CS5014

ID : 180029410

Report for the Practical 1

Introduction

The aim of the practical is to build a model which takes 8 inputs and gives an output of 2 variables. The model predicts Cooling and Heating Load based on the parameters of the building. All buildings have the same volume but different dimensions, are built from the same material. Input values are: "Relative Compactness", "Surface Area", "Wall Area", "Roof Area", "Overall Height", "Orientation", "Glazing area", "Glazing area distribution". In the report and in the code naming (x1, x2, x3.., y1, y2) is used for simplicity.

- X1 = Relative Compactness
- X2 = Surface Area",
- X3 = "Wall Area
- X4 = Roof Area",
- X5 = "Overall Height
- X6 = Orientation
- X7 = Glazing area
- X8 = Glazing area distribution
- Y1 = Cooling
- Y2 = Heating

Procedure

The code is written in Python 3.72 (Worked on lab machines), using OOP design structure. Contains main file which runs all application, also Data class which is responsible for the data processing and train which is responsible for training. Samples were all correct, so there was no need to delete any row.

Loading data

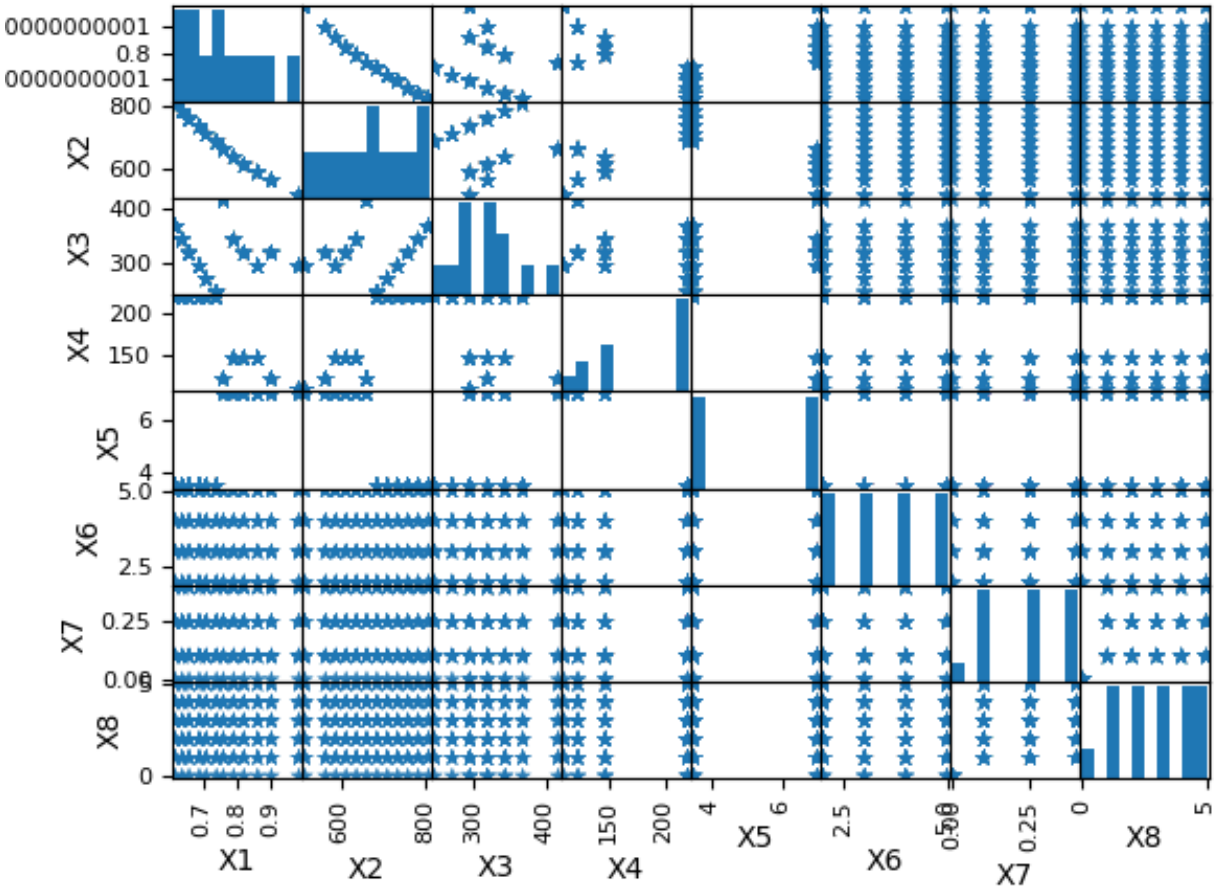
Data is loaded in the main file, in the method called *loadData()*. As the data was in csv file, it was read using pandas *read_csv* function.

