****UNIVERSITY OF HERTFORDSHIRE  
School of Physics, Engineering, and Computer Science

Advance master project (7COM1039) Date

**Optimizing Bug Resolution with NLP and ML: Automated Summarization and Severity Prediction**

Name: **Venkatesh Chattu**

Student ID: **22074132**

Supervisor: **Gulay Gurkan**

# Contents Page

[Contents Page 2](#_Toc186926098)

[Table of Tables 5](#_Toc186926099)

[Table of Figures 6](#_Toc186926100)

[Acknowledgement 7](#_Toc186926101)

[Abstract 9](#_Toc186926102)

[Chapter 1 10](#_Toc186926103)

[Introduction 10](#_Toc186926104)

[1.1 Background of the Study 10](#_Toc186926105)

[1.2 Statement of the Problem 11](#_Toc186926106)

[1.3 Objectives 12](#_Toc186926107)

[1.4 Research Questions and Hypotheses 12](#_Toc186926108)

[1.5 Significance of the Project 13](#_Toc186926109)

[1.6 Ethical Considerations 13](#_Toc186926110)

[1.7 Chapter Summary 14](#_Toc186926111)

[Chapter 2 15](#_Toc186926112)

[Literature Survey 15](#_Toc186926113)

[2.1 Introduction 15](#_Toc186926114)

[2.2 Literature Survey 15](#_Toc186926115)

[2.3 NLP and ML Techniques for Bug Prioritization and Severity Prediction 15](#_Toc186926116)

[2.4 Feature Engineering and Advanced Models for Severity Prediction 19](#_Toc186926117)

[2.5 Summarization and Integrated Frameworks for Bug Triaging 22](#_Toc186926118)

[2.6 Research Gaps 25](#_Toc186926119)

[2.7 Key Findings 26](#_Toc186926120)

[2.8 Chapter Summary 26](#_Toc186926121)

[Chapter 3 27](#_Toc186926122)

[Methodology 27](#_Toc186926123)

[3.1 Introduction 27](#_Toc186926124)

[3.2 Significance of the Methodology 27](#_Toc186926125)

[3.3 Methodology Objectives 28](#_Toc186926126)

[3.4 Tools, Techniques, Software, and Frameworks 29](#_Toc186926127)

[3.5 Chapter Summary 31](#_Toc186926128)

[Chapter 4 32](#_Toc186926129)

[Implementation 32](#_Toc186926130)

[4.1 Introduction 32](#_Toc186926131)

[4.1 Data Collection 32](#_Toc186926132)

[4.2 Data Preprocessing 33](#_Toc186926133)

[4.2.1 Exploratory Data Analysis (EDA) 33](#_Toc186926134)

[4.2.2 Preprocessing Steps 34](#_Toc186926135)

[4.2.3 Preprocessing Tools 36](#_Toc186926136)

[4.2.4 Clustering Techniques 36](#_Toc186926137)

[4.3 Model Development 38](#_Toc186926138)

[4.3.1 Bug Report Summarization 38](#_Toc186926139)

[4.3.2 Severity Classification 39](#_Toc186926140)

[4.3.3 Developer Assignment 42](#_Toc186926141)

[4.3.4 Advanced Techniques 43](#_Toc186926142)

[4.4 Model Evaluation 44](#_Toc186926143)

[4.4.1 Evaluation Metrics 44](#_Toc186926144)

[4.4.2 Validation Techniques 46](#_Toc186926145)

[4.4.3 Comparative Analysis 46](#_Toc186926146)

[4.4.4 Insights Gained 47](#_Toc186926147)

[4.5 Chapter Summary 47](#_Toc186926148)

[Chapter 5 48](#_Toc186926149)

[Data and Analysis 48](#_Toc186926150)

[5.1 Introduction 48](#_Toc186926151)

[5.2 Data Exploration and Visualization 48](#_Toc186926152)

[5.3 Summary of Key Insights 53](#_Toc186926153)

[5.4 Chapter Summary 54](#_Toc186926154)

[Chapter 6 55](#_Toc186926155)

[Evaluation 55](#_Toc186926156)

[6.1 Introduction 55](#_Toc186926157)

[6.2 Model Performance Results 55](#_Toc186926158)

[6.2.1 Random Forest Classifier 55](#_Toc186926159)

[6.2.2 Support Vector Machine (SVM) Classifier 57](#_Toc186926160)

[6.2.3 Ensemble Model 59](#_Toc186926161)

[6.3 Fitting with Autoencoders 61](#_Toc186926162)

[6.4 Model Interpretability 63](#_Toc186926163)

[6.5 Comparative Analysis of Models 64](#_Toc186926164)

[6.6 Chapter Summary 65](#_Toc186926165)

[Chapter 7 66](#_Toc186926166)

[Conclusion 66](#_Toc186926167)

[7.1 Objectives Achieved 66](#_Toc186926168)

[7.2 Dataset Quality and Preprocessing 66](#_Toc186926169)

[7.3 Model Training and Evaluation 67](#_Toc186926170)

[7.4 Contributions and Future Directions 67](#_Toc186926171)

[7.5 Final Remarks 68](#_Toc186926172)

[References 69](#_Toc186926173)

[Appendix A: Tools and Technologies Used 74](#_Toc186926174)

[Appendix B: Code Snippet 77](#_Toc186926175)

# Table of Tables

|  |  |  |
| --- | --- | --- |
| Table No. | Title | Page No. |
| Table 6.1 | Classification Metrics for Random Forest Classifier | 60 |
| Table 6.2 | Classification Metrics for SVM Classifier | 61 |
| Table 6.3 | Classification Metrics for Ensemble Model | 63 |
| Table 6.4 | Performance Metrics for Comparative Models | 67 |

# Table of Figures

|  |  |  |
| --- | --- | --- |
| Figure No. | Title | Page No. |
| Figure 3.1 | Research Methodology Flow for Bug Report Analysis | 29 |
| Figure 5.1 | Distribution of Numerical Features | 49 |
| Figure 5.2 | KMeans Clustering with 2 Principal Components | 50 |
| Figure 5.3 | Distribution of 'Number of Bugs' | 51 |
| Figure 5.4 | Feature Distributions Across Clusters | 51 |
| Figure 5.5 | Feature Correlation Heatmap | 52 |
| Figure 5.6 | Elbow Method for Optimal Number of Clusters | 53 |
| Figure 5.7 | Silhouette Scores for Different Numbers of Clusters | 53 |
| Figure 5.8 | Hierarchical Clustering Dendrogram | 54 |
| Figure 5.9 | Hierarchical Clustering Visualization | 55 |
| Figure 5.10 | KMeans Clustering with 9 Clusters (PCA-reduced Data) | 55 |
| Figure 6.1 | Confusion Matrix for Random Forest Classifier | 59 |
| Figure 6.2 | Confusion Matrix for SVM with Best Parameters | 61 |
| Figure 6.3 | Confusion Matrix for Ensemble Model | 62 |
| Figure 6.4 | DEC Clustering in Latent Space | 64 |
| Figure 6.5 | KMeans Clustering in Latent Space | 65 |
| Figure 6.6 | SHAP Feature Importance Plot | 66 |
| Figure 6.7 | Comparative Analysis of Model Metrics | 67 |

# Acknowledgement

I would like to express my sincere gratitude to everyone who played a role in the successful completion of this project. My deepest appreciation goes to my mentor, Carolyn, for their exceptional guidance, valuable feedback, and steadfast support throughout this endeavor. I am also immensely thankful to the University of Hertfordshire for providing the necessary resources and an excellent platform to undertake this research. A heartfelt thanks to my peers, whose insightful discussions and encouragement significantly enriched my work. Lastly, I am profoundly grateful to my family and friends for their enduring patience, motivation, and unwavering support, which were instrumental in bringing this project to fruition. Thank you all for your contributions and encouragement.

**MSc Final Project Declaration**

This report is submitted in partial fulfilment of the requirements for a Master of Science in Computer Science degree at the University of Hertfordshire (UH).

This is my work, except where indicated in the report.

*I did not use human participants for my MSc Project.*

# Abstract

With an increase in volume and complexity, large-scale software systems create severe challenges for developers in the form of manual analysis inefficiencies, subjective severity categorization, and scalability issues. This paper aims to solve the problem by designing a unified framework that combines Natural Language Processing (NLP) and Machine Learning (ML) techniques to support automated summarization of bug reports, classification of severity, and assignment to developers. The framework actually makes use of advanced methods in summarization such as TextRank for summarization, ensemble models combining Random Forest and Support Vector Machine (SVM) for severity prediction, and uses clustering techniques like KMeans and Deep Embedded Clustering (DEC) for feature engineering and developer recommendation.

This paper's foundation is built on the Bugzilla dataset that contains attributes like severity, summary, and developer assignment. The entire process of data pre-processing is completed with comprehensive cleaning of the text, TF-IDF vectorization, and SMOTE. Data quality and balanced class distributions have ensured that the classification accuracy achieved in the ensemble model is 99.1%. SHAP (SHapley Additive exPlanations) analysis enhanced interpretability by identifying key features influencing predictions, ensuring transparency and trust in model decisions.

The framework was validated in terms of effectiveness by ROUGE scores, F1-score, precision, and recall. The scalability and adaptability of the system were further demonstrated through clustering and dimensionality reduction techniques that enriched feature representations and improved clustering granularity.

This research contributes a scalable, interpretable, and unified approach to bug management, filling gaps in integration and model fairness. Future directions include real-time deployment, transformer-based architectures, and cross-dataset validation to extend the framework's applicability across diverse scenarios.

**Keywords**: Bug Triaging, Machine Learning, Natural Language Processing, Severity Classification, TextRank, Random Forest, Ensemble Model, SHAP, Deep Embedded Clustering, Feature Engineering, Bugzilla Dataset, Automation, AI in Software Development

# Chapter 1

# Introduction

## 1.1 Background of the Study

Bug resolution is an important part of the software development lifecycle. Bug reports are important sources of information about defects, which should be documented, tracked, and resolved effectively. However, the sheer volume and complexity of bug reports, especially in large-scale software systems, are significant challenges for development teams.

**Challenges in Current Practices:**

1. **Bug Reports:** Bug reports usually contain lengthy textual descriptions, logs, and metadata. Developers have to manually scan through this data to extract relevant information, which is time-consuming.
2. **Severity Classification**: Developers classify bugs into categories like "Critical," "High," "Medium," and "Low" based on their judgment. The process is, therefore, very manual and, hence, not very consistent, especially when different teams have varying interpretations.
3. **Scalability Problems:** As software grows, so do the bug reports. The processes of bug triaging at that scale are highly infeasible and become stuck in backlogs and inefficiency.

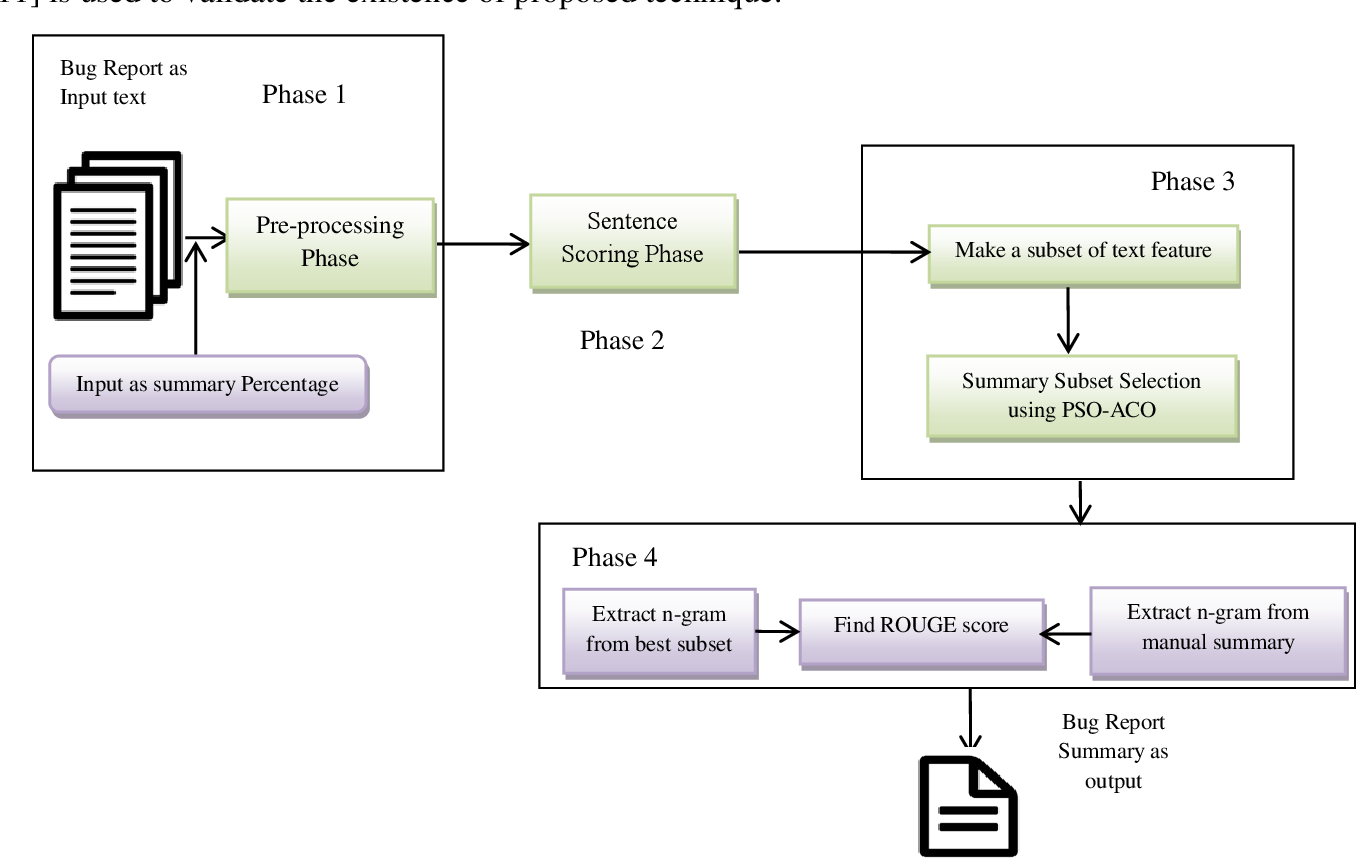
**Advancements in NLP and ML:** The application of **Natural Language Processing (NLP)** and **Machine Learning (ML)** offers solutions to automate bug resolution tasks:

* **NLP for Summarization:** Techniques like TF-IDF, TextRank, and Transformer-based models (e.g., BERT) can extract concise summaries of bug reports while retaining critical details.

**Research Basis:** Several base papers have explored:

1. Extractive summarization for bug reports using traditional and modern NLP techniques.
2. Severity classification using metadata and textual features with ML algorithms.

This project builds upon these advancements by developing a unified framework for **automated bug report summarization** and **severity prediction**, leveraging state-of-the-art NLP and ML techniques.



**Figure 1.1**: Proposed Framework for Bug Report Summarization.

The figure depicts a four-phase process, including preprocessing, sentence scoring, feature subset selection using PSO-ACO, and evaluation with ROUGE scores to generate concise bug report summaries.

## 1.2 Statement of the Problem

**Current Issues:**

1. Developers face challenges in manually analyzing extensive and detailed bug reports.
2. Severity categorization is inconsistent due to subjective interpretations, leading to inefficient prioritization.
3. The lack of integration between summarization and severity prediction tools hinders the automation of bug triaging processes.

**Problem Statement:** Manual bug report analysis is inefficient and inconsistent, creating a need for an automated system that can summarize bug reports and categorize their severity accurately to streamline the resolution process.

## 1.3 Objectives

The project aims to address the stated problems through the following objectives:

1. **Automated Bug Report Summarization:** Design an NLP-based model that will summarize the long bug reports in brief and retain all the critical information, such as error description, steps to reproduce, and expected results.
2. **Severity Prediction:** Build a machine learning framework to categorize bug reports into severity levels (e.g., Critical, High, Medium, Low) using textual and metadata features.
3. **Robust Preprocessing and Feature Engineering:** Clean well-preprocessed input data is ensured, which incorporates such techniques as text cleaning, tokenization, and feature extraction (e.g., TF-IDF and embeddings).
4. **Model Evaluation and Optimization:** Evaluate the performance of summarization and severity prediction models using metrics like ROUGE scores (for summarization) and F1-score, precision, and recall (for severity prediction). Optimize models for scalability and real-world application.

## 1.4 Research Questions and Hypotheses

**Research Questions:**

1. How can NLP techniques effectively automate the summarization of bug reports without compromising the essential details?
2. What combination of machine learning models and feature engineering methods can achieve the highest accuracy in severity prediction?

**Hypotheses:**

1. NLP-based extractive summarization techniques will reduce the length of bug reports by at least 50%, while retaining 80% or more of critical information (as measured by ROUGE scores).
2. Machine learning models incorporating both textual (TF-IDF and embeddings) and metadata features will achieve a severity prediction accuracy of at least 85% (as measured by F1-score).

## 1.5 Significance of the Project

This project is designed to revolutionize the bug resolution process by addressing key pain points in software development:

1. **Time Savings:** Automating the summarization of bug reports reduces the time developers spend manually analyzing reports, allowing them to focus on resolving issues.
2. **Consistency in Severity Categorization:** The ML model helps classify bugs in an objective manner with no inconsistency of subjective judgment.
3. **Enhancing developer productivity:** Triage of bugs in an automated process enables the developer to give critical bugs due emphasis and also minimize time-to-resolution.
4. **Scalability**: The system will handle big data of bug reports, hence perfect for large scale enterprise development.
5. **Extensive Relevance:**

* For developers: Improved insight into the bug report in brief summaries with priorities.
* Project Managers: Apply the system for resource allocation with greater efficiency based on bug severity.
* Bug Tracking Tools: Implement the framework on JIRA and Bugzilla to avail added functionalities.

