Configuration Manual

MSc Practicum 2
Master of Science in Cybersecurity

Prasanth Sriramulu Deenadayala Babu
Student ID: 23173459

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Prasanth Sriramulu Deenadayala Babu
Student ID:	23173459
Programme:	MSc Cyber Security **Year:** Jan 2024
Module:	MSc Practicum part 2
Lecturer:	Vikas Sahni
Submission Due Date:	12-12-2024
Project Title:	Improving Network and IoT Intrusion Detection Through Machine Learning Algorithms
Word Count:	840 **Page Count:** 14

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: Prasanth Sriramulu Deenadayala Babu

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

Prasanth Sriramulu Deenadayala Babu
Student ID: 23173459

1 Introduction

This Configuration Manual presents the detailed depiction of steps and procedures used in detecting intrusions in IoT systems. It is by a trial of the experimental setup, that is this document illustrates all the necessary software settings and tools.

2 System Configuration

The specification of the system used for this practicum is:

- Operating System: Windows 11, 64-bit operating system
- Processor: AMD Ryzen 3 3250U
- RAM: 8 GB
- Hard drive: 512GB SSD

3 Software Tools

A comprehensive description of tools used for this practicum:

- Integrated Development Environment (IDE): Google Colaboratory (Google colab)
- Coding language: Python 3.10
- Data verification tool: Microsoft excel
- Storage: PC/Google Drive
- Connectivity: Stable internet connectivity for using Google colab (cloud-based IDE)

4 Implementation

To implement this project, python libraries and their version used for evaluating the dataset:

- Scikit-Learn - 1.3.1【6】.
- NumPy - 1.26.0【6】【9】.
- Seaborn - 0.12.3【6】.
- Pandas - 2.1.3【7】【8】.
- Matplotlib - 3.8.1【6】.
- Imblearn - 0.12.4【7】【9】.
- confusion_matrix
- accuracy_score
- Label Encoder
- Standard Scaler
- Random Forest Classifier
- correlation_matrix

- Naive Bayes
- Decision Tree
- K-Nearest Neighbors
- Logistic Regression

5 Execution of Network Intrusion Detection Dataset:

1. All the analyses, visualizations, and models in this paper had the Python libraries imported.

```python
# import relevant modules
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import imblearn

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
import sys # import the sys module
np.set_printoptions(threshold=sys.maxsize) # Use sys.maxsize to print the whole array
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

Fig 1: Importing the libraries

2. Load the data on the notebook

```python
[ ] train = pd.read_csv("Train_data.csv")
    test = pd.read_csv("Test_data.csv")
```

Fig 2: Importing the dataset

```python
[ ] # Descriptive statistics
    train.describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	root_shell
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.00000	25192.000000	25192.000000	25192.000000	25192.000000	25192.000000
mean	305.054104	2.433063e+04	3.491847e+03	0.000079	0.023738	0.00004	0.198039	0.001191	0.394768	0.227850	0.001548
std	2686.555640	2.410805e+06	8.883072e+04	0.008910	0.260221	0.00630	2.154202	0.045418	0.488811	10.417352	0.039316
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	2.790000e+02	5.302500e+02	0.000000	0.000000	0.00000	0.000000	0.000000	1.000000	0.000000	0.000000
max	42862.000000	3.817091e+08	5.151385e+06	1.000000	3.000000	1.00000	77.000000	4.000000	1.000000	884.000000	1.000000

Fig 3: Training the dataset

3. Finding the class of distributions in the dataset.

```
[ ] # Attack Class Distribution
    train['class'].value_counts()
```

class	count
normal	13449
anomaly	11743

dtype: int64

Fig 4: Attack Class Distribution

4. Implementing Scaling numerical method

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance
cols = train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(test.select_dtypes(include=['float64','int64']))

# turn the result back to a dataframe
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)
```

Fig 5: Importing Standard Scaler

5. Enabling Label Encoder

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

# extract categorical attributes from both training and test sets
cattrain = train.select_dtypes(include=['object']).copy()
cattest = test.select_dtypes(include=['object']).copy()

# encode the categorical attributes
traincat = cattrain.apply(encoder.fit_transform)
testcat = cattest.apply(encoder.fit_transform)

# separate target column from encoded data
enctrain = traincat.drop(['class'], axis=1)
cat_Ytrain = traincat[['class']].copy()
```

