EE219 - UCLA

WINTER 2019

PROJECT 4: REPORT

Regression Analysis

GROUP MEMBERS

Anchal Goyanka
Nandan Parikh
Pratik Mangalore
Srishti Majumdar

Dataset 1

Working with Network Backup Dataset.

Dataset 1 : Part 1 : Loading Dataset

A. Plotting backup sizes for all workflows for 21 days

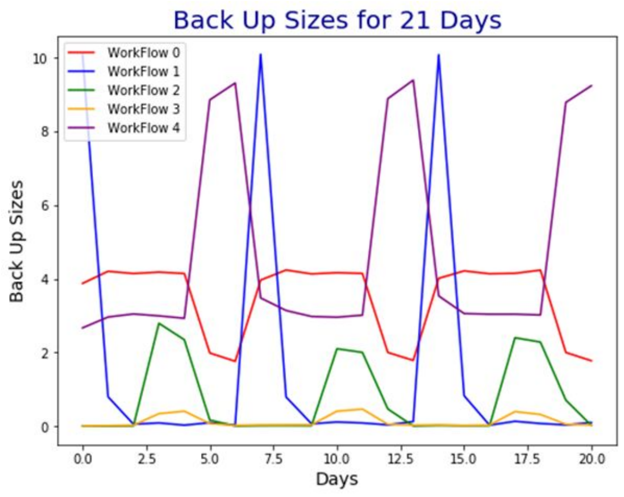


Figure 1.1.1: Backup sizes for 21 days

B. Plotting backup sizes for all workflows for 105 days

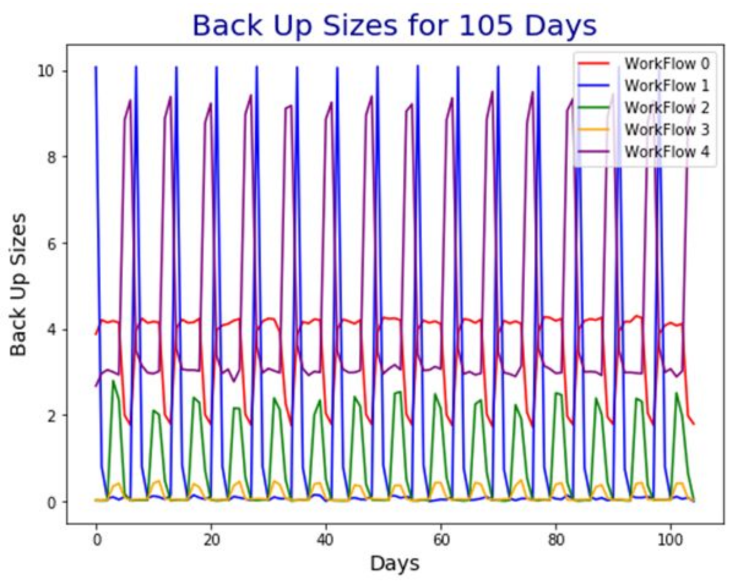


Figure 1.1.2: Backup sizes for 105 days

C. Identifying any repeating patterns

When we observe the plots in Figures 1.1 and 1.2, we notice that every week, for all workflows, the same pattern is repeated. Figure 1.1 can be referred to check the repeating patterns in more detail.

Dataset 1 : Part 2 : Predict Backup Size Given Other Attributes

A. Linear Regression Model

After mapping categorical features to numerical values, we run the linear regression model. Figure 1.2.1.1 and Figure 1.2.1.2 help visualise the performance of our linear regression model.

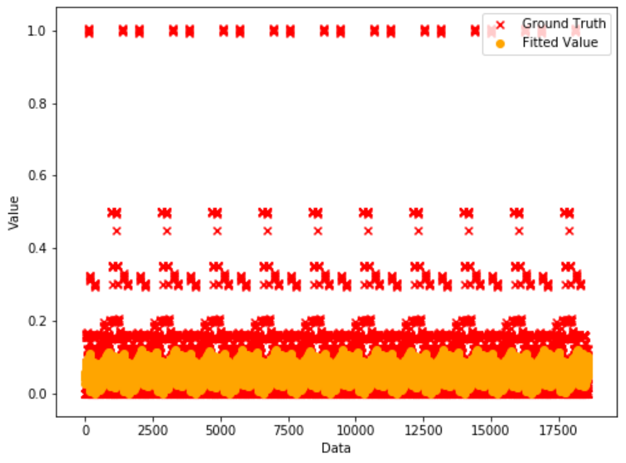


Figure 1.2.1.1: Linear Regression- Fitted Values and True Values

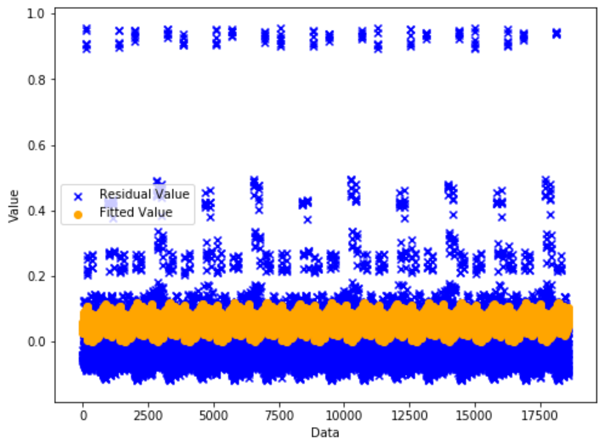


Figure 1.2.1.2: Linear Regression- Fitted Values and Residual Values

Linear Regression- Error Scores	
Training RMSE	Testing RMSE
0.10183435819796753	0.1019394462420986

B. Random Forest Model

1. Random Forest Case 1

We consider the following parameters:

- Number of trees: 20
- Depth of each tree: 4
- Bootstrap: True
- Maximum number of features: 5

Figure 1.2.2.1 and Figure 1.2.2.2 help visualise the performance of our random forest model.

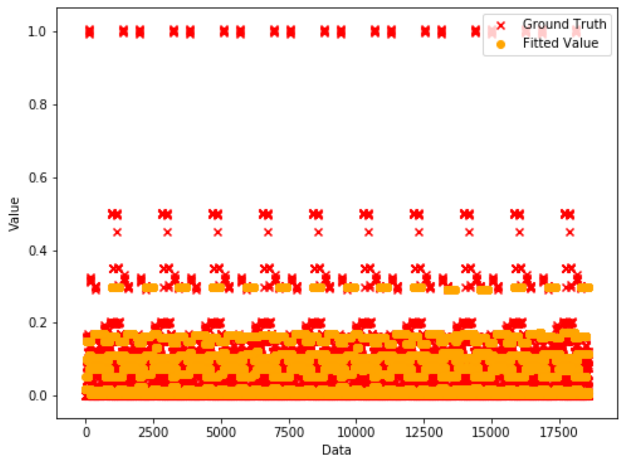


Figure 1.2.2.1: Random Forest Case I- Fitted Values and True Values

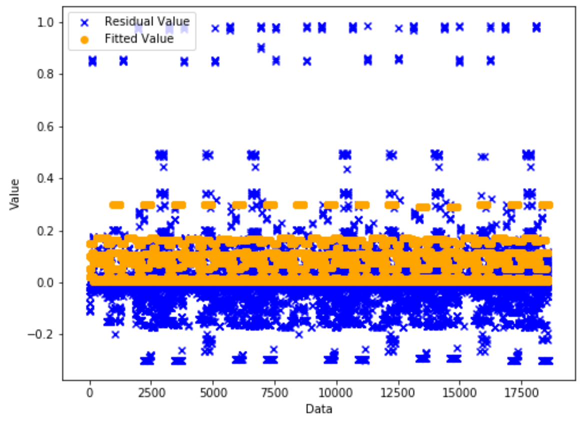


Figure 1.2.2.2: Random Forest Case I- Fitted Values and Residual Values

Random Forest Case I- Error Scores		
Training RMSE	Testing RMSE	OOB Error
0.07581450158836613	0.07609190129765114	0.5407740764974224

2. Varying Parameters: Number of Trees & Maximum Features

We vary the number of trees from 1 to 200 and maximum features from 1 to 5. We note down the OOB score for the trees and average testing RMSE as shown in figures 1.2.2.3 and 1.2.2.4. While sweeping we also check for the configuration of that gives us the least OOB Error.

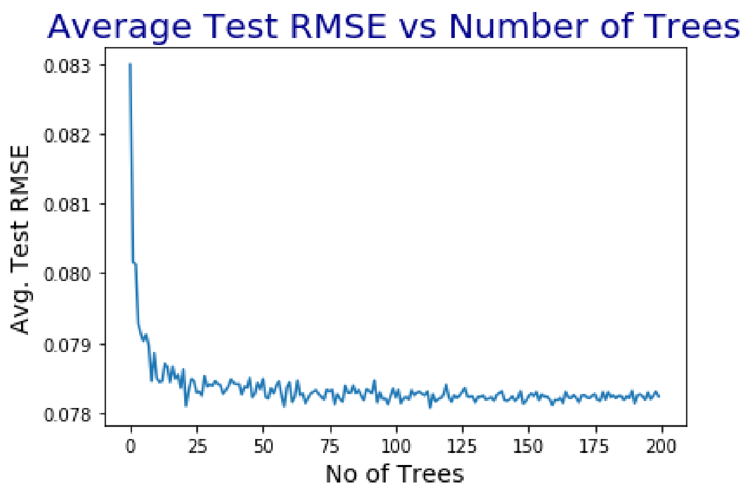


Figure 1.2.2.3: Random Forest- Average Testing RMSE against Number of Trees

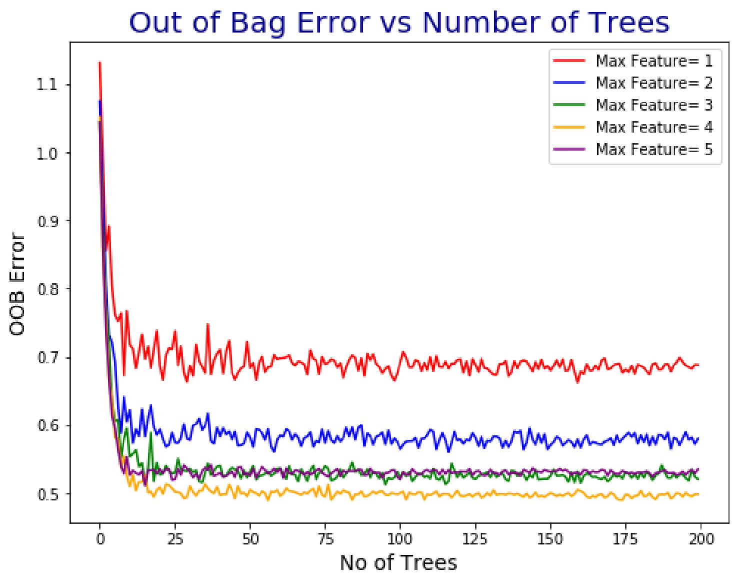


Figure 1.2.2.4: Random Forest- OOB error against Number of Trees

From the above, we get best configuration as:

Random Forest Sweeping Result				
Number of Trees	Maximum Features	Training RMSE	Testing RMSE	Min OOB Error Obtained
57	4	0.07317906198397026	0.07329351585472486	0.4891873119246919

3. Experimenting on another feature

Now we experiment on another feature, Maximum Depth of Tree. We vary the depth from 1- 10 and note down the OOB Error and testing Rmse.

Note: We keep number of trees as 57 and maximum features as 4 throughout.

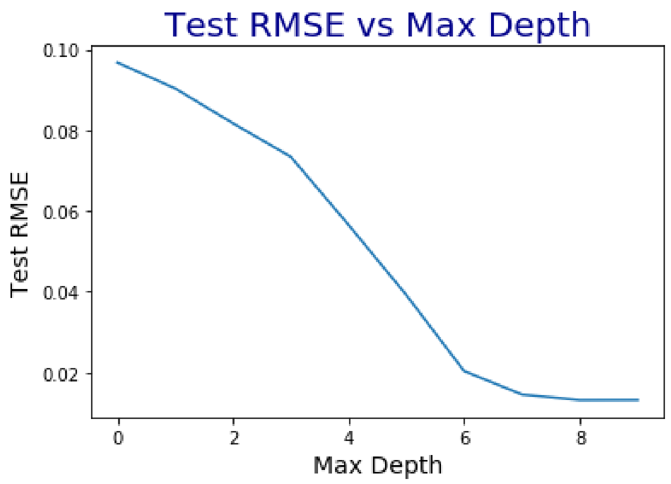


Figure 1.2.2.5: Best Random Forest- Test RMSE against Maximum Depth of Trees

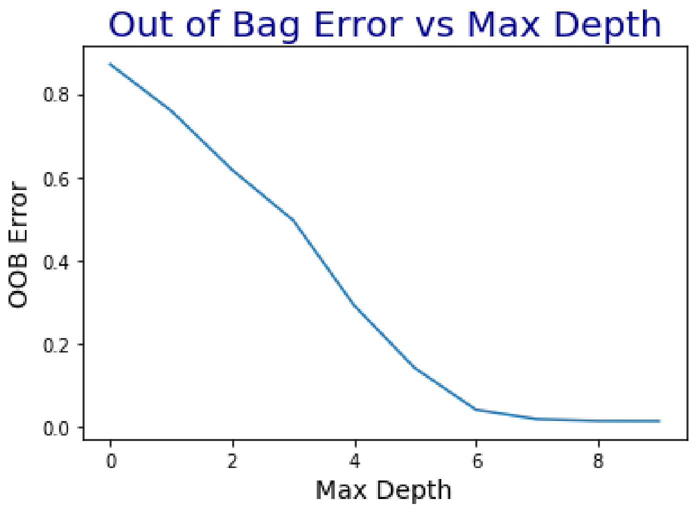


Figure 1.2.2.5: Best Random Forest- OOB Error against Maximum Depth of Trees

We discover the best configuration as:

Best Random Forest					
Number of Trees	Maximum Features	Maximum Depth	Training RMSE	Testing RMSE	Min OOB Error Obtained
57	4	10	0.011158964638163496	0.013110727513667871	0.014925038017956194

To visualise our best Random Forest, we show plots 1.2.2.6 and 1.2.2.7.

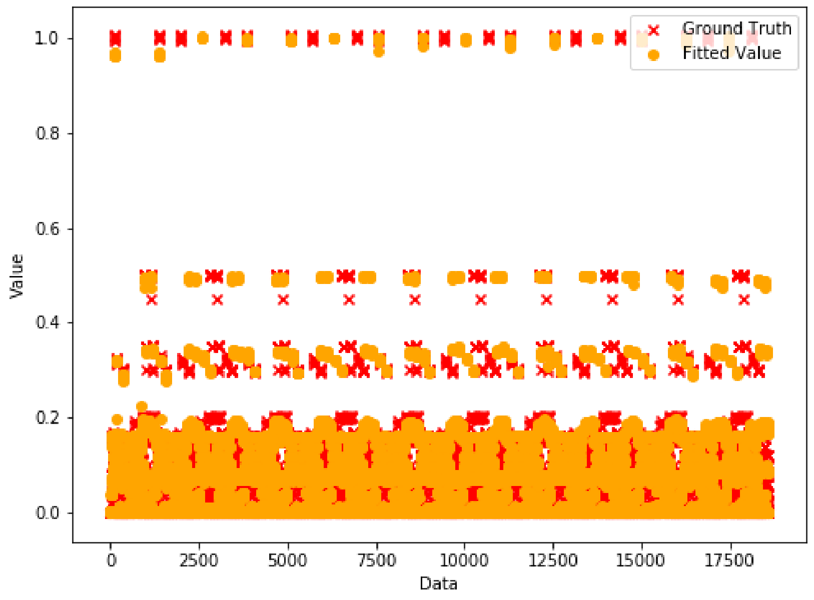


Figure 1.2.2.6: Best Random Forest- Fitted Values and True Values

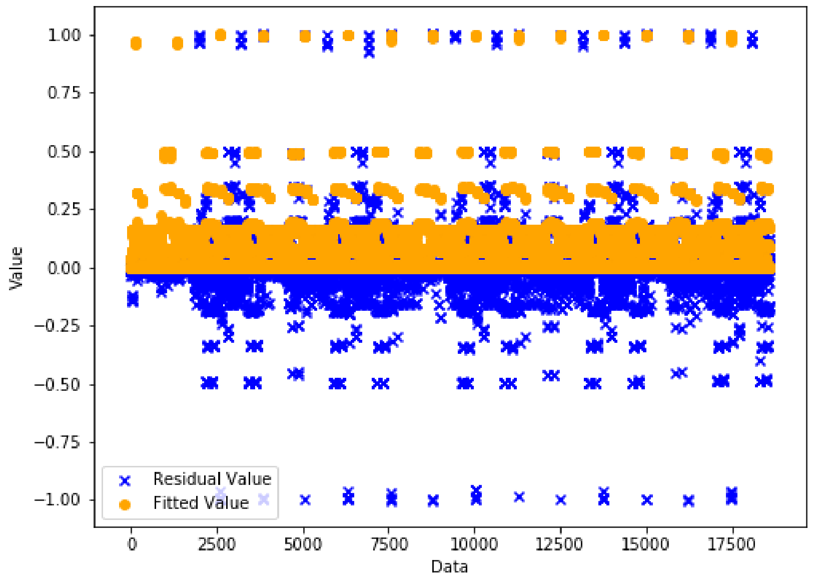


Figure 1.2.2.7: Best Random Forest- Fitted Values and Residual Values

4. Reporting Feature Importance

We check the feature important for the best Random Forest obtained,when max_depth is set 4 (2.b.ii) as well as post experimenting on depth (2.b.iii).

