National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Anudeep Yarlagadda
Student ID: X23338466

School of Computing
National College of Ireland

Supervisor: SHERESH ZAHOOR

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Anudeep Yarlagadda		
Student ID:	X23338466		
Programme:	MSc Artificial Intelligence	**Year:**	2024-2025
Module:	MSc (Research) Practicum		
Supervisor:	Sheresh Zahoor		
Submission Due Date:	11th Aug 2024		
Project Title:	Configuration Manual		
Word Count:	1116	**Page Count:**	13

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: Anudeep Yarlagadda

Date: 8th Aug 2025

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: Network intrusion for medical IOT

1. Introduction

1.1 Background

With the rapid growth of **Internet of Medical Things (IoMT)**, smart healthcare systems are increasingly vulnerable to cyber-attacks due to their dependency on real-time data communication. Ensuring the **security and integrity of patient data** is critical, especially in environments involving continuous biometric monitoring. Traditional security methods fall short in such settings, making **machine learning-based Intrusion Detection Systems (IDS)** a promising solution.

This project focuses on developing a **machine learning pipeline for detecting network intrusions** in **medical IoT environments** using the **WUSTL-EHMS-2020 dataset**. This dataset is unique because it combines **network traffic data** with **biometric signals**, simulating a real-world healthcare monitoring system under both normal and attack conditions. The two primary types of attacks covered are **man-in-the-middle attacks**: **spoofing** (violating confidentiality) and **data injection** (violating integrity).

1.2 Aim of the study

The aim of this project is to **develop, evaluate, and interpret** various machine learning models to accurately detect intrusions in medical IoT networks and to provide **explainability** for the predictions using **Explainable AI (XAI)** tools such as **LIME**.

1.3 Research Objectives

This study has a few research objectives as follows:

2 To explore and understand the WUSTL-EHMS-2020 dataset, which combines real-time network flow metrics with patient biometric data from a medical IoT testbed.

3 To clean, preprocess, and normalize the dataset, handling missing values, encoding labels, removing outliers, and applying feature scaling techniques.

4 To address class imbalance in the dataset using SMOTE (Synthetic Minority Oversampling Technique) to ensure fair model training.

5 To identify the most significant features influencing intrusion detection using feature importance techniques.

6 To develop and train multiple machine learning models including Logistic Regression, SVM, Random Forest, Decision Tree, Naïve Bayes, LightGBM, AdaBoost, Gradient Boost, and a Super Learner ensemble model.

7 To evaluate model performance using metrics such as the confusion matrix, classification report, and accuracy, precision, recall, and F1-score.

8 To apply Explainable AI (XAI) techniques using LIME to interpret the decisions made by the best-performing model and understand the contribution of biometric and network features in intrusion detection.

9 To demonstrate the importance of explainability and transparency in AI systems used in healthcare security environments.

9.1 Research Questions

How effectively can machine learning models detect network intrusions in a medical IoT environment using both biometric and network flow data, and how can explainable AI techniques help interpret these model decisions?

The manual is organized into the following six sections, excluding this introduction:

- **Environmental Setup**
- **Libraries Required**
- **Dataset**
- **Results**
- **User Interface**

1. Environmental Setup

2.1 Hardware Requirements

- 16GB RAM.
- 500 GB HardDrive.
- Intel. Core i5
-

2.2 Software Requirements

- Windows 11
- Python

1.3 Programming Prerequisites

- Python
- Colab
- Vs code

For implementing the code, Google Colab has been used since they are hosted in cloud the code is centralized and can be accessed anytime anywhere, also they have all the packages readily available.
Colab Notebook with a Tesla T4 GPU was used to accomplish the work. Colab Notebook maintains that the model training session is not interrupted due to a lack of RAM & GPU.

