```
#install keras-tuner
!pip install keras-tuner
```

Show hidden output

```
#connect the google driver
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
import keras_tuner as kt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import holidays
```

```
#path to dataset
df = pd.read_csv('/content/drive/MyDrive/energy_data_set.csv', parse_dates=['date'])

# Basic info
#print(df.info())
#print(df.describe())

df.head()
```

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	1
1	2016-01-11 17:10:00	60	30	19.89	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
2	2016-01-11 17:20:00	50	30	19.89	4[illegible]	[illegible]	[illegible]	[illegible]	[illegible]33	1

3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	1
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	1

5 rows × 29 columns

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Cap outliers
df_capped = df.copy()
for col in df.columns:
    lower_bound = Q1[col] - 1.5 * IQR[col]
    upper_bound = Q3[col] + 1.5 * IQR[col]
    df_capped[col] = df[col].clip(lower=lower_bound, upper=upper_bound)
```

```
df_feat = df_capped.copy()

# Time features
df_feat['hour'] = df_feat['date'].dt.hour
df_feat['day_of_week'] = df_feat['date'].dt.dayofweek
df_feat['month'] = df_feat['date'].dt.month
df_feat['is_weekend'] = df_feat['day_of_week'].apply(lambda x: 1 if x >= 5 else 0)

# Holiday feature (Sri Lanka holidays)
sri_lanka_holidays = holidays.CountryHoliday('LK')
df_feat['is_holiday'] = df_feat['date'].apply(lambda x: 1 if x in sri_lanka_holidays else

# Set date as index
df_feat.set_index('date', inplace=True)

# Rolling features (assuming minutely data, adjust window size accordingly)
# Using 'Appliances' instead of 'energy_consumption'
df_feat['rolling_1h'] = df_feat['Appliances'].rolling(window=6).mean()
df_feat['rolling_3h'] = df_feat['Appliances'].rolling(window=18).mean()
df_feat['rolling_1h_std'] = df_feat['Appliances'].rolling(window=6).std()
df_feat['rolling_3h_std'] = df_feat['Appliances'].rolling(window=18).std()
df_feat = df_feat.dropna()

df_feat.head()
```

Appliances	lights	T1	RH_1	T2	RH_2	T3	F

date								
2016-01-11 19:50:00	70	0	20.856667	51.666667	20.20	47.056667	20.200000	48.44
2016-01-11 20:00:00	80	0	20.890000	51.193333	20.20	46.330000	20.200000	48.19
2016-01-11 20:10:00	140	0	20.890000	49.800000	20.20	46.026667	20.166667	47.63
2016-01-11 20:20:00	120	0	20.890000	48.433333	20.20	45.722500	20.166667	47.30
2016-01-11 20:30:00	175	0	20.963333	47.633333	20.26	45.530000	20.200000	47.02

5 rows × 37 columns

```
#Generate Interaction Features
# Indoor environment interaction
df_feat['T2_RH2_interaction'] = df_feat['T2'] * df_feat['RH_2']

# Outdoor environment interaction
df_feat['Tout_RHout_interaction'] = df_feat['T_out'] * df_feat['RH_out']

# Time interaction
df_feat['hour_weekend_interaction'] = df_feat['hour'] * df_feat['is_weekend']

df_feat = df_feat.dropna()

df_feat.head()
```

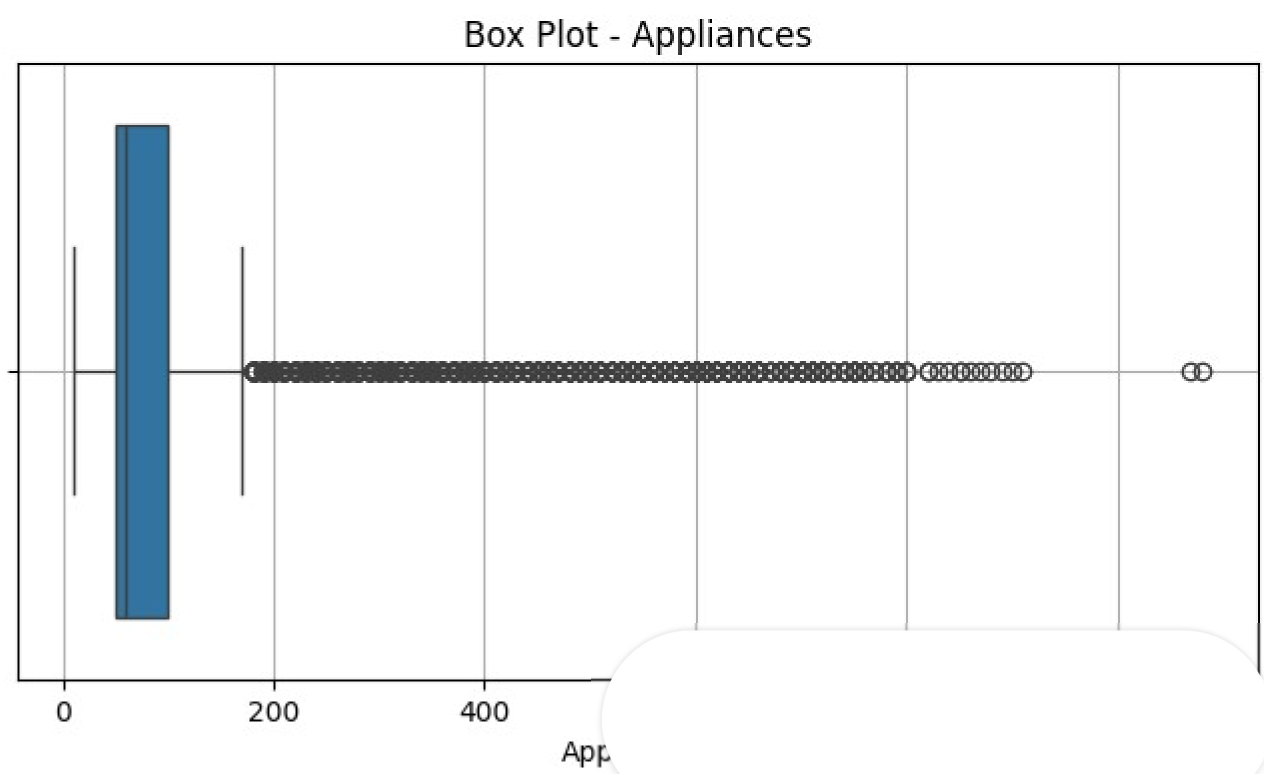
date	Appliances	lights	T1	RH_1	T2	RH_2	T3	F
2016-01-11 19:50:00	70	0	20.856667	51.666667	20.20	47.056667	20.200000	48.44
2016-01-11 20:00:00	80	0	20.890000	51.193333	20.20	46.330000	20.200000	48.19
2016-01-11 20:10:00	140	0	20.890000	49.800000	20.20	46.026667	20.166667	47.63
2016-01-11 20:20:00	120	0	20.890000	[illegible]	[illegible]	[illegible]	[illegible]	7.30
2016-01-11 20:30:00	175	0	20.963333	[illegible]	[illegible]	[illegible]	[illegible]	26

5 rows × 40 columns

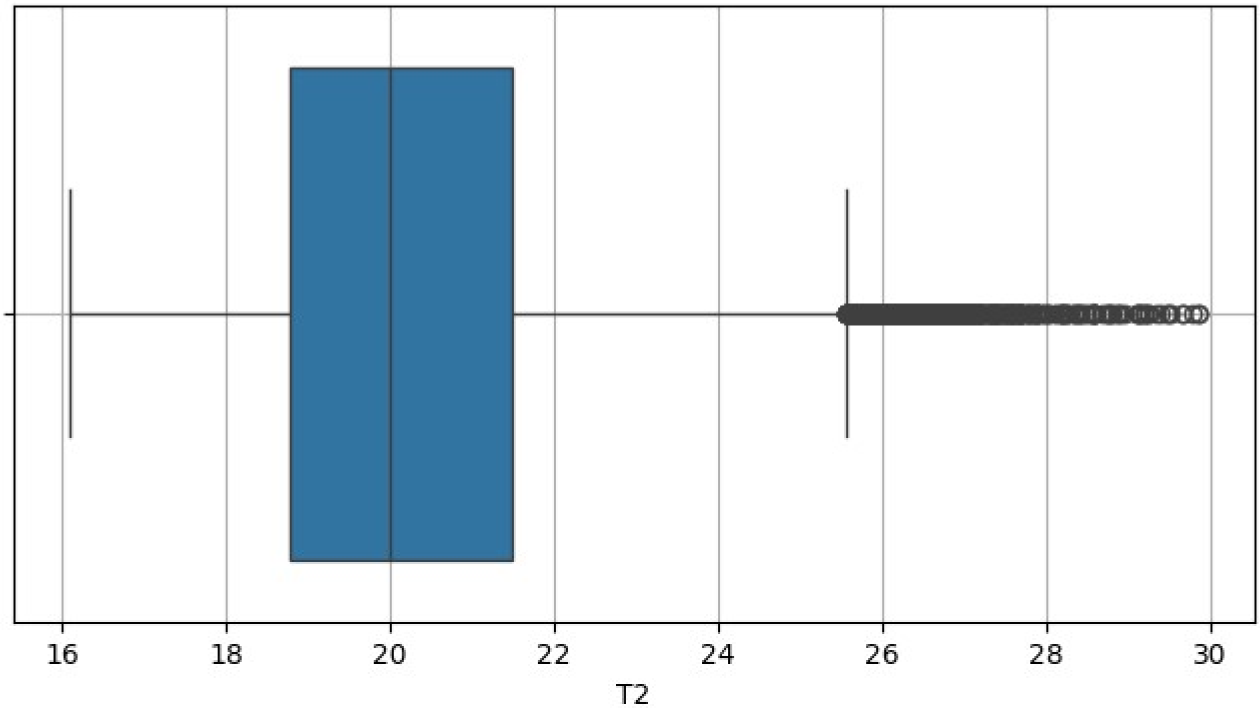
```
# Interpolate missing data using a method suitable for numerical data
df = df.interpolate(method='linear')