Fig 6: Executing Label Encoder

```
train_x = pd.concat([sc_traindf,enctrain],axis=1)
train_y = train['class']
train_x.shape

(25192, 40)

[ ] test_df = pd.concat([sc_testdf,testcat],axis=1)
    test_df.shape

(22544, 40)
```

Fig 7: Test and Train Value of class

6. Executing the Feature Selection for required model

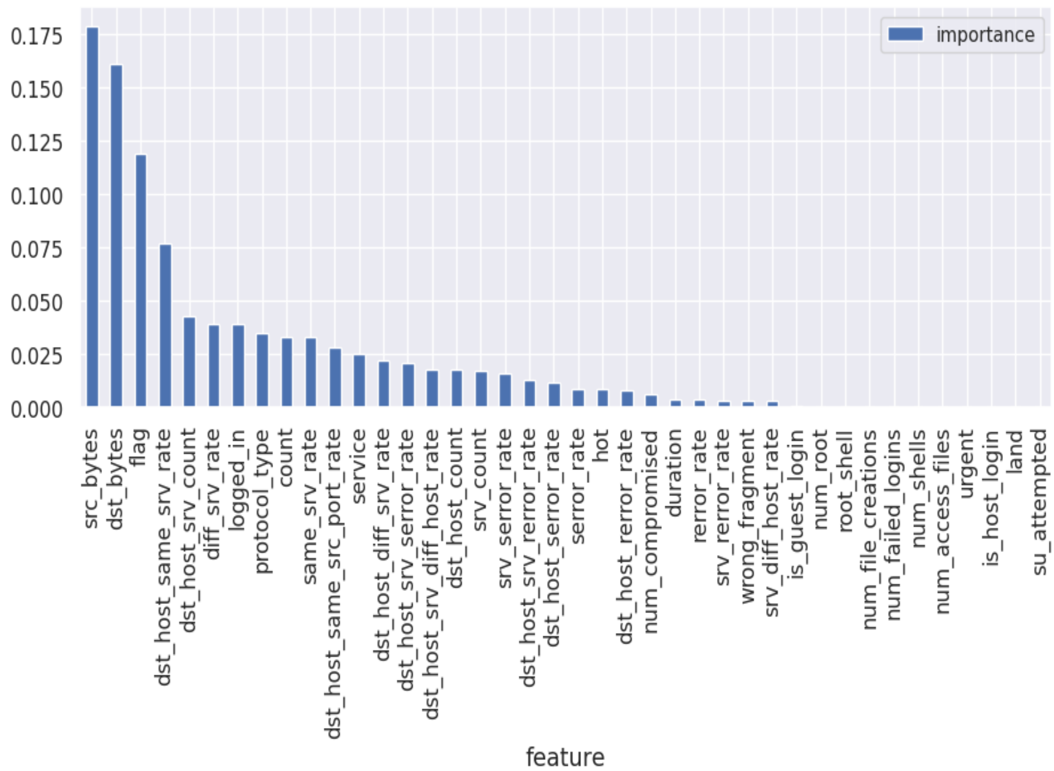

Fig 8: Importance Feature Selection

```
[ ] from sklearn.feature_selection import RFE
    import itertools
    rfc = RandomForestClassifier()

    # create the RFE model and select 10 attributes
    rfe = RFE(rfc, n_features_to_select=15)
    rfe = rfe.fit(train_x, train_y)

    # summarize the selection of the attributes
    feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), train_x.columns)]
    selected_features = [v for i, v in feature_map if i==True]

    selected_features

    ['src_bytes',
     'dst_bytes',
     'logged_in',
     'count',
     'srv_count',
     'same_srv_rate',
     'diff_srv_rate',
     'dst_host_srv_count',
     'dst_host_same_srv_rate',
     'dst_host_diff_srv_rate',
     'dst_host_same_src_port_rate',
     'dst_host_srv_diff_host_rate',
     'protocol_type',
     'service',
     'flag']
```

Fig 9: Selected Feature of dataset

7. Data partition for Test and Train value at 70% and 30%

```
[ ] from sklearn.model_selection import train_test_split

    X_train,X_test,Y_train,Y_test = train_test_split(train_x,train_y,train_size=0.70, random_state=2)
```

