Configuration Manual for Distributed Intrusion Detection System for Cloud Environments Using Deep Learning Machine Algorithms

MSc Research Project
Masters in Cloud Computing

Oyindeinbofa Pibowei
Student ID: x20165765

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Oyindeinbofa Pibowei		
Student ID:	X20165765		
Programme:	Master of Science in Computing	**Year:**	2022
Module:	……Research Project…….……		
Lecturer:	……Prof. Vikas Sahni……		
Submission Due Date:	……26th APRIL 2022…….……		
Project Title:	Configuration Manual for Distributed Intrusion Detection System for Cloud Environments Using Deep Learning Machine Algorithms		
Word Count:	5305	**Page Count:**	31

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: ……Oyindeinbofa Pibowei……

Date: ……26th APRIL 2022……

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 for Distributed Intrusion Detection System for Cloud Environments Using Deep Learning Machine Algorithms

Oyindeinbofa Pibowei
X20165765

Link to Video: Oyindeinbofa_Pibowei_x20165765_Research_Project_Presentation-20220421_105510-Meeting Recording.mp4

1 Introduction

This report is aim at producing a manual which is a comprehensive guideline for creating the code implementation setup of the research on "Distributed Intrusion Detection System for Cloud Environments Using Deep Learning Machine Algorithms". The research was carried out to investigate how well deep learning and machine learning algorithm can classify a network traffic as attack traffic or a non-attack traffic using historical dataset. In carrying out this experiment, three historic (3) datasets were used in this code implementation and the datasets includes KDD99 dataset, CIC IDS dataset and NSL KDD dataset. Three machine learning algorithms (Logistic Regression, Random Forest Classifier and Gradient Boost Classifier) and a deep learning algorithm (Artificial Neural Network) were used to analyse the datasets in this research experiment.

The rest of this report is presented as follows: Section 2 discusses system specifications for hardware and software components used in this code implementation, Section 3 covers software installation, Anaconda environment setup and the installation of Python libraries used in this code implementation. Section 4 will discuss code implementation and evaluation of 4 model implemented and how well the models performed with the analysis of each dataset, Section 5 discusses the concluding or final statements and Section 6 contains a list of references.

2 System Specification

2.1 System Hardware Requirement

The PC used in implementing the code for this work has the following configuration settings

RAM: 16GB
Processor Type: Intel core i7 with 1.99GHz processing speed
Storage Capacity: 512GB SSD.

2.2 System Software Requirement

2.2.1 Operating System (Microsoft windows 10)

Windows 10 OS is the underlying platform on which the other software used for the project implementation will be installed.

2.2.2 Anaconda Navigator V2.1.1

Anaconda Navigator is package manager for Python programming language and it is used manage Python versions and manage different work space on a single machine.

2.2.3 Jupyter Notebook V.6.4.5

Jupyter Notebook is the programming IDE that interpretes code segment interactively on the browser.

2.2.4 Web Browser (Mozilla Firefox Browser V.98.0.2)

Mozilla Firefox is used by Jupyter Notebook display the code snippet and the output of the code snippet.

3 Software Installation, Environment Setup and Python Libraries

3.1 Software Installation for Anaconda

STEP 1:
Go to Anaconda website and download Anaconda installer file

STEP 2:
Locate the downloaded installer file and double click it to launch the installer. On the welcome to Anaconda page, click on “Next” button.

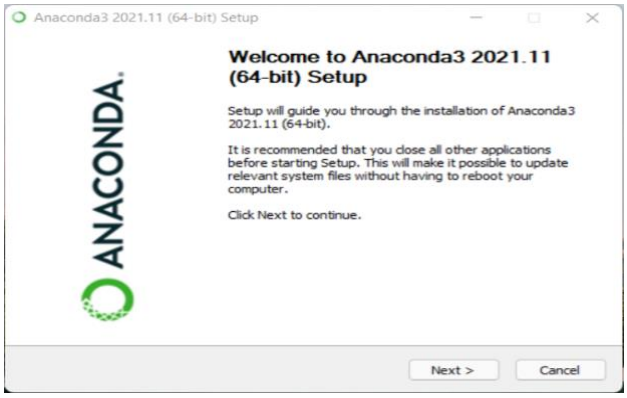

Figure 1. Anaconda installer welcome page

STEP 3:
On the license agreement, read the licensing terms and click on the button labelled “I Agree” to continue

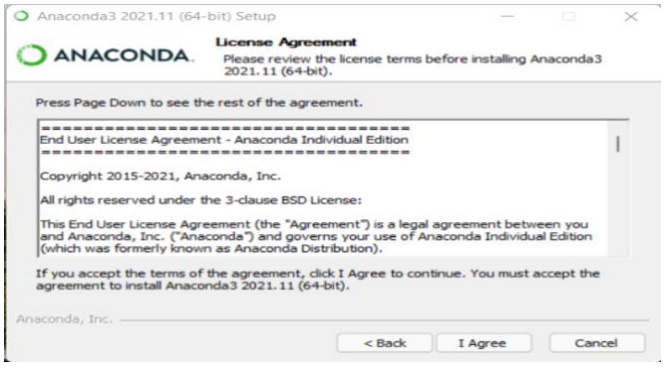

Figure 2. Anaconda installer license agreement page

STEP 4:
Click on the button labelled “Next” on the Select installation type page

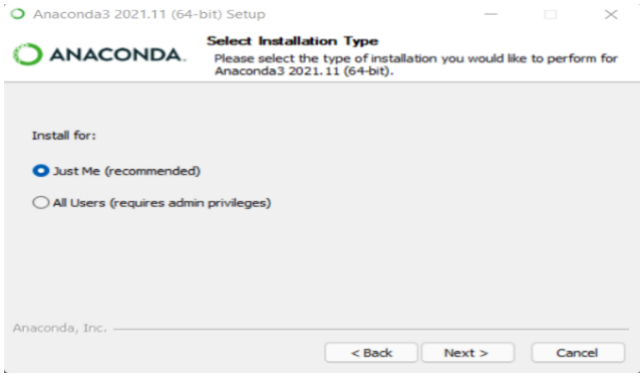

Figure 3. Anaconda installer select installation type

STEP 5:
Click on the button labelled “Next” on the Choose install location

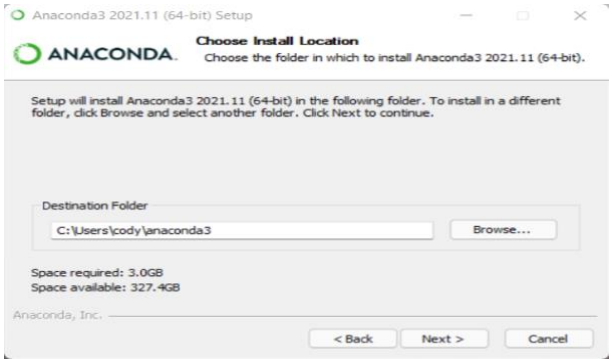

Figure 4. Anaconda installer choose install location page

STEP 6:
Click on the button labelled “Install” on the Advanced installation options

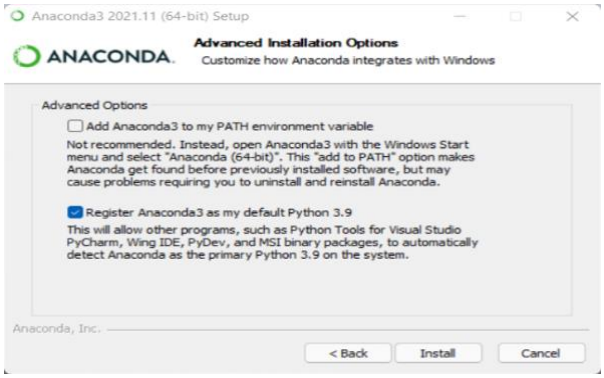

Figure 5. Anaconda installer advance installation options page

STEP 7:

On the Installation Complete page, allow the installation to run to completion and click on the button labelled “Next”

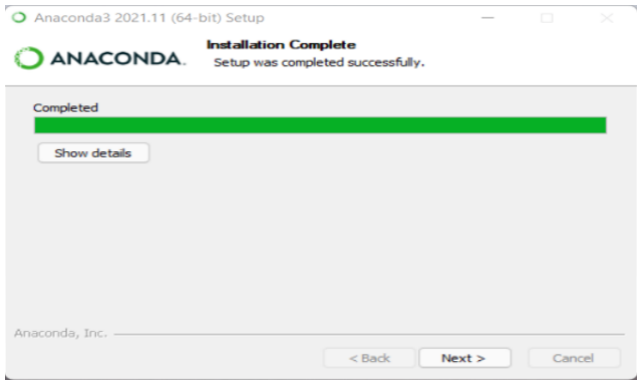

Figure 6. Anaconda installer installation complete page

STEP 8:
Click on the button labelled “Next” on the Install PyCharm

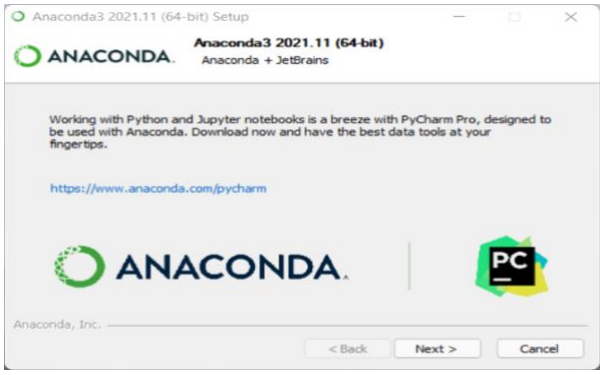

Figure 7. Anaconda installer pycharm install option page

STEP 9:
Click on the button labelled “Finish” on the Thank you page

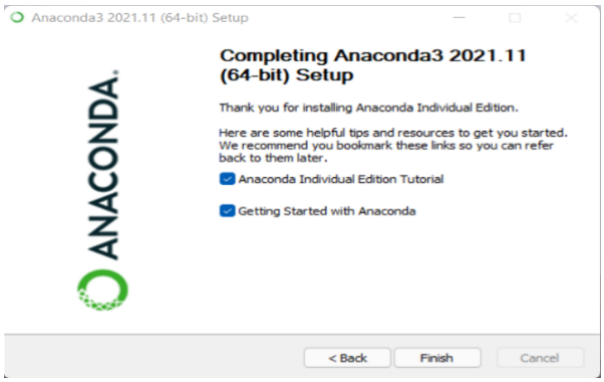

Figure 8. Anaconda installer completing anaconda3 setup

3.2 Environment Setup and Python Libraries

The Anaconda installation created an environment called "base" pre-installing some of Python's libraries needed to carry out this coding implementation of the project work. Some other libraries will be installed to completely setup the environment for carrying out this coding exercise

STEP 1:
Click on the Windows icon, expand Anaconda folder and click on Anaconda Prompt to start Anaconda command line.

