**Final Project: Report**

By

Ashywar Anand Bhardwaj – 100946247  
Shivkumar Trivedi – 100949871  
Harsh Yogeshbhai Dave – 100945546  
Virendersingh Jaibirsingh Pachar – 100945547

Data Analytics for Business Decision Making, Durham College

DATA1202-03 – Data Analysis Tools Analytics

Sk. Md. Mizanur Rahman  
18-Apr-2024

Table of Contents

[Introduction 3](#_Toc164375639)

[Project Objectives 3](#_Toc164375640)

[Methodology 3](#_Toc164375641)

[Data acquisition and processing 4](#_Toc164375642)

[Exploratory data analysis(EDA) 4](#_Toc164375643)

[Data Summary 4](#_Toc164375644)

[Statistical Summary of the Cybersecurity Dataset 5](#_Toc164375645)

[Data Integrity Summary 5](#_Toc164375646)

[Visualization 6](#_Toc164375647)

[Histograms: 6](#_Toc164375648)

[Correlation Heatmap 9](#_Toc164375649)

[Spliting The Dataset 10](#_Toc164375650)

[Machine Learning Classifiers 10](#_Toc164375651)

[Performance Evaluation 10](#_Toc164375652)

[Confusion Matrix 11](#_Toc164375653)

[Appendix (Python Script) 13](#_Toc164375654)

[1. Data Description 13](#_Toc164375655)

[2. Exploratory data analysis(EDA) 13](#_Toc164375656)

[2.1 Data Summary and Statistical Summary 13](#_Toc164375657)

[2.2 Data Integrity Summary 13](#_Toc164375658)

[2.3 Histograms 14](#_Toc164375659)

[2.4 Correlation Heatmap 15](#_Toc164375660)

[3. Load and Split the Dataset 15](#_Toc164375661)

[4. Initialisation of Machine Learning Classifiers 16](#_Toc164375662)

[5. Training the Machine Learning Classifiers 16](#_Toc164375663)

[6. Testing the Machine Learning Classifiers 16](#_Toc164375664)

[7. Results of the Machine Learning Classifiers Testing 17](#_Toc164375665)

[8.Generate the Confusion Matrix 18](#_Toc164375666)

# Introduction

Data Analytics and Machine Learning are two key fields that have potential applications for solving emerging cybersecurity challenges, owing to their capacity to recognize patterns and trends from the available data sources. This project focuses on the development and deployment of three separate machine learning classifiers on a cybersecurity dataset. Subsequently, the classifiers are tested on the separation diagnostic test and evaluated on how effectively they distinguished the malicious instances from the benign ones. Ultimately, the classifiers obtained various perception gaps, which offer a dynamic understanding of machine learning with cybersecurity capabilities. This paper outlines the construction, results, and critical interpretation of these classifiers and presents them in terms of actualized performance and avenues for performances.

# Project Objectives

The project’s main objective is the development of a simple machine learning system capable of processing an available cybersecurity dataset to reveal insights that can be acted upon. Specifically, the objectives are:

* **Prepare a dataset:** Get a dataset and preprocess it so that the input data is ready.
* **Model development:** Develop three distinct machine learning classifiers.
* **Model evaluation:** Apply testing procedures, evaluate the classifiers, and explain the outcome.

# Methodology

The methodology comprises the following steps:

1. **Data acquisition and processing:** Downloading the dataset and processing it to ensure that the portion of each class (e.g., malware and benign) is balanced.
2. **Exploratory data analysis :** An in-depth analysis to understand the data’s distribution and characteristics.
3. **Model training :** Development of three independent machine learning models and training them.
4. **Testing and validation:** Evaluating their performance and generating confusion matrices .
5. **Result discussion:** Finally, analyze the result and discuss how they perform by identifying their level of performance and possible drawbacks or how they can be useful in the real cybersecurity context.

# Data acquisition and processing

Curated data in this project are specifically for cybersecurity applications and intended for malware and benign software processes. It gives a balanced view over probably occurred cybersecurity threats, analyzed, and classified with thoroughness.

**Dataset Characteristics:**

* Total Instances: 100,000
* Number of features: 33 features.
* Class Distribution:
  + Malware Instances: 50,000
  + Benign Instances: 50,000

The features include both numerical and categorical data types, which describe the different types of characteristics pertaining to the software process: process behavior, network activity, and file signatures. The balanced distribution between malware and benign instances ensures that the models, if learned effectively over this training set, learn not to predict a class for the problems, but rather generalize and predict accurately on new data.

