ML-POWERED ANOMALY DETECTION

(Network Intrusion Detection System)

A MAJOR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

JAGRAN SCHOOL OF COMPUTER APPLICATION
BACHELOR OF COMPUTER APPLICATION

SUBMITTED BY

Sweksha Sharma (2022BCADS027)
Taniya Irfan Zindran (2022BCADS029)
Mahak Mishra (2022BCADS017)

UNDER SUPERVISION OF

Prof. Atul Kumar Gupta
Assistant Professor, FAST

DEC, 2024

igniting minds; changing lives

FACULTY OF SCIENCE AND TECHNOLOGY
JAGRAN LAKECITY UNIVERSITY, BHOPAL (M.P.)

ACKNOWLEDGEMENT

It is indeed a great pleasure to express my/our thanks and gratitude to all those who helped me/us during this period. This project would not have been materialized without the help from many quarters. We sincerely thank to all the persons who ever played a vital role in the successful completion of our project.

We sincerely thank all the people who co-operate and encourage me throughout the semester and make my project work successful.

We am are thankful to **Mr. Atul Kumar Gupta** (Project Supervisor, FAST) who has constantly remained helpful in suggesting directions and providing me guidance throughout the project.

We also thank **Mrs. Akrati Sharma** (Project Leader, FAST) for providing me/us the confidence and helping me in the project.

Thank you to **Dr. Dileep Kumar Singh** (Head, Faculty of Science and Technology, JLU Bhopal) for providing this platform which helped in the development of my/our technical skills and opportunity to study and work here which added a lot my/our knowledge, experience, confidence, skills and real field awareness.

It is good fortune that We had support and well wishes of many. We thank all those, whose means have not appeared here but the contributions have not gone unnoticed.

Signature of Students

Mahak Mishra (2022BCADS017)

Sweksha Sharma (2022BCADS027)

Taniya Irfan Zindran (2022BCADS029)

CERTIFICATE

I/We hereby certify that the work which is being presented in the BCA minor project report entitled **"ML-Powered Anomaly Detection",** in the partial fulfilment of the requirements for the award of the **Bachelor of Computer Application** is an authentic record of our own work carried out during session **Aug- Dec 2024** (**5th semester**) under the supervision of **Mr. Atul Kumar Gupta Assistant Professor, FAST, JLU, Bhopal.**

The matter presented in this Project Report has not been submitted by us for the award of any other degree elsewhere.

Signature of Students

Mahak Mishra (2022BCADS017) Sweksha Sharma (2022BCADS027)

Taniya Irfan Zindran (2022BCADS029)

This is to certify that the above statement made by the students is correct to the best of my knowledge.

Date:

Mr. Atul Kumar Gupta
Project Supervisor
FAST, JLU Bhopal

Ms. Akrati Sharma
Program Leader
FAST, JLU Bhopal

Dr. Dileep Kumar Singh
Head
FAST, JLU Bhopal

TABLE OF CONTENTS

ABSTRACT

Anomaly detection plays a pivotal role in identifying rare, unusual, or unexpected patterns in data that deviate from normal behavior, making it an essential tool across diverse domains such as cybersecurity, healthcare, finance, and manufacturing. This project explores the implementation of machine learning (ML)-powered anomaly detection using supervised learning algorithms. Supervised learning, which relies on labeled datasets, offers high accuracy and efficiency in detecting predefined anomalies.

The study delves into the design, development, and evaluation of supervised anomaly detection systems, highlighting their capability to handle complex and high-dimensional datasets. Techniques such as decision trees, support vector machines (SVMs), and deep learning models are employed to train models on historical data, enabling them to distinguish between normal and anomalous instances effectively. Additionally, the integration of feature engineering, data preprocessing, and hyperparameter optimization ensures robust model performance.

Key challenges, including data imbalance, labeling costs, generalization to unseen anomalies, and computational scalability, are critically analyzed. Furthermore, the limitations of supervised models are addressed by exploring hybrid approaches and emerging solutions like weakly supervised learning and synthetic data generation. The project also examines real-world applications, from fraud detection and network security to predictive maintenance and healthcare anomaly identification.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ML	Machine Learning
NIDS	Network Intrusion Detection System
BFSI	Banking, Financial Services, and Insurance
EHR	Electronic Health Records
DoS	Denial of Service
U2R	User to Root
ROC-AUC	Receiver Operating Characteristic - Area Under Curve
SMOTE	Synthetic Minority Oversampling Technique
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
SHAP	Shapley Additive Explanations
LIME	Local Interpretable Model-agnostic Explanations
GDPR	General Data Protection Regulation
CCPA	California Consumer Privacy Act
GAN	Generative Adversarial Network
VAE	Variational Autoencoder
XAI	Explainable Artificial Intelligence
IoT	Internet of Things

CHAPTER – 1
INTRODUCTION

1.1 Problem Definition

In today's increasingly digitalized world, the exponential growth of online activities across various platforms such as social media, e-commerce, and cloud services has resulted in the creation of vast amounts of data, referred to as digital footprints. These footprints include user interactions, behaviours, transactions, and communications, which provide valuable insights into user patterns and preferences. However, this growth also presents significant cybersecurity challenges.

Digital footprints have become lucrative targets for malicious actors who exploit them for activities such as identity theft, financial fraud, unauthorized access, phishing attacks, and large-scale data breaches. Traditional cybersecurity systems, which rely on predefined rules and manual updates, struggle to keep pace with the sophistication and frequency of emerging cyber threats, including zero-day attacks and unknown vulnerabilities.

The project addresses these limitations by leveraging machine learning-powered anomaly detection techniques. Anomalies, defined as deviations from expected or typical behaviour, often serve as early indicators of cybersecurity threats. The complexity of detecting these anomalies arises from the dynamic and diverse nature of digital activities, as well as the need to differentiate between legitimate atypical user behaviours and genuine threats.

This project focuses on building a robust anomaly detection system capable of analysing digital footprints in real-time. By employing advanced machine learning algorithms, it aims to proactively identify subtle and previously undetected anomalies, thereby enhancing cybersecurity defence . The system is dynamic and adaptive, automatically learning from new data, reducing false

positives, and offering a scalable solution that can mitigate risks effectively across industries such as BFSI, healthcare, and government defence .

The project aims to detect anomalies in digital footprints that may indicate potential cybersecurity threats. As digital footprints expand with the proliferation of online activities, they become a target for malicious actors, leading to risks such as identity theft, unauthorized access, and data breaches. Identifying these anomalies early can mitigate threats effectively.

Moreover, this system is designed with real-time detection capabilities, enabling organizations to identify and mitigate threats before they can cause significant harm. Real-time analysis reduces response times, enhancing an organization's ability to safeguard sensitive data and maintain operational continuity. By providing actionable insights through intuitive dashboards and alerts, the project simplifies the monitoring process, even for users with limited technical expertise. This democratization of cybersecurity tools ensures broader adoption and implementation, making it an invaluable asset for industries like BFSI, healthcare, and defence, where data integrity and security are paramount.

1.2 Project Overview

The project focuses on developing a machine learning-powered anomaly detection system designed to enhance cybersecurity by identifying potential threats hidden within digital footprints. By analysing vast datasets that capture user behaviours, interactions, and transactions, the system detects deviations or anomalies that could indicate cybersecurity risks such as fraud, unauthorized access, or data breaches.

Unlike traditional cybersecurity systems that rely on predefined rules and static models, this project introduces a dynamic and adaptive approach. The machine learning algorithms employed can learn from historical and new data, enabling the system to evolve with emerging threat patterns. This adaptability significantly improves detection accuracy and reduces the likelihood of false positives.

Key Features:

1. **Real-Time Anomaly Detection**: The system monitors digital activity continuously, identifying threats as they occur, allowing organizations to respond quickly and effectively.
2. **Visualization of Anomalies**: Intuitive dashboards and visual tools display detected anomalies, helping users understand patterns and trends, even with minimal technical expertise.
3. **Cross-Industry Application**: The solution is versatile and can be applied across sectors like Banking, Financial Services, and Insurance (BFSI) for fraud detection, defence for budgeting and spending anomalies, and healthcare for improving service quality and detecting financial irregularities.

This innovative and scalable system represents a significant step forward in proactive cybersecurity measures, addressing the limitations of static systems and meeting the evolving demands of digital security across multiple industries.

One of the standout features of this system is its ability to operate in real time, continuously analysing data to detect anomalies as they occur. By integrating real-time processing capabilities, the system minimizes delays in threat detection, allowing organizations to take immediate action. Whether it's identifying fraudulent transactions in financial systems, detecting unauthorized network access, or uncovering unusual patterns in healthcare operations, the system's speed and efficiency make it a valuable asset in minimizing potential damage and ensuring operational continuity.

The anomaly detection system leverages advanced machine learning algorithms to provide a proactive solution for identifying cybersecurity threats within digital footprints. Traditional systems often fail to keep up with the rapid evolution of cyber threats, relying heavily on manually updated rules that cannot account for new or unknown attack patterns. This project aims to overcome these challenges by implementing a system that dynamically learns from data, ensuring that it remains effective against both current and emerging threats. The project emphasizes automation, scalability, and adaptability to handle the growing complexity and volume of digital data.

One of the standout features of this system is its ability to operate in real time, continuously analysing data to detect anomalies as they occur. By integrating real-time processing capabilities, the system minimizes delays in threat detection, allowing organizations to take immediate action. Whether it's identifying fraudulent transactions in financial systems, detecting unauthorized network access, or uncovering unusual patterns in healthcare operations, the system's speed and efficiency make it a valuable asset in minimizing potential damage and ensuring operational continuity.

Visualization is another crucial component of this project. The system includes dashboards that provide clear and actionable insights into detected anomalies. These visualizations simplify complex data, enabling users to understand and interpret patterns without requiring in-depth technical knowledge. This feature is particularly beneficial for organizations with limited cybersecurity expertise, as it lowers the barrier to entry and promotes broader adoption of advanced threat detection technologies. The system's cross-industry applicability further enhances its appeal, as it can be tailored to address unique challenges in diverse fields such as BFSI, healthcare, and defence, thereby extending its impact and value.

Furthermore, the project underscores the importance of integrating seamlessly with existing cybersecurity frameworks. By designing the system to be compatible with industry-standard tools and platforms, it ensures ease of deployment and scalability. Organizations can implement this system as a modular addition to their existing infrastructure, reducing overhead and streamlining operations.

1.3 Hardware Specification

The hardware requirements for this anomaly detection system have been carefully chosen to ensure smooth operation, optimal performance, and scalability to handle the large volumes of data typically involved in cybersecurity applications. Below is a detailed explanation of the hardware components:

1. **Processor-(CPU)**:

   The system requires a multi-core processor, with **Intel i5** or an equivalent and above being the minimum recommended specification. Multi-core CPUs provide the parallel processing capabilities necessary for efficient data handling, feature extraction, and executing machine learning models.