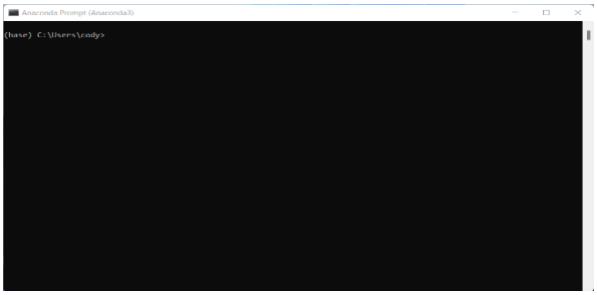

Figure 9. launch anaconda prompt

STEP 2:
Create a new conda environment for carrying out this code implementation using the following command "`conda create -n venv python=3.6`". Following the instruction on command prompt to complete the installation. After a successful creation of the venv environment, enter the following command "`conda activate venv`" to activate the newly created environment

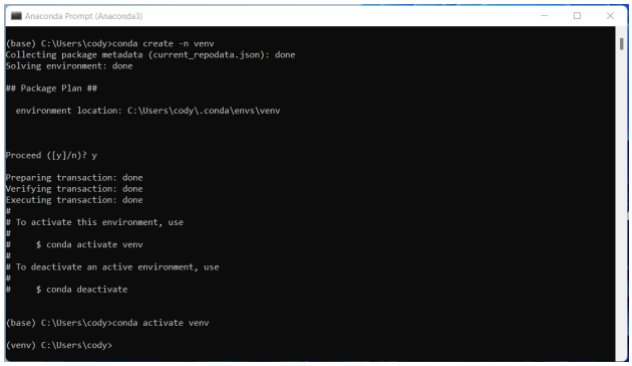

Figure 10. Anaconda prompt showing the create environment command and conda command for activating the created environment

STEP 3:
After activating the venv environment, use the following commands to install all Python's libraries used in this code implementation

- "`conda install -c conda-forge notebook`"
- "`conda install -c conda-forge jupyterlab`"
- "`conda install -c conda-forge nb_conda_kernels`"
- "`conda install -c anaconda pandas`"
- "`conda install -c conda-forge seaborn`"
- "`conda install -c intel scikit-learn`"

- “conda install scikit-learn-intelex”
- “conda install -c conda-forge imbalanced-learn”
- “conda install -c conda-forge keras”

4 Implementation and Evaluation

After a successful installation Anaconda and installing all libraries required to carry out this code implementation, this section shows the actual code implementation for this research work as shown below:

4.1 Start a new project

On the launch Anaconda main page, click on the button labelled “Launch” on the cell with the Jupyter Notebook load the interactive Python IDE on the browser

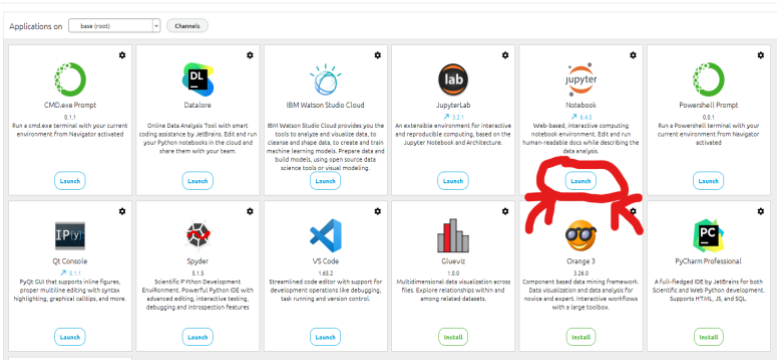

Figure 11. Launched anaconda navigator home page

On the Jupyter Notebook start up page, locate the new button on top right corner and click the button and click also on the “Python 3” to start up a new project

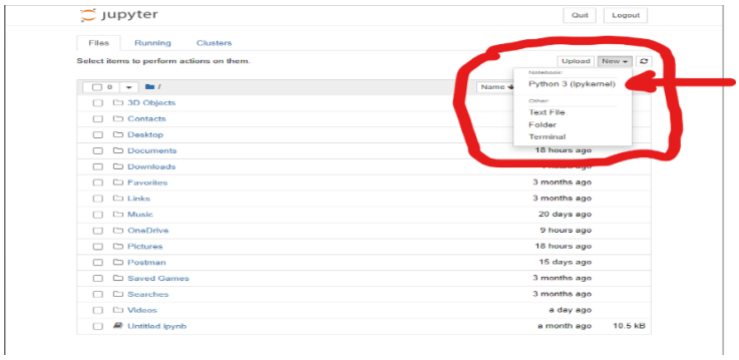

Figure 12. Jupyter notebook home page showing how to start up a new project

4.2 Import libraries

After successfully launching a new project, on the first code snippet all Python libraries used to carry out this project will be added to this code snippet. Every time this block is updated, the button labelled “Run” should be clicked to import the library into the IDE

```
In [1]:
import os
import time
import itertools

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, r
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from imblearn.over_sampling import SMOTE

from keras.models import Sequential
from keras.layers import *
from keras.wrappers.scikit_learn import KerasClassifier
from keras.metrics import *

from IPython.core.interactiveshell import InteractiveShell
```

Figure 13. The code snippet showing all Python libraries for carrying out this code implement

4.3 Defined global variables used to track data generated while executing codes

Also, the second cell was dedicated to declaring global variables that will be used to store and track output generated by executing codes in different blocks as they are being executed. Figure 14 below shows all global variables defined while implementing the code base for this research work.

```
In [2]:
InteractiveShell.ast_node_interactivity = "all"
pd.options.display.max_seq_items = 2000
pd.options.display.max_rows = 4000

df_column_names = dict()
features_name = dict()
analysis_dataset = dict()
analysis_result = dict()

ids_nslkdd_filename_1 = 'KDDTrain+.txt'
ids_nslkdd_filename_2 = 'KDDTest+.txt'
ids_nslkdd_filename_3 = 'KDDTrain+_20Percent.txt'
ids_kdd99_filename = 'ids_kddcup99.csv'
ids_cic_filename = 'ids_cic.csv'

nslkdd_colname = 'nslkdd'
kdd99_colname = 'kdd99'
cic_colname = 'cic'

x_train_label = "x_train"
x_test_label = "x_test"
y_train_label = "y_train"
y_test_label = "y_test"

analysis_result[kdd99_colname] =  dict()
analysis_result[nslkdd_colname] =  dict()
analysis_result[cic_colname] =  dict()

analysis_dataset[kdd99_colname] =  dict()
analysis_dataset[nslkdd_colname] =  dict()
analysis_dataset[cic_colname] =  dict()

dependent_variable_col_name = 'attack'

ids_category_label = ['Non Attack', 'Attack']
```

Figure 14. The code snippet showing all global variable used to store data generate while executing code snippets

4.4 Defined functions

In carrying out this research code implementation, 3 different datasets will be used in the analysis, meaning that tasks repetition will be a common occurrence and code reuse will be a way to minimize the amount of code written. Functions were defined to implement code reuse and minimize the number of codes written while carrying out this project code implementation.

Figure 15 below shows the function in the code implementation of the project that is used to retrieve column names for a given dataset. The function is used to remove space character, underscore character, hyphen and forward slash character that is present in the column name of the dataset.

```
In [3]:  def generate_column_names_for_dataset(df, key, df_column_names):
             col_names =  [col.lower().replace(' ', '').replace('_', '').replace('-',
             df_column_names[key] = col_names
             df.reset_index(drop=True, inplace = True)
             return col_names
```

Figure 15. The function to generate the list of column names from the dataset

Figure 16 below shows the function in the code implementation of the project that is used to remove all missing values in a given dataset. The function uses numpy's properties to remove rows with missing values and return all rows that have been determined to have valid data.

```
In [4]:  def operattion_to_remove_missing_or_nan_from_dataset(df):
             df.dropna(inplace=True)
             df = df.replace([np.inf, -np.inf], np.nan)
             df.dropna(inplace=True)
             return df
```

Figure 16. The function to remove missing value from a dataset

Figure 17 below shows the function in the code implementation of the project that is used to rename columns in a dataset. The function accepts 3 arguments namely the dataset with column names to be changed, a list of column names to be changed and a list of the names to update the column names.

```
In [5]:  def rename_columns_in_dataset(df, list_old_col_names, list_new_col_names):
             col_names_replacement_dict = dict()

             if not (len(list_old_col_names) == len(list_new_col_names)):
                 print("\n Column names was not change due to differences in the lengt
                 pass
             else:
                 for i in range(len(list_old_col_names)):
                     col_names_replacement_dict[str(list_old_col_names[i]).lower()] =

                 df.rename(columns=col_names_replacement_dict, inplace=True)

             return df
```

