Kingdom of Saudi Arabia
Ministry of Higher Education
King Abdul Aziz University
Faculty of Computing and IT
Computer Science Department

Cyber Threat Detection Using Artificial Intelligence

Submitted by

Muhammed Talat Joharji

Mohannad Abdullah Aljaddawi

Hashem Mohsen Baroom

Supervisor

Asif Irshad Khan

11, 1445 – 5, 2024

CYBER THREAT DETECTION USING ARTIFICIAL INTELLIGENCE

Submitted by
Muhammed Talat Joharji
2037729
Mohannad Abdullah AL-Jaddawi
2036459
Hashem Mohsin Baroom
2037062

Faculty of Computing and Information Technology

King Abdul Aziz University

Jeddah – Saudi Arabia

DHUL-QA’DAH 1445 – MAY 2024

بسم الله الرحمن الرحيم

قال تعالى: ﴿لَّا خَيْرَ فِي كَثِيرٍ مِّن نَّجْوَاهُمْ إِلَّا مَنْ أَمَرَ بِصَدَقَةٍ أَوْ مَعْرُوفٍ أَوْ إِصْلَاحٍ بَيْنَ النَّاسِ ۚ وَمَن يَفْعَلْ ذَٰلِكَ ابْتِغَاءَ مَرْضَاتِ اللَّهِ فَسَوْفَ نُؤْتِيهِ أَجْرًا عَظِيمًا﴾

قال رسولُ اللهِ صلَّى اللهُ عليه وسلَّم (أحَبُّ النَّاسِ إلى اللهِ أنفَعُهم للنَّاسِ وأحَبُّ الأعمالِ إلى اللهِ سُرورٌ تُدخِلُه على مُسلِمٍ أو تكشِفُ عنه كُربةً) أخرجه الطبراني: 6026

Certification of Research

I certify that I have reviewed this project done by: Muhammed Talat Joharji- 2037729, Mohannad AL-Jaddawi - 2036459, and Hashem Mohsin Baroom – 2037062 during the 1st or 2nd semester of the academic year 2024. And I approve the project submission for discussion by the project responsible committee to grade it as the graduation project degree in Student Department.

Supervisors: Dr. Asif I Khan

Signature………………… :

Declaration

We certify that the implementation of this graduation project has been done by our own efforts. This includes the preparation and writing of the analysis, design, programming and documentation. We have got assistance during the implementation period of this project, after Allah, from the references mentioned in this thesis, and Allah is the witnesses of what we say.

Student Name: Muhammed Talat Joharji
Student ID: 2037729
E-mail: mmohammedhassnjoharji@stu.kau.edu.sa
Mobile: +966504290924

Signature: [illegible]

Student Name: Mohannad Abdullah Al Jaddawi
Student ID: 2036459
E-mail: mabdulhafeezaljaddawi@stu.kau.edu.sa
Mobile: +966543200094

Signature: [illegible]

Student Name: Hashem Mohsin Barrom
Student ID: 2037062
E-mail: hhbaroom@stu.kau.edu.sa
Mobile: +966569163059

Signature: [illegible]

Cyber Threat Detection Using Artificial Intelligence

Abstract

In response to the accelerating sophistication of cyber threats, this research endeavors to enhance cybersecurity through the implementation of artificial intelligence (AI) techniques, specifically focusing on the domain of anomaly detection. The project's primary objective is to develop strong AI models capable of effectively identifying unusual patterns and potential cyber threats within network logs. To achieve this, we employ feature engineering to clean valid characteristics for accurate anomaly detection. Developing well-established machine learning and deep learning frameworks such as scikit-learn, and TensorFlow, comprehensive testing and validation processes are undertaken to precisely evaluate the models' performance metrics.

The results obtained exhibit promising outcomes, showcasing notable achievements in terms of accuracy, precision, recall, F1-score, and specificity. These metrics collectively affirm the effectiveness of our AI-based approach in securing cybersecurity defenses. Considering the dynamic and evolving nature of cyber threats, this research contributes significantly to the ongoing discussion on encouraging digital resistance. By handing a practical defense mechanism grounded in advanced AI algorithms, our work addresses the constraint need for adaptive and effective solutions to counter the ever-evolving landscape of cybersecurity challenges.

The project's focus on anomaly detection is particularly relevant, given that it is a critical component of modern cybersecurity defenses. By identifying unusual patterns and behaviors, our AI models can help detect and prevent cyber-attacks before they cause significant damage. This research has the potential to revolutionize the field of cybersecurity, providing a powerful tool for protecting against emerging threats and safeguarding sensitive data.

الكشف عن التهديدات السيبرانية باستخدام الذكاء الاصطناعي

استفتاحية

مشروع الكشف عن التهديدات السيبرانية باستخدام الذكاء الاصطناعي يهدف إلى تعزيز الأمن السيبراني من خلال تنفيذ تقنيات الذكاء الاصطناعي، مع التركيز بشكل خاص على مجال كشف الأخطاء غير العادية. الهدف الرئيس للمشروع هو تطوير نماذج ذكاء اصطناعي قوية قادرة على تحديد الأنماط غير العادية والتهديدات السيبرانية المحتملة بشكل فعال داخل سجلات الشبكة وبيانات سلوك النظام. لتحقيق هذا الهدف، نستخدم هندسة الميزات لتنظيف الخصائص الصالحة للكشف عن الأخطاء غير العادية بدقة. يتم إجراء عمليات اختبار وتحقق شاملة باستخدام إطارات تعلم الآلة المعتمدة بشكل جيد مثل scikit-learn و TensorFlow لتقييم بدقة مقاييس أداء النماذج.

تظهر النتائج المحصلة نتائج مشجعة، حيث تعرض إنجازات ملحوظة من حيث accuracy و precision و recall و specificity و F1-score تؤكد هذه المقاييس بشكل جماعي فعالية نهجنا القائم على الذكاء الاصطناعي في تأمين الدفاعات السيبرانية. نظرًا للطبيعة الديناميكية والمتطورة للتهديدات السيبرانية، تسهم هذه الدراسة بشكل كبير في المناقشة المستمرة حول تشجيع المقاومة الرقمية. من خلال تقديم آلية دفاعية عملية مرتكزة على خوارزميات الذكاء الاصطناعي المتقدمة، يتناول عملنا الحاجة المحدودة لحلول متكيفة وفعالة لمواجهة المناظر السيبرانية المتطورة باستمرار. يعد التركيز على كشف الأخطاء غير العادية في المشروع ذو صلة خاصة، نظرًا لأنه عنصر حرج في الدفاعات السيبرانية الحديثة.

من خلال تحديد الأنماط والسلوكيات غير العادية، تساعد نماذج الذكاء الاصطناعي الخاصة بنا في الكشف عن الهجمات السيبرانية ومنعها قبل أن تسبب أضرار كبيرة. تتمتع هذه الدراسة بإمكانية ثورية في مجال الأمن السيبراني، حيث توفر أداة قوية للحماية من التهديدات الناشئة وحماية البيانات الحساسة.

Dedicated to

To our families who guided us, to our teachers who shared knowledge, and to our community that we're eager to assist.

Acknowledgement

We extend our heartfelt gratitude to Dr. Asif Khan, our project supervisor, for his invaluable guidance, expertise, and continuous support throughout our Senior Project on Cyber Threat Detection Using Artificial Intelligence. Also, we express our sincere gratitude to Dr. Emad Jaha for his invaluable guidance and thoughtful care, particularly in assisting us with the theory of AI. His support and encouragement have not only motivated us but also significantly enhanced our confidence in our project. Furthermore, we express our gratitude to our course supervisors for their invaluable assistance in guiding us through the development of this project over the past year. We are also thankful for the principles and guidelines they provided, which were significant in completing this report.

Table of Content

List of Figures

List of Tables

List of Symbols / Abbreviation

TP:	True-Positive
TN:	True-Negative
FP:	False-Positive
FN:	False-Negative
TNR:	True Negative Rate
TPR:	True Positive Rate
FNR:	False Negative Rate
FPR:	False Positive Rate

GRU: Gated Recurrent Unit

XGBoost: Extreme Gradient Boosting

Chapter 1

PROJECT INITIATION

Chapter 1
Project Initiation

In this project, we're diving into 'Cyber Threat Detection Using Artificial Intelligence' to boost cybersecurity with the help of Artificial Intelligence. The project's goals are well thought out and perfectly fit into our project's broader topic. They're clearly defined, matching the project's purpose, and offering real benefits to everyone involved. The specific title of this project is: "AI Cyber Threat Detection: Enhancing Cybersecurity Through Artificial Intelligence."

The aim of this project is carefully designed to fall within the scope of the broad project topic and address the identified objectives. These aims are clearly stated and aligned with the purpose of the project. They provide a tangible outcome, benefiting both the project and its stakeholders. Our main vision is as follows:

1. **Develop Anomaly Detection Models.**
   - **Statement:** Develop and implement efficient Anomaly Detection models for enhanced cybersecurity.
   - **Clarity of Purpose:** The project aims to create advanced machine learning and deep learning models capable of identifying anomalies in network traffic and the purpose is to enhance the accuracy and efficiency of cyber threat detection.
2. **Evaluation and Testing of Detection Tools**
   - **Statement:** Evaluate the performance of the developed tools and validate their functionality to ensure accurate threat detection.
   - **Clarity of Purpose:** This aim emphasizes rigorous testing and evaluation of the tools created in the project. It aims to validate the accuracy and reliability of the detection models and systems, providing actionable insights for improvements.
3. **Balancing Biased Datasets and Enhancing Accuracy**
   - **Statement:** Implement strategies to balance biased datasets and improve the accuracy of the detection system.
   - **Clarity of Purpose:** This aim focuses on addressing the challenges of biased datasets by employing techniques to balance them. Enhancing accuracy is crucial for the effective functioning of the cyber threat detection system.

These aims collectively define the purpose and tangible outcomes of the project, aligning with the objectives outlined in the earlier sections. They provide a clear direction for the project's execution and highlight the potential impact it can have on the field of cybersecurity.

1.1 Problem Definition:

Detecting cyber-attacks while surfing the internet presents a significant challenge. Our project addresses several key issues in this regard. Firstly, hackers constantly innovate new attack methods, making defense efforts increasingly difficult. Secondly, traditional network security systems rely on static databases and struggle to adapt to the rapidly evolving threat landscape. Lastly, a widespread lack of awareness and knowledge about cybersecurity leaves individuals and organizations vulnerable to exploitation.

1.2 Project Objectives:

The project aims to develop an AI-based Cyber Threat Detection system for identifying anomalies in network logs. Key objectives include optimizing data processing efficiency, implementing performance evaluation tools, ensuring reliability, and exploring interoperability. In addition, the system will be capable to be upgraded into more advanced topics like, real-time detection, inference, Automated Response for the future work with fully documented details of the essential steps of the project that might help on developing these fields. Data quality challenges will be addressed, and optional features like cyber-attack type labels will be exploited if time permit.

1.3 Project Scope:

The project's scope involves developing and implementing AI-driven anomaly detection models within the field of cybersecurity. The focus will be on monitoring network traffic to swiftly identify potential cyber threats. Here are the included and excluded content of the project:

1. **Included**
   - Data Collection.
   - Data Pre-processing.
   - Feature Engineering.
   - Feature Selection.
   - Anomaly Detection Module.
   - Performance Evaluation Tools.
2. **Excluded**
   - Real-Time Processing.
   - System User Interface.
   - Integration to Other Anti-Virus Applications.

1.4 **Anomaly Detection:**

Anomaly detection is a realistic and comprehensive solution to the identified problems. Anomaly detection employs advanced machine learning and deep learning algorithms, to identify patterns that acquired by the expected behaviour within network traffic. By focusing on anomalies, system can rapidly detect potential cyber threats, thus enhancing overall cybersecurity.

1.5 **Methods to Find Information**

In our exploration of methods to gather crucial insights, document reviews stand as a rich resource for theoretical knowledge on AI-based cyber threat detection. While they offer a comprehensive understanding, they may lack the practical side and access to registered details. In parallel, testing provides experimental data and performance metrics vital for project development and validation, although it can be resource-intensive and primarily technical in nature. Together, these methods guide our journey to enhance the AI-based cyber threat detection system.

1.5.1 **Document Reviews**

Strengths of document reviews provide a wealth of existing information and research relevant to the project's context. It allows for a comprehensive understanding of the theoretical aspects of cyber threat detection using AI. However, its weaknesses may not offer practical insights, as they rely on existing publications. Also, the information obtained may be limited to what's publicly available and may not include sensitive details.

1.5.2 **Testing**

Likewise, the strengths of testing provide realistic data and performance metrics, crucial for project development and evaluation. It allows for systematic validation of the AI-based cyber threat detection models. Nevertheless, comprehensive testing can be resource-intensive and time-consuming. Moreover, it primarily focuses on the project's technical aspects and may not provide insights into the broader cybersecurity landscape.

1.5.3 **Target Users:**

Cybersecurity experts and analysts responsible for safeguarding organizational assets and data. Businesses and individuals are seeking advanced cybersecurity measures to protect their digital presence, providing a safer digital environment, and minimizing the potential risks associated with cyber threats.

1.6 **Most Effective Approach**

For the purposes of our project, which involves the development of AI-driven cyber threat detection models, the most effective approach is a combination of "Document Reviews" and "Testing", although "Document Reviews" provide the essential theoretical foundation and background knowledge required for AI-based cyber threat detection. They help you understand the existing landscape and the state of the art. In contrast "Testing" is crucial for validating the practical application of the models and collecting empirical data to assess their effectiveness. In most cases, it aligns with the project's technical focus. To sum up, combining these approaches, we strike a balance between theoretical understanding and practical validation, ensuring that the project is well-informed, and its outcomes are carefully tested.

Chapter 2
PROJECT PLANNING

Chapter 2
Project Planning

2.1 Project Scheduling

Our project starts from 22/8/2023 which was 1st week of the semester and ends on 5th of December. Table 1.1 shows the project plan or the Work Break Down Structure (WBS):

Task Name	Duration	Start	Finish
CyberThreatDetectionUsingAi	**115 days**	**Thu 10/19/23**	**Wed 3/27/24**
Project Initiation	**2 wks**	**Thu 10/19/23**	**Wed 11/1/23**
Find Project Idea	1 wk	Thu 10/19/23	Wed 10/25/23
Define Problem Statement	3 days	Thu 10/26/23	Mon 10/30/23
Develop Project Team	2 days	Thu 10/19/23	Fri 10/20/23
Define Scope & Requirements	2 days	Tue 10/31/23	Wed 11/1/23
Project Planning	**7 days**	**Thu 11/2/23**	**Fri 11/10/23**
Define Project Milestone	1 day	Thu 11/2/23	Thu 11/2/23
Develop Project Schedule (Gantt Chart)	1 day	Fri 11/3/23	Fri 11/3/23
Read Literature Review	5 days	Mon 11/6/23	Fri 11/10/23
Project Specification	**10 days**	**Mon 11/13/23**	**Fri 11/24/23**
Gather Relevent Datasets	3 days	Mon 11/13/23	Wed 11/15/23
Explore Feature Engineering	1 day	Thu 11/16/23	Thu 11/16/23
Define Requirements	7 days	Thu 11/16/23	Fri 11/24/23
Define Tools & Softwares	3 days	Fri 11/17/23	Tue 11/21/23
Design & Skills	**6 days**	**Wed 11/22/23**	**Wed 11/29/23**
Create High-Level Model	3 days	Mon 11/27/23	Wed 11/29/23
Define Required Skills	2 days	Wed 11/22/23	Thu 11/23/23
Initialize Dataset	**15 days**	**Thu 11/16/23**	**Wed 12/6/23**
Data Collection	1 wk	Thu 11/16/23	Wed 11/22/23
Data Preprocessing	1 wk	Thu 11/23/23	Wed 11/29/23
Feature Selection	1 wk	Thu 11/30/23	Wed 12/6/23
Implementation	**60 days**	**Thu 12/7/23**	**Wed 2/28/24**
Model Training	40 days	Thu 12/7/23	Wed 1/31/24
Model Validation	10 days	Thu 2/1/24	Wed 2/14/24
Model Testing	10 days	Thu 2/15/24	Wed 2/28/24
Testing	**20 days**	**Thu 2/29/24**	**Wed 3/27/24**
Machine Learning Testing	1 mon	Thu 2/29/24	Wed 3/27/24
Deep Learning Testing	1 mon	Thu 2/29/24	Wed 3/27/24

Table 2.1.1 – Project Work Breakdown Structure Milestone

This project schedule outlines the journey from project initiation to the testing phase. As shown in Figure 1.1, It includes phases like data collection, model training and testing. While real-time monitoring and automation aren't in scope, the project aims to enhance cybersecurity through AI-driven anomaly detection. Additionally, it covered the second phase of the senior project which concerns the technical implementation yet, it is not accurately assigned in correct time but as an initial scheduling and milestone of the project identification.

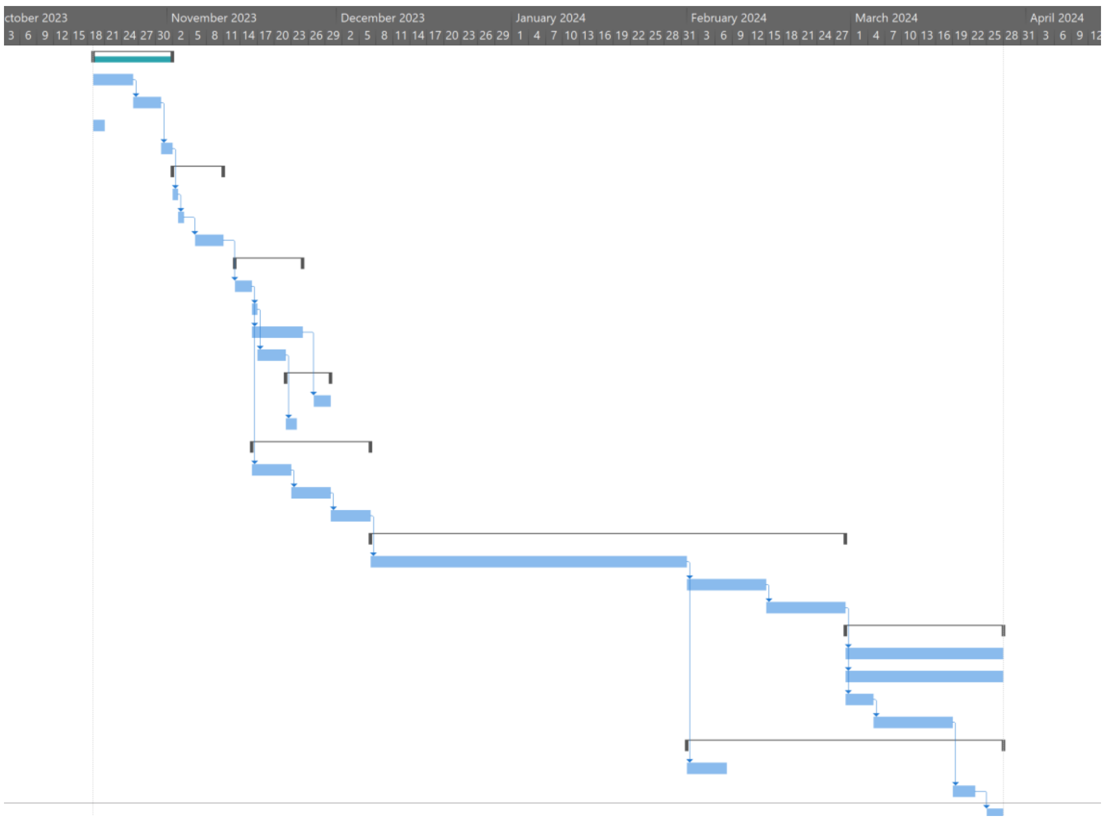


Figure 2.1.1 – Project Gantt Chart

2.2 Literature Review

In this section, this study will provide details of different papers or research regarding various perspectives for the project idea including techniques used, official dataset understanding, Anomaly Detection in cyber security and prediction methods for machine learning and deep learning. Also, a PROMPT table will be given for each paper or research with some explanation for the key points related to the project in [Table 2.1, Table 2.2, and Table 2.3].

2.3 Research of Security Process Using Anomaly Detection

The first one is by Elsevier B.V. Information Fusion/ARRS, 2023. This thorough paper, which drew on the protocols of the National Institute of Standards and Technology (NIST), covered a wide range of security procedures. The usefulness of AI in navigating through various security stages was at the heart of its theory. Notably, K-means and Random Forest were primarily used in the research as classification or clustering methods from machine learning. The study made the case that these techniques are essential for many detection modalities, such as the detection of nuclear power plants, network traffic, and OS classifications. The broad scope, meanwhile, diverged from our project's exclusive concentration on cyberspace anomaly identification.

2.3.1 PROMPT Criteria of the paper:

• **Presentation:** Although the article takes a holistic approach, readers looking for specific cybersecurity information may find it difficult to navigate due to its broad reach.

• **Relevance:** Although the paper covers AI security, our study is looking at specific cybersecurity applications, thus its broad lens may not be totally relevant.

• **Objectivity:** Although it encompasses a range of methods and procedures, it maintains an impartial attitude by avoiding any inherent prejudice.

• **Approach:** The article used a variety of machine learning techniques; however, its approaches might not be as specialized for cyber-attacks.

• **Provenance:** The publication by Elsevier B.V. and the mention of NIST protocols maintain the validity of the paper.

• **Timeliness:** Although more comprehensive in its approach, it is a recent addition that was published in 2023.

Table 2.1 shows the PROMPT criteria of Elsevier paper explaining the similarities, differences and summary for our project and their paper.

Criteria	Elsevier B.V. Paper	Our Project Report
Research Summary	Expansive coverage on AI security processes.	Focused on anomaly detection in cybersecurity.
Similarities and Differences	Both emphasize the importance of AI in security. However, the paper is broader in scope.	Project narrows down to anomaly detection in the cyber world.
Information Collection and Problem Addressing	Uses a general approach to address security issues, not just cyber threats.	Specifically targets anomalies in network.
Gaps or Shortcomings Addressed by Our Project	The broad nature lacks a focused approach to cybersecurity.	The project specifically addresses this by focusing on cyber anomalies without veering into risk analysis.

Table 2.2 – 1st research paper PROMPT Criteria.

2.4 **Research Study for Detecting Abnormal Behavior Using Deep Learning Techniques**

The second research by Katanosh Morovat; and Brajendra Panda, IEEE, 2020. They examine cybersecurity anomaly detection in depth, highlighting the value of convolutional neural networks (CNN) in this work. They support CNN because of its intrinsic ability to recognize patterns on its own, which is consistent with our goal of quick danger detection.

2.4.1 **PROMPT criteria for the paper:**

- **Presentation:** The research's discussion of CNN's function in anomaly identification is clear and precise.

- **Relevance:** High significance, particularly regarding CNN's usefulness in spotting network anomalies.
- **Objectivity:** Although the research focuses on CNN, it doesn't seem to be biased because it emphasizes the significance of this strategy in the context.
- **Approach:** CNN is used, with a focus on its self-teaching feature, which helps in anomaly identification.
- **Provenance:** Having been published by IEEE, it is a reliable source for technological information.
- **Timeliness:** It was published in 2020, making it quite recent, even though technology changes quickly.

Table 2.2 shows the PROMPT criteria of IEEE paper explaining the similarities, differences and summary for our project and their paper.

Criteria	**IEEE Paper**	**Our Project Report**
Research Summary	Detailed on CNN's role in anomaly detection.	Focused on ML-driven and deep learning-driven anomaly detection.
Similarities and Differences	Both advocate for smart systems in anomaly detection. The paper leans on DL, while our project is ML-based.	Focused approach on ML and deep learning for cybersecurity.
Information Collection and Problem Addressing	Uses CNN for discerning patterns and anomalies.	ML models for network anomaly detection.
Gaps or Shortcomings Addressed by Our Project	Emphasis on DL and CNN might not cater to ML-specific approaches.	The project provides insights into the ML-driven and deep learning-driven approach for network anomaly detection

Table 2.3 – 2nd research paper PROMPT Criteria.

2.5 **Research Study on Explaining Importance of Machine Learning for Security Purposes**

The third one is by Jan Vávra et al., ScienceDirect/MICR1, 2021 and in this paper, the transformative power of technology is highlighted, with a focus on industrial control system cybersecurity. By utilizing machine learning algorithms such as artificial neural networks, recurrent neural networks (LSTM), isolation forests, and the OCSVM algorithm, the study provides a formidable security mechanism. This system's skill in meeting the unique requirements of industrial control systems highlights its applicability to the goals of our project.

2.5.1 **PROMPT Requirements for the paper:**

- **Presentation:** The article explains the nuances of ML in cybersecurity for industrial systems in a thorough and well-organized manner.
- **Relevance:** Very relevant, particularly in the area of cybersecurity machine learning and deep learning.
- **Objectivity:** The report offers an unbiased, comprehensive analysis of several ML methods.
- **Approach:** Utilizes a variety of ML algorithms to combat cyber threats in industrial systems.
- **Provenance:** The work has credibility because it was written by numerous researchers and was published on ScienceDirect.
- **Timeliness:** The work, which was published in 2021, is a current and significant contribution.

Table 2.3 shows the PROMPT criteria of ScienceDirect paper explaining the similarities, differences and summary for our project and their paper.

Criteria	ScienceDirect Paper	Our Project Report
Research Summary	Focuses on ML for industrial control system security.	Emphasizes ML and Deep Learning for network anomaly detection
Similarities and Differences	Both projects employ ML for security, but the paper zeroes in on industrial systems.	Our project is broader in application, addressing network anomalies.
Information Collection and Problem Addressing	Uses a variety of ML algorithms tailored for industrial control systems.	Utilizes ML and deep learning models for general cyber anomalies.
Gaps or Shortcomings Addressed by Our Project	Its role focus on industrial systems might not accommodate to broader cybersecurity needs.	Our project fills this by catering to general cyber threat detection.

Table 2.4 – 3rd research paper PROMPT Criteria.

Chapter 3
PROJECT SPECIFICATION

Chapter 3
Project Specification

This chapter focuses on requirements for the project as functional, non-functional and data in details with explaining the skills required for the project, initial prototype and architecture with some results discussing the accuracy tools for evaluation metrics mentioned above in Chapter 1. Our system can hold 1 types of datasets in which they consist of Anomaly feature, due to our comprehensive system that holds labelled dataset for prediction, training, and testing. The system should adhere to the following functional, non-functional and data requirements:

3.1 Functional Requirements

1. **Data Input Module:**
    - It ensures that the system has access to relevant data for analysis.
    - Being dynamic on entering relevant dataset and able to preprocess it.
2. **Data Preprocessing:**
    - Data preprocessing is a critical step where the system cleans and transforms the raw data to make it suitable for analysis.
    - Tasks may include handling missing values, data normalization, and converting data into a usable format.
3. **Feature Engineering:**
    - Feature engineering involves the identification and extraction of key features (attributes or variables) from the preprocessed data.
    - These features are selected to improve the system's ability to detect anomalies effectively.
4. **Feature Selection:**
    - focuses on identifying and retaining the most relevant features from the dataset to improve the accuracy and efficiency of our machine and deep learning models.

