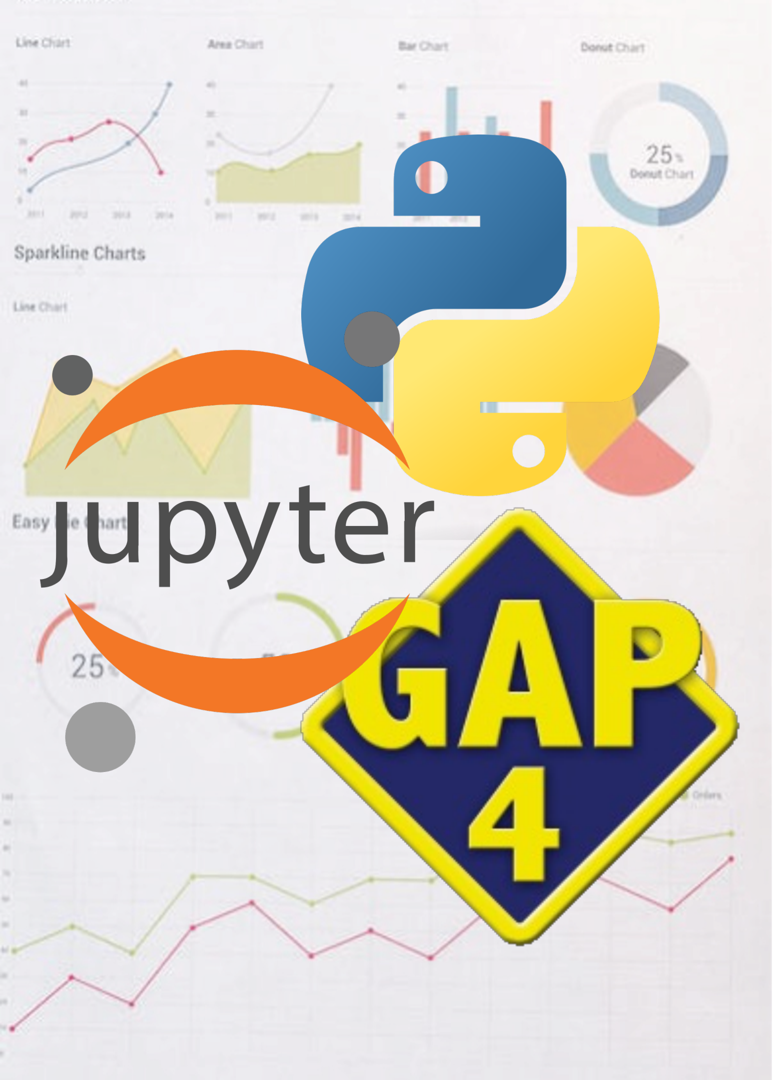


DATA ANALYTICS FRAMEWORK FOR GAP PACKAGE MANAGEMENT

THEA HOLTLUND JACOBSEN, 220024877

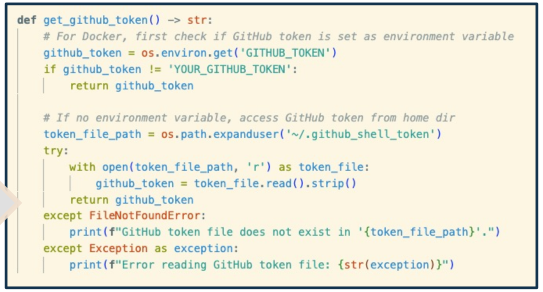
SCHOOL OF COMPUTER SCIENCE, AUGUST 2023

PROJECT OBJECTIVES AND ACHIEVEMENTS

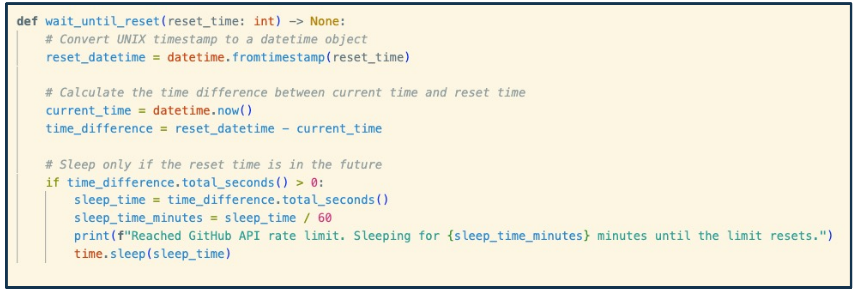
Primary: I.	Collect data, such as repository metadata, commit history, issue data, pull requests, and user information, on GAP packages from GitHub.	Fully achieved
Primary: II.	Customise data outputs to some extent, by leaving room for the user to in some way specify what information they want to extract, changing parameters used to call the data from GitHub so that the overview corresponds to their specific needs.	Partially achieved
Primary: III.	Allow a user to get an overview of the current situation for GAP packages from GitHub, by exploring metrics such as the number of packages matching a set of given criteria, for the collected data.	Fully achieved
Primary: IV.	Allow a user to monitor packages redistributed with GAP, for purposes such as assisting a release manager. Metrics in this sense could include new releases or packages close to release.	Fully achieved
Primary: V.	Allow a user to explore any problems for GAP packages, related to metrics such as issues and pull requests, in a convenient way.	Fully achieved
Secondary: VI.	Create some insight on the GAP package community. Metrics that could be interesting to explore include actions, trends and behaviour of contributors.	Fully achieved
Secondary: VII.	Establish some processes for data cleaning and pre-processing, through measures such as accounting for call limitations, missing values and removing irrelevant entries.	Fully achieved
Secondary: VIII.	Obtain insight on the history of GAP packages, their dynamics and the corresponding community, through historical trends and collaboration patterns.	Partially achieved
