# Import libraries

import pandas as pd

import numpy as np

import tensorflow as tf

from statsmodels.tsa.statespace.sarimax import SARIMAX

from prophet import Prophet

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import matplotlib.pyplot as plt

import seaborn as sns

import yaml

from google.colab import drive

from pmdarima import auto\_arima

from sklearn import \_\_version\_\_ as sklearn\_version

print("scikit-learn version:", sklearn\_version)

# Mount Google Drive

drive.mount('/content/drive', force\_remount=True)

# Verify file paths

import os

print("SmartCityData contents:", os.listdir('/content/drive/MyDrive/SmartCityData'))

print("ukdale\_h5 contents:", os.listdir('/content/drive/MyDrive/SmartCityData/ukdale\_h5'))

print("ukdale contents:", os.listdir('/content/drive/MyDrive/SmartCityData/ukdale'))

print("metadata contents:", os.listdir('/content/drive/MyDrive/SmartCityData/metadata'))

print("refit contents:", os.listdir('/content/drive/MyDrive/SmartCityData/refit'))

# Dataset paths

uk\_dale\_path = '/content/drive/MyDrive/SmartCityData/ukdale\_h5/ukdale.h5'

refit\_path = '/content/drive/MyDrive/SmartCityData/refit/House\_1.csv'

desnz\_path = '/content/drive/MyDrive/SmartCityData/Headline\_HEE\_tables\_June\_2025.xlsx'

lsoa\_path = '/content/drive/MyDrive/SmartCityData/LSOA\_domestic\_elec\_2010-2023.xlsx'

# Load datasets with error handling

try:

uk\_dale = pd.read\_hdf(uk\_dale\_path, key='building1/elec/meter5')

print("UK-DALE loaded, shape:", uk\_dale.shape)

print("UK-DALE head:", uk\_dale.head())

except Exception as e:

print(f"Error loading UK-DALE: {e}")

try:

refit = pd.read\_csv(refit\_path)

print("REFIT loaded, shape:", refit.shape)

print("REFIT head:", refit.head())

except Exception as e:

print(f"Error loading REFIT: {e}")

try:

desnz = pd.read\_excel(desnz\_path, sheet\_name='T1.1', skiprows=5)

print("DESNZ loaded, shape:", desnz.shape)

except Exception as e:

print(f"Error loading DESNZ: {e}")

try:

lsoa = pd.read\_excel(lsoa\_path, sheet\_name='2010', skiprows=3)

print("LSOA loaded, shape:", lsoa.shape)

except Exception as e:

print(f"Error loading LSOA: {e}")

# Preprocessing

if 'uk\_dale' in locals() and 'refit' in locals():

refit['Time'] = pd.to\_datetime(refit['Time'])

refit.set\_index('Time', inplace=True)

# DESNZ: Convert Installation Month to datetime

if 'desnz' in locals():

desnz.columns = ['Installation Month', 'Total Measures', 'Total Households']

desnz['Installation Month'] = pd.to\_datetime(desnz['Installation Month'], format='%B %Y', errors='coerce')

# LSOA: Use sheet name as year

if 'lsoa' in locals():

lsoa.columns = ['Local Authority Code', 'Local Authority', 'MSOA Code', 'MSOA', 'LSOA Code', 'LSOA', 'Meters', 'Total Consumption (kWh)', 'Mean Consumption (kWh/meter)', 'Median Consumption (kWh/meter)']

lsoa['Year'] = pd.to\_datetime('2010-01-01')

# Filter data for variability (reduced size to avoid timeout)

max\_points = 1000 # Further reduced to 1,000

uk\_dale = uk\_dale[uk\_dale[('power', 'active')] > 0].head(max\_points)

refit = refit[refit['Appliance8'] > 0].head(max\_points)

if 'desnz' in locals():

desnz = desnz[:250]

if 'lsoa' in locals():

lsoa = lsoa[:250]

# Filter invalid data (if Issues column exists) and clean NaN/Inf

if 'Issues' in refit.columns:

refit = refit[refit['Issues'] == 0]

uk\_dale = uk\_dale.replace([np.inf, -np.inf], np.nan).dropna()

refit = refit.replace([np.inf, -np.inf], np.nan).dropna()