Best Random Forest Feature Importance					
Model as per Question	Week #	Day of Week	Backup Start Time - Hour of Day	Work-Flow-ID	File Name
2.b.ii	1.93110315e-04	3.63638280e-01	8.80103742e-02	4.69758603e-01	7.83996325e-02
2.b.iii	0.00333184	0.33482775	0.34267674	0.23952885	0.07963482

5. Visualising Trees

We visualise a tree from our best RF model (when number of trees is 114, maximum features is 5 and maximum depth is 4) obtained in 2.b.ii using export_graphviz as shown in Figure 1.2.2.8. The root node is Work-Flow-ID which is also the most important feature according to the feature importance reported by the regressor.

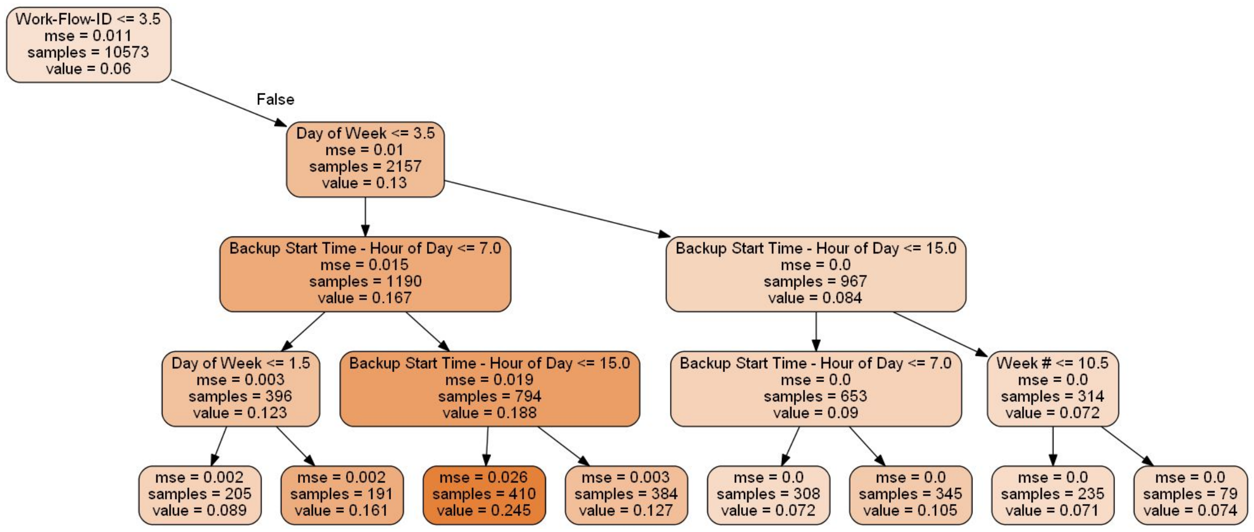


Figure 1.2.2.8 (a): Best Random Forest Visualisation Right Half: Trees- 57, Max Depth- 4, Max Feature-4

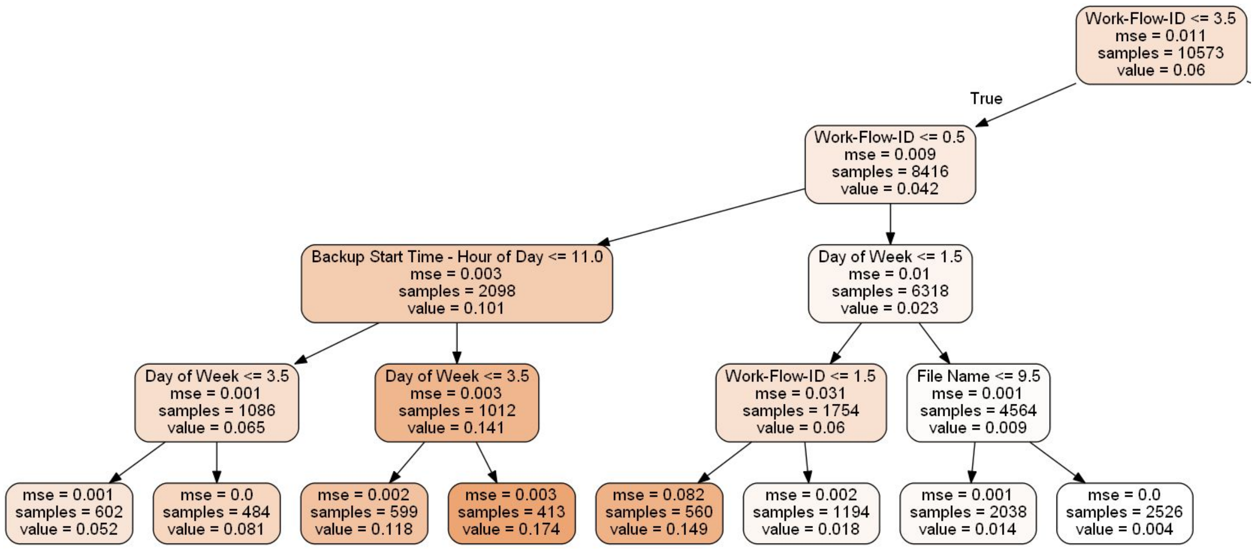


Figure 1.2.2.8 (b): Best Random Forest Visualisation Left Half: Trees- 57, Max Depth- 4, Max Feature-4

We also visualise a tree from our best RF model (when number of trees is 57, maximum features is 4 and maximum depth is 10) after experimenting on Maximum depth as per 2.b.iii. This is shown in Figure 1.2.2.9. As the question asks for maximum depth of 4 to be shown, we set max_depth= 4 as a parameter. The root node in this case also is Work-Flow-ID. However, Backup Start Time - Hour of Day is the most important feature according to the feature importance reported by the regressor. This is because Random Forest uses gini impurity, and as mentioned in the project description, “the importance of each feature is the averaged decreased variance for each node split with this feature in the forest and weighted by the number of samples it splits”. Hence, the root node may not always be the most important feature.

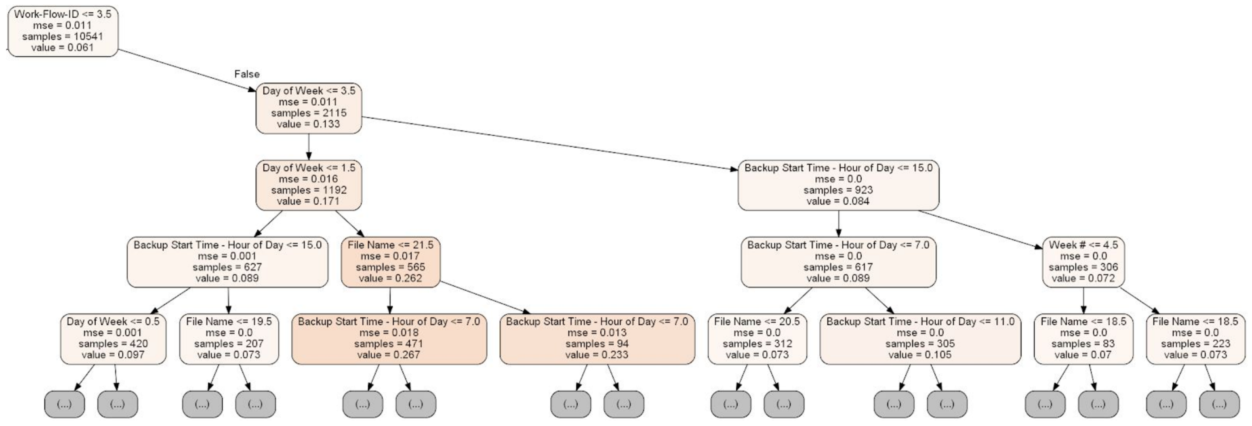


Figure 1.2.2.9 (a): Best Random Forest Visualisation Right Half: Trees- 57, Max Depth- 10, Max Feature-4

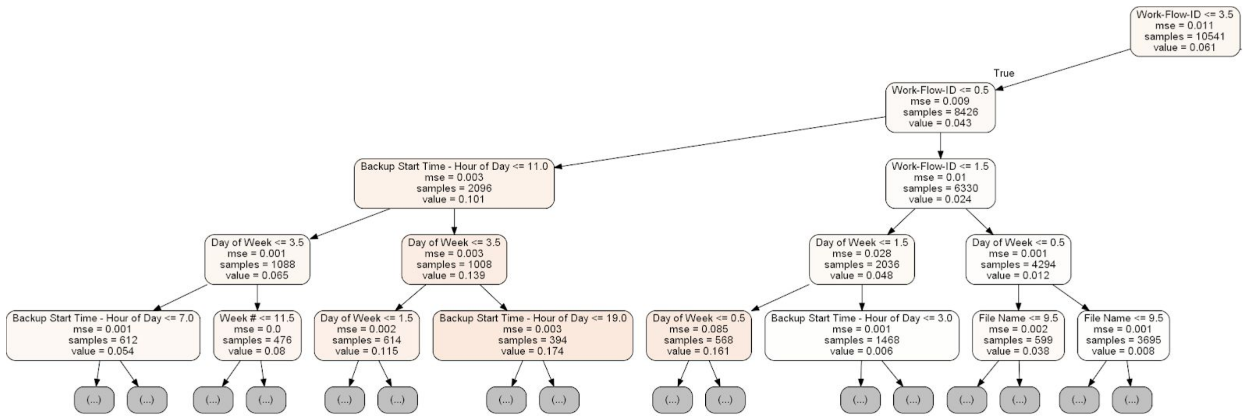


Figure 1.2.2.9 (b): Best Random Forest Visualisation Left Half: Trees- 57, Max Depth- 10, Max Feature-4

Q 2C. Neural Network Model

In this question, we train a neural network with one layer and different activation functions and different hidden units. We get the following results and graphs:

The test rmse for Relu is the least among all activation function. Tanh performs poorer than Relu but better than logistic. Logistic function gives comparatively high test rmses than other activation function, no matter what the number of hidden units is.
We get the best combination with Relu as hidden units reach 350 with rmse of 0.026020130193598844.

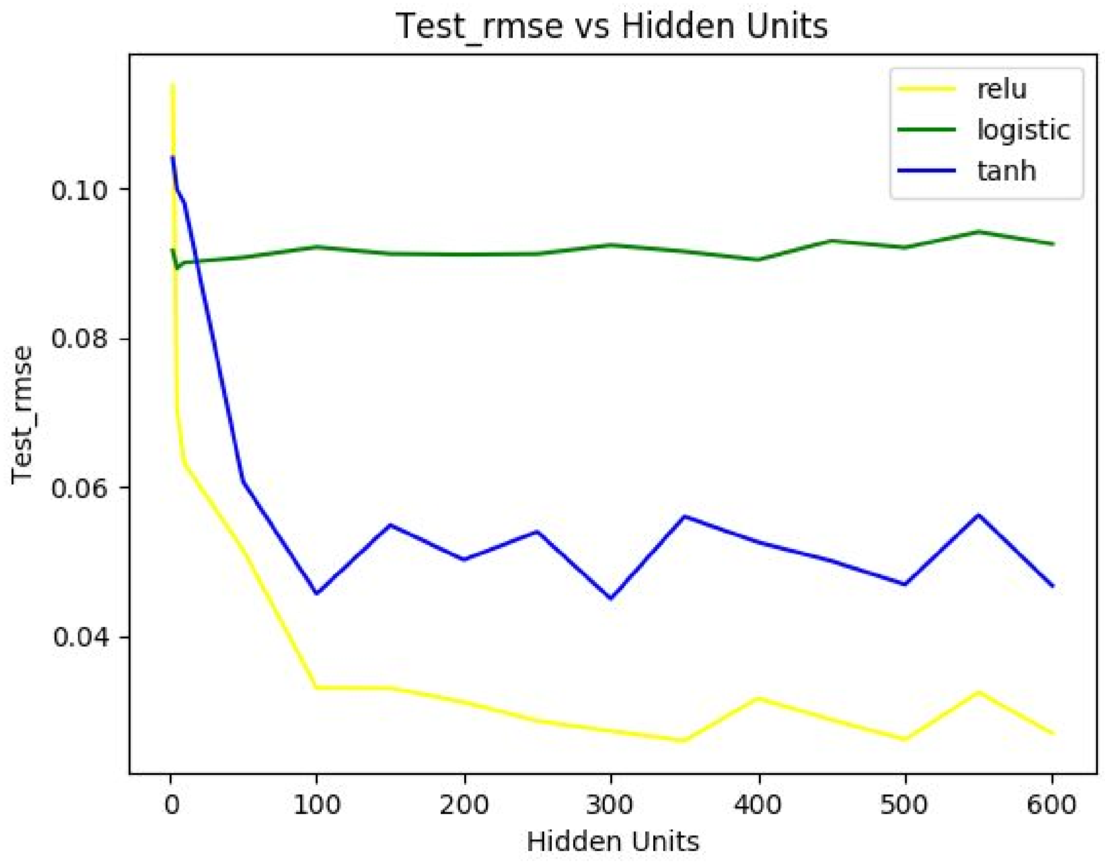


Figure 1.2.3.1: test-RMSE vs the number of hidden units for each activation function

