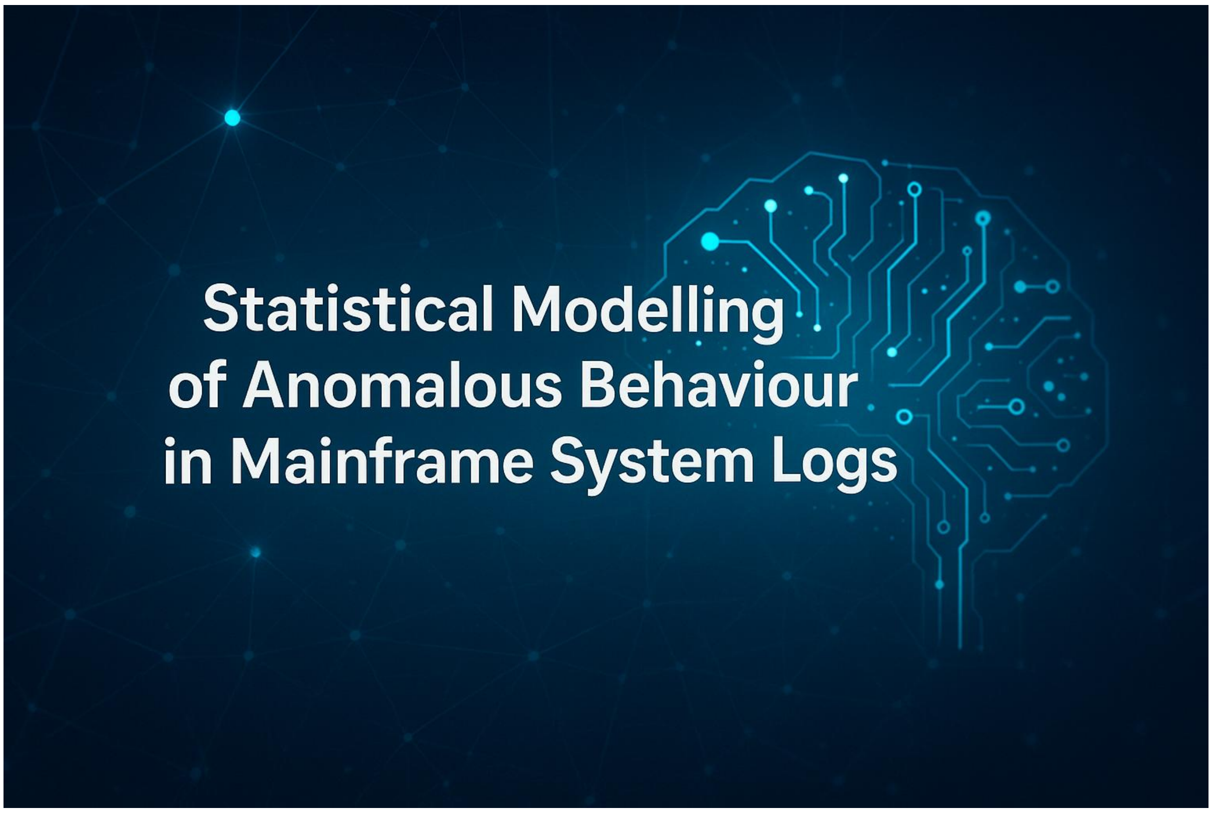


Investigating Statistical Methods for Identifying Irregular Patterns in Mainframe Log Data.

23025191 | Amirthavarshini Vimaleshwara Raja

Advance Computer Science Masters Project- 7COM1039- 0901- 2025

Supervisor | Kelechi Emerole

MSc FPR Declaration

This report is submitted in partial fulfilment of the requirement for the degree of: Master of Science in Data Science and Analytics, at the University of Hertfordshire (UH).

I hereby declare that the work presented in this project and report is entirely my own, except where explicitly stated otherwise. All sources of information and ideas, whether quoted directly or paraphrased, have been properly referenced in accordance with academic standards. I understand that any failure to properly acknowledge the work of others could constitute plagiarism and may result in academic penalties.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

Table of Contents

"Understanding anomalous behaviour in mainframe system logs through statistical modelling."

1. Abstract:

This project explores log-based anomaly detection in Hadoop Distributed File System (HDFS) logs using simple, interpretable statistical techniques rather than deep-learning or black-box models. The objective is to detect abnormal log windows and to examine how different unsupervised methods perform on real system data. Four approaches are considered: Euclidean distance, Mahalanobis distance, a multivariate Gaussian likelihood score, and Local Outlier Factor (LOF). An annotated HDFS benchmark dataset is used to derive event-count features per block, and anomaly thresholds are individually tuned by maximizing the F1-score on a validation split.

The calibrated methods are then applied to an unlabelled subset of HDFS logs, with thresholds selected using percentile-based rules. The results show that distance-based and Gaussian methods achieve higher accuracy but tend to miss more anomalies, while LOF captures a larger number of anomalies at the cost of very low accuracy. When applied to unlabelled data, several log windows are consistently flagged as anomalous, suggesting strong deviations from normal event patterns. Overall, the study demonstrates that straightforward statistical methods can provide an effective and interpretable baseline for log anomaly detection and highlights the benefit of combining multiple detectors for more robust system monitoring.

Chapter1: Introduction

Mainframe systems continue to play a critical role in many large-scale and mission-critical computing environments. Every day, they generate billions of log entries that record how the system is behaving, how resources are being used, and where potential problems may be developing. When used effectively, these logs offer valuable insight into system health, performance bottlenecks, and early warning signs of failures or security issues. However, the volume and complexity of this data make manual analysis unrealistic, which highlights the need for dependable automated anomaly detection.

Many existing log monitoring solutions rely on fixed thresholds, predefined rules, or black-box machine learning models. While these approaches can detect well-known issues, they often struggle to capture subtle or unexpected behaviour and typically provide little explanation for why an alert was triggered. In practice, this can lead to missed problems, excessive false alarms, and reduced confidence from system operators who need to understand the reasons behind alerts to act on them.

Recent studies suggest that simpler and more transparent statistical methods offer a promising alternative. Techniques such as moving averages, distance-based and density-based measures, and probabilistic models have shown that it is possible to detect abnormal behaviour in complex log data while still maintaining interpretability. These approaches make it easier to reason about what constitutes normal system behaviour and why certain patterns are flagged as anomalous.

Building on this idea, this project develops a statistically grounded anomaly detection framework for mainframe system logs. The focus is on cleaning and structuring log data, extracting meaningful features, and applying a range of unsupervised statistical models to distinguish routine system noise from genuine signs of risk. By comparing multiple detection methods and examining their outputs, the project aims to provide an early warning system that operators can trust. one that not only detects anomalies, but also clearly explains the behaviour behind them.

1.1 Problem Overview

This project focuses on a practical and important problem: how to reliably identify anomalous behaviour in Hadoop Distributed File System (HDFS) logs using methods that are both transparent and unsupervised. In real operational environments, system operators need monitoring tools that can automatically highlight unusual log patterns without depending on manually crafted rules or brittle heuristics. Just as importantly, these tools must be efficient and easy to understand, so that alerts can be trusted and acted upon during incident response. Given the high business cost of failures in large-scale storage systems. whether through missed issues or excessive false alarms, finding dependable anomaly detection methods is critical.

While there has been extensive research on log-based anomaly detection, many existing solutions still fall short in practice. Recent advances in machine learning and deep learning have produced powerful models, but they often come with significant drawbacks: high computational demands, complex tuning requirements, and limited transparency in how decisions are made. In operational settings, this lack of interpretability can be just as problematic as poor detection performance. In addition, anomaly thresholds are frequently chosen using fixed values or informal heuristics, with little quantitative justification, making such systems hard to adapt and trust over time.

Another key challenge is that real system logs are usually unlabelled, which makes objective evaluation difficult. Much of the existing work on HDFS logs focuses on supervised or sequence-based methods, while simpler statistical approaches are often treated as baseline comparisons with minimal explanation. As a result, there is limited understanding of how classical, interpretable methods behave when they are carefully tuned, evaluated, and then applied to real, unlabelled data.

This project addresses these issues by systematically exploring four well-established unsupervised anomaly detection techniques: Euclidean distance, Mahalanobis distance, a multivariate Gaussian likelihood score, and Local Outlier Factor (LOF). Raw HDFS logs are first parsed and structured, then transformed into time-windowed event count features following a standard log analysis pipeline. Each method is tuned using a labelled HDFS benchmark dataset, with anomaly thresholds selected by maximising the F1-score on a validation split. This provides a fair and quantitative basis for comparison.

Once tuned, the methods are applied to an unlabelled subset of HDFS logs. In this setting, percentile-based thresholds are used to identify anomalous log windows based on the distribution of anomaly scores. The project covers the full pipeline, from data preprocessing and feature construction to model implementation, evaluation, and visualisation. Overall, the aim is to show that simple, interpretable statistical techniques can form a solid and trustworthy foundation for log-based anomaly detection in large-scale distributed systems.

1.2 Aim

To explore how four classical unsupervised anomaly detection algorithms behave when detecting anomalous windows in HDFS logs, with equal emphasis on benchmark evaluation and practical use.

1.3 Objectives

The objectives of this project are to transform raw HDFS system logs into structured, time-windowed event count features suitable for anomaly detection, and to apply four classical unsupervised anomaly detection techniques, Euclidean distance, Mahalanobis distance, a multivariate Gaussian likelihood score, and Local Outlier Factor, to this data. The project also aims to tune method-specific anomaly thresholds using a labelled HDFS benchmark dataset by maximising the F1-score on a validation split. In addition, it seeks to examine how these tuned methods behave when applied to unlabelled HDFS logs through percentile-based thresholding and qualitative analysis of detected anomalies. Finally, the project aims to compare the approaches in terms of their precision, recall trade-offs, the number of anomalies identified, and their interpretability and suitability for real-world operational use.

1.4 Research Question

This project explores how statistical techniques can be used to improve the detection of abnormal behaviour in HDFS system logs within organisational settings. Building on the work presented in the Interim Project Report (IPR), it focuses on addressing two key research questions:

1. Which types of classical statistical or machine-learning models work best for detecting unusual behaviour in HDFS logs, and how can they be built from the ground up?
2. Can models developed from first principles. such as Euclidean distance, Mahalanobis distance, Gaussian likelihood, and Local Outlier Factor. provide competitive anomaly detection performance compared to traditional fixed thresholds used in log monitoring and simple baselines?

1.5 Novelty

A systematic and transparent approach to evaluating classical statistical anomaly detection methods on HDFS log data. Four widely used statistical detectors are implemented end to end from scratch, rather than relying on black-box libraries, allowing their behaviour and assumptions to be clearly examined. A benchmark-driven strategy is adopted for anomaly threshold selection, using F1-score optimisation on labelled HDFS data instead of arbitrary or heuristic cut-offs. The analysis combines quantitative performance evaluation on a labelled benchmark with qualitative inspection of anomalies detected in unlabelled HDFS logs. This combined perspective highlights precision, recall trade-offs and provides practical insight into how such methods behave in real monitoring scenarios, particularly when compared with more complex deep learning approaches.

