

# Configuration Manual

MSc Research Project  
MSc Cybersecurity

Mariam Uleyele Isedu  
Student ID: X22151079

School of Computing  
National College of Ireland

Supervisor: Mr Joel Aleburu

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Mariam Uleyele Isedu  
**Student ID:** X22151079  
**Programme:** MSc in Cybersecurity **Year:** ...2023 -2024  
**Module:** MSc Research Project  
**Lecturer:** Mr Joel Aleburu  
**Submission Due Date:** 12-08-2024  
**Project Title:** ...Network Intrusion Detection: A Cooperative Security in the IoT Ecosystem Using Ensemble ML Algorithm  
**Word Count:** 1988 **Page Count:** 17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Mariam Isedu  
**Date:** 11<sup>th</sup> of August, 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Mariam Uleyele Isedu  
Student ID: X22151079

## 1 Introduction

The configuration manual provides a systematic technique for the project titled 'Network Intrusion Detection: A Cooperative Security in The IoT Ecosystem Using Ensemble ML Algorithm'. The process encompasses the process workflow, implementation and validation required for the research. The purpose of this configuration manual is to provide any individual that encounters this research work with guidance and assistance at each step of the process including the instructions and tools used, and enabling them to achieve the required outcomes and results which are presented in a research report.

### **Project Overview:**

The research focuses on malware (botnet) detection in an IoT network environment by introducing a 'fog node' created with machine learning models and in this case with the utilization of 'stacking ensemble ML technique'. The proposed model is intended to be a proactive approach to protect IoT devices and network users from attack intrusion that can spread in within the network and then lead to a DoS/DDoS attack. In order to accomplish the objective, our proposed model utilizes the predictive strength of multiple machine learning models like the Random forest (RF) and Extreme gradient boost (XGBoost). The model explored a dataset, which was pre-processed and then features were selected using the SMOTE technique before proceeding to split the dataset into a train and test set for the model training and performance evaluation (accuracy, recall, precision and F1-score). And finally, we simulate and validate our model on a simulated-attack dataset generated from a network simulator.

## 2 Hardware and Software Requirements

All processes and methodologies carried out during the course of this research were achieved by using various hardware and software tools. Hardware and software requirements may vary based on system type and resources need to implement a chosen model. The requirements stated in this configuration document matched the research objectives and chosen machine learning models, hence why it was used.

### 2.1 Hardware Requirements

In the course of implementing this research work, below are the basic hardware requirements that were utilized to achieved the results:

- Operating System: Windows 10 Pro
- RAM: 16.0 GB
- Processor: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
- Storage: 512 GB SSD
- System Type: 64-bit operating system, x64-based processor
- Processor: Intel Core i7

## **2.2 Software Requirements**

In order to ensure the implementation of this research work was carried out to achieve the required results, some software tools and solutions were leveraged at various stages as required. The software used to attain our goal and model is subject to change based on recent research, system capacity and resources available. Some of the basic requirements include:

- Open-source Google Colab Platform
- Microsoft Excel – to view and analyse dataset in CSV format
- Python 3.12 version
- Virtual Machine – VMware workstation 16 Pro: for simulation purpose
- Ubuntu 24.04 – Linux OS to be installed on the VMware with better terminal usage
- Mininet Network Simulator – for network topology creation, attack simulation and dataset generation used for model validation.

## **3 Dataset Description**

The dataset utilised in this study was acquired from the 'Kaggle' platform, which is publicly available for researchers. The 'UNB CIC IoT 2023 dataset' created by Neto et al. (2023) from the University of Brunswick Centre of Cybersecurity was selected based on its inclusion of network traffic characteristics from 105 Internet of Things (IoT) devices. These devices were subjected to seven different forms of cyberattacks, including Denial of Service, Distributed Denial of Service (DDoS), botnets, and brute-force attacks. The entire dataset contained a CSV folder which comprised 169 separate pieces of datasets in multiple excel sheets. Each excel sheet had the same number of labels (47 columns) but distinct instances. The 'Part 00000' containing 238,688 rows was selected for this research for ease of iteration, faster processing time, quick balance and analysis, simpler debugging, and classification.

## **4 Project Implementation on Google Colab Platform**

The Implementation phase is one of the very crucial phase of the research, and various phases were considered in order to ensure a 'systematic' approach is followed to enable adjustment if need be, updates and easy review of phases to ensure they meet the needed requirements to meet the objective of the research. Phases followed include to first import the necessary python libraries and dataset into the google colab environment.

### **4.1 Dataset Pre-Processing**

After the python libraries and dataset importation, next is to carry out data pre-processing which include the data cleaning, checking for null and missing values, dropping duplicate

instances, sampling using the SMOTE technique and encoding to change categorical data to numerical data type for easy computation.

```
# Import the required Libraries

import sklearn, os
from tqdm import tqdm
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay, accuracy_score
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import matplotlib
import matplotlib.pyplot as plt

[ ] # Some display options

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

[ ] # Let's pick one file to work with (Part-00000) for easy computation

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/part-00000-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-c000.csv')
len(df)
df.info()
```

**Figure 1:** Libraries and Dataset Importation into the Google Colab Environment

Next we carry out standardization on the dataset, drop the label column and then check for missing and null values, and in the case of the research the dataset used had no null values.

```
[ ] # Next we Standardize the data

from sklearn.preprocessing import StandardScaler, RobustScaler

features = df.drop('label', axis=1)
scaler = StandardScaler()
scaled_features = pd.DataFrame(scaler.fit_transform(features), columns=features.columns)
scaled_df = pd.concat([scaled_features, df['label']], axis=1)

del df

scaled_df.info()
```

**Figure 2:** Standardization of the dataset

Now, check for the missing values; using the '*isnull().sum()*' command

```
[ ] # Checking for Missing values

#Note: No missing values found
scaled_df.isnull().sum()
```

flow_duration	0
Header_Length	0
Protocol_Type	0
Duration	0
Rate	0
Srate	0
Drate	0
fin_flag_number	0
syn_flag_number	0
rst_flag_number	0
psh_flag_number	0
ack_flag_number	0
ece_flag_number	0
cwr_flag_number	0
ack_count	0
syn_count	0
fin_count	0
urg_count	0
rst_count	0
HTTP	0
HTTPS	0
DNS	0
Telnet	0
SMTP	0
SSH	0
IRC	0
TCP	0
UDP	0
DHCP	0
ARP	0
ICMP	0
IPv	0
LLC	0
Tot sum	0
Min	0
Max	0
AVG	0

**Figure 3:** Checking for Missing and null values

Next we want to see our label value counts in order to have a good knowledge of the attack-class counts to ensure we are able to choose the right samples for implementation.

