

Framework for Software Testing Data Using Correlation Coefficient and Apriori Algorithm



Author

Muhammad Umar Sajjad

FALL 2017-MS-17(CSE) 00000205797

Supervisor

Dr. Usman Qamar

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD

August 2021

Framework for Software Testing Data Using Correlation Coefficient and Apriori Algorithm

Author

Muhammad Umar Sajjad

FALL 2017-MS-17(CSE) 00000205797

A thesis submitted in partial fulfillment of the requirements for the degree of
MS Software Engineering

Thesis Supervisor:

Dr. Usman Qamar

Thesis Supervisor's Signature: _____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD

August 2021

DECLARATION

I declare that this research work titled “*Framework for Software Testing Data Using Correlation Coefficient and Apriori Algorithm*” is my own work under the direction of Dr. Usman Qamar. The work has not been presented somewhere else for assessment. The material that has been used from other sources it has been appropriately acknowledged.

Signature of Student

Muhammad Umar Sajjad

FALL 2017-MS-17(CSE) 00000205797

LANGUAGE CORRECTNESS CERTIFICATE

Any type of spelling, typing, grammatical and semantic mistakes are not present in this research work. The thesis is formatted in accordance with the template provided for MS thesis work

Signature of Student

Muhammad Umar Sajjad

FALL 2017-MS-17(CSE) 00000205797

Signature of Supervisor

Dr. Usman Qamar

COPYRIGHT STATEMENT

- In the text of this thesis, the copyright belongs to the author. Any type of copy can be made by taking written permission from the author. Further copyright details can be taken from the NUST College of EME library. Any copy made must contain this copyright page.
- Further information about copyrights can be obtained from Library of College of EME NUST.

ACKNOWLEDGEMENTS

With the blessings of **Almighty Allah (SWT)**, I was able to do this work. Indeed, none be worthy of praise but the Almighty Allah. In addition, my admirations be upon **Holy Prophet Hazrat Muhammad (PBUH)** and his Holy Household.

I would like to express my special thanks to my supervisor **Dr. Usman Qamar** for his generous help throughout my thesis, and for being available even for the pettiest of issues.

I am thankful to **Dr. Wasi Haider** and **Dr. Urooj Fatima** for an excellent guidance throughout this journey and for being part of my evaluation committee.

Lastly, I would like to thank the individuals who helped me in this period.

“I dedicate this to my phenomenal parents and siblings”

ABSTRACT

In this modern era of science and technology, the use of software and computer-aided programs has increased very rapidly. With the increase in use, the data and size of the computer software's are also increased. Due to which the collection of large amounts of software testing data to support the software development and maintenance process has become difficult. With the development of the software, there is a need to assure the quality of the software. Software testing is the only solution to find the quality of the software and there is a need to find the defects in the software before delivering it to the clients. Almost 50% of the projects failed due to low quality and poor software testing. So, based on this problem we realized the need to use the latest data mining techniques for predicting defects in software.

In this research study, we used Data Mining (DM) techniques to predict defects from the software testing data. With the help of data mining techniques, we can improve the reliability and quality of the software. First, we have identified some available software testing datasets and selected data based on the parameters and requirements of the research study. For this purpose, we have explored related studies in the Literature Review (LR) and identified some defect prediction datasets & techniques. Based on the literature review, we have found different defect prediction techniques and chose the best one for designing and implementing research methodology. After data selection, we found the correlation between the different parameters of the software testing dataset using the correlation analysis. Further, applied data cleaning & transformation for preprocessed data. Processed data contains on the continuous data, so we transformed data into discrete data while using clustering (grouping) techniques. Then we implemented Apriori algorithm under Association Rule Mining (ARM) technique for predicting defects in software testing data. Apriori algorithm provided the supports and confidence in multiple iterations, and we got more accurate results. This proposed framework is based on Market Basket Analysis (MBA) and found the most frequent defects while using Association Rule Mining.

Keywords: *Software Testing, Defect Prediction, Data Mining, Market Basket Analysis, Association Rule Mining, Apriori Algorithm*

Table of Contents

DECLARATION	i
LANGUAGE CORRECTNESS CERTIFICATE	ii
COPYRIGHT STATEMENT	iii
ACKNOWLEDGEMENTS	iv
ABSTRACT	vi
CHAPTER 1: INTRODUCTION	2
1.1. Background	2
1.2. Thesis Overview	2
1.3. Problem Statement	3
1.4. Proposed Solution	3
1.5. Objectives	3
1.6. Work Contribution	4
1.7. Research Questions	4
1.8. Software Testing	5
1.9. Data Mining Techniques for Software Test Data	6
1.10. Tools & Techniques	6
1.11. Summary	6
CHAPTER 2: LITERATURE REVIEW	8
2.1. Literature Review & Search Process	8
2.2. Factors of Software Testing Defect	9
2.3. Code Size Role in Defect Prediction	10
2.4. Impact of Code Size on Defect Detection	12
2.5. Tools & Techniques for Defect Prediction	13
2.6. Other Related Studies	14
2.7. NASA Available Datasets	16
CHAPTER 3: PROPOSED METHODOLOGY	18
3.1. Proposed Methodology	18

3.2.	Testing Data Collection	19
3.3.	Selective Data Extraction.....	20
3.3.1.	Spearman Rank Correlation	21
3.4.	Data Preprocessing	21
3.4.1	Handle Missing Data.....	22
3.4.2	Parameters Scaling	22
3.4.3	Parameters Discretization.....	22
3.5.	Model Creation and Implementation	22
3.5.1	Market Basket Analysis (MBA).....	23
3.5.2	Association Rule Mining Algorithm.....	24
3.6.	Apriori Framework Evaluation.....	25
3.6.1	Expected Outcomes.....	25
3.6.2	Evaluate Support & Confidence.....	26
3.7.	Summary.....	26
CHAPTER 4: MODEL IMPLEMENTATION		28
4.1.	Python Required Packages.....	29
4.2.	Import Testing Data	29
4.3.	Parameters Extraction Implementation.....	30
4.3.1.	Spearman Rank Correlation	30
4.3.2.	Interlinked Parameters.....	31
4.4.	Data Cleaning & Transformation	31
4.4.1.	Introducing Two New Parameters.....	31
4.5.	Discretization of Processed Parameters	32
4.5.1.	Labels Discretized Data	33
4.5.2.	Occurrence of Variables.....	33
4.6.	Implementation of Apriori Algorithm Based Model	34
4.5.1.	Apriori Algorithm Implementation	34
4.7.	Defect Rules Evaluation	35

CHAPTER 5: MODEL RESULTS & VISUALIZATION.....	37
5.1. Parameters Extraction Result.....	37
5.1.1. Dataset 1 Important Parameters Result	37
5.1.2. Dataset 2 Important Parameters Result	38
5.1.3. Extracted Parameters Visualization Dataset 1	39
5.1.4. Extracted Parameters Visualization Dataset 2	41
5.2. New Parameters Results	42
5.3. Discretization Results	43
5.3.1. Test Case Density Per 100 Lines Parameter	43
5.3.2. Defect Density Per 1000 Lines Parameter	44
5.4. Association Rules Analysis Results.....	44
CHAPTER 6: CONCLUSION AND FUTURE WORK.....	49
6.1. Conclusion	49
6.2. Future Work.....	50
References.....	51

List of Figures

- Figure 1.1:** Research Study Process
- Figure 1.2:** Testing Process
- Figure 2.1:** Literature Review Process
- Figure 2.2:** Research Papers Search Process
- Figure 3.1:** Proposed Approach
- Figure 3.2:** Testing Dataset 1
- Figure 3.3:** Testing Dataset 2
- Figure 3.4:** Testing Dataset 3
- Figure 3.5:** Correlation Analysis Process
- Figure 3.6:** "Spearman Rank Correlation" formula
- Figure 3.7:** "Market Basket Analysis" flow diagram
- Figure 3.8:** "Association Rule Mining"
- Figure 4.1:** Implementation process of Model
- Figure 4.2:** Required Packages
- Figure 4.3:** Testing data implementation
- Figure 4.4:** Correlation Analysis Implementation
- Figure 4.5:** Data Transformation Implementation
- Figure 4.6:** Data Transformation Implementation
- Figure 4.7:** Discretization Implementation
- Figure 4.8:** Labels Implementation
- Figure 4.9:** Variables Occurrence
- Figure 4.10:** Model Flow Diagram
- Figure 4.11:** Implementation of Apriori Algorithm
- Figure 5.1:** Correlation Analysis of Dataset 1
- Figure 5.2:** Correlation Analysis of Dataset 2
- Figure 5.3:** CodeSize Parameter Correlation Result (Data 1)
- Figure 5.4:** TestCaseNumber Parameter Correlation Result (Data 1)
- Figure 5.5:** DefectNumber Parameter Correlation Result (Data 1)

Figure 5.6: CodeSize Parameter Correlation Result (Data 2)

Figure 5.7: TestCaseNumber Parameter Correlation Result (Data 2)

Figure 5.8: DefectNumber Parameter Correlation Result (Data 2)

Figure 5.9: Two News Parameters

Figure 5.10: Test Case Density Support Level

Figure 5.11: Item sets Support Level

Figure 5.12: Final Defect Rules

List of Tables

Table 2.1: Identification of Defect Prediction Techniques & Datasets

Table 2.2: Identification of NASA Defect Prediction Datasets

Table 3.1: Testing Data Parameters

Table 3.2: Expected Outcomes

Table 5.1: Correlation Analysis Results (Dataset 1)

Table 5.2: Correlation Analysis Results (Dataset 2)

Table 5.3: Test Case Density Per 100 lines

Table 5.4: Defect Density Per 1000 lines

Table 5.5: Test Case Density Supporting Level

Table 5.6: Supporting Level of Item sets

Table 5.7: Supporting Level & Confidence of Item sets

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

1.1. Background

Software testing is an important part of software development because success or failure of a software product depends on it. With the improvement of the technologies, every single task shifted on the software's and applications. For implementing a successful and according to the requirements product, it's very important to perform software testing. It means software reliability & quality can be judge with the help of testing phase. Today, most of the software applications failed due to lack of quality & reliability and it's due to a large software system or enterprise system & a huge amount of data. Due to large software systems, software testing also generated a lot of data which based on the software size.

Software engineering is one of the most focused fields in this era. Due to the advancement in technology the amount of the data generated during the different phases of software development process have increased very rapidly. Specially in phase of the software testing and software quality check. To perform analysis at such an extensive data we need a different approach. We can use the different kinds of statistical techniques to find relationships between the different variables of both quantitative and qualitative data. The techniques like the correlation and association rule mining can be used to find the most related features and variables [1]. Similar kind of correlation and Apriori based association finding techniques will be used in this paper to find out the relationship between the line of codes and defects figured out during the software testing process.

Central focus should be to figure out the rules from old software test data set to run the activities related to SDLC in future. Test data is data which has been specifically designed to use in tests, typically of a software system. To attain the above stated goals Data Engineering based techniques should be used to purify, analyze, and express data using existing technology or developing new algorithms for the software testing process. This process is used to discover use full knowledge and information for software development team and other person involves in development activities like managers or technical leads [2].

1.2. Thesis Overview

This study is established for defect prediction using a software testing dataset. Software testing is one of the important parts of the software development life cycle (SDLC). Software development follows all the processes of SDLC, and software testing provided the quality of the software product. According to the Standish group chaos report, the success

rate for software projects was only 29% last year. More than 50% of these projects were failed due to Quality we cannot deliver what's client wants. So, we need to overcome this kind of problem and need to explore the latest data mining techniques for defect prediction. In this research study, first, we have explored the defect prediction techniques and proposed a data mining-based framework that can predict defects in software testing data.

For conducting this research work, first, we have analyzed different datasets and selected one dataset. So, identification of dataset and defect prediction techniques is the first step. After that, we found the most important features using Spearman Ran Correlation. Further, we have proposed a framework and implement it while using python language and packages. The complete research work was done under Correlation Analysis & the Association Rule Mining techniques.

1.3. Problem Statement

When we perform extensive testing at a large software system or enterprise system we got a huge amount of data, so it becomes difficult to find out the main defect factors which affect the performance of the software. All Parameters of the selected dataset have not been interlinked. Perform analysis on these Parameters are effects on cost and time. So, need to perform correlation analysis on these parameters.

1.4. Proposed Solution

Basically, the main identified problem of this research study is to defect prediction from a huge amount of software testing data. For this purpose, we need to choose a latest trends or techniques which can perform defect prediction automatically and save human efforts, cost, and time. Data Mining is one of the effective solutions which mostly used for getting important information from huge amount of data or raw data. In our case, we need to implement such kind of solution which can predict defects from the software testing data. We will use Apriori Algorithm under Association Rule Mining technique for solving this problem.

1.5. Objectives

For solving this problem, required to identification of defect prediction techniques and tools. There is a lot of software testing dataset available for defect prediction, required to explore the dataset and select the best one which related to the study. Also, there is need to find out the feature extraction techniques for getting the interlinked parameters or features in

the data. We also need to propose a defect prediction framework while using identified techniques such as Association Rule Mining.

1.6. Work Contribution

Most of the studies discussed different techniques for testing defect prediction and we have used Association Rule Mining. Finding an association between defects is the main contribution of our research work. We have also explored the defect prediction & classification techniques in data mining. Data Mining is one of the latest solutions to predicting and finding associations between testing defects.

We have designed a complete research study process step by step which indicates our research contribution in the testing defect prediction. We have implemented a new framework that can find associations between software testing defects. For this purpose, we have used the Apriori algorithm under the Association Rule Mining technique.

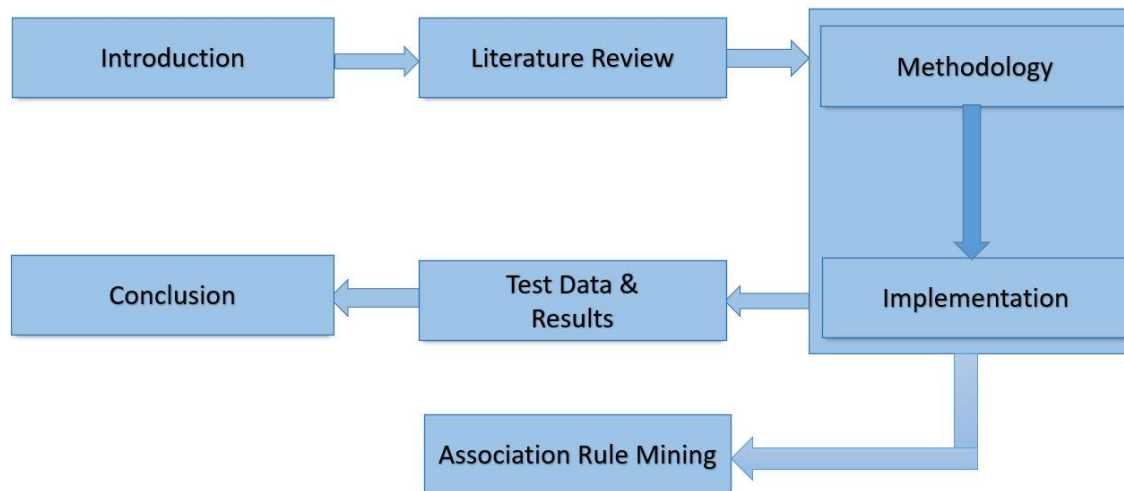


Figure 1.1: Research Study Process

Figure 1.1 shows the complete research study process, and it starts from the introduction chapter, follow up the literature review chapter in which we have explored the related studies according to the research study. After that, we have proposed a methodology and further implement the methodology. At last, we have discussed the results and compared them with the related studies.

