National College of Ireland

Configuration Manual

MSc Research Project
MSc Cyber Security

Jatinder Singh Saini
Student ID: x21173656

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jatinder Singh Saini
Student ID:	x21173656
Programme:	MSc Cyber Security **Year:** 2023/24
Module:	Research Project
Lecturer:	Diego Lugones
Submission Due Date:	31/01/2024
Project Title:	Enhancing Malware Detection in PE Files Using Hybrid Ensemble Learning Techniques
Word Count:	957 **Page Count:** 16

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: Jatinder Singh Saini

Date: 30/01/2024

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

Jatinder Singh Saini
X21173656

1 Introduction

Using hybrid ensemble learning algorithms i.e. combining Random Forest, Gradient Boosting, and AdaBoost, into a stacking classifier, this configuration manual describes the suggested work and model used to detect malware in PE files. When it came to improving malware detection systems, the suggested technique worked well. This document contains the specifics of the setup that was used for this study. Hardware and software specs, as well as the environments and languages used, are detailed in this document.

2 System Configurations

2.1 Hardware Setup

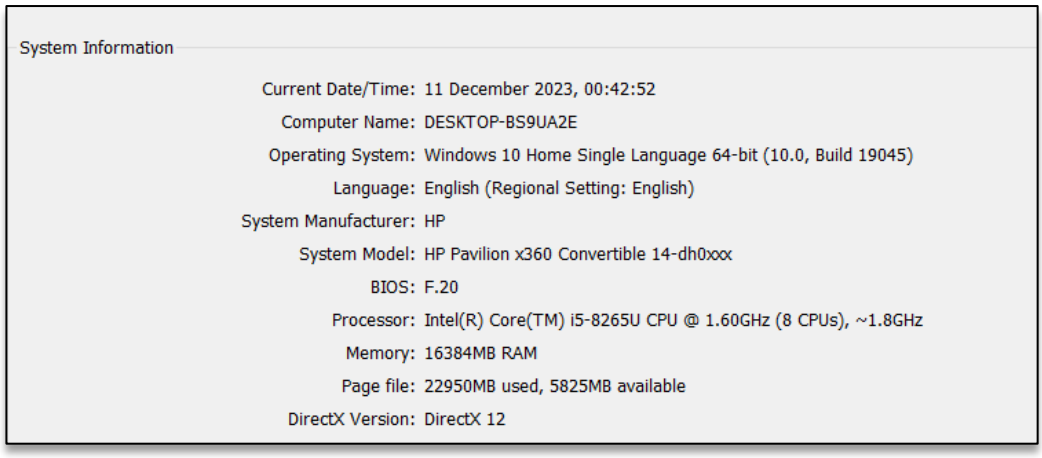

Figure 1: System Config

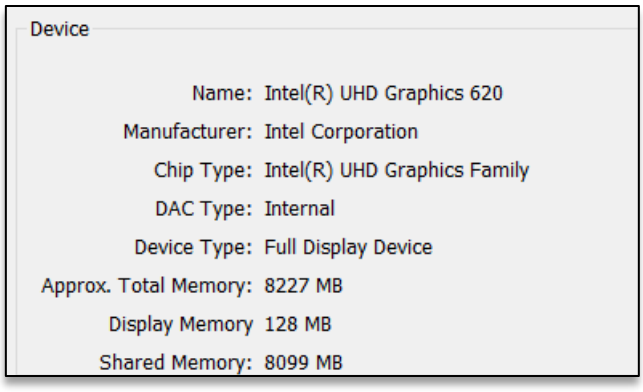

Figure 2: GPU Memory

2.2 Software Setup

The following Software Setup is needed to install and successfully execute the research project:

1. **Python - 3.10.9:** Python is an interpreted language that is both flexible and fast. It is great for scripting and for integrating components in a way that makes code clear and logical across projects [1].

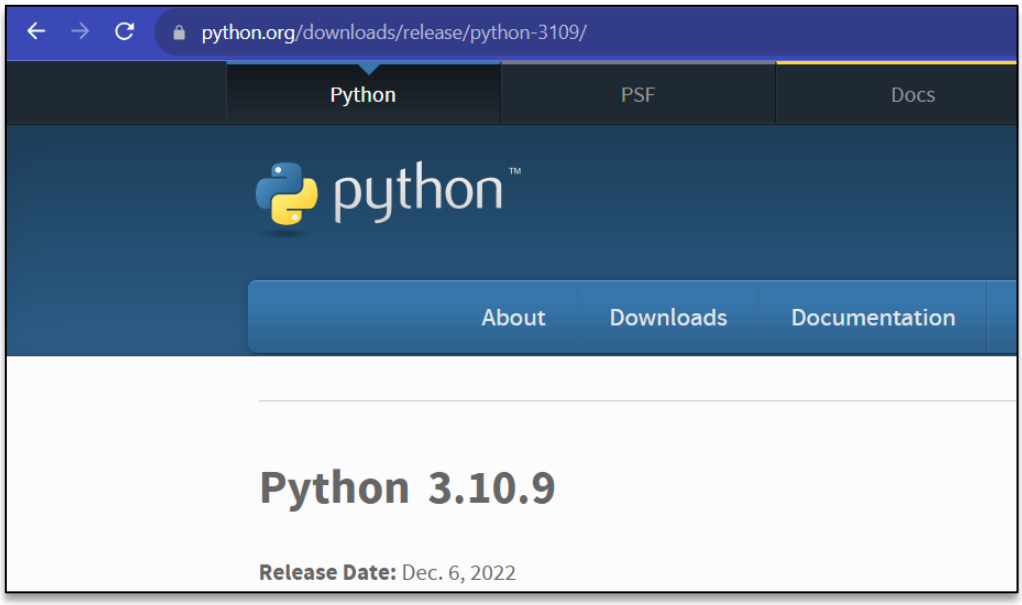

Figure 3: Python version used

2. **Visual Studio Code:** Microsoft VS code is quite a popular text editor and can be used on Windows, Linux, and macOS. It is an open-source and free editor, which was used for developing the application in this project

