Creating an early warning system for detecting high-frequency abnormalities in quality by neural network

Bahare Nasir

Context

1. Introduction
2. Data preparation
    - Import data
    - Data normalization
    - creating normalized data matrices
    - Normalization plot
    - Checking the correlation between the data
    - P-value test
    - Plot variable correlations
    - Binary classification
3. Creating a Neural Network model
4. A summary of how to create a multivariate control chart and how to use it
5. Research overview

Introduction

The lecture focuses on the research conducted in a sugar factory in Qazvin, Iran, specifically regarding the quality grading of raw syrup in the sugar production system. The researchers aimed to control and predict the behavior of the system by using control charts, statistical monitoring, and neural network models.

The research involved studying the system, identifying input and output variables, and collecting refined experimental data. Various neural network models were trained, validated, and tested using different configurations. The most suitable model was selected based on performance indicators.

Parameters such as pH of raw syrup, BX of raw syrup, and other factors related to water and extraction processes were considered for simulating syrup quality.

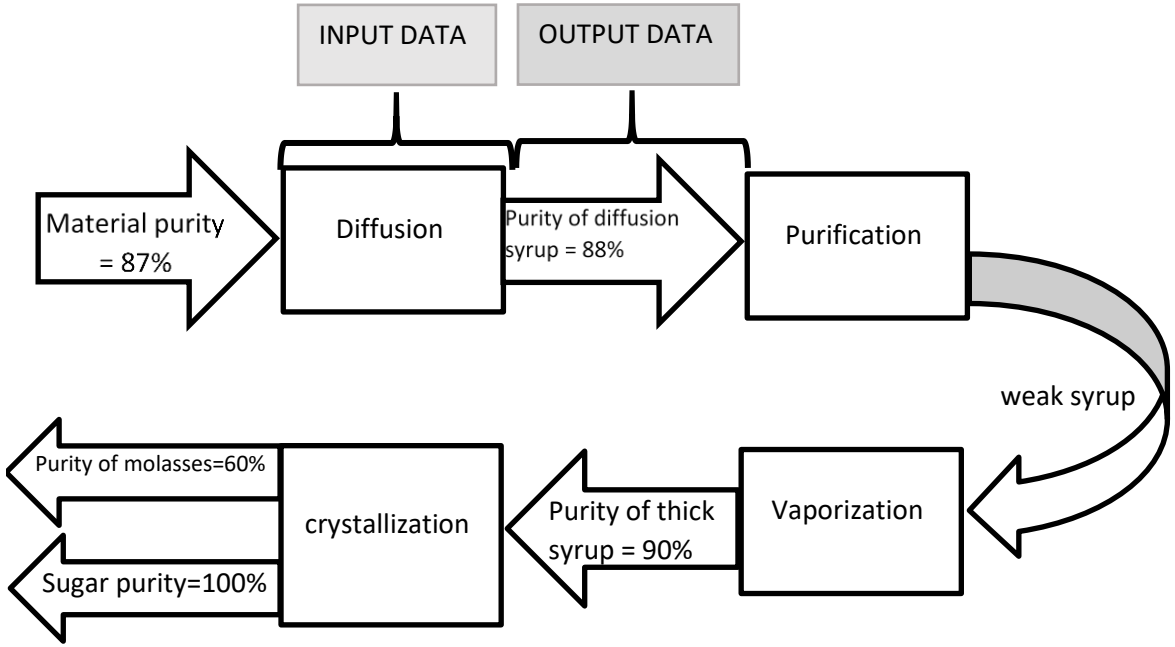
After selecting the best neural network model, a control chart was chosen to analyze the residuals of the model. This chart helps detect significant changes in the process level, allowing corrective actions to be taken promptly.

Introduction

One notable aspect of the research is the provision of a reliable model that can predict the system's behavior under new operational conditions. This predictive capability helps in avoiding potential damages or quality reduction by providing early warnings.

Overall, the research highlights the importance of quality grading in the sugar production system and demonstrates the use of statistical monitoring, control charts, and neural network models to control, predict, and ensure the quality of the raw syrup production process.

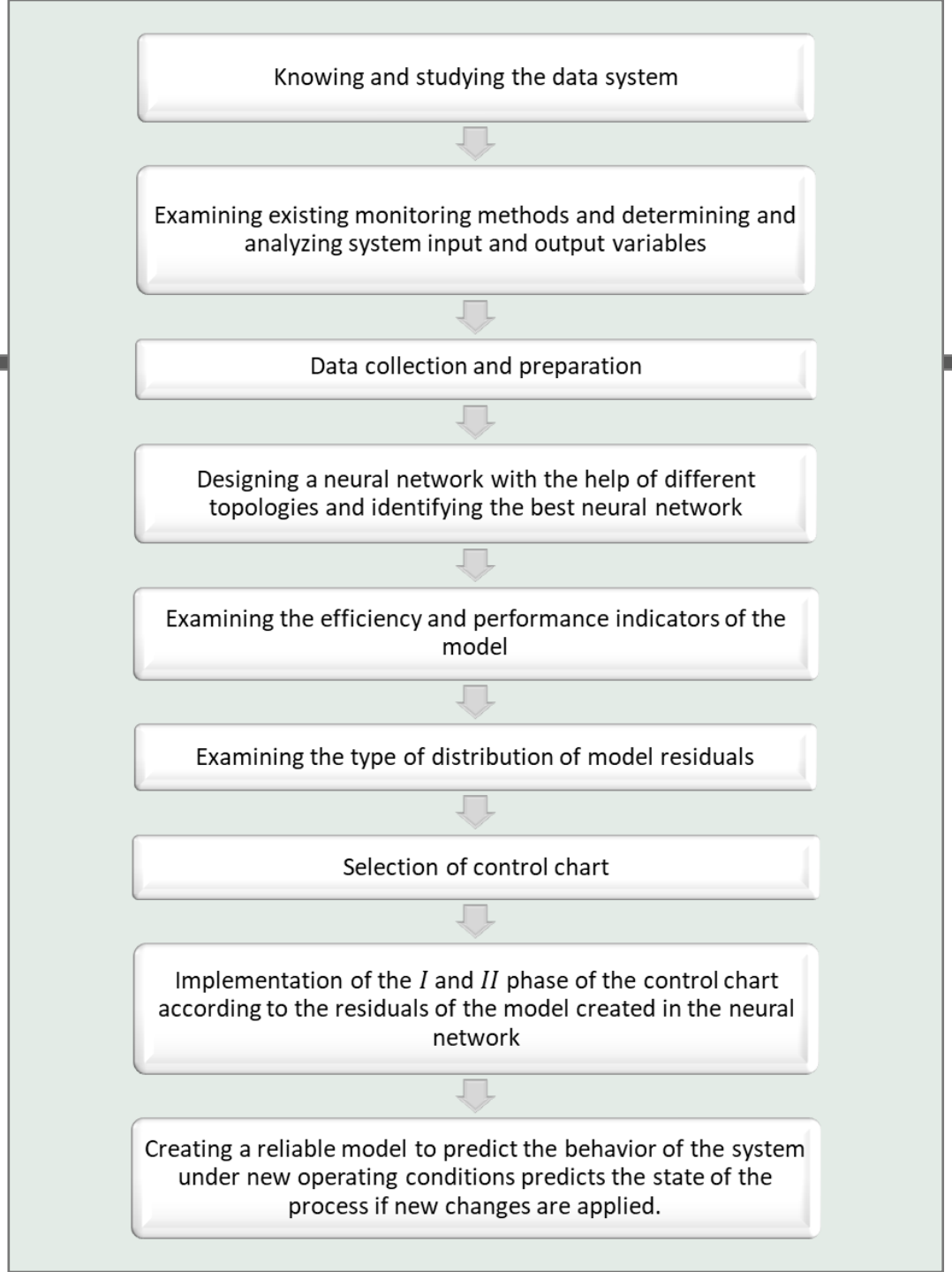
A summary of the stages of the sugar production process from sugar beet:



One of the most important stages of sugar extraction in the sugar beet and sugarcane industry is diffusion. The main purpose of diffusion is to obtain raw syrup with maximum purity and at least 100 percent sugar residue.

The challenge is related to the production of raw syrup, which is a fundamental step in getting the desired sugar, according to research done by experts in product quality.

Suggested research methodology



Data analysis and review

The data used in this research includes two categories.

The first category contains 2236 data for 5 primary variables (Diffusion, bagasse extraction, pH of fresh water, pH of water in diffusion, BX of water in diffusion) are in the process of converting sugar beet into raw syrup and are used as input data.

The second category includes 2236 data from 2 variables (BX of raw syrup and pH of raw syrup) which are formed as a result of the process of the previous part and are used as target data.

Data preparation

Step 1: Import data:

data = readtable("C:\Users\Bahare\Desktop\data\data1.csv");

- Input:

inputdata = [data.X1, data.X2, data.X3, data.X4, data.X5]

- Output:

outputdata = [data.Y1 , data.Y2]

Data preparation

Step 2: Data normalization:

It should be done and saved for each column separately: (we have 7 columns, of which 5 belong to input data and 2 belong to output data).

```
data = readtable("C:\Users\Bahare\Desktop\data\data1.csv");
Data = data.X1;
normalize_min_max=[0 1];
Data_size=size(Data);
Data_normalized=zeros(Data_size(1),Data_size(2));
Data_max=max(max(Data));
Data_min=min(min(Data));
Ratio=abs(normalize_min_max(2)-
normalize_min_max(1))/(abs(Data_max- Data_min));
for nn=1: Data_size(1)
   for mm=1: Data_size(2)
      Data_normalized(nn,mm)=normalize_min_max(1)+(
Data(nn,mm)- Data_min)*Ratio;
   end
end
Data_normalized
```

Data preparation

Step 3: creating normalized data matrices:

- Input:

input = [X1, X2, X3, X4, X5,]

- Output:

output = [Y1, Y2,]

