Assignment-1

Software Tools and Techniques for CSE

Shardul Junagade (23110297)
Repository: cs202-stt

September 8, 2025

Contents

Lab 4: Exploration of Different Diff Algorithms on Open-Source Repositories

Repository Link: cs202-stt/lab4

1.1 Introduction, Setup, and Tools

1.1.1 Introduction

In this lab, I compared two diff algorithms – **Myers** and **Histogram** – to see how they behave on real-world open-source repositories. The main task was to extract per-file diffs for each modified file in the commit history, identify where the two algorithms produced different results, and analyze whether these mismatches were more common in code files, test files, or documentation.

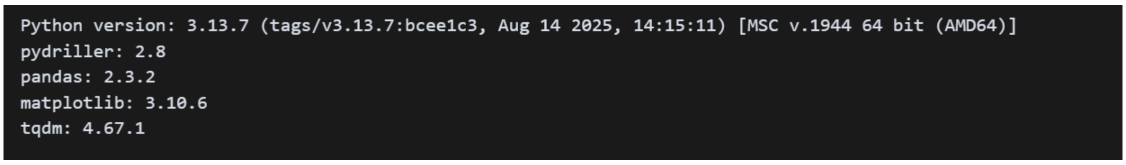
Myers Algorithm: The Myers algorithm is the **default in Git** and is based on finding the shortest edit script between two versions of a file. It is efficient and works well in general, but sometimes produces diffs that are harder to read, especially when code blocks are moved or reordered.

Histogram Algorithm: The Histogram algorithm, on the other hand, tries to anchor diffs on *rare lines first* (lines that appear less frequently), which often makes the changes more meaningful and easier to follow.

The motivation for this lab was to understand whether the choice of diff algorithm makes a practical difference for developers, reviewers, and automated tools.

1.1.2 Environment and Tools

- **Operating System:** Windows 11
- **Terminal:** PowerShell 7
- **Code Editor:** Visual Studio Code - Insiders
- **Python version:** 3.13.7
- **PyDriller version:** 2.8
- **SEART GitHub:** `https://seart-ghs.si.usi.ch/`
- **Notebooks:**
  - `lab4/diff_extract.ipynb` (data extraction)
  - `lab4/diff_analyse.ipynb` (analysis and plotting)



```
Python version: 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)]
pydriller: 2.8
pandas: 2.3.2
matplotlib: 3.10.6
tqdm: 4.67.1
```

Environment Details

1.2 Methodology and Execution

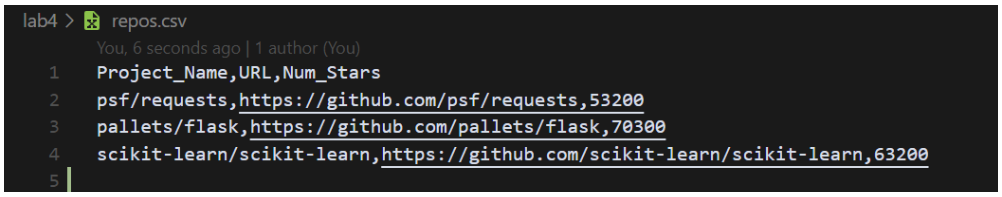
1.2.1 Repository Selection and Criteria

My selection was based on:

- **Activity:** Projects with regular commits and long histories.
- **Popularity:** Well-recognized repositories with high stars and forks.
- **Variety of files:** Containing source code, tests, and documentation.
- **Language consistency:** All three projects are Python-based, making analysis simpler.

Normally, I would start with a larger set of repositories using tools like the SEART GitHub Search Engine and then narrow down based on these criteria. The chosen projects already satisfied these requirements.

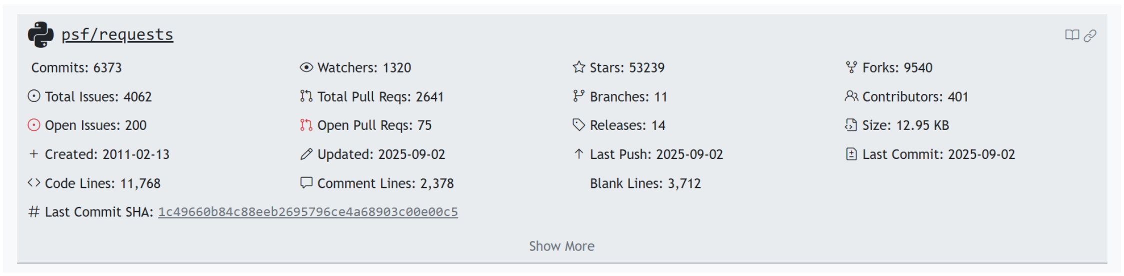
I selected three well-known Python projects to get varied histories and file types:



```
Project_Name,URL,Num_Stars
psf/requests,https://github.com/psf/requests,53200
pallets/flask,https://github.com/pallets/flask,70300
scikit-learn/scikit-learn,https://github.com/scikit-learn/scikit-learn,63200
```

Repository Selection

1. psf/requests – requests is a widely used Python library for making HTTP requests. It is popular because of its simple API, reliability, and extensive use in both academic and industry projects. With over 30,000 commits, 10,000 forks, and 50,000 stars on GitHub, it is considered a highly influential and well-maintained project. Its rich commit history makes it suitable for analyzing diff algorithm discrepancies. The project was selected directly from GitHub due to its popularity and long development history.

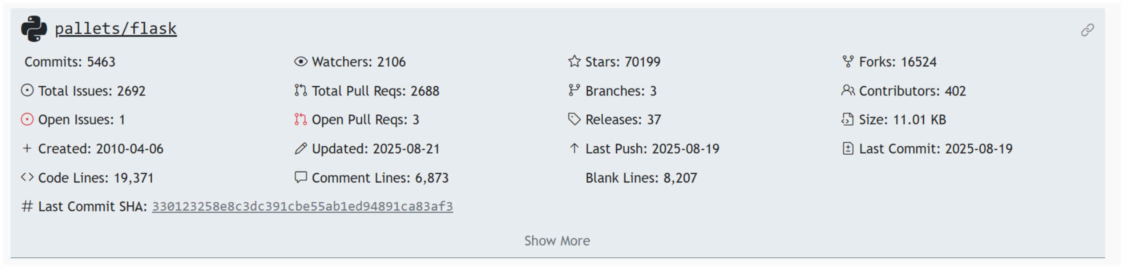

psf/requests

Commits: 6373	Watchers: 1320	Stars: 53239	Forks: 9540
Total Issues: 4062	Total Pull Reqs: 2641	Branches: 11	Contributors: 401
Open Issues: 200	Open Pull Reqs: 75	Releases: 14	Size: 12.95 KB
Created: 2011-02-13	Updated: 2025-09-02	Last Push: 2025-09-02	Last Commit: 2025-09-02
Code Lines: 11,768	Comment Lines: 2,378	Blank Lines: 3,712	

Last Commit SHA: 1c49660b84c88eeb2695796ce4a68903c00e00c5

GitHub metrics for psf/requests

2. pallets/flask – Flask is a lightweight Python web framework known for its simplicity and flexibility. It is widely adopted in both small and large-scale web applications. Flask has more than 15,000 commits, 6,000 forks, and 65,000 stars on GitHub, reflecting its popularity and active maintenance. The repository contains extensive test suites and documentation, which makes it an ideal candidate for this lab. Its selection was done using GitHub search and project metrics.

