

```
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, Dropout, BatchNormalization # layer architectures
from tensorflow.keras.optimizers import Adam # optimizer used
from tensorflow.keras.models import load_model # to load saved model
from tensorflow.keras.losses import Huber
from tensorflow.keras.backend import clear_session # to clean the memory used in cross validation
from tensorflow.keras.callbacks import EarlyStopping # to stop the training early for best generalization
from tensorflow.keras.metrics import RootMeanSquaredError # to observe rsme stats of models
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score # for final evaluation
import random # for random search hypertuning
from gc import collect # to clear memory
import json # to save a model stats
```

```
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')
```

```
In [5]: temp_df = temp_df.sort_values(['device_type', 'home_id', 'timestamp'])
train_df = train_df.sort_values(['device_type', 'home_id', 'timestamp'])

train_df.head(100000)[['home_id', 'device_type']].value_counts()
```

```
Out [5]: home_id  device_type
1         air_conditioner    24057
2         air_conditioner    24057
3         air_conditioner    24057
4         air_conditioner    24057
5         air_conditioner     3772
Name: count, dtype: int64
```

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 [6]: val_rec = [] # stores record belonging to validation set
test_rec = [] # stores record belonging to test set

for _, group in temp_df.groupby(['device_type', 'home_id']): # 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 [7]: SEQ_LEN = 12      # sequence length = records in a sequence (12 * 1/4 so 3 hours)
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, dtype = np.float32), np.array(y, dtype = np.float32)

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: (1202250, 12, 30) , (1202250,)
Val Shape: (257150, 12, 30) , (257150,)
Test Shape: (257200, 12, 30) , (257200,)
```

Input shape: (timesteps=12, features=30)

train samples = 1202250

val samples = 257150

test samples = 257200

```
In [8]: len(X_train[0])      # 1 sequence contains 3 * 4 = 12 records
```

Out [8]: 12

```
In [9]: y_train
```

Out [9]: array([0., 0., 0., ..., 0., 0., 0.], dtype=float32)

```
In [10]: X_train[0][:3]
# sequence content (first three):
```

```
Out [10]: array([[1.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
5.0000000e+00, 1.0000000e+00, 1.1219512e-01, 4.3126684e-02,
9.0825689e-01, 8.2800001e-02, 1.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
0.0000000e+00, 1.8031999e-01, 0.0000000e+00, 0.0000000e+00,
7.7801039e-03, 7.5626303e-03],
[1.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
5.0000000e+00, 1.0000000e+00, 1.0243902e-01, 1.6172506e-02,
7.6933157e-01, 6.1000001e-02, 1.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
7.7801039e-03, 7.5626303e-03],
[1.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
5.0000000e+00, 1.0000000e+00, 2.0487805e-01, 1.7789757e-01,
5.4128438e-01, 9.6500002e-02, 1.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 1.0000000e+00, 0.0000000e+00,
0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
4.0141144e-03, 7.5626303e-03]], dtype=float32)
```

```
In [11]: type(X_train[0][0][0])
```

Out [11]: numpy.float32

In [12]: Redesign = True

```
def build_lstm_model():

    # Model Design Rationale:
    # - Two-layer stacked LSTM for hierarchical temporal feature learning
    # - First LSTM (64 units) captures short-term temporal patterns across the full sequence
    # - Second LSTM (32 units) compresses the sequence into a context vector
    # - Light dropout (0.1) for regularization without disrupting memory
    # - Dense projection layer adds non-linear feature interaction
    # - Softplus output enforces non-negative predictions with smooth gradients
    # - Huber loss improves robustness to spikes and outliers in the target
    # - Architecture favors generalization, stability, and causal validity

    model = Sequential([
        Input(shape = (SEQ_LEN, 30)),
        LSTM(
            units = 64,
            return_sequences = True,
            dropout = 0.1,
            recurrent_dropout = 0.0
        ),
        LSTM(
            units = 32,
            return_sequences = False,
            dropout = 0.1,
            recurrent_dropout = 0.0
        ),
        Dense(
            units = 32,
            activation = "relu"
        ),
        Dense(
            units = 1,
            activation = "softplus"
        )
    ])

