Group Coursework Submission Form

Specialist Masters Programme

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Please list all names of group members: (Surname, first name) 1. A-Un, Lily 2. Chen, Yujie 3. Joshi, Rutva Dharmendra	4. Kapoor, Karishma 5. Low, Jia Meng 6. 7. **GROUP NUMBER:** 11
MSc in: Business Analytics	
Module Code: SMM695	
Module Title: Data Management Systems	
Lecturer: Matteo Devigili	**Submission Date: 21 July 2023**

Marker's Comments (if not being marked on-line):

Deduction for Late Submission:		**Final Mark:**	%

Report

SMM695- Database Management Systems

CONTENTS

Task I- Cleaning, Manipulating data

We designed a database to efficiently manage and track bug reports and their associated information. Our database consists of seven tables, each serving specific purpose in organizing and recording bug-related data.

Tables Overview:

1. **Bugs Table-** It stores detailed information about individual bugs, including their unique identification numbers (id), status, priority level, severity, type, classification, platform, product, component, and resolution. This table helps in monitoring the progress of bug and will help in prioritizing resolution based on priority & severity.

2. **Employees Table-** It contains information about the employees or users involved in the bug tracking process. It records bug id, name, email, nickname and real name.

3. **History Table-** This table keeps a comprehensive history of changes made to bug reports. It shows evolution of each bug's lifecycle and provides record of updates, such as status changes and additions of test cases. It records bug id, editor, edit date, changes added, changes field, changes removed.

4. **Involver Table-** It tracks the involvement of various individuals in the bug resolution process. It records the assignee responsible for handling each bug, the creator of the bug report, and the number of votes or support each bug has received.

5. **Time Table-** It shows time-related data for each bug report, including creation time, the most recent change, and the date of the last resolution.

6. **Bug_fixed_cleaned_history Table:** This section of the report outlines the process of creating and refining the history table named `bug_fixed_cleaned_history` under the `bugzilla` schema. The table was curated to capture the lifecycle of bug fixes in a streamlined manner and was used as the foundational data source for our comprehensive analysis of bugs that had been assigned, reopened, and ultimately resolved.

Data transformation was a key step in preparing our dataset. This process began by implementing a time-lapse evaluation between subsequent edits on the same bug. Using the SQL window function `LEAD`, we computed the `end_date` for each edit and the number of days (`days`) between current and next edits.

Next, we categorised the assignment status of bugs. We added an `assignee` column, which was derived from the `changes_added` field. This new attribute helped us classify bugs as 'Pending' when they were new or had not undergone any changes, or to carry over the status from a previous row when the bug was reopened.

Subsequently, we amalgamated data from multiple sources, including the historical records of changes and creation times for new bugs, into a comprehensive dataset. This combined dataset, obtained via a nested SQL subquery, gave us a holistic view of a bug's journey from its creation to its current status.

We then filtered this combined dataset to focus on bugs that had been fixed, had a status of 'NEW' at some point, and had been reassigned at least once. This filtered data provided the basis for the `bug_fixed_cleaned_history` table.

Once our transformed data was in place, we moved on to data refinement. This phase included adding a unique identifier for each record in the table. The `history_id` column was auto incremented for each row and established as the primary key of the table, ensuring a unique reference for each record.

We then performed a data cleaning operation, eliminating any rows in which `end_date` was either null or earlier than `edit_date`. These rows were considered invalid due to the chronological sequence expected in a bug's lifecycle.

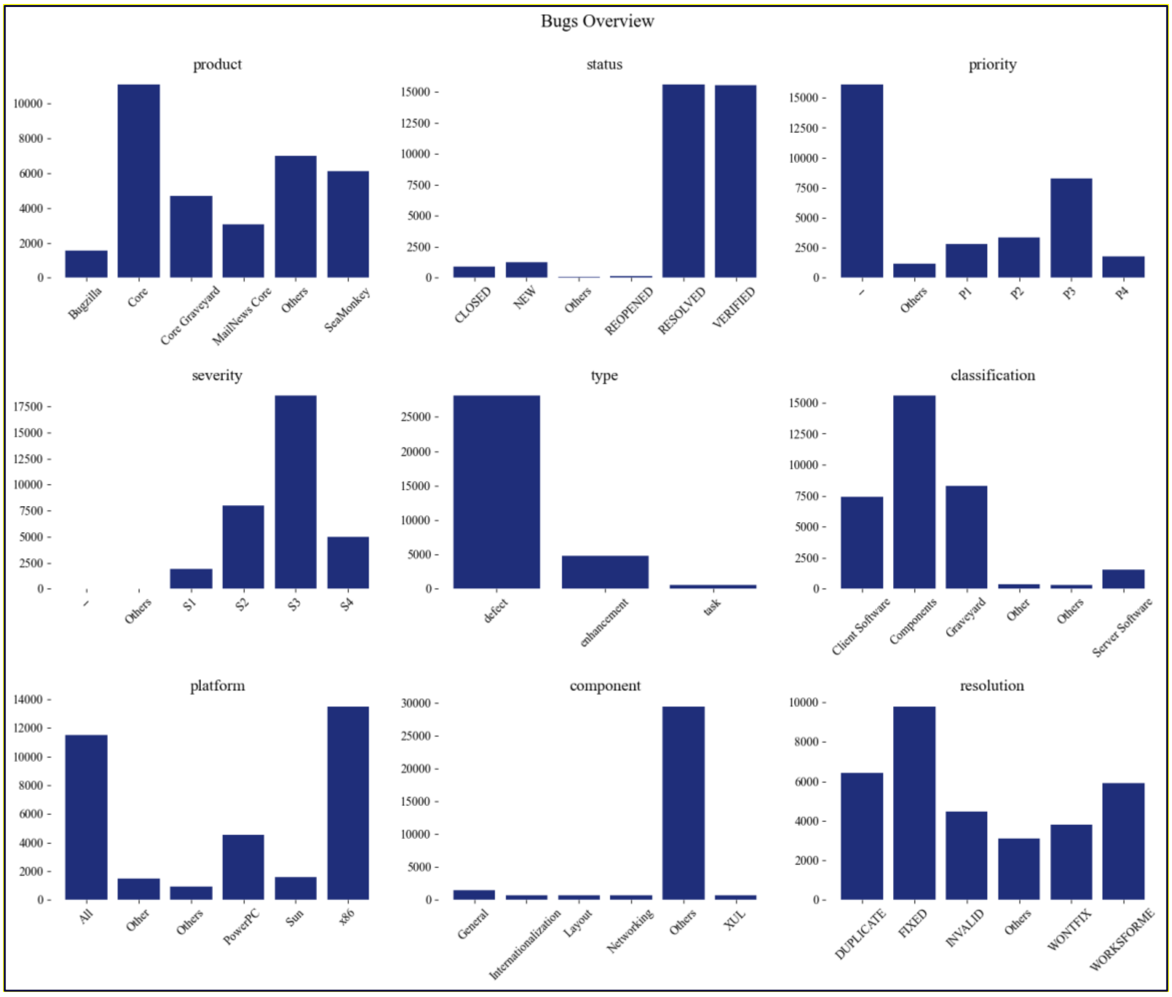
To validate our data cleaning process, we executed a check to ensure no remaining records violated our rules concerning `end_date` and `edit_date`. A null result from this check confirmed the successful cleaning of our data.