1.6 Feasibility, Commercial Context, and Risk

From a technical perspective, the work is feasible as it builds on publicly available HDFS log benchmarks and makes use of well-established statistical techniques that can be applied efficiently to medium-scale datasets. These methods do not require specialised hardware or extensive computational resources, which makes them practical for real-world deployment. From a commercial and economic standpoint, improving anomaly detection in HDFS systems can help reduce downtime, avoid service-level agreement (SLA) penalties, and support more reliable data-driven services. Industry reports suggest that even short system outages can cost large organisations thousands of dollars per minute and may lead to SLA violations or contract breaches.

At the same time, several risks and limitations need to be considered. False positives may increase the workload for system operators and reduce confidence in automated alerts if not carefully managed. In addition, benchmark datasets may not fully represent all production environments, particularly when log formats, workloads, or failure patterns differ from those seen in practice. There is also strong competition from more advanced sequence-based and deep learning approaches, which can capture richer temporal patterns but typically come with greater complexity, higher computational costs, and reduced interpretability.

The operational, commercial, and ethical implications of these risks, including issues related to system reliability and the use of automation, are examined in more detail in the evaluation and conclusion chapters.

1.7 Report Structure

The rest of this report is structured as follows.

- **Chapter 1 – Introduction** Introduces the problem, motivation, objectives, scope of the work, and overall context of log-based anomaly detection in HDFS systems.
- **Chapter 2 – Literature Review** Reviews existing research on log parsing, HDFS benchmark datasets, and classical and modern approaches to log-based anomaly detection.
- **Chapter 3 – Methodology** Explains how the HDFS log data is prepared and transformed into features for analysis and outlines the overall experimental setup. It also describes how the four anomaly detection models are implemented and how their thresholds are tuned using the labelled HDFS dataset.
- **Chapter 4 – Quality & Results** Presents quantitative benchmark results, visualisations of anomaly scores, and qualitative analysis of anomalies detected in unlabelled HDFS logs.
- **Chapter 5 – Evaluation and Conclusion** Discusses the findings, evaluates the methods, addresses ethical and commercial implications, outlines limitations, and suggests future work.
- **Chapter 6- References** This section provides the complete list of sources consulted and cited throughout the report.
- **Chapter 7- Appendices** It includes additional supporting material, such as extra figures, tables, and technical details, for reference.

Chapter2: Literature Review

As modern large-scale systems continue to grow and complexity, automated log analysis has become an essential part of reliability engineering. These systems generate massive amounts of log data every day, far more than can be examined manually. As a result, much of the existing research on log analytics describes a common processing pipeline that starts with parsing and structuring raw logs, followed by feature extraction and anomaly detection. Within this area, the HDFS benchmark dataset has emerged as a widely used reference for evaluating log-based anomaly detection methods.

Once logs are converted into a structured form, a variety of detection techniques can be applied. Traditional statistical approaches, such as Euclidean distance, Mahalanobis distance, and multivariate Gaussian likelihood, identify anomalies by measuring how much a log window differs from what is considered normal system behaviour. Density-based methods like Local Outlier Factor (LOF) look at anomalies from a different angle, comparing the local density of a log window to that of its neighbours. This makes them particularly useful when the data contains clusters of varying density or irregular patterns.

More recent research has increasingly focused on deep learning and sequence-based models that learn normal log event patterns over time, often using recurrent neural networks or transformer-based architectures. These methods can achieve strong results on benchmark datasets, but they come with important trade-offs. They are typically harder to train, require more computational resources, and offer limited transparency into how decisions are made. Several comparative studies have shown that when classical methods are carefully implemented and tuned, they can still perform surprisingly well. However, in many log-based studies, these simpler techniques are treated only as baseline comparisons, with little attention paid to implementation choices, threshold selection, or how well they transfer from labelled benchmarks to unlabelled, real-world logs. This work responds to that gap by taking a closer, more systematic look at classical anomaly detection methods, implementing them from scratch on HDFS event-count features, tuning thresholds using F1-score optimisation on benchmark data and then analysing how they behave when applied to unlabelled HDFS log windows.

2.1 Examination of dataset

Most work on log-based anomaly detection makes use of structured benchmark datasets, where raw system logs are first cleaned up and organised into templates, and each log message is assigned an event identifier. The HDFS benchmark dataset is one of the most used examples. It contains block-level logs collected from a real Hadoop cluster, along with labels that indicate whether each block execution behaved normally or showed signs of a problem. Because these labels are available, researchers can clearly measure how well different anomaly detection methods perform using standard metrics such as precision, recall, and F1-score.

In day-to-day system operations, however, things are rarely this neat. Operational logs are usually unlabelled, and engineers must rely only on timestamps, system components, and observed event patterns to identify potential issues. To better reflect this reality, this work uses both types of data. The labelled HDFS benchmark is used to build features and tune anomaly detection thresholds, while a separate, unlabelled subset of HDFS logs grouped into fixed time windows, is used to explore how the tuned models behave when labels are no longer available. This approach allows the study to balance controlled evaluation with a more realistic view of how these methods would perform in practice.

2.2 Related works

Early research in log analysis was mainly about turning messy, human-written log messages into something that computers could work with at scale. System logs were never designed for automated analysis, so researchers first focused on log parsing and template extraction to convert free-text messages into structured event identifiers (He et al., 2021). This step was crucial, as it made it possible to analyse system behaviour in a consistent way and opened the door to automated log monitoring and anomaly detection.

Once logs could be represented in a structured form, researchers started exploring ways to identify abnormal behaviour. Many of the earliest approaches are based on distance and probability. Methods such as Euclidean distance, Mahalanobis distance, and multivariate Gaussian models learn what normal system activity looks like and then measure how much new observations deviate from that pattern (Chandola et al., 2009; Xu et al., 2018). These techniques are still widely used today because they are simple, efficient, and easy to understand, especially when applied to straightforward features like event counts.

Another line of work looks at anomalies from a more local point of view. Local Outlier Factor (LOF), introduced by Breunig et al. (2000), compares the density around a data point to the densities of its nearest neighbours. This makes it useful when the data contains clusters with very different characteristics. However, later studies point out that LOF often flags many anomalies in noisy or high-dimensional log data. While this helps catch many unusual cases, it can also result in a high number of false alarms, which is a challenge in real monitoring environments (Campos et al., 2016; He et al., 2021).

In recent years, attention has increasingly shifted toward deep-learning and sequence-based approaches. Models such as Deep Log use recurrent neural networks to learn normal sequences of log events and have shown strong results on datasets like HDFS (Du et al., 2017). More recent surveys also highlight the success of transformer-based and representation-learning approaches for log anomaly detection (Zhang et al., 2023). At the same time, these models are often complex, require significant computational resources, and provide limited insight into why a particular log window is flagged as anomalous. Because of this, several studies argue that classical statistical methods can still be very effective when they are carefully implemented and tuned, especially in settings where transparency and operational simplicity matter (He et al., 2021; Zhang et al., 2023).

The HDFS benchmark dataset introduced by Xu et al. (2018) has become one of the most used datasets for evaluating log-based anomaly detection methods. It provides structured logs with block-level anomaly labels, making it possible to compare different techniques using metrics such as precision, recall, and F1-score. Despite this, classical methods are often included only as brief baselines, with little detail on implementation choices, threshold selection, or how models tuned on labelled data behave when applied to unlabelled operational logs (He et al., 2021).

This literature review draws mainly on peer-reviewed journal articles, academic books, and leading conference papers to ensure that the discussion is grounded in reliable and authoritative sources. The studies were selected for their methodological rigour, relevance to log-based anomaly detection, and contribution to ongoing debates in the field. By bringing these works together, the review highlights recurring themes, such as the trade-off between precision and recall, as well as open questions around interpretability and model complexity.

The literature is organised thematically to show how different approaches to log analysis attempt to address the challenges most relevant to this project. Each group of studies is linked back to the central research question, clarifying how prior work shapes the scope and motivation of the investigation. While deep and sequence-based methods dominate recent research, there is relatively little work that takes a careful, transparent look at how classical statistical methods perform on HDFS logs.

Where previous research leaves gaps, especially in the detailed evaluation of classical, interpretable approaches, this project positions itself as a focused response. The hypothesis is grounded in existing literature but aims to extend current practice by revisiting simple, from-scratch statistical models and examining how they behave when tuned on labelled benchmarks and applied to unlabelled operational logs. Overall, this review not only summarises prior work but also connects it clearly to the aims, objectives, and hypothesis of the present study.

2.3 Comparison table of key studies

Study	Dataset	Type of Method	How Performance Is Measured	How It Relates to This Work
Survey on Automated Log Analysis	Multiple system log datasets	Overview of classical and machine-learning approaches	Discusses common benchmarks and evaluation metrics	Sets the overall context for log analysis and supports the use of HDFS as a standard benchmark.
Surveys on Deep Log Anomaly Detection	Log datasets, including HDFS	Deep learning and sequence-based models	Uses metrics such as F1-score and AUC	Shows that deep models perform well but are complex and hard to interpret, motivating a simpler approach here.
Mahalanobis-Based Outlier Detection Studies	Tabular and multivariate data	Statistical distance and probabilistic methods	Threshold-based distance analysis	Supports the use of Mahalanobis distance and Gaussian models on structured log features.

LOF Applications and Reviews	Various anomaly detection domains	Local density-based methods (LOF)	Score-based thresholds, often high recall	Motivates using LOF as a complementary method with different strengths and weaknesses.
Comparative Log Anomaly Detection Studies	HDFS and other log datasets	Classical vs. ML vs. deep methods	Precision, recall, and F1-score on benchmarks	Highlights the importance of fair comparison and threshold tuning, which this work addresses in detail.

Overall, this comparison highlights that although recent research is largely driven by deep and highly sophisticated models, there is still clear value in carefully evaluating classical statistical techniques on standard log datasets. Transparent and reproducible baselines remain important, both for understanding the problem space and for supporting practical deployment. By implementing and comparing four classical methods on HDFS logs, this work aims to offer clear, interpretable reference points and to shed light on their strengths and limitations when viewed alongside more complex approaches.

Chapter 3: Methodology

A practical data-science workflow for log-based anomaly detection. It begins with collecting and cleaning structured HDFS logs and transforming them into numerical representations that capture system behaviour over time, including both block-level and fixed time-window event-count features. Four classical unsupervised anomaly detection methods, Euclidean distance, Mahalanobis distance, a multivariate Gaussian likelihood model, and Local Outlier Factor, are then implemented from scratch and used to assign anomaly scores to each log window. Their performance is evaluated on a labelled HDFS benchmark dataset using precision, recall, and F1-score, and separate detection thresholds are tuned for each method based on these metrics. Finally, the tuned models are applied to unlabelled HDFS logs to study how they behave in more realistic operational settings, alongside a discussion of the ethical, legal, and practical considerations involved in deploying automated anomaly detection systems in real-world environments. *(Key implementation snippets are provided in Appendix 7.1.)*

3.1 Tools and Techniques

All the work is implemented in ***Python 3*** using a small set of well-known data-science libraries. ***Pandas*** is used to load the HDFS logs, clean the data, and perform the grouping and aggregation steps needed for feature construction. ***NumPy*** handles the core numerical operations, such as computing means, covariance matrices, and distance measures. ***Matplotlib*** is used to visualise anomaly scores and timelines, making it easier to understand how the different methods behave. ***Scikit-learn*** is included only for basic utilities, such as splitting data into training and validation sets, and is deliberately not used for anomaly detection itself. This helps keep the implementation transparent and consistent with the goal of building the models from scratch.