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28
5	0.90	563.5	318.5	122.50	7.0	3	0.0	0	21.46	25.38
6	0.90	563.5	318.5	122.50	7.0	4	0.0	0	20.71	25.16
7	0.90	563.5	318.5	122.50	7.0	5	0.0	0	19.68	29.60
8	0.86	588.0	294.0	147.00	7.0	2	0.0	0	19.50	27.30
9	0.86	588.0	294.0	147.00	7.0	3	0.0	0	19.95	21.97
10	0.86	588.0	294.0	147.00	7.0	4	0.0	0	19.34	23.49
11	0.86	588.0	294.0	147.00	7.0	5	0.0	0	18.31	27.87
12	0.82	612.5	318.5	147.00	7.0	2	0.0	0	17.05	23.77
13	0.82	612.5	318.5	147.00	7.0	3	0.0	0	17.41	21.46
14	0.82	612.5	318.5	147.00	7.0	4	0.0	0	16.95	21.16
15	0.82	612.5	318.5	147.00	7.0	5	0.0	0	15.98	24.93
16	0.79	637.0	343.0	147.00	7.0	2	0.0	0	28.52	37.73
17	0.79	637.0	343.0	147.00	7.0	3	0.0	0	29.90	31.27
18	0.79	637.0	343.0	147.00	7.0	4	0.0	0	29.63	30.93
19	0.79	637.0	343.0	147.00	7.0	5	0.0	0	28.75	39.44
20	0.76	661.5	416.5	122.50	7.0	2	0.0	0	24.77	29.79
21	0.76	661.5	416.5	122.50	7.0	3	0.0	0	23.93	29.68
22	0.76	661.5	416.5	122.50	7.0	4	0.0	0	24.77	29.79
23	0.76	661.5	416.5	122.50	7.0	5	0.0	0	23.93	29.40
24	0.74	686.0	245.0	220.50	3.5	2	0.0	0	6.07	10.90
25	0.74	686.0	245.0	220.50	3.5	3	0.0	0	6.05	11.19
26	0.74	686.0	245.0	220.50	3.5	4	0.0	0	6.01	10.94
27	0.74	686.0	245.0	220.50	3.5	5	0.0	0	6.04	11.17
28	0.71	710.5	269.5	220.50	3.5	2	0.0	0	6.37	11.27
29	0.71	710.5	269.5	220.50	3.5	3	0.0	0	6.40	11.72
..	...	...	...	...	...	..	...	..	...	...

Data processing

After the data is read it needs to be processed and cleaned if needed, and this is done in *Data.py* class. Initialiser of the Data class saves and stores dataframe object generated by Pandas, also it splits input and output and stores their values in numpy array, which is used later for plotting graphs.