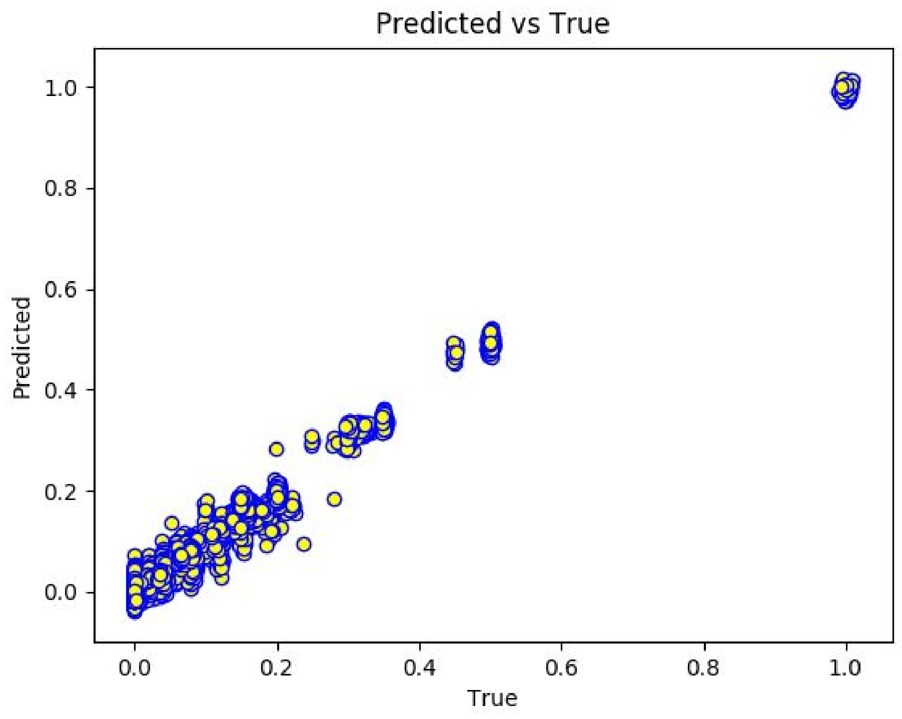


Figure1. 2.3.2: Predicted vs True values for Best parameters of NN

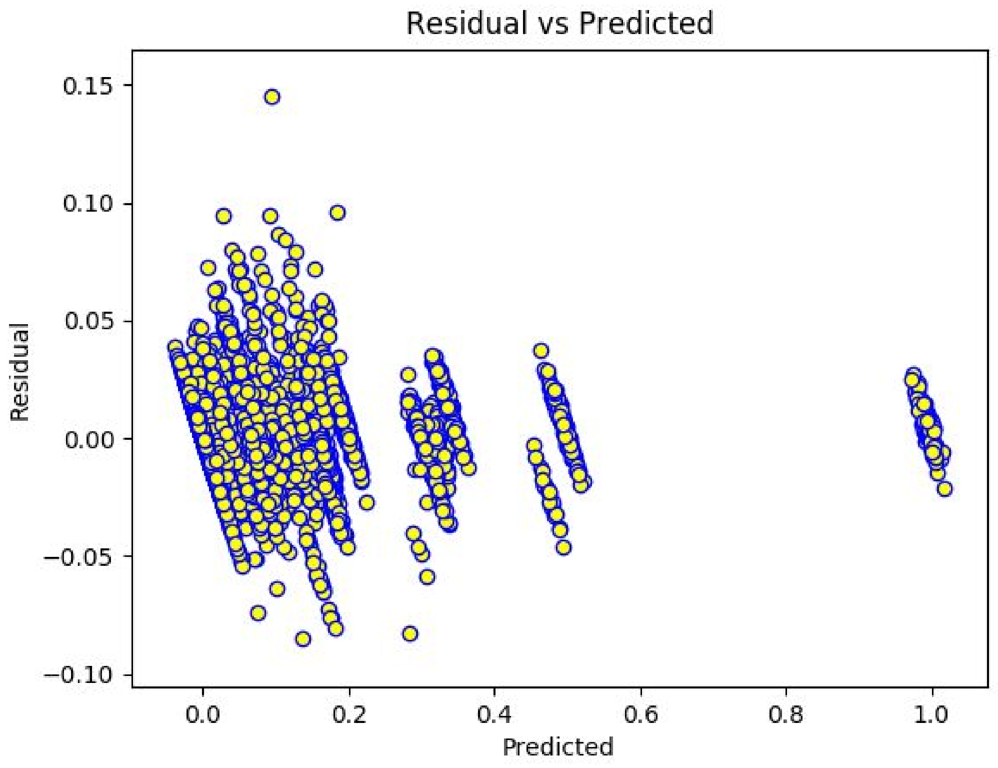


Figure 1. 2.3.3: Predicted vs Residual for Best parameters of NN

Train and test rmses for each model that we trained are as follows:

```
test_rmse, relu, hidden_units= 2 ,  0.11388287730674199
test_rmse, relu, hidden_units= 5 ,  0.07060006250381666
test_rmse, relu, hidden_units= 10 ,  0.06321299101198263
test_rmse, relu, hidden_units= 50 ,  0.05161640196621994
test_rmse, relu, hidden_units= 100 ,  0.03311751283646015
test_rmse, relu, hidden_units= 150 ,  0.03307760923116892
test_rmse, relu, hidden_units= 200 ,  0.031206045879051604
test_rmse, relu, hidden_units= 250 ,  0.02867855367710533
test_rmse, relu, hidden_units= 300 ,  0.027328787260479437
test_rmse, relu, hidden_units= 350 ,  0.026020130193598844
test_rmse, relu, hidden_units= 400 ,  0.03170920806237606
test_rmse, relu, hidden_units= 450 ,  0.028837472071673316
test_rmse, relu, hidden_units= 500 ,  0.026180827266551427
test_rmse, relu, hidden_units= 550 ,  0.03251435016962443
test_rmse, relu, hidden_units= 600 ,  0.027016231849672286
test_rmse, logistic, hidden_units= 2 ,  0.09174484041188642
test_rmse, logistic, hidden_units= 5 ,  0.0893573078969053
test_rmse, logistic, hidden_units= 10 ,  0.09014725400260395
test_rmse, logistic, hidden_units= 50 ,  0.09084257603700854
test_rmse, logistic, hidden_units= 100 ,  0.09220472528790356
test_rmse, logistic, hidden_units= 150 ,  0.09133004055744293
test_rmse, logistic, hidden_units= 200 ,  0.09122068862312045
test_rmse, logistic, hidden_units= 250 ,  0.09131153949497132
test_rmse, logistic, hidden_units= 300 ,  0.0924635275252481
test_rmse, logistic, hidden_units= 350 ,  0.09163090792729761
test_rmse, logistic, hidden_units= 400 ,  0.09053518192385371
test_rmse, logistic, hidden_units= 450 ,  0.09305694502938729
test_rmse, logistic, hidden_units= 500 ,  0.0921466169178649
test_rmse, logistic, hidden_units= 550 ,  0.0942634915820797
test_rmse, logistic, hidden_units= 600 ,  0.0926435737494942
test_rmse, tanh, hidden_units= 2 ,  0.10418805964665834
test_rmse, tanh, hidden_units= 5 ,  0.09996156895344993
test_rmse, tanh, hidden_units= 10 ,  0.09818064347474179
test_rmse, tanh, hidden_units= 50 ,  0.06077253827966476
test_rmse, tanh, hidden_units= 100 ,  0.04572096593589347
test_rmse, tanh, hidden_units= 150 ,  0.054953059159604814
test_rmse, tanh, hidden_units= 200 ,  0.05028383984134684
test_rmse, tanh, hidden_units= 250 ,  0.0540423056433197
test_rmse, tanh, hidden_units= 300 ,  0.04505084397048233
test_rmse, tanh, hidden_units= 350 ,  0.05608590336423856
```

```
test_rmse, tanh, hidden_units= 400 ,  0.05264574590001223
test_rmse, tanh, hidden_units= 450 ,  0.05012052601777807
test_rmse, tanh, hidden_units= 500 ,  0.04697060434408466
test_rmse, tanh, hidden_units= 550 ,  0.05631368706139469
test_rmse, tanh, hidden_units= 600 ,  0.04681432514572661
train_rmse, relu, hidden_units= 2 ,  0.08542381159518289
train_rmse, relu, hidden_units= 5 ,  0.052072203224088474
train_rmse, relu, hidden_units= 10 ,  0.032620675740130355
train_rmse, relu, hidden_units= 50 ,  0.020082532654340873
train_rmse, relu, hidden_units= 100 ,  0.01601231848613785
train_rmse, relu, hidden_units= 150 ,  0.014952577685953811
train_rmse, relu, hidden_units= 200 ,  0.01465384321649075
train_rmse, relu, hidden_units= 250 ,  0.013942090468593373
train_rmse, relu, hidden_units= 300 ,  0.01333553053155597
train_rmse, relu, hidden_units= 350 ,  0.01259625167565331
train_rmse, relu, hidden_units= 400 ,  0.012629601219328273
train_rmse, relu, hidden_units= 450 ,  0.01219382240862641
train_rmse, relu, hidden_units= 500 ,  0.011763144718540643
train_rmse, relu, hidden_units= 550 ,  0.011510807767816622
train_rmse, relu, hidden_units= 600 ,  0.012082412548808815
train_rmse, logistic, hidden_units= 2 ,  0.0897216998701248
train_rmse, logistic, hidden_units= 5 ,  0.08828160705397597
train_rmse, logistic, hidden_units= 10 ,  0.08834756393618026
train_rmse, logistic, hidden_units= 50 ,  0.08845229320316521
train_rmse, logistic, hidden_units= 100 ,  0.08907783590810667
train_rmse, logistic, hidden_units= 150 ,  0.0887199648831235
train_rmse, logistic, hidden_units= 200 ,  0.08937213183632091
train_rmse, logistic, hidden_units= 250 ,  0.08965544950840061
train_rmse, logistic, hidden_units= 300 ,  0.09082852347183243
train_rmse, logistic, hidden_units= 350 ,  0.08952540902830414
train_rmse, logistic, hidden_units= 400 ,  0.09019521866428677
train_rmse, logistic, hidden_units= 450 ,  0.09171457415397492
train_rmse, logistic, hidden_units= 500 ,  0.09022998075910325
train_rmse, logistic, hidden_units= 550 ,  0.09032506275185892
train_rmse, logistic, hidden_units= 600 ,  0.09261211283086107
train_rmse, tanh, hidden_units= 2 ,  0.0883768475386614
train_rmse, tanh, hidden_units= 5 ,  0.08651532085615904
train_rmse, tanh, hidden_units= 10 ,  0.07550705424815493
train_rmse, tanh, hidden_units= 50 ,  0.04152107656318378
train_rmse, tanh, hidden_units= 100 ,  0.0344543234066136
train_rmse, tanh, hidden_units= 150 ,  0.03392564131814946
train_rmse, tanh, hidden_units= 200 ,  0.040353953601632464
train_rmse, tanh, hidden_units= 250 ,  0.030216463883239474
train_rmse, tanh, hidden_units= 300 ,  0.03202522420990067
```

```
train_rmse, tanh, hidden_units= 350 ,  0.03480745487635288
train_rmse, tanh, hidden_units= 400 ,  0.03506895576947376
train_rmse, tanh, hidden_units= 450 ,  0.03713354279631236
train_rmse, tanh, hidden_units= 500 ,  0.02758135128560451
train_rmse, tanh, hidden_units= 550 ,  0.042295778113377834
train_rmse, tanh, hidden_units= 600 ,  0.0348547515673073
```

Q2 d. Predicting backup size for individual workflows

Q2 d 1.Linear regression Model for individual workflows

Here we use a simple linear regression model to do backup size prediction for each workflows separately.
Following are the graphs for each model trained:

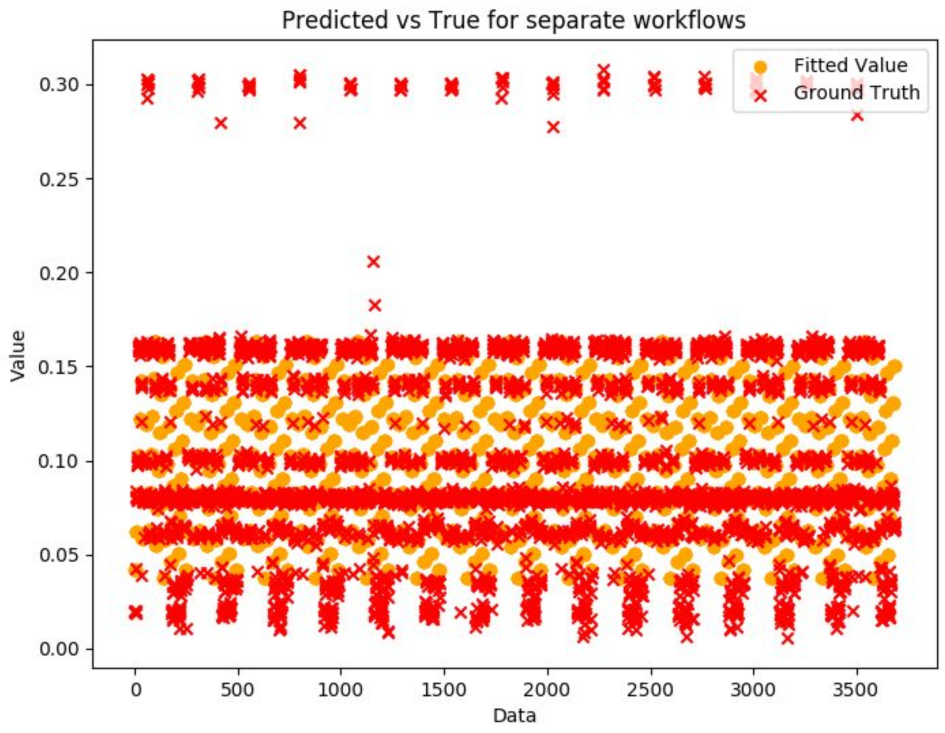


Figure 1.2.4.1.1: Predicted vs True for Linear Regression Model for workflow1

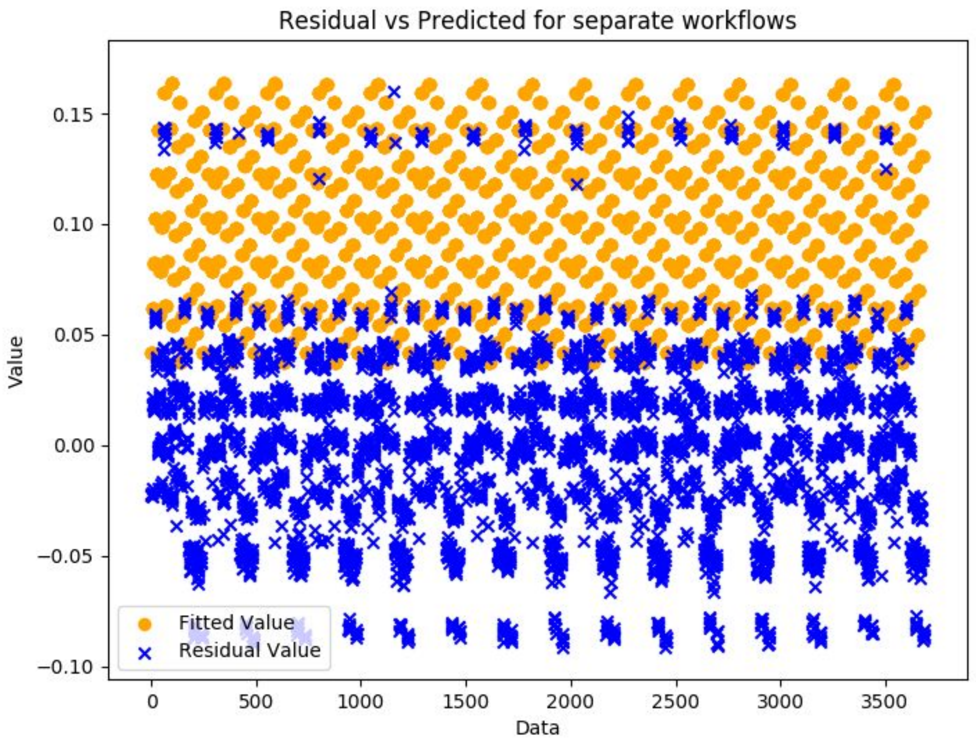


Figure 1.2.4.1.2: Predicted vs Residual for Linear Regression Model for workflow1

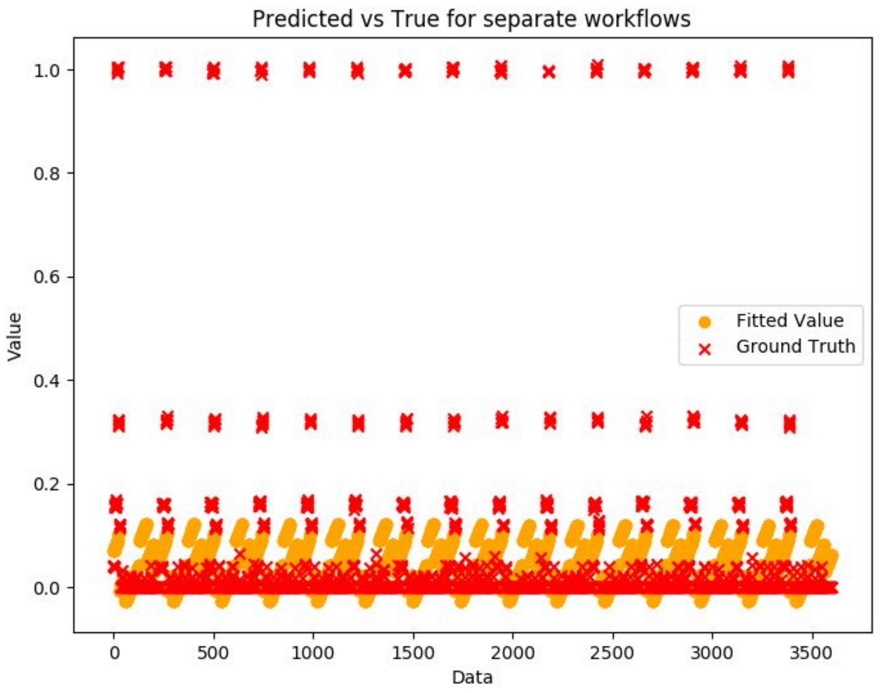


Figure 1.2.4.1.3: Predicted vs True for Linear Regression Model for workflow2

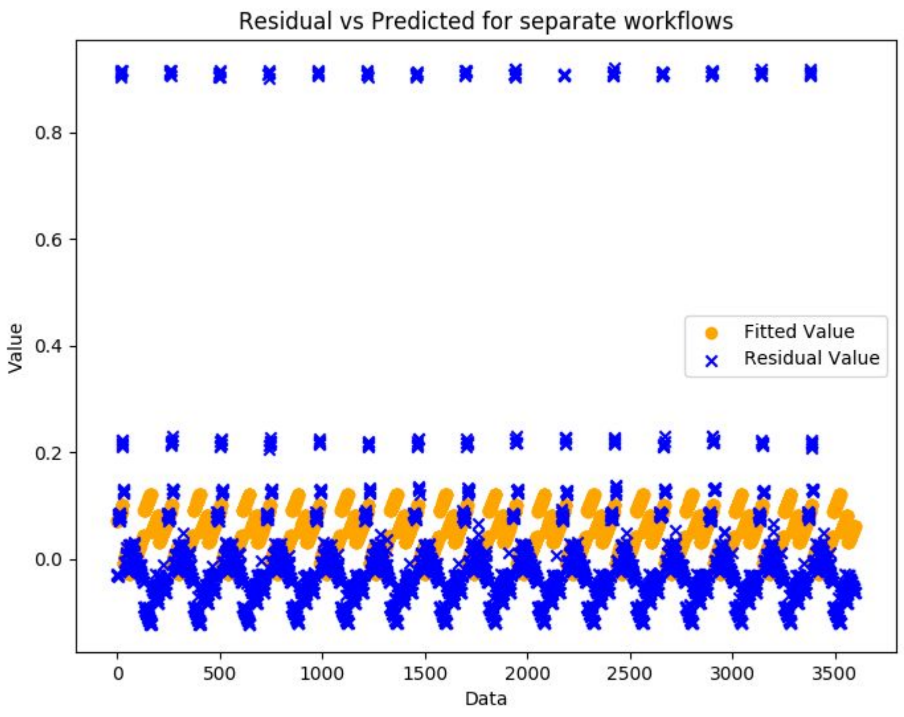


Figure 1.2.4.1.4: Predicted vs Residual for Linear Regression Model for workflow2