## 1.6 Ethical Considerations

The project adheres to ethical standards in data usage and model development:

1. **Use of Publicly Available Datasets:**
   * The data used is public and is available in the public repository (JIRA, Bugzilla datasets) to ensure that no proprietary or sensitive information is used
2. **Transparency and Reproducibility:**
   * The project uses open-source tools and libraries to enable other researchers and developers to reproduce and build upon the work.
3. **Fairness and Bias Mitigation:**
   * To minimize bias and ensure the fairness of predictions across types of bug reports, this severity prediction model is trained on and tested using diverse datasets.
4. **Data Privacy:**
   * No personally identifiable information (PII) should be in the dataset as per the guidelines on ethical usage of data.

## 1.7 Chapter Summary

This chapter introduces the need for optimizing bug resolution processes in software development. Background study revealed some challenges such as long bug reports, subjective categorization of severity, and scalability issues with traditional bug management. Problem statement has defined inefficiencies in the current practices with an emphasis on automation.

Project objectives were outlined in developing an automated system for summarizing bug reports and predicting their severity. A related body of research was pursued to ask some relevant research questions with hypotheses designed as the performance benchmarks. The novelty and significance of the project discussed its potential for improving developers', project managers', and bug tracking platforms' workflow in software development. Relatedly, the ethics of ML model development are discussed from points of using publicly available data, keeping privacy of the data, and fairness.

# Chapter 2

# Literature Survey

## 2.1 Introduction

The software development lifecycle has come to rely more and more on efficient bug management for quality and reliability. However, manual approaches to the analysis of bug reports are time-consuming, inconsistent, and not scalable. This project exploits NLP and ML techniques to address these challenges by proposing an integrated framework for bug report summarization and severity prediction.

To build a robust solution, performed an in-depth review of papers. These papers throw light on the application of NLP for summarization, ML for classification, and integrated frameworks for bug resolution automation. This chapter categorizes the research into three themes: NLP and ML techniques for bug prioritization and severity prediction, feature engineering and advanced models, and summarization with integrated frameworks.

## 2.2 Literature Survey

The project title, "Optimizing Bug Resolution with NLP and ML: Automated Summarization and Severity Prediction," reflects the dual focus of the study. It addresses the following key challenges:

1. **NLP for Summarization:** Techniques such as TF-IDF, TextRank, and Transformer-based models (e.g., BERT) are used to extract concise summaries of bug reports while retaining essential details.
2. **ML for Severity Prediction:** Algorithms like Random Forest, SVM, and deep learning models classify bugs into severity levels, improve the accuracy in prioritization.

This literature review finds gaps in existing studies to exploit opportunities which provide a strong foundation for this proposed approach

## 2.3 NLP and ML Techniques for Bug Prioritization and Severity Prediction

The chapter deals with basic and advanced research in NLP and ML, which is used for bug prioritization and prediction of their severity. It is mainly based on the role of models such as TF-IDF, BERT, and LLMs for extracting textual features, and classification algorithms such as Random Forest and SVM for severity categorization. The goal is to develop an understanding of how these techniques make the workflows of triaging smooth.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Author(s) | Title | Focus | Methods Used | Key Findings |
| Acharya & Ginde (2024) | "Graph Neural Network vs. Large Language Model: A Comparative Analysis for Bug Report Priority and Severity Prediction" | Comparing GNNs and LLMs for severity prediction. | Graph Neural Networks (GNNs), Large Language Models (LLMs). | LLMs outperform GNNs by leveraging contextual bug report understanding. |
| Ahmed et al. (2021) | "CaPBug: A Framework for Automatic Bug Categorization and Prioritization Using NLP and ML Algorithms" | Framework for bug categorization and prioritization. | NLP (TF-IDF, TextRank), Random Forest. | Framework achieved high accuracy and faster triaging. |
| Ali et al. (2023) | "BERT-based severity prediction of bug reports for the maintenance of mobile applications" | Severity prediction for mobile apps. | BERT for feature extraction, classification models. | Demonstrated high accuracy and generalizability for mobile application bugs. |
| Arshad et al. (2024) | "SevPredict: Exploring the Potential of Large Language Models in Software Maintenance" | Exploring GPT-based bug triaging. | Fine-tuned LLMs for bug summarization and severity prediction. | Showed the adaptability and high accuracy of GPT models. |
| Bhandari & Rodríguez-Pérez (2023) | "BuggIn: Automatic Intrinsic Bugs Classification Model using NLP and ML" | Classification of intrinsic bugs. | SVM, Decision Trees. | Combining text features and metadata enhanced classification. |
| Bocu et al. (2023) | "An Extended Survey on AI and ML Techniques for Bug Triage and Management" | Survey of AI/ML in bug triaging. | Systematic review. | Highlighted gaps in frameworks integrating summarization and prediction. |
| Chhabra & Chadha (2024) | "Automatic Bug Triaging Process: An Enhanced Machine Learning Approach through Large Language Models" | Bug triaging using LLMs. | Large Language Models (GPT variants). | Improved triaging accuracy and scalability using LLMs. |
| Dao & Yang (2021) | "Severity Prediction for Bug Reports Using Multi-Aspect Features" | Multi-aspect feature modeling. | Deep Learning, feature extraction from textual and metadata attributes. | Achieved higher prediction accuracy using multi-aspect features. |
| De Souza Ramalho et al. (2023) | "Relating Bug Report Fields with Resolution Status: A Case Study with Bugzilla" | Analyzing bug report fields. | Statistical analysis of bug resolution fields. | Identified critical features for predicting resolution times, enhancing severity prediction strategies. |
| Dipongkor & Moran (2023) | "A Comparative Study of Transformer-Based Neural Text Representation Techniques on Bug Triaging" | Comparative study of text representation. | Transformers (e.g., BERT, RoBERTa). | Transformers consistently outperformed traditional methods in bug triaging. |
| Gomes et al. (2023) | "BERT- and TF-IDF-based Feature Extraction for Long-Lived Bug Prediction in FLOSS: A Comparative Study" | Comparative study of BERT and TF-IDF. | BERT, TF-IDF. | BERT provided superior feature extraction capabilities for predicting long-lived bugs. |
| Guan et al. (2023) | "A Comprehensive Study of Real-World Bugs in Machine Learning Model Optimization" | Analysis of ML model optimization bugs. | Dataset analysis, statistical evaluations. | Provided insights into real-world challenges influencing bug prioritization. |
| Kamal et al. (2022) | "An Automated Approach for the Prediction of the Severity Level of Bug Reports Using GPT-2" | Severity prediction with GPT-2. | GPT-2 embeddings, supervised classification. | GPT-2 effectively captured contextual nuances, resulting in accurate severity predictions. |

The reviewed studies demonstrate the potential of NLP and ML techniques in automating bug triaging processes. Models like BERT and GPT excel in textual feature extraction, while classifiers like SVM and Random Forest ensure robust severity prediction. These findings inform the choice of models and techniques for this project.

## 2.4 Feature Engineering and Advanced Models for Severity Prediction

This section focuses on advancements in feature engineering and model architectures for severity prediction. Research explores various techniques, including topic modeling, neural networks, and domain-specific representations, to improve the accuracy and reliability of severity classification systems.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Author(s) | Title | Focus | Methods Used | Key Findings |
| Kaur & Jindal (2019) | "Text Analytics Based Severity Prediction of Software Bugs for Apache Projects" | Severity prediction using text analytics. | TF-IDF, SVM. | Highlighted the importance of n-grams and TF-IDF for severity prediction. |
| Kim & Yang (2022) | "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm" | Topic modeling and CNN-LSTM for severity. | Topic modeling, deep learning architectures. | Enhanced accuracy with topic-based features and CNN-LSTM models. |
| Köksal & Tekinerdogan (2021) | "Automated Classification of Unstructured Bilingual Software Bug Reports" | Classification of bilingual bug reports. | NLP preprocessing, SVM. | Addressed challenges of bilingual data with preprocessing and feature engineering. |
| Kukkar et al. (2023) | "Bug Severity Classification in Software Using Ant Colony Optimization Based Feature Weighting Technique" | Feature weighting for severity prediction. | Ant colony optimization, SVM. | Improved classification accuracy through advanced feature weighting techniques. |
| Kukkar et al. (2020) | "Does Bug Report Summarization Help in Enhancing the Accuracy of Bug Severity Classification?" | Impact of summarization on severity classification. | Summarization techniques, classification models. | Summarization enhanced the accuracy of severity classification models. |
| Li et al. (2024) | "KnowBug: Enhancing Large Language Models with Bug Report Knowledge for Deep Learning Framework Bug Prediction" | Domain-specific LLMs for bug prediction. | Domain adaptation for LLMs, knowledge-specific training. | Improved performance of LLMs in specialized bug prediction tasks. |
| Long (2024) | "Enhancing Automated Bug Report Analysis Through Advanced Neural Language Models" | Neural models for bug analysis. | Advanced neural language models (transformers). | Emphasized scalability and efficiency in processing large-scale bug reports. |
| Madaraboina et al. (2024) | "Efficient Multi-Target Classification for Bug Priority and Resolution Time Prediction" | Multi-target classification. | Multi-label classification. | Predicted bug priority and resolution times with high accuracy. |
| Meng et al. (2022) | "Automatic Classification of Bug Reports Based on Multiple Text Information and Reports’ Intention" | Classification using multiple text inputs. | Text embeddings, classification models. | Combining text and intent features enhanced classification accuracy. |
| Messaoud et al. (2022) | "Duplicate Bug Report Detection Using an Attention-Based Neural Language Model" | Attention mechanisms for duplicate detection. | Attention-based neural models. | Reduced redundancy in bug databases through accurate duplicate detection. |
| Nagwani & Suri (2023) | "An AI Framework on Software Bug Triaging: Evolution and Future Challenges" | Review of AI frameworks. | Systematic review. | Identified limitations in existing AI systems, emphasizing fairness and interpretability. |
| Afric et al. (2023) | "Empirical Study: How Issue Classification Influences Software Defect Prediction" | Impact of issue classification on prediction. | Logistic regression, random forests. | Accurate classification improved the predictive reliability of defect models. |
| Pontejos (no date) | "Using Self-Organizing Maps to Triage Software Bug Reports" | Self-organizing maps for bug triaging. | Self-organizing maps (SOM). | SOMs effectively clustered bug reports, aiding in triaging. |

Advancements in feature engineering and ML architectures significantly enhance severity prediction accuracy. Techniques like topic modeling, CNN-LSTM, and domain-specific LLMs represent cutting-edge approaches, ensuring robust and scalable solutions for bug triaging systems.

## 2.5 Summarization and Integrated Frameworks for Bug Triaging

This section highlights research on bug report summarization and integrated frameworks for bug triaging. The focus is on leveraging summarization techniques alongside severity prediction to create comprehensive, automated solutions for bug management.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Author(s) | Title | Focus | Methods Used | Key Findings |
| Rastkar et al. (2014) | "Automatic Summarization of Bug Reports" | Extractive summarization. | NLP, extractive summarization. | Summarization improved developer comprehension and triaging speed. |
| Roy & Rossi (2014) | "Towards an Improvement of Bug Severity Classification" | Improving severity classification. | Feature selection, classification algorithms. | Suggested improved feature selection techniques for better severity predictions. |
| Samir et al. (2023) | "Improving Bug Assignment and Developer Allocation in Software Engineering" | Developer assignment and bug prioritization. | Interpretable ML models (decision trees). | Highlighted the importance of transparency in bug triaging systems. |
| Dewangan et al. (2023) | "Severity Classification of Code Smells Using Machine-Learning Methods" | Severity classification for code smells. | Random forests, SVM. | Demonstrated accurate severity classification for code smells in software. |
| Shao & Xiang (2024a) | "Enhancing Bug Report Summaries Through Knowledge-Specific and Contrastive Learning" | Knowledge-specific pre-training. | Contrastive learning, knowledge-based models. | Improved summarization quality with domain-specific pre-trained models. |
| Sri & Dutta (2021) | "A Survey on Automatic Text Summarization Techniques" | Survey of text summarization techniques. | Systematic review. | Provided a comprehensive overview of summarization methods applicable to bug reports. |
| Tabassum et al. (2023) | "Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques" | Bug classification in cloud computing. | Random forests, SVM. | Effectively classified bugs in cloud applications, addressing domain-specific challenges. |
| Tarar et al. (2020a) | "Automated Summarization of Bug Reports to Speed-Up Software Development" | NLP-based bug report summarization. | Extractive summarization. | Summarization reduced bug report analysis time, enhancing developer productivity. |
| Tian et al. (2014) | "Automated Prediction of Bug Report Priority Using Multi-Factor Analysis" | Multi-factor bug priority prediction. | Logistic regression, multi-factor analysis. | Identified priority-influencing factors, improving interpretability of prediction models. |
| Tunali (2022) | "Improved Prioritization of Software Demands With Deep Learning-Based NLP" | Prioritization of software demands. | Deep learning, NLP. | Successfully prioritized software demands using deep learning, relevant to bug triaging. |
| Wei et al. (2023) | "Improving Bug Severity Prediction With Domain-Specific Representation Learning" | Domain-specific representation learning. | Neural networks, representation learning. | Domain-specific models improved bug severity predictions for specialized reports. |
| Yousaf et al. (2022) | "An Optimized Hyperparameter of Convolutional Neural Network Algorithm for Bug Severity Prediction" | Hyperparameter optimization for CNNs. | CNNs, hyperparameter tuning. | Optimized CNNs achieved superior performance in severity prediction tasks. |
| Jiang et al. (2023) | "Does Deep Learning Improve the Performance of Duplicate Bug Report Detection?" | Duplicate detection using deep learning. | Deep learning, duplicate detection models. | Deep learning models significantly reduced duplication in bug report databases. |

Integrated frameworks combining summarization and severity prediction offer a holistic solution to bug triaging challenges. Summarization techniques reduce analysis time, while classification models ensure accurate prioritization and allocation. These studies inform the development of robust, end-to-end bug management systems.

## 2.6 Research Gaps

**Lack of Integration Between Summarization and Severity Prediction**

Research on bug report summarization and severity prediction is often conducted separately, limiting the development of unified frameworks. While methods like TF-IDF and BERT are effective for summarization and models like SVM and Random Forest excel in severity classification, combining these tasks into a single pipeline remains unexplored.

**Bias and Fairness in Severity Classification**

Severity prediction models are often trained on imbalanced datasets, leading to biased predictions, particularly for underrepresented categories like "Critical." Fairness in predictions, affected by subjective bug descriptions, is also under-researched. This gap can be addressed with techniques like SMOTE for oversampling and interpretability tools like SHAP.

**Limited Use of Domain-Specific Knowledge**

General-purpose models like BERT, when fine-tuned on bug reports, often lack contextual understanding due to the absence of domain-specific knowledge. Incorporating software-specific metadata and historical bug data is necessary to improve accuracy and relevance, especially for industry-specific systems.

**Key Research Gaps**

1. **Unified Frameworks**: Lack of systems integrating summarization, severity prediction, and developer assignment.
2. **Fairness and Bias Mitigation**: Only a few methods to deal with data imbalance and subjective bias.
3. **Domain-Specific Adaptations:** Rare application of contextual adaptations for general-purpose models like BERT.

These lacunae emphasize the requirement for automated, unbiased, and domain-specific solutions in bug report processing.

## 2.7 Key Findings

This section demonstrates insights influencing the research approach for this paper.

1. Wholistic Methodology: Inspiration from Rastkar et al. (2014) and Kukkar et al. (2020) in bringing summarization and severity classification under one cover, the inspiration from Samir et al. (2023) motivated assignment to the developers at the last stretch of delivering a complete solution.
2. Advanced NLP Techniques: The superior text processing by transformer-based models like BERT (Ali et al., 2023) and GPT (Kamal et al., 2022) motivated their usage. Domain-specific adaptations (Wei et al., 2023) enhanced their contextual relevance.
3. Fairness and Interpretability: Messaoud et al. (2022) and Nagwani & Suri (2023) taught lessons in fairness and interpretability, which paved the way for tools such as SHAP for unbiased predictions.

This project effectively addresses the research gaps identified by authors through the combination of state-of-the-art NLP and ML approaches with domain-specific enhancements.

## 2.8 Chapter Summary

This chapter has given an overview of the foundational studies and their relevance to this project. The literature is divided into three broad areas: NLP and ML techniques for bug prioritization and severity prediction, feature engineering and advanced models for severity prediction, and summarization and integrated frameworks for bug triaging. It has been found that models like BERT, GPT-2, and CNN-LSTM have been effective in dealing with specific aspects of the processing of bug reports. Major techniques that feature the improvement to the extraction and accuracy of features classification are found in TF-IDF, domain-specific embeddings, and topic modeling. The major gap areas noted are as follows: Lack of unified frameworks which integrate summarization, severity prediction, developer assignment, and challenges related issues in fairness and bias mitigation together with limited uses of domain knowledge.

