National College of Ireland

Bachelor of Science HONS in Computing (BSHCEDA)

Data Analytics

2021 / 2022

Robert Soldan

181077800

x18107800@student.ncirl.ie

Bug Prediction In Large Scale Software Development

Technical Report

Contents

Executive Summary

Software has grown more complex, and the cost of fixing software bugs is increasing. The ability to predict the occurrence of bugs after the contribution of a developer, or to predict what types of bugs that contribution might cause can be an asset to any company which develops software at a large scale.

This project explored the possibility of predicting a bug occurring after a contribution by using the data from the title, body, and tags of the pull request (PR). Additionally, the same data was be used to predict the category of a bug.

The data set was scraped from *VSCode's* GitHub repository, and it comprises of pull request information and issues information. Issues were assigned to pull requests according to their dates, issues being after pull requests. The analysis used three machine learning models to perform the predictions: Random Forest, Decision Trees, and Linear Support Vector Machine (lSVM).

The results did not show any significant levels of accuracy in the predictions for the models tested. This may have been due to inconsistent labeling across the project, and under sampling when testing for the bug categories.

1. Introduction

1.1 Background

As software gets increasingly complex, the time spent dealing with bugs increases. A team can spend up to six months just trying to identify the source of a bug and fix it (*Software teams and their knowledge networks in large-scale software development | Elsevier Enhanced Reader* 2022). There are methods in place to track bugs, feature work, component work, bug fixing. All this data is generated to make it easier to track work, who has done what and to help engineers fix issues and bugs when they appear. When dealing with a bug, it is helpful to have breadcrumbs leading you back to the original Pull Request and seeing the code creating the code in its original context.

Data for large scale projects is readily available for open-source projects within the GitHub API. The data used for this analysis was collected from the VSCode GitHub repository (*Visual Studio Code - Open Source ("Code - OSS")* 2021). This project has over 50 releases, 5 000 active issues, 117 000 closed issues, 230 open pull requests and 10 382 closed pull requests, so it is an excellent candidate for this analysis.

Software bug prediction is traditionally performed by looking at source code (Delphine Immaculate, Farida Begam, and Floramary 2019). Machine Learning lends itself perfectly to that, but to prepare the data set which contains the source code there is a lot of human intervention needed. To minimize that effort but still get decent results, one option is to reduce the number of features considered when performing the analysis (Shivaji et al. 2009).

Another option is to not look at the code at all but look at other factors which can be used as predictors for bugs in software. Reducing Features to Improve Bug Prediction (*A Novel Machine Learning Approach for Bug Prediction | Elsevier Enhanced Reader* 2022) takes the actual number of bugs as a dependent variable and then uses linear regression on other metrics to decide on the best metrics to use as independent variables.

The analysis attempted to use the titles and body of PRs as independent variables, and number of bugs or category of bugs as dependent variables. Machine Learning and Natural Language Processing (NLP) aided in this process.

1.2 Aims

This analysis aimed to implement a system able to predict the occurrence of bugs and / or type of bugs appearing because of a pull request (PR). We looked at the contents of the title, body, and labels of the pull request, as well as information related to other details, such as date or if the pull request is a bug fix associated with a certain bug. The main benefactors of this project would be large scale projects, since those would have generated enough data needed for this type of analysis. Smaller projects could benefit as well, if they use a popular software stack (i.e., MERN stack for web development).

This approach can also be used to identify problem areas of a code base, by looking at the type of pull requests (PRs) and the type of issues they are addressing.

1.3 Technology

The main programming language used in the analysis is Python, and various python libraries were added and used. The main library was pandas' (*pandas - Python Data Analysis Library* 2021), which is widely used in data science for its ease in both data analysis and data manipulation.

The main IDE used was Data Spell (*JetBrains DataSpell* 2021), which is geared towards data analysis and its main advantage is that it can handle Jupyter Notebook files locally, with a local Jupyter notebook server running. Python within DataSpell was used for most of the operations in this analysis. The reason for choosing the Jupyter Notebook as the main format for the analysis is its flexibility and modularity when it comes to manipulating and visualizing data. Other python libraries used are:

- *numpy* for mathematical functions

- *seaborn* for correlation graphs and other data visualizations
- *wordcloud* for generating wordcloud visualization
- *json* for manipulationg JSON files
- *requests* for grabbing API data
- *ATOMClassifier* from the atom package for NLP machine learning

Aside from data analysis, acquiring the data set was used using three simple python scripts, which were written, tested and ran using *PyCharm IDE.*

Since everything is going to be developed and tested on a Linux machine, the project also made use of bash for general trouble shooting, setting up the *DataSpell* and *jupyter notebook* environment and running quick tests for the API via curl before extracting the data from the API.

For project management, *Only Office's* project management tool was used to keep track of general tasks and milestones while *Nextcloud Deck* kept track of specific tasks. Git was used for version control for the main jupyter notebook file and for the python data extraction files.

1.4 Structure

Below there is a quick overview of the project structure:

- *Executive Summary* of the report, highlighting major points and describing the results
- *Introduction* presents the background and motivation of the project, what it aims to solve, a brief overview of the technologies used and the present summary
- *Data* is a detailed description of the data set, where it was sourced from and an outline of exploratory analysis
- *Methodology* is a description of how to get from the data to the analysis itself. The methods are explained in detail, including preprocessing.
- *Analysis* detail the approach taken in processing the data and justify it

- *Results* present the results of the analysis using figures, graphs, and tables to outline the conclusions
- *Conclusions* explain the advantages and disadvantages of the approach taken in the analysis. Other approaches were discussed, learning from the chosen approach
- *Further Development or Research* explores what path the project would take with additional time and resources

2. Data

2.1 Overview

The data is collected using the GitHub API from VSCode's GitHub page, via three python scripts. There are three main data sets, collected from three endpoints:

https://api.GitHub.com/repos/microsoft/vscode/issues for VSCode's GitHub issues. This is where users can report any issue. These reports often come from people who are not developers or contributors to the project, from product users. This is the largest data set out of the three, the resulting file being almost 1GB in size.

Data set properties

issues.json	
Initial Format	JSON
Size	997.7 MB
After conversion to pandas dataframe	
Columns	29
Rows	135172

https://api.GitHub.com/repos/microsoft/vscode/pulls for VSCode's pull requests. These are the pull requests submitted by developers. They can be regular developers hired by Microsoft, regular developers who are not hired but volunteers, or just infrequent or one-time contributors. PRs can be either open or closed. The closed one can be merged or simply closed.

Data set properties

pulls.json	
Initial Format	JSON
Size	259.7 MB
After conversion to pandas dataframe	
Columns	36
Rows	10896

https://api.GitHub.com/repos/microsoft/vscode/releases for the releases. This is the smallest of the sets, with only 53 entries. The most important field for this set is the release date *published_at*, since this helped to track down issues and PRs. Due to its small size, this data set was not of much use to this analysis.

Data set properties

releases.json	
Initial Format	JSON
Size	172.0 kB
After conversion to pandas dataframe	
Columns	19
Rows	54

Together, the three sets had a total of 1.3GB, or 146 031 lines of data (records), when converted in a table format.

The data is freely available by using the GitHub API (*Endpoints available for GitHub Apps* 2021). It is recommended to register with GitHub and acquire an API key. There is no cost to it and accessing the API without a key is limiting. The data sets used in this project are available through the link provided in the code submission for this project.

These three data sets are suitable because they offer a large quantity of data relevant to the analysis. The focus was on the *title* and *body* for the issues and PRs. Then each issue can have its date linked to a certain PR.

2.2 Descriptive Statistics

Descriptive statistics are applied to the data sets to get a better understanding of the layout of the data sets and to identify if key areas are lacking.

For the issues data set, just under 20% of entries which have no labels are assigned to them. And a small number of entries with no body:

```
df_no_labels = df_issues[df_issues.labels.str.len() == 0]
len(df_no_labels)
```

```
24762
```

```
df_no_body = df_issues[df_issues['body'] == '']
len(df_no_body)
```

```
2387
```

The issues data set has 24 762 entries with no labels, and 2387 entries with no body. 18% of the entries have no labels, and 1.7% have no body text. This highlights the importance of not using only labels but converting the title and body text into labels - like tags. Entries with no body text can be discarded, since they represent a small percentage of the total.

For the pull requests data set, the situation is even more accentuated:

```
df_prs_no_labels = df_prs[df_prs.labels.str.len() == 0]
len(df_prs_no_labels)
```

```
8773
```

```
df_prs_no_body = df_prs[df_prs['body'] == '']
len(df_prs_no_body)
```

```
1144
```

The pull requests data set has 8773 entries with no labels and 1133 entries with no body. 81% of the entries have no labels at all, and 10% have no body text. For this data set, all rows were kept, and keywords were extracted from *title*, even if there was no *body* text or label.

