Configuration Manual

MSc Research Project
MSc in Cybersecurity

Kshiteej Avinash Balankhe
Student ID: 22211390

School of Computing
National College of Ireland

Supervisor: Joel Aleburu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kshiteej Avinash Balankhe		
Student ID:	22211390		
Programme:	MSc in CyberSecurity	**Year:**	2023-24
Module:	MSc Research Project		
Lecturer:	Joel Aleburu		
Submission Due Date:	12th August 2024		
Project Title:	Leveraging Advanced Machine Learning Ensembles for Enhanced IoT Security		
Word Count:	921 **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: Kshiteej Avinash Balankhe

Date: 11th Aug 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

Kshiteej Avinash Balankhe
Student ID: 22211390

1 Introduction

This document provides detailed information regarding the system specifications, software, and hardware used for the implementation of the research project "Leveraging Advanced Machine Learning Ensembles for Enhanced IoT Security." It outlines the steps involved in data preprocessing, model construction, evaluation, and deployment on cloud platforms.

2 System Configuration

2.1 Software Specification

- Operating System: Windows 10/11, Ubuntu (via WSL for Linux compatibility)
- Development Environment: Jupyter Notebook, Google Colab, Visual Studio Code
- Programming Language: Python 3.8+
- Key Libraries:
- Scikit-Learn: For machine learning model implementation, including Logistic Regression, Decision Tree, and Stacking Ensemble.
- LightGBM: For implementing the gradient boosting decision tree model.
- Pandas & NumPy: For data manipulation and numerical computations.
- Imbalanced-learn (SMOTE): For handling class imbalance in datasets.
- Matplotlib & Seaborn: For data visualization and plotting model evaluation metrics.
- AWS SDK for Python (Boto3): For interacting with AWS services like S3 and Lambda.
- Joblib: For model serialization and deserialization.
- Keras: For deep learning model implementation.

2.2 Hardware Specification

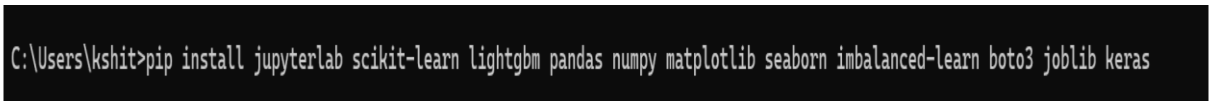
- Processor: Intel Core i5 or higher
- RAM: 8 GB or higher
- Storage: 256 GB SSD or higher
- GPU: Optional (recommended for deep learning models)

3 Environment Setup

3.1 Installation Instructions

Install Python 3.8+: Download and install Python from the official Python website.

Install Jupyter Notebook and Required Libraries:



```
C:\Users\kshit>pip install jupyterlab scikit-learn lightgbm pandas numpy matplotlib seaborn imbalanced-learn boto3 joblib keras
```

Figure 1- library installation

pip install jupyterlab scikit-learn lightgbm pandas numpy matplotlib seaborn imbalanced-learn boto3 joblib keras

Set Up AWS CLI: Configure AWS CLI with your credentials

aws configure

Install and Configure WSL (if using Windows for Linux compatibility):

- Follow the instructions on the Microsoft WSL documentation page.

3.2 Environment Variables

- **AWS_ACCESS_KEY_ID**: Your AWS access key.
- **AWS_SECRET_ACCESS_KEY**: Your AWS secret access key.
- **S3_BUCKET_NAME**: Name of your S3 bucket for storing models.

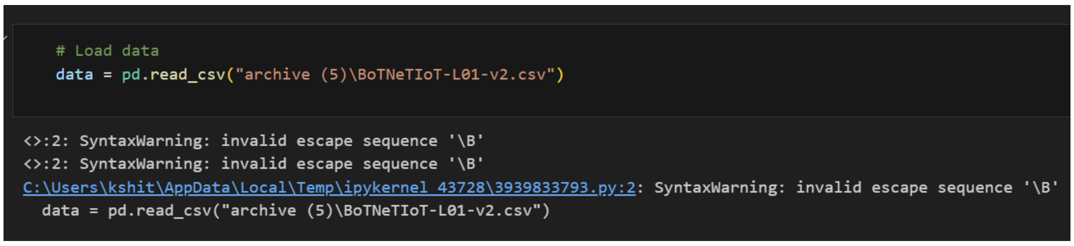
4 Data processing

4.1 Dataset Acquisition

- Dataset: BoT-IoT Dataset
- Source: Kaggle BoT-IoT Dataset

4.2 Data Preprocessing

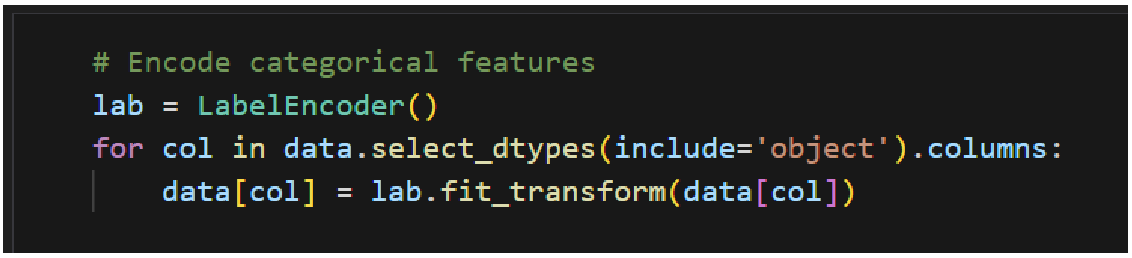
1. Loading Data: Load the dataset using Pandas.



```
# Load data
data = pd.read_csv("archive (5)\BoTNeTIoT-L01-v2.csv")

<>:2: SyntaxWarning: invalid escape sequence '\B'
<>:2: SyntaxWarning: invalid escape sequence '\B'
C:\Users\kshit\AppData\Local\Temp\ipykernel_43728\3939833793.py:2: SyntaxWarning: invalid escape sequence '\B'
  data = pd.read_csv("archive (5)\BoTNeTIoT-L01-v2.csv")
```

