Appliance Energy Prediction
Using the
Energy Dataset

S. Kabisek

19/04/2025

Table of Contents

1. Introduction

Intelligent systems with the ability to predict consumption trends are required due to the growing demand for effective energy management in modern homes and businesses. The challenge of forecasting appliance energy use with the Appliance Energy Prediction Dataset is addressed in this study. Building a deep learning model that reliably predicts future energy use based on temporal and environmental factors is the main goal.

From exploratory data analysis and feature engineering to the creation, assessment, and refinement of deep learning architectures like LSTM or GRU, the process entails a full pipeline.

2. Data Insights

2.1. Structure of the dataset

- The provided dataset was in shape of 19735 rows and 29 columns, having neither null values nor duplicated records, ensuring a clean foundation for analysis and modeling.

2.2. Summary

	Appliances	lights	T1	RH_1	T2
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	97.694958	3.801875	21.686571	40.259739	20.341219
std	102.524891	7.935988	1.606066	3.979299	2.192974
min	10.000000	0.000000	16.790000	27.023333	16.100000
25%	50.000000	0.000000	20.760000	37.333333	18.790000
50%	60.000000	0.000000	21.600000	39.656667	20.000000
75%	100.000000	0.000000	22.600000	43.066667	21.500000
max	1080.000000	70.000000	26.260000	63.360000	29.856667

	RH_2	T3	RH_3	T4	RH_4
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	40.420420	22.267611	39.242500	20.855335	39.026904
std	4.069813	2.006111	3.254576	2.042884	4.341321
min	20.463333	17.200000	28.766667	15.100000	27.660000
25%	37.900000	20.790000	36.900000	19.530000	35.530000
50%	40.500000	22.100000	38.530000	20.666667	38.400000
75%	43.260000	23.290000	41.760000	22.100000	42.156667
max	56.026667	29.236000	50.163333	26.200000	51.090000

	...	T9	RH_9	T_out	Press_mm_hg
count	...	19735.000000	19735.000000	19735.000000	19735.000000
mean	...	19.485828	41.552401	7.411665	755.522602
std	...	2.014712	4.151497	5.317409	7.399441
min	...	14.890000	29.166667	-5.000000	729.300000
25%	...	18.000000	38.500000	3.666667	750.933333
50%	...	19.390000	40.900000	6.916667	756.100000
75%	...	20.600000	44.338095	10.408333	760.933333
max	...	24.500000	53.326667	26.100000	772.300000

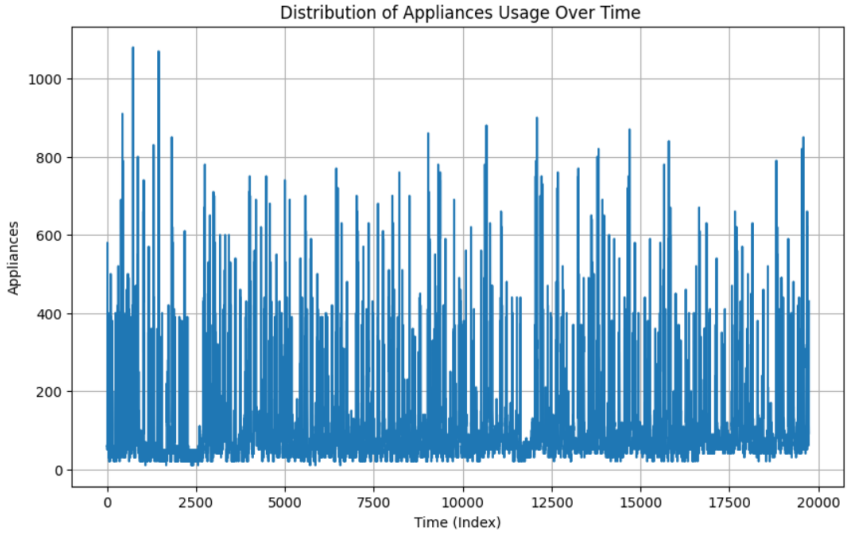
	RH_out	Windspeed	Visibility	Tdewpoint	rv1
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	79.750418	4.039752	38.330834	3.760707	24.988033
std	14.901088	2.451221	11.794719	4.194648	14.496634
min	24.000000	0.000000	1.000000	-6.600000	0.005322
25%	70.333333	2.000000	29.000000	0.900000	12.497889
50%	83.666667	3.666667	40.000000	3.433333	24.897653
75%	91.666667	5.500000	40.000000	6.566667	37.583769
max	100.000000	14.000000	66.000000	15.500000	49.996530

	rv2
count	19735.000000
mean	24.988033
std	14.496634
min	0.005322
25%	12.497889
50%	24.897653
75%	37.583769
max	49.996530

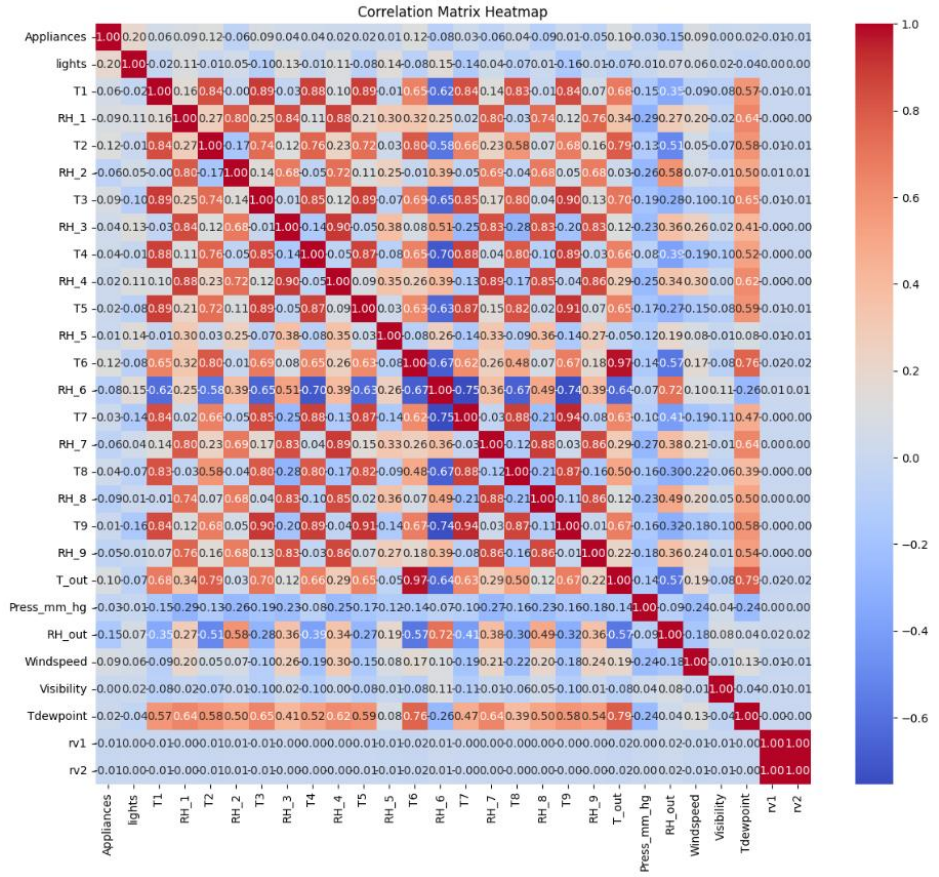
[8 rows x 28 columns]

