Energy Consumption Prediction in Low Energy Residential Buildings

Introduction:

This report focuses on predicting energy consumption in a low-energy residential building using a dataset with **over 19,000 records**, measured **every 10 minutes**. The dataset includes internal temperature, humidity, and external weather data. Our objective is to identify the most relevant predictors for energy consumption and build robust regression models for accurate prediction.

Drawing from **two papers** that explored energy prediction, we will employ **Random Forest** and **Long Short-Term Memory (LSTM) models** for our analysis. These models were chosen due to their proven success in previous studies:

- **Paper 1** explored different regression models, including linear regression, Support Vector Machines (SVM), Random Forest (RF), and Gradient Boosting (GBM), to predict energy use. They used the Boruta algorithm for feature selection and identified key predictors. Recursive Feature Elimination (RFE) helped determine the optimal number of variables.

- **Paper 2** focused on machine learning models such as SVM, K Nearest Neighbor (KNN), RF, Extreme Random Forest (ERF), and LSTM. They used the Pearson correlation coefficient to identify the most significant features and employed performance metrics like RMSE and R-squared to evaluate model effectiveness.

Our approach incorporates insights from these studies, emphasizing Random Forest and LSTM due to their versatility and proven performance in energy prediction. The evaluation framework will include **key performance measures (MSE, RMSE, MAE, R2)** and cross-validation techniques to ensure model robustness. By using these methods, we aim to develop models that not only achieve high accuracy but also provide reliable predictions for energy use in residential buildings.

- paper1: https://www.sciencedirect.com/science/article/abs/pii/S0378778816308970?via%3Dihub

- paper2:
https://www.researchgate.net/publication/339680663_Prediction_model_of_household_appliance_energy_consumption_based_on_machine_learning

Dataset:

The attributes in this dataset encompass a diverse range of environmental and energy-related indicators, providing valuable insights into factors that influence energy consumption in a low-energy residential building. This dataset is well-suited for **regression analysis** to predict total **energy usage**, considering both internal environmental conditions and external weather-related variables. Further details on the dataset's columns and features are outlined in the accompanying metadata can be found on the code section under Metadata heading.

The energy prediction dataset was loaded into df dataframe shown below

Loading Data to df

```
In [48]: # Reading the UCI_data and storing it in dataframe called df
df = pd.read_csv('C:/Users/arvin/OneDrive/Documents/Study/Data_Science_Sem3/Computational_Machine_Learning/Assignment_2/UCI-elect

In [49]: # printing the first five rows using head() to see if the data is loaded properly
df.head()
```

Out[49]:

	date	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	T5	...	RH_9	T_out	Press_mm_hg	RH_out	W
0	2016-04-19 20:30:00	22.200000	39.500000	20.566667	37.656667	22.230000	37.030000	22.318571	36.610000	20.633333	...	33.90	9.70	766.100000	65.5	
1	2016-03-05 04:40:00	20.356667	37.126667	17.566667	40.230000	20.890000	37.663333	18.700000	36.260000	18.463333	...	41.09	0.30	740.333333	99.0	
2	2016-03-14 12:40:00	20.926667	38.790000	21.100000	35.526667	21.600000	36.290000	21.000000	34.826667	18.100000	...	38.76	4.40	768.466667	72.0	
3	2016-01-22 15:30:00	18.290000	38.900000	17.290000	39.260000	18.390000	39.326667	16.100000	38.790000	16.100000	...	39.20	3.35	760.600000	82.0	
4	2016-02-10 00:40:00	22.290000	42.333333	21.600000	40.433333	22.666667	43.363333	19.100000	40.900000	19.290000	...	43.73	3.20	738.900000	88.0	

- The data is loaded properly

Next exploration of data is crucial to gain insights into the structure and identifying potential issues. The data has 19735 observations with 28 columns. For this dataset, functions like **info(), isna()** and **describe()** were employed to detect missing values and to generate summary statistics.

```
# Using Describe method to see the summary stastistics
df.describe()
```

	T1	RH_1	T2	RH_2	
count	19735.000000	19735.000000	19735.000000	19735.000000	19
mean	21.686571	40.259739	20.341219	40.420420	
std	1.606066	3.979299	2.192974	4.069813	
min	16.790000	27.023333	16.100000	20.463333	
25%	20.760000	37.333333	18.790000	37.900000	
50%	21.600000	39.656667	20.000000	40.500000	
75%	22.600000	43.066667	21.500000	43.260000	
max	26.260000	63.360000	29.856667	56.026667	

