National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Shahna Shahul Hameed
Student ID: x22235094

School of Computing
National College of Ireland

Supervisor: Sheresh Zahoor

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Shahna Shahul Hameed
Student ID:	x22235094
Programme:	Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Sheresh Zahoor
Submission Due Date:	12/12/2024
Project Title:	DeepDefend: Optimized Multi-Model Approach for Network Intrusion Detection Using Deep Learning and IoT Security Enhancement
Word Count:	835
Page Count:	54

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:	Shahna Shahul Hameed
Date:	11th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shahna Shahul Hameed
x22235094

1 Introduction

This configuration manual gives a detailed instructions for replicating the setup, execution, and shows all the experimentation that is involved in the proposed project: "DeepDefend: Optimized Multi-Model Approach for Network Intrusion Detection Using Deep Learning and IoT security Enhancement". The documentation describes the software tools, system requirements, and configuration steps that were required to demonstrate the experiments, train models and deploy the web application.

2 System Configuration

This section gives the details of the system's hardware requirements that were used to support this proposed study.

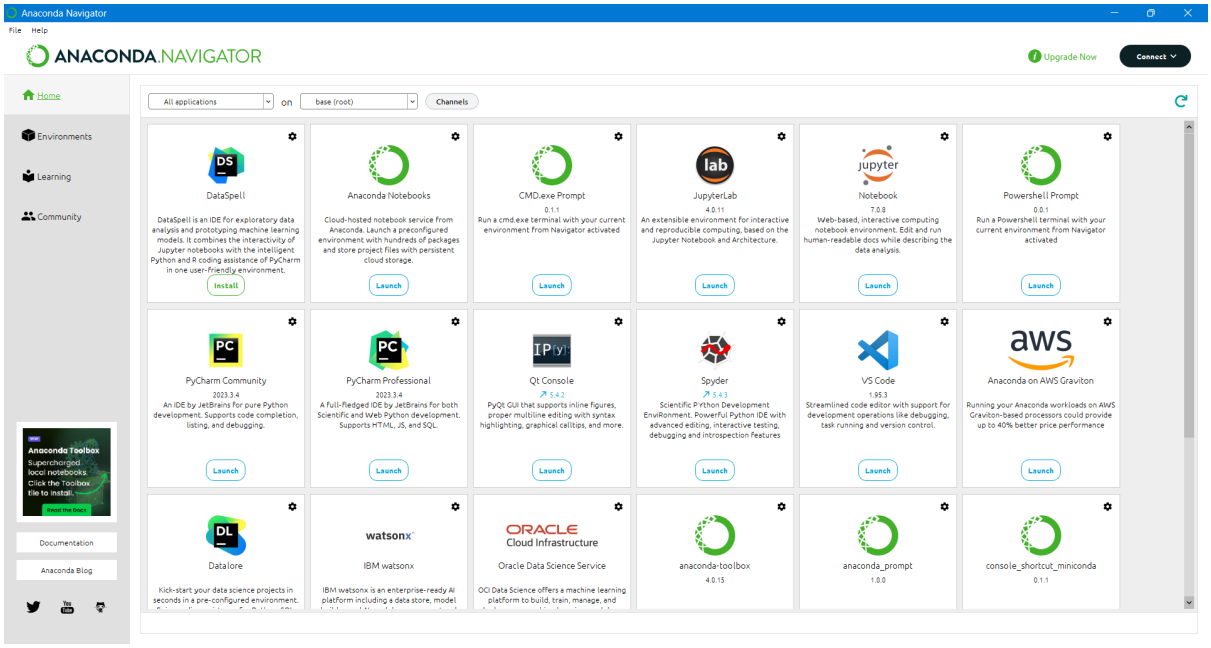
2.1 Hardware Requirements

- Operating System: Windows 11
- Processor: 13th Gen Intel(R) Core(TM) i5-1340P 1.90 GHz
- RAM: 16 GB
- Storage: 512 GB SSD
- System Type: 64-bit operating system, x64-based processor
- GPU: Intel(R) Iris(R) Xe Graphics

2.2 Software Requirements

- Python Version: 3.11.1
- IDE/Platform Used: Jupyter Notebook (via Anaconda)

Python being the primary programming language that was used throughout the work, version 3.11.1 version is utilized. IDE platform that is used in this study is Jupyter Notebook for whole implementation and execution of this study, through a comprehensive package called Anaconda Navigator.



2.3 Required Python Libraries

1. **Numpy:** This provides support for numerical computations and matrix operations. Numpy is useful to handle array-based structures in the preprocessing steps. In this study, nu,py is sed for efficient data manipulation and computation in the models requiring mathematical operations on arrays.

2. **Pandas:** Pandas are required for versatile data manipulation and analysis library. It is used extensively for reading datasets like UNSW-NB15, NSL-KDD and preprocessing steps such as handling missing values, feature extraction and data splitting.

3. **Matplotlib:** This generates visualization like heatmaps, feature importance charts, and accuracy plots in the study to give more detailed insights into data and model performance.

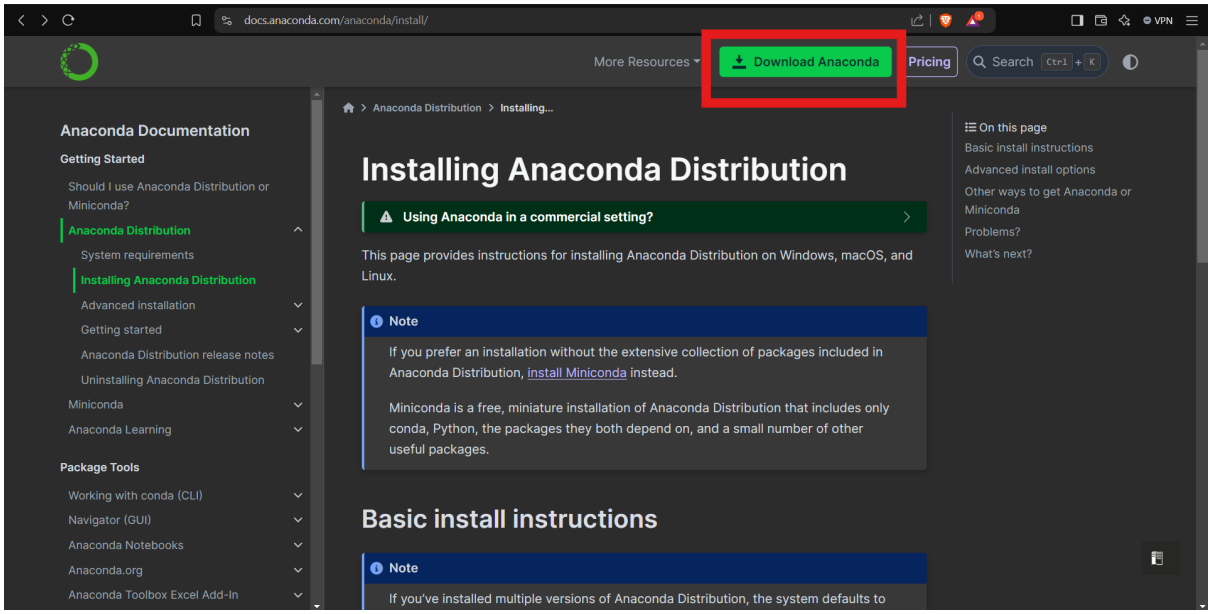
4. **Seaborn:** Built on Matplotlib, this library gives high-level visualization capabilities for statistical graphics, and in the study it is used to create correlation heatmaps and distribution plots which helps in feature selection and explanatory data analysis.

5. **Scikit-learn:** This implements various ML algorithms like Random Forest, Gradient Boosting and evaluation metrics such as confusion matrices and classification reports.

6. **TensorFlow/Keras:** It is used for building, training, and evaluation CNN, LSTM and Hybrid CNN-LSTM models for intrusion detection.

7. **Flask:** This hosts the web interface that allows users to upload datasets and interact with the pre-trained intrusion detection models.

8. **Joblib:** This saves and loads trained machine learning models for use in the Flask web application.



3 Installation

3.1 Software Installation

1. **Install Anaconda:** Download and install Anaconda Navigator which is recommended to manage environments and to run Jupyter Notebooks.

2. **Install Required Libraries:**

Importing Modules and Libraries

[1]:
```
# importing required libraries
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

import pickle
from os import path

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

Figure 1: Importing necessary libraries and modules in UNSW_Intrusion notebook

Imports and Data Setup

[7]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy.stats import shapiro
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
WARNING:tensorflow:From C:\Program Files\Anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Figure 2: Importing necessary libraries and modules in Cyber_Intrusion notebook

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.naive_bayes import GaussianNB

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn.metrics import confusion_matrix
```

Figure 3: Importing necessary libraries and modules in ToN_IoT notebook

```
[2]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import warnings
     warnings.filterwarnings('ignore')
```

Figure 4: Importing necessary libraries in Train_Model notebook

4 Dataset Preparation

4.1 Description of Datasets

1. **UNSW-NB15:** Network traffic daatset for multi-class intrusion detection.

Importing Datasets

```
[2]: data = pd.read_csv('C:/Users/shahn/OneDrive/Desktop/CODE_Intrusion_Detection_2024/datasets/UNSW_NB15.csv')
```

```
[3]: data.head(n=5)
```

[3]:

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	...	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm
0	1	0.121478	tcp	-	FIN	6	4	258	172	74.087490	...	1	1	0	0	0	
1	2	0.649902	tcp	-	FIN	14	38	734	42014	78.473372	...	1	2	0	0	0	
2	3	1.623129	tcp	-	FIN	8	16	364	13186	14.170161	...	1	3	0	0	0	
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	13.677108	...	1	3	1	1	0	
4	5	0.449454	tcp	-	FIN	10	6	534	268	33.373826	...	1	40	0	0	0	

5 rows × 45 columns

```
[4]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175341 entries, 0 to 175340
Data columns (total 45 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   id                 175341 non-null  int64
 1   dur                175341 non-null  float64
 2   proto              175341 non-null  object
 3   service            175341 non-null  object
 4   state              175341 non-null  object
 5   spkts              175341 non-null  int64
 6   dpkts              175341 non-null  int64
 7   sbytes             175341 non-null  int64
 8   dbytes             175341 non-null  int64
 9   rate               175341 non-null  float64
 10  sttl               175341 non-null  int64
 11  dttl               175341 non-null  int64
 12  sload              175341 non-null  float64
 13  dload              175341 non-null  float64
 14  sloss              175341 non-null  int64
 15  dloss              175341 non-null  int64
 16  sinpkt             175341 non-null  float64
 17  dinpkt             175341 non-null  float64
 18  sjit               175341 non-null  float64
 19  djit               175341 non-null  float64
 20  swin               175341 non-null  int64
 21  stcpb              175341 non-null  int64
 22  dtcpb              175341 non-null  int64
 23  dwin               175341 non-null  int64
 24  tcprtt             175341 non-null  float64
 25  synack             175341 non-null  float64
 26  ackdat             175341 non-null  float64
 27  smean              175341 non-null  int64
 28  dmean              175341 non-null  int64
 29  trans_depth        175341 non-null  int64
 30  response_body_len  175341 non-null  int64
 31  ct_srv_src         175341 non-null  int64
 32  ct_state_ttl       175341 non-null  int64
 33  ct_dst_ltm         175341 non-null  int64
 34  ct_src_dport_ltm   175341 non-null  int64
 35  ct_dst_sport_ltm   175341 non-null  int64
 36  ct_dst_src_ltm     175341 non-null  int64
 37  is_ftp_login       175341 non-null  int64
 38  ct_ftp_cmd         175341 non-null  int64
 39  ct_flw_http_mthd   175341 non-null  int64
 40  ct_src_ltm         175341 non-null  int64
 41  ct_srv_dst         175341 non-null  int64
 42  is_sm_ips_ports    175341 non-null  int64
 43  attack_cat         175341 non-null  object
 44  label              175341 non-null  int64
dtypes: float64(11), int64(30), object(4)
memory usage: 60.2+ MB
```

2. **NSL-KDD:** This is the improved version of existing KD'99 dataset which is widely used in various intrusion detection works.

[3]:
```
# open the .txt file here in the csv format
df = pd.read_csv('C:/Users/shahn/OneDrive/Desktop/CODE_Intrusion_Detection_2024/Jupyter_Code/NSL_KDD/KDDTrain+.csv',header=None)

# show all the columns
pd.set_option('display.max_columns', None)

# got from Kaggle NB
df.columns = ['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot'
,'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creations'
,'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','serror_rate'
,'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count'
,'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate'
,'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','outcome','level']
```

[4]:
```
print(df.shape)
df.head()
```

```
(125973, 43)
```

[4]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_att
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	
1	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	
2	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	
3	0	tcp	http	SF	232	8153	0	0	0	0	0	1	0	0	
4	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	

3. **ToN_IoT:** This is an IoT_specific dtatset with multiple attack types.

[5]:
```
fridge = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Fridge.csv")
garage_door = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Garage_Door.csv")
gps = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_GPS_Tracker.csv")
modbus = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Modbus.csv")
motion_light = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Motion_Light.csv")
thermostat = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Thermostat.csv")
weather = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Weather.csv")
```

[4]:
```
fridge.head()
```

[4]:

	date	time	fridge_temperature	temp_condition	label	type
0	31-Mar-19	12:36:52	13.10	high	0	normal
1	31-Mar-19	12:36:53	8.65	high	0	normal
2	31-Mar-19	12:36:54	2.00	low	0	normal
3	31-Mar-19	12:36:55	4.80	low	0	normal
4	31-Mar-19	12:36:56	10.70	high	0	normal

```
[7]: fridge.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 587076 entries, 0 to 587075
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   date                587076 non-null  object
 1   time                587076 non-null  object
 2   fridge_temperature  587076 non-null  float64
 3   temp_condition      587076 non-null  object
 4   label               587076 non-null  int64
 5   type                587076 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 26.9+ MB
```

All the datasets to be placed in a "/datasets" directory.