[1] https://www.python.org/downloads/release/python-3109/

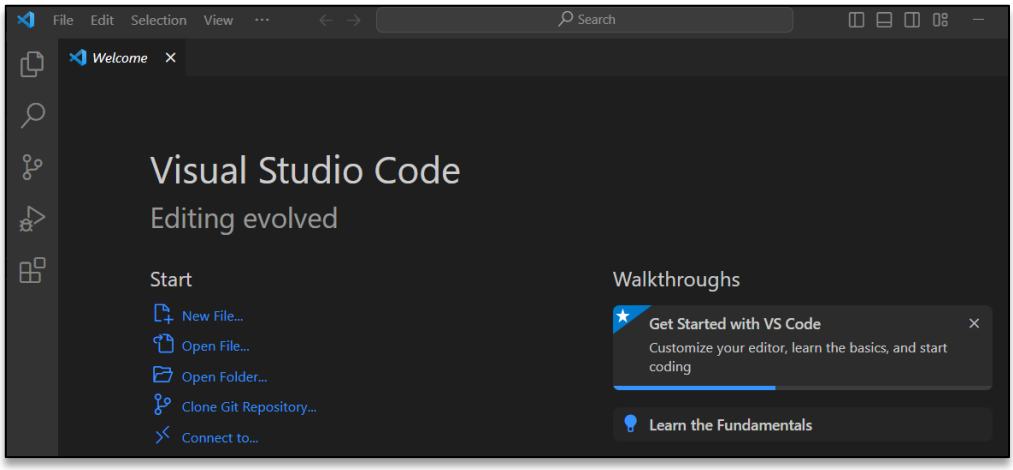

Figure 4: Visual Studio code editor

3. **Google Colab**: Python coders may easily use GPUs through Google Colab, a free cloud-based tool for interactive and collaborative coding.

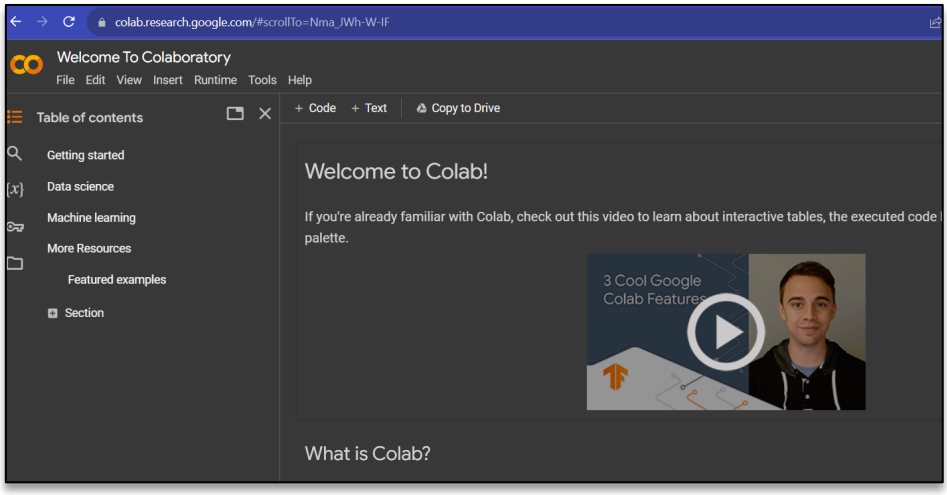

Figure 5: Interface of Google Colab

4. **Libraries Configuration:**

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Plotly
- Scikit-learn
- Pefile
- Flask
- imblearn

3 Implementation

The section contains detailed instructions to run the project on any Windows system.

1. Browse the (Google Colab[2]) URL: https://colab.research.google.com, the following interface will be displayed, as shown in "Figure 6".

