```
import kagglehub


# Download latest version
path = kagglehub.dataset_download("dinithpriyankara/research-dataset")

print("Path to dataset files:", path)
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/dinithpriyankara/research-dataset/versions/1
```

```
import shutil
import os

destination_path = "/content"
if os.path.exists(path):
    try:
        shutil.copytree(path, destination_path + "/research-dataset", dirs_exist_ok = True)
        print("Files copied successfully to:", destination_path + "/research-dataset")
    except Exception as e:
        print(f"Error copying files: {e}")
else:
    print(f"Source directory not found: {source_path}")
```

```
Files copied successfully to: /content/research-dataset
```

Data Preparation

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, OrdinalEncoder
from sklearn.model_selection import train_test_split


url = "research-dataset/Steel_industry_data.csv"
df = pd.read_csv(url)


df.head()
```

	date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	N
0	01/01/2018 00:15	3.17	2.95	0.0	0.0	73.21	100.0	9
1	01/01/2018 00:30	4.00	4.46	0.0	0.0	66.77	100.0	18
2	01/01/2018 00:45	3.24	3.28	0.0	0.0	70.28	100.0	27
3	01/01/2018 01:00	3.31	3.56	0.0	0.0	68.09	100.0	36
4	01/01/2018 01:15	3.82	4.50	0.0	0.0	64.72	100.0	45

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Convert categorical variables
encoder = OrdinalEncoder()
df[['WeekStatus', 'Load_Type']] = encoder.fit_transform(df[['WeekStatus', 'Load_Type']])
```

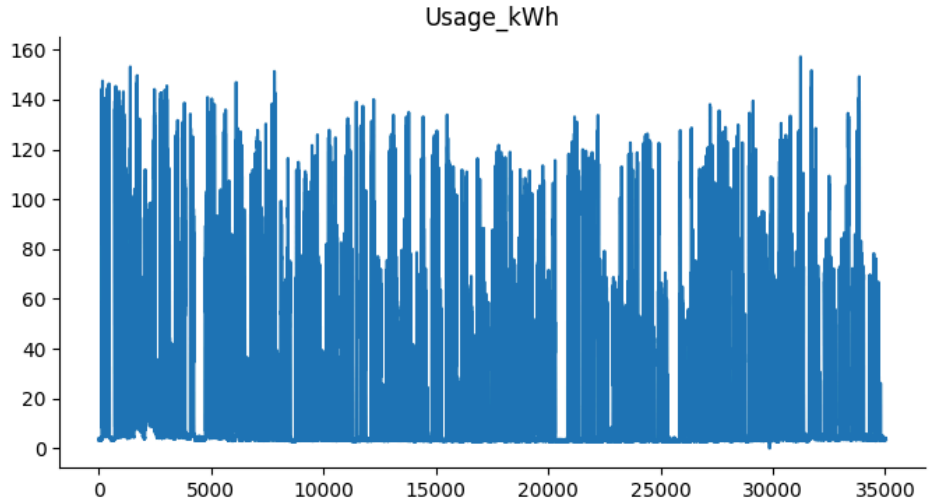
```
df.head()
```

	date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	N
0	01/01/2018 00:15	3.17	2.95	0.0	0.0	73.21	100.0	9
1	01/01/2018 00:30	4.00	4.46	0.0	0.0	66.77	100.0	18
2	01/01/2018 00:45	3.24	3.28	0.0	0.0	70.28	100.0	27
3	01/01/2018 01:00	3.31	3.56	0.0	0.0	68.09	100.0	36
4	01/01/2018 01:15	3.82	4.50	0.0	0.0	64.72	100.0	45

Next steps: Generate code with df | View recommended plots | New interactive sheet

Start coding or generate with AI.

```
from matplotlib import pyplot as plt
df['Usage_kWh'].plot(kind='line', figsize=(8, 4), title='Usage_kWh')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
df['date'] = pd.to_datetime(df['date'], dayfirst=True)
```

```
df.head()
```

	date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM
0	2018-01-01 00:15:00	3.17	2.95	0.0	0.0	73.21	100.0	900
1	2018-01-01 00:30:00	4.00	4.46	0.0	0.0	66.77	100.0	1800
2	2018-01-01 00:45:00	3.24	3.28	0.0	0.0	70.28	100.0	2700
3	2018-01-01 01:00:00	3.31	3.56	0.0	0.0	68.09	100.0	3600
4	2018-01-01 01:15:00	3.82	4.50	0.0	0.0	64.72	100.0	4500

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df = df.resample('H', on='date').agg({
    'Usage_kWh': 'mean',
    'Lagging_Current_Reactive.Power_kVarh': 'mean',
    'Leading_Current_Reactive_Power_kVarh': 'mean',
    'CO2(tCO2)': 'mean',
```

```
    'Lagging_Current_Power_Factor': 'mean',
    'Leading_Current_Power_Factor': 'mean',
    'NSM': 'first',
    'WeekStatus': 'first',
    'Load_Type': 'first'
})
```

