Analyzing GitHub Issues in the Rails framework

Student: Marouane El Malhi, emalhimarouane@gmail.com
Lecturer: Mohammed Sayagh, mohammed.sayagh@etsmtl.ca

In this section, I will explain how I extracted data from the Rails repository on GitHub. The use of GitHub CLI ‘gh‘ and ‘jq‘ has been inspired by [1].

Task 1: Data Extraction

(a) **Setting up GitHub CLI and jq**

First, I installed the necessary packages to help me extract issues data from GitHub:

(i) *GitHub CLI (‘gh‘)*: This tool allows me to interact with the GitHub API from the command line.

(ii) *jq*: A tool to handle JSON data.

After installation, I created a GitHub access token and saved it in the file ‘my-token.txt‘. To login to my GitHub account using this token, I run the following command:

```
gh auth login --with-token < mytoken.txt
```

This command connects my system to my GitHub account using the token.

(b) **Extracting Issues**

I created the shell script ‘fetch_rails_issues.sh‘ to fetch all issues from the Rails repository. Using GitHub CLI to get issues in all states (both open and closed) using the ‘state = all‘ parameter. The code in my shell script automatically saves the extracted data in a JSON file.

(c) **Data Preprocessing**

Next I loaded the JSON data into my Jupyter notebook. The data contained many fields, so I removed the irrelevant fields to keep only what I needed for analysis. I then sorted the issues by the created_at‘ field and selected the last 500 issues.

(d) **Data Organization**

To simplify the data I separated it into three different dataframes, one for users, one for labels and one for issues. I kept primary keys and foreign keys to easily join the data later when needed.

In the next section we will discover the evolution of issues over time.

[1] GitHub Script Source: https://github.com/joshjohanning/github-misc-scripts/blob/main/gh-cli/find-attachments-in-repositories.sh

Task 2: How do the number of issues evolve over time?

To discover how the number of issues evolves over time, I plotted the number of issues by year. The figure 1 shows my line chart.
As we can see in the chart: From 2012 to 2016, the number of issues went up and down. In 2012, there were 3 issues, which increased to 7 in 2013. However, it dropped to 0 in 2014, then rose again to 2 in 2015, and returned to 0 in 2016.
Starting from 2017, the number of issues has consistently increased. There was 1 issue in 2017, 5 in 2018, 5 in 2019, 7 in 2020, the numbers continued to rise: 9 in 2021, 15 in 2022, 44 in 2023 and 402 in 2024.

I explain the lower number of issues in earlier years compared to recent years by while I focused on the last 500 issues based on the updated date. The choice of using the updated date rather than the created date has made because I thought if an old issue is updated recently, it suggests that it is still relevant and become an issue in the current date. This could happen if new context is added making it important for the community again. Therefore, it is normal that not all old issues will be updated and become new ones, which helps explain the fewer numbers of those issues.

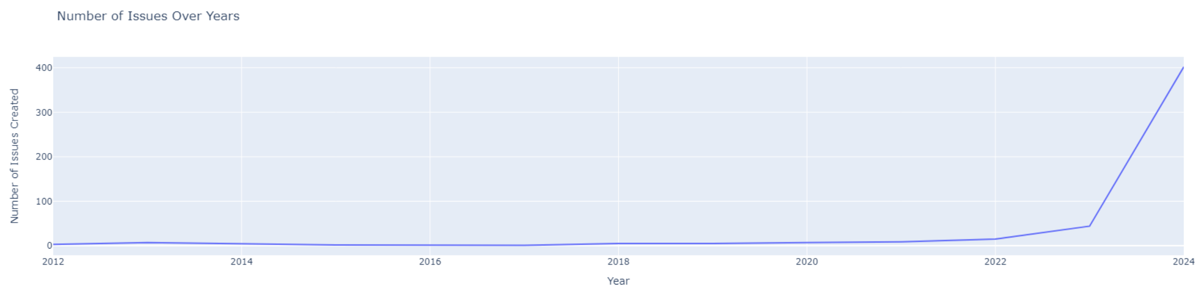

Figure 1: Number of issues by year.

In the upcoming section we will see if are there any periods in which we get more issues?

Task 3: Are there any periods in which we get more issues?