- Through progressive analysis and selection, this process ensures that only the most discriminative attributes are used for anomaly detection, enhancing the system's overall performance in identifying potential cyber threats.

5. **Anomaly Detection Module:**
   - This module is the heart of the system, where machine learning and deep learning models are utilized to detect and classify anomalies or unusual patterns within the data.
   - It distinguishes normal behavior from potential cyber threats and provides indications of suspicious activities.
6. **Performance Evaluation Tools:**
   - To ensure the effectiveness of the project models, the system includes performance evaluation tools.
   - These tools allow the assessment of how well the models are performing in terms of accuracy, precision, recall, F1-score, specificity and other relevant metrics.

3.2 **Non-Functional Requirements**

1. **Reliability:**
   - Reliability A cornerstone of any security system, reliability ensures the system consistently and accurately detects threats, even during periods of network traffic.
   - This means the system can be depended upon to perform its critical function regardless of fluctuations in network activity.
2. **Resource Efficiency:**
   - Resource efficiency involves optimizing the use of computing and memory resources.
   - The system should execute its algorithms in a way that minimizes resource consumption and reduces hardware requirements.

3. **Adaptability:**
   - Regular updates and refinements to algorithms, coupled with continuous monitoring of system performance, are essential to ensure effective threat detection.

These non-functional requirements collectively establish the criteria for the AI-based Cyber Threat Detection system's capabilities, performance, reliability, and adaptability. They provide a comprehensive framework for the design, implementation, and evaluation of the system, ensuring it meets the expected standards of functionality and performance while also enabling efficient maintenance and adaptation to be evolving cyber threats.

3.3 Data Requirements

1. **Data Types:**
   - The system should be capable of handling network logs. These may include logs related to network traffic.
2. **Data Volume:**
   - The system should efficiently manage varying data volumes, as network and system data can vary significantly in size.
3. **Data Quality:**
   - Data quality is crucial. The data should be accurate, complete, and consistent.
   - The system should account for missing or incomplete records.

The system is capable of manipulating and trained to a real dataset from the real-time packets applications and mostly taken from Avast databases which consists of a large datasets of network traffic which can help on defining anomalies and malicious behavior. Additionally, since these datasets used for training are real datasets which can users obtain from applications like, Wireshark, it is not needed to search for diverse datasets with different features because, the dataset used is efficient and scalable with users and entities respectively.

3.4 Software Tools and Hardware Equipment

- **Python:** To create the machine learning and deep learning models, use Python (with libraries like TensorFlow and Scikit-learn). Model construction, training, validation and testing are facilitated by the extensive ecosystem of libraries available in Python.

3.4.1 Software and Platforms

Machine Learning and AI Platforms:

- **Python:** Widely used for machine learning and deep learning with a vast ecosystem of libraries like TensorFlow and Scikit-learn.

3.4.2 Development Environment and Collaborative Tools:

- **Jupyter Notebooks:** For collaborative AI model development and testing.

3.4.3 Hardware Equipment

High-Performance Workstations: For model development and testing. These should have:

- Powerful multi-core processors.
- High RAM capacity (32GB or more).
- SSDs for faster data access.
- GPU acceleration (like NVIDIA's CUDA-enabled GPUs) for deep learning tasks if considered in future phases.

Chapter 4
MODEL DESIGN & ESSENTIAL SKILLS

Chapter 4
Model Design & Essential Skills

4.1 **High-Level Design**

In the high-level structure illustrated in Figure 3.1, The cyber threat detection system functions in three key stages: Data Processing, Machine Learning/Deep Learning Analysis, and Evaluation.

Data Processing prepares raw data for analysis. This involves collecting data from relevant sources (network traffic logs, system logs, etc.), cleaning the data to ensure quality, and potentially selecting or engineering features that enhance threat detection. Non-numerical features like text are converted into a format usable by machine learning algorithms.

The Machine Learning & Deep Learning AI Module analyzes data to detect potential threats. Machine learning algorithms (XGBoost, Logistic Regression, K-Nearest Neighbors, Random Forest, Decision Tree and Gaussian Naive Bayes) are trained on labeled data (already classified as benign or malicious) to learn patterns that differentiate between normal and malicious activity. Deep learning algorithm (GRU), a subfield of machine learning, can be used alongside traditional machine learning for complex threat detection tasks, particularly when dealing with large amounts of unstructured data like network traffic. Both approaches may involve hyperparameter tuning for optimal performance. Finally, the trained models are tested and validated on dataset to assess their effectiveness.

Evaluation: measures the system's performance using metrics like accuracy, precision, recall, F1-score, and specificity. This helps gauge the system's ability to correctly identify threats while minimizing false positives (normal activity classified as threats). Our cyber threat detection system utilizes a flexible design that adapts to the chosen machine learning algorithms, the characteristics of your specific data, and your desired functionalities. This allows for customization to optimize threat detection for your particular needs. Additionally, the system can utilize both traditional machine and deep learning techniques for a more comprehensive approach. This combined approach can be especially beneficial when dealing with complex data types or large datasets.

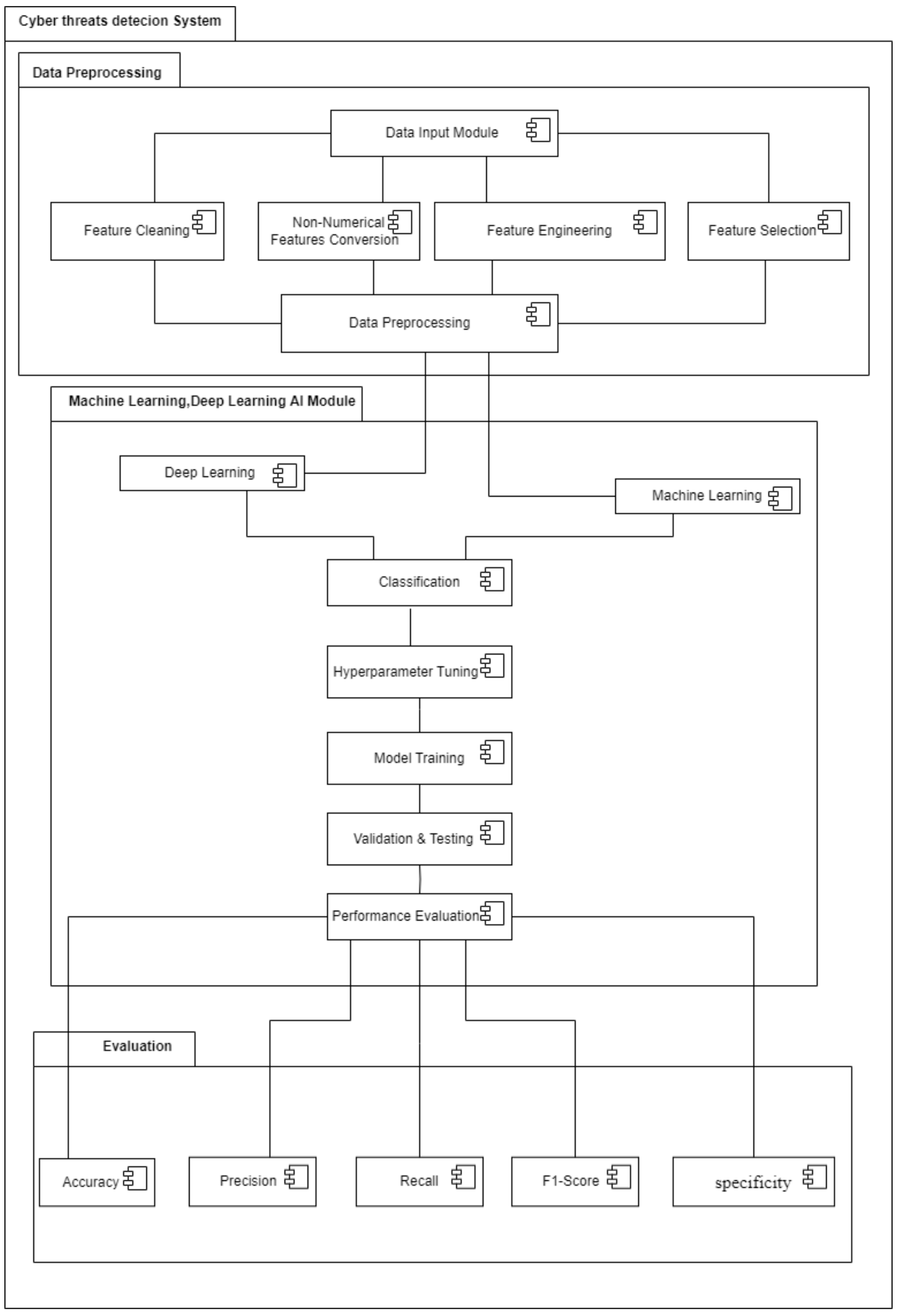


Figure 4.1.1 – CyberThreatDetection High Level Diagram

4.2 Essential Skills for The Project

In the tracing of successfully completing the "Cyber Threat Detection Using Artificial Intelligence" project, our roles as computer science students with a solid foundation in the field are supreme. To ensure our project's successful execution, we have identified key skills that are essential to our success. These skills encompass machine learning and artificial intelligence expertise, Deep Learning, skillful programming in Python, and the ability to perform effective data preprocessing and analysis. The most important skills needed to complete the project are as follows:

4.2.1 Machine Learning and Artificial Intelligence Skills:

- **Reason:** Machine learning and AI are at the core of the project. Developing anomaly detection models requires strong skills in ML and AI, including understanding algorithms, data preprocessing, feature engineering, and model training.
- **Skills to be Developed:** While we have a foundational understanding of ML and AI, we need to further develop our skills in these areas. This includes diving deeper into advanced ML algorithms and deep learning models.
- **How to Tackle:** Enrolling in advanced machine learning and deep learning courses and workshops can enhance our expertise. Participating in hands-on projects and collaborating with professors or experts in the field can also provide valuable insights.

4.2.2 Deep Learning Skills:

- **Reason:** Deep learning plays a crucial role in the project's objectives. Developing effective anomaly detection systems requires a strong foundation in deep learning techniques, including neural network architectures, optimization algorithms, and model evaluation methods.
- **Skills to be Developed:** While we possess a basic understanding of deep learning principles, there is a need to further hone our skills in this domain. This involves delving deeper into advanced neural network architectures,

such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

- **How to Tackle:** Engaging in specialized deep learning courses and workshops can facilitate the enhancement of our expertise in this field. Actively participating in research projects or collaborations with experienced practitioners can offer invaluable practical experience and insights into cutting-edge developments in deep learning technology.

4.2.3 **Programming Skills in Python:**

- **Reason:** Python is the primary programming language for developing machine learning models. Proficiency in Python is essential for data preprocessing, model building, and system implementation.
- **Skills to be Developed:** We already have a basic understanding of Python, but to excel in this project, we need to enhance our Python skills. This includes mastering Python libraries like NumPy, Pandas, Scikit-learn, and TensorFlow.
- **How to Tackle:** We plan to take advanced Python courses that focus on data science and machine learning. Additionally, we'll dedicate time to hands-on coding, work on real-world projects, and participate in coding competitions.

4.2.4 **Data Preprocessing and Analysis**:

- **Reason:** Data preprocessing is a critical step in ensuring data quality for model training. Skills in data cleaning, feature engineering, and data analysis are essential.
- **Skills to be Developed:** While the familiarity with data preprocessing concepts achieved, the aim is to become capable in this area. Being as skilled as possible in data cleaning, feature engineering, feature selection and data analysis.
- **How to Tackle:** Our plan is to practice data preprocessing and analysis on real datasets with the guidance from mentors and online resources.

Chapter 5
DATA PREPROCESSING

Chapter 5
Data Preprocessing

5.1 Data Collection

Stratosphere Laboratory is a labeled dataset from Avast databases with malicious and benign IoT network traffic. The system was built to utilize network monitoring tools, such as Wireshark as a future work, to capture real-time network traffic in diverse environments. Applying the similarities between the datasets provided and the displayed data in Wireshark to serve as the foundation for training and evaluating the model, empowering the model to detect and mitigate potential threats effectively.

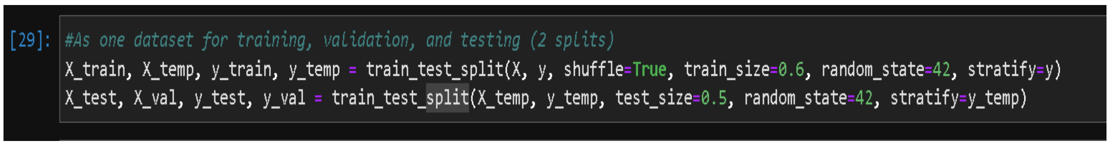
5.2 Data Preprocessing

After specifying the dataset and its features, begun the preprocessing process; a python file (**preprocessing_data.py**) has taking charge and outputs a cleaned and consistent format of the original dataset as much as possible to work as the model input. This process involves the following:

- **Data Transforming:** The code performs various data transformations such as converting categorical variables to numerical codes, converting data types, handling missing values.
- **Data Cleaning:** The code replaces missing or erroneous values with appropriate replacements, such as converting '-' to 0 or -1, rounding floating-point values, and converting timestamps to integers to ensure data quality and consistency.
- **Data Formatting:** The code creates tables to map original categorical values to their corresponding numerical codes.
- **Label Encoding:** The code encodes categorical labels into numerical codes 0 or 1. This is a typical preprocessing step, especially for classification tasks, to transform labels into a format suitable for training machine learning models.
- **Data Splitting:** Although this process is not included in **preprocessing_data.py** file, splitting the dataset into training, validation, and testing sets, the dataset has been partitioned into three subsets: a training

dataset comprising 60% of the total dataset, and two separate validation and testing datasets, each containing 20% of the total remaining 40% of the entire dataset. This partitioning ensures a balanced allocation of samples for model training, hyperparameter tuning, and final performance evaluation.

Figure 5.2.1 shows the code for data splitting where the parameters are defined to make the split more reliable and efficient by using stratify and shuffle which makes the splitted data more applicable for training to give better results and generalization.



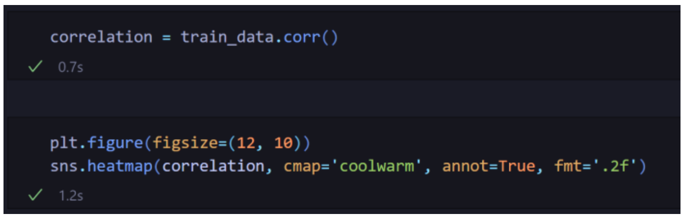
```
[29]: #As one dataset for training, validation, and testing (2 splits)
      X_train, X_temp, y_train, y_temp = train_test_split(X, y, shuffle=True, train_size=0.6, random_state=42, stratify=y)
      X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)
```

Figure 5.2.1 – Using train_test_split() method to split the dataset.

5.3 Feature Selection

Once the problem to be addressed was explicitly defined and the objectives of the machine learning model were understood, a subset of related features from the initial feature set had to be selected. This selection aimed to enhance model efficiency, reduce dimensionality, and augment interpretability which can help later on when using exhaustive functions like grid search or Random Forest on train faster and easier. This procedure encompasses the following steps:

- **Data Exploration and Analysis:** Just exploring the dataset to understand the distribution of features, identify missing values, and detect outliers.
- **Feature Importance Ranking:** Utilization of techniques such as correlation analysis and feature importance scores to overview their relevance to the target variable and identified features that have the strongest relationship with the target variable and are most likely to contribute to model performance using the **corr()** method from **pandas** as shown below Figure 5.2:



```
correlation = train_data.corr()
✓ 0.7s

plt.figure(figsize=(12, 10))
sns.heatmap(correlation, cmap='coolwarm', annot=True, fmt='.2f')
✓ 1.2s
```

Figure 5.3.1 – Correlation code from train_data which is the preprocessed data with only dropping critical features.

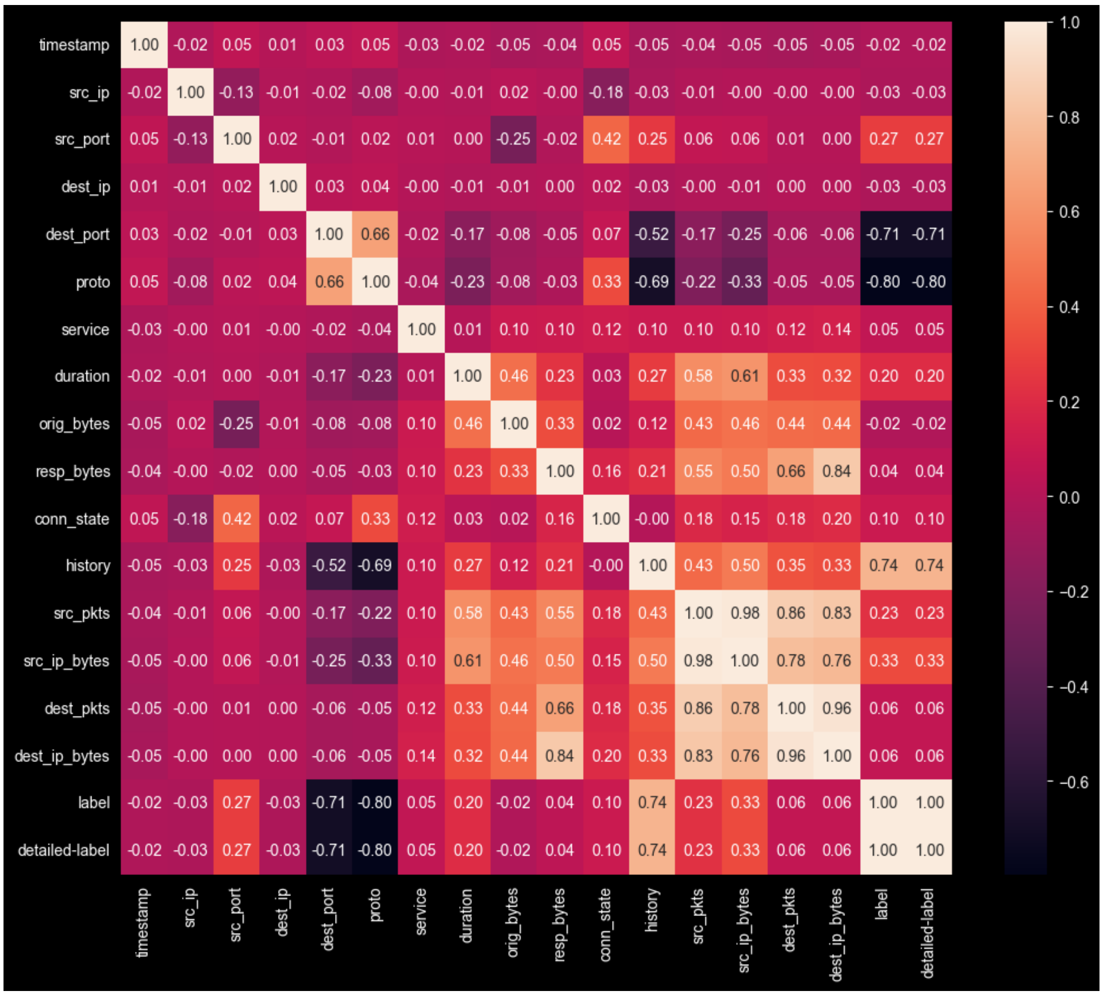


Figure 5.3.2 – Correlation chart of the dataset's features.

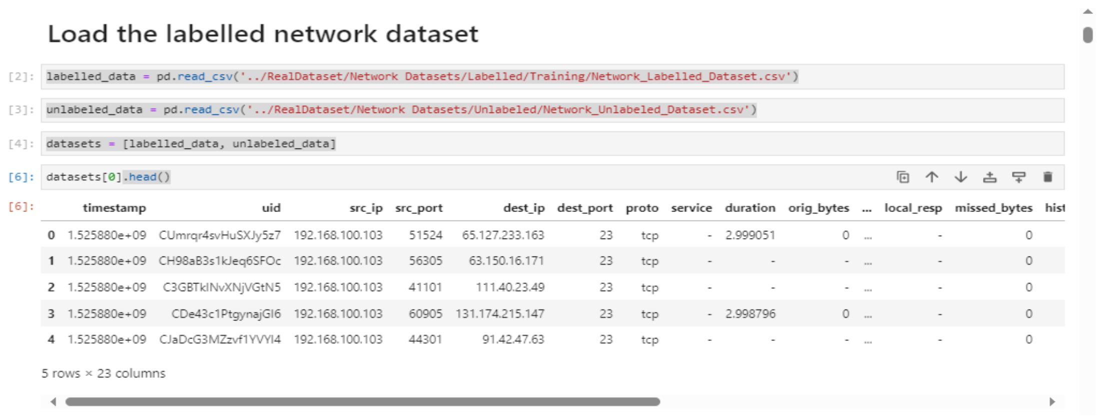
The heatmap generated using the corr() function offers several distinct advantages in data analysis and visualization. Firstly, it provides a clear and intuitive representation of the correlation between different variables in a dataset. By assigning colors to different correlation values, the heatmap allows for quick identification of strong positive or negative correlations, as well as variables that are uncorrelated. This visual aid is particularly helpful in identifying patterns and relationships within complex datasets, enabling data scientists and analysts to gain deeper insights into the underlying data structure. Additionally, the heatmap helps in detecting multicollinearity, where variables are highly correlated with each other, aiding in feature selection and model building processes. Moreover, the heatmap serves as a valuable tool for communication and presentation of findings, making it easier for stakeholders to grasp the relationships between variables and make informed decisions based on data-driven insights.

5.4 Data Preprocessing Implementation

In this section, the focus will be on the ongoing progress for the implementation of the system, pointing into the steps taken as discussed in previous section with more detailed process regarding the actual code implementation with screenshots for the essential phases developed for the future or upcoming activities which divided into various sections as below:

5.4.1 Data Filtering and Modification:

In this step, the system carefully filters the dataset to align it with the specific requirements of the project. Needless columns were identified and removed, reforming the dataset for optimal efficiency. Importantly, in the previous section "Methods and Approaches" there was a file named "preprocessing_data.py" for preprocessing phase which will be explained in detail with step by step as if the code was in one file just for simplicity for now. As shown in figure 5.4.1, the dataset is loaded and divided into two parts, labelled_data and unlabeled_data. The primary distinction lies in the presence of the "label" column in labelled_data, which signifies whether a packet is categorized as healthy (Benign) or harmful (Malicious). A value of 1 indicates a malicious packet, while a value of 0 indicates a benign packet. labelled_data is crucial for the educational process, enabling the model to learn to differentiate between harmful and benign packets. Afterwards, the model is presented with the unlabeled_data, which lacks the "label" column. This allows us to assess the model's learning validity, measure expectations, accuracy, and evaluate the extent of its proficiency.


Load the labelled network dataset

```
[2]: labelled_data = pd.read_csv('../RealDataset/Network Datasets/Labelled/Training/Network_Labelled_Dataset.csv')
[3]: unlabeled_data = pd.read_csv('../RealDataset/Network Datasets/Unlabeled/Network_Unlabeled_Dataset.csv')
[4]: datasets = [labelled_data, unlabeled_data]
[6]: datasets[0].head()
```

[6]:

	timestamp	uid	src_ip	src_port	dest_ip	dest_port	proto	service	duration	orig_bytes	...	local_resp	missed_bytes	hist
0	1.525880e+09	CUmrqr4svHuSXJy5z7	192.168.100.103	51524	65.127.233.163	23	tcp	-	2.999051	0	...	-	0	
1	1.525880e+09	CH98aB3s1kJeq6SFOc	192.168.100.103	56305	63.150.16.171	23	tcp	-	-	-	...	-	0	
2	1.525880e+09	C3GBTkINvXNjVGtN5	192.168.100.103	41101	111.40.23.49	23	tcp	-	-	-	...	-	0	
3	1.525880e+09	CDe43c1PtgynajGI6	192.168.100.103	60905	131.174.215.147	23	tcp	-	2.998796	0	...	-	0	
4	1.525880e+09	CJaDcG3MZzvf1YVYI4	192.168.100.103	44301	91.42.47.63	23	tcp	-	-	-	...	-	0	

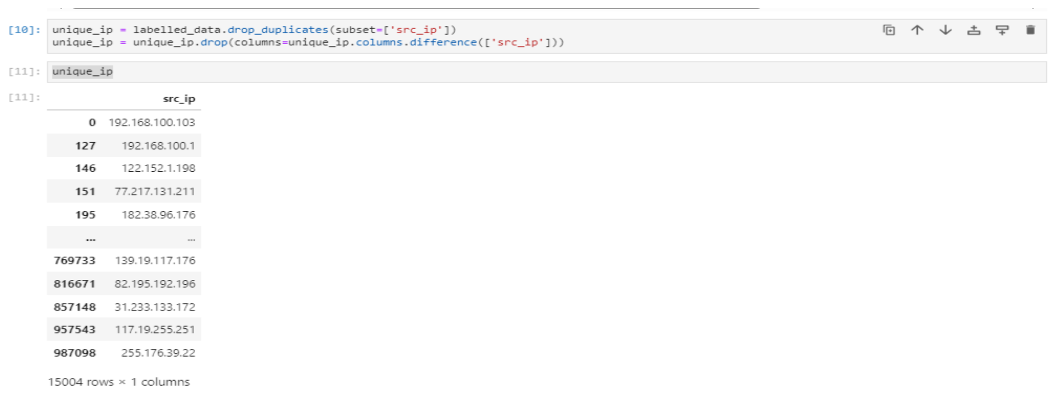
5 rows × 23 columns

Figure 5.4.1 – Load the labelled and unlabeled dataset.

Noteworthy, some columns or features *as shown in figure 12* were empty or having one type of record. For example, the feature ‘local_resp’ consists of nothing except ‘-‘ value which represents null or NA value. Also, for the feature ‘uid’ there are only 100% different values since, it is a unique id for each packet, hence giving for each row or packet a unique id which can mislead the ai in training and optimizing the model. The code showing for each of these features which meant to be dropped to remove from the dataset the counts their corresponding unique or distinct values or records given for each feature. And as given, all except ‘uid’ has one value in all rows.