    model.compile(
        optimizer=Adam(learning_rate = 0.001),
        loss=Huber(delta=0.1),
        metrics=[
            "mae",
            RootMeanSquaredError(name="rmse")
        ]
    )

    return model

if Redesign:
    baseline_lstm = build_lstm_model()

    baseline_lstm.fit(
        X_train, y_train,
        batch_size = 64,
        epochs = 3,
        validation_data = (X_val, y_val),
        shuffle = False
    )

    baseline_lstm.save("../saved_objects/baseline_lstm_model.keras")
    history_baseline = baseline_lstm.history
    print()
    print()
    print(history_baseline.history)

    with open("../saved_objects/baseline_lstm_stats.json", "w") as f:
        json.dump(history_baseline.history, f)
else:
    baseline_lstm = load_model("../saved_objects/baseline_lstm_model.keras")

    with open("../saved_objects/baseline_lstm_stats.json", "r") as f:
        stats = json.load(f)
```

```

for k, v in stats.items():
    print(f"{k}: {v[-1]}")

```

```

Epoch 1/3
[1m18786/18786[0m [32m-----[0m[37m[0m [1m320s[0m 17ms/step - loss: 7.8094e-04 - mae: 0.0190 - rmse: 0.0466 - val_loss: 0.0190
Epoch 2/3
[1m18786/18786[0m [32m-----[0m[37m[0m [1m515s[0m 27ms/step - loss: 6.8244e-04 - mae: 0.0170 - rmse: 0.0430 - val_loss: 0.0170
Epoch 3/3
[1m18786/18786[0m [32m-----[0m[37m[0m [1m452s[0m 21ms/step - loss: 6.7412e-04 - mae: 0.0167 - rmse: 0.0427 - val_loss: 0.0167

```

```

{'loss': [0.0007809381349943578, 0.0006824405863881111, 0.000674117065500468], 'mae': [0.019003186374902725, 0.016950739547610283, 0.016715487465262]}

```

The LSTM baseline converges cleanly in 3 epochs. Validation loss and MAE are slightly lower than training, which suggests no overfitting and good temporal generalization. This is before any hyperparameter tuning

```

In [13]: def temporal_cv_lstm(X, y, splits=5):

    fold_size = len(X) // (splits + 1)
    cv_metrics = []

    for i in range(splits):
        train_end = fold_size * (i + 1)
        val_end = train_end + fold_size

        X_tr, y_tr = X[:train_end], y[:train_end]
        X_val, y_val = X[train_end:val_end], y[train_end:val_end]

        clear_session()
        print()
        print(f"split {i + 1}")
        model = build_lstm_model()

        history = model.fit(
            X_tr, y_tr,
            validation_data=(X_val, y_val),
            epochs=3,
            batch_size=64,
            shuffle=False
        )

        cv_metrics.append({
            "val_rmse": history.history["val_rmse"][-1],
            "val_mae": history.history["val_mae"][-1],
            "val_loss": history.history["val_loss"][-1]
        })

    return cv_metrics

if Redesign:
    cv_res = temporal_cv_lstm(X_train, y_train)
    joblib.dump(cv_res, "../saved_objects/cv_lstm_result.joblib");
else:
    cv_res = joblib.load("../saved_objects/cv_lstm_result.joblib");

```

WARNING:tensorflow:From C:\Users\rutuj\AppData\Roaming\Python\Python313\site-packages\keras\src\backend\tensorflow_backend.py:82: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

