National College of Ireland

Configuration Manual

Academic Internship
MSc Cyber Security

Benhur Kachhap
Student ID: 22168494

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Benhur Kachhap
Student ID:	22168494
Programme:	MSc Cyber Security **Year:** 2023
Module:	Academic Internship
Supervisor:	Vikas Sahni
Submission Due Date:	14/12/2023
Project Title:	Novel Approach in Intrusion Detection Systems Using Mutual Information-based Gradient Boosting Machine
Word Count:	1570 **Page Count** 19

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: Benhur Kachhap

Date: 14/12/2023

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

Benhur Kachhap
Student ID: 22168494

1 Introduction

The purpose of this document is to explain the implementation process undertaken during this research project. This document also includes the hardware and software requirements used during implementation. This configuration manual also contains snippets of the Python code that was used during the development of the project "**Novel Approach in Intrusion Detection Systems Using Mutual Information-based Gradient Boosting Machine.**"

1.1 System Hardware

The system hardware which was used during this research project development was as follows:

- **Processor:** AMD Ryzen 5 5625U with Radeon Graphics, 2301 Mhz, 6 Core(s), 12 Logical Processor(s)
- **Installed Physical Memory (RAM):** 16.0 GB
- **Operating System:** Microsoft Windows 11 Home Single Language Version: 10.0.22.621 Build 22621
- **GPU:** AMD Radeon (TM) Graphics
- **Storage:** 1TB HDD and 512 SSD

1.2 System Software

- Python ver. 3.9.13 [MSC v.1916 64 bit (AMD64)]
- Jupyter Notebook 6.4.12
- Anaconda Navigator 2.3.2

1.3 Software Installation

A detailed description of the steps taken in installing the tools is presented here:

• Python 3.9.13 can be downloaded and installed from
https://www.python.org/ftp/python/3.9.13/python-3.9.13-embed-amd64.zip

• Visit Anaconda's website to download and install the Anaconda Navigator to have the Jupyter Notebook application which allows you to create and edit documents that display the input and output of a Python script.

2 Project Development

The entire project was coded in the Python language with the use of Jupyter Notebook. Some dataset exploration was performed using Microsoft Excel. The implementation is separated into Dataset preparation, Feature Selection, Classification models, and Evaluation. This section of the configuration manual guides you through how to set up the Python environment for building the ML models and executing them.

2.1 Libraries Utilized

To implement this project, the following Python libraries were used:

- Scikit-Learn: 1.0.2
- Numpy: 1.21.5
- Seaborn: 0.11.2
- Pandas: 1.4.4
- Matplotlib: 3.5.2
- Scipy: 1.9.1

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
import time
```

2.2 Dataset Preparation

The project commenced with a large CSV file containing extensive network traffic records. This dataset, comprising 37412 rows and 34 columns, represented port statistics and their respective descriptions per port on each switch within the simulated Software-Defined Network (SDN). These stats help us understand how each part of the network performs during the simulated data transfer. A crucial aspect of this dataset was the 33rd column, which labeled each record with a specific type of traffic, such as ‘Normal’, ‘Overflow’, ‘PortScan’, ‘TCP-SYN’, ‘Diversion’, and ‘Blackhole’.

The collected dataset is imported to the Jupyter Notebook and the preliminary preparation of the dataset is performed which is described below:

Importing Data

```
# Ask the user for the dataset path
dataset_path = input("Please enter the path to your dataset: ")

# Load the data
try:
    data = pd.read_csv(dataset_path)
    print("Dataset loaded successfully!")
except Exception as e:
    print(f"An error occurred while loading the dataset: {e}")
```

```
Please enter the path to your dataset: D:\Dataset\UNR-IDD.csv
Dataset loaded successfully!
```

Figure 1: Code for Importing the Dataset

To load the dataset, the above Python code is created that will ask the user for the path of the dataset. Once the correct path for the dataset is fed the code will access the dataset.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37411 entries, 0 to 37410
Data columns (total 34 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Switch ID                     37411 non-null  object
 1   Port Number                   37411 non-null  object
 2   Received Packets              37411 non-null  int64
 3   Received Bytes                37411 non-null  int64
 4   Sent Bytes                    37411 non-null  int64
 5   Sent Packets                  37411 non-null  int64
 6   Port alive Duration (S)       37411 non-null  int64
 7   Packets Rx Dropped            37411 non-null  int64
 8   Packets Tx Dropped            37411 non-null  int64
 9   Packets Rx Errors             37411 non-null  int64
 10  Packets Tx Errors             37411 non-null  int64
 11  Delta Received Packets        37411 non-null  int64
 12  Delta Received Bytes          37411 non-null  int64
 13  Delta Sent Bytes              37411 non-null  int64
 14  Delta Sent Packets            37411 non-null  int64
 15  Delta Port alive Duration (S) 37411 non-null  int64
 16  Delta Packets Rx Dropped      37411 non-null  int64
 17   Delta Packets Tx Dropped     37411 non-null  int64
 18  Delta Packets Rx Errors       37411 non-null  int64
 19  Delta Packets Tx Errors       37411 non-null  int64
 20  Connection Point              37411 non-null  int64
 21  Total Load/Rate               37411 non-null  int64
 22  Total Load/Latest             37411 non-null  int64
 23  Unknown Load/Rate             37411 non-null  int64
 24  Unknown Load/Latest           37411 non-null  int64
 25  Latest bytes counter          37411 non-null  int64
 26  is_valid                      37411 non-null  bool
 27  Table ID                      37411 non-null  int64
 28  Active Flow Entries           37411 non-null  int64
 29  Packets Looked Up             37411 non-null  int64
 30  Packets Matched               37411 non-null  int64
 31  Max Size                      37411 non-null  int64
 32  Label                         37411 non-null  object
 33  Binary Label                  37411 non-null  object
dtypes: bool(1), int64(29), object(4)
```

Figure 2: Imported Dataset Characteristics

The names of the attributes and the type of the attribute in the dataset are shown in the above image which is obtained by using the *data.info() function*.
The missing values and the duplicates available in the dataset can be determined by using the below snippet of the code:

Data Pre-Processing

```
# Checking for missing values and duplicates in the dataset
missing_values = data.isnull().sum()
duplicates = data.duplicated()

# Number of duplicate rows
num_duplicates = duplicates.sum()

# Return the summary of missing values and number of duplicates
missing_values, num_duplicates
```

Figure 3: Pre-processing/Cleaning of Dataset

The further cleaning of the dataset is done by dropping the duplicates in the dataset using the code snippet below:

```
# Removing the duplicate row
data_clean = data.drop_duplicates()
```

Figure 4: Removing the Duplicates

2.3 Exploratory Data Analysis (EDA)

- **Histograms:** Histograms are produced by the script to visually represent the distribution of individual numerical features across columns.