8 rows × 27 columns

```
# checking sturcture of the column types and s
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 28 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   date             19735 non-null   object
 1   T1               19735 non-null   float64
 2   RH_1             19735 non-null   float64
 3   T2               19735 non-null   float64
 4   RH_2             19735 non-null   float64
 5   T3               19735 non-null   float64
 6   RH_3             19735 non-null   float64
 7   T4               19735 non-null   float64
 8   RH_4             19735 non-null   float64
 9   T5               19735 non-null   float64
 10  RH_5             19735 non-null   float64
 11  T6               19735 non-null   float64
 12  RH_6             19735 non-null   float64
 13  T7               19735 non-null   float64
 14  RH_7             19735 non-null   float64
 15  T8               19735 non-null   float64
 16  RH_8             19735 non-null   float64
 17  T9               19735 non-null   float64
 18  RH_9             19735 non-null   float64
 19  T_out            19735 non-null   float64
 20  Press_mm_hg      19735 non-null   float64
 21  RH_out           19735 non-null   float64
 22  Windspeed        19735 non-null   float64
 23  Visibility       19735 non-null   float64
 24  Tdewpoint        19735 non-null   float64
 25  rv1              19735 non-null   float64
 26  rv2              19735 non-null   float64
 27  TARGET_energy    19735 non-null   int64
dtypes: float64(26), int64(1), object(1)
memory usage: 4.2+ MB
```

As we can see there are no null values present in the dataset and you can refer to the describe function results more clearly in the .ipynb code file under describe heading.

Inference from describe:

- Temperature features **(T1, T2, ..., T5) vary moderately**, with standard deviations between **1.6 and 2.2 degrees**, indicating possible seasonal changes or indoor environmental control.
- Humidity features **(RH_1, RH_2, ..., RH_9) vary widely**, with **standard deviations from 3.9 to 9**. The **high variability in RH_9** might require further analysis or normalisation.
- The **TARGET_energy** variable has a significant range from **10 Wh to 1110 Wh**, with a standard deviation of 104 Wh. This range suggests fluctuations in activity, seasonality.
- The variability in energy consumption and humidity points to **potential outliers**. Weather-related features like windspeed and visibility show fluctuations, indicating changing weather conditions. Addressing these anomalies is key for accurate predictions.

Performing Downcasting:

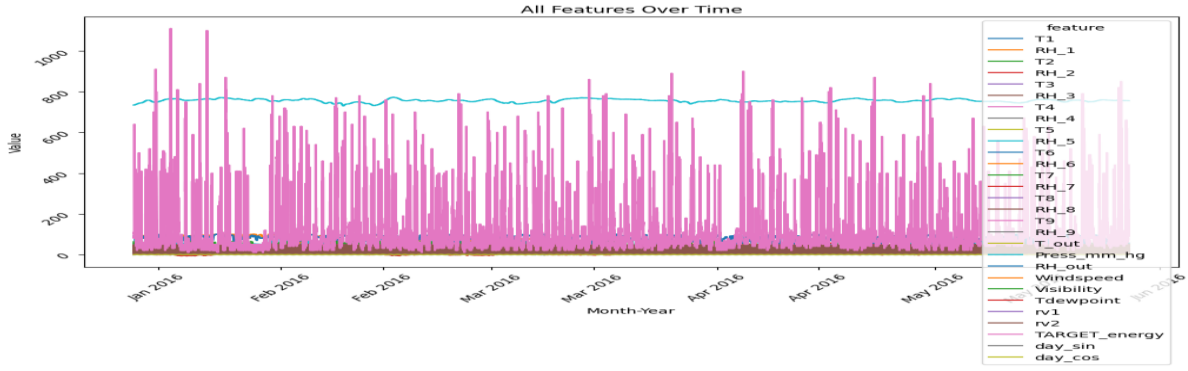
- Similarly, all the objects are in **float64** type except **Target_energy** which is in **int64** type and date is an **object**. You can refer to the code function(downcast_memory_usage(data)) in the .ipynb code file for a much better understanding. We performed downcasting integer and float data types to the smallest possible data type to save memory and improve processing speed. After converting the data types were **reduced to int16 and float16** smaller representations that still encompass the range of data without loss of information.

EDA: Exploratory Data Analysis:

Checking for unique dates (check code for better understanding)

- The data spans from **January to May 2016**, with consistent **10-minute intervals** between each data point.
- This regular spacing creates a reliable time-series structure, useful for analysing trends and patterns in energy consumption.
- With data collected at this interval, we can examine fluctuations within the day, identify peak usage times, and detect broader trends from January to May.