Fig 10: Splitting of Test and Train

8. The following show the different machine learning algorithms used in my model

```
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

Figure 11: Importing Model selection

```
# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
DTC_Classifier.fit(X_train, Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Fig 12: Train and Test results with Model selection

9. Evaluating with the selected model

```
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
```

```
# List of models
models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))
```

```
============================== Naive Baye Classifier Model Evaluation ==============================
Cross Validation Mean Score:
 0.9071666840303904
Model Accuracy:
 0.9071679709651809
Confusion matrix:
 [[7000 1245]
 [ 392 8997]]
Classification report:
               precision    recall  f1-score   support

     anomaly       0.95      0.85      0.90      8245
      normal       0.88      0.96      0.92      9389

    accuracy                           0.91     17634
   macro avg       0.91      0.90      0.91     17634
weighted avg       0.91      0.91      0.91     17634
```

```
============================== Decision Tree Classifier Model Evaluation ==============================
Cross Validation Mean Score:
 0.9960869883971739
Model Accuracy:
 1.0
Confusion matrix:
 [[8245    0]
 [   0 9389]]
Classification report:
               precision    recall  f1-score   support

     anomaly       1.00      1.00      1.00      8245
      normal       1.00      1.00      1.00      9389

    accuracy                           1.00     17634
   macro avg       1.00      1.00      1.00     17634
weighted avg       1.00      1.00      1.00     17634
```

```
============================== KNeighborsClassifier Model Evaluation ==============================
Cross Validation Mean Score:
 0.9914370153431007
Model Accuracy:
 0.9937620505840989
Confusion matrix:
 [[8168   77]
 [  33 9356]]
Classification report:
               precision    recall  f1-score   support

     anomaly       1.00      0.99      0.99      8245
      normal       0.99      1.00      0.99      9389

    accuracy                           0.99     17634
   macro avg       0.99      0.99      0.99     17634
weighted avg       0.99      0.99      0.99     17634
```

```
============================== LogisticRegression Model Evaluation ==============================
Cross Validation Mean Score:
 0.9538955835690297
Model Accuracy:
 0.9553703073607803
Confusion matrix:
 [[7764  481]
 [ 306 9083]]
Classification report:
               precision    recall  f1-score   support

     anomaly       0.96      0.94      0.95      8245
      normal       0.95      0.97      0.96      9389

    accuracy                           0.96     17634
   macro avg       0.96      0.95      0.96     17634
weighted avg       0.96      0.96      0.96     17634
```

Fig 13: Results of Model selection

6 Execution of IoT Device Network Logs Dataset:

1. For all analyses, visualizations, and models in this paper, libraries should be installed.

```
[ ]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
```

Fig 14: Importing the libraries to be used

2. Load the dataset on the notebook

```
[ ]  # Load the dataset
     file_path = 'Preprocessed_data.csv'
     data = pd.read_csv(file_path)
```

Fig 15: Importing and the dataset to the notebook

```
[ ]  print("Available columns in the dataset:")
     print(data.columns)

     Available columns in the dataset:
     Index(['frame.number', 'frame.time', 'frame.len', 'eth.src', 'eth.dst',
            'ip.src', 'ip.dst', 'ip.proto', 'ip.len', 'tcp.len', 'tcp.srcport',
            'tcp.dstport', 'Value', 'normality'],
           dtype='object')
```

Fig 16: Checking the Dataset columns

3. Data analysis and Visualization

```
# Display the first few rows of the dataset
print(data.head())

   frame.number       frame.time  frame.len          eth.src          eth.dst  \
0             1  123722736684743         54   87971959760497  167275820076079
1             2  123722736773147         62   87971959760497  167275820076079
2             3  123722736824792         62  167275820076079   87971959760497
3             4  123722736836228         54  167275820076079   87971959760497
4             5  123722749684991         54   87971959760497  167275820076079

       ip.src      ip.dst  ip.proto  ip.len  tcp.len  tcp.srcport  \
0   192168035  1921680121       6.0    40.0      0.0      49279.0
1   192168035  1921680121       6.0    48.0      0.0      56521.0
2  1921680121   192168035       6.0    48.0      0.0         80.0
3  1921680121   192168035       6.0    40.0      0.0         80.0
4   192168035  1921680121       6.0    40.0      0.0      56521.0

   tcp.dstport  Value  normality
0         80.0  -99.0          0
1         80.0  -99.0          0
2      56521.0  -99.0          0
3      49279.0  -99.0          0
4         80.0  -99.0          0
```

