# Configuration Manual

MSc Research Project
MSc in Cybersecurity

Sneha Sivaram
Student ID: X23192054

School of Computing
National College of Ireland

Supervisor: Prof. Liam Mccabe

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | ……. ……………………………Sneha Sivaram………………………………………………… |
| **Student ID:** | …………………………………………X23192054…………………………………………..…… |
| **Programme:** | …………Msc in Cyber Security…………………… **Year:** …………2024…….. |
| **Module:** | …………………………………………Msc Research Project……………………………………..……… |
| **Lecturer:** | ……………………………………Prof. Liam Mccabe……………………………………..……… |
| **Submission Due Date:** | ………………………………………………12/12/2024…………………………………………..……… |
| **Project Title:** | Integrating Explainable AI (XAI) for Improved Malware Detection and Analysis |
| **Word Count:** | …………………1094…………………… **Page Count:** …………………10………..……..……… |

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:** ………………………………………Sneha Sivaram…………………………………………

**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): | |

# Integrating Explainable AI (XAI) for Improved Malware Detection and Analysis

**Configuration Manual**

Sneha Sivaram

Student ID: x23192054

## 1. Introduction

The configuration manual consists of the tools and technologies used for implementing the entire research. It provides a step-by-step guide for configuring, setting up, and running the XAI-based malware detection system.

## 2. Experimental Setup

### 2.1 System Requirements

**Hardware specifications:**

- **Processor:** ARM64 or x64-based processor
- **Memory:** Minimum 8 GB RAM
- **Storage:** Minimum of 5 GB of space

**Operating System:** macOS (Catalina or later), Linux, Windows 10/11

**Internet Requirements:** A stable internet connection is necessary for installing libraries and configuring the email alert system.

### 2.2 Software Requirements

**Python environment:** Python 3.10.5

- This version was selected for compatibility with specific libraries such as (LIME, scikit-learn, etc.)

**IDE and Tools:**

- **IDE:** Visual Studio Code
- **Version:** 1.94.2

**Required Libraries:** Install the following python libraries:

- pandas
- scikit-learn
- matplotlib
- seaborn
- lime
- numpy
- joblib
- requests

## 3. Installation Guide

### 3.1 Install Python 3.10.5

- **Download Python:** Visit the python 3.10.5 download page and select the installer suitable for the operating system.
- **Install Python:** Run the downloaded installer and follow the on-screen instructions.
- **Verify Installation:** Open the terminal/cmd and type: '*python3 --version'* (Ensure it displays **Python 3.10.5.**

### 3.2 Install Visual Studio Code

- **Download VS Code:** Visit the Visual Studio code download page and download the version suitable for the desired operating system.
- **Install VS code:** Follow the installation prompts.
- **Set Up Python Extension:** Open VS code, go to extensions and install the Python extension by Microsoft.

### 3.3 Install Jupyter Notebook

- **Open Visual Studio Code:** Go to extensions search for Jupyter and install the extension.

### 3.4 Install Required Libraries

- **Open Jupyter Notebook:** Open VS code go to files, create a new file and select Jupyter Notebook.
- **Install the libraries using Jupyter Notebook:** Run this command: '*!pip install pandas scikit-learn matplotlib seaborn lime numpy joblib requests'* This will install the required libraries.

## 4. Project Setup

### 4.1 Files used

| File | Description |
|---|---|
| MalwareMemoryDump.csv | Dataset for training and testing the models. |
| XAI_MalwareDetectionProject.ipynb | Main python script for the malware detection system |
| logistic_regression_model.pkl | Saved Logistic Regression model (best model) |
| standard_scaler.pkl | Saved scaler for feature scaling. |

### 4.2 Loading the Project in VS Code

- Open VS Code and navigate to the project directory.
- Open the main Python script through jupyter notebook.
- Before Running the code select the Python interpreter and choose Python 3.10.5.