To see the all picture, scatter matrix is plotted for each feature using pandas scatter_matrix function:

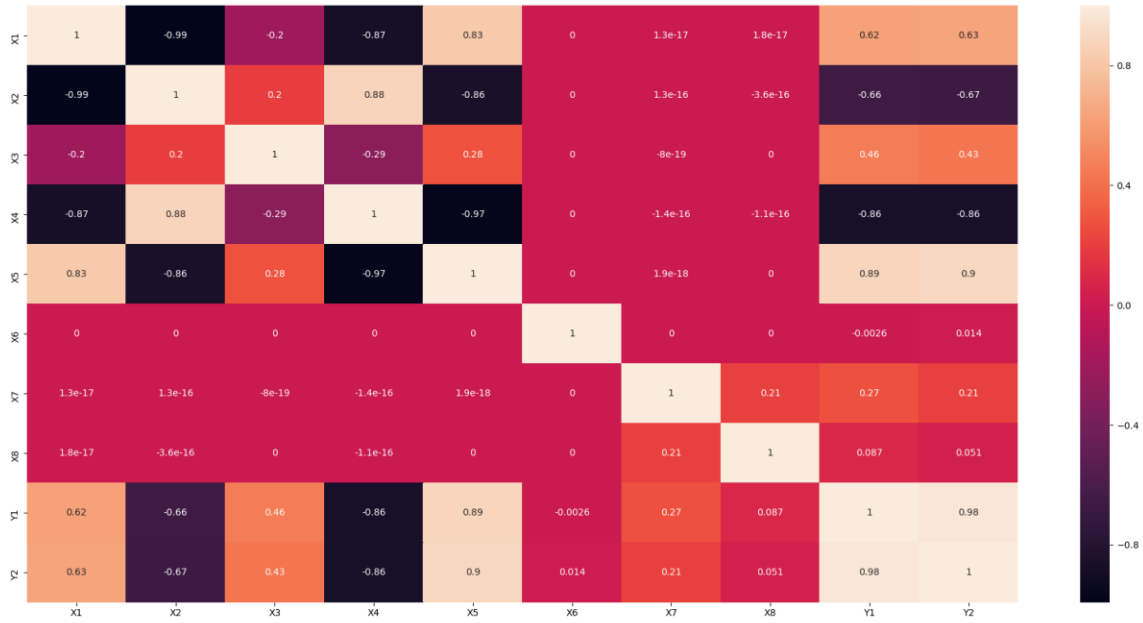


After the analysis it was achieved that the X1 and X2 have some patter in common, as they illustrate the same graph but in reflection.

After some analysis it is identified that some features, particularly x1 or x2 and x4 or x5 can be dropped, because they are correlated. This is possible from the correlation matrix.

```
[768 rows x 10 columns]
              X1            X2            X3            X4            X5        X6            X7            X8        Y1        Y2
X1  1.000000e+00 -9.919015e-01 -2.037817e-01 -8.688234e-01  8.277473e-01  0.000000  1.283986e-17  1.764620e-17  0.622272  0.634339
X2 -9.919015e-01  1.000000e+00  1.955016e-01  8.807195e-01 -8.581477e-01  0.000000  1.318356e-16 -3.558613e-16 -0.658120 -0.672999
X3 -2.037817e-01  1.955016e-01  1.000000e+00 -2.923165e-01  2.809757e-01  0.000000 -7.969726e-19  0.000000e+00  0.455671  0.427117
X4 -8.688234e-01  8.807195e-01 -2.923165e-01  1.000000e+00 -9.725122e-01  0.000000 -1.381805e-16 -1.079129e-16 -0.861828 -0.862547
X5  8.277473e-01 -8.581477e-01  2.809757e-01 -9.725122e-01  1.000000e+00  0.000000  1.861418e-18  0.000000e+00  0.889431  0.895785
X6  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  1.000000  0.000000e+00  0.000000e+00 -0.002587  0.014290
X7  1.283986e-17  1.318356e-16 -7.969726e-19 -1.381805e-16  1.861418e-18  0.000000  1.000000e+00  2.129642e-01  0.269841  0.207505
X8  1.764620e-17 -3.558613e-16  0.000000e+00 -1.079129e-16  0.000000e+00  0.000000  2.129642e-01  1.000000e+00  0.087368  0.050525
Y1  6.222722e-01 -6.581202e-01  4.556712e-01 -8.618283e-01  8.894307e-01 -0.002587  2.698410e-01  8.736759e-02  1.000000  0.975862
Y2  6.343391e-01 -6.729989e-01  4.271170e-01 -8.625466e-01  8.957852e-01  0.014290  2.075050e-01  5.052512e-02  0.975862  1.000000
```