5.4.2 **Handling Data Types:**

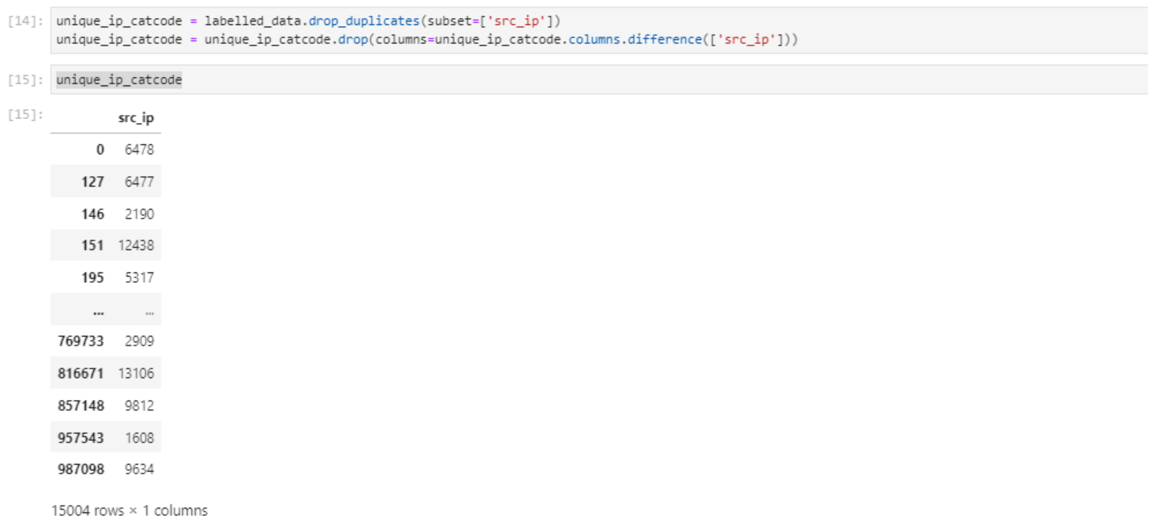
A significant aspect of our preprocessing involved the careful handling of data types where some columns required conversion to suitable data type which in our case for a classification models an integer data type to facilitate modeling and manipulation. In the current step as shown in the figure 5.4.2, IP Address is mainly a string data type which cannot be represented in most of classification models’ format thus, the system identified and retained unique IP addresses by creating a new DataFrame, 'unique_ip,' through the removal of duplicate rows based on the 'src_ip' column from the original 'labelled_data' DataFrame. Subsequently, to streamline the DataFrame and focus uniquely on IP addresses, further steps taken for refining 'unique_ip' by dropping columns other than 'src_ip.' This process ensures that our analysis concentrates on distinct IP addresses, eliminating redundancy and facilitating a more targeted exploration of the dataset with the importance of saving the real data form to come back to it in postprocessing steps in the upcoming progress.



```
[10]: unique_ip = labelled_data.drop_duplicates(subset=['src_ip'])
      unique_ip = unique_ip.drop(columns=unique_ip.columns.difference(['src_ip']))

[11]: unique_ip

[11]:
                 src_ip
      0     192.168.100.103
      127     192.168.100.1
      146     122.152.1.198
      151    77.217.131.211
      195     182.38.96.176
      ...               ...
      769733  139.19.117.176
      816671  82.195.192.196
      857148  31.233.133.172
      957543  117.19.255.251
      987098   255.176.39.22

      15004 rows × 1 columns
```

Figure 5.4.2 – Eliminating duplicate rows based on the 'src_ip'.

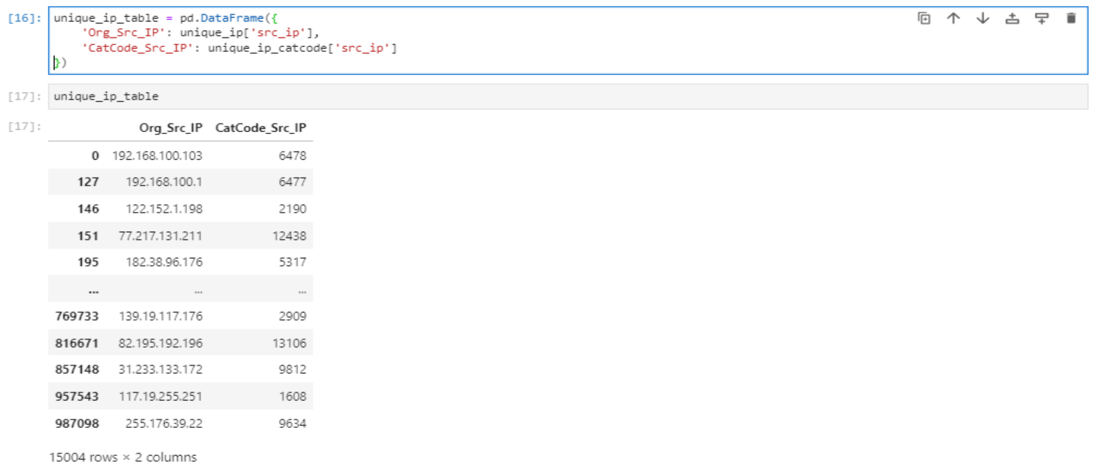
As the next step in figure 5.4.3, a new dataframe, 'unique_ip_catcode,' is initialized by eliminating duplicate rows based on the 'src_ip' column from the 'labelled_data' DataFrame using drop_duplicates(). Further refinement involves discarding unnecessary columns, leaving only the 'src_ip' column in 'unique_ip_catcode.' This strategic process results in a dataset exclusively dedicated to distinct IP addresses, accompanied by their categorical codes, providing a streamlined and focused dataset composed for advanced analysis and modelling.



```
[14]: unique_ip_catcode = labelled_data.drop_duplicates(subset=['src_ip'])
      unique_ip_catcode = unique_ip_catcode.drop(columns=unique_ip_catcode.columns.difference(['src_ip']))

[15]: unique_ip_catcode

[15]:
         src_ip
      0    6478
    127    6477
    146    2190
    151   12438
    195    5317
    ...     ...
 769733    2909
 816671   13106
 857148    9812
 957543    1608
 987098    9634

15004 rows × 1 columns
```

Figure 5.4.3 – Unique table for the ip to be able to reserve and save the original values.

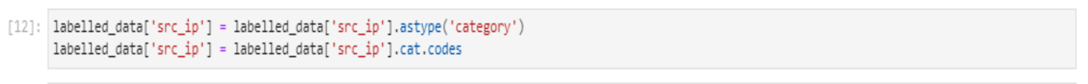
In addition to these 2 steps, combining these 2 dataframes will make a unique table identifying each cat code correspondent with the original value of the IP Address. As showing figure 5.4.4:



```
[16]: unique_ip_table = pd.DataFrame({
          'Org_Src_IP': unique_ip['src_ip'],
          'CatCode_Src_IP': unique_ip_catcode['src_ip']
      })

[17]: unique_ip_table

[17]:
             Org_Src_IP  CatCode_Src_IP
      0  192.168.100.103            6478
    127    192.168.100.1            6477
    146    122.152.1.198            2190
    151   77.217.131.211           12438
    195    182.38.96.176            5317
    ...              ...             ...
 769733   139.19.117.176            2909
 816671   82.195.192.196           13106
 857148   31.233.133.172            9812
 957543   117.19.255.251            1608
 987098    255.176.39.22            9634

15004 rows × 2 columns
```

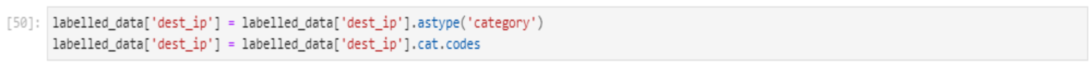
Figure 5.4.4 – Replace original value of ('src_ip', 'dest_ip', ‘proto’, ‘service’, ‘conn_state’, ‘history’) columns with corresponding cat codes.

It is important to note that this approach is used in different features such as Src_Ip, Dest_ip, protocol, service, history, conn state, and detailed label. These lines convert the 'src_ip', 'dest_ip', ‘proto’, ‘service’, ‘conn_state’, ‘history’ columns in the 'labelled_data' DataFrame into categorical data types and then replace the original value with their corresponding cat codes, facilitating more efficient processing and analysis of the data. As shown in figure 5.4.5:



```
[12]: labelled_data['src_ip'] = labelled_data['src_ip'].astype('category')
      labelled_data['src_ip'] = labelled_data['src_ip'].cat.codes
```

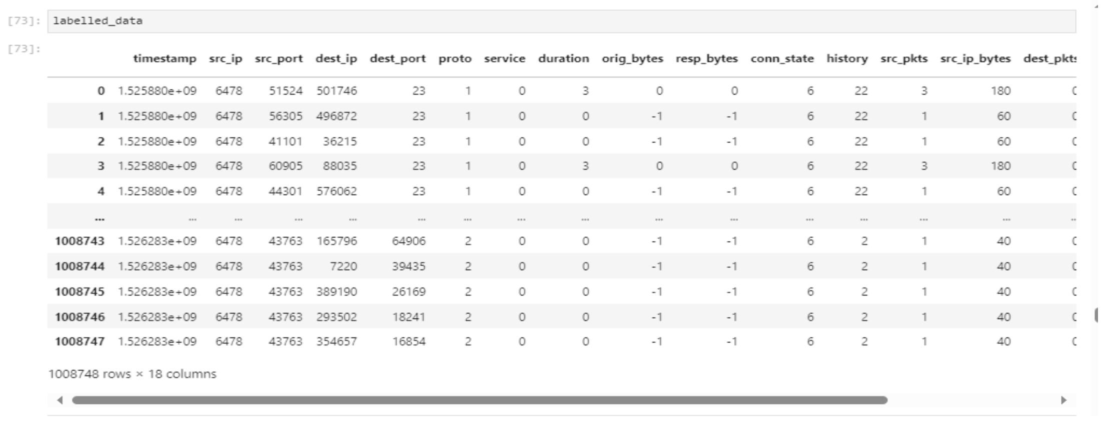
Figure 5.4.5 – Replace original value of ('src_ip') column with corresponding cat codes.



```
[50]: labelled_data['dest_ip'] = labelled_data['dest_ip'].astype('category')
      labelled_data['dest_ip'] = labelled_data['dest_ip'].cat.codes
```

Figure 5.4.6 – Replace original value of ('dest_ip') column with corresponding cat codes.

Figure 5.4.7 shows the output of the labelled_data:



```
[73]: labelled_data
```

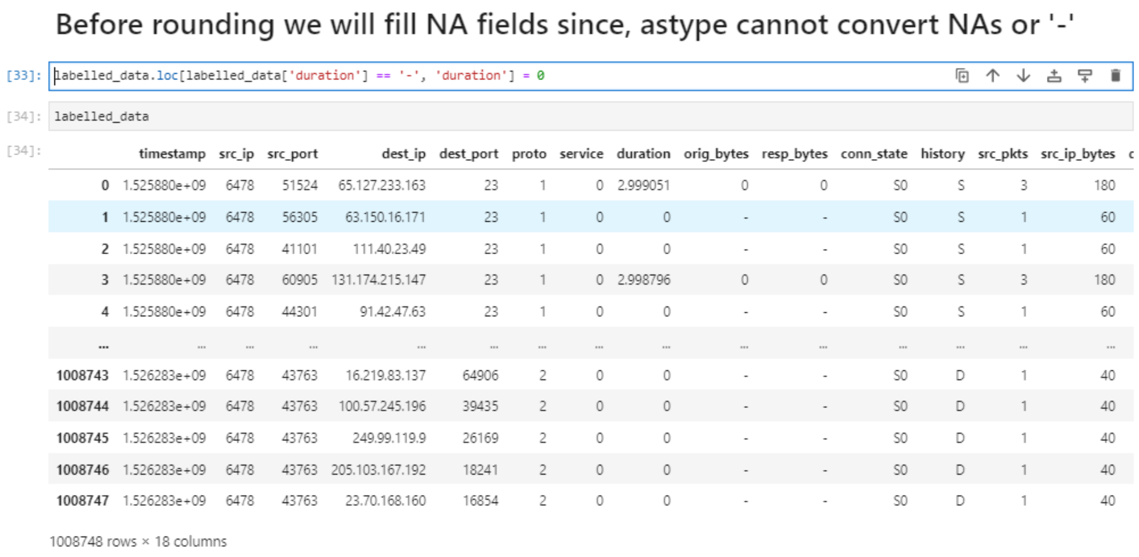
[73]:

	timestamp	src_ip	src_port	dest_ip	dest_port	proto	service	duration	orig_bytes	resp_bytes	conn_state	history	src_pkts	src_ip_bytes	dest_pkts
0	1.525880e+09	6478	51524	501746	23	1	0	3	0	0	6	22	3	180	0
1	1.525880e+09	6478	56305	496872	23	1	0	0	-1	-1	6	22	1	60	0
2	1.525880e+09	6478	41101	36215	23	1	0	0	-1	-1	6	22	1	60	0
3	1.525880e+09	6478	60905	88035	23	1	0	3	0	0	6	22	3	180	0
4	1.525880e+09	6478	44301	576062	23	1	0	0	-1	-1	6	22	1	60	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1008743	1.526283e+09	6478	43763	165796	64906	2	0	0	-1	-1	6	2	1	40	0
1008744	1.526283e+09	6478	43763	7220	39435	2	0	0	-1	-1	6	2	1	40	0
1008745	1.526283e+09	6478	43763	389190	26169	2	0	0	-1	-1	6	2	1	40	0
1008746	1.526283e+09	6478	43763	293502	18241	2	0	0	-1	-1	6	2	1	40	0
1008747	1.526283e+09	6478	43763	354657	16854	2	0	0	-1	-1	6	2	1	40	0

1008748 rows × 18 columns

Figure 5.4.7 – Display labelled_data.

Afterward, the dataset still not ready for modelling part since there would be some null or NA data in some of the features as shown in figure 5.4.8. The system handles missing or undefined values in the 'duration' column of the 'labelled_data' DataFrame. Specifically, it locates rows where 'duration' is marked as '-', indicating missing data, and replaces those instances with a value of 0. This operation ensures that the dataset is appropriately prepared for successive analysis, modifying potential issues associated with missing duration values.



Before rounding we will fill NA fields since, astype cannot convert NAs or '-'

```
[33]: labelled_data.loc[labelled_data['duration'] == '-', 'duration'] = 0
```

```
[34]: labelled_data
```

[34]:

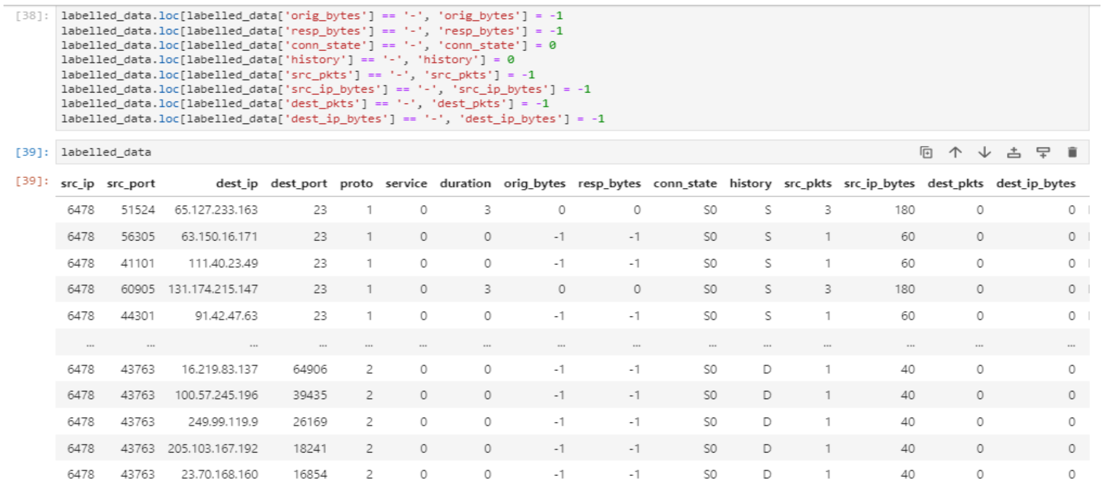
	timestamp	src_ip	src_port	dest_ip	dest_port	proto	service	duration	orig_bytes	resp_bytes	conn_state	history	src_pkts	src_ip_bytes	c
0	1.525880e+09	6478	51524	65.127.233.163	23	1	0	2.999051	0	0	S0	S	3	180	
1	1.525880e+09	6478	56305	63.150.16.171	23	1	0	0	-	-	S0	S	1	60	
2	1.525880e+09	6478	41101	111.40.23.49	23	1	0	0	-	-	S0	S	1	60	
3	1.525880e+09	6478	60905	131.174.215.147	23	1	0	2.998796	0	0	S0	S	3	180	
4	1.525880e+09	6478	44301	91.42.47.63	23	1	0	0	-	-	S0	S	1	60	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1008743	1.526283e+09	6478	43763	16.219.83.137	64906	2	0	0	-	-	S0	D	1	40	
1008744	1.526283e+09	6478	43763	100.57.245.196	39435	2	0	0	-	-	S0	D	1	40	
1008745	1.526283e+09	6478	43763	249.99.119.9	26169	2	0	0	-	-	S0	D	1	40	
1008746	1.526283e+09	6478	43763	205.103.167.192	18241	2	0	0	-	-	S0	D	1	40	
1008747	1.526283e+09	6478	43763	23.70.168.160	16854	2	0	0	-	-	S0	D	1	40	

1008748 rows × 18 columns

Figure 5.4.8 – Change the value (-) in the duration column to zero.

5.4.3 **Dealing with Missing Values:**

This code snip in figure 5.4.9, addresses missing or undefined values in various columns of the 'labelled_data' DataFrame related to network connection attributes. For columns like 'orig_bytes,' 'resp_bytes,' 'src_pkts,' 'src_ip_bytes,' 'dest_pkts,' and 'dest_ip_bytes,' instances marked as '-' are replaced with -1, indicating the absence of data. Additionally, for categorical columns such as 'conn_state' and 'history,' '-' values are substituted with 0 to standardize the representation of missing or undefined data across different types of attributes. This systematic handling of missing values ensures a consistent and coherent dataset for subsequent analysis and model development.



```
[38]: labelled_data.loc[labelled_data['orig_bytes'] == '-', 'orig_bytes'] = -1
      labelled_data.loc[labelled_data['resp_bytes'] == '-', 'resp_bytes'] = -1
      labelled_data.loc[labelled_data['conn_state'] == '-', 'conn_state'] = 0
      labelled_data.loc[labelled_data['history'] == '-', 'history'] = 0
      labelled_data.loc[labelled_data['src_pkts'] == '-', 'src_pkts'] = -1
      labelled_data.loc[labelled_data['src_ip_bytes'] == '-', 'src_ip_bytes'] = -1
      labelled_data.loc[labelled_data['dest_pkts'] == '-', 'dest_pkts'] = -1
      labelled_data.loc[labelled_data['dest_ip_bytes'] == '-', 'dest_ip_bytes'] = -1
```

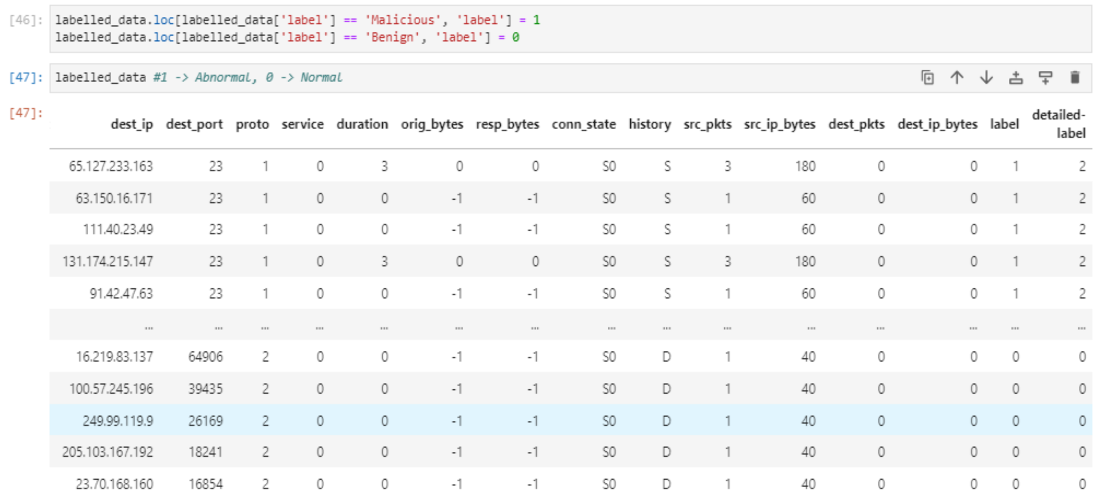
```
[39]: labelled_data
```

[39]:

src_ip	src_port	dest_ip	dest_port	proto	service	duration	orig_bytes	resp_bytes	conn_state	history	src_pkts	src_ip_bytes	dest_pkts	dest_ip_bytes
6478	51524	65.127.233.163	23	1	0	3	0	0	S0	S	3	180	0	0
6478	56305	63.150.16.171	23	1	0	0	-1	-1	S0	S	1	60	0	0
6478	41101	111.40.23.49	23	1	0	0	-1	-1	S0	S	1	60	0	0
6478	60905	131.174.215.147	23	1	0	3	0	0	S0	S	3	180	0	0
6478	44301	91.42.47.63	23	1	0	0	-1	-1	S0	S	1	60	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6478	43763	16.219.83.137	64906	2	0	0	-1	-1	S0	D	1	40	0	0
6478	43763	100.57.245.196	39435	2	0	0	-1	-1	S0	D	1	40	0	0
6478	43763	249.99.119.9	26169	2	0	0	-1	-1	S0	D	1	40	0	0
6478	43763	205.103.167.192	18241	2	0	0	-1	-1	S0	D	1	40	0	0
6478	43763	23.70.168.160	16854	2	0	0	-1	-1	S0	D	1	40	0	0

Figure 5.4.9 – Handling missing values.

Finally, converting the label feature into binary form as 0 for normal or benign state and 1 for abnormal or malicious state. The code shown in figure 5.4.10, transforms categorical labels in the 'labelled_data' DataFrame, replacing 'Malicious' with 1 and 'Benign' with 0. This binary encoding simplifies the classification task, assigning 1 for malicious instances and 0 for benign ones. The comment clarifies the mapping, indicating 1 as 'Abnormal' and 0 as 'Normal' for better interpretation.



```
[46]: labelled_data.loc[labelled_data['label'] == 'Malicious', 'label'] = 1
      labelled_data.loc[labelled_data['label'] == 'Benign', 'label'] = 0
```

```
[47]: labelled_data #1 -> Abnormal, 0 -> Normal
```

[47]:

	dest_ip	dest_port	proto	service	duration	orig_bytes	resp_bytes	conn_state	history	src_pkts	src_ip_bytes	dest_pkts	dest_ip_bytes	label	detailed-label
	65.127.233.163	23	1	0	3	0	0	S0	S	3	180	0	0	1	2
	63.150.16.171	23	1	0	0	-1	-1	S0	S	1	60	0	0	1	2
	111.40.23.49	23	1	0	0	-1	-1	S0	S	1	60	0	0	1	2
	131.174.215.147	23	1	0	3	0	0	S0	S	3	180	0	0	1	2
	91.42.47.63	23	1	0	0	-1	-1	S0	S	1	60	0	0	1	2
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	16.219.83.137	64906	2	0	0	-1	-1	S0	D	1	40	0	0	0	0
	100.57.245.196	39435	2	0	0	-1	-1	S0	D	1	40	0	0	0	0
	249.99.119.9	26169	2	0	0	-1	-1	S0	D	1	40	0	0	0	0
	205.103.167.192	18241	2	0	0	-1	-1	S0	D	1	40	0	0	0	0
	23.70.168.160	16854	2	0	0	-1	-1	S0	D	1	40	0	0	0	0

Figure 5.4.10 – Change the label column Malicious to 1 and the benign to zero.

Chapter 6
PROJECT IMPLEMENTATION

Chapter 6
Project Implementation

This comprehensive research provides an understanding of our ongoing work to improve our capabilities to detect network activity anomalies, aiming to refine our methodologies and approaches by applying knowledge from previous activities, resulting in better performance, and implementing a consistent system for detecting anomalies. In the implementation phase, this research will go through the process of creating the project starting from data collection until coding the essential parts of the system which resulting in various steps and processes like methods and approaches, data preprocessing, modelling, model training, where all of the steps will be explained in detail.

6.1 Multi-Algorithm Approach

Comprehensive Algorithm Exploration: We employed a diverse set of algorithms, including XGBoost, Logistic Regression, K-Nearest Neighbors, Random Forest, Decision Tree, Gaussian Naive Bayes, and GRU. This comprehensive exploration allows us to identify the most effective approach for accurate and reliable threat detection.

Maximizing Detection Efficacy: Each algorithm has unique strengths, allowing us to thoroughly assess their performance in threat detection. By selecting the best performing algorithm, we ensure the system's overall efficacy in detecting threats with high precision and reliability.

Strategic Investment for the Future: Evaluating multiple algorithms is not only beneficial for the current project but also serves as a valuable resource for future implementations. Our findings empower individuals to replicate, extend, or directly apply them in real-world cyber threat detection projects. This comprehensive algorithm evaluation equips them to choose the most suitable approach for their specific needs.

Fostering Innovation in Cybersecurity: By sharing our comparative analysis, we contribute to the broader understanding of machine learning and AI methodologies in cybersecurity. This knowledge exchange fosters further innovation and development of advanced threat detection techniques for a more secure digital landscape.

6.2 **Model Selection**

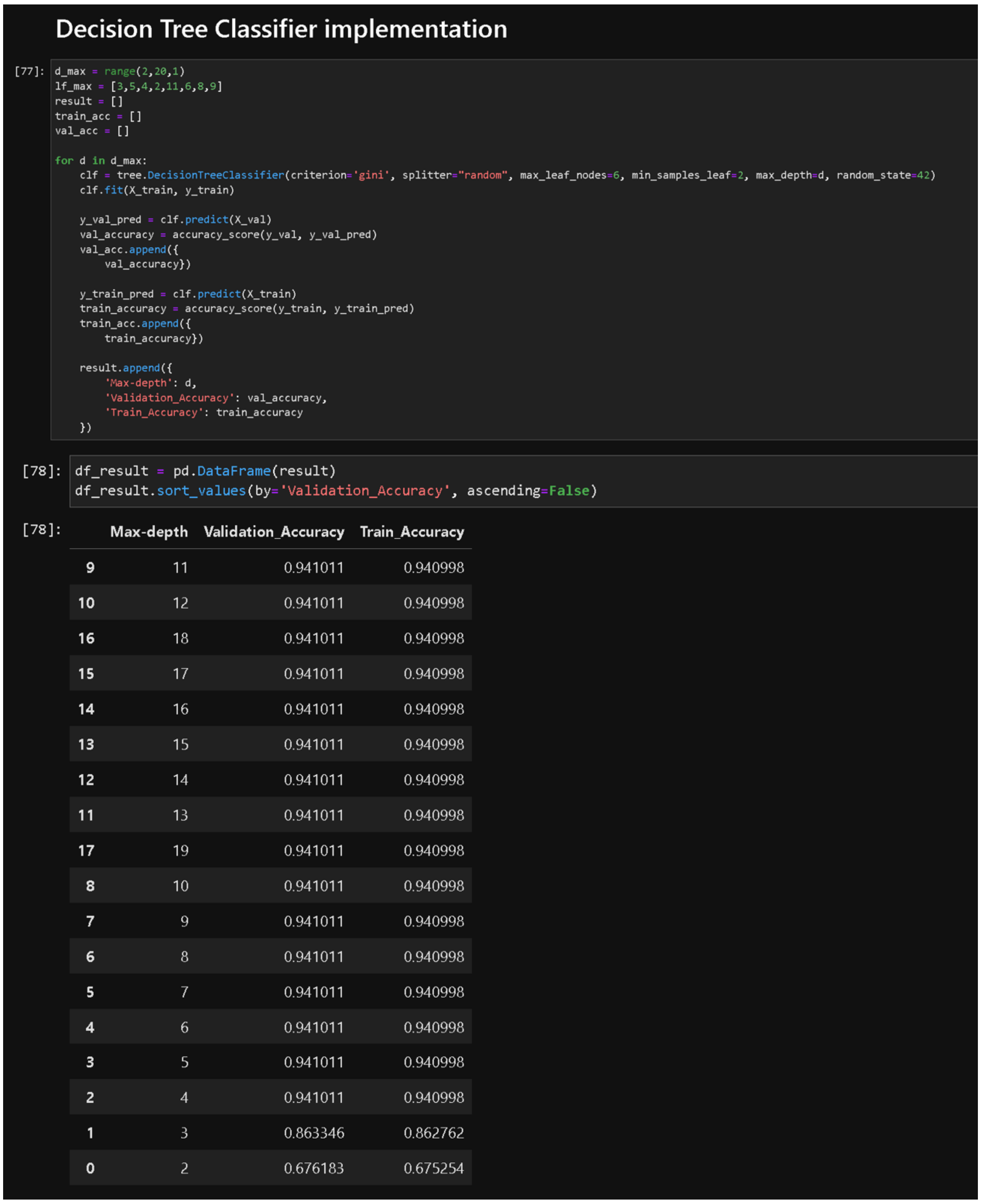
Acquiring knowledge about Decision Tree Classification during the CPCS-331 (Artificial Intelligence) course prompted consideration of initiating modeling with this algorithm. Given the modest familiarity with this algorithm and the availability of the Decision Tree Classifier within the machine learning library ***sklearn*** under the ***tree*** class, it appeared to be a reasonable starting point. However, exploration of alternative algorithms that may better serve the objectives of the model remains open as progress continues. Initially, determining the best hyperparameters, which play a crucial role in the training process of machine learning algorithms, posed a challenge. The two known methods found, Random Search and Grid Search, proved to be too exhaustive for the existing hardware and consumed excessive time. Fortunately, in this research grid search is applied and implemented in decision tree to represent the quality grid search gives. It is important to note that, the implementation will be discussed later on this chapter.