# Check data variability

print("UK-DALE power active stats (filtered):", uk\_dale[('power', 'active')].describe())

print("REFIT Appliance8 stats (filtered):", refit['Appliance8'].describe())

# ARIMA Model

def run\_arima(data, column):

train\_size = int(len(data) \* 0.8)

train, test = data[column][:train\_size], data[column][train\_size:]

model = auto\_arima(train, seasonal=True, m=24, trace=False)

fit = model.fit(train)

pred = fit.predict(n\_periods=len(test))

rmse = np.sqrt(mean\_squared\_error(test, pred))

mae = mean\_absolute\_error(test, pred)

r2 = r2\_score(test, pred)

return pred, {'RMSE': rmse, 'MAE': mae, 'R2': r2}

# LSTM Model

def run\_lstm(data, column):

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(data[[column]])

train\_size = int(len(scaled\_data) \* 0.8)

train, test = scaled\_data[:train\_size], scaled\_data[train\_size:]

def create\_sequences(data, seq\_length=24):

X, y = [], []

for i in range(len(data) - seq\_length):

X.append(data[i:i+seq\_length])

y.append(data[i+seq\_length])

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

X\_train, y\_train = create\_sequences(train)

model = tf.keras.Sequential([

tf.keras.layers.LSTM(50, return\_sequences=True, input\_shape=(24, 1)),

tf.keras.layers.LSTM(50),

tf.keras.layers.Dense(1)

])

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

model.fit(X\_train, y\_train, epochs=5, batch\_size=32, verbose=0)

X\_test, y\_test = create\_sequences(test)

pred = model.predict(X\_test)

pred = scaler.inverse\_transform(pred)

y\_test = scaler.inverse\_transform(y\_test)

rmse = np.sqrt(mean\_squared\_error(y\_test, pred))

mae = mean\_absolute\_error(y\_test, pred)

r2 = r2\_score(y\_test, pred)

return pred, {'RMSE': rmse, 'MAE': mae, 'R2': r2}

# Prophet Model

def run\_prophet(data, column):

if isinstance(column, tuple):

print("Data index (UK-DALE):", data.index[:5])

print("Data column values (UK-DALE):", data[column].head())

data\_index = data.index.tz\_localize(None) if data.index.tz is not None else data.index

df = pd.DataFrame({'ds': data\_index, 'y': data[column].values})

print("Prepared DataFrame (UK-DALE):", df.head())

else:

print("Data index (REFIT):", data.index[:5])

print("Data column values (REFIT):", data[column].head())

df = pd.DataFrame({'ds': data.index, 'y': data[column].values})

print("Prepared DataFrame (REFIT):", df.head())

train\_size = int(len(df) \* 0.8)

train, test = df[:train\_size], df[train\_size:]

model = Prophet(yearly\_seasonality=False, weekly\_seasonality=True, daily\_seasonality=False, seasonality\_mode='additive')

model.fit(train)

future = model.make\_future\_dataframe(periods=len(test), freq='s')

forecast = model.predict(future)

pred = forecast['yhat'][train\_size:train\_size + len(test)]

print("Prediction length:", len(pred), "Test length:", len(test))

rmse = np.sqrt(mean\_squared\_error(test['y'], pred))

mae = mean\_absolute\_error(test['y'], pred)

r2 = r2\_score(test['y'], pred)

return pred, {'RMSE': rmse, 'MAE': mae, 'R2': r2}

# Run models with error handling

try:

arima\_pred\_uk, arima\_metrics\_uk = run\_arima(uk\_dale, ('power', 'active'))

print("ARIMA (UK-DALE) completed")

except Exception as e:

print(f"Error in ARIMA (UK-DALE): {e}")

try:

lstm\_pred\_uk, lstm\_metrics\_uk = run\_lstm(uk\_dale, ('power', 'active'))

print("LSTM (UK-DALE) completed")

except Exception as e:

print(f"Error in LSTM (UK-DALE): {e}")

try:

prophet\_pred\_uk, prophet\_metrics\_uk = run\_prophet(uk\_dale, ('power', 'active'))

print("Prophet (UK-DALE) completed")