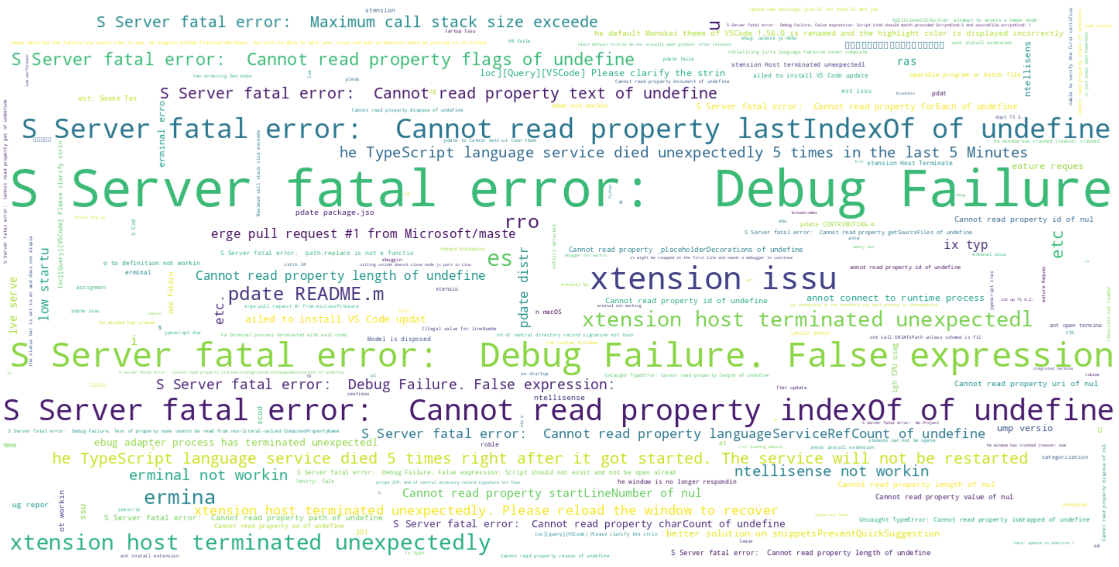
2.3 Exploratory Analysis

Exploratory data analysis (EDA) gave a quick look at the data before doing additional transformations. Using visual tools outlined issues with the data set and help to identify patterns.

As a quick way to visualize the data sets, a *wordcloud* script was run against the titles in the data sets. Wordcloud is a type of visualization where the most frequently used words or phrases in a data set are displayed larger. So, the more common the phrase is, the larger it appeared in the image.

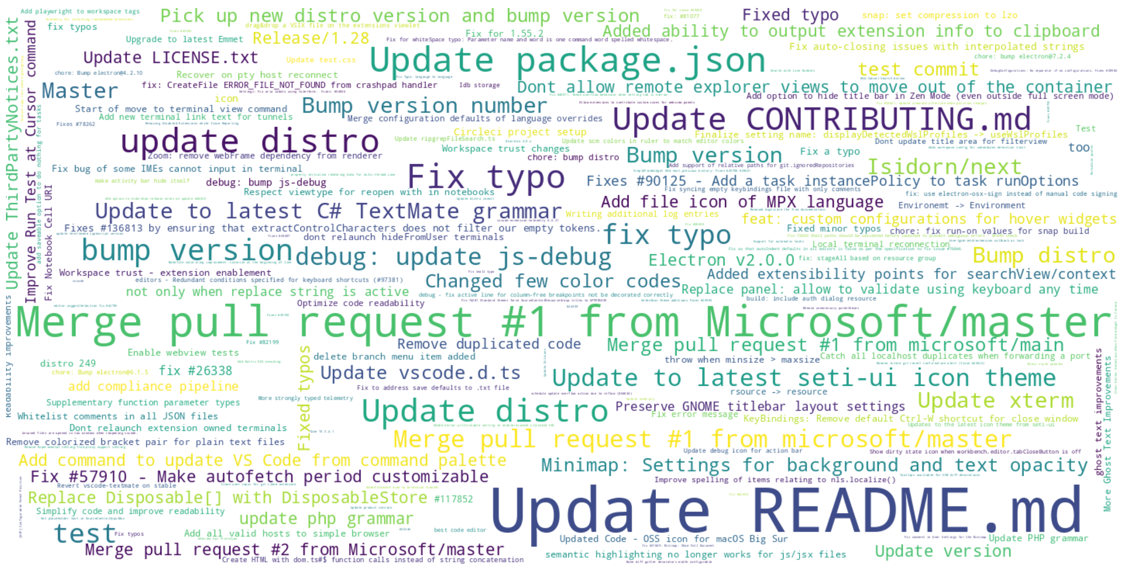
2.3.1 Issues Dataset

For the *issues* data set, it is obvious that *Server fatal Error* is a widespread problem encountered by users, but a few unrendered characters also are visible in the representation, from a non-Latin alphabet. The data set should be cleaned of non-Latin characters.



2.3.2 Pull Requests Dataset

For the pull requests data set, there are a few words which must be eliminated since most of the titles seem to contain information not necessarily relevant to the PR itself.



3. Methodology

The methodology used for this project is KDD (Knowledge, Discovery & Data Mining). KDD is an iterative process, which benefited multiple data sets analysis. Similar methodology was employed in Deep learning in static, metric-based bug prediction (Ferenc et al. 2020) which explored bug prediction via deep learning algorithms.

KDD has five stages:

Selection -> Pre-processing -> Transformation -> Data mining -> Evaluation

3.1 Selection

The selection process for this analysis consisted of two main steps.

3.1.1 Choosing the data sets

First, a decision was made in relation to the data to be collected. GitHub's API offers a vast selection of data to pick from. One of the more important decisions regarding this was whether to include user data in the analysis. The data collected is mostly user generated, so collecting user's data does make sense. But this analysis wants to point out issues in the overall approach when managing a project, and not study or analyze individual contributions to the project. On top of that, contributions to open-source projects are not evenly distributed between the users. On large projects, there can be at least two main types of contributors: Paid contributors who consistently push new code to the project and volunteer contributors who may or may not be consistent or may even be one-time contributors. This could easily be a source of bias when analyzing data regarding users since the users who contribute more can also be involved in producing more bugs.

So, the decision was made to not include any user related at all. The three chosen data sets were *issues, pull requests* and *releases* since the data found in these sets are going to be the most relevant to the analysis.

3.1.2 Choosing the project

The second decision was to select the project which was the subject of this analysis. Again, this could have been handled in two separate ways:

- Select one large project with lots of reported issues, PRs, and releases
- Select multiple smaller - medium projects using the same stack or similar technologies

Both approaches have cons and pros, but the inconsistency of choosing the second option (different labels, different approaches in issue reporting, different contribution rules) has pushed the decision towards the first option, choosing one large project.

As for which large project to choose for the analysis, there was initially a short-list of open-source projects to choose from: VSCode, Linux, freeCodeCamp.

Linux, although present on GitHub, the GitHub repo is currently just an updated clone of their GitLab repository. And they were moved on GitLab recently, so the dates might not be reliable or exact.

freeCodeCamp is considered as the largest open-source project on GitHub, but the project is formed out of multiple projects, so there is no one freeCodeCamp project where people contribute to, making it harder to perform a concise analysis.
VSCode is seen as the second largest open-source project on GitHub and its main application, the VSCode IDE is a large monolith project. It also has the advantage of consistent, paid developers so they also have a strict curation when it comes to reporting and creating pull requests. This has made VSCode the final decision for the project to be the subject of this analysis.

3.2 Pre-processing

This stage consisted of importing the data into the jupyter notebook and cleaning it. Since this is human generated data, natural language and markdown, a fair amount of cleaning was required. All three data sets needed similar cleaning for their *title* and *body* fields.

First, the data was imported into python data frame format by pandas' *read_json('file')* command, into three pandas data frames.

Then, the cleaning process was applied to the data frames.

- *title* have more rigid rules, so it was easier to clean overall. Main things which were removed are emojis and non-English alphabet characters.
- *body* text allows markdown and some html tags. That makes it so there was a lot of cleaning to be performed.

````python
df_issues = df_issues.replace(r'\r+|\n+|\t+','', regex=True) # newlines, tabs
df_issues = df_issues.replace(r'<[^>]*>', '', regex=True) # html tags
df_issues = df_issues.replace(r"```.*?```", '', regex=True) # markdown code tags
````

- All non-English alphabet characters also were eliminated

4. Transformation

This stage consists of destructive or shape altering modifications of the data sets. After the main data cleaning process, the next steps were also part of the transformation. *Body* and *title* data were stripped of common English words to isolate key words in issue reporting and pull requests.

Because the data sets were scraped and they are individual, the transformation process was complex and extensive. From the data sets we had to do a series of transformations which brought us to a data set with a predictor (dependent variable), and with independent variables.

The analysis is tested the possibility of predicting issues or bugs based off PRs and their content. It made sense to take the independent variables from the *PRs* data set, and the dependent variable from the *issues* data set.

For the independent variables *body, title* and *user* were taken into consideration. For the issues variable, labels assigned to issues were used. This can be done since most of the issues do make use of labels.