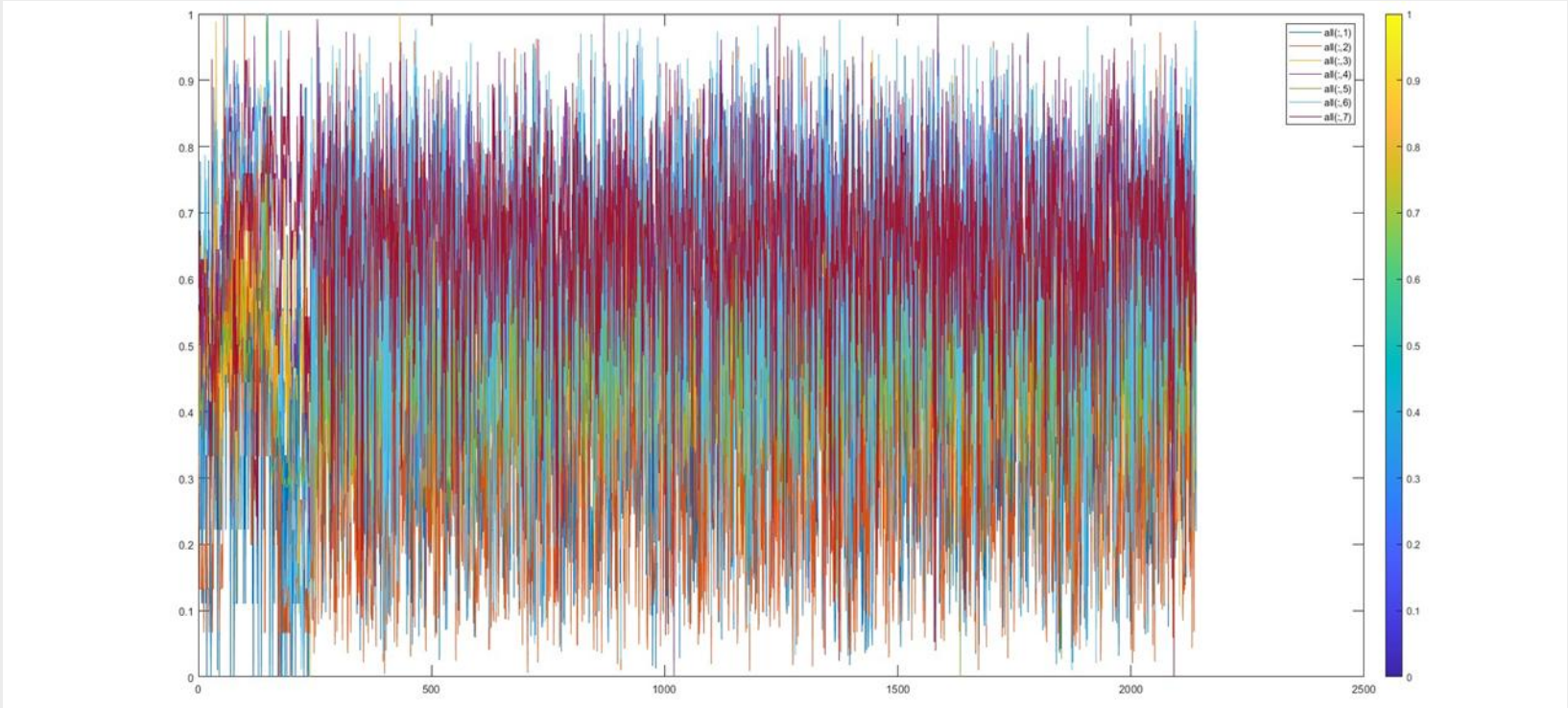
- All

all = [X1, X2, X3, X4, X5, Y1, Y2,]

Data preparation

Step 4: Normalization plot:

plot (all,'DisplayName','all') → ALL



Data preparation

Step 5: Checking the correlation between the data:

It should be done and saved for each column separately: (we have 7 columns, of which 5 belong to input data and 2 belong to output data).

One of the most important steps in the satisfactory development of the prediction model is the selection of appropriate input variables. Because it is trying to introduce a network with the least error and the highest correlation.

The most common method in evaluating the relationship between phenomena and the factors affecting them is to examine the correlation between them.

Therefore, Pearson's correlation coefficient is used in the following formula to calculate the correlation between variables:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) - (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

In this equation, 'x' is the independent variable and 'y' is the dependent variable and 'r' is a number between [±1]. The closer the 'r' number is to [±1], the stronger the relationship.

Data preparation

Step 5: Checking the correlation between the data:

allcorr = corrcoef(all,'Rows','complete')

	1	2	3	4	5	6	7
1	1	0.2122	0.0167	0.1171	0.1547	0.2301	0.0443
2	0.2122	1	0.1806	0.3019	0.4842	0.5802	0.0404
3	0.0167	0.1806	1	0.1691	0.1351	0.2332	0.0803
4	0.1171	0.3019	0.1691	1	0.1851	0.3417	0.4692
5	0.1547	0.4842	0.1351	0.1851	1	0.6420	-0.1316
6	0.2301	0.5802	0.2332	0.3417	0.6420	1	-0.0721
7	0.0443	0.0404	0.0803	0.4692	-0.1316	-0.0721	1

Conclusion:
According to the analysis of the obtained numbers, we can conclude that there is a relative correlation between the investigated data and we also see a degree of relative dependence between the variables of the first station (1 to 5) and the second station (6 and 7).

Data preparation

Step 6: P-value test:

```
load('allcorr.mat')
[RHO , PVAL] = corr(allcorr)
```

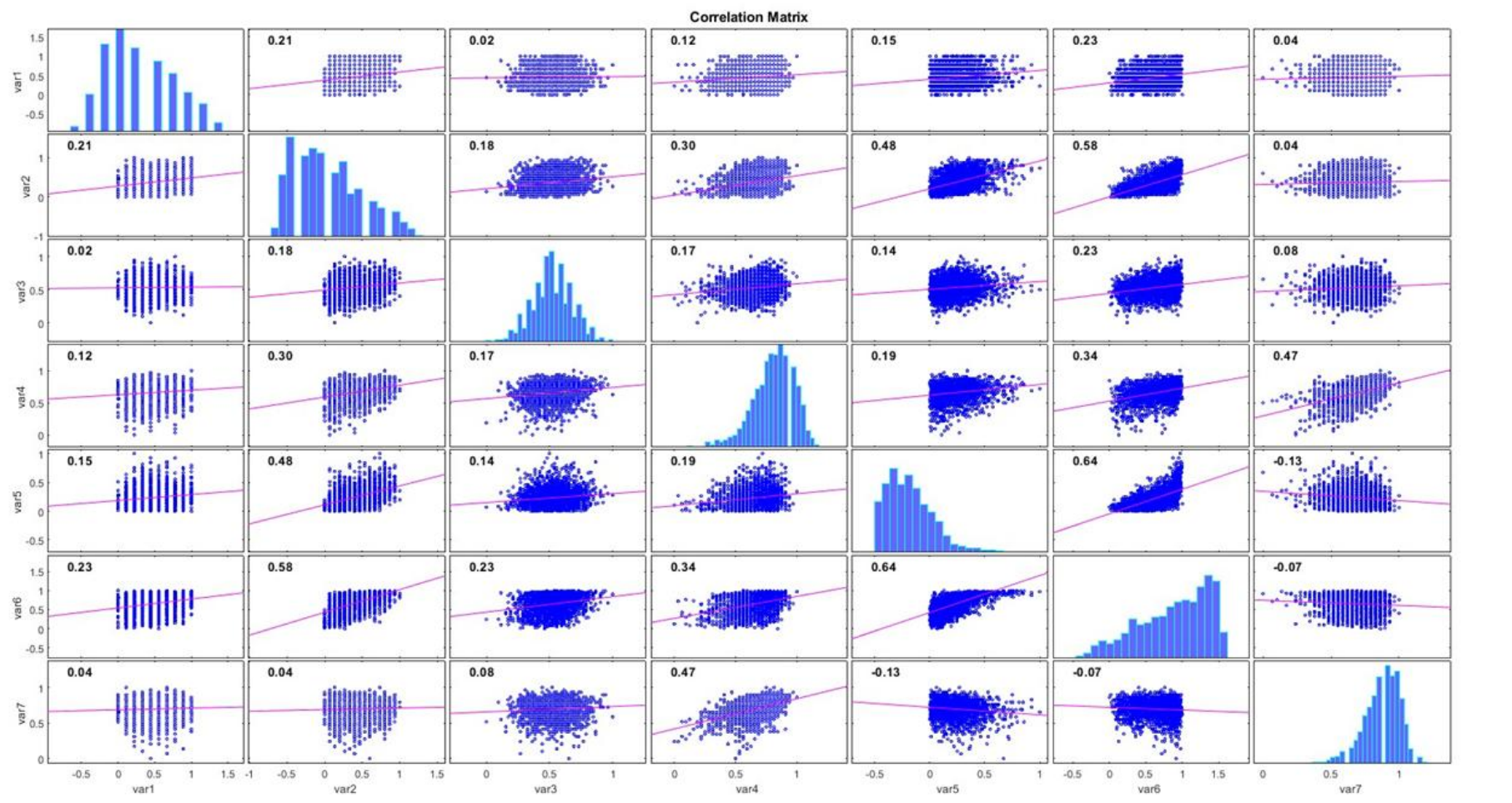
	1	2	3	4	5	6	7
1	1	0.9041	0.3380	0.4173	0.8942	0.9376	0.5089
2	0.9041	1	0.7246	0.8056	0.1318	0.0721	0.1930
3	0.3380	0.7246	1	0.6374	0.7586	0.8730	0.6730
4	0.4173	0.8056	0.6374	1	0.5366	0.7851	0.2115
5	0.8942	0.1318	0.7586	0.5366	1	0.0205	0.0496
6	0.9376	0.0721	0.8730	0.7851	0.0205	1	0.0572
7	0.5089	0.1930	0.6730	0.2115	0.0496	0.0572	1

Conclusion:
According to the analysis of the obtained numbers, it shows that since there is a significance above 0.5 between most of the variables, and also the variables of the first station (1 to 5) and the second station (6 to 9) have a significant relationship with each other. As a result, the following information can be checked in the neural network.

Data preparation

Step 7: Plot variable correlations:

R=corrplot(all,rows="complete")



Data preparation

Note:

The amount and type of the correlation coefficient graph between the X5 and Y1 variables indicate an abnormality between them. With the help of Binary Classification and the Ratio formula (=Y1/X5=), we can identify and eliminate these anomalies.

Here is the code to understand how many classes are needed and also classify them.

Data preparation

Step 8: Binary classification between input X5 and output Y1:

First, we need to specify how many classifications we should have:

The Elbow method is a technique used in classification to determine the optimal number of clusters in a dataset. It helps to identify the point at which adding more clusters does not significantly improve the model's performance.

- Data Preparation
- calculates the ratio between the output and input data by dividing
- input data, output data, and ratio are combined into a single matrix and define The range of cluster numbers to evaluate
- K-means Clustering
- Calculate Sum of Squared Errors (SSE) and Plot it
- Determine the Elbow Point:

  we can Identify number of clusters where (SSE) begins to level off significantly. It indicates the optimal number of clusters where the addition of more clusters does not result in a substantial reduction in SSE. The Elbow point is typically identified as the point on the SSE graph where the curve forms a noticeable bend or elbow.
- Choose the Optimal Number of Clusters