```
<ipython-input-445-d222863be33a>:1: FutureWarning: 'H' is deprecated and will be removed in a future version, please use 'h' instead.
  df = df.resample('H', on='date').agg({
```

```
df.head()
```

date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM	I
2018-01-01 00:00:00	3.4575	3.5375	0.0	0.0	70.1400	100.0	0	
2018-01-01 01:00:00	3.5025	3.9400	0.0	0.0	66.5475	100.0	3600	
2018-01-01 02:00:00	3.5300	4.1675	0.0	0.0	64.7400	100.0	7200	
2018-01-01 03:00:00	3.4550	4.0500	0.0	0.0	65.0675	100.0	10800	
2018-01-01 04:00:00	3.6175	4.4100	0.0	0.0	63.5175	100.0	14400	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
fet=df.columns
```

```
# df=df.drop(columns=['date','Day_of_week'])
```

```
df.head(5)
```

date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM
2018-01-01 00:00:00	3.4575	3.5375	0.0	0.0	70.1400	100.0	0
2018-01-01 01:00:00	3.5025	3.9400	0.0	0.0	66.5475	100.0	3600
2018-01-01 02:00:00	3.5300	4.1675	0.0	0.0	64.7400	100.0	7200
2018-01-01 03:00:00	3.4550	4.0500	0.0	0.0	65.0675	100.0	10800
2018-01-01 04:00:00	3.6175	4.4100	0.0	0.0	63.5175	100.0	14400

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Normalization
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
```

```
scaled_data.shape
```

```
(8760, 9)
```

```
# Sliding window transformation
def create_sliding_windows(data, window_size=1):
    X, y = [], []
    for i in range(len(data)-window_size):
        X.append(data[i:i+window_size, 1:])
        y.append(data[i+window_size, 0])
    return np.array(X), np.array(y)
```

```
data={}
```

```
window_sizes=[1,4,8,12,16]
```

```
for window in window_sizes:
  X, y = create_sliding_windows(scaled_data, window)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.18, shuffle=False)
```

```
  data[f"win{window}"] = {
      "X_train": X_train,
      "X_test": X_test,
      "y_train": y_train,
      "y_test": y_test
  }
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
# pip install tensorflow

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

Start coding or generate with AI.

```
def build_lstm_model(model_type, input_shape):
    model = Sequential()

    if model_type == 'single':
        model.add(LSTM(64, input_shape=input_shape))
    elif model_type == 'double':
        model.add(LSTM(64, return_sequences=True, input_shape=input_shape))
        model.add(LSTM(64))
    elif model_type == 'bidirectional':
        model.add(Bidirectional(LSTM(64), input_shape=input_shape))

    model.add(Dropout(0.1))
    model.add(Dense(1))
    return model
```

Start coding or generate with AI.

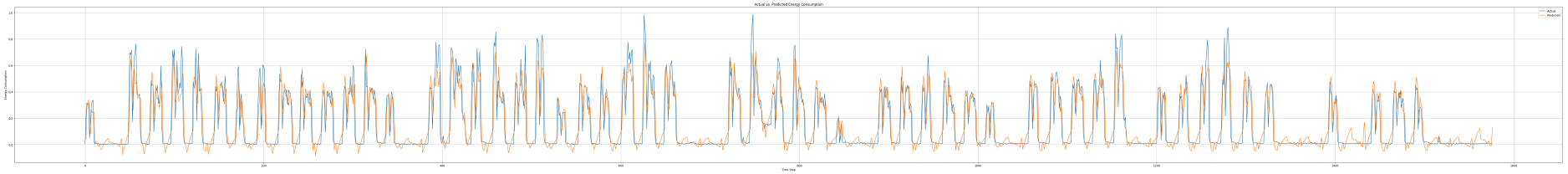
Single layer

```
import tensorflow.keras.backend as K
def rmse(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_true - y_pred)))



single_models={}


for window_size in window_sizes:
  X_train=data[f"win{window_size}"]["X_train"]
  X_test=data[f"win{window_size}"]["X_test"]
  y_train=data[f"win{window_size}"]["y_train"]
  y_test=data[f"win{window_size}"]["y_test"]

  print("==> Windows ",window_size)
  print("-->",X_train.shape,y_train.shape,X_test.shape)

  model = build_lstm_model('single', (window_size, X_train.shape[2]))
  model.compile(optimizer='adam', loss=rmse, metrics=['mae'])
  history = model.fit(
      X_train, y_train,
      epochs=50,
      batch_size=32,
      validation_split=0.2,
      verbose=1
  )
  single_models[f"win{window_size}"]={
      "model":model,
      "history":history
  }
```