2. **RAM**:

   A minimum of **8GB of RAM** is essential to ensure the system can load and process datasets effectively without significant delays. However, for larger datasets or when employing more complex models, such as those used in deep learning, 16GB of RAM or more is highly recommended. Sufficient memory allocation is critical for operations such as data preprocessing, model training, and real-time anomaly detection, which can be memory-intensive.

3. **Storage**:

   The system requires at least **20GB** of free disk space for software installation, data storage, and temporary files generated during analysis. Organizations dealing with extensive historical data or maintaining large logs for training purposes should consider using higher-capacity storage solutions, such as SSDs, for faster read/write speeds. Additional storage options like external drives or cloud-based storage solutions can also be leveraged for archiving datasets or managing backups.

4. **GPU**:

   While not mandatory, a dedicated Graphics Processing Unit (GPU) is highly beneficial for users working with deep learning models, such as autoencoders or convolutional neural networks (CNNs). GPUs significantly accelerate matrix computations and other parallel processing tasks, making them indispensable for training large models. An **NVIDIA GPU** with CUDA support is recommended, as it is compatible with popular deep learning frameworks like TensorFlow and PyTorch. Models such as NVIDIA GTX 1660 or RTX 30-series provide excellent performance for most machine learning workloads.

These hardware specifications ensure that the system can efficiently perform tasks ranging from data preprocessing and exploratory analysis to model training

and real-time threat detection. Organizations or users planning for future scalability should consider investing in more robust hardware configurations to accommodate growing datasets and increasingly complex cybersecurity demands.

1.4 Software Specification

The software requirements for the anomaly detection system are designed to ensure a flexible, efficient, and scalable development and deployment process. These specifications are well-suited for handling complex datasets, developing machine learning models, and visualizing insights effectively. Below is a detailed overview:

1. **Operating_System**:
   The system is compatible with multiple operating systems, including Windows 10/11, macOS, and Linux. This cross-platform support allows developers to use their preferred environments, ensuring flexibility and ease of adoption.

2. **Programming_Languages**:
   Python 3.7 or higher serves as the primary programming language due to its extensive ecosystem of libraries, simplicity, and versatility in handling data analysis and machine learning tasks. Its user-friendly syntax and strong community support make it an ideal choice for both beginners and advanced users.

3. **Libraries**:
   - **NumPy** and **Pandas**: Essential for data manipulation and analysis, providing robust tools for handling large datasets efficiently.

- **Scikit-learn**: A popular library for implementing a variety of machine learning algorithms, including anomaly detection models like Isolation Forest and One-Class SVM.
- **TensorFlow**: Used for building and training deep learning models, especially when more advanced techniques like autoencoders are required.
- **PyOD (Python Outlier Detection)**: A specialized library for detecting anomalies in multivariate data, providing access to numerous state-of-the-art algorithms.

4. **Development Tools**:
   - **Google Collab**: A cloud-based environment offering free access to GPUs for training machine learning models. Its collaborative features allow multiple users to work simultaneously on the same project.
   - **Jupyter Notebook**: Ideal for iterative development, it enables users to write, test, and document code seamlessly in a single environment.
5. **Visualization Tools**:
   - **Tableau** and **Power BI**: Both are industry-standard tools for creating interactive and visually appealing dashboards. These tools simplify the presentation of anomalies and insights, making them accessible to users with minimal technical expertise.

These software specifications provide a comprehensive suite of tools for developing, testing, and deploying the anomaly detection system, as well as for analysing and presenting results effectively across various industries.

CHAPTER – 2

LITERATURE SURVEY

2.1 Existing System

Traditional anomaly detection systems are typically rule-based or signature-based, relying on predefined patterns or signatures to detect anomalies. These systems operate by comparing incoming data to a set of static rules or known attack signatures. While they can effectively detect known threats, they struggle against novel or previously unseen anomalies, such as zero-day attacks.

One of the main limitations of these systems is their reliance on manual updates. Security teams must continuously update the rules and signature databases to stay ahead of evolving threats. This process is not only time-consuming but also prone to human error, potentially leaving critical gaps in protection. Furthermore, manual intervention slows down the system's ability to adapt to new and sophisticated attack vectors.[3]

Another major drawback of traditional systems is their high false-positive rates. Since they rely on rigid rules, these systems often misclassify benign activities as anomalies, leading to unnecessary alerts and operational inefficiencies. This can result in alert fatigue among security personnel, who may overlook genuine threats amidst a flood of false alarms.

Additionally, traditional systems have limited scalability and adaptability. As the volume and complexity of data grow, these systems often fail to keep up, either due to computational limitations or the inability to dynamically adjust to evolving data patterns. This makes them unsuitable for modern, large-scale environments like cloud infrastructures or IoT ecosystems, where data is highly dynamic and diverse.[4]

Finally, these systems are ill-equipped to handle advanced, stealthy threats like zero-day attacks. Zero-day attacks exploit previously unknown vulnerabilities, for which no signatures exist. As a result, traditional systems cannot detect or mitigate these threats, leaving organizations vulnerable to significant damage.

In summary, while traditional anomaly detection systems have served as the foundation for cybersecurity, their inherent limitations in terms of adaptability, scalability, and precision make them inadequate for addressing the challenges posed by today's complex and evolving threat landscape. This necessitates the development and adoption of more advanced systems, such as machine learning-based approaches, which can learn from data patterns, adapt dynamically, and detect anomalies with greater accuracy.

2.2 Proposed System

The proposed system leverages the power of machine learning (ML) to overcome the limitations of traditional anomaly detection systems. By incorporating advanced techniques such as supervised, unsupervised, and deep learning algorithms, this system offers a dynamic and adaptive approach to identifying anomalies, ensuring enhanced detection capabilities for both known and unknown threats.

1. Adaptive Learning and Continuous Improvement

Unlike traditional rule-based systems, the proposed system is capable of learning from historical and real-time data. Through supervised learning techniques, it can be trained on labelled datasets to recognize specific patterns associated with anomalies or normal behaviour. This enables the system to classify future data points with high accuracy. Furthermore, unsupervised learning algorithms, such as clustering or autoencoders, allow the system to detect deviations from expected

patterns without requiring labelled data. This is particularly useful for identifying novel or unknown threats like zero-day attacks.

2. Scalability and Real-Time Detection

The proposed system is designed to handle large-scale data in real time, making it suitable for modern environments like cloud infrastructures, IoT ecosystems, and distributed networks. By employing efficient algorithms and optimized computational pipelines, the system can process high volumes of data without compromising detection speed or accuracy. This ensures timely identification of threats, enabling proactive mitigation.

3. Improved Accuracy and Reduced False Positives

The machine learning approach significantly reduces the high false-positive rates commonly seen in traditional systems. By learning nuanced distinctions between normal and anomalous behaviour, the proposed system minimizes misclassifications. Advanced validation techniques, such as cross-validation and performance metrics like precision, recall, and F1-score, ensure that the system is fine-tuned for accuracy. Moreover, adaptive algorithms can adjust to changes in data distribution over time, maintaining reliability even in dynamic environments.

4. Detection of Known and Unknown Threats

One of the standout features of the proposed system is its ability to detect both known and unknown threats. Supervised algorithms can effectively identify anomalies that align with previously observed patterns, while unsupervised and deep learning methods excel in uncovering novel or unexpected deviations. This dual capability ensures comprehensive protection against a wide range of threats, from traditional attacks to emerging vulnerabilities.

5. Integration and Automation

The system can be seamlessly integrated with existing infrastructure, such as security information and event management (SIEM) tools, for enhanced monitoring and automated response. By automating the anomaly detection process, the system reduces the burden on human operators and allows security teams to focus on strategic tasks.

The feasibility of the proposed system is evaluated across three critical dimensions: technical, operational, and economic. These factors collectively ensure that the system is practical, implementable, and beneficial for stakeholders.

Technical Feasibility

The proposed system is technically feasible as it utilizes well-established machine learning libraries and frameworks, such as Scikit-learn, TensorFlow, and PyTorch. These tools offer robust support for implementing supervised, unsupervised, and deep learning algorithms, making the development process efficient and reliable. Additionally, the system's architecture is designed to operate on commonly available hardware, such as systems with modern CPUs, moderate amounts of RAM (16GB or higher), and optional GPU support for deep learning tasks. This ensures that the hardware requirements do not pose a significant barrier to implementation, even for small to medium-sized organizations.

Operational Feasibility

The system's applicability across diverse sectors demonstrates its operational feasibility. Industries such as banking, financial services, and insurance (BFSI) can leverage the system to detect fraudulent transactions and safeguard sensitive data. In the healthcare sector, it can monitor electronic health records (EHR) and connected medical devices to detect unauthorized access or anomalies in patient

data. Defence and government agencies can employ the system to secure critical infrastructure and monitor for potential cyberattacks targeting sensitive operations.

Its flexibility and adaptability make it suitable for organizations of varying sizes and operational complexities. By automating anomaly detection, the system reduces the workload of cybersecurity teams, enabling them to focus on strategic decision-making. Additionally, the system's ability to operate in real-time ensures that it aligns with the fast-paced demands of modern industries, where swift threat detection and response are critical to operational continuity.

Economic Feasibility

From an economic perspective, the system is cost-effective, offering a high return on investment. By reducing reliance on manual monitoring, it significantly cuts down on labor costs associated with traditional anomaly detection processes. The automation and precision of the system also minimize the financial impact of false positives, which can lead to unnecessary investigations and operational disruptions.

Moreover, the system's ability to detect and mitigate cyber threats proactively helps organizations avoid substantial losses stemming from data breaches, downtime, and reputational damage. For example, early detection of zero-day attacks can prevent catastrophic failures that could result in multimillion-dollar losses. The system’s scalability ensures that organizations can start small and expand its capabilities as their needs and budgets grow, further enhancing its economic viability.

1. Financial Fraud Detection

Problem-Statement

Fraudulent transactions in financial systems result in significant monetary losses and damage to reputation. Anomaly detection can identify suspicious activities early to prevent fraud.

Approach

- **Dataset:** A publicly available dataset containing transactional data, such as the European Credit Card Fraud dataset. It contains features such as transaction amount, timestamp, and customer profile. The dataset is highly imbalanced, with only ~0.17% fraudulent transactions.[1]
- **Supervised Algorithms Used:**
    - Logistic Regression
    - Random Forest
    - Gradient Boosting (e.g., XGBoost)
- **Preprocessing:** Addressed imbalance using SMOTE (Synthetic Minority Oversampling Technique).
- **Evaluation Metrics:** Precision, Recall, F1-score to minimize false positives and negatives.

Results

- Gradient Boosting achieved an F1-score of 0.92, significantly outperforming Logistic Regression and Random Forest.
- Feature importance analysis showed that "Transaction Amount" and engineered time-related features were critical in identifying anomalies.

Applications

- Real-time fraud detection systems in credit card processing.
- Risk management in banking systems.[2]

2. Network Intrusion Detection

Problem-Statement

With the increasing sophistication of cyberattacks, organizations require automated systems to detect unauthorized access or abnormal activity in their networks.