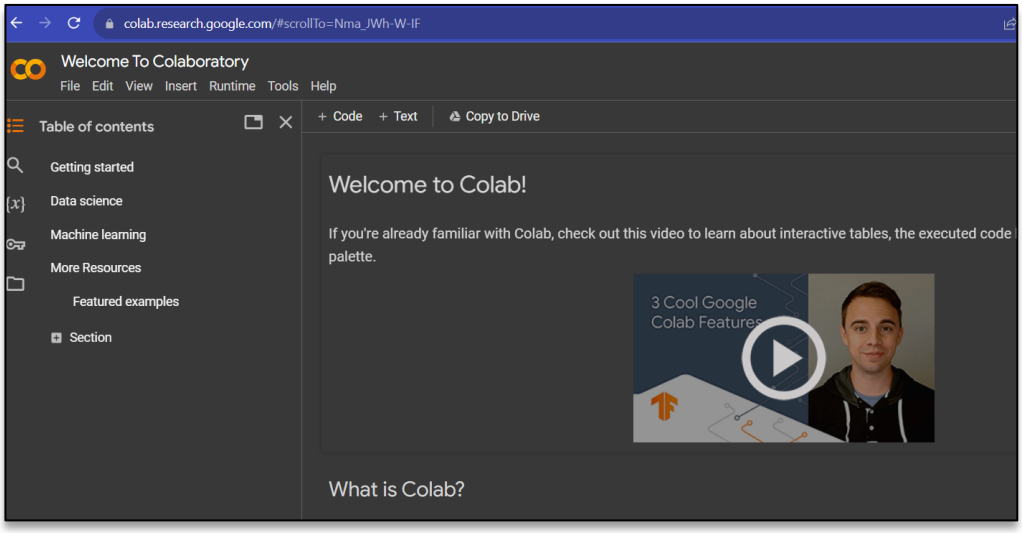

Figure 6: Google Colaboratory Interface

2. To run the code, the necessary libraries are imported into the project, as shown in "Figure 7".

```
#importing all libraries
import os
import six
import sys
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
import plotly.figure_factory as ff
from sklearn import preprocessing
from sklearn import ensemble
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
sys.modules['sklearn.externals.six'] = six
from mlxtend.classifier import StackingClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier, GradientBoostingClassifier
from sklearn.metrics import confusion_matrix, classification_report , accuracy_score
from sklearn.feature_selection import mutual_info_classif, SelectFromModel, SelectPercentile
import warnings
warnings.filterwarnings("ignore")
```

Figure 7: Libraries

3. The dataset was downloaded, imported into the notebook, and read using the panda's library before the code could be run, as shown in "Figure 8".

[2] https://colab.research.google.com/

```
#loading dataset file
df = pd.read_csv('/content/drive/MyDrive/pe_malware_classification/Data/dataset_malwares.csv')
df.head()
```

Figure 8: Dataset loading

4. Selecting only necessary columns based on the "Pefile" library helps to enhance the model efficiency and interpretability in malware classification, as shown in "Figure 9".

```
#only taking below columns for training(removed unnecessary columns)
df = df[['e_magic', 'e_cblp', 'e_cp', 'e_crlc', 'e_cparhdr', 'e_minalloc',
       'e_maxalloc', 'e_ss', 'e_sp', 'e_csum', 'e_ip', 'e_cs', 'e_lfarlc',
       'e_ovno', 'e_oemid', 'e_oeminfo', 'e_lfanew', 'Machine',
       'NumberOfSections', 'TimeDateStamp', 'PointerToSymbolTable',
       'NumberOfSymbols', 'SizeOfOptionalHeader', 'Characteristics', 'Magic',
       'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode',
       'SizeOfInitializedData', 'SizeOfUninitializedData',
       'AddressOfEntryPoint', 'BaseOfCode', 'ImageBase', 'SectionAlignment',
       'FileAlignment', 'MajorOperatingSystemVersion',
       'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion',
       'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfHeaders',
       'CheckSum', 'SizeOfImage', 'Subsystem', 'DllCharacteristics',
       'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve',
       'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'Malware']]
df.head()
```

Figure 9: Taking on necessary columns

5. "Figure 10" shows, the dataset shape and information for data cleaning.

```
[ ] #shape of dataset
    df.shape

    (19611, 53)

    #information of datatype
    df.info()

    Data columns (total 53 columns):
     #   Column                 Non-Null Count  Dtype
    ---  ------                 --------------  -----
     0   e_magic                19611 non-null  int64
     1   e_cblp                 19611 non-null  int64
     2   e_cp                   19611 non-null  int64
     3   e_crlc                 19611 non-null  int64
     4   e_cparhdr              19611 non-null  int64
     5   e_minalloc             19611 non-null  int64
     6   e_maxalloc             19611 non-null  int64
     7   e_ss                   19611 non-null  int64
     8   e_sp                   19611 non-null  int64
     9   e_csum                 19611 non-null  int64
```

Figure 10: Info of dataset

6. Verifying that the dataset does not include any null values. The dataset used in this research does not contain any null values. This is shown in the below "Figure 11".

```
#checking null values in the dataset
df.isna().sum()
```

```
e_magic                 0
e_cblp                  0
e_cp                    0
e_crlc                  0
e_cparhdr               0
e_minalloc              0
e_maxalloc              0
e_ss                    0
e_sp                    0
e_csum                  0
e_ip                    0
e_cs                    0
e_lfarlc                0
e_ovno                  0
e_oemid                 0
e_oeminfo               0
e_lfanew                0
Machine                 0
```

Figure 11: Checking null values in the dataset

7. Displaying if any unique values are present in each column, as shown in “Figure 12”.

```
#number of unique value in each columns
for i in df.columns:
  print(i,df[i].unique().size)
```

```
e_magic 1
e_cblp 39
e_cp 43
e_crlc 26
e_cparhdr 22
e_minalloc 20
e_maxalloc 17
e_ss 13
e_sp 26
e_csum 16
e_ip 28
e_cs 23
e_lfarlc 16
e_ovno 14
e_oemid 87
e_oeminfo 96
```

Figure 12: Displaying unique values in column

8. “Figure 13” the data set is imbalanced which might not result in better performance of the models. The below figure shows the distribution of samples i.e. (malware and no malware).

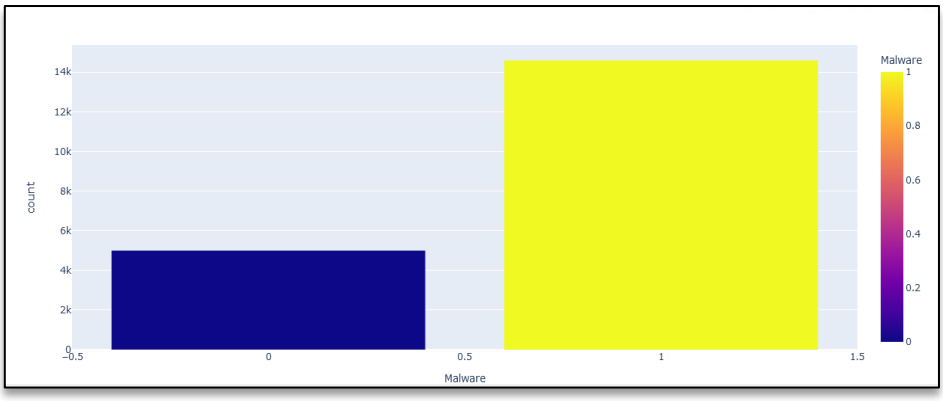

Figure 13: Distribution of samples in the dataset

9. "Figure 14" shows the data set is imbalanced, so to balance the dataset SMOTE () function is used. This results in removing unwanted noise in the data set and making it easier for a model to perform well once the training is completed.

