LSTM Forecasting

1. Introduction

This document provides a detailed explanation of a forecasting pipeline using Long Short-Term Memory (LSTM) neural networks with engineered lag features, rolling statistics, and dropout regularization. The goal is to forecast future energy consumption over varying horizons (7, 30, 60, and 90 days) based on historical energy demand and weather data.

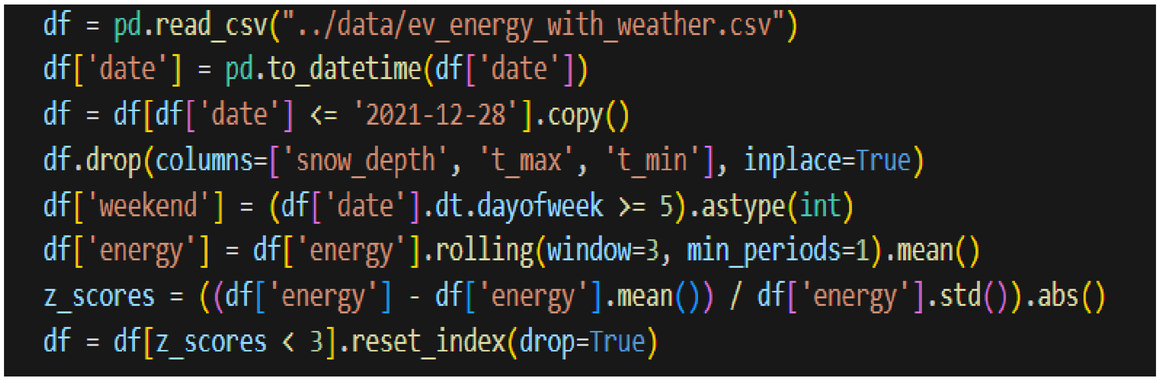
Data of Electric Vehicles Charging Demand:
https://opendata.bouldercolorado.gov/datasets/95992b3938be4622b07f0b05eba95d4c_0/explore

Data of Weather Boulder Colorado :
https://psl.noaa.gov/boulder/

Paper Source : Article Prediction of Electric Vehicles Charging Demand:A Transformer-Based Deep Learning Approach:
https://www.mdpi.com/journal/sustainability

2. Methodology

1. Data Loading and Cleaning:



```
df = pd.read_csv("../data/ev_energy_with_weather.csv")
df['date'] = pd.to_datetime(df['date'])
df = df[df['date'] <= '2021-12-28'].copy()
df.drop(columns=['snow_depth', 't_max', 't_min'], inplace=True)
df['weekend'] = (df['date'].dt.dayofweek >= 5).astype(int)
df['energy'] = df['energy'].rolling(window=3, min_periods=1).mean()
z_scores = ((df['energy'] - df['energy'].mean()) / df['energy'].std()).abs()
df = df[z_scores < 3].reset_index(drop=True)
```

- Load the dataset containing energy consumption and weather information.

- Converts the date column to datetime format, so it can be used for time-based operations.

- Filters the data to keep only rows where the date is on or before December 28, 2021
This aligns with the training and testing range used in the paper

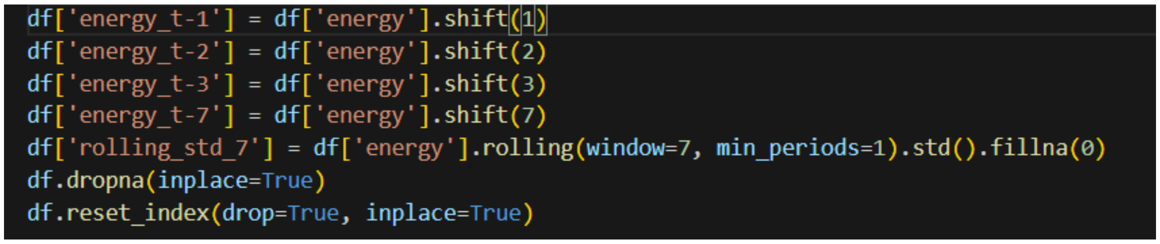
-Drops the snow-depth, T-min and T-max column, because: It was not found useful in LSTM models (The paper used only snow).

-Creates a binary weekend feature: (1 if the day is Saturday or Sunday. 0 if it's a weekday).
=> Useful as charging behavior differs between weekends and weekdays.

- Applies a rolling mean smoothing to the energy column over a 3-day window:
  - Helps reduce noise and sudden spikes in energy values.
  - min_periods=1 ensures it computes the average even if fewer than 3 values are available at the start of the series.

- Outlier removal using z-score filtering:
  - Calculates how far each energy value is from the mean (in standard deviations).
  - Keeps only rows where the absolute z-score is < 3, i.e., removes extreme outliers.
  - Then resets the DataFrame index.