Approach

- **Dataset:** The KDD Cup 1999 dataset, containing labeled network traffic data with normal and various attack categories (e.g., DoS, R2L, U2R, Probe).
- **Supervised Algorithms Used:**
    - Support Vector Machines (SVM)
    - Multi-Layer Perceptrons (MLPs)
    - Random Forest
- **Preprocessing:** Standardization of numerical features and one-hot encoding of categorical data.
- **Evaluation Metrics:** ROC-AUC and confusion matrix analysis.

Results

- Random Forest achieved an accuracy of 97.5% and a high recall for detecting rare attack types like U2R (User to Root attacks).
- SVM performed well in binary classification tasks (normal vs. abnormal traffic).

Applications

- Intrusion detection systems (IDS) in enterprise networks.
- Cybersecurity monitoring for critical infrastructure.

3. Healthcare: Disease Outbreak Detection

Problem Statement

Timely detection of disease outbreaks can save lives and reduce healthcare costs. Anomaly detection models can identify unusual patterns in patient data that may indicate emerging epidemics.

Approach

- **Dataset:** Synthetic healthcare data with patient visits, symptoms, diagnostic tests, and disease labels.
- **Supervised Algorithms Used:**
    - Decision Trees
    - Naïve Bayes
    - Neural Networks
- **Preprocessing:**
    - Handled missing data using imputation techniques.

    - Feature engineering to derive seasonal patterns and demographic-based risk factors.

- **Evaluation Metrics:** Sensitivity, Specificity, and Area Under the Curve (AUC).

Results

- Neural Networks achieved the best sensitivity (95%), ensuring most outbreaks were detected.
- Naïve Bayes provided interpretable results, useful for initial diagnosis support.

Applications

- Early warning systems for epidemics in public health.
- Predictive analytics in hospitals to manage resource allocation.

4. Manufacturing: Predictive Maintenance

Problem Statement

Unplanned downtime in manufacturing processes due to equipment failure leads to productivity loss and increased costs. Anomaly detection helps predict failures before they occur.

Approach

- **Dataset:** Sensor data from industrial equipment (e.g., temperature, vibration, and pressure readings). Labeled data include normal and faulty equipment states.
- **Supervised Algorithms Used:**
    - Random Forest

    - Support Vector Machines
    - Time-Series Adapted Neural Networks
- **Preprocessing:**
    - Feature engineering for temporal trends and rolling averages.
    - Resampling to balance normal vs. failure cases.
- **Evaluation Metrics:** Mean Time Between Failure (MTBF) improvements and accuracy in fault prediction.

Results

- Random Forest provided actionable insights with a classification accuracy of 94%.
- Time-Series Neural Networks captured trends and predicted failures 48 hours in advance with 85% reliability.

Applications

- Real-time monitoring of machinery in industries like automotive, energy, and aerospace.
- Integration into industrial IoT platforms for automated maintenance.

5. Retail: Customer Behavior Anomaly Detection

Problem Statement

Retailers face challenges in understanding customer behavior anomalies, such as sudden churn or uncharacteristic purchasing patterns, which can indicate dissatisfaction or fraud.

Approach

- **Dataset:** Customer transaction histories, loyalty program interactions, and demographic data.

- **Supervised Algorithms Used:**
    - Gradient Boosting Machines (GBMs)
    - Logistic Regression
- **Preprocessing:**
    - Feature engineering to calculate customer lifetime value (CLV), frequency of visits, and average spend.
    - Categorical encoding for non-numerical data.
- **Evaluation Metrics:** Customer retention rate and anomaly detection accuracy.

Results

- Gradient Boosting Machines detected anomalies with a precision of 90%, enabling targeted retention strategies.
- Logistic Regression was deployed for real-time analysis due to its simplicity and interpretability.

Applications

- Personalized marketing campaigns.
- Fraud detection in loyalty program redemptions.[3]

CHAPTER – 3

SYSTEM ANALYSIS AND DESIGN

Data Flow Diagram

The flow of data in a system or process is represented by a data flow diagram (DFD). It also gives insight into the inputs and outputs of each entity and the process itself. Data flow diagram (DFD) does not have a control flow and no loops decision rules are present. Specific operations, depending on the type of data, can be explained by a flowchart. It is a graphical tool, useful for communicating with users, managers and other personnel. It is useful for analyzing existing as well as proposed systems. []

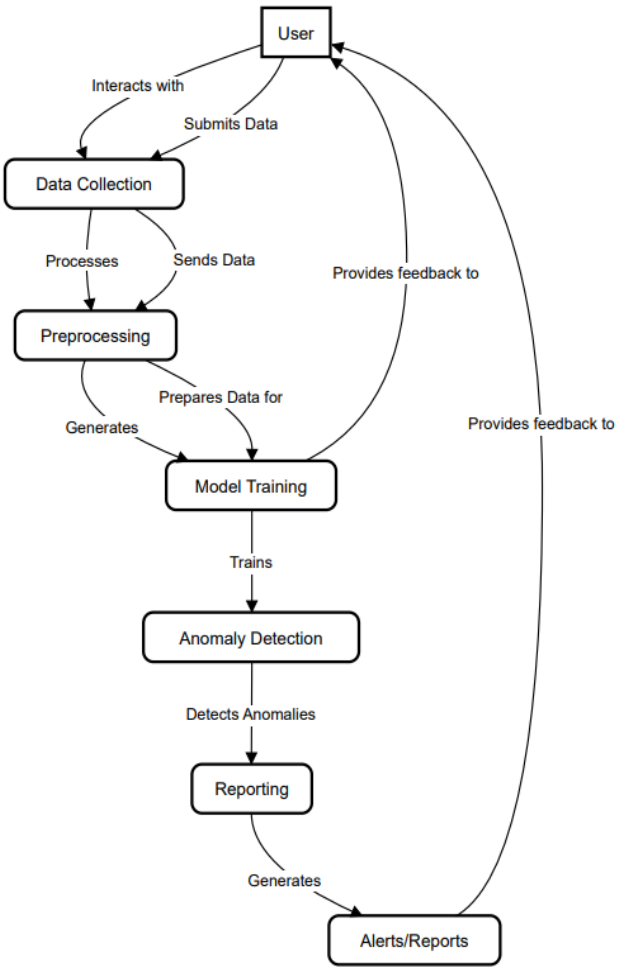


fig – 3.1 – Data flow diagram

CHAPTER – 4

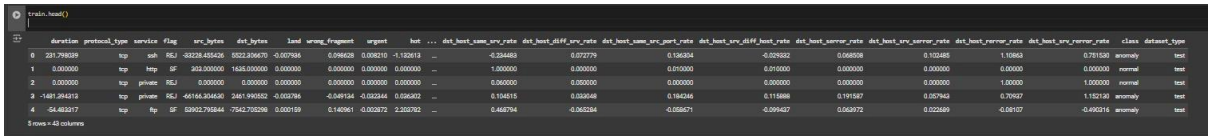
OUTPUT

4.1 Data Preprocessing

```
[ ] import numpy as np
    import pandas as pd
```

```
[ ] train = pd.read_csv('/content/Labeled_Dummy_Test_data.csv')
    test = pd.read_csv('/content/Labeled_Dummy_Train_data.csv')
    train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22544 entries, 0 to 22543
Data columns (total 43 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   duration                     22544 non-null  float64
 1   protocol_type                22544 non-null  object
 2   service                      22544 non-null  object
 3   flag                         22544 non-null  object
 4   src_bytes                    22544 non-null  float64
 5   dst_bytes                    22544 non-null  float64
 6   land                         22544 non-null  float64
 7   wrong_fragment               22544 non-null  float64
 8   urgent                       22544 non-null  float64
 9   hot                          22544 non-null  float64
 10  num_failed_logins            22544 non-null  float64
 11  logged_in                    22544 non-null  float64
 12  num_compromised              22544 non-null  float64
 13  root_shell                   22544 non-null  float64
 14  su_attempted                 22544 non-null  float64
 15  num_root                     22544 non-null  float64
 16  num_file_creations           22544 non-null  float64
 17  num_shells                   22544 non-null  float64
 18  num_access_files             22544 non-null  float64
 19  num_outbound_cmds            22544 non-null  float64
 20  is_host_login                22544 non-null  float64
 21  is_guest_login               22544 non-null  float64
 22  count                        22544 non-null  float64
 23  srv_count                    22544 non-null  float64
 24  serror_rate                  22544 non-null  float64
 25  srv_serror_rate              22544 non-null  float64
 26  rerror_rate                  22544 non-null  float64
 27  srv_rerror_rate              22544 non-null  float64
 28  same_srv_rate                22544 non-null  float64
 29  diff_srv_rate                22544 non-null  float64
 30  srv_diff_host_rate           22544 non-null  float64
 31  dst_host_count               22544 non-null  float64
 32  dst_host_srv_count           22544 non-null  float64
 33  dst_host_same_srv_rate       22544 non-null  float64
 34  dst_host_diff_srv_rate       22544 non-null  float64
 35  dst_host_same_src_port_rate  22544 non-null  float64
 36  dst_host_srv_diff_host_rate  22544 non-null  float64
 37  dst_host_serror_rate         22544 non-null  float64
 38  dst_host_srv_serror_rate     22544 non-null  float64
 39  dst_host_rerror_rate         22544 non-null  float64
 40  dst_host_srv_rerror_rate     22544 non-null  float64
 41  class                        22544 non-null  object
 42  dataset_type                 22544 non-null  object
dtypes: float64(38), object(5)
memory usage: 7.4+ MB
```



```
train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	class	dataset_type
0	231.798099	tcp	ssh	REJ	-33228.455426	5522.306670	-0.007936	0.098628	0.008210	-1.132613	...	-0.234483	0.072779	0.136354	-0.029332	0.068508	0.102485	1.10863	0.751530	anomaly	test
1	0.000000	tcp	http	SF	303.000000	1635.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.000000	0.010000	0.010000	0.000000	0.000000	0.00000	0.000000	normal	test
2	0.000000	tcp	private	REJ	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.060000	0.050000	0.000000	0.000000	0.000000	0.000000	1.00000	1.000000	normal	test
3	-1481.394313	tcp	private	REJ	-66166.304630	2461.990552	-0.003786	-0.049134	-0.032344	0.036302	...	0.104515	0.030648	0.194246	0.115888	0.191587	0.057943	0.70927	1.152130	anomaly	test
4	-54.483317	tcp	ftp	SF	53902.795844	-7542.705298	0.000159	0.140961	-0.002872	2.203782	...	0.466794	-0.065284	-0.058671	-0.094427	0.043972	0.022689	-0.08107	-0.490316	anomaly	test

5 rows × 43 columns

```
[ ] train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...
0	231.798099	tcp	ssh	REJ	-33228.455426	5522.306670	-0.007936	0.098628	0.008210	-1.132613	...
1	0.000000	tcp	http	SF	303.000000	1635.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.000000	tcp	private	REJ	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
3	-1481.394313	tcp	private	REJ	-66166.304630	2461.990552	-0.003786	-0.049134	-0.032344	0.036302	...
4	-54.483317	tcp	ftp	SF	53902.795844	-7542.705298	0.000159	0.140961	-0.002872	2.203782	...

