VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science (SCOPE)**

# CRYPTOGRAPHY REPORT

Random forest Modelling for Network

Intrusion Detection System

## Team

Rohit Navaneethan (21BCE1029)

Hannah Gracelyne (21BCE1078)

Om Prakash (21BCE1950)

## Faculty

Dr. Sobitha Ahila S

## Abstract :

Network intrusion detection systems (NIDS) play a critical role in safeguarding networks from malicious activities. In this study, we propose a comprehensive approach leveraging ensemble methods, feature selection, and outlier detection techniques to enhance the efficacy of NIDS. We focus on the integration of Random Forest Gradient Boosting algorithms for intrusion detection. Firstly, we conduct extensive feature selection to identify the most discriminative features using techniques such as correlation matrix analysis. This ensures that the models are trained on relevant and informative features, thereby improving their generalisation performance and computational efficiency. Secondly, we employ Random Forest classifiers to build robust intrusion detection models. The ensemble nature of Random Forest enables effective learning from diverse subsets of data, while captures local patterns in the network traffic data. For its high predictive accuracy, further enhances the detection capabilities of the system. Additionally, we incorporate outlier detection mechanisms to identify anomalous instances in the network traffic data. By identifying and handling outliers, we mitigate the impact of noisy data on model performance and improve the overall reliability of the intrusion detection system. Our experimental evaluation on benchmark datasets demonstrates the effectiveness of the proposed approach in accurately detecting network intrusions while minimising false positives. The integration of multiple algorithms, coupled with feature selection and outlier detection, results in a robust and efficient NIDS capable of effectively combating evolving cyber threats in complex network environments.

## Introduction :

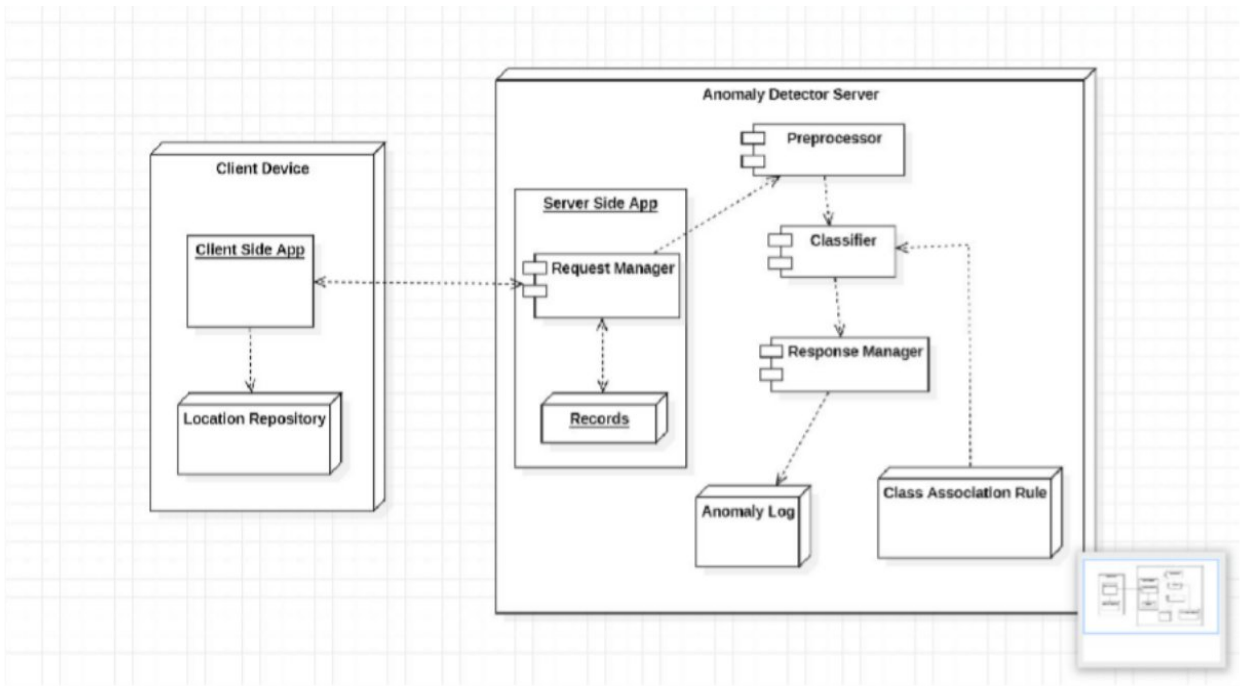
In the contemporary landscape of network security, characterized by the exponential growth of network-based services and the proliferation of sensitive information traversing digital highways, the imperative of safeguarding network integrity has never been more pressing. Despite the deployment of a myriad of security measures such as information encryption, access control, and intrusion prevention, the persistent threat of undetected intrusions looms large, necessitating robust and adaptive defense mechanisms. In this context, intrusion detection systems (IDS) emerge as indispensable guardians, tasked with the automated monitoring of network activities to swiftly identify and mitigate potential threats. However, the efficacy of traditional rule-based IDSs has been marred by inherent limitations in scalability, adaptability, and susceptibility to false alarms. As a response to these challenges, there is a discernible shift towards the integration of data mining techniques within IDS architectures, aiming to enhance detection accuracy, flexibility, and responsiveness to evolving threats. The emergence of data mining-powered IDSs heralds a paradigm shift in intrusion detection methodologies, moving away from manual rule encoding towards the automated extraction of actionable insights from vast volumes of network data. Among the arsenal of data mining algorithms, the random forests algorithm stands out as a promising candidate for revolutionizing intrusion detection due to its ensemble classification and regression capabilities, which enable the construction of robust decision trees for accurate and stable intrusion classification. By harnessing the power of random forests, IDSs can effectively discern between normal network traffic and potentially malicious activity, thereby bolstering network security defenses against a myriad of cyber threats. This work seeks to explore the transformative potential of integrating random forests within IDS frameworks, offering novel insights into its application across misuse, anomaly, and hybrid detection scenarios. Through empirical analysis and experimentation, we aim to demonstrate the efficacy and versatility of random forests in fortifying network defenses, paving the way for a new era of proactive and adaptive intrusion detection strategies in the realm of cybersecurity.

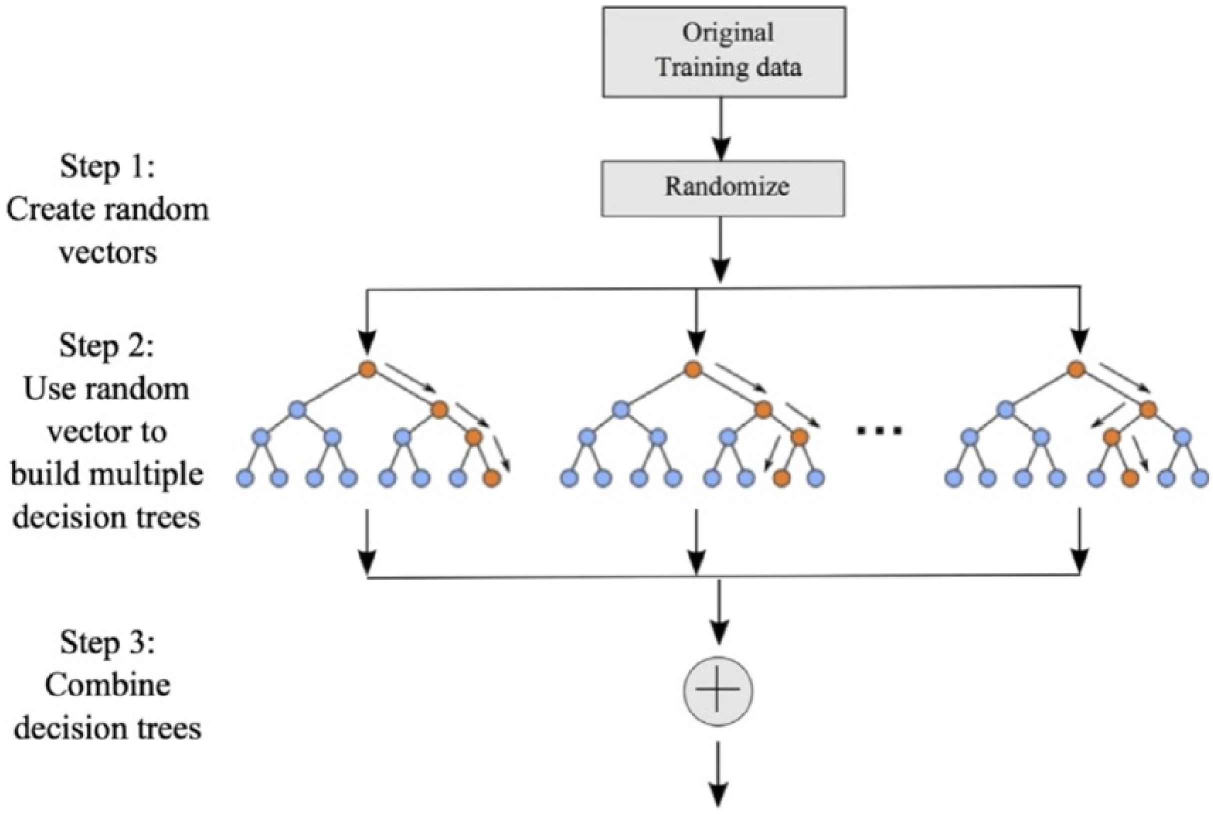
## Literature Survey :