Plotting All Features over time

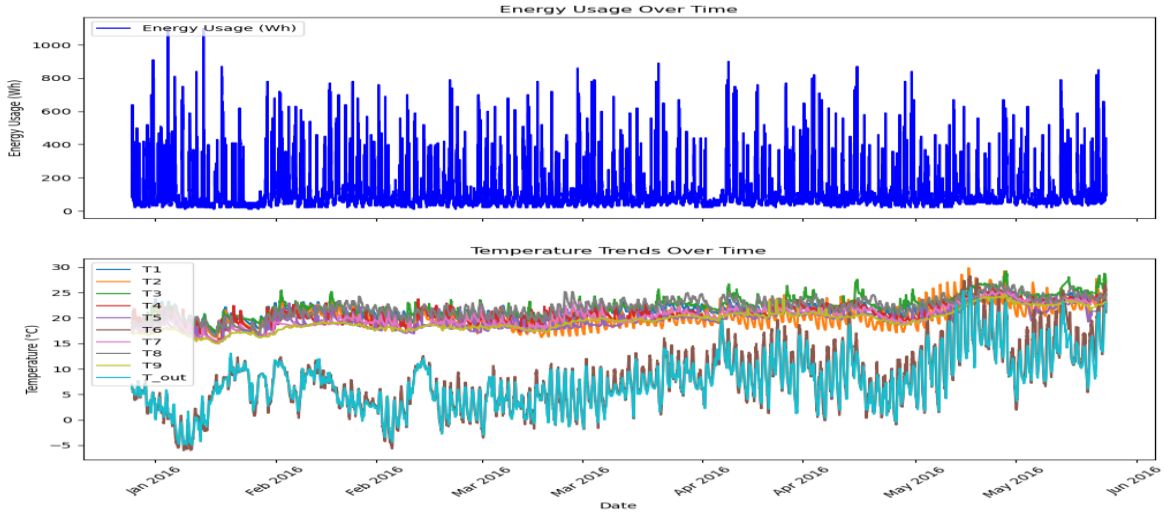


(Plot 1: All Features over time)

Interpretation from All features over time graph

- **Energy Usage Trends**: The graph shows clear peaks and valleys in appliance usage, indicating fluctuations in activity levels. The peaks could represent busy daytime periods, while the valleys might correspond to quieter nighttime hours.
- **Data Gaps**: The graph highlights significant breaks in appliance usage at specific times, like the end of January and early April. These interruptions may be due to data collection issues or other anomalies that need further investigation.

Target_energy vs Temperature features(T1,T2..., T_out)



Plot 2: Target_energy vs Temperature features (T1, T2,…, T_out)

Inference from Energy Usage and Temperature Plots

- **Energy Usage Trends:** The plot shows peaks and valleys, indicating varying energy demand. Peaks likely represent high-activity periods, while valleys suggest quieter times, possibly due to work hours or human routines.
- **Temperature Patterns:** Indoor temperatures vary across rooms, potentially correlating with energy usage, possibly linked to HVAC systems. Outdoor temperature (`T_out`) has a different pattern, likely due to changing weather.
- **Correlation and Anomalies:** There might be a link between temperature changes and energy usage, but gaps in the data suggest possible anomalies or collection issues that require further examination.
- **Implications for Analysis:** Understanding when energy usage peaks could guide energy management strategies. The temperature data provides a broader perspective to understand external factors affecting energy consumption.

Similarly, we have plotted time series EDA with the below combinations and inference is given below

1. **Target_energy** against **Humidity** features **(RH1, RH2,… RHout),**
2. **Target_energy** vs **Outside** weather temperature features **(['T_out',..., 'Windspeed', 'Visibility', 'Tdewpoint']),**
3. **Target_energy** vs **Random** variable features **(rv1, rv2)**

Inference for all these plots below, you can get a better understanding of the plot by seeing the code file

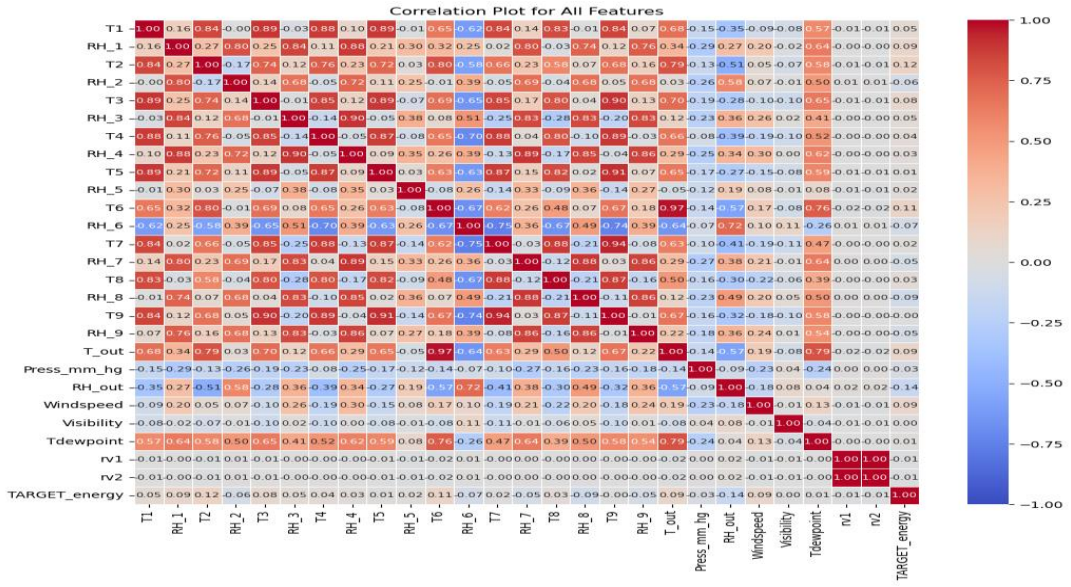
Energy usage in the building appears to be influenced by several factors, particularly humidity levels, temperature, and dewpoint. Peaks in appliance usage tend to align with spikes in bathroom humidity **(RH_5)**, suggesting a correlation with activities like showering or bathing.