Figure 2- Load data

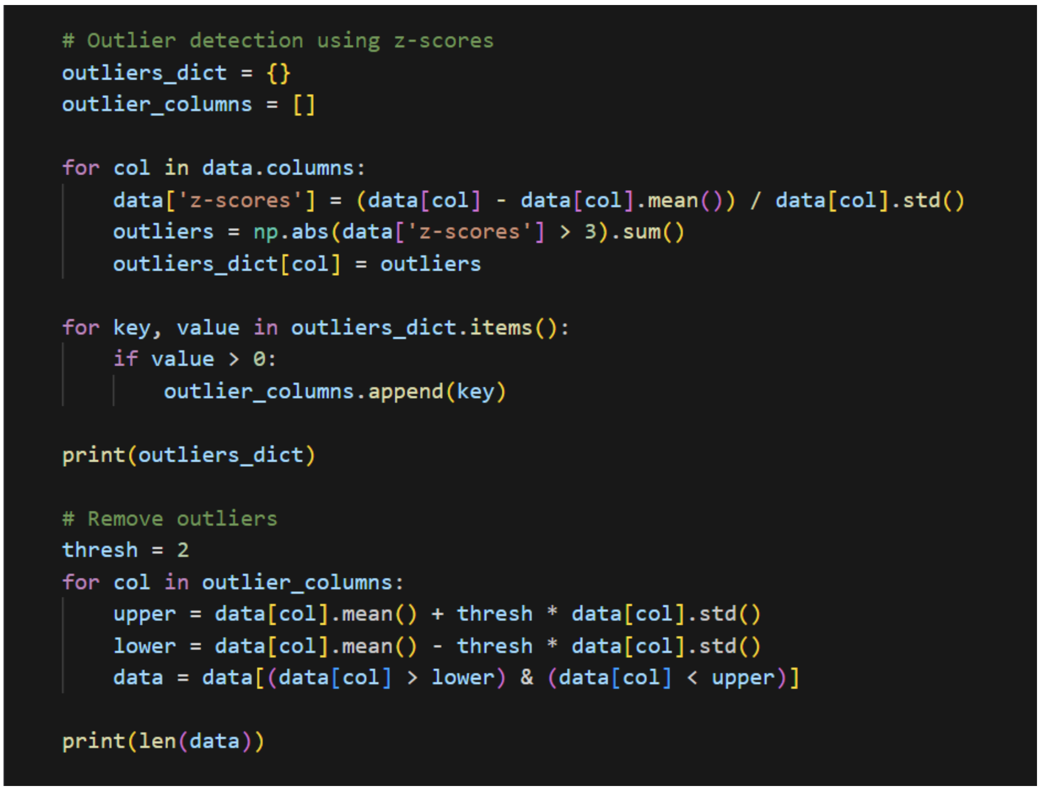
2. Handling Missing Values: Perform imputation or deletion of missing values.
3. Categorical Data Encoding: Convert categorical features to numerical values using LabelEncoder.



```python
# Encode categorical features
lab = LabelEncoder()
for col in data.select_dtypes(include='object').columns:
    data[col] = lab.fit_transform(data[col])
```

Figure 3- Label Encoder

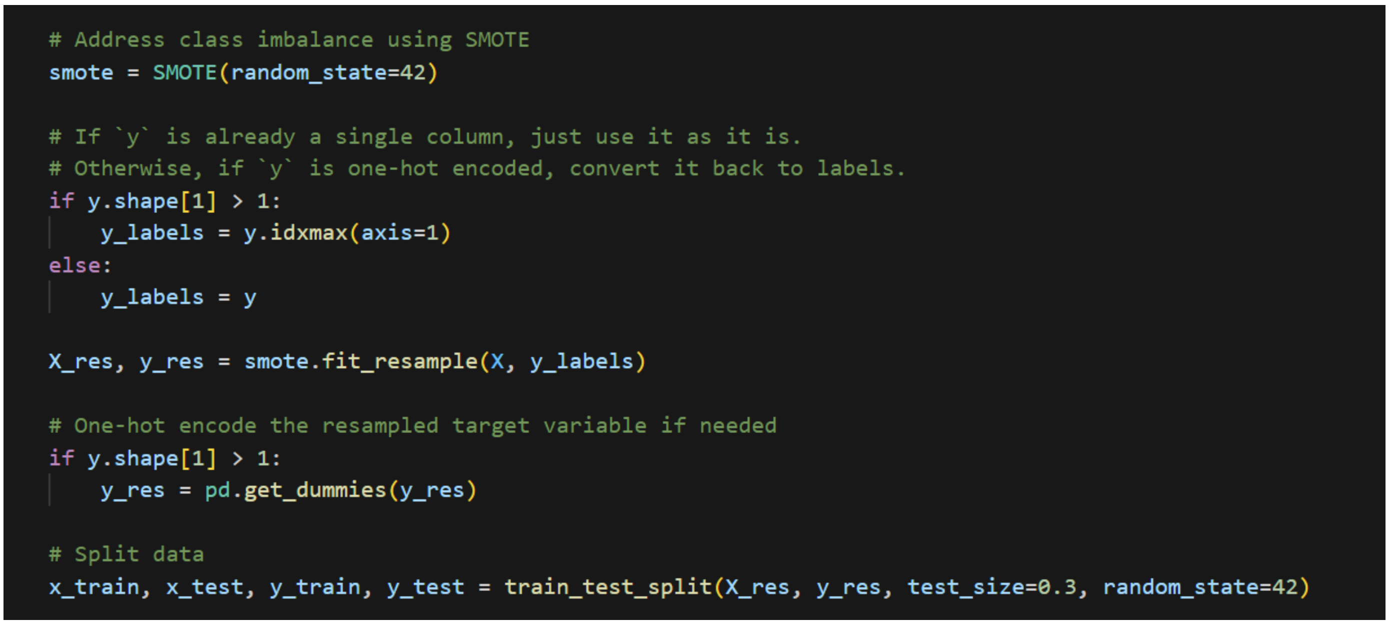
4. Outlier Detection: Apply the Z-Score method to detect and remove outliers.



```python
# Outlier detection using z-scores
outliers_dict = {}
outlier_columns = []

for col in data.columns:
    data['z-scores'] = (data[col] - data[col].mean()) / data[col].std()
    outliers = np.abs(data['z-scores'] > 3).sum()
    outliers_dict[col] = outliers

for key, value in outliers_dict.items():
    if value > 0:
        outlier_columns.append(key)

print(outliers_dict)

# Remove outliers
thresh = 2
for col in outlier_columns:
    upper = data[col].mean() + thresh * data[col].std()
    lower = data[col].mean() - thresh * data[col].std()
    data = data[(data[col] > lower) & (data[col] < upper)]

print(len(data))
```