# Exploratory data analysis(EDA)

## Data Summary

Data Shape: 100,000 rows × 35 columns

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **Column Name** | **Non-Null Count** | **Data Type** |  | **Index** | **Column Name** | **Non-Null Count** | **Data Type** |
| 0 | hash | 100,000 | object |  | 18 | shared\_vm | 100,000 | int64 |
| 1 | millisecond | 100,000 | int64 |  | 19 | exec\_vm | 100,000 | int64 |
| 2 | classification | 100,000 | object |  | 20 | reserved\_vm | 100,000 | int64 |
| 3 | state | 100,000 | int64 |  | 21 | nr\_ptes | 100,000 | int64 |
| 4 | usage\_counter | 100,000 | int64 |  | 22 | end\_data | 100,000 | int64 |
| 5 | prio | 100,000 | int64 |  | 23 | last\_interval | 100,000 | int64 |
| 6 | static\_prio | 100,000 | int64 |  | 24 | nvcsw | 100,000 | int64 |
| 7 | normal\_prio | 100,000 | int64 |  | 25 | nivcsw | 100,000 | int64 |
| 8 | policy | 100,000 | int64 |  | 26 | min\_flt | 100,000 | int64 |
| 9 | vm\_pgoff | 100,000 | int64 |  | 27 | maj\_flt | 100,000 | int64 |
| 10 | vm\_truncate\_count | 100,000 | int64 |  | 28 | fs\_excl\_counter | 100,000 | int64 |
| 11 | task\_size | 100,000 | int64 |  | 29 | lock | 100,000 | int64 |
| 12 | cached\_hole\_size | 100,000 | int64 |  | 30 | utime | 100,000 | int64 |
| 13 | free\_area\_cache | 100,000 | int64 |  | 31 | stime | 100,000 | int64 |
| 14 | mm\_users | 100,000 | int64 |  | 32 | gtime | 100,000 | int64 |
| 15 | map\_count | 100,000 | int64 |  | 33 | cgtime | 100,000 | int64 |
| 16 | hiwater\_rss | 100,000 | int64 |  | 34 | signal\_nvcsw | 100,000 | int64 |
| 17 | total\_vm | 100,000 | int64 |  |  |  |  |  |

* The dataset is structured in a data frame, with each column uniformly having 100,000 non-null entries, indicating no missing values across all features.
* The data types are predominantly integer (int64), with exceptions for hash and classification which are object types, typically strings

## Statistical Summary of the Cybersecurity Dataset

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ature** | **Count** | **Mean** | **Std** | **Min** | **Max** |
| **millisecond** | 100,000 | 499.5 | 288.68 | 0 | 999 |
| **state** | 100,000 | 157,768.30 | 936,172.60 | 0 | 43,266,050 |
| **usage\_counter** | 100,000 | 0 | 0 | 0 | 0 |
| **prio** | 100,000 | 3,069,706,000 | 296,306 | 3,069,190,000 | 3,070,222,000 |
| **static\_prio** | 100,000 | 18,183.90 | 4,609.79 | 13,988 | 31,855 |
| **normal\_prio** | 100,000 | 0 | 0 | 0 | 0 |
| **policy** | 100,000 | 0 | 0 | 0 | 0 |
| **vm\_pgoff** | 100,000 | 0 | 0 | 0 | 0 |
| **vm\_truncate\_count** | 100,000 | 15,312.74 | 3,256.48 | 9,695 | 27,157 |
| **task\_size** | 100,000 | 0 | 0 | 0 | 0 |
| **nivcsw** | 100,000 | 32.99 | 52.73 | 0 | 365 |
| **min\_flt** | 100,000 | 2.05 | 13.88 | 0 | 256 |
| **maj\_flt** | 100,000 | 117.92 | 3.12 | 112 | 120 |
| **fs\_excl\_counter** | 100,000 | 1.11 | 2.16 | 0 | 18 |
| **utime** | 100,000 | 385,415.45 | 10,144.04 | 371,782 | 421,913 |
| **stime** | 100,000 | 4.06 | 0.82 | 3 | 7 |
| **gtime** | 100,000 | 1.66 | 3.26 | 0 | 15 |

* Features such as **usage\_counter**, **normal\_prio**, **policy**, **vm\_pgoff**, and **task\_size** exhibit no variability (constant values), indicating they may not contribute to model differentiation in this dataset.
* The table provides a broad overview of the range and distribution of each feature, which is critical for the training and evaluation of effective machine learning classifiers in cybersecurity.