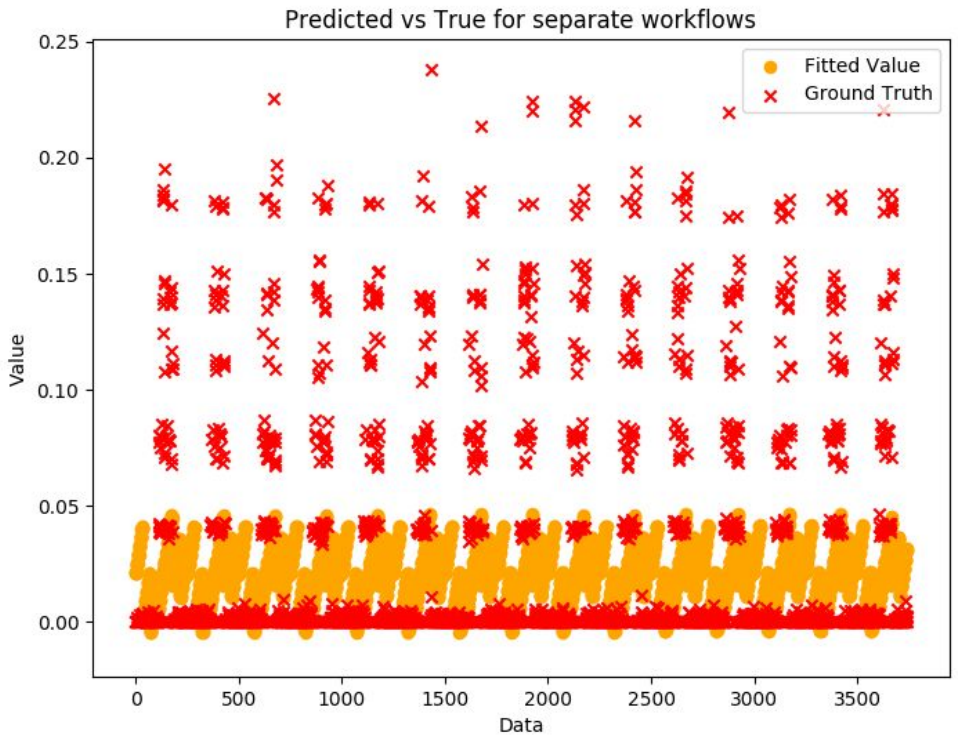


Figure1.2.4.1.5: Predicted vs True for Linear Regression Model for workflow3

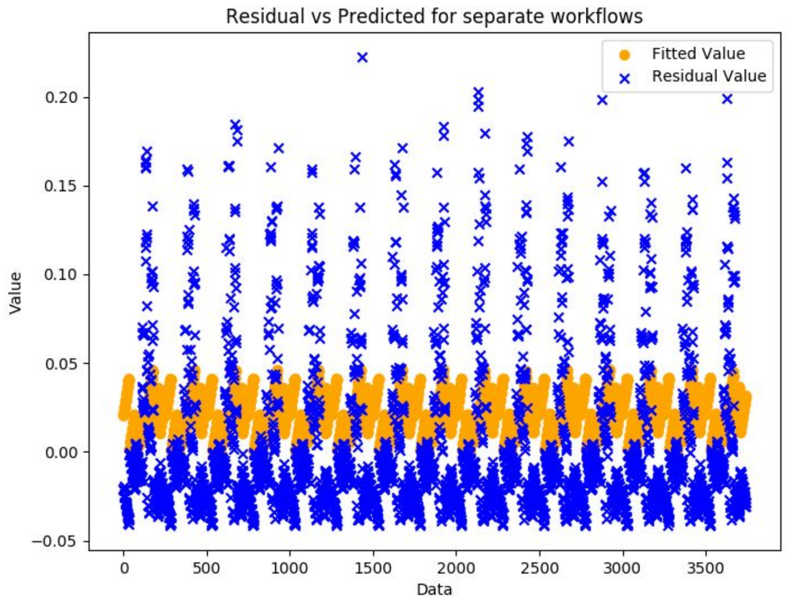


Figure 1.2.4.1.6: Predicted vs Residual for Linear Regression Model for workflow3

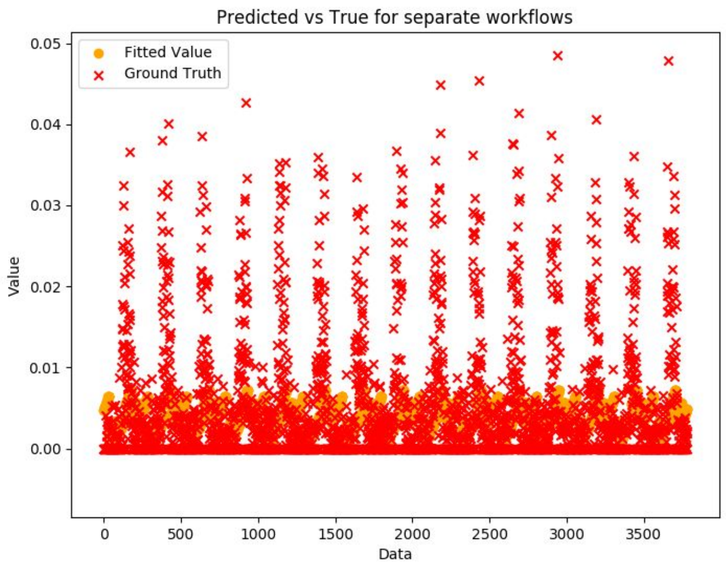


Figure 1.2.4.1..7: Predicted vs True for Linear Regression Model for workflow4

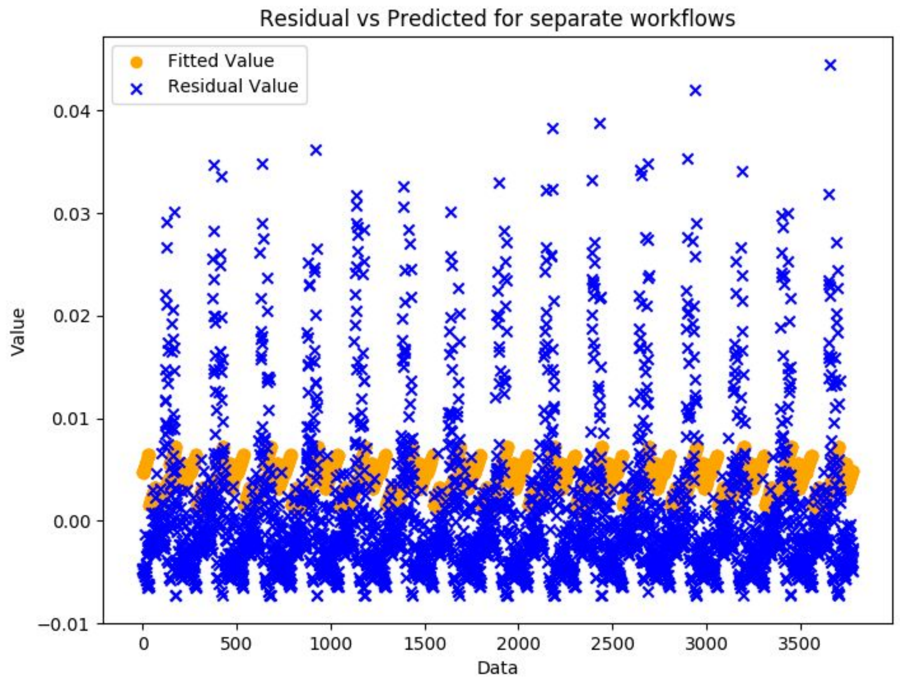


Figure 1.2.4.1.8: Predicted vs Residual for Linear Regression Model for workflow4

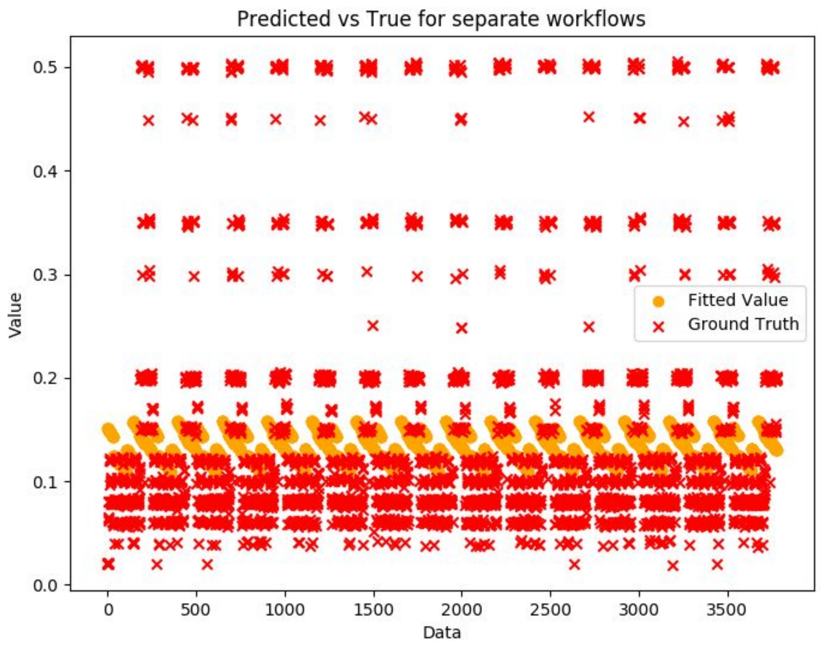


Figure 1.2.4.1.9: Predicted vs True for Linear Regression Model for workflow5

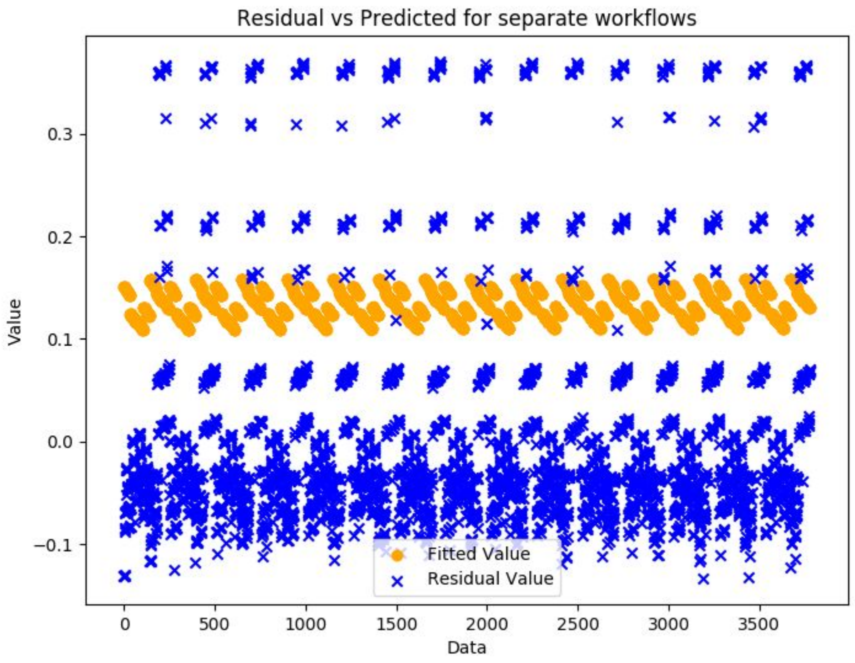


Figure 1.2.4.1.10: Predicted vs Residual for Linear Regression Model for workflow5

Following are the test and train rmses for each workflows:

```
test_rmse, workflow  0 ,  0.04327587388903004
train_rmse, workflow  0 ,  0.04297525111970858
test_rmse, workflow  1 ,  0.15949645935571188
train_rmse, workflow  1 ,  0.15959276434478978
test_rmse, workflow  2 ,  0.04199382761688264
train_rmse, workflow  2 ,  0.04223615898138002
test_rmse, workflow  3 ,  0.007072371214759496
train_rmse, workflow  3 ,  0.007117121987969881
test_rmse, workflow  4 ,  0.10277989283004366
train_rmse, workflow  4 ,  0.10297632091171938
```

Comparing these values for separate workflows with the linear regression done in previous questions, we see that the mean rmses for test and train decreased for workflows 1, 3 and 4. It means that the performance becomes better if we do workflows separately generally. But the value is worse for workflow 2 by a factor of 0.5, which is still okay. And for workflow 5, the value is same as the overall rmse value. So we can say that in general, doing regression on separate workflows is better and that the fit has improved.

Q2d 2.Polynomial Features

In this question we fit a regression model on individual workflows but with polynomial features. We fit and predict models for degrees from 1 to 10 and plot the average train test rmse against degree as follows:

We see that the train rmse for each workflow decreases as the degree increases from 1 to 10. This is expected since, at higher degrees we expect the model to become more complex and fit the train data completely. But it is not the case with the test rmses. For example as you could see form the graphs below, the test rmses decreases with degree till certain point for all workflows except 2nd, and starts increasing after that. This shows that after this point , if we use higher degrees we overfit the data.
Also it can be seen that cross validation helps in this setting, because it helps in generalizing the dataset more. It helps in some manner to avoid over overfitting since we are considering different sets of training and test data. It also helps in controlling the complexity of the model because if we did not use cross validation, it could be the case that the degree at which rmses start increasing is higher than we get right now with cross validation. And this in turn helps avoid over-fitting.
Following are the graphs for each workflows that shows the test and train rmses for different degrees from 1 to 10.

