

```
In [1]: # -----INSTALLS-----

# %pip install tensorflow
```

```
In [2]: # -----IMPORTS-----

import joblib      # to load saved dataframes, scalers and models or save new
import pandas as pd # for dataframe operations
import numpy as np  # for numpy array while creating sequences
from tensorflow.keras.models import Sequential # to create sequential model
from tensorflow.keras.layers import LSTM, Input, Dense # layer architectures in model
from tensorflow.keras.optimizers import Adam # optimizer used
```

C:\Users\rutuj\AppData\Roaming\Python\Python313\site-packages\keras\src\export\tf2onnx_lib.py:8: FutureWarning: In the future 'np.object' will be defined as the corresponding NumPy scalar.
if not hasattr(np, "object"):

```
In [3]: train_df = joblib.load('../saved_objects/train_df.joblib') # train data
temp_df = joblib.load('../saved_objects/temp_df.joblib') # dataframe to split into val and test
```

Both of the above dataset are scaled and encoded

```
In [4]: temp_df.columns
```

```
Out [4]: Index(['home_id', 'device_type', 'timestamp', 'user_present', 'status',
               'is_weekend', 'hour_of_day', 'day_of_week', 'month_of_year',
               'indoor_temp', 'outdoor_temp', 'humidity', 'light_level',
               'device_type_air_conditioner', 'device_type_fridge',
               'device_type_light', 'device_type_tv', 'device_type_washer',
               'room_bedroom', 'room_kitchen', 'room_laundry_room', 'room_living_room',
               'activity_away', 'activity_cooking', 'activity_idle',
               'activity_sleeping', 'activity_watching_tv', 'energy_lag_1H',
               'energy_lag_1D', 'energy_lag_1W', 'energy_roll_mean_1hr',
               'energy_roll_mean_12hr', 'energy_roll_mean_24hr', 'energy_kWh'],
              dtype='object')
```

home_id, device_type, timestamp are metadata used to group the data in a sequence but won't be feeded into model

Splitting temp_df in val and test set

```
In [5]: val_rec = [] # stores record belonging to validation set
test_rec = [] # stores record belonging to test set

for _, group in temp_df.groupby(['home_id', 'device_type']): # each set (train, test, val) must have order
    n = len(group)
    split = int(n / 2) # temp_df -> 0.5 test + 0.5 val OR overall_df -> 0.3 * 0.5 = 0.15

    val_rec.append(group.iloc[:split])
    test_rec.append(group.iloc[split:])

# convert the records to dataframe
val_df = pd.concat(val_rec)
test_df = pd.concat(test_rec)

print(train_df.shape)
print(val_df.shape)
print(test_df.shape)
```

```
(1202850, 34)
(257750, 34)
(257800, 34)
```

LSTM expects 3D input: (samples, timesteps, features)

```
In [6]: SEQ_LEN = 24 # sequence length = records in a sequence (24 i.e 6 hours)
target_col = 'energy_kWh'
DROP_COLS = ['home_id', 'device_type', 'timestamp']

def create_sequences(df, seq_len, target_col):
    X, y = [], []

    feature_cols = [c for c in df.columns if c not in DROP_COLS + [target_col]]

    for _, group in df.groupby(['home_id', 'device_type']):
        group = group.sort_values('timestamp')
```

```

data = group[feature_cols].values
target = group[target_col].values

for i in range(seq_len, len(group)):
    X.append(data[i-seq_len:i])
    y.append(target[i])

return np.array(X), np.array(y)

X_train, y_train = create_sequences(train_df, SEQ_LEN, 'energy_kWh')
X_val, y_val = create_sequences(val_df, SEQ_LEN, 'energy_kWh')
X_test, y_test = create_sequences(test_df, SEQ_LEN, 'energy_kWh')

print("Train Shape: ", X_train.shape, ', ', y_train.shape)
print("Val Shape: ", X_val.shape, ', ', y_val.shape)
print("Test Shape: ", X_test.shape, ', ', y_test.shape)