## Data Integrity Summary

**Missing Values:** The dataset contains no missing values across all 35 features. This means that there is completeness in the data, whereby a very high level of completeness is realized in all the 35 features contained in the data.

**Duplicate Entries:** There are no duplicated entries in the dataset. This ensures that the entries made are unique in nature, which may further point towards increased reliability of carrying out data analysis or machine learning applications that might follow.

## **Visualization**

We perform several visualizations on the dataset to understand the same. We have divided these based on the type of visualization being done:

### Histograms:

1. The histogram shows an even distribution between two classes, 'malware' and 'benign', indicating a balanced dataset

The histogram shows an even distribution between two classes, 'malware' and 'benign', indicating a balanced dataset.








Figure 1

1. The histograms compare the distributions of the 'prio', 'maj\_flt', and 'min\_flt' features between the 'malware' and 'benign' classes. It shows that 'prio' has overlapping but clear and distinct distributions between both classes, 'maj\_flt' has a maximum concentration around one range of values for the 'benign' class, and 'min\_flt' registers most of its values towards the lower range for the 'malware' class.

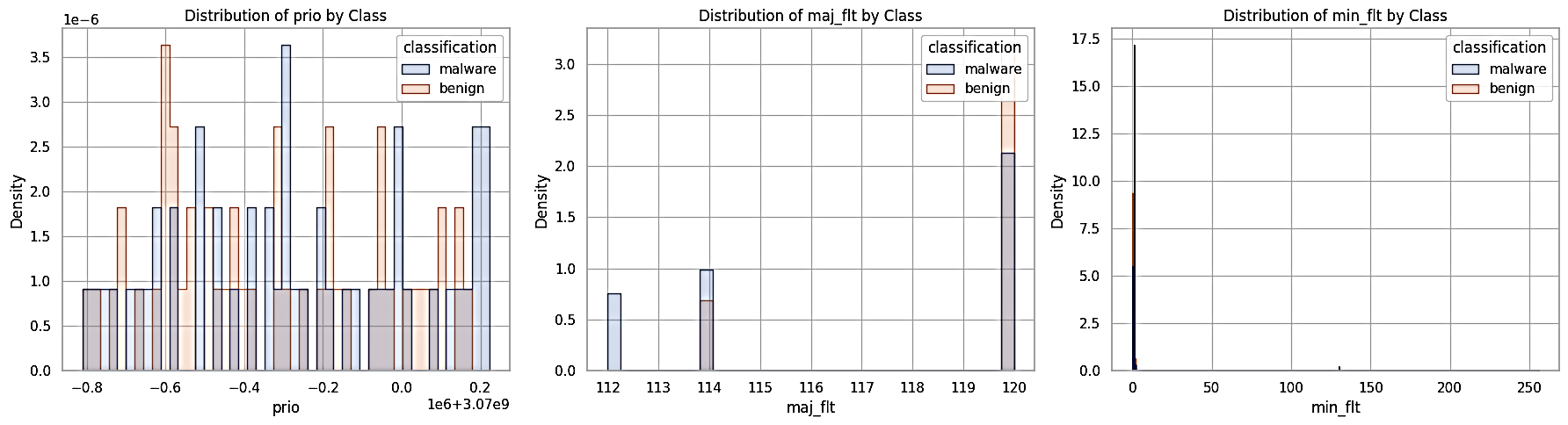


Figure 2

1. The histograms provide a graphical distribution of the different features within the cybersecurity data. These graphical features will help establish the frequency point for the data of each feature, hence giving the respective distribution and any possible skewness or outliers.

