Paper OS13

Maximizing GitHub APIs: Journey to Open-Source Metrics

Alanah Jonas, GSK, London, United Kingdom

Abstract:

Over recent years, many pharma companies have been adopting open-source technologies. But how can we effectively measure our progress on this journey? At GSK we have been using GitHub for code development in our study reporting. This paper will explore the innovative utilisation of GitHub APIs as a powerful tool for extracting information from GitHub to enable us to track and analyse open-source efforts. By harnessing the capabilities of GitHub APIs, we can gain invaluable insights into open-source language usage in Biostatistics as a whole and by disease area. Attendees will discover how GSK extracts pertinent information from GitHub repositories to generate comprehensive language usage metrics. By leveraging GitHub APIs, we empower our decision-makers with actionable insights to make changes to drive open-source adoption more effectively.

Introduction

In recent years, many pharma companies have been increasingly adopting open-source technologies. The shift towards these technologies has been driven by the need for greater flexibility, innovation, and collaboration. However, as open-source tools become more prevalent, a crucial question arises: how can organizations effectively measure their progress in this adoption?

The Journey of Open Source in the Pharma Industry

The pharmaceutical industry's journey towards open-source adoption has been gradual yet significant. Historically, the industry relied heavily on proprietary software for data analysis and reporting. However, as the benefits of open-source tools became more apparent—such as cost savings, increased collaboration, and the ability to customize tools to meet specific needs—more companies began exploring these options. Over the years, the use of open-source tools in pharma has grown, with R emerging as a key player. This timeline of adoption highlights the industry’s shift towards embracing open-source solutions to enhance efficiency and innovation.

How Much R Is Being Used in GSK Biostatistics?

At GSK, the adoption of R has been a critical focus. As we increasingly integrate R into our workflows, it becomes essential to understand the extent of its usage. How do we know if teams are adopting R effectively? Are there gaps in adoption that need to be addressed? These questions are vital for ensuring that we are fully leveraging R's capabilities to improve our study reporting processes.

Tracking R Adoption and Identifying Training Needs

One of the challenges in monitoring the adoption of R is determining how widespread its use is across different teams and studies. It’s important not only to track which teams are using R but also to identify those

that may need additional training or support. Recognizing teams that are successfully adopting R allows us to celebrate their progress and share best practices, while identifying teams that are struggling helps us target them for further support and training.

The adoption of R and other open-source tools is transforming the pharmaceutical industry by enabling professionals to streamline their processes, enhance data analysis, and improve regulatory submissions. As we continue this transition, having a clear understanding of where and how R is being used within our organization is crucial.

The Solution: GitHub APIs

To address these challenges, we have turned to GitHub APIs. GitHub, a web-based platform widely used for version control and collaborative software development, plays a central role in our code development processes. At GSK, GitHub has been integral to our study reporting workflows, where it serves as a repository for our code and a hub for collaboration among our biostatistics teams.

Following work done on another project, the idea emerged to use GitHub APIs as a tool to extract information about programming language usage from our repositories. This approach allows us to measure open-source adoption, particularly the use of R, in a more systematic and automated way. By utilizing the GitHub API, we aim to create an automated KPI dashboard that tracks the adoption of R across our studies, providing insights that were previously difficult to obtain through manual methods.

What is GitHub?

