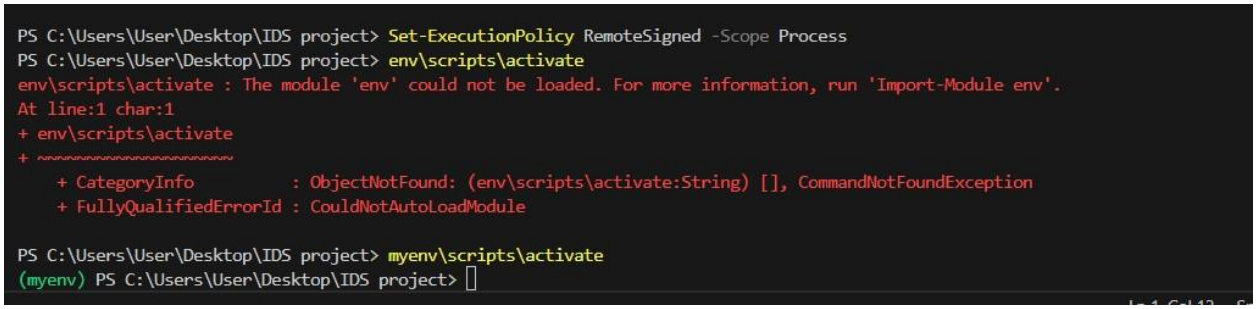
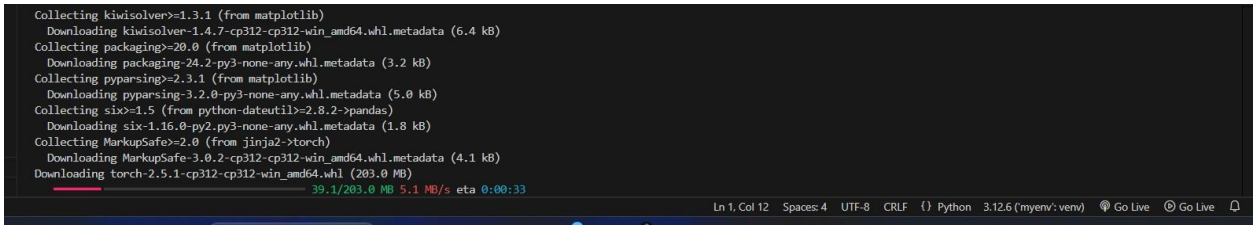
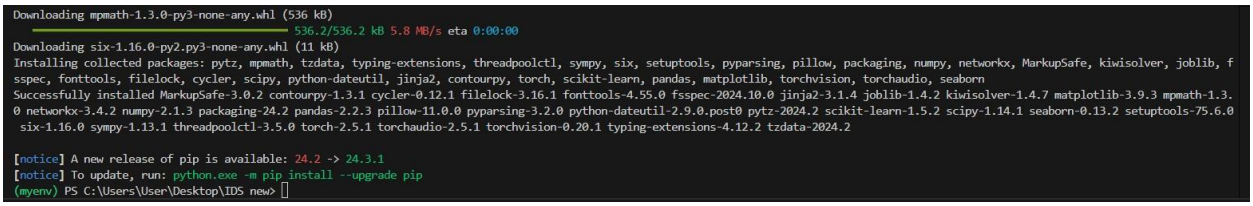
Intrusion Detection System Development (Machine Learning)