**Key Observations:**

* + The feature columns like `millisecond`, `state`, `vm\_truncate\_count`, `free\_area\_cache`, `map\_count`, and few others tend to have a skewed distribution with values in a particular range.
  + Many histograms, such as json data., have a bimodal distribution (`nvcsw`, `nivcsw`, `lock`, `utime`, `stime`), hinting at the presence of two dominant groups within these features.
  + `usage\_counter`, `normal\_prio`, `policy`, `vm\_pgoff`, `task\_size`, and much more show only a few values distributed narrowly among each other. Most likely, this is showing the low variance of that particular feature; on the other hand, most of the instances of that cluster have values almost similar.
  + `maj\_flt` is densely distributed at specific intervals, most likely indicating categorical or binned numerical data.

This could introduce a performance impact on machine learning algorithms and data transformation or normalization requirements since the distribution shapes may vary from uniform to highly skewed. These histograms are important for exploratory data analysis, providing a fast and effective way of understanding the data's inherent structure and informing decisions on preprocessing.

A graph of a graph

Description automatically generated with medium confidence

Figure 3

A graph of a graph

Description automatically generated with medium confidence

Figure 4:

### Correlation Heatmap

The heatmap in this figure plots the correlation matrix of features in the cybersecurity dataset. It expresses the linear relationship between each pair of variables. Strong positive correlations (values close to 1, in dark red) mean that two attributes are increasing together, whereas strong negative (close to -1, in dark blue) indicate a rising-to-falling relationship. The intensity of color varies with the degree of correlation.

From this heatmap, we can discern several key points:

* Vm\_truncate\_count, for example, has a huge positive correlation with themselves. Well, this is expected since a diagonal implies the correlation of a variable with itself.
* Some feature pairs show marked positive correlation, though it is not too familiar; most features have low to moderate correlation to one another.
* The lack of high, robust correlations suggests that, in turn, there is less redundancy of a high nature between features

A screenshot of a computer screen

Description automatically generated

Figure 5

# Spliting The Dataset

The dataset is split into a Training set and a Test set, with 80% going as the training dataset and 20% going as the test dataset.

# Machine Learning Classifiers

We are using the machine learning classifiers mentioned below for the Cyversecurity dataset.

1. Logistic Regression:

Suitable for binary classification problems. Assumes a linear relationship between the features and the log odds of the target variable.Outputs probabilities of belonging to each class using the logistic function. Interpretable coefficients indicate the influence of each feature on the prediction.

1. Decision Tree:  
   A supervised learning algorithm was used for classification and regression tasks. Makes decisions by recursively splitting the data based on feature values. Creates a tree structure where every internal node represents a feature, and every leaf node represents a class label (in classification).
2. Random Forest:

The ensemble learning method is based on decision trees. Constructs, during training, several decision trees and outputs the mode (classification) or average (regression) prediction of the individual trees. Reduces overfitting compared to single decision trees by combining predictions from multiple models.

## Performance Evaluation

Each classifier was tested using the same testing set to ensure a fair comparison. The following metrics were used to evaluate each model:

* **Accuracy**: Measures the overall correctness of the model across all predictions.
* **Precision**: Indicates the proportion of positive identifications that were actually correct.
* **Recall** (Sensitivity): Measures the proportion of actual positives that were correctly identified.
* **F1-Score**: Weighted average of Precision and Recall. Useful in cases of class imbalance.

The performance results for each classifier are summarized in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **Logistic Regression** | 1 | 25.00% | 50.00% | 33.33% |
| **Decision Tree** | 1 | 1.00 | 1.00 | 100.00% |
| **Random Forest** | 1 | 100.00% | 100.00% | 100.00% |

The Decision Tree and Random Forest classifiers both scored perfectly in all metrics, meaning that all instances were classified correctly in the testing set. The Logistic Regression model, on the other hand, had significantly low performances, meaning it could not have contributed well to this particular task. It is worth noting that while this shows 100% accuracy, pointing towards excellent performance, it may also be an indication of overfitting. Therefore, these results further need validation before claiming them to be generalizable for unseen data.

Top of Form

## Confusion Matrix

A blue squares with white text

Description automatically generated

The image displays three confusion matrices: one of the Logistic Regression, the Decision Tree, and the Random Forest classifiers. It's used to measure classifications' performance, indicating the actual vs. predicted classifications.

Here is an analysis of each:

1. Logistic Regression Confusion Matrix:
   1. True Positives (TP): 0 (The model did not correctly predict any malware instances.)
   2. True Negatives (TN): 0 (The model did not correctly predict any benign instances.)
   3. False Positives (FP): 0 (The model did not incorrectly predict malware when it was actually benign.)
   4. False Negatives (FN): 10,000 (The model incorrectly labeled all malware instances as benign.)