As further exploration and learning about the hyperparameters of the Decision Tree Classifier unfolded, many sources indicated that two hyperparameters are considered the most important in regular training situations:

1. ***max_leaf_nodes*** (integer, default=None):
    - This hyperparameter controls the maximum number of leaf nodes to be allowed in the decision tree.
    - When set to an integer value, the tree is grown in a best-first fashion, where the best nodes are selected based on the relative reduction in impurity.
    - If set to **None**, the tree can grow without any restriction on the number of leaf nodes, potentially leading to a more complex model.

2. ***max_depth*** (integer, default=None):
    - This hyperparameter specifies the maximum depth of the decision tree.
    - When set to an integer value, the tree nodes are expanded until reaching the specified maximum depth or until all leaf nodes become pure (contain only samples of one class) or contain fewer samples than the min_samples_split parameter.
    - If set to None, the tree nodes are expanded until all leaves are pure or until all leaves contain fewer samples than the min_samples_split parameter.

So, to determine a good initial hyperparameters the system will implement a very basic code that require ***for_loop*** to go through some predetermined values stored in ***d_max*** array and fixed the value of ***max_leaf_nodes*** = 6 as it is figured to be the most suitable value for most of the cases, and as the loop goes through all ***d_max*** entries the **validate accuracy** and the **training accuracy** automatically, then appended to the ***result*** array to be displayed at last with ***pd.DataFrame(result)*** method as a table. Figure 6.2.1 shows that at the top of the table, the best max_depth in this case is a tree with a depth of 11.



Decision Tree Classifier implementation

```
[77]: d_max = range(2,20,1)
      lf_max = [3,5,4,2,11,6,8,9]
      result = []
      train_acc = []
      val_acc = []

      for d in d_max:
          clf = tree.DecisionTreeClassifier(criterion='gini', splitter="random", max_leaf_nodes=6, min_samples_leaf=2, max_depth=d, random_state=42)
          clf.fit(X_train, y_train)

          y_val_pred = clf.predict(X_val)
          val_accuracy = accuracy_score(y_val, y_val_pred)
          val_acc.append({
              val_accuracy})

          y_train_pred = clf.predict(X_train)
          train_accuracy = accuracy_score(y_train, y_train_pred)
          train_acc.append({
              train_accuracy})

          result.append({
              'Max-depth': d,
              'Validation_Accuracy': val_accuracy,
              'Train_Accuracy': train_accuracy
          })
```

```
[78]: df_result = pd.DataFrame(result)
      df_result.sort_values(by='Validation_Accuracy', ascending=False)
```

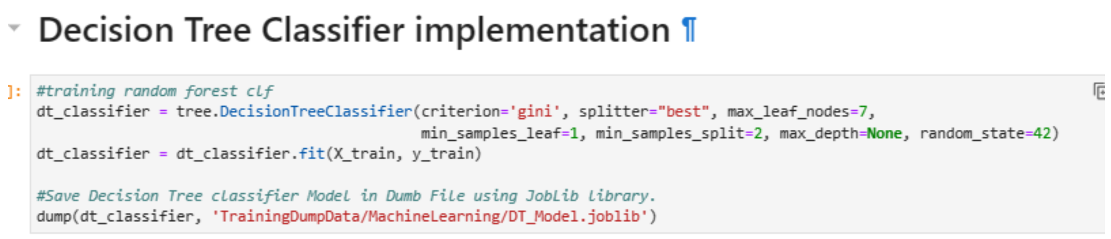
[78]:

	Max-depth	Validation_Accuracy	Train_Accuracy
9	11	0.941011	0.940998
10	12	0.941011	0.940998
16	18	0.941011	0.940998
15	17	0.941011	0.940998
14	16	0.941011	0.940998
13	15	0.941011	0.940998
12	14	0.941011	0.940998
11	13	0.941011	0.940998
17	19	0.941011	0.940998
8	10	0.941011	0.940998
7	9	0.941011	0.940998
6	8	0.941011	0.940998
5	7	0.941011	0.940998
4	6	0.941011	0.940998
3	5	0.941011	0.940998
2	4	0.941011	0.940998
1	3	0.863346	0.862762
0	2	0.676183	0.675254

Figure 6.2.1 - Training and Testing of a dummy parameters with Decision Tree Classifier algorithm.

6.3 Model Training

After the hyperparameters has been set up for the Decision Tree Classifier model and preprocessing the project's dataset, comes the training step where all previous processes and modifications will be used to provide a proper training to the model as much as possible. Calling ***tree.DecisionTreeClassifier()*** method with the hyperparameters that has been set up, then training the model with ***.fit(X_train , y_train)*** method. Figure 6.3.1 shows the code for the training and fitting of the decision tree model using the hyper-parameters used in the random search.



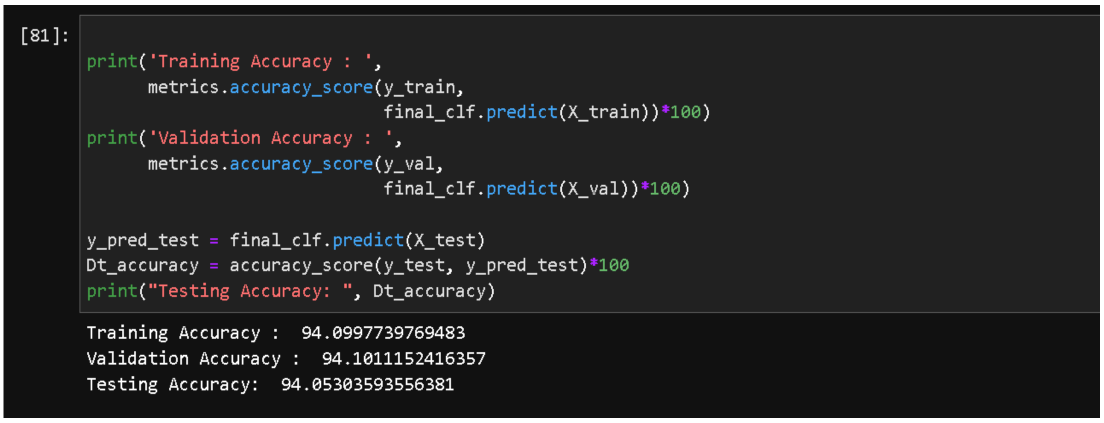
Decision Tree Classifier implementation ¶

```
]: #training random forest clf
dt_classifier = tree.DecisionTreeClassifier(criterion='gini', splitter="best", max_leaf_nodes=7,
                                           min_samples_leaf=1, min_samples_split=2, max_depth=None, random_state=42)
dt_classifier = dt_classifier.fit(X_train, y_train)

#Save Decision Tree classifier Model in Dumb File using Joblib library.
dump(dt_classifier, 'TrainingDumpData/MachineLearning/DT_Model.joblib')
```

Figure 6.3.1 – Training the Decision Tree Classifier model after setting up the hyperparameters and then Saving a Trained Model using Joblib.

6.4 **Model Evaluation**

As mentioned in the preprocessing step, the dataset has been split to a training dataset comprising 60% of the total dataset, and two separate validation and testing datasets, each containing 20% of the total remaining 40% of the entire dataset, with that partitioning in mind the validation, training, and testing accuracies are calculated with ***metrics.accuracy()*** built-in method, and these are the results shown below Figure 6.4.1:



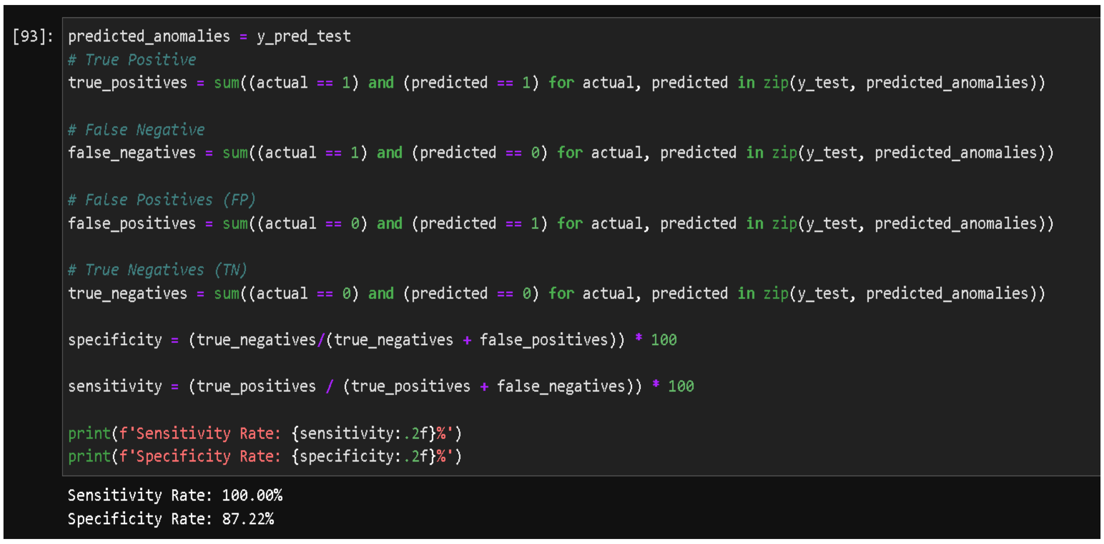
```
[81]:
print('Training Accuracy : ',
      metrics.accuracy_score(y_train,
                             final_clf.predict(X_train))*100)
print('Validation Accuracy : ',
      metrics.accuracy_score(y_val,
                             final_clf.predict(X_val))*100)

y_pred_test = final_clf.predict(X_test)
Dt_accuracy = accuracy_score(y_test, y_pred_test)*100
print("Testing Accuracy: ", Dt_accuracy)

Training Accuracy :  94.0997739769483
Validation Accuracy :  94.1011152416357
Testing Accuracy:  94.05303593556381
```

Figure 6.4.1 – Training, Validation, and Testing accuracies are displayed.

To prove the validity of these calculations, it is very important to include both Sensitivity and Specificity [1] which are important performance metrics used to evaluate the effectiveness of a binary classification model, such as in anomaly detection.

[1] Please refer to Appendix B to understand these metrics.

The results calculated by using the formulas above for both **TNR** and **TPR** are shown below Figure 6.4.2:



```
[93]: predicted_anomalies = y_pred_test
      # True Positive
      true_positives = sum((actual == 1) and (predicted == 1) for actual, predicted in zip(y_test, predicted_anomalies))

      # False Negative
      false_negatives = sum((actual == 1) and (predicted == 0) for actual, predicted in zip(y_test, predicted_anomalies))

      # False Positives (FP)
      false_positives = sum((actual == 0) and (predicted == 1) for actual, predicted in zip(y_test, predicted_anomalies))

      # True Negatives (TN)
      true_negatives = sum((actual == 0) and (predicted == 0) for actual, predicted in zip(y_test, predicted_anomalies))

      specificity = (true_negatives/(true_negatives + false_positives)) * 100

      sensitivity = (true_positives / (true_positives + false_negatives)) * 100

      print(f'Sensitivity Rate: {sensitivity:.2f}%')
      print(f'Specificity Rate: {specificity:.2f}%')

      Sensitivity Rate: 100.00%
      Specificity Rate: 87.22%
```

Figure 6.4.2 - Calculating both Sensitivity and Specificity.

In this case, sensitivity of 100.00% indicates that the model correctly identifies all positive instances (anomalies) out of all actual positive instances and specificity Rate of 87.22% indicates that the model correctly identifies 87.22% of negative instances (non-anomalies) out of all actual negative instances.

The evaluation metrics of this particular model with these hyper-parameters shown in figure 6.4.3 as follows:

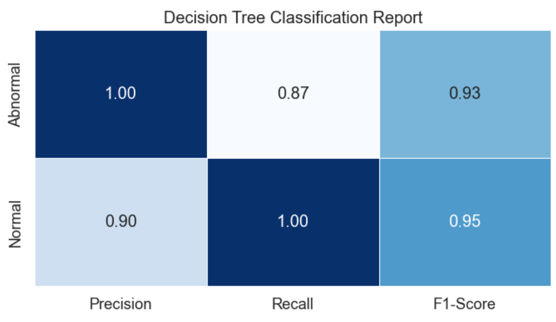
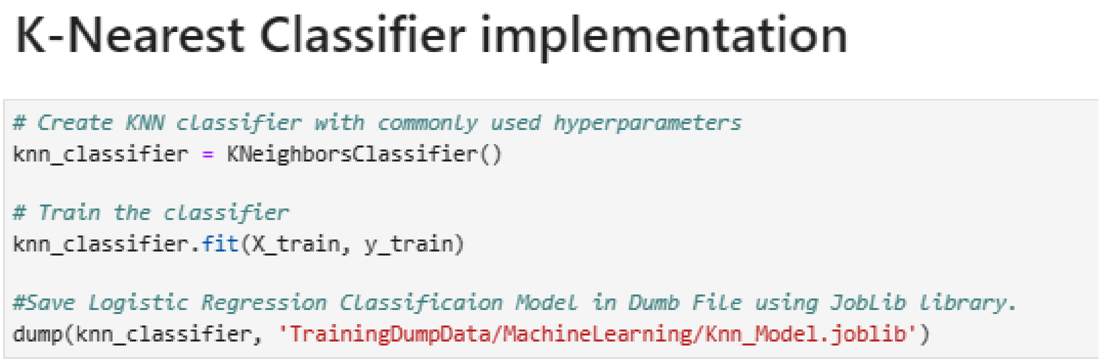


Figure 6.4.3 - Evaluation Metrics of Decision Tree Classifier.

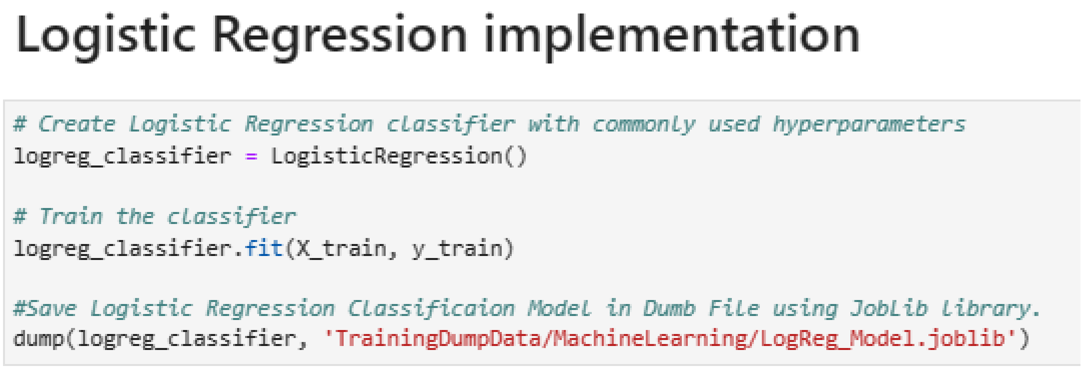
To enhance the clarity regarding the project's advancement, a comparison was conducted between the Decision Tree Classifier and alternative models constructed and trained using various algorithms. Figures [6.4.4, 6.4.5, 6.4.6, 6.4.7, 6.4.8] shows the code for each model algorithm which will be visualized afterwards into a bar chart.



K-Nearest Classifier implementation

```
# Create KNN classifier with commonly used hyperparameters
knn_classifier = KNeighborsClassifier()

# Train the classifier
knn_classifier.fit(X_train, y_train)

#Save Logistic Regression Classificaion Model in Dumb File using JobLib library.
dump(knn_classifier, 'TrainingDumpData/MachineLearning/Knn_Model.joblib')
```

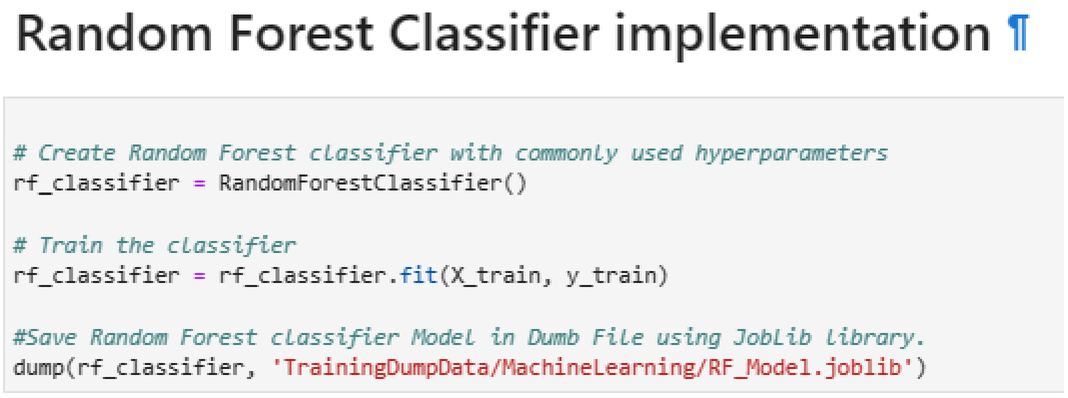
Figure 6.4.4 – Implementation of K-Nearest Classifier algorithm with default hyperparameters' values and Saving a Trained Model using Joblib.



Logistic Regression implementation

```
# Create Logistic Regression classifier with commonly used hyperparameters
logreg_classifier = LogisticRegression()

# Train the classifier
logreg_classifier.fit(X_train, y_train)

#Save Logistic Regression Classificaion Model in Dumb File using JobLib library.
dump(logreg_classifier, 'TrainingDumpData/MachineLearning/LogReg_Model.joblib')
```

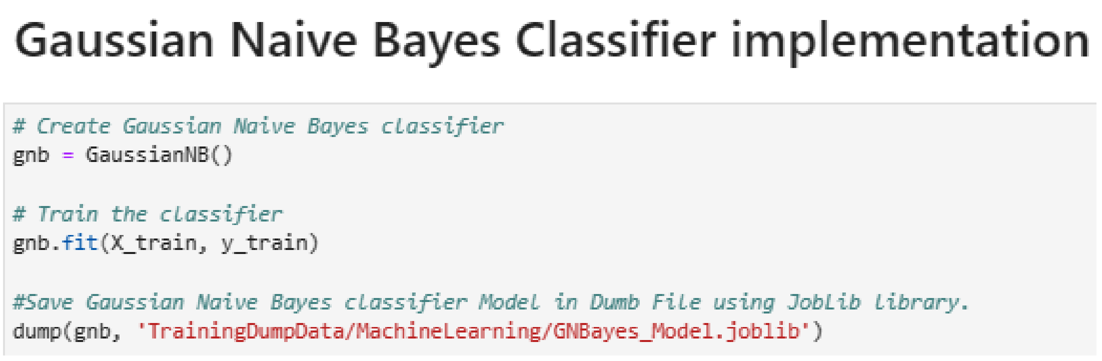
Figure 6.4.5 – Implementation of Logistic Regression algorithm with default hyperparameters' values and Saving a Trained Model using Joblib.



Random Forest Classifier implementation ¶

```
# Create Random Forest classifier with commonly used hyperparameters
rf_classifier = RandomForestClassifier()

# Train the classifier
rf_classifier = rf_classifier.fit(X_train, y_train)

#Save Random Forest classifier Model in Dumb File using JobLib library.
dump(rf_classifier, 'TrainingDumpData/MachineLearning/RF_Model.joblib')
```

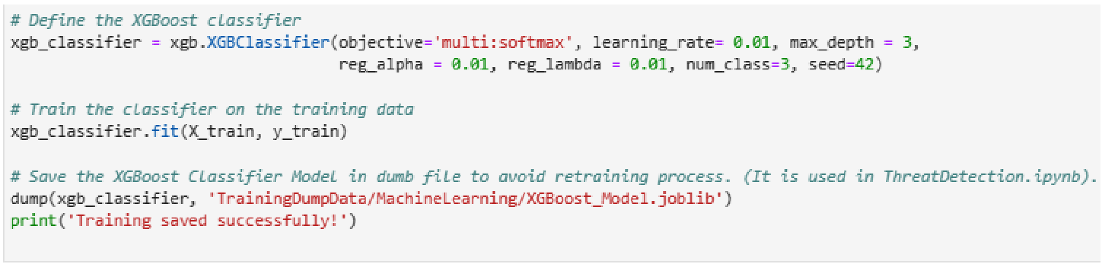
Figure 6.4.6 – Implementation of Random Forest Classifier algorithm with default hyperparameters' values and Saving a Trained Model using Joblib.



Gaussian Naive Bayes Classifier implementation

```python
# Create Gaussian Naive Bayes classifier
gnb = GaussianNB()

# Train the classifier
gnb.fit(X_train, y_train)

#Save Gaussian Naive Bayes classifier Model in Dumb File using Joblib library.
dump(gnb, 'TrainingDumpData/MachineLearning/GNBayes_Model.joblib')
```

Figure 6.4.7 Implementation of Gaussian Naïve Bayes Classifier algorithm with default hyperparameters' values and Saving a Trained Model using Joblib.



```python
# Define the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(objective='multi:softmax', learning_rate= 0.01, max_depth = 3,
                                   reg_alpha = 0.01, reg_lambda = 0.01, num_class=3, seed=42)

# Train the classifier on the training data
xgb_classifier.fit(X_train, y_train)

# Save the XGBoost Classifier Model in dumb file to avoid retraining process. (It is used in ThreatDetection.ipynb).
dump(xgb_classifier, 'TrainingDumpData/MachineLearning/XGBoost_Model.joblib')
print('Training saved successfully!')
```

Figure 6.4.8 – Implementation of Gaussian Naïve Bayes Classifier algorithm with default hyperparameters' values and Saving a Trained Model using Joblib.

Upon examining the bar chart in Figure 6.4.9, it is evident that while Decision Tree Classifier model's testing accuracy is not optimal, it outperforms the Gaussian Naive Bayes algorithm. Thus, it may be sensible to explore alternative algorithms for model training. Furthermore, it would be beneficial to delve deeper into the hyperparameters of these alternative algorithms to harness the full potential of the dataset features. This strategic approach will enable us to optimize model performance and enhance the predictive capabilities of the model.

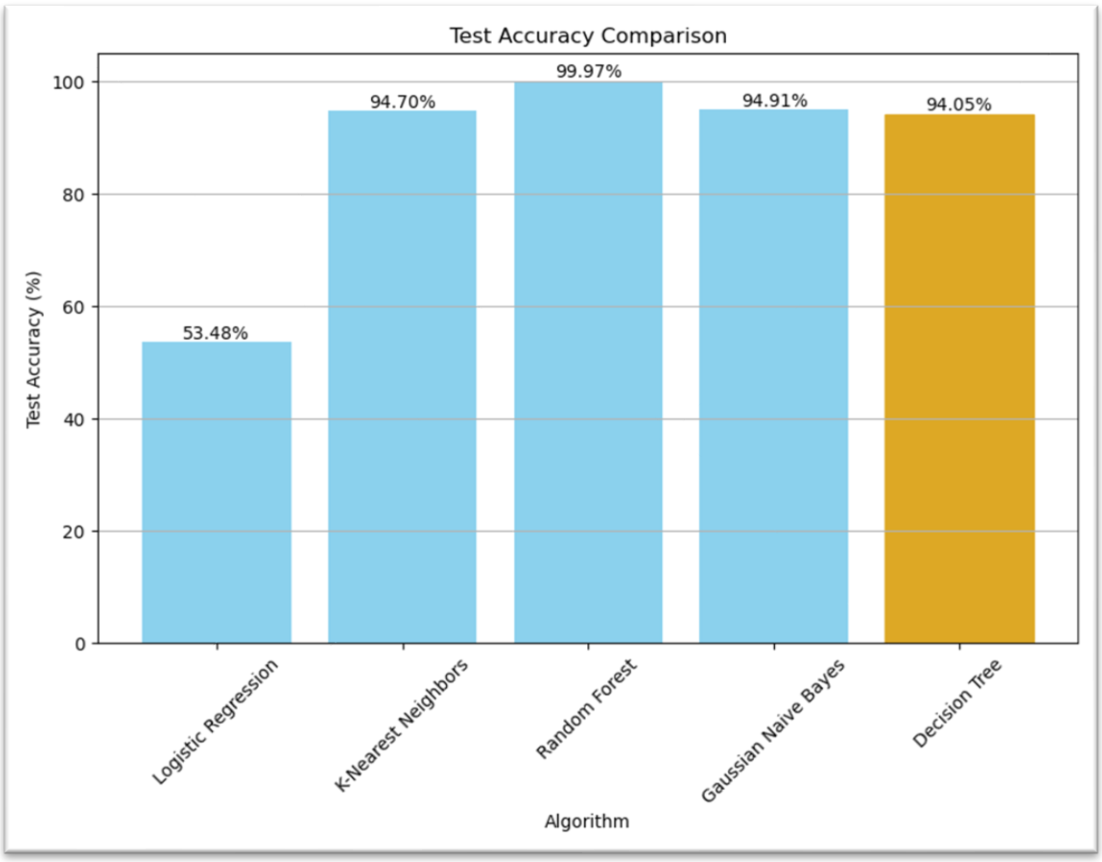


Figure 6.4.9 – Comparison between the Decision Tree Classifier model and the other algorithms.