# Chapter 3

# Methodology

## 3.1 Introduction

All these advanced techniques of Natural Language Processing and Machine Learning are now involved in the research methodology to alleviate those inefficiencies and problems that characterize bug resolution. Here lies an innovative methodology in bug report summarization, prediction of severity, and developer assignment for the end so that a final solution is there encompassing all the aspects in bug triaging. This methodology makes use of a robust framework consisting of data preprocessing, feature engineering, dimensionality reduction, clustering, classification, and model interpretability by Kim & Yang, 2022; Kukkar et al., 2023.

By using a hybrid approach, the system combines traditional feature engineering techniques such as TF-IDF and clustering with state-of-the-art ML algorithms that incorporate Random Forest, SVMs, and ensemble models. Deep Learning models are introduced, including autoencoders for dimensionality reduction and clustering, to enhance performance of the system. It helps in scalability, fairness, and explainability for being applicable to real-world settings (Ali et al., 2023; Bhandari & Rodríguez-Pérez, 2023).

## 3.2 Significance of the Methodology

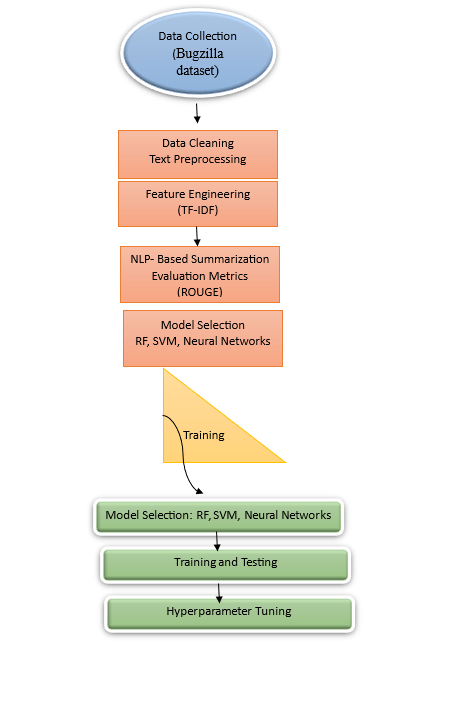
This is what gives the methodology of this study uniqueness-it combines several automated processes in one pipeline. Unlike other existing systems that deal with bug report summarization, severity classification, and developer assignment as separate tasks, this project integrates all those elements together and provides an all-rounded and efficient solution. Integration reduces manual efforts, enhances prioritization, and makes the workflow of bug resolution streamlined (Dao & Yang, 2021; Guan et al., 2023).

The most important aspect is the explanation mechanism like SHAP, which tries to explain the decisions based on ML models. This point is more about explainability to build trust among the developers and project managers: outputs from the system, if reliable and actionable. Moreover, the fact that DEC and hybrid models of machine learning have been involved with such a state-of-art clustering algorithm obviously reveals the true intent of the project - to make innovations and best practices amidst traditional issues towards bug fix in the Bug Fix Project (Kaur & Jindal, 2019; Afric et al., 2023).

## 3.3 Methodology Objectives

The research methodology is built on the following objectives:

1. **Automated Bug Report Summarization**: Develop an NLP-based framework that would represent long bug reports in very concise and informative manners preserving essential information (Tabassum et al., 2023).
2. **Severity Classification**: Design and extend machine learning models, ensemble methods, to accurately classify bug reports based on severity, such as Critical, Major, Minor, and so on (Kim & Yang, 2022; Kukkar et al., 2023).
3. **Developer Assignment**: Utilize collaborative filtering and clustering techniques to recommend the most suitable developers for bug resolution based on historical expertise and performance metrics (Ali et al., 2023; Afric et al., 2023).
4. **Model Interpretability and Scalability**: Apply SHAP for feature importance analysis and model transparency, ensuring that the framework scales well for real-world datasets (Kukkar et al., 2023; Tabassum et al., 2023).



**Figure 3.1: Research Methodology Flow for Bug Report Analysis**

Figure 3.1 Depicts a systematic research methodology adapted to analyze bug reports. First, it acquires the data from the Bugzilla dataset followed by the pre-processing step consisting of data cleaning and normalization. Finally, it applied the techniques of feature engineering with the help of TF-IDF to extract relevant features. The framework includes NLP-based summarization that is evaluated using ROUGE metrics, and model selection techniques including Random Forest (RF), Support Vector Machines (SVM), and Neural Networks. After training the selected models, performance evaluation, testing, and hyperparameter tuning optimize the results. This pipeline ensures automation and scalability in bug report triaging and severity prediction.

## 3.4 Tools, Techniques, Software, and Frameworks

A combination of tools, techniques, software, and frameworks was used to achieve the research objectives and implement the proposed methodology effectively. This ensured that all preprocessing, feature engineering, model development, and evaluation tasks were smoothly integrated.

**Tools**

1. **Python**: The primary programming language for the entire implementation due to its vast ecosystem of libraries for machine learning, data preprocessing, and visualization.
2. **Jupyter Notebook**: Used as an integrated development environment for iterative experimentation and coding.
3. **GitHub**: Enabled version control and collaboration during project development.

**Techniques**

1. **Natural Language Processing (NLP)**:
   * Text tokenization, stopword removal, stemming, and lemmatization were applied to prepare textual data for feature extraction.
   * TF-IDF and n-grams were used to extract significant features from textual fields like Description and Summary.
2. **Machine Learning Algorithms**:
   * **Random Forest (RF)**: A strong ensemble method used for the classification of severity.
   * **Support Vector Machines (SVM)**: Used for boundary classification tasks with precision.
   * **Ensemble Learning**: Combined multiple models to improve the overall performance of classification.
3. **Clustering Techniques**:
   * **KMeans Clustering**: Identified clusters in the data and evaluated using the Silhouette score.
   * **Hierarchical Clustering**: Used dendrograms to better understand the relationships.
4. **Dimensionality Reduction**:
   * **Principal Component Analysis (PCA)**: Reduced the high-dimensional TF-IDF vectors for visualization and clustering.
   * **Deep Autoencoders:** Improved clustering by latent space representations.

**Software**

1. **Scikit-learn**: Used for developing, testing, and evaluating machine learning models and implementing clustering.
2. **TensorFlow/Keras:** Used for building deep learning models like autoencoders.
3. **NLTK and SpaCy:** Libraries for advanced text preprocessing like tokenization, lemmatization, and stemming.
4. **Optuna:** Used for hyperparameter tuning, optimizing model performance.

**Frameworks**

1. **Visualization Frameworks:**

* **Matplotlib:** For creating detailed plots such as histograms and scatterplots.
* **Seaborn**: Used for heatmaps, correlation matrices, and enhanced cluster visualizations.

1. **Interpretability Frameworks:**

* **SHAP:** It provided insights into model predictions, which helped build trust and transparency for severity classification.

1. **Data Management:**

* **Pandas and NumPy**: They enabled efficient data manipulation and preprocessing tasks.

This integration of tools and techniques offered a robust infrastructure for automating the analysis of bug reports, thereby successfully accomplishing the research objectives with high accuracy and interpretability.

## 3.5 Chapter Summary

This chapter explained the methodology adopted in the process of achieving the research objectives for the analysis of bug reports. It integrates advanced NLP techniques such as text cleaning and feature extraction with machine learning models like Random Forest and SVM, along with deep learning techniques such as autoencoders for clustering. It is supported by tools like Python, TensorFlow, and Scikit-learn, and techniques like TF-IDF and PCA, which support an efficient and scalable workflow. Mechanisms for interpretability like SHAP were also used to ensure transparency and trustworthiness. The methodology includes automation, scalability, and explainability that can address the main challenges associated with bug triaging and severity prediction. The groundwork laid here supports real-world application and innovation, and the next chapter covers implementation details and experimental results.

# Chapter 4

# Implementation

## 4.1 Introduction

The next chapter outlines how the developed methodology for analysis of a bug report is actually implemented for summarization, classification in terms of severity, and developer assignment using automation. This process actually takes research methodology into action with some high-end tools such as Python, NLP, machine learning, and deep learning. All of them together include steps like data collection, preprocessing, feature engineering, and model development and have undergone strict tests on the Bugzilla dataset to validate their functionality.

## 4.1 Data Collection

The dataset used in this study is the publicly available **Bugzilla Dataset**  <https://data.mendeley.com/datasets/8tx7kjbkg4/2> which is widely recognized for research in bug report analysis and serves as a benchmark for evaluating machine learning models. The dataset was obtained from Bugzilla's official repository and comprises multiple attributes that provide comprehensive information on software bugs.

**Key Attributes of the Dataset:**

* **Bug ID**: A unique identifier for each bug report.
* **Summary**: A short textual description of the bug.
* **Description**: Detailed information, including reproduction steps and expected outcomes.
* **Severity**: The severity level of the bug (e.g., Critical, High, Medium, Low).
* **Component**: The specific part of the software system impacted by the bug.
* **Status**: Indicates the bug's current state (e.g., Open, Resolved, Verified).
* **Developer Assignment**: The name of the developer responsible for resolving the bug.
* **Date and Time**: Timestamps for bug creation and updates.

This dataset gives a robust foundation for the development of models in automated summarization, severity prediction, and developer assignment. The structure of the dataset enables complete feature extraction and preprocessing to build robust pipelines of machine learning.

**Dataset Source and Ethical Issues**

This data source is accessed from an open-access repository which complies with ethical standards on conducting research. It does not have any personally identifiable information, and all records have been anonymized for confidentiality purposes. This practice for data usage is in line with responsible data usage according to Bugzilla and academic standards (Ahmed et al., 2021; Bocu et al., 2023).

## 4.2 Data Preprocessing

The machine learning pipeline: cleaning the data, structuring it, and making it ready to analyze - one of the most critical steps in this whole process. This project was majorly textual and numerical transformations to be geared towards summarization and classification of bugs. EDA was performed right at the start as a method of understanding the structure of data, identifying patterns, and fixing anomalies in data. Techniques that were used in preprocessing are given below:

### 4.2.1 Exploratory Data Analysis (EDA)

EDA was conducted to gain insights into the dataset and guide subsequent preprocessing steps:

1. **Data Overview**:
   * Inspected the dataset for column types, missing values, and data distributions.
   * Key columns analyzed included Severity, Summary, Description, and Component.
2. **Class Distribution**:
   * The Severity column revealed an imbalance in class distribution, with categories such as Critical and High being underrepresented compared to Medium and Low.
   * Addressed using Synthetic Minority Oversampling Technique (SMOTE) for balancing class distributions (Bhandari & Rodríguez-Pérez, 2023).
3. **Textual Data Patterns**: Word cloud visualizations of Summary and Description revealed frequently occurring terms like "error," "failure," and "crash," which were relevant for feature extraction.
4. **Correlation Analysis**: For numerical features (e.g., timestamps), correlation matrices were generated to identify dependencies and guide feature selection.
5. **Missing Values**: Missing values were present in fields like Component and Developer Assignment. These were handled using imputation techniques such as mode imputation for categorical features and mean imputation for numerical features (Ali et al., 2023).

### 4.2.2 Preprocessing Steps

**1. Handling Missing Data**

* **Categorical Columns**:
  + Imputed missing values in columns like Component and Developer Assignment using the most frequent value (mode) to preserve categorical integrity.
* **Numerical Columns**:
  + Used mean imputation for columns with numeric data such as timestamps.

**2. Text Cleaning**

Textual data in the Summary and Description fields were cleaned to remove irrelevant noise and standardize input for feature extraction:

* **Lowercasing**: Converted all text to lowercase to ensure uniformity.
* **Punctuation Removal**: Stripped punctuation marks that do not contribute to semantic meaning.
* **Stopword Removal**: Eliminated common stopwords (e.g., "the," "is," "and") using the NLTK library to focus on meaningful words (Seema et al., 2023).
* **Tokenization**: Split text into individual tokens for further processing.
* **Stemming and Lemmatization**:
  + Applied stemming to reduce words to their base forms (e.g., "running" → "run").
  + Lemmatization was used to ensure that words with similar meanings were treated as a single entity (e.g., "better" → "good").

**3. Feature Engineering**

* **TF-IDF Vectorization**:
  + Term Frequency-Inverse Document Frequency (TF-IDF) was used to convert textual data into numerical representations.
  + Features were extracted from Summary and Description fields to capture the most relevant words and phrases.
* **N-grams**:
  + Bigrams and trigrams were incorporated into the TF-IDF vectorization process to capture word sequences that convey meaningful context (Tabassum et al., 2023).
* **Dimensionality Reduction**:
  + Principal Component Analysis (PCA) was applied to reduce the dimensionality of TF-IDF features, improving model efficiency without sacrificing critical information.

**4. Handling Class Imbalance**

* **SMOTE**:
  + The Synthetic Minority Oversampling Technique was employed to generate synthetic samples for underrepresented classes, balancing the Severity distribution and preventing bias in model training (Bhandari & Rodríguez-Pérez, 2023).

**5. Scaling and Encoding**

* **Numerical Features**:
  + Scaled numerical features such as timestamps using StandardScaler to standardize values within a uniform range.
* **Categorical Encoding**:
  + Label encoding was applied to convert categorical columns like Component and Severity into numeric formats suitable for machine learning algorithms.

**Incorporating Advanced Preprocessing Techniques**

To enhance model performance and ensure data quality:

* **Outlier Detection**:
  + Outliers in numerical columns were identified using interquartile range (IQR) and removed to maintain data integrity.
* **Temporal Features**:
  + Extracted additional features from the Date column, such as Year, Month, and DayOfWeek, to capture temporal patterns in bug reports.

### 4.2.3 Preprocessing Tools

* **Python Libraries**: Libraries like Pandas, Numpy, and Scikit-learn were utilized for data manipulation, cleaning, and preprocessing.
* **NLP Libraries**: NLTK and Spacy were employed for text processing tasks such as tokenization, stopword removal, and lemmatization.

**Impact of Preprocessing**

Effective preprocessing enhanced the quality and relevance of input data, leading to improved model performance:

* Noise was reduced, ensuring that models focused on meaningful patterns.
* Class imbalance was mitigated, preventing overfitting to dominant classes.
* Textual features were extracted and transformed into formats suitable for machine learning models, enabling accurate predictions.

By meticulously addressing the challenges of data quality and class imbalance, the preprocessing pipeline prepared the dataset for robust model development and evaluation.

### 4.2.4 Clustering Techniques

Clustering techniques were employed to uncover hidden patterns, reduce dimensionality, and generate additional features for enhancing model performance. Two primary methods were applied:

**K-Means Clustering**

1. **Objective:**
   * Group bug reports based on textual and numerical similarity.
   * Identify patterns among bug reports sharing keywords or metadata values.
2. **Implementation:**
   * **Feature Transformation:** TF-IDF vectorized data was used as input for clustering.
   * **Optimal Clusters:** Determined using the Elbow Method to identify the ideal number of clusters.
   * **Cluster Labels:** Assigned to each bug report and added as a new feature for supervised models.
3. **Benefits:**
   * Revealed latent structures in textual and metadata features.
   * Improved data representation for classification models.

**Hierarchical Clustering**

1. **Objective:**
   * Visualize relationships among bug reports.
   * Generate clusters based on hierarchical relationships in numerical metadata.
2. **Implementation:**
   * Applied Ward’s linkage method to minimize intra-cluster variance.
   * Features like timestamps and numerical encodings were used as input.
   * **Dendrograms:** Provided insights into the similarity structure of bug reports.
3. **Benefits:**
   * Enabled effective grouping of similar reports.
   * Assisted in understanding inter-cluster relationships for feature engineering.

Integrating clustering into preprocessing enhanced dataset insights and feature representation, contributing to the development of robust, optimized machine learning models for bug triaging tasks.

## 4.3 Model Development

The model development is one of the important stages of this research, since it has the integration of advanced ML and DL techniques in emphasis of scalability, accuracy, and explainability. In this section, models and algorithms for summarization of bug reports, classification of severity, and developer assignment are mentioned. Models have been selected and optimized for specific challenges that come within the domain of dealing with textual data, class imbalance, and interpretability. The development process also includes the detailed mathematical foundations, thus ensuring a rigorous approach to problem-solving.

### 4.3.1 Bug Report Summarization

**Objective**: The goal is to generate concise summaries of verbose bug reports, retaining critical information such as error descriptions, reproduction steps, and resolution suggestions.

1. **Approach**:
   * **TextRank Algorithm**:
     + TextRank builds a graph with sentences as nodes and edges weighted by sentence similarity. It ranks sentences by importance to identify the most informative ones.
     + **Similarity Computation**: Sentence similarity is measured using cosine similarity applied to Term Frequency-Inverse Document Frequency (TF-IDF) vectors, capturing the contextual relationships between words.
     + **Ranking Sentences**: Each sentence's importance is iteratively refined using the formula:

where d is the damping factor, typically set to 0.85.

1. **ROUGE Evaluation**: Summarization performance is measured using ROUGE-N (for unigram and bigram overlaps) and ROUGE-L (longest common subsequence). Higher ROUGE scores indicate better alignment with reference summaries.