```

split 1
Epoch 1/3
[1m3131/3131[0m [32m-----[0m[37m[0m [1m139s[0m 42ms/step - loss: 0.0012 - mae: 0.0281 - rmse: 0.0614 - val_loss: 0.0015
Epoch 2/3
[1m3131/3131[0m [32m-----[0m[37m[0m [1m75s[0m 24ms/step - loss: 8.7436e-04 - mae: 0.0215 - rmse: 0.0484 - val_loss: 0.0015
Epoch 3/3
[1m3131/3131[0m [32m-----[0m[37m[0m [1m127s[0m 38ms/step - loss: 8.3356e-04 - mae: 0.0202 - rmse: 0.0474 - val_loss: 0.0015

split 2
Epoch 1/3
[1m6262/6262[0m [32m-----[0m[37m[0m [1m181s[0m 28ms/step - loss: 9.5768e-04 - mae: 0.0230 - rmse: 0.0512 - val_loss: 0.0015
Epoch 2/3
[1m6262/6262[0m [32m-----[0m[37m[0m [1m179s[0m 29ms/step - loss: 8.1870e-04 - mae: 0.0197 - rmse: 0.0470 - val_loss: 0.0015
Epoch 3/3
[1m6262/6262[0m [32m-----[0m[37m[0m [1m168s[0m 27ms/step - loss: 7.9225e-04 - mae: 0.0190 - rmse: 0.0463 - val_loss: 0.0015

split 3
Epoch 1/3
[1m9393/9393[0m [32m-----[0m[37m[0m [1m243s[0m 25ms/step - loss: 9.4554e-04 - mae: 0.0223 - rmse: 0.0532 - val_loss: 0.0015
Epoch 2/3
[1m9393/9393[0m [32m-----[0m[37m[0m [1m230s[0m 25ms/step - loss: 7.2188e-04 - mae: 0.0179 - rmse: 0.0440 - val_loss: 0.0015
Epoch 3/3
[1m9393/9393[0m [32m-----[0m[37m[0m [1m268s[0m 25ms/step - loss: 7.0798e-04 - mae: 0.0175 - rmse: 0.0437 - val_loss: 0.0015

split 4
Epoch 1/3
[1m12524/12524[0m [32m-----[0m[37m[0m [1m309s[0m 24ms/step - loss: 8.0729e-04 - mae: 0.0199 - rmse: 0.0473 - val_loss: 0.0015
Epoch 2/3
[1m12524/12524[0m [32m-----[0m[37m[0m [1m327s[0m 24ms/step - loss: 7.0000e-04 - mae: 0.0174 - rmse: 0.0434 - val_loss: 0.0015
Epoch 3/3
[1m12524/12524[0m [32m-----[0m[37m[0m [1m298s[0m 24ms/step - loss: 6.8838e-04 - mae: 0.0171 - rmse: 0.0431 - val_loss: 0.0015

split 5

```

```
Epoch 1/3  
[1m15655/15655[0m [32m-----[0m[37m[0m [1m380s[0m 24ms/step - loss: 7.9759e-04 - mae: 0.0194 - rmse: 0.0471 - val_loss: 0.00092  
Epoch 2/3  
[1m15655/15655[0m [32m-----[0m[37m[0m [1m369s[0m 24ms/step - loss: 7.0749e-04 - mae: 0.0174 - rmse: 0.0438 - val_loss: 0.00024  
Epoch 3/3  
[1m15655/15655[0m [32m-----[0m[37m[0m [1m362s[0m 23ms/step - loss: 6.9839e-04 - mae: 0.0171 - rmse: 0.0435 - val_loss: 0.00024
```

```
In [14]: rmse = [f["val_rmse"] for f in cv_res]  
mae = [f["val_mae"] for f in cv_res]  
loss = [f["val_loss"] for f in cv_res]  
  
print("Temporal CV results:")  
print(f"RMSE → mean: {np.mean(rmse):.5f}, std: {np.std(rmse):.5f}")  
print(f"MAE → mean: {np.mean(mae):.5f}, std: {np.std(mae):.5f}")  
print(f"LOSS → mean: {np.mean(loss):.5f}, std: {np.std(loss):.5f}")  
  
print("\nRMSE:")  
for val in rmse:  
    print(val)  
  
print("\nMAE:")  
for val in mae:  
    print(val)  
  
print("\nLOSS:")  
for val in loss:  
    print(val)
```

```
Temporal CV results:  
RMSE → mean: 0.04825, std: 0.00594  
MAE → mean: 0.02304, std: 0.00565  
LOSS → mean: 0.00092, std: 0.00024
```