Figure 17. The function to rename column name in a dataset

Figure 18 below shows the function in the code implementation of the project that is used to plot the pie chart for the 3 datasets showing attack traffics to non-attack traffics. The function received five arguments and these arguments includes a tuple data structure containing the 3 datasets, a tuple containing the titles for the 3 plots, a list containing the class label for categorizing a traffic as an attack or a non-attack, a main title and the dependent variable column name.

```
In [6]:  def plot_pie_chart_for_dependent_variable(df_tuple, title_tuple, ids_category

             colors = {1: 'r', 0: 'gray'}
             print(colors)
             fig, axes = plt.subplots(1, 3, figsize=(20,10), dpi=144, tight_layout=Fa
             plt.suptitle(main_title)

             for ax, df, title in zip(axes, df_tuple, title_tuple):
                 count = df[dependent_variable_col_name].value_counts().to_frame().sor
                 print(count.index)
                 ax.pie(count[dependent_variable_col_name], labels=ids_category_label,
                 ax.set_title(title)
```

Figure 18. The function to plot a pie chart for given set of datasets

Figure 19 below shows the function in the code implementation of the project that is used to show the correlation chart for all the features excluding the dependent column for the 3 datasets. The function receives 1 argument, a tuple containing the independent variables in each dataset for all dataset.

```
In [7]:  def plot_data_correlation(df_tuple):

             fig, axes = plt.subplots(3, 1,tight_layout=False)

             for ax, df in zip(axes, df_tuple):
                 df = df[[col for col in df if df[col].nunique() > 1]]
                 corr = df.corr()
                 plt.figure(figsize=(15,12))
                 #sns.heatmap(corr, annot=True, annot_kws={"size": 14})
                 sns.heatmap(corr)
                 sns.set_style("white")

             #plt.xticks(fontsize=14)
             #plt.yticks(fontsize=14)
             plt.show()
```

Figure 19. The function to plot the correlation heat map for a given set of datasets

Figure 20 below shows the function in the code implementation of the project that is used to retrieve the column names of highly correlated features (independent variables) in the dataset. The function receives two arguments and they include the dataset whose independent variable correlation values will be tested and threshold used to determine the highly correlated features.

```
In [9]:  def estimate_correlation_without_negative_correlation(df, threshold):
             col_corr = set()
             corr_matrix = df.corr()

             for i in range(len(corr_matrix.columns)):
                 for j in range(i):
                     if (corr_matrix.iloc[i,j]) > threshold:
                         col_name = corr_matrix.columns[i]
                         col_corr.add(col_name)

             return col_corr
```

Figure 20. The function used to estimate the correlation between features in a given dataset

Figure 21 below shows the function in the code implementation of the project that is used to retrieve categorical variable in the dataset. The function receives the dataset as an argument and retrieves a list of the column names of all categorical variables.

```
In [10]:  def get_categorical_attributes(df):
              categorical_attributes = list()

              numeric_cols = df.select_dtypes("number").columns
              categorical_cols = df.select_dtypes("object").columns

              numeric_cols= list(set(numeric_cols))
              categorical_attributes= list(set(categorical_cols))

              for col_name in numeric_cols:
                  unique_values = df[col_name].nunique()
                  if unique_values < 10:
                      categorical_attributes.append(col_name)
              return categorical_attributes
```

Figure 21. The function used to retrieve column names for all categorical variables

Figure 22 below shows the function in the code implementation of the project that is used to split a dataset into a train set and a test set. The function receives two arguments the independent variables (features) and dependent variable

```
In [12]:  def split_dataset_into_train_test_set(X, y):
              return train_test_split(X, y, test_size = 0.3, random_state = 0)
```

Figure 22. The function used to split dependent and independent variables into training and testing dataset

Figure 23 below shows the function in the code implementation of the project that is used to performs Logistic Regression analysis. The function accepts two arguments which is a dictionary data structure containing all datasets and another dictionary to track analysis result. The function also invokes another function ***perform_model_analysis*** which helps to train the model and use the test set evaluate the performance of the model.

```
In [13]:  def perform_logistic_regression_modelling_and_analysis(dataset_dict, analysis
              model_label = "LR"

              for key in dataset_dict:
                  lr = None
                  lr = LogisticRegression(max_iter=1200000)
                  result = perform_model_analysis(lr, dataset_dict[key], key, model_lab
                  analysis_result[key][model_label] = result
                  print()
                  print()
```

Figure 23. The function used to implement Logistic Regression (LR) classification analysis

Figure 24 below shows the function in the code implementation of the project that is used to perform Random Forest classification analysis. The function accepts two dictionary data structure as argument with one argument containing all datasets and another argument to track analysis result. The function also invokes another function ***perform_model_analysis*** which helps to train the model and use the test set evaluate the performance of the model.

```
In [14]:  def perform_random_forest_classifier_modelling_and_analysis(dataset_dict, ana
              model_label = "RFC"

              for key in dataset_dict:
                  rfc = None
                  rfc = RandomForestClassifier(n_estimators=30)
                  result = perform_model_analysis(rfc, dataset_dict[key], key, model_la
                  analysis_result[key][model_label] = result
                  print()
                  print()
```

Figure 24. The function used to implement Random Forest (RFC) classification analysis

Figure 25 below shows the function in the code implementation of the project that is used to perform Gradient Boost classification analysis. The function accepts two dictionary data structure as argument with one argument containing all datasets and another argument to track analysis result. The function also invokes another function ***perform_model_analysis*** which helps to train the model and use the test set evaluate the performance of the model.

```
In [15]:  def perform_gradient_boost_classifier_modelling_and_analysis(dataset_dict, ar
              model_label = "GBC"

              for key in dataset_dict:
                  gbc = None
                  gbc = GradientBoostingClassifier(random_state=0)
                  result = perform_model_analysis(gbc, dataset_dict[key], key, model_la
                  analysis_result[key][model_label] = result
                  print()
                  print()
```

Figure 25. The function used to implement Gradient Boost (GBC) classification analysis

Figure 26 below shows the function in the code implementation of the project that is used to perform model analysis by the three machine learning algorithms implemented in this project.

The function accepts four argument and these includes the model to be analysed, the dataset dictionary, the key to accessing the dataset in the dictionary and the label used to mark the model. The function invokes two other functions name ***train_model*** to train the target model and ***run_analysis*** to evaluate model perform after training the model. The function displays the summary information showing all performance metric for visualising how well the model performed.

```
In [16]:
def perform_model_analysis(model, target_dataset_dict, dataset_key, model_key):
    train_model(model, target_dataset_dict['x_train'], target_dataset_dict['y_train'], dataset_key,
    pred = run_analysis(model, target_dataset_dict['x_test'], dataset_key, model_key)

    print("\nAnalysis Summary for " + model_key + " model on " + dataset_key + " dataset")
    print("==============================================================================\n=======

    print("\nClassication Report\n")
    print(classification_report(pred, target_dataset_dict['y_test'], target_names=["Normal", "Attac

    print("\nConfusion Matrix\n")
    conf_mat = confusion_matrix(target_dataset_dict['y_test'], pred)
    plt.figure(figsize=(8,6))
    ax_plot = sns.heatmap(conf_mat, annot=True)
    ax_plot.set_title("Seaborn Confusion Matrix Plot")
    ax_plot.set_xlabel('Predicted Values')
    ax_plot.set_ylabel('Actual Values')
    ax_plot.xaxis.set_ticklabels(["False", "True"])
    ax_plot.yaxis.set_ticklabels(["Negative", "Positive"])
    plt.show()


    print("\nModel Result on different metrics\n")
    acc_score = accuracy_score(target_dataset_dict['y_test'], pred)
    pre_score = precision_score(target_dataset_dict['y_test'], pred)
    rec_score = recall_score(target_dataset_dict['y_test'], pred)
    auc_score = roc_auc_score(target_dataset_dict['y_test'], pred)
    print(f"""
    Accuracy: {acc_score:.2f} %
    Precision: {pre_score:.2f} %
    Recall: {rec_score:.2f} %
    AUC Score: {auc_score:.2f} %
    """)

    print("\n\n")

    result = {
        "Accuracy": acc_score,
        "Precision": pre_score,
        "Recall": rec_score,
        "AUC": auc_score
    }

    return result
```

Figure 26. The function used by the implemented models (LR, RFC, GBC) to perform training and testing for given datasets and display results

Figure 17 below shows the function in the code implementation of the project that is used to train model being analysed. The function accepts four arguments and these includes ***model*** which is the instance of the model being analysed, ***x_train*** is the independent variables of the training dataset, ***y_train*** is the dependent variable of the training set and ***model_key*** is the labelled name for the model used in analysis. The function returns a trained model instance after a successful training.

```
In [17]:
def train_model(model, x_train, y_train, dataset_key, model_key):
    start_time = time.time()
    model.fit(x_train, y_train.values.ravel())
    end_time = time.time()
    print("Training time for " + model_key + " model on " + dataset_key + " dataset: ", end_time-st
    return model
```

Figure 27. The function used for training model using the training set of a target dataset

Figure 28 below shows the function in the code implementation of the project that is used to evaluate the performance of a trained machine learning model. The function accepts four arguments and these includes ***model*** which has already been trained, ***x_test*** is the independent variables of the used in prediction or evaluate the trained model, ***dataset_key*** is used to identify the dataset being analysed and ***model_key*** is the labelled name for the model used in analysis. The function returns a prediction generated by the model.

```
In [18]:  def run_analysis(model, x_test, dataset_key, model_key):
              start_time = time.time()
              predict_result = model.predict(x_test)
              end_time = time.time()
              print("Testing time for " + model_key + " model on " + dataset_key + " dataset: ", end_time-sta
              return predict_result
```

Figure 28. The function used for testing model and generating predictions for a testing set of a target dataset