IDE	**Google Colab Notebook on Cloud**
Program Language	**Python**
Computation	**Tesla T4 GPU**
Visualization Library	**Matplotlib, Seaborn,**
Modelling Library	**Sklearn, lightgbm, mlxtend**

2. Libraries Required

All the libraries required for building this research project are mentioned in table 1 along with their usage:

pandas	It offers data structures and operations for manipulating numerical tables and time series
numpy	It is used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices
sklearn	Used for implementing Label Encoding
matplotlib	It is used for plotting various graphical visualisations
sklearn.metrics	For generating various performance metrics: Accuracy, Confusion matrix , Model:- RandomForestClassifier, LogisticRegression, svm, DecisionTreeClassifier, GaussianNB, AdaBoostClassifier GradientBoostingClassifier
lightgbm	LGBMClassifier
Imblearn	Smote for over sampling
mlxtend	Stacking classifier
lime	Lime is used for highlight the area which were important for prediction

3. Data Set Details

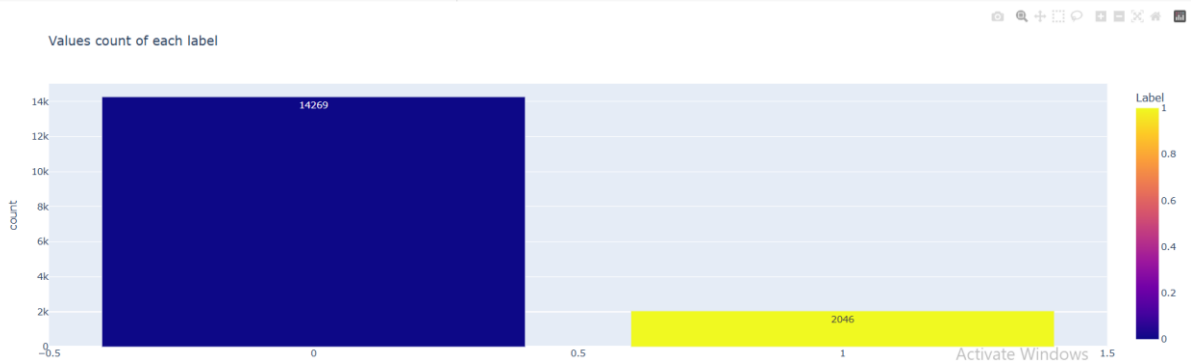
The WUSTL-EHMS-2020 dataset was created using a real-time Enhanced Healthcare Monitoring System (EHMS) testbed [1]. This testbed collects both the network flow metrics and patients' biometrics due to the scarcity of a dataset that combines these biometrics.

The EHMS testbed is divided into four parts: medical sensors, gateway, network, and control with visualization, as shown in Fig. 1. The data flow starts from the sensors attached to the patent's body, to the gateway. Then the gateway sends the data to the server for visualization via the switch and router. An attacker can intercept these data before they arrive at the server. The Intrusion Detection System (IDS) is responsible for capturing the real-time network flow traffic and the patient's biometric data and abnormalities detection.The type of attacks included in this dataset is man-in-the-middle attacks: spoofing and data injection. The spoofing attack is only to sniff the packets between the gateway and the server, which violates the patient's data confidentiality. The data injection attack is used to modify the packets on-the-fly, which violates the data's integrity.

The network flow traffic and the patient's biometric data were captured and stored as a "CSV" file by the Audit Record Generation and Utilization System (ARGUS) tool [2]. Table 1 presents the statistical information on the WUSTL-EHMS-2020 dataset. This dataset consists of 44 features where 35 features are network flow metrics, eight patients' biometric features, and one feature for the label [1]. We used the Source MAC address feature to label the data where the samples with the attacker laptop MAC addresses are labeled as 1, while the rest as 0.