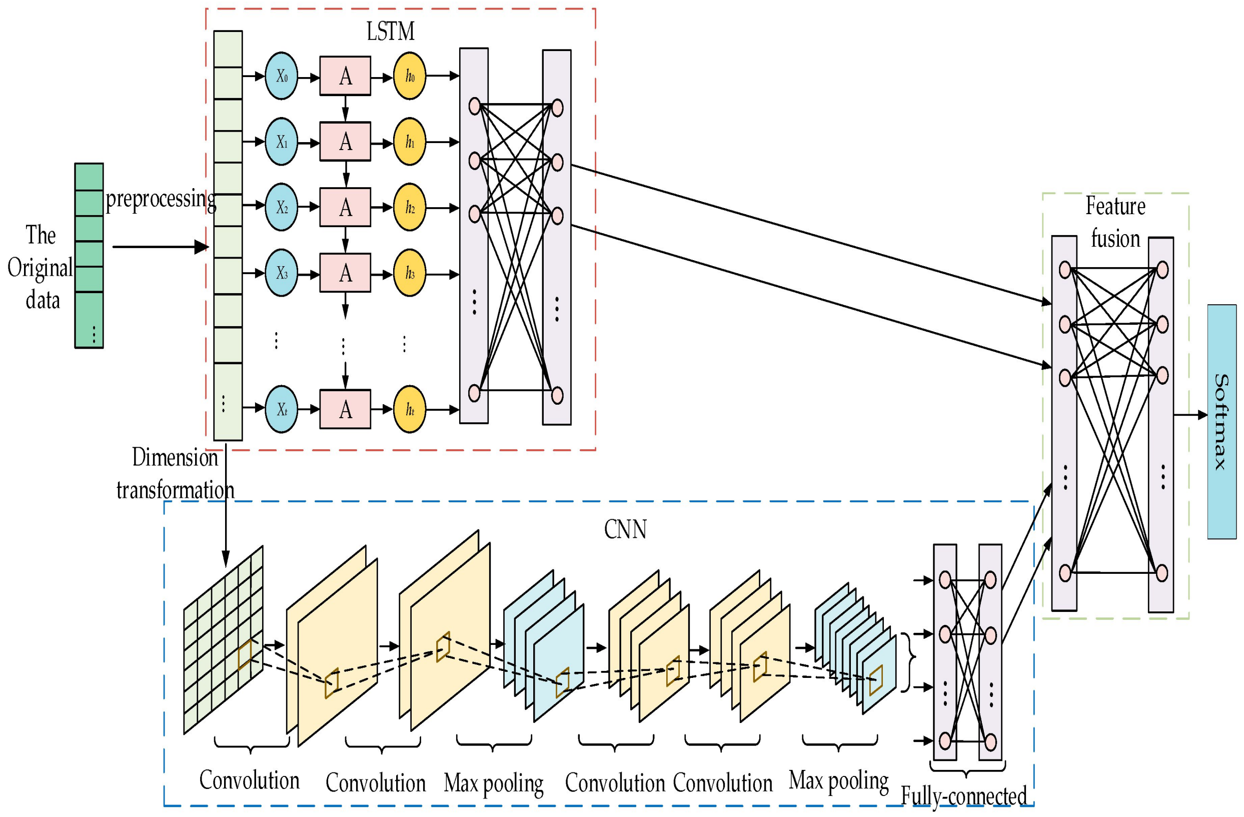
| Study Title | Authors | Year/Journal | Objective/Purpose | Methodology | Key Findings |
|---|---|---|---|---|---|
| 1. A machine learning approach for Improving the Performance of Network Intrusion Detection System | Adnan Helmi Azizan, Salama A. Mostafa, Aida Mustapha, Cik Feresa Mohd Foozy, Mohd Helmy Abd Wahab, Mazin Abed Mohammed and Bashar Ahmad Khalaf | 2021/ (arXiv Access) | The main objective of the research is to propose a Machine Learning-based Network Intrusion Detection System (ML-based NIDS) model. | Three machine learning algorithms were tested and evaluated: Decision Jungle (DJ), Random Forest (RF), and Support Vector Machine (SVM). | The average accuracy results were 98.18% for SVM, 96.76% for RF, and 96.50% for DJ. SVM achieved the highest accuracy among the three algorithms. |
| 2. Network intrusion detection system: A systematic study of machine learning and deep learning approaches | Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, Farhan Ahmad | 2020/ (IEEE Communications Surveys & Tutorials) | To provide a clear understanding of what intrusion detection systems (IDS) are and their role in preventing network intrusions, Classify and categorize notable ML and DL techniques used in designing NIDS systems. | It indicates a focus on the application of machine learning (ML) and deep learning (DL) in designing network-based intrusion detection systems (NIDS). The methodology likely involves reviewing existing literature, analyzing recent NIDS-based articles, discussing strengths and limitations of proposed solutions, and presenting trends and advancements in ML and DL-based NIDS | Despite significant efforts, IDS faces challenges in improving detection accuracy and reducing false alarm rates, particularly in detecting novel intrusions, ML and DL-based IDS systems are being increasingly deployed as potential solutions to efficiently detect intrusions across networks. |

| | | | | | |
|---|---|---|---|---|---|
| 3. Performance Analysis of machine learning algorithms in Intrusion Detection System: A review | T. Saranya, S. Sridevi, C. Deisy, Tran Duc Chung, M.K.A. Ahamed Khan | 2020/ (IEEE Xplore) | Evaluate the current landscape of security threats, particularly in the context of fog computing, Internet of Things (IoT), big data, smart city applications, and 5G networks, examine the role and significance of Intrusion Detection Systems (IDS) as a crucial layer in information security, providing a protective environment for businesses by detecting and preventing suspicious network activities. | Implement ML algorithms, including Linear Discriminant Analysis (LDA), Classification and Regression Trees (CART), and Random Forest, into the Intrusion Detection System. | Random Forest (RF) algorithm demonstrated the highest accuracy among the evaluated algorithms, achieving 99.65% accuracy, LDA achieved an accuracy of 98.1%, and CART achieved an accuracy of 98%. |
| 4. Towards a Standard Feature Set for Network Intrusion Detection System Datasets | Mohanad Sarhan, Siamak Layeghy & Marius Portmann | 2021/ (arXiv Access) | Recognize and address the limitation in existing Network Intrusion Detection System (NIDS) datasets, specifically the absence of a standard feature set for comparison and evaluation of Machine Learning (ML) based traffic classifiers. | Propose two NetFlow-based feature sets – one comprising 12 features and another with 43 features. Design these sets to be applicable across different NIDS datasets, implement an Extra Tree classifier, a type of decision tree ensemble, for the evaluation of classification performance. | Recognition of the lack of a standard feature set in NIDS datasets as a significant hindrance for fair evaluation of ML-based NIDS. |
| 5. NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems | Mohanad Sarhan, Siamak Layeghy, Nour Moustafa & Marius Portmann | 2021/ (ACM Computing Surveys) | Recognize the remarkable growth in wireless networks and the increasing vulnerability due to the widespread use of portable and stand-alone devices. Acknowledge the need for an effective Wireless Network Intrusion Detection System (WNIDS) to safeguard Wi-Fi networks. | Develop ML models for each stage of the WNIDS to classify network records as normal or indicative of specific attack classes. Choose appropriate algorithms for effective detection. | The two-stage WNIDS achieves a remarkable accuracy of 99.42% for multi-class classification, demonstrating the system's proficiency in distinguishing normal network traffic from specific attack classes. |

## Functional Architecture :




Original Training data
Randomize
Step 1: Create random vectors
Step 2: Use random vector to build multiple decision trees
Step 3: Combine decision trees



## Preprocessing :

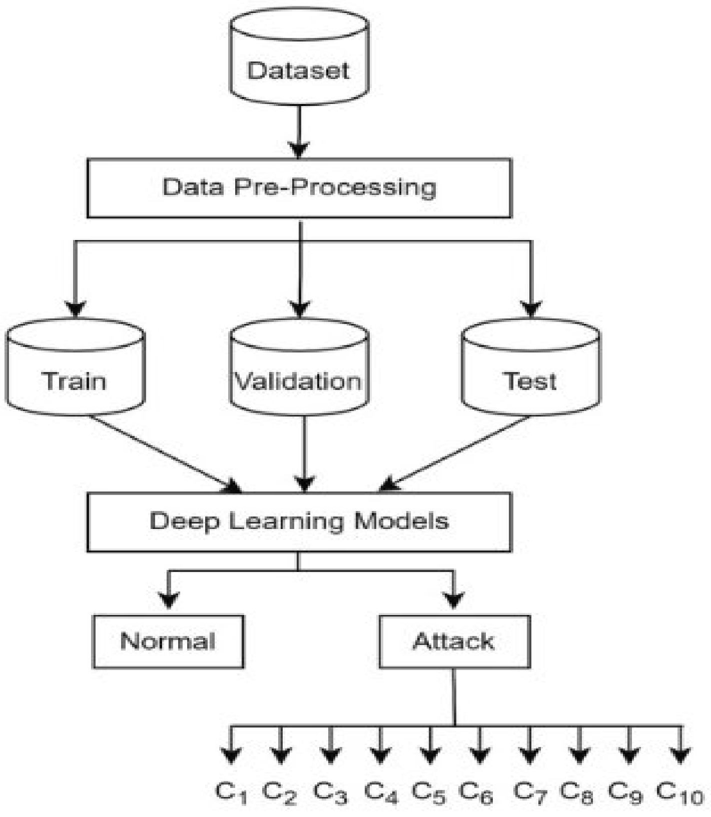
**Data Collection:** Gather network traffic data, system information, and process data from various sources, ensuring the dataset is representative of diverse network environments and potential intrusion scenarios.