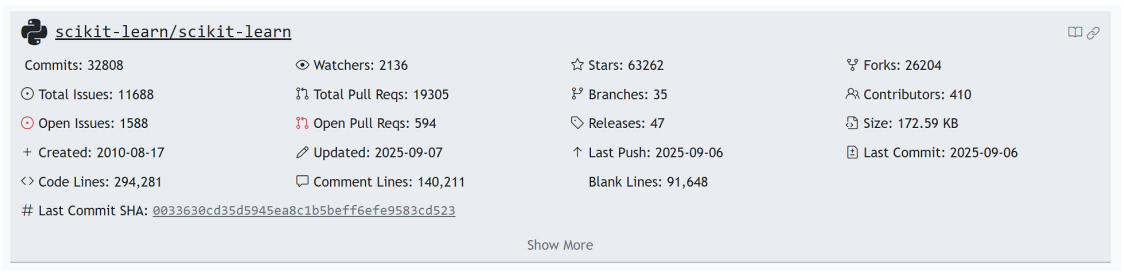

pallets/flask

Commits: 5463	Watchers: 2106	Stars: 70199	Forks: 16524
Total Issues: 2692	Total Pull Reqs: 2688	Branches: 3	Contributors: 402
Open Issues: 1	Open Pull Reqs: 3	Releases: 37	Size: 11.01 KB
Created: 2010-04-06	Updated: 2025-08-21	Last Push: 2025-08-19	Last Commit: 2025-08-19
Code Lines: 19,371	Comment Lines: 6,873	Blank Lines: 8,207	

Last Commit SHA: 330123258e8c3dc391cbe55ab1ed94891ca83af3

GitHub metrics for pallets/flask

3. scikit-learn/scikit-learn – scikit-learn is one of the most important machine learning libraries in Python. It is heavily used in both research and industry for tasks such as classification, regression, clustering, and model evaluation. The repository has over 27,000 commits, 25,000 forks, and 60,000 stars on GitHub, which highlights its active community and long development history. Due to its diverse codebase and well-documented commit history, it was selected as a representative project for this study.



scikit-learn/scikit-learn

Commits: 32808 · Watchers: 2136 · Stars: 63262 · Forks: 26204
Total Issues: 11688 · Total Pull Reqs: 19305 · Branches: 35 · Contributors: 410
Open Issues: 1588 · Open Pull Reqs: 594 · Releases: 47 · Size: 172.59 KB
Created: 2010-08-17 · Updated: 2025-09-07 · Last Push: 2025-09-06 · Last Commit: 2025-09-06
Code Lines: 294,281 · Comment Lines: 140,211 · Blank Lines: 91,648
Last Commit SHA: 0033630cd35d5945ea8c1b5beff6efe9583cd523

Show More

GitHub metrics for scikit-learn/scikit-learn

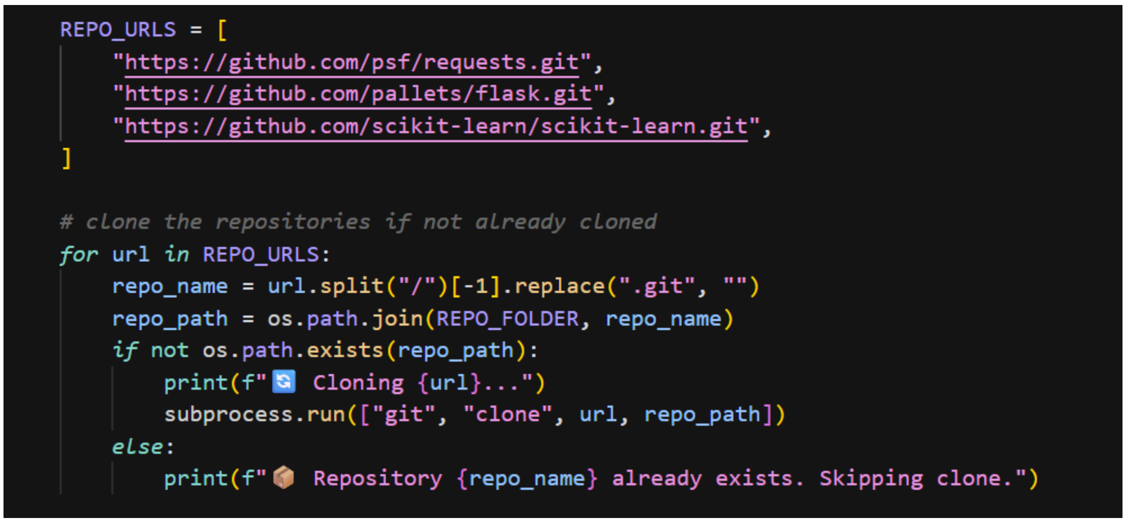
These repositories represent some of the most widely adopted projects in the Python ecosystem. Each of them has tens of thousands of stars on GitHub (Requests: 53k, Flask: 71k, Scikit-learn: 63k), thousands of commits (ranging from 5k to over 32k), and a large contributor base of around 400 developers each. Their popularity, long commit history, active development, and extensive documentation make them excellent candidates for studying diff algorithm behavior for the purpose of this lab.

1.2.2 Diff Extraction Pipeline and Discrepancy Handling

Notebook link: lab4/diff_extract.ipynb

CSV Files: lab4/results

I cloned each repository under `lab4/repos/` using the `git clone` command.



```python
REPO_URLS = [
    "https://github.com/psf/requests.git",
    "https://github.com/pallets/flask.git",
    "https://github.com/scikit-learn/scikit-learn.git",
]

# clone the repositories if not already cloned
for url in REPO_URLS:
    repo_name = url.split("/")[-1].replace(".git", "")
    repo_path = os.path.join(REPO_FOLDER, repo_name)
    if not os.path.exists(repo_path):
        print(f"🔄 Cloning {url}...")
        subprocess.run(["git", "clone", url, repo_path])
    else:
        print(f"📦 Repository {repo_name} already exists. Skipping clone.")
```