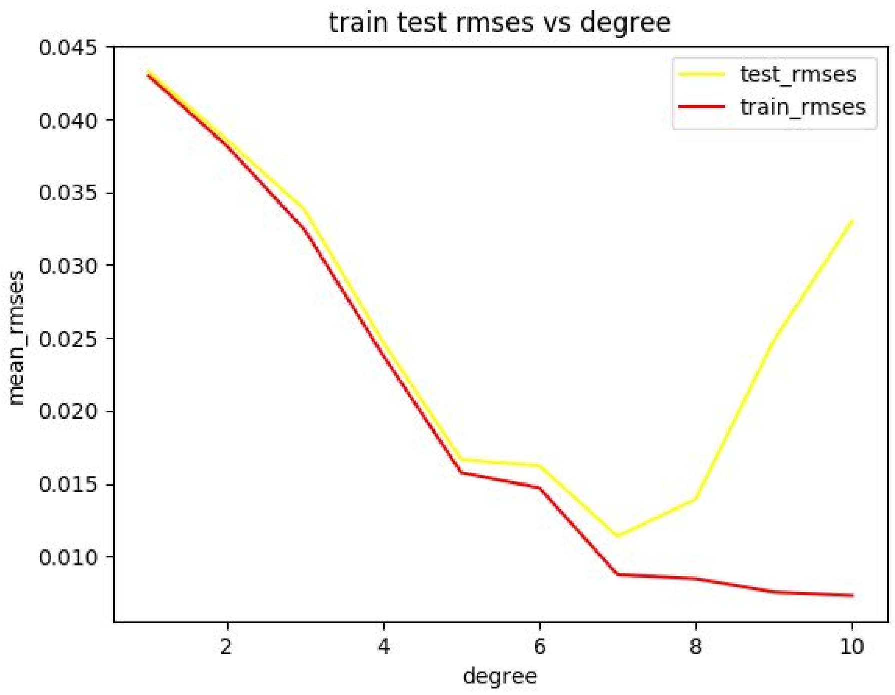


Figure 1.2.4.2.1: Average test and train rmses against degree for workflow1

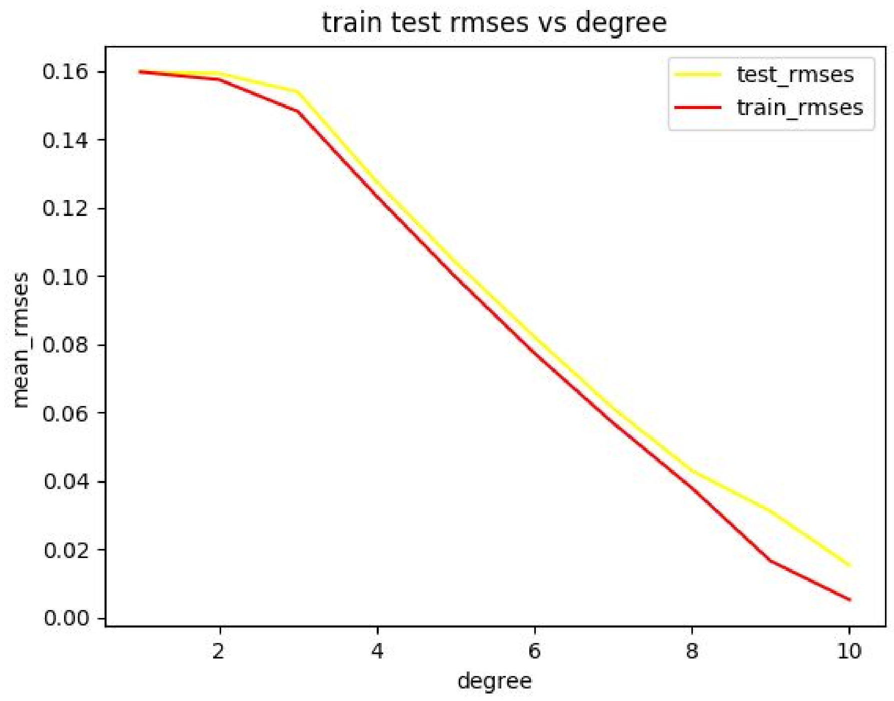


Figure 1.2.4.2.2: Average test and train rmses against degree for workflow2

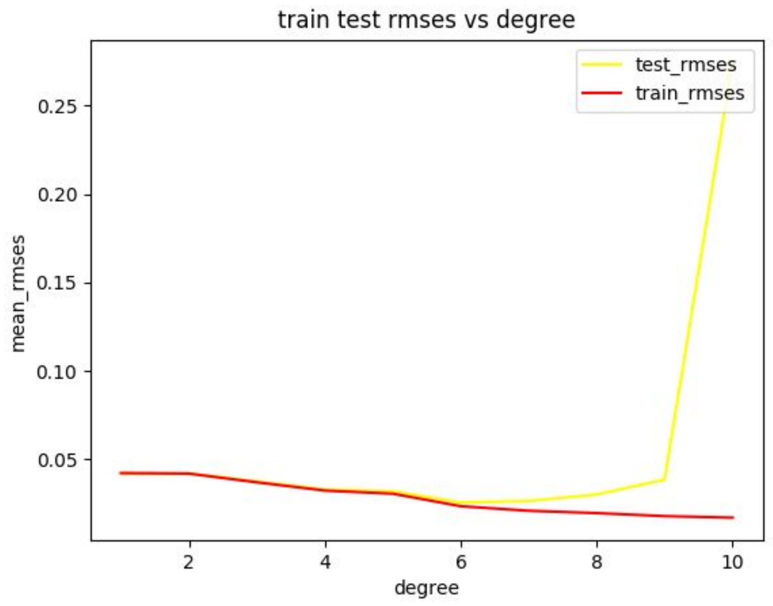


Figure 1.2.4.2.3: Average test and train rmses against degree for workflow3

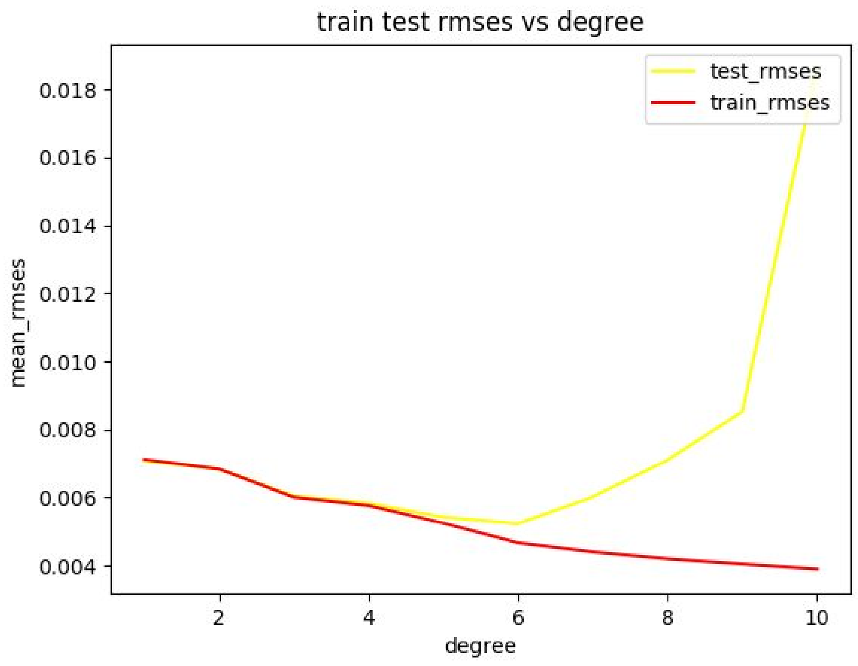


Figure 1.2.4.2.4: Average test and train rmses against degree for workflow4

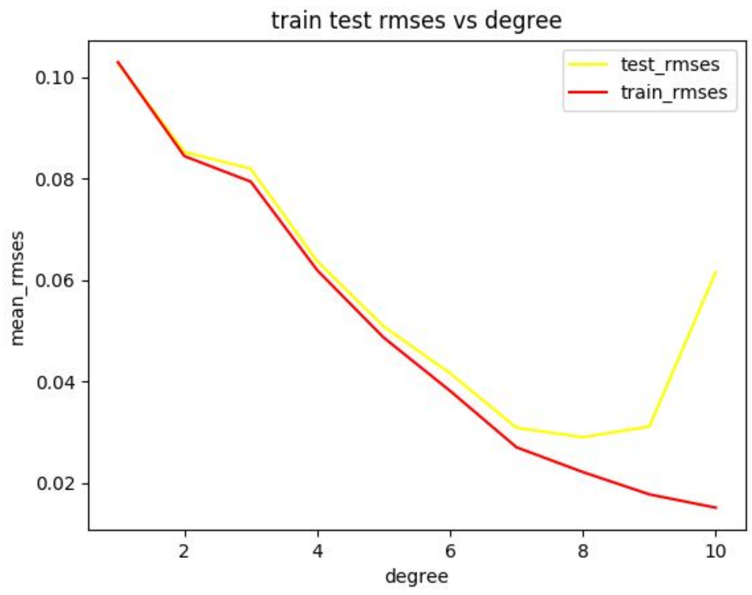


Figure 1.2.4.2.5: Average test and train rmses against degree for workflow5

From these graphs we say that the optimum degree or the thresholds at which the generalizations get worse for different workflows is as follows:

Workflow	Optimum degree of polynomial
1	7
2	10
3	6
4	6
5	8

Next we plot the Predicted vs True graphs and Predicted vs Residual graphs for each workflows and for the best degree found:

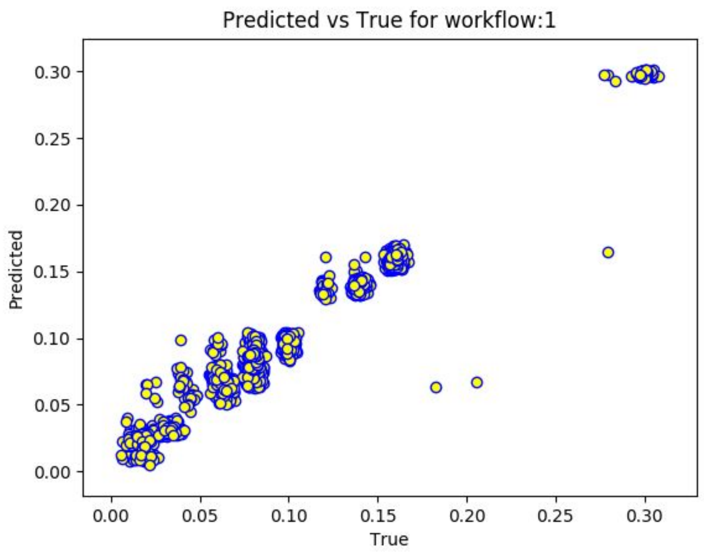


Figure 1.2.4.2.6: Predicted vs True for Regression with Polynomial Features on workflow1 with degree 7

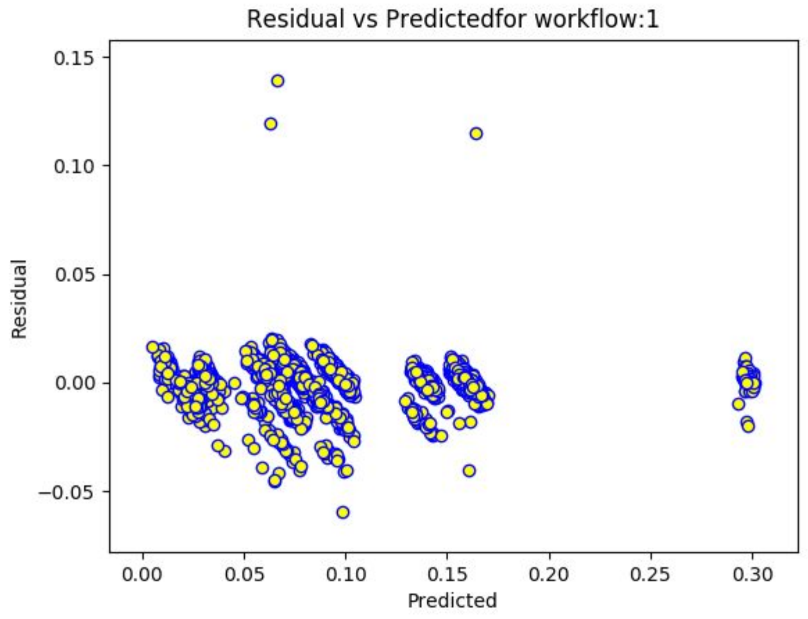


Figure 1.2.4.2.7: Predicted vs Residual for Regression with Polynomial Features on workflow1 with degree 7

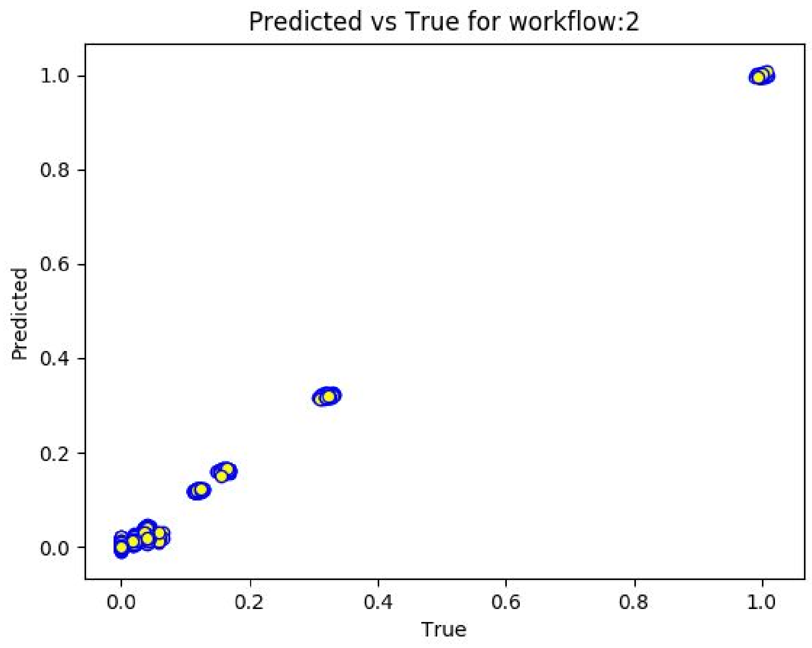


Figure 1.2.4.2.8: Predicted vs True for Regression with Polynomial Features on workflow2 with degree 10

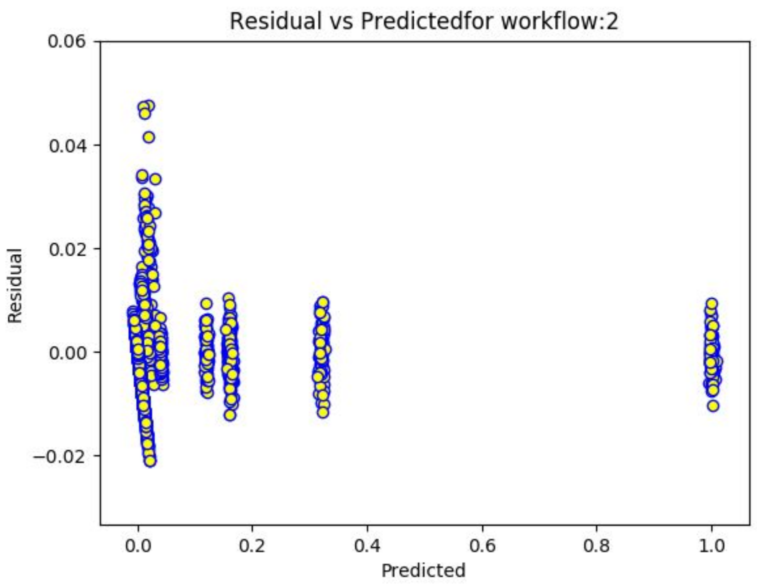


Figure 1.2.4.2.9: Predicted vs Residual for Regression with Polynomial Features on workflow2 with degree 10

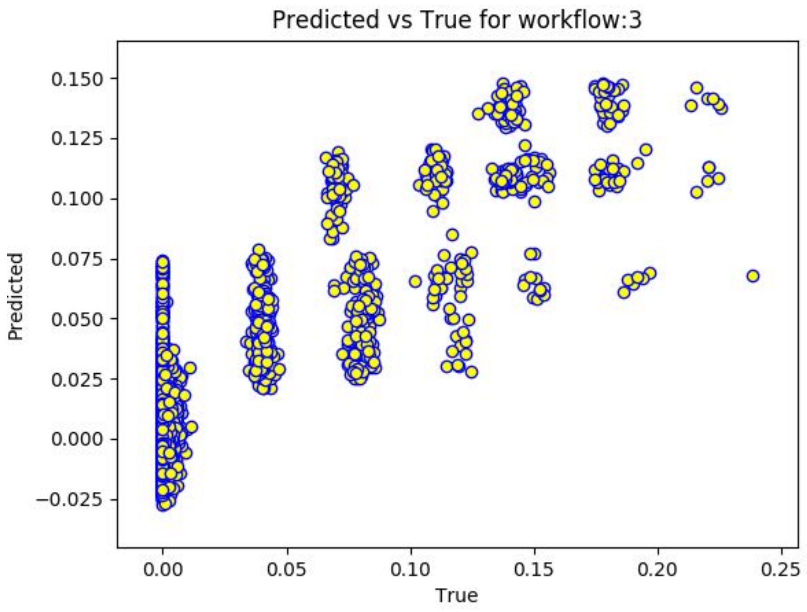


Figure 1.2.4.2.10: Predicted vs True for Regression with Polynomial Features on workflow3 with degree 6