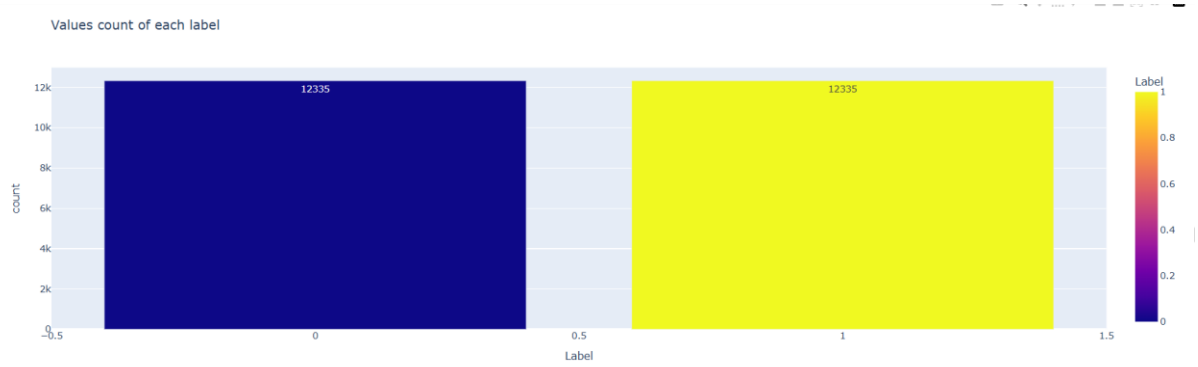
Link :- https://www.cse.wustl.edu/~jain/ehms/index.html

The number of records against each target class is depicted in figure



Data Balance (Using Smote)

The data obtained from the repository is imbalanced in nature; and hence needs to be balanced so that respective algorithms can be performed on them. For this purpose, a simple approach of duplicating minority class is performed using SMOTE. SMOTE is abbreviated to Synthetic Mining Oversampling Technique. The problem of data imbalance is resolved in the dataset by oversampling the minority classes of the dataset.



Coding Implemetation:

- Mounting Google drive:

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

- Importing the Libraries:

```
[3]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import plotly.express as px
     import warnings
     from sklearn import metrics
     from sklearn.preprocessing import LabelEncoder
     from sklearn.preprocessing import MinMaxScaler
     from imblearn.over_sampling import SMOTE
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
     from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
     from lightgbm import LGBMClassifier
     from sklearn.model_selection import KFold
     from mlxtend.classifier import StackingClassifier
     from math import sqrt
     from numpy import hstack
     from numpy import vstack
     from numpy import asarray
     from sklearn import svm
     warnings.filterwarnings('ignore')
```

- Load Training Data:

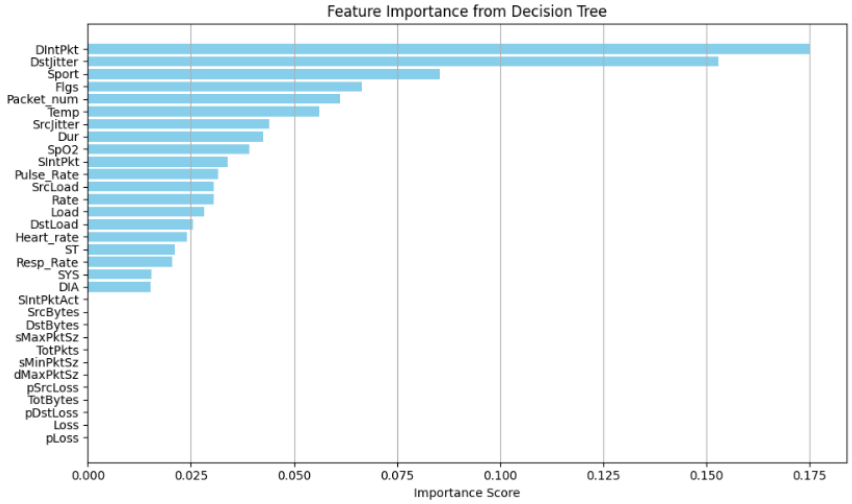
```
df=pd.read_csv('/content/drive/MyDrive/network_intrusion(medical_iot)/Data/wustl-ehms-2020_with_attacks_categories.csv')
df.head()
```