1.7. Research Questions

Here in this section, we state what problem we're going to address. What questions are there in our minds to be answered while and after successfully conducting the research? And what hypothesis we have formulated. The hypotheses we have formulated during the research

study for the research proposal include the following:

RQ1: What are the factors which affect Software testing?

RQ2: What are the tools and techniques which can be used for defect prediction in software testing?

RQ3: What type of datasets are available for the prediction of the defects in software testing?

RQ4: How we can propose a framework which can predict defects in software testing?

1.8. Software Testing

In the software development life cycle, the success of software is dependent on software testing. Testers must confirm the software product should be error and bug-free. For this purpose, software testing generates a lot of testing data when testers perform testing to enhance the software quality and reliability. Figure 1.2 represents the complete process of software testing.



Figure 1.2: Testing Process

Requirement Analysis: This is the first stage of the testing process, and it is set-up for the identification of the required techniques and tools which verify the software testing requirements.

Test Plan Creation: Test planning includes delivering a report that depicts a general methodology and test goals.

Test Case Creation: The conditions and factors under which an analyzer will inspect if a product works effectively in little, fathomable test steps.

Test Case Execution: Test execution includes running the predetermined test on a system either physically or with the help of automatic tools.

Defect Logging: It is a process of discovering abandons in the application under test or product by testing or recording criticism from clients and making new forms of the item that fix the imperfections or the customer's feedback

Defect Fix & Re-Verification: When developer makes important code changes and confirms the progressions then analyzer do the retesting of the changed code which developer has given to him to check if the imperfection sorted out.

1.9. Data Mining Techniques for Software Test Data

Data mining technique can mine the important information from raw dat. "In this modern era of science and technology, the use of software and computer-aided programs has increased very rapidly. With the increase in use, the data and size of the computer software are also increased". Due to which the collection of a large amount of software testing data to support the software development and maintenance process has become difficult. When we test software lot of data is generated with different features, different data mining techniques are used to find the relation b/w Data Set Parameters.

1.10. Tools & Techniques

Defects Prediction implemented using Association Rule Mining & Apriori Algorithm. Implemented using some techniques and libraries. Details are given bellow.

- Google Colab
- MS Excel
- Python Language

1.11. Summary

The chapter is all about the introduction of the research work and we have discussed the background of the study. Also discussed the thesis context, problem statement, work contribution, and research questions. We have found there is a problem with the large software system and a huge amount of testing data. For solving this problem, we need to explore data mining techniques that can handle defects prediction in testing data. Defect prediction manually takes too much time, cost, and human effort. We'll handle these problems with the help of the Association Rule Mining technique and predict the defect rules.

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

In this chapter, we have focused on the related studies and try to find out the answers to some research questions. First, we have analyzed the factors which affect the software testing and identified them. Further, we have explored the studies which use the different datasets for defect prediction for getting the suitable dataset for implementation of defect prediction. Also explored the related tool and techniques which can use for defect prediction.

2.1. Literature Review & Search Process

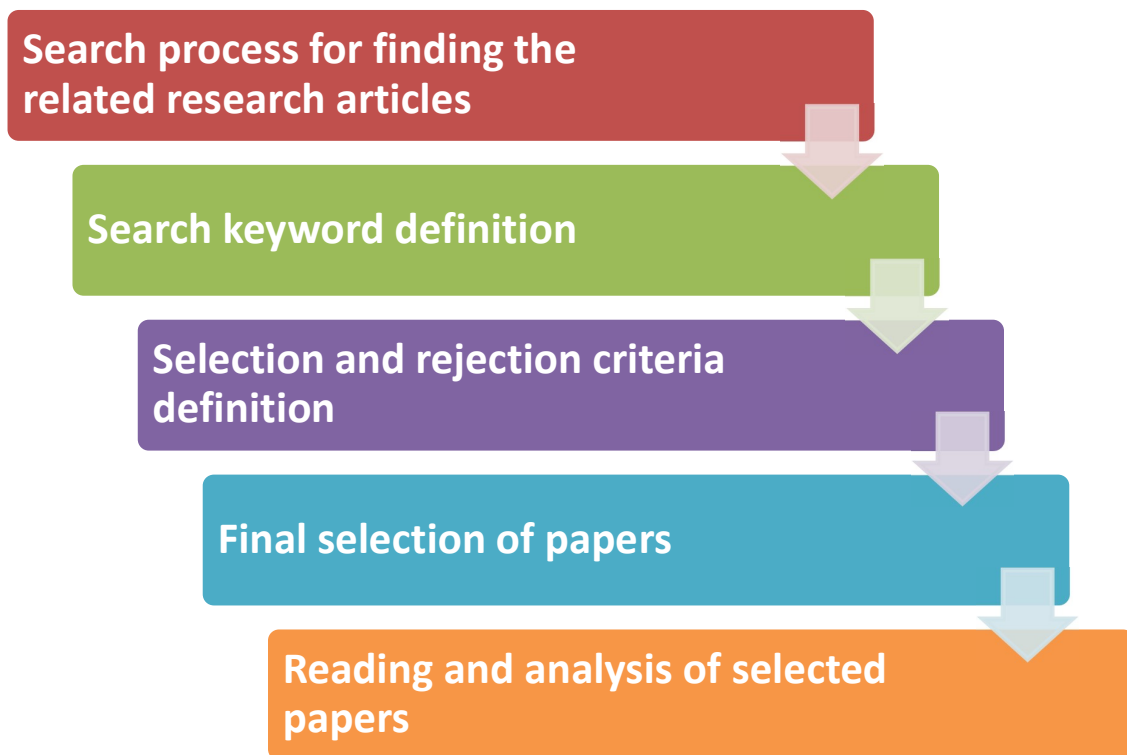


Figure 2.1: Literature Review Process

To conduct effective research a strong knowledge base is required. To make a strong knowledge base we have reviewed state of the art more than 80 research papers only most related and latest papers are included references section. More than 90 percent of paper we selected for the literature review are published in last 15 Years. To perform a quality literature review we only included paper from best ranked online databases like the Springer, Science Direct or Elsevier and only included those conference papers which are ranked and index by Scopus.



Figure 2.2: Research Papers Search Process

2.2. Factors of Software Testing Defect

There are various factors influencing on the quality of the software testing, like line of code and test cases. However, the degree to which they utilize their influence on the effectiveness is debatable. Line of code and test cases have been the widely researched topics in the factors influencing the effectiveness. There still could be many other factors, which have an impact on it. This section discusses the influential factors in further detail.

Wong et al. [14] reported from their experiments that the relationship between line of code and test cases was higher than the relationship between efficiency and size. They also mentioned that there was no reduction in the fault detection efficiency of a test set when the size of the test set was reduced, and the line of code kept constant. This result of their experiment highlighted that line of code was a more important factor than size in determining the efficiency of a test set. Furthermore, Wong et al. [15] also concluded from that the test cases that did not add to the line of code of test set were not likely to be efficient in detecting faults. Adding test cases could contribute to the size of the test set, but the defect detection efficiency of the test set remained unchanged.

Work by Rothermal et al [3] stated that there could be other factors apart from line of code and test cases., which could also have an impact on the efficiency of the test suites. A novel idea to measure efficiency was introduced by Cai and Lyu [16]. They measured efficiency between the variables of execution time, line of code and test cases. Their main approach was that defect detection was related to both the time the software experiences

under testing, which was called the execution time and the fraction of code that has been covered by testing, called as the test case. Measuring reliability by including the execution time along with line of code proved to be more accurate than a single measure of only according to their results.

Chen et al. [17] defined that a testing effort can be termed as effective if and only if it either increases any coverage criteria or reveals the presence of a fault. They observed that the reliability of a program would increase only if the number of defects were being reduced by the addition of new test cases. Redundant test cases should be avoided, as they do not contribute to the testing efficiency, even if they improve the test set size. Another disadvantage of having a redundant test case was that it increased the testing effort, i.e., the time spent to execute the test case. Chen et al. also emphasized that a test case can be considered effective only if it is executing some uncovered part of the program and/or the test case causes some defects to be triggered; otherwise, the test case is considered non-effective. Other than size and line of code, a new measure of execution time can also have an impact on the efficiency of the software testing.

2.3. Code Size Role in Defect Prediction

The role of code size on the effectiveness of an analysis has been a widely discussed topic. Many researchers have studied the impact of increasing line of code on the efficiency and presented their solutions. This section looks at them in detail.

Andrews et al. determined that test of a program were related to real defects that occur in a program. It was suggested that using test case to measure line of code adequacy allows us to find more statistically significant properties of the study. Furthermore, used test cases to assess and compare the test line of code criteria and found that test cases can be used to predict the presence of defects. The results obtained stated that usage of test cases to detect defects was like what would have been obtained with actual defects [20].

Taghi & Kehan et al. indicated that size and line of code are important for assessing test cases and there exists a non-linear relationship between these three variables of size, line of code, and defect numbers. Apart from line of code, that has been widely researched, size is also important factor for improving the efficiency. They concluded that both size and line of code independently influence the efficiency of a testing, and a linear relationship exists between logarithm of size and line of code [21].

Work by Ro thermal et al. hypothesized and concluded that line of code is an

important factor in analyzing analysis efficiency. Results obtained from experiments by Garg [4] also show a strong connection between software reliability and line of code, under any criteria. Software reliability here can be defined as the measure of successful functioning of a software system, in a particular environment at a given time. Results from the experiment also showed that line of code seemed to relate more to reliability than the number of faults that were found in the program [3].

Subbarao [5] also opined that though the efficiency of using line of code is debatable, the code data gives an important insight on the efficiency of the tests. Subbarao noted that code shows what part of the source code is thoroughly executed. Subbarao also stated that measuring line of code and ensuring the gradual increase of code in a project would lead to development of software that would be probably free of severe bugs. Tracking coverage has ensured high quality software to be developed.

Cai and Lyu et al. reported that code is a reasonable indicator for the capability of defect detection on a normal test set. They also noted that the effect of line of code on defect detection varied based on the test set. And the co-relation between line of code and efficiency is high for exceptional test cases and weak for normal test cases. Another interesting result of their work was that the relationship between code and defect was higher in case of structural testing than random testing. Though not conclusive, the result moves in a direction that functional test cases are more effective than random test cases in determining the defect detection efficiency of a test set [6].

Hutchins et al. performed experiments to analyze whether defect detection increases when the line of code levels of test sets is increased. Measurement of defect detection effectiveness was done by examining the number of faulty versions of the subject programs that were detected by the test cases. After evaluating both data flow and control flow line of code criteria, it was concluded that both are effective adequacy measures. Adding test cases to improve code size proved to be beneficial [2].

Similar studies were conducted by Frankl and Weiss [1] to understand the relationship between code size and test cases. It was concluded that “error - exposing” ability of test cases showed an increase as more test elements were covered, but the dependence in general was non-linear. Frankl and Iakounenko also stated that for the subject programs with naturally occurring faults, the likelihood of finding the presence of a defect increased with increasing code size levels. So, it was hypothesized that better code size of an analysis led to

better defect-finding efficiency [7].

2.4. Impact of Code Size on Defect Detection

The impact of code size on the detection of defects has also been of interest. There is still an on-going debate on whether increasing code size helps in detecting more defects. This section describes the various research works discussing code size and defect detection.

Though Lyu et al. report of no strong correlation between the total number of defects detected in a program and the code size measure of the program, it was hypothesized that the more thorough a program is covered during testing, the greater number of defects it can identify [8].

Kramer et al. reasoned that many bugs would still be undetected, even if “complete” code size were achieved, asserting that “completeness” is measured only with respect to a given population of tests. It was noted that good testing involved trade-offs based on thoughtful judgment, rather than just trying to achieve “complete” coverage. Instead of improving code size to improve efficiency of a test case, Kramer debated that testing must be prioritized in such a way that the test strategy that is most likely to find bugs must be selected and implemented on the software program [9].

Smith and Williams et al. also asserted through their experimental results that though code size is being used widely by researchers and developers, and sometimes even as a stopping criterion for unit testing, the confirmation surrounding the use of code size as a software reliability predictor is not conclusive [10].

Gomes et al. reported extra number of resources would have to be invested to achieve 100% test case. Gomes also added that the earlier a bug is found during testing, the cheaper it is to fix it and code size is one of the best ways to detect defect in the early stage. However, Gomes also argued that every test created is also a test that will eventually require maintenance, and therefore suggested that it is important to choose what to test wisely. Knowing what goals need to be achieved by accomplishing high-test code size is very important, and Gomes suggested that if the goals could be achieved by a lesser expensive way, it should be the best way from the business perspective [11].

Ruiz et al. also reported an experimental result in which defect software achieved 100% code size which directed towards an important conclusion that high code size does not automatically reflect a healthy code. Only by functional testing of the application, it can be ensured that the application works as a whole and performs correctly [12].

Glover et al. stated that a high code size percentage alone does not ensure the quality

of the code. Code that has high impact, meant that a lot of code was exercised, though it did not imply that the code had been exercised well. Highly covered code is not defect free, though it was less likely to consist of defects. Glover also opined that test cases tools are important and stressed that they should be used for evaluation of the quality of code and for assessing functional testing [13].

2.5. Tools & Techniques for Defect Prediction

Zhongbin et al. proposed a method to predict the defect in testing data. They have used NASA testing data and applied some classification techniques. First, they have converted the imbalanced class data into multiclass labels data and provided balance data to the defect prediction model. They have implemented four classification algorithms and six conventional methods; they have found classification algorithms provided the best results as compared to conventional methods [18].

Huanjing et al. focused on the feature extraction and ranking methods for defect prediction. They have determined important features from the collected data based on different feature ranking methods. Performed classification models on 16 different defect prediction datasets for finding the results of the best one approach and dataset. Finally, they found that ensemble techniques for ranking algorithms and classification algorithms shown effective results [19].

Shuo & Xin et al. collected different imbalanced defect prediction datasets and try to apply different ensemble techniques to balance the data. Also investigated the imbalanced learning techniques such as Adaboost, G-mean, and others. The adaBoost algorithm shows the best result with different training data and ensemble techniques. The authors claimed that the selected imbalanced dataset shows 32.29% accurate defect prediction results with ensemble class imbalance techniques [20].