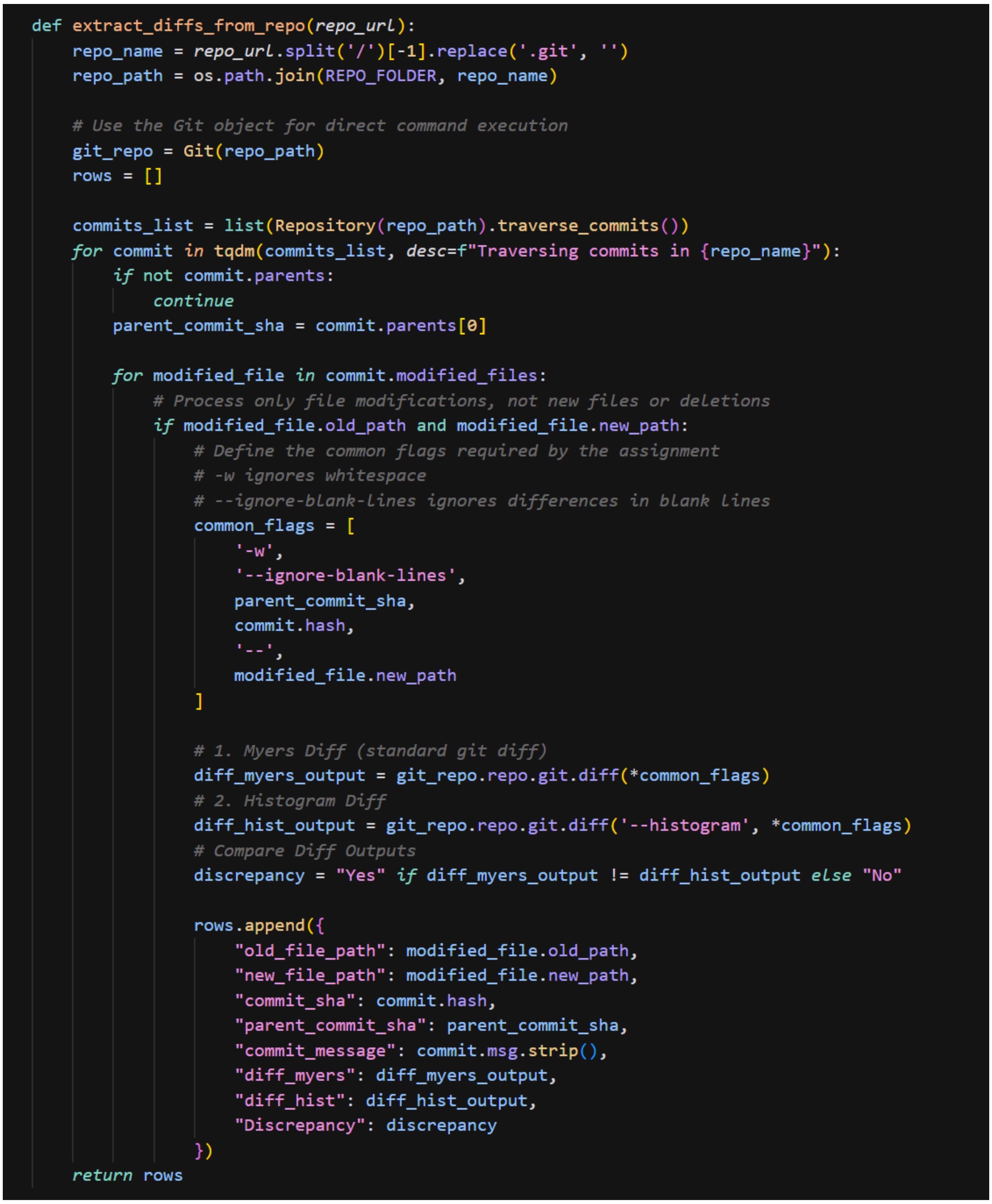
Git Clone

Then, I traversed each commit and:

- Extracted modified files (excluding newly added/deleted files).
- Computed two diffs per file:
  - Myers (`git diff`)
  - Histogram (`git diff --histogram`)
- Ignored whitespace and blank line changes using flags `-w` and `--ignore-blank-lines`. I also handled edge cases like if a commit is a root commit, i.e., has no parent.

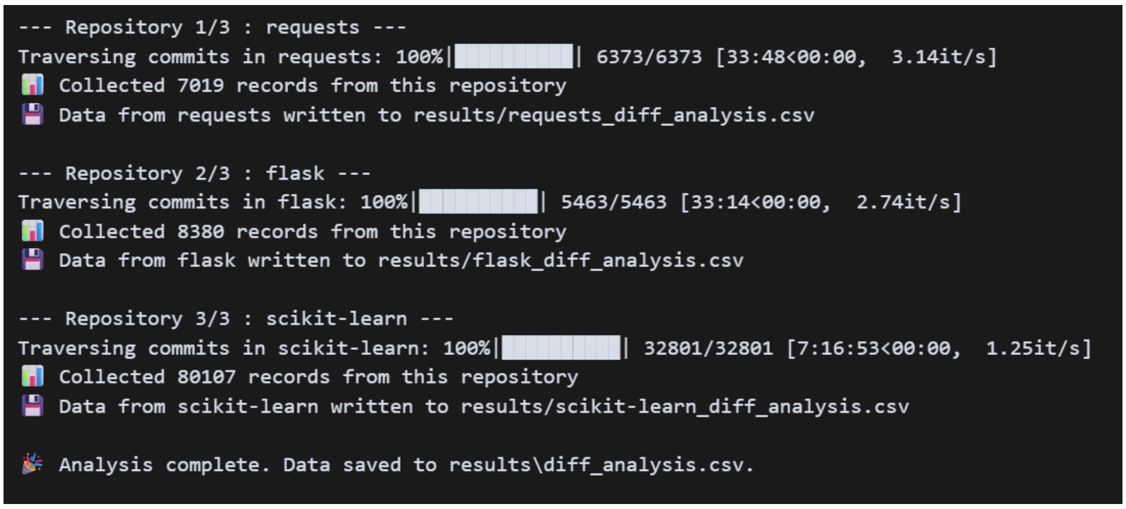
For each file-modifying commit I stored the following data:

- commit_sha, parent_commit_sha
- old_file_path, new_file_path
- commit_message
- diff_myers (plain text)
- diff_hist (plain text)
- Discrepancy - A discrepancy was marked as **"Yes"** if the Myers and Histogram diffs were different, otherwise **"No"**.



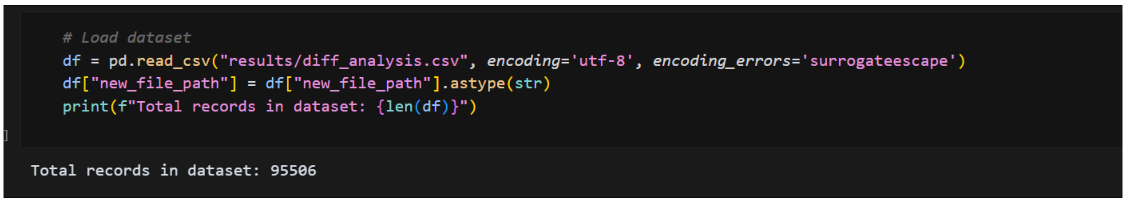
```python
def extract_diffs_from_repo(repo_url):
    repo_name = repo_url.split('/')[-1].replace('.git', '')
    repo_path = os.path.join(REPO_FOLDER, repo_name)

    # Use the Git object for direct command execution
    git_repo = Git(repo_path)
    rows = []

    commits_list = list(Repository(repo_path).traverse_commits())
    for commit in tqdm(commits_list, desc=f"Traversing commits in {repo_name}"):
        if not commit.parents:
            continue
        parent_commit_sha = commit.parents[0]

        for modified_file in commit.modified_files:
            # Process only file modifications, not new files or deletions
            if modified_file.old_path and modified_file.new_path:
                # Define the common flags required by the assignment
                # -w ignores whitespace
                # --ignore-blank-lines ignores differences in blank lines
                common_flags = [
                    '-w',
                    '--ignore-blank-lines',
                    parent_commit_sha,
                    commit.hash,
                    '--',
                    modified_file.new_path
                ]

                # 1. Myers Diff (standard git diff)
                diff_myers_output = git_repo.repo.git.diff(*common_flags)
                # 2. Histogram Diff
                diff_hist_output = git_repo.repo.git.diff('--histogram', *common_flags)
                # Compare Diff Outputs
                discrepancy = "Yes" if diff_myers_output != diff_hist_output else "No"

                rows.append({
                    "old_file_path": modified_file.old_path,
                    "new_file_path": modified_file.new_path,
                    "commit_sha": commit.hash,
                    "parent_commit_sha": parent_commit_sha,
                    "commit_message": commit.msg.strip(),
                    "diff_myers": diff_myers_output,
                    "diff_hist": diff_hist_output,
                    "Discrepancy": discrepancy
                })
    return rows
```