Figure 29 below shows the function in the code implementation of the project that is used to build an artificial neural network (ANN) model. The function uses keras Sequential module and keras layer Dense module to build the neural network using ***input_dim*** argument to update the input_dim for the implemented model. The function also returns the model instance created.

```
In [19]:  def build_ANN_model(input_dim):
              model = Sequential()
              model.add(Dense(input_dim, input_dim=input_dim, activation = 'relu',kernel_initializer='random_
              model.add(Dense(1,activation='sigmoid',kernel_initializer='random_uniform'))
              model.add(Dense(2,activation='softmax'))
              model.compile(
                  loss ='categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = [
                      'accuracy'
                  ]
              )
              model.summary()
              return model
```

Figure 29. The function used to build ANN model

Figure 30 below shows the function in the code implementation of the project that is used to instantiate ANN model, run model analysis and display the result of the generated analysis for different dataset being analysed. The function accepts two dictionary data structure as argument with one argument containing all datasets and another argument to track analysis result. The function also uses a for loop to create new ANN model when analysing a dataset, train the model by invoking ***run_ANN_analysis()*** function and generates a prediction by using the model to evaluate the test set. After a successful evaluation, the model will then present and display model performance using different metrics.

```
In [20]:  def run_ANN_model(dataset_dict, analysis_result):
              model_label = "ANN"

              for key in dataset_dict:
                  input_dim = dataset_dict[key]['x_train'].shape[1]
                  ann = build_ANN_model(input_dim)
                  ann = KerasClassifier(lambda: build_ANN_model(input_dim), epochs=100, batch_size=100)
                  ann = run_ANN_analysis(ann, dataset_dict[key], key, model_label)

                  ann_pred = ann.predict(dataset_dict[key]['x_test'])
                  print("\nAnalysis Summary for " + model_label + " model on " + key + " dataset")
                  print("==============================================================================\n===

                  print("\nClassication Report\n")
                  print(classification_report(ann_pred, dataset_dict[key]['y_test'], target_names=["Normal",

                  print("\nConfusion Matrix\n")
                  conf_mat = confusion_matrix(dataset_dict[key]['y_test'], ann_pred)
                  plt.figure(figsize=(8,6))
                  ax_plot = sns.heatmap(conf_mat, annot=True)
                  ax_plot.set_title("Seaborn Confusion Matrix Plot")
                  ax_plot.set_xlabel('Predicted Values')
                  ax_plot.set_ylabel('Actual Values')
                  ax_plot.xaxis.set_ticklabels(["False", "True"])
                  ax_plot.yaxis.set_ticklabels(["Negative", "Positive"])
                  plt.show()


                  print("\nModel Result on different metrics\n")
                  acc_score = accuracy_score(dataset_dict[key]['y_test'], ann_pred)
                  pre_score = precision_score(dataset_dict[key]['y_test'], ann_pred)
                  rec_score = recall_score(dataset_dict[key]['y_test'], ann_pred)
                  auc_score = roc_auc_score(dataset_dict[key]['y_test'], ann_pred)
                  print(f"""
                  Accuracy: {acc_score:.2f} %
                  Precision: {pre_score:.2f} %
                  Recall: {rec_score:.2f} %
                  AUC Score: {auc_score:.2f} %
                  """)

                  print("\n\n")

                  result = {
                      "Accuracy": acc_score,
                      "Precision": pre_score,
                      "Recall": rec_score,
                      "AUC": auc_score
                  }

                  analysis_result[key][model_label] = result

                  print()
                  print()
```

Figure 30. The function used to implement ANN analysis by training and testing the model for a given datasets and display results

Figure 31 below shows the function in the code implementation of the project that is used to train ANN model. The function accepts four arguments and these includes ***model*** which is the instance of the ANN model being analysed, ***target_dataset_dict*** contains the training dataset dependent and independent variables used to train the model, ***dataset_key*** is used to identify the dataset being analysed and ***model_key*** is the labelled name for the model used in analysis. The function returns a trained model instance after a successful training.

```
In [21]: def run_ANN_analysis(model, target_dataset_dict, dataset_key, model_key):
             start = time.time()
             model.fit(target_dataset_dict['x_train'], target_dataset_dict['y_train'].values.ravel())
             end = time.time()
             return model
```

Figure 31. The function used for training ANN model using the training set of a target dataset

Figure 32 below shows the function in the code implementation of the project that is used to set the content of a table data. The function accepts four arguments which includes ***text*** the data to be displayed on the table cell, ***pos*** the table cell position were the table data will occupy, ***item_count*** the cell count for table for displaying summary and ***count_size*** is cell maximum size.

```
In [22]: def set_table_content(text, pos, item_count, content_size):
             table_content = "| "
             try:
                 padding = content_size - len(text)
                 table_content += text
             except:
                 padding = content_size
                 table_content += ""

             for num in range((padding-3)):
                 table_content += " "

             if pos == 1:
                 table_content += " "
             elif pos == item_count:
                 table_content += "|\n"
             else:
                 table_content += " "

             return table_content
```

Figure 32. The function used to set table content while draw summary for output of an analysis

Figure 33 below shows the function in the code implementation of the project that is used to draw summary table for a target dataset on all machine learning and neural network models implemented against all evaluation metrics. The function accepted four arguments which included ***dataset*** a unique id for identifying a dataset, ***analysis_result*** used to hold all output from analysing models, ***model_list*** holds a list of all models analysed and ***metric_list*** holds a list of all evaluation metrics used evaluate the performance of each model

```
In [23]:  def draw_table_for_current_dataset(dataset, analysis_result, model_list, metric_list):
              table_data = ""
              #content_width = 25
              table_width = 100
              next_line = "\n"

              horizontal_bar = ""
              for count in range(table_width):
                  horizontal_bar += "="

              horizontal_bar += next_line

              table_data += horizontal_bar
              pos = 1
              item_count = len(metric_list) + 1
              content_size = int( table_width / item_count )
              table_data += set_table_content("", pos, item_count, content_size)
              for metric_ind in range(len(metric_list)):
                  pos += 1
                  table_data += set_table_content(metric_list[metric_ind], pos, item_count, content_size)

              table_data += horizontal_bar

              for model in model_list:
                  pos = 1
                  table_data += set_table_content(model, pos, item_count, content_size)
                  for metric_ind in range(len(metric_list)):
                      pos += 1
                      table_data += set_table_content(to_two_decimal_place(analysis_result[dataset][model][me

                  table_data += horizontal_bar

              print(table_data)
              print()
              print()
              print()
              print()
```

Figure 33. The function used to generate analysis output summary table for a particular dataset

Figure 34 below shows the function in the code implementation of the project that is used to iteratively draw the summary tables for all dataset used in this code implementation for this research project showing all machine learning and neural network models implemented against all evaluation metrics. The function accepted four arguments which included ***dataset_list*** used to store a list of all dataset, ***analysis_result*** used to hold all output from analysing models, ***model_list*** holds a list of all models analysed and ***metric_list*** holds a list of all evaluation metrics used evaluate the performance of each model

```
In [24]:  def draw_summary_table_dataset_evaluation(analysis_result, dataset_list, model_list, metric_list):
              for dataset in dataset_list:
                  print()
                  print("\nModel Summary for " + dataset.upper() + " dataset")
                  print("==========================================================")
                  print("==========================================================\n")
                  print()
                  draw_table_for_current_dataset(dataset, analysis_result, model_list, metric_list)
                  print()
                  print()
                  print()
                  print()
                  print()
```

Figure 34. The function used to generate analysis output summary table for all dataset

Figure 35 below shows the function in the code implementation of the project that is used to convert a metric score to percentage with two decimal place value. The function accepts one argument ***num*** which is the number to be displayed as a percentage with two decimal places

```
In [25]:  def to_two_decimal_place(num):
              mul_val = 100 * num
              return "{:.2f}".format(mul_val)
```

Figure 35. The function used to convert analysis score to percentage with two decimal places

Figure 36 below shows the function in the code implementation of the project that is used to render a plot for all evaluation metrics for a particular model and dataset. The function accepts five arguments and uses Python matplotlib library to plot a bar chart.

```
In [26]: def plot_performance_for_target_model(dataset, analysis_result, model, names, values):
             fig = plt.figure()
             ax = fig.add_axes([0,0,1,1])
             ax.bar(names,values)
             plt.xlabel('Evaluation Metrics')
             plt.ylabel('Performance measured in Percentage(%)')
             plt.title("Different Metrics of " + model + " Model")
             plt.show()
             print()
             print()
             print()
```

Figure 36. The function used to plot bar chart for particular model and a particular dataset

Figure 37 below shows the function in the code implementation of the project that is used to generate four set of bar chart for a particular dataset. The function iteratively create bar chart for the list of model implement to analyse a dataset by invoke ***plot_performance_for_target_model()*** function as shown in figure 36 above. The function accepts four arguments and these includes ***dataset, analysis_result, model_list and metric_list.***

```
In [27]: def show_plot_for_current_dataset(dataset, analysis_result, model_list, metric_list):
             for model in model_list:
                 names = list()
                 values = list()

                 for metric in metric_list:
                     names.append(metric)
                     values.append(analysis_result[dataset][model][metric])

                 plot_performance_for_target_model(dataset, analysis_result, model, names, values)
```

Figure 37. The function is used to generate the data used to plot the bar chart and display the bar chart for a particular model and a particular dataset

Figure 38 below shows the function in the code implementation of the project that is used to iteratively generate bar chat for all dataset analysed in this research work. The function accepts four arguments which includes ***dataset_list*** used to store a list of all dataset, ***analysis_result*** used to hold all output from analysing models, ***model_list*** holds a list of all models analysed and ***metric_list*** holds a list of all evaluation metrics used evaluate the performance of each model.

```
In [28]: def show_model_performance_plot(analysis_result, dataset_list, model_list, metric_list):
             for dataset in dataset_list:
                 print()
                 print("\nModel Performance Plot for " + dataset.upper() + " dataset")
                 print("==========================================================")
                 print("==========================================================\n")
                 print()
                 show_plot_for_current_dataset(dataset, analysis_result, model_list, metric_list)
                 print()
                 print()
                 print()
                 print()
                 print()
```

Figure 38. The function is used to display the bar chart for all implemented models and datasets