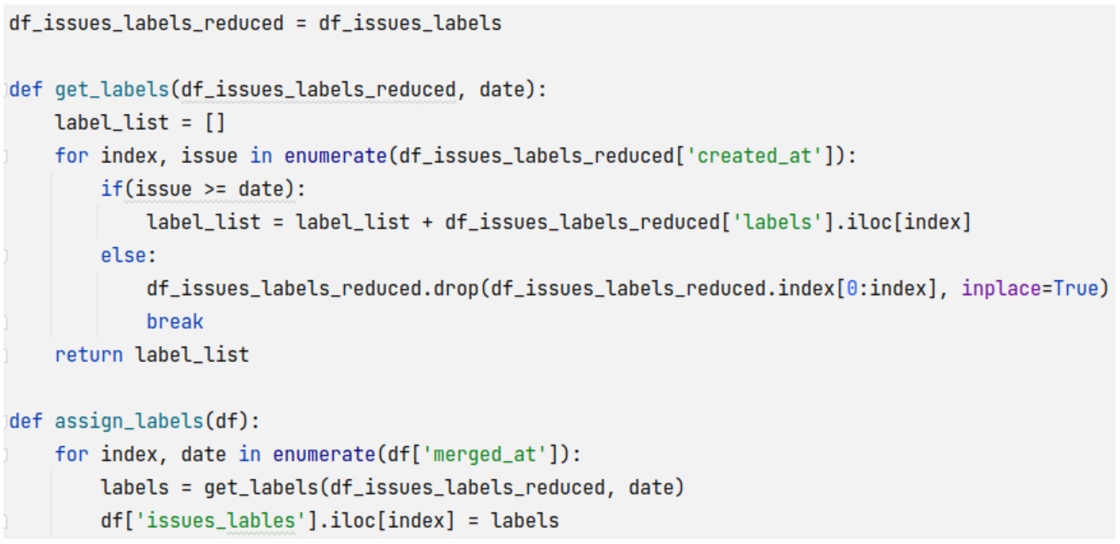
Next step was to decide how to match the two data sets. There were about 10 000 entries in the *PRs* data set and over 100 000 entries in the *issues* data set. The goal was to match them by date. The labels from issues which had a date following a PR, but before the next PR had their labels associated with that PR. The dates to be considered were the *merge* date for PRs and *created* date for issues. We were interested only in merged PRs, so the unmerged PRs where dropped from the data set, which reduced the data set to about 7800 entries. For example, below is an example of a PR and its associated issues.

PR Title	Date
Terminal now supports linking local file paths containing '$'	2021-10-22 14:53:00+00:00
allowed characters: string -> record	2021-10-24 08:29:00+00:00

Issue Title	Date
TypeScript: Error: Promise did not implement oncancel()	2021-10-23 19:24:00+00:00
[vb] Block comment is not colored	2021-10-24 10:02:00+00:00

In this case, the PR was associated with the first issue, second PR with the second issue, because the date of the first issue is before the second PR and after the first PR.

The labels were stored in a new PR data frame, in the *issues_lables* column. To achieve this while collecting info from over 90 000 rows of issues and storing them in over 7 000 rows of PRs, looping through all the issues for each PR row was avoided, because this would end up at 630 000 000 complete loops. This was shortened to a maximum of the original 90 000 complete loops by sorting both the *PRs* data set and the *Issues* data set by date. That means that iteration was done in order,and could quit the search for issues associated with a PR after passing the date range and move on to the next PR, and then continue going through the issues from where it were previously. The code can be seen below:

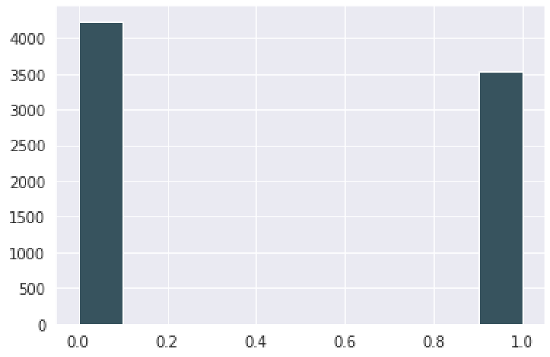

```
df_issues_labels_reduced = df_issues_labels

def get_labels(df_issues_labels_reduced, date):
    label_list = []
    for index, issue in enumerate(df_issues_labels_reduced['created_at']):
        if(issue >= date):
            label_list = label_list + df_issues_labels_reduced['labels'].iloc[index]
        else:
            df_issues_labels_reduced.drop(df_issues_labels_reduced.index[0:index], inplace=True)
            break
    return label_list

def assign_labels(df):
    for index, date in enumerate(df['merged_at']):
        labels = get_labels(df_issues_labels_reduced, date)
        df['issues_lables'].iloc[index] = labels
```

This assigned each PR with a list of labels. The next step is to categorize those issues. The analysis tried two approaches.

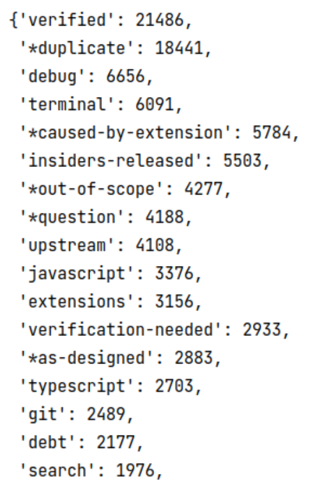
The first approach is to make use of the verified label. This label checks if the issue has been verified (*Labels ů microsoft/vscode* 2022). Another column was be added, which contained the number of verified occurrences after a PR is merged. These were then transformed to categorical variables. Results showed that PRs had the verified tag mentioned up to 154 times per PR:

```
10570    154
3962     153
10627    120
10337    112
7787     106
        ...
7539       0
4278       0
4248       0
4246       0
5224       0
Name: verified, Length: 7819, dtype: int64
```

But because over half of the entries had 0 *verified* labels, the 2 chosen groups were 0 and above 0. This gives us an almost even distribution:



The second approach was to isolate the topmost relevant labels to be used as categories. Relevancy in this case is indicated by what the label indicates. The label had to indicate that the issue was pointing at a section of the project (editor, extensions). The categories were arranged in a python dictionary, with the label *name* as key and the total count in the issues data set as value and sorted the dictionary by values:



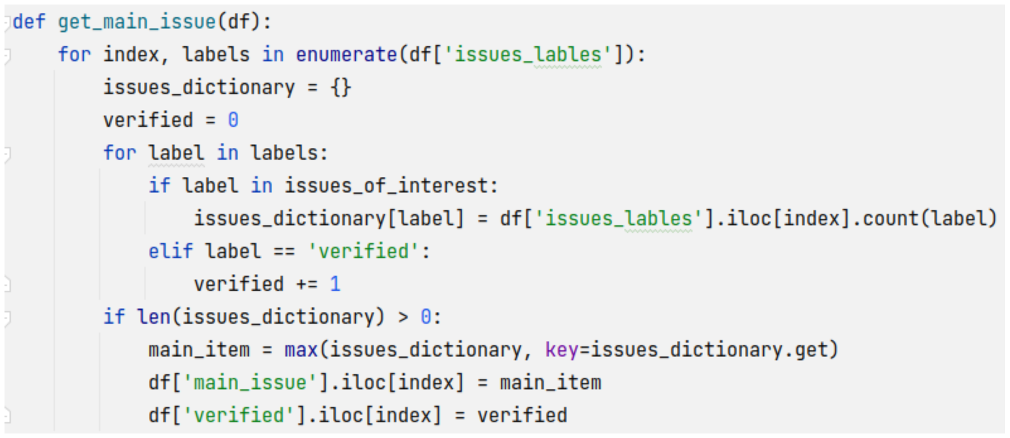
```
{'verified': 21486,
 '*duplicate': 18441,
 'debug': 6656,
 'terminal': 6091,
 '*caused-by-extension': 5784,
 'insiders-released': 5503,
 '*out-of-scope': 4277,
 '*question': 4188,
 'upstream': 4108,
 'javascript': 3376,
 'extensions': 3156,
 'verification-needed': 2933,
 '*as-designed': 2883,
 'typescript': 2703,
 'git': 2489,
 'debt': 2177,
 'search': 1976,
```

It stood out that not all labels were relevant. Some labels, such as *verified, duplicate, debug, as-designed* pointed out to the nature of the issue itself, and not to what area of the project is referenced. Based on this list, the following labels were be used as categories:

terminal, extensions, typescript, search, api, file-explorer

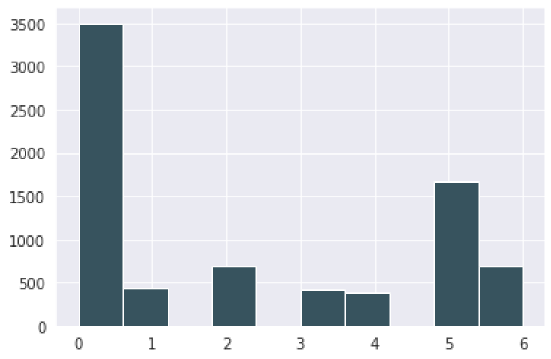
Since each PR had multiple issues assigned to it, there were repeating labels. Each PR had one main category assigned to it, depending on which is the most common label found in it.

This was achieved by a method which looped through the set and assign the most found label as the *main_issue*. This method also assigned the *verified* attribute which was used in the first part of the analysis.



```
def get_main_issue(df):
    for index, labels in enumerate(df['issues_lables']):
        issues_dictionary = {}
        verified = 0
        for label in labels:
            if label in issues_of_interest:
                issues_dictionary[label] = df['issues_lables'].iloc[index].count(label)
            elif label == 'verified':
                verified += 1
        if len(issues_dictionary) > 0:
            main_item = max(issues_dictionary, key=issues_dictionary.get)
            df['main_issue'].iloc[index] = main_item
            df['verified'].iloc[index] = verified
```

In that state, the set was unbalanced:



To address this, the data set was under sampled. Over sampling was an option but because of the size of the data set it was impractical with the current hardware. On top of that, over sampling with natural language data is not practical.

```
                 3526
terminal         1687
typescript        686
extensions        684
api               435
file-explorer     416
search            385
Name: main_issue_category, dtype: int64
```

The first value was *no label*, so those entries were eliminated from the set. Under sampling was done to match the lowest occurring label, *search*, and all other four occurring labels were randomly reduced.