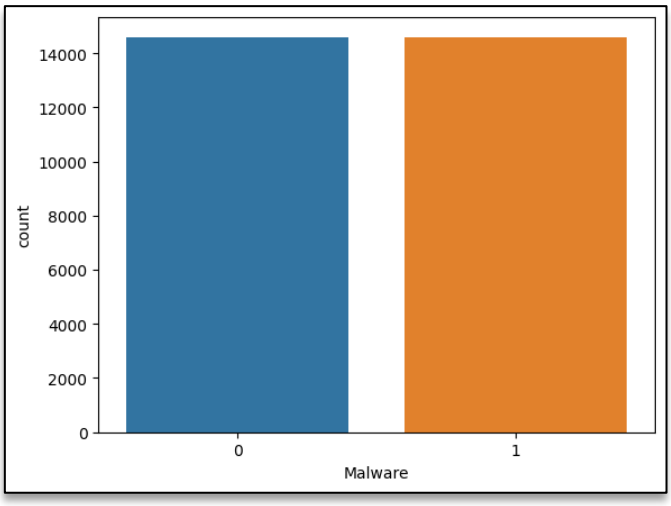

Figure 14: Balanced Dataset

10. Splitting dataset for model's training and testing phase, so that class distribution in the target variable is maintained in both sets. In this project, the dataset is split into an 80:20 ratio as shown in the below "Figure 15".

```
#splitting data into train and test with ratio of 80:20
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,stratify=y)
```

Figure 15: Splitting the dataset

11. The code for model building and evaluation metrics for the classification of samples (malware or benign) are shown below in "Figure 16,17,18,19":

AdaBoost Classifier

```
adb_model = AdaBoostClassifier(n_estimators=4)
adbclf = adb_model
adbclf.fit(X_train,y_train)
y_pred = adbclf.predict(X_test)
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlOrBr" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

Figure 16: Model building and evaluation metrices for AdaBoost

Gradient Boosting Classifier

```
gbc_model = GradientBoostingClassifier(n_estimators=8)
gbc = gbc_model
gbc.fit(X_train,y_train)
y_pred = gbc.predict(X_test)
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlOrBr" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

Figure 17: Model building and evaluation metrices for Gradient Boosting

Random Forest Classifier

```
#RandomForest Classifier
rf_clf = RandomForestClassifier(n_estimators=2, criterion='gini', max_depth=3,)
rf = rf_clf
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlOrBr" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

Figure 18: Model building and evaluation metrices for Random Forest

Stacking Classifier

```
#Stacking classifier
clf = StackingClassifier(classifiers = [RandomForestClassifier(),GradientBoostingClassifier()], meta_classifier = AdaBoostClassifier())
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
#accuracy score
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlOrBr" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```

Figure 19: Model building and evaluation metrices for Stacking Classifier

Correlation Matrix: A correlation is a measure of the degree to which two variables are directly related to one another. We can predict one variable from another through relationships. Below “Figure 20” shows the correlation matrix.