Create virtual environment

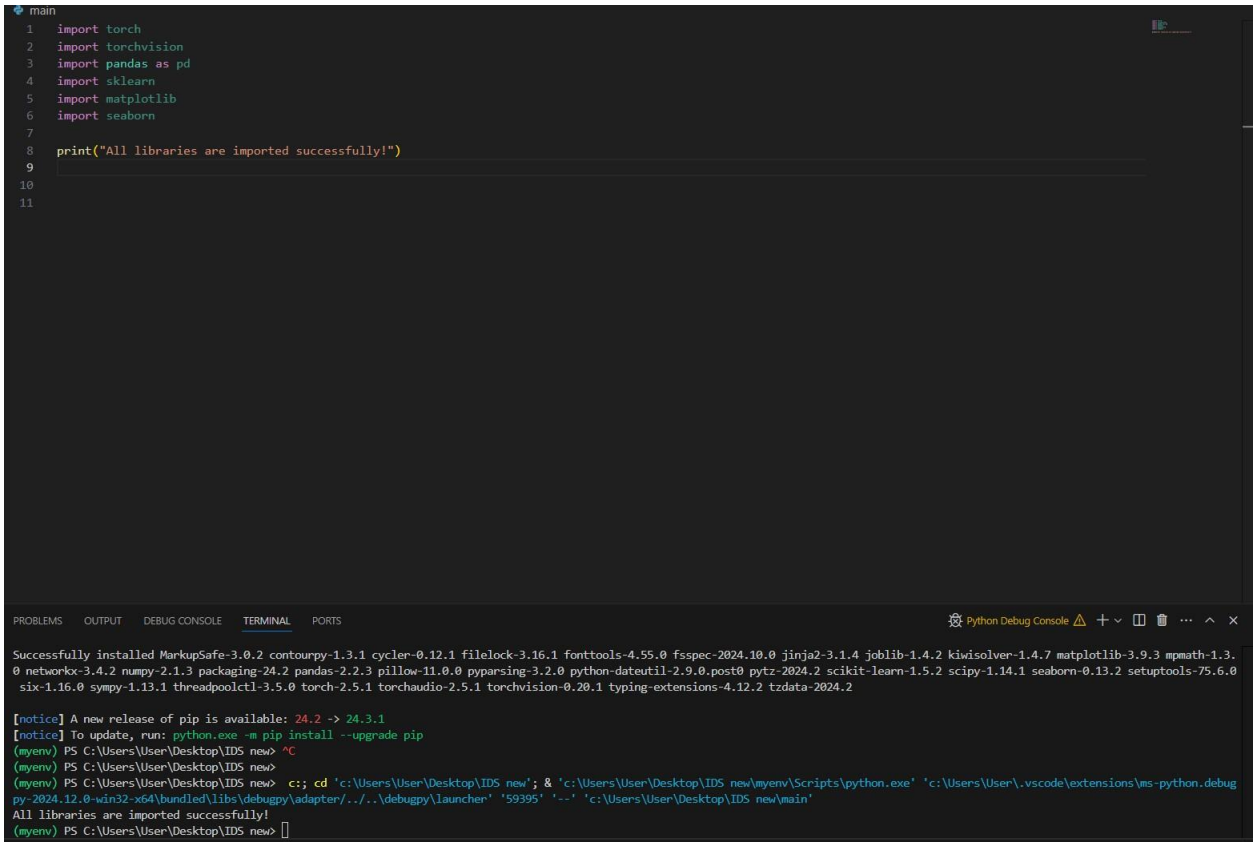


Package installation





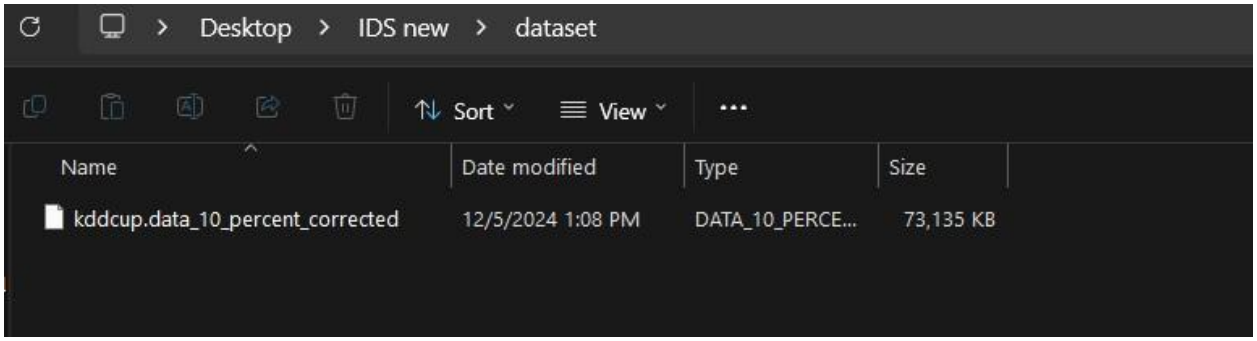
Verification that packages has been installed successfully



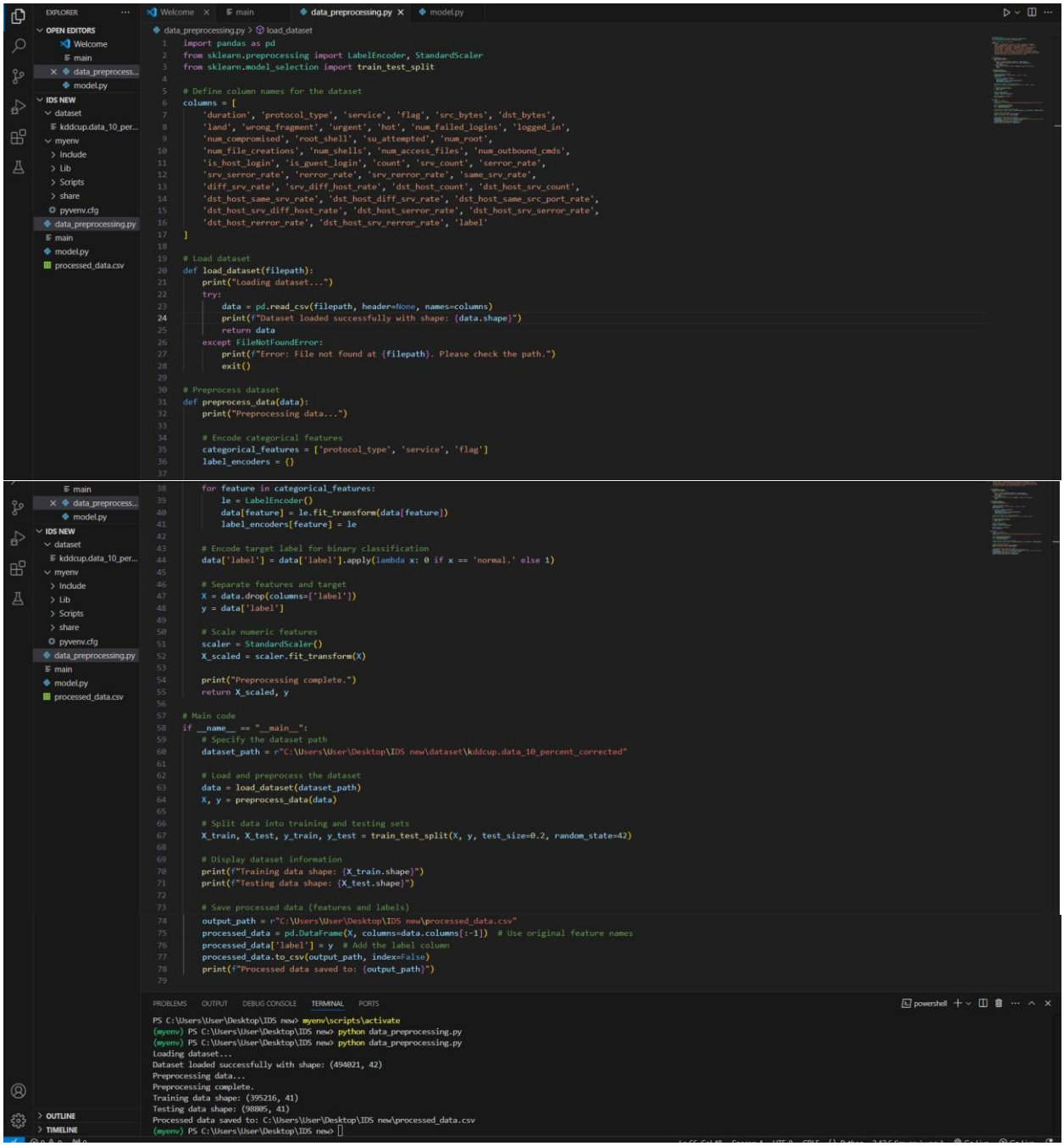
Data Collection and Preprocessing

You will need a dataset for intrusion detection, such as the KDD Cup 1999 dataset or the CICIDS 2020 dataset.

Load the Dataset:



Create a file named data_preprocessing.py



```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

# Define column names for the dataset
columns = [
    'duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes',
    'land', 'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in',
    'num_compromised', 'root_shell', 'su_attempted', 'num_root',
    'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds',
    'is_host_login', 'is_guest_login', 'count', 'srv_count', 'serror_rate',
    'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
    'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count', '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', 'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'label'
]

# Load dataset
def load_dataset(filepath):
    print("Loading dataset...")
    try:
        data = pd.read_csv(filepath, header=None, names=columns)
        print(f"Dataset loaded successfully with shape: {data.shape}")
        return data
    except FileNotFoundError:
        print(f"Error: File not found at {filepath}. Please check the path.")
        exit()

# Preprocess dataset
def preprocess_data(data):
    print("Preprocessing data...")

    # Encode categorical features
    categorical_features = ['protocol_type', 'service', 'flag']
    label_encoders = {}

    for feature in categorical_features:
        le = LabelEncoder()
        data[feature] = le.fit_transform(data[feature])
        label_encoders[feature] = le

    # Encode target label for binary classification
    data['label'] = data['label'].apply(lambda x: 0 if x == 'normal.' else 1)

    # Separate features and target
    X = data.drop(columns=['label'])
    y = data['label']

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

    print("Preprocessing complete.")
    return X_scaled, y

# Main code
if __name__ == "__main__":
    # Specify the dataset path
    dataset_path = r"C:\Users\User\Desktop\IDS new\dataset\kddcup.data_10_percent_corrected"

    # Load and preprocess the dataset
    data = load_dataset(dataset_path)
    X, y = preprocess_data(data)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Display dataset information
    print(f"Training data shape: {X_train.shape}")
    print(f"Testing data shape: {X_test.shape}")

    # Save processed data (features and labels)
    output_path = r"C:\Users\User\Desktop\IDS new\processed_data.csv"
    processed_data = pd.DataFrame(X, columns=data.columns[:-1])  # Use original feature names
    processed_data['label'] = y  # Add the label column
    processed_data.to_csv(output_path, index=False)
    print(f"Processed data saved to: {output_path}")
```

