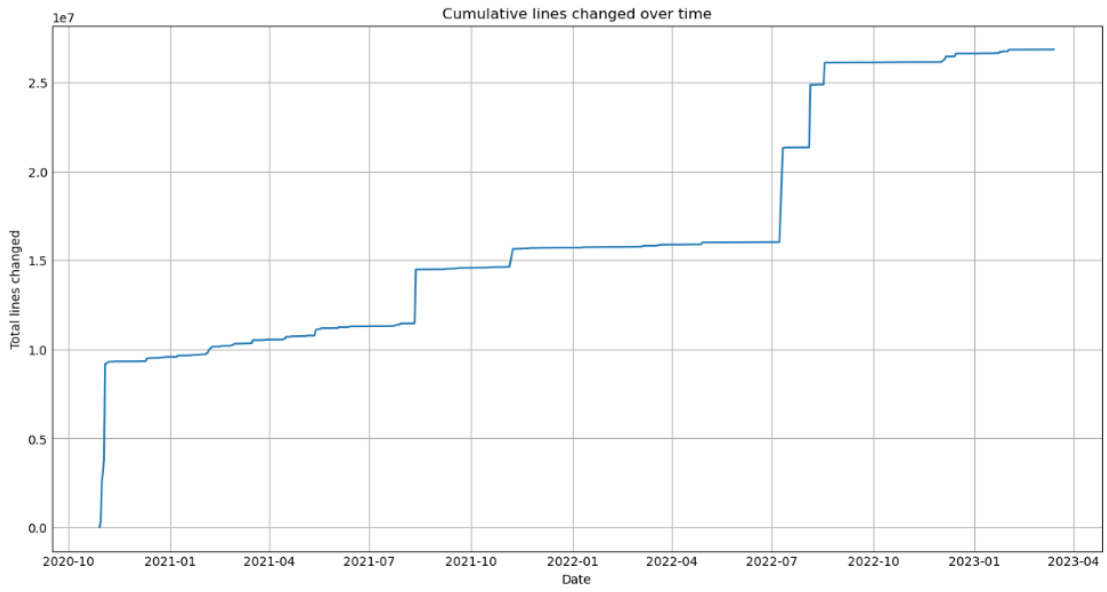
	author	committed_date	message	parents	committer	filename	insertions	deletions	lines
	Ryan Garcia	2023-03-14 21:49:40+00:00	Merge branch 'enable-mtls-for-neuvector-metric...	(4989e8adf01ecd40016362e424b279c5e9b704d4, 488...	Ryan Garcia	chart/templates/neuvector/values.yaml	9.0	2.0	11.0
	Ryan Garcia	2023-03-14 21:49:40+00:00	Merge branch 'enable-mtls-for-neuvector-metric...	(4989e8adf01ecd40016362e424b279c5e9b704d4, 488...	Ryan Garcia	chart/templates/neuvector/values.yaml	1.0	1.0	2.0
	Ryan Garcia	2023-03-14 21:49:40+00:00	Merge branch 'enable-mtls-for-neuvector-metric...	(4989e8adf01ecd40016362e424b279c5e9b704d4, 488...	Ryan Garcia	chart/values.yaml	9.0	2.0	11.0
	Ryan Garcia	2023-03-14 21:49:40+00:00	Merge branch 'enable-mtls-for-neuvector-metric...	(4989e8adf01ecd40016362e424b279c5e9b704d4, 488...	Ryan Garcia	chart/values.yaml	1.0	1.0	2.0
	Brett Charrier	2023-03-14 21:49:40+00:00	Enable mTLS for Neuvector metrics\n	(b7c7731260cb9993c8b437103e5e4f526bf70296,)	Ryan Garcia	chart/templates/neuvector/values.yaml	9.0	2.0	11.0

From the comments in the code above, you can see we're pulling the stats on files changed for a random commit as a Pandas Series, and then applying the same effects to the remainder of the data in the commits DataFrame through the join function from Pandas.