2.3. Visualizations and Observations

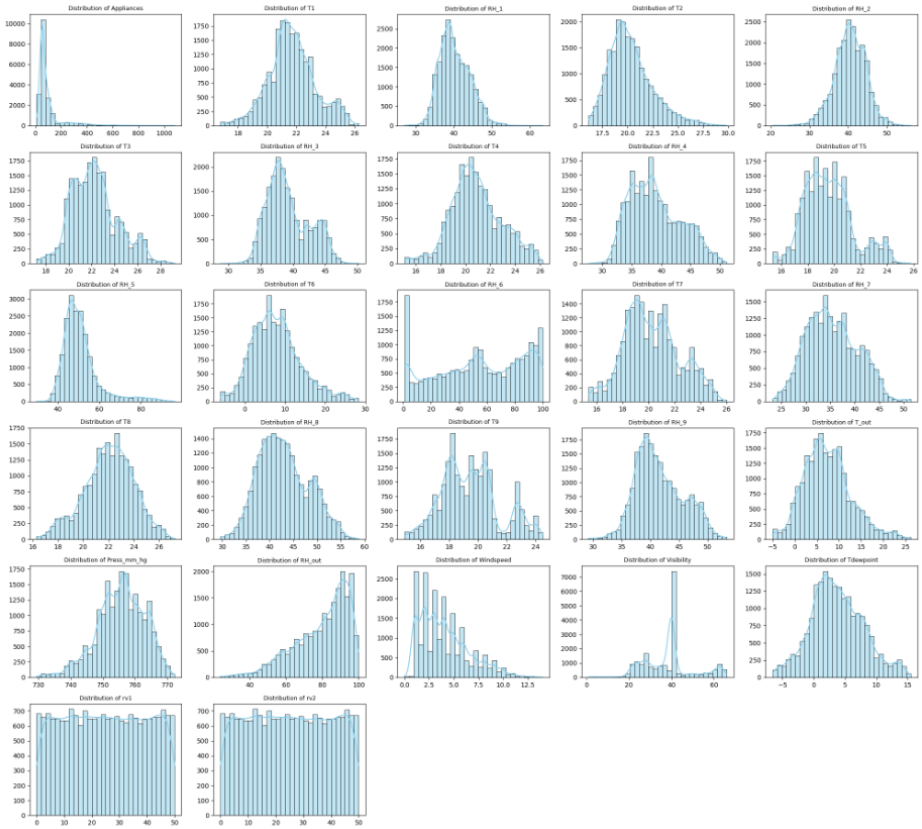
- **Line graph**: This chart shows appliance energy usage over time, highlighting frequent fluctuations and peak consumption patterns. It helps visualize temporal trends and supports identifying high and low usage periods.



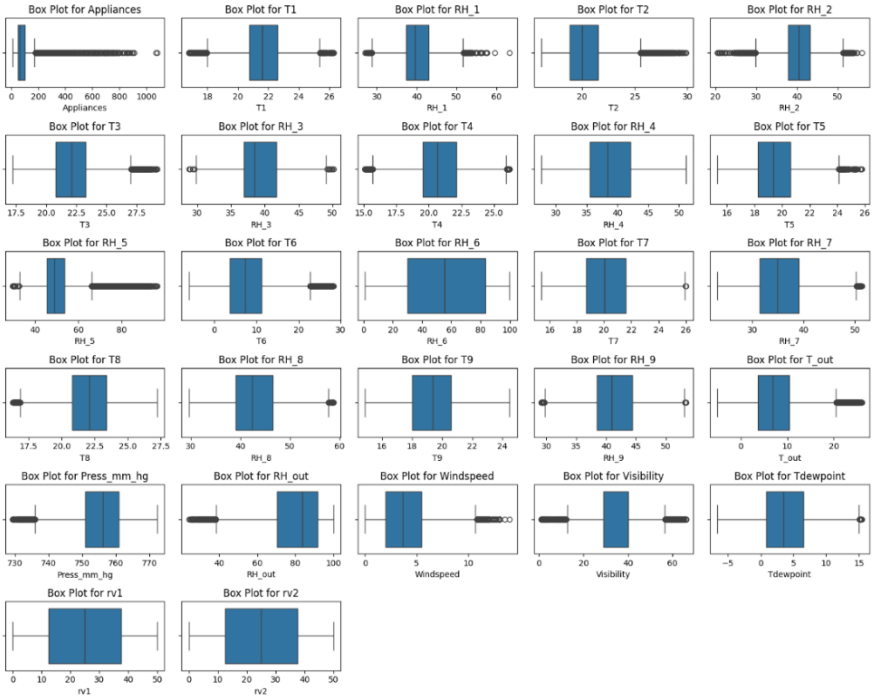
- **Heatmap**: This heatmap visualizes the correlation between numerical features, helping identify strong positive or negative relationships, which is useful for feature selection and understanding variable dependencies.



- **Histogram**: Histograms with KDE plots for each feature, used to visualize data distribution, identify skewness, and detect outliers for better preprocessing and model performance.



- **Box Plot**: box plots for each feature to visualize data spread, detect outliers, and compare distributions, aiding in data cleaning and understanding variable variability.



3. Preprocessing

3.1. Handling missing values

- To handle missing data, I initially checked each column, especially the key ones, for any missing values. However, I found that the dataset had no missing value at all.

```
null_values = data.isna().sum()
print(null_values)
```

```
date           0
Appliances     0
lights         0
T1             0
RH_1           0
T2             0
RH_2           0
T3             0
RH_3           0
T4             0
RH_4           0
T5             0
RH_5           0
T6             0
RH_6           0
T7             0
RH_7           0
T8             0
RH_8           0
T9             0
RH_9           0
T_out          0
Press_mm_hg    0
RH_out         0
Windspeed      0
Visibility     0
Tdewpoint      0
rv1            0
rv2            0
dtype: int64
```

There are no null / missing values in entire dataset

3.2. Handling duplicated records / Data cleaning

- As the next step, to prevent duplication in the analysis, I checked for and removed any duplicate rows. However, there were no duplicate records found in the entire dataset.

```
duplicate_records = data[data.duplicated()]
print(duplicate_records)
```

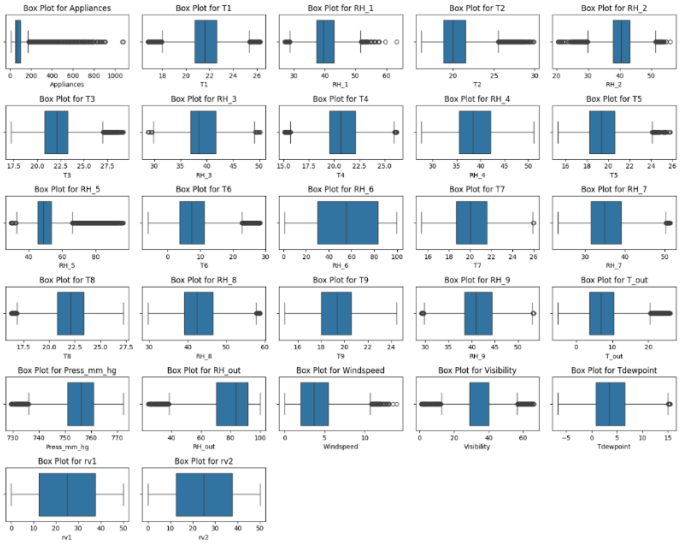
```
Empty DataFrame
Columns: [date, Appliances, lights, T1, RH_1, T2, RH_2, T3, RH_3, T4, RH_4, T5, RH_5, T6, RH_6, T7, RH_7, T8, RH_8, T9, RH_9, T_out, Press_mm_hg, RH_ou
t, Windspeed, Visibility, Tdewpoint, rv1, rv2]
Index: []

[0 rows x 29 columns]
```

