Study of Regression and classification models for energy consumption prediction and outlier detection

ABSTRACT:

The goal of this project is to predict the energy consumption of appliances using regression models and to conduct an outlier analysis using classification techniques. The energy consumption data is obtained from a house that contains various appliances, and the data includes information such as the date and time of usage, weather conditions, and appliance usage. To predict energy consumption, we used a combination of regression models with Stochastic Gradient Descent (SGD) with and without Principal Component Analysis (PCA) and regularization. Additionally, we performed feature engineering and selection to optimize the performance of the regression models. For outlier classification, we used Logistic Regression, Gaussian Naïve Bayes, and Neural Networks with K-fold cross-validation and average classifier. We also created a stacked ensemble model for outlier classification. The models were evaluated using performance metrics such as accuracy, precision, recall, and F1 score. Our study provides valuable insights for energy conservation and cost-saving measures, and future work could explore the use of additional features or data sources to improve the model performance.

INTRODUCTION:

The importance of energy conservation motivated us to explore the problem of predicting energy consumption. Our goal is to use regression models to predict and identify outliers that may be obstructive to prediction using classification techniques. We conducted this project as part of our master's program in Data Analytics for the Statistical Learning course.

We used a combination of regression models with Stochastic Gradient Descent (SGD) for prediction. Specifically, we used two versions of the model: one with PCA dimensionality reduction and another without. We also applied regularization techniques to optimize the performance of the models.

For outlier detection, we used three classification models: Logistic Regression, Gaussian Naïve Bayes, and Neural Networks. We applied K-fold cross-validation and an average classifier on the Neural Network to enhance its performance. Additionally, we created a stacked ensemble model using the three classifiers to improve the detection of outliers.

Overall, our approach involved a combination of regression and classification techniques to predict energy consumption and detect outliers accurately. The performance of the models was evaluated using metrics such as Root mean squared error (RMSE), accuracy, precision, recall, and F1 score.

The report is structured as follows: the Data Description section provides an overview of the dataset used in the project, the Methods section describes the regression and classification techniques used in detail, the EDA section details the feature engineering and selection process, and the Results section presents the performance metrics and analysis of the models. Finally, the Conclusion section summarizes the findings and implications of the study.

DATA DESCRIPTION:

The dataset used in this project is the "Appliances energy prediction" dataset, which is publicly available from the UCI Machine Learning Repository (dataset - link). The dataset contains energy consumption information from a house in Belgium, over a period of four and a half months collected at 10-minute intervals. The features include information about temperature and humidity in different rooms of the house and the outside temperature taken from the nearest airport and wind-speed in m/s, visibility in km, and T-dew point in Â°C.

The raw dataset consists of 19736 records and 29 features, data type is int64 for every column except for data column. This is converted into 19736 records and 24 features including the bias for the regression. For classification the raw dataset is converted into 33535 records and 24 features for classification based on the following data preprocessing steps,

- Features “Appliances” and “Lights” were combined using sum function to form a new feature “usage” which is the target variable for regression.
- The target variable for prediction is the combined feature of appliance’s energy usage in Wh (Watt Hour) and light fixture’s energy usage in Wh (Watt Hour).
- Unwanted features such as “lights”, “date”, “rv1”, “rv2” were dropped.
- Dataset was separated as X, y and covered into NumPy arrays for regression and split into test and train datasets using Sci-kit learn for classification.
- For outlier classification, outliers were considered class 1 and non-outliers as class 0. A new feature for outliers was created using the Z-score method with threshold of 2 standard deviation.

However, these outliers contributed to a very small proportion of dataset, which is not ideal to run machine learning models for classification. So, we performed over-sampling of class 1 to make the outliers proportional to the non-outliers in the training dataset.

Assumptions: We assumed that the temperature and humidity in each room are independent of each other. This assumption helped us to simplify the model and focus on predicting energy consumption.

