Summary:
This report presents the results of a study on using a Sugeno fuzzy inference model to estimate heating and cooling loads for 768 different building shapes. The dataset consists of eight features and output variables. The Energy Efficiency Dataset available on Kaggle was used to train a Sugeno fuzzy inference model, and its performance was evaluated based on how well it predicted heating and cooling loads.

Introduction:
Buildings account for a significant portion of energy consumption in modern society, and reducing energy consumption in buildings is critical to achieving sustainable development goals. The aim of this study is to investigate the effectiveness of using a Sugeno fuzzy inference model based on eight different features to predict heating and cooling loads for buildings. The dataset containing 768 building shapes simulated in Ecotect was obtained from Kaggle.
The dataset includes the following parameters:
• Relative Compactness (X1): This feature represents the relative compactness of the building calculated as the ratio of the building volume to the building shell volume. It is a dimensionless parameter.
• Surface Area (X2): This feature represents the total surface area of the building.
• Wall Area (X3): This feature represents the total area of the walls of the building.
• Roof Area (X4): This feature represents the total area of the roof of the building.
• Overall Height (X5): This feature represents the total height of the building measured from the ground to the roof.
• Orientation (X6): This feature specifies the direction of the building and is expressed as an angle (in degrees) between 0 and 360.
• Glazing Area (X7): This feature represents the total glazing area of the building.
• Glazing Area Distribution (X8): This feature represents the distribution of glazing area and is a dimensionless parameter.
The predicted results are as follows:
• Heating Load (y1): This output represents the heating load per square meter (kWh/m2) of the building.
• Cooling Load (y2): This output represents the cooling load per square meter (kWh/m2) of the building.

Materials and Methods:
A Sugeno fuzzy inference model was used to estimate the heating and cooling loads of buildings. To train the Sugeno fuzzy inference model, 700 out of the 768 rows in the dataset were used for training, and the remaining 68 rows were used for testing. The training set was used to train the model, and the test set was used to evaluate its performance. This section will also explain all the code snippets written to train the model.
The first step is to export the data from the .csv file to the MATLAB program.

```
clear;
clc;
% read the data table from file
cd C:\Users\omart\Desktop\School\'4th Term'\'7. Yapay Zeka'\homework2\;
table = readtable('energy.csv');
% convert data table to array
data = table2array(table);
table
```

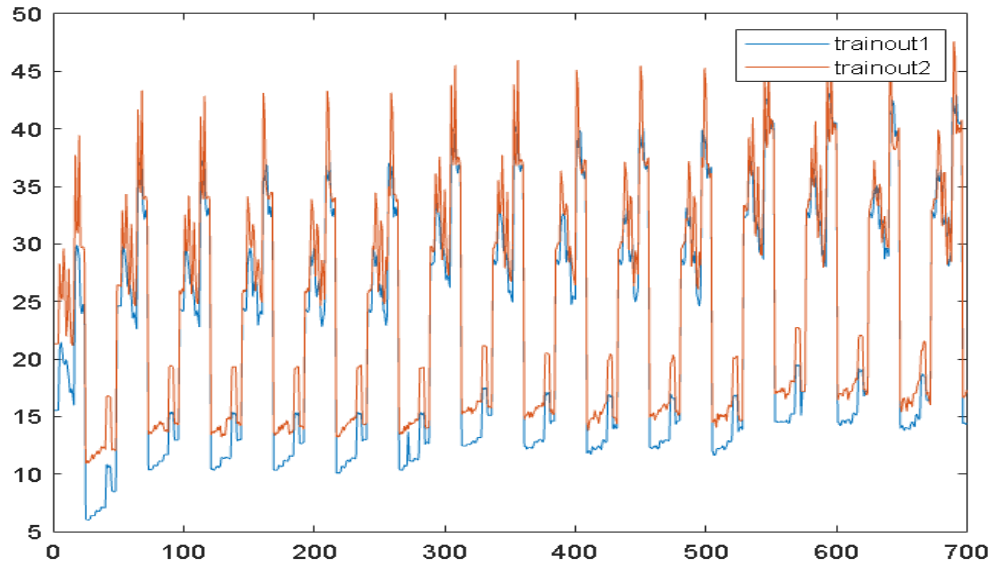
table = 768×10 table

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
1	0.98	514.5	294	110.25	7	2	0	0	15.55	21.33
2	0.98	514.5	294	110.25	7	3	0	0	15.55	21.33
3	0.98	514.5	294	110.25	7	4	0	0	15.55	21.33
4	0.98	514.5	294	110.25	7	5	0	0	15.55	21.33
5	0.9	563.5	318.5	122.5	7	2	0	0	20.84	28.28
6	0.9	563.5	318.5	122.5	7	3	0	0	21.46	25.38
7	0.9	563.5	318.5	122.5	7	4	0	0	20.71	25.16
8	0.9	563.5	318.5	122.5	7	5	0	0	19.68	29.6