5. Analysis

The analysis process followed KDD methodology. This methodology is preferred due to its iterative nature. For this analysis, because this was mostly human generated data, a trial an error approach was used when it comes to cleaning and transforming the data to see what gave the best results. KDD also accentuates the pre-processing and transformation stages of the analysis, which was relied on heavily.

Overall, KDD is a process best suited for learning and identifying the best way to do a certain analysis (*KDD and Data Mining* 2021). This project is the first attempt at this approach and KDD helped with identifying pitfalls in the analysis approach or large issues / inaccuracies with the data sets. KDD is not without its faults, being expensive and time-consuming, so not exactly appropriate for input coming from a data warehouse. But this analysis identified the best way of processing and transforming the data. Subsequent implementations can follow a different, more efficient methodology once all the steps and their pitfalls are clear. KDD is more well suited for research-oriented work, such as this analysis, while other methodologies, such as SEMMA or CRISP are best suited for business needs (Shafique and H. Qaiser 2014).

When choosing which attributes to use as an independent variable, the first thought was to make use of the label feature present in GitHub and many other versioning systems. Although this would be ideal, in practice the labels were not used at all or just sometimes (as seen above in descriptive data analysis) or used for other things (Do not merge, Do not review). The decision was made to incorporate key words from the title and body of the issue and / or pull request to identify the areas addressed with the issue or pull request.

The analysis made use of the atom package in python, which streamlined the preprocessing needed to perform NLP related machine learning. Before applying any ML techniques, the data was cleaned and tokenized using *atom's clean()* and *tokenize()* functions.

The ML techniques were tested against two dependent variables with the same set of independent variables.

First, the dependent variable had a value of either 0 or 1. 0 if the PR resulted in issues, 1 if the PR did not result in issues.

Second, the dependent variable was categorical, a list of 6 labels from the issues attributes:

terminal, extensions, typescript, search, api, file-explorer

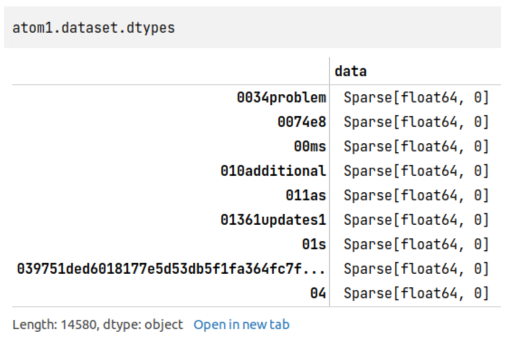
After tokenization, the values were vectorized using the term frequency - inverse document frequency (tf-idf) strategy. Since there was a large amount of data, vectorizing by this strategy made sure that very frequent terms were not in the way of taking rarer, but more relevant terms into consideration (S. Qaiser and Ali 2018).

For each of these multiple ML techniques were applied, recording the results, after the tokenization and vectorization of the independent variables and transformations to dependent variable. To help with the decision of which techniques to use, *atom's available_models()* was used. This method provided a list of models which can be used on the current data set. Since the data sets were balanced, accuracy was used as a metric for the tested models. One other option was to look at weighted average for the f1 scores, but that would be more appropriate for unbalanced sets.

6. Results

With the data set transformed and ready for the final analysis, we now have two data sets, each with different dependent variables, as mentioned above: First data set has a binary dependent variable, looking at the PRs which may or may not have caused bugs. The second data set has a dependent variable which can be either one of the 6 labels mentioned in the Analysis section.

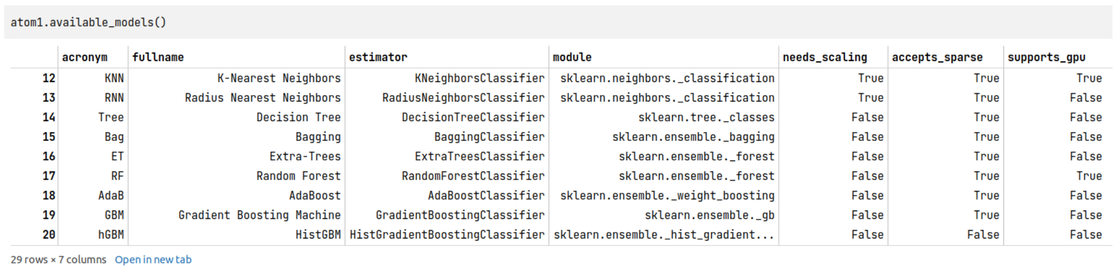
Since the data set is quite large, during the vectorization process we obtained a data frame of sparse arrays.



```
atom1.dataset.dtypes
```

```
                                          data
                        0034problem  Sparse[float64, 0]
                             0074e8  Sparse[float64, 0]
                               00ms  Sparse[float64, 0]
                      010additional  Sparse[float64, 0]
                              011as  Sparse[float64, 0]
                      01361updates1  Sparse[float64, 0]
                                01s  Sparse[float64, 0]
039751ded6018177e5d53db5f1fa364fc7f...  Sparse[float64, 0]
                                 04  Sparse[float64, 0]
Length: 14580, dtype: object   Open in new tab
```

That means that when selecting the model from *atom's avalable_models()* method, we must make sure that the model accepts sparse.



```
atom1.available_models()
```

	acronym	fullname	estimator	module	needs_scaling	accepts_sparse	supports_gpu
12	KNN	K-Nearest Neighbors	KNeighborsClassifier	sklearn.neighbors._classification	True	True	True
13	RNN	Radius Nearest Neighbors	RadiusNeighborsClassifier	sklearn.neighbors._classification	True	True	False
14	Tree	Decision Tree	DecisionTreeClassifier	sklearn.tree._classes	False	True	False
15	Bag	Bagging	BaggingClassifier	sklearn.ensemble._bagging	False	True	False
16	ET	Extra-Trees	ExtraTreesClassifier	sklearn.ensemble._forest	False	True	False
17	RF	Random Forest	RandomForestClassifier	sklearn.ensemble._forest	False	True	True
18	AdaB	AdaBoost	AdaBoostClassifier	sklearn.ensemble._weight_boosting	False	True	False
19	GBM	Gradient Boosting Machine	GradientBoostingClassifier	sklearn.ensemble._gb	False	True	False
20	hGBM	HistGBM	HistGradientBoostingClassifier	sklearn.ensemble._hist_gradient...	False	False	False

29 rows × 7 columns Open in new tab

Both sets have three models applied to them:

- Random Forest
- Decision Trees
- lSVM (linear Support Vector Machine)

Python's *atom* again facilitates running these three models and interpreting the results.

```
atom.run(models="RF", metric="accuracy")
```

```
atom.run(models="Tree", metric="accuracy")
```

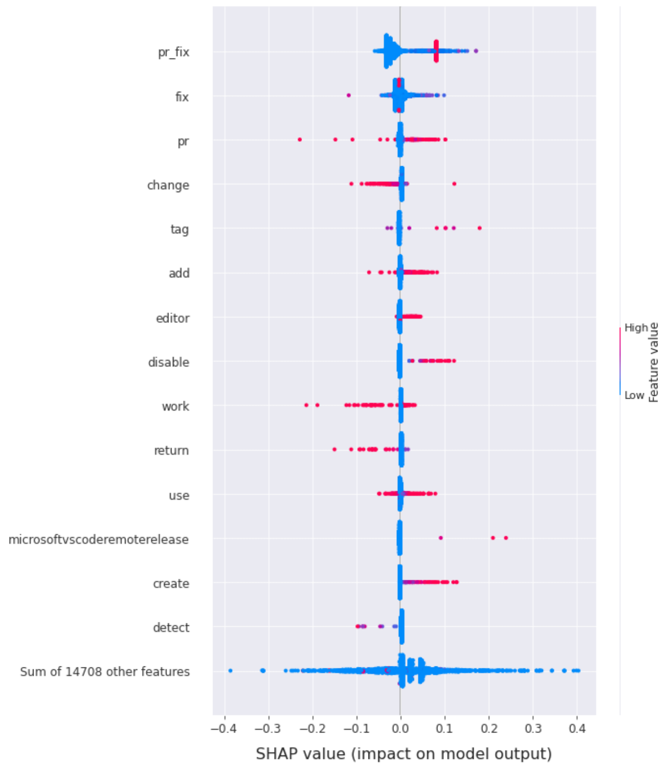
```
atom.run(models="lSVM", metric="accuracy")
```

6.1 First Set

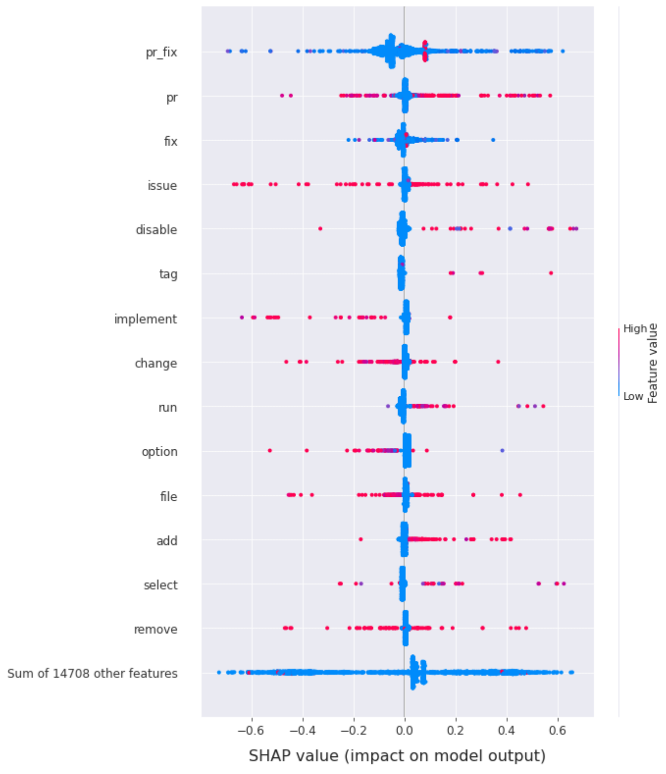
In the transformation section of this analysis, it was mentioned that the first set is balanced. The dependent variable has a value of 0 if the PR does not have any subsequent issues with the *"verified"* label. And a value of 1 if it does have issues with the *"verified"* label.