Tertiary: IX.	Ensure code can be accessed an executed by all individuals to the greatest extent possible, through tools such as Docker and Binder.	Partially achieved

DATA EXTRACTION

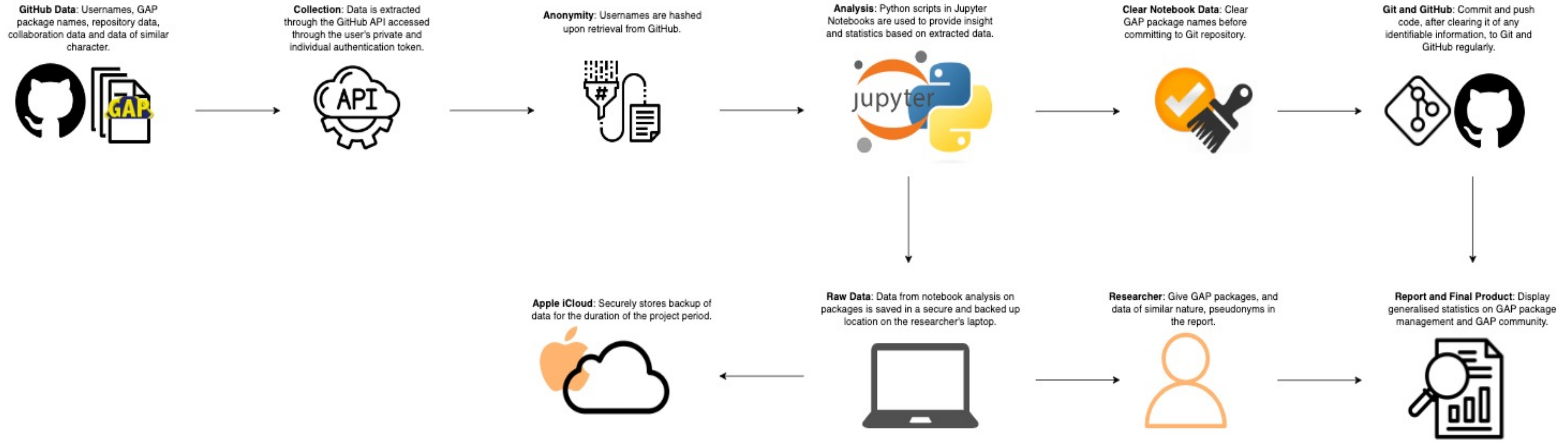
GET DATA · ANALYSE · VISUALISE · DECISIONS



```python
def get_github_token() -> str:
    # For Docker, first check if GitHub token is set as environment variable
    github_token = os.environ.get('GITHUB_TOKEN')
    if github_token != 'YOUR_GITHUB_TOKEN':
        return github_token

    # If no environment variable, access GitHub token from home dir
    token_file_path = os.path.expanduser('~/.github_shell_token')
    try:
        with open(token_file_path, 'r') as token_file:
            github_token = token_file.read().strip()
        return github_token
    except FileNotFoundError:
        print(f"GitHub token file does not exist in '{token_file_path}'.")
    except Exception as exception:
        print(f"Error reading GitHub token file: {str(exception)}")
```



```python
def wait_until_reset(reset_time: int) -> None:
    # Convert UNIX timestamp to a datetime object
    reset_datetime = datetime.fromtimestamp(reset_time)

    # Calculate the time difference between current time and reset time
    current_time = datetime.now()
    time_difference = reset_datetime - current_time

    # Sleep only if the reset time is in the future
    if time_difference.total_seconds() > 0:
        sleep_time = time_difference.total_seconds()
        sleep_time_minutes = sleep_time / 60
        print(f"Reached GitHub API rate limit. Sleeping for {sleep_time_minutes} minutes until the limit resets.")
        time.sleep(sleep_time)
```



GitHub Data: Usernames, GAP package names, repository data, collaboration data and data of similar character.