With a full DataFrame, we want to now visualize the lines changed over time and per day. To make those visualizations, we'll use the following code:

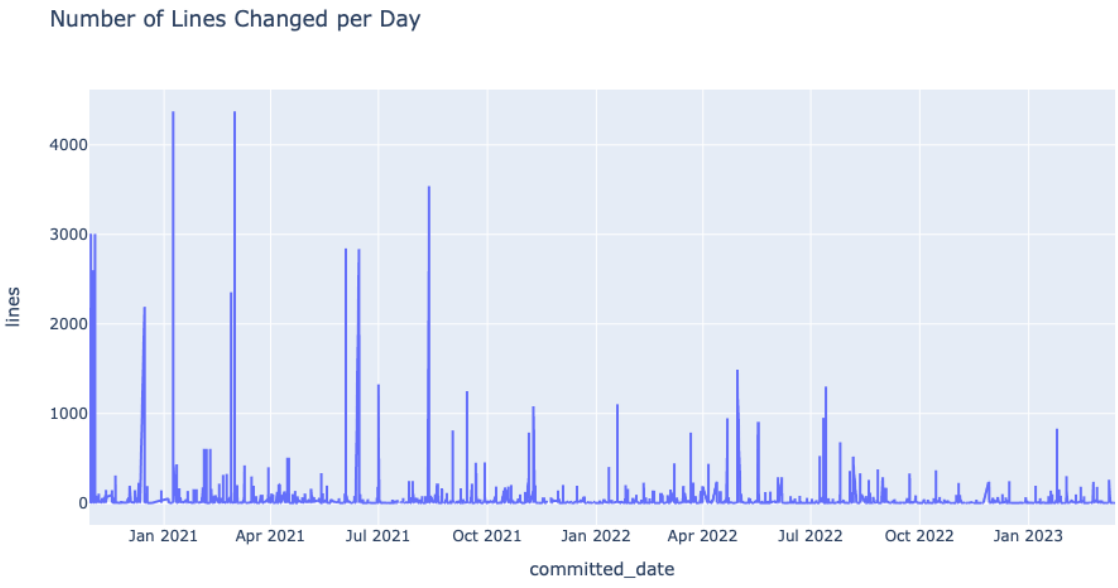
Cumulative Lines Changed Over Time:

```
plt.figure(figsize=(12, 6))
plt.style.use('_mpl-gallery')

# grouping the commits by date since there are multiple for each date
typically
commits_by_date =
commits.groupby(commits['committed_date'].dt.date).sum(numeric_only=T
rue)

# plotting the cumulative lines changed over time
plt.plot(commits_by_date.index, commits_by_date['lines'].cumsum())
plt.xlabel('Date')
plt.ylabel('Total lines changed')
plt.title('Cumulative lines changed over time')

cursor = mplcursors.cursor(ax)
cursor.connect('add', lambda sel: sel.annotation.set_text(
    f"{sel.index}\n{sel.artist.get_xdata()[sel.target.index]:%Y-%m-
%d}\n{sel.target[sel.index]:.0f} lines changed"))

plt.show()
```



Number of Lines Changed per Day:

```
fig = px.line(commits, x='committed_date', y='lines', title='Number
of Lines Changed per Day')
fig.show(renderer='notebook')
```



Future Iterations

To continue improving on this idea, we plan to incorporate machine learning topic areas such as natural language processing (NLP) for identifying other similar projects (based on the code, project descriptions, etc) as well as linear regression (to identify what the timeline would look like for similar projects and their anticipated scale). As we work on these areas of expansion, we'll continue to update here on our labs site.

Usage

If you want to try this out for yourself, clone the source code (https://github.com/INflow-Federal/project-analysis) repository on our GitHub and open the jupyter notebook in your conda environment in terminal using `jupyter notebook project-analysis.ipynb`. Any questions? Feel free to contact us and ask away.

---

1. https://notebook.community/feststelltaste/software-analytics/prototypes/Reading%20Git%20logs%20with%20Pandas%202.0%20with%20bonus (https://notebook.community/feststelltaste/software-analytics/prototypes/Reading%20Git%20logs%20with%20Pandas%202.0%20with%20bonus) ↩
2. https://github.com/conda-forge/miniforge (https://github.com/conda-forge/miniforge) ↩
3. https://docs.python.org/3/library/venv.html (https://docs.python.org/3/library/venv.html) ↩
4. https://pypi.org/project/python-dateutil/ (https://pypi.org/project/python-dateutil/) ↩
5. https://pandas.pydata.org (https://pandas.pydata.org) ↩
6. https://gitpython.readthedocs.io/en/stable/ (https://gitpython.readthedocs.io/en/stable/) ↩
7. https://plotly.com (https://plotly.com) ↩
8. https://matplotlib.org (https://matplotlib.org) ↩
9. https://numpy.org (https://numpy.org) ↩
10. https://mplcursors.readthedocs.io/en/stable/ (https://mplcursors.readthedocs.io/en/stable/) ↩