National College of Ireland

Configuration Manual

MSc Research Project
MS in Cybersecurity

Vignesh Kannan
Student ID: 22203699

School of Computing
National College of Ireland

Supervisor: Naill Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vignesh Kannan
Student ID:	22203699
Programme:	Ms in cybersecurity **Year:** 2023 - 2024
Module:	Practicum
Lecturer:	Prof. Niall Heffernan
Submission Due Date:	12.08.2024
Project Title:	Advanced Intrusion Detection for IOT Devices
Word Count:	857 **Page Count:** 11

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: Vignesh Kannan

Date: 12.08.2024

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

Vignesh Kannan
22203699

1 Introduction

This is a configuration handbook, which contains the fundamental sets to build this project work. This project works intends to create a model using machine learning – SVC & Random Forest and deep learning – ANN algorithms to identify if any intrusions such as of Benign, dos, password, scanning, or xss occurs. This Configuration handbook is essential and includes all the necessary hardware requirements, software, and implementation techniques which were used to develop this project.

2 Requirement of Hardware

Operating system: Windows 11 home (64 bit)
RAM: 8GB/16 GB
Storage: 1TB HDD or SSD
Processor: 12th gen - Intel core i5 @ 3.20GHz 2.50 GHz
Graphics card: 2 GB/4 GB - NVIDA GeForce or AMD Radeon (Optional)
System type: 64-bit operating system (x64-based processor).

3 Requirement of Software

The following are the required software that are used for developing this project:

Python 3.7: It is a general programming language widely used in AI/ML for its simplicity, extensive libraries, and ability to handle complex data structures.

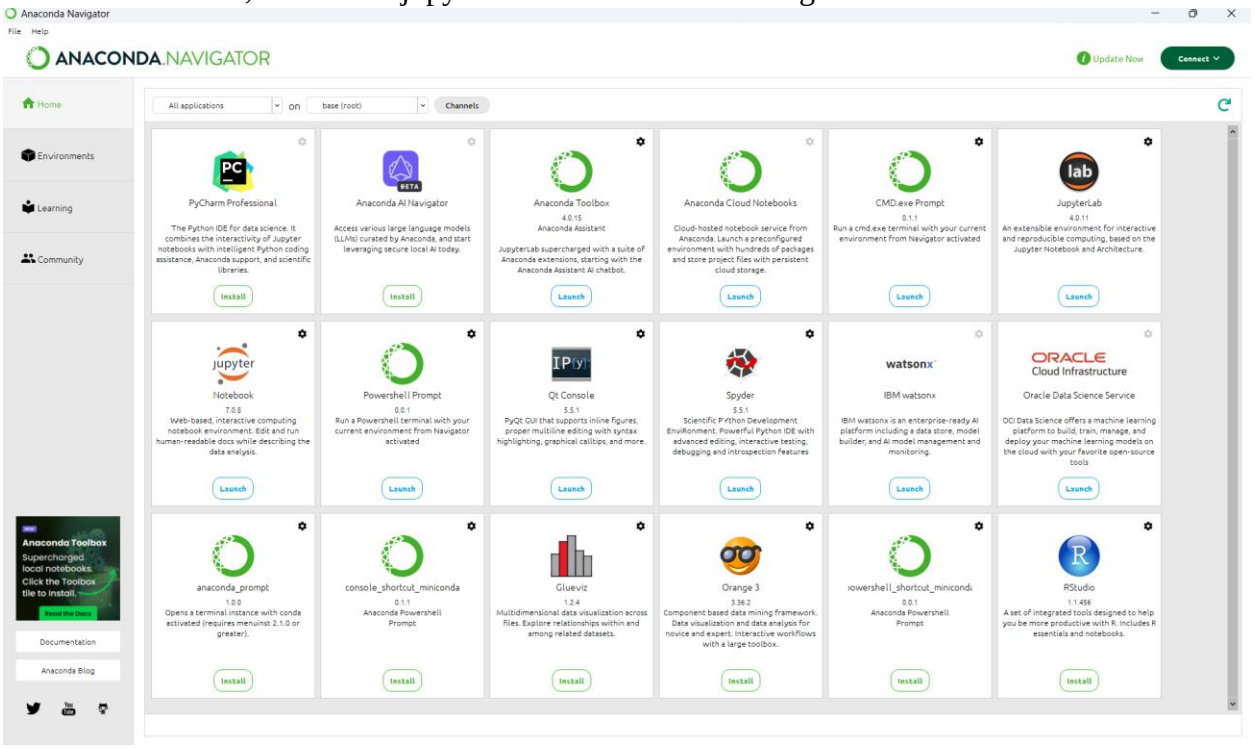
Anaconda distribution is actually the easiest way to set up an AI/ML development environment, abstracting away all issues with package management and deployment.

