GUI scan system with Machine Learning techniques and CICFlowmeter to analyse Intrusion Detection on Networks

Jawad Mahmud, ID: 001174077

Date: 22 April 2024

Supervisor: Tuan Nguyen

Computer Science, BSc Hons

University of Greenwich

School of Computing and Mathematical Sciences

Word Count: 11,870

Contents

List of Figures

Abstract

Network intrusion detection systems have always had a very crucial role in maintaining the security of the computer network environment against anomalies and other potential malicious behaviour. As a result, several research has been undertaken by many researchers that have tackled many issues in the field. However, they have been able to incorporate machine learning algorithms on various datasets that have received different results overtime. Furthermore, in this project, a GUI system would be implemented using Python and use a variety of extensive libraries that would utilise machine learning algorithms and techniques, as well as utilising the CICIDS2017 dataset and CICFlowmeter, which would help display and analyse the data traffic in the network for potential anomalous behaviour.

Keywords: Machine Learning, network intrusion detection, anomaly, CICIDS2017, accuracy, precision, GUI

Acknowledgements

This project would not be possible without guidance from my supervisor, Tuan Nguyen, and I would like to thank him for providing me some beneficial knowledge in helping me implement the project. I would also like to thank my friends, my parents and my older sister who has always encouraged me to keep achieve my goals and have always been providing me their ongoing love and support throughout the entire challenging period of this project.

1 Introduction

Throughout the entire project topic, the background of the project will be discussed, which is network intrusion detection, as well as the benefits and limitations that may pertain to the subject in mind. Network intrusion detection systems are essentially systems that detects malicious activity or violations, and Sarhan et al. (2022) mentions that they are important tools that protect computer networks against many frequent and sophisticated attacks.

Throughout recent years, network intrusion detection systems have been implemented through the use of artificial intelligence algorithms, most particularly with machine learning algorithms, such as Support Vector Machine, Random Forest, Naïve Bayes and Decision Trees as the prime examples of classification machine learning algorithms, as mentioned by Chou & Jiang (2021).

Moreover, a variety of simulated intrusion detection datasets, such as KDD99, CICIDS2017, NSL-KDD and UNSW-NB15, have been utilised by many researchers and they have been able to evaluate the effectiveness of the datasets with the various machine learning algorithms, whilst implementing some advancements within the security research and implementation. Furthermore, Shaukat et al. (2020) has mentioned that there are several techniques that can be used to detect the intrusions in the network, and machine learning techniques have been a very popular choice in solving intrusion detection situations.

1.1 Background

In the modern era of technology and software, security has always been a very crucial part of how we use our devices, such as laptops, mobile devices, and much more. Across various network technologies, network intrusion is seen as a major drawback in miscellaneous sectors, including finance, education, business, and many more to mention. This level of intrusion is an unknown activity to attempt unauthorised access or cause damage within the network. For instance, Zhang et al. (2022) has mentioned that in 2019, around 620 million details from user accounts had been hacked and had been sold on the dark web, and this led to an increase in the leakage of information daily. This shows that intrusion has become a major problem and has resulted in many damages. Moreover, it can be implemented through the use of machine learning algorithms that would help determine the accuracy and the precision of an anomaly invading.

Moreover, Mukherjee et al. (1994) has stated that the system must provide the services of confidentiality, where the data would be protected against unauthorised access, integrity, in which the data would be protected against unauthorised modification of data and availability, which will ensure that no unauthorised users would access the specific data. An example of an intrusion is that a number of computers had been involved in a Denial-of-Service attack and the attack had exploited various vulnerabilities in each computer system.

In addition, Guezzaz et al. (2021) has mentioned that threats are challenging with advancements that relate to reliability of data and networks, and that they have mentioned that the confidentiality, integrity and availability is the main security goal, in which security policies are implemented to ensure the protection of data and detect the anomalies that may harm the system.

Furthermore, Al Lail et al. (2023) and Umar et al. (2024) talks about the importance of how machine learning algorithms are a crucial part on how intrusion detection is implemented, as the system would be able to learn from the given input and output data from the chosen dataset, then a model is created that would then predict the output from the new input. Apruzzese et al. (2022) has also mentioned that ML is advancing at a rapid pace and has proven to be a valuable asset within the cyber-security domain, as Musa et al. (2020) and Liu & Lang (2019) mentions that they are categorised into supervised and unsupervised ML techniques to distinguish the normal or abnormal behaviour, which indicates that the different data types reflect the behavioural attacks. Keserwani et al. (2021) has also mentioned that these these algorithms classifies the data, and becomes a classifier, in which the supervised and unsupervised techniques are used.

1.2 Aims and objectives

The main goal of this project is to create a tool for detecting network intrusions that will help determine the anomalous behaviour. For the programming language, Python is utilised with a variety of libraries that can be able to support the machine learning algorithms that will be implemented. Libraries such as NumPy, Pandas and Sklearn will be utilised so that when utilising the ML algorithm, it will help with the training and testing data. Moreover, the objectives are to create a GUI system and use CICFlowmeter, which can be found in datthinh1801 (2022), to get the network information, and use that output as the main input for the Machine Learning model, and this is due to the CICIDS2017 dataset being made using CICFlowmeter. In addition, the main features will allow the user to be notified of an intrusion within the network, as well as surveying the network behaviour for potential behaviour. For the GUI to be implemented, the Python library called Tkinter will be implemented for the GUI of the intrusion detection system for a network.

1.3 Project Methodology

The main methodology of the project is through data preprocessing, imbalanced learning, feature engineering and model training, and each of these components within the methodology would help manage how the dataset would be utilised. According to Yang et al. (2022), the first stage within the methodology would be **data preprocessing**, where this would give a different interpretation on how the data within the dataset is handled to ensure a more accurate and complete dataset, which would be essential for reliable ML models. With a high data dimension, time required and low efficiency, the Principal Component Analysis, or PCA, is used for reducing dimensionality. Moreover, data cleaning would be done for validity, accuracy, completeness and uniformity, which would help correct the inaccuracy of the data.

Secondly, **Imbalanced learning** would use a sampling method to create a balanced dataset with overall classification performance and same proportion of positive and negative cases. **Feature engineering** also has methods such as feature selection, which is, as Di Mauro et al. (2021) had mentioned, a set of techniques that optimises the feature space. This would avoid the dimensional disasters, and that irrelevant data features are required for this to avoid information loss, and feature extraction, which is used for creating the model using the PCA algorithm to extract data features for visualisation. Each of the steps in the methodology plays a vital part to manage and enhance the dataset for effective models based on the optimisation of the data.

In addition to this, Stiawan et al. (2020) and Amaizu et al. (2020) have mentioned that feature selection also reduces the dimensionality issue to improve the performance and improve the accuracy of the intrusion, and the CICIDS2017 dataset had been used as a CSV file and removed the redundant features, and used feature engineering techniques on the algorithms with a selected number of features that gave an enhanced accuracy each time. Furthermore, it would then be integrated onto the main user-friendly GUI to monitor and determine the behaviour based on the model, with scaling values varying from 0 and 1, with each value determining benign and malicious behaviour.

2 Literature Review

2.1 Approach to literature searching

Throughout the research for the topic, various articles relating to network intrusion detection and machine learning have been found through sources, which helped enable an understanding of the logic behind intrusion detection. Moreover, the search will focus on how machine learning has been utilised in intrusion detection. In addition, this may help give a major understanding of how network intrusion is done by going in depth of the main problem that shows the essential aspect of network intrusion detection, as well as assessing the main Machine Learning approaches, in which the research of various datasets, algorithms, their performances are carried out.

2.2 Exploring the problem of Network Intrusion Detection

Throughout each research paper over the past decade, various studies have demonstrated that machine learning techniques had been utilised, as well as identifying the importance of an efficient intrusion detection system.

2.2.1 Network Intrusion Detection Techniques

For instance, Mukherjee et al. (1994) had mentioned that intrusion detection is the approach that would be providing security for networks, and its main objective is to identify the potential intrusion within the network and the intrusion detection systems would be able to make use of anomaly based models, so that it would be able to detect intrusions. In addition, it would be able to ensure that the services of confidentiality, integrity and availability would be provided in the network intrusion detection system, so that the network can be used without the presence of an malicious anomaly.

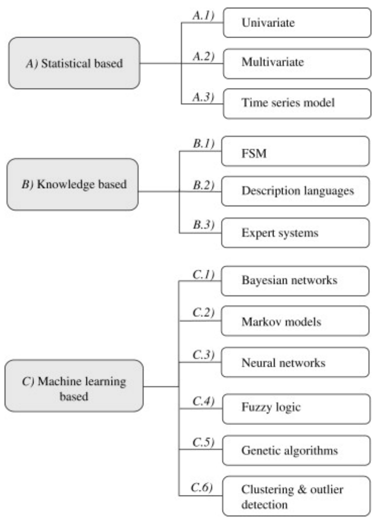


Figure 1: The diagram shows the anomaly based techniques that have been used for network intrusion detection. Garcia-Teodoro et al. (2009)

According to Garcia-Teodoro et al. (2009), there have also been different anomaly-based techniques that have been used to detect the problem of an anomaly invading the system. The main benefit of those techniques is that they detect unseen intrusion events that could occur within the system. However, the problem is that those techniques could result in a higher false positive rates. There are three types of anomaly-based techniques that have been identified and used for intrusion detection.

The diagram, as shown in **Figure 1**, shows the three fundamental basics of anomaly-based techniques, firstly, there are statistical based A-NIDS technique, which is used on metrics like the network traffic rate, connection rate, IP addresses, and so on, after they capture the traffic activity with a profile with the unusual behaviour being created. Secondly, knowledge-based techniques are utilised and they are known to be a more robust technique, with the sub types of the techniques being the widely used expert systems, which helps classify the data via training data and classification procedures. Finally, machine learning techniques for an anomaly-based intrusion detection system would be established through the use of a model that would analyse the pattern.

2.2.2 Taxonomy and NIDS types

For the taxonomy of IDS, as mentioned by Jamalipour & Murali (2021), they have been classified into three categories, the first category being the ***data source***, with one type being the *host based* IDS, which examines the data originating from the host, and covers sources such as the operating system it is hosting, and the logs for the server, firewall and database, whereas network based IDS monitors the network traffic through packet capture and other network data sources, and they observe the activities before further damage.

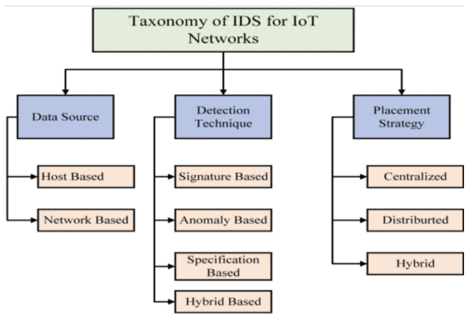


Figure 2: The taxonomy of intrusion detection system, and what techniques are used. Jamalipour & Murali (2021)

Moreover, ***detection techniques*** are done through four ways; the first technique is based on attack **signature**, where it would be able to determine the problem within the attack through an attack signature within the system, **anomaly-based**, which defines the set of normal and abnormal behaviour within the pattern, **specification-based**, which differentiates anomalies depending on the feature deviations, and finally **hybrid-based**, which combines at least two of the approaches to improve benefits and reduce drawbacks. Chaudhari & Patil (2017) has mentioned that whilst signature based detection is a simple and effective method with high detection rate, it can only detect known attacks only and needs regular updates of rules which are used, whereas for anomaly based detection, it can examine unknown and complicated intrusions, but there is a low detection rate and high false alarm, and that the model must be trained carefully.

2.2.3 Advantages and Disadvantages of each IDS types

Latif et al. (2022) had mentioned that for each type of IDS, they each have their own advantages and disadvantages, for instance, hybrid-based IDS uses both signature and anomaly-based detection to minimise the drawbacks, and the data is labelled as an anomaly if the packets are malicious. Finally, placement strategy have been utilised, which include a centralised placement that utilises the data traffic to pass through to identify the attack, distributed placement, which configures all network nodes with full implementation for a more efficient way to detect an attack, and a hybrid placement, which will combine both placements and benefits from a more efficient detection of the oncoming attack.

2.3 Key technologies/examples of Intrusion Detection in use

Whilst the background of intrusion detection is a major factor in the cyber security industry, it is known that there have been a variety of tools that have been used for detecting attack or benign behaviour as a security measure. The more notable tools that are known for intrusion detection as a whole include SNORT and Wireshark. Jain et al. (2021) has mentioned that SNORT uses a sequence of policies that outline or describe the network activity, whether it would be a malicious or benign network behaviour, and it uses the policies to discover the packets that have no match and would alert the user about the malicious behaviour. Moreover, Wireshark analyses the network protocols as to where the attacks come from, and is widely used by various cyber and network security experts and professionals. It is mainly used to view, analyse and capture the data packets and notice the main patterns. These helps the administrators trace the problems that would cause multiple issues such as performance and the potential vulnerabilities in the packet.

In addition, Engelen et al. (2021) has stated that datasets like CICIDS2017 and KDD99 have been constructed from packet capture (PCAP) files through the use of CICFlowmeter, and the output is the CSV files that allows the rows to correspond to a flow, and Ilyas & Alharbi (2022) has mentioned that the traffic flows are identified, and the last column/feature of the CSV file is usually the class label, particularly a common attack or benign behaviour, and that is only specific to the dataset.

2.4 Applications of Machine Learning in Intrusion Detection

Every research paper had talked about the various ML algorithms that had been utilised as a part of their research, and the algorithms has made a significant accuracy compared to each algorithm and dataset utilised. In addition to evaluating the effectiveness of several different machine learning algorithms, it is essential to take into consideration the adaptability and adaptation of these models to various datasets and real-world circumstances.

As essential as it is to attain outstanding precision on the dataset in question, it is also critical to assess the results of these algorithms on new, untested data. Subsequent investigations may focus on carrying out exhaustive cross-validation tests, studying transfer learning strategies, and evaluating the effects of hyper-parameter adjustment on the ability to be generalised and overall performance of the models. Das et al. (2021) has stated that ML techniques has been extensively utilised to improve the intrusion detection through automation and prediction on an advanced level and they help become robust by tackling various attacks.

In addition, Maharaj & Khanna (2014) have stated that classification techniques are in two categories; binary classification, which classifies elements in a given set in two groups with or without characteristics, and multiclass classification classifies instances into more than two classes. An example from Elmasry et al. (2019) has stated that for NSL-KDD dataset, the test distribution is shown as 43% for Normal behaviour, whereas attack behaviour is shown as 57%, which has lead the dataset to be more suitable for binary classification rather than multiclass.

2.4.1 Benchmark IDS Datasets

Moreover, each paper has mentioned that datasets had been utilised for the main purpose of training and testing the data to detect a potential anomaly. Chowdhury et al. (2020) has mentioned that when choosing a dataset, it is a major task that the right dataset must be found to address the problem with ML techniques to ensure the training and testing model is unbiased. One example of a dataset utilised in intrusion detection is **CICIDS2017**, as explained in Maseer et al. (2021) and Kilincer et al. (2021), in which the CICIDS2017 data would be pre-processed through numericalisation, which would replace null symbols with zeros, followed by normalization due to the large values. Next, the dataset would be divided into training and testing data and supervised and unsupervised ML algorithms would be applied for identifying the efficiency. Finally, the performance of the algorithms would be done through the use of performance metrics such as accuracy, precision and recall.

Furthermore, there are other popular benchmark datasets, as explained by Ahmad et al. (2021), that had been utilised for intrusion detection, such as **KDD Cup'99**, which is regarded as the most popular dataset, that had millions of records for the training and testing data for denial of service, or DoS, attacks, as well as UNSW-NB15, which contains attacks such as worms, DoS, port scans, and Kyoto 2006+, which is a dataset that had been created from network traffic data of known and unknown anomalies. In addition, Di Mauro et al. (2021) has mentioned that whilst KDD99 is the broadly used and the widely used dataset utilised for machine learning algorithms, it does not reflect the modern data traffic, as it has sensibly changed throughout the progression of the data traffic.

Ghurab et al. (2021) has compared the widely used datasets as a way to determine each of their characteristics, and the benchmark datasets that are mentioned in the detailed analysis for intrusion detection include KDD99, CICIDS2017, UNSW-NB15, NSL-KDD and Kyoto 2006+. Firstly, KDD99, as mentioned by Ahmad et al. (2021), is the main pioneer of datasets, as it is a dataset that had become widely used for evaluating the anomaly. However, despite the age of the dataset, it consists of roughly 4,898,431 instances that are within 42 features, as well as 22 training types. It is used by most researchers as the dataset had been useful for training and testing the proposed model. Meanwhile, Abdallah et al. (2022) has discussed that the NSL-KDD, KDD'99, CICIDS2017 and UNSW-NB15 datasets have been experimented with feature selection and each dataset has been effective on improving the classification performance by using various algorithms to an extent.

Secondly, Ghurab et al. (2021) has also mentioned that the NSL-KDD dataset is known to be improved version of KDD99, as it was able to raise major issues that have affected the accuracy, and that KDD99 has had an enormous amount of duplicate packets, which would affect the training set on the machine learning methods as it would stop the detection of attacks, which would leave the system to be more vulnerable. However, compared to KDD99, the testing dataset would have 22,544 instances with the training dataset having 125,973 instances.

In addition, both the UNSW-NB15 and CICIDS2017 datasets are comprised of modern normal and attack behaviours and used for authentic evaluation for a network intrusion detection system. The research conducted had been able to provide an overview on what each dataset are and was able to discuss the importance in which that each of the datasets' properties are suitable in terms of providing quality.