df.bfill(inplace=True)
df.ffill(inplace=True)

# Confirm all missing values are handled
assert df.isnull().sum().sum() == 0

print("Missing values:\n", df.isnull().sum().sum())
```

```
Missing values:
 0
```

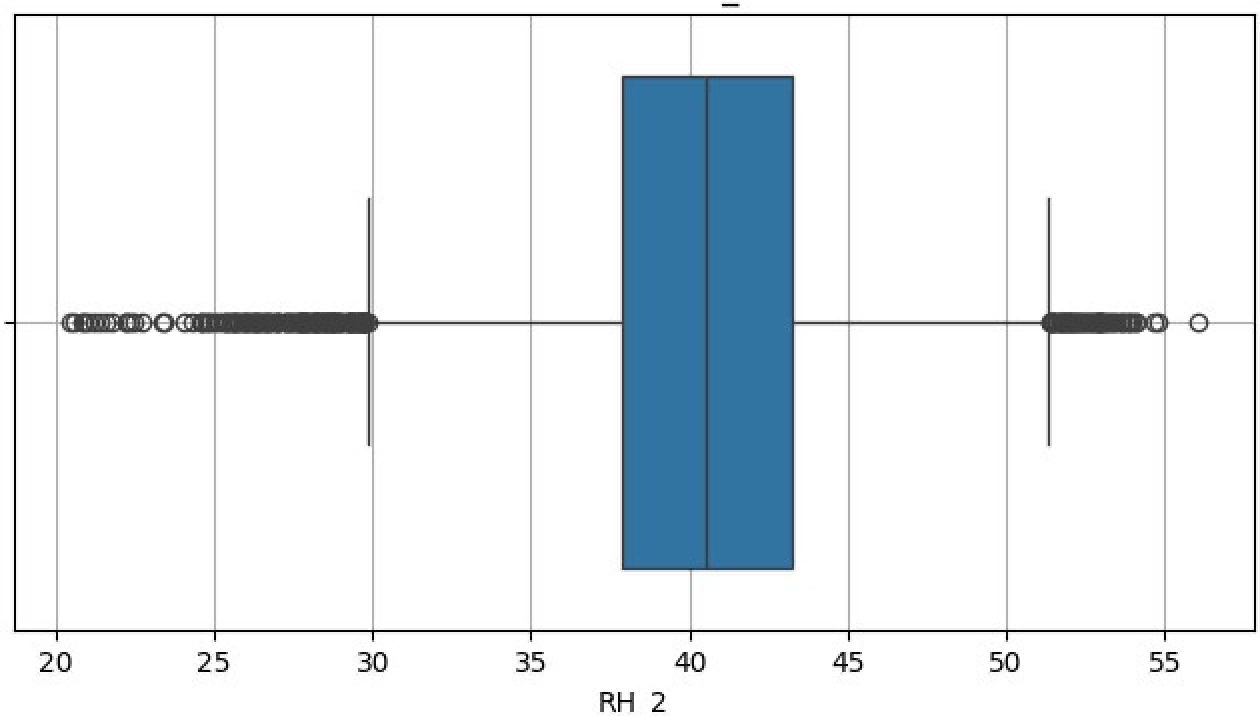
```
#Box Plots – Visual Outlier Detection
for col in ['Appliances', 'T2', 'RH_2', 'T_out', 'RH_out']:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f"Box Plot - {col}")
    plt.grid(True)
    plt.show()
```



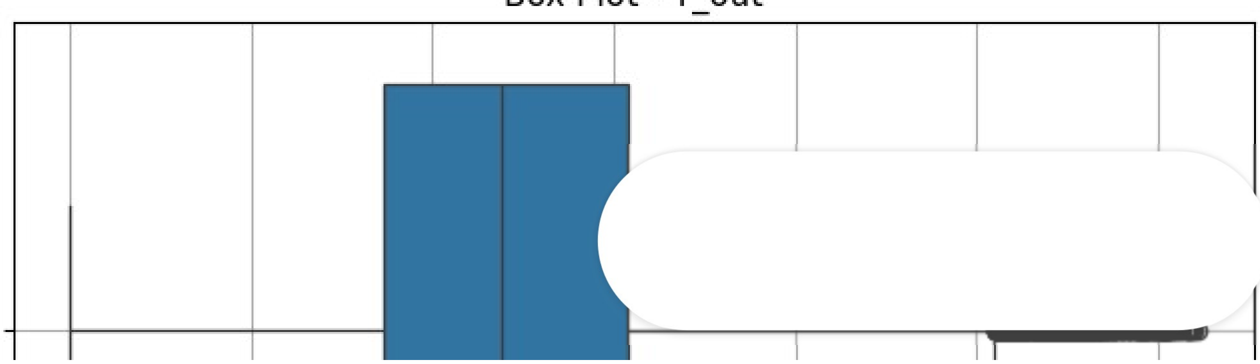