METHODS:

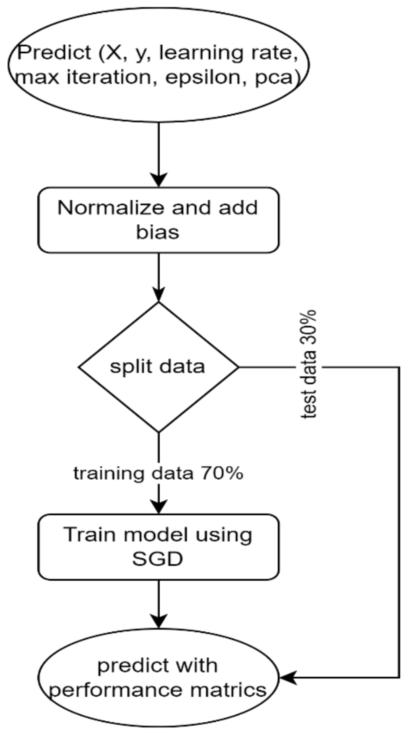
We employed various machine learning algorithms to tackle prediction and classification problems. In this section, we will delve into the specifics of each algorithm.

Linear regression is a supervised learning algorithm used for predicting a continuous target variable based on one or more input features. It assumes a linear relationship between the independent variables and the dependent variable and tries to find the best-fit line that can minimize the sum of squared errors between the predicted and actual values.

Implementation of Linear Regression:

The regression implementation is a linear regression class that is performed using Stochastic Gradient Descent (SGD). The input parameters include,

- the learning rate
- maximum iterations
- epsilon
- regularization parameters


Predict (X, y, learning rate, max iteration, epsilon, pca)
Normalize and add bias
split data
test data 30%
training data 70%
Train model using SGD
predict with performance matrics

The linear regression class normalizes the data, adds a bias term, and splits the dataset into train and test sets. The fit() method trains the model and returns the weights, RMSE, and SSE. It allows the user to choose whether or not PCA or regularization should be used for regression. The linear regression class automatically captures six components since the default variance captured is set to 95%.

Since we found during EDA our dataset is highly correlated, we decided to do PCA to reduce collinearity. Principal Component Analysis (PCA) can be used to reduce the dimensionality of the input features and improve the performance of the linear regression model.

Implementation of PCA:

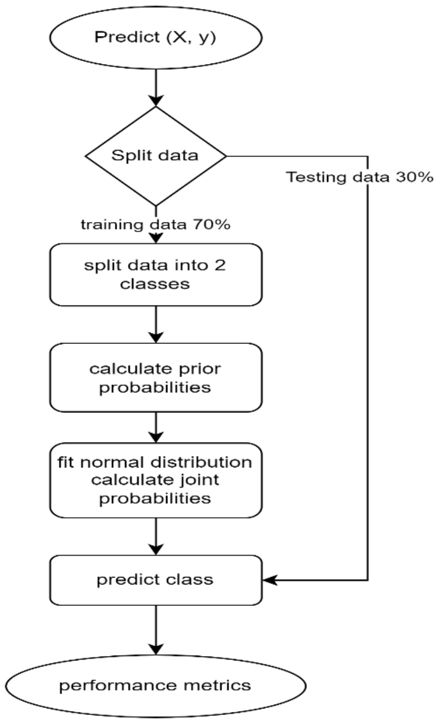
The PCA implementation is a Python function that takes in a 2D NumPy array X and another parameter that specifies the required variance percentage. The output includes the transform matrix, eigenvalue, eigenvector, explained variance, and the covariance matrix.

Outlier Classification:

Gaussian Naive Bayes is a probabilistic classification algorithm that is based on Bayes' theorem. It is used for binary classification tasks where the input features are continuous variables and assumed to be normally distributed and independent. In Gaussian Naive Bayes, the probability of an input belonging to a class is calculated by multiplying the prior probability of the class with the conditional probability of the input given that it belongs to that class.