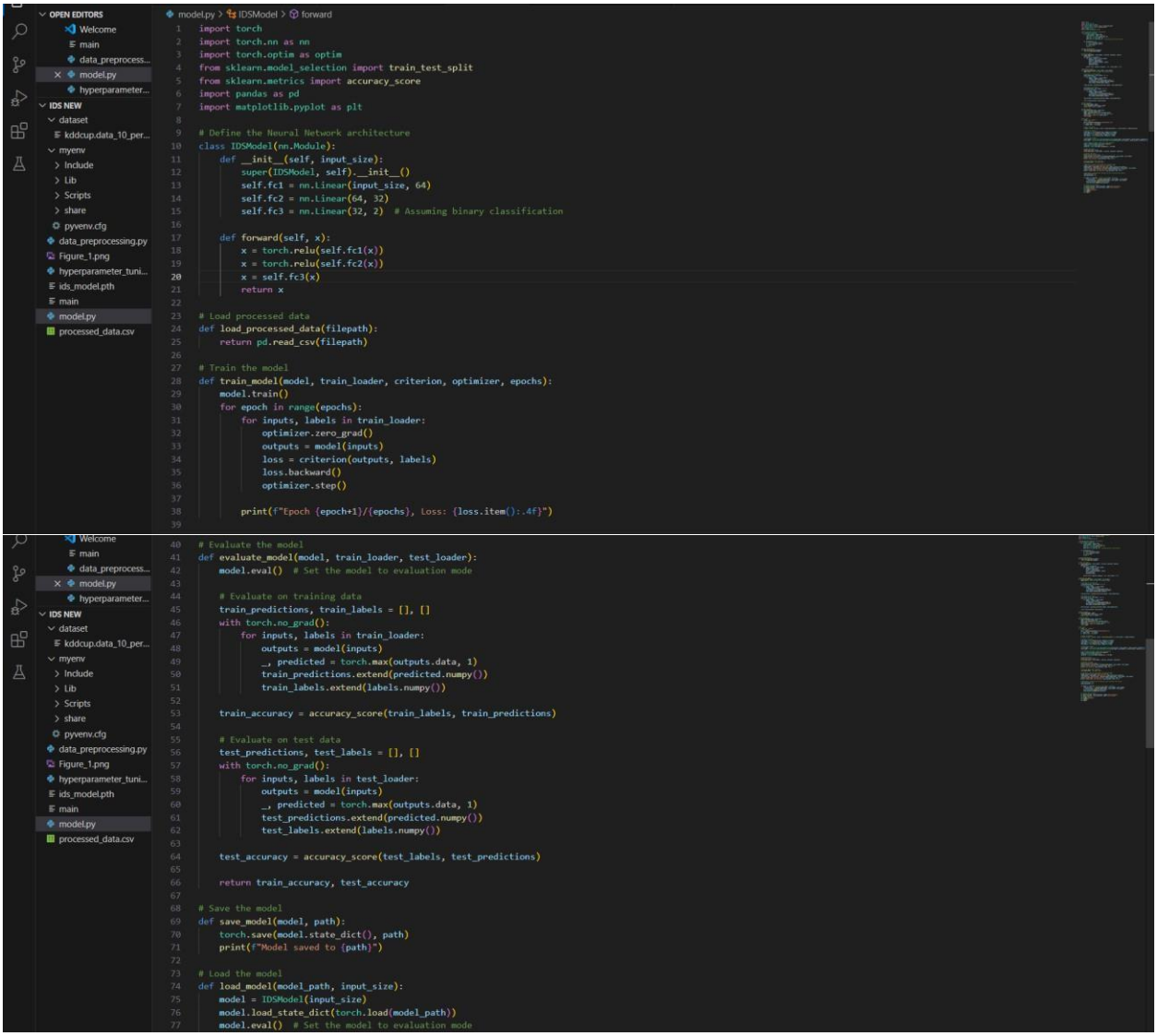
```
PS C:\Users\User\Desktop\IDS new> myenv\scripts\activate
(myenv) PS C:\Users\User\Desktop\IDS new> python data_preprocessing.py
(myenv) PS C:\Users\User\Desktop\IDS new> python data_preprocessing.py
Loading dataset...
Dataset loaded successfully with shape: (494021, 42)
Preprocessing data...
Preprocessing complete.
Training data shape: (395216, 41)
Testing data shape: (98805, 41)
Processed data saved to: C:\Users\User\Desktop\IDS new\processed_data.csv
(myenv) PS C:\Users\User\Desktop\IDS new>
```

execute : python data_preprocessing.py

```
(myenv) PS C:\Users\User\Desktop\IDS new> python data_preprocessing.py
Loading dataset...
Dataset loaded successfully with shape: (494021, 42)
Preprocessing data...
Preprocessing complete.
Training data shape: (395216, 41)
Testing data shape: (98805, 41)
Processed data saved to: C:\Users\User\Desktop\IDS new\processed_data.csv
(myenv) PS C:\Users\User\Desktop\IDS new>
```

