**DEEP LEARNING MODEL CODE**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import seaborn as sns

from sklearn.metrics import mean\_squared\_error

# Load the dataset

data\_path = 'E:/CYBER SECURITY/PROJECT/jena\_climate\_2009\_2016.csv'

data = pd.read\_csv(data\_path)

# Select relevant columns and preprocess data

relevant\_cols = ['Date Time', 'p (mbar)', 'T (degC)', 'Tpot (K)', 'Tdew (degC)', 'rh (%)', 'VPmax (mbar)', 'VPact (mbar)', 'VPdef (mbar)', 'sh (g/kg)', 'H2OC (mmol/mol)', 'rho (g/m\*\*3)', 'wv (m/s)', 'max. wv (m/s)', 'wd (deg)']

data = data[relevant\_cols]

data.dropna(inplace=True)

data['Date Time'] = pd.to\_datetime(data['Date Time'])

df = pd.read\_csv(data\_path)

titles = [

"Pressure",

"Temperature",

"Temperature in Kelvin",

"Temperature (dew point)",

"Relative Humidity",

"Saturation vapor pressure",

"Vapor pressure",

"Vapor pressure deficit",

"Specific humidity",

"Water vapor concentration",

"Airtight",

"Wind speed",

"Maximum wind speed",

"Wind direction in degrees",

]

feature\_keys = [

"p (mbar)",

"T (degC)",

"Tpot (K)",

"Tdew (degC)",

"rh (%)",

"VPmax (mbar)",

"VPact (mbar)",

"VPdef (mbar)",

"sh (g/kg)",

"H2OC (mmol/mol)",

"rho (g/m\*\*3)",

"wv (m/s)",

"max. wv (m/s)",

"wd (deg)",

]

colors = [

"blue",

"orange",

"green",

"red",

"purple",

"brown",

"pink",

"gray",

"olive",

"cyan",

]

def show\_heatmap(data):

plt.matshow(data.corr())

plt.xticks(range(data.shape[1]), data.columns, fontsize=14, rotation=90)

plt.gca().xaxis.tick\_bottom()

plt.yticks(range(data.shape[1]), data.columns, fontsize=14)

cb = plt.colorbar()

cb.ax.tick\_params(labelsize=14)

plt.title("Feature Correlation Heatmap", fontsize=14)

plt.show()

show\_heatmap(df)

# Prepare features and target

sequence\_length = 72 # Number of time steps in the input sequence (past 3 days)

X = []

y = []

for i in range(len(data) - sequence\_length):

X.append(data[relevant\_cols[2]].values[i:i+sequence\_length])

y.append(data[relevant\_cols[2]].values[i+sequence\_length])

X = np.array(X)

y = np.array(y)

# 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)

# Standardize the data

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Reshape data for LSTM input (samples, time steps, features)

X\_train = X\_train.reshape((X\_train.shape[0], sequence\_length, 1))

X\_test = X\_test.reshape((X\_test.shape[0], sequence\_length, 1))

# Build the LSTM model

model = Sequential()

model.add(LSTM(64, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dense(1, activation='linear')) # Linear activation for regression

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Print the model summary

model.summary()

# Plot the training and validation loss

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

print("-------------------------------------------------")

# Make predictions on the test data

y\_pred = model.predict(X\_test).flatten()

# Calculate mean squared error on the test set

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error on Test Set:", mse)

print("-------------------------------------------------")

# Calculate root mean squared error (RMSE)

rmse = np.sqrt(mse)

print("Root Mean Squared Error on Test Set:", rmse)

print("-------------------------------------------------")

# Calculate mean absolute error (MAE)

mae = np.mean(np.abs(y\_test - y\_pred))

print("Mean Absolute Error on Test Set:", mae)

print("-------------------------------------------------")

# Calculate R-squared (R2) score

total\_variance = np.var(y\_test)

r2 = 1 - (mse / total\_variance)

print("R-squared (R2) Score on Test Set:", r2)

print("-------------------------------------------------")

# Classify the predictions as normal (0) or anomalous (1) based on the threshold

scaling\_factor = 1.0

threshold = np.mean(np.abs(y\_test - y\_pred)) \* scaling\_factor

y\_pred\_class = np.where(np.abs(y\_test - y\_pred) > threshold, 1, 0)

y\_test\_class = np.where(np.abs(y\_test - y\_pred) > threshold, 1, 0)

print(":")

print(":")

print(":")

# Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test\_class, y\_pred\_class)

# Display the confusion matrix

print("Confusion Matrix:")

print(conf\_matrix)

# Plot the confusion matrix

plt.figure(figsize=(5, 5))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues")

plt.title("Confusion Matrix")

plt.xlabel("Predicted Class")

plt.ylabel("True Class")

plt.show()

print(":")

print(":")

print(":")

# Print the real and predicted values

print("Real , Predicted and Real vs Predicted:")

print("-------------------------------------------------")

print("{:<10} {:<10} {:<10}".format("Real", "Predicted", "Real vs Predicted"))

for i in range(min(len(y\_test), 10)):

print("{:<10.2f} {:<10.2f} {:<10.2f}".format(y\_test[i], y\_pred[i], y\_test[i] - y\_pred[i]))

print(":")

print(":")

print(":")

# Classify the predictions as normal (0) or anomalous (1) based on the threshold

y\_pred\_class = np.where(np.abs(y\_test - y\_pred) > threshold, 1, 0)

# Get the indices of the detected anomalies

attack\_indices = np.where(y\_pred\_class == 1)[0]

print(":")

print(":")

print(":")

# Print attacker data vs predicted

print("\nAttacker Data vs Predicted (First 10 lines):")

print("-------------------------------------------------")

for idx in attack\_indices[:10]:

attacker\_value = y\_test[idx]

predicted\_value = y\_pred[idx]

attacker\_vs\_predicted = attacker\_value - predicted\_value

print("{:<10.2f} {:<10.2f}".format(attacker\_value, attacker\_vs\_predicted))

print(":")

print(":")

print(":")

# Set print options to display all elements of the array

#np.set\_printoptions(threshold=np.inf)

print("Detected Attack Indices:")

print("------------------------------------------------------------------------------------")

#print(attack\_indices)

print(attack\_indices)

# Reset print options to default

#np.set\_printoptions(threshold=5)

print(":")

print(":")

print(":")

# Denoise data by removing detected anomalies

denoised\_data = np.delete(X\_test, attack\_indices, axis=0)

print("Denoised Data:")

print(denoised\_data)