In [14]:
```
import numpy as np
import pandas as pd
import glob
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
```

In [15]:
```
# Prepare the data

train_files = glob.glob("../dataset/train/*.csv")
data = pd.concat([pd.read_csv(f) for f in train_files], ignore_index=True)

#sort by time
data = data.sort_values(by="time")
print(data.head())

# For Tablue
data.to_csv("../dataset/combined_turbine_data.csv", index=False)

# Use only el_power for forecasting\
series = data[["el_power"]].values

print(series[:2])

# scale between 0 & 1
scaler = MinMaxScaler()
series_scaled = scaler.fit_transform(series)

print("series_scaled:" ,series_scaled[:2])
```

```
           time  input_voltage     el_power
22910  810.00000            3.0  1102.949693
0      810.07028           10.0  1228.791720
32098  811.04000            3.0  1217.413110
22911  811.04000            3.0  1199.403786
1      811.06938           10.0  1223.041745
[[1102.94969345]
 [1228.79172023]]
series_scaled: [[0.07341755]
 [0.12772877]]
```

In [16]:
```
# Create sequences for LSTM

def create_Seq(data, window_size):
    X = []
    y = []
```

```
    print("original data shape:", data.shape)
    #print(data[:5], len(data))

    for i in range(window_size, len(data)):

        X.append(data[i-window_size:i])
        y.append(data[i])

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

WINDOW_SIZE = 20

X, y = create_Seq(series_scaled, WINDOW_SIZE)
print("X shape:", X.shape)
print("y shape:", y.shape)

# create time sequence aligned with y
times = data['time'].values
times_seq = times[WINDOW_SIZE:]
print("times_seq shape:", times_seq.shape)
```

```
original data shape: (52940, 1)
X shape: (52920, 20, 1)
y shape: (52920, 1)
times_seq shape: (52920,)
```

In [17]:
```
# Train/Test Split

split = int(0.8 * len(X))

X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
# split times sequence to align with y_test
times_seq_train, times_seq_test = times_seq[:split], times_seq[split:]
```

In [18]:
```
# Build LSTM Model\

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

model_lstm = Sequential()

model_lstm.add(LSTM(64, return_sequences=True, input_shape=(WINDOW_SIZE,1)))
model_lstm.add(Dropout(0.2))

model_lstm.add(LSTM(32))
model_lstm.add(Dropout(0.2))
```

```
model_lstm.add(Dense(1))

model_lstm.compile(optimizer="adam", loss="mse", metrics=["mae"])

model_lstm.summary()
```

```
c:\Work\MS-AI\course-7\Project\IoT_Gas_Turbine\course7_Project_venv\lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 20, 64)	16,896
dropout_2 (Dropout)	(None, 20, 64)	0
lstm_3 (LSTM)	(None, 32)	12,416
dropout_3 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

Total params: 29,345 (114.63 KB)

Trainable params: 29,345 (114.63 KB)

Non-trainable params: 0 (0.00 B)

In [19]:
```
# Train the model

history_lstm = model_lstm.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2, verbose=1)
```

