

# **INTRODUCTION**

The Appliances Energy Prediction dataset includes variables related to home appliances, weather patterns, and other environmental elements. Its main goal is to predict a household's energy usage based on the input features. Each instance in the dataset represents 10 minutes, totaling more than 19,735 instances. The information was collected over 4.5 months. The primary objective is for Machine Learning research to create and evaluate models for energy prediction tasks that can help optimize energy utilization.

## **ABSTRACT**

Appliance energy prediction is critical in energy management systems for efficient energy utilization, resulting in cost savings using Machine Learning Algorithms (Linear Regression, Ridge Regression, Lasso Regression, and Neural Networks). It involves predicting the energy consumption of appliances based on various input features such as temperature, humidity, time of day, and past usage patterns. Appliance energy prediction focuses on reducing energy consumption, greenhouse gas emissions, and overall energy costs.

## **PROBLEM STATEMENT**

The problem of appliance energy prediction arises due to the growing need to reduce household energy consumption. Machine learning algorithms offer a promising approach to address this problem and the ability to identify relevant features and select appropriate algorithms.

## **AIM**

Appliance energy prediction aims to develop machine learning models that accurately estimate energy consumption based on historical data and external factors. Machine learning algorithms must be trained on large datasets of appliance usage data. The resulting models should be able to predict energy consumption to provide actionable insights accurately.

## DATA DESCRIPTION

The dataset was collected from the UCI Machine Learning Repository.

Link for the dataset: [Appliance Energy Prediction Dataset](#)

**Number of Instances = 19735**

**Number of Attributes = 29**

date time year-month-day hour:minute:second  
Appliances, energy use in Wh  
lights, energy use of light fixtures in the house in Wh  
T1, Temperature in kitchen area, in Celsius  
RH\_1, Humidity in kitchen area, in %  
T2, Temperature in living room area, in Celsius  
RH\_2, Humidity in living room area, in %  
T3, Temperature in laundry room area  
RH\_3, Humidity in laundry room area, in %  
T4, Temperature in office room, in Celsius  
RH\_4, Humidity in office room, in %  
T5, Temperature in bathroom, in Celsius  
RH\_5, Humidity in bathroom, in %  
T6, Temperature outside the building (north side), in Celsius  
RH\_6, Humidity outside the building (north side), in %  
T7, Temperature in ironing room , in Celsius  
RH\_7, Humidity in ironing room, in %  
T8, Temperature in teenager room 2, in Celsius  
RH\_8, Humidity in teenager room 2, in %  
T9, Temperature in parents room, in Celsius  
RH\_9, Humidity in parents room, in %  
To, Temperature outside (from Chievres weather station), in Celsius  
Pressure (from Chievres weather station), in mm Hg  
RH\_out, Humidity outside (from Chievres weather station), in %  
Wind speed (from Chievres weather station), in m/s  
Visibility (from Chievres weather station), in km  
Tdewpoint (from Chievres weather station), Å°C  
rv1, Random variable 1, nondimensional  
rv2, Random variable 2, nondimensional

# EXPLORATORY DATA ANALYSIS

## NULL VALUE CHECK

Checking for null values in the dataset for all the variables.

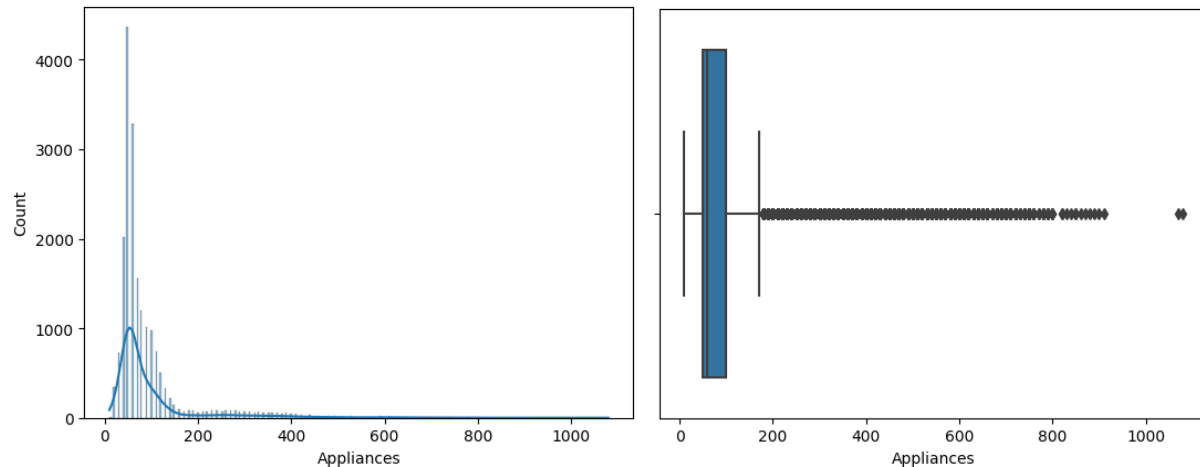
The dataset had no null values. Hence no treating or imputation was required

```
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
```

## TARGET VARIABLE

Appliances are the target variable in our dataset. The variable we are attempting to estimate or predict using data from other variables in the dataset is the target variable in machine learning. It goes by the names dependent variable, the response variable, and the outcome variable.

The appliance variable gives us the energy consumption of a household in Wh.



