Open Ended Lab

Lab Title: **Building model for forecasting using the CityLearn dataset**

Objective
The objective of this lab is to build a complete time-series forecasting pipeline using the CityLearn dataset. The aim is to:

- Understand and handle real-world data by performing comprehensive data preprocessing, including missing value handling, outlier treatment, and feature engineering.
- Combine multiple data sources (building energy usage, weather, pricing, carbon intensity) to create a multi-feature dataset.
- Apply cyclic encoding, one-hot encoding, and normalization to prepare the dataset for machine learning.
- Train an LSTM (Long Short-Term Memory) neural network model to predict future electricity load values.
- Evaluate the model's performance using metrics like Root Mean Squared Error (RMSE) and visualize training progress.

1. Introduction
In this open-ended lab, we work with the CityLearn dataset, which consists of multiple data sources including Building 2 electricity load data, carbon intensity, energy pricing, and weather conditions. The objective is to merge these datasets, preprocess the data, and train a predictive model using an LSTM network.

2. Dataset Description
The dataset includes hourly time-series data from the CityLearn environment with the following components:

- Building 2: Contains non-shiftable electrical load data.
- Carbon Intensity: Reflects the CO2 emissions of electricity generation.
- Pricing: Hourly energy pricing data.
- Weather: Includes temperature, humidity, solar generation, and wind speed.

3. Data Preprocessing
The preprocessing pipeline includes the following steps:

- Missing Data Handling: Forward-fill, backward-fill, and column dropping for excessive NaNs.
- Outlier Detection: IQR-based outlier capping was applied to numeric features.

- Holiday Feature: A binary 'is_holiday' column was introduced using the U.S. federal holiday calendar.
- Feature Engineering: Included cyclic encoding (hour, month) and one-hot encoding for 'day of week'.
- Normalization: Features were scaled using `StandardScaler`.
- Data Split: The dataset was divided into train (70%), validation (20%), and test (10%) splits.

4. LSTM Model
An LSTM model is implemented for time-series prediction using the following structure:

- LSTM Layer: 64 units with tanh activation.
- Dense Layer: 32 units with ReLU activation.
- Output Layer: Single neuron to predict the load.

5. Code Implementation
The following code is used to implement and train the LSTM model on the preprocessed dataset:

Code:

```
import numpy as np

import pandas as pd

import os

import joblib

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from sklearn.metrics import mean_squared_error



print("✅ Libraries loaded successfully.")
```

```
# === CONFIGURATION ===

LOOK_BACK = 24

TARGET_COL = 0  # Index of the target column

EPOCHS = 10

BATCH_SIZE = 32

PATIENCE = 10


DATA_DIR = r"C:\Users\PMLS\ml\Machine Learning\Machine Learning lab\Open ended lab"

MODEL_PATH = os.path.join(DATA_DIR, "lstm_model.h5")

CHECKPOINT_PATH = os.path.join(DATA_DIR, "E1-cp-best.h5")


print("✅ Configuration set.")


# === DATA LOADING ===

def load_dataset(path):

   return pd.read_csv(path)


train_df = load_dataset(os.path.join(DATA_DIR, "train_data.csv"))

val_df   = load_dataset(os.path.join(DATA_DIR, "val_data.csv"))

test_df  = load_dataset(os.path.join(DATA_DIR, "test_data.csv"))


print("✅ Data loaded successfully.")
```

```
train_df.head()


# === SEQUENCE CREATION ===
def create_sequences(df, look_back, target_idx):
    X, y = [], []
    data = df.values
    for i in range(len(data) - look_back):
        X.append(data[i:i + look_back])
        y.append(data[i + look_back, target_idx])
    return np.array(X), np.array(y)


X_train, y_train = create_sequences(train_df, LOOK_BACK, TARGET_COL)
X_val, y_val     = create_sequences(val_df, LOOK_BACK, TARGET_COL)
X_test, y_test   = create_sequences(test_df, LOOK_BACK, TARGET_COL)


print(f"Train shape: {X_train.shape}, {y_train.shape}")
print(f"Validation shape: {X_val.shape}, {y_val.shape}")
print(f"Test shape: {X_test.shape}, {y_test.shape}")


# === MODEL BUILDING ===
model = Sequential([
    LSTM(64, activation='tanh', input_shape=(LOOK_BACK,
```

```
X_train.shape[2])),
    Dense(32, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')

print("✅ Model compiled.")
model.summary()

# === CALLBACKS ===
early_stop = EarlyStopping(monitor='val_loss', patience=PATIENCE, restore_best_weights=True)
checkpoint_cb = ModelCheckpoint(
    filepath=CHECKPOINT_PATH,
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

print("✅ Callbacks initialized.")

# === TRAINING ===
history = model.fit(
```

```
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[early_stop, checkpoint_cb],
    verbose=1
)

print("✅ Training complete.")

# === EVALUATION ===
val_loss = model.evaluate(X_val, y_val, verbose=0)
test_loss = model.evaluate(X_test, y_test, verbose=0)

val_rmse = np.sqrt(val_loss)
test_rmse = np.sqrt(test_loss)

print(f"\n✅ Validation RMSE: {val_rmse:.4f}")
print(f"✅ Test RMSE: {test_rmse:.4f}")

# === SAVE FINAL MODEL ===
model.save(MODEL_PATH)
print(f"\n✅ Model saved to: {MODEL_PATH}")
```