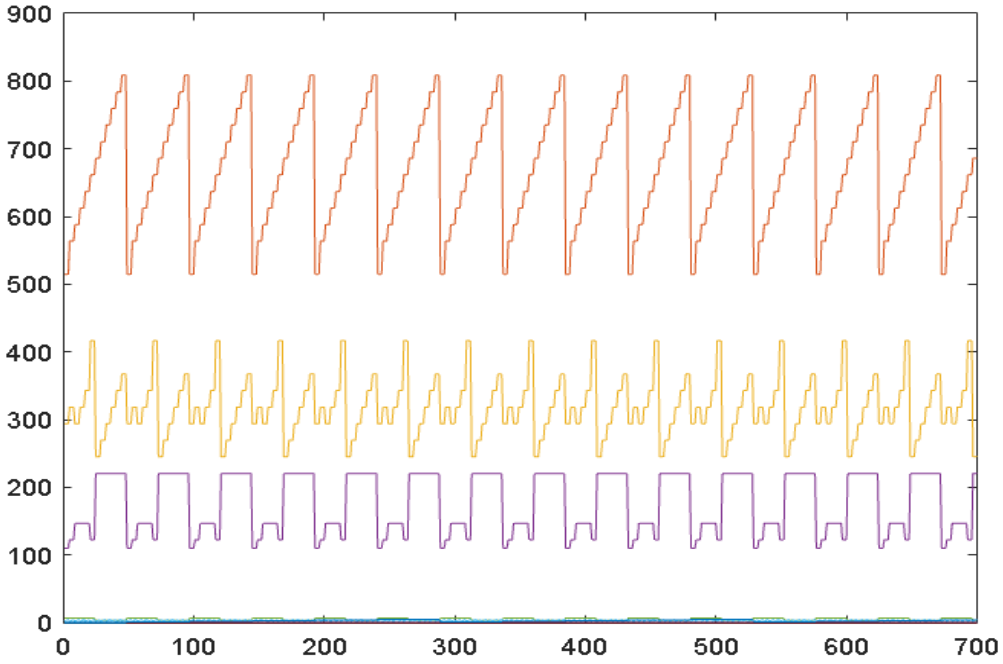
⋮

Then, the data was split into training and test sets for each output and visualized.

```
% split data into training and validation sets
% train
train_in = data(1:700,1:8);
train_out1 = data(1:700,9);
train_out2 = data(1:700,10);
% test
test_in = data(701:768,1:8);
test_out1 = data(701:768,9);
test_out2 = data(701:768,10);
% plot input and output training data
plot(train_out1)
hold on
plot(train_out2)
legend('trainout1','trainout2');
hold off
```




```
plot(train_in)
```



In the next step, two separate Sugeno Fuzzy Inference Systems (FIS) were created using the 'genfis' function for both Y1 (Heating Load) and Y2 (Cooling Load) outputs.

```
% generate sugeno fis
fis1 = genfis(train_in,train_out1)
fis2 = genfis(train_in,train_out2)
```

```
fis1 =
  sugfis with properties:

                      Name: "fis"
                 AndMethod: "prod"
                  OrMethod: "max"
           ImplicationMethod: "prod"
           AggregationMethod: "sum"
       DefuzzificationMethod: "wtaver"
    DisableStructuralChecks: 0
                    Inputs: [1×8 fisvar]
                   Outputs: [1×1 fisvar]
                     Rules: [1×256 fisrule]
```

```
fis2 =
  sugfis with properties:

                      Name: "fis"
                 AndMethod: "prod"
                  OrMethod: "max"
           ImplicationMethod: "prod"
           AggregationMethod: "sum"
       DefuzzificationMethod: "wtaver"
    DisableStructuralChecks: 0
                    Inputs: [1×8 fisvar]
                   Outputs: [1×1 fisvar]
                     Rules: [1×256 fisrule]
```