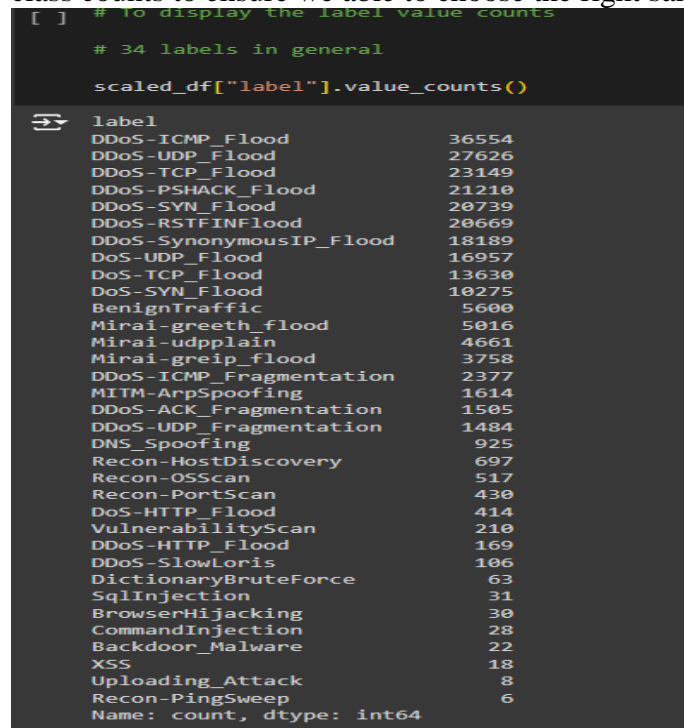


Figure 4: Label Value Counts

Figure 4 shows 34 labels that have in-balanced counts and repetition of attack types.

Next we remapped the 34 labels into 8 labels of known attacks as it relates to each attack-class.

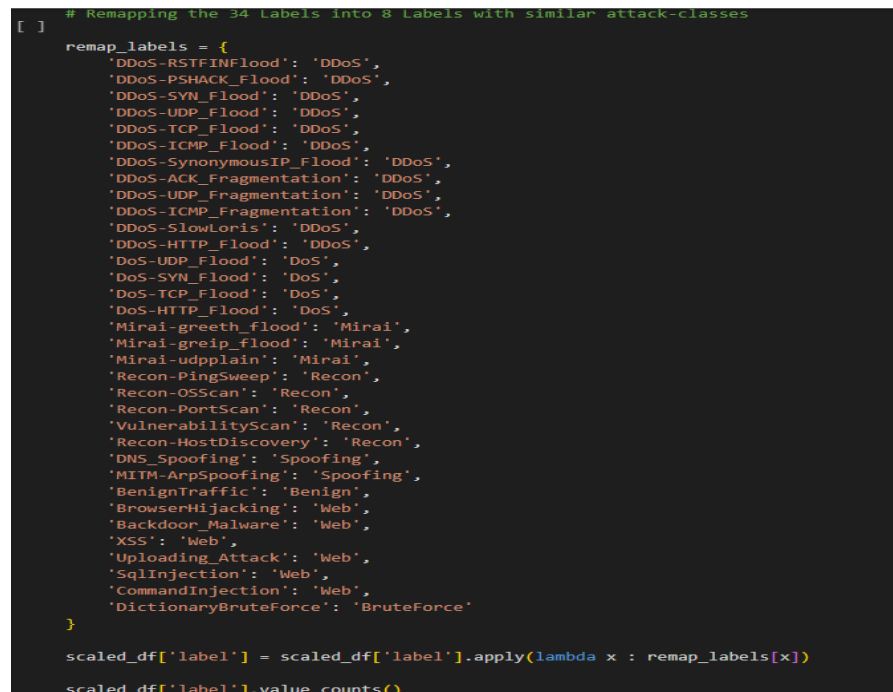


Figure 5: Remapping 34 Labels into 8 Labels

After remapping the attack-classes, we decided to choose a sample of 1000 for easy processing, but the samples were still not balanced. Next we carried out SMOTE for the balancing and label encoding to convert categorical values into numerical values. See code segments and results of these actions below:

```
[ ] # Dataset is imbalanced. Picking n samples from each class in a crude attempt to balance it.

n = 1000

# Taking n samples from each class
sampled_df = scaled_df.groupby('label', group_keys=False).apply(lambda x: x.head(n))

sampled_df.value_counts('label')
```

Figure 6: Picking 1000 samples of each attack class

```
[ ] # Separating features and labels
X = sampled_df.drop('label', axis=1)
y = sampled_df['label']

# Apply SMOTE Technique to the training data
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Create a new DataFrame with the resampled dataset
resampled_df = pd.DataFrame(X_resampled, columns=X.columns)
resampled_df['label'] = y_resampled

resampled_df.value_counts('label')
```

Figure 7: Dropping Label column

```
# Carry out 'Label Encoding' to remap categorical labels to numerics

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
resampled_df['label'] = label_encoder.fit_transform(resampled_df['label'])

resampled_df.value_counts('label')
```

Figure 8: Encoding Labels into numerical values

Below are the results from achieved from Figure 5, 6, 7 and 8 simultaneously: after the below then we drop the label column.

label	count
DDoS	173777
DoS	41276
Mirai	13435
Benign	5600
Spoofing	2539
Recon	1860
Web	137
BruteForce	63

label	count
Benign	1000
BruteForce	1000
DDoS	1000
DoS	1000
Mirai	1000
Recon	1000
Spoofing	1000
Web	1000

label	count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000

Figure 9: Remapped attack-class, SMOTE application and Label Encoding application

Check the Skewness

```
# Check the Skewness of the dataset
resampled_df.skew()
```

feature	skewness
flow_duration	47.000656
Header_Length	4.241799
Protocol_Type	2.421568
Duration	2.006376
Rate	23.497356
Srate	23.497356
Drate	0.000000
fin_flag_number	7.812731
syn_flag_number	3.078197
rst_flag_number	5.401072
psh_flag_number	4.707999
ack_flag_number	0.239402
ecr_flag_number	0.000000
cwr_flag_number	0.000000
ack_count	3.947955
syn_count	1.132819
fin_count	35.560084

Figure 10: Dataset Skewness

## 4.2 Train\_Test Split

The pre-processed dataset is divided using the 80/20 ratio for train and test sub-dataset. And then we defined our evaluation metrics which include: Accuracy, Recall, Precision and F1-score

```
[ ] # Split dataset into "Train set" and "Test set" on 80/20 ratio
X_train, X_test, y_train, y_test = train_test_split(resampled_df.drop('label', axis=1), resampled_df['label'], test_size=0.2, random_state=42)

[ ] print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(6400, 46)
(1600, 46)
(6400,)
(1600,)

[ ] # Definition of the performance metrics (Accuracy, Recall, Precision and F1-score)
def get_perf_metrics(y_pred, y_test):
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_pred, y_test, average='macro')
    precision = precision_score(y_pred, y_test, average='macro')
    f1 = f1_score(y_pred, y_test, average='macro')
    return [accuracy, recall, precision, f1]
```

Figure 11: Train\_Test Split and Performance Metrics Definition

From Figure 10 above, after carrying out the train and split, our test set became 1600 instances.

## 4.3 Application of the Machine Learning (ML) Models

After the pre-processing phase, next we carry out ML models application on the dataset. The models explored in the course of this research include; Random Forest (RF)<sup>1</sup>, Logistic Regression (LR), Extreme Gradient Boost (XGBoost), Support Vector Machine (SVM)<sup>2</sup>, Stacked Model (Experiment 5 = (SVM+XGBoost=>RF)) and our proposed model as a solution for DDoS attack detection using the Stacking technique<sup>3</sup> with minimum resources and less memory space is Stacked (RF+XGBoost=>RF).