In conclusion, we successfully transformed and cleaned our data in preparation for our study. Our refined dataset, `bug_fixed_cleaned_history`, accurately captures the lifecycle of bug fixes and provides a robust base for effective and insightful analysis.

Task 2- Descriptive Statistics

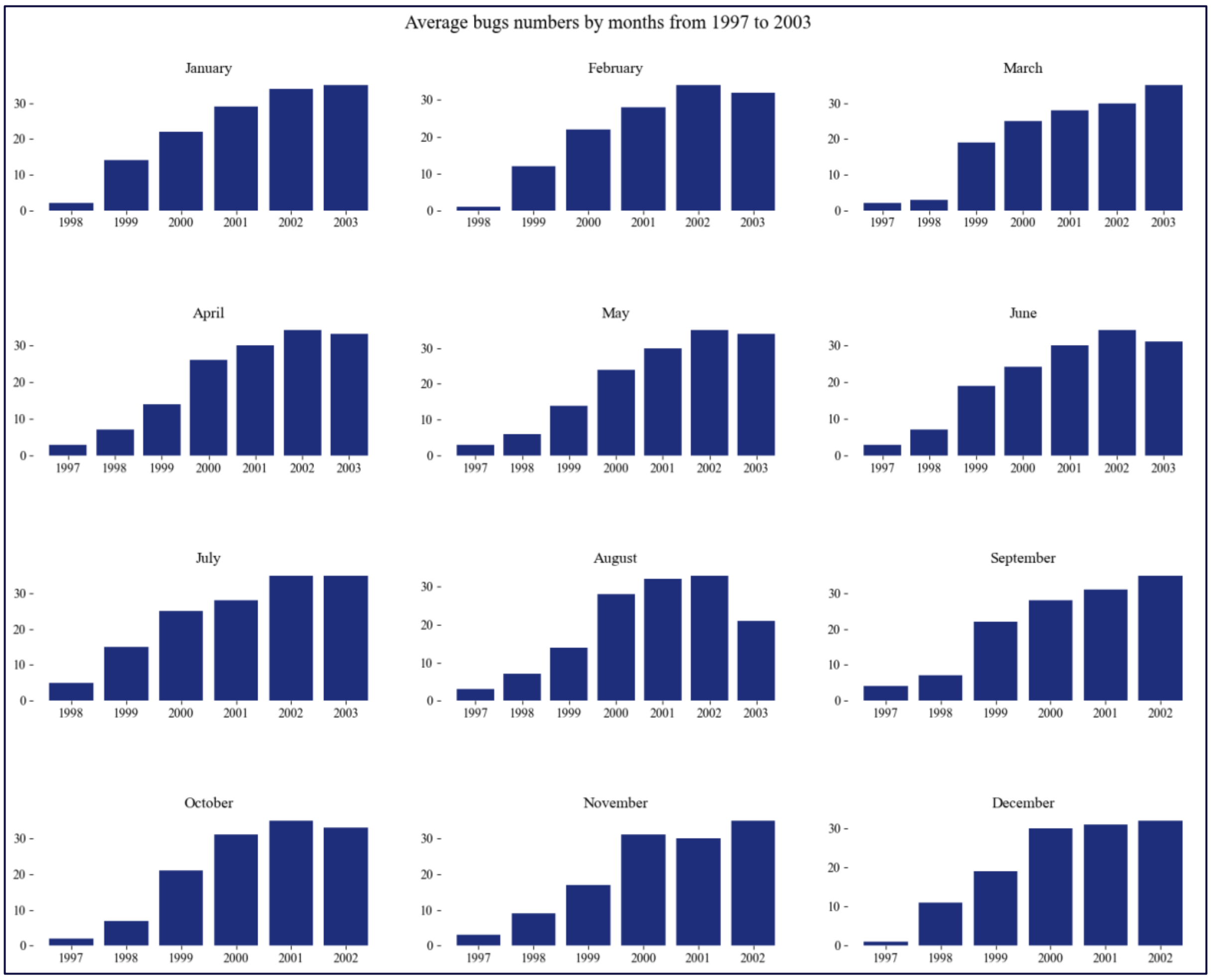
1. Overview of the bugs

To begin, let's take an overview of the bugs over the entire period, as shown in **Graph 1:** The top 3 products with the highest number of bugs are Core, Others, and SeaMonkey. The majority of bugs have been verified and resolved. Regarding priority, most bugs were not assigned any priority, followed by P3, backlog, and P2 (fix in the next release cycle or the following: nightly + 1 or nightly + 2). Additionally, the majority of bugs have a severity level of S3 (Normal) and fall under the defect type. In terms of classification, most bugs belong to the "Other" component. Furthermore, the highest number of bugs is reported on the X86 platform, and the common resolutions include "fixed," "duplicate," and "worksforme."



Graph 1

Upon delving deeper into the bug numbers, we have plotted the average bugs numbers per month from 1997 to 2003, as depicted in **Graph2.** From the results, we did not identify any specific months with significantly higher bug numbers compared to others. Instead, the trend indicates a gradual increase in bug numbers over the years, especially from 1999 to 2002.



