Configuration Manual for Network Intrusion Detection System Using Supervised and Deep Learning Machine Learning Algorithms

MSc Academic Research Project
Cybersecurity

Ayodele Oluwagbayi Jolayemi
Student ID: x21139288

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ayodele Oluwagbayi Jolayemi……………………………………………………………

Student ID: x21139288……………………………………………………………………………………

Programme: Cybersecurity…………………………………………… **Year:** 2024…………………..

Module: Msc Research Project Configuration Manual………………………………………

Lecturer: Rejwanul Haque………………………………………………………………………………

Submission Due Date: 31/Jan/2024……………………………………………………………………………………

Project Title: Network Intrusion Detection using Supervised and deep Learning Machine Learning Algorithms……………………………………………………………

Word Count: 2165…………………………… **Page Count:** 26…………………………………………

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: Ayodele Oluwagbayi Jolayemi…………………………………………………………

Date: 31/Jan/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):	

Network Intrusion Detection using Supervised and Deep Learning Machine Learning Algorithms

Ayodele Oluwagbayi Jolayemi
Student ID: x21139288

1 Introduction

The purpose of this report is to create a comprehensive manual that will serve as a guide for creating the experiment setup and coding documentation for the research project titled "Network Intrusion Detection using Supervised and Deep Learning Machine Learning Algorithms.". The purpose of the research was to find out how well-supervised learning and deep learning machine learning algorithms can use historical datasets to classify network traffic as either attack or non-attack traffic. The experiment was conducted using two (2) datasets, namely the UNSW NB15 dataset and the CICIDS 2017 dataset, which were implemented as code. This study employed two deep learning algorithms, Convolution Neural Network, and Recurrent Neural Network, and three supervised learning algorithms, namely Logistic Regression, Naïve Bayes, and Decision Tree Classifier, to analyse the datasets.

The rest of the documentation is structured: Section 2 covers system requirements for the hardware and software components used in this code implementation; Section 3 covers software installation, setting up the Anaconda environment, and installing the Python libraries used in this code implementation. The implementation of the models' code, their assessment, and how well the models worked with each dataset's analysis are covered in Section 4. The final or concluding remarks are covered in Section 5, and a list of references is included in Section 6.

2 System Requirement

2.1 Hardware Requirement

Below is the hardware specification used to develop the code implementation

RAM 8 GB 2133 MHz LPDDR3

Processor 2.3 GHz Quad-Core Intel

Storage 512 GB SSD

2.2 Software Requirement

Mac OS Sonoma 14.0 serves as the stable foundation for our project's software environment. Its compatibility and versatility enable efficient integration of diverse software, enhancing project effectiveness and efficiency.

- Anaconda Navigator 2.5.0 - is a package and environment manager for Python, streamlines Python version management and workspace isolation on one machine. It's crucial for data scientists, developers, and researchers, enhancing reproducibility and workflow efficiency.
- Jupyter Notebook 6.5.4 - is a web-based interactive IDE for coding and data analysis. Users can write, run, and visualize code in real-time, making it versatile, collaborative, and popular among researchers and data scientists.
- Google Chrome Browser 119.0 - Jupyter Notebook uses google chrome to render the IDE both the code snippet and its corresponding output.

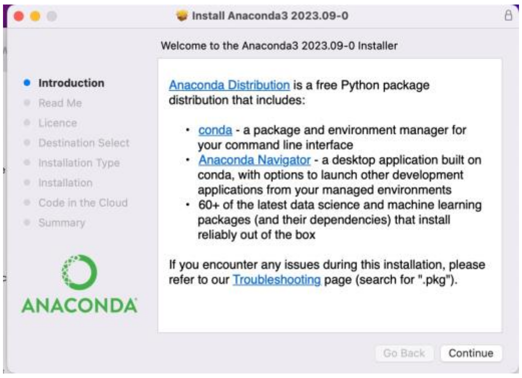
3 Software Installation and Python Libraries

This section will cover the installation of the Anaconda Navigator software and provide a comprehensive list of Python libraries required for the successful implementation of this research project coding.

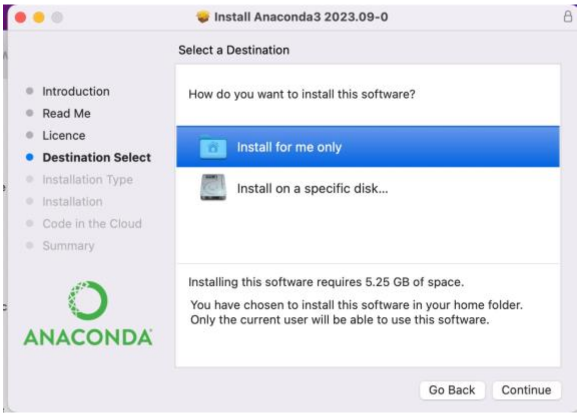
3.1 Anaconda Navigator Installation

To set up Anaconda on your Mac OS, acquire the Mac OS Anaconda installer from the official Anaconda website. After successfully downloading the installer file, find it on your computer and then follow the instructions provided below.

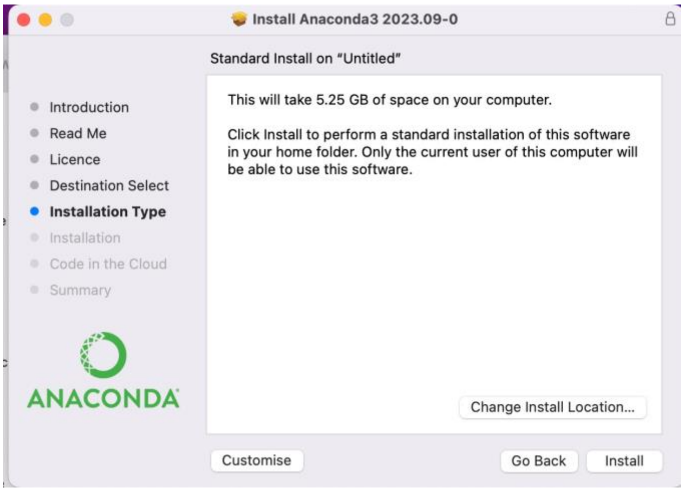
- Open the downloaded file by double-clicking it, and then click "Continue" to initiate the installation process.



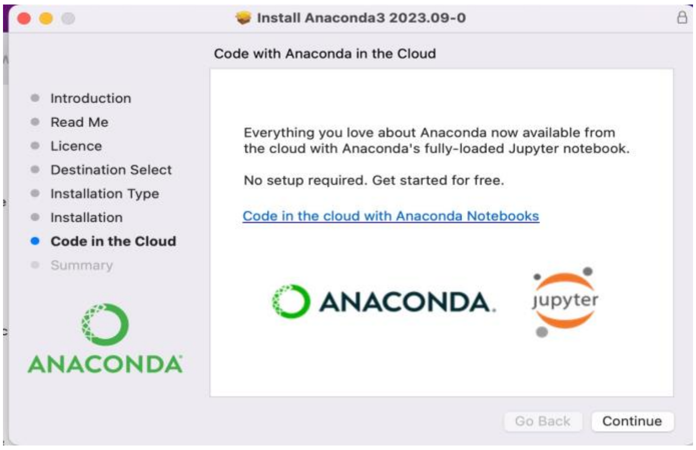
- Respond to the inquiries presented on the Introduction, Read Me, and License screens
- Anaconda suggests selecting "Install for me only." If you prefer not to install Anaconda Distribution in your home folder, you can opt for "Install on a specific disk." Then, proceed by clicking the "Install" button.



- Click **Install**.



- Once the install is complete, click **Continue**.
- Optional: To learn more about Anaconda's cloud notebook service, go to



Or click **Continue** to proceed.

- A successful installation displays the following screen, click **Close** to exit installer.