5 Implementation

5.1 Implementation of Cyber_Intrusion

1. The preprocessing of UNSW-NB15 dataset and generation of heatmaps for feature correlation is performed.
2. Load "Cyber_Intrusion.ipynb" in Jupyter Notebook.
3. Execute preprocessing cells sequentially.
4. Heatmaps for correlation analysis.
5. Preprocessed dataset saved as processed_unsw.csv

Data Preprocessing

[10]:
```
# save attribute and label strings
all_attribute_names = intru_table.columns.drop('label')

intru_label_names = ['Backdoor_Malware', 'BenignTraffic', 'BrowserHijacking', 'CommandInjection', 'DDoS-ACK_Fragmentation', 'DDoS-HTTP_Flood', 'DDoS-ICMP
                     'DDoS-PSHACK_Flood', 'DDoS-RSTFINFlood', 'DDoS-SYN_Flood', 'DDoS-SlowLoris', 'DDoS-SynonymousIP_Flood', 'DDoS-TCP_Flood', 'DDoS-UDP_
                     'DNS_Spoofing', 'DictionaryBruteForce', 'DoS-HTTP_Flood', 'DoS-SYN_Flood', 'DoS-TCP_Flood', 'DoS-UDP_Flood', 'MITM-ArpSpoofing', 'Mi
                     'Mirai-udpplain', 'Recon-HostDiscovery', 'Recon-OSScan', 'Recon-PingSweep', 'Recon-PortScan', 'SqlInjection', 'Uploading_Attack', 'V
```

[11]:
```
# complete one-hot encoding on intrusion labels
ohe = OneHotEncoder()
ohe_intru_labels = pd.DataFrame(ohe.fit_transform(intru_table['label'].values.reshape(-1,1)).toarray(), columns=intru_label_names)
```

[12]:
```
# we decided to remove the following attributes since they provide no meaningful contributions to the model
exclude = ['ece_flag_number', 'cwr_flag_number', 'SMTP', 'Telnet', 'IRC', 'Tot sum', 'Min', 'Max', 'AVG', 'Std', 'Tot size', 'Covariance', 'Variance']

relevant = ['flow_duration', 'Header_Length', 'Protocol Type', 'Duration', 'Rate', 'Srate', 'Drate',
        'syn_count', 'fin_count', 'urg_count', 'rst_count', 'IPv', 'LLC', 'DNS', 'SSH', 'TCP', 'UDP',
        'DHCP', 'ARP', 'ICMP', 'IAT', 'Number', 'Mangitue', 'Radius', 'Weight']

# remove unwanted data attribute columns
attribute_data = intru_labels_removed.drop(columns=exclude)

# split data for all model runs
X_train, X_test, y_train, y_test = train_test_split(attribute_data, ohe_intru_labels, train_size=0.9, random_state=5)
```

Figure 5: Data Preprocessing of Cyber_Intrusion

[13]:
```
# check if each attribute is normally distributed
for col_name in attribute_data:
    print(col_name, shapiro(attribute_data[col_name]), "\n\tmin:", np.min(attribute_data[col_name]), "; max:", np.max(attribute_data[col_name]))
```

```
flow_duration ShapiroResult(statistic=0.003912689874606579, pvalue=3.287366402848467e-239)
        min: 0.0 ; max: 99435.76178
Header_Length ShapiroResult(statistic=0.1559501272391252 7, pvalue=1.657367141034325e-232)
        min: 0.0 ; max: 9815555.0
Protocol Type ShapiroResult(statistic=0.6352051479259249, pvalue=4.925866299250941e-200)
        min: 0.0 ; max: 47.0
Duration ShapiroResult(statistic=0.18466471163268572, pvalue=4.0550473173632e-231)
        min: 0.0 ; max: 255.0
Rate ShapiroResult(statistic=0.05924111192858483, pvalue=6.924870220670144e-237)
        min: 0.0 ; max: 7340032.0
Srate ShapiroResult(statistic=0.05924111192858483, pvalue=6.924870220670144e-237)
        min: 0.0 ; max: 7340032.0
Drate ShapiroResult(statistic=9.673064369997153e-05, pvalue=2.2960230502648243e-239)
        min: 0.0 ; max: 0.848465429
fin_flag_number ShapiroResult(statistic=0.3146954186658635, pvalue=3.2703875468487175e-224)
        min: 0 ; max: 1
syn_flag_number ShapiroResult(statistic=0.49781337233979617, pvalue=4.072253047902139e-212)
        min: 0 ; max: 1
rst_flag_number ShapiroResult(statistic=0.3231803133049236, pvalue=1.0134578413708053e-223)
        min: 0 ; max: 1
psh_flag_number ShapiroResult(statistic=0.31798828376526755, pvalue=5.064864595949719e-224)
        min: 0 ; max: 1
ack_flag_number ShapiroResult(statistic=0.3855051669308578, pvalue=6.25496855367773e-220)
        min: 0 ; max: 1
ack_count ShapiroResult(statistic=0.3320400435897358, pvalue=3.3476254818623606e-223)
        min: 0.0 ; max: 4.0
syn_count ShapiroResult(statistic=0.5663950769094286, pvalue=1.602892806221714e-206)
        min: 0.0 ; max: 9.69
fin_count ShapiroResult(statistic=0.32056961653993765, pvalue=7.146160098971559e-224)
        min: 0.0 ; max: 77.6
urg_count ShapiroResult(statistic=0.05756365903671634, pvalue=5.863420209241335e-237)
        min: 0.0 ; max: 4136.7
rst_count ShapiroResult(statistic=0.09728359740844761, pvalue=3.248066022733782e-235)
        min: 0.0 ; max: 9331.5
HTTP ShapiroResult(statistic=0.2195085625971428, pvalue=2.260465979889006e-229)
        min: 0 ; max: 1
HTTPS ShapiroResult(statistic=0.23880164156491357, pvalue=2.2485706436126917e-228)
        min: 0 ; max: 1
DNS ShapiroResult(statistic=0.002044631451050938, pvalue=2.7572413394210803e-239)
```

[14]:
```
# attributes are NOT normally distributed (all p-values < 0.05) - normalize traffic attribute data in train and test sets
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=attribute_data.columns)
X_test = pd.DataFrame(scaler.fit_transform(X_test), columns=attribute_data.columns)
```

Data Visualization

```
[15]: # show output class distribution
classes_true_count = []
for col in ohe_intru_labels:
    count = (ohe_intru_labels[col] == 1).sum()
    classes_true_count.append(count)
class_distru = pd.DataFrame({'Intrusion Output Class': ohe_intru_labels.columns, 'Number of True Occurences in Dataset': classes_true_count})
fig, ax = plt.subplots()
sns.barplot(x='Number of True Occurences in Dataset', y='Intrusion Output Class', data=class_distru)
for i in ax.containers:
    ax.bar_label(i, fontsize=7)
```

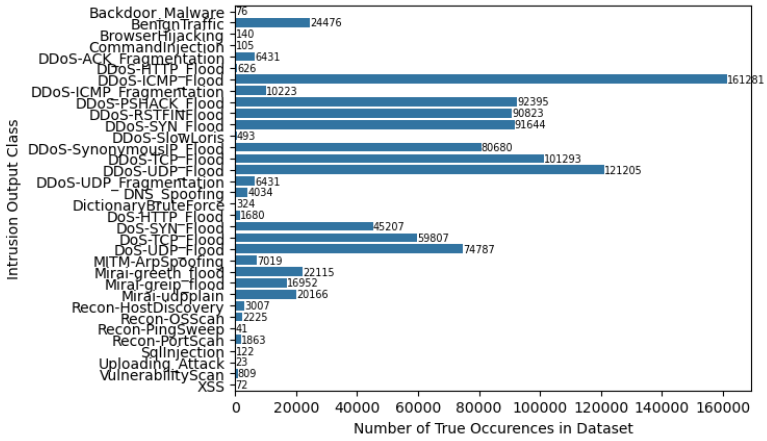


Figure 6: Visualization of class distribution

Fitting the Initial Model and Measuring Its Performance

```
[17]: # setup neural net for predicting intrusion class
classifier = Sequential()
classifier.add(Dense(units = 36, activation = 'relu', input_dim = 33))
classifier.add(Dense(units = 35, activation = 'relu'))
classifier.add(Dense(units = 34, activation = 'softmax'))
classifier.compile(optimizer = "SGD", loss = 'CategoricalCrossentropy')
classifier.fit(X_train.astype(float), y_train, epochs = 50)
```

```
[18]: # INITIAL STATS
# measure performance on training data
intru_predictions = pd.DataFrame(((classifier.predict(X_train.astype(float))) > 0.5).astype(int), columns=ohe_intru_labels.columns)
print("Model Training Accuracy (avg.):", accuracy_score(y_train, intru_predictions))
print("Model Training Precision (avg.):", precision_score(y_train, intru_predictions, average='macro'))
print("Model Training Recall (avg.):", recall_score(y_train, intru_predictions, average='macro'))
initial_MSE_train = mean_squared_error(y_train, intru_predictions)
print("Model Training MSE:", initial_MSE_train)
print(classification_report(y_train, intru_predictions, target_names=ohe_intru_labels.columns))

# measure performance on testing data
intru_predictions = pd.DataFrame(((classifier.predict(X_test.astype(float))) > 0.5).astype(int), columns=ohe_intru_labels.columns)
print("Model Testing Accuracy (avg.):", accuracy_score(y_test, intru_predictions))
print("Model Testing Precision (avg.):", precision_score(y_test, intru_predictions, average='macro'))
print("Model Testing Recall (avg.):", recall_score(y_test, intru_predictions, average='macro'))
initial_MSE_test = mean_squared_error(y_test, intru_predictions)
print("Model Testing MSE:", initial_MSE_test)
print(classification_report(y_test, intru_predictions, target_names=ohe_intru_labels.columns))
```

```
29492/29492 [==============================] - 101s 3ms/step
Model Training Accuracy (avg.): 0.7859835098869682
Model Training Precision (avg.): 0.6106376168257833
Model Training Recall (avg.): 0.46039651525372266
Model Training MSE: 0.011884046245419163
                        precision    recall  f1-score   support

       Backdoor_Malware       0.00      0.00      0.00        68
          BenignTraffic       0.83      0.73      0.78     22033
       BrowserHijacking       0.00      0.00      0.00       126
       CommandInjection       0.00      0.00      0.00        88
 DDoS-ACK_Fragmentation       0.98      0.98      0.98      5796
        DDoS-HTTP_Flood       0.70      0.51      0.59       579
        DDoS-ICMP_Flood       1.00      1.00      1.00    145241
DDoS-ICMP_Fragmentation       1.00      0.97      0.99      9235
      DDoS-PSHACK_Flood       1.00      1.00      1.00     83181
       DDoS-RSTFINFlood       1.00      1.00      1.00     81774
         DDoS-SYN_Flood       0.67      0.94      0.78     82578
         DDoS-SlowLoris       0.93      0.03      0.06       448
DDoS-SynonymousIP_Flood       0.93      0.53      0.67     72679
         DDoS-TCP_Flood       0.64      0.97      0.77     90906
         DDoS-UDP_Flood       0.71      0.95      0.82    109148
 DDoS-UDP_Fragmentation       0.99      0.98      0.98      5782
           DNS_Spoofing       0.48      0.21      0.30      3611
   DictionaryBruteForce       0.00      0.00      0.00       294
        DoS-HTTP_Flood       0.85      0.61      0.71      1482
         DoS-SYN_Flood       0.55      0.44      0.49     40693
         DoS-TCP_Flood       0.59      0.08      0.14     53793
         DoS-UDP_Flood       0.83      0.38      0.52     67239
      MITM-ArpSpoofing       0.82      0.51      0.63      6292
    Mirai-greeth_flood       0.76      0.04      0.08     19923
     Mirai-greip_flood       0.44      0.97      0.60     15262
       Mirai-udpplain       1.00      0.99      0.99     18125
   Recon-HostDiscovery       0.71      0.42      0.53      2682
          Recon-OSScan       1.00      0.00      0.00      2010
        Recon-PingSweep       0.00      0.00      0.00        36
         Recon-PortScan       0.53      0.04      0.07      1673
           SqlInjection       0.00      0.00      0.00       114
      Uploading_Attack       0.00      0.00      0.00        20
     VulnerabilityScan       0.81      0.39      0.52       746
                   XSS       0.00      0.00      0.00        60

             micro avg       0.81      0.79      0.80    943717
             macro avg       0.61      0.46      0.47    943717
          weighted avg       0.82      0.79      0.76    943717
           samples avg       0.79      0.79      0.79    943717
```

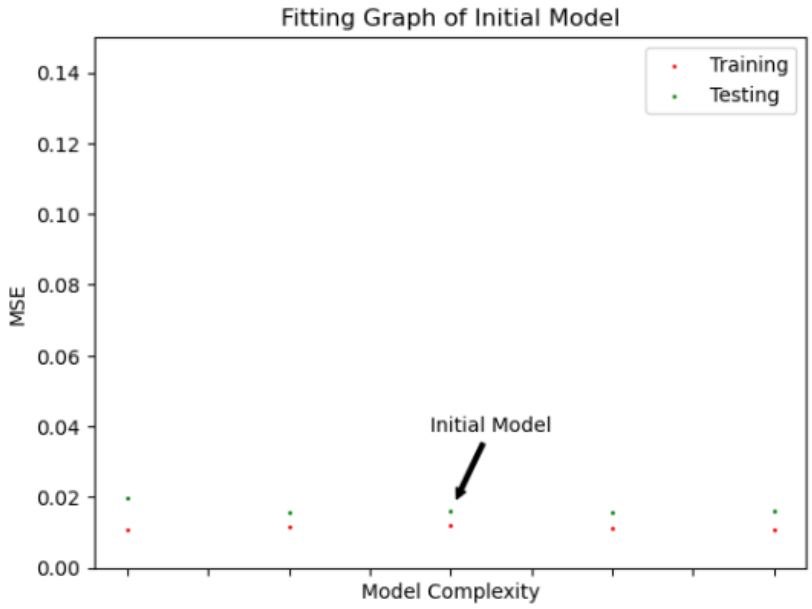
```
3277/3277 [==============================] - 10s 3ms/step
Model Testing Accuracy (avg.): 0.7177897728356444
Model Testing Precision (avg.): 0.5214589950343226
Model Testing Recall (avg.): 0.4087378332241005
Model Testing MSE: 0.01591003183016135
```

	precision	recall	f1-score	support
Backdoor_Malware	0.00	0.00	0.00	8
BenignTraffic	0.84	0.60	0.70	2443
BrowserHijacking	0.00	0.00	0.00	14
CommandInjection	0.00	0.00	0.00	17
DDoS-ACK_Fragmentation	0.94	0.99	0.96	635
DDoS-HTTP_Flood	0.68	0.40	0.51	47
DDoS-ICMP_Flood	1.00	1.00	1.00	16040
DDoS-ICMP_Fragmentation	1.00	0.97	0.98	988
DDoS-PSHACK_Flood	1.00	1.00	1.00	9214
DDoS-RSTFINFlood	1.00	1.00	1.00	9049
DDoS-SYN_Flood	0.89	0.01	0.01	9066
DDoS-SlowLoris	0.00	0.00	0.00	45
DDoS-SynonymousIP_Flood	0.38	0.94	0.54	8001
DDoS-TCP_Flood	0.64	0.99	0.78	10387
DDoS-UDP_Flood	0.71	0.96	0.82	12057
DDoS-UDP_Fragmentation	0.55	0.98	0.71	649
DNS_Spoofing	0.45	0.18	0.26	423
DictionaryBruteForce	0.00	0.00	0.00	30
DoS-HTTP_Flood	0.75	0.74	0.74	198
DoS-SYN_Flood	0.36	0.11	0.17	4514
DoS-TCP_Flood	0.71	0.02	0.04	6014
DoS-UDP_Flood	0.86	0.38	0.52	7548
MITM-ArpSpoofing	0.78	0.47	0.59	727
Mirai-greeth_flood	0.57	0.99	0.72	2192
Mirai-greip_flood	0.50	0.00	0.00	1690
Mirai-udpplain	0.99	0.74	0.85	2041
Recon-HostDiscovery	0.76	0.25	0.38	325
Recon-OSScan	0.29	0.01	0.02	215
Recon-PingSweep	0.00	0.00	0.00	5
Recon-PortScan	0.48	0.07	0.12	190
SqlInjection	0.00	0.00	0.00	8
Uploading_Attack	0.00	0.00	0.00	3
VulnerabilityScan	0.60	0.10	0.16	63
XSS	0.00	0.00	0.00	12
micro avg	0.74	0.72	0.73	104858
macro avg	0.52	0.41	0.40	104858
weighted avg	0.79	0.72	0.66	104858
samples avg	0.72	0.72	0.72	104858