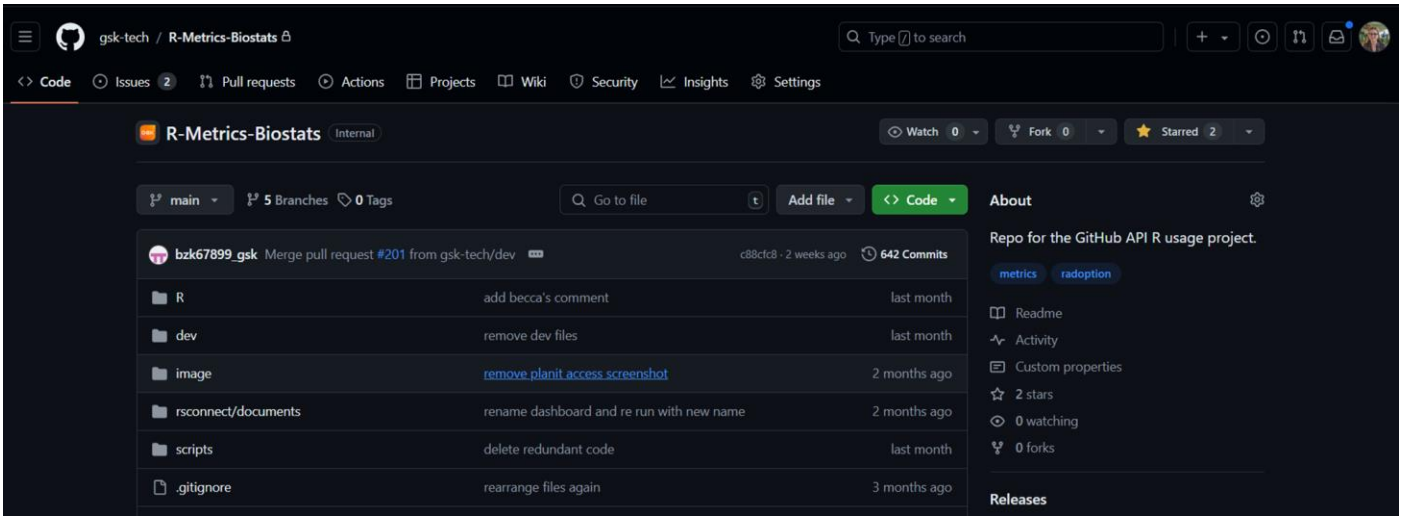
GitHub is a web-based platform that facilitates version control and collaborative software development. It allows developers to manage code, track changes, and work together on projects efficiently. The platform's key features include:

- **Version Control:** GitHub keeps track of every change made to the codebase, enabling developers to easily revert to previous versions if needed.
- **Collaboration:** Multiple developers can work on the same project simultaneously, with tools for pull requests, code reviews, and discussion.
- **Repository (Repo):** A centralized storage location where your code resides, making it accessible to all team members.
- **Branching & Merging:** Developers can work on different features or bug fixes in isolation (branches) and merge them back into the main codebase when ready.

Why Use GitHub?

GitHub offers several advantages for software development, including:

- **Collaboration:** It simplifies working together on a project, even when team members are in different locations.
- **Transparency:** GitHub provides a clear history of code changes, decisions, and contributions, making it easier to track progress and understand the rationale behind decisions.
- **Security:** The platform offers options for private repositories and integrates with various security tools to protect your code.
- **Integration:** GitHub connects with a wide range of tools for continuous integration, deployment, and more, streamlining the development process.



What are APIs/GitHub APIs?

What is an API?

An API, or Application Programming Interface, is a set of rules and tools that allow different software applications to communicate with each other. To understand an API, think of it as a waiter in a restaurant. You (the application) tell the waiter (the API) what you want from the kitchen (another application), and the waiter brings it to you. APIs enable different software systems to interact seamlessly, exchanging data and commands.

What is the GitHub API?

The GitHub API allows developers to interact with GitHub programmatically, enabling them to access data and features of GitHub within their own applications. This provides a way to automate tasks, gather data, and integrate GitHub features into other applications or services. GitHub offers two main APIs for interacting with its platform: the REST API and the GraphQL API. Both have their own use cases, advantages, and limitations. Here's a summary of the differences between them and when to use each one:

REST API

1. **Architecture**: Based on REST (Representational State Transfer) principles, it uses standard HTTP methods like GET, POST, PUT, and DELETE.
2. **Endpoints**: Has a fixed set of endpoints, each designed for a specific purpose. For example, GET /repos/{owner}/{repo} retrieves repository information.
3. **Data Retrieval**: Returns all available data from an endpoint. If you only need specific fields, you'll still receive the full response.
4. **Pagination**: Paged responses with a default limit, often requiring multiple requests for large data sets.
5. **Ease of Use**: Easier for beginners and for simple, straightforward tasks. You can test it easily using standard HTTP clients or curl.
6. **Rate Limits**: Rate limits are generally lower than GraphQL, but limits depend on the authenticated user and their role (authenticated vs. unauthenticated).