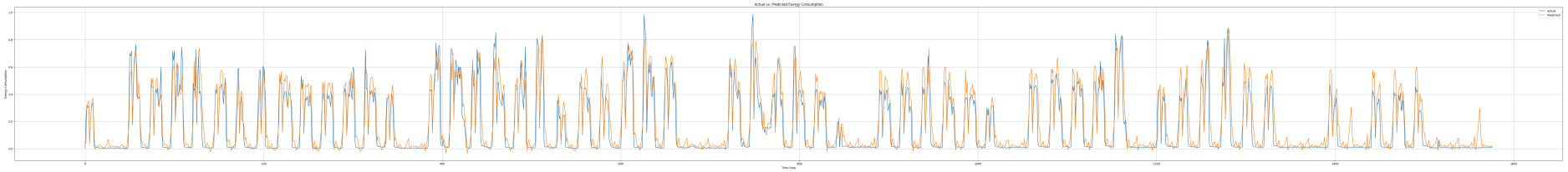
```
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0746 - mae: 0.0471 - val_loss: 0.0651 - val_mae: 0.0399
Epoch 35/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - loss: 0.0765 - mae: 0.0472 - val_loss: 0.0646 - val_mae: 0.0399
Epoch 36/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - loss: 0.0728 - mae: 0.0453 - val_loss: 0.0738 - val_mae: 0.0482
Epoch 37/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0752 - mae: 0.0474 - val_loss: 0.0643 - val_mae: 0.0400
Epoch 38/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0751 - mae: 0.0468 - val_loss: 0.0653 - val_mae: 0.0406
Epoch 39/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0719 - mae: 0.0449 - val_loss: 0.0640 - val_mae: 0.0382
Epoch 40/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 0.0740 - mae: 0.0454 - val_loss: 0.0637 - val_mae: 0.0385
Epoch 41/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0730 - mae: 0.0452 - val_loss: 0.0635 - val_mae: 0.0391
Epoch 42/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0744 - mae: 0.0462 - val_loss: 0.0671 - val_mae: 0.0423
Epoch 43/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0740 - mae: 0.0463 - val_loss: 0.0665 - val_mae: 0.0400
Epoch 44/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0743 - mae: 0.0460 - val_loss: 0.0703 - val_mae: 0.0429
Epoch 45/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - loss: 0.0709 - mae: 0.0435 - val_loss: 0.0651 - val_mae: 0.0405
Epoch 46/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0718 - mae: 0.0440 - val_loss: 0.0631 - val_mae: 0.0378
Epoch 47/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 0.0740 - mae: 0.0450 - val_loss: 0.0642 - val_mae: 0.0375
Epoch 48/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0725 - mae: 0.0452 - val_loss: 0.0632 - val_mae: 0.0377
Epoch 49/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - loss: 0.0690 - mae: 0.0428 - val_loss: 0.0640 - val_mae: 0.0402
Epoch 50/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 0.0712 - mae: 0.0440 - val_loss: 0.0655 - val_mae: 0.0396
```

```
for window_size in window_sizes:
  X_test=data[f"win{window_size}"]["X_test"]
  y_test=data[f"win{window_size}"]["y_test"]
  predict=single_models[f"win{window_size}"]["model"].predict(X_test)

  r = range(0, len(y_test))
  plt.figure(figsize=(100, 10))
  plt.plot(r, y_test, label='Actual' )
  plt.plot(r, predict, label='Predicted')

  plt.legend()
  plt.title('Actual vs. Predicted Energy Consumption')
  plt.xlabel('Time Step')
  plt.ylabel('Energy Consumption')
  plt.grid(True)
  plt.show()
```

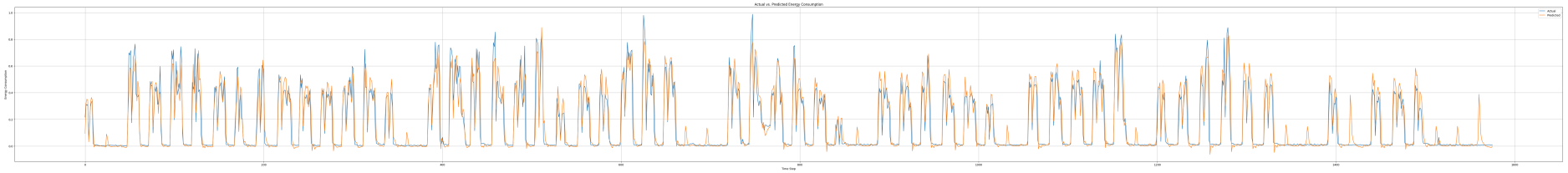
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```