```
Epoch 1/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 16s 13ms/step - loss: 0.0584 - mae: 0.1842 - val_loss: 0.0789 - val_mae: 0.2284
Epoch 2/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - loss: 0.0431 - mae: 0.1521 - val_loss: 0.0556 - val_mae: 0.1961
Epoch 3/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.0401 - mae: 0.1426 - val_loss: 0.0554 - val_mae: 0.1933
Epoch 4/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - loss: 0.0376 - mae: 0.1353 - val_loss: 0.0645 - val_mae: 0.2033
Epoch 5/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - loss: 0.0349 - mae: 0.1286 - val_loss: 0.0660 - val_mae: 0.1978
Epoch 6/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - loss: 0.0333 - mae: 0.1242 - val_loss: 0.0944 - val_mae: 0.2309
Epoch 7/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.0324 - mae: 0.1214 - val_loss: 0.1420 - val_mae: 0.2847
Epoch 8/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.0320 - mae: 0.1199 - val_loss: 0.1119 - val_mae: 0.2524
Epoch 9/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.0314 - mae: 0.1182 - val_loss: 0.1115 - val_mae: 0.2571
Epoch 10/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - loss: 0.0310 - mae: 0.1171 - val_loss: 0.1075 - val_mae: 0.2545
Epoch 11/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - loss: 0.0305 - mae: 0.1156 - val_loss: 0.1568 - val_mae: 0.2974
Epoch 12/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 12s 12ms/step - loss: 0.0303 - mae: 0.1144 - val_loss: 0.1289 - val_mae: 0.2654
Epoch 13/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 22s 13ms/step - loss: 0.0301 - mae: 0.1140 - val_loss: 0.1426 - val_mae: 0.2868
Epoch 14/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.0297 - mae: 0.1127 - val_loss: 0.2181 - val_mae: 0.3631
Epoch 15/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.0295 - mae: 0.1118 - val_loss: 0.1120 - val_mae: 0.2492
Epoch 16/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.0293 - mae: 0.1113 - val_loss: 0.1219 - val_mae: 0.2661
Epoch 17/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.0292 - mae: 0.1106 - val_loss: 0.1286 - val_mae: 0.2715
Epoch 18/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 0.0291 - mae: 0.1106 - val_loss: 0.1332 - val_mae: 0.2821
Epoch 19/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.0287 - mae: 0.1093 - val_loss: 0.2004 - val_mae: 0.3392
Epoch 20/20
1059/1059 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - loss: 0.0287 - mae: 0.1092 - val_loss: 0.1589 - val_mae: 0.3011
```

In [20]:

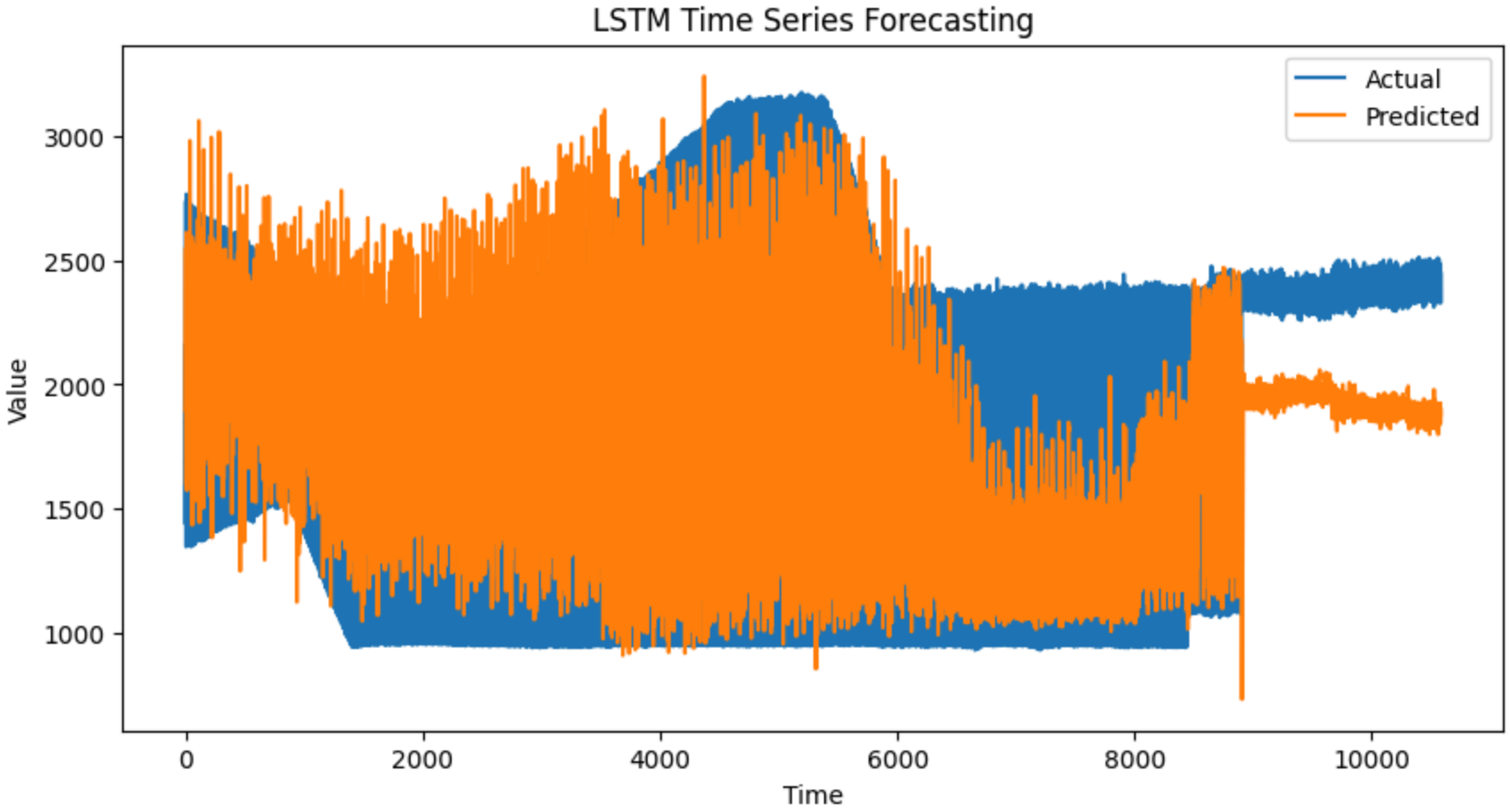
```python
# Evaluate the model

loss, mae = model_lstm.evaluate(X_test, y_test)
print("Test MSE Loss:", loss)
print("Test MAE:", mae)
```

```
331/331 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.0608 - mae: 0.2022
Test MSE Loss: 0.06082330271601677
Test MAE: 0.2021859735250473
```

In [21]:
```python
# Convert predictions back to original scale

predictions = model_lstm.predict(X_test)

# Inverse scale
predictions_inv = scaler.inverse_transform(predictions)
y_test_inv = scaler.inverse_transform(y_test)
```

```
331/331 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```

In [22]:
```python
# Visualization

plt.figure(figsize=(10,5))
plt.plot(y_test_inv, label='Actual')
plt.plot(predictions_inv, label='Predicted')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('LSTM Time Series Forecasting')
plt.legend()
plt.show()
```



In [23]:
```python
# Save predictions to CSV for Tableau visualization

lstm_results = pd.DataFrame({
    "time": times_seq_test,
    "Actual_el_power": y_test_inv.flatten(),
    "Predicted_el_power": predictions_inv.flatten()
})

lstm_results.to_csv("lstm_test_predictions.csv", index=False)

print("Saved LSTM predictions.")
```

```
Saved LSTM predictions.
```