### Experiment 1 – Random Forest (RF)

```
# Experiment 1 - the Random Forest (RF) Algorithm exploration

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Getting the feature importances
rf_feature_importances = rf_model.feature_importances_

# Create a DataFrame to display feature importances
importances_random_forest_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': rf_feature_importances})
print(importances_random_forest_df.sort_values(by='Importance', ascending=False))

plt.bar(range(len(rf_feature_importances)), rf_feature_importances)
plt.show()

# Make predictions on the test-set
rf_y_pred = rf_model.predict(X_test)

# Calculate the performance metrics
rf_metrics = get_perf_metrics(rf_y_pred, y_test)

print("[Accuracy, Recall, Precision, F1 Score]")
print(rf_metrics)
```

	Feature	Importance
39	IAT	0.149793
41	Magnitude	0.059082
1	Header_Length	0.057537
18	rst_count	0.053055
0	flow_duration	0.050046

Figure 12: Random Forest Experiment

<sup>1</sup> <https://scikit-learn.org/stable/modules/ensemble.html#histogram-based-gradient-boosting>

<sup>2</sup> <https://scikit-learn.org/stable/modules/svm.html>

<sup>3</sup> <https://scikit-learn.org/stable/modules/ensemble.html#stacking>

From the Figure 12 above, the top 5 features that influence the RF results are IAT, magnitude, Header\_lenght, Rst\_count, and Flow\_duration.