Figure 39 below shows the function in the code implementation of the project that is used to generate multiple bar chart for a particular dataset for all evaluation metrics group by models

implemented. The function accepts four arguments and they include ***dataset, analysis_result, model_list and metric_list*** and uses Python matplotlib library to plot multiple bar chart.

```
In [29]:  def show_multiple_plot_for_current_dataset(dataset, analysis_result, model_list, metric_list):
              accuracy_scor = list()
              precision_scor = list()
              recall_scor = list()
              auc_scor = list()

              for model in model_list:
                  for metric in metric_list:
                      if metric.lower() == 'accuracy':
                          accuracy_scor.append(analysis_result[dataset][model][metric])
                      elif metric.lower() == 'precision':
                          precision_scor.append(analysis_result[dataset][model][metric])
                      elif metric.lower() == 'recall':
                          recall_scor.append(analysis_result[dataset][model][metric])
                      elif metric.lower() == 'auc':
                          auc_scor.append(analysis_result[dataset][model][metric])
                      else:
                          pass

              x_axis = np.arange(len(model_list))

              plt.figure(figsize=(20,7))

              plt.bar(x_axis + 0.00, accuracy_scor, width = 0.25, label = 'Accuracy')
              plt.bar(x_axis + 0.25, precision_scor, width = 0.25, label = 'Precision')
              plt.bar(x_axis + 0.50, recall_scor, width = 0.25, label = 'Recall')
              plt.bar(x_axis + 0.75, auc_scor, width = 0.25, label = 'AUC')

              plt.xticks(x_axis, model_list)
              plt.xlabel('Evaluation Metrics')
              plt.ylabel('Performance measured in Percentage(%)')
              #plt.title("Different Metrics of " + model + " Model")
              plt.legend()
              plt.show()
              print()
              print()
              print()
```

Figure 39. The function is used to plot a multiple bar chart for all implemented models and a particular dataset

Figure 40 below shows the function in the code implementation of the project that is used to iteratively generate multipe bar chat for all dataset analysed in this research work. The function accepts four arguments which includes ***dataset_list***, ***analysis_result***, ***model_list*** and ***metric_list***. The function also invokes ***show_multiple_plot_for_current_dataset()*** as it help in the multiple bar chart creation.

```
In [30]:  def show_model_performance_in_one_plot(analysis_result, dataset_list, model_list, metric_list):
              for dataset in dataset_list:
                  print()
                  print("\nModel Performance Using Multiple Bar Plot for " + dataset.upper() + " dataset")
                  print("==========================================================")
                  print("==========================================================\n")
                  print()
                  show_multiple_plot_for_current_dataset(dataset, analysis_result, model_list, metric_list)
                  print()
                  print()
                  print()
                  print()
                  print()
```

Figure 40. The function is used to display multiple bar chart for all implemented models and datasets

4.5 Read dataset using Pandas' read_csv() method

In carrying out the code implementation, three well datasets were used to analyse the models implemented and these datasets includes KDD99 dataset, CIC IDS dataset and NSL KDD dataset. These datasets were downloaded from Kaggle website and Python's Pandas library was used load the datasets as Panda's dataframe into the Jupyter notebook IDE. Figure 41 below show the code snippet used to import all datasets into the IDE

```
In [31]:
ids_nslkdd_filename_1 = 'KDDTrain+.txt'
ids_nslkdd_filename_2 = 'KDDTest+.txt'
ids_nslkdd_filename_3 = 'KDDTrain+_20Percent.txt'

ids_kdd99_filename = 'ids_kddcup99.csv'

ids_cic_filename = 'ids_cic.csv'

ids_nslkdd_columns =(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','w
                      ,'hot','num_failed_logins','logged_in','num_compromised','root_shell','su_atte
                      ,'num_file_creations','num_shells','num_access_files','num_outbound_cmds','is_
                      ,'count','srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_rerror
                      ,'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','d
                      ,'dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_hos
                      ,'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate',

try:
    ids_nslkdd_df_1 = pd.read_csv(os.getcwd() + "/dataset/" + ids_nslkdd_filename_1)
    ids_nslkdd_df_2 = pd.read_csv(os.getcwd() + "/dataset/" + ids_nslkdd_filename_2)
    ids_nslkdd_df_3 = pd.read_csv(os.getcwd() + "/dataset/" + ids_nslkdd_filename_3)

    ids_kdd99_df = pd.read_csv(os.getcwd() + "/dataset/" + ids_kdd99_filename)
    ids_cic_df = pd.read_csv(os.getcwd() + "/dataset/" + ids_cic_filename)

    ids_nslkdd_df_1.columns = ids_nslkdd_columns
    ids_nslkdd_df_2.columns = ids_nslkdd_columns
    ids_nslkdd_df_3.columns = ids_nslkdd_columns

    ids_nslkdd_df = pd.concat([ids_nslkdd_df_1, ids_nslkdd_df_2, ids_nslkdd_df_3], ignore_index=Tru

    ids_nslkdd_df.columns =   ids_nslkdd_df.columns.str.strip()
    ids_nslkdd_df.columns =   ids_nslkdd_df.columns.str.replace(',', '')
    ids_nslkdd_df.columns =   ids_nslkdd_df.columns.str.replace(' ', '_')

except NameError:
    ids_kdd99_df = None
    ids_nslkdd_df = None
    ids_cic_df = None

    print("\n\n")
    print(NameError)
    print("\n\n")

    print("Error: Data set files is not in the root directory.")

    print("""

      Please use the URL to download the dataset from kaggle ===>>>

      https://www.kaggle.com/toobajamal/kdd99-dataset?select=kddcup99_csv.csv
      https://www.kaggle.com/asthana12/cicids2017/download
      https://www.kaggle.com/avk256/nsl-kdd-anomaly-detection/data

      """)
```

Figure 41. code snippet used to import dataset into the IDE

4.6 Data Exploration, Run Experiments and Experiment Result

After successfully importing the datasets into the Jupyter notebook IDE, the datasets were pre-processed, missing values were removed and column names were modified by removing space characters, hyphen characters, underscore characters, other special characters. Figures 42, 43 and 44 below shows the first five rows in each dataset used in model analysis.

```
In [48]: ids_kdd99_df.head()
```

Out[48]:

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	hot	...	dsthostsamesrvrate	dsthostdiffsrvrate	dsthostsamesrcpo
0	0	tcp	http	SF	181	5450	0	0	0	0	...	1.0	0.0	
1	0	tcp	http	SF	239	486	0	0	0	0	...	1.0	0.0	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	1.0	0.0	
3	0	tcp	http	SF	219	1337	0	0	0	0	...	1.0	0.0	
4	0	tcp	http	SF	217	2032	0	0	0	0	...	1.0	0.0	

5 rows x 43 columns

Figure 42. A view of the first five records in the KDD99 dataset

```
In [49]: ids_nslkdd_df.head()
```

Out[49]:

	duration	protocoltype	service	flag	srcbytes	dstbytes	land	wrongfragment	urgent	hot	...	dsthostdiffsrvrate	dsthostsamesrcportrate	dsthostsrvdiff
0	0	udp	other	SF	146	0	0	0	0	0	...	0.60	0.88	
1	0	tcp	private	S0	0	0	0	0	0	0	...	0.05	0.00	
2	0	tcp	http	SF	232	8153	0	0	0	0	...	0.00	0.03	
3	0	tcp	http	SF	199	420	0	0	0	0	...	0.00	0.00	
4	0	tcp	private	REJ	0	0	0	0	0	0	...	0.07	0.00	

5 rows × 44 columns

Figure 43. A view of the first five records in the NSL KDD dataset

```
In [50]: ids_cic_df.head()
```

Out[50]:

	destinationport	flowduration	totalfwdpackets	totalbackwardpackets	totallengthoffwdpackets	totallengthofbwdpackets	fwdpacketlengthmax	fwdpacketlen
0	54865	3	2	0	12	0	6	
1	55054	109	1	1	6	6	6	
2	55055	52	1	1	6	6	6	
3	46236	34	1	1	6	6	6	
4	54863	3	2	0	12	0	6	

5 rows × 80 columns

Figure 44. A view of the first five records in the CIC IDS dataset

Figure 45 below shows the use of the ***plot_pie_chart_for_dependent_variable()*** function to generate pie chart displaying percentage ratio for Attack traffic to Non Attack traffic in each dataset used in this project code implementation. From the pie chart labelled KDD 99 dataset, 80.3% of the network traffic were Attack and 19.7% were Non-Attack traffic. The CIC IDS dataset had 56.7% of the network traffic were Attack and 43.3% were Non-Attack traffic and NSL KDD dataset had 49.1% of the network traffic were Attack and 50.9% were Non-Attack traffic.

```
In [51]: dataset_tuple = ids_kdd99_df, ids_cic_df, ids_nslkdd_df
         title_tuple = "KDD 99 DATASET", "CIC IDS DATASET", "NSL KDD DATASET"

         plot_pie_chart_for_dependent_variable(dataset_tuple, title_tuple, ids_category_label, "Attack to Non-Attack Ratio in all data

         {1: 'r', 0: 'gray'}
         Int64Index([0, 1], dtype='int64')
         Int64Index([0, 1], dtype='int64')
         Int64Index([0, 1], dtype='int64')
```

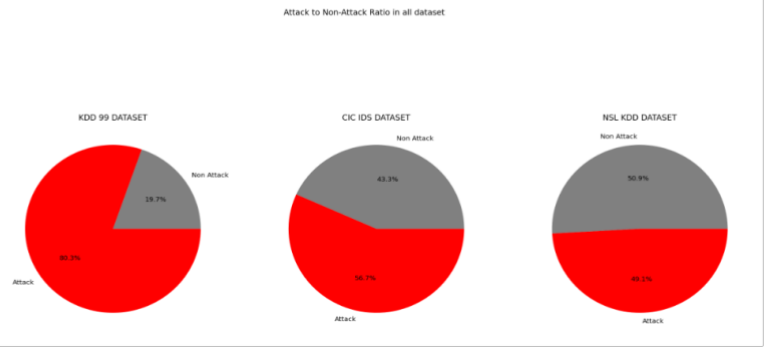

Figure 45. Pie charts showing the Attack traffic to Non Attack traffic ratio in each dataset used in this code implementation