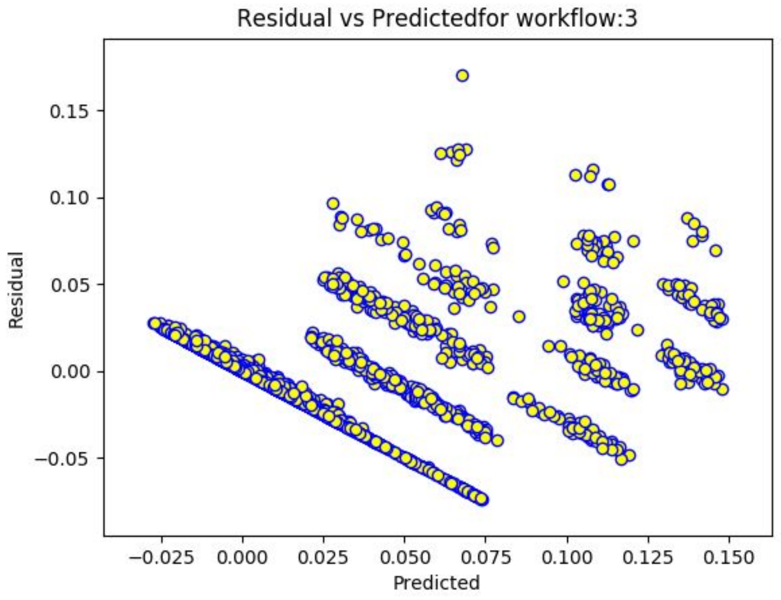


Figure 1.2.4.2.11: Predicted vs Residual for Regression with Polynomial Features on workflow3 with degree 6

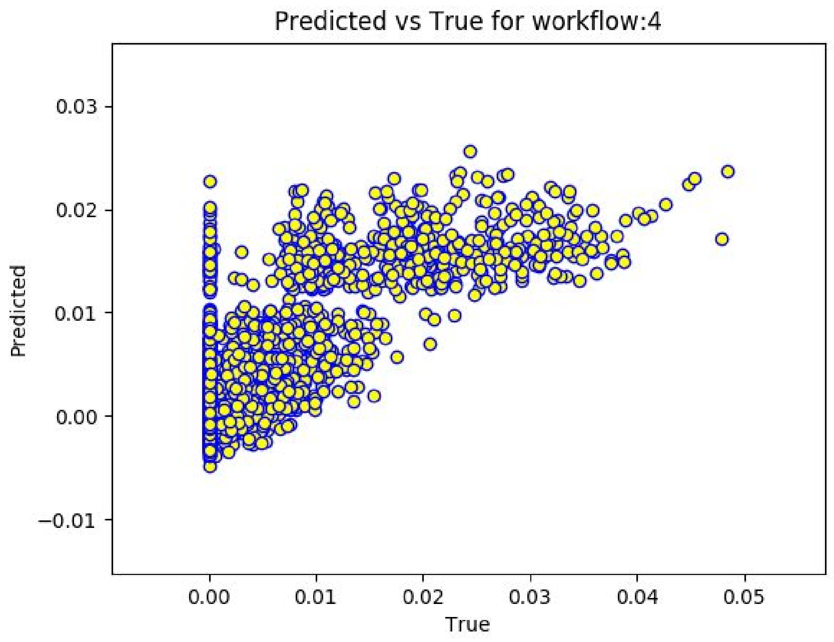


Figure 1.2.4.2.12: Predicted vs True for Regression with Polynomial Features on workflow4 with degree 6

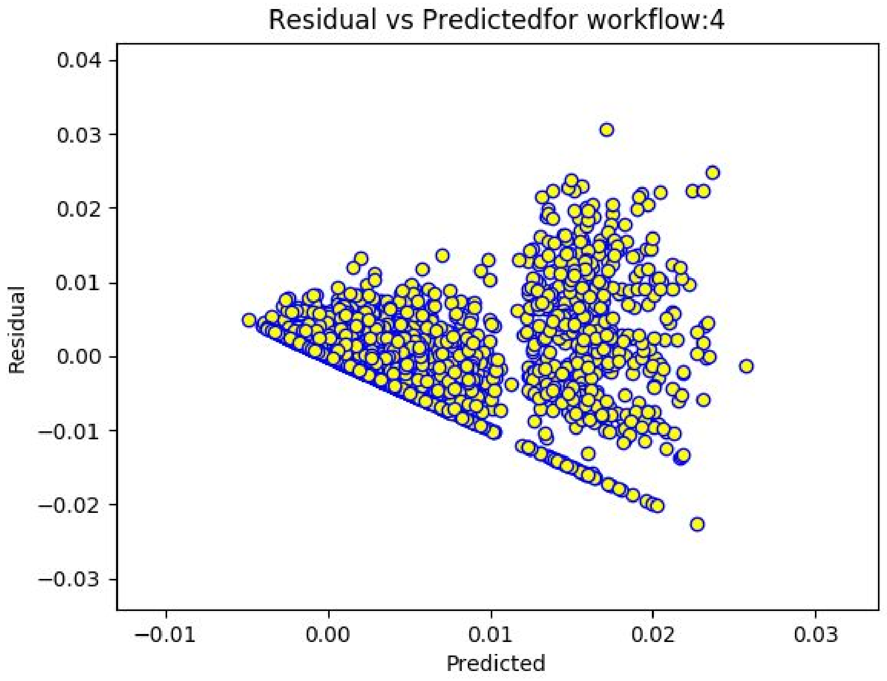


Figure 1.2.4.2.13: Predicted vs Residual for Regression with Polynomial Features on workflow4 with degree 6

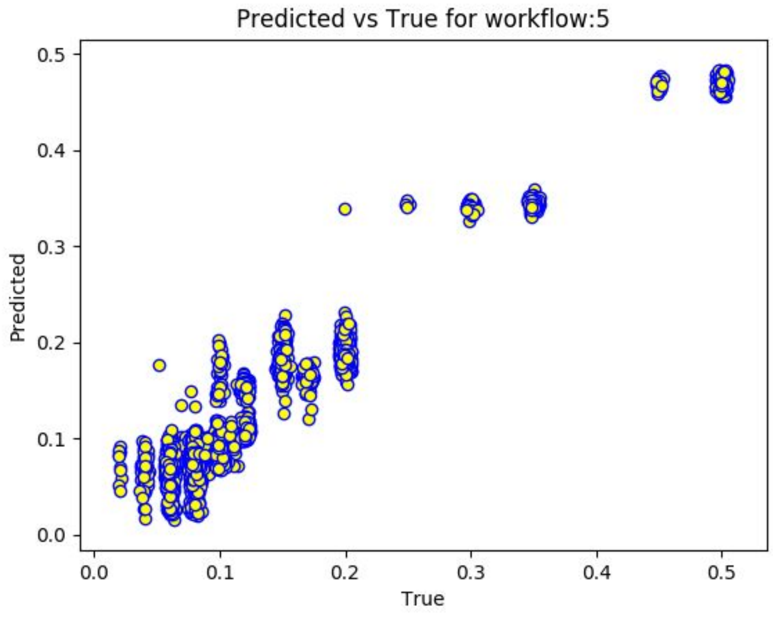


Figure 1.2.4.2.14: Predicted vs True for Regression with Polynomial Features on workflow5 with degree 8

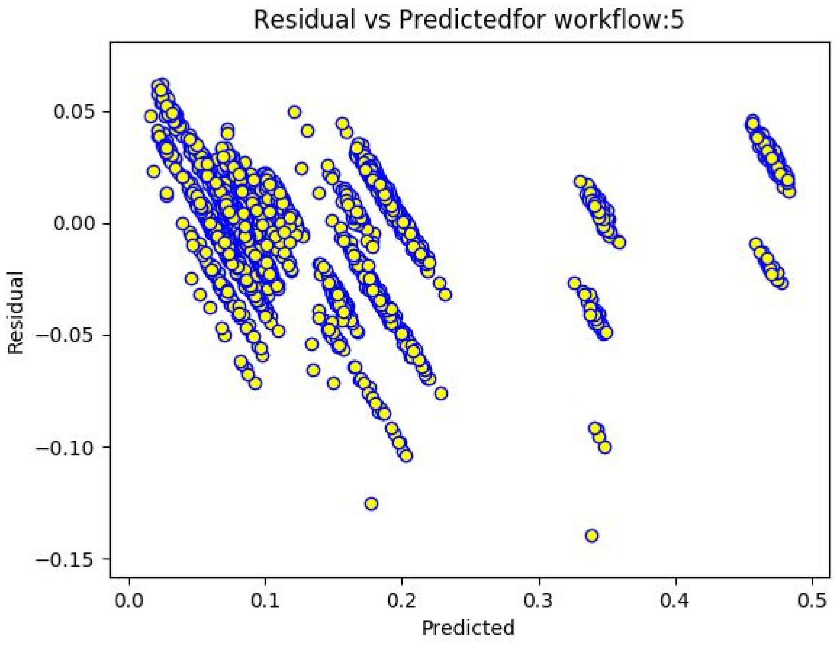


Figure 1.2.4.2.15: Predicted vs Residual for Regression with Polynomial Features on workflow5 with degree 8

Test and train rmses for each workflow and each degree are as follows:

```
test_rmse, workflow  0 , degree:  1 ,  0.04327587388903004
train_rmse, workflow  0 , degree:  1 ,  0.04297525111970858
test_rmse, workflow  0 , degree:  2 ,  0.0386047579618308
train_rmse, workflow  0 , degree:  2 ,  0.03818439548002145
```

```
test_rmse, workflow  0 , degree:  3 ,  0.03377385252810488
train_rmse, workflow  0 , degree:  3 ,  0.032391464581958455
test_rmse, workflow  0 , degree:  4 ,  0.024722141305702765
train_rmse, workflow  0 , degree:  4 ,  0.023805502694520534
test_rmse, workflow  0 , degree:  5 ,  0.016651824620731937
train_rmse, workflow  0 , degree:  5 ,  0.01575899085949822
test_rmse, workflow  0 , degree:  6 ,  0.016232833011823464
train_rmse, workflow  0 , degree:  6 ,  0.014707211931603672
test_rmse, workflow  0 , degree:  7 ,  0.01137790198823268
train_rmse, workflow  0 , degree:  7 ,  0.008751421629165774
test_rmse, workflow  0 , degree:  8 ,  0.013927253845192191
train_rmse, workflow  0 , degree:  8 ,  0.008474926187516702
test_rmse, workflow  0 , degree:  9 ,  0.024822425185193214
train_rmse, workflow  0 , degree:  9 ,  0.007554792358476207
test_rmse, workflow  0 , degree:  10 ,  0.03300734908058791
train_rmse, workflow  0 , degree:  10 ,  0.0073200575006313415
test_rmse, workflow  1 , degree:  1 ,  0.15949645935571188
train_rmse, workflow  1 , degree:  1 ,  0.15959276434478978
test_rmse, workflow  1 , degree:  2 ,  0.15912725797846788
train_rmse, workflow  1 , degree:  2 ,  0.1573186465357357
test_rmse, workflow  1 , degree:  3 ,  0.15372803484444245
train_rmse, workflow  1 , degree:  3 ,  0.1480079052842391
test_rmse, workflow  1 , degree:  4 ,  0.12749461983903504
train_rmse, workflow  1 , degree:  4 ,  0.12321956713200102
test_rmse, workflow  1 , degree:  5 ,  0.10397022272631193
train_rmse, workflow  1 , degree:  5 ,  0.09962273586574773
test_rmse, workflow  1 , degree:  6 ,  0.08212514482135427
train_rmse, workflow  1 , degree:  6 ,  0.07738389307834655
test_rmse, workflow  1 , degree:  7 ,  0.06127577781367999
train_rmse, workflow  1 , degree:  7 ,  0.05693399761612833
test_rmse, workflow  1 , degree:  8 ,  0.04298206406274489
train_rmse, workflow  1 , degree:  8 ,  0.03790700322413739
test_rmse, workflow  1 , degree:  9 ,  0.031092699547331314
train_rmse, workflow  1 , degree:  9 ,  0.016571462798945308
test_rmse, workflow  1 , degree:  10 ,  0.015324917793198134
train_rmse, workflow  1 , degree:  10 ,  0.005241070751220829
test_rmse, workflow  2 , degree:  1 ,  0.04199382761688264
train_rmse, workflow  2 , degree:  1 ,  0.04223615898138002
test_rmse, workflow  2 , degree:  2 ,  0.04198049372355791
train_rmse, workflow  2 , degree:  2 ,  0.041940817962475216
test_rmse, workflow  2 , degree:  3 ,  0.037469377642255566
train_rmse, workflow  2 , degree:  3 ,  0.037004537390636735
test_rmse, workflow  2 , degree:  4 ,  0.032928978808642616
train_rmse, workflow  2 , degree:  4 ,  0.03234497871818231
```

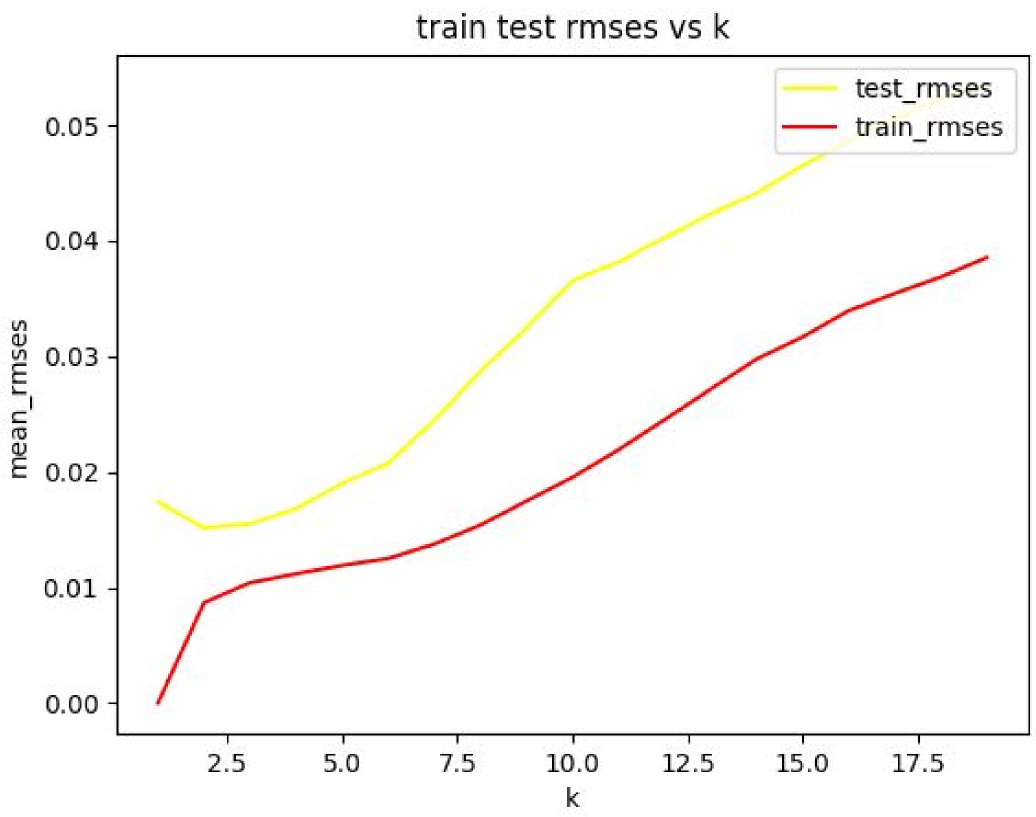
```
test_rmse, workflow  2 , degree:  5 ,  0.03183193840512403
train_rmse, workflow  2 , degree:  5 ,  0.030568553787008197
test_rmse, workflow  2 , degree:  6 ,  0.02562549487957258
train_rmse, workflow  2 , degree:  6 ,  0.02349311006493931
test_rmse, workflow  2 , degree:  7 ,  0.02642020220864636
train_rmse, workflow  2 , degree:  7 ,  0.02091545857895347
test_rmse, workflow  2 , degree:  8 ,  0.030095131230459905
train_rmse, workflow  2 , degree:  8 ,  0.019604000864808714
test_rmse, workflow  2 , degree:  9 ,  0.038517574147953634
train_rmse, workflow  2 , degree:  9 ,  0.017886522659967492
test_rmse, workflow  2 , degree:  10 ,  0.27459188891580405
train_rmse, workflow  2 , degree:  10 ,  0.017023750725439837
test_rmse, workflow  3 , degree:  1 ,  0.007072371214759496
train_rmse, workflow  3 , degree:  1 ,  0.007117121987969881
test_rmse, workflow  3 , degree:  2 ,  0.006838016884920396
train_rmse, workflow  3 , degree:  2 ,  0.006837140790934411
test_rmse, workflow  3 , degree:  3 ,  0.006044055190419834
train_rmse, workflow  3 , degree:  3 ,  0.006008848070157553
test_rmse, workflow  3 , degree:  4 ,  0.005833795591432769
train_rmse, workflow  3 , degree:  4 ,  0.005765394087161567
test_rmse, workflow  3 , degree:  5 ,  0.005423711565364666
train_rmse, workflow  3 , degree:  5 ,  0.005246334266913354
test_rmse, workflow  3 , degree:  6 ,  0.005229926350555998
train_rmse, workflow  3 , degree:  6 ,  0.004667132275746276
test_rmse, workflow  3 , degree:  7 ,  0.006019885407070697
train_rmse, workflow  3 , degree:  7 ,  0.004401683972813092
test_rmse, workflow  3 , degree:  8 ,  0.007102487610334085
train_rmse, workflow  3 , degree:  8 ,  0.004199897171012637
test_rmse, workflow  3 , degree:  9 ,  0.00852167558465463
train_rmse, workflow  3 , degree:  9 ,  0.004047320671621609
test_rmse, workflow  3 , degree:  10 ,  0.01860750731978953
train_rmse, workflow  3 , degree:  10 ,  0.0039013200634168592
test_rmse, workflow  4 , degree:  1 ,  0.10277989283004366
train_rmse, workflow  4 , degree:  1 ,  0.10297632091171938
test_rmse, workflow  4 , degree:  2 ,  0.08526983113427175
train_rmse, workflow  4 , degree:  2 ,  0.08444164887081398
test_rmse, workflow  4 , degree:  3 ,  0.08194540153032412
train_rmse, workflow  4 , degree:  3 ,  0.07940866117418277
test_rmse, workflow  4 , degree:  4 ,  0.06370401639464238
train_rmse, workflow  4 , degree:  4 ,  0.061943316222407875
test_rmse, workflow  4 , degree:  5 ,  0.050879908409826965
train_rmse, workflow  4 , degree:  5 ,  0.04866931135828308
test_rmse, workflow  4 , degree:  6 ,  0.041645140465083975
train_rmse, workflow  4 , degree:  6 ,  0.03816221430519741
```

```
test_rmse, workflow  4 , degree:  7 ,  0.030882626037181404
train_rmse, workflow  4 , degree:  7 ,  0.027032517417602632
test_rmse, workflow  4 , degree:  8 ,  0.029013629632057992
train_rmse, workflow  4 , degree:  8 ,  0.02214434967158869
test_rmse, workflow  4 , degree:  9 ,  0.031127229214722062
train_rmse, workflow  4 , degree:  9 ,  0.017748210470764414
test_rmse, workflow  4 , degree:  10 ,  0.06159583032935263
train_rmse, workflow  4 , degree:  10 ,  0.015122998470034477
```