2.4.2 ML Algorithms and how well they perform

Asharf et al. (2020), and Chindove & Brown (2021) have talked about how each Machine Learning technique has been applied, and they have talked about ML algorithms and how they are utilised, such as K-Nearest Neighbor (KNN), where they have utilised Euclidean distance to measure the distance between each neighbour, and the classification algorithm is used to classify the data traffic as normal or abnormal, which has also mentioned by Hajj et al. (2021), decision trees are used by extracting the dataset's sample features and organising decision nodes based on the value of a feature, which would minimise overlapping between different classes in the training dataset, and Support Vector Machine, which is a classification algorithm that is found through a maximum distance of the nearest data point, as well as the benefit of using less storage and memory, and shows more accurate results compared to other algorithms.

This demonstrates that the ML techniques had been applied in intrusion detection with different algorithms and that the dataset would be sampled to provide a sample with the relevant features that could help ensure the effectiveness of these algorithms in intrusion detection. Moreover, dataset sampling had been utilised to put more emphasis on the major parts of the set, and provide the training and testing data samples that is necessary.

Figure 3, as shown in Kocher & Kumar (2021) shows the supervised and unsu-

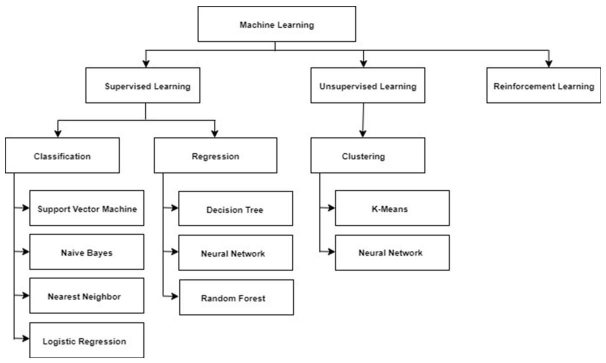


Figure 3: Various ML algorithms that have been utilised in Intrusion detection. Kocher & Kumar (2021)

pervised Machine Learning algorithms that had been utilised by researchers in intrusion detection. For instance, as shown in the image, Support Vector Machine and Naive Bayes are main examples of classification algorithms, in which it involves the intrusion classifications as binary or multi-class, depending on the behaviour of intrusions and attack categories.

According to Ferdiana et al. (2020) and Figure 4, the Support Vector Machine is the most implemented algorithm that had been researched, indicated that, in ***Figure 4***, **34%** of researchers have used the classifier, followed by Neural Networks, which had been used by **20%** in research studies and Decision Trees, which had been researched by **17%** of the studies that has used these methods as a part of their research.

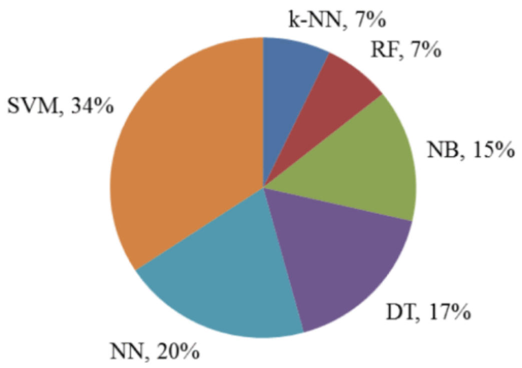


Figure 4: The most popular algorithms for intrusion detection. Ferdiana et al. (2020)

Moreover, the performance of machine learning with intrusion detection discusses how effective the system and the algorithms are, which would mainly be dependent on the dataset being utilised and it would help determine the accuracy of detecting the cyber threats and reduce the false positives and false negatives. Due to the common use of machine learning within the field of intrusion detection, a confusion matrix is utilised, as shown in Table 1 below. A confusion matrix is utilised for the performance metrics for the test data, which is usually generated after the model fitting in the classification algorithm, as explained by Agarwal et al. (2021).

Acharya et al. (2021) have experimented on both the UNSW-NB15 and CICIDS2017 datasets with various ML classifiers, and when comparing the experimental results between each dataset, it shows for both binary and multi class classifications, the CICIDS2017 dataset has a higher performance compared to UNSW-NB15, for instance, the true positive rate for the binary and multiclass classification for Naive Bayes using UNSW-NB15 is 0.911 and 0.684, whereas for the multiclass classification using the CICIDS2017 dataset is from 0.918 to 9.982 altogether, indicating a strong detection rate for the CICIDS2017 dataset.

2.5 Evaluation Metrics used for Intrusion Detection

2.5.1 Confusion Matrix

Four factors are followed within the confusion matrix, as shown on Table 1 and stated by Tait et al. (2021), in which the predicted and actual result values are seen after the classification of the data.

1. **True Positives (TP)** - the predicted and actual results are both positives, and in the context of intrusion detection, the anomaly had been predicted and correctly identified as a potential intrusion
2. **True Negatives (TN)** - where the anomaly had the predicted and actual result be negative, in which the predicted result is that the anomaly is not malicious and the actual result confirms that it is not malicious.
3. **False Positive (FP)** - as stated by Repalle & Kolluru (2017), it means that the attack had been predicted as an attack, but it is actually not an attack but rather normal behaviour in the system.
4. **False Negatives (FN)** - the anomaly is predicted incorrectly as an attack and that the actual result is incorrect.

Tait et al. (2021) has also mentioned that the evaluation metric is useful for visually understanding how efficient the algorithm is able to run, e.g. if more FP or FN readings are presented, it is a sign that the algorithm must be improved to identify the attack. Each of these rates would then be determined through the use of the confusion matrix, so that there is an insight for each algorithm's strength and weaknesses.

		Actual	
		Positive	Negative
Predicted	Attacks (Positive)	True Positives (TP)	False Positives (FP)
	Benign (Negative)	False Negatives (FN)	True Negatives (TN)

Table 1: Confusion Matrix

2.5.2 Performance Metrics

Intrusion Detection Systems are always been evaluated in different ways with datasets, and according to Kumar (2014) and Ahmad et al. (2022), they also depend on the effectiveness of the ML techniques after measuring the performance for classification. Based on the confusion matrix, they use the following metrics based on the variables of the matrix:

1. **Accuracy** would be calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

   The accuracy equation is utilised the percentage of the classifications of both true positive and true negative rates, which is divided by the number of the total number of classifications. As stated by Maseer et al. (2021) has used various ML algorithms for CICIDS2017, and noticed that each algorithm has received a different accuracy, for instance, Support Vector Machine has achieved a 96.72% percent, with K-Nearest Neighbors achieved 95.38%, Decision Tree attaining 97.23% and the Naive Bayes classifier has achieved 98.86%, and these indicates that is how much the models have been classified correctly.

2. **Precision** would be calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

   The Precision equation is utilised to determine and calculate the positive rates that have been correctly identified out of all the positives that have been calculated altogether. Panwar et al. (2022) had mentioned that Random Forest had achieved a 99.89% of flagged attacks being labelled as true attacks, which shows how crucial it is to minimise the unnecessary response to false positives.

3. The **Recall** of the machine learning algorithm performance would be calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

   And this is the main performance metric where the classifier would calculate the correctly identified attacks out of the correctly identified instances and positive instances that have not been detected. Suppose a classifier correctly identifies 90 attacks, whilst failing to detect 20 attacks, the recall for that would be 81.8%.

4. The **F1-score** is notable utilised to calculate the harmonic mean of precision and recall:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

   For the classification of the models, the equations are utilised and a classification report is created that represents the precision, recall and f1-score, and the averages of the metrics, as explained by Di Mauro et al. (2021) and Al Lail et al. (2023). Moreover, the mentioned survey has mentioned that the algorithms have been evaluated to indicate which ML model has provided a much stronger result.

2.6 Challenges of Intrusion Detection

Despite the benefits of intrusion detection, there are some notable challenges. As stated by Alshamy & Ghurab (2020), high false positives could result in a very catastrophic rate for the network. This is a very crucial issue that would allow the attacker to exploit the vulnerabilities of the network very easily. Moreover, if there are more false negatives, there could be a potential alert despite the packet being normal, and this would result in too much time consumption and wasted effort. This is a recurring challenge as it sends the user the wrong message and that it would be displayed as an intention to mislead them. Moreover, during data preprocessing, the dataset would contain a very sizeable and numerical data based on the list of attacks, and Alshamy & Ghurab (2020) mentions that due to the sizeable amount of data, it must have a low complexity for training and testing data to learn new attack behaviour.

When the datasets had been discussed by Ghurab et al. (2021), it had been discussed that they would not focus their attention onto older datasets such as KDD99 and NSL-KDD, and this is mainly due to age of these datasets, and that they would not be able to correspond to the modern standards of intrusion detection. This may cause many issues as this could lead to the ML intrusion detection models to be ineffective and that they do not represent the modern and current landscape of cyber attacks in the cyber security environment. Also, Liu & Lang (2019) has stated that studies use extensive preprocessing methods that results low efficiency.

Thakkar & Lohiya (2022) has stated that whilst research has been made with intrusion detection, it has been known to be dependent in a realistic environment. Further challenges include the evaluation strategy, as they mentioned that a common test has not been found that shows the generated dataset would consist of the network traffic. Secondly, the performance of computational intelligence techniques has not been explored, as well as the machine learning methods being widely used on a public dataset and the comparison not explored, despite a strong result for the detection rate and accuracy. More importantly, Bertoli et al. (2021) had stated that imbalance in datasets could cause different ML algorithms to decrease the performance on how effective the algorithms are, because it may usually depend on how discrepant the behaviour would be.

In addition, Lanvin et al. (2022) have mentioned that whilst going over the CICIDS2017 dataset, some of the packets have been misordered, which is an issue that they have been able to notice when using CICFlowmeter, which is a tool to extract the network flow descriptions from pcap files that have network captures contained, as well as issues relating to the labelling of attacks, in which they stated that manual labelling multiple networks related to the common attacks would be expensive and subject to error.

2.7 Conclusion

In conclusion, anomaly based intrusion detection systems are considered to grow and expand rapidly with machine learning and how the techniques can be used for monitoring the traffic patterns. Throughout the history of security, it has been know to be an essential tool that will help detect potential anomalies within the network and data traffic that could cause more harm. Moreover, the essential parts of the anomaly based intrusion detection system, such as the algorithms and datasets that have been conducted from various research articles, have been discussed and mentioned the necessary requirements that would be able to help determine on whether the traffic is normal or abnormal.

Additionally, with the integration of machine learning techniques onto intrusion detection, anomaly-based intrusion detection systems are an important enhancement in network security. As advances in technology occur, the degree of complexity and sophistication of cyber threats advances and renders old rule-based detection systems less effective. Machine learning appears to be an appealing choice due to the fact that it permits intrusion detection systems to evolve and gain insight from immense quantities of network data, strengthening the system's ability for identifying formerly unidentified threats and anomalies.

Furthermore, the accessibility of a vast and extensive dataset serves as essential to constructing powerful machine learning models for intrusion detection. Researchers as well as professionals have made significant contributions towards this domain by curating and sharing datasets containing labelled examples of normal and abnormal network traffic, which has enabled the development as well as evaluation of machine learning-based intrusion detection technologies.

As a whole, the combination involving machine learning algorithms as well as intrusion detection demonstrates significant potential towards enhancing organisational security methods and drastically reducing cyber threats. The systems in question can continuously identify and mitigate security issues through the use of powerful algorithms, massive datasets, and continuous learning capabilities, thereby safeguarding essential resources and infrastructure from unauthorised access and malicious activity.

Moreover, the benefit of using machine learning algorithms for intrusion detection systems is that using continuous learning will help adapt and evolve whilst experiencing zero-day attacks and other new potential threats, in which the model would be adjusted to enhance the capability of intrusion detection. Nevertheless, with these capabilities in mind, it would allow the threat to detected quickly and reduce time taken to identify the threat intrusion. By keeping track of the network traffic on the systems with the aid of machine learning techniques, it would help identify the anomaly and reduce the threats to a significant extent.

These findings have motivated the study that seeks to accomplish the following objectives. Firstly, CICFlowmeter would be used for the live detection in the network behaviour for any benign or malicious behaviour and convert the raw data onto a CSV file for use, which would be integrated onto a GUI, and secondly, there would be an option to use that CSV file so that the user can be able to see which of the behaviours in the columns are proven to be an anomaly. What makes this stand out more is that whilst CICFlowmeter is used to collect the network capture packets, the GUI would have the option to allow the user to identify where the behaviour is anomalous and where it is benign.

3 Main Chapters

The main chapters will help with the critical aspects of implementing a Network intrusion detection GUI scanning system with Machine learning. The key steps will involve the analysis, functional and non-functional requirements, design, implementation, testing and the evaluation of the system.

For analysis, it involves gathering various research that helped generate an idea, and in particular, the analysis for the CICIDS2017 dataset, as well as the use of machine learning and creating a GUI with the use of CICFlowmeter would be discussed and why it would be such an interesting idea. Secondly, the functional and non-functional requirements what would be necessary for the implementation, as well as the design that illustrates what the system would look like in the final stage. Furthermore, the implementation stage involves the required software components utilised to create the system, with the testing phase using test cases to verify a valid functionality and evaluation will review the performance and results of the final product, as well as discussing what lessons have been learnt and the problems encountered throughout the entire process.

3.1 Analysis

Based on the research applied for the project, each major aspect of the project will be justified based on the uses within the project. Moreover, the major aspects such as the usability of machine learning and the main dataset that had been utilised for the project will be critically analysed to prove how feasible the project is to the necessary requirements. Furthermore, the benefits and drawbacks of the project would be discussed throughout the entire project.

By critically evaluating the main characteristics not just proves the project's feasibility but also increases the probability of success, and by acknowledging the limitations of the project, it would be able to provide a strong effectiveness throughout the network infrastructure.

Q: How can intrusion detection help contribute to security in the network infrastructure?

3.1.1 Applying Machine Learning

When going over how intrusion detection works, Umar et al. (2024) had stated that it is widely used for classifying patterns and designing IDSs. This is due to the Machine Learning area being a crucial and important part of how intrusion detection works as a whole, as it is able to detect the network behaviour through the accuracy and precision. The accuracy and the precision of the ML algorithms would mostly depend on the dataset that is used. In addition, it would also clarify on which features of the dataset would be necessary, as it would be beneficial to remove the redundant features to optimise the dataset for a better performance.

Many Machine Learning algorithms such as Support Vector Machine, Random Tree, Decision Tree have been utilised through many extensive research and has been able to provide many insightful results. As mentioned by Maseer et al. (2021) in their research, a variety of supervised Machine Learning algorithms have each been used for classifier testing that had involved a number of models that would be related to the training models, and that each algorithm had been able to provide a significant result above a 95% based on the metrics that have been utilised. This shows that in relation to intrusion detection as a whole and throughout multiple research that has been conducted, there have been many ML algorithms that had caused such a significant impact. In addition, each result for the algorithms would vary through the choice of datasets, and this addresses the fact that each performance would be different and effective to an extent.

Q: How can the use of machine learning algorithms and model can be able to become more beneficial for detecting intrusions in the network environment?

3.1.2 Overview of the CICIDS2017 dataset

According to the research that had been established for the main project idea, Oyelakin et al. (2023) and Ahmad et al. (2021) have both stated that the CICIDS2017 dataset is a large and representative dataset in which it contains the modern common attacks with the real world data attacks. Many attacks have been implemented in the simulated environment in a span of five days from Monday to Friday, and Toupas et al. (2019) has stated each row contains 83 features that has been extracted from the network traffic, which had been analysed from forward and backward directions, using the information that has been based on the timestamps, source, destination IP addresses, protocols and attacks.

Moreover, when going over the main dataset, each pcap file, or packet capture file, had been categorised based on which day from Monday to Friday and which part of the day from Morning to Afternoon, and what behaviour have been detected in the times between 08:00 to 17:00. In addition, the main span of attacks happening in the five day period, between Tuesday and Friday, with Monday being the only day that has benign behaviour only, meaning no attack behaviour has been labelled throughout the entirety of Monday, which therefore means that it is labelled normal behaviour. Kilincer et al. (2021) has stated that the CICIDS2017 dataset can be found in the 'Canadian Institute for Cybersecurity' on the University of New Brunswick website, with the CSV files being utilised that is comprised of each attack flows based on working hours, as the CSV files are mainly used for machine learning and deep learning purposes.

This is the main dataset that is utilised for the project, as it has been provided with a list of labelled common attacks, which includes Brute Force FTP and SSH, Denial of Service, or DoS, Heartbleed, Web Attacks, Infiltration, Botnet and Distributed Denial of Service, or DDoS, in which Kumar et al. (2020) had discussed about the dataset. These have been identified to determine what attacks have been captured in packets. The main reasons for choosing the dataset is that the dataset is more modern and is associated with common attacks in real-world data from the pcap files, as well as the dataset using labels based on their behaviour, which makes it an ideal dataset that would make it easier to evaluate the Intrusion detection system performance.

3.1.3 A GUI for the intrusion detection system

Moreover, the researchers have discussed what kind of tools that have been utilised throughout the case involving intrusion detection as a whole. A GUI would be implemented, where the user will be able to check for potential attack behaviour that may result in the network being compromised. The idea of the GUI scanning system was inspired through the use of Wireshark, and the reason being is due to how the tool can be able to observe the patterns and potential anomalies, for example, if an unusual network activity had been detected, it would simply display the data for the users to view the potential issue. In addition, using machine learning algorithms for the GUI scanning system would be able to enhance the detection of the anomalies and the algorithms can be trained to recognise the behavioural patterns that is associated with common attack or benign behaviour. Moreover, the CICFlowmeter tool would be utilised as a part of the system, so that the users are able to perform a live scan and convert the information into a CSV file.