In this task, I plotted the number of issues by month (figure 2) and by quarter (figure 3).
The number of issues changes each month. In January, there are about 15 issues which increases in February and March. Then drops in April returning to 15 issues. The highest number of issues occurs in September with 252 issues. After September, the number decreases. October having the lowest count (only 7 issues). This shows that more issues are created in the middle of the year, especially in August (72 issues) and September. While the numbers are lowest in October and at the beginning and end of the year.

The number of issues also changes across the quarters of the year. In the first quarter (January to March) there are around 51 issues. The second quarter (April to June) the number increase to 66 issues. In the third quarter (July to September) the number of issues peaks at about 346. Finally, in the fourth quarter (October to December) the number decreases to its minimum value around 37 issues.

This result aligns with what we observed with the monthly data. The highest number of issues occurs in August and September (both in the third quarter). The lowest number is in October, which is in the fourth quarter. This indicates that more issues are submitted in the middle of the year (third quarter), while fewer are submitted in the last part of the year (fourth quarter). We can conclude that this community is more active in the middle of the year.

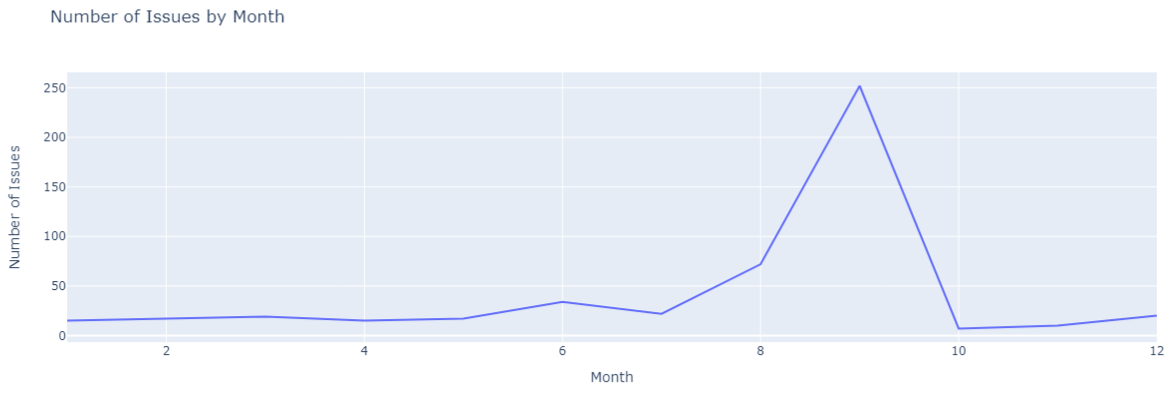

Figure 2: Number of issues by month.

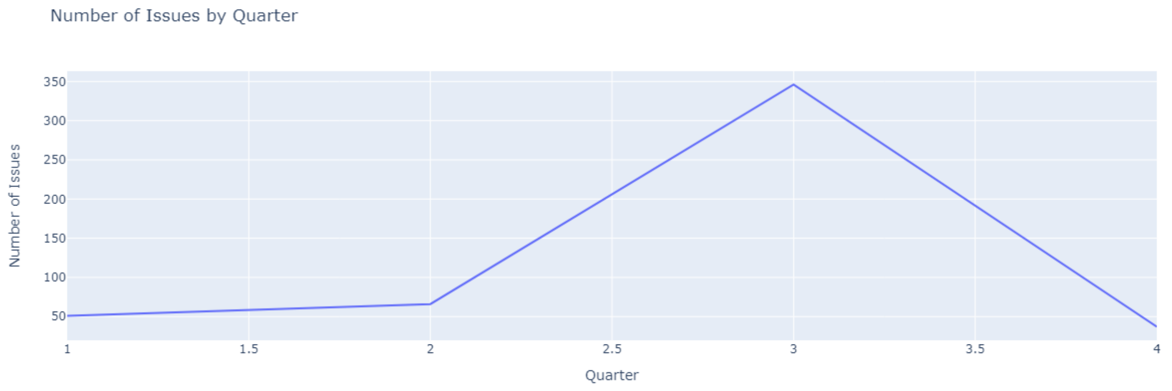

Figure 3: Number of issues by quarter.

In this section we will move to analyze users side and discover if is there anyone who reports more issues than others?

Task 4: Is there anyone who reports more issues than others?