When to Use the REST API:

- When you need a quick and easy way to interact with GitHub.
- For straightforward tasks like fetching a list of repositories or users.
- When working with existing tooling or libraries that are designed for REST.
- When you need stable, well-documented endpoints for specific tasks.

GraphQL API

1. **Architecture**: Based on GraphQL, a flexible query language that allows clients to specify the structure of the response.
2. **Endpoints**: A single endpoint (/graphql) handles all queries, and you can retrieve data from multiple resources in a single request.
3. **Data Retrieval**: You define exactly what data you need, which reduces the size of the response and avoids over-fetching or under-fetching.
4. **Pagination**: Uses cursor-based pagination, which is more efficient and flexible compared to offset-based pagination in REST.
5. **Ease of Use**: Has a steeper learning curve compared to REST, but it's highly powerful and efficient once mastered.
6. **Rate Limits**: Higher rate limits than REST, based on complexity of queries rather than the number of requests.

When to Use the GraphQL API:

- When you need to fetch specific data fields and avoid over-fetching.
- When you want to combine multiple requests into a single query to minimize the number of API calls.
- For complex data structures and relationships that require flexible querying.
- When efficiency and performance are priorities, especially in client-side applications.

Summary

- **REST API**: Simpler and well-suited for straightforward, single-resource tasks. Ideal for quick integrations or when using tools that work better with REST.
- **GraphQL API**: More flexible and efficient, particularly for complex queries involving multiple resources or when you need fine-grained control over the response data.

Why Use GitHub APIs?

GitHub APIs offer several benefits:

- **Automation:** Automate repetitive tasks such as issue tracking, code deployment, or repository management.
- **Integration:** Integrate GitHub data and features into other tools, dashboards, or applications, enhancing your development workflows.

- **Customization:** Build custom workflows or tools tailored to your specific needs using GitHub's data and features.

What Information Can Be Extracted from a Repo?

The GitHub API allows you to extract a wide range of information from a repository, including:

- **Repository Details:** Information about repositories, such as name, description, size, visibility (public/private), and default branch.
- **Files and Contents:** The contents of files and directories within a repository.
- **Commits:** The commit history of a repository, including details like author, message, and changes.
- **Branches & Tags:** Lists of branches and tags within a repository, including information on the latest commit associated with each.
- **Language Usage:** Detailed information about the programming languages used in a repository, including the percentage of code written in each language.
- **Issues:** Details about issues in a repository, including title, description, status (open/closed), assignees, and labels.
- **Pull Requests:** Information on pull requests, such as author, status (open/closed/merged), associated commits, and reviews.
- **Contributors:** A list of contributors to a repository, along with the number of commits each has made.
- **Collaborators:** A list of collaborators on a repository and their roles (e.g., admin, write, read).

The gh Package

The gh package allows you to interact with the GitHub API directly from R. This package is a versatile tool for automating a wide range of tasks related to GitHub, leveraging the API's full power without leaving your R environment. The flexibility of the `gh()` function allows you to craft custom API requests suited to your specific needs.

Getting Started with gh

1. **Installation:**
    - Install the package with `install.packages("gh")`.
    - Load the package with `library(gh)`.
2. **Authentication:**
    - Set up an environment variable called GITHUB_PAT equal to your GitHub Personal Access Token (PAT).
    - Example: `Sys.setenv(GITHUB_PAT ="your_personal_access_token")`.
3. **Making Requests:**
    - Use the `gh()` function to make API requests.

Example Requests Using REST API and the gh package:

1. **Get a Repo:**

Retrieve details about a specific repository here using the R-Metrics-Biostats repo.