Outside temperature **(T_out)** and dewpoint **(Tdewpoint)** show a daily cycle, with **both peaking** during the **day** and **dropping** at **night**, correlating with higher energy usage. This suggests that daytime appliance usage is linked to resident activity.

Outdoor humidity **(RH_6, RH_out)** follows a distinct pattern from **indoor humidity**, indicating the impact of external weather conditions. Other weather variables, such as **windspeed** and **visibility**, show **little correlation** with energy usage. We can't make any clear interpretations between target variable and random variable because the random variable plot is so cluttered.

Overall, these observations suggest that energy usage is **closely related to indoor and outdoor temperature and humidity trends**. Further analysis could help identify additional correlations and improve energy management strategies, offering opportunities to increase energy efficiency.

Correlation Matrix



Plot 3: Correlation Matrix

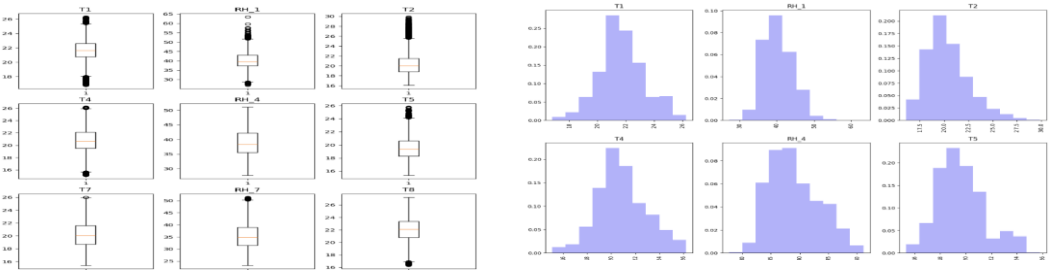
Correlations

Most correlation coefficients have absolute values **below 0.2**, indicating a **weak linear relationship** with **TARGET_energy**. This suggests that these variables have a minimal impact on energy usage.

Positive Weak Correlations: The following variables show a weak positive correlation with TARGET_energy: **T2, T6, RH_1, T_out, Windspeed, T3, T1, RH_3, T4, T8, RH_4, RH_5, T7, T5, Tdewpoint, Visibility.**

Negative Weak Correlations: The following variables exhibit a weak negative correlation with TARGET_energy: **T9, rv2, rv1, Press_mm_hg, RH_9, RH_7, RH_2, RH_6, RH_8, RH_out.**

Outlier Detection and Histogram:



Plot 4: Box Plot and Histogram (Refer code file for complete Chart)

- We can see outliers present in features **T1, RH1, T2, RH2, T3, Rh5, T6, T_out, RH_out, Windspeed, Visibility and Target_energy**.

- As we can see **T2, T3, RH6, Windspeed, T6** is right skewed and features like `**RH_out, Press_mm_hg**` are left skewed. The histogram shows varying scales across features which will be resolved after feature scaling

Creating New Time-Based Features

```
# Creating time-based features from a date column ('date' is the time column)
data_copy['hour'] = pd.to_datetime(data_copy['date']).dt.hour  # Extracting the hour
data_copy['day_of_week'] = pd.to_datetime(data_copy['date']).dt.dayofweek  # Extracting the day of the week
data_copy['is_weekend'] = data_copy['day_of_week'].apply(lambda x: 1 if x in [5, 6] else 0)  # 1 for weekend, 0 for weekday
data_copy['month'] = pd.DatetimeIndex(data_copy['date']).month # Extracting the month
```

In this code, we are trying to enhance the dataset by creating four new features from the 'date' column:

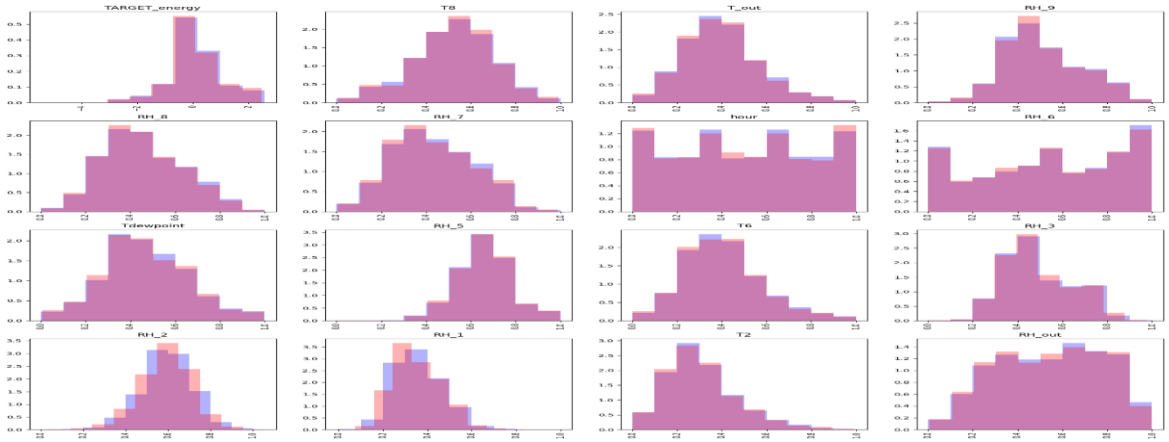
Hour: Extracted the **hour of the day** from the 'date' column. **Day of the Week:** Extracted the day of the **week** from the 'date' column. **Weekend Indicator:** Created a binary indicator to distinguish between **weekdays** and **weekends**. **Month:** Extracted the **month** from the 'date' column. These features add **valuable temporal information** to the dataset.

Splitting data into Train and Test and Feature scaling

Data is splitted into **80% train set** and **20% test set**.

Ensuring fairness in analysis is important. I plotted histograms for the train, and test. The histograms show varying scales before feature scaling, indicating potential differences in the distributions of the data and once it is scaled all the features have come to a single scale. You can refer the histogram chart for original data before transformation in the code file under section **'Plotting Histogram Before Feature Scaling'**

Feature scaling and transformation were conducted to enhance model performance. Initially, **RobustScaler** was employed to standardize features and mitigate **outliers**. **PowerTransformer** was then applied to address skewed distributions. Additionally, **min-max scaling** was performed on specified attributes, while log-normal transformation followed by min-max scaling was applied to others. Finally, the target variable underwent power transformation. Notably, employing robust scaler initially aided in outlier removal, contributing to improved model effectiveness.



Plot 5: Histogram after Feature Scaling

Baseline Model 1: Random Forest:

Fitting, Predicting and Evaluating Baseline Random Forest Model 1

- The Baseline Random Forest model 1 explained **73.3%** of the **variance (R^2 = 0.73254)** with an **average prediction error of 0.52284 units (RMSE).** The **mean absolute error was 0.359090,** indicating a **good overall fit.**

```
# Initializing and training the Random Forest model with random state as 100
rf_model = RandomForestRegressor(random_state=100)
rf_model.fit(X_train_scaled, y_train_scaled)

# Performing prediction for the test set
predictions = rf_model.predict(X_test_scaled)

# Calculating Evaluation Metrics
rf_r2 = r2_score(y_test_scaled, predictions)
rf_mse = mean_squared_error(y_test_scaled, predictions)
rf_rmse = np.sqrt(rf_mse)
rf_mae = mean_absolute_error(y_test_scaled, predictions)

# Printing the Results
print("Random Forest- 2 Model Performance:")
print(f"R^2 Score: {rf_r2:.5f}")
print(f"Mean Squared Error: {rf_mse:.5f}")
print(f"Root Mean Squared Error: {rf_rmse:.5f}")
print(f"Mean Absolute Error: {rf_mae:.5f}")
```

Steps followed to improve the Baseline Model

1. **Feature selection using Boruta Regression Model:** Identifying the most important features.
2. **Eliminating Features with less importance:** Removing Features which contribute less to the model
3. **Performing RandomSearchCV:** Conducting a Randomised search to find the best hyperparameters

Boruta Feature Selection:

```
X = X_train_scaled
y = y_train_scaled
# Initialize Random Forest classifier for Boruta
rf = RandomForestRegressor(n_jobs=-1, random_state=42)  # Random Forest with parallel processing

# Initialize Boruta with Random Forest
boruta = BorutaPy(rf, n_estimators='auto', random_state=42)

# Fit Boruta to the data to identify important features
boruta.fit(X.values,y)

# Get the names of all features and their importance status
feature_importance = {
    'feature': X.columns,
    'support': boruta.support_,  # True if feature is important, False otherwise
    'ranking': boruta.ranking_  # Rank of the feature (1 is the most important)
}

# Convert to a DataFrame for easier visualization
feature_importance_df = pd.DataFrame(feature_importance)

print("Important Features (Boruta):")
print(feature_importance_df[feature_importance_df['support']])  # Show only important features

print("\nFeature Importance Ranking by order:")
print(feature_importance_df.sort_values(by='ranking'))  # Display all features sorted by ranking
```

```
Feature Importance Ranking by order:
         feature  support  ranking
0             T1     True        1
29   day_of_week     True        1
28          hour     True        1
27        day_cos     True        1
26        day_sin     True        1
23      Tdewpoint     True        1
22     Visibility     True        1
21      Windspeed     True        1
20         RH_out     True        1
19   Press_mm_hg     True        1
18          T_out     True        1
17           RH_9     True        1
16             T9     True        1
14             T8     True        1
15           RH_8     True        1
5            RH_3     True        1
12             T7     True        1
11           RH_6     True        1
10             T6     True        1
9            RH_5     True        1
8              T5     True        1
7            RH_4     True        1
6              T4     True        1
13           RH_7     True        1
1            RH_1     True        1
2              T2     True        1
4              T3     True        1
3            RH_2     True        1
31          month    False        2
25            rv2    False        3
24            rv1    False        4
30     is_weekend    False        5
```