except Exception as e:

print(f"Error in Prophet (UK-DALE): {e}")

try:

arima\_pred\_refit, arima\_metrics\_refit = run\_arima(refit, 'Appliance8')

print("ARIMA (REFIT) completed")

except Exception as e:

print(f"Error in ARIMA (REFIT): {e}")

try:

lstm\_pred\_refit, lstm\_metrics\_refit = run\_lstm(refit, 'Appliance8')

print("LSTM (REFIT) completed")

except Exception as e:

print(f"Error in LSTM (REFIT): {e}")

try:

prophet\_pred\_refit, prophet\_metrics\_refit = run\_prophet(refit, 'Appliance8')

print("Prophet (REFIT) completed")

except Exception as e:

print(f"Error in Prophet (REFIT): {e}")

# Collect results

results = pd.DataFrame({

'Model': ['ARIMA (UK-DALE)', 'LSTM (UK-DALE)', 'Prophet (UK-DALE)', 'ARIMA (REFIT)', 'LSTM (REFIT)', 'Prophet (REFIT)'],

'RMSE': [arima\_metrics\_uk['RMSE'] if 'arima\_metrics\_uk' in locals() else np.nan,

lstm\_metrics\_uk['RMSE'] if 'lstm\_metrics\_uk' in locals() else np.nan,

prophet\_metrics\_uk['RMSE'] if 'prophet\_metrics\_uk' in locals() else np.nan,

arima\_metrics\_refit['RMSE'] if 'arima\_metrics\_refit' in locals() else np.nan,

lstm\_metrics\_refit['RMSE'] if 'lstm\_metrics\_refit' in locals() else np.nan,

prophet\_metrics\_refit['RMSE'] if 'prophet\_metrics\_refit' in locals() else np.nan],

'MAE': [arima\_metrics\_uk['MAE'] if 'arima\_metrics\_uk' in locals() else np.nan,

lstm\_metrics\_uk['MAE'] if 'lstm\_metrics\_uk' in locals() else np.nan,

prophet\_metrics\_uk['MAE'] if 'prophet\_metrics\_uk' in locals() else np.nan,

arima\_metrics\_refit['MAE'] if 'arima\_metrics\_refit' in locals() else np.nan,

lstm\_metrics\_refit['MAE'] if 'lstm\_metrics\_refit' in locals() else np.nan,

prophet\_metrics\_refit['MAE'] if 'prophet\_metrics\_refit' in locals() else np.nan],

'R2': [arima\_metrics\_uk['R2'] if 'arima\_metrics\_uk' in locals() else np.nan,

lstm\_metrics\_uk['R2'] if 'lstm\_metrics\_uk' in locals() else np.nan,

prophet\_metrics\_uk['R2'] if 'prophet\_metrics\_uk' in locals() else np.nan,

arima\_metrics\_refit['R2'] if 'arima\_metrics\_refit' in locals() else np.nan,

lstm\_metrics\_refit['R2'] if 'lstm\_metrics\_refit' in locals() else np.nan,

prophet\_metrics\_refit['R2'] if 'prophet\_metrics\_refit' in locals() else np.nan]

})

print("Results Table (Table 2):")

print(results)

# Visualization (Figure 7: UK-DALE)

if 'arima\_pred\_uk' in locals() and 'lstm\_pred\_uk' in locals() and 'prophet\_pred\_uk' in locals():

plt.figure(figsize=(10, 6))

plt.plot(uk\_dale.index[-len(arima\_pred\_uk):], uk\_dale[('power', 'active')][-len(arima\_pred\_uk):], label='Actual', color='#1f77b4')

plt.plot(uk\_dale.index[-len(arima\_pred\_uk):], arima\_pred\_uk, label='ARIMA', color='#ff7f0e')

plt.plot(uk\_dale.index[-len(lstm\_pred\_uk):], lstm\_pred\_uk, label='LSTM', color='#2ca02c')

plt.plot(uk\_dale.index[-len(prophet\_pred\_uk):], prophet\_pred\_uk, label='Prophet', color='#d62728')

plt.title('Predicted vs Actual Energy Consumption (UK-DALE-2017 Lighting)')

plt.xlabel('Time')

plt.ylabel('Power (W)')

plt.legend()

plt.grid(True)

plt.savefig('/content/drive/MyDrive/figure\_7\_ukdale.png')

plt.show()