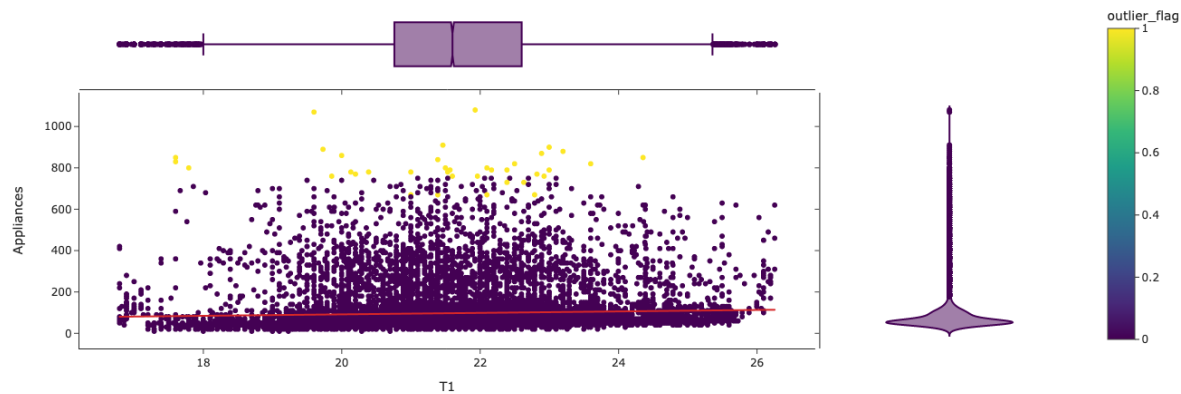
The data is skewed concentration towards the lower end of the Appliances value, as seen in the distribution above. Based on the position of the median value in the box plot, we may decide whether a distribution is skewed. When the median is closer to the bottom of the box and the whisker is shorter on the lower end of the box, the distribution is right-skewed (or "positively skewed").

## OUTLIER DETECTION

### OUTLIER DETECTION ON TARGET VARIABLE

#### DBSCAN

Data points closely spaced together are grouped using the clustering algorithm DBSCAN, which also finds isolated noise spots. A radius surrounds each data point, and the number of additional data points within that radius is counted. Core points are those when the number of points inside the radius is greater than or equal to a predetermined threshold. Border points are those points that are a part of the radius of core points but are not core themselves. Noise points are neither core nor border points nor fall within any core point's radius.



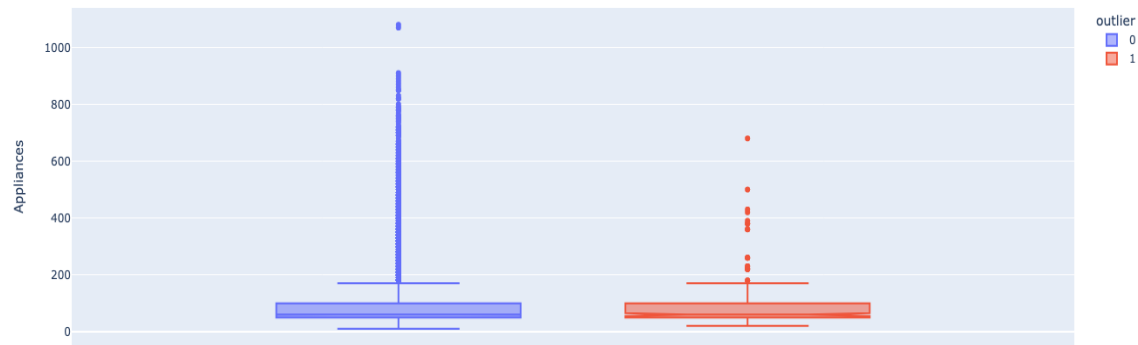
The points in yellow were flagged as outliers by the DBSCAN algorithm. These points were plotted in a scatterplot for the Appliance vs. T1 variable. The outliers were not removed as there was no conclusive evidence that these data points were erroneous. Even though outliers can occasionally be the consequence of mistakes, they can occasionally hold vital information and offer insights into the underlying patterns and relationships in the data. Forty-three data points were flagged as outliers from the DBSCAN algorithm. These values with high target variables or outliers could be electrical surges and cannot be removed from the data.

## OUTLIER DETECTION ON INPUT VARIABLES

### LOCAL OUTLIER FACTOR

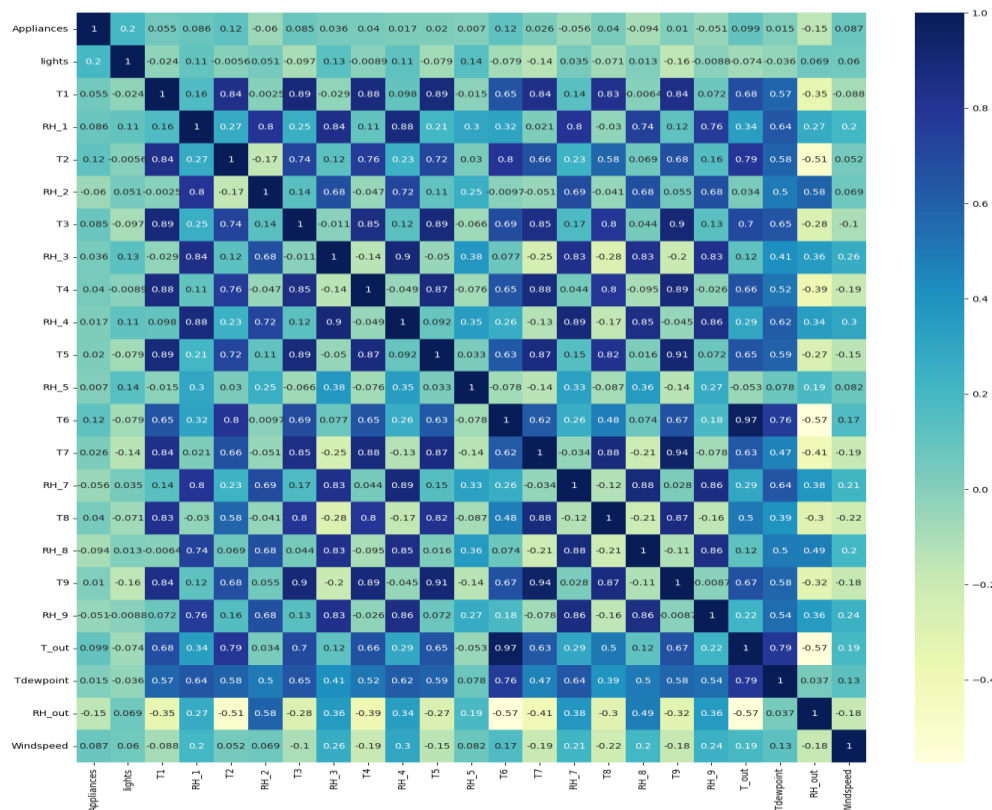
LOF is an unsupervised machine learning algorithm that detects outliers in a dataset. Each data point is given a score based on its local density in relation to its  $k$  closest neighbors. In contrast to a data point close to its neighbors, a data point far from its neighbors will have a low local density and be more likely to be an outlier. LOF exceptionally performs well with high dimensional data and is robust to data density variations and clusters or noise. However, if the dataset is too large, it can be computationally expensive.