To answer this question, I plotted two charts. The first chart, figure 4 shows the number of issues reported by each user. The second chart, figure 5, groups users by issue counts.
The results indicate that most users report only 1 issue. Out of 297 unique users, 233 have submitted just 1 issue. A smaller group of 34 users reported 2 issues each, and only 10 users

reported 3 issues. As the number of issues increases, fewer users are involved. For example, only 1 user reported 10 issues, while the user with the most issues submitted a total of 25. This shows that the majority of users are occasional reporters, while only a small number of users consistently report a higher number of issues.

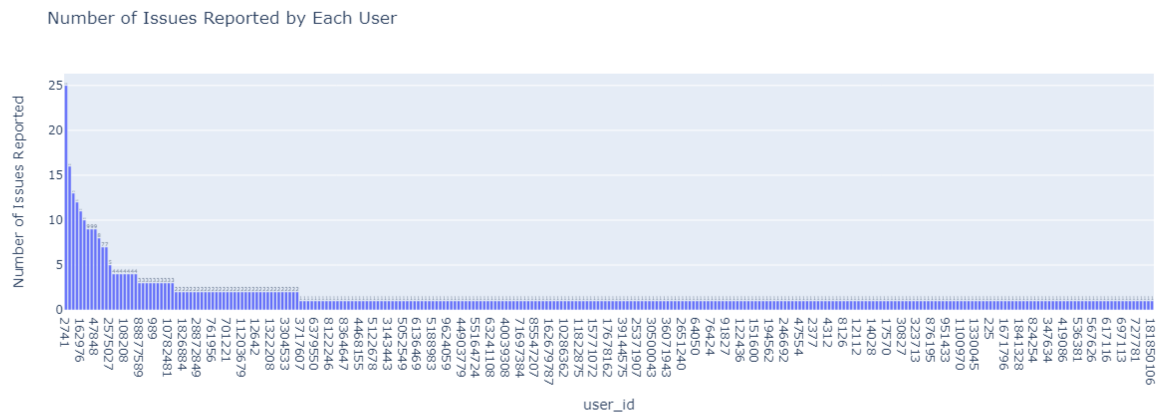

Figure 4: Number of issues reported by each user.

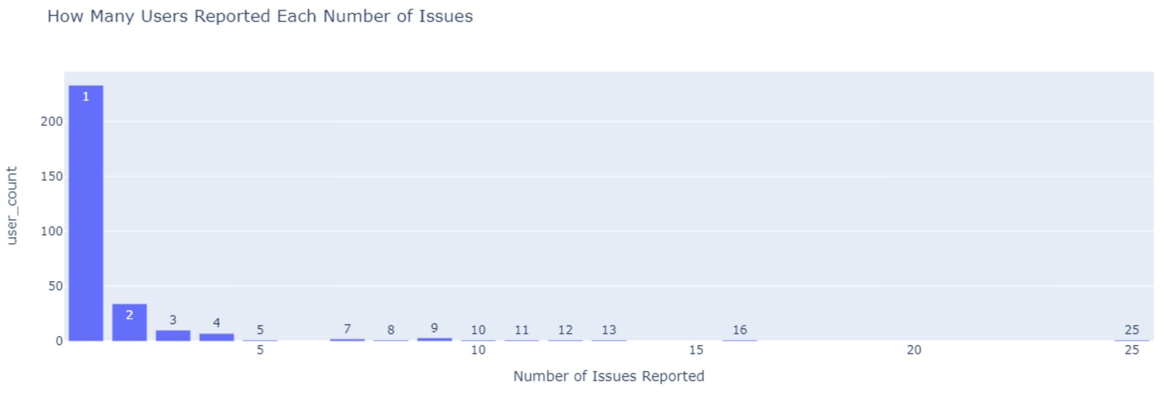

Figure 5: Number of users with specific issue counts.

Task 5: What is the most popular category (label)?

To determine the most popular category (label), I counted the number of occurrences of each label. The results clearly show that the most popular label is **activerecord**, with **140** occurrences.

- *activerecord(140), railties(103), docs(69), actionpack(51), activesupport(42)*: 40
- *actionview, With reproduction steps*: 23
- *stale, activestorage*: 21

- *attached PR, activemodel*: 17
- *ready(16), activejob(14), actionmailbox(13), actiontext(12), actioncable(11)*: between 16 and 11
- *actionmailer, pinned*: 8
- *PostgreSQL(7), third party issue(6), rails foundation(5), more-information-needed(4), needs feedback(3)*: between 7 and 3
- *good first issue, ci issues*: 2
- *encryption, accepted, regression, needs backport, needs work, engines, asset pipeline, security*: 1

This illustrates the distribution of occurrences among various labels, with **activerecord** leading significantly.