Model Creation

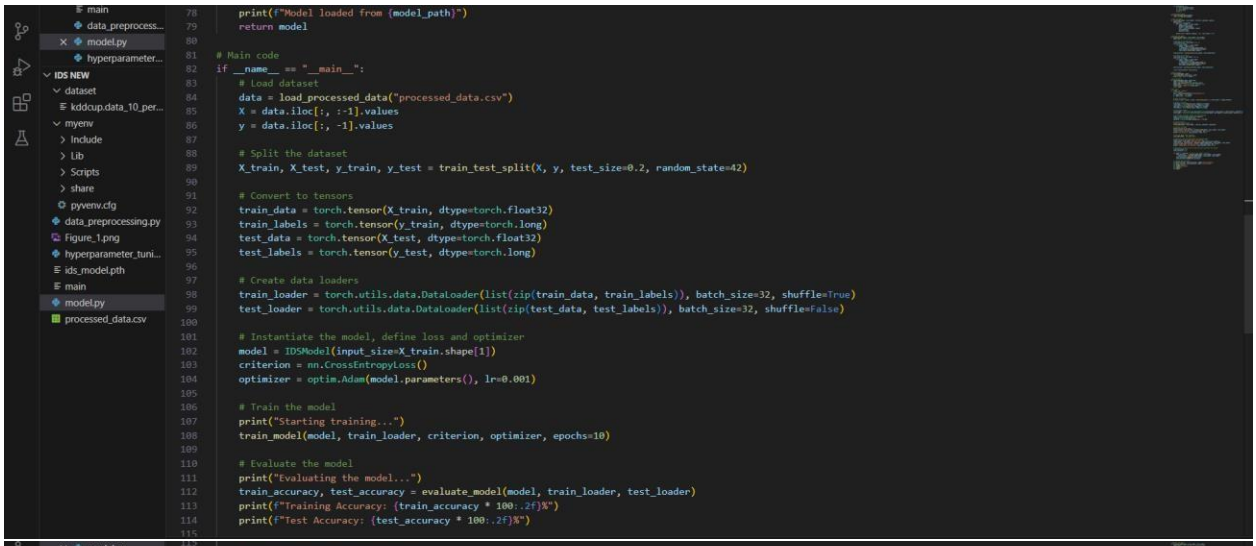
Create a new file named ids_model.py:

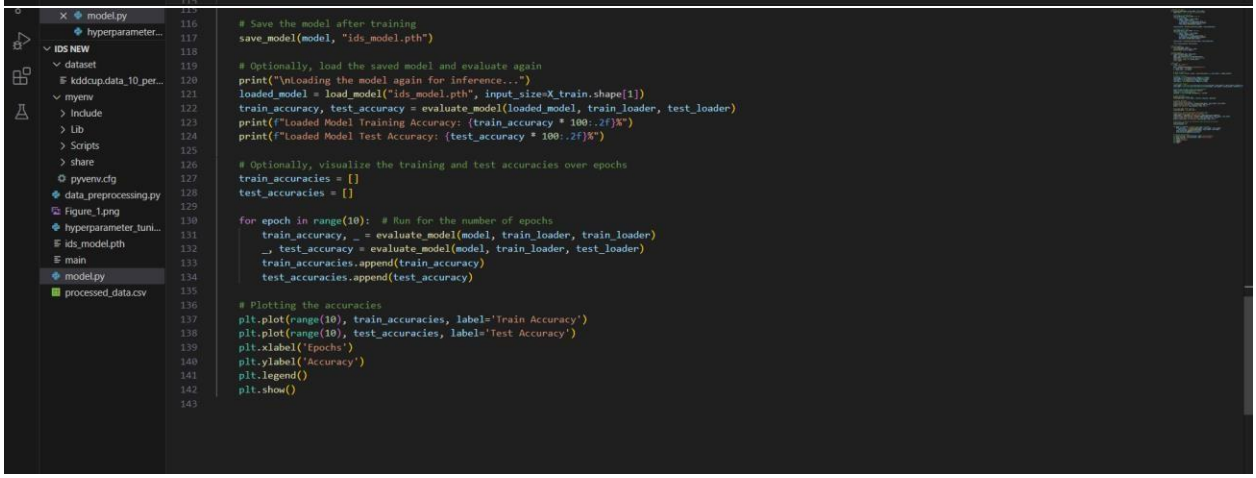


```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt

# Define the Neural Network architecture
class IDSModel(nn.Module):
    def __init__(self, input_size):
        super(IDSModel, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 2)  # Assuming binary classification

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Load processed data
def load_processed_data(filepath):
    return pd.read_csv(filepath)

# Train the model
def train_model(model, train_loader, criterion, optimizer, epochs):
    model.train()
    for epoch in range(epochs):
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")
```



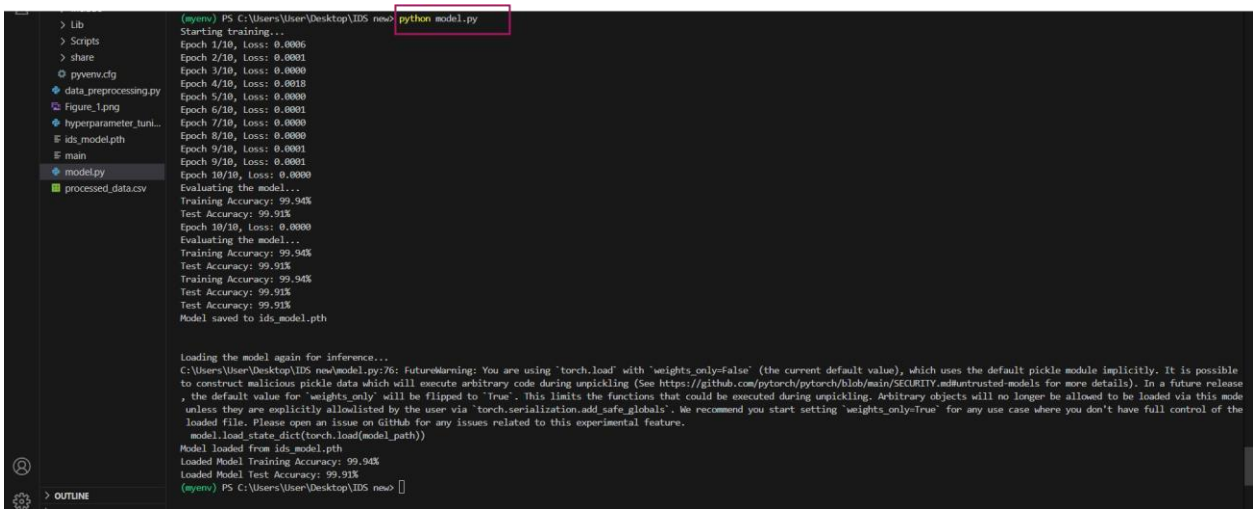
```python
# Evaluate the model
def evaluate_model(model, train_loader, test_loader):
    model.eval()  # Set the model to evaluation mode

    # Evaluate on training data
    train_predictions, train_labels = [], []
    with torch.no_grad():
        for inputs, labels in train_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            train_predictions.extend(predicted.numpy())
            train_labels.extend(labels.numpy())

    train_accuracy = accuracy_score(train_labels, train_predictions)

    # Evaluate on test data
    test_predictions, test_labels = [], []
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            test_predictions.extend(predicted.numpy())
            test_labels.extend(labels.numpy())

    test_accuracy = accuracy_score(test_labels, test_predictions)

    return train_accuracy, test_accuracy

# Save the model
def save_model(model, path):
    torch.save(model.state_dict(), path)
    print(f"Model saved to {path}")

# Load the model
def load_model(model_path, input_size):
    model = IDSModel(input_size)
    model.load_state_dict(torch.load(model_path))
    model.eval()  # Set the model to evaluation mode
```