Figure 46 below shows the use of the ***plot_data_correlation()*** function to generate correlation chart using Python Seaborn ***heatmap()*** method and Pandas dataframe ***corr()*** method on each dataset used in this code implementation. The function displays three correlation charts as shown figures 47, 48 and 49 below.

```
In [54]:  dataset_tuple = features_ids_kdd99, features_ids_cic, features_ids_nslkdd
          plot_data_correlation(dataset_tuple)
```

Figure 46. The code snippet showing the invocation plot_data_correlation() function

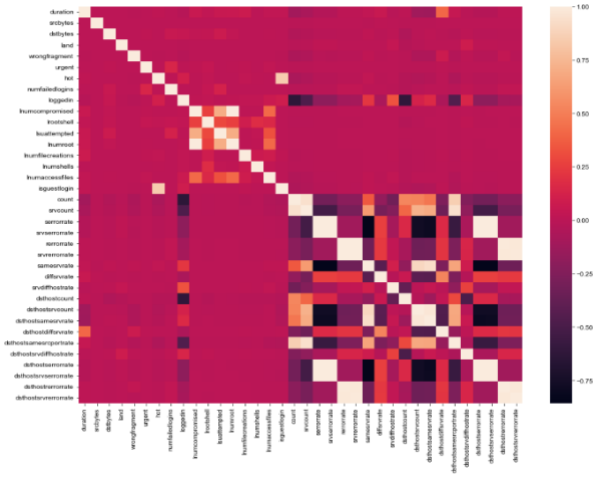

Figure 47. Correlation heat map for KDD99 dataset

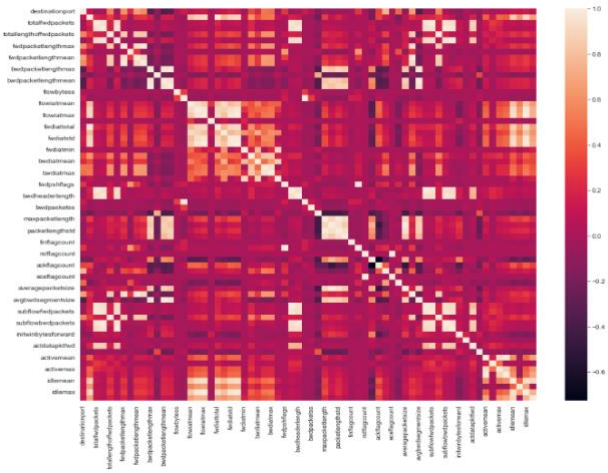

Figure 48. Correlation heat map for CIC IDS dataset

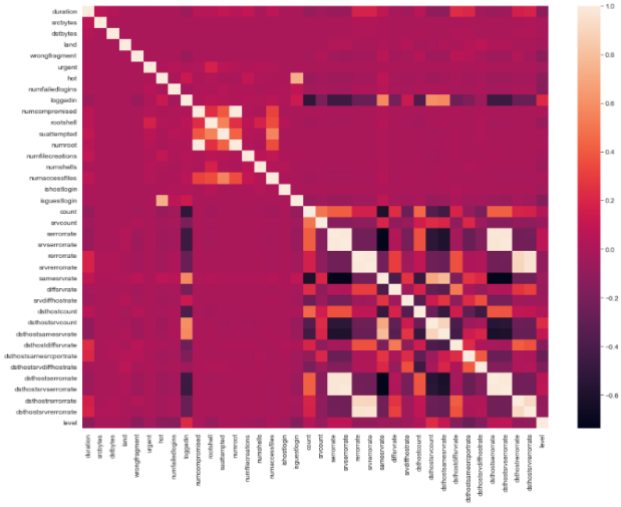

Figure 49. Correlation heat map for NSL KDD dataset

Figure 50 below shows the use of the ***estimate_correlation_without_negative_correlation()*** function to generate a list highly correlated features column names from independent variables in KDD99 dataset. The generated list of highly correlated features will be used to drop the columns from the KDD99 dataset.

```
In [55]: corr_features_ids_kdd99 = estimate_correlation_without_negative_correlation(features_ids_kdd99, 0.85)
         corr_features_ids_kdd99

Out[55]: {'dsthostrerrorrate',
          'dsthostsamesrcportrate',
          'dsthostsamesrvrate',
          'dsthostserrorrate',
          'dsthostsrvcount',
          'dsthostsrvrerrorrate',
          'dsthostsrvserrorrate',
          'lnumroot',
          'srvcount',
          'srvrerrorrate',
          'srvserrorrate'}
```

Figure 50. Code snippets used to retrieve highly correlated features in KDD99 dataset

Figure 51 below shows the use of the ***estimate_correlation_without_negative_correlation()*** function to generate a list highly correlated features column names from independent variables in CIC IDS dataset. The generated list of highly correlated features will be used to drop the columns from the CIC IDS dataset.

```
In [56]: corr_features_ids_cic = estimate_correlation_without_negative_correlation(features_ids_cic, 0.85)
         corr_features_ids_cic

Out[56]: {'actdatapktfwd',
          'activemax',
          'activemin',
          'averagepacketsize',
          'avgbwdsegmentsize',
          'avgfwdsegmentsize',
          'bwdheaderlength',
          'bwdiatmax',
          'bwdiatmin',
          'bwdpacketlengthmean',
          'bwdpacketlengthstd',
          'eceflagcount',
          'flowiatmax',
          'flowiatstd',
          'fwdheaderlength',
          'fwdheaderlength.1',
          'fwdiatmax',
          'fwdiatmean',
          'fwdiatstd',
```

Figure 51. Code snippets used to retrieve highly correlated features in CIC IDS dataset

Figure 52 below shows the use of the ***estimate_correlation_without_negative_correlation()*** function to generate a list highly correlated features column names from independent variables in NSL_KDD dataset. The generated list of highly correlated features will be used to drop the columns from the NSL_KDD dataset.

```
In [57]: corr_features_ids_nslkdd = estimate_correlation_without_negative_correlation(features_ids_nslkdd, 0.85)
         corr_features_ids_nslkdd

Out[57]: {'dsthostrerrorrate',
          'dsthostsamesrvrate',
          'dsthostserrorrate',
          'dsthostsrvrerrorrate',
          'dsthostsrvserrorrate',
          'numroot',
          'srvrerrorrate',
          'srvserrorrate'}
```

Figure 52. Code snippets used to retrieve highly correlated features in NSL KDD dataset

Figure 53 below shows the use of generated list of highly correlated features for the three datasets to remove or drop the column on the generated lists. This operation is done by invoking the ***drop()*** on each dataset instance use in this code implementation.

```
In [58]: features_ids_kdd99.drop(corr_features_ids_kdd99, axis=1,  inplace = True)

In [59]: features_ids_cic.drop(corr_features_ids_cic, axis=1,  inplace = True)

In [60]: features_ids_nslkdd.drop(corr_features_ids_nslkdd, axis=1,  inplace = True)
```

Figure 53. Code snippets used to drop highly correlated features in all dataset

Figure 54 below shows the use of ***get_categorical_attributes()*** generate a list categorical attributes in the KDD99 dataset.

```
In [61]: cat_var_ids_kdd99 = get_categorical_attributes(features_ids_kdd99)
         cat_var_ids_kdd99

Out[61]: ['flag',
          'service',
          'attacktype',
          'protocoltype',
          'lnumaccessfiles',
          'land',
          'wrongfragment',
          'numfailedlogins',
          'loggedin',
          'lsuattempted',
          'lnumshells',
          'ishostlogin',
          'isguestlogin',
          'lrootshell',
          'urgent',
          'lnumoutboundcmds']
```

Figure 54. Code snippets used to retrieve categorical variable in KDD99 dataset

Figure 55 below shows the use of ***get_categorical_attributes()*** generate a list categorical attributes in the CIC IDS dataset.

```
In [70]: cat_var_ids_cic = get_categorical_attributes(features_ids_cic)
         cat_var_ids_cic

Out[70]: ['attacktype',
          'cweflagcount',
          'rstflagcount',
          'ackflagcount',
          'fwdpshflags',
          'pshflagcount',
          'fwdavgbulkrate',
          'bwdavgbulkrate',
          'bwdavgpacketsbulk',
          'bwdpshflags',
          'fwdavgbytesbulk',
          'urgflagcount',
          'fwdurgflags',
          'bwdavgbytesbulk',
          'fwdavgpacketsbulk',
          'downupratio',
          'minsegsizeforward',
          'bwdurgflags',
          'finflagcount']
```

Figure 55. Code snippets used to retrieve categorical variable in CIC IDS dataset

Figure 56 below shows the use of ***get_categorical_attributes()*** generate a list categorical attributes in the NSL KDD dataset.

```
In [73]:  cat_var_ids_nslkdd = get_categorical_attributes(features_ids_nslkdd)
          cat_var_ids_nslkdd

Out[73]: ['flag',
          'service',
          'attacktype',
          'protocoltype',
          'land',
          'rootshell',
          'numoutboundcmds',
          'wrongfragment',
          'numfailedlogins',
          'loggedin',
          'suattempted',
          'ishostlogin',
          'isguestlogin',
          'urgent',
          'numshells']
```

Figure 56. Code snippets used to retrieve categorical variable in NSL KDD dataset