Taghi & Kehan et al. identified feature selection while using two different methods individual & repetitive feature selection. All the feature selection techniques are implemented with six different ranking methods or algorithms including boosting, voting, and AdaBoost. They have concluded that the repetitive feature selection technique with AdaBoost provided effective and better accurate results [21].

Marc & Johan et al. performed the "defect prediction" while using a "support vector machine" and they have used an ensemble "support vector machine" in the implementation. Binary "Support Vector Machine" and ensemble support vector machine-implemented two

classifiers for compared the results. They have used Ijenn1 testing data for predicting defects based on the ensemble SVM model [22].

Issam & lahouari et al. implemented feature extraction and ensemble techniques for predicting defects in the NASA testing data. They have found approximately 1.0 AUC scores with the help of a weighted support vector machine classifier. The greedy forward selection approach shows the best features as compared to the other approaches [23].

Table 2.1: Identification of Defect Prediction Techniques & Datasets

Authors Names	Paper Reference	Publication Year	Testing Data	Techniques/Methods
Zhongbin, Xiaoyan	[18]	2012	NASA	Multiclass Code-based ensemble Learning
Huanjing, Taghi	[19]	2012	LLTS Dataset	Ensemble Learning
Shuo & Xin	[20]	2013	Mc2	AdaBoost Ensemble learning
Taghi & Kehan	[21]	2014	Jml	RUSBoost & AdaBoost classifiers
Marc & Johan	[22]	2014	Ijenn1	SVM ensemble method
Issam & Lahouari	[23]	2015	NASA	W-SVM Model
Fernando & Wikan	[24]	2019	NASA	Naïve Bayes & Association Rule Mining

2.6. Other Related Studies

To conduct effective research a strong knowledge base is required. To make a strong knowledge base we have reviewed state of the art more than 100 research papers only most related and latest papers are included references section. We choose a well-defined selection and rejection criterion mentioned by Kitchen Ham in [33].

More than 90 percent of paper we selected for the literature review are published after

2015. To perform a quality literature review we only included paper from best ranked online databases like the Springer, Science Direct or Elsevier and only included those conference papers which are ranked and index by Scopus. A study was conducted in 2015 about the fault prediction of different kind of software systems using the expert systems and the data mining applications [34].

Similarly [35] conducted another research and presented the “HySOM” a software fault detection and prediction model using the supervised learning techniques and also applied the same techniques on the data set of NASA software systems. Correlation based model is derived for both functional size of the software and the effort required for the software testing process [38].

Analysis of test coverage is performed using Bisayan and correlation-based data engineering techniques and applied them on the case study for finding the relationship between the amount of the testing data and the effort required to find out the defects from such a large amount of code [39].

The applications of the data engineering and process mining for finding out the defects in different software processes using the different data mining-based software testing techniques is shown in the [40].

In this paper we have applied the correlation and Apriori based association rule mining technique to find out the association between the line of codes and the defects at a data set which is contains the software testing data after applying different testing techniques. Similar kind of the work is performed by the [41] in which he also applied the correlation and association rule mining to find out the associations in different aspects using the software testing data set.

Data mining is domain for mining the knowledge form huge amount of data using different association rules here in this paper author used Apriori based association mining technique to mine relationships and associations in testing data variables. Different degrees of level of confidence and level of support can be used to figure out the relationships and association rules between different variable of testing data [41].

Yang et al. in [42] talks about the effort estimation in all stages of the "software development life cycle (SDLC)" according to the author the testing phase requires 21.57% of effort which also requires large amount of time but use of the data mining approaches this effort can be reduced significantly. As the process of maintenance is started the rate of modification in the software is also increased significantly so it is very necessary to conduct the regression testing.

Retest a system using Regression testing also increase the cost and time to maintain or develop the system. To make debugging easier prioritize test case in regression testing. Authors discuss to prioritize and rank the test cases using Association Rules Mining. In this process we save the whole history of faults which occurred during any phase of the development. It helps to keep record which component of system contain a greater number of faults. When we change something in system, we test these parts of system in details. Using this approach, we prioritize the "test cases" with respect to "Average Percentage of defect Detection (APFD)" [43].

2.7. NASA Available Datasets

We have identified different datasets available online sources and published by NASA data bank.

Table 2.2: Identification of NASA Defect Prediction Datasets

Testing Data	Observations	Parameters	Defect%
Mc2	161	39	32.29
Kc2	522	21	20.49
Jm1	10885	21	19.35
Kc1	2109	21	15.45
Pc4	1458	37	12.20
Pc3	1563	37	10.23
Cm1	498	21	9.83
Kc3	458	39	9.38
Mw1	403	37	7.69
Pc1	1109	21	6.94

Chapter 3

Proposed Methodology

CHAPTER 3: PROPOSED METHODOLOGY

Overall research study based on the proposed methodology. This chapter describes the main components of the research methodology. We have proposed our research methodology while following Correlation Analysis & Association Rule Mining technique.

3.1. Proposed Methodology

For proposing a detailed methodology, first, need to collect data from reliable sources. After data collection requires important features for implementing the model. Model results and analysis are also part of the methodology which will show the importance of implemented model for defect prediction.

Our Methodology consists of following Steps.

- Testing Data Collection
- Selective Data Extraction
- Data Pre-processing
- Model Creation and Implementation
- Results and Feedback

All of the above steps are the main components of our methodology. Figure 3.1 shows the visual form of our proposed methodology. Further, we have discussed each step or component in detail.

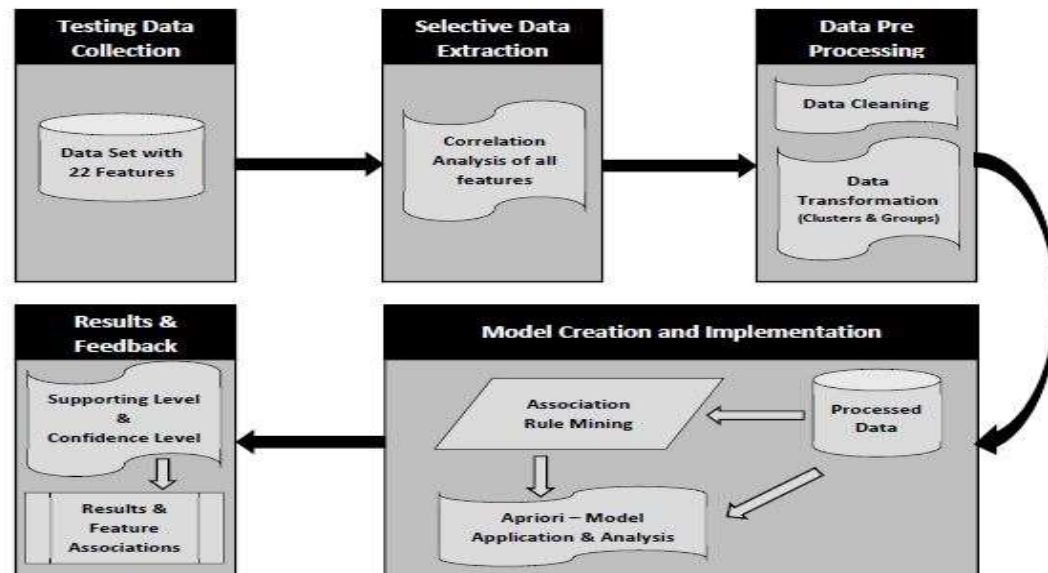


Figure 3.1: Proposed Approach

3.2. Testing Data Collection

We have explored the different available datasets which can use for defect prediction in software testing. After completed the analysis and exploration, we have found a reliable testing dataset that is suitable and related to the research study requirements. This testing dataset collected from the "NASA Data bank ". We have collected three dataset which are all related to the software testing data. Table 3.1 shows the complete parameters of all three-testing dataset.

Table 3.1: Testing Data Parameters

Parameters		
<ul style="list-style-type: none"> loc v(g) ev(g) iv(g) n v l d 	<ul style="list-style-type: none"> i e b t IOComment Codesize (loc) IOBlank Testcasenumber 	<ul style="list-style-type: none"> locCodeAndComment Uni_qop Total_op Total_opnd branch_Count defects defectnumber

Figure 3.2 represents the testing dataset 1 and there is a total of 10885 records and 22 parameters. All the parameters will use in the Correlation Analysis.

H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
1.3	1.3	1.3	1.3	1.3	1.3	1.3	2	2	86233	2	1717	2	1.2	1.2	1.2	1.2	1.4	b'false'	187
1	1	1	1	1	1	1	1	1	57413	1	2976	1	1	1	1	1	1	b'true'	304
0.05	20.31	55.85	23029.1	0.38	1279.39	51	10		94745	8	2844	1	17	96	112	86	13	b'true'	358
0.06	17.06	254.87	74202.67	1.45	4122.37	129	29		102821	28	3855	2	17	135	329	271	5	b'true'	350
0.06	17.19	34.86	10297.3	0.2	572.07	28	1		101797	6	3029	0	11	16	76	50	7	b'true'	331
0.08	12.25	47.55	7135.87	0.19	396.44	19	0		18153	5	1373	0	14	24	69	42	3	b'true'	232
0	0	0	0	0	0	0	0		5707	0	1555	0	0	0	0	0	17	b'true'	138
0.36	2.8	18.11	142.01	0.02	7.89	5	0		10006	1	1605	0	4	5	9	7	1	b'true'	113
0	0	0	0	0	0	0	0		130358	0	5647	0	0	0	0	0	3	b'true'	650
0	0	0	0	0	0	0	0		82357	0	4524	0	0	0	0	0	43	b'true'	463
0	0	0	0	0	0	0	0		105533	0	423	0	0	0	0	0	19	b'true'	76
0	0	0	0	0	0	0	0		70617	0	1966	0	0	0	0	0	21	b'true'	331
0.15	6.87	24.34	1150.68	0.06	63.93	8	0		21904	2	2530	0	11	12	22	15	5	b'true'	211
0.06	17.35	40.1	12067.3	0.23	670.41	29	1		160606	16	4565	0	19	23	87	42	7	b'true'	549
0	0	0	0	0	0	0	0		106913	0	6064	0	0	0	0	0	15	b'true'	633
0	0	0	0	0	0	0	0		31930	0	1354	0	0	0	0	0	43	b'true'	60
0.5	2	13.5	54	0.01	3	2	0		9684	6	116	0	4	4	5	4	1	b'true'	50
0.01	97	99.72	938311.1	3.22	52128.39	139	92		74496	17	5127	0	32	64	1081	388	97	b'true'	556
0.02	40.95	28.35	47536.38	0.39	2640.91	59	0		38483	16	354	0	7	10	167	117	15	b'true'	50
0.03	32.64	52.53	55961.02	0.57	3108.95	69	0		11166	14	2139	0	26	47	161	118	13	b'true'	150
0.03	33.41	61.94	69127.22	0.69	3840.4	81	13		30067	14	821	0	27	59	176	146	33	b'true'	103
0.04	25.33	36.63	23506.58	0.31	1305.92	34	0		41978	13	626	0	19	24	107	64	11	b'true'	59

Figure 3.2: Testing Dataset 1

Figure 3.3 represents the testing dataset 2 and there is a total of 497 records and 22 parameters. All the collected parameters will participate in the Correlation Analysis, and we will find out the most interlinked parameters.

K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
e	b	t	size(KB)	commenti	code_size(LOC)	IOBlank	test_case_num	locCodeA	uniqOp	uniqOpnd	totalOp	totalOpnc	branchCo	defects	number_of_de	Test Case Dens	Defect Dens
1.3	1.3	1.3	2	2	60030	2	1833	2	1.2	1.2	1.2	1.2	1.4	b'false'	249	3.053473263	4.147926
1	1	1	1	1	84583	1	2607	1	1	1	1	1	1	b'true'	318	3.082179634	3.7596207
2936.77	0.1	163.15	1	0	173234	6	7058	0	15	15	44	19	9	b'false'	886	4.07425794	5.1144694
3447.89	0.07	191.55	0	0	18493	3	3502	0	16	8	31	16	7	b'false'	407	18.93689504	22.008327
5999.58	0.12	333.31	0	0	98477	3	4615	0	16	12	46	26	11	b'false'	590	4.686373468	5.9912467
5999.58	0.12	333.31	0	0	24989	3	4447	0	16	12	46	26	11	b'false'	370	17.79583017	14.806515
69.74	0.01	3.87	0	0	55957	1	1358	0	4	5	6	5	1	b'false'	233	2.426863484	4.1639116
604.36	0.03	33.58	0	0	34426	7	564	0	10	7	14	9	3	b'false'	70	1.638296636	2.033469
7820.87	0.18	434.49	12	16	96332	13	7389	0	15	20	69	38	9	b'false'	752	7.670348379	7.8063364
30377.95	0.45	1687.66	8	35	11000	22	281	0	15	37	129	110	29	b'false'	51	2.554545455	4.6363636
17773.08	0.29	987.39	11	28	93282	16	3943	0	19	27	96	59	9	b'false'	547	4.226967689	5.8639395
1287.55	0.05	71.53	2	4	67191	4	922	0	9	8	19	16	3	b'false'	162	1.372207587	2.4110372
21659.58	0.26	1203.31	10	17	67823	23	2285	0	17	13	114	43	13	b'false'	288	3.369063592	4.2463471
12929.85	0.16	718.32	10	17	40171	23	1553	0	14	9	70	35	13	b'false'	62	3.865972966	1.543402
35852.6	0.43	1991.81	2	15	37001	40	3905	0	22	28	161	70	9	b'false'	293	10.55376882	7.9187049
9958	0.22	553.22	3	20	66103	23	355	0	19	25	80	40	9	b'false'	50	0.537040679	0.7563953
2480.95	0.09	137.83	6	5	23778	8	4168	0	14	13	40	17	3	b'false'	514	17.52880814	21.61662
222.03	0.02	12.33	1	1	27618	2	496	0	8	5	10	5	1	b'false'	50	1.79593019	1.8104135
8684.99	0.25	482.5	2	0	97162	19	4724	0	12	34	69	66	3	b'false'	635	4.86198308	6.5354768
222.03	0.02	12.33	1	1	101622	0	6046	0	8	5	10	5	1	b'false'	567	5.949499124	5.5795005
1265.83	0.04	70.32	0	0	59081	1	330	0	10	5	15	12	3	b'false'	102	0.558555204	1.7264434
211.89	0.02	11.77	0	0	67032	0	3719	0	7	4	9	5	1	b'false'	347	5.548096432	5.1766321

Figure 3.3: Testing Dataset 2

Figure 3.4 represents the testing dataset 3 and there is a total of 2108 records and 22 parameters. Again, we will find the most important and interlinked variables or parameters. All of the interlinked parameters will use for the implementation of the Apriori algorithm.