REST API call:

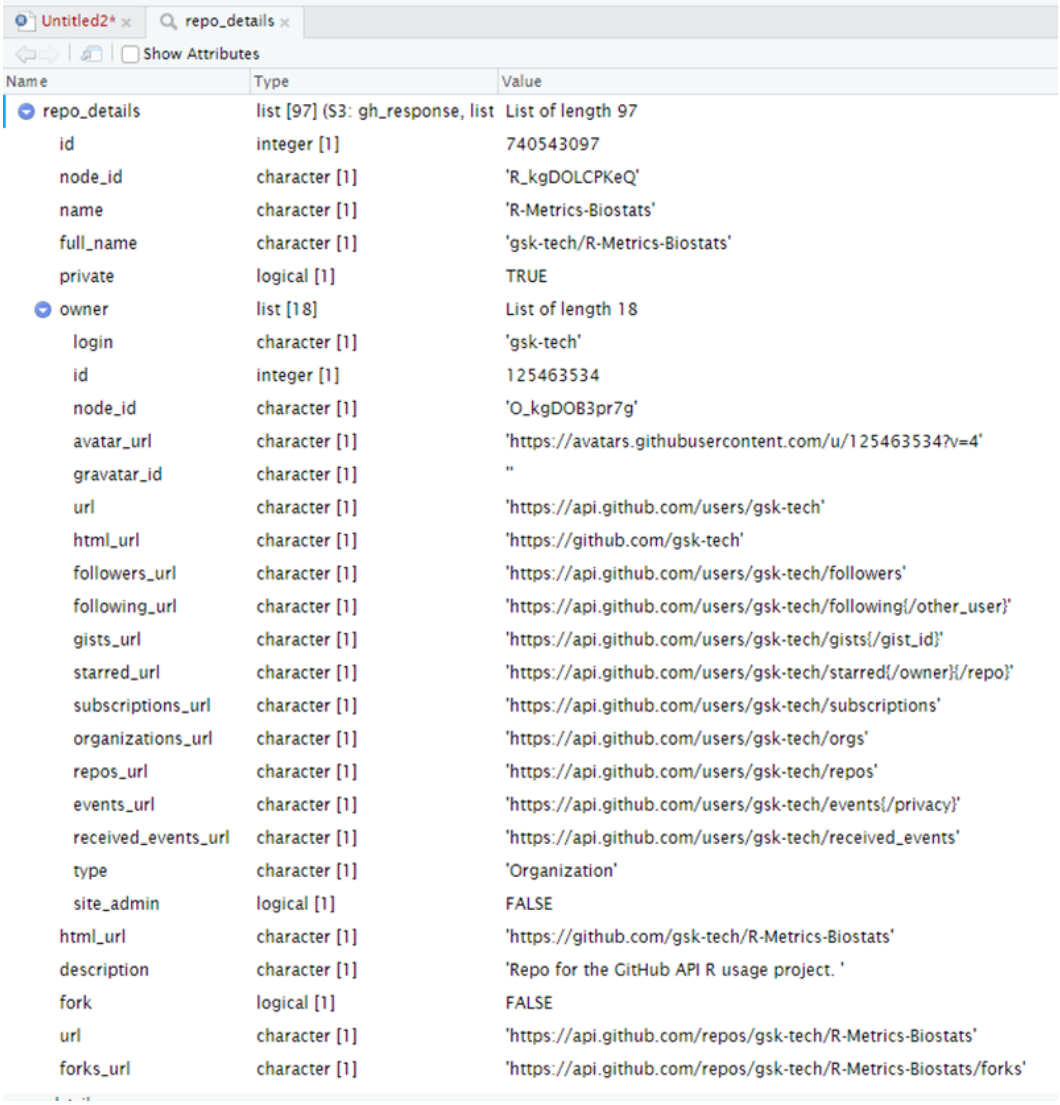
GET /repos/{owner}/{repo}

Load the required packages gh and tidyverse.

```
library(gh)
library(tidyverse)
```

We then run the gh function with repo set to “R-Metrics-Biostats” and the owner of all our internal biostats repos being “gsk-tech” so I set that as the owner. This will then give us all the repo details:

```
repo_details <- gh("/repos/gsk-tech/R-Metrics-Biostats")
```