Code Snippet for Diff Extraction



```
--- Repository 1/3 : requests ---
Traversing commits in requests: 100%|██████████| 6373/6373 [33:48<00:00,  3.14it/s]
📊 Collected 7019 records from this repository
💾 Data from requests written to results/requests_diff_analysis.csv

--- Repository 2/3 : flask ---
Traversing commits in flask: 100%|██████████| 5463/5463 [33:14<00:00,  2.74it/s]
📊 Collected 8380 records from this repository
💾 Data from flask written to results/flask_diff_analysis.csv

--- Repository 3/3 : scikit-learn ---
Traversing commits in scikit-learn: 100%|██████████| 32801/32801 [7:16:53<00:00,  1.25it/s]
📊 Collected 80107 records from this repository
💾 Data from scikit-learn written to results/scikit-learn_diff_analysis.csv

🎉 Analysis complete. Data saved to results\diff_analysis.csv.
```

Code Output for Diff Extraction

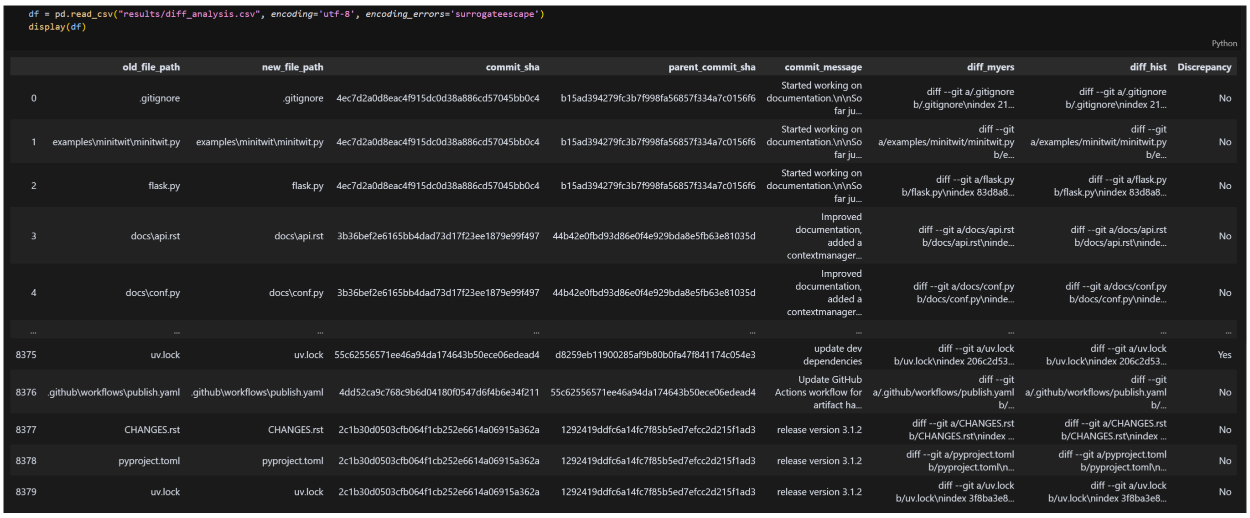


```python
# Load dataset
df = pd.read_csv("results/diff_analysis.csv", encoding='utf-8', encoding_errors='surrogateescape')
df["new_file_path"] = df["new_file_path"].astype(str)
print(f"Total records in dataset: {len(df)}")
```

```
Total records in dataset: 95506
```

Number of rows extracted

This gave me a dataset with 95,506 entries. Then, I saved the extracted data into `lab4/results/diff_analysis.csv`. Since scikit-learn has a huge number of commits, the resulting CSV file exceeded 1GB in size, and could not be pushed to GitHub, so I uploaded the results folder to OneDrive instead.



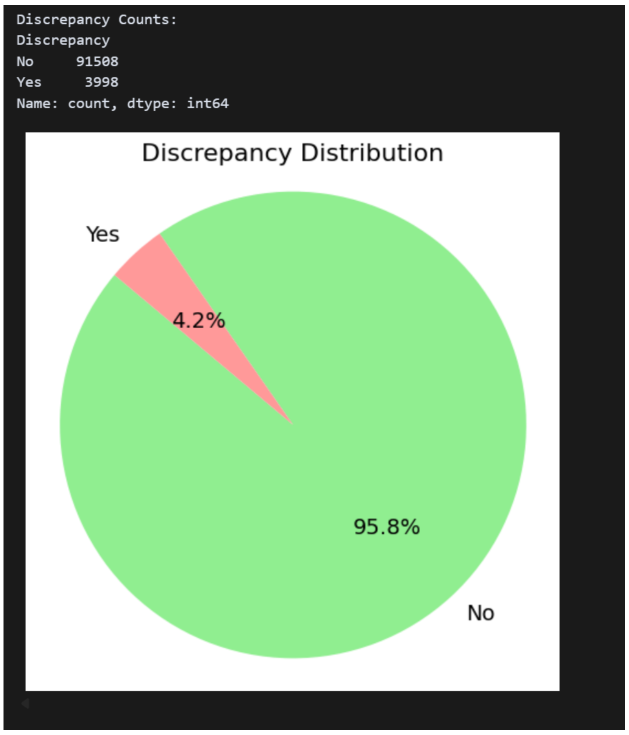
```python
df = pd.read_csv("results/diff_analysis.csv", encoding='utf-8', encoding_errors='surrogateescape')
display(df)
```