Fig 17: Top 5 records of attacks on dataset

4. Identifying the missing value and finding value of columns and rows

```
       frame.number    frame.time      frame.len       eth.src       eth.dst  \
count  477426.000000  4.774260e+05  477426.000000  4.774260e+05  4.774260e+05
mean    52917.357471  1.256618e+14     120.658661  1.294121e+14  1.607822e+14
std     32439.729155  2.064214e+12      88.273425  4.478838e+13  5.072488e+13
min         1.000000  1.237227e+14      42.000000  3.755968e+13  1.101089e+12
25%     27547.000000  1.243387e+14      42.000000  8.797196e+13  1.399112e+14
50%     47328.500000  1.249082e+14      98.000000  1.104254e+14  1.672758e+14
75%     78486.000000  1.256493e+14     176.000000  1.672758e+14  1.672758e+14
max    125158.000000  1.305135e+14    3484.000000  2.070699e+14  2.814750e+14

             ip.src        ip.dst       ip.proto         ip.len  \
count  4.774260e+05  4.774260e+05  477426.000000  477426.000000
mean   8.622713e+08  1.206924e+09       2.858263      97.242800
std    2.019765e+09  3.675777e+09       3.284435      97.168551
min    0.000000e+00  0.000000e+00      -1.000000       0.000000
25%    0.000000e+00  0.000000e+00      -1.000000       0.000000
50%    1.921680e+08  1.921680e+09       6.000000      84.000000
75%    1.921680e+09  1.921680e+09       6.000000     162.000000
max    1.722172e+11  2.552553e+11      17.000000    3470.000000

             tcp.len    tcp.srcport    tcp.dstport         Value  \
count  477426.000000  477426.000000  477426.000000  4.774260e+05
mean       60.844678   23722.759349    4528.371894  5.649960e-01
std        87.682770   27906.683645   15426.452846  3.188912e+03
min         0.000000       0.000000       0.000000 -9.900000e+01
25%         0.000000       0.000000       0.000000 -5.000000e+00
50%         0.000000       0.000000       0.000000 -3.000000e+00
75%       110.000000   55068.000000      80.000000 -2.000000e+00
max      3418.000000   65534.000000   65534.000000  2.202219e+06

           normality
count  477426.000000
mean        2.489808
std         1.706533
min         0.000000
25%         1.000000
50%         2.000000
75%         4.000000
max         5.000000
```

Fig 18: Determing value of dataset

5. Finding the class of distributions in the dataset.

normality	
1	82285
4	79052
0	79035
5	79032
2	79020
3	79002

dtype: int64

Fig 19: Attack Class Distribution

6. Handling missing and dropping irrelevent values

```
[ ] # Drop irrelevant columns (assuming columns like 'frame.time', 'eth.src', etc., are identifiers)
    columns_to_drop = ['frame.time', 'eth.src', 'eth.dst', 'frame.number']
    data_cleaned = data.drop(columns=columns_to_drop, errors='ignore')

    # Handle missing values by filling them with the median of each column
    data_cleaned.fillna(data_cleaned.median(), inplace=True)

    # Separate features (X) and labels (y)
    # Exclude the 'Attack_Type' column and 'normality' as labels for now
    X = data_cleaned.drop(columns=['Attack_Type', 'normality'], errors='ignore')
```

Fig 20: Data Cleaning

7. Implementing Scaling numerical method

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler

    # Standardize numeric features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Convert back to DataFrame for easier understanding
    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

```
# Step 2: Prepare features (X) and target (y)
X = data.drop(columns=['normality', 'Attack_Type'], errors='ignore')  # Features
y = data['normality']  # Target variable

# Step 3: Split dataset into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Fig 21: Data splitting into 70% of training and 30% of testing

8. Executing the feature selection of correlation matrix for turning model to better performance.

