National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Deepika Porkodian Suganraj		
Student ID:	22185712		
Programme:	Masters in Cybersecurity	**Year:**	2023
Module:	Msc Research Project		
Supervisor:	Jawad Salahuddin		
Submission Due Date:	31/1/2024		
Project Title:	Information Security and Data Protection: A Review of Challenges and Influencing Factor Faced In IT		
Word Count:	1444 **Page Count** 8		

Signature:	Deepika Porkodian Suganraj
Date:	31/1/2024

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

Deepika Porkodian Suganraj

Student ID: x22185712

1 Introduction

This code combines deep neural networks and machine learning to coordinate the creation and evaluation of an intrusion detection system. It starts with handling data and visualisation and then uses a Deep Belief Network-Gated Recurrent Unit (DBN-GRU) architecture to train the model. Model building and evaluation are done with TensorFlow and Keras, displaying metrics like accuracy, precision, recall, and F1-score. The code ends with Receiver Operating Characteristic (ROC) curves and confusion matrices that visually depict the model's learning curve.

2 Hardware Requirements

The proposed project will be working under the following hardware requirements.

- Processor (CPU): Apple M1 chip
- Memory (RAM): 8GB

3 Software Requirements

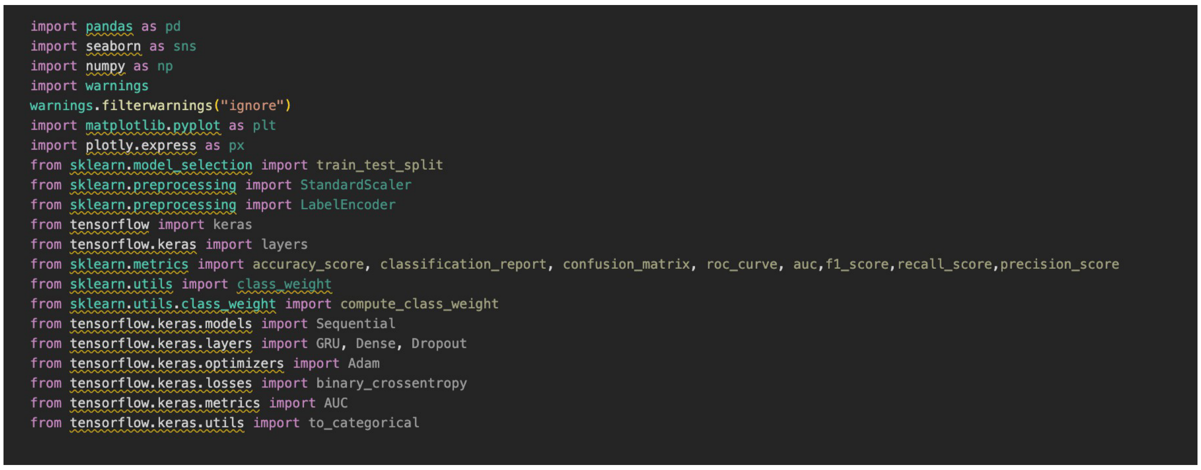
- Operating system – Macbook
- Python - The main programming language for creating and executing machine learning algorithms is Python.
- Using pip to install Jupyter NotebookScikit-Learn: Tasks involving data pre-processing and evaluation are performed with Scikit-Learn. It offers resources for feature selection, data preprocessing, and model assessment.
- Kearas and TensorFlow: Two well-known deep learning frameworks are TensorFlow and Keras. Neural network model construction and training is done with them. A complete set of tools for creating and implementing machine learning models is offered by TensorFlow, while Keras is an advanced neural network API that operates on top of TensorFlow.
- Hyperparameter optimisation, regularisation, and long-term backpropagation:These phrases describe certain methods used in the testing and training phases. Regularisation aids in preventing overfitting, long-term backpropagation is a technique for updating weights and biases during training, and hyperparameter optimisation adjusts the model's parameters for best performance.
- NSL-KDD99 Information Set: The intrusion detection system is trained and assessed using the NSL-KDD99 dataset as a benchmark. It has attributes including flags, network-related data, and protocol classifications. The dataset has thorough documentation and is updated frequently.

