Prevention of Cyber-Attack Using Intrusion Detection System With Machine Learning Algorithm

Ayush Ale

Submitted in accordance with the requirements for the degree of
Master of Science (Level 7)

York St John University

Department of Computer Science

Supervisor's Name: Dr. Soonleh Ling

Date of Submission: 28th Jan 2025

Abstract

In present modern society, digital security or cyber security is very important for protecting critical property and sensitive data. An Intrusion Detection System (IDS) is specially created to monitor network or system activities for any breach policy violation or malicious actions about standard operations. However, the complexity of attacks is increasing, which makes it harder to monitor with IDS alone. This research dives into integrating intrusion detection systems (IDS) with machine learning algorithms to increase security awareness and prevent cyber-attacks. The focus of this project is to gather information about IDS, its limitations in identifying complicated cyber-attacks, and its combination with machine learning algorithms for enhanced performance. Detailed info on the current cybersecurity environment, intrusion detection systems, and machine learning ideas are covered to achieve our determined goal.

Five machine learning models — logistic regression, decision trees, random forest, support vector machines, and K-Nearest Neighbors — are used to create a network intrusion detection prediction model. The dataset that the driving network traffic collects comes from an internet source. Different factors are used to determine which model is the best, heat map, false positive rate, confusion matrices, precision, f1 score including accuracy. With these prediction model results, we can determine which is good for the network intrusion detection prediction model.

Acknowledgments

I would very much like to express my sincere gratitude to my lecturers at York St. John University for their invaluable guidance and support throughout my dissertation journey. I am particularly indebted to our module leader, Amjad Alam Sir, whose expertise and encouragement were instrumental in shaping my research. I would also like to extend my heartfelt thanks to my supervisor, Dr. Soonleh Ling, for his unwavering belief in my abilities and her dedication to helping me achieve my academic goals.

I would like to express my sincere gratitude to everyone listed above, whose support made this dissertation a reality.

Ayush Ale

28th Jan 2025

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

In this chapter starting process of the project is covered along with background information, the project's overall goals, issues and answers, introduction to the Intrusion Detection System (IDS).

Intrusion Detection System in cyber-security networks

An Intrusion Detection System (IDS) is a security mechanism used to monitor computers and their networks. It helps us to detect any suspicious behavior in computer systems or networks. It acts as an alarm system between the internet and the internal network system. IDS is a security software that sits between your computer and the internet and protects and alerts you from any harmful attacks. It monitors network traffic or system activities for suspicious actions, policy violations, or other security breaches. There are two types of IDS systems: Network-based Intrusion Detection System (NIDS) and Host-based Intrusion Detection System (HIDS). Intrusion Detection System looks over every activity that's happening in the system and the network and looks for abnormal activities. It alerts the security personnel if any abnormal activity is detected during monitoring. NIDS analyses incoming and outgoing traffic across the entire network for suspicious activities different from standard activity. On the other hand, HIDS works along with the host systems or devices, monitoring system logs, finding corrupted files, and application activity for any unusual behaviors. Both types of IDSs can use signature-based detection, which is one of the most direct and well-known methods of identifying malicious activity. Signature-based detection monitors network traffic, compares it with well-known signatures, and generates an alert when the match is found.

The Intrusion Detection System (IDS) plays an important role as it monitors everything within your network. IDSs analyze network traffic while effectively comparing it against a huge database of identified malicious attack patterns. This database may include unauthorized login attempts, attempts to exploit vulnerabilities or stealing of data from personal or corporate devices. If IDS detects any activity that is out of the ordinary and matches any known attack signature, it sets a red flag which alerts security admins for further monitoring. With continuous monitoring and analysis of the network, IDSs provide corporations with an important advantage, the power to identify and respond to potential attacks before they can do much harm. This early warning system allows security teams to take immediate action, patching vulnerabilities, blocking unauthorized access attempts, or containing an attack before it can harm and compromise sensitive data. IDSs play an important defense layer that protects your system from a wide range of threats lurking in the digital shadows.

Along with intrusion detection systems for robust security firewall is also an important factor. A firewall is a network security device, which can be found in hardware or software-based, whose job is to monitor every incoming and outgoing traffic based on a specific defined set of rules it accepts, rejects, or drops specific packets. Accept means it allows traffic to flow, reject in the sense of blocking the traffic with an unreachable error, and drop means it blocks traffic and doesn't reply to anything. It is essentially the wall that separates the outside network from the private internal network.

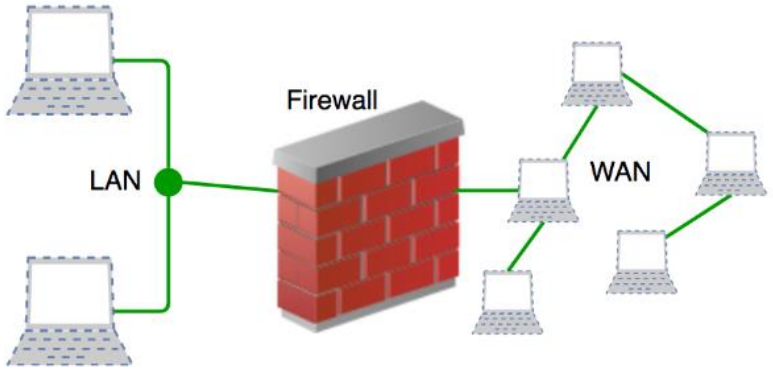


Figure 1: Firewall between networks

Working on an Intrusion Detection System along with a firewall in the network

Intrusion Detection System (IDS) works by monitoring network or system to identify suspicious activity or policy violations. It functions like an alarm system that watches over incoming and outgoing data and systems for suspicious activity. When it detects some suspicious activity, it generates an alert and informs security personnel about that incident. IDS collects info about the date the alert was created, info on potential danger, the nature of that incident, and possible action to be taken. Alerts are also categorized by danger level. For monitoring activity of system and network IDS gets logs from different sources. For network-based IDS, it takes logs from network traffic and for host IDS it gets logs from the host system. After the collection of data, it filters them into standard format while combining them with different sources to get complete data.

The graphic below shows different types of IDS along with the location of IDS placed in different types of IDS:

- **Network-based IDS (NIDS):** NIDS is placed in between the internal network and the firewall. If attacks slip from the firewall, they can be intercepted by IDS. It collects data packets across the network and examines these packets for malicious activity.

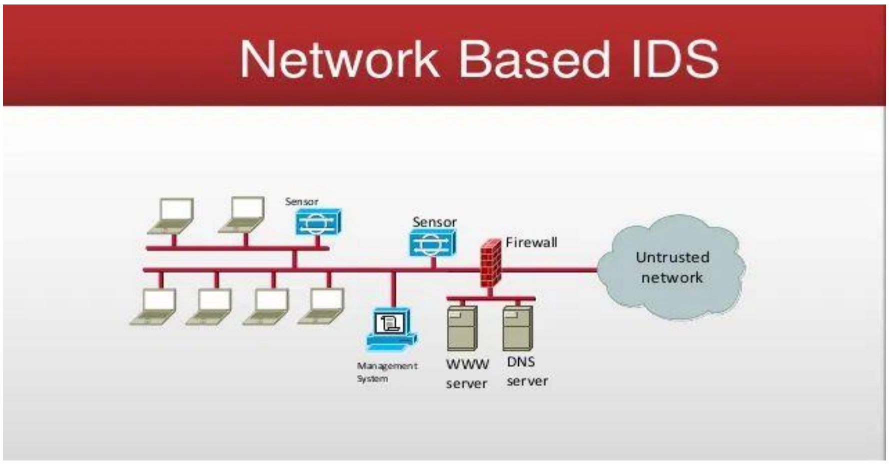


Figure 2: Network-based IDS

Source: (Cyberhoot, 2020)

- **Host-based IDS (HIDS):** Host-based IDS (HIDS) is placed on the individual host system, which monitors specific hosts for attacks. It tracks logs from different fields in the system looks for suspicious activity and reports to administrators.

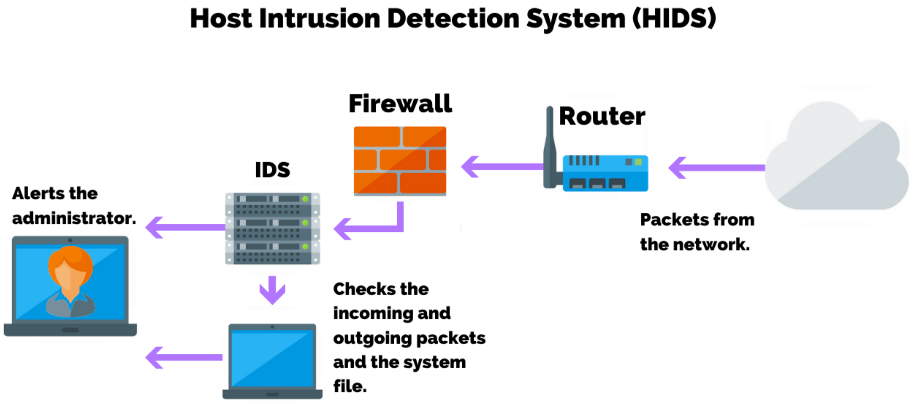

Host Intrusion Detection System (HIDS)
Firewall
Router
IDS
Alerts the administrator.
Packets from the network.
Checks the incoming and outgoing packets and the system file.

Figure 3: Host-based IDS

Source: (purplesec, 2019)

- **Hybrid-based IDS (HYIDS):** Hybrid IDS (HYIDS) use both NIDS and HIDS where NIDS monitors networks, while HIDS investigates specific hosts. Its goal is to merge different systems to achieve better results.

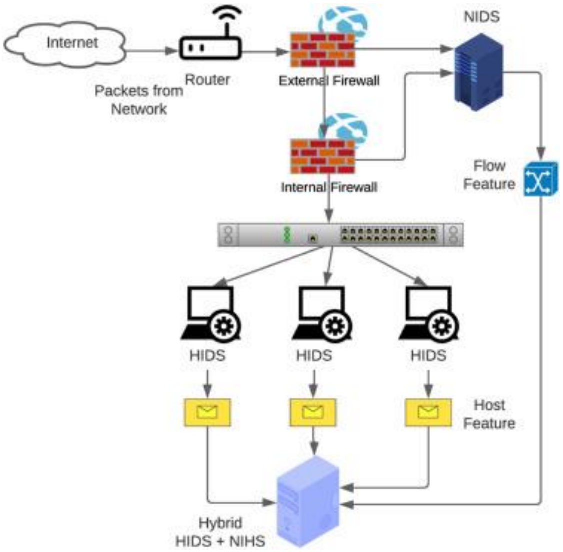


Figure 4: Hybrid-based IDS

Source: (ARS.els, n.d.)

In this way, IDS monitors the system or network to find malware and other questionable activities.

An intrusion detection system is combined with the firewall in every scenario. Firewall acts as a first line of defense before IDS. Firewall is used in many environments for monitoring as well as filtering any incoming or outgoing network traffic in the form of external hardware or internal software. The main goal of a firewall is like IDS in blocking malicious traffic requests and data packets. A firewall is different based on its deployment architectures network-based mostly software, host-based (hardware), and cloud-based.

Firewall along with IDS matches network traffic against a set of predetermined rules which are designed to filter out harmful traffic. These rules consist of patterns of different types of attacks that have happened in history around the world. After the rule is matched it acts for that traffic. For example, Rules state that general employees from the HR department cannot access data from the code server. Rules can be defined in a firewall by looking into the company's needs and requirements.

The firewall maintains separate rules for both incoming and outgoing cases. Most of the traffic reaching the firewall falls under three major transport layer protocols: TCP, UDP, or ICMP. Every packet consists of a source and destination address. TCP and UDP bot consist of their port numbers, while ICMP uses type codes different from port numbers. It is very difficult to cover every aspect of an attack, so a firewall is used in combination with an intrusion detection system.

Machine Learning Algorithm

Machine learning algorithms are computational models that make it possible for computers to understand patterns and make predictions based on the given data. These algorithms are the base for many modern artificial intelligences. It is divided into four types: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised learning includes a training model with known data, where desired output is determined. Algorithm like Logistic Regression, Support Vector Machines (SVM) K-Nearest Neighbors (k-NN) comes under supervised learning. Unsupervised learning works with unlabelled data and tries to find hidden patterns in input data. Algorithm like k-Means, Gaussian Mixture Models (GMM), Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Aprior Algorithms comes under unsupervised learning. Reinforcement Learning focuses on training agents to make different types of decisions rewarding them for good ones and punishing them for bad decisions. Q-learning, Proximal Policy Optimization (PPO), and Temporal Difference (TD) Learning come under reinforcement learning. Ensemble Learning combines multiple models to improve performance by taking good aspects from each model. Bosting (uses decision trees) and boosting (AdaBoost, Gradient Boosting) come under ensemble learning.

Linear regression is a way of finding the linear relationship between a dependent variable and an independent variable. Decision trees split data into subsets based on input characteristics, it looks like a tree in the process. Random forest combines multiple decision trees to create a stronger and more accurate predictive model. Support Vector Machines (SVM) is a powerful classifier that finds a hyperplane separating different classes by maximizing the margin between them. Neural networks use layers of interconnected nodes to create complex patterns in data. It is used for image recognition, speech recognition, and natural language processing. Different types of algorithms are used for different purposes as they cover a specific field in the journey of predicting decisions. To get our desired result we need test data from different sources. The algorithm can create predictions based on supervised learning using pre-determined rules that make it suitable for detecting abnormality.

Aim

For developing a prediction model that can detect network intrusion, understanding how intrusion detection system helps in preventing cybercrime on networks and sampling several models based on performance or assessment criteria.

Objectives

For safeguarding against cyber-attacks implementation of an intrusion detection system (IDS) is essential, and by combining it with machine learning algorithms, it can predict and intercept assaults. To avoid cyberattacks following are the primary goals of intrusion detection systems combining machine learning algorithms:

- Understand the history of intrusion detection systems.
- Comprehend and understand how it protects data.
- Find what intrusion detection system protects the network and what cannot.
- Understand the difference between network and host intrusion detection systems.
- Comprehend the cost of installing an intrusion detection system.
- Build a prediction-making model for a network intrusion detection system using a machine learning algorithm.
- Application of these models in modern world scenarios to identify shortcomings and improve them over time.
- Make sure whether the model performs well on different datasets and if it can be replicated for new data.

Research Questions

To strengthen the theory, it's important to carefully choose study questions. Research questionnaires are mainly used to describe the problem and determine how each consumer perceives it. Additionally, some sub-questions are linked to the main question to explore key aspects of the research problem. These sub-questions help separate evaluation-related issues and highlight important topics. According to McCombes (2019), using sub-questions is essential for improving the structure of the research and can be very helpful when writing a thesis.

The following is a list of the primary research questions for the thesis.

How Intrusion detection system prediction model detect intrusion in a Network?

The following procedures are used by intrusion detection systems (IDS) to detect malicious activity:

- Data Collection: IDS collects data about incoming and outgoing packages in the network traffic.
- Data Preprocessing: Data is filtered, standardized, and divided into information.

- Signature-Based Detection: IDS looks at the traffic packet and compares it with a list of signatures of cyber-attacks. A match may suggest an intrusion.

Why do we need an intrusion prevention system?

Protect your information from unauthorized access: IDS helps prevent unauthorized incoming and outgoing network traffic.

How firewall detect and prevent intrusion or malware from the secure network?

- Packet inspection: Examining incoming and outgoing data packets to spot unknown virus signatures or odd patterns.
- Access control: Enforcing access restrictions to permit or reject traffic according to pre-established standards, barring potentially hazardous connections.
- Deep packet inspection: Preventing attacks at the network level by scanning packet content for malware or harmful payloads.

How Intrusion detection system work to secure an insecure network?

A network is protected by IDS:

- Traffic Analysis and Detection: IDS repeatedly scans network traffic for unusual activity.
- Protecting Against Unauthorized Access: Unsecured networks are often more vulnerable to attacks like hacking or Dos. IDS can identify suspicious access patterns.
- Blocking Malicious Traffic: The IDS system can be integrated with an intrusion prevention system (IPS) to block malicious traffic once the threat is detected automatically.

What are the algorithms that are used for the detection of intrusion on the network?

The most typical intrusion detection methods are:

- Signature-Based Detection: Network traffic patterns are compared with known attack signatures.

- Anomaly-Based detection: Applies statistical analysis to find network behavioral differences from the norm.
- Heuristic-based detection: Makes use of heuristics to identify possibly suspicious behavior and establishes criteria to alert it.
- Machine Learning-Based Detection: ML models are used to identify and learn problematic network behavior patterns.

Which machine learning algorithms have the best accuracy or score for detection?

The optimal machine learning method for intrusion detection may be determined by several variables, but some of them are as follows:

- Random forest
- Logistics Regression
- K-Nearest Neighbours (k-NN)
- Decision Tree

Different algorithms' use may differ based on specific datasets, properties, and current requirements. It is important to test all of them and pick what works best for our specific environment. Intrusion Detection System is complex, so rigorous research and study are required to determine answers to major research questionnaires and their sub-questions. Throughout the investigation, several challenges may arise.

Project Outline

This project has a total of six chapters, which are briefly detailed below.

CHAPTERS	Brief Description
Chapter 1	This chapter consists of an introduction to intrusion detection systems, how they work, as well as the purpose, goals, and research questions.
Chapter 2	This chapter consists of a literature evaluation that dives into earlier studies on related projects and provides critical information on published literature, including books, and research papers.

Chapter3	This chapter covers methodology details on data collection, data analysis technique, and selection criteria for each as well as tools used for this process. Implementation and development of several solutions for determining the result. Each process is elaborated and explained clearly in this chapter.
Chapter 4	This chapter covers results evaluation and comparison. It includes all the experiment’s specifics. An in-depth description of the research study’s key conclusion and insights. The results are compared and represented graphically.
Chapter 5	This chapter discusses an overview of the research effort, flaws, and recommendations for improvement in the future. As a continuation of previous work, it includes future projects.

Table 1: Project outline

Chapter 2: Literature Reviews

Introduction

A literature review is an overview of existing research on a specific topic. It helps identify the strengths and weaknesses of previous studies, points out areas where more research is needed, and suggests new research ideas. This section will focus on reviewing research about how effective intrusion detection systems (IDS) are at preventing cyberattacks on networks.

Existing Study on Intrusion Detection System (ISD)

There have been several studies and histories written about Intrusion Detection since about 1970s. However, neither of the studies focus on group review research that explains IDS technology, nor do they offer a comprehensive scope and depth on the topic. The foundation of IDS was introduced by a seminal report titled "Computer Security Threat Monitoring and Surveillance." Published in 1980 by James P. Anderson (Anderson, 1980). This report laid the foundation for IDS by proposing the use of audit trails to monitor and analyze system activities to detect suspicious behavior. Audit trails are records that track all activities and changes made within a system. The idea of using audit trails for security monitoring began to surface during the 1970s. Audit trails can be used to detect unauthorized access or anomalies within the system. IDS has developed beyond the initial function of monitoring system logs. IDS was initially created to detect malicious activities in the system, but at present it can respond to any threats and real-time analysis of security events.

An IDS's initial version was deployed during the 1980s and it was composed of audit logs that were analyzed for monitoring individual host systems. IDS is one of the most popular technologies used to stop cyberattacks on networks. An IDS is a piece of software that monitors the system for unusual or malicious activities. Several studies are highlighting the impact of Intrusion Detection Systems (IDS) on companies and their effectiveness in preventing cybercrimes. IDSs are critical in detecting and mitigating various types of cyber threats, such as malware, unauthorized access, and Denial of Service (DoS) attacks. A systematic review by Paté-Cornell et al. discusses the probabilistic risk analysis framework for cybersecurity, indicating that effective IDS implementations can significantly reduce losses from cyberattacks by identifying and mitigating threats in real time. This study emphasizes the importance of using historical data and anticipated incident scenarios to improve risk management and decision-making processes for organizations (Paté-Cornell et al., 2018). Another study by Mukhopadhyay et al. illustrates how IDS technology can lower the probability of successful cyberattacks and estimates the financial impact of these attacks on organizations. Their research shows that proactive measures, such as IDS, are crucial for minimizing potential losses and ensuring better preparedness against cyber threats (Mukhopadhyay et al., 2019).

Intrusion Detection Systems (IDS) face several key challenges and can also create some issues themselves. A study by Aljanabi, Ismail, and Ali (2021) explores these challenges and needs in detail. One significant challenge is the scarcity and imbalance of high-quality, labeled datasets that accurately represent real-world attack scenarios, which is critical for training effective machine learning models. (Ali, et al., 2021). Moreover, the dynamic and heterogeneous nature of industrial environments makes it difficult to develop generalized models that can adapt to constant changes. Additionally, real-time processing demands high computational resources, which can be a constraint in resource-limited settings. IDS also faces issues related to explainability and trust. Many machine learning models, particularly deep learning ones, are often seen as black boxes, making it difficult for operators to trust and understand their decisions. This lack of transparency can hinder regulatory compliance and forensic analysis, which require clear explanations of how decisions are made. (Khraisat, et al., 2019).

As a result, the implementation of IDS can introduce problems such as high false positive rates, where normal behavior is incorrectly flagged as malicious. This can lead to alert fatigue, where operators become desensitized to warnings, potentially missing actual threats. The integration of IDS into existing systems can also be complex and costly, requiring significant resources and expertise to manage and maintain effectively. IDS is essential for enhancing security but addressing these challenges and issues is also crucial for their effective implementation and operation in industrial and another complex environments.

The evolution of Intrusion Detection was thoroughly reviewed in a study written by the SANS Institute and published in 2001 (Institue, 2001). This paper highlights the origins and evolution of Intrusion Detection Systems (IDS). It discusses the early methodologies and technologies used in IDS, such as signature-based detection and anomaly detection. The study underscores the initial challenges faced by IDS, particularly high false positive rates and the need for more sophisticated detection mechanisms to keep up with evolving cyber threats. As IDS technologies advance different types of IDS technologies are created as discussed in the article published by Biermann, Cloete, and Venter in 2001 (Biermann, 2001). This article compares various IDS technologies available in the early 2000s. It evaluates these systems based on their detection methods, such as network-based and host-based IDS, and their placement strategies. The study identifies strengths and weaknesses of early IDS, noting that while they were effective in detecting known attack patterns, they often struggled with new or unknown threats. The paper emphasizes the importance of balancing detection accuracy with minimizing false positives. Another study authored by Verwoerd and Hunt in 2002 talks about different intrusion detection techniques, including anomaly detection and signature-based detection (Verwoerd, 2002). They provide a detailed analysis of the advantages and limitations of these approaches. The study highlights the difficulty in creating effective anomaly detection systems due to the need for accurate baselines of normal behavior. Additionally, the paper discusses the limitations of signature-based systems in detecting novel attacks and the necessity for regular updates to signature databases.

Intrusion Detection Systems (IDS) have significantly enhanced cybersecurity by providing improved detection capabilities, regulatory compliance, increased network visibility, effective incident response, and cost savings. According to Dimitri Antonenko on BusinessTechWeekly.com, IDS monitors network traffic for suspicious activities and alerts security teams to potential threats, allowing for early identification and mitigation before significant damage occurs. IDS also helps organizations comply with regulatory requirements and improve their overall security posture by providing detailed insights into network activities and aiding in the identification of vulnerabilities (Antonenko, 2022). However, IDS face challenges such as handling high volumes of false positives, requiring skilled personnel to interpret alerts, and having limitations in preventing sophisticated attacks. Despite these challenges, IDS remains a critical component in modern cybersecurity strategies, offering a proactive approach to threat detection and response that traditional methods lack. (Nicholls, 2017).

Origination and Evolution of Intrusion Detection System

History of the Development of Different Intrusion Detection System

- 1980s – First Generation – Early Concepts and Foundation
- 1990s – Second Generation – Initial Commercialization and Diversification
- 2000s – Third Generation – Hybrid Systems and Enhanced Detection Methods
- 2000s-10s –Fourth Generation - Machine Learning and Real-time Analysis
- 2010s-present – Fifth Generation - Next-Generation IDS and AI Integration

First Generation – Early Concepts and Foundation

In the 1970s, the concept of using audit trails for security monitoring began to take shape, with these trails recording system activities to detect unauthorized access. This idea was further developed in 1980 when James P. Anderson published his foundational report "Computer Security Threat Monitoring and Surveillance," which proposed using audit trails to monitor and analyze system activities to identify suspicious behavior. In 1986, Dorothy E. Denning introduced her influential "Intrusion-Detection Model," which used statistical methods to detect anomalies in system usage patterns, marking deviations from normal behavior that could indicate intrusions. (DENNING, 1987**.** During the 1980s, research and prototyping efforts in IDS primarily focused on Host-Based IDS (HIDS), which monitored individual host systems for suspicious activities by analyzing audit logs and other system data. One of the first notable IDS implementations was the Haystack project, developed for the U.S. Air Force, which used statistical anomaly detection to identify abnormal system usage. This period established the theoretical and practical foundation for anomaly detection in IDS. (Khraisat, 2019).

Second Generation – Initial Commercialization and Diversification

In the 1990s, the development and commercialization of Intrusion Detection Systems (IDS) saw significant advancements. Network-based IDS (NIDS) emerged, such as the UC Davis' Network Security Monitor, which focused on analyzing network traffic for suspicious patterns, marking the start of network-focused intrusion detection. (Wallen, 2020). Concurrently, Host-based IDS (HIDS) systems were developed to monitor individual hosts for signs of compromise through log analysis and system call monitoring, complementing network-based approaches. This period also saw the release of commercial IDS products by companies like Internet Security Systems (ISS), which introduced solutions like Real Secure that integrated both network and host-based detection capabilities to offer comprehensive security. (Khraisat, 2019). The rise of the internet and computer networks further propelled the development of NIDS, exemplified by the release of the open-source NIDS Snort in 1998, which gained widespread adoption due to its flexibility and effectiveness. This decade also witnessed the introduction of hybrid IDS solutions, combining both network-based and host-based approaches to enhance detection capabilities. These advancements laid a robust foundation for the evolution of IDS technology into a critical component of cybersecurity strategies. (Annals, 2017).

Third Generation – Hybrid Systems and Enhanced Detection Methods

The 2000s marked significant advancements in Intrusion Detection Systems (IDS) through the development of hybrid systems and enhanced detection methods. Hybrid IDS, which combined network-based and host-based approaches, provided broader security coverage and were more effective at detecting a wider range of threats. Signature-based detection methods utilized databases of known attack signatures, allowing IDS to quickly identify and respond to specific, known threats, thus improving the speed and accuracy of detection (Gala, 2023). Concurrently, anomaly-based detection methods continued to develop, enabling IDS to identify previously unknown threats by comparing current behavior against established baselines. This era also saw the emergence of Intrusion Prevention Systems (IPS), which built upon IDS technology by not only detecting but also actively preventing and blocking intrusions, such as terminating malicious connections in real time. (Khraisat, et al., 2019). The integration of both IDS and IPS capabilities in hybrid solutions further enhanced comprehensive security coverage, combining the strengths of host-based and network-based detection to provide robust intrusion detection and prevention mechanisms.

Fourth Generation - Machine Learning and Real-time Analysis

In the 2010s, Intrusion Detection Systems (IDS) advanced significantly with the integration of machine learning and real-time analysis. Machine learning techniques improved IDS accuracy by enabling systems to learn from large datasets and adapt to new and evolving threats, making them more effective and responsive. (Khraisat, et al., 2019). Advanced behavioral analysis techniques further enhanced IDS capabilities by enabling a deeper understanding of user and system behavior, improving the detection of sophisticated and evolving threats. Enhancements in processing power and algorithms allowed IDS to analyze data in real time, providing quicker detection and response to potential threats, which became critical for modern cybersecurity needs. (Sarker, 2021). Additionally, the move to cloud environments led to the development of virtual firewalls that operate within virtualized environments and scale with cloud infrastructure. These firewalls integrated with broader security ecosystems, including Security Information and Event Management (SIEM) systems, threat intelligence platforms, and advanced malware protection, enhancing overall threat detection and response capabilities. (Gala, 2023). The incorporation of machine learning and artificial intelligence into IDS provided advanced algorithms for detecting complex and previously unknown threats, while User and Entity Behaviour Analytics (UEBA) focused on identifying anomalies in user and entity activities, detecting insider threats, and sophisticated attacks based on behavioral patterns.

Fifth Generation - Next-Generation IDS and AI Integration

In the 2020s, Intrusion Detection Systems (IDS) have evolved to incorporate advanced features such as threat intelligence integration, automated response mechanisms, and deep learning, significantly enhancing detection accuracy and reducing false positives.

(Zhang, et al., 2024). The integration of artificial intelligence (AI) has allowed IDS to continuously learn from new data, predict potential threats, and adapt to changing attack patterns, providing sophisticated analysis and handling complex, multi-vector attacks more effectively. (García, et al., 2012).

IDS has become a critical component of Security Information and Event Management (SIEM) systems, enabling real-time analysis and correlation of security events across an organization's IT infrastructure. This integration facilitates a comprehensive approach to threat management. Automated response systems have also become more common, allowing for faster and more effective mitigation of threats. (Arreche, 2024).

Additionally, IDS technology has adapted to secure cloud services and Internet of Things (IoT) devices, addressing unique challenges such as scalability and the diversity of connected devices. These systems are now designed to protect diverse and dynamic infrastructures, ensuring robust security in modern environments. Behavioral analysis and User Entity Behaviour Analytics (UEBA) are also integrated into modern IDS, focusing on the behavior of users and entities to identify anomalies that traditional signature-based methods might miss, further enhancing security capabilities. (Arreche, 2024).

Understanding Intrusion Detection System (IDS)

Types of Intrusion Detection System

- **Network-based IDS (NIDS):** NIDS monitors network traffic for suspicious activity by analyzing incoming and outgoing packets (Annals, 2017). Positioned at strategic points within the network to monitor traffic to and from all devices. Example: Snort.
- **Host-based IDS (HIDS):** HIDS monitors a single host for suspicious activity by analyzing events within that specific system, such as file changes, system logs, and other host-specific activities. Example: OSSEC.
- **Hybrid IDS:** Combines elements of both NIDS and HIDS to provide comprehensive monitoring and detection capabilities (Gala, 2023). Offers broader security coverage by integrating network and host-based approaches.

Detection Methods

- **Signature-based Detection**: Uses predefined patterns (signatures) of known threats to detect and identify malicious activities (Robinette, 2024). Highly effective against known attacks but less effective against new, unknown threats.
- **Anomaly-based Detection**: Establishes a baseline of normal behavior and then detects deviations from this baseline that could indicate potential threats. Effective in identifying zero-day attacks and novel threats that do not match any known signatures (NETWORKS, n.d.).
- **Heuristic-based Detection**: Uses algorithms to identify suspicious behavior that may not conform to normal activity or known attack patterns. Can be more adaptive and capable of detecting previously unseen threats (Robinette, 2024).

Intrusion Detection System Functions

- **Monitoring and Analysis**: Continuously monitors network or system activities to detect suspicious actions. Analyzes data in real-time or near-real-time to identify potential threats (Antonenko, 2022).
- **Alerting**: Sends alerts to administrators or security personnel when potential threats are detected. Alerts can be configured to trigger based on specific criteria or severity levels.
- **Logging**: Logs all detected events for further analysis and forensic investigation. Provides a historical record of all security-related activities and potential incidents (Ali, et al., 2021).
- **Prevention and Response**: While traditional IDS are primarily passive and do not act against detected threats, modern systems often integrate with Intrusion Prevention Systems (IPS) to actively block or mitigate threats.

Challenges faced by Intrusion Detection System

- **False Positives**: One of the main challenges of IDS is the high number of false positives, where legitimate activities are incorrectly flagged as malicious. This can overwhelm administrators and reduce the effectiveness of the system (Nicholls, 2017).

- **Performance Impact**: Monitoring and analyzing network traffic or system activities can be resource-intensive, potentially impacting system performance.
- **Evasion Techniques**: Attackers continuously develop techniques to evade detection by IDS, such as encryption, fragmentation, and obfuscation of malicious payloads.

Machine Learning Based Intrusion Detection System (IDS)

Intrusion Detection System (IDS) based on machine learning can predict and detect potential cyber-attacks in computer networks. These systems collect data from network traffic, system logs, and other sources throughout the network and look for any out-of-the-ordinary patterns. With the help of updated and different types of datasets for training and learning, machine learning-based IDS can improve and evolve continuously to accurately identify new and rising threats and attack patterns. Some of the important aspects of IDS-based machine learning are given below.

- **Data collection**: A dataset is collected from different sources that contain normal and anomaly network traffic. Machine learning algorithm uses these datasets as their training sources.
- **Feature extraction**: From raw data, essential features have been determined to reflect network activity or events in the system. These features can be traffic flow parameters, packet header information, payload content, or statistical measurements extracted from data.
- **Training set preparation**: Training and testing data sets are created from collected datasets. The testing set is used for evaluating how efficiently the trained model performs, whereas the training set is utilized for developing machine learning algorithms.
- **Selection of algorithms**: For testing and training datasets there a different types of algorithms that can be useful for decision trees, support vector machines (SVM), random forests, and neural networks.
- **Model training**: During the training session, the Selected machine learning algorithm learns about patterns and behavior of normal and anomaly network traffic.

- **Model Evaluation**: The testing set is used to evaluate the trained model's performance using measures including accuracy, precision, recall, and F1-score. It allows for fine-tuning any model flaws or restrictions.
- **Deployment**: If the model performs better then it can be implemented as a component of IDS. It continuously examines system logs and real-time network traffic, sending warnings or acting when it discovers irregularities or possible security concerns.

Security Threats IDS prediction model

IDS are designed to find security threats in the network and alert security personnel for incident handling. However, the IDS system is exposed to a variety of security vulnerabilities.

- Adversarial Attacks: IDS can be victim to adversarial attacks, where the attacker tampers input data or trains the model in inaccurate or intentionally false data, which can result in false threat detection or inaccuracy in its prediction capabilities.
- Data Poisoning: Like adversarial attacks, data poisoning involves training models with misleading or inaccurate data. By interfering with the training process, an attacker can change the behavior of the model resulting in incorrect prediction or faulty detection ability.
- False positives and Negatives: During the monitoring process, IDS can flag normal action as an attack, and genuine attacks are overlooked. IDS creates false positives and false negatives. False positives might make administrators less sensitive to alerts because of alert fatigue, whereas false negatives compromise the network's security posture.
- Model Overfitting: IDS prediction models may be overfitted, which occurs when the model gets overly centralized on the training set and is unable to generalize successfully to untested data. Overfitting can lead to a lack of performance, a rise in false positives or negatives, and a decrease in the ability to distinguish real threats.

It is important to continuously evaluate and improve IDS prediction models to reduce these risks. Utilizing prevention mechanisms like an adversarial defense mechanism, strong model training, and ongoing concept drift monitoring. Additionally, fairness and bias should be considered while building models to ensure data are handled correctly.

Comparison

Security Threats IDS Prediction Model (ST-IDS) and Machine Learning -Intrusion Detection System (ML-IDS) are important elements in the modern cybersecurity environment, focused on detecting and mitigating security threats, attacks, and intrusions within systems and networks. These two are compared in the following:

1. **Purpose**:
   a. ML-IDS: By examining past data and finding patterns or behaviors that are different from normal, ML-IDS detects anomalies and intrusions in a network or system. It focuses on the identification of unauthorized network activity in real-time or almost real-time.
   b. ST-IDS: Based on different data analyses and threat intelligence, ST-IDS predicts, and forecasts prospective security risks and vulnerabilities based on behavior and patterns. It seeks a proactive identification approach to new threats and weaknesses.
2. **Data sources**:
   a. ML-IDS: ML-IDS trains its intrusion detection models using real-time network traffic which includes data packets, network flows, and logs as well as system logs and event data from host or network devices and collects behavior data profile of normal system activity.
   b. ST-IDS: Foreseeing possible threats and vulnerabilities requires the collection of threat intelligence, which ST-IDS obtains from a variety of sources including historical attack data, network traffic logs, vulnerability databases, dark web monitoring, and industry reports.
3. **Timing**:
   a. ML-IDS: ML-IDS works in real-time or almost real-time to find abnormalities or continuing intrusions as they happen.
   b. ST-IDS: Future threats are the main emphasis of ST-IDS operations, and it makes predictions about prospective threats and vulnerabilities that could materialize in the future.
4. **Training**:
   a. ML-IDS: For building a baseline of typical network behavior, ML-IDS models need to be trained on historical data which are mainly labeled

datasets. The system picks up new information continuously and adjusts to network changes.

b. ST-IDS: Security intelligence and historical threat data are used to train ST-IDS models to find trends and patterns in possible threats and vulnerabilities.

5. **Detection Techniques**:

a. ML-IDS: Anomaly-based detection, clustering, and classification are just a few of the machine learning algorithms that ML-IDS utilizes to identify security threats.

b. ST-IDS: Predictive modeling, data mining, and statistical analysis are used by ST-IDS to find new threats and weaknesses.

6. **Use Cases**:

a. ML-IDS: ML-IDS is useful for real-time threat detection since it can identify known and unidentified intrusions as well as abnormalities.

b. ST-IDS: ST-IDS is useful for zero-day vulnerabilities, and early warning systems. Organizations that wish to participate and get ready for potential security threats and vulnerabilities might benefit from ST-IDS since it takes a proactive approach to security.

7. **Alerting and Response**:

a. ML-IDS: When an intrusion or anomaly is discovered, ML-IDS sends immediate notification, enabling quick reaction and remediation.

b. ST-IDS: By providing information about potential risks in the future, ST-IDS enables organizations to plan security measures and take preventative steps.

In conclusion, the roles played by ML_IDS and ST-IDS in cybersecurity are distinct but complimentary. While ST-IDS is more interested in foreseeing potential threats and vulnerabilities, ML-IDS is more concerned with real-time intrusion detection. Both technologies can be employed to strengthen overall cybersecurity defenses depending on the particular security requirements of an organization.

Summary

This chapter explains current research on intrusion detection systems and their brief history and operations. Additionally, it describes the inner workings of IDS and the type of IDS available at present. Detailed info about machine learning algorithms and approaches to acquiring prediction models are discussed here. In the end, threats IDS-based machine learning can face are also discussed.

Chapter 3: Methodology

A systematic technique used to conduct research, study, or handle an issue is presented in this chapter. It outlines the steps, procedures, and techniques utilized in data collection, information analysis, and decision-making. This chapter includes a brief description of each phase.

Introduction

This chapter focuses on the latest intrusion detection systems (IDS), and different machine learning methods like Decision Trees, Random Forests, K-nearest neighbors classifiers (KNN), and Logistic Regression. As well as a system used to train models as well as programming languages like Python for writing codes. Algorithms are trained in model to enhance their functionality and accuracy through various techniques like data cleaning, feature engineering, feature selection, and data analysis.

Research Design

General research strategies are used to answer the research question in this section. This study consists of a combination of general research strategies, consisting of both qualitative and quantitative research methods, often integrated to enhance the depth of analysis. The research methodology for the IDS predictive model is designed using three common research approaches:

Qualitative research approaches: This approach is used to examine and understand events, with a focus on individual and social contexts. It focuses on gathering and analyzing non-numerical data from sources such as interviews, observations, documents, and artifacts. Qualitative research helps in exploring complex events, as it provides rich insights into social, cultural, and subjective aspects of the research topic.

Quantitative research design: It is employed to collect and analyze numerical data. It is often used to test different hypotheses examine relationships or correlations between variables and generalize findings to a larger population. Quantitative research involves structured data collection methods and statistical analysis as well as providing consistency.

