Analysing commit messages

June 13, 2021

1 Introduction

With the rise of open source software, more and more big corporations incorporate free software in their stack. This also means that the amount of meaningful software that is available online on platforms like GitHub [1], is ever increasing. GitHub, as the name already suggest, offers the ability to host Git repositories. Git is a distributed version control system [2]. In this paper an effort has been made to analyse Git commits and gather information on their intended basic operation by looking at the message of the commit. Therefore repositories of the top 10 most wanted programming language according to [3] have been investigated. To get a more accurate representation of each language, 100000 commits have been chosen. To enhance diversity, only the last 10000 commits of each selected repository were used. The analysis has been done in Python.

2 Methodology

As already alluded to in section 1 the goal of the project is to analyse commit messages. To achieve this repositories of the top 10 most wanted languages. The reason for using this list as reference, is because it has a great assortment of languages that are in high demand and also relevant in today's development climate. The languages in question are:

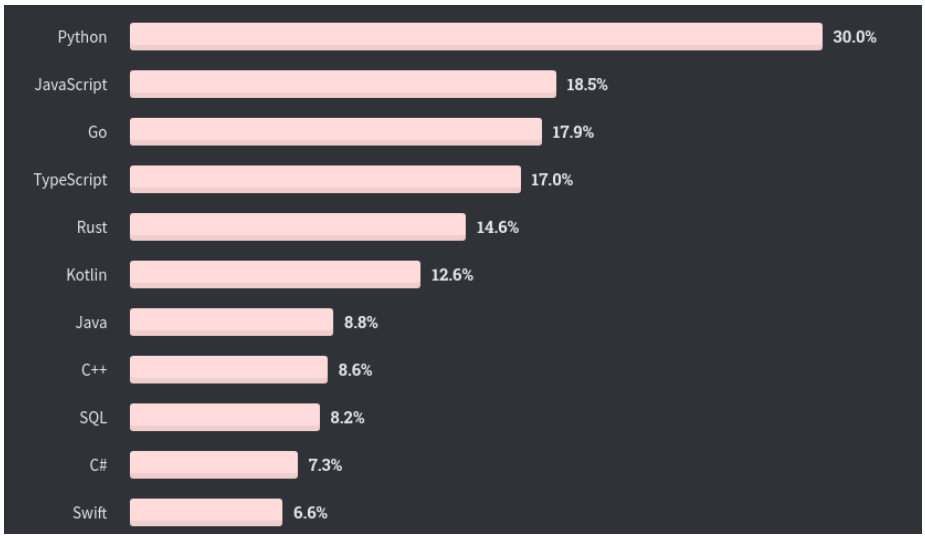

Figure 1: top 10 most wanted languages [3]

It has to be noted that not all laguages depicted in Figure 1 have been chosen since SQL is not suited for analysis and therefore Swift has been chosen instead. Regardless of the percentages seen in Figure 1, 100000 commits of each language have been considered for analysis, where 10000 was the maximum per repository. The language of choice for the programming part of the analysis is Python. Python is well suited for this project as it has rich support for natural language processing related tasks. Additionally the language is great for quick prototyping of ideas and with the help of tools like Jupyter Notebook, document with graphs and text can be created in no time.

3 Challenges

Unfortuantelly the project could not be completed without its fair share of challenges. Starting off with the scraping process which was not as straight forward as it originally might have seemed. Since all repositories are from GitHub it was an obvious choice to use the GitHub API are in this case PyGithub for everything regarding scraping. After some testing however it was clear that this is not a feasable strategy as the GitHub API imposes a strict request limit of 5000 requests per hour and since we have to go over each commit in order to get the commit message, 5000 requests are reached in no time. Ultimately rethinking our strategy was the only option. This did not mean that we had to abonden PyGithub completely as we still used it to query the repositories. Gathering the commits was only really possible by actually cloning the repo and going over the commits with Git itself. Usually this is obviously a reasonable way of fetching commits, however with the amount of repositories we are working with, this

was rather cumbersome, since the disk space requirements were considerable. Furthermore each scrape run was also time intensive considering the fact that each repository had to be coloned and processed.