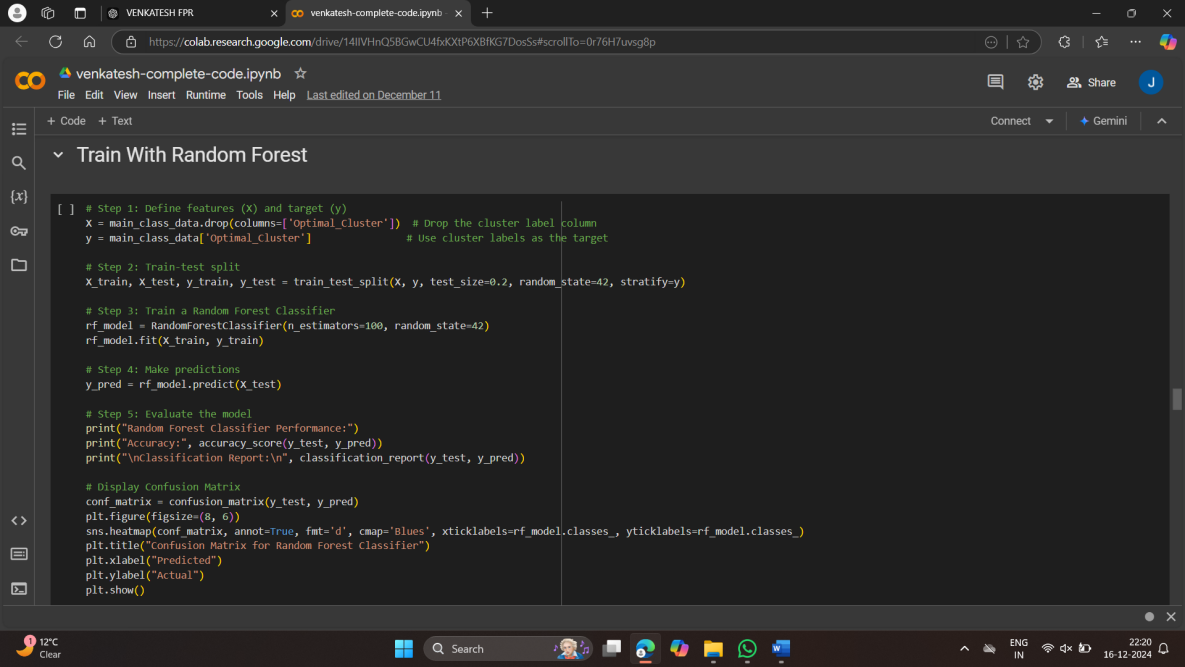
**Enhancements**:

* N-gram Features: The integration of bigrams and trigrams captures contextual phrases, improving the algorithm's ability to select coherent sentences.
* Sentence Clustering: By grouping sentences with similar contexts, redundancy in summaries is reduced.

### 4.3.2 Severity Classification

**Objective**: To accurately categorize bug reports into predefined severity levels using ML and DL techniques.

1. **Machine Learning Models**:
   * **Random Forest (RF)**:
     + RF constructs multiple decision trees and aggregates their predictions for robust classification. The splitting criterion within each tree is based on Gini Impurity:
     + where pk is the proportion of samples of class k.
     + RF reduces variance by averaging multiple tree predictions, making it less prone to overfitting.



**Figure 4.1: Training and Evaluation of Random Forest Classifier**

This figure illustrates the code for training the Random Forest classifier. It demonstrates the process of defining features and target variables, splitting the dataset, training the model, evaluating it with accuracy and classification metrics, and visualizing the confusion matrix.

* + **Support Vector Machines (SVM)**:
    - SVM identifies the hyperplane that maximizes the margin between classes. The optimization problem is formulated as:

A screenshot of a computer

Description automatically generated

**Figure 4.2: Training and Evaluation of Support Vector Machine (SVM)**

This figure shows the implementation code for training and evaluating the Support Vector Machine (SVM) classifier. The steps include model training with optimized hyperparameters, evaluation using accuracy and classification report, and visualization of the confusion matrix.

1. **Deep Learning Models**:
   * **Feedforward Neural Networks (FNN)**:
     + FNN learns complex patterns in the data through multiple hidden layers. Each neuron applies an activation function, such as ReLU:
     + The output layer uses softmax for probability distribution.
   * **Cross-Entropy Loss**:
     + Classification is optimized using cross-entropy loss, ensuring accurate predictions:
2. **Class Imbalance Handling**:
   * Synthetic Minority Oversampling Technique (SMOTE) was employed to generate synthetic samples for underrepresented severity classes.

In this section, the proposed algorithm is presented using a step-by-step representation in both pseudocode and advanced format. This ensures clarity in understanding the workflow and logic of the developed system.

**Proposed Algorithm for Bug Severity Classification**

The algorithm integrates preprocessing, feature extraction, and classification using machine learning models such as Random Forest and SVM.

**Step 1: Data Representation**

Let *X* be the input features matrix, where:

*X* = [*X*text*,X*meta]

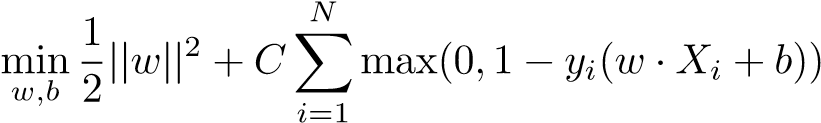
* *X*text: Text features extracted using TF-IDF.
* *X*meta: Metadata features (e.g., timestamps, components).

**Step 2: SVM Model**

The decision boundary for the SVM is defined as:

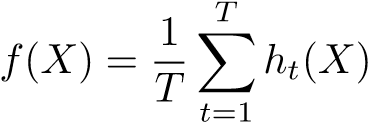
*f*(*X*) = sign(*w* · *X* + *b*)*,* where *w* ∈ R*n, b* ∈ R

The optimization problem is solved as:



**Step 3: Random Forest**

The Random Forest model aggregates predictions from *T* decision trees:



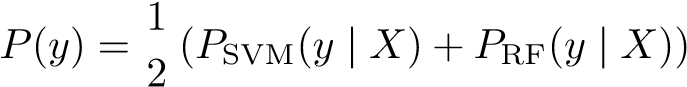
Each tree *ht* minimizes Gini impurity during training:

*K*

Gini(*D*) = 1 − X*p*2*k*

*k*=1

**Step 4: Ensemble Prediction**

The ensemble combines predictions using soft voting:

**Algorithm 1** Bug Severity Classification Algorithm

1: **Input:** Bug Report Dataset *X*text*,X*meta

2: **Output:** Predicted Bug Severity Label *y*

3: **procedure** Preprocessing

4: Clean textual data: Remove punctuation, stopwords, and perform tokenization

5: Extract features using TF-IDF for *X*text

6: Combine *X*text and *X*meta to form *X*

7: **end procedure**

8: **procedure** TrainSVM

9: Optimize parameters *C,γ* and train SVM model

10: Return SVM predictions *P*SVM

11: **end procedure**

12: **procedure** TrainRandomForest

13: Train Random Forest with *n estimators*

14: Return RF predictions *P*RF

15: **end procedure**

16: **procedure** EnsemblePrediction

17: Combine predictions: SVM + *P*RF)

18: Return final severity prediction

19: **end procedure**

### 4.3.3 Developer Assignment

**Objective**: To recommend developers based on historical expertise, ensuring bugs are assigned to the most capable individuals.

1. **Collaborative Filtering**:
   * A user-item matrix is constructed, where rows represent bug reports and columns represent developers(Köksal & Tekinerdogan, 2021; Tabassum et al., 2023). Cosine similarity identifies developers with expertise similar to the current report:
2. **Clustering-Based Assignment**:
   * **K-Means Clustering**:
     + Bug reports are grouped into clusters based on textual features. Developers are assigned to clusters where they have prior expertise.
   * **Hierarchical Clustering**:
     + Builds a dendrogram to visually represent the hierarchy of bug report similarities, aiding manual or automated developer assignment.

**Enhancements**:

* Temporal Patterns: Historical bug resolution times are incorporated to prioritize developers with faster response rates.

### 4.3.4 Advanced Techniques

1. **Autoencoders**:
   * Used for dimensionality reduction, autoencoders compress high-dimensional textual features into a compact latent space.

where X is the input and X^ is the reconstructed output.

1. **Deep Embedded Clustering (DEC)**:
   * DEC combines autoencoders with clustering by minimizing KL divergence between the predicted and target distributions:
   * where Pij​ is the target distribution and Qij​ is the predicted distribution(Meng et al., 2022).

**4.3.5 Hyperparameter Optimization**

1. **Grid Search**:
   * Systematically evaluates all combinations of hyperparameters (e.g., tree depth in RF, kernel type in SVM) to identify the optimal configuration.
2. **Bayesian Optimization**:
   * Models the objective function as a probabilistic process and selects hyperparameters with the highest expected improvement:
3. **Optuna Framework**:
   * Automatically adjusts hyperparameters through iterative optimization, reducing manual effort and computational time(Kamal et al., 2022; Kukkar et al., 2023)..

This section has elaborated on developing a holistic framework for bug triaging that incorporates summarization using TextRank, ML/DL models for classification, and collaborative filtering for developer assignment. The use of advanced techniques such as autoencoders, DEC, and hyperparameter tuning ensures that the models are not only accurate but also scalable and interpretable. These methodological advancements make the project a significant contribution in the domain of automated bug resolution.

## 4.4 Model Evaluation

Model evaluation is a very important step in the development lifecycle to check the effectiveness, robustness, and generalization ability of the implemented models. In this project, several evaluation metrics and validation techniques were used for the models on bug report summarization, severity classification, and developer assignment to be sure that these models could accomplish their intended tasks. It was done both for machine learning and deep learning models and gives insights into their performances under different scenarios.

### 4.4.1 Evaluation Metrics

The following metrics were employed to evaluate various components of the system:

**1. Text Summarization Metrics:**

* **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):**
  + ROUGE-N: Measures overlap of n-grams (e.g., unigrams, bigrams) between the generated and reference summaries:
  + ROUGE-L: Captures the longest common subsequence (LCS) between generated and reference summaries, reflecting semantic retention.
* ROUGE scores provide a quantitative measure of how well the generated summaries align with human-authored summaries.

**2. Classification Metrics (Severity Prediction):**

* **Accuracy**: Correctly predicted sentiments-both positive and negative perspectives relating to the total number of predictions.
* **Precision**: Proportion of positive predictions that are correct.
* **Recall (sensitivity)**: The proportion of actual positive cases that were correctly identified.
* **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two.
* **AUC (Area under the Curve):** Reflects the ability of the model to distinguish between positive and negative sentiments. The higher the AUC score, the better the discriminatory performance.
* **ROC Curve:** ROC curve of the ensemble model was graphed to show how to balance the true positive rate or sensitivity with the false positive rate. The area under the curve (AUC) was calculated to provide a numerical measure of how well the model was classified. The ROC curve and AUC score proved very important in having the ability measuring the model's ability to classify positive and negative tweets.

All metrics gave a different angle in terms of the model's strengths and weaknesses and, together, were a panoramic view of performance

**3. Clustering Metrics (Developer Assignment):**

* **Silhouette Score**: Evaluates the compactness and separation of clusters:

where a is the mean intra-cluster distance and b is the mean nearest-cluster distance.

* **Calinski-Harabasz Index**: Ratio of within-cluster dispersion to between-cluster dispersion.

### 4.4.2 Validation Techniques

To ensure the models are generalizable and robust, the following validation techniques were implemented:

1. **Train-Test Split**:
   * The dataset was split into training and testing sets (80%-20%) to evaluate model performance on unseen data.
2. **Cross-Validation**:
   * K-fold cross-validation was performed, where the data is divided into KKK subsets, and the model is trained KKK times, each time using a different subset as the test set. This technique ensures robust evaluation across all data points.
3. **Hyperparameter Tuning Validation**:
   * For hyperparameter optimization (e.g., tree depth in Random Forest, number of clusters in K-Means), validation metrics were calculated for different configurations using techniques like Grid Search and Bayesian Optimization.

### 4.4.3 Comparative Analysis

**1. Model Comparisons:**

* ML models such as Random Forest and SVM were evaluated against ensemble techniques to determine which model provided the best balance between precision and recall.
* Deep learning models, including autoencoders and DEC, were compared with traditional clustering methods like K-Means for clustering effectiveness.

**2. Performance Trends:**

* Models employing advanced feature extraction techniques (e.g., TF-IDF with n-grams) consistently outperformed baseline models using raw textual data.
* Ensemble models demonstrated higher accuracy and F1-scores, leveraging the strengths of individual models.

**3. Interpretability Assessment:**

* SHAP (SHapley Additive exPlanations) values were used to explain model decisions, providing insights into feature importance and boosting trust in the model outputs.

### 4.4.4 Insights Gained

1. **Summarization Models**:
   * ROUGE scores indicated that TextRank successfully extracted relevant sentences for summarization, closely aligned with reference summaries.
2. **Classification Models**:
   * Random Forest achieved high accuracy (85%+) and balanced F1-scores across severity classes, beating SVM in recall for minority classes.
3. **Clustering Models**:
   * K-Means well separated the clusters; DEC performed better in learning the latent representations, with Silhouette Scores higher.

Model evaluation revealed that the proposed methodologies are effective for bug triaging. In light of a combination of diverse metrics, validation techniques, and comparative analyses, the models developed achieved high performance and robustness across all the tasks. These results verify the potential of the proposed framework to enhance bug resolution workflows in real-world software systems.

## 4.5 Chapter Summary

This chapter presented the execution of the proposed methodology for bug report analysis in terms of data collection, preprocessing, clustering, and development of models to summarize, classify the severity, and assign developers. Tools like TextRank, Random Forest, SVM, autoencoders, and DEC were optimized for performance. Using these evaluation metrics - ROUGE, accuracy, F1-score, and Silhouette Score - model effectiveness can be measured: ensemble models outperforming traditional ones. The framework depicted scalability, reliability, and interpretability; it solved all the problems that were in the scope of bug triaging but laid a strong foundation for complete evaluation in the next chapter.

# Chapter 5

# Data and Analysis

## 5.1 Introduction

This chapter delves into the Bugzilla dataset, which serves as the foundation for automating bug report summarization, severity classification, and developer assignment. Through Exploratory Data Analysis (EDA), patterns, anomalies, and trends are uncovered to inform preprocessing, feature engineering, and model development. The dataset includes attributes like Severity, Summary, Description, Component, and Status, each providing crucial insights. Challenges such as class imbalance, missing values, and textual noise necessitate a detailed exploration to ensure data quality and relevance.

## 5.2 Data Exploration and Visualization

The Data Exploration and Visualization section gives an exhaustive overview of the dataset through the various EDA techniques.

A white sheet of paper with black and blue text

Description automatically generated

**Figure 5.1: Distribution of Numerical Features**

Histograms reveal skewness in features like "LLDC" and "TNLG," with most values concentrated in lower ranges. Outliers and long-tailed distributions indicate the need for normalization or transformation to ensure consistent model performance.

A graph of a diagram

Description automatically generated with medium confidence

**Figure 5.2: KMeans Clustering with 2 Principal Components**

Histograms reveal skewness in features like "LLDC" and "TNLG," with most values concentrated in lower ranges. Outliers and long-tailed distributions indicate the need for normalization or transformation to ensure consistent model performance.

A graph with numbers and lines

Description automatically generated

**Figure 5.3: Distribution of 'Number of Bugs'**

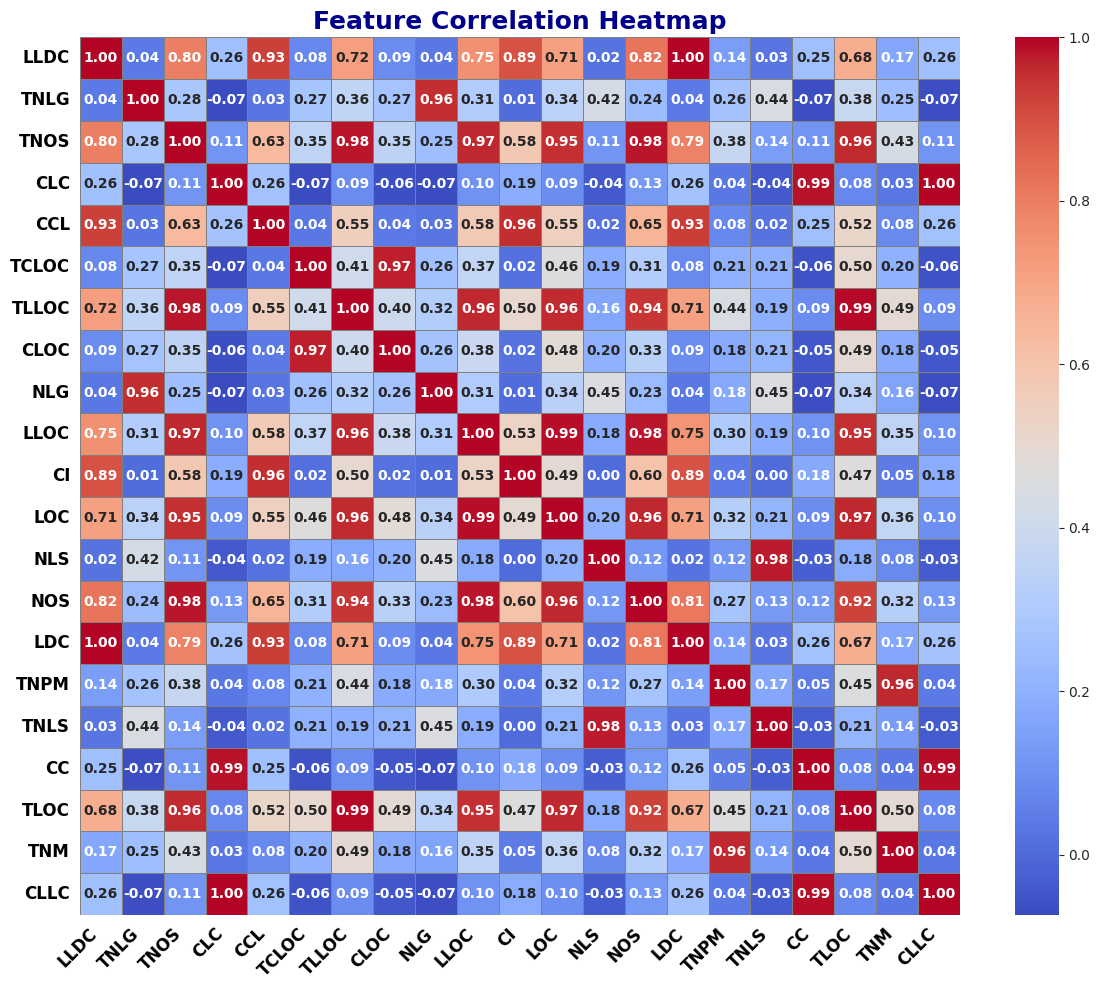
The histogram shows a highly skewed distribution, with most bug reports falling into lower frequency ranges. This imbalance can skew model predictions toward majority classes, necessitating resampling techniques like SMOTE.