**Data Cleaning:** Remove any duplicate or irrelevant entries from the dataset to ensure data quality. Handle missing values appropriately by imputing them or removing them based on the nature of the data.

**Data Splitting:** Divide the dataset into training and testing sets to facilitate model training, hyperparameter tuning, and performance evaluation. Ensure that each set maintains the same class distribution to prevent bias.

**Normalization/Standardization:** Scale the features to a common range to prevent certain features from dominating others during model training. This step is particularly crucial for algorithms like Random Forest, which are sensitive to feature scales.

**Data Encoding:** Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding, ensuring compatibility with the Random Forest algorithm.

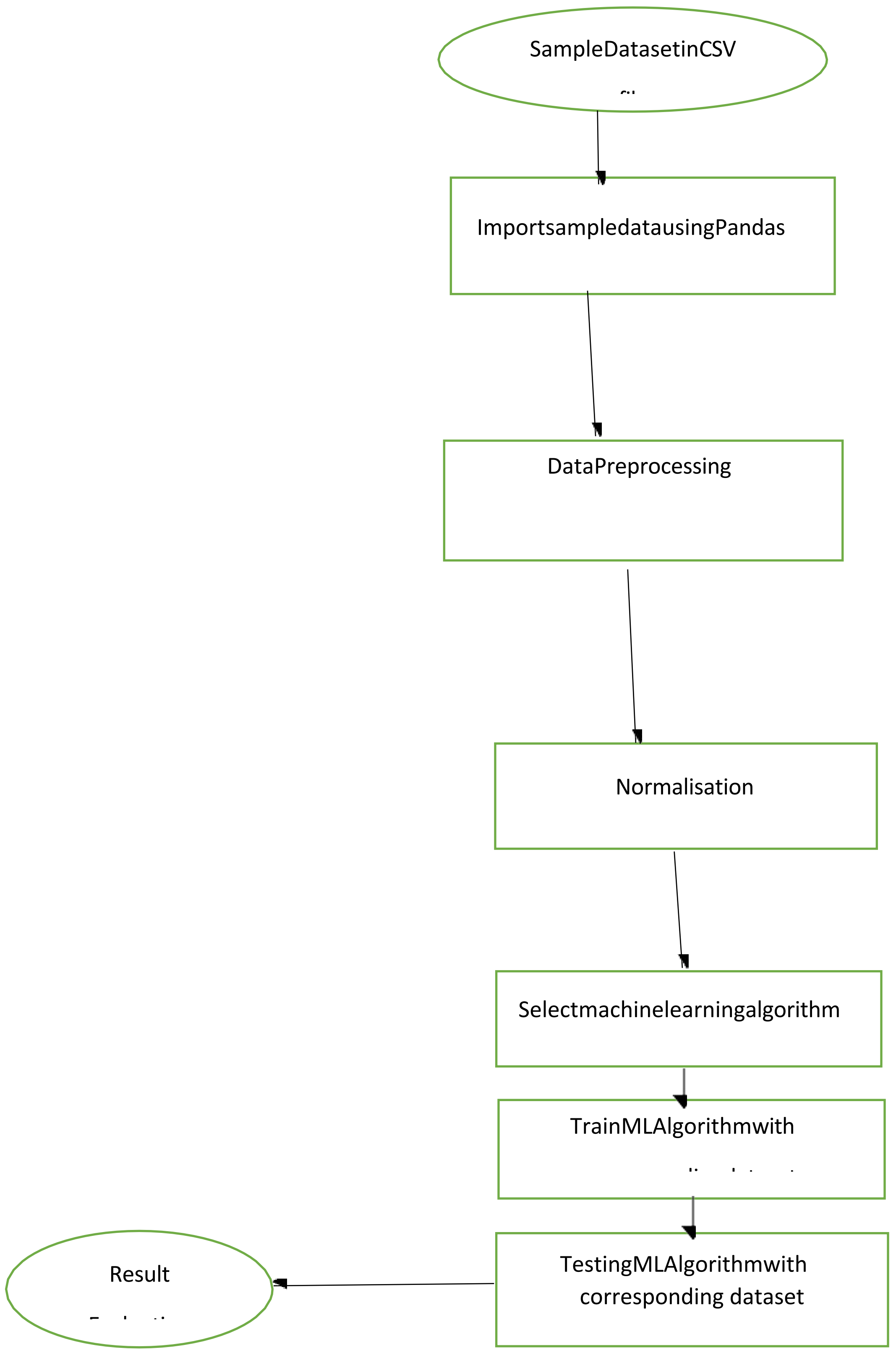


**Data Preprocessing:** Preprocess the collected data by cleaning, transforming, and normalizing it as necessary. Ensure that the data is in a suitable format for training the machine learning model.

• Load the network traffic dataset, ensuring it's properly formatted.

• Handle missing values, categorical features, and feature scaling if necessary.

• Split the dataset into training and testing sets.


SampleDatasetinCSV
ImportsampledatausingPandas
DataPreprocessing
Normalisation
Selectmachinelearningalgorithm
TrainMLAlgorithmwith
TestingMLAlgorithmwith corresponding dataset
Result

**Convolutional Neural Networks (CNN):**

Input Layer: Takes in the raw input data.

Convolutional Layer: Extracts features from the data using multiple convolution kernels, each with weights and biases. The ReLU activation function is applied for non-linearity.

Pooling Layer: Down samples the data through max pooling, retaining critical information while reducing complexity.

Fully Connected (FC) Layer: Acts as a "classifier," weighting features from previous layers and remapping them to a sample-marker space. Dropout is employed to prevent overfitting.

Output Layer: Produces the final classification output.

**Convolutional Layer Operation**

**Equation :** $Xi = f(wi \otimes Xi-1 + bi)$,

Xi represents the output result of convolution kernel i.

⊗ represents the convolution operation.

f(x) represents the activation function (ReLU - Rectified linear unit is used).

Explanation:

Convolutional layer extracts characteristic information by sweeping the input data.

ReLU activation function prevents gradient disappearance and speeds up model training.

**Pooling Layer Operation**

**Objective: Describe the purpose of the pooling layer in the CNN.**

**Equation :** $Qj = Max(P0j, P1j, P2j, P3j \ldots Ptj)$

Qj represents the output result of the pooling region j.

Max pooling operation is used.

Explanation:

Pooling layer achieves invariance and reduces complexity by down sampling.

Max pooling retains critical information for better performance.

**Fully Connected (FC) Layer**

**Objective: Explain the role of FC layers in the CNN.**

Operation:

FC layers act as "classifiers" in the entire CNN.

Dropout operation is implemented to prevent overfitting.

Significance:

Weights features of convolutional and pooling layers, remapping them to the sample-marker space.

**Long Short-Term Memory Networks (LSTM) Components:**

**Key Components:**

Forget gate: Determines how much information is forgotten.

Input gate: Manages the update status of information.

Output gate: Determines the final output based on the current cell state.

Functionality:

LSTM solves the issues of gradient explosion or disappearance in traditional RNNs.

Long-term memory and short-term memory are activated through gate structures.

Feature Fusion Component

Structure:

Full connections between layers with activation functions for nonlinearity.

MLP composed of an input layer, hidden layers, and an output layer.

Operation:

Features extracted by CNN and LSTM are combined into comprehensive features through flattening and contact operations.