Q2e. K nearest neighbor regression

Here we train a K-NN regression model for k ranging from 1 to 20 and try to find the best k at which we get optimal value for rmse.

Here is a plot of rmses for different k:



Test and train rmses for each k are as follows:

```
test_rmse, k: 0  , 0.017407110261034614
train_rmse, k: 0  , 0.0
test_rmse, k: 1  , 0.015149849170743163
train_rmse, k: 1  , 0.008683025988898896
test_rmse, k: 2  , 0.0155099696611216
```

```
train_rmse, k: 2  , 0.010412240878535779
test_rmse, k: 3  , 0.016830773180788565
train_rmse, k: 3  , 0.011184526185249618
test_rmse, k: 4  , 0.01899673628681915
train_rmse, k: 4  , 0.01190088776521769
test_rmse, k: 5  , 0.02076559872838425
train_rmse, k: 5  , 0.012498971789202801
test_rmse, k: 6  , 0.02445932931972508
train_rmse, k: 6  , 0.013755052364488093
test_rmse, k: 7  , 0.028708711309163852
train_rmse, k: 7  , 0.015401103733115623
test_rmse, k: 8  , 0.03244050727668646
train_rmse, k: 8  , 0.017466694273721026
test_rmse, k: 9  , 0.03654700965281769
train_rmse, k: 9  , 0.019530007553642575
test_rmse, k: 10  , 0.03816090664490212
train_rmse, k: 10  , 0.021912821910683288
test_rmse, k: 11  , 0.040263385443098175
train_rmse, k: 11  , 0.024508777036152382
test_rmse, k: 12  , 0.042318082711834334
train_rmse, k: 12  , 0.02715388841038891
test_rmse, k: 13  , 0.04411854678622097
train_rmse, k: 13  , 0.029768800493840807
test_rmse, k: 14  , 0.04650669705551942
train_rmse, k: 14  , 0.03167955932187029
test_rmse, k: 15  , 0.04864697661286001
train_rmse, k: 15  , 0.03394701029550154
test_rmse, k: 16  , 0.050522769684044774
train_rmse, k: 16  , 0.03543833347468385
test_rmse, k: 17  , 0.05231889868994873
train_rmse, k: 17  , 0.0368761035248794
test_rmse, k: 18  , 0.05345457990036038
train_rmse, k: 18  , 0.038541346356291896
```

Now we plot the Predicted vs True graphs and Predicted vs Residual graphs using some different k's so that we get an understanding.

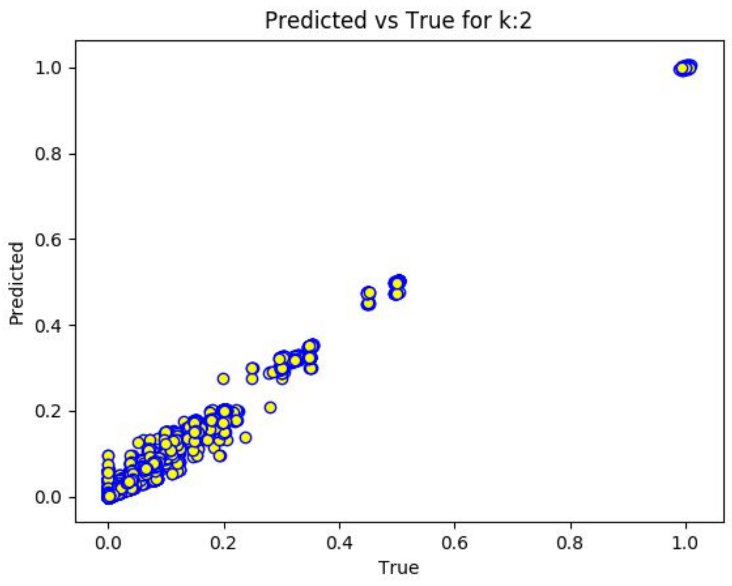


Figure 1.2.5.2: Predicted vs TrueI for KNN regression with k = 2

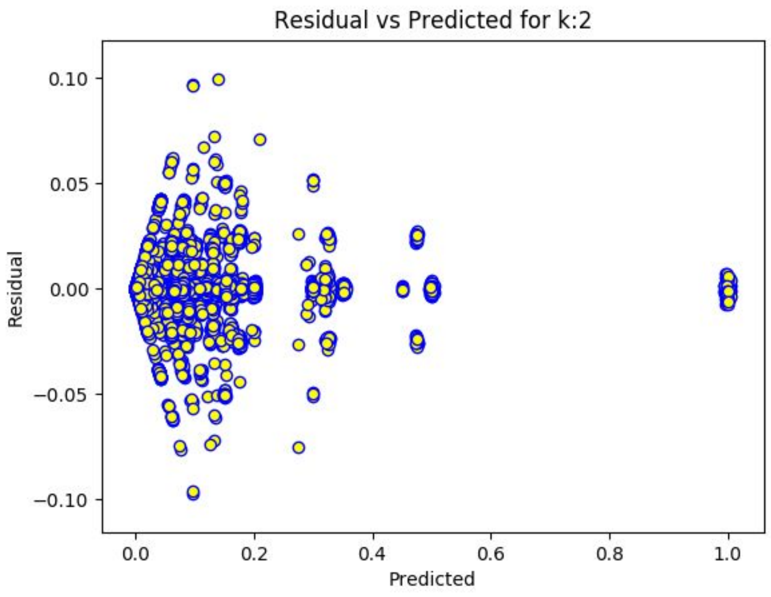


Figure 1.2.5.3: Predicted vs Residual for KNN regression with k = 2

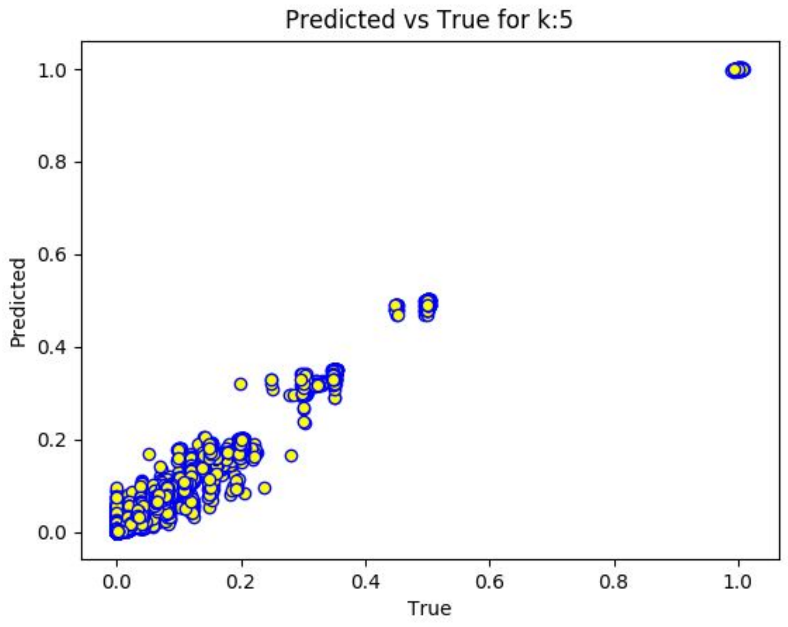


Figure 1.2.5.4: Predicted vs Truel for KNN regression with k = 5

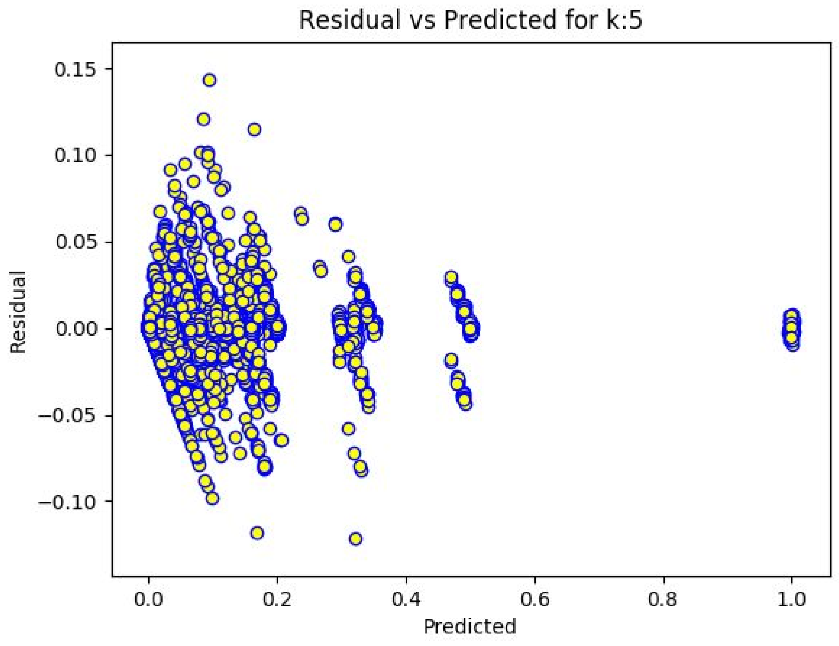


Figure 1.2.5.5: Predicted vs Residual for KNN regression with k = 5

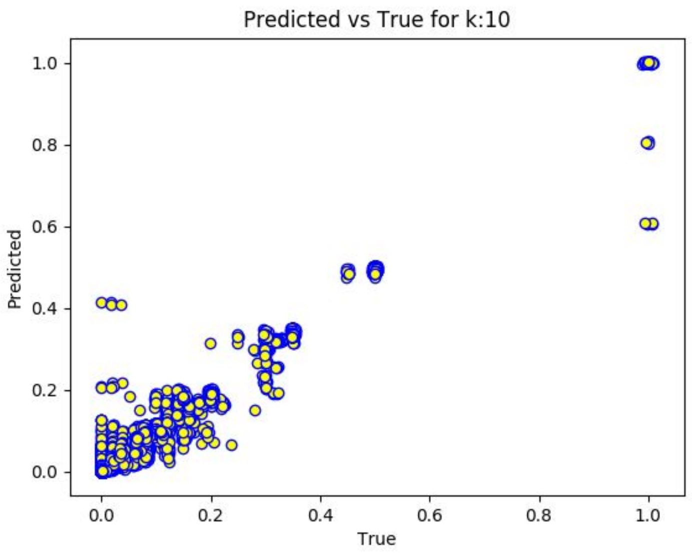


Figure 1.2.5.6: Predicted vs Truel for KNN regression with k = 10

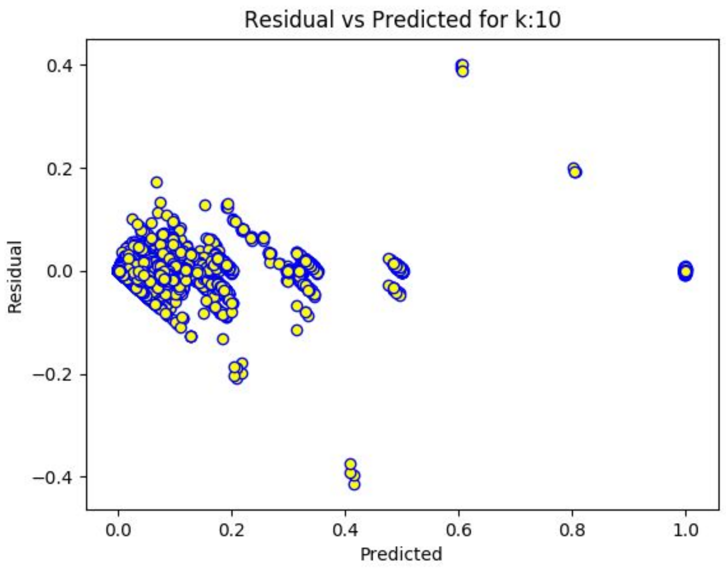


Figure 1.2.5.7: Predicted vs Residual for KNN regression with k = 10

We see that the best k is 2. Also when we look at the Predicted vs Real graphs for k =2, the points seem to be fairly nearer to the x=y axis. Whereas the graphs for k as 5 or 10 looks more dispersed than the graph for k = 2. So the best parameter is k =2

Part 3

Linear Regression can not handle categorical data. Hence, we map categorical features to numerical values to ensure that the model works on our data. Linear regression can work well with sparse features if there is a regularisation parameter involved, for example, L2 regularisation. This helps prevent overfitting.

Random Forest can handle categorical information. However, we feel that it won’t work well with sparse data. Sparse data will lead to splits that are not optimal (due to lack of information to calculate gain). Additionally, due to this a lot of computation power will get wasted. RF performs better than Linear Regression as can be seen in tables mentioned in Dataset 1 Part 2 A and B 1.

Neural networks are found to work best with sparse features when we give it one hot encoded data. If there are high number of hidden units and activation function used is relu, it works really well. On the other hand linear regression models do not do as better as neural networks with sparse features. This could be because the complexity of a function learnt through linear regression is limited to the dimension of input features. But with neural networks, we can fit a highly complex function that is learnt in a higher dimension space by increasing number of hidden units.

In terms of categorical features, K-NN is find to work really good with an optimal k of 2. It even outperforms neural network results. But Random forests are seen to perform even better than K-NN. So random forests give best results for categorical features.