There are no duplicacted records in entire dataset

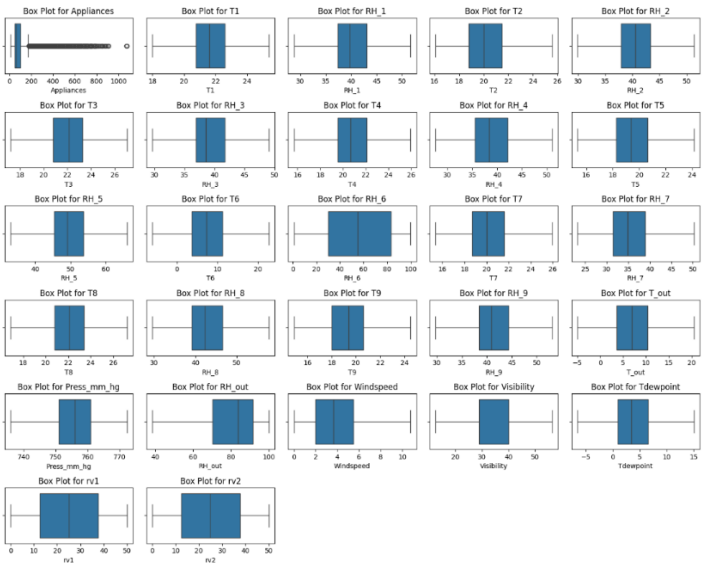
3.3. Detecting Outliers and solving

- Outliers have the potential to skew data patterns and produce unreliable outcomes for machine learning algorithms as well as statistical analysis. We enhance the dataset's overall quality and dependability by addressing them.
- To detect outliers in the dataset, I used the **IQR (Interquartile Range) method**, which identifies values that significantly differ from the majority of the data. After detecting the outliers, I applied the **Capping method** to handle them. This technique limits extreme values by replacing them with more reasonable boundary values, ensuring they don't negatively impact the analysis.

  Before solving outliers:



After solving outliers:



Note: Since "Appliances" is the target variable, didn't make any changes in it.

3.4. Feature scaling

- Standardization was used for feature scaling, as it transforms data to have a mean of 0 and a standard deviation of 1. This method is well-suited for the dataset because it balances the influence of all features, preventing those with larger scales from overpowering the model.

  This approach is especially effective for deep learning models, helping them train faster and more accurately. By ensuring consistent feature scales, standardization supports better learning and improves model performance.

```
print(scaled_data.head())
```

```
                  date  Appliances    lights        T1      RH_1        T2  \
0  2016-01-11 17:00:00          60  3.301264 -1.139072  1.863478 -0.528718
1  2016-01-11 17:10:00          60  3.301264 -1.139072  1.634348 -0.528718
2  2016-01-11 17:20:00          50  3.301264 -1.139072  1.534580 -0.528718
3  2016-01-11 17:30:00          50  4.561378 -1.139072  1.475395 -0.528718
4  2016-01-11 17:40:00          60  4.561378 -1.139072  1.543034 -0.528718

       RH_2        T3      RH_3        T4  ...        T9      RH_9     T_out  \
0  1.092582 -1.245155  1.686863 -0.912635  ... -1.217324  0.958298 -0.147777
1  1.075633 -1.245155  1.705307 -0.912635  ... -1.200778  0.965526 -0.170232
2  1.051570 -1.245155  1.749367 -0.948663  ... -1.233869  0.951070 -0.192686
3  1.042363 -1.245155  1.769859 -0.966677  ... -1.233869  0.926976 -0.215141
4  1.027297 -1.245155  1.769859 -0.966677  ... -1.233869  0.926976 -0.237595

   Press_mm_hg    RH_out  Windspeed  Visibility  Tdewpoint       rv1       rv2
0    -2.684267  0.828905   1.226233    1.799947   0.367016 -0.807974 -0.807974
1    -2.684267  0.828905   1.088599    1.799947   0.343175 -0.440240 -0.440240
2    -2.684267  0.828905   0.950965    1.687977   0.319333  0.252109  0.252109
3    -2.684267  0.828905   0.813331    1.320076   0.295491  1.408801  1.408801
4    -2.684267  0.828905   0.675697    0.952175   0.271649 -1.028122 -1.028122

[5 rows x 29 columns]
```

4. Feature Engineering

4.1. Features created and selected

4.1.1. Time-Based Features

- **Hour**
  Extracted from the date column using .dt.hour, this feature represents the hour of the day (ranging from 0 to 23).
  Relevance: Energy usage varies significantly during the day. For example, consumption typically increases during mornings and evenings when occupants are active. This feature captures such diurnal consumption trends.

- **Day of Week**
  Created by extracting the day index using .dt.dayofweek, where Monday is represented as 0 and Sunday as 6.
  Relevance: Useful in identifying behavioral patterns across the week. It helps distinguish energy usage differences on weekdays versus weekends, which can inform model predictions.

- **Weekend**
  A binary feature derived by applying a condition on the day_of_week column, where values greater than or equal to 5 are marked as 1 (i.e., Saturday and Sunday), and the rest as 0.
  Relevance: Appliance usage behavior tends to differ between weekends and workdays. This feature helps the model learn such periodic changes.

4.1.2. Lagged Features

- **Appliances Lag 10**
  Created by shifting the Appliances column by 10 time intervals (10 minutes in this dataset with 1-minute granularity).

- **Appliances Lag 30**
  A 30-minute lag applied to the Appliances variable using .shift(30).

- **Appliances Lag 60**
  A 60-minute lag applied to the Appliances variable using .shift(60).

Relevance: Lagged features introduce the concept of historical dependence, allowing the model to learn autocorrelation in appliance usage. Since energy consumption often follows previous usage trends (e.g., air conditioners running continuously for hours), lag features help capture this behavior and temporal dynamics.

4.1.3. Statistical Rolling Features

Calculated on the target variable (Appliances) to capture moving averages and local fluctuations in energy usage:

- **Rolling Mean (Window: 3, 6, 12)**
  Rolling average values computed using .rolling(window).mean() over 3, 6, and 12 time steps respectively.
- **Rolling Standard Deviation (Window: 3, 6, 12)**
  Measures of volatility computed using .rolling(window).std() over the same time windows.

Relevance: These features help smooth out short-term fluctuations and capture local trends in energy consumption. The mean gives insight into local average behavior, while the standard deviation reflects variability or instability in usage.

4.1.4. Interaction Features

Interaction features are generated by multiplying pairs of existing variables to capture more complex relationships that may not be linearly visible:

- **T1 × RH_1 Interaction**
  Represents the interaction between indoor temperature in the kitchen (T1) and humidity (RH_1).
- **T_out × RH_out Interaction**
  Represents the interaction between the outdoor temperature (T_out) and outdoor humidity (RH_out).
- **T1 × T2 Interaction**
  A product of indoor temperatures (T1 and T2), capturing combined thermal conditions across different rooms.
- **T2 × RH_2 Interaction**
  Combines temperature (T2) and humidity (RH_2) in another indoor zone to capture localized environmental influence.