Python

	old_file_path	new_file_path	commit_sha	parent_commit_sha	commit_message	diff_myers	diff_hist	Discrepancy
0	.gitignore	.gitignore	4ec7d2a0d8eac4f915dc0d38a886cd57045bb0c4	b15ad394279fc3b7f998fa56857f334a7c0156f6	Started working on documentation.\n\nSo far ju...	diff --git a/.gitignore b/.gitignore\nindex 21...	diff --git a/.gitignore b/.gitignore\nindex 21...	No
1	examples\minitwit\minitwit.py	examples\minitwit\minitwit.py	4ec7d2a0d8eac4f915dc0d38a886cd57045bb0c4	b15ad394279fc3b7f998fa56857f334a7c0156f6	Started working on documentation.\n\nSo far ju...	diff --git a/examples/minitwit/minitwit.py b/e...	diff --git a/examples/minitwit/minitwit.py b/e...	No
2	flask.py	flask.py	4ec7d2a0d8eac4f915dc0d38a886cd57045bb0c4	b15ad394279fc3b7f998fa56857f334a7c0156f6	Started working on documentation.\n\nSo far ju...	diff --git a/flask.py b/flask.py\nindex 83d8a8...	diff --git a/flask.py b/flask.py\nindex 83d8a8...	No
3	docs\api.rst	docs\api.rst	3b36bef2e6165bb4dad73d17f23ee1879e99f497	44b42e0fbd93d86e0f4e929bda8e5fb63e81035d	Improved documentation, added a contextmanager...	diff --git a/docs/api.rst b/docs/api.rst\ninde...	diff --git a/docs/api.rst b/docs/api.rst\ninde...	No
4	docs\conf.py	docs\conf.py	3b36bef2e6165bb4dad73d17f23ee1879e99f497	44b42e0fbd93d86e0f4e929bda8e5fb63e81035d	Improved documentation, added a contextmanager...	diff --git a/docs/conf.py b/docs/conf.py\ninde...	diff --git a/docs/conf.py b/docs/conf.py\ninde...	No
...	...	...	...	...	...	...	...	...
8375	uv.lock	uv.lock	55c62556571ee46a94da174643b50ece06edead4	d8259eb11900285af9b80b0fa47f841174c054e3	update dev dependencies	diff --git a/uv.lock b/uv.lock\nindex 206c2d53...	diff --git a/uv.lock b/uv.lock\nindex 206c2d53...	Yes
8376	.github\workflows\publish.yaml	.github\workflows\publish.yaml	4dd52ca9c768c9b6d04180f0547d6f4b6e34f211	55c62556571ee46a94da174643b50ece06edead4	Update GitHub Actions workflow for artifact ha...	diff --git a/.github/workflows/publish.yaml b/...	diff --git a/.github/workflows/publish.yaml b/...	No
8377	CHANGES.rst	CHANGES.rst	2c1b30d0503cfb064f1cb252e6614a06915a362a	1292419ddfc6a14fc7f85b5ed7efcc2d215f1ad3	release version 3.1.2	diff --git a/CHANGES.rst b/CHANGES.rst\nindex ...	diff --git a/CHANGES.rst b/CHANGES.rst\nindex ...	No
8378	pyproject.toml	pyproject.toml	2c1b30d0503cfb064f1cb252e6614a06915a362a	1292419ddfc6a14fc7f85b5ed7efcc2d215f1ad3	release version 3.1.2	diff --git a/pyproject.toml b/pyproject.toml\n...	diff --git a/pyproject.toml b/pyproject.toml\n...	No
8379	uv.lock	uv.lock	2c1b30d0503cfb064f1cb252e6614a06915a362a	1292419ddfc6a14fc7f85b5ed7efcc2d215f1ad3	release version 3.1.2	diff --git a/uv.lock b/uv.lock\nindex 3f8ba3e8...	diff --git a/uv.lock b/uv.lock\nindex 3f8ba3e8...	No

Dataset Preview

Out of the 95,506 entries, around 4,000 (4.2%) had discrepancies between Myers and Histogram diffs, which can be seen in the following image.



```
Discrepancy Counts:
Discrepancy
No      91508
Yes      3998
Name: count, dtype: int64
```

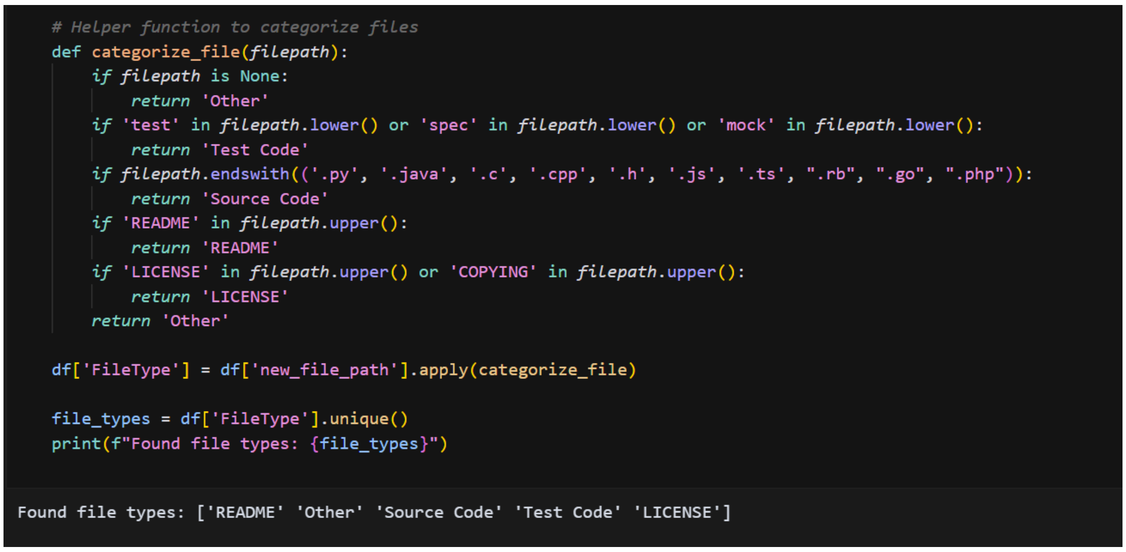
Discrepancy Distribution

1.2.3 File Type Categorization and Statistics

Notebook link: lab4/diff_analyse.ipynb

I categorized files into:

- **Source Code** (extensions: .py, .java, .c, .cpp, .h, .js, .ts, .rb, .go, .php)
- **Test Code** (paths containing `test`, `spec`, or `mock`)
- **README** (files named README)
- **LICENSE** (files named LICENSE or COPYING)
- **Other** (all remaining files)