6.5 Deep Learning (GRU)

In this section, we delve into the practical steps involved in building and training a Gated Recurrent Unit (GRU)-based neural network[2] for threat detection. As cybersecurity threats continue to evolve in complexity and sophistication, there is a growing need for advanced detection mechanisms capable of identifying anomalous behaviors indicative of potential security breaches. Leveraging TensorFlow's Keras API and the GRU architecture, we embark on a journey to develop a robust threat detection model that can effectively analyze sequential data and detect patterns associated with cyber threats.

6.5.1 Model Architecture:

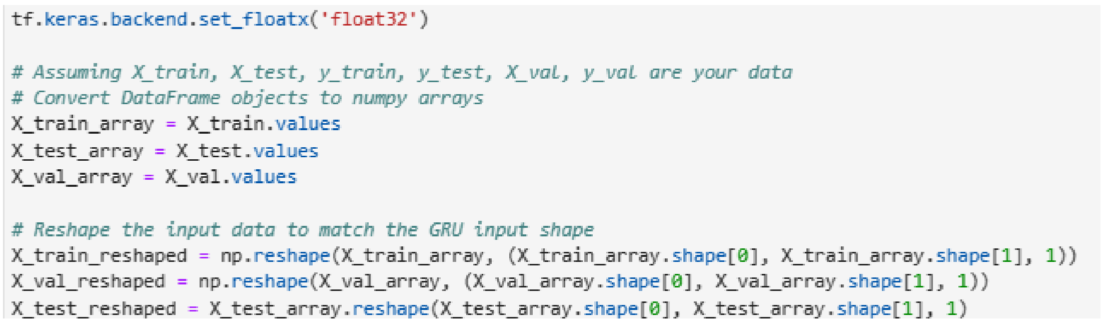
Our model architecture is built upon the foundation of TensorFlow's Keras API, chosen for its simplicity in model construction. This API facilitates a sequential design

[2] Please refer to Appendix A to fully understand how GRU works.

approach, where layers are automatically stacked one after another. At the heart of our architecture lies the GRU layer, dedicated to processing sequential data. In our example, we employ 64 units within the GRU layer to effectively capture temporal dependencies. Additionally, we incorporate a Dense layer for binary classification, discerning between "threat" and "no threat" instances.

6.5.2 **Data Pre-processing:**

Pre-processing of input data is a crucial step preceding model training. In Figure 6.5.2, We standardize the data by converting its type to float32, while also considering normalization to enhance training efficiency. Furthermore, begin by reshaping the data to align with the format expected by the GRU layer.



```
tf.keras.backend.set_floatx('float32')

# Assuming X_train, X_test, y_train, y_test, X_val, y_val are your data
# Convert DataFrame objects to numpy arrays
X_train_array = X_train.values
X_test_array = X_test.values
X_val_array = X_val.values

# Reshape the input data to match the GRU input shape
X_train_reshaped = np.reshape(X_train_array, (X_train_array.shape[0], X_train_array.shape[1], 1))
X_val_reshaped = np.reshape(X_val_array, (X_val_array.shape[0], X_val_array.shape[1], 1))
X_test_reshaped = X_test_array.reshape(X_test_array.shape[0], X_test_array.shape[1], 1)
```

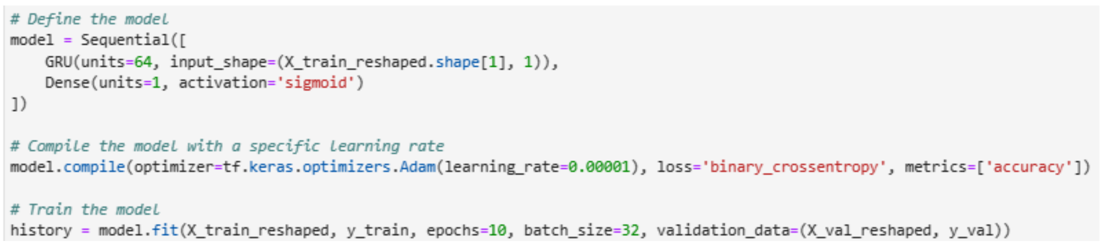
Figure 6.5.2.1 – reshaping the data with the format expected by the GRU layer.

6.5.3 **Model Compilation:**

During the model compilation phase, we configure optimization parameters to bolster model performance. The Adam Optimizer dynamically adjusts model weights throughout training to minimize loss. To mitigate overfitting, we set a conservative learning rate of 0.00001, controlling the magnitude of weight updates. Additionally, the Binary Cross-Entropy Loss function is employed to measure the difference between predicted probabilities and actual labels, aiding model convergence effectively.

6.5.4 **Model Training:**

In the model training phase, we define the number of epochs, indicating how many times the entire dataset is passed through the network. In Figure 6.5.3 shows that each training step involves a batch size of 32, determining the number of data samples utilized to update model weights. Throughout training, diligent performance monitoring is conducted, tracking both training and validation accuracy, as well as loss metrics. This meticulous monitoring allows us to identify potential issues and optimize hyperparameters, such as the learning rate, to ensure model effectiveness.



```
# Define the model
model = Sequential([
    GRU(units=64, input_shape=(X_train_reshaped.shape[1], 1)),
    Dense(units=1, activation='sigmoid')
])

# Compile the model with a specific learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32, validation_data=(X_val_reshaped, y_val))
```

Figure 6.5.4.1 – Deep learning training model

By following these steps and using the strengths of GRU architecture, you can build a powerful threat detection model capable of analyzing sequential data and identifying anomalous patterns.

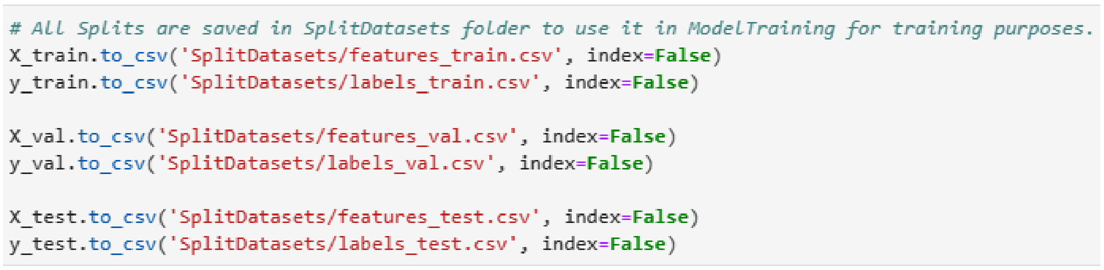
After training and saving results, we return to ThreatDetection.ipynb to efficiently load trained models without retraining and make predictions on validation and testing datasets. Subsequently, we calculate key metrics like Accuracy, Precision, Recall, F1-score, and Specificity Rate to assess model performance.

Carefully storing training results simplifies data access and management, facilitating seamless retrieval when needed. Saving trained models eliminates the need for repeated training with each code execution, optimizing resource utilization and minimizing computational overhead."

6.6 **Efficient Data Management and Model Training**

Our project adopts a systematic approach to streamline data processing and model training, resulting in significant time and resource savings. The core files, ThreatDetection.ipynb, and ModelTraining.ipynb, play essential roles in this process.

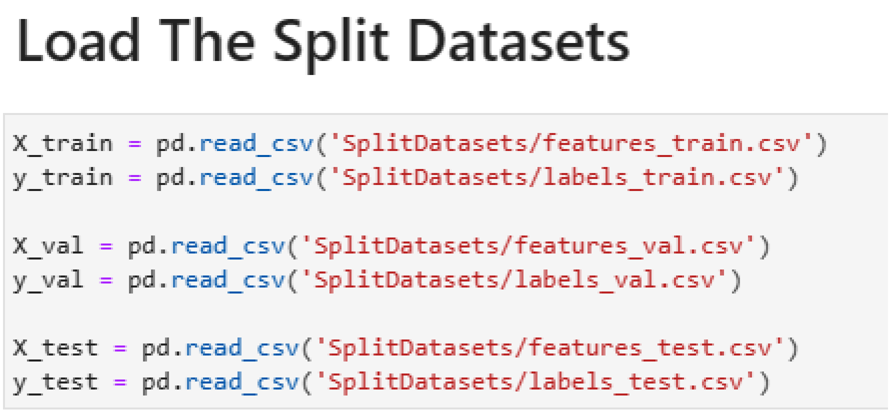
In ThreatDetection.ipynb, we focus on data splitting, storage, and model evaluation post-training. After dividing the dataset into training, validation, and testing subsets, we efficiently store the data in the SplitDatasets folder, ensuring easy access and management for subsequent model training and evaluation.



```python
# All Splits are saved in SplitDatasets folder to use it in ModelTraining for training purposes.
X_train.to_csv('SplitDatasets/features_train.csv', index=False)
y_train.to_csv('SplitDatasets/labels_train.csv', index=False)

X_val.to_csv('SplitDatasets/features_val.csv', index=False)
y_val.to_csv('SplitDatasets/labels_val.csv', index=False)

X_test.to_csv('SplitDatasets/features_test.csv', index=False)
y_test.to_csv('SplitDatasets/labels_test.csv', index=False)
```

Figure 6.5.4 – Store the data in the Split Datasets folder.

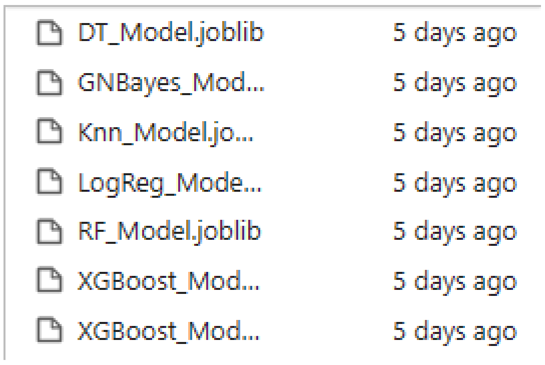
ModelTraining.ipynb focuses on training models using the dataset located in the SplitDatasets folder and storing the results, utilizing both machine learning and deep learning algorithms. By diligently saving trained models' results using joblib, we reduce execution time and avoid retraining with each code execution.



Load The Split Datasets

```python
X_train = pd.read_csv('SplitDatasets/features_train.csv')
y_train = pd.read_csv('SplitDatasets/labels_train.csv')

X_val = pd.read_csv('SplitDatasets/features_val.csv')
y_val = pd.read_csv('SplitDatasets/labels_val.csv')

X_test = pd.read_csv('SplitDatasets/features_test.csv')
y_test = pd.read_csv('SplitDatasets/labels_test.csv')
```

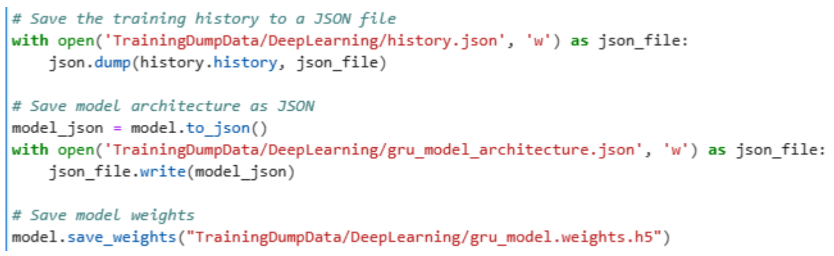
Figure 6.5.4 – Load the Split Datasets.



DT_Model.joblib	5 days ago
GNBayes_Mod...	5 days ago
Knn_Model.jo...	5 days ago
LogReg_Mode...	5 days ago
RF_Model.joblib	5 days ago
XGBoost_Mod...	5 days ago
XGBoost_Mod...	5 days ago

Figure 6.5.4 – This file stores trained machine learning models.

In deep learning, especially with GRU, storing three crucial variables becomes essential to save the training. We save the training history in JSON format to track the model's performance over epochs, ensuring reproducibility by storing the model architecture as JSON, and saving model weights to retain learned patterns and optimize performance.



```python
# Save the training history to a JSON file
with open('TrainingDumpData/DeepLearning/history.json', 'w') as json_file:
    json.dump(history.history, json_file)

# Save model architecture as JSON
model_json = model.to_json()
with open('TrainingDumpData/DeepLearning/gru_model_architecture.json', 'w') as json_file:
    json_file.write(model_json)

# Save model weights
model.save_weights("TrainingDumpData/DeepLearning/gru_model.weights.h5")
```

Figure 6.5.4 – Save training history (JSON), model architecture (JSON), and weights (HDF5).

After training and saving results, we return to ThreatDetection.ipynb to efficiently load trained models without retraining and make predictions on validation and testing datasets. Subsequently, we calculate key metrics like Accuracy, Precision, Recall, F1-score, and Specificity Rate to assess model performance. Carefully storing training results simplifies data access and management, facilitating seamless retrieval when needed. Saving trained models eliminates the need for repeated training with each code execution, optimizing resource utilization and minimizing computational overhead.

Chapter 7
PROJECT TESTING

Chapter 7
Project testing

In the ongoing testing phase of our project, we evaluate the performance and efficacy of the trained models and assess their effectiveness in detecting cyber anomalies. Graphical visualizations serve as a pivotal tool in this assessment, providing real-time insights into the behaviour and effectiveness of our models. By plotting key metrics such as precision, recall, and F1-score across various thresholds or parameter values, we gain a dynamic understanding of how our models perform under different conditions.

7.1 Machine Learning

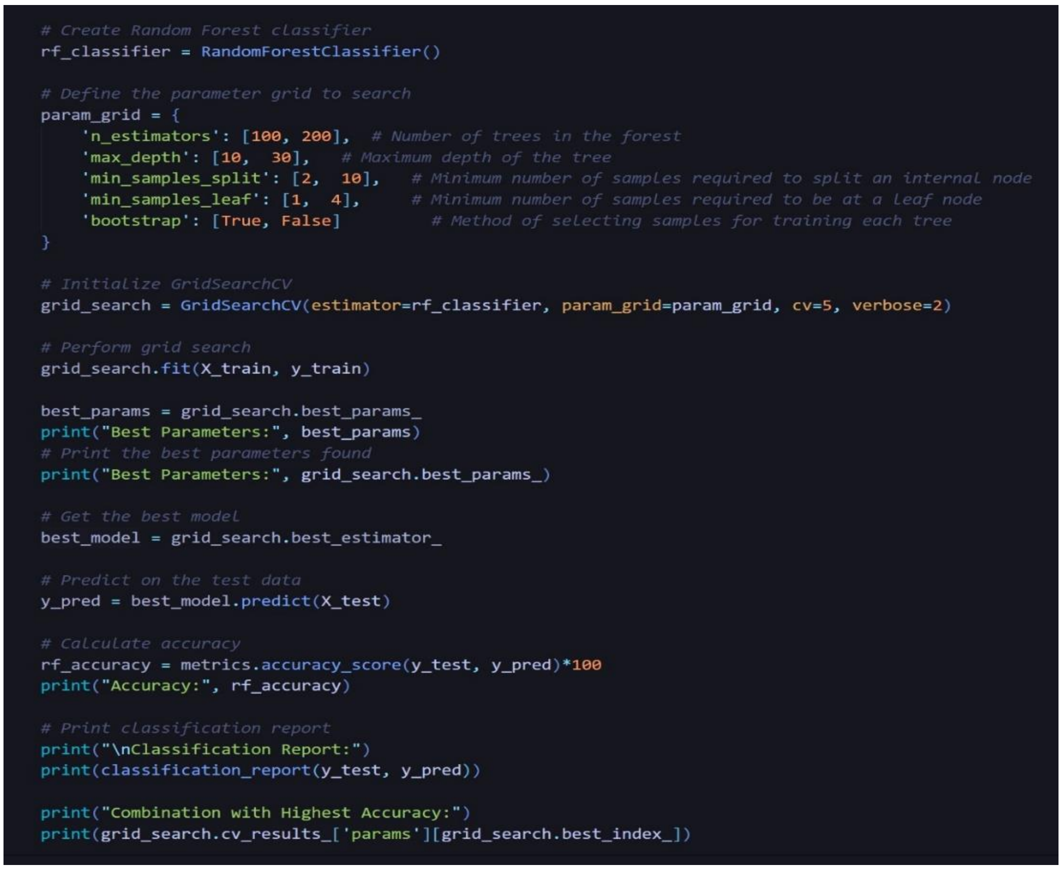
7.1.1 Grid Search (Hyperparameter Tuning)

Given the complexity and significance of detecting cyber anomalies in network traffic, hyperparameter tuning is crucial to optimize the performance of the chosen machine learning model. Whether grid search is the best method for this purpose depends on several factors including the size of hyperparameter space, computational resources, time constraints, and the specific requirements for model accuracy and generalization. Therefore, grid search can be a beneficial tool. Here's why:

- **Comprehensive Search:** Grid search systematically explores multiple combinations of parameters, ensuring that you test a wide range of possibilities and likely find an optimal solution within the defined grid.
- **Improvement in Model Performance:** By thoroughly searching through the hyperparameters, you can potentially improve the model's ability to accurately detect anomalies.

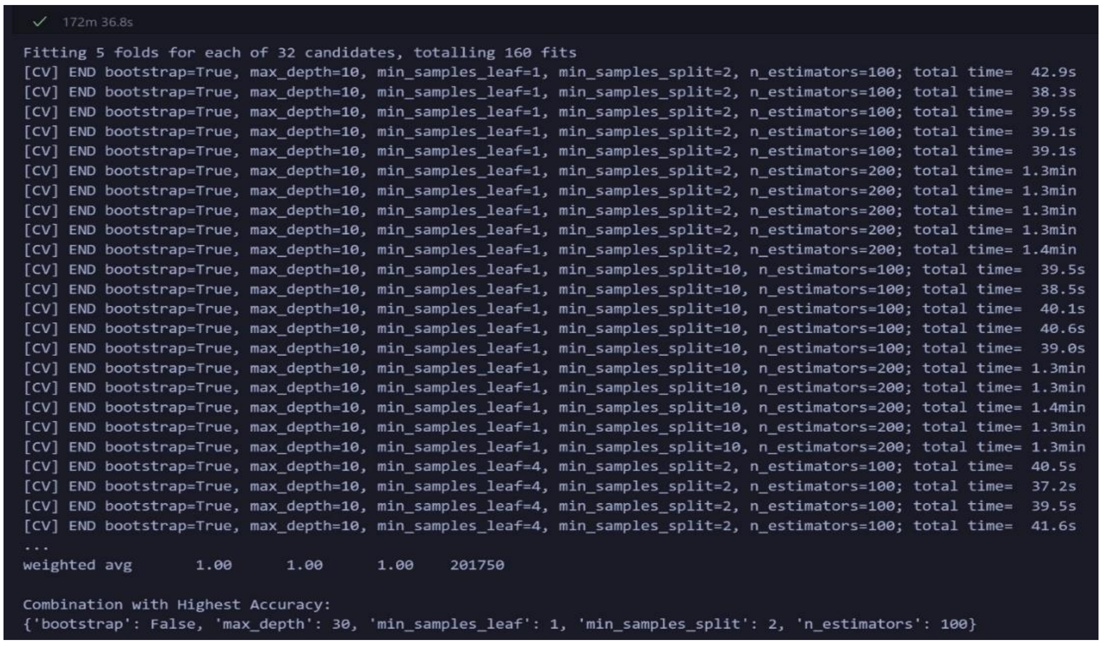
7.1.2 Random Forest (after Grid Search)

Following the implementation of the grid search method to identify the optimal combination of hyperparameters, the outcome was a disappointing overfit in comparison to the default hyperparameter values outlined in the report part 1 as shown below Figures 7.1.2.1, 7.1.2.2, 7.1.2.3:



```python
# Create Random Forest classifier
rf_classifier = RandomForestClassifier()

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 200],  # Number of trees in the forest
    'max_depth': [10,  30],   # Maximum depth of the tree
    'min_samples_split': [2,  10],   # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1,  4],     # Minimum number of samples required to be at a leaf node
    'bootstrap': [True, False]        # Method of selecting samples for training each tree
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, verbose=2)

# Perform grid search
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print("Best Parameters:", best_params)
# Print the best parameters found
print("Best Parameters:", grid_search.best_params_)

# Get the best model
best_model = grid_search.best_estimator_

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

# Calculate accuracy
rf_accuracy = metrics.accuracy_score(y_test, y_pred)*100
print("Accuracy:", rf_accuracy)

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("Combination with Highest Accuracy:")
print(grid_search.cv_results_['params'][grid_search.best_index_])
```

Figure 7.1.2.1 - Grid Search method for Random Forest hyperparameters tuning.



```
✓ 172m 36.8s

Fitting 5 folds for each of 32 candidates, totalling 160 fits
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=  42.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=  38.3s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=  39.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=  39.1s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=  39.1s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 1.4min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=  39.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=  38.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=  40.1s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=  40.6s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=  39.0s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time= 1.4min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time= 1.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=  40.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=  37.2s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=  39.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=  41.6s
...
weighted avg       1.00      1.00      1.00    201750

Combination with Highest Accuracy:
{'bootstrap': False, 'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

Figure 7.1.2.2 – Best hyperparameters combination with highest accuracy.

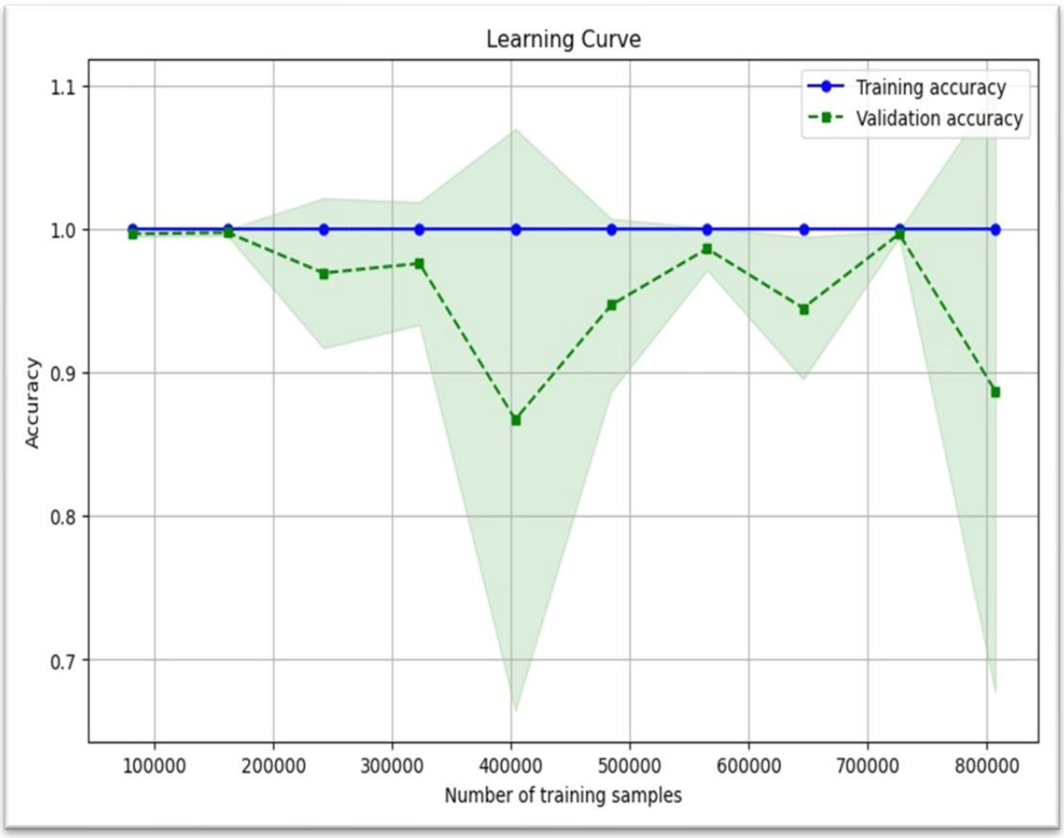
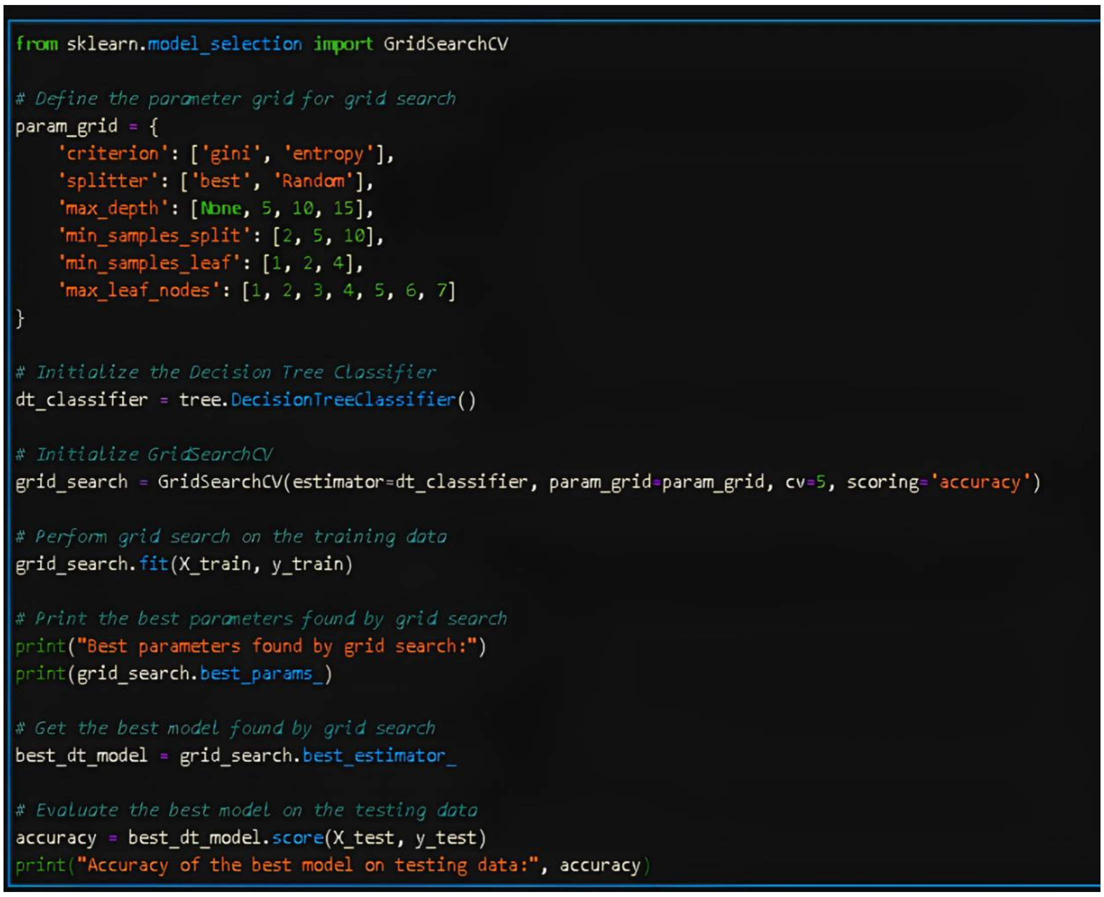


Figure 7.1.2.3 – Random Forest (after Grid Search) overfit.

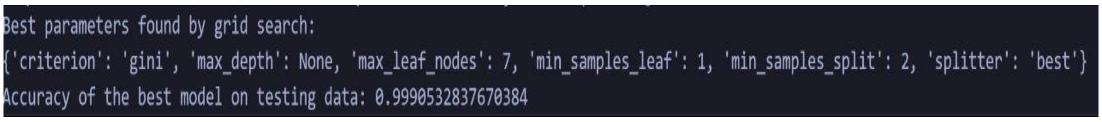