In this section, I will suggest and answer a question that I came up with myself.
My question is: How does a user's profile (such as their followers, following, group memberships etc.) impact the results of their reported issues? This includes factors like how quickly issues are resolved, whether they are resolved at all, and the level of community interaction they receive.

Task 6: Impact of User Profile on Issue Reporting

To explore the question, **How does a user's profile (like their followers, who they follow, and group memberships) impact the results of their reported issues? This includes how quickly issues are resolved, if they're resolved, and how much others interact with them?**, I analyzed the relationship between user profile metrics and issue metrics such as **resolution success, closure time, and community engagement**.

(a) **Motivation:** Upon extracting the data, I discovered that some information is hidden behind URLs like `followers_url`, `following_url`, `starred_url`, `organizations_url`, and `repos_url`. I began to consider how this hidden information might relate to our issues metrics. Specifically, I hypothesized that a user's profile could influence the resolution success of an issue, the closure time for resolved issues, and community engagement metrics such as comments and reactions.

(b) **Resolution success:** The first aspect I examined was resolution success. The results indicate that the number of followers, following, organizations, repositories, and starred repositories significantly impact the resolution success of an issue. As these numbers increase tend to have the issues closed more frequently (see Figure 6).

conclusion: Being more active and connected on GitHub could increase the chances of your issues being successfully resolved.

(c) **Closure time:** Next, I investigated closure time for resolved issues. The analysis shows that closed issues tend to be closed faster for users who have more followers, follow more people, belong to more organizations, manage more repositories, and have more starred repositories. The linear regression analysis reveals the following slopes (see 7 and Table 1). Being part of more organizations speeds up issue closure times, with a strong slope of -7.38. In comparison, having more followers, following others, and having more repositories or starred repositories also helps reduce closure times, but not as much.

conclusion: Active and well-connected users in the GitHub community not only have a higher chance of issue resolution but also a quicker resolutions.

(d) **Community engagement:** Finally, I investigated community engagement by examining interactions such as comments and reactions on reported issues. Figure 8 presents a pair plot illustrating that as the number of followers, following, organizations, repositories, and starred repositories increases, so does community interaction with issues.

The following tables summarizes the slopes and P-values of user metrics related to engagement (see Tables 2 and 3).

We see varied effects depending on user metrics. Users with higher metrics like followers and organizations receive more positive reactions such as 'heart', 'rocket' and 'hooray' and support from the community. However, metrics like 'confused' and '-1' reactions tend to decrease as user metrics increase, indicating that more active users may also attract confusion in their interactions. **conclusion:** When a user follows more people, it usually leads to less engagement overall. This means that users with a high number of people they follow have a lower attention or value of their interactions. Regarding comments, as users' metrics increase, the number of comments they get tends to go down. This might mean that well-known users receive fewer, but more useful comments that focus on quality rather than quantity. This is important because it shows that the community gives more thoughtful attention to active users.

In conclusion, active GitHub users, along with an engaged community, tend to generate more relevant and interesting issues that are resolved and interacted with positively and quickly.

User Metric	**Slope**
num_followers	-1.5195
num_following	-2.7615
num_starred_repos	-1.8698
num_repositories	-1.9705
num_organizations	-7.3772

Table 1: Slopes of closure time regression analysis.

In the upcoming section, I will use two models from HuggingFace to automatically classify issues in a Rails project based on their descriptions

Task 7: Use a model from HuggingFace to automatically classify issues in a Rails project based on their descriptions

The Notebook can be found at: Kaggle Notebook

(a) **Data Preparation:** In the data preparation phase, I first filtered the dataset to only include issues where no fields were empty. This left us with 411 issues out of the original

User Metric	followers	following	organizations	repositories	starred_repos
comments	-0.0652	-0.1163	-0.0015	-0.1779	-0.1180
confused	-2.1934	-0.8898	-0.2866	-1.5862	-1.9158
eyes	0.3417	-0.4542	0.0973	-0.6635	-0.1498
heart	0.5999	-0.0292	0.1028	0.0371	0.3293
hooray	0.8503	-0.0607	0.1356	0.1231	0.4980
laugh	0.0000	0.0000	0.0000	0.0000	0.0000
-1	-0.0175	-0.3369	0.0270	-0.5193	-0.2647
+1	0.2803	-0.0472	0.0475	-0.0085	0.1402
rocket	1.6420	-0.1739	0.2832	-0.0881	0.8058
reactions_total	0.2649	-0.0267	0.0449	0.0067	0.1402

Table 2: Slopes: user metrics and their impact on community engagement.