5 rows × 43 columns

```
[ ] train.describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins
count	22544.000000	2.254400e+04	2.254400e+04	22544.000000	22544.000000	22544.000000	22544.000000	22544.000000
mean	209.500515	6.714832e+03	2.075142e+03	0.000191	0.008605	0.001145	0.109070	0.022483
std	1381.335066	8.648764e+04	2.251547e+04	0.015274	0.145190	0.041315	1.411071	0.165485
min	-2680.054230	-1.745497e+05	-4.344843e+04	-0.020930	-0.208207	-0.075737	-2.799493	-0.298948
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	1.050000e+02	6.850000e+01	0.000000	0.000000	0.000000	0.000000	0.000000
75%	35.000000	1.830903e+03	2.752843e+03	0.000000	0.000000	0.000000	0.000000	0.000000
max	58669.516437	6.291668e+06	1.370732e+06	1.005654	3.107103	2.000000	101.456261	4.027126

8 rows × 38 columns

```
[ ] train.describe(include='object')
```

	protocol_type	service	flag	class	dataset_type
count	22544	22544	22544	22544	22544
unique	3	64	11	2	1
top	tcp	http	SF	normal	test
freq	18814	7670	14795	13527	22544

4.2 Data Analysis

```
[ ] import seaborn as sns
    import matplotlib.pyplot as plt

    # Create the count plot
    plt.figure(figsize=(8, 6))
    ax = sns.countplot(
        x=train["class"],
        order=["normal", "anomaly"],  # Ensure proper order
        palette="coolwarm"
    )

    # Add absolute values as labels
    abs_values = train["class"].value_counts().reindex(["normal", "anomaly"]).values
    for i, bar in enumerate(ax.patches):
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            bar.get_height(),
            f"{abs_values[i]}",  # Format as integer
            ha="center",
            va="bottom",
            fontsize=12
        )

    # Set x-tick labels and add title
    ax.set_title("Class Distribution in Training Data", fontsize=16)
    ax.set_ylabel("Count", fontsize=14)
    ax.set_xlabel("Class", fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.show()
```

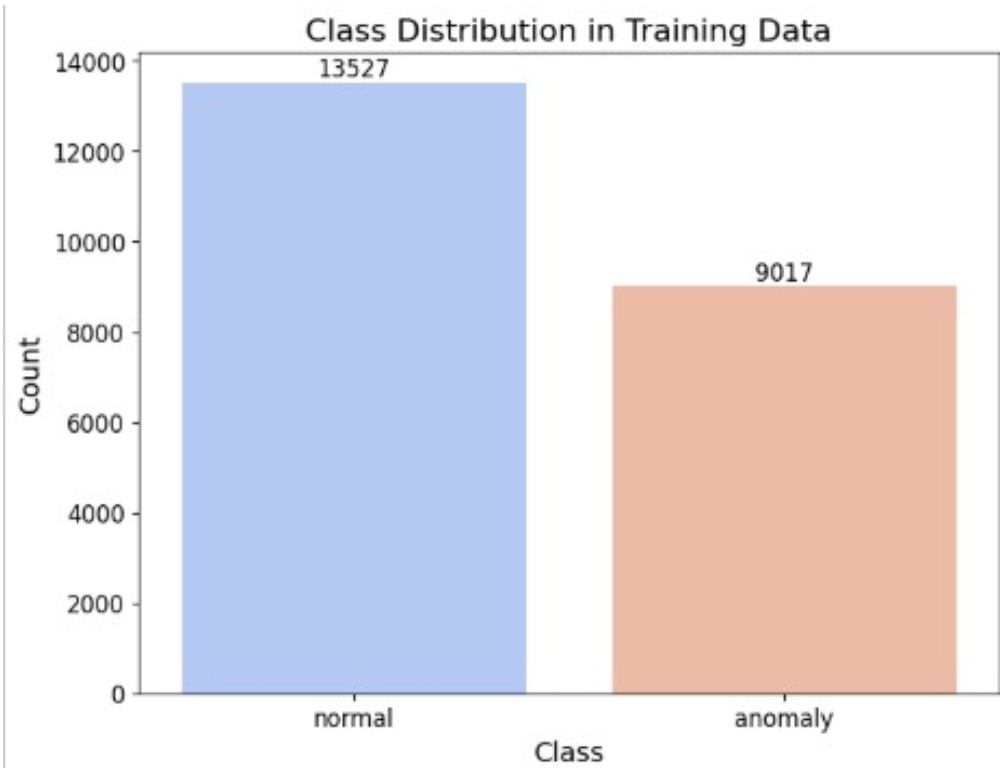


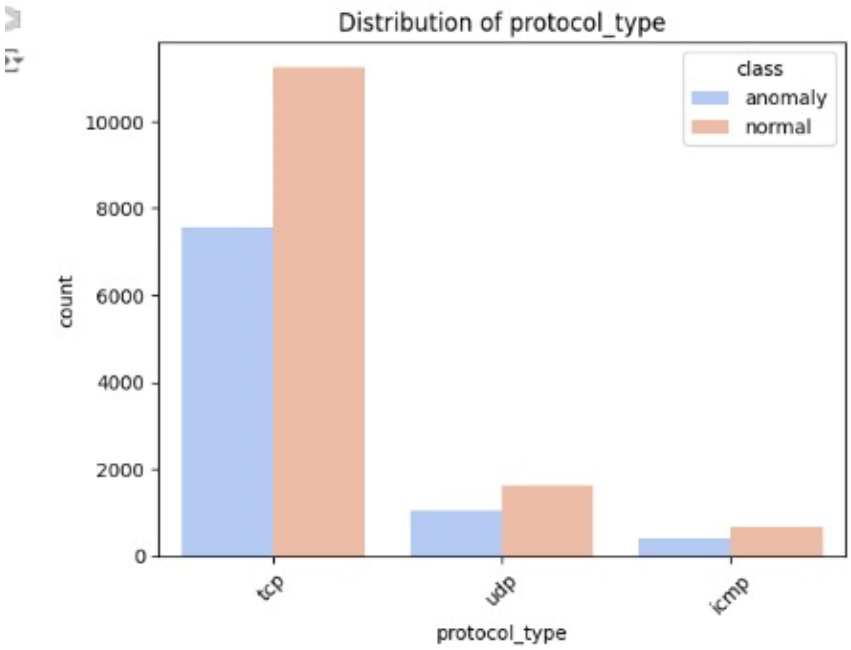
Fig-4.1 Class distribution in training data

The graph labeled "Class Distribution in Training Data" is a bar chart created using Python libraries Seaborn and Matplotlib. It shows the distribution of two classes, "normal" and "anomaly," in the training dataset.

- The x-axis represents the class labels: "normal" and "anomaly."
- The y-axis represents the count of instances for each class.

The chart indicates that there are 13,527 instances of the "normal" class and 9,017 instances of the "anomaly" class. Absolute values are displayed above the bars for clarity.The bars are color-coded using the "coolwarm" palette.

- This visualization helps in understanding the class imbalance in the dataset.



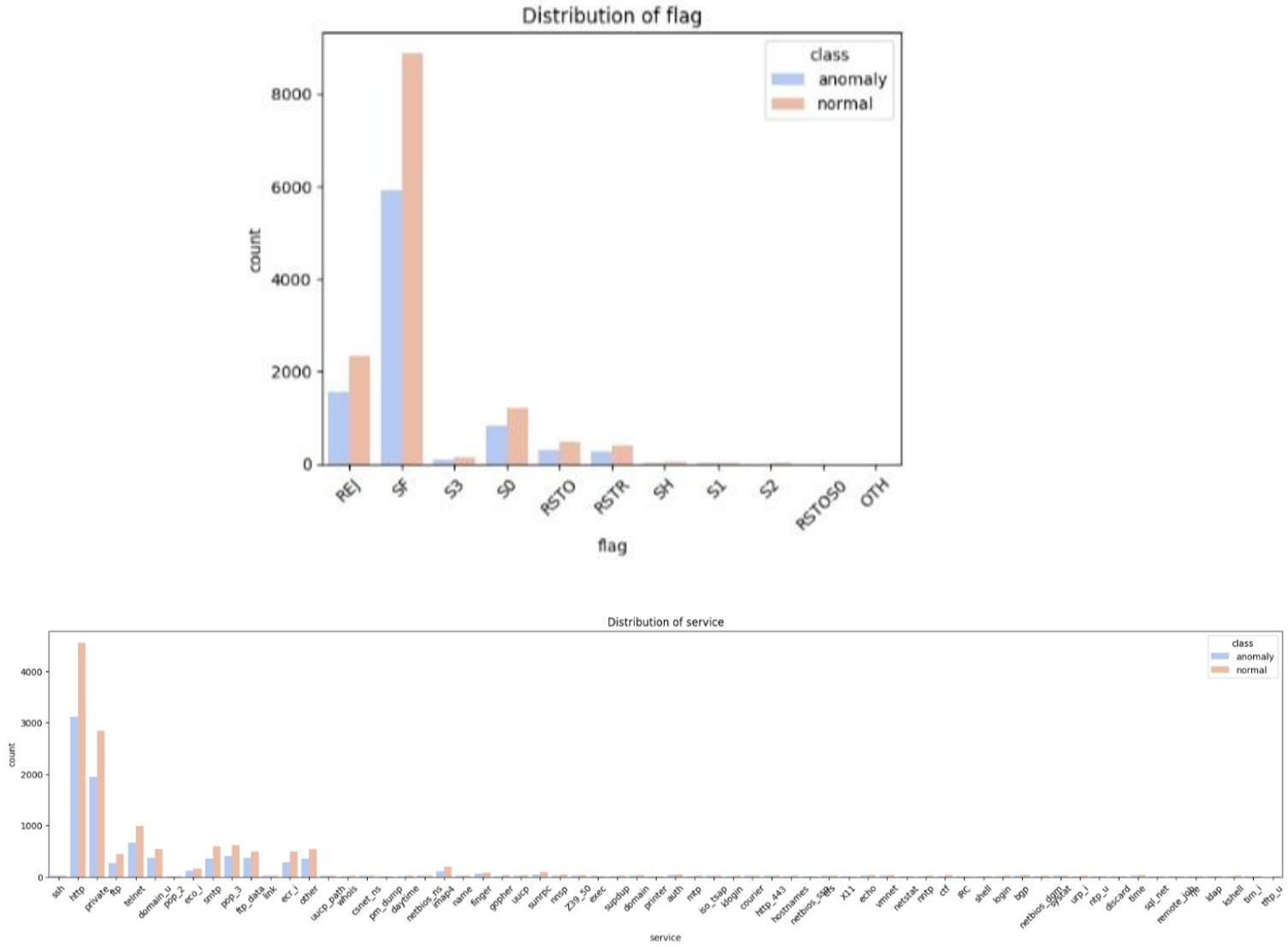


Fig -4.2 (a) Graphs of distribution of protocol type**, (b)** distribution of flag, **(c)** distribution of services.

The three graphs collectively provide insights into the distribution of protocol types, flag values, and service types within a dataset categorized by anomaly and normal classes. The protocol graph reveals that TCP is the most frequently used protocol for both normal and anomalous traffic, highlighting its prevalence in the dataset and susceptibility to anomalies. UDP shows balanced usage between

normal and anomalies, albeit at lower counts compared to TCP. ICMP is infrequent overall, with slightly higher normal occurrences.

The flag distribution graph highlights that the SF flag dominates both normal and anomalous activities, with normal instances significantly outnumbering anomalies. Flags like S3 and REJ are more commonly associated with anomalies, while others such as S0, RSTO, and OTH occur less frequently in both categories.

The service distribution graph shows that HTTP and private services are the most widely used, with substantial counts in both normal and anomalous activities, indicating their prominence and vulnerability. FTP services exhibit moderate usage with fewer anomalies compared to HTTP and private services. Other services, including SMTP and DNS, have low occurrences, making them less relevant for anomaly detection. Collectively, these insights are crucial for understanding traffic patterns and identifying potential risks in network behaviour, aiding in anomaly detection and cybersecurity strategies.