4 Steps Involved for the Proposed approach

- Loading Data
- Data Visualisation, Label Encoding
- Data Analysis
- Setting Up the Target and Feature Variables
- Compute Metrics (F1-Score, Accuracy, Precision, and Recall)
- Perplexity Matrix ROC Curve

5 Key Functions in Project

1. Setting up Python



```python
import pandas as pd
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc,f1_score,recall_score,precision_score
from sklearn.utils import class_weight
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.metrics import AUC
from tensorflow.keras.utils import to_categorical
```

Figure 1

Pandas Reference:

- pd: A Pandas library alias for data analysis and manipulation.

Libraries for Seaborn and Matplotlib:

- ‘Sns’: For statistical data visualisation, sns is the Seaborn library's alias.
- plt: An alias for the Matplotlib package, which is utilised in the production of interactive, animated, and static graphics.

NumPy Library:

- np: A NumPy library alias used for performing numerical operations on matrices and arrays.

Warnings Module:

- warnings.filterwarnings("ignore"): sets up the code to run without regard for warning messages.

Plotly Express Library:

- Px: An alias for the Plotly Express library, which is utilised in the production of interactive visualisations

Science-Learn Library:

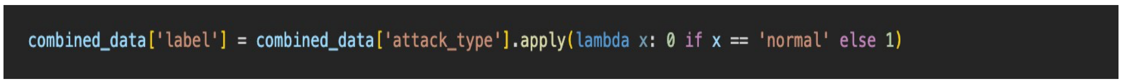
- The function train_test_split divides the dataset into sets for testing and training.
- StandardScaler: A class for scaling to unit variance and subtracting the mean from features to standardise them.
- The class LabelEncoder is used to encode numerical values for categorical labels.
- Model performance evaluation functions and metrics include accuracy_score, classification_report, confusion_matrix, roc_curve, auc, f1_score, recall_score, and precision_score.
- class_weight: A function that determines class weights in datasets that are unbalanced.

Libraries for TensorFlow and Keras:

- The primary library for creating and refining machine learning models is called TensorFlow.
- Keras is a TensorFlow-based high-level neural network API.
- Sequential: Layer stack model in a linear fashion.
- Several layers are employed to build the neural network model: GRU, Dense, and Dropout.
- Adam: Gradient-based optimisation technique for optimisation.
- Loss function for binary classification issues is binary_crossentropy.
- AUC stands for area under the ROC curve, which is a model evaluation statistic.
- To_categorical: For categorical variables, this function translates a class vector to a binary class matrix.
- TensorFlow library installation is done with the!pip install tensorflow command.

2. Downloading Dataset from Kaggle

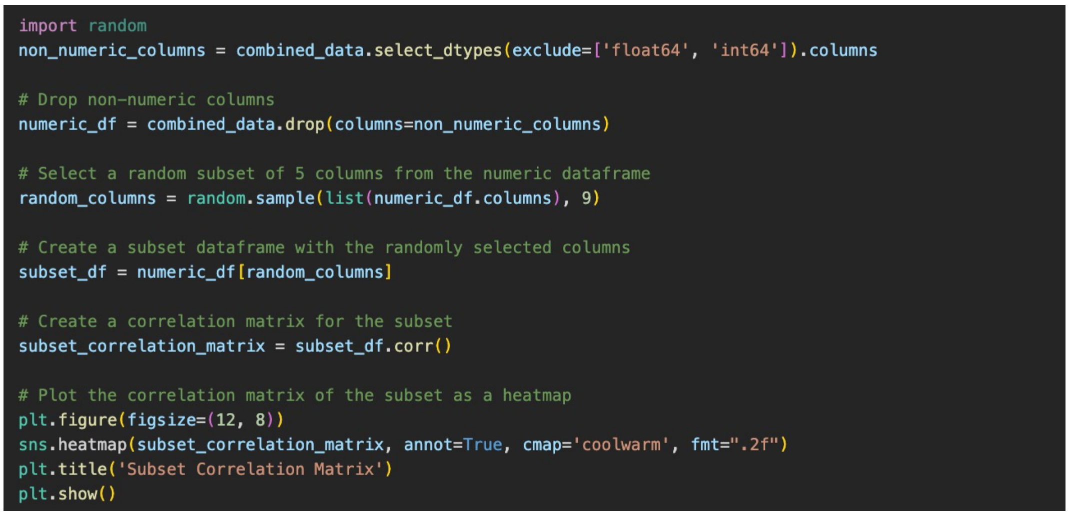
3. For the purpose of binary classification for intrusion detection or anomaly identification, instances with the 'attack_type' attribute set to 'normal' are labelled as 0, whereas instances that are not normal are labelled as 1.



```
combined_data['label'] = combined_data['attack_type'].apply(lambda x: 0 if x == 'normal' else 1)
```