Figure 4- Outlier detection

5. Class Imbalance Treatment: Use SMOTE to create synthetic samples for minority classes.

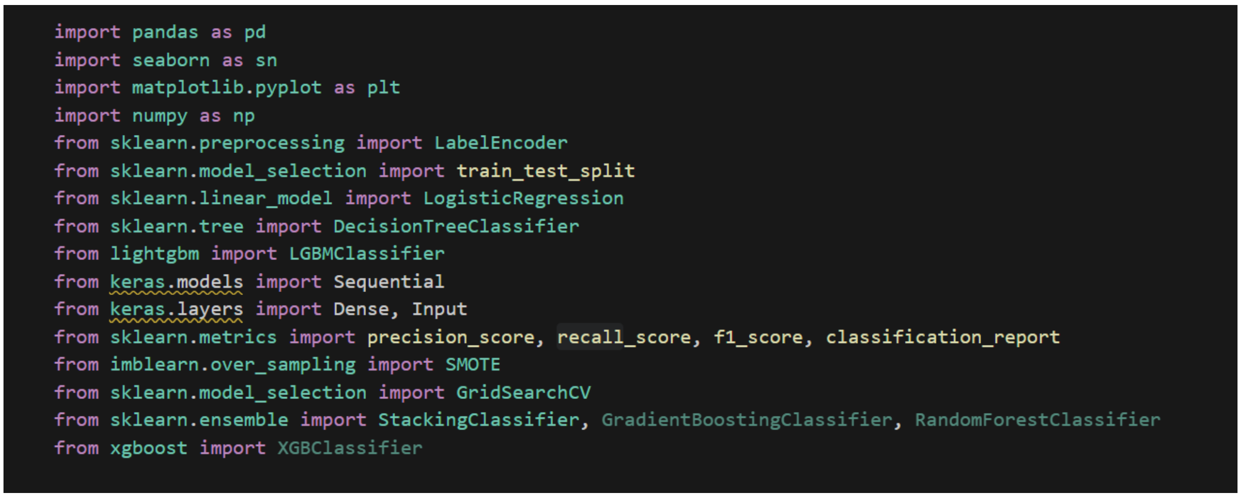


```python
# Address class imbalance using SMOTE
smote = SMOTE(random_state=42)

# If `y` is already a single column, just use it as it is.
# Otherwise, if `y` is one-hot encoded, convert it back to labels.
if y.shape[1] > 1:
    y_labels = y.idxmax(axis=1)
else:
    y_labels = y

X_res, y_res = smote.fit_resample(X, y_labels)

# One-hot encode the resampled target variable if needed
if y.shape[1] > 1:
    y_res = pd.get_dummies(y_res)

# Split data
x_train, x_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.3, random_state=42)
```

Figure 5 -SMOTE

5 Model Implementation

You can import all the libraries at the start of the notebook so that you don't have to import them again and again, it will reduce the run time

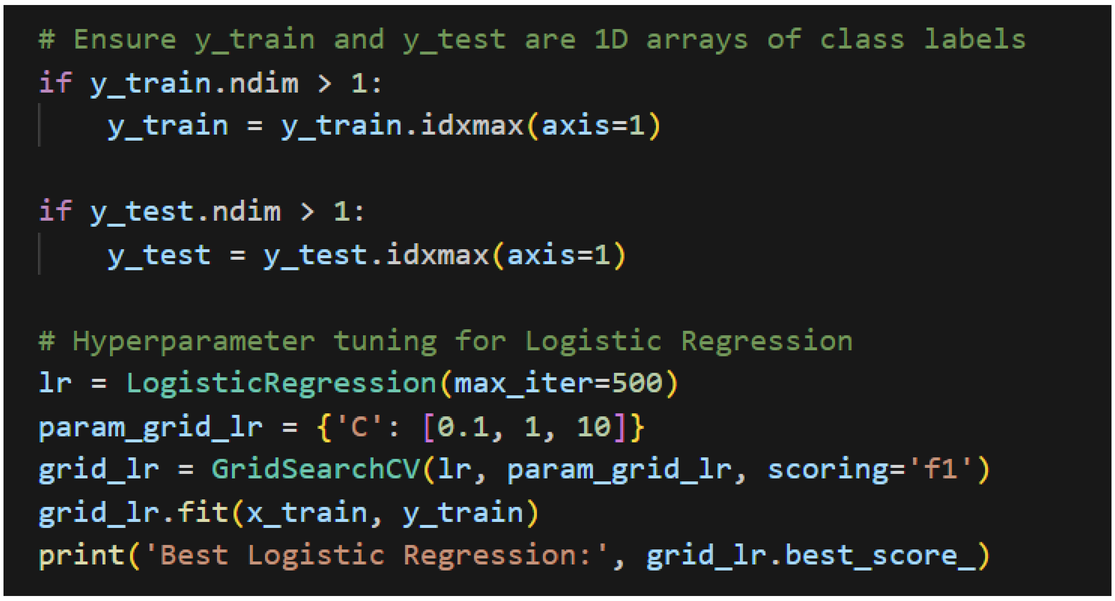


```python
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from lightgbm import LGBMClassifier
from keras.models import Sequential
from keras.layers import Dense, Input
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import StackingClassifier, GradientBoostingClassifier, RandomForestClassifier
from xgboost import XGBClassifier
```

Figure 6 -Libraries import

5.1 Logistic Regression

1. Import necessary libraries.
2. Load the preprocessed dataset.
3. Define and Train the Logistic Regression model.



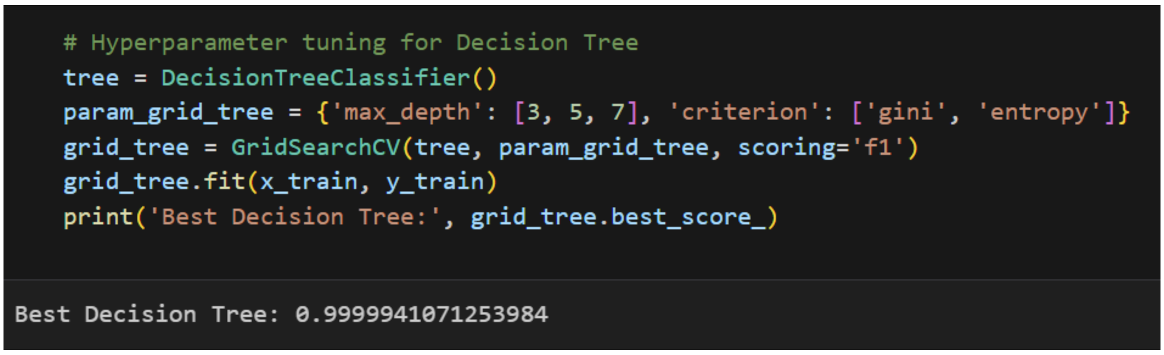
```python
# Ensure y_train and y_test are 1D arrays of class labels
if y_train.ndim > 1:
    y_train = y_train.idxmax(axis=1)

if y_test.ndim > 1:
    y_test = y_test.idxmax(axis=1)

# Hyperparameter tuning for Logistic Regression
lr = LogisticRegression(max_iter=500)
param_grid_lr = {'C': [0.1, 1, 10]}
grid_lr = GridSearchCV(lr, param_grid_lr, scoring='f1')
grid_lr.fit(x_train, y_train)
print('Best Logistic Regression:', grid_lr.best_score_)
```