Mixed research design: It combines good aspects of both qualitative and quantitative approaches, offering a more detailed understanding of the research topic. By using qualitative methods, researchers can make generalized claims, on the other hand, qualitative methods allow them to capture detailed and different insights. The demand for mixed design depends on research goals, the nature of the topic discussed, and the resources available for use. The mixed design gives researchers a new viewpoint to investigate complex issues, which in accordance increases the creditability of results.

Proposed Methodology

Different types of learning techniques are studied to identify intrusion in network traffic. These algorithms identify both regular and abnormal network connections. The steps for competing predictions are shown in the below figure. The first stage is to import data from KAGGLE (an open-source data collection on the internet). The second step is to clean raw data by removing duplicates and null values as well as unnecessary columns. Label Encoder is used to transform categorical data into numerical data, and features are chosen for training data with the use of feature maps. KNN, Decision Tree (DT), Random Forest, and Logistic Regression are utilized for training the model. The time of training, test score, and training score are calculated for each algorithm and are compared according to their precision, recall, accuracy, performance, and f1 score.

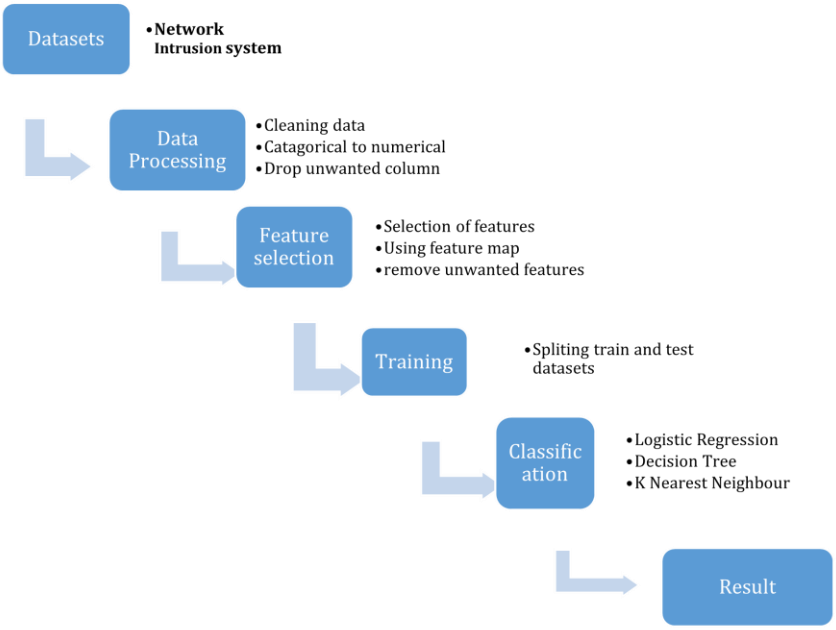


Figure 5: Steps for prediction for this research

Used tools and Environment

To achieve the project’s objective, we need basic software and hardware. For creating a model, multiple machine-learning strategies are offered. Numerous tools and environments are deployed for the development of prediction models. The table below is a list of hardware and software used to perform this project.

S. N	Software/Hardware	Features
1	RAM	8 GB

2	Processor	Intel(R) Core (TM) i5-1235U CPU @ 12 MB cache, 10 cores, 12 threads, 4.40GHz
3	Programming Language	Python
4	Operating system	Windows 11 Pro
5	Libraries used	Pandas, Seaborn, scikit Learn, NumPy, matplotlib
6	Tool	MS-wood, Adobe Photoshop
7	Platform	Jupyter Notebook

Python

Python is a popular programming language created by Guido van Rossum and released in 1991. It is used to construct a prediction model for an intrusion detection system. It is used widely in different sectors for web development, system dataset management, and data processing. Python is supported in multiple platforms such as Linux, Windows, Mac, and Raspberry Pi. It consists of a simple learning curve and an active community for support. Since Python is an interpreted language, code can be executed quickly, which allows for fast prototyping. It supports both functional and procedural programming styles.

Figure 6: Python programming language

Jupyter Notebook

For this project, I'll be using Jupyter Notebook which is a Python-based development environment. Jupyter is open-source software that allows users to create and share documents with features including live code share, equations, text, and visualizations. It supports multiple programming languages including Julia, Python, and Ruby with over 100 additional kernels available. It comes with a default kernel, IPython for creating and running Python codes.

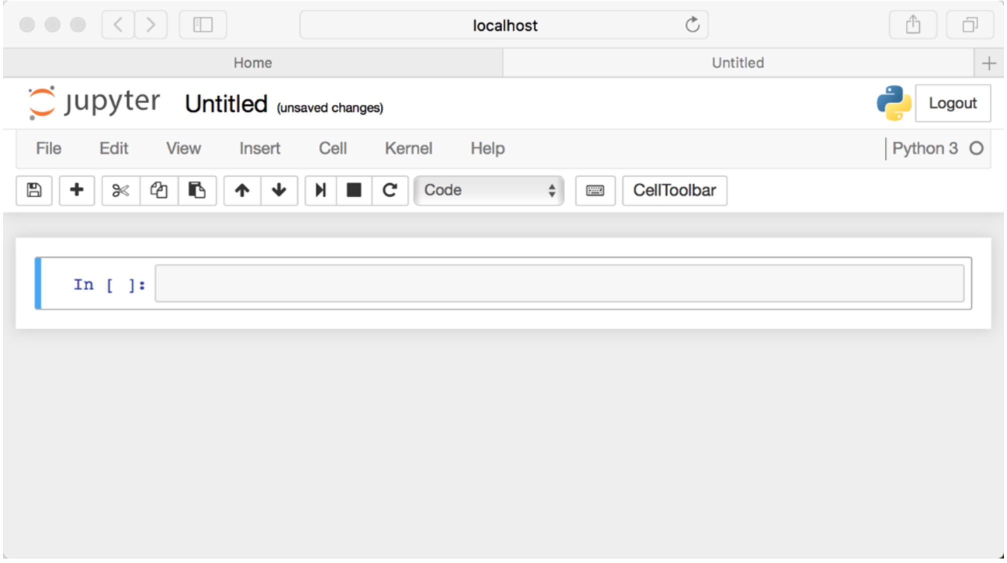


Figure 7: Starting of Jupyter

Libraries

NumPy

NumPy library is a core library in Python used for numerical and scientific computing. It stands for "Numerical Python". It provides support for large, multi-dimensional arrays and matrices. Additionally, it consists of complex mathematical procedures for using arrays. NumPy is commonly used in data science and machine learning along with datasets and performs mathematical computation or prepares data analysis.

Matplotlib

Matplotlib is a widely used Python toolkit for creating static, animated, and interactive visualizations in different formats. It provides a vast selection of plotting tools and customization options to enable the creation of several types of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and many more. Matplotlib is a versatile and important tool for data visualization in Python. It is widely used in data analysis, scientific research, and data visualization. It provides detailed control over plot design while also remaining easy to use for basic tasks.

Pandas

Pandas are one of the widely used and most popular Python libraries for data manipulation and analysis. It is used for easy handling of structured data, such as time series data and numerical tables. It provides data structures that are easy to use and data analysis tools efficiently improve working with structured data like CSV files, Excel spreadsheets, SQL databases, and more. It allows users to effectively process, analyze, and manipulate data before applying machine learning models or creating visualization.

Seaborn

Seaborn is a powerful Python visualization based on Matplotlib that provides a high-level interface for creating statistical graphics. It is specifically designed to make complex visualization easier. It makes graphs look good visually. It is a great tool for anyone working in data analysis, machine learning, or data science.

Scikit Learn

Scikit-Learn(sklearn) is a popular machine-learning library in Python. It is used for different types of machine learning applications such as model selection, regression, clustering, dimensionality reduction, classification, and data preparation. It provides a wide range of simple and efficient tools for data analysis and modeling, which is built on top of NumPy, SciPy, and Matplotlib. It is well suited for small and medium-sized machine learning problems. It is a simple tool for both beginners and experienced users.

Algorithms

Random Forest (RF)

Random forest algorithm is a collaborative learning method that is used mainly for classification and regression tasks. During the training phase of the algorithm, it creates multiple decision trees that resemble a forest. It is based on the concept of decision trees, but it improves it by reducing overfitting and increasing accuracy using multiple trees and randomization. The random forest technique separates the dataset into multiple subgroups. The best split is found by randomly selecting several parameters at each node.

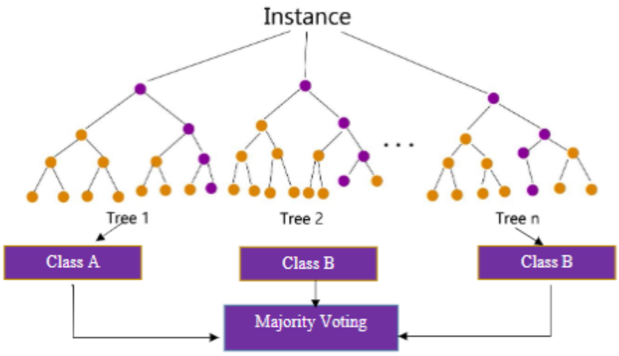


Figure 8:: A flowchart illustrating the use of random forests

(Source- (Atawodi, 2019))

Highlight: Random Forest builds multiple decision trees and combines their predictions. It is known as bagging (Bootstrap Aggregating). With the help of aggregating results into multiple models, the final prediction can be more accurate and less likely to fall under overfitting compared to the single decision tree. Random Forest generates each decision tree using a random subset of data which means each tree is trained in a slightly different way in data which results in less variance and overfitting.

Steps in building Random Forest:

1. Bootstrapping: Randomly pick data points from the training set with replacements to create multiple new datasets. Some data points might appear more than once in each new dataset.
2. Train Trees: For each new dataset build a decision tree. But instead of using all features, only a random selection of features is considered when making decisions at each node.
3. Make Predictions: After training the trees, use them to make predictions:
    a. For Classification: Each tree "votes" on the class, and the class with the most votes is the final prediction.
    b. For Regression: Take the average of all the tree predictions to get the result.

Decision Tree (DT)

Decision Tree (DT) is a simple and powerful supervised machine learning algorithm used for both classification and regression issues. It is a model containing core nodes that describe different t-feature tests, branching nodes that represent the outcome of decisions, and leaf nodes that represent class labels or target values.

Key Concepts of Decision Trees:

- Root Node: It’s the topmost node of the tree that represents the entire dataset.
- Internal Nodes: It represents decisions or tests on specific features.
- Branches: These present because of the decision made at the parent node.
- Leaf Nodes (Terminal Nodes): It is the final nodes of the tree, where predictions or results are made.

Decision trees have several limitations, such as overfitting where the model becomes so specific to training data that it doesn’t work on new data. Small changes in training data can lead to a completely different tree being learned. A decision tree is biased toward more features which can lead to biased models.

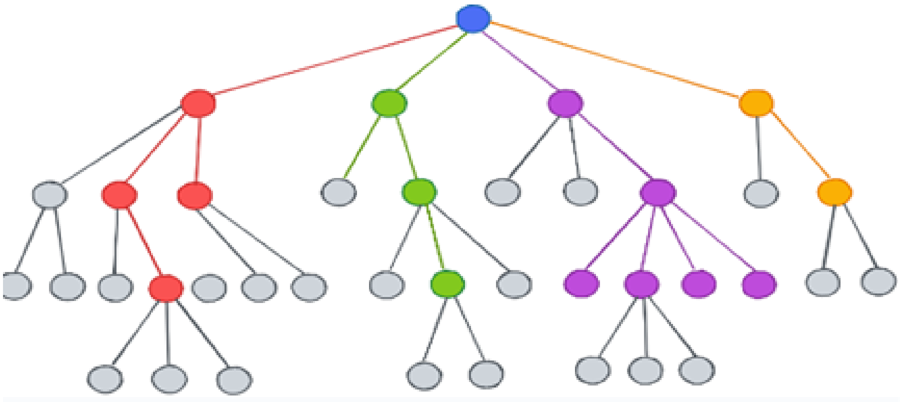


Figure 9: Decision tree

Source- (algodaily, n.d.)

Highlight: Decision Tree is the most popular and effective machine learning algorithm for classification and regression tasks. It is a powerful model that splits data based on feature value which helps in making predictions. It is easy to use and the fact that there is less requirement for arithmetic makes it a popular choice. The decision Tree name suggests that it displays prediction in the form of a series of feature-based splits using a flowchart looking like a tree.

Logistic Regression (LR)

Logistic Regression is a widely used machine learning algorithm mainly for binary classification problems. Despite having regression in the name, it is not a classification algorithm, nor a regression algorithm. It is widely used in a range of fields, including marketing, finance, healthcare, and social sciences. It describes the relationship between a set of independent variables and binary outcomes by determining the probability that the result will fall into one of several different groups. It is a simple and effective model making it the first choice for binary classification tasks. It is a beginning point for many classification problems and with proper feature engineering, it can output a good variety of datasets.

K-Nearest Neighbors Algorithm (KNN)

The K-Nearest Neighbors (KNN) algorithm is a simple and powerful supervised machine learning algorithm used for classification and regression issues. It uses a non-parametric method, meaning it doesn’t make decisions about data distribution, instead it predicts based on input data. KNN is easy to understand and implement which makes it popular against other similar algorithms. KNN doesn’t make any assumptions about data which makes it a more flexible algorithm to work with. Among many advantages, it does consist of shortcomings also. KNN requires the calculation of the distance between input and every single data point in the training set which is inefficient for large datasets. KNN requires more storage as it stores entire training datasets.

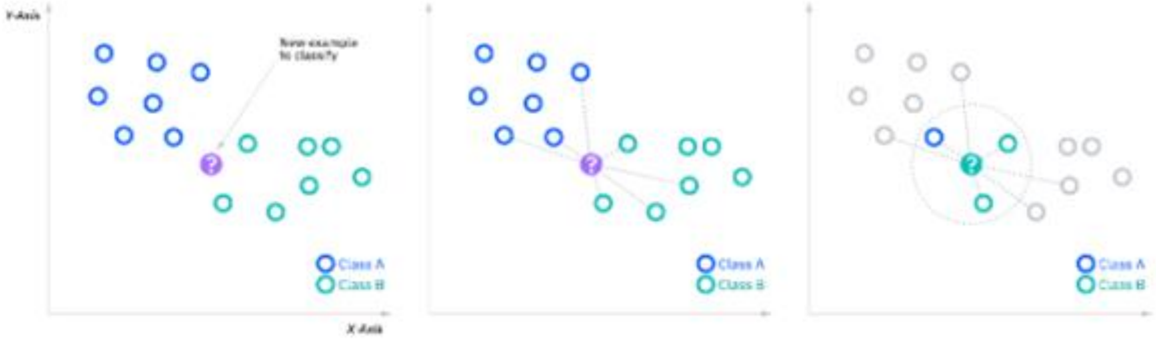


Figure 10: KNN example

Source- (IBM, n.d.)

Highlight: The k-nearest neighbors' algorithm is simple, and intuitive for classification and regression tasks when data distribution is not known. It is a supervised learning classifier that uses proximity to create classifications or predictions about the grouping of single data points. Although it is seen as capable of doing classification or regression issues, it is commonly used for classification methods because it depends on the idea that comparable points can be discovered close to one another. By calculating the distance between test data and all training points, it is trying to predict the proper class for test data. However, it can become inefficient for large datasets and high dimensional data due to the need for distance calculations.

Conceptual Design

The research on this project starts with the collection of data and ends with a prediction model, analysis, comparison, and so on. This flow diagram represents the conceptual design for this project.

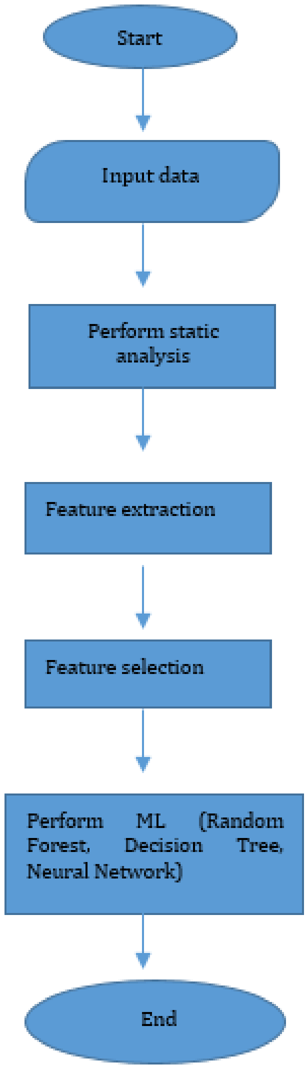


Figure 11v: Conceptual Design

Data Collection and Preprocessing

We will take the dataset and with some processing give out a predictive model. It also provides information about the data collected, where it was collected from, the type of data, different types of data processing, and the end expected result.

Data Collection

Data is sourced from the online open-source website KAGGLE, which stores various types of data sets. The connection to the data set source is at APPENDEX 1 below. There are two data sets present: the training dataset and the test dataset. The test datasets have 821495 rows and 16 columns, and the train datasets consist of 1364524 rows and 16 columns.

Data Description

This dataset was produced by Lancaster University, and it contains network flow data, specifically related to network intrusion detection. It consists of data types with malicious behaviors, outliners, normal traffic, and telemetry. Malicious datasets consist of data like emerging attack vectors which are identified by the composition of honeypots deployed within Lancaster University's address space. Outliners mean that it is unclear if this group of traffic is malicious or not. Normal traffic indicates normal legitimate traffic. The data is also correlated with external Cyber Threat Intelligence (CTI) sources, which are responsible for the identification and labeling of these threats. Each network connection is a sequence of TCP packets that start and end at specific times and enable data to flow from the host IP address to a targeted IP address based on specific protocols. Each network connection is classified as benign, malicious, or outlier, each of which corresponds to a certain type of assault. Each record is around 100 bytes in size. The dataset includes 16 features for each connection, with 15 quantitative and 1 qualitative attribute.

A dataset's significance is determined by various factors, including publication verification, valuation, transdisciplinary application of data, and longitudinal study. This dataset is valuable in cybersecurity research, system testing, and training. Cybersecurity monitoring involves collecting data from sensors on cloud and on-premises systems, which support good threat detection and response decisions.

The class variable is made up of two groups (BHOSALE, 2019):

- Benign or Normal Traffic
- Malicious or Harmful Traffic
- Outliner or Unusual Traffic

Analysis of data

As stated in the table below, the data is assessed based on the characteristics' names, descriptions, and types. Data types are assessed according to whether they are continuous or discrete, along with a brief description of their attributes. The header rows in the datasets are referred to as features in the table below.

Name	Type	Description
Src_ip	Discrete	Source IP address anonymized to its autonomous system.
Src_port	Discrete	Source port number.
Dest_ip	Discrete	The destination IP address of the flow.
Dest_port	Discrete	Destination port number.
Protocol	Discrete	Protocol used e.g. TCP, UDP
Bytes_in	Continuous	Number of bytes transmit from source to destination.
Bytes_out	Continuous	Number of bytes transmitted from destination to source
Num_pkts_in	Continuous	Number of packets sent from source to destination

Num_pkts_out	Continuous	Number of packets sent from destination to source
entropy	Continuous	Entropy in bit per byte of data fields, ranges from 0 to 8.
Total_entropy	Continuous	Total entropy in bytes over all bytes in data fields.
Mean_ipt	Continuous	Mean of inter-packet arrival time.
Time_start	Continuous	The start time is in seconds since the Unix epoch.
Time_end	Continuous	End time in seconds since the Unix epoch.
Duration	Continuous	Duration of flow in microsecond precision.
Label	Discrete	The label assigned in the dataset: Benign, Outlier, or Malicious

Table 2: Individual Flow Characteristics

Name	Type	Description
Src_ip	Discrete	Source IP address anonymized to its autonomous system.
Dest_ip	Discrete	Destination IP address anonymized similarly to src_ip
Protocol	Discrete	Protocol type used TCP or UPD
Count	Continuous	Number of connections to the same host in the 2-second window.
Serror_rate	Continuous	Percentage of connection with SYN errors in 2 2-second window.
Rerror_rate	Continuous	Percentage of connection with REJ error in the 2-second window.
Same_srv_rate	Continuous	Percentage of connections using the same service in 2 2-second window.
Diff_srv_rate	Continuous	Percentage of connections using different services in 2 2-second window.
Srv_count	Continuous	Several connections to the same service within 2 seconds.
Srv_serror_rate	Continuous	Percentage of SYN errors for connections using the same service in 2 2-second window.
Srv_rerror_rate	Continuous	Percentage of REJ errors for connections using the same service in 2 2-second window.
Srv_diff_host_rate	Continuous	Percentage of connections to different hosts for the same service in the 2-second window.

Table 3: Time-based Characteristics

Name	Type	Description
Label	Discrete	The label indicates network traffic is benign, an outlier, or malicious.
Duration	Continuous	Duration of flow measured in milliseconds.
Bytes_in	Continuous	Number of bytes transmitted from source to destination

Bytes_out	Continuous	Number of bytes transmit from destination to source.
Num_pkts_in	Continuous	Number of packets sent from source to destination.
Num_pkts_out	Continuous	Number of packets sent from destination to source.
Entropy	Continuous	Entropy in bits per byte of data fields.
Total_entropy	Continuous	Total entropy in bytes across all data fields.
Time_start	Continuous	Flow's start time in seconds using unix epoch.
Time_end	Continuous	Flow's end time in seconds using unix epoch.

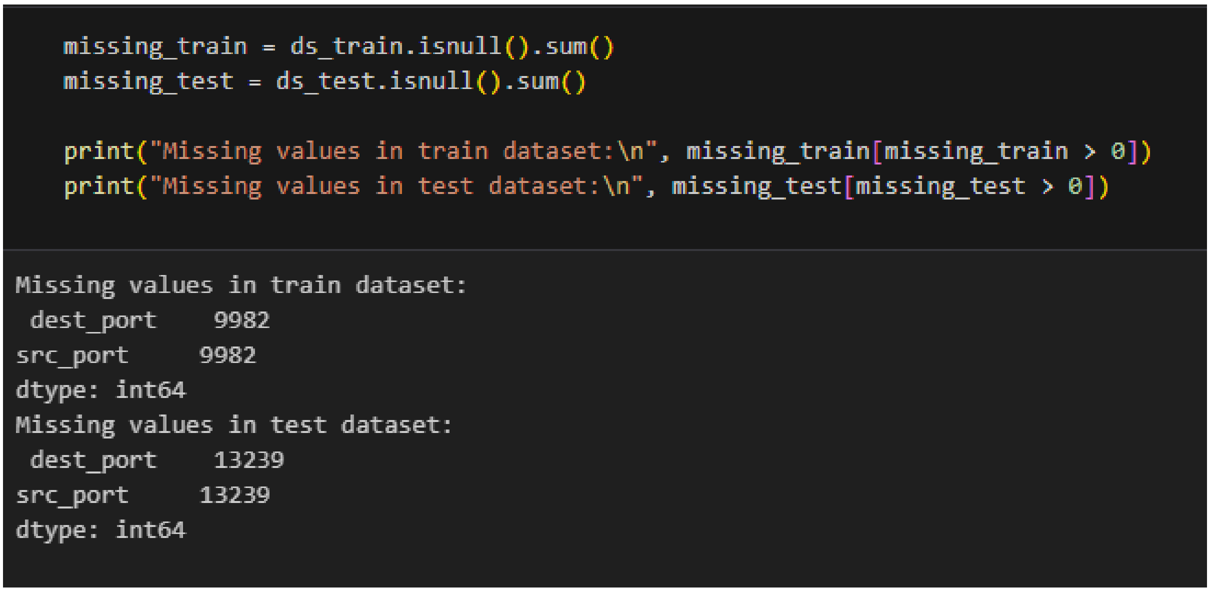
Table 4: Attack/Threat Detection Characteristics

Data Cleaning and Pre-processing

After the summary of each data type feature along with continuous or discrete evaluation. Raw data is analyzed by looking into inaccurate data, formatting problems, and inconsistent values. It starts with identifying and handling missing values, and removing duplicates. The next step would be normalizing data types and formatting.

Handling missing values

The handling of missing values in a dataset is a crucial step in data cleaning for model correctness. The following screenshots illustrate steps for dealing with missing values.



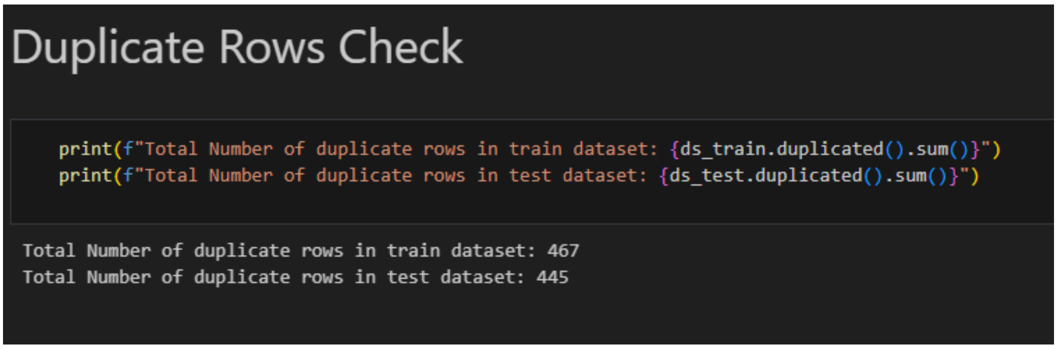
```
missing_train = ds_train.isnull().sum()
missing_test = ds_test.isnull().sum()

print("Missing values in train dataset:\n", missing_train[missing_train > 0])
print("Missing values in test dataset:\n", missing_test[missing_test > 0])
```

```
Missing values in train dataset:
 dest_port    9982
src_port     9982
dtype: int64
Missing values in test dataset:
 dest_port    13239
src_port     13239
dtype: int64
```

Figure 12: Handling missing value

Handling duplication

Finding duplication is an important step in data pre-processing which ensures data integrity and quality. It makes the model free from over-fitting and errors. Duplicate values are created due to errors in data collection, merging of datasets, or input mistakes. In the below figure, duplication is being handled but as it shows a total number of duplicates as 0, the current dataset does not consist of any duplicates.



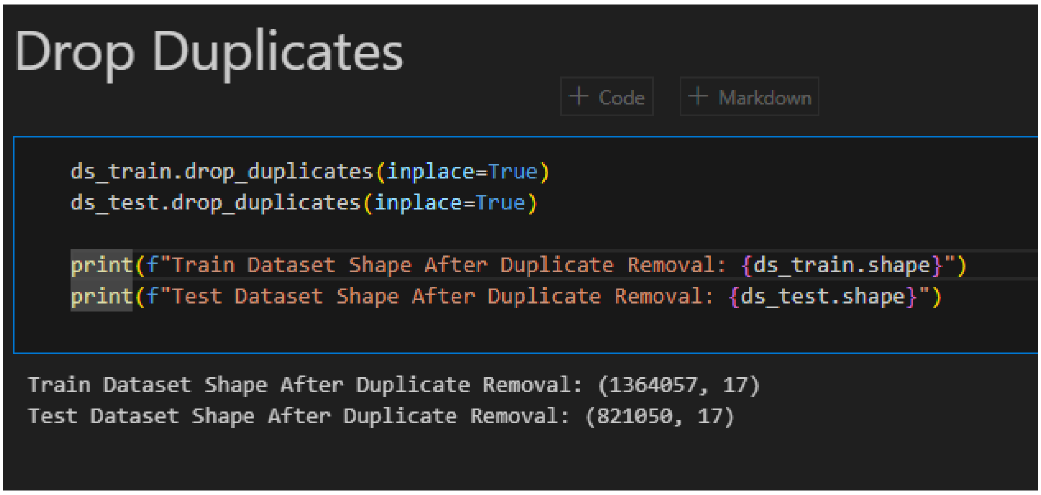
Duplicate Rows Check

```
print(f"Total Number of duplicate rows in train dataset: {ds_train.duplicated().sum()}")
print(f"Total Number of duplicate rows in test dataset: {ds_test.duplicated().sum()}")
```

```
Total Number of duplicate rows in train dataset: 467
Total Number of duplicate rows in test dataset: 445
```

Figure 13: Checking for duplicates

After the check is complete, we can drop duplicates as shown below:



Drop Duplicates

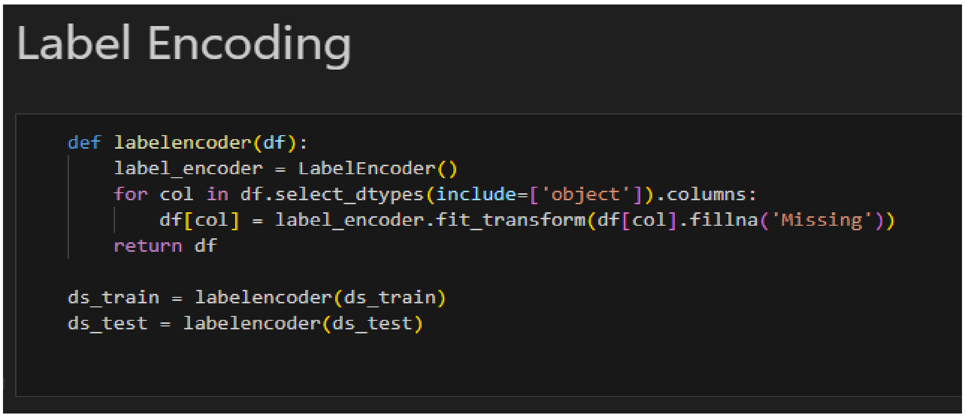
```
ds_train.drop_duplicates(inplace=True)
ds_test.drop_duplicates(inplace=True)

print(f"Train Dataset Shape After Duplicate Removal: {ds_train.shape}")
print(f"Test Dataset Shape After Duplicate Removal: {ds_test.shape}")
```

```
Train Dataset Shape After Duplicate Removal: (1364057, 17)
Test Dataset Shape After Duplicate Removal: (821050, 17)
```

Figure 14: Handling duplication

Encoding categorical data

It is the process of converting non-numerical categories into numeric values so it can also be used in machine learning models. Label Encoder is used for this process. Label Encoding assigns a unique integer to each category.

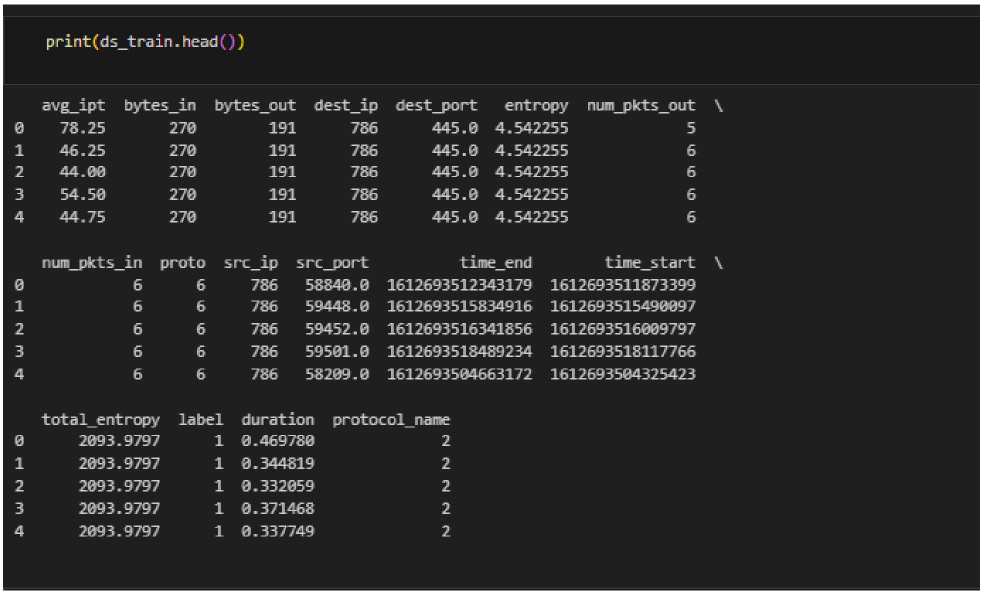


Label Encoding

```
def labelencoder(df):
    label_encoder = LabelEncoder()
    for col in df.select_dtypes(include=['object']).columns:
        df[col] = label_encoder.fit_transform(df[col].fillna('Missing'))
    return df

ds_train = labelencoder(ds_train)
ds_test = labelencoder(ds_test)
```

Figure 15: Encoding categorical data

From the illustration, classes like normal or anomaly are converted into 1 or 0 respectively as in the below figure.



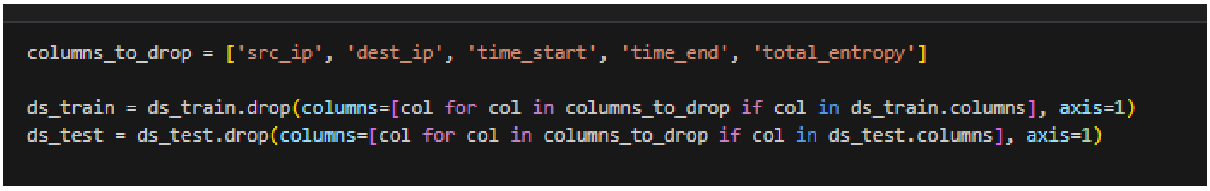
```
print(ds_train.head())
```

```
   avg_ipt  bytes_in  bytes_out  dest_ip  dest_port   entropy  num_pkts_out  \
0    78.25       270        191      786      445.0  4.542255             5
1    46.25       270        191      786      445.0  4.542255             6
2    44.00       270        191      786      445.0  4.542255             6
3    54.50       270        191      786      445.0  4.542255             6
4    44.75       270        191      786      445.0  4.542255             6

   num_pkts_in  proto  src_ip  src_port           time_end         time_start  \
0            6      6     786   58840.0  1612693512343179  1612693511873399
1            6      6     786   59448.0  1612693515834916  1612693515490097
2            6      6     786   59452.0  1612693516341856  1612693516009797
3            6      6     786   59501.0  1612693518489234  1612693518117766
4            6      6     786   58209.0  1612693504663172  1612693504325423

   total_entropy  label  duration  protocol_name
0      2093.9797      1  0.469780              2
1      2093.9797      1  0.344819              2
2      2093.9797      1  0.332059              2
3      2093.9797      1  0.371468              2
4      2093.9797      1  0.337749              2
```

Figure 16: Illustration after categorical conversion

Dropping columns

In these datasets, there are many unnecessary and duplicate columns. So, dropping these datasets helps to reduce noise and improve model efficiency. So unwanted and duplicate columns are dropped as shown in the figure below.

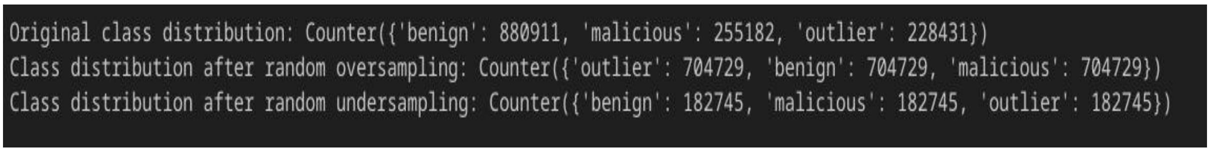


```
columns_to_drop = ['src_ip', 'dest_ip', 'time_start', 'time_end', 'total_entropy']

ds_train = ds_train.drop(columns=[col for col in columns_to_drop if col in ds_train.columns], axis=1)
ds_test = ds_test.drop(columns=[col for col in columns_to_drop if col in ds_test.columns], axis=1)
```

Figure 17: Dropping unwanted columns

Data set balancing

Data from the model must be balanced to be accurate; otherwise, errors may result. For our dataset, we used class distribution with random oversampling and undersampling to balance out all the labels.

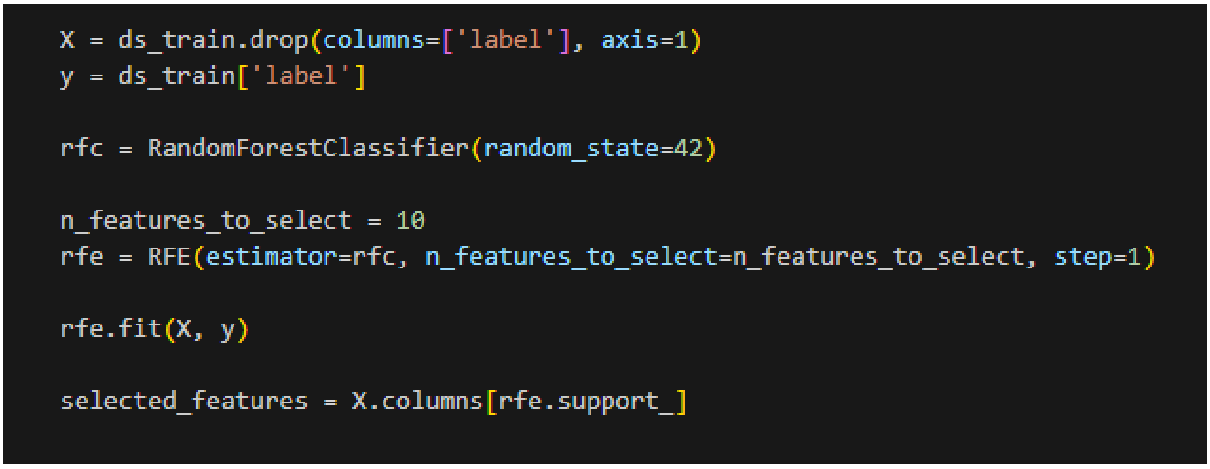


```
Original class distribution: Counter({'benign': 880911, 'malicious': 255182, 'outlier': 228431})
Class distribution after random oversampling: Counter({'outlier': 704729, 'benign': 704729, 'malicious': 704729})
Class distribution after random undersampling: Counter({'benign': 182745, 'malicious': 182745, 'outlier': 182745})
```

Figure 18: Data Set Balancing

Feature Selection

The scikit-learn library's class RFE (Recursive Feature Elimination) is used to choose features. The RFE (Recursive Feature Elimination) is a powerful feature selection tool that helps identify the most important features in datasets by recursively removing the least important ones. It is used to determine the most significant characteristics using this class. This process involves fitting a machine learning model (estimator) on the data and ranking features based on their importance (e.g., coefficients in linear models or features importance in tree-based models). It continues until the desired number of features is reached. To improve model performance and reduce overfitting, feature selection is widely employed in machine learning projects.