K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
e	b	t	size(KB)	commenti	code_size(LOC)	IOBlank	test_case_num	locCodeA	uniqOp	uniqOpnd	totalOp	totalOpnc	branchCo	defects	number_of_de	Test Case Dens	Defect Dens
1.3	1.3	1.3	2	2	60030	2	1833	2	1.2	1.2	1.2	1.2	1.4	b'false'	249	3.053473263	4.147926
1	1	1	1	1	84583	1	2607	1	1	1	1	1	1	b'true'	318	3.082179634	3.7596207
2936.77	0.1	163.15	1	0	173234	6	7058	0	15	15	44	19	9	b'false'	886	4.07425794	5.1144694
3447.89	0.07	191.55	0	0	18493	3	3502	0	16	8	31	16	7	b'false'	407	18.93689504	22.008327
5999.58	0.12	333.31	0	0	98477	3	4615	0	16	12	46	26	11	b'false'	590	4.686373468	5.9912467
5999.58	0.12	333.31	0	0	24989	3	4447	0	16	12	46	26	11	b'false'	370	17.79583017	14.806515
69.74	0.01	3.87	0	0	55957	1	1358	0	4	5	6	5	1	b'false'	233	2.426863484	4.1639116
604.36	0.03	33.58	0	0	34426	7	564	0	10	7	14	9	3	b'false'	70	1.638296636	2.033469
7820.87	0.18	434.49	12	16	96332	13	7389	0	15	20	69	38	9	b'false'	752	7.670348379	7.8063364
30377.95	0.45	1687.66	8	35	11000	22	281	0	15	37	129	110	29	b'false'	51	2.554545455	4.6363636
17773.08	0.29	987.39	11	28	93282	16	3943	0	19	27	96	59	9	b'false'	547	4.226967689	5.8639395
1287.55	0.05	71.53	2	4	67191	4	922	0	9	8	19	16	3	b'false'	162	1.372207587	2.4110372
21659.58	0.26	1203.31	10	17	67823	23	2285	0	17	13	114	43	13	b'false'	288	3.369063592	4.2463471
12929.85	0.16	718.32	10	17	40171	23	1553	0	14	9	70	35	13	b'false'	62	3.865972966	1.543402
35852.6	0.43	1991.81	2	15	37001	40	3905	0	22	28	161	70	9	b'false'	293	10.55376882	7.9187049
9958	0.22	553.22	3	20	66103	23	355	0	19	25	80	40	9	b'false'	50	0.537040679	0.7563953
2480.95	0.09	137.83	6	5	23778	8	4168	0	14	13	40	17	3	b'false'	514	17.52880814	21.61662
222.03	0.02	12.33	1	1	27618	2	496	0	8	5	10	5	1	b'false'	50	1.79593019	1.8104135
8684.99	0.25	482.5	2	0	97162	19	4724	0	12	34	69	66	3	b'false'	635	4.86198308	6.5354768
222.03	0.02	12.33	1	1	101622	0	6046	0	8	5	10	5	1	b'false'	567	5.949499124	5.5795005
1265.83	0.04	70.32	0	0	59081	1	330	0	10	5	15	12	3	b'false'	102	0.558555204	1.7264434
211.89	0.02	11.77	0	0	67032	0	3719	0	7	4	9	5	1	b'false'	347	5.548096432	5.1766321

Figure 3.4: Testing Dataset 3

3.3. Selective Data Extraction

Next step in the proposed methodology is selective data extraction. With the help of this component, we can get most interlinked parameters from the testing datasets. There are 24 parameters in the original dataset, and we need to know relationship or correlation between different parameters. For this purpose, we will implement the correlation analysis using Spearman Rank Correlation technique.

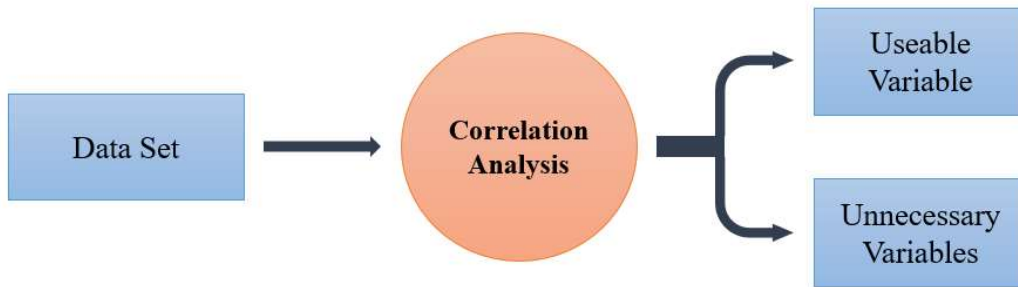


Figure 3.5: Correlation Analysis Process

3.3.1. Spearman Rank Correlation

"Spearman Rank Correlation" is a simple correlation analysis that shows whether there is any relationship between two variables. Pearson correlation co-efficient is "non-parametric test" and mostly used for finding the relation between parameters or interlinked parameters. It obtained by finding the difference between paired ranks. Based on the co-efficient value obtained, the relationship between the two variables can be described. If the co-efficient value obtained is 0, then the variables are independent of each other. If the co-efficient value obtained is 1, then the variables have the presence of a strong relationship between them. We will implement the Pearson correlation analysis between the variables. Figure 3.5 represents the formula for "Spearman Rank Correlation" analysis.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

ρ = Spearman's rank correlation coefficient

d_i = difference between the two ranks of each observation

n = number of observations

Figure 3.6: "Spearman Rank Correlation" formula

3.4. Data Preprocessing

Data preprocessing is the primary part of the data analysis or model implementation. Our collected testing data contains the missing values and irrelevant data so need to clean the data first. For this purpose, we applied some data preprocessing techniques such as remove missing data and transforms data in the suitable format. Data mining models specially

"Apriori algorithm can't deal with continuous numerical value". So, need to apply clustering algorithm to deal with the parameter's discretization.

3.4.1 Handle Missing Data

For handling missing values in the testing data, we have performed some data preprocessing or cleaning techniques such as replace the values with average data or remove the values. So, we have replaced the missing values with the average of that parameter. All the data contains continuous values, so it is easy to replace with the average of the data. In this way, there are not any missing or null values in the testing data and it's ready for further analysis.

3.4.2 Parameters Scaling

Parameter's scaling is one of the data preprocessing techniques which helps to found most suitable and important parameters. We applied parameters scaling to discover the most related data and introduced two new variables or parameters while using most interlinked parameters. We will discuss these related and introduced parameters in the implementation chapter. Formulas which we used for introducing two new Parameters are:

$$\text{Test Case Density Per 100 line} = 100 * \left(\frac{\text{TestCaseDesigned}}{\text{SoftwareSize}} \right)$$

$$\text{Defect Density Per 1000 line} = 1000 * \left(\frac{\text{NumberOfDefects}}{\text{SoftwareSize}} \right)$$

3.4.3 Parameters Discretization

Processed data contains continuous numerical values, and we need to implement the Apriori algorithm under Association Rule Mining. But there is a problem with the continuous data. "Apriori algorithm can't deal with a continuous numerical value". So, we need to apply a clustering algorithm to deal with the parameter's discretization. With the help of the clustering algorithm, we can generate groups of continuous values and assign them a specific label. Labels can easily use in the implementation of the Apriori algorithm.

3.5. Model Creation and Implementation

Next step in the methodology is model creation and implementation. For this purpose, we have already cleaned and transformed data into suitable format. Further, developed a data mining framework that provides us correlation analysis and defect detection for testing data. In the Model creation and implementation process, we have utilized testing processed

datasets. Predicted the defect which repeated in data. Detecting the defects which repeated in testing data, and all these defect combinations analyzed from the collected testing data.

For analyzing the association between defect, we have used "Market Basket Analysis with the help of Association Rule Mining & Apriori algorithm". Using "Market basket analysis", we have tried to predict defect in testing data. It will decrease cost, time and manpower for predicting defect in the software testing.

3.5.1 Market Basket Analysis (MBA)

"Market Basket Analysis" uses a pattern of collected data. "This analysis is based on the if-else or if-then condition". Suppose software 'A' has 'T' defect then most likely there will be 'D' defect in the software testing. "Market basket analysis" is a main approach in which we can implement association rule mining with the help of Apriori algorithm".

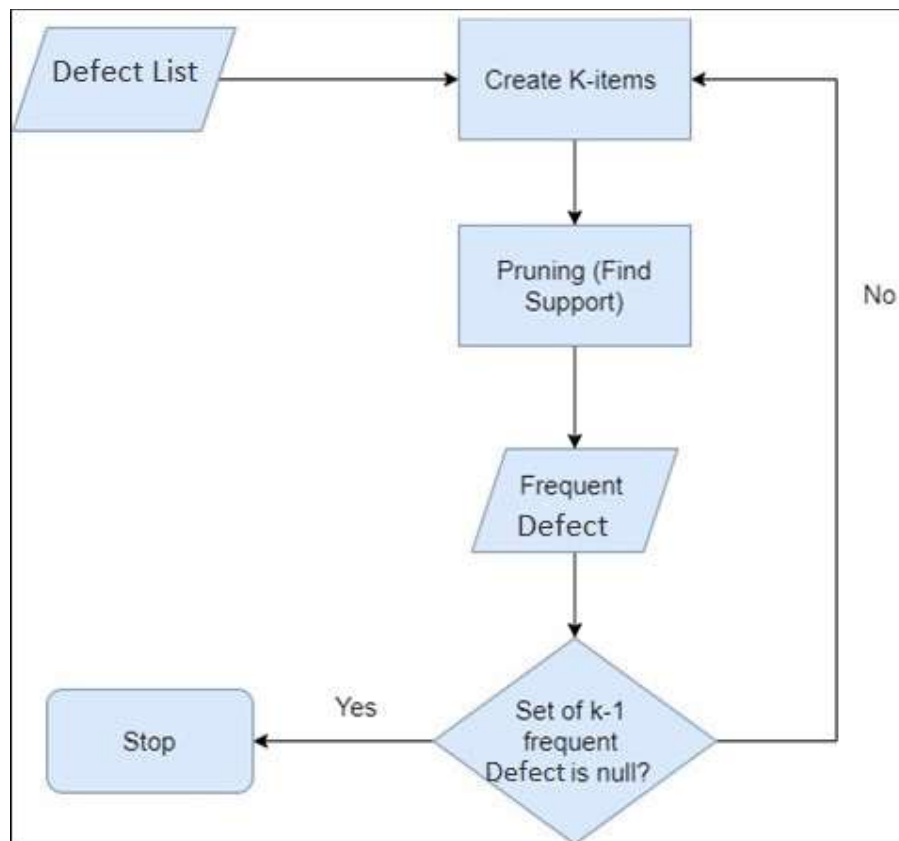


Figure 3.7: "Market Basket Analysis" flow diagram

3.5.2 Association Rule Mining Algorithm

Apriori based Association calculation algorithm is also a well-known branch of the data engineering process sometimes it is also termed as "Market Basket Analysis". It is used to find the relationship between various entities and items related to the supermarkets and economic systems. In terms of data mining, we mostly use it to find out the relationships and the associations between different items which are hard to find manually.

Association rules general formula Level of Support: Level of support is defined as real time occurrence of X and Y at a same time. $\text{Support}(X \rightarrow Y) = P(X \cup Y)$ Level of Confidence: The Likelihood of occurrence of item set X with likelihood of occurrence of item set Y $\text{Confidence}(X \rightarrow Y) = P(X|Y)$

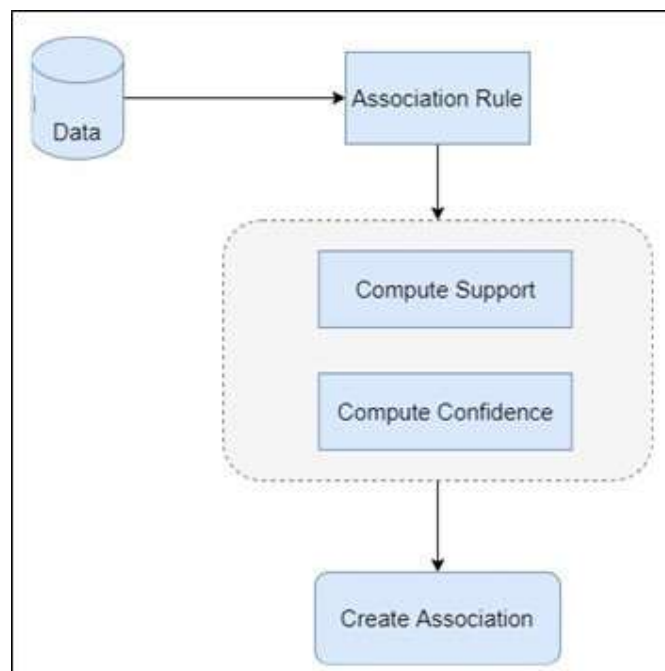


Figure 3.8: "Association Rule Mining"

Minimal level of support and minimal level of confidence A value defined the data engineer based on his experience is known as the "Threshold Value" this value is used to change the level of support. Some time it is also known as the "Minimum level of support" While on the other hand another type of the threshold value which is used to change the level of the confidence is known as the "Minimum level of confidence" It also helps us in figuring out the variables with lowest possible relationship or association. But any association rule will be strong most if both threshold of the level of support and the threshold of the confidence are fulfilled.

Set of items is combination of total items is known as X set of items if it owns X no.

of items. e.g. {xyz, abc, efg} is a set of 3 items Transaction amount of any set of items is known as occurrence incidence it's also known as level of supporting.

3.6. Apriori Framework Evaluation

When our proposed framework will implement, we need to evaluate the results of the framework. For this purpose, we have used defect rules results for evaluation and find out the best rules based on the support level and confidence.

The result will generate the combinations of the rules or defect rules which occurred together in multiple testing data based on the minimum threshold values. We will set different support, confidence, and lift values and evaluate the model.

Suppose defect Y and D occurred in multiple testing data, then maybe T also occurred with D & Y in another association rule. So, we can generate multiple defect rules with the help of an implemented model.

3.6.1 Expected Outcomes

Expected outcomes are the rules which we want to generate from the implemented model. If we have T & D type defects in our testing data, then we can expect output based on the occurrence of defects together. In table 3.2, we have designed a pattern of defects that occurred, and we have designed basket 5 which represents the question mark for the expected output. Now it depends on the performance of the model.

Table 3.2: Expected Outcomes

Basket	Defect1	Defect2	Outcomes
1	T1	D2	T3
2	T2	D1	D4
3	T3	T1	D2
4	T4	D4	T2
5	T5	D5	? [D1 is expected]

In the above table, we can see a combination of the defect which occurred together. Based on the occurrence of these defect combinations, we can expect the output for the 5 number series. If our implemented model provides our results according to our expectations then we can say, the model provided accurate results otherwise model results will not be acceptable.