Data Visualization

```
# Selecting numerical columns for histograms as large numbers of unique categorical values can lead to cluttered visuals.
numerical_columns = data_clean.select_dtypes(include=['int64', 'float64']).columns

# Plotting histograms
data_clean[numerical_columns].hist(bins=50, figsize=(20,15))
plt.tight_layout()  # This will ensure that the plots do not overlap
plt.show()
```

Figure 5: Code for histogram plotting.

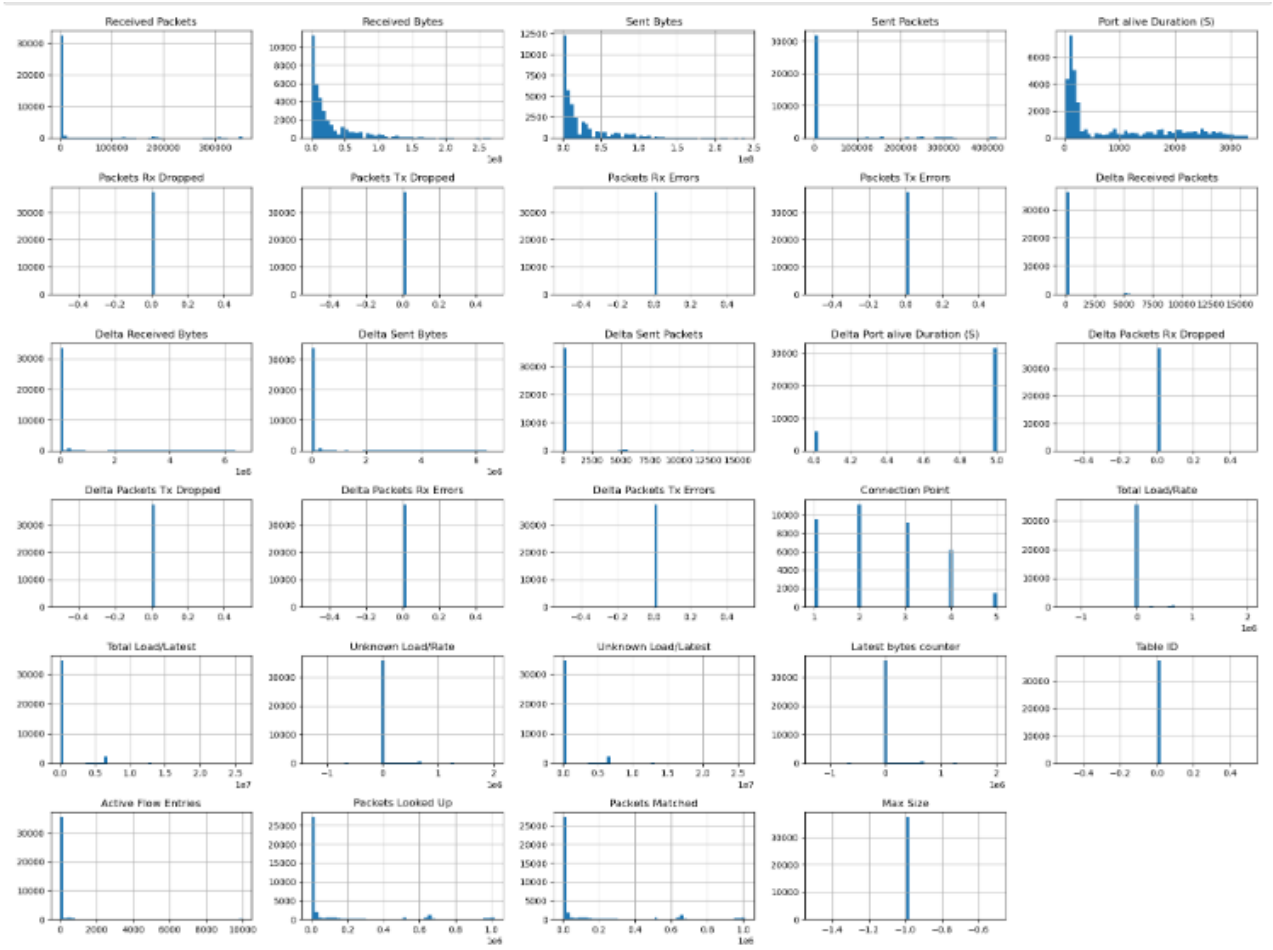

Figure 6: Histogram plots

- **Box Plots:** To identify any outliers and provide a more comprehensive view of the distribution, box graphs are generated for the initial five numerical columns.

```
# For box plots, we'll plot for the first few numerical columns for readability
for column in numerical_columns[:5]:
    plt.figure(figsize=(6, 3))
    sns.boxplot(x=data_clean[column])
    plt.title(f'Box Plot - {column}')
    plt.tight_layout()
    plt.show()
```