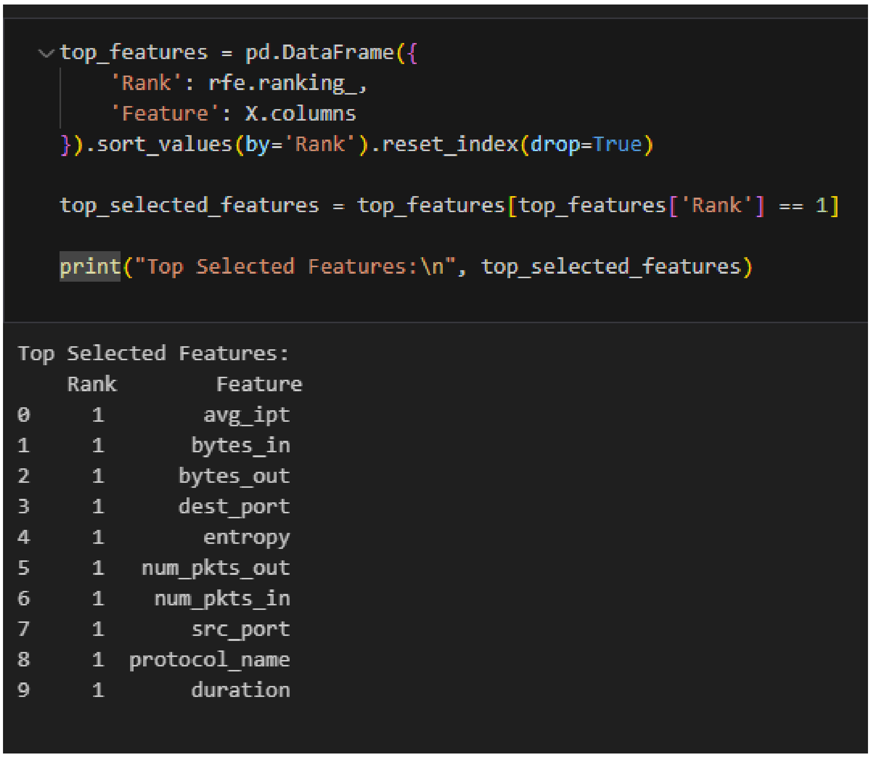


```
X = ds_train.drop(columns=['label'], axis=1)
y = ds_train['label']

rfc = RandomForestClassifier(random_state=42)

n_features_to_select = 10
rfe = RFE(estimator=rfc, n_features_to_select=n_features_to_select, step=1)

rfe.fit(X, y)

selected_features = X.columns[rfe.support_]
```

Figure 19: Feature Selection

As we can see library is imported for feature selection from the above figure. The n_features_to_select argument in the RFE specifies how many features from the dataset you want to use. In this example, 10 features are selected and a random forest model is used for feature ranking. After fitting using ‘RFE’, we may access selected features using the method ‘get support’, which returns a Boolean array where True indicates selected features. The boolean mask is stored in variable feature_map. It is important to list each Boolean value with the appropriate column name by iterating through a list of “X.train.columns” that have been enumerated.

The top 10 selected features are shown in the figure below.

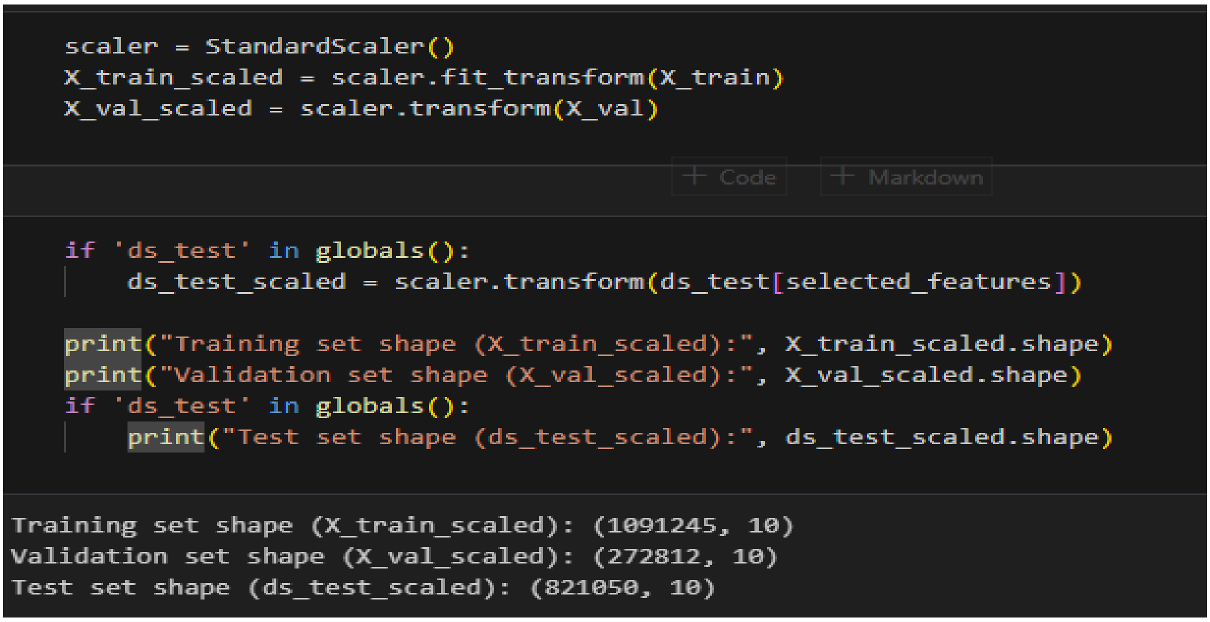


```
top_features = pd.DataFrame({
    'Rank': rfe.ranking_,
    'Feature': X.columns
}).sort_values(by='Rank').reset_index(drop=True)

top_selected_features = top_features[top_features['Rank'] == 1]

print("Top Selected Features:\n", top_selected_features)
```

```
Top Selected Features:
    Rank        Feature
0     1         avg_ipt
1     1        bytes_in
2     1       bytes_out
3     1       dest_port
4     1         entropy
5     1     num_pkts_out
6     1      num_pkts_in
7     1        src_port
8     1   protocol_name
9     1        duration
```

Figure 20: Top 10 features

Standard scaling is used to transform features to have a mean of 0 and a standard deviation of 1. In the following code below we are using StandardScaler to standardize both training data X and test data ds_test.



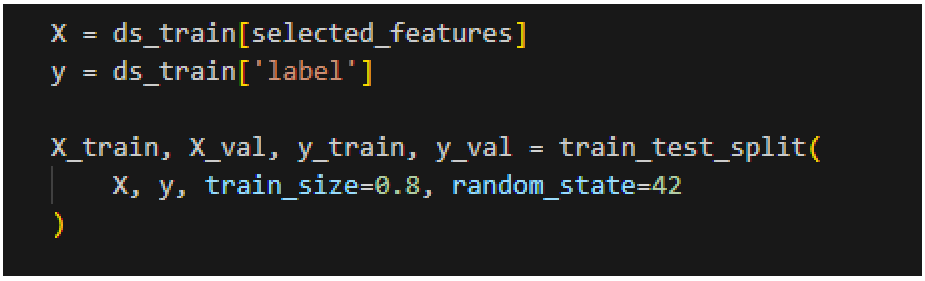
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

```
if 'ds_test' in globals():
    ds_test_scaled = scaler.transform(ds_test[selected_features])

print("Training set shape (X_train_scaled):", X_train_scaled.shape)
print("Validation set shape (X_val_scaled):", X_val_scaled.shape)
if 'ds_test' in globals():
    print("Test set shape (ds_test_scaled):", ds_test_scaled.shape)
```

```
Training set shape (X_train_scaled): (1091245, 10)
Validation set shape (X_val_scaled): (272812, 10)
Test set shape (ds_test_scaled): (821050, 10)
```

Figure 21: Standard Scaling

Using train_test_split from scikit-learn to split the dataset into training and testing subsets. In the given below code function takes feature matrix X (which contains input data)and target vector y (which contains corresponding labels or outputs).

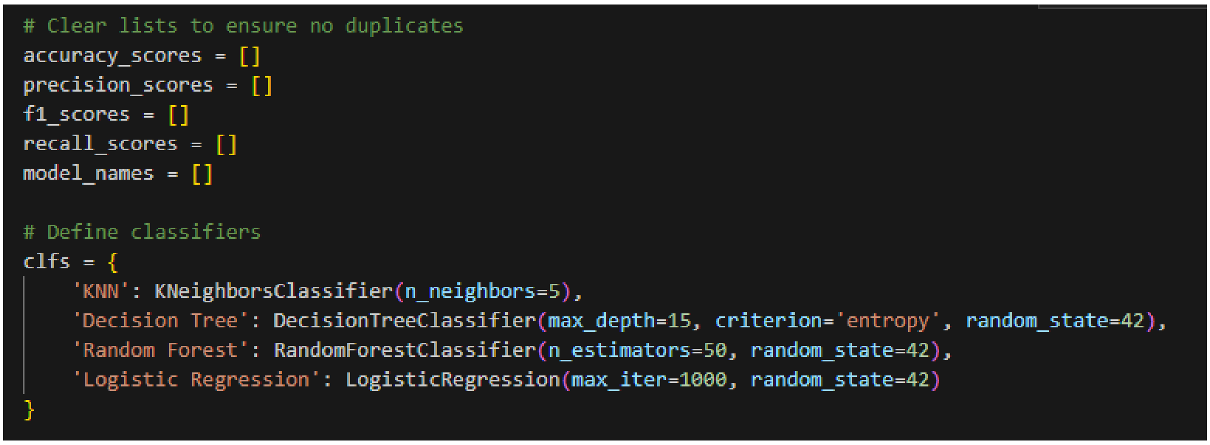


```
X = ds_train[selected_features]
y = ds_train['label']

X_train, X_val, y_train, y_val = train_test_split(
    X, y, train_size=0.8, random_state=42
)
```

Figure 22: Dataset splitting

Building Model

Depending on the nature of data and to achieve results there are specific machine learning models. Before the selection of the model's data goes through multiple steps like dataset selection. Cleaning and data wrangling, detection of missing values. Null values, duplications, and changing categorical data into numerical. The main aim of this research is to develop a prediction model for an intrusion detection system. For that use case Decision Tree (DT), Logistic Regression (LR), and k-nearest Neighbors (KNN) are used in this project. During the preparation of training and testing two separate datasets are used for training and testing purposes.



```
# Clear lists to ensure no duplicates
accuracy_scores = []
precision_scores = []
f1_scores = []
recall_scores = []
model_names = []

# Define classifiers
clfs = {
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Decision Tree': DecisionTreeClassifier(max_depth=15, criterion='entropy', random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=50, random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42)
}
```

Figure 23: Defining classifiers

Decision Tree

Decision Tree Classifier trains with a maximum depth of 15 and 'entropy' creation on training data (X_train, y_train). After fitting the model, it predicts test data (X_test) and evaluates its performance by calculating accuracy, and precision and generating a classification report.

K-Nearest Neighbors:

K-Nearest Neighbors (KNN) classifier on training data (X-train, y_train) and uses it to predict test set (X_test). It evaluates model performance by calculating accuracy and precision scores and generates classification reports.

Logistic Regression

The logistic regression model is used for classification. The maximum number of 1000 is set for the number of iterations the algorithm should take to converge during the training process. It is used to train the logistic regression model on training data (X_train and y_train) and uses it to predict the target variable on the test set (X_test).

Random Forest

A Random Forest classifier is used in training data. It trains data with a specific number of decision trees in this case 50. It is used to predict new or unseen data with the help of all the decision trees.

Summary

In this chapter, we looked at different aspects of data, the origin of the dataset, type of data present in the dataset. Here we analyse data to get a general idea of what types of data are present in training and test datasets. Datasets afterward are cleaned and pre-processed to remove any null value or duplicate data through data cleaning and pre-processing. In addition, feature selection is used in data to select features, and the selection of the model to be used in this project is also done here. Decision Tree, Logistic regression, and K-Nearest Neighbor are used for the prediction model.

Chapter 4: Result And Analysis

This section consists of the results and analysis of the trained model along with the outcome of the project. This model helps determine which model is best for the project along with a conclusion of testing the model with the unseen dataset.

Used Software and Hardware

To complete the whole project the used software and hardware are mentioned in the table below:

S.N	Software/Hardware	Features
1	RAM	8 GB
2	Processor	Intel(R) Core (TM) i5-1235U CPU @ 12 MB cache, 10 cores, 12 threads, 4.40GHz
3	Programming Language	Python
4	Operating system	Windows 10 Pro

5	Libraries used	Pandas, Seaborn, sci-kit Learn, NumPy, matplotlib, and so on
6	Tool	MS-word, Adobe Photoshop
7	Platform	Jupyter Notebook

Table 5: Software and hardware used

Programming

Python is used to write all the codes in this project, and it is more versatile and supports many visual libraries which makes it the top contender. Python comes with a simple syntax structure which makes it easy to learn. It also consists of an active helpful community. It is widely used to go to language when it comes to machine learning problems. It consists of one of the most comprehensive collections of modules, packages, and libraries in programming language. It also supports an easy-to-use machine learning algorithm which helps in the implementation of any algorithm hassle-free. It also supports a variety of visual representations which make it easier to execute.

Parameters

The dataset obtained from KAGGLE consists of 880911 Normal connections(benign), 228431 suspicious connections (outlier), and 255182 anomaly(malicious) connections. There are 1364524 rows and 16 columns in the training data set. Whereas the test dataset consists of 821495 rows and 16 columns. The datasets are divided and extracted according to primary features using the feature selection process. With that process, a total of 17634 rows and 10 columns of data were chosen for modeling or prediction. The whole sample of connections between benign, malicious, and malicious events is shown in the table below.

Connection	**Numbers**
Benign	880911
Malicious	255182
Outlier	228431

Table 6: Train labels

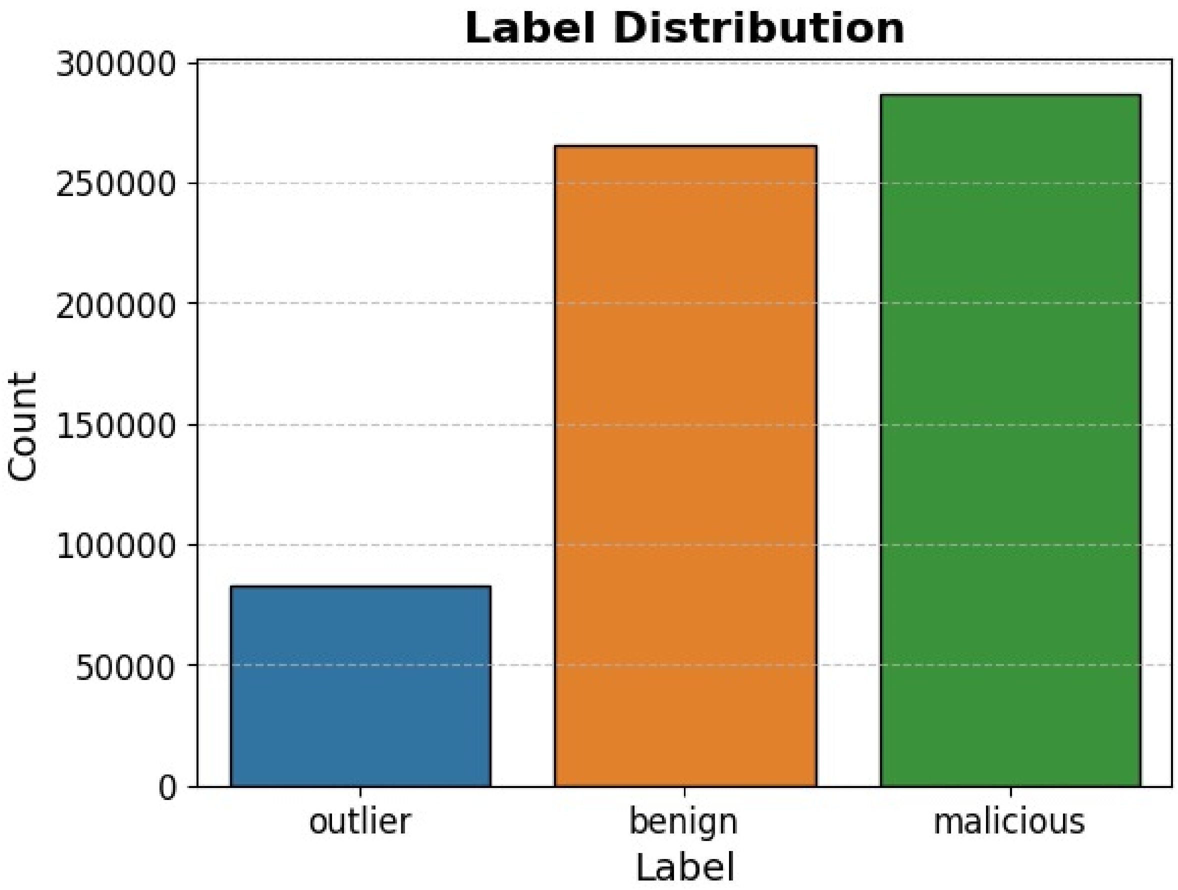


Figure 24: Label diagram of the train

Comparing of Output

Dataset is evaluated with three machine learning models which determine the effectiveness and precision of intrusion detection or accurate prediction of any attacks. Models include k-nearest Neighbor (KNN), Decision Tree, and Logistic Regression. For comparison purposes, we are using indicators like precision, accuracy, recall, and f1-score.

Decision Tree Classifier

We can see how well the prediction model is performed. The accuracy of the classification task is defined as the percentage of instances in the test set that were correctly classified. After testing the Decision Tree model on datasets, accuracy exceeds 96%. The precision of the classification model is used to evaluate its accuracy in identifying positive cases. This metric focuses on the proportion of correctly predicted positive cases. This metric focuses on the proportion of correctly predicted positive cases (true positives) to all cases predicted as positive, including false positives. In simple terms, precision assesses a model's ability to correctly identify real positive events from those it expected to be positive. Benign achieved 100%, outlier achieved

93%, and malicious achieved a 94% precision score with the use of the Decision Tree model. Recall, also known as sensitivity or the true positive rate, is an important metric in machine learning. It measures the proportion of correctly predicted positive instances (true positives) out of all actual positive cases in the dataset (true positives and false negatives combined). Recall indicates how effectively a model captures every actual positive case. A high recall value means the model has a low incidence of false negatives and can correctly identify most positive instances. On the other hand, a low recall score suggests a high proportion of false negatives, indicating that the model misses many positive occurrences. In this case, recall is 79% for outlier type and 98% for malicious connection types. The F1 score, a commonly used evaluation statistic in machine learning, combines both recall and precision into a single value. It provides a comprehensive evaluation of a model's performance by considering both its ability to correctly identify positive instances (precision) and its ability to capture all true positive events (recall). The F1 score for these datasets or models is 96%. Additionally, the macro average scores 96% and weighted average scores 96%, reflect consistent performance across the model's categories.

Evaluation Metrics	Benign	Malicious	Outlier
Precision	100%	94%	93%
F1-Score	100%	96%	85%
Recall	100%	98%	79%
Accuracy	96%		

Figure 25: Train Decision Tree Classifier

K-Nearest Neighbors Classifier

We can conclude that the samples are correctly identified by the model based on the accuracy of the KNN model, which is approximately 90%, as shown in the figure above. Following labels benign achieved 100%, outlier achieved 66% and malicious achieved 87% in precision respectively. The recall scores for malicious and outlier connections are 92% and 90%, respectively. As seen, the F1 scores are similarly high. This suggests that the KNN model performs better than Logistic Regression on this dataset.

From the results below, we can infer that each model has distinct accuracy, precision, recall, and F1-score values. Not all models perform equally well with these datasets. The Random Forest model has the highest accuracy at 98%, followed by Decision Tree at 96%, KNN at 90% and Logistic Regression at 72%. Random Forest emerge as the best model for prediction, given their highest accuracy, recall, precision, and F1 score. They are effective at distinguishing between different types of connections and minimizing both false positives and false negatives. On the other hand, the Decision tree model also performs well, with only a small difference compared to the KNN model. Ultimately, the choice of the best prediction model may depend on several factors, such as training time, prediction time, model complexity, and other practical considerations.

Evaluation Metrics	Benign	Malicious	Outlier
Precision	100%	87%	66%
F1-Score	100%	90%	58%
Recall	100%	92%	51%
Accuracy	90%		

Table 7: KNN model classifier

Random Forest Classifier

Random Forest has of highest accuracy at 98%. According to the table below, this model's accuracy for the class representing connection types is 98%. The accuracy differs for outlier, benign, and malicious connection types. With a precision of 97% for malicious and 95% for outlier connections, a large proportion of positive cases are correctly identified. Since the primary goal of anomaly detection is to identify anomalies, the recall rate for both outlier and malicious connections is at 88%. Therefore, this model is effective at detecting anomalies. Finally, the F1 scores are 98% for malicious connections and 84% for outliers, indicating that the model has slightly lower recall and precision performance compared to the Decision Tree model.

Evaluation Metrics	Benign	Malicious	Outlier
Precision	100%	97%	95%
F1-Score	100%	98%	91%
Recall	100%	99%	88%
Accuracy	98%		

Table 8: Random Forest classifier

Logistic Regression Classifier

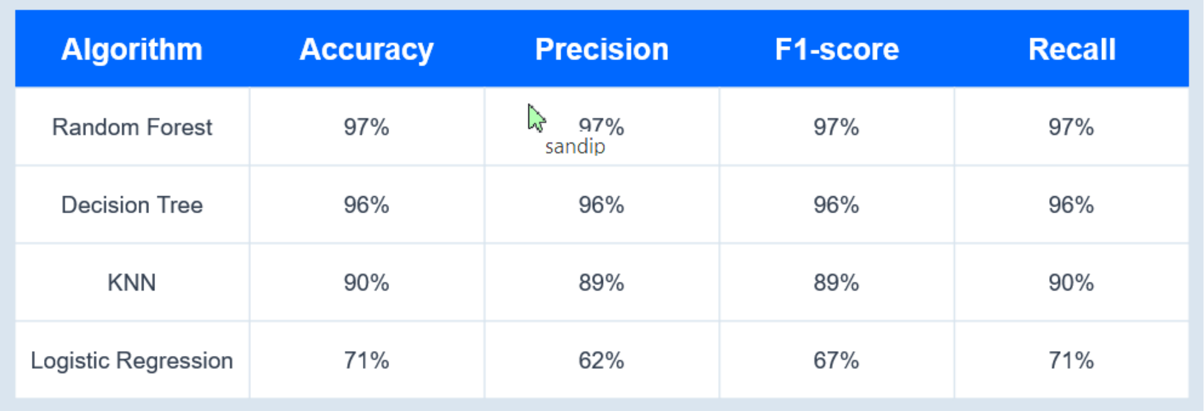
Compared to the Random Forest, the accuracy of Logistic Regression is much lower at 72%. According to the chart above, this model's accuracy for the class representing connection types is 72%. The accuracy differs for normal and anomalous connection types. With a precision of 67% for anomalies and 78% for normal connections, a large proportion of positive cases are correctly identified. Since the primary goal of anomaly detection is to identify anomalies, the recall rate for normal connections is 85%, while for anomalies it is 81%. Therefore, this model is effective at detecting anomalies. Finally, the F1 scores are 81% for normal connections and 73% for anomalies, indicating that the model has slightly lower recall and precision performance compared to the Decision Tree model.

Evaluation Metrics	Benign	Malicious	Outlier
Precision	78%	67%	0%
F1-Score	81%	73%	0%
Recall	85%	81%	0%

Accuracy	72%

Table 9: Logistic Regression classifier

Evaluation of following matrix



Algorithm	Accuracy	Precision	F1-score	Recall
Random Forest	97%	97%	97%	97%
Decision Tree	96%	96%	96%	96%
KNN	90%	89%	89%	90%
Logistic Regression	71%	62%	67%	71%

Confusion Matrix

Decision Tree

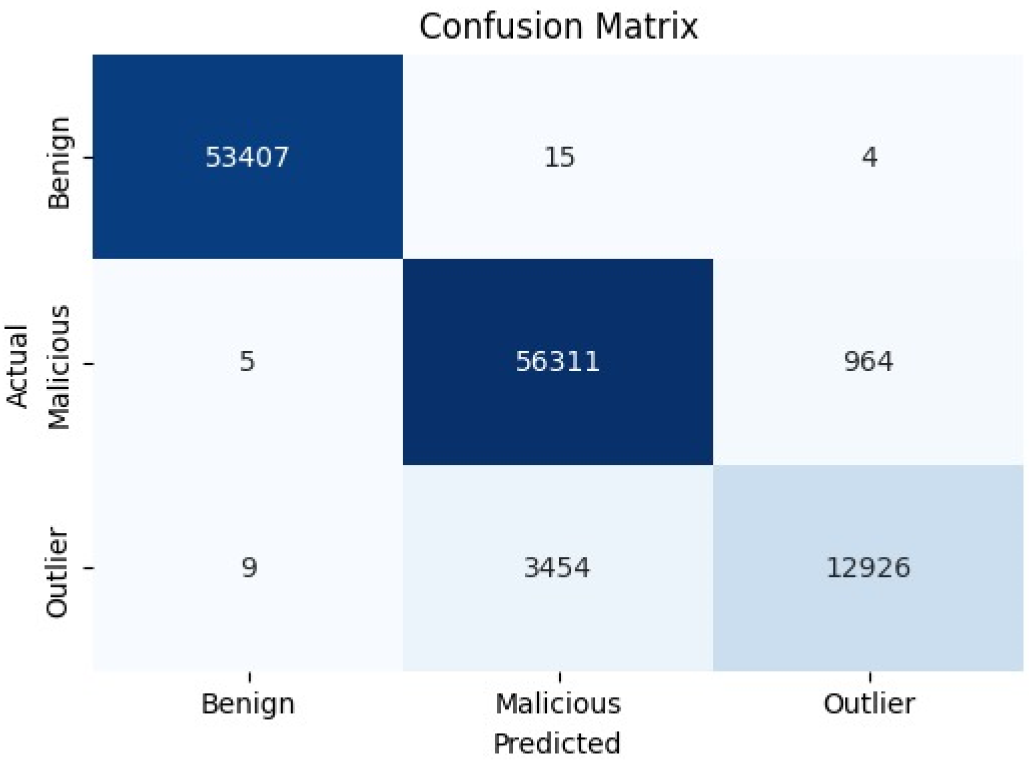


Figure 26: Decision Tree confusion matrix

The Decision Tree model's performance, evaluated using a confusion matrix, showcases its effectiveness in predicting test data across normal and anomaly classes. Out of a total of 821495 samples, the model misclassified 5 instances of the normal or benign class as risky, which are considered false positives. Conversely, 15 samples from the malicious class were incorrectly identified, constituting false negatives. In the case of the outlier, only 4 samples was predicted to be normal while 964 samples were identified as malicious. Notably, the model accurately classified 53407 samples as normal (true negatives) and 38206 samples as malicious (true positives). Overall, the model demonstrates a strong capability to correctly identify both benign and malicious samples, exhibiting a high percentage of accurate predictions while maintaining a low rate of misclassification for both false positives and false negatives.

Logistic Regression:

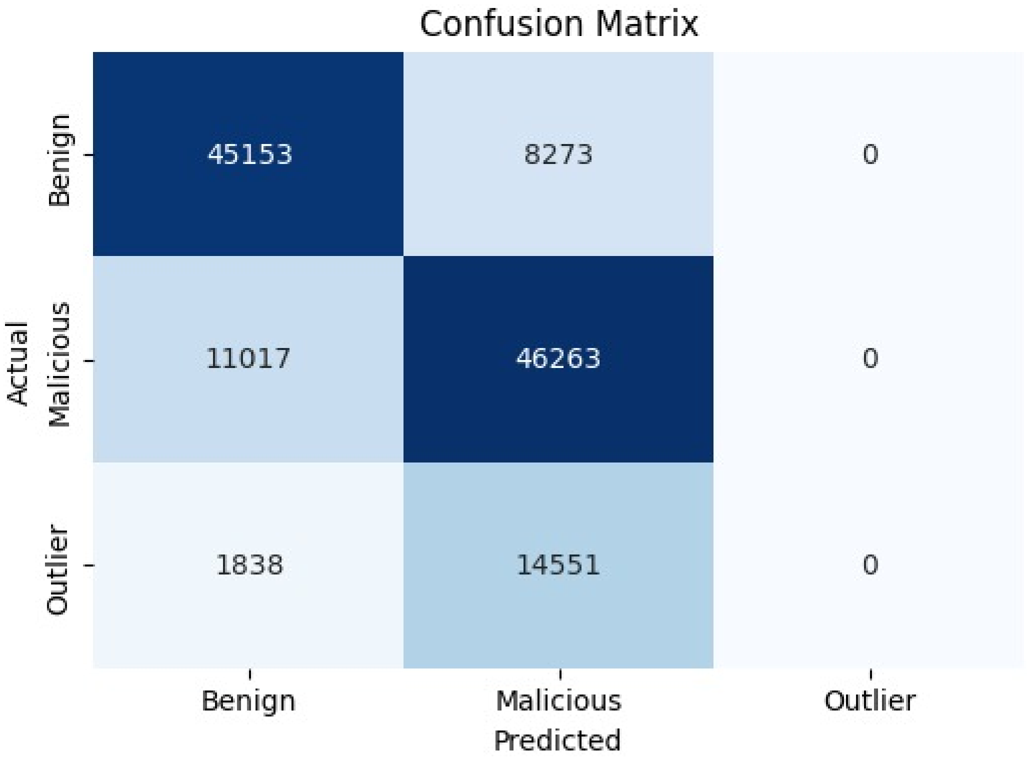


Figure 27: Confusion Matrix for logistic regression

The correctness of the benign and malicious connection samples is assessed using a confusion matrix, where the model identified 45153 true positives and 46263 true negatives. However, 11017 benign connections were incorrectly classified as malicious, representing false positives, while 8273 malicious samples were wrongly predicted as normal, leading to false negatives. Although the model successfully identified 46263 test samples as malicious (true negatives), the presence of a considerable number of false positives and false negatives indicates significant misclassification issues. This results in suboptimal overall model performance, highlighting the need for further improvements.

K-Nearest Neighbour:

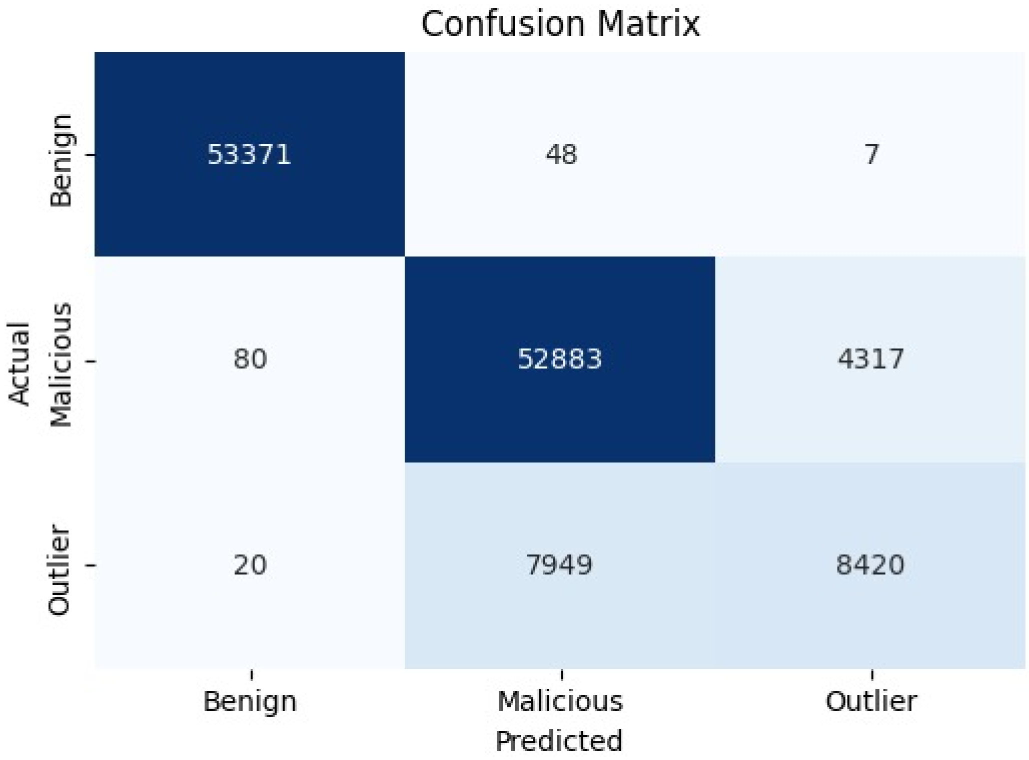


Figure 28:

Confusion matrix on KNN

Among the total test samples, 53371 benign connections were correctly classified as true positives. Additionally, 52883 samples were accurately identified as malicious connections, representing true negatives. However, 80 normal connection samples were incorrectly classified as malicious, indicating false positives. Furthermore, 48 samples of malicious connections were mistakenly categorized as normal, resulting in false negatives. Despite correctly identifying a significant number of both benign and malicious samples, the presence of these misclassifications suggests that the model's performance could benefit from further optimization.

Random Forest

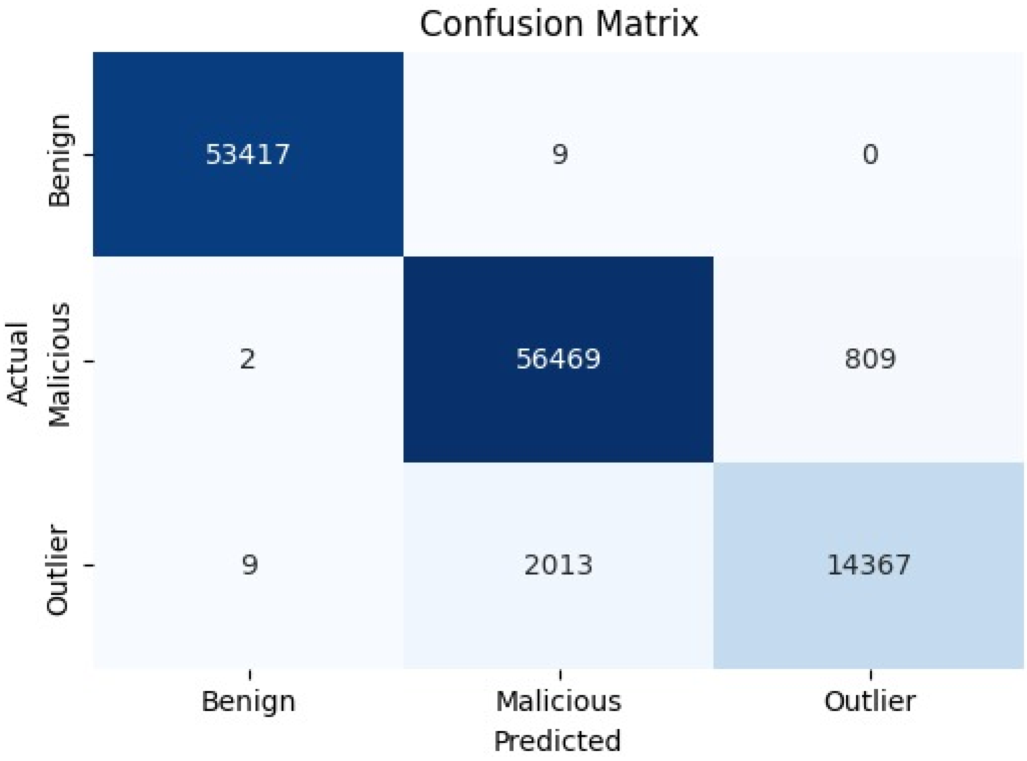


Figure 29: Confusion Matrix for random forest

Among the total test samples, 53417 benign connections were correctly classified as true positives. Additionally, 56469 samples were accurately identified as malicious connections, representing true negatives. However, 2 normal connection samples were incorrectly classified as malicious, indicating false positives. Furthermore, 9 samples of malicious connections were mistakenly categorized as normal, resulting in false negatives. Despite incorrectly identifying a significant number of outlier samples, the presence of these precise classifications suggests that the model's performance is the best overall in comparison to other algorithms.

Overall, the model has higher percentages of true positive and true negative values than false positive and false negative values. As a prediction model, it is seen to be doing well.

Validation loss and training loss

A Cross validation measures how well the model narrows the gap between the actual values observed in the training dataset and the results it predicts. The training loss, to put it simply, demonstrates how well the model's predictions match the predicted outcomes during the training phase. The validation loss, which evaluates how effectively the model generalizes to new, untested data, is another sign of a model's performance. A separate validation dataset that is not used during training, evaluates the discrepancy between real values and the model's predictions. Information regarding the model's likelihood to perform well on new data that it has never encountered is provided by the validation loss. By demonstrating how effectively the model can forecast results from fresh data, the validation loss gives an evaluation of the model's overall performance on untrained data.

In machine learning, three basic scenarios might occur during model training and affect both the training loss and the validation loss. These situations are perfectly, excessively, and inadequately suitable. The model is said to have been perfectly fitted when both the training loss and the validation loss are negligible. This scenario assumes that the model has learned the basic relationships and patterns in the training data and is capable of generalizing effectively to new data. A model's performance is said to be optimal when it achieves this balance between minimum training and validation losses. Overfitting occurs when a model excels on the training data, as shown by a considerably low training loss, but fails to generalize to new data, as shown by a greater validation loss. In other words, the model has memorized the training data too thoroughly and has picked up noise or irrelevant patterns. It finds it difficult to generate precise predictions based on unreliable data as a result. When a model is overly complicated or trained on insufficient data, overfitting can happen. The inability to perceive the underlying relationships and patterns in the training data causes a model to underfit, which results in significant training and validation losses. In this situation, the model is either too simple or unable to accommodate the complexity of the data. Its inability to generalize is indicated by its poor performance on both the training set and new data. Underfitting can be beaten by utilizing a more complex model or lengthening the training time.

When a model is trained on a known dataset, it often performs better and fits the training data better. It is expected that because the model was trained on data that is completely foreign to it, the validation data won't instantly match or generalize well to it. Simply said, because the model is trained without access to the validation data, it is typical for it to struggle at first to accurately predict or match the validation data. The model should perform better on the validation data as it develops, which should result in a reduced difference between the model's predictions on the training and validation data.

Here are some figures of different models having test and training accuracy

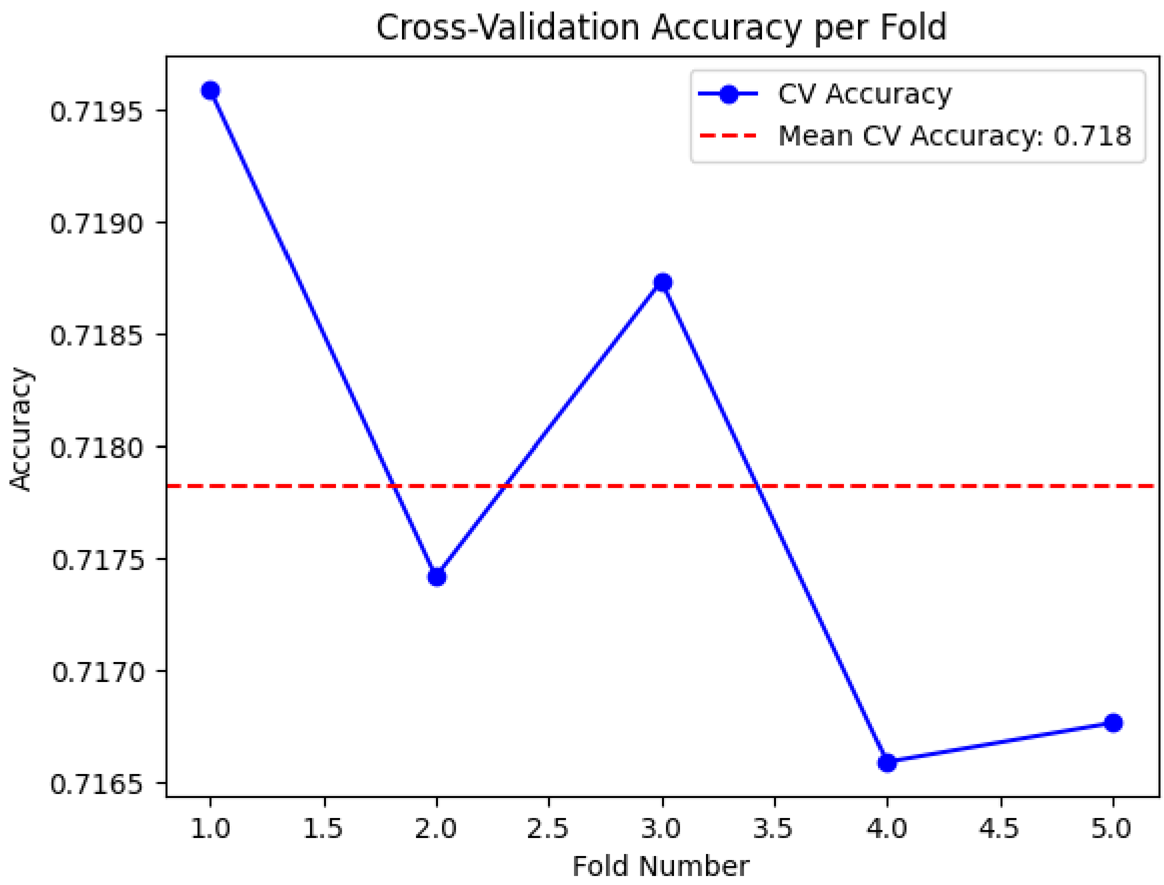


Figure 30: Cross Validation accuracy per fold

Evaluation

The optimal machine learning model is decided by several factors. Wherein some of the factors, such as Fit Time, Predict Time, or Training Time, confusion matrix, accuracy, precision, F1 score, and so on, are previously discussed. In addition, certain more distinct characteristics must be considered. All the models are examined and analyzed in this part based on a variety of criteria, including their performance and accuracy.