Collection: Data is extracted through the GitHub API accessed through the user's private and individual authentication token.

Anonymity: Usernames are hashed upon retrieval from GitHub.

Analysis: Python scripts in Jupyter Notebooks are used to provide insight and statistics based on extracted data.

Clear Notebook Data: Clear GAP package names before committing to Git repository.

Git and GitHub: Commit and push code, after clearing it of any identifiable information, to Git and GitHub regularly.

Apple iCloud: Securely stores backup of data for the duration of the project period.

Raw Data: Data from notebook analysis on packages is saved in a secure and backed up location on the researcher's laptop.

Researcher: Give GAP packages, and data of similar nature, pseudonyms in the report.

Report and Final Product: Display generalised statistics on GAP package management and GAP community.

DATA ANALYSIS

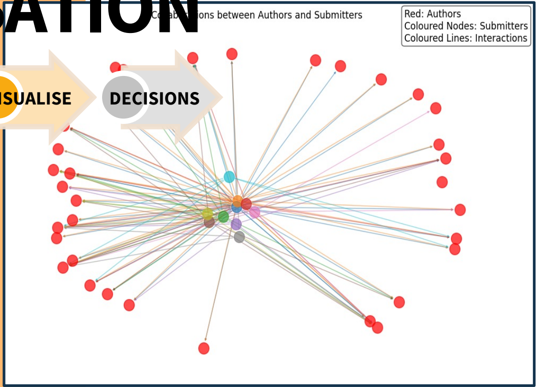
GET DATA · ANALYSE · VISUALISE · DECISIONS

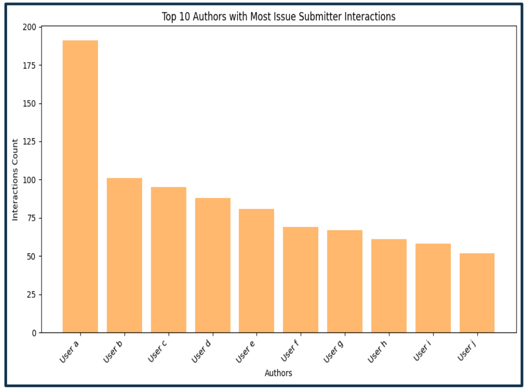
Repository	Test File Count	Tested Versions	Required Version	CI Has Data	PackageInfo Has Data
Repository A	12	4.12, 4.11, 4.10, 4.9	4.11.0	Yes	Yes
Repository B	1	4.11, 4.10, 4.9	4.7.4	No	Yes
Repository C	10	4.12, 4.11	4.8	Yes	Yes
Repository D	6	None	4.5	Yes	Yes
Repository E	3	4.12, 4.11, 4.10, 4.9	4.11.1	No	Yes

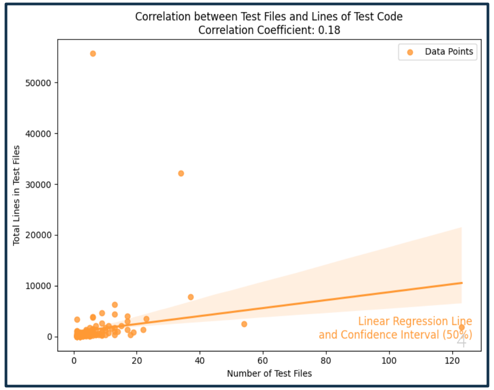
```
Total number of authors for all GAP packages: 132
Total number of submitters for all GAP packages: 192
Total number of authors who were also submitters for all GAP packages: 98
Total number of inactive contributors for all GAP packages: 53
```

DATA VISUALISATION

GET DATA · ANALYSE · VISUALISE · DECISIONS


Co ons between Authors and Submitters
Red: Authors
Coloured Nodes: Submitters
Coloured Lines: Interactions


Top 10 Authors with Most Issue Submitter Interactions
Interactions Count
User a
User b
User c
User d
User e
User f
User g
User h
User i
User j
Authors


Correlation between Test Files and Lines of Test Code
Correlation Coefficient: 0.18
Data Points
Total Lines in Test Files
Number of Test Files
Linear Regression Line
and Confidence Interval (50%)