This set of tools is widely used in both research and industry, is well supported by existing literature, and is more than sufficient for working with the medium-scale HDFS datasets used in this project. More complex frameworks like Spark or TensorFlow were considered, but they were not necessary given the size of the data and would have added extra complexity without clear benefits, especially given the focus on simplicity, interpretability, and understanding how the methods work.

3.2 Data Collection, Cleaning and Feature Engineering

This project is based on two structured HDFS log datasets: a labelled benchmark dataset with block-level Normal and Anomaly labels, and a separate unlabelled subset that follows the same logging format. Each log entry records basic information such as the timestamp, log level, system component, message content, and an event template or event ID obtained through prior log parsing. The labelled dataset is used to evaluate the anomaly detection methods and tune their thresholds, while the unlabelled dataset reflects a more realistic operational setting where no ground-truth information about anomalies is available.

The data preparation process starts by loading the structured log files into pandas and checking that all columns have consistent data types. Date and time fields are merged into a single timestamp, and any records with missing or invalid timestamps are removed. Missing values in fields such as process ID, log level, and component are handled using explicit placeholders or the most common values, helping to retain potentially useful log entries. Log levels are standardised, duplicate records are removed, and the logs are ordered by time. The cleaned logs are then saved as an intermediate dataset to ensure that the feature engineering and modelling steps can be reproduced consistently.

Feature engineering follows a straightforward event-count approach that is widely used in log-based anomaly detection. For the labelled benchmark dataset, log entries are grouped by block ID so that each block represents a single execution instance. The frequency of each event ID within a block is then counted, resulting in a block-by-event feature matrix, with the corresponding Normal or Anomaly label attached. For the unlabelled dataset, logs are instead grouped into fixed-length time windows, such as five-minute intervals, using their timestamps. Within each window, counts of event IDs and log levels are computed to form a window-by-event feature matrix. These event-count representations provide a clear and interpretable view of system behaviour, capturing how often different events occur over a given block or time window, and are well suited to the classical statistical models used later in the study.

3.3 Statistical Models for Anomalies detection

This project uses four well-established unsupervised statistical methods to assign an anomaly score to each HDFS log block or time window. All four models work on the same event-count feature representations and are implemented from scratch in Python, which makes their behaviour easy to inspect and compare. Together, they reflect different ways of thinking about anomalies, looking at overall distance from normal behaviour, modelling probability, and examining local density, which allows a clear comparison of how each method responds to unusual patterns in HDFS log data.

The choice of these methods is supported by a wide range of surveys and comparative studies that consistently identify distance- and density-based techniques as strong baseline approaches for log-based and multivariate anomaly detection. Euclidean and Mahalanobis distance provide simple, intuitive measures of how much a log window deviates from normal behaviour, while multivariate Gaussian models add a probabilistic view of anomalies. Local Outlier Factor complements these global approaches by detecting local irregularities that may be missed when using a single global model. Compared with deep-learning techniques, these methods require less data, involve fewer tuning decisions, and make it easier to understand why a particular log window is flagged as anomalous. Their mathematical simplicity also makes them suitable for implementation from the ground up, aligning with the requirements set out in the Interim Project Report and supporting their use in practical system-monitoring environments.

3.3.1 Euclidean Distance

Euclidean distance measures how far a log feature vector $x$is from the average pattern of normal behaviour. First, a mean feature vector $\mu$is computed from the training data to represent typical system activity. The anomaly score for a given log window is then calculated as

$$\boldsymbol{d_E(x) = \| x - \mu \|_2}$$

This value represents the straight-line distance between the observed event-count pattern and the normal profile in feature space. A small distance indicates behaviour close to normal, while a larger distance suggests a stronger deviation and a higher likelihood of anomalous behaviour. Euclidean distance is widely used as a baseline anomaly detector because it is simple to compute, easy to interpret, and does not rely on additional parameters or complex modelling assumptions.

3.3.2 Mahalanobis Distance

While Euclidean distance treats each feature independently, Mahalanobis distance considers the relationships between features by incorporating the covariance structure of the data. This makes it particularly suitable for multivariate log features where event counts may be correlated. After computing the mean feature vector $\mu$and the covariance matrix $\Sigma$from the training data, the anomaly score for a feature vector $x$is defined as

$$\boldsymbol{d_M(x) = (x - \mu)^T \Sigma^{-1} (x - \mu)}$$

This score measures how far $x$lies from the centre of the data when distances are adjusted for both the variance of individual features and their correlations with one another. In this transformed space, directions with high variance contribute less to the distance, while unusual combinations of correlated features stand out more clearly. As a result, points that lie far from the main multivariate cluster of normal behaviour receive large Mahalanobis distances and are flagged as potential anomalies. Mahalanobis distance is a well-established method for multivariate outlier detection and is closely related to probabilistic Gaussian models.

3.3.3 Multivariate Gaussian Likelihood

If the event-count feature vectors representing normal behaviour are assumed to follow a multivariate Gaussian distribution, the same mean vector $\mu$and covariance matrix Σcan be used to define a probabilistic anomaly score. Rather than measuring distance directly, this approach evaluates how likely a given observation is under the fitted Gaussian model. The anomaly score for a feature vector $x$is computed as the (negative) quadratic form

$$\boldsymbol{s_G(x) = -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

where constant terms of the Gaussian likelihood are omitted since they do not affect ranking. This score reflects how probable $x$is according to the learned normal distribution: higher values indicate more typical behaviour, while very low values correspond to patterns that are unlikely under the model. Log windows with low likelihood scores are therefore treated as anomalous. This Gaussian likelihood approach is widely used in multivariate anomaly detection because it offers a clear probabilistic interpretation—anomalies are simply those observations whose probability of occurring under normal system behaviour falls below a chosen threshold.

3.3.4 Local Outlier Factor (LOF)

Local Outlier Factor (LOF) is a density-based anomaly detection method that identifies unusual points by comparing their local density to that of their surrounding neighbours. Instead of relying on a single global notion of normal behaviour, LOF focuses on how isolated a point is within its immediate neighbourhood. For each data point x, the method first identifies its k-nearest neighbours and computes a local reachability density (LRD), which reflects how densely the point is surrounded by other points.

The LOF score is then defined as

$$\textbf{LOF}_k(x) = \frac{\textbf{average LRD of } x\textbf{'s neighbours}}{\textbf{LRD of } x}$$

This ratio compares the density around $x$to the densities around its neighbours. A score close to 1 indicates that the point lies in a region of similar density and is likely normal. Values significantly greater than 1 suggest that the point is in a much sparser region than its neighbours and is therefore a potential outlier. Unlike global distance-based methods, LOF can detect anomalies that are only locally unusual, even if they do not appear extreme when viewed against the entire dataset. This makes it particularly useful for log data with heterogeneous patterns or clusters of varying density.

For each of the four models, the anomaly scores are eventually turned into simple yes-or-no anomaly decisions using thresholds chosen in a consistent way. These thresholds are first learned from the labelled HDFS benchmark and then applied to the unlabelled HDFS logs using percentile-based rules. Using the same scoring and thresholding process for all methods makes the comparison fair and straightforward, allowing their precision, recall trade-offs and real-world usefulness for HDFS log monitoring to be clearly assessed.

3.4 Test strategy, Results and Validation

The testing strategy in this work follows a two-layer approach that combines quantitative evaluation on labelled data with qualitative analysis on unlabelled logs. Evaluation begins with the labelled HDFS benchmark dataset, where each block is already marked as normal or anomalous. The data are split into training and validation sets using stratified sampling so that the proportion of anomalies remains consistent across both subsets. Each model is fitted using only the training data, and anomaly scores are then computed on the validation set. To convert these scores into anomaly decisions, a range of candidate thresholds is explored using score percentiles. For each threshold, predictions are generated and evaluated using precision, recall, and F1-score, and the threshold that maximises F1 on the validation set is selected as the operating point for that model. This provides a clear, data-driven alternative to arbitrary threshold selection.

Before analysing performance, basic correctness checks are carried out to ensure the implementation behaves as expected. These include verifying matrix dimensions, confirming that distance values are non-negative, and inspecting a small number of manually computed examples. The main quantitative results on the benchmark dataset consist of precision, recall, and F1-scores at the tuned thresholds, along with confusion matrix counts. These results help reveal how conservative or aggressive each method is in practice. The benchmark findings show that Euclidean distance, Mahalanobis distance, and the Gaussian likelihood model tend to produce relatively small sets of high-confidence anomalies, while Local Outlier Factor identifies a much larger number of anomalies. Although LOF achieves high recall, this comes at the cost of lower precision, reflecting a clear trade-off between sensitivity and false alarms.

After threshold tuning, the same models are applied to the unlabelled HDFS dataset, where logs are represented as fixed-length time windows. Since no ground-truth labels are available, performance metrics such as F1-score cannot be computed. Instead, threshold selection is guided by the percentile positions identified during benchmark tuning. For example, if a method performs best when flagging the top few percent of scores on the benchmark data, a similar percentile is used when analysing the unlabelled logs. Results on the unlabelled dataset are therefore presented in terms of the number of anomalies detected by each method, the degree of overlap between methods, and a closer inspection of log windows flagged by multiple models. Windows identified as anomalous by all four methods are examined in detail to understand the underlying event patterns.

Validation focuses on ensuring that the observed results are meaningful and not artefacts of arbitrary design choices. On the labelled benchmark, the use of a separate validation set for threshold selection and the reporting of multiple metrics provide a basic level of internal validation. Sensitivity to threshold choice is further explored by analysing how performance changes across different percentile cut-offs. Where possible, the observed precision–recall trade-offs are compared qualitatively with findings reported in existing HDFS and log-anomaly studies, which often describe similar behaviour for global distance-based methods versus local density-based approaches. On the unlabelled dataset, validation is necessarily more qualitative. Log windows flagged by multiple methods are examined directly in the raw logs to confirm that they correspond to unusual event combinations or bursts of specific event IDs. This cross-method agreement and manual inspection help increase confidence that the models are identifying genuinely unusual system behaviour rather than random noise.

3.5 Ethical, Legal, Social, and Professional Issues

The ethical and legal concerns associated with this work are minimal, as it uses only system-level logs from benchmark datasets and controlled test environments. No personal data or information that could identify individual users is involved. However, if models like these were to be used in real production systems, there would still be important responsibilities to consider. False positives could overwhelm operators with unnecessary alerts, while false negatives could allow real issues to go unnoticed. Both situations can reduce trust in monitoring systems and have real operational consequences.