```
# Step 2: Compute the correlation matrix
correlation_matrix = data_cleaned.corr()

# Step 3: Visualize the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

```python
# Step 5: Drop features with high multicollinearity
def drop_highly_correlated_features(correlation_matrix, threshold=0.8):
    """Drop one of each pair of features with correlation higher than the threshold."""
    columns_to_drop = set()
    for i in range(len(correlation_matrix.columns)):
        for j in range(i):
            if abs(correlation_matrix.iloc[i, j]) > threshold:
                colname = correlation_matrix.columns[i]
                columns_to_drop.add(colname)
    return list(columns_to_drop)

high_corr_features = drop_highly_correlated_features(correlation_matrix)
print("Features to drop due to high multicollinearity:", high_corr_features)
```

```
Features to drop due to high multicollinearity: ['tcp.len', 'tcp.srcport', 'ip.len']
```

```python
# Step 6: Final feature set after selection
final_features = [col for col in selected_features if col not in high_corr_features]
print("Final Selected Features:", final_features)
```

```
Final Selected Features: ['frame.len', 'ip.proto', 'tcp.dstport']
```

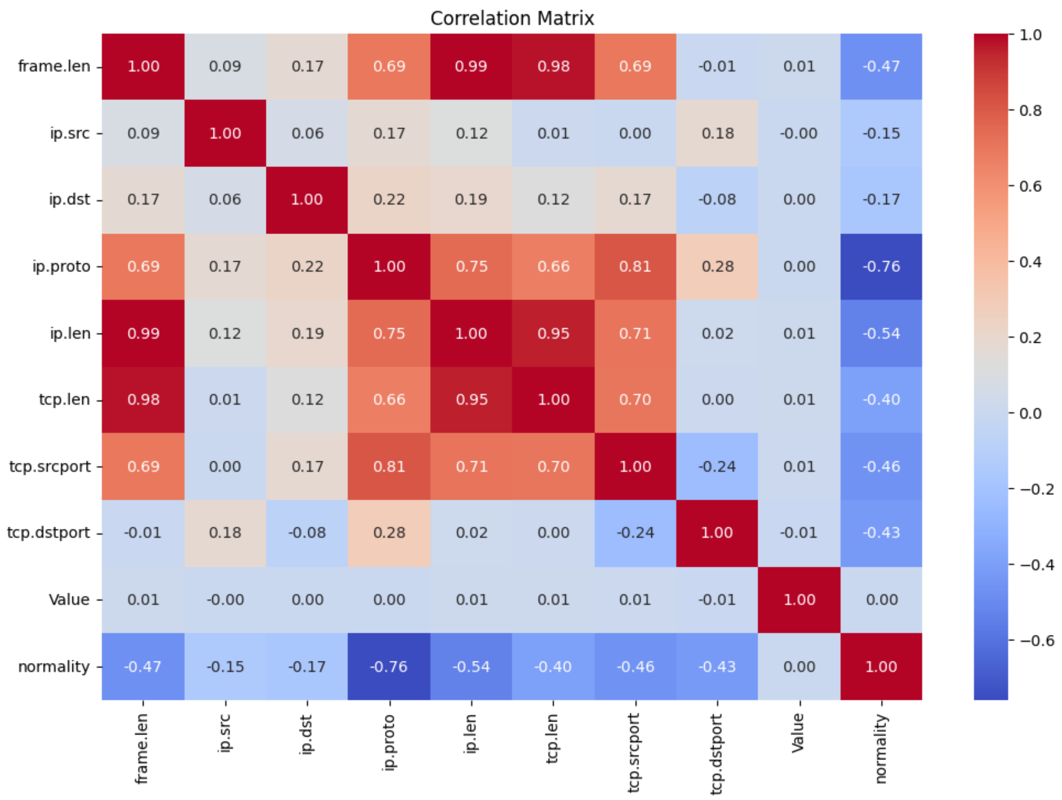

Fig 22: Calculating selected features with correlation_matrix

9. The following show the different machine learning algorithms used in this model

```
# 1. Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
nb_class_report = classification_report(y_test, nb_predictions, output_dict=True)

print("Naive Bayes Accuracy:", nb_accuracy)
print("\nClassification Report for Naive Bayes:\n", classification_report(y_test, nb_predictions))
```

```
Naive Bayes Accuracy: 0.5715991286619935

Classification Report for Naive Bayes:
              precision    recall  f1-score   support

           0       1.00      0.00      0.01     23710
           1       0.67      1.00      0.80     24685
           2       0.79      0.64      0.71     23706
           3       0.58      0.99      0.73     23701
           4       0.29      0.13      0.18     23716
           5       0.43      0.64      0.51     23710

    accuracy                           0.57    143228
   macro avg       0.63      0.57      0.49    143228
weighted avg       0.63      0.57      0.49    143228
```