- The lower the ranking number, the more significant the feature. Features with support=False don't contribute meaningfully to prediction. So, we will be removing the last **4 features** which are **month**, **rv1**, **rv2** and **is_weekend** where **support = False**.

Random Forest Model 2

- The Random Forest model showed improved performance compared to the previous model. It achieved an **R^2 score** of **0.74181**, increased by **0.8%** indicating better explanation of the variance in the target variable, you can refer Table1 below for better understanding

Hyperparameter Tuning - Performing Random Search and Grid Search

- First, we randomly explore a predefined range of hyperparameters through **random search**. This involves assessing **various combinations of hyperparameters** without any order.
- Next, we **narrow down the range of hyperparameters** that show promise based on the **random search results**. **Grid search CV** then systematically evaluates all possible combinations within this refined range to identify the optimal hyperparameters for the model.

Random Forest Model 3 Using RandomSearchCV

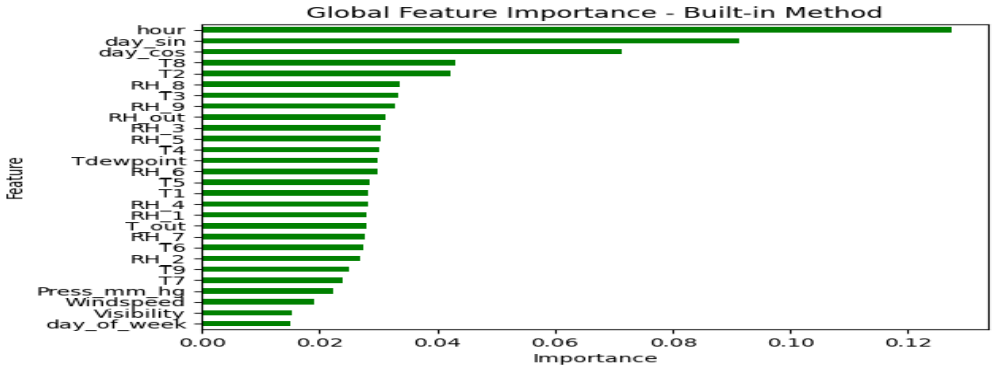
- We used **RandomSearchCV** with the following hyperparameters: **n_estimators (200 to 2000)** choosing **10 n_estimators**, **max_features ('auto' and 'sqrt')**, **max_depth (10 to 110 and**

None), **min_samples_split (2, 5, 10)**, **min_samples_leaf (1, 2, 4)**, and **bootstrap (True and False)**. These parameters were explored to find the optimal model configuration and the best hyperparameters obtained for this model is:

- **Number of Estimators:** 600
- **Minimum Samples Split:** 2
- **Minimum Samples Leaf**: 2
- **Maximum Features:** 'sqrt'
- **Maximum Depth:** 110
- **Bootstrap**: False
- This model performed **best** with an R2 score of **76.7%,** see the **table 1** below for better understanding.

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

Plotting Feature Importance of our Final Random Forest Model 3:



Plot 6: Feature Importance of our Final Model

- From the bar chart, it's evident that features such as **hour**, **day_sin**, **day_cos**, **T8**, and **T2** have the **highest importance** in predicting energy consumption. These variables likely capture significant patterns in **energy usage**, such as **time of day** and **temperature fluctuations**.
- Conversely, **day_of_week**, **visibility**, **windspeed**, and **press_mm_hg** appear to have **lower importance**, suggesting they contribute less to the model's predictive power.

Model Performance with Different Subset and Hyperparameters

Model Performance of Random Forest						
			Testing			
Model	**Features**	**Tuning Hyperparameters**	**R2**	**MSE**	**RMSE**	**MAE**
Baseline Random Forest Model 1	All Features	Standard	0.73254	0.27336	0.52284	0.3591
Random Forest Model 2	All Features except month, rv1, rv2, is_weekend	Standard	0.74181	0.26389	0.5137	0.3528
Random Forest Model 3	All Features except month, rv1, rv2, is_weekend	RandomSearchCV	0.76757	0.23756	0.4874	0.3351

Table 1: Random Forest Model Performance

Comparing our RF model results with paper 1:

In order to compare our results with the paper we trained our **highest performing model** with **unscaled data**. Below are the comparisons between our results

Test Results	**RMSE**	**R2**	**MAE**

paper 1	**68.84**	**0.54**	**31.85**
Final rf model (unscaled data)	**59.04**	**0.66**	**27.06**

Inference:

Through the use of scaling techniques to reduce the impact of outliers and hyperparameter tuning we were able to reduce the **RMSE by 14.24%**, **increase r2 by 22.22%** and **reduce MAE by 15.04%**.