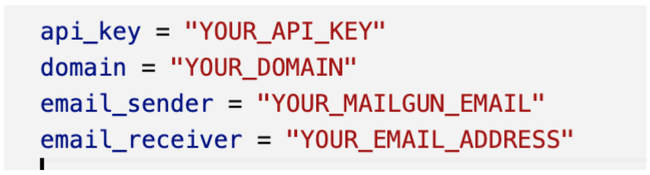
## 5. Execution Guide

### Step 1: Setting up the code

- Open the main Python script 'XAI_MalwareDetectionProject.ipynb'
- Set the file path correctly pointing to the dataset stored in the local system.

### Step 2: Email Alert System Setup

- Visit Mailgun and create an account
- Obtain your API Key and Domain Name
- Replace the credentials in the code:



```python
api_key = "YOUR_API_KEY"
domain = "YOUR_DOMAIN"
email_sender = "YOUR_MAILGUN_EMAIL"
email_receiver = "YOUR_EMAIL_ADDRESS"
```

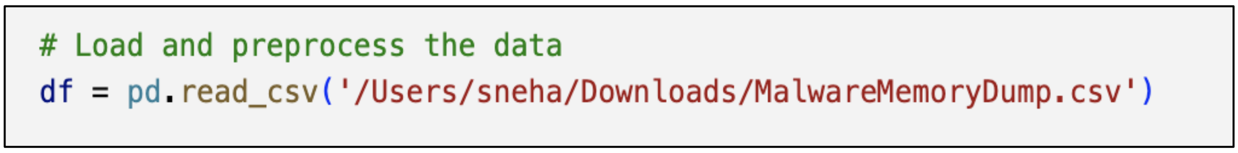
### Step 3: Activate Email Alerts

- Run the script
- When malware is detected, an alert email will sent with prediction probability and a detailed LIME explanation.

- The email alert system is set up only for the best model- Logistic Regression

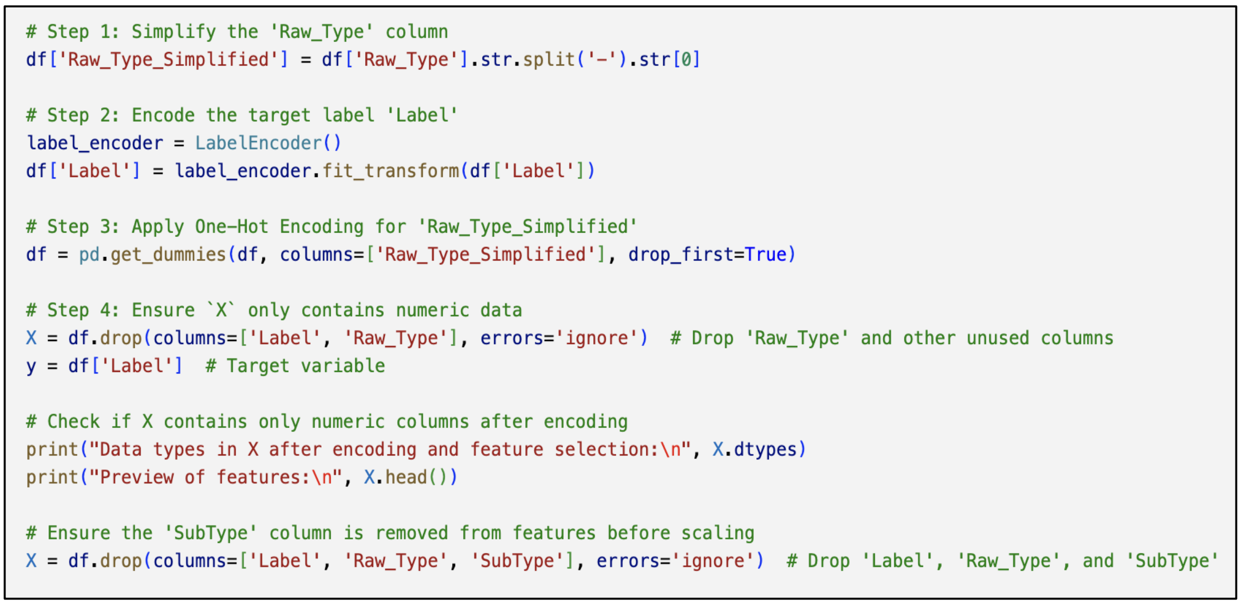
## 6. Preprocessing the Data

- The dataset MalwareMemoryDump.csv was loaded using the pandas library.
- These are the key preprocessing steps:
- Simplifying the Raw_type columns
- Encode the target label with LabelEncoder
- Applying OneHotEncoding for Raw_type_simplified features
- Ensuring only numeric data is present in the training dataset
- Removing unwanted columns



```python
# Load and preprocess the data
df = pd.read_csv('/Users/sneha/Downloads/MalwareMemoryDump.csv')
```