Figure 7: Code for histogram plotting

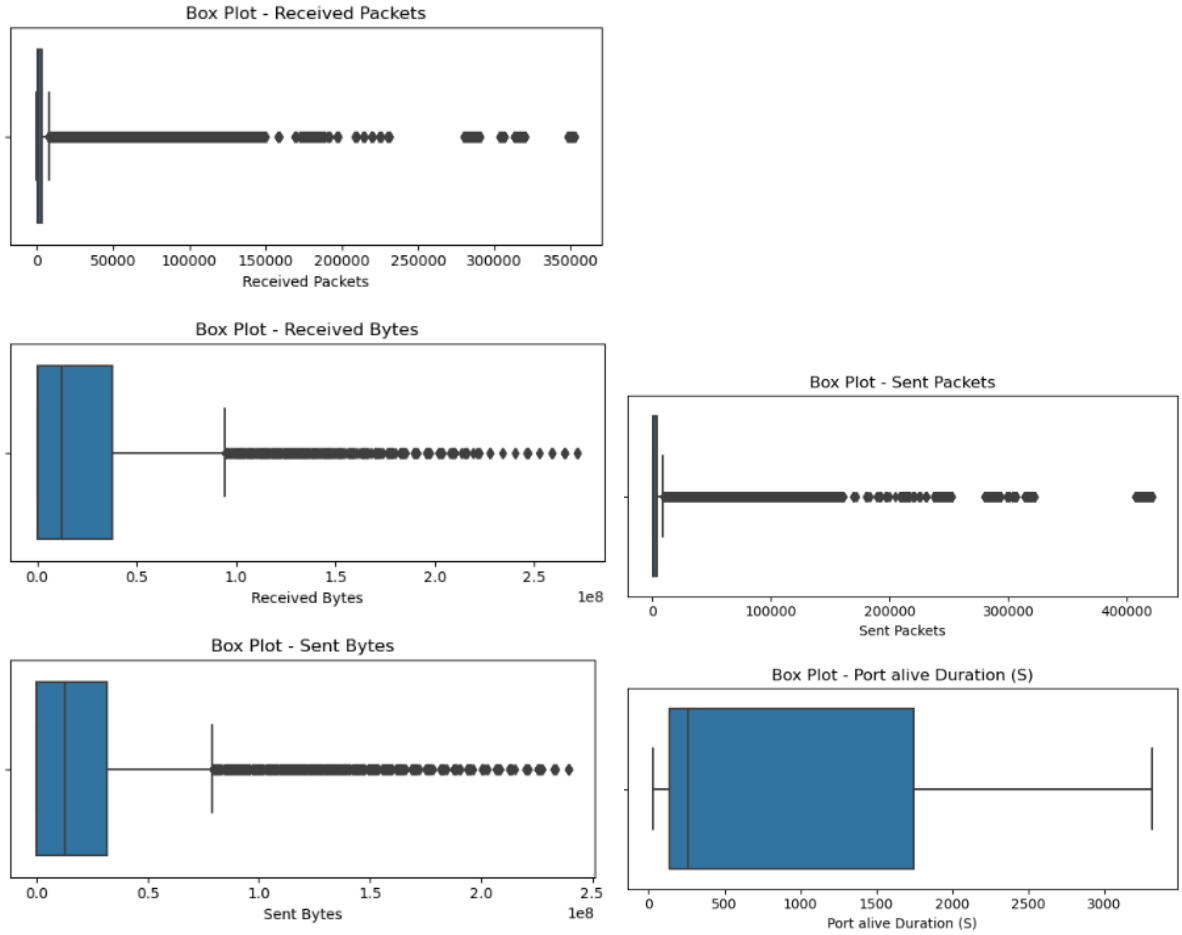

Figure 8: Box plots

- **Pair Plot:** The initial four numerical columns are utilized to generate a pair plot, which offers valuable insights into the distributions and pairwise relationships.

```
# Pair Plot for a subset of variables
selected_columns = numerical_columns[:4]  # Selecting the first 4 numerical columns for the pair plot
sns.pairplot(data_clean[selected_columns], diag_kind="kde")
plt.suptitle('Pair Plot for Selected Variables', y=1.02)  # Making space for the title
plt.show()
```