Relevance: Interaction terms allow the model to understand more complex dependencies and interrelations between features. For instance, humidity and temperature together might more accurately reflect thermal comfort or HVAC usage than either feature alone.

4.2. Handling missing values

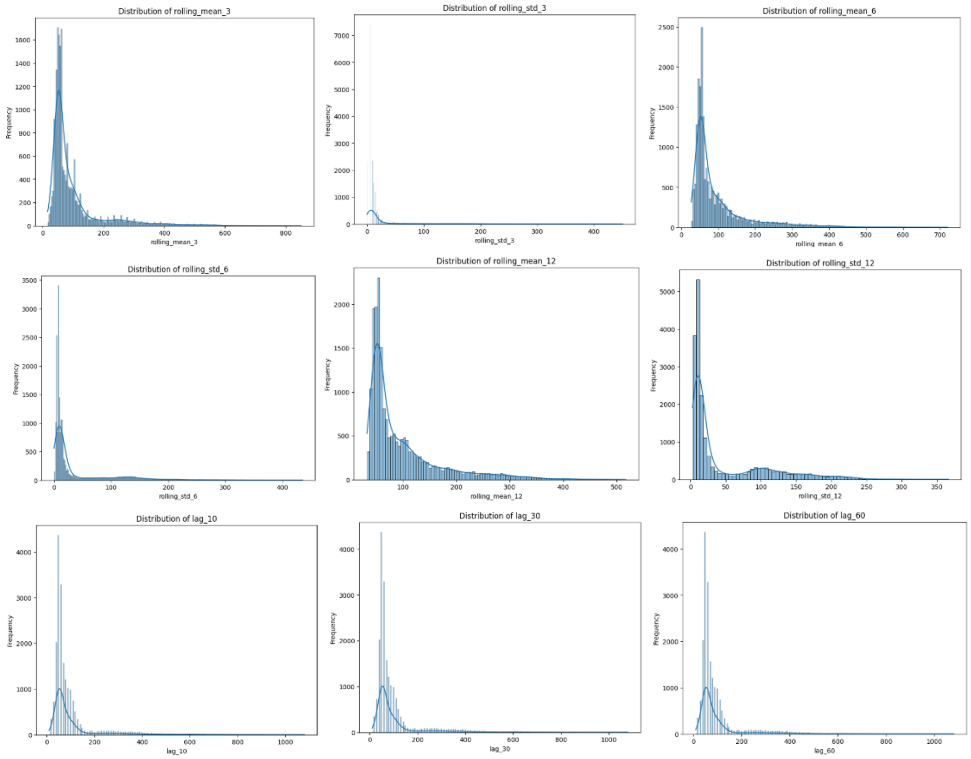
After generating the lagged, rolling, and interaction features, some of the new columns contained null (NaN) values. This occurred because:

- Lag features create NaNs for the initial rows due to the shift.
- Rolling operations generate NaNs for the initial rows that don't meet the window size.

```
: date                          0
  Appliances                    0
  lights                        0
  T1                            0
  RH_1                          0
  T2                            0
  RH_2                          0
  T3                            0
  RH_3                          0
  T4                            0
  RH_4                          0
  T5                            0
  RH_5                          0
  T6                            0
  RH_6                          0
  T7                            0
  RH_7                          0
  T8                            0
  RH_8                          0
  T9                            0
  RH_9                          0
  T_out                         0
  Press_mm_hg                   0
  RH_out                        0
  Windspeed                     0
  Visibility                    0
  Tdewpoint                     0
  rv1                           0
  rv2                           0
  hour                          0
  month                         0
  year                          0
  rolling_mean_3                2
  rolling_std_3                 2
  rolling_mean_6                5
  rolling_std_6                 5
  rolling_mean_12              11
  rolling_std_12               11
  lag_10                        1
  lag_30                        3
  lag_60                        6
  in_temp_hum_interaction       0
  out_temp_hum_interaction      0
  T1_T2_interaction             0
  T2_RH_2_interaction           0
  temp_diff                     0
  temp_ratio                    0
  day                           0
  weekday                       0
  weekend                       0
  dtype: int64
```

To address this:

- **Imputation Strategy**: First saw the distribution diagram of columns with missing values.



- The features involved (especially those related to appliance usage) were not normally distributed and showed skewness. So, missing values filled using **Median** value. Using the median, which is robust to outliers, preserves the central tendency without being influenced by extreme values.

Relevance: Proper handling of missing values ensures that the model receives complete and consistent data during training, thereby preventing performance degradation or biased learning due to missing entries.

- After filling missing values:

```
rolling_mean_3          0
rolling_std_3           0
rolling_mean_6          0
rolling_std_6           0
rolling_mean_12         0
rolling_std_12          0
lag_10                  0
lag_30                  0
lag_60                  0
```

4.3. Feature selection Techniques

- **Tree-Based Feature Importance**: Used to assess the contribution of each feature in ensemble models (like Random Forest).
  Why used:
  - Helps identify the features that most strongly influence predictions.
  - Quick and works well for datasets with both numerical and categorical variables.
- **Recursive Feature Elimination (RFE)**: Iteratively removed least important features to identify the optimal subset.
  Why used:
  - Efficient in reducing model complexity and overfitting.
  - Helps find a minimal feature subset that gives the best model performance.
- **Lasso Regularization (L1)**: Shrinks coefficients of less important features to zero, effectively selecting variables.
  Why used:
  - Automatically performs feature selection by eliminating irrelevant features.
  - Particularly useful when dealing with high-dimensional datasets.
- **Permutation Feature Importance**: Evaluated feature impact by measuring model performance drop when each feature is randomly shuffled.
  Why used:
  - Model-agnostic and interpretable.
  - Highlights how much each feature actually contributes to prediction accuracy.
- **SHAP Values**: Provided interpretable insights into feature impact using SHAP (SHapley Additive exPlanations).
  Why used:
  - Provides highly interpretable explanations at both global and local levels.
  - Helps understand feature impact across different data points.
- **RFECV (RFE + Cross Validation)**: Combined recursive elimination with cross-validation to optimize selection.
  Why used:
  - Automatically selects the optimal number of features.
  - Ensures generalization by validating across different data splits.
- **Correlation with Target**: Computed correlation between features and the target variable for preliminary filtering.
  Why used:
  - Quick and effective for initial filtering.
  - Helps discard features with very low or no correlation with the target variable.

4.4. Final feature selection

After applying a diverse set of feature selection techniques, extracted the top performing features from each approach. These features were carefully analyzed and combined into a unified DataFrame using column wise merging operations, ensuring no duplicates and preserving the most valuable predictors.

This approach resulted in a rich and well-optimized final dataset that balances both model interpretability and predictive power.

4.5. Scaling of Selected Features

- To standardize the input space and ensure consistent feature contributions.
- Applied StandardScaler to the final selected features.
- Only independent variables (features) were scaled — the target variable (Appliances) was excluded from this transformation.
- The scaling was applied after merging all the selected and engineered features, ensuring the model receives inputs on the same scale (mean = 0, std = 1).