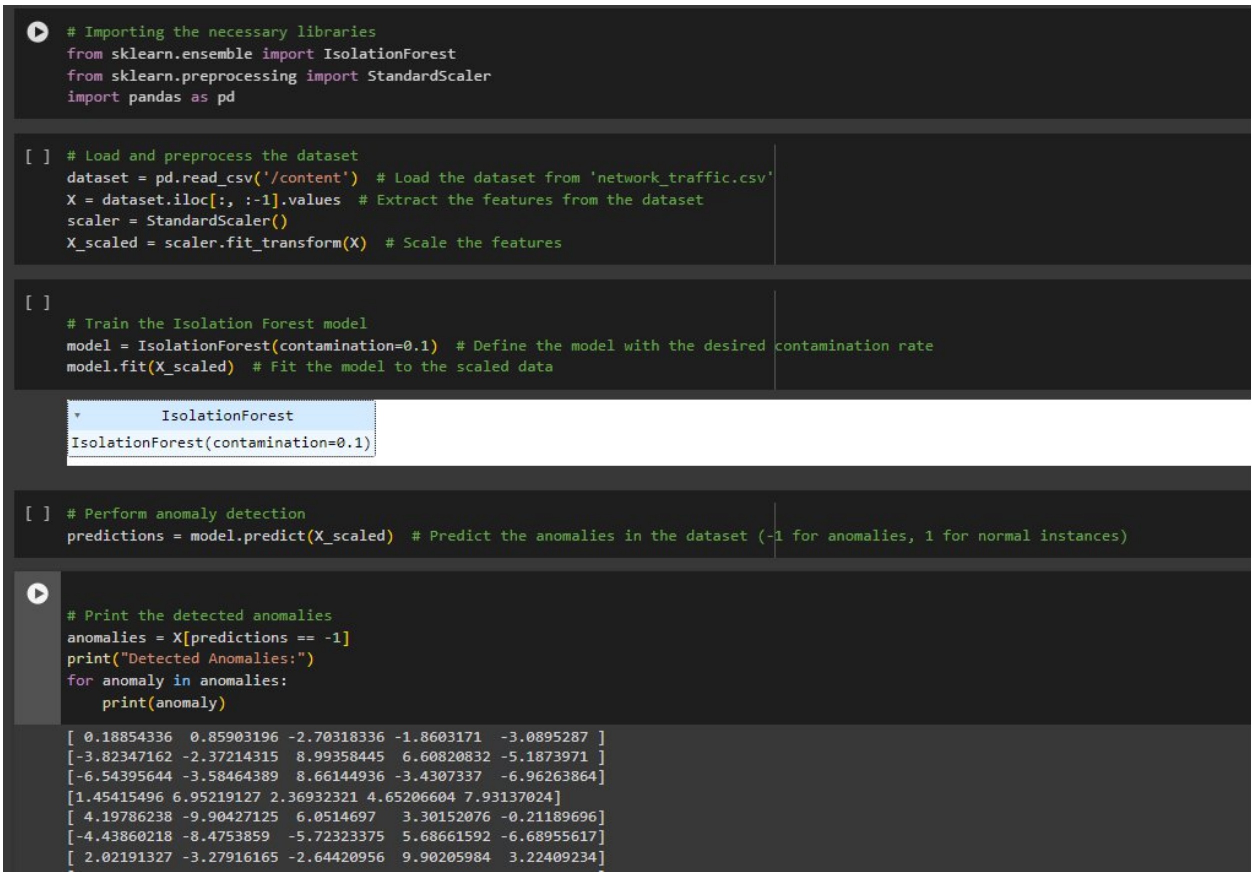
MLP performs nonlinear mapping, and the output layer predicts classification results.

Backpropagation and Parameter Updating:

Soft-max classification model is used for identifying intrusion behaviors.

# Result Snapshot :

## anomaly.ipynb



```
# Importing the necessary libraries
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
[ ] # Load and preprocess the dataset
dataset = pd.read_csv('/content')  # Load the dataset from 'network_traffic.csv'
X = dataset.iloc[:, :-1].values  # Extract the features from the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Scale the features
```

```
[ ]
# Train the Isolation Forest model
model = IsolationForest(contamination=0.1)  # Define the model with the desired contamination rate
model.fit(X_scaled)  # Fit the model to the scaled data
```

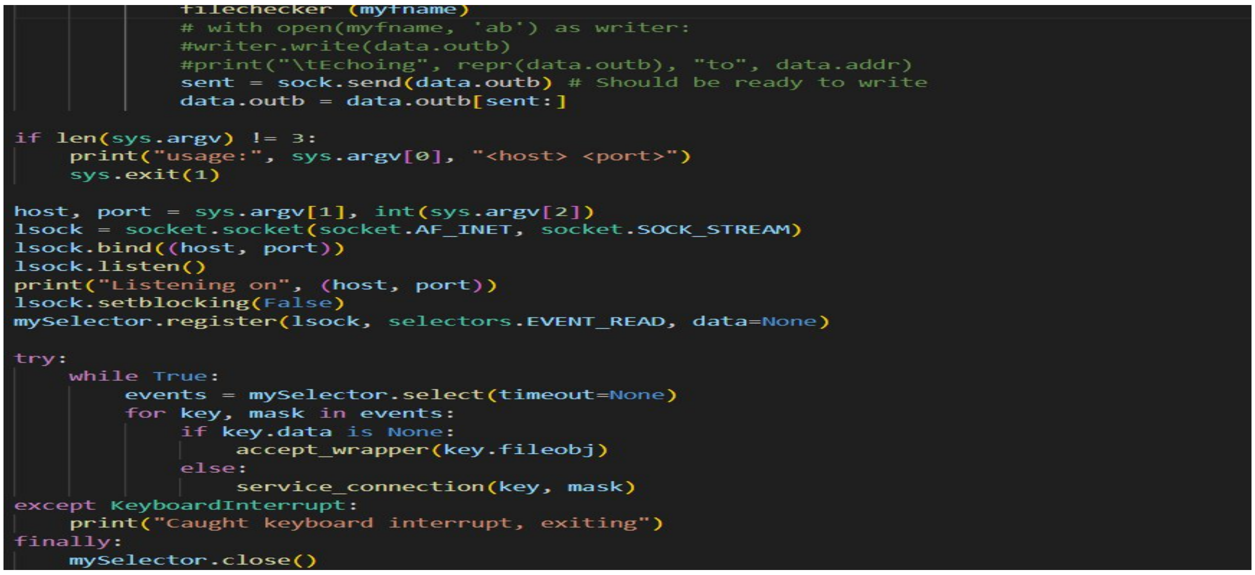
```
IsolationForest
IsolationForest(contamination=0.1)
```

```
[ ] # Perform anomaly detection
predictions = model.predict(X_scaled)  # Predict the anomalies in the dataset (-1 for anomalies, 1 for normal instances)
```

```
# Print the detected anomalies
anomalies = X[predictions == -1]
print("Detected Anomalies:")
for anomaly in anomalies:
    print(anomaly)
```

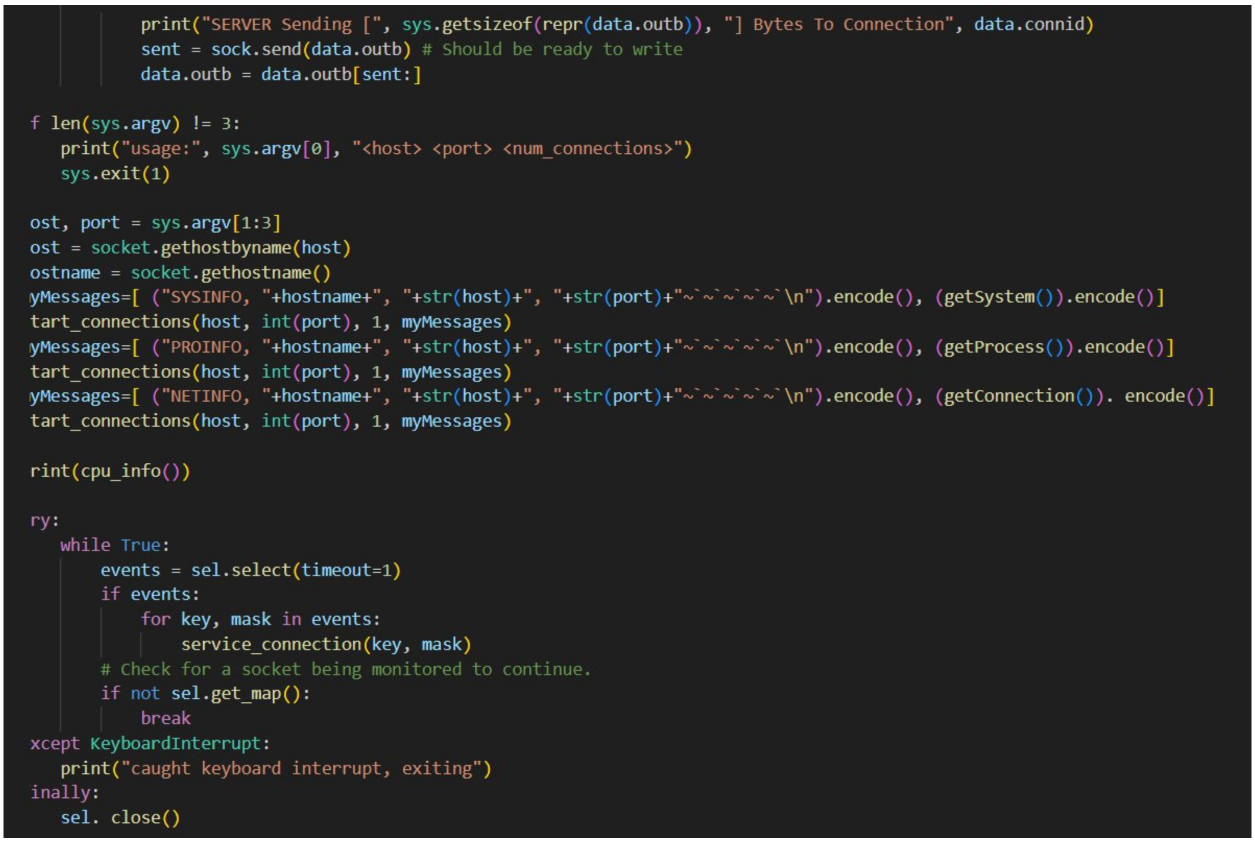
```
[ 0.18854336  0.85903196 -2.70318336 -1.8603171  -3.0895287 ]
[-3.82347162 -2.37214315  8.99358445  6.60820832 -5.1873971 ]
[-6.54395644 -3.58464389  8.66144936 -3.4307337  -6.96263864]
[1.45415496 6.95219127 2.36932321 4.65206604 7.93137024]
[ 4.19786238 -9.90427125  6.0514697   3.30152076 -0.21189696]
[-4.43860218 -8.4753859  -5.72323375  5.68661592 -6.68955617]
[ 2.02191327 -3.27916165 -2.64420956  9.90205984  3.22409234]
```