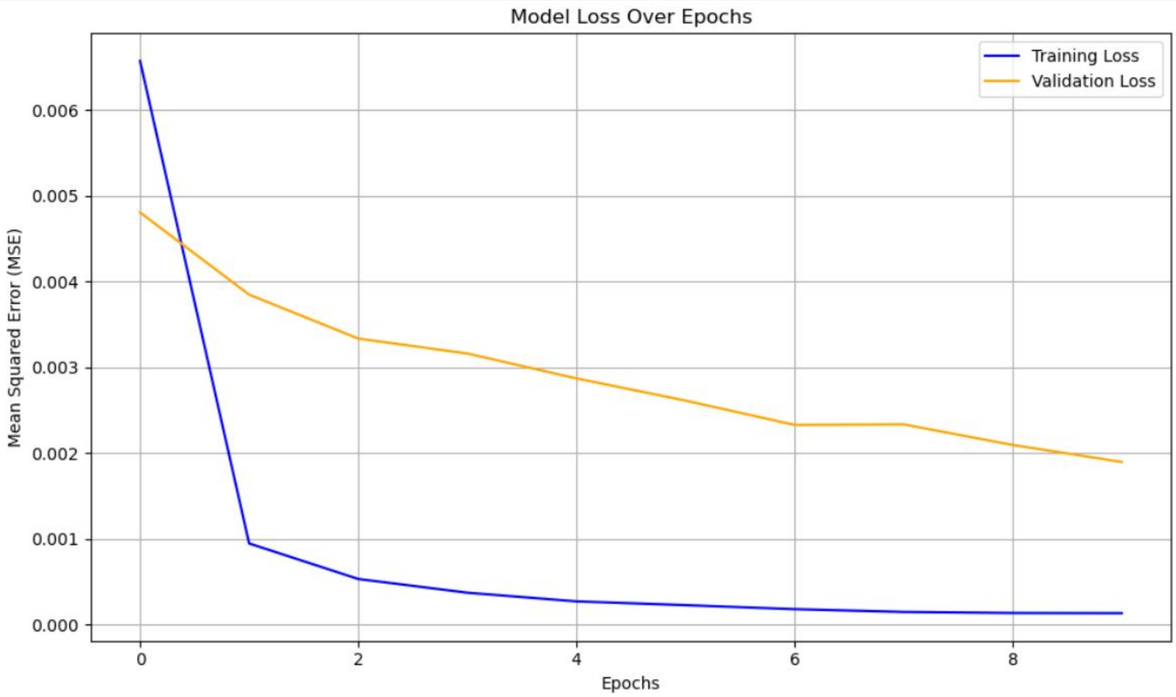
```
print(f"✅ Best checkpoint saved to: {CHECKPOINT_PATH}")



import matplotlib.pyplot as plt



# === PLOT TRAINING AND VALIDATION LOSS ===

plt.figure(figsize=(10, 6))

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

plt.plot(history.history['val_loss'], label='Validation Loss',
color='orange')

plt.title('Model Loss Over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Mean Squared Error (MSE)')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```

Results:
The LSTM model was trained on the preprocessed dataset and evaluated using RMSE. The validation and test RMSE values give insight into the model's generalization performance.

The final model training yielded the following results:

- **Validation RMSE**: 0.0435
- **Test RMSE**: 0.0350

The training and validation loss curves indicated that the model converged well and avoided overfitting due to the use of early stopping and model checkpointing.



Conclusion

This lab demonstrates the complete pipeline for time-series prediction using real-world datasets. By merging multiple data sources including energy consumption, carbon intensity, pricing, and weather data, we built a robust dataset that underwent thorough preprocessing. An LSTM model was designed and trained to predict future electricity load values. The results, evaluated using RMSE, showed that the model can learn the temporal dependencies effectively.