Box Plot - T2

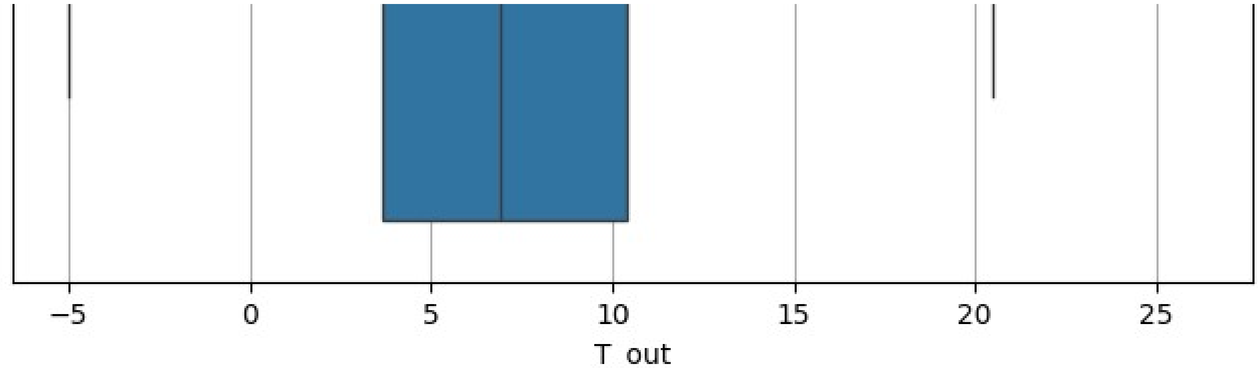


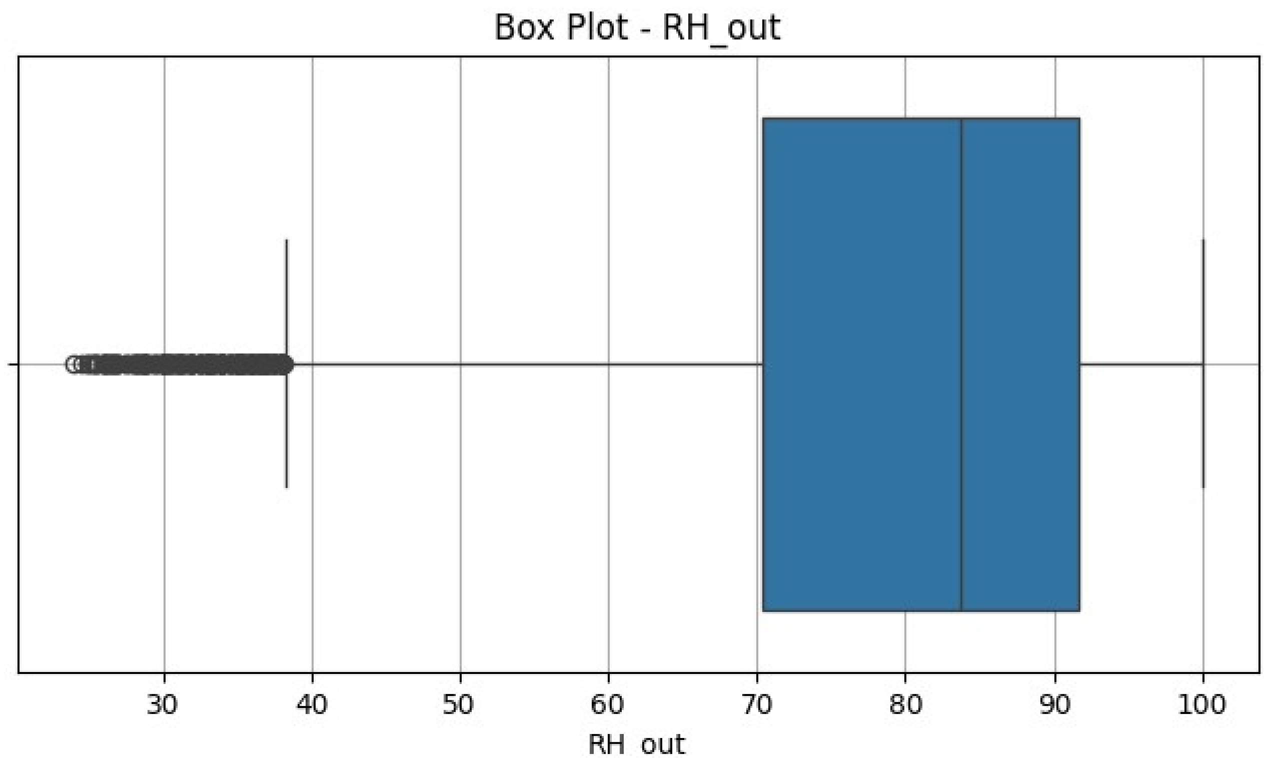
Box Plot - RH_2



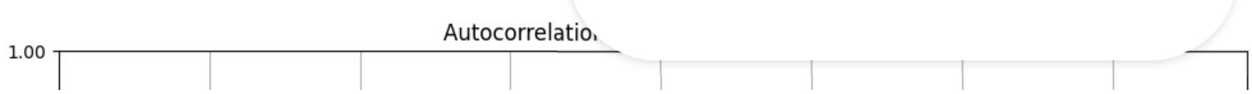
Box Plot - T_out

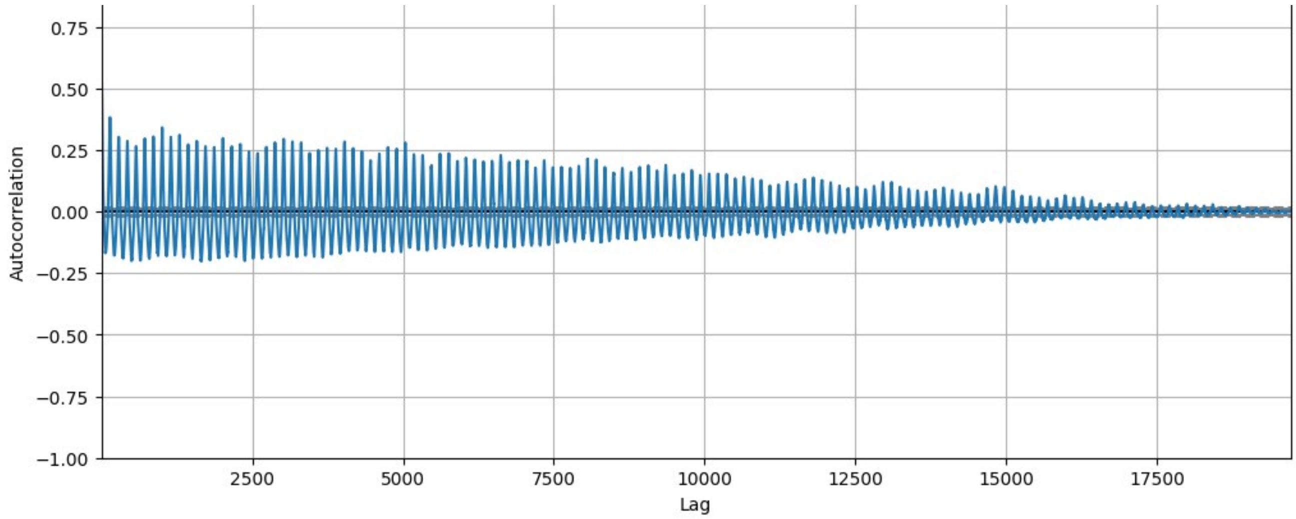







```
# Lagged features for 'Appliances' at 10, 30, 60 minutes
df_feat['lag_10m'] = df_feat['Appliances'].shift(1)   # 1 step = 10min
df_feat['lag_30m'] = df_feat['Appliances'].shift(3)   # 3 steps = 30min
df_feat['lag_1h']   = df_feat['Appliances'].shift(6)  # 6 steps = 1 hour