```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'Random'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_leaf_nodes': [1, 2, 3, 4, 5, 6, 7]
}

# Initialize the Decision Tree Classifier
dt_classifier = tree.DecisionTreeClassifier()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Perform grid search on the training data
grid_search.fit(X_train, y_train)

# Print the best parameters found by grid search
print("Best parameters found by grid search:")
print(grid_search.best_params_)

# Get the best model found by grid search
best_dt_model = grid_search.best_estimator_

# Evaluate the best model on the testing data
accuracy = best_dt_model.score(X_test, y_test)
print("Accuracy of the best model on testing data:", accuracy)
```

Figure 7.1.2.3 – Grid Search method for Decision Tree Classifier hyperparameters tuning.

7.1.3 **Decision Tree Classifier (after Grid Search)**

Regarding the DecisionTreeClassifier() model initially discussed in part 1 of the report, it has significantly improved the overall accuracy compared to the test results obtained in the same part when only basic hyperparameter tuning was applied. This enhancement is illustrated in the figures below Figure 7.1.5:



```
Best parameters found by grid search:
{'criterion': 'gini', 'max_depth': None, 'max_leaf_nodes': 7, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
Accuracy of the best model on testing data: 0.9990532837670384
```

Figure 7.1.3.1 – Grid Search method for Decision Tree Classifier hyperparameters tuning.

The improvement in accuracy is evident when comparing the models trained without hyperparameter tuning to those that experienced the process. In the Testing section, we will dive deeper into a more detailed and advanced explanation of the decision tree and grid search techniques. Similarly, the learning curve gives interesting lines which will be discussed in testing section.

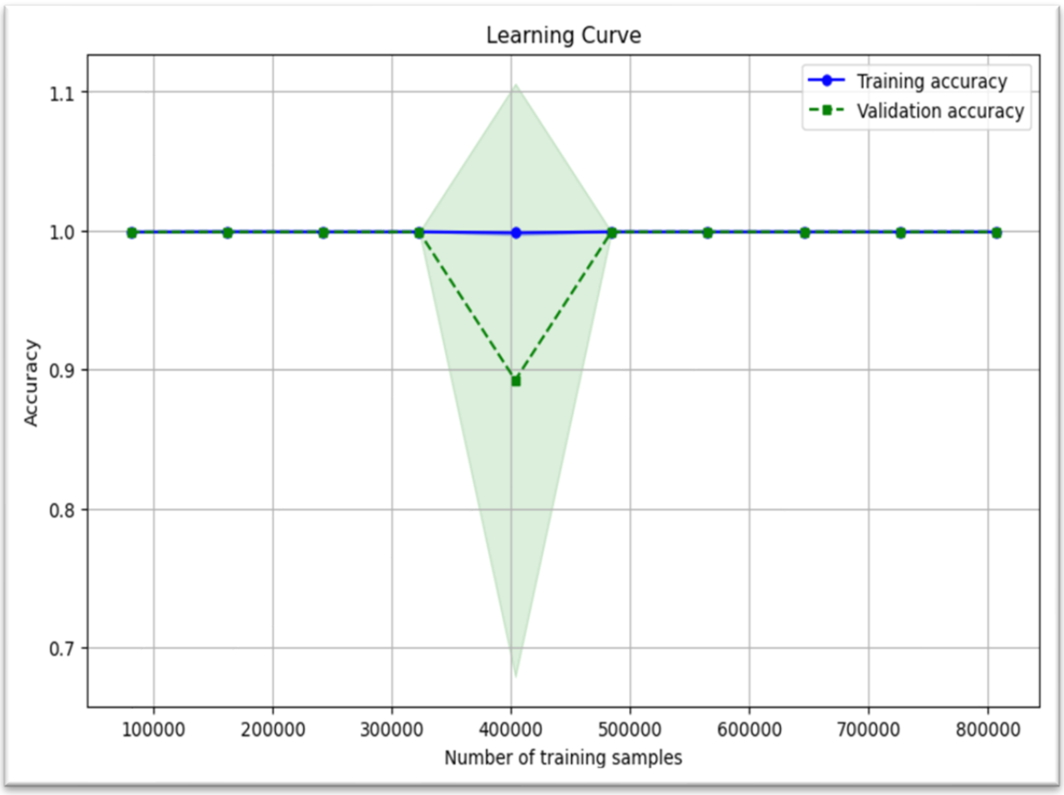


Figure 7.1.3.2 – Decision Tree Classifier (after Grid Search) no overfit

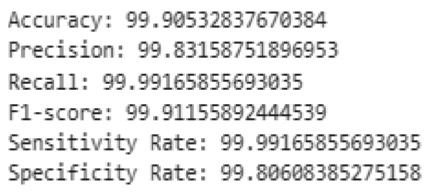
7.1.3.1 Decision Tree Performance Analysis

The Decision Tree model achieved exceptional performance on the anomaly detection task. Here's a detailed breakdown of its evaluation metrics[3]:

- **Accuracy:** 99.9053% - This metric indicates the model's ability to correctly classify both normal and anomalous instances. The high accuracy signifies the model's effectiveness in distinguishing between the two classes.
- **Precision:** 99.8316% - Precision reflects the proportion of predicted anomalies that are truly anomalous. This value suggests that the model generates very few false positives (normal instances incorrectly classified as anomalies).

[3] Please Refer to Appendix B to fully understand the evaluation metrics.

- **Recall (Sensitivity):** 99.9917% - Recall, also known as sensitivity, measures the model's capacity to identify actual anomalies. The exceptionally high recall indicates the model rarely misses true anomalies (false negatives).
- **F1-score:** 99.9116% - The F1-score is a harmonic mean between precision and recall, providing a balanced view of the model's performance. The high F1-score further emphasizes the model's effectiveness in both precision and recall.
- **Specificity:** 99.8061% - Specificity represents the model's ability to correctly identify normal instances. This value, though slightly lower than other metrics, demonstrates the model's proficiency in not mistakenly classifying normal data as anomalies.



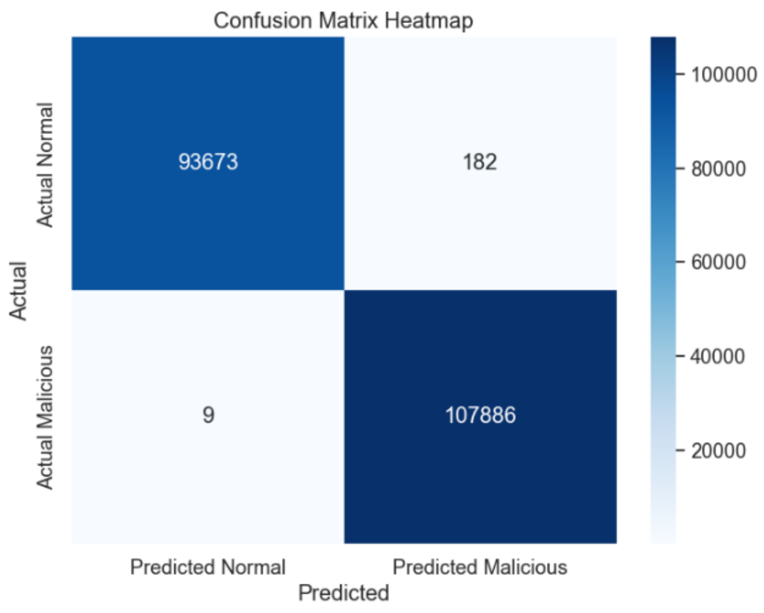
```
Accuracy: 99.90532837670384
Precision: 99.83158751896953
Recall: 99.99165855693035
F1-score: 99.91155892444539
Sensitivity Rate: 99.99165855693035
Specificity Rate: 99.80608385275158
```

Figure 7.1.3.3 evaluation metrics of Decision Tree Classifier.

7.1.3.2 Visualizing Performance with Confusion Matrix

The confusion matrix further corroborates our model's performance, revealing that the majority of samples are correctly classified. Specifically, 93,673 normal samples are accurately identified, along with 107,886 malicious samples. Despite these high accuracies, a small number of misclassifications are evident, with 182 normal samples mistakenly identified as malicious and 9 malicious samples incorrectly labeled as normal. These findings underscore the robustness of our model in accurately detecting anomalies, albeit with some room for improvement in minimizing misclassifications.

Figure 7.1.3.4 – Confusion Matrix Heatmap of Decision Tree Classifier



7.1.3.3 Analysis Conclusion

The Decision Tree model demonstrates remarkable performance in anomaly detection. Its high accuracy, precision, recall, and F1-score highlight its ability to accurately distinguish between normal and anomalous data. While the specificity is slightly lower, the overall results suggest the model is effective with minimal misclassifications. By incorporating the confusion matrix heatmap (if available), you can further strengthen the analysis and provide a more comprehensive visual representation of the model's performance.

7.1.3.4 XGBoost classifier Performance Analysis

The XGBoost classifier demonstrates high accuracy across all phases of training, validation, and testing. It effectively distinguishes between normal and malicious samples, with a strong balance between precision and recall. While misclassifications are minimal, further optimization may be needed to reduce false positives and negatives. Overall, the model shows promise in accurately detecting anomalies within the dataset.

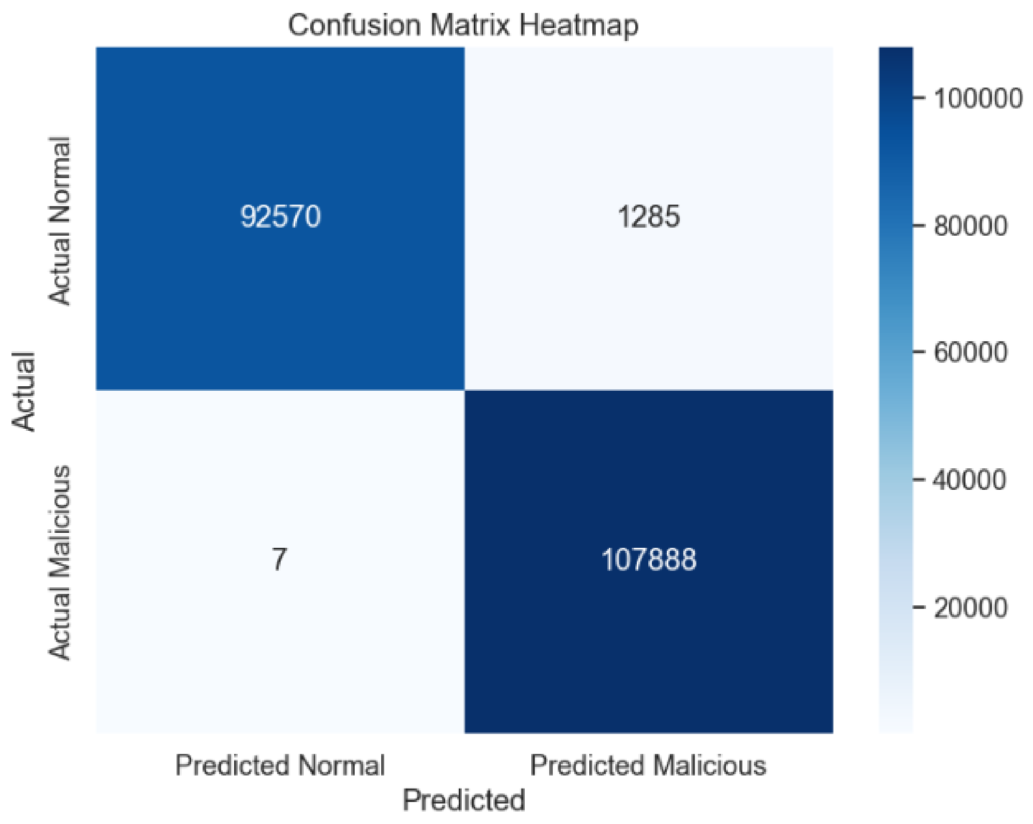
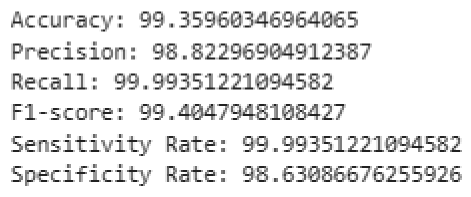


Figure 7.1.3.4.1 – Confusion Matrix Heatmap of XGBoost Classifier



```
Accuracy: 99.35960346964065
Precision: 98.82296904912387
Recall: 99.99351221094582
F1-score: 99.4047948108427
Sensitivity Rate: 99.99351221094582
Specificity Rate: 98.63086676255926
```

Figure 7.1.3.4.2 - Evaluation Metrics of XGBoost Classifier

7.1.3.5 Logistic Regression classifier Performance Analysis

The Logistic Regression model exhibits poor performance in accurately classifying normal samples, with no correct identifications. Conversely, it shows perfect recall for malicious samples, indicating it correctly identifies all instances of malicious activity. However, this comes at the cost of misclassifying a significant number of normal samples as malicious. The model's precision and specificity rates are notably low, suggesting a high rate of false positives. Further optimization and feature engineering may be necessary to improve its performance.

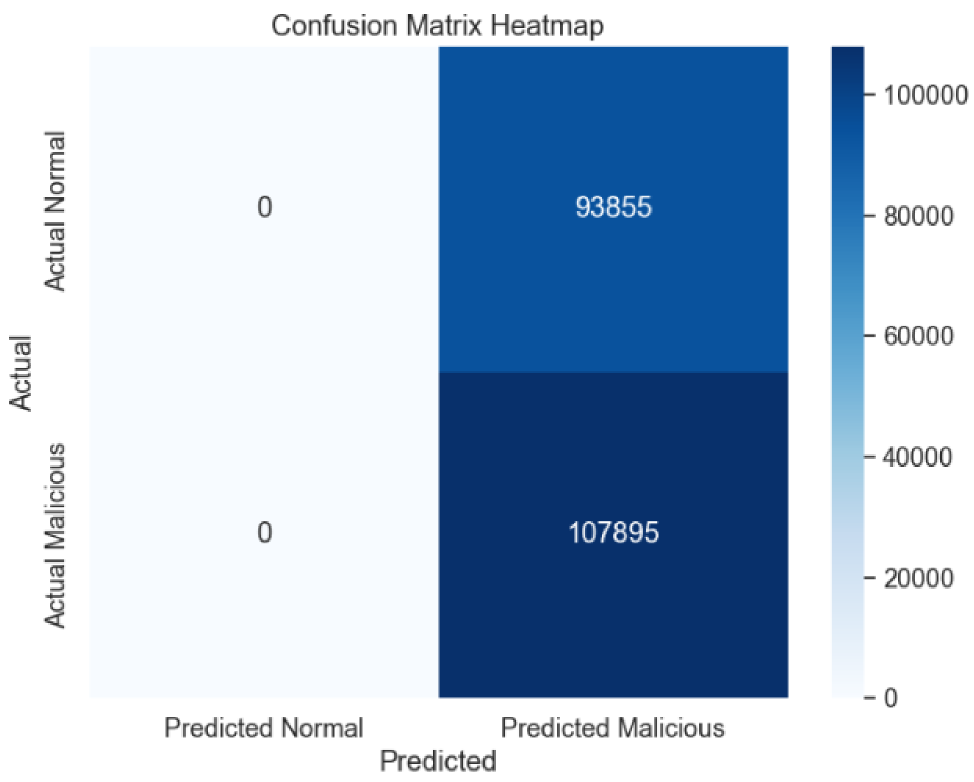
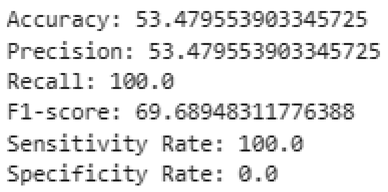


Figure 7.1.3.5.1 Confusion Matrix Heatmap of Logistic Regression classifier



```
Accuracy: 53.479553903345725
Precision: 53.479553903345725
Recall: 100.0
F1-score: 69.68948311776388
Sensitivity Rate: 100.0
Specificity Rate: 0.0
```

Figure 7.1.3.5.2 - Evaluation Metrics of Logistic Regression classifier

7.1.3.6 K-Nearest Neighbors (KNN) classifier Performance Analysis

The K-Nearest Neighbor (KNN) model demonstrates respectable accuracy and performance across all phases of training, validation, and testing. It effectively distinguishes between normal and malicious samples with relatively high precision and recall rates. Additionally, it shows reasonable sensitivity and specificity in identifying anomalies, with a moderate number of false positives and negatives. Overall, the KNN model proves to be a dependable approach for detecting cybersecurity threats within the dataset.

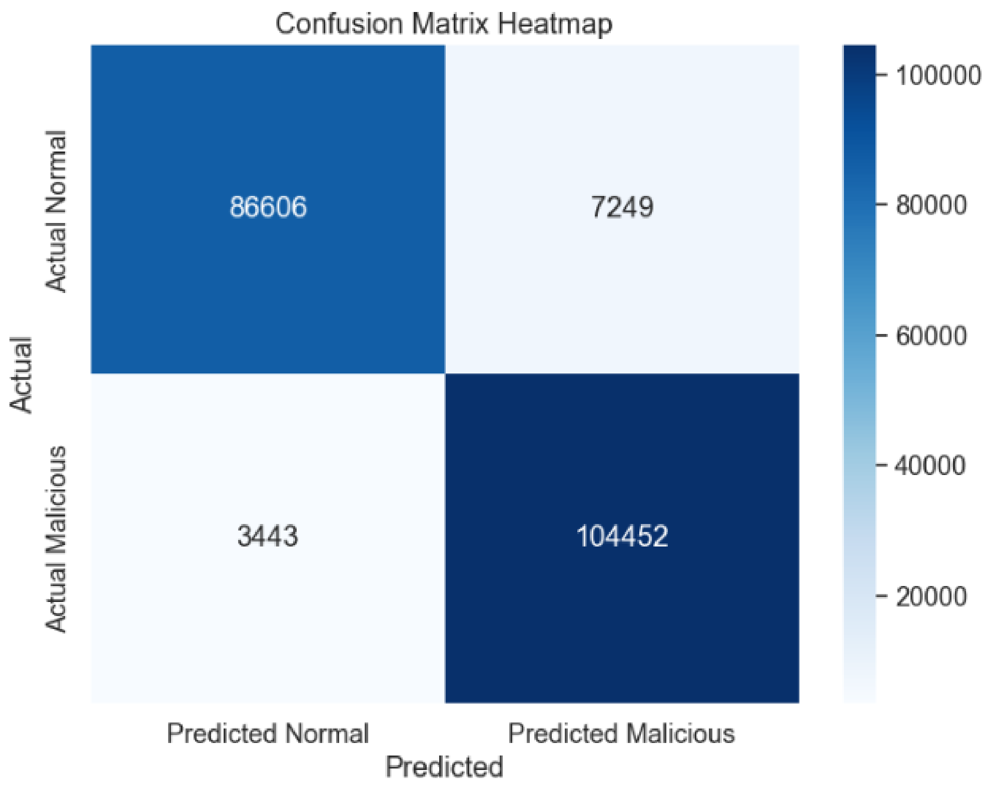
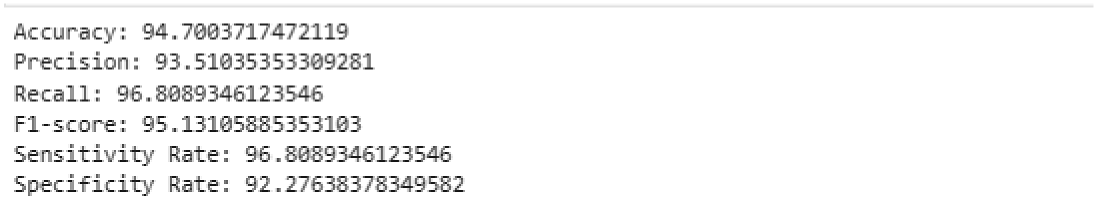


Figure 7.1.3.6.1 – Confusion Matrix Heatmap of K-Nearest Neighbors (KNN) classifier



```
Accuracy: 94.7003717472119
Precision: 93.51035353309281
Recall: 96.8089346123546
F1-score: 95.13105885353103
Sensitivity Rate: 96.8089346123546
Specificity Rate: 92.27638378349582
```

Figure 7.1.3.6.2 - Evaluation Metrics of K-Nearest Neighbors (KNN) classifier.

7.1.3.7 Random Forest Classifier Performance Analysis

The Random Forest Classifier achieves exceptional accuracy and performance across all phases of training, validation, and testing. It effectively distinguishes between normal and malicious samples with high precision and recall rates. Additionally, it demonstrates robustness in detecting anomalies, with minimal false positives and negatives. Overall, the Random Forest Classifier proves to be a reliable and effective model for identifying cybersecurity threats within the dataset.

7.1.3.8 Gaussian Naive Bayes Classifier Performance Analysis

The Gaussian Naive Bayes model demonstrates decent accuracy and effectiveness in classifying normal and malicious samples. While it exhibits a high recall rate, indicating its ability to identify all malicious instances, it shows slightly lower precision. Despite this, the model maintains reasonable specificity, ensuring accurate identification of normal instances. Overall, the Gaussian Naive Bayes model presents a reliable approach for detecting cybersecurity threats within the dataset.

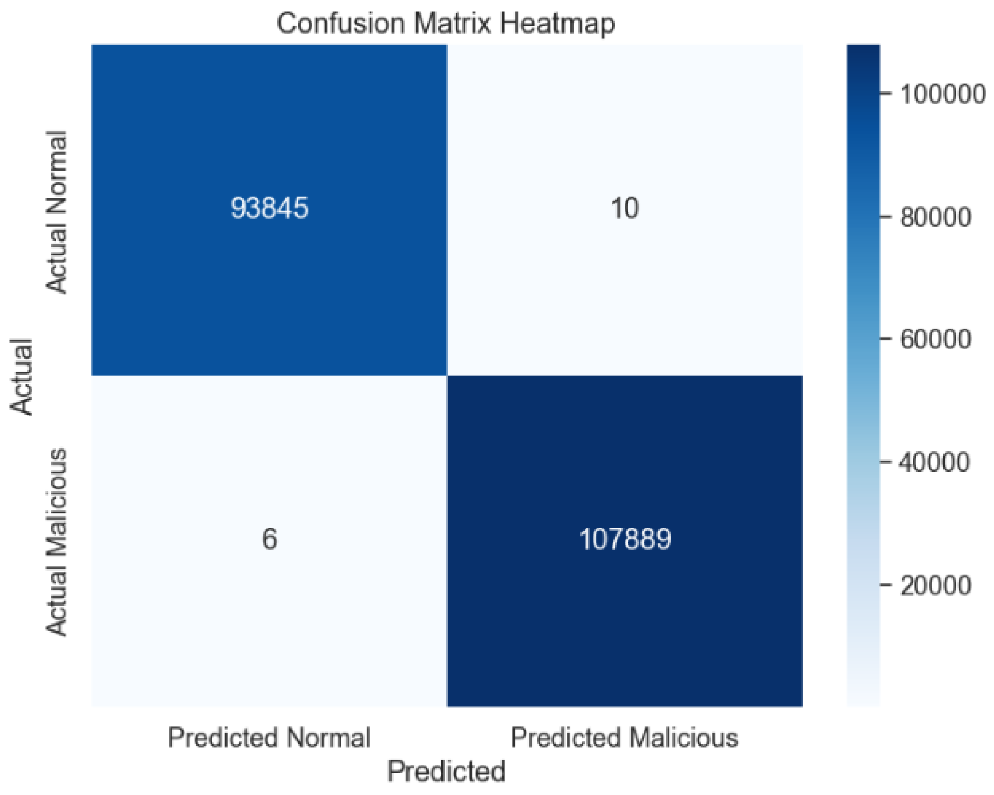
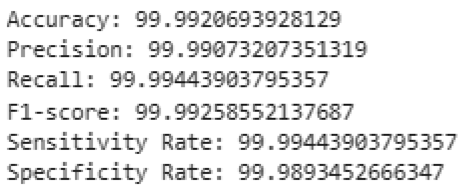


Figure 7.1.3.8.1 – Confusion Matrix Heatmap of Random Forest Classifier



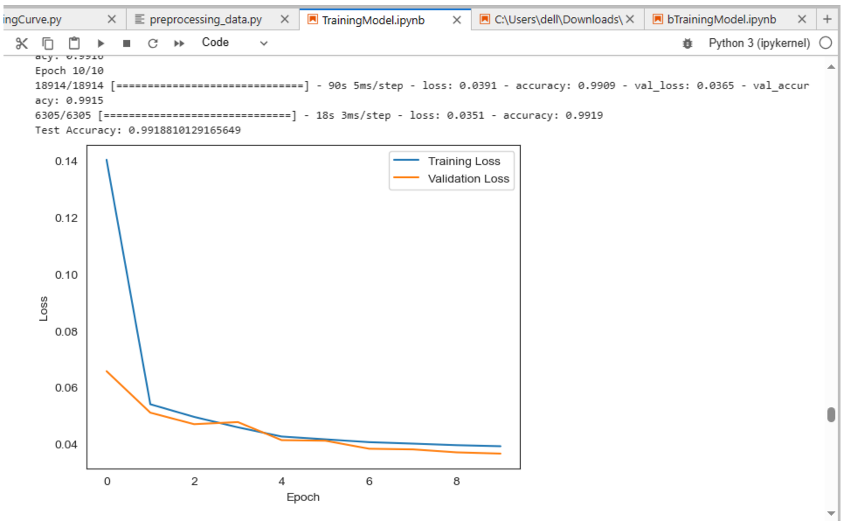
```
Accuracy: 99.9920693928129
Precision: 99.99073207351319
Recall: 99.99443903795357
F1-score: 99.99258552137687
Sensitivity Rate: 99.99443903795357
Specificity Rate: 99.9893452666347
```

Figure 7.1.3.8.2 - Evaluation Metrics of Random Forest Classifier

7.2 **Deep Learning**

7.2.1 **Training and Validation Loss:**

The Figure 5 plot shows how the loss, indicating the model's error, changes over epochs during training and validation. As training progresses, both the training and validation loss decrease. This indicates that the model is learning and becoming more accurate in predicting the target variable.