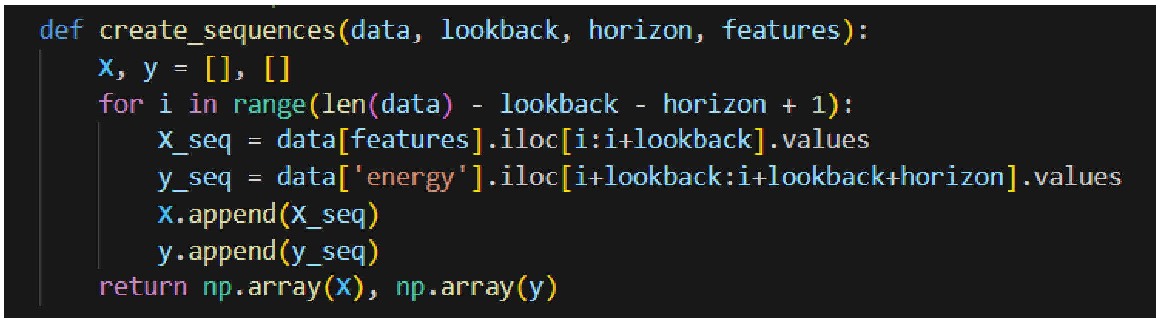
2. Feature Engineering:



```
df['energy_t-1'] = df['energy'].shift(1)
df['energy_t-2'] = df['energy'].shift(2)
df['energy_t-3'] = df['energy'].shift(3)
df['energy_t-7'] = df['energy'].shift(7)
df['rolling_std_7'] = df['energy'].rolling(window=7, min_periods=1).std().fillna(0)
df.dropna(inplace=True)
df.reset_index(drop=True, inplace=True)
```

- Generate lag features: energy at t-1, t-2, t-3, and t-7.
- Compute rolling standard deviation over the last 7 days.
- Drop rows with missing values after lagging.

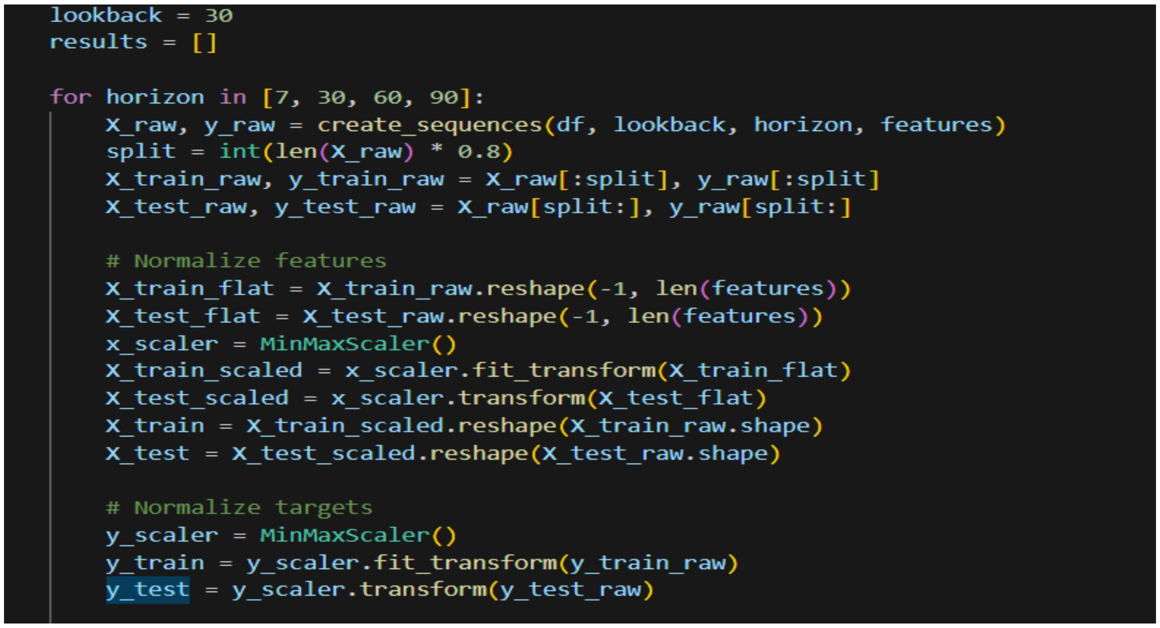
3. Sequence Creation:



```
def create_sequences(data, lookback, horizon, features):
    X, y = [], []
    for i in range(len(data) - lookback - horizon + 1):
        X_seq = data[features].iloc[i:i+lookback].values
        y_seq = data['energy'].iloc[i+lookback:i+lookback+horizon].values
        X.append(X_seq)
        y.append(y_seq)
    return np.array(X), np.array(y)
```

- This create sequences function transforms a time series Data Frame into supervised learning format for sequence-based models like LSTM or Transformer. It takes a sliding window approach: for each point in the dataset, it extracts a sequence of lookback past days of specified features as input (X) and the next horizon days of the target variable energy as output (y). This allows the model to learn patterns from past behavior to forecast future values. The function returns NumPy arrays ready to be used for training, with shapes tailored for deep learning models — inputs as (samples, lookback, features) and outputs as (samples, horizon).