Graph 2

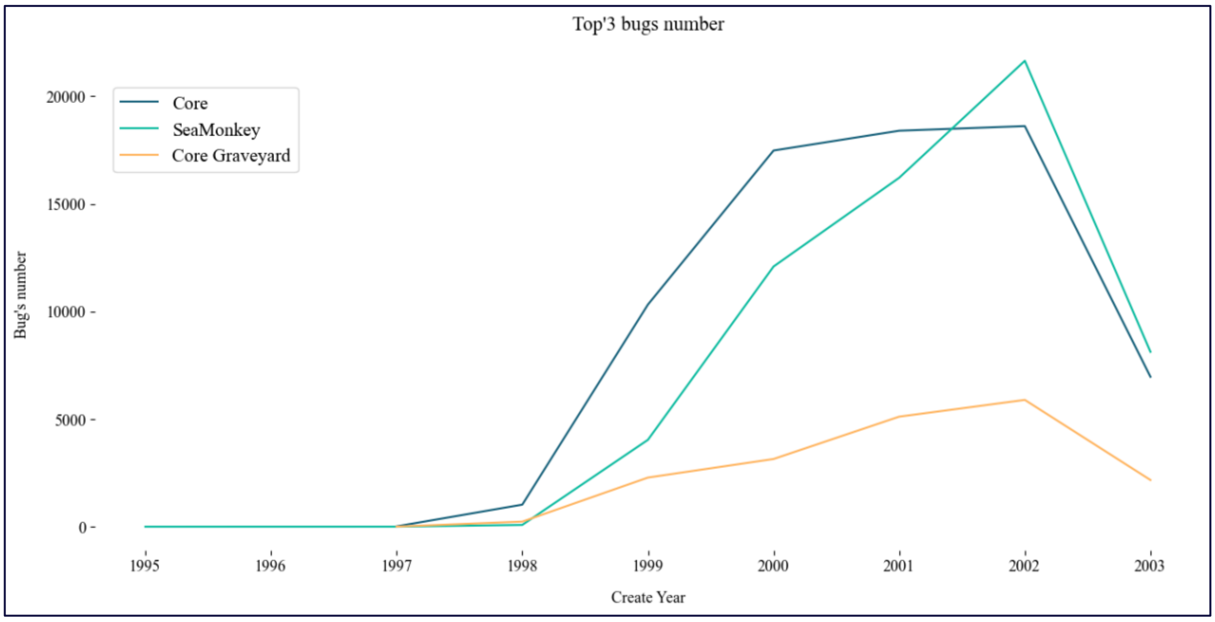
2. Time Trends

How have the number of Top3 bugs changed over the period?

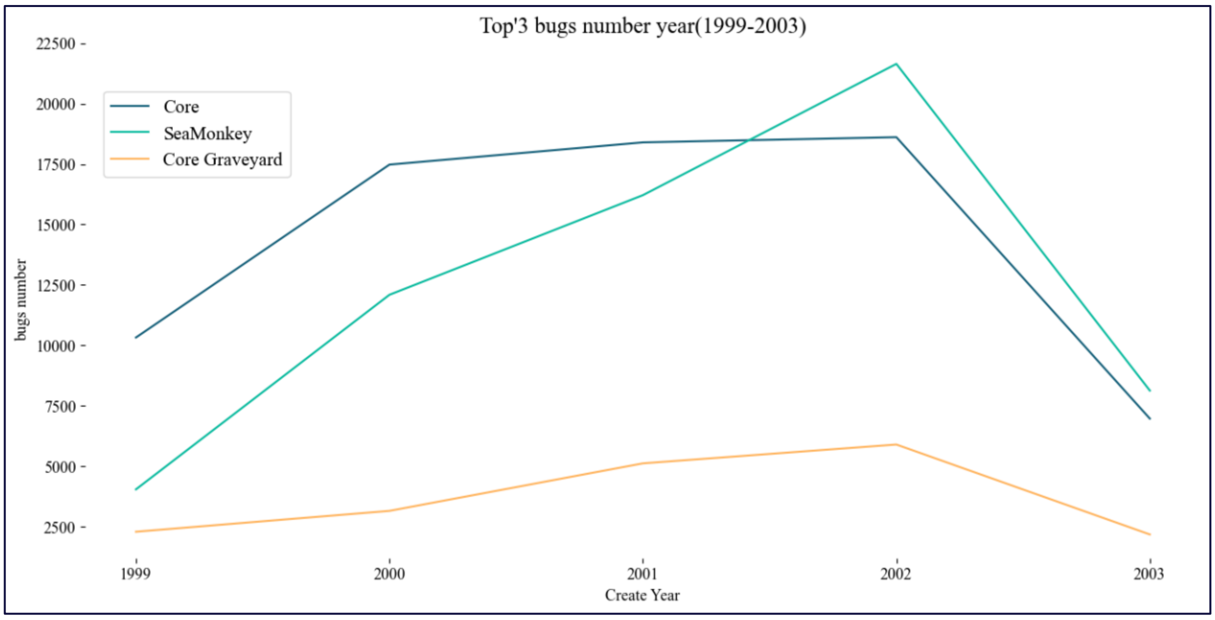
Starting with displaying the number of bugs by product over the period, we can observe that Core, SeaMonkey, and Core Graveyard account for a total of 76,124 cases (36%), 68,769 cases (32%), and 20,023 cases (9%) respectively out of the total of 213,313 bug cases. Therefore, these three products combined make up 77% of the total.

Next, when we consider the bug type with the highest number among the top three products, we find that the defect type has the highest count. Specifically, the defect type represents 72,820 cases (96% of the total) for Core, 62,200 cases (90% of the total) for SeaMonkey, and 18,845 cases (94% of the total) for Core Graveyard.

To visually represent the number of bugs for these top three products and the defect type over the entire period (1995 to 2003), we plotted a graph (**Graph 3**). The graph indicates a significant increase in bug counts for the top three products from 1997 to 1999. However, after 1999 to 2002, all products except "Core" continued to experience an upward trend. "Core" remained relatively stable during this period. Subsequently, from 2002 to 2003, all products witnessed a significant decrease in bug counts. To focus on the years 1999-2003, we created **Graph 4**.