Jupyter Notebook is an interactive coding environment that simplifies prototyping and experimentations with AI/ML algorithms, while the Jupyter notebook IDE is used to run the code, the python libraries are utilized for analysis and visualization of various graphs and data.
Notepad++: A popular code editor for writing and editing Python scripts.

Anaconda Navigator with GUI makes it easier to manage environments, packages, and workflows in AI/ML development. (Download Anaconda Distribution, n.d.)

https://www.anaconda.com/download

By downloading Anaconda navigation from the link given above, the anaconda navigator along with Jupyter notebook and necessary python libraries will be installed in the system. After the installation is done, launch the jupyter notebook from the navigator.



4. Python Libraries Installed

The following python libraries which can be installed in the environment using the import command are the once that are needed for this research:

Flask: It is a lightweight web framework to serve AI/ML models as web applications.

Keras: Keras is a high-level neural networks API that is easy to build and fast to train, implemented with flexibility in order to carry out deep learning.

TensorFlow: TensorFlow is an open-source software library applied to large-scale machine learning/deep learning models.

OpenCV: OpenCV is used as a computer vision library applied for real-time image and video processing in AI/ML.

Matplotlib: Matplotlib is a plotting library for static, animated, and interactive visualizations within AI/ML projects.

Scikit-learn: It is used the machine learning library that contains tools for data mining and data analysis.

NumPy: NumPy is a general n-dimensional array-processing package; it defines large arrays with many useful methods and properties.

Pandas: It is package for manipulation and analysis library, which is used for structuring data into tabular forms, which are called DataFrames. (Best Python libraries for Machine Learning, 2019)

Yagmail: Yagmail is a python module that is designed to simplify the process of sending email through gmail by using SMTP protocol. The main use of this module is to allow users to send emails with very small amount of code, making it user-friendly for developers in AI/ML. (Send Email Using yagmail in Python, 2024)

```
[1]: import warnings
     warnings.filterwarnings("ignore")

     import os
     import pandas as pd
     pd.set_option("display.max_columns",None)
     import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import MinMaxScaler
     from tqdm import tqdm
     import pickle
     from sklearn.utils import resample
```

5 Description of Dataset:

NF-ToN-IoT-v2 is actually selected because it includes various categories of network attacks, from benign to DDoS attack, password attack, scanning, ransomware, and injection attack. This variation is what makes it very rich for research in different aspects of network security. Additionally, it is well-built for research within the Internet of Things domains that give insights into how different attack vectors impact different IoT environments. Such a wide-ranging coverage allows researchers to design and test strong security measures against the cyber threats. (Mohanad Sarhan, n.d.)

https://rdm.uq.edu.au/files/a4ad7080-ef9c-11ed-a964-b70596e96ad5

6. Data Importing and preprocessing

Data Importing

[2]:
```
df = pd.read_csv("dataset/NF-ToN-IoT-v2-002.csv", nrows=5000000)
```

Preliminary Analysis

[3]:
```
df.head()
```

[3]:

	IPV4_SRC_ADDR	L4_SRC_PORT	IPV4_DST_ADDR	L4_DST_PORT	PROTOCOL	L7_PROTO	IN_BYTES	IN_PKTS	OUT_BYTES	OUT_PKTS	TCP_FLAGS	CLIENT_TCP_FLAGS
0	192.168.1.193	49235	192.168.1.33	4444	6	0.0	155392	202	34552	149	24	24
1	192.168.1.193	49228	192.168.1.152	1880	6	0.0	1600	40	35741	65	24	16
2	192.168.1.152	0	192.168.1.193	0	1	0.0	212	2	0	0	0	0
3	192.168.1.169	65317	239.255.255.250	1900	17	0.0	165	1	0	0	0	0
4	192.168.1.79	60766	192.168.1.255	15600	17	0.0	63	1	0	0	0	0