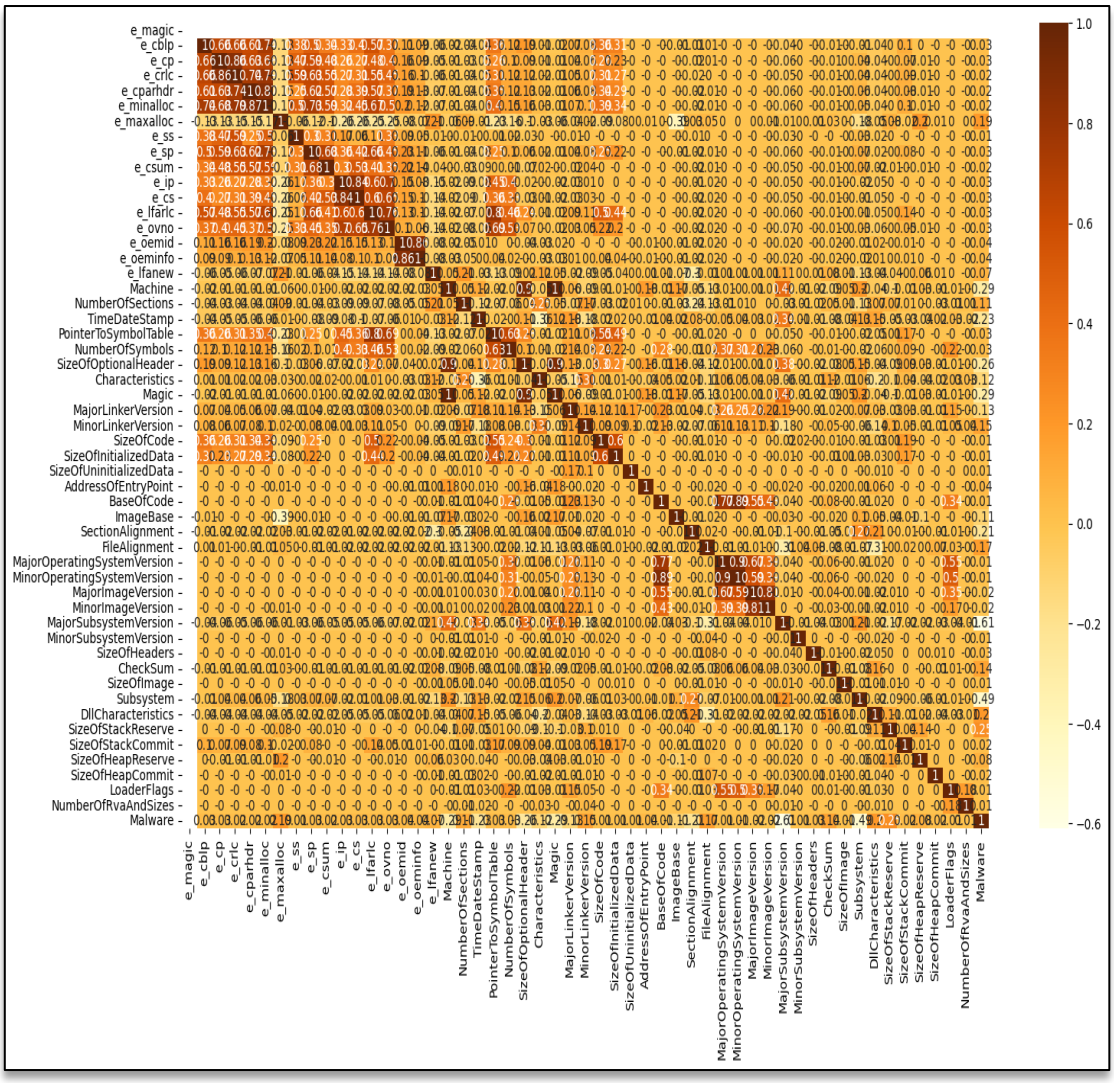

Figure 20: Correlation Matrix

Feature Importance Bar Chart for Data Visualization Only: This is used in the project to get an idea of which feature, or column is contributing more to enhancement of the model performance. The feature importance is shown in below “Figure 21”.

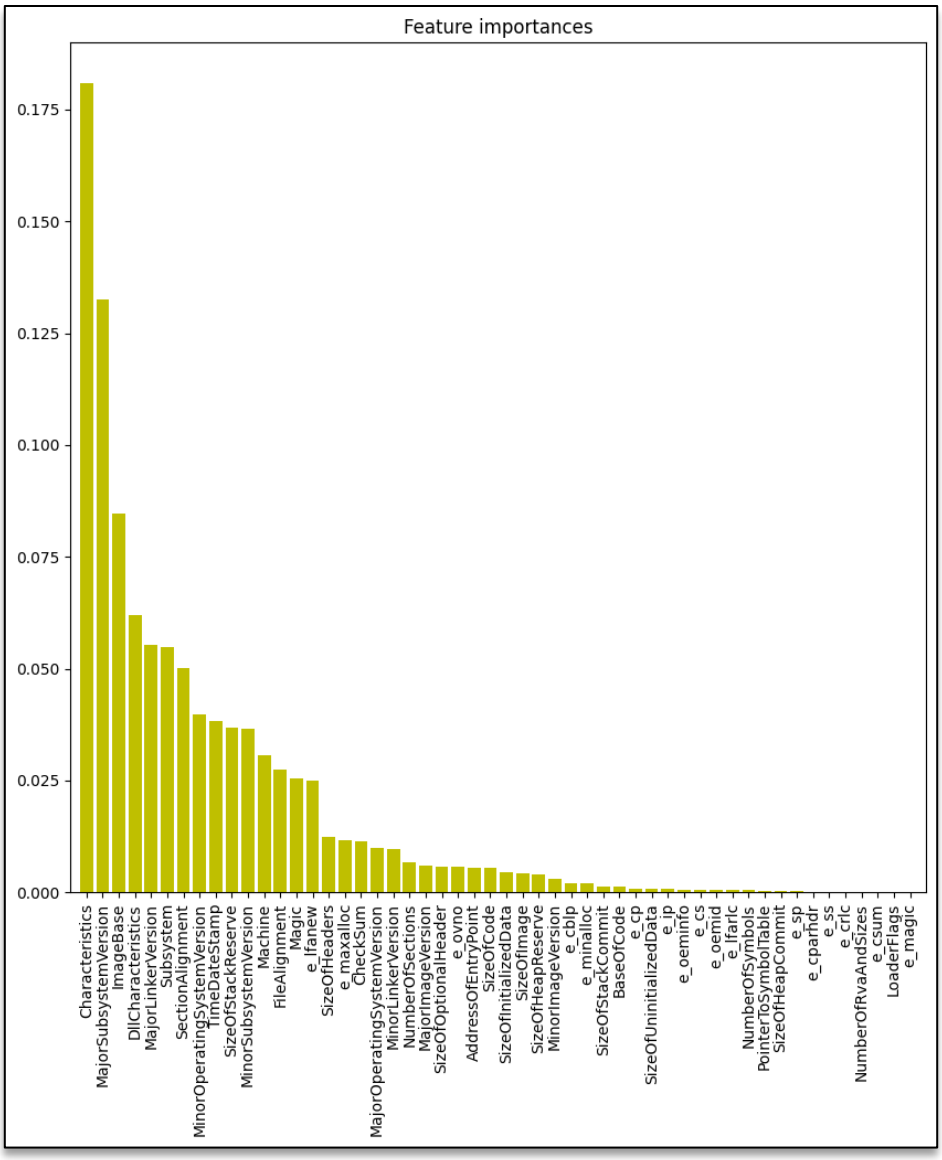

Figure 21: Feature Importance

4 Confusion Matrix, Classification Report and Web Application (Running on Local Machine)

The below results show how the model has performed:

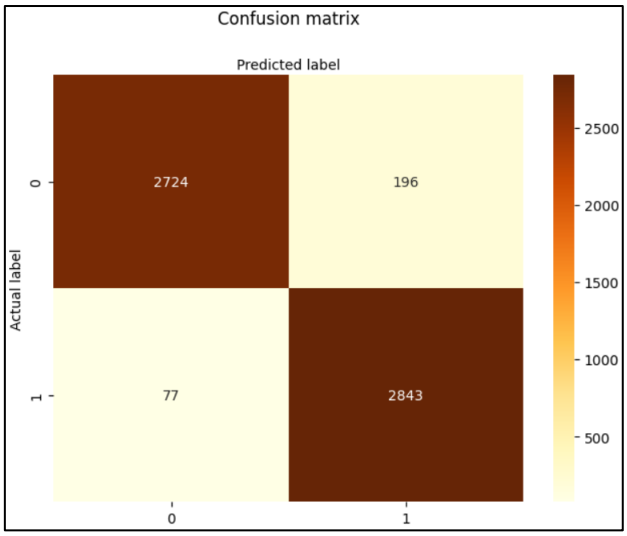

Figure 22: Confusion matrix (CM) for AdaBoost

	precision	recall	f1-score	support
0	0.97	0.93	0.95	2920
1	0.94	0.97	0.95	2920
accuracy			0.95	5840
macro avg	0.95	0.95	0.95	5840
weighted avg	0.95	0.95	0.95	5840

Figure 23: Classification Report (CLF) for AdaBoost

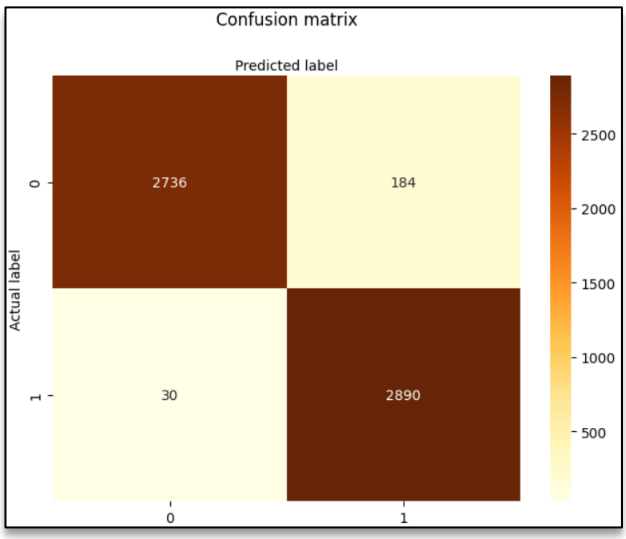

Figure 24: CM for Gradient Boosting

	precision	recall	f1-score	support
0	0.99	0.94	0.96	2920
1	0.94	0.99	0.96	2920
accuracy			0.96	5840
macro avg	0.96	0.96	0.96	5840
weighted avg	0.96	0.96	0.96	5840

Figure 25: CLF for Gradient Boosting

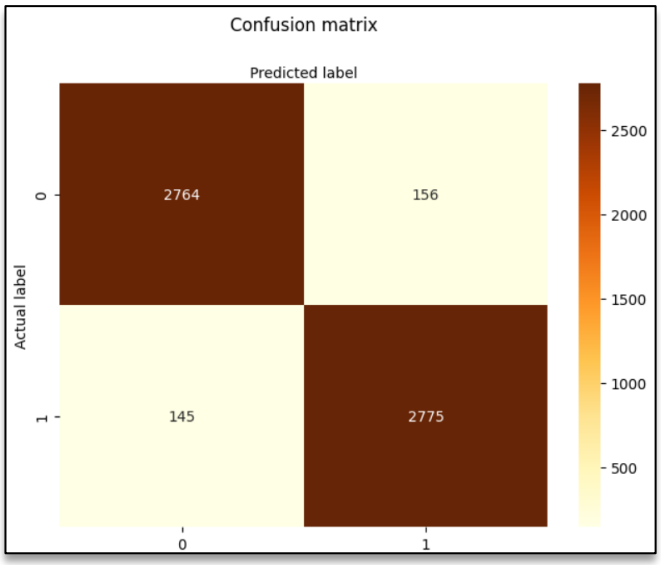

Figure 26: CM for Random Forest (RF)

	precision	recall	f1-score	support
0	0.95	0.95	0.95	2920
1	0.95	0.95	0.95	2920
accuracy			0.95	5840
macro avg	0.95	0.95	0.95	5840
weighted avg	0.95	0.95	0.95	5840

Figure 27: CLF for RF

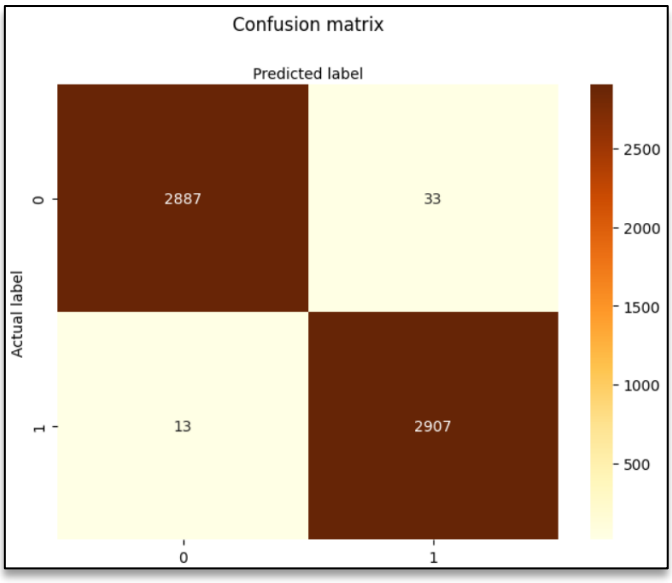

Figure 28: CM for Stacking Classifier

	precision	recall	f1-score	support
0	1.00	0.99	0.99	2920
1	0.99	1.00	0.99	2920
accuracy			0.99	5840
macro avg	0.99	0.99	0.99	5840
weighted avg	0.99	0.99	0.99	5840