Graph 3



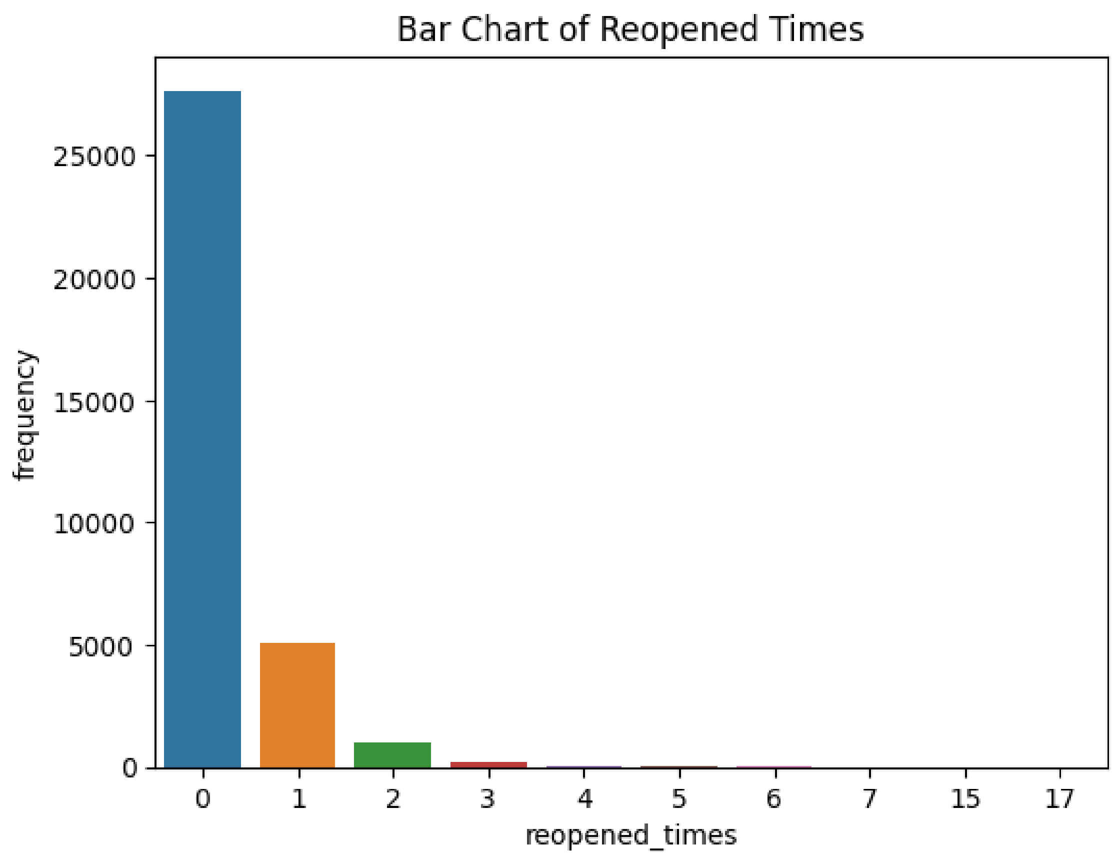
Graph 4

However, additional investigation is needed to identify other potential reasons for the stability of Core's bug cases during the mentioned period. For instance, changes in the employee team responsible for the Core product could be a contributing factor.

3. History Table

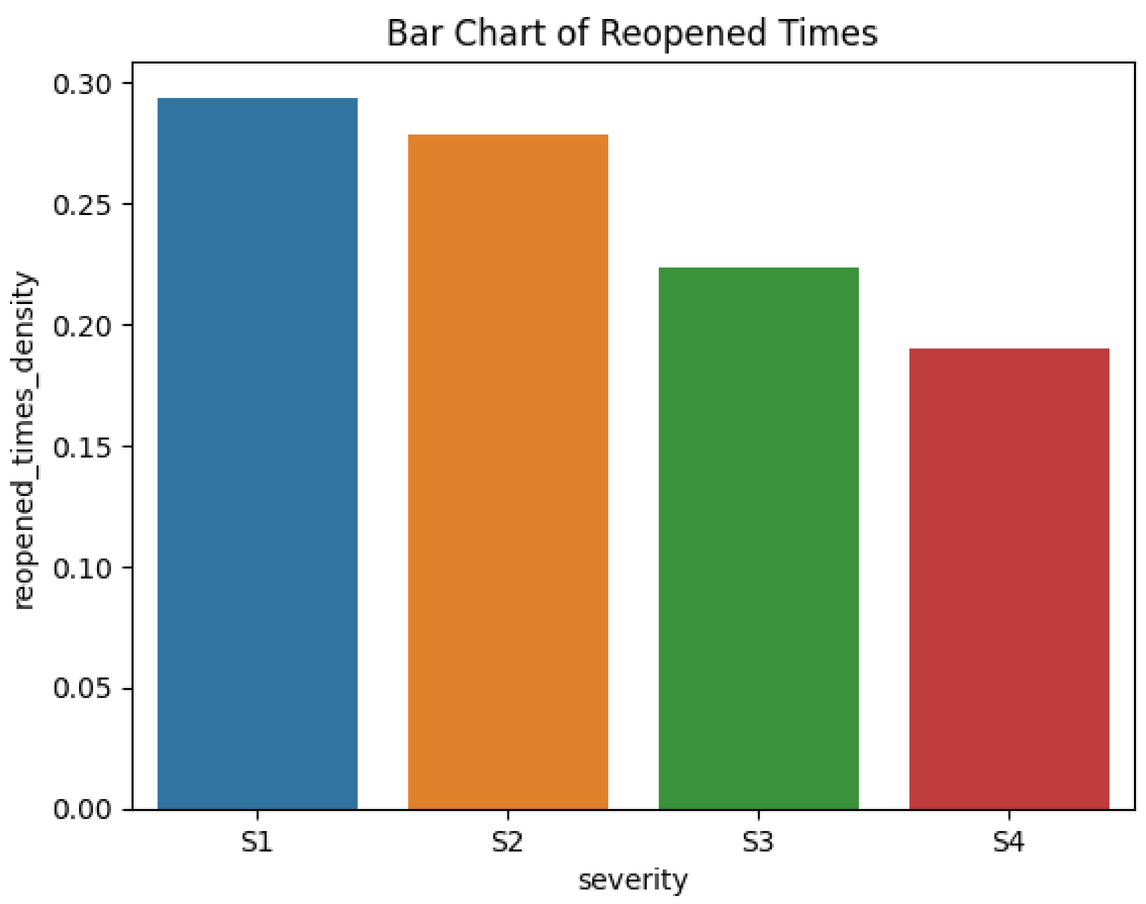
The frequency of reopening bugs

It is found that some bugs reopened after the first time they were resolved. It makes the situation more complicated because the difference between the creation time and the time of last resolved might not be accurate at all.