**Fig 2. Loading the Dataset**

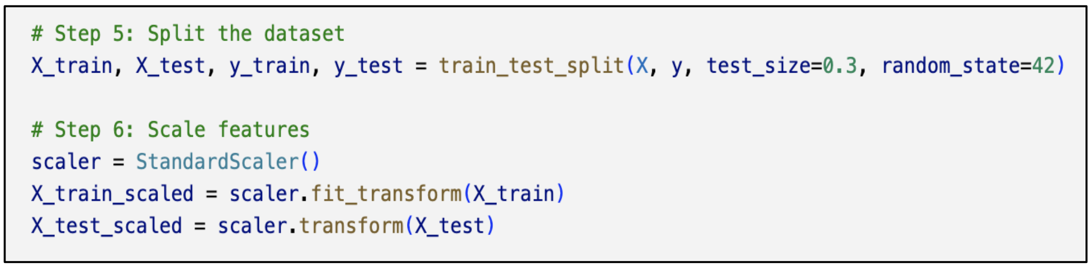


```python
# Step 1: Simplify the 'Raw_Type' column
df['Raw_Type_Simplified'] = df['Raw_Type'].str.split('-').str[0]

# Step 2: Encode the target label 'Label'
label_encoder = LabelEncoder()
df['Label'] = label_encoder.fit_transform(df['Label'])

# Step 3: Apply One-Hot Encoding for 'Raw_Type_Simplified'
df = pd.get_dummies(df, columns=['Raw_Type_Simplified'], drop_first=True)

# Step 4: Ensure `X` only contains numeric data
X = df.drop(columns=['Label', 'Raw_Type'], errors='ignore')  # Drop 'Raw_Type' and other unused columns
y = df['Label']  # Target variable

# Check if X contains only numeric columns after encoding
print("Data types in X after encoding and feature selection:\n", X.dtypes)
print("Preview of features:\n", X.head())

# Ensure the 'SubType' column is removed from features before scaling
X = df.drop(columns=['Label', 'Raw_Type', 'SubType'], errors='ignore')  # Drop 'Label', 'Raw_Type', and 'SubType'
```