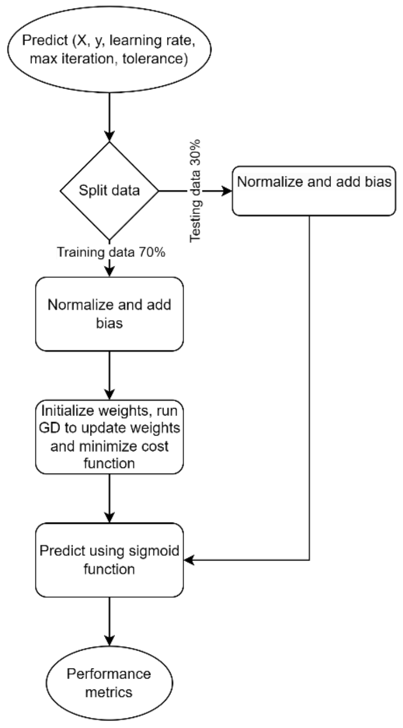
Implementation of Gaussian Naïve Bayes:

Our Gaussian Naive Bayes model is a class that takes in inputs X and y as 2D numpy arrays. This class auto-splits the data and calculates the joint probability of occurrence of X and y. It creates the following attributes


Predict (X, y)
Split data
Testing data 30%
training data 70%
split data into 2 classes
calculate prior probabilities
fit normal distribution calculate joint probabilities
predict class
performance metrics

which can be referenced outside of the class: y_pred for the test split, and y_0 and y_1 probability. It can do the same thing for the entire dataset. It also includes a method that prints the performance matrix of this model of the test dataset.

Logistic Regression is a binary classification algorithm that predicts the probability of a dependent variable. It estimates the probability of an event using the logistic function, which maps real-valued input to a value between 0 and 1. The model is trained with maximum likelihood estimation and optimized using gradient descent.


Predict (X, y, learning rate, max iteration, tolerance)
Split data
Testing data 30%
Normalize and add bias
Training data 70%
Normalize and add bias
Initialize weights, run GD to update weights and minimize cost function
Predict using sigmoid function
Performance metrics

Gradient descent algorithm

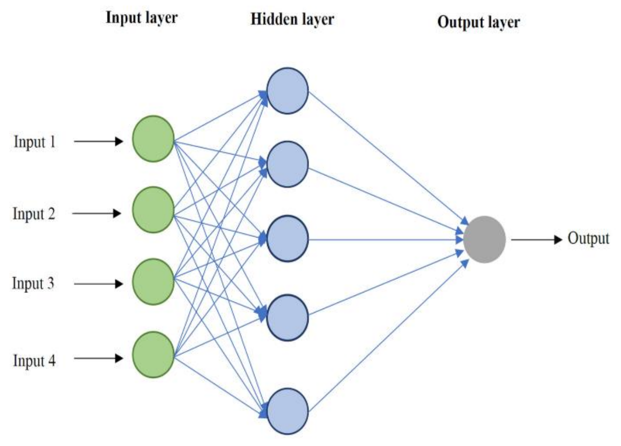
$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$

(Image source - link)

Implementation of Logistic Regression:

Our logistic regression with gradient descent model takes in inputs X and y, the maximum number of iterations, learning rate, and tolerance. It also auto-splits the data, normalizes the train and test, and adds the bias term before initializing weights randomly and calling the gradient descent method. It includes a method to return the predictions of the given input set, another to return the probability of belonging to class 1 of the given input set, and one to return the performance matrix of the test split. It saves the weights as the attributes and saves the y1 probability of the test and the entire dataset.

Neural networks (NNs) are machine learning algorithm that can be used for classification tasks. They consist of an input layer, one or more hidden layers, and an output layer. Each layer consists of neurons that are connected to neurons in the adjacent layers. During training, the weights of these connections are adjusted to minimize a loss function. This is done using optimization algorithms such as gradient descent.