Fig 23: Navie Bayes

```
# Cross-validation for Decision Tree
cv_scores = cross_val_score(dt_model, X_train, y_train, cv=5, scoring='accuracy')  # 5-fold cross-validation
```

```
# Fit the model on the training data
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)

# Evaluate the model
dt_accuracy = accuracy_score(y_test, dt_predictions)
dt_class_report = classification_report(y_test, dt_predictions, output_dict=True)
```

```
# Print the cross-validation scores and accuracy on test set
print("Decision Tree Cross-Validation Scores:", cv_scores)
print("Average Cross-Validation Score:", np.mean(cv_scores))
print("Decision Tree Accuracy on Test Set:", dt_accuracy)
print("\nClassification Report for Decision Tree:\n", classification_report(y_test, dt_predictions))
```

```
Decision Tree Cross-Validation Scores: [0.99236984 0.99370138 0.99329743 0.84785829 0.99262407]
Average Cross-Validation Score: 0.9639701989298418
Decision Tree Accuracy on Test Set: 0.9931507805736308

Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     23710
           1       0.98      1.00      0.99     24685
           2       0.99      1.00      1.00     23706
           3       1.00      0.99      1.00     23701
           4       1.00      1.00      1.00     23716
           5       0.99      1.00      0.99     23710

    accuracy                           0.99    143228
   macro avg       0.99      0.99      0.99    143228
weighted avg       0.99      0.99      0.99    143228
```

Fig 24: Decision Tree

```
# 3. K-Nearest Neighbors
knn_model = KNeighborsClassifier(n_neighbors=5)  # Default 5 neighbors
knn_model.fit(X_train_scaled, y_train)  # Use scaled features
knn_predictions = knn_model.predict(X_test_scaled) # Now X_test_scaled is defined
knn_accuracy = accuracy_score(y_test, knn_predictions)
knn_class_report = classification_report(y_test, knn_predictions, output_dict=True)

print("KNN Accuracy:", knn_accuracy)
print("\nClassification Report for KNN:\n", classification_report(y_test, knn_predictions))
```

```
KNN Accuracy: 0.9993855949953919

Classification Report for KNN:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     23710
           1       1.00      1.00      1.00     24685
           2       1.00      1.00      1.00     23706
           3       1.00      1.00      1.00     23701
           4       1.00      1.00      1.00     23716
           5       1.00      1.00      1.00     23710

    accuracy                           1.00    143228
   macro avg       1.00      1.00      1.00    143228
weighted avg       1.00      1.00      1.00    143228
```

Fig 25: K-Nearest Neighbors (KNN)

```
# 4. Logistic Regression (ensure no iteration limit issues)
lr_model = LogisticRegression(max_iter=500, random_state=42)  # Increase max_iter to avoid warnings
lr_model.fit(X_train_scaled, y_train)  # Use scaled features
lr_predictions = lr_model.predict(X_test_scaled)
lr_accuracy = accuracy_score(y_test, lr_predictions)
lr_class_report = classification_report(y_test, lr_predictions, output_dict=True)

print("Logistic Regression Accuracy:", lr_accuracy)
print("\nClassification Report for Logistic Regression:\n", classification_report(y_test, lr_predictions))
```

```
Logistic Regression Accuracy: 0.8803585891026894

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

           0       1.00      0.99      1.00     23710
           1       0.99      1.00      0.99     24685
           2       1.00      1.00      1.00     23706
           3       1.00      0.99      1.00     23701
           4       0.64      0.68      0.66     23716
           5       0.66      0.62      0.64     23710

    accuracy                           0.88    143228
   macro avg       0.88      0.88      0.88    143228
weighted avg       0.88      0.88      0.88    143228
```

Fig 26: Logistic Regression