Untitled2* | repo_details

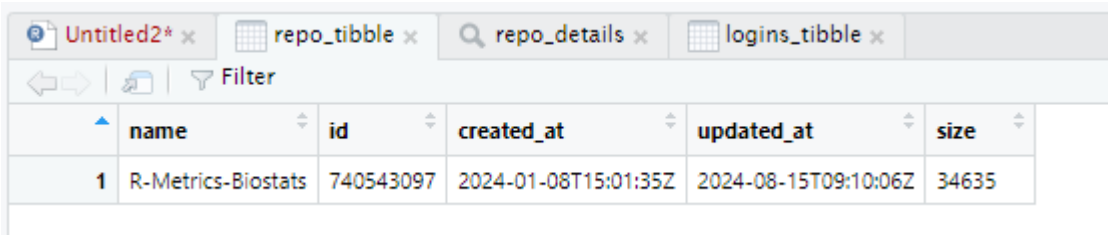
Show Attributes

Name	Type	Value
repo_details	list [97] (S3: gh_response, list	List of length 97
id	integer [1]	740543097
node_id	character [1]	'R_kgDOLCPKeQ'
name	character [1]	'R-Metrics-Biostats'
full_name	character [1]	'gsk-tech/R-Metrics-Biostats'
private	logical [1]	TRUE
owner	list [18]	List of length 18
login	character [1]	'gsk-tech'
id	integer [1]	125463534
node_id	character [1]	'O_kgDOB3pr7g'
avatar_url	character [1]	'https://avatars.githubusercontent.com/u/125463534?v=4'
gravatar_id	character [1]	''
url	character [1]	'https://api.github.com/users/gsk-tech'
html_url	character [1]	'https://github.com/gsk-tech'
followers_url	character [1]	'https://api.github.com/users/gsk-tech/followers'
following_url	character [1]	'https://api.github.com/users/gsk-tech/following{/other_user}'
gists_url	character [1]	'https://api.github.com/users/gsk-tech/gists{/gist_id}'
starred_url	character [1]	'https://api.github.com/users/gsk-tech/starred{/owner}{/repo}'
subscriptions_url	character [1]	'https://api.github.com/users/gsk-tech/subscriptions'
organizations_url	character [1]	'https://api.github.com/users/gsk-tech/orgs'
repos_url	character [1]	'https://api.github.com/users/gsk-tech/repos'
events_url	character [1]	'https://api.github.com/users/gsk-tech/events{/privacy}'
received_events_url	character [1]	'https://api.github.com/users/gsk-tech/received_events'
type	character [1]	'Organization'
site_admin	logical [1]	FALSE
html_url	character [1]	'https://github.com/gsk-tech/R-Metrics-Biostats'
description	character [1]	'Repo for the GitHub API R usage project. '
fork	logical [1]	FALSE
url	character [1]	'https://api.github.com/repos/gsk-tech/R-Metrics-Biostats'
forks_url	character [1]	'https://api.github.com/repos/gsk-tech/R-Metrics-Biostats/forks'

Then, I’ve decided to keep, the repo name, id, creation and last updated dates and the size of the repo and save those values into a tibble.

```
# Extract the desired fields and convert them into a tibble
repo_tibble <- tibble(
  name = repo_details$name,
  id = repo_details$id,
  created_at = repo_details$created_at,
```

```
  updated_at = repo_details$updated_at,
  size = repo_details$size
)
```



Untitled2* | repo_tibble | repo_details | logins_tibble

Filter

	name	id	created_at	updated_at	size
1	R-Metrics-Biostats	740543097	2024-01-08T15:01:35Z	2024-08-15T09:10:06Z	34635