```
RMSE:  
0.058134786784648895  
0.04132916405797005  
0.048137035220861435  
0.05048685520887375  
0.04317043349146843
```

```
MAE:  
0.03429104760289192  
0.019628798589110374  
0.020513473078608513  
0.02101912349462509  
0.019748428836464882
```

```
LOSS:  
0.0013647950254380703  
0.0006703899707645178  
0.0008490064647048712  
0.0009566462831571698  
0.0007471515564247966
```

The model demonstrates strong temporal stability with low variance across folds, indicating reliable generalization rather than split-specific optimization.

Consistent performance across temporal folds suggests the model captures underlying patterns instead of overfitting to favorable windows.

Early Stopping for final model

```
In [15]: if Redesign:  
    early_stop = EarlyStopping(  
        monitor="val_loss",  
        patience=3,  
        restore_best_weights=True  
    )  
  
    model_ = build_lstm_model()  
  
    history = model_.fit(  
        X_train,  
        y_train,  
        validation_data=(X_val, y_val),  
        epochs=20,          # upper bound  
        batch_size=64,  
        shuffle=False,  
        callbacks=[early_stop],  
        verbose=1  
    )  
  
    model_.save("../saved_objects/FinalModel.keras")  
  
    final_metrics = {  
        "final_val_mae": history.history["val_mae"][-1],
```

```

    "final_val_rmse": history.history["val_rmse"][-1]
}

with open("../saved_objects/final_model_metrics.json", "w") as f:
    json.dump(final_metrics, f)

print()
print("Final Metrics:")
print(final_metrics)

else:
    with open("../saved_objects/final_model_metrics.json", "r") as f:
        final_metrics = json.load(f)

    print("Loaded final model metrics:")
    for k, v in final_metrics.items():
        print(f"{k}: {v}")

    model_ = load_model("../saved_objects/FinalModel.keras")

```

```

Epoch 1/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m457s[0m 24ms/step - loss: 7.5196e-04 - mae: 0.0185 - rmse: 0.0453 - val_loss: 0.0429
Epoch 2/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m496s[0m 24ms/step - loss: 6.8213e-04 - mae: 0.0170 - rmse: 0.0429 - val_loss: 0.0429
Epoch 3/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m441s[0m 23ms/step - loss: 6.7386e-04 - mae: 0.0167 - rmse: 0.0427 - val_loss: 0.0427
Epoch 4/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m437s[0m 23ms/step - loss: 6.7040e-04 - mae: 0.0166 - rmse: 0.0426 - val_loss: 0.0426
Epoch 5/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m443s[0m 24ms/step - loss: 6.6863e-04 - mae: 0.0166 - rmse: 0.0425 - val_loss: 0.0425
Epoch 6/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m687s[0m 33ms/step - loss: 6.6635e-04 - mae: 0.0165 - rmse: 0.0425 - val_loss: 0.0425
Epoch 7/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m451s[0m 24ms/step - loss: 6.6574e-04 - mae: 0.0165 - rmse: 0.0425 - val_loss: 0.0425
Epoch 8/20
[1m18786/18786[0m [32m-----[0m[37m[0m [1m441s[0m 23ms/step - loss: 6.6408e-04 - mae: 0.0164 - rmse: 0.0424 - val_loss: 0.0424

Final Metrics:
{'final_val_mae': 0.018767107278108597, 'final_val_rmse': 0.04375569522380829}

```

```

In [16]: y_pred = model_.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Test MAE : {mae:.5f}")
print(f"Test RMSE : {rmse:.5f}")
print(f"Test R² : {r2:.4f}")

```