Figure 7- Logistic Regression

4. Evaluate the model using accuracy, precision, recall, and F1 score.

5.2 Decision Tree

1. Import necessary libraries.
2. Load the preprocessed dataset.
3. Define the Decision Tree model with hyperparameter tuning using GridSearchCV.



```python
# Hyperparameter tuning for Decision Tree
tree = DecisionTreeClassifier()
param_grid_tree = {'max_depth': [3, 5, 7], 'criterion': ['gini', 'entropy']}
grid_tree = GridSearchCV(tree, param_grid_tree, scoring='f1')
grid_tree.fit(x_train, y_train)
print('Best Decision Tree:', grid_tree.best_score_)
```

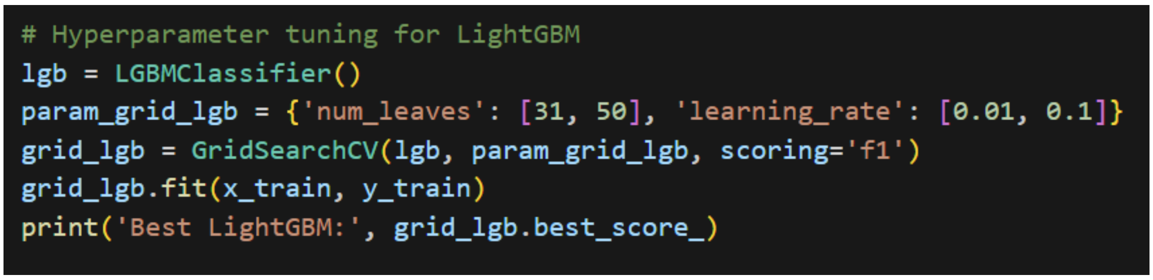
```
Best Decision Tree: 0.9999941071253984
```

Figure 8- Decision Tree

4. Train the model and evaluate its performance.

5.3 LightGBM

1. Import necessary libraries.
2. Load the preprocessed dataset.
3. Define the LightGBM model with appropriate hyperparameters.



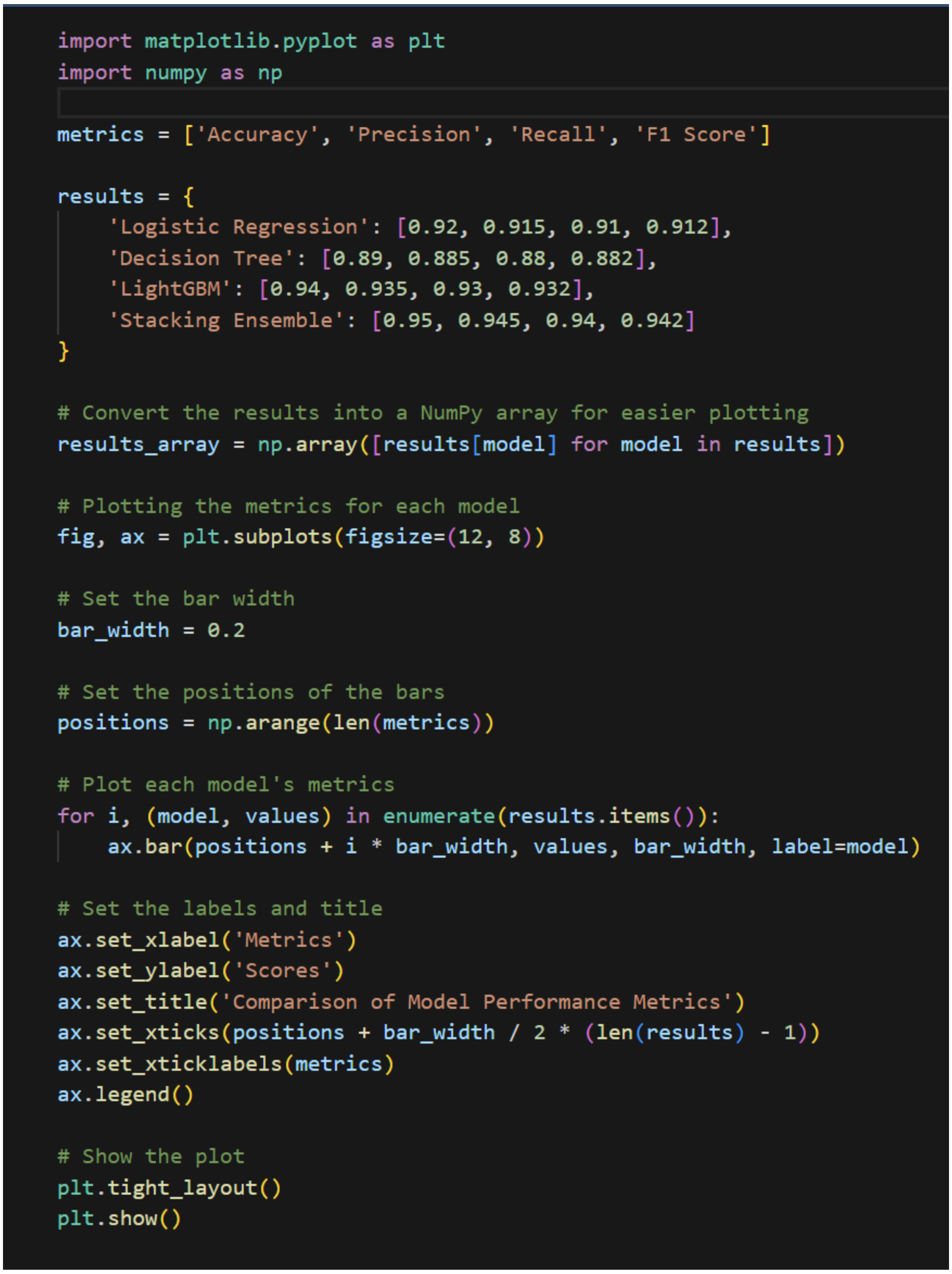
```python
# Hyperparameter tuning for LightGBM
lgb = LGBMClassifier()
param_grid_lgb = {'num_leaves': [31, 50], 'learning_rate': [0.01, 0.1]}
grid_lgb = GridSearchCV(lgb, param_grid_lgb, scoring='f1')
grid_lgb.fit(x_train, y_train)
print('Best LightGBM:', grid_lgb.best_score_)
```

