1. INTRODUCTION

1.1 Introduction to Forecasting Energy Consumption

In recent decades, studying energy consumption issues has been a popular academic area. Energy issues are critical to society's security and well-being [1]. Energy is one of the most significant resources for industrial output, according to economic theories, and anticipating energy consumption is a crucial phase for industry and energy sector macro-planning [2].

Energy supply-demand planning on a long-term basis must meet the needs of countries' long-term development. Accurate predictions can assist decision-makers in determining the amount and trend of future energy consumption to properly schedule and plan supply system operations.

Forecasting the consumption load is a key aspect of power distribution systems economic and safe operations planning. According to the forecasting horizon, there are primarily three types of energy consumption studies in the literature [3]. The majority of long-term forecasting (5–20 years) is used for resource management and development spending. Short-term forecasting (an hour to a week) is generally used for scheduling and analysis of the distribution network, whereas mid-term forecasting (a month to 5 years) is mostly used for planning power production resources and rates [2][4]. The focus of the paper is on short-term forecasting. Because energy demand is affected by a variety of factors such as time, climate, socioeconomic, and demographic factors, precisely anticipating consumption is both crucial and difficult. So, it is important to predict the energy consumed beforehand to make the generation more feasible and also to save energy it plays a vital role.

1.2 Introduction to Machine Learning for prediction

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves. Here, Machine learning approaches are mainly used in the extraction of different patterns from various fields from a lower level to a higher level. In this case, labeled data is present so it is

supervised machine learning and as the forecast is continuous it is a regression model. This project uses regression models such as linear regression, lasso regression, ridge regression, random forest, K-Nearest Neighbour Regressor, Gradient Boosting, XGBoost, and Extra tree regressor [18].

1.3 Data Summary

Name	Description	Unit
T1	Kitchen Temperature	Celsius
T2	Living Room Temperature	Celsius
T3	Laundry Room Temperature	Celsius
T4	Office Temperature	Celsius
T5	Bathroom Temperature	Celsius
T6	Temperature outside building	Celsius
T7	Ironing Room Temperature	Celsius
T8	Teenager Room Temperature	Celsius
T9	Parents Room Temperature	Celsius
T_out	Outside Temp (Weather station)	Celsius
T_dew point	Dew point Temp (Weather station)	Celsius

Name	Description	Unit
Data	Time stamp of sensor data	Date time
RH_1	Kitchen Humidity	Percentage
RH_2	Living Room Humidity	Percentage
RH_3	Laundry Room Humidity	Percentage
RH_4	Office Humidity	Percentage
RH_5	Bathroom Humidity	Percentage
RH_6	Humidity Outside (North)	Percentage
RH_7	Ironing Room Humidity	Percentage
RH_8	Teenager Room Humidity	Percentage
RH_9	Parents Room Humidity	Percentage
RH_out	Outside Humidity (Weather station)	Percentage
Pressure	Outside Pressure (Weather station)	mmHg
Wind Speed	Outside Wind Speed (Weather Station)	m/s
Visibility	Visibility (Weather Station)	Km

Name	Description	Unit
Rv1	Random Variable 1	-
Rv2	Random Variable 2	-
Lights	Energy used by lights	Wh
	TARGET VARIABLE	
Appliances	Total Energy Used by appliances	Wh

Table1. Data description and units of features

1.4 Data Scaling

Temperature	-6 to 30 c
Humidity	1 to 100%
Wind Speed	0 to 14m/s
Visibility	1 to 66km
Pressure	729 to 772 mmHg
Appliance Energy Usage	10 to 1080Wh

Table2. Data Scaling of features

The project provides answers to inquiries about how energy consumption varies hourly, day by day, and month by month. The second section would concentrate on feature engineering and selection. The final section would involve modeling and comparing many models to choose the best one. The dataset is good enough and only requires scaling for data processing.

Prediction:

"Prediction" refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome, such as whether or not a customer will churn in 30 days. The algorithm will generate probable values for an unknown variable for each record in the new data, allowing the model builder to identify what that value will most likely be.

Analysis:

Data Analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Electrical Energy:

The total work done in an electric circuit is called electrical energy i.e.

Electrical energy = Electrical power × Time

$$V\ I\ (t) = I^2 * R = \frac{V^2}{R}$$

Units: Electrical energy is measured in kilowatt-hours (kWh).

One kilowatt-hour (kWh) of electrical energy is expended in a circuit if 1 kW (1000 watts) of power is supplied for 1 hour.

Electric Power:

The rate at which work is done in an electric circuit is called its electric power i.e.

$$\text{Electric power} = \frac{\text{Work done in electric circuit}}{\text{Time.}}$$

Unit of electric power. The basic unit of electric power is joules/sec or watt.

The power consumed in a circuit is 1 watt if a p.d of 1 V causes 1 A current to flow through the circuit.

Power in watts = Voltage in volts × Current in amperes

The bigger units of electric power are kilowatts (kW) and megawatts (MW).

1 kW = 1000 watts;

1 MW = 10^6watts or 10^3KW

Energy Consumption:

Energy consumption is the amount of energy or power used

The power consumption of small devices is usually measured in Watts, and the power consumption of larger devices is measured in **kilowatts (kW)**, or 1,000 Watts

Sensor:

A sensor is a device that monitors and responds to environment input. Light, heat, motion, pressure, or any of a variety of other environmental phenomena could constitute the specific input. The output is usually a signal that is transformed to a human readable display at the sensor location or sent over a network for reading or additional processing.

ZigBee:

Zigbee is a standards-based wireless technology developed to enable low-cost, low-power wireless machine-to-machine (M2M) and internet of things (IoT) networks. Zigbee is for low-data rate, low-power applications and is an open standard. These devices are used to store the data and used for

Home Automation: ZigBee technology proves to be the most reliable technology in realizing home automation. Different applications like controlling and monitoring energy consumption, water management, light control etc. have been made easier through automation using ZigBee technology.

Smart Metering: Zigbee remote operations in smart metering include energy consumption response, pricing support, security over power theft, etc.

Smart Grid monitoring: Zigbee operations in this smart grid involve remote temperature monitoring, fault locating, reactive power management.

1.5 Objectives:

- To analyze the patterns of energy consumption.
- To predict energy consumption with the help of previously available data
- To provide necessary information to the power plant for generation.
- Using machine learning concepts to automate the predictions.

2. EXPLORATORY DATA ANALYSIS

2.1 Energy consumption with temperature & humidity

The average temperature outside is about 7.5 degrees. The temperature fluctuates from -6 to 28 degrees. Inside the building, the average temperature has been approximately 20 degrees in all rooms. It has a temperature range of 14 to 30 degrees. Warming appliances have been put in place to keep the insides of the building warm, according to this. As a result, there would be a link between indoor temperature and energy use. Outside the building, the average humidity has been higher than within. The average humidity in the children's and parent's rooms is significantly higher, indicating that the residents of this building spend most of their time in these areas. Because the humidity levels are so high, some appliances, such as dehumidifiers, may be necessary. As a result, there would be a link between humidity and energy use.

2.2 Energy consumption with time

There are two instances of peak hour activity. The first is at 11 a.m., and the second is at 6 p.m as shown in Fig1. The peak at 11 a.m. is shallow and low, whereas the peak at 6 p.m. is taller and sharper. The energy usage of appliances is roughly 50 Wh during the sleeping hours (10 PM - 6 AM). After 6 a.m., energy use gradually increases until 11 a.m. (probably due to morning chores). At roughly 3 p.m., it progressively drops to around 100 Wh. After that, until 6 p.m., energy use skyrockets (probably due to requirement lights in rooms). However, as night falls and everybody in the house retires to bed around 10 p.m., appliance energy consumption drops to 50 Wh. In Belgium, it appears that the month has no impact on energy use. However, due to the warm weather, may month's energy use may be lower.

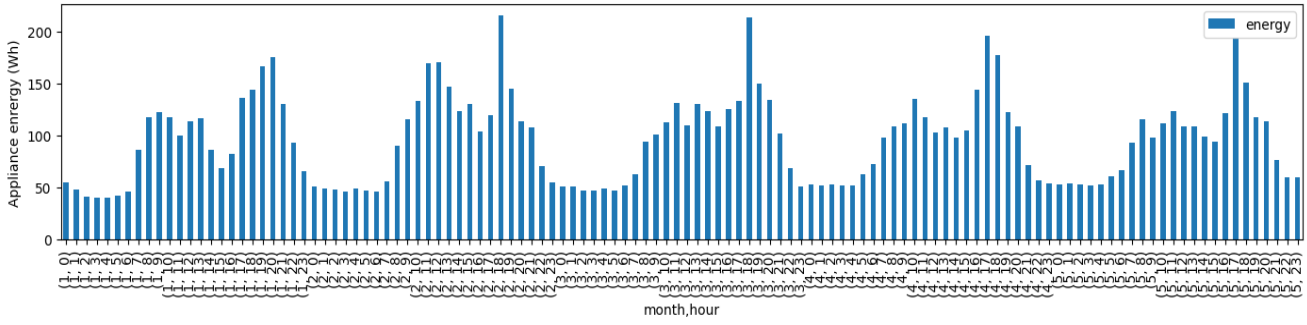

Fig1. Energy trend over months for every hour

From Fig2 it is observed that weekend energy use is higher than weekday energy usage. This could be because weekends are holidays, and most of the building residents stay at home and use appliances.

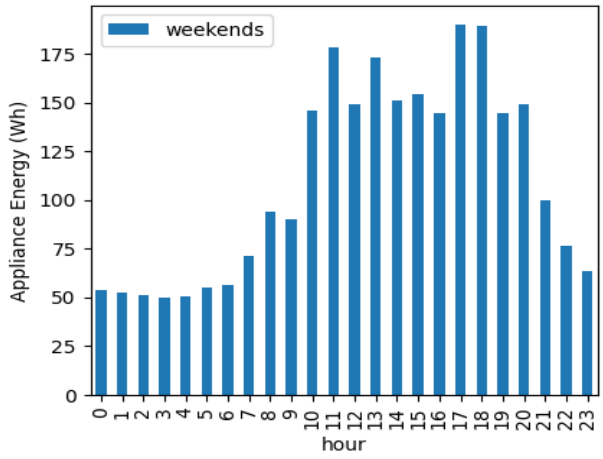

Fig2(i). Hourly energy on weekends

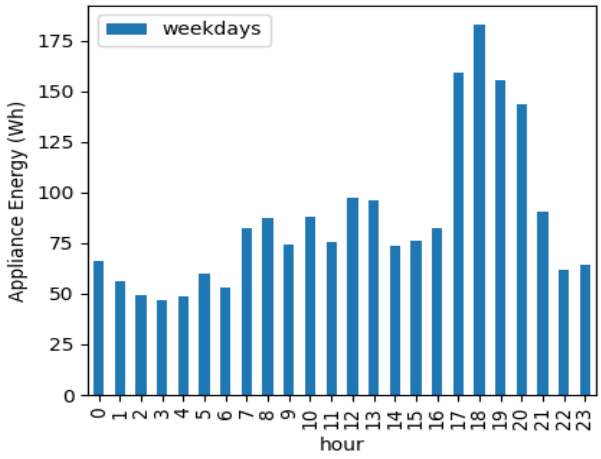

Fig2(ii). Hourly energy on weekdays

2.3 Distribution of Energy consumed

The data does not follow a normal distribution. Because the data is skewed as observed in Fig3(i). So, it necessitates the use of transformation. Here log10 transformation is used. Because many models have the underlying assumption of normal distribution, this type of transformation has a positive impact on modeling. After applying transformation, the distribution looks as in Fig3(ii).

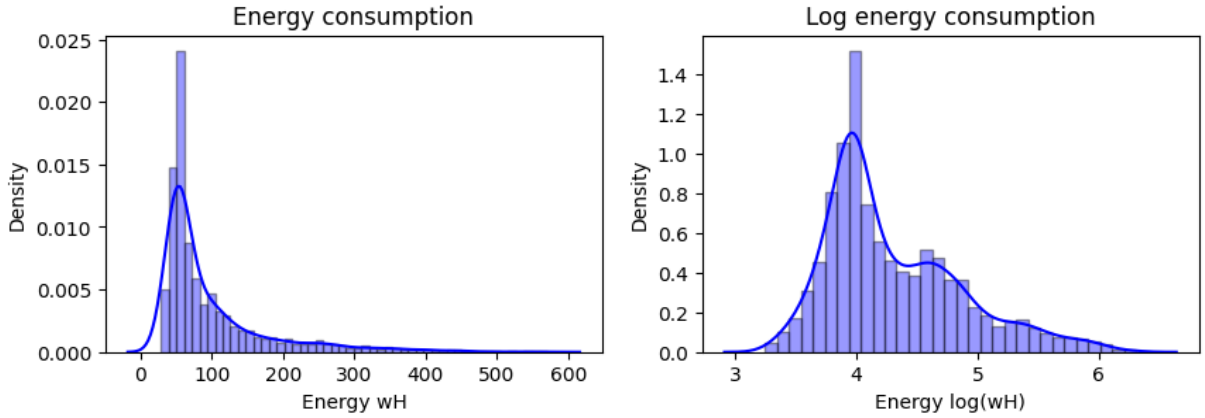

Fig3(i). Energy distribution before transformation. **Fig3(ii).**Energy distribution after transformation.

