National College of Ireland

Configuration Manual

MSc Research Project
MSc IN CYBERSECURITY

SANDRA RAVI
Student ID: 23178302

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	SANDRA RAVI
Student ID:	23178302
Programme:	MSc in Cybersecurity **Year:** 2024
Module:	MSc Research Project
Lecturer:	Niall Heffernan
Submission Due Date:	12/12/2024
Project Title:	BUILDING AN INTEGRATED IoT SECURITY FRAMEWORK FOR SMART HOMES
Word Count:	1114 **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:	SANDRA RAVI
Date:	12/12/2024

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

SANDRA RAVI
Student ID: 23178302

1. System Requirement

Here's an ideal setup for my project:

Desktop or Laptop

- **CPU**: Intel i5
- **RAM**: 16 GB
- **GPU**: NVIDIA RTX 3060
- **Storage**: 256 GB SSD
- **Google Colab**: Free GPU support for small-scale experiments.
- **AWS/GCP**: On-demand high-performance machines for model training.

2. Data Configuration

2.1. Dataset Overview

The datasets utilized in this study are pivotal for understanding and enhancing IoT security in smart homes. The BoTNeTIoT-L01 dataset is the most recent dataset containing traffic from nine IoT devices, sniffed using Wireshark in a local network through a central switch. It includes two prominent botnet attacks: Mirai and Gafgyt. The dataset contains twenty-three statistically engineered features extracted from the .pcap files. These features were computed over a 10-second time window with a decay factor of 0.1, facilitating detailed temporal analysis. The RT_IOT2022 dataset complements this by including various attack types, thus providing a broader evaluation spectrum for the proposed security framework.

BoTNeTIoT-L01 dataset

L0.1_mean	HH_L0.1_std	HH_L0.1_magnitude	...	HpHp_L0.1_mean	HpHp_L0.1_std	HpHp_L0.1_magnitude	HpHp_L0.1_radius	HpHp_L0.1_covariance	HpHp_L0.1_pcc	Device_Name	Attack	Attack_subType	label
98.0	0.000000e+00	98.000000	...	98.0	0.000000	98.000000	0.000000e+00	0.0	0.0	Danmini_Doorbell	gafgyt	combo	0
98.0	1.348699e-06	138.592929	...	98.0	0.000001	138.592929	1.818989e-12	0.0	0.0	Danmini_Doorbell	gafgyt	combo	0
66.0	0.000000e+00	114.856432	...	66.0	0.000000	114.856432	0.000000e+00	0.0	0.0	Danmini_Doorbell	gafgyt	combo	0
74.0	0.000000e+00	74.000000	...	74.0	0.000000	74.000000	0.000000e+00	0.0	0.0	Danmini_Doorbell	gafgyt	combo	0
74.0	9.536743e-07	74.000000	...	74.0	0.000000	74.000000	0.000000e+00	0.0	0.0	Danmini_Doorbell	gafgyt	combo	0

RT_IOT2022 dataset

t	fwd_data_pkts_tot	bwd_data_pkts_tot	...	active.std	idle.min	idle.max	idle.tot	idle.avg	idle.std	fwd_init_window_size	bwd_init_window_size	fwd_last_window_size	Attack_type
5	3	3	...	0.0	2.972918e+07	2.972918e+07	2.972918e+07	2.972918e+07	0.0	64240.0	26847.0	502.0	MQTT_Publish
5	3	3	...	0.0	2.985528e+07	2.985528e+07	2.985528e+07	2.985528e+07	0.0	64240.0	26847.0	502.0	MQTT_Publish
5	3	3	...	0.0	2.984215e+07	2.984215e+07	2.984215e+07	2.984215e+07	0.0	64240.0	26847.0	502.0	MQTT_Publish
5	3	3	...	0.0	2.991377e+07	2.991377e+07	2.991377e+07	2.991377e+07	0.0	64240.0	26847.0	502.0	MQTT_Publish
5	3	3	...	0.0	2.981470e+07	2.981470e+07	2.981470e+07	2.981470e+07	0.0	64240.0	26847.0	502.0	MQTT_Publish

