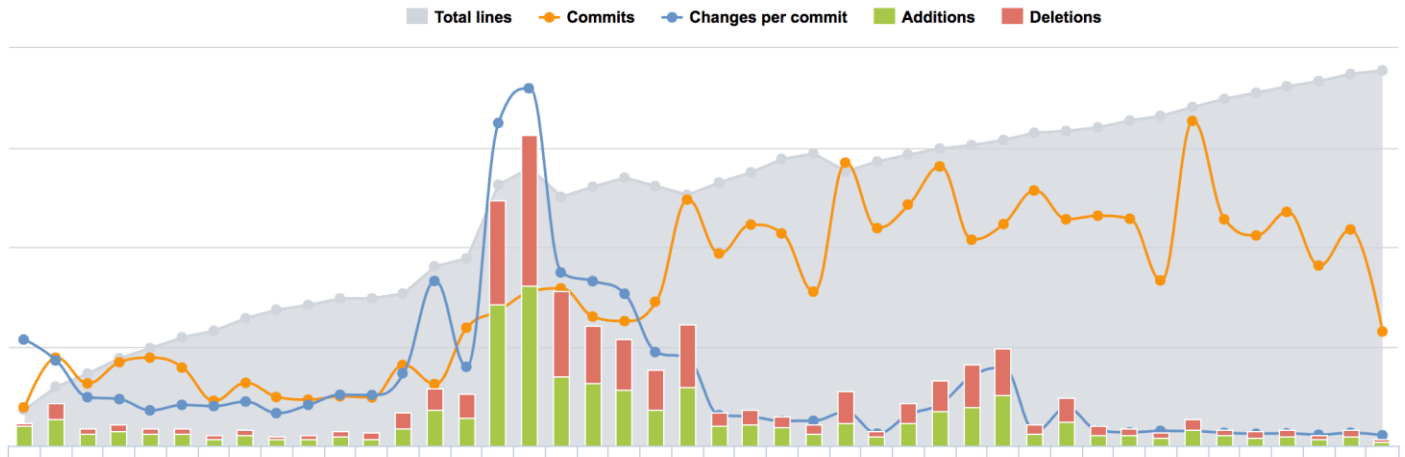


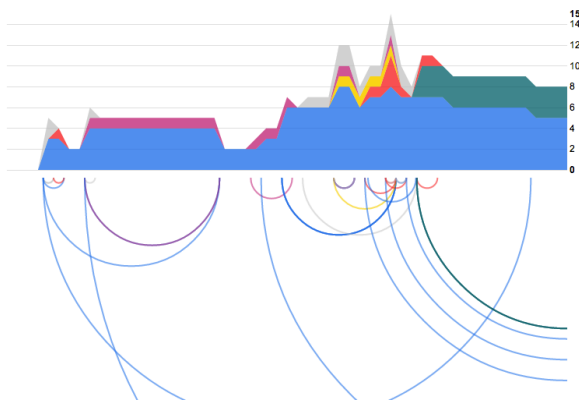
# 1 Introduction

This assignment is to write a solution which fetches data from GitHub, processes it and then visualises it. We have been given free reign of which data we fetch and how we choose to visualise it. I am hoping to be as ambitious as possible so that I can produce an interesting visualisation.

## 2 Research



I came across this graph which is offered by EazyBI. It is aesthetically pleasing and shows plenty of useful information, but it offers metrics found in almost all GitHub analysis solutions. I realised that my solution would have to be a lot different than this as the assignment is not to copy an existing visualisation, but to produce an original one.



I stumbled across this graph, which is meant to visualise GitHub issues. The graph isn't very clear but I found the idea of issue tracking to be useful. I then decided to read through the GitHub API to see can I find anything interesting to visualise.