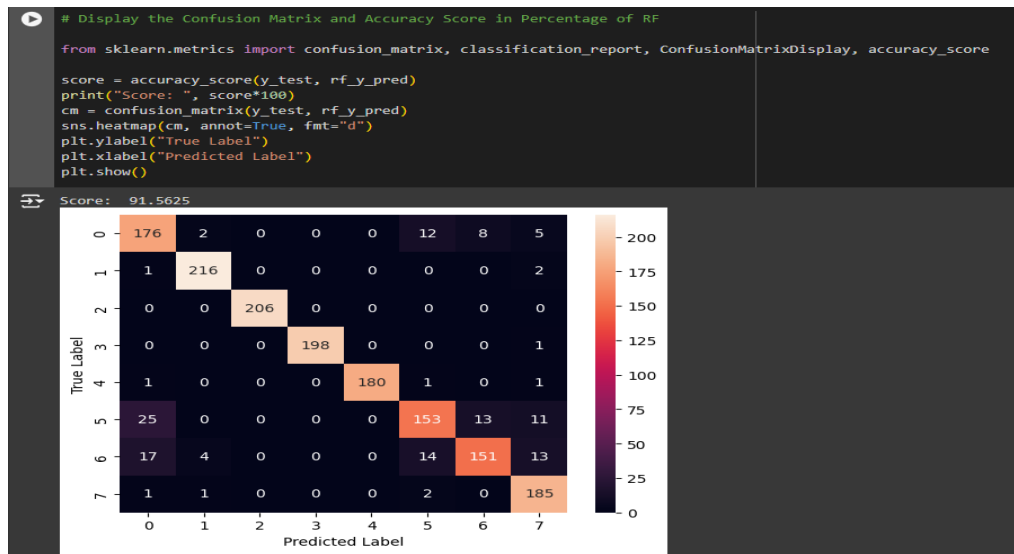


Figure 13: Confusion Matrix and Accuracy score of RF model

## Experiment 2 - Logistic Regression (LR)

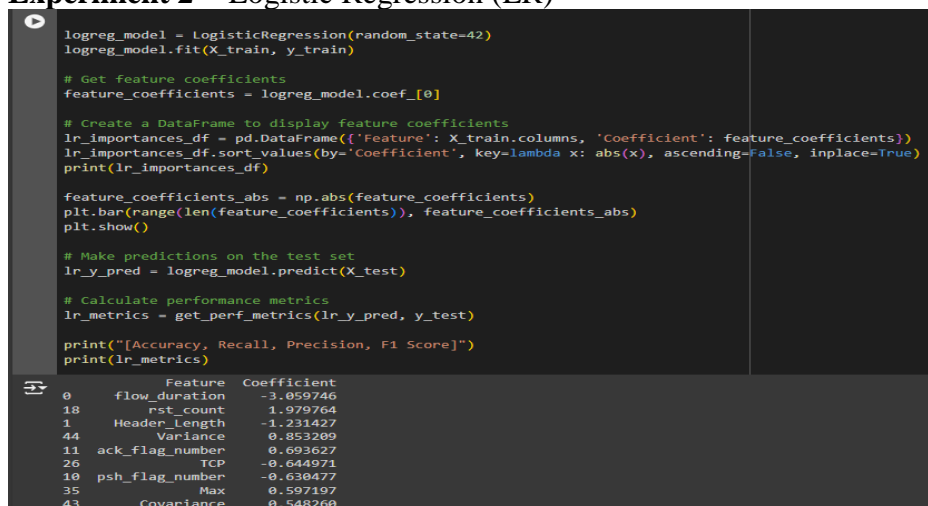


Figure 14: Logistic Regression Experiment

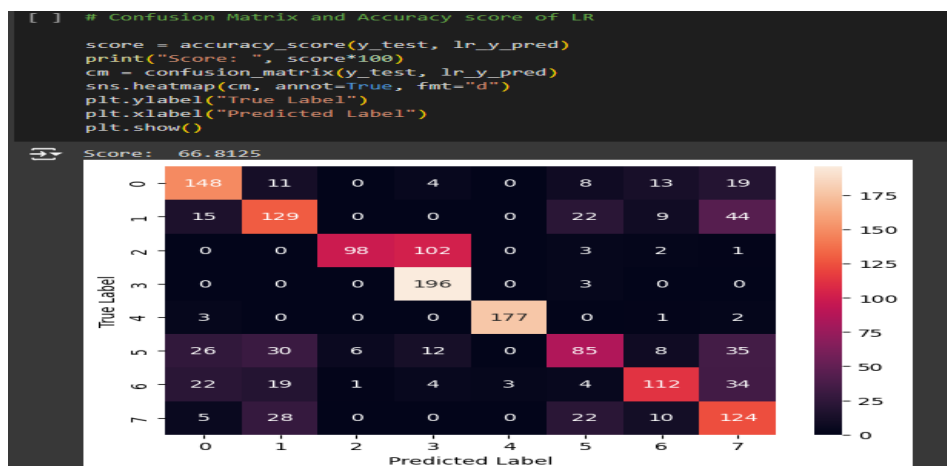


Figure 15: Confusion Matrix and Accuracy score of LR model

### Experiment 3 – Extreme Gradient Boost (XGBoost)

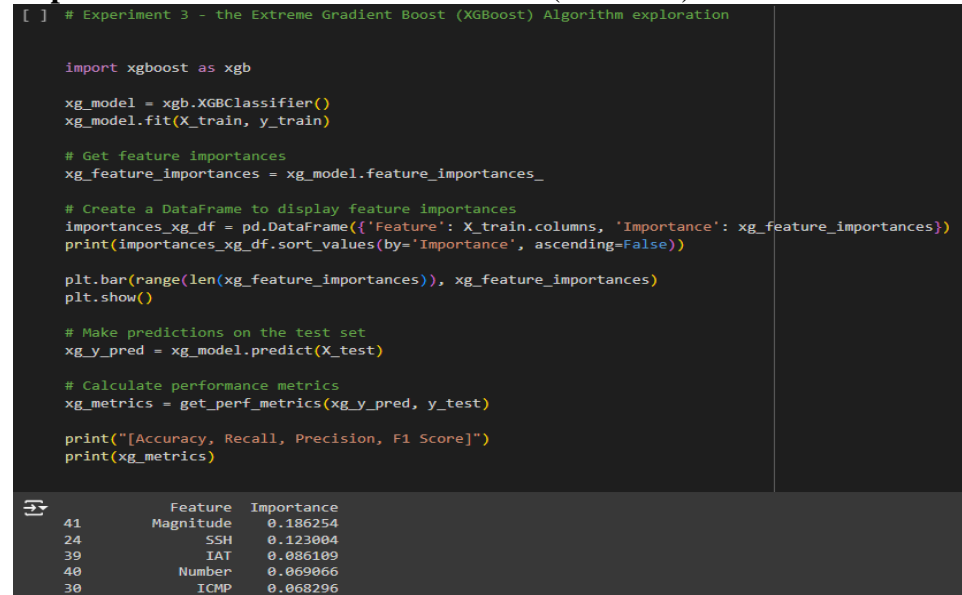


Figure 16: XGBoost Experiment

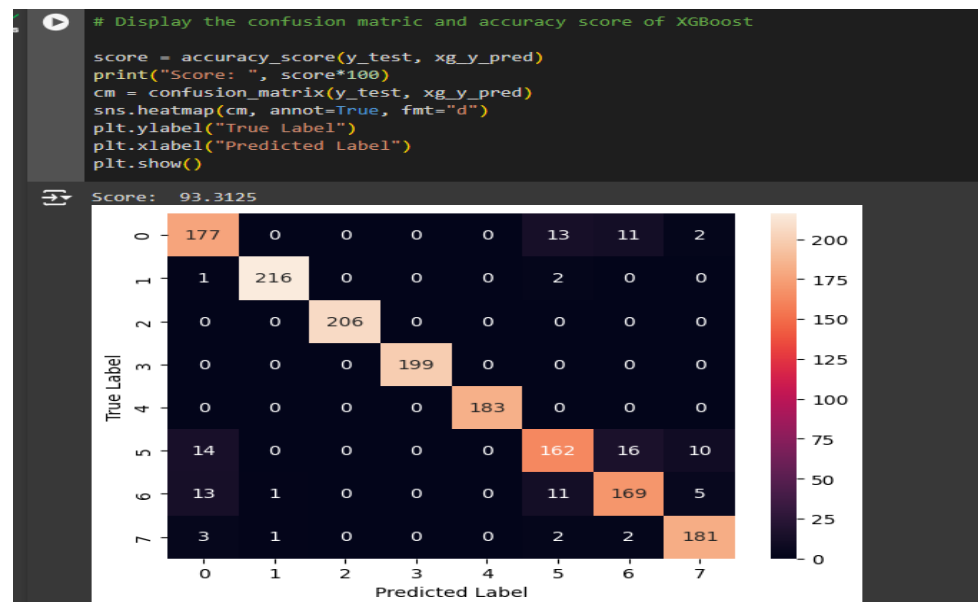


Figure 17: Confusion Matrix and Accuracy score of XGBoost model

## Experiment 4 – Support Vector Machine (SVM)

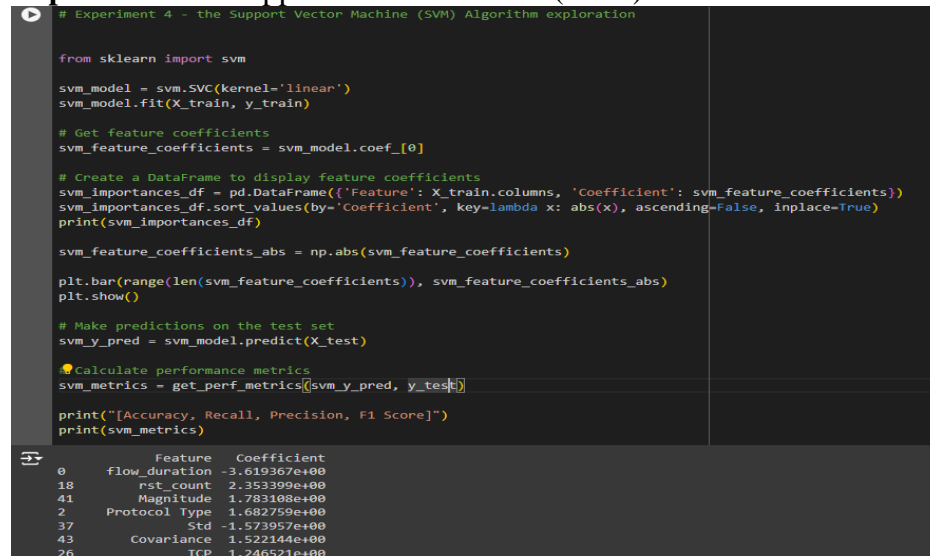


Figure 18: SVM Experiment

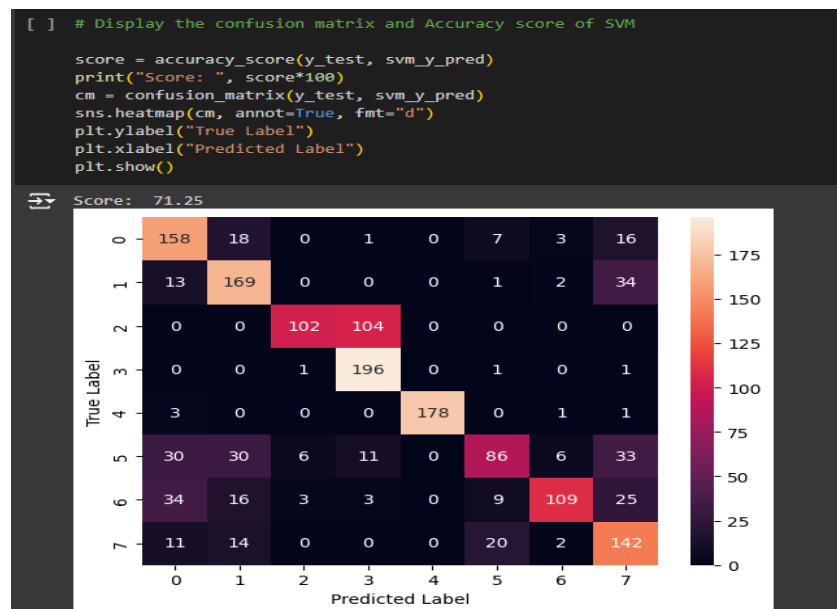


Figure 19: Confusion Matrix and Accuracy score of SVM model

## Experiment 5 – Stacked (SVM+XGBoost=>RF)

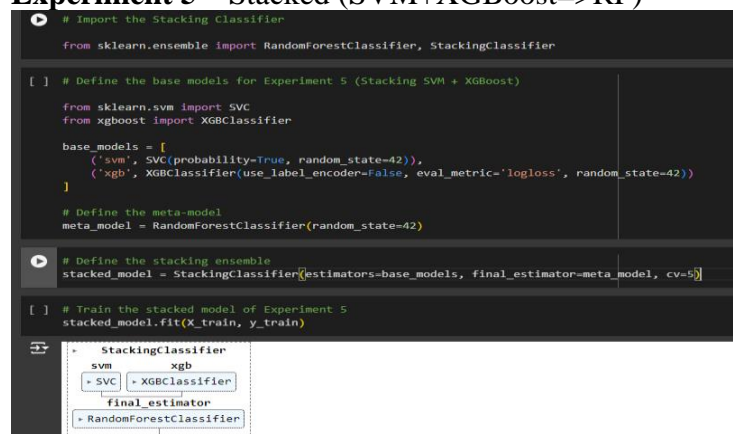
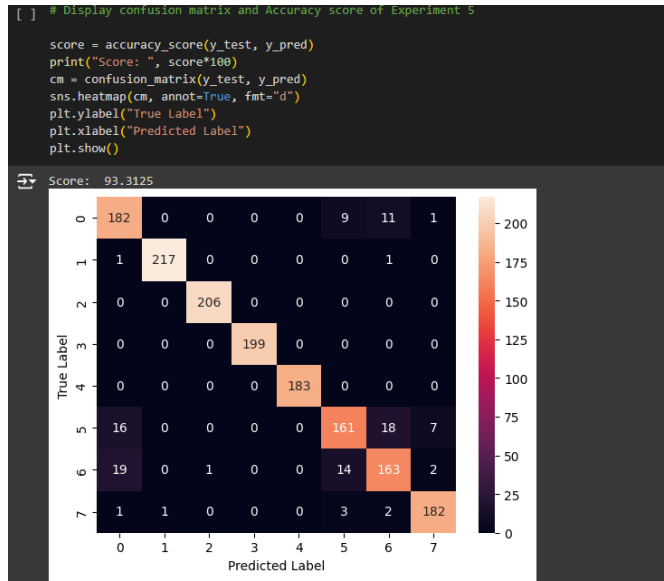
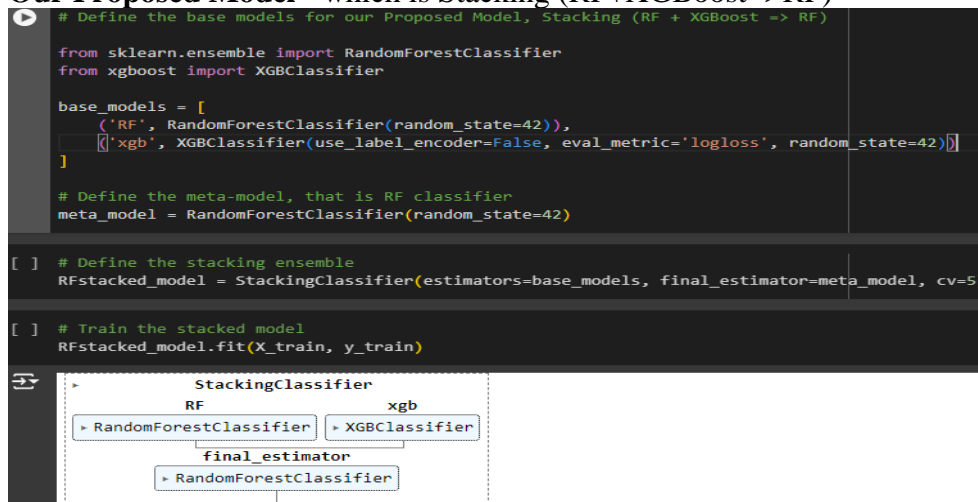


Figure 20: Stacking (SVM+XGBoost=>RF)

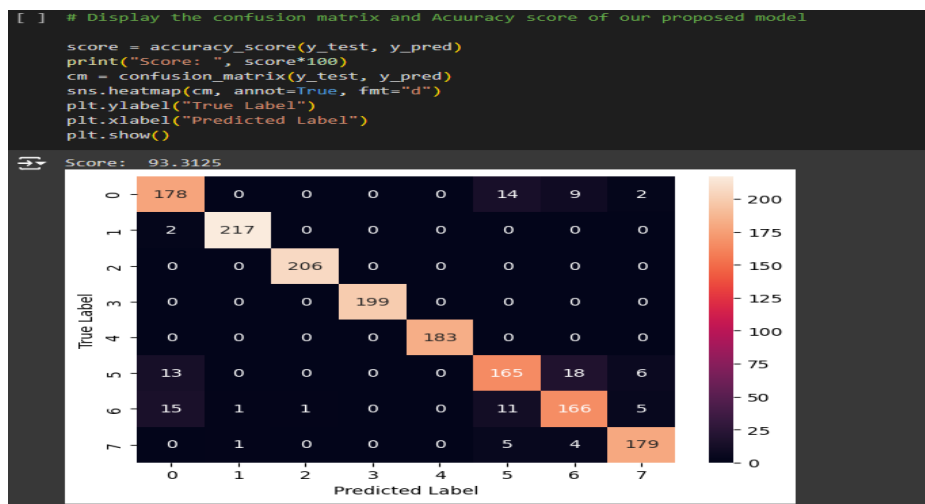


**Figure 21:** Confusion Matrix and Accuracy score of Experiment 5 (Stacking SVM+XGBoost=>RF) model

### Our Proposed Model - which is Stacking (RF+XGBoost=>RF)



**Figure 22:** Our Proposed Model Experiment Stacking (RF+XGBoost=>RF)



**Figure 23:** Confusion Matrix and Accuracy score of our Proposed Model

## 5 Evaluation

Evaluation of the models was done using the above stated metrics: Accuracy, Recall, Precision and F1-score across all models and experiments:

Models and Experiments	Accuracy	Recall	Precision	F1-Score
Experiment 1- Random Forest (RF)	0.915625	0.916523	0.915867	0.914595
Experiment 2- Logistic Regression (LR)	0.668125	0.697858	0.673208	0.666283
Experiment 3- Extreme Gradient Boost (XGBoost)	0.933125	0.932762	0.933390	0.932899
Experiment 4- Support Vector Machine (SVM)	0.7125	0.749363	0.715946	0.707919
Experiment 5: Stacked (SVM + XGBoost) => RF	0.9325	0.932756	0.932674	0.932397
<b>Proposed Model:</b> Stacked (RF + XGBoost) => RF	0.933125	0.932861	0.933226	0.932985

**Table 1:** Performance Comparison of all Models

## 6 Simulation and Validation

To commence the simulation and validation of our proposed model, we first downloaded and installed our virtual machine and in this case we used the VMware Workstation 16 Pro as its freely available on the internet along with its installation guide. Next we install an OS, we used the Ubuntu 24.04 which is a new version of Linux to ensure our models run on recent resources and can be scalable. Then we go ahead to download and install our network simulator (Mininet network simulator) and all necessary libraries.