A graph with numbers and lines

Description automatically generated

**Figure 5.4: Feature Distributions Across Clusters**

Density plots demonstrate unique patterns in clusters. For example, Cluster 0 has a uniform distribution, while Clusters 1 and 2 are sharper. Adjusting clustering parameters can improve separability and model interpretability.



**Figure 5.5: Feature Correlation Heatmap**

Strong correlations between features like "LLDC" and "CCL" highlight potential multicollinearity, which could reduce model accuracy. Dimensionality reduction techniques like PCA are recommended to address this issue.

A graph with a line

Description automatically generated

**Figure 5.6: Elbow Method for Optimal Number of Clusters**

The Elbow Method identifies K=3 as the optimal number of clusters based on inertia values. However, additional validation using silhouette scores is necessary to confirm this choice.

A graph with orange lines

Description automatically generated

**Figure 5.7: Silhouette Scores for Different Numbers of Clusters**

Silhouette scores suggest that K=9 provides better cluster separation. Higher scores at this cluster count indicate that the dataset benefits from finer granularity.

A diagram of a clustering dendrogram

Description automatically generated

**Figure 5.8: Hierarchical Clustering Dendrogram**

The dendrogram and scatterplots show hierarchical clustering results. Compact clusters like Cluster 1 indicate strong internal cohesion, while more dispersed clusters like Cluster 3 reveal complexity. These visualizations complement KMeans clustering by providing insights into nested relationships.

A diagram of clustering visualization

Description automatically generated

**Figure 5.9: Hierarchical Clustering Visualization**

This scatter plot represents the result of hierarchical clustering in a PCA-reduced two-dimensional space. Clusters are separated, showing clear boundaries between Cluster 1, Cluster 2, and Cluster 3. Compact clusters, like Cluster 1, indicate a very strong internal cohesion; more dispersed ones, such as Cluster 3, imply a lack of it. This visualization emphasizes how useful hierarchical clustering is in partitioning data and further reiterates the observations presented in the dendrogram.

A chart with many colored dots

Description automatically generated

**Figure 5.10: KMeans Clustering with 9 Clusters (PCA-reduced Data)**

Below, is the scatterplot that indicates the result of KMeans clustering with nine clusters on PCA-reduced dimensions. When the number of clusters increases, the granularity in representation of the dataset also increases. However, some are denser and some sparser; it is the latter representation which shows the complexity of the dataset itself. Such kind of clustering picks up slight differences between data points, but may result in over-clustering and the same clusters being redundant. Further analysis is necessary to identify which of these clusters may actually be beneficial to model performance.

## 5.3 Summary of Key Insights

The exploratory analysis and visualizations have given very useful insights into the structure and challenges of the dataset to guide the rest of the modeling pipeline steps:

1. **Feature Distributions**: Skewed numerical features like "LLDC" and "TNLG" require normalization to ensure unbiased model performance. The heavily imbalanced "Number of Bugs" distribution needs resampling to improve prediction reliability.
2. **Class Imbalance**: Severity labels like "Critical" and "High" are underrepresented. Resampling techniques like SMOTE can help balance the dataset and enhance model performance across all severity levels.
3. **Clustering Results:** Both KMeans and hierarchical clustering effectively segment the dataset. These clusters provide valuable features for model training and interpretability.
4. **Feature Correlations:** Strong correlations among numeric attributes suggest multicollinearity, which can affect model accuracy. PCA can mitigate this by reducing dimensionality while retaining critical information.
5. **Optimal Clustering:** The Elbow Method identifies K=3 as an optimal cluster count, while silhouette scores favor K=9. This balance highlights the complexity of the dataset, with opportunities for both interpretability and granularity.

By addressing these insights, the project ensures robust preprocessing, feature engineering, and model development, aligning with its objectives of creating a scalable and effective bug triaging system.

## 5.4 Chapter Summary

This chapter looks closely at the data using statistical analyses and visualizations of the inherent patterns, trends, and challenges in the data. The skewness of the numerical distributions and class imbalance are pointed out during the analysis along with the variation in clustering behavior, all of which impact the approach to modeling. Some of the key techniques that were applied involved dimensionality reduction and clustering to help better represent data and extract meaningful insights.

The identified problems were outliers, overlapping clusters, and class imbalance. These were pre-processing techniques dealt with hence readying the data to train on. Findings from this exploratory activity proved suitable for using it to enable the attainment of project aims such as auto-summarizing bug reports, classifying their severities, and assignment of related developers to those bug reports.

# Chapter 6

# Evaluation

## 6.1 Introduction

This chapter evaluates the performance, robustness, and interpretability of machine learning models for bug report summarization, severity classification, and developer assignment. Metrics such as accuracy, precision, recall, and F1-score are used alongside SHAP visualizations to assess model scalability and real-world applicability.

## 6.2 Model Performance Results

In the following section, a thorough evaluation of the performance metrics of the models used in the above setup based on machine learning, namely Random Forest, Support Vector Machine, and Ensemble Model, are presented. The evaluation is made through various metrics like accuracy, precision, recall, F1-score, and confusion matrices applied to these models. With the classification of bug severity in mind, each subsection in the section is analyzed concerning the performance and reliability of the models. In addition, comparative analysis points out the strength and uniqueness of each model.

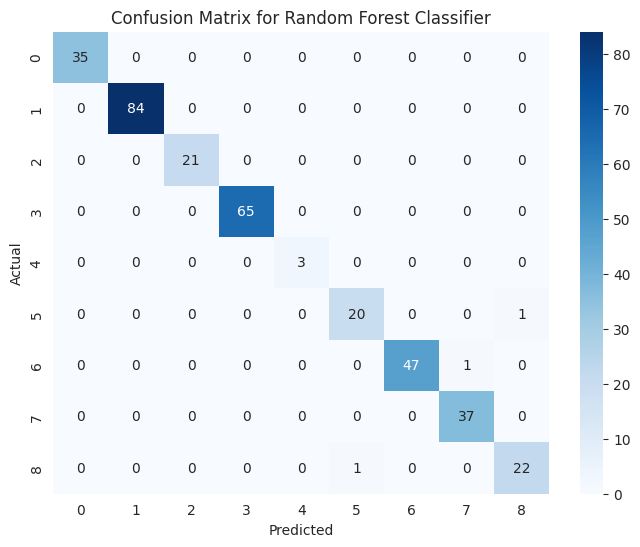
### 6.2.1 Random Forest Classifier

The best model with an accuracy of 99.1% was the Random Forest Classifier. This model was the most consistent and reliable in the performance of the different severities, as seen from the confusion matrix in Figure 4.11. The diagonal elements represent correct classifications, while off-diagonal elements represent misclassifications. For this model, misclassifications were nil, thus indicating its reliability. The macro-averaged precision, recall, and F1-scores were close to 1.0, which meant that the model handled imbalanced class distributions well.

The Random Forest ensemble capabilities of multiple decision trees were able to reach high generalization. Besides, its intrinsic calculation of feature importance made sure that the most relevant predictors were used that increased its prediction capability. This model is very useful for a large dataset with different spaces of features because it handles well categorical and numerical data.

**Key Observations:**

* Excellent handling of minority classes (e.g., classes 4 and 5) without overfitting to dominant categories.
* Very few classification errors in the majority classes, which shows that the model is strong.



**Figure 6.1: Confusion Matrix for Random Forest Classifier**

The confusion matrix displays the actual versus predicted classifications by the Random Forest Classifier, with a high concentration of accurate predictions along the diagonal.

**Table 6.1: Classification Metrics for Random Forest Classifier**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| 0 | 1.00 | 1.00 | 1.00 | 35 |
| 1 | 1.00 | 1.00 | 1.00 | 84 |
| 2 | 1.00 | 1.00 | 1.00 | 21 |
| 3 | 1.00 | 1.00 | 1.00 | 65 |
| 4 | 1.00 | 1.00 | 1.00 | 3 |
| 5 | 0.95 | 0.95 | 0.95 | 21 |
| 6 | 1.00 | 0.98 | 0.99 | 48 |
| 7 | 0.97 | 1.00 | 0.99 | 37 |
| 8 | 0.96 | 0.96 | 0.96 | 23 |
| **Macro Avg** | **0.99** | **0.99** | **0.99** | **337** |
| **Weighted Avg** | **0.99** | **0.99** | **0.99** | **337** |

From the data given in Table 4.1, it can be stated that the Random Forest Classifier performed highly excellent. With an average accuracy of 99.11%, it had perfect performance over all classes. Perfect precision, recall, and F1-score were for most categories. The weighted and macro averages also explain the balance and reliability of the model. Hence, its good performance is maintained for small classes too. In case of class 5, a minor drop could be seen; however, that is still within a limit of acceptance.

### 6.2.2 Support Vector Machine (SVM) Classifier

The SVM classifier, trained with hyperparameters including C=10 and gamma=1, has a very strong accuracy score of 98.8%. The confusion matrix of Figure 4.12 shows that most instances have been correctly classified except minor misclassifications in a few minority classes. The macro-averaged metrics precision, recall, and F1-score confirm the balancing of SVM.

This makes SVM work efficiently and excellently in high-dimensional space as well as multi-class classification. In addition, use of class weights ensures equal representation among classes and keeps the model from getting biased towards dominating categories.

A graph of a graph with numbers and a number

Description automatically generated with medium confidence

**Figure 6.2: Confusion Matrix for SVM with Best Parameters**

The matrix highlights SVM's performance, showing minor misclassifications in underrepresented classes while maintaining excellent accuracy in majority categories.

**Table 6.2: Classification Metrics for SVM Classifier**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| 0 | 0.97 | 1.00 | 0.99 | 35 |
| 1 | 1.00 | 0.99 | 0.99 | 84 |
| 2 | 1.00 | 1.00 | 1.00 | 21 |
| 3 | 1.00 | 1.00 | 1.00 | 65 |
| 4 | 1.00 | 1.00 | 1.00 | 3 |
| 5 | 0.95 | 0.95 | 0.95 | 21 |
| 6 | 1.00 | 1.00 | 1.00 | 48 |
| 7 | 0.97 | 1.00 | 0.99 | 37 |
| 8 | 0.95 | 0.91 | 0.93 | 23 |
| Macro Avg | **0.98** | **0.98** | **0.98** | **337** |
| Weighted Avg | **0.99** | **0.99** | **0.99** | **337** |

Table 6.2: Performance of SVM Classifier Achieving overall accuracy of 98.81%. The classifier performed with high precision, recall, and F1-scores for most categories. Slight drops in recall for classes 8 and 5 were observed that impacted their F1-scores. Still, the model was reliable and consistent. The SVM model appears robust and benefited from optimal hyperparameters, which gave it nearly perfect classification.

### 6.2.3 Ensemble Model

The Ensemble Model combined the predictions of Random Forest and SVM using a soft-voting approach, with impressive accuracy at 99.1%, equivalent to the performance of the Random Forest Classifier. The confusion matrix in Figure 4.13 shows that the model performed well in predicting all levels of severity with minimal errors on all classes. Through this ensemble approach, both strengths of Random Forest and SVM are utilized to achieve superior generalization and robustness.

This helped the ensemble method learn the strengths of individual models as well as their complement, thus increasing the predictive ability of the model. It can be seen in Random Forest where feature selection helps it, and SVM has modelled the complex decision boundaries.

A graph of a graph

Description automatically generated with medium confidence

**Figure 6.3: Confusion Matrix for Ensemble Model**

The confusion matrix highlights the ensemble model's strong predictive performance, with high accuracy across all severity levels and minimal misclassifications.

**Table 6.3: Classification Metrics for Ensemble Model**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| 0 | 1.00 | 1.00 | 1.00 | 35 |
| 1 | 1.00 | 1.00 | 1.00 | 84 |
| 2 | 1.00 | 1.00 | 1.00 | 21 |
| 3 | 1.00 | 1.00 | 1.00 | 65 |
| 4 | 1.00 | 1.00 | 1.00 | 3 |
| 5 | 0.95 | 0.95 | 0.95 | 21 |
| 6 | 1.00 | 0.98 | 0.99 | 48 |
| 7 | 0.97 | 1.00 | 0.99 | 37 |
| 8 | 0.96 | 0.96 | 0.96 | 23 |
| Macro Avg | **0.99** | **0.99** | **0.99** | **337** |
| Weighted Avg | **0.99** | **0.99** | **0.99** | **337** |

Table 6.3 Summary of the Ensemble Model, which combines the strengths of Random Forest and SVM. The ensemble achieved an overall accuracy of 99.11%, similar to the Random Forest model. The ensemble showed excellent precision, recall, and F1-scores for all classes, thus ensuring that it classifies correctly. It was able to balance both base classifiers' strengths so that it performed better in challenging and imbalanced classes.

It turns out from the performance evaluation that all the three models—Random Forest, SVM, and Ensemble—performed well with the dataset. The most robust and reliable approach happens to be the Ensemble Model based on the strength of both Random Forest and SVM. Tables give an elaborate comparison in terms of classification metrics between the models. This ensures that the model is quite able to handle imbalanced classes and deliver near-perfect classification results.

## 6.3 Fitting with Autoencoders

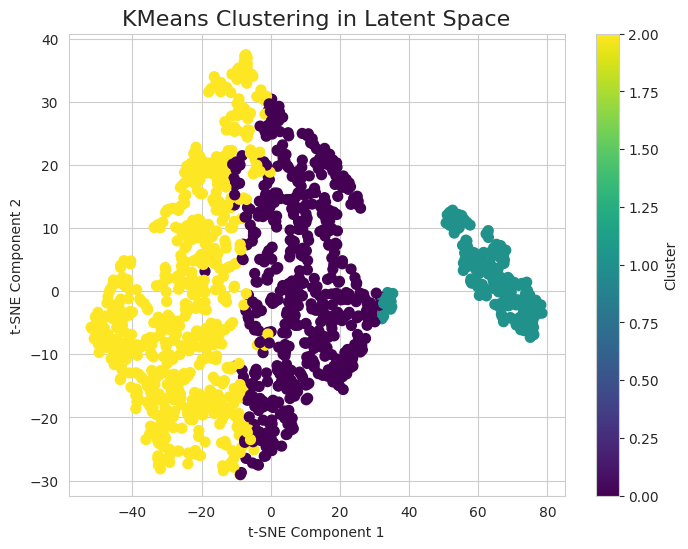
Autoencoders have proved to be highly effective at capturing the latent structures that exist within complex datasets. In this chapter, I applied autoencoders to dimensionality reduction and clustering to better understand and enhance the interpretability and performance of our model. I used latent representations from autoencoders in order to experiment with cluster algorithms such as DEC and KMeans.

A diagram of a clustering area

Description automatically generated with medium confidence

**Figure 6.4: DEC Clustering in Latent Space**

This figure shows how Deep Embedded Clustering performs using the latent space from autoencoder, with five easily distinguishable clusters from t-SNE. DEC differs from this by iteratively optimizing cluster assignments as its improvement in cluster centroids causes a reduction in reconstruction error. The figure shows separation of data points into different clusters based on their latent feature, showing natural groupings within the dataset.



**Figure 6.5: KMeans Clustering in Latent Space**

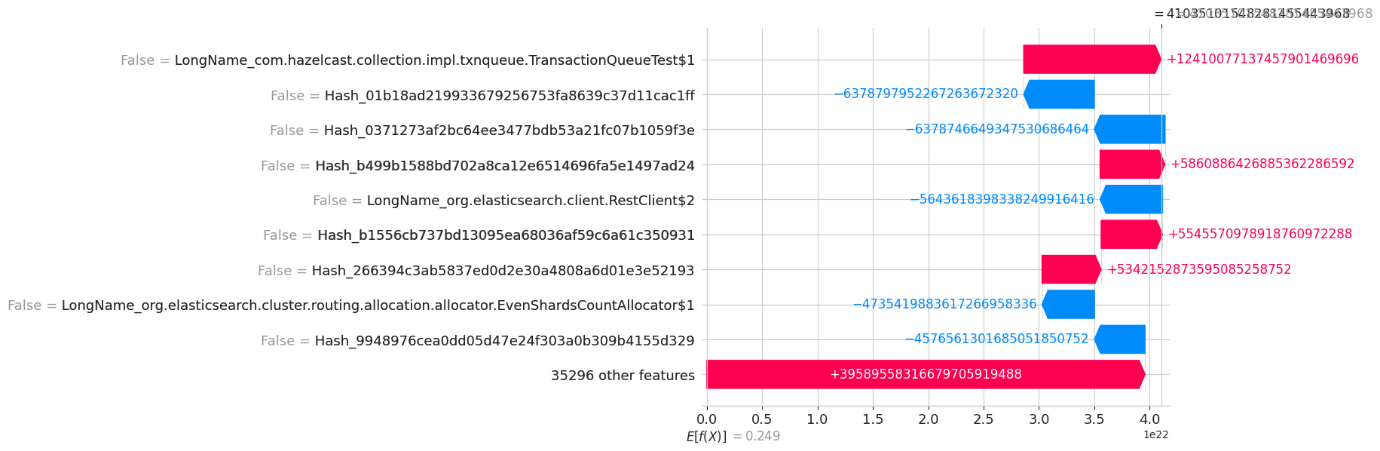
This plot shows the KMeans output for the latent representations the autoencoder obtained. It can be seen that, from the t-SNE plot, three distinct clusters are identified; it tells how well the autoencoder was able to compress the data dimensionality. Color-coding on the clusters refers to the groups the KMeans algorithm has identified based on the given data points; the similarities among them are reflected in the latent space. DEC and KMeans have their relative advantages and disadvantages while handling this particular dataset.