# Visualization (Figure 8: REFIT)

if 'arima\_pred\_refit' in locals() and 'lstm\_pred\_refit' in locals() and 'prophet\_pred\_refit' in locals():

plt.figure(figsize=(10, 6))

plt.plot(refit.index[-len(arima\_pred\_refit):], refit['Appliance8'][-len(arima\_pred\_refit):], label='Actual', color='#1f77b4')

plt.plot(refit.index[-len(arima\_pred\_refit):], arima\_pred\_refit, label='ARIMA', color='#ff7f0e')

plt.plot(refit.index[-len(lstm\_pred\_refit):], lstm\_pred\_refit, label='LSTM', color='#2ca02c')

plt.plot(refit.index[-len(prophet\_pred\_refit):], prophet\_pred\_refit, label='Prophet', color='#d62728')

plt.title('Predicted vs Actual Energy Consumption (REFIT Television Site)')

plt.xlabel('Time')

plt.ylabel('Power (W)')

plt.legend()

plt.grid(True)

plt.savefig('/content/drive/MyDrive/figure\_8\_refit.png')

plt.show()

scikit-learn version: 1.7.0

Mounted at /content/drive

SmartCityData contents: ['metadata.tgz', 'ukdale.zip', 'ukdale.h5.zip', 'Headline\_HEE\_tables\_June\_2025.xlsx', 'LSOA\_domestic\_elec\_2010-2023.xlsx', 'LSOA\_domestic\_gas\_2010-2023.xlsx', 'ukdale\_h5', 'ukdale', 'metadata', 'Processed\_Data\_CSV.7z', 'refit']

ukdale\_h5 contents: ['ukdale.h5']

ukdale contents: ['house\_1', 'house\_2', 'house\_3', 'house\_4', 'house\_5', 'metadata']

metadata contents: ['building3.yaml', 'building1.yaml', 'building2.yaml', 'UK-DALE\_ukerc\_edc\_metadata.yaml', 'building5.yaml', 'meter\_devices.yaml', 'building4.yaml', 'dataset.yaml']

refit contents: ['House\_1.csv', 'House\_2.csv', 'House\_3.csv', 'House\_4.csv', 'House\_5.csv', 'House\_6.csv', 'House\_7.csv', 'House\_8.csv', 'House\_9.csv', 'House\_10.csv', 'House\_11.csv', 'House\_12.csv', 'House\_13.csv', 'House\_15.csv', 'House\_16.csv', 'House\_17.csv', 'House\_18.csv', 'House\_19.csv', 'House\_20.csv', 'House\_21.csv', 'MetaData\_Tables.xlsx', 'CLEAN\_READ\_ME\_081116.txt']

UK-DALE loaded, shape: (19555935, 1)

UK-DALE head: physical\_quantity power

type active

2012-11-09 22:28:18+00:00 0.0

2012-11-09 22:28:24+00:00 0.0

2012-11-09 22:28:30+00:00 0.0

2012-11-09 22:28:36+00:00 0.0

2012-11-09 22:28:42+00:00 0.0

REFIT loaded, shape: (6960008, 12)

REFIT head: Time Unix Aggregate Appliance1 Appliance2 \

0 2013-10-09 13:06:17 1381323977 523 74 0

1 2013-10-09 13:06:31 1381323991 526 75 0

2 2013-10-09 13:06:46 1381324006 540 74 0

3 2013-10-09 13:07:01 1381324021 532 74 0

4 2013-10-09 13:07:15 1381324035 540 74 0

Appliance3 Appliance4 Appliance5 Appliance6 Appliance7 Appliance8 \

0 69 0 0 0 0 0

1 69 0 0 0 0 0

2 68 0 0 0 0 0

3 68 0 0 0 0 0

4 69 0 0 0 0 0

Appliance9

0 1

1 1

2 1

3 1

4 1

DESNZ loaded, shape: (156, 3)

LSOA loaded, shape: (41731, 10)