The models are built by using the methods described above in the Analysis section. Bee swarm plots highlight the impact on model output certain terms have, when trying to predict that there are no bugs associated with the PRs:

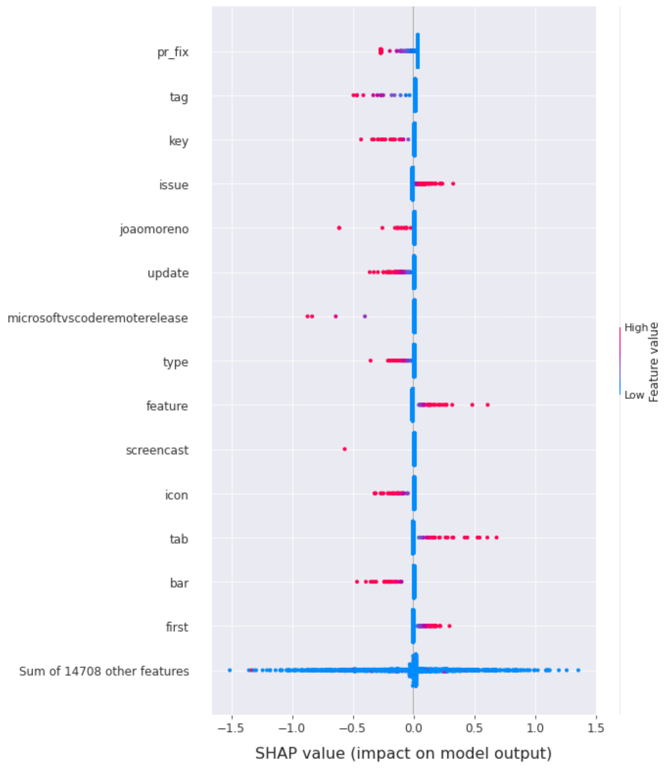
Random Forest



Decision Trees

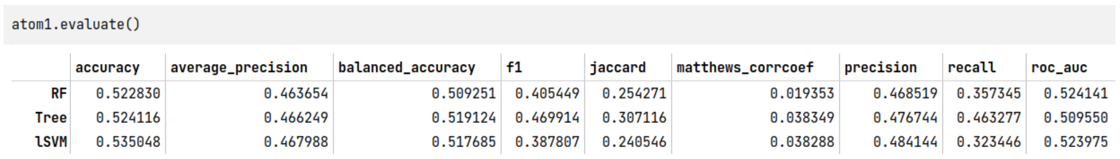


lSVM



The plots show that *pr_fix* is the main word for the three models, with slight differences for the next few other terms.

The test evaluation results from the three models can be seen in the table below:



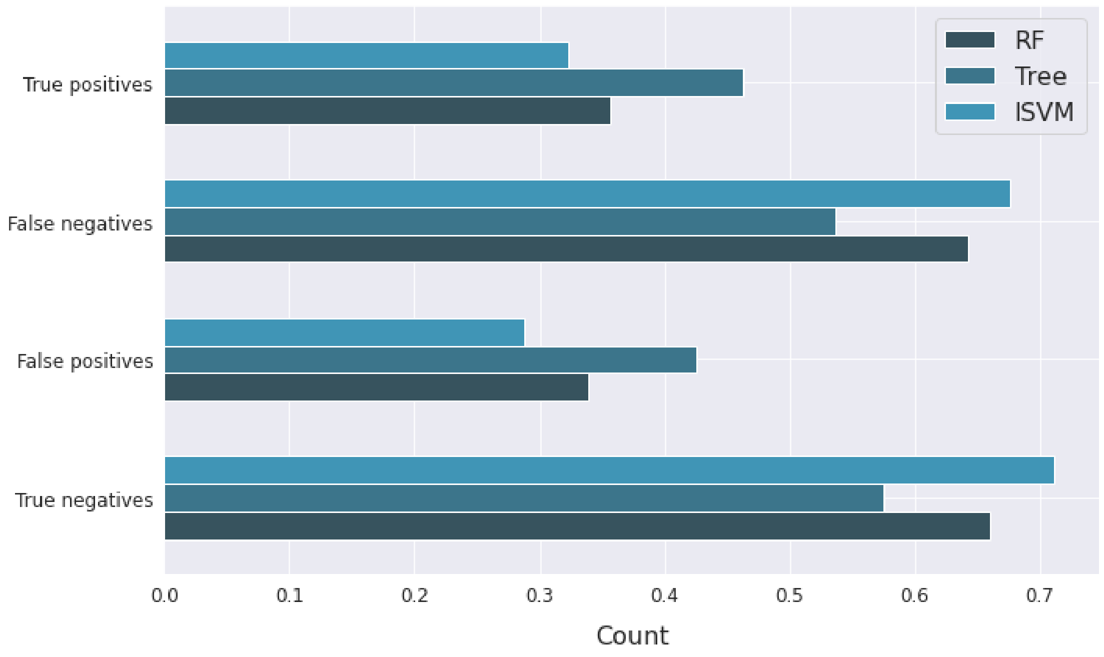
```
atom1.evaluate()
```

	accuracy	average_precision	balanced_accuracy	f1	jaccard	matthews_corrcoef	precision	recall	roc_auc
RF	0.522830	0.463654	0.509251	0.405449	0.254271	0.019353	0.468519	0.357345	0.524141
Tree	0.524116	0.466249	0.519124	0.469914	0.307116	0.038349	0.476744	0.463277	0.509550
lSVM	0.535048	0.467988	0.517685	0.387807	0.240546	0.038288	0.484144	0.323446	0.523975

It is immediately obvious that the results are not conclusive.

Accuracy and precision are around 50 percent, which is where it would be for random guesses. Area Under Curve (*roc_auc*) is close to 50 percent as well, which shows that no clear correlation can be drawn between the independent and dependent variables.

Next, a normalized confusion matrix plot comparing the three models confirms the fact that the results are inconclusive.



All three models have similar performance. Decision Tree has the highest true positive rate at about 0.46, while lSVM holds the highest true negative rate with 0.71. Random Forest and lSVM have the highest false negative rates with 0.64 and 0.68 respectively, while Decision Tree has the highest false positive rate with 0.43.

Based on these scores, we can conclude that neither of the models is able to accurately predict the possibility of a verified bug occurring after a certain PR. It is also interesting to note that this graph points out the fact that the Decision Tree method tends towards declaring that a PR will produce bugs, shown here by the high true positives and high false positives, while the other two methods tend towards declaring that the PR will not produce a bug.

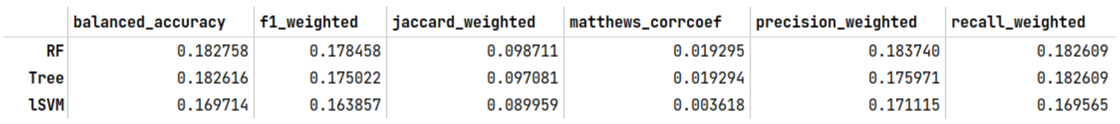
6.2 Second Set

The second set was originally unbalanced, so it had to be balanced. As pointed out in the Transformation section of this analysis, the set was under sampled by taking out random entries which had a main tag other than the search label. The dependent variable is categorical, one of six labels:

terminal, extensions, typescript, search, api, file-explorer

This label represents the main issue found in subsequent issues to a PR. Main issue was considered the label with most occurrences in the total number of issues.