**Note:** See links to download and installation guides below;

- <https://docs.vmware.com/en/VMware-Workstation-Pro/16.0/workstation-pro-16-user-guide.pdf>
- <https://medium.com/@florenceify74/how-to-download-install-and-run-ubuntu-in-vmware-workstation-ce5f2d4d0438>
- <https://mininet.org/overview/>

After correctly installing the above and required libraries like Python and Pyshark, next we create our network topology that contains switch, controller, host, communication link (TCLink).

```
mima2@mima2-VMware-Virtual-Platform:~/project-mini-mima$ python3.12 mi
mima.py mininet/ mininet-2.3.0.dev6.dist-info/
mima2@mima2-VMware-Virtual-Platform:~/project-mini-mima$ python3.12 mi
mima.py mininet/ mininet-2.3.0.dev6.dist-info/
mima2@mima2-VMware-Virtual-Platform:~/project-mini-mima$ python3.12 mima.py
*** Mininet must run as root ***
mima2@mima2-VMware-Virtual-Platform:~/project-mini-mima$ pip3.12 install pyshark -t .
Collecting pyshark
  Downloading pyshark-0.6-py3-none-any.whl.metadata (806 bytes)
Collecting lxml (from pyshark)
  Downloading lxml-5.2.2-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (3.4 kB)
```

Figure 24: Python, Mininet and Pyshark Downloads

Next we create our topology using Python scripts, see below

```
1 from mininet.net import Mininet
2 from mininet.node import RemoteController, OVSKernelSwitch
3 from mininet.cli import CLI
4 from mininet.log import setLogLevel, info
5 from mininet.link import TCLink
6 import os
7 import time
8
9 def startNetwork():
10     net = Mininet(controller=RemoteController, link=TCLink, switch=OVSKernelSwitch)
11
12     info("**** Adding controller\n")
13     net.addController('c0', controller=RemoteController, ip='127.0.0.1', port=6633)
14
15     info("**** Adding hosts\n")
16     h1 = net.addHost('h1', ip='10.0.0.1')
17     h2 = net.addHost('h2', ip='10.0.0.2')
18
19     info("**** Adding switch\n")
20     s1 = net.addSwitch('s1')
21
22     info("**** Creating links\n")
23     net.addLink(h1, s1)
24     net.addLink(h2, s1)
25
26     info("**** Starting network\n")
27     net.start()
28
29     info("**** Starting tcpdump to capture packets\n")
30     pcap_file = 'mirai_attack_traffic.pcap'
31     if os.path.exists(pcap_file):
32         os.remove(pcap_file)
33     os.system(f"sudo tcpdump -i s1-eth1 -w {pcap_file} &")
34
35     info("**** Starting tcpdump to capture packets\n")
36     pcap_file = 'attack_traffic.pcap'
37     if os.path.exists(pcap_file):
38         os.remove(pcap_file)
39     os.system(f"sudo tcpdump -i s1-eth1 -w {pcap_file} &")
40
41     info("**** Simulating attack: UDP flood from h1 to h2\n")
42     # Install 'hping3' on your system to run this command
43     h1.cmd('hping3 --udp -p 80 -i u10000 10.0.0.2 &')
44
45     # Allow some time for the attack to generate traffic
46     time.sleep(30)
47
48     info("**** Running CLI\n")
49     CLI(net)
50
51     info("**** Stopping network\n")
52     net.stop()
53
54 if __name__ == '__main__':
55     setLogLevel('info')
56     startNetwork()
57
```

Figure 25: Topology creation and attack simulation

Below is the results and outputs of our topology creation and attack simulation:

```
• mima2@mima2-VMware-Virtual-Platform:~/project-mini-mima$ sudo python3.12 attack-simulator.py
[sudo] password for mima2:
**** Adding controller
**** Adding hosts
**** Adding switch
**** Creating links
**** Starting network
**** Configuring hosts
h1 h2
**** Starting controller
c0
**** Starting 1 switches
s1 ...
**** Starting tcpdump to capture packets

**** Simulating attack: UDP flood from h1 to h2
tcpdump: listening on s1-eth1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
**** Running CLI
**** Starting CLI:
mininet> exit
**** Stopping network
**** Stopping 1 controllers
c0
**** Stopping 2 links
tcpdump: pcap_loop: The interface disappeared
5117 packets captured
5142 packets received by filter
0 packets dropped by kernel
.
**** Stopping 1 switches
s1
**** Stopping 2 hosts
h1 h2
**** Done
• mima2@mima2-VMware-Virtual-Platform:~/project-mini-mima$ python3.12 pcap-to-csv.py
```

Figure 26: Results of Topology creation, attack simulation, Pcap file and conversion to CSV

After simulating an attack and generating the a Pcap file. Next we need to read the pcap file with 5117 packets captured using Pyshark, and then converted our Pcap file into CSV format

in other for to ensure we can easily analyse and pre-process it to fit the features used for model training.

We have to ensure our Pcap file is organized accordingly to fit the features of the dataset we used to train our model. So we used a python code to ensure some features are captured, see below screenshots of the python code used to ensure features matches training dataset:

```
pcap-to-csv.py
1  import pysnark
2  import pandas as pd
3  import numpy as np
4
5  # Read the pcap file
6  cap = pyshark.FileCapture('attack_traffic.pcap')
7
8  # Initialize lists to store the extracted features
9  flow_duration = []
10 header_length = []
11 protocol_type = []
12 duration = []
13 rate = []
14 srate = []
15 drate = []
16 fin_flag_number = []
17 syn_flag_number = []
18 rst_flag_number = []
19 psh_flag_number = []
20 ack_flag_number = []
21 ece_flag_number = []
22 cwr_flag_number = []
23 ack_count = []
24 syn_count = []
25 fin_count = []
26 urg_count = []
27 rst_count = []
28 http = []
29 https = []
30 dns = []
31 telnet = []
32 smtp = []
33 ssh = []
34 irc = []
35 tcp = []
36 udp = []
37 dhcp = []
38 arp = []
39 icmp = []
40 ipv = []
41 llc = []
42 tot_sum = []
43 min_val = []
44 max_val = []
45 avg = []
46 std = []
47 tot_size = []
48 iat = []
49 number = []
50 magnitude = []
51 radius = []
52 covariance = []
53 variance = []
54 weight = []
55 label = []
56
57 # Custom logic to extract features from each packet
58 for packet in cap:
59     try:
60         # Example of extracting some features, more features should be added based on your needs
61         flow_duration.append(packet.length)
62         header_length.append(packet.length) # Replace with actual header length
63         protocol_type.append(packet.transport_layer)
64         duration.append(packet.sniff_time) # Replace with actual duration calculation
65         rate.append(packet.length) # Replace with actual rate calculation
66         srate.append(packet.length) # Replace with actual srate calculation
67         drate.append(packet.length) # Replace with actual drate calculation
68
69         # TCP flags
70         if hasattr(packet, 'tcp'):
71             fin_flag_number.append(packet.tcp.flags_fin)
```

```

69     # TCP flags
70     if hasattr(packet, 'tcp'):
71         fin_flag_number.append(packet.tcp.flags_fin)
72         syn_flag_number.append(packet.tcp.flags_syn)
73         rst_flag_number.append(packet.tcp.flags_reset)
74         psh_flag_number.append(packet.tcp.flags_push)
75         ack_flag_number.append(packet.tcp.flags_ack)
76         ece_flag_number.append(packet.tcp.flags_ece)
77         cwr_flag_number.append(packet.tcp.flags_cwr)
78         ack_count.append(1 if packet.tcp.flags_ack else 0)
79         syn_count.append(1 if packet.tcp.flags_syn else 0)
80         fin_count.append(1 if packet.tcp.flags_fin else 0)
81         urg_count.append(1 if packet.tcp.flags_urg else 0)
82         rst_count.append(1 if packet.tcp.flags_reset else 0)
83     else:
84         fin_flag_number.append(0)
85         syn_flag_number.append(0)
86         rst_flag_number.append(0)
87         psh_flag_number.append(0)
88         ack_flag_number.append(0)
89         ece_flag_number.append(0)
90         cwr_flag_number.append(0)
91         ack_count.append(0)
92         syn_count.append(0)
93         fin_count.append(0)
94         urg_count.append(0)
95         rst_count.append(0)
96
97     # Protocols
98     http.append(1 if 'HTTP' in packet else 0)
99     https.append(1 if 'HTTPS' in packet else 0)