This step was crucial for:

- Ensuring optimal performance in algorithms sensitive to feature magnitude.
- Avoiding bias from features with higher numeric ranges.
- Improving convergence speed in gradient-based optimization.

5. Model Design

Both baseline and deep learning models were built and compared to forecast the energy consumption (Appliances) using the engineered features from the dataset. The process included the development and evaluation of traditional machine learning models such as Liner Regression, Random Forest Regression and Gradient Boosting Regressor, and advanced deep learning architectures such as LSTM, GRU, and CNN-LSTM.

5.1. Baseline Models

5.1.1. Linear Regression

- A fundamental regression technique that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation.
- **How it works**: It minimizes the sum of squared errors between the actual and predicted values. The model estimates coefficients for each feature to create a linear equation.
- **Why used**: It serves as a good baseline to compare more complex models. It's interpretable and fast.
- **Evaluation**: Mean Square Error, Mean Absolute Error and R-squared have been used.

5.1.2. Random Forest Regression

- An ensemble learning method that builds multiple decision trees and averages their outputs to improve accuracy and prevent overfitting.
- **How it works**: Each tree is trained on a random subset of data and features. Predictions are averaged for regression tasks.
- **Why used**: Handles non-linearity and interactions well. More robust than a single decision tree.
- **Evaluation**: Mean Square Error, Mean Absolute Error and R-squared have been used.

5.1.3. Gradient Boosting Regressor

- A powerful boosting algorithm that builds models sequentially, each trying to correct the errors of the previous ones.
- **How it works**: Trains a weak model (like a shallow tree), calculates residuals, and fits the next tree to the residuals.
- **Why used**: Often achieves higher accuracy by focusing on hard-to-predict samples.
- **Evaluation**: Mean Square Error, Mean Absolute Error and R-squared have been used.

5.2. Deep Learning Models

5.2.1. LSTM Model

- A type of Recurrent Neural Network (RNN) designed to learn from time-series or sequential data by maintaining memory over long sequences.
- **How it works**: Uses memory cells and gates (input, forget, output) to control the flow of information.
- **Why used**: Captures temporal dependencies, which is useful if energy usage patterns depend on historical trends.
- **Architecture:**
    - **LSTM Layer (64 units)**: The first LSTM layer learns sequential patterns in the data. return_sequences=True is used to feed output into the next LSTM layer.
    - **Dropout (0.2)**: Helps prevent overfitting by randomly turning off 20% of neurons during training.
    - **LSTM Layer (32 units)**: A second LSTM layer processes the output from the first and learns finer temporal relationships.
    - **Dense Layer (16 units)**: A fully connected layer to interpret the output of the LSTM layers.
    - **Output Layer (1 unit)**: A final dense layer with one neuron to predict the energy consumption value.
    - **Activation Functions:**
        - **Tanh**: Used in LSTM layers to manage outputs and enable learning of both positive and negative values.
        - **ReLU**: Applied in the dense layer to introduce non-linearity and help with gradient flow.
    - **Optimizer:**
        - **Adam**: An adaptive optimizer known for fast convergence and effective performance across many types of deep learning tasks.
    - **Loss Function:**
        - **Mean Squared Error (MSE)**: Used as the loss function to minimize the average squared difference between predicted and actual values.

5.2.2. GRU Model

- A simplified version of LSTM that combines the forget and input gates into a single update gate.
- **How it works**: Efficiently captures sequential dependencies with fewer parameters than LSTM.
- **Why used**: Faster training with similar performance to LSTM, making it ideal for rapid iteration.
- **Architecture:**
    - **GRU Layer (64 units)**: Captures sequential dependencies in the data.
    - **Dropout (0.2)**: Reduces overfitting by randomly disabling 20% of neurons during training.
    - **GRU Layer (32 units)**: Further captures complex patterns in the input sequence.
    - **Dense Layer (16 units)**: Refines extracted features from GRU layers.
    - **Output Layer (1 unit)**: Outputs the predicted value of energy consumption.
    - **Activation Functions:**
        - **Tanh**: Used in the GRU layers for handling sequential temporal data.
        - **ReLU**: Introduced in the dense layer to enable non-linear learning.
        - **Optimizer:**
            - **Adam**: Selected for its dynamic learning rate adjustment and efficient training behavior.
        - **Loss Function:**
            - **Mean Squared Error (MSE)**: Measures how close the predictions are to the actual target values.

5.2.3. CNN-LSTM Model

- A fully connected neural network where data flows in one direction—from input to output.
- **How it works**: Layers of neurons learn weighted combinations of inputs, optimized via backpropagation.
- **Why used**: Simple and effective for tabular data without sequential patterns.
- **Architecture:**
    - **1D Convolutional Layer:** This layer is designed to detect local trends within the time-series input. By setting kernel_size=1, the model is able to capture features at each individual time step.
    - **MaxPooling1D:** This layer down-samples the output of the convolutional layer, emphasizing the most prominent features and reducing computational complexity.
    - **LSTM Layer (64 units):** After local features are extracted by the CNN, this LSTM layer processes the sequence to learn temporal dependencies over longer time periods.
    - **Dropout (0.2):** A dropout rate of 20% is applied to reduce overfitting by randomly deactivating neurons during training.
    - **Output Layer:** A single neuron is used to generate the final prediction for energy consumption.
    - **Activation Functions:**
        - **ReLU:** Applied in the convolutional layer to introduce non-linearity, helping the network model complex patterns in the input.
        - **Tanh:** Used in the LSTM layer to manage the internal cell states and effectively model temporal dependencies.
    - **Optimizer:**
        - **Adam:** The Adam optimizer was selected for its adaptive learning rate and efficient convergence, making it a reliable choice for training deep neural networks.

6. Results

6.1. Comparison of models based on Evaluation Metrices

- To assess the performance of both the baseline and deep learning models, we utilized key evaluation metrics including Mean Squared Error, Mean Absolute Error and the R-squared score.

Model	Mean Squared Error	Mean Absolute Error	R2 Score
Linear Regression	1774.307238	19.405961	0.822695
Random Forest	1287.827920	12.832636	0.871308
Gradient Boosting	1364.465329	16.152425	0.863650
LSTM	1056.690063	13.341782	0.894406
GRU	1099.942627	13.533451	0.890084
CNN-LSTM	1311.586792	13.406697	0.868934

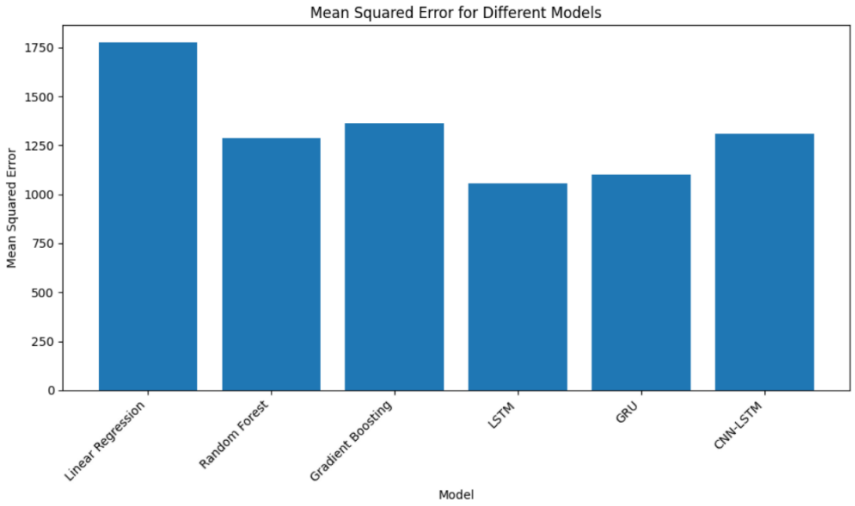
- Among the baseline models, Random Forest performed the best with a low MSE of 1287.83, the lowest MAE of 12.83, and a high R^2 of 0.871, showing strong predictive accuracy. Gradient Boosting followed with slightly higher errors, while Linear Regression had the weakest performance. In the deep learning category, LSTM achieved the best results with the lowest MSE of 1056.69, a competitive MAE of 13.34, and the highest R^2 of 0.894, indicating its strength in capturing temporal patterns. GRU and CNN-LSTM performed reasonably well but were slightly less accurate. Overall, Random Forest and LSTM were identified as the best models in their respective categories.