3.2 Python Library Installation

The Anaconda installation has established a "base" environment, which comes pre-equipped with essential Python libraries required for the project's coding implementation. Additional libraries will be installed to fully configure the environment for the coding exercises. The Table 1 below show the list of all Python libraries that must be install before starting code

Python Library	Conda Installation Command
pandas	`conda install -c anaconda pandas`
seaborn	`conda install -c conda-forge seaborn`
scikit-learn	`conda install -c intel scikit-learn`
scikit-learn-intelex	`conda install scikit-learn-intelex`
tqdm	`conda install -c conda-forge tqdm`
tensorflow	`conda install -c conda-forge tensorflow`
scikeras	`pip install scikeras`

Table 1. Python library names and their installation command

4 Implementation and Evaluation

Following a successful installation of Anaconda and the installation of all the necessary libraries for this code implementation, this section presents the actual code implementation for the research project, as depicted below:

4.1 Start a new project

From the main Anaconda page, activate the Jupyter Notebook by clicking the "Launch" button within the cell. This action will open the interactive Python IDE in your web browser.

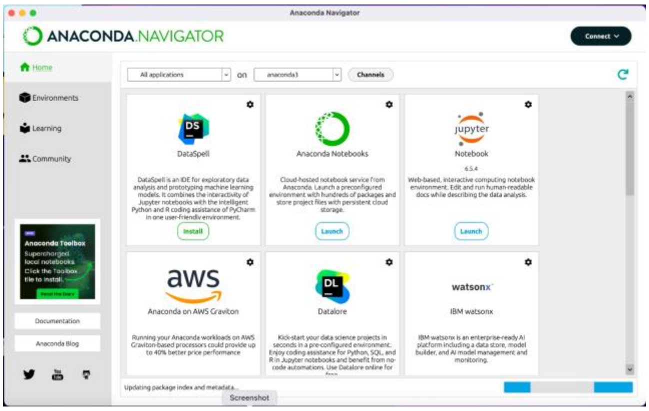


Figure 1. Anaconda navigator launch screen

On the Jupyter Notebook startup page, find the "New" button situated in the upper right corner. Click on it and then select "Python 3" to initiate a new project.

4.2 Import Python Libraries

Upon successfully initiating a new project, the initial code snippet will include all Python libraries utilized in this project. Whenever this block is modified, click the "Run" button to import the library into the IDE.

Import Python's Libraries

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

        import os
        import sys
        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

        import platform
        import time

        import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        from tqdm import tqdm

        from sklearn.model_selection import train_test_split, GridSearchCV
        from sklearn.preprocessing import MinMaxScaler, StandardScaler

        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import GaussianNB
        from sklearn.tree import DecisionTreeClassifier

        from sklearn.ensemble import ExtraTreesClassifier

        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout, MaxPooling1D, Flatten, Activation
        from tensorflow.keras.layers import LSTM,Convolution1D
        from tensorflow.keras import callbacks
        from scikeras.wrappers import KerasClassifier

        from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, confusion_matrix
        from sklearn.exceptions import ConvergenceWarning

        from IPython.core.interactiveshell import InteractiveShell
        from IPython.core.display import HTML as Center

        Center("""
            <style>
                .output_png {
                    display: table-cell;
                    text-align: center;
                    vertical-align: middle;
                }
            </style>
        """)

        %matplotlib inline
        plt.style.use("ggplot")
```

Figure 2. Imported Python libraries code snippets

4.3 Global Variables and Helper Functions

Efficient memory utilization, simplified code maintenance, code reuse, and reduction of code redundancy are achieved through the optimization strategy of defining global variables and helper functions.

Global Variables Declaration

```
In [2]: InteractiveShell.ast_node_interactivity = "all"
        pd.options.display.max_seq_items = 200
        pd.options.display.max_rows = 200

        color = ['#008800','#AA0000', '#ff9999','#66b3ff','#99ff99','#ffcc99']
        object_columns = ["srcip", "dstip", "proto", "state", "service", "ctftpcmd", "trafficcategory"]
        int_columns = ["sport", "dsport"]
        object_columns_backup = ["srcip", "sport", "dstip", "dsport", "proto", "state", "service", "ctftpcmd", "trafficcatego
        zero_columns = ["bwdpshflags", "bwdurgflags", "fwdavgbytes/bulk", "fwdavgpackets/bulk", "fwdavgbulkrate", "bwdavgbyte

        cicids = "CICIDS_2017"
        unsw_nb15 = "UNSW_NB15"
        experimet_1 = "EXPERIMENT_I"
        experimet_2 = "EXPERIMENT_II"

        attack_type = "trafficcategory"
        dependent_variable = "traffictype"

        base_directory = os.getcwd()
        datasets_directory = os.path.join(base_directory, "datasets")
        cicids_2017_dataset_directory = os.path.join(datasets_directory, cicids)
        unsw_nb15_dataset_directory = os.path.join(datasets_directory, unsw_nb15)
        dataset_directory_list = [cicids_2017_dataset_directory, unsw_nb15_dataset_directory]

        loaded_dataset_names = list()
        dfs = dict()

        standard_scaler = MinMaxScaler()

        experiment_results = {
            cicids: {
                experimet_1: dict(),
                experimet_2: dict()
            },
            unsw_nb15: {
                experimet_1: dict(),
                experimet_2: dict()
            }
        }

        data_result_list = [cicids, unsw_nb15]
        experi_result_list = [experimet_1, experimet_2]
```

Figure 3. Declared global variables code snippets

```
In [5]: def show_pie_chart_visualization(df, labels, chart_title, dependent_variable):
            color_dist = {}

            count = df[dependent_variable].value_counts().to_frame().sort_index()

            for index in count.index:
                color_dist[index] = color[index]

            fig = plt.figure(figsize=(5,5), dpi=144)
            traffic_classification = [df[dependent_variable].value_counts()[0], df[dependent_variable].value_counts()[1]]
            plt.title(chart_title)
            plt.pie(traffic_classification, labels=labels, autopct='%1.1f%%', colors=[color_dist[c] for c in count.index])
            plt.axis('equal')
            plt.legend(loc = 0)
            plt.show()
```

Figure 4. Function definition for generating pie chart showing the binary class distribution in the dependent variable

```
In [7]: def balance_dataset_evenly(df, dependent_variable):
            category_a = df[df[dependent_variable] == 0]
            category_b = df[df[dependent_variable] == 1]

            cat_a_record_count = category_a.shape[0]
            cat_b_record_count = category_b.shape[0]

            cat_a_record_count = 15000
            category_a = category_a.sample(n=cat_a_record_count)

            if cat_a_record_count <= cat_b_record_count:
                category_b = category_b.sample(n=cat_a_record_count)
            else:
                category_a = category_a.sample(n=cat_b_record_count)

            df = pd.concat([category_a, category_b], ignore_index=True)

            return df
```

Figure 5. Function definition for generating a balanced binary class distribution in the dependent variable

```
In [8]: def convert_pandas_object_to_int_type_using_mapping(df, columns):
            for column in columns:
                df[column] = df[column].map(create_map_for_given_category(df[column].unique()))
            return df
```

```
In [9]: def create_map_for_given_category(category_list):
            mapping = dict()
            for category_index in range(len(category_list)):
                mapping[category_list[category_index]] = category_index
            return mapping
```

Figure 6. Functions definition for convert string variable to numeric variable using mapping operations

```
In [10]: def show_correlation_heatmap_visualization(df):
             corr = df.corr()
             mask = np.triu(np.ones_like(corr, dtype=bool))
             cmap = sns.diverging_palette(230, 20, as_cmap=True)
             plt.figure(figsize=(25,20))
             sns.heatmap(corr, mask=mask, cmap=cmap, annot=True, annot_kws={"size": 14}, linewidths=0.5)
             sns.set_style("white")
             plt.xticks(fontsize=14)
             plt.yticks(fontsize=14)
             plt.show()
```

Figure 7. Function definition for generating correlation heatmap for a given data frame

```
In [11]: def perform_multicollinearity_analysis(df, threshold):
             columns = set()
             corr_estimate_for_features = dict()
             corr = df.corr()

             for i in range(len(corr.columns)):
                 for j in range(i):
                     corr_estimate = corr.iloc[i,j]
                     if corr_estimate >= threshold or corr_estimate <= (threshold * (-1)):
                         column = corr.columns[i]
                         columns.add(column)
                         # corr_estimate_for_features[column] = corr_estimate
                         corr_estimate_for_features[column] = "{} - {}: {}".format(column, corr.columns[j], corr_estimate)
             return columns, corr_estimate_for_features
```

Figure 8. Function definition for perform multicollinearity analysis using correlation estimates for a given data frame and a specified threshold