	Dir	Flgs	SrcAddr	DstAddr	Sport	Dport	SrcBytes	DstBytes	SrcLoad	DstLoad	...	Temp	SpO2	Pulse_Rate	SYS	DIA	Heart_rate	Resp_Rate	ST	Attack Category	Label
0	->	e	10.0.1.172	10.0.1.150	58059	1111	496	186	276914.0	92305.0	...	28.9	0	0	0	0	0	0	0.0	normal	0
1	->	e	10.0.1.172	10.0.1.150	58062	1111	496	186	230984.0	76995.0	...	28.9	0	0	0	0	78	17	0.4	normal	0
2	->	e	10.0.1.172	10.0.1.150	58065	1111	496	186	218470.0	72823.0	...	28.9	89	104	0	0	78	17	0.4	normal	0
3	->	e	10.0.1.172	10.0.1.150	58067	1111	496	186	203376.0	67792.0	...	28.9	89	104	0	0	79	17	0.4	normal	0
4	->	e	10.0.1.172	10.0.1.150	58069	1111	496	186	235723.0	78574.0	...	28.9	89	101	0	0	79	17	0.4	normal	0

5 rows × 45 columns

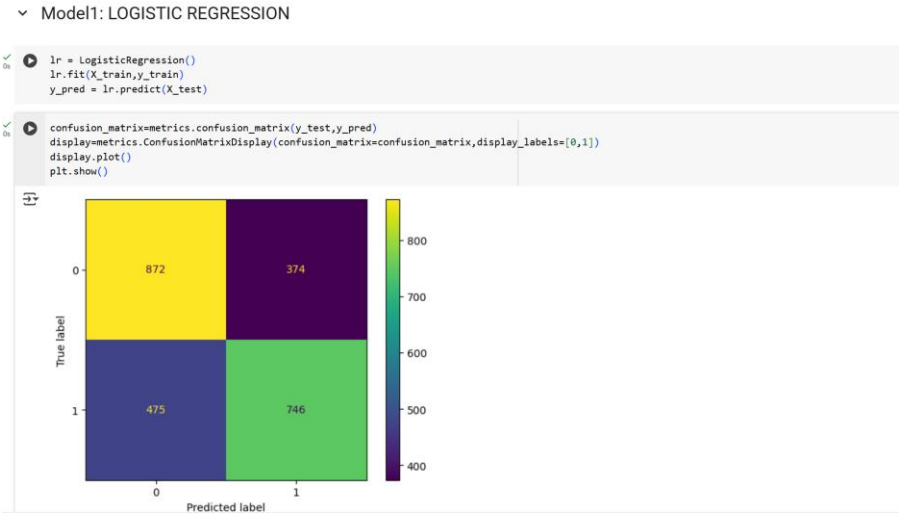
- Feature Selection:

```
[29]

dtcmodel = RandomForestClassifier(max_depth=10)
dtcmodel.fit(X,y)
importances = dtcmodel.feature_importances_
feature_names = X.columns  # Or use list of column names if it's a NumPy array

# Create DataFrame for easy sorting and plotting
feat_imp_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feat_imp_df = feat_imp_df.sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10, 6))
plt.barh(feat_imp_df['Feature'], feat_imp_df['Importance'], color='skyblue')
plt.xlabel('Importance Score')
plt.title('Feature Importance from Decision Tree')
plt.gca().invert_yaxis()  # Most important at the top
plt.grid(True, axis='x')
plt.tight_layout()
plt.show()
```