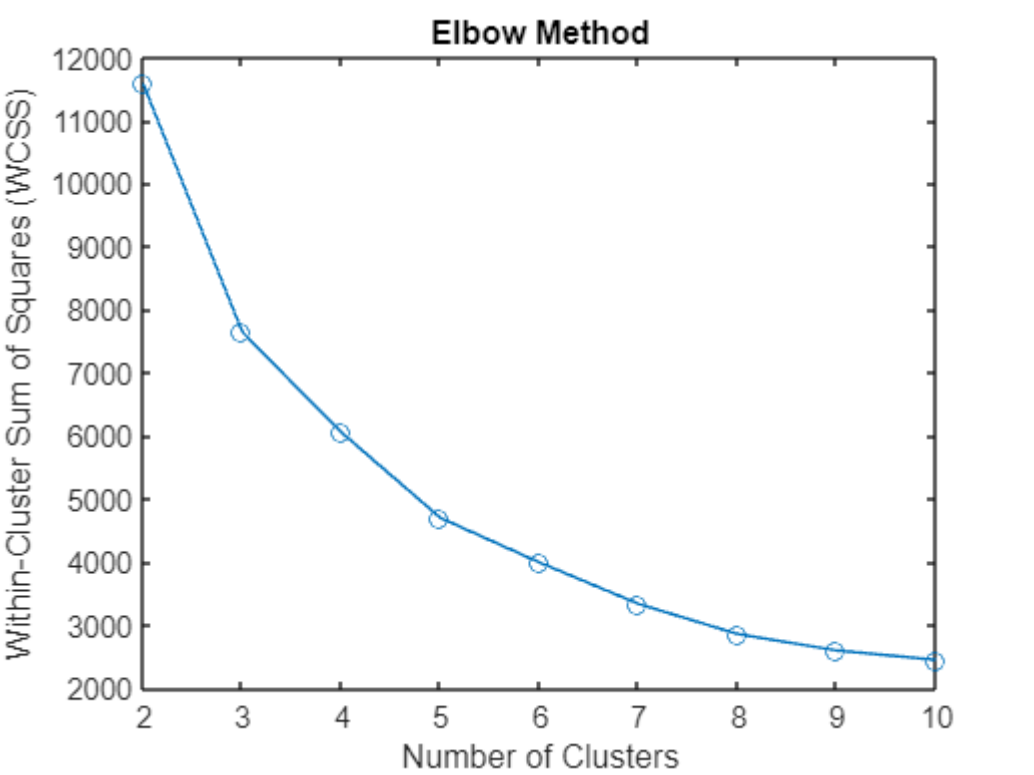
Code:

```
% Step 1: Prepare your data
data1 = readtable("data1.csv");
head(data1)
inputData = data1.X5;
outputData = data1.Y1;

% Calculate the ratio between Y and X
ratio = outputData ./ inputData;
% Combine input, output, and ratio into a single matrix
data = [inputData, outputData, ratio];

% Define the range of cluster numbers to evaluate
min_clusters = 2;  % Start from at least 2 clusters
max_clusters = 10;

% Perform k-means clustering for different cluster
numbers
wcss = zeros(max_clusters - min_clusters + 1, 1);
for num_clusters = min_clusters:max_clusters
    [~, ~, sumd] = kmeans(data, num_clusters);
    wcss(num_clusters - min_clusters + 1) = sum(sumd);
end
```

```
% Plot the within-cluster sum of squares (WCSS) as a
function of cluster number
figure;
plot(min_clusters:max_clusters, wcss, 'o-');
xlabel('Number of Clusters');
ylabel('Within-Cluster Sum of Squares (WCSS)');
title('Elbow Method');
% Find the elbow point using the "knee" of the curve
diff_wcss = diff(wcss);
[~, optimal_clusters] = max(diff_wcss);
optimal_clusters = optimal_clusters + min_clusters - 1;
% Perform k-means clustering with the optimal number of
clusters
[idx, ~, ~, ~] = kmeans(data, optimal_clusters);

% Add cluster labels as a new column in the original data
table
data1.Cluster = idx;
% Print the number of optimal clusters
disp(['Optimal number of clusters: '
num2str(optimal_clusters)]);
% Print the data table with cluster labels
disp(data1);
```

Final Result of Elbow Method

Optimal number of clusters: 9



Data preparation

Step 8: Binary Classification by neural network in order to have a better prediction:

```
% Step 2: Prepare your data for classification
data1.Cluster = cellstr(num2str(data1.Cluster)); % Convert numeric to cell array
of strings
data1.Cluster = categorical(data1.Cluster, {'1','2','3','4','5','6','7','8','9'}, 'Ordinal',
true);
% The partition randomly divides the observations into kfolds, each of which
has approximately the same number of observations.
rng("default") % For reproducibility of the partition
c = cvpartition(2236, "KFold", 5);
trainingIndices = training(c, 1); % Indices for the training set
validationIndices = test(c, 1); % Indices for the validation set
dataa1Train = data1(trainingIndices, :);
dataa1Validation = data1(validationIndices, :);
% Train a neural network classifier using the training data
Mdl = fitcnet(dataa1Train, "Cluster");
% Get more information about the training history of the neural network model
Mdl.TrainingHistory
% Evaluate the performance of the classifier on the validation set by computing
the validation set classification error
validationPredictions = predict(Mdl, dataa1Validation);
validationAccuracy = 1 - loss(Mdl, dataa1Validation, "Cluster", "LossFun",
"classiferror");
% Create the validation confusion matrix
validationConfusionMatrix = confusionmat(dataa1Validation.Cluster,
validationPredictions);
% Display the validation confusion matrix
disp("Validation Confusion Matrix:");
disp(validationConfusionMatrix);
```

Data preparation

Step 8: Binary Classification by neural network:

Note:

After removing anomalies with the help of this method and doing it again in order to finalize the data, we will have:

(This summary is about the final result after removing outliers and reclassifying to ensure how the data is predicted)

Data preparation

Step 8: Binary Classification by neural network:

1. Result summary:

Model 2.1: Neural Network
Status: Trained
Model is favorite

Training Results

Accuracy (Validation)	99.7%
Total cost (Validation)	Not applicable
Prediction speed	~40000 obs/sec
Training time	5.7057 sec

Model Hyperparameters

Preset: Narrow Neural Network
Number of fully connected layers: 1
First layer size: 10
Activation: ReLU
Iteration limit: 1000
Regularization strength (Lambda): 0
Standardize data: Yes

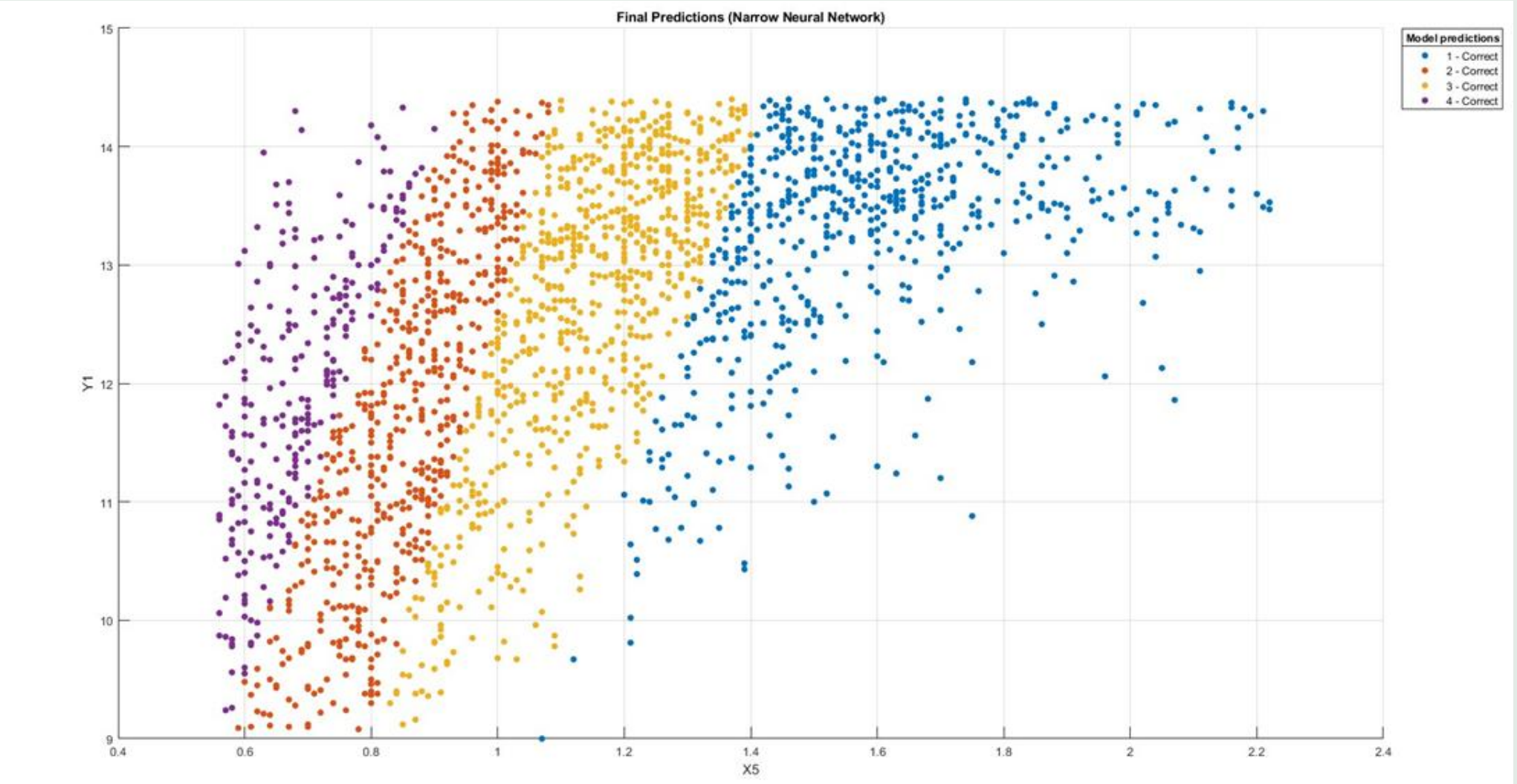
Feature Selection: 2/2 individual features selected

	Select	Features
1	☑	X5
2	☑	Y1

Data preparation

Step 8: Binary Classification by neural network:

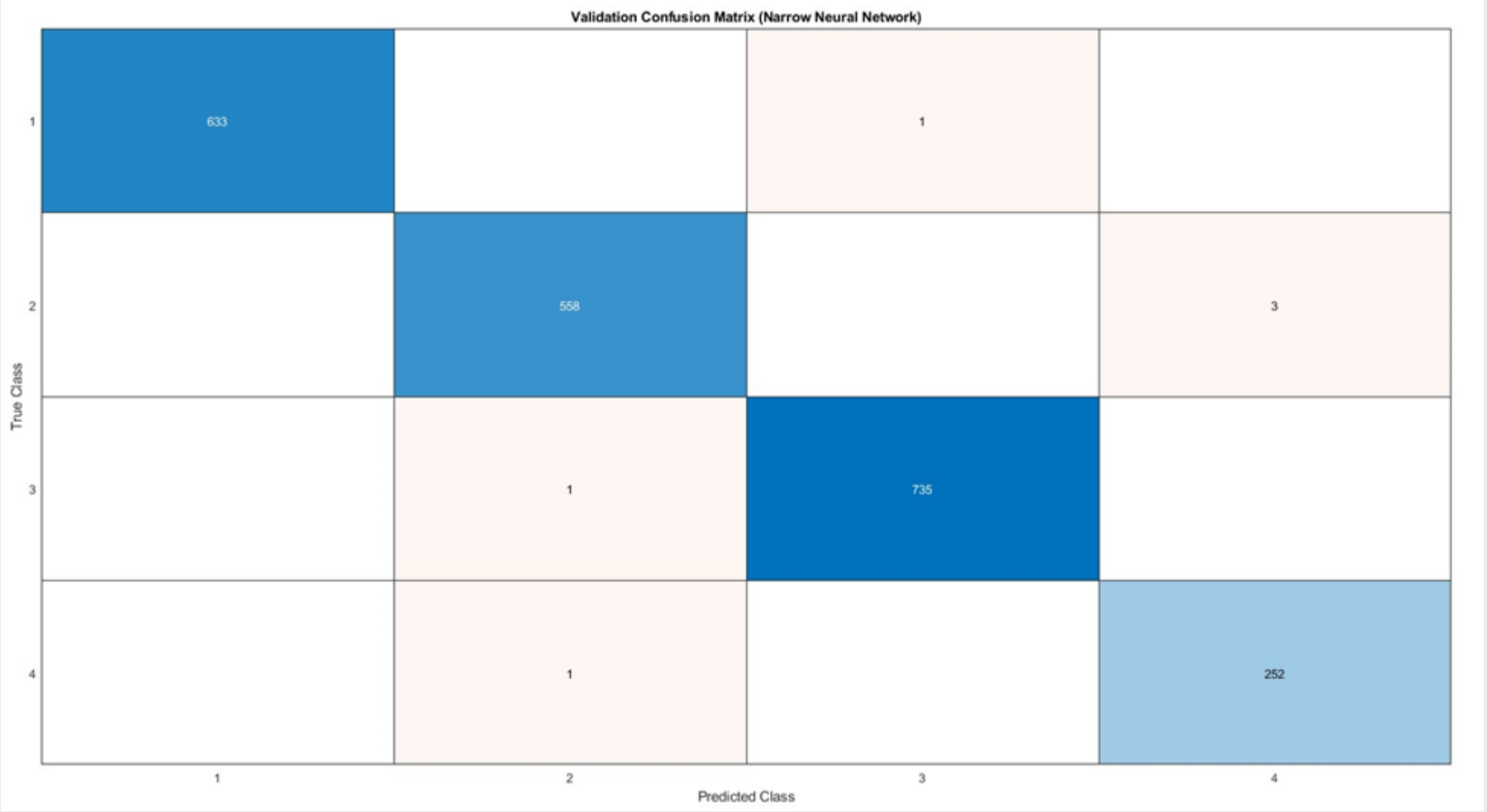
2. Scatter plot:



Data preparation

Step 8: Binary Classification by neural network:

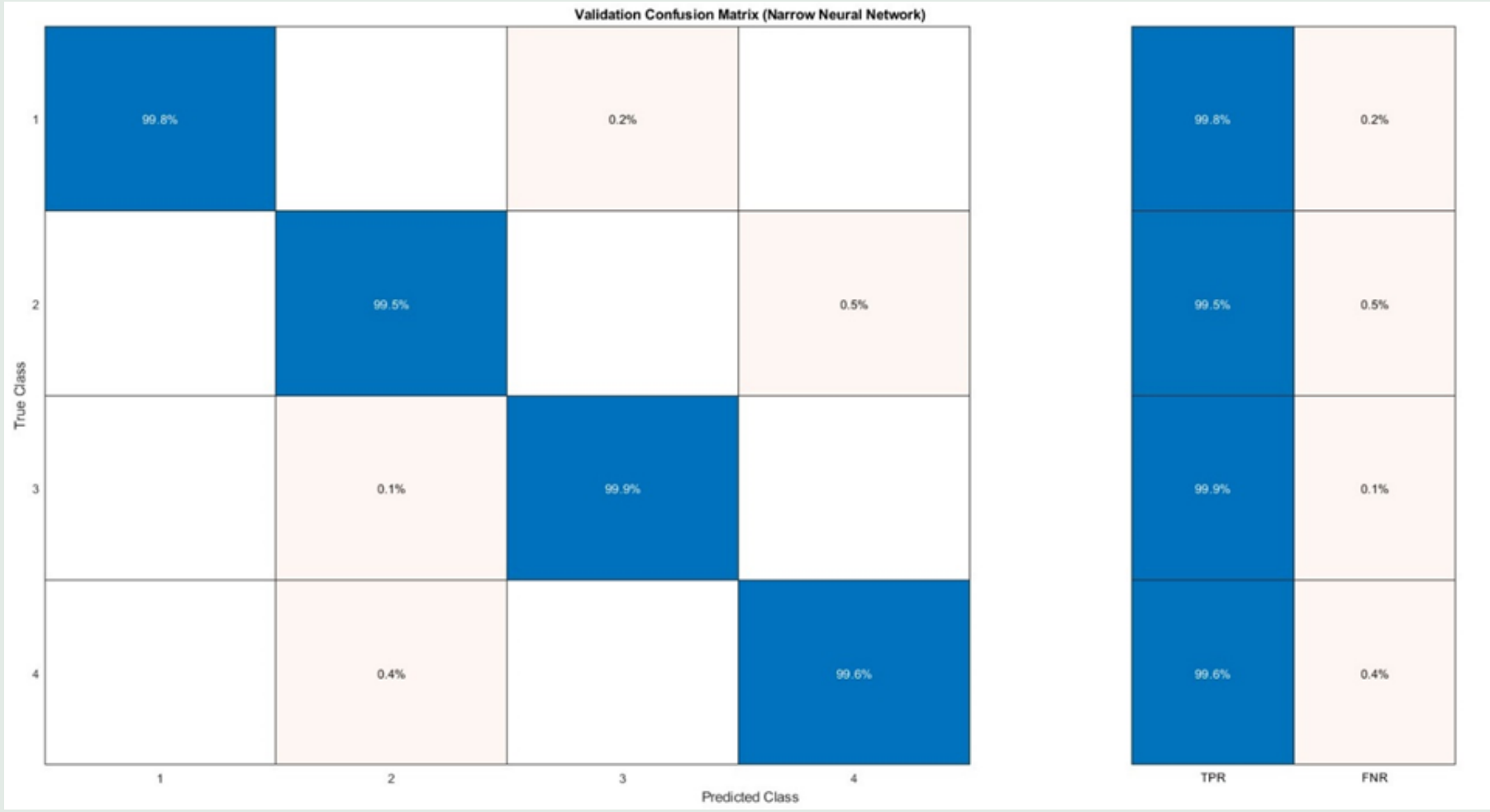
3. Confusion Matrix: Number of observations:



Data preparation

Step 8: Binary Classification by neural network:

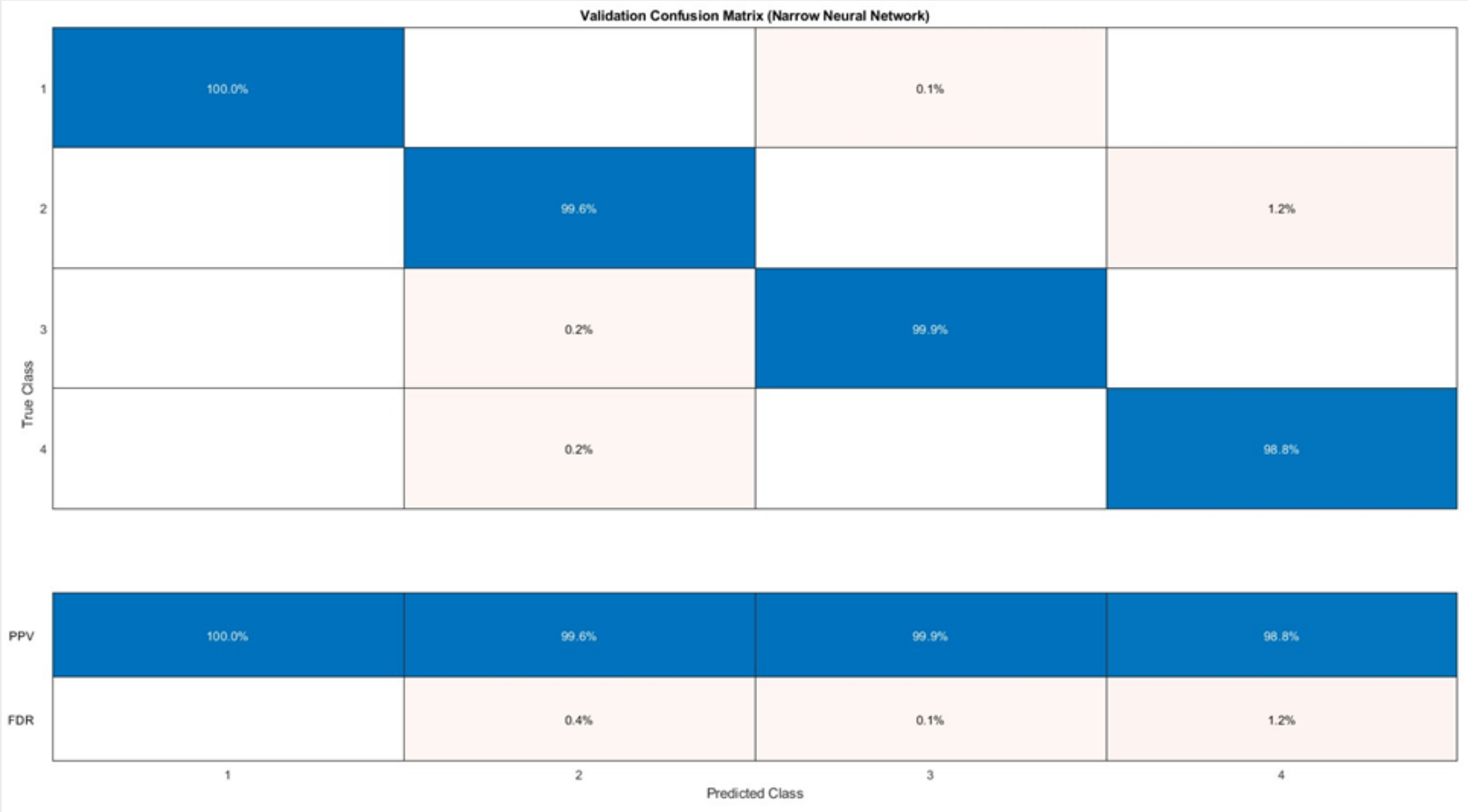
3. Confusion Matrix:



Data preparation

Step 8: Binary Classification by neural network:

3. Confusion Matrix:



Data preparation

Step 8: Binary Classification by neural network:

Note:

The classification matrix represents the relationship between true classes (rows) and predicted classes (columns). The diagonal cells represent instances where the true class and predicted class match. When these cells are highlighted in blue, it indicates that the classifier has correctly classified observations of that true class.

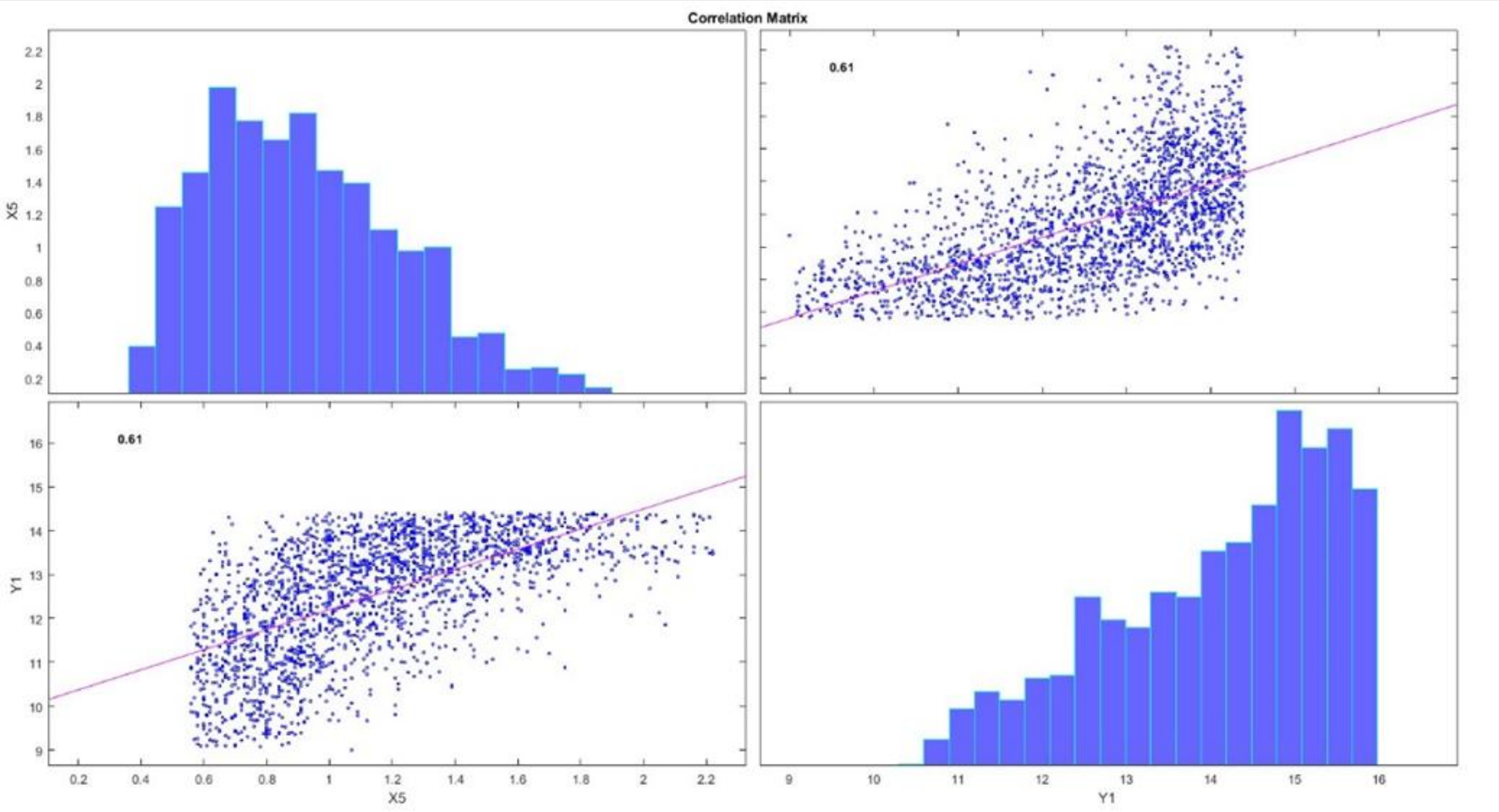
The default view of the matrix displays the number of observations in each cell, providing an overview of the classification performance.