Figure 2 : Assigns a binary label (0 for 'normal', 1 for other attack types) based on the 'attack_type' column in the DataFrame 'combined_data'.

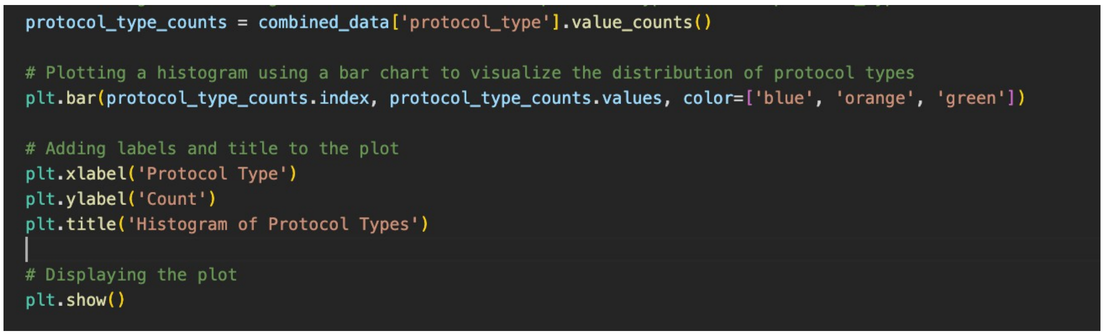
3. Data Visualization :
Using seaborn and matplotlib, this code generates a subset DataFrame with the columns that are randomly selected from the 'combined_data' DataFrame, computes the correlation matrix for the subset, and displays the results as a heatmap. The pairwise relationships between the numerical columns that were selected at random can be seen in the heatmap.



```python
import random
non_numeric_columns = combined_data.select_dtypes(exclude=['float64', 'int64']).columns

# Drop non-numeric columns
numeric_df = combined_data.drop(columns=non_numeric_columns)

# Select a random subset of 5 columns from the numeric dataframe
random_columns = random.sample(list(numeric_df.columns), 9)

# Create a subset dataframe with the randomly selected columns
subset_df = numeric_df[random_columns]

# Create a correlation matrix for the subset
subset_correlation_matrix = subset_df.corr()

# Plot the correlation matrix of the subset as a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(subset_correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Subset Correlation Matrix')
plt.show()
```

Figure 3: Randomly selecting 8 numeric columns from the DataFrame, creates a subset, computes the correlation matrix, and visualizes it as a heatmap using Seaborn.