Model1: LOGISTIC REGRESSION

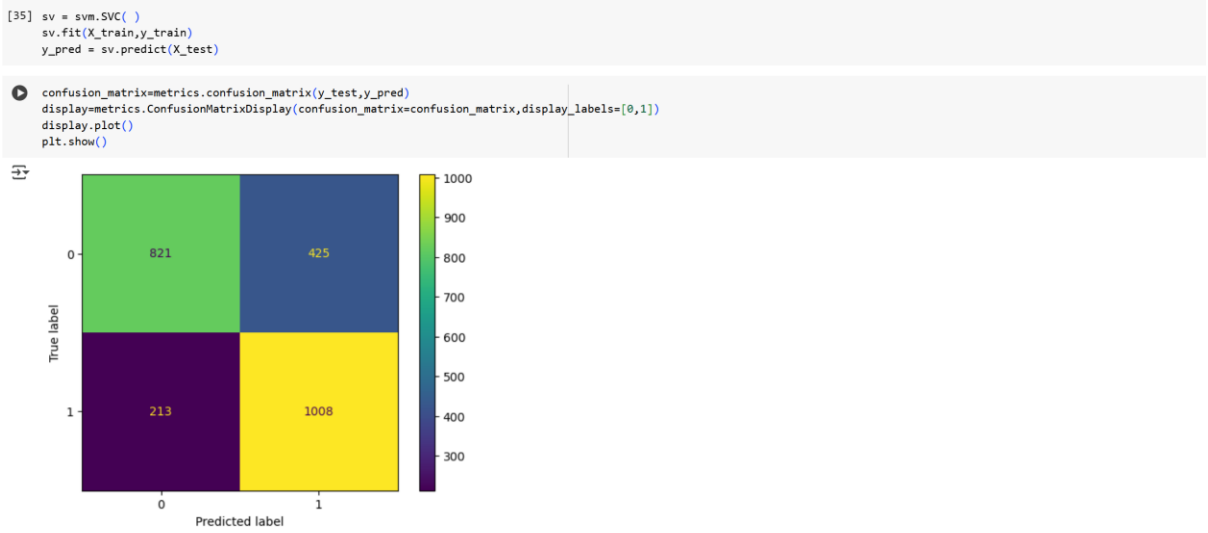
Output of the logistic regression model:



Accuracy Score: 0.66

Model2: SVM

Output of the SVM:



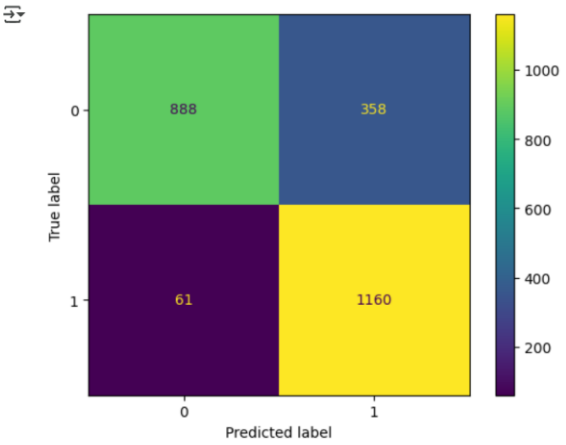
Accuracy Score: 0.74

Model3: RANDOM FOREST

Output of the RANDOM FOREST:

```
[38] dtcmodel = RandomForestClassifier(max_depth=10)
     dtcmodel.fit(X_train,y_train)
     y_pred = dtcmodel.predict(X_test)
```

```
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
display.plot()
plt.show()
```



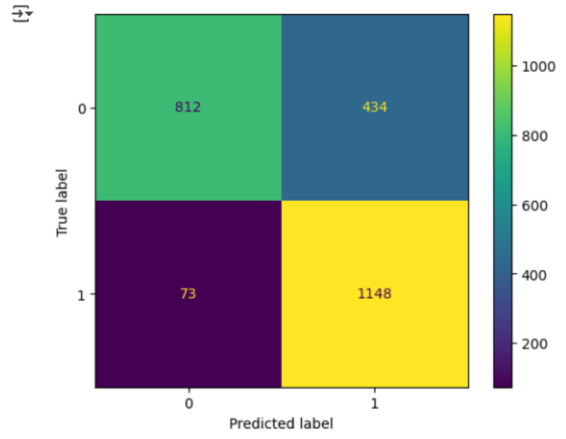
Accuracy Score: 0.83

Mode4: DECISION TREE

Output of DECISION TREE:

```
[41] dt = DecisionTreeClassifier(max_depth=10)
     dt.fit(X_train,y_train)
     y_pred = dt.predict(X_test)
```

```
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
display.plot()
plt.show()
```