```
In [12]: def split_dataset_into_training_and_testing_set(independent_variables, dependent_variable):
             scaled_independent_variables = standard_scaler.fit_transform(independent_variables)
             indep_train, indep_test, dep_train, dep_test = train_test_split(scaled_independent_variables, dependent_variable,
             return indep_train, indep_test, dep_train, dep_test
```

Figure 9. Function definition for scaling and splitting a dataset into training and testing sets

```
In [13]: def show_model_analysis_summary(actual_values, predicted_values, model_info):
             summary_header = "======== {} ========".format(model_info.upper())

             print("\n\n")
             print(summary_header)
             print(make_horizontal_line(len(summary_header)))
             print(make_horizontal_line(len(summary_header)))
             print("\n\n")

             show_confusion_matrix_visualization(actual_values, predicted_values)
             print("\n\n")

             estimated_accuracy_score = accuracy_score(actual_values, predicted_values)
             estimated_f_score = f1_score(actual_values, predicted_values)
             estimated_auc_score = accuracy_score(actual_values, predicted_values)
             evaluation_title = "EVALUATION METRIC SUMMARY"
             print(evaluation_title)
             print(make_horizontal_line(len(evaluation_title)))
             print(f"""ACCURACY      :   {estimated_accuracy_score:.4f}""")
             print(f"""F1 SCORE      :   {estimated_f_score:.4f}""")
             print(f"""AUC SCORE     :   {estimated_auc_score:.4f}""")
             print("\n\n\n\n\n\n\n")

             return {
                 "accuracy": estimated_accuracy_score,
                 "f1_score": estimated_f_score,
                 "auc_score": estimated_auc_score
             }
```

Figure 10. Function definition for generating summary analysis report for a give machine learning model

```
[15]: def run_logistic_regression_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          model = LogisticRegression(random_state=42)
          model.fit(train_x, train_y.values.ravel())
          predictions = model.predict(test_x)
          model_info = "Logistic Regression Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

```
[16]: def run_hyper_parameters_tunned_logistic_regression_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          param_grid = {
              'penalty' : ['l1', 'l2', 'elasticnet'],
              'C' : np.logspace(-4, 4, 20),
              'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
              'max_iter' : [2500, 5000]
          }
          model = LogisticRegression(random_state=42)
          model = GridSearchCV(model, param_grid = param_grid, cv = 5, verbose=0, n_jobs=-1)
          model.fit(train_x, train_y.values.ravel())
          predictions = model.predict(test_x)
          model_info = "Logistic Regression Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

Figure 11. Functions definition for performing Logistic Regression analysis for regular and hyper-parameter tuned models

```python
[17]: def run_naive_bayes_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          model = GaussianNB()
          model.fit(train_x, train_y.values.ravel())
          predictions = model.predict(test_x)
          model_info = "Naive Bayes Classifier Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

```python
[18]: def run_hyper_parameters_tunned_naive_bayes_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          param_grid = {
              'var_smoothing': np.logspace(0,-9, num=100)
          }
          model = GaussianNB()
          model = GridSearchCV(model, param_grid = param_grid, cv = 5, verbose=0, n_jobs=-1)
          model.fit(train_x, train_y.values.ravel())
          predictions = model.predict(test_x)
          model_info = "Naive Bayes Classifier Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

Figure 12. Functions definition for performing Naïve Bayes analysis for regular and hyper-parameter tuned models

```python
[19]: def run_decision_tree_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          model = DecisionTreeClassifier(max_depth=10, criterion='entropy', random_state=42)
          model.fit(train_x, train_y.values.ravel())
          predictions = model.predict(test_x)
          model_info = "Decision Tree Classifier Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

```python
[20]: def run_hyper_parameters_tunned_decision_tree_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          param_grid = {
              "criterion": ["gini", "entropy"],
              "max_depth": range(1, 10),
              "min_samples_split": range(1, 10),
              "min_samples_leaf": range(1, 5)
          }
          model = DecisionTreeClassifier(random_state=42)
          model = GridSearchCV(model, param_grid = param_grid, cv = 5, verbose=0, n_jobs=-1)
          model.fit(train_x, train_y.values.ravel())
          predictions = model.predict(test_x)
          model_info = "Decision Tree Classifier Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

Figure 13. Functions definition for performing Decision Tree analysis for regular and hyper-parameter tuned models

```python
[25]: def run_convolution_neural_network_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          batch_size = 32
          epochs = 10
          x_train = np.reshape(train_x, (train_x.shape[0], train_x.shape[1],1))
          x_test = np.reshape(test_x, (test_x.shape[0], test_x.shape[1],1))
          model = KerasClassifier(lambda: build_convolution_neural_network(train_x.shape[1]), epochs=epochs, batch_size=batch_size)
          model.fit(x_train, train_y)
          predictions = model.predict(x_test)
          model_info = "Convolution Neural Network Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

```python
[26]: def run_hyper_parameters_tunned_convolution_neural_network_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
          param_grid = {
              'batch_size': [256, 128, 512],
              'epochs': [30, 20, 10],
              'unit': [16, 32, 64],
              'dim': [train_x.shape[1]]
          }
          x_train = np.reshape(train_x, (train_x.shape[0], train_x.shape[1],1))
          x_test = np.reshape(test_x, (test_x.shape[0], test_x.shape[1],1))
          model = KerasClassifier(build_fn=build_hyper_parameters_convolution_neural_network, dim=[train_x.shape[1]], unit=[32, 16, 8]
          model = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
          model.fit(x_train, train_y)
          predictions = model.predict(x_test)
          model_info = "Convolution Neural Network Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
          evaluations = show_model_analysis_summary(test_y, predictions, model_info)
          return evaluations
```

Figure 14. Functions definition for performing Convolution Neural Network (CNN) analysis for regular and hyper-parameter tuned models

```
[27]: def run_recurrent_neural_network_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
    batch_size = 32
    epochs = 10
    x_train = np.reshape(train_x, (train_x.shape[0], 1, train_x.shape[1]))
    x_test = np.reshape(test_x, (test_x.shape[0], 1, test_x.shape[1]))
    model = KerasClassifier(lambda: build_recurrent_neural_network(train_x.shape[1]), epochs=epochs, batch_size=batch_size)
    model.fit(x_train, train_y)
    predictions = model.predict(x_test)
    model_info = "Recurrent Neural Network Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
    evaluations = show_model_analysis_summary(test_y, predictions, model_info)
    return evaluations
```

```
[28]: def run_hyper_parameters_tunned_recurrent_neural_network_analysis(train_x, train_y, test_x, test_y, dataset, experiment):
    param_grid = {
        'batch_size': [256, 128, 512],
        'epochs': [30, 20, 10],
        'unit': [16, 32, 64],
        'dim': [train_x.shape[1]]
    }
    x_train = np.reshape(train_x, (train_x.shape[0], 1, train_x.shape[1]))
    x_test = np.reshape(test_x, (test_x.shape[0], 1, test_x.shape[1]))
    model = KerasClassifier(build_fn=build_hyper_parameters_recurrent_neural_network, dim=[train_x.shape[1]], unit=[32, 16, 8])
    model = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
    model.fit(x_train, train_y)
    predictions = model.predict(x_test)
    model_info = "Recurrent Neural Network Model Analysis Summary for {} Dataset ({})".format(dataset, experiment)
    evaluations = show_model_analysis_summary(test_y, predictions, model_info)
    return evaluations
```

Figure 15. Functions definition for performing Recurrent Neural Network (RNN) analysis for regular and hyper-parameter tuned models

```
In [29]: def perform_feature_selection_using_feature_importance(independent_variables, dependent_variable, num_features=10):
    model = ExtraTreesClassifier(verbose=0, random_state=42, n_jobs=-1)
    num_features = int(independent_variables.shape[1] * 0.66)

    tuned_model = ExtraTreesClassifier(verbose=0, random_state=42, n_jobs=-1)
    tuned_model.fit(independent_variables, dependent_variable)
    print(tuned_model.feature_importances_)

    feat_importances = pd.Series(tuned_model.feature_importances_, index=independent_variables.columns)
    feat_importances.nlargest(num_features).plot(kind='barh')
    plt.show()

    return feat_importances.nlargest(num_features).index
```