[4]:
```
df.shape
```

[4]:
```
(5000000, 45)
```

Data preprocessing:

[10]:
```
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.dropna(inplace=True)
```

[11]:
```
df['Attack'].value_counts()
```

[11]:
```
Benign        2641663
scanning      2324473
password        12918
dos              7402
xss              6188
ddos             4098
injection        2723
mitm              502
ransomware         33
Name: Attack, dtype: int64
```

[12]:
```
df = df.drop(labels=df[df['Attack']=='mitm'].index)
df = df.drop(labels=df[df['Attack']=='ransomware'].index)
df = df.drop(labels=df[df['Attack']=='injection'].index)
df = df.drop(labels=df[df['Attack']=='ddos'].index)
```

[18]:
```
class_labels = df['Attack'].unique().tolist()
class_labels.sort()

print(class_labels)
```

```
['Benign', 'dos', 'password', 'scanning', 'xss']
```

[19]:
```
class_dict={}
for idx, label in enumerate(class_labels):
    class_dict[label] = idx
print(class_dict)
```

```
{'Benign': 0, 'dos': 1, 'password': 2, 'scanning': 3, 'xss': 4}
```

[20]:
```
df['Attack'] = df['Attack'].map(class_dict)
df.head()
```

[24]:
```
target_feature = 'Attack'
all_features = df.columns.tolist()
all_features.remove(target_feature)
corr = df[all_features].corrwith(df[target_feature])
```

[25]:
```
corr_df = pd.DataFrame(corr).reset_index()
corr_df.columns = ['Features', 'Importance']
corr_df.head(10)
```

[27]:
```
corr_df = corr_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)
corr_df = corr_df.dropna()
corr_df = corr_df.loc[corr_df['Importance'] >= 0]
corr_df
```

[27]:

	Features	Importance
0	PROTOCOL	0.263568
1	SHORTEST_FLOW_PKT	0.231396
2	DNS_QUERY_ID	0.224501
3	DNS_QUERY_TYPE	0.183878
4	MIN_IP_PKT_LEN	0.133002
5	L4_DST_PORT	0.084734
6	FLOW_DURATION_MILLISECONDS	0.073958

[29]:
```
with open(file="trained_models/selected_features.pkl", mode="wb") as file:
    pickle.dump(obj=corr_df['Features'].values.tolist(), file=file)
```

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

[30]:

	L4_SRC_PORT	L4_DST_PORT	PROTOCOL	L7_PROTO	IN_BYTES	IN_PKTS	OUT_BYTES	OUT_PKTS	TCP_FLAGS	CLIENT_TCP_FLAGS	SERVER_TCP_FLAGS	FLOW_DURATI
0	60683	53	17	0.0	50	1	50	1	0	0	0	
1	34246	384	6	0.0	48	1	0	0	2	2	0	
2	41289	53	17	0.0	70	1	258	1	0	0	0	
3	49968	80	6	7.0	581	5	667	3	30	30	26	
4	38281	53	17	0.0	108	2	54	1	0	0	0	

[34]:
```
X=df.drop(labels='Attack',axis=1)
X.head()
```

[34]:

	PROTOCOL	SHORTEST_FLOW_PKT	DNS_QUERY_ID	DNS_QUERY_TYPE	MIN_IP_PKT_LEN	L4_DST_PORT	FLOW_DURATION_MILLISECONDS	MAX_IP_PKT_LEN	LONGEST_
0	17	50	55868	28	50	53	0	50	
1	6	48	0	0	0	384	0	48	
2	17	70	5493	28	70	53	0	258	
3	6	52	0	0	52	80	0	555	
4	17	54	27713	1	54	53	4294952	54	

[35]:
```
y=df[['Attack']]
y.head()
```

[35]:

	Attack
0	4
1	3
2	4
3	2
4	4

Data Normalization:

[36]:
```
scaler = MinMaxScaler()
scaler = scaler.fit(X.values)
scaled_X = scaler.transform(X.values)

df = pd.DataFrame(data=scaled_X,columns=X.columns)
df['Attack'] = y.values.ravel()
df.head()
```

[36]:

	PROTOCOL	SHORTEST_FLOW_PKT	DNS_QUERY_ID	DNS_QUERY_TYPE	MIN_IP_PKT_LEN	L4_DST_PORT	FLOW_DURATION_MILLISECONDS	MAX_IP_PKT_LEN	LONGEST_
0	1.0000	0.057592	0.852829	0.651163	0.140845	0.000809	0.000000	0.014946	
1	0.3125	0.052356	0.000000	0.000000	0.000000	0.005859	0.000000	0.013587	
2	1.0000	0.109948	0.083851	0.651163	0.197183	0.000809	0.000000	0.156250	
3	0.3125	0.062827	0.000000	0.000000	0.146479	0.001221	0.000000	0.358016	
4	1.0000	0.068063	0.423041	0.023256	0.152113	0.000809	0.999997	0.017663	

[37]:
```
with open(file='trained_models/scaler.pkl',mode='wb') as file:
    pickle.dump(obj=scaler,file=file)
```

Data splitting:

```
[38]: X = df.drop(labels='Attack', axis=1)
      y = df[['Attack']]
      X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42,shuffle=True,stratify=y)
      print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(40000, 27) (10000, 27) (40000, 1) (10000, 1)
```

Saving the splitted data:

```
[39]: X_train.to_csv('train_test_data/X_train.csv',index=False)
      X_test.to_csv('train_test_data/X_test.csv',index=False)
      y_train.to_csv('train_test_data/y_train.csv',index=False)
      y_test.to_csv('train_test_data/y_test.csv',index=False)
```

---

7. Model Training and Testing Summary

Algorithm 1 - Support Vector Classifier:

```
[7]: from sklearn import svm
     svc_model = svm.SVC()
```

```
[8]: print("="*35,"Model Training","="*35)
     print("Model training started...")
     svc_model.fit(X_train.values, y_train.values.ravel())
     print("Model training completed.",'\n')

     print("="*35,'Model Prediction',"="*35)
     print("Model prediction started...")
     svc_predictions=svc_model.predict(X_test.values)
     print("Model prediction completed.",'\n')
     print("Model predictions:",list(svc_predictions))
```

```
[9]: y_true = y_test.values.ravel().tolist()
     print(y_true)
```

```
[10]: class_labels = ['Benign', 'dos', 'password', 'scanning', 'xss']
```

Accuracy Score

```
[11]: svc_model_accuracy=accuracy_score(y_true=y_true,y_pred=svc_predictions)
      print("Validation accuracy of Support Vector Classifier model is {:.2f}%".format(svc_model_accuracy*100))
```

```
Validation accuracy of Support Vector Classifier model is 78.57%
```

Classification Report

```
[12]: print(classification_report(y_true=y_true,y_pred=svc_predictions, target_names=class_labels))
```

```
              precision    recall  f1-score   support

      Benign       0.89      0.97      0.93      2000
         dos       0.55      0.66      0.60      2000
    password       0.94      0.96      0.95      2000
    scanning       0.99      0.90      0.94      2000
         xss       0.56      0.43      0.49      2000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.78     10000
weighted avg       0.79      0.79      0.78     10000
```

Algorithm 2 - Random Forest Classifier:

```
[15]: from sklearn.ensemble import RandomForestClassifier
      rfc_model = RandomForestClassifier(n_estimators=500)
      rfc_model = rfc_model.fit(X_train.values, y_train.values.ravel())
```

```
[16]: rfc_prediction = rfc_model.predict(X_test.values)
      print(rfc_prediction.tolist())
```

[17]:
```
y_true = y_test.values.ravel().tolist()
print(y_true)
```

Accuracy Score

[18]:
```
rfc_model_accuracy=accuracy_score(y_true=y_true,y_pred=rfc_prediction)
print("Validation accuracy of RandomForestClassifier model is {:.2f}%".format(rfc_model_accuracy*100))
```

```
Validation accuracy of RandomForestClassifier model is 95.61%
```