Figure 57 below shows the use of ***split_dataset_into_train_test_set()*** to split all datasets into their training dataset and testing dataset respectively. The training set is labelled X_train and y_train, while the testing set is labelled X_test and y_test and the names of the dataset is appended to the variables used to store these data. The variable having the "X" character attached to the variable names are the independent variable while character "y" denotes the dependent variable.

```
In [86]:
X_train_ids_kdd99, X_test_ids_kdd99, y_train_ids_kdd99, y_test_ids_kdd99 = split_dataset_into_train_test_set(features_ids_kdc
X_train_ids_cic, X_test_ids_cic, y_train_ids_cic, y_test_ids_cic = split_dataset_into_train_test_set(features_ids_cic, target
X_train_ids_nslkdd, X_test_ids_nslkdd, y_train_ids_nslkdd, y_test_ids_nslkdd = split_dataset_into_train_test_set(features_ids
```

Figure 57. Code snippets for splitting datasets into training and testing sets

Figure 58 below shows the use MinMaxScalar module to normalize dataset values before attempting to use the dataset in analysing the models implemented. To normalize the dataset, an instance of the ***MinMaxScaler*** module from Python's sklearn.preprocessing library was created. The ***fit_transform()*** method of the instance MinMaxScalar is called and the independent variables in the training and testing set are used as argument on each dataset to normalize the dataset.

```
In [87]:  sc = MinMaxScaler()
          # sc = StandardScaler()

          X_train_ids_kdd99 = sc.fit_transform(X_train_ids_kdd99)
          X_test_ids_kdd99 = sc.fit_transform(X_test_ids_kdd99)

          X_train_ids_cic = sc.fit_transform(X_train_ids_cic)
          X_test_ids_cic = sc.fit_transform(X_test_ids_cic)

          X_train_ids_nslkdd = sc.fit_transform(X_train_ids_nslkdd)
          X_test_ids_nslkdd = sc.fit_transform(X_test_ids_nslkdd)
```

Figure 58. Code snippets used for normalizing the independent variables in of all dataset

Figure 59 below shows the use SMOTE module remove the imbalance in the dataset. To remove the imbalance in the dataset, an instance of the ***SMOTE*** module from Python imblearn.over_sampling library was created and ***fit_resample()*** method was applied to training set for each dataset.

```
In [88]:  random_oversampler = SMOTE()

          X_train_ids_kdd99, y_train_ids_kdd99 = random_oversampler.fit_resample(X_train_ids_kdd99, y_train_ids_kdd99)

          X_train_ids_cic, y_train_ids_cic = random_oversampler.fit_resample(X_train_ids_cic, y_train_ids_cic)

          X_train_ids_nslkdd, y_train_ids_nslkdd = random_oversampler.fit_resample(X_train_ids_nslkdd, y_train_ids_nslkdd)
```

Figure 59. Code snippets used to remove imbalance in the datasets

Figure 60 below shows the addition all normalized and balanced dataset to a dictionary data structure in order to store all datasets record in a single variable.

```
In [92]:  analysis_dataset[kdd99_colname][x_train_label] = X_train_ids_kdd99
          analysis_dataset[kdd99_colname][x_test_label] = X_test_ids_kdd99
          analysis_dataset[kdd99_colname][y_train_label] = y_train_ids_kdd99
          analysis_dataset[kdd99_colname][y_test_label] = y_test_ids_kdd99

          analysis_dataset[cic_colname][x_train_label] = X_train_ids_cic
          analysis_dataset[cic_colname][x_test_label] = X_test_ids_cic
          analysis_dataset[cic_colname][y_train_label] = y_train_ids_cic
          analysis_dataset[cic_colname][y_test_label] = y_test_ids_cic

          analysis_dataset[nslkdd_colname][x_train_label] = X_train_ids_nslkdd
          analysis_dataset[nslkdd_colname][x_test_label] = X_test_ids_nslkdd
          analysis_dataset[nslkdd_colname][y_train_label] = y_train_ids_nslkdd
          analysis_dataset[nslkdd_colname][y_test_label] = y_test_ids_nslkdd
```

Figure 60. Code snippets for adding both training and testing dataset to analysis_dataset variable

Figure 61 below shows the code execution to run logistic regression analysis for all datasets in this project code implementation.

```
In [93]:  perform_logistic_regression_modelling_and_analysis(analysis_dataset, analysis_result)

          Training time for LR model on kdd99 dataset:  13.67406177520752
          Testing time for LR model on kdd99 dataset:  0.02100229263305664

          Analysis Summary for LR model on kdd99 dataset
          ==============================================================================
          ==============================================================================


          Classication Report

                        precision    recall  f1-score   support

               Normal       1.00      0.98      0.99     30054
               Attack       0.99      1.00      1.00    118152

             accuracy                           0.99    148206
            macro avg       1.00      0.99      0.99    148206
         weighted avg       0.99      0.99      0.99    148206
```

Figure 61. Code snippets for executing Logistic Regression analysis

Figure 62 below shows the code execution to run random forest classification analysis for all datasets in this project code implementation.

```
In [94]:  perform_random_forest_classifier_modelling_and_analysis(analysis_dataset, analysis_result)

          Training time for RFC model on kdd99 dataset:  14.852149486541748
          Testing time for RFC model on kdd99 dataset:  0.37799978256225586

          Analysis Summary for RFC model on kdd99 dataset
          ==============================================================================
          ==============================================================================


          Classication Report

                        precision    recall  f1-score   support

               Normal       1.00      1.00      1.00     29355
               Attack       1.00      1.00      1.00    118851

             accuracy                           1.00    148206
            macro avg       1.00      1.00      1.00    148206
         weighted avg       1.00      1.00      1.00    148206
```

Figure 62. Code snippets for executing Random Forest classification analysis

Figure 63 below shows the code execution to run gradient boost classification analysis for all datasets in this project code implementation.

```
In [95]:  perform_gradient_boost_classifier_modelling_and_analysis(analysis_dataset, analysis_result)

Training time for GBC model on kdd99 dataset:  106.10530543327332
Testing time for GBC model on kdd99 dataset:  0.33299684524536133

Analysis Summary for GBC model on kdd99 dataset
==============================================================================
==============================================================================


Classication Report

              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     29355
      Attack       1.00      1.00      1.00    118851

    accuracy                           1.00    148206
   macro avg       1.00      1.00      1.00    148206
weighted avg       1.00      1.00      1.00    148206
```

Figure 63. Code snippets for executing Gradient Boost classification analysis

Figure 64 below shows the code execution to run artificial neural network (ANN) analysis for all datasets in this project code implementation.

```
In [96]:  run_ANN_model(analysis_dataset, analysis_result)

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 31)                992
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 32
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 4
=================================================================
Total params: 1,028
Trainable params: 1,028
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 31)                992
```

Figure 64. Code snippets for executing Artificial Neural Network (ANN) analysis

Figure 65 below shows the code execution for displaying tabular summary of all dataset analysed showing the four model implement and the four evaluation metrics used in the analysis.

```
In [97]:  dataset_list = [
              nslkdd_colname, kdd99_colname, cic_colname
          ]

          model_list = [
              'LR', 'RFC', 'GBC', 'ANN'
          ]

          metric_list = [
              'Accuracy', 'Precision', 'Recall', 'AUC'
          ]

          draw_summary_table_dataset_evaluation(analysis_result, dataset_list, model_list, metric_list)
```

Figure 65. Code snippets used to display model analysis tables for all dataset

Model Summary for NSLKDD dataset

	Accuracy	Precision	Recall	AUC
LR	99.60	99.42	99.75	99.60
RFC	100.00	100.00	100.00	100.00
GBC	100.00	100.00	100.00	100.00
ANN	100.00	100.00	100.00	100.00

Figure 66. Summary table showing all implemented models and all evaluation metrics for NSL KDD dataset

Model Summary for KDD99 dataset

	Accuracy	Precision	Recall	AUC
LR	99.48	99.97	99.38	99.63
RFC	100.00	100.00	100.00	100.00
GBC	100.00	100.00	100.00	100.00
ANN	99.99	99.98	100.00	99.97

Figure 67. Summary table showing all implemented models and all evaluation metrics for KDD99 dataset

Model Summary for CIC dataset

	Accuracy	Precision	Recall	AUC
LR	100.00	100.00	100.00	100.00
RFC	100.00	100.00	100.00	100.00
GBC	100.00	100.00	100.00	100.00
ANN	100.00	100.00	100.00	100.00

Figure 68. Summary table showing all implemented models and all evaluation metrics for CIC IDS dataset

Figure 69 below shows the code execution for displaying all bar chart of all dataset analysed showing the four model implement and the four evaluation metrics used in the analysis.

```
In [98]: show_model_performance_plot(analysis_result, dataset_list, model_list, metric_list)
```

Figure 69. Code snippets used to display bar charts for all model and evaluation metrics for all dataset

Figure 70 below shows the output of the execution of figure 69 above for the NSL KDD dataset for the four models implemented. The plot labelled ***A*** is the bar chart for Logistic Regression analysis, ***B*** is the bar chart for Random Forest classification analysis, ***C*** is the bar chart for Gradient Boost classification analysis and ***D*** is for Artificial Neural Network (ANN) analysis.