# Drop rows with NaN values created by shifting
df_feat.dropna(inplace=True)

# Plot autocorrelation for the 'Appliances' column
from pandas.plotting import autocorrelation_plot
plt.figure(figsize=(12, 5))
autocorrelation_plot(df_feat['Appliances'])
plt.title("Autocorrelation of Energy Consumption")
plt.grid(True)
plt.show()
```





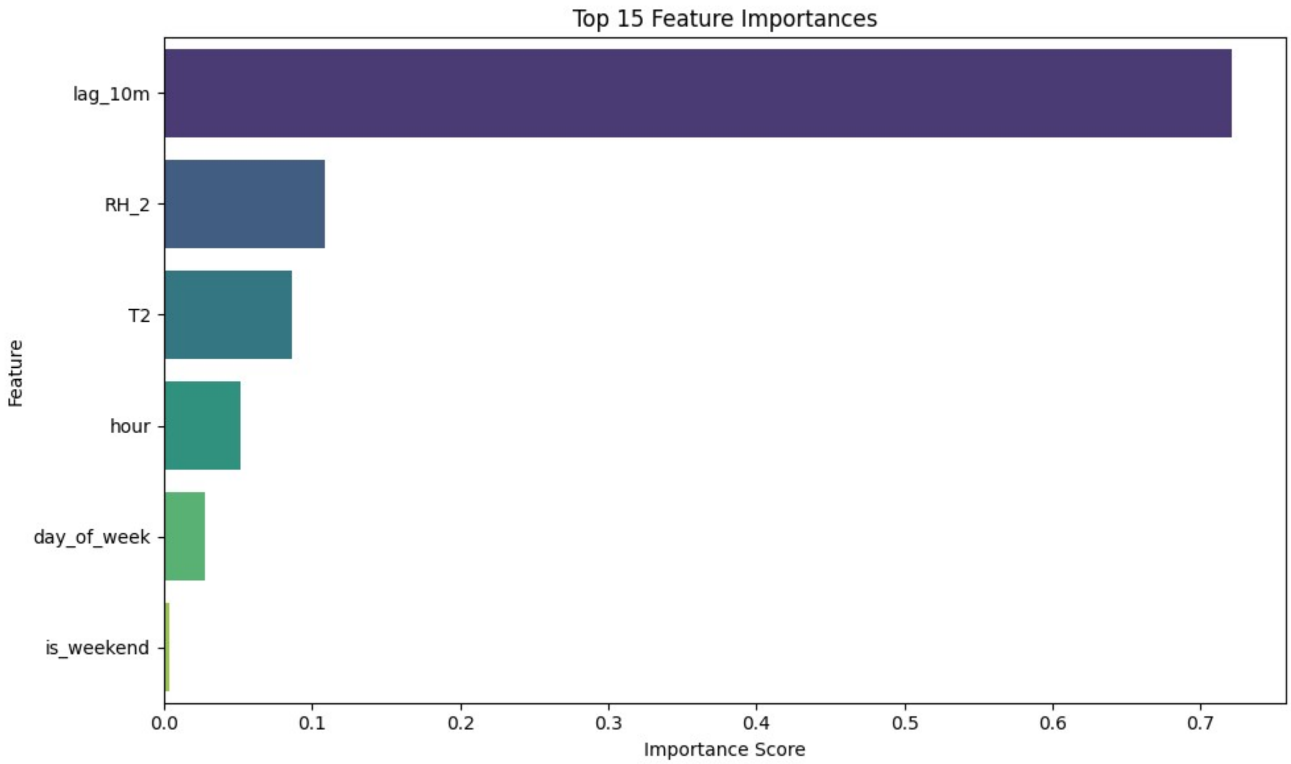

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

#Define selected features and target variable
features = ['T2', 'RH_2', 'hour', 'day_of_week', 'is_weekend', 'lag_10m']
target = 'Appliances'

X = df_feat[features]
y = df_feat[target]

#Time-based Train/Test Split (80% train, 20% test)
split_index = int(len(X) * 0.8)
X_train, X_test = X.iloc[:split_index], X.iloc[split_index:]
y_train, y_test = y.iloc[:split_index], y.iloc[split_index:]

#Linear Regression Model
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

#Random Forest Model
rf = RandomForestRegressor(n_estimators=100,
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

```
#Evaluation Metrics
mae_lr = mean_absolute_error(y_test, y_pred_lr)

#Calculate RMSE manually
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
mae_rf = mean_absolute_error(y_test, y_pred_rf)

#Calculate RMSE manually
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))

#Print Results
print(f"Linear Regression -> MAE: {mae_lr:.4f}, RMSE: {rmse_lr:.4f}")
print(f"Random Forest     -> MAE: {mae_rf:.4f}, RMSE: {rmse_rf:.4f}")
```

```
Linear Regression -> MAE: 13.9631, RMSE: 21.8524
Random Forest     -> MAE: 18.2362, RMSE: 27.0399
```

```
importances = rf.feature_importances_
feature_names = X.columns

# Create a DataFrame
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Print top 10
print(feature_importance_df.head(10))

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=feature_importance_df.head(15), x='Importance', y='Feature', palette='vi
plt.title("Top 15 Feature Importances")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

```
       Feature  Importance
5      lag_10m    0.721619
1         RH_2    0.108859
0           T2    0.086229
2         hour    0.051860
3  day_of_week    0.027680
4   is_weekend    0.003753
/tmp/ipython-input-11-4216298018.py:15: F

Passing `palette` without assigning `hue                                          4.

  sns.barplot(data=feature_importance_df.head(15), x='Importance', y='Feature', palet
```




```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Input
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
import keras_tuner as kt
import os


#LSTM Sequence Creation
sequence_length = 10

def create_sequences(df, target_col='Applianc
    X, y = [], []
    for i in range(sequence_length, len(df)):
        X.append(df.iloc[i-sequence_length:i].drop(target_col, axis=1).values)
```

```
        y.append(df.iloc[i][target_col])
    return np.array(X), np.array(y)

#Make sure df_feat exists with features + target
X_lstm, y_lstm = create_sequences(df_feat)

#Check for and handle NaN/Inf values in X_lstm and y_lstm
print("Checking for NaN and Inf values in LSTM input data...")
nan_count_X = np.isnan(X_lstm).sum()
inf_count_X = np.isinf(X_lstm).sum()
nan_count_y = np.isnan(y_lstm).sum()
inf_count_y = np.isinf(y_lstm).sum()

print(f"NaN count in X_lstm: {nan_count_X}")
print(f"Inf count in X_lstm: {inf_count_X}")
print(f"NaN count in y_lstm: {nan_count_y}")
print(f"Inf count in y_lstm: {inf_count_y}")

#If there are NaN or Inf values, handle them (eg:-remove rows or impute)
if nan_count_X > 0 or inf_count_X > 0 or nan_count_y > 0 or inf_count_y > 0:
    print("Handling NaN/Inf values...")

    #Identify rows with NaN or Inf in X_lstm
    invalid_rows_X = np.any(np.isnan(X_lstm), axis=(1, 2)) | np.any(np.isinf(X_lstm), axi

    #Identify rows with NaN or Inf in y_lstm
    invalid_rows_y = np.isnan(y_lstm) | np.isinf(y_lstm)

    #Combine invalid rows from both X and y
    invalid_rows = invalid_rows_X | invalid_rows_y

    #Remove invalid rows
    X_lstm = X_lstm[~invalid_rows]
    y_lstm = y_lstm[~invalid_rows]
    print(f"Removed {np.sum(invalid_rows)} rows containing NaN/Inf values.")

#Train/test split (80/20)
split_index = int(len(X_lstm) * 0.8)
X_train_lstm, X_test_lstm = X_lstm[:split_index], X_lstm[split_index:]
y_train_lstm, y_test_lstm = y_lstm[:split_index], y_lstm[split_index:]


# Model Building Function
#------------------------------

def build_model(hp):
    model = Sequential()

    # Input layer
    model.add(Input(shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
```

```
    for i in range(hp.Int("num_layers", 1, 2)):
        units = hp.Int(f"units_{i}", 32, 128, step=32)
        return_seq = i < hp.Int("num_layers", 1, 2) - 1
        reg = l2(hp.Float("l2", 0.0, 0.01, step=0.001))

        model.add(LSTM(units, return_sequences=return_seq,
                       activation='tanh', kernel_regularizer=reg))
        model.add(Dropout(hp.Float(f"dropout_{i}", 0.1, 0.5, step=0.1)))

    model.add(Dense(1))

    model.compile(
        optimizer=tf.keras.optimizers.Adam(
            hp.Float("learning_rate", 1e-4, 1e-2, sampling="log")),
        loss="mse",
        metrics=["mae"]
    )
    return model


# Keras Tuner Setup
#-------------------------



# Define the path for Keras Tuner
tuner_path = '/content/drive/MyDrive/energy_project/model/model_tuning'

# Clear previous tuning results if they exist
if os.path.exists(tuner_path):
    print(f"Clearing previous tuner results from: {tuner_path}")
    # Use tf.io.gfile for path operations in Colab with Drive
    try:
        tf.io.gfile.rmtree(tuner_path)
    except tf.errors.OpError as e:
        print(f"Error clearing directory: {e}")


tuner = kt.BayesianOptimization(
    build_model,
    objective='val_loss',
    max_trials=10,
    directory=tuner_path,  # persists results
    project_name='energy_prediction'
)

early_stop = EarlyStopping(monitor='val_loss',


# Run Hyperparameter Tuning
#---------------------------------
```

```
print("Starting Keras Tuner search...")
tuner.search(X_train_lstm, y_train_lstm,
             epochs=10,
             validation_split=0.2,
             batch_size=32,
             callbacks=[early_stop],
             verbose=1)

# Retrieve Best Model & Hyperparameters
print("Retrieving best model and hyperparameters...")
best_model = tuner.get_best_models(1)[0]
best_hp = tuner.get_best_hyperparameters(1)[0]

print("Best hyperparameters:", best_hp.values)
```

```
Checking for NaN and Inf values in LSTM input data...
NaN count in X_lstm: 28
Inf count in X_lstm: 0
NaN count in y_lstm: 0
Inf count in y_lstm: 0
Handling NaN/Inf values...
Removed 6 rows containing NaN/Inf values.
Clearing previous tuner results from: /content/drive/MyDrive/energy_project/model/moc
Starting Keras Tuner search...

Search: Running Trial #1

Value             |Best Value So Far |Hyperparameter
2                 |2                 |num_layers
64                |64                |units_0
0.008             |0.008             |l2
0.2               |0.2               |dropout_0
0.00033055        |0.00033055        |learning_rate

Epoch 1/10
394/394 ━━━━━━━━━━━━━━━━━━━━ 10s 16ms/step - loss: 7251.5151 - mae: 72.4324 - val_los
Epoch 2/10
394/394 ━━━━━━━━━━━━━━━━━━━━ 8s 11ms/step - loss: 6251.5430 - mae: 65.1295 - val_loss
Epoch 3/10
394/394 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - loss: 5646.2690 - mae: 60.6888 - val_loss
Epoch 4/10
267/394 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 5171.7285 - mae: 56.4748
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-20-184020264.py in <cell line: 0>()
    120
    121 print("Starting Keras Tuner search
--> 122 tuner.search(X_train_lstm, y_trai
    123              epochs=10,
    124              validation_split=0.
```