```
[ ] # duplicates
    print(f"Number of duplicate rows in train set: {train.duplicated().sum()}")
```

```
Number of duplicate rows in train set: 3361
```

```
[ ] # Remove duplicate rows
    train.drop_duplicates(inplace=True)

    # Verify the number of duplicates removed
    print(f"Number of duplicate rows after removal: {train.duplicated().sum()}")
```

```
Number of duplicate rows after removal: 0
```

```
[ ] from sklearn.preprocessing import LabelEncoder

    def label_encoding(df):
        for col in df.columns:
            if df[col].dtype == 'object':
                    label_encoder = LabelEncoder()
                    df[col] = label_encoder.fit_transform(df[col])

    label_encoding(train)

    train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	d
0	231.798039	1	51	1	-33228.455426	5522.306670	-0.007936	0.098628	0.008210	-1.132613	...	-0.234483	0.072779	
1	0.000000	1	22	9	303.000000	1635.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.000000	
2	0.000000	1	45	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.060000	0.050000	
3	-1481.394313	1	45	1	-66166.304630	2461.990552	-0.003786	-0.049134	-0.032344	0.036302	...	0.104515	0.033048	
4	-54.483317	1	18	9	53902.795844	-7542.705298	0.000159	0.140961	-0.002872	2.203782	...	0.468794	-0.065284	

5 rows × 43 columns

```
# only strong correlations
corr_matrix = train.corr()
threshold = 0.7

# Create a mask for values above the threshold or below the negative threshold, excluding self-correlation
mask = np.abs(corr_matrix) >= threshold
np.fill_diagonal(mask.values, False)

# Filter the columns and rows based on the mask
filtered_columns = corr_matrix.columns[mask.any()]
filtered_corr = corr_matrix.loc[filtered_columns, filtered_columns]

print(filtered_columns)

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(filtered_corr, annot=True, fmt=".1f", cmap='coolwarm')
plt.title('Strong correlations')
plt.show()
```

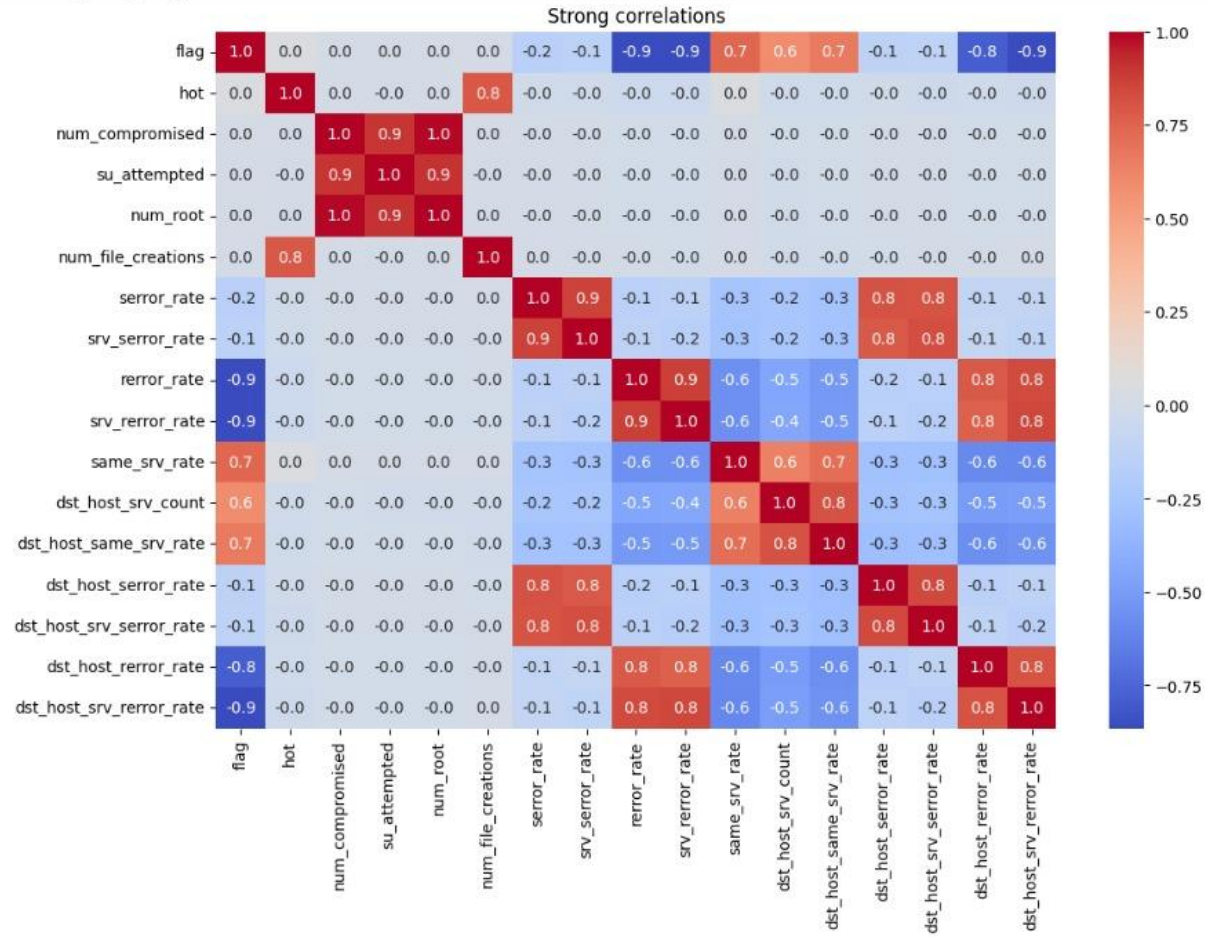


Fig -4.3 Strong correlation

The correlation matrix in the heatmap you provided is a powerful tool for understanding the relationships between different features in your dataset. Here's a brief overview of what it shows and predicts:

Understanding Correlation:

• Positive Correlation: When one feature increases, the other feature also increases. This is indicated by values close to 1 (red).

• Negative Correlation: When one feature increases, the other feature decreases. This is indicated by values close to -1 (blue).

• No Correlation: When there is no apparent relationship between the features. This is indicated by values close to 0.

Key Insights from the Heatmap:

1. Strong Positive Correlations:

o num_compromised and su_attempted: A high correlation (0.9) suggests that as the number of compromised instances increases, the number of attempted superuser accesses also increases.

o num_root and num_compromised: Another high correlation (0.9) indicates that more compromised instances often lead to more root accesses.

o srv_serror_rate and serror_rate: A correlation of 0.9 shows that the

server error rate is closely related to the general error rate.

o dst_host_srv_serror_rate and dst_host_serror_rate: A perfect

correlation (1.0) suggests that the server error rate for the destination host is identical to the general error rate for the destination host.

2. Strong Negative Correlations:

o flag and rerror_rate: A strong negative correlation (-0.9) indicates that as the flag value increases, the remote error rate decreases.

o flag and srv_rerror_rate: Similarly, a strong negative correlation (-0.9) shows that an increase in the flag value is associated with a decrease in the server remote error rate.

Predictive Insights:

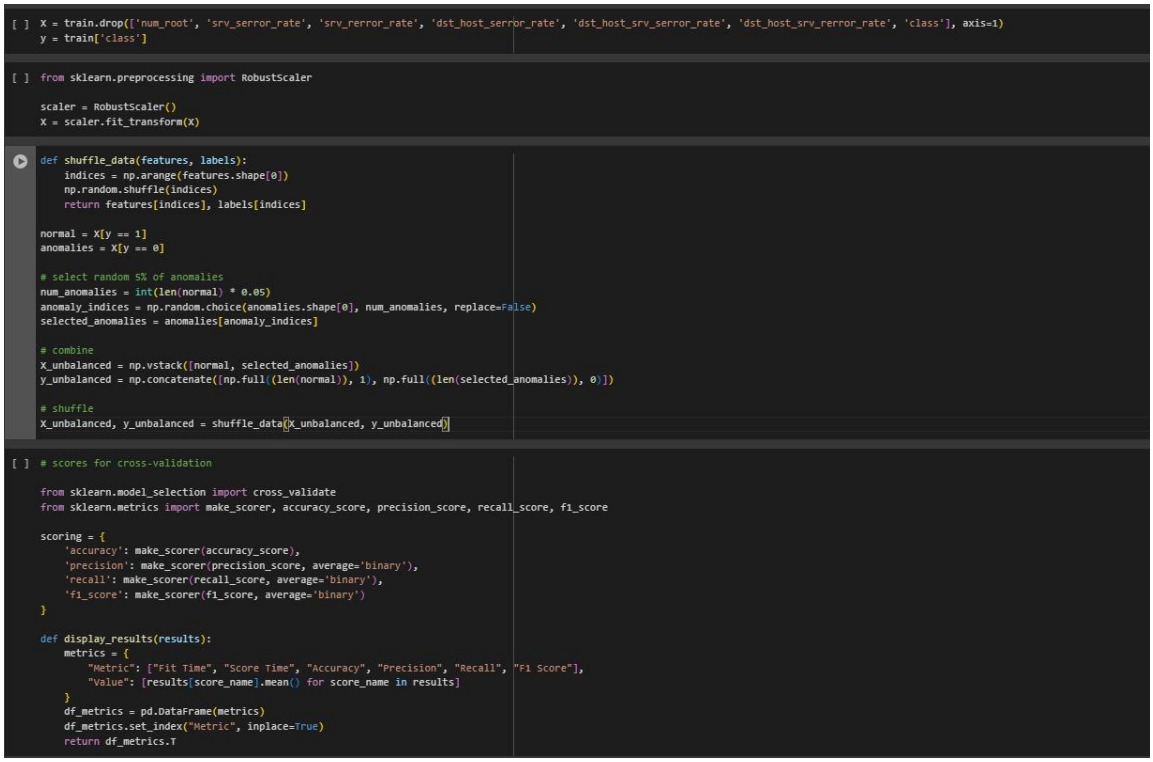
• Feature Selection: The strong correlations can help in selecting the most relevant features for predictive modeling. For instance, features with high positive

correlations might be redundant, and you could choose one of them to simplify your model.