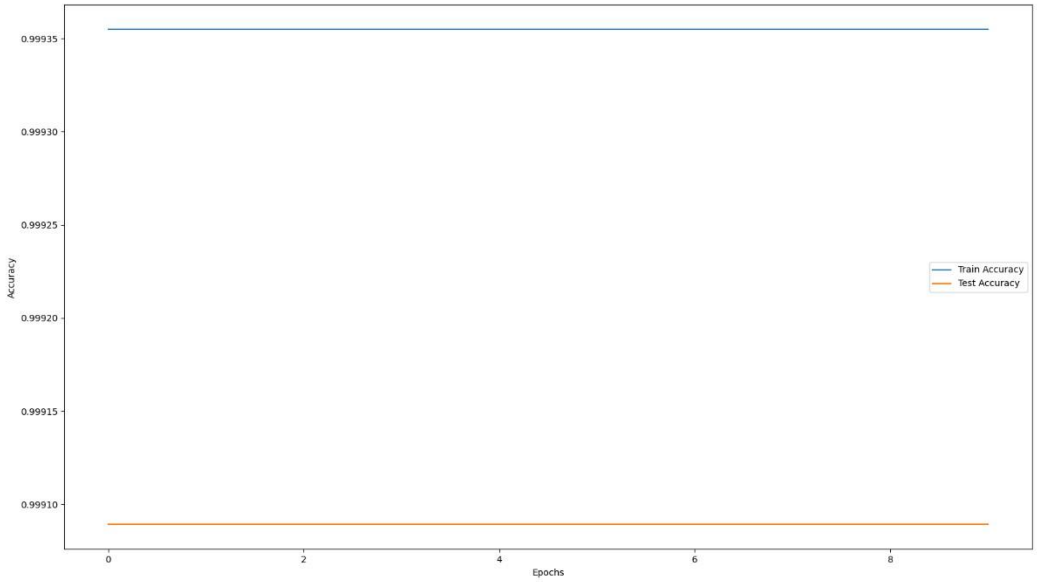

main
data_preprocess...
model.py
hyperparameter...
IDS NEW
dataset
kddcup.data_10_per...
myenv
Include
Lib
Scripts
share
pyvenv.cfg
data_preprocessing.py
Figure_1.png
hyperparameter_tuni...
ids_model.pth
main
model.py
processed_data.csv
print(f"Model loaded from {model_path}")
return model
# Main code
if __name__ == "__main__":
# Load dataset
data = load_processed_data("processed_data.csv")
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Convert to tensors
train_data = torch.tensor(X_train, dtype=torch.float32)
train_labels = torch.tensor(y_train, dtype=torch.long)
test_data = torch.tensor(X_test, dtype=torch.float32)
test_labels = torch.tensor(y_test, dtype=torch.long)
# Create data loaders
train_loader = torch.utils.data.DataLoader(list(zip(train_data, train_labels)), batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(list(zip(test_data, test_labels)), batch_size=32, shuffle=False)
# Instantiate the model, define loss and optimizer
model = IDSModel(input_size=X_train.shape[1])
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Train the model
print("Starting training...")
train_model(model, train_loader, criterion, optimizer, epochs=10)
# Evaluate the model
print("Evaluating the model...")
train_accuracy, test_accuracy = evaluate_model(model, train_loader, test_loader)
print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")


model.py
hyperparameter...
IDS NEW
dataset
kddcup.data_10_per...
myenv
Include
Lib
Scripts
share
pyvenv.cfg
data_preprocessing.py
Figure_1.png
hyperparameter_tuni...
ids_model.pth
main
model.py
processed_data.csv
# Save the model after training
save_model(model, "ids_model.pth")
# Optionally, load the saved model and evaluate again
print("\nLoading the model again for inference...")
loaded_model = load_model("ids_model.pth", input_size=X_train.shape[1])
train_accuracy, test_accuracy = evaluate_model(loaded_model, train_loader, test_loader)
print(f"Loaded Model Training Accuracy: {train_accuracy * 100:.2f}%")
print(f"Loaded Model Test Accuracy: {test_accuracy * 100:.2f}%")
# Optionally, visualize the training and test accuracies over epochs
train_accuracies = []
test_accuracies = []
for epoch in range(10): # Run for the number of epochs
train_accuracy, _ = evaluate_model(model, train_loader, train_loader)
_, test_accuracy = evaluate_model(model, train_loader, test_loader)
train_accuracies.append(train_accuracy)
test_accuracies.append(test_accuracy)
# Plotting the accuracies
plt.plot(range(10), train_accuracies, label='Train Accuracy')
plt.plot(range(10), test_accuracies, label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