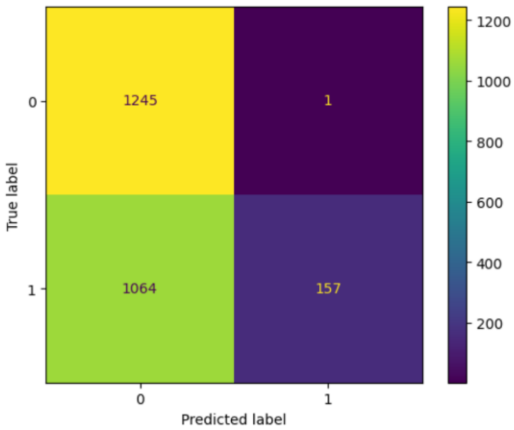
Accuracy Score: 0.79

Model5: NAIVE BAYES

Output of the NAIVE BAYES:

```
[44] nb=GaussianNB()
     nb.fit(X_train,y_train)
     y_pred = nb.predict(X_test)
```

```
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
display.plot()
plt.show()
```



Accuracy Score: 0.57

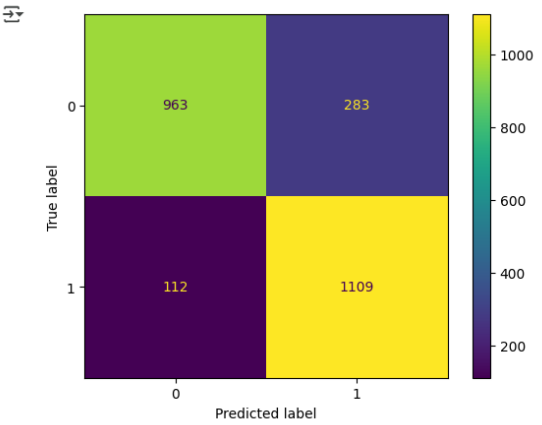
Model6: LGBM

Output of the LGBM:

```
lgbm = LGBMClassifier(max_depth=3)
lgbm.fit(X_train, y_train)
y_pred = lgbm.predict(X_test)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[48] confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
     display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
     display.plot()
     plt.show()
```



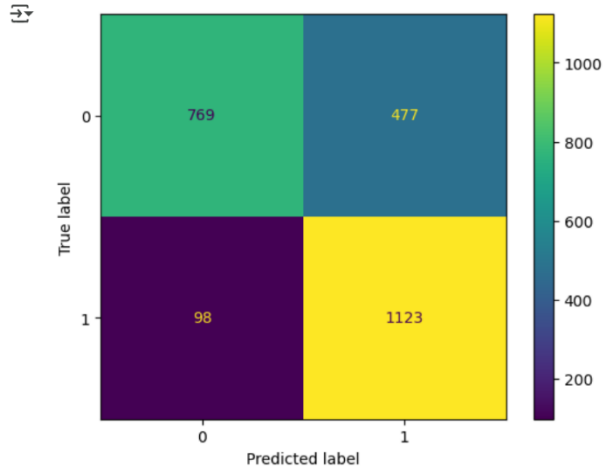
Accuracy Score: 0.84

Model7: AdaBoost Classifier

Output of the AdaBoost Classifier:

```
[50] ada = AdaBoostClassifier(n_estimators=100)
     ada.fit(X_train, y_train)
     y_pred = ada.predict(X_test)
```

```
[51] confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
     display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
     display.plot()
     plt.show()
```