This is more visible from the plotted correlation matix, which is plotted using Seaborn library:



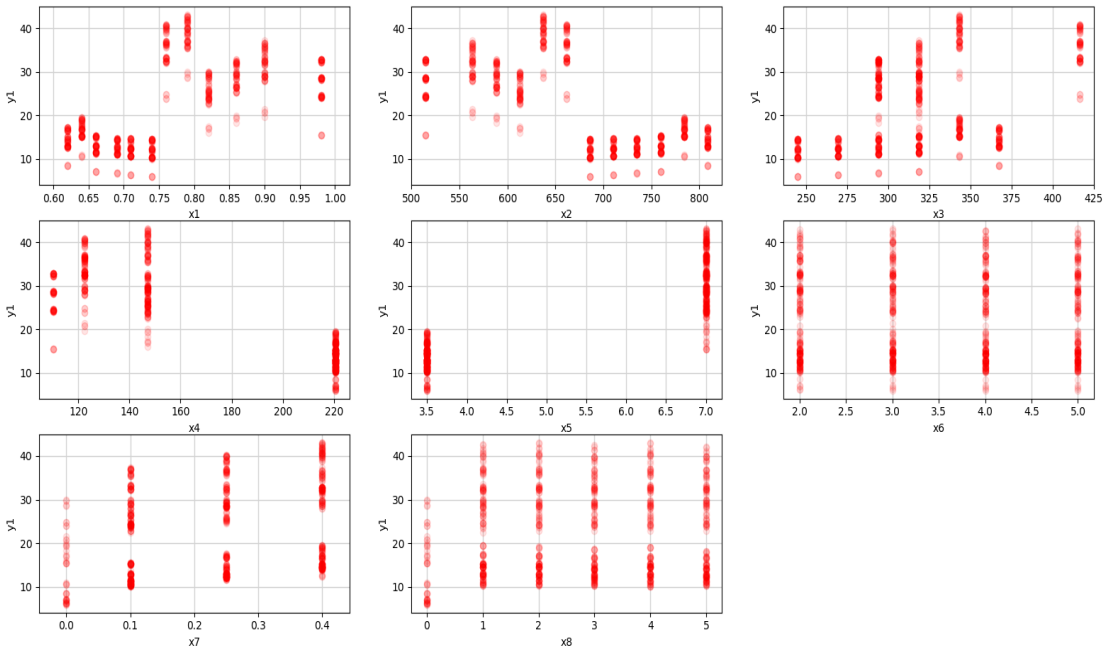
From the figure, it is obvious that X1 and X2 (Relative Compactness, Surface Area) are correlated. Same with the values of X4 and X5(Roof Area, Overall Height). Both variables are inversely correlated, which makes since, as the volume is constant, the roof area should get smaller when the overall height increases.