3.6.2 Evaluate Support & Confidence

"Apriori algorithm" based model can evaluate with the help of support level, confidence, and lift. We have different formulas for calculating these values. But in starting the model, we need to assign minimum support, confidence, and lift values. Association Rules are generated based on the minimum values of these formulas. We can find support, confidence, and lift as:

$$Support = \frac{freq(X, Y)}{N}$$

$$Confidence = \frac{freq(X, Y)}{freq(X)}$$

$$Lift = \frac{Support}{Supp(X) \times Supp(Y)}$$

3.7. Summary

This chapter discussed the detailed methodology of the framework or model which we have proposed for conducting our research study. We have collected data from the NASA data bank and this dataset can be used for testing defect prediction. Further, we'll use Spearman Rank Correlation Analysis for finding the most important or interlinked parameters from testing data. Also, we need to introduce two new parameters which can use for defect prediction in the Apriori algorithm. We'll use the discretization technique to handle the continuous data and assigned the data group a specific label. In the end, we'll apply the Apriori algorithm on the collected labels and find out the association between defects.

Chapter 4

Model Implementation

CHAPTER 4: MODEL IMPLEMENTATION

Model Implementation follows the methodology of this research and we have discussed all the implementation process in detail in this chapter. Based on the methodology, we have collected testing data first and then processed the data according to our model implementation requirements. We have implemented the Apriori algorithm with the help of Association Rule Mining, so we need to process data that can use as an input for the model. Figure 4.1 shows the complete process of the implementation, and we can see in this figure there is correlation analysis after data collection. Correlation Analysis provided us very important parameters from collected data that are interlinked and have a positive strong relationship between them.

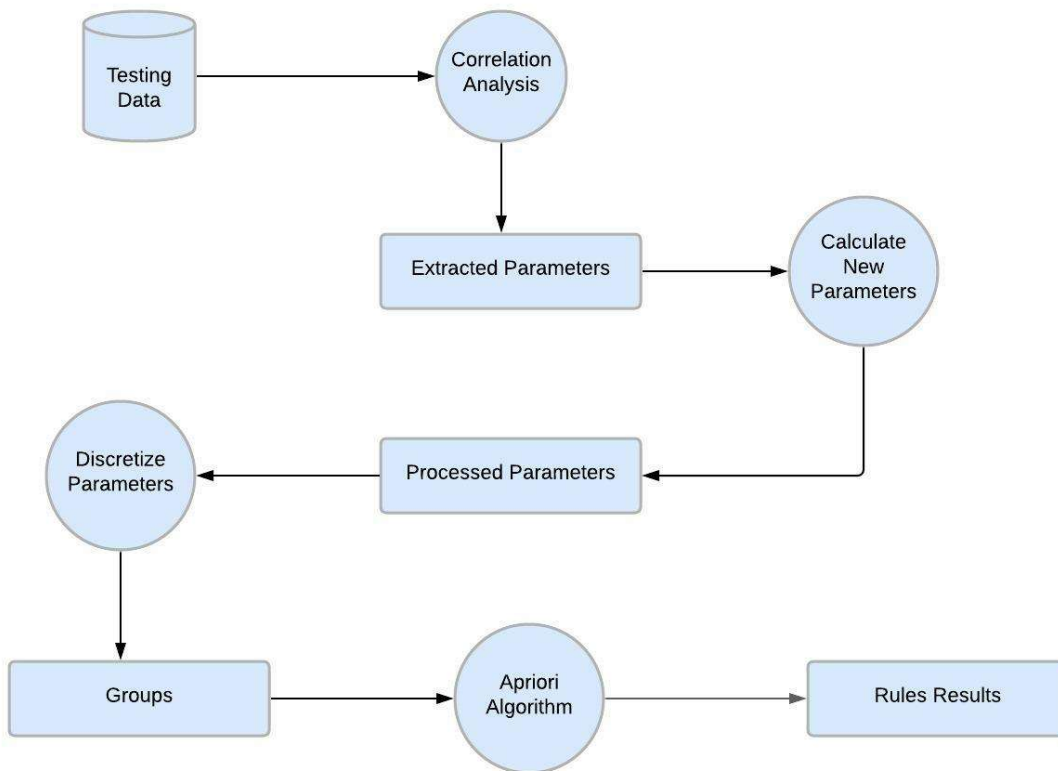


Figure 4.1: Implementation process of Model

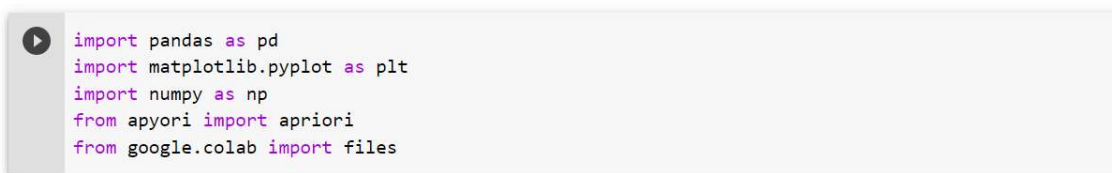
After finding important parameters from the collected testing data, we need to find any other variables which can be calculated from extracted parameters. So, we introduced two new parameters while using different formulas which we have discussed in the methodology chapter. After processing the data, we have found five parameters that can use for the model implementation. All of these parameters include the continuous values, and the

Apriori algorithm can't deal with the continuous values. So, we have converted these continuous values in the form of groups and assigned them specific labels with the help of the clustering (grouping) technique. Further applied Apriori algorithm on the preprocessed & transformed data and generated the defect rules based on the support and confidence values.

4.1. Python Required Packages

We have implemented our proposed framework in python language, and we need to import some packages which are required for the implementation. Figure 4.2 shows the screen shot of the python code and we have imported pandas, matplotlib, numpy, apriori, and google.colab packages.

▾ Importing Necessary libraries



```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from apyori import apriori
from google.colab import files
```

Figure 4.2: Required Packages

Panda's package is required for reading datasets and dealing with data frames. It also can use for data manipulation and transformation. It means it participated in the data preprocessing. Import NumPy package which can handle mathematics calculations, array, and matrix interpretation. Matplotlib package required for the result visualization, we have plotted correlation bar & stem plots with the help of this package. Further, import the apriori package under apyori library and is used for the association rule mining. Google.colab library used for importing files from google and we can also attach file from the local system.

4.2. Import Testing Data

The next step of the implementation is importing testing data and we have three datasets in the testing data. So, imported all the datasets while using google.colab and pandas packages. Figure 4.3 shows the implementation of importing testing data in the google colab environment. There are 24 parameters in the dataset, and we need to explore the important parameters from the data. So, further, we have implemented correlation analysis for dealing with all parameters.

```
[ ] from google.colab import files
    uploaded = files.upload()

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session.
Saving cm1_new.csv to cm1_new.csv
```

Printing Parameters in Dataset

```
[ ] data = pd.read_csv('./cm1_new.csv')
    print( 'Available columns are : ', data.columns )

Available columns are : Index(['Unnamed: 0', 'loc', 'v(g)', 'ev(g)', 'iv(g)', 'n', 'v', 'l', 'd', 'i',
    'e', 'b', 't', 'size(KB)', 'commentRate', 'code_size(LOC)', 'lOBlank',
    'test_case_number', 'locCodeAndComment', 'uniqOp', 'uniqOpnd',
    'totalOp', 'totalOpnd', 'branchCount', 'defects', 'number_of_defects'],
    dtype='object')
```

Figure 4.3: Testing data implementation

4.3. Parameters Extraction Implementation

In the original data, there are 24 parameters and Correlation based techniques are used to find out the relationships between the different features of the data. It shows the positive results if the selected features have directly proportional relationship and shows the negative results vice versa. There are the two types of the correlations (a) Spearman Correlation (b) Pearson Correlation. The Pearson method can be applied only when the data satisfies the normal distribution. But sometimes data may not be in normal distributions. Similarly, in this case data don't follows the normal distribution curve. That's why we applied the Spearman Rank Correlation to find out the relationship between the required features.

4.3.1. Spearman Rank Correlation

▾ Spearman Correlation

```
[ ] colCorr = data.corr( method = 'spearman' )
```

▾ Stem Plot Creation and Saving

```
columnNames = colCorr.columns
for col in columnNames :
    plt.figure(figsize = (12,7))
    plt.stem(colCorr[col])
    plt.plot(0.42*np.ones((len(columnNames),1)), 'g--')
    plt.plot(-0.42*np.ones((len(columnNames),1)), 'g--')
    plt.title(col)
    plt.xlabel('Variables')
    plt.ylabel('Cross-Correlation')
    plt.xticks(np.arange(len(columnNames)),columnNames, rotation = 90)
    plt.savefig(col + '_stem.png')
    plt.show()
```

Figure 4.4: Correlation Analysis Implementation

Spearman correlation analysis was implemented because it is most suitable according to our testing data. Used corr (method = 'spearman') function in python language for finding a correlation between parameters. And we have found, three parameters have a positive strong relationship between them.

4.3.2. Interlinked Parameters

The correlation analysis shows that some part of software information Test Data is useless and need to wipe off.

Qualified Parameters are given below

- Software Size (LOC)
- Number of Test Case Designed
- Number of Defects

All the above parameters show a positive and strong correlation, so that's why we have selected these three parameters for further implementation.

4.4. Data Cleaning & Transformation

The next step is the data cleaning & transformation after parameters extraction. In the implementation of data cleaning, we have replaced all the missing values with the average or mean of the data. Further, In the data transformation, we have introduced two new variables while performing calculations on three extracted parameters.

▼ Transformation of Data

```
[ ] import pandas as pd
df = pd.read_csv('./cm1_new.csv')
df_focus = df[df.columns[-2:]]
df_focus = df_focus.rename(columns = {df_focus.columns[0]: 'Test_Case',
                                     df_focus.columns[1]: 'Defects_Density'}, inplace = False)

[ ] df_focus.head()
```

Figure 4.5: Data Transformation Implementation

Figure 4.5 shows the implementation of the data transformation, we can see there are two new variables test_case and Defects_denisty which we have formulated from the extracted parameters. We have performed calculations in the Excel data file and show parameters in the python environment.

4.4.1. Introducing Two New Parameters

Figure 4.6 shows the two new parameters which we have calculated. Test Case Density Per 100-line parameter calculated from total case designed and software size

parameters. Defect Density Per 1000-line parameter calculated from number of defects and software size parameters. Formulas are discussed in the methodology chapter and these two new parameters are also used for the implementation of the Apriori algorithm.

```
[ ] import pandas as pd
    df = pd.read_csv('./cm1_new.csv')
    df_focus = df[df.columns[-2:]]
    df_focus.head()
```

	Test Case Density Per 100 line	Defect Density per 1000 line
0	3.053473	4.147926
1	3.082180	3.759621
2	4.074258	5.114469
3	18.936895	22.008327
4	4.686373	5.991247

Figure 4.6: Data Transformation Implementation

4.5. Discretization of Processed Parameters

After processed the parameters, we have found five important parameters including two new parameters. Further, need to implement the Apriori algorithm testing data all selected parameters consist of the continuous numeric values. Apriori algorithm can't deal with a continuous numerical value. The clustering algorithm is applied to deal with the parameter's discretization.

▾ Test Case Density Per 100 line Clustering

```
[ ] # df_focus.Test_Case.astype('string')
    datacopy.Test_Case[ ( df_focus.Test_Case.astype(float) < 3 ) & ( df_focus.Test_Case.astype(float) >= 0 ) ] = 'Y1'
    datacopy.Test_Case[ ( df_focus.Test_Case.astype(float) < 9 ) & ( df_focus.Test_Case.astype(float) >= 3 ) ] = 'Y2'
    datacopy.Test_Case[ ( df_focus.Test_Case.astype(float) < 14 ) & ( df_focus.Test_Case.astype(float) >= 9 ) ] = 'Y3'
    datacopy.Test_Case[ ( df_focus.Test_Case.astype(float) < 20 ) & ( df_focus.Test_Case.astype(float) >= 14 ) ] = 'Y4'
    datacopy.Test_Case[ ( df_focus.Test_Case.astype(float) >= 20 ) ] = 'Y5'
```

▾ Defect Density Per 1000 line Clustering

```
[ ] # df_focus.Test_Case.astype('string')
    datacopy.Defects_Density[ ( df_focus.Defects_Density.astype(float) < 3 ) & ( df_focus.Defects_Density.astype(float) >= 0 ) ] = 'Q1'
    datacopy.Defects_Density[ ( df_focus.Defects_Density.astype(float) < 6 ) & ( df_focus.Defects_Density.astype(float) >= 3 ) ] = 'Q2'
    datacopy.Defects_Density[ ( df_focus.Defects_Density.astype(float) < 10 ) & ( df_focus.Defects_Density.astype(float) >= 6 ) ] = 'Q3'
    datacopy.Defects_Density[ ( df_focus.Defects_Density.astype(float) < 16 ) & ( df_focus.Defects_Density.astype(float) >= 10 ) ] = 'Q4'
    datacopy.Defects_Density[ ( df_focus.Defects_Density.astype(float) < 30 ) & ( df_focus.Defects_Density.astype(float) >= 16 ) ] = 'Q5'
    datacopy.Defects_Density[ ( df_focus.Defects_Density.astype(float) >= 30 ) ] = 'Q5'
```

Figure 4.7: Discretization Implementation

Figure 4.7 shows, how we have implemented the discretization while using the clustering technique. For both new parameters, we have implemented clusters or groups. For example, from 0 to <3 values for test case density parameter we have made a group or

cluster. From ≥ 3 to < 6 values for test case density parameter we have made another group or cluster. There is another cluster which covers from ≥ 6 to < 10 values for test case density parameter. Same clustering technique applied for the second parameter which is defect density per 10000 lines.

4.5.1. Labels Discretized Data

We need to assign labels to each cluster of the data, for this purpose we have defined two types of labels. Y series labels and Q series labels assigned one label to every cluster. For instance, Y1 assigned from 0 to < 3 values for test case density parameter. Y2 assigned from ≥ 3 to < 9 values for test case density parameter. Y3 assigned to another cluster which covers from ≥ 9 to < 14 values for test case density parameter. Y4 assigned to cluster which covers from ≥ 14 to < 20 values for test case density parameter.

```
[ ] labels1 = [ 'Y1', 'Y2', 'Y3', 'Y4', 'Y5' ]
labels2 = [ 'Q1', 'Q2', 'Q3', 'Q4', 'Q5' ]
for label in labels1:
    temp = datacopy[ datacopy.Test_Case == label ]
    print( np.array(np.unique(temp, return_counts=True)).T )
```

Figure 4.8: Labels Implementation

Q series labels assigned one label to every cluster of the defect density per 1000-line parameter. For instance, Q1 assigned from 0 to < 3 values for defect density parameter. Q2 assigned from ≥ 3 to < 6 values for defect density parameter. Q3 assigned to another cluster which covers from ≥ 6 to < 10 values for defect density parameter. Q4 assigned to cluster which covers from ≥ 10 to < 16 values for defect density parameter.