The test evaluation results for the three models:

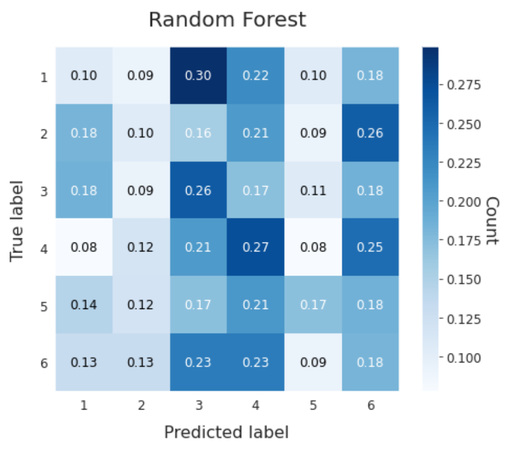


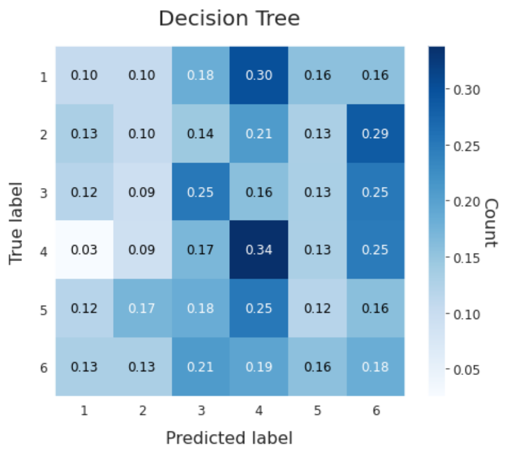
	balanced_accuracy	f1_weighted	jaccard_weighted	matthews_corrcoef	precision_weighted	recall_weighted
RF	0.182758	0.178458	0.098711	0.019295	0.183740	0.182609
Tree	0.182616	0.175022	0.097081	0.019294	0.175971	0.182609
lSVM	0.169714	0.163857	0.089959	0.003618	0.171115	0.169565

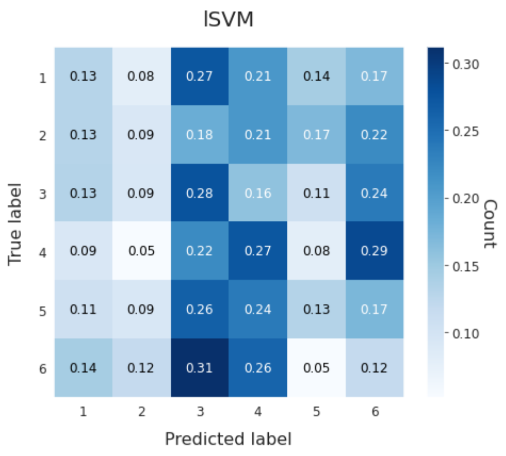
The results are inferior to the first set. This is caused by the fact that the set was under sampled, while trying to predict six categories, as opposed to the first set which was not under sampled and dealt with only two categories.

Accuracy and weighted f1 score are below 20%, which proves the inability of the models to make any accurate predictions using this data.

The confusion matrices for the models do confirm the inadequacy of the constructed models:







These results could be improved by oversampling, but that would not be enough. Reducing the categories down to a maximum of four will be necessary to have any useful result with the current data.

From the two sets, the first set clearly performed better under the three ML models, but it still does not have enough accuracy to predict occurrence of bugs based off the contents of the PR body and title.

7. Conclusion

This analysis tried a different approach to bug prediction. Tags, title and body text of the PRs and issues were used to try to predict the results, while other researchers used the code, alongside past bug occurrences. Not using the code in the analysis has the purpose of simplifying and universalizing the process, alongside a reduction in computational complexity.

From the results we can see that our model was not capable to accurately predict either the existence of a bug after a certain PR or the main category of bugs overall after a certain PR.

The three models tested, Random Forest, Decision Tree and lSVM had comparable results and the algorithms bias could also be seen in the results, with Decision Tree tending more towards deciding that a PR will produce a bug and the other two towards deciding that a PR will not produce a bug.

If we put the two sets against each other, we can see that the set in which we just account for a PR either causing a bug or not performs better than the set where we try to predict the bug category. This is caused by two factors. The under sampling of the bug category set has reduced the amount of data available. Secondly, working with a reduced data set and trying to predict 6 categories, as opposed to two, will yield worse results.

8. Future Development and Research

Although this analysis did not show promising results, these results can be improved with a few adjustments.

With additional time and resources, the second data set, used to predict the category of the main bug can be over sampled instead of under sampled. Fir the first data set, better labeling would produce better results in this case.

On the development side of things, companies should engage in strict tagging practices for PRs and issues. This will improve the overall accuracy of these and similar models.

9. Bibliography

A Novel Machine Learning Approach for Bug Prediction | Elsevier Enhanced Reader (2022). DOI: 10.1016/j.procs.2016.07.271. URL: https://reader.elsevier.com/reader/sd/pii/S1877050916315174?token=3E174267937D42EB9BC369A0BCB78091A5D7E89565603602C45F2205878C8F0CE0930D63EFB4E5C33FD6451FFCB2804D&originRegion=eu-west-1&originCreation=20220507132744 (visited on 05/07/2022).

Delphine Immaculate, S., M. Farida Begam, and M. Floramary (Mar. 2019). "Software Bug Prediction Using Supervised Machine Learning Algorithms". In: *2019 International Conference on Data Science and Communication (IconDSC)*. 2019 International Conference on Data Science and Communication (IconDSC), pp. 1–7. DOI: 10.1109/IconDSC.2019.8816965.

Endpoints available for GitHub Apps (2021). GitHub Docs. URL: https://docs.github.com/en/rest/overview/endpoints-available-for-github-apps (visited on 12/19/2021).

Ferenc, Rudolf et al. (July 1, 2020). "Deep learning in static, metric-based bug prediction". In: *Array* 6, p. 100021. ISSN: 2590-0056. DOI: 10.1016/j.array.2020.100021. URL: https://www.sciencedirect.com/science/article/pii/S2590005620300060 (visited on 12/19/2021).

JetBrains DataSpell (2021). *JetBrains DataSpell: The IDE for Data Scientists*. JetBrains. URL: https://www.jetbrains.com/dataspell/ (visited on 12/19/2021).

KDD and Data Mining (2021). Data Science Process Alliance. URL: https://www.datascience-pm.com/kdd-and-data-mining/ (visited on 12/22/2021).

Labels ů microsoft/vscode (2022). GitHub. URL: https://github.com/microsoft/vscode (visited on 05/02/2022).

pandas - Python Data Analysis Library (2021). URL: https://pandas.pydata.org/ (visited on 12/19/2021).

Qaiser, Shahzad and Ramsha Ali (July 16, 2018). "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents". In: *International Journal of Computer Applications* 181.1, pp. 25–29. ISSN: 09758887. DOI: 10.5120/ijca2018917395. URL: http://www.ijcaonline.org/archives/volume181/number1/qaiser-2018-ijca-917395.pdf (visited on 05/08/2022).

Shafique, Umair and Haseeb Qaiser (2014). “A Comparative Study of Data Mining Process Models (KDD, CRISP-DM and SEMMA)”. In: 12.1, p. 6.

Shivaji, Shivkumar et al. (Nov. 2009). “Reducing Features to Improve Bug Prediction”. In: *2009 IEEE/ACM International Conference on Automated Software Engineering*. 2009 IEEE/ACM International Conference on Automated Software Engineering. ISSN: 1938-4300, pp. 600–604. DOI: 10.1109/ASE.2009.76.

Software teams and their knowledge networks in large-scale software development | Elsevier Enhanced Reader (2022). DOI: 10.1016/j.infsof.2017.01.003. URL: https://reader.elsevier.com/reader/sd/pii/S0950584917300435?token=94412BFF5DD7AD687BDA070206D628C2F1C3467E1503A67183194322AD6557BFCEE1A64FFE336F011577AB7EB7A13124&originRegion=eu-west-1&originCreation=20220507123447 (visited on 05/07/2022).

Visual Studio Code - Open Source ("Code - OSS") (Dec. 19, 2021). original-date: 2015-09-03T20:23:38Z. URL: https://github.com/microsoft/vscode (visited on 12/19/2021).

10. Appendices

10.1 Project Proposal

10.1.1 Objectives

The main and final goal of this project is to implement a system which is able to predict with reasonable accuracy and confidence future bugs within a software project. Due to the amount of data needed for predictions this can be successfully applied mainly to large scale projects, which have high numbers of commits, reported issues, pull requests and also released versions. This could be applied to smaller projects as well, as long as they use a popular software stack in their development (i.e. MERN stack for web development). In order to reach the main goal, there are several objectives needed for success.

First objective is to collect the data. Luckily, data for large scale projects is readily available for open source projects within the Github API. Using that, I plan to collect data related to issues, pull requests and releases from a large project, VSCode (*Visual Studio Code - Open Source ("Code - OSS")* 2021). With over 50 releases, 5 000 active issues, 117 000 closed issues, 230 open pull requests and 10 382 closed pull requests I believe that this will be a good starting point for the project.

The next step is to prepare the collected data. The API endpoints I will be using contain granular data and metadata, so I will have to carefully select the data I need. Data from different endpoints will have to be linked so that issues correspond to certain pull requests and pull requests are associated with their relevant releases.

After that, the data must be cleaned. I will be focusing on collecting tags, and then collecting key words from titles and body text for the issues and pull requests.

Last, I will have to analyse the data to try to predict bugs and types of bugs which may appear and include the process and the findings in a detailed report.

10.1.2 Background

I currently work in software development. I have noticed that we do take the effort to tag our pull requests, issues and contributions so that we can track and find our previous work more easily. But on top of that, the data generated by being thorough with tagging and naming of issues during development could be used more efficiently to attempt to predict bugs and issues which might appear when adding new features or making certain changes.

During software development, sometimes you will have issues appear without being sure what exactly caused the issue, or developers might spend precious time tracking down the source of the issue in order to fix it. With a bug prediction system in place, these issues could be either prevented entirely, or simply expected. So when the issue does appear you will know exactly what causes it.