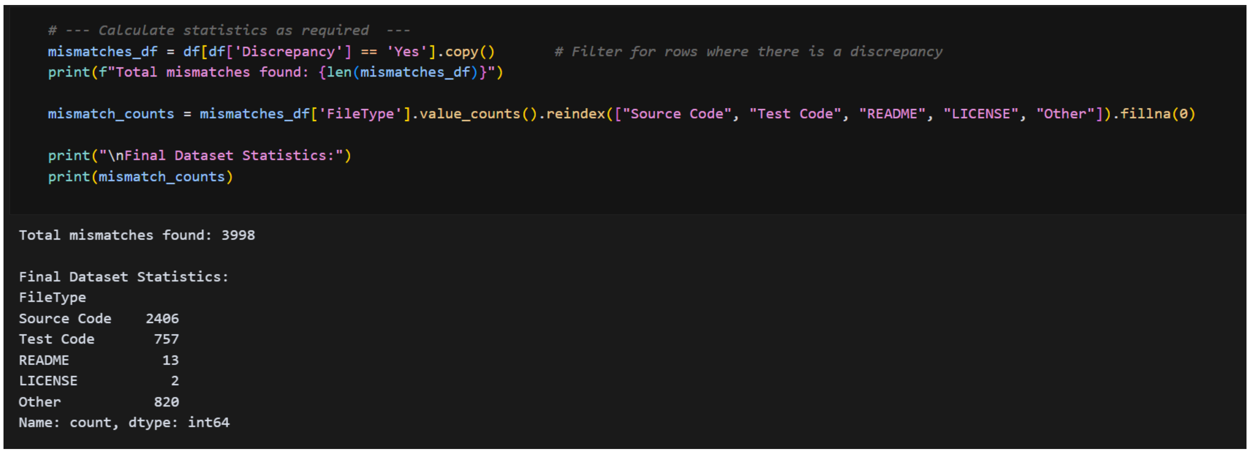


```python
# Helper function to categorize files
def categorize_file(filepath):
    if filepath is None:
        return 'Other'
    if 'test' in filepath.lower() or 'spec' in filepath.lower() or 'mock' in filepath.lower():
        return 'Test Code'
    if filepath.endswith(('.py', '.java', '.c', '.cpp', '.h', '.js', '.ts', ".rb", ".go", ".php")):
        return 'Source Code'
    if 'README' in filepath.upper():
        return 'README'
    if 'LICENSE' in filepath.upper() or 'COPYING' in filepath.upper():
        return 'LICENSE'
    return 'Other'

df['FileType'] = df['new_file_path'].apply(categorize_file)

file_types = df['FileType'].unique()
print(f"Found file types: {file_types}")
```

```
Found file types: ['README' 'Other' 'Source Code' 'Test Code' 'LICENSE']
```

Code Snippet for File Categorization

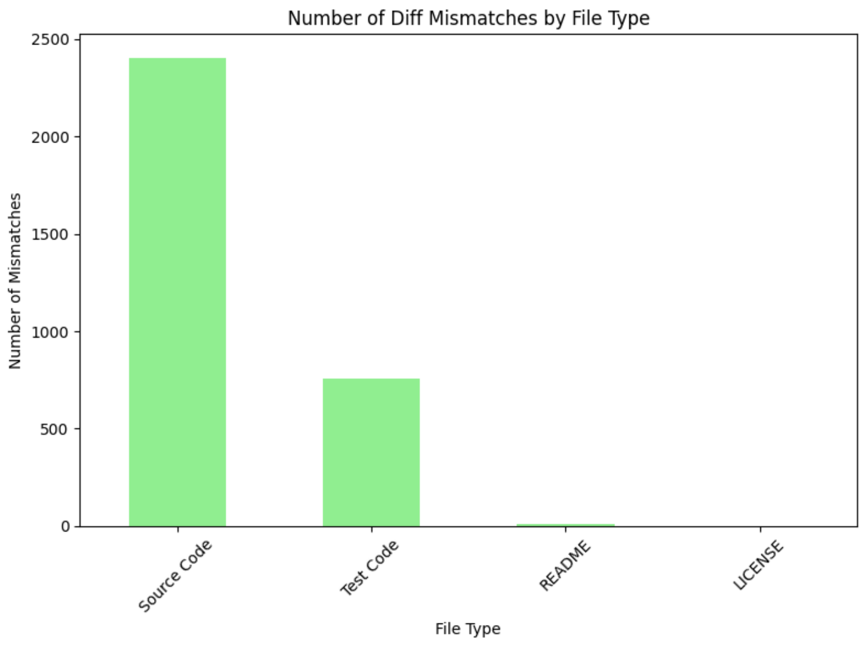
From rows where `Discrepancy == 'Yes'`, I counted mismatch counts per category.

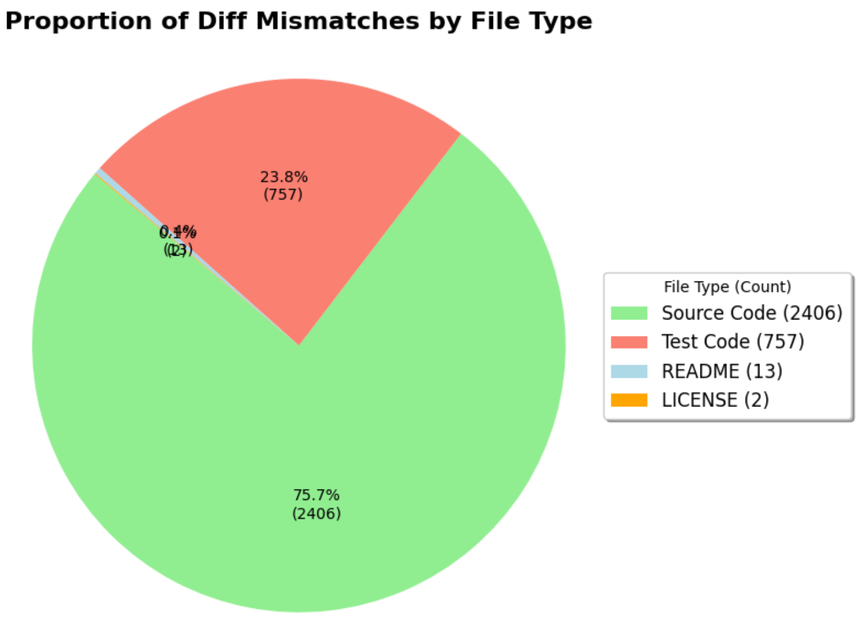


```
    # --- Calculate statistics as required  ---
    mismatches_df = df[df['Discrepancy'] == 'Yes'].copy()       # Filter for rows where there is a discrepancy
    print(f"Total mismatches found: {len(mismatches_df)}")

    mismatch_counts = mismatches_df['FileType'].value_counts().reindex(["Source Code", "Test Code", "README", "LICENSE", "Other"]).fillna(0)

    print("\nFinal Dataset Statistics:")
    print(mismatch_counts)
```

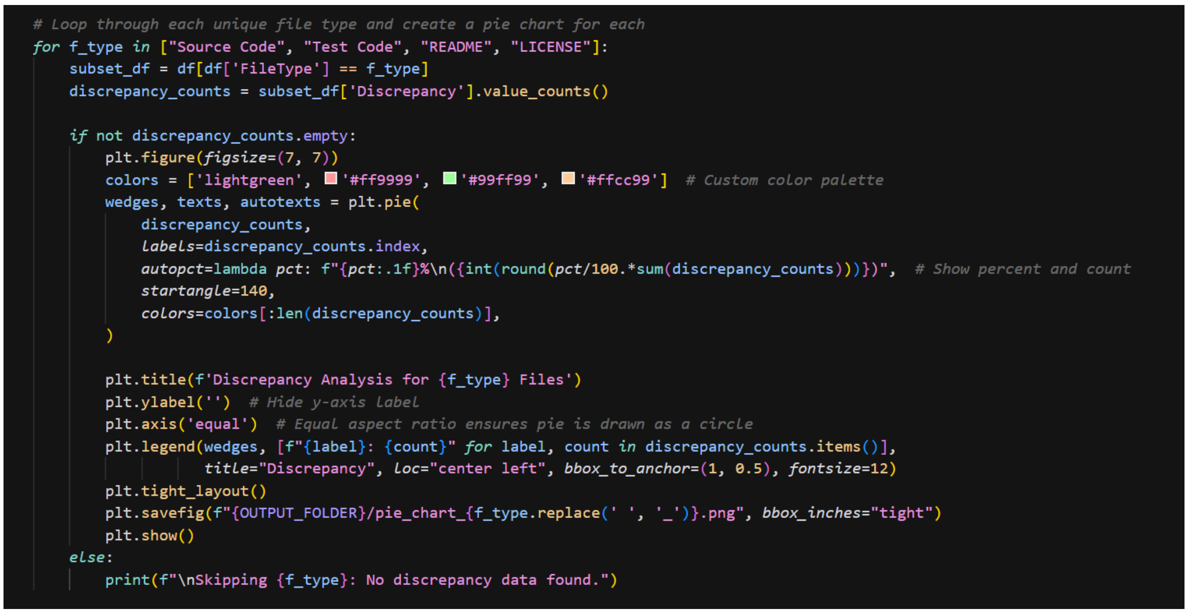
```
Total mismatches found: 3998

Final Dataset Statistics:
FileType
Source Code    2406
Test Code       757
README           13
LICENSE           2
Other           820
Name: count, dtype: int64
```