Classification Report

[19]:
```
print(classification_report(y_true=y_true,y_pred=rfc_prediction, target_names=class_labels))
```

```
              precision    recall  f1-score   support

      Benign       1.00      0.99      1.00      2000
         dos       0.90      0.90      0.90      2000
    password       0.99      0.99      0.99      2000
    scanning       1.00      1.00      1.00      2000
         xss       0.90      0.90      0.90      2000

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

[21]:
```
with open(file="trained_models/RandomForestClassifier_model.pkl", mode="wb") as file:
    pickle.dump(obj=rfc_model, file=file)
```

Algorithm 3 - Artificial Neuron Network:

[22]:
```
from tensorflow.keras.utils import to_categorical
```

[23]:
```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(40000, 27) (10000, 27) (40000, 5) (10000, 5)
```

[24]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout, Input
from tensorflow.keras.regularizers import L2
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping
```

[25]:
```
model = Sequential()

model.add(Input(shape=(X_train.shape[1],)))

model.add(Dense(units=128, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=256, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=256, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=256, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=512, activation='relu', kernel_regularizer=L2(l2=0.0001)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Dense(units=5, activation='sigmoid'))
```

```
# Adding optimizer
optimizer = Adam(learning_rate=0.001)


model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

[26]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               3584

 batch_normalization (BatchN  (None, 128)              512
 ormalization)

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 256)               33024

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 256)               65792

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 dropout_2 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 256)               65792

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 512)               131584

 batch_normalization_4 (Batc  (None, 512)              2048
 hNormalization)

 dropout_4 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 307,973
Trainable params: 305,157
Non-trainable params: 2,816
_________________________________________________________________
```

[27]:
```
history = model.fit(X_train, y_train, batch_size=32, epochs=50, validation_data=(X_test, y_test))
```

```
Epoch 1/50
1250/1250 [==============================] - 5s 3ms/step - loss: 0.7050 - accuracy: 0.7229 - val_loss: 0.5367 - val_accuracy: 0.7672
Epoch 2/50
1250/1250 [==============================] - 4s 3ms/step - loss: 0.5526 - accuracy: 0.7633 - val_loss: 0.5008 - val_accuracy: 0.7650
Epoch 3/50
1250/1250 [==============================] - 4s 3ms/step - loss: 0.5295 - accuracy: 0.7663 - val_loss: 0.4927 - val_accuracy: 0.7712
Epoch 4/50
1250/1250 [==============================] - 3s 3ms/step - loss: 0.5080 - accuracy: 0.7659 - val_loss: 0.4624 - val_accuracy: 0.7876
Epoch 5/50
1250/1250 [==============================] - 4s 3ms/step - loss: 0.4902 - accuracy: 0.7686 - val_loss: 0.4473 - val_accuracy: 0.7665
Epoch 6/50
1250/1250 [==============================] - 4s 3ms/step - loss: 0.4770 - accuracy: 0.7695 - val_loss: 0.4705 - val_accuracy: 0.7548
Epoch 7/50
1250/1250 [==============================] - 3s 3ms/step - loss: 0.4643 - accuracy: 0.7699 - val_loss: 0.4473 - val_accuracy: 0.7623
Epoch 8/50
1250/1250 [==============================] - 3s 3ms/step - loss: 0.4588 - accuracy: 0.7713 - val_loss: 0.4256 - val_accuracy: 0.7799
Epoch 9/50
```

```
[29]: prediction = model.predict(X_test.values, batch_size=32, verbose=1)
      print(prediction)
```

```
313/313 [==============================] - 1s 1ms/step
[[6.13786817e-01 4.11204517e-01 1.86165273e-02 2.73904800e-02
  8.74648690e-01]
 [2.34757662e-02 9.34700906e-01 5.55917621e-02 8.48197937e-03
  9.28197742e-01]
 [8.98978412e-02 5.67724109e-02 9.97257233e-01 7.11001754e-01
  2.00586200e-01]
 ...
 [9.98269677e-01 5.17994165e-04 3.88073921e-03 8.95396829e-01
  9.92864370e-04]
 [2.63957083e-02 9.36978579e-01 6.76130652e-02 6.38335943e-03
  9.26956534e-01]
 [1.41730011e-01 2.06030011e-02 1.19224936e-01 9.98266101e-01
  4.43536043e-02]]
```

```
[30]: predicted_labels = prediction.argmax(axis=1).tolist()
      print(predicted_labels[:50])
```

```
[4, 1, 2, 3, 0, 1, 2, 1, 0, 1, 2, 2, 1, 0, 2, 1, 1, 0, 3, 0, 1, 3, 3, 1, 1, 2, 2, 2, 1, 1, 1, 0, 1, 1, 0, 3, 2, 2, 2, 0, 1, 3, 2, 2, 2, 3, 0, 4, 2, 3]
```

```
[31]: true_labels = y_test.argmax(axis=1).tolist()
      print(true_labels[:50])
```

```
[4, 4, 2, 3, 0, 1, 2, 1, 0, 1, 2, 2, 4, 0, 2, 1, 1, 0, 3, 0, 1, 3, 3, 4, 1, 2, 2, 2, 1, 1, 4, 0, 2, 1, 0, 3, 2, 2, 2, 0, 1, 3, 2, 2, 2, 3, 0, 1, 2, 3]
```