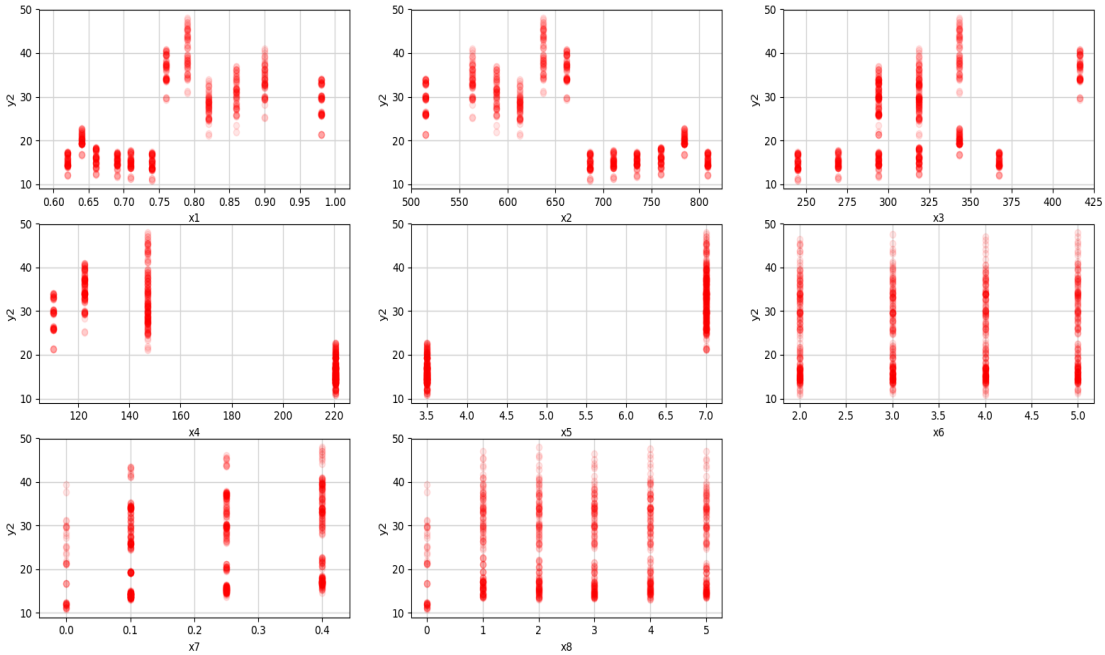
Split the data

To train the model the data was split into training and testing, using pandas train_test_split method. The method splitDataToTrainAndTest() returns the tupple of 4 variables: train_x, train_y, test_x, test_y. For testing 35% of all dataset was taken, using th pandas train_test_split method, which picks random data points from dataset. There was no need of the validation set till some point.

Training

First training method is Linear Regression. If plot all input-output graphs, the linear regression is not the best model for this dataset. OBJ


y1
x1
x2
x3
x4
x5
x6
x7
x8


y2
x1
x2
x3
x4
x5
x6
x7
x8

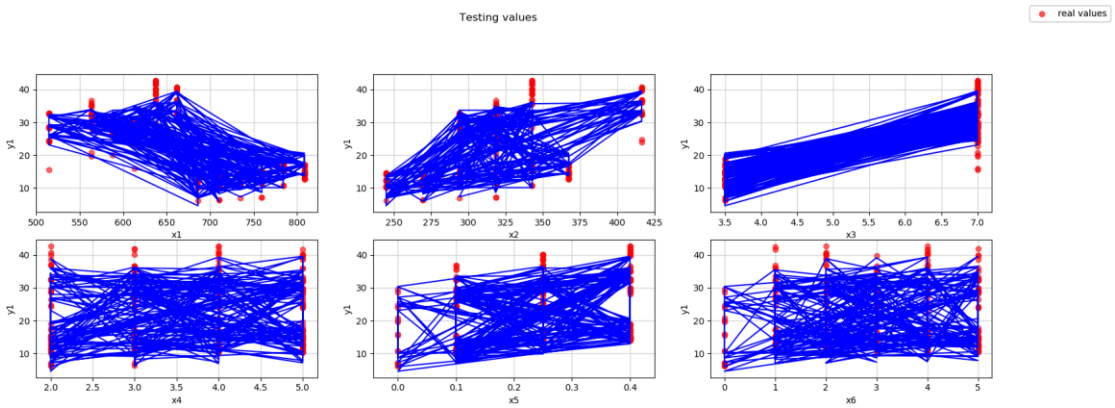
The graphs show that linear regression will not give very accurate result. Moreover, the graphs for Y1 and Y2 are very similar, this can be justified by correlation matrix from the section before, as both outputs are linearly correlated. Hence, the results for both outputs will be similar after each training.