## Issue

`class github.Issue.Issue`

This class represents Issues. The reference can be found here <https://docs.github.com/en/rest/reference/issues>

**assignee**

Type: `github.NamedUser.NamedUser`

**assignees**

Type: list of `github.NamedUser.NamedUser`

**body**

Type: string

**closed\_at**

Type: `datetime.datetime`

Calls: `GET /repos/{owner}/{repo}/issues/{issue_number}/events`

Return type: `github.PaginatedList.PaginatedList` of `github.IssueEvent.IssueEvent`

**get\_labels()**

Calls: `GET /repos/{owner}/{repo}/issues/{number}/labels`

Return type: `github.PaginatedList.PaginatedList` of `github.Label.Label`

**remove\_from\_assignees(\*assignees)**

Calls: `DELETE /repos/{owner}/{repo}/issues/{number}/assignees`

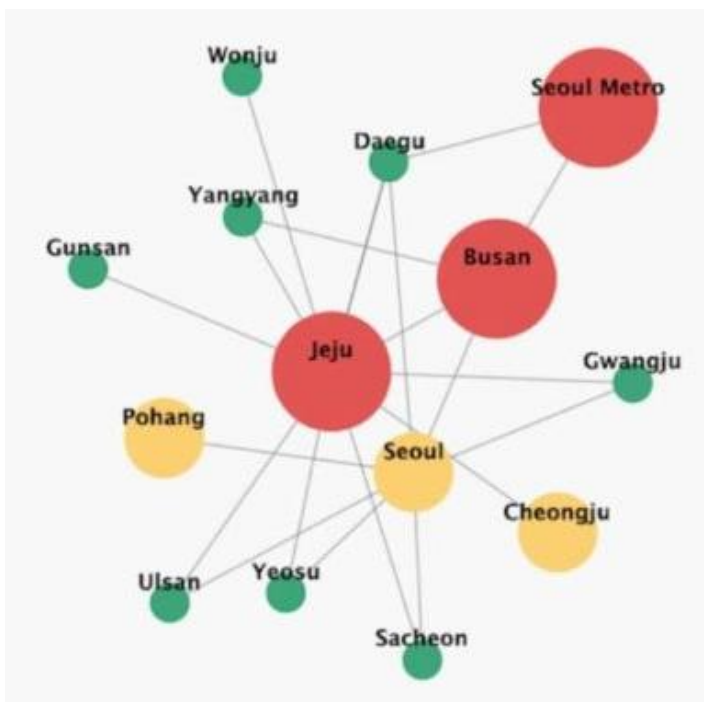
Parameters: `assignee` – `github.NamedUser.NamedUser` or string

Return type: None

I found the `get_labels()` call and realised I could make this central to my visualisation. Many GitHub repositories use labels to separate issues into many categories such by severity, platform, and language. I could visualise this which could help developers see which categories have the most issues.

Issues can have multiple labels such as a severity label followed by a platform one. I realised that I can visualise how these labels connect to each other to give the user more insight.

I then started looking through visualisation packages in Python.



I came across this node visualisation and realised I could use this format to visualise issues. My original plan was to use matplotlib and networkX to do this but realised I should use PyVis as it is more tailored for visualising nodes in Python. Now all I had to do was make this idea a reality.

---





## 2 Approach

I decided to use PyVis for the reasons outlined above and because it has a feature allowing me to choose the best settings to use on the fly before pasting them into the options for the module. My next big decision was on how I was going to store the data. I originally planned to use a mongo database but decided it was overkill for the purpose I needed it for and that I would use SQLite3 instead as it is more straightforward and because of time constraints.

### 2.1 populateDB.py



This part of the program serves the purpose of retrieving the data from GitHub and storing it in a SQLite3 db. It takes the repository address as the input.

It iterates over all of the issues in the repository and totals the number of times each label is used. It also records what labels each label is used in conjunction with. If an issue has no labels, it is put in the “unlabelled” category. Here is what the database output looks like:

Table:  MAIN   



	LABEL	COUNT	NEIGHBOURSBOOL
	Filter	Filter	Filter
1	dependencies	1	0
2	unlabeled	24	0
3	documentation	2	1
4	need feedback	12	1
5	enhancement	21	1
6	bug	13	1
7	feature-request	1	0
8	to-test	8	1
9	help wanted	8	1
10	out-of-scope	3	1
11	support	6	1
12	bug fix	3	1
13	invalid	2	1
14	need-rebase	5	1
15	feature	9	1
16	censorship	2	1