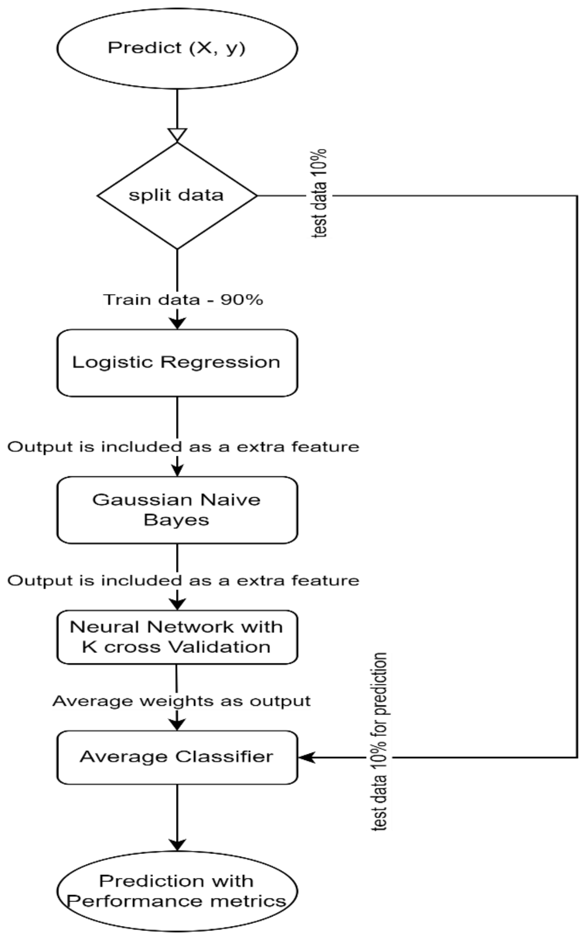
Implementation of Neural Network:

Our neural network is implemented as a python class with inputs X, and y. It includes k-fold cross-validation and an average classifier. It uses the KFold classifier to split the dataset into train and test sets for each fold. The model architecture is defined using the Keras Sequential API with three layers: the first layer has ReLU activation and the number of neurons can be specified as an inputs, the second layer has 1000 neurons and ReLU activation, and the output layer has sigmoid activation. The model is compiled with binary cross-entropy loss, accuracy metric, and Adam optimizer.

The Neural Network is implemented as a python class has the following methods:

- fit() method, the model is trained using k-fold cross-validation. For each fold, the data is split into train and test sets, and the model is trained for 20 epochs. The best weights are saved after each fold, and the final model is constructed by averaging the weights over all folds.
- predict() method that takes the test set as input and returns the predicted labels.
- error() method that calculates the average absolute error between the predicted and true labels
- performance_metrics() method that calculates and prints the accuracy, precision, recall, and F1 score of the model on the test set.

The average classifier method takes the predictions of the logistic regression, Gaussian Naive Bayes, and neural network models as input and returns the average of the predictions.


Predict (X, y)
split data
test data 10%
Train data - 90%
Logistic Regression
Output is included as a extra feature
Gaussian Naive Bayes
Output is included as a extra feature
Neural Network with K cross Validation
Average weights as output
Average Classifier
test data 10% for prediction
Prediction with Performance metrics

Stacked Ensemble model implemented in the project combines the predictions of three different models - Logistic Regression, Gaussian Naive Bayes, and an Average Neural Network Classifier. The Stacked Ensemble model uses the predictions of these three models as input features to train an average-classifier, which then makes the final prediction.

The Stacked Ensemble model is implemented as a Python class with the following methods:

(i) init: Initializes the Stacked Ensemble object with the input data X and y.

(ii) fit: Fits the Logistic Regression, Gaussian Naive Bayes, and Average Neural Network Classifier models on the input data X and y. The predictions of each model is then used as input features to train a next model and finally the average-classifier.

(iii)predict: Predicts the class label using the meta-classifier.

(iv)performance_metrics: Computes the performance metrics like accuracy, precision, recall, and F1 score for the predicted labels.