Mismatch Counts



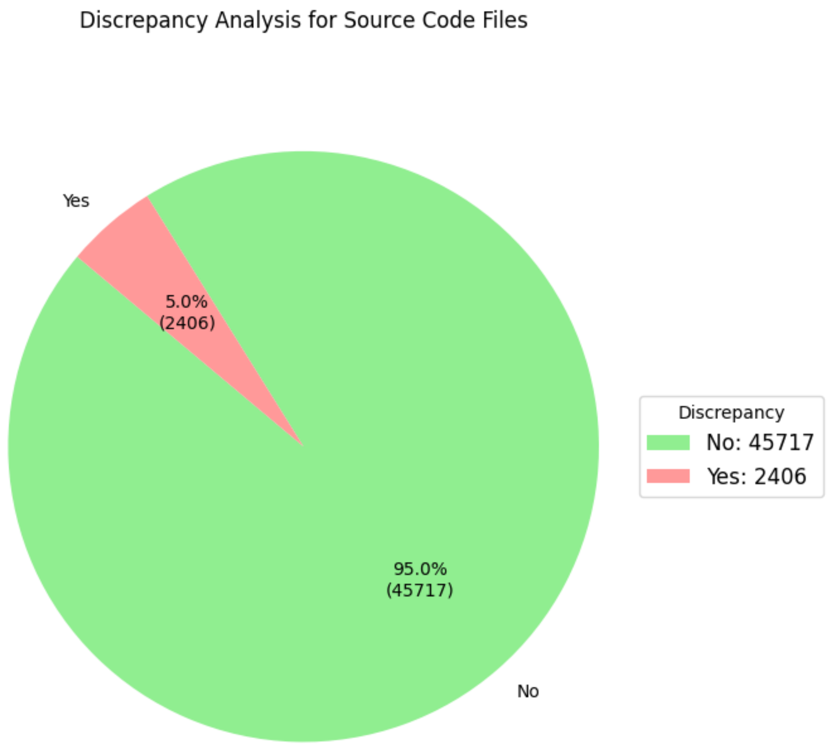


Finally, I plotted discrepancy distribution pie charts for each file type (Source Code, Test Code, README, and LICENSE) to visually compare how often the two diff algorithms disagreed across different categories.

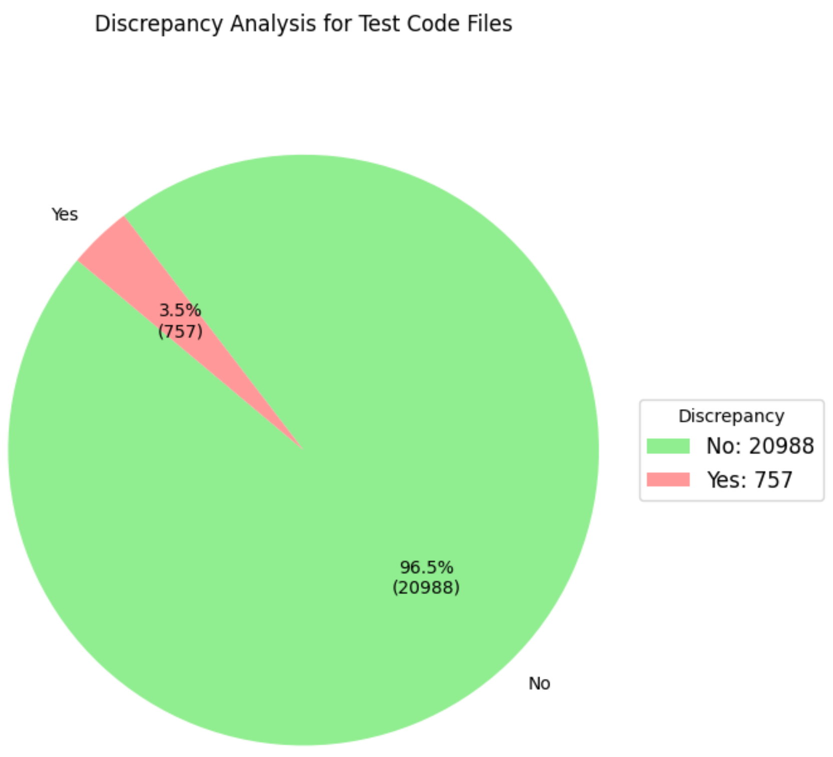


```python
# Loop through each unique file type and create a pie chart for each
for f_type in ["Source Code", "Test Code", "README", "LICENSE"]:
    subset_df = df[df['FileType'] == f_type]
    discrepancy_counts = subset_df['Discrepancy'].value_counts()

    if not discrepancy_counts.empty:
        plt.figure(figsize=(7, 7))
        colors = ['lightgreen', '#ff9999', '#99ff99', '#ffcc99']  # Custom color palette
        wedges, texts, autotexts = plt.pie(
            discrepancy_counts,
            labels=discrepancy_counts.index,
            autopct=lambda pct: f"{pct:.1f}%\n({int(round(pct/100.*sum(discrepancy_counts)))})",  # Show percent and count
            startangle=140,
            colors=colors[:len(discrepancy_counts)],
        )

        plt.title(f'Discrepancy Analysis for {f_type} Files')
        plt.ylabel('')  # Hide y-axis label
        plt.axis('equal')  # Equal aspect ratio ensures pie is drawn as a circle
        plt.legend(wedges, [f"{label}: {count}" for label, count in discrepancy_counts.items()],
                   title="Discrepancy", loc="center left", bbox_to_anchor=(1, 0.5), fontsize=12)
        plt.tight_layout()
        plt.savefig(f"{OUTPUT_FOLDER}/pie_chart_{f_type.replace(' ', '_')}.png", bbox_inches="tight")
        plt.show()
    else:
        print(f"\nSkipping {f_type}: No discrepancy data found.")
```