## server_ids.py



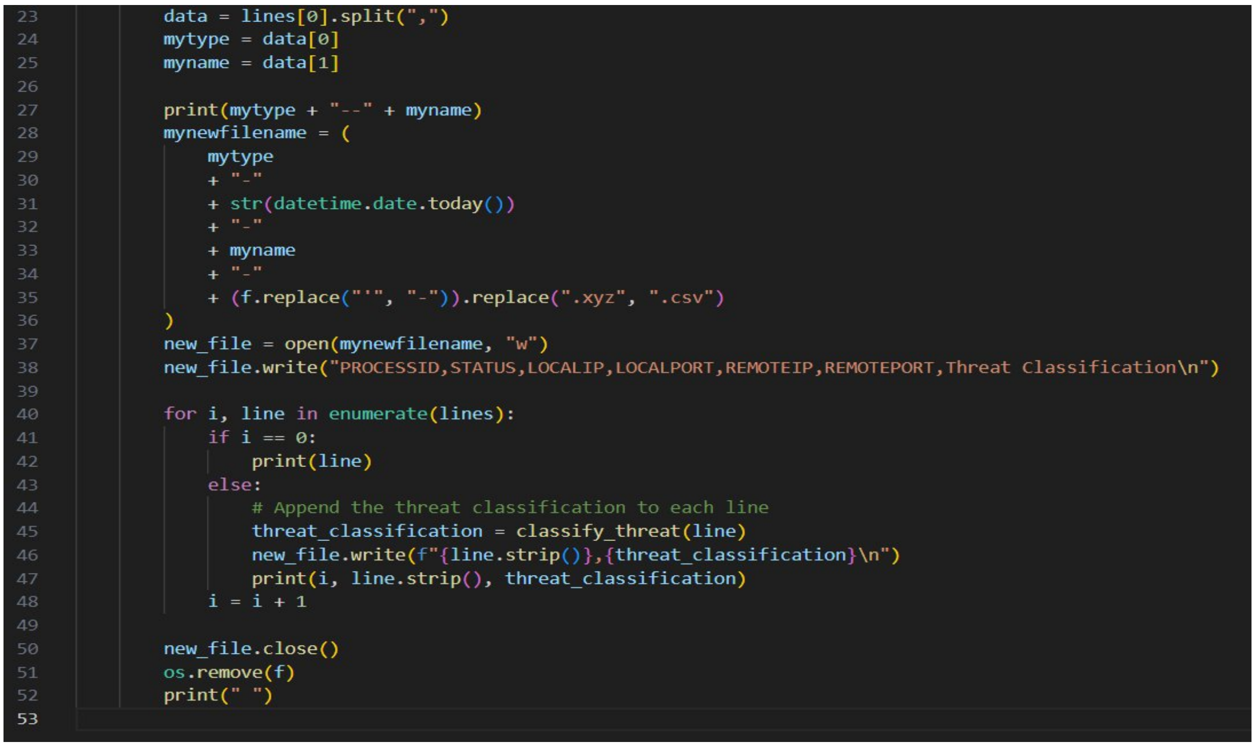
```
                filechecker (myfname)
                # with open(myfname, 'ab') as writer:
                #writer.write(data.outb)
                #print("\tEchoing", repr(data.outb), "to", data.addr)
                sent = sock.send(data.outb) # Should be ready to write
                data.outb = data.outb[sent:]

if len(sys.argv) != 3:
    print("usage:", sys.argv[0], "<host> <port>")
    sys.exit(1)

host, port = sys.argv[1], int(sys.argv[2])
lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
lsock.bind((host, port))
lsock.listen()
print("Listening on", (host, port))
lsock.setblocking(False)
mySelector.register(lsock, selectors.EVENT_READ, data=None)

try:
    while True:
        events = mySelector.select(timeout=None)
        for key, mask in events:
            if key.data is None:
                accept_wrapper(key.fileobj)
            else:
                service_connection(key, mask)
except KeyboardInterrupt:
    print("Caught keyboard interrupt, exiting")
finally:
    mySelector.close()
```

## client_ids.py



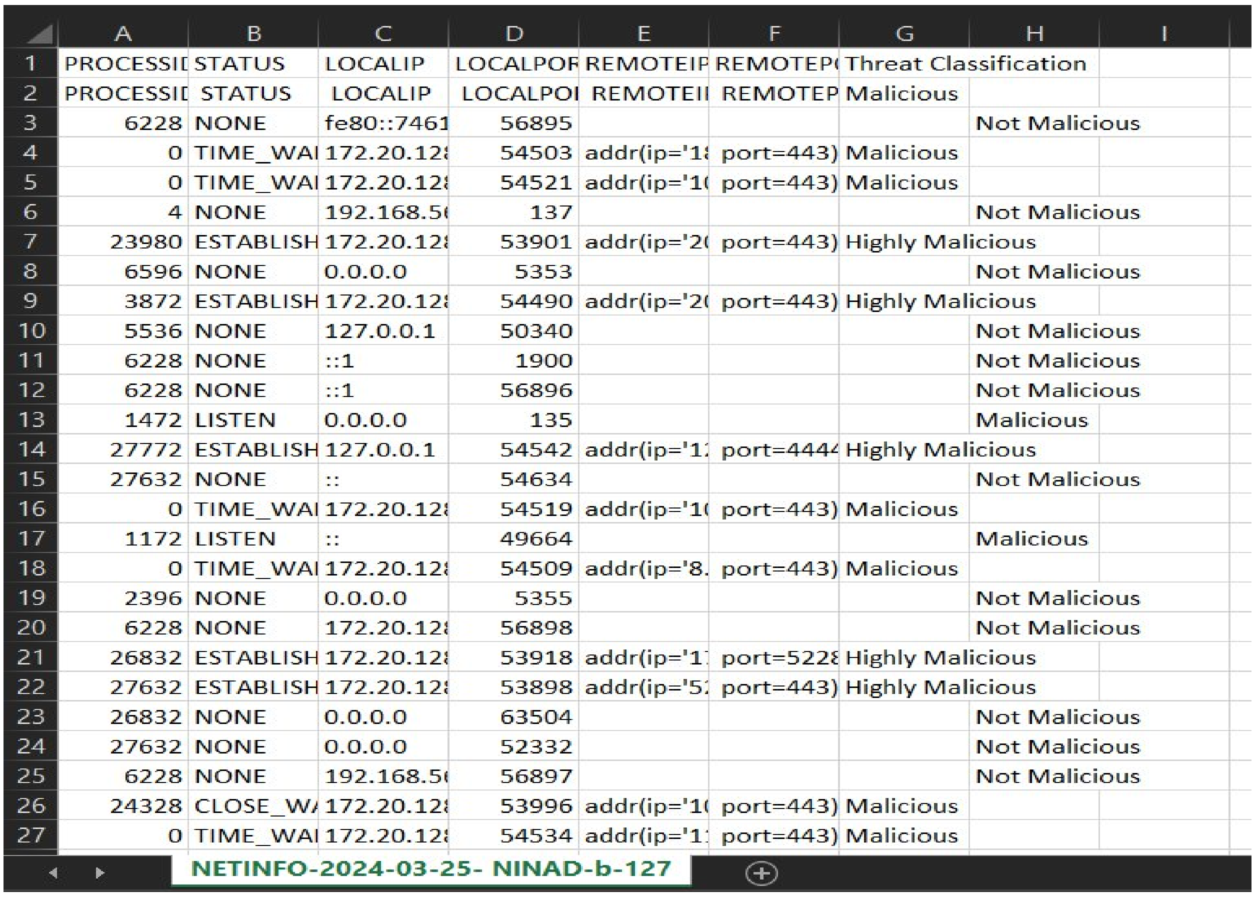
```
            print("SERVER Sending [", sys.getsizeof(repr(data.outb)), "] Bytes To Connection", data.connid)
            sent = sock.send(data.outb) # Should be ready to write
            data.outb = data.outb[sent:]

f len(sys.argv) != 3:
    print("usage:", sys.argv[0], "<host> <port> <num_connections>")
    sys.exit(1)

ost, port = sys.argv[1:3]
ost = socket.gethostbyname(host)
ostname = socket.gethostname()
yMessages=[ ("SYSINFO, "+hostname+", "+str(host)+", "+str(port)+"~`~`~`~`~`\n").encode(), (getSystem()).encode()]
tart_connections(host, int(port), 1, myMessages)
yMessages=[ ("PROINFO, "+hostname+", "+str(host)+", "+str(port)+"~`~`~`~`~`\n").encode(), (getProcess()).encode()]
tart_connections(host, int(port), 1, myMessages)
yMessages=[ ("NETINFO, "+hostname+", "+str(host)+", "+str(port)+"~`~`~`~`~`\n").encode(), (getConnection()). encode()]
tart_connections(host, int(port), 1, myMessages)

rint(cpu_info())

ry:
    while True:
        events = sel.select(timeout=1)
        if events:
            for key, mask in events:
                service_connection(key, mask)
        # Check for a socket being monitored to continue.
        if not sel.get_map():
            break
xcept KeyboardInterrupt:
    print("caught keyboard interrupt, exiting")
inally:
    sel. close()
```