2.2. Data Pre-processing Steps

```
[ ] from collections import Counter

    # Identify categorical columns (non-numeric data types)
    categorical_cols = df.select_dtypes(include=['object']).columns

    # Use Label Encoding to convert categorical columns to numeric
    for col in categorical_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])

    # Handle missing values in numeric columns
    numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
    num_imputer = SimpleImputer(strategy='mean')  # Replace missing values with the mean value
    df[numeric_cols] = num_imputer.fit_transform(df[numeric_cols])

    # Separate features and target
    target_column = 'Attack_type'
    X = df.drop(target_column, axis=1)
    y = df[target_column]

    # Scale features using StandardScaler
    scaler = StandardScaler()
    X = scaler.fit_transform(X)


    # Remove or re-label classes with only one sample in the target variable
    # 1. Identify classes with only one sample:
    class_counts = y.value_counts()
    classes_to_remove = class_counts[class_counts == 1].index

    # Re-label these classes to a more frequent class
```

Label Encoding for Categorical Features:

```
# Re-label these classes to a more frequent class
# (Choose a suitable class label to replace them with)
df.loc[y.isin(classes_to_remove), target_column] = y.mode()[0] #Replace with the mode

# After applying either option, update X and y
X = df.drop(target_column, axis=1)
y = df[target_column]
# --- Changes end here ---

# Split the data into training and testing sets (80-20 split with stratification)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Print class distribution before SMOTE
print("Class distribution before SMOTE:")
print("Training set:", Counter(y_train))
print("Testing set:", Counter(y_test))

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Print class distribution after SMOTE
print("\nClass distribution after SMOTE (training set):")
print(Counter(y_train_resampled))

print("\nData is ready for model training with SMOTE and stratification!")
```

Selected categorical columns with select_dtypes and then applied LabelEncoder on them to get numerical form that can be accepted by the machine learning algorithm.

Handling Missing Values in Numeric Columns:

Selected numeric columns and then used SimpleImputer by setting strategy to mean. Then missing values get replaced with the mean of that respective column.

Feature and Target Separation:

Splitting feature set X and target y into the place where target column is 'Attack_type'. Feature Scaling:

Feature Scaling:

Standardized all feature values with StandardScaler to have mean of 0 and SD of 1, which improves model convergence during training.

Handling Classes with Single Samples:

Found classes in the target variable with only one sample and re-labeled them to the most frequent class (mode) to avoid errors or instability during model training.

Train-Test Split with Stratification:

Split the data into 80% training and 20% testing sets with stratification so that both sets maintain class distribution.

Handling Class Imbalance Using SMOTE:

SMOTE to the training data to generate synthetic samples for minority classes to balance the distribution of classes.

Class Distribution Check:

Checked and compared the class distributions before and after applying SMOTE to ensure that there is proper handling of imbalances in the training set.

2.3. Model Configuration of RT_IOT2022

2.3.1 Model Type: Decision Tree Classifier

```
[ ]
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    import seaborn as sns
    import matplotlib.pyplot as plt

    # Initialize and train the Decision Tree Classifier
    dt_classifier = DecisionTreeClassifier(random_state=42)  # You can tune hyperparameters here
    dt_classifier.fit(X_train_resampled, y_train_resampled)

    # Make predictions on the test set
    y_pred = dt_classifier.predict(X_test)

    # Evaluate the model
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))

    # Plot the confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()
```

A random state of 42 is used to initialize the Decision Tree Classifier in order to ensure reproducibility. To compensate for the class imbalance, it is trained on a preprocessed and SMOTE-resampled dataset. In addition to label-encoding categorical variables, StandardScaler is used to scale the training data features. To impute missing values, the mean method is employed. The model's performance is assessed using a stratified test set, a thorough classification report, and a confusion matrix after it has been trained using the resampled training set. The classifier's default settings are employed, but depending on particular needs, they can be adjusted to maximize performance.

2.3.2 **Model Type:** Random Forest Classifier

```
[ ]  #  code for above to train random forest  clasifier

    from sklearn.ensemble import RandomForestClassifier

    # Initialize and train the Random Forest Classifier
    rf_classifier = RandomForestClassifier(random_state=42)  # You can tune hyperparameters here
    rf_classifier.fit(X_train_resampled, y_train_resampled)

    # Make predictions on the test set
    y_pred = rf_classifier.predict(X_test)

    # Evaluate the model
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))

    # Plot the confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix - Random Forest")
    plt.show()
```

The Random Forest Classifier is set up with a random state of 42 for reproducibility, using default hyperparameters that can be tuned for better performance. It is trained on a balanced dataset prepared through SMOTE resampling to deal with class imbalance in the training data. The features are scaled using StandardScaler, and categorical variables are encoded using label encoding, with missing values imputed using the mean. The model will predict outcomes on a stratified test set. Its performance will be assessed using accuracy, a detailed classification report, including precision, recall, and F1-score, and a confusion matrix. A confusion matrix visualization can emphasize the predictive performance of the classifier across classes.