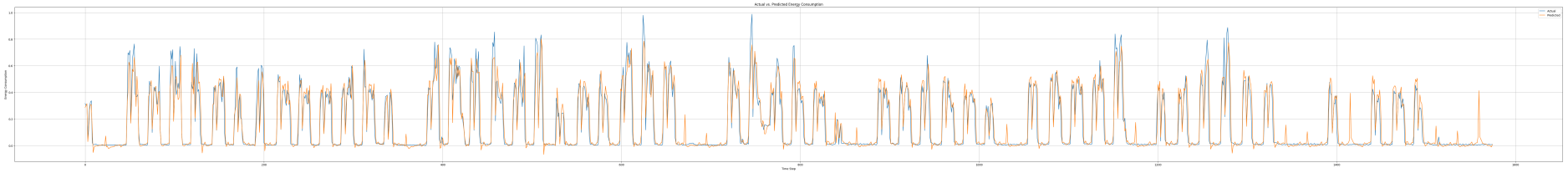
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
```



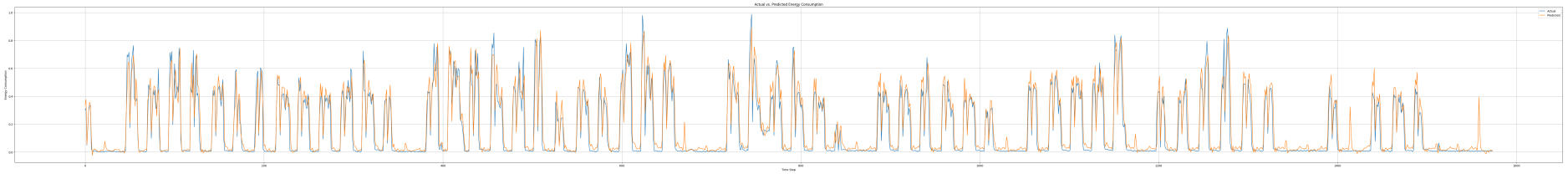
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
```



```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
```



```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
```



Double

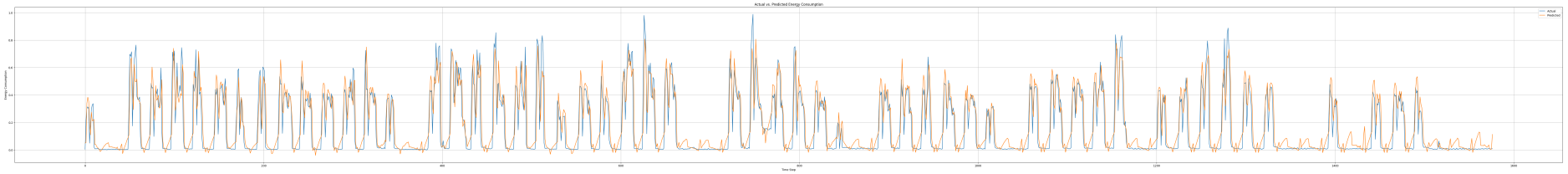
```
dou_models={}


for window_size in window_sizes:
  X_train=data[f"win{window_size}"]["X_train"]
  X_test=data[f"win{window_size}"]["X_test"]
```

```
y_train=data[f"win{window_size}"]["y_train"]
y_test=data[f"win{window_size}"]["y_test"]

print("==> Windows ",window_size)
print("-->",X_train.shape,y_train.shape,X_test.shape)
model = build_lstm_model('double', (window_size, X_train.shape[2]))
model.compile(optimizer='adam', loss=rmse , metrics=['mae'])
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)
dou_models[f"win{window_size}"]={
    "model":model,
    "history":history
}
```

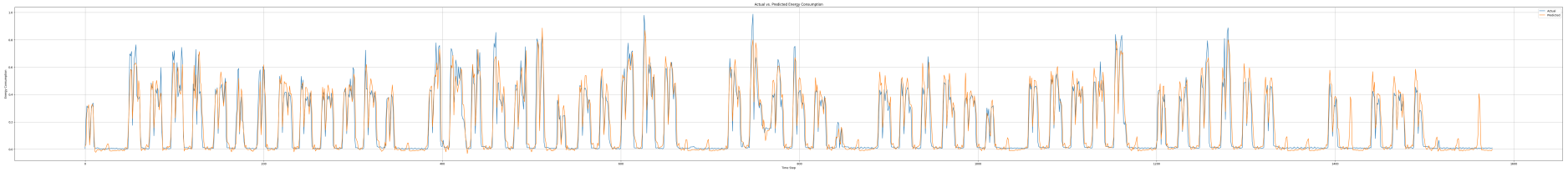
```
Epoch 42/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - loss: 0.0727 - mae: 0.0446 - val_loss: 0.0644 - val_mae: 0.0391
Epoch 43/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.0725 - mae: 0.0441 - val_loss: 0.0636 - val_mae: 0.0379
Epoch 44/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.0722 - mae: 0.0434 - val_loss: 0.0639 - val_mae: 0.0377
Epoch 45/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 5s 30ms/step - loss: 0.0699 - mae: 0.0430 - val_loss: 0.0657 - val_mae: 0.0406
Epoch 46/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - loss: 0.0725 - mae: 0.0437 - val_loss: 0.0639 - val_mae: 0.0386
Epoch 47/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.0735 - mae: 0.0443 - val_loss: 0.0667 - val_mae: 0.0407
Epoch 48/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - loss: 0.0718 - mae: 0.0437 - val_loss: 0.0641 - val_mae: 0.0388
Epoch 49/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0711 - mae: 0.0427 - val_loss: 0.0650 - val_mae: 0.0394
Epoch 50/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.0713 - mae: 0.0431 - val_loss: 0.0640 - val_mae: 0.0399
```

```
for window_size in window_sizes:
  X_test=data[f"win{window_size}"]["X_test"]
  y_test=data[f"win{window_size}"]["y_test"]
  predict=dou_models[f"win{window_size}"]["model"].predict(X_test)

  r = range(0, len(y_test))
  plt.figure(figsize=(100, 10))
  plt.plot(r, y_test, label='Actual' )
  plt.plot(r, predict, label='Predicted')

  plt.legend()
  plt.title('Actual vs. Predicted Energy Consumption')
  plt.xlabel('Time Step')
  plt.ylabel('Energy Consumption')
  plt.grid(True)
  plt.show()
```