This model did not do very well at all. It missed all the malware predictions, bringing out more interesting questions around the 50% recall: it missed all actual positive (malware) cases.

1. Decision Tree Confusion Matrix:
   1. TP: 10,000 (The model correctly predicted all malware instances.)
   2. TN: 10,000 (The model correctly predicted all benign instances.)
   3. FP: 0 (The model made no errors predicting benign instances as malware.)
   4. FN: 0 (The model made no errors predicting malware instances as benign.)

This model appears to have achieved perfect classification with no misclassifications.

1. Random Forest Confusion Matrix:
   1. TP: 10,000 (The model correctly predicted all malware instances.)
   2. TN: 10,000 (The model correctly predicted all benign instances.)
   3. FP: 0 (The model made no errors predicting benign instances as malware.
   4. FN: 0 (The model made no errors predicting malware instances as benign.)

Similar to the Decision Tree, the Random Forest also appears to have achieved perfect classification.

The Decision Tree and Random Forest confusion matrices suggest exceptionally high performance. However, in practical scenarios, perfect results can indicate overfitting, especially if the testing data are not representative or if there was some leak of information at some point regarding that data during model training. These results must be taken cautiously, and further research is advised on model validation so that the model generalizes well with new, unseen data.

# Appendix (Python Script)

## **1. Data Description**

import pandas as pd

# Load the dataset

data = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/dataset.csv')

# Number of instances (rows)

num\_instances = data.shape[0]

# Number of features (excluding the identifier 'hash' and the class label 'classification')

num\_features = data.shape[1] - 2

# Number of instances from each class

class\_distribution = data['classification'].value\_counts()

# Display the results

print(f"Number of Instances: {num\_instances}")

print(f"Number of Features: {num\_features}")

print("Class Distribution:")

print(class\_distribution)

## **2. Exploratory data analysis(EDA)**

### **2.1 Data Summary and Statistical Summary**

import pandas as pd

def data\_overview(dataset\_path):

dataset = pd.read\_csv(dataset\_path)

print("Data Shape:", dataset.shape)

print("\nData Info:")

print(dataset.info())

print("\nData Description:")

print(dataset.describe(include='all'))

# Example usage:

data\_overview('/content/drive/MyDrive/Colab Notebooks/dataset.csv')

### **2.2 Data Integrity Summary**

def data\_cleaning(dataset\_path):

dataset = pd.read\_csv(dataset\_path)

initial\_shape = dataset.shape

dataset.drop\_duplicates(inplace=True)

dataset.dropna(inplace=True) # Remove rows with any missing data

cleaned\_shape = dataset.shape

dataset.to\_csv('cleaned\_dataset.csv', index=False)

return initial\_shape, cleaned\_shape

# Example usage:

initial\_shape, cleaned\_shape = data\_cleaning('/content/drive/MyDrive/Colab Notebooks/dataset.csv')

print(f"Initial dataset shape: {initial\_shape}, Cleaned dataset shape: {cleaned\_shape}")

### **2.3 Histograms**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

# Suppress all warnings

warnings.filterwarnings('ignore')

def perform\_detailed\_eda(dataset\_path):

# Load the dataset

data = pd.read\_csv(dataset\_path)

# Set the aesthetics for the plots

sns.set(style="whitegrid")

# Plot for class distribution

plt.figure(figsize=(8, 5))

sns.countplot(x='classification', data=data)

plt.title('Distribution of Classes')

plt.xlabel('Classification')

plt.ylabel('Frequency')

plt.show()

# Selecting key numerical features for detailed histograms

features\_to\_plot = ['prio', 'maj\_flt', 'min\_flt']

# Creating histograms for selected features

fig, axes = plt.subplots(nrows=1, ncols=len(features\_to\_plot), figsize=(18, 5))

for i, feature in enumerate(features\_to\_plot):

sns.histplot(data=data, x=feature, hue='classification', element='step', stat='density', common\_norm=False, ax=axes[i])

axes[i].set\_title(f'Distribution of {feature} by Class')

axes[i].set\_xlabel(f'{feature}')

axes[i].set\_ylabel('Density')

plt.tight\_layout()

plt.show()