10.1.3 State of the Art

In October 2020, there was a hackathon organized by Machine Hack which covers pretty much this subject (*GitHub Bugs Prediction Challenge* 2021). My approach is different by improving on the focus of data set and it's specificity. I plan on targeting a single large open source project in order to get data which is cohesive, compared to using an extremely large data set which contains data from different types of projects, programming languages or even from projects which are not software development related (repos comprising only out of config files). The hackathon was also only collection info from titles. I plan on collecting body text data as well, and process that.

10.1.4 Data

As mentioned before, the data needed for the project will all be collected from the Github API, for the VSCode project.

Github has a completely open API which can be queried by anyone even without any credentials, but it is limited. Credentials are required only to have a high allowance of queries.

To collect the data, I will be using mainly three end points:

`https://api.github.com/repos/microsoft/vscode/issues` for issues
`https://api.github.com/repos/microsoft/vscode/pulls` for pull requests
`https://api.github.com/repos/microsoft/vscode/releases` for releases

A comprehensive list of endpoints is available in the official Github API documentation (*Endpoints available for GitHub Apps* 2021).

Each of the endpoints have all the data needed to perform my analysis. The main attributes of interest for each of the endpoints are `title`, `body`, and `created_at`. On top of that, the endpoints contain more data which will be considered to be included in the analysis, or will be used in excluding certain results from the analysis. For example, the `pulls` endpoint contains a field called `type` which can be either *User* (individual user / contributor) or *Organization* (Microsoft). This can be included in the analysis to determine if the type of user who contributes the pull request has any influence.

In order to be able to build and test smoothly, I plan on first download all the needed data from the Github API in `.json`, to avoid hitting API query caps.

10.1.5 Methodology & Analysis

The methodology used for the analysis will be KDD (Knowledge, Discovery & Data Mining). KDD has the advantage of being iterative. Since the project will require 3 data sets to be blended into one, the KDD process can be applied to each data set and then again at each step of combining the data in order to ensure that each step of the process is done correctly and that the output is relevant. This will also help in exploring the viability of including additional fields from any endpoint into the analysis, such as the `type` field mentioned above in the *Data* section. Also, similar methodology has been used in Deep learning in static, metric-based bug prediction (Ferenc et al. 2020) which explores bug prediction via deep learning algorithms.

KDD has five main stages: Selection -> Pre-processing -> Transformation -> Data mining -> Evaluation. These will also be the main project activities and be part of the milestones.

Selection

For selection, I have identified the three main API endpoints which I will be using, and the minimum fields needed in order to perform the analysis. On top of that, I will explore the possibility of using additional fields from each endpoint which may be fit to the analysis.

Pre-processing

At this stage, I will collect the raw data from the API and storing it into `.json` files. After that, the data can be easily loaded into python for further processing. Here, I will isolate only the attributes which are of interest for the analysis so the data set will be ready for the next step.

Transformation

In order to be processed, the data needs to be curated first. The main approach used will be trimming the most common words in the English data from the `title` and `body` attributes content and storing the remaining words in arrays. To have the data ready for the next step, I will combine the three separate data sets into one, with relations between them by the date of the release, then the date of the pull request

Data mining

At this stage the data will be split between one training set and one testing set. Then, the data set will be fed to a machine learning algorithm which will attempt to identify correlations between certain types of pull requests and certain types of issues.

Interpretation

Here we will look at the results of the previous step and draw conclusions about the success and viability of the method.

10.1.6 Technical Details

I will use python as a primary language for the project. Python is one of the most popular languages for data science (Costa 2020) and it has a wide variety of libraries used in data analysis.

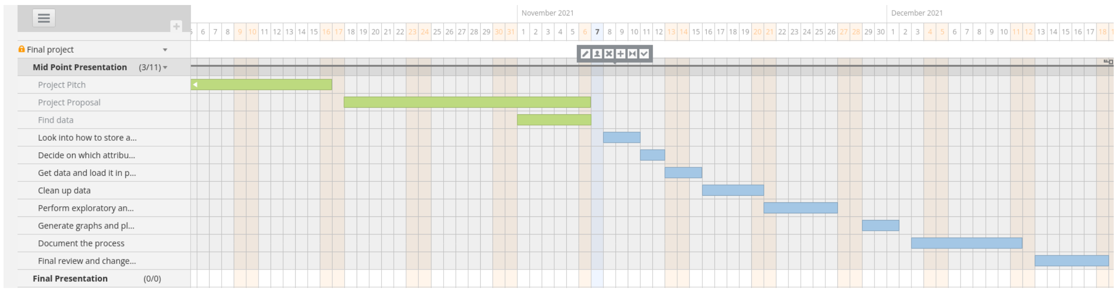
A few of the libraries which I will use in the analysis are:

- numpy - will be useful when manipulating arrays
- json - to be able to interact with the JSON format in which data will be after being retrieved from the API
- datetime - to extract and manipulate the date and time after processing the json files
- wordcloud - will be used in some of the earlier visualizations. After processing the `title` and `body` text attributes and filtering out the most common words it will be interesting to visualize the remaining data in a word cloud format
- matplotlib - to create graphs

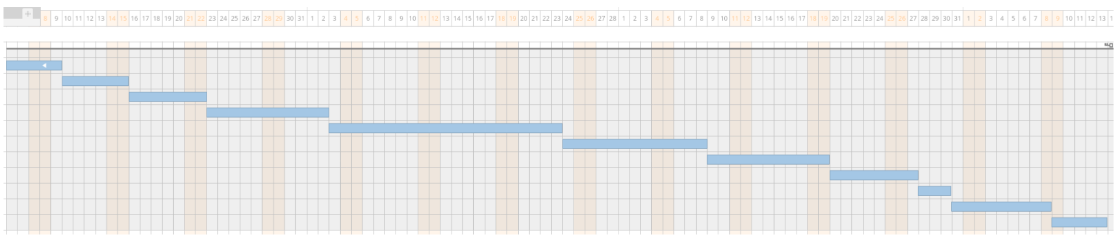
I will use either *python* or *bash* to automate the process of getting the data from Github's API and into files.

10.1.7 Project Plan

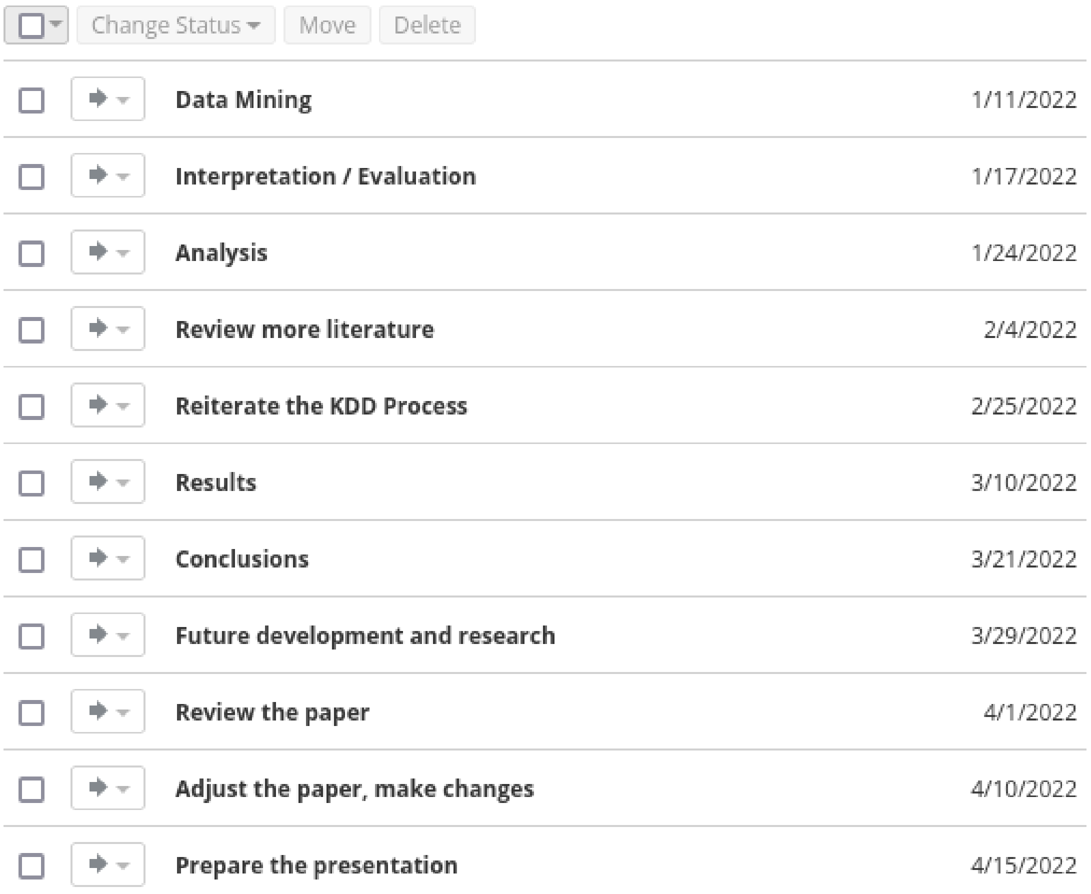
Below we can see the plan from October until December, for the mid point submission and presentation.