**Fig 3. Data Preprocessing**

### Splitting and Scaling the data

- The dataset was split into training and testing dataset using a 70-30 split.
- Features were scaled using StandardScaler.



```python
# Step 5: Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 6: Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

**Fig 4. Splitting and scaling the dataset**

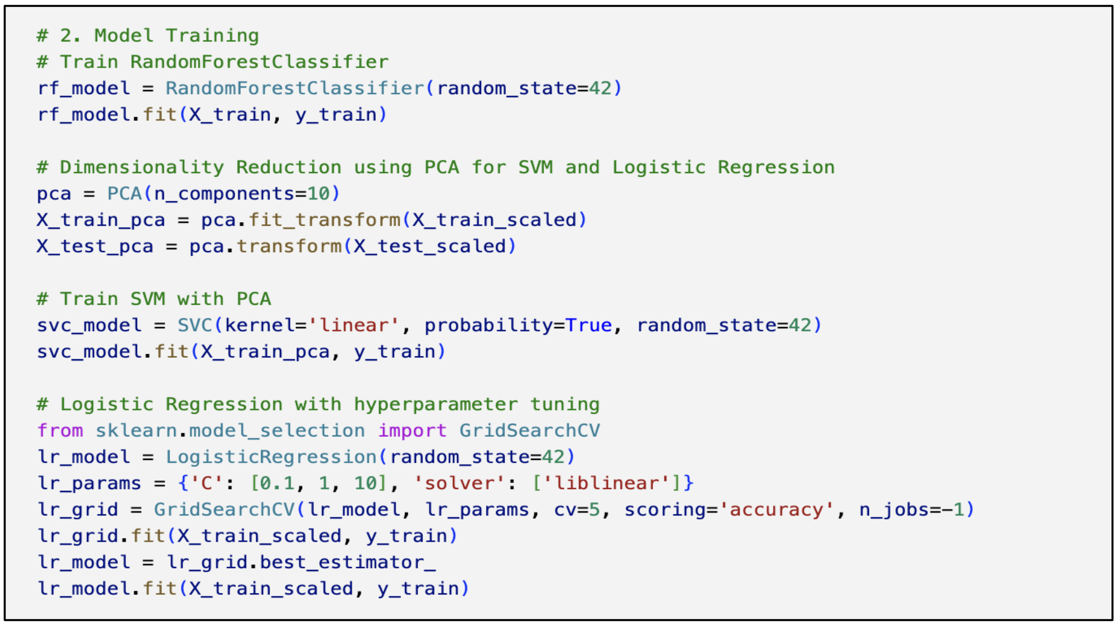
## Model Training

Three models were trained for comparison:

- **Random Forest Classifier:** Trained on the raw features

- **Support Vector Machine (SVM):** Trained after dimensionality reduction using principal component analysis (PCA).

- **Logistic Regression:** Hyperparameter tuning performed using GridSearchCV

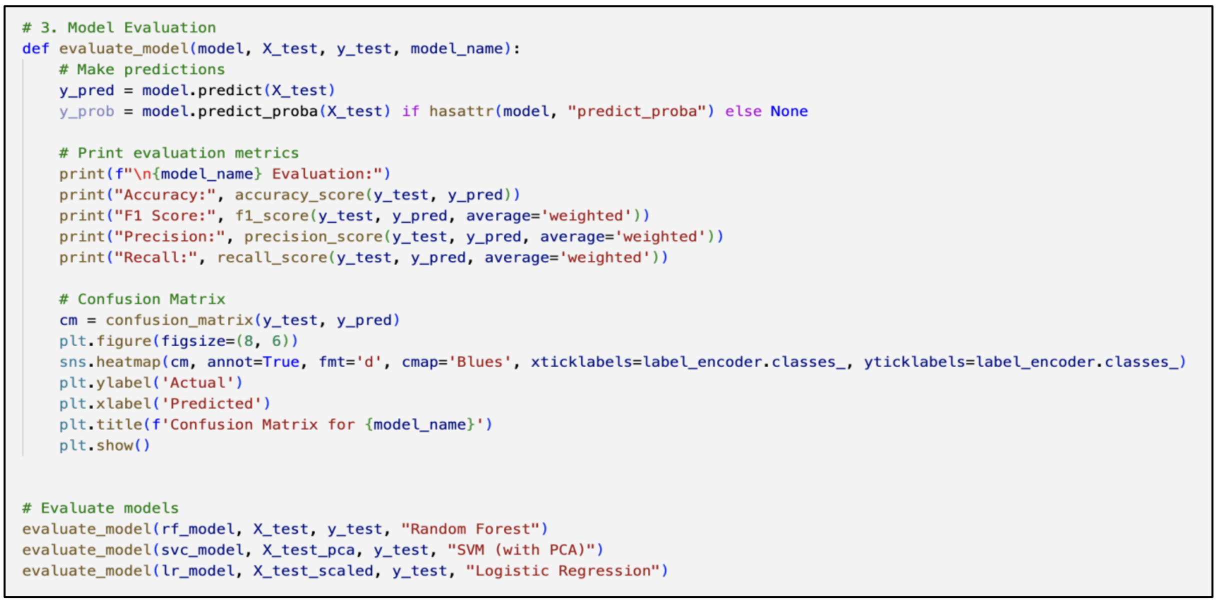


```python
# 2. Model Training
# Train RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Dimensionality Reduction using PCA for SVM and Logistic Regression
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Train SVM with PCA
svc_model = SVC(kernel='linear', probability=True, random_state=42)
svc_model.fit(X_train_pca, y_train)