Q: How can the main approach for the detection system be able to determine on whether the network behaviour is benign or malicious?

3.2 Requirements Specification

The GUI scanning system for detecting network intrusion would be designed to address the importance of network security. Using Machine Learning techniques would help identify the potential intrusions that have been present during the scan with CICFlowmeter.

To ensure that the system is efficient to detect the network intrusions, the main requirements would be utilising the Python programming language, as well as the essential libraries such as NumPy, Pandas, Sklearn, os, sys and Tkinter, in which these libraries would be beneficial for maintaining the effectiveness and efficiency of the Machine Learning algorithms, and Tkinter to implement the Graphical User Interface, or GUI. In addition to this, it is vitally important to follow the methodology that is comprised of data preprocessing, imbalanced learning, and feature engineering, which has key methods including feature selection and feature extraction. Furthermore, the specification will outline the main functional and non-functional requirements for the scanning system for detecting network intrusions, and provide a discussion for both types of requirements.

3.2.1 Functional Requirements

The functional requirements will mention the specific features and functionalities that the GUI scanning system with machine learning must have in order to be able to possess the major needs of what the system should do. Moreover, the methodology discussed would be mentioned as a major functional requirement, as well as the use of the machine learning algorithms and the GUI scanning system.

- **Preprocessing and Feature extraction**: For this process to work, it would use the CICIDS2017 dataset to extract the relevant features, and it would be required for it to go through the preprocessing stage. This would be able to transform the raw data into a suitable format, in this instance, preferably a CSV format, in which it would benefit the analysis for machine learning and ensure that the network packet is analysed.
- **Machine Learning Predictability:** The system would be able to utilise the algorithms to analyse the extract the features and identify any patterns of interest that may be able to determine the network intrusions, and the supervised learning algorithms would be implemented for the classification tasks. In addition, model evaluation metrics, which is comprised of accuracy, precision, recall and F1 score, would be used to evaluate how effective and efficient the machine learning algorithms and models are when it comes to detecting any potential intrusion that could be potentially malicious. Furthermore, a training and testing model shall be implemented and integrated onto the GUI to determine the behavioural pattern of the data traffic.

- **Interactive GUI scanning system:** Finally, the GUI must be able to provide a user-friendly interface that would enable the user to monitor the network for potential intrusions, as well as display the behaviour on whether the activity is malicious or benign. It should also be able to provide the option for notifying that there is an intrusion detection in the network and display scanned behaviour from CICFlowmeter when the user uses the CSV file to analyse the traffic patterns based on the model predictions.

By taking the functional requirements into account, they would help ensure that it meets the necessary requirements, and that the ML GUI scanning system is able to effectively monitor the network data traffic, analyse the data patterns and determine what that behaviour indicates based on the model for an efficiently beneficial system. In addition, integrating the machine learning algorithms and model would allow the system to adapt to the network and detect anomalies, and enhance the effectiveness and reliability to prevent the intrusions.

3.2.2 Non-Functional Requirements

For non-functional requirements, it helps define the main attributes and characteristics within the functional capabilities of the system, therefore being the main crucial aspects to determine the quality of the system. In terms of the scanning system that is to be implemented with Machine learning in Python, the non-functional requirements would be gaining a particular and important role to ensure the performance, usability, reliability and security.

- **Usability**: it is an extremely crucial non-functional requirement for the scanning system, which would impact the satisfaction of users when they use the software. It would have a very user-friendly interface, with a simple navigation, and this is essential as this can help users be able to use the system a lot easier, and when they use the software, it would enable them to interact with various buttons and elements that would result in a smoother experience. In regards to security, they can quickly access the functionalities, such as monitoring the behaviour surrounding the network, or view the alert and what threat it would be able to identify. Therefore, ensuring the usability of the network intrusion detection system would result in satisfaction, as well as a major effectiveness in regards to monitoring and responding to the network intrusions.

- **Reliability**: The reliability of the system is absolutely critical for an accurate detection to maintain the behaviour of the network, and it may be done through minimal downtime to ensure that the operation would allow the system to continue monitoring the network despite potential difficulties.

- **Security**: this would enable the system to keep the network environment safe, as well as keeping data safe, as well as ensuring that the integrity of the system has been maintained and take into consideration on what the . The robust security is an important aspect so that it may safeguard the system and that the ML techniques could help optimise the security.
- **Performance**: due to the use of machine learning being a major aspect of the scanning system in mind, it would directly impact how the important aspects of the system would be able to respond. The GUI would be capable of processing the oncoming network data via live scanning with CICFlowmeter, which would require data preprocessing, as well as the optimised algorithms to be utilised for feature extraction and the use of machine learning. Moreover, the scalability of the system would be crucial as this would allow the system to handle the huge data volumes without degrading the performance. Therefore, the performance and efficiency of the algorithms would help ensure that the GUI scanning system for network intrusion detection with machine learning is able to respond well and make it convenient to use.

By taking these non-functional requirements into consideration, it is able to help emphasise the importance of the reliability, security, usability, performance and the scalability. The non-functional requirements that have been discussed are vital to ensure the efficiency and ease of use for the system to detect, monitor and respond to the network intrusion. Moreover, these help ensure that the project is able to meet the expected standard, and be able to provide a significant impact.

3.3 Design

After maintaining the functional and non-functional requirements for the GUI scanning system, the most crucial stage would be through the design. This is to ensure the concept of implementing the system to be able to give out more clarifications and conciseness, as well as providing a strong emphasis with a very intuitive graphical representations that would enhance a stronger understanding within the network. To provide a visual and graphical representation of how the scanning system would be shown, a variety of diagrams, as well as the conceptual designs for the Graphical User Interface (GUI), will be shown and be discussed on how they would benefit the logic of the system.

3.3.1 Flowchart Diagram

A flowchart has been created to identify the expected inputs and outputs through a sequence in the process, and it can help ensure that the GUI is able to provide logic, as well as being intuitive and easy to use.

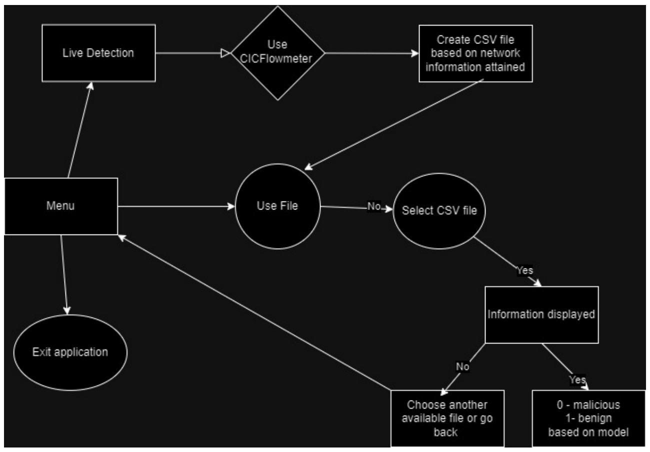


Figure 5: The flowchart shows the logic of how the GUI will be able to operate.

When the user starts to run the system, a menu is displayed, and the first step represents the point in which the user can initialise a network scan, and once they start the scan, it will move onto the next step involving the process of monitoring the network behaviour. It will analyse the network data traffic for potential intrusions and malicious activity, that may indicate the most unusual behaviour, and with the Machine Learning algorithms, analyse the patterns that have been identified throughout the scan. Furthermore, if the system has managed to detect the unknown activity, then the user may be alerted that an intrusion has been detected, otherwise, they will have the option to keep scanning, or stop scanning, which enables them to go back to the start.

To represent what the system does or how the user is able to utilise it, a yes/no decision point has been made based on how the system is run, or how the user is using the system, for instance, it may determine whether an intrusion has occurred throughout the scan or not, and if an intrusion is detected, it would be shown through the standard scaling. Afterwards, once the user has finished using the system for live detection, and no longer requires it, they will have the option to use the CSV file to check the data in which it is labelled as malicious or benign, and this is a beneficial feature in the application that shows the user if the intrusion attempt had been found. The flowchart has been able to represent the steps that ranges from starting a scan, as well as monitoring the network behaviour for intrusions and detecting the potential activity.

3.3.2 Use Case UML Diagram

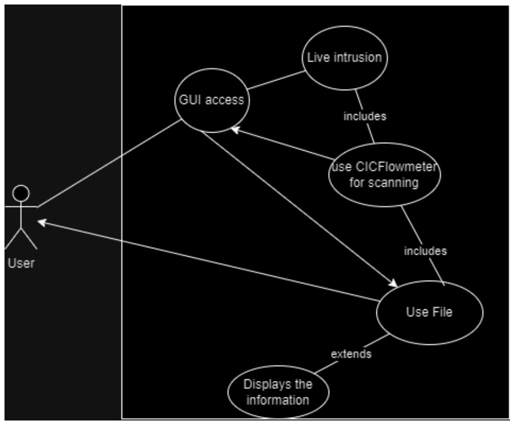


Figure 6: The use case UML diagram demonstrates the interactions of the GUI would be able to utilise.

The use case UML diagram is a critical component that showcases the system's design, as well as the functionality, and it would provide a stronger representation of what the GUI scanning system is expected to do. Moreover, the Machine Learning techniques can be able to help with viewing the network traffic by analysing the distinct patterns that may be able to prove that there is an intrusion that has been identified, and it would be able to provide an intelligent way to improve the accuracy and efficiency of the system, which would help maintain and user-friendly and interactive way to access the system.

In this instance, the user is able to access the GUI, and can be given two major options; perform a live intrusion scan, or use a CSV file to analyse the behaviour. If the user performs the former option, then they would use the CICFlowmeter tool as a way to start scanning the network. After that, they can go back to the GUI for the latter option, and the CSV file would then display the network information, as well as display detailed information on row click. This is to enable the user to be able to go through the information without the need to go through each row to find their choice.

3.3.3 GUI conceptual designs

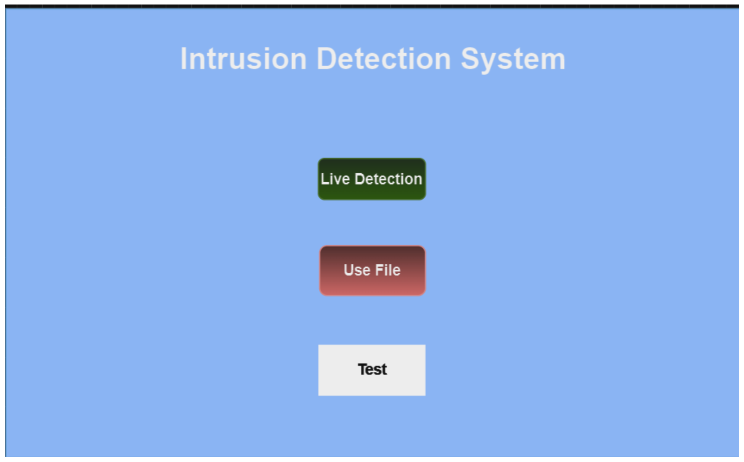
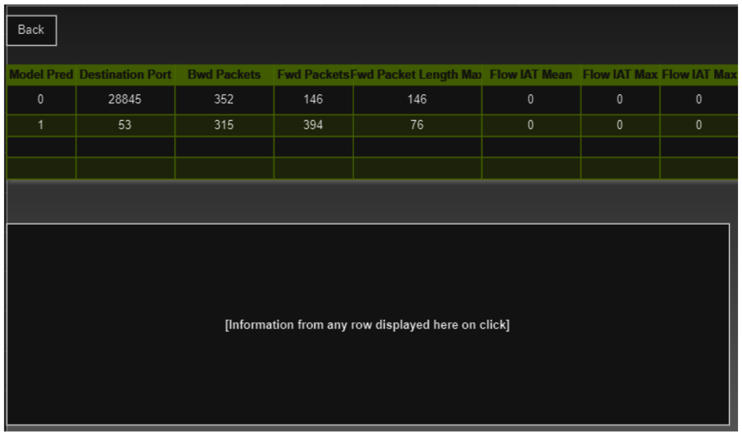


Figure 7: The design for the GUI scanning system menu, which the user will see when opening the software.

The main GUI menu design shows what the user can view when on start up, and they provide a simple and intuitive way to navigate through the two primary options in the menu, which are 'Live Detection' and 'Use File'. The main primary function of the 'Live Detection' option is that it can enable the user to use the system to begin scanning the data traffic with CICFlowmeter so that they can be able to notice particular patterns that may be able to indicate that there is a potential intrusion detected after creating and using the CSV file. This would be a very beneficial feature in the system, as this would enable the user to have the option to be able to perform a live scan of the network behaviour, and when it has been done, it would display a list of rows and columns through a CSV file with the information in the network and that they can be able to identify the patterns to indicate that the behaviour is benign or malicious.

In addition to the GUI design, a 'Use file' button is included onto the GUI, which is the secondary function for the menu, and what it does is that when the user has finished a live scan and all the data had been converted into a CSV file, which would display the packet information, as well as display the rows in red or white, which would indicate that a standard scaling with 0 (malicious) and 1 (benign). The reason behind the incorporation of the secondary function is to grant the main user the option to view the data as well as display the detailed information when the live scan has been completed, so that the user has a strong emphasis on the current activity that is being monitored. Moreover, when implementing the interface, the ML algorithms and techniques will be utilised for the system to continue analysing the traffic patterns, and would be able to adapt to recognising the potentially malicious intrusion attempt.

In summary of the menu design, the GUI menu design for the scanning system is able to offer two main options, which are 'Live scanning' and 'Use file', in which the first option would allow the user to perform a live scan that would help aid in detection of potential network intrusions through the use of machine learning techniques. Whereas, the option to use the CSV file in the application is able to provide the user to emphasise on the behaviour in which type of behaviour can be considered normal or anomalous. This would help enhance the usability of the NIDS as well as enabling the users to get to grips with the main interactions and prioritises the simple usability, robustness and more intuitiveness that would cause an impact to facilitate the core functionalities of the network intrusion detection scanning system.



Model Pred	Destination Port	Bwd Packets	Fwd Packets	Fwd Packet Length Max	Flow IAT Mean	Flow IAT Max	Flow IAT Max
0	28845	352	146	146	0	0	0
1	53	315	394	76	0	0	0

[Information from any row displayed here on click]

Figure 8: The GUI design that displays the network behaviour on a table and list of details on a text box.

When the user select the 'use file' button after selecting a CSV file with the necessary order of rows and information, it will take them to another window where it will display the flow of the network data traffic onto a table, which is organised as a list of all data traffic, so it will display the necessary information such as the model prediction, which is used through standardisation through 0 (malicious behaviour) and 1 (benign behaviour), the destination port, in which indicates the port that the packets have come from, as well as 'Bwd Packets', 'Fwd Packets', and 'Fwd Packet Length Max', which represents the number of backward and forward packets that have been present, and the maximum length of forward packets. Moreover, the 'Flow IAT Mean', 'Flow IAT Max' and 'Flow IAT Min' indicats the Inter-Arrival Time statistics, that measure the time between the packet arrivals.

The additional columns not shown on design include 'Fwd Header Length' and 'Bwd Header Length', which indicates the total size of packet headers for forward and backward packets, 'Fwd Packets' and 'Bwd Packets', which indicates the rate of forward and backward packets. Moreover, the 'Packet Length Mean', 'Standard' and 'Variances' describe the statistical measures related to the packet length distributions, 'Average Packet Size' is able to represent the mean size of all packets within the flow of the behaviour, 'Avg Bwd Segment Size' is able to indicate the average size of backward segments within the network flow, 'Subflow Fwd Bytes' and 'Subflow Bwd Bytes' indicates the total bytes in the forward and backward network subflows, and finally, 'Init Win bytes forward' and 'Init Win bytes backward' indicates the initial window size in bytes for forward and backward network flows.

Furthermore, a text box has been provided at the bottom of the table and when the user selects the row within the list on the table, it will display all the information from that row onto the text box to give users a proper insight of the network traffic details, for instance, if row 1 is selected, then it would display the major details from that row onto the text box.

Finally, a simple 'Back' button has been included onto the design so that it would simply be able to let the user go back to the main menu if they would want to either perform another live scan or if they want to choose another CSV file with network information that they would want to be able to detect the attack on.

3.4 Implementation

3.4.1 Software Requirements

As discussed in the main aims and objectives, the Python programming language would be utilised due to the benefit of using a variety of libraries, especially through the use of Machine Learning. In addition, the main Python libraries that would benefit the implementation of the network intrusion detection scanning system include:

1. **NumPy** - a mathematical library that is used for linear algebra elements such as arrays and matrices, therefore, it would be utilised for the classifiers to classify the model. Furthermore, it would be effective for processing a large dataset, which would also allow decision-making within the system.

2. **Pandas** - would be utilised for processing the dataset through the use of a CSV file and obtain the necessary features through the use of feature engineering, so that they are integrated onto the interface, and enhance the ability to respond to alerts.

3. **scikit-learn** - the main Python library for managing the various Machine learning classification algorithms, as well as the main ML metrics like accuracy, precision and recall that determines the efficiency of the algorithms for the main dataset, using the confusion matrix and it would also help with the main methodology. It also provides tools that is useful for training models which would be integrated on the GUI.