LSTM

Due to the **temporal nature** of the data we decided to utilise a **Convolutional Neural Network (CNN)** and **Long Short-Term Memory model (LTSM)** to predict the energy expenditure. CNN's are good at extracting spatial features and patterns through convolutional layers, efficiently capturing local patterns in data trends. LSTM's are a type of **Recurrent Neural Network (RNN)** that excel at **learning long-term** dependencies as well as sequential patterns. The combination of CNN-LSTM model effectively captures both local and long term dependencies making it a powerful predictor for time series data.

Data preparation

The data for this model was scaled using the same steps as the Random forest model. In order to get the data in a format that can be utilised by the CNN-LSTM model it had to be reshaped using the following functions:

Univariate:

```
def df_to_X_y(df, window_size):
    df_as_np = df.to_numpy()
    X = []
    y = []
    for i in range(len(df_as_np) - window_size):
        row = [[a] for a in df_as_np[i:i + window_size]]
        X.append(row)
        label = df_as_np[i + window_size]
        y.append(label)
    return np.array(X), np.array(y)
```

Multivariate:

```
def df_to_X_y_2(df, window_size):
    df_as_np = df.to_numpy()
    X = []
    y = []
    for i in range(len(df_as_np) - window_size):
        row = [r for r in df_as_np[i:i + window_size]]
        X.append(row)
        label = df_as_np[i + window_size][0]
        y.append(label)
    return np.array(X), np.array(y)
```

These functions reshaped the data into the following formats:

- **Univariate :** [[e1], [e2], … , [en]] , [en+1]
- **Multivariate:** [[e1,v1, … , k1], [e2,v2, … , k2], … , [en, vn, … , kn]] , [en+1]
- **en**= energy value at time n
- **n**= window size
- **V,...,k** = variables v to k.

The **CNN-LSTM model** will then use the values from the specified **window size n** to predict the energy values at **time step n+1**.

Baseline Model.

For the baseline model, we used a **univariate model** that predicted energy expenditure from previous energy values. Baseline Model settings: **{window size=** 6**, learning_rate =** 0.0001**, n_filters =** 64**, kernel_size =** 2**, features =** TARGET_energy **, loss_value =** mse**}**

Performance on test set : { **R2:** 0.70134, **MSE:** 0.006, **MAE:** 0.054, **RMSE:** 0.075 }

Feature selection:

To select features we **paired each feature with the target energy** value and used **multivariate models** to see if the feature **decreased the MSE of the model.**

The following table shows the **reduction percentage** of adding each feature to the baseline model. The feature that **reduced** the **error** the **most** was the **T_8** feature reducing **MSE by 6.1%**

After calculating the reductions in MSE of each variable we tested combinations of multivariate models, adding features based on % increase threshold limits. Eg. add all features that decreased MSE by 1%, 2%, etc… . The **highest-performing model** utilised the features **[T8, day_cos].**

Feature	MSE_Reduction (%)	RH_8	-5.291
T1	-0.706	T9	1.598
RH_1	-1.33	RH_9	0.707
T2	1.512	T_out	0.501
RH_2	1.087	Press_mm_hg	1.148
T3	0.103	RH_out	0.514
RH_3	1.487	Windspeed	1.405
T4	1.362	Visibility	0.761
RH_4	-3.001	Tdewpoint	0.953
T5	-0.417	rv1	0.334
RH_5	1.666	rv2	-2.623
T6	2.333	day_sin	-3.1
RH_6	1.996	day_cos	3.945
T7	-6.139		
RH_7	0.936		
T8	6.179		

Parameter tuning:

Next, we tested all combinations of the following parameter settings with the optimal feature set.

- **Learning rate:** 0.001, 0.00001, 0.000001, 0.0000001
- **Kernel size:** 4, 6
- **Number of filters:** 64, 128, 256