2. **Get Repo Languages:**

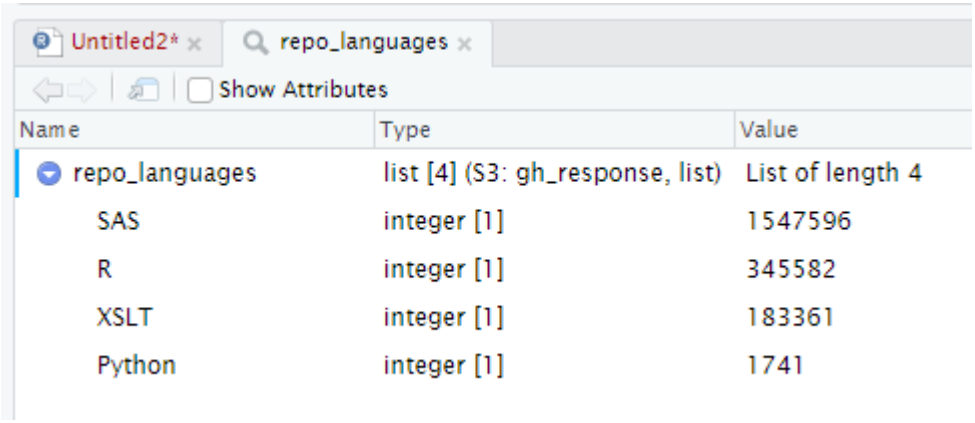
Extract information about the languages used in a repository, here I've removed the name of the repo ive used so replace {repo} with the name of a repo.

REST API call:

GET /repos/{owner}/{repo}/languages

```
repo_languages <- gh("/repos/gsk-tech/{repo}/languages") %>%
  as_tibble() %>%
  pivot_longer(cols = everything(),
               names_to = "language",
               values_to = "bytes")
```

Output from gh call:

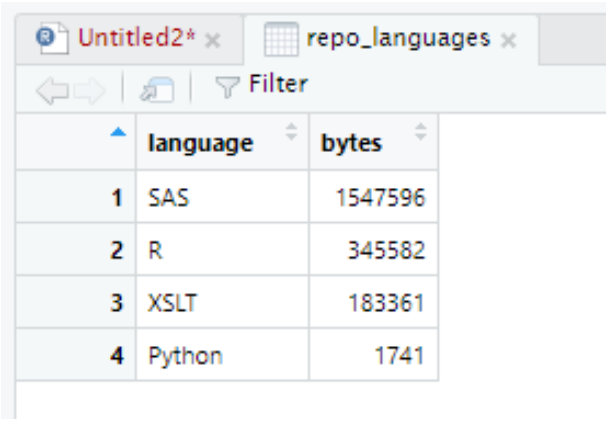


Untitled2* | repo_languages

Show Attributes

Name	Type	Value
repo_languages	list [4] (S3: gh_response, list)	List of length 4
SAS	integer [1]	1547596
R	integer [1]	345582
XSLT	integer [1]	183361
Python	integer [1]	1741

Output after converting to a tibble:



Untitled2* | repo_languages

Filter

	language	bytes
1	SAS	1547596
2	R	345582
3	XSLT	183361
4	Python	1741

3. **Get Repo Contributors:**

List the contributors to a repository and their contributions.

REST API call:

GET /repos/{owner}/{repo}/contributors

```
repo_contributors <- gh("/repos/gsk-tech/R-Metrics-Biostats/contributors")
```

Untitled2* | repo_contributors | repo_languages | repo_details

Show Attributes

Name	Type	Value
repo_contributors	list [4] (S3: gh_response, list)	List of length 4
[[1]]	list [19]	List of length 19
login	character [1]	'nd258262_gsk'
id	integer [1]	153532089
node_id	character [1]	'U_kgDOCSa2uQ'
avatar_url	character [1]	'https://avatars.githubusercontent.com/u/153532089?v=4'
gravatar_id	character [1]	''
url	character [1]	'https://api.github.com/users/nd258262_gsk'
html_url	character [1]	'https://github.com/nd258262_gsk'
followers_url	character [1]	'https://api.github.com/users/nd258262_gsk/followers'
following_url	character [1]	'https://api.github.com/users/nd258262_gsk/following{/other_user}'
gists_url	character [1]	'https://api.github.com/users/nd258262_gsk/gists{/gist_id}'
starred_url	character [1]	'https://api.github.com/users/nd258262_gsk/starred{/owner}{/repo}'
subscriptions_url	character [1]	'https://api.github.com/users/nd258262_gsk/subscriptions'
organizations_url	character [1]	'https://api.github.com/users/nd258262_gsk/orgs'
repos_url	character [1]	'https://api.github.com/users/nd258262_gsk/repos'
events_url	character [1]	'https://api.github.com/users/nd258262_gsk/events{/privacy}'
received_events_url	character [1]	'https://api.github.com/users/nd258262_gsk/received_events'
type	character [1]	'User'
site_admin	logical [1]	FALSE
contributions	integer [1]	113
[[2]]	list [19]	List of length 19
[[3]]	list [19]	List of length 19
[[4]]	list [19]	List of length 19

```
# Extract just the login names and convert to a tibble
logins_tibble <- tibble(contributors = map_chr(repo_contributors, "login"))
```