The histogram above indicates that reopened bugs are relatively rare in the dataset. With above 25000 bugs not being reopened at all, and about 5000 bugs only reopened once. Whilst, there are only a few bugs reopening more than twice.

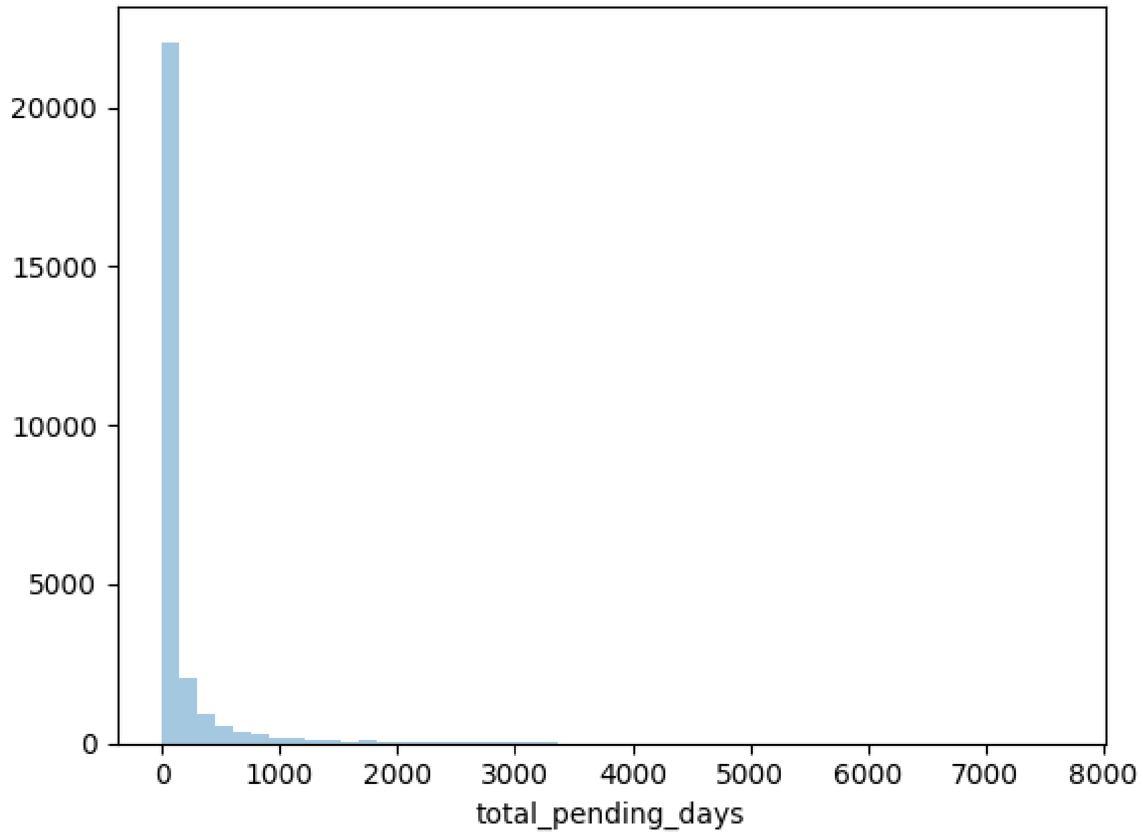
What types of bugs are likely to be reopened?



The bar chart shows the distribution of reopened bugs across different severity. Since the numbers of bugs in each category are different, density was used here. As the severity level increases (S1 being the highest), it is more likely for a bug to be reopened.

Pending Days

Another very interesting fact in the data is that bugs were not taken once they were reported. Bugs had to wait for some time to be assigned.

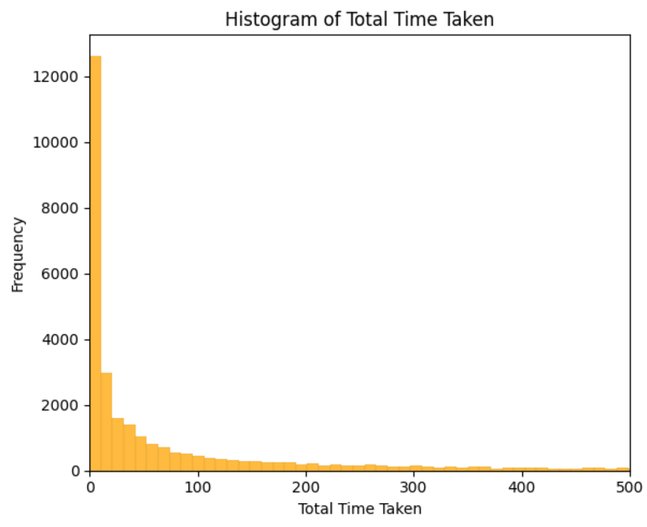


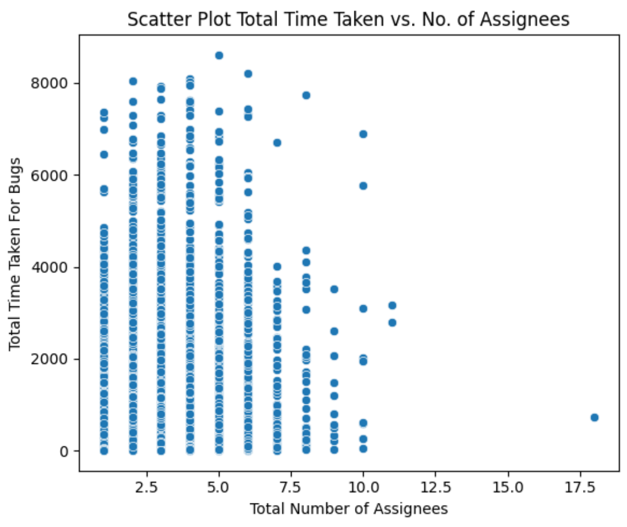
The distribution shown above indicated a heavily right-skewed distribution. With most of bugs began being fixed closely to 0 days, while there were also some bugs waited for years to be actually worked on.