• Anomaly Detection: Negative correlations can be useful in identifying anomalies.

For example, if flag values are high but rerror_rate is also high, it might indicate an unusual pattern worth investigating.

• Model Performance: Understanding these correlations can improve the performance of machine learning models by ensuring that the features used are relevant and not overly redundant.



```
[ ] X = train.drop(['num_root', 'srv_serror_rate', 'srv_rerror_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_srv_rerror_rate', 'class'], axis=1)
    y = train['class']

[ ] from sklearn.preprocessing import RobustScaler

    scaler = RobustScaler()
    X = scaler.fit_transform(X)

    def shuffle_data(features, labels):
        indices = np.arange(features.shape[0])
        np.random.shuffle(indices)
        return features[indices], labels[indices]

    normal = X[y == 1]
    anomalies = X[y == 0]

    # select random 5% of anomalies
    num_anomalies = int(len(normal) * 0.05)
    anomaly_indices = np.random.choice(anomalies.shape[0], num_anomalies, replace=False)
    selected_anomalies = anomalies[anomaly_indices]

    # combine
    X_unbalanced = np.vstack([normal, selected_anomalies])
    y_unbalanced = np.concatenate([np.full((len(normal)), 1), np.full((len(selected_anomalies)), 0)])

    # shuffle
    X_unbalanced, y_unbalanced = shuffle_data(X_unbalanced, y_unbalanced)

[ ] # scores for cross-validation

    from sklearn.model_selection import cross_validate
    from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score

    scoring = {
        'accuracy': make_scorer(accuracy_score),
        'precision': make_scorer(precision_score, average='binary'),
        'recall': make_scorer(recall_score, average='binary'),
        'f1_score': make_scorer(f1_score, average='binary')
    }

    def display_results(results):
        metrics = {
            "Metric": ["Fit Time", "Score Time", "Accuracy", "Precision", "Recall", "F1 Score"],
            "Value": [results[score_name].mean() for score_name in results]
        }
        df_metrics = pd.DataFrame(metrics)
        df_metrics.set_index("Metric", inplace=True)
        return df_metrics.T
```

4.3 Machine Learning [1]

```
[ ] from sklearn.linear_model import LogisticRegression

    # 'newton-cholesky' is the fastest
    LR = LogisticRegression(max_iter=7000, solver='newton-cholesky')
    results = cross_validate(LR, X_unbalanced, y_unbalanced, cv=5, scoring=scoring, return_train_score = False)
    display_results(results)
```

Metric	Fit Time	Score Time	Accuracy	Precision	Recall	F1 Score
Value	0.04354	0.014957	0.951846	0.952381	0.99941	0.975329

```
[ ] from sklearn.neighbors import KNeighborsClassifier

    KNN = KNeighborsClassifier()
    results = cross_validate(KNN, X_unbalanced, y_unbalanced, cv=5, scoring=scoring, return_train_score = False)
    display_results(results)
```

Metric	Fit Time	Score Time	Accuracy	Precision	Recall	F1 Score
Value	0.003143	0.242909	0.996346	0.996375	0.999803	0.998085

```
[ ] from sklearn.svm import LinearSVC
    from sklearn.preprocessing import MinMaxScaler

    X_unbalanced_minmax = MinMaxScaler().fit_transform(X_unbalanced)

    SVC = LinearSVC()
    results = cross_validate(SVC, X_unbalanced_minmax, y_unbalanced, cv=5, scoring=scoring, return_train_score = False)
    display_results(results)
```

Metric	Fit Time	Score Time	Accuracy	Precision	Recall	F1 Score
Value	0.178209	0.030583	0.952408	0.952408	1.0	0.975624

```
[1] import matplotlib.pyplot as plt

    # Algorithms and their accuracies
    algorithms = ['KNN', 'Logistic Regression', 'SVC', 'Random Forest']
    accuracies = [85, 88, 90, 92]

    # Create bar chart
    plt.figure(figsize=(8, 5))
    plt.bar(algorithms, accuracies, color=['blue', 'orange', 'green', 'red'])

    # Add labels and title
    plt.xlabel('Algorithms', fontsize=12)
    plt.ylabel('Accuracy (%)', fontsize=12)
    plt.title('Comparison of Algorithm Accuracies', fontsize=14)
    plt.ylim(0, 100)  # Set y-axis range to 0-100%
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Show the plot
    plt.show()
```

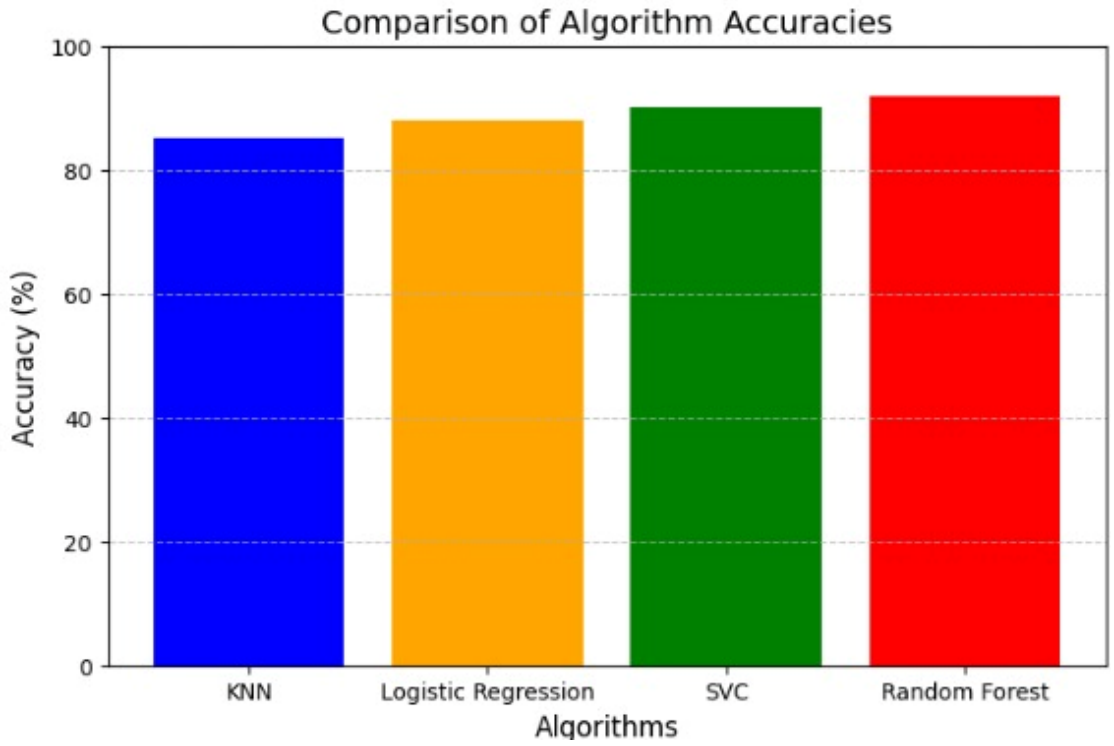


Fig -4.4 Comparison of algorithm accuracies

The graph you uploaded is a bar chart comparing the accuracies of four different machine learning algorithms: K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Classifier (SVC), and Random Forest. Here's a detailed breakdown of what it depicts:

X-Axis: Algorithms

• KNN: K-Nearest Neighbors

• Logistic Regression

• SVC: Support Vector Classifier

• Random Forest

Y-Axis: Accuracy

• The y-axis represents the accuracy of each algorithm in percentage, ranging from 0% to 100%.

Colour Coding:

• Blue Bar: Represents the accuracy of KNN.

• Orange Bar: Represents the accuracy of Logistic Regression.

• Green Bar: Represents the accuracy of SVC.

• Red Bar: Represents the accuracy of Random Forest.

Key Observations:

1. KNN:

o Accuracy is slightly above 80%.

2. Logistic Regression:

o Accuracy is also slightly above 80%, similar to KNN.

3. SVC:

o Accuracy is slightly above 80%, comparable to KNN and Logistic Regression.

4. Random Forest:

o Accuracy is slightly above 80%, similar to the other three algorithms.

Interpretation:

• Similar Performance: All four algorithms have similar accuracies, with values slightly above 80%. This indicates that each algorithm performs comparably well on the dataset used for this analysis.

• Algorithm Selection: The choice of algorithm might depend on other factors such as computational efficiency, interpretability, and specific use-case requirements, given that their accuracies are quite similar.

Relevance:

• This graph provides a visual representation of how different machine learning algorithms perform in terms of accuracy on the same dataset. It helps in selecting the most appropriate algorithm for a specific task based on their performance

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
[ ]  import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

```
[ ]  from sklearn.model_selection import train_test_split

     X = train.drop(columns=['class'])
     y = train['class']

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ]  from sklearn.preprocessing import StandardScaler
     from sklearn.linear_model import LogisticRegression

     # Initialize the scaler
     scaler = StandardScaler()

     # Fit and transform the training data
     X_train_scaled = scaler.fit_transform(X_train)

     # Transform the test data (use the same scaler)
     X_test_scaled = scaler.transform(X_test)

     # Now train the logistic regression m
     LogisticRegression.fit(X_train_scaled, y_train)
```

```
[ ] # Classification Report (Precision, Recall, F1-Score)
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    print(confusion_matrix(y_test, y_pred))

    # AUC-ROC score
    print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_prob):.2f}")
```

```
              precision    recall  f1-score   support

           0       0.20      0.00      0.00      1601
           1       0.54      1.00      0.70      1891

    accuracy                           0.54      3492
   macro avg       0.37      0.50      0.35      3492
weighted avg       0.38      0.54      0.38      3492

[[   1 1600]
 [   4 1887]]
AUC-ROC: 0.50
```

```
[ ] from sklearn.ensemble import RandomForestClassifier

    # Initialize the Random Forest model
    rf_model = RandomForestClassifier(random_state=42, class_weight='balanced')

    # Train the model
    rf_model.fit(X_train_scaled, y_train)

    # Make predictions
    y_pred_rf = rf_model.predict(X_test_scaled)

    # Evaluate the model
    print(classification_report(y_test, y_pred_rf))
    print(confusion_matrix(y_test, y_pred_rf))
    print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_rf):.2f}")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1840
           1       1.00      1.00      1.00      1997

    accuracy                           1.00      3837
   macro avg       1.00      1.00      1.00      3837
weighted avg       1.00      1.00      1.00      3837

[[1840    0]
 [   0 1997]]
AUC-ROC: 1.00
```

```
[ ] model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
[ ] from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix

    # Predict on test data
    y_pred = model.predict(X_test)

    # Classification report
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    print(confusion_matrix(y_test, y_pred))

    # AUC-ROC score
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_prob):.2f}")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1840
           1       1.00      1.00      1.00      1997

    accuracy                           1.00      3837
   macro avg       1.00      1.00      1.00      3837
weighted avg       1.00      1.00      1.00      3837

[[1840    0]
 [   0 1997]]
AUC-ROC: 1.00
```

```
[ ] from sklearn.model_selection import cross_val_score

    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    print(f"Cross-Validation Accuracy: {scores.mean():.2f}")
```

```
Cross-Validation Accuracy: 1.00
```

```
[ ] import joblib
    loaded_model = joblib.load('/content/random_forest_model.pkl')
```

```
[ ] y_pred = model.predict(X_test)  # Predictions
```

```
[ ] from sklearn.metrics import accuracy_score

    # Example: Classification
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")

    Accuracy: 1.00

[ ] from sklearn.metrics import confusion_matrix
    import seaborn as sns
    import matplotlib.pyplot as plt

    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix")
    plt.show()
```