Figure 9: Code for Pair plotting

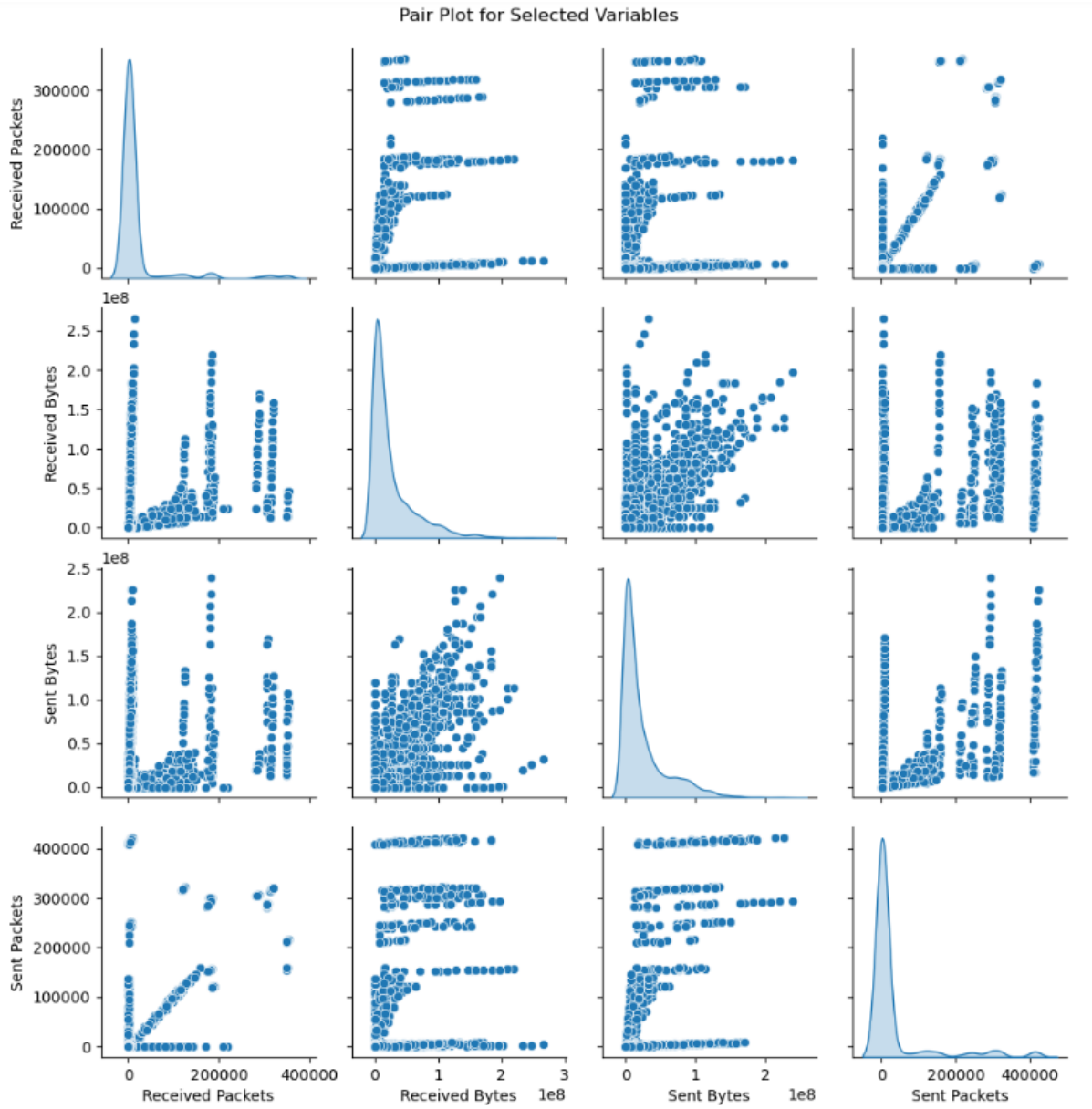

Figure 10: Pair plots

- **Scatter Plots:** The relationship between each of the chosen numerical features and the categorical 'Label' column is visually represented with scatter plots.

```
# Ensure "Label" is categorical and not numerical for proper representation
data_clean['Label'] = data_clean['Label'].astype('category')

for column in selected_columns:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(data=data_clean, x=column, y='Label', hue='Label', palette="deep")
    plt.title(f'Scatter Plot of {column} with Label')
    plt.show()
```

Figure 11: Code for Scatter plotting.

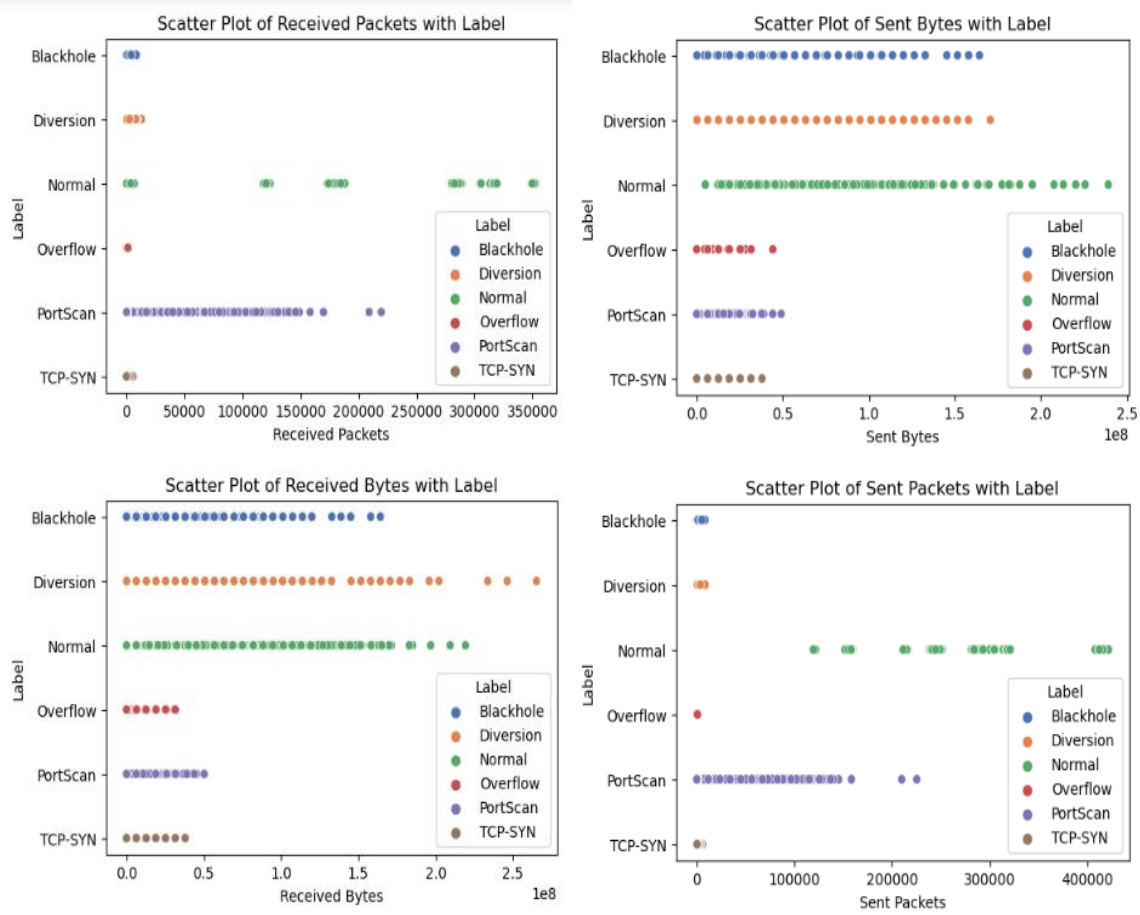

Figure 12: Scatter plots

- **Count Plot:** The 'Label' column is accompanied by a count plot that illustrates the distribution of distinct classes within the dataset presenting the number of instances for each class in a lucid visual manner.

```
#  Count Plot for 'Label'
plt.figure(figsize=(8, 5))
sns.countplot(x='Label', data=data_clean, palette='viridis')
plt.title('Count Plot for Label')
plt.show()
```