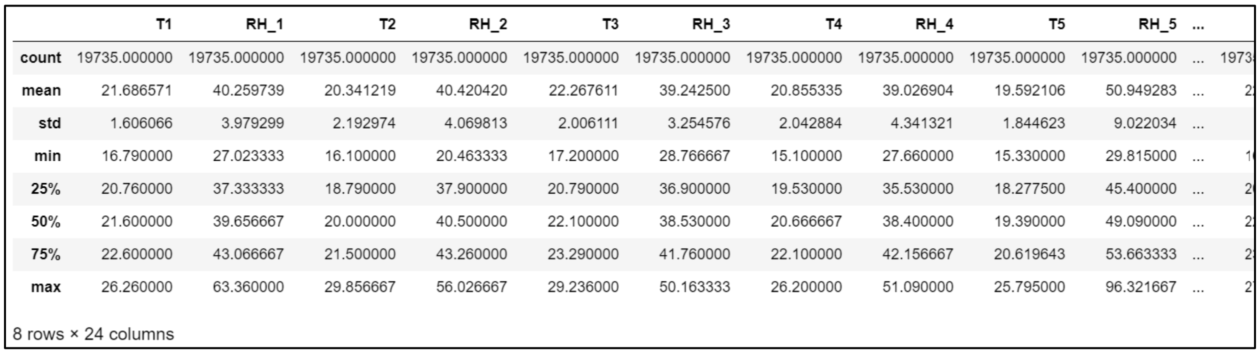
The Stacked Ensemble model combines the strengths of each individual model to make a more accurate prediction. The Logistic Regression model provides a linear decision boundary, while the Gaussian Naive Bayes model can handle non-linear decision boundaries. The Average Neural Network Classifier can learn complex non-linear relationships in the data.

By combining the predictions of these three models, the Stacked Ensemble model can make more accurate predictions than any individual model. The "fit" method fits a logistic regression model on the input data, creates a new dataset with the logistic regression output, fits a Gaussian Naive Bayes model on the new dataset, creates another new dataset with the Gaussian Naive Bayes output, and fits an average neural classifier on the final dataset. The "predict" method uses the trained model to predict the target variable for new data.

EXPLORATORY DATA ANALYSIS:

In the exploratory data analysis (EDA) phase, we first checked for any duplicate entries in the dataset, and none were found.

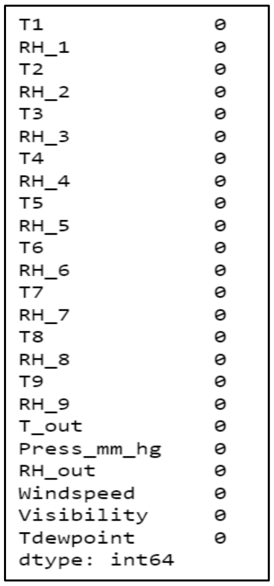
We then used the describe() method from pandas to get some general statistics about the dataset, including the mean, standard deviation, minimum and maximum values, and quartiles.



	T1	RH_1	T2	RH_2	T3	RH_3	T4	RH_4	T5	RH_5	...	
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	...	1973
mean	21.686571	40.259739	20.341219	40.420420	22.267611	39.242500	20.855335	39.026904	19.592106	50.949283	...	2
std	1.606066	3.979299	2.192974	4.069813	2.006111	3.254576	2.042884	4.341321	1.844623	9.022034	...	
min	16.790000	27.023333	16.100000	20.463333	17.200000	28.766667	15.100000	27.660000	15.330000	29.815000	...	1
25%	20.760000	37.333333	18.790000	37.900000	20.790000	36.900000	19.530000	35.530000	18.277500	45.400000	...	2
50%	21.600000	39.656667	20.000000	40.500000	22.100000	38.530000	20.666667	38.400000	19.390000	49.090000	...	2
75%	22.600000	43.066667	21.500000	43.260000	23.290000	41.760000	22.100000	42.156667	20.619643	53.663333	...	2
max	26.260000	63.360000	29.856667	56.026667	29.236000	50.163333	26.200000	51.090000	25.795000	96.321667	...	2