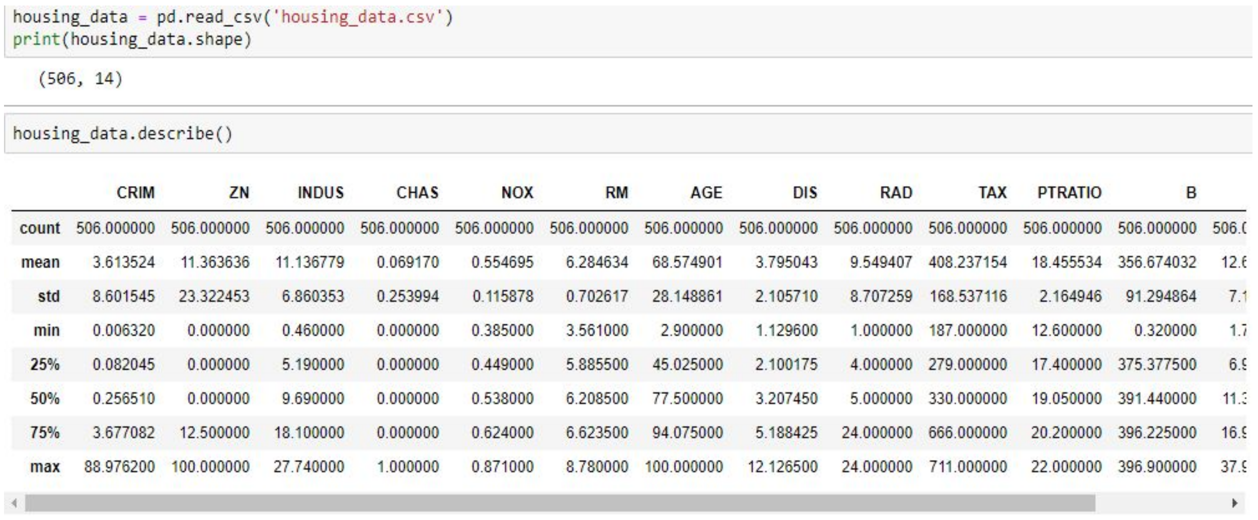
Overall random forests and neural networks are found to generate better results with least test rmses.

Dataset 2

Predict median price value of owner-occupied homes based on the 13 features provided.

Dataset 2 : Part 1 : Loading the dataset

The dataset was loaded and its shape and statistics were computed as a basic analysis step. -



```
housing_data = pd.read_csv('housing_data.csv')
print(housing_data.shape)
```

```
(506, 14)
```

```
housing_data.describe()
```

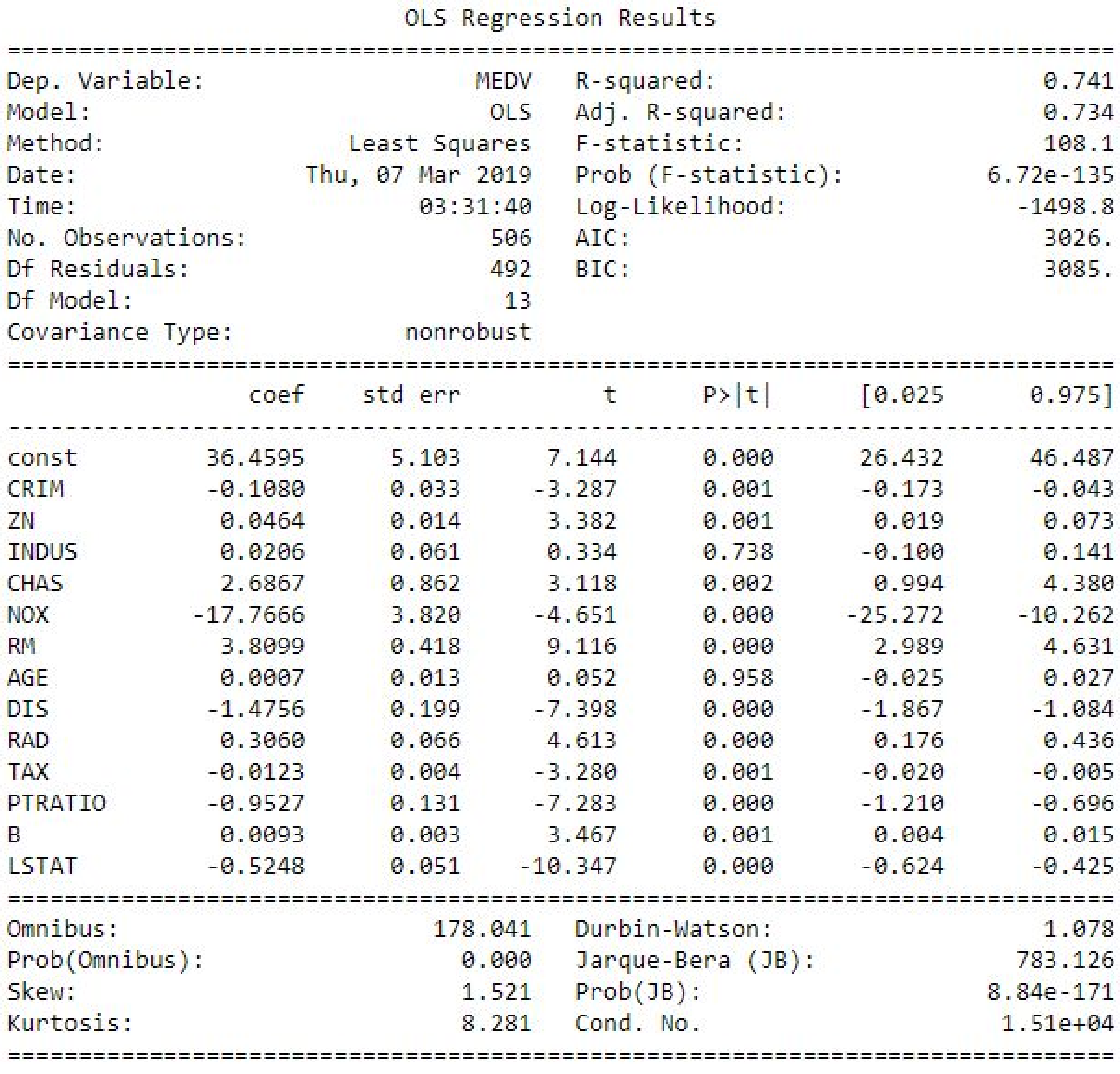
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.

Figure 2.1.1: Visualizing the dataset

Dataset 2 : Part 2 : Fitting a linear regression model

2a) MEDV was set as the target variable, the rest of the columns were used as the features. An OLS model was fit to the data and OLS was set as the penalty function.

2b) An ordinary least squares linear regression model was fit on the entire dataset, so as to obtain the significance values of each of the 13 features in the training data. The following results were obtained -



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.741
Model:                            OLS   Adj. R-squared:                  0.734
Method:                 Least Squares   F-statistic:                     108.1
Date:                Thu, 07 Mar 2019   Prob (F-statistic):          6.72e-135
Time:                        03:31:40   Log-Likelihood:                -1498.8
No. Observations:                 506   AIC:                             3026.
Df Residuals:                     492   BIC:                             3085.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         36.4595      5.103      7.144      0.000      26.432      46.487
CRIM          -0.1080      0.033     -3.287      0.001      -0.173      -0.043
ZN             0.0464      0.014      3.382      0.001       0.019       0.073
INDUS          0.0206      0.061      0.334      0.738      -0.100       0.141
CHAS           2.6867      0.862      3.118      0.002       0.994       4.380
NOX          -17.7666      3.820     -4.651      0.000     -25.272     -10.262
RM             3.8099      0.418      9.116      0.000       2.989       4.631
AGE            0.0007      0.013      0.052      0.958      -0.025       0.027
DIS           -1.4756      0.199     -7.398      0.000      -1.867      -1.084
RAD            0.3060      0.066      4.613      0.000       0.176       0.436
TAX           -0.0123      0.004     -3.280      0.001      -0.020      -0.005
PTRATIO       -0.9527      0.131     -7.283      0.000      -1.210      -0.696
B              0.0093      0.003      3.467      0.001       0.004       0.015
LSTAT         -0.5248      0.051    -10.347      0.000      -0.624      -0.425
==============================================================================
Omnibus:                      178.041   Durbin-Watson:                   1.078
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              783.126
Skew:                           1.521   Prob(JB):                    8.84e-171
Kurtosis:                       8.281   Cond. No.                     1.51e+04
==============================================================================
```

Figure 2.2.b: Significance values and coefficients for OLS

Based of the significance values, we can see that the following two features - **INDUS** and **AGE** (the 3rd and 7th features in the data) are both insignificant, since their p-values are greater than the threshold.

After obtaining these, we have performed 10 - fold cross validation and calculated the average RMSE obtained for each train and test fold. From the values obtained above, we were able to obtain the minimum average train and test RMSE as -

Min Train Average RMSE - **4.609066917948425**
Min Test Average RMSE - **5.18084567934025**

The following two plots were also obtained -
The first plot has fitted values as Y-Axis and the true values as X-Axis. The blue color indicates training points and the orange color indicates the testing points.

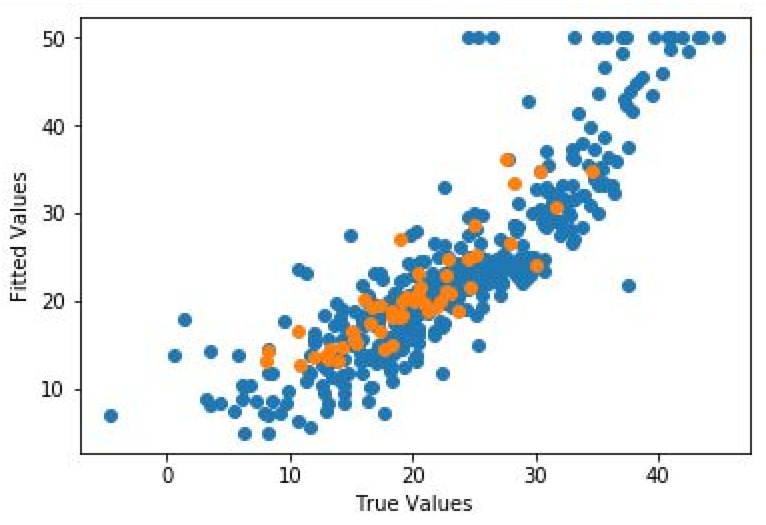


Figure 2.2.b.1: Fitted Values vs True Values

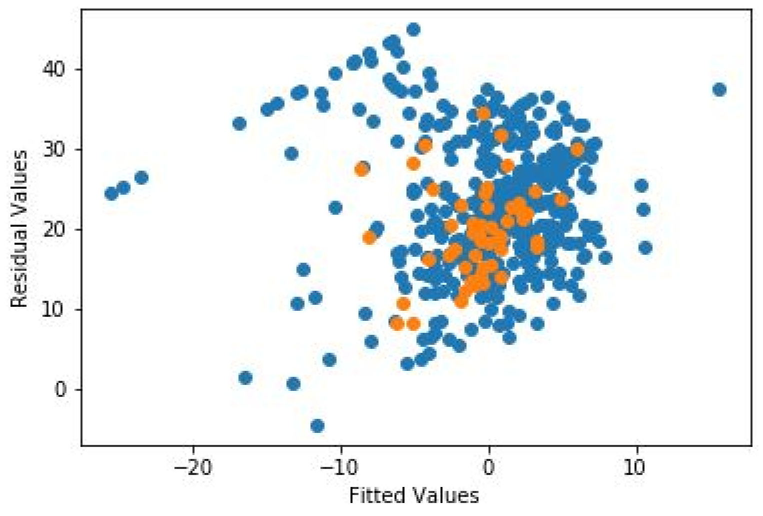


Figure 2.2.b.2: Residual Values vs Fitted Values

Dataset 2 : Part 3 : Performing regularization

The point of regularization is to prevent overfitting and to “smoothen” the function which is fit for the data. Here we have performed the regularization using three techniques and we will compare which technique performs the best.

3a.1) Ridge Regularization -

This is the Tikonov regularization and puts a penalty on the weights, the penalty in this case being the L2 norm of the weights. What this intuitively means is that larger weights are penalized while smaller weights aren’t penalized as much. Therefore, the model will have smaller parameter values and hence the function fit by the model will be smoother. The following values were obtained over the entire dataset. We can also see below that the coefficients obtained are far lesser in absolute value than the ones obtained from OLS.

The model is now affected less by noisy data points (more robust) and we can see that the overall RMSE obtained now was far lesser than the OLS case (~5.18).
Best RMSE - **4.679573505950083**
The best alpha obtained was **0.1**.

Column Name	Ridge Regularization Coefficients
CRIM	-1.07473720e-01
ZN	4.65716366e-02
INDUS	1.59989982e-02
CHAS	2.67001859e+00
NOX	-1.66846452e+01
RM	3.81823322e+00
AGE	-2.69060598e-04
DIS	-1.45962557e+00
RAD	3.03515266e-01
TAX	-1.24205910e-02
PTRATIO	-9.40758541e-01
B	9.36807461e-03
LSTAT	-5.25966203e-01

Table 2.3.a.1: Ridge Regression coefficients

3a.2) Lasso Regularization -

The Lasso regression applies the L1 norm on the weights. Larger weights aren't penalized as much as Ridge Regression. Weights close to 0 will becoming squashed and hence Lasso regularization is used for feature selection. The following values were obtained over the entire dataset. We can also see below that the coefficients obtained are far lesser in absolute value than the ones obtained from OLS.

The model is now affected less by noisy data points (more robust) and we can see that the overall RMSE obtained now was far lesser than the OLS case (~5.18).

Best RMSE - **4.683094652065242**
The best alpha obtained was **0.01**.

Column Name	Ridge Regularization Coefficients
CRIM	-1.06228354e-01
ZN	4.68603033e-02
INDUS	6.47427472e-03
CHAS	2.50419017e+00
NOX	-1.43944776e+01
RM	3.81418583e+00
AGE	-1.81830336e-03
DIS	-1.42215548e+00
RAD	2.98525812e-01
TAX	-1.26273172e-02
PTRATIO	-9.16368723e-01
B	9.48546597e-03
LSTAT	-5.30481493e-01

Table 2.3.a.1: Lasso Regression coefficients

Note how in L1 regularization, the coefficients which are already low (as obtained in L2) are even lower now.

3a.3) Elastic Net Regularization -

Elastic Net tries to find a balance between L1 and L2 regularization. We have tested over multiple values of alpha, l1 and l2 (the latter two of which controlled by the parameter called l1_ratio in the code).

The values of alpha over which we tested are - **[0.0001, 0.0005, 0.001,0.005,0.01]**
The values of l1_ratio over which we tested are - **[0.01,0.1,0.5,0.9,0.99]**

RMSE - **4.684092931069658**
The best alpha obtained was **0.01**, and best L1_ratio obtained was **0.99.**
Even in this case we see that the coefficient values have reduced than the case of OLS.

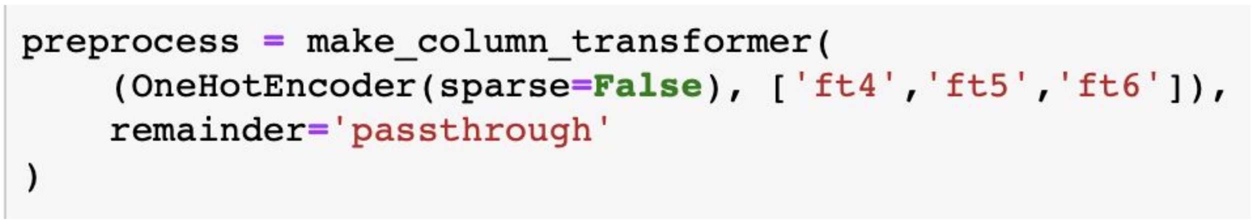
We conclude that **Ridge Regression** performed the best, since it reduced the overall RMSE by the greatest margin.

Dataset 3

Predict user's car insurance based on the 6 features provided.

Dataset 3 : Part 1 : Feature Preprocessing

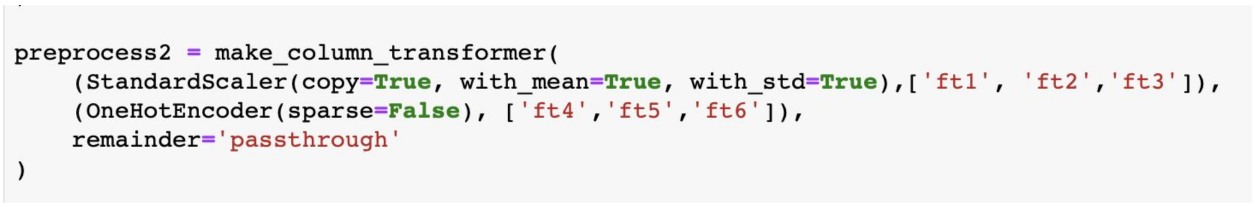
a) Feature Encoding : One hot encoding ft4, ft5, ft6



```
preprocess = make_column_transformer(
    (OneHotEncoder(sparse=False), ['ft4','ft5','ft6']),
    remainder='passthrough'
)
```

Figure 3.1.1 : One Hot encoder