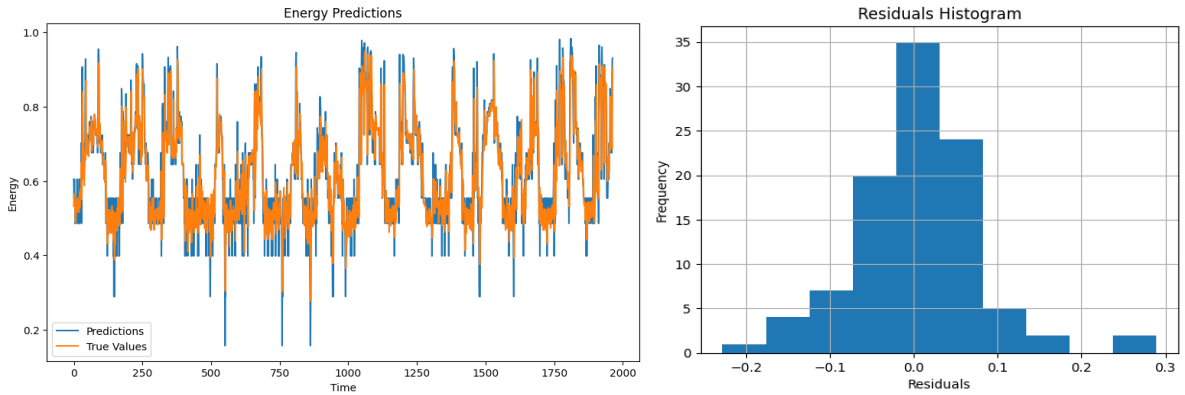
Best performing parameters: learning rate = 1*10^-5, **Kernel size** = 6, **number of filters** = 256

Window size tuning:

Finally, we tested the following window sizes to find the optimal value: 6, 9, 12, 15. Best performing model had a **window size of 15**

Final results:

Base R2: 0.70134 ,**Final R2:** 0.72078 ,**R2 increase:** 2.77124%
Base MSE: 0.00561 ,**Final MSE**: 0.00526 ,**MSE decrease:** 6.21571%
Base MAE: 0.05389 ,**Final MAE**: 0.05193 ,**MAE decrease:** 3.64534%
Base RMSE: 0.07492 ,**Final RMSE:** 0.07255 ,**RMSE decrease:** 3.15771%



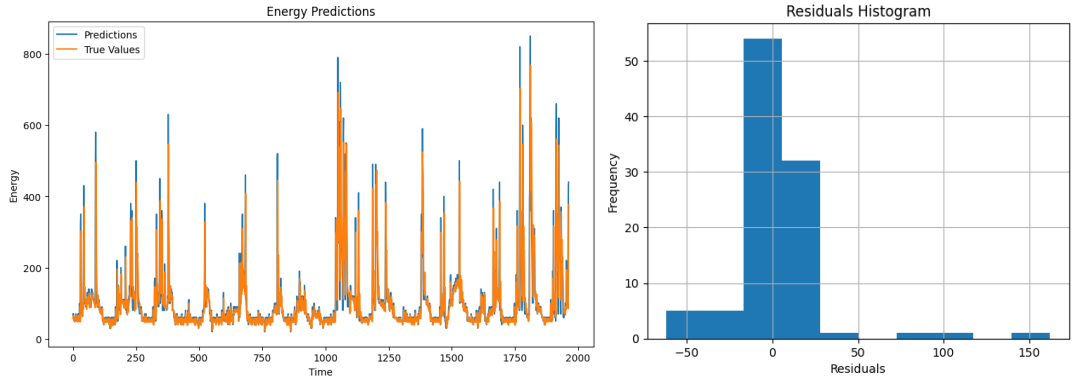
Plot 7: LSTM Final Model Energy Predictions & Residual Analysis

The energy prediction plot for the test set shows that the LSTM model is able to successfully predict the trends of the time-series data. The residual graph for our CNN-LSTM model is **normally distributed around 0**, suggesting that the model is **effective in capturing temporal patterns** in the data, with **consistent variance and independent errors**. Indicating reliability of the model's predictions.

Comparing our LSTM model results with paper2:

In order to compare our results with the paper we trained our highest performing model with unscaled data. Below are the comparisons between our results

Test Results	RMSE	R2
paper2	21.96	0.97
Final LSTM model (unscaled data)	66.02	0.51



Plot 8: LSTM unscaled Final Model Energy Predictions & Residual Analysis

The graphs of the unscaled predictions shows **slightly worse predictions**, **mostly occurring in high values of energy consumption**. From our EDA, we saw the energy data **had a lot of outliers** with much **higher values than the IQR**. The residual plot is skewed towards under predicting the energy consumption, again this is likely due to many outliers with high energy values.

Inference:

Our CNN-LSTM model **had a lower performance** than the **LSTM model in paper 2**. **Utilising unscaled data greatly reduced** the **performance** of the model. From the EDA this is likely due to the large amount of **outliers in the TARGET_energy**. **In future** to improve the performance we would **train the model on scaled data** and implement a function to **inversely scale the predictions**. This would likely **increase the r2** and **decrease RMSE significantly**.

References:

1. Candanedo, L. M., Feldheim, V., & Deramaix, D. (2017). Data driven prediction models of energy use of appliances in a low-energy house. Energy and Buildings, 140, 81–97. https://doi.org/10.1016/j.enbuild.2017.01.083
2. Xiang, L., Xie, T., & Xie, W. (2020). Prediction model of household appliance energy consumption based on machine learning. Journal of Physics: Conference Series, 1453, 012064. https://doi.org/10.1088/1742-6596/1453/1/012064