4.5.2. Occurrence of Variables

```
[ ] np.array(np.unique(datacopy, return_counts=True)).T

array([[ 'Q1', 115],
       [ 'Q2', 218],
       [ 'Q3',  89],
       [ 'Q4',  38],
       [ 'Q5',  38],
       [ 'Y1', 159],
       [ 'Y2', 261],
       [ 'Y3',  35],
       [ 'Y4',  22],
       [ 'Y5',  21]], dtype=object)
```

Figure 4.9: Variables Occurrence

After assigning labels to each group or cluster of values, we have checked the occurrence of the labels or variables in the complete testing data. In figure 4.9, we can see there are 115 observations in the data which have a Q1 label. In the same way, there are 218 records or observations with Q2 labels or variables. All the Q labels represent the defect density parameter values, and all the Y labels represent the test case density parameter values.

4.6. Implementation of Apriori Algorithm Based Model

Last but not the least, implementation of the Apriori algorithm under the Association Rule Mining technique. Apriori algorithm needs input data for finding the rules between defects and all the labels or variables represent defects in the testing data. Figure 4.10 shows the flow diagram of the model implementation. Model required input labels and applied the Apriori algorithm for finding the supports values. After checking the supports condition, the model decided association whether an association is strong or weak between defects.

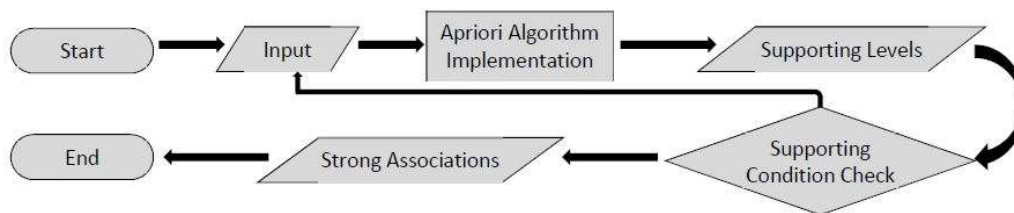


Figure 4.10: Model Flow Diagram

4.5.1. Apriori Algorithm Implementation

"Apriori Algorithm" implementation can be done with the help of apyori library in the python language. Apyori library provided us Apriori package and can be imported while using "from apyori import apriori". We have set support, lift, and confidence values in the implementation and find out the defect's rules based on the different values. Apriori function is available under the Apyori library, and we have used it with different parameters such as min_support, min_confidence, min_lift, and min_length. From this function, we have collected a list of defect rules and assign them to the association_results variable. Figure 4.11 shows the implementation of the Apriori algorithm with different values of support, confidence, lift, and length.


```

▶ from apyori import apriori
support = 0.25
lift = 0
a_length = 2
confidence = 0.5
# association_rules = apriori(datacopy.values)
association_rules = apriori(datacopy.values, min_support=support,
                             min_confidence=confidence, min_lift=lift, min_length=a_length)
association_results = list(association_rules)
for item in association_results:
    # first index of the inner list
    # Contains base item and add item
    print(item)

```

Figure 4.11: Implementation of Apriori Algorithm

We have used for loop to display the results of the association between defects. The list function is used for generating a list of the rules because the Apriori function returns the rules, not in the list form. So, we need to represent in a suitable form that's why we have implemented the list for the association rules.

4.7. Defect Rules Evaluation

Further, we need to evaluate the results or defect rules which we have found from the Apriori algorithm under the association rule mining technique. We have evaluated rules based on the different values of minimum confidence, minimum support, and minimum lift values. All of the values provided us different defect rules and based on the different defect rules we have decided which one is the best defect rule.

Chapter 5

Model Results & Visualization

CHAPTER 5: MODEL RESULTS & VISUALIZATION

In this chapter, we have discussed model results in detail. We have used different data pre-processing, parameters extraction, and Apriori algorithm implementation techniques. Apriori algorithm implemented under Association Rule Mining technique, and we have used different values of support, confidence, and lift for evaluation of the results.

5.1. Parameters Extraction Result

We have found three important and interlinked parameters form parameters extraction. Correlation analysis shows that some part of software information Test Data is useless and need to wipe off.

5.1.1. Dataset 1 Important Parameters Result

First, we have performed the correlation analysis for the first dataset. From this dataset, we have found the above three discussed parameters and these parameters show a positive and strong relationship for the first dataset.

The figure shows a dense matrix of correlation coefficients between 26 variables. The diagonal elements are all 1.0, representing self-correlation. Other values range from approximately -0.0029 to 0.5779. The variables listed are: Unnamed, loc, v(g), n, i, d, e, b, t, IOCode, IOComme, codeSize, IOBlank, testCase, locCode, Ar, unig, Op, unis, Opn, total, Op, total, Opn, branch, Coi, and defectNumber.

Figure 5.1: Correlation Analysis of Dataset 1

Parameter’s software size, the number of test cases design, and the number of defects is the important parameters because these three parameters show a strong and positive correlation between them. That’s why we have selected these three parameters.

Table 5.1: Correlation Analysis Results (Dataset 1)

Software Size		Number of Test Case Designed	Number of Defects
Software Size	1	0.57	0.668

Number of test case designed	0.57	1	0.91
Number of defects	0.668	0.91	1

Table 5.1 represents Software size and number of test case designed show 0.57 correlation value and Software size and number of defects show 0.668 correlation value. Number of test case designed, and number of defects show 0.91 correlation value.

5.1.2. Dataset 2 Important Parameters Result

After first dataset, we have performed the correlation analysis for the second dataset.

Figure 5.2: Correlation Analysis of Dataset 2

Figure 5.2 shows the Parameter's software size, the number of test cases design, and the number of defects is the important parameters because these three parameters show a strong and positive correlation between them. That's why we have selected these three parameters.

Table 5.2: Correlation Analysis Results (Dataset 2)

Software Size	1	0.57	0.634
Number of test case designed	0.57	1	0.90

Number of defects	0.634	0.90	1
--------------------------	-------	------	---

Table 5.2 represents Software size and number of test case designed show 0.57 correlation value and Software size and number of defects show 0.634 correlation value. Number of test case designed, and number of defects show 0.90 correlation value.

5.1.3. Extracted Parameters Visualization Dataset 1

This section shows the visualization of extracted parameters from the correlation analysis. Figure 5.3 shows Code size correlation stem visualization, it has a strong and positive relationship with test cases and the number of defects. We have set a correlation criterion which is 0.45, all the above values of 0.45 are all positively correlated.

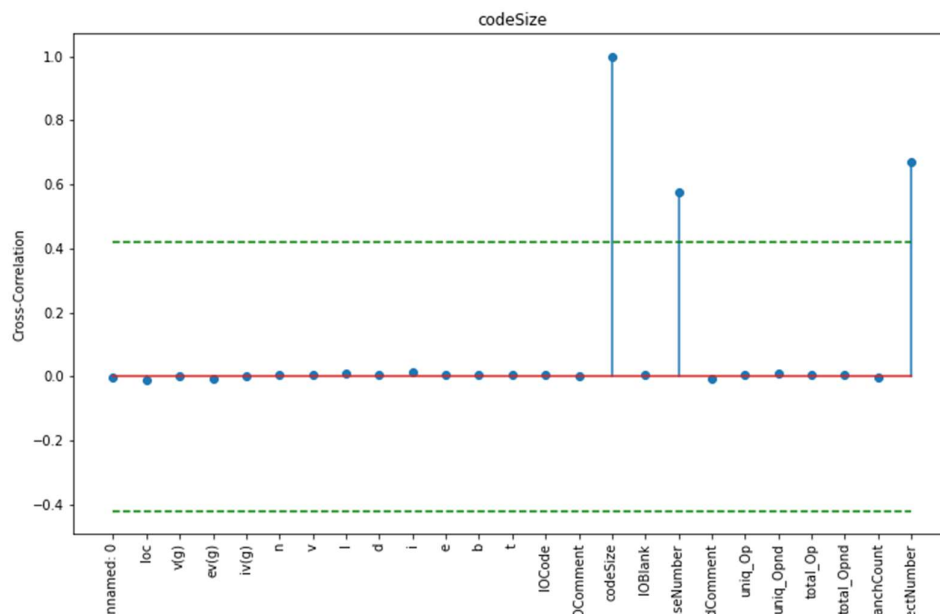


Figure 5.3: CodeSize Parameter Correlation Result (Data 1)

Next, there is a visualization of the second most important parameter. Figure 5.4 shows test case number correlation analysis stem visualization, it shows a positive and strong relationship with code size and the number of defects. Because these two parameters have a correlation value above 0.45.

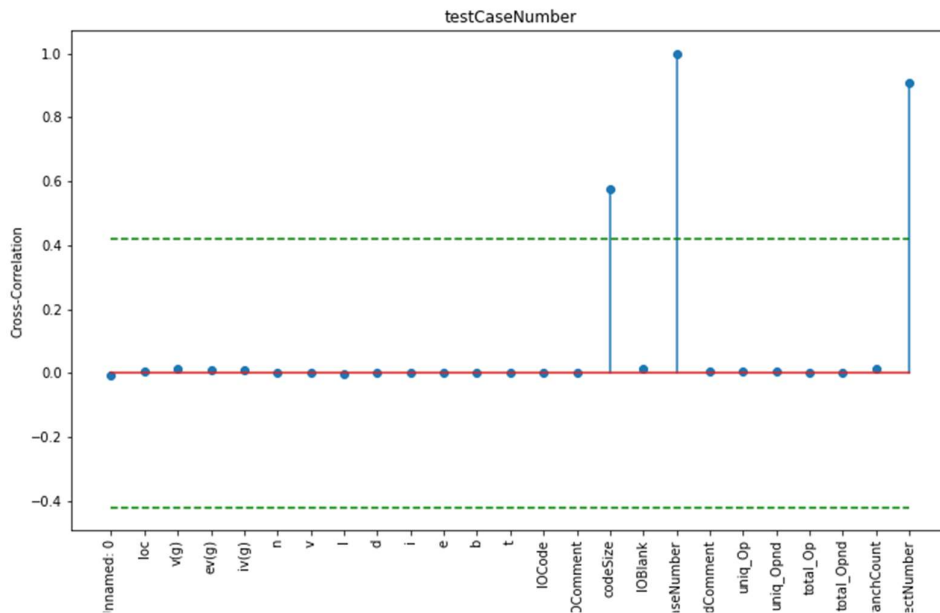


Figure 5.4: TestCaseNumber Parameter Correlation Result (Data 1)

Further, there is a visualization of the third interlinked parameter which is number of defects. Figure 5.5 represents number of defects correlation stem visualization; it shows a positive and strong correlation with code size and number of test cases. Because these two parameters have a correlation value above 0.45.

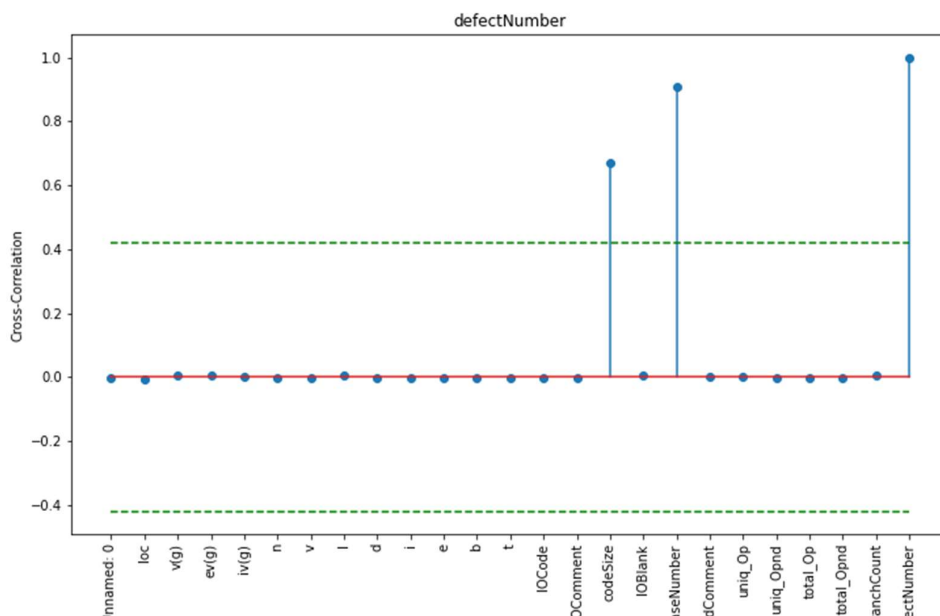


Figure 5.5: DefectNumber Parameter Correlation Result (Data 1)

5.1.4. Extracted Parameters Visualization Dataset 2

In this section, we have visualized correlation analysis for the second selected testing dataset. We have extracted the same parameters from the second testing data with the help of correlation analysis. Figure 5.6 shows Code size correlation stem visualization, it has a strong and positive relationship with test cases and the number of defects. Same as the first dataset correlation analysis, we have used a minimum positive correlation value is 0.45.