Figure 13: Code for Count plot

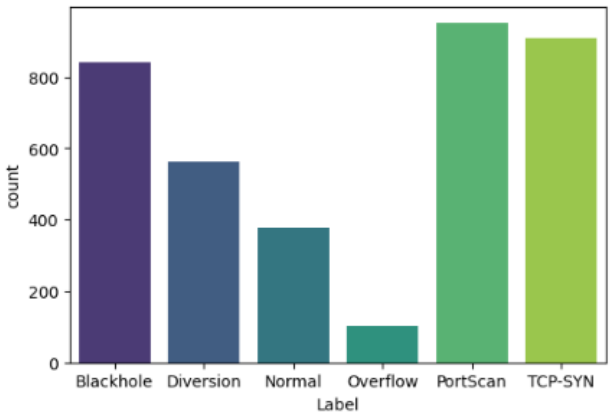

Figure 13: Count plot

2.4 Converting String Values to Numerical

For compatibility with machine learning algorithms, all dataset values needed to be numerical. However, variables like Switch ID, Port Number, and is_valid were in string format. A Python script was developed to convert these strings into numerical values. Each subset underwent this conversion process.

2.5 Splitting and Categorization of the Dataset

The first significant task was to segment the dataset based on the target and independent variables in the dataset. Data is split into 80% training and 20% testing.

Data Preparation (Separating,Splitting, Scaling)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import VarianceThreshold

# Step 1: Data Preparation

# 1. Separate Features and Target
X = data_clean.drop(['Label'], axis=1)  # dropping the target variable and additional labels
y = data_clean['Label']  # we're considering 'Label' as the target variable

# 2. Encoding Categorical Variables
# Identifying categorical columns and converting them
categorical_cols = X.select_dtypes(include=['object', 'category']).columns

# If there are any categorical variables, convert them to numerical
if not categorical_cols.empty:
    for col in categorical_cols:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])

# 3. Train-Test Split
# Splitting the dataset into 70% training data and 20% test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Scale the feature variables to ensure that distance-based algorithms work accurately
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Returning the shapes of the resulting datasets as confirmation
X_train_scaled.shape, X_test_scaled.shape, y_train.shape, y_test.shape

# Perform VarianceThreshold
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(X_train)
X_train_filter = constant_filter.fit_transform(X_train_scaled)
X_test_filter = constant_filter.transform(X_test_scaled)
X_train_filter.shape, X_test_filter.shape
```

```
((29928, 22), (7482, 22))
```

Figure 14: Data Preparation

2.6 Feature Selection

Mutual Information, a statistical measure, is used to gauge the dependency of one variable on another. This approach helps in pinpointing the features that have the most significant impact on the target variable.

Implementation: The Python code uses the mutual_info_classif function from sklearn to calculate mutual information scores for features in the training set. These scores are then sorted to highlight the top features.

Feature Selection using Mutual Information

```
import pandas as pd
from sklearn.feature_selection import mutual_info_classif

# Convert NumPy array to a Pandas DataFrame
X_train_filter_df = pd.DataFrame(X_train_filter, columns=X_train.columns[constant_filter.get_support()])

# Now perform feature selection using mutual information
mutual_info = mutual_info_classif(X_train_filter_df, y_train)
mi_df = pd.DataFrame(mutual_info, index=X_train_filter_df.columns, columns=['Mutual Information'])
mi_df_sorted = mi_df.sort_values(by='Mutual Information', ascending=False)
print("Top features based on mutual information:")
print(mi_df_sorted.head(10))

# Select top features, e.g., top 6 features
top_features = mi_df_sorted.head(6).index
X_train_mi = X_train[top_features]
X_test_mi = X_test[top_features]
X_train_mi.shape, X_test_mi.shape
```

```
Top features based on mutual information:
                        Mutual Information
Port alive Duration (S)            1.128499
Packets Looked Up                  1.089561
Packets Matched                    1.083104
Received Bytes                     0.956963
Sent Bytes                         0.822549
Sent Packets                       0.696671
Received Packets                   0.693659
Binary Label                       0.324002
Delta Sent Bytes                   0.306441
Delta Received Bytes               0.268735

((29928, 6), (7482, 6))
```

Figure 5: Feature Selection using Mutual Information

```
# Display top features based on mutual information in a bar plot
plt.figure()
mi_df_sorted.plot.bar()
plt.title('Top features based on mutual information')
plt.xlabel('Features')
plt.ylabel('Mutual Information')
plt.tight_layout()
plt.show()
```

Figure 7: Code to showcase top MI features in a box plot.

2.7 Classification Models

An assortment of classification models, including Gradient Boosting Classifier, Random Forest Classifier, KNN Neighbours, and Gaussian Naive Bayes is investigated. Adopting a diversified approach enables a thorough comprehension of which models exhibit optimal performance for the given dataset.

Implementation: The provided code illustrates the sequential steps involved in setting up a Gradient Boosting Classifier, training it using the given dataset, and preparing it to generate predictions.

```
# Define available models
available_models = {
    "Gradient Boosting Classifier": GradientBoostingClassifier(n_estimators=50, max_depth=5, learning_rate=0.1, random_state=0),
    "Random Forest Classifier": RandomForestClassifier(),
    "KNN Neighbours": KNeighborsClassifier(n_neighbors=5),
    "Gaussian Naive Bayes": GaussianNB()
}

# Define parameter grids for each model (these are example grids and might need to be updated)
param_grids = {
    "Gradient Boosting Classifier": {'n_estimators': [50, 100, 150], 'max_depth': [3, 4, 5],'learning_rate': [0.05,0.1,0.2],
                                     'random_state':[0]},
    "Random Forest Classifier": {'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20]},
    "KNN Neighbours": {'n_neighbors': [3, 5, 7, 10], 'weights': ['uniform', 'distance']},
}

# Initialize dictionaries to store accuracy scores and training time
accuracy_scores_default = {}
accuracy_scores_tuned = {}
training_times_default = {}
classification_reports_default = {}
classification_reports_tuned = {}