For this reason, transparency and human oversight are essential. System operators need to understand why a particular log window has been flagged so they can judge its significance and respond appropriately. This project addresses these concerns by deliberately using simple, interpretable models, openly discussing the trade-offs between precision and recall, and treating anomaly detection as a support tool rather than a fully automated decision-making system. In doing so, the work promotes responsible use of anomaly detection while keeping people firmly in the loop.

3.6 Addressing the issues

The ethical and legal risks associated with this work are largely addressed through the choice of data and the way the models are used. Only public HDFS benchmark logs and internal system logs are analysed, and these contain purely technical information such as timestamps, system components, and event templates. No personal data or user-identifiable information is involved, and there is no attempt to link log entries to individuals or external systems. As a result, the privacy impact is minimal, and formal ethics approval is not required under typical institutional guidelines.

The project also places a strong emphasis on transparency. By using interpretable statistical methods, Euclidean distance, Mahalanobis distance, Gaussian likelihood, and Local Outlier Factor, the reasons why a particular log window is flagged can be explained in simple terms, such as unusual distances or low local density, rather than opaque model internals. This supports professional accountability by allowing operators to understand and verify alerts against the raw logs.

In addition, all evaluation is carried out offline using labelled benchmark data before any consideration of real-world use. Anomaly thresholds are chosen using a quantitative, F1-score, based approach, and model behaviour on unlabelled logs is assessed through qualitative inspection, particularly for windows flagged by multiple methods. Finally, the system is clearly framed as a decision-support tool rather than an automated response mechanism. Its role is to highlight suspicious log windows for further human investigation, which helps reduce the risk of harm caused by false positives or missed anomalies in operational environments.

Chapter 4: Quality and Results

It brings together the results of the anomaly detection experiments and looks at how well the four statistical models meet the goals of the project. Using the labelled HDFS benchmark, it reports precision, recall, and F1-scores for Euclidean distance, Mahalanobis distance, multivariate Gaussian likelihood, and Local Outlier Factor at their tuned thresholds, supported by tables and visualisations. These results make it clear that the different methods behave in different ways, particularly in terms of the balance between precision and recall, and they are discussed in relation to findings from previous work on log analysis.

The section then turns to the unlabelled HDFS logs to see how the same models behave in a more realistic setting where no ground-truth information is available. The analysis focuses on how many anomalies each method detects, how much overlap there is between the different detectors, and how anomaly scores change over time. Log windows that are flagged by several methods are examined more closely by inspecting the raw logs. Together, these results help show where interpretable statistical methods work well, where they fall short, and how practical they are for real-world HDFS log monitoring.

4.1 Dataset Description

The structured log data generated from a production-like Hadoop Distributed File System (HDFS) environment. Rather than working with raw log text, the logs have already been parsed into a structured format where each row corresponds to a single log event. Each entry contains fields such as the event timestamp, log level, system component, message content, and an event identifier derived from log templates. The date and time fields are combined into a single timestamp, allowing the logs to be ordered chronologically and grouped into fixed time windows. This structured format makes it easier to analyse system behaviour and extract meaningful features for anomaly detection.

4.1.1 Labelled HDFS Benchmark Dataset

The first dataset is a labelled HDFS benchmark that is commonly used in log anomaly detection research. In this dataset, log entries are grouped into execution units, such as HDFS blocks, and each unit is labelled as either *Normal* or *Anomalous*. For each execution unit, event-count features are constructed by counting how often different event IDs and log levels occur. This produces a compact feature representation for each block, along with a corresponding ground-truth label.

This benchmark dataset is used for quantitative evaluation and model tuning. Because the true anomaly labels are available, it allows standard performance metrics such as precision, recall, and F1-score to be calculated. It is also used to select appropriate anomaly thresholds in a principled way, by choosing thresholds that maximise F1-score on a validation split.

4.1.2 Unlabelled HDFS Log Subset

To reflect a more realistic operational setting, the study also uses an unlabelled subset of HDFS logs drawn from the same structured log source. This dataset covers a continuous period of system activity and contains 2,000 structured log entries. Since no ground-truth anomaly labels are available, this data more closely resembles real-world monitoring scenarios.

The logs are grouped into fixed 5-minute time windows using their timestamps. For each window, the number of occurrences of each event ID (E1–E14) and log level is counted, along with an additional feature that records the total number of log events in that window. This results in a window-level feature matrix with 454-time windows and 17 numeric features. These count-based features provide an intuitive view of how system activity changes over time and serve as the input to all four anomaly detection models.

4.1.3 How the Two Datasets Fits Together

The two datasets serve complementary purposes. The labelled HDFS benchmark provides a controlled environment for evaluating and tuning the anomaly detection methods, while the unlabelled dataset allows the tuned models to be tested in a more realistic, label-free setting. Thresholds learned from the benchmark are transferred to the unlabelled data using percentile-based rules, after which anomalous windows are identified and inspected in the raw logs. Together, these datasets support both rigorous evaluation and practical assessment of how interpretable statistical methods behave when applied to real HDFS system logs.

4.2 Preprocessing the Data

Before any modelling is carried out, the features extracted from the HDFS logs are cleaned up further to make sure they are consistent and easy to work with across all experiments. Only numeric features are kept for analysis, including counts of log levels (such as INFO and WARNING), counts of each event ID (E1–E14), and the total number of log events in each time window. Non-numeric information, such as component names, message text, or raw timestamps, is left out at this stage, as it cannot be used directly by distance- or density-based statistical models. This results in a clean, purely numeric feature matrix that is well suited to computing anomaly scores.

To make sure that no single feature dominates the analysis simply because it occurs more often than others, all numeric features are standardised using z-score normalisation. This step brings every feature onto the same scale, with values centred around zero and similar levels of variation. The same standardisation process is applied consistently to both the labelled benchmark data and the unlabelled time-windowed data, ensuring that all four models operate on comparable inputs. The final, normalised feature matrices are then used directly by the Euclidean distance, Mahalanobis distance, Gaussian likelihood, and Local Outlier Factor models in the remainder of the study.

4.3 Exploratory Data Analysis (EDA)

The structure of the HDFS feature matrices is first explored. In the time-windowed dataset, simple descriptive statistics reveal that most event and log-level counts are highly skewed. Many time windows contain no occurrences of rare events, while a small number show sharp spikes in activity. These spikes are particularly noticeable for INFO messages, certain event IDs, and the overall TotalCount. Histograms and boxplots make these long-tailed distributions clear, highlighting a pattern of sparse activity punctuated by occasional bursts. This suggests that any anomaly detection method needs to cope well with both sparse data and sudden increases in log volume.

A similar exploratory analysis of the labelled HDFS benchmark highlights a strong class imbalance, with normal execution units greatly outnumbering anomalous ones. Because of this imbalance, simple accuracy would be misleading, which motivates the use of precision, recall, and F1-score as evaluation metrics. Visualisations of basic activity-related features, such as total event counts per block or per window, also show that anomalies tend to occur at unusually high or unusually low activity levels compared to the bulk of the data. These observations support the use of global distance-based methods like Euclidean and Mahalanobis distance, as well as probabilistic thresholds based on Gaussian likelihoods. Overall, the insights gained from this exploratory analysis help guide the choice of percentile-based thresholds and provide useful context for understanding why different methods flag different numbers of anomalies in the HDFS logs.

4.4 Model Selection and Assessment

All four models are trained on part of the data and then tested on a separate validation set, where their performance is measured using precision, recall, and F1-score. Thresholds are chosen specifically to give the best F1-score for each method. When looking at the results, a clear pattern emerges. The Euclidean distance, Mahalanobis distance, and multivariate Gaussian models tend to be quite conservative. They usually flag only a small number of log windows as anomalous, but when they do, those cases are typically strong and obvious anomalies. This leads to high precision, but it also means that many more subtle or borderline issues are missed, resulting in lower recall.

The Local Outlier Factor model behaves very differently. At its tuned threshold, LOF tends to flag many more windows as anomalous, especially in areas of the feature space where data points are sparse. This allows it to catch a larger share of potential anomalies, giving it higher recall, but it also generates many more false alarms, which lowers its precision. In practical terms, LOF is much more aggressive than the global models.

Taken together, these results clearly show the trade-off between precision and recall that is often discussed in anomaly detection research. The global distance- and probability-based models favour precision and produce fewer, more trustworthy alerts, while LOF favours coverage and casts a wider net. There is no single “best” method across all situations. Instead, the right choice depends on what matters most in an HDFS monitoring setup, whether the priority is to avoid alert overload or to ensure that as many potential problems as possible are brought to an operator’s attention.

4.5 Technical Challenges and Solution

Several technical challenges emerged while working through this project, particularly during data preparation, model implementation, and evaluation. One of the first issues involved cleaning the raw HDFS logs. The logs contained separate date and time fields, along with a small number of malformed entries, which made consistent ordering difficult. This was resolved by carefully merging the date and time into a single timestamp and removing invalid rows, resulting in a clean and reliable dataset for analysis.

Feature construction introduced further difficulties. Representing system behaviour using fixed-length, event-count windows led to sparse and high-dimensional feature matrices, with many events occurring only rarely. To manage this, the analysis was restricted to numeric features, which were then standardised to ensure that no single feature dominated the results. This helped stabilise the models while still preserving meaningful variations in system activity.

Implementing some of the anomaly detection methods from scratch also proved challenging. In particular, Mahalanobis distance and Local Outlier Factor require repeated distance calculations, which become increasingly expensive as the dataset grows. This constraint influenced several design decisions, including keeping the feature representation compact and prioritising clarity and efficiency over more complex modelling choices.

Choosing appropriate anomaly thresholds was one of the most difficult aspects of the project. Small changes in threshold values often resulted in large swings in the number of detected anomalies, and without reliable labels in the original data, it was unclear how to judge performance. To overcome this, a labelled HDFS benchmark dataset was introduced following supervisor guidance. This allowed standard metrics such as precision, recall, and F1-score to be used for threshold tuning, providing a more confident basis for evaluation before applying the same models to unlabelled logs.

4.6 Result Comparison

The results are best understood by looking at both the numbers from the labelled benchmark and the patterns observed in the unlabelled HDFS logs. On the benchmark dataset, the performance of Euclidean distance, Mahalanobis distance, the Gaussian likelihood model, and Local Outlier Factor is summarised using F1-scores and simple anomaly counts. A clear trend emerges from these results. The three global methods behave very similarly: they flag only a small number of anomalies and tend to do so with reasonable confidence, leading to modest but stable F1-scores. LOF, on the other hand, marks many more instances as anomalous, but this comes with lower precision, meaning a larger share of its alerts are false positives. This contrast clearly illustrates the trade-off between conservative and aggressive detection strategies when working with HDFS logs.

When the same models are applied to the unlabelled HDFS data, the overall picture remains consistent. LOF again flags the largest number of time windows, while Euclidean distance, Mahalanobis distance, and the Gaussian model often agree on a small core set of windows that stand out clearly from the rest. Examining the overlap between methods shows that windows flagged by all four models are relatively rare but particularly interesting. When these windows are inspected in the raw logs, they usually correspond to bursts of activity or unusual combinations of events, suggesting that they are strong candidates for genuinely abnormal behaviour.