To assess the classifier's performance for each class individually, there is an option to plot the True Positive Rates (TPR) and False Negative Rates (FNR) under the "Plot" section. TPR represents the proportion of correctly classified observations for each true class, while FNR represents the proportion of incorrectly classified observations for each true class. The plot summarizes the performance per true class in the last two columns on the right.

Data preparation

Step 8: Binary Classification by neural network:

Plot variable correlations between X5 and Y1, after classification:



Creating a Neural Network model

Comparison between cascade neural network and feedforward neuralnetwork:

```
% Step 1: Load your normalized dataset
load('all.mat', 'all'); % Replace 'all.mat' with the filename of your
normalized dataset
inputs = all(:, 1:5); % Assuming your normalized input data is stored
in columns 1 to 5
outputs = all(:, 6:7); % Assuming your normalized output data is
stored in columns 6 and 7

% Step 2: Split the dataset into training and testing sets
split_ratio = 0.8; % 80% for training, 20% for testing
num_samples = size(inputs, 1);
num_train_samples = round(split_ratio * num_samples);

train_inputs = inputs(1:num_train_samples, :)';
train_outputs = outputs(1:num_train_samples, :)';

test_inputs = inputs(num_train_samples+1:end, :)';
test_outputs = outputs(num_train_samples+1:end, :)';

% Step 3: Create and train different network architectures
hidden_units = [50, 30 25, 25]; % Number of hidden units in the
hidden layer
```

```
% Feedforwardnet
net_feedforward = feedforwardnet(hidden_units);
net_feedforward = train(net_feedforward, train_inputs,
train_outputs);
predicted_outputs_feedforward = net_feedforward(test_inputs)';
mse_feedforward = mean((predicted_outputs_feedforward -
test_outputs').^2);
mae_feedforward = mean(abs(predicted_outputs_feedforward -
test_outputs'));
r_squared_feedforward = 1 - (sum((predicted_outputs_feedforward
- test_outputs').^2) / sum((test_outputs' - mean(test_outputs')).^2));

% Cascadeforwardnet
net_cascade = cascadeforwardnet;
net_cascade = train(net_cascade, train_inputs, train_outputs);
predicted_outputs_cascade = net_cascade(test_inputs)';
mse_cascade = mean((predicted_outputs_cascade -
test_outputs').^2);
mae_cascade = mean(abs(predicted_outputs_cascade -
test_outputs'));
r_squared_cascade = 1 - (sum((predicted_outputs_cascade -
test_outputs').^2) / sum((test_outputs' - mean(test_outputs')).^2));
```

Creating a Neural Network model

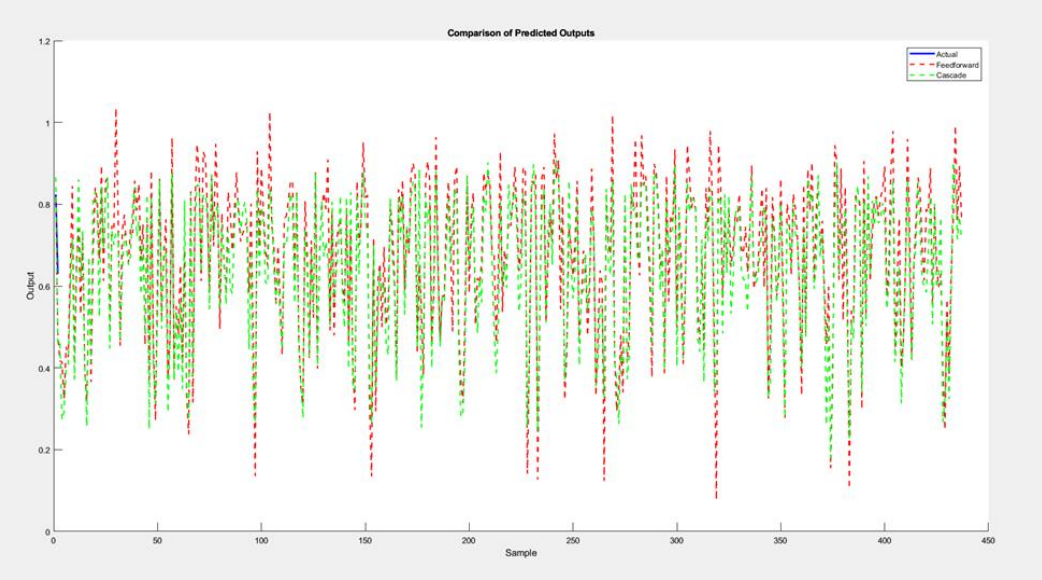
Comparison between cascade neural network and feedforward neural network:

```
% Step 4: Compare the performance of different network architectures
mse_values = [mse_feedforward, mse_cascade];
mae_values = [mae_feedforward, mae_cascade];
r_squared_values = [r_squared_feedforward, r_squared_cascade];
networks = {'Feedforward', 'Cascade'};

% Display the evaluation metrics for each network
disp('Evaluation Metrics:');
for i = 1:length(networks)
    disp(networks{i});
    disp(['MSE: ', num2str(mse_values(i))]);
    disp(['MAE: ', num2str(mae_values(i))]);
    disp(['R-squared: ', num2str(r_squared_values(i))]);
    disp(' ‘);
end

% Plot the predicted outputs for comparison
figure;
hold on;
plot(test_outputs(:, 1), 'b', 'LineWidth', 2);
plot(predicted_outputs_feedforward(:, 1), 'r--', 'LineWidth', 1.5);
plot(predicted_outputs_cascade(:, 1), 'g--', 'LineWidth', 1.5);
legend('Actual', 'Feedforward', 'Cascade');
xlabel('Sample');
ylabel('Output');
title('Comparison of Predicted Outputs');
hold off;
```

The result of the comparison:
Evaluation Metrics:

Feedforward ✗
MSE: 0.035438
MAE: 0.15107
R-squared: 0.44194

Cascade ✔
MSE: 0.01374
MAE: 0.089134
R-squared: 0.80565



Cascadeforwardnet:

Intruduction:

- CascadeForwardNet is a type of feedforward neural network architecture
- Used for pattern recognition and regression tasks
- Offers efficient training and improved generalization capabilities

Architecture

- Multilayer perceptron (MLP) network
- Consists of interconnected nodes called neurons
- Unique feature: Cascade structure
- Gradual addition of hidden layers during training

Cascadeforwardnet:

Training Process: Cascade Training

Step 1:

- Start with a small number of hidden layers
- Training performed on a subset of data
- Evaluation of performance using validation set
- If performance criteria met, freeze current hidden layers
- Otherwise, add a new hidden layer and repeat training

Training Process: Consolidation Training

- Step 2:
- Consolidate previously frozen hidden layers
- Train the network with expanded set of hidden layers
- Preserves learned knowledge from earlier stages
- Allows incorporation of additional hidden layers

Code: (cascadeforwardnet)

```
% Load the data
load('all.mat', 'all');
load('input.mat', 'input');
load('output.mat', 'output');

% Prepare the data
x = input';
t = output';

% Choose a Training Function
trainFcn = 'trainlm';  % Levenberg-Marquardt
backpropagation.

% Create a Fitting Network
hiddenLayerSize = [50, 30, 25, 25];
net = cascadeforwardnet(hiddenLayerSize, trainFcn);

% Set regularization parameters
net.performParam.regularization = 0.01; % Adjust
regularization strength (increase if overfitting)
```

```
% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Choose a Performance Function
net.performFcn = 'mse';  % Mean Squared Error

% Train the Network
[net, tr] = train(net, x, t);

% Test the Network
y = net(x);
e = gsubtract(t, y);
performance = perform(net, t, y);
```

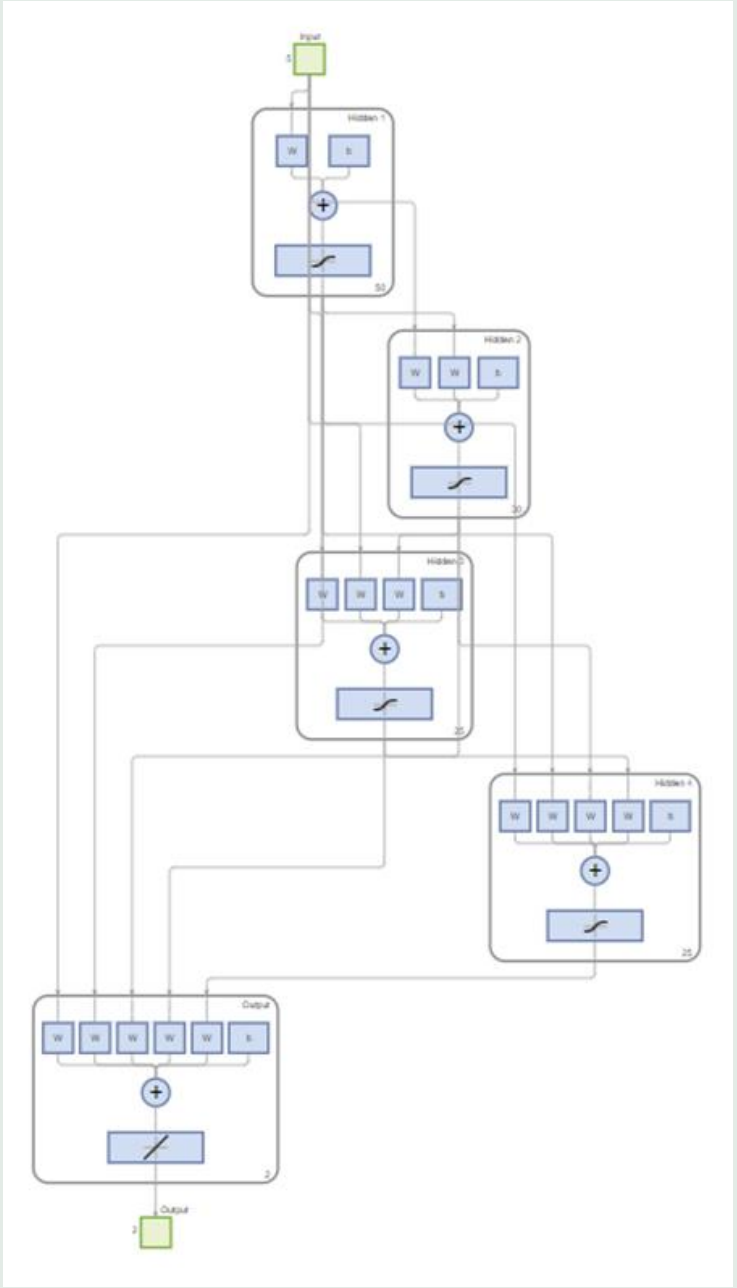
Code: (cascadeforwardnet)

```
% Display the mean squared error (MSE)
disp('Mean Squared Error (MSE):');
disp(['Cascade: ', num2str(testPerformance)]);

% Generate predictions using the trained network
test_inputs = x(:, tr.testInd);
predicted_outputs = net(test_inputs)';
actual_outputs = t(:, tr.testInd)';

% Plot the predicted outputs for comparison
figure;
hold on;
plot(actual_outputs(:, 1), 'b', 'LineWidth', 2);
plot(predicted_outputs(:, 1), 'g--', 'LineWidth', 1.5);
legend('Actual', 'Predicted');
xlabel('Sample');
ylabel('Output');
title('Comparison of Predicted Outputs');
hold off;
```

Result of the code

This diagram represents 4-layer which contain 50 neuron in first hidden layer, 30 neuron in second hidden layer, 25 neuron in third hidden layer and 25 neuron in final hidden layer.
The model that has 5 input variables and 2 output variables.