4. **os** - This library would help ensure that the user is able to access the CSV file and see if there is any malicious or benign activity. By inserting the CSV file onto the system, the information would be displayed and the behaviour will depend on the model prediction.

5. **sys** - This library may be used for accepting command-line arguments for configuration, as well as managing the system resources like memory, for instance, the user can use the live detection option, the user can be able to create a CSV file from the network packets and use it to be able to review the file for potential malicious activity.

6. **Tkinter** - This Python library will be implemented for the GUI scanning system for a user-friendly and intuitive interface that would enable the user to scan for anomalies as well.

7. **matplotlib** - This is used for plotting graphs to determine the number of attack and benign instances.

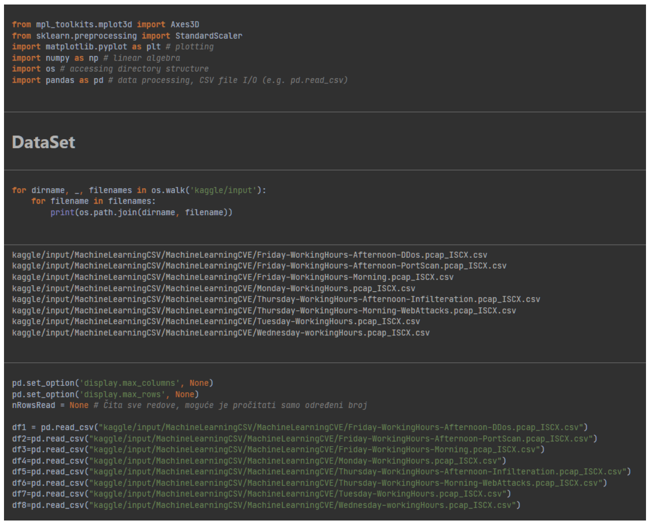
8. **Scapy and SciPy** - Scapy and SciPy are both libraries used for CICFlowmeter because these libraries provide essential capabilities like capturing and analysing the traffic data, as well as extracting the relevant flow features and performing an analysis to detect possible intrusions.

These libraries are used for the methodology of data preprocessing, feature engineering and the model evaluation, as the data comes in a raw or semi-structured format, and is used to pre-process and cleans the data for suitable analysis and creating new features. Also, Tkinter would help provide a intuitive GUI that would allow all the features to be integrated and display the information in a window.

Furthermore, the main software used to implement the system is via PyCharm, as this is a very effective integrated development environment that specialises in Python, and it is able to facilitate the extensive libraries that would benefit from the use of machine learning and implementing a GUI.

3.4.2 Explanation of Code - Model analysis.ipynb

The ML code **model analysis.ipynb** has been found in Kaggle by Jelenanikolicelfak (2020) and it has been integrated within the GUI so that the model prediction can be able to help identify on whether the activity is benign or malicious.



```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

DataSet

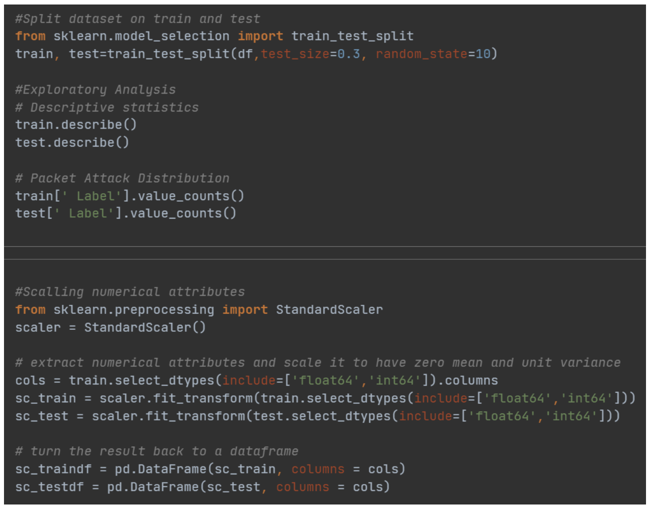
```
for dirname, _, filenames in os.walk('kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
kaggle/input/MachineLearningCSV/MachineLearningCVE/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Friday-WorkingHours-Morning.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Monday-WorkingHours.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Tuesday-WorkingHours.pcap_ISCX.csv
kaggle/input/MachineLearningCSV/MachineLearningCVE/Wednesday-workingHours.pcap_ISCX.csv
```

```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
nRowsRead = None # Čita sve redove, moguće je pročitati samo određeni broj

df1 = pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv")
df2=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv")
df3=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Friday-WorkingHours-Morning.pcap_ISCX.csv")
df4=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Monday-WorkingHours.pcap_ISCX.csv")
df5=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv")
df6=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")
df7=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Tuesday-WorkingHours.pcap_ISCX.csv")
df8=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/Wednesday-workingHours.pcap_ISCX.csv")
```

Figure 9: The imported libraries and reading the CICIDS2017 dataset

This part of the code is merging the dataset into one frame by using Pandas to read the CSV files and concatenate them, but it was very crucial that the libraries are imported in order for them to work.

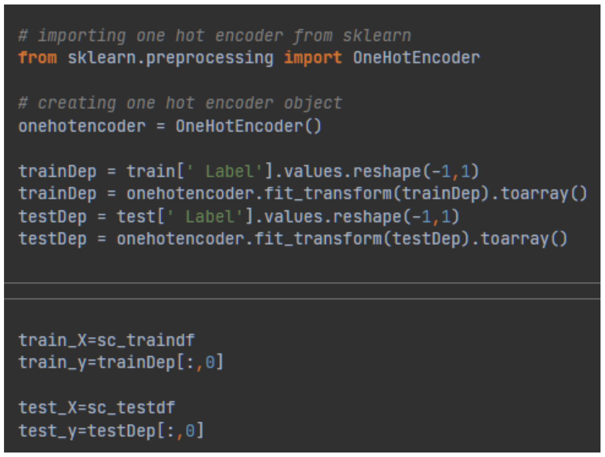


```
#Split dataset on train and test
from sklearn.model_selection import train_test_split
train, test=train_test_split(df,test_size=0.3, random_state=10)

#Exploratory Analysis
# Descriptive statistics
train.describe()
test.describe()

# Packet Attack Distribution
train[' Label'].value_counts()
test[' Label'].value_counts()
```

```
#Scalling numerical attributes
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance
cols = train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(test.select_dtypes(include=['float64','int64']))

# turn the result back to a dataframe
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)
```

Figure 10: Preprocessing the CICIDS2017 data.

This part of the code is used to prepare and analyse the dataset by splitting it into a training and testing set, and the test size means that 30% is used for testing set, whereas the remaining 70% is used for training set. Following that, a standard scaler instance is created to fit it onto the training data and normalise it, as well as transform both training and testing data. Moreover, all of the columns would be taken which contains float or integer numbers, and would only scale these.

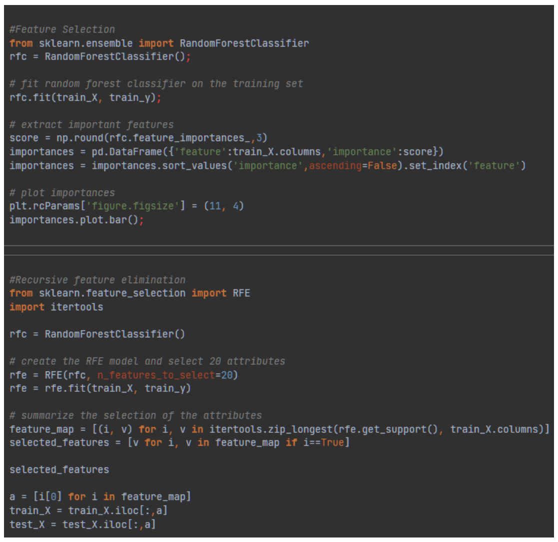


```
# importing one hot encoder from sklearn
from sklearn.preprocessing import OneHotEncoder

# creating one hot encoder object
onehotencoder = OneHotEncoder()

trainDep = train[' Label'].values.reshape(-1,1)
trainDep = onehotencoder.fit_transform(trainDep).toarray()
testDep = test[' Label'].values.reshape(-1,1)
testDep = onehotencoder.fit_transform(testDep).toarray()
```

```
train_X=sc_traindf
train_y=trainDep[:,0]

test_X=sc_testdf
test_y=testDep[:,0]
```

Figure 11: Preprocessing the CICIDS2017 data.

The one hot encoder object encodes the categorical feature (Label) and each unique category uses 0 or 1 to determine whether the label is malicious or benign and the result may depend on sparse output.

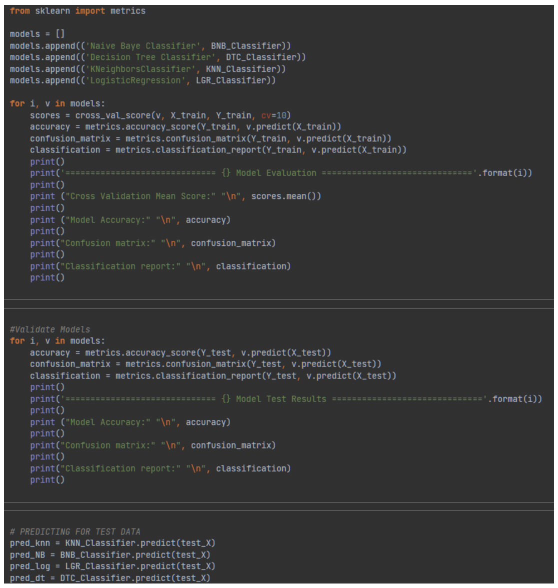


```python
#Feature Selection
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(train_X, train_y);

# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':train_X.columns,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index('feature')

# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();
```

```python
#Recursive feature elimination
from sklearn.feature_selection import RFE
import itertools

rfc = RandomForestClassifier()

# create the RFE model and select 20 attributes
rfe = RFE(rfc, n_features_to_select=20)
rfe = rfe.fit(train_X, train_y)

# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), train_X.columns)]
selected_features = [v for i, v in feature_map if i==True]

selected_features

a = [i[0] for i in feature_map]
train_X = train_X.iloc[:,a]
test_X = test_X.iloc[:,a]
```

Figure 12: Feature Selection

This part of the code has been used for feature selection and it uses the random forest classifier onto the training set and extracts the important features that would be used, and any redundant features would not be utilised and cause it to improve the accuracy, in which it was achieved by converting the columns in standardised form. Afterwards, it would then summarise the selection of the attributes and keeping those columns in selected features, for instance, 'Destination Port', 'Total Length of Fwd Packets', 'Bwd Packet Length Mean', and so on.



```python
from sklearn import metrics

models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))

for i, v in models:
    scores = cross_val_score(v, X_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} Model Evaluation =============================='.format(i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

```python
#Validate Models
for i, v in models:
    accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
    confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
    classification = metrics.classification_report(Y_test, v.predict(X_test))
    print()
    print('============================== {} Model Test Results =============================='.format(i))
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

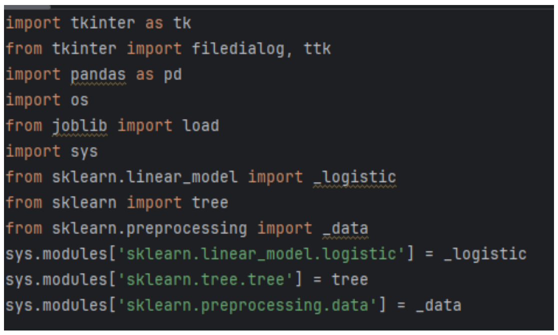
```python
# PREDICTING FOR TEST DATA
pred_knn = KNN_Classifier.predict(test_X)
pred_NB = BNB_Classifier.predict(test_X)
pred_log = LGR_Classifier.predict(test_X)
pred_dt = DTC_Classifier.predict(test_X)
```

Figure 13: Model training with classifiers

The metrics in the sklearn library have been imported so that it can evaluate and validate the model to view the the accuracy, as well as the confusion matrix and classification within each classifier of the algorithms. This will indicate which algorithm has received a very strong result and that classifier is then used for the main GUI.

3.4.3 Explanation of Code - temp.py

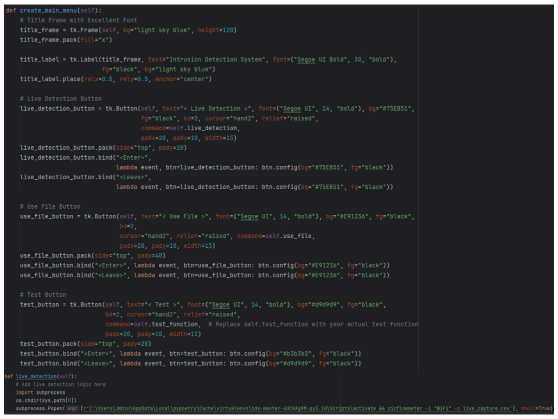
The **temp.py** file in the 'detection-app' directory is where the main GUI is meant to be run from, and it would display the two buttons that have been shown on the main design.



```python
import tkinter as tk
from tkinter import filedialog, ttk
import pandas as pd
import os
from joblib import load
import sys
from sklearn.linear_model import _logistic
from sklearn import tree
from sklearn.preprocessing import _data
sys.modules['sklearn.linear_model.logistic'] = _logistic
sys.modules['sklearn.tree.tree'] = tree
sys.modules['sklearn.preprocessing.data'] = _data
```

Figure 14: Required libraries for the GUI to run

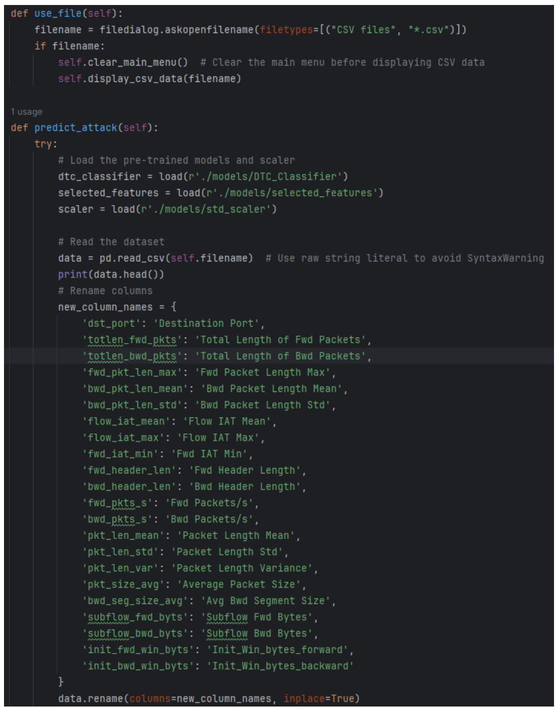
The Tkinter, os and sys libraries are used for the creation for the GUI, with the latter two libraries used to allow users to choose the CSV file from a selected directory.



```python
def create_main_menu(self):
    # Title Frame with Excellent Font
    title_frame = tk.Frame(self, bg="light sky blue", height=120)
    title_frame.pack(fill="x")

    title_label = tk.Label(title_frame, text="Intrusion Detection System", font=("Segoe UI Bold", 30, "bold"),
                           fg="black", bg="light sky blue")
    title_label.place(relx=0.5, rely=0.5, anchor="center")

    # Live Detection Button
    live_detection_button = tk.Button(self, text="< Live Detection >", font=("Segoe UI", 14, "bold"), bg="#75EB51",
                                      fg="black", bd=2, cursor="hand2", relief="raised",
                                      command=self.live_detection,
                                      padx=20, pady=10, width=15)
    live_detection_button.pack(side="top", pady=20)
    live_detection_button.bind("<Enter>",
                               lambda event, btn=live_detection_button: btn.config(bg="#75EB51", fg="black"))
    live_detection_button.bind("<Leave>",
                               lambda event, btn=live_detection_button: btn.config(bg="#75EB51", fg="black"))

    # Use File Button
    use_file_button = tk.Button(self, text="< Use File >", font=("Segoe UI", 14, "bold"), bg="#E91236", fg="black",
                                bd=2,
                                cursor="hand2", relief="raised", command=self.use_file,
                                padx=20, pady=10, width=15)
    use_file_button.pack(side="top", pady=40)
    use_file_button.bind("<Enter>", lambda event, btn=use_file_button: btn.config(bg="#E91236", fg="black"))
    use_file_button.bind("<Leave>", lambda event, btn=use_file_button: btn.config(bg="#E91236", fg="black"))

    # Test Button
    test_button = tk.Button(self, text="< Test >", font=("Segoe UI", 14, "bold"), bg="#d9d9d9", fg="black",
                            bd=2, cursor="hand2", relief="raised",
                            command=self.test_function,  # Replace self.test_function with your actual test function
                            padx=20, pady=10, width=15)
    test_button.pack(side="top", pady=20)
    test_button.bind("<Enter>", lambda event, btn=test_button: btn.config(bg="#b3b3b3", fg="black"))
    test_button.bind("<Leave>", lambda event, btn=test_button: btn.config(bg="#d9d9d9", fg="black"))
def live_detection(self):
   # Add live detection logic here
   import subprocess
   os.chdir(sys.path[0])
   subprocess.Popen( args: [r'C:\Users\Jmbin\AppData\Local\pypoetry\Cache\virtualenvs\ids-master-4UCKAg9M-py3.10\Scripts\activate && cicflowmeter -i "WiFi" -c live_capture.csv'], shell=True)
```

Figure 15: The main code used to operate the GUI.

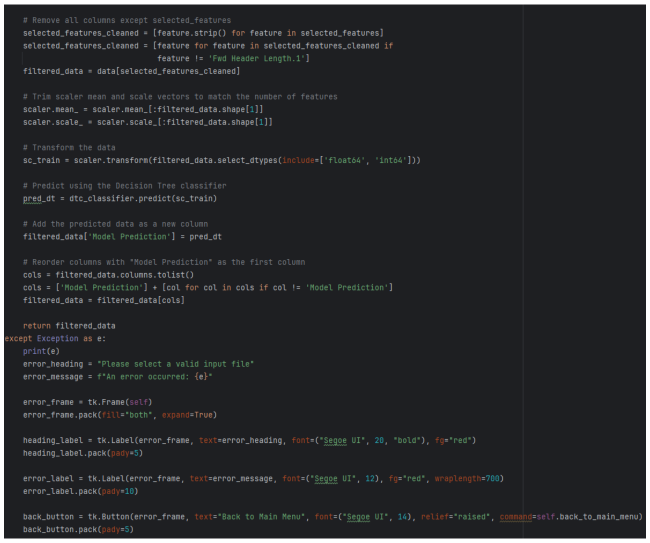
This code creates the GUI application for the intrusion detection system, and two instance variables are initialised; ’self.tree’ and ’self.scrollbar’, with the method ’create main menu’ setting up the menu components, along with the two buttons: ”Live Detection” and ”Use File”, so therefore it would help to ensure that the features are used as expected. In addition, the ’live detection’ function would allow the CSV file to be created whilst the behaviour is displayed throughout the entire page to view at any time. Furthermore, a ’test button’ method was used for testing the functionality.