# Plotting histograms for all numerical columns with considerations for the number of features

num\_cols = data.select\_dtypes(include=['number']).columns

num\_rows = (len(num\_cols) - 1) // 3 + 1

plt.figure(figsize=(15, 5 \* num\_rows))

for idx, col in enumerate(num\_cols):

plt.subplot(num\_rows, 3, idx + 1)

sns.histplot(data[col], kde=True, bins=15, color='skyblue')

plt.title(f'Histogram of {col}')

plt.tight\_layout()

plt.show()

# Example usage:

perform\_detailed\_eda('/content/drive/MyDrive/Colab Notebooks/dataset.csv')

### **2.4 Correlation Heatmap**

import pandas as pd

import numpy as np # Add this line to import NumPy

import seaborn as sns

import matplotlib.pyplot as plt

def plot\_correlation\_heatmap(dataset\_path):

dataset = pd.read\_csv(dataset\_path)

# Selecting only numeric columns for correlation analysis

numeric\_dataset = dataset.select\_dtypes(include=[np.number])

plt.figure(figsize=(12, 10))

sns.heatmap(numeric\_dataset.corr(), annot=True, cmap='coolwarm')

plt.show()

# Example usage:

plot\_correlation\_heatmap('/content/drive/MyDrive/Colab Notebooks/dataset.csv')

## 3. **Load and Split the Dataset**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Load the dataset

data\_path = '/content/drive/MyDrive/Colab Notebooks/dataset.csv'

dataset = pd.read\_csv(data\_path)

# Split the dataset into features and target variable

X = dataset.drop(columns=['classification'])

y = dataset['classification']

# Convert categorical columns to numerical using one-hot encoding if necessary

X = pd.get\_dummies(X, drop\_first=True)

# Split the data into training and testing sets, ensuring a balanced split with respect to the target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Print the confirmation and details of the dataset split

print("Dataset split into training and testing sets successfully.")

print(f"Training data shape: {X\_train.shape} (features), {y\_train.shape} (target)")

print(f"Testing data shape: {X\_test.shape} (features), {y\_test.shape} (target)")

## 4. **Initialisation of Machine Learning Classifiers**

rom sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

# Initialize the classifiers

logistic\_regression\_classifier = LogisticRegression(max\_iter=1000, random\_state=42)

print("Logistic Regression classifier has been initialized.")

decision\_tree\_classifier = DecisionTreeClassifier(random\_state=42)

print("Decision Tree classifier has been initialized.")

random\_forest\_classifier = RandomForestClassifier(random\_state=42)

print("Random Forest classifier has been initialized.")

## 5. **Training the Machine Learning Classifiers**

import time

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

# Initialize the classifiers

logistic\_regression\_classifier = LogisticRegression(max\_iter=1000, random\_state=42)

decision\_tree\_classifier = DecisionTreeClassifier(random\_state=42)

random\_forest\_classifier = RandomForestClassifier(random\_state=42)

# Train Logistic Regression

start\_time = time.time()

logistic\_regression\_classifier.fit(X\_train, y\_train)

elapsed\_time = time.time() - start\_time

print(f"Logistic Regression training completed in {elapsed\_time:.2f} seconds.")

# Train Decision Tree Classifier

start\_time = time.time()

decision\_tree\_classifier.fit(X\_train, y\_train)

elapsed\_time = time.time() - start\_time

print(f"Decision Tree training completed in {elapsed\_time:.2f} seconds.")

# Train Random Forest Classifier

start\_time = time.time()

random\_forest\_classifier.fit(X\_train, y\_train)

elapsed\_time = time.time() - start\_time

print(f"Random Forest training completed in {elapsed\_time:.2f} seconds.")

## 6. **Testing the Machine Learning Classifiers**

import time

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

# Assuming classifiers are already initialized and trained from previous examples

# Test Logistic Regression

start\_time = time.time()

y\_pred\_logistic = logistic\_regression\_classifier.predict(X\_test)

elapsed\_time = time.time() - start\_time

print(f"Testing with Logistic Regression completed in {elapsed\_time:.2f} seconds.")