1. Network Diagram:



Result of the code

2. Training Results:

Training Results

Training finished: Met validation criterion

Training Progress

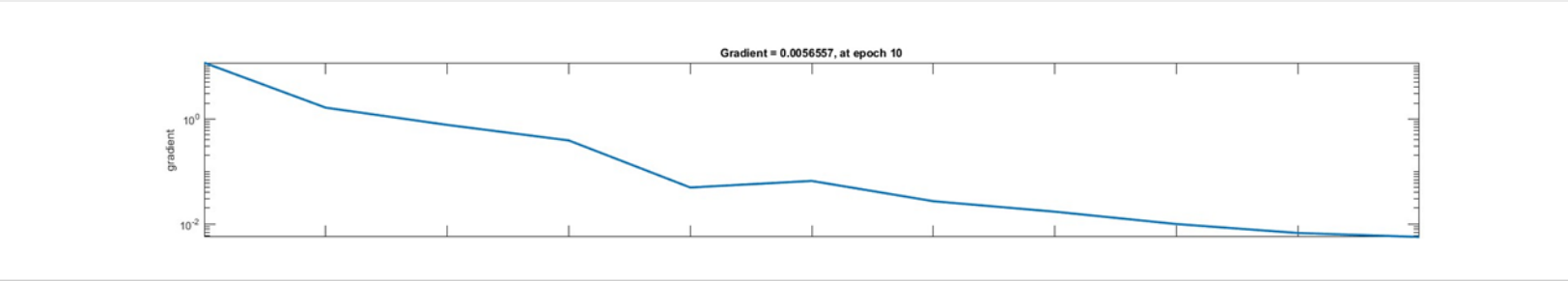
Unit	Initial Value	Stopped Value	Target Value
Epoch	0	10	1000
Elapsed Time	-	00:06:05	-
Performance	1.9	0.0133	0
Gradient	11.7	0.00566	1e-07
Mu	0.001	0.0001	1e+10
Validation Checks	0	6	6

2. Training Results

- From 0 to 1000 epochs, with 10 iterations, the lowest MSE has been reached in this model.
- In Performance, it is explained that at the beginning of the work, the error index was 1.9, which reached 0.0133 in the best case, which is close to zero, which means that the proposed model has a good performance in terms of fitting errors.
- The initial gradient value of 11.7 indicates a large magnitude of gradients at the beginning of training. This suggests a significant rate of change in the loss function concerning the network's weights. This is expected in the early stages when the network parameters are far from optimal.

The current gradient value of 0.00566 indicates a reduction in gradient magnitude after several iterations or epochs of training. A smaller gradient value signifies a decreased rate of change in the loss function concerning the weights. This suggests that the network parameters are getting closer to or converging towards a local minimum of the loss function.

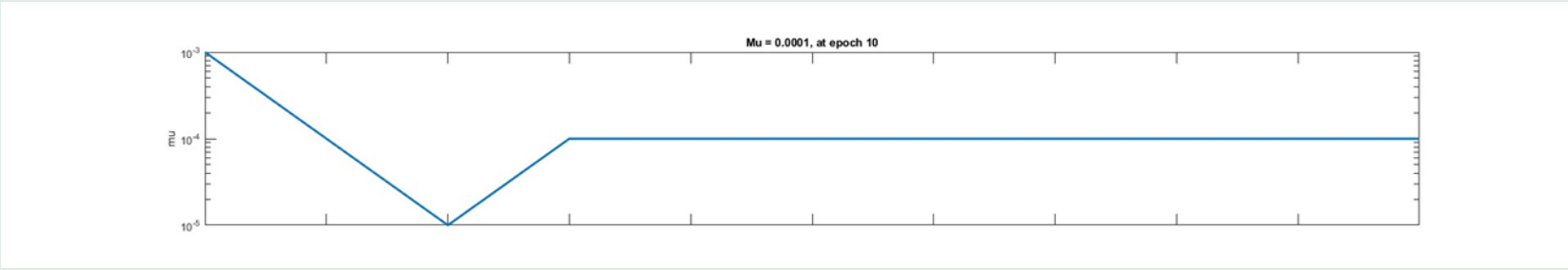
The decrease in gradient magnitude from the initial value to the current value indicates that the network weights have been adjusted during training, resulting in smaller updates over time. This can be viewed as progress in optimizing the network's performance.



2. Training Results

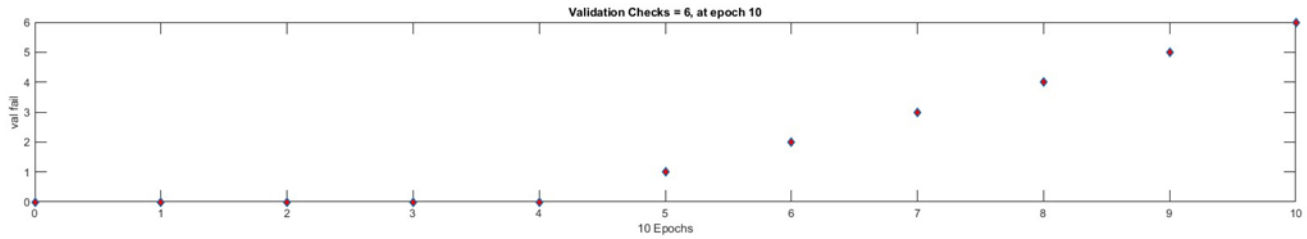
- the parameter "Mu" typically refers to the momentum term. Momentum is a hyperparameter used in gradient-based optimization algorithms, such as gradient descent with momentum, to accelerate the convergence and overcome local minima.

the parameter "Mu", starting with Mu = 0.001 and gradually reducing it to Mu = 0.0001 indicates a decreasing momentum schedule. This means that during the initial stages of training, the momentum term has a larger impact on the weight updates, leading to more substantial changes. As training progresses, the momentum term is gradually reduced, resulting in smaller updates and potentially finer adjustments to the weights.

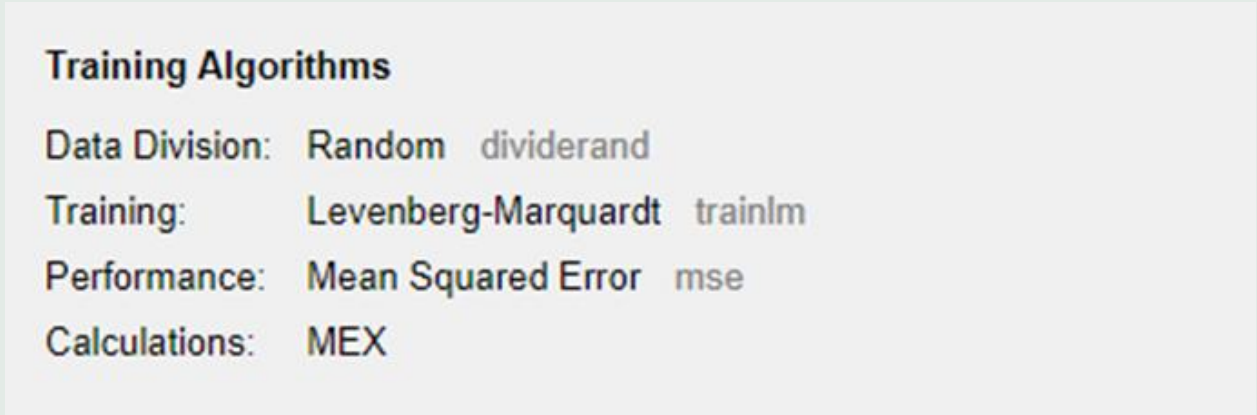


2. Training Results

- In the validation check, we notice that the model is stopped after 6 times of failure in training. By stopping the training after a certain number of validation failures, the early stopping mechanism helps prevent the model from memorizing the training data too closely and allows it to find a better balance between fitting the training data and generalizing to unseen data. This can lead to improved performance on new, unseen examples.



3. **Training Algorithms**

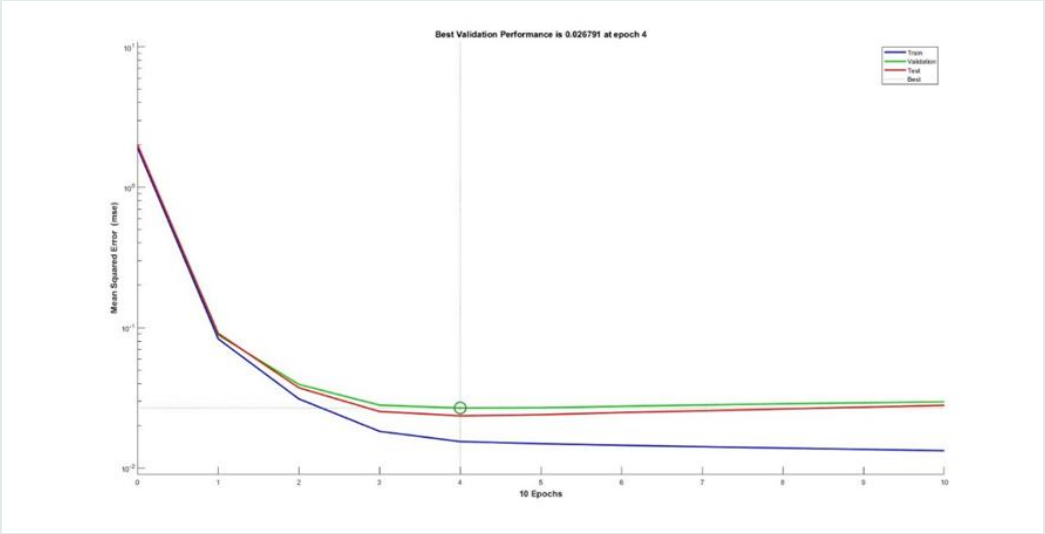


It shows the algorithm used in which the data is randomly taken from all the sample space and does not belong to a specific section. Also, LM algorithm has been used to train MLP neural network according to the volume of samples.
Error performance index Performance is considered according to the type of neural network, the fitting of errors.

4. Training plots

Performance:

The performance plot shows the value of the performance function (mean squared error) versus the number of iterations.