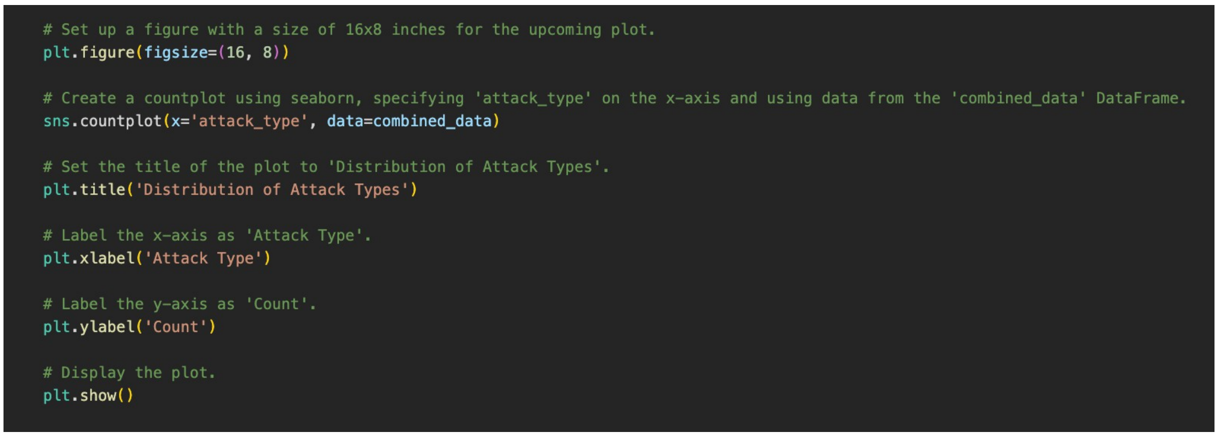
4. The distribution of protocol types in the 'combined_data' DataFrame is shown by the histogram that is produced by this function. The counts of each protocol type are displayed using a bar chart with various colours, using the 'protocol_type' column as a source. Clear labelling and titling of the plot are included. The x-axis shows the protocol types, while the y-axis shows the count.



```python
protocol_type_counts = combined_data['protocol_type'].value_counts()

# Plotting a histogram using a bar chart to visualize the distribution of protocol types
plt.bar(protocol_type_counts.index, protocol_type_counts.values, color=['blue', 'orange', 'green'])

# Adding labels and title to the plot
plt.xlabel('Protocol Type')
plt.ylabel('Count')
plt.title('Histogram of Protocol Types')

# Displaying the plot
plt.show()
```

Figure 4: Extracting and counting occurrences of different protocol types in the 'protocol_type' column

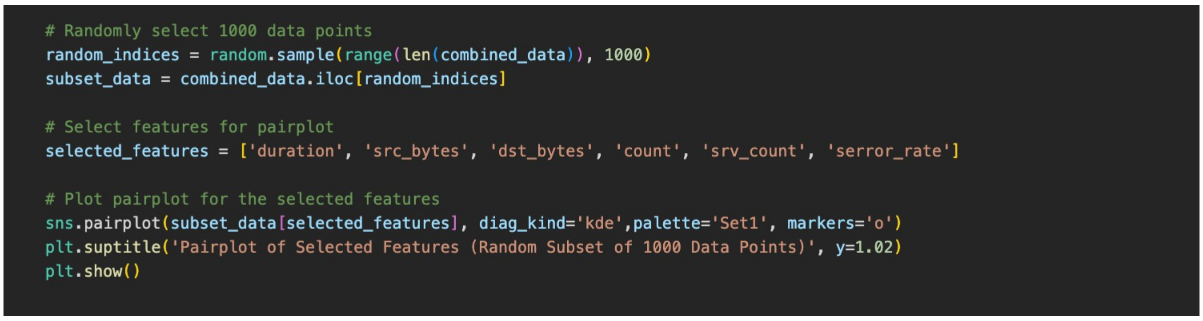
5. This code shows the distribution of protocol types ('protocol_type') from the 'combined_data' DataFrame as a countplot created with Seaborn. A set figure size is applied to the plot, and axis labels are provided for better understanding. Data exploration is improved by the countplot that is produced, which offers information about the frequency of each protocol type.



```python
# Set up a figure with a size of 16x8 inches for the upcoming plot.
plt.figure(figsize=(16, 8))

# Create a countplot using seaborn, specifying 'attack_type' on the x-axis and using data from the 'combined_data' DataFrame.
sns.countplot(x='attack_type', data=combined_data)

# Set the title of the plot to 'Distribution of Attack Types'.
plt.title('Distribution of Attack Types')

# Label the x-axis as 'Attack Type'.
plt.xlabel('Attack Type')

# Label the y-axis as 'Count'.
plt.ylabel('Count')

# Display the plot.
plt.show()
```