Figure 16. Function definition for performing feature selection using feature importance for a given data frame

```
In [30]: def show_implemented_models_evaluation_tables(analysis_summary, models, metrics):
    table = ""
    max_table_width = 100
    num_column = len(metrics) + 1
    content_width = int(max_table_width / num_column)
    if max_table_width > (content_width * num_column):
        max_table_width = (content_width * num_column)
    next_line = "\n"

    horizontal_line = ""
    for count in range(max_table_width):
        horizontal_line += "="

    horizontal_line += next_line

    table += horizontal_line
    pos = 1

    table += draw_table_content("", pos, num_column, content_width)
    for idx in range(len(metrics)):
        pos += 1
        table += draw_table_content(" ".join(metrics[idx].upper().split("_")), pos, num_column, content_width)

    table += horizontal_line

    for label in models:
        pos = 1
        table += draw_table_content(label, pos, num_column, content_width)
        for idx in range(len(metrics)):
            pos += 1
            value = "{:.4f}".format(analysis_summary[label][metrics[idx]])
            table += draw_table_content(value, pos, num_column, content_width)

        table += horizontal_line

    print(table)
    print()
```

Figure 17. Function definition for generating summary table for all implemented model for all datasets and experiment performed

4.4 Load Datasets

During the code implementation, two distinct datasets, namely the CICIDS 2017 dataset and the UNSW NB15 dataset, were employed for analyzing the implemented models. These datasets were obtained from the Canadian Institute for Cybersecurity website and the UNSW website. The Python Pandas library was utilized to load these datasets into the Jupyter Notebook IDE as Pandas data frames.

Load Datasets ¶

```
In [32]: for directory in dataset_directory_list:
             try:
                 file_list = os.listdir(directory)
                 dataset_name = get_directory_name(directory)
                 loaded_dataset_names.append(dataset_name)
                 dfs[dataset_name] = None
                 file_counter = 0

                 print()
                 print("PROCESSING {} DATA FILES".format(dataset_name))
                 print()

                 if len(file_list) > 0:
                     for file in tqdm(file_list, total=len(file_list)):
                         if file_counter == 0:
                             dfs[dataset_name] = pd.read_csv(os.path.join(directory, file), sep=",")
                         else:
                             df = pd.read_csv(os.path.join(directory, file), sep=",")
                             dfs[dataset_name] = pd.concat([dfs[dataset_name], df], ignore_index=True)

                         file_counter += 1

             except Exception as e:
                 print("Error Loading data files: ")
                 print(dir(e))
                 print("\n\n")
                 print(e.args)
                 print("\n\n")
                 print(e.with_traceback())

PROCESSING CICIDS_2017 DATA FILES

100%|██████████| 8/8 [00:17<00:00,  2.18s/it]

PROCESSING UNSW_NB15 DATA FILES

100%|██████████| 4/4 [00:12<00:00,  3.12s/it]
```

Figure 18. Code snippets used to import dataset as Panda's data frame to for performing analysis

4.5 Data Preprocessing and Exploratory Data Analysis

```
In [34]: for df_name in loaded_dataset_names:
             clean_dataframe_column_names(dfs[df_name])

In [35]: for df_name in loaded_dataset_names:
             for col_name in object_columns:
                 if col_name in dfs[df_name].columns:
                     dfs[df_name][col_name] = dfs[df_name][col_name].astype(str)

In [36]: dfs['CICIDS_2017'][dependent_variable] = dfs['CICIDS_2017']["label"].apply(lambda x: 0 if x == "BENIGN" else 1)
         dfs['CICIDS_2017'] = dfs['CICIDS_2017'].drop(["label"], axis=1)

In [37]: dfs['UNSW_NB15'].rename(columns={"attackcat": attack_type, "label": dependent_variable}, inplace = True)
```

Figure 19. Code snippets used to (i) clean columns in the two datasets, (ii) enforce data integrity for IP address columns, (iii) create dependent variable column in CICIDS dataset and rename columns in UNSW NB15 dataset

```python
[46]: dfs['CICIDS_2017'] = dfs['CICIDS_2017'].replace([np.inf, -np.inf], np.nan)
      dfs['CICIDS_2017'].dropna(inplace = True)
      dfs['CICIDS_2017'] = dfs['CICIDS_2017'].drop(zero_columns, axis=1)
```

```python
[47]: dfs['UNSW_NB15'] = dfs['UNSW_NB15'].replace([np.inf, -np.inf], np.nan)
      dfs['UNSW_NB15'] = dfs['UNSW_NB15'].drop(["ctflwhttpmthd" ,"isftplogin", "trafficcategory"], axis=1)
      dfs['UNSW_NB15'].dropna(inplace = True)
```

Figure 20. Code snippets for handling missing data in both datasets

```python
In [50]: labels = [
             "Normal Traffics",
             "Intrusion Attrack Traffics"
         ]

         chart_title = "Network Traffic Binary Class Distribution Pie Chart for CICIDS 2017 Dataset"

         show_pie_chart_visualization(dfs['CICIDS_2017'], labels, chart_title, dependent_variable)

         print("\n\n\n\n\n")
```

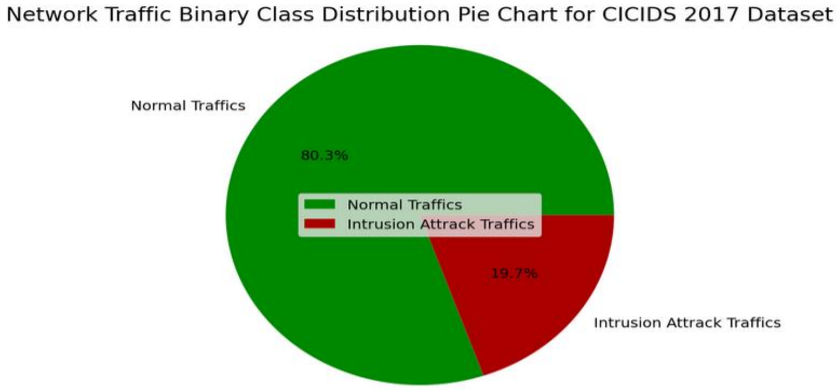


Figure 21. Code Snippets for showing the normal traffics to intrusion attack traffic ratio in CICIDS 2017 dataset

```python
In [53]: labels = [
             "Normal Traffics",
             "Intrusion Attrack Traffics"
         ]

         chart_title = "Network Traffic Binary Class Distribution Pie Chart for UNSW NB15 Dataset"

         show_pie_chart_visualization(dfs['UNSW_NB15'], labels, chart_title, dependent_variable)

         print("\n\n\n\n\n")
```

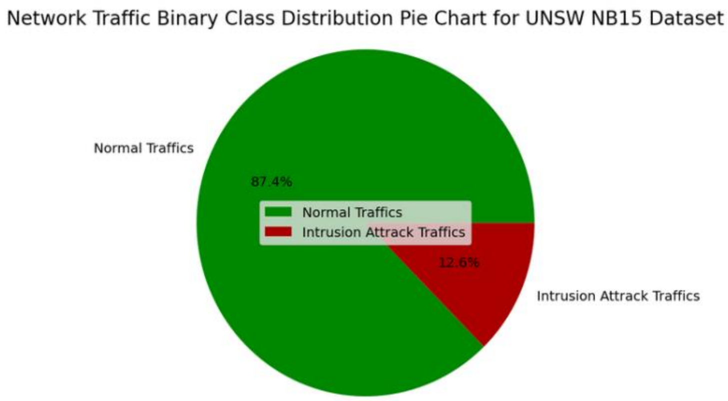


Figure 22. Code Snippets for showing the normal traffics to intrusion attack traffic ratio in UNSW NB15 dataset

```
In [56]: if len(dfs['CICIDS_2017'].select_dtypes(include=['object']).columns):
             dfs['CICIDS_2017'] = convert_pandas_object_to_int_type_using_mapping(dfs['CICIDS_2017'], dfs['CICIDS_2017'].selec

In [57]: if len(dfs['UNSW_NB15'].select_dtypes(include=['object']).columns):
             dfs['UNSW_NB15'] = convert_pandas_object_to_int_type_using_mapping(dfs['UNSW_NB15'], dfs['UNSW_NB15'].select_dtyp
```