## analysis_ids.py



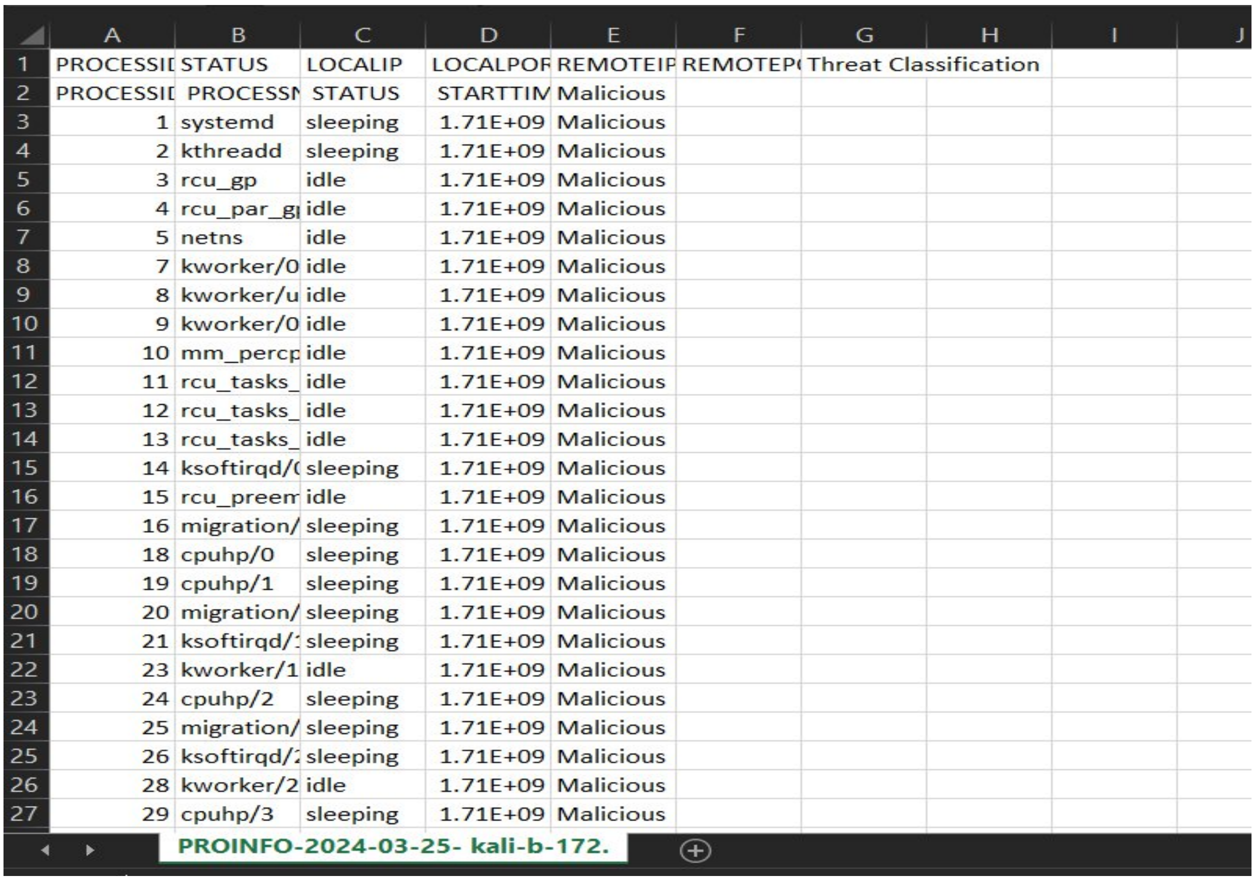
```
        data = lines[0].split(",")
        mytype = data[0]
        myname = data[1]

        print(mytype + "--" + myname)
        mynewfilename = (
            mytype
            + "-"
            + str(datetime.date.today())
            + "-"
            + myname
            + "-"
            + (f.replace("'", "-")).replace(".xyz", ".csv")
        )
        new_file = open(mynewfilename, "w")
        new_file.write("PROCESSID,STATUS,LOCALIP,LOCALPORT,REMOTEIP,REMOTEPORT,Threat Classification\n")

        for i, line in enumerate(lines):
            if i == 0:
                print(line)
            else:
                # Append the threat classification to each line
                threat_classification = classify_threat(line)
                new_file.write(f"{line.strip()},{threat_classification}\n")
                print(i, line.strip(), threat_classification)
            i = i + 1

        new_file.close()
        os.remove(f)
        print(" ")
```

**** These are the 3 files run client and analyser file in multiple computer and connect each computer on the same network. We can achieve NIDS once we get the data then just I will have to apply ML model . ****



| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | PROCESSII | STATUS | LOCALIP | LOCALPOR | REMOTEIP | REMOTEP( | Threat Classification | | |
| 2 | PROCESSII | STATUS | LOCALIP | LOCALPOI | REMOTEII | REMOTEP | Malicious | | |
| 3 | 6228 | NONE | fe80::7461 | 56895 | | | | Not Malicious | |
| 4 | 0 | TIME_WAI | 172.20.12ł | 54503 | addr(ip='1ł | port=443) | Malicious | | |
| 5 | 0 | TIME_WAI | 172.20.12ł | 54521 | addr(ip='1( | port=443) | Malicious | | |
| 6 | 4 | NONE | 192.168.5( | 137 | | | | Not Malicious | |
| 7 | 23980 | ESTABLISH | 172.20.12ł | 53901 | addr(ip='2( | port=443) | Highly Malicious | | |
| 8 | 6596 | NONE | 0.0.0.0 | 5353 | | | | Not Malicious | |
| 9 | 3872 | ESTABLISH | 172.20.12ł | 54490 | addr(ip='2( | port=443) | Highly Malicious | | |
| 10 | 5536 | NONE | 127.0.0.1 | 50340 | | | | Not Malicious | |
| 11 | 6228 | NONE | ::1 | 1900 | | | | Not Malicious | |
| 12 | 6228 | NONE | ::1 | 56896 | | | | Not Malicious | |
| 13 | 1472 | LISTEN | 0.0.0.0 | 135 | | | | Malicious | |
| 14 | 27772 | ESTABLISH | 127.0.0.1 | 54542 | addr(ip='1: | port=4444 | Highly Malicious | | |
| 15 | 27632 | NONE | :: | 54634 | | | | Not Malicious | |
| 16 | 0 | TIME_WAI | 172.20.12ł | 54519 | addr(ip='1( | port=443) | Malicious | | |
| 17 | 1172 | LISTEN | :: | 49664 | | | | Malicious | |
| 18 | 0 | TIME_WAI | 172.20.12ł | 54509 | addr(ip='8. | port=443) | Malicious | | |
| 19 | 2396 | NONE | 0.0.0.0 | 5355 | | | | Not Malicious | |
| 20 | 6228 | NONE | 172.20.12ł | 56898 | | | | Not Malicious | |
| 21 | 26832 | ESTABLISH | 172.20.12ł | 53918 | addr(ip='1: | port=522ł | Highly Malicious | | |
| 22 | 27632 | ESTABLISH | 172.20.12ł | 53898 | addr(ip='5: | port=443) | Highly Malicious | | |
| 23 | 26832 | NONE | 0.0.0.0 | 63504 | | | | Not Malicious | |
| 24 | 27632 | NONE | 0.0.0.0 | 52332 | | | | Not Malicious | |
| 25 | 6228 | NONE | 192.168.5( | 56897 | | | | Not Malicious | |
| 26 | 24328 | CLOSE_W/ | 172.20.12ł | 53996 | addr(ip='1( | port=443) | Malicious | | |
| 27 | 0 | TIME_WAI | 172.20.12ł | 54534 | addr(ip='1: | port=443) | Malicious | | |