Linear Regression

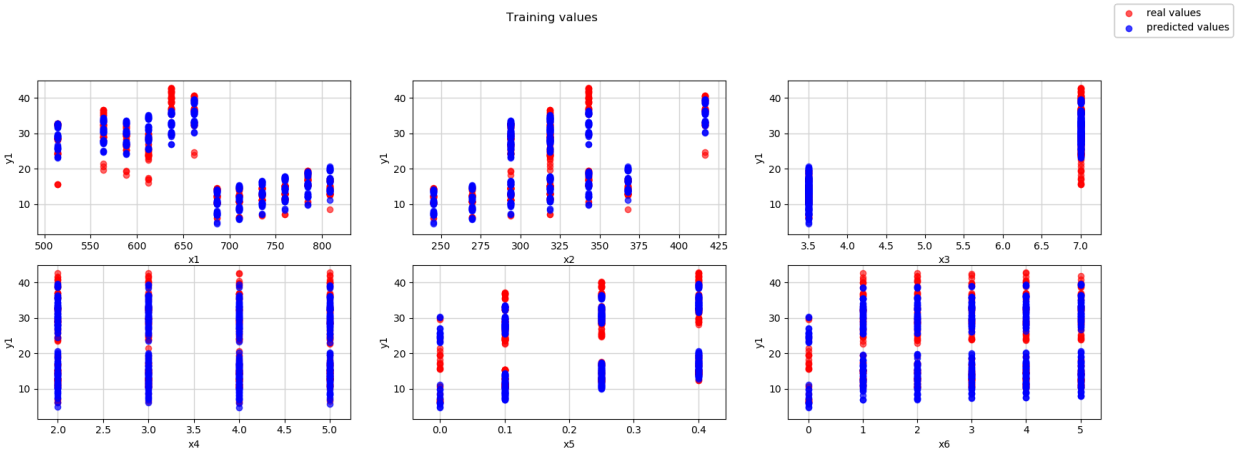
Mean squared error for the linear regression is : 9.44.

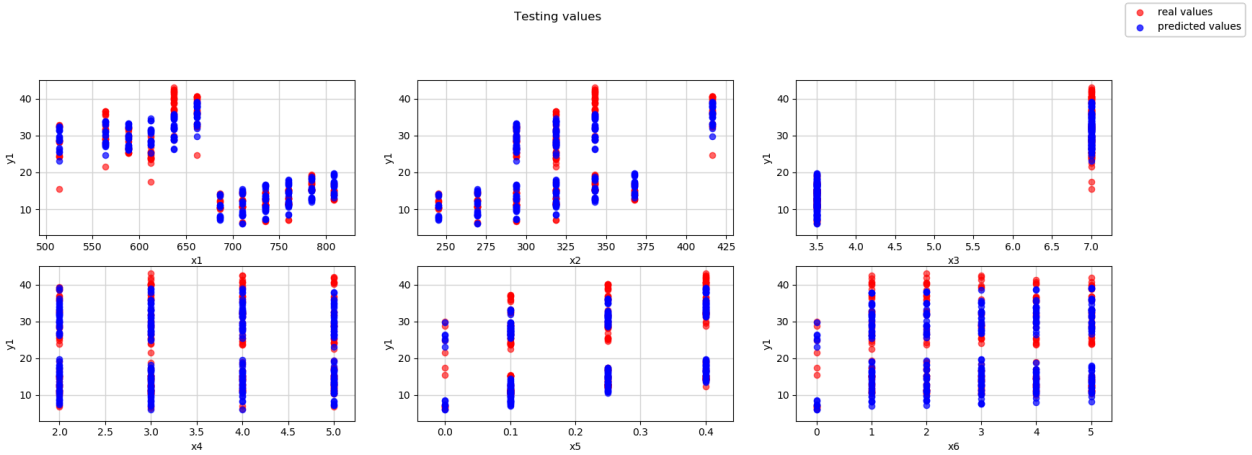
Accuracy acquired using r2_score was 89%.

To see the model, it was plotted on top of the scatter plot of the dataset. However, as there are lots of inputs, it is not obvious how does the model look like in 6 dimensions.



Due to this, another graph was plotted, scatter plot, where the actual datapoints and predicted ones are on the same graph. So ,it is obvious if the model is doing everythin right.