Untitled2* | logins_tibble

Filter

	contributors
1	nd258262_gsk
2	aj641247_gsk
3	bzk67899_gsk
4	bca99787_gsk

To collect information for multiple repositories, I created individual functions for each API call, such as get_contributors. I then used lapply to apply these functions to a column containing all the repository names (repos_all$repo_name). The results are combined into a single data frame using bind_rows:

```
repos_contributor <- bind_rows(lapply(repos_all$repo_name, get_contributors))
```

This approach allows for easy aggregation of data from multiple repositories in a streamlined manner.

Transformations

- Using GitHub APIs, we have accessed repo data to gather detailed information about various repositories. This included fetching metadata such as repository names, dates, contributors and language usage.
- Merged repo contributors with employee Database to identify repos associated with Biostatistics employees enabling us to identify the Biostats repos.
- Tied repos with studies in GSKs Study Database, using study and project ids, to identify Biostats study repos, so we can distinguish between study and non-study repos and also this enabled us to identify therapy areas for our repos.

We then calculated the following metrics on R usage.

- **Relative R** = R code size out of (R and SAS)
- **Any R** = Total Repos with any usage of R.
- **Substantial R** = These are the repos where R code size (out of R and SAS) is >=10%

Impact

Non studies have higher R usage, due to more flexibility in areas such as tool development and generally staff working in these areas have more open-source programming experience. However, we did find that R is used frequently in studies, but the amount in each is low.

Success, Obstacles, and Next Steps

Successes:

- We successfully used GitHub APIs to gather information about repositories within GSK Biostatistics.
- We eliminated the need for manual reporting of R usage, providing a true sense of how R adoption is progressing.

Obstacles:

- We cannot access private repositories, which limits the comprehensiveness of our analysis.
- It's challenging to determine how many people on a study are actively writing R code.
- Measuring progress over time remains a difficult task.
- We face issues related to API rate limits and processing time.

Next Steps:

- We plan to apply the use of GitHub APIs to other areas, such as assessing the health of our GitHub repositories and building a Biostats Tool Catalogue.

- We will also focus on identifying areas where further training is needed to support R adoption.

Conclusion

This report highlights the increasing importance of open-source technologies, particularly R, within GSK Biostatistics. To effectively measure R adoption and identify areas needing support, we leveraged GitHub APIs. These APIs allowed us to automate the tracking of open-source usage, providing accurate insights without manual effort.

Using GitHub APIs, we've gained a clearer understanding of R adoption, identified teams requiring additional training, and improved our ability to monitor progress. Despite challenges like accessing private repositories and managing API limitations, we've set the stage for broader API applications, such as assessing repository health and building a Biostats Tool Catalogue.

In summary, GitHub APIs have significantly enhanced our ability to track and analyze open-source adoption, enabling us to move beyond manual processes and drive our open-source strategy forward.

REFERENCES

About GitHub and Git:

https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

What is GitHub: https://www.coursera.org/articles/what-is-github

GitHub REST API Documentation: GitHub REST API Docs

GitHub GraphQL API Documentation: GitHub GraphQL API Docs

gh Package: https://gh.r-lib.org/

ACKNOWLEDGMENTS

A big thank you to my colleagues Niladri Dasgupta, Becca Krouse, and Ben Arancibia for their crucial contributions to this GitHub API project.

Contact information:

Your comments and questions are valued and encouraged.

Contact the author at:

Alanah Jonas

GSK HQ

79 New Oxford Street
London
WC1A 1DG
United Kingdom

Email: alanah.x.jonas@gsk.com