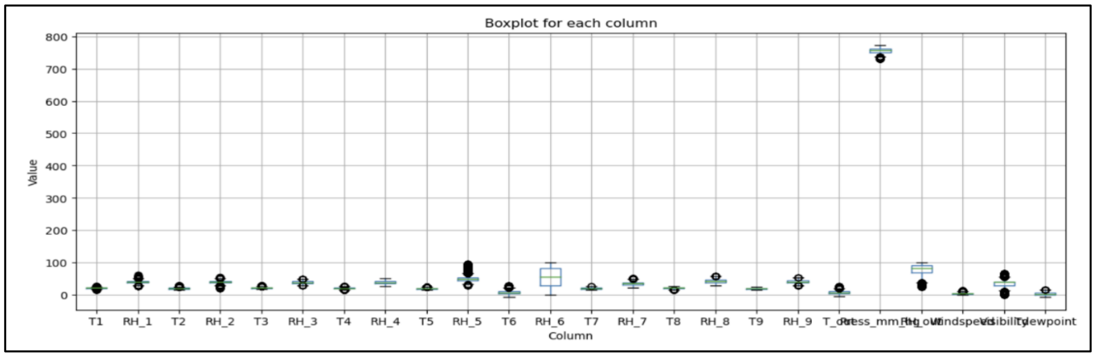
8 rows × 24 columns

Next, we checked for missing values in the dataset, and found that there were none.