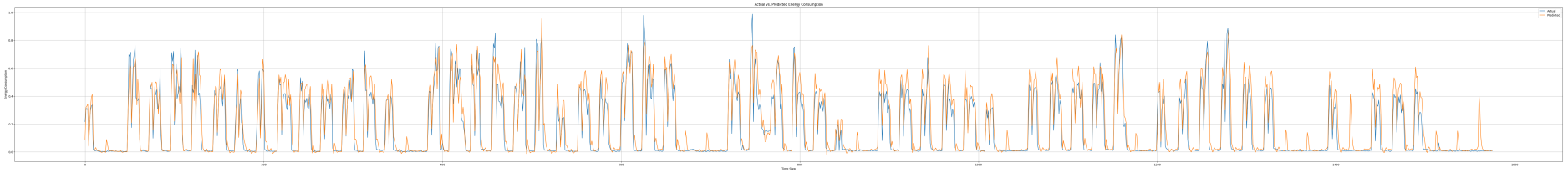
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
```



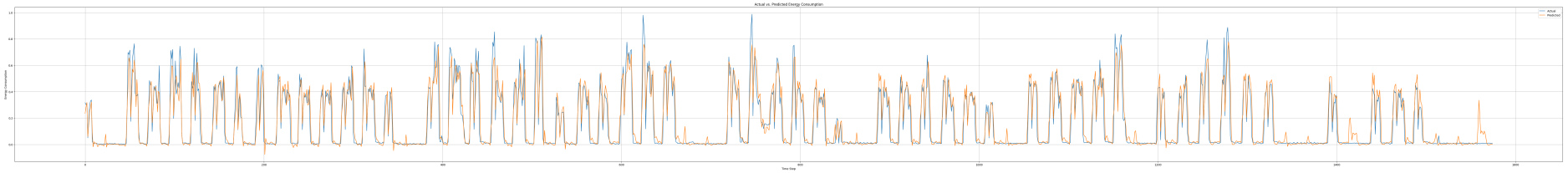
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step
```



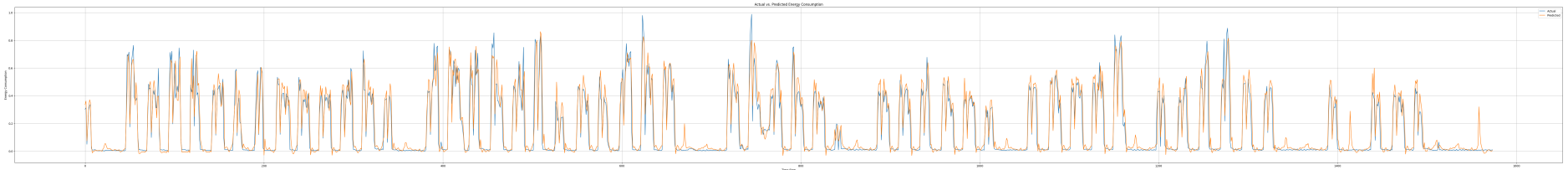
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
```



```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step
```



```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step
```



Bidirectional

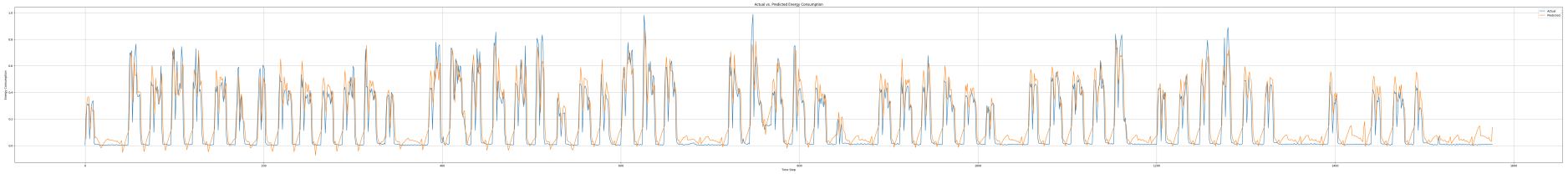
```
bidirec_models={}


for window_size in window_sizes:
  X_train=data[f"win{window_size}"]["X_train"]
  X_test=data[f"win{window_size}"]["X_test"]
```