2.3.3 **Model Type:** LightGBM Classifier

```
[ ]  import re # import regex module for renaming

    # Function to clean column names
    def clean_column_names(df):
        """Replaces special characters and spaces in column names with underscores."""
        df = df.rename(columns=lambda x: re.sub('[^A-Za-z0-9_]+', '_', x))
        return df

    # Clean column names in your DataFrames
    X_train_resampled = clean_column_names(pd.DataFrame(X_train_resampled))
    X_test = clean_column_names(pd.DataFrame(X_test))

    # Initialize and train the LightGBM Classifier
    lgb_classifier = lgb.LGBMClassifier(random_state=42)  # You can tune hyperparameters here
    lgb_classifier.fit(X_train_resampled, y_train_resampled)

    # Make predictions on the test set
    y_pred = lgb_classifier.predict(X_test)

    # Evaluate the model
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))

    # Plot the confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix - LightGBM")
    plt.show()
```

The LightGBM Classifier is used with a random state of 42 to ensure reproducibility and default hyperparameters, which can be fine-tuned for better performance. It was trained on a resampled balanced dataset with SMOTE to reduce class imbalance. Features were preprocessed by scaling using StandardScaler, label encoding of categorical variables, and mean imputation of missing values.

2.3.4 **Model Type:** Xgboost Classifier

```
[ ] # code for above to train Xgboost

    !pip install xgboost

    import xgboost as xgb
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    import matplotlib.pyplot as plt
    import seaborn as sns

    # Initialize and train the XGBoost Classifier
    xgb_classifier = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss') # Use_label_encoder is deprecated
    xgb_classifier.fit(X_train_resampled, y_train_resampled)

    # Make predictions on the test set
    y_pred = xgb_classifier.predict(X_test)

    # Evaluate the model
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))

    # Plot the confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix - XGBoost")
    plt.show()
```

The XGBoost Classifier is set up with a random state of 42 for the sake of reproducibility, and it uses the logloss evaluation metric. It will use default hyperparameters, namely gradient-boosted decision trees, and supports multi-class classification. The performance of the model is tested using a stratified test set on accuracy, classification report with precision, recall, F1-score, and a confusion matrix, and the confusion matrix is displayed to gain insights into how well the model predicts other classes.

2.3.5 Model Type: MultiLayer Perception

```
[ ]  #  code for above to train multilayer perception

     from sklearn.neural_network import MLPClassifier

     # Initialize and train the Multilayer Perceptron Classifier
     mlp_classifier = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42) # You can tune hyperparameters here
     mlp_classifier.fit(X_train_resampled, y_train_resampled)

     # Make predictions on the test set
     y_pred = mlp_classifier.predict(X_test)

     # Evaluate the model
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("\nClassification Report:\n", classification_report(y_test, y_pred))

     # Plot the confusion matrix
     cm = confusion_matrix(y_test, y_pred)
     plt.figure(figsize=(10, 7))
     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
     plt.xlabel("Predicted")
     plt.ylabel("Actual")
     plt.title("Confusion Matrix - MLP")
     plt.show()

     # Plot training loss curve (if available)
     plt.plot(mlp_classifier.loss_curve_)
     plt.title("MLP Training Loss Curve")
     plt.xlabel("Epochs")
     plt.ylabel("Loss")
     plt.show()
```

This Multilayer Perceptron Classifier is set up with one hidden layer of 100 neurons, a maximum of 500 iterations for training, and a random state of 42 for reproducibility. It uses the adam optimizer and the ReLU activation function by default, suited for non-linear problems. Evaluation Metrics Accuracy All the metrics are detailed in the classification report- that includes precision, recall and F1 score A confusion matrix that is visualized for assessing predictive performance. Training Loss Curve-over epochs.

2.4 Model Congfiguration of RT_IOT2022

2.4.1 Random Forest Classifier

```
[ ]  from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, accuracy_sco

    # Train Random Forest Classifier
    rf_classifier = RandomForestClassifier(random_state=42)
    rf_classifier.fit(x_train_smote, y_train_smote)
    rf_predictions = rf_classifier.predict(x_test)

    # Evaluate the model with standard metrics
    print("\nRandom Forest Classifier Metrics:")
    print("Accuracy:", accuracy_score(y_test, rf_predictions))
    print("Precision (Weighted):", precision_score(y_test, rf_predictions, average='weighted'))
    print("Recall (Weighted):", recall_score(y_test, rf_predictions, average='weighted'))
    print("F1 Score (Weighted):", f1_score(y_test, rf_predictions, average='weighted'))

    # Classification report for detailed evaluation
    print("\nClassification Report:")
    print(classification_report(y_test, rf_predictions, digits=3))

    # Confusion Matrix
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, rf_predictions))

    # ROC-AUC Score (for binary or one-vs-rest classification)
    if len(set(y_test)) == 2:  # Binary classification
        roc_auc = roc_auc_score(y_test, rf_classifier.predict_proba(x_test)[:, 1])
        print("\nROC-AUC Score (Binary):", roc_auc)
    else:  # Multiclass classification
        roc_auc = roc_auc_score(y_test, rf_classifier.predict_proba(x_test), multi_class='ovr', aver
        print("\nROC-AUC Score (Multiclass):", roc_auc)
```