Figure 23. Code snippet for performing mapping on Panda's object data type to numeric data type

```
In [62]: cicids_columns, cicids_corr_estimate_for_features = perform_multicollinearity_analysis(dfs['CICIDS_2017'].drop([depen

In [63]: print("\n\n")

         label_text = "Highly Correlated Features From {} Dataset".format(" ".join("CICIDS_2017".split("_")))
         cicids_columns = list(cicids_columns)
         print(label_text)
         print(make_horizontal_line(len(label_text)))
         print(make_horizontal_line(len(label_text)))
         print("")
         for col_indx in range(len(cicids_columns)):
             print("{}.\t{}".format((col_indx+1), "{}".format(cicids_corr_estimate_for_features[cicids_columns[col_indx]])))

         print("\n\n\n\n")
```

Figure 24. Code snippets performing multicollinearity analysis on CICIDS 2017 dataset

```
In [65]: unsw_nb15_columns, unsw_nb15_corr_estimate_for_features = perform_multicollinearity_analysis(dfs['UNSW_NB15'].drop([dependent

In [66]: print("\n\n")

         label_text = "Highly Correlated Features From {} Dataset".format(" ".join("UNSW_NB15".split("_")))
         unsw_nb15_columns = list(unsw_nb15_columns)
         print(label_text)
         print(make_horizontal_line(len(label_text)))
         print(make_horizontal_line(len(label_text)))
         print("")
         for col_indx in range(len(unsw_nb15_columns)):
             print("{}.\t{}".format((col_indx+1), "{}".format(unsw_nb15_corr_estimate_for_features[unsw_nb15_columns[col_indx]])))

         print("\n\n\n\n")
```

Figure 25. Code snippets performing multicollinearity analysis on UNSW NB15 dataset

```
In [67]: dfs['UNSW_NB15'].drop(unsw_nb15_columns, axis=1,  inplace = True)
```

```
In [68]: show_correlation_heatmap_visualization(dfs['CICIDS_2017'].drop([dependent_variable], axis=1))
```

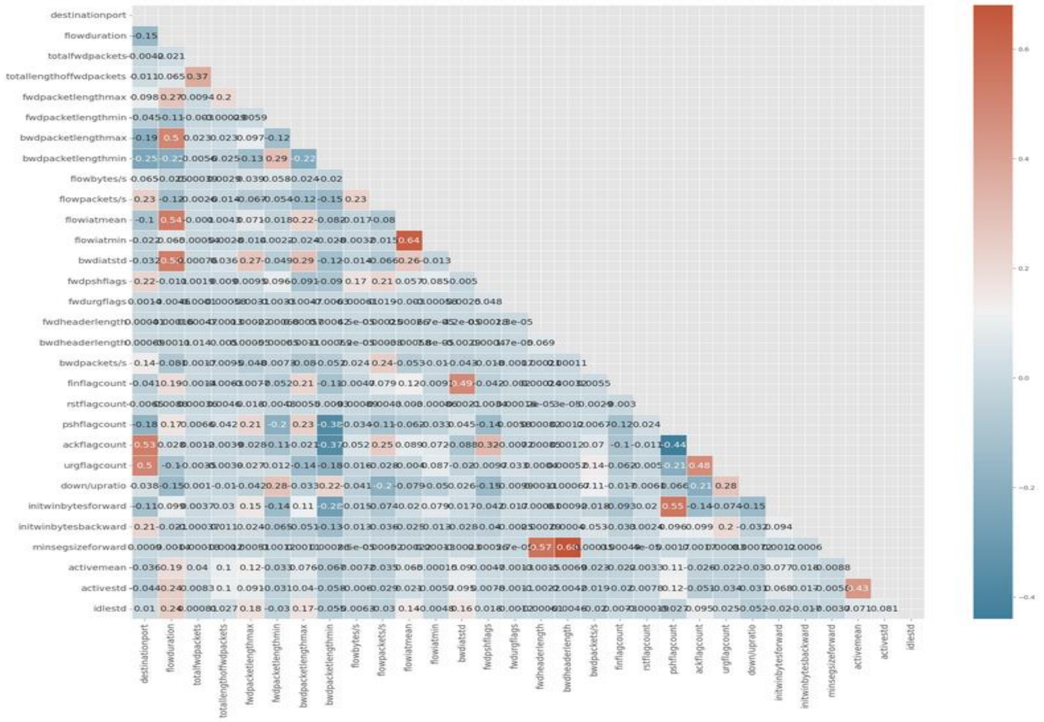


Figure 26. Correlation heatmap after removing highly correlated independent variables from CICIDS 2017 dataset

```
In [69]: show_correlation_heatmap_visualization(dfs['UNSW_NB15'].drop([dependent_variable], axis=1))
```

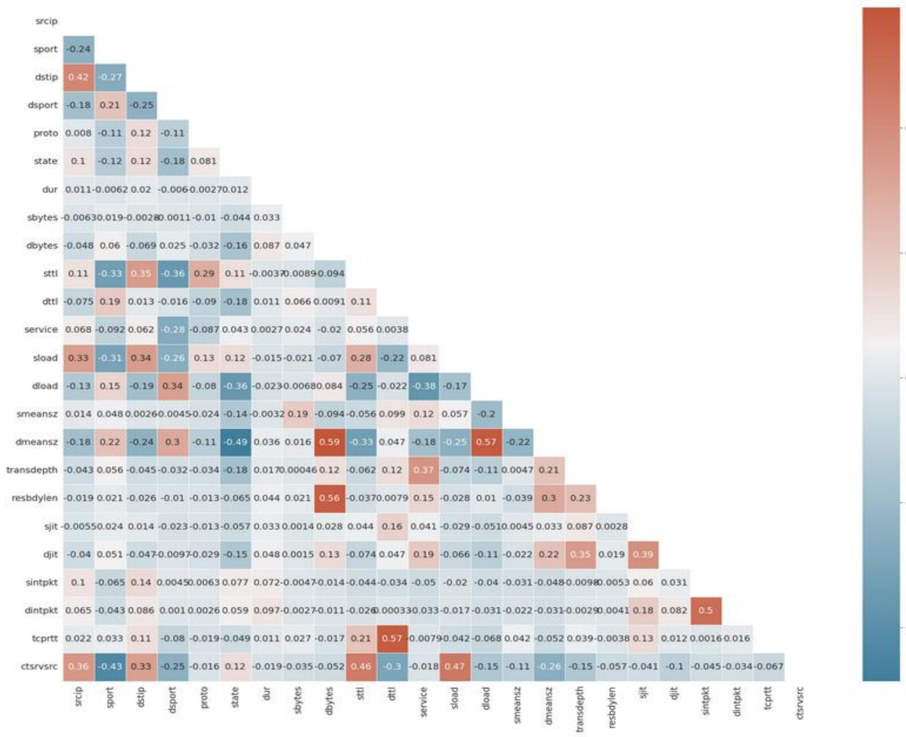


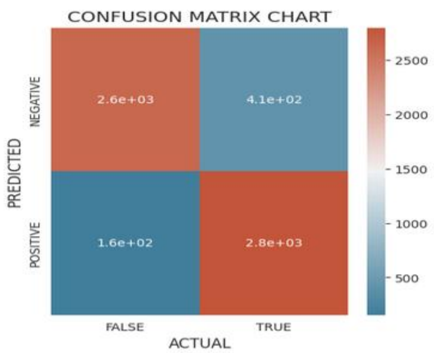
Figure 27. Correlation heatmap after removing highly correlated independent variables from UNSW NB15 dataset

```
In [70]:  dfs['CICIDS_2017'] = balance_dataset_evenly(dfs['CICIDS_2017'], dependent_variable)

In [71]:  dfs['UNSW_NB15'] = balance_dataset_evenly(dfs['UNSW_NB15'], dependent_variable)

In [72]:  dfs['CICIDS_2017'][dependent_variable] = dfs['CICIDS_2017'][dependent_variable].astype('category')
          dfs['UNSW_NB15'][dependent_variable] = dfs['UNSW_NB15'][dependent_variable].astype('category')

In [73]:  cicids_dependent_variable = dfs['CICIDS_2017'][dependent_variable]
          unsw_nb15_dependent_variable = dfs['UNSW_NB15'][dependent_variable]

          cicids_independent_variables = dfs['CICIDS_2017'].drop([dependent_variable], axis=1)
          unsw_nb15_independent_variables = dfs['UNSW_NB15'].drop([dependent_variable], axis=1)

          cicids_columns = cicids_independent_variables.columns
          unsw_nb15_columns = unsw_nb15_independent_variables.columns

In [74]:  cicids_indep_train, cicids_indep_test, cicids_dep_train, cicids_dep_test = split_dataset_into_training_and_testing_set(cicids
          unsw_nb15_indep_train, unsw_nb15_indep_test, unsw_nb15_dep_train, unsw_nb15_dep_test = split_dataset_into_training_and_testir
```