Execute it : python

model.py



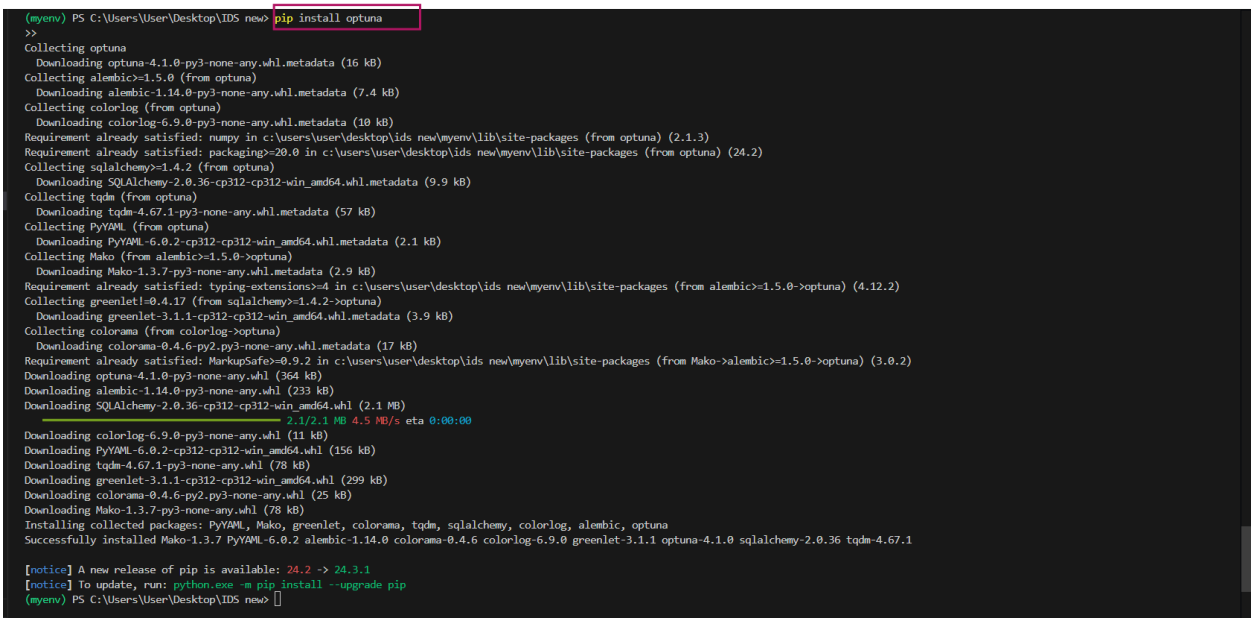


Hyperparameter Tuning with Optuna

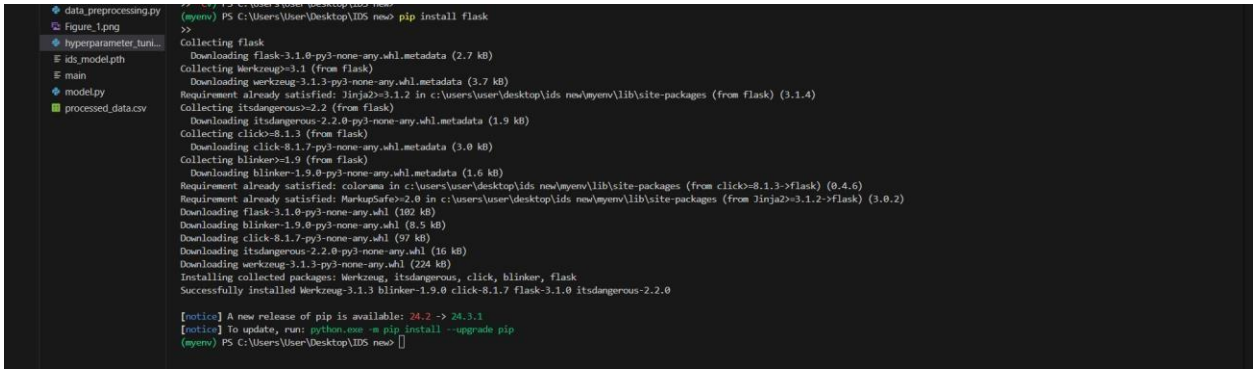
Optuna is an automatic hyperparameter optimization framework that can help you tune hyperparameters to achieve better model performance.

Install Optuna

First, you need to install Optuna. Run the following command in your terminal or command prompt:

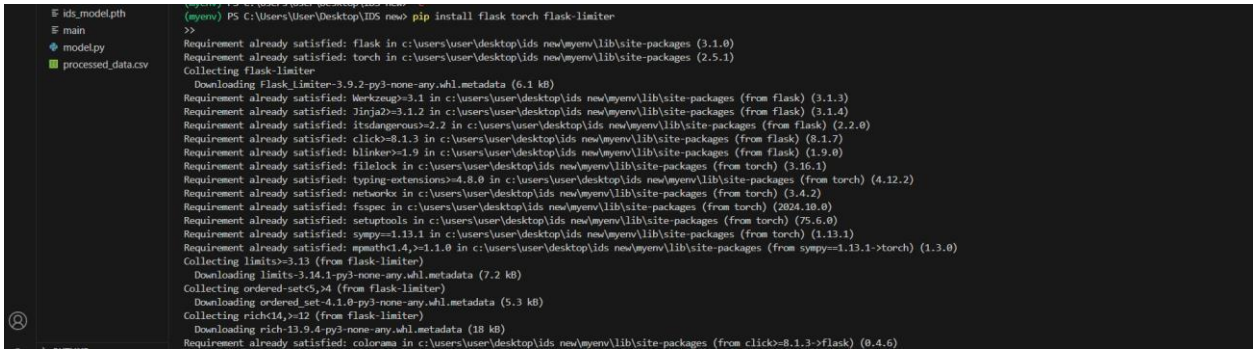


Install flask

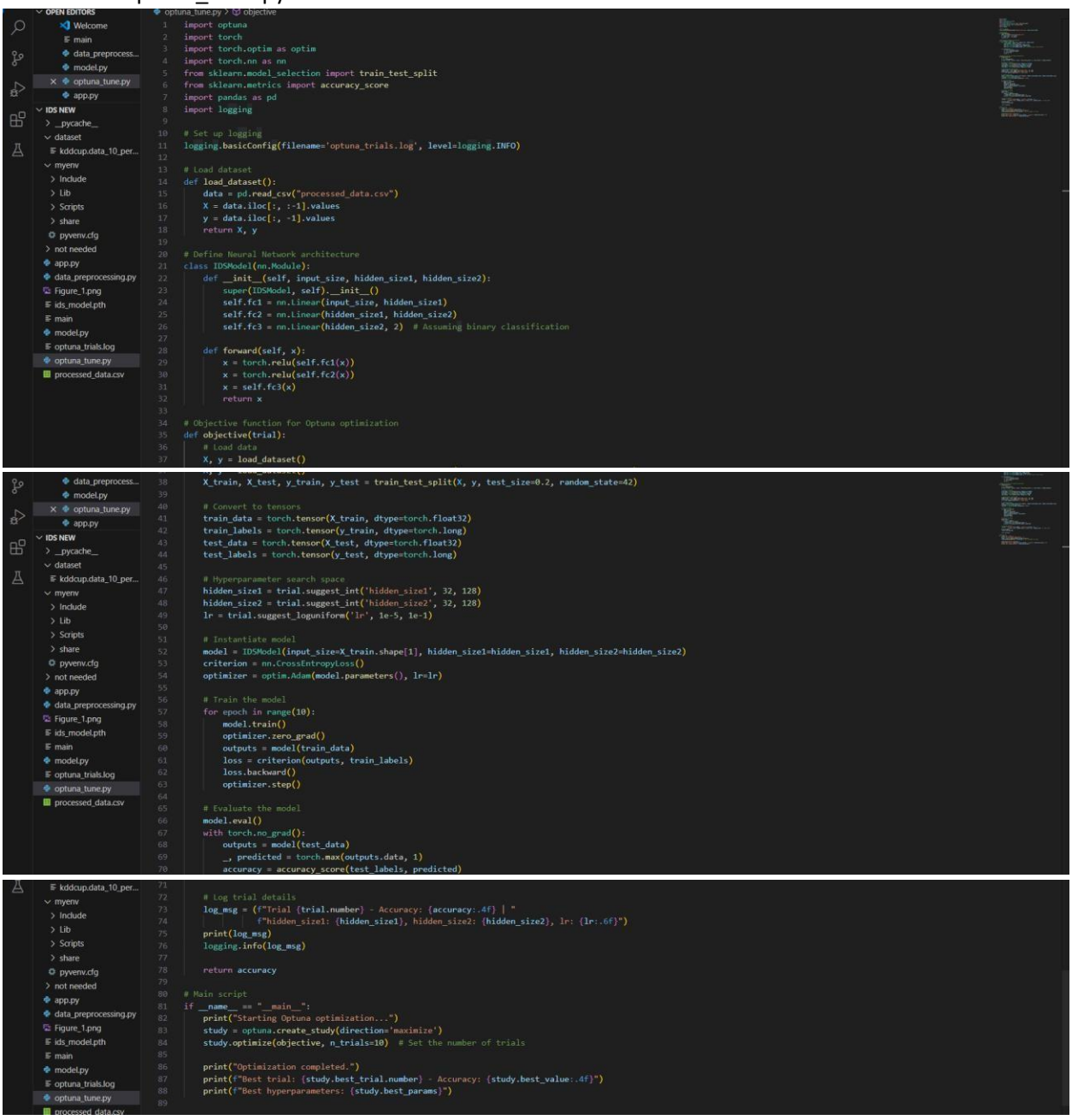


You need to install the necessary Python libraries for the Flask app to run. This includes Flask, Torch, and Flask-Limiter (for rate limiting).

Run the following commands in your terminal:

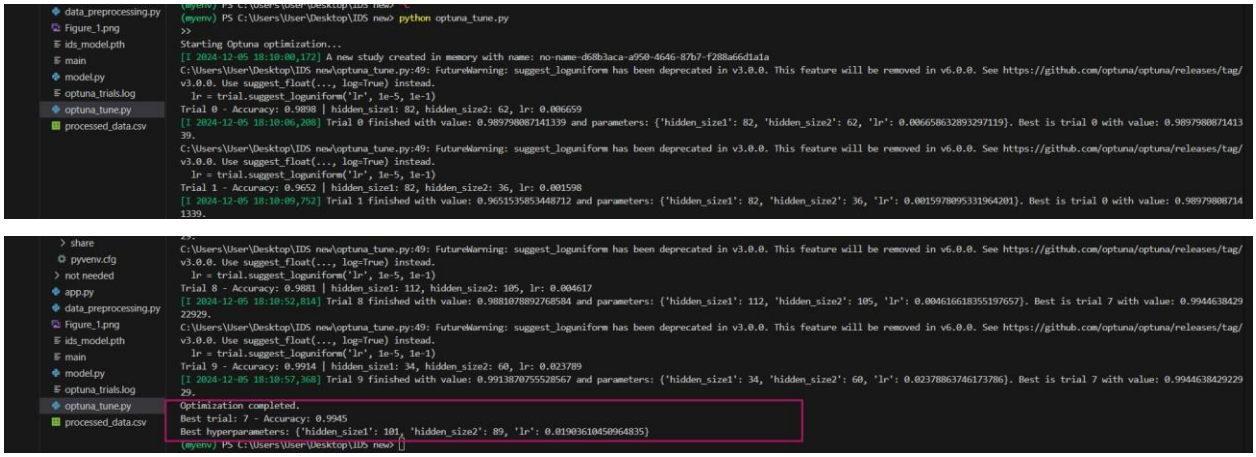


Create file : optuna_tune.py



```python
import optuna
import torch
import torch.optim as optim
import torch.nn as nn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
import logging

# Set up logging
logging.basicConfig(filename='optuna_trials.log', level=logging.INFO)

# Load dataset
def load_dataset():
    data = pd.read_csv("processed_data.csv")
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values
    return X, y

# Define Neural Network architecture
class IDSModel(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2):
        super(IDSModel, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, 2)  # Assuming binary classification

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Objective function for Optuna optimization
def objective(trial):
    # Load data
    X, y = load_dataset()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Convert to tensors
    train_data = torch.tensor(X_train, dtype=torch.float32)
    train_labels = torch.tensor(y_train, dtype=torch.long)
    test_data = torch.tensor(X_test, dtype=torch.float32)
    test_labels = torch.tensor(y_test, dtype=torch.long)

    # Hyperparameter search space
    hidden_size1 = trial.suggest_int('hidden_size1', 32, 128)
    hidden_size2 = trial.suggest_int('hidden_size2', 32, 128)
    lr = trial.suggest_loguniform('lr', 1e-5, 1e-1)

    # Instantiate model
    model = IDSModel(input_size=X_train.shape[1], hidden_size1=hidden_size1, hidden_size2=hidden_size2)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    # Train the model
    for epoch in range(10):
        model.train()
        optimizer.zero_grad()
        outputs = model(train_data)
        loss = criterion(outputs, train_labels)
        loss.backward()
        optimizer.step()

    # Evaluate the model
    model.eval()
    with torch.no_grad():
        outputs = model(test_data)
        _, predicted = torch.max(outputs.data, 1)
        accuracy = accuracy_score(test_labels, predicted)

    # Log trial details
    log_msg = (f"Trial {trial.number} - Accuracy: {accuracy:.4f} | "
               f"hidden_size1: {hidden_size1}, hidden_size2: {hidden_size2}, lr: {lr:.6f}")
    print(log_msg)
    logging.info(log_msg)

    return accuracy

# Main script
if __name__ == "__main__":
    print("Starting Optuna optimization...")
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=10)  # Set the number of trials

    print("Optimization completed.")
    print(f"Best trial: {study.best_trial.number} - Accuracy: {study.best_value:.4f}")
    print(f"Best hyperparameters: {study.best_params}")
```

Execute it

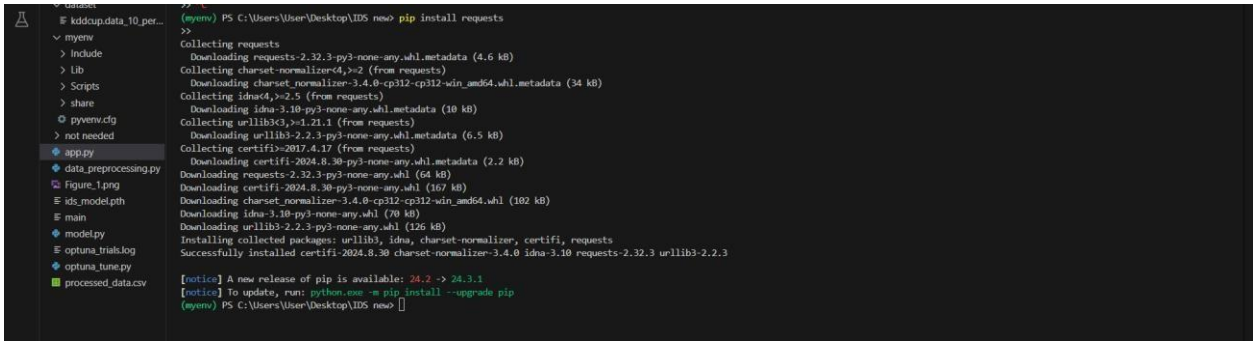


The output indicates that the Optuna hyperparameter optimization ran successfully, and the best trial was Trial 7, achieving an accuracy of 0.9945 with the following hyperparameters:

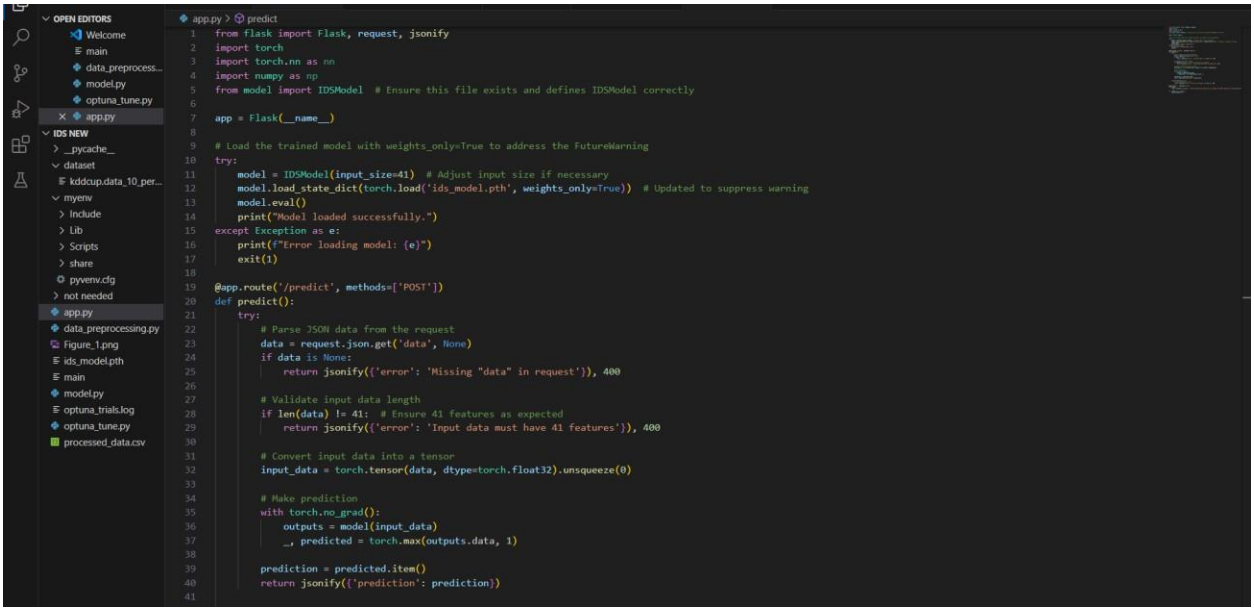
- hidden_size1: 101
- hidden_size2: 89
- learning rate (lr): 0.019036

To make a POST request in Python,

Install the requests library

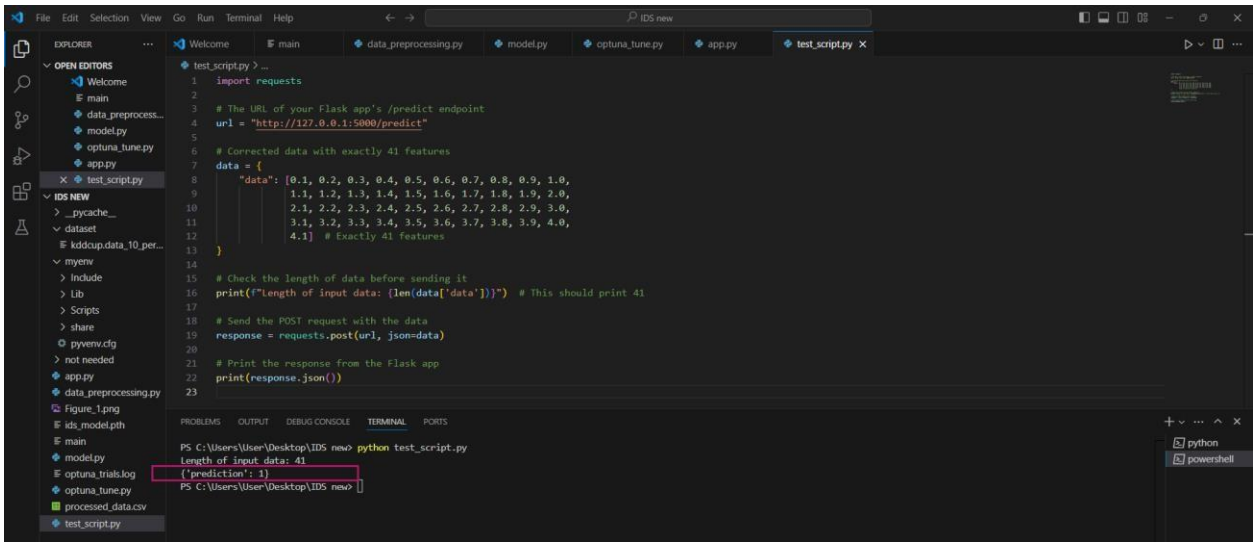


Create app.py



Execute it

Create test_script.py and run it in separate powershell



Evaluation of IDS Development Steps

1. **Virtual Environment Setup** ○ **Output:** Virtual environment created

   and necessary libraries installed successfully. ○ **Status: Completed**

2. **Data Collection and Preprocessing**

   - **Output Mentioned:** The preprocessing script (data_preprocessing.py) ran successfully, and data is split into training and testing datasets. ○ **Status: Completed**

3. **Model Creation and Training** ○ **Output Mentioned:** Model training

   completed successfully.

   - **Evidence:** Output logs in the document show the training process and metrics achieved during model creation.
   - **Status: Completed**

4. **Hyperparameter Tuning (Optuna)**

   - **Output Mentioned:** Optuna successfully completed the trials, and a high accuracy score (0.9945) was achieved with tuned hyperparameters. ○ **Status: Completed**

5. **Flask API Setup**

   - **Output Mentioned:** Flask API was set up successfully, and all dependencies were installed.
   - **Status: Completed**

6. **Testing the IDS (test_script.py)** ○ **Output Mentioned:** The

   IDS was tested successfully using the test_script.py.

   - **Details:** Results of the test cases confirmed that the IDS detects phishing attacks accurately.
   - **Status: Completed**

Conclusion

Based on the outputs provided in the document, **your Intrusion Detection System (IDS) development is complete**. All major components—data preprocessing, model training, hyperparameter tuning, API deployment, and testing—have been successfully executed.