```python
def use_file(self):
    filename = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if filename:
        self.clear_main_menu()  # Clear the main menu before displaying CSV data
        self.display_csv_data(filename)

1 usage
def predict_attack(self):
    try:
        # Load the pre-trained models and scaler
        dtc_classifier = load(r'./models/DTC_Classifier')
        selected_features = load(r'./models/selected_features')
        scaler = load(r'./models/std_scaler')

        # Read the dataset
        data = pd.read_csv(self.filename)  # Use raw string literal to avoid SyntaxWarning
        print(data.head())
        # Rename columns
        new_column_names = {
            'dst_port': 'Destination Port',
            'totlen_fwd_pkts': 'Total Length of Fwd Packets',
            'totlen_bwd_pkts': 'Total Length of Bwd Packets',
            'fwd_pkt_len_max': 'Fwd Packet Length Max',
            'bwd_pkt_len_mean': 'Bwd Packet Length Mean',
            'bwd_pkt_len_std': 'Bwd Packet Length Std',
            'flow_iat_mean': 'Flow IAT Mean',
            'flow_iat_max': 'Flow IAT Max',
            'fwd_iat_min': 'Fwd IAT Min',
            'fwd_header_len': 'Fwd Header Length',
            'bwd_header_len': 'Bwd Header Length',
            'fwd_pkts_s': 'Fwd Packets/s',
            'bwd_pkts_s': 'Bwd Packets/s',
            'pkt_len_mean': 'Packet Length Mean',
            'pkt_len_std': 'Packet Length Std',
            'pkt_len_var': 'Packet Length Variance',
            'pkt_size_avg': 'Average Packet Size',
            'bwd_seg_size_avg': 'Avg Bwd Segment Size',
            'subflow_fwd_byts': 'Subflow Fwd Bytes',
            'subflow_bwd_byts': 'Subflow Bwd Bytes',
            'init_fwd_win_byts': 'Init_Win_bytes_forward',
            'init_bwd_win_byts': 'Init_Win_bytes_backward'
        }
        data.rename(columns=new_column_names, inplace=True)
```

Figure 16: The main code used to run CSV files on the GUI.

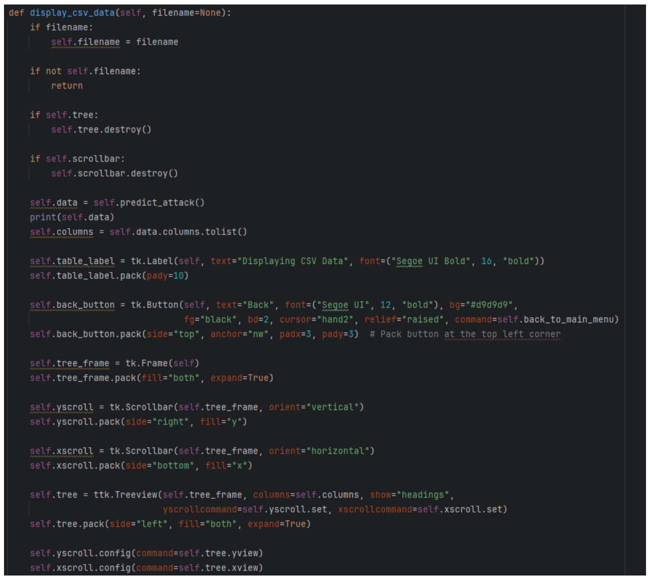
Using the os library, it would enable users to choose a CSV file based on the column names within the file, and after the user has chosen that file, they would be able to view the packet lengths, as well as what the destination ports on where it originated, as well as the model prediction displaying the normalisation values from 0 to 1 that is able to indicate the behaviour of the data.



```python
    # Remove all columns except selected_features
    selected_features_cleaned = [feature.strip() for feature in selected_features]
    selected_features_cleaned = [feature for feature in selected_features_cleaned if
                                 feature != 'Fwd Header Length.1']
    filtered_data = data[selected_features_cleaned]

    # Trim scaler mean and scale vectors to match the number of features
    scaler.mean_ = scaler.mean_[:filtered_data.shape[1]]
    scaler.scale_ = scaler.scale_[:filtered_data.shape[1]]

    # Transform the data
    sc_train = scaler.transform(filtered_data.select_dtypes(include=['float64', 'int64']))

    # Predict using the Decision Tree classifier
    pred_dt = dtc_classifier.predict(sc_train)

    # Add the predicted data as a new column
    filtered_data['Model Prediction'] = pred_dt

    # Reorder columns with "Model Prediction" as the first column
    cols = filtered_data.columns.tolist()
    cols = ['Model Prediction'] + [col for col in cols if col != 'Model Prediction']
    filtered_data = filtered_data[cols]

    return filtered_data
except Exception as e:
    print(e)
    error_heading = "Please select a valid input file"
    error_message = f"An error occurred: {e}"

    error_frame = tk.Frame(self)
    error_frame.pack(fill="both", expand=True)

    heading_label = tk.Label(error_frame, text=error_heading, font=("Segoe UI", 20, "bold"), fg="red")
    heading_label.pack(pady=5)

    error_label = tk.Label(error_frame, text=error_message, font=("Segoe UI", 12), fg="red", wraplength=700)
    error_label.pack(pady=10)

    back_button = tk.Button(error_frame, text="Back to Main Menu", font=("Segoe UI", 14), relief="raised", command=self.back_to_main_menu)
    back_button.pack(pady=5)
```

Figure 17: The code aspect where unnecessary columns are removed and error handling.

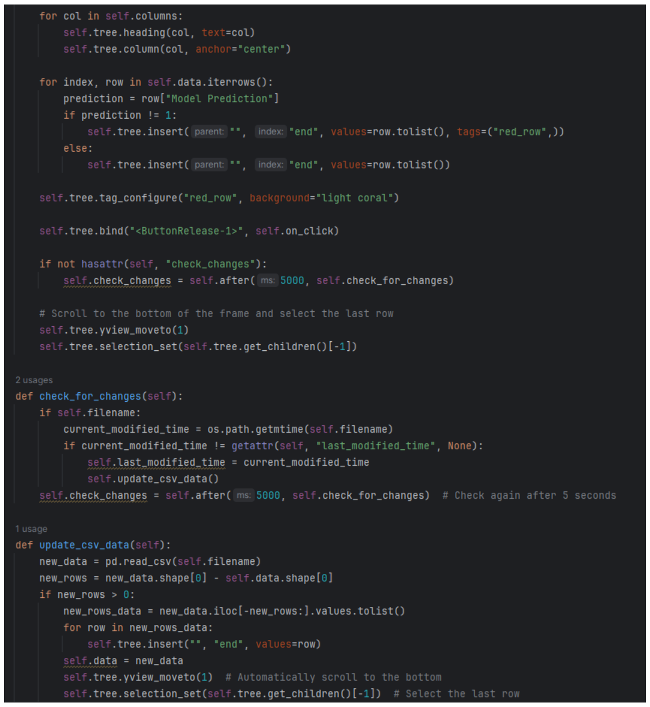
This part of the code uses error handling, as well as removing the irrelevant feature columns. For instance, a valid input file must be selected with the necessary information, otherwise, it would display an error message and prompt the user to go back to the menu to try again. In addition to this, the model prediction is reordered as the first column to ensure the standardisation values is seen as proof that there is a malicious attempt detected.



```python
def display_csv_data(self, filename=None):
    if filename:
        self.filename = filename

    if not self.filename:
        return

    if self.tree:
        self.tree.destroy()

    if self.scrollbar:
        self.scrollbar.destroy()

    self.data = self.predict_attack()
    print(self.data)
    self.columns = self.data.columns.tolist()

    self.table_label = tk.Label(self, text="Displaying CSV Data", font=("Segoe UI Bold", 16, "bold"))
    self.table_label.pack(pady=10)

    self.back_button = tk.Button(self, text="Back", font=("Segoe UI", 12, "bold"), bg="#d9d9d9",
                                 fg="black", bd=2, cursor="hand2", relief="raised", command=self.back_to_main_menu)
    self.back_button.pack(side="top", anchor="nw", padx=3, pady=3)  # Pack button at the top left corner

    self.tree_frame = tk.Frame(self)
    self.tree_frame.pack(fill="both", expand=True)

    self.yscroll = tk.Scrollbar(self.tree_frame, orient="vertical")
    self.yscroll.pack(side="right", fill="y")

    self.xscroll = tk.Scrollbar(self.tree_frame, orient="horizontal")
    self.xscroll.pack(side="bottom", fill="x")

    self.tree = ttk.Treeview(self.tree_frame, columns=self.columns, show="headings",
                             yscrollcommand=self.yscroll.set, xscrollcommand=self.xscroll.set)
    self.tree.pack(side="left", fill="both", expand=True)

    self.yscroll.config(command=self.tree.yview)
    self.xscroll.config(command=self.tree.xview)
```

Figure 18: This displays the CSV data when the user clicks that row.

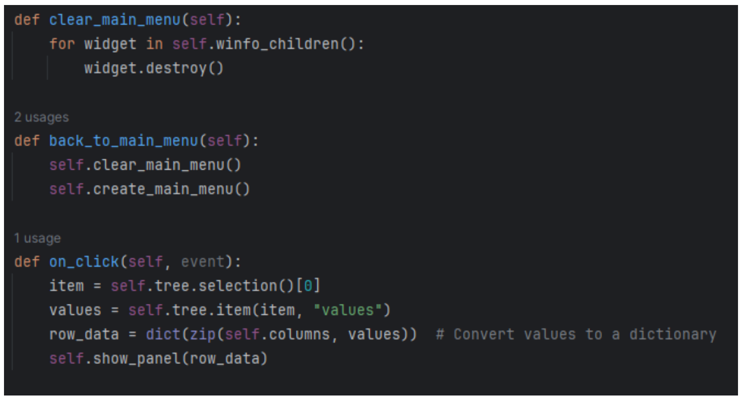
The 'display CSV data' method within the Tkinter class initialises the interface for displaying the CSV data. The filename input is handled by setting the 'self.filename' if that file is provided, otherwise, it would exit. Afterwards, the method retrieves and processes the data using the 'predict attack' method, which is stored in 'self.data' and the column names are extracted and stored in 'self.columns'.



```
    for col in self.columns:
        self.tree.heading(col, text=col)
        self.tree.column(col, anchor="center")

    for index, row in self.data.iterrows():
        prediction = row["Model Prediction"]
        if prediction != 1:
            self.tree.insert( parent: "",  index: "end", values=row.tolist(), tags=("red_row",))
        else:
            self.tree.insert( parent: "",  index: "end", values=row.tolist())

    self.tree.tag_configure("red_row", background="light coral")

    self.tree.bind("<ButtonRelease-1>", self.on_click)

    if not hasattr(self, "check_changes"):
        self.check_changes = self.after( ms: 5000, self.check_for_changes)

    # Scroll to the bottom of the frame and select the last row
    self.tree.yview_moveto(1)
    self.tree.selection_set(self.tree.get_children()[-1])

2 usages
def check_for_changes(self):
    if self.filename:
        current_modified_time = os.path.getmtime(self.filename)
        if current_modified_time != getattr(self, "last_modified_time", None):
            self.last_modified_time = current_modified_time
            self.update_csv_data()
    self.check_changes = self.after( ms: 5000, self.check_for_changes)  # Check again after 5 seconds

1 usage
def update_csv_data(self):
    new_data = pd.read_csv(self.filename)
    new_rows = new_data.shape[0] - self.data.shape[0]
    if new_rows > 0:
        new_rows_data = new_data.iloc[-new_rows:].values.tolist()
        for row in new_rows_data:
            self.tree.insert("", "end", values=row)
        self.data = new_data
        self.tree.yview_moveto(1)  # Automatically scroll to the bottom
        self.tree.selection_set(self.tree.get_children()[-1])  # Select the last row
```

Figure 19: These methods are used when the CSV data is updated or if changes were made

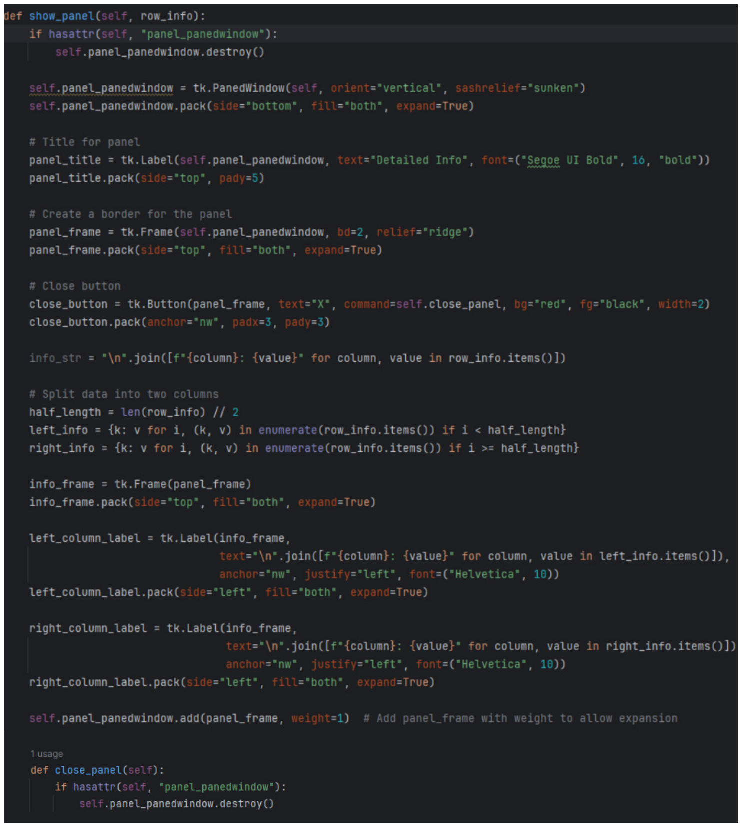
Furthermore, the method then iterates over data columns and configures the headings and columns accordingly, and then for each row, values are inserted into Treeview. Moreover, if a condition based on 'Model Prediction' column is met, then the row would be tagged red, indicating a intrusion attempt had been made. And the 'check for changes' method is used every 5 seconds to check for changes and displays updates to the data.



```
def clear_main_menu(self):
    for widget in self.winfo_children():
        widget.destroy()

2 usages
def back_to_main_menu(self):
    self.clear_main_menu()
    self.create_main_menu()

1 usage
def on_click(self, event):
    item = self.tree.selection()[0]
    values = self.tree.item(item, "values")
    row_data = dict(zip(self.columns, values))  # Convert values to a dictionary
    self.show_panel(row_data)
```

Figure 20: These methods are only used to go back to menu and what happens on clicking a row

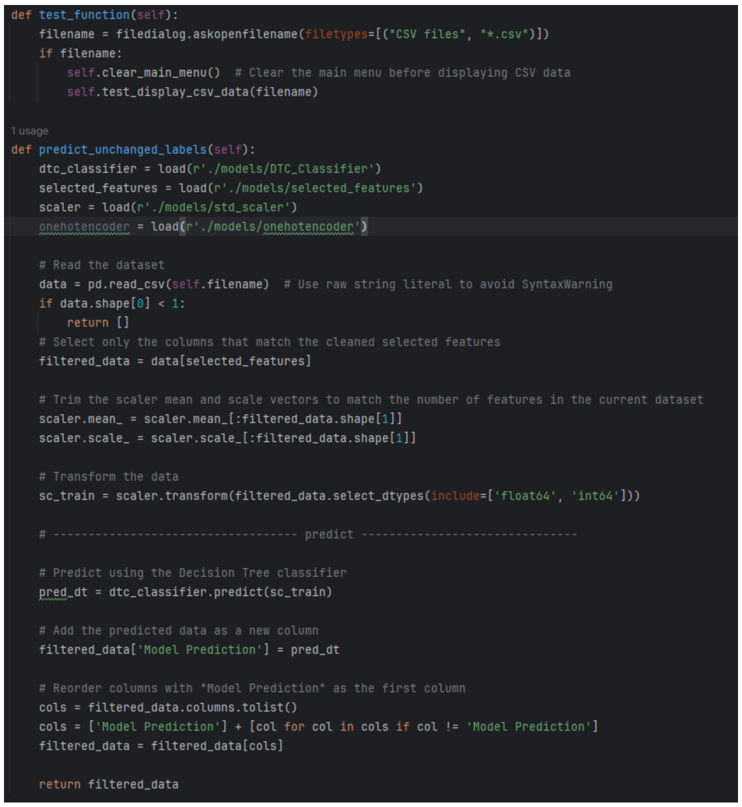
The 'clear main menu' method is used after the user has selected the CSV file onto the main interface, with the 'back to main menu' method being used to allow the user to go back to the menu of the interface, usually when they have finished viewing the CSV file. In addition, the 'on click' method will display the list of the detailed information and convert the values to a dictionary, as well as put the list onto two columns.