Result Analysis

```
[32]: class_labels = [
          'Normal',
          'Anomly'
      ]
```

```
[33]: class_labels = ['Benign', 'dos', 'password', 'scanning', 'xss']
```

```
[34]: model_accuracy = accuracy_score(
          y_true=true_labels,
          y_pred=predicted_labels
      )

      print(f"Validation accuracy of ArtificialNeuralNetwork model is {model_accuracy*100:.2f}%")
```

```
Validation accuracy of ArtificialNeuralNetwork model is 78.29%
```

```
[35]: print(classification_report(y_true=true_labels,y_pred=predicted_labels , target_names=class_labels))
```

```
              precision    recall  f1-score   support

      Benign       0.98      0.92      0.95      2000
         dos       0.52      0.97      0.67      2000
    password       0.95      0.99      0.97      2000
    scanning       0.94      0.97      0.96      2000
         xss       0.64      0.06      0.12      2000

    accuracy                           0.78     10000
   macro avg       0.81      0.78      0.73     10000
weighted avg       0.81      0.78      0.73     10000
```

8. Inference File:

The script provided below, reads in an Excel file, runs it through a pre-trained model of the Random Forest Classifier for detecting threats, and if detected, it logs the results in the computer, and sends an email alerts to the provided email address.

```
[1]: import warnings
     warnings.filterwarnings("ignore")
     import os
     import pandas as pd
     import numpy as np
     import yagmail
     import pickle
     from joblib import dump, load
```

```
[2]: with open(file='trained_models/RandomForestClassifier_model.pkl',mode='rb') as file:
         rfc_model=pickle.load(file=file)
```

```
[3]: class_labels = ['Benign', 'dos', 'password', 'scanning', 'xss']
     print(class_labels)
```

```
['Benign', 'dos', 'password', 'scanning', 'xss']
```

```
[4]: filename='user_input/file3.xlsx'
```

Setting up the Excel file, for testing, should be done in the following way:

i. Take the header row from X-testing file which is used as a header of a newly created Excel worksheet.

ii. And select any row between 2nd and nth positions from x test file and paste it below the header.

iii. At the top of first column, create a new column and name it as “IP_ADD” in the first cell.

iv. Type any random IP address below the IP_ADD (in the first column <-> second row).

v. Save the excel file for testing purposes only.

```
[5]: df=pd.read_excel(filename)
     df.head()
```

[5]:

	IP_ADD	PROTOCOL	SHORTEST_FLOW_PKT	DNS_QUERY_ID	DNS_QUERY_TYPE	MIN_IP_PKT_LEN	L4_DST_PORT	FLOW_DURATION_MILLISECONDS	MAX_IP_PKT_L
0	192.102.16.101	1	0.115183	0.790472	0.27907	0	0.000809	0	0.029

1 rows × 28 columns

```
[6]: # Drop 'IP_Add' column
     input_data = df.drop(columns=['IP_ADD'])
```

```
[7]: input_data.head()
```