Figure 5: Distribution of attack types

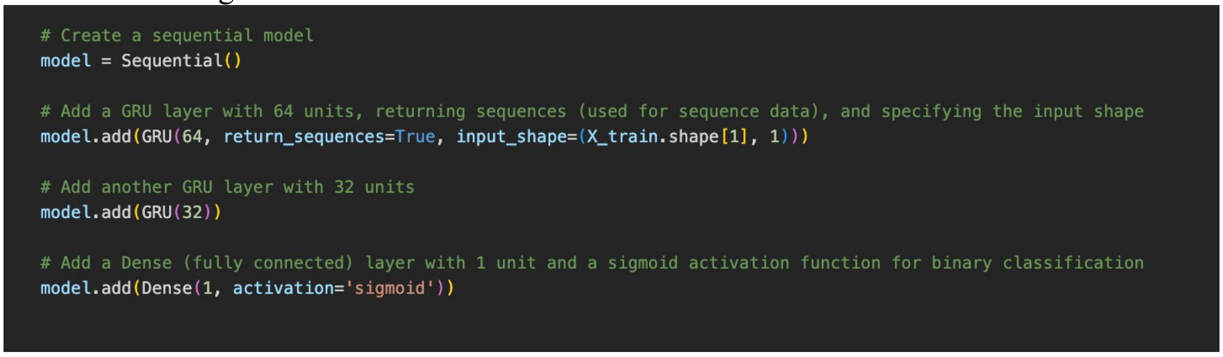
6. This code constructs a subset ('subset_data') from 1000 randomly selected data points from the 'combined_data' DataFrame. Next, given a set of chosen numerical features ('duration','src_bytes', 'dst_bytes', 'count','srv_count','serror_rate'), a pairplot is produced using Seaborn. The diagonal kernel density plots reveal information on the distribution of the individual variables, while the pairplot illustrates the relationships between these features. For clarity, we have named the final figure "Pairplot of Selected Features (Random Subset of 1000 Data Points)".



```python
# Randomly select 1000 data points
random_indices = random.sample(range(len(combined_data)), 1000)
subset_data = combined_data.iloc[random_indices]

# Select features for pairplot
selected_features = ['duration', 'src_bytes', 'dst_bytes', 'count', 'srv_count', 'serror_rate']

# Plot pairplot for the selected features
sns.pairplot(subset_data[selected_features], diag_kind='kde',palette='Set1', markers='o')
plt.suptitle('Pairplot of Selected Features (Random Subset of 1000 Data Points)', y=1.02)
plt.show()
```

Figure 6: Pairplot of selected features (Random subset of 1000 Data points)

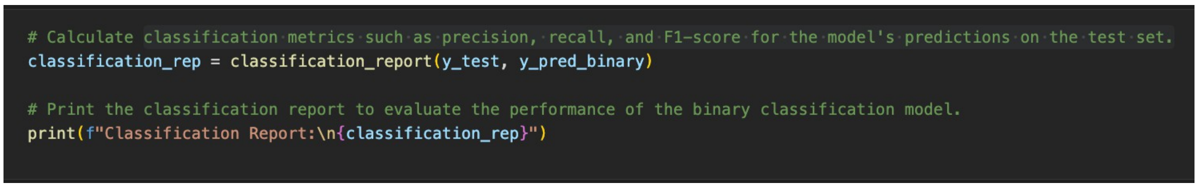
7. Model Training



```python
# Create a sequential model
model = Sequential()

# Add a GRU layer with 64 units, returning sequences (used for sequence data), and specifying the input shape
model.add(GRU(64, return_sequences=True, input_shape=(X_train.shape[1], 1)))

# Add another GRU layer with 32 units
model.add(GRU(32))

# Add a Dense (fully connected) layer with 1 unit and a sigmoid activation function for binary classification
model.add(Dense(1, activation='sigmoid'))
```