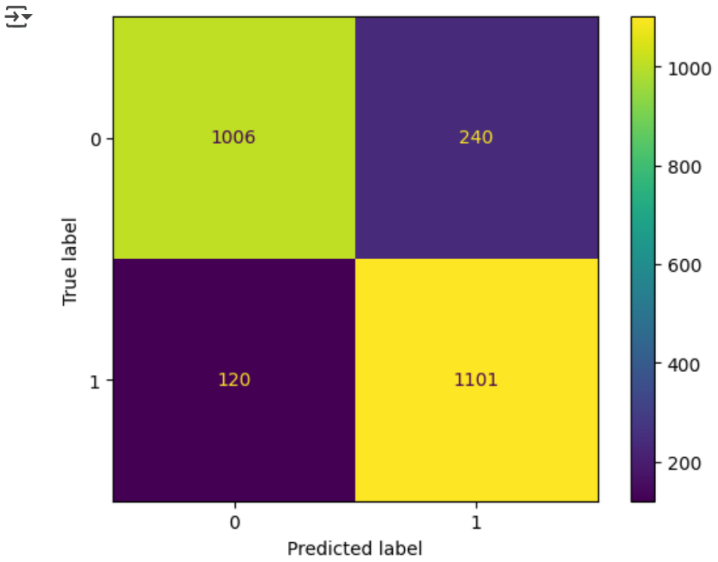
Accuracy Score: 0.77

Model8: Gradient Boosting Classifier

Output of the Gradient Boosting Classifier:

```
[53] gb = GradientBoostingClassifier( max_depth=3)
     gb.fit(X_train, y_train)
     y_pred = gb.predict(X_test)
```

```
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
display.plot()
plt.show()
```



Accuracy Score: 0.85

Model9: SUPER LEARNER

Output of the SUPER LEARNER:

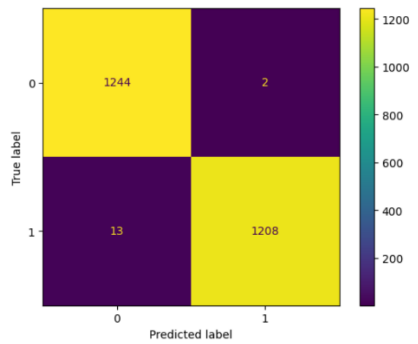
```
[56] from re import VERBOSE
     # create a list of base-models
     def get_models():
         models = list()
         models.append(LGBMClassifier(verbose=-1))
         models.append(AdaBoostClassifier())
         models.append(RandomForestClassifier())
         return models


     # collect out of fold predictions form k-fold cross validation
     def get_out_of_fold_predictions(X, y, models):
       meta_X, meta_y = list(), list()
       # define split of data
       kfold = KFold(n_splits=10, shuffle=True)
       # enumerate splits
       for train_ix, test_ix in kfold.split(X):
         fold_yhats = list()
         # Use .iloc to index by integer position
         train_X, test_X = X.iloc[train_ix], X.iloc[test_ix]
         train_y, test_y = y[train_ix], y[test_ix]
         meta_y.extend(test_y)
         # fit and make predictions with each sub-model
         for model in models:
           model.fit(train_X, train_y)
           yhat = model.predict(test_X)
           # store columns
           fold_yhats.append(yhat.reshape(len(yhat),1))
         # store fold yhats as columns
         meta_X.append(hstack(fold_yhats))
       return vstack(meta_X), asarray(meta_y)

     # fit all base models on the training dataset
     def fit_base_models(X, y, models):
       for model in models:
         model.fit(X, y)
```

```
[55]        model.fit(train_X, train_y)
            yhat = model.predict(test_X)
            # store columns
            fold_yhats.append(yhat.reshape(len(yhat),1))
          # store fold yhats as columns
          meta_X.append(hstack(fold_yhats))
        return vstack(meta_X), asarray(meta_y)

      # fit all base models on the training dataset
      def fit_base_models(X, y, models):
        for model in models:
          model.fit(X, y)

      # fit a meta model
      def fit_meta_model(X, y):
        model = GradientBoostingClassifier()
        model.fit(X, y)
        return model

      # evaluate a list of models on a dataset
      def evaluate_models(X, y, models):
        for model in models:
          yhat = model.predict(X)
          acc = accuracy_score(y, yhat)
          print('%s: %.3f' % (model.__class__.__name__, acc*100))

      # make predictions with stacked model
      def super_learner_predictions(X, models, meta_model):
        meta_X = list()
        for model in models:
          yhat = model.predict(X)
          meta_X.append(yhat.reshape(len(yhat),1))
        meta_X = hstack(meta_X)
        # predict
        return meta_model.predict(meta_X)
```

```
[57] models = get_models()
     # get out of fold predictions
     meta_X, meta_y = get_out_of_fold_predictions(newx, y, models)
     print('Meta ', meta_X.shape, meta_y.shape)
     # fit base models
     fit_base_models(newx, y, models)
     # fit the meta model
     meta_model = fit_meta_model(meta_X, meta_y)
     # evaluate base models
     evaluate_models(X_test, y_test, models)
     # evaluate meta model
     yhat = super_learner_predictions(X_test, models, meta_model)
     print('Super Learner: %.3f' % (accuracy_score(y_test, yhat) * 100))
```

```
Meta  (24670, 3) (24670,)
LGBMClassifier: 97.203
AdaBoostClassifier: 74.422
RandomForestClassifier: 100.000
Super Learner: 99.392
```

```
[58] #confusion matrix
     confusion_matrix=metrics.confusion_matrix(y_test,yhat)
     display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,display_labels=[0,1])
     display.plot()
     plt.show()
```




```
[59] print(classification_report(y_test, yhat))
```