Cross Validation

Cross-validation may help assess if one approach consistently outperforms another or whether their results are comparable across multiple data divisions. To evaluate the relative performance of the two algorithms, the assessment metrics obtained from each iteration of each method may be compared using cross-validation. This technique analyses the performance throughout many data splits rather than relying just on the train-test split, enabling a more accurate and equitable comparison. Therefore, cross-validation aids in selecting the appropriate method and determining whether there is a significant performance difference between the algorithms.

Models	Mean Score	Cross Validation
Logistic Regression	60.11%	0.60216048, 0.59102637, 0.59017579, 0.60734052, 0.60498015
Random Forest	96.14%	0.96315784, 0.96123681, 0.95784219, 0.95684915, 0.95847231
Decision Tree	95.46%	0.95432946, 0.95461299, 0.95489651, 0.95489651, 0.95404424
K-Nearest Neighbour	90.53%	0.90355543, 0.90072016, 0.90582365, 0.90213779, 0.90411798

Table 10: Comparison of models using cross-validation

Logistic Regression

Here, we select five cross-validation scores for each model as shown in the table above. This model's typical score is 94.11%. This model is not as effective in detecting intrusion from the datasets and targeted variables as other models used for prediction, as shown by the fact that all five cross-validation scores are less than 0.95, which may be regarded as poorer than other models used for prediction.

Decision Tree

We can observe from the above table that this model performs well. It has a value of more than 99% and a mean score of 99.46% after doing five cross-validations, indicating strong performance consistency and efficacy. This suggests that the model is not over-fitted and that it can function well with new data.

K-Nearest Neighbour (KNN)

We can observe from the above table that this model performs fairly well. It has a value of more than 99% and a mean score of 99.46% after doing five cross-validations, indicating strong performance consistency and efficacy. This suggests that the model is not over-fitted and that it can function well with new data.

F1-Score

Models	F1-Score
Decision Tree	96
Logistic Regression	67
K-Nearest Neighbour	90
Random Forest	98

Table 11: F1-Score of models

Another way to assess the effectiveness or performance of the model is to use the F1-score. It combines the ideas of precision and recall into a single score to give a fair evaluation of the model's accuracy. According to the above table, Random Forest performed better than the other models, scoring 98, followed by Decision Tree with 96, KNN with 90, and lastly, Logistic Regression with 67, which had the lowest score. The graphic below shows a bar graph of the f1-score.

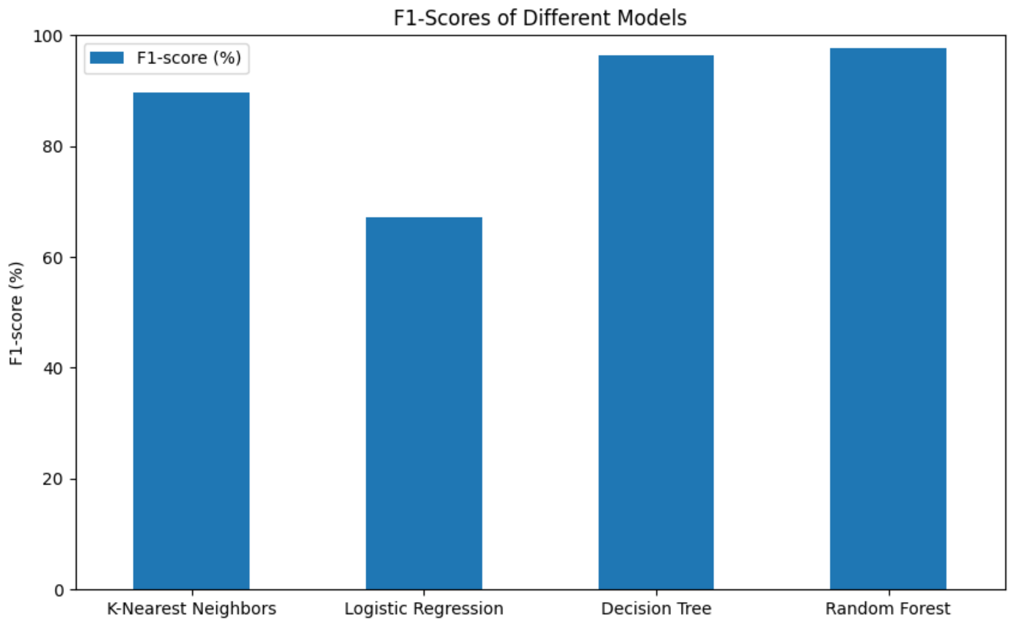


Figure 31: F1-Score

User Interactive frontend

To make the evaluation and prediction of the dataset accurate and easy to use for the general user, a user-friendly representation of the project is produced. For the graphical user interface, we will use Python and libraries like Tkinter, pandas, and Seaborn. The application gives users the option users to upload train and test datasets, train machine learning models, and visualize the results of predictions in an interactive environment.

GUI Interface and layout

The app consists of main windows and its layout is defined using Tkinter for interfaces. It also consists of features like adjustment dynamically as per user preference. It consists of a widows button including uploading datasets, training models, predicting results, and lastly option for closing the application.

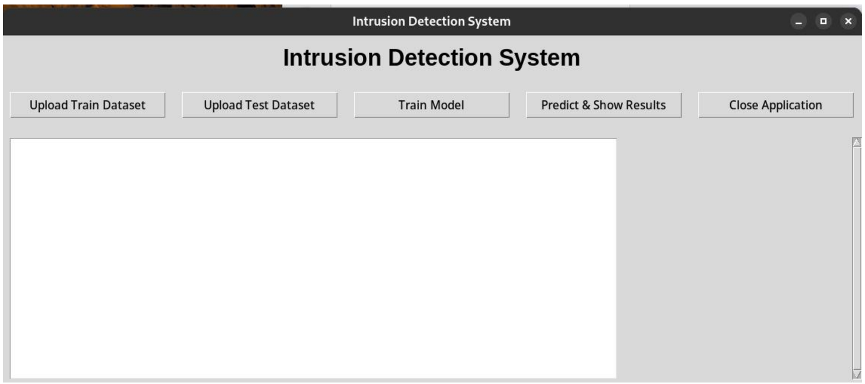
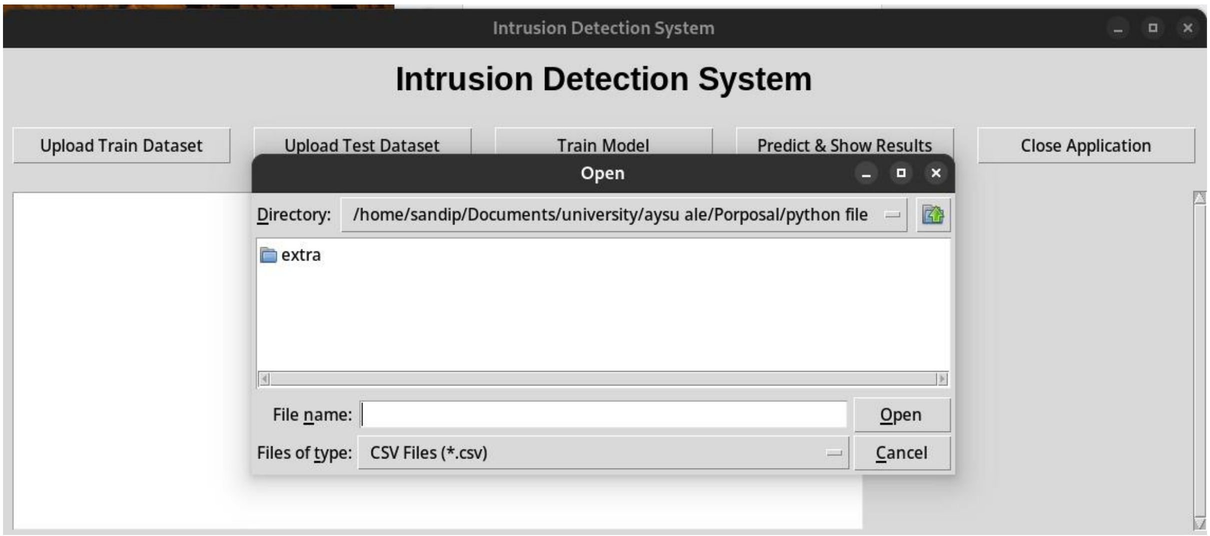


Figure 32: GUI for IDS

It consist of Button for uploading train model as well as test model separately, which opens door to making the model more robust.



Directory: /home/sandip/Documents/university/aysu ale/Porposal/python file

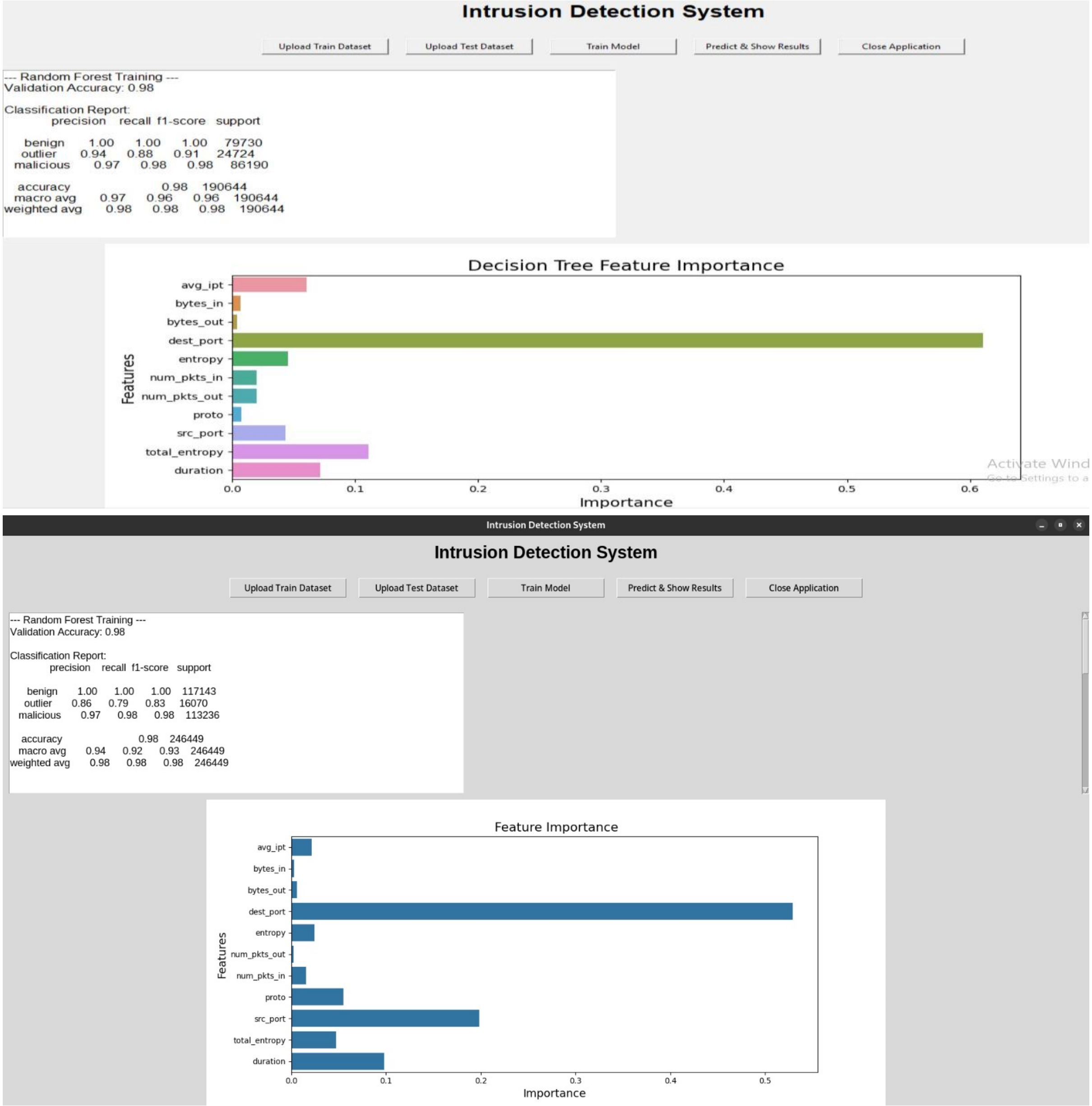
File name:

Files of type: CSV Files (*.csv)

Figure 33: Uploading of the dataset in the Application

Prediction and Visualization

After the datasets are uploaded and set, the model is trained witch clicking the train model button. We are training the model in Random Forest, Decision Tree and K-Nearest Neighbor algorithm which also displays along with its respective accuracy and evaluation metrics.



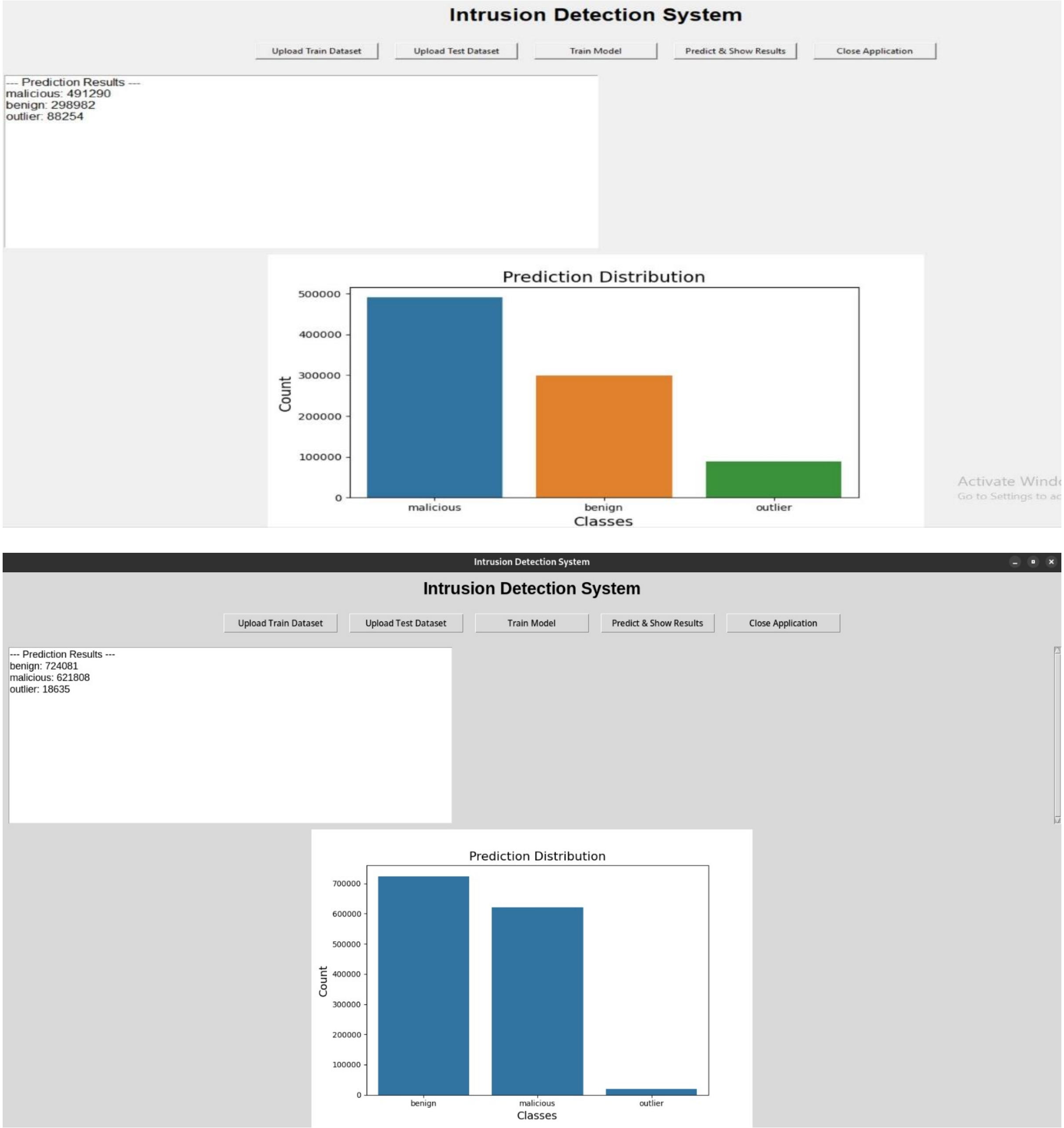
```
--- Random Forest Training ---
Validation Accuracy: 0.98

Classification Report:
           precision    recall  f1-score   support

   benign       1.00      1.00      1.00     79730
  outlier       0.94      0.88      0.91     24724
malicious       0.97      0.98      0.98     86190

 accuracy                           0.98    190644
macro avg       0.97      0.96      0.96    190644
weighted avg    0.98      0.98      0.98    190644
```

```
--- Random Forest Training ---
Validation Accuracy: 0.98

Classification Report:
           precision    recall  f1-score   support

   benign       1.00      1.00      1.00    117143
  outlier       0.86      0.79      0.83     16070
malicious       0.97      0.98      0.98    113236

 accuracy                           0.98    246449
macro avg       0.94      0.92      0.93    246449
weighted avg    0.98      0.98      0.98    246449
```

Figure 34: Trained model in IDS app

Prediction

The trained random forest model is used for prediction on the test dataset. The result counts the predicted value for benign, outlier, and malicious and is also outputted in the bar plot. The bar plot represents the predicted result of the trained model against the test dataset that which user has uploaded.



```
--- Prediction Results ---
malicious: 491290
benign: 298982
outlier: 88254
```

```
--- Prediction Results ---
benign: 724081
malicious: 621808
outlier: 18635
```

Figure 35: Prediction Result

Chapter 5: Conclusion And Future Implications

IDS

An Intrusion Detection System (IDS) serves as a proactive and efficient network security solution by utilizing machine learning techniques and predictive analytics to identify and respond to potential threats in real time. IDS can significantly outperform traditional security measures by enhancing threat detection, enabling quicker responses, and providing greater resilience against sophisticated attacks.

IDS employs advanced techniques such as anomaly detection, behavioral analysis, and pattern recognition to identify deviations from normal network behavior. By analyzing vast volumes of network data, IDS can effectively detect both known and unknown threats, including zero-day attacks and adaptive malware. The integration of machine learning allows the IDS to continuously learn from emerging threats, improving its detection accuracy over time.

Moreover, IDS enhances incident response effectiveness by reducing detection errors and providing comprehensive information about identified threats. It generates alerts based on detected anomalies, enabling security professionals to prioritize and investigate potential security concerns promptly. When integrated with existing security frameworks, an IDS can enforce dynamic security rules based on real-time risk assessments, thereby significantly reducing the impact of attack vectors.

Additionally, IDS offers valuable logging, auditing, and reporting functionalities that assist in forensic analysis, compliance requirements, and ongoing security policy enhancement. Organizations can leverage the data collected by IDS to gain insights into attack patterns, identify trends, and refine their security strategies to better defend against emerging threats.

To implement an IDS as a predictive model, businesses can utilize a variety of open-source and commercial solutions available in the market. Notable IDS applications include Snort, Suricata, and Bro/Zeek. These technologies facilitate the deployment of IDS as a predictive model that can seamlessly integrate with existing security systems.

However, it is important to note that the effectiveness of an intrusion detection system can be compromised by noise. A high false alarm rate may arise from corrupt DNS data, erroneous packets generated by software bugs, or local packets that evade detection. Consequently, the number of actual attacks is often significantly lower than the volume of false alarms generated by the system.

Future Work and Implication

The integration of predictive modeling techniques into Intrusion Detection Systems (IDS) will have a huge influence on future network security. Here are a few possible outcomes in the future.

- Advance Threat Detection: IDS-based firewalls that use predictive modeling can identify complex and newly developing threats. Machine learning algorithms can analyze massive amounts of data, see trends, and spot abnormalities that might indicate targeted assaults, zero-day attacks, or advanced persistent threats (APTs).
- Behavioural analysis
- Threat Intelligence Integration
- Autonomous Response
- Cloud and IOT security

Summary

Intrusion Detection Systems (IDS) based on machine learning play a crucial role in enhancing cybersecurity by identifying and responding to unauthorized access and various cyber threats. These systems complement traditional firewalls by implementing advanced predictive modeling techniques that improve threat detection capabilities.

An IDS continuously analyses network traffic, applying machine learning algorithms to recognize patterns and anomalies that may indicate potential intrusions or malicious activity. This proactive filtering reduces the likelihood of successful attacks, protecting sensitive data and maintaining system integrity. By effectively distinguishing between normal and suspicious behavior, an IDS can significantly lower the risk of data breaches.

Moreover, IDS can facilitate network segmentation, which enhances security by limiting lateral movement within the network. This division helps contain potential damage if an attacker breaches one segment, thereby safeguarding critical assets.

In conclusion, integrating machine learning-based IDS with existing firewall technologies strengthens an organization's overall security posture. It enables quicker identification and response to complex and evolving threats, ultimately improving the protection of connections, confidential data, and electronic assets in today's dynamic threat landscape.

References

algodaily, n.d. *Getting to Know Decision Trees.* [Online]
Available at: https://algodaily.com/lessons/decision-trees-basics
[Accessed 27 April 2023].

Alsmadi, I., Al-Hubaishi, A., Al-Mansoori, A. & Al-Hinai, A., 2017. Industrial control systems security and firewalls. *Journal of Information Security and Applications,* Volume 35, pp. 15-35.

Aronson, J. P., Brownlee, N., Byrum, F. & Ramsey, M. S., 1997. *Site Security Handbook.* [Online]
Available at: https://www.ietf.org/rfc/rfc2196.txt?
[Accessed 01 November 2022].

Arsene, L., 2015. *The Evolution of Firewalls: Past, Present & Future.* [Online]
Available at: https://www.informationweek.com/partner-perspectives/the-evolution-of-firewalls-past-present-and-future/a/d-id/1318814?
[Accessed 02 November 2022].

Aruba, n.d. *How a next-generation firewall works.* [Online]
Available at: https://www.arubanetworks.com/faq/what-is-next-gen-firewall/
[Accessed 12 November 2022].

Atawodi, I. S., 2019. *A MACHINE LEARNING APPROACH TO NETWORKINTRUSION DETECTION SYSTEM USINGK NEAREST NEIGHBOR AND RANDOM FOREST,* University of Southern Mississippi: s.n.

Atlassian, n.d. *The Agile Coach.* [Online]
Available at: https://www.atlassian.com/agile
[Accessed 20 November 2022].

BHOSALE, S., 2019. *Network Intrusion Detection.* [Online]
Available at: https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection
[Accessed 02 April 2022].

Biggio, B., 2012. Poisoning Attacks against Support Vector Machines. *In Proceedings of the 29th International Conference on Machine Learning.*

Braden, R., . D. C., S. C. & . C. H., 1994. Report of IAB Workshop on security in the Internet architecture. Feburary 8-10.

Briesemeister, L. R., Varma, A. L. & Thuraisingham, B., 2015. A Survey of Machine Learning Techniques for Cybersecurity. *ACM Computing Surveys,* 48(02).

Carvalho, J. C., de Araújo, A. A. L. & Dantas, M. A. R., 2011. Intrusion Detection Systems: A Machine Learning Approach Based on Principal Component Analysis and Artificial Neural Networks. *Expert Systems with Applications,* 38(1), pp. 235-243.

Cheswick, W. R. & Bellovin, S. M., 1994. *Firewalls and Internet Security: Repelling the Wily Hacker.* s.l.:Addison-Wesley..

Cheswick, w. R., Bellovin, S. M. & Rubin, A. D., 1994. *Firewalls and Internet Security Repelling the Wily Hacker.* 02 ed. Addison-Wesley: One Jacob Way.

Cheswick, W. R., B. S. M. & R. A. D., 2003. Firewalls and Internet Security:.

Chollet, F., 2018. *Deep learning with Python.* s.l.:Manning Publications..

Choon, T. L. & Hussin, K., 2011. Malaysian 3D Property Legislation-A Preliminary Approach. *nternational SurveyingResearch Journal,* 1(1), pp. 30-31.

CISCO, n.d. *Cisco Adaptive Security Appliance (ASA).* [Online]
Available at: https://www.cisco.com/c/en/us/products/security/asa-firepower-services/index.html
[Accessed 22 April 2023].

Cisco, n.d. *What is a Firewall?.* [Online]
Available at: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html
[Accessed 18 April 2023].

Corbitt, T., 2002. Protect your computer system with a security policy. *Management Services,* 46(2), pp. 20-21.

Danchev, D., 2003. *Building and Implementing a Successful Information Security Policy.* [Online]
Available at: https://techgenix.com/building_implementing_security_policy/
[Accessed 01 November 2022].

Debar, H., Dacier, M. & Wespi, A., 1999. Towards a taxonomy of intrusion-detection systems. *Computer Networks,* 31(23-24), pp. 805-822.

Deshpande, C., 2022. *What Is Firewall: Types, How Does It Work, Advantages & Its Importance.* [Online]
Available at: https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-firewall#:~:text=Firewalls%20are%20network%20security%20systems,in%20enterprise%20and%20personal%20settings.
[Accessed 27 November 2022].

Driscoll, M., n.d. *Jupyter Notebook: An Introduction.* [Online]
Available at: https://realpython.com/jupyter-notebook-introduction/
[Accessed 02 December 2022].

Fortinet, n.d. *Features of a Next-Generation Firewall.* [Online]
Available at: https://axiansuk.co.uk/fortinet?utm_campaign=Paid%20Search%20-%20Fortinet%202022&utm_source=ppc&utm_medium=google&utm_term=fortigate%20security&utm_campaign=Fortinet+-

+2022&utm_source=adwords&utm_medium=ppc&hsa_acc=1045064206&hsa_cam=18536284029&hsa_grp=1
[Accessed 24 April 2023].

García, S., Díaz-Verdejo, . J. & Riquelme, J. C., 2012. Evolutionary Intrusion Detection. *IEEE Transactions on Evolutionary Computation,* 16(2), pp. 206-219.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G. & Vázquez, E., 2009. Anomaly-based network intrusion detection. *Techniques, systems and challenges. Computers & Security,* 28(2), pp. 18-28.

Garcia-Teodoro, P., D.-V. J., M.-F. G. & V. E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security,* 28(1-2), pp. 18-28.

Garfinkel, S., Rosenblum, M. & Adams, K., 2018. Web application firewalls: A critical evaluation.. *IEEE Security & Privacy,* 16(3), pp. 18-26.

Goodfellow, I., 2016. *Deep Learning. In Deep Learning.* s.l.:MIT Press.

Hastie, T., T. R. & F. J., 2009. The elements of statistical learning: data mining, inference, and prediction. *Springer Science & Business Media.*

Ho, T. K., 1995. *Random Decisio.* s.l.:Bell Labs.

Howard, 2021. *What Is a Next-Generation Firewall?.* [Online]
Available at: https://community.fs.com/blog/what-is-a-next-generation-firewall.html
[Accessed 28 April 2023].

Hub, K., 2022. *How much does a firewall cost.* [Online]
Available at: https://exa.net.uk/how-much-does-a-firewall-cost/#:~:text=A%20firewall%20can%20cost%20anywhere,look%20at%20the%20different%20models.
[Accessed 20 April 2023].

IBM, n.d. *K-Nearest Neighbors Algorithm.* [Online]
Available at: https://www.ibm.com/topics/knn
[Accessed 28 April 2023].

Juniper, 2022. *What is IDS and IPS?.* [Online]
Available at: https://www.juniper.net/gb/en/research-topics/what-is-ids-ips.html
[Accessed 01 April 2023].

Kokel, H. & Sachdeva, N., 2020. A Survey of Intrusion Detection Systems Using Machine Learning Techniques.

Kolias, C., K. G., . S. A. & V. J., 2017. Intrusion detection in 21st century: A survey. *Journal of Network and Computer Applications,* Volume 75, pp. 303-322.

Lab, K., 2018. *Targeted attacks: A true menace.* [Online]
Available at: https://media.kaspersky.com/en/enterprise-

security/Cyber_Threat_Evolution_2018_Final.pdf
[Accessed 18 December 2022].

Leaf, n.d. *What is a Cyber Attack.* [Online]
Available at: https://leaf-it.com/10-ways-prevent-cyber-attacks/
[Accessed 06 October 2022].

learn, S., n.d. *sklearn.linear_model.LogisticRegression.* [Online]
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[Accessed 28 April 2023].

Macfarlane, R., 2009. *An Integrated Firewall Policy Validation Too.* [Online]
Available at: https://napier-repository.worktribe.com/output/224845/an-integrated-firewall-policy-validation-tool
[Accessed 01 November 2022].

matplotlib, 2022. *Getting started.* [Online]
Available at: org/stable/users/getting_started/index.html
[Accessed 25 April 2023].

McCombes, S., 2019. *Scribbr.* [Online]
Available at: https://www.scribbr.com/research-process/research-question-examples/
[Accessed 29 October 2022].

Mehmood, A., Ahmad, I., Khattak, A. M. & Aslam, B., 2018. Distributed Denial of Service (DDoS) attack mitigation using firewalls. *Journal of Computers,* 13(7), pp. 794-802.

Mirjalili, S. & Gani, A., 2016. Intrusion Detection Systems Using Ensemble Techniques. *Journal of Network and Computer Applications,* Volume 60, pp. 82-105.

Moustafa, N. & Slay, J., 2015. The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 dataset. *Journal of Information Security and Applications,* Volume 27, pp. 1-14.

Muffett, A., 2000. *Proper Care and Feeding of Firewalls.* [Online]
Available at:
https://www.researchgate.net/publication/2382533_Proper_Care_and_Feeding_of_Firewalls
[Accessed 01 November 2022].

nitishkumarpatel191, 2022. *Hardware vs. Software Firewalls: A Guide for SMBs in 2022.* [Online]
Available at: https://www.geeksforgeeks.org/difference-between-hardware-firewall-and-software-firewall/
[Accessed 20 April 2023].

NumPy, n.d. *NumPy Documentation.* [Online]
Available at: https://numpy.org/doc/
[Accessed 26 April 2023].

pandas, 2022. *pandas documentation.* [Online]
Available at: https://pandas.pydata.org/pandas-docs/stable/
[Accessed 25 April 2023].

pandas, n.d. *Pandas.* [Online]
Available at: https://pandas.pydata.org/
[Accessed 05 April 2023].

point, J. t., n.d. *Agile Model.* [Online]
Available at: https://www.javatpoint.com/agile-vs-waterfall-model
[Accessed 22 November 2022].

Press., C., 2012. *Network Security First-Step: Firewalls.* [Online]
Available at: https://www.ciscopress.com/articles/article.asp?p=1823359&seqNum=7
[Accessed 18 April 2023].

PyPI, 2022. *Matplotlib.* [Online]
Available at: https://pypi.org/project/matplotlib/
[Accessed 05 April 2023].

Question pro, n.d. *Question pro.* [Online]
Available at: https://www.questionpro.com/blog/quantitative-data/#Quantitative_Data_Definition
[Accessed 08 November 2022].

Roesch, M., 1999. Snort—lightweight intrusion detection for networks. *In LISA,* Volume 99, pp. 229-238.

Sahay, M., 2022. *Who Invented the Firewall? History, Types, and Generations of Firewall..* [Online]
Available at: https://www.thepcinsider.com/who-invented-firewall-history-evolution-types-generations/
[Accessed 10 November 2022].

Scarfone, K. & Mell, P., 2007. Guide to intrusion detection and prevention systems (IDPS). *National Institute of Standards and Technology,* pp. 800-894.

Schimmel, J., 1997. A historical look at firewall technologies.. 22(7), p. Section 13.

Schneier, B., 2000. *Secrets and Lies: Digital Security in a Networked World.* s.l.:Wiley..

school, W., n.d. *Python Introduction.* [Online]
Available at: https://www.w3schools.com/python/python_intro.asp
[Accessed 30 November 2022].

Schuba, C. & Lodin, S., 1998. Firewalls fend off invasions from the Net. *IEEE Spectrum,* 35(2), pp. 26-34.

Schuba & Ludwig, C., 1997. *On the modeling, design, and implementation of firewall technology.* [Online]
Available at:
https://www.proquest.com/openview/d9c5f6732f0e337e56bb9a0b2835ea1f/1?pq-origsite=gscholar&cbl=18750&diss=y
[Accessed 06 November 2022].

scikit-learn, n.d. *Machine Learning in Python.* [Online]
Available at: https://scikit-learn.org/stable/
[Accessed 26 April 2023].

seaborn, n.d. *An introduction to seaborn.* [Online]
Available at: https://seaborn.pydata.org/tutorial/introduction
[Accessed 05 April 2023].

Security, N., 2021. *Palo Alto Networks Next-Generation Firewalls.* [Online]
Available at: https://www.paloaltonetworks.com/resources/videos/palo-alto-networks-ml-powered-next-generation-firewall
[Accessed 22 April 2023].

Sharafaldin, I., L. A. H. & G. A. A., 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP),* pp. 108-116.

Skowyra, R. & Seredynski, F., 2018. An intrusion detection system based on hybrid clustering and support vector machines for telecommunication networks. *Expert Systems with Applications,* Volume 113, pp. 24-34.

Sommer, R. & Paxson, V., 2010. Outside the Closed World: On Using Machine Learning For Network Intrusion Detection. *in Proceedings of the IEEE Symposium on Security and Privacy.*

Statistics how to, n.d. *Statistics how to.* [Online]
Available at: https://www.statisticshowto.com/primary-data-secondary/
[Accessed 09 November 2022].

Tan, T. H. & Kho, L. C., 2019. The effectiveness of firewall against phishing and spear-phishing attacks: A review. *International Journal of Advanced Computer Science and Applications,* 10(10), pp. 83-89.

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A., 2009. A Detailed Analysis of the KDD CUP 99 Data Set. *n Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications.*

TechTarget, n.d. *firewall.* [Online]
Available at: https://www.techtarget.com/searchsecurity/definition/firewall
[Accessed 18 April 2023].

W3schools, n.d. *Cyber Security Firewalls.* [Online]
Available at: https://www.w3schools.com/cybersecurity/cybersecurity_firewalls.php
[Accessed 14 November 2022].

Wang, W., S. M. & Y. L., 2017. Deep learning for network intrusion detection. *A survey. IEEE Communications Surveys & Tutorials,* 19(2), pp. 940-960.

Whitman, M. E. & Mattord, . H. J., 2018. *Principles of Information Security.Cengage Learning.* s.l.:s.n.

Xinzhou, H., 2021. Research on Computer Network Security Based on Firewall Technology. *Journal of Physics: Conference Series,* p. 1744.

Zerkalica, R., n.d. Intrusion Detection Systems with SnortAdvanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID. ISBN.

Zwicky, E. D., Cooper, S. & Chapman, D. B., 2000. *Building Internet Firewalls, 2nd Edition.* 02 ed. s.l.:O'Reilly Media, Inc..

Zwicky, E. D., Cooper, S. & Chapman, D. B., 2013. *Building Internet Firewalls.* 2 ed. s.l.:O'Reilly Media.

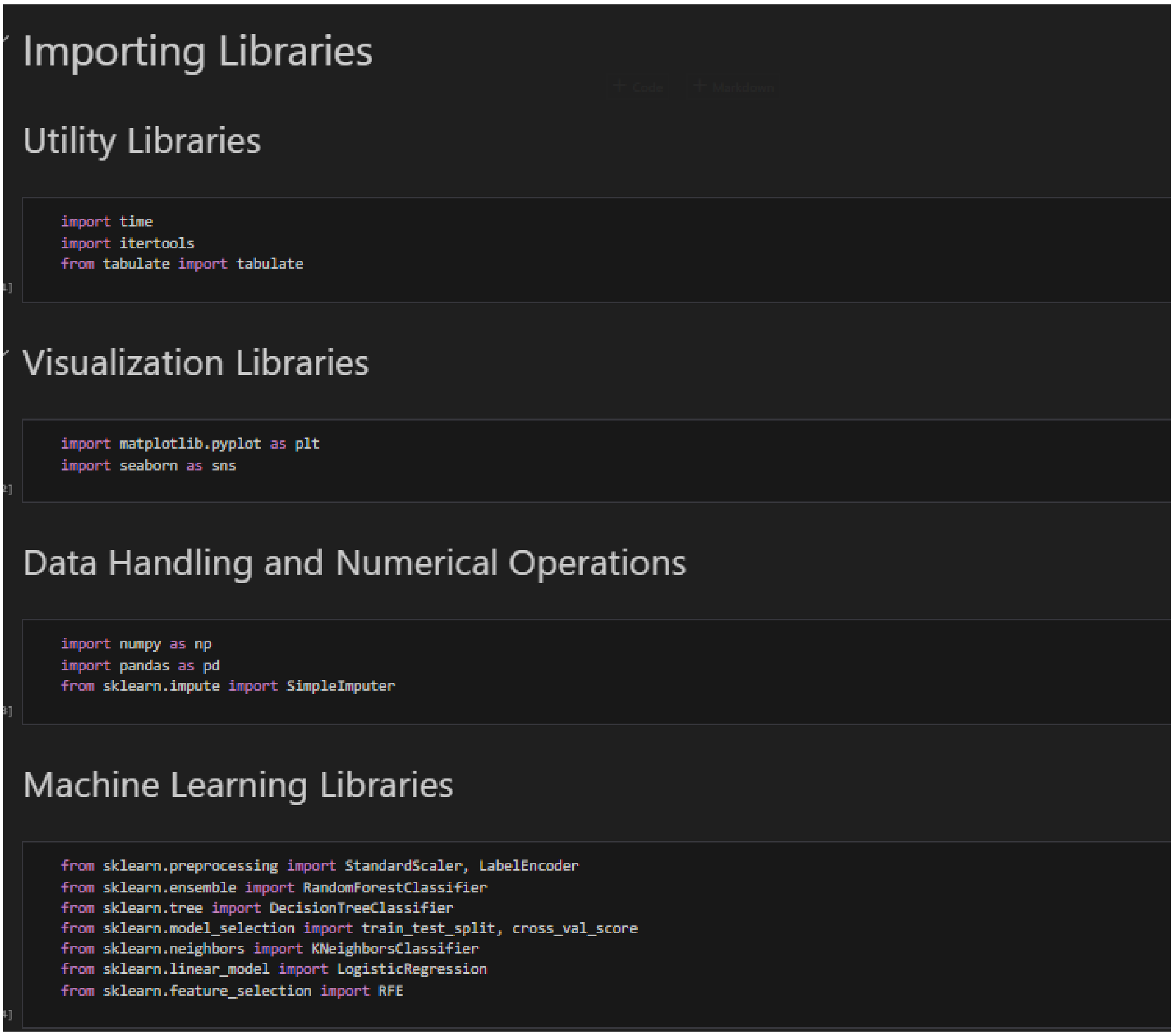
Appendixes

Appendix 1

<u>Datasets</u>: Train and test datasets open sources: https://www.kaggle.com/datasets/mryanm/luflow-network-intrusion-detection-data-set

Appendix 2

The prediction model code is attached in this section.