Figure 29: CLF for Stacking Classifier

The stacking classifier achieves a 99% accuracy rate by combining Gradient Boosting, Random Forest, and AdaBoost classifiers.

Web Application:

The “Figure 30” shows the code for the application. The models after the implementation were integrated into the web application and run using the VS code editor.

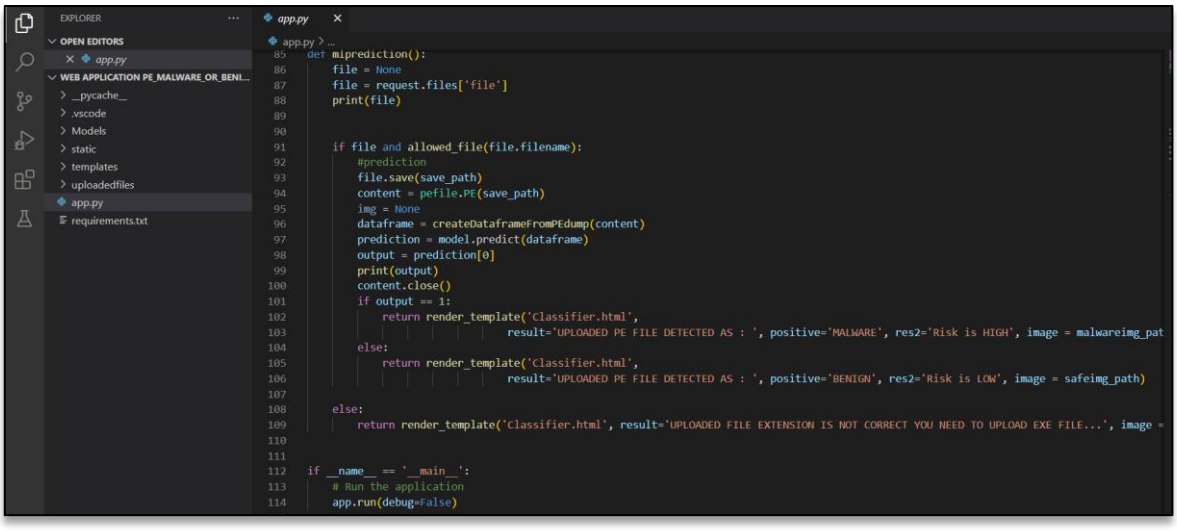

Figure 30: Web Application Code

"Figure 31" shows the interface of the application. For the classification of the model whether it is malware or safe file, click on the "Pe File Classifier" button.

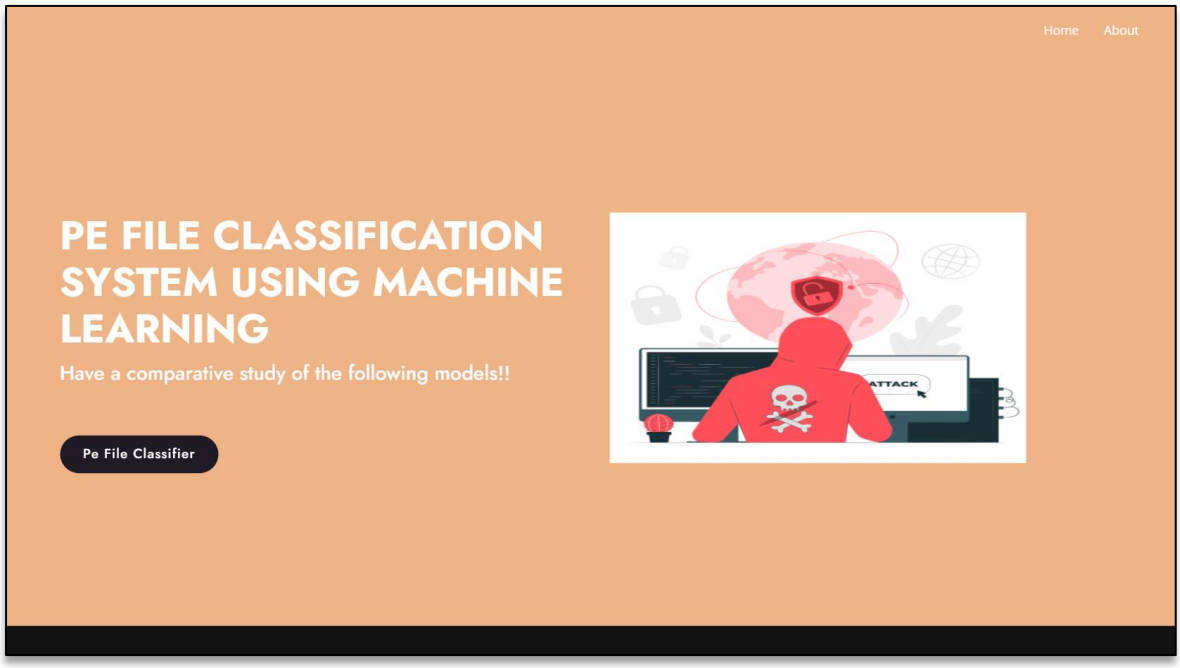

Figure 31: Web application interface

Now, the classification can be done using the PE file sample by providing the sample file to the application to check the risk of the file. If the file that is provided to the application is malicious, then the application provides the following output, as shown in "Figure 32" and if the file is safe, the application displays the output, as shown in "Figure 33".