Energy usage for regular data points and the data points with outliers



The boxplot above shows the distribution of the outliers identified from the input variables. These data points, however, have very similar distribution for the target variable without outliers. Both the regular data points and the outliers have a median of 60. The potential outlier values were not removed because there was no significant outlier behavior concerning the target variable.

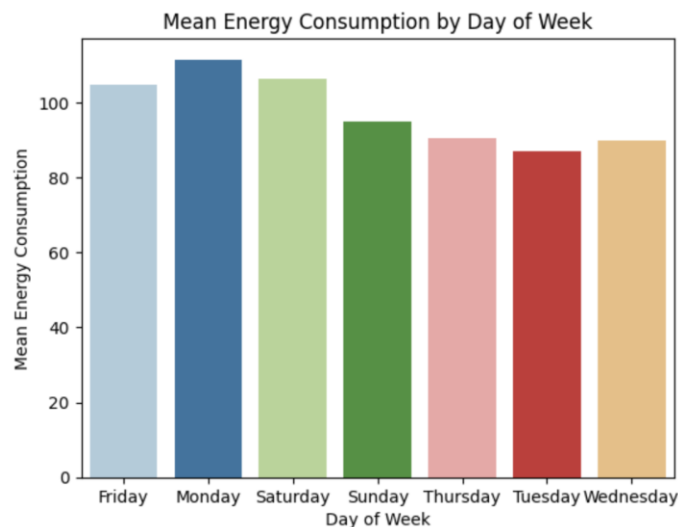
## INPUT VARIABLE CORRELATION



## CORRELATION PLOT

The correlation coefficients between several variables are shown on a correlation plot, often referred to as a heatmap or a correlation matrix. It is an effective technique for finding trends and connections between variables in sizable datasets. The Pearson correlation coefficient is used for the analysis. It measures the strength of the linear link between two continuous variables. On a scale from -1 to +1, it assesses the magnitude and direction of the link between two variables. There is no correlation when the correlation coefficient is zero, a perfect negative correlation is shown by a correlation coefficient of one, and a perfect positive correlation is indicated by a correlation coefficient of one. The chart shows that the temperature variables (T1, T2, T3..) are highly correlated as they represent the temperature in different house rooms. Similarly, the humidity variables (RH\_1, RH\_2, RH\_3..) are highly correlated. Hence these highly correlated variables are removed as a part of feature selection to avoid multicollinearity.

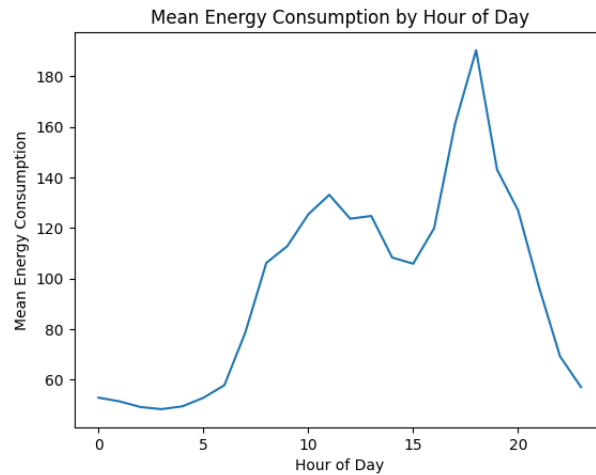
## MEAN ENERGY CONSUMPTION BY DAY OF WEEK



During our analysis of appliance energy consumption, we observed a noticeable difference in the mean energy consumption across different days of the week. To illustrate this finding, we generated a bar plot displaying each day's mean energy consumption. The plot reveals that the mean energy consumption is higher on Fridays, Saturdays, and Sundays than on other weekdays.

(Monday through Thursday). Specifically, we noticed that energy consumption on weekends tends to be higher on average than on weekdays.

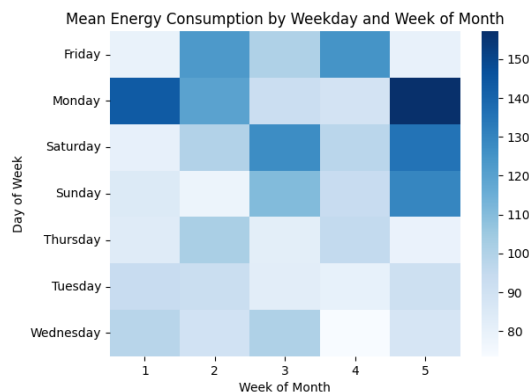
### MEAN ENERGY CONSUMPTION BY HOUR OF DAY



To further understand the daily pattern of appliance energy consumption, we analyzed the data by grouping it by the hour of the day. We then calculated the mean energy consumption for each hour and created a line plot to visualize the results.