Importing Libraries

Utility Libraries

```
import time
import itertools
from tabulate import tabulate
```

Visualization Libraries

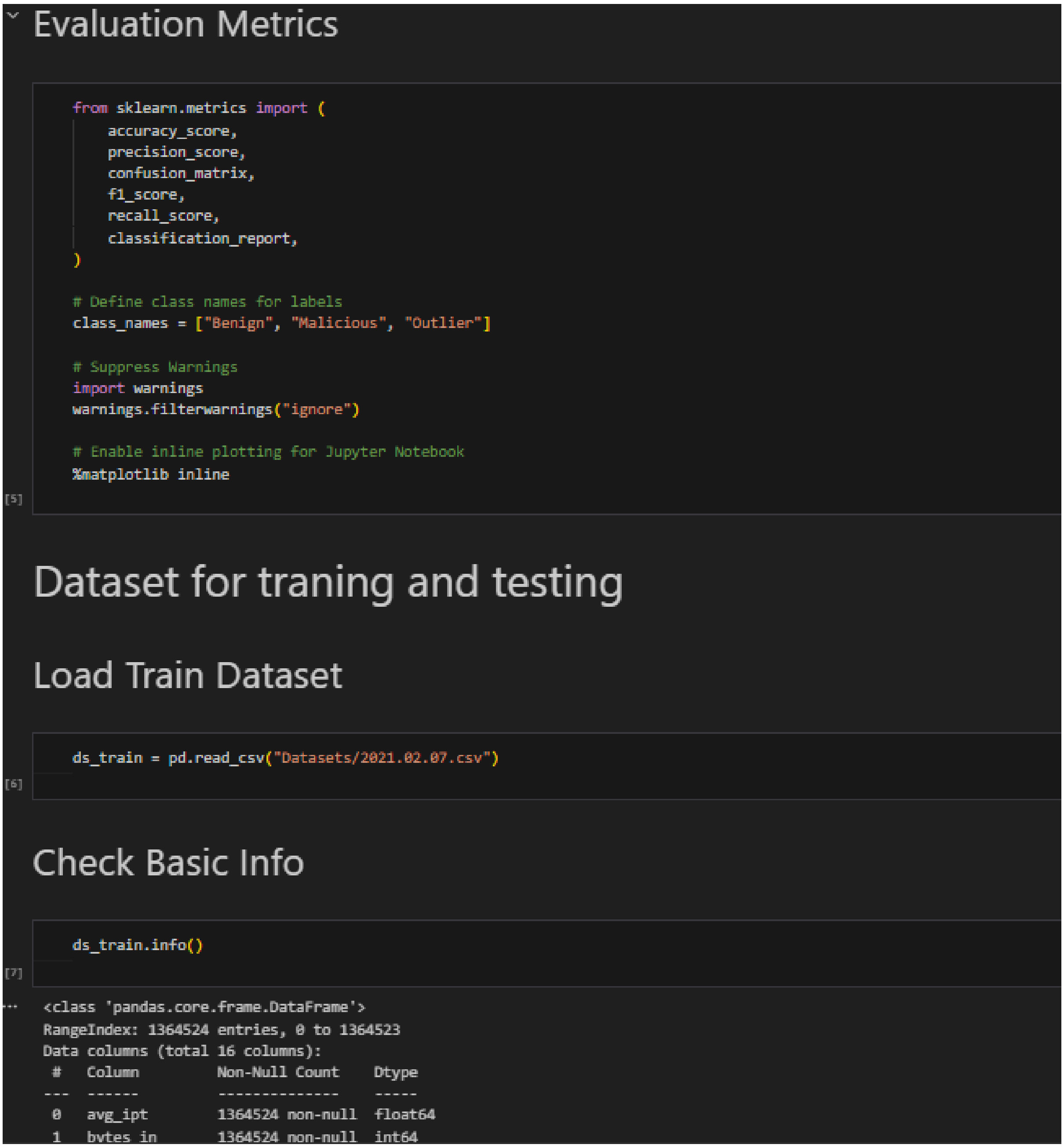
```
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Handling and Numerical Operations

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
```

Machine Learning Libraries

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
```



Evaluation Metrics

```
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    confusion_matrix,
    f1_score,
    recall_score,
    classification_report,
)

# Define class names for labels
class_names = ["Benign", "Malicious", "Outlier"]

# Suppress Warnings
import warnings
warnings.filterwarnings("ignore")

# Enable inline plotting for Jupyter Notebook
%matplotlib inline
```

[5]

Dataset for traning and testing

Load Train Dataset

```
ds_train = pd.read_csv("Datasets/2021.02.07.csv")
```

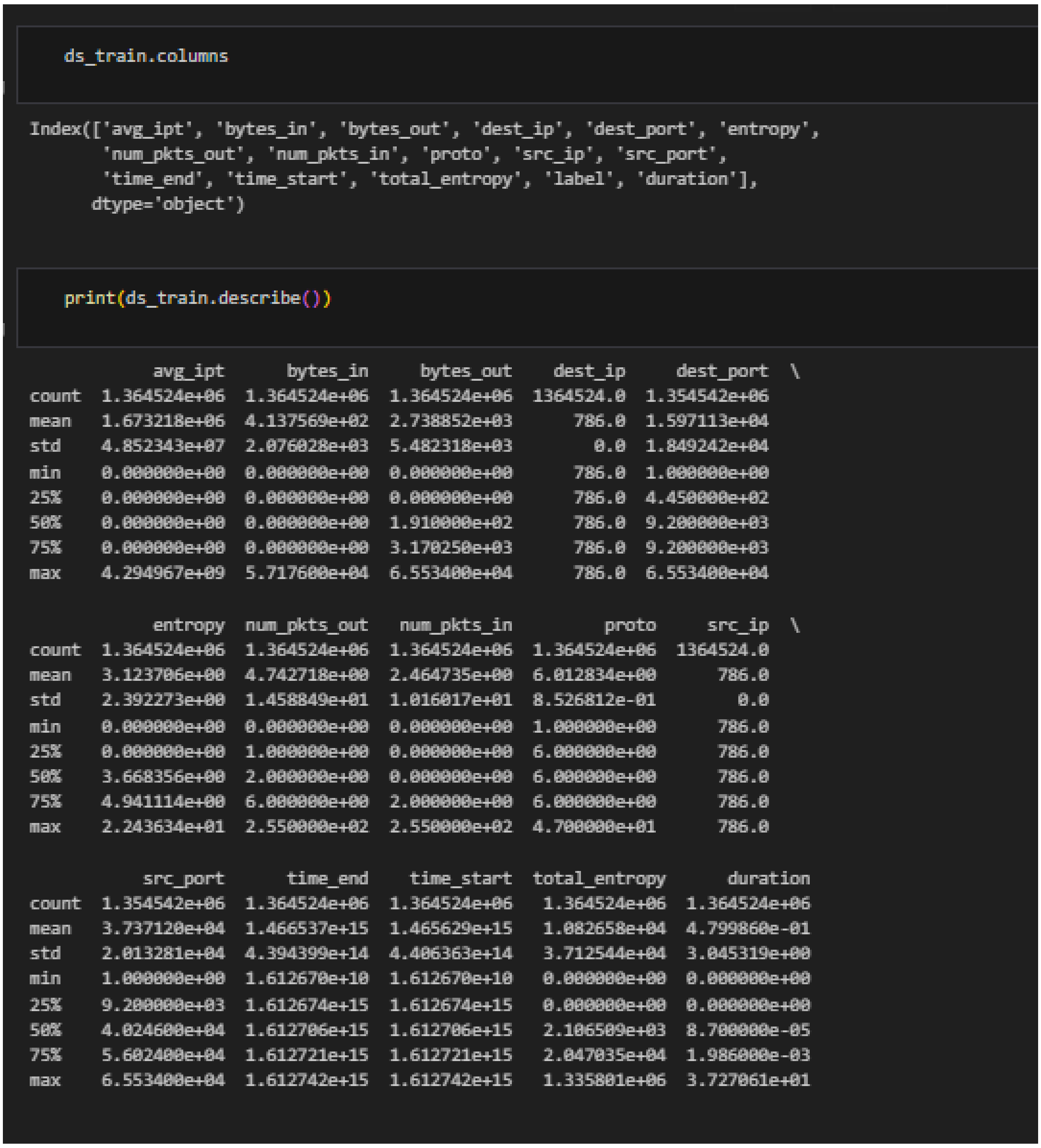
[6]

Check Basic Info

```
ds_train.info()
```

[7]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1364524 entries, 0 to 1364523
Data columns (total 16 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   avg_ipt        1364524 non-null  float64
 1   bytes_in       1364524 non-null  int64
```

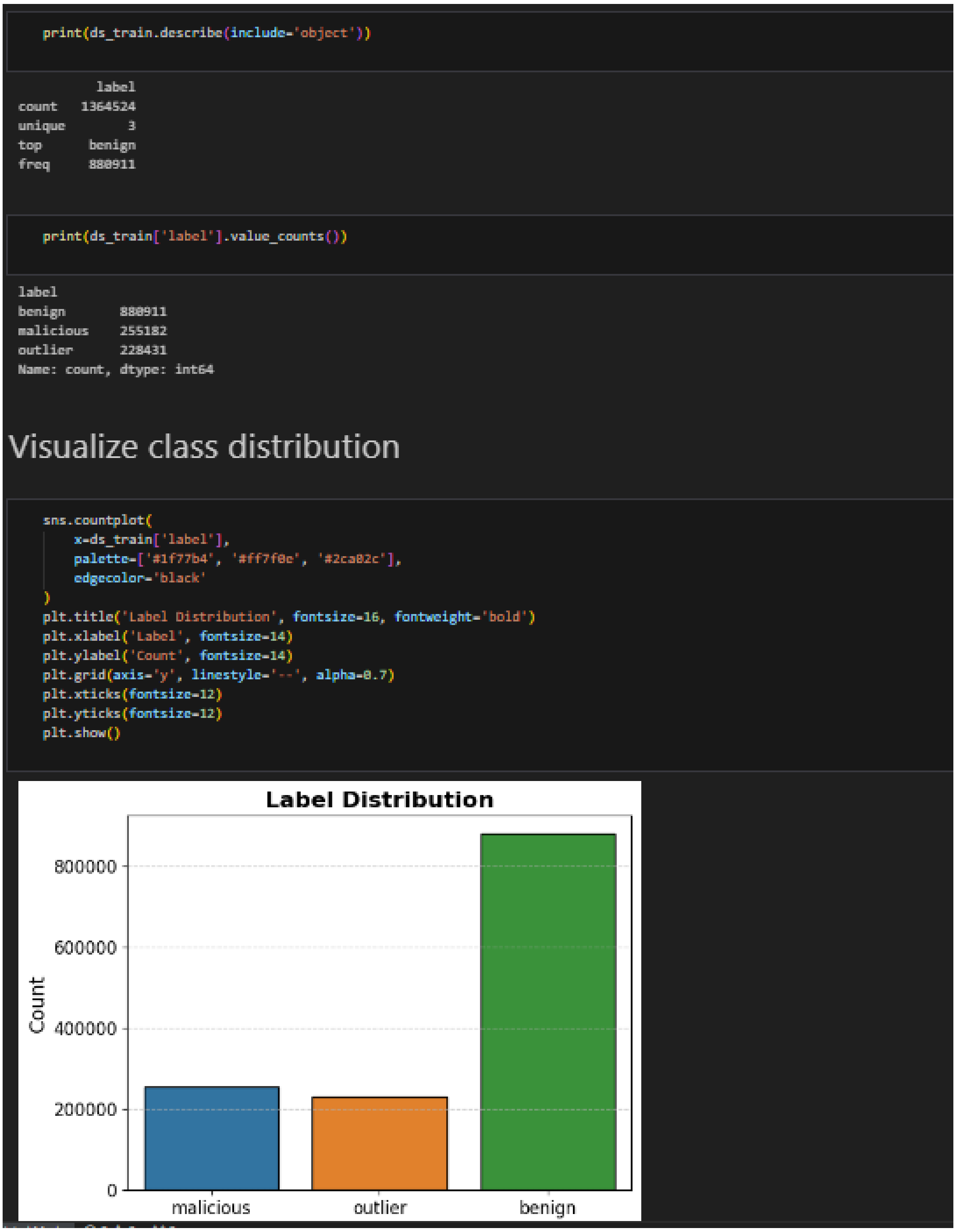


```
ds_train.columns
```

```
Index(['avg_ipt', 'bytes_in', 'bytes_out', 'dest_ip', 'dest_port', 'entropy',
       'num_pkts_out', 'num_pkts_in', 'proto', 'src_ip', 'src_port',
       'time_end', 'time_start', 'total_entropy', 'label', 'duration'],
      dtype='object')
```

```
print(ds_train.describe())
```

```
            avg_ipt      bytes_in     bytes_out    dest_ip     dest_port  \
count  1.364524e+06  1.364524e+06  1.364524e+06  1364524.0  1.354542e+06
mean   1.673218e+06  4.137569e+02  2.738852e+03      786.0  1.597113e+04
std    4.852343e+07  2.076028e+03  5.482318e+03        0.0  1.849242e+04
min    0.000000e+00  0.000000e+00  0.000000e+00      786.0  1.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00      786.0  4.450000e+02
50%    0.000000e+00  0.000000e+00  1.910000e+02      786.0  9.200000e+03
75%    0.000000e+00  0.000000e+00  3.170250e+03      786.0  9.200000e+03
max    4.294967e+09  5.717600e+04  6.553400e+04      786.0  6.553400e+04

            entropy  num_pkts_out   num_pkts_in         proto     src_ip  \
count  1.364524e+06  1.364524e+06  1.364524e+06  1.364524e+06  1364524.0
mean   3.123706e+00  4.742718e+00  2.464735e+00  6.012834e+00      786.0
std    2.392273e+00  1.458849e+01  1.016017e+01  8.526812e-01        0.0
min    0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00      786.0
25%    0.000000e+00  1.000000e+00  0.000000e+00  6.000000e+00      786.0
50%    3.668356e+00  2.000000e+00  0.000000e+00  6.000000e+00      786.0
75%    4.941114e+00  6.000000e+00  2.000000e+00  6.000000e+00      786.0
max    2.243634e+01  2.550000e+02  2.550000e+02  4.700000e+01      786.0

           src_port      time_end    time_start  total_entropy      duration
count  1.354542e+06  1.364524e+06  1.364524e+06   1.364524e+06  1.364524e+06
mean   3.737120e+04  1.466537e+15  1.465629e+15   1.082658e+04  4.799860e-01
std    2.013281e+04  4.394399e+14  4.406363e+14   3.712544e+04  3.045319e+00
min    1.000000e+00  1.612670e+10  1.612670e+10   0.000000e+00  0.000000e+00
25%    9.200000e+03  1.612674e+15  1.612674e+15   0.000000e+00  0.000000e+00
50%    4.024600e+04  1.612706e+15  1.612706e+15   2.106509e+03  8.700000e-05
75%    5.602400e+04  1.612721e+15  1.612721e+15   2.047035e+04  1.986000e-03
max    6.553400e+04  1.612742e+15  1.612742e+15   1.335801e+06  3.727061e+01
```



```
print(ds_train.describe(include='object'))
```

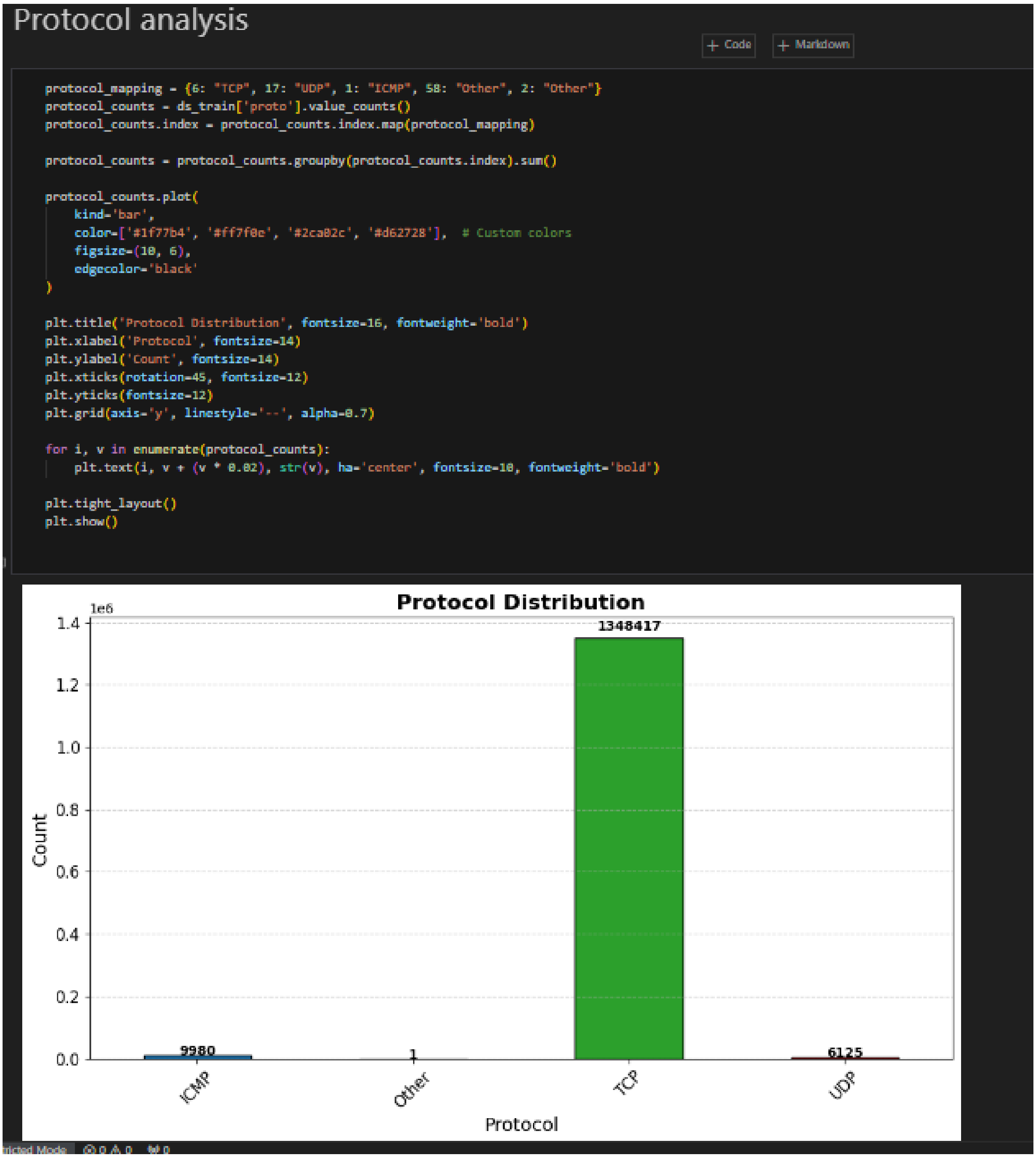
```
          label
count   1364524
unique        3
top      benign
freq     880911
```

```
print(ds_train['label'].value_counts())
```

```
label
benign       880911
malicious    255182
outlier      228431
Name: count, dtype: int64
```

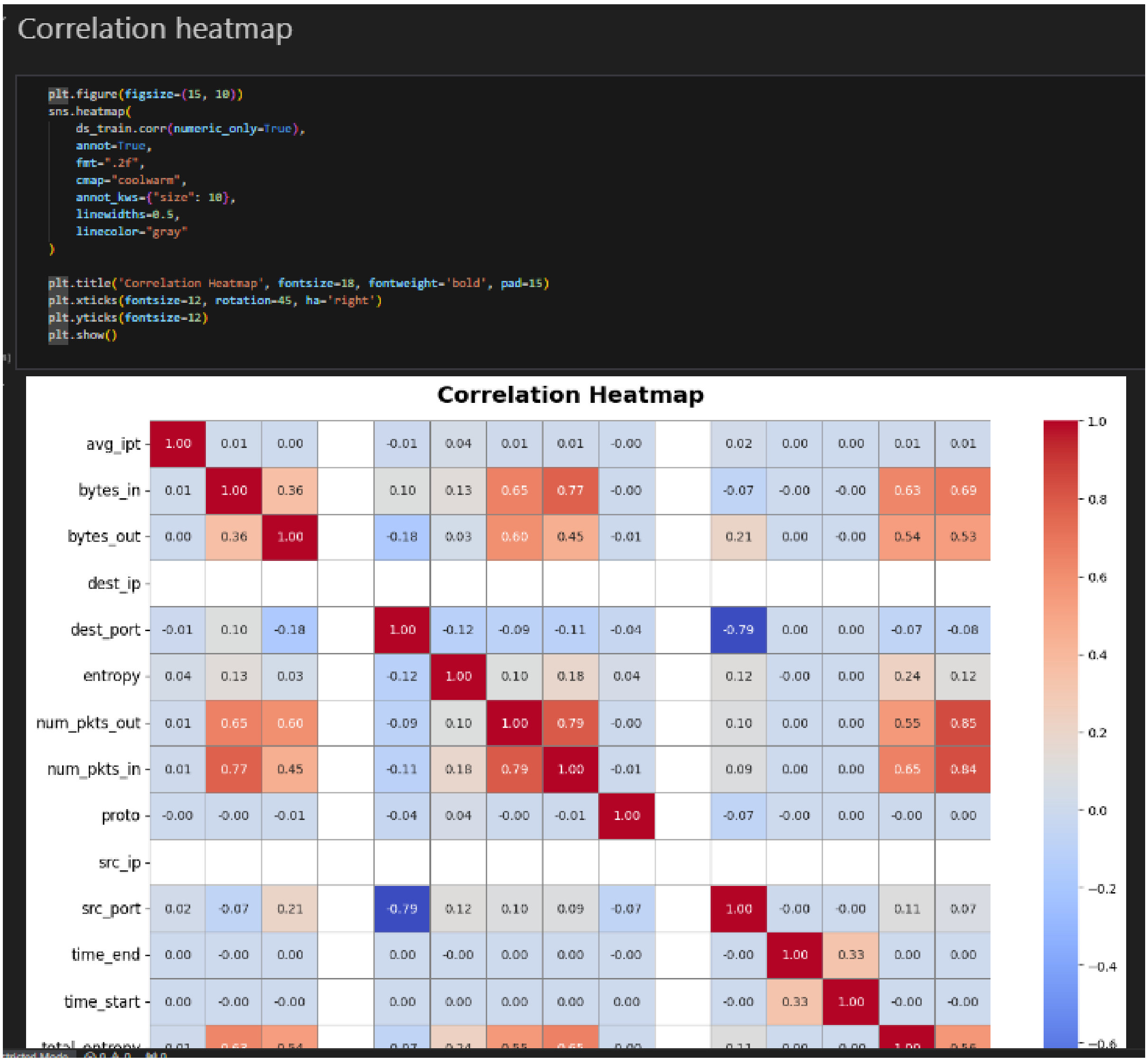
Visualize class distribution

```
sns.countplot(
    x=ds_train['label'],
    palette=['#1f77b4', '#ff7f0e', '#2ca02c'],
    edgecolor='black'
)
plt.title('Label Distribution', fontsize=16, fontweight='bold')
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



Protocol analysis

```
protocol_mapping = {6: "TCP", 17: "UDP", 1: "ICMP", 58: "Other", 2: "Other"}
protocol_counts = ds_train['proto'].value_counts()
protocol_counts.index = protocol_counts.index.map(protocol_mapping)

protocol_counts = protocol_counts.groupby(protocol_counts.index).sum()

protocol_counts.plot(
    kind='bar',
    color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728'],  # Custom colors
    figsize=(10, 6),
    edgecolor='black'
)

plt.title('Protocol Distribution', fontsize=16, fontweight='bold')
plt.xlabel('Protocol', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for i, v in enumerate(protocol_counts):
    plt.text(i, v + (v * 0.02), str(v), ha='center', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()
```

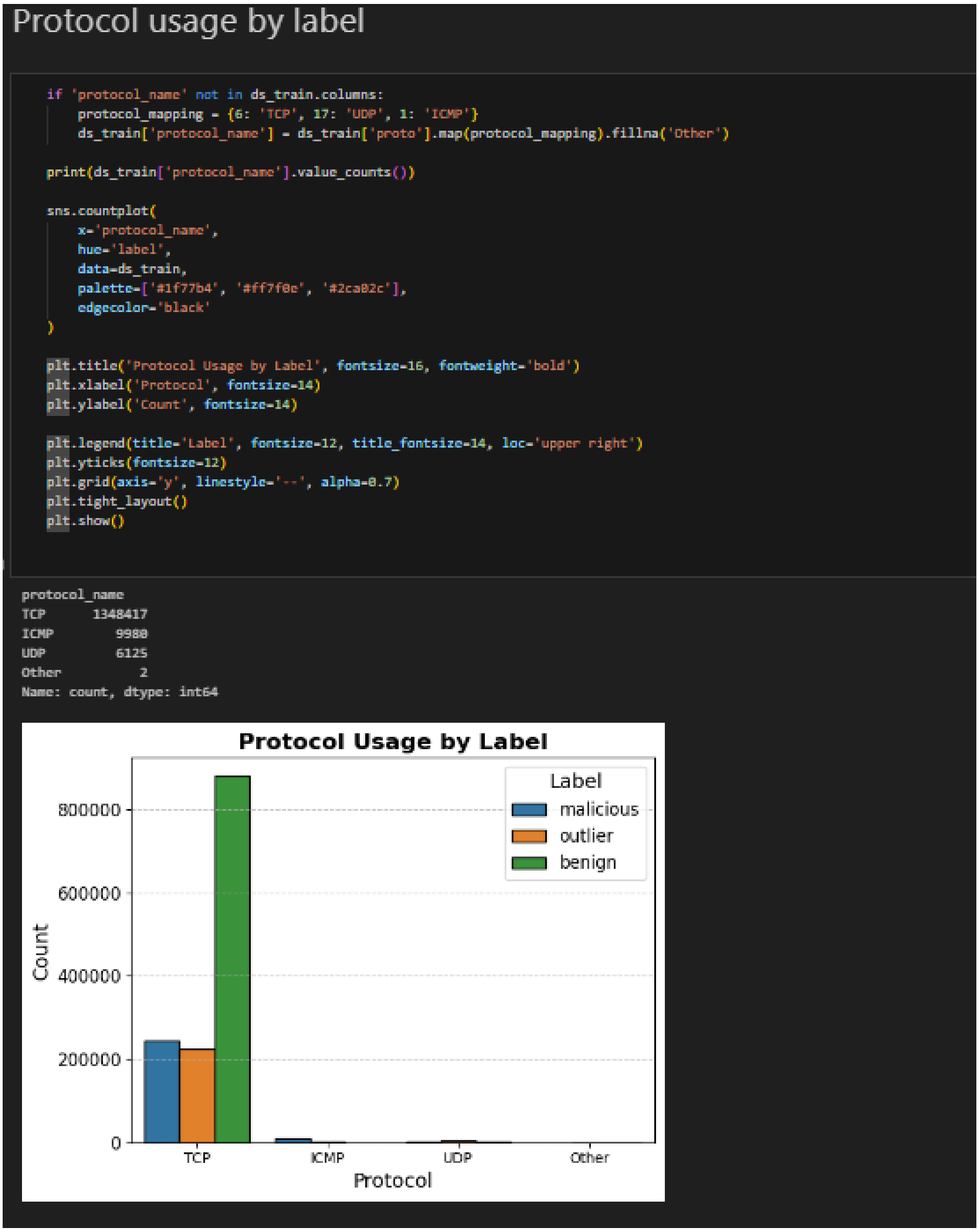


Correlation heatmap

```
plt.figure(figsize=(15, 10))
sns.heatmap(
    ds_train.corr(numeric_only=True),
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    annot_kws={"size": 10},
    linewidths=0.5,
    linecolor="gray"
)

plt.title('Correlation Heatmap', fontsize=18, fontweight='bold', pad=15)
plt.xticks(fontsize=12, rotation=45, ha='right')
plt.yticks(fontsize=12)
plt.show()
```

Correlation Heatmap

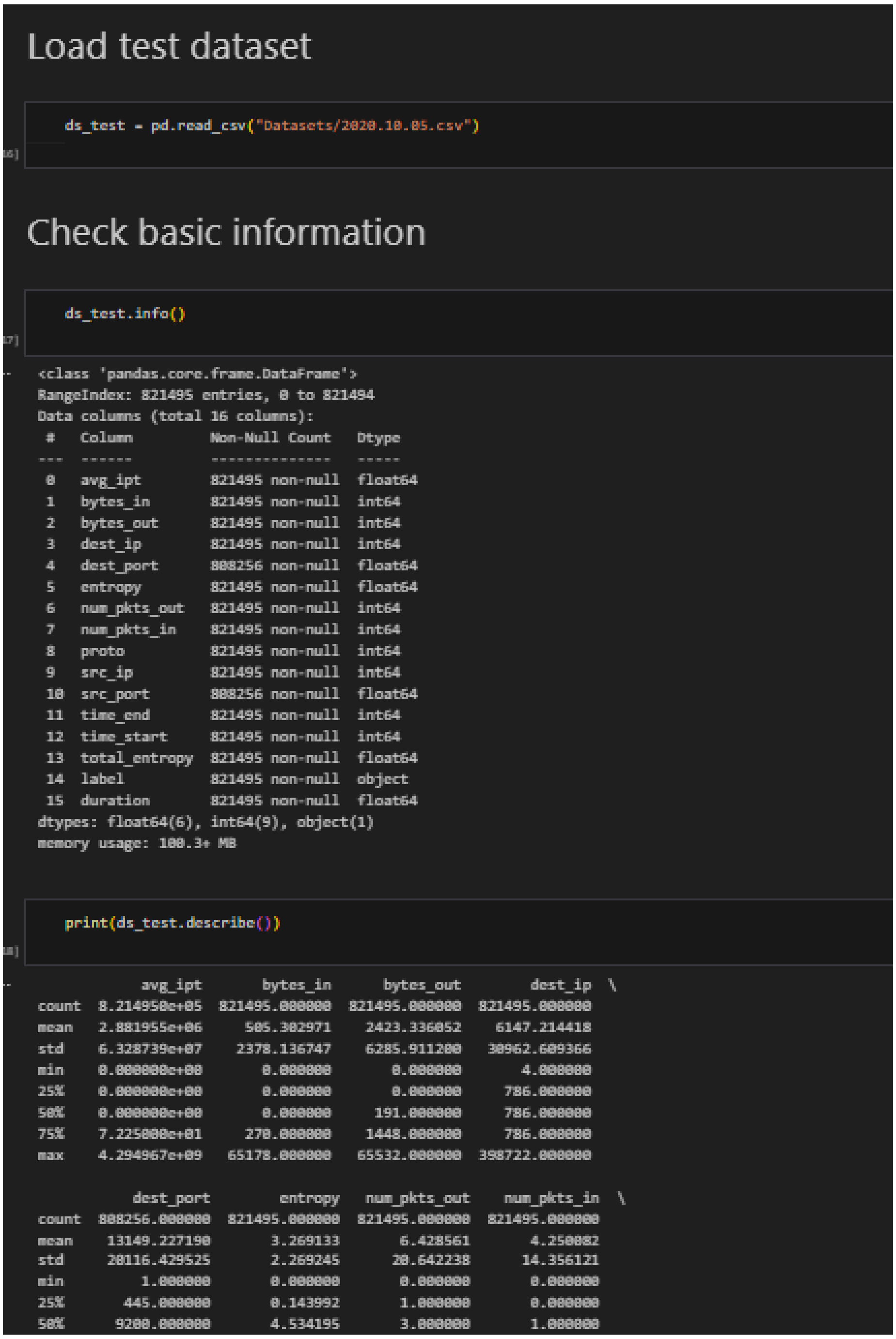
avg_ipt	1.00	0.01	0.00		-0.01	0.04	0.01	0.01	-0.00		0.02	0.00	0.00	0.01	0.01
bytes_in	0.01	1.00	0.36		0.10	0.13	0.65	0.77	-0.00		-0.07	-0.00	-0.00	0.63	0.69
bytes_out	0.00	0.36	1.00		-0.18	0.03	0.60	0.45	-0.01		0.21	0.00	-0.00	0.54	0.53
dest_ip															
dest_port	-0.01	0.10	-0.18		1.00	-0.12	-0.09	-0.11	-0.04		-0.79	0.00	0.00	-0.07	-0.08
entropy	0.04	0.13	0.03		-0.12	1.00	0.10	0.18	0.04		0.12	-0.00	0.00	0.24	0.12
num_pkts_out	0.01	0.65	0.60		-0.09	0.10	1.00	0.79	-0.00		0.10	0.00	0.00	0.55	0.85
num_pkts_in	0.01	0.77	0.45		-0.11	0.18	0.79	1.00	-0.01		0.09	0.00	0.00	0.65	0.84
proto	-0.00	-0.00	-0.01		-0.04	0.04	-0.00	-0.01	1.00		-0.07	-0.00	0.00	-0.00	0.00
src_ip															
src_port	0.02	-0.07	0.21		-0.79	0.12	0.10	0.09	-0.07		1.00	-0.00	-0.00	0.11	0.07
time_end	0.00	-0.00	0.00		0.00	-0.00	0.00	0.00	-0.00		-0.00	1.00	0.33	0.00	0.00
time_start	0.00	-0.00	-0.00		0.00	0.00	0.00	0.00	0.00		-0.00	0.33	1.00	-0.00	-0.00
total_entropy	0.01	0.63	0.54		-0.07	0.24	0.55	0.65	0.00		0.11	0.00	0.00	1.00	0.56



Protocol usage by label

```
if 'protocol_name' not in ds_train.columns:
    protocol_mapping = {6: 'TCP', 17: 'UDP', 1: 'ICMP'}
    ds_train['protocol_name'] = ds_train['proto'].map(protocol_mapping).fillna('Other')

print(ds_train['protocol_name'].value_counts())

sns.countplot(
    x='protocol_name',
    hue='label',
    data=ds_train,
    palette=['#1f77b4', '#ff7f0e', '#2ca02c'],
    edgecolor='black'
)

plt.title('Protocol Usage by Label', fontsize=16, fontweight='bold')
plt.xlabel('Protocol', fontsize=14)
plt.ylabel('Count', fontsize=14)

plt.legend(title='Label', fontsize=12, title_fontsize=14, loc='upper right')
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
protocol_name
TCP      1348417
ICMP        9980
UDP         6125
Other          2
Name: count, dtype: int64
```



Load test dataset

```
ds_test = pd.read_csv("Datasets/2020.10.05.csv")
```

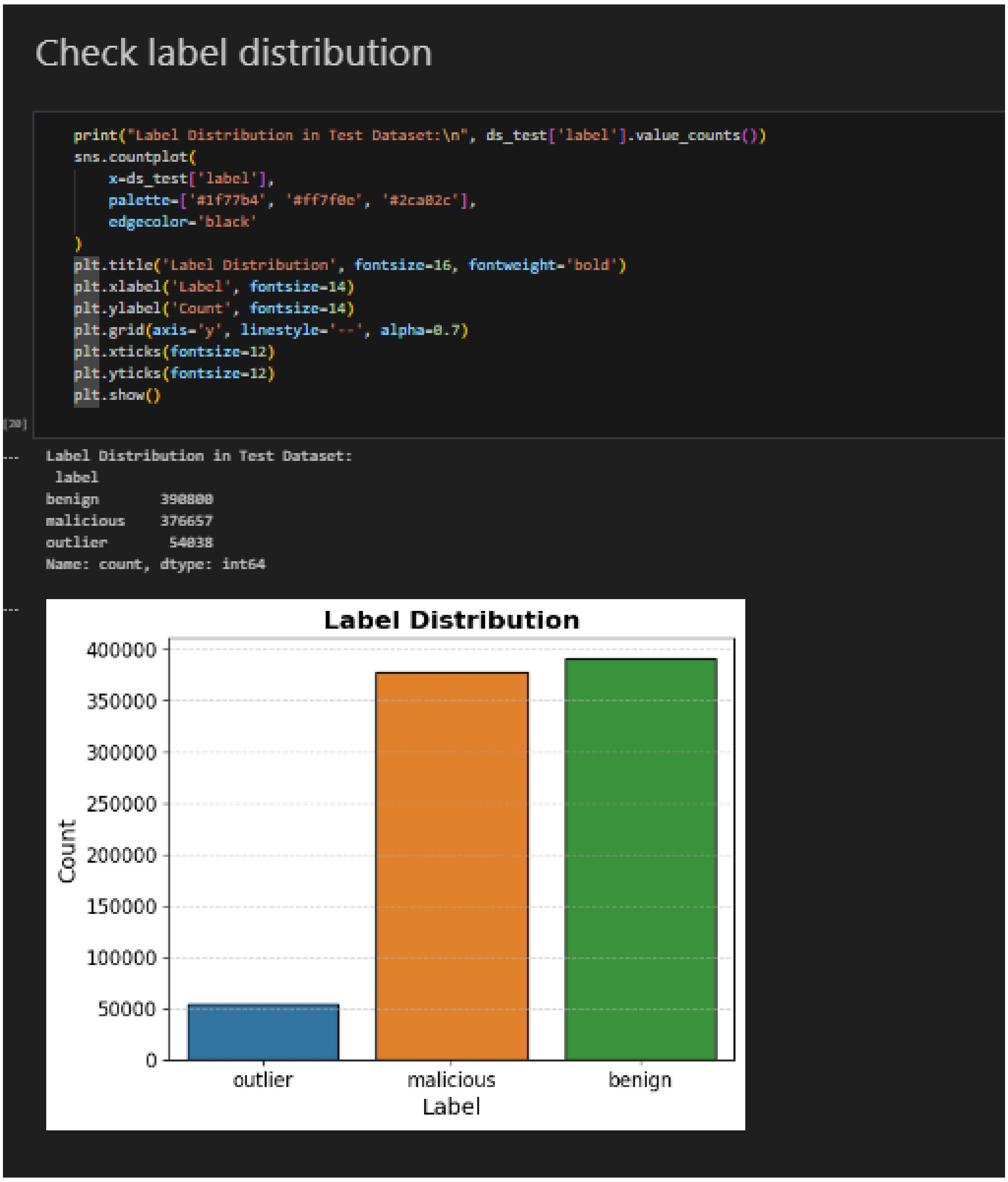
Check basic information

```
ds_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 821495 entries, 0 to 821494
Data columns (total 16 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   avg_ipt        821495 non-null  float64
 1   bytes_in       821495 non-null  int64
 2   bytes_out      821495 non-null  int64
 3   dest_ip        821495 non-null  int64
 4   dest_port      808256 non-null  float64
 5   entropy        821495 non-null  float64
 6   num_pkts_out   821495 non-null  int64
 7   num_pkts_in    821495 non-null  int64
 8   proto          821495 non-null  int64
 9   src_ip         821495 non-null  int64
 10  src_port       808256 non-null  float64
 11  time_end       821495 non-null  int64
 12  time_start     821495 non-null  int64
 13  total_entropy  821495 non-null  float64
 14  label          821495 non-null  object
 15  duration       821495 non-null  float64
dtypes: float64(6), int64(9), object(1)
memory usage: 100.3+ MB
```