Figure 7: Building and Compiling GRU Model

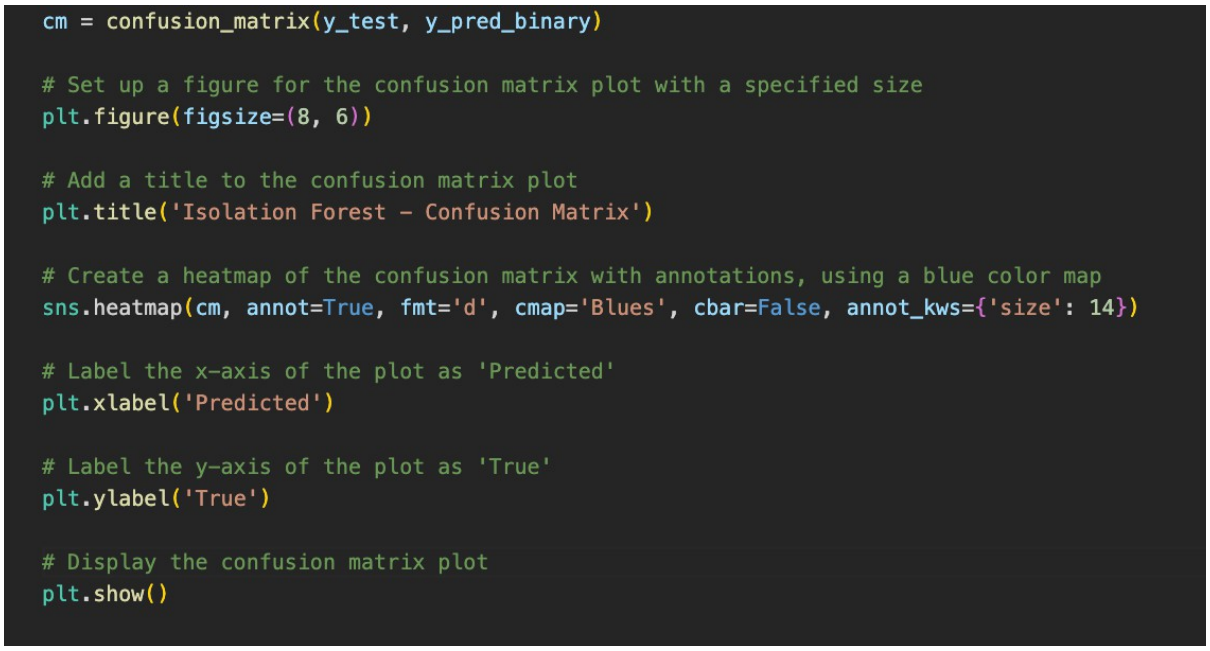
8. The true labels (y_test) and predicted labels (y_pred_binary) of a binary classification model are compared using this code to assess the model's performance. Metrics like precision, recall, F1-score, and support are included in the report for every class using the method proposed.



```python
# Calculate classification metrics such as precision, recall, and F1-score for the model's predictions on the test set.
classification_rep = classification_report(y_test, y_pred_binary)

# Print the classification report to evaluate the performance of the binary classification model.
print(f"Classification Report:\n{classification_rep}")
```

Figure 8: classification of metrics such as precision, recall, and F1-score for the model's predictions on the test set.