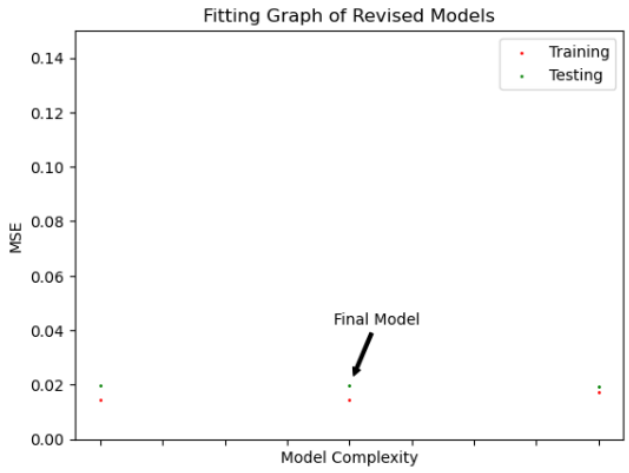
[19]:
```
# K-fold cross validation to generalize initial model performance (10 folds)
classifier = Sequential()
classifier.add(Dense(units = 36, activation = 'relu', input_dim = 33))
classifier.add(Dense(units = 35, activation = 'relu'))
classifier.add(Dense(units = 34, activation = 'softmax'))
classifier.compile(optimizer = "SGD", loss = 'CategoricalCrossentropy')
kfold_model = KFold(n_splits=10)

# calculate accuracy and MSE scores for each fold iteration
scaler = MinMaxScaler()
X = scaler.fit_transform(attribute_data)
y = ohe_intru_labels.values
all_accuracy = []
all_mse = []
count = 0
for train_index, test_index in kfold_model.split(X):
    count += 1
    classifier.fit(X[train_index].astype(float), y[train_index])
    yhat_pred_test = pd.DataFrame(((classifier.predict(X[test_index].astype(float))) > 0.5).astype(int))
    iter_accuracy = accuracy_score(y[test_index], yhat_pred_test)
    print("\tIteration", count, "Accuracy:", iter_accuracy)
    all_accuracy.append(iter_accuracy)
    iter_mse = mean_squared_error(y[test_index], yhat_pred_test)
    print("\tIteration", count, "MSE:", iter_mse)
    all_mse.append(iter_mse)

# print results
print("Overall Avg. K-Fold Accuracy:", np.mean(all_accuracy))
print("Overall Avg. K-Fold MSE:", np.mean(all_mse))
```

```
29492/29492 [==============================] - 107s 4ms/step - loss: 0.7231
3277/3277 [==============================] - 11s 3ms/step
        Iteration 1 Accuracy: 0.6926033302180091
        Iteration 1 MSE: 0.016283085360257514
29492/29492 [==============================] - 103s 3ms/step - loss: 0.5284
3277/3277 [==============================] - 11s 3ms/step
        Iteration 2 Accuracy: 0.767609529077419
        Iteration 2 MSE: 0.012699808031702257
29492/29492 [==============================] - 105s 4ms/step - loss: 0.4952
3277/3277 [==============================] - 11s 3ms/step
        Iteration 3 Accuracy: 0.7688874477865304
        Iteration 3 MSE: 0.012645112213379888
29492/29492 [==============================] - 103s 3ms/step - loss: 0.4837
3277/3277 [==============================] - 10s 3ms/step
        Iteration 4 Accuracy: 0.7648343474031547
        Iteration 4 MSE: 0.012716076531510963
29492/29492 [==============================] - 104s 4ms/step - loss: 0.4776
3277/3277 [==============================] - 11s 3ms/step
        Iteration 5 Accuracy: 0.6997463235995346
        Iteration 5 MSE: 0.013544928547626877
29492/29492 [==============================] - 102s 3ms/step - loss: 0.4732
3277/3277 [==============================] - 11s 3ms/step
        Iteration 6 Accuracy: 0.7747694479147792
        Iteration 6 MSE: 0.012402324959089944
29492/29492 [==============================] - 103s 3ms/step - loss: 0.4700
3277/3277 [==============================] - 11s 3ms/step
        Iteration 7 Accuracy: 0.7107584615237895
        Iteration 7 MSE: 0.013231184879799887
29492/29492 [==============================] - 102s 3ms/step - loss: 0.4673
3277/3277 [==============================] - 10s 3ms/step
        Iteration 8 Accuracy: 0.7731481922999895
        Iteration 8 MSE: 0.012627842176095288
29492/29492 [==============================] - 102s 3ms/step - loss: 0.4655
3277/3277 [==============================] - 11s 3ms/step
        Iteration 9 Accuracy: 0.7824084228997588
        Iteration 9 MSE: 0.012148758336984432
29492/29492 [==============================] - 106s 4ms/step - loss: 0.4629
3277/3277 [==============================] - 10s 3ms/step
        Iteration 10 Accuracy: 0.7897708307504506
        Iteration 10 MSE: 0.011722407379461888
Overall Avg. K-Fold Accuracy: 0.7524536333473415
Overall Avg. K-Fold MSE: 0.013002152841590894
```

[28]:
```
complexities = [1, 2, 3, 4, 5]
train_mse = [low1_MSE_train, low2_MSE_train, initial_MSE_train, high1_MSE_train, high2_MSE_train]
test_mse = [low1_MSE_test, low2_MSE_test, initial_MSE_test, high1_MSE_test, high2_MSE_test]
plt.scatter(complexities, train_mse, s = 1, c='r', label='Training')
plt.scatter(complexities, test_mse, s = 1, c='g', label='Testing')
plt.xlabel('Model Complexity')
plt.xticks(visible=False)
plt.ylim(0, 0.15)
plt.ylabel('MSE')
plt.title('Fitting Graph of Initial Model')
plt.annotate('Initial Model', (complexities[2], test_mse[2]), textcoords='offset points', xytext=(-10, 40),
             arrowprops=dict(arrowstyle='simple', facecolor='black', shrinkB=7))
plt.legend()
plt.show()
```



[41]:
```
complexities = [1, 2, 3]
# Revision2, Revision3, Revision4 (final)
train_mse = [0.014385329438616347, 0.014464414371750703, 0.017478791180121554]
test_mse = [0.019767811704834607, 0.01966991800961266, 0.019265973989256432]
plt.scatter(complexities, train_mse, s = 1, c='r', label='Training')
plt.scatter(complexities, test_mse, s = 1, c='g', label='Testing')
plt.xlabel('Model Complexity')
plt.xticks(visible=False)
plt.ylim(0, 0.15)
plt.ylabel('MSE')
plt.title('Fitting Graph of Revised Models')
plt.annotate('Final Model', (complexities[1], test_mse[1]), textcoords='offset points', xytext=(-10, 40),
             arrowprops=dict(arrowstyle='simple', facecolor='black', shrinkB=7))
plt.legend()
plt.show()
```



5.2 Implementation of PPGO_IoT

1. The evaluation of IoT-specific datasets using various ML classifiers are done here.
2. Load datasets such as GPS Tracker, Fridge, and Thermostat.
3. Train models such as Random Forest and Decision Trees.
4. Classification reports for each dataset.
5. Accuracy comparison for classifiers.

```
[28]: import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Define PPGO algorithm
def ppgo_initialize(num_parameters):
    # Initialize parameters randomly or using some strategy
    initial_parameters = np.random.randn(num_parameters)  # Example: Random initialization
    return initial_parameters

def ppgo_update(current_parameters):
    # Implement parameter update logic based on PPGO algorithm
    # This could involve various operations such as mutation, crossover, etc.
    updated_parameters = current_parameters + np.random.randn(len(current_parameters)) * 0.1  # Example: Random update
    return updated_parameters

def perpetual_pigeon_galvanized_optimization(objective_function, initial_parameters, num_iterations):
    current_parameters = initial_parameters
    for i in range(num_iterations):
        updated_parameters = ppgo_update(current_parameters)
        loss = objective_function(updated_parameters)
        print(f"Iteration {i+1}, Loss: {loss}")
        current_parameters = updated_parameters
```

Figure 7: Optimization using PPGO algorithm

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Encode target labels using one-hot encoding
label_encoder = LabelEncoder()
y_train_encoded = to_categorical(label_encoder.fit_transform(y_train))
y_test_encoded = to_categorical(label_encoder.transform(y_test))
```

Figure 8: Splitting the dataset into training and testing

```python
# Define LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(units=num_classes, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Get the shapes of model weights
shapes = [w.shape for w in model.get_weights()]

# Run PPGO optimization
num_parameters = sum(np.prod(shape) for shape in shapes)  # Number of parameters in the model
initial_parameters = ppgo_initialize(num_parameters)

def objective_function(parameters):
    # Reshape parameters to match the shapes of model weights
    weights = []
    start = 0
    for shape in shapes:
        weight_size = np.prod(shape)
        weight = parameters[start:start+weight_size].reshape(shape)
        weights.append(weight)
        start += weight_size
    model.set_weights(weights)
    loss, _ = model.evaluate(X_test_lstm, y_test_encoded, verbose=0)
    return loss

perpetual_pigeon_galvanized_optimization(objective_function, initial_parameters, num_iterations=100)

# Evaluate the model with the optimized weights
loss, accuracy = model.evaluate(X_test_lstm, y_test_encoded)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

Figure 9: Deep Learning with LSTM

5.3 Implementation of ToN_IoT

1. Feature Selection and ML model evaluation for ToN_IoT dataset is done.
2. Used correlation matrices for selection of key features.
3. Train models like Logistic Regression and SVM.
4. Feature importance visualizations.
5. Confusion matrices for model predictions.

[5]:
```
fridge = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Fridge.csv")
garage_door = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Garage_Door.csv")
gps = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_GPS_Tracker.csv")
modbus = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Modbus.csv")
motion_light = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Motion_Light.csv")
thermostat = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Thermostat.csv")
weather = pd.read_csv(r"C:\Users\shahn\OneDrive\Desktop\CODE_Intrusion_Detection_2024\TON_IOT\IoT_Weather.csv")
```

[4]:
```
fridge.head()
```

[4]:

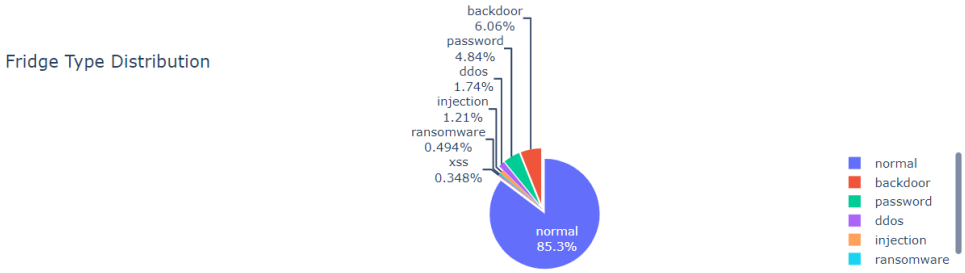
	date	time	fridge_temperature	temp_condition	label	type
0	31-Mar-19	12:36:52	13.10	high	0	normal
1	31-Mar-19	12:36:53	8.65	high	0	normal
2	31-Mar-19	12:36:54	2.00	low	0	normal
3	31-Mar-19	12:36:55	4.80	low	0	normal
4	31-Mar-19	12:36:56	10.70	high	0	normal

Figure 10: Preprocessing of ToN_IoT notebook

```
fridge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 587076 entries, 0 to 587075
Data columns (total 6 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   date                 587076 non-null  object
 1   time                 587076 non-null  object
 2   fridge_temperature   587076 non-null  float64
 3   temp_condition       587076 non-null  object
 4   label                587076 non-null  int64
 5   type                 587076 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 26.9+ MB
```

[8]:
```
type_counts = fridge['type'].value_counts()

fig = go.Figure(data=[go.Pie(labels=type_counts.index, values=type_counts)])
fig.update_traces(textinfo='percent+label', pull=[0.1, 0.1, 0.1, 0.1])
fig.update_layout(title='Fridge Type Distribution')

fig.show()
```



[11]:
```
fridge['temp_condition'] = fridge['temp_condition'].apply(lambda x:x.strip())
fridge['time'] = fridge['time'].apply(lambda x: x.strip())
```

[12]:
```
# number of seconds that have passed since midnight

def time_to_seconds(time_str):
    h, m, s = map(int, time_str.split(':'))
    return h * 3600 + m * 60 + s

fridge['time_seconds'] = fridge['time'].apply(time_to_seconds)
# target variable + features
X = fridge.drop(['date', 'time', 'label', 'type'], axis=1)
y = fridge['type']
```

[13]:
```
# encoding categorical variables
label_encoder = LabelEncoder()
X['temp_condition'] = label_encoder.fit_transform(X['temp_condition'])
# training + testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train
```

[13]:

	fridge_temperature	temp_condition	time_seconds
189639	7.65	0	15495
304660	1.00	1	35883
408687	4.95	1	74706
460624	11.25	0	45642
197265	5.05	1	15525
...	...	...	...
110268	6.95	0	15108
259178	5.85	1	76677
365838	8.55	0	10669
131932	7.10	0	15229
121958	10.55	0	15175

469660 rows × 3 columns

[20]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
k = 5  # Choose an appropriate value for K
knn_classifier = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
knn_classifier.fit(X_train_scaled, y_train)
y_pred = knn_classifier.predict(X_test_scaled)
```

[14]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.7756694147305307
```

[15]:
```
# Get classification report
classification_rep = classification_report(y_test, dt_predictions)

# Print classification report
print("Decision Tree Classification Report:")
print(classification_rep)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

    backdoor       0.12      0.12      0.12      7192
        ddos       0.08      0.07      0.08      2090
   injection       0.22      0.24      0.23      1415
      normal       0.88      0.89      0.88    100027
    password       0.19      0.17      0.18      5707
  ransomware       0.12      0.13      0.13       577
         xss       0.14      0.14      0.14       408

    accuracy                           0.78    117416
   macro avg       0.25      0.25      0.25    117416
weighted avg       0.77      0.78      0.77    117416
```

Figure 11: Decision Tree of fridge dataset

[16]:
```
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest Accuracy: 0.7855317844246099
```

[17]:
```
classification_rep = classification_report(y_test, rf_predictions)

# Print classification report
print("Random Forest Classifier Report:")
print(classification_rep)
```

```
Random Forest Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.13      0.11      0.12      7192
        ddos       0.07      0.06      0.06      2090
   injection       0.23      0.25      0.24      1415
      normal       0.88      0.90      0.89    100027
    password       0.18      0.15      0.16      5707
  ransomware       0.13      0.11      0.12       577
         xss       0.15      0.13      0.14       408

    accuracy                           0.79    117416
   macro avg       0.25      0.24      0.25    117416
weighted avg       0.77      0.79      0.78    117416
```

Figure 12: Random Forest of fridge dataset

[18]:
```
gb_classifier = GradientBoostingClassifier(random_state=42)
gb_classifier.fit(X_train, y_train)
gb_predictions = gb_classifier.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_predictions)
print("Gradient Boosting Accuracy:", gb_accuracy)
```

```
Gradient Boosting Accuracy: 0.8518941200517817
```

[19]:
```
classification_rep = classification_report(y_test, gb_predictions)

# Print classification report
print("Gradient Boosting Classifier Report:")
print(classification_rep)
```

```
Gradient Boosting Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7192
        ddos       1.00      0.00      0.00      2090
   injection       0.00      0.00      0.00      1415
      normal       0.85      1.00      0.92    100027
    password       1.00      0.00      0.00      5707
  ransomware       0.00      0.00      0.00       577
         xss       0.00      0.00      0.00       408

    accuracy                           0.85    117416
   macro avg       0.41      0.14      0.13    117416
weighted avg       0.79      0.85      0.78    117416
```

Figure 13: Gradient Boosting of fridge dataset

[24]:
```
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print("Naive Bayes Accuracy:", nb_accuracy)
```

```
Naive Bayes Accuracy: 0.8519026367786332
```