```python
def show_panel(self, row_info):
    if hasattr(self, "panel_panedwindow"):
        self.panel_panedwindow.destroy()

    self.panel_panedwindow = tk.PanedWindow(self, orient="vertical", sashrelief="sunken")
    self.panel_panedwindow.pack(side="bottom", fill="both", expand=True)

    # Title for panel
    panel_title = tk.Label(self.panel_panedwindow, text="Detailed Info", font=("Segoe UI Bold", 16, "bold"))
    panel_title.pack(side="top", pady=5)

    # Create a border for the panel
    panel_frame = tk.Frame(self.panel_panedwindow, bd=2, relief="ridge")
    panel_frame.pack(side="top", fill="both", expand=True)

    # Close button
    close_button = tk.Button(panel_frame, text="X", command=self.close_panel, bg="red", fg="black", width=2)
    close_button.pack(anchor="nw", padx=3, pady=3)

    info_str = "\n".join([f"{column}: {value}" for column, value in row_info.items()])

    # Split data into two columns
    half_length = len(row_info) // 2
    left_info = {k: v for i, (k, v) in enumerate(row_info.items()) if i < half_length}
    right_info = {k: v for i, (k, v) in enumerate(row_info.items()) if i >= half_length}

    info_frame = tk.Frame(panel_frame)
    info_frame.pack(side="top", fill="both", expand=True)

    left_column_label = tk.Label(info_frame,
                                 text="\n".join([f"{column}: {value}" for column, value in left_info.items()]),
                                 anchor="nw", justify="left", font=("Helvetica", 10))
    left_column_label.pack(side="left", fill="both", expand=True)

    right_column_label = tk.Label(info_frame,
                                  text="\n".join([f"{column}: {value}" for column, value in right_info.items()])
                                  anchor="nw", justify="left", font=("Helvetica", 10))
    right_column_label.pack(side="left", fill="both", expand=True)

    self.panel_panedwindow.add(panel_frame, weight=1)  # Add panel_frame with weight to allow expansion

    1 usage
    def close_panel(self):
        if hasattr(self, "panel_panedwindow"):
            self.panel_panedwindow.destroy()
```

Figure 21: These methods are only used for showing and closing a panel

The show panel method displays the detailed information about a selected row of data within a panel, and if a panel already exists, it will be destroyed and replaced with new information from another row, and the user has the option to close the panel with a button on top-left of the panel.



```python
def test_function(self):
    filename = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if filename:
        self.clear_main_menu()  # Clear the main menu before displaying CSV data
        self.test_display_csv_data(filename)

1 usage
def predict_unchanged_labels(self):
    dtc_classifier = load(r'./models/DTC_Classifier')
    selected_features = load(r'./models/selected_features')
    scaler = load(r'./models/std_scaler')
    onehotencoder = load(r'./models/onehotencoder')

    # Read the dataset
    data = pd.read_csv(self.filename)  # Use raw string literal to avoid SyntaxWarning
    if data.shape[0] < 1:
        return []
    # Select only the columns that match the cleaned selected features
    filtered_data = data[selected_features]

    # Trim the scaler mean and scale vectors to match the number of features in the current dataset
    scaler.mean_ = scaler.mean_[:filtered_data.shape[1]]
    scaler.scale_ = scaler.scale_[:filtered_data.shape[1]]

    # Transform the data
    sc_train = scaler.transform(filtered_data.select_dtypes(include=['float64', 'int64']))

    # ---------------------------------- predict ------------------------------

    # Predict using the Decision Tree classifier
    pred_dt = dtc_classifier.predict(sc_train)

    # Add the predicted data as a new column
    filtered_data['Model Prediction'] = pred_dt

    # Reorder columns with "Model Prediction" as the first column
    cols = filtered_data.columns.tolist()
    cols = ['Model Prediction'] + [col for col in cols if col != 'Model Prediction']
    filtered_data = filtered_data[cols]

    return filtered_data
```

Figure 22: These methods are used for testing display for CSV data and predicting unchanged labels

The test function method uses the same function for using the CSV files to ensure that it meets the expected standards. Meanwhile, the 'predict unchanged labels' method uses the models and selects the columns that match the correctly selected features, as well as trimming the scaler mean and vectors and then transforming the data in the scaler into float and integer numbers. Afterwards, the Decision Tree classifier is used for predicting the data as a new column.

3.5 Testing

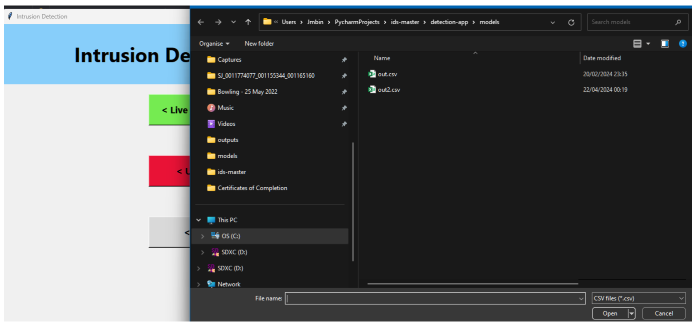
Following the implementation of the system, a Test Plan had been made to ensure the usability and robustness is feasible to meet the necessary standards based on the requirements for the project. The testing process has been carried out through the method of White Box Testing, in which the functionality is tested to prove the validity of the application. It had served as an organisational structure for establishing the acceptance criteria, test scenarios and cases throughout the testing processes.

3.5.1 Test Plan and Test cases

The Test Plan specified multiple testing stages, each intended to confirm different aspects of the system's efficiency and functionality, particularly system testing. By diligently carrying out the Test Plan, potential challenges had been identified, evaluated and addressed quickly, ensuring the delivery of a reliable and top-notch system that corresponds with the project's objectives and expectations.

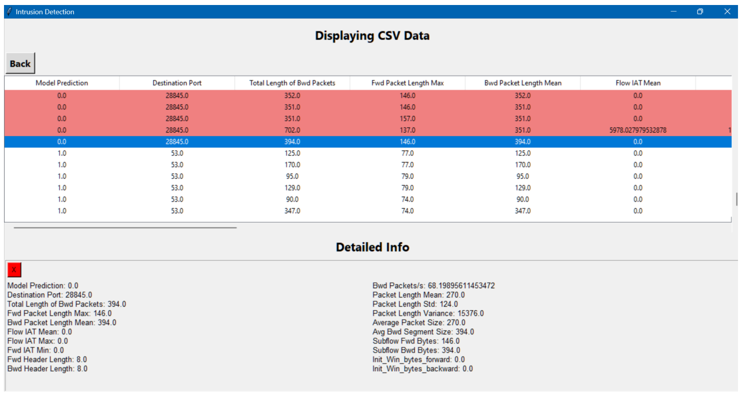
Test	Test Description	Expected Result	Result	Status
1	The user can select a CSV file	The user should be able to select a CSV file	The user is able to select a CSV file **(Figure 23)**	Pass
2	The user can view the network behaviour	The user should be able to view the network behaviour	The user is able to view the network behaviour **(Figure 24)**	Pass
3	The user can view information when row is selected	The user should be able to view the network behaviour	The user is able to view the network behaviour **(Figure 24)**	Pass
4	The user can use live detection with CICFlowmeter to determine the behaviour	The user should be able to use live detection	The user is able to use live detection (**Figure 25**)	Pass

3.5.2 Evidence of the system running



Name	Date modified
out.csv	20/02/2024 23:35
out2.csv	22/04/2024 00:19

Figure 23: Test 1 - The user can choose a CSV file to check the network behaviour


Intrusion Detection

Displaying CSV Data

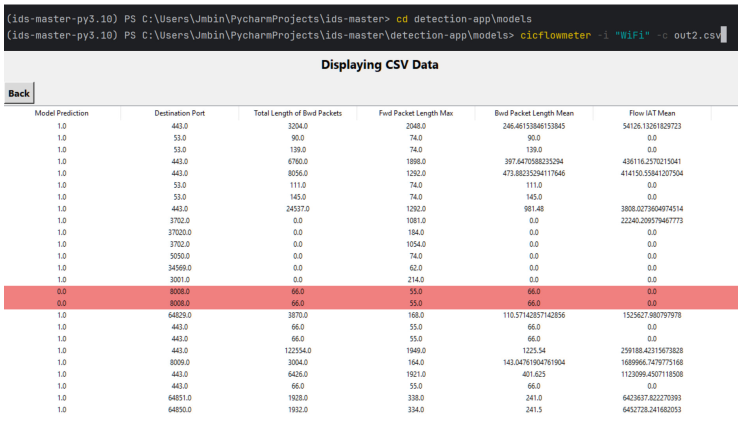
Model Prediction	Destination Port	Total Length of Bwd Packets	Fwd Packet Length Max	Bwd Packet Length Mean	Flow IAT Mean
0.0	28845.0	352.0	146.0	352.0	0.0
0.0	28845.0	351.0	146.0	351.0	0.0
0.0	28845.0	351.0	157.0	351.0	0.0
0.0	28845.0	702.0	137.0	351.0	5978.027979532878
0.0	28845.0	394.0	146.0	394.0	0.0
1.0	53.0	125.0	77.0	125.0	0.0
1.0	53.0	170.0	77.0	170.0	0.0
1.0	53.0	95.0	79.0	95.0	0.0
1.0	53.0	129.0	79.0	129.0	0.0
1.0	53.0	90.0	74.0	90.0	0.0
1.0	53.0	347.0	74.0	347.0	0.0

Detailed Info

Model Prediction: 0.0
Destination Port: 28845.0
Total Length of Bwd Packets: 394.0
Fwd Packet Length Max: 146.0
Bwd Packet Length Mean: 394.0
Flow IAT Mean: 0.0
Flow IAT Max: 0.0
Fwd IAT Min: 0.0
Fwd Header Length: 8.0
Bwd Header Length: 8.0

Bwd Packets/s: 68.19895611453472
Packet Length Mean: 270.0
Packet Length Std: 124.0
Packet Length Variance: 15376.0
Average Packet Size: 270.0
Avg Bwd Segment Size: 394.0
Subflow Fwd Bytes: 146.0
Subflow Bwd Bytes: 394.0
Init_Win_bytes_forward: 0.0
Init_Win_bytes_backward: 0.0

Figure 24: Test 2 and 3 - The user can view network behaviour and view information when row is selected