Figure 28. Code snippets for (i) Eliminate the class imbalance in the datasets, (ii) create the dependent variable data frame for both datasets, (iii) create the independent variables data frame for both datasets and (iv) Split the datasets in training and testing data for both datasets

4.6 Experiment I

```
[76]:  experiment_results[cicids][experimet_1]["LR"] = run_logistic_regression_analysis(cicids_indep_train, cicids_dep_train, cicids

======== LOGISTIC REGRESSION MODEL ANALYSIS SUMMARY FOR CICIDS 2017 DATASET (EXPERIMENT I) ========
===================================================================================================
===================================================================================================
```

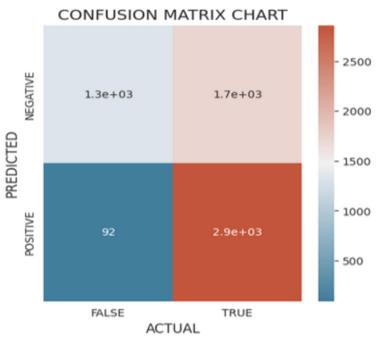



```
EVALUATION METRIC SUMMARY
=========================
ACCURACY        :    0.9053
F1 SCORE        :    0.9078
AUC SCORE       :    0.9053
```

Figure 29. Code snippet for running the Logistic Regression analysis and the summary output of the analysis for CICIDS 2017 dataset for experiment 1

```
In [77]:  experiment_results[cicids][experimet_1]["NB"] = run_naive_bayes_analysis(cicids_indep_train, cicids_dep_train, cicids_indep_test
```

```
======== NAIVE BAYES CLASSIFIER MODEL ANALYSIS SUMMARY FOR CICIDS 2017 DATASET (EXPERIMENT I) ========
======================================================================================================
======================================================================================================
```

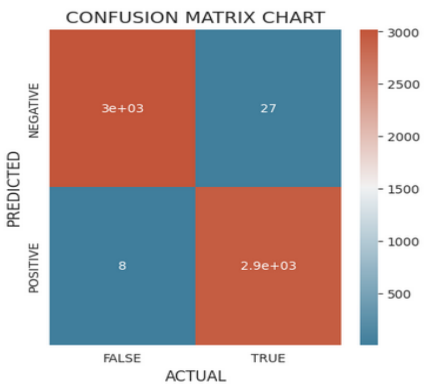



```
EVALUATION METRIC SUMMARY
=========================
ACCURACY        :   0.6972
F1 SCORE        :   0.7589
AUC SCORE       :   0.6972
```

Figure 30. Code snippet for running the Naïve Bayes analysis and the summary output of the analysis for CICIDS 2017 dataset for experiment 1

```
[78]:  experiment_results[cicids][experimet_1]["DT"] = run_decision_tree_analysis(cicids_indep_train, cicids_dep_train, cicids_
```

```
======== DECISION TREE CLASSIFIER MODEL ANALYSIS SUMMARY FOR CICIDS 2017 DATASET (EXPERIMENT I) ========
========================================================================================================
========================================================================================================
```



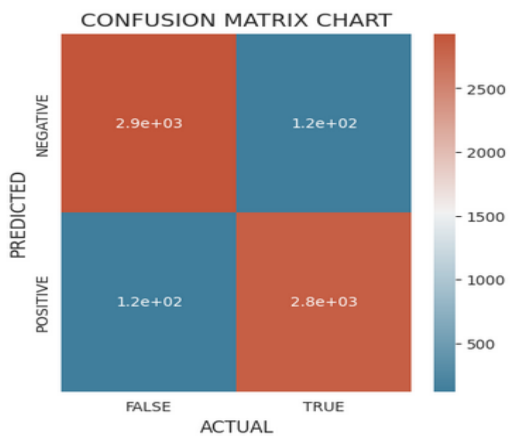

```
EVALUATION METRIC SUMMARY
=========================
ACCURACY        :   0.9942
F1 SCORE        :   0.9941
AUC SCORE       :   0.9942
```

Figure 31. Code snippet for running the Decision Tree analysis and the summary output of the analysis for CICIDS 2017 dataset for experiment 1

```
[84]:  experiment_results[cicids][experimet_1]["CNN"] = run_convolution_neural_network_analysis(cicids_indep_train, cicids_dep_train,

Epoch 1/10
750/750 [==============================] - 11s 13ms/step - loss: 0.4631 - accuracy: 0.8513
Epoch 2/10
750/750 [==============================] - 9s 12ms/step - loss: 0.3037 - accuracy: 0.9235
Epoch 3/10
750/750 [==============================] - 10s 13ms/step - loss: 0.2452 - accuracy: 0.9345
Epoch 4/10
750/750 [==============================] - 8s 10ms/step - loss: 0.2142 - accuracy: 0.9388
Epoch 5/10
750/750 [==============================] - 8s 11ms/step - loss: 0.2009 - accuracy: 0.9416
Epoch 6/10
750/750 [==============================] - 6s 9ms/step - loss: 0.1868 - accuracy: 0.9449
Epoch 7/10
750/750 [==============================] - 8s 11ms/step - loss: 0.1762 - accuracy: 0.9472
Epoch 8/10
750/750 [==============================] - 7s 9ms/step - loss: 0.1669 - accuracy: 0.9490
Epoch 9/10
750/750 [==============================] - 8s 11ms/step - loss: 0.1570 - accuracy: 0.9510
Epoch 10/10
750/750 [==============================] - 9s 12ms/step - loss: 0.1518 - accuracy: 0.9523
188/188 [==============================] - 1s 3ms/step
```

Figure 32. Code snippet for running the Convolution Neural Network (CNN) analysis for CICIDS 2017 dataset for experiment 1

```
======== CONVOLUTION NEURAL NETWORK MODEL ANALYSIS SUMMARY FOR CICIDS 2017 DATASET (EXPERIMENT I) ========
==========================================================================================================
==========================================================================================================
```



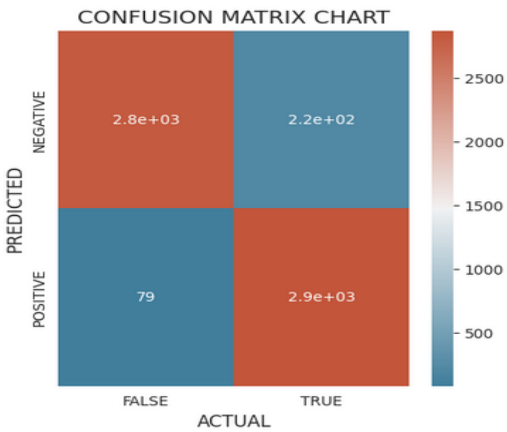

```
EVALUATION METRIC SUMMARY
=========================
ACCURACY       :    0.9598
F1 SCORE       :    0.9592
AUC SCORE      :    0.9598
```