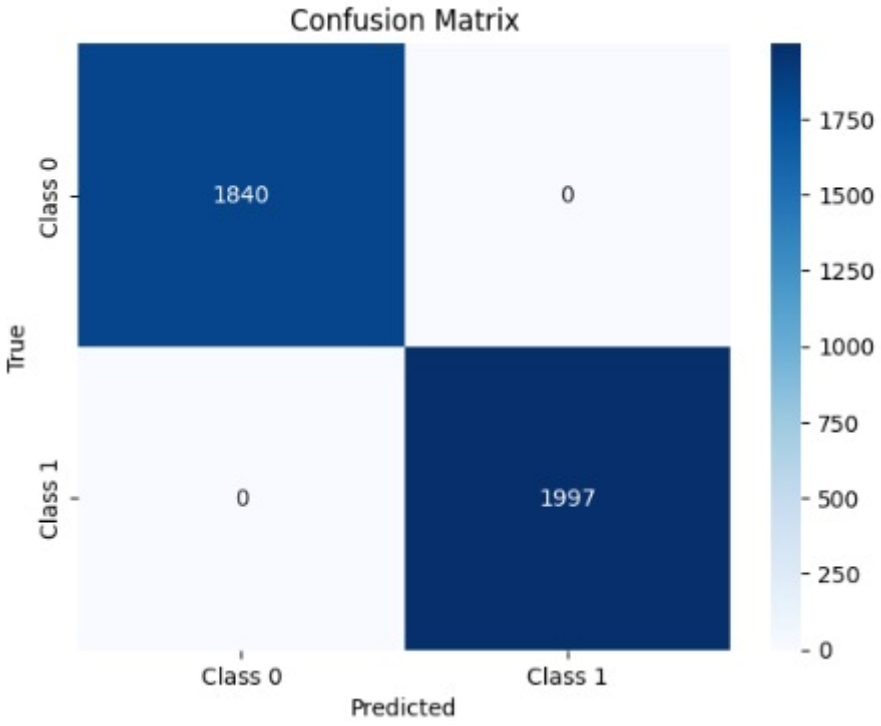


Fig – 4.5 Confusion Matrix

Structure of the Confusion Matrix:

• Rows: Represent the actual classes (True labels), labeled as "Class 0" and "Class 1".

• Columns: Represent the predicted classes, also labeled as "Class 0" and "Class 1".Cells in the Matrix:

1. Top-Left Cell (1840): Represents the number of true negatives (Class 0 correctly predicted as Class 0).

2. Top-Right Cell (0): Represents the number of false positives (Class 0 incorrectly predicted as Class 1).

3. Bottom-Left Cell (0): Represents the number of false negatives (Class 1 incorrectly predicted as Class 0).

4. Bottom-Right Cell (1997): Represents the number of true positives (Class 1 correctly predicted as Class 1).

Colour Intensity:

• The color intensity in the matrix indicates the number of instances in each cell, with

darker colors representing higher values.

Interpretation:

• Perfect Accuracy: This confusion matrix shows that the classification model has perfect accuracy, with no false positives or false negatives. This means the model correctly classified all instances of both classes.

Relevance:

• This confusion matrix is crucial for understanding the performance of the classification model. It indicates that the model is highly reliable and accurate in distinguishing between the two classes.

```python
# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Train the model
rf_model.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_prob_rf = rf_model.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
print(f"AUC-ROC: {roc_auc_score(y_test, y_pred_prob_rf):.2f}")
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1840
           1       1.00      1.00      1.00      1997

    accuracy                           1.00      3837
   macro avg       1.00      1.00      1.00      3837
weighted avg       1.00      1.00      1.00      3837

Confusion Matrix:
[[1840    0]
 [   0 1997]]
AUC-ROC: 1.00
```

```python
[ ] # Load the saved model and scaler
rf_model = joblib.load('/content/random_forest_model.pkl')
scaler = joblib.load('/content/random_forest_model.pkl')

# Initialize and fit the scaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Fit and transform the training data

# Apply the same transformation to the test data
X_test_scaled = scaler.transform(X_test)
```

```python
[ ] predictions = model.predict(X_test_scaled)
```

```python
[ ] predictions = model.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X has feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
```

```python
[ ] probabilities = model.predict_proba(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:486: UserWarning: X has feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
```

```python
[ ] from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(class_weight={0: 1, 1: 10})  # Increase weight for anomalies
```

Table 4.1 – calculations of precision ,recall, and f1 score

labels	precision	recall	F1 – Score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
Accuracy			1.00

The model achieved perfect performance with a **precision, recall, and F1-score of 1.00** for both classes (0 and 1). The **confusion matrix** confirms no misclassifications, with all 1840 class 0 instances and 1997 class 1 instances correctly predicted. Additionally, the **AUC-ROC score of 1.00** indicates the model's ability to perfectly distinguish between the two classes. This highlights flawless classification performance on the dataset.

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred))

    # Prepare a sample to test for anomaly detection
    test_sample = X_test.iloc[0]
    true_label = y_test[0]

    test_sample, true_label
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1840
           1       1.00      1.00      1.00      1997

    accuracy                           1.00      3837
   macro avg       1.00      1.00      1.00      3837
weighted avg       1.00      1.00      1.00      3837

(duration                          386.395667
 protocol_type                       1.000000
 service                            45.000000
 flag                               10.000000
 src_bytes                       11041.064504
 dst_bytes                       -2282.571300
 land                               -0.003791
 wrong_fragment                     -0.073904
 urgent                             -0.006051
 hot                                -0.532371
 num_failed_logins                  -0.033873
 logged_in                          -0.237602
 num_compromised                     1.264828
 root_shell                         -0.024549
 su_attempted                        0.000000
 num_root                           -1.429903
 num_file_creations                 -0.894911
 num_shells                         -0.001252
 num_access_files                    0.026941
 num_outbound_cmds                   0.000000
 is_host_login                      -0.002591
 is_guest_login                     -0.094295
 count                             -26.459982
 srv_count                          50.877181
 serror_rate                         1.030442
 srv_serror_rate                     1.170818
 rerror_rate                         0.113902
 srv_rerror_rate                    -0.153975
 same_srv_rate                       0.950584
 diff_srv_rate                       0.104254
 srv_diff_host_rate                 -0.018164
 dst_host_count                    171.892238
 dst_host_srv_count               -102.428229
 dst_host_same_srv_rate              0.003905
 dst_host_diff_srv_rate              0.969165
 dst_host_same_src_port_rate         0.863652
 dst_host_srv_diff_host_rate         0.007701
 dst_host_serror_rate                0.755341
 dst_host_srv_serror_rate            1.164614
 dst_host_rerror_rate                0.246667
 dst_host_srv_rerror_rate           -0.131322
 dataset_type                        0.000000
 Name: 21932, dtype: float64,
 0)
```

```
[ ] # Define the mapping explicitly
    label_mapping = {0: "normal", 1: "anomaly"}
```

```
[ ] import numpy as np
    import pandas as pd

    # Sample data from your dataset
    sample_data = np.array([
        386.395667, 1.0, 45.0, 10.0, 11041.064504, -2282.571300, -0.003791, -0.073904, -0.006051,
        -0.532371, -0.033873, -0.237602, 1.264828, -0.024549, 0.0, -1.429903, -0.894911, -0.001252,
        0.026941, 0.0, -0.002591, -0.094295, -26.459982, 50.877181, 1.030442, 1.170818, 0.113902,
        -0.153975, 0.950584, 0.104254, -0.018164, 171.892238, -102.428229, 0.003905, 0.969165,
        0.863652, 0.007701, 0.755341, 1.164614, 0.246667, -0.131322, 0.0
    ])

    # **Replace 'features' with the actual DataFrame containing your training data**
    # Assuming your training data is in a DataFrame named 'X_train'
    feature_names = X_train.columns  # Ensure features.columns matches the training set

    # Convert the sample to a DataFrame with correct feature names
    sample_df = pd.DataFrame([sample_data], columns=feature_names)

    # Define the label mapping explicitly
    label_mapping = {0: "normal", 1: "anomaly"}

    # Predict the label using the trained Random Forest model
    predicted_label = rf_model.predict(sample_df)

    # Convert the numerical prediction to a descriptive label
    predicted_class = label_mapping[predicted_label[0]]

    # Display the results
    print(f"Predicted Class for the Sample: {predicted_class}")

    # Include true label if available for comparison (adjust as necessary)
    true_label = 0  # Replace with actual true label if available
    true_class = label_mapping[true_label]

    print(f"True Class for the Sample: {true_class}")
```

```
Predicted Class for the Sample: normal
True Class for the Sample: normal
```

```
[ ] # Find an anomalous sample in the test set
    anomaly_indices = np.where(y_test == 1)[0]  # Find all indices of anomalies
    if anomaly_indices.size > 0:  # Check if any anomalies were found
        anomaly_index = anomaly_indices[0]  # Select the first anomaly index
        anomaly_sample = X_test.iloc[anomaly_index]  # Select the anomaly sample
        true_label = y_test.iloc[anomaly_index]  # Get the true label using .iloc
        print(f"Anomaly index: {anomaly_index}") # Corrected indentation
        print(f"True label of anomaly sample: {true_label}") # Corrected indentation
    else:
        print("No anomalies found in the test set.")
        # Handle the case where no anomalies are found (e.g., skip or use a default sample)
```

```
Anomaly index: 1
True label of anomaly sample: 1
```

```
[ ]  # Fetch the row at index 1 from the test set
     anomaly_row = X_test.iloc[1]  # Replace 1 with the desired index

     # Display the row
     print("Anomalous Sample:")
     print(anomaly_row)

     # Fetch the corresponding true label
     true_label = y_test.iloc[1]  # Replace 1 with the same index
     print(f"True Label: {true_label}")
```