```
print(ds_test.describe())
```

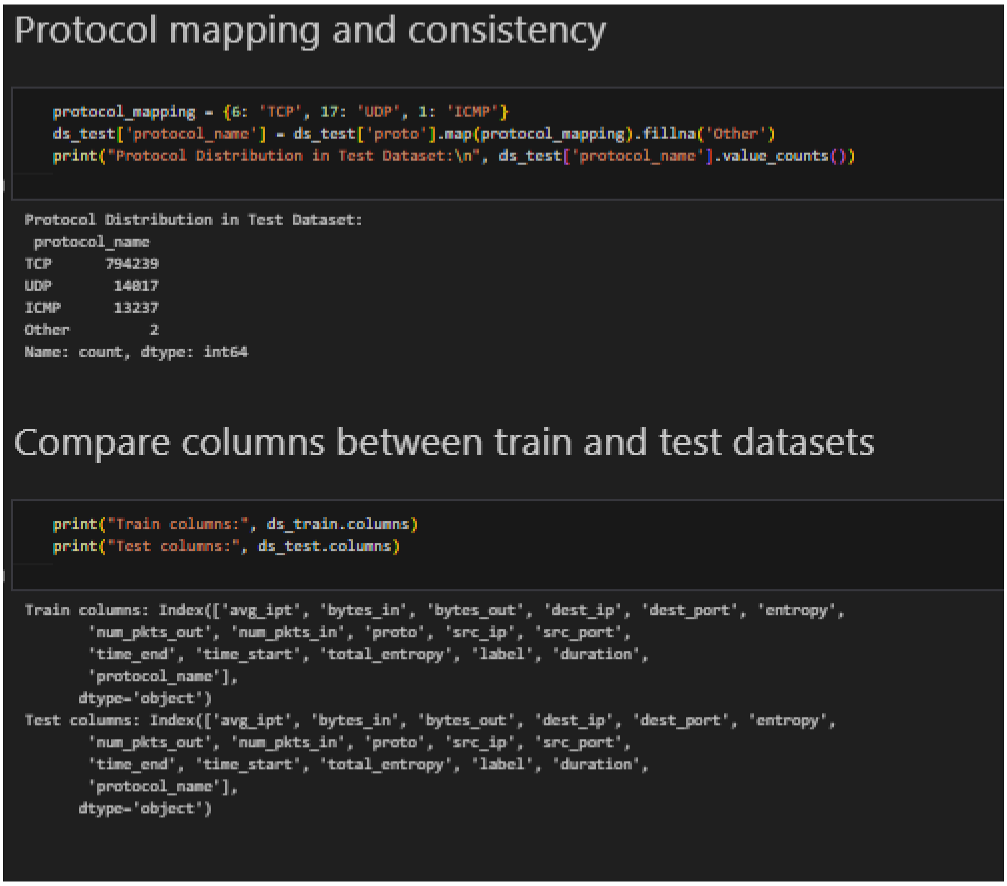
```
            avg_ipt       bytes_in      bytes_out        dest_ip  \
count  8.214950e+05  821495.000000  821495.000000  821495.000000
mean   2.881955e+06     505.302971    2423.336052    6147.214418
std    6.328739e+07    2378.136747    6285.911200   30962.609366
min    0.000000e+00       0.000000       0.000000       4.000000
25%    0.000000e+00       0.000000       0.000000     786.000000
50%    0.000000e+00       0.000000     191.000000     786.000000
75%    7.225000e+01     270.000000    1448.000000     786.000000
max    4.294967e+09   65178.000000   65532.000000  398722.000000

           dest_port        entropy   num_pkts_out    num_pkts_in  \
count  808256.000000  821495.000000  821495.000000  821495.000000
mean    13149.227190       3.269133       6.428561       4.250082
std     20116.429525       2.269245      20.642238      14.356121
min         1.000000       0.000000       0.000000       0.000000
25%       445.000000       0.143992       1.000000       0.000000
50%      9200.000000       4.534195       3.000000       1.000000
```



Check label distribution

```python
print("Label Distribution in Test Dataset:\n", ds_test['label'].value_counts())
sns.countplot(
    x=ds_test['label'],
    palette=['#1f77b4', '#ff7f0e', '#2ca02c'],
    edgecolor='black'
)
plt.title('Label Distribution', fontsize=16, fontweight='bold')
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

```
Label Distribution in Test Dataset:
 label
benign       390800
malicious    376657
outlier       54038
Name: count, dtype: int64
```



Protocol mapping and consistency

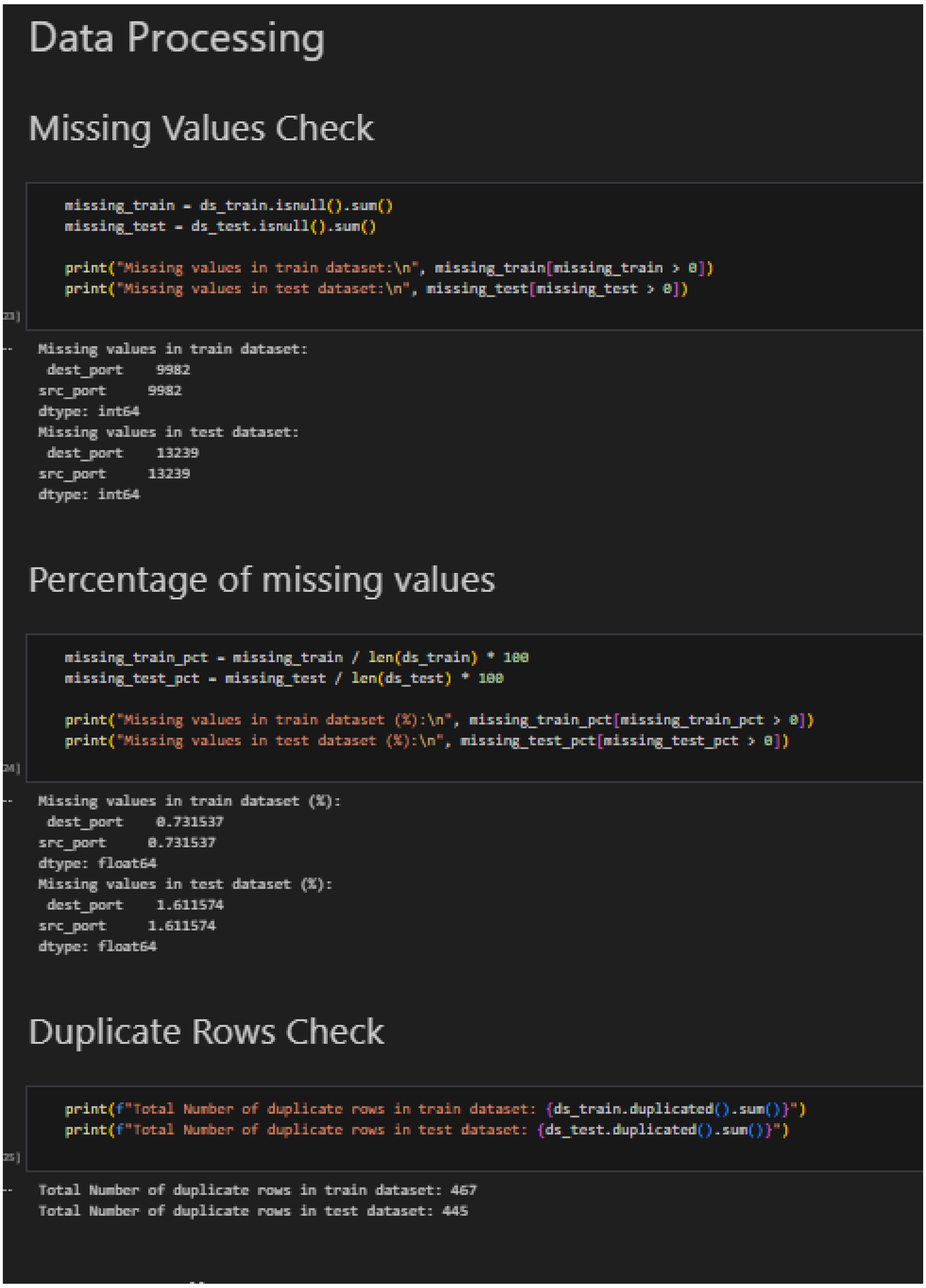
```
protocol_mapping = {6: 'TCP', 17: 'UDP', 1: 'ICMP'}
ds_test['protocol_name'] = ds_test['proto'].map(protocol_mapping).fillna('Other')
print("Protocol Distribution in Test Dataset:\n", ds_test['protocol_name'].value_counts())
```

```
Protocol Distribution in Test Dataset:
 protocol_name
TCP      794239
UDP       14817
ICMP      13237
Other         2
Name: count, dtype: int64
```

Compare columns between train and test datasets

```
print("Train columns:", ds_train.columns)
print("Test columns:", ds_test.columns)
```

```
Train columns: Index(['avg_ipt', 'bytes_in', 'bytes_out', 'dest_ip', 'dest_port', 'entropy',
       'num_pkts_out', 'num_pkts_in', 'proto', 'src_ip', 'src_port',
       'time_end', 'time_start', 'total_entropy', 'label', 'duration',
       'protocol_name'],
      dtype='object')
Test columns: Index(['avg_ipt', 'bytes_in', 'bytes_out', 'dest_ip', 'dest_port', 'entropy',
       'num_pkts_out', 'num_pkts_in', 'proto', 'src_ip', 'src_port',
       'time_end', 'time_start', 'total_entropy', 'label', 'duration',
       'protocol_name'],
      dtype='object')
```



Data Processing

Missing Values Check

```
missing_train = ds_train.isnull().sum()
missing_test = ds_test.isnull().sum()

print("Missing values in train dataset:\n", missing_train[missing_train > 0])
print("Missing values in test dataset:\n", missing_test[missing_test > 0])
```

```
Missing values in train dataset:
 dest_port    9982
src_port     9982
dtype: int64
Missing values in test dataset:
 dest_port    13239
src_port     13239
dtype: int64
```

Percentage of missing values

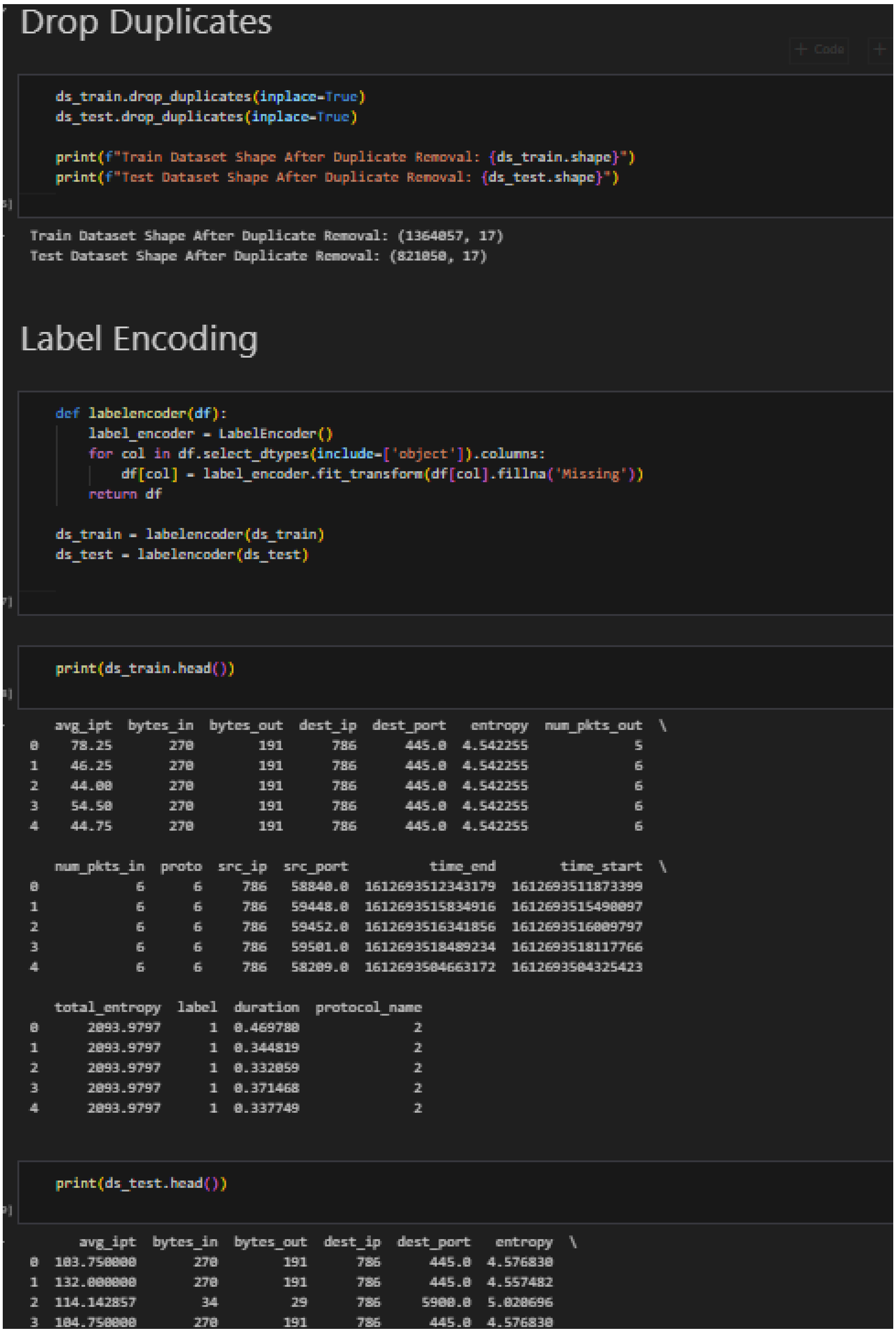
```
missing_train_pct = missing_train / len(ds_train) * 100
missing_test_pct = missing_test / len(ds_test) * 100

print("Missing values in train dataset (%):\n", missing_train_pct[missing_train_pct > 0])
print("Missing values in test dataset (%):\n", missing_test_pct[missing_test_pct > 0])
```

```
Missing values in train dataset (%):
 dest_port    0.731537
src_port     0.731537
dtype: float64
Missing values in test dataset (%):
 dest_port    1.611574
src_port     1.611574
dtype: float64
```

Duplicate Rows Check

```
print(f"Total Number of duplicate rows in train dataset: {ds_train.duplicated().sum()}")
print(f"Total Number of duplicate rows in test dataset: {ds_test.duplicated().sum()}")
```

```
Total Number of duplicate rows in train dataset: 467
Total Number of duplicate rows in test dataset: 445
```



Drop Duplicates

```
ds_train.drop_duplicates(inplace=True)
ds_test.drop_duplicates(inplace=True)

print(f"Train Dataset Shape After Duplicate Removal: {ds_train.shape}")
print(f"Test Dataset Shape After Duplicate Removal: {ds_test.shape}")
```

```
Train Dataset Shape After Duplicate Removal: (1364057, 17)
Test Dataset Shape After Duplicate Removal: (821050, 17)
```

Label Encoding

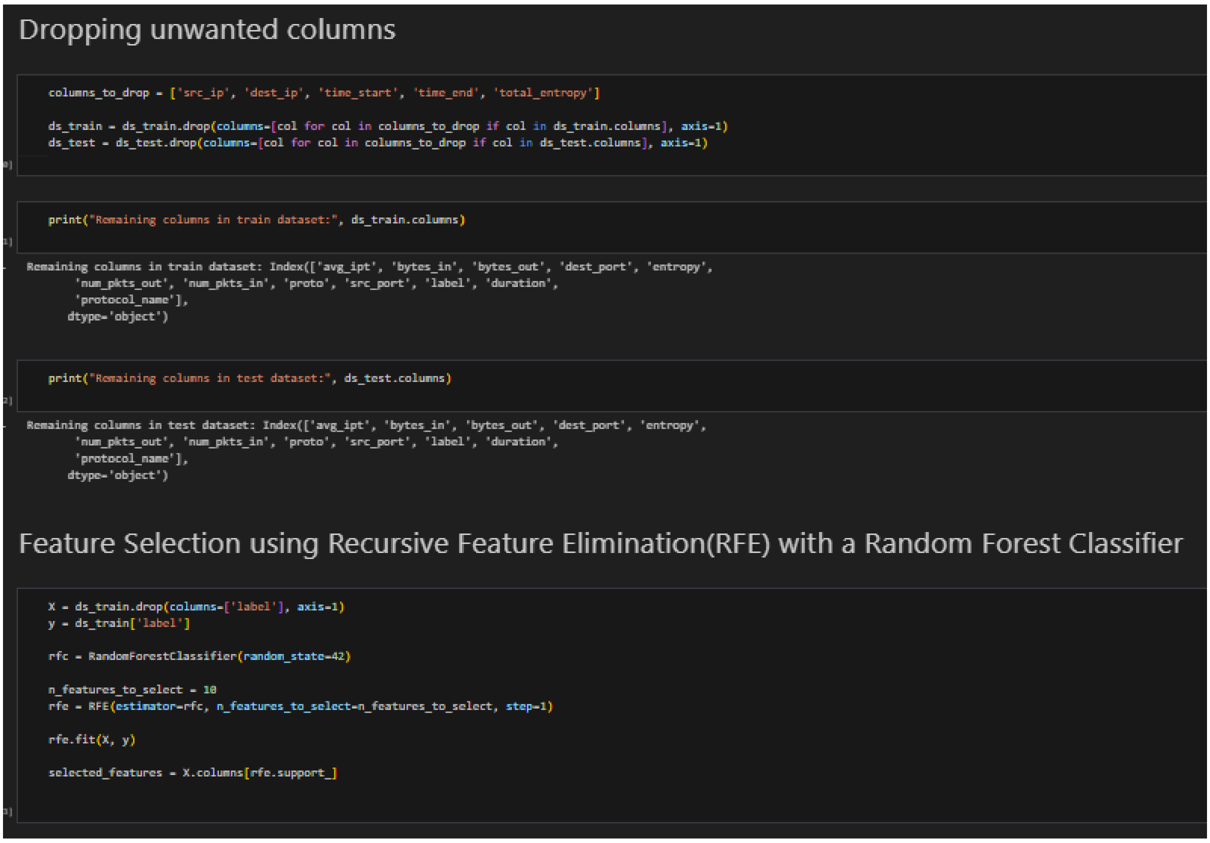
```
def labelencoder(df):
    label_encoder = LabelEncoder()
    for col in df.select_dtypes(include=['object']).columns:
        df[col] = label_encoder.fit_transform(df[col].fillna('Missing'))
    return df

ds_train = labelencoder(ds_train)
ds_test = labelencoder(ds_test)
```

```
print(ds_train.head())
```

```
   avg_ipt  bytes_in  bytes_out  dest_ip  dest_port   entropy  num_pkts_out  \
0    78.25       270        191      786      445.0  4.542255             5
1    46.25       270        191      786      445.0  4.542255             6
2    44.00       270        191      786      445.0  4.542255             6
3    54.50       270        191      786      445.0  4.542255             6
4    44.75       270        191      786      445.0  4.542255             6

   num_pkts_in  proto  src_ip  src_port           time_end         time_start  \
0            6      6     786   58840.0  1612693512343179  1612693511873399
1            6      6     786   59448.0  1612693515834916  1612693515490097
2            6      6     786   59452.0  1612693516341856  1612693516009797
3            6      6     786   59501.0  1612693518489234  1612693518117766
4            6      6     786   58209.0  1612693504663172  1612693504325423

   total_entropy  label  duration  protocol_name
0      2093.9797      1  0.469780              2
1      2093.9797      1  0.344819              2
2      2093.9797      1  0.332059              2
3      2093.9797      1  0.371468              2
4      2093.9797      1  0.337749              2
```

```
print(ds_test.head())
```

```
      avg_ipt  bytes_in  bytes_out  dest_ip  dest_port   entropy  \
0  103.750000       270        191      786      445.0  4.576830
1  132.000000       270        191      786      445.0  4.557482
2  114.142857        34         29      786     5900.0  5.020696
3  104.750000       270        191      786      445.0  4.576830
```



Dropping unwanted columns

```
columns_to_drop = ['src_ip', 'dest_ip', 'time_start', 'time_end', 'total_entropy']

ds_train = ds_train.drop(columns=[col for col in columns_to_drop if col in ds_train.columns], axis=1)
ds_test = ds_test.drop(columns=[col for col in columns_to_drop if col in ds_test.columns], axis=1)
```

```
print("Remaining columns in train dataset:", ds_train.columns)
```

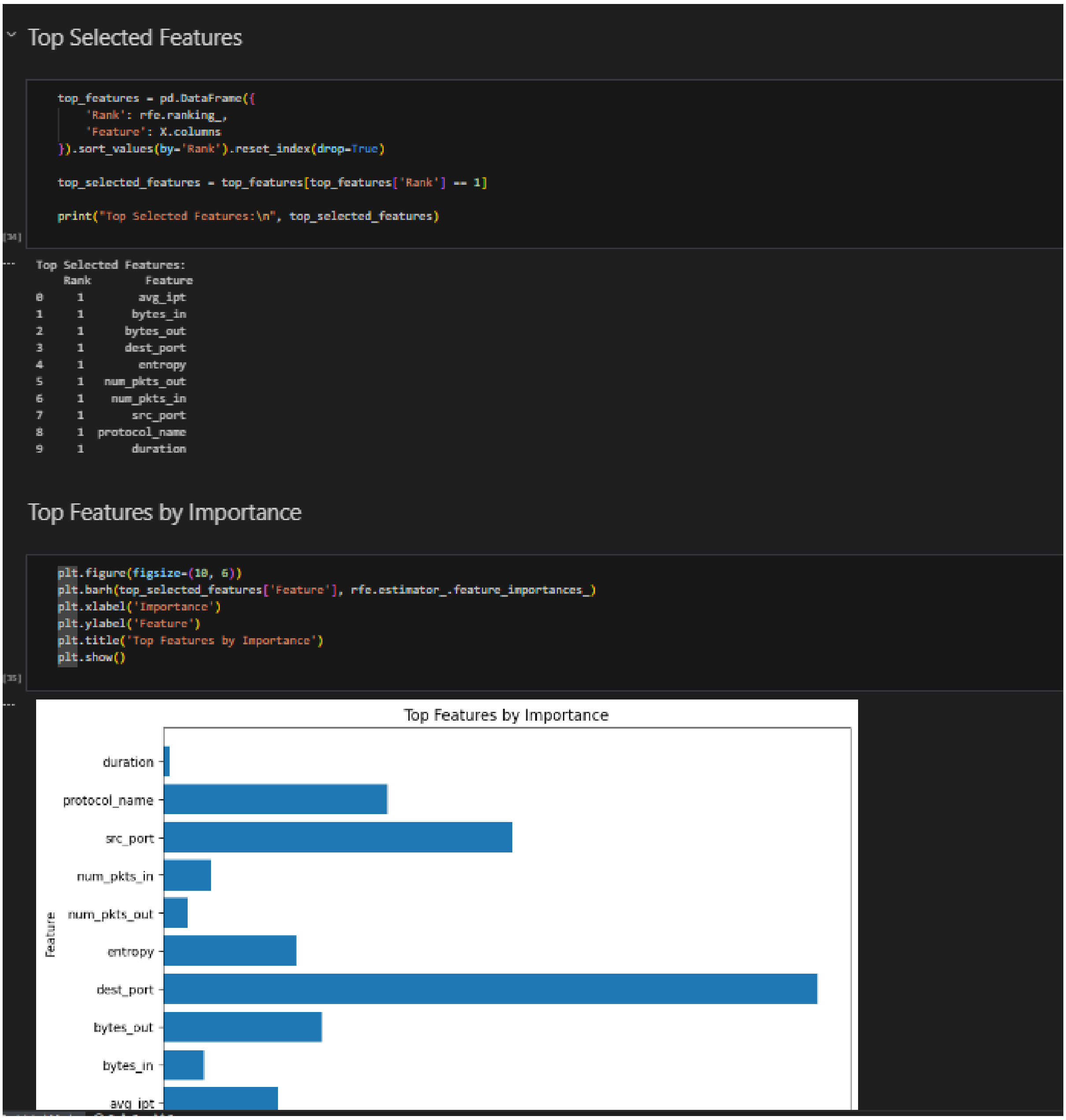
```
Remaining columns in train dataset: Index(['avg_ipt', 'bytes_in', 'bytes_out', 'dest_port', 'entropy',
       'num_pkts_out', 'num_pkts_in', 'proto', 'src_port', 'label', 'duration',
       'protocol_name'],
      dtype='object')
```

```
print("Remaining columns in test dataset:", ds_test.columns)
```

```
Remaining columns in test dataset: Index(['avg_ipt', 'bytes_in', 'bytes_out', 'dest_port', 'entropy',
       'num_pkts_out', 'num_pkts_in', 'proto', 'src_port', 'label', 'duration',
       'protocol_name'],
      dtype='object')
```

Feature Selection using Recursive Feature Elimination(RFE) with a Random Forest Classifier

```
X = ds_train.drop(columns=['label'], axis=1)
y = ds_train['label']

rfc = RandomForestClassifier(random_state=42)

n_features_to_select = 10
rfe = RFE(estimator=rfc, n_features_to_select=n_features_to_select, step=1)

rfe.fit(X, y)

selected_features = X.columns[rfe.support_]
```



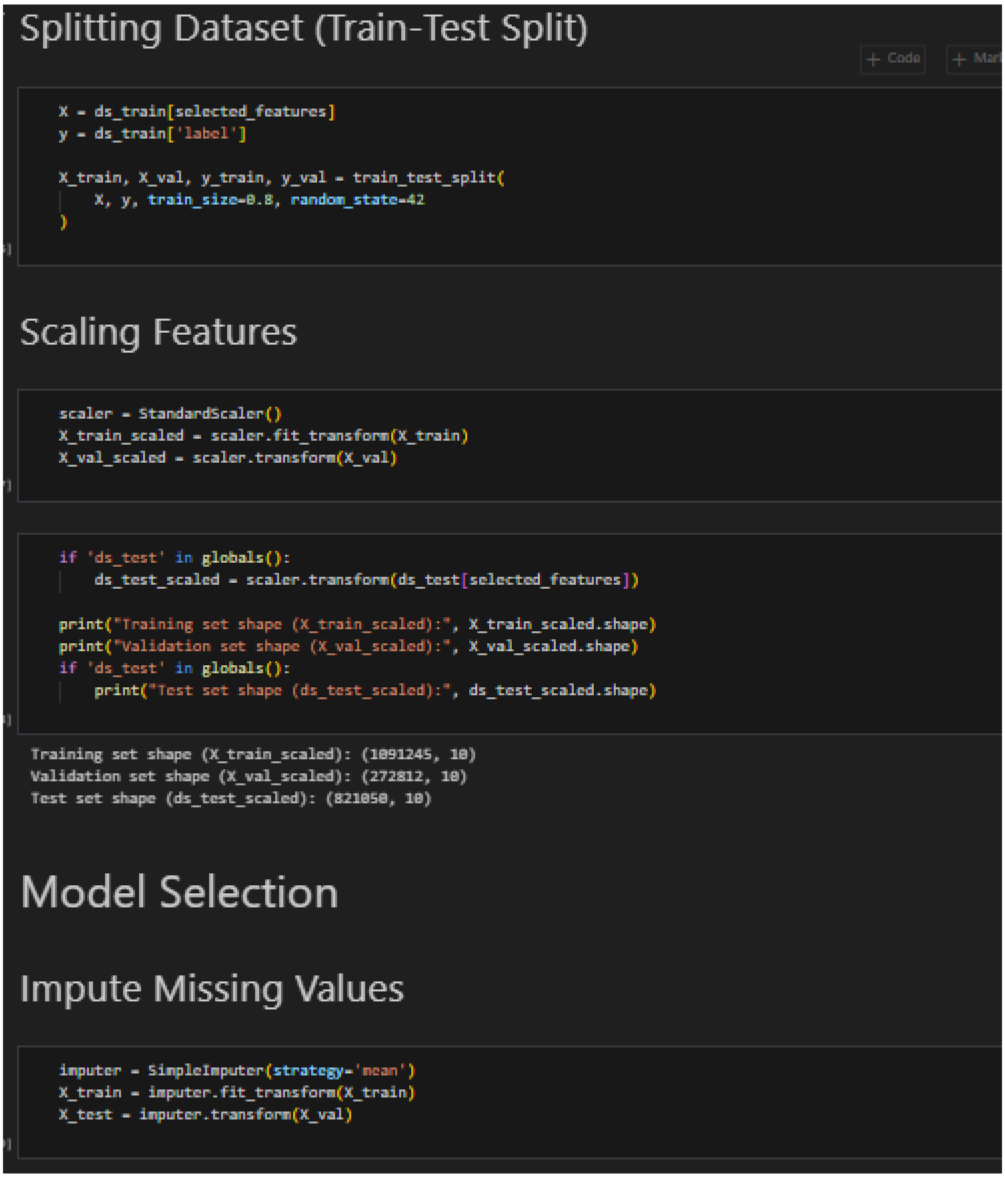
Top Selected Features

```
top_features = pd.DataFrame({
    'Rank': rfe.ranking_,
    'Feature': X.columns
}).sort_values(by='Rank').reset_index(drop=True)

top_selected_features = top_features[top_features['Rank'] == 1]

print("Top Selected Features:\n", top_selected_features)
```

```
Top Selected Features:
    Rank         Feature
0     1         avg_ipt
1     1        bytes_in
2     1       bytes_out
3     1       dest_port
4     1         entropy
5     1    num_pkts_out
6     1     num_pkts_in
7     1        src_port
8     1   protocol_name
9     1        duration
```

Top Features by Importance

```
plt.figure(figsize=(10, 6))
plt.barh(top_selected_features['Feature'], rfe.estimator_.feature_importances_)
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Top Features by Importance')
plt.show()
```



Splitting Dataset (Train-Test Split)

```
X = ds_train[selected_features]
y = ds_train['label']

X_train, X_val, y_train, y_val = train_test_split(
    X, y, train_size=0.8, random_state=42
)
```

Scaling Features

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

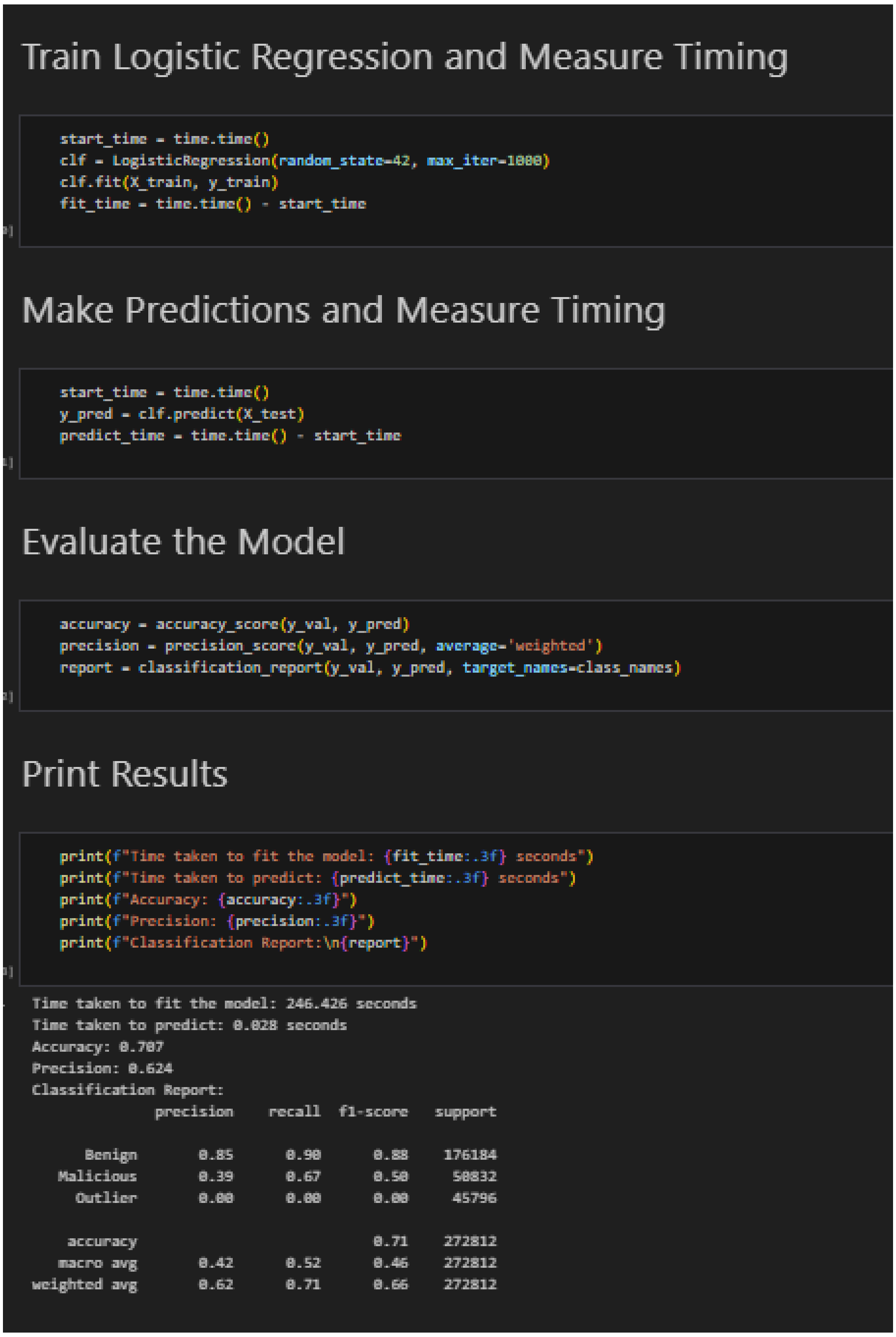
```
if 'ds_test' in globals():
    ds_test_scaled = scaler.transform(ds_test[selected_features])

print("Training set shape (X_train_scaled):", X_train_scaled.shape)
print("Validation set shape (X_val_scaled):", X_val_scaled.shape)
if 'ds_test' in globals():
    print("Test set shape (ds_test_scaled):", ds_test_scaled.shape)
```

```
Training set shape (X_train_scaled): (1091245, 10)
Validation set shape (X_val_scaled): (272812, 10)
Test set shape (ds_test_scaled): (821050, 10)
```

Model Selection

Impute Missing Values

```
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_val)
```



Train Logistic Regression and Measure Timing

```
start_time = time.time()
clf = LogisticRegression(random_state=42, max_iter=1000)
clf.fit(X_train, y_train)
fit_time = time.time() - start_time
```

Make Predictions and Measure Timing

```
start_time = time.time()
y_pred = clf.predict(X_test)
predict_time = time.time() - start_time
```

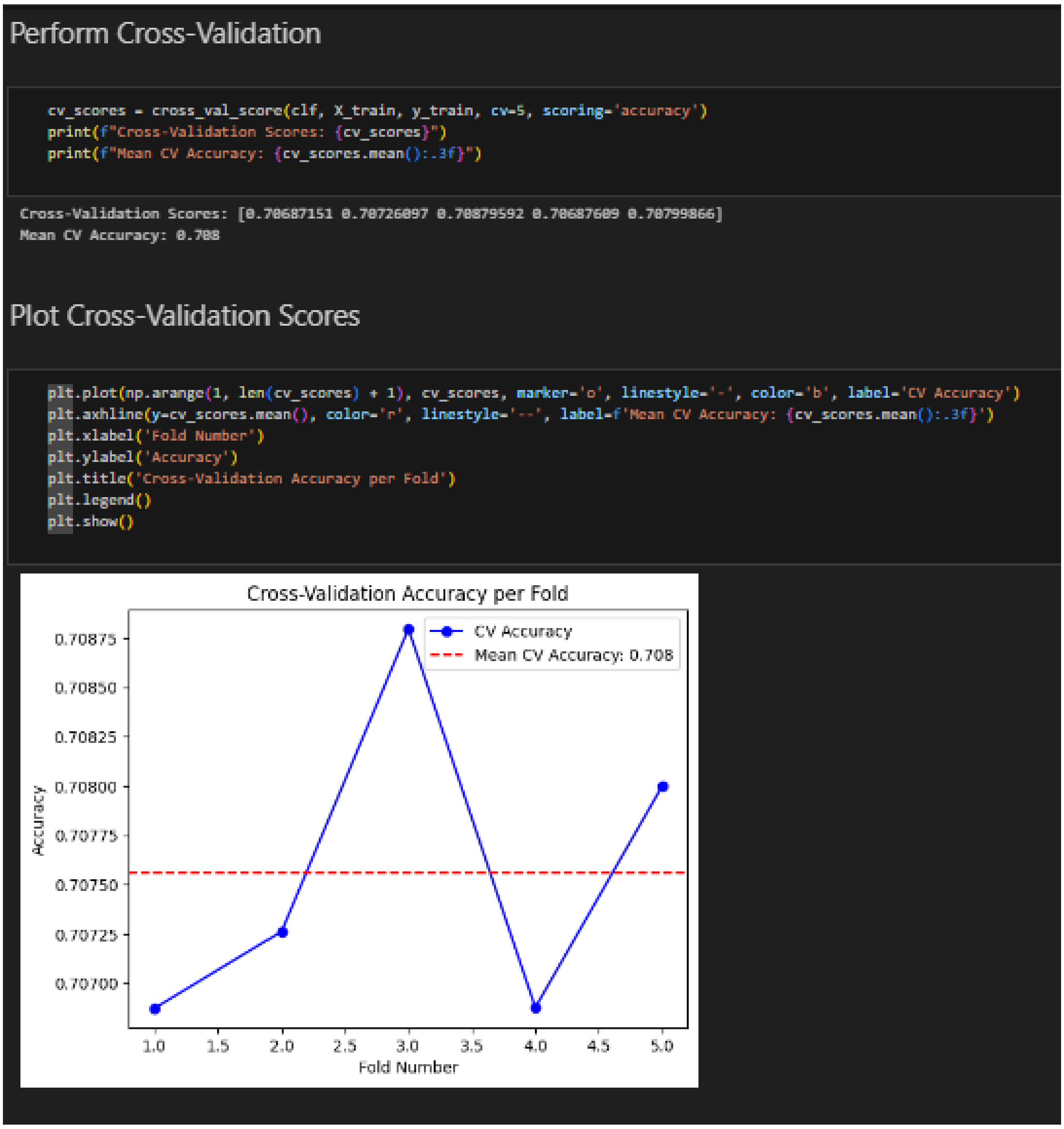
Evaluate the Model

```
accuracy = accuracy_score(y_val, y_pred)
precision = precision_score(y_val, y_pred, average='weighted')
report = classification_report(y_val, y_pred, target_names=class_names)
```

Print Results

```
print(f"Time taken to fit the model: {fit_time:.3f} seconds")
print(f"Time taken to predict: {predict_time:.3f} seconds")
print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Classification Report:\n{report}")
```

```
Time taken to fit the model: 246.426 seconds
Time taken to predict: 0.028 seconds
Accuracy: 0.707
Precision: 0.624
Classification Report:
              precision    recall  f1-score   support

      Benign       0.85      0.90      0.88    176184
   Malicious       0.39      0.67      0.50     50832
     Outlier       0.00      0.00      0.00     45796

    accuracy                           0.71    272812
   macro avg       0.42      0.52      0.46    272812
weighted avg       0.62      0.71      0.66    272812
```