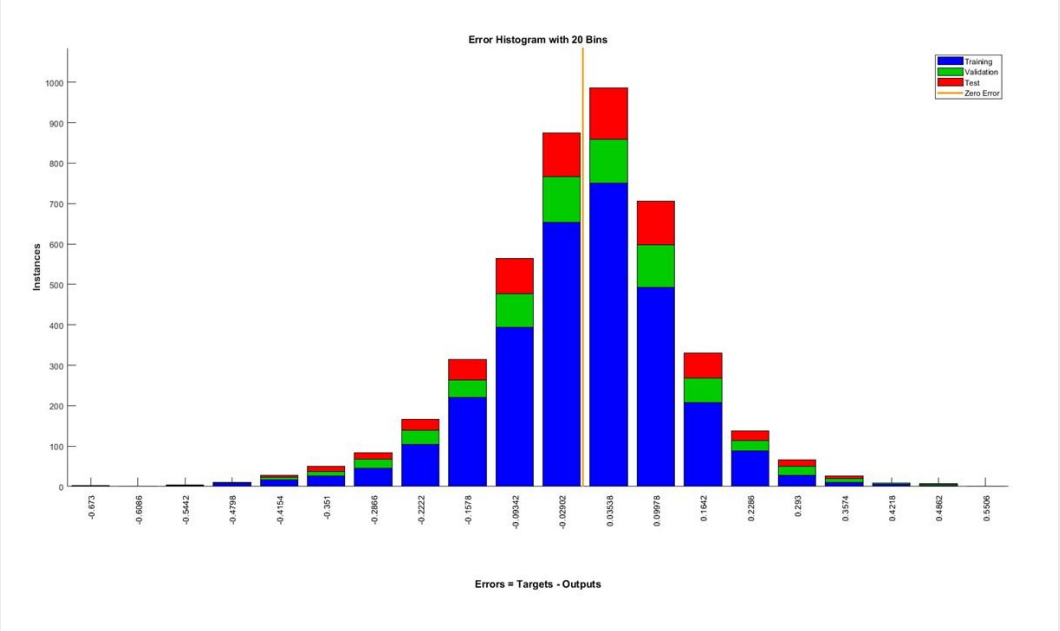
A performance chart examines how to reduce errors. The value of MSE in the last iteration is small enough.

And according to this review, the best performance is at epoch=4. Because in this value, all 3 graphs have an acceptable performance and have relatively the same features in the model.

No more fitting has occurred from 6 repetitions onwards after 4 repetitions.

4. Training plots

Error Histogram with 20 Bins

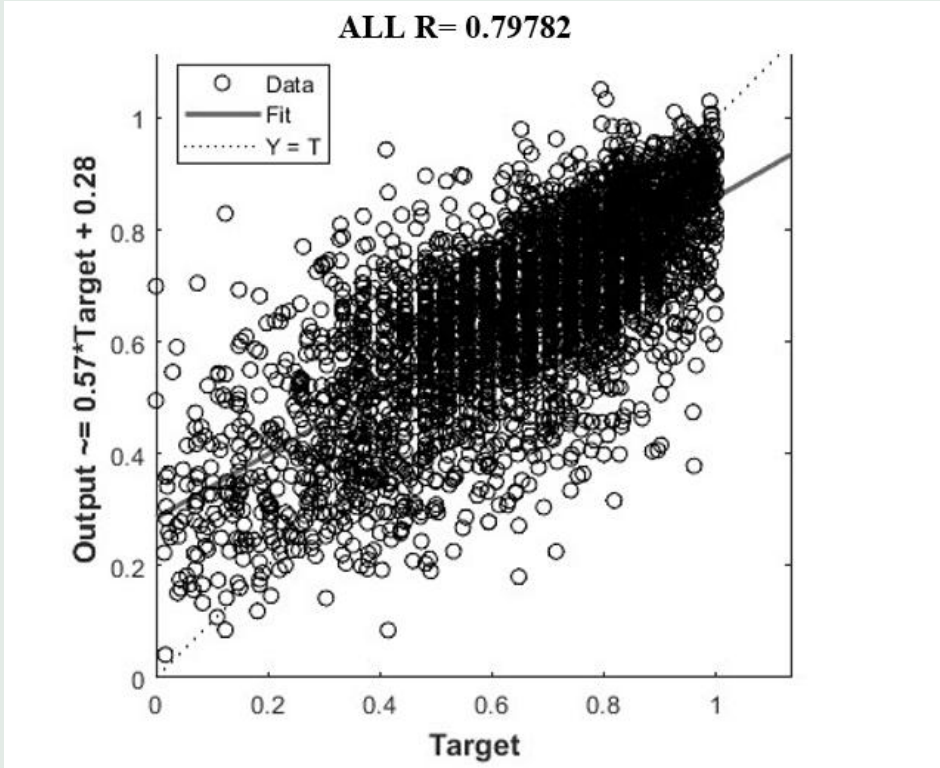


Error Histogram shows the distribution of errors. The error curve around zero (-0.2 and 0.2) is normal, which indicates low error and variance.

4. Training plots

- **Regression:**

This diagram shows the validation of the network based on the regression diagram of total data, which shows the relationship between the network output and the goals



4. Training plots

A plot of predicted versus observed values is useful in assessing the validity of the network. In an ideal scenario, where the network performs well, the regression line on the plot should have an angle of 45 degrees.

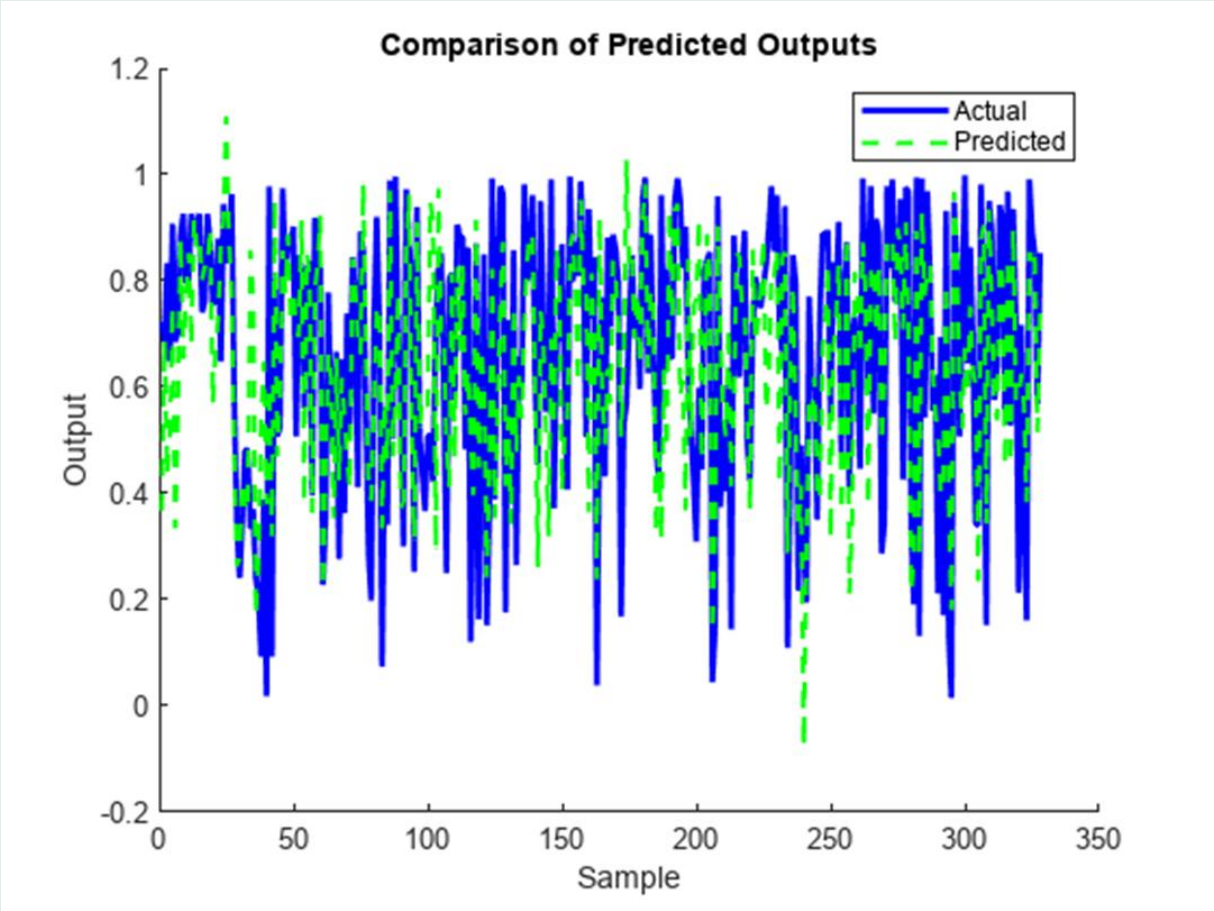
The plot can also be used to examine other indicators of the model's quality, such as the coefficient of determination (R). R represents the linear relationship between the output values and the targets. A value of 1 indicates a perfect linear relationship, while 0 indicates no linear relationship. Therefore, a higher R value suggests a better fit and a relatively strong linear relationship between the output values and the targets.

In practice, achieving perfect training where the output of the network exactly matches the target is rare. The dotted lines on the plot represent this perfect training scenario, which rarely occurs. The bold lines represent the best fitting regression line between the outputs and targets.

If the quality of the network is poor, retraining can be done by selecting the "Retrain" option. This involves changing the initial weights of the network and adjusting the amount of skewness.

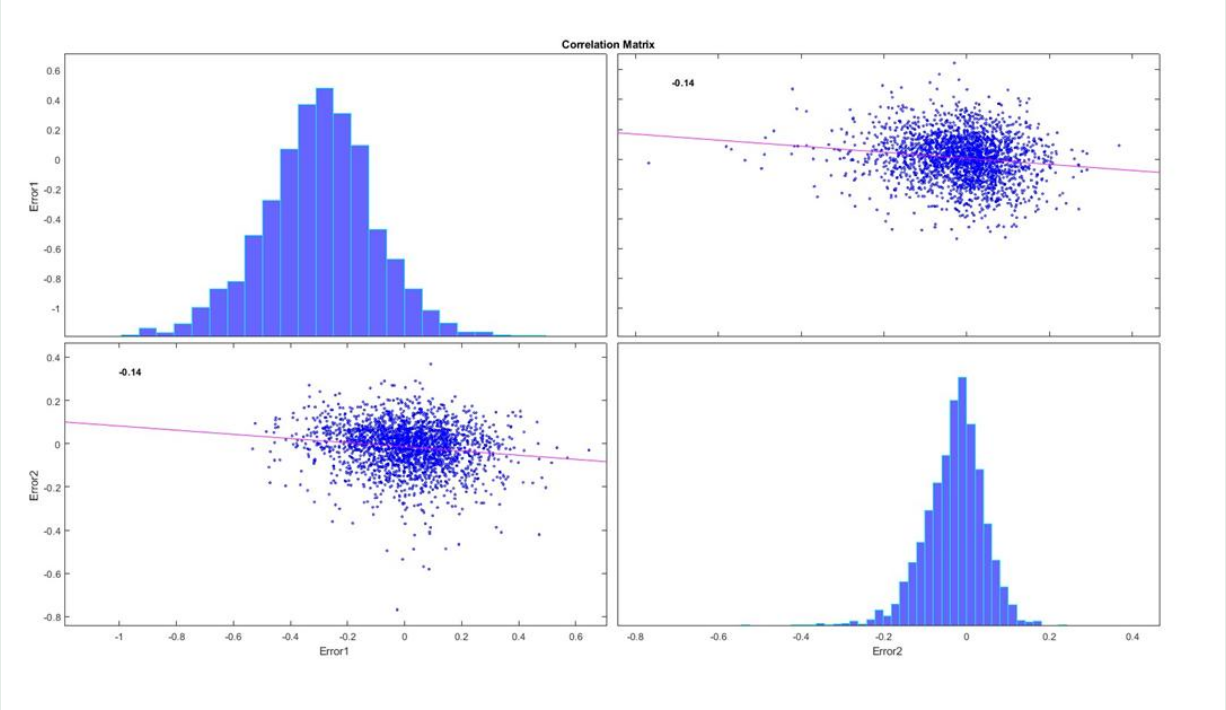
In the trained neural network model, the data shows relatively high correlation, but none of the regression lines in the plots have an angle of 45 degrees. This indicates a high variance between the data points and the regression line, suggesting the presence of outlier data in the dataset.