# Test Decision Tree Classifier

start\_time = time.time()

y\_pred\_decision\_tree = decision\_tree\_classifier.predict(X\_test)

elapsed\_time = time.time() - start\_time

print(f"Testing with Decision Tree completed in {elapsed\_time:.2f} seconds.")

# Test Random Forest Classifier

start\_time = time.time()

y\_pred\_random\_forest = random\_forest\_classifier.predict(X\_test)

elapsed\_time = time.time() - start\_time

print(f"Testing with Random Forest completed in {elapsed\_time:.2f} seconds.")

## 7. **Results of the Machine Learning Classifiers Testing**

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import pandas as pd

# Adjusting the precision\_score call to handle zero division cases

precision\_logistic = precision\_score(y\_test, y\_pred\_logistic, average='weighted', zero\_division=0)

precision\_decision\_tree = precision\_score(y\_test, y\_pred\_decision\_tree, average='weighted', zero\_division=0)

precision\_random\_forest = precision\_score(y\_test, y\_pred\_random\_forest, average='weighted', zero\_division=0)

# Calculate metrics for Logistic Regression

accuracy\_logistic = accuracy\_score(y\_test, y\_pred\_logistic)

recall\_logistic = recall\_score(y\_test, y\_pred\_logistic, average='weighted')

f1\_logistic = f1\_score(y\_test, y\_pred\_logistic, average='weighted')

# Calculate metrics for Decision Tree

accuracy\_decision\_tree = accuracy\_score(y\_test, y\_pred\_decision\_tree)

recall\_decision\_tree = recall\_score(y\_test, y\_pred\_decision\_tree, average='weighted')

f1\_decision\_tree = f1\_score(y\_test, y\_pred\_decision\_tree, average='weighted')

# Calculate metrics for Random Forest

accuracy\_random\_forest = accuracy\_score(y\_test, y\_pred\_random\_forest)

recall\_random\_forest = recall\_score(y\_test, y\_pred\_random\_forest, average='weighted')

f1\_random\_forest = f1\_score(y\_test, y\_pred\_random\_forest, average='weighted')

# Create a dictionary of metrics for each classifier

data = {

'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],

'Logistic Regression': [

f"{accuracy\_logistic:.2%}",

f"{precision\_logistic:.2%}",

f"{recall\_logistic:.2%}",

f"{f1\_logistic:.2%}"

],

'Decision Tree': [

f"{accuracy\_decision\_tree:.2%}",

f"{precision\_decision\_tree:.2%}",

f"{recall\_decision\_tree:.2%}",

f"{f1\_decision\_tree:.2%}"

],

'Random Forest': [

f"{accuracy\_random\_forest:.2%}",

f"{precision\_random\_forest:.2%}",

f"{recall\_random\_forest:.2%}",

f"{f1\_random\_forest:.2%}"

]

}

# Convert the dictionary to a DataFrame

results\_df = pd.DataFrame(data)

# Set 'Metric' as the index for better readability

results\_df.set\_index('Metric', inplace=True)

# Print the DataFrame

print("Model Performance Metrics:")

print(results\_df)

## **8.Generate the Confusion Matrix**

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

# Generate confusion matrix for Logistic Regression

cm\_logistic = confusion\_matrix(y\_test, y\_pred\_logistic)

# Generate confusion matrix for Decision Tree

cm\_decision\_tree = confusion\_matrix(y\_test, y\_pred\_decision\_tree)

# Generate confusion matrix for Random Forest

cm\_random\_forest = confusion\_matrix(y\_test, y\_pred\_random\_forest)

# Plotting confusion matrices

fig, ax = plt.subplots(1, 3, figsize=(18, 6))

ConfusionMatrixDisplay(cm\_logistic, display\_labels=logistic\_regression\_classifier.classes\_).plot(ax=ax[0], cmap='Blues')

ax[0].set\_title('Logistic Regression Confusion Matrix')

ConfusionMatrixDisplay(cm\_decision\_tree, display\_labels=decision\_tree\_classifier.classes\_).plot(ax=ax[1], cmap='Blues')

ax[1].set\_title('Decision Tree Confusion Matrix')

ConfusionMatrixDisplay(cm\_random\_forest, display\_labels=random\_forest\_classifier.classes\_).plot(ax=ax[2], cmap='Blues')

ax[2].set\_title('Random Forest Confusion Matrix')

plt.tight\_layout()

plt.show()