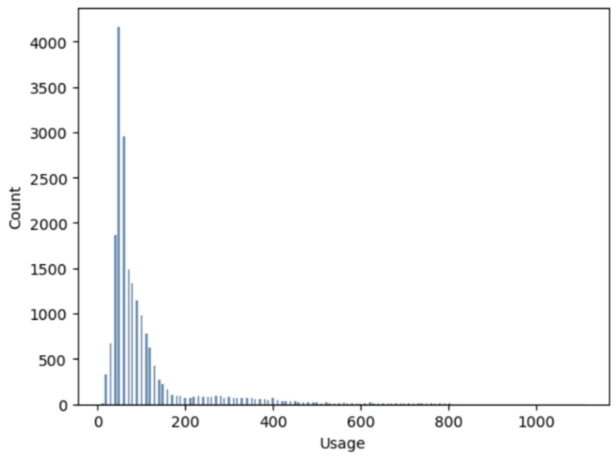
```
T1              0
RH_1            0
T2              0
RH_2            0
T3              0
RH_3            0
T4              0
RH_4            0
T5              0
RH_5            0
T6              0
RH_6            0
T7              0
RH_7            0
T8              0
RH_8            0
T9      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

To identify potential outliers, we created box plots for each of the numerical



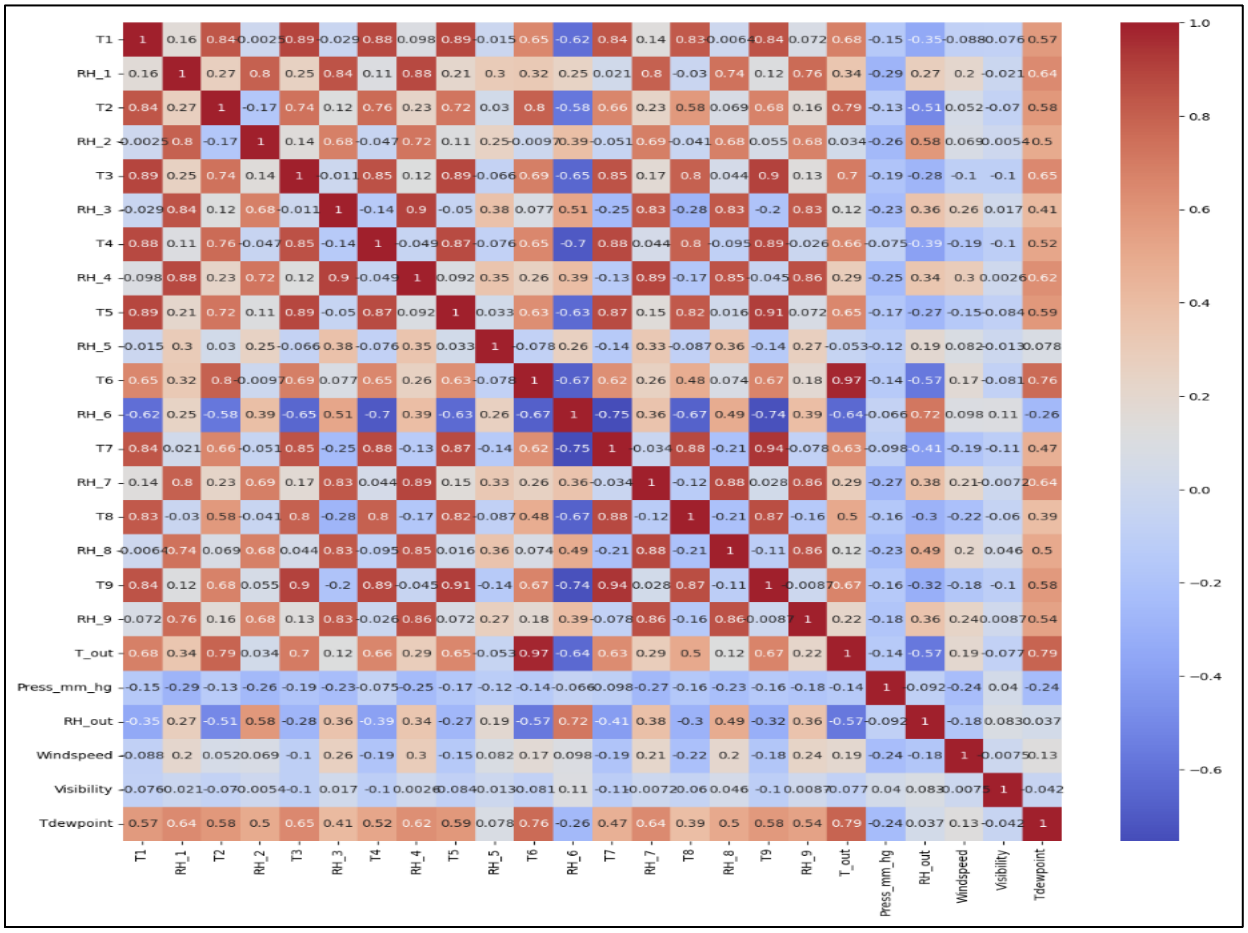
features and visually inspected them for any points outside the whiskers.

We created a distribution chart of the target variable.



The distribution represented in the graph resembles a normal distribution.

Finally, we generated a correlation matrix to examine the relationships between the features in the dataset.



There were significant correlations between many of the features, indicating that multicollinearity was an issue. This is one of the reasons we converted the prediction model into outlier classification.

Overall, the EDA phase helped us to gain a better understanding of the dataset and identify any potential issues that needed to be addressed in the subsequent steps of the project.

RESULTS:

Regression Models:

We evaluated the performance of two versions of the regression model: one with PCA dimensionality reduction and another without, and with regularization parameter values of 10, 20, and 30. The Root mean squared error (RMSE) was used as the performance metric. The results are presented in Table 1.

Table 1: Performance of regression models

Regularization Parameter	PCA	Regularization	RMSE
10	yes	yes	108.97
10	no	no	208.24
10	yes	no	107.18
10	no	yes	130.22
20	yes	yes	111.54
20	no	no	119.28
20	yes	no	106.34
20	no	yes	143.83
30	yes	yes	114.26
30	no	no	119.67
30	yes	no	107.48
30	no	yes	128.37

We observed that the model with PCA and regularization (with a regularization parameter of 10) had the lowest RMSE of 107.18, indicating that this model performed better than the others.

Table 2: Shows the number of Principal Components with its explained variances.

PCA	Explained Variance
1	68.19355598
2	77.1366101
3	84.95426102
4	90.81630069
5	94.61825205
6	97.41288076
7	98.07668203
8	98.52698594
9	98.96817791
10	99.19852654
11	99.39575218
12	99.54919055
13	99.69623548
14	99.77228119
15	99.8250553
16	99.8663525
17	99.90250968
18	99.92992919
19	99.9530747
20	99.96988646
21	99.98234862
22	99.99015241
23	99.99579068
24	100

Classification Models:

We evaluated the performance of three classification models: Logistic Regression, Gaussian Naïve Bayes, and Neural Networks. We also applied K-fold cross-validation and an average classifier on the Neural Network to enhance its performance.

Additionally, we created a stacked ensemble model using the three classifiers to improve the detection of outliers. The performance metrics used for evaluation were accuracy, precision, recall, and F1 score.

The **Logistic Regression model** was trained on the dataset, and the results for different hyperparameters are presented in Table 3.

Table 3: Performance of Logistic Regression

Max Iteration	Learning Rate	Accuracy	Precision	Recall	F1 Score
5000	0.00001	0.66	0.66	0.67	0.67
5000	0.0001	0.6	0.6	0.63	0.61
5000	0.001	0.58	0.59	0.54	0.57
5000	0.01	0.58	0.59	0.54	0.57
5000	0.1	0.58	0.59	0.55	0.57

The highest accuracy and F1 score achieved were 0.66 and 0.67, respectively, with a learning rate of 0.00001 and 5000 iterations.

The **Gaussian Naïve Bayes** model was trained on the dataset, and the results are presented in Table 1. The model achieved an accuracy of 0.58, a precision of 0.59, a recall of 0.54, and an F1 score of 0.57.