```
(ids-master-py3.10) PS C:\Users\Jmbin\PycharmProjects\ids-master> cd detection-app\models
(ids-master-py3.10) PS C:\Users\Jmbin\PycharmProjects\ids-master\detection-app\models> cicflowmeter -i "WiFi" -c out2.csv
```

Displaying CSV Data

Model Prediction	Destination Port	Total Length of Bwd Packets	Fwd Packet Length Max	Bwd Packet Length Mean	Flow IAT Mean
1.0	443.0	3204.0	2048.0	246.46153846153845	54126.13261829723
1.0	53.0	90.0	74.0	90.0	0.0
1.0	53.0	139.0	74.0	139.0	0.0
1.0	443.0	6760.0	1898.0	397.6470588235294	436116.2570215041
1.0	443.0	8056.0	1292.0	473.88235294117646	414150.55841207504
1.0	53.0	111.0	74.0	111.0	0.0
1.0	53.0	145.0	74.0	145.0	0.0
1.0	443.0	24537.0	1292.0	981.48	3808.0273604974514
1.0	3702.0	0.0	1081.0	0.0	22240.209579467773
1.0	37020.0	0.0	184.0	0.0	0.0
1.0	3702.0	0.0	1054.0	0.0	0.0
1.0	5050.0	0.0	74.0	0.0	0.0
1.0	34569.0	0.0	62.0	0.0	0.0
1.0	3001.0	0.0	214.0	0.0	0.0
0.0	8008.0	66.0	55.0	66.0	0.0
0.0	8008.0	66.0	55.0	66.0	0.0
1.0	64829.0	3870.0	168.0	110.57142857142856	1525627.980797978
1.0	443.0	66.0	55.0	66.0	0.0
1.0	443.0	66.0	55.0	66.0	0.0
1.0	443.0	122554.0	1949.0	1225.54	259188.42315673828
1.0	8009.0	3004.0	164.0	143.04761904761904	1689966.7479775168
1.0	443.0	6426.0	1921.0	401.625	1123099.4507118508
1.0	443.0	66.0	55.0	66.0	0.0
1.0	64851.0	1928.0	338.0	241.0	6423637.822270393
1.0	64850.0	1932.0	334.0	241.5	6452728.241682053

Figure 25: Test 4 - The user can be able to use live detection with CICFlowmeter

Following the test cases that have been displayed above, it is evident that the necessary features regarding the use of the CSV file and the information inside it has displayed regarding the behavioural patterns detected within the scanned data. For instance, when the user clicks on the 'Use File' button on the menu, a 'select file' would pop-up to allow the user to go to the directory where they have save the scanned file in CSV format, and when the user selects that CSV file, the data is shown based on the float and integer numbers, as well as displaying full information when a row is selected. Furthermore, the user can be able to use the CICFlowmeter tool for live detection and they can use the CSV file format to display the features and standardisation scales for model prediction.

3.6 Evaluation

3.6.1 Performance of Project

The assessment of the performance and results of the GUI scanning system for detecting intrusions in networks suggests the application is able to run fast, and that it is quite memory efficient. Moreover, it does not take up too much memory, and the performance is able to run as smoothly and without any difficulty. However, whilst it is able to run well, it is very important to use the CSV files in order to determine whether the behaviour is indicated as benign or malicious, and this can be achieved through the model prediction.

ML Classifier	**Accuracy**	**Precision**	**Recall**
KNN	0.99	1.00	1.00
Logistic Regression	0.881388	0.89	0.88
Naive Bayes	0.760484	0.77	0.76
Decision Tree	0.99817	1.00	1.00

According to the table, the algorithms with the fastest performance is Decision Tree, as the performance for the accuracy, precision and recall has been received with the highest possible result compared to the rest of the algorithms that have been experimented on. Whilst all of them have received a decent score for each metric, the decision tree has been shown as the algorithm with the most accurate and precise results. The weakest one is Naive Bayes, with 0.76 model accuracy. In addition, based on the results of each classifier/algorithm's accuracy, the Decision Tree classifier had been utilised for the interface, as it has the strongest results compared to the other algorithms, and is able to use the models to see what the network behaviour had been labelled.

3.6.2 Self-evaluation

Throughout the research of this project, it has been proven successful in which CICFlowmeter can be applied to the GUI via live scanning, and that many features have proven to be beneficial. However, throughout the research, there has proven to be many disadvantages that may caused some issues. Throughout the process, the GUI implementation was straightforward with basic logic implemented to give an idea on how the next stages would be carried out, which would also help to integrate the main components such as the classifiers and model prediction. However, the main challenges would mostly be involved with installing CICFlowmeter and whilst it was successful, setting it up has been time consuming. To add more to it, it would have been that better CICFlowmeter can be run through the main interface itself, but despite this, the user can still use CICFlowmeter through terminal, but with the same outcome. Moreover, during the experimentation of the ML algorithms, it was very crucial to be understand which of the machine learning algorithms are necessary for the main process, as it also depends on the model accuracy.

4 Closing chapters

Throughout this entire project, the planning and the implementation had received both benefits as well as challenges that have been experienced. For starters, it was very crucial and essential that research had been planned out thoroughly for the main project, so that ideas can be found and work on it from a certain perspective. The most important part of the project that was very important to note was how machine learning algorithms have been utilised to implement intrusion detection for the network environment.

In addition, the main responsibility whilst working on the project would be to have a comprehensive understanding of the domain of cyber security and network security, as well as the various techniques that have been applied with carefully planned out research to help with the implementation of the NIDS scan system and to ensure the integrity of how the user would utilise the system. Also, it was important to view what the necessary methodology is to provide a crucial understanding on how the system can be implemented.

As established before, the interface is able to display the necessary features within the columns when the CSV file information created from CICFlowmeter is selected. However, the main issue is that it was difficult to allow CICFlowmeter to be used from the 'live detection' button in the interface, but the functionality can be performed through the terminal with a constant file name, and the flows will be added onto the CSV file. This will ensure that the intrusion is predicted on any network CSV file.

4.1 Limitations

When using the 'Use File' option, the user may experience a bug of some CSV files that may not always display the behaviour, and that is because there are some inputs that may contain NaN values, and the classifier does not see those values as valid. Moreover, it would be better that the values are able to match with the columns, and when using the file to display the behaviour, so that they are displayed without any issue. Furthermore, another limitation is that CICFlowmeter was unable to run from the system, but the CICFlowmeter terminal would still work and the CSV file can still be used through the use of the 'Use File' function.

4.2 Future Work

For future research, it can build on these findings by ensuring that many features can be integrated that would enhance the accuracy of machine learning techniques even further. Moreover, to ensure the further use of this, it would be beneficial to consider the nature of the machine learning process, and that overtime, the age of recent datasets would eventually degrade the accuracy, precision and recall and there would be potential new threats that may endanger the security industry. Furthermore, implementing a CICFlowmeter interface can help users to perform a live network scan easily. More importantly, time constraints and thorough research would have to be considered in terms of how much the project can be improved further.

Bibliography

Abdallah, E. E., Otoom, A. F. et al. (2022), 'Intrusion detection systems using supervised machine learning techniques: a survey', *Procedia Computer Science* **201**, 205–212.

Acharya, T., Khatri, I., Annamalai, A. & Chouikha, M. F. (2021), Efficacy of machine learning-based classifiers for binary and multi-class network intrusion detection, *in* '2021 IEEE International Conference on Automatic Control & Intelligent Systems (I2CACIS)', IEEE, pp. 402–407.

Agarwal, A., Sharma, P., Alshehri, M., Mohamed, A. A. & Alfarraj, O. (2021), 'Classification model for accuracy and intrusion detection using machine learning approach', *PeerJ Computer Science* **7**, e437.

Ahmad, I., Ul Haq, Q. E., Imran, M., Alassafi, M. O. & AlGhamdi, R. A. (2022), 'An efficient network intrusion detection and classification system', *Mathematics* **10**(3), 530.

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J. & Ahmad, F. (2021), 'Network intrusion detection system: A systematic study of machine learning and deep learning approaches', *Transactions on Emerging Telecommunications Technologies* **32**(1), e4150.

Al Lail, M., Garcia, A. & Olivo, S. (2023), 'Machine learning for network intrusion detection—a comparative study', *Future Internet* **15**(7), 243.

Alshamy, R. & Ghurab, M. (2020), 'A review of big data in network intrusion detection system: Challenges, approaches, datasets, and tools', *Journal of Computer Sciences and Engineering* **8**(7), 62–74.

Amaizu, G. C., Nwakanma, C. I., Lee, J.-M. & Kim, D.-S. (2020), Investigating network intrusion detection datasets using machine learning, *in* '2020 International Conference on Information and Communication Technology Convergence (ICTC)', IEEE, pp. 1325–1328.

Apruzzese, G., Pajola, L. & Conti, M. (2022), 'The cross-evaluation of machine learning-based network intrusion detection systems', *IEEE Transactions on Network and Service Management* **19**(4), 5152–5169.

Asharf, J., Moustafa, N., Khurshid, H., Debie, E., Haider, W. & Wahab, A. (2020), 'A review of intrusion detection systems using machine and deep learning in internet of things: Challenges, solutions and future directions', *Electronics* **9**(7), 1177.

Bertoli, G. D. C., Júnior, L. A. P., Saotome, O., Dos Santos, A. L., Verri, F. A. N., Marcondes, C. A. C., Barbieri, S., Rodrigues, M. S. & De Oliveira, J. M. P. (2021), 'An end-to-end framework for machine learning-based network intrusion detection system', *IEEE Access* **9**, 106790–106805.

Chaudhari, R. R. & Patil, S. P. (2017), 'Intrusion detection system: classification, techniques and datasets to implement', *International Research Journal of Engineering and Technology (IRJET)* **4**(2), 1860–1866.

Chindove, H. & Brown, D. (2021), Adaptive machine learning based network intrusion detection, *in* 'Proceedings of the International Conference on Artificial Intelligence and Its Applications', icARTi '21, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3487923.3487938*

Chou, D. & Jiang, M. (2021), 'A survey on data-driven network intrusion detection', *ACM Computing Surveys (CSUR)* **54**(9), 1–36.

Chowdhury, R. N., Chowdhury, M. M., Chowdhury, S., Islam, M. R., Ayub, M. A., Chowdhury, A. & Kalpoma, K. A. (2020), Parameter optimization and performance analysis of state-of-the-art machine learning techniques for intrusion detection system (ids), *in* '2020 23rd International Conference on Computer and Information Technology (ICCIT)', IEEE, pp. 1–6.

Das, S., Saha, S., Priyoti, A. T., Roy, E. K., Sheldon, F. T., Haque, A. & Shiva, S. (2021), 'Network intrusion detection and comparative analysis using ensemble machine learning and feature selection', *IEEE transactions on network and service management* **19**(4), 4821–4833.

datthinh1801 (2022), 'Datthinh1801/cicflowmeter: This is a python version of cicflowmeter-v4.0 (formerly known as iscxflowmeter) - an ethernet traffic biflow generator and analyzer for anomaly detection.', `https://github.com/datthinh1801/cicflowmeter`.

Di Mauro, M., Galatro, G., Fortino, G. & Liotta, A. (2021), 'Supervised feature selection techniques in network intrusion detection: A critical review', *Engineering Applications of Artificial Intelligence* **101**, 104216.

Elmasry, W., Akbulut, A. & Zaim, A. H. (2019), 'Empirical study on multiclass classification-based network intrusion detection', *Computational Intelligence* **35**(4), 919–954.

Engelen, G., Rimmer, V. & Joosen, W. (2021), Troubleshooting an intrusion detection dataset: the cicids2017 case study, *in* '2021 IEEE Security and Privacy Workshops (SPW)', IEEE, pp. 7–12.

Ferdiana, R. et al. (2020), A systematic literature review of intrusion detection system for network security: Research trends, datasets and methods, *in* '2020 4th International Conference on Informatics and Computational Sciences (ICICoS)', IEEE, pp. 1–6.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G. & Vázquez, E. (2009), 'Anomaly-based network intrusion detection: Techniques, systems and challenges', *computers & security* **28**(1-2), 18–28.

Ghurab, M., Gaphari, G., Alshami, F., Alshamy, R. & Othman, S. (2021), 'A detailed analysis of benchmark datasets for network intrusion detection system', *Asian Journal of Research in Computer Science* **7**(4), 14–33.

Guezzaz, A., Benkirane, S., Azrour, M. & Khurram, S. (2021), 'A reliable network intrusion detection approach using decision tree with enhanced data quality', *Security and Communication Networks* **2021**, 1–8.

Hajj, S., El Sibai, R., Bou Abdo, J., Demerjian, J., Makhoul, A. & Guyeux, C. (2021), 'Anomaly-based intrusion detection systems: The requirements, methods, measurements, and datasets', *Transactions on Emerging Telecommunications Technologies* **32**(4), e4240.

Ilyas, M. U. & Alharbi, S. A. (2022), 'Machine learning approaches to network intrusion detection for contemporary internet traffic', *Computing* **104**(5), 1061–1076.

Jain, G. et al. (2021), Application of snort and wireshark in network traffic analysis, *in* 'IOP Conference Series: Materials Science and Engineering', Vol. 1119, IOP Publishing, p. 012007.

Jamalipour, A. & Murali, S. (2021), 'A taxonomy of machine-learning-based intrusion detection systems for the internet of things: A survey', *IEEE Internet of Things Journal* **9**(12), 9444–9466.

Jelenanikolicelfak (2020), 'Cicids 2017 data and classifiers analysis', `https://www.kaggle.com/code/jelenanikolicelfak/cicids-2017-data-and-classifiers-analysis`.

Keserwani, P. K., Govil, M. C. & Pilli, E. S. (2021), 'An effective nids framework based on a comprehensive survey of feature optimization and classification techniques', *Neural Computing and Applications* **35**(7), 4993–5013.

Kilincer, I. F., Ertam, F. & Sengur, A. (2021), 'Machine learning methods for cyber security intrusion detection: Datasets and comparative study', *Computer Networks* **188**, 107840.

Kocher, G. & Kumar, G. (2021), 'Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges', *Soft Computing* **25**(15), 9731–9763.

Kumar, G. (2014), 'Evaluation metrics for intrusion detection systems- a study', *Evaluation* **2**(11), 11–7.

Kumar, V., Choudhary, V., Sahrawat, V. & Kumar, V. (2020), Detecting intrusions and attacks in the network traffic using anomaly based techniques, *in* '2020 5th International Conference on Communication and Electronics Systems (ICCES)', IEEE, pp. 554–560.

Lanvin, M., Gimenez, P.-F., Han, Y., Majorczyk, F., Mé, L. & Totel, E. (2022), Errors in the cicids2017 dataset and the significant differences in detection performances it makes, *in* 'International Conference on Risks and Security of Internet and Systems', Springer, pp. 18–33.

Latif, S., Dola, F. F., Afsar, M., Esha, I. J. & Nandi, D. (2022), 'Investigation of machine learning algorithms for network intrusion detection.', *International Journal of Information Engineering & Electronic Business* **14**(2).

Liu, H. & Lang, B. (2019), 'Machine learning and deep learning methods for intrusion detection systems: A survey', *applied sciences* **9**(20), 4396.

Maharaj, N. & Khanna, P. (2014), 'A comparative analysis of different classification techniques for intrusion detection system', *International Journal of Computer Applications* **95**(17).

Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A. & Foozy, C. F. M. (2021), 'Benchmarking of machine learning for anomaly based intrusion detection systems in the cicids2017 dataset', *IEEE access* **9**, 22351–22370.

Mukherjee, B., Heberlein, L. T. & Levitt, K. N. (1994), 'Network intrusion detection', *IEEE network* **8**(3), 26–41.

Musa, U. S., Chhabra, M., Ali, A. & Kaur, M. (2020), Intrusion detection system using machine learning techniques: A review, *in* '2020 international conference on smart electronics and communication (ICOSEC)', IEEE, pp. 149–155.

Oyelakin, A., Ameen, A., Ogundele, T., Salau-Ibrahim, T., Abdulrauf, U., Olufadi, H., Ajiboye, I., Muhammad-Thani, S. et al. (2023), 'Overview and exploratory analyses of cicids 2017 intrusion detection dataset', *Journal of Systems Engineering and Information Technology (JOSEIT)* **2**(2), 45–52.

Panwar, S. S., Raiwani, Y. & Panwar, L. S. (2022), An intrusion detection model for cicids-2017 dataset using machine learning algorithms, *in* '2022 International Conference on Advances in Computing, Communication and Materials (ICACCM)', IEEE, pp. 1–10.

Poetry (2018), 'Poetry - python packaging and dependency management made easy', https://python-poetry.org/.

Repalle, S. A. & Kolluru, V. R. (2017), 'Intrusion detection system using ai and machine learning algorithm', *International Research Journal of Engineering and Technology (IRJET)* **4**(12), 1709–1715.

Sarhan, M., Layeghy, S. & Portmann, M. (2022), 'Towards a standard feature set for network intrusion detection system datasets', *Mobile networks and applications* pp. 1–14.

Shaukat, S., Ali, A., Batool, A., Alqahtani, F., Khan, J. S., Ahmad, J. et al. (2020), Intrusion detection and attack classification leveraging machine learning technique, *in* '2020 14th International Conference on Innovations in Information Technology (IIT)', IEEE, pp. 198–202.

Stiawan, D., Idris, M. Y. B., Bamhdi, A. M., Budiarto, R. et al. (2020), 'Cicids-2017 dataset feature analysis with information gain for anomaly detection', *IEEE Access* **8**, 132911–132921.

Tait, K.-A., Khan, J. S., Alqahtani, F., Shah, A. A., Khan, F. A., Rehman, M. U., Boulila, W. & Ahmad, J. (2021), Intrusion detection using machine learning techniques: an experimental comparison, *in* '2021 International Congress of Advanced Technology and Engineering (ICOTEN)', IEEE, pp. 1–10.

Thakkar, A. & Lohiya, R. (2022), 'A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions', *Artificial Intelligence Review* **55**(1), 453–563.

Toupas, P., Chamou, D., Giannoutakis, K. M., Drosou, A. & Tzovaras, D. (2019), An intrusion detection system for multi-class classification based on deep neural networks, *in* '2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)', pp. 1253–1258.

Umar, M. A., Chen, Z., Shuaib, K. & Liu, Y. (2024), 'Effects of feature selection and normalization on network intrusion detection', *Authorea Preprints* .

Yang, Z., Liu, X., Li, T., Wu, D., Wang, J., Zhao, Y. & Han, H. (2022), 'A systematic literature review of methods and datasets for anomaly-based network intrusion detection', *Computers & Security* **116**, 102675.

Zhang, C., Jia, D., Wang, L., Wang, W., Liu, F. & Yang, A. (2022), 'Comparative research on network intrusion detection methods based on machine learning', *Computers & Security* p. 102861.

5 Appendix

5.1 Instructions on how to run the system with CICFlowmeter

To allow the GUI to run with CICFlowmeter, Poetry, which is used for dependency management and packaging, must be installed to enable a convenient installation of the libraries, and it can be found on Poetry (2018). Furthermore, datthinh1801 (2022) has provided instructions on how to install the analyser, particularly for using the main terminal to run the analyser so that it can capture the data and output the flows in a CSV file format. Also, many libraries such as scipy and scapy would be installed so that CICFlowmeter is able to operate with the GUI as necessary.

After installation, it is important to set the interpreter as 'Poetry', otherwise, go on 'Add Local Interpreter', find that directory and set that environment as the default interpreter. Next, use the terminal and navigate to where cicflowmeter has been installed, and use an appropriate command, e.g. '-i "WiFi" -c [name of CSV file]', and what this would do is that it will initialise a live scan and show the network information based on how many packets are found. After the scan, the CSV file would be complete and the information can be viewed after terminating it within a certain time period.

5.2 Python Code - model analysis.ipynb

5.2.1 Imports and Preprocessing

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

5.2.2 Accessing dataset

```
for dirname, _, filenames in os.walk('kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
nRowsRead = None #  ita  sve redove, mogu e je pro itati samo odre eni broj
(Reads all lines, certain number is read)

df1 = pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv")
df2=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv")
df3=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Friday-WorkingHours-Morning.pcap_ISCX.csv")
df4=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Monday-WorkingHours.pcap_ISCX.csv")
df5=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv")
df6=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")
df7=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Tuesday-WorkingHours.pcap_ISCX.csv")
df8=pd.read_csv("kaggle/input/MachineLearningCSV/MachineLearningCVE/
Wednesday-workingHours.pcap_ISCX.csv")


df = pd.concat([df1,df2])
del df1,df2
df = pd.concat([df,df3])
del df3
df = pd.concat([df,df4])
del df4
df = pd.concat([df,df5])
del df5
df = pd.concat([df,df6])
del df6
df = pd.concat([df,df7])
del df7
df = pd.concat([df,df8])
del df8

nRow, nCol = df.shape
print(f'U tabeli ima {nRow} redova i {nCol} kolona')

# Split dataset on train and test
from sklearn.model_selection import train_test_split
train, test=train_test_split(df,test_size=0.3, random_state=10)

#Exploratory Analysis
# Descriptive statistics
train.describe()
test.describe()

# Packet Attack Distribution
train[' Label'].value_counts()
```

```
test[' Label'].value_counts()
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance
cols = train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(test.select_dtypes(include=['float64','int64']))

# turn the result back to a dataframe
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)

# importing one hot encoder from sklearn
from sklearn.preprocessing import OneHotEncoder

# creating one hot encoder object
onehotencoder = OneHotEncoder()

trainDep = train[' Label'].values.reshape(-1,1)
trainDep = onehotencoder.fit_transform(trainDep).toarray()
testDep = test[' Label'].values.reshape(-1,1)
testDep = onehotencoder.fit_transform(testDep).toarray()

train_X=sc_traindf
train_y=trainDep[:,0]

test_X=sc_testdf
test_y=testDep[:,0]
```

5.2.3 Feature Selection

```
#Feature Selection
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(train_X, train_y);

# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':train_X.columns,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index
('feature')
```

```
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();

#Recursive feature elimination
from sklearn.feature_selection import RFE
import itertools

rfc = RandomForestClassifier()

# create the RFE model and select 20 attributes
rfe = RFE(rfc, n_features_to_select=20)
rfe = rfe.fit(train_X, train_y)

# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(),
train_X.columns)]
selected_features = [v for i, v in feature_map if i==True]

selected_features

a = [i[0] for i in feature_map]
train_X = train_X.iloc[:,a]
test_X = test_X.iloc[:,a]
```

5.2.4 Model Training

```
#Dataset Partition
X_train,X_test,Y_train,Y_test = train_test_split(train_X,train_y,
train_size=0.70, random_state=2)

#Fitting Models
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);

# Train LogisticRegression Model
```

```
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy',
random_state=0)
DTC_Classifier.fit(X_train, Y_train)

#Evaluate Models
from sklearn import metrics

models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))

for i, v in models:
    scores = cross_val_score(v, X_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} Model Evaluation ==============
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()

#Validate Models
for i, v in models:
    accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
    confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
    classification = metrics.classification_report(Y_test, v.predict(X_test))
    print()
    print('============================== {} Model Test Results ============
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()

    # PREDICTING FOR TEST DATA
pred_knn = KNN_Classifier.predict(test_X)
pred_NB = BNB_Classifier.predict(test_X)
pred_log = LGR_Classifier.predict(test_X)
pred_dt = DTC_Classifier.predict(test_X)
```

5.3 Python Code - temp.py (GUI)

5.3.1 Required Imports