Taken together, these quantitative and qualitative observations help explain how the different methods behave in practice. The global models provide fewer but more interpretable and high-confidence alerts, while LOF offers broader coverage at the cost of generating more false alarms. These findings align closely with the project’s goal of understanding the strengths and limitations of classical, interpretable anomaly detection methods, and show why combining multiple approaches can give a more balanced and informative view of HDFS system behaviour.

Chapter 5: Evaluation and Conclusion

This final section looks back at how well the project achieved its original aims and what the results tell us about using classical, unsupervised methods to detect anomalies in HDFS logs. The results from the labelled HDFS benchmark show a clear pattern. Euclidean distance, Mahalanobis distance, and the multivariate Gaussian model behave in a conservative way: they raise fewer alerts and tend to be precise, but they also miss many of the more subtle anomalies. Local Outlier Factor behaves very differently, flagging a much larger number of cases and achieving higher recall, but at the cost of many false alarms. This contrast reflects the well-known trade-off in anomaly detection between cautious, global models and more aggressive, local-density approaches.

When these tuned models are applied to the unlabelled HDFS logs, the same behaviour carries over. The global distance and probabilistic methods repeatedly agree on a small set of log windows that clearly stand out from normal system behaviour, while LOF highlights many additional windows, some of which appear to reflect noise or minor variations rather than real problems. Looking more closely at the windows flagged by all four methods often reveals clear bursts of activity or unusual event combinations. These cases stand out as strong anomaly candidates and suggest that combining different perspectives can help operators decide where to focus their attention.

Overall, this project shows that simple, from-scratch statistical methods can still play a useful role in HDFS log monitoring when they are applied carefully. By tuning thresholds on a labelled benchmark and transferring them to unlabelled data in a consistent way, these methods provide an interpretable layer of decision support that can help guide human investigation. While they cannot fully replace more advanced sequence-based or deep learning models in complex environments, they offer a transparent and practical starting point and can complement more sophisticated techniques in real-world monitoring systems.

5.1 Future works

There are many ways this work could be taken further. One obvious direction is to move beyond simple event-count features and look more closely at the order in which log events occur. Sequence-based representations, such as n-grams or learned embeddings of event sequences, could capture temporal patterns that are lost when logs are reduced to aggregate counts. These richer representations could either be used with the same classical statistical models explored in this project or serve as inputs to more advanced approaches, including LSTM-based models, transformers, or newer contrastive-learning techniques that have shown promise in recent studies.

Another area for future improvement is scalability. The from-scratch implementations of Mahalanobis distance and LOF used here work well on medium-sized datasets, but applying them to large, real production systems would require more efficient solutions. This might involve approximate nearest-neighbour methods, incremental updates to model parameters, or the use of distributed processing frameworks. There is also scope to make the system more adaptive by incorporating limited feedback from operators, allowing thresholds or models to be adjusted over time in a semi-supervised way that better reflects how monitoring tools are used in practice.

Finally, it would be valuable to place these classical methods in a broader context by comparing them more systematically with modern deep-learning and large language model–based approaches across multiple log datasets. Doing so would help clarify where simple, interpretable methods are still effective and where more complex models genuinely add value. This kind of comparison could provide clearer guidance for practitioners deciding how much complexity is really needed for reliable log anomaly detection in real-world systems.

5.2 Limitations

Like any practical study, this work comes with several limitations. To begin with, the experiments are based on relatively small and well-structured HDFS datasets that use simple event-count features. While this makes the analysis manageable and interpretable, it does not fully reflect the scale or complexity of logs generated by large production clusters. Aggregating logs into fixed 5-minute windows also means that the exact order of events is lost, so anomalies that depend on specific sequences or rare ordering patterns may not be captured.

There are also limitations related to the modelling choices. The classical statistical methods used here rely on assumptions about the structure of the data and on distance and covariance estimates that can become less reliable when features are very sparse or high-dimensional. In addition, the from-scratch implementation of Local Outlier Factor is computationally expensive, scaling poorly as the number of log windows grows. Without further optimisation, this makes it difficult to use LOF for continuous or real-time monitoring in large systems.

Finally, the evaluation on the unlabelled HDFS data is necessarily indirect. Without ground-truth labels, conclusions have to be drawn from score patterns, overlaps between methods, and manual inspection of selected log windows. While this offers useful qualitative insight, it also means that claims about real-world performance should be treated with care. Overall, these limitations highlight the need to view the results as illustrative rather than definitive, and they point to clear opportunities for improvement in future work.

5.3 Project Management Reflection

The project was approached in a series of manageable stages rather than being treated as a single, fixed task. The work began with cleaning the HDFS logs and constructing reliable timestamps, before moving on to window-based feature engineering and the implementation of several classical anomaly-detection methods. Early milestones were deliberately practical, focusing on getting a complete processing pipeline working on the raw logs before gradually adding Euclidean, Mahalanobis, Gaussian, and LOF models and comparing their behaviour.

As the project progressed, it became clear that some adjustments were needed. The original labels, which were largely based on WARNING log events, did not provide a strong enough signal for reliable threshold tuning or meaningful performance evaluation. Following discussion and guidance from the supervisor, the project plan was revised to include a labelled HDFS benchmark dataset. This change significantly improved the quality of the evaluation by allowing precision, recall, and F1-score to be computed in a principled way.

Importantly, this adjustment did not increase the project scope beyond what was realistic. Instead of expanding to multiple datasets or complex deep-learning architectures, the focus was kept on a small set of interpretable classical models and a single log source. This helped ensure that the project remained achievable within the available time and resources while still delivering meaningful and well-supported results.

5.4 Key Insights Gained

Working through this project led to several insights that only became clear through hands-on experimentation rather than from the literature alone. From a technical point of view, the work highlighted how different classical anomaly-detection methods make very different trade-offs. Distance- and density-based scores were often highly sensitive to threshold choices, and while simpler statistical models were easier to understand and explain, they could miss more subtle forms of abnormal behaviour that richer models are designed to capture.

One of the most important research lessons came from working with different types of labels. Using weak or noisy labels made it difficult to evaluate performance in a meaningful way, particularly when tuning thresholds. Introducing a benchmark dataset with reliable ground-truth labels made a noticeable difference, allowing precision, recall, and F1-score to be used with much greater confidence and making model comparisons far more trustworthy.

The project also reinforced a practical insight about how anomaly detection systems are best used in real settings. Rather than acting as fully automated decision-makers, log-based anomaly detectors work most effectively as decision-support tools. Having human operators review flagged log windows, adjust thresholds over time, and interpret results using domain knowledge proved to be just as important as the choice of algorithm itself.

Chapter 6: References

Du, M., Li, F., Zheng, G. and Srikumar, V. (2017) *DeepLog: Anomaly detection and diagnosis from system logs through deep learning*. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS). New York: ACM, pp. 1285–1298. Available at: https://dl.acm.org/doi/10.1145/3133956.3134015 (Accessed: 20th October 2025).

Fu, Q., Lou, J.G., Wang, Y. and Li, J. (2009) ‘Execution anomaly detection in distributed systems through unstructured log analysis’, *Proceedings of the 2009 IEEE International Conference on Data Mining (ICDM)*. Miami, FL: IEEE, pp. 149–158. Available at: https://ieeexplore.ieee.org/document/5360315 (Accessed: 20th October 2025).

Jia, Y., Zhang, Y., Wu, J. and Wang, Y. (2024) ‘Distributed log-driven anomaly detection system based on continual learning’, *arXiv preprint*. Available at: https://arxiv.org/html/2406.07976v1 (Accessed: 20th October 2025).

Khan, Z.A., Yang, M., He, S. and Jiang, J. (2024) ‘Impact of log parsing on deep learning-based anomaly detection’, *Empirical Software Engineering*, 29, 87. Available at: https://eprints.whiterose.ac.uk/id/eprint/216228/1/s10664-024-10533-w.pdf (Accessed 20th October 2025).

Khan, Z.A., He, S., Yang, M. and Jiang, J. (2024) ‘An empirical study of the impact of log parsers on anomaly detection methods’, *Empirical Software Engineering*, 29, 41. Available at: https://yanmeng.github.io/papers/EMSE221.pdf (Accessed: 20th October 2025).

Landauer, M., Skopik, F. and Wurzenberger, M. (2023) ‘Deep learning for anomaly detection in log data: A survey’, *Journal of Systems and Software*, 201, 111666. Available at: https://www.sciencedirect.com/science/article/pii/S2666827023000233 (Accessed: 3rd November 2025).

Lestander, T. (2025) *Anomaly Detection in Large Scale Log Data*. MSc thesis. Umeå University. Available at: https://umu.diva-portal.org/smash/get/diva2:1983804/FULLTEXT01.pdf (Accessed: 3rd November 2025).

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D. and Liu, Y. (2019) ‘LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs’, *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. Macao: IJCAI, pp. 4739–4745. Available at: https://www.ijcai.org/proceedings/2019/0660.pdf (Accessed: 3rd November 2025).

Phillips, E. (2021) *Automated anomaly detection on HDFS log files*. GitHub repository. Available at: https://github.com/ericphillips99/hdfs_log_anomaly_detection (15th November 2025).

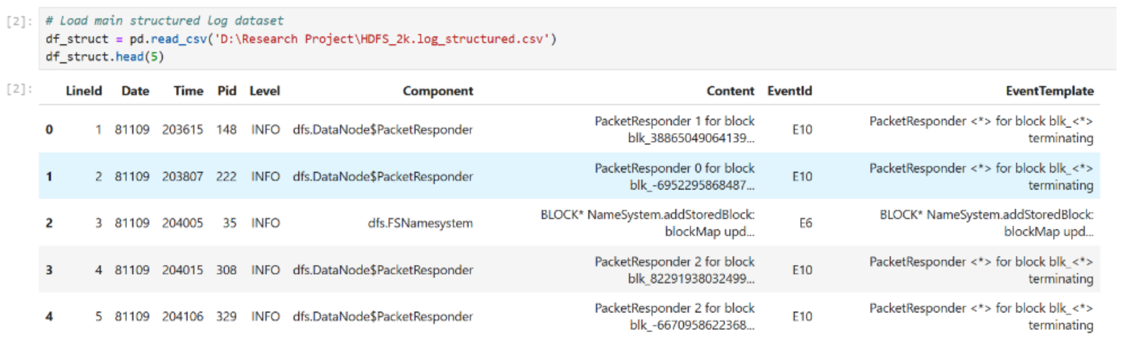
Wei, X., Zhou, W., Liu, Y. and Li, D. (2023) ‘Log-based anomaly detection for distributed systems: State of the art, industry experience, and open issues’, *Journal of Software: Evolution and Process*, 35(7), e2650. Available at: https://onlinelibrary.wiley.com/doi/10.1002/smr.2650 (Accessed: 15th November 2025).

Wu, J., Zhang, Y., Wang, Y. and Pei, D. (2025) ‘System log anomaly detection based on contrastive learning and semantic features’, *Scientific Reports*, 15, 2208. Available at: https://www.nature.com/articles/s41598-025-22208-7 (Accessed: 15th November 2025).