Figure 33. Analysis summary for Convolution Neural Network (CNN) for CICIDS 2017 dataset for experiment 1

```
[85]:  warnings.filterwarnings("ignore")

       experiment_results[cicids][experimet_1]["RNN"] = run_recurrent_neural_network_analysis(cicids_indep_train, cicids_dep_train, cic

Epoch 1/10
750/750 [==============================] - 7s 6ms/step - loss: 0.4130 - accuracy: 0.8102
Epoch 2/10
750/750 [==============================] - 4s 5ms/step - loss: 0.2140 - accuracy: 0.9148
Epoch 3/10
750/750 [==============================] - 4s 6ms/step - loss: 0.1819 - accuracy: 0.9322
Epoch 4/10
750/750 [==============================] - 5s 7ms/step - loss: 0.1660 - accuracy: 0.9380
Epoch 5/10
750/750 [==============================] - 6s 8ms/step - loss: 0.1561 - accuracy: 0.9435
Epoch 6/10
750/750 [==============================] - 5s 7ms/step - loss: 0.1492 - accuracy: 0.9457
Epoch 7/10
750/750 [==============================] - 6s 8ms/step - loss: 0.1453 - accuracy: 0.9462
Epoch 8/10
750/750 [==============================] - 6s 8ms/step - loss: 0.1402 - accuracy: 0.9484
Epoch 9/10
750/750 [==============================] - 6s 8ms/step - loss: 0.1367 - accuracy: 0.9499
Epoch 10/10
750/750 [==============================] - 6s 8ms/step - loss: 0.1328 - accuracy: 0.9511
188/188 [==============================] - 1s 3ms/step
```

Figure 34. Code snippet for running the Recurrent Neural Network (RNN) analysis for CICIDS 2017 dataset for experiment 1

```
======== RECURRENT NEURAL NETWORK MODEL ANALYSIS SUMMARY FOR CICIDS 2017 DATASET (EXPERIMENT I) ========
========================================================================================================
========================================================================================================
```




```
EVALUATION METRIC SUMMARY
=========================
ACCURACY        :    0.9503
F1 SCORE        :    0.9507
AUC SCORE       :    0.9503
```

Figure 35. Analysis summary for Recurrent Neural Network (RNN) for CICIDS 2017 dataset for experiment 1

4.7 Experiment II

```
[89]:  cicids_selected_features = perform_feature_selection_using_feature_importance(cicids_independent_variables, cicids_depe
```

```
[9.50017019e-02 6.38611736e-02 3.14476986e-02 1.74582226e-02
 2.41849261e-02 2.16001586e-02 1.12769778e-01 5.18184960e-02
 6.32979619e-03 2.03046949e-02 2.60920166e-02 9.86862334e-03
 1.49895800e-02 1.65760877e-02 1.15122894e-05 2.82949794e-02
 2.11827370e-02 8.21133595e-03 3.04768346e-02 2.29363173e-06
 1.07923402e-01 9.26762872e-02 2.81679465e-02 4.91343766e-02
 5.41294749e-02 2.38516480e-02 3.54010950e-02 3.05726347e-03
 2.49848164e-03 2.67737830e-03]
```

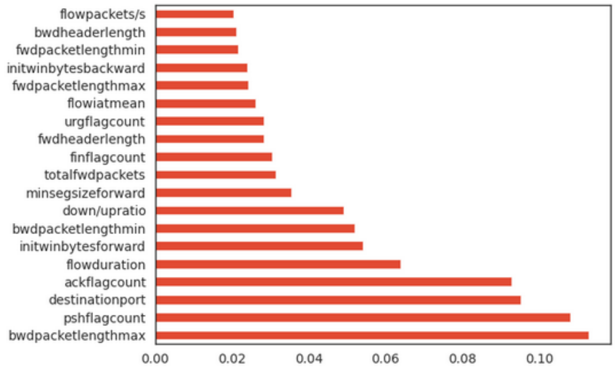


Figure 36. Code snippet for performing feature selection using feature importance for CICIDS dataset

```
[90]:  unsw_nb15_selected_features = perform_feature_selection_using_feature_importance(unsw_nb15_independent_
```

```
[0.01919897 0.06157124 0.04809679 0.0624246  0.01295275 0.02412259
 0.00324817 0.0048786  0.00224817 0.48044941 0.11519363 0.00825161
 0.00517405 0.02434133 0.01217176 0.05037917 0.00180686 0.00053175
 0.00180989 0.00242415 0.00264345 0.00225633 0.01037219 0.04345255]
```

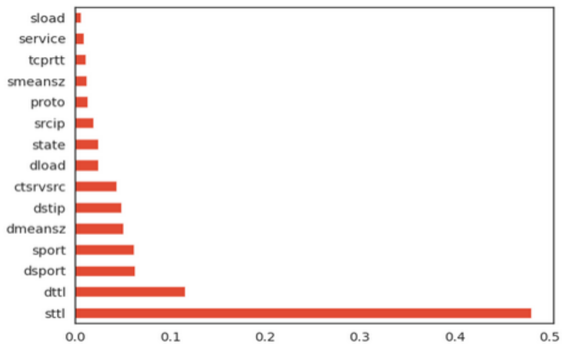
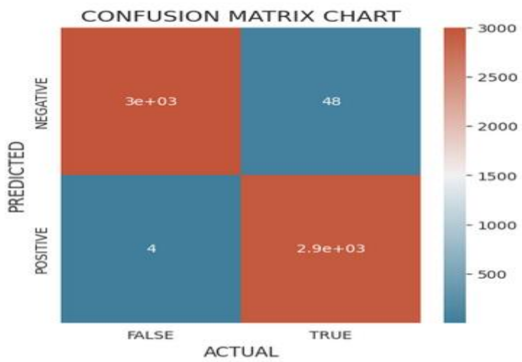


Figure 37. Code snippet for performing feature selection using feature importance for UNSW NB15 dataset

```
In [91]:  cicids_selected_features = cicids_selected_features.tolist()
          unsw_nb15_selected_features = unsw_nb15_selected_features.tolist()

In [92]:  cicids_indep_train, cicids_indep_test, cicids_dep_train, cicids_dep_test = split_dataset_into_training_and_testing_set(cicids_in
          unsw_nb15_indep_train, unsw_nb15_indep_test, unsw_nb15_dep_train, unsw_nb15_dep_test = split_dataset_into_training_and_testing_s
```

Figure 38. Code snippets for splitting selected features into training and testing dataset for both datasets

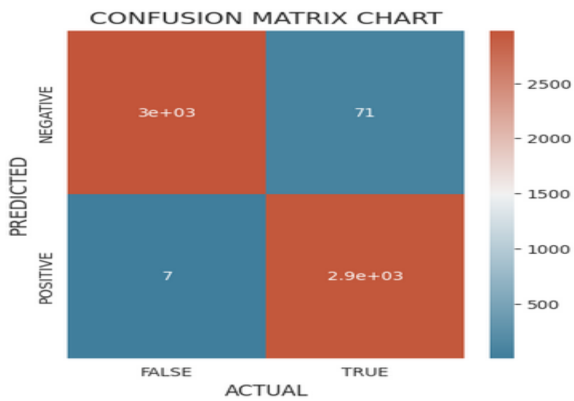
```
[98]:  experiment_results[unsw_nb15][experimet_2]["LR"] = run_hyper_parameters_tunned_logistic_regression_analys

======== LOGISTIC REGRESSION MODEL ANALYSIS SUMMARY FOR UNSW NB15 DATASET (EXPERIMENT II) ========
==================================================================================================
==================================================================================================
```




```
EVALUATION METRIC SUMMARY
=========================
ACCURACY        :    0.9913
F1 SCORE        :    0.9913
AUC SCORE       :    0.9913
```

Figure 39. Code snippet for running the hyper-parameter tuned Logistic Regression analysis and the summary output of the analysis for UNSWNB15 dataset for experiment 2

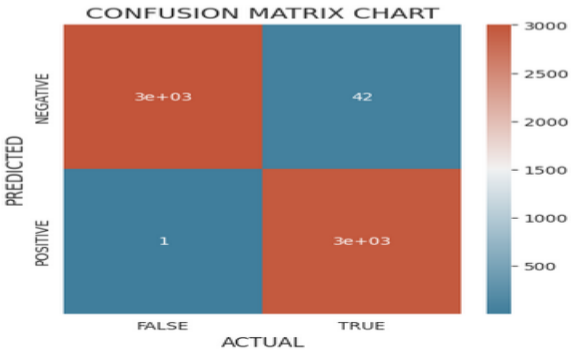
```
[99]:  experiment_results[unsw_nb15][experimet_2]["NB"] = run_hyper_parameters_tunned_naive_bayes_analysis(unsw_

======== NAIVE BAYES CLASSIFIER MODEL ANALYSIS SUMMARY FOR UNSW NB15 DATASET (EXPERIMENT II) ========
=====================================================================================================
=====================================================================================================
```




```
EVALUATION METRIC SUMMARY
=========================
ACCURACY        :    0.9870
F1 SCORE        :    0.9869
AUC SCORE       :    0.9870
```