# Logistic Regression with hyperparameter tuning
from sklearn.model_selection import GridSearchCV
lr_model = LogisticRegression(random_state=42)
lr_params = {'C': [0.1, 1, 10], 'solver': ['liblinear']}
lr_grid = GridSearchCV(lr_model, lr_params, cv=5, scoring='accuracy', n_jobs=-1)
lr_grid.fit(X_train_scaled, y_train)
lr_model = lr_grid.best_estimator_
lr_model.fit(X_train_scaled, y_train)
```

**Fig 5. Model Training**

## Model Evaluation

- The trained models were evaluated using metrics such as accuracy, f1 score, precision, recall and a confusion matrix.



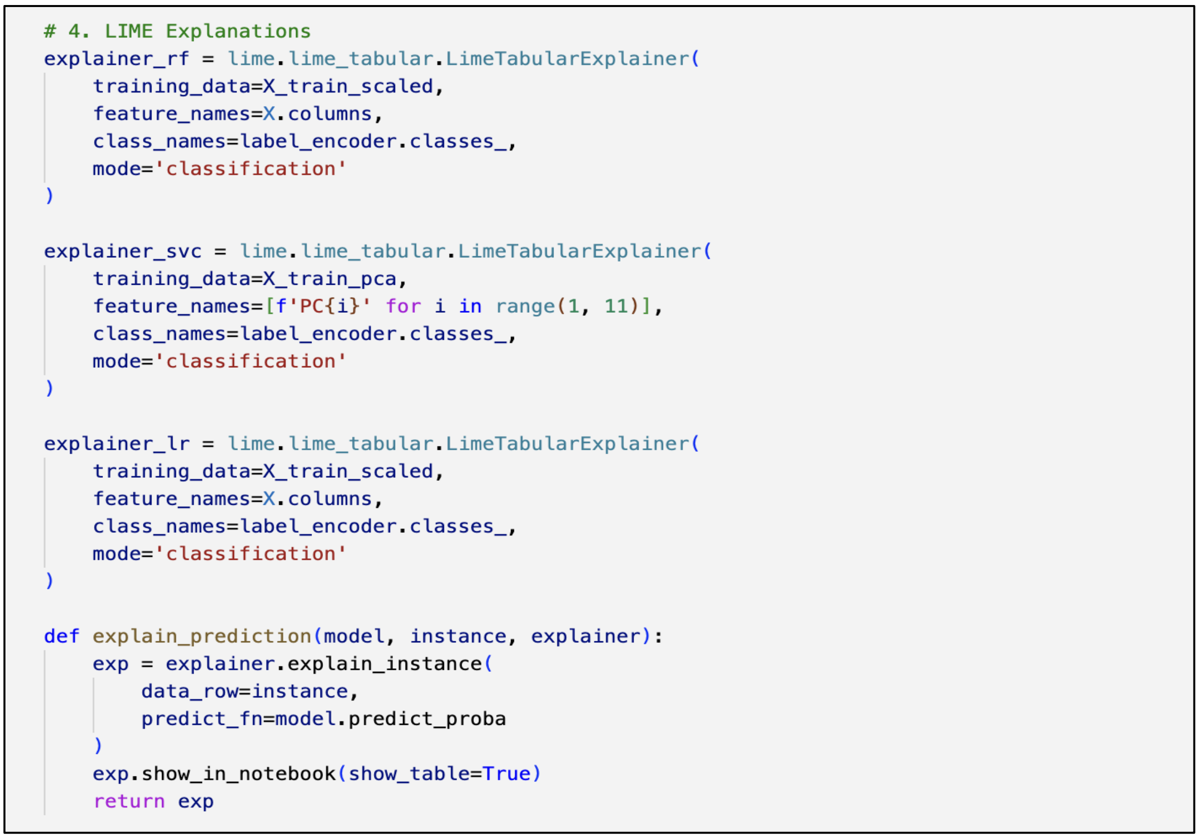
```python
# 3. Model Evaluation
def evaluate_model(model, X_test, y_test, model_name):
    # Make predictions
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test) if hasattr(model, "predict_proba") else None

    # Print evaluation metrics
    print(f"\n{model_name} Evaluation:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))
    print("Precision:", precision_score(y_test, y_pred, average='weighted'))
    print("Recall:", recall_score(y_test, y_pred, average='weighted'))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title(f'Confusion Matrix for {model_name}')
    plt.show()