2.4.2 Logistic Regression

```
[ ]  import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

    # Train Logistic Regression
    lr_classifier = LogisticRegression(random_state=42, max_iter=1000)  # Increased max
    lr_classifier.fit(x_train_smote, y_train_smote)
    lr_predictions = lr_classifier.predict(x_test)

    # Evaluate the model
    print("\nLogistic Regression Classifier Metrics:")
    print("Accuracy:", accuracy_score(y_test, lr_predictions))
    print("Precision (Weighted):", precision_score(y_test, lr_predictions, average='wei
    print("Recall (Weighted):", recall_score(y_test, lr_predictions, average='weighted'
    print("F1 Score (Weighted):", f1_score(y_test, lr_predictions, average='weighted'))

    # Classification report for detailed evaluation
    print("\nClassification Report:\n", classification_report(y_test, lr_predictions, d

    # Confusion Matrix
    cm = confusion_matrix(y_test, lr_predictions)

    # Plot the Confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=set(y_test), yticklа
    plt.title("Confusion Matrix - Logistic Regression")
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.show()
```

2.4.3 Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Train Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(x_train_smote, y_train_smote)
dt_predictions = dt_classifier.predict(x_test)

# Evaluate the model
print("\nDecision Tree Classifier Metrics:")
print("Accuracy:", accuracy_score(y_test, dt_predictions))
print("Precision (Weighted):", precision_score(y_test, dt_predictions, average='weighted'))
print("Recall (Weighted):", recall_score(y_test, dt_predictions, average='weighted'))
print("F1 Score (Weighted):", f1_score(y_test, dt_predictions, average='weighted'))

# Classification report for detailed evaluation
print("\nClassification Report:\n", classification_report(y_test, dt_predictions, digits=3))

# Confusion Matrix
cm = confusion_matrix(y_test, dt_predictions)

# Plot the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=set(y_test), yticklabels=set(y_test))
plt.title("Confusion Matrix - Decision Tree Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

2.4.4 LGBM Classifier

```
[ ] from lightgbm import LGBMClassifier
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

    # Train LGBM Classifier
    lgbm_classifier = LGBMClassifier(random_state=42)
    lgbm_classifier.fit(x_train_smote, y_train_smote)
    lgbm_predictions = lgbm_classifier.predict(x_test)

    # Evaluate the model
    print("\nLGBM Classifier Metrics:")
    print("Accuracy:", accuracy_score(y_test, lgbm_predictions))
    print("Precision (Weighted):", precision_score(y_test, lgbm_predictions, average='weighted'))
    print("Recall (Weighted):", recall_score(y_test, lgbm_predictions, average='weighted'))
    print("F1 Score (Weighted):", f1_score(y_test, lgbm_predictions, average='weighted'))

    # Classification report for detailed evaluation
    print("\nClassification Report:\n", classification_report(y_test, lgbm_predictions, digits=3))

    # Confusion Matrix
    cm = confusion_matrix(y_test, lgbm_predictions)

    # Plot the Confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges', xticklabels=set(y_test), yticklabels=set(y_test))
    plt.title("Confusion Matrix - LGBM Classifier")
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.show()
```

2.4.5 XGBoost Classifier

```
[ ] xgb_predictions = xgb_classifier.predict(x_test)

    # Evaluate the model
    print("\nXGBoost Classifier Metrics:")
    print("Accuracy:", accuracy_score(y_test, xgb_predictions))
    print("Precision (Weighted):", precision_score(y_test, xgb_predictions, average='weighted'))
    print("Recall (Weighted):", recall_score(y_test, xgb_predictions, average='weighted'))
    print("F1 Score (Weighted):", f1_score(y_test, xgb_predictions, average='weighted'))

    # Classification report for detailed evaluation
    print("\nClassification Report:\n", classification_report(y_test, xgb_predictions, digits=3))

    # Confusion Matrix
    cm = confusion_matrix(y_test, xgb_predictions)

    # Plot the Confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=set(y_test), yticklabels=set(y_test))
    plt.title("Confusion Matrix - XGBoost Classifier")
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.show()

    # Optional: ROC-AUC Score
    if len(set(y_test)) == 2:  # Binary classification
        roc_auc = roc_auc_score(y_test, xgb_classifier.predict_proba(x_test)[:, 1])
        print("\nROC-AUC Score (Binary):", roc_auc)
    else:  # Multiclass classification
        roc_auc = roc_auc_score(y_test, xgb_classifier.predict_proba(x_test), multi_class='ovr', average='weighted')
        print("\nROC-AUC Score (Multiclass):", roc_auc)
```