2.4 Correlation with Heat Map

The Pearson correlation is used to determine how inter-dependent the features are. It has been discovered that characteristics have a range of -0.09 to 0.02 with energy. The temp_kitchen has correlation of 0.06 with energy and temp_living has correlation of 0.12 with energy. In the similar way others are having correlations among the features which is indicated in the Fig4. below. It is seen that all the temperature

related features are having more correlation among them and humidity related features are having the similar kind of correlation among them. The correlation observed is lower than expected. This necessitates a concentration on feature engineering to obtain a greater number of linked features with the dependent feature for modeling purposes. Some having high correlation between them as 1 are the correlation between same features. High correlation is indicated by light colors and low correlation by dark colors.

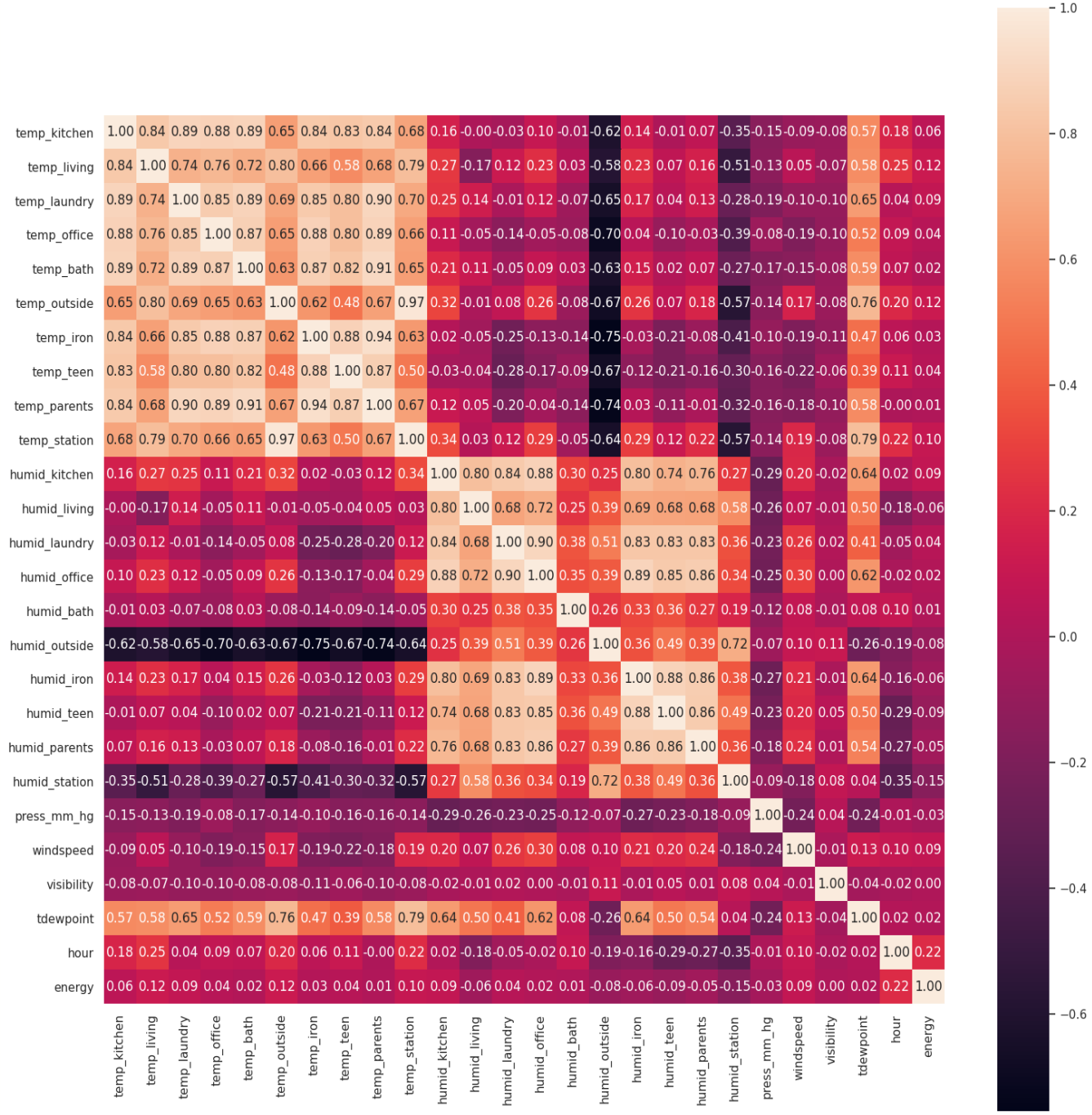

Fig4. Heat map of the given features

3. FEATURE ENGINEERING

The second section of the project is about feature engineering. Different significant features are retrieved from the features provided in this section. EDA helps to know relevant features that can be extracted.

The intention of feature engineering is to achieve two primary goals:

1. Preparing an input dataset that is compatible with and best fits the machine learning algorithm.
2. Improving the performance of machine learning models

Feature engineering involves leveraging data mining techniques to extract features from raw data along with the use of domain knowledge. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning. Features are also referred to as 'variables' or 'attributes' as they affect the output of a process. Feature engineering involves several processes. Feature selection, construction, transformation, and extraction are some key aspects of feature engineering.

Feature selection involves choosing a set of features from a large collection. Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible. Feature selection also improves the quality of the output obtained from algorithms.

3.1 Outlier removal

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. Examination of the data for unusual observations that are far removed from the mass of data.

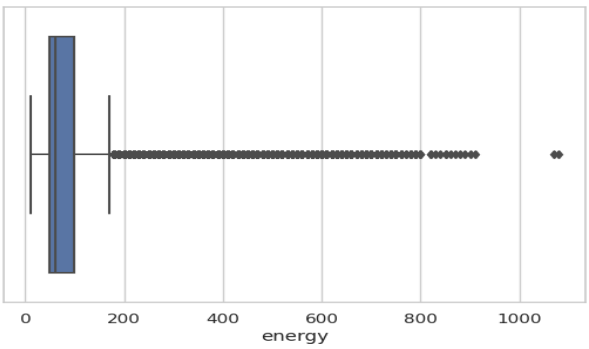

Fig5. Box plot of energy feature

These points are often referred to as outliers. These data points are out of the quartile ranges as in Fig5. These points unnecessarily increase the tail of the distribution. As a result, the model may become biased because of high values. As these are high values in models as to decrease the error it might tend towards those values. So, it is important to remove such outliers as these don't encourage the model to generalize the situation for prediction.

From the box plot as shown in Fig5, it can be observed that after 200Wh there are outliers. But it is important to understand that there might be a sudden increase in energy at certain times and should be accepted. So here only the top 0.5% which are unusual are removed. The number of the 0.5% top values of the load is 39 and they have a power load higher than 740 Wh.

3.2 Feature Extraction from energy consumed

From Fig3a. it can be seen that the data is skewed. Many machine learning models have an underlying assumption of normally distributed data. So, to help the model to perform better from energy a new feature is extracted. So, a new feature named log energy is added to the energy feature by performing the log10 transformation to the energy feature. This decreases the skewness of the data as visualized in Fig3b.

3.3 Feature extraction from temperature and Humidity

Temperature and humidity features are highly reliant on their own features, thus by averaging them, a new feature is created from one of the temperature features and one from the humidity features.

Temperature and humidity are also inversely connected, according to the data analysis. This reason can be used to multiply each of the temperature and humidity features of the same room, yielding an additional 9 features.

3.4 Features from timestamps

Apart from features like temperature and humidity levels in rooms and outside, there is another crucial column date, which contains timestamps for each of the sensor data samples. The hour data for each sample is derived from the timestamps, and the goal is to see if there is a pattern of energy usage over a day. For 4.5 months, the figure below depicts the average energy usage of appliances at a given hour of the day.

Hours and their relationship with energy consumption were explored in the analysis section. The 24-hour day can be divided into three classes based on energy consumption patterns: Class 1: 10 PM - 6 AM, Class 2: 6 AM - 3 PM, and Class 3: 3 PM - 10 PM so that the model can learn from multi-class classifiers[9] as shown in Fig6**.**

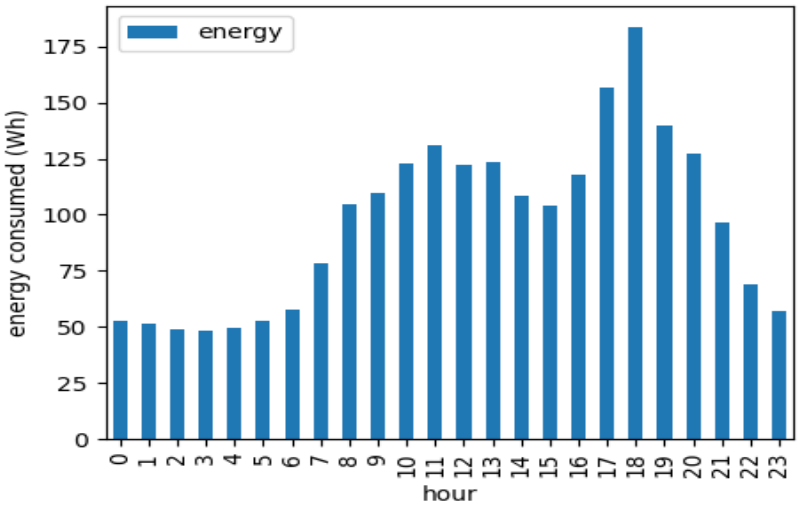

Fig6. Average hourly energy consumption trend

3.5 Feature extraction using PCA

Principal Component Analysis (PCA) which is a dimension-reduction and feature extraction tool, aims at reducing a large set of variables to a small set that still contains most of the information in the large set. So, this reduced set is much easier to analyze and interpret. In this work Principal Component analysis (PCA) is one of the most popular techniques in artificial intelligence for dimensionality and reduction which helps in increasing correlations between features without any prior knowledge. Understanding the relationship between features presents several challenges, but PCA mitigates them. PCA is important because it does not involve training or labeling, but automatically finds relationships.

Step1: Take the whole data set with dimension 'd'.

Step2: Compute the mean of every dimension of the whole data set.

Step3: Compute the covariance matrix of the whole data set.

$$\text{Cov(X,Y)} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})$$

Step4: Compute Eigen vectors and corresponding eigen values for the covariance matrix A.

$Av = \lambda v$, Where A is square matrix, λ *is eigen values*

Step5: Sort the eigen vectors by decreasing eigen values and choose K eigen vectors with Largest eigen values to form dXK dimensional matrix W.

Step6: Transform the samples onto the new substances.

$$y = w^T \times x$$

This step involves finding the PCs by using the eigenvalues and the eigenvectors with the largest eigenvalue will be called the first PC i.e PC1, and the eigenvalues with the least eigenvalue will be called the least PC i.e., PC8. We will be arranging the eigenvalues in the descending order $e1 \geq e2 \geq .. \geq e8$. Based on the eigenvalues we will find the corresponding eigenvectors. By doing this we will be getting PCs.

Here firstly PCA on temperatures is performed and with the help of a heat map PCs are selected based on variability explained and correlation. Here PC1 and PC4 explain high variability and high correlation respectively so these two features are taken forward. After this PCA on humidity features is performed. For humidity features also PC1 and PC4 explain high variability and high correlation respectively so these two features are taken forward.

So, an additional 16 features are retrieved using feature engineering and observed characteristics are now more dependent on energy use, with correlations ranging from -0.21 to 0.55 as shown in Fig20. These additional features are so useful in modelling. So, with the help of feature engineering, we managed to increase the correlation. This would yield better results than the baseline model built. The highly related features make the model to learn better and can generalize better to make good predictions.

4. MODELING & EVALUATION

Now having good features, the next stage is to model using different regression models and compare them to find the optimum regression model for energy prediction. Because this is a regression problem, the metric utilized will be R2 (R squared), which gives a measure of the variance of the target to a variable that can be explained using the given characteristics. It is a measure of how well the model fits the data in a practical sense. It can be expressed numerically as:

The percentage of R^2 score is $R^2 = (1 - \frac{SS_{Regression}}{SS_{Total}})*100$where,

$SS_{Regression}$= Total Residual Sum of Squares computed from the best fit line

SS_{Total} = Total residual sum of squares measured from the mean

Using the 'r2_score ()' function of the metrics module of the scikit-learn library.

4.1 Linear Models

Linear Regression is the most basic Regression algorithm. There is no need for further complexity if a Linear model can adequately describe the facts. We can use Regularization approaches to penalize the coefficient values of the features as a modification to the original Least Squares Regression because higher values contribute to overfitting and loss of generalization. Linear models benefit tremendously from regularization approaches. In addition, there are only a few circumstances in which a Linear model may fit the data effectively without Regularization. Regularization transforms the problem of Linear Regression into Lasso or Ridge Regression, depending on whether we add the absolute values of coefficients or their squares to our loss function. The R2 score of linear regression without feature engineering, which is used as a baseline model, is 17%

4.1.1 Linear Regression

The linear regression model consists of a predictor variable and a dependent variable related linearly to each other. We try to find the relationship between the independent variable (input) and a corresponding dependent variable (output).