5. Comparison of predicted output



A summary of how to create a multivariate control chart and how to use it

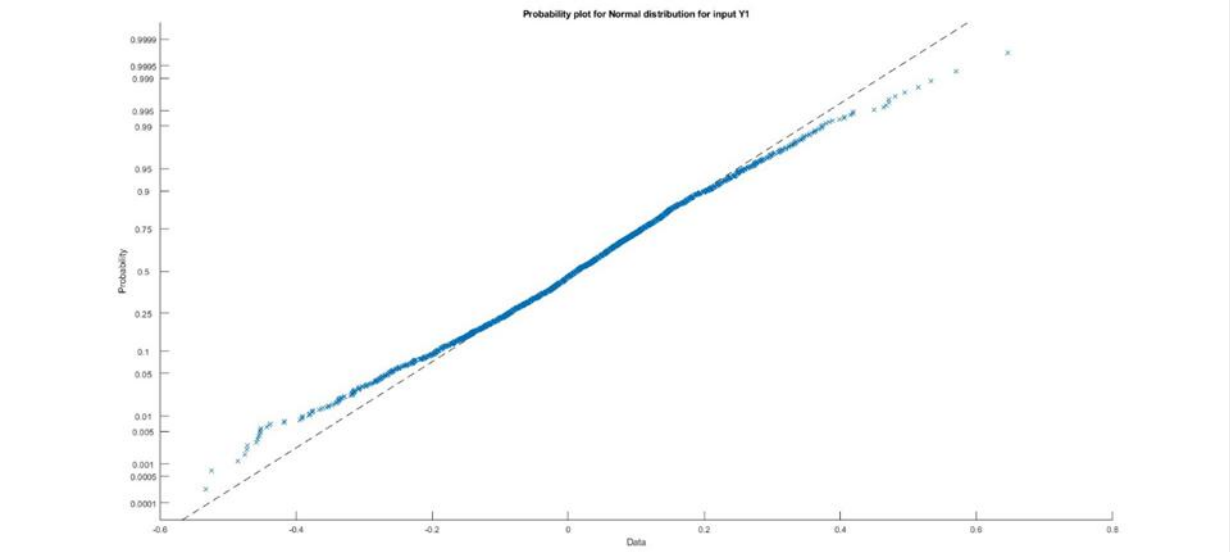
Checking Residuals correlation :



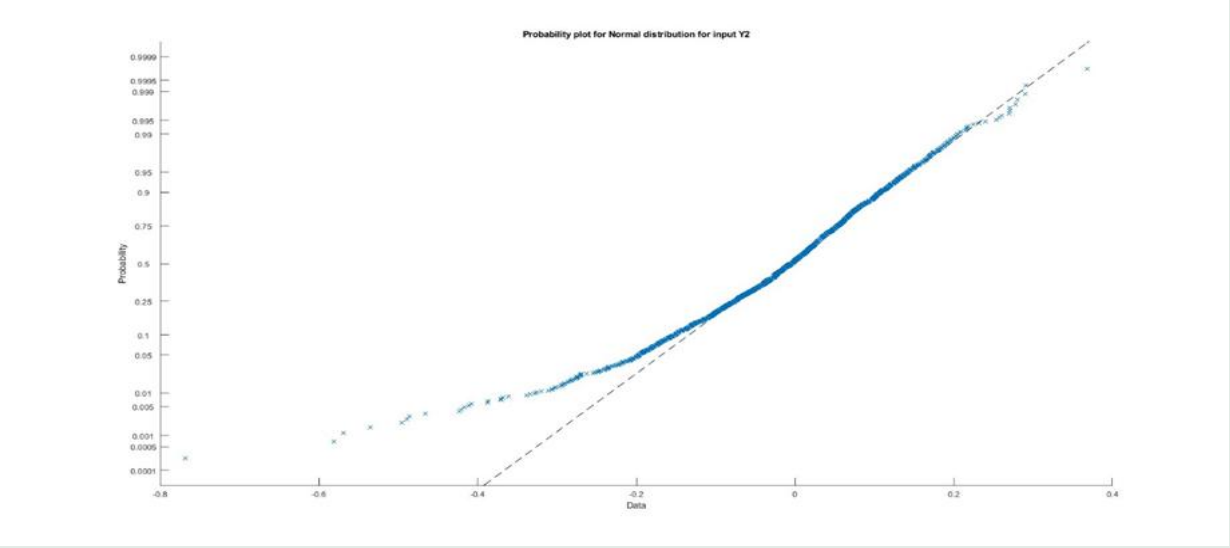
According to the fact that the correlation coefficient is -0.14 and it is considered a number close to zero, and the p-value is equal to 0.52, we can conclude that the errors are independent from each other and there is no significant relationship between the two output errors and it should be Check which one separately.

With the help of Probability Plot, we get the density function of **error outputs**:

Y1:



Y2:



According to the obtained information, because the average in both values is very close to zero and the errors are independent from each other, we can use Shewhart and non-Shewhart control charts such as EWMA to obtain warning control limits.
In this research, since we have more than one variable under investigation and we also need to quickly identify and discover small changes in the process, it is better to use the EWMA control chart.

UCL= μ_"0" +Lσ√((λ/"2-λ"))
LCL= μ_"0" -Lσ√((λ/"2-λ"))

Control chart for error output Y1:

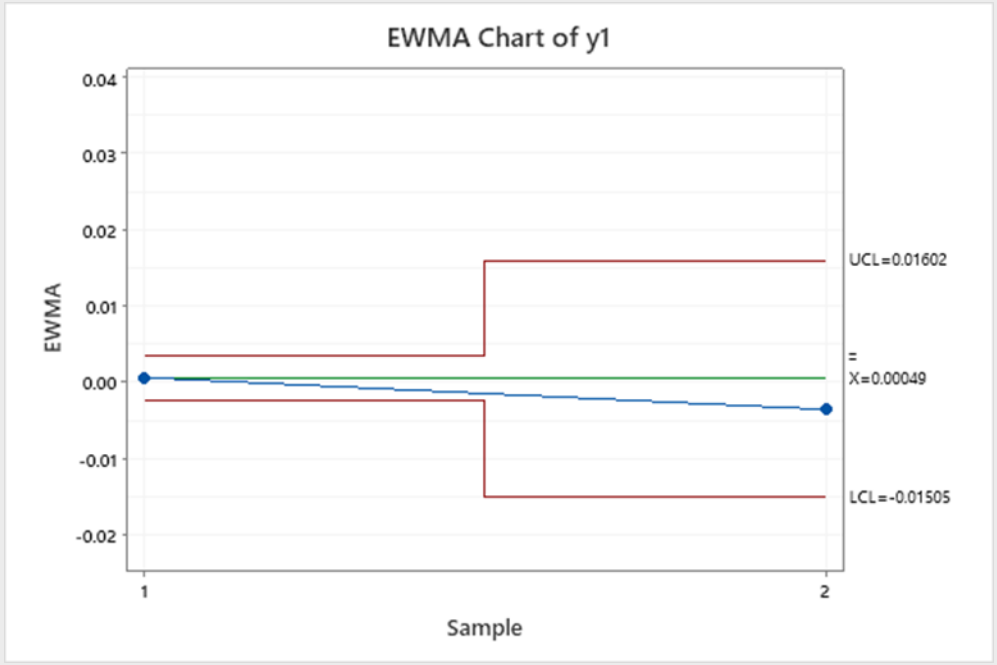
UCL= 0.01602
$\bar{x}$=0.00049
LCL=-0.01505

Control chart for error output Y2:

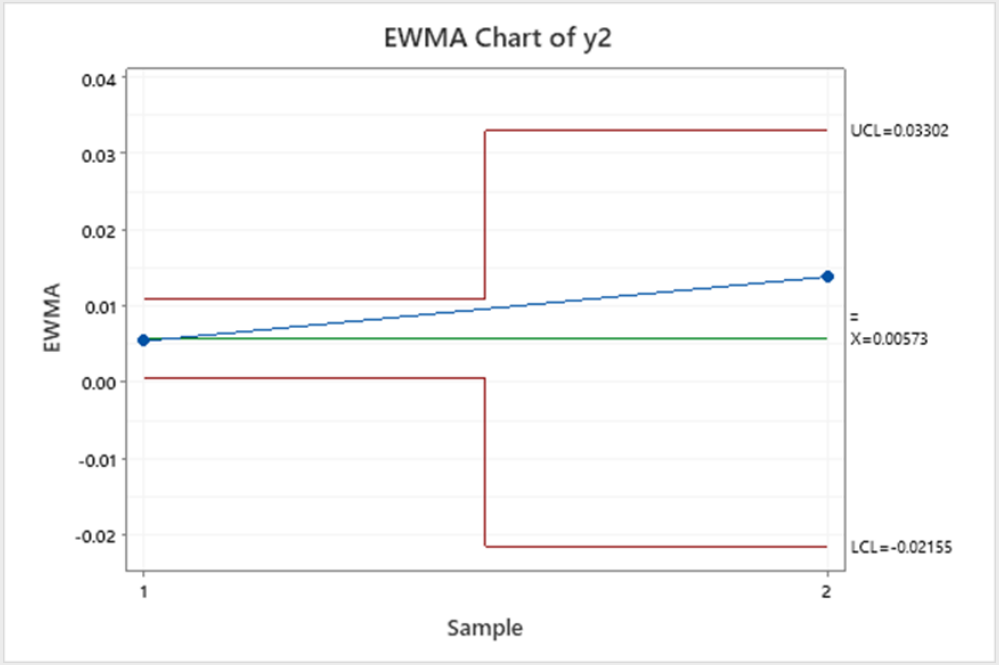
UCL= 0.03302
$\bar{x}$=0.00573
LCL=-0.02155

Control Chart

Y1:



Y2:



The control chart is made in two phases as follows

Phase I:

In Phase I, the objective is to obtain a set of controlled data. Control charts are used to determine if the data is related to a process that is statistically under control. Historical data is used in Phase I to assess the process's dispersion over time, evaluate its stability, and estimate the parameters of the controlled process. The main goal is to establish a process that is statistically under control.

Phase II:

In Phase II, new data is analyzed based on the results obtained in Phase I. Assuming that the process was under control in Phase I, the goal is to check if the process remains under control for subsequent observations. Control charts are used to monitor and identify any shifts or trends in the process based on the control limits established in Phase I. The performance of control charts in Phase II is often measured by the average sequence length, which is the number of samples taken before an out-of-control warning is observed. The control limits obtained from Phase I are used to assess the process's characteristics and determine if it is under control.

It is crucial to identify any trends, shifts, deviations, or unusual points in the data during Phase I, as they can impact the control limits for Phase II. Calculating control limits based on data from an unstable or out-of-control process can decrease the accuracy and efficiency of the control chart procedure in Phase II. Therefore, detecting and addressing any unusual points before calculating the control limits is essential for the effectiveness of the control chart process.

Research overview