[25]:
```
classification_rep = classification_report(y_test, nb_predictions)

# Print classification report
print("Naive Bayes Classifier Report:")
print(classification_rep)
```

```
Naive Bayes Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7192
        ddos       0.00      0.00      0.00      2090
   injection       0.00      0.00      0.00      1415
      normal       0.85      1.00      0.92    100027
    password       0.00      0.00      0.00      5707
  ransomware       0.00      0.00      0.00       577
         xss       0.00      0.00      0.00       408

    accuracy                           0.85    117416
   macro avg       0.12      0.14      0.13    117416
weighted avg       0.73      0.85      0.78    117416
```

Figure 14: Naive Bayes of fridge dataset

[26]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda_classifier = LinearDiscriminantAnalysis()
lda_classifier.fit(X_train, y_train)
lda_predictions = lda_classifier.predict(X_test)
lda_accuracy = accuracy_score(y_test, lda_predictions)
print("Linear Discriminant Analysis Accuracy:", lda_accuracy)
```

```
Linear Discriminant Analysis Accuracy: 0.8519026367786332
```

[27]:
```
classification_rep = classification_report(y_test, lda_predictions)

# Print classification report
print("Linear Discriminant Analysis Report:")
print(classification_rep)
```

```
Linear Discriminant Analysis Report:
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7192
        ddos       0.00      0.00      0.00      2090
   injection       0.00      0.00      0.00      1415
      normal       0.85      1.00      0.92    100027
    password       0.00      0.00      0.00      5707
  ransomware       0.00      0.00      0.00       577
         xss       0.00      0.00      0.00       408

    accuracy                           0.85    117416
   macro avg       0.12      0.14      0.13    117416
weighted avg       0.73      0.85      0.78    117416
```

Figure 15: LDA of fridge dataset

[28]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
qda_classifier = QuadraticDiscriminantAnalysis()
qda_classifier.fit(X_train, y_train)
qda_predictions = qda_classifier.predict(X_test)
qda_accuracy = accuracy_score(y_test, qda_predictions)
print("Quadratic Discriminant Analysis Accuracy:", qda_accuracy)
```

```
Quadratic Discriminant Analysis Accuracy: 0.8519026367786332
```

[29]:
```
classification_rep = classification_report(y_test, qda_predictions)

# Print classification report
print("Quadratic Discriminant Analysis Report:")
print(classification_rep)
```

```
Quadratic Discriminant Analysis Report:
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7192
        ddos       0.00      0.00      0.00      2090
   injection       0.00      0.00      0.00      1415
      normal       0.85      1.00      0.92    100027
    password       0.00      0.00      0.00      5707
  ransomware       0.00      0.00      0.00       577
         xss       0.00      0.00      0.00       408

    accuracy                           0.85    117416
   macro avg       0.12      0.14      0.13    117416
weighted avg       0.73      0.85      0.78    117416
```

Figure 16: QDA of fridge dataset

[30]:
```
garage_door.head()
```

[30]:

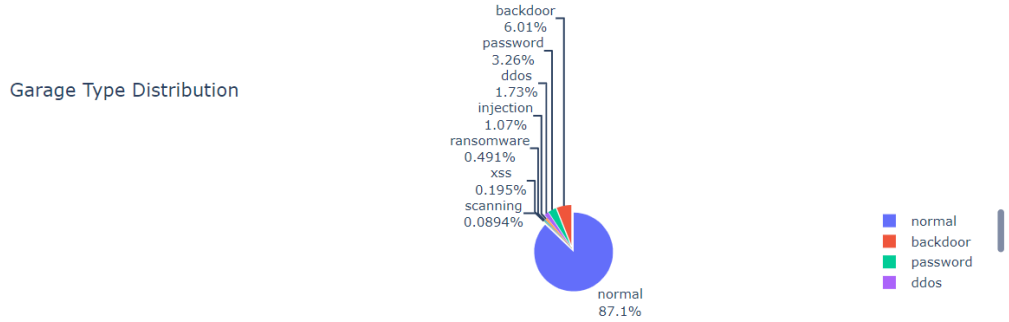
	date	time	door_state	sphone_signal	label	type
0	1-Apr-19	20:53:44	open	true	0	normal
1	1-Apr-19	20:53:49	closed	false	0	normal
2	1-Apr-19	20:53:49	open	true	0	normal
3	1-Apr-19	20:53:54	closed	false	0	normal
4	1-Apr-19	20:53:54	open	true	0	normal

[31]:
```
garage_door.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 591446 entries, 0 to 591445
Data columns (total 6 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   date           570111 non-null  object
 1   time           570111 non-null  object
 2   door_state     541177 non-null  object
 3   sphone_signal  541177 non-null  object
 4   label          591446 non-null  int64
 5   type           591446 non-null  object
dtypes: int64(1), object(5)
memory usage: 27.1+ MB
```

Figure 17: Preprocessing of garage_door dataset

[32]:
```
type_counts = garage_door['type'].value_counts()

fig = go.Figure(data=[go.Pie(labels=type_counts.index, values=type_counts)])
fig.update_traces(textinfo='percent+label', pull=[0.1, 0.1, 0.1, 0.1])
fig.update_layout(title='Garage Type Distribution')

fig.show()
garage_door['label'].value_counts()
```



[32]:
```
label
0    515443
1     76003
Name: count, dtype: int64
```

Figure 18: Distribution of garage_door dataset

[37]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.8651365288697269
```

[38]:
```
# Get classification report
classification_rep = classification_report(y_test, dt_predictions)

# Print classification report
print("Decision Tree Classification Report:")
print(classification_rep)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

    backdoor       0.36      0.37      0.36      7228
        ddos       0.31      0.36      0.34      2032
   injection       0.19      0.21      0.20      1199
      normal       0.94      0.94      0.94    103123
    password       0.42      0.38      0.40      3792
  ransomware       0.15      0.12      0.14       577
    scanning       0.00      0.00      0.00       114
         xss       0.44      0.48      0.46       225

    accuracy                           0.87    118290
   macro avg       0.35      0.36      0.35    118290
weighted avg       0.87      0.87      0.87    118290
```

Figure 19: Decision Tree of garage_door dataset

[39]:
```
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest Accuracy: 0.8735818750528362
```

[40]:
```
# Get classification report
classification_rep = classification_report(y_test, rf_predictions)

# Print classification report
print("Random Forest Classification Report:")
print(classification_rep)
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

    backdoor       0.44      0.34      0.38      7228
        ddos       0.31      0.31      0.31      2032
   injection       0.19      0.19      0.19      1199
      normal       0.94      0.95      0.94    103123
    password       0.44      0.51      0.47      3792
  ransomware       0.19      0.16      0.18       577
    scanning       0.00      0.00      0.00       114
         xss       0.44      0.56      0.49       225

    accuracy                           0.87    118290
   macro avg       0.37      0.38      0.37    118290
weighted avg       0.87      0.87      0.87    118290
```

Figure 20: Random Forest of garage_door dataset

[41]:
```
gb_classifier = GradientBoostingClassifier(random_state=42)
gb_classifier.fit(X_train, y_train)
gb_predictions = gb_classifier.predict(X_test)
gb_accuracy = accuracy_score(y_test, gb_predictions)
print("Gradient Boosting Accuracy:", gb_accuracy)
```

```
Gradient Boosting Accuracy: 0.8926198326147603
```

[42]:
```
# Get classification report
classification_rep = classification_report(y_test, gb_predictions)

# Print classification report
print("Random Forest Classification Report:")
print(classification_rep)
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

    backdoor       0.90      0.28      0.43      7228
        ddos       0.33      0.00      0.00      2032
   injection       0.41      0.07      0.11      1199
      normal       0.93      0.98      0.95    103123
    password       0.37      0.74      0.49      3792
  ransomware       0.00      0.00      0.00       577
    scanning       0.00      0.00      0.00       114
         xss       0.50      0.01      0.03       225

    accuracy                           0.89    118290
   macro avg       0.43      0.26      0.25    118290
weighted avg       0.89      0.89      0.87    118290
```

Figure 21: Gradient Boosting of garage_door dataset

[44]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
k = 5  # Choose an appropriate value for K
knn_classifier = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
knn_classifier.fit(X_train_scaled, y_train)
y_pred = knn_classifier.predict(X_test_scaled)
knn_accuracy = accuracy_score(y_test, y_pred)
print("KNNs Accuracy:", knn_accuracy)
```

```
KNNs Accuracy: 0.8611294276777411
```

[45]:
```
classification_rep = classification_report(y_test, y_pred)
print("KNNs Classifier Report:")
print(classification_rep)
```

```
KNNs Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.42      0.38      0.40      7228
        ddos       0.02      0.01      0.02      2032
   injection       0.21      0.24      0.22      1199
      normal       0.93      0.95      0.94    103123
    password       0.29      0.25      0.27      3792
  ransomware       0.20      0.12      0.15       577
    scanning       0.00      0.00      0.00       114
         xss       0.26      0.21      0.23       225

    accuracy                           0.86    118290
   macro avg       0.29      0.27      0.28    118290
weighted avg       0.85      0.86      0.86    118290
```

[47]:
```
y_pred = logistic_model.predict(X_test_scaled)
ll_accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy:", ll_accuracy)
```

```
Logistic Regression Accuracy: 0.872736495054527
```

[48]:
```
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Classification Report:
              precision    recall  f1-score   support

    backdoor       0.65      0.23      0.34      7228
        ddos       0.00      0.00      0.00      2032
   injection       0.00      0.00      0.00      1199
      normal       0.89      0.98      0.93    103123
    password       0.17      0.05      0.08      3792
  ransomware       0.00      0.00      0.00       577
    scanning       0.00      0.00      0.00       114
         xss       0.00      0.00      0.00       225

    accuracy                           0.87    118290
   macro avg       0.21      0.16      0.17    118290
weighted avg       0.82      0.87      0.84    118290
```

Figure 22: Logistic Regression of garage_door dataset

[49]:
```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print("Naive Bayes Accuracy:", nb_accuracy)
```

```
Naive Bayes Accuracy: 0.8526164510947671
```

[50]:
```
classification_rep = classification_report(y_test, nb_predictions)

# Print classification report
print("Naive Bayes Classifier Report:")
print(classification_rep)
```

```
Naive Bayes Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.22      0.10      0.14      7228
        ddos       0.00      0.00      0.00      2032
   injection       0.27      0.25      0.26      1199
      normal       0.92      0.95      0.94    103123
    password       0.26      0.57      0.35      3792
  ransomware       0.00      0.00      0.00       577
    scanning       0.00      0.00      0.00       114
         xss       0.00      0.00      0.00       225

    accuracy                           0.85    118290
   macro avg       0.21      0.23      0.21    118290
weighted avg       0.83      0.85      0.84    118290
```

Figure 23: Naive Bayes of garage_door dataset

[37]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.8651365288697269
```

[38]:
```
# Get classification report
classification_rep = classification_report(y_test, dt_predictions)

# Print classification report
print("Decision Tree Classification Report:")
print(classification_rep)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

    backdoor       0.36      0.37      0.36      7228
        ddos       0.31      0.36      0.34      2032
   injection       0.19      0.21      0.20      1199
      normal       0.94      0.94      0.94    103123
    password       0.42      0.38      0.40      3792
  ransomware       0.15      0.12      0.14       577
    scanning       0.00      0.00      0.00       114
         xss       0.44      0.48      0.46       225

    accuracy                           0.87    118290
   macro avg       0.35      0.36      0.35    118290
weighted avg       0.87      0.87      0.87    118290
```

Figure 24: LDA of garage_door dataset

[37]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.8651365288697269
```

[38]:
```
# Get classification report
classification_rep = classification_report(y_test, dt_predictions)

# Print classification report
print("Decision Tree Classification Report:")
print(classification_rep)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

    backdoor       0.36      0.37      0.36      7228
        ddos       0.31      0.36      0.34      2032
   injection       0.19      0.21      0.20      1199
      normal       0.94      0.94      0.94    103123
    password       0.42      0.38      0.40      3792
  ransomware       0.15      0.12      0.14       577
    scanning       0.00      0.00      0.00       114
         xss       0.44      0.48      0.46       225

    accuracy                           0.87    118290
   macro avg       0.35      0.36      0.35    118290
weighted avg       0.87      0.87      0.87    118290
```

Figure 25: QDA of garage_door dataset

5.4 Implementation of UNSW-NB15

1. This shows the comprehensive evaluation using UNSW-NB15 dataset.
2. Preprocess the dataset by handling missing values.
3. Train ML models and evaluate their performance.
4. Accuracy, precision, recall, and F1-score for each model.
5. Comparison charts for UNSW-specific models.

Binary Classification

[26]:
```
plt.figure(figsize=(8,8))
plt.pie(data.label.value_counts(),labels=['normal','abnormal'],autopct='%0.2f%%')
plt.title("Pie chart distribution of normal and abnormal labels",fontsize=16)
plt.legend()
plt.savefig('C:/Users/shahn/OneDrive/Desktop/CODE_Intrusion_Detection_2024/plots_UNSW/Pie_chart_binary.png')
plt.show()
```

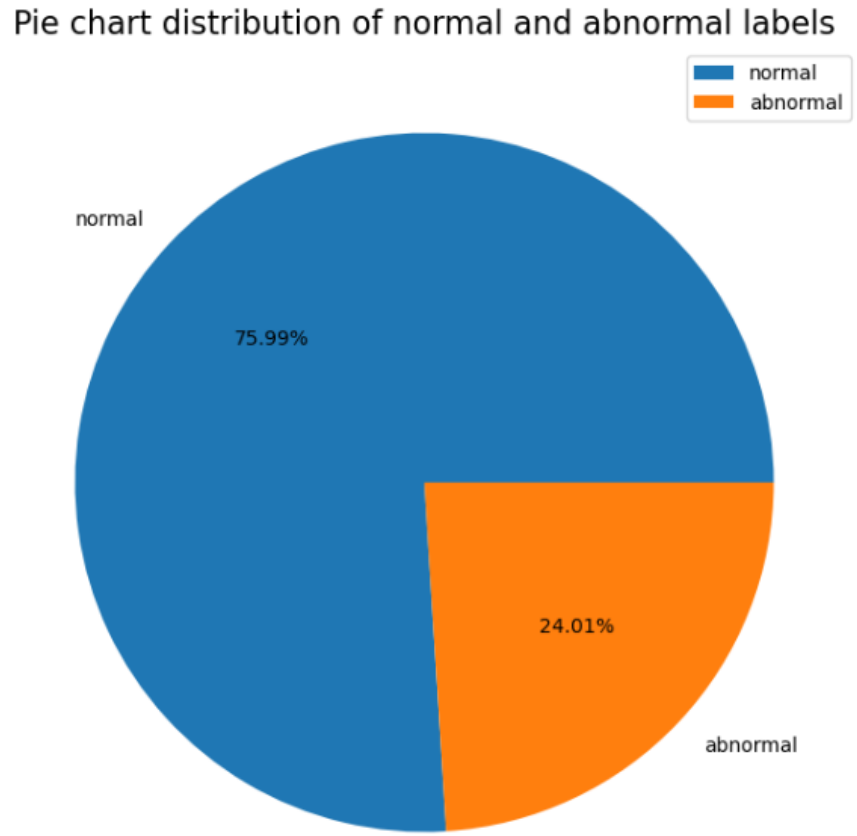


Figure 26: Visualization of Binary classification

Multi-class Classification

```
[27]: plt.figure(figsize=(8,8))
plt.pie(data.attack_cat.value_counts(),labels=data.attack_cat.unique(),autopct='%0.2f%%')
plt.title('Pie chart distribution of multi-class labels')
plt.legend(loc='best')
plt.savefig('C:/Users/shahn/OneDrive/Desktop/CODE_Intrusion_Detection_2024/plots_UNSW/Pie_chart_multi.png')
plt.show()
```