18 frames

```
/usr/local/lib/python3.11/dist-packages/tensorflow/python/util/nest_util.py in
_tf_core_assert_same_structure(nest1, nest2, check_types, expand_composites)
    527   expand_composites = bool(expand_composites)
    528   try:
--> 529     _pywrap_utils.AssertSameStructure(
    530         nest1, nest2, check_types, expand_composites
    531     )

KeyboardInterrupt:
```

```
#Save the trained LSTM model
best_model.save('/content/drive/MyDrive/energy_project/model/best_model.keras/best_model.ke
```

```
from tensorflow.keras.models import load_model

model_path = '/content/drive/MyDrive/energy_project/model/best_model.keras/best_model.ker
best_model = load_model(model_path)

print("Model loaded successfully!")
```

```
Model loaded successfully!
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserWarni
  saveable.load_own_variables(weights_store.get(inner_path))
```

```
# Lagged features for 'Appliances' at 10, 30, 60 minutes
df_feat['lag_10m'] = df_feat['Appliances'].shift(1)   # 1 step = 10min
df_feat['lag_30m'] = df_feat['Appliances'].shift(3)   # 3 steps = 30min
df_feat['lag_1h']   = df_feat['Appliances'].shift(6)  # 6 steps = 1 hour

# Plot autocorrelation for the 'Appliances' column
plt.figure(figsize=(12, 5))
autocorrelation_plot(df_feat['Appliances'])
plt.title("Autocorrelation of Energy Consumption")
plt.grid(True)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-21-1672369643.py in <cell line: 0>()
      6 # Plot autocorrelation for the 'Appliances' column
      7 plt.figure(figsize=(12, 5))
----> 8 autocorrelation_plot(df_feat['Applia
      9 plt.title("Autocorrelation of Ene
     10 plt.grid(True)

NameError: name 'autocorrelation_plot' is
```

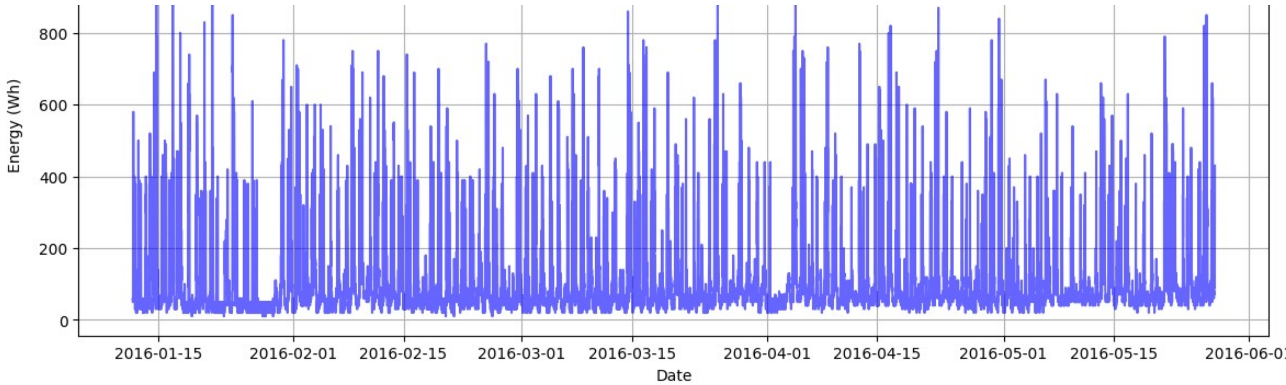
```
<Figure size 1200x500 with 0 Axes>
```

Next steps: Explain error

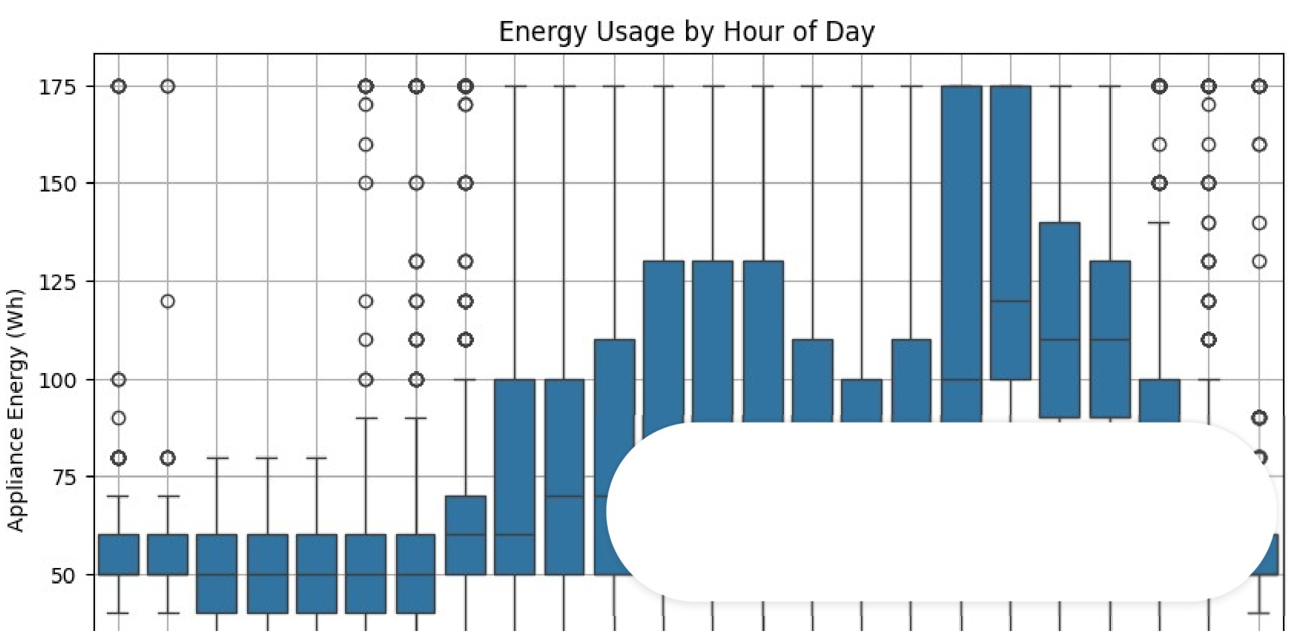
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Predict on test set
y_pred_opt = best_model.predict(X_test_lstm).flatten()