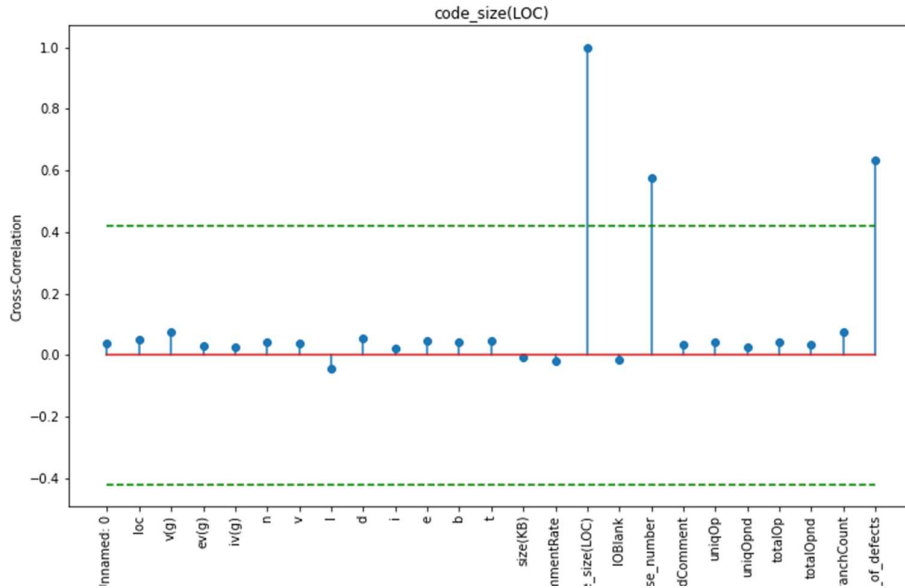


Figure 5.6: CodeSize Parameter Correlation Result (Data 2)

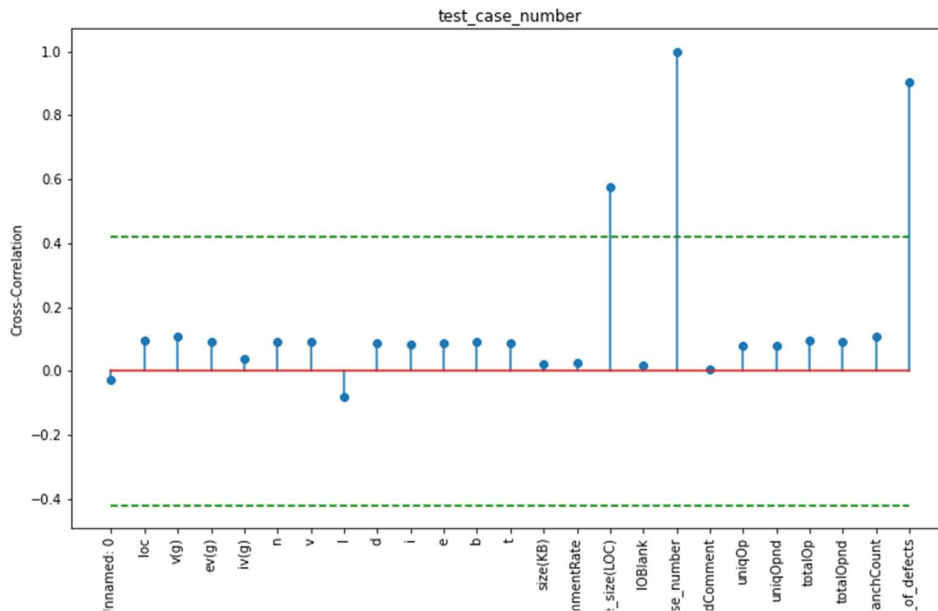


Figure 5.7: TestCaseNumber Parameter Correlation Result (Data 2)

Next, there is a visualization of the second most important parameter. Figure 5.7 shows test case number correlation analysis stem visualization, it shows a positive and strong relationship with code size and the number of defects.

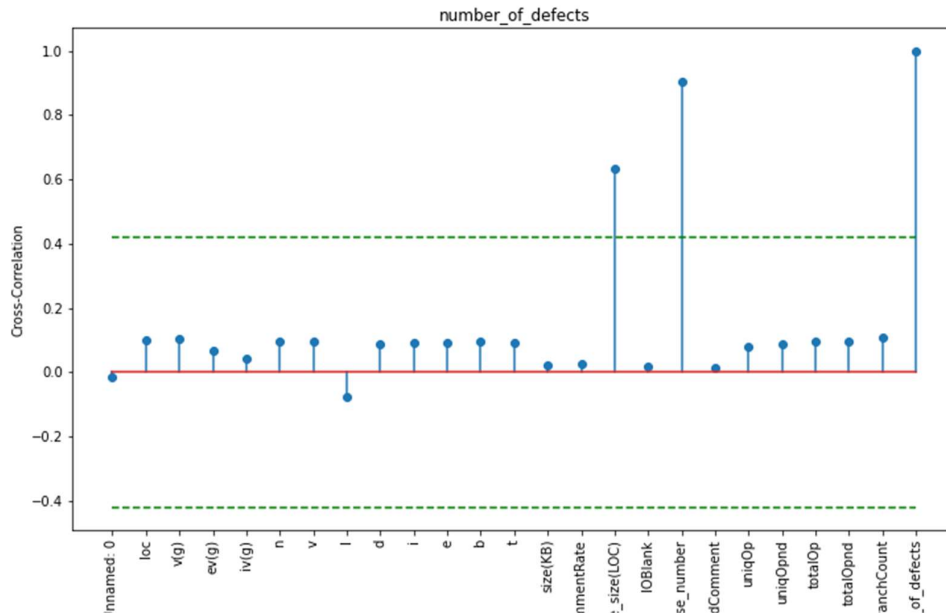


Figure 5.8: DefectNumber Parameter Correlation Result (Data 2)

Further, there is a visualization of the third interlinked parameter which is number of defects. Figure 5.8 represents number of defects correlation stem visualization; it shows a positive and strong correlation with code size and number of test cases.

5.2. New Parameters Results

After finding a correlation between parameters, we have found there are only three parameters that show the positive and strong relationship between them. Spearman Correlation Analysis excluded 19 variables because there were 22 variables in each dataset. From all datasets, we have found only three parameters that have an impact on the defect prediction in testing data.

For defect prediction, we need to calculate two new parameters based on the line of code. We have calculated two parameters, one with 100 lines of code and the second with 1000 lines of code. These two variables or parameters can use for predicting the defects in testing data. We have used formulas for calculating these two parameters and named these as Test Case Density Per 100 lines, and Defect Density Per 1000 lines.

K	L	M	N	O	P	Q	R	S	T	U	V	W	Y	Z	AA	AB	
e	b	t	size(KB)	comment	code_size(LOC)	IOBlank	test_case_num	locCodeA	uniqOp	uniqOpnd	totalOp	totalOpn	defects	num	Test Case Density Per 100 line	Defect Density per 1000 line	
1.3	1.3	1.3	2	2	60030	2	1833	2	1.2	1.2	1.2	1.2	1.2	b'false'	249	3.053473263	4.147926037
1	1	1	1	1	84583	1	2607	1	1	1	1	1	1	b'true'	318	3.082179634	3.759620728
2936.77	0.1	163.15	1	0	173234	6	7058	0	15	15	44	19	b'false'	886	4.07425794	5.114469446	
3447.89	0.07	191.55	0	0	18493	3	3502	0	16	8	31	16	b'false'	407	18.93689504	22.00832748	
5999.58	0.12	333.31	0	0	98477	3	4615	0	16	12	46	26	b'false'	590	4.686373468	5.991246687	
5999.58	0.12	333.31	0	0	24989	3	4447	0	16	12	46	26	b'false'	370	17.79583017	14.80651487	
69.74	0.01	3.87	0	0	59597	1	1358	0	4	5	6	5	b'false'	233	2.426863484	4.163911575	
604.36	0.03	33.58	0	0	34426	7	564	0	10	7	14	9	b'false'	70	1.638296636	2.033346889	
7820.87	0.18	434.49	12	16	96332	13	7389	0	15	20	69	38	b'false'	752	7.670348379	7.80633642	
30377.95	0.45	1687.66	8	35	11000	22	281	0	15	37	129	110	b'false'	51	2.554545455	4.636363636	
17773.08	0.29	987.39	11	28	93282	16	3943	0	19	27	96	59	b'false'	547	4.226967689	5.863939452	
1287.55	0.05	71.53	2	4	67191	4	922	0	9	8	19	16	b'false'	162	1.372207587	2.411037192	
21659.58	0.26	1203.31	10	17	67823	23	2285	0	17	13	114	43	b'false'	288	3.369063592	4.246347109	
12929.85	0.16	718.32	10	17	40171	23	1553	0	14	9	70	35	b'false'	62	3.865972966	1.543401957	
35852.6	0.43	1991.81	2	15	37001	40	3905	0	22	28	161	70	b'false'	293	10.55376882	7.9187049	
9958	0.22	553.22	3	20	66103	23	355	0	19	25	80	40	b'false'	50	0.537040679	0.756395322	
2480.95	0.09	137.83	6	5	23778	8	4168	0	14	13	40	17	b'false'	514	17.52880814	21.61662041	
222.03	0.02	12.33	1	1	27618	2	496	0	8	5	10	5	b'false'	50	1.79593019	1.810413498	
8684.99	0.25	482.5	2	0	97162	19	4724	0	12	34	69	66	b'false'	635	4.86198308	6.535476833	
222.03	0.02	12.33	1	1	101622	0	6046	0	8	5	10	5	b'false'	567	5.949499124	5.579500502	
1265.83	0.04	70.32	0	0	59081	1	330	0	10	5	15	12	b'false'	102	0.558555204	1.726443357	

Figure 5.9: Two News Parameters

Figure 5.9 shows the results of the Two new parameters, the last two columns AA and AB in the excel data represent them. After calculating two parameters, there were five Parameters that can participate in the defect prediction using Apriori Algorithm. All of the variables or parameters depend on the continuous values. So, we have performed discretization to normalize the values. We have used clustering and assign labels to each group of values.

5.3. Discretization Results

To normalize or discretize the values of the parameters, we have applied the clustering technique to generate a group of values.

5.3.1. Test Case Density Per 100 Lines Parameter

In table 5.3, we can see there are five groups of value for one hundred lines. Further, we have assigned labels to each cluster or group such as Y1, Y2, Y3, Y4, and Y5. All of these defect labels and we have used them in the Apriori Algorithm for predicting defect rules.

Table 5.3: Test Case Density Per 100 lines

Sr	Scope (one hundred line)	Scope Label
1	[0,3]	Y1
2	(3,9]	Y2
3	(9,14]	Y3
4	(14,20]	Y4
5	(20, ∞]	Y5

From 0 to 3 values there is Y1 type defect, from 3 to 9 values there is Y2 type defect, from 9 to 14 values there is Y3 type defect, from 14 to 20 values there is Y4 type defect, and from 20 to ∞ there is Y5 type defect in testing data. All these defect ranges assigned to the Test Case Density Per 100 lines parameter.

5.3.2. Defect Density Per 1000 Lines Parameter

In table 5.4, we can see there are five groups of value for one thousand lines. We have assigned labels to each cluster or group such as Q1, Q2, Q3, Q4, Q5, and Q6. All these defect labels and we have used them in the Apriori Algorithm for predicting defect rules.

Table 5.4: Defect Density Per 1000 lines

Sr	Scope (one Thousand line)	Scope Label
1	[0,3]	Q1
2	(3,6]	Q2
3	(6,10]	Q3
4	(10,16]	Q4
5	(16,30]	Q5
6	(30, ∞]	Q6

From 0 to 3 values there is Q1 type defect, from 3 to 6 values there is Q2 type defect, from 6 to 10 values there is Q3 type defect, from 10 to 16 values there is Q4 type defect, from 16 to 30 values there is Q5 type defect, and from 30 to ∞ there is Q6 type defect in testing data. All these defect ranges assigned to the Defect Density Per 1000 lines parameter.

5.4. Association Rules Analysis Results

First, test case density is calculated and then we check the level of support for the different orders sets. As displayed in the following table number 5.5.

Table 5.5: Test Case Density Supporting Level

Sr	Order set	Supporting Level
1	Y1	0.354213
2	Y2	0.461389
3	Y3	0.02838
4	Y4	0.066834

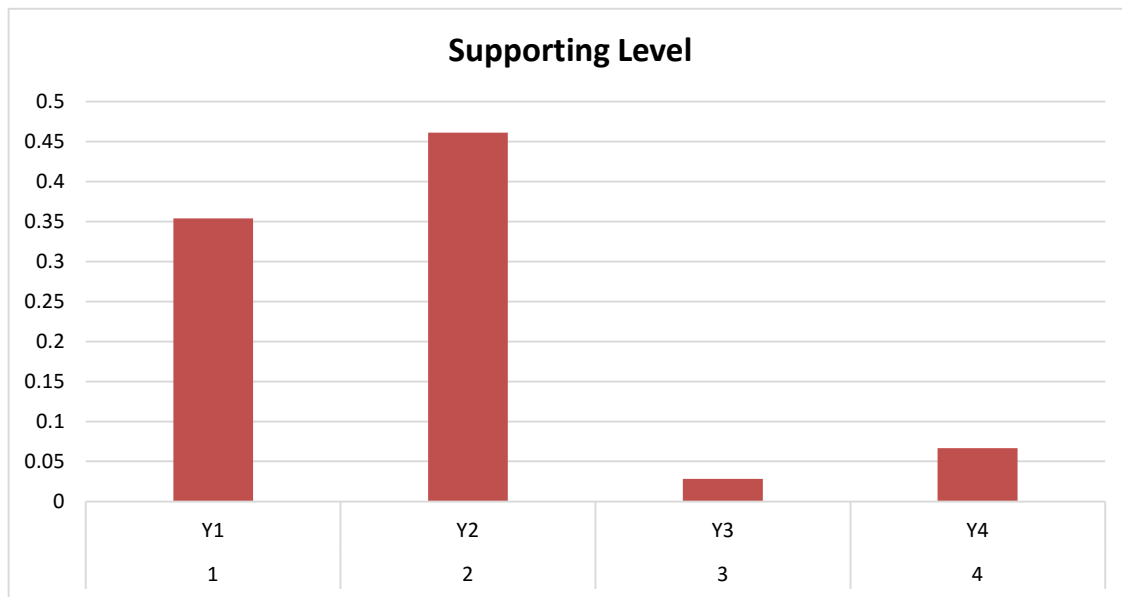


Figure 5.10: Test Case Density Support Level

The level of support of Y1 and Y2 greater than the threshold value, so we can say the most recurrent items in the given sets are Y1 and Y2. After finding the density ratios we figured out the most common set of items in the following table 5.6 using the items of test case set and defect sets.

Table 5.6: Supporting Level of Item sets

Sr	Set of Items	Supporting Level
1	Y1, Q1	0.26293
2	Y1, Q2	0.065812
3	Y1, Q3	0.024274
4	Y1, Q4	0.001325
5	Y1, Q5	0.001325
6	Y1, Q6	0.0
7	Y2, Q1	0.272670
8	Y2, Q2	0.247863
9	Y2, Q3	0.055556
10	Y2, Q4	0.014530
11	Y2, Q5	0.0
12	Y2, Q6	0.004017

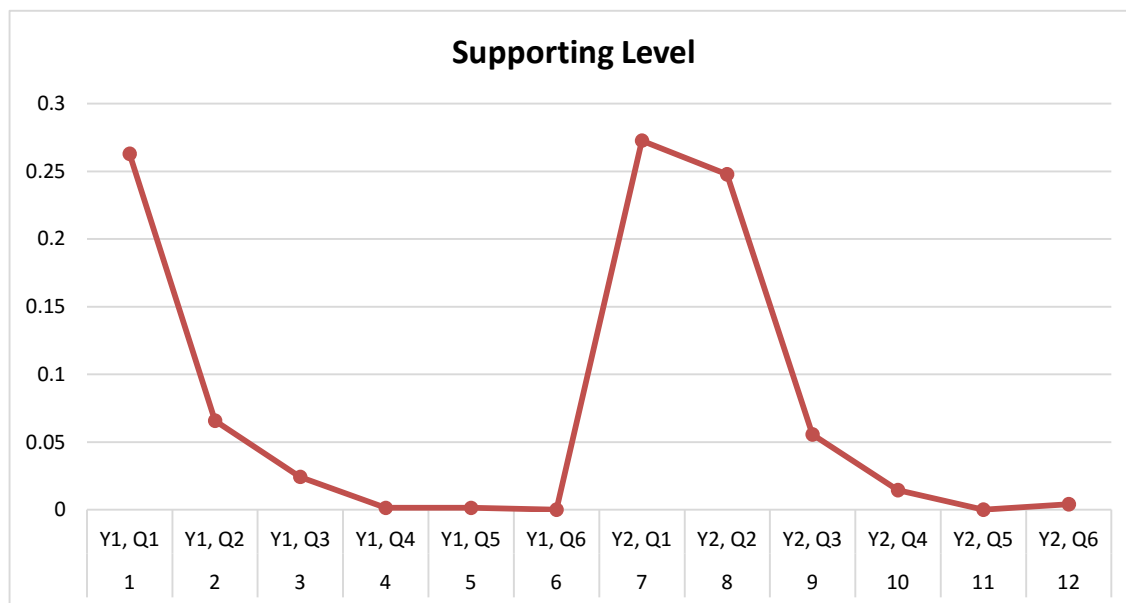


Figure 5.11: Item sets Support Level

The most common set of items are trimmed and checked with threshold level of ten percent. And the threshold for the level of confidence is set as thirty five percent. The then the level of confidence is figured out for both Y1 and Y2.