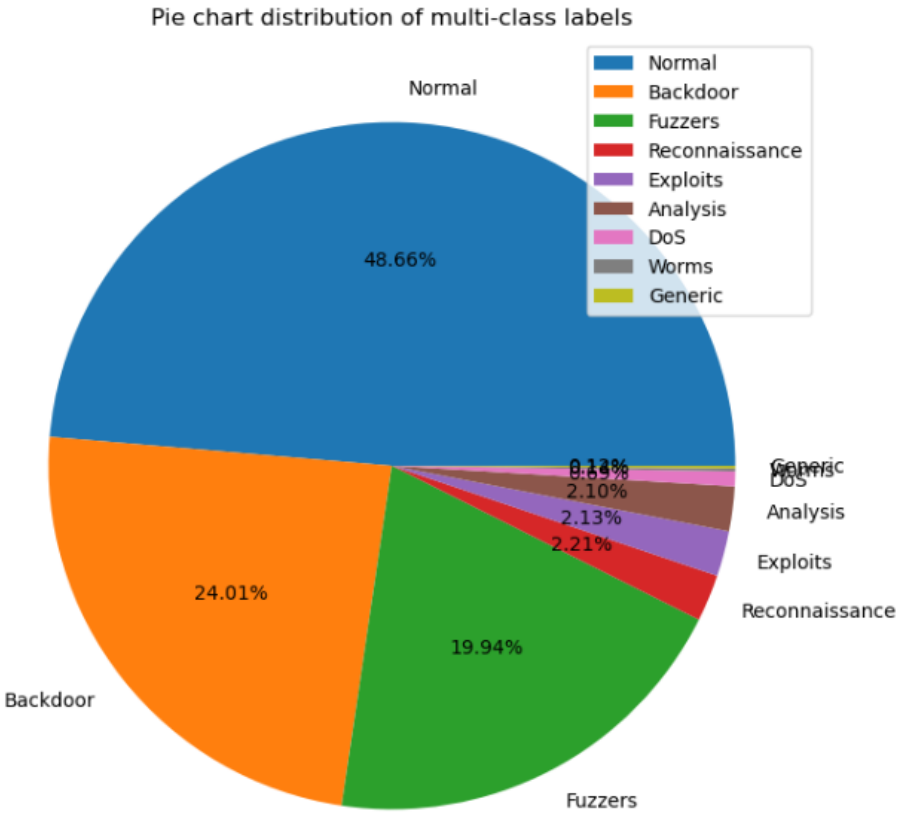


Figure 27: Visualization of Multi-class classification

Correlation Matrix for Binary Labels

[53]:
```
# Correlation Matrix for Binary Labels
plt.figure(figsize=(20,8))
corr_bin = bin_data[num_col].corr()
sns.heatmap(corr_bin,vmax=1.0,annot=False)
plt.title('Correlation Matrix for Binary Labels',fontsize=16)
plt.savefig('C:/Users/shahn/OneDrive/Desktop/CODE_Intrusion_Detection_2024/plots_UNSW/correlation_matrix_bin.png')
plt.show()
```

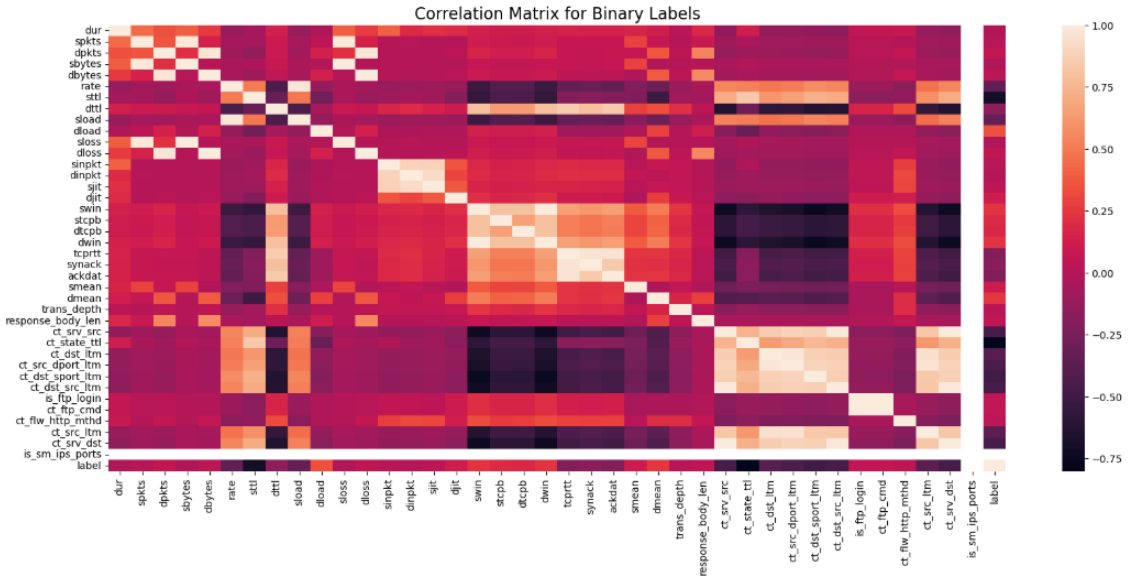


Figure 28: Correlation Matrix for Binary Labels

Correlation Matrix for Multi-class Labels

[54]:
```
num_col = list(multi_data.select_dtypes(include='number').columns)
```

[55]:
```
# Correlation Matrix for Multi-class Labels
plt.figure(figsize=(20,8))
corr_multi = multi_data[num_col].corr()
sns.heatmap(corr_multi,vmax=1.0,annot=False)
plt.title('Correlation Matrix for Multi Labels',fontsize=16)
plt.savefig('C:/Users/shahn/OneDrive/Desktop/CODE_Intrusion_Detection_2024/plots_UNSW/correlation_matrix_multi.png')
plt.show()
```

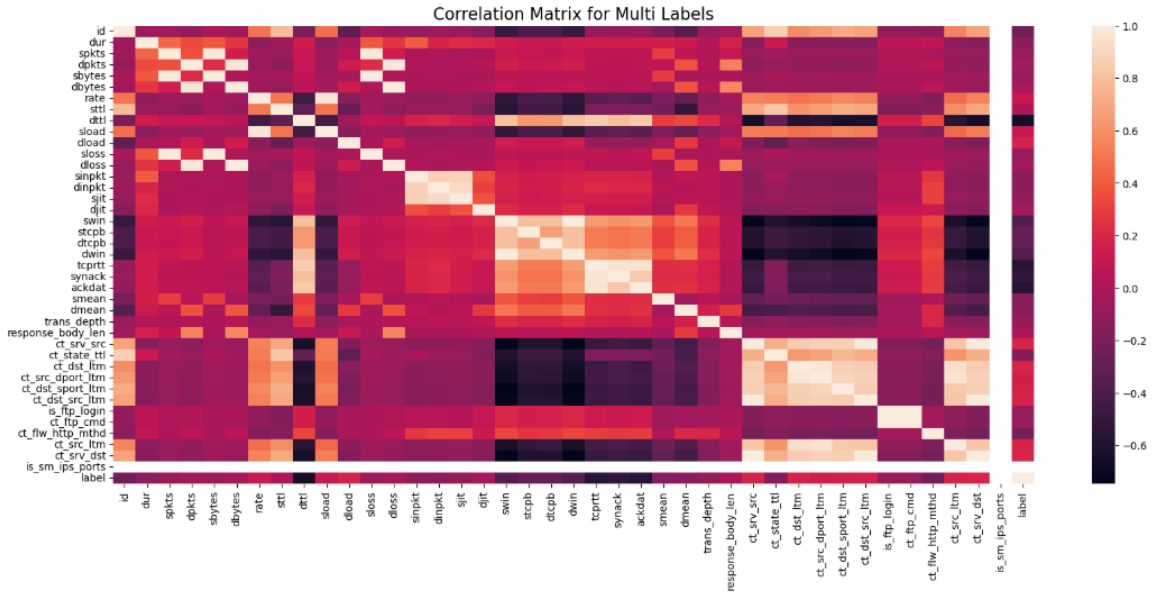


Figure 29: Correlation Matrix for Multi-class labels

	A	B	C	D	E	F	G	H
1		dttl	swin	dwin	tcprtt	synack	ackdat	label
2	3	0.99213	1	1	0	0	0	6
3	11	0.99213	1	1	0.08797	0.103	0.05498	6
4	15	0	0	0	0	0	0	6
5	17	0.99213	1	1	0.08071	0.11537	0.04291	6
6	21	0.99213	1	1	0.1036	0.1074	0.06975	6
7	22	0.99213	1	1	0.07542	0.10249	0.04202	6
8	28	0.99213	1	1	0.05695	0.08548	0.02881	6
9	30	0.99213	1	1	0.07084	0.07229	0.04811	6
10	31	0.99213	1	1	0.06751	0.09819	0.03529	6
11	32	0.99213	1	1	0.06119	0.05654	0.04368	6
12	34	0.99213	1	1	0.07875	0.05629	0.06215	6
13	35	0.99213	1	1	0.0586	0.09039	0.02877	6
14	39	0.99213	1	1	0.06008	0.06972	0.03777	6
15	41	0.99213	1	1	0.07036	0.06592	0.0499	6
16	43	0.99213	1	1	0.05152	0.04037	0.03938	6
17	44	0.99213	1	1	0.07816	0.10302	0.0447	6
18	45	0.99213	1	1	0.05965	0.05856	0.04134	6
19	47	0.99213	1	1	0.0491	0.03991	0.03702	6
20	48	0.99213	1	1	0.05433	0.04198	0.04174	6
21	56	0.99213	1	1	0.04607	0.03297	0.03635	6
22	60	0.99213	1	1	0.04081	0.02268	0.03455	6
23	63	0.99213	1	1	0.05121	0.02529	0.04449	6
24	64	0.99213	1	1	0.04022	0.02041	0.03475	6
25	68	0.99213	1	1	0.06388	0.05189	0.04817	6

Figure 30: Features saved as "multi_data.csv"

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	rate	sttl	sload	dload	ct_srv_src	ct_state_t	ct_dst_ltm	ct_src_dp	ct_dst_sp	ct_dst_src	ct_src_ltm	ct_srv_dst	label	
3	1.3677107	0.1383928	1.1893137	0.00015	0	0.1666666	0.02	0	0	0.039216	0.016949	0	1	
11	4.2520966	0.1383928	5.1937988	0.000364	0.019608	0.1666666	0	0	0	0.019608	0	0	1	
15	0.4999999	0.9955357	0.1197916	0	0	0.3333333	0	0	0	0.0588235	0.016949	0	1	
17	4.3195885	0.1383928	6.8287612	0.000869	0.039216	0.1666666	0	0	0	0.019608	0	0.039216	1	
21	4.4376467	0.9955357	9.2348251	0.000236	0.098039	0.1666666	0	0	0	0	0.016949	0.039216	1	
22	2.5915973	0.1383928	2.3072416	0.000322	0	0.5	0	0	0	0.019608	0.016949	0	1	
28	3.1111314	0.9955357	6.6848661	0.000594	0.098039	0.1666666	0	0	0	0.019608	0	0.1372549	1	
30	3.4435450	0.1383928	3.5391991	0.00563	0.098039	0.1666666	0	0	0	0.019608	0.016949	0	1	
31	5.4623912	0.1383928	1.3984733	0.000291	0.078431	0.1666666	0	0	0	0.0588235	0	0.1372549	1	
32	5.0376939	0.1383928	7.4083735	0.002091	0.039216	0.1666666	0	0	0	0	0	0.1372549	1	
34	5.0563613	0.9955357	1.0417040	0.000269	0.039216	0.1666666	0	0	0	0	0	0.039216	1	
35	3.8610096	0.1383928	8.8535096	0.001499	0.098039	0.1666666	0	0	0	0.039216	0	0.019608	1	
39	5.1932942	0.9955357	8.1626086	0.000277	0.098039	0.1666666	0	0	0	0.039216	0	0	1	
41	3.1912768	0.1383928	4.4206648	0.000784	0.098039	0.1666666	0	0	0	0.019608	0	0.1372549	1	
43	4.9882333	0.1383928	7.3763910	0.002486	0.078431	0.1666666	0	0	0	0.0588235	0	0.1372549	1	
44	3.1976033	0.1383928	4.1664678	0.001668	0.078431	0.1666666	0	0	0	0.0588235	0	0.1372549	1	
45	8.4473813	0.1383928	9.3352839	0.0260275	0.098039	0.1666666	0	0	0	0.7647058	0	0.1372549	1	
47	5.7355892	0.1383928	1.0182749	0.004155	0.098039	0.1666666	0	0	0	0.019608	0	0	1	
48	4.4047123	0.1383928	4.7018093	0.000587	0.019608	0.1666666	0.02	0	0	0.019608	0	0.019608	1	
56	5.5814011	0.1383928	8.6411395	0.002538	0.078431	0.1666666	0	0	0	0.0588235	0	0.039216	1	
60	6.5728021	0.1383928	4.1512434	0.001434	0.098039	0.1666666	0	0	0	0	0	0.019608	1	
63	3.7938214	0.9955357	6.5709978	0.000247	0.098039	0.1666666	0	0	0	0.039216	0	0	1	
64	6.5099927	0.1383928	9.2082138	0.005248	0.078431	0.1666666	0	0	0	0.0588235	0	0.1372549	1	
68	6.7309893	0.1383928	5.3019896	0.0150552	0.098039	0.1666666	0	0	0	0	0	0.039216	1	
91	5.7614107	0.1383928	8.4743557	0.00061	0	0.1666666	0.02	0	0	0.039216	0	0.019608	1	
100	9.1749499	0	1.6072597	0.000101	0.039216	0	0.16	0.02	0	0.019608	0.050847	0.078431	1	

Figure 31: Features saved as "bin_data.csv"

5.5 Train_Model

1. Training CNN and hybrid CNN-LSTM models for intrusion detection.
2. Import libraries and load preprocessed datasets.
3. Train CNN and CNN-LSTM models.
4. Model Evaluation metrics such as accuracy and F1-score
5. Plots of training vs. validation accuracy.