---

Table:  bug 

	LABEL	COUNT
	Filter	Filter
1	need feedback	1
2	help wanted	3

---

Table:  need\_rebase 

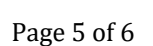
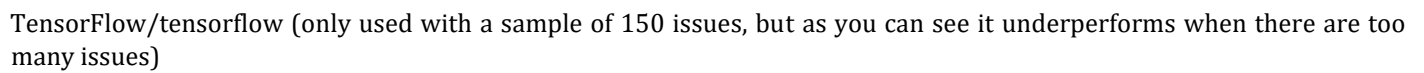
	LABEL	COUNT
	Filter	Filter
1	bug fix	2
2	to-test	2
3	need feedback	2
4	feature	2
5	documentation	1
6	enhancement	1

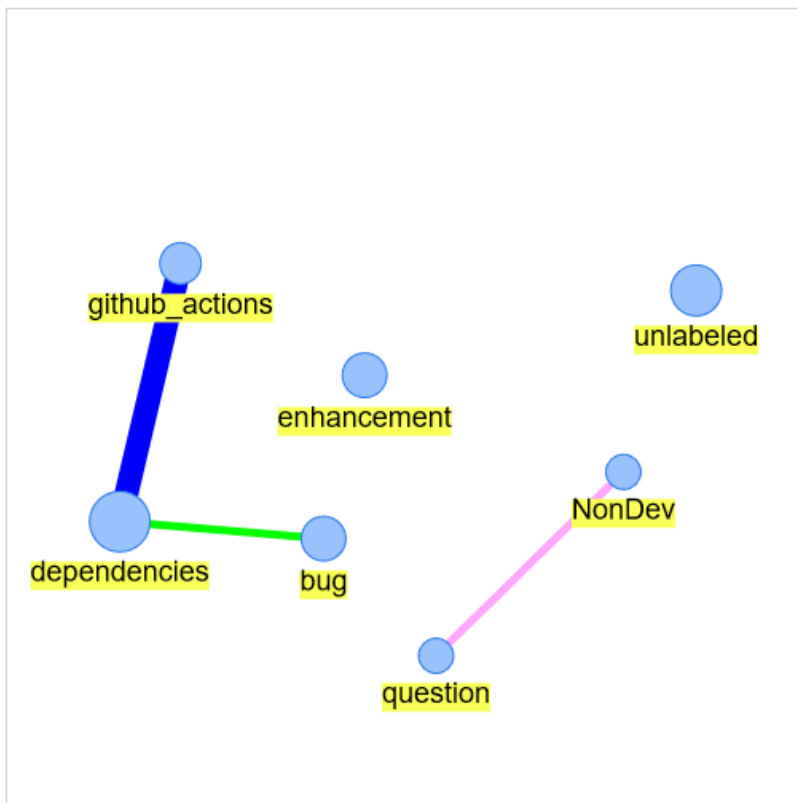
An improvement I would make if I had more time would be to add code to deal with GitHub rate limiting. I could minimise the number of calls my program has to make and write code to make it pause if it's going to go over the rate limit.

## 2.2 displayData.py

This script creates our output. It retrieves the data from the SQLite3 database and creates a "Network" object using `pyvis.network`. We then use `.show` to create a html file of the output and use the `OS` module to open it in the default browser.

hyperledger/firefly





### 3 User Interaction

The only user input is the repo address, which is in the format username/repo.

### 4 Environment

I tested my work on Windows 10 but it should also work on Linux and MacOS. The only place where there could be an issue is the `os.system` command, which automatically opens the browser to show the PyVis output, the command used won't work in Linux and MacOS so the visualisation won't open itself automatically.

### 5 Reflection

I am happy with my finished project. My aim was not to create a perfect visualisation by following what someone else has done before, but to try something new and produce an interesting and original visualisation. If I did the project again, I would consider adding more to the .HTML pages that are output so that they feel less sparse. I would also implement unit testing code, but didn't feel it was very necessary for such a small-scale project.