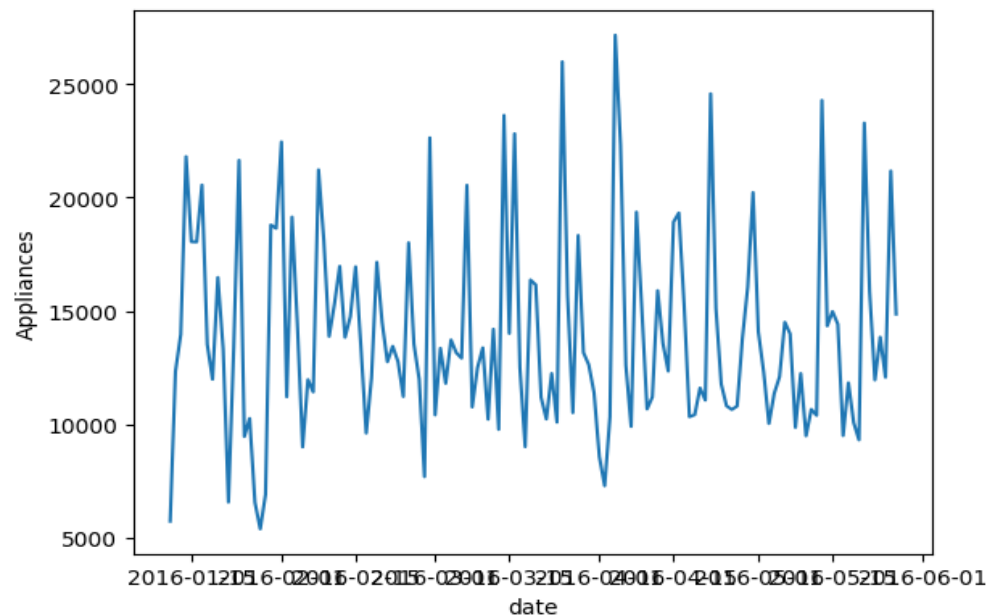
The plot shows an apparent increase in energy consumption during the evening, peaking around 7 PM. This pattern suggests that people's daily routines and activities significantly influence energy usage.

### MEAN ENERGY CONSUMPTION BY WEEKDAY AND WEEK OF MONTH





The power consumption was significantly lower midweek, irrespective of the week of the month. Whereas the weekend had ups and down through the weeks.



Visualizing the energy variations throughout 4.5 months. Periodic surges and declines were seen in the data

## DATA PREPROCESSING

### FEATURE ENGINEERING

#### RFE LOGISTIC REGRESSION

Recursive Feature Elimination (RFE) is a robust feature selection algorithm to select the best set of features for a machine learning model. RFE aims to reduce the number of features until the required number is obtained by removing the least significant features from a given set of features. RFE was used with Logistic Regression to get the best 20 features in our case.

RFE is an algorithm that selects the most important features for a given task by recursively training an estimator and removing the least important feature until the desired number of features is reached.

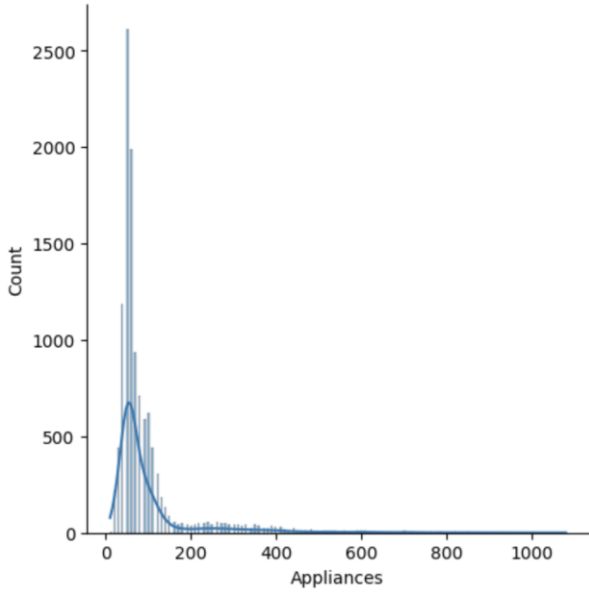
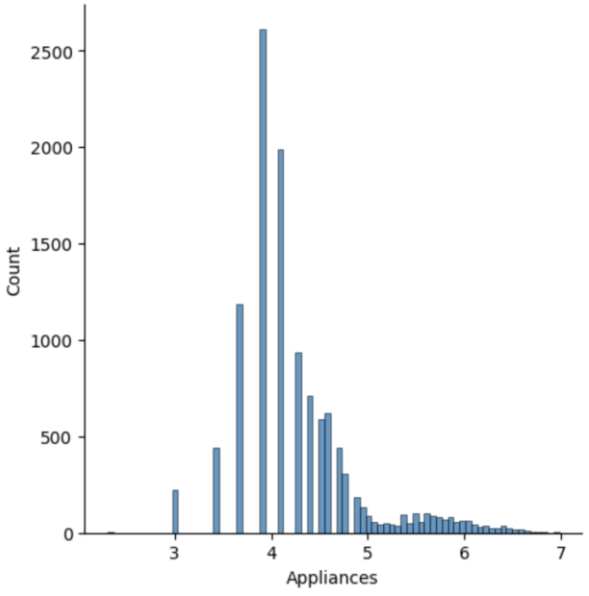
FEATURES SELECTED FROM RFE

```
Index(['lights', 'T1', 'RH_1', 'T2', 'RH_2', 'T3', 'RH_3', 'T4', 'RH_4', 'T5',  
      'T6', 'T7', 'RH_7', 'T8', 'RH_8', 'T9', 'RH_9', 'T_out', 'Tdewpoint',  
      'Windspeed'],  
      dtype='object')
```