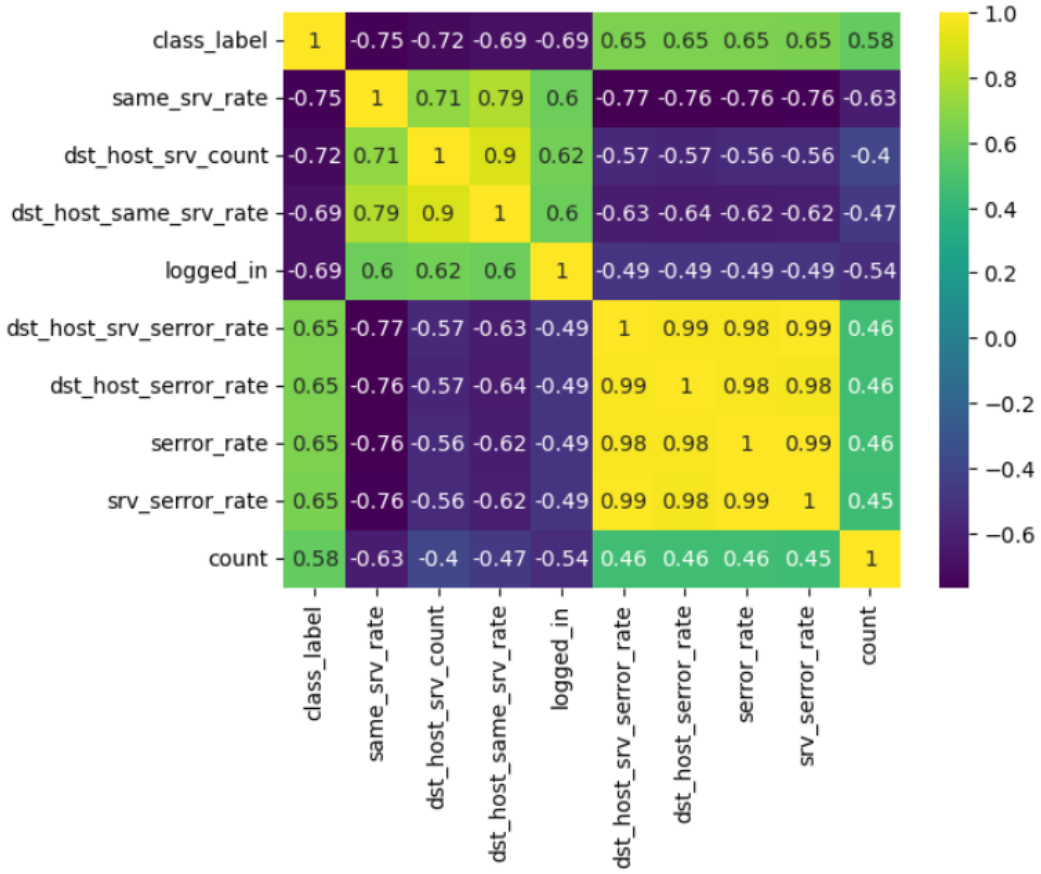
[14]:
```
# find the correlation coefficient between the numerical variables with class label and make it absolute
corr = abs(df_numerical.corr()['class_label']).sort_values(ascending=False)
# print(corr)

# pick only correlation values greater than 0.5
corr = corr[corr > 0.5]
print(corr)
```

```
class_label                1.000000
same_srv_rate              0.751913
dst_host_srv_count         0.722535
dst_host_same_srv_rate     0.693803
logged_in                  0.690171
dst_host_srv_serror_rate   0.654985
dst_host_serror_rate       0.651842
serror_rate                0.650652
srv_serror_rate            0.648289
count                      0.576444
Name: class_label, dtype: float64
```

[15]:

```
# now save the above correlation values in a list and check for direct and inverse relation between the variables wrt class label
corr_list = corr.index.tolist()
print(corr_list)

# now plot the correlation matrix
# plt.figure(figsize=(15,15))
sns.heatmap(df_numerical[corr_list].corr(), annot=True, cmap='viridis')
plt.show()
```

```
['class_label', 'same_srv_rate', 'dst_host_srv_count', 'dst_host_same_srv_rate', 'logged_in', 'dst_host_srv_serror_rate', 'dst_host_serror_rate', 'serro
r_rate', 'srv_serror_rate', 'count']
```



[16]:
```
# do one hot encoding for the categorical variables and check its correlation with class label
df_categorical = pd.get_dummies(df_categorical, drop_first=True)
df_categorical.head()
```

[16]:

	protocol_type_tcp	protocol_type_udp	service_X11	service_Z39_50	service_aol	service_auth	service_bgp	service_courier	service_csnet_ns	service_ctf	service_daytim
0	True	False	False	False	False	False	False	False	False	False	Fals
1	False	True	False	False	False	False	False	False	False	False	Fals
2	True	False	False	False	False	False	False	False	False	False	Fals
3	True	False	False	False	False	False	False	False	False	False	Fals
4	True	False	False	False	False	False	False	False	False	False	Fals

[17]:
```
# now concatenate the numerical and categorical variables
df_new = pd.concat([df_numerical, df_categorical], axis=1)
df_new.head()
```

[17]:

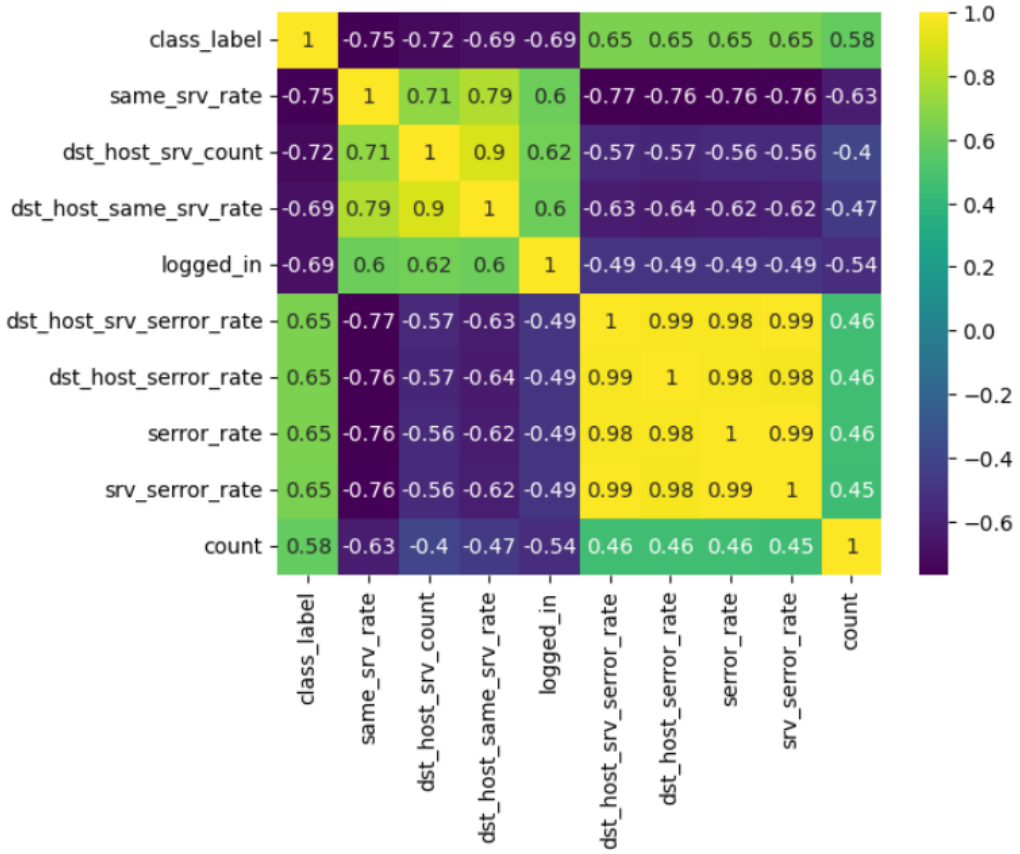
	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root	num_file_
0	0	491	0	0	0	0	0	0	0	0	0	0	0	
1	0	146	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	232	8153	0	0	0	0	0	1	0	0	0	0	
4	0	199	420	0	0	0	0	0	1	0	0	0	0	

[18]:
```
# now find the correlation between the new dataframe and class label
corr = abs(df_new.corr()['class_label']).sort_values(ascending=False)
corr = corr[corr > 0.5]
# corr_list = corr.index.tolist()
# print(corr_list)
corr
```

[18]:
```
attack_class_normal          1.000000
attack_type_normal           1.000000
class_label                  1.000000
flag_SF                      0.756286
same_srv_rate                0.751913
attack_type_neptune          0.747336
dst_host_srv_count           0.722535
dst_host_same_srv_rate       0.693803
logged_in                    0.690171
dst_host_srv_serror_rate     0.654985
dst_host_serror_rate         0.651842
serror_rate                  0.650652
flag_S0                      0.650206
srv_serror_rate              0.648289
count                        0.576444
service_http                 0.562312
Name: class_label, dtype: float64
```

[19]:

```
# now plot the correlation matrix
# plt.figure(figsize=(15,15))
sns.heatmap(df_new[corr_list].corr(), annot=True, cmap='viridis')
plt.show()
```



[20]:

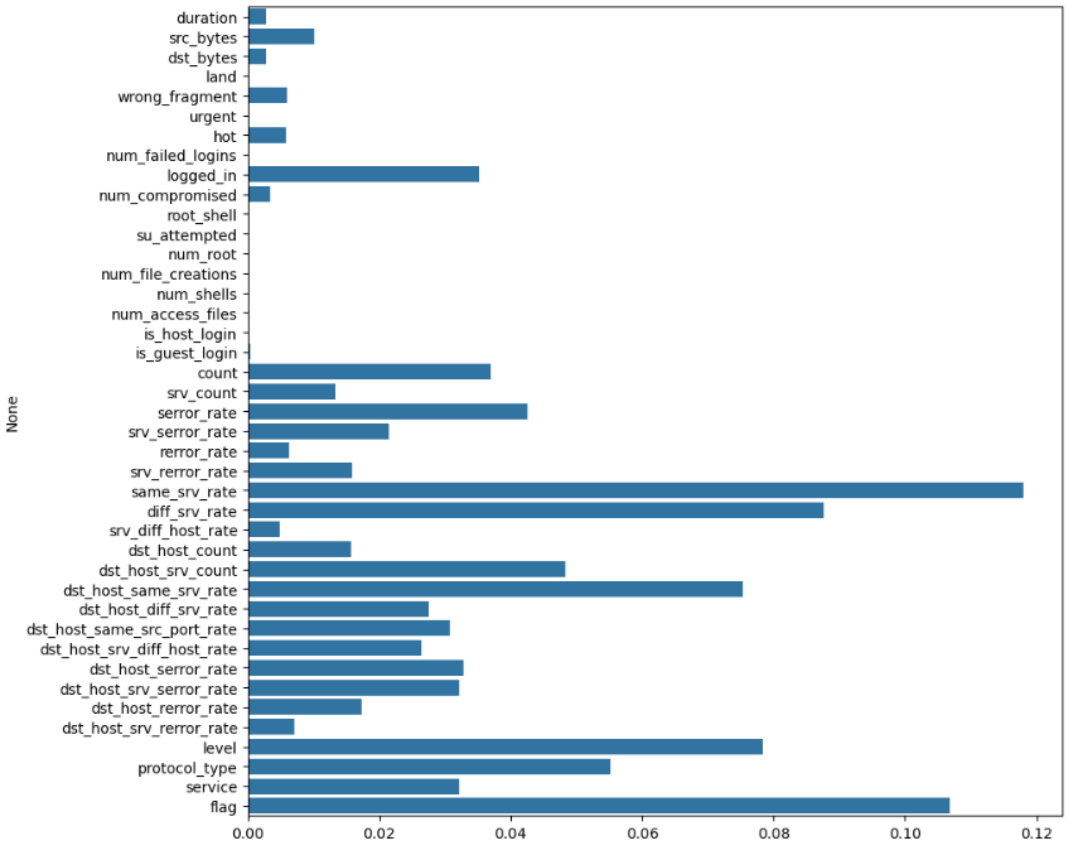
```
df_numerical = df.select_dtypes(include=np.number)
df_categorical = df.select_dtypes(exclude=np.number)


# label encoding for the categorical variables in df data
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_e_categorical = df_categorical.apply(le.fit_transform)
# df_e_categorical.head()

# normalize the numerical variables
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_n_numerical = pd.DataFrame(scaler.fit_transform(df_numerical), columns=df_numerical.columns)


# concat the encoded and normalized datasets
df_ne = pd.concat([df_n_numerical, df_e_categorical], axis=1)
# df_ne.head()

# drop attack_class and attack_type columns
df_ne.drop(['attack_class', 'attack_type'], axis=1, inplace=True)
df_ne.head()
```

[20]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root	n
0	0.0	3.558064e-07	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	1.057999e-07	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.000000e+00	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	1.681203e-07	6.223962e-06	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
4	0.0	1.442067e-07	3.206260e-07	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	

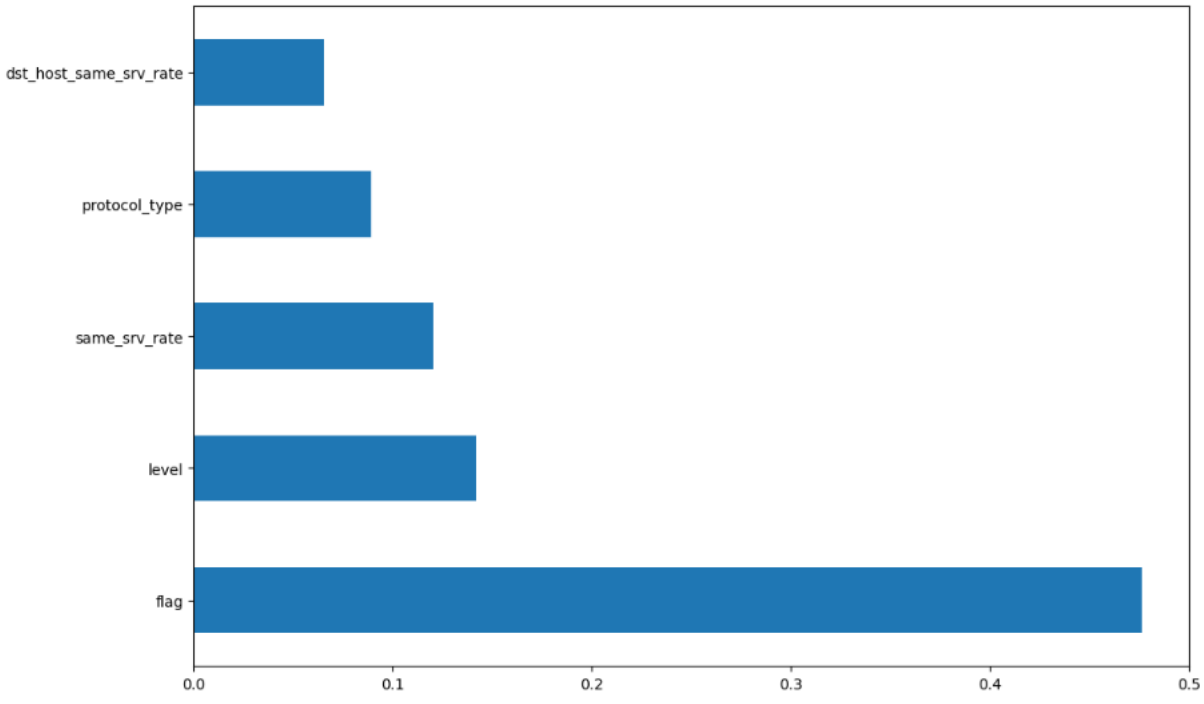
[22]:
```
# perfrom random forest feature selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split

X = df_ne.drop(['class_label'], axis=1)
y = df_ne['class_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# fit the model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# select the features
sel = SelectFromModel(rf)
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot all the selected features
plt.figure(figsize=(10,10))
sns.barplot(x=sel.estimator_.feature_importances_, y=X_train.columns)
plt.show()
```

```
(88181, 41) (37792, 41) (88181,) (37792,)
16
Index(['logged_in', 'count', 'serror_rate', 'same_srv_rate', 'diff_srv_rate',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'level', 'protocol_type', 'service',
       'flag'],
      dtype='object')
```