The following code first sets up the initial FISs (fis1 and fis2) and then trains these FISs using the 'anfis' function. In the initial lines, the 'InitialFIS' option in the 'anfisOptions' function specifies the initial FIS to be used before starting the training process. The 'EpochNumber' option determines how many times the FIS will be trained. ANFIS (Adaptive Neuro-Fuzzy Inference System) algorithm is a machine learning method used to analyze and model datasets using both neural network and fuzzy logic principles. It is used to learn the relationships between inputs and outputs in a dataset.

```
% setting initial fis
opt1 = anfisOptions('InitialFIS',fis1,'EpochNumber',5);
opt2 = anfisOptions('InitialFIS',fis2,'EpochNumber',5);
% tune fis
fis1 = anfis([train_in train_out1],opt1)
```

```
ANFIS info:
     Number of nodes: 555
     Number of linear parameters: 2304
     Number of nonlinear parameters: 48
     Total number of parameters: 2352
     Number of training data pairs: 700
     Number of checking data pairs: 0
     Number of fuzzy rules: 256
Start training ANFIS ...

1     0.238602
2     0.237863
3     0.237011
4     0.236058
Step size increases to 0.011000 after epoch 5.
5     0.234926

Designated epoch number reached. ANFIS training completed at epoch 5.
```

```
Minimal training RMSE = 0.234926
```

```
fis2 = anfis([train_in train_out2],opt2)
```

```
ANFIS info:
     Number of nodes: 555
     Number of linear parameters: 2304
     Number of nonlinear parameters: 48
     Total number of parameters: 2352
     Number of training data pairs: 700
     Number of checking data pairs: 0
     Number of fuzzy rules: 256
Start training ANFIS ...

1     0.673007
2     0.670924
3     0.668825
4     0.666658
Step size increases to 0.011000 after epoch 5.
5     0.664464

Designated epoch number reached. ANFIS training completed at epoch 5.

Minimal training RMSE = 0.664464
```

Next, using 'fis1' and 'fis2', output predictions are made for input data ('test_in') in the test set. The difference between 'test_out1' and 'y1' ('error1') is calculated. This shows how much the predictions of the 'fis1' model differ from the actual test results. Similarly, the difference between 'test_out2' and 'y2' can be calculated as another measure of error. These errors can be used to evaluate and improve the performance of the model.

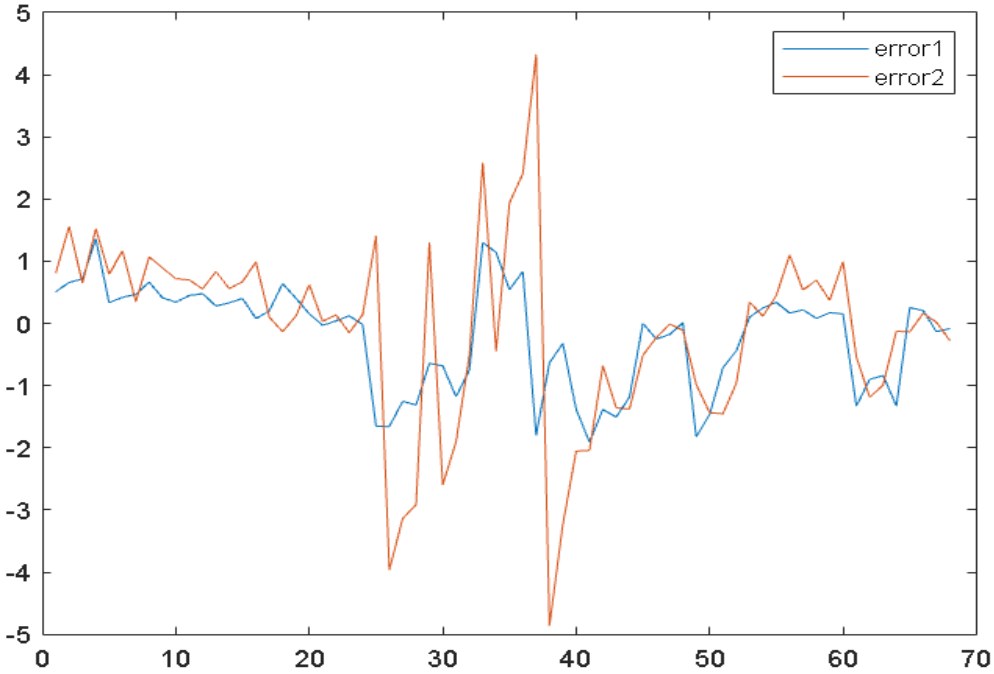
```
% y1 and y2 are test results
y1 = evalfis(fis1,test_in);
y2 = evalfis(fis2,test_in);
% calculate error for both outputs
error1 = test_out1-y1
error2 = test_out2-y2
```

```
error2 = 68×1
     0.8093
     1.5573
     0.6471
     1.5240
     0.7928
     1.1656
     0.3463
     1.0686
     0.8903
     0.7187
      ⋮
```

```
error1 = 68×1
     0.5095
     0.6586
     0.7156
     1.3587
     0.3312
     0.4195
     0.4673
     0.6638
     0.4102
     0.3422
      ⋮
```

```
plot(error1)
```

```
hold on
plot(error2)
legend('error1','error2');
hold off
```