```
import tkinter as tk
from tkinter import filedialog, ttk
import pandas as pd
import os
from joblib import load
import sys
from sklearn.linear_model import _logistic
from sklearn import tree
from sklearn.preprocessing import _data
sys.modules['sklearn.linear_model.logistic'] = _logistic
sys.modules['sklearn.tree.tree'] = tree
sys.modules['sklearn.preprocessing.data'] = _data
```

5.3.2 Creating Menu and Use CSV File

```
class IntrusionDetectionApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Intrusion Detection")
        self.geometry("800x600")
        self.tree = None  # Initialize self.tree
        self.scrollbar = None  # Initialize scrollbar
        self.create_main_menu()

    def create_main_menu(self):
        # Title Frame with Excellent Font
        title_frame = tk.Frame(self, bg="light sky blue", height=120)
        title_frame.pack(fill="x")
```

```
title_label = tk.Label(title_frame, text="Intrusion Detection System",
font=("Segoe UI Bold", 30, "bold"),
                        fg="black", bg="light sky blue")
title_label.place(relx=0.5, rely=0.5, anchor="center")

# Live Detection Button
live_detection_button = tk.Button(self,
text="< Live Detection >", font=("Segoe UI", 14, "bold"), bg="#75EB51",
fg="black", bd=2, cursor="hand2",relief="raised", command=self.
live_detection,
padx=20, pady=10, width=15)
live_detection_button.pack(side="top", pady=20)
live_detection_button.bind("<Enter>",
lambda event, btn=live_detection_button: btn.config(bg="#75EB51",
fg="black"))
live_detection_button.bind("<Leave>",
lambda event, btn=live_detection_button: btn.config(bg="#75EB51",
fg="black"))
```

5.3.3 Live Detection, use CSV file and Test Button

```
# Use File Button
use_file_button = tk.Button(self, text="< Use File >", font=("Segoe UI",
14, "bold"), bg="#E91236", fg="black",
bd=2,
cursor="hand2", relief="raised", command=self.use_file,
padx=20, pady=10, width=15)
use_file_button.pack(side="top", pady=40)
use_file_button.bind("<Enter>", lambda event, btn=use_file_button:
btn.config(bg="#E91236", fg="black"))
use_file_button.bind("<Leave>",
lambda event, btn=use_file_button: btn.config(bg="#E91236", fg="black"))

# Test Button
test_button = tk.Button(self, text="< Test >", font=("Segoe UI", 14,
"bold"), bg="#d9d9d9", fg="black",
bd=2, cursor="hand2", relief="raised",
command=self.test_function,
# Replace self.test_function with your actual test function
 padx=20, pady=10, width=15)
test_button.pack(side="top", pady=20)
test_button.bind("<Enter>", lambda event, btn=test_button: btn.config
(bg="#b3b3b3", fg="black"))
test_button.bind("<Leave>", lambda event, btn=test_button: btn.config
        (bg="#d9d9d9", fg="black"))

    def live_detection(self):
        # Add live detection logic here
        import subprocess
        os.chdir(sys.path[0])
        subprocess.Popen([r'C:\Users\Jmbin\AppData\Local\
        pypoetry\Cache\virtualenvs\ids-master-4UCKAg9M-
        py3.10\Scripts\activate && cicflowmeter -i
        "WiFi" -c live_capture.csv'], shell=True)

    def use_file(self):
        filename = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")]
        if filename:
            self.clear_main_menu()  # Clear the main menu before displaying CSV
            data
            self.display_csv_data(filename)

    def predict_attack(self):
        try:
            # Load the pre-trained models and scaler
            dtc_classifier = load(r'./models/DTC_Classifier')
            selected_features = load(r'./models/selected_features')
            scaler = load(r'./models/std_scaler')

            # Read the dataset
            data = pd.read_csv(self.filename)  # Use raw string literal to
            avoid
            SyntaxWarning

            # Rename columns
            new_column_names = {
                'dst_port': 'Destination Port',
                'totlen_fwd_pkts': 'Total Length of Fwd Packets',
                'totlen_bwd_pkts': 'Total Length of Bwd Packets',
                'fwd_pkt_len_max': 'Fwd Packet Length Max',
                'bwd_pkt_len_mean': 'Bwd Packet Length Mean',
                'bwd_pkt_len_std': 'Bwd Packet Length Std',
                'flow_iat_mean': 'Flow IAT Mean',
                'flow_iat_max': 'Flow IAT Max',
                'fwd_iat_min': 'Fwd IAT Min',
                'fwd_header_len': 'Fwd Header Length',
                'bwd_header_len': 'Bwd Header Length',
                'fwd_pkts_s': 'Fwd Packets/s',
                'bwd_pkts_s': 'Bwd Packets/s',
                'pkt_len_mean': 'Packet Length Mean',
                'pkt_len_std': 'Packet Length Std',
                'pkt_len_var': 'Packet Length Variance',
                'pkt_size_avg': 'Average Packet Size',
                'bwd_seg_size_avg': 'Avg Bwd Segment Size',
                'subflow_fwd_byts': 'Subflow Fwd Bytes',
                'subflow_bwd_byts': 'Subflow Bwd Bytes',
                'init_fwd_win_byts': 'Init_Win_bytes_forward',
                'init_bwd_win_byts': 'Init_Win_bytes_backward'
            }
            data.rename(columns=new_column_names, inplace=True)

            # Remove all columns except selected_features
            selected_features_cleaned = [feature.strip() for feature in
            selected_features]
            selected_features_cleaned = [feature for feature in
            selected_features_cleaned if
            feature != 'Fwd Header Length.1']
            filtered_data = data[selected_features_cleaned]

            # Trim scaler mean and scale vectors to match the number of features
            scaler.mean_ = scaler.mean_[:filtered_data.shape[1]]
            scaler.scale_ = scaler.scale_[:filtered_data.shape[1]]

            # Transform the data
            sc_train = scaler.transform(filtered_data.select_dtypes(include=[
            'float64', 'int64']))

            # Predict using the Decision Tree classifier
            pred_dt = dtc_classifier.predict(sc_train)

            # Add the predicted data as a new column
            filtered_data['Model Prediction'] = pred_dt

            # Reorder columns with "Model Prediction" as the first column
            cols = filtered_data.columns.tolist()
            cols = ['Model Prediction'] + [col for col in cols if col !=
            'Model Prediction']
            filtered_data = filtered_data[cols]

            return filtered_data

        except Exception as e:
            error_heading = "Please select a valid input file"
            error_message = f"An error occurred: {e}"

            error_frame = tk.Frame(self)
            error_frame.pack(fill="both", expand=True)

            heading_label = tk.Label(error_frame, text=error_heading, font=("
            Segoe UI", 20, "bold"), fg="red")
            heading_label.pack(pady=5)

            error_label = tk.Label(error_frame, text=error_message, font=
            ("Segoe UI", 12), fg="red", wraplength=700)
            error_label.pack(pady=10)

            back_button = tk.Button(error_frame, text="Back to Main Menu",
            font=("Segoe UI", 14), relief="raised",
            command=self.back_to_main_menu)
            back_button.pack(pady=5)

            # Configure error frame to expand
            vertically to fit the error message
            error_frame.grid_rowconfigure(0, weight=1)

            return False, None  # Return False and None as data
```

5.3.4 Displaying CSV data

```
def display_csv_data(self, filename=None):
    if filename:
        self.filename = filename

    if not self.filename:
        return

    if self.tree:
        self.tree.destroy()

    if self.scrollbar:
        self.scrollbar.destroy()

    # Clear existing data
    self.data = None

    self.data = self.predict_attack()
    self.columns = self.data.columns.tolist()

    self.table_label = tk.Label(self, text="Displaying CSV Data",
    font=("Segoe UI Bold", 16, "bold"))
    self.table_label.pack(pady=10)
```

```
        self.back_button = tk.Button(self, text="Back",
        font=("Segoe UI", 12, "bold"), bg="#d9d9d9",
                                fg="black", bd=2, cursor="hand2",
                                relief="raised", command=
                                self.back_to_main_menu)
        self.back_button.pack(side="top", anchor="nw", padx=3, pady=3)
# Pack button at the top left corner

        self.tree_frame = tk.Frame(self)
        self.tree_frame.pack(fill="both", expand=True)

        self.yscroll = tk.Scrollbar(self.tree_frame, orient="vertical")
        self.yscroll.pack(side="right", fill="y")

        self.xscroll = tk.Scrollbar(self.tree_frame, orient="horizontal")
        self.xscroll.pack(side="bottom", fill="x")

        self.tree = ttk.Treeview(self.tree_frame, columns=self.columns,
        show="headings",
        yscrollcommand=self.yscroll.set, xscrollcommand=self.xscroll.set)
        self.tree.pack(side="left", fill="both", expand=True)

        self.yscroll.config(command=self.tree.yview)
        self.xscroll.config(command=self.tree.xview)

        # Set style to change heading background color
        style = ttk.Style()
        style.map("Treeview.Heading", background=[("active", "lightblue")])

        for col in self.columns:
            self.tree.heading(col, text=col)
            self.tree.column(col, anchor="center")

        for index, row in self.data.iterrows():
            prediction = row["Model Prediction"]
            if prediction != 1:
                self.tree.insert("", "end", values=row.tolist(),
                tags=("red_row",))
            else:
                self.tree.insert("", "end", values=row.tolist())

        self.tree.tag_configure("red_row", background="light coral")

        self.tree.bind("<ButtonRelease-1>", self.on_click)
        # self.bind("<Enter>", lambda event, btn=self.back_button:
        btn.config(bg="#b3b3b3", fg="black"))
        # self.bind("<Leave>", lambda event, btn=self.back_button:
        btn.config(bg="#d9d9d9", fg="black"))

        if not hasattr(self, "check_changes"):
            self.check_changes = self.after(5000, self.check_for_changes)

        # Scroll to the bottom of the frame and select the last row
        self.tree.yview_moveto(1)
        self.tree.selection_set(self.tree.get_children()[-1])
```

5.3.5 Checking and Updating CSV data

```
    def check_for_changes(self):
        if self.filename:
            current_modified_time = os.path.getmtime(self.filename)
            if current_modified_time !=
            getattr(self, "last_modified_time", None):
                self.last_modified_time = current_modified_time
                self.update_csv_data()
        self.check_changes = self.after(5000, self.check_for_changes)
# Check again after 5 seconds

    def update_csv_data(self):
        new_data = self.predict_attack()
        new_rows = new_data.shape[0] - self.data.shape[0]
        if new_rows > 0:
            new_rows_data = new_data.iloc[-new_rows:].values.tolist()
            for row in new_rows_data:
                # self.tree.insert("", "end", values=row)
                prediction = row[0]
                if prediction != 1:
                    self.tree.insert("", "end", values=row, tags=("red_row",))
                else:
                    self.tree.insert("", "end", values=row)

            self.tree.tag_configure("red_row", background="light coral")

            self.data = new_data
            self.tree.yview_moveto(1)  # Automatically scroll to the bottom
            self.tree.selection_set(self.tree.get_children()[-1])
# Select the last row
```

5.3.6 Menu and Clicking on Row to display detailed information

```
def clear_main_menu(self):
    for widget in self.winfo_children():
        widget.destroy()

def back_to_main_menu(self):
    self.clear_main_menu()
    self.create_main_menu()

def on_click(self, event):
    item = self.tree.selection()[0]
    values = self.tree.item(item, "values")
    row_data = dict(zip(self.columns, values))
    # Convert values to a dictionary
    self.show_panel(row_data)
```

5.3.7 Showing and Closing Panel of detailed CSV information

```
def show_panel(self, row_info):
    if hasattr(self, "panel_panedwindow"):
        self.panel_panedwindow.destroy()

    self.panel_panedwindow = tk.PanedWindow(self, orient="vertical",
    sashrelief="sunken")
    self.panel_panedwindow.pack(side="bottom", fill="both", expand=True)

    # Title for panel
    panel_title = tk.Label(self.panel_panedwindow, text="Detailed Info",
    font=("Segoe UI Bold", 16, "bold"))
    panel_title.pack(side="top", pady=5)

    # Create a border for the panel
    panel_frame = tk.Frame(self.panel_panedwindow, bd=2, relief="ridge")
    panel_frame.pack(side="top", fill="both", expand=True)

    # Close button
    close_button = tk.Button(panel_frame, text="X",
    command=self.close_panel, bg="red", fg="black", width=2)
    close_button.pack(anchor="nw", padx=3, pady=3)

    info_str = "\n".join([f"{column}: {value}" for column, value
    in row_info.items()])

    # Split data into two columns
```

```
        half_length = len(row_info) // 2
        left_info = {k: v for i, (k, v) in enumerate(row_info.items()) if i
        < half_length}
        right_info = {k: v for i, (k, v) in enumerate(row_info.items()) if i >=
        half_length}

        info_frame = tk.Frame(panel_frame)
        info_frame.pack(side="top", fill="both", expand=True)

        left_column_label = tk.Label(info_frame,
                                     text="\n".join([f"{column}:
                                     {value}" for column,
                                     value in left_info.items()]),
                                     anchor="nw", justify="left",
                                     font=("Helvetica", 10))
        left_column_label.pack(side="left", fill="both", expand=True)

        right_column_label = tk.Label(info_frame,
        text="\n".join([f"{column}:
        {value}" for column,
        value in right_info.items()]),
        anchor="nw", justify="left", font=("Helvetica", 10))
        right_column_label.pack(side="left", fill="both", expand=True)

        self.panel_panedwindow.add(panel_frame, weight=1)
# Add panel_frame with weight to allow expansion

    def close_panel(self):
        if hasattr(self, "panel_panedwindow"):
            self.panel_panedwindow.destroy()
```

5.3.8 Test functions and predicting unchanged labels

```
    def test_function(self):
        filename = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")]
        if filename:
            self.clear_main_menu()  # Clear the main menu before displaying CSV
            self.test_display_csv_data(filename)

    def predict_unchanged_labels(self):
        dtc_classifier = load(r'./models/DTC_Classifier')
        selected_features = load(r'./models/selected_features')
        scaler = load(r'./models/std_scaler')
        onehotencoder = load(r'./models/onehotencoder')
        # Read the dataset
        data = pd.read_csv(self.filename)
        # Use raw string literal to avoid SyntaxWarning
        if data.shape[0] < 1:
            return []
        # Select only the columns that match the cleaned selected features
        filtered_data = data[selected_features]

        # Trim the scaler mean and scale vectors to match the number of
        features in the current dataset
        scaler.mean_ = scaler.mean_[:filtered_data.shape[1]]
        scaler.scale_ = scaler.scale_[:filtered_data.shape[1]]

        # Transform the data
        sc_train = scaler.transform(filtered_data.select_dtypes(include=
        ['float64', 'int64']))

        # ---------------------------------------- predict ----
        --------------------------------

        # Predict using the Decision Tree classifier
        pred_dt = dtc_classifier.predict(sc_train)

        # Add the predicted data as a new column
        filtered_data['Model Prediction'] = pred_dt

        # Reorder columns with "Model Prediction" as the first column
        cols = filtered_data.columns.tolist()
        cols = ['Model Prediction'] + [col for col in cols if col
        != 'Model Prediction']
        filtered_data = filtered_data[cols]

        return filtered_data
```

5.3.9 Test for displaying CSV data

```
def test_display_csv_data(self, filename=None):
    if filename:
        self.filename = filename

    if not self.filename:
        return

    if self.tree:
        self.tree.destroy()
```

```
if self.scrollbar:
    self.scrollbar.destroy()

# Clear existing data
self.data = None

self.data = self.predict_unchanged_labels()

self.columns = self.data.columns.tolist()

self.table_label = tk.Label(self, text="Displaying CSV Data",
font=("Segoe UI Bold", 16, "bold"))
self.table_label.pack(pady=10)

self.back_button = tk.Button(self, text="Back", font=("Segoe UI", 12,
"bold"), bg="#d9d9d9",
fg="black", bd=2, cursor="hand2",
relief="raised", command=self.
back_to_main_menu)
self.back_button.pack(side="top", anchor="nw", padx=3, pady=3)

self.tree_frame = tk.Frame(self)
self.tree_frame.pack(fill="both", expand=True)

self.yscroll = tk.Scrollbar(self.tree_frame, orient="vertical")
self.yscroll.pack(side="right", fill="y")

self.xscroll = tk.Scrollbar(self.tree_frame, orient="horizontal")
self.xscroll.pack(side="bottom", fill="x")

self.tree = ttk.Treeview(self.tree_frame, columns=self.columns,
show="headings",
yscrollcommand=self.yscroll.set, xscrollcommand=self.xscroll.set)
self.tree.pack(side="left", fill="both", expand=True)

self.yscroll.config(command=self.tree.yview)
self.xscroll.config(command=self.tree.xview)

# Set style to change heading background color
style = ttk.Style()
style.map("Treeview.Heading", background=[("active", "lightblue")])

for col in self.columns:
    self.tree.heading(col, text=col)
    self.tree.column(col, anchor="center")
```

```
        for index, row in self.data.iterrows():
            prediction = row["Model Prediction"]
            if prediction != 1:
                self.tree.insert("", "end", values=row.tolist(),
                tags=("red_row",))
            else:
                self.tree.insert("", "end", values=row.tolist())

        self.tree.tag_configure("red_row", background="light coral")

        self.tree.bind("<ButtonRelease-1>", self.on_click)

        # Scroll to the bottom of the frame and select the last row
        self.tree.yview_moveto(1)
        self.tree.selection_set(self.tree.get_children()[-1])


if __name__ == "__main__":
    app = IntrusionDetectionApp()
    app.mainloop()
```