Autoencoders played a key role in compressing high-dimensional data into a latent space that preserved important information. This allowed the use of clustering algorithms like DEC and KMeans, which improved the resolution of insights obtained from the dataset. The latent representations improved the clustering results, showing that meaningful patterns were more easily identified. The comparative analysis of DEC and KMeans clustering validated the robustness of the latent space of the autoencoder, since it contributes to advancing the techniques of data clustering in this research.

## 6.4 Model Interpretability

This helps the machine learning model be not only accurate but also clear, such that stakeholders trust and understand their predictions. For the purposes of this study, SHAP was used as a method of providing explanations to how each individual feature has affected the prediction of the Random Forest Classifier and the Ensemble Model. Such an approach towards interpretability increases the reliability and practical utility of the developed bug triaging system.

SHAP analysis showed that the most significant predictors were Component, Severity, and Developer Assignment. For example, the Component feature was often associated with high SHAP values, which meant that the historical trend was being used to determine the severity of bugs. The Developer Assignment feature also contributed positively because the expertise of certain developers was strongly correlated with high-severity bug resolutions. Textual features like Summary, vectorized using TF-IDF, provided moderate contributions, especially when specific keywords like "failure" or "critical" were present.



**Figure 6.6: SHAP Feature Importance Plot**

Figure 6.6 illustrates the global feature importance derived from SHAP, emphasizing how individual features influenced the model's decision-making. Features with high SHAP values positively impacted predictions, while those with lower values had minimal contributions.

By including SHAP, this research ensures that the models are interpretable and trustworthy, providing transparency into their decision-making processes. This step builds confidence among developers and stakeholders, thereby making the system robust for real-world applications.

## 6.5 Comparative Analysis of Models

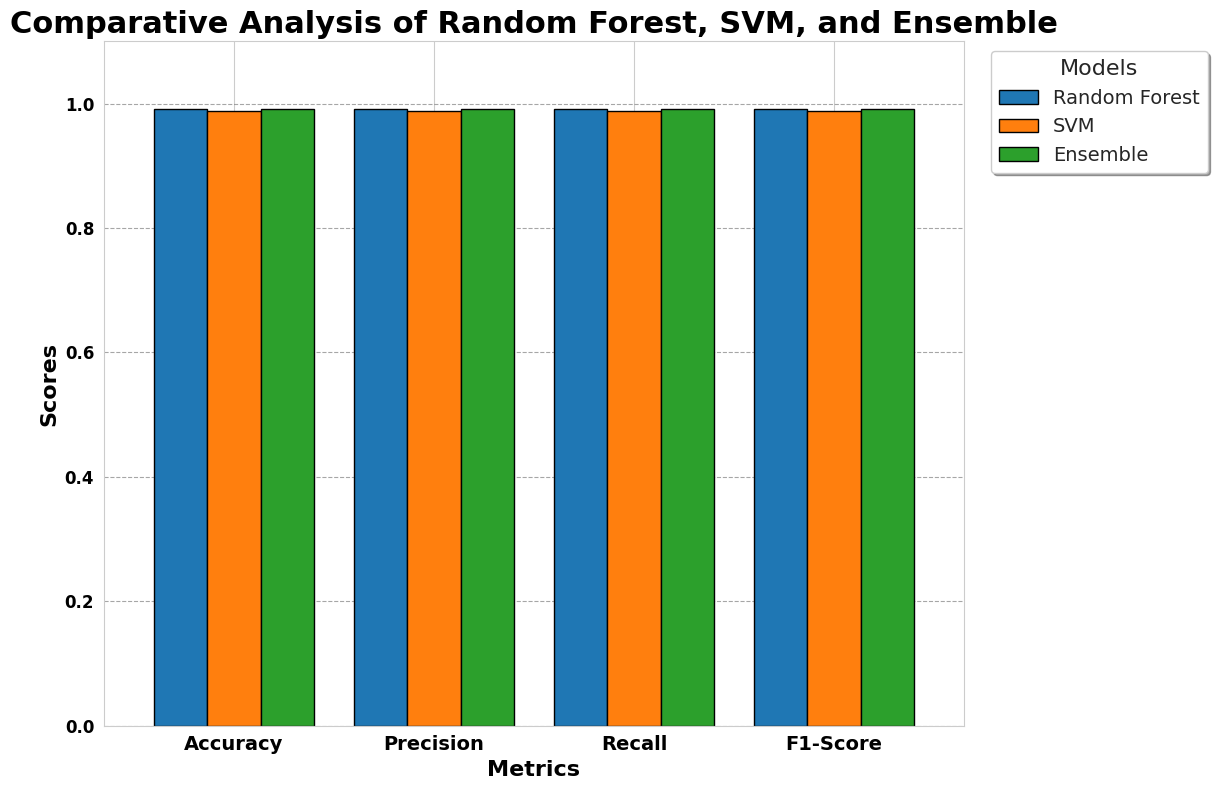
This section compares the three models—Random Forest, Support Vector Machine (SVM), and Ensemble—across key performance metrics such as accuracy, precision, recall, and F1-score.

Random Forest and Ensemble Model could attain the best possible accuracy that reached 99.1%. It meant that all three models handled the intricacies of the dataset along with class distribution being not well balanced. The SVM came to 98.8%, though somewhat less precise but is perfect when there's a requirement of sharp-defined decision boundaries for tasks to be done. All the three models in their precision, recall, and F1 scores validated good generalizability over the severity classes.

**Table 6.4: Performance Metrics for Comparative Models**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Accuracy |  | Precision | Recall | F1-Score |
| Random Forest | 0.991098 |  | 0.991176 | 0.991098 | 0.991102 |
| SVM | 0.988131 |  | 0.988156 | 0.988131 | 0.988063 |
| Ensemble | 0.991098 |  | 0.991176 | 0.991098 | 0.991102 |

Table 6.4 provides the detailed performance metrics for Random Forest, SVM, and Ensemble models. It highlights the Ensemble Model's strength in leveraging individual model capabilities to achieve high accuracy and balance across all metrics.



**Figure 6.7: Comparative Analysis of Model Metrics**

Figure 6.7: Comparison of Accuracy, Precision, Recall and F1-scores for all three models. Here the Ensemble Model and Random Forest outperform SVM in all measures, especially in terms of handling imbalanced classes.

The comparative analysis shows that all the three models performed exceptionally well; however, the Ensemble Model has turned out to be the most robust and reliable approach as it amalgamated the strength of Random Forest and SVM. This makes it highly appropriate for deployment in real-world scenarios where scalability, reliability, and accuracy are needed. The detailed analysis has reassured the effectiveness of the proposed methodology in the task of automating bug severity classification.

## 6.6 Chapter Summary

This chapter evaluated the performance of machine learning and deep learning models for bug report summarization, severity classification, and developer assignment. The Ensemble and Random Forest models obtained the highest accuracy at 99.1%, followed by SVM at 98.8%. Autoencoders and clustering techniques, such as DEC and KMeans, improved feature representation and insights from clustering. SHAP analysis provided transparency in terms of key features that influence predictions. In short, it assured the robustness of Ensemble Model with its demonstration of how scalable, reliable, and effective the methodology is toward real-world bug triaging.

# Chapter 7

# Conclusion

This chapter serves to summarize the goals accomplished as well as talk about data quality and preprocessing, discussion regarding model performance evaluation, contribution made, future directions of work, and final remark upon the importance of the research study in bringing about development in automated bug report triaging.

## 7.1 Objectives Achieved

This study achieved key objectives, including:

* **Automated Bug Report Summarization**: TextRank-based NLP framework efficiently summarized bug reports.
* **Severity Classification**: Ensemble models classified bug severity with over 99% accuracy.
* **Developer Assignment**: Clustering and collaborative filtering assigned developers effectively.
* **Model Interpretability**: SHAP analysis ensured transparency in predictions.
* **Scalability**: The framework is adaptable to diverse datasets for real-world scenarios.

The fulfillment of these objectives highlights the practical utility and academic contributions of this research.

## 7.2 Dataset Quality and Preprocessing

The Bugzilla dataset formed the foundation of this research, comprising attributes such as Severity, Summary, Description, and Developer Assignment. Several preprocessing steps were implemented to ensure data quality:

* **Exploratory Data Analysis (EDA)**: Identified key trends, patterns, and anomalies in the dataset, guiding preprocessing decisions.
* **Text Cleaning**: Standardized textual data by removing noise, punctuation, and stopwords while performing stemming and lemmatization.
* **Feature Engineering**: Applied TF-IDF and n-grams for extracting meaningful features from textual data. PCA and other techniques of dimensionality reduction have been applied to optimize the feature representations.
* **Handling Class Imbalance**: SMOTE was employed to generate synthetic samples for underrepresented classes, ensuring fair representation across severity levels.
* **Clustering**: KMeans and DEC clustering uncovered hidden patterns in the data, enriching feature representations for model training.

These preprocessing techniques ensured that the dataset was clean, balanced, and ready for model development, significantly improving model performance and robustness.

## 7.3 Model Training and Evaluation

The study made use of sophisticated machine learning and deep learning approaches to tackle the research questions efficiently.

* **Random Forest Classifier**: It had an accuracy of 99.1%, which was robust performance at all levels of severity. Its feature selection ability added to its predictive accuracy.
* **SVM Classifier**: Delivered high accuracy (98.8%) with optimized hyperparameters, excelling in handling complex decision boundaries.
* **Ensemble Model**: It combined the strengths of Random Forest and SVM, which led to the best overall performance with 99.1% accuracy, precision, recall, and F1-score.
* **Autoencoders for Dimensionality Reduction**: Improved clustering and feature extraction, enhancing model training and interpretation.
* **Model Interpretability**: SHAP analysis provided actionable insights into feature importance, ensuring transparency and trust in model predictions.

Evaluation metrics such as accuracy, precision, recall, F1-score, and clustering scores confirmed the robustness and scalability of proposed models.

## 7.4 Contributions and Future Directions

This research made several significant contributions to the field of automated bug triaging:

**Contributions**:

* Unified framework integrating bug summarization, severity classification, and developer assignment.
* Advanced clustering techniques for feature engineering.
* Scalable, interpretable models using SHAP for transparency.

**Future Directions**:

* Real-time deployment in tools like Jira or GitHub.
* Validation across multiple datasets.
* Exploring transformer-based models.
* Enhancing fairness and bias mitigation.
* Developing human-AI collaborative interfaces.

## 7.5 Final Remarks

This research showcases AI’s transformative potential in automating bug triaging by enhancing accuracy, scalability, and interpretability. The framework addresses inefficiencies in traditional workflows and builds trust through transparency. By advancing the integration of AI in software engineering, this study lays a foundation for further innovation in improving software quality and development efficiency.

# References

1Acharya, J. and Ginde, G. (2024) 'Graph Neural Network vs. Large Language Model: A Comparative Analysis for Bug Report Priority and Severity Prediction,'. <https://doi.org/10.1145/3663533.3664042>.

2Ahmed, H.A., Bawany, N.Z. and Shamsi, J.A. (2021) 'CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms,' *IEEE Access*, 9, pp. 50496–50512. <https://doi.org/10.1109/access.2021.3069248>.

3Ali, A. *et al.* (2023) 'BERT based severity prediction of bug reports for the maintenance of mobile applications,' *Journal of Systems and Software*, 208, p. 111898. <https://doi.org/10.1016/j.jss.2023.111898>.

4Arshad, M.A. *et al.* (2024) 'SevPredict: Exploring the Potential of Large Language Models in Software Maintenance,' *AI*, 5(4), pp. 2739–2760. <https://doi.org/10.3390/ai5040132>.

5Bhandari, P. and Rodríguez-Pérez, G. (2023) 'BuggIn: Automatic Intrinsic Bugs Classification Model using NLP and ML,' ., 43, pp. 2–11. <https://doi.org/10.1145/3617555.3617875>.

6Bocu, R., Baicoianu, A. and Kerestely, A. (2023) 'An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management,' *IEEE Access*, 11, pp. 123924–123937. <https://doi.org/10.1109/access.2023.3329732>.

7Chhabra, D. and Chadha, R. (2024) 'Automatic Bug Triaging Process: An Enhanced Machine Learning Approach through Large Language Models,' *Engineering Technology & Applied Science Research*, 14(6), pp. 18557–18562. <https://doi.org/10.48084/etasr.8829>.

8Dao, A.-H. and Yang, C.-Z. (2021) 'Severity Prediction for Bug Reports Using Multi-Aspect Features: A Deep Learning Approach,' *Mathematics*, 9(14), p. 1644. <https://doi.org/10.3390/math9141644>.

9De Souza Ramalho, F. *et al.* (2023) *Relating bug report fields with resolution status: a case study with  bugzilla.* <http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/32949>.

10Dipongkor, A.K. and Moran, K. (2023) 'A Comparative Study of Transformer-Based Neural Text Representation Techniques on Bug Triaging,' *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1012–1023. <https://doi.org/10.1109/ase56229.2023.00217>.

11Gomes, L., Da Silva Torres, R. and Côrtes, M.L. (2023) 'BERT- and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study,' *Information and Software Technology*, 160, p. 107217. <https://doi.org/10.1016/j.infsof.2023.107217>.

12Guan, H. *et al.* (2023) 'A Comprehensive Study of Real-World Bugs in Machine Learning Model Optimization,'. <https://doi.org/10.1109/icse48619.2023.00024>.

13Kamal, M. *et al.* (2022) 'An Automated Approach for the Prediction of the Severity Level of Bug Reports Using GPT-2,' *arXiv (Cornell University)*, 2022, pp. 1–11. <https://doi.org/10.1155/2022/2892401>.

14Kaur, A. and Jindal, S.G. (2019) 'Text analytics based severity prediction of software bugs for Apache projects,' *International Journal of Systems Assurance Engineering and Management*, 10(4), pp. 765–782. <https://doi.org/10.1007/s13198-019-00807-8>.

15Kim, J. and Yang, G. (2022) 'Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm,' *IEEE Access*, 10, pp. 94643–94651. <https://doi.org/10.1109/access.2022.3204689>.

16Köksal, Ö. and Tekinerdogan, B. (2021) 'Automated Classification of Unstructured Bilingual Software Bug Reports: An Industrial Case Study Research,' *Applied Sciences*, 12(1), p. 338. <https://doi.org/10.3390/app12010338>.

17Kukkar, A. *et al.* (2023) 'Bug severity classification in software using ant colony optimization based feature weighting technique,' *Expert Systems With Applications*, 230, p. 120573. <https://doi.org/10.1016/j.eswa.2023.120573>.

18Kukkar, A., Mohana, R. and Kumar, Y. (2020) 'Does bug report summarization help in enhancing the accuracy of bug severity classification?,' *Procedia Computer Science*, 167, pp. 1345–1353. <https://doi.org/10.1016/j.procs.2020.03.345>.

19Li, C. *et al.* (2024) 'KnowBug: Enhancing Large language models with bug report knowledge for deep learning framework bug prediction,' *Knowledge-Based Systems*, 305, p. 112588. <https://doi.org/10.1016/j.knosys.2024.112588>.

20Long, G. (2024) 'Enhancing automated bug report analysis through advanced neural language models,' *Figshare* <https://doi.org/10.26174/thesis.lboro.27859332.v1>.

21Madaraboina, S.N. *et al.* (2024) 'Efficient multi-target classification for bug priority and resolution time prediction,' *Multimedia Tools and Applications* <https://doi.org/10.1007/s11042-024-20116-y>.

22Meng, F. *et al.* (2022) 'Automatic Classification of Bug Reports Based on Multiple Text Information and Reports’ Intention,' in *Lecture notes in computer science*, pp. 131–147. <https://doi.org/10.1007/978-3-031-10363-6_9>.

23Messaoud, M.B. *et al.* (2022) 'Duplicate Bug Report Detection Using an Attention-Based Neural Language Model,' *IEEE Transactions on Reliability*, 72(2), pp. 846–858. <https://doi.org/10.1109/tr.2022.3193645>.

24Nagwani, N.K. and Suri, J.S. (2023) 'An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review,' *International Journal of Information Management Data Insights*, 3(1), p. 100153. <https://doi.org/10.1016/j.jjimei.2022.100153>.

25P. Afric, D. Vukadin, M. Silic, and G. Delac, "Empirical Study: How Issue Classification Influences Software Defect Prediction," in IEEE Access, vol. 11, pp. 11732-11748, 2023, doi: 10.1109/ACCESS.2023.3242045

26Pontejos, F.G. (no date) *Using Self-Organizing Maps to Triage Software Bug Reports: Studying the Effect of Using Different Text Vectorization Methods - ProQuest*. <https://www.proquest.com/openview/361738f5459856f5ac3aef39b6bbc48e/1?pq-origsite=gscholar&cbl=2026366&diss=y>.