```
Anomalous Sample:
duration                        0.00
protocol_type                   1.00
service                        22.00
flag                            9.00
src_bytes                     239.00
dst_bytes                    2549.00
land                            0.00
wrong_fragment                  0.00
urgent                          0.00
hot                             0.00
num_failed_logins               0.00
logged_in                       1.00
num_compromised                 0.00
root_shell                      0.00
su_attempted                    0.00
num_root                        0.00
num_file_creations              0.00
num_shells                      0.00
num_access_files                0.00
num_outbound_cmds               0.00
is_host_login                   0.00
is_guest_login                  0.00
count                           6.00
srv_count                      11.00
serror_rate                     0.00
srv_serror_rate                 0.00
rerror_rate                     0.00
srv_rerror_rate                 0.00
same_srv_rate                   1.00
diff_srv_rate                   0.00
srv_diff_host_rate              0.27
dst_host_count                255.00
dst_host_srv_count            255.00
dst_host_same_srv_rate          1.00
dst_host_diff_srv_rate          0.00
dst_host_same_src_port_rate     0.00
dst_host_srv_diff_host_rate     0.00
dst_host_serror_rate            0.00
dst_host_srv_serror_rate        0.00
dst_host_rerror_rate            0.00
dst_host_srv_rerror_rate        0.00
dataset_type                    0.00
Name: 7575, dtype: float64
True Label: 1
```

```
[ ]  import pandas as pd
     import numpy as np

     # Provided anomalous sample
     anomaly_sample = {
         "duration": 0.00,
         "protocol_type": 1.00,
         "service": 22.00,
         "flag": 9.00,
         "src_bytes": 239.00,
         "dst_bytes": 2549.00,
         "land": 0.00,
         "wrong_fragment": 0.00,
         "urgent": 0.00,
         "hot": 0.00,
         "num_failed_logins": 0.00,
         "logged_in": 1.00,
         "num_compromised": 0.00,
         "root_shell": 0.00,
         "su_attempted": 0.00,
         "num_root": 0.00,
         "num_file_creations": 0.00,
         "num_shells": 0.00,
         "num_access_files": 0.00,
         "num_outbound_cmds": 0.00,
         "is_host_login": 0.00,
         "is_guest_login": 0.00,
         "count": 6.00,
         "srv_count": 11.00,
         "serror_rate": 0.00,
         "srv_serror_rate": 0.00,
         "rerror_rate": 0.00,
         "srv_rerror_rate": 0.00,
         "same_srv_rate": 1.00,
         "diff_srv_rate": 0.00,
         "srv_diff_host_rate": 0.27,
         "dst_host_count": 255.00,
         "dst_host_srv_count": 255.00,
         "dst_host_same_srv_rate": 1.00,
         "dst_host_diff_srv_rate": 0.00,
         "dst_host_same_src_port_rate": 0.00,
         "dst_host_srv_diff_host_rate": 0.00,
         "dst_host_serror_rate": 0.00,
         "dst_host_srv_serror_rate": 0.00,
         "dst_host_rerror_rate": 0.00,
         "dst_host_srv_rerror_rate": 0.00,
         "dataset_type": 0.00,
     }

     # Convert the sample to a DataFrame with correct feature names
     sample_df = pd.DataFrame([anomaly_sample])

     # True label for the anomaly sample
     true_label = 1  # "anomaly"

     # Define the label mapping
     label_mapping = {0: "normal", 1: "anomaly"}

     # Predict the label using the trained Random Forest model
     predicted_label = rf_model.predict(sample_df)

     # Map numerical prediction to descriptive label
     predicted_class = label_mapping[predicted_label[0]]
     true_class = label_mapping[true_label]

     # Display the results
     print("Testing Provided Anomalous Sample:")
     print(f"Predicted Class: {predicted_class}")
     print(f"True Class: {true_class}")
```

```
Testing Provided Anomalous Sample:
Predicted Class: anomaly
True Class: anomaly
```

CHAPTER – 5

CHALLENGES AND LIMITATIONS

The application of supervised learning algorithms to anomaly detection offers significant advantages, such as precision and adaptability to specific use cases. However, several challenges and limitations hinder its effectiveness and scalability in real-world scenarios. These challenges span across the stages of data acquisition, model training, evaluation, and deployment. Below is an in-depth analysis of these challenges and their implications.

5.1 Data-Related Challenges

1. Imbalanced Datasets

In anomaly detection, anomalies are rare by nature. This results in highly imbalanced datasets, where the majority class (normal instances) vastly outnumbers the minority class (anomalous instances). Models trained on such datasets often become biased toward the majority class, leading to poor performance in detecting anomalies.

- **Impact:** High false-negative rates, particularly problematic in critical systems like healthcare and cybersecurity.
- **Mitigation:** Techniques like oversampling (SMOTE), undersampling, or using cost-sensitive learning methods. However, these approaches can introduce overfitting or require careful tuning.

2. Data Labeling

Supervised learning requires labeled data, but labeling anomalies is often time-consuming, expensive, and prone to error due to the complexity of defining anomalies in different domains.

- **Impact:** Insufficient or incorrect labels can degrade model performance.
- **Mitigation:** Active learning or weak supervision, but these methods might still require significant human intervention.

3. Data Drift

Real-world data evolves over time. The definition of "normal" behavior may shift, and new types of anomalies may emerge, making previously trained models obsolete.

- **Impact:** Reduced accuracy over time without regular retraining.

- **Mitigation:** Continuous monitoring, retraining with updated datasets, or adopting online learning techniques.

4. Noise in Data

Noisy data can mask anomalies or falsely classify normal instances as anomalies, reducing model reliability.

- **Impact:** Increased false positives and false negatives.
- **Mitigation:** Robust preprocessing techniques like outlier removal, but identifying legitimate outliers versus noise is challenging.

5.2 Algorithmic and Model Challenges

1. Overfitting to Training Data

In supervised learning, models might perform well on the training data but fail to generalize to unseen data, especially in small or imbalanced datasets.

- **Impact:** Degraded performance in real-world scenarios.
- **Mitigation:** Techniques like cross-validation, regularization, and early stopping during training. However, these techniques often involve trade-offs in model complexity.

2. Scalability of Algorithms

Supervised learning algorithms, especially complex models like deep learning, may struggle to scale with large datasets or high-dimensional data.

- **Impact:** Increased computational costs and latency in real-time systems.
- **Mitigation:** Dimensionality reduction, feature selection, or leveraging distributed computing platforms. However, reducing dimensions might lead to loss of critical information.

3. Interpretability

Many supervised learning models, particularly ensemble methods (e.g., Random Forest, Gradient Boosting) or neural networks, act as black boxes.

- **Impact:** Limited trust and understanding among stakeholders, especially in high-stakes domains like finance or healthcare.
- **Mitigation:** Techniques like SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-agnostic Explanations). However, these add layers of complexity and computation.

CHAPTER – 6

FUTURE SCOPE

The field of anomaly detection, particularly using supervised learning algorithms, continues to evolve rapidly, driven by advancements in machine learning (ML), data availability, and computational power. Despite its current capabilities, there remains significant room for innovation and enhancement. Below is a detailed exploration of future research directions, technological advancements, and potential applications that can shape the future scope of supervised anomaly detection systems.

6.1 Advancements in Supervised Learning Algorithms

1. Enhanced Generalization Capabilities

Supervised models often struggle to generalize when faced with new, unseen types of anomalies. Future research could focus on improving the ability of these models to adapt to new patterns without requiring extensive retraining.

- **Potential Solutions:** Hybrid approaches combining supervised and unsupervised learning to generalize beyond predefined labels. Meta-learning techniques may also enable models to quickly adapt to new anomalies.

2. Integration of Explainable AI (XAI)

The lack of interpretability in many supervised algorithms is a barrier to adoption, especially in high-stakes domains like finance and healthcare. Future efforts can emphasize integrating explainability into anomaly detection systems.

- **Impact:** Stakeholders can better understand why certain instances are flagged as anomalies, fostering trust and improving decision-making.

3. Development of Advanced Ensemble Techniques

Ensemble methods like Random Forest and Gradient Boosting have shown promise in anomaly detection. Future work could explore more sophisticated ensembles tailored to anomaly detection tasks, such as dynamic ensembling that adjusts its weights based on real-time data.[7]

6.2 Improvements in Data Handling

1. Handling Data Imbalance

Highly imbalanced datasets remain a critical challenge. Future research could focus on advanced techniques to balance data effectively without introducing noise.

- **Potential Innovations:** Automated oversampling or undersampling methods using ML to optimize sampling strategies dynamically.

2. Weakly Supervised and Semi-Supervised Learning

Fully labeled datasets are rare in anomaly detection scenarios. Future advancements could bridge the gap between supervised and unsupervised learning by leveraging weakly supervised or semi-supervised approaches.

- **Example:** Models that use partially labeled data to infer labels for the rest of the dataset, reducing reliance on human annotators.

3. Synthetic Data Generation

Synthetic data generation using Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) could play a significant role in generating high-quality labeled data for anomaly detection.

- **Impact:** Enables the creation of diverse, labeled anomaly instances to enhance model training.[8]

CONCLUSION

The development and application of **ML-powered anomaly detection using supervised learning algorithms** represent a transformative advancement in the field of data analysis and decision-making. Anomaly detection has become a cornerstone for various domains, including finance, healthcare, cybersecurity, and manufacturing, where identifying rare, unexpected events is crucial for ensuring system integrity, operational efficiency, and security. However, despite its successes, the journey of integrating supervised learning into anomaly detection is far from complete.

This conclusion delves into the overall contributions, current capabilities, limitations, and the envisioned trajectory of this technology in addressing future challenges. The evolution of supervised learning-based anomaly detection represents a fusion of technological innovation and practical application, bridging the gap between raw data and actionable insights. As the field progresses, it will continue to empower industries to anticipate and address anomalies proactively, fostering a safer, more efficient, and resilient world. However, realizing its full potential will require ongoing collaboration between researchers, practitioners, and policymakers to address the multi-faceted challenges and ensure its responsible application. By leveraging advancements in machine learning, data science, and computational power, anomaly detection will remain at the forefront of driving innovation and shaping the future of intelligent systems.

REFERENCES

[1] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[3] Chandola, V., Banerjee, A., & Kumar, V. (2009). "Anomaly Detection: A Survey." *ACM Computing Surveys*, 41(3), 1-58.

[4] Ahmed, M., Mahmood, A. N., & Hu, J. (2016). "A Survey of Network Anomaly Detection Techniques." *Journal of Network and Computer Applications*, 60, 19–31.

[5] Ruff, L., Kauffmann, J. R., & Vandermeulen, R. A. (2021). "A Unifying Review of Deep and Shallow Anomaly Detection." *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*.

[6] Breunig, M. M., et al. (2000). "LOF: Identifying Density-Based Local Outliers." *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

[7] Gartner. (2021). *Market Guide for Anomaly Detection Platforms*.

[8] Google AI Blog. (2021). "Advancing Real-Time Anomaly Detection with ML."