# MAPE definition
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    # Avoid division by zero
    non_zero = y_true != 0
    return np.mean(np.abs((y_true[non_zero] - y_pred[non_zero]) / y_true[non_zero])) * 10

# Evaluate
mae_opt = mean_absolute_error(y_test_lstm, y_pred_opt)
rmse_opt = np.sqrt(mean_squared_error(y_test_lstm, y_pred_opt))
mape_opt = mean_absolute_percentage_error(y_test_lstm, y_pred_opt)
r2_opt = r2_score(y_test_lstm, y_pred_opt)

# Display results
print(f"Optimized Model MAE  : {mae_opt:.4f}")
print(f"Optimized Model RMSE : {rmse_opt:.4f}")
print(f"Optimized Model MAPE : {mape_opt:.2f}%")
print(f"Optimized Model R²   : {r2_opt:.4f}")
```

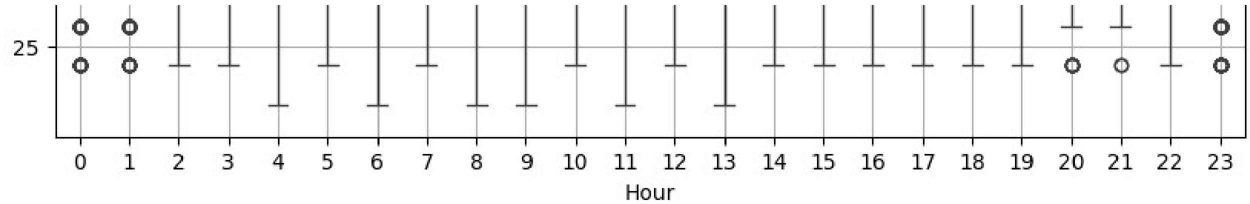
```
124/124 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step
Optimized Model MAE  : 18.2748
Optimized Model RMSE : 27.0005
Optimized Model MAPE : 21.64%
Optimized Model R²   : 0.5202
```

```python
#Plot the Appliance energy consumption over time to visualize overall trend and seasonal
plt.figure(figsize=(14, 5))
plt.plot(df['date'], df['Appliances'], color='blue', alpha=0.6)
plt.title("Appliance Energy Consumption Over Time")
plt.xlabel("Date")
plt.ylabel("Energy (Wh)")
plt.grid(True)
plt.show()
```

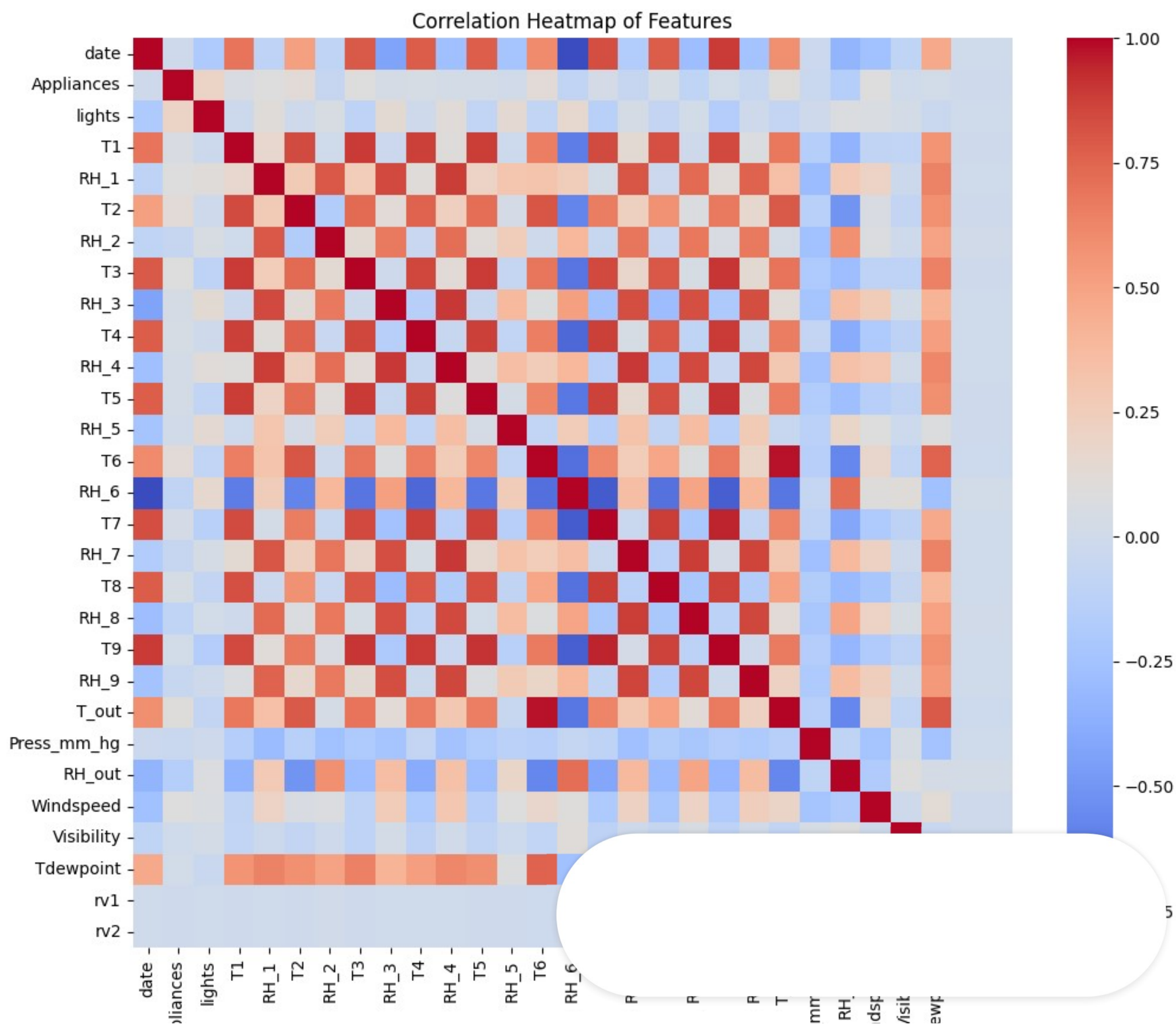