2.4.6 MLP Classifier

```
[ ] mlp_predictions = mlp_classifier.predict(x_test)

    # Evaluate the model
    print("\nMLP Classifier Metrics:")
    print("Accuracy:", accuracy_score(y_test, mlp_predictions))
    print("Precision (Weighted):", precision_score(y_test, mlp_predictions, average='weighted'))
    print("Recall (Weighted):", recall_score(y_test, mlp_predictions, average='weighted'))
    print("F1 Score (Weighted):", f1_score(y_test, mlp_predictions, average='weighted'))

    # Classification report for detailed evaluation
    print("\nClassification Report:\n", classification_report(y_test, mlp_predictions, digits=3))

    # Confusion Matrix
    cm = confusion_matrix(y_test, mlp_predictions)

    # Plot the Confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=set(y_test), yticklabels=set(y_test))
    plt.title("Confusion Matrix - MLP Classifier")
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.show()

    # Optional: ROC-AUC Score
    if len(set(y_test)) == 2:  # Binary classification
        roc_auc = roc_auc_score(y_test, mlp_classifier.predict_proba(x_test)[:, 1])
        print("\nROC-AUC Score (Binary):", roc_auc)
    else:  # Multiclass classification
        roc_auc = roc_auc_score(y_test, mlp_classifier.predict_proba(x_test), multi_class='ovr', average='weighted')
        print("\nROC-AUC Score (Multiclass):", roc_auc)
```

References

1. **Abubakar, A., & Pranggono, B.** (2023). *Machine Learning for Intrusion Detection in IoT Networks: A Review*. Published in *Journal of Network Security and Applications*.
   Description: A comprehensive review of machine learning techniques for detecting intrusions in IoT environments.

2. **Al-Garadi, M. A., Mohamed, A., et al.** (2022). *IoT Intrusion Detection Using Ensemble Machine Learning Models*. Published in *IEEE Internet of Things Journal*.
   Description: Explores the use of ensemble methods like Random Forest and Gradient Boosting for intrusion detection.

3. **Meidan, Y., Bohadana, M., et al.** (2022). *ProfilIoT: A Machine Learning-Based Approach for IoT Intrusion Detection*. Published in *Proceedings of ACM IoT Conference*.
   Description: Introduces ProfilIoT, a scalable framework for intrusion detection using supervised learning.

4. **AI in IoT Security: Challenges and Solutions**
   Source: Palo Alto Networks Research
   Link: https://www.paloaltonetworks.com/iot-security
   Description: Discusses AI and machine learning's role in ensuring IoT system security against modern threats.

5. **Yang, T., & Zhao, W.** (2023). *Deep Learning Techniques for IoT Intrusion Detection: Challenges and Solutions*. Published in *Springer Cybersecurity Series*.
   Description: Focuses on deep learning methods such as CNNs and RNNs for detecting intrusions in IoT networks.

6. **Shafiq, M., et al.** (2023). *Anomaly Detection in IoT Devices Using Hybrid ML Models*. Published in *Journal of Cyber-Physical Systems*.
   Description: Investigates hybrid models combining machine learning and rule-based systems for IoT security.

7. **RT-IoT-Intrusion** by *iotSecTools*
   Link: https://github.com/iotSecTools/RT-IoT-Intrusion
   Description: Provides an analysis pipeline for IoT datasets like RT_IOT2022 with feature engineering and model training.

8. **IoT-Security-ML** by *data4iot*
   Link: https://github.com/data4iot/IoT-Security-ML

Description: Focuses on supervised and unsupervised learning techniques for intrusion detection.

9. **IoT-Intrusion-DeepLearning** by *AI-IoT-Labs*
   Link: https://github.com/AI-IoT-Labs/IoT-Intrusion-DeepLearning
   Description: Implements deep learning models like CNN-GRU for detecting anomalies in IoT traffic.