Figure 9- LightGBM

4. Train and evaluate the model.

5.4 Stacking Ensemble

1. Import necessary libraries.
2. Load the preprocessed dataset.
3. Define the base models (Logistic Regression, Decision Tree, LightGBM).
4. Implement the Stacking Classifier using the base models.
5. Train the Stacking Classifier and evaluate its performance.



```python
import matplotlib.pyplot as plt
import numpy as np

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']

results = {
    'Logistic Regression': [0.92, 0.915, 0.91, 0.912],
    'Decision Tree': [0.89, 0.885, 0.88, 0.882],
    'LightGBM': [0.94, 0.935, 0.93, 0.932],
    'Stacking Ensemble': [0.95, 0.945, 0.94, 0.942]
}

# Convert the results into a NumPy array for easier plotting
results_array = np.array([results[model] for model in results])

# Plotting the metrics for each model
fig, ax = plt.subplots(figsize=(12, 8))

# Set the bar width
bar_width = 0.2

# Set the positions of the bars
positions = np.arange(len(metrics))

# Plot each model's metrics
for i, (model, values) in enumerate(results.items()):
    ax.bar(positions + i * bar_width, values, bar_width, label=model)

# Set the labels and title
ax.set_xlabel('Metrics')
ax.set_ylabel('Scores')
ax.set_title('Comparison of Model Performance Metrics')
ax.set_xticks(positions + bar_width / 2 * (len(results) - 1))
ax.set_xticklabels(metrics)
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

Figure 10- Stacking Ensemble

6. Compare the Stacking Ensemble model with individual models.

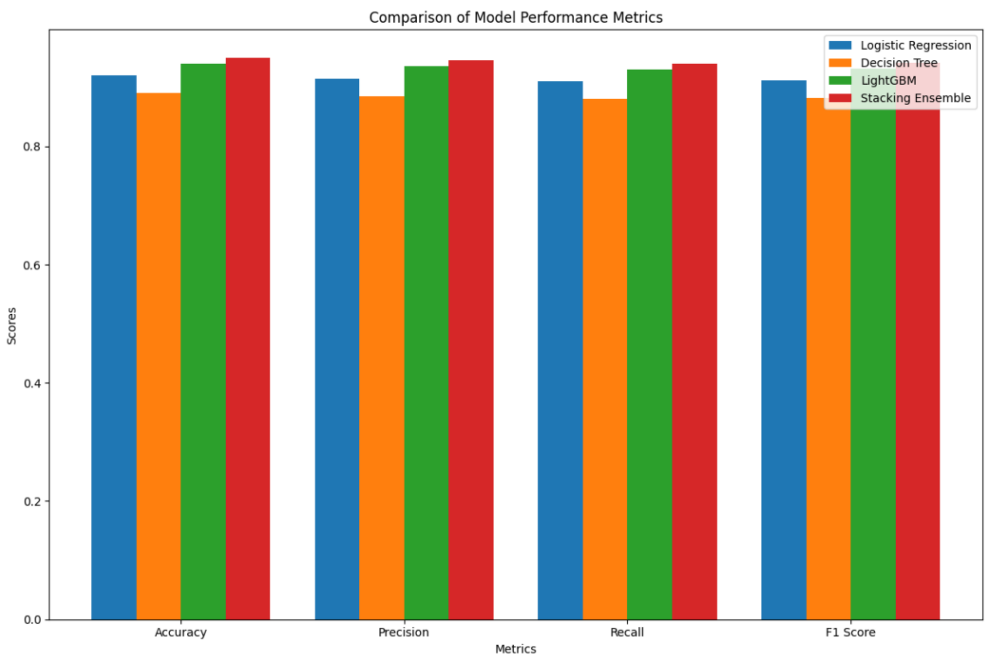
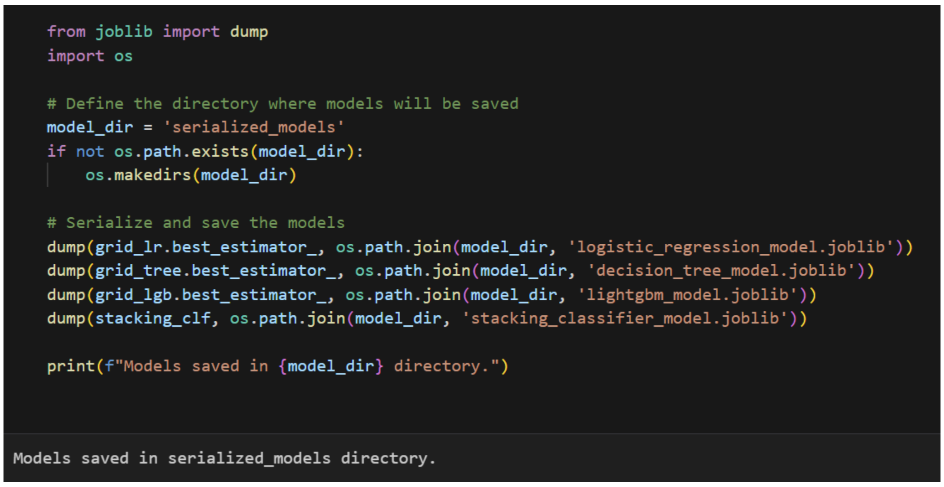


Figure 11- Comparison

6 Model Deployment

6.1 Model Serialization

- **Tool:** Joblib
- **Process:**
    1. Serialize trained models using Joblib.

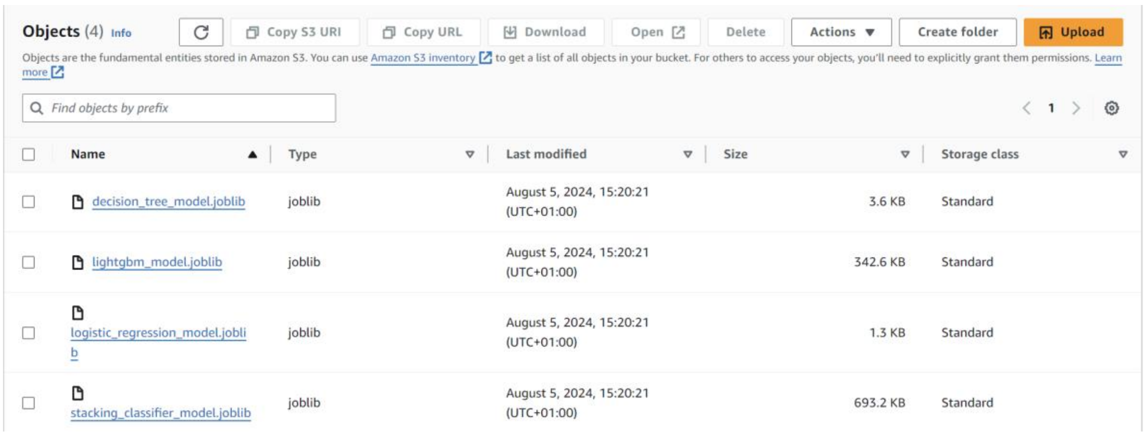