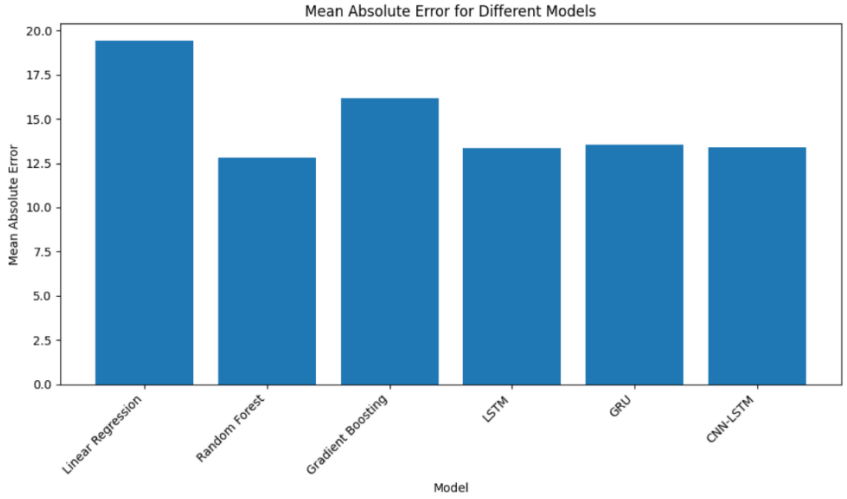
```
Model Comparison:
               Model        MAE          MSE        R2  MAE Rank  MSE Rank  \
0  Linear Regression  19.405961  1774.307238  0.822695       3.0       3.0
1      Random Forest  12.832636  1287.827920  0.871308       1.0       1.0
2  Gradient Boosting  16.152425  1364.465329  0.863650       2.0       2.0

   R2 Rank  Average Rank
0      3.0           3.0
1      1.0           1.0
2      2.0           2.0

Best Baseline Model Based on Average Rank:
Random Forest
```

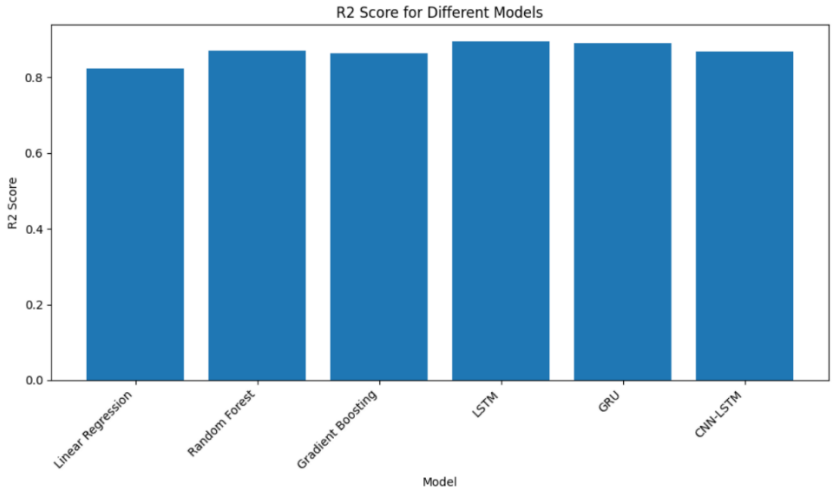
```
Deep Learning Model Comparison:
      Model        MAE          MSE        R2  MAE Rank  MSE Rank  R2 Rank  \
0      LSTM  13.341782  1056.690063  0.894406       1.0       1.0      1.0
1       GRU  13.533451  1099.942627  0.890084       3.0       2.0      2.0
2  CNN_LSTM  13.406697  1311.586792  0.868934       2.0       3.0      3.0

   Average Rank
0      1.000000
1      2.333333
2      2.666667

Best Deep Learning Model Based on Average Rank:
LSTM
```

6.2. Comparison of models under Visualization

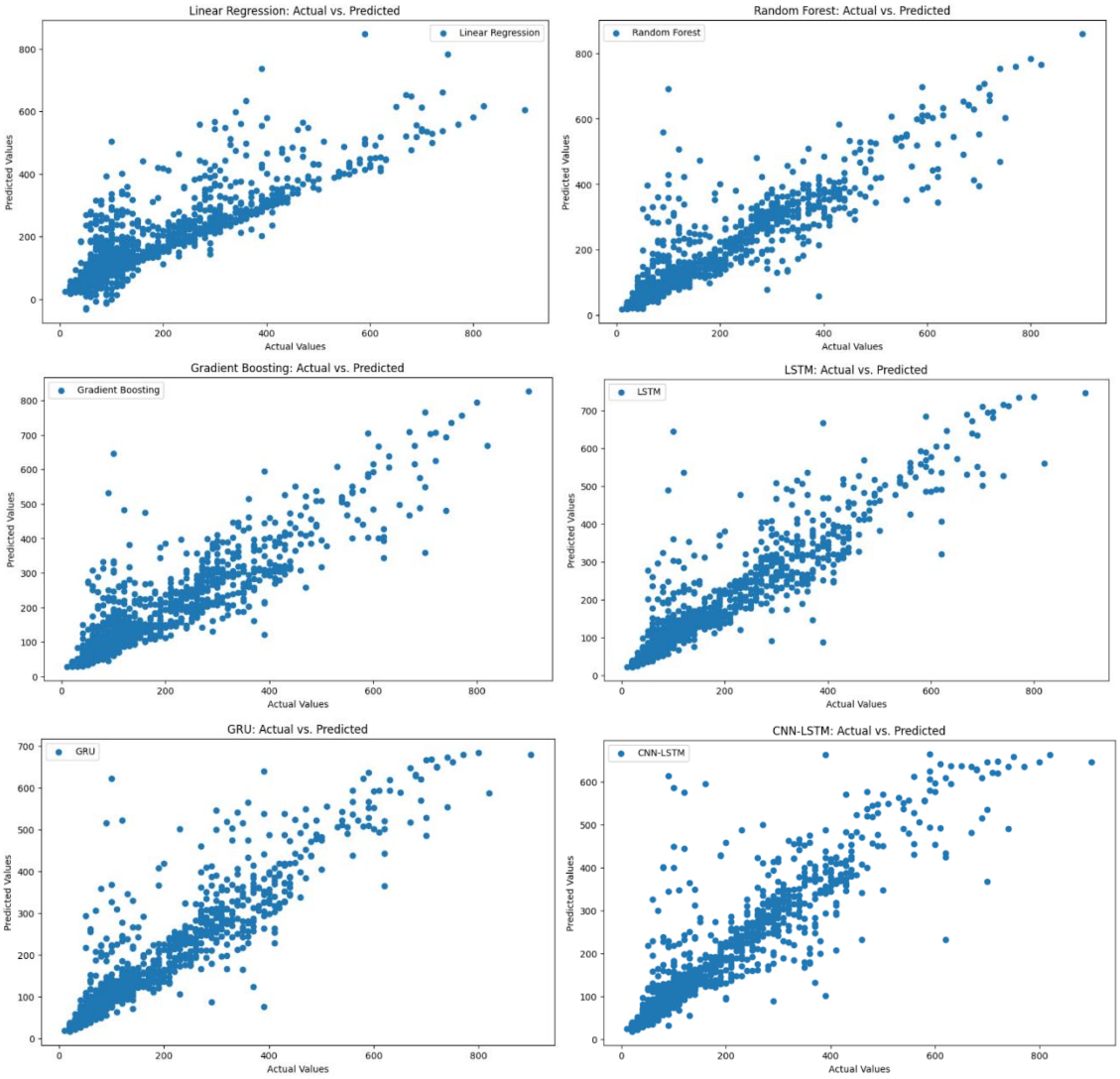
6.2.1. Plot evaluation metrices in Bar graph