User Metric	followers	following	organizations	repositories	starred_repos
comments	0.3053	0.0016	0.8782	0.0009	0.0367
confused	0.6780	0.7721	0.7242	0.7239	0.6835
eyes	0.7693	0.5024	0.5869	0.5026	0.8850
heart	3.4409e-09	0.6271	3.7485e-11	0.6728	0.0003
hooray	2.7343e-04	0.6568	1.5875e-04	0.5377	0.0170
laugh	NaN	NaN	NaN	NaN	NaN
minus1	0.9919	0.7388	0.9195	0.7251	0.8640
plus1	5.2004e-04	0.3173	1.2971e-04	0.9019	0.0519
rocket	4.1865e-05	0.4591	4.0860e-06	0.7975	0.0246
reactions_total	5.3555e-09	0.3196	1.0291e-10	0.8639	0.0006

Table 3: P-values: user metrics and their impact on community engagement.

500 and 33 unique labels. To simplify the classification task, I mapped the 33 labels into 15 labels by merging similar labels into a unified tag. Below is the mapping used to consolidate the labels Figure.9. Then, I filtered out labels that appeared fewer than 7 times as they could be considered outliers.

(b) **First Model:** The first model I used is the zero-shot classification model 'facebook/bart-large-mnli' [a]. Using two T4 GPUs from Kaggle notebooks, I accelerated the inference process. I used the 8 valid labels from the dataset as candidate labels and for each input type (title, body, and title+body), the model computed the probability percentage for each label.

In the evaluation step, I vary the threshold and chose the labels with scores above the threshold. If more than 5 labels had scores above the threshold, I selected only the top 5 labels with the highest scores, since the maximum number of labels per issue in my data was 5. Precision, recall, and F1 scores were then calculated. The results are summarized in the following table.4

(c) **Second Model:** The second model I used is the cross-encoder model 'cross-encoder/nli-deberta-v3-small' [b].

For this model, I created pairs for each input type with all possible labels. Specifically, for each issue, I paired the input text with each candidate label in the following way:

- For a *text* input, I created pairs like:

  for *label* in candidate labels:

  ($text$, "This issue is about *label*").

This allows the model to predict a class from three categories: ‘contradiction‘, ‘entailment‘, and ‘neutral‘.

In the evaluation step, I extracted valid labels from the predicted scores, specifically selecting those with a ‘neutral‘ or ‘entailment‘ relationship with the input text. I then calculated precision, recall, and F1 score for each input type.

The average precision, recall, and F1 scores for each input type are summarized in the table.5.

(d) **Discussion:** The results show differing performances between the two models. The zero-shot classification model, `facebook/bart-large-mnli`, performed better, especially with the title and body combined, achieving an F1 Score of **0.3512**, It had a high recall of **0.8368**, indicating it identified many relevant cases, this suggests that the title and body complement each other, providing more context. Its precision was highest at **0.2437** with the title input, meaning that the body add more context that let the model predict false labels. the model performed best with both inputs combined: title + body $\implies$ title $\implies$ body.

On the other hand, the `cross-encoder/nli-deberta-v3-small` model had lower scores overall. Its best F1 Score was **0.2733** for the title input, demonstrating that the title carried the most informative content for this model as well. Here, the importance of inputs is ranked as title $\implies$ title + body $\implies$ body, indicating that while the title is crucial. This suggests that the cross-encoder model may require further optimization or additional training data to improve its ability to leverage contextual information effectively.

[a]Huggingface facebook/bart-large-mnli link
[b]Huggingface cross-encoder/nli-deberta-v3-small link

Input Type	Threshold	F1 Score	Precision	Recall
title + body	0.0610	**0.3512**	0.2314	**0.8368**
title	0.0630	0.3456	**0.2437**	0.7055
body	0.0610	0.3347	0.2207	0.7934

Table 4: Zero-shot classification results using ‘facebook/bart-large-mnli‘

Input Type	F1 Score	Precision	Recall
title + body	0.1750	0.1327	0.3857
title	**0.2733**	**0.1859**	**0.6741**
body	0.1688	0.1310	0.3667