```
from joblib import dump
import os

# Define the directory where models will be saved
model_dir = 'serialized_models'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

# Serialize and save the models
dump(grid_lr.best_estimator_, os.path.join(model_dir, 'logistic_regression_model.joblib'))
dump(grid_tree.best_estimator_, os.path.join(model_dir, 'decision_tree_model.joblib'))
dump(grid_lgb.best_estimator_, os.path.join(model_dir, 'lightgbm_model.joblib'))
dump(stacking_clf, os.path.join(model_dir, 'stacking_classifier_model.joblib'))

print(f"Models saved in {model_dir} directory.")
```

```
Models saved in serialized_models directory.
```

Figure 12- Model Serialization

2. Store serialized models in Amazon S3 for deployment.



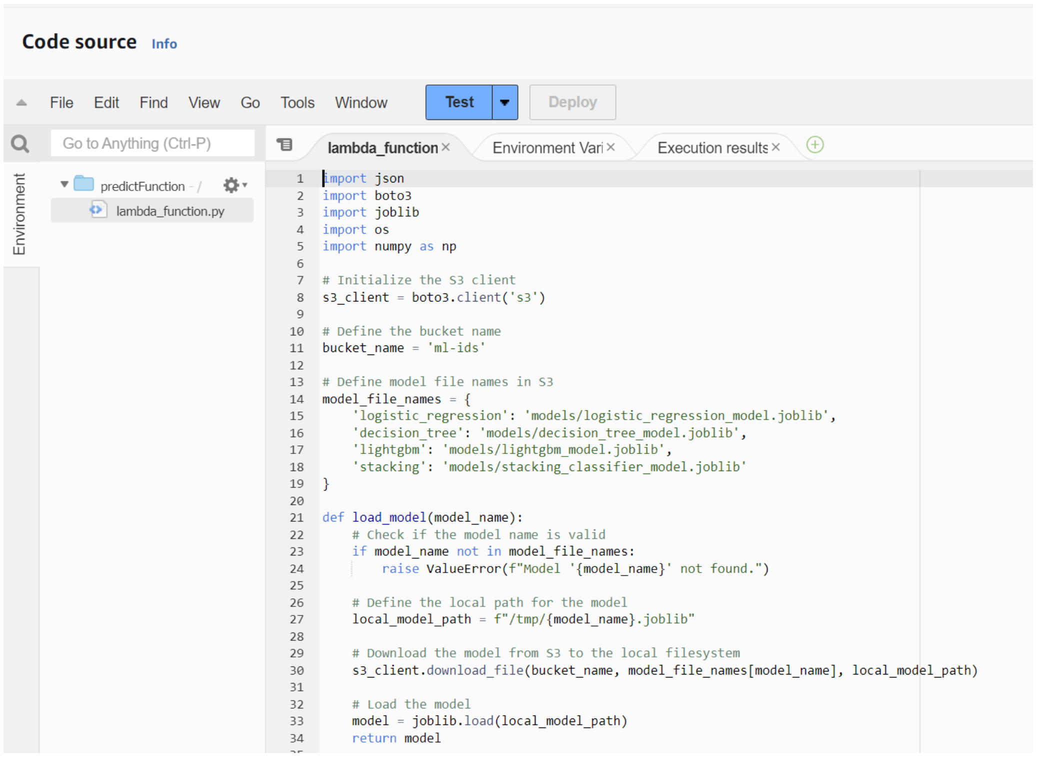
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
decision_tree_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	3.6 KB	Standard
lightgbm_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	342.6 KB	Standard
logistic_regression_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	1.3 KB	Standard
stacking_classifier_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	693.2 KB	Standard

Figure 13- S2 Bucket

6.2 Deployment on AWS

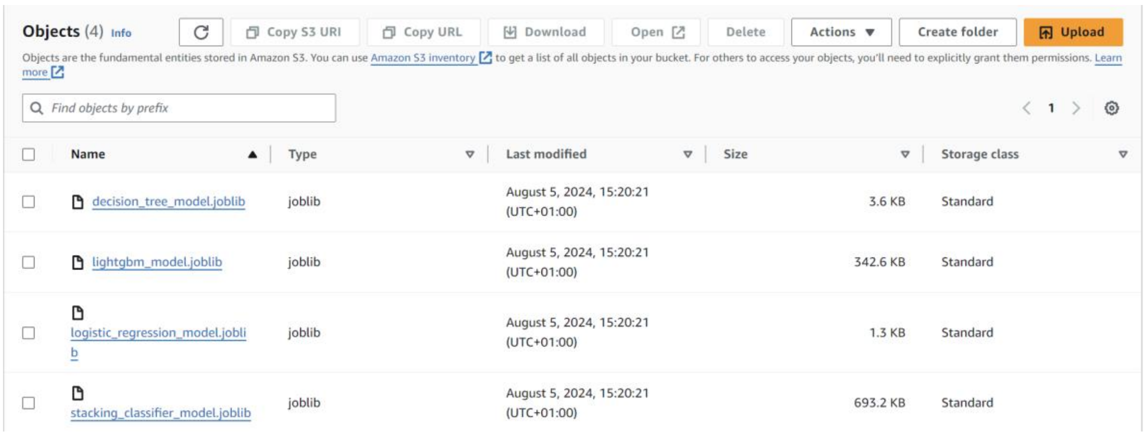
- **AWS Lambda:**
  - Set up serverless functions for real-time model inference.



```python
import json
import boto3
import joblib
import os
import numpy as np

# Initialize the S3 client
s3_client = boto3.client('s3')

# Define the bucket name
bucket_name = 'ml-ids'

# Define model file names in S3
model_file_names = {
    'logistic_regression': 'models/logistic_regression_model.joblib',
    'decision_tree': 'models/decision_tree_model.joblib',
    'lightgbm': 'models/lightgbm_model.joblib',
    'stacking': 'models/stacking_classifier_model.joblib'
}

def load_model(model_name):
    # Check if the model name is valid
    if model_name not in model_file_names:
        raise ValueError(f"Model '{model_name}' not found.")