# Define the function for model training and evaluation
def train_and_evaluate_model(model_choice, X_train_data, X_test_data, y_train, y_test):
    # Check if the user's model choice is valid
    if model_choice in available_models:
        model = available_models[model_choice]  # Initialize the model

         # Grid search for the best parameters
        param_grid = param_grids.get(model_choice, {})
        grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
        grid_search.fit(X_train_data, y_train)
        best_model = grid_search.best_estimator_
        print(best_model)

        # Train and evaluate the model with default hyperparameters
        start_time = time.time()
        y_pred_default = model.fit(X_train_data, y_train).predict(X_test_data)
        training_time_default = time.time() - start_time
        accuracy_default = accuracy_score(y_test, y_pred_default)
        accuracy_scores_default[model_choice] = accuracy_default * 100
        training_times_default[model_choice] = training_time_default

        # Tuned model predictions
        y_pred_tuned = best_model.predict(X_test_data)
        accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
        accuracy_scores_tuned[model_choice] = accuracy_tuned * 100
```

Figure 6: Classification Models

2.8 Evaluation

The models are assessed in terms of their predictive accuracy and their ability to accurately classify the various classifications present in the dataset.

Implementation: Using the *accuracy_score* and *classification_report* provided by *sklearn.metrics*, the evaluation is carried out. These functions furnish comprehensive performance metrics and insights regarding the accuracy of the model across multiple classes.

```
        # Visualization of the confusion matrix for the tuned model
        plt.subplot(1, 2, 2)
        sns.heatmap(confusion_matrix(y_test, y_pred_tuned), annot=True, fmt='g')
        plt.title('Confusion Matrix (Tuned Model)')
        plt.show()

        # Classification report for the default model
        print(f"\nClassification Report ({model_choice} - Default Model):")
        print(classification_report(y_test, y_pred_default))
        classification_report_default = classification_report(y_test, y_pred_default, output_dict=True)
        classification_reports_default[model_choice] = classification_report_default
        print(f"Training time ({model_choice} - Default Model): {training_time_default:.4f} seconds")

        # Classification report for the tuned model
        print("\nClassification Report (Tuned Model):")
        print(classification_report(y_test, y_pred_tuned))
        classification_report_tuned = classification_report(y_test, y_pred_tuned, output_dict=True)
        classification_reports_tuned[model_choice] = classification_report_tuned


    else:
        print("Invalid model selection. Please choose from the available models.")
```

Figure 7: Code for evaluation of Models

Evaluation of all the models without Mutual Information Feature Selection

```
# Training all the top models
print("Available models:")
for model_name in available_models.keys():
    print(f"- {model_name}")

# Iterate over each model
for model_name in available_models.keys():
    print(f"{bold_text}\nTraining and evaluating {model_name}...{reset_text}")


    # Call the function with the user's model choice and X_train_scaled for Baseline model evaluation
    train_and_evaluate_model(model_name, X_train_scaled, X_test_scaled, y_train, y_test)
```

Figure 8: Code for evaluation of all the Models without MI feature selection

```
Training and evaluating Gradient Boosting Classifier...
GradientBoostingClassifier(learning_rate=0.2, max_depth=5, n_estimators=150,
                           random_state=0)
```

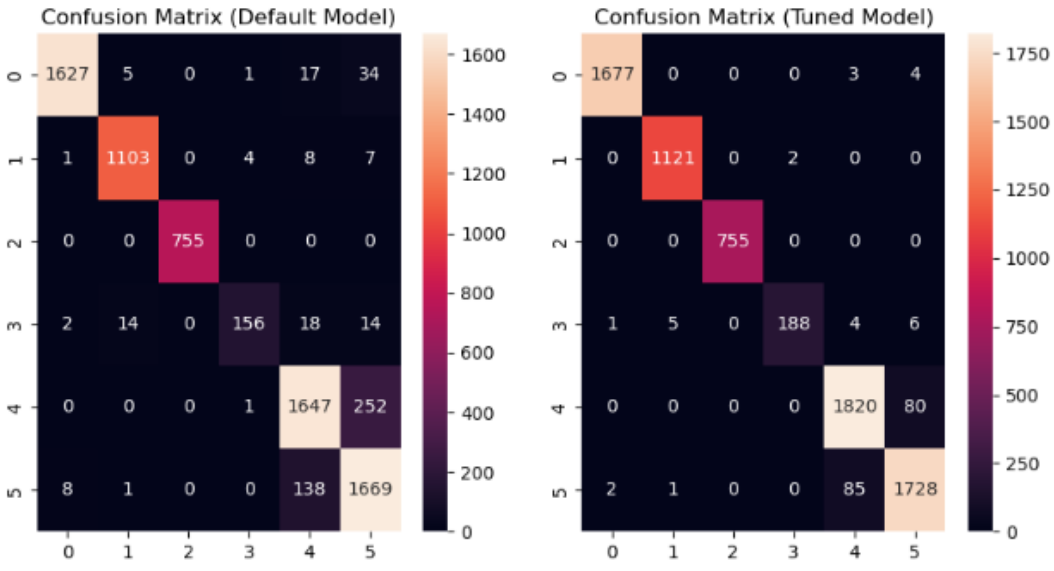

Figure 8: Confusion matrices of Default & Tuned GBM (without MI)

Evaluation of all the models with Mutual Information Feature Selection

```
# Ask the user for the model they want to use
print("Available models:")
for model_name in available_models.keys():
    print(f"- {model_name}")

# Iterate over each model
for model_name in available_models.keys():
    print(f"{bold_text}\nTraining and evaluating {model_name}...{reset_text}")