```

```

99     https.append(1 if 'HTTPS' in packet else 0)
100     dns.append(1 if 'DNS' in packet else 0)
101     telnet.append(1 if 'Telnet' in packet else 0)
102     smtp.append(1 if 'SMTP' in packet else 0)
103     ssh.append(1 if 'SSH' in packet else 0)
104     irc.append(1 if 'IRC' in packet else 0)
105     tcp.append(1 if 'TCP' in packet else 0)
106     udp.append(1 if 'UDP' in packet else 0)
107     dhcp.append(1 if 'DHCP' in packet else 0)
108     arp.append(1 if 'ARP' in packet else 0)
109     icmp.append(1 if 'ICMP' in packet else 0)
110     ipv.append(1 if 'IPV' in packet else 0)
111     llc.append(1 if 'LLC' in packet else 0)
112

```

```

113     # placeholders for unimplemented features
114     tot_sum.append(0)
115     min_val.append(0)
116     max_val.append(0)
117     avg.append(0)
118     std.append(0)
119     tot_size.append(0)
120     iat.append(0)
121     number.append(0)
122     magnitude.append(0)
123     radius.append(0)
124     covariance.append(0)
125     variance.append(0)
126     weight.append(0)
127     label.append(0)
128 except AttributeError:
129     continue

```

```

131     # Create DataFrame
132     df = pd.DataFrame({
133         'flow_duration': flow_duration,
134         'Header_Length': header_length,
135         'Protocol Type': protocol_type,
136         'Duration': duration,
137         'Rate': rate,
138         'Srate': srate,
139         'Drate': drate,
140         'fin_flag_number': fin_flag_number,
141         'syn_flag_number': syn_flag_number,
142         'rst_flag_number': rst_flag_number,
143         'psh_flag_number': psh_flag_number,
144         'ack_flag_number': ack_flag_number,
145         'ece_flag_number': ece_flag_number,
146         'cwr_flag_number': cwr_flag_number,
147         'ack_count': ack_count,
148         'syn_count': syn_count,
149         'fin_count': fin_count,
150         'urg_count': urg_count,
151         'rst_count': rst_count,
152         'HTTP': http,
153         'HTTPS': https,
154         'DNS': dns,
155         'Telnet': telnet,
156         'SMTP': smtp,
157         'SSH': ssh,
158         'IRC': irc,
159         'TCP': tcp,
160         'UDP': udp,
161         'DHCP': dhcp,
162         'ARP': arp,

```

```

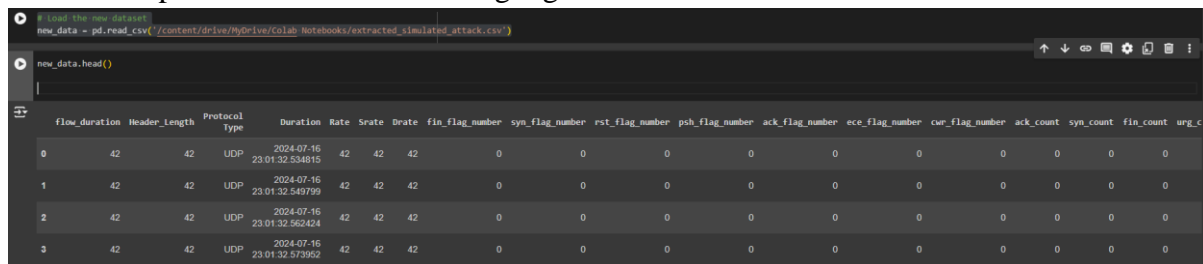
159     'TCP': tcp,
160     'UDP': udp,
161     'DHCP': dhcp,
162     'ARP': arp,
163     'ICMP': icmp,
164     'IPv': ipv,
165     'LLC': llc,
166     'Tot sum': tot_sum,
167     'Min': min_val,
168     'Max': max_val,
169     'AVG': avg,
170     'Std': std,
171     'Tot size': tot_size,
172     'IAT': iat,
173     'Number': number,
174     'Magnitude': magnitude,
175     'Radius': radius,
176     'Covariance': covariance,
177     'Variance': variance,
178     'Weight': weight,
179     'label': label
180 })
181
182 # Save to CSV for later use
183 df.to_csv('extracted_simulated_attack.csv', index=False)

```

**Figures 27:** the above 6 snapshots (27a, 27b, 27c, 27d, 27e, 27f) show reading of the Pcap file into CSV format to fit our same features as our training dataset.

## Validation:

Next we Import our CSV file into the google colab environment for validation:



The screenshot shows a Google Colab notebook with the following code and output:

```
# Load the new dataset
new_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/extracted_simulated_attack.csv')

new_data.head()
```

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	Drate	fin_flag_number	syn_flag_number	rst_flag_number	psh_flag_number	ack_flag_number	ece_flag_number	cwr_flag_number	ack_count	syn_count	fin_count	urg_c
0	42	42	UDP	2024-07-16 23:01:32.534815	42	42	42	0	0	0	0	0	0	0	0	0	0	0
1	42	42	UDP	2024-07-16 23:01:32.549799	42	42	42	0	0	0	0	0	0	0	0	0	0	0
2	42	42	UDP	2024-07-16 23:01:32.562424	42	42	42	0	0	0	0	0	0	0	0	0	0	0
3	42	42	UDP	2024-07-16 23:01:32.573952	42	42	42	0	0	0	0	0	0	0	0	0	0	0

