# Libraries Importing

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)

# Google Drive Mounting

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

# Verifying the 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'))

# Paths of datasets

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'

# Loading 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()