```

```

Train Shape: (1201650, 24, 30) , (1201650,)
Val Shape: (256550, 24, 30) , (256550,)
Test Shape: (256600, 24, 30) , (256600,)

```

Input shape: (timesteps=24, features=30)

train samples = 1201650

test samples = 256600

val samples = 256600

```
In [7]: len(X_train[0])      # 1 sequence contains 24 records
```

```
Out [7]: 24
```

```
In [8]: y_train
```

```
Out [8]: array([0., 0., 0., ..., 0., 0., 0.])
```

```
In [9]: X_train[0][:3]
# sequence content (first three):
```

```
Out [9]: array([[1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
5.00000000e+00, 1.00000000e+00, 1.12195122e-01, 4.31266846e-02,
9.08256881e-01, 8.28000000e-02, 1.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
0.00000000e+00, 1.80320000e-01, 0.00000000e+00, 0.00000000e+00,
7.78010417e-03, 7.56263021e-03],
[1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
5.00000000e+00, 1.00000000e+00, 1.02439024e-01, 1.61725067e-02,
7.69331586e-01, 6.10000000e-02, 1.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
7.78010417e-03, 7.56263021e-03],
[1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
5.00000000e+00, 1.00000000e+00, 2.04878049e-01, 1.77897574e-01,
5.41284404e-01, 9.65000000e-02, 1.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
4.01411458e-03, 7.56263021e-03]])
```

```
In [10]: Redesign = True

# Design choice:
# - Stacked LSTMs to capture short-term + longer temporal dependencies
# - First LSTM returns full sequences for deeper temporal learning
# - Second LSTM compresses sequence into a context vector
# - Dense layers map learned temporal representation to final prediction
if Redesign:
    model = Sequential([
        Input(shape = (24,30)),
        LSTM(
            units = 64,
            return_sequences = True,
            dropout = 0.2,
            recurrent_dropout = 0.1
        ),
        LSTM(
            units = 32,
            return_sequences = False,
            dropout = 0.2,

```

```

        recurrent_dropout = 0.1
    ),
    Dense(
        units = 16,
        activation = "relu"
    ),
    Dense(
        units = 1
    )
])

# Compile model
# Optimizer: Adam (adaptive learning rate, stable for noisy time series)
# Loss: MSE (penalizes larger errors more, standard for regression)
# Metric: MAE (interpretable in original target scale)

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="mse",
    metrics=["mae"]
)

model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 24, 64)	24,320
lstm_1 (LSTM)	(None, 32)	12,416
dense (Dense)	(None, 16)	528
dense_1 (Dense)	(None, 1)	17

Total params: 37,281 (145.63 KB)

Trainable params: 37,281 (145.63 KB)

Non-trainable params: 0 (0.00 B)

```

In [11]: TRAIN_MODEL = True          # flip to True only when you WANT to retrain

if TRAIN_MODEL:
    model.fit(
        X_train, y_train,
        batch_size=256,
        epochs=3,
        validation_data=(X_val, y_val),
        shuffle=False          # shuffle=False is critical for time series to preserve temporal order
    )
    model.save("../saved_objects/lstm_baseline.keras")
else:
    model = load_model("../saved_objects/lstm_baseline.keras")

```

```

Epoch 1/3
[1m4694/4694[0m [32m-----[0m[37m[0m [1m1369s[0m 287ms/step - loss: 0.0025 - mae: 0.0257 - val_loss: 0.0019 - val_mae: 0.0018
Epoch 2/3
[1m4694/4694[0m [32m-----[0m[37m[0m [1m1637s[0m 345ms/step - loss: 0.0019 - mae: 0.0200 - val_loss: 0.0018 - val_mae: 0.0018
Epoch 3/3
[1m4694/4694[0m [32m-----[0m[37m[0m [1m310s[0m 66ms/step - loss: 0.0019 - mae: 0.0193 - val_loss: 0.0018 - val_mae: 0.0018

```

The LSTM baseline converges cleanly in one epoch. Validation loss and MAE are slightly lower than training, which suggests no overfitting and good temporal generalization. This is before any hyperparameter tuning

Baseline LSTM model ready for hyperparameter tuning once reviewed