[23]:
```
# best features by gradient boosting
from sklearn.ensemble import GradientBoostingClassifier  # or GradientBoostingRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split

X = df_ne.drop(['class_label'], axis=1)
y = df_ne['class_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# fit the model
gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train, y_train)

# select the features
sel = SelectFromModel(gb)
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot the feature importance
plt.figure(figsize=(12,8))
feat_importances = pd.Series(gb.feature_importances_, index=X_train.columns)
feat_importances.nlargest().plot(kind='barh')
plt.show()
```

```
(88181, 41) (37792, 41) (88181,) (37792,)
5
Index(['same_srv_rate', 'dst_host_same_srv_rate', 'level', 'protocol_type',
       'flag'],
      dtype='object')
```



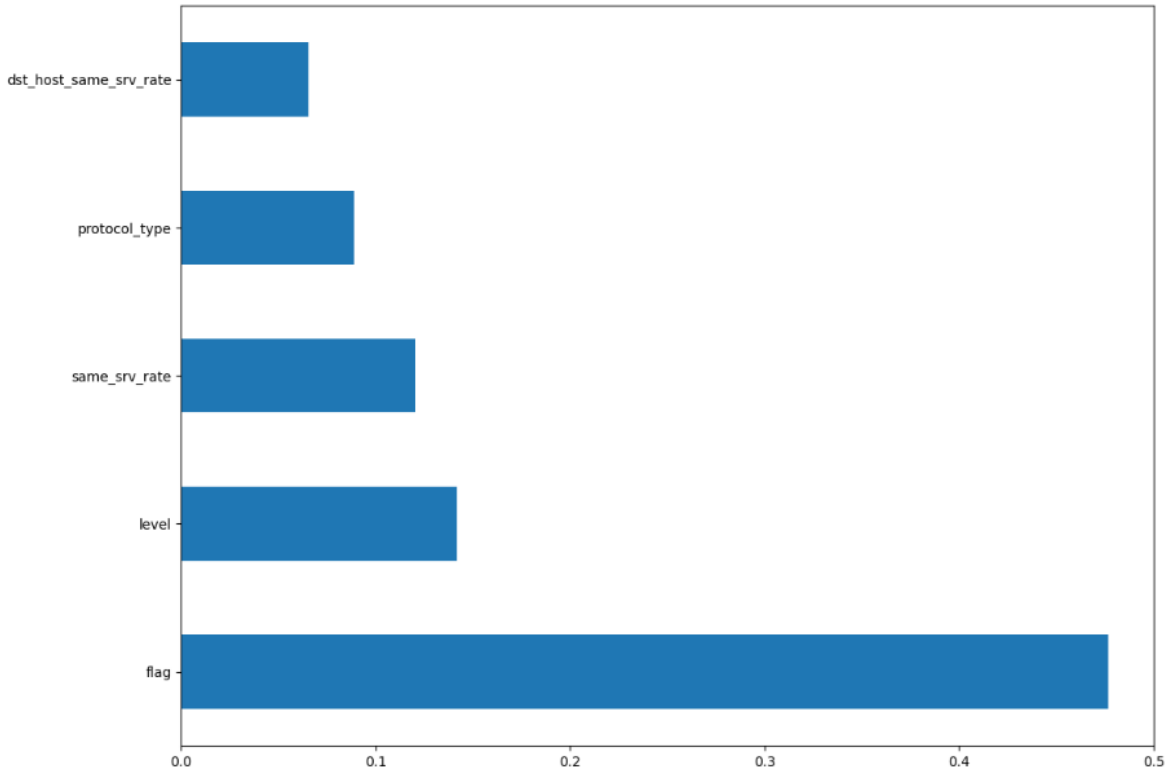
[24]:
```
# drop attack_class and attack_type columns
df.drop(['attack_class', 'attack_type'], axis=1, inplace=True)


df_numerical = df.select_dtypes(include=np.number)
df_categorical = df.select_dtypes(exclude=np.number)


# one hot encoding of the categorical variables
df_he_categorical = pd.get_dummies(df_categorical, drop_first=True)
df_he_categorical.head()

# normalize the numerical variables
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_n_numerical = pd.DataFrame(scaler.fit_transform(df_numerical), columns=df_numerical.columns)

# concat the encoded and normalized datasets
df_he_n = pd.concat([df_n_numerical, df_he_categorical], axis=1)
df_he_n.head()
```

[24]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root
0	0.0	3.558064e-07	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.057999e-07	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.000000e+00	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	1.681203e-07	6.223962e-06	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	1.442067e-07	3.206260e-07	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

[25]:
```
# fit the model
gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train, y_train)

# select the features
sel = SelectFromModel(gb)
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot the feature importance
plt.figure(figsize=(12,8))
feat_importances = pd.Series(gb.feature_importances_, index=X_train.columns)
feat_importances.nlargest().plot(kind='barh')
plt.tight_layout()
```

```
5
Index(['same_srv_rate', 'dst_host_same_srv_rate', 'level', 'protocol_type',
       'flag'],
      dtype='object')
```

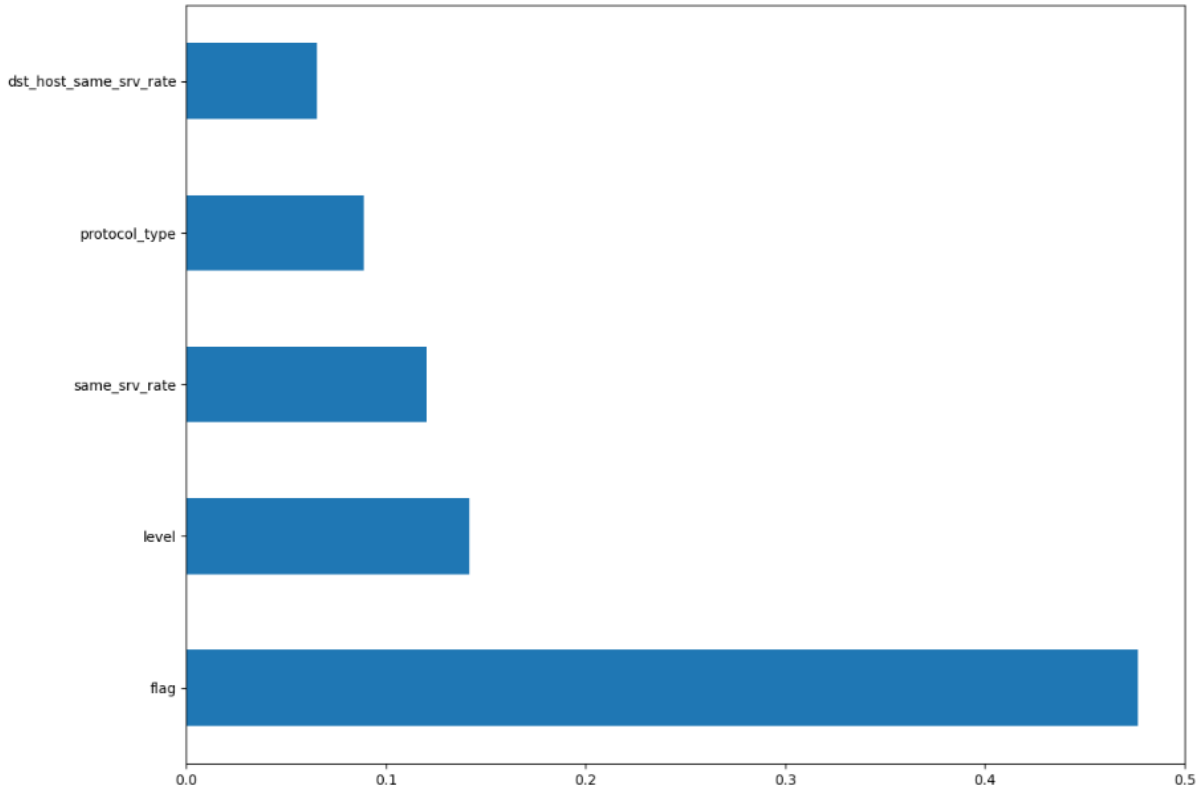


[26]:

```
from xgboost import XGBClassifier
# fit the model
xgb = XGBClassifier(n_estimators=100, random_state=42)
xgb.fit(X_train, y_train)

# select the features
sel = SelectFromModel(xgb)
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot the feature importance
plt.figure(figsize=(12,8))
feat_importances = pd.Series(gb.feature_importances_, index=X_train.columns)
feat_importances.nlargest().plot(kind='barh')
plt.tight_layout()
plt.show()
```

```
8
Index(['src_bytes', 'dst_bytes', 'logged_in', 'count', 'dst_host_srv_count',
       'dst_host_same_src_port_rate', 'level', 'protocol_type'],
      dtype='object')
```

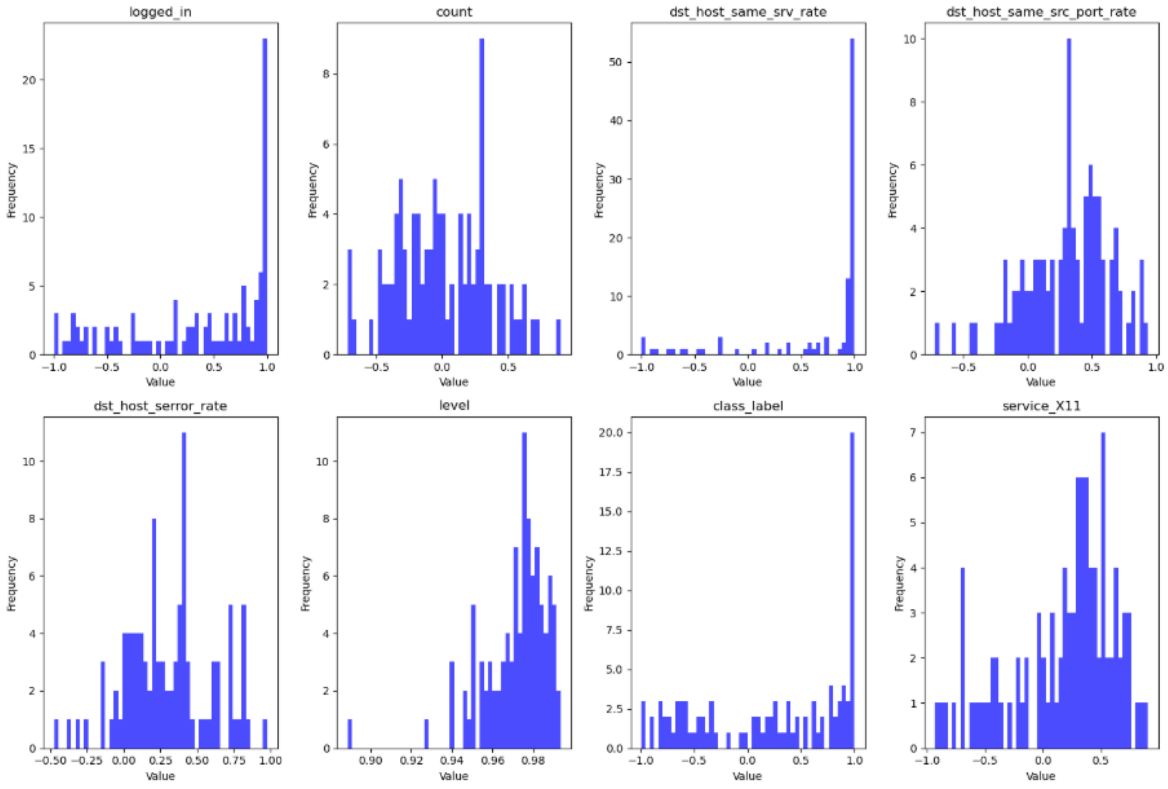
```
8
Index(['src_bytes', 'dst_bytes', 'logged_in', 'count', 'dst_host_srv_count',
       'dst_host_same_src_port_rate', 'level', 'protocol_type'],
      dtype='object')
```

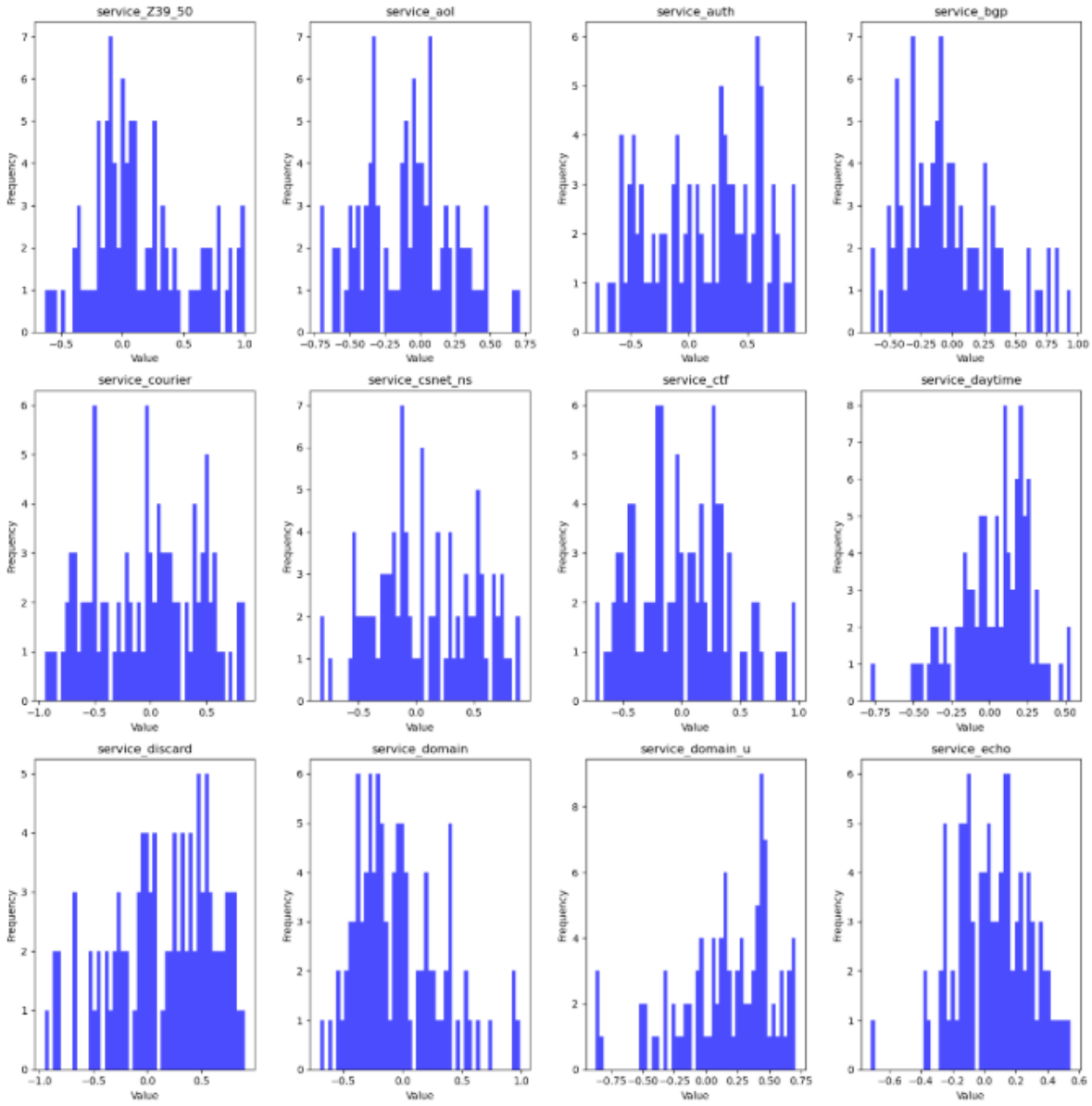


[39]:
```python
# Train the GAN
for epoch in range(epochs + 1):
    idx = np.random.randint(0, df.shape[0], batch_size)
    real_data = df.iloc[idx]
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    generated_data = generator.predict(noise, verbose=0)

    real_labels = np.ones((batch_size, 1))
    fake_labels = np.zeros((batch_size, 1))

    d_loss_real = discriminator.train_on_batch(real_data, real_labels)
    d_loss_fake = discriminator.train_on_batch(generated_data, fake_labels)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    valid_labels = np.ones((batch_size, 1))

    g_loss = gan.train_on_batch(noise, valid_labels)

    # Print the progress
    if epoch % 100 == 0:
        print(f"Epoch {epoch}, D Loss: {d_loss}, G Loss: {g_loss}")
```