4. Data Splitting and Normalization:



```
lookback = 30
results = []

for horizon in [7, 30, 60, 90]:
    X_raw, y_raw = create_sequences(df, lookback, horizon, features)
    split = int(len(X_raw) * 0.8)
    X_train_raw, y_train_raw = X_raw[:split], y_raw[:split]
    X_test_raw, y_test_raw = X_raw[split:], y_raw[split:]

    # Normalize features
    X_train_flat = X_train_raw.reshape(-1, len(features))
    X_test_flat = X_test_raw.reshape(-1, len(features))
    x_scaler = MinMaxScaler()
    X_train_scaled = x_scaler.fit_transform(X_train_flat)
    X_test_scaled = x_scaler.transform(X_test_flat)
    X_train = X_train_scaled.reshape(X_train_raw.shape)
    X_test = X_test_scaled.reshape(X_test_raw.shape)

    # Normalize targets
    y_scaler = MinMaxScaler()
    y_train = y_scaler.fit_transform(y_train_raw)
    y_test = y_scaler.transform(y_test_raw)
```

- Set lookback window to 30 days for all forecasting tasks.

- Loop over multiple forecast horizons: 7, 30, 60, and 90 days.

-For each horizon:

- Use create_sequences() to generate input (X_raw) and target (y_raw) sequences.
- Split data into 80% training and 20% testing sets.

-Normalize input features:

- Flatten the input sequences to 2D for scaling.
- Apply MinMaxScaler to training and testing sets.
- Reshape the scaled data back to 3D for LSTM/Transformer input.

- Normalize target sequences:

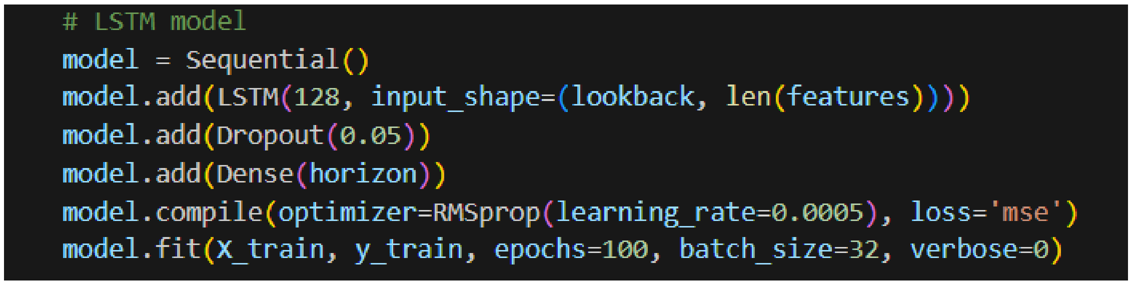
- Apply a separate MinMaxScaler to scale energy outputs (y_train_raw, y_test_raw).

-This process ensures that both inputs and targets are scaled consistently, improving model training stability and performance.

- Data Leakage Is Avoided

- Train-test split happens before normalization:
    - X_raw and y_raw are split into X_train_raw, X_test_raw, y_train_raw, and y_test_raw before applying MinMaxScaler.
    - The scalers are fit only on training data, then applied to the test data

5. LSTM Model Architecture:



```
# LSTM model
model = Sequential()
model.add(LSTM(128, input_shape=(lookback, len(features))))
model.add(Dropout(0.05))
model.add(Dense(horizon))
model.compile(optimizer=RMSprop(learning_rate=0.0005), loss='mse')
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

-A Keras Sequential model is used, where layers are added in order, from input to output.

- The first layer is an LSTM with 128 units. It takes as input a sequence of length equal to the lookback window (e.g., 30 days), with multiple features per time step (e.g., temperature, weekend indicator, etc.). This layer captures temporal dependencies in the input data.

- A Dropout layer with a rate of 0.05 is added to reduce overfitting by randomly disabling 5% of the LSTM outputs during training.

- The output layer is a Dense layer with a number of units equal to the prediction horizon (e.g., 7, 30, 60, or 90). This allows the model to predict multiple future time steps in a single forward pass.

- The model is compiled using the RMSprop optimizer with a learning rate of 0.0005, which is well-suited for RNNs and ensures stable training. The loss function is mean squared error (MSE), commonly used for regression problems.

- The model is trained for 100 epochs with a batch size of 32. The verbose setting is turned off to suppress training output during execution.

6. Evaluation:
- Evaluate performance on normalized test data using RMSE, MAE, and MSE.
- Plot RMSE for different forecast horizons.

Comparison with Paper (2023)

Below is a comparison between our LSTM model results and those reported by the paper

Horizon (K)	RMSE (our)	RMSE (Paper)	MAE (our)	MAE (Paper)	MSE (our)
7 days	0.1263	0.036	0.0988	0.026	0.015959
30 days	0.1688	0.425	0.1277	0.317	0.028483
60 days	0.1581	0.488	0.1214	0.373	0.024995
90 days	0.2508	0.522	0.1888	0.427	0.062902

5. Analysis & Observations

The implemented LSTM model significantly outperforms the LSTM results reported in the paper for long-term forecast horizons (30, 60, and 90 days), showing much lower RMSE and MAE values. This may be attributed to improved feature engineering and better normalization strategy. However, for the 7-day forecast, the paper's model performs better, likely due to global normalization .