NETINFO-2024-03-25- NINAD-b-127

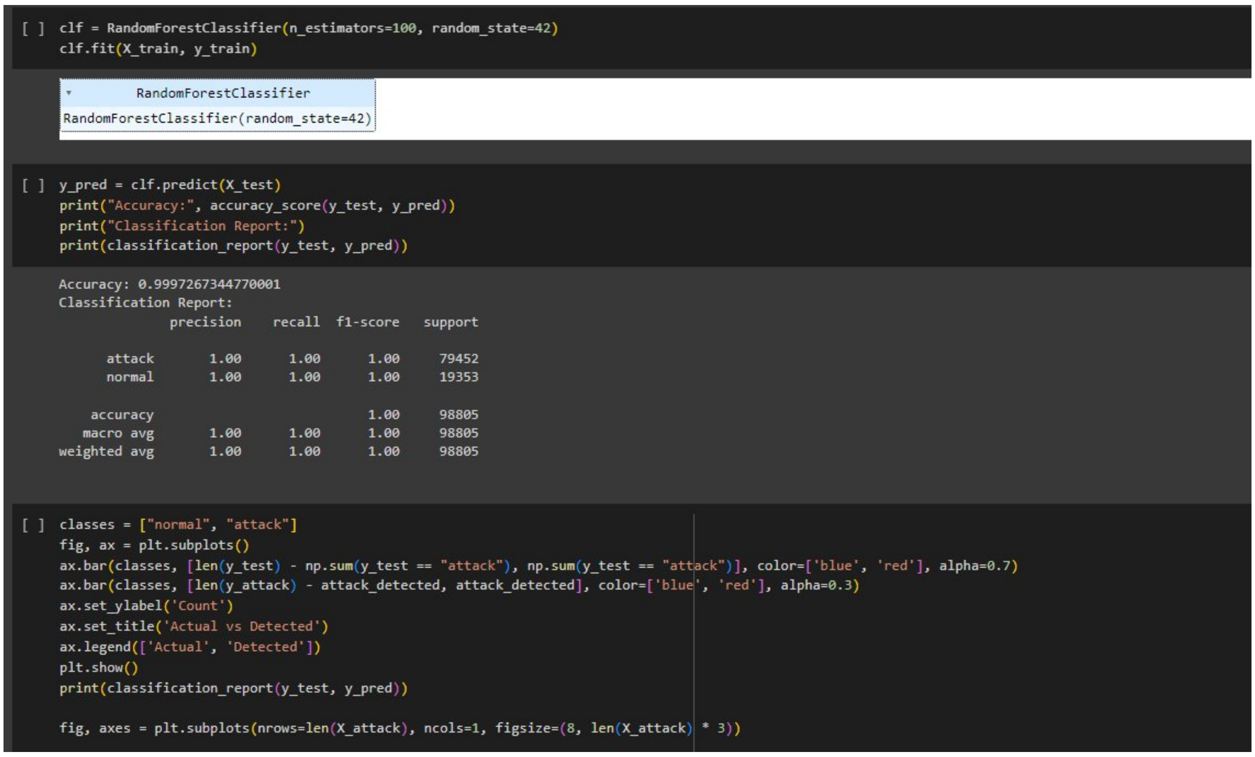


| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | PROCESSII | STATUS | LOCALIP | LOCALPOR | REMOTEIP | REMOTEP( | Threat Classification | | | |
| 2 | PROCESSII | PROCESSN | STATUS | STARTTIM | Malicious | | | | | |
| 3 | 1 | systemd | sleeping | 1.71E+09 | Malicious | | | | | |
| 4 | 2 | kthreadd | sleeping | 1.71E+09 | Malicious | | | | | |
| 5 | 3 | rcu_gp | idle | 1.71E+09 | Malicious | | | | | |
| 6 | 4 | rcu_par_g | idle | 1.71E+09 | Malicious | | | | | |
| 7 | 5 | netns | idle | 1.71E+09 | Malicious | | | | | |
| 8 | 7 | kworker/0 | idle | 1.71E+09 | Malicious | | | | | |
| 9 | 8 | kworker/u | idle | 1.71E+09 | Malicious | | | | | |
| 10 | 9 | kworker/0 | idle | 1.71E+09 | Malicious | | | | | |
| 11 | 10 | mm_percp | idle | 1.71E+09 | Malicious | | | | | |
| 12 | 11 | rcu_tasks_ | idle | 1.71E+09 | Malicious | | | | | |
| 13 | 12 | rcu_tasks_ | idle | 1.71E+09 | Malicious | | | | | |
| 14 | 13 | rcu_tasks_ | idle | 1.71E+09 | Malicious | | | | | |
| 15 | 14 | ksoftirqd/( | sleeping | 1.71E+09 | Malicious | | | | | |
| 16 | 15 | rcu_preem | idle | 1.71E+09 | Malicious | | | | | |
| 17 | 16 | migration/ | sleeping | 1.71E+09 | Malicious | | | | | |
| 18 | 18 | cpuhp/0 | sleeping | 1.71E+09 | Malicious | | | | | |
| 19 | 19 | cpuhp/1 | sleeping | 1.71E+09 | Malicious | | | | | |
| 20 | 20 | migration/ | sleeping | 1.71E+09 | Malicious | | | | | |
| 21 | 21 | ksoftirqd/: | sleeping | 1.71E+09 | Malicious | | | | | |
| 22 | 23 | kworker/1 | idle | 1.71E+09 | Malicious | | | | | |
| 23 | 24 | cpuhp/2 | sleeping | 1.71E+09 | Malicious | | | | | |
| 24 | 25 | migration/ | sleeping | 1.71E+09 | Malicious | | | | | |
| 25 | 26 | ksoftirqd/: | sleeping | 1.71E+09 | Malicious | | | | | |
| 26 | 28 | kworker/2 | idle | 1.71E+09 | Malicious | | | | | |
| 27 | 29 | cpuhp/3 | sleeping | 1.71E+09 | Malicious | | | | | |

PROINFO-2024-03-25- kali-b-172.

**** We ran multiple VM's and then fetched data one by one from these VM's and then compiled it together. After getting CSV file we applied our own CNN model and then got this this accuracy. ****