    # Call the function with the user's model choice and X_train_mi for MI feature selected model evaluation
    train_and_evaluate_model(model_name, X_train_mi, X_test_mi, y_train, y_test)
```

Figure 9: Code for evaluation of all the Models with MI feature selection

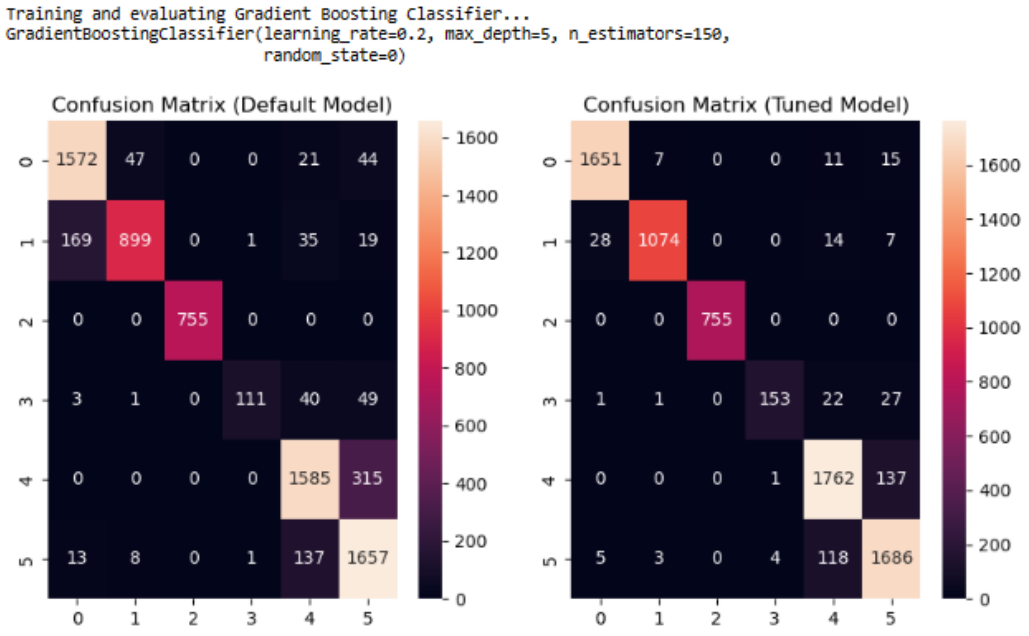

Figure 10: Confusion matrices of Default & Tuned GBM (with MI)

2.9 Perform paired t-tests on the obtained performance metrics.

The code performs paired t-tests to assess the significance of the observed improvements with and without MI feature selection.

```python
import numpy as np
from scipy import stats

# List to store p-values for each classifier
p_values_accuracy = []
combined_accuracy_values_before = []
combined_accuracy_values_after = []

# Iterate over each model
for model_choice in available_models.keys():
    # Get metric values before and after feature selection for the current model
    accuracy_d_before_values = df_metrics_before.loc[model_choice, 'Accuracy_D_Before']
    accuracy_t_before_values = df_metrics_before.loc[model_choice, 'Accuracy_T_Before']
    accuracy_d_after_values = df_metrics_after.loc[model_choice, 'Accuracy_D_After']
    accuracy_t_after_values = df_metrics_after.loc[model_choice, 'Accuracy_T_After']

    # Combine accuracy values before and after into single arrays
    combined_accuracy_values_before.extend([accuracy_d_before_values, accuracy_t_before_values])
    combined_accuracy_values_after.extend([accuracy_d_after_values, accuracy_t_after_values])

# Perform paired t-test
t_statistic, p_value = stats.ttest_rel(combined_accuracy_values_before, combined_accuracy_values_after)

# Calculate Cohen's d
mean_diff = np.mean(combined_accuracy_values_before) - np.mean(combined_accuracy_values_after)
std_before = np.std(combined_accuracy_values_before, ddof=1)
std_after = np.std(combined_accuracy_values_after, ddof=1)
pooled_std = np.sqrt(((len(combined_accuracy_values_before) - 1) * std_before**2 + (len(combined_accuracy_values_after) - 1) *
                      std_after**2) / (len(combined_accuracy_values_before) + len(combined_accuracy_values_after) - 2))
cohens_d = mean_diff / pooled_std

# Print results for the paired t-test and Cohen's d
print(f"T-Statistic: {t_statistic}")
print(f"P-Value: {p_value}")
print(f"Cohen's d: {cohens_d}")

# Perform adjustments for multiple comparisons if needed (e.g., Bonferroni correction)
# For example:
# alpha_adjusted = 0.05 / len(available_models)

# Print adjusted p-values
print(f"\nAdjusted p-value for accuracy: {p_value}")  # Adjusted p-value
```

Figure 11: Code for paired t-test

2.10 Comparison

The code compares four different ML classification methods to the MIGBM model and shows the evaluation metrics along with the comparison.

Implementation 1: This code iterates through different models and prints their corresponding accuracy scores. It compares the default accuracy scores with the tuned (possibly optimized or adjusted) accuracy scores for each model, providing a side-by-side comparison for evaluation purposes including other performance metrics also.

```python
# Extract Precision, Recall, F1-score, Support, and accuracy from classification reports
precision_recall_f1_support_accuracy = {}

from scipy import stats

# Iterate over each model
for model_choice in available_models.keys():
    report_default = classification_reports_default[model_choice]
    report_tuned = classification_reports_tuned[model_choice]
    training_time_default = training_times_default[model_choice]
    accuracy_after_default = accuracy_scores_default[model_choice]/100
    accuracy_after_tuned = accuracy_scores_tuned[model_choice]/100

    precision_recall_f1_support_accuracy[model_choice] = {
        'Precision': report_default['weighted avg']['precision'],
        'Recall': report_default['weighted avg']['recall'],
        'F1-score': report_default['weighted avg']['f1-score'],
        'Support': report_default['weighted avg']['support'],
        'Accuracy Default (%)': accuracy_scores_default[model_choice],
        'Accuracy Tuned (%)': accuracy_scores_tuned[model_choice],
        'Accuracy_D_After': accuracy_after_default,
        'Accuracy_T_After': accuracy_after_tuned,
        'Training time (Default in Seconds)': training_times_default[model_choice]
    }


# Convert to DataFrame for better display
import pandas as pd
df_metrics_after = pd.DataFrame(precision_recall_f1_support_accuracy).T
print(df_metrics_after)
```

Figure 9: Code for displaying all model's performance metrics.

