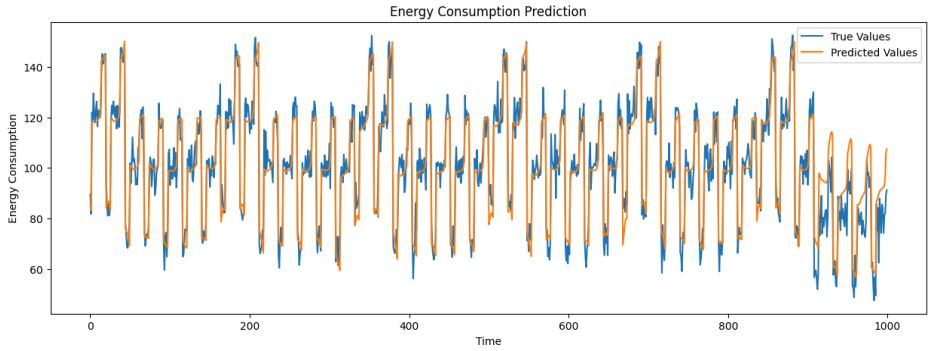
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout


df = pd.read_csv('energy_consumption_data.csv')
feature_cols = ['Day_of_Week', 'Hour_of_Day', 'Month_of_Year', 'Week_of_Month', 'Temperature',
                'Energy_Consumption_Lag1', 'Energy_Consumption_Lag24', 'Is_Holiday', 'Is_Weekend']

target_col = 'Energy_Consumption'
df = df.bfill()
df = pd.get_dummies(df, columns=['Is_Weekend'])
feature_cols = [col for col in df.columns if col in feature_cols or col.startswith('Is_Weekend')]
scaler_X = StandardScaler()
scaler_y = StandardScaler()
scaled_features = scaler_X.fit_transform(df[feature_cols])
scaled_target = scaler_y.fit_transform(df[[target_col]])

sequence_length = 24
X = []
y = []

for i in range(len(scaled_features) - sequence_length):
    X.append(scaled_features[i:i + sequence_length])
    y.append(scaled_target[i + sequence_length])

X = np.array(X)
y = np.array(y)


train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
model = Sequential([
    LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True),
    Dropout(0.2),
    LSTM(100),
    Dropout(0.2),
    Dense(50, activation='relu'),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model with adjusted epochs and batch size
history = model.fit(X_train, y_train, epochs=25, batch_size=32, validation_split=0.1, verbose=1)
```

```
Epoch 1/25
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(**kwargs)
197/197 ━━━━━━━━━━━━━━━━━━━━ 19s 66ms/step - loss: 0.5349 - val_loss: 0.1838
Epoch 2/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 11s 57ms/step - loss: 0.1547 - val_loss: 0.1233
Epoch 3/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 17s 39ms/step - loss: 0.1126 - val_loss: 0.0888
Epoch 4/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 8s 42ms/step - loss: 0.0887 - val_loss: 0.0636
Epoch 5/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - loss: 0.0893 - val_loss: 0.0518
Epoch 6/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - loss: 0.0677 - val_loss: 0.0462
Epoch 7/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 8s 43ms/step - loss: 0.0667 - val_loss: 0.0493
Epoch 8/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - loss: 0.0608 - val_loss: 0.0554
Epoch 9/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 35ms/step - loss: 0.0654 - val_loss: 0.0508
Epoch 10/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 43ms/step - loss: 0.0638 - val_loss: 0.0678
Epoch 11/25
```

```
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - loss: 0.0690 - val_loss: 0.0449
Epoch 12/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - loss: 0.0585 - val_loss: 0.0509
Epoch 13/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 35ms/step - loss: 0.0588 - val_loss: 0.0550
Epoch 14/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 11s 40ms/step - loss: 0.0569 - val_loss: 0.0527
Epoch 15/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 12s 60ms/step - loss: 0.0532 - val_loss: 0.0473
Epoch 16/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 16s 38ms/step - loss: 0.0591 - val_loss: 0.0430
Epoch 17/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 44ms/step - loss: 0.0535 - val_loss: 0.0487
Epoch 18/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - loss: 0.0556 - val_loss: 0.0467
Epoch 19/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 35ms/step - loss: 0.0549 - val_loss: 0.0651
Epoch 20/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 11s 39ms/step - loss: 0.0532 - val_loss: 0.0442
Epoch 21/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 11s 43ms/step - loss: 0.0522 - val_loss: 0.0557
Epoch 22/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 10s 44ms/step - loss: 0.0517 - val_loss: 0.0426
Epoch 23/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - loss: 0.0471 - val_loss: 0.0448
Epoch 24/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 11s 38ms/step - loss: 0.0489 - val_loss: 0.0449
Epoch 25/25
197/197 ━━━━━━━━━━━━━━━━━━━━ 11s 44ms/step - loss: 0.0494 - val_loss: 0.0438
```

```
# Predict on test set
y_pred = model.predict(X_test)

# Inverse transform to get actual values
y_test_inv = scaler_y.inverse_transform(y_test)
y_pred_inv = scaler_y.inverse_transform(y_pred)

# Calculate performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')

# Plot results
plot_range = 1000
plt.figure(figsize=(15, 5))
plt.plot(y_test_inv[:plot_range], label='True Values')
plt.plot(y_pred_inv[:plot_range], label='Predicted Values')
plt.xlabel('Time')
plt.ylabel('Energy Consumption')
plt.title('Energy Consumption Prediction')
plt.legend()
plt.show()
```

```
55/55 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step
Mean Squared Error: 0.07999794716260769
Mean Absolute Error: 0.21821460482890215
R-squared: 0.8904332824706678
```



Double-click (or enter) to edit