Configuration Manual

M.Sc. Research Project
M.Sc. in Cybersecurity

Srikari Surampudi
Student ID: 23178485

School of Computing
National College of Ireland

Supervisor: Dr. Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Srikari Surampudi
Student ID:	23178485
Programme:	M.Sc. in Cybersecurity **Year:** 2024-25
Module:	M.Sc. Research Project (Practicum Part2)
Lecturer:	Dr. Imran Khan
Submission Due Date:	12-12-2024
Project Title:	A Comparative Study of ML Models for Data Loss Prevention
Word Count:	931 **Page Count:** 12

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: Srikari Surampudi

Date: 11-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Srikari Surampudi
Student ID: 23178485

Introduction

This thesis aims to compare various machine learning models in cybersecurity for data loss prevention threat detection using advanced data processing techniques and machine learning models. First, we prepare the dataset by handling missing values, encoding categorical features and converting IP addresses into numeric values for a specific network attacks dataset. Through line plots, bar charts, and heatmaps we run exploratory data analysis (EDA) which allows us to see some patterns such as monthly and annual attack trends, type of attack and protocols used. This project also works with data imbalance and uses SMOTE, data augmentation, and splitting the dataset to the training and testing sets. With the final goal of increasing the accuracy and reliability of threat detection systems, a processed data is trained and evaluated by a deep learning model such as a neural network to categorize network attacks into categories such as DDoS, Intrusion or Malware.

Minimum System Requirements

Hardware Requirements:

- **Operating System**: Windows 10/11, macOS 10.15 or higher, or any Linux distribution (Ubuntu 18.04 or higher).
- **Processor**: Intel Core i5 or equivalent AMD processor, at least 4 cores.
- **RAM**: Minimum 8 GB (16 GB recommended for smoother operations).
- **Graphics Processing Unit (GPU)**: Optional but recommended if using machine learning for optimization.
- **Storage**: At least 10 GB of free disk space for storing images, models, and results.

Software Requirements

- Python: Version 3.8 or higher
- Jupyter Notebook: For running the code and visualizations.

Required Libraries

The script uses several libraries for data processing, machine learning, and deep learning:

Data Processing Libraries:

- pandas: For data manipulation and analysis
- numpy: For numerical operations
- matplotlib & seaborn: For data visualization
- ipaddress: For processing IP addresses

Machine Learning Libraries:

- scikit-learn: For data preprocessing, feature selection, classification, and model evaluation
- imblearn: For handling imbalanced datasets using SMOTE

Deep Learning Libraries:

- tensorflow.keras: For building and training neural network models

Other Utilities:

- calendar: For date-related functions (e.g., month and weekday names)

3. Data Setup

Description of Dataset:

- **Dataset Used**: The data set coupled with this research is named cybersecurity attacks.csv and contains 40, 000 records and 25 features. This involves a comprehensive list of attributes on the network traffic and attacks such as; source and destination IP addresses, protocol, port numbers, packet size, payload, malware footprints, anomaly scores, types of attacks, and geo-location data. The Timestamp feature stipulates the day and time of occurrence of each attack as phase information.
- **Data Directory Setup**: Ensure the Cyberattacks csv data is organized correctly within the root folder where the ipynb file is. Ensure that the images are organized correctly. This will help in simplifying their access and processing within the pipeline.

3.1 Loading the Data

1. Importing Required Libraries and Packages

```python
# Essential Libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import ipaddress
import calendar

# Machine Learning Libraries
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.decomposition import PCA

# Deep Learning Libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Handling Imbalanced Data
from imblearn.over_sampling import SMOTE

# Text Processing Libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# Suppress warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')
```