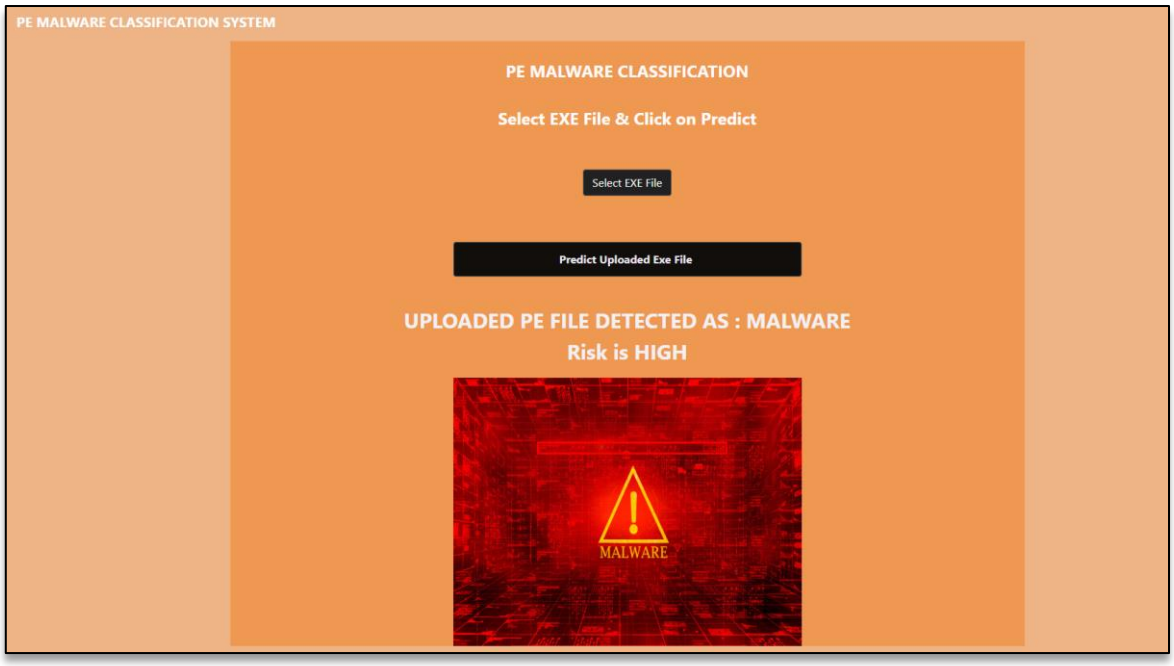

Figure 32: The file is detected as malware

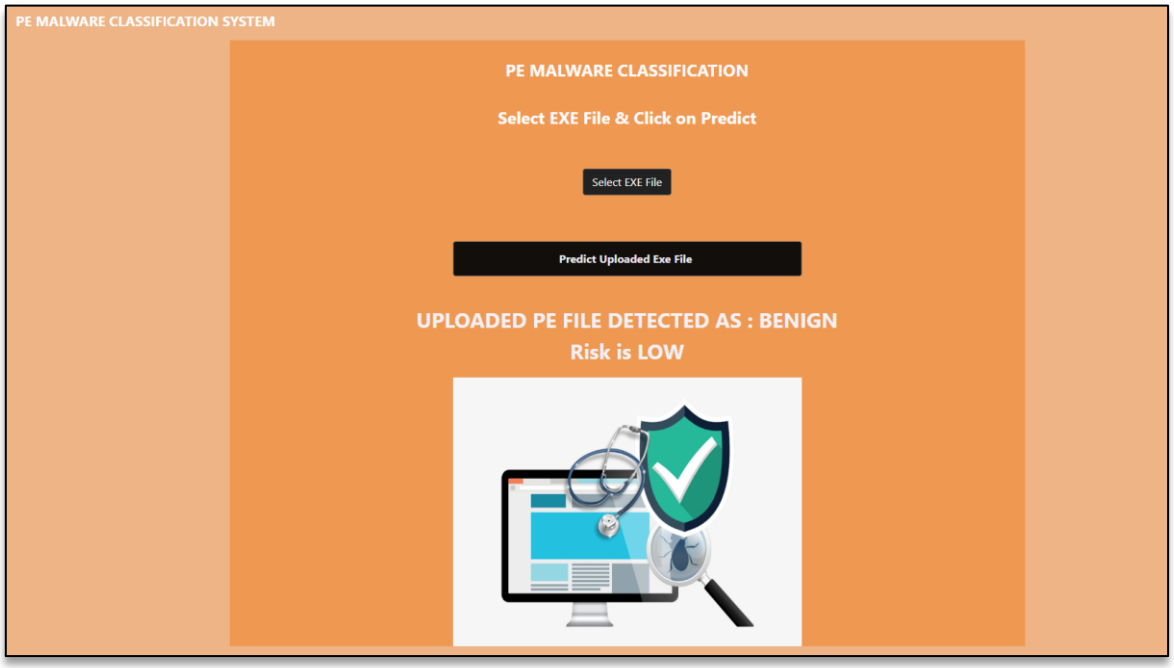

Figure 33: The file is detected as benign