Table 5: Classification results using ‘cross-encoder/nli-deberta-v3-small‘

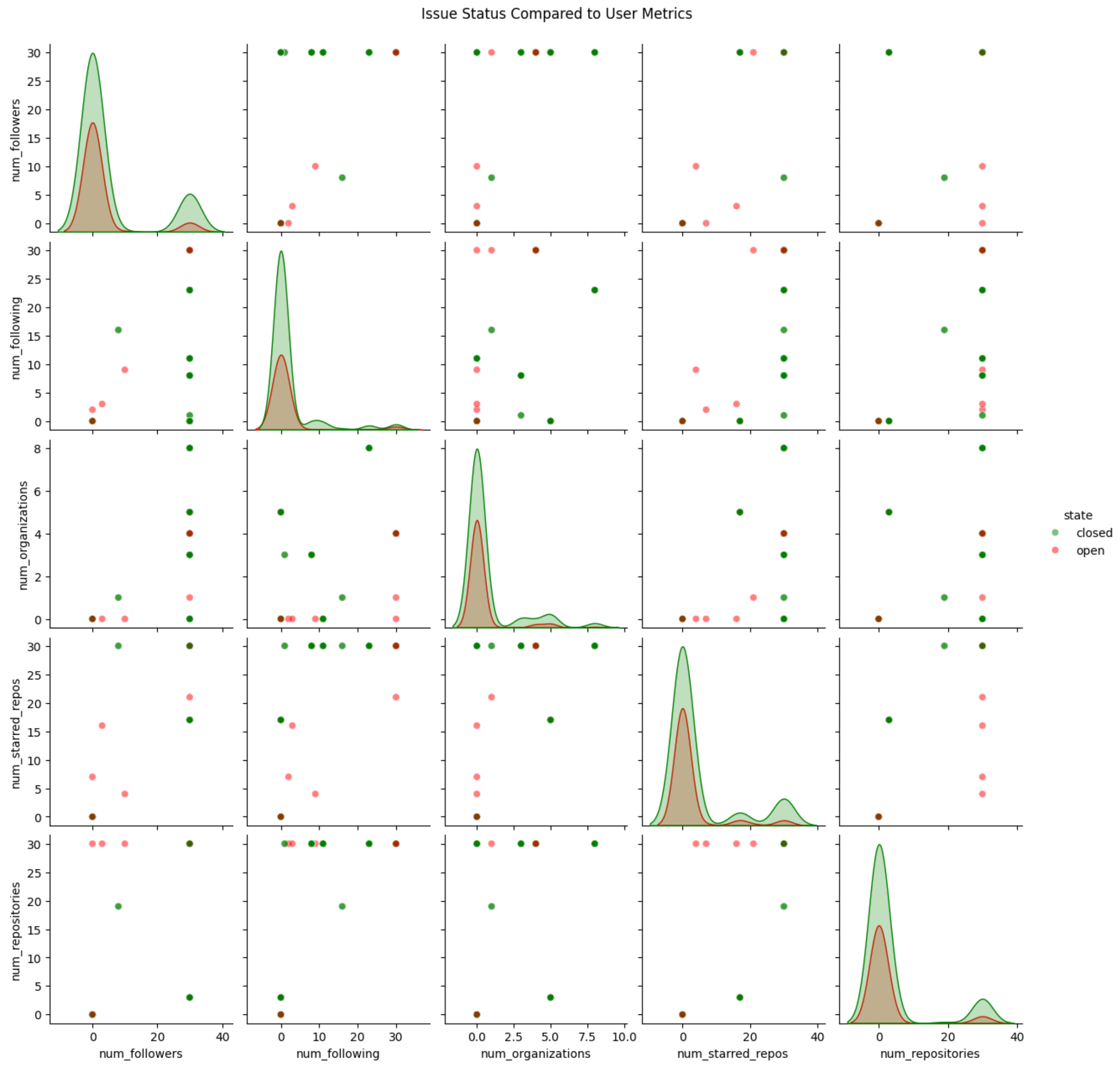

Figure 6: Impact of user profile metrics on resolution success.