Zhang, D., Chen, Q., Lin, J. and Chen, J. (2023) *Drill: Log-based anomaly detection for large-scale distributed systems*. Proceedings of the 45th International Conference on Software Engineering (ICSE). Melbourne: IEEE/ACM, pp. 1525–1536. Available at: https://ieeexplore.ieee.org/document/10177460 (Accessed: 11th December 2025).

Zhang, D., Lin, J., Chen, Q. and Chen, J. (2024) ‘System logs anomaly detection: Are we on the right path?’, *Applied Artificial Intelligence*, 38(1), pp. 1–23. Available at: https://www.tandfonline.com/doi/full/10.1080/08839514.2024.2440692 (Accessed: 11th December 2025).

Chapter 7: Appendices

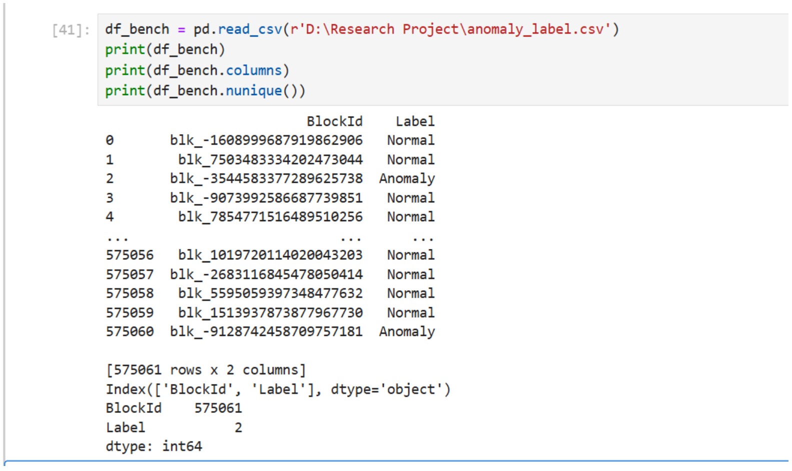


```
[2]: # Load main structured log dataset
     df_struct = pd.read_csv('D:\Research Project\HDFS_2k.log_structured.csv')
     df_struct.head(5)
```

[2]:

	LineId	Date	Time	Pid	Level	Component	Content	EventId	EventTemplate
0	1	81109	203615	148	INFO	dfs.DataNode$PacketResponder	PacketResponder 1 for block blk_38865049064139...	E10	PacketResponder <*> for block blk_<*> terminating
1	2	81109	203807	222	INFO	dfs.DataNode$PacketResponder	PacketResponder 0 for block blk_-6952295868487...	E10	PacketResponder <*> for block blk_<*> terminating
2	3	81109	204005	35	INFO	dfs.FSNamesystem	BLOCK* NameSystem.addStoredBlock: blockMap upd...	E6	BLOCK* NameSystem.addStoredBlock: blockMap upd...
3	4	81109	204015	308	INFO	dfs.DataNode$PacketResponder	PacketResponder 2 for block blk_82291938032499...	E10	PacketResponder <*> for block blk_<*> terminating
4	5	81109	204106	329	INFO	dfs.DataNode$PacketResponder	PacketResponder 2 for block blk_-6670958622368...	E10	PacketResponder <*> for block blk_<*> terminating

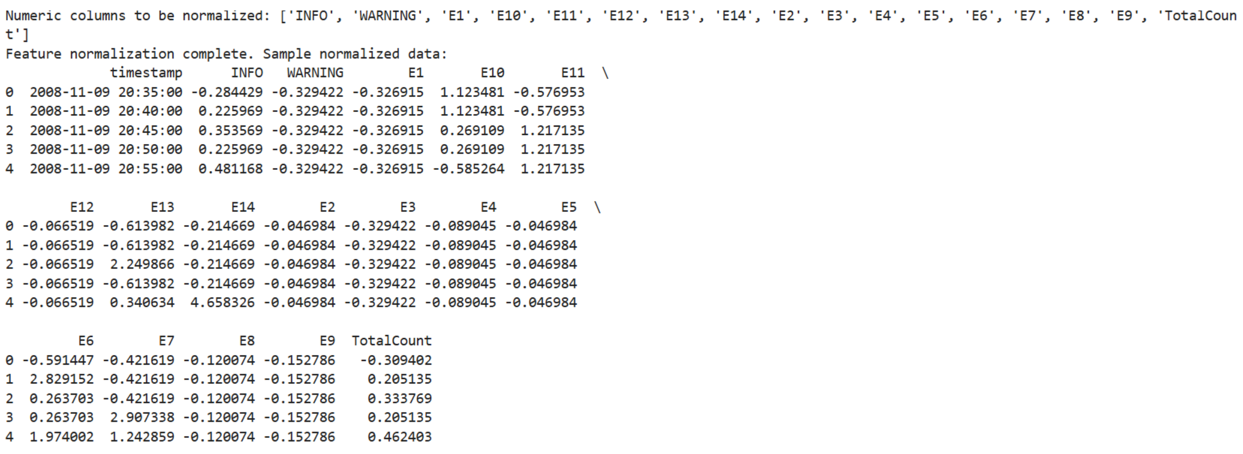
Unlabelled Dataset



```
[41]: df_bench = pd.read_csv(r'D:\Research Project\anomaly_label.csv')
      print(df_bench)
      print(df_bench.columns)
      print(df_bench.nunique())
```

```
                     BlockId    Label
0       blk_-1608999687919862906   Normal
1        blk_7503483334202473044   Normal
2       blk_-3544583377289625738  Anomaly
3       blk_-9073992586687739851   Normal
4        blk_7854771516489510256   Normal
...                          ...      ...
575056   blk_1019720114020043203   Normal
575057  blk_-2683116845478050414   Normal
575058   blk_5595059397348477632   Normal
575059   blk_1513937873877967730   Normal
575060  blk_-9128742458709757181  Anomaly

[575061 rows x 2 columns]
Index(['BlockId', 'Label'], dtype='object')
BlockId    575061
Label           2
dtype: int64
```

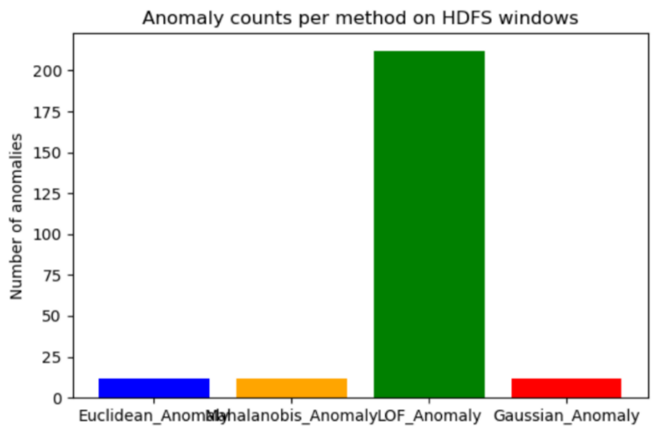
Labelled Dataset



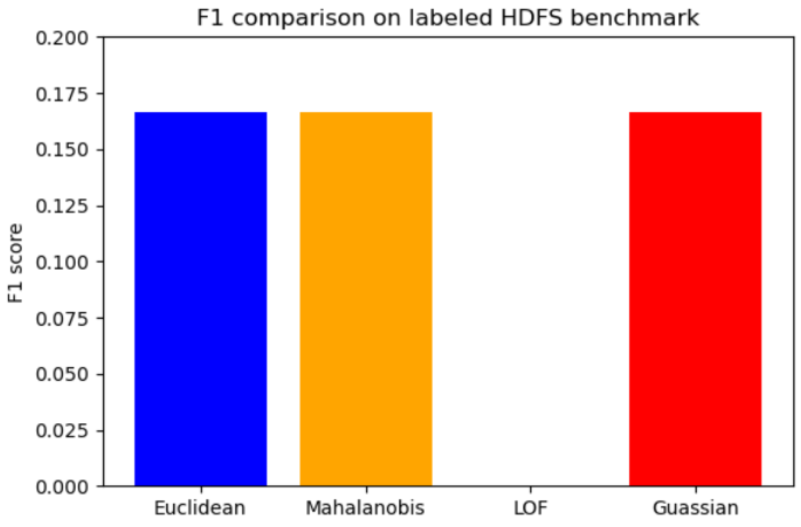
```
Numeric columns to be normalized: ['INFO', 'WARNING', 'E1', 'E10', 'E11', 'E12', 'E13', 'E14', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'E9', 'TotalCoun
t']
Feature normalization complete. Sample normalized data:
             timestamp      INFO   WARNING        E1       E10       E11  \
0  2008-11-09 20:35:00 -0.284429 -0.329422 -0.326915  1.123481 -0.576953
1  2008-11-09 20:40:00  0.225969 -0.329422 -0.326915  1.123481 -0.576953
2  2008-11-09 20:45:00  0.353569 -0.329422 -0.326915  0.269109  1.217135
3  2008-11-09 20:50:00  0.225969 -0.329422 -0.326915  0.269109  1.217135
4  2008-11-09 20:55:00  0.481168 -0.329422 -0.326915 -0.585264  1.217135

        E12       E13       E14        E2        E3        E4        E5  \
0 -0.066519 -0.613982 -0.214669 -0.046984 -0.329422 -0.089045 -0.046984
1 -0.066519 -0.613982 -0.214669 -0.046984 -0.329422 -0.089045 -0.046984
2 -0.066519  2.249866 -0.214669 -0.046984 -0.329422 -0.089045 -0.046984
3 -0.066519 -0.613982 -0.214669 -0.046984 -0.329422 -0.089045 -0.046984
4 -0.066519  0.340634  4.658326 -0.046984 -0.329422 -0.089045 -0.046984

         E6        E7        E8        E9  TotalCount
0 -0.591447 -0.421619 -0.120074 -0.152786   -0.309402
1  2.829152 -0.421619 -0.120074 -0.152786    0.205135
2  0.263703 -0.421619 -0.120074 -0.152786    0.333769
3  0.263703  2.907338 -0.120074 -0.152786    0.205135
4  1.974002  1.242859 -0.120074 -0.152786    0.462403
```

Feature Normalization's output

Method	Count	Threshold
Euclidean_Anomaly	12	97.5
Mahalanobis_Anomaly	12	97.5
Gaussian_Anomaly	12	97.5
LOF_Anomaly	212	2.5