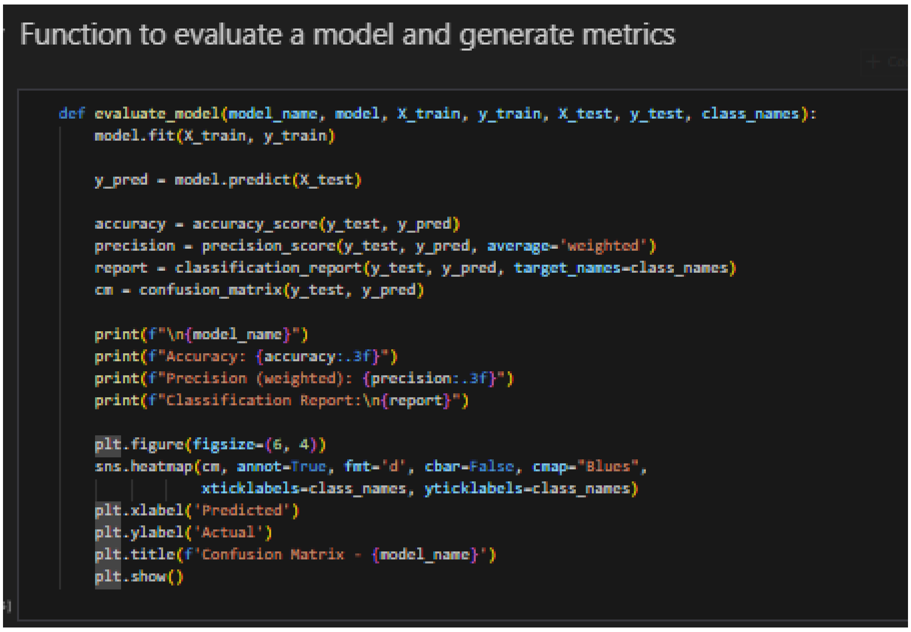
Perform Cross-Validation

```
cv_scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')
print(f"Cross-Validation Scores: {cv_scores}")
print(f"Mean CV Accuracy: {cv_scores.mean():.3f}")
```

```
Cross-Validation Scores: [0.70687151 0.70726097 0.70879592 0.70687609 0.70799866]
Mean CV Accuracy: 0.708
```

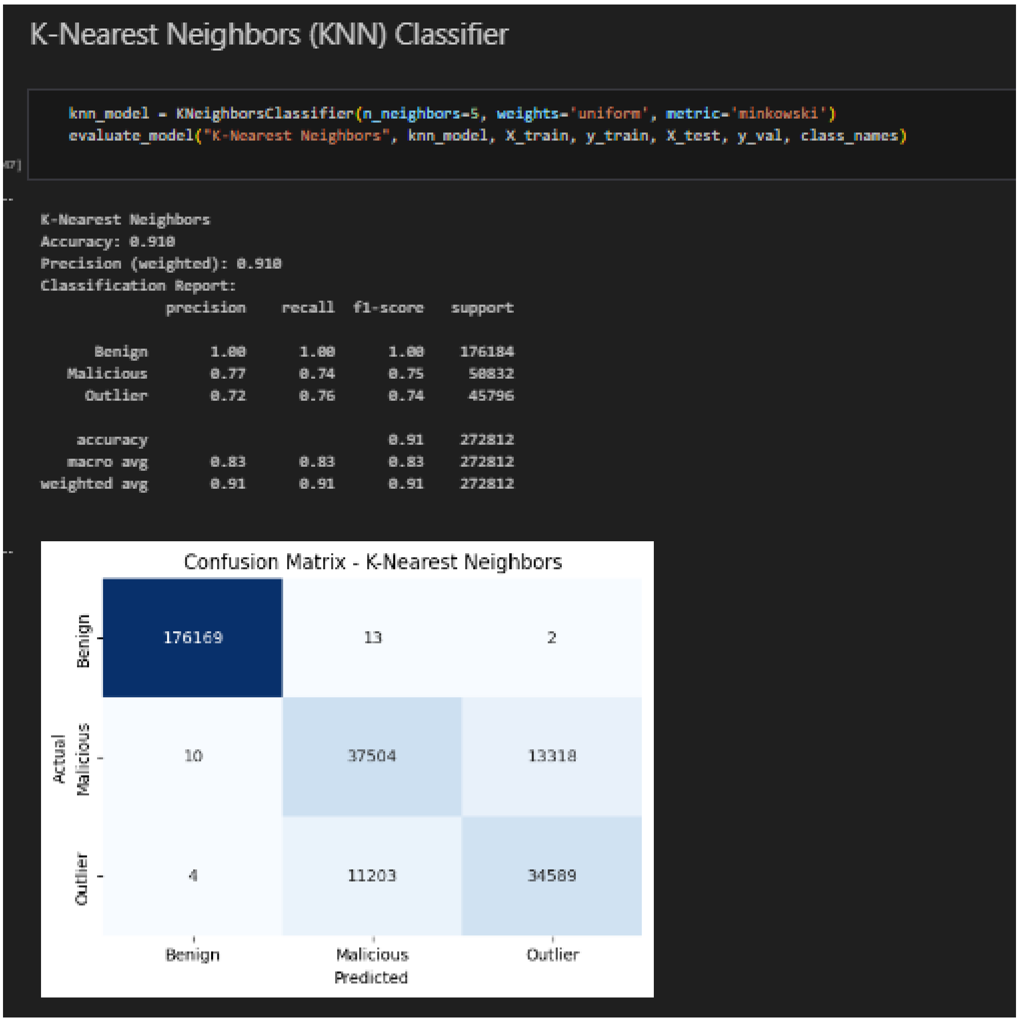
Plot Cross-Validation Scores

```
plt.plot(np.arange(1, len(cv_scores) + 1), cv_scores, marker='o', linestyle='-', color='b', label='CV Accuracy')
plt.axhline(y=cv_scores.mean(), color='r', linestyle='--', label=f'Mean CV Accuracy: {cv_scores.mean():.3f}')
plt.xlabel('Fold Number')
plt.ylabel('Accuracy')
plt.title('Cross-Validation Accuracy per Fold')
plt.legend()
plt.show()
```



Function to evaluate a model and generate metrics

```
def evaluate_model(model_name, model, X_train, y_train, X_test, y_test, class_names):
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    report = classification_report(y_test, y_pred, target_names=class_names)
    cm = confusion_matrix(y_test, y_pred)

    print(f"\n{model_name}")
    print(f"Accuracy: {accuracy:.3f}")
    print(f"Precision (weighted): {precision:.3f}")
    print(f"Classification Report:\n{report}")

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cbar=False, cmap="Blues",
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()
```



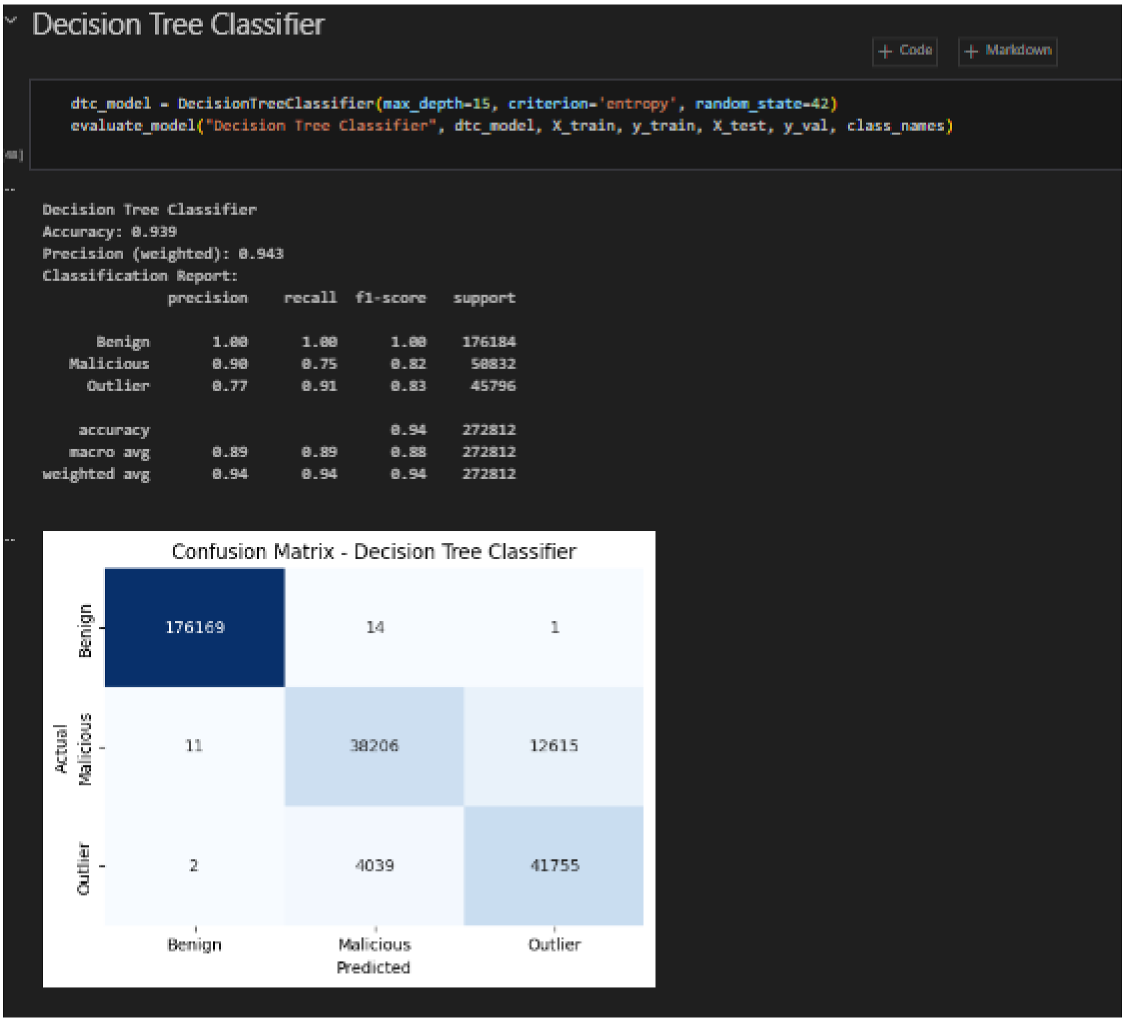
K-Nearest Neighbors (KNN) Classifier

```
knn_model = KNeighborsClassifier(n_neighbors=5, weights='uniform', metric='minkowski')
evaluate_model("K-Nearest Neighbors", knn_model, X_train, y_train, X_test, y_val, class_names)
```

```
K-Nearest Neighbors
Accuracy: 0.910
Precision (weighted): 0.910
Classification Report:
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00    176184
   Malicious       0.77      0.74      0.75     50832
     Outlier       0.72      0.76      0.74     45796

    accuracy                           0.91    272812
   macro avg       0.83      0.83      0.83    272812
weighted avg       0.91      0.91      0.91    272812
```

Confusion Matrix - K-Nearest Neighbors

	Benign	Malicious	Outlier
Benign	176169	13	2
Malicious	10	37504	13318
Outlier	4	11203	34589



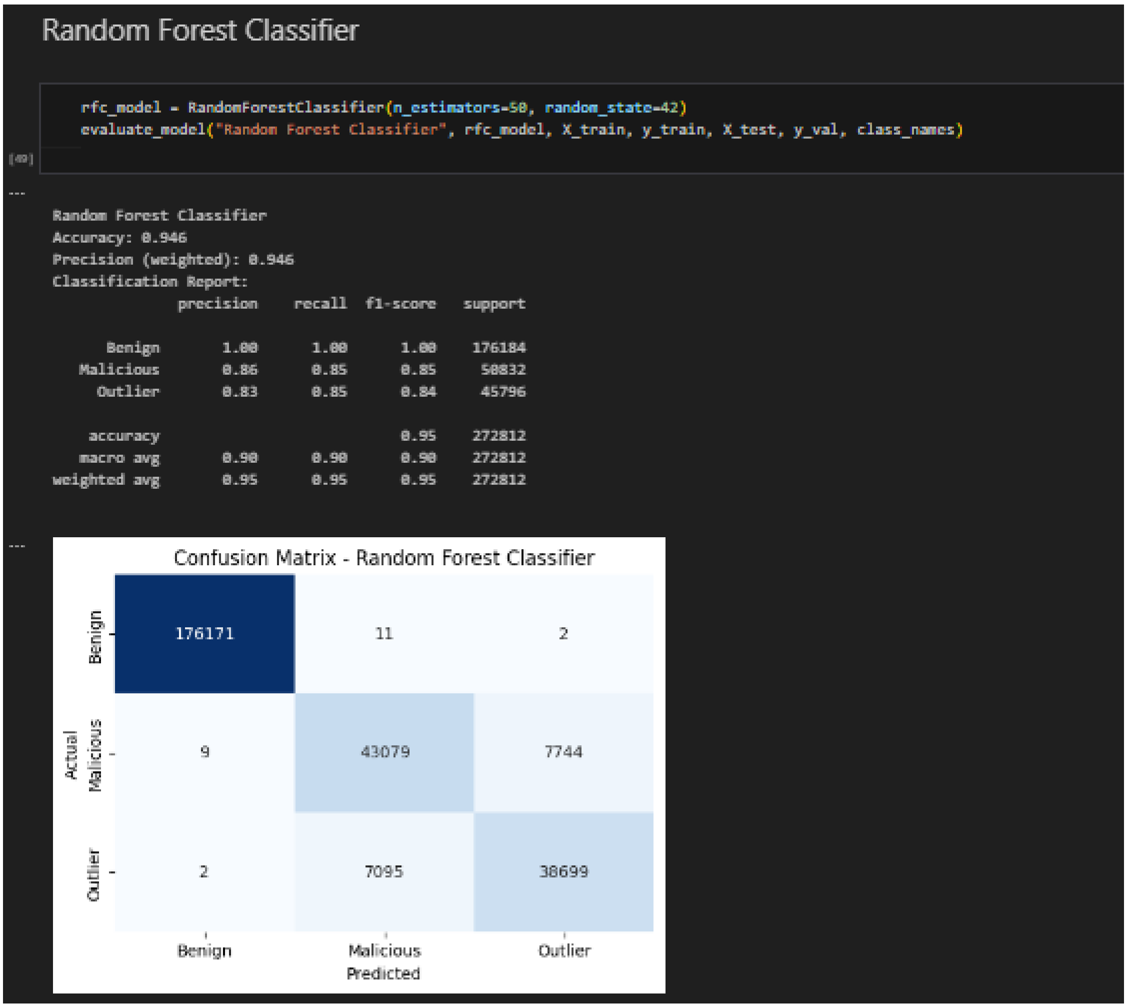
Decision Tree Classifier

```
dtc_model = DecisionTreeClassifier(max_depth=15, criterion='entropy', random_state=42)
evaluate_model("Decision Tree Classifier", dtc_model, X_train, y_train, X_test, y_val, class_names)
```

```
Decision Tree Classifier
Accuracy: 0.939
Precision (weighted): 0.943
Classification Report:
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00    176184
   Malicious       0.90      0.75      0.82     50832
     Outlier       0.77      0.91      0.83     45796

    accuracy                           0.94    272812
   macro avg       0.89      0.89      0.88    272812
weighted avg       0.94      0.94      0.94    272812
```

Confusion Matrix - Decision Tree Classifier

Actual \ Predicted	Benign	Malicious	Outlier
Benign	176169	14	1
Malicious	11	38206	12615
Outlier	2	4039	41755



Random Forest Classifier

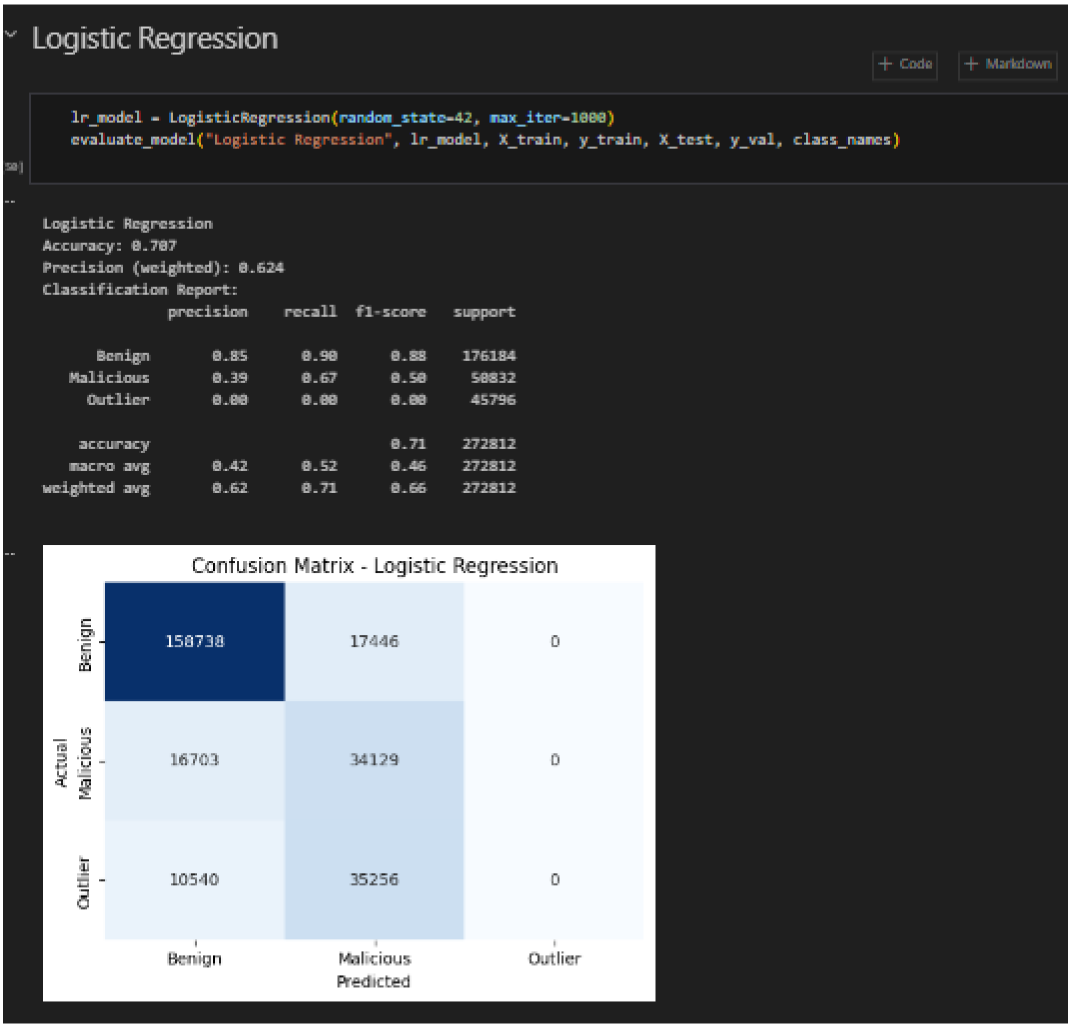
```
rfc_model = RandomForestClassifier(n_estimators=50, random_state=42)
evaluate_model("Random Forest Classifier", rfc_model, X_train, y_train, X_test, y_val, class_names)
```

[49]

```
Random Forest Classifier
Accuracy: 0.946
Precision (weighted): 0.946
Classification Report:
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00    176184
   Malicious       0.86      0.85      0.85     50832
     Outlier       0.83      0.85      0.84     45796

    accuracy                           0.95    272812
   macro avg       0.90      0.90      0.90    272812
weighted avg       0.95      0.95      0.95    272812
```

Confusion Matrix - Random Forest Classifier

Actual \ Predicted	Benign	Malicious	Outlier
Benign	176171	11	2
Malicious	9	43079	7744
Outlier	2	7095	38699



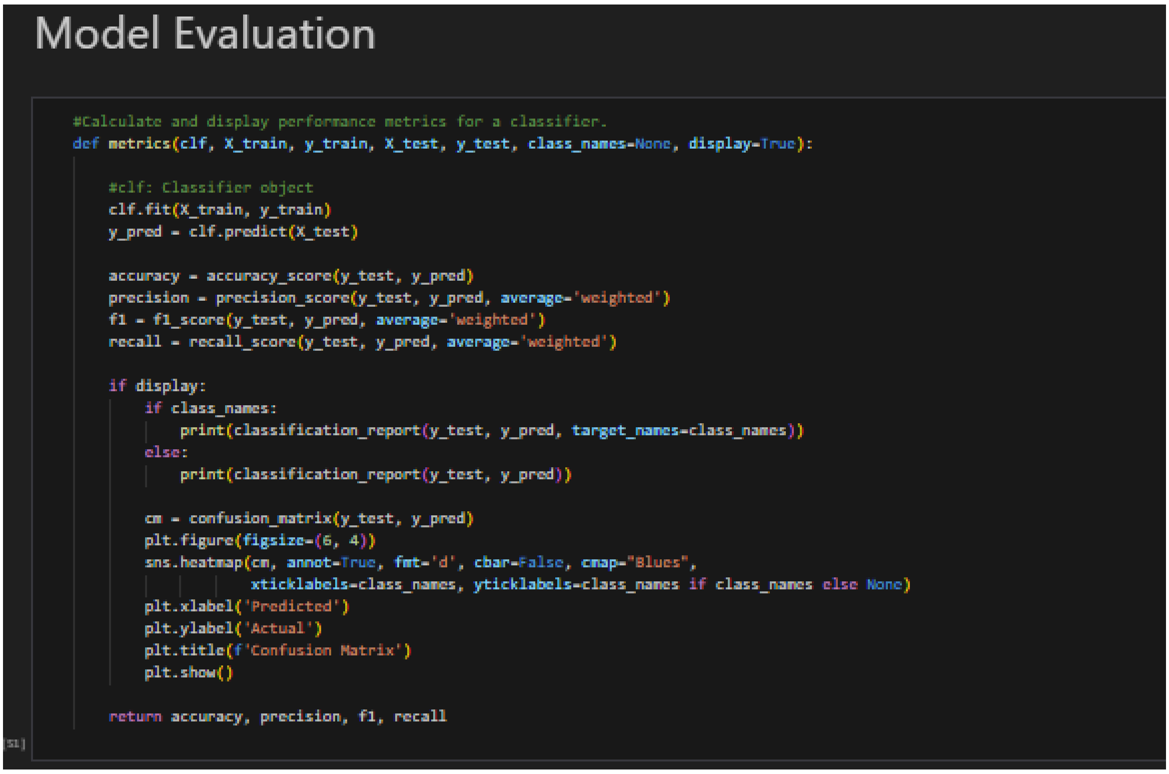
Logistic Regression

```
lr_model = LogisticRegression(random_state=42, max_iter=1000)
evaluate_model("Logistic Regression", lr_model, X_train, y_train, X_test, y_val, class_names)
```

```
Logistic Regression
Accuracy: 0.707
Precision (weighted): 0.624
Classification Report:
              precision    recall  f1-score   support

      Benign       0.85      0.90      0.88    176184
   Malicious       0.39      0.67      0.50     50832
     Outlier       0.00      0.00      0.00     45796

    accuracy                           0.71    272812
   macro avg       0.42      0.52      0.46    272812
weighted avg       0.62      0.71      0.66    272812
```

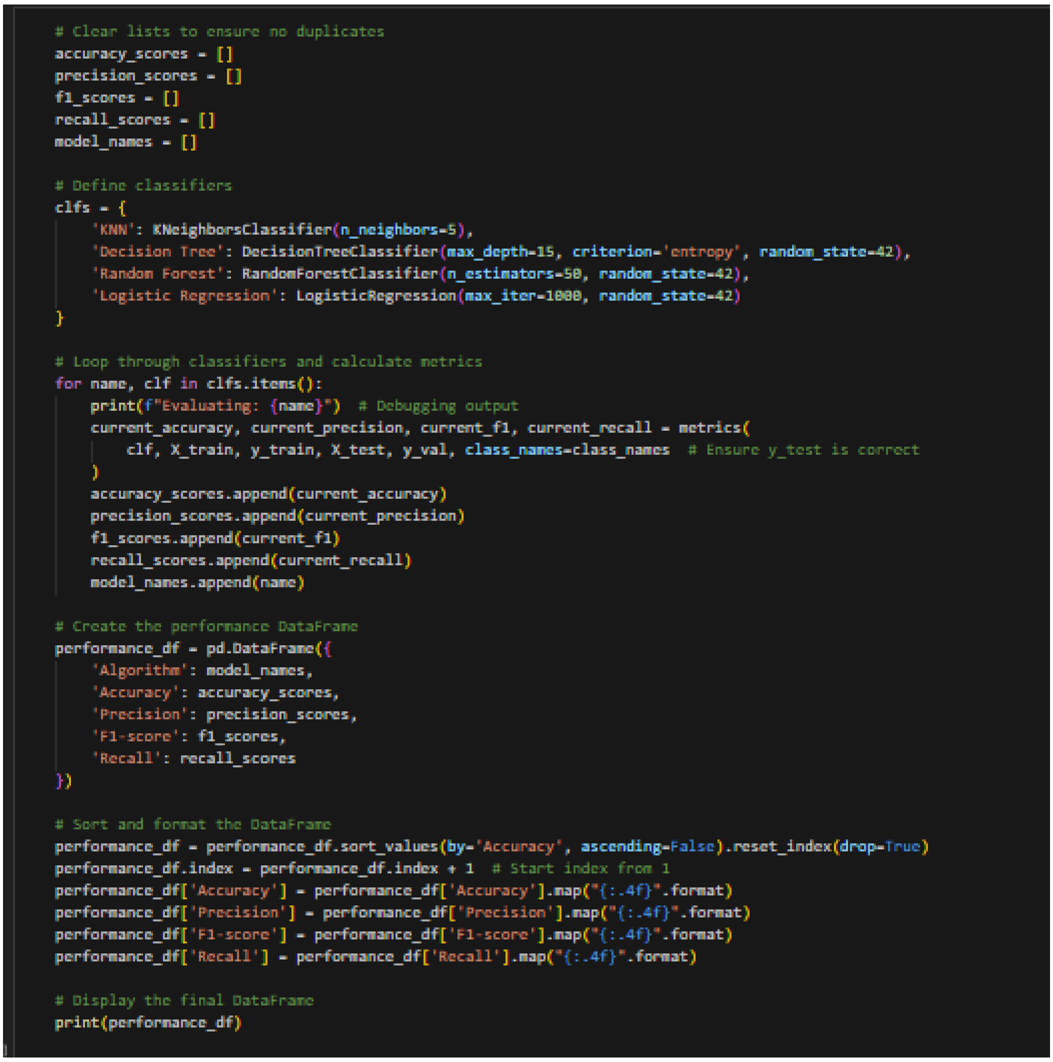
Confusion Matrix - Logistic Regression

Actual \ Predicted	Benign	Malicious	Outlier
Benign	158738	17446	0
Malicious	16703	34129	0
Outlier	10540	35256	0

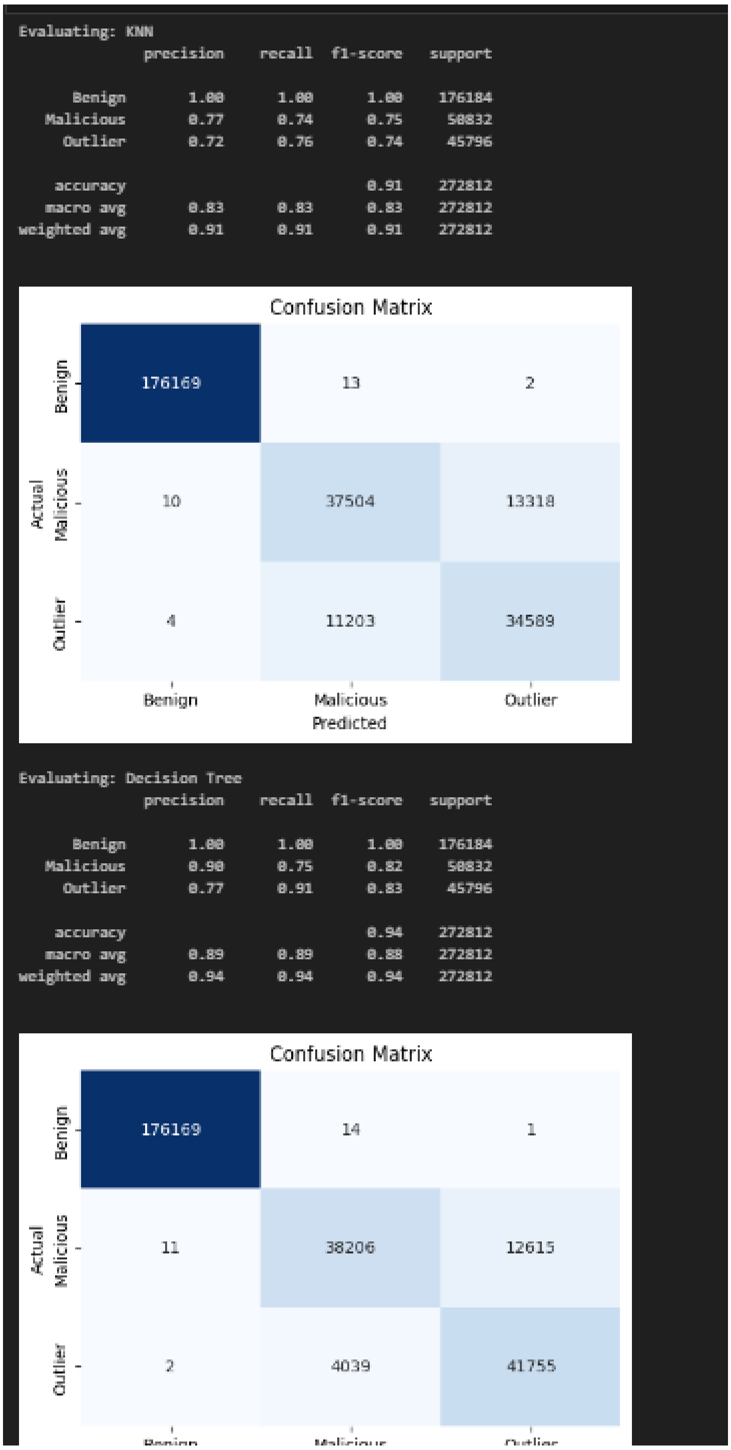


Model Evaluation

```
#Calculate and display performance metrics for a classifier.
def metrics(clf, X_train, y_train, X_test, y_test, class_names=None, display=True):

    #clf: Classifier object
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')

    if display:
        if class_names:
            print(classification_report(y_test, y_pred, target_names=class_names))
        else:
            print(classification_report(y_test, y_pred))

        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt='d', cbar=False, cmap="Blues",
                    xticklabels=class_names, yticklabels=class_names if class_names else None)
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.title(f'Confusion Matrix')
        plt.show()

    return accuracy, precision, f1, recall
```



```python
# Clear lists to ensure no duplicates
accuracy_scores = []
precision_scores = []
f1_scores = []
recall_scores = []
model_names = []

# Define classifiers
clfs = {
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Decision Tree': DecisionTreeClassifier(max_depth=15, criterion='entropy', random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=50, random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42)
}

# Loop through classifiers and calculate metrics
for name, clf in clfs.items():
    print(f"Evaluating: {name}")  # Debugging output
    current_accuracy, current_precision, current_f1, current_recall = metrics(
        clf, X_train, y_train, X_test, y_val, class_names=class_names  # Ensure y_test is correct
    )
    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
    f1_scores.append(current_f1)
    recall_scores.append(current_recall)
    model_names.append(name)

# Create the performance DataFrame
performance_df = pd.DataFrame({
    'Algorithm': model_names,
    'Accuracy': accuracy_scores,
    'Precision': precision_scores,
    'F1-score': f1_scores,
    'Recall': recall_scores
})

# Sort and format the DataFrame
performance_df = performance_df.sort_values(by='Accuracy', ascending=False).reset_index(drop=True)
performance_df.index = performance_df.index + 1  # Start index from 1
performance_df['Accuracy'] = performance_df['Accuracy'].map("{:.4f}".format)
performance_df['Precision'] = performance_df['Precision'].map("{:.4f}".format)
performance_df['F1-score'] = performance_df['F1-score'].map("{:.4f}".format)
performance_df['Recall'] = performance_df['Recall'].map("{:.4f}".format)

# Display the final DataFrame
print(performance_df)
```



```
Evaluating: KNN
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00    176184
   Malicious       0.77      0.74      0.75     50832
     Outlier       0.72      0.76      0.74     45796

    accuracy                           0.91    272812
   macro avg       0.83      0.83      0.83    272812
weighted avg       0.91      0.91      0.91    272812
```

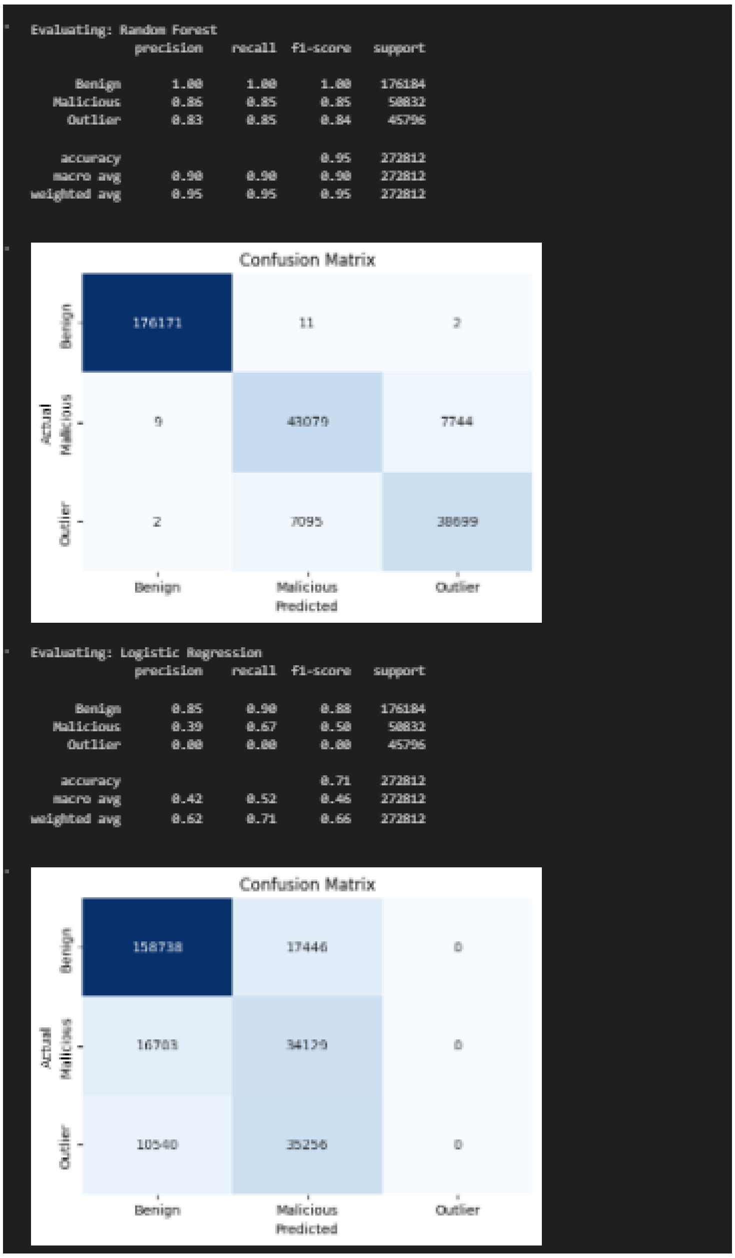
Confusion Matrix

Actual \ Predicted	Benign	Malicious	Outlier
Benign	176169	13	2
Malicious	10	37504	13318
Outlier	4	11203	34589

```
Evaluating: Decision Tree
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00    176184
   Malicious       0.90      0.75      0.82     50832
     Outlier       0.77      0.91      0.83     45796

    accuracy                           0.94    272812
   macro avg       0.89      0.89      0.88    272812
weighted avg       0.94      0.94      0.94    272812
```

Confusion Matrix

Actual \ Predicted	Benign	Malicious	Outlier
Benign	176169	14	1
Malicious	11	38206	12615
Outlier	2	4039	41755



```
Evaluating: Random Forest
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00    176184
   Malicious       0.86      0.85      0.85     50832
     Outlier       0.83      0.85      0.84     45796

    accuracy                           0.95    272812
   macro avg       0.90      0.90      0.90    272812
weighted avg       0.95      0.95      0.95    272812
```

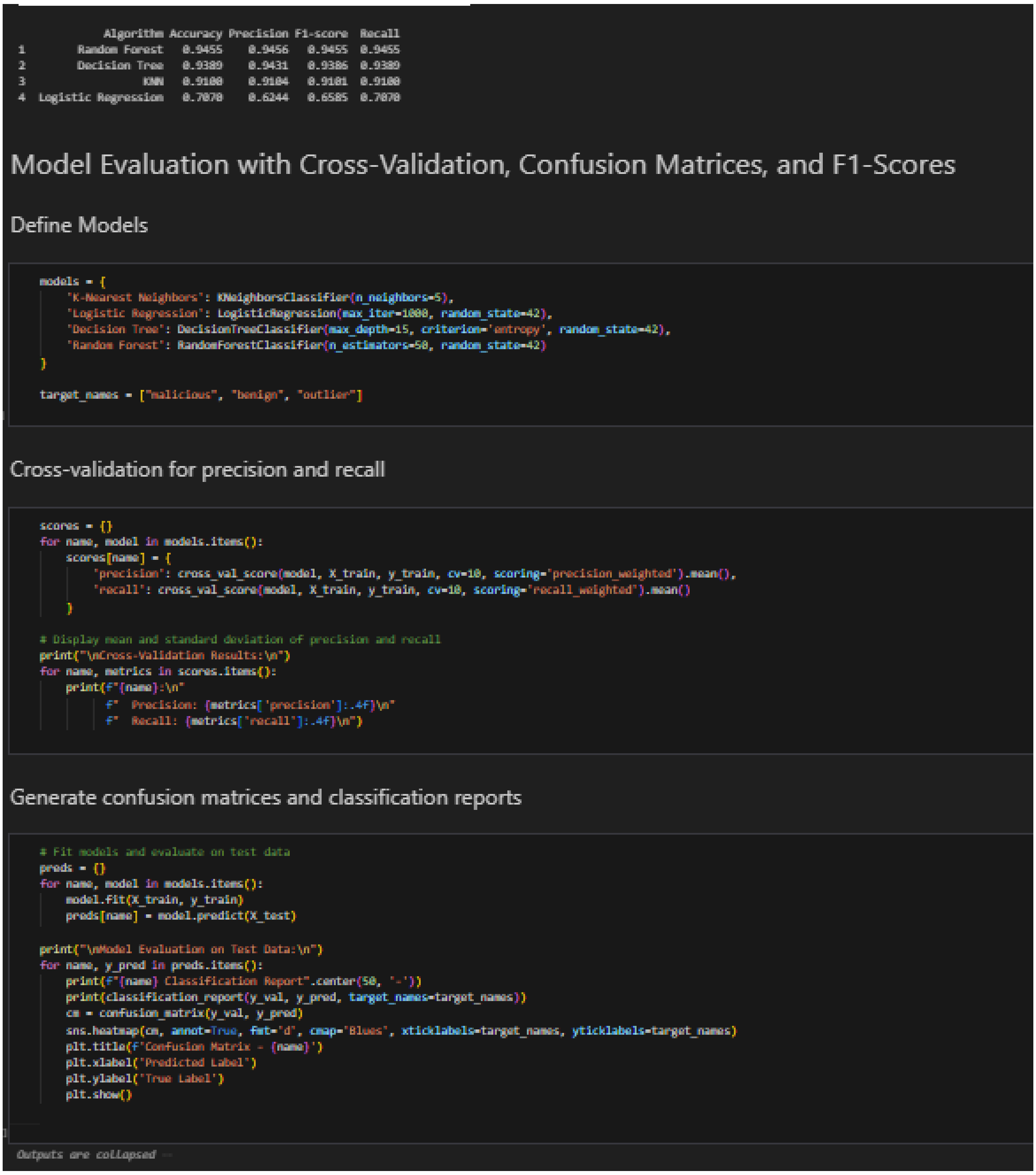
Confusion Matrix

	Benign	Malicious	Outlier
Benign	176171	11	2
Malicious	9	43079	7744
Outlier	2	7095	38699

```
Evaluating: Logistic Regression
              precision    recall  f1-score   support

      Benign       0.85      0.90      0.88    176184
   Malicious       0.39      0.67      0.50     50832
     Outlier       0.00      0.00      0.00     45796

    accuracy                           0.71    272812
   macro avg       0.42      0.52      0.46    272812
weighted avg       0.62      0.71      0.66    272812
```