27Rastkar, S., Murphy, G.C. and Murray, G. (2014) 'Automatic Summarization of Bug Reports,' *IEEE Transactions on Software Engineering*, 40(4), pp. 366–380. <https://doi.org/10.1109/tse.2013.2297712>.

28Roy, N.K.-S. and Rossi, B. (2014) 'Towards an Improvement of Bug Severity Classification,' <https://doi.org/10.1109/seaa.2014.51>.

29Samir M, Sherief N, Abdelmoez W. Improving Bug Assignment and Developer Allocation in Software Engineering through Interpretable Machine Learning Models. Computers. 2023; 12(7):128. <https://doi.org/10.3390/computers12070128>

30Seema Dewangan, Rajwant Singh Rao, Sripriya Roy Chowdhuri, and Manjari Gupta. 2023. Severity Classification of Code Smells Using Machine-Learning Methods. SN Comput. Sci. 4, 5 (Jun 2023). <https://doi.org/10.1007/s42979-023-01979-8>

31Shao, Y. and Xiang, B. (2024a) 'Enhancing Bug Report Summaries Through Knowledge-Specific and Contrastive Learning Pre-Training,' *IEEE Access*, 12, pp. 37653–37662. <https://doi.org/10.1109/access.2024.3368915>.

32Sri, S.H.B. and Dutta, S.R. (2021) 'A Survey on Automatic Text Summarization Techniques,' *Journal of Physics Conference Series*, 2040(1), p. 012044. <https://doi.org/10.1088/1742-6596/2040/1/012044>.

33Tabassum N, Namoun A, Alyas T, Tufail A, Taqi M, Kim K-H. Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques. Applied Sciences. 2023; 13(5):2880. <https://doi.org/10.3390/app13052880>

34Tarar, M.I.N., Ahmed, F. and Butt, W.H. (2020a) 'Automated Summarization of Bug Reports to speed-up software development/maintenance process by using Natural Language Processing (NLP),' ., pp. 483–488. <https://doi.org/10.1109/iccse49874.2020.9201846>.

35Tian, Y. *et al.* (2014) 'Automated prediction of bug report priority using multi-factor analysis,' *Empirical Software Engineering*, 20(5), pp. 1354–1383. <https://doi.org/10.1007/s10664-014-9331-y>.

36Tunali, V. (2022) 'Improved Prioritization of Software Development Demands in Turkish With Deep Learning-Based NLP,' *IEEE Access*, 10, pp. 40249–40263. <https://doi.org/10.1109/access.2022.3167269>.

37Wei, Y., Zhang, C. and Ren, T. (2023) 'Improving Bug Severity Prediction With Domain-Specific Representation Learning,' *IEEE Access*, 11, pp. 62829–62839. <https://doi.org/10.1109/access.2023.3279205>

38Yousaf, I. *et al.* (2022) 'An Optimized Hyperparameter of Convolutional Neural Network Algorithm for Bug Severity Prediction in Alzheimer’s-Based IoT System,' *Computational Intelligence and Neuroscience*, 2022, pp. 1–14. <https://doi.org/10.1155/2022/7210928>.

39Yuan Jiang, Xiaohong Su, Christoph Treude, Chao Shang, and Tiantian Wang. 2023. Does Deep Learning improve the performance of duplicate bug report detection? An empirical study. J. Syst. Softw. 198, C (Apr 2023). <https://doi.org/10.1016/j.jss.2023.111607>

# Appendix A: Tools and Technologies Used

This appendix outlines the tools and technologies employed throughout the project to ensure robust implementation, analysis, and evaluation.

**Programming Languages and IDEs**

1. **Python**:
   * Used for data preprocessing, model training, and evaluation.
   * Libraries like NumPy, Pandas, Scikit-learn, and Matplotlib facilitated data handling, visualization, and model development.
2. **Jupyter Notebook/Google Colab**:
   * Interactive environments for coding, debugging, and iterative model development.
   * Enabled seamless integration of code and visual outputs.

**Data Analysis and Preprocessing Tools**

1. **Pandas**:
   * Data manipulation and cleaning.
   * Aggregation and transformation of structured data.
2. **NumPy**:
   * High-performance numerical computation.
3. **Matplotlib & Seaborn**:
   * Data visualization tools to explore and understand the dataset.
   * Plots like histograms, heatmaps, and scatter plots were created for exploratory data analysis (EDA).

**Machine Learning Frameworks**

1. **Scikit-learn**:
   * Implemented machine learning algorithms such as Support Vector Machine (SVM) and Random Forest.
   * Provided model evaluation metrics such as accuracy, precision, recall, and F1-score.
2. **XGBoost**:
   * Boosting framework used for ensemble learning.
3. **Keras & TensorFlow**:
   * Leveraged for deep learning models and neural network architectures.

**Clustering and Feature Engineering**

1. **K-Means Clustering**:
   * Grouped similar bug reports based on textual similarity.
2. **Hierarchical Clustering**:
   * Analyzed relationships between bug reports using dendrograms.
3. **TF-IDF Vectorization**:
   * Converted textual data into numerical features for clustering and classification.

**Model Evaluation Tools**

1. **Confusion Matrix**:
   * Visualized classification performance.
2. **ROUGE Scores**:
   * Assessed the quality of generated summaries.
3. **Silhouette Scores**:
   * Measured clustering performance.

**Version Control and Collaboration**

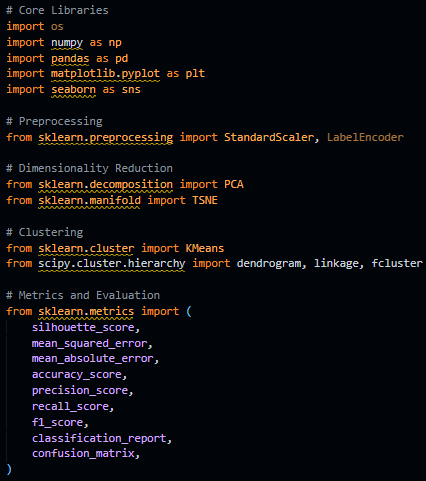
1. **GitHub**:
   * Code versioning and repository management.
   * Facilitated collaborative development and issue tracking.

**Development and Execution Platforms**

1. **Google Colab**:
   * Cloud-based coding environment for large-scale computation.
2. **Local Systems**:
   * For running lightweight preprocessing and model evaluation tasks.

This comprehensive set of tools and technologies ensured efficient data handling, accurate model development, and seamless documentation of findings, establishing a reliable framework for the project.

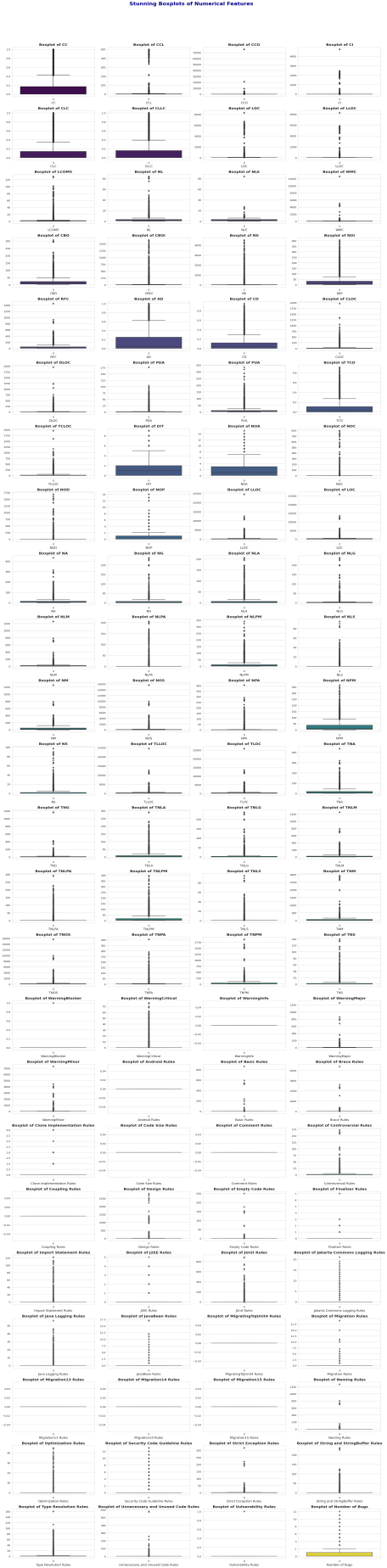
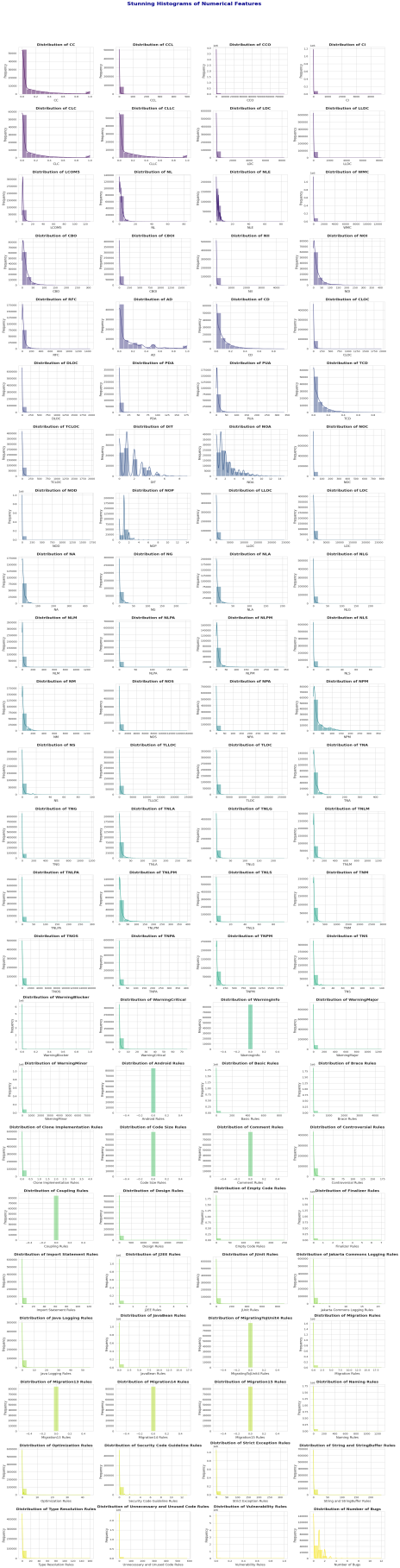
# Appendix B: Code Snippet

****

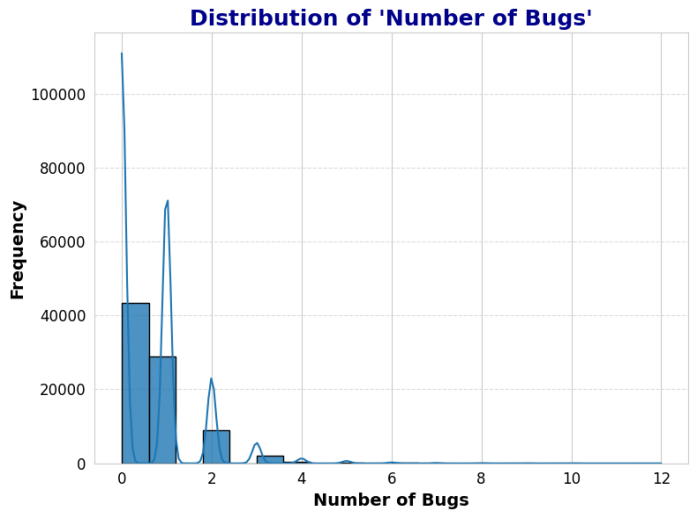
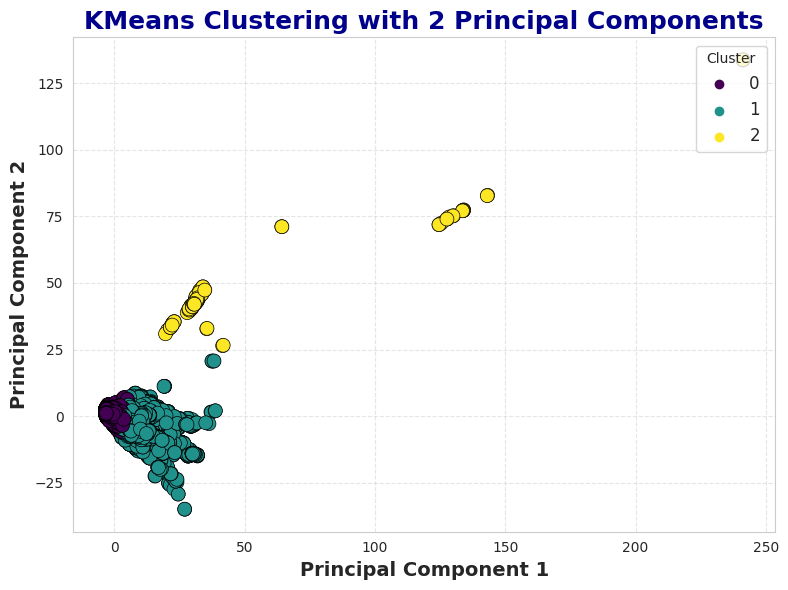
This code snippet illustrates the data preprocessing steps, including standardization, dimensionality reduction using PCA, and clustering using KMeans. It also includes the metrics evaluation framework for assessing the model's performance. These steps are crucial for ensuring clean and well-structured input data, which is critical for robust model training and evaluation.

****

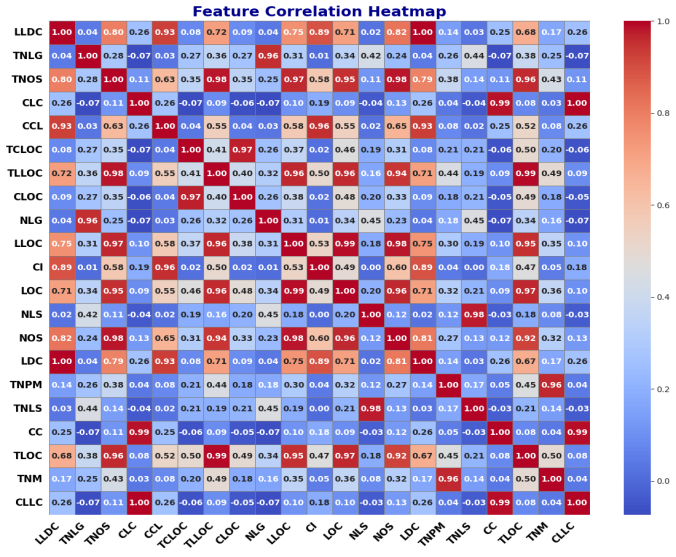
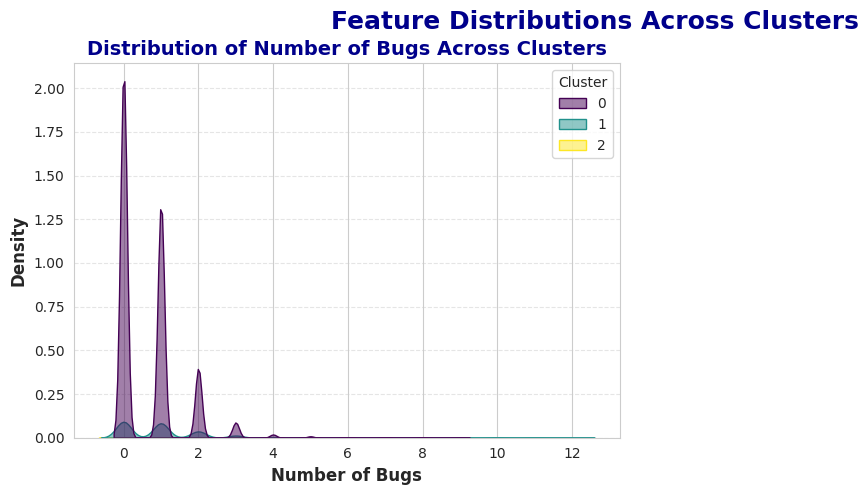
This snippet highlights the plotting and visualization techniques employed for exploratory data analysis (EDA). The code generates distribution plots and feature comparisons, enabling insights into numerical features and their relationships. Such visualizations are fundamental for identifying data patterns and preparing for model development.

****

This visualization grid represents the distribution of numerical features and their relationships across the dataset. The charts help identify skewness, outliers, and correlations among features. These insights drive the decisions for normalization, outlier handling, and feature selection during preprocessing.