By calculating the statistical values, the mean of pending days is to be 134.37 days while the median is merely 14 days, again indicating the heavily right-skewed distribution. And surprisingly, the maximum of pending days is 7629 days, about 20 years.

4. Average time taken in terms of people

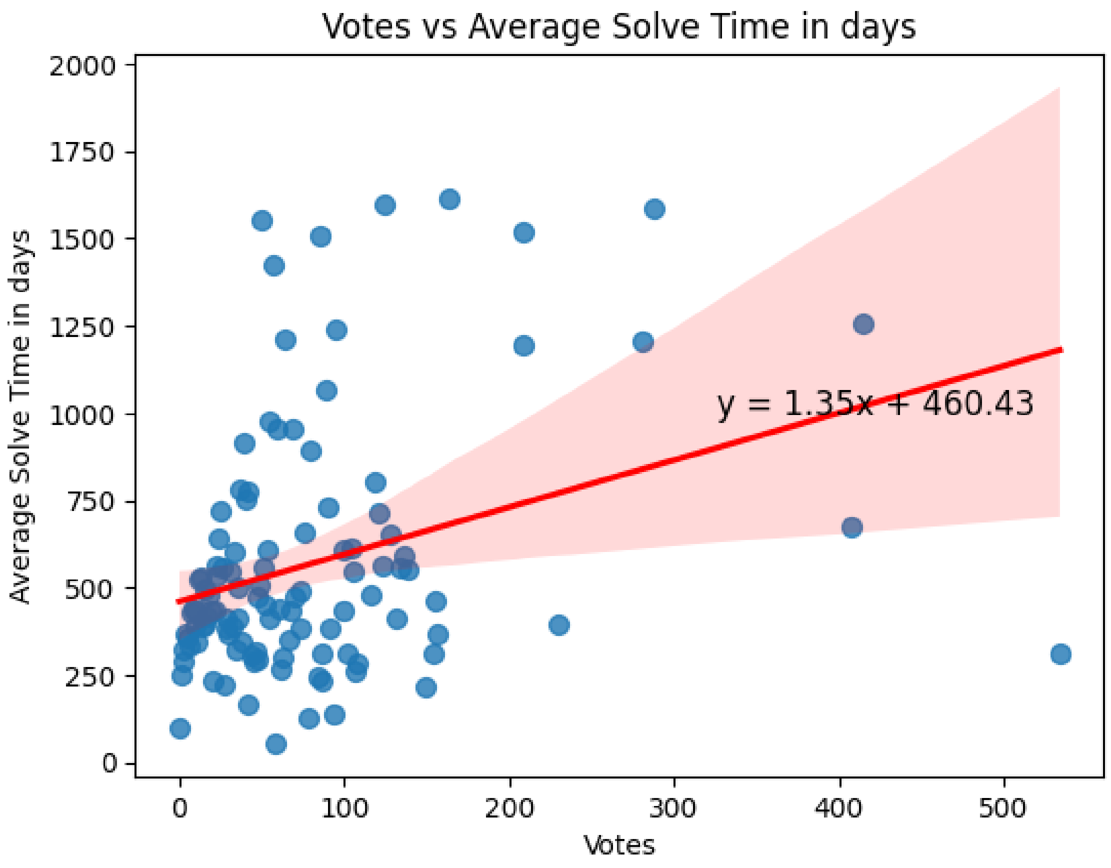
Total Time Taken vs. Total Assignee Involved





On the left-hand side, it can be observed that most bugs took short time to be fixed with more than 12000 bugs were fixed in the time range from 0 to 10 days. However, the distribution shows a heavy right tail. There is going to be some bugs with excessive amount of time absorbed.

On the right, the scatter plot shows a simple illustration about the relationship between total time taken and total number of assignees involved. Most bugs have less than 5 assignees throughout the fixing history, while the portion for bugs with more than 5 assignees is also considerably sufficient. The scatter plot shows a downward trend but not the trend is not evident enough.



The above visualization reveals a clear trend: as the number of votes increases, so does the average time to resolve a bug. This indicates that the complexity and time required to solve a bug is likely to attract more attention, and consequently, more votes from the community.

More complex bugs tend to intrigue and challenge the community, drawing their attention and votes. These complex bugs, which inherently demand more time to resolve, become a center of interest, inviting engagement from both directly and indirectly involved parties.

However, while our data does suggest a relationship between resolution time and vote count, it's crucial to remember that this trend could be influenced by additional factors. A more comprehensive analysis would be required to fully understand the intricate dynamics at play. The correlation observed provides an interesting starting point for further investigations into the matter.