Out of the selected variables, variables with collinearity have been removed to avoid multicollinearity

DATA NORMALIZATION

Data normalization is the process of converting numerical data into a standard format to eliminate repetition and inconsistencies in the data, known as data normalization. Data normalization seeks to enhance data integrity and eliminate data redundancy. It was previously observed that the data for the target variable is right skewed. The skewness can be reduced by applying log normalization

	
Before log transformation	After log transformation
Skewness for Appliances = 3.4381770799767053	Skewness for Appliances = 1.1488229881452883

## ENCODING

`get_dummies` creates a new dataframe with dummy columns from 'Monday' to 'Sunday', resulting in a binary indicator showing whether the day of the week is present.

```
#encoding the day_of_week column
weekdays_encoded = pd.get_dummies(data.day_of_week)
#run the below line only once as concat keeps adding more columns everytime it's run
data = pd.concat([data, weekdays_encoded], axis=1)
data = data.drop('day_of_week', axis=1)
data
```

## TEST TRAIN SPLIT

The testing set assesses the model's performance on data it has never seen before, whereas the training set is used to train the model. As any machine learning technique's ultimate objective is to estimate how well an algorithm will perform on new data.

The dataset is split into 3 parts: train, test, and validate with sizes 60%, 20%, and 20%, respectively

## FEATURE SCALING

Feature scaling is normalizing or standardizing the dataset so that all the variables are on a similar scale. It entails converting feature values to a common scale with comparable magnitudes and ranges. This is crucial because many machine learning algorithms compare data points dependent on their distance, and features with higher values might dominate and skew the findings.

A standard scaler is being used to scale the values. The process involves making the mean 0 and the standard deviation 1. The data points are subtracted by their mean and then divided by the standard deviation.

## PRINCIPAL COMPONENTS ANALYSIS (PCA)

PCA is a dimensionality reduction used in machine learning. PCA is beneficial when the number of input variables is very high while training an ML model. The fundamental goal of PCA is finding the linear combinations of the original variables that best capture the range of the dataset's variance. The direction in the data that captures the most variation is represented by the first principal component, and each following component captures the remaining variation present.

The number of principal components was chosen so that the variables capture more than 95% of the variance in the dataset. Upon analysis, it was found that 9 principle components would capture the required variance.

```
# Find the number of components that explain 95% of the variance
n_components = np.argmax(cumulative_variance >= 0.95) + 1

# Print the number of components
print("Number of components to keep:", n_components)

# Create a new PCA object with the desired number of components
pca_transformer = PCA(n_components=n_components)
```

Number of components to keep: 9

Singular value decomposition for PCA is used to reduce the number of variables from 27 to 9

	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
0	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	45.566667	17.166667	55.20	7.026667	84.256667	17.200000	41.626667	18.2	48.900000
1	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	45.992500	17.166667	55.20	6.833333	84.063333	17.200000	41.560000	18.2	48.863333
2	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	45.890000	17.166667	55.09	6.560000	83.156667	17.200000	41.433333	18.2	48.730000
3	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	45.723333	17.166667	55.09	6.433333	83.423333	17.133333	41.290000	18.1	48.590000
4	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	45.530000	17.200000	55.09	6.366667	84.893333	17.200000	41.230000	18.1	48.590000

Input dataframe before scaling and PCA

	0	1	2	3	4	5	6	7	8
0	1.408017	-2.147457	-0.031982	1.207412	-2.390233	-1.189787	0.642336	0.078394	-1.843586
1	6.329950	-0.285269	-1.602949	1.351615	1.034210	-1.374462	-0.663433	0.471408	1.907273
2	-3.995582	0.309158	1.897239	0.064220	-1.525201	-0.954149	0.520891	0.681419	-0.008062
3	8.211530	3.925952	2.279858	0.782828	1.952691	-0.629898	1.301788	0.429977	-0.635332
4	1.254552	-3.345780	1.122456	-1.682227	0.609123	1.779095	-0.025160	0.778885	-0.042839

Input dataframe after scaling and PCA

In the above image, variable 0 is the first principal component and captures the highest variance, followed by variables 1,2,3, etc.  $\text{Variance}(0) > \text{Variance}(1) > \text{Variance}(2)$  and so on

## MODEL IMPLEMENTATION

### LINEAR REGRESSION

Linear Regression is a statistical technique employed to establish the relationship between a dependent variable and one or more independent variables. The dependent variable represents the outcome to be predicted, while the independent variables serve as predictors for the dependent variable.

The primary objective of linear Regression is to identify the most suitable linear equation that accurately represents the data. Such a linear equation takes  $y = ax + b$ , where  $y$  denotes the dependent variable,  $x$  symbolizes the independent variable,  $a$  signifies the line's slope, and  $b$  corresponds to the y-intercept.

### Gradient Descent

The cost function of Linear Regression is given by:

$$\theta_j := \theta_j - \alpha (h_{\theta}(x^{(i)}) - y^{(i)})x$$

Where:

$\theta_j$  is the  $j$ -th model parameter (weight).

$\alpha$  is the learning rate, which controls the step size of the updates

$h_{\theta}(x)$  is the hypothesis function, which is a linear function of the features:  $h_{\theta}(x) = \theta^T x$

$x^{(i)}$  is the feature vector of the  $i$ -th training example

$y^{(i)}$  is the target value of the  $i$ -th training example

By deducting the product of the learning rate and the difference between the predicted and actual value for a single training example, multiplied by the  $j^{\text{th}}$  characteristic of that training example, the update algorithm determines the new value of the weights.

Iteratively updating the model parameters until convergence, this procedure is repeated for each training example in random order to minimize the cost function, sometimes the mean squared error between the predicted and actual values.