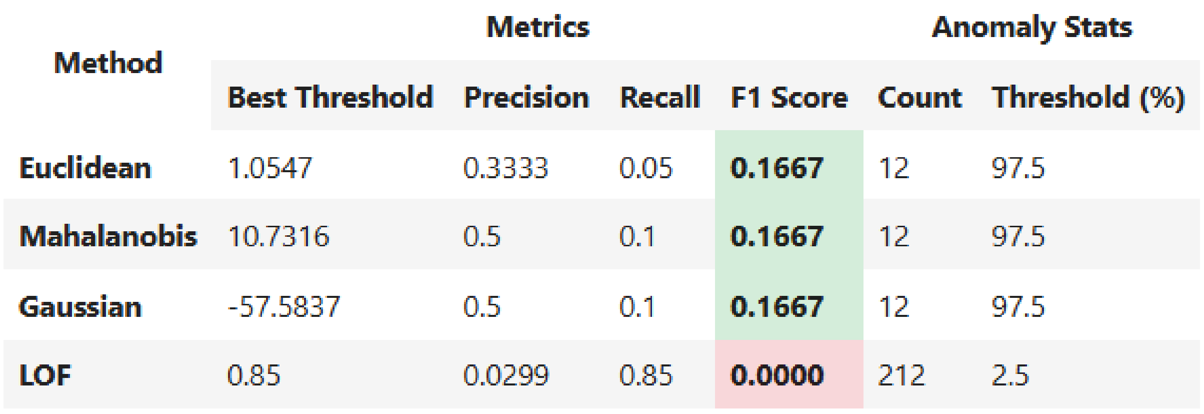


Unlabelled Data's Statistical Methods Visualization



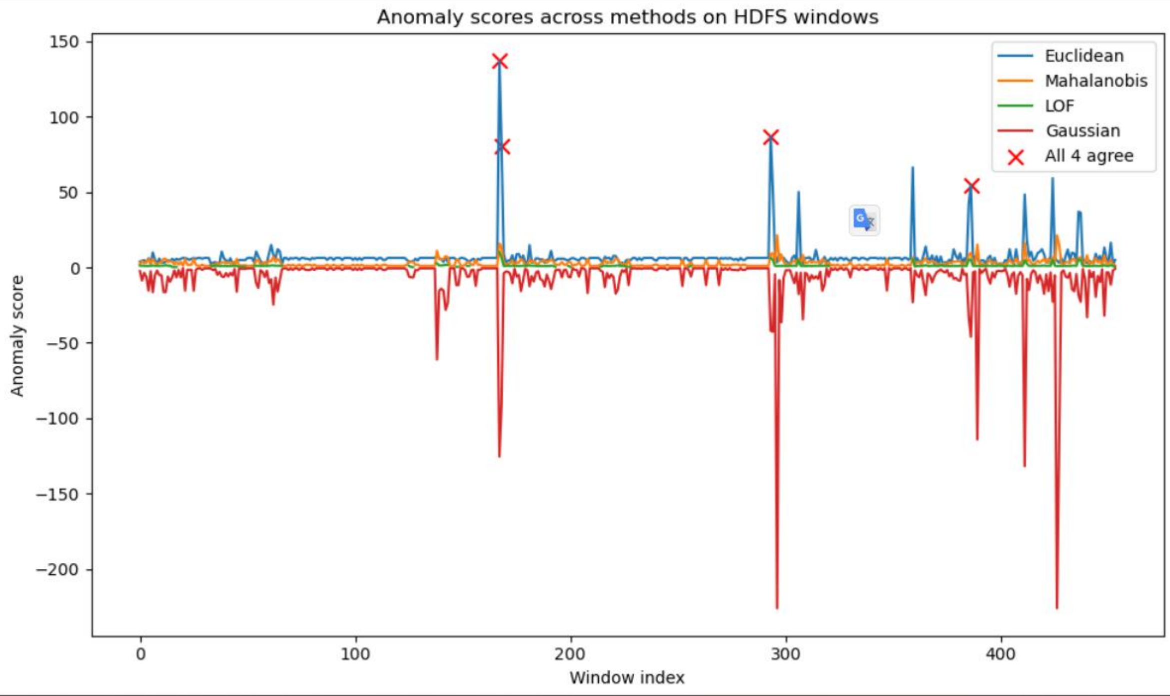
Method	Best Threshold	Precision	Recall	F1 Score
Euclidean	1.0546695930792247	0.3333333333333333	0.05	0.16666666666666669
Mahalanobis	10.73160675353624	0.5	0.1	0.16666666666666669
LOF	0.85	0.029876977152899824	0.85	0
Gaussian	-57.58369175627232	0.5	0.1	0.16666666666666669

Labelled Data's Statistical Methods Visualization



Method	Metrics				Anomaly Stats	
	Best Threshold	Precision	Recall	F1 Score	Count	Threshold (%)
Euclidean	1.0547	0.3333	0.05	**0.1667**	12	97.5
Mahalanobis	10.7316	0.5	0.1	**0.1667**	12	97.5
Gaussian	-57.5837	0.5	0.1	**0.1667**	12	97.5
LOF	0.85	0.0299	0.85	**0.0000**	212	2.5

Comparing both datasets (Metrics and Anomalies) Visualization



7.1 Code Snippets

It presents selected code snippets that illustrate the core data processing, feature engineering, and anomaly detection workflow described in *Chapter 3*.

Importing Libraries

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
```

```
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
```

Loading Datasets

```
# Load main structured log dataset
df_struct = pd.read_csv('D:\Research Project\HDFS_2k.log_structured.csv')
df_struct.head(5)

print(df_struct.columns)

# Display the first few rows of Date and Time columns
print(df_struct[['Date', 'Time']].head())

# Convert Date and Time columns to strings first
df_struct['Date'] = df_struct['Date'].astype(str).str.zfill(6)
df_struct['Time'] = df_struct['Time'].astype(str).str.zfill(6)

# Combine and convert to datetime; format: YYMMDDHHMMSS
df_struct['timestamp'] = pd.to_datetime(df_struct['Date'] + df_struct['Time'],
format='%y%m%d%H%M%S', errors='coerce')

# Show output
print(df_struct[['Date', 'Time', 'timestamp']].head())
```

Data Cleaning

```
# Drop rows where timestamp couldn't be parsed
df_struct = df_struct[df_struct['timestamp'].notnull()]
```

```
# Handle missing values in other columns
# Fill missing Pid, Level, Component with explicit 'Missing' or mode
for col in ['Pid', 'Level', 'Component', 'Content', 'EventId', 'EventTemplate']:
    if df_struct[col].isnull().any():
        if df_struct[col].dtype == 'object':
            df_struct[col].fillna('Missing', inplace=True)
        else:
            # For non-object columns, fill with mode or a constant
            df_struct[col].fillna(df[col].mode()[0], inplace=True)

# Normalize log level entries (e.g., upper case and standardize common levels)
df_struct['Level'] = df_struct['Level'].str.upper()
df_struct['Level'] = df_struct['Level'].replace({'WARN': 'WARNING', 'ERR': 'ERROR'})

# Remove duplicate rows for cleanliness
df_struct.drop_duplicates(inplace=True)
```

Saving the Cleaned dataset

```
# Sort by timestamp for windowing
df_struct.sort_values('timestamp', inplace=True)

# Save cleaned dataset for reproducibility
df_struct.to_csv('D:\Research Project\HDFS_cleaned.csv', index=False)

print("Preprocessing complete. Data saved to HDFS_cleaned.csv.")
print(df_struct.head())
print(df_struct.info())
```

Feature Engineer

```
# Load cleaned data
df = pd.read_csv('D:\Research Project\HDFS_cleaned.csv', parse_dates=['timestamp'])

# Set your time window (e.g., 1 hour)
window = '5min'  # or '10T' for 10 minutes, '1D' for 1 day, etc.

# Set timestamp as index for resampling
df.set_index('timestamp', inplace=True)

level_counts = df.groupby([pd.Grouper(freq=window), 'Level']).size().unstack(fill_value=0)
event_counts = df.groupby([pd.Grouper(freq=window), 'EventId']).size().unstack(fill_value=0)
total_counts = df.resample(window).size().rename('TotalCount')
feature_matrix = pd.concat([level_counts, event_counts, total_counts], axis=1).fillna(0)

print(feature_matrix.head())

print(feature_matrix.describe())

print(feature_matrix.columns)       # Lists all feature columns
print(feature_matrix.shape)         # (number of rows, number of columns)

 print(feature_matrix.dtypes)
```

```
feature_matrix = feature_matrix.reset_index()  # 'timestamp' becomes a column
feature_matrix.to_csv('D:/Research Project/HDFS_features_5min.csv', index=False)
```

Loading Feature Matrix data

```
# Load feature matrix
featurematrix = pd.read_csv('D:/Research Project/HDFS_features_5min.csv')

# Identify ONLY numeric columns for normalization
numeric_cols = featurematrix.select_dtypes(include=['int64', 'float64']).columns.tolist()
print("Numeric columns to be normalized:", numeric_cols)

# Initialize the scaler
scaler = StandardScaler()

# Transform only the numeric columns
featurematrix_scaled = featurematrix.copy()
featurematrix_scaled[numeric_cols] = scaler.fit_transform(featurematrix[numeric_cols])

# Save the normalized result (you can set your desired path)
featurematrix_scaled.to_csv('D:/Research Project/features_scaled.csv', index=False)

# Print first few rows for confirmation
print("Feature normalization complete. Sample normalized data:")
print(featurematrix_scaled.head())
```

Statistical Models from Scratch

Euclidean Distance

```
# X_orig: your original feature matrix on HDFS subset
X_orig = featurematrix[numeric_cols].values

mu_e = X_orig.mean(axis=0)

def euclidean_distance(x, mu):
    diff = x - mu
    return np.sqrt(np.sum(diff**2))

eucl_scores = np.array([euclidean_distance(x, mu_e) for x in X_orig])

# choose percentile guided by benchmark, e.g. 97.5th
thr_e = np.percentile(eucl_scores, 97.5)
featurematrix['Euclidean_score'] = eucl_scores
featurematrix['Euclidean_Anomaly'] = (eucl_scores > thr_e).astype(int)
```

Mahalanobis Distance

```
mu_m = X_orig.mean(axis=0)
cov = np.cov(X_orig, rowvar=False)
inv_cov = np.linalg.pinv(cov)

def mahalanobis_distance(x, mu, inv_cov):
    diff = x - mu
    return np.sqrt(diff.T @ inv_cov @ diff)

mah_scores = np.array([mahalanobis_distance(x, mu_m, inv_cov) for x in X_orig])
```

```
    thr_m = np.percentile(mah_scores, 97.5)
    featurematrix['Mahalanobis_score'] = mah_scores
    featurematrix['Mahalanobis_Anomaly'] = (mah_scores > thr_m).astype(int)
```

LOF

```
    n = X_orig.shape[0]
    k = 20

    dists = np.zeros((n, n))
    for i in range(n):
        diff = X_orig - X_orig[i]
        dists[i] = np.sqrt(np.sum(diff**2, axis=1))

    neighbors = np.argsort(dists, axis=1)[:, 1:k+1]
    k_dist = np.array([dists[i, neighbors[i, -1]] for i in range(n)])

    reach_dist = np.zeros_like(dists)
    for i in range(n):
        for j in neighbors[i]:
            reach_dist[i, j] = max(k_dist[j], dists[i, j])

    lrd = np.zeros(n)
    for i in range(n):
        rd_sum = np.sum(reach_dist[i, neighbors[i]])
        lrd[i] = k / rd_sum if rd_sum > 0 else 0.0

    lof_scores = np.zeros(n)
    for i in range(n):
        if lrd[i] > 0:
            lof_scores[i] = np.sum(lrd[neighbors[i]] / lrd[i]) / k
        else:
            lof_scores[i] = np.inf

    finite_lof = lof_scores[np.isfinite(lof_scores)]
    thr_l = np.percentile(finite_lof, 97.5)
    featurematrix['LOF_score'] = lof_scores
    featurematrix['LOF_Anomaly'] = (lof_scores > thr_l).astype(int)
```

Gaussian log-score