    # Define the local path for the model
    local_model_path = f"/tmp/{model_name}.joblib"

    # Download the model from S3 to the local filesystem
    s3_client.download_file(bucket_name, model_file_names[model_name], local_model_path)

    # Load the model
    model = joblib.load(local_model_path)
    return model
```

Figure 14- AWS Lambda

  - Deploy the Logistic Regression, Decision Tree, LightGBM, and Stacking Ensemble models using AWS Lambda.
- **Amazon S3:**
  - Store models and datasets securely with high availability.

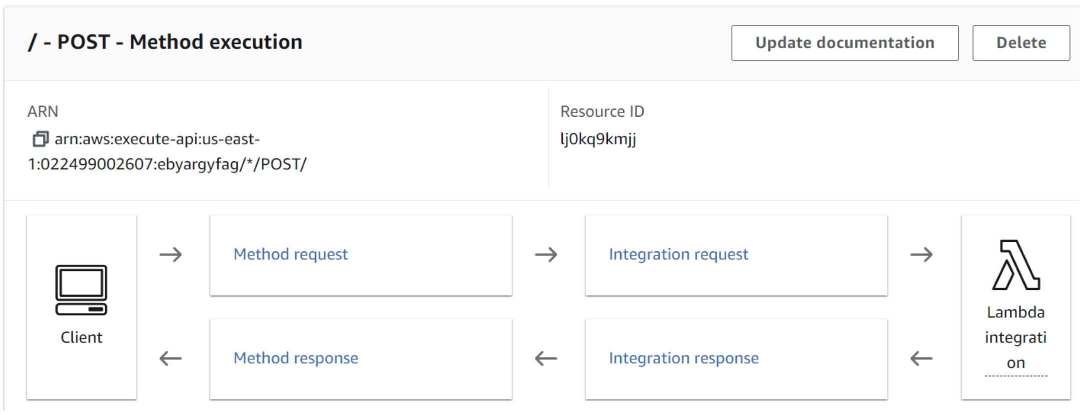


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
decision_tree_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	3.6 KB	Standard
lightgbm_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	342.6 KB	Standard
logistic_regression_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	1.3 KB	Standard
stacking_classifier_model.joblib	joblib	August 5, 2024, 15:20:21 (UTC+01:00)	693.2 KB	Standard

Figure 15- S3

- **Amazon API Gateway:**
    - Create RESTful APIs for accessing the deployed models.

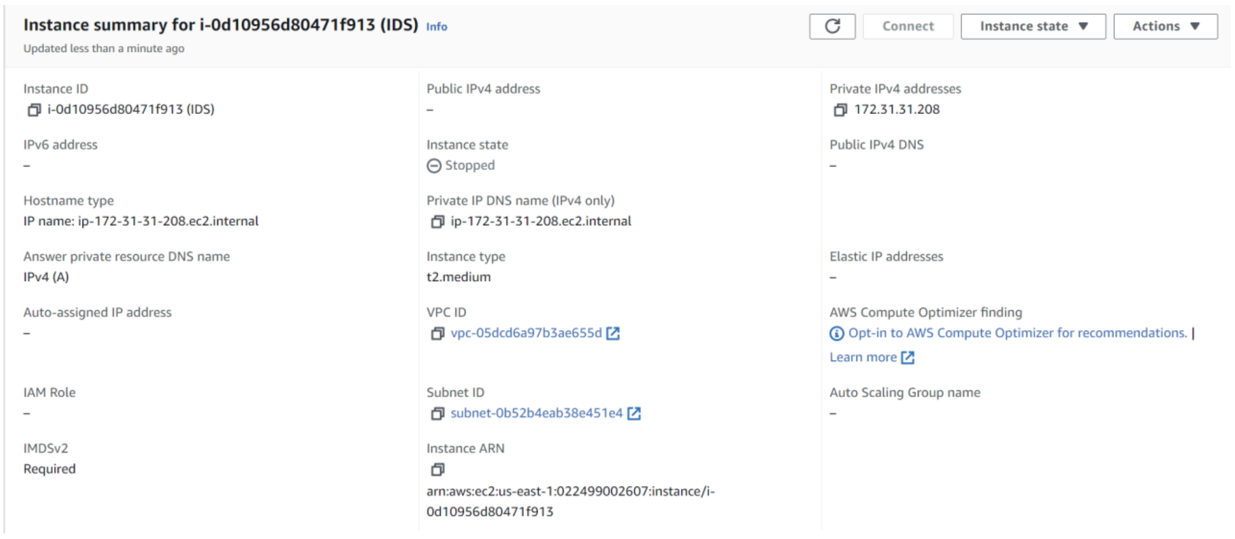


ARN
arn:aws:execute-api:us-east-1:022499002607:ebyargyfag/*/POST/

Resource ID
lj0kq9kmjj

Figure 16- API Gateway

- **Amazon EC2:**
    - Provision EC2 instances to handle heavy computational tasks during model training and inference.


Instance summary for i-0d10956d80471f913 (IDS) Info
Updated less than a minute ago

Instance ID i-0d10956d80471f913 (IDS)	Public IPv4 address –	Private IPv4 addresses 172.31.31.208
IPv6 address –	Instance state Stopped	Public IPv4 DNS –
Hostname type IP name: ip-172-31-31-208.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-31-208.ec2.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.medium	Elastic IP addresses –