UK-DALE power active stats (filtered): count 1000.000000

mean 500.838989

std 722.779724

min 1.000000

25% 56.750000

50% 215.000000

75% 337.000000

max 3812.000000

Name: (power, active), dtype: float64

REFIT Appliance8 stats (filtered): count 1000.000000

mean 32.256000

std 3.997805

min 8.000000

25% 30.000000

50% 31.000000

75% 32.000000

max 48.000000

Name: Appliance8, dtype: float64

DEBUG:cmdstanpy:input tempfile: /tmp/tmpwx8mpdh3/314l64uo.json

DEBUG:cmdstanpy:input tempfile: /tmp/tmpwx8mpdh3/wupjbe2w.json

DEBUG:cmdstanpy:idx 0

DEBUG:cmdstanpy:running CmdStan, num\_threads: None

DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan\_model/prophet\_model.bin', 'random', 'seed=43023', 'data', 'file=/tmp/tmpwx8mpdh3/314l64uo.json', 'init=/tmp/tmpwx8mpdh3/wupjbe2w.json', 'output', 'file=/tmp/tmpwx8mpdh3/prophet\_modelt98y89d1/prophet\_model-20250710143735.csv', 'method=optimize', 'algorithm=lbfgs', 'iter=10000']

14:37:35 - cmdstanpy - INFO - Chain [1] start processing

INFO:cmdstanpy:Chain [1] start processing

LSTM (UK-DALE) completed

Data index (UK-DALE): DatetimeIndex(['2012-11-10 10:23:18+00:00', '2012-11-10 10:23:24+00:00',

'2012-11-10 10:23:30+00:00', '2012-11-10 10:23:36+00:00',

'2012-11-10 10:23:42+00:00'],

dtype='datetime64[ns, Europe/London]', freq=None)

Data column values (UK-DALE): 2012-11-10 10:23:18+00:00 7.0

2012-11-10 10:23:24+00:00 11.0

2012-11-10 10:23:30+00:00 10.0

2012-11-10 10:23:36+00:00 10.0

2012-11-10 10:23:42+00:00 3.0

Name: (power, active), dtype: float32

Prepared DataFrame (UK-DALE): ds y

0 2012-11-10 10:23:18 7.0

1 2012-11-10 10:23:24 11.0

2 2012-11-10 10:23:30 10.0

3 2012-11-10 10:23:36 10.0

4 2012-11-10 10:23:42 3.0

14:37:35 - cmdstanpy - INFO - Chain [1] done processing

14:39:01 - cmdstanpy - INFO - Chain [1] start processing

INFO:cmdstanpy:Chain [1] start processing

14:39:01 - cmdstanpy - INFO - Chain [1] done processing

INFO:cmdstanpy:Chain [1] done processing

LSTM (REFIT) completed

Data index (REFIT): DatetimeIndex(['2013-10-09 17:04:48', '2013-10-09 17:04:50',

'2013-10-09 17:05:03', '2013-10-09 17:05:05',

'2013-10-09 17:05:17'],

dtype='datetime64[ns]', name='Time', freq=None)

Data column values (REFIT): Time

2013-10-09 17:04:48 48

2013-10-09 17:04:50 48

2013-10-09 17:05:03 32

2013-10-09 17:05:05 32

2013-10-09 17:05:17 31

Name: Appliance8, dtype: int64

Prepared DataFrame (REFIT): ds y

0 2013-10-09 17:04:48 48

1 2013-10-09 17:04:50 48

2 2013-10-09 17:05:03 32

3 2013-10-09 17:05:05 32

4 2013-10-09 17:05:17 31

Prediction length: 200 Test length: 200

Prophet (REFIT) completed

Results Table (Table 2):

Model RMSE MAE R2

0 ARIMA (UK-DALE) 636.530604 313.245522 -0.216241

1 LSTM (UK-DALE) 305.487763 135.358749 0.744462

2 Prophet (UK-DALE) 617.022249 304.643128 -0.142833

3 ARIMA (REFIT) 2.707635 2.507384 -1.178624

4 LSTM (REFIT) 2.129928 1.604155 -0.200211

5 Prophet (REFIT) 2.052097 1.557110 -0.251405