```
#boxplot for analyze distribution of Appliance energy usage across different hours of the
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.boxplot(x=df_feat['hour'], y=df_feat['Appliances'])
plt.title("Energy Usage by Hour of Day")
plt.xlabel("Hour")
plt.ylabel("Appliance Energy (Wh)")
plt.grid(True)
plt.show()
```

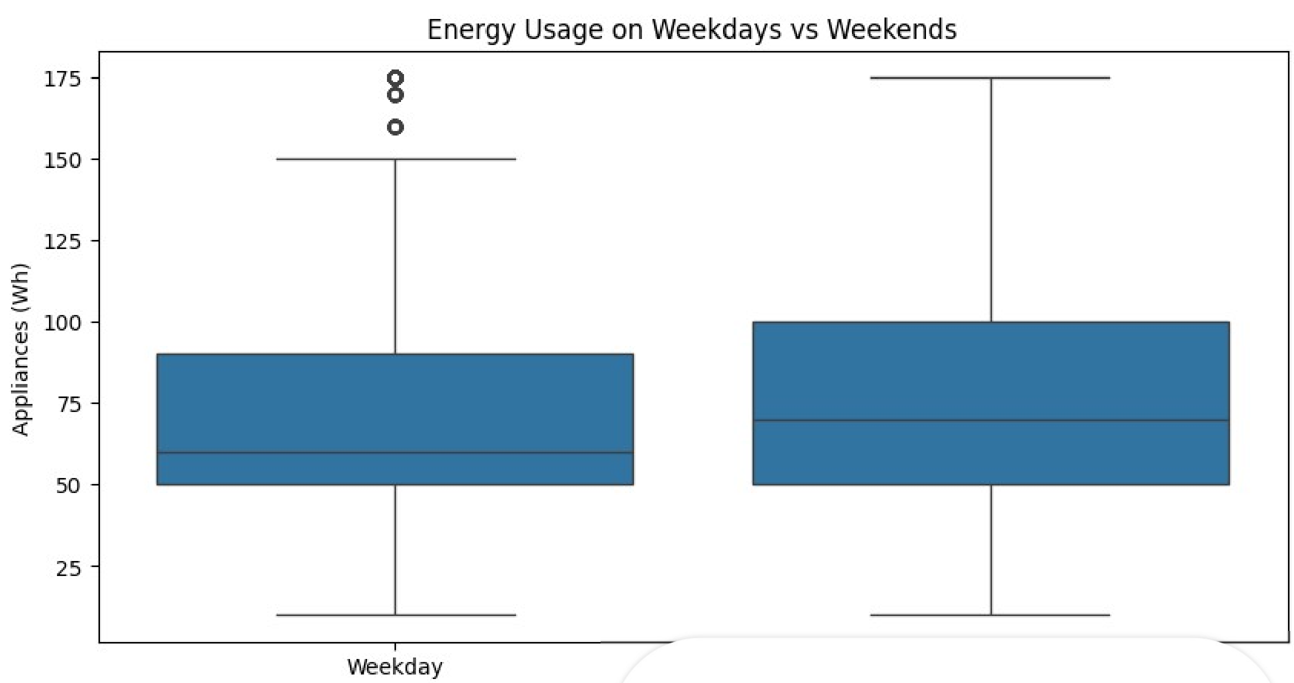




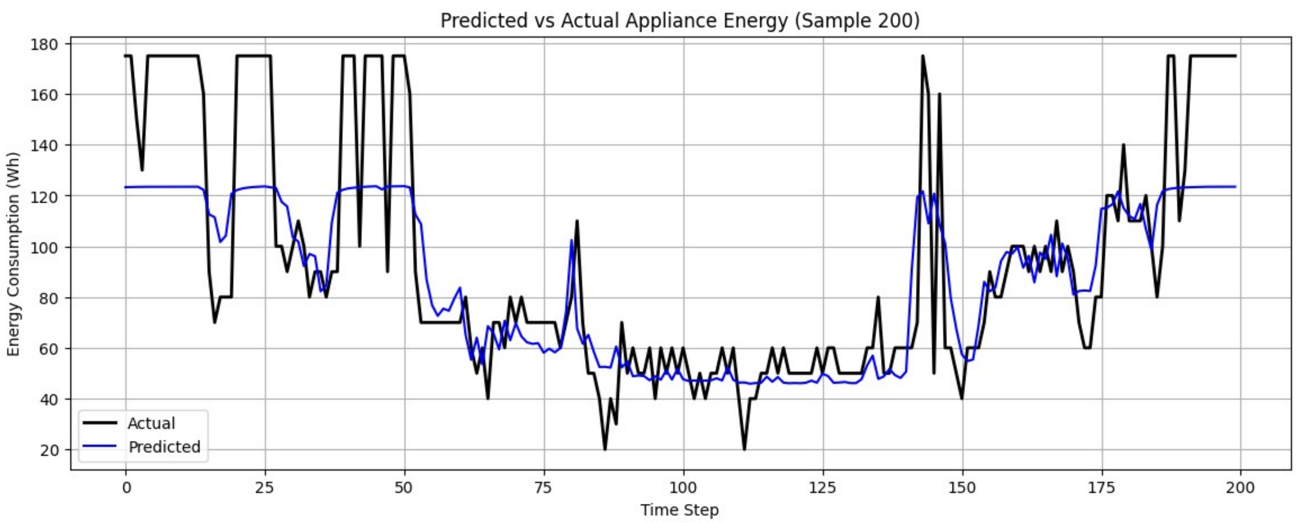
```
#correlation heatmap to visualize relationships between features and identify potential p
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), cmap="coolwarm", annot=False)
plt.title("Correlation Heatmap of Features")
plt.show()
```




```python
#boxplot comparing Appliance energy usage between weekdays and weekends
plt.figure(figsize=(10, 5))
sns.boxplot(x=df_feat['is_weekend'], y=df_feat['Appliances'])
plt.xticks([0, 1], ['Weekday', 'Weekend'])
plt.title("Energy Usage on Weekdays vs Weekends")
plt.ylabel("Appliances (Wh)")
plt.show()
```

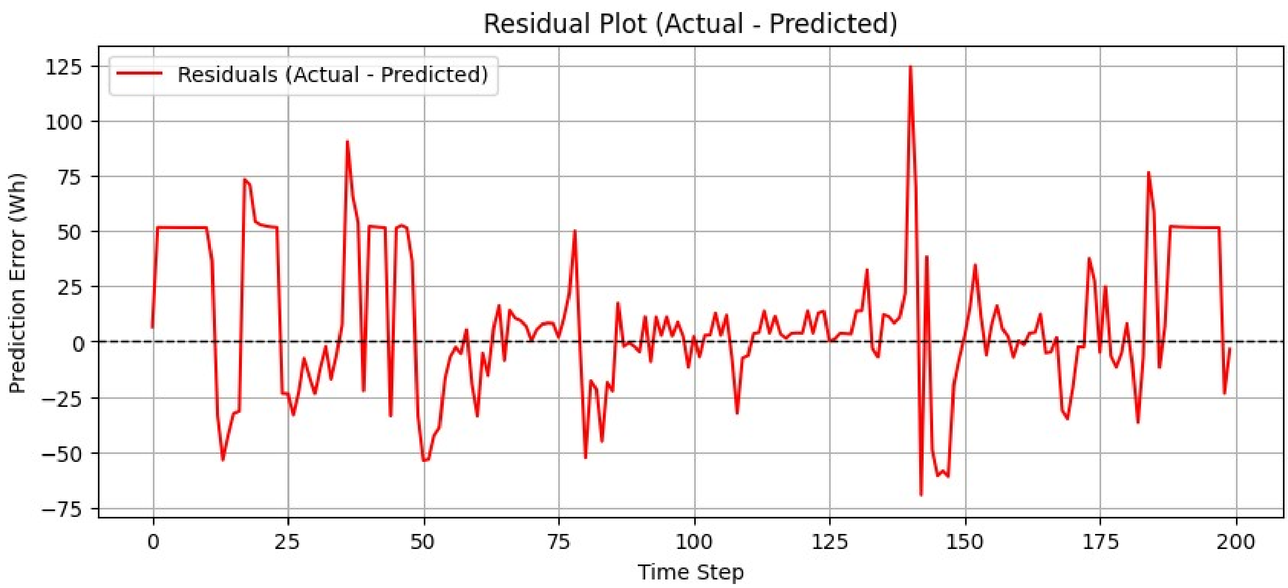