All of them are greater than the thirty five percent. After performing all above extensive calculations the final association and relationships are shown in the following table number 5.7.

Table 5.7: Supporting Level & Confidence of Item sets

Sr	Item set	Supporting Level	Confidence Level
1	Y1, Q1	0.251282	0.251282/0.364103 = 0.6901
2	Y2, Q1	0.194872	0.194872/0.451282 = 0.4318
3	Y2, Q2	0.158974	0.158974/0.451282 = 0.3523

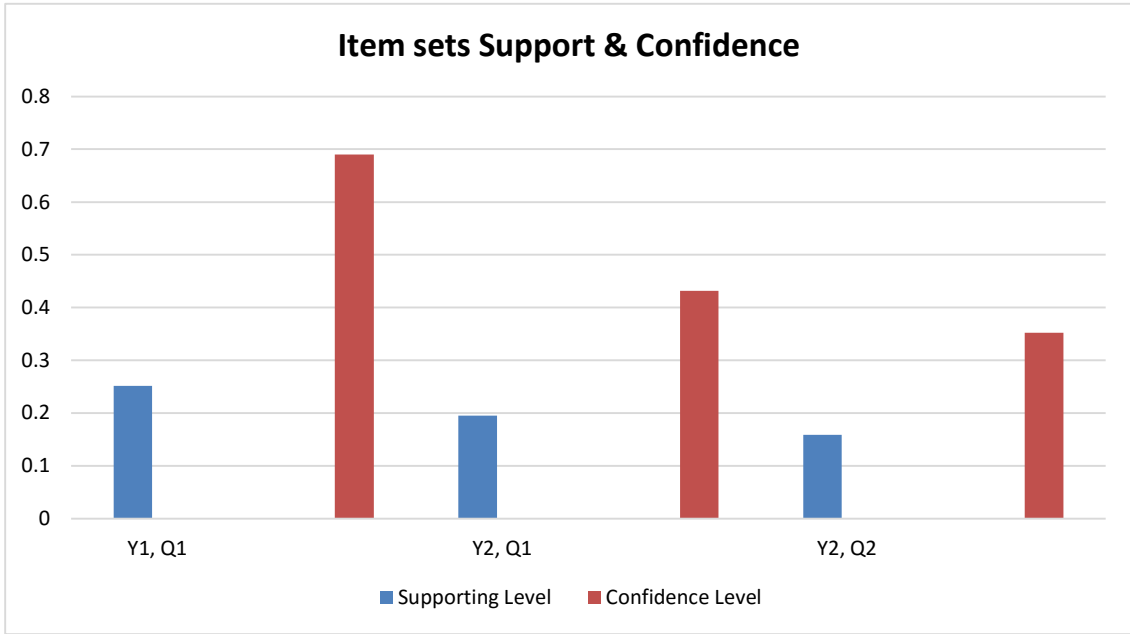


Figure 5.12: Final Defect Rules

Chapter 6

Conclusion and Future Work

CHAPTER 6: CONCLUSION AND FUTURE WORK

This chapter is about the discussion of the complete research study in the form of a conclusion and a discussion of the future work. We can improve this research study and results while using different approaches and techniques in the future.

6.1. Conclusion

In common words as we all know the software testing is one of the most important phases of the software development life cycle which helps in reducing the number of defects and increasing the quality of the software.

In this research study, we used Data Mining (DM) techniques to predict defects from the software testing data. With the help of data mining techniques, we can improve the reliability and quality of the software. First, we have identified some available software testing datasets and selected data based on the parameters and requirements of the research study. For this purpose, we have explored related studies in the Literature Review (LR) and identified some defect prediction datasets & techniques. Based on the literature review, we have found different defect prediction techniques and chose the best one for designing and implementing research methodology.

We first applied the Spearman Rank Correlation to figure out the relevant features then we applied the Apriori based association rule mining technique to generate the associations among these variables. Only those parameters are used which effects most the quality of the software. ie size of the software (“Software Size-KBs”), time for the starting of testing of the software (“Start Time”), the language (“Language”) in which it is being developed, and the total number of the test cases (“No of Test Cases”) and etc.

The association algorithm cannot be applied at the continuous data, so we calculated the density functions with respect to size of the software. Then we used iterative Apriori technique to find the associations between these density functions. At last, after finding the associations we analyzed that the number of the test cases applied and the size of the software the two most associated variables with the number of defects in the software product.

In last, we have found different association rules for defects of testing data. All the defect rules were examined were based on the support level, confidence, and lift values. We have implemented an Apriori algorithm with different values of support level, confidence, and lift. Based on the values, we have found different scores and different defect rules. Y1, Q1 rule, Y2, Q2 rule, and Y2, Q1 rule show the best support level and confidence as compared to other results.

6.2. Future Work

Based on the research study, we have found some important defect rules and we can improve our study in the future.

- Inclusion of more features for proper effort estimation of the test cases creation
- Use of more advanced feature selection and machine learning techniques to conduct a more rigid analysis
- Improved data set collections

References

- [1] P.G.Frankl and S.N.Weiss, "An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing," in *IEEE Trans. Software Eng.*, 1993, vol. 19, no. 8, pp. 774-787.
- [2] M.Hutchins and H.Foster and T.Goradia and T.J.Ostrand, "Experiments of the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria," in *16th International Conference on Software Engineering*, Sorrento, 1994, pp. 191-200.
- [3] G.Rothermel and M.J.Harrold and J.V.Ronne and C.Hong, "Empirical studies of test-suite reduction," in *Softw. Test., Verif. Reliab.*, 2002, vol. 12, no. 3, pp. 219-249.
- [4] P.Garg, "Investigating coverage-reliability relationship and sensitivity of reliability to errors in the operational profile," in *Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, pp. 21-35.
- [5] M.Subbarao. (2008, Oct.). Is Code Coverage Important? [Online]. Available: <http://java.dzone.com/articles/is-code-coverage-important>
- [6] X.Cai and M.R.Lyu, "The effect of code coverage on fault detection under different testing profiles," in *ACM SIGSOFT Software Engineering Notes*, 2005, vol. 30, no. 4, pp.1-7.
- [7] P.G.Frankl and O.Iakounenko, "Further Empirical Studies of Test Effectiveness," in *SIGSOFT FSE*, 1998, pp. 153-162.
- [8] J.R.Horgan and S.London and M.R.Lyu, "Achieving Software Quality with Testing Coverage Measures," in *IEEE Computer*, 1994, vol. 27, no. 7, pp. 60-69.
- [9] C.Kaner (1996, May). Software Negligence and Testing Coverage. [Online]. Available:http://www.kaner.com/pdfs/negligence_and_testing_coverage.pdf
- [10] B.Smith and L.Williams. (2008, Sep.). A Survey on Code Coverage as a Stopping Criterion for Unit Testing. [Online]. Available: ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2008/TR-2008-22.pdf
- [11] G.Handerson. (2009, Mar.). How Much Test Coverage Is Enough? [Online]. Available: <http://www.summa-tech.com/blog/2009/03/24/how-much-test-coverage-is-enough/>
- [12] A.Ruiz. (2010, Apr.). When 100% Coverage Gives Us a False Sense of Security. [Online]. Available: <http://java.dzone.com/articles/when-100-coverage-gives-us/>

- [13] A.Glover. (2006, Jan.). In pursuit of code quality: Don't be fooled by the coverage report. [Online]. Available: <http://www.ibm.com/developerworks/java/library/j-cq01316/>
- [14] W.E.Wong and J.R.Horgan and S.London and A.P.Mathur, "Effect of Test Set Size and Block Coverage on the Fault Detection Effectiveness," in *Proceedings of the Fifth International Symposium on Software Reliability Engineering*, 1994, pp. 230-238.
- [15] J.R.Horgan and S.London and M.R.Lyu, "Achieving Software Quality with Testing Coverage Measures," in *IEEE Computer*, 1994, vol. 27, no. 7, pp. 60-69.
- [16] X.Cai and M.R.Lyu, "Software Reliability Modeling with Test Coverage: Experimentation and Measurement with A Fault-Tolerant Software Project," in *The 18th IEEE International Symposium on Software Reliability*, Sweden, 2007, pp. 17-26.
- [17] M.Chen and M.R.Lyu and W.E.Wong, "Incorporating Code Coverage in the Reliability Estimation for Fault-Tolerant Software," in *IEEE Symposium on Reliable Distributed Systems*, 1997, pp. 45-52.
- [18] Sun, Z., Song, Q., & Zhu, X. (2012). Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6), 1806-1817.
- [19] Wang, H., Khoshgoftaar, T. M., & Napolitano, A. (2012). Software measurement data reduction using ensemble techniques. *Neurocomputing*, 92, 124-132.
- [20] Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability*, 62(2), 434-443.
- [21] Khoshgoftaar, T. M., Gao, K., & Napolitano, A. (2014, August). Improving software quality estimation by combining feature selection strategies with sampled ensemble learning. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)* (pp. 428-433). IEEE.
- [22] Claesen, M., De Smet, F., Suykens, J., & De Moor, B. (2014). EnsembleSVM: A library for ensemble learning using support vector machines. *arXiv preprint arXiv:1403.0745*.
- [23] Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58, 388-402.
- [24] F. M. Tua and W. Danar Sunindyo, "Software Defect Prediction Using Software Metrics with Naïve Bayes and Rule Mining Association Methods," 2019 5th International Conference on Science and Technology (ICST), 2019, pp. 1-5, doi:

10.1109/ICST47872.2019.9166448.

- [25] W.N.Venables and D.M.Smith and R Development Core. (2008, Feb.). Introduction to R. [Online]. Available: <http://www.statpower.net/Content/312/Handout/R-intro.pdf>
- [26] C. Atkinson, F. Barth and D. Brenner, "Software Testing Using Test Sheets," *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, 2010, pp. 454-459, doi: 10.1109/ICSTW.2010.21.
- [27] A. Mori, "Anomaly Analyses to Guide Software Testing Activity," *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 427-429, doi: 10.1109/ICST46399.2020.00055.
- [28] E. Enoiu, G. Tukseferi and R. Feldt, "Towards a Model of Testers' Cognitive Processes: Software Testing as a Problem Solving Approach," *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2020, pp. 272-279, doi: 10.1109/QRS-C51114.2020.00053.
- [29] J. F. Leathrum and K. A. Liburdy, "Automated testing of open software standards," *Proceedings of IEEE International Test Conference - (ITC)*, 1993, pp. 854-858, doi: 10.1109/TEST.1993.470616.
- [30] P. Leesutthipornchai and D. Pradubsuwun. Association Extraction from Functional Testing Scenarios. In *Proceedings of the 2019 International Electronics Communication Conference (IECC '19)*. Association for Computing Machinery, New York, NY, USA, 27–31. DOI:<https://doi.org/10.1145/3343147.3343151>
- [31] F. Matloob et al., "Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review," in *IEEE Access*, vol. 9, pp. 98754-98771, 2021, doi: 10.1109/ACCESS.2021.3095559.
- [32] C. Atkinson, F. Barth and D. Brenner, "Software Testing Using Test Sheets," *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, 2010, pp. 454-459, doi: 10.1109/ICSTW.2010.21.
- [33] Toure, F., Badri, M. & Lamontagne, L. *Innovations Syst Softw Eng* (2018) 14: 15. <https://doi.org/10.1007/s11334-017-0306-1>
- [34] Nalepa G.J. (2018) Using Rules to Support Software Testing. In: *Modeling with Rules Using Semantic Knowledge Engineering*. Intelligent Systems Reference Library, vol 130. Springer, Cham
- [35] A. Kitchenham, B. (2007). Kitchenham, B.: Guidelines for performing Systematic Literature Reviews in software engineering. EBSE Technical Report EBSE-2007-01.

- [36] Ezgi Erturk and Ebru Akcapinar Sezer. 2015. A comparison of some soft computing methods for software fault prediction. *Expert Syst. Appl.* 42, 4 (March 2015), 1872-1879. DOI: <http://dx.doi.org/10.1016/j.eswa.2014.10.025>
- [37] G. Abaei, A. Selamat, H. Fujita. An empirical study based on semisupervised hybrid self-organizing map for software fault prediction, *Knowl.- Based Syst.*, 74 (2015), pp. 28-39
- [38] D. Čeke, B. Milašinović. Early effort estimation in web application development. *J. Syst. Softw.*, 103 (2015), pp. 219-237
- [39] D. Cotroneo, D. D. Leo, R. Natella and R. Pietrantuono, "Prediction of the Testing Effort for the Safety Certification of Open-Source Software: A Case Study on a Real-Time Operating System," 2016 12th European Dependable Computing Conference (EDCC), Gothenburg, 2016, pp. 141- 152. doi: 10.1109/EDCC.2016.22
- [40] Keith B., Vega V. (2017) Process mining applications in software engineering. In: Mejia J., Muñoz M., Rocha Á., San Feliu T., Peña A. (eds) *Trends and Applications in Software Engineering. CIMPS 2016. Advances in Intelligent Systems and Computing*, vol 537. Springer, Cham
- [41] Rehman S., Sharma A. (2017) Privacy Preserving Data Mining Using Association Rule Based on Apriori Algorithm. In: Singh D., Raman B., Luhach A., Lingras P. (eds) *Advanced Informatics for Computing Research. Communications in Computer and Information Science*, vol 712. Springer, Singapore
- [42] Yang Y, He M, Li M, Wang Q, Boehm BW (2008) Phase distribution of software development effort. In: *Proceedings of the second ACM-IEEE international symposium on empirical software engineering and measurement, ESEM '08*, pp 61–69. ACM, New York, NY, USA
- [43] Acharya A.A., Mahali P., Mohapatra D.P. (2015) Model Based Test Case Prioritization Using Association Rule Mining. In: Jain L., Behera H., Mandal J., Mohapatra D. (eds) *Computational Intelligence in Data Mining - Volume 3. Smart Innovation, Systems and Technologies*, vol 33. Springer, New Delhi