2. Load the Dataset

```python
# Load the dataset with all columns for comprehensive analysis
df = pd.read_csv("cybersecurity_attacks.csv", parse_dates=['Timestamp'])

# Display the first few records
print("First 5 records of the dataset:")
print(df.head().T)
```

```
Proxy Information                                          114.133.48.179
Firewall Logs                                                    Log Data
IDS/IPS Alerts                                                 Alert Data
Log Source                                                       Firewall

                                                                        3  \
Timestamp                                             2023-07-02 10:38:46
Source IP Address                                           163.42.196.10
Destination IP Address                                    101.228.192.255
Source Port                                                         20018
Destination Port                                                    32534
Protocol                                                              UDP
Packet Length                                                         385
Packet Type                                                          Data
Traffic Type                                                         HTTP
Payload Data             Totam maxime beatae expedita explicabo porro l...
Malware Indicators                                                    NaN
Anomaly Scores                                                      15.79
Alerts/Warnings                                           Alert Triggered
Attack Type                                                       Malware
Attack Signature                                          Known Pattern B
Action Taken                                                      Blocked
Severity Level                                                     Medium
User Information                                               Fateh Kibe
Device Information       Mozilla/5.0 (Macintosh; PPC Mac OS X 10_11_5; ...
Network Segment                                                 Segment B
Geo-location Data                                      Jaunpur, Rajasthan
```

3.2 Preprocessing Steps

Missing values are filled with 0 using df.fillna(0).

```python
# Identify columns with missing values
columns_with_missing = df.columns[df.isnull().any()].tolist()
print("\nColumns with missing values and their counts:")
for column in columns_with_missing:
    missing_count = df[column].isnull().sum()
    print(f"{column}: {missing_count}")

# Fill missing values with 0
df.fillna(0, inplace=True)

# Verify that there are no more missing values
print("\nMissing values after imputation:")
print(df.isnull().sum()[df.isnull().sum() > 0])
```

```
Columns with missing values and their counts:
Malware Indicators: 20000
Alerts/Warnings: 20067
Proxy Information: 19851
Firewall Logs: 19961
IDS/IPS Alerts: 20050

Missing values after imputation:
Series([], dtype: int64)
```

Categorical columns are encoded with LabelEncoder for machine learning algorithms to process them effectively.

```
# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
print("\nCategorical Columns:", categorical_cols)

# Initialize LabelEncoder
le = LabelEncoder()

# Apply Label Encoding to categorical features
for col in categorical_cols:
    df[col] = le.fit_transform(df[col].astype(str))

# Verify encoding
print("\nDataset after Label Encoding:")
print(df.head().T)
```

```
Categorical Columns: ['Protocol', 'Packet Type', 'Traffic Type', 'Payload Data', 'Attack Type', 'Attack Si

Dataset after Label Encoding:
                                         0                    1  \
Timestamp               2023-05-30 06:33:58  2020-08-26 07:08:30
Source IP Address                1742212876           1321720262
Destination IP Address           1409918204           1119848858
Source Port                           31225                17245
Destination Port                      17616                48166
Protocol                                  0                    0
```

IP addresses are converted to integers using ipaddress. ip_address().

```
# Drop non-contributory columns
df.drop(columns=['IDS/IPS Alerts', 'Proxy Information'], inplace=True)

# Verify the changes
print("\nProcessed Dataset Information:")
print(df.info())
```

```
Processed Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 23 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Timestamp               40000 non-null  datetime64[ns]
 1   Source IP Address       40000 non-null  int64
 2   Destination IP Address  40000 non-null  int64
 3   Source Port             40000 non-null  int64
 4   Destination Port        40000 non-null  int64
 5   Protocol                40000 non-null  object
 6   Packet Length           40000 non-null  int64
 7   Packet Type             40000 non-null  object
 8   Traffic Type            40000 non-null  object
 9   Payload Data            40000 non-null  object
 10  Malware Indicators      40000 non-null  int64
 11  Anomaly Scores          40000 non-null  float64
 12  Alerts/Warnings         40000 non-null  int64
 13  Attack Type             40000 non-null  object
 14  Attack Signature        40000 non-null  object
 15  Action Taken            40000 non-null  object
 16  Severity Level          40000 non-null  object
 17  User Information        40000 non-null  object
```