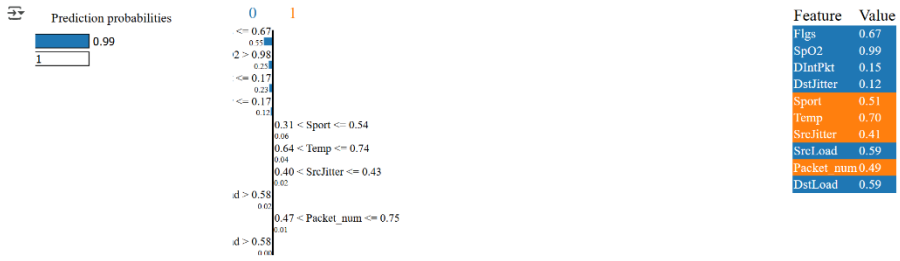
```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1246
           1       1.00      0.99      0.99      1221

    accuracy                           0.99      2467
   macro avg       0.99      0.99      0.99      2467
weighted avg       0.99      0.99      0.99      2467
```

Accuracy Score: 0.99

TESTING MODEL WITH LIME:

```
from lime.lime_tabular import LimeTabularExplainer
import numpy as np

# Create LIME explainer
explainer = LimeTabularExplainer(
    training_data=newx.values,
    feature_names=newx.columns,
    class_names=np.unique(y).astype(str),
    mode='classification'
)

#  predict_proba function for stacked model
def stacked_predict_proba(X):
    # Convert to DataFrame if needed
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X, columns=newx.columns)

    # Get predictions from base models
    meta_X = []
    for model in models:
        yhat = model.predict(X)
        meta_X.append(yhat.reshape(len(yhat), 1))
    meta_X = np.hstack(meta_X)

    # Get probability prediction from meta-model
    return meta_model.predict_proba(meta_X)

#Choose one instance from X_test to explain
i = 0  # index number
instance = X_test.iloc[i].values

# Generate explanation
exp = explainer.explain_instance(
    data_row=instance,
    predict_fn=stacked_predict_proba
)
```

```
exp = explainer.explain_instance(
    data_row=instance,
    predict_fn=stacked_predict_proba
)

#  explanation
exp.show_in_notebook(show_table=True)
```




```
[ ] Start coding or generate with AI.
```