```
y_train=data[f"win{window_size}"]["y_train"]
y_test=data[f"win{window_size}"]["y_test"]

print("==> Windows ",window_size)
print("-->",X_train.shape,y_train.shape,X_test.shape)

model = build_lstm_model('bidirectional', (window_size, X_train.shape[2]))
model.compile(optimizer='adam', loss=rmse, metrics=['mae'])
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)
bidirec_models[f"win{window_size}"]={
    "model":model,
    "history":history
}
```

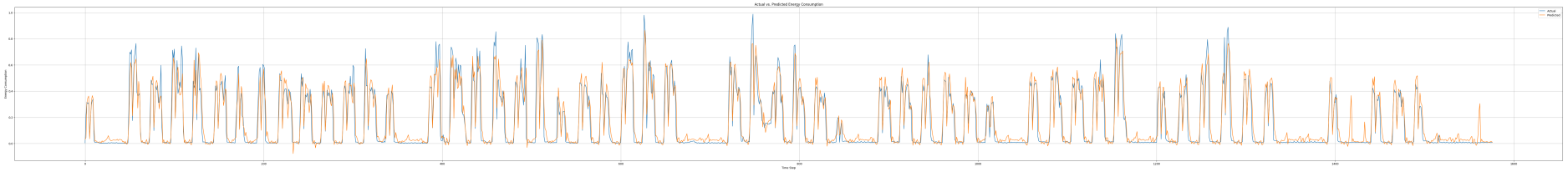
```
Epoch 42/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 6s 21ms/step - loss: 0.0710 - mae: 0.0443 - val_loss: 0.0658 - val_mae: 0.0401
Epoch 43/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0753 - mae: 0.0471 - val_loss: 0.0649 - val_mae: 0.0390
Epoch 44/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0729 - mae: 0.0453 - val_loss: 0.0655 - val_mae: 0.0410
Epoch 45/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 0.0739 - mae: 0.0465 - val_loss: 0.0653 - val_mae: 0.0399
Epoch 46/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - loss: 0.0754 - mae: 0.0470 - val_loss: 0.0647 - val_mae: 0.0395
Epoch 47/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - loss: 0.0727 - mae: 0.0459 - val_loss: 0.0648 - val_mae: 0.0416
Epoch 48/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 0.0725 - mae: 0.0449 - val_loss: 0.0643 - val_mae: 0.0391
Epoch 49/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - loss: 0.0699 - mae: 0.0439 - val_loss: 0.0653 - val_mae: 0.0398
Epoch 50/50
180/180 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - loss: 0.0728 - mae: 0.0452 - val_loss: 0.0643 - val_mae: 0.0392
```

```
for window_size in window_sizes:
  X_test=data[f"win{window_size}"]["X_test"]
  y_test=data[f"win{window_size}"]["y_test"]
  predict=bidirec_models[f"win{window_size}"]["model"].predict(X_test)

  r = range(0, len(y_test))
  plt.figure(figsize=(100, 10))
  plt.plot(r, y_test, label='Actual' )  # Plot actual values
  plt.plot(r, predict, label='Predicted')  # Plot predicted values

  plt.legend()
  plt.title('Actual vs. Predicted Energy Consumption')
  plt.xlabel('Time Step')
  plt.ylabel('Energy Consumption')
  plt.grid(True)  # Optional: Add grid lines
  plt.show()
```

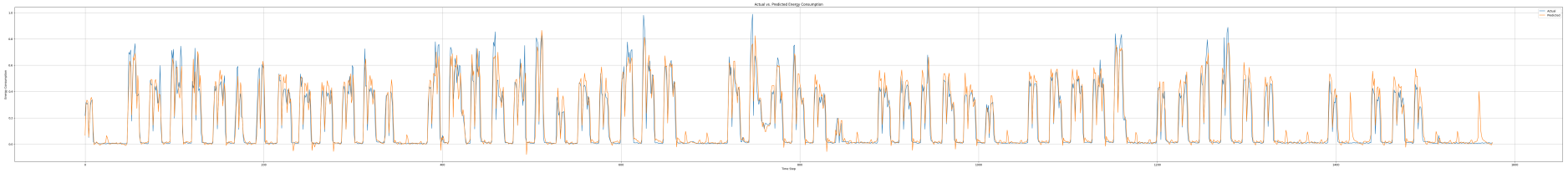
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
```