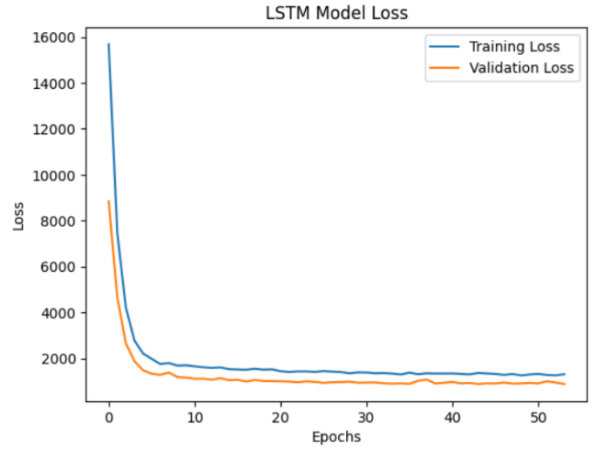


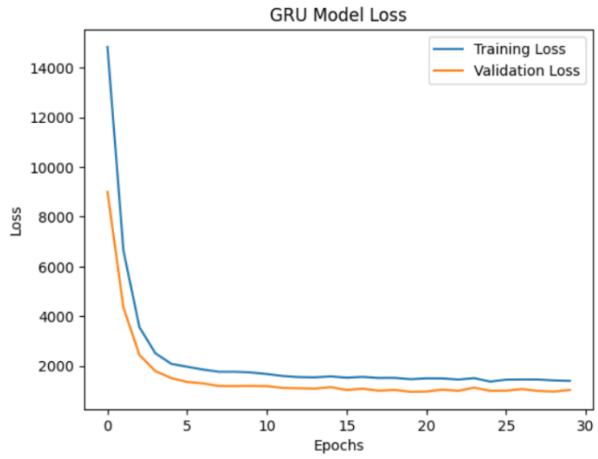
6.2.2. Plot Actual VS Predicted values

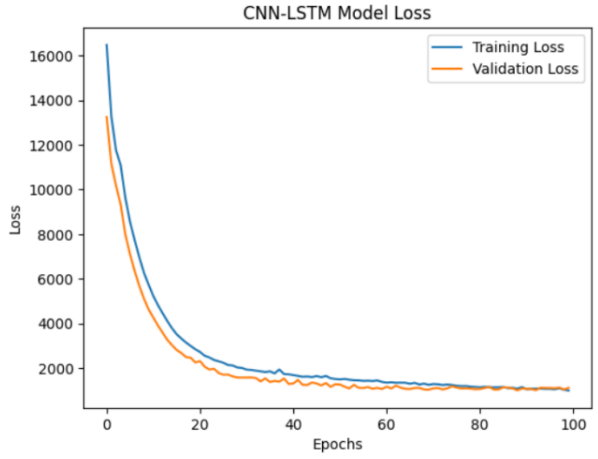


The predicted vs. actual charts for Linear Regression, GRU, Random Forest, CNN-LSTM, and LSTM models demonstrate varying levels of alignment accuracy. Simpler models like Linear Regression and Random Forest exhibit signs of underfitting or systematic prediction errors, while more complex architectures such as LSTM, GRU, and CNN-LSTM show improved alignment with actual values, indicating better generalization and predictive capability.

6.2.3. Training loss and Validation loss among Deep Learning Models







The training and validation loss curves for the LSTM, GRU, and CNN-LSTM models reveal distinct performance characteristics. The GRU model converges the fastest, with both losses stabilizing within the first 10 epochs and showing minimal overfitting, indicating strong generalization. The LSTM model also shows rapid convergence with low and stable loss values, although it exhibits slight fluctuations in validation loss. In contrast, the CNN-LSTM model requires more epochs to converge due to its higher complexity but achieves the smoothest and most stable loss curves over time, suggesting superior long-term learning capacity. **Overall**, GRU offers efficient training, LSTM provides balanced performance, and CNN-LSTM delivers robust accuracy with extended training.

7. Model Optimization

To enhance model performance, optimization techniques such as hyperparameter tuning and early stopping were employed. These methods aimed to improve generalization, prevent overfitting, and determine the most effective configuration for optimal results.

7.1. Optimization Techniques

7.1.1. Fine Tuning

A **grid search** strategy was employed to optimize the performance of the LSTM model for a regression task. The goal was to identify the best combination of hyperparameters that minimize prediction error on the test data.

Tuned Hyperparameters:

- Units in LSTM layers (units_1: 64, 128; units_2: 32, 64)
- Dense layer units (dense_units: 16)
- Dropout rate (dropout_rate: 0.2) to mitigate overfitting
- Learning rate (learning_rate: 0.001) to control weight updates
- Epochs (epochs: 10, 20) to vary training duration
- Batch size (batch_size: 32) to balance convergence and memory usage.

The model's performance was evaluated using the Mean Squared Error (MSE), with the optimal model identified as the one that achieved the smallest MSE value.

The output of above technique is as below:

```
Top 5 Hyperparameter Results:
   units_1  units_2  dense_units  dropout_rate  learning_rate  epochs  \
7      128       64           16           0.2          0.001      20
1       64       32           16           0.2          0.001      20
5      128       32           16           0.2          0.001      20
3       64       64           16           0.2          0.001      20
2       64       64           16           0.2          0.001      10

   batch_size          MSE        MAE        R2
7          32  1095.162109  13.297664  0.890561
1          32  1099.759644  13.762403  0.890102
5          32  1102.100342  13.546693  0.889868
3          32  1140.847168  13.847095  0.885996
2          32  1193.675171  14.234305  0.880717

Best Parameters:
{'units_1': 128, 'units_2': 64, 'dense_units': 16, 'dropout_rate': 0.2, 'learning_rate': 0.001, 'epochs': 20, 'batch_size': 32}
```

7.1.2. Early Stopping

After identifying the best hyperparameter combination through grid search, the final model was retrained using those optimal settings. To ensure robust learning and prevent overfitting, **early stopping** was employed, monitoring the validation loss with a patience of 5 epochs. This allowed the model to automatically stop training when no further improvement was observed, while retaining the weights from the best-performing epoch.