**Figure 28:** Importation of the Validation Dataset generated from the simulated attack

From the Figure 28 above, we can see that the duration is captured in **year\_day\_time**. But our trained dataset has duration in seconds. Next we pre-process our validation dataset to ensure our model can run well with it.

```

[ ] import pandas as pd

def extract_time(timestamp_str):
    """
    Remove the date and hour portion from a timestamp string, keeping only the time part.

    Args:
    - timestamp_str (str): The timestamp string in the format 'YYYY-MM-DD HH:MM:SS.ssssss'

    Returns:
    - str: The time part of the timestamp in the format 'MM:SS.ssssss'
    """
    # Split the string by space to separate date and time
    _, time_part = timestamp_str.split(' ')

    # Split the time part by ':' to separate hours, minutes, and seconds
    parts = time_part.split(':', 2)

    # Return the parts after the hour
    if len(parts) == 3:
        return f"{parts[1]}:{parts[2]}"
    else:
        raise ValueError("Timestamp string format is incorrect")

# Apply the extract_time function to the 'Duration' column
new_data['Duration'] = new_data['Duration'].apply(extract_time)

# Display the updated DataFrame
print("\nUpdated DataFrame:")
print(new_data)

```

```
def time_to_seconds(time_str):
    """
    Convert a time string in the format MM:SS.X to total seconds.
    Args:
    - time_str (str): The time string in the format MM:SS.X
    Returns:
    - float: The total time in seconds
    """
    # Split the time string into minutes and seconds
    try:
        minutes, seconds = time_str.split(':')
        seconds, fraction = seconds.split('.')
    except ValueError:
        raise ValueError("Time string must be in the format MM:SS.X")

    # Convert the components to numbers
    minutes = float(minutes)
    seconds = float(seconds)
    fraction = float(fraction) if fraction else 0.0

    # Compute the total time in seconds
    total_seconds = minutes * 60 + seconds + fraction / (10 ** len(str(fraction)))
    return total_seconds

# Apply the time_to_seconds function to the 'duration' column
new_data['Duration'] = new_data['Duration'].apply(time_to_seconds)

# Display the updated DataFrame
print("\nUpdated DataFrame:")

Updated DataFrame:

new_data.head(40)
```

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	Drate	fin_flag_number	syn_flag_number	rst_flag_number	psh_flag_number	ack_flag_number
0	42	42	17.0	92.005348	42	42	42	0	0	0	0	0
1	42	42	17.0	92.005400	42	42	42	0	0	0	0	0

**Figure 29:** Converting Duration to seconds (29a and 29b).

Next we carry out other pre-processing steps as we did with our training dataset: Standardization, checking for missing and null values, and dropping the label column so our model can predict. See below steps:

```
[ ] # Next we Standardize the data

from sklearn.preprocessing import StandardScaler, RobustScaler

features = new_data.drop('label', axis=1)
scaler = StandardScaler()
scaled_features = pd.DataFrame(scaler.fit_transform(features), columns=features.columns)
scaled_new_data = pd.concat([scaled_features, new_data['label']], axis=1)

del new_data

scaled_new_data.head()
```

```
[ ] # Checking for Missing values

#Note: 21 null values found among the protocol type

scaled_new_data.isnull().sum()
```

```
flow_duration      0
Header_Length      0
Protocol Type      21
Duration            0
Rate               0
Srate              0
Drate              0
fin_flag_number    0
syn_flag_number    0
rst_flag_number    0
psh_flag_number    0
ack_flag_number    0
ece_flag_number    0
cwr_flag_number    0
ack_count          0
syn_count          0
```

```
[ ] # Drop samples (rows) with missing values
cleaned_new_data = scaled_new_data.dropna()
```

```
[ ] cleaned_new_data.isnull().sum()
```

```
flow_duration      0
Header_Length      0
Protocol Type      0
Duration            0
Rate               0
Srate              0
Drate              0
fin_flag_number    0
syn_flag_number    0
rst_flag_number    0
psh_flag_number    0
ack_flag_number    0
```

**Figure 30:** 30a, 30b and 30c above shows pre-processing of the validation dataset

```
# Dropping the label column to enable prediction
labelless_new_data = cleaned_new_data.drop('label', axis=1) |
```

**Figure 31:** Validation dataset after dropping label column

Next we carry out validation of our model on the '*labelless\_new\_data*' to check its performance and prediction strength.

**Recall:** our remapped label from the pre-processing of our dataset used to train the models

label	count
Benign	1000
BruteForce	1000
DDoS	1000
DoS	1000
Mirai	1000
Recon	1000
Spoofing	1000
Web	1000

label	count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000

**Figure 32:** label encoded into numerical values using label encoder

```
[ ] RFstacked_prediction= RFstacked_model.predict(labelless_new_data)
RFstacked_prediction

array([2, 2, 2, ..., 2, 2, 2])

import numpy as np

ini_array = RFstacked_prediction

# Get a tuple of unique values
# and their frequency in
# numpy array
unique, frequency = np.unique(ini_array,
                              return_counts = True)

# print unique values array
print("Unique Values:",
      unique)

# print frequency array
print("Frequency Values:",
      frequency)

Unique Values: [2 5]
Frequency Values: [5081 15]
```

**Figure 33:** Our proposed model Prediction on the validation dataset

From Figure 30 above, we can see that based on the label encoding, the attack classes ranged from Benign = 0, Brute-force = 1, DDoS = 2, DoS = 3, Mirai = 4, Reconnaissance = 5, Spoofing = 6, and Web = 7.

And from **Figure 31**, we can see that our proposed model predicted (2 = DDoS and 5 = Reconnaissance). Hence we can say our proposed model did perform well by detecting signs of DDoS attack from the network traffic with frequency of 5081 and Reconnaissance frequency of 15.

## References

- Alghamdi, R., Bellaiche, M., 2022. Evaluation and Selection Models for Ensemble Intrusion Detection Systems in IoT. *IoT* 3, 285–314. <https://doi.org/10.3390/iot3020017>
- Gupta, R., 2023. Accuracy, Precision, Recall, F-1 Score, Confusion Matrix, and AUC-ROC. Medium. URL <https://medium.com/@riteshgupta.ai/accuracy-precision-recall-f-1-score-confusion-matrix-and-auc-roc-1471e9269b7d> (accessed 6.3.24).
- Mousavi, S.A., Sadeghi, M., Sirjani, M.S., 2023. A Comparative Evaluation of Machine Learning Algorithms for IDS in IoT network, in: 2023 14th International Conference on Information and Knowledge Technology (IKT), pp. 168–174. <https://doi.org/10.1109/IKT62039.2023.10433047>
- Neto, E.C.P., Dadkhah, S., Ferreira, R., Zohourian, A., Lu, R., Ghorbani, A.A., 2023. CICIOT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. *Sensors* 23, 5941. <https://doi.org/10.3390/s23135941>
- Nguyen, H.-T., Ngo, Q.-D., Nguyen, D.-H., Le, V.-H., 2020. PSI-rooted subgraph: A novel feature for IoT botnet detection using classifier algorithms. *ICT Express* 6, 128–138. <https://doi.org/10.1016/j.ict.2019.12.001>
- Singh, A., Prakash, J., Kumar, G., Jain, P., Ambati, L., 2024. Intrusion Detection System: A Comparative Study of Machine Learning-Based IDS. *Journal of Database Management* 35, 1–25. <https://doi.org/10.4018/JDM.338276>