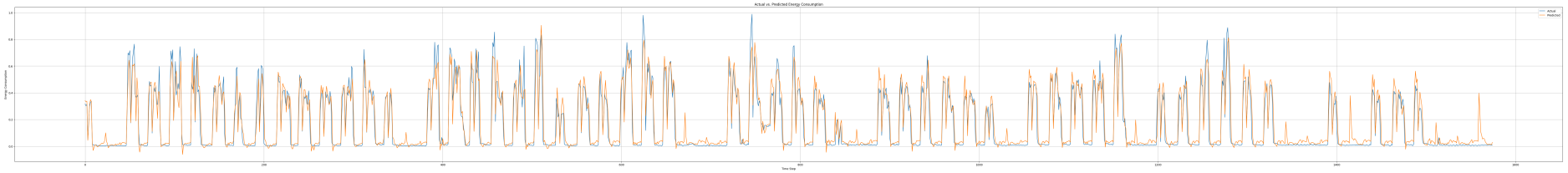
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
```



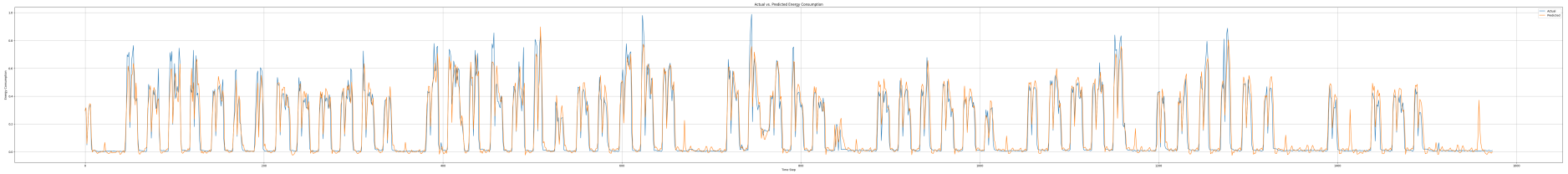
```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
```



```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step
```



```
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step
```



+ Code + Text

Start coding or generate with AI.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, root_mean_squared_error


# def willmott_index(y_true, y_pred):
#     numerator = np.sum((y_pred - y_true)**2)
#     denominator = np.sum((np.abs(y_pred - np.mean(y_true)) +
```

```
#                         np.abs(y_true - np.mean(y_true)))**2)
#     return 1 - (numerator / denominator)


def evaluate_model(model, X_test, y_test):

    y_pred = model.predict(X_test)

    rmse = root_mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return {"RMSE": rmse, "MAE": mae, "R2": r2}
metrics = evaluate_model(model, X_test, y_test)
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

Start coding or generate with AI.

Testing

Single layer eve

```
for window_size in window_sizes:
  model=single_models[f"win{window_size}"]["model"]
  X_test=data[f"win{window_size}"]["X_test"]
  y_test=data[f"win{window_size}"]["y_test"]
  metrics = evaluate_model(model, X_test, y_test)
  print(f"Metrics for window size {window_size}: {metrics}")
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Metrics for window size 1: {'RMSE': 0.10419658349150898, 'MAE': 0.06767097038529116, 'R2': 0.7723281909980052}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Metrics for window size 4: {'RMSE': 0.09454865158594396, 'MAE': 0.06058591145385965, 'R2': 0.8125381152485122}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Metrics for window size 8: {'RMSE': 0.07817660733939687, 'MAE': 0.045286841660131576, 'R2': 0.8718757981243315}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Metrics for window size 12: {'RMSE': 0.07963392666660324, 'MAE': 0.044600750038582666, 'R2': 0.8671343703260552}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Metrics for window size 16: {'RMSE': 0.07903504524971819, 'MAE': 0.04712928178685975, 'R2': 0.869169301319893}
```

```
for window_size in window_sizes:
  model=dou_models[f"win{window_size}"]["model"]
  X_test=data[f"win{window_size}"]["X_test"]
  y_test=data[f"win{window_size}"]["y_test"]
  metrics = evaluate_model(model, X_test, y_test)
  print(f"Metrics for window size {window_size}: {metrics}")
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Metrics for window size 1: {'RMSE': 0.1036623401214516, 'MAE': 0.06773681203724248, 'R2': 0.774656872618303}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

```
Metrics for window size 4: {'RMSE': 0.08623863722660322, 'MAE': 0.05098465898672615, 'R2': 0.8440425698974137}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Metrics for window size 8: {'RMSE': 0.086327525416349, 'MAE': 0.05165949782499582, 'R2': 0.8437657961771989}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Metrics for window size 12: {'RMSE': 0.08120903774254105, 'MAE': 0.04674195975508196, 'R2': 0.8618263858689397}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Metrics for window size 16: {'RMSE': 0.07651779476485285, 'MAE': 0.043737355995781324, 'R2': 0.8773704488785582}
```

```
for window_size in window_sizes:
  model=bidirec_models[f"win{window_size}"]["model"]
  X_test=data[f"win{window_size}"]["X_test"]
  y_test=data[f"win{window_size}"]["y_test"]
  metrics = evaluate_model(model, X_test, y_test)
  print(f"Metrics for window size {window_size}: {metrics}")
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Metrics for window size 1: {'RMSE': 0.11153474359796996, 'MAE': 0.07644063864709245, 'R2': 0.7391308959520844}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Metrics for window size 4: {'RMSE': 0.07878822075484261, 'MAE': 0.047949035294149615, 'R2': 0.8698258093077587}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Metrics for window size 8: {'RMSE': 0.07607364054600586, 'MAE': 0.04368029610307087, 'R2': 0.8786762194442721}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Metrics for window size 12: {'RMSE': 0.07350894590412564, 'MAE': 0.045551592409925554, 'R2': 0.8867868749179133}
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Metrics for window size 16: {'RMSE': 0.07921331527304018, 'MAE': 0.04523612216034968, 'R2': 0.868578436968097}
```