A linear model makes a forecast by adding a constant named the bias term to the weighted sum of the input features (also called the intercept term) [12].

There are few assumptions to be taken care of when fitting a linear regression model. Before that, there are few terms to be known are Homoscedasticity and Multicollinearity. Homoscedasticity (meaning "same variance") describes a situation in which the error term is the same across all values of the independent variables. Heteroscedasticity (the violation of homoscedasticity) is present when the size of the error term differs across values of an independent variable. (Scatter plot: Residual vs Fitted value)

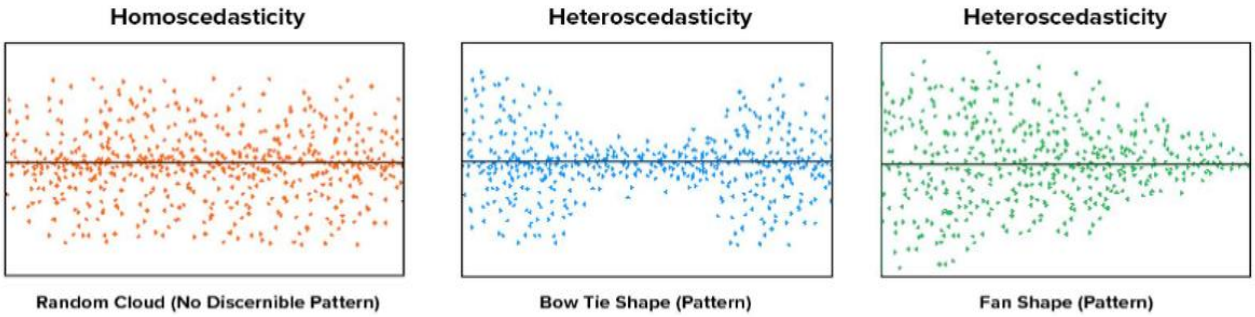

Fig7. Homoscedasticity Vs Heteroscedasticity

Heteroscedasticity does not cause bias in the coefficient estimates, it does make them less precise. Dealing with Heteroscedasticity:

1. Log-transformation of features
2. Outlier treatment

Multicollinearity: Multicollinearity occurs when independent variables in a regression model are correlated. This correlation is a problem because independent variables should be independent. If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results. Dealing with Multicollinearity

1. Remove some of the highly correlated independent variables.
2. Linearly combine the independent variables, such as adding them together.
3. Perform an analysis designed for highly correlated variables, such as principal components analysis

Assumptions of Linear Regression:

1. The relation between the dependent and independent variables should be almost linear.

2. The mean of residuals should be zero or close to 0 as much as possible. It is done to check whether our line is actually the line of "best fit".
3. There should be homoscedasticity or equal variance in a regression model. This assumption means that the variance around the regression line is the same for all values of the predictor variable (X).
4. There should not be multicollinearity in the regression model. Multicollinearity generally occurs when there are high correlations between two or more independent variables.

Model prediction y = $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots\ldots \theta_n x_n$

$\hat{y}$ = predicted value.

n = number of features.

x_i = ith feature value.

θ_j = jth model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$)

Setting the parameters of a model so that it best fits the training set is referred to as training it. The coefficients are features are determined by the model by trial and error and gives the best fit line which low error as an output prediction. For this process, linear regression uses gradient descent.

Gradient Descent is a fairly general optimization process that may identify the best solutions to a wide variety of problems. From Fig8, Gradient Descent's general concept is to iteratively change parameters to minimize additional cost functions. The learning rate hyperparameter determines the size of the steps, which is an important parameter in Gradient Descent[12].

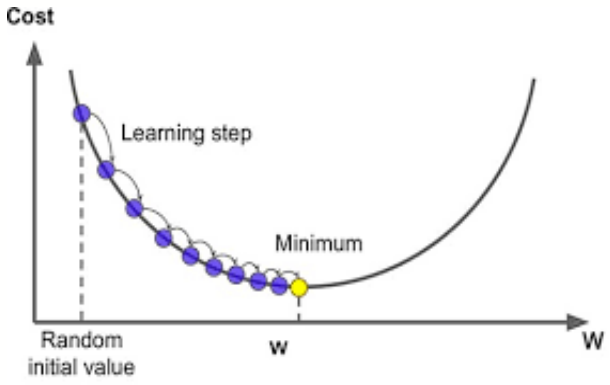

Fig8. Gradient descent in Linear regression

To do so, we'll need a metric to determine how well (or poorly) the model matches the training data.

Advantages of linear regression:

- Linear regression is straightforward to use and the output coefficients are easier to interpret.
- When you know that the relationship between the independent and dependent variables is linear, this approach is the best to employ because it is less complex than other algorithms.
- Linear Regression is susceptible to over-fitting but it can be avoided using some dimensionality reduction techniques, regularization (L1 and L2) techniques and cross-validation.

Disadvantages of linear regression:

- On the other hand, in the linear regression approach, outliers can have a significant impact on the regression, and the technique's bounds are linear.
- It assumes a straight-line relationship between them and it also assumes independence between attributes.

Regularization of a linear model is often accomplished by restricting the model's weights. Ridge Regression, Lasso Regression, and which implement alternative techniques to restrict the weights.

4.1.2. Ridge Regression

Ridge Regression (also known as Tikhonov regularization) is a regularized variant of Linear Regression in which the cost function is given a regularization term of $\alpha \sum_{i=1}^{n} \theta_i^2$

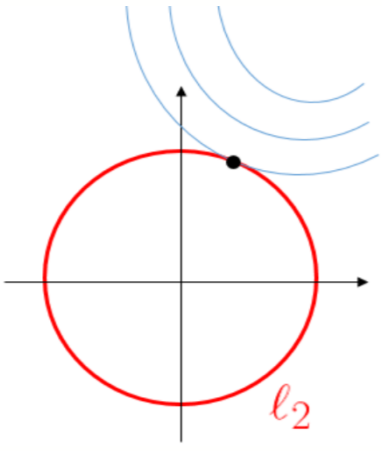

Fig9. Ridge regression with L2 normalization

This forces the learning algorithm to suit the data while also keeping the model weights as low as possible. It's important to keep in mind that the regularization term should only be applied to the cost function during training. You'll want to evaluate the model's performance using the unregularized performance metric once it's been trained.

$F(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$

The type of regularization term to utilize is determined by the penalty hyperparameter. When you choose "l2," SGD will add a regularization term to the cost function equal to half the square of the l2. Ridge Regression is simply the norm of the weight vector[12][13].

Advantages of ridge regression:

- Ridge Regression solves the problem of overfitting by adding a bias term, to fit the model according to the true values of the data.
- The ridge estimator is especially good at improving the least-squares estimate when multicollinearity is present.

Disadvantages of ridge regression:

- Ridge regression, although improving the test accuracy, uses all the input features in the dataset, unlike step-wise methods that only select a few important features for regression.
- Ridge regression reduces the coefficients theta to very low values if the feature is not important, but it won't completely make them zero, hence still using the feature in our model. Lasso regression overcomes this drawback.

4.1.3. Lasso Regression

Least Absolute Shrinkage and Selection Operator Regression (also known as Lasso Regression) is another regularized variant of Linear Regression: it adds a regularization factor to the cost function, same as Ridge Regression, but it employs the l1 norm instead of half the square of the l2 norm, uses the weight vector. The fact that Lasso Regression tries to fully exclude the weights of the least significant variables is a key aspect (i.e., set them to zero) [13].

$F(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} \theta_i$

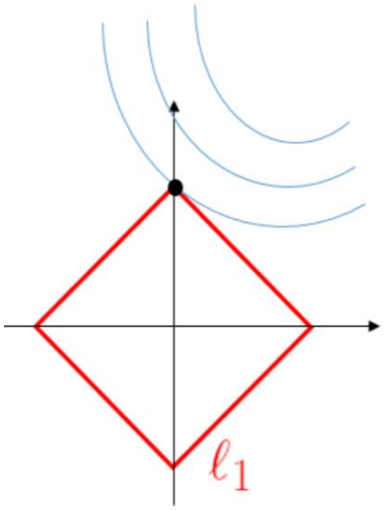

Fig10. Lasso regression with L1 normalization

Advantages of lasso regression:

- By reducing the co-efficient to zero, you can select features.
- Avoids overfitting

Disadvantages of lasso regression:

- A number of features will be highly skewed.
- LASSO will select only one feature from a group of correlated features, the selection is arbitrary in nature.

4.2 Models based on trees

Tree-based Regression models are the next type of method [12]. When compared to Linear models, Tree-based models have a significant advantage in terms of robustness against outliers. Because no linear link has been found between any attribute and the target variable, regression trees are likely to outperform linear models.

Given a large number of characteristics, a Decision Tree will almost certainly overfit the data. As a result, it could be skipped while modeling but it is the base of every other ensemble technique and instead go straight to the ensemble methods listed below, which include building multiple regressors on copies of the same training data and combining their output either through mean, median, mode (Bagging), or growing

trees sequentially (i.e. each tree is built from the data of the previous tree) and using a weighted average of these weak learners (a learner that performs just as well as the strongest learner).

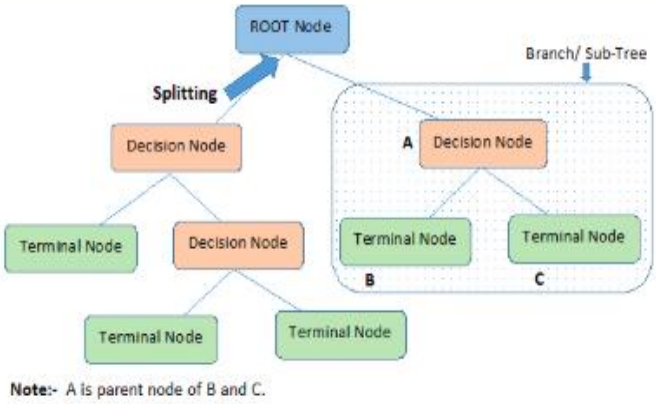

Fig11. Overview of Decision tree split

Random Forests is a common Bagging technique that works well with high-dimensional data like ours. Extra Trees Regressor takes it a step further by randomizing splits. A form of boosting approach is Gradient Boosting Machines. It creates an additive model in which performance is continually improving[13].

4.2.1 Random Forests

A Random Forest is an ensemble of Decision Trees that are normally trained using the bagging method (or occasionally pasting), with max samples set to the size of the training set [24]. You can use the Random Forest Classifier class instead of generating a Bagging Classifier and feeding it a Decision Tree Classifier, which is more convenient and optimized for Decision Trees (similarly, there is a Random Forest Regressor class for regression tasks) [23].

When splitting a node, the Random Forest method incorporates extra randomization by searching for the best feature among a random selection of features rather than searching for the best feature. This leads to more tree variety, which (again) trades a higher bias for a lower variance, resulting in a stronger overall model [13].

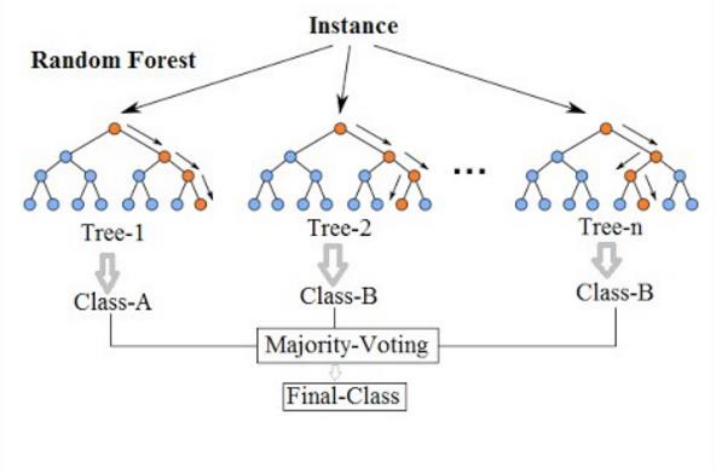

Fig12. Overview Random Forest

Advantages:

- Unlike Decision Tree and other algorithms, the Random Forest algorithm is less prone to overfitting
- The Random Forest method generates the relevance of characteristics, which is really useful.

Disadvantages:

- A little change in the data can cause the Random Forest algorithm to shift dramatically.
- When compared to other algorithms, the Random Forest algorithm computations might be quite complex.

4.2.2 Extra-Trees

When building a tree in a Random Forest, only a random subset of the characteristics are examined for splitting at each node (as discussed earlier). Instead of searching for the best possible thresholds, it is feasible to make trees even more random by employing random thresholds for each feature (like regular Decision Trees do) [12]. Extremely Randomized is the name given to a forest made up of such extremely random trees (or Extra-Trees for short) [26]. Once again, higher bias is exchanged for reduced variance. It also makes Extra-Trees significantly faster to train than standard Random Forests, because one of the most time-consuming jobs of tree growth is determining the best feasible threshold for each feature at each node. The Extra Trees Regressor class in Scikit-Learn [11] can be used to generate an Extra-Trees classifier. It has the same API as the Random Forest Regressor class [13][26].