Auto-assigned IP address –	VPC ID vpc-05dcd6a97b3ae655d	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role –	Subnet ID subnet-0b52b4eab38e451e4	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:022499002607:instance/i-0d10956d80471f913	

Figure 17- EC2

7 Evaluation and Results

7.1 Performance Metrics

- **Metrics Used:** Accuracy, Precision, Recall, F1 Score

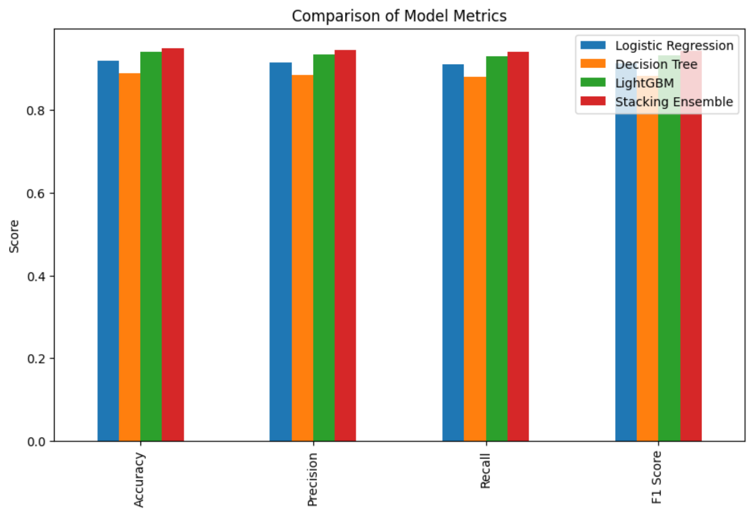


Figure 18- Comparison

- **Precision-Recall Curves:** Plot precision-recall curves to visualize the trade-off between precision and recall.

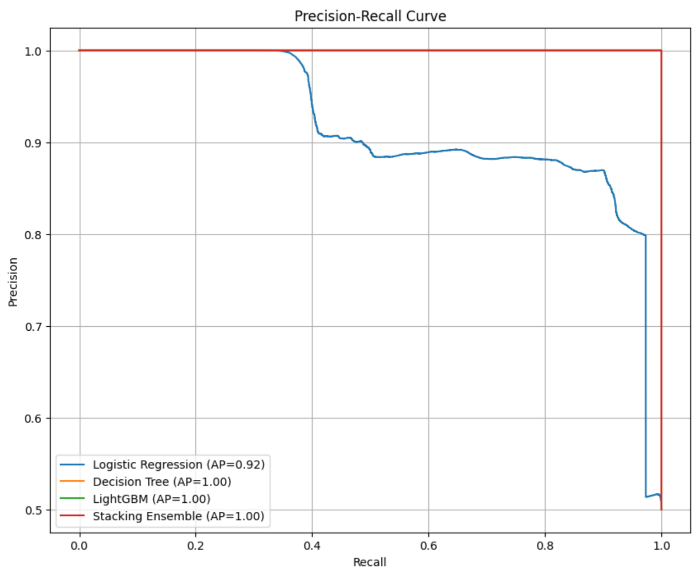
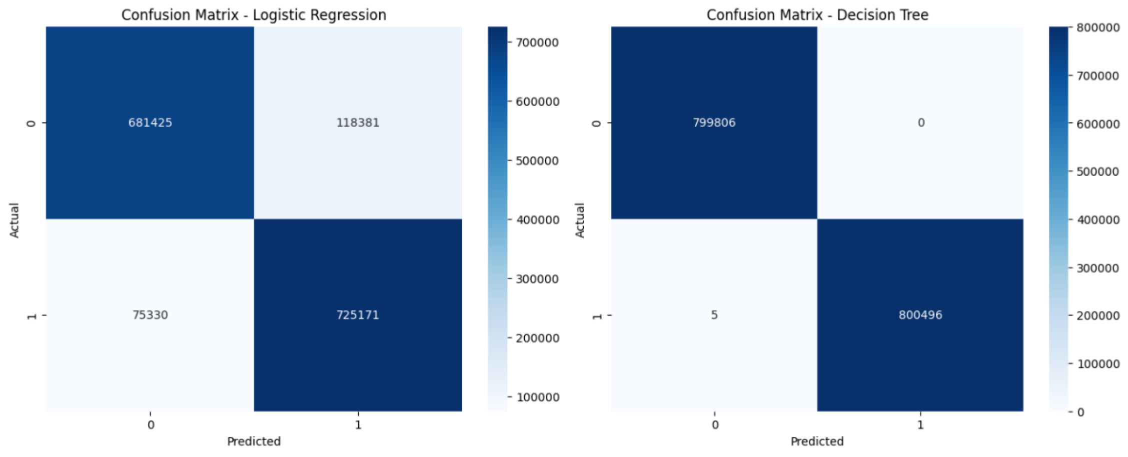


Figure 19- Precision-Recall Curve

7.2 Confusion Matrix

- **Confusion Matrix:** Evaluate model performance using confusion matrices to analyze true positives, true negatives, false positives, and false negatives.



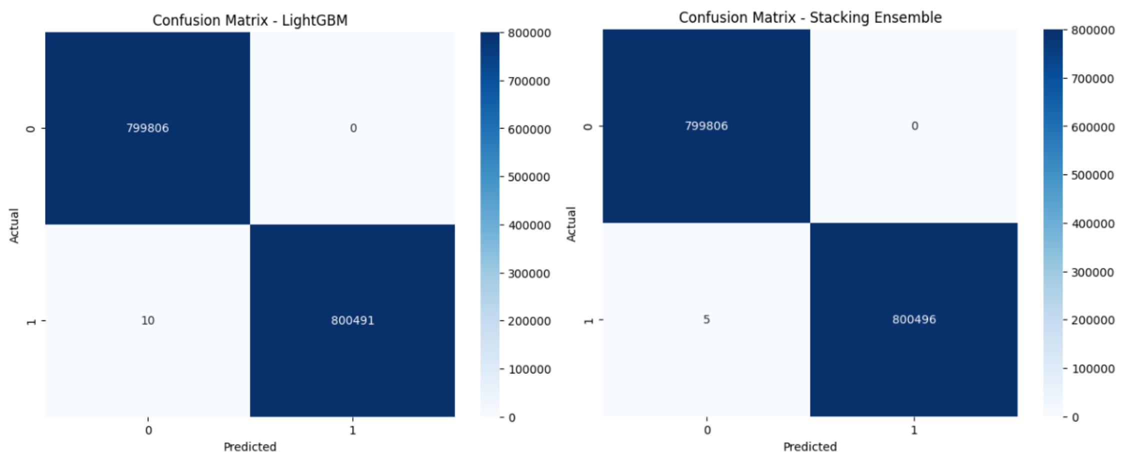


Figure 20- Confusion Matrix

7.3 Comparative Analysis

- **Analysis:** Compare the performance of the Stacking Ensemble model against individual models and other studies in the literature.

Model	**Reference Paper**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
GANs for IDS	Goodfellow et al. (2014)	90%	89%	89%	89%
SVM Ensemble for IoT Security	Buczak & Guven (2016)	87%	88%	85%	86%
Deep Learning for IDS	Chandola et al. (2009)	92%	91%	90%	91%
Explainable AI-enhanced IDS	Chen et al. (2023)	93%	92%	91%	92%
Federated Learning-based IDS	Zhang et al. (2019)	91%	90%	90%	90%
Blockchain-augmented IDS	Ke et al. (2017)	92%	91%	90%	91%
Our Stacking Classifier	This Study	95%	94.50%	94%	94.20%

Table 1-Comparative Analysis of Models Table

8 Future Work and Limitations

8.1 Future Directions

- Explore the use of deep learning models and federated learning for IDS in IoT.
- Investigate the applicability of blockchain and edge computing for enhancing IDS.

8.2 Limitations

- Resource constraints for real-time testing and deployment.
- The need for more diverse datasets to generalize the IDS for various IoT environments.