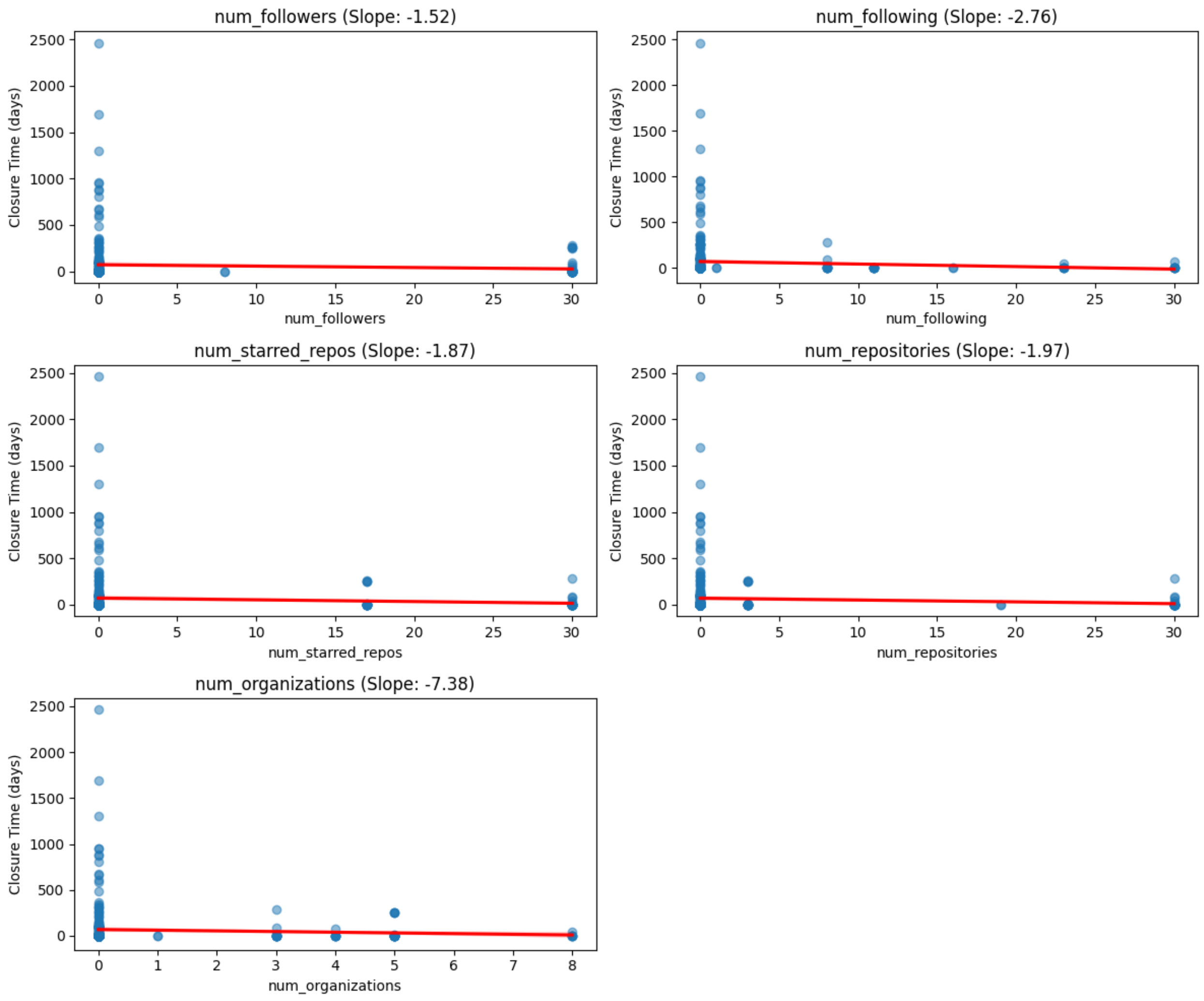

Figure 7: Closure time vs. user profile metrics.