```
    def gaussian_log_score(x, mu, inv_cov):
        diff = x - mu
        return -0.5 * (diff.T @ inv_cov @ diff)

    gauss_scores = np.array([gaussian_log_score(x, mu_m, inv_cov) for x in X_orig])

    # low log-score = anomaly → e.g. 2.5th percentile
    thr_g = np.percentile(gauss_scores, 2.5)
    featurematrix['Gaussian_score'] = gauss_scores
    featurematrix['Gaussian_Anomaly'] = (gauss_scores < thr_g).astype(int)

    featurematrix[['Euclidean_Anomaly',
                   'Mahalanobis_Anomaly',
                   'Gaussian_Anomaly',
                   'LOF_Anomaly']].sum()
```

```
counts = featurematrix[['Euclidean_Anomaly',
                        'Mahalanobis_Anomaly',
                        'LOF_Anomaly',
                        'Gaussian_Anomaly']].sum()
plt.figure(figsize=(6,4))
plt.bar(counts.index, counts.values, color=['blue','orange','green','red'])
plt.ylabel('Number of anomalies')
plt.title('Anomaly counts per method on HDFS windows')
plt.tight_layout()
plt.show()



# how many windows are anomalous in at least one method
featurematrix['Any_Anomaly'] = (
    (featurematrix['Euclidean_Anomaly'] == 1) |
    (featurematrix['Mahalanobis_Anomaly'] == 1) |
    (featurematrix['Gaussian_Anomaly'] == 1) |
    (featurematrix['LOF_Anomaly'] == 1)
).astype(int)

print("Any_Anomaly:", featurematrix['Any_Anomaly'].sum())

# rows where all four agree
agree_all = featurematrix[
    (featurematrix['Euclidean_Anomaly'] == 1) &
    (featurematrix['Mahalanobis_Anomaly'] == 1) &
    (featurematrix['Gaussian_Anomaly'] == 1) &
    (featurematrix['LOF_Anomaly'] == 1)
]
```

```
print("Agree_all count:", len(agree_all))
print(agree_all[['timestamp','Euclidean_score',
                 'Mahalanobis_score','Gaussian_score',
                 'LOF_score']].head())
```

Loading labelled data

```
df_bench = pd.read_csv(r'D:\Research Project\anomaly_label.csv')
print(df_bench)
print(df_bench.columns)
print(df_bench.nunique())
```

HDFS data's content column vs Anomaly label's Block Id column

```
def extract_block_id(text):
    m = re.search(r'blk_[0-9\-]+', str(text))  # matches blk_ followed by digits and -
    return m.group(0) if m else None

df_struct['BlockId'] = df_struct['Content'].apply(extract_block_id)

print(df_struct[['Content', 'BlockId']].head(10))
print("Missing BlockId ratio:", df_struct['BlockId'].isna().mean())
```

Data Cleaning and Feature Engineering

```
print(df_bench.dtypes)
print(df_bench.isna().sum())



df_struct2 = df_struct.dropna(subset=['BlockId'])

# event-count features per BlockId
counts = df_struct2.groupby(['BlockId', 'EventId']).size().unstack(fill_value=0)

# align labels
labels = df_bench.set_index('BlockId')['Label']
X_bench = counts.values
y_bench = labels.loc[counts.index].values

print(X_bench.shape, y_bench.shape)
print(pd.Series(y_bench).value_counts())
```

Statistical Models from scratch and Model Evaluation

```
y = np.where(y_bench == 'Anomaly', 1, 0)

X_train, X_val, y_train, y_val = train_test_split(
    X_bench, y, test_size=0.3, random_state=0, stratify=y)
```

Euclidean Distance

```
mu = X_train.mean(axis=0)

def euclidean_distance(x, mu):
    diff = x - mu
    return np.sqrt(np.sum(diff**2))

scores_val = np.array([euclidean_distance(x, mu) for x in X_val])

# try thresholds over a grid of percentiles
candidates = np.percentile(scores_val, np.linspace(80, 99.5, 40))

f1_eu = 0
best_thr = None

for thr in candidates:
    y_pred = (scores_val > thr).astype(int)

    TP = np.sum((y_val == 1) & (y_pred == 1))
    FP = np.sum((y_val == 0) & (y_pred == 1))
    FN = np.sum((y_val == 1) & (y_pred == 0))

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
    recall    = TP / (TP + FN) if (TP + FN) > 0 else 0.0
    f1        = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0

    if f1 > f1_eu:
        f1_eu = f1
        best_thr = thr

print("Best Euclidean threshold:", best_thr)
print("Precision:", precision, "Recall:", recall, "F1:", f1_eu)
```

Mahalanobis Distance

```
# fit on training
mu_m = X_train.mean(axis=0)
cov = np.cov(X_train, rowvar=False)
inv_cov = np.linalg.pinv(cov)  # pseudo-inverse

def mahalanobis_distance(x, mu, inv_cov):
    diff = x - mu
    return np.sqrt(diff.T @ inv_cov @ diff)

scores_val_m = np.array([mahalanobis_distance(x, mu_m, inv_cov) for x in X_val])

candidates = np.percentile(scores_val_m, np.linspace(50, 99.5, 50))

f1_ma = 0
best_thr_m = None
best_prec = 0
best_rec = 0

for thr in candidates:
    y_pred = (scores_val_m > thr).astype(int)

    TP = np.sum((y_val == 1) & (y_pred == 1))
    FP = np.sum((y_val == 0) & (y_pred == 1))
    FN = np.sum((y_val == 1) & (y_pred == 0))

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
```

```
        recall    = TP / (TP + FN) if (TP + FN) > 0 else 0.0
        f1        = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0

        if f1 > f1_ma:
            f1_ma, best_thr_m, best_prec, best_rec = f1, thr, precision, recall

    print("Mahalanobis best thr:", best_thr_m)
    print("Precision:", best_prec, "Recall:", best_rec, "F1:", f1_ma)
```

LOF

```
    Xv = X_val
    n = Xv.shape[0]
    k = 20

    # distance matrix
    dists = np.zeros((n, n))
    for i in range(n):
        diff = Xv - Xv[i]
        dists[i] = np.sqrt(np.sum(diff**2, axis=1))

    neighbors = np.argsort(dists, axis=1)[:, 1:k+1]
    k_dist = np.array([dists[i, neighbors[i, -1]] for i in range(n)])

    reach_dist = np.zeros_like(dists)
    for i in range(n):
        for j in neighbors[i]:
            reach_dist[i, j] = max(k_dist[j], dists[i, j])
```

```
lrd = np.zeros(n)
for i in range(n):
    rd_sum = np.sum(reach_dist[i, neighbors[i]])
    lrd[i] = k / rd_sum if rd_sum > 0 else 0.0

lof_val = np.zeros(n)
for i in range(n):
    if lrd[i] > 0:
        lof_val[i] = np.sum(lrd[neighbors[i]] / lrd[i]) / k
    else:
        lof_val[i] = np.inf

finite_scores = lof_val[np.isfinite(lof_val)]
candidates = np.percentile(finite_scores, np.linspace(80, 99.5, 40))

f1_lof = 0
best_thr_l = None
best_prec = 0
best_rec = 0

for thr in candidates:
    y_pred = (lof_val > thr).astype(int)  # high LOF = anomaly

    TP = np.sum((y_val == 1) & (y_pred == 1))
    FP = np.sum((y_val == 0) & (y_pred == 1))
    FN = np.sum((y_val == 1) & (y_pred == 0))

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
```

```
        recall    = TP / (TP + FN) if (TP + FN) > 0 else 0.0
        f1        = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0

        if f1 > f1_lof:
            best_f1, best_thr_l, best_prec, best_rec = f1, thr, precision, recall

    print("LOF best thr:", best_thr_l)
    print("Precision:", best_prec, "Recall:", best_rec, "F1:", f1_lof)
```

Guassian

```
    # reuse mu_m and inv_cov from above

    def gaussian_log_score(x, mu, inv_cov):
        diff = x - mu
        return -0.5 * (diff.T @ inv_cov @ diff)  # higher = more normal, lower = more
anomalous

    scores_val_g = np.array([gaussian_log_score(x, mu_m, inv_cov) for x in X_val])

    # candidate thresholds on log-scores (low score = anomaly)
    candidates = np.percentile(scores_val_g, np.linspace(1, 50, 50))  # lower half

    f1_ga = 0
    best_thr_g = None
    best_prec = 0
    best_rec = 0
```

```
for thr in candidates:
    y_pred = (scores_val_g < thr).astype(int)  # low score → anomaly

    TP = np.sum((y_val == 1) & (y_pred == 1))
    FP = np.sum((y_val == 0) & (y_pred == 1))
    FN = np.sum((y_val == 1) & (y_pred == 0))

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
    recall    = TP / (TP + FN) if (TP + FN) > 0 else 0.0
    f1        = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0

    if f1 > f1_ga:
        f1_ga, best_thr_g, best_prec, best_rec = f1, thr, precision, recall

print("Gaussian best thr:", best_thr_g)
print("Precision:", best_prec, "Recall:", best_rec, "F1:", f1_ga)
```

Visualization

```
methods = ['Euclidean', 'Mahalanobis', 'LOF', 'Guassian']
f1_scores = [f1_eu, f1_ma, f1_lof, f1_ga]

plt.figure(figsize=(6,4))
plt.bar(methods, f1_scores, color=['blue','orange','green','red'])
plt.ylabel('F1 score')
plt.title('F1 comparison on labeled HDFS benchmark')
plt.ylim(0, max(f1_scores)*1.2)
plt.tight_layout()
plt.show()
```

Comparing both datasets (Metrics and Anomalies) Visualization

```
idx = featurematrix.index  # or a range(len(featurematrix))
eu  = featurematrix['Euclidean_score']
mah = featurematrix['Mahalanobis_score']
gau = featurematrix['LOF_score']
lof = featurematrix['Gaussian_score']

plt.figure(figsize=(10, 6))
plt.plot(idx, eu,  label='Euclidean')
plt.plot(idx, mah, label='Mahalanobis')
plt.plot(idx, gau, label='LOF')
plt.plot(idx, lof, label='Gaussian')

# mark windows where all four agree as anomalies
agree_all_mask = (
    (featurematrix['Euclidean_Anomaly']==1) &
    (featurematrix['Mahalanobis_Anomaly']==1) &
    (featurematrix['LOF_Anomaly']==1) &
    (featurematrix['Gaussian_Anomaly']==1)
)
plt.scatter(idx[agree_all_mask], eu[agree_all_mask],
            color='red', marker='x', s=80, label='All 4 agree')

plt.xlabel('Window index')
plt.ylabel('Anomaly score')
plt.title('Anomaly scores across methods on HDFS windows')
plt.legend()
plt.tight_layout()
plt.show()
```