The plan from January to April, when the paper should be in its final stages:



And the task list for January - April:



Data Mining	1/11/2022
Interpretation / Evaluation	1/17/2022
Analysis	1/24/2022
Review more literature	2/4/2022
Reiterate the KDD Process	2/25/2022
Results	3/10/2022
Conclusions	3/21/2022
Future development and research	3/29/2022
Review the paper	4/1/2022
Adjust the paper, make changes	4/10/2022
Prepare the presentation	4/15/2022

10.1.8 Bibliography

Costa, Claire D. (Aug. 24, 2020). *Top Programming Languages for Data Science in 2020*. Medium. URL: https://towardsdatascience.com/top-programming-languages-for-data-science-in-2020-3425d756e2a7 (visited on 11/07/2021).

Endpoints available for GitHub Apps (2021). GitHub Docs. URL: https://docs.github.com/en/rest/overview/endpoints-available-for-github-apps (visited on 12/19/2021).

Ferenc, Rudolf et al. (July 1, 2020). "Deep learning in static, metric-based bug prediction". In: *Array* 6, p. 100021. ISSN: 2590-0056. DOI: 10.1016/j.array.2020.100021. URL: https://www.sciencedirect.com/science/article/pii/S2590005620300060 (visited on 12/19/2021).

GitHub Bugs Prediction Challenge (2021). URL: https://machinehack.com/hackathons/predict_github_issues_embold_sponsored_hackathon/overview (visited on 11/03/2021).

Visual Studio Code - Open Source ("Code - OSS") (Dec. 19, 2021). original-date: 2015-09-03T20:23:38Z. URL: https://github.com/microsoft/vscode (visited on 12/19/2021).

10.2 Reflective Journals

10.2.1 October

What?

October is the fist month for the project so it was mostly spent researching the topics I would be interested in.

Since I am working in software development, both of my main ideas revolved around exploring data which is related to the field. One of my ideas was to explore how contributions to open source projects have been affected by the COVID pandemic, more specifically to compare contribution from large companies against contribution from individual volunteers. But when looking more carefully at the idea, it becomes obvious that there aren't many options for predictive analysis, even if the findings might be interesting to look at.

Main project I will use is VS Code. It has over 5 000 open issues, over 10 000 pull requests and 50 releases at the time of writing this document. Pull requests and issues will be grouped by date, according to the release dates. That way, any specific release will have certain pull requests which were merged before the date of release. The issues reported between releases will be assigned to the release which immediately predates them. Then I will perform analysis on the tags associated with pull requests, release and issues. On top of that, the titles and text of pull requests and issues will be processed by stripping them of common English words in order to extract key words. These words will be added to the tags so that they can be included in the analysis. The goal is to try to determine what kind of issues might arise from different type or pull requests, releases, or in general, what kind of issues will appear when adding certain features.

This can be extremely useful when preparing new features for a software project because you can preemptively take actions to prevent certain bugs / issues from appearing. So I decided to go with my second idea, which is to use data collected from github about large open source projects in order to try to predict bugs. Github API data is freely available to use. I plan on getting project data for at least one large open source project.

First, I had to submit a short project pitch in video form. Next, we were assigned project supervisors.

So What?

Currently I am waiting on a final approval for the project, from my project supervisor and from a second reviewer.

In the meantime, I am exploring the technologies I can use. I will most likely use *python3* and it's data processing specific libraries, such as pandas, *numpy* and *ntlk* or *tensorflow* for text analysis. For data visualization, I will make use mainly of *matplotlib* for numerical data and *wordcloud* for word cloud type visualizations of tags and key words in releases.

Now What?

The next step is to get the data from the github API. I have already tested the api endpoints I will be using to make sure that I can get everything I need. Next, I will set up a Jupyter Notebook for my project. Since I will be getting a lot of information from the github API, I plan of getting the data separately and saving it in *.json* files, and importing and using that in *Jupyter Notebook*. This will avoid me reaching github API quotas while testing and running my code.

10.2.2 November

What?

This month I received approval for my project idea. At the beginning of the month, I also submitted my Project Plan.

The Project Plan covers objectives, background, state of the art, data, methodology and technical details. At this point I was not able to go int a lot of details regarding the data or analysis since I was not yet very familiar with the dataset or with what analysis I was goint to perform.

I did know where the data sets are going to be sourced from though, and I was able to come up with a comprehensive project plan for all the steps until the mid term submission.

After the Project Plan submission, I started work on sourcing the data sets. This has been achieved by using three python scripts which access the *GitHub API* and get the JSON data in three *.json* files. The data sets total 1.3 GB of drive space and they will be loaded in *Data Spell* IDE and perform analysis using *python*. I was also able to look into the data sets in *Data Spell* and identify which attributes are going to be most useful and do the initial clean up of the data, alongside with some exploratory analysis.

So What?

This sets up the stage for the Technical Report which will be submitted as part of the midterm submission in the end of December. Getting to the point of having some descriptive statistics of the data sets and the basic exploratory analysis was important as now I have a good idea of how the data sets look and how I will approach the project and the Technical report.

Now What?

Next is to start writing the Technical Report and fine tune the technical part of the analysis. More descriptive statistics can be added to have a better idea of what the dataset contains. I should also do more research on methodologies and analysis and try to identify more exploratory analysis methods.

10.2.3 December

What?

December is the last month of the semester and the submission month for the midterm project. The month has been busy with work on the project (and other submissions). Good progress has been made as part of the midterm submission and the project is beginning to take shape.

I have done work on the first four sections of the technical report: *Introduction*, *Data*, *Methodology* and *Analysis*.

I started to work with the data sets, which means that I was able to expand on the initial Project Proposal with actual results for the *Data* section, specifically descriptive statistics to have a look at the data sets and some exploratory analysis. For *Methodology*, I was able to fill in the *Selection*, *Preprocessing* and *Transformation* sections. And for *Analysis*, I covered the theory as an intro.

As part of the submission, my Project Plan received a small update as well.

So What?

This has been all part of the mid term submission due at the end of December. All of the sections are still a work in progress, since as I go forward with the analysis I will be able to fill in more sections - *Analysis* covers only the theory intro for example.

Now What?

Next I will focus on filling in the Data and Methodology sections while classes catch up and I learn how to implement analysis methods as well. *Methodology* section needs more input and info regarding similar academic work. And *Data* will need more examples for descriptive statistics and exploratory analysis.

10.2.4 January

What?

January was split between exams and off from college between the two semesters. During the exam period I did not make progress to the project, but during the break I reviewed my mid term submission to find areas where I can improve.

So What?

The main area of improving is adding more academic research references. I need to support and motivate my approach better, and academic references will help with that. Had my first supervisor meeting as well, but there was not much to report and talk about since the results are not out yet at this point.

One other thing to complete is the *Project Showcase* section, so I took advantage of the downtime and filled in that section.

Now What?

Mid point submission feedback will be out at the beginning of February. The supervisor meeting will be in the same day, so that will provide me with a good direction to take when starting out working on the project again.

10.2.5 February

What?

February is the kick-off month for the new semester. This month I also received mid-term grade and feedback. After that I was able to start working again on the project, with a better idea of the direction I should be heading towards.

So What?

The grade I got for the midterm submission is not what I would have wished for, but expected considering the circumstances - first semester was overloaded with assignments and assessments, and I ended up rushing a few sections of the submission in order to meet the deadline.

I also received valuable feedback from the supervisor. During the scheduled feedback meeting the supervisor pointed out where I am lacking, areas such as not being clear enough about my objectives and lacking a clear plan for the whole approach. Aside from that, the supervisor directed me to look into alternative approaches. I will be checking quality of pull requests against the user name, and also check for the *PR diff* and see how that correlates with issues in the code base.

Now What?

I will be cleaning up certain points of my submission to clarify the sections which were pointed out by my supervisor. Then I am going to focus on my data set and try to identify the best methods to use when processing the data.

10.2.6 March

What?

March consisted on building up the technical part of the project. During this period, classes also proceeded and we touched on and learnt new methods which will prove themselves useful in my analysis.

So What?

The new methods learnt have helped in doing analysis on the quality of a pull request against user names. I also plan on looking at *PR diffs* and correlate them to issues in order to find patterns, especially related to *diffs* of certain sizes, or *PRs* done in certain sections of the project.

Now What?

Next steps will be to try to finalize the technical part and have a fully formed analysis code-side of the project. Then I can start writing down the analysis part of the project and the conclusion. I also have to do a sweep of the code and see what parts might be redudant and remove them, and refine the whole analysis, since currently I have different attempts and approaches for the same types of analysis.

10.2.7 April

What?

April was the month I finished off the technical part of the project. I ended up focusing on *Random Forest* for the ML part of the analysis.

To prepare the data, I applied the KDD process a few times, since I wanted to test out different attributes and how they influence the quality of the predictions.

On top of that, the data needed additional transformations to prepare it for the analysis. I used the most occuring labels from *issues* as the dependent variable and then used the text from the body of the *PRs* as an independent variable.

So What?

The results for the predictions are not satisfactory, and this is a strong indicator that simply using information from the PR body, title or user is most definitely not sufficient if the purpose is to predict bug occurrences or certain types of bugs.

More analysis on the actual code itself could help more with this. From the data transformation process I learnt that the way data is registered and structured by github is not good for performing any kind of data driven analysis. There was a lot of data transformation necessary in order to get the data in a usable form.

Now What?

Currently the technical report is being written and about to be finished soon. I would like to do a bit more analysis and experiment more with the data if time allows. Working with such a vast data set has been interesting and was an opportunity to learn much about scraping, transformation and data mining.

There have been a few points where I had to employ methods tailored to the fact that my data set was large, so that processes wouldn't end up taking hours, and working with different data sets scraped by me forced me to think out of the box when creating a usable data set.