4.2.3 Gradient Boosting

Gradient Boosting is another prominent boosting strategy. Gradient Boosting, like AdaBoost, operates by successively adding predictors to an ensemble, with each one correcting the one before it. Unlike AdaBoost, this method seeks to adapt the new predictor to the residual errors created by the prior predictor rather than modifying the instance weights at each iteration. The paper [14] has clearly shown the mathematical procedure behind gradient boosting. A subsample hyperparameter is also supported by the Gradient Boosting Regressor class, which sets the fraction of training instances to be used for each tree's training. If subsample=0.25, for example, each tree is trained on 25% of the training cases, chosen at random. As you would have guessed, a bigger bias is exchanged for lower variance. It also makes training much more efficient [21][22].

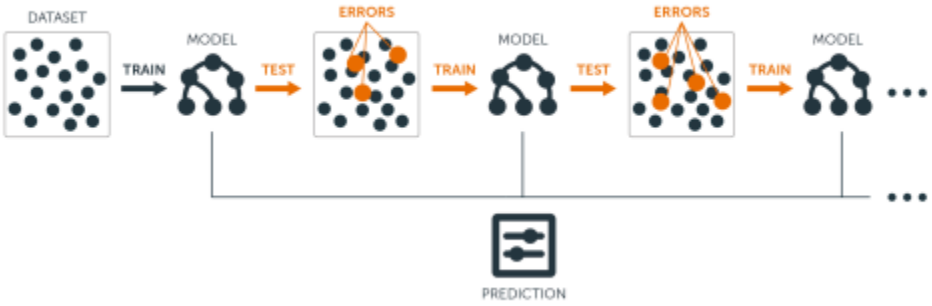

Fig13.Flow of Gradient Boosting

Advantages of Gradient Boosting:

- There is no need to pre-process data; it typically works well with categorical and numerical values as it is.
- Imputation is not required to handle missing data.

Disadvantages of Gradient Boosting:

- Boosting the gradient Models will continue to be improved in order to reduce all faults. This can lead to overfitting by exaggerating outliers.
- Computationally intensive - frequently necessitates a large number of trees (>1000), which can be time and memory-consuming.

4.2.4 XGBoost

XGBoost is a machine learning method that has recently won Kaggle contests for structured or tabular data. XGBoost is a high-speed and high-performance implementation of gradient boosted decision trees. The model's implementation incorporates features from the scikit-learn[11] and R implementations, as well as new features such as regularisation. Gradient boosting is supported in three different ways: The learning rate is included in the gradient boosting algorithm, also known as the gradient boosting machine. Subsampling at the row, column, and column per split levels with stochastic gradient boosting. With both L1 and L2 regularisation, regularised gradient boosting is possible[13]. Along with these models neural networks can also be used. With the help of PCA dimensional reduction can be done[25] to address PCA to select the transformed features that can be more related to the target variable. The discipline of tweaking the hyperparameters of these algorithms to achieve optimal performance is known as hyperparameter optimization. One of the most obvious methods to perform is manually selecting hyperparameters but it is not feasible as we cannot select values that would help in the optimization. Because there are so many choices and options, this technique does not scalable. Many strategies for hyperparameter optimization have been proposed. Grid search, Random search, and Bayesian optimization are three well-known strategies[22].

Advantages of XG Boost:

- Lots of flexibility - can optimize on a variety of loss functions and has various hyperparameter tuning options, making the function fit quite versatile.
- There is no need to pre-process data; it typically works well with categorical and numerical values.

Disadvantages of XG Boost:

- Because of the approach's high flexibility, there are a lot of variables that interact and have a big impact on how it behaves (number of iterations, tree depth, regularization parameters, etc.). During tuning, this necessitates a huge grid search.
- Less interpretable although this is easily addressed with various tools (variable importance, partial dependence plots, etc.

All of the models are modeled using the steps below:

The final set of features used are really important. These features are solely responsible for the better performance of the model[6]. With the help of a correlation plot, these features are chosen. These are the final set of features taken into consideration for modeling. final_features = ['th_kitchen', 'th_living', 'th_laundry','th_office', 'session', 'th_teen', 'th_parents', 'press_mm_hg', 'windspeed', 'avg_house_temp', 'hour', 'avg_house_hum', 'temp_outside', 'humid_outside', 'weekday', 'month','log_energy']. Here for the train-test split 80-20 is used.

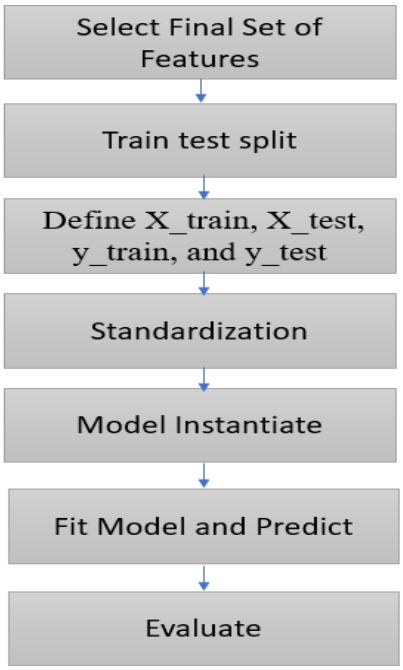

Fig14. Steps for modeling of ML algorithm

4.3 Hyperparameter tuning

Hyperparameters are important because they directly control the behavior of the training algorithm and have a significant impact on the performance of the model is being trained. Efficiently search the space of possible hyperparameters. Easy to manage a large set of experiments for hyperparameter tuning. For hyper-parameters selection is performed by GridsearchCV. In gridsearchCV, every combination of hyperparameters is evaluated and the best params method output the best hyperparameters. There are many different hyperparameters, here three of them are considered.

max_depth: *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

n_estimators: *int, default=100*
The number of trees in the forest.

max_features:*int, float or {"auto", "sqrt", "log2"}, default=None*
The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split.
- If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features).
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

5. CODE & IMPLEMENTATION

5.1 Mounting drive & Importing data

Introduction to Colab platform: The web IDE for python that enables machine learning using cloud-based storage is now published by google. In late 2017, this internal device had a rather quiet publicity and is scheduled to have a major effect in engineering, artificial intelligence and data-science work. "It is an environment of a Jupyter notebook which requires no setup and runs completely within the cloud." It's an integrated drive platform on the web. It makes working in a group easier. Since this only requires the drive link in notebooks similar to Google Docs to change the platform online from different sites at the same time.

```
# Mounting drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Importing libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import warnings
#Reading data
missing_values = ['N/a', 'na', 'np-nan','None', 'none']
df=pd.read_csv("/content/drive/MyDrive/Batch17_Mainproject/Data_notebook/energydata_complete.csv", na_values= missing_values,parse_dates=['date'])
```

5.2 Understanding data

```
# Shape of the data set
df.shape
```

```
(19735, 29)
```

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19735 entries, 0 to 19734
Data columns (total 29 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         19735 non-null  datetime64[ns]
 1   Appliances   19735 non-null  int64
 2   lights       19735 non-null  int64
 3   T1           19735 non-null  float64
 4   RH_1         19735 non-null  float64
 5   T2           19735 non-null  float64
 6   RH_2         19735 non-null  float64
 7   T3           19735 non-null  float64
 8   RH_3         19735 non-null  float64
 9   T4           19735 non-null  float64
 10  RH_4         19735 non-null  float64
 11  T5           19735 non-null  float64
 12  RH_5         19735 non-null  float64
 13  T6           19735 non-null  float64
 14  RH_6         19735 non-null  float64
 15  T7           19735 non-null  float64
 16  RH_7         19735 non-null  float64
 17  T8           19735 non-null  float64
 18  RH_8         19735 non-null  float64
 19  T9           19735 non-null  float64
 20  RH_9         19735 non-null  float64
 21  T_out        19735 non-null  float64
 22  Press_mm_hg  19735 non-null  float64
 23  RH_out       19735 non-null  float64
 24  Windspeed    19735 non-null  float64
 25  Visibility   19735 non-null  float64
 26  Tdewpoint    19735 non-null  float64
 27  rv1          19735 non-null  float64
 28  rv2          19735 non-null  float64
dtypes: datetime64[ns](1), float64(26), int64(2)
memory usage: 4.4 MB
```

Renaming columns

```
# Lowercase the column names
df.columns = [x.lower() for x in df.columns]
energy_df = df.rename(columns=temp_and_humid_dict)

temp_dict =  {
    't1' : 'temp_kitchen', 't2' : 'temp_living', 't3' : 'temp_laundry',
    't4' : 'temp_office', 't5' : 'temp_bath', 't6' : 'temp_outside',
    't7' : 'temp_iron', 't8' : 'temp_teen', 't9' : 'temp_parents', 't_out' : 'temp_station'}

humid_dict = {
    'rh_1' :'humid_kitchen', 'rh_2' : 'humid_living', 'rh_3' : 'humid_laundry',
    'rh_4' : 'humid_office', 'rh_5' : 'humid_bath', 'rh_6' : 'humid_outside',
    'rh_7' : 'humid_iron', 'rh_8' : 'humid_teen', 'rh_9' : 'humid_parents', 'rh_out' : 'humid_station'
```

}

energy_df[temp_dict.values()].describe()

	temp_kitchen	temp_living	temp_laundry	temp_office	temp_bath	temp_outside	temp_iron	temp_teen	temp_parents	temp_station
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	21.686571	20.341219	22.267611	20.855335	19.592106	7.910939	20.267106	22.029107	19.485828	7.411665
std	1.606066	2.192974	2.006111	2.042884	1.844623	6.090347	2.109993	1.956162	2.014712	5.317409
min	16.790000	16.100000	17.200000	15.100000	15.330000	-6.065000	15.390000	16.306667	14.890000	-5.000000
25%	20.760000	18.790000	20.790000	19.530000	18.277500	3.626667	18.700000	20.790000	18.000000	3.666667
50%	21.600000	20.000000	22.100000	20.666667	19.390000	7.300000	20.033333	22.100000	19.390000	6.916667
75%	22.600000	21.500000	23.290000	22.100000	20.619643	11.256000	21.600000	23.390000	20.600000	10.408333
max	26.260000	29.856667	29.236000	26.200000	25.795000	28.290000	26.000000	27.230000	24.500000	26.100000

energy_df[humid_dict.values()].describe()

	humid_kitchen	humid_living	humid_laundry	humid_office	humid_bath	humid_outside	humid_iron	humid_teen	humid_parents	humid_station
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000
mean	40.259739	40.420420	39.242500	39.026904	50.949283	54.609083	35.388200	42.936165	41.552401	79.750418
std	3.979299	4.069813	3.254576	4.341321	9.022034	31.149806	5.114208	5.224361	4.151497	14.901088
min	27.023333	20.463333	28.766667	27.660000	29.815000	1.000000	23.200000	29.600000	29.166667	24.000000
25%	37.333333	37.900000	36.900000	35.530000	45.400000	30.025000	31.500000	39.066667	38.500000	70.333333
50%	39.656667	40.500000	38.530000	38.400000	49.090000	55.290000	34.863333	42.375000	40.900000	83.666667
75%	43.066667	43.260000	41.760000	42.156667	53.663333	83.226667	39.000000	46.536000	44.338095	91.666667
max	63.360000	56.026667	50.163333	51.090000	96.321667	99.900000	51.400000	58.780000	53.326667	100.000000

5.3 Data driven analysis of energy usage

energy_df.energy.describe()

```
count    19735.000000
mean        97.694958
std        102.524891
min         10.000000
25%         50.000000
50%         60.000000
75%        100.000000
max       1080.000000
Name: energy, dtype: float64
```

```
sns.histplot(energy_df['energy'], color = 'red', bins=50)
```

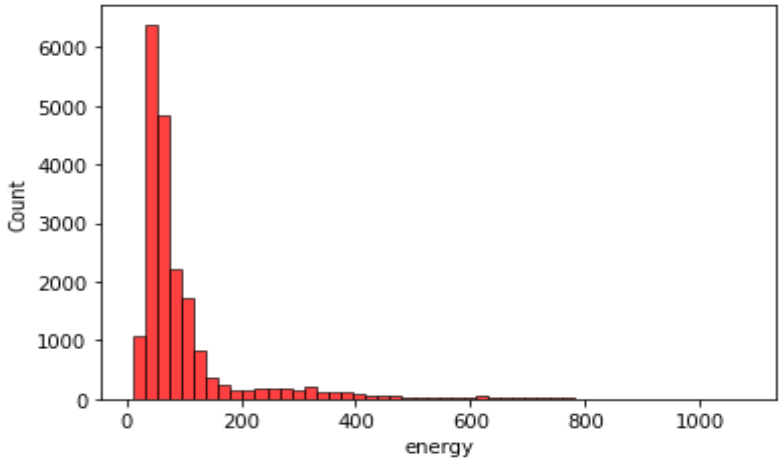

Fig15. Histplot of Energy consumption