XAI

```
!pip install shap
```

```
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages (0.47.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from shap) (1.14.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-packages (from shap) (24.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from shap) (4.13.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->shap) (1.17.0)
```

```
import shap
```

```
background = X_train[np.random.choice(X_train.shape[0], 100, replace=False)]
```

Start coding or generate with AI.

```
!pip install tf_keras
```

```
Requirement already satisfied: tf_keras in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: tensorflow<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tf_keras) (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (4.13.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow<2.19,>=2.18->tf_keras) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow<2.19,>=2.18->tf_keras) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow<2.19,>=2.18->tf_keras) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow<2.19,>=2.18->tf_keras) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow<2.19,>=2.18->tf_keras) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow<2.19,>=2.18->tf_keras) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow<2.19,>=2.18->tf_keras) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow<2.19,>=2.18->tf_keras) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow<2.19,>=2.18->tf_keras) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow<2.19,>=2.18->tf_keras) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow<2.19,>=2.18->tf_kera
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow<2.19,>=2.18->tf_keras) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorflow<2.19,>=2.18->tf_keras)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow<2.19,>=2.18->tf_keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow<2.19,>=2.18->tf_keras) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow<2.19,>=2.18->tf_keras) (0.1.2
```

```
import shap
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error


def explain_model(model, X_train, X_test):
    # Create background samples with explicit RNG
    rng = np.random.default_rng(42)
    background_samples = X_train[rng.choice(X_train.shape[0], 50, replace=False)]

    # Ensure model is in inference mode
```

```
    model.trainable = False

    # Use KernelExplainer for better LSTM compatibility
    explainer = shap.KernelExplainer(
        model.predict,
        background_samples,
        rng=rng
    )

    # Calculate SHAP values with proper input reshaping
    return explainer.shap_values(X_test[:200], nsamples=100)
```

Start coding or generate with AI.

Start coding or generate with AI.

```
model=single_models['win12']['model']
X_test=data['win12']['X_test']
y_test=data['win12']['y_test']
X_train=data['win12']['X_train']
```

```
X_test.shape, y_test.shape, X_train.shape
```

```
((1575, 12, 8), (1575,), (7173, 12, 8))
```

```
rx=X_test.reshape(-1,9)
ry=y_test.reshape(-1)
rxt=X_train.reshape(-1,9)
```

```
rx.shape, ry.shape
```

```
((16800, 9), (1575,))
```

```
class LSTMWrapper:
    def __init__(self, model, time_steps, feature_names):
        self.model = model
        self.time_steps = time_steps
        self.feature_names = feature_names

    def predict(self, X):
        # For SHAP analysis, we'll focus on the last time step only
        # Shape X from (num_samples, features) to (num_samples, time_steps, features)
        # by repeating the same values for all time steps
        samples = X.shape[0]
        reshaped_X = np.zeros((samples, self.time_steps, X.shape[1]))

        # Copy the same sample across all time steps (we'll analyze feature importance
        # by varying values at a consistent state)
        for i in range(samples):
            reshaped_X[i, :, :] = np.tile(X[i, :], (self.time_steps, 1))
```

```
        return self.model.predict(reshaped_X, verbose=0).flatten()

background_data = X_test[:100, -1, :]

background_data.shape
```

```
(100, 8)
```

```
feature_names =fet.drop('Usage_kWh')



feature_names = list(feature_names)



feature_names
```

```
['Lagging_Current_Reactive.Power_kVarh',
 'Leading_Current_Reactive_Power_kVarh',
 'CO2(tCO2)',
 'Lagging_Current_Power_Factor',
 'Leading_Current_Power_Factor',
 'NSM',
 'WeekStatus',
 'Load_Type']
```

```
type(feature_names)
```

```
list
```

```
model_wrapper = LSTMWrapper(model, 12, feature_names)

explainer = shap.KernelExplainer(model_wrapper.predict, background_data)



X_to_explain = X_test[:200, -1, :]



# Calculate SHAP values
shap_values = explainer.shap_values(X_to_explain)
```

```
100%                                                 200/200 [12:39<00:00,  4.23s/it]
```

```
feature_names
```

```
['Lagging_Current_Reactive.Power_kVarh',
 'Leading_Current_Reactive_Power_kVarh',
```

```
      'CO2(tCO2)',
      'Lagging_Current_Power_Factor',
      'Leading_Current_Power_Factor',
      'NSM',
      'WeekStatus',
      'Load_Type']


plt.figure(figsize=(10, 8))
shap.summary_plot(
    shap_values,
    X_to_explain,
    feature_names=feature_names,
    plot_type="violin",
    show=False
)
plt.tight_layout()
plt.savefig("lstm_shap_summary_plot.png")
plt.show()
plt.close()
```

```
<ipython-input-497-2cc3b1395ccf>:2: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use the globa
  shap.summary_plot(
```