### Stochastic Gradient Descent

A model's parameters are updated using the stochastic gradient descent (SGD) optimization algorithm based on the cost function's gradient for a single training example at a time

The cost function of Stochastic Linear Regression is given by:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x)(x^{(i)}) - y^{(i)})x^{(i)}$$

Where,

- $\theta_j$  is the  $j$ -th model parameter (weight)
- $\alpha$  is the learning rate, which controls the step size of the updates
- $m$  is the number of training examples
- $h_{\theta}(x)$  is the hypothesis function, which is a linear function of the features:  $h_{\theta}(x) = \theta^T x$
- $x^{(i)}$  is the feature vector of the  $i$ -th training example
- $y^{(i)}$  is the target value of the  $i$ -th training example

The method to update the weights computes the new value by deducting the product of the learning rate, the mean difference between predicted and actual values for each training example, and the  $j^{\text{th}}$  feature of the  $i^{\text{th}}$  training example. This calculation is divided by the number of training examples  $m$ . The process is repeated until convergence to minimize the cost function  $J(\text{weight})$ , which typically refers to the mean squared error between the predicted and actual values.

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Gradient Descent	0.0001	0.001	139.9	0.39
1	Gradient Descent	0.001	0.01	139.7	0.38
2	Gradient Descent	0.01	0.1	69.01	1.70
3	Stochastic Gradient Descent	0.0001	0.001	134.3	0.46
4	Stochastic Gradient Descent	0.001	0.01	99.01	1.03
5	Stochastic Gradient Descent	0.01	0.1	72.63	1.61

Linear regression results before PCA

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Gradient Descent	0.0001	0.001	139.9	0.39
1	Gradient Descent	0.001	0.01	139.8	0.3
2	Gradient Descent	0.01	0.1	97.86	1.06
3	Stochastic Gradient Descent	0.0001	0.001	135.6	0.44
4	Stochastic Gradient Descent	0.001	0.01	111.2	0.80
5	Stochastic Gradient Descent	0.01	0.1	94.3	1.13

Linear regression results after PCA

## RIDGE REGRESSION

Ridge regression, alternatively referred to as L2 regularization or Tikhonov regularization, is a method employed to tackle the issue of overfitting in linear regression models. Overfitting arises when a model captures the noise in the training data, leading to suboptimal performance when applied to new, unseen data. By incorporating a regularization term into the linear Regression objective function, ridge regression penalizes large coefficients, aiding in overfitting prevention.

### Gradient Descent

Ridge regression algorithm employs a mean squared error loss function while integrating L2 regularization in its methodology.

The cost function of Ridge Regression is given by:

$$J(w) = \frac{1}{2m} \left[ \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \sum_{j=1}^n w_j^2 \right]$$

Where,

$m$  is the number of training examples

$y^{(i)}$  is the actual output of the  $i$ -th training example

$x^{(i)}$  is the feature vector of the  $i$ -th training example

$w$  is the vector of model parameters

$\lambda$  is the regularization parameter

The update rule for Ridge Regression with Gradient Descent is:

$$w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} w - y^{(i)}) x_j^{(i)} + 2\lambda w_j \right]$$

This update rule gets used repeatedly until convergence, that is, until the change in the cost function becomes insignificant compared to the tolerance value or until a maximum number of iterations has been reached.

### Stochastic Gradient Descent

SGD is an optimization technique that changes the model parameters based on the gradient of the cost function with respect to a single training example at a time. The cost function has a regularization term to avoid overfitting in Ridge Regression.

Ridge Regression's cost function is given by:

$$J(w) = \frac{1}{2m} \left[ \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \sum_{j=1}^n w_j^2 \right]$$

The variables here are the same as above.



The update rule for Ridge Regression with Stochastic Gradient Descent is:

$$w_j = w_j - \alpha \left[ \frac{1}{2} (x_j^{(i)} w - y^{(i)}) x_j^{(i)} + \lambda w_j \right]$$

For each SGD iteration in Ridge Regression, a training example is chosen randomly from the dataset, and the model parameters are updated using the gradient of the cost function for that training example.

## RESULTS

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Gradient Descent	0.0001	0.001	139.6	0.40
1	Gradient Descent	0.001	0.01	133.04	0.47
2	Gradient Descent	0.01	0.1	94.06	1.13
3	Stochastic Gradient Descent	0.0001	0.001	139.65	0.40
4	Stochastic Gradient Descent	0.001	0.01	133.04	0.47
5	Stochastic Gradient Descent	0.01	0.1	94.06	1.13

Ridge regression results before PCA

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Gradient Descent	0.0001	0.001	139.4	0.40
1	Gradient Descent	0.001	0.01	134.89	0.45
2	Gradient Descent	0.01	0.1	108.52	0.85
3	Stochastic Gradient Descent	0.0001	0.001	139.46	0.40
4	Stochastic Gradient Descent	0.001	0.01	135.05	0.45
5	Stochastic Gradient Descent	0.01	0.1	109.27	0.84

Ridge regression results after PCA

## LASSO REGRESSION

Lasso regression, sometimes referred to as the least absolute shrinkage and selection operator, is a form of linear Regression that employs shrinkage. This technique minimizes the magnitude of the coefficients within a model, enhancing its interpretability and generalization to fresh data. Lasso regression drives the coefficients of insignificant variables to zero, effectively excluding them from the model. This is particularly beneficial when dealing with numerous variables, as it helps to mitigate overfitting.