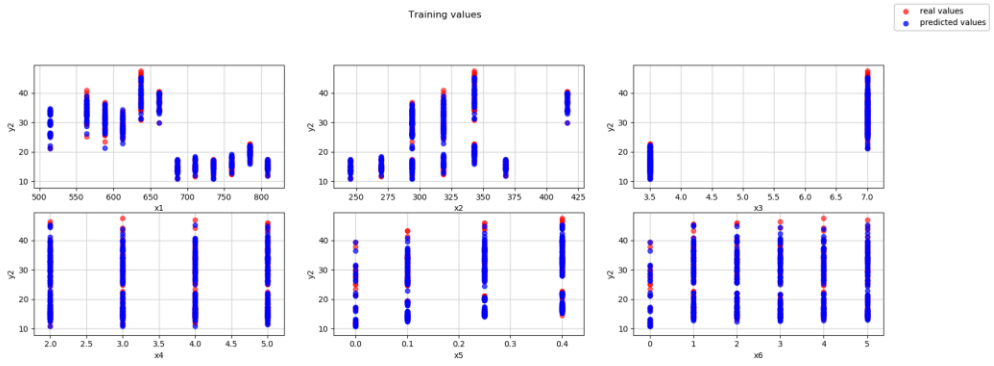
In fact, the accuracy of the trainig data (seen data) should be higher, but as soon as the model is not doing well, both of the graphs achieve medium quality. It is obvious that in some points model cannot predict some data.

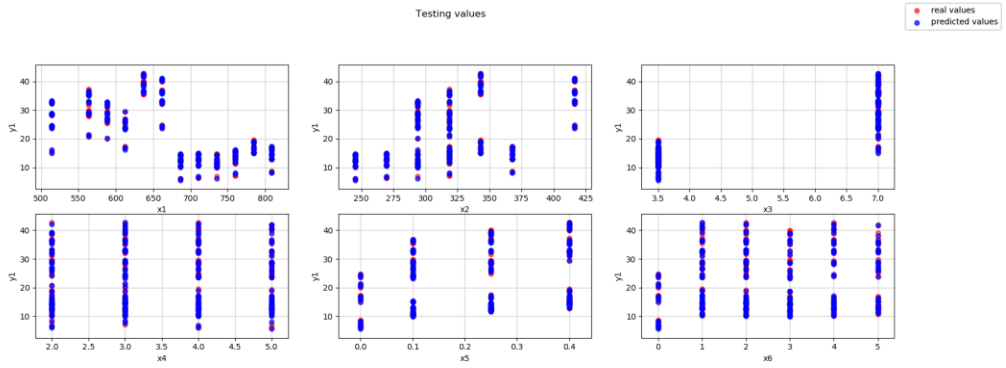
Polynomial regression

Unfortunately, it is not possible to visualize all inputs together and we do not see if the data follows the linear pattern. As soon as the linear regression gives high error, it is possible to increase the order of the equation and make it polynomial regression.

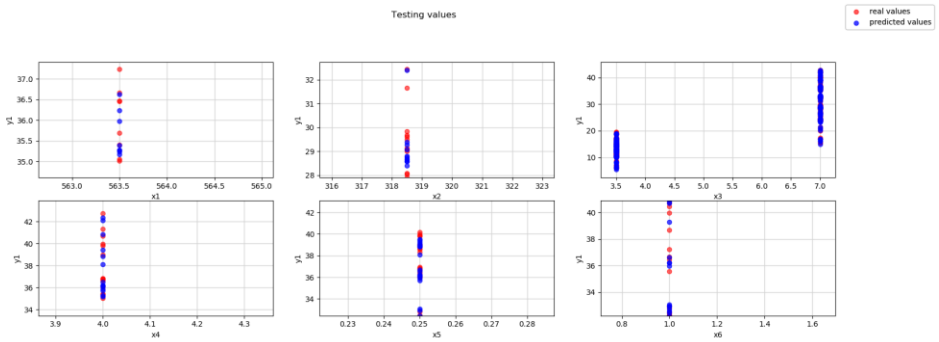
In case of polynomial regression, it was required to choose the order of the polynomial, for simplicity it was assigned to 4 at the beginning. By transforming their input data into the higher order polynomial, this was performed again using linear regression.