Table 4: Performance of Gaussian Naïve Bayes

Model	Accuracy	Precision	Recall	F1 Score
Gauissian Naïve Bayes	0.58	0.59	0.54	0.57

The **Neural Network model** with an average classifier and k-fold cross-validation showed promising results.

Table 5: Performance of Neural Network with K-fold cross validation and Average Classifier

Epochs	Accuracy	Precision	Recall	F1 Score
5	0.74	0.72	0.81	0.76
10	0.75	0.71	0.85	0.77
15	0.87	0.84	0.92	0.88
20	0.89	0.84	0.96	0.90

The best performance was observed at 20 epochs, where the model achieved an accuracy of 0.89 and an F1 score of 0.90.

The average classifier and k-fold cross-validation techniques were used to minimize the risk of overfitting and to improve the generalization of the model. The average classifier technique involves training multiple models on different training subsets and averaging their predictions to obtain the final prediction. The k-fold cross-validation technique involves dividing the dataset into k subsets and training the model on k-1 subsets while validating it on the remaining subset. This process is repeated k times, and the results are averaged to obtain the final performance metric.

The **Stacked Ensemble model** combines multiple base models to improve the overall performance of the model.

Table 6: Performance of Stacked Ensemble

Dataset	Accuracy	Precision	Recall	F1 Score
Training	0.889	0.868	0.917	0.892
Test	0.887	0.864	0.917	0.89

The stacked ensemble model achieved an accuracy of 0.887 on the test dataset and a precision of 0.864, recall of 0.917, and an F1 score of 0.89. The performance on the training dataset was also impressive, with an accuracy of 0.889 and an F1 score of 0.892.

Bias variance tradeoff – for stacked ensemble

One of the key advantages of the stacked ensemble model is its ability to minimize the bias and variance tradeoff. The bias-variance tradeoff refers to the tradeoff between overfitting and underfitting in a model. A model with high bias will underfit the data, while a model with high variance will overfit the data. The stacked ensemble model can reduce the bias by combining multiple models and increase the model's robustness by reducing the variance.

If the training accuracy is 88.9% and the test accuracy is 88.7%, it indicates that the model is not overfitting or underfitting the training data since both the accuracies are very close. Since the test accuracy is slightly lower than the training accuracy, it suggests that the model is marginally overfitting on the unseen data.

CONCLUSION:

Logistic Regression: The model achieved an accuracy of 0.58, precision of 0.59, recall of 0.54, and F1 score of 0.57.

Gaussian Naive Bayes: The model achieved an accuracy of 0.58, precision of 0.59, recall of 0.54, and F1 score of 0.57.

Average Neural Network with 5-fold cross-validation: The model achieved an accuracy ranging from 0.74 to 0.89, precision ranging from 0.71 to 0.84, recall ranging from 0.81 to 0.96, and F1 score ranging from 0.76 to 0.9 across different epochs.

Stacked Ensemble: The model achieved an accuracy of 0.887, precision of 0.864, recall of 0.917, and F1 score of 0.89 on the test set.

Regularization and PCA: The best RMSE was achieved with regularization parameter 10, PCA on and regularization on, which resulted in an RMSE of 108.97.

Overall, the stacked ensemble model had the best performance on the test set with an accuracy of 0.887, precision of 0.864, recall of 0.917, and F1 score of 0.89. However, the average neural network with 15 or 20 epochs also performed well with an accuracy of 0.87 or 0.89 respectively.