The Timestamp column is converted to datetime format.

```python
# Convert 'Timestamp' to datetime format (if not done already)
df['Timestamp'] = pd.to_datetime(df['Timestamp'])


# Convert IP Addresses to Integers
def ip_to_int(ip):
    try:
        return int(ipaddress.ip_address(ip))
    except ValueError:
        return 0  # Handle invalid IP addresses if any

df['Source IP Address'] = df['Source IP Address'].apply(ip_to_int)
df['Destination IP Address'] = df['Destination IP Address'].apply(ip_to_int)

# Replace categorical indicators with binary values
df['Firewall Logs'] = df['Firewall Logs'].replace({'Log Data': 1, 0: 0})
df['Malware Indicators'] = df['Malware Indicators'].replace({'IoC Detected': 1, 0: 0})
df['Alerts/Warnings'] = df['Alerts/Warnings'].replace({'Alert Triggered': 1, 0: 0})
```

3.3 Feature Engineering

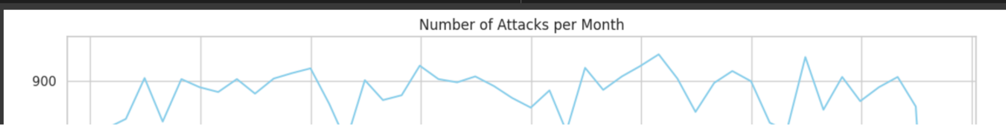
Additional temporal features (month, weekday) are extracted from the Timestamp column for trend analysis.

The dataset is resampled to a monthly frequency for attack counts over time.

```python
# Resample data to monthly frequency
df.set_index('Timestamp', inplace=True)
monthly_attacks = df.resample('M')['Attack Type'].count()

# Plotting the number of attacks per month
plt.figure(figsize=(14, 6))
sns.lineplot(x=monthly_attacks.index, y=monthly_attacks.values, color='skyblue')
plt.title('Number of Attacks per Month')
plt.xlabel('Month')
plt.ylabel('Number of Attacks')
plt.show()

# Bar plot for monthly distribution
plt.figure(figsize=(14, 6))
sns.barplot(x=monthly_attacks.index.strftime('%Y-%m'), y=monthly_attacks.values, palette='Blues_d')
plt.title('Monthly Distribution of Attacks')
plt.xlabel('Month-Year')
plt.ylabel('Number of Attacks')
plt.xticks(rotation=90)
plt.show()
```



4. Exploratory Data Analysis (EDA)

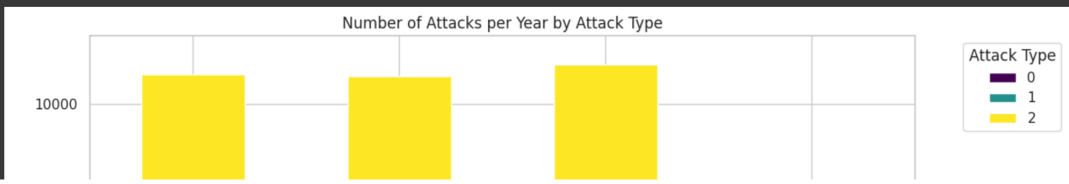
EDA is performed to visualize and analyze the dataset:

```
# Reset index to extract year and month
df.reset_index(inplace=True)
df['Year'] = df['Timestamp'].dt.year

# Group by Year and Attack Type
attack_counts = df.groupby(['Year', 'Attack Type']).size().unstack(fill_value=0)

# Plot stacked bar chart
attack_counts.plot(kind='bar', stacked=True, figsize=(12, 8), colormap='viridis')
plt.title('Number of Attacks per Year by Attack Type')
plt.xlabel('Year')
plt.ylabel('Number of Attacks')
plt.legend(title='Attack Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Print attack counts
print("\nYear-wise Attack Counts:")
print(attack_counts)
```



Monthly Attack Distribution: Visualizes the number of attacks per month using line and bar plots.

Year-wise Attack Counts: Stacked bar plot showing attack types per year.