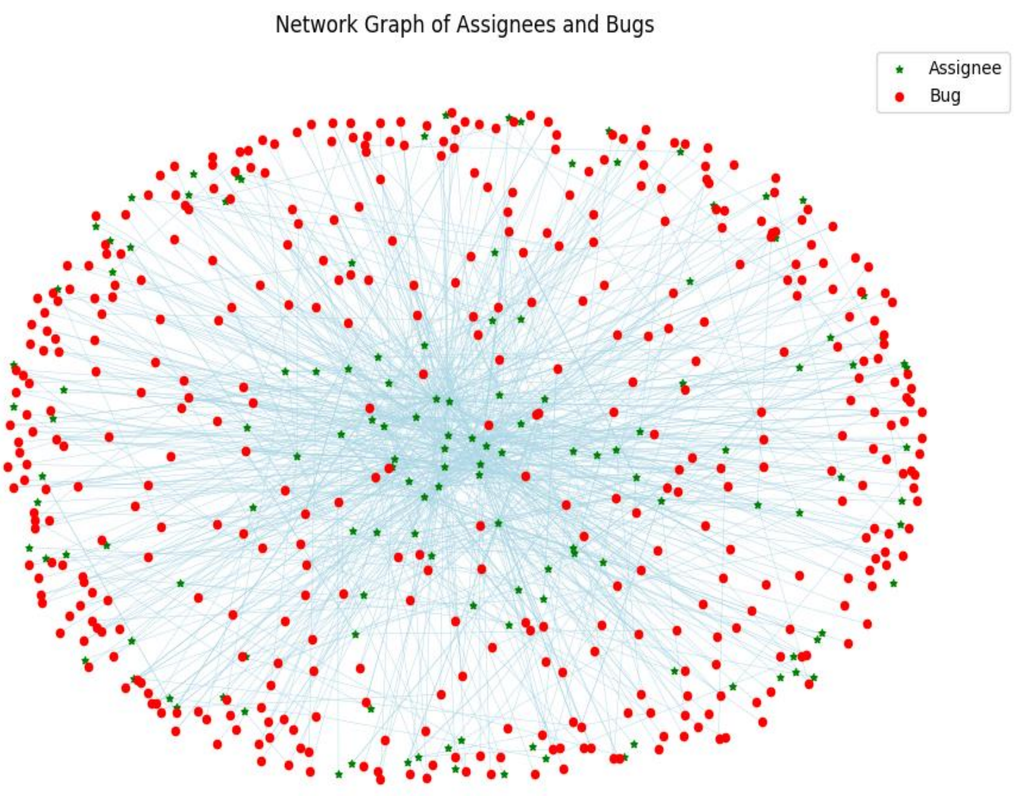
Task 3- Insightful data analysis

From the exploration in Part 2, analysis on time taken for bugs is considered to be most interesting topic to be further explored. Noted that in this part, the focus is only on the bugs which are verified fixed and if it was reopened, the bug has to be resolved in the end otherwise will not be included.

As mentioned earlier, the time taken for each bug can not be simply calculated the difference between the time of creation of the bug and the time of last resolved of the bug. Therefore, the first attempt in Part 3 is going to work around the database and try to retrieve the time each assignee spent on several bugs. Fortunately, the table from Part 2 can be of great use here.

Other than the new metric for time taken for bugs, metrics from network analysis were also adopted. Based on the information regarding assignees working on bugs, it becomes possible to find out the relation among assignees with edges being the common bugs assignees have been worked on together.

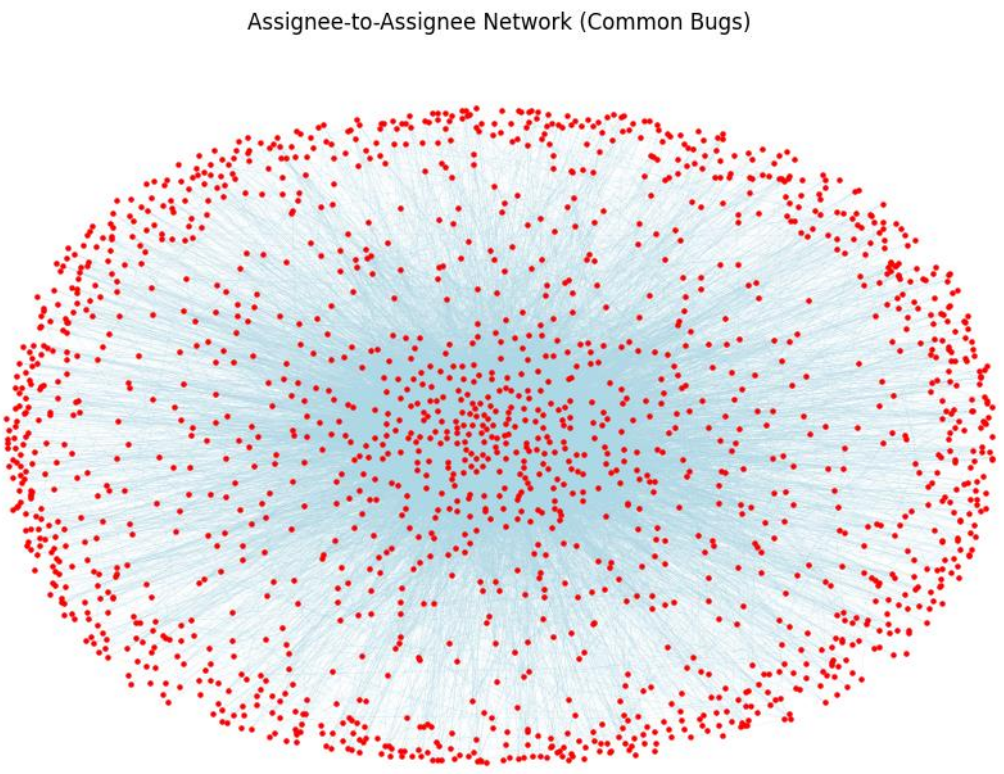
To start with, only the first 2032 rows were selected to be in the network graph for initial illustration. The network created here is a two mode network with one being bugs (marked red circle) and the other one being assignees (marked green star). In the graph below, the spread out for both types of nodes can be observed, with some assignees lying in the centre while some others in the outer border of the network.