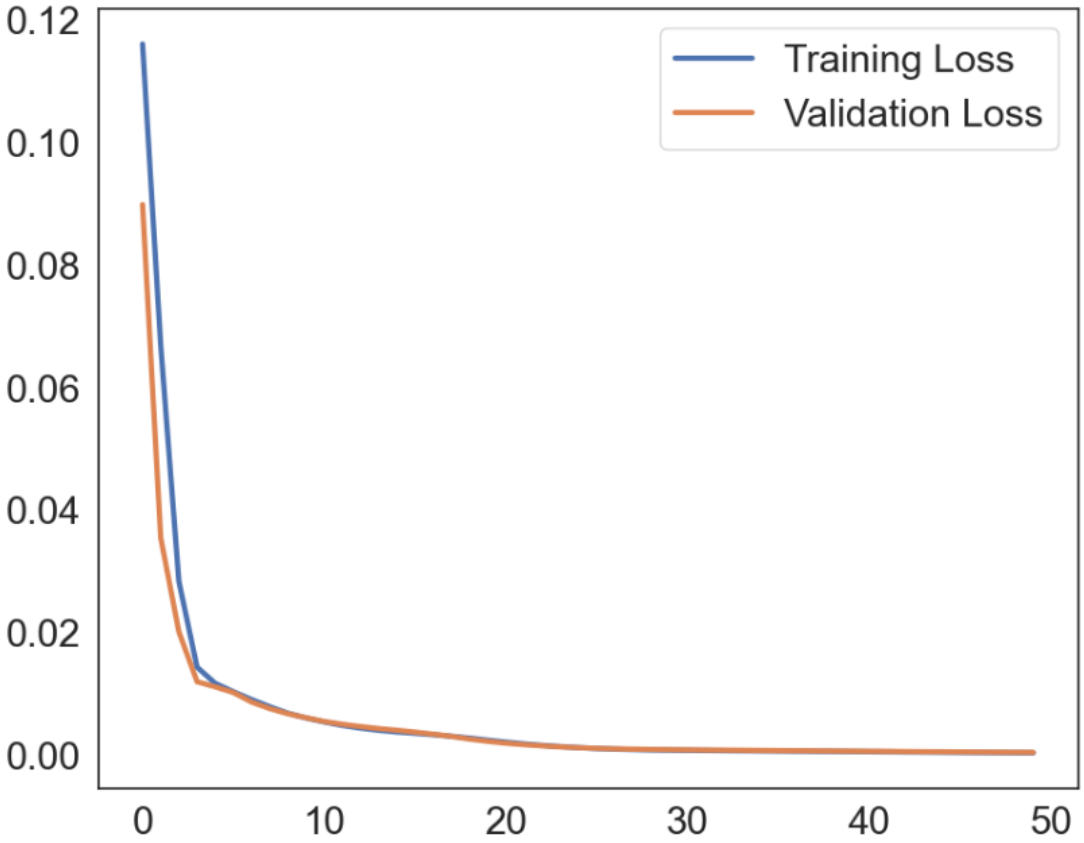
```
WARNING:tensorflow:From C:\Program Files\Anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecate
d. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

Epoch 0, D Loss: 0.7589017152786255, G Loss: 0.57380211353302
Epoch 100, D Loss: 0.7977718412876129, G Loss: 0.4618147015571594
Epoch 200, D Loss: 0.6338235139846802, G Loss: 0.6685913801193237
Epoch 300, D Loss: 0.6078830361366272, G Loss: 0.7505393028259277
Epoch 400, D Loss: 0.6138206720352173, G Loss: 0.7552889585494995
Epoch 500, D Loss: 0.6587792634963989, G Loss: 0.7068058252334595
Epoch 600, D Loss: 0.6692794263362885, G Loss: 0.7583783864974976
Epoch 700, D Loss: 0.6653631627559662, G Loss: 0.750003457069397
Epoch 800, D Loss: 0.6962503492832184, G Loss: 0.7321480512619019
Epoch 900, D Loss: 0.6993010342121124, G Loss: 0.6830166578292847
Epoch 1000, D Loss: 0.7098850011825562, G Loss: 0.6749182939529419
```

[41]:
```
import matplotlib.pyplot as plt

# Defining subplot format
features_per_row = 4
num_rows = len(df.columns) // features_per_row
fig, axs = plt.subplots(num_rows, features_per_row, figsize=(15, 5 * num_rows))

# Visualize the generated samples
for row in range(num_rows):
    for col in range(features_per_row):
        feature_idx = row * features_per_row + col
        if feature_idx < len(df.columns):
            axs[row, col].hist(generated_data[:, feature_idx], bins=50, color='blue', alpha=0.7)
            axs[row, col].set_title(df.columns[feature_idx])
            axs[row, col].set_xlabel('Value')
            axs[row, col].set_ylabel('Frequency')

plt.suptitle('Generated Data Distribution', y=1.02)
plt.tight_layout()
plt.show()
```




service_Z39_50
service_aol
service_auth
service_bgp
service_courier
service_csnet_ns
service_ctf
service_daytime
service_discard
service_domain
service_domain_u
service_echo
Frequency
Value

```
[74]: plt.figure(figsize=(10,8))
      sns.set(font_scale = 2)
      sns.set_style("white")
      plt.plot(history.history["loss"], label="Training Loss",linewidth=3.0)
      plt.plot(history.history["val_loss"], label="Validation Loss",linewidth=3.0)
      plt.legend()

[74]: <matplotlib.legend.Legend at 0x1c80e401490>
```



6 Web Application

6.1 Purpose

This web application is just a demonstration for uploading datasets and predicting intrusion attack types' categories using pre-trained models.

6.2 Running the Application

1. Navigate to the folder that contains "app.py".
2. Copy the path of the folder.
3. Open Anaconda Prompt and change directory to this folder, by pasting the path.
4. Run the web application by entering the following command:

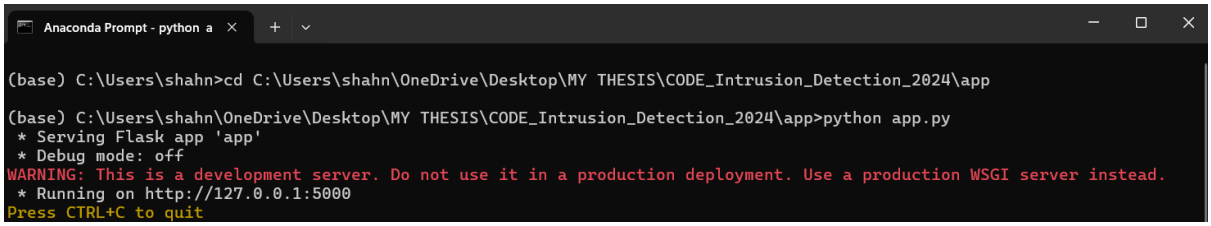


Figure 32: Running the web application in Anaconda Prompt

5. Open the web interface by using a browser.

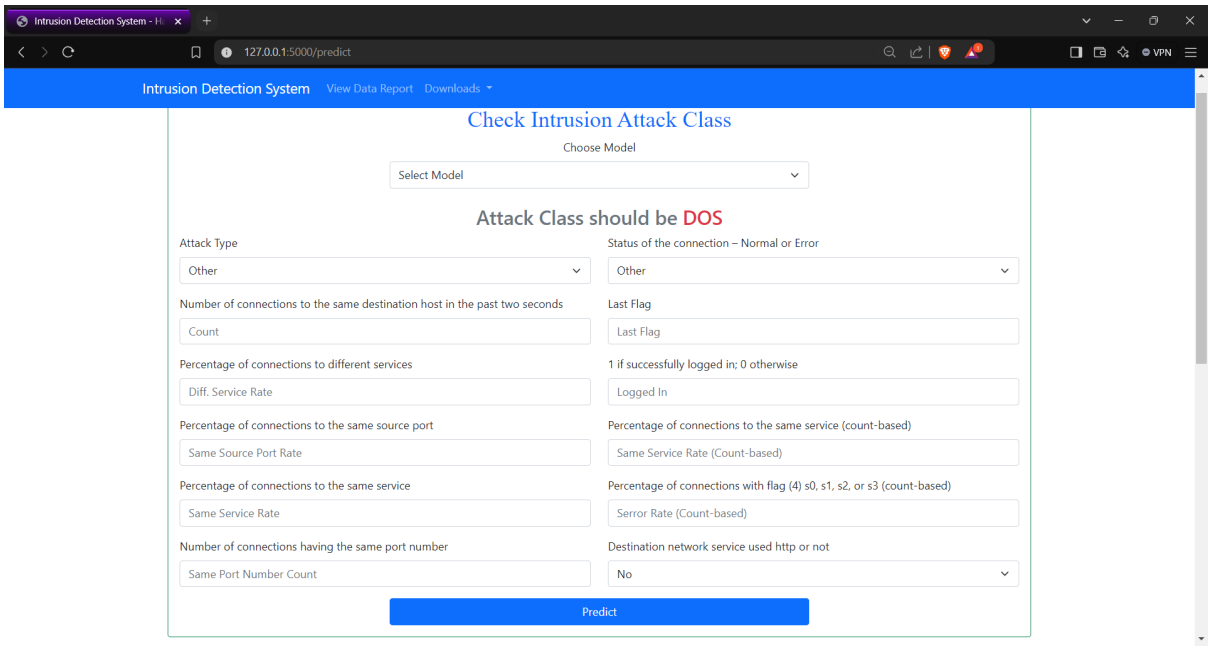


Figure 33: Website homepage

6.3 Directory structure

6.4 Inputs and Outputs

- **Input:** CSV file with network traffic data
- **Output:** Predicted attack classes displayed on the web interface

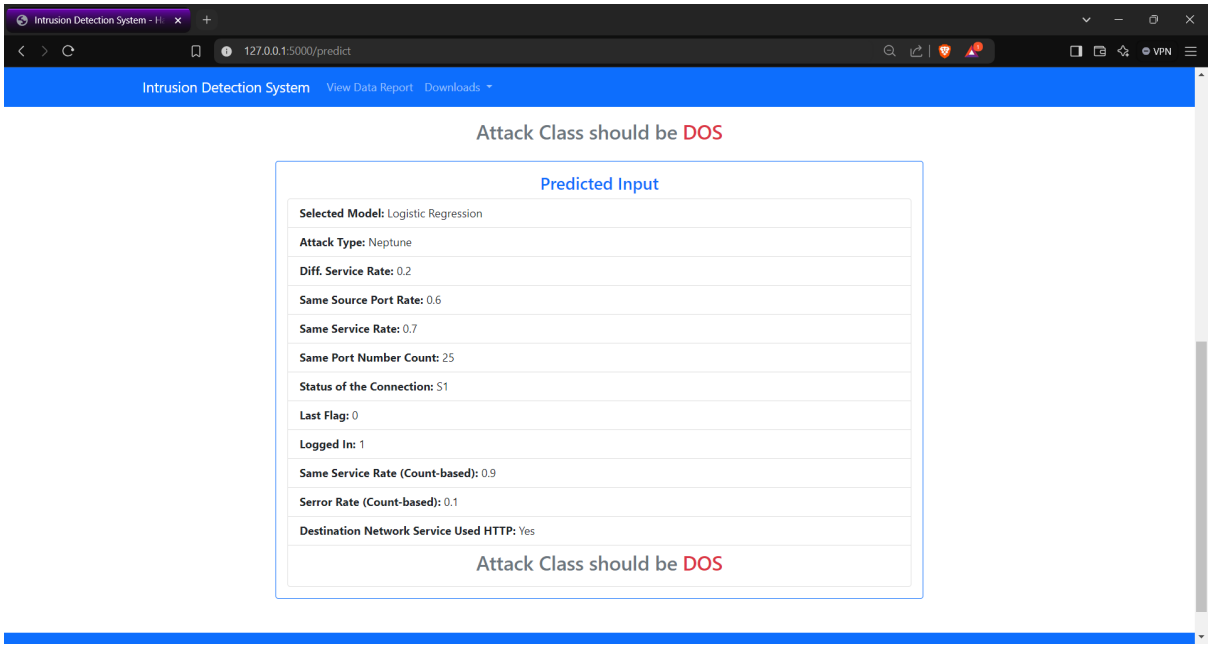


Figure 34: Attack Classes predictions - Result 1

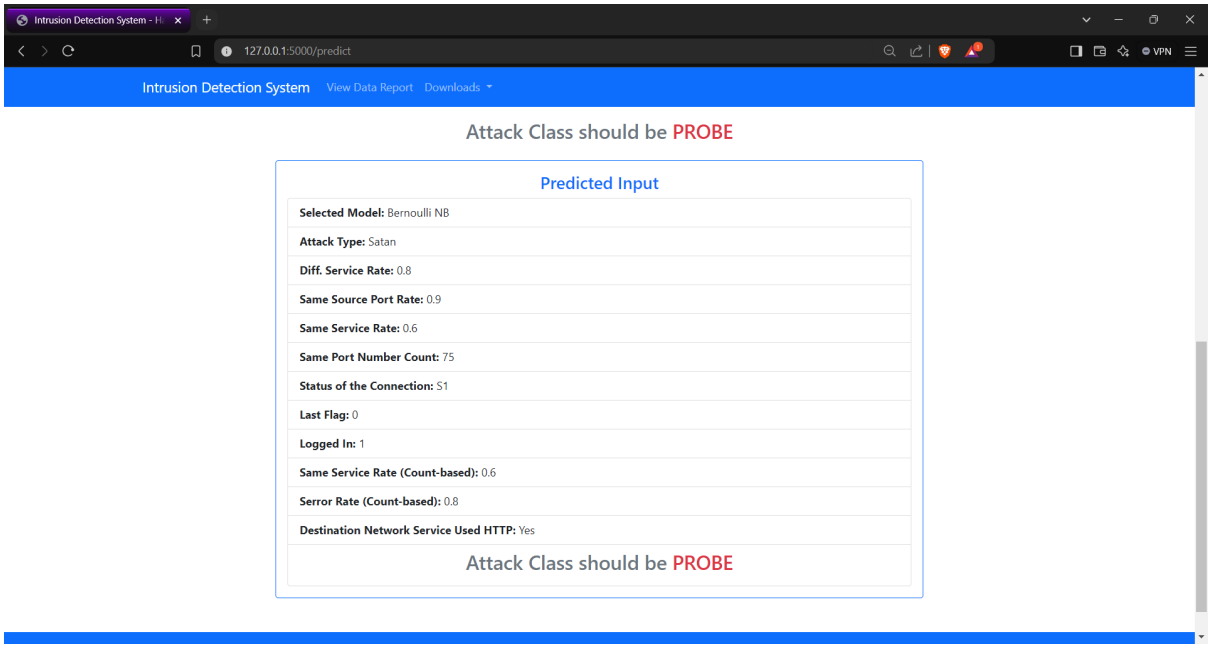


Figure 35: Attack Classes predictions - Result 2