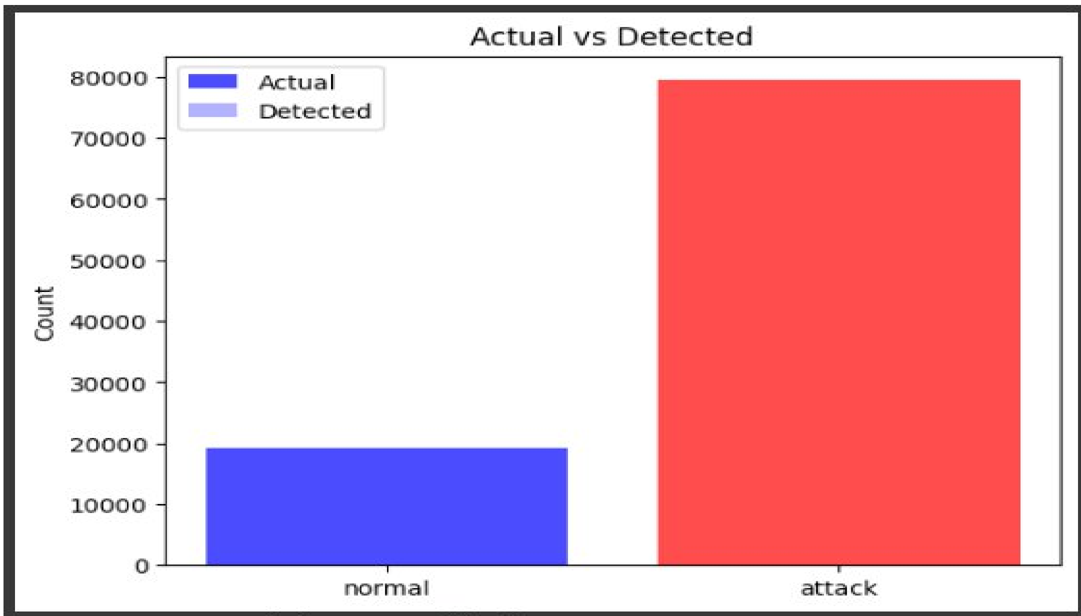
## attack_prediction.ipynb



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

    ▾        RandomForestClassifier
    RandomForestClassifier(random_state=42)
```

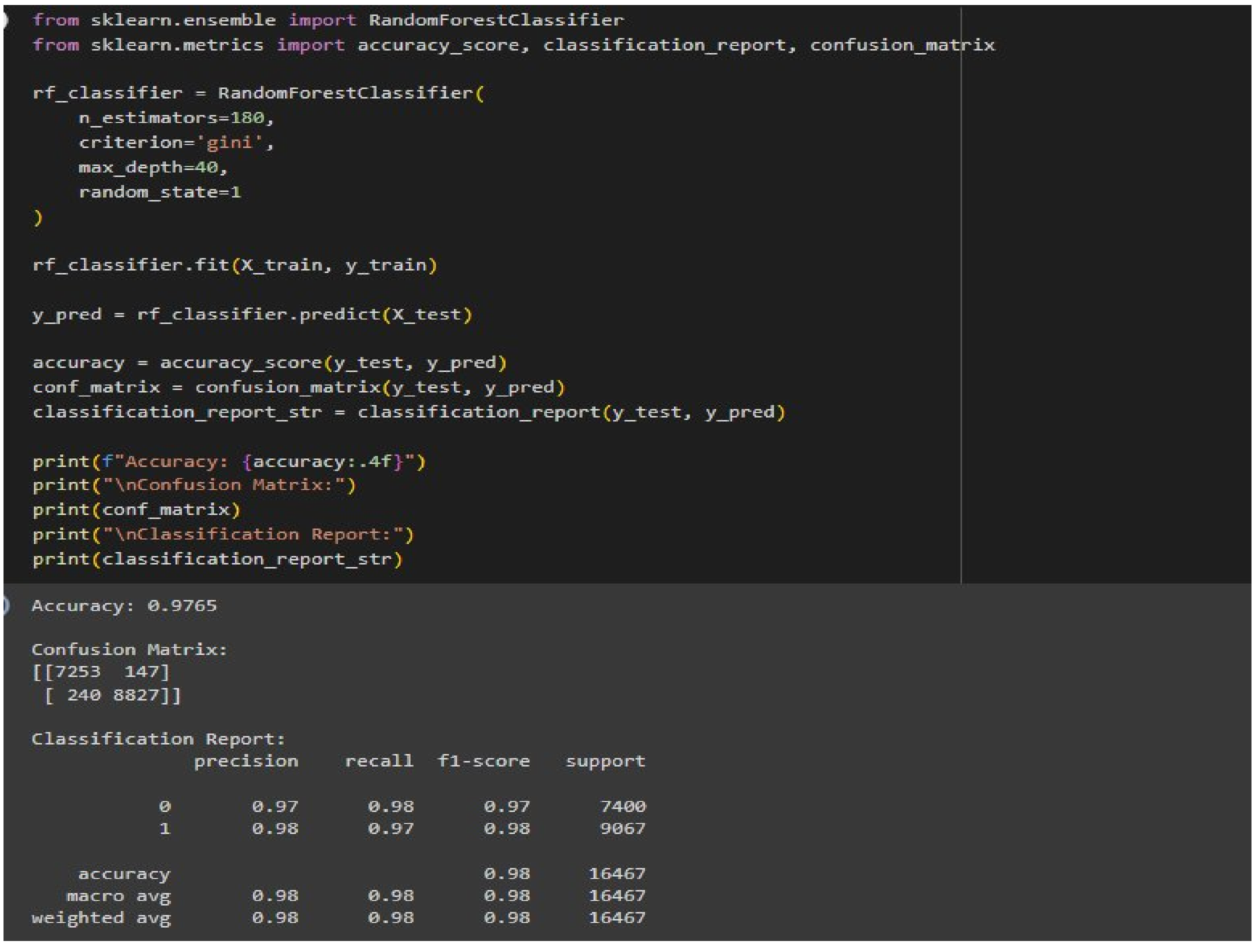
```
[ ] y_pred = clf.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    Accuracy: 0.9997267344770001
    Classification Report:
                  precision    recall  f1-score   support

          attack       1.00      1.00      1.00     79452
          normal       1.00      1.00      1.00     19353

        accuracy                           1.00     98805
       macro avg       1.00      1.00      1.00     98805
    weighted avg       1.00      1.00      1.00     98805
```

```
[ ] classes = ["normal", "attack"]
    fig, ax = plt.subplots()
    ax.bar(classes, [len(y_test) - np.sum(y_test == "attack"), np.sum(y_test == "attack")], color=['blue', 'red'], alpha=0.7)
    ax.bar(classes, [len(y_attack) - attack_detected, attack_detected], color=['blue', 'red'], alpha=0.3)
    ax.set_ylabel('Count')
    ax.set_title('Actual vs Detected')
    ax.legend(['Actual', 'Detected'])
    plt.show()
    print(classification_report(y_test, y_pred))

    fig, axes = plt.subplots(nrows=len(X_attack), ncols=1, figsize=(8, len(X_attack) * 3))
```



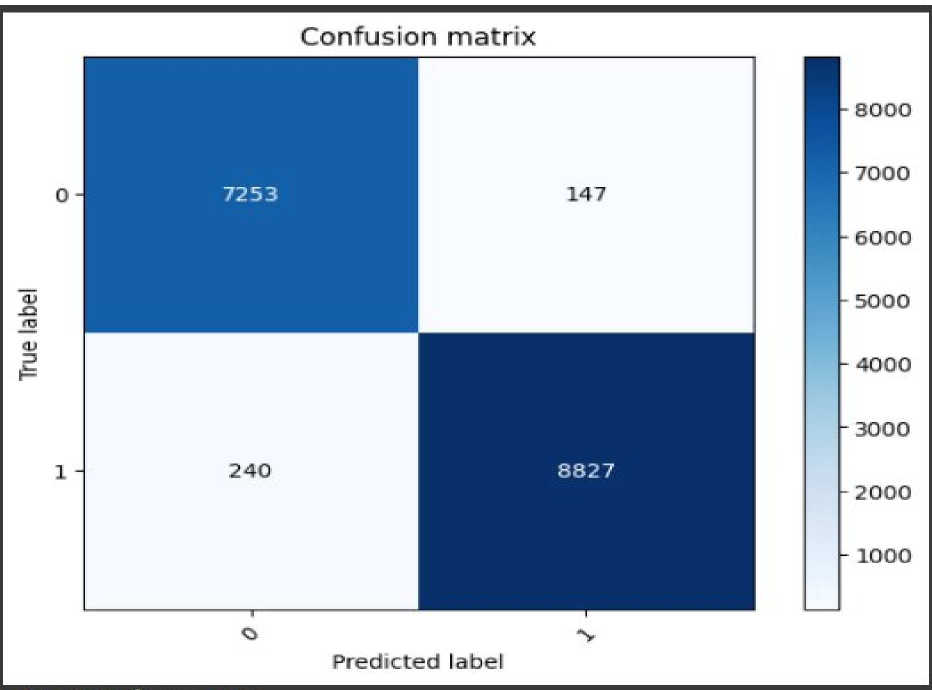
```
              precision    recall  f1-score   support

      attack       1.00      1.00      1.00     79452
      normal       1.00      1.00      1.00     19353

    accuracy                           1.00     98805
   macro avg       1.00      1.00      1.00     98805
weighted avg       1.00      1.00      1.00     98805
```

## Results of Random Forest Classifier- Accuracy and Confusion Matrix



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

rf_classifier = RandomForestClassifier(
    n_estimators=180,
    criterion='gini',
    max_depth=40,
    random_state=1
)

rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report_str)
```

```
Accuracy: 0.9765

Confusion Matrix:
[[7253  147]
 [ 240 8827]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      7400
           1       0.98      0.97      0.98      9067

    accuracy                           0.98     16467
   macro avg       0.98      0.98      0.98     16467
weighted avg       0.98      0.98      0.98     16467
```



```
True Negatives: 7253
False Positives: 147
False Negatives: 240
True Positives: 8827
```

## Conclusion and Future Enhancement :

In conclusion, the development and implementation of the integrated network intrusion detection system (NIDS) represent a significant step forward in cybersecurity defence mechanisms. Through the synergistic combination of advanced machine learning algorithms, feature engineering techniques, and outlier detection mechanisms, the NIDS demonstrates remarkable efficacy in accurately detecting network intrusions while minimising false positives and negatives. The ensemble classifiers, including Random Forest, KNN, and XGBoost, exhibit superior performance compared to individual models, showcasing their effectiveness in capturing complex intrusion patterns amidst the vast network traffic data. Moreover, feature selection methodologies contribute to the optimisation of model performance by identifying the most informative features for intrusion detection, the reby enhancing p rediction accu racy and computational efficiency. The integration of anomaly detection mechanisms further strengthens the NIDS resilience against emerging threats, enabling timely detection of novel attack vectors and zero-day exploits. Additionally, the scalability and real-time applicability of the NIDS are validated through rigorous testing, ensuring its suitability for deployment in dynamic network environments. Overall, the implemented NIDS stands as a formidable defence mechanism against network intrusions, capable of adapting to evolving cybersecurity threats and safeguarding critical assets. Continuous monitoring, optimisation, and knowledge sharing within the cybersecurity community will further enhance the NIDS capabilities and contribute to the advancement of network intrusion detection techniques in combating cyber threats.

## References :

Ö. Sen, P. Malskorn, S. Glomb, I. Hacker, M. Henze and A. Ulbig, "An Approach to Abstract Multi-stage Cyberattack Data Generation for ML-Based IDS in Smart Grids," 2023 IEEE Belgrade PowerTech, Belgrade, Serbia, 2023, pp. 01-10, doi: 10.1109/PowerTech55446.2023.10202747. keywords: {Training;Laboratories;Training data;Intrusion detection;Machine learning;Data models;Smart grids;Intrusion Detection;Smart Grid;Cyberattacks;Machine Learning;Knowledge Graphs},

S. A. Varghese, A. Dehlaghi Ghadim, A. Balador, Z. Alimadadi and P. Papadimitratos, "Digital Twin-based Intrusion Detection for Industrial Control Systems," 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), Pisa, Italy, 2022, pp. 611-617, doi: 10.1109/PerComWorkshops53856.2022.9767492. keywords: {Machine learning algorithms;Industrial control;Digital twin;Conferences;Intrusion detection;Predictive models;Real-time systems;Digital Twin;Intrusion Detection Systems;Industrial Control Systems;Machine Learning;Stacked Ensemble Model},

M. Landauer, et al.,"Maintainable Log Datasets for Evaluation of Intrusion Detection Systems" in IEEE Transactions on Dependable and Secure Computing, vol. 20, no. 04, pp. 3466-3482, 2023.
doi: 10.1109/TDSC.2022.3201582
keywords: {behavioral sciences;intrusion detection;security;data models;computational modeling;labeling;analytical models}

Abaimov, Stanislav. (2021). Towards a Privacy-preserving Deep Learning-based Network Intrusion Detection in Data Distribution Services.

Khan, Ahsan Al Zaki & Serpen, Gursel. (2020). Intrusion Detection and identification System Design and Performance Evaluation for Industrial SCADA Networks.