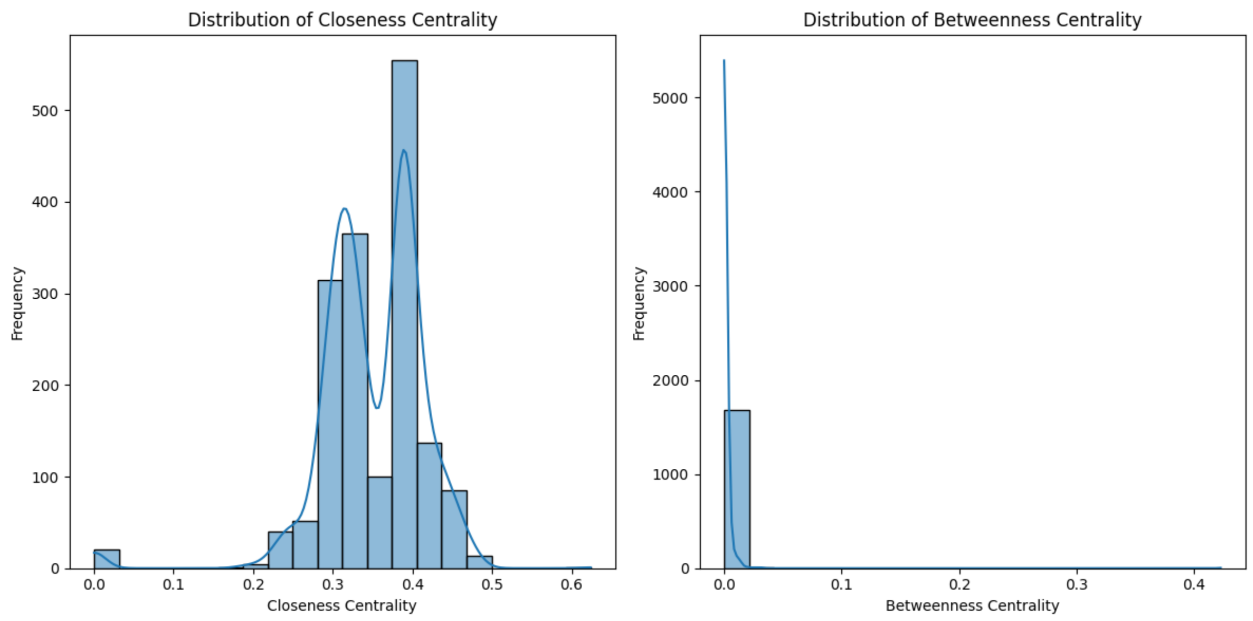


Although multiple mode network is powerful in capturing information in high dimension, it can be difficult to calculate the metrics such as closeness centrality and betweenness centrality. Hence, the two mode network was projected onto a single mode network with nodes only being assignees.

With the edges being the common bugs, the spread out for assignees in the network becomes clearer. Majority of assignees are on the edge of the network eclipse. Meanwhile, there are also many assignees lies in the centre of the network.

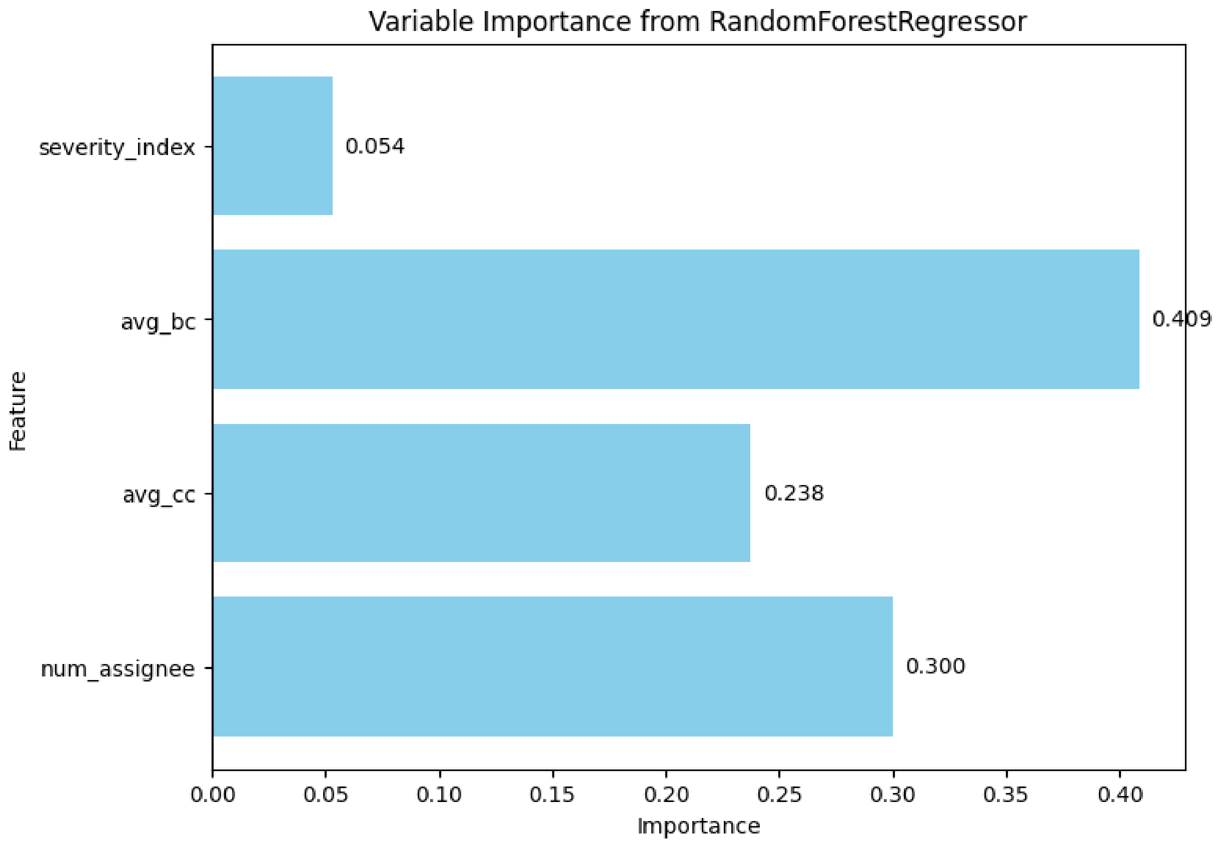
Closeness centrality and betweenness centrality were calculated to obtain numeric insights of the network. The distribution closeness centrality indicates a bi-model distribution with a huge gap in terms of frequency between 0.3 and 0.4. For betweenness centrality, it can be observed the betweenness centrality is low and close to 0 across the whole network. It indicates that there was limited number of brokers/bridges in the network. In practical tense, most people were focusing only on small portions of bugs instead of spreading their attention to a bigger picture.





Then, the average closeness centrality and betweenness centrality for each bug was calculated. Bugs with higher average value in closeness centrality indicates the people working on this bug are more of a crucial role in the whole network. While, bugs with higher average values indicated the people working on this bug are more of a broker/bridge in the whole network.

Finally, a random forest regression model was fitted to check the variable importance in the feature space of number of assignees involved in each bug, average closeness centrality, average betweenness centrality and severity. A training process was also implemented to check the power of the combination of these four features.



Although acting as a broker is not popular among the network, betweenness centrality matters the most when it comes to time taken for fixing bugs. It is a bit surprising to find out that severity does not play a vital role in the task. At the same time, number of assignees involved in the task also carry weights.

The random forest regression model only gives a R squared value of 0.3281, explaining about 32.81% of the variation in the dataset. But it carries valuable insights for the first attempt. To increase the accuracy, more variables should be included in the analysis, like priority, commentors, cc relations, etc.