```
Epoch 10/10
18914/18914 [==============================] - 90s 5ms/step - loss: 0.0391 - accuracy: 0.9909 - val_loss: 0.0365 - val_accur
acy: 0.9915
6305/6305 [==============================] - 18s 3ms/step - loss: 0.0351 - accuracy: 0.9919
Test Accuracy: 0.9918810129165649
```

Figure 7.2.1 – GRU Loss Learning Curve.

7.2.2 **Training & Validation Accuracy:**

Training and Validation Accuracy: This plot shows the accuracy of the model on both the training and validation datasets across epochs. As training progresses, the accuracy of the model improves for both the training and validation sets.

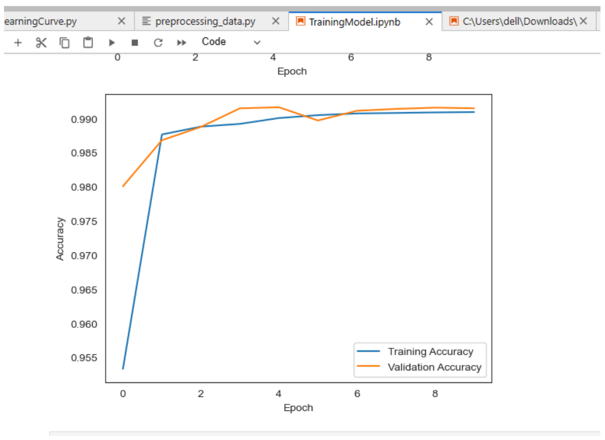
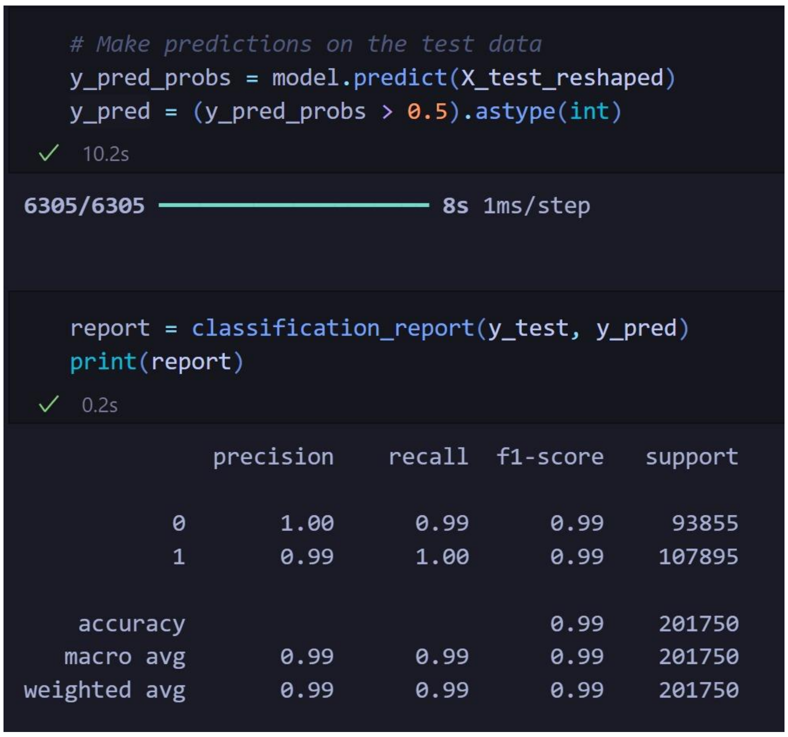


Figure 7.2.2. GRU Accuracy Learning Curve.

7.3 Evaluation Metrics:

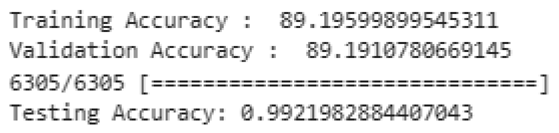
After training the model, we use it to predict outcomes on our test data. The *model.predict(X_test_reshaped)* line applies our trained model to the test dataset, generating predicted probabilities for each sample. Next, as shown in figure 7.3.1, we convert these probabilities into class labels using *(y_pred_probs > 0.5).astype(int)*. Here, we set a threshold of 0.5; if a probability exceeds this threshold, the sample is labelled as Malicious(1); otherwise, it's labelled as Benign(0). By doing this, we transform the model's continuous predictions into discrete class labels, allowing for easier interpretation and evaluation of the model's performance. The result of the classification report will be discussed on testing phase.



```
# Make predictions on the test data
y_pred_probs = model.predict(X_test_reshaped)
y_pred = (y_pred_probs > 0.5).astype(int)
✓ 10.2s

6305/6305 ━━━━━━━━━━━━━━━━━━━━ 8s 1ms/step

report = classification_report(y_test, y_pred)
print(report)
✓ 0.2s

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     93855
           1       0.99      1.00      0.99    107895

    accuracy                           0.99    201750
   macro avg       0.99      0.99      0.99    201750
weighted avg       0.99      0.99      0.99    201750
```

Figure 7.3.1. Prediction for Deep Learning GRU Model.

Previously, during the machine learning phase, several verifications were conducted. Now, to highlight the distinctions between machine learning and deep learning outcomes, we will replicate the same steps. Our initial focus is on comparing the training, validation, and testing accuracies within the context of deep learning, as illustrated in Figure 7.3.1.

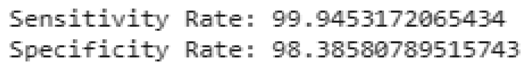


```
Training Accuracy :  89.19599899545311
Validation Accuracy :  89.1910780669145
6305/6305 [==============================]
Testing Accuracy: 0.9921982884407043
```

Figure 7.3.1 – All Accuracies (Deep Learning).

In our analysis, the beginning was by verifying the initial accuracies. Surprisingly, there is no evidence of overfitting, as both validation and testing accuracies exceed the training accuracy. This suggests that the model has been well-trained and presents effective generalization.

Moving beyond accuracy, sensitivity and specificity are essential for testing purposes. Sensitivity (recall) in the deep learning model is comparable to that of the machine learning model, but it performs even better. We will confirm this improvement through the confusion matrix analysis. Meanwhile, specificity (true negative rate) is slightly lower in deep learning compared to machine learning. However, this reduction is not concerning, as sensitivity remains the more critical consideration. Ultimately, there exists a trade-off between sensitivity and specificity for both machine learning and deep learning models. Figure 7.3.2 shows the sensitivity and specificity in deep learning.



```
Sensitivity Rate: 99.9453172065434
Specificity Rate: 98.38580789515743
```

Figure 7.3.2. Sensitivity and Specificity (Deep Learning).

After that, to demonstrate the improved sensitivity of the deep learning model compared to its machine learning complement, a heatmap plot is generated as sin Figure 7.3.3. The plot reveals that only 9 samples were falsely predicted as malicious when they were, in fact, benign. This count is significantly lower than the corresponding false predictions in the machine learning model. However, it’s worth noting that the number of incorrect predictions for normal samples is higher in deep learning than in the machine learning approach.

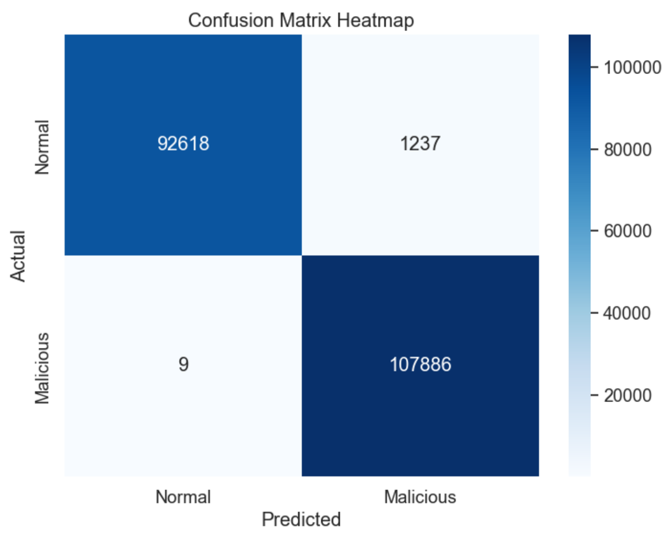


Figure 7.3.3. Confusion Matrix (Deep Learning).

Finally, let's underline the importance of evaluation metrics in assessing both the generality and accuracy of our model. As represented in Figure 7.3.4, our deep learning model closely aligns with the performance of the machine learning. However, there's a bright side: our deep learning model offers improved safety. This advantage arises from its higher sensitivity accuracy, which is a critical factor in certain applications.

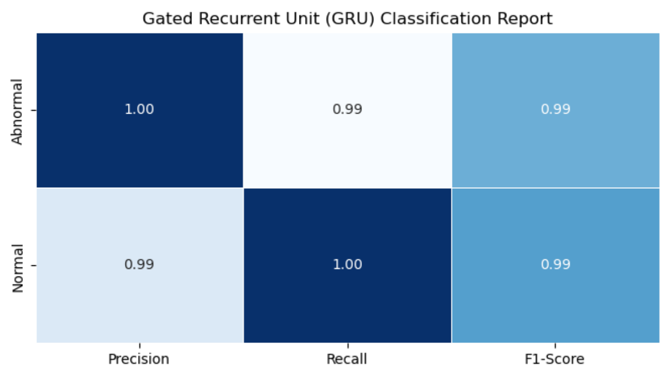


Figure 7.3.4. Classification Report for Evaluation Metrics (Deep Learning)

7.4 Overall Analysis Summary

In conclusion, while exploring various machine learning models like Random Forest initially showed promise, the challenge of overfitting encouraged exploration of alternative approaches. Using Decision Trees with grid search hyperparameter tuning facilitated the overfitting issue and returned improved results. The testing phase participated a crucial role in validating these findings, confirming the effectiveness of the refined approach. Comprehensive testing across both machine learning and deep learning models provided valuable understandings into their capabilities and limitations, assisting knowledgeable decisions in future projects.

Figure 7.4.1 shows the evaluation metrics for all models used for machine learning which can be beneficial to represent the most effective and applicable models in our specific dataset. The graph shows that our dataset has linear and non-linear values because of the poor performance in Logistic Regression which in most cases, it deals with linearity or curved classification. Random forest might give high performance yet, as discussed earlier, it was causing overfitting.

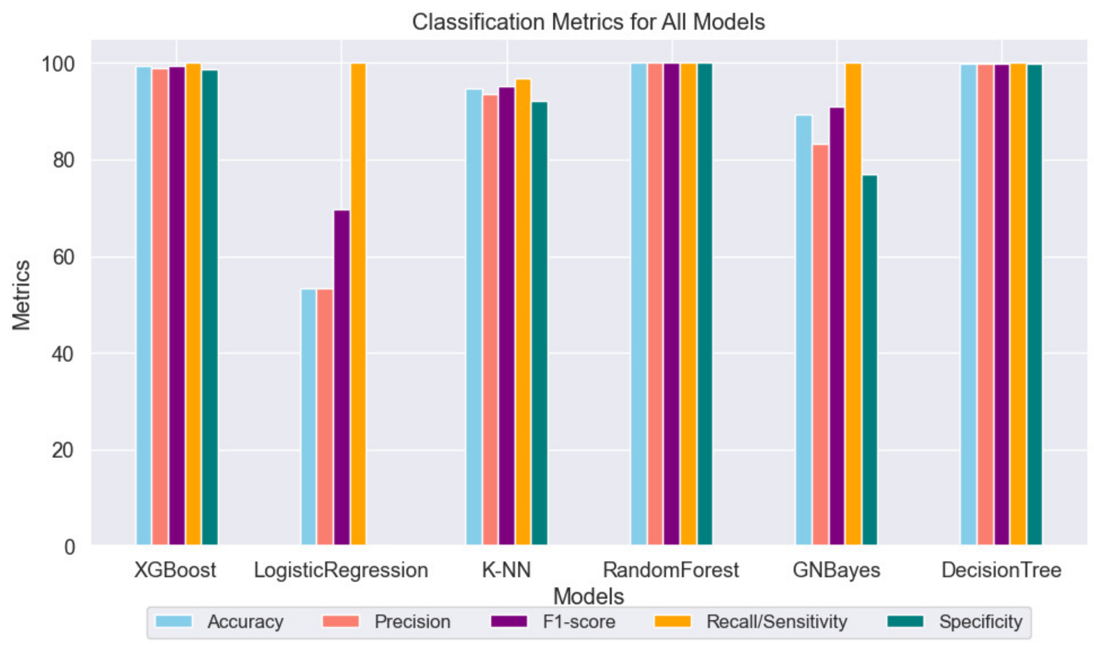


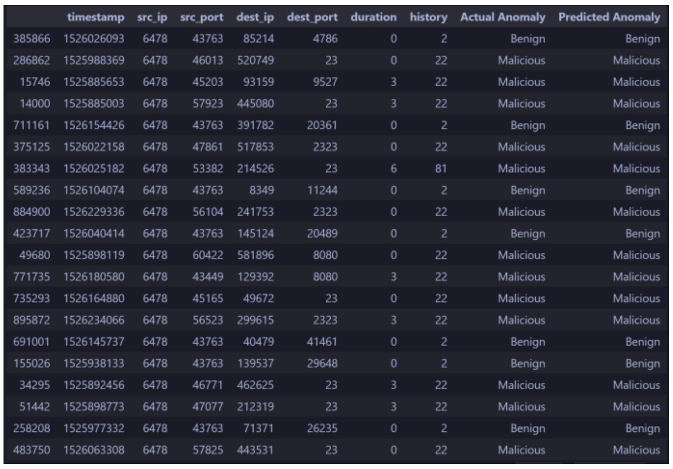
Figure 7.4.1 - Evaluation metrics for all machine learning models.

Chapter 8
PROJECT RESULT & FUTURE WORK

Chapter 8
Project Result & Future Work

8.1 Result

The following figure 11, shows a portion of the first 20 heads which is a function that prints or displays a peak of the dataframe of the actual and predicted labels in the testing dataset with the representation of Benign and Malicious labels.



	timestamp	src_ip	src_port	dest_ip	dest_port	duration	history	Actual Anomaly	Predicted Anomaly
385866	1526026093	6478	43763	85214	4786	0	2	Benign	Benign
286862	1525988369	6478	46013	520749	23	0	22	Malicious	Malicious
15746	1525885653	6478	45203	93159	9527	3	22	Malicious	Malicious
14000	1525885003	6478	57923	445080	23	3	22	Malicious	Malicious
711161	1526154426	6478	43763	391782	20361	0	2	Benign	Benign
375125	1526022158	6478	47861	517853	2323	0	22	Malicious	Malicious
383343	1526025182	6478	53382	214526	23	6	81	Malicious	Malicious
589236	1526104074	6478	43763	8349	11244	0	2	Benign	Benign
884900	1526229336	6478	56104	241753	2323	0	22	Malicious	Malicious
423717	1526040414	6478	43763	145124	20489	0	2	Benign	Benign
49680	1525898119	6478	60422	581896	8080	0	22	Malicious	Malicious
771735	1526180580	6478	43449	129392	8080	3	22	Malicious	Malicious
735293	1526164880	6478	45165	49672	23	0	22	Malicious	Malicious
895872	1526234066	6478	56523	299615	2323	3	22	Malicious	Malicious
691001	1526145737	6478	43763	40479	41461	0	2	Benign	Benign
155026	1525938133	6478	43763	139537	29648	0	2	Benign	Benign
34295	1525892456	6478	46771	462625	23	3	22	Malicious	Malicious
51442	1525898773	6478	47077	212319	23	3	22	Malicious	Malicious
258208	1525977332	6478	43763	71371	26235	0	2	Benign	Benign
483750	1526063308	6478	57825	443531	23	0	22	Malicious	Malicious

Figure 8.1.1 – The actual and predicted of 20 tuples or rows.

The dataset used in this analysis was the testing data, which was obtained by splitting the entire dataset using the `split` function from scikit-learn. This test data was not used in the training process. Instead, it was reserved to evaluate the model's ability to accurately predict anomalies, as indicated by the high accuracy achieved during the training and validation phases. The last two columns of the dataset are necessary for understanding the performance of the binary classification model. The "Actual Anomaly" column contains the true labels of anomalies, which were not provided to the model during prediction. The "Predicted Anomaly" column, on the other hand, contains the model's predictions, derived from the y_pred output of the Deep Learning model, specifically the Gated Recurrent Unit (GRU) model, on the test data split.

1. High Accuracy in Anomaly Detection:

The models developed, particularly Decision Tree and Deep Learning(GRU), demonstrated high accuracy in distinguishing between normal and malicious samples.

For instance, Decision Tree showed high accuracy across all phases of training, validation, and testing, with a strong balance between precision and recall. This high accuracy is indicative of the models' efficiency and their ability to generalize well to unseen data. The models effectively minimized false positives and false negatives, ensuring reliable detection of cybersecurity threats.

2. Handling of Dynamic Threats:

The GRU (Gated Recurrent Unit) model's capability to capture long-term dependencies enabled it to adapt to changing threat landscapes. This adaptability empowered the system to detect emerging threats and anomalous patterns, allowing for active measures to mitigate potential security breaches before they escalated. The GRU model's success underscores the importance of using advanced neural networks in cybersecurity, as they are capable of learning complex temporal patterns that traditional models might miss.

3. Effective Use of Real Datasets:

The system was trained using real datasets from Avast databases and dataset like Wireshark packets data, ensuring that the training data was relevant and effective for detecting anomalies and malicious behavior. This approach helped in obtaining efficient and scalable results for users and entities. By utilizing real-world data, the models were exposed to a variety of threat vectors and network behaviors.

4. Result Summary:

Overall, the developed models, particularly the Decision Tree and Deep Learning (GRU), were reliable and effective in identifying cybersecurity threats within the dataset. The evaluation metrics indicated that these models performed well, although further optimization is needed to reduce false positives and negatives to ensure even greater accuracy and reliability. In contrast, Logistic Regression exhibited poor accuracy due to the nature of the dataset, making it less suitable for cybersecurity packet data. The lower precision and specificity rates of Logistic Regression underscored its limitations in this context.

8.2 Future Work

Moving forward, our project will focus on several key areas to enhance cyber threat detection capabilities. These include:

1. **Automated Response:**

Integrating an automated response mechanism is essential to enhance the system's effectiveness. This feature would enable the system to not only detect threats but also take immediate action to mitigate them. For instance, upon identifying a potential threat, the system could automatically isolate the affected segment of the network,Or alert the user to the presence of a potential threat, block suspicious IP addresses, or initiate a predefined set of countermeasures. This would significantly reduce the response time and minimize the damage caused by cyber-attacks.

2. **Predictive Threat Analysis:**

Move beyond just detection. Explore incorporating threat intelligence and future prediction algorithms to anticipate potential attacks and proactively strengthen defenses.

3. **Real-time API:**

Developing a real-time API would allow seamless integration of the threat detection system with various applications and services. This API could provide real-time threat data to security dashboards, alert systems, and other cybersecurity tools, ensuring that administrators and security professionals are immediately informed of any detected anomalies. Additionally, a real-time API could facilitate the sharing of threat intelligence with other systems, enhancing the overall cybersecurity ecosystem.

4. **Attack-type Classification:**

Enhancing the system to classify the type of attack can provide more detailed insights into the nature of detected threats. By using machine learning algorithms capable of distinguishing between different types of cyber-attacks, such as phishing, DDoS, ransomware, and more, the system can offer tailored responses and remediation strategies. This granularity would enable more precise and effective countermeasures, improving the overall security posture.

5. Anti-Virus Integration:

Integrating the threat detection system with existing antivirus software would create a more comprehensive security solution. By sharing threat intelligence and detection capabilities, the combined system can provide better protection against a wider range of threats. For example, the AI-based anomaly detection system could identify suspicious behaviors that traditional antivirus might miss, while the antivirus can handle known malware signatures, creating a layered defense strategy.

6. User Behavior Analysis:

Incorporating user behavior analysis can enhance the system's ability to detect insider threats and other sophisticated attacks. By monitoring and analyzing user activities, the system can identify deviations from normal behavior patterns that may indicate malicious intent. This proactive approach can help in identifying threats that traditional detection methods might overlook, such as compromised accounts or insider attacks.

7. Adaptive Learning:

Implementing adaptive learning mechanisms would allow the system to evolve continuously based on new data and emerging threats. By incorporating feedback loops and continuous learning models, the system can stay up to date with the latest threat landscape and improve its detection capabilities over time. This would ensure the system remains effective even as attackers develop new methods and strategies.

8. Mobile and IoT Device Security:

Extending the threat detection capabilities to mobile and IoT devices is an important area for future work. With the increasing number of connected devices, ensuring their security is crucial. Developing specialized models and detection techniques tailored to the unique characteristics of these devices can provide better protection against the growing range of cyber threats targeting them.

9. Scalability Enhancements:

Ensuring the system can scale effectively to handle large volumes of data is crucial for its deployment in enterprise environments. Future work could focus on optimizing the algorithms and infrastructure to process data in real-time, maintain high accuracy, and reduce latency. This might involve using distributed computing frameworks, cloud-based resources, and advanced data processing techniques.

10. Zero-Day Threat Protection:

enhancing zero-day threat protection is crucial. This involves advancing our system's capabilities to swiftly identify and respond to threats that are previously unknown or have no established defense. To achieve this, we are focusing on integrating cutting-edge techniques such as anomaly detection, which identifies unusual patterns in network traffic or system behavior that may indicate a zero-day attack. Additionally, behavioral analysis plays a pivotal role by scrutinizing deviations from normal user or system behavior, helping to flag potential threats early. These techniques combined aim to fortify our system against emerging cyber threats, ensuring proactive defense and rapid response capabilities.

These future work directions aim to enhance the capabilities and effectiveness of the AI-based cyber threat detection system, ensuring it remains consistent, adaptive, and comprehensive in the face of evolving cyber threats.

References[4]

[1] Vávra, J., Hromada, M., Lukáš, L., & Dworzecki, J. (2021, May 24). Adaptive anomaly detection system based on machine learning algorithms in an industrial control environment. International Journal of Critical Infrastructure Protection. https://www.sciencedirect.com/science/article/pii/S187454822100038X

[2] Kaur, R., Gabrijelčič, D., & Klobučar, T. (2023, April 7). Artificial Intelligence for cybersecurity: Literature review and future research directions. https://www.sciencedirect.com/science/article/pii/S1566253523001136#fig0003

[3] K. Morovat and B. Panda, "A Survey of Artificial Intelligence in Cybersecurity," 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2020, pp. 109-115, doi: 10.1109/CSCI51800.2020.00026.

[4] Nighania, K. (2019, January 30). Various ways to evaluate a machine learning model's performance. Medium. https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15

[5] Lawton, G. (2023, August 31). What is anomaly detection? everything you need to know. Enterprise AI. https://www.techtarget.com/searchenterpriseai/definition/anomaly-detection

[6] Brownlee, J. (2021, July 6). A gentle introduction to long short-term memory networks by the experts. MachineLearningMastery.com. https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

[7] Saunders, M., Lewis, P., & Thornhill, A. (2019). Research Methods for Business Students. Pearson

[4] We preferred APA 7th edition citation style, we made sure it is applicable on CS field as well.

[8] Gül, R. (n.d.). What is state-of-art artificial intelligence?. Plugger.ai - Plug AI into your apps. https://www.plugger.ai/blog/what-is-state-of-art-artificial-intelligence#:~:text=State%2Dof%2Dthe%2Dart%20means%20the%20%22newest%22,in%20an%20era%20of%20consensus.

[9] Sklearn.svm.oneclasssvm. scikit. (2007). https://scikit - learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

[10] Home. Gated Recurrent Unit Networks. (2014). https://www.geeksforgeeks.org/gated-recurrent-unit-networks/amp/

[11] Joseph, R. (2022, October 18). Grid search for model tuning. Medium. https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e

[12] Srivastava, T. (2024, January 8). *12 important model evaluation metrics for Machine Learning Everyone should know (updated 2023)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

[13] Kostadinov, S. (2019, November 10). *Understanding GRU networks*. Medium. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

[15] The pandas development team. (2020, February). pandas-dev/pandas: Pandas (Version latest). Zenodo. https://doi.org/10.5281/zenodo.3509134

[16] Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

[17] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

[18] Muralidhar, K. (2023, July 7). *Learning curve to identify overfitting and underfitting in machine learning*. Medium. https://towardsdatascience.com/learning-curve-to-identify-overfitting-underfitting-problems-133177f38df5

[19] Twin, A. (2021, October 22). *Understanding overfitting and how to prevent it*. Investopedia. https://www.investopedia.com/terms/o/overfitting.asp

[20] Pandian, S. (2022, October 20). *A comprehensive guide on hyperparameter tuning and its techniques*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/

[21] GfG, T. (2023, December 7). *Hyperparameter tuning*. GeeksforGeeks. https://www.geeksforgeeks.org/hyperparameter-tuning/

[22] Saturn Cloud. (2023, October 30). *How to import python file as module in Jupyter Notebook*. Saturn Cloud Blog. https://saturncloud.io/blog/how-to-import-python-file-as-module-in-jupyter-notebook/

Appendix A
THEORETICAL EXPLANATION OF ESSENTIAL MODELS

Appendix A Theoretical Explanation of Essential Models

A.1: Decision Tree Classification

A.1.1: Overview

Decision tree classification is a popular machine learning technique used for both classification and regression tasks. It operates by recursively partitioning the input space into regions that are homogeneous with respect to the target variable. At each step, the algorithm selects the feature that best splits the data, based on criteria such as Entropy or Gini impurity, to maximize the purity of the resulting subsets. Decision trees are intuitive and interpretable models, as they mimic the human decision-making process by following a tree-like structure of if-else conditions. However, they can suffer from overfitting if not properly pruned or regularized.

A.1.2: Relevance to Anomaly Detection

Decision trees are particularly useful for anomaly detection in scenarios where anomalies exhibit distinct characteristics or can be separated by clear decision boundaries. For example, in network traffic analysis, decision trees can identify unusual patterns or behaviors that may indicate a cyber-attack, such as a distributed denial-of-service (DDoS) attack or port scanning activity. By analyzing features such as packet size, protocol type, and source IP address, decision tree models can flag anomalous network traffic for further investigation.

A.1.3: Technical Details

A.1.3.1: Algorithm Description

The decision tree algorithm operates by selecting the best split at each node based on a metric like Gini impurity or Entropy, which measures the ‘information gain’. Each decision or split point divides the data into subsets that correspond to the most discriminative features for separating different classes, especially distinguishing normal from abnormal patterns. This hierarchical division continues until a stopping criterion is met, which might be a maximum tree depth, a minimum number of samples per leaf, or no further improvement in purity.

A.1.3.2 Feature Importance

Decision trees provide an inherent mechanism for feature importance evaluation by looking at the depth at which a feature is used to split the data and the amount of ‘information gain’ each feature brings. This aspect is particularly valuable in anomaly detection, where understanding which features most strongly correlate with anomalies can provide insights into the nature of the anomalies and inform security measures.

A.1.4 Decision Tree Configuration for Anomaly Detection

A.1.4.1 Hyperparameter Tuning

Important parameters in configuring decision trees for anomaly detection include **max_depth**, which controls the depth of the tree to prevent overfitting; **min_samples_split**, which defines the minimum number of samples required to split an internal node; and **criterion**, which might be set to "**gini**" for Gini impurity or "Entropy" for information gain.

A.1.4.2 Handling Imbalanced Data

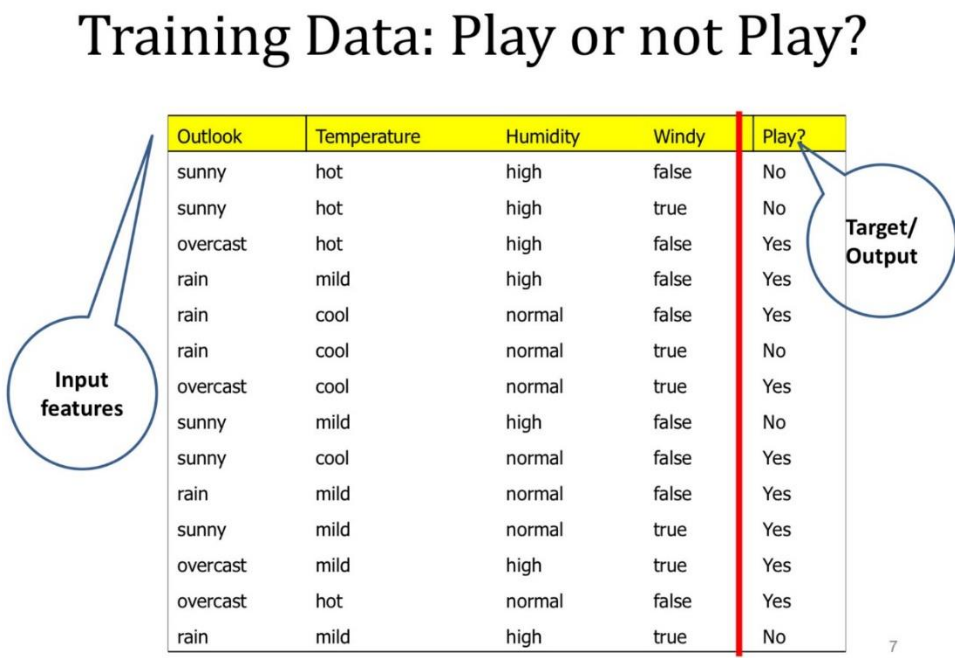
By default, decision tree algorithms do not have intrinsic mechanisms to handle imbalanced data effectively. The primary reason is that these algorithms typically aim to minimize error by focusing on the majority class, potentially overlooking the minority class, which is often the class of interest in anomaly detection scenarios. Here are a few key points on why and how external techniques are employed to address this imbalance:

- **Bias Towards Majority Class:** Decision trees and many other conventional machine learning algorithms are susceptible to bias when trained on imbalanced datasets. They tend to favor the majority class because splitting criteria like Gini impurity or entropy naturally optimize for the most frequent outcomes, leading to poor classification performance on the rare class.

- **Impact on Model Performance:** In the context of anomaly detection, this can result in a high rate of false negatives, where actual anomalies are not detected. This is particularly critical because the cost of missing an anomaly (such as a fraudulent transaction or a network intrusion) can be very high.

In anomaly detection, addressing imbalanced datasets is crucial because such imbalances can lead to biased models that favor the majority class, often neglecting the critical minority class which usually represents the anomalies. Use ***Principal Component Analysis*** **(PCA)** or ***Kernel PCA*** techniques for linear and nonlinear dimensionality reduction respectively to solve high-dimensionality, which can exacerbate the problems caused by an imbalanced dataset and using ensemble methods like ***Random Forests*** or boosting to average predictions or focus on correcting misclassifications.

A.1.5 Implementation Example



Training Data: Play or not Play?

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Figure A.1.5.1 – Decision Tree Feature Explanation.

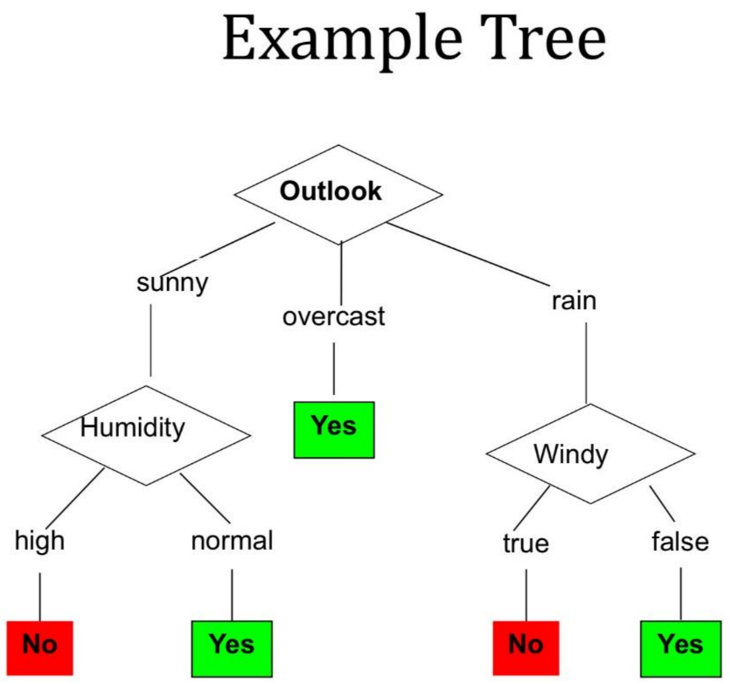


Figure A.1.5.2 – Decision Tree visualization for the given table mentioned in figure A.1.5.

In figure A.1.5.1 and figure A.1.5.2, Several key mathematical concepts are used to build decision trees, particularly concerning how to choose the best split at each node of the tree. Here is one of the methods, explained mathematically:

A.1.5.1 Entropy Calculation

Entropy for a dataset is defined mathematically as:

$$H(S) = -\sum_{i=1}^{n} pi \ \log_2(pi)$$

where:

- ***H(S)*** is the entropy of set ***S***.
- ***pi*** is the proportion (or probability) of the number of elements in class ***i*** to the number of elements in set ***S***.
- ***n*** is the number of classes.

A.1.5.2 Information Gain

Information Gain is the reduction in entropy or surprise by transforming a dataset and is one of the most critical metrics used to form the decision nodes in a decision tree. It is calculated as the difference between the entropy of the parent node and the sum of the entropies of the child nodes — weighted by the proportion of elements that split into the child nodes. The feature that has the highest expected ***IG*** will be chosen as the successor ***Decision Node*** to the ***root*** of the decision tree. Mathematically, Information Gain ***IG*** for a split on a feature ***A*** can be expressed as:

$$IG(S,A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

where:

- ***IG(S,A)*** is the information gain of a set ***S*** when split using feature ***A***.
- ***Values(A)*** are the distinct values of feature ***A***.
- ***/Sv/*** is the number of elements in subset ***Sv***, which results from splitting ***S*** by value ***v*** of feature ***A***.
- ***H(Sv)*** is the entropy of subset ***Sv***.

A.2 XGBoost Classification

A.2.1: Overview

XGBoost, short for Extreme Gradient Boosting, is a highly efficient implementation of gradient boosting that has been designed to be both computationally efficient and highly effective. It uses a scalable and flexible framework that supports various regression and classification problems, including the handling of large-scale and high-dimensional data. XGBoost achieves its efficiency and performance through systems optimizations and algorithmic enhancements such as parallel processing, tree pruning, handling sparse data, and regularization.

A.2.2: Relevance to Anomaly Detection

In anomaly detection, the ability to efficiently process large volumes of data and effectively differentiate between 'normal' and 'anomalous' is crucial. XGBoost's robustness against overfitting—thanks to its built-in regularization (both L1 and L2), and its effectiveness in sparse scenarios where anomalies are a minority, makes it well-suited for these tasks. Its model-building speed and performance allow it to handle the typically vast datasets involved in anomaly detection, such as transaction records, network traffic, or sensor data streams.

A.2.3: Technical Details

A.2.3.1: Algorithm Description

XGBoost (eXtreme Gradient Boosting) is designed around the gradient boosting framework, but with significant improvements in terms of speed, efficiency, and performance. The following techniques enable the XGBOOST algorithm to work as it is right now:

- **Ensemble Learning:** XGBoost builds an ensemble of decision trees in a sequential manner. Each tree is added one at a time, and each new tree aims to correct the mistakes made by the previously built trees. This process continues until a specified number of trees have been added or no further improvements can be made.

- **Gradient Boosting Framework:** The core of the XGBoost algorithm is the gradient boosting decision tree (GBDT). In GBDT, each new tree is fit on the residual errors (gradients) of the preceding trees. Essentially, each tree attempts to predict the residuals of the previous trees. Then, the predictions from all trees are summed together to make the final prediction.

- **Regularization:** Unlike standard gradient boosting, XGBoost incorporates a regularization term (L1 and L2 regularization) in its loss function, which helps reduce overfitting and improves model generalization. This regularization term penalizes complex models (those with more parameters) and helps simplify the overall learned model, enhancing its performance on unseen data.

- **Optimized Split Finding Algorithm:** XGBoost uses a quantile sketch algorithm to propose possible split points efficiently, based on percentiles, which helps handle different data distributions effectively. It also uses an approximate greedy algorithm to find the best split. Here, candidate splits are evaluated by computing the gain from making a split, taking both the loss reduction and the regularization into account.

A.2.3.2 Feature Handling

- **Handling Numerical and Categorical Variables**

  XGBoost can handle both numerical and categorical variables directly. For categorical variables, it does not require one-hot encoding as many other algorithms do; it can manage them internally by converting them into numerical representations, which reduces memory footprint and speeds up computations.

- **Missing Data:**

  During training, XGBoost assigns a default direction for missing values at each node in the decision trees. It chooses the direction (either go to the left or right child of a node) that best minimizes the loss, effectively learning the best way to handle missing data during the training process. This means that when a prediction is made for a data point with missing values, it will traverse the trees using the paths specifically learned for handling absences, which might be different across different nodes and trees.

- **Sparse-aware Implementation**

  XGBoost is optimized to handle sparse data, arising either from missing values, zero entries, or encoding of categorical variables. Its data structure and algorithm can efficiently process sparse data without wasting computations and memory, which is highly advantageous when dealing with high-dimensional data with many zeros or missing values.

A.2.2 XGBoost Configuration for Anomaly Detection

A.2.2.1 Hyperparameter Tuning

The parameters will be explained as follows:

- **Practical Implications of ‘max_depth’**

  The ‘max_depth’ parameter sets the maximum depth of the trees in the model. This means it limits the number of splits that can be made from the root node to the leaf node.

  1. **Controlling Model Complexity**

     - **Shallower Trees:** This prevents the model from capturing too much detail from the training data, reducing the risk of overfitting. However, it might not capture complex patterns adequately.

     - **Deeper Trees:** Allows the model to capture more complex relationships in the data. However, this also increases the risk of overfitting, as the model may learn noise and specific details of the training data that do not generalize well to new data.

  2. **Balancing Bias and Variance**

     - **Bias:** A low ‘max_depth’ might result in underfitting, where the model is too simple to capture the underlying patterns in the data (high bias).

     - **Variance:** A high ‘max_depth’ might lead to overfitting, where the model captures noise and specific patterns in the training data that do not generalize to new data (high variance).

- **Practical Implications of ‘min_child_weight’**

  The ‘min_child_weight’ parameter sets the minimum sum of instance weight (hessian) needed in a child. It essentially defines the minimum number of samples required to create a new node in the tree.

1. **Preventing Overfitting**

   - **Higher Values:** Setting hyperparameter to a higher value makes the algorithm more conservative. It ensures that nodes are not split unless there is a sufficient number of instances. This can prevent the model from learning patterns that are too specific to the training data (overfitting).

   - **Lower Values (non-negative value):** A lower value allows the tree to split even when there are relatively few instances, potentially capturing more detailed patterns but also increasing the risk of overfitting.

2. **Handling Imbalanced Data**

   In anomaly detection, where anomalies are rare, setting a higher 'min_child_weight' can help in ensuring that splits are made only when there is enough data, thereby preventing the model from focusing too much on minority classes with very few samples.

- **Practical Implications of 'eta (learning rate)'**

The 'eta' parameter, also known as the learning rate, scales the contribution of each tree added to the model. It controls how quickly the model adapts to the training data.

1. Controlling the Learning Process:

- **A lower learning rate (e.g., 0.01):** Makes the learning process slower and more gradual. This helps in building a more robust model by allowing the addition of more trees (iterations), each contributing slightly. It reduces the risk of overfitting as the model generalizes better to unseen data.
- **A higher learning rate (e.g., 0.3):** Speeds up the learning process, but each tree's contribution is larger, which can lead to overfitting if not controlled properly.

2. Combining with Other Parameters:

When using a lower ‘eta’, it is common to increase the number of trees built (higher n_estimators value) to ensure the model has enough capacity to learn. Conversely, with a higher ‘eta’, fewer number of trees built (lower n_estimators value) might be needed.

- **Practical Implications of using ‘subsample’**

  The ‘subsample’ parameter in XGBoost controls the fraction of the total dataset to be randomly sampled and used to train each individual tree. Its value between 0 and 1.

  - Setting it to 1 means that all training data is used to build each tree before each boosting iteration.
  - Setting it to a value less than 1, say 0.5, means that each tree is built using 50% of the training data randomly sampled for each round of boosting.

- **Practical Implications of using ‘colsample_bytree’**

The colsample_bytree parameter in XGBoost specifies the fraction of features (columns) to be randomly sampled and used to train each individual tree in the model. It is a value between 0 and 1.

  - **Setting it to 1**: Means that all features are used for training each tree.
  - **Setting it to a value less than 1**: Means that only a subset of the features is used for each tree.

A.2.3 Example on How Tuning Can Improve the Model’s Performance.

For instance, in tackling a binary classification issue where your model is overfitting, consider reducing the max_depth, boosting the subsample, and lowering the colsample_bytree. This approach will simplify the model, reducing the likelihood of overfitting.

Conversely, if your model is underfitting, enhance its complexity by increasing the max_depth, reducing the subsample, and boosting the colsample_bytree. These adjustments can help the model better discern the underlying patterns in the data.

If the model isn't converging, lowering eta (learning rate) might help. This change slows down the updates but can make them more precise.

Lastly, if the model's performance on the test set is unsatisfactory, increasing the number of trees n_estimators might enhance its accuracy and overall performance.