# Evaluate models
evaluate_model(rf_model, X_test, y_test, "Random Forest")
evaluate_model(svc_model, X_test_pca, y_test, "SVM (with PCA)")
evaluate_model(lr_model, X_test_scaled, y_test, "Logistic Regression")
```

**Fig 6. Model Evaluation**

## 7. Explainable AI with LIME

### LIME (Local Interpretable Model-Agnostic Explanations) Explanations

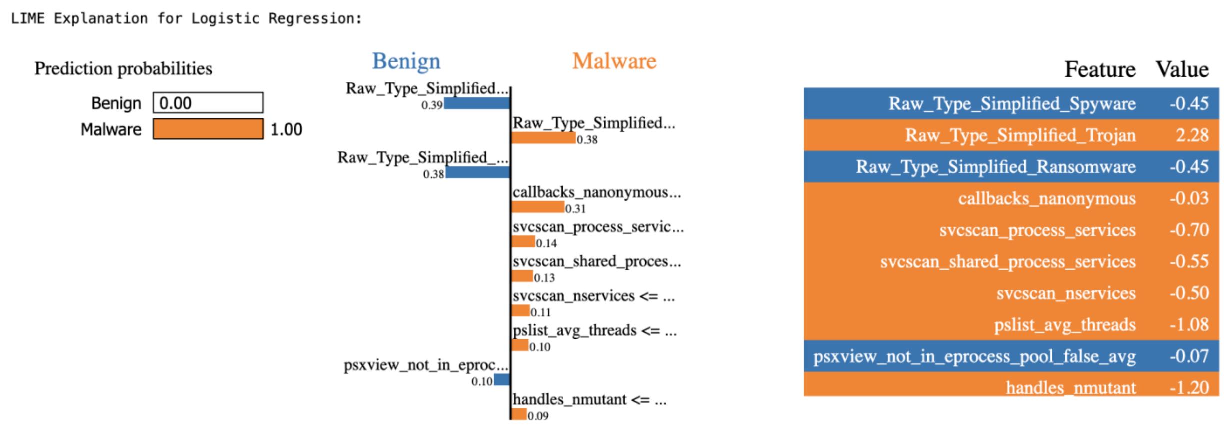
- The LIME library was used for model explanations, providing insights into predictions for individual instances.



```python
# 4. LIME Explanations
explainer_rf = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_scaled,
    feature_names=X.columns,
    class_names=label_encoder.classes_,
    mode='classification'
)

explainer_svc = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_pca,
    feature_names=[f'PC{i}' for i in range(1, 11)],
    class_names=label_encoder.classes_,
    mode='classification'
)

explainer_lr = lime.lime_tabular.LimeTabularExplainer(
    training_data=X_train_scaled,
    feature_names=X.columns,
    class_names=label_encoder.classes_,
    mode='classification'
)

def explain_prediction(model, instance, explainer):
    exp = explainer.explain_instance(
        data_row=instance,
        predict_fn=model.predict_proba
    )
    exp.show_in_notebook(show_table=True)
    return exp
```

**Fig 7. LIME integration to all models**

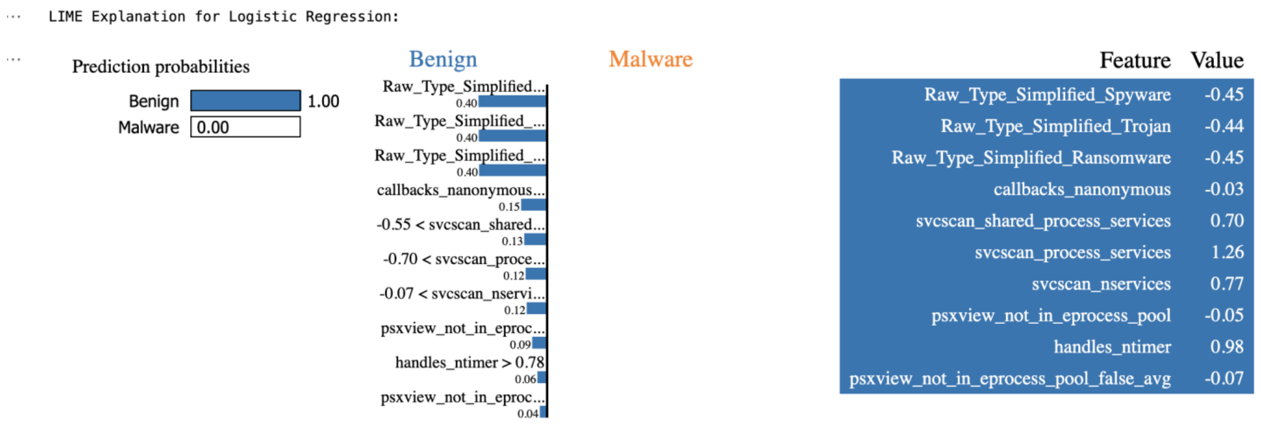
This is what the LIME visualisations look like:



```
LIME Explanation for Logistic Regression:
```

Prediction probabilities
Benign 0.00
Malware 1.00
Benign
Malware
Raw_Type_Simplified... 0.39
Raw_Type_Simplified... 0.38
Raw_Type_Simplified_... 0.38
callbacks_nanonymous... 0.31
svcscan_process_servic... 0.14
svcscan_shared_proces... 0.13
svcscan_nservices <= ... 0.11
pslist_avg_threads <= ... 0.10
psxview_not_in_eproc... 0.10
handles_nmutant <= ... 0.09

| Feature | Value |
|---|---|
| Raw_Type_Simplified_Spyware | -0.45 |
| Raw_Type_Simplified_Trojan | 2.28 |
| Raw_Type_Simplified_Ransomware | -0.45 |
| callbacks_nanonymous | -0.03 |
| svcscan_process_services | -0.70 |
| svcscan_shared_process_services | -0.55 |
| svcscan_nservices | -0.50 |
| pslist_avg_threads | -1.08 |
| psxview_not_in_eprocess_pool_false_avg | -0.07 |
| handles_nmutant | -1.20 |



```
...   LIME Explanation for Logistic Regression:
...
```

Prediction probabilities
Benign 1.00
Malware 0.00
Benign
Malware
Raw_Type_Simplified... 0.40
Raw_Type_Simplified_... 0.40
Raw_Type_Simplified_... 0.40
callbacks_nanonymous... 0.15
-0.55 < svcscan_shared... 0.13
-0.70 < svcscan_proce... 0.12
-0.07 < svcscan_nservi... 0.12
psxview_not_in_eproc... 0.09
handles_ntimer > 0.78 0.06
psxview_not_in_eproc... 0.04

| Feature | Value |
|---|---|
| Raw_Type_Simplified_Spyware | -0.45 |
| Raw_Type_Simplified_Trojan | -0.44 |
| Raw_Type_Simplified_Ransomware | -0.45 |
| callbacks_nanonymous | -0.03 |
| svcscan_shared_process_services | 0.70 |
| svcscan_process_services | 1.26 |
| svcscan_nservices | 0.77 |
| psxview_not_in_eprocess_pool | -0.05 |
| handles_ntimer | 0.98 |
| psxview_not_in_eprocess_pool_false_avg | -0.07 |

## 8. Conclusion

This manual provides all the steps to set up and run the malware detection system effectively. It covers the software and hardware required and the detailed instructions to execute the code. This guide also highlights the use of LIME for a better understanding of the model's predictions and includes an email alert system for sending alert notifications of potential threats.

The system is designed to work with Python 3.10.5 for compatibility. Future updates may improve for newer versions.

## REFERENCES

Visual Studio Code, (n.d.). *Visual Studio Code*. Available at: https://code.visualstudio.com/ (Accessed: 2 December 2024).

Microsoft, (n.d.). *Jupyter - Visual Studio Code Marketplace*. Available at: https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter (Accessed: 2 December 2024).

Python Software Foundation (2022) *Python 3.10.5*. Available at: https://www.python.org/downloads/release/python-3105/ (Accessed: 2 December 2024)

Mailgun (n.d.) *Email API: Send transaction or marketing emails effortlessly*. Available at: https://www.mailgun.com/products/send/email-api/ (Accessed: 2 December 2024)