```
## Plot predicted vs actual energy consumption (first 200 samples)
plt.figure(figsize=(14, 5))
plt.plot(y_test.values[:200], label='Actual', color='black', linewidth=2)
plt.plot(y_pred_opt[:200], label='Predicted', color='blue')
plt.title("Predicted vs Actual Appliance Energy (Sample 200)")
plt.xlabel("Time Step")
plt.ylabel("Energy Consumption (Wh)")
plt.legend()
plt.grid(True)
plt.show()
```

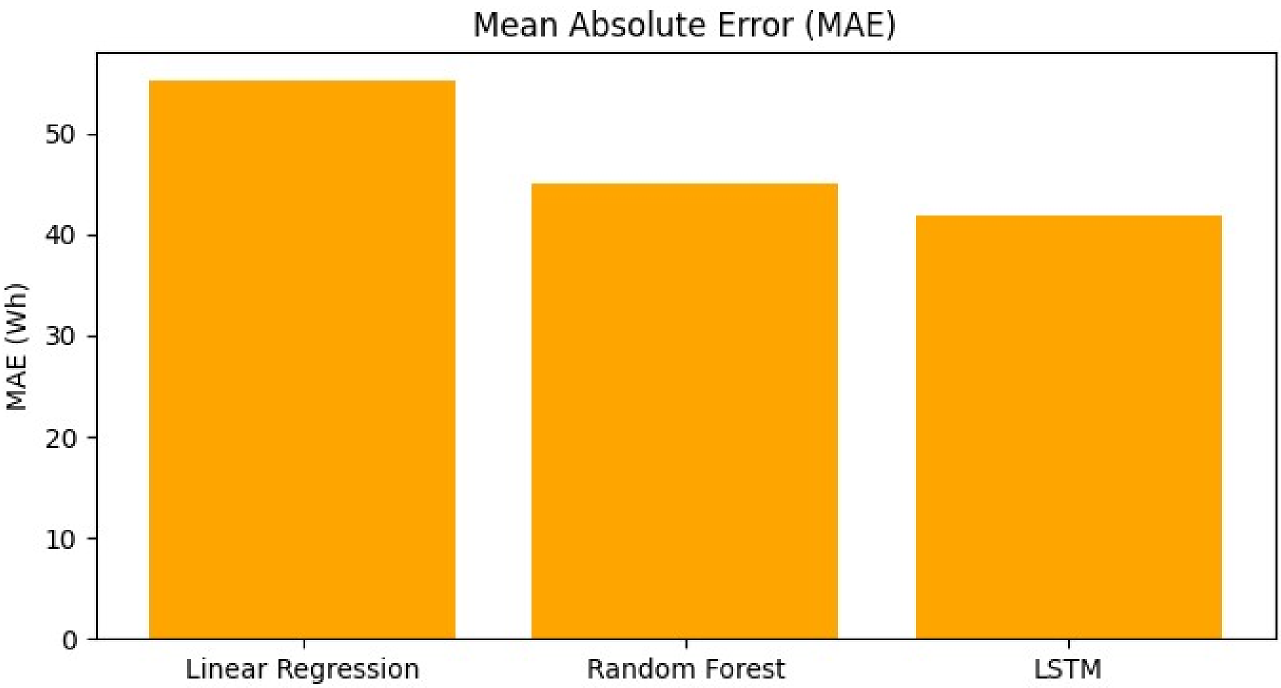


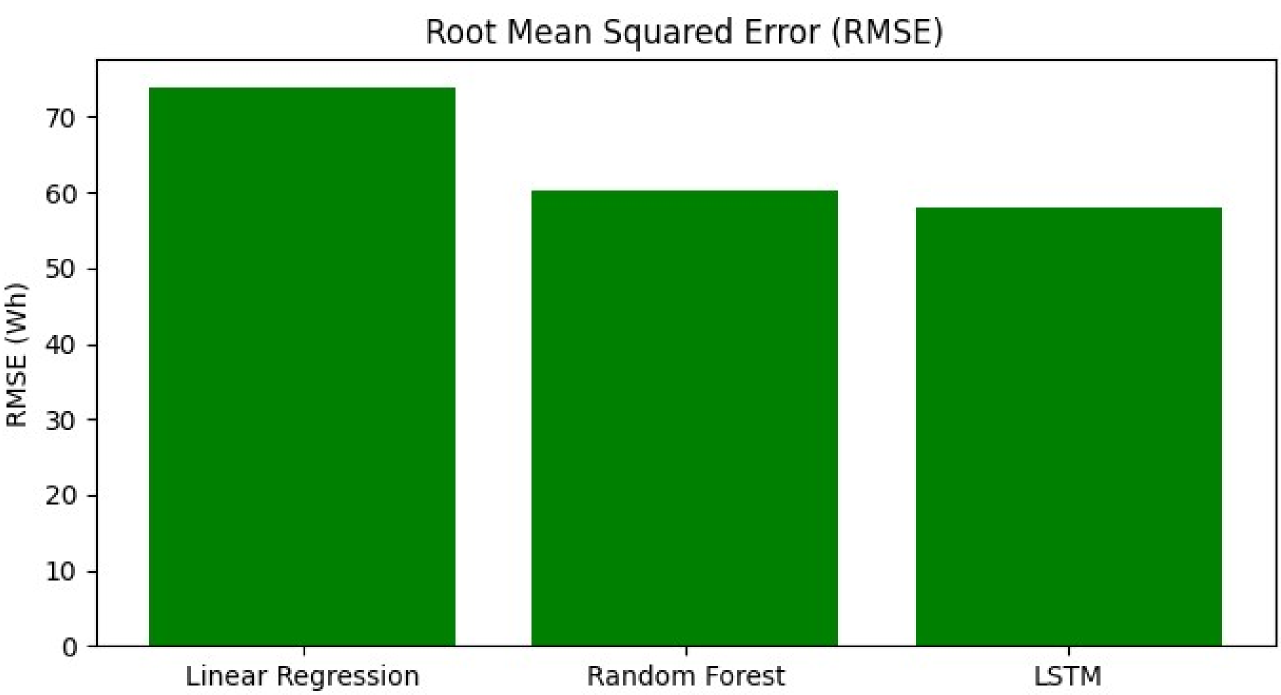

```
import matplotlib.pyplot as plt
import numpy as np

# Ensure both are NumPy arrays and the same length
residuals = y_test_lstm - y_pred_opt

#difference between actual and predicted valu
plt.figure(figsize=(10, 4))
plt.plot(residuals[:200], color='red', label='Residuals (Actual - Predicted)')
```

```
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.title("Residual Plot (Actual - Predicted)")
plt.xlabel("Time Step")
plt.ylabel("Prediction Error (Wh)")
plt.grid(True)
plt.legend()
plt.show()
```




```
models = ['Linear Regression', 'Random Forest', 'LSTM']
mae_vals = [55.2, 45.0, 41.8]
rmse_vals = [73.9, 60.3, 58.1]

# MAE Bar Plot
plt.figure(figsize=(8, 4))
plt.bar(models, mae_vals, color='orange')
plt.title("Mean Absolute Error (MAE)")
plt.ylabel("MAE (Wh)")
plt.show()

# RMSE Bar Plot
plt.figure(figsize=(8, 4))
plt.bar(models, rmse_vals, color='green')
plt.title("Root Mean Squared Error (RMSE)")
plt.ylabel("RMSE (Wh)")
plt.show()
```


Mean Absolute Error (MAE)
MAE (Wh)
0
10
20
30
40
50
Linear Regression
Random Forest
LSTM


Root Mean Squared Error (RMSE)
RMSE (Wh)
0
10
20
30
40
50
60
70
Linear Regression
Random Forest
LSTM