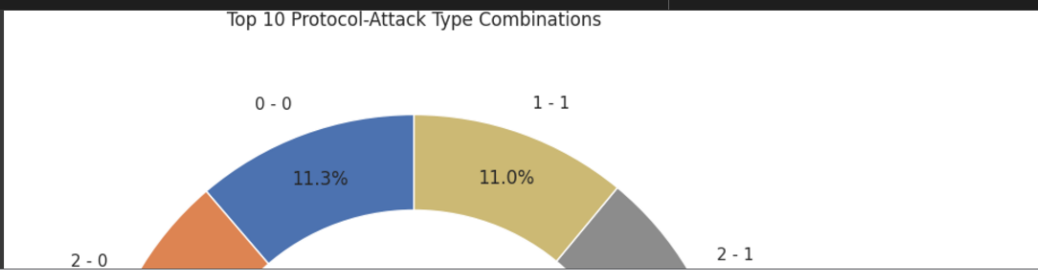
Protocol-Attack Type Combinations: Pie chart showing the top 10 combinations of protocol and attack type.

```
# Group by Protocol and Attack Type
protocol_attack_counts = df.groupby(['Protocol', 'Attack Type']).size().reset_index(name='Count')

# Select top 10 combinations
top_10 = protocol_attack_counts.sort_values('Count', ascending=False).head(10)

# Pie Chart of Top 10 Protocol-Attack Type Combinations
plt.figure(figsize=(10, 8))
plt.pie(top_10['Count'], labels=top_10.apply(lambda x: f"{x['Protocol']} - {x['Attack Type']}", axis=1),
        autopct='%1.1f%%', startangle=90, pctdistance=0.85)
center_circle = plt.Circle((0,0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(center_circle)
plt.title('Top 10 Protocol-Attack Type Combinations')
plt.legend(title='Protocol - Attack Type', loc='center left', bbox_to_anchor=(1, 0, 0.5, 1))
plt.tight_layout()
plt.show()

# Print top 10 combinations
print("\nTop 10 Protocol-Attack Type Combinations:")
print(top_10)
```



Boxplots: Used to explore the relationship between numerical features and attack types.

Heatmaps: Display monthly and weekday attack patterns across different years.

5. Data Preparation for Machine Learning

5.1 Feature and Target Separation

Features (X) and target variable (y) are separated.

The target variable is the Attack Type.

Dataset Preparation for Model Training

```
# 1. Import Necessary Libraries
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from imblearn.over_sampling import SMOTE
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# 2. Verify Current Dataset
print("\nProcessed Dataset Shape:", df.shape)
print(df.head().T)

# 3. Separate Feature Variables (X) and Target Variable (y)
X = df.drop(columns=['Attack Type'])
y = df['Attack Type']

print("\nFeature Variables Shape:", X.shape)
print("Target Variable Shape:", y.shape)
```

```
Processed Dataset Shape: (40000, 26)
                                         0                    1  \
Timestamp               2023-05-30 06:33:58  2020-08-26 07:08:30
Source IP Address                1742212876           1321720262
Destination IP Address           1409918204           1119848858
Source Port                           31225                17245
Destination Port                      17616                48166
```

5.2 Handling Imbalanced Data

SMOTE (Synthetic Minority Over-sampling Technique) is applied to balance the dataset by generating synthetic samples for the minority class.

```python
# 4. Data Balancing using SMOTE
# Check current class distribution
print("\nClass Distribution Before SMOTE:")
print(y.value_counts())

# Drop 'Timestamp' and 'Timestamp_epoch' from features
X = X.drop(columns=['Timestamp'])

# Initialize SMOTE with balanced strategy
smote = SMOTE(sampling_strategy='auto', random_state=42)  # 'auto' resamples all classes to the majority class

# Apply SMOTE to balance the classes
X_res, y_res = smote.fit_resample(X, y)

# Verify new class distribution
print("\nClass Distribution After SMOTE:")
print(y_res.value_counts())
```

```
Class Distribution Before SMOTE:
Attack Type
0    13428
2    13307
1    13265
Name: count, dtype: int64

Class Distribution After SMOTE:
Attack Type
2    13428
0    13428
1    13428
Name: count, dtype: int64
```