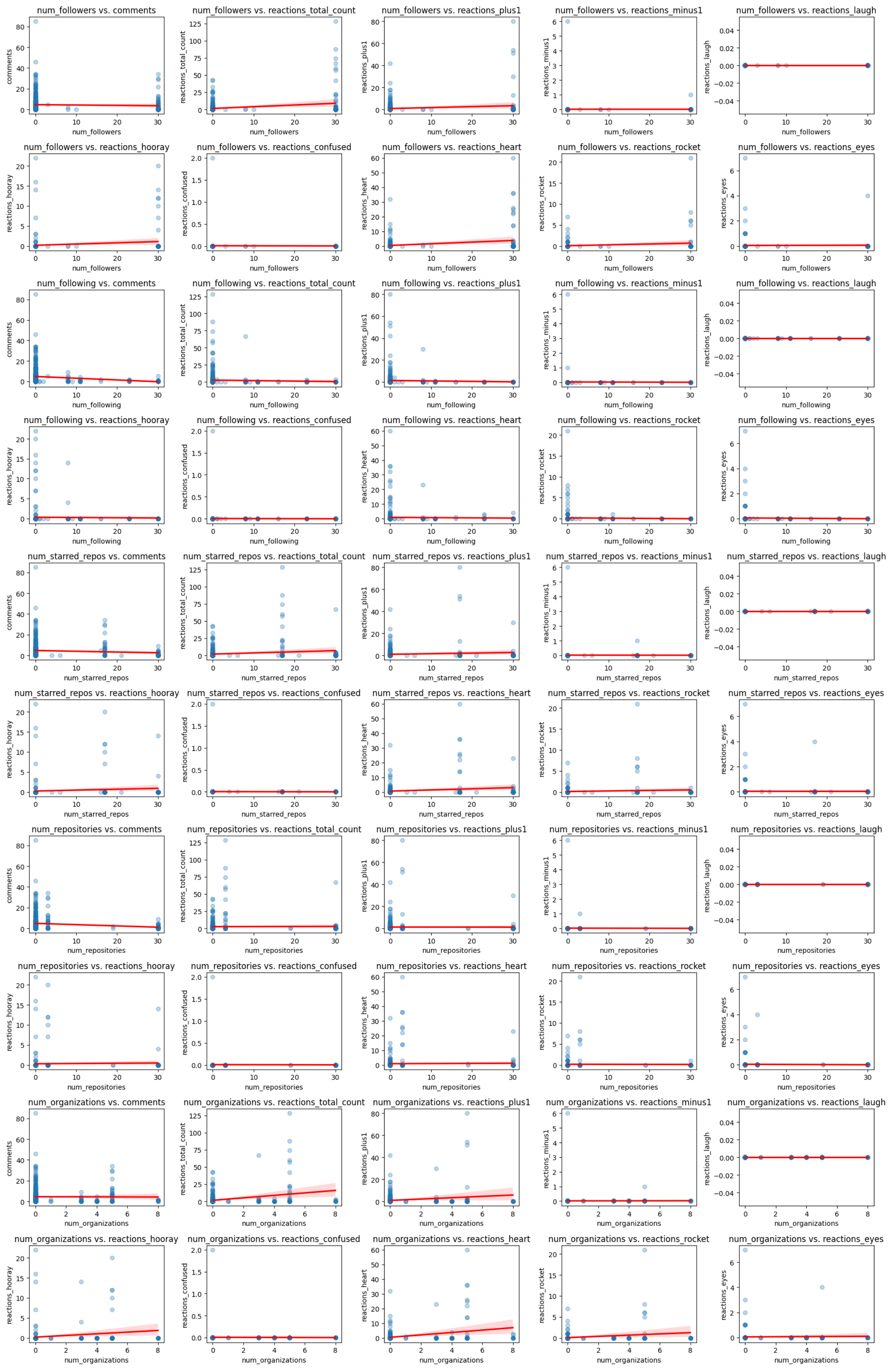

Figure 8: Community engagement metrics based on user profiles.

```
label_mapping = {
    'activerecord': 'active',
    'actionpack': 'action',
    'actionview': 'action',
    'actionmailer': 'action',
    'activemodel': 'active',
    'activesupport': 'active',
    'activestorage': 'active',
    'actionmailbox': 'action',
    'actiontext': 'action',
    'actioncable': 'action',
    'railties': 'railties',
    'With reproduction steps': 'Issue Handling',
    'attached PR': 'Issue Handling',
    'stale': 'Issue Status',
    'pinned': 'Issue Status',
    'ready': 'Issue Status',
    'more-information-needed': 'Feedback Needed',
    'needs feedback': 'Feedback Needed',
    'good first issue': 'Contribution Welcoming',
    'needs backport': 'Contribution Welcoming',
    'needs work': 'Contribution Welcoming',
    'docs': 'docs',
    'security': 'security',
    'PostgreSQL': 'PostgreSQL',
    'third party issue': 'third party issue',
    'ci issues': 'ci issues',
    'regression': 'regression',
    'asset pipeline': 'asset pipeline',
    'engines': 'engines',
    'encryption': 'security',
    'accepted': 'Issue Status'
}
```

Figure 9: Label Mapping Dictionary