```
                            Precision    Recall  F1-score  Support  \
Gradient Boosting Classifier  0.931952  0.929832  0.930075   7482.0
Random Forest Classifier      0.948190  0.947207  0.947213   7482.0
KNN Neighbours                0.867762  0.866613  0.866381   7482.0
Gaussian Naive Bayes          0.733904  0.676958  0.673253   7482.0

                            Accuracy Default (%)  Accuracy Tuned (%)  \
Gradient Boosting Classifier             92.983160           97.420476
Random Forest Classifier                 94.720663           94.627105
KNN Neighbours                           86.661321           88.305266
Gaussian Naive Bayes                     67.695803           67.695803

                            Accuracy_D_Before  Accuracy_T_Before  \
Gradient Boosting Classifier          0.929832           0.974205
Random Forest Classifier              0.947207           0.946271
KNN Neighbours                        0.866613           0.883053
Gaussian Naive Bayes                  0.676958           0.676958

                            Training time (Default in Seconds)
Gradient Boosting Classifier                         37.368581
Random Forest Classifier                              3.816516
KNN Neighbours                                        4.082073
Gaussian Naive Bayes                                  0.047805
```

Figure 10: Performance evaluation results (without MI)

```
                                Precision    Recall  F1-score  Support  \
Gradient Boosting Classifier     0.884740  0.879310  0.878280   7482.0
Random Forest Classifier         0.933985  0.933039  0.932679   7482.0
KNN Neighbours                   0.821020  0.823978  0.821564   7482.0
Gaussian Naive Bayes             0.689965  0.621492  0.618380   7482.0

                                Accuracy Default (%)  Accuracy Tuned (%)
Gradient Boosting Classifier               87.931034           94.640470
Random Forest Classifier                   93.303929           93.330660
KNN Neighbours                             82.397755           86.647955
Gaussian Naive Bayes                       62.149158           62.149158

                                Accuracy_D_After  Accuracy_T_After  \
Gradient Boosting Classifier            0.879310          0.946405
Random Forest Classifier                0.933039          0.933307
KNN Neighbours                          0.823978          0.866480
Gaussian Naive Bayes                    0.621492          0.621492

                                Training time (Default in Seconds)
Gradient Boosting Classifier                             20.339331
Random Forest Classifier                                  3.619737
KNN Neighbours                                            0.171044
Gaussian Naive Bayes                                      0.013519
```

Figure 10: Performance evaluation results (with MI)

Implementation 2: This code generates a comparative bar plot displaying the accuracy scores of different machine-learning models.

```python
import matplotlib.pyplot as plt

# Assume accuracy_scores_default and accuracy_scores_tuned are dictionaries with model names as keys and accuracy scores

fig, ax = plt.subplots(figsize=(9, 4))
bar_width = 0.35  # Width of the bars

# Plotting the bars for Default Model
default_bars = ax.bar([x - bar_width/2 for x in range(len(accuracy_scores_default))],
                      accuracy_scores_default.values(),
                      width=bar_width,
                      label='Default Model')

# Plotting the bars for Tuned Model
tuned_bars = ax.bar([x + bar_width/2 for x in range(len(accuracy_scores_tuned))],
                    accuracy_scores_tuned.values(),
                    width=bar_width,
                    label='Tuned Model')

# Setting x-axis ticks and labels
ax.set_xticks(range(len(accuracy_scores_default)))
ax.set_xticklabels(accuracy_scores_default.keys(), rotation=0, ha='right')  # Rotating labels for better readability

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy of Different ML Models')
ax.legend()

# Function to add value labels within the bars
def autolabel(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 1),  # 3 points vertical offset for better readability
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(default_bars)
autolabel(tuned_bars)

plt.tight_layout()  # Adjusts layout to prevent clipping of labels
plt.show()
```

Figure 10: Comparison using bar plot.

2.11 Additional Functionality

This code provides additional functionality to evaluate any model and feature selection combination of your choice for faster results and evaluation.

Evaluation of model and Feature Selection Technique of your choice

```
while True:
    print("Available models:")
    for model_name in available_models.keys():
        print(f"- {model_name}")

    model_choice = input("Please select a model for evaluation: ")
    print()

    selection_techniques = {"Mutual Information Feature Selection", "No Feature Selection"}
    print("Selection Techniques:")
    for selection_technique in selection_techniques:
        print(f"- {selection_technique}")

    selection_technique_choice = input("Please select a selection technique: ")
    print()

    # Call the function with the user's model choice and Feature Selection Technique
    if selection_technique_choice == "Mutual Information Feature Selection":
        print(f"\n{bold_text}Training and evaluating {model_choice} with {selection_technique_choice}...{reset_text}")
        train_and_evaluate_model(model_choice, X_train_mi, X_test_mi, y_train, y_test)
    elif selection_technique_choice == "No Feature Selection":
        print(f"\n{bold_text}Training and evaluating {model_choice} with {selection_technique_choice}...{reset_text}")
        train_and_evaluate_model(model_choice, X_train_scaled, X_test_scaled, y_train, y_test)
    else:
        print("Invalid Selection Technique. Please try again.\n")
        continue
    print("=========================================================================================================")
    rerun = input("\nDo you want to rerun the code? (yes/no): ")
    if rerun.lower() != "yes":
        break
print(f"\n{bold_text}THANK YOU FOR YOUR TIME{reset_text}")
```

```
Available models:
- Gradient Boosting Classifier
- Random Forest Classifier
- KNN Neighbours
- Gaussian Naive Bayes

Please select a model for evaluation:
```

Figure 11: Additional code to run the model combination of your choice.

References

Python release python 3.9.13 (2023) Python.org. Available at: https://www.python.org/downloads/release/python-3913/ (Accessed: 01 December 2023).

Installing on windows# (2023) Installing on Windows - Anaconda documentation. Available at: https://docs.anaconda.com/free/anaconda/install/windows/ (Accessed: 01 December 2023).

Python - P-value (no date) *Online Tutorials, Courses, and eBooks Library*. Available at: https://www.tutorialspoint.com/python_data_science/python_p_value.htm (Accessed: 06 December 2023).