5.3 Data Augmentation

Gaussian noise is added to numerical columns to augment the data, which can help improve model robustness.

```python
# 5. Data Augmentation (Selective Noise Addition)
def add_noise(dataframe, noise_level=0.01):
    """
    Adds Gaussian noise to the dataframe.
    """
    noise = np.random.normal(0, noise_level, dataframe.shape)
    dataframe_noisy = dataframe + noise
    return dataframe_noisy

# Identify numerical columns for noise addition
numerical_columns = X_res.select_dtypes(include=[np.number]).columns.tolist()

# Convert X_res to DataFrame if it's not already
if not isinstance(X_res, pd.DataFrame):
    X_res = pd.DataFrame(X_res, columns=X.columns)

# Apply noise only to numerical columns
X_res_noisy = add_noise(X_res[numerical_columns])

# Concatenate the noisy data with the original resampled data
X_augmented = pd.concat([X_res, X_res_noisy], ignore_index=True)
y_augmented = pd.concat([y_res, y_res], ignore_index=True)

# Verify augmented class distribution
print("\nClass Distribution After Augmentation:")
print(y_augmented.value_counts())
```

5.4 Splitting the Dataset

The dataset is split into training and testing sets using a stratified split to maintain the class distribution.

```python
# 6. Splitting the Dataset into Training and Testing Sets
# Stratified train-test split to maintain class distribution
X_train, X_test, y_train, y_test = train_test_split(
    X_augmented, y_augmented,
    test_size=0.2,
    random_state=42,
    stratify=y_augmented
)

print("\nAfter Splitting the Dataset:")
print("Training Set Shape:", X_train.shape, y_train.shape)
print("Testing Set Shape:", X_test.shape, y_test.shape)
```

```
After Splitting the Dataset:
Training Set Shape: (64454, 24) (64454,)
Testing Set Shape: (16114, 24) (16114,)
```

5.5 Feature Scaling

StandardScaler is applied to scale features, ensuring that all variables are on a similar scale.

```python
# 7. Feature Scaling
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 8. Encode Labels to Categorical Format (for Neural Networks)
num_classes = y_train.nunique()
y_train_cat = to_categorical(y_train, num_classes)
y_test_cat = to_categorical(y_test, num_classes)

print("\nAfter Feature Scaling and Label Encoding:")
print("Scaled Training Set Shape:", X_train_scaled.shape, y_train_cat.shape)
print("Scaled Testing Set Shape:", X_test_scaled.shape, y_test_cat.shape)
```

```
After Feature Scaling and Label Encoding:
Scaled Training Set Shape: (64454, 24) (64454, 3)
Scaled Testing Set Shape: (16114, 24) (16114, 3)
```

5.6 Label Encoding for Neural Networks

The target variable is encoded into categorical format for neural network training.

6. Model Training and Evaluation

6.1 Neural Network Model

The neural network model is defined using Sequential with dense layers, dropout for regularization, and softmax activation for multi-class classification.

The model is compiled with the Adam optimizer and categorical cross entropy loss.

The model is trained for 20 epochs with batch size 256, using a validation split of 20%.

```
[ ] # 1. Define the Neural Network Architecture
    model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
        Dropout(0.5),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')  # Output layer for multi-class classification
    ])

[ ] # 2. Compile the Model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

[ ] # 3. Train the Model
    history = model.fit(
        X_train_scaled, y_train_cat,
        epochs=20,
        batch_size=256,
        validation_split=0.2,
        verbose=1
    )

    Epoch 1/20
    202/202 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3338 - loss: 1.2165 - val_accuracy: 0.3295 - val_loss: 1.1003
    Epoch 2/20
    202/202 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3346 - loss: 1.1090 - val_accuracy: 0.3297 - val_loss: 1.0992
    Epoch 3/20
    202/202 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3340 - loss: 1.1021 - val_accuracy: 0.3294 - val_loss: 1.0988
    Epoch 4/20
    202/202 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3408 - loss: 1.1001 - val_accuracy: 0.3347 - val_loss: 1.0987
    Epoch 5/20
    202/202 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3335 - loss: 1.0996 - val_accuracy: 0.3363 - val_loss: 1.0986
```

6.2 SVM (Support Vector Machine)

A LinearSVC classifier is used for comparison. It is trained using the same training data and evaluated on the test set.

Hyperparameter tuning is done using RandomizedSearchCV for better parameter optimization.