[7]:

	PROTOCOL	SHORTEST_FLOW_PKT	DNS_QUERY_ID	DNS_QUERY_TYPE	MIN_IP_PKT_LEN	L4_DST_PORT	FLOW_DURATION_MILLISECONDS	MAX_IP_PKT_LEN	LONGEST_
0	1	0.115183	0.790472	0.27907	0	0.000809	0	0.029891	

1 rows × 27 columns

```
[8]: input_data.values
```

```
[8]: array([[1.00000000e+000, 1.15183246e-001, 7.90471538e-001,
             2.79069767e-001, 0.00000000e+000, 8.08728161e-004,
             0.00000000e+000, 2.98913043e-002, 2.98913043e-002,
             0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
             3.77635582e-003, 0.00000000e+000, 0.00000000e+000,
             0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
             0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
             4.16032867e-004, 0.00000000e+000, 0.00000000e+000,
             0.00000000e+000, 0.00000000e+000, 6.14150650e-164]])
```

```
[9]: rfc_prediction = rfc_model.predict(input_data.values)
     print(rfc_prediction.tolist())
     ClassIndex = rfc_prediction[0]
     ClassLabel = class_labels[rfc_prediction[0]]
```

```
[4]
```

```
[10]: print(f'Model predicted class is: {ClassIndex}')
      print(f'Model predicted label is: {ClassLabel}')
```

```
Model predicted class is: 4
Model predicted label is: xss
```

```
[11]: ip = df['IP_ADD'].tolist()
      print(ip[0])
```

```
192.102.16.101
```

```
[12]: def update_logfile(ip_address=None, predicted_attack=None):
          new_data = {'IP Address': [str(ip_address).strip()],
                      'Found Attack': [predicted_attack]}
          new_row_df = pd.DataFrame(new_data)

          try:
              df = pd.read_csv("LOG.csv")
          except FileNotFoundError:
              df = pd.DataFrame(columns=['IP Address', 'Found Attack'])

          # df = df.append(new_row_df, ignore_index=True)
          df = pd.concat([df, new_row_df], ignore_index=True)
          df.to_csv("LOG.csv", index=False)
          return True
```

```
[13]: if ClassLabel != 'Benign':
          update_logfile(ip_address=ip[0], predicted_attack=ClassLabel)
          print("its a attack")
      else:
          print("not a attack")
```

```
its a attack
```

[14]:
```
def send_email(receiver, subject, content, file_path):
    try:
        yag = yagmail.SMTP('vinothvignesh624612@gmail.com', 'adjcudvripsohfnt')
        yag.send(to=receiver, subject=subject, contents=[content, file_path])
        print("Email sent successfully")
        return True
    except Exception as e:
        print(f"Failed to send email: {e}")
        return False
```

[15]:
```
recipient_email = "vigneshvickyodc01@gmail.com"
email_content = f"""
                <p>Your uploaded file is classified as <strong>"Abnormal"</strong> with a <strong>{ClassLabel}</strong> attack detected. Immediate action

                Best regards,
                <br>
                Vignesh <br>
                <strong>Attack File is below</strong>
            """
if ClassLabel in class_labels[1:]:
    email_sent = send_email(receiver=recipient_email, subject='$$ Suspicious Activity Detected $$', content=email_content, file_path=filename)
    if email_sent:
        emailAlert = f"Email sent successfully to {recipient_email}"
        print(emailAlert)
    else:
        emailAlert = f"Failed to send email to {recipient_email}"
        print(emailAlert)
```

```
Email sent successfully
Email sent successfully to vigneshvickyodc01@gmail.com
```

9. References

Best Python libraries for Machine Learning. (2019, 01 18). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

Download Anaconda Distribution. (n.d.). Retrieved from Anaconda: https://www.anaconda.com/download

Mohanad Sarhan. (n.d.). *Dataset File*. Retrieved from https://rdm.uq.edu.au/files/a4ad7080-ef9c-11ed-a964-b70596e96ad5

Send Email Using yagmail in Python. (2024, 06 28). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/send-email-using-yagmail-in-python/