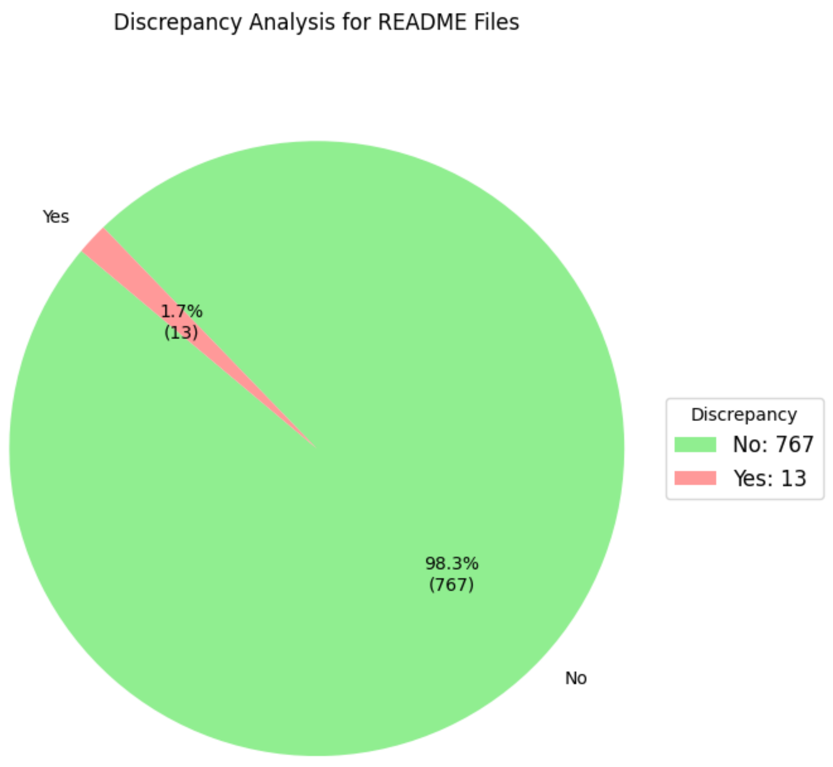
Code Snippet for Plotting File-wise Pie Charts



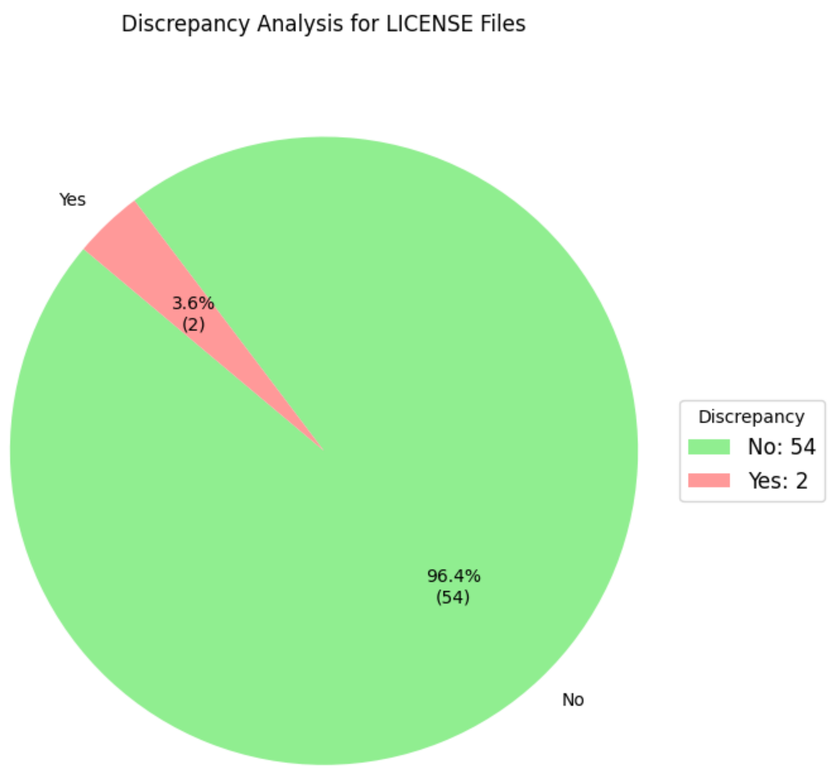
Source Code Pie Chart



Test Code Pie Chart



README Pie Chart



LICENSE Pie Chart

1.3 How to decide which performs better? (Myers vs Histogram)

If I had to automatically determine which algorithm performed better, I would not rely only on the raw mismatch count as we can't figure out which algorithm performed better with just that metric. Instead, I would try to design a small evaluation framework around the usefulness of the produced diff. The basic idea is that a "better" diff should be more consistent, easier to read, and should align with the way developers usually expect changes to be shown.

My approach would be:

1. **Define quality metrics:** For each diff, measure aspects such as number of changed lines reported, length of the diff, and whether moved blocks are detected cleanly. For example, Histogram shows block movements more clearly, while Myers sometimes fragments them into multiple small changes.
2. **Cross-check with file type:** For source code files, I would give preference to the algorithm that minimizes noise (e.g., fewer redundant changes). For text-heavy files like README or LICENSE, correctness is simpler, so the shorter diff is often better.
3. **Ground truth validation:** On a smaller subset, one could ask developers to label which diff looks clearer or more accurate. These labels could then serve as training data for an automatic scoring function.
4. **Automated scoring:** Using the above, each diff can be assigned a score (e.g., based on readability, compactness, and block preservation). Summing these scores across thousands of files would give an aggregate measure of which algorithm is generally more effective.

1.4 Results and Analysis

- Total number of modified file diffs analyzed: **95506**
- Number of mismatches found: **3998**
- **Overall discrepancy rate:** Around **4.2%** of all modified file diffs showed a mismatch between Myers and Histogram.

File Type	Mismatches
Source Code	**2406**
Test Code	**757**
README	**12**
LICENSE	**2**
Other	**820**

- **Source code files** had the most mismatches, since Histogram groups reordered/moved blocks differently than Myers.
- **Test files** also showed mismatches, often due to block movements or repeated patterns.
- **README and LICENSE files** had very few mismatches, especially after ignoring whitespace/blank lines.

1.5 Discussion and Conclusion

1.5.1 Challenges

Working with the full commit histories turned out to be quite time-consuming, especially for large projects like scikit-learn. Some commits were huge and took a long time to process, and I also ran into a few issues while saving the results into CSV files. To get around the encoding errors, I had to use surrogate escapes, which made the process more stable. On the positive side, adding the flags to ignore whitespace and blank lines really helped – it reduced a lot of unnecessary noise in the diffs and made the comparisons cleaner.

1.5.2 What I learned

Through this lab, I got a hands-on understanding of how git diffs actually work and why they matter in practice. Until now, I had only used `git diff` without thinking much about the algorithm behind it. I learned that Git mainly uses two algorithms – Myers and Histogram – and that they can produce different outputs for the same commit.

Working through real repositories helped me see these differences clearly. Myers is the default and works well in most cases, but Histogram often produces cleaner results when code blocks are moved or reordered. This gave me a much clearer picture of how diff algorithms affect the way changes are displayed, and why reviewers or tools might prefer one over the other in certain contexts.

1.6 References

[1] `https://git-scm.com/docs/git-diff`

[2] `https://github.com/ishepard/pydriller`

[3] SEART GitHub Search Engine: `https://seart-ghs.si.usi.ch/`

[4] Lab Document: Google Doc