```
# Define the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(objective='multi:softmax',
learning_rate= 0.01, max_depth = 3, reg_alpha = 0.01, reg_lambda =
0.01, num_class=3, seed=42)

# Train the classifier on the training data
xgb_classifier.fit(X_train, y_train)

# Save the XGBoost Classifier Model in dumb file to avoid retraining
process.
(It is used in ThreatDetection.ipynb).
dump(xgb_classifier,
'TrainingDumpData/MachineLearning/XGBoost_Model.joblib')

# This model name is used to ease the visualization process. Change it
for each model.
model_name = 'XGBoost'

# Load XGBoost model, where it is already trained from
'ModelTraining.ipynb' code file.
xgb_classifier =
load('TrainingDumpData/MachineLearning/XGBoost_Model.joblib')

# Make predictions on the testing data
y_pred = xgb_classifier.predict(X_test)
```

A.3 Gated Recurrent Unit (GRU)

In the realm of deep learning, recurrent neural networks (RNNs) have revolutionized our capabilities across diverse domains such as natural language processing, time series analysis, and cybersecurity. One notable variant of RNNs, the Gated Recurrent Unit (GRU), has emerged as a formidable tool for understanding sequential patterns and dependencies within data.

GRU, a type of neural network architecture, is purpose-built to tackle the challenges associated with processing sequential data while overcoming the vanishing gradient problem often encountered in standard RNNs. Its adaptive gating mechanisms facilitate efficient memory management and the acquisition of temporal dependencies, rendering it highly effective for tasks involving time series data and sequential events.

The adaptive nature of GRU allows it to dynamically adjust its processing based on the input data, enabling it to capture long-term dependencies and adapt to evolving contexts. This capability makes GRU well-suited for applications in cybersecurity, where the detection of anomalous behavior and emerging threats requires a nuanced understanding of sequential patterns in data.

Overall, GRU represents a significant advancement in deep learning techniques, offering enhanced capabilities for modeling and understanding sequential data, and contributing to the development of more sophisticated solutions in cybersecurity and beyond.

Advantages of GRU:

- **Efficient Memory Management:**
    - GRU offers streamlined memory management compared to LSTM (Long Short-Term Memory) networks, resulting in a simpler architecture with fewer parameters to train. This simplicity translates to faster training times and reduced computational resource requirements, making it more efficient.
- **Efficient Learning of Temporal Patterns:**

  - In cybersecurity, threats often unfold as sequences of events over time, such as network activity logs or system events. GRU excels at capturing these temporal patterns within sequential data, enabling our model to discern and comprehend the evolving nature of cyber threats effectively. By accurately detecting patterns in time-series data, GRU enhances our ability to identify and mitigate potential security risks.

- **Adaptability to Dynamic Threat Landscapes:**
  - The landscape of cyber threats is constantly evolving, necessitating adaptive and flexible detection mechanisms. GRU's capability to capture long-term dependencies and learn from past events enables our model to adapt to changing threat landscapes. This adaptability empowers us to detect emerging threats and anomalous patterns, allowing for proactive measures to mitigate potential security breaches before they escalate.
- **Reduced Risk of Vanishing Gradient:**
  - GRU addresses a common challenge encountered in training deep neural networks known as the "vanishing gradient" problem. By incorporating reset and update gates, GRU dynamically decides whether to retain or update information during backpropagation, preventing gradients from becoming too small. This mitigates the risk of information loss during training, resulting in more effective and stable training processes for deep learning models.

A.3.1 Key Components of GRU:

The GRU unit comprises essential components that facilitate its ability to learn temporal sequences of data and comprehend context. These components work together to enable effective processing of sequential data. Here's a detailed overview of the key elements of the GRU unit and their functionalities:

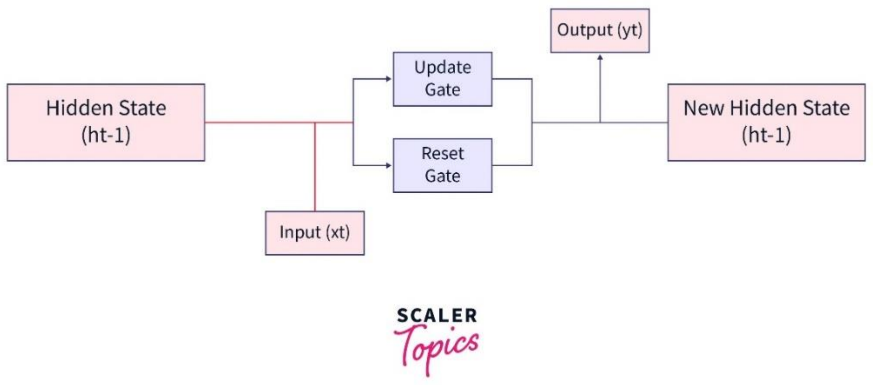


Figure A.3.1.1 – GRU Neural Network Model.

Update Gate (z):

The update gate plays a crucial role in determining how much of the previous information should be retained and passed to the next unit. It computes a value between 0 and 1, which represents the proportion of new information relative to the previous information. When this value is closer to 0, it indicates a preference for relying on the previous information, whereas a value closer to 1 suggests a higher reliance on new information.

Reset Gate (r):

In contrast to the update gate, the reset gate determines the degree to which the model should depend on previous information. A value close to 0 prompts the unit to discard most of the previous information, prioritizing the incorporation of new data. Conversely, a value close to 1 allows the unit to retain a significant portion of the previous information and rely on it more heavily.

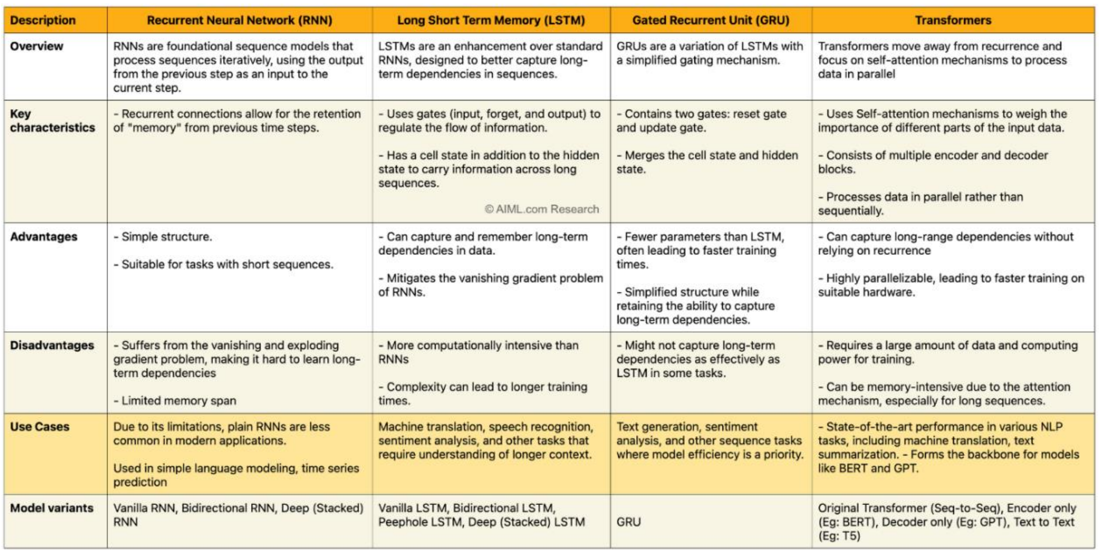
Hidden State:

Following the computation of the update gate and reset gate values, the hidden state of the unit is updated. This process involves blending the previous hidden state with the candidate hidden state, guided by the values of the update gate. By dynamically adjusting the hidden state based on these gate values, the unit effectively updates its memory and adapts to the current input.

Candidate Hidden State:

The candidate hidden state represents the internally updated information that may potentially become the new hidden state. It is computed based on the current input and the previous hidden state, with adjustments made according to the reset gate value. This candidate hidden state, combined with the previous hidden state, contributes to the calculation of the updated hidden state, enabling the unit to capture relevant information for subsequent processing.

Through these interconnected components, GRU units empower the network to dynamically engage with sequential data and adapt to context based on both previous states and new data inputs. This capability enhances the model's ability to learn and understand complex temporal patterns, making it highly effective in tasks involving sequential data processing, such as cybersecurity threat detection.



Description	Recurrent Neural Network (RNN)	Long Short Term Memory (LSTM)	Gated Recurrent Unit (GRU)	Transformers
Overview	RNNs are foundational sequence models that process sequences iteratively, using the output from the previous step as an input to the current step.	LSTMs are an enhancement over standard RNNs, designed to better capture long-term dependencies in sequences.	GRUs are a variation of LSTMs with a simplified gating mechanism.	Transformers move away from recurrence and focus on self-attention mechanisms to process data in parallel
Key characteristics	- Recurrent connections allow for the retention of "memory" from previous time steps.	- Uses gates (input, forget, and output) to regulate the flow of information. - Has a cell state in addition to the hidden state to carry information across long sequences. © AIML.com Research	- Contains two gates: reset gate and update gate. - Merges the cell state and hidden state.	- Uses Self-attention mechanisms to weigh the importance of different parts of the input data. - Consists of multiple encoder and decoder blocks. - Processes data in parallel rather than sequentially.
Advantages	- Simple structure. - Suitable for tasks with short sequences.	- Can capture and remember long-term dependencies in data. - Mitigates the vanishing gradient problem of RNNs.	- Fewer parameters than LSTM, often leading to faster training times. - Simplified structure while retaining the ability to capture long-term dependencies.	- Can capture long-range dependencies without relying on recurrence - Highly parallelizable, leading to faster training on suitable hardware.
Disadvantages	- Suffers from the vanishing and exploding gradient problem, making it hard to learn long-term dependencies - Limited memory span	- More computationally intensive than RNNs - Complexity can lead to longer training times.	- Might not capture long-term dependencies as effectively as LSTM in some tasks.	- Requires a large amount of data and computing power for training. - Can be memory-intensive due to the attention mechanism, especially for long sequences.
Use Cases	Due to its limitations, plain RNNs are less common in modern applications. Used in simple language modeling, time series prediction	Machine translation, speech recognition, sentiment analysis, and other tasks that require understanding of longer context.	Text generation, sentiment analysis, and other sequence tasks where model efficiency is a priority.	- State-of-the-art performance in various NLP tasks, including machine translation, text summarization. - Forms the backbone for models like BERT and GPT.
Model variants	Vanilla RNN, Bidirectional RNN, Deep (Stacked) RNN	Vanilla LSTM, Bidirectional LSTM, Peephole LSTM, Deep (Stacked) LSTM	GRU	Original Transformer (Seq-to-Seq), Encoder only (Eg: BERT), Decoder only (Eg: GPT), Text to Text (Eg: T5)

Figure A.3.1.2 – Differences between RNN, LSTM, and GRU models.

Appendix B
EVALUATION METRICS EXPLANATION

Appendix B
Evaluation Metrics Explanation

True Positive (TP): This occurs when the prediction and the reality are both positive. In other words, the model correctly predicts the positive class. For example, in medical testing, a true positive would be if a test correctly identifies a disease that a patient actually has.

False Positive (FP): This happens when the prediction is positive, but the reality is negative. The model incorrectly predicts the positive class. In terms of a medical test, this would be a scenario where the test indicates a patient has a disease when they actually do not. This is also known as a "Type I error."

True Negative (TN): This occurs when both the prediction and the reality are negative. The model correctly predicts the negative class. For example, if a medical test shows that a patient does not have a disease and they indeed do not have it, that's a true negative.

False Negative (FN): This happens when the prediction is negative, but the reality is positive. The model incorrectly predicts the negative class. In a medical context, this would mean the test fails to detect a disease that the patient actually has. This is also referred to as a "Type II error."

1. **Accuracy:** this factor tells us how often our model is right. Its formula as following: $\frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$.
2. **Precision:** it is used to determine the ability of the model to classify positive values correctly. It tells us when the model predicts a positive value and how often it is right. Its formula: $\frac{\text{TP}}{\text{TP+FP}}$.
3. **Recall:** it is used to determine how often correct positive values are captured. Its formula as following: $\frac{\text{TP}}{\text{TP+FN}}$.
4. **F1-Score:** this is used when we want to take both precision and recall in our consideration. Its formula as following: $\frac{2*\text{Precision}*\text{Recall}}{\text{Precision+Recall}}$.
5. **Sensitivity (True Positive Rate)**:

- Sensitivity measures the proportion of actual positives that are correctly identified by the model.
- It is also known as the True Positive Rate (TPR).
- Sensitivity is calculated using the following formula:

  $\boldsymbol{Sensitivity}\ (\boldsymbol{TPR}) = \frac{TP}{(TP+FN)}$ Where:

  - ***TP*** **(True Positives)** is the number of instances that are positive and are correctly classified as positive by the model.
  - ***FN*** **(False Negatives)** is the number of instances that are positive but are incorrectly classified as negative by the model.
- Sensitivity is expressed as a percentage, and a higher sensitivity indicates that the model is better at correctly identifying positive instances.

6. **Specificity (True Negative Rate)**:

- Specificity measures the proportion of actual negatives that are correctly identified by the model.
- It is also known as the True Negative Rate (TNR).
- Specificity is calculated using the following formula:

  $\boldsymbol{Specificity}\ (\boldsymbol{TNR}) = \frac{TN}{(TN+FP)}$ Where:

  - ***TN*** **(True Negatives)** is the number of instances that are negative and are correctly classified as negative by the model.
  - ***FP*** **(False Positives)** is the number of instances that are negative but are incorrectly classified as positive by the model.
- Specificity is expressed as a percentage, and a higher specificity indicates that the model is better at correctly identifying negative instances.

It is worth noting that in anomaly detection, **True Positives** are more important than **True Negatives**, and the effect of **False Positives** are less dangerous than **False Negatives.** Overall, achieving a balance between true-positive detections for effective threat identification and true-negative identifications for minimizing false alarms is important for optimizing the performance and reliability of the anomaly detection system.