### Gradient Descent

The Lasso regression algorithm employs the same mean squared error loss function as Ridge and integrates L1 regularization into its methodology. Consequently, it follows the same sequence of steps as Ridge. The cost function for Lasso Regression, denoted as  $J(\theta)$ , is expressed as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y - \hat{y})^2 + \lambda \sum_{j=1}^n |w_j|$$

Where,

$\lambda \sum_{j=1}^n |w_j|$  is the penalty (L1 Regularization).

m = number of training samples

y = actual output

y\_hat = prediction term

$\lambda$  = regularization parameter

wj = coefficients of the features

### Stochastic Gradient Descent

$$J(w, b) = (1/2) * (y_i - (w^T x_i + b))^2 + \lambda * \|w\|_1$$

where,

$(1/2)*(y_i - (w^T x_i + b))^2$  is the squared error term for the  $i$ th training sample.

$\lambda$  is the regularization parameter that controls the amount of L1 regularization (non-negative)

$\|w\|_1$  is the L1 norm of the weight or theta vector  $w$ .

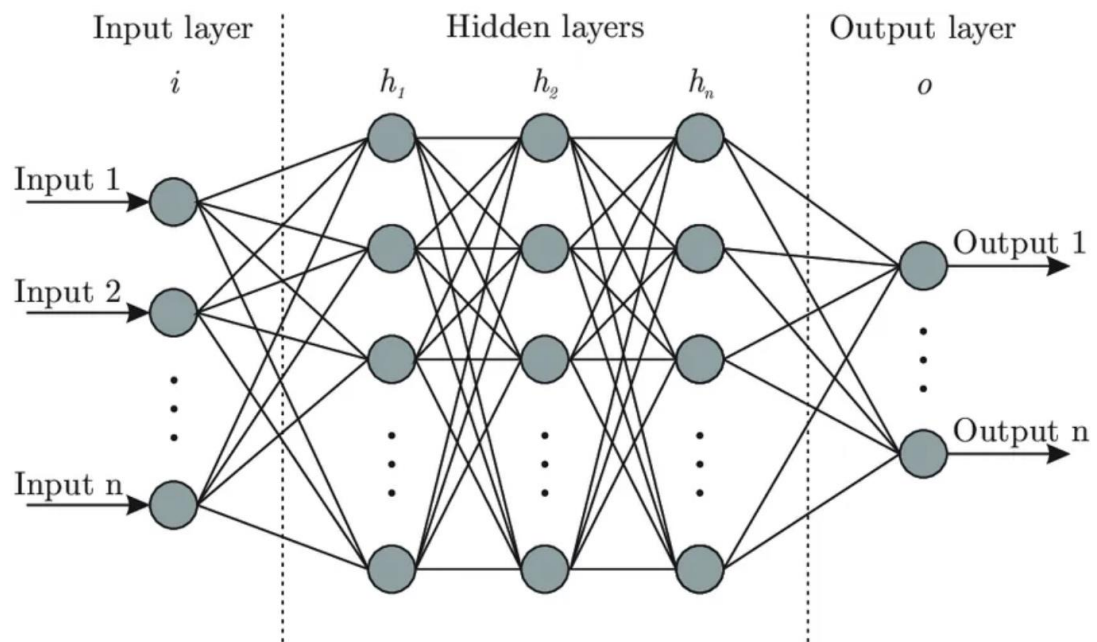
	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Gradient Descent	0.0001	0.001	138.7	0.40
1	Gradient Descent	0.001	0.01	129.02	0.52
2	Gradient Descent	0.01	0.1	79.6	1.45
3	Stochastic Gradient Descent	0.0001	0.001	138.7	0.41
4	Stochastic Gradient Descent	0.001	0.01	129.4	0.52
5	Stochastic Gradient Descent	0.01	0.1	79.82	1.45

Lasso regression results before PCA

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Gradient Descent	0.0001	0.001	139.08	0.40
1	Gradient Descent	0.001	0.01	131.83	0.49
2	Gradient Descent	0.01	0.1	99.84	1.02
3	Stochastic Gradient Descent	0.0001	0.001	139.08	0.40
4	Stochastic Gradient Descent	0.001	0.01	131.89	0.49
5	Stochastic Gradient Descent	0.01	0.1	99.93	1.01

Lasso regression results after PCA

## NEURAL NETWORK



```
# Define the model
model = Sequential()
model.add(Dense(64, input_shape=(25,), activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='linear'))

# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')
```

The first layer of this neural network is a dense layer with an activation function of 'ReLU' with 64 neurons, with input shape = 25.

The second layer is a dropout layer with a dropout rate of 0.2 that randomly drops out a fraction of neurons which helps to prevent overfitting.

The third layer is a dense layer with an activation function of 'relu' with 32 neurons.

The fourth layer is a dense layer with an activation function of 'linear.'

Neural networks have below components:

### ReLU Activation Function

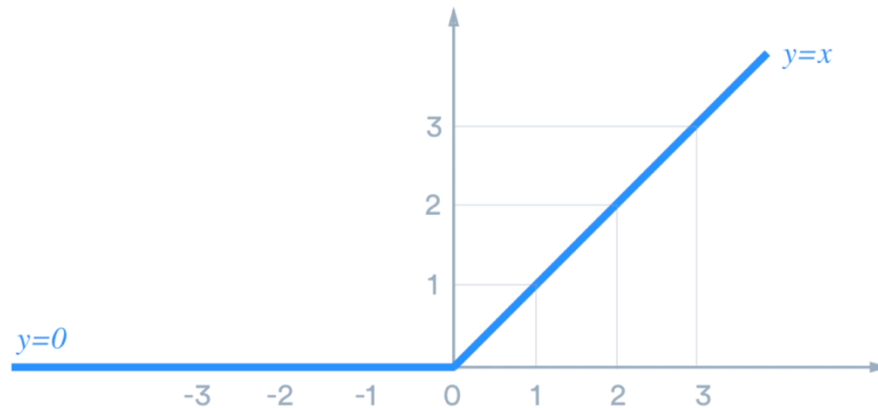
ReLU is the most used activation function in machine learning models.