Figure 40. Code snippet for running the hyper-parameter tuned Naïve Bayes analysis and the summary output of the analysis for UNSW NB15 dataset for experiment 2

```
[100]:  experiment_results[unsw_nb15][experimet_2]["DT"] = run_hyper_parameters_tunned_decision_tree_analysis(unsw_

======== DECISION TREE CLASSIFIER MODEL ANALYSIS SUMMARY FOR UNSW NB15 DATASET (EXPERIMENT II) ========
=====================================================================================================
=====================================================================================================
```




```
EVALUATION METRIC SUMMARY
=========================
ACCURACY       :    0.9928
F1 SCORE       :    0.9928
AUC SCORE      :    0.9928
```

Figure 41. Code snippet for running the hyper-parameter tuned Decision Tree analysis and the summary output of the analysis for UNSW NB15 dataset for experiment 2

```
In [105]:  experiment_results[unsw_nb15][experimet_2]["CNN"] = run_hyper_parameters_tunned_convolution_neural_network_analysis(unsw_nb15_in

Epoch 1/30
75/75 [==============================] - 2s 14ms/step - loss: 0.6277 - accuracy: 0.7707
Epoch 2/30
75/75 [==============================] - 1s 14ms/step - loss: 0.3011 - accuracy: 0.9476
Epoch 3/30
75/75 [==============================] - 2s 31ms/step - loss: 0.1122 - accuracy: 0.9795
Epoch 4/30
75/75 [==============================] - 1s 16ms/step - loss: 0.0774 - accuracy: 0.9869
Epoch 5/30
75/75 [==============================] - 2s 28ms/step - loss: 0.0653 - accuracy: 0.9894
Epoch 6/30
75/75 [==============================] - 1s 14ms/step - loss: 0.0604 - accuracy: 0.9898
Epoch 7/30
75/75 [==============================] - 1s 10ms/step - loss: 0.0577 - accuracy: 0.9900
Epoch 8/30
75/75 [==============================] - 1s 10ms/step - loss: 0.0555 - accuracy: 0.9901
Epoch 9/30
75/75 [==============================] - 1s 13ms/step - loss: 0.0515 - accuracy: 0.9904
Epoch 10/30
```

```
In [106]:  experiment_results[unsw_nb15][experimet_2]["RNN"] = run_hyper_parameters_tunned_recurrent_neural_network_analysis(unsw_nb15_inde

Epoch 1/30
75/75 [==============================] - 3s 9ms/step - loss: 0.6923 - accuracy: 0.5846
Epoch 2/30
75/75 [==============================] - 1s 8ms/step - loss: 0.6435 - accuracy: 0.8807
Epoch 3/30
75/75 [==============================] - 1s 9ms/step - loss: 0.3492 - accuracy: 0.9585
Epoch 4/30
75/75 [==============================] - 1s 9ms/step - loss: 0.1049 - accuracy: 0.9865
Epoch 5/30
75/75 [==============================] - 1s 9ms/step - loss: 0.0596 - accuracy: 0.9901
Epoch 6/30
75/75 [==============================] - 1s 9ms/step - loss: 0.0495 - accuracy: 0.9907
Epoch 7/30
75/75 [==============================] - 1s 10ms/step - loss: 0.0451 - accuracy: 0.9909
Epoch 8/30
75/75 [==============================] - 1s 10ms/step - loss: 0.0425 - accuracy: 0.9911
Epoch 9/30
75/75 [==============================] - 1s 7ms/step - loss: 0.0409 - accuracy: 0.9910
Epoch 10/30
```

Figure 42. Code snippets for (i) Hyper-parameter tuned Convolution Neural Network (CNN) analysis for UNSW NB15 dataset for experiment 2 and (ii) Hyper-parameter tuned Recurrent Neural Network (RNN) analysis for UNSW NB15 dataset for experiment 2

```
[107]:  data_result_list = [cicids, unsw_nb15]
        experi_result_list = [experimet_1, experimet_2]

        for dataset in data_result_list:
            for experiment in experi_result_list:

                implemented_models = list(experiment_results[dataset][experiment].keys())
                if len(implemented_models) > 0:

                    summary_table_header = "{} implementation summary table for {} dataset".format(" ".join(experiment.split("_")), " ".
                    print("\n\n")
                    print(summary_table_header)
                    print(make_horizontal_line(len(summary_table_header)))
                    print(make_horizontal_line(len(summary_table_header)))
                    print("\n")

                    evaluation_metrics = list(experiment_results[dataset][experiment][implemented_models[0]].keys())
                    show_implemented_models_evaluation_tables(experiment_results[dataset][experiment], implemented_models, evaluation_me
                    print("\n\n\n\n")
```

Figure 43. Code snippet displaying summary tables for the implemented models, experiments and datasets

EXPERIMENT I IMPLEMENTATION SUMMARY TABLE FOR CICIDS 2017 DATASET

	ACCURACY	F1 SCORE	AUC SCORE
LR	0.9053	0.9078	0.9053
NB	0.6972	0.7589	0.6972
DT	0.9942	0.9941	0.9942
CNN	0.9598	0.9592	0.9598
RNN	0.9503	0.9507	0.9503

EXPERIMENT II IMPLEMENTATION SUMMARY TABLE FOR CICIDS 2017 DATASET

	ACCURACY	F1 SCORE	AUC SCORE
LR	0.9220	0.9236	0.9220
NB	0.8183	0.8354	0.8183
DT	0.9940	0.9939	0.9940
CNN	0.9577	0.9579	0.9577
RNN	0.9565	0.9566	0.9565

Figure 44. Implemented models summary tables for experiment 1 and experiment 2 for CICIDS 2017 dataset

EXPERIMENT I IMPLEMENTATION SUMMARY TABLE FOR UNSW NB15 DATASET

	ACCURACY	F1 SCORE	AUC SCORE
LR	0.9907	0.9906	0.9907
NB	0.9778	0.9779	0.9778
DT	0.9927	0.9926	0.9927
CNN	0.9920	0.9919	0.9920
RNN	0.9922	0.9921	0.9922

EXPERIMENT II IMPLEMENTATION SUMMARY TABLE FOR UNSW NB15 DATASET

	ACCURACY	F1 SCORE	AUC SCORE
LR	0.9913	0.9913	0.9913
NB	0.9870	0.9869	0.9870
DT	0.9928	0.9928	0.9928
CNN	0.9918	0.9918	0.9918
RNN	0.9918	0.9918	0.9918

Figure 45. Implemented models summary tables for experiment 1 and experiment 2 for UNSW NB15 dataset

5 Conclusion

This documentation outlines the procedures for executing the code implementation in our research project. The implementation involves the analysis of the CICIDS 2017 dataset and UNSW NB15 dataset, employing three supervised learning algorithms (Logistic Regression, Naïve Bayes, and Decision Tree Classifier) and two deep learning algorithms (Convolutional Neural Network - CNN and Recurrent Neural Network - RNN). The provided code snippets, generated figures, and presented tables play a pivotal role in realizing the research objectives.

Researchers intending to replicate this study, utilizing the same datasets and algorithms, can anticipate obtaining comparable results. The documented instructions serve as a comprehensive guide for achieving consistent outcomes, ensuring reproducibility in subsequent analyses.

References

IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB. (n.d.). https://www.unb.ca/cic/datasets/ids-2017.html

Installing on macOS — Anaconda documentation. (n.d.). anaconda.com. https://docs.anaconda.com/free/anaconda/install/mac-os/

The UNSW-NB15 Dataset | UNSW Research. (n.d.). https://research.unsw.edu.au/projects/unsw-nb15-dataset

Wikipedia contributors. (2023a, August 25). *Anaconda (Python distribution)*. Wikipedia. https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

Wikipedia contributors. (2023b, November 5). *Project Jupyter*. Wikipedia. https://en.wikipedia.org/wiki/Project_Jupyter#Jupyter_Notebook