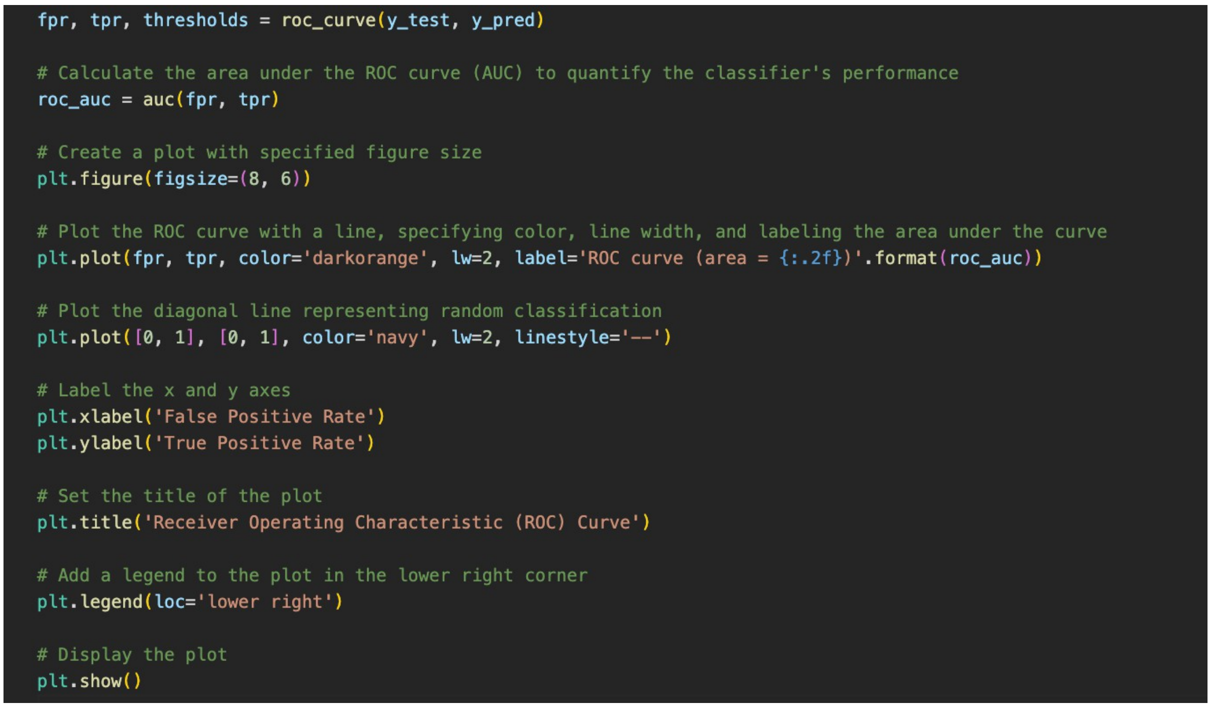
9. In order to assess how well an Isolation Forest model performs in binary classification, this code creates a confusion matrix diagram. It visualises true positive, true negative, false positive, and false negative predictions using a heatmap annotated with Seaborn. The graphic sheds light on the model's categorization errors and accuracy.



```python
cm = confusion_matrix(y_test, y_pred_binary)

# Set up a figure for the confusion matrix plot with a specified size
plt.figure(figsize=(8, 6))

# Add a title to the confusion matrix plot
plt.title('Isolation Forest - Confusion Matrix')

# Create a heatmap of the confusion matrix with annotations, using a blue color map
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, annot_kws={'size': 14})

# Label the x-axis of the plot as 'Predicted'
plt.xlabel('Predicted')

# Label the y-axis of the plot as 'True'
plt.ylabel('True')

# Display the confusion matrix plot
plt.show()
```

Figure 9: Importing the confusion_matrix function from the scikit-learn library

10. To assess a classifier's performance, this code computes and presents the Receiver Operating Characteristic (ROC) curve. It compares the False Positive Rate (FPR) to the True Positive Rate (Sensitivity) at various classification criteria. The plot shows the classifier's discriminating power, and the area under the ROC curve (AUC) assesses the classifier's overall performance.



```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Calculate the area under the ROC curve (AUC) to quantify the classifier's performance
roc_auc = auc(fpr, tpr)

# Create a plot with specified figure size
plt.figure(figsize=(8, 6))

# Plot the ROC curve with a line, specifying color, line width, and labeling the area under the curve
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))

# Plot the diagonal line representing random classification
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

# Label the x and y axes
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

# Set the title of the plot
plt.title('Receiver Operating Characteristic (ROC) Curve')

# Add a legend to the plot in the lower right corner
plt.legend(loc='lower right')

# Display the plot
plt.show()
```

Figure 10: Calculating the ROC curve values (False Positive Rate, True Positive Rate, and Thresholds) using predicted and true labels