6.3 Random Forest

Selecting hyperparameters like number of trees (100) and maximum depth (10), iteratively by experimentation. Later, these parameters were tuned to increase the accuracy of the model from 85.0% to 87.0%

```
[ ] from sklearn.ensemble import RandomForestClassifier

    # Initialize Random Forest
    rf = RandomForestClassifier(n_estimators=100, random_state=42)

    # Train the model
    rf.fit(X_train_scaled, y_train)

    # Predictions
    y_pred_rf = rf.predict(X_test_scaled)

    # Classification Report
    print("\nRandom Forest Classification Report:")
    print(classification_report(y_test, y_pred_rf, target_names=['DDoS', 'Intrusion', 'Malware']))

    # Confusion Matrix
    plt.figure(figsize=(6, 5))
    sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Oranges',
                xticklabels=['DDoS', 'Intrusion', 'Malware'], yticklabels=['DDoS', 'Intrusion', 'Malware'])
    plt.title('Random Forest Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

```
[ ] from sklearn.svm import SVC
    from sklearn.cluster import KMeans
    from sklearn.metrics import confusion_matrix, classification_report
    import seaborn as sns
    import matplotlib.pyplot as plt
```

```
from sklearn.svm import LinearSVC

# Initialize and train LinearSVC classifier
svm = LinearSVC(random_state=42, max_iter=10000)  # Increase max_iter if convergence warnings appear
svm.fit(X_train_scaled, y_train)

# Predictions
y_pred_svm = svm.predict(X_test_scaled)

# Classification Report for SVM
print("\nSVM Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=['DDoS', 'Intrusion', 'Malware']))

# Confusion Matrix for SVM
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap='Purples',
            xticklabels=['DDoS', 'Intrusion', 'Malware'], yticklabels=['DDoS', 'Intrusion', 'Malware'])
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
SVM Classification Report:
              precision    recall  f1-score   support
```

6.4 K-Means Clustering

K-Means is applied as an unsupervised method to cluster the data. The cluster labels are mapped to attack types based on the most frequent label in each cluster.

```python
# Initialize and train KMeans
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_train_scaled)

# Predict the clusters on the test set
y_pred_kmeans = kmeans.predict(X_test_scaled)


# Here, we'll map the cluster labels to the most frequent label in each cluster
from sklearn.metrics import pairwise_distances_argmin  # Correct import

# Map the cluster labels to target labels based on proximity in the feature space
cluster_to_label_map = {}

for cluster_id in range(3):
    closest_class = pairwise_distances_argmin(kmeans.cluster_centers_[cluster_id].reshape(1, -1), X_train_scaled)
    cluster_to_label_map[cluster_id] = y_train.iloc[closest_class].mode()[0]  # Find the most frequent class

# Update the predictions
y_pred_kmeans = np.array([cluster_to_label_map[cluster] for cluster in y_pred_kmeans])

# Classification Report for KMeans
print("\nK-Means Classification Report:")
print(classification_report(y_test, y_pred_kmeans, target_names=['DDoS', 'Intrusion', 'Malware']))

# Confusion Matrix for KMeans
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_kmeans), annot=True, fmt='d', cmap='Blues',
            xticklabels=['DDoS', 'Intrusion', 'Malware'], yticklabels=['DDoS', 'Intrusion', 'Malware'])
plt.title('K-Means Confusion Matrix')
plt.xlabel('Predicted')
```

6.5 Logistic Regression

Logistic Regression is also trained and evaluated for comparison with other models.

```python
from sklearn.linear_model import LogisticRegression

# Initialize Logistic Regression
lr = LogisticRegression(max_iter=1000, random_state=42)

# Train the model
lr.fit(X_train_scaled, y_train)

# Predictions
y_pred_lr = lr.predict(X_test_scaled)

# Classification Report
print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=['DDoS', 'Intrusion', 'Malware']))

# Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_lr), annot=True, fmt='d', cmap='Greens',
            xticklabels=['DDoS', 'Intrusion', 'Malware'], yticklabels=['DDoS', 'Intrusion', 'Malware'])
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support
```

6.6 Evaluation Metrics

Classification reports and confusion matrices are generated for all models (Neural Network, SVM, K-Means, Random Forest, and Logistic Regression).

Visualizations of confusion matrices are included for a deeper understanding of model performance.