As seen in the graphs, error values are mostly low and maximum error values are below 5. This may not have a significant impact as our outputs are temperature values, but in more sensitive applications, even these small differences can cause significant problems.

```
% calculate mse for each output
mse1 = mean(error1.*error1)
```

```
mse1 = 0.7055
```

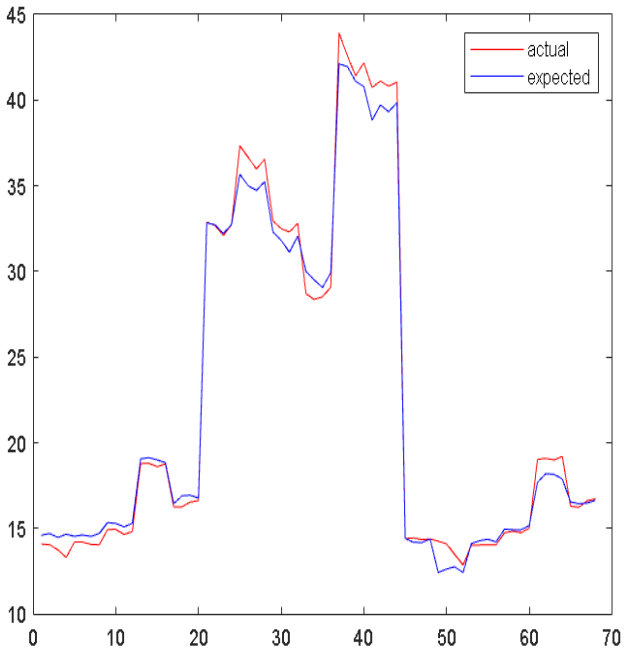
```
mse2 = mean(error2.*error2)
```

```
mse2 = 2.3087
```

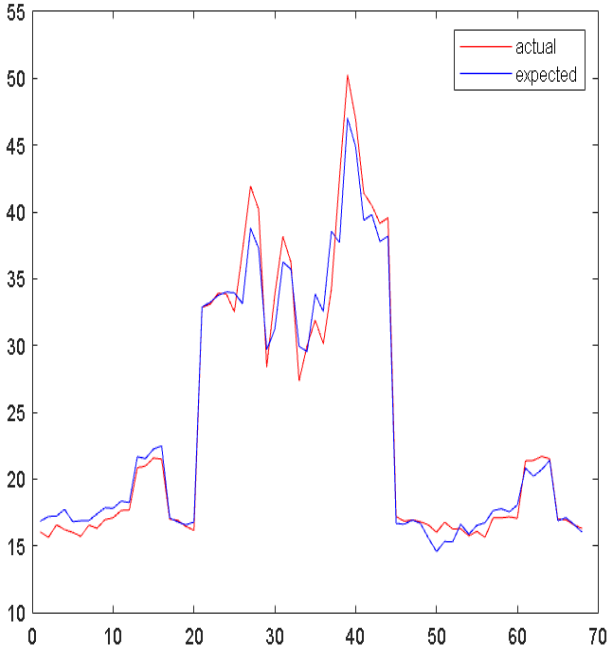
This code calculates the mean squared error (MSE) values for each output by taking the average of the squares of the "error1" and "error2" error arrays. This is a measure that shows how different the prediction is from the actual value for each output. As a result, the "mse1" value was calculated as 0.7055 and the "mse2" value as 2.3087. These results were visualized using the following code to examine them further.

```
% wanted output and fis results comparison
% for first output
plot(y1,'r')
hold on
plot(test_out1,'b')
```

```
legend('actual','expected');
hold off
% for second output
plot(y2,'r')
hold on
plot(test_out2,'b')
legend('actual','expected');
hold off
```



Heating Load Graph



Cooling Load Graph

References:

- https://www.mathworks.com/help/fuzzy/anfis.html?searchHighlight=anfis&s_tid=srchtitle_anfis_1
- https://www.kaggle.com/datasets/elikplim/eergy-efficiency-dataset