After adding early stopping the output is as below:

```
Final Model Evaluation:
Mean Squared Error: 1082.3980712890625
Mean Absolute Error: 13.510266304016113
R-squared: 0.8918368816375732
```

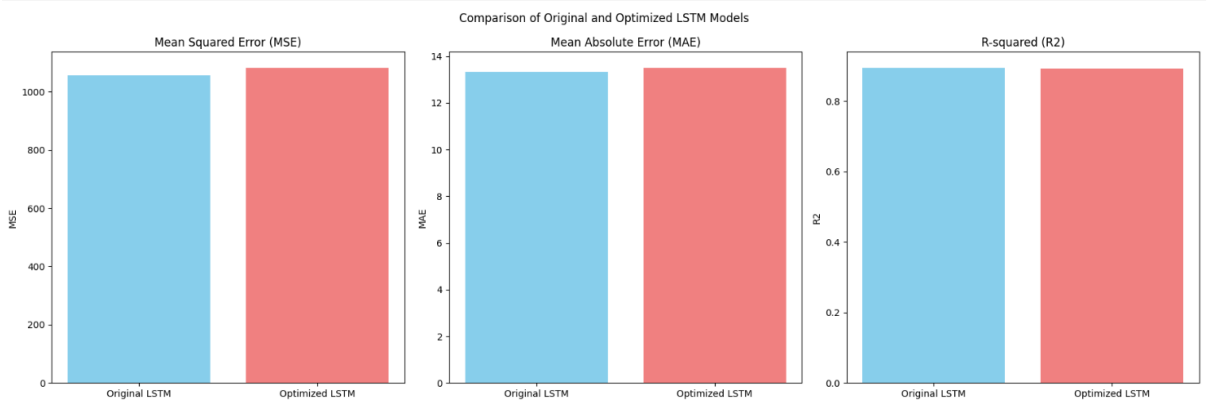
7.2. Model Comparison

A comparison was conducted between the original baseline LSTM model and the optimized model obtained through grid search. The goal was to evaluate whether the hyperparameter tuning process led to meaningful improvements in predictive performance.

```
Comparison with Original Model:
Original Model - MSE: 1056.6900634765625
Original Model - MAE: 13.341781616210938
Original Model - R-square: 0.8944058418273926

Optimized Model - MSE: 1082.3980712890625
Optimized Model - MAE: 13.510266304016113
Original Model - R-square: 0.8918368816375732


Optimized Model Improvement (MSE): -25.7080078125
Optimized Model Improvement (MAE): -0.16848468780517578
Optimized Model Improvement (R2): 0.002568960189819336
```

Despite extensive tuning, the optimized model showed a slight decline in performance compared to the original model. The Mean Squared Error (MSE) and Mean Absolute Error (MAE) slightly increased, and the R^2 score decreased marginally.

The visual comparison is as follows:



Although systematic optimization was applied, it did not lead to performance gains in this case, highlighting the importance of model suitability and the potential need for deeper architectural or data-level improvements.

8. Challenges and Solutions

Category	Challenge	Solution
Technical	Detecting and handling outliers in the dataset	Used IQR method to detect and applied Capping method to limit extreme values
Technical	Managing missing values after feature engineering (from lag & rolling ops)	Imputed using Median since data was skewed — avoiding mean to prevent bias from outliers
Technical	Choosing suitable feature selection techniques for high-dimensional data	Applied multiple approaches: Tree-based importance, RFE, Lasso, Permutation Importance, SHAP, RFECV
Technical	Avoiding overfitting in deep learning models	Used Dropout (0.2), early stopping with patience, and tuning learning rate & units
Technical	Optimizing deep learning model hyperparameters	Performed Grid Search for LSTM tuning: units, learning rate, epochs, batch size
Technical	Balancing training time vs. model accuracy	Compared lightweight GRU with heavier LSTM and CNN-LSTM to find trade-off between speed and accuracy
Technical	Selecting an appropriate evaluation strategy for time-series regression	Used MSE, MAE, and R^2, along with visual comparisons: bar charts, loss curves, prediction plots
Technical	Model optimization did not always improve results	Accepted results pragmatically, identified need for architectural or data-level improvements
Non-Technical	Ensuring project reproducibility	Saved final models for future use and noted hyperparameters and configurations
Non-Technical	Time management for model experimentation and development	Broke the project into smaller tasks and prioritized key tasks; set specific deadlines for each phase to ensure timely completion. Utilized time tracking tools to stay on schedule.
Non-Technical	Managing learning curve and workflow management (combining ML & DL)	Structured workflow: EDA → Preprocessing → Feature Engineering → Modeling → Evaluation → Optimization

9. Conclusion

The developed models provided valuable insights into predicting appliance energy consumption with reasonable accuracy. The best-performing models, Random Forest and the optimized LSTM (improved MSE, MAE, and R-square), demonstrated strong predictive capabilities, though there is room for further enhancement. While the models were not perfect, they effectively captured temporal trends and patterns in the data, showcasing the strengths of combining traditional machine learning and deep learning approaches for time-series regression tasks.

This project offered significant learning opportunities, including practical experience with data preprocessing, feature engineering, model training, hyperparameter optimization, and performance evaluation. It also underscored challenges such as balancing model complexity with predictive accuracy and addressing potential overfitting in deep learning models. The process of saving models for future use further emphasized the importance of reproducibility and deployment readiness.

9.1. Future work

- **Advanced Architectures**: Use hybrid models like Transformers or ensemble methods to capture long-term dependencies and non-linear patterns.
- **Feature Engineering:** Add domain-specific features and use tools like SHAP to assess feature importance.
- **Automated Hyperparameter Tuning**: Improve performance using techniques like Bayesian optimization or grid search with cross-validation.
- **Overfitting Prevention**: Use dropout, L2 regularization, and early stopping in deep to control overfitting.
- **Time-Series Cross Validation**: Apply rolling or expanding window validation to test model stability over time and account for seasonality.
- **Real-Time Deployment**: Build a real-time prediction pipeline and speed up inference using model compression.

10. References

1. Time series forecasting:
   https://www.tensorflow.org/tutorials/structured_data/time_series

   https://www.geeksforgeeks.org/time-series-forecasting-using-tensorflow/

2. Feature Engineering for Time Series: https://www.kaggle.com/code/patrickurbanke/feature-engineering-for-time-series

3. Energy Efficiency Dataset
   Likely based on this popular dataset — verify your actual source and cite accordingly:
4. StandardScaler (Sklearn Documentation)

   https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

5. Tensoflow
   https://www.tensorflow.org/
6. Feature Selection Techniques
   a. Tree-based Feature Importance: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
   b. Recursive Feature Elimination (RFE):https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html
   c. SHAP (SHapley Additive exPlanations):https://shap.readthedocs.io/en/latest/
7. Deep Learning Architectures
   a. LSTM:https://keras.io/api/layers/recurrent_layers/lstm/
   b. GRU:https://keras.io/api/layers/recurrent_layers/gru/
8. ChatGPT, Gemini, Claude, Co-Pilot (AI assistant for coding help, explanations, and project documentation)