Confusion Matrix

	Benign	Malicious	Outlier
Benign	158738	17446	0
Malicious	16703	34129	0
Outlier	10540	35256	0



```
            Algorithm Accuracy Precision F1-score  Recall
1       Random Forest   0.9455    0.9456   0.9455  0.9455
2       Decision Tree   0.9389    0.9431   0.9386  0.9389
3                 KNN   0.9100    0.9104   0.9101  0.9100
4 Logistic Regression   0.7070    0.6244   0.6585  0.7070
```

Model Evaluation with Cross-Validation, Confusion Matrices, and F1-Scores

Define Models

```
models = {
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(max_depth=15, criterion='entropy', random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=50, random_state=42)
}

target_names = ["malicious", "benign", "outlier"]
```

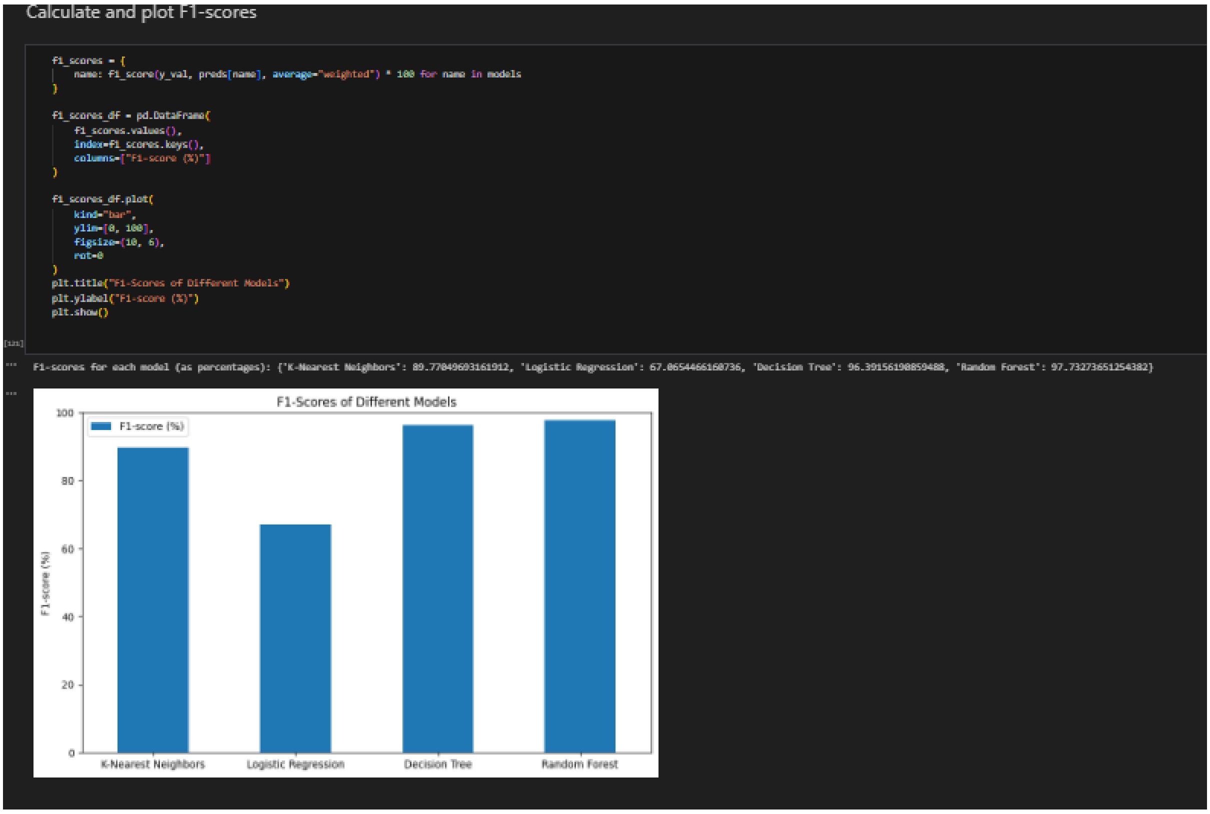
Cross-validation for precision and recall

```
scores = {}
for name, model in models.items():
    scores[name] = {
        'precision': cross_val_score(model, X_train, y_train, cv=10, scoring='precision_weighted').mean(),
        'recall': cross_val_score(model, X_train, y_train, cv=10, scoring='recall_weighted').mean()
    }

# Display mean and standard deviation of precision and recall
print("\nCross-Validation Results:\n")
for name, metrics in scores.items():
    print(f"{name}:\n"
          f"  Precision: {metrics['precision']:.4f}\n"
          f"  Recall: {metrics['recall']:.4f}\n")
```

Generate confusion matrices and classification reports

```
# Fit models and evaluate on test data
preds = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    preds[name] = model.predict(X_test)

print("\nModel Evaluation on Test Data:\n")
for name, y_pred in preds.items():
    print(f"{name} Classification Report".center(50, '-'))
    print(classification_report(y_val, y_pred, target_names=target_names))
    cm = confusion_matrix(y_val, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=target_names, yticklabels=target_names)
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

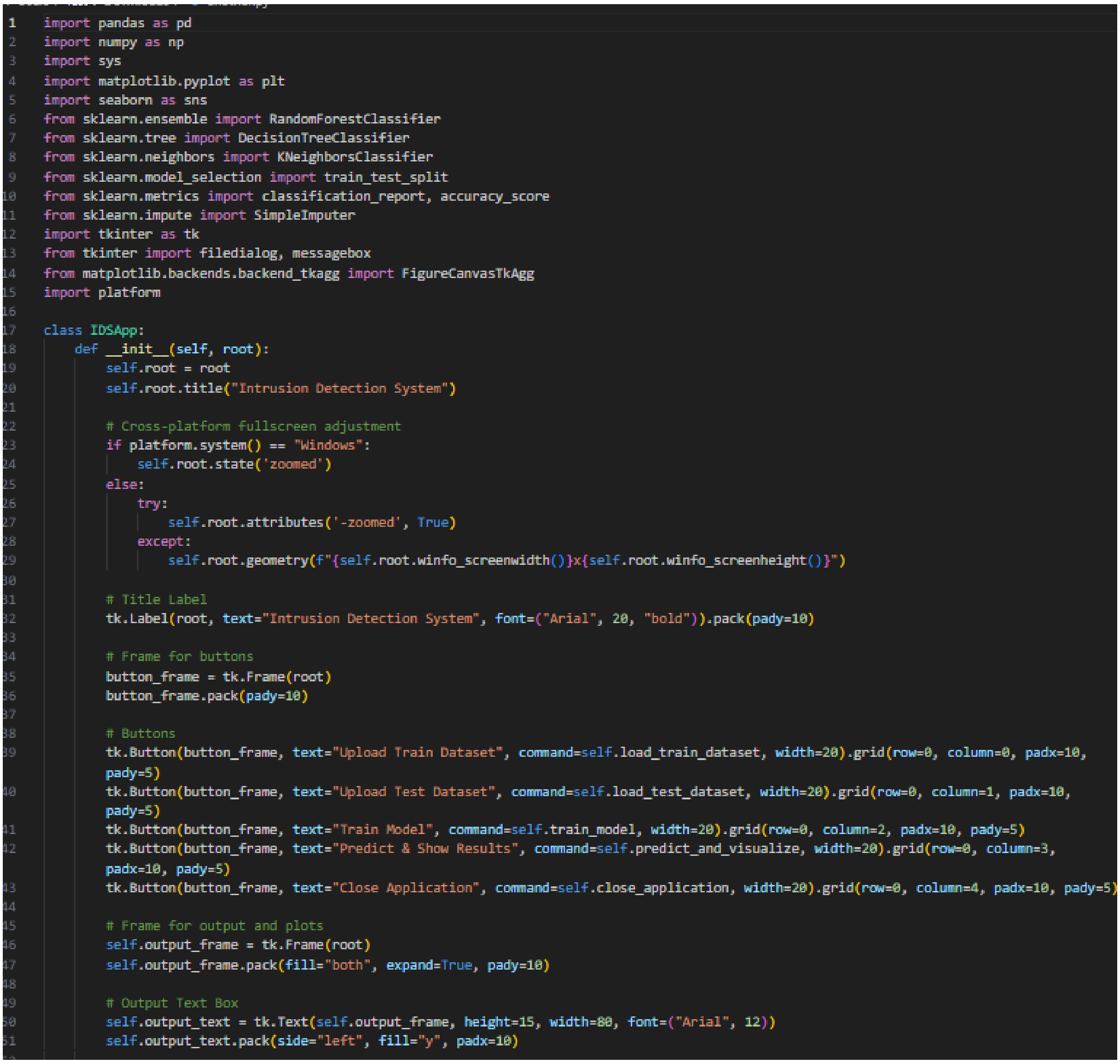
Outputs are collapsed ...



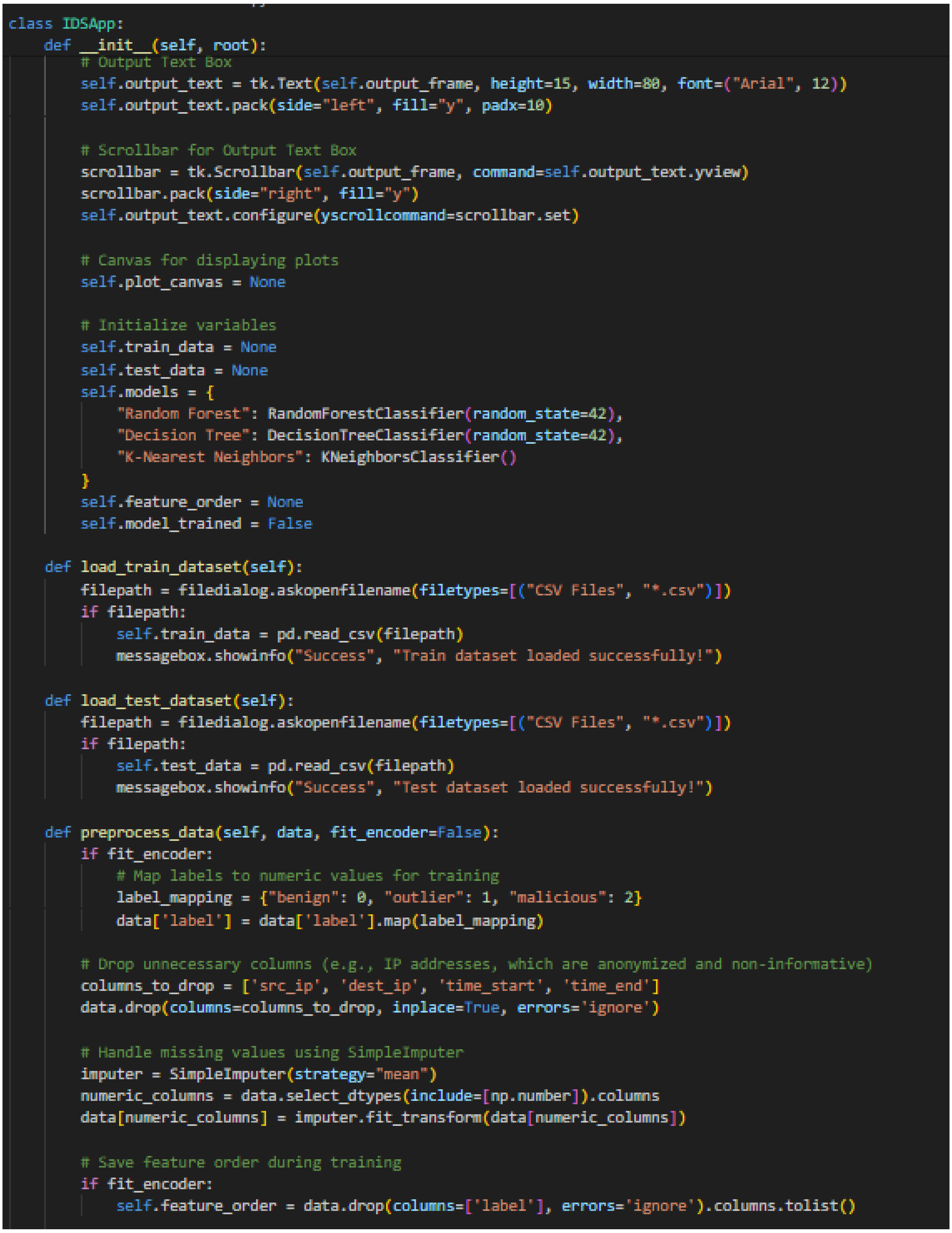
Calculate and plot F1-scores

```
f1_scores = {
    name: f1_score(y_val, preds[name], average="weighted") * 100 for name in models
}

f1_scores_df = pd.DataFrame(
    f1_scores.values(),
    index=f1_scores.keys(),
    columns=["F1-score (%)"]
)

f1_scores_df.plot(
    kind="bar",
    ylim=[0, 100],
    figsize=(10, 6),
    rot=0
)
plt.title("F1-Scores of Different Models")
plt.ylabel("F1-score (%)")
plt.show()
```

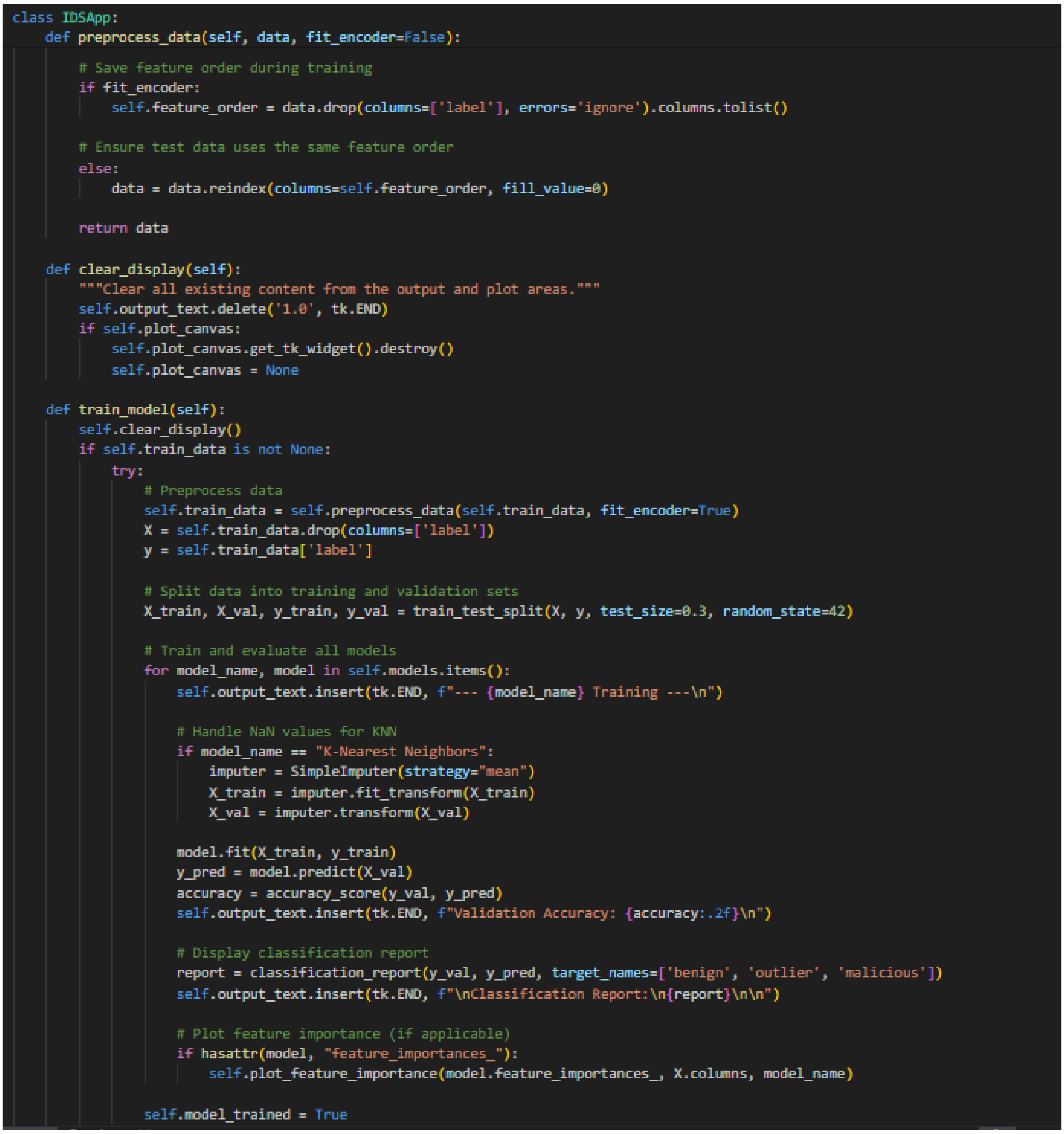
```
F1-scores for each model (as percentages): {'K-Nearest Neighbors': 89.77049693161912, 'Logistic Regression': 67.0654466160736, 'Decision Tree': 96.39156198859488, 'Random Forest': 97.73273651254382}
```



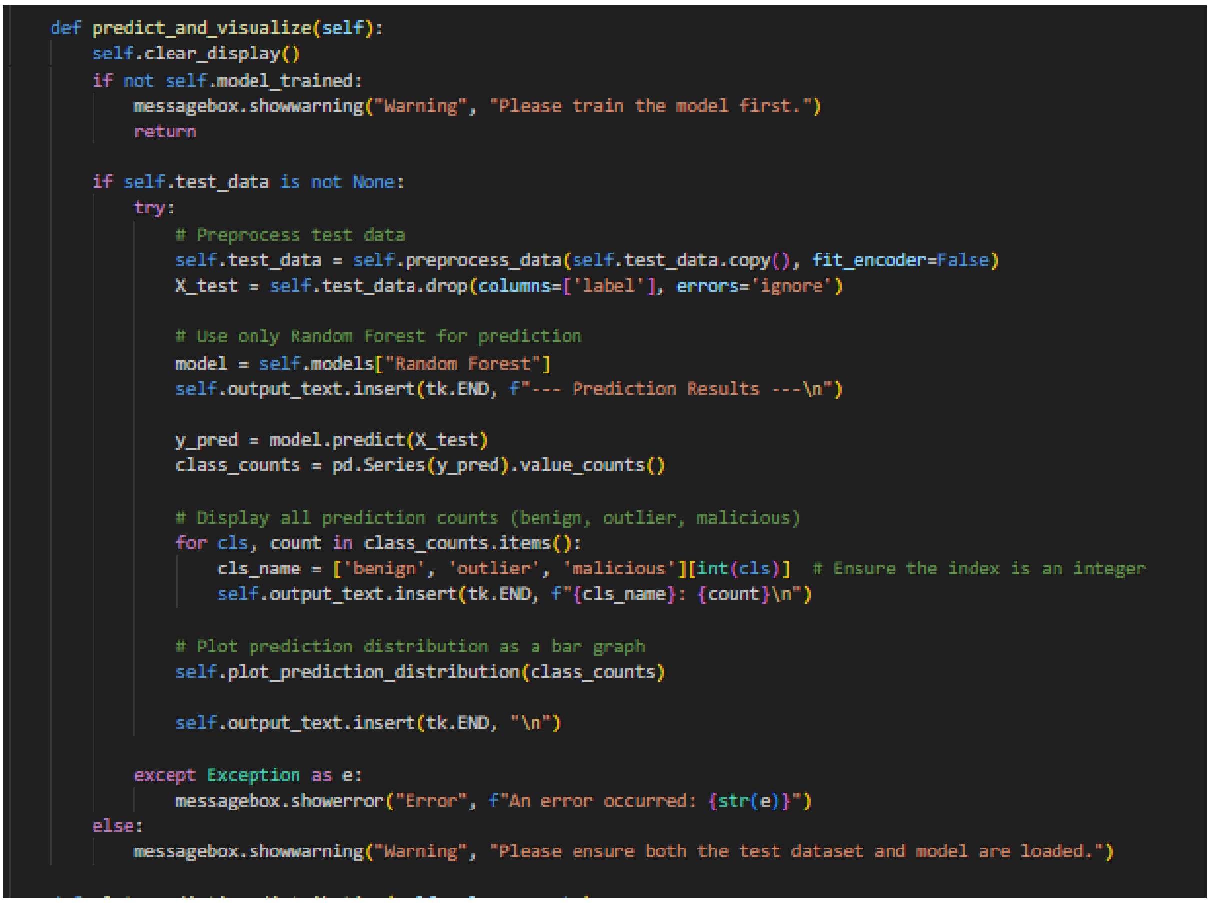
```python
import pandas as pd
import numpy as np
import sys
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer
import tkinter as tk
from tkinter import filedialog, messagebox
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import platform

class IDSApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intrusion Detection System")

        # Cross-platform fullscreen adjustment
        if platform.system() == "Windows":
            self.root.state('zoomed')
        else:
            try:
                self.root.attributes('-zoomed', True)
            except:
                self.root.geometry(f"{self.root.winfo_screenwidth()}x{self.root.winfo_screenheight()}")

        # Title Label
        tk.Label(root, text="Intrusion Detection System", font=("Arial", 20, "bold")).pack(pady=10)

        # Frame for buttons
        button_frame = tk.Frame(root)
        button_frame.pack(pady=10)

        # Buttons
        tk.Button(button_frame, text="Upload Train Dataset", command=self.load_train_dataset, width=20).grid(row=0, column=0, padx=10,
        pady=5)
        tk.Button(button_frame, text="Upload Test Dataset", command=self.load_test_dataset, width=20).grid(row=0, column=1, padx=10,
        pady=5)
        tk.Button(button_frame, text="Train Model", command=self.train_model, width=20).grid(row=0, column=2, padx=10, pady=5)
        tk.Button(button_frame, text="Predict & Show Results", command=self.predict_and_visualize, width=20).grid(row=0, column=3,
        padx=10, pady=5)
        tk.Button(button_frame, text="Close Application", command=self.close_application, width=20).grid(row=0, column=4, padx=10, pady=5)

        # Frame for output and plots
        self.output_frame = tk.Frame(root)
        self.output_frame.pack(fill="both", expand=True, pady=10)

        # Output Text Box
        self.output_text = tk.Text(self.output_frame, height=15, width=80, font=("Arial", 12))
        self.output_text.pack(side="left", fill="y", padx=10)
```



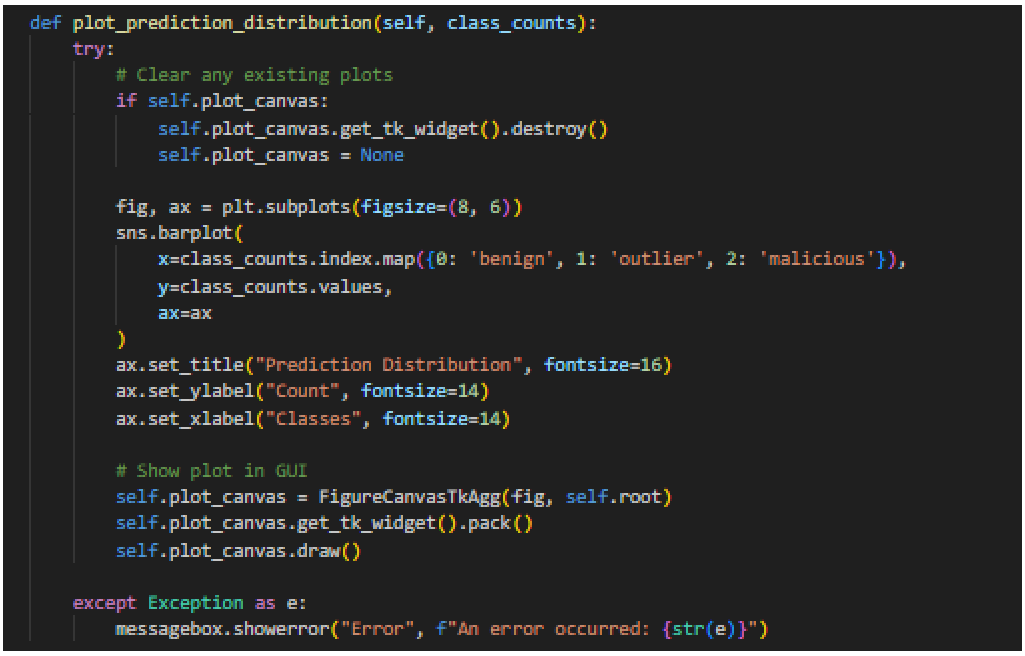
```
class IDSApp:
    def __init__(self, root):
        # Output Text Box
        self.output_text = tk.Text(self.output_frame, height=15, width=80, font=("Arial", 12))
        self.output_text.pack(side="left", fill="y", padx=10)

        # Scrollbar for Output Text Box
        scrollbar = tk.Scrollbar(self.output_frame, command=self.output_text.yview)
        scrollbar.pack(side="right", fill="y")
        self.output_text.configure(yscrollcommand=scrollbar.set)

        # Canvas for displaying plots
        self.plot_canvas = None

        # Initialize variables
        self.train_data = None
        self.test_data = None
        self.models = {
            "Random Forest": RandomForestClassifier(random_state=42),
            "Decision Tree": DecisionTreeClassifier(random_state=42),
            "K-Nearest Neighbors": KNeighborsClassifier()
        }
        self.feature_order = None
        self.model_trained = False

    def load_train_dataset(self):
        filepath = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
        if filepath:
            self.train_data = pd.read_csv(filepath)
            messagebox.showinfo("Success", "Train dataset loaded successfully!")

    def load_test_dataset(self):
        filepath = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
        if filepath:
            self.test_data = pd.read_csv(filepath)
            messagebox.showinfo("Success", "Test dataset loaded successfully!")

    def preprocess_data(self, data, fit_encoder=False):
        if fit_encoder:
            # Map labels to numeric values for training
            label_mapping = {"benign": 0, "outlier": 1, "malicious": 2}
            data['label'] = data['label'].map(label_mapping)

        # Drop unnecessary columns (e.g., IP addresses, which are anonymized and non-informative)
        columns_to_drop = ['src_ip', 'dest_ip', 'time_start', 'time_end']
        data.drop(columns=columns_to_drop, inplace=True, errors='ignore')

        # Handle missing values using SimpleImputer
        imputer = SimpleImputer(strategy="mean")
        numeric_columns = data.select_dtypes(include=[np.number]).columns
        data[numeric_columns] = imputer.fit_transform(data[numeric_columns])

        # Save feature order during training
        if fit_encoder:
            self.feature_order = data.drop(columns=['label'], errors='ignore').columns.tolist()
```



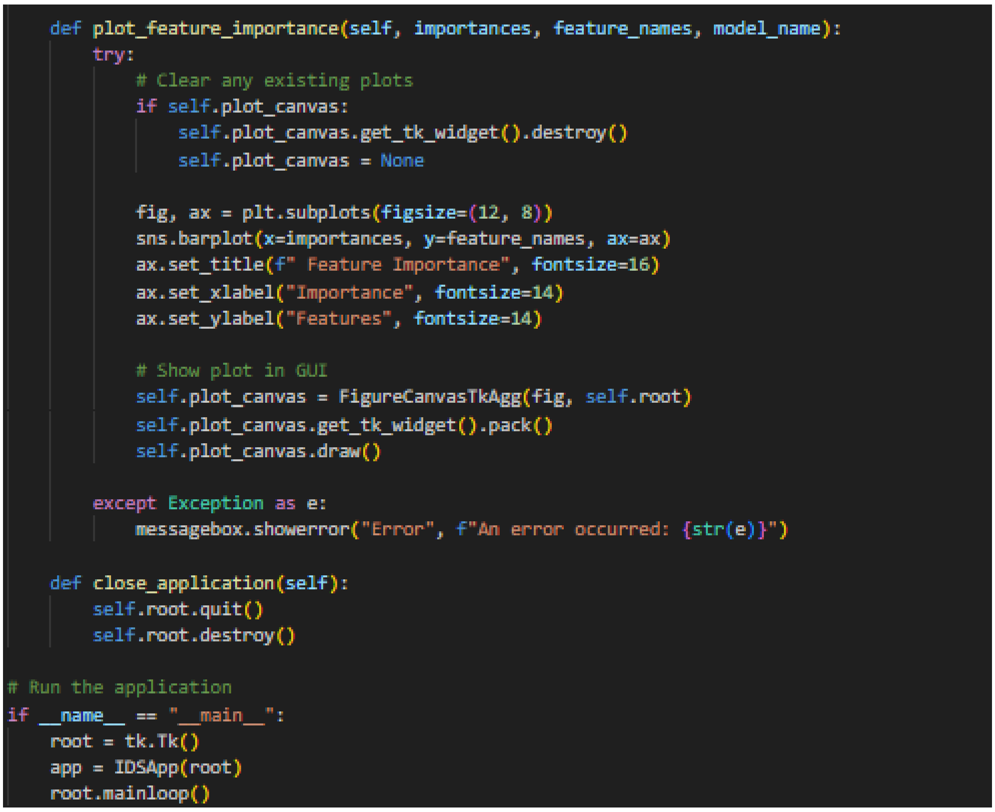
```python
class IDSApp:
    def preprocess_data(self, data, fit_encoder=False):

        # Save feature order during training
        if fit_encoder:
            self.feature_order = data.drop(columns=['label'], errors='ignore').columns.tolist()

        # Ensure test data uses the same feature order
        else:
            data = data.reindex(columns=self.feature_order, fill_value=0)

        return data

    def clear_display(self):
        """Clear all existing content from the output and plot areas."""
        self.output_text.delete('1.0', tk.END)
        if self.plot_canvas:
            self.plot_canvas.get_tk_widget().destroy()
            self.plot_canvas = None

    def train_model(self):
        self.clear_display()
        if self.train_data is not None:
            try:
                # Preprocess data
                self.train_data = self.preprocess_data(self.train_data, fit_encoder=True)
                X = self.train_data.drop(columns=['label'])
                y = self.train_data['label']

                # Split data into training and validation sets
                X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)

                # Train and evaluate all models
                for model_name, model in self.models.items():
                    self.output_text.insert(tk.END, f"--- {model_name} Training ---\n")

                    # Handle NaN values for KNN
                    if model_name == "K-Nearest Neighbors":
                        imputer = SimpleImputer(strategy="mean")
                        X_train = imputer.fit_transform(X_train)
                        X_val = imputer.transform(X_val)

                    model.fit(X_train, y_train)
                    y_pred = model.predict(X_val)
                    accuracy = accuracy_score(y_val, y_pred)
                    self.output_text.insert(tk.END, f"Validation Accuracy: {accuracy:.2f}\n")

                    # Display classification report
                    report = classification_report(y_val, y_pred, target_names=['benign', 'outlier', 'malicious'])
                    self.output_text.insert(tk.END, f"\nClassification Report:\n{report}\n\n")

                    # Plot feature importance (if applicable)
                    if hasattr(model, "feature_importances_"):
                        self.plot_feature_importance(model.feature_importances_, X.columns, model_name)

                self.model_trained = True
```



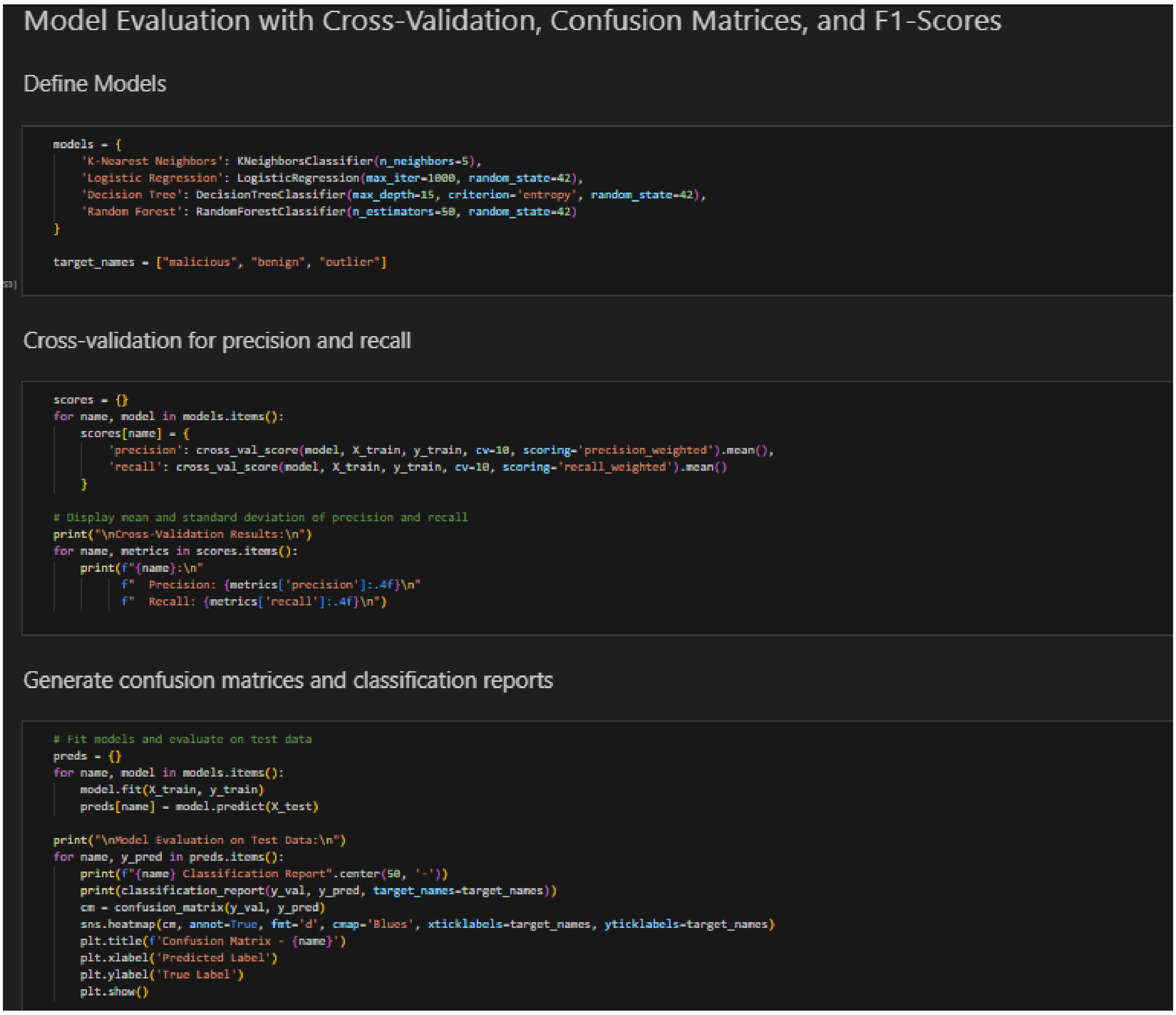
```python
    def predict_and_visualize(self):
        self.clear_display()
        if not self.model_trained:
            messagebox.showwarning("Warning", "Please train the model first.")
            return

        if self.test_data is not None:
            try:
                # Preprocess test data
                self.test_data = self.preprocess_data(self.test_data.copy(), fit_encoder=False)
                X_test = self.test_data.drop(columns=['label'], errors='ignore')

                # Use only Random Forest for prediction
                model = self.models["Random Forest"]
                self.output_text.insert(tk.END, f"--- Prediction Results ---\n")

                y_pred = model.predict(X_test)
                class_counts = pd.Series(y_pred).value_counts()

                # Display all prediction counts (benign, outlier, malicious)
                for cls, count in class_counts.items():
                    cls_name = ['benign', 'outlier', 'malicious'][int(cls)]  # Ensure the index is an integer
                    self.output_text.insert(tk.END, f"{cls_name}: {count}\n")

                # Plot prediction distribution as a bar graph
                self.plot_prediction_distribution(class_counts)

                self.output_text.insert(tk.END, "\n")

            except Exception as e:
                messagebox.showerror("Error", f"An error occurred: {str(e)}")
        else:
            messagebox.showwarning("Warning", "Please ensure both the test dataset and model are loaded.")
```



```python
    def plot_prediction_distribution(self, class_counts):
        try:
            # Clear any existing plots
            if self.plot_canvas:
                self.plot_canvas.get_tk_widget().destroy()
                self.plot_canvas = None

            fig, ax = plt.subplots(figsize=(8, 6))
            sns.barplot(
                x=class_counts.index.map({0: 'benign', 1: 'outlier', 2: 'malicious'}),
                y=class_counts.values,
                ax=ax
            )
            ax.set_title("Prediction Distribution", fontsize=16)
            ax.set_ylabel("Count", fontsize=14)
            ax.set_xlabel("Classes", fontsize=14)

            # Show plot in GUI
            self.plot_canvas = FigureCanvasTkAgg(fig, self.root)
            self.plot_canvas.get_tk_widget().pack()
            self.plot_canvas.draw()

        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")
```



```
    def plot_feature_importance(self, importances, feature_names, model_name):
        try:
            # Clear any existing plots
            if self.plot_canvas:
                self.plot_canvas.get_tk_widget().destroy()
                self.plot_canvas = None

            fig, ax = plt.subplots(figsize=(12, 8))
            sns.barplot(x=importances, y=feature_names, ax=ax)
            ax.set_title(f" Feature Importance", fontsize=16)
            ax.set_xlabel("Importance", fontsize=14)
            ax.set_ylabel("Features", fontsize=14)

            # Show plot in GUI
            self.plot_canvas = FigureCanvasTkAgg(fig, self.root)
            self.plot_canvas.get_tk_widget().pack()
            self.plot_canvas.draw()

        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")

    def close_application(self):
        self.root.quit()
        self.root.destroy()

# Run the application
if __name__ == "__main__":
    root = tk.Tk()
    app = IDSApp(root)
    root.mainloop()
```



Model Evaluation with Cross-Validation, Confusion Matrices, and F1-Scores

Define Models

```python
models = {
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(max_depth=15, criterion='entropy', random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=50, random_state=42)
}

target_names = ["malicious", "benign", "outlier"]
```

Cross-validation for precision and recall

```python
scores = {}
for name, model in models.items():
    scores[name] = {
        'precision': cross_val_score(model, X_train, y_train, cv=10, scoring='precision_weighted').mean(),
        'recall': cross_val_score(model, X_train, y_train, cv=10, scoring='recall_weighted').mean()
    }

# Display mean and standard deviation of precision and recall
print("\nCross-Validation Results:\n")
for name, metrics in scores.items():
    print(f"{name}:\n"
          f"  Precision: {metrics['precision']:.4f}\n"
          f"  Recall: {metrics['recall']:.4f}\n")
```

Generate confusion matrices and classification reports

```python
# Fit models and evaluate on test data
preds = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    preds[name] = model.predict(X_test)

print("\nModel Evaluation on Test Data:\n")
for name, y_pred in preds.items():
    print(f"{name} Classification Report".center(50, '-'))
    print(classification_report(y_val, y_pred, target_names=target_names))
    cm = confusion_matrix(y_val, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=target_names, yticklabels=target_names)
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

Dataset is evaluated with three machine learning models which determine the effectiveness and precision of intrusion detection or accurate prediction of any attacks. Models include k-nearest Neighbor (KNN), Decision Tree, and Logistic Regression. For comparison purposes, we are using indicators like precision, accuracy, recall, and f1-score.