```
sns.distplot(energy_df["energy"],color= 'green',bins= 50 )
```

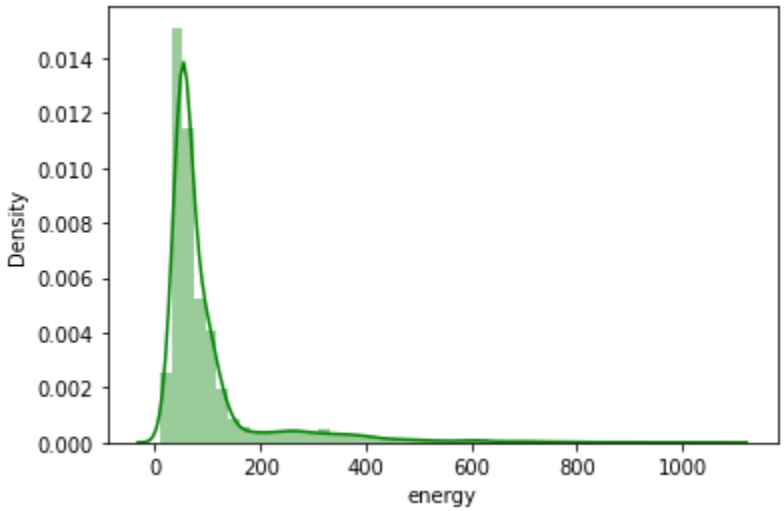

Fig16. Density plot of Energy consumption

5.4 Trends seen with time

```
# Append more columns to the Data Frame based on datetime

energy_df = energy_df.set_index('date')
energy_df['month'] = energy_df.index.month
energy_df['weekday'] = energy_df.index.weekday
energy_df['hour'] = energy_df.index.hour
```

```
energy_df['week'] = energy_df.index.week
```

5.4.1 # Energy consumed Vs hour plot

```
plt.style.use('default')
fig, ax = plt.subplots(1,1,figsize=(5,4))
plt.grid(color='w', linestyle='solid')
ax.set_axisbelow(True)
energy_df.groupby('hour').agg({'energy' : 'mean'}).plot.bar(ax=ax)
ax.set_ylabel('energy consumed (Wh)')
```

The output of Energy consumed Vs hour plot is explained in Fig6.

5.4.2 # Energy Vs month,hour plot

```
fig, ax = plt.subplots(1,1,figsize=(20,4))
energy_df.groupby(['month','hour']).agg({'energy' : 'mean'}).plot.bar(ax=ax)
ax.set_ylabel('Appliance energy (Wh)')
```

The output of Energy trend over months for every hour is explained in Fig1.

5.4.3 # Energy consumed Vs hour plots on weekdays and weekends

```
fig, ax = plt.subplots(1,2,figsize=(10,4))
week_df = energy_df.groupby(['weekday','hour']).agg({'energy':'mean'}).reset_index(0)
week_df[week_df.weekday==0].energy.plot.bar(ax=ax[0], label='weekends')
week_df[week_df.weekday==1].energy.plot.bar(ax=ax[1], label='weekdays')
ax[0].legend(loc='best')
ax[1].legend(loc='best')
ax[0].set_ylabel('Appliance Energy (Wh)')
ax[1].set_ylabel('Appliance Energy (Wh)')
```

The output here is Fig2(i),Fig2(ii) is explained in Chapter2.

```
#temperature variation in month, hour
fig, ax = plt.subplots(1,1,figsize=(20,5))
```

energy_df.groupby(['month','hour']).agg({'temp_outside' : 'mean'}).plot.bar(ax=ax)

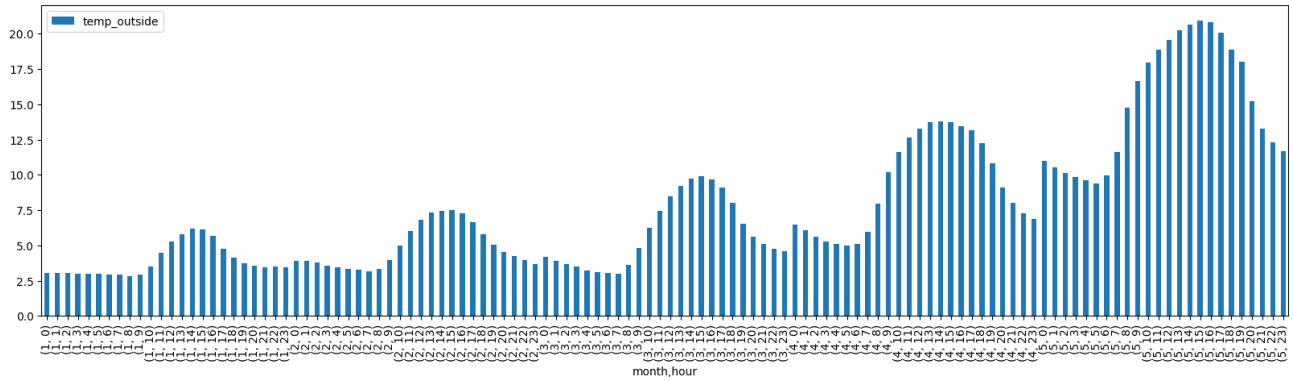

Fig17. Temperature variation in month, hour

#humidity variation in month,hour

fig, ax = plt.subplots(1,1,figsize=(20,5))

energy_df.groupby(['month','hour']).agg({'humid_outside' : 'mean'}).plot.bar(ax=ax)

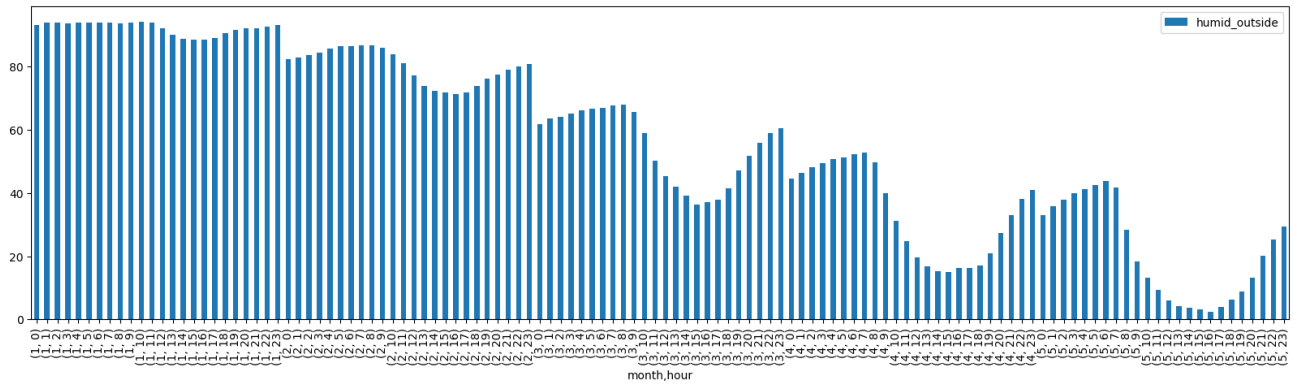

Fig18. Humidity variation in month, hour

energy_df[['temp_outside', 'humid_outside','energy']].corr()

	temp_outside	humid_outside	energy
temp_outside	1.000000	-0.671619	0.122468
humid_outside	-0.671619	1.000000	-0.087261
energy	0.122468	-0.087261	1.000000

5.4.4 # Pearson Correlation among the variables

```
col = ['temp_kitchen','temp_living', 'temp_laundry', 'temp_office', 'temp_bath', 'temp_outside',
'temp_iron', 'temp_teen', 'temp_parents', 'temp_station', 'humid_kitchen', 'humid_living',
'humid_laundry', 'humid_office', 'humid_bath', 'humid_outside', 'humid_iron', 'humid_teen',
'humid_parents', 'humid_station','press_mm_hg', 'windspeed', 'visibility', 'tdewpoint','hour','energy']

corr = energy_df[col].corr()
plt.figure(figsize = (18,18))
sns.set(font_scale=1)
sns.heatmap(corr, cbar = True, annot=True, square = True, fmt = '.2f', xticklabels=col, yticklabels=col)
plt.show();

# The output heat map of these features is in Fig4.
```

5.5 Feature Engineering:

The process of extracting additional characteristics from raw data in order to improve the learning algorithm's predictive power. Engineered features should collect information that isn't readily visible in the original feature set.

```
# Find outliers

sorted_energy = energy_df.sort_values('energy',ascending=False)
sorted_energy.reset_index(drop=True,inplace = True)
print("The number of the 0.5% top values of load is",len(sorted_energy.head
(len(sorted_energy)//500)),"and they have power load higher than", sorted_energy.energy[39], "Wh.")
```

5.5.1 # outlier detection with box plot

```
sns.set(style="whitegrid")
ax = sns.boxplot(sorted_energy.energy)
```

The number of the 0.5% top values of load is 39 and they have power load higher than 740 Wh.

The output of Box plot of energy feature is shown in Fig5.

```
#Outliers removal
energy_df = energy_df.dropna()
energy_df = energy_df.drop(energy_df[(energy_df.energy>740)|(energy_df.energy<0)].index)
```

5.5.2 # log energy

```
energy_df['log_energy'] = np.log(energy_df.energy)
```

5.5.3 # Average house temperature and humidity

```
energy_df['avg_house_temp'] =(energy_df.temp_kitchen + energy_df.temp_living + energy_df.temp_laundry + energy_df.temp_office + energy_df.temp_bath + energy_df.temp_iron + energy_df.temp_teen + energy_df.temp_parents)/8

energy_df['avg_house_hum'] =(energy_df.humid_kitchen + energy_df.humid_living + energy_df.humid_laundry + energy_df.humid_office + energy_df.humid_bath + energy_df.humid_iron + energy_df.humid_teen + energy_df.humid_parents)/8
```

5.5.4 # Multiplicative features

```
# Products of several features to remove additive assumption
energy_df['hour*lights'] = energy_df.hour * energy_df.lights
energy_df['th_kitchen'] = energy_df.temp_kitchen * energy_df.humid_kitchen
energy_df['th_living'] = energy_df.temp_living * energy_df.humid_livng
energy_df['th_laundry'] = energy_df.temp_laundry * energy_df.humid_laundry
energy_df['th_office'] = energy_df.temp_office * energy_df.humid_office
energy_df['th_bath'] = energy_df.temp_bath * energy_df.humid_bath
energy_df['th_iron'] = energy_df.temp_iron * energy_df.humid_iron
energy_df['th_teen'] = energy_df.temp_teen * energy_df.humid_teen
energy_df['th_parents'] = energy_df.temp_parents * energy_df.humid_parents
energy_df['th_outside'] = energy_df.temp_outside * energy_df.humid_outside

energy_df.shape
(19697, 45)
```

```
 # Calculate average energy load per weekday and hour
def code_mean(data, cat_feature, real_feature):
    """
    Returns a dictionary where keys are unique categories of the cat_feature,
    and values are means over real_feature
    """
    return dict(data.groupby(cat_feature)[real_feature].mean())
 # Functions to be used from the plots

def daily(x,energy_df=energy_df):
    return energy_df.groupby('weekday')[x].mean()
def hourly(x,energy_df=energy_df):
    return energy_df.groupby('hour')[x].mean()

def monthly_daily(x,energy_df=energy_df):
    by_day = energy_df.pivot_table(index='weekday',
                                   columns=['month'],
                                   values=x,
                                   aggfunc='mean')
    return round(by_day, ndigits=2)
# Plot of Mean Energy Consumption per Hour of a Day

hourly('energy').plot(figsize=(10,8))
plt.xlabel('Hour')
plt.ylabel('Appliances consumption in Wh')
ticks = list(range(0, 24, 1))
plt.title('Mean Energy Consumption per Hour of a Day')

plt.xticks(ticks);
```

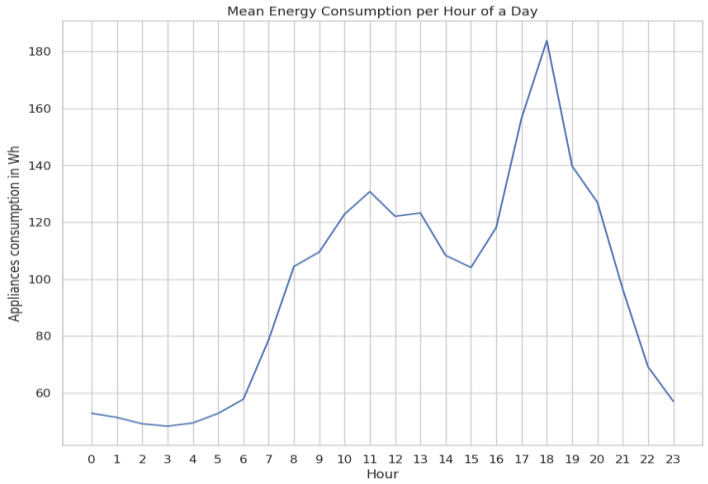

Fig19. Mean energy consumption hour of a day (Load curve)

```
# Histogram of Appliance's consumption


f, axes = plt.subplots(1, 2,figsize=(10,3))
sns.distplot(energy_df_hour.energy,hist=True,color='blue',hist_kws={'edgecolor':'black'},ax=axes[0])
axes[0].set_title("Energy consumption")
axes[0].set_xlabel('Energy wH')
sns.distplot(energy_df_hour.log_energy,hist=True,color='blue',hist_kws={'edgecolor':'black'},ax =axes[1
])
axes[1].set_title("Log energy consumption")
axes[1].set_xlabel('Energy log(wH)')
#The output here of energy distribution before and after log transformation is explained at Fig3(i), Fig3(ii).
def create_session(x):
  if x <= 6 or x >= 22:
    return 1
  elif x>6 and x <=15:
    return 2
  else:
    return 3
```

5.5.5 # Creating new column ‘session’

```
energy_df['session'] = energy_df['hour'].apply(lambda x : create_session(x))
```

5.5.6 # Pearson Correlation among the variables after feature engineering

```
col = ['th_kitchen', 'th_living', 'th_laundry', 'th_office', 'th_teen', 'th_parents', 'press_mm_hg', 'windspeed',
'avg_house_temp','avg_house_hum','hour*lights','Tdewpoint','hour','temp_outside','humid_outside','sessi
on','weekday','month','log_energy']
corr = energy_df_hour[col].corr()
plt.figure(figsize = (18,18))
sns.set(font_scale=1)
sns.heatmap(corr, cbar = True, annot=True, square = True, fmt = '.2f', xticklabels=col, yticklabels=col)
plt.show();
```

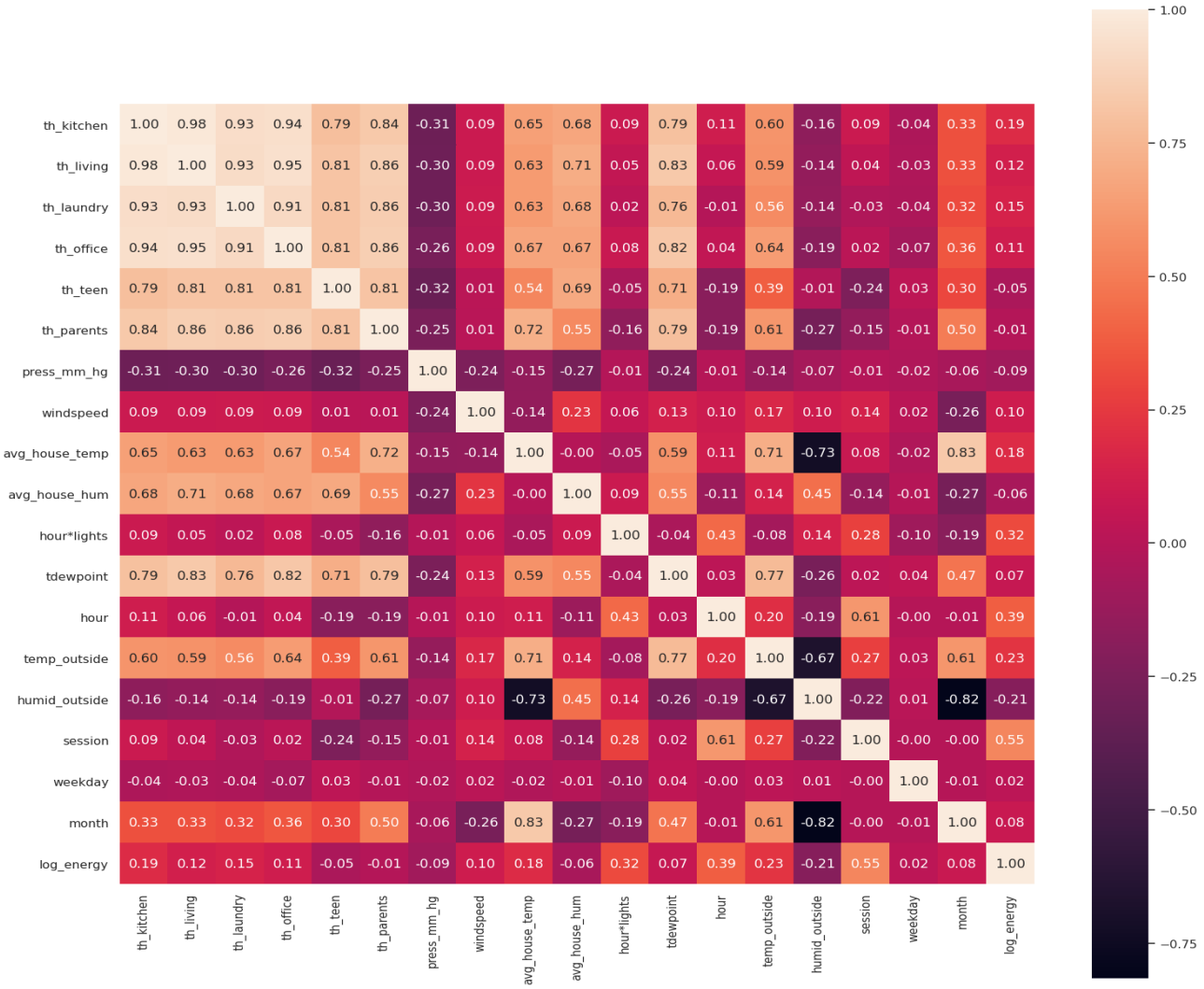

Fig20. Pearson Correlation among the variables after feature engineering.

5.6 Modelling:

A Machine Learning model is a file that has been trained to recognize certain types of patterns. We train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
ExtraTreesRegressor
import xgboost as xgb
import time
from math import sqrt
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV


final_features = ['th_kitchen', 'th_living', 'th_laundry', 'th_office',   'th_teen', 'th_parents', 'press_mm_hg',
'windspeed', 'avg_house_temp', 'avg_house_hum',  'hour*lights', 'tdewpoint','hour', 'temp_outside',
'humid_outside', 'session','weekday','month','log_energy']

final_train_df, final_test_df = train_test_split(energy_df[final_features], test_size = 0.2, random_state =
1)

X_train, y_train = final_train_df.drop('log_energy', axis=1), final_train_df['log_energy']
X_test, y_test = final_test_df.drop('log_energy', axis=1), final_test_df['log_energy']

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.values.reshape([-1,1])).flatten()
y_test = sc_y.transform(y_test.values.reshape([-1,1])).flatten()
models =   ['Lasso: ', Lasso()],
          ['Linear:',linear_model.LinearRegression()],
          ['Ridge: ', Ridge()],
          ['KNeighbors: ',  neighbors.KNeighborsRegressor()],
          ['RandomForest ',RandomForestRegressor()],
          ['ExtraTrees :',ExtraTreesRegressor()],
          ['GradientBoost: ', GradientBoostingRegressor()],
          ['XGBoost: ', xgb.XGBRegressor()]]

 # Predicting and evaluating models
model_data = []
for name,curr_model in models :
    curr_model_data = {}
    curr_model.random_state = 78
    curr_model_data["Name"] = name
    start = time.time()
    curr_model.fit(X_train,y_train)
    end = time.time()
    curr_model_data["Train_Time"] = end - start
    curr_model_data["Train_R2_Score"] = 100*r2_score (y_train, curr_model. predict (X_train))
    curr_model_data["Test_R2_Score"] = 100*r2_score(y_test,curr_model.predict(X_test))
    model_data.append(curr_model_data)
results_df = pd.DataFrame(model_data)
#Refer Table4 in 6.2 before hyper parameter tuning for results and explanation.

results_df.plot.bar(x="Name", y=['Test_R2_Score' , 'Train_R2_Score' ], title = 'Results' , width = .6,
figsize= (6,3))
#Refer Fig23 in 6.2 Before hyper parameter tuning for variations with different algorithms by barplot.
```

```
etr_model = ExtraTreesRegressor()
rf_model = RandomForestRegressor()
etr_model.fit(X_train,y_train)
rf_model.fit(X_train,y_train)
y2_pred = etr_model.predict(X_test)
y3_pred = rf_model.predict(X_test)


fig = plt.figure(figsize=(10,5))
ax = plt.axes(facecolor ='white')
plt.plot(y_test[:100],label='Target value',color='b')
plt.plot(y2_pred[:100],label=' Tree Prediction ', linestyle='-', color='g')
plt.plot(y3_pred[:100],label='Random forest Prediction ', linestyle='-', color='r')
plt.legend(loc=1)
#For result in Refer Fig24 in 6.2 for prediction graph variations with extra-tree and random forest with actual value.


y1_pred = lin_model.predict(df_X_test)
fig = plt.figure(figsize=(10,5))
plt.plot(y_test[:100],label='Target value',color='b')
plt.plot(y1_pred[:100],label='Linear Prediction ', linestyle='-', color='y')
plt.legend(loc=1)
```

5.7 PCA:

Principal Component Analysis is basically a statistical procedure to convert a set of observation of possibly correlated variables into a set of values of linearly uncorrelated variables.

```
train_energy_df, test_energy_df = train_test_split(energy_df, test_size=0.2, random_state=1)
```

5.7.1 # PCA on temperatures

```
pca = PCA()
pca.fit(train_energy_df[temp_cols])
temp_pca = pca.transform(energy_df[temp_cols])
```

```
for i in range(temp_pca.shape[1]):
  energy_df[f'temp_pca{i+1}'] = temp_pca[:,i]
```

5.7.2 # PCA on humidity

```
pca = PCA()
pca.fit(train_energy_df[humid_cols])
humid_pca = pca.transform(energy_df[humid_cols])
for i in range(humid_pca.shape[1]):
  energy_df[f'humid_pca{i+1}'] = humid_pca[:,i]
final_features = ['temp_pca1', 'temp_pca2','temp_pca6','humid_pca1', 'humid_pca2', 'humid_pca4',
'temp_outside', 'humid_outside', 'weekday', 'session', 'windspeed', 'press_mm_hg', 'log_energy']
fig,ax = plt.subplots(1,1,figsize=(20,20))
sns.heatmap(energy_df[final_features].corr(), ax = ax, annot=True)
```

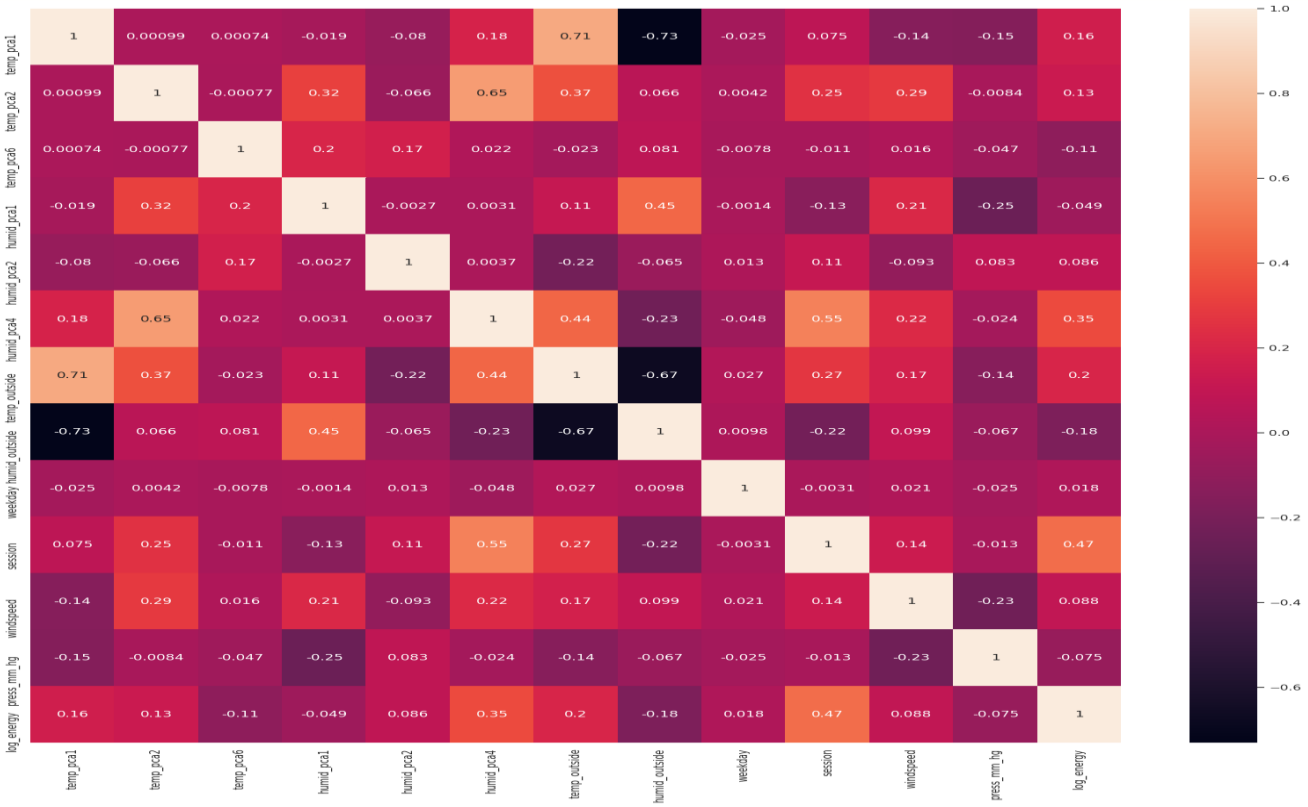

Fig21. Pearson Correlation among the variables of final features

5.8 Hyper parameter tuning

```
param_grid = [{
            'max_depth': [80, 150, 200,250],
            'n_estimators' : [100,150,200,250],
            'max_features': ["auto", "sqrt", "log2"]
           }]
reg = ExtraTreesRegressor(random_state=40)
grid_search = GridSearchCV(estimator = reg, param_grid = param_grid, cv = 5, n_jobs = -1 , scoring='r2'
, verbose=2)
grid_search.fit(pca_X_train, pca_y_train)
```

```
grid_search.best_params_
```

```
{'max_depth': 80, 'max_features': 'sqrt', 'n_estimators': 250}
```

```
grid_search.best_estimator_
```

```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=80, max_features='sqrt', max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=250, n_jobs=None, oob_score=False,
                    random_state=40, verbose=0, warm_start=False)
```

```
results_df_pca_hy =  pd.DataFrame(pca_model_data)
#Refer Table5 in 6.3 after hyper parameter tuning for results and explanation.
```

```
results_df_pca_hy.plot.bar(x="Name", y=['Test_R2_Score' , 'Train_R2_Score' ], title = 'Results' , width =
.6 ,figsize= (6,3))
```

```
#Refer Fig25 in 6.3 After hyper parameter tuning for variations with different algorithms by barplot.
```

```
fig = plt.figure(figsize=(10,5))
```

```
ax = plt.axes(facecolor ='white')

y2_pred = etr_model.predict(pca_X_test)
y3_pred = rf_model.predict(pca_X_test)

plt.plot(y_test[:100],label='Target value',color='b')
plt.plot(y2_pred[:100],label=' Tree Prediction ', linestyle='-', color='g')
plt.plot(y3_pred[:100],label='Random forest Prediction ', linestyle='-', color='r')
plt.legend(loc=1)

#For result Refer Fig26 in 6.3 for prediction graph variations with extra-tree and random forest with actual value.

feature_indices = np.argsort(grid_search.best_estimator_.feature_importances_)
importances = grid_search.best_estimator_.feature_importances_
indices = np.argsort(importances)[::-1]
names = [final_train_df.columns[i] for i in indices]

plt.figure(figsize=(6,3))
plt.title("Feature Importance")
plt.bar(range(pca_X_train.shape[1]), importances[indices])
plt.xticks(range(pca_X_train.shape[1]), names, rotation=90)
plt.show()
# The results for feature importance of extra tree regressor model refer Fig27.
```

6. RESULTS AND PREDICTION GRAPHS

6.1 Results for Baseline model

S. No	Name	Train_Time	Train_R2_Score	Test_R2_Score
1.	Lasso	0.00996	0.0000	-0.0777
2.	Linear	0.021046	16.9169	16.9655
3.	Ridge	0.008213	16.9168	16.9659

Table3. R2 score of Linear Models before feature engineering

The above Table3. shows the baseline model results. For the baseline model the selected models are linear models and the modelling is done before feature engineering with the raw data. When the modelling is done with raw data only 17% of variance is only predicted.

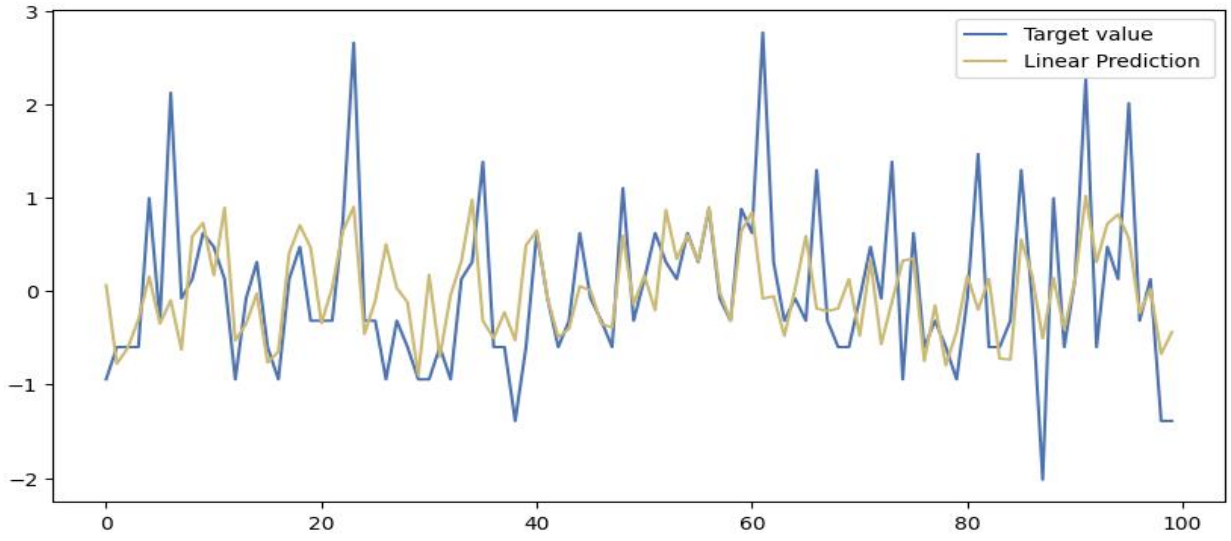

Fig22. Energy Prediction of Baseline model (Linear Regression) before feature engineering

The below Fig22. Shows the prediction graph for 100 samples. Blue color line indicates the target value which is actual value and the prediction done with linear regression algorithm is shown by yellow line. It is observed that the model was not able to perform well.

6.2 Results after feature engineering without hyperparameter tuning

S. No	Name	Train_Time	Train_R2_Score	Test_R2_Score
1.	Lasso	0.010408	0.0000	-0.0110
2.	Linear	0.017416	30.0561	28.7402
3.	Ridge	0.007331	30.0561	28.7407
4.	K-Neighbors	0.250363	75.7320	61.0030
5.	Random Forest	22.978521	95.6592	67.4624
6.	Extra Trees Regressor	6.360532	100.0000	71.000213
7.	Gradient Boosting	7.515340	47.1071	42.2821
8.	XGBoost	1.823460	46.9366	42.1626

Table4. R2 score of different algorithms after feature engineering

The above Table4. shows the results for different algorithms after feature engineering. Linear and Ridge has 28% R2 score on test data. Random forest has 67% and Extra-tree regressor has 71%. line.

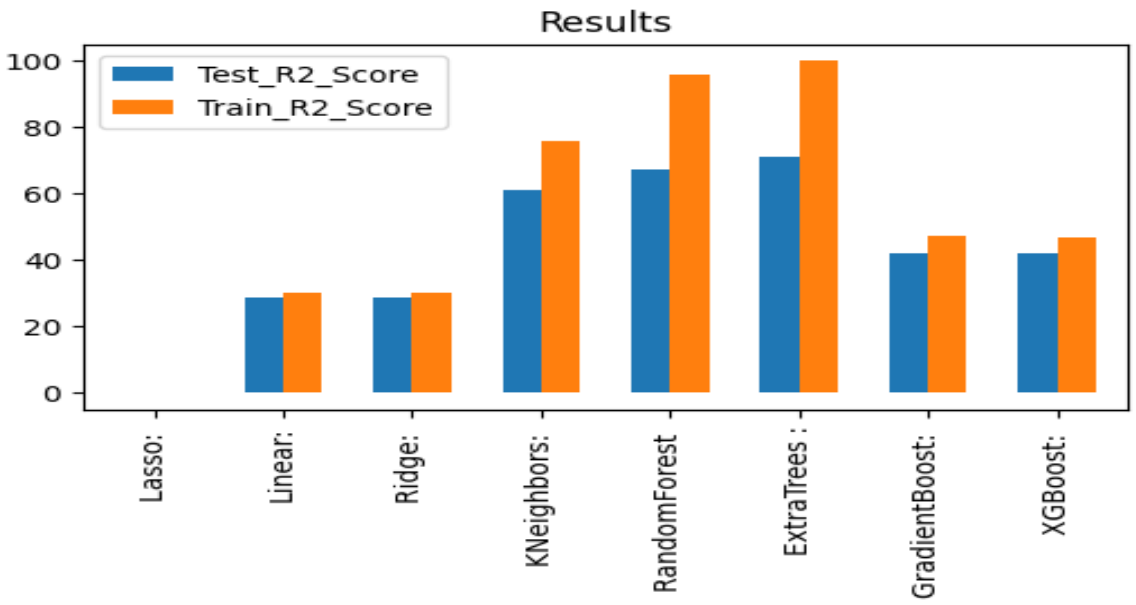

Fig23. R2 score for train and test sets before hyperparameter tuning

The above Fig23. Shows the bar plot visualizing the results. It can be seen that extra-trees is performing high than all other algorithms. It can be seen that linear and ridge are having very close test and train scores.

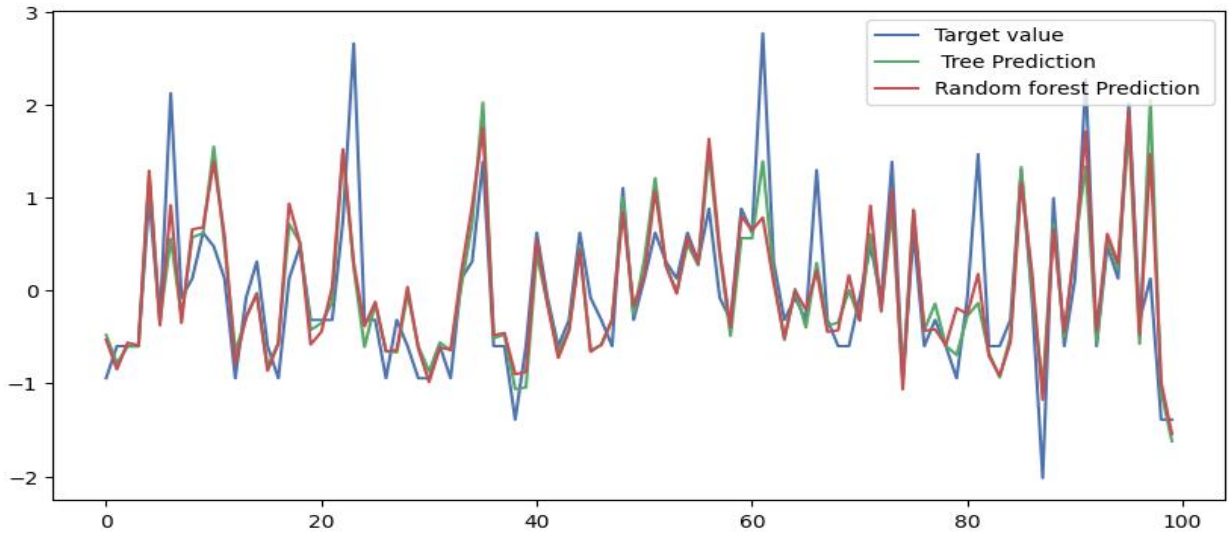

Fig24. Energy Prediction for ExtraTrees and Random forest algorithms.

The above Fig24. Shows the prediction graph for 100 samples. Blue color line indicates the target value which is actual value. The prediction done with extra tree algorithm is shown by green and random forest with red color. It can observed that extra tree algorithm is able to mimic the original values and was able able generalize more than all the other models.

6.3 Results with hyperparameter tuning

S. No	Name	Train_Time	Train_R2_Score	Test_R2_Score
1.	Lasso	0.013325	0.0000	-0.0110
2.	Linear	0.01014	26.9782	27.1519
3.	Ridge	0.007331	26.9782	27.1520
4.	K-Neighbors	0.035377	77.3739	64.0166
5.	Random Forest	15.88206	95.7882	70.1538
6.	Extra Trees Regressor	4.9309	100.0000	74.4962

7.	Gradient Boosting	5.104979	43.06144	40.2229
8.	XGBoost	1.23693	43.13197	39.9348

Table5. R2 score of different algorithms after hyperparameter tuning

The above Table5. shows the results for different algorithms after feature engineering and hyper parameter tuning. Linear and Ridge has 27% R2 score on test data. Random forest has 70% and Extra-tree regressor has 74% scores.

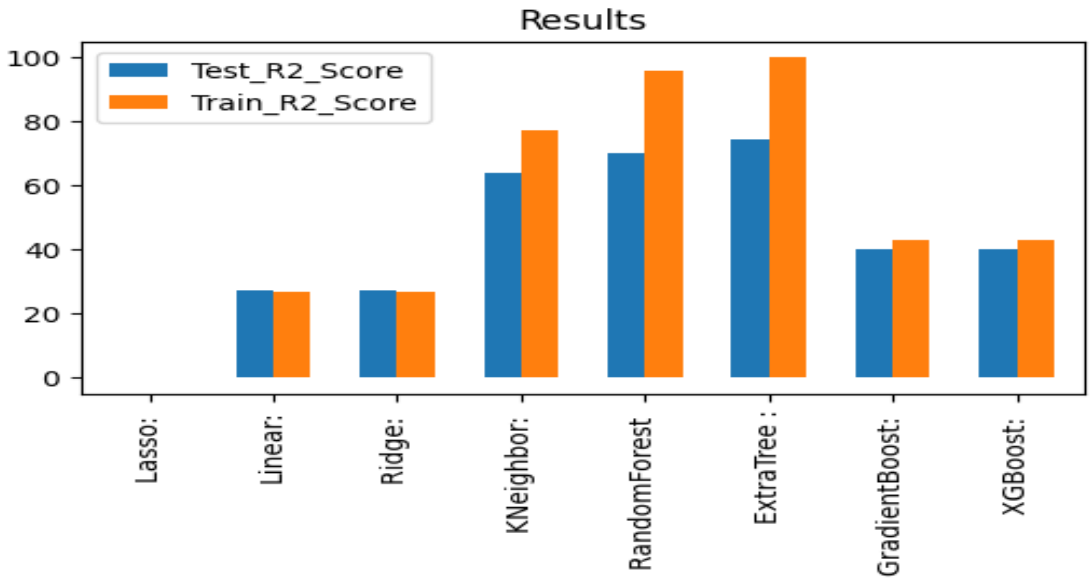

Fig25. R2 score for train and test sets after hyperparameter tuning

The above Fig25. Shows the bar plot visualizing the results. It can be seen that extra-trees is performing high than all other algorithms. It can be seen that linear and ridge are having very close test and train scores. After hyper parameter tuning it is observed that the test and train scores have come slightly closer. It seems the models more promising now than before.

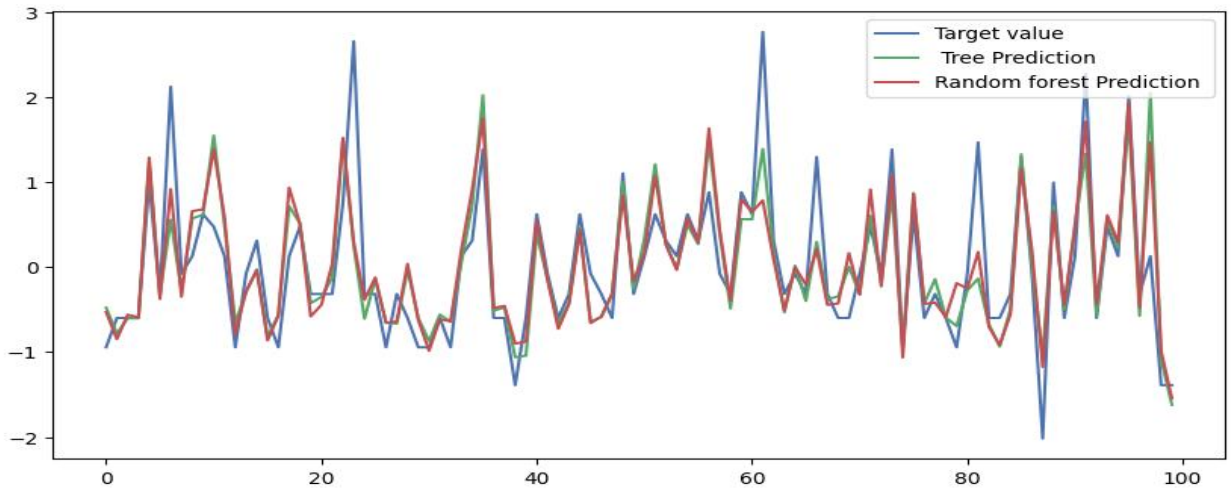

Fig26. Energy Prediction for Extratree and Random Forest algorithms after tuning.

The above Fig26. Shows the prediction graph for 100 samples. Blue color line indicates the target value which is actual value. The prediction done with extra tree algorithm is shown by green and random forest with red color. It can observed that extra tree algorithm is able to mimic the original values and was able able generalize more than all the other models. The winner model is extra-tree regressor. The prediction rate has increased by 4% after hyper parameter tuning. However, the sudden peaks are difficult to predict as they are not happening in general.

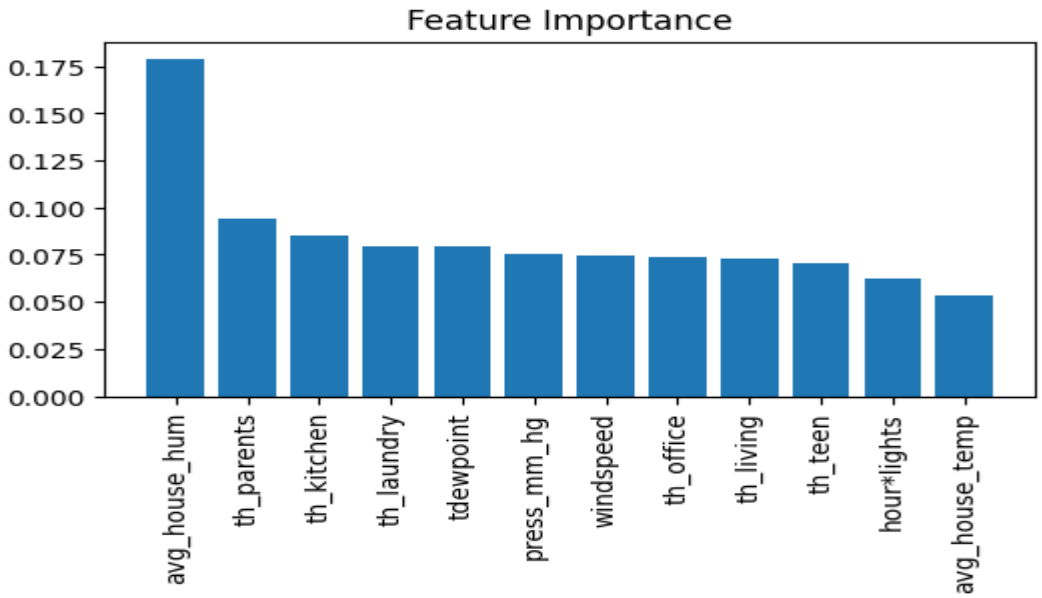

Fig27. Feature importance for ExtraTrees regressor model

Features important here are average house humidity and multiplicative features engineered in chapter 3 are seen to have more importance as seen in Fig27.

Extra tree regressor is performing incredibly well, and the Random Forest regressor is also working well, as shown in the table above. Additional hyperparameter tuning was done on the Extra tree regressor, which resulted in an R^2 score of 74.5% utilising the optimal parameters as {'max_depth': 80, 'max_features': 'sqrt', 'n_estimators': 250}. As a result, predictions can be made using these models.

Lasso is at worst in this case this might be because features that are highly correlated are discarded by making the coefficients zero except only the one which is selected. As a result, the model is not at all able to capture the pattern here.

7. CONCLUSION

The home energy consumption prediction is done here based on linear models (Linear Regression, Ridge Regression, Lasso Regression) and tree-based models (Random Forest, Extra Tree, Gradient Boosting, XG Boosting). Initially preprocessing of data is performed, where some features are extracted from the given data and as the units and ranges of the features are different data is normalized. Here to select the final set of features, heat map generated using correlations is used. Hyperparameter tuning is also used with Grid search CV to determine the optimal parameters in the model. Finally, the prediction performance of each model was evaluated and compared. The results show that among the seven prediction models established by the machine learning method, EXTRA TREE has achieved remarkable results in the training set and the testing set comparatively. Tree models has seen good prediction performance in the train and test sets, but linear models didn't perform good.

The prediction performance is determined with the help of R2 score where it gives variability that is been predicted and it should be high for the ideal model. Extra tree is much better than that of the other machine learning models and the predicted value of Extra Tree model is close to the actual value.

Despite being substantially linearly associated within themselves, the temperature/humidity features exhibited a little linear (Pearson) connection with the dependent variable. Although fascinating characteristics were engineered by modifying features, when it comes to energy use, the time of day is crucial. Extra Trees Regressor is the best algorithm to deploy for this dataset (tree-based algorithm). The untuned model could explain 71% of the variance (R^(2) score = 0.710) on the test set, while the tuned model could explain 74.5% of the variance (R^(2) score = 0.7449). Average house humidity and other features which are engineered are showing more feature importance. When dealing with a data set with most of its features having a little correlation with the dependent feature, tree-based models are by far the best models.

8. REFERENCES

[1] Md. Sumon Shahriar and M. Sabbir Rahman. 2015. Urban Sensing and Smart Home Energy Optimisations: A Machine Learning Approach. In Proceedings of the 2015 International Workshop on Internet of Things towards Applications (IoT-App '15). Association for Computing Machinery, New York, NY, USA, 19–22. DOI:https://doi.org/10.1145/2820975.2820979.

[2] Bishwajit Dutta, Vignesh Adhinarayanan, and Wu-chun Feng. 2018. GPU power prediction via ensemble machine learning for DVFS space exploration. In Proceedings of the 15th ACM International Conference on Computing Frontiers (CF '18). Association for Computing Machinery, New York, NY, USA, 240–243. DOI:https://doi.org/10.1145/3203217.3203273

[3] Feinberg E.A., Genethliou D. (2005) Load Forecasting. In: Chow J.H., Wu F.F., Momoh J. (eds) Applied Mathematics for Restructured Electric Power Systems. Power Electronics and Power Systems. Springer, Boston, MA.

[4] Ghalehkhondabi, I., Ardjmand, E., Weckman, G.R. and Young, W.A., 2017. An overview of energy demand forecasting methods published in 2005–2015. Energy Systems, 8(2), pp.411-447.

[5] Alawadi, S., Mera, D., Fernández-Delgado, M., Alkhabbas, F., Olsson, C.M. and Davidsson, P., 2020. A comparison of machine learning algorithms for forecasting indoor temperature in smart buildings. Energy Systems, pp.1-17.

[6] Dodier, R. H., and Henze, G. P. (February 12, 2004). "Statistical Analysis of Neural Networks as Applied to Building Energy Prediction ." ASME. J. Sol. Energy Eng. February 2004; 126(1): 592–600.

[7] Ahmad, T., Chen, H., Huang, R., Yabin, G., Wang, J., Shair, J., Akram, H.M.A., Mohsan, S.A.H. and Kazim, M., 2018. Supervised based machine learning models for short, medium and long-term energy prediction in distinct building environment. Energy, 158, pp.17-32

[8] Basu, K., Debusschere, V. and Bacha, S., 2012, October. Appliance usage prediction using a time series based classification approach. In IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society (pp. 1217-1222). IEEE.

[9] Ahmad, T., Chen, H., Huang, R., Yabin, G., Wang, J., Shair, J., Akram, H.M.A., Mohsan, S.A.H. and Kazim, M., 2018. Supervised based machine learning models for short, medium and long-term energy predic*tion in distinct building environment. Energy, 158, pp.17-32.

[10]Dutta, Bishwajit & Adhinarayanan, Vignesh & Feng, Wu. (2018). GPU power prediction via ensemble machine learning for DVFS space exploration. 240-243. 10.1145/3203217.3203273.

[11]Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 12, null (2/1/2011), 2825–2830.

[12] Gavin Hackeling. 2014. Mastering Machine Learning With scikit-learn. Packt Publishing.

[13] Géron, A. (2017), 'Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligentsystems' , O'Reilly Media , Paperback .

[14] Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[15] Putatunda, S. and Rama, K., 2018, November. A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost. In Proceedings of the 2018 International Conference on Signal Processing and Machine Learning (pp. 6-10).

[16] Sarker, I.H., 2021. Machine learning: Algorithms, real-world applications and research directions. SN Computer Science, 2(3), pp.1-21.

[17] Azuaje, Francisco & Witten, Ian & E, Frank. (2006). Witten IH, Frank E: Data Mining: Practical Machine Learning Tools and Techniques. Biomedical Engineering Online - BIOMED ENG ONLINE. 5. 1-2.

[18] Sarker, I.H., Alqahtani, H., Alsolami, F., Khan, A.I., Abushark, Y.B. and Siddiqui, M.K., 2020. Context pre-modeling: an empirical analysis for classification based user-centric context-aware predictive modeling. Journal of Big Data, 7(1), pp.1-23.

[19] Ng, A., 2011. Advice for applying machine learning. In Machine learning.

[20] s: A. Zeng, H. Ho, Y. Yu, Prediction of building electricity usage using Gaussian Process Regress ion, Journal of Building Engineering (2019), doi: https://doi.org/10.1016/ j.jobe.2019.101054.

[21] Wang, R., Lu, S. and Feng, W., 2020. A novel improved model for building energy consumption prediction based on model integration. Applied Energy, 262, p.114561.

[22]Touzani, S., Granderson, J. and Fernandes, S., 2018. Gradient boosting machine for modeling the energy consumption of commercial buildings. Energy and Buildings, 158, pp.1533-1543.

[23] Ahmad T, Chen H, Nonlinear autoregressive and random forest approaches to forecasting electricity load for utility energy management systems, Sustainable Cities and Society (2018), https://doi.org/10.1016/j.scs.2018.12.013

[24] Chen, Y. and Tan, H., 2017. Short-term prediction of electric demand in building sector via hybrid support vector regression. Applied energy, 204, pp.1363-1374.

[25] Kangji Li, Chenglei Hu, Guohai Liu, Wenping Xue, Building's electricity consumption prediction using optimized artificial neural networks and principal component analysis, Energy & Buildings (2015), http://dx.doi.org/10.1016/j.enbuild.2015.09.002

[26] Geurts, Pierre & Ernst, Damien & Wehenkel, Louis. (2006). Extremely Randomized Trees. Machine Learning. 63. 3-42. 10.1007/s10994-006-6226-1.