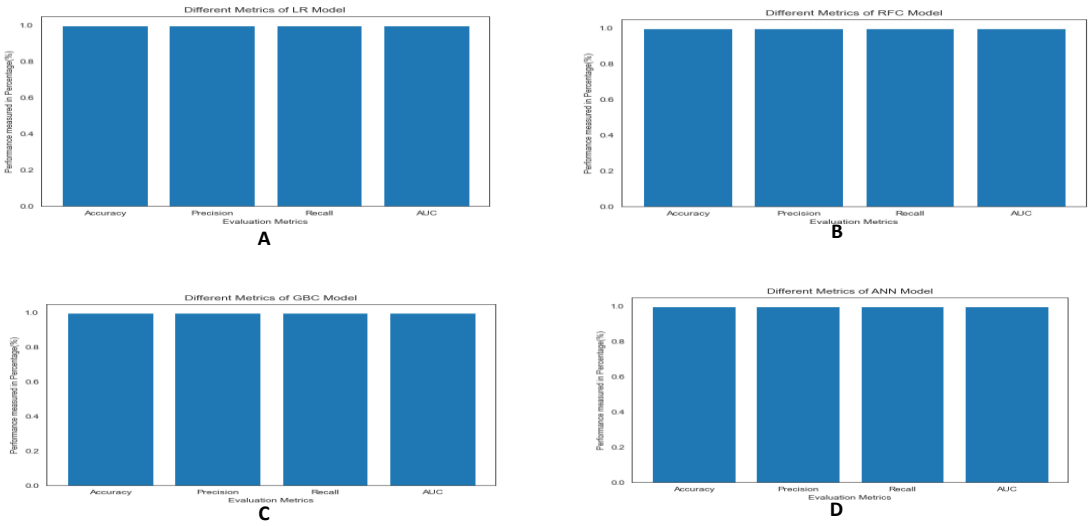

Figure 70. Bar plots for all models implemented and evaluation metrics for NSL KDD dataset

Figure 71 below shows the output of the execution of figure 69 above for the KDD 99 dataset for the four models implemented. The plot labelled ***A*** is the bar chart for Logistic Regression analysis, ***B*** is the bar chart for Random Forest classification analysis, ***C*** is the bar chart for Gradient Boost classification analysis and ***D*** is for Artificial Neural Network (ANN) analysis.

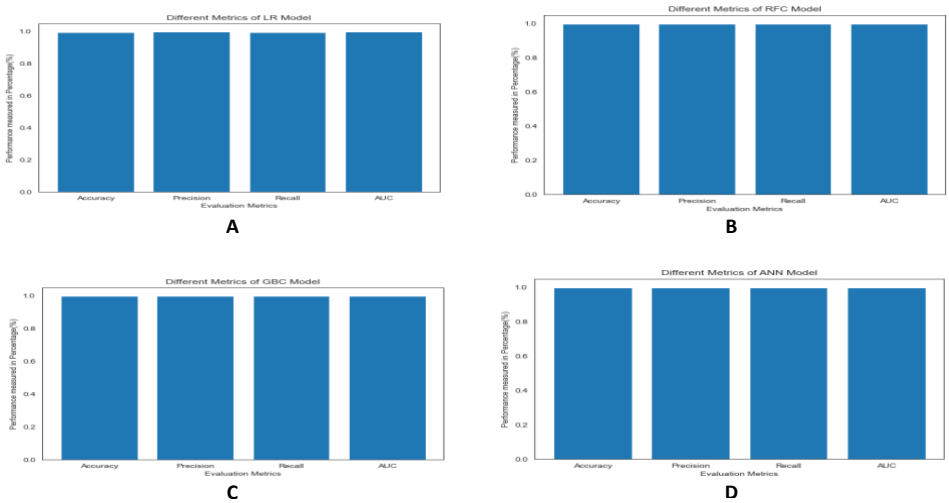

Figure 71. Bar plots for all models implemented and evaluation metrics for KDD 99 dataset

Figure 72 below shows the output of the execution of figure 69 above for the CIC IDS dataset for the four models implemented. The plot labelled ***A*** is the bar chart for Logistic Regression analysis, ***B*** is the bar chart for Random Forest classification analysis, ***C*** is the bar chart for Gradient Boost classification analysis and ***D*** is for Artificial Neural Network (ANN) analysis.

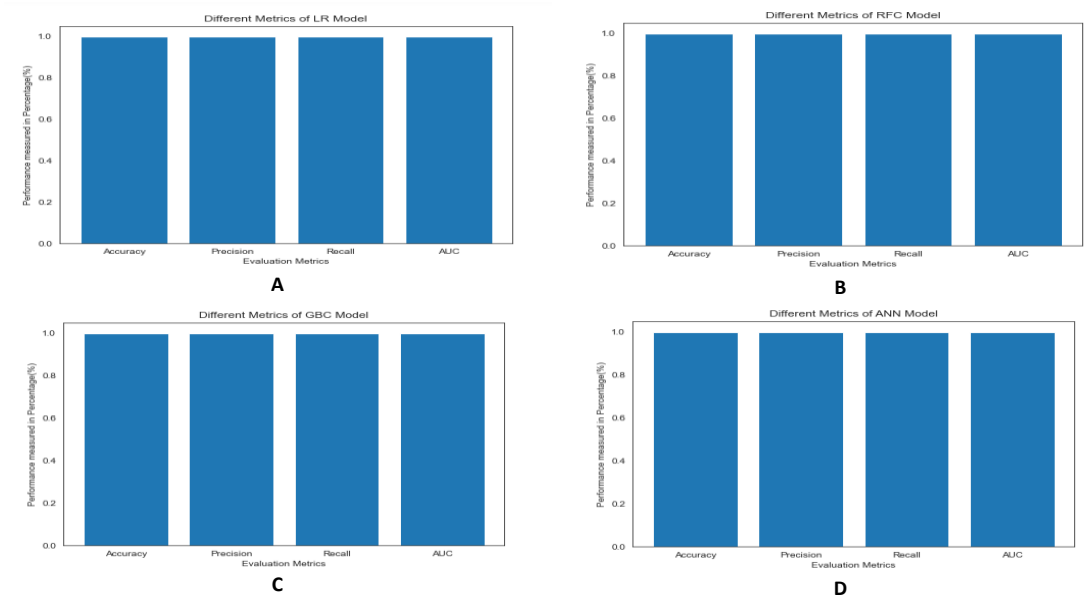

Figure 72. Bar plots for all models implemented and evaluation metrics for CIC IDS dataset

Figure 73 below shows the code execution for displaying all multiple bar chart of all dataset analysed showing the four model implement and the four evaluation metrics used in the analysis.

```
In [99]:  show_model_performance_in_one_plot(analysis_result, dataset_list, model_list, metric_list)

Model Performance Using Multiple Bar Plot for NSLKDD dataset
```

Figure 73. Code snippets used to display multiple bar charts for all model and evaluation metrics for all dataset

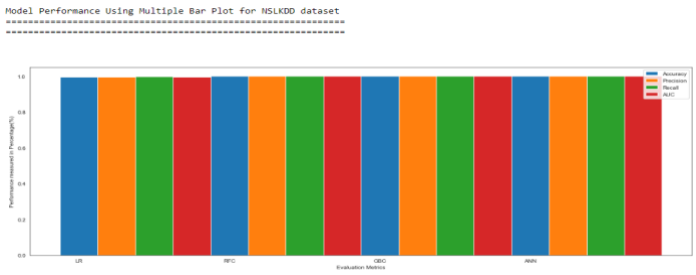

Figure 74. Multiple bar plots for all models implemented and evaluation metrics for NSL KDD dataset

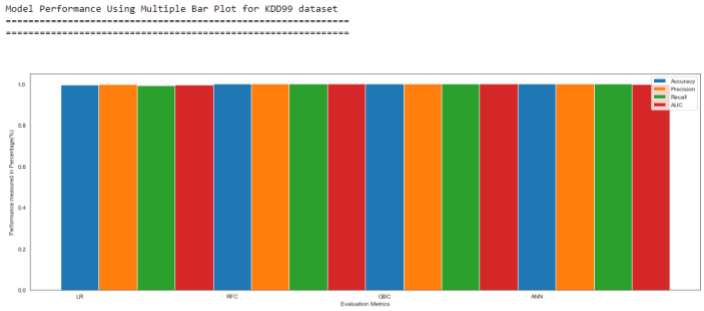

Figure 75. Multiple bar plots for all models implemented and evaluation metrics for KDD 99 dataset

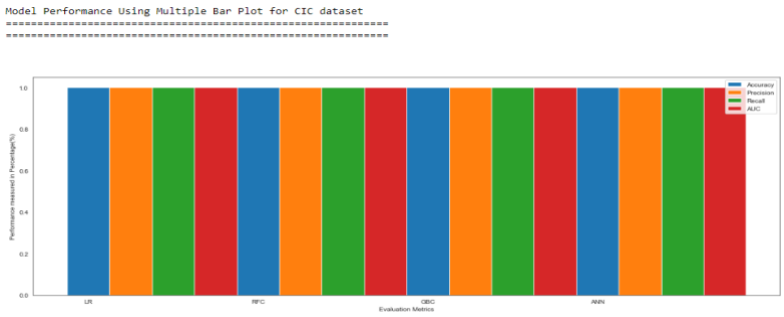

Figure 76. Multiple bar plots for all models implemented and evaluation metrics for CIC IDS dataset

5 Conclusion

The guidelines enumerated in this configuration manual documentation will aid researcher who intends to implement same code implementation as described in this research report using all three datasets is guaranteed to get the same outcomes as the result obtained in this work. The code snippets, the charts generated and summary table shown in this report were used to achieve the objectives and goal as set form the outset of the project work.

6 References

Docs.anaconda.com. 2022. *Installing on Windows — Anaconda documentation*. [online] Available at: <https://docs.anaconda.com/anaconda/install/windows/> [Accessed 15 April 2022].

Docs.conda.io. 2022. *Managing environments — conda 4.12.0.post33+077616b2 documentation*. [online] Available at: <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html> [Accessed 15 April 2022].

En.wikipedia.org. 2022. *Anaconda (Python distribution) - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)> [Accessed 15 April 2022].

En.wikipedia.org. 2022. *Project Jupyter - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Project_Jupyter#Jupyter_Notebook> [Accessed 15 April 2022].

Kaggle.com. 2022. *CIC-IDS-2017*. [online] Available at: <https://www.kaggle.com/datasets/asthana12/cicids2017> [Accessed 15 April 2022].

Kaggle.com. 2022. *KDD99 dataset*. [online] Available at: <https://www.kaggle.com/datasets/toobajamal/kdd99-dataset> [Accessed 15 April 2022].

Kaggle.com. 2022. *NSL-KDD Anomaly detection*. [online] Available at: <https://www.kaggle.com/avk256/nsl-kdd-anomaly-detection/data> [Accessed 15 April 2022].