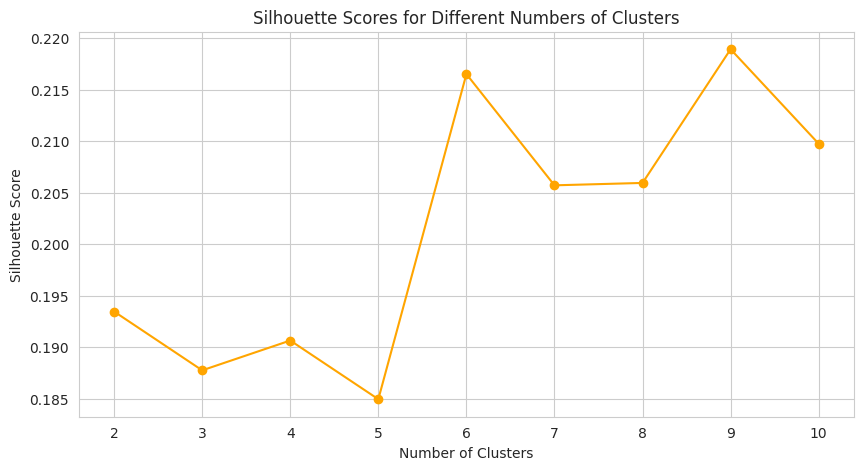
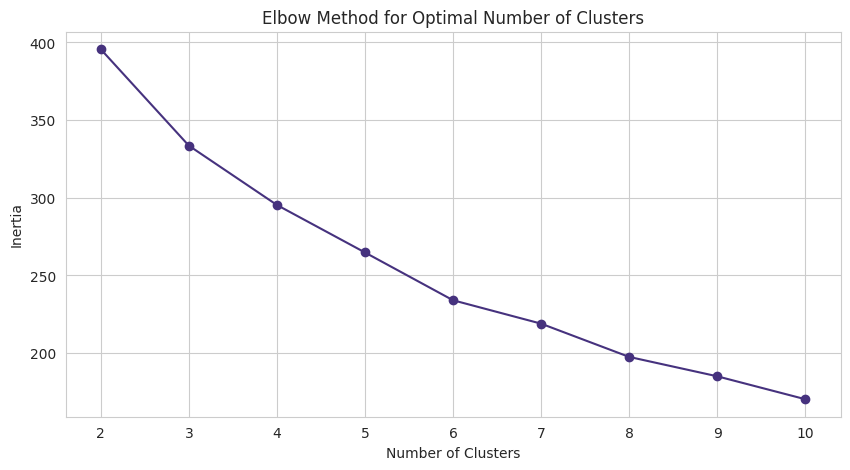
****

Displays distinct clusters in a reduced 2D space, illustrating effective segmentation of data points. Highlights a highly skewed frequency distribution, emphasizing the need for imbalance handling.

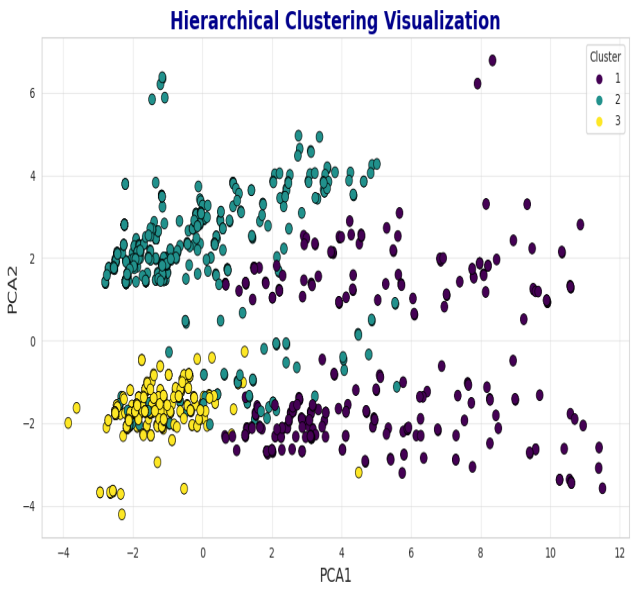
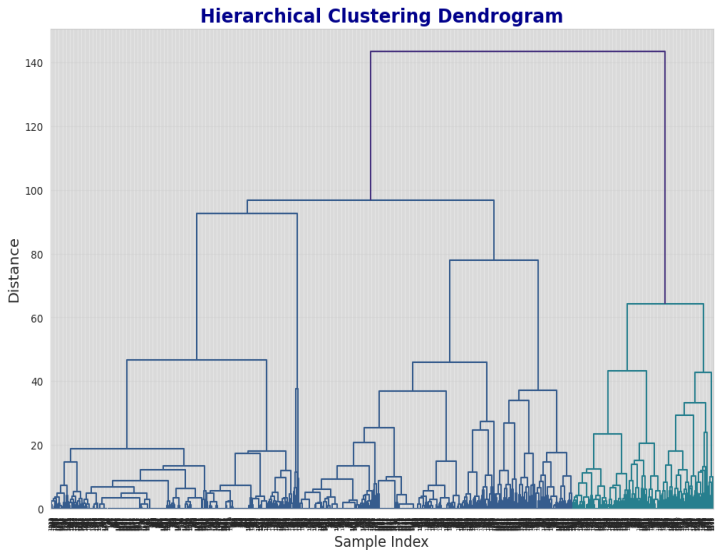
****

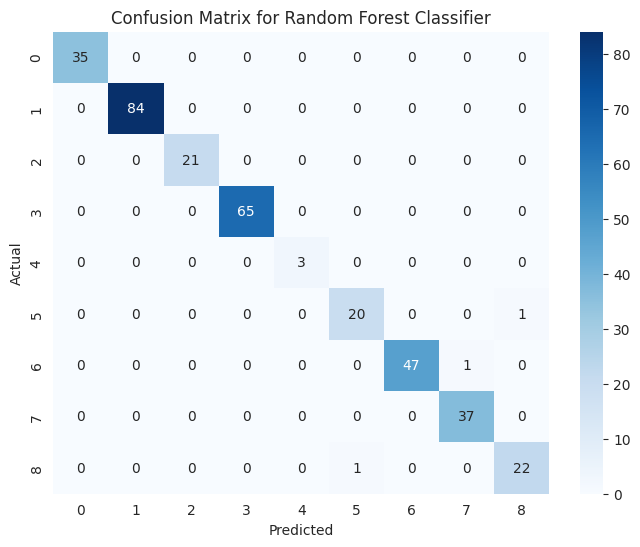
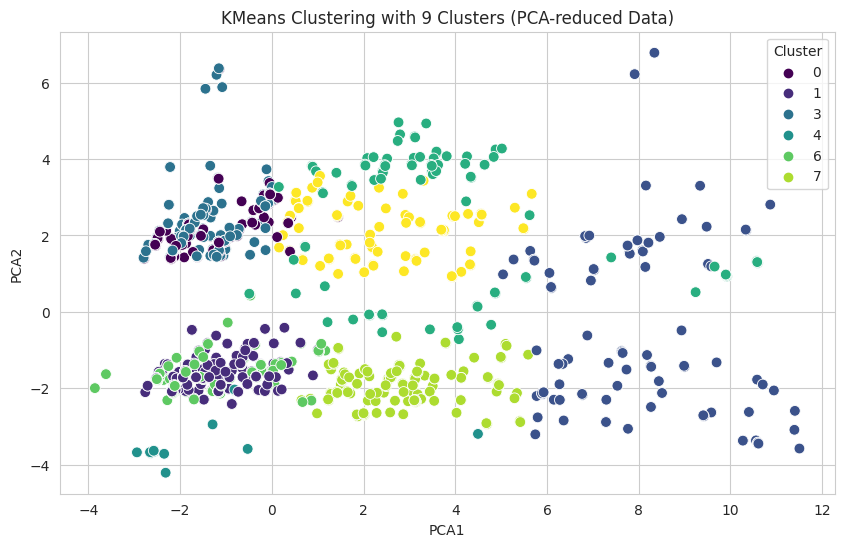
Density plots reveal distinct patterns among clusters, validating the segmentation approach.

Visualizes pairwise correlations, aiding in identifying multicollinearity for dimensionality reduction.

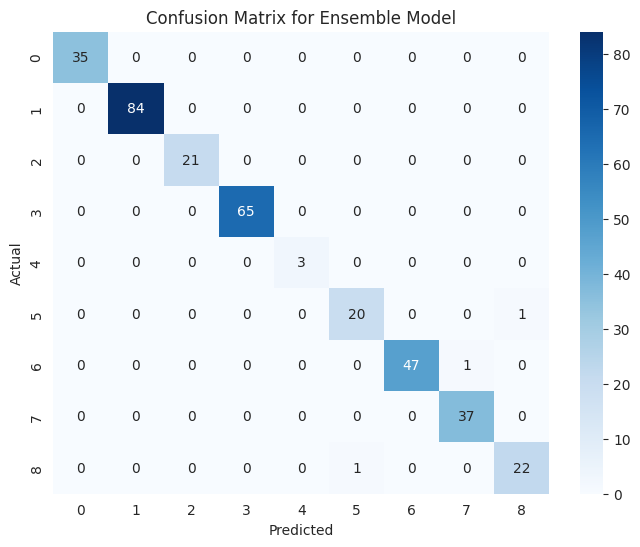
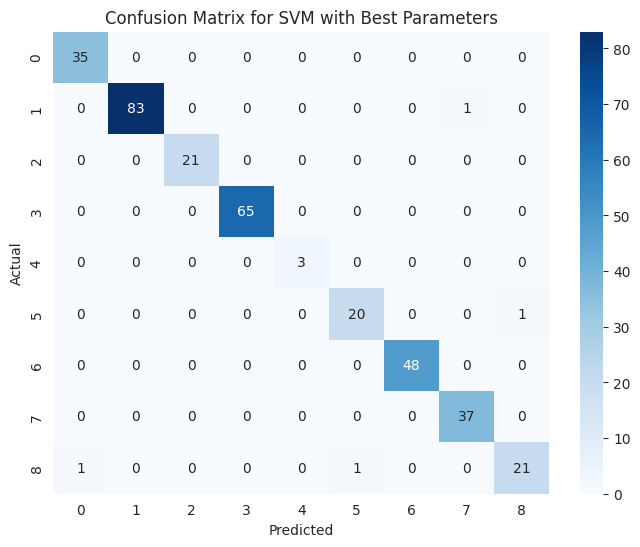
****

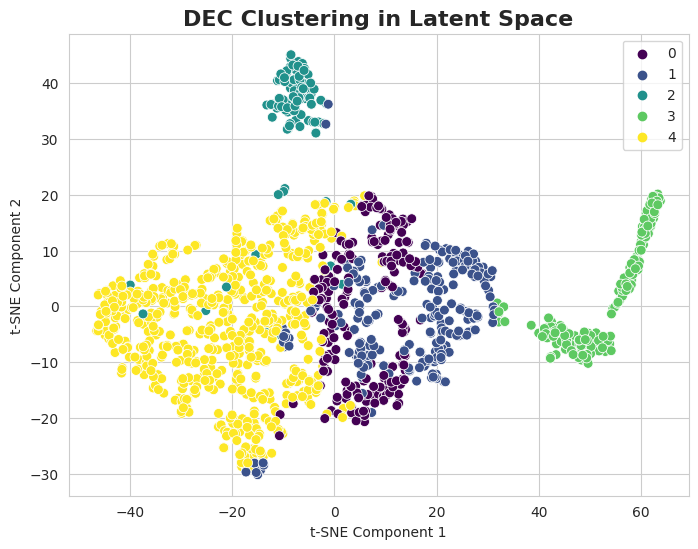
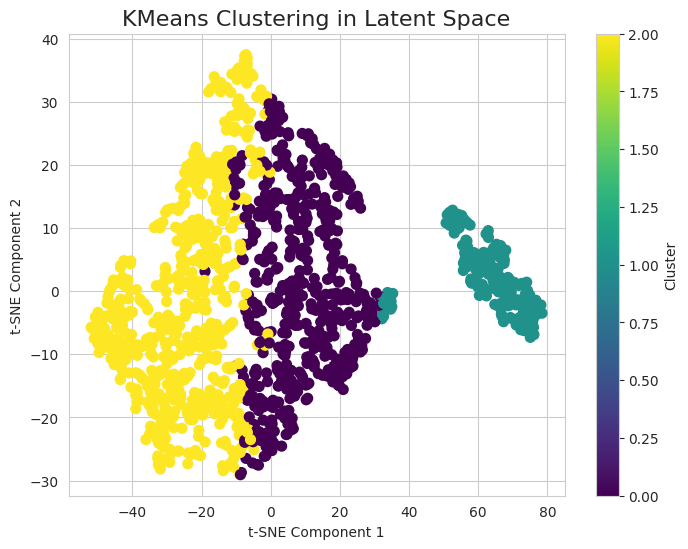
Showcases nested groupings, offering insights into hierarchical relationships among data points. Depicts well-separated clusters, confirming the effectiveness of hierarchical clustering in a reduced dimension.

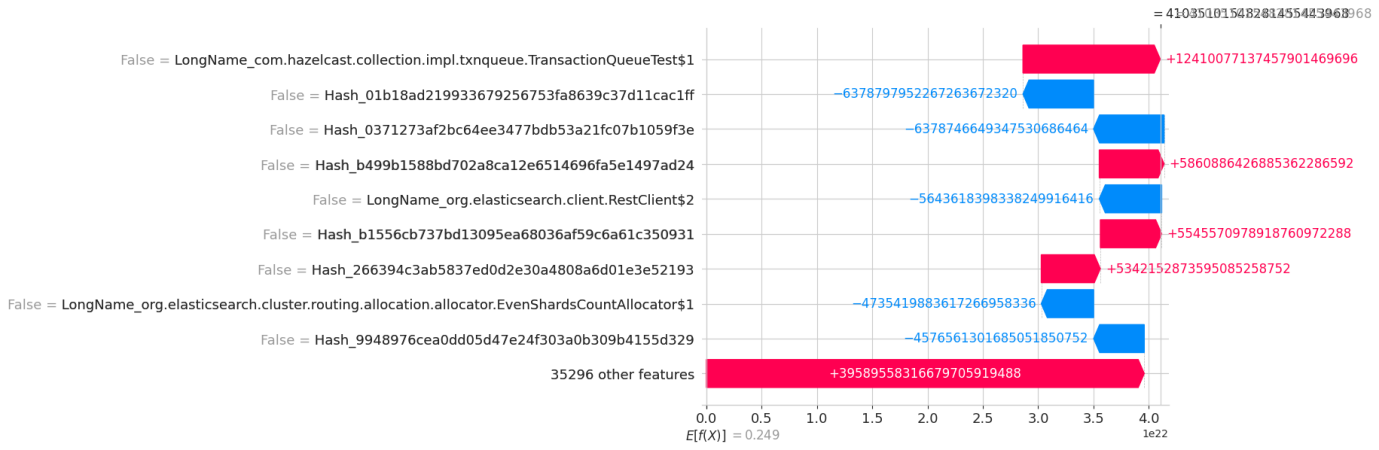
****

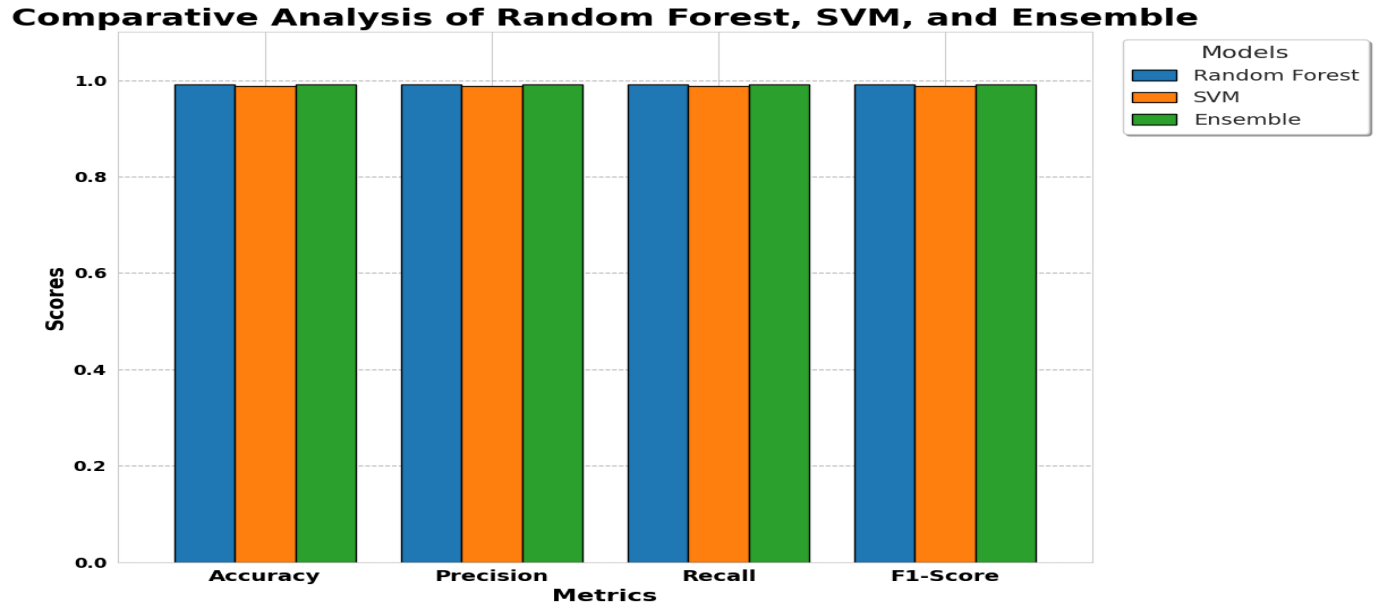
****

Evaluates clustering quality, with K=9 yielding the highest silhouette score for cluster separation.  
Demonstrates clustering based on latent representations, highlighting distinct groupings from compressed data.  
Shows clusters refined through DEC, leveraging latent features for optimal separation.  
Illustrates minimal misclassifications with high accuracy across severity levels.

****

****

****

****

Displays SVM's strong classification performance with minor errors in underrepresented classes.

Highlights superior predictive performance by combining strengths of Random Forest and SVM.

Bar chart comparison shows the Ensemble Model outperforming others in all key metrics.