```

[1m8038/8038[0m [32m-----[0m[37m[0m [1m67s[0m 8ms/step
Test MAE : 0.01548
Test RMSE : 0.03795
Test R² : 0.3030

```

A Test R^2 of 0.303, indicating that the model captures a meaningful portion of the variance despite the presence of sharp spikes and high noise in the target signal.

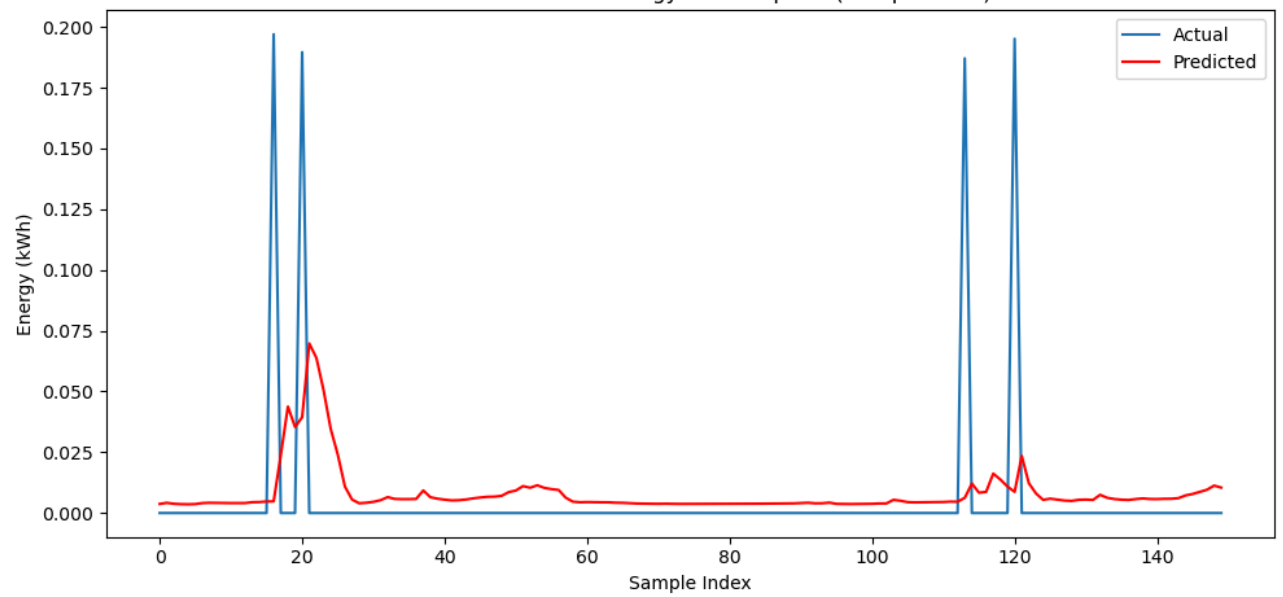
The low MAE (0.01548) reflects strong average predictive accuracy, while the higher RMSE (0.03795) highlights the impact of occasional large deviations, consistent with the known spike-heavy nature of the data.

```

In [19]: import matplotlib.pyplot as plt
sample_size = 150 # avoid clutter
plt.figure(figsize=(10, 5))
plt.plot(y_test[:sample_size], label="Actual")
plt.plot(y_pred[:sample_size], label="Predicted", color='red')
plt.xlabel("Sample Index")
plt.ylabel("Energy (kWh)")
plt.title("Actual vs Predicted Energy Consumption (Sample View)")
plt.legend()
plt.tight_layout()
plt.show()
plt.savefig('../reports/Milestone3/predictedVSactual_lstm.png')

```

Actual vs Predicted Energy Consumption (Sample View)



<Figure size 640x480 with 0 Axes>

Metric	Linear Regression	LSTM
Test MAE	0.01609	0.01548
Test RMSE	0.02586	0.03795
Test R ²	0.7084	0.3030

So the LSTM is predicting closer on average, but it struggles with larger spikes compared to Linear Regression.