Polynomial regression with order 4 gave the MSE of 1.439, which is almost 7 times better.





Here, it is more visible that the model does worse on the testing set, however it is also obvious that it predicts much better than the linear regression of the order 1.



If zoom the testing data input output graphs, the model will look is not so accurate.

Finding the best order for the polynomial

Order 4 predicted everything very well, but is it enough, or is it the maximum performance, is still a question. To find this out, simple for loop was written which iterated over polynomial orders starting from 1 to 9.

```
Order-  1  MSE =  9.986488122959162
Order-  2  MSE =  5.822005585394419
Order-  3  MSE =  2.7748643563871305
Order-  4  MSE =  1.4196683506454724
Order-  5  MSE =  1.950703740269009
Order-  6  MSE =  2.4250203786189397
Order-  7  MSE =  2.275274613328075
Order-  8  MSE =  3.0862273998129326
Order-  9  MSE =  4.299421233059823
Best Order-  4  MSE =  1.4196683506454724
```

The best order which gave minimum error was polynomial with the order 4.

Evaluation - Cross Validation

The evaluation method cross validation is used.

In cross validation the data needs to be split in equal sized subsets, in our case 50. It should be trained on 49 and 1 set should be used for test. All this procedure needs to be run 50 times with altering the testing data. And the result should be averaged at the end.

Was not the best idea to use this approach, in fact for some reason it was giving very high error on one part of the data.

```
MSE K Fold =  14.405251261240592
MSE K Fold =  1.668487071311439
MSE K Fold =  1.197723380959681
MSE K Fold =  0.5912749782208094
MSE K Fold =  0.9722497014734444
MSE K Fold =  1.0093758050699675
MSE K Fold =  0.6533895444770963
MSE K Fold =  1.523388788908503
MSE K Fold =  0.8108751230885154
MSE K Fold =  1.7220359814488886
MSE K Fold =  2.4622381674076608
MSE K Fold =  1.4709488578059262
MSE K Fold =  0.6026362228870216
MSE K Fold =  2.679927972151758
MSE K Fold =  2.642754478016269
MSE K Fold =  0.919287419745275
MSE K Fold =  0.8546123989489655
MSE K Fold =  0.5105164586713844
MSE K Fold =  0.6228400826374143
MSE K Fold =  2.2326293929500545
MSE K Fold =  1.8639348502602215
MSE K Fold =  1.664472487848191
MSE K Fold =  0.5881948585619456
MSE K Fold =  4.513172106247606
MSE K Fold =  0.9317241598386883
MSE K Fold =  2.087459759428547
MSE K Fold =  1.5469525773710087
MSE K Fold =  2.3544161203688825
MSE K Fold =  1.541536316970976
MSE K Fold =  1.9296655645168117
MSE K Fold =  0.8782910258818136
MSE K Fold =  0.7762664386506415
MSE K Fold =  1.033185631318628
MSE K Fold =  0.3260438786489979
MSE K Fold =  0.7090973767888762
MSE K Fold =  1.675332209928296
MSE K Fold =  0.6204866944265277
MSE K Fold =  0.9734838338852891
MSE K Fold =  2.7554491191343584
MSE K Fold =  1.4332870208929809
MSE K Fold =  2.3162184569234547
MSE K Fold =  0.6342709015530402
MSE K Fold =  0.3776863762335792
MSE K Fold =  2.563581852389883
```

Average MSE = 2.515476

It should be noted that the reason of MSE being so low, is that because it uses polynomial regression.

Conclusion

The data was processed, correlation between features was analysed. Some features were eliminated.

Linear regression was applied to predict output, the order of polynomial was increased, the best degree was found. Also, cross validation was applied to train the model.