The below function defines it:

$$f(x) = \max(0, a) = \max(0, \sum_{i=1}^n w_i x_i + b)$$

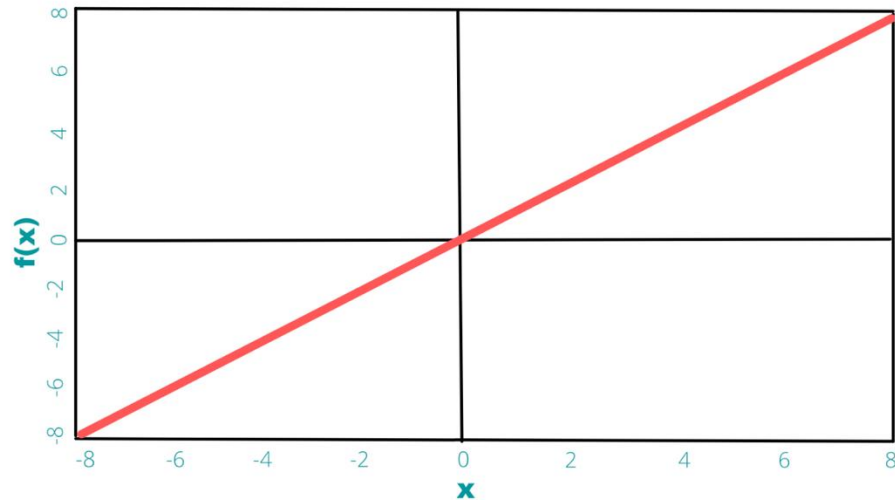
The derivative of the ReLU function:

$$f'(x) = \frac{\partial f(x)}{\partial x} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$



If x is positive, then the output is equal to x. If x is negative, the output is equal to 0.

## Linear Activation Function



$$f(x) = wx + b$$

where  $x$  is the input vector,  $w$  is the weight vector,  $b$  is the bias term, and  $f(x)$  is the output.

The most common way to construct a linear function in a neural network is as a dense layer that is completely connected, where each input is multiplied by a weight and then added together with a bias term.

## Loss Function

Mean Squared Error is commonly used as a loss function used in regression problems. Between the expected and actual values, it calculates the average squared difference.

```
R2: 0.96  
RMSE: 2.99  
SSE: 177022.95
```

Neural networks result before PCA

```
R2: 0.40  
RMSE: 79.73  
SSE: 125462280.20
```

Neural networks result after PCA

## BIAS VARIANCE TRADEOFF

Bias-variance tradeoff describes the tradeoff between a model's capacity to fit the training data and its capacity to generalize to new, unobserved data. A model with high variance is likely to overfit the data, which means that it fits the training data too closely and does not generalize well to new data, while a model with high bias is likely to underfit the data, failing to capture the true relationship between the features and the target variable. Results for the best-performing hyperparameters for each algorithm are given below

Without PCA:

Linear Regression: bias- 11105.64 variance-4225.60

Ridge regression: bias- 13788.14 variance-1466.68

Lasso regression: bias- 12091.05 variance-2684.00

With PCA:

Linear Regression: bias- 11774.64 variance-1161.31

Ridge regression: bias- 13859.99 variance-683.22

Lasso regression: bias- 12152.19 variance-1057.72

The bias-variance tradeoff aims to find the sweet spot between bias and variance. Since the values of the models are very close, a conclusion cannot be made about the model performance solely based on this. Additional accuracy metrics should be considered to conclude the best-performing model.

## RESULTS SUMMARY

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Linear Regression (Gradient Descent)	0.0001	0.001	139.9	0.39
1	Linear Regression (Stochastic Gradient Descent)	0.0001	0.001	134.3	0.46
2	Ridge Regression (Gradient Descent)	0.001	0.01	133.04	0.47
3	Ridge Regression (Stochastic Gradient Descent)	0.001	0.01	133.04	0.47
4	Lasso Regression (Gradient Descent)	0.001	0.01	129.02	0.52
5	Lasso Regression (Stochastic Gradient Descent)	0.001	0.01	129.4	0.52
6	Neural Nets	-	-	2.99	0.96

Before PCA

	Approach	Learning Rate	Lambda Value	RMSE	R <sup>2</sup>
0	Linear Regression (Gradient Descent)	0.0001	0.001	139.9	0.39
1	Linear Regression (Stochastic Gradient Descent)	0.001	0.01	111.2	0.80
2	Ridge Regression (Gradient Descent)	0.01	0.1	108.52	0.85
3	Ridge Regression (Stochastic Gradient Descent)	0.01	0.1	109.27	0.84
4	Lasso Regression (Gradient Descent)	0.001	0.01	131.83	0.49
5	Lasso Regression (Stochastic Gradient Descent)	0.001	0.01	131.89	0.49
6	Neural Nets	-	-	79.73	0.40

After PCA

## CONCLUSION

### FINDINGS

- While using the gradient descent and stochastic gradient approaches for the Regression models, LASSO did the best, followed by Ridge and Linear
- The Regression models did better after applying PCA than before applying PCA
- PCA dropped the number of features to 9 principal components, which captured more than 95% variance. This brought down the number of features from the original 29
- Neural Networks had the best performance overall before applying PCA
- Neural Networks performed significantly worse after applying PCA. It had the worst performance among all models after PCA
- LASSO had the same performance for both stochastic and normal gradient descent approaches

### FUTURE SCOPE

- Check other evaluation metrics like Adjusted R<sup>2</sup> since R<sup>2</sup> performance degrades as the number of features increases



- Approaching the problem statement with Classification algorithms and comparing how well it did with Regression
- Exploring this as a Time Series problem to forecast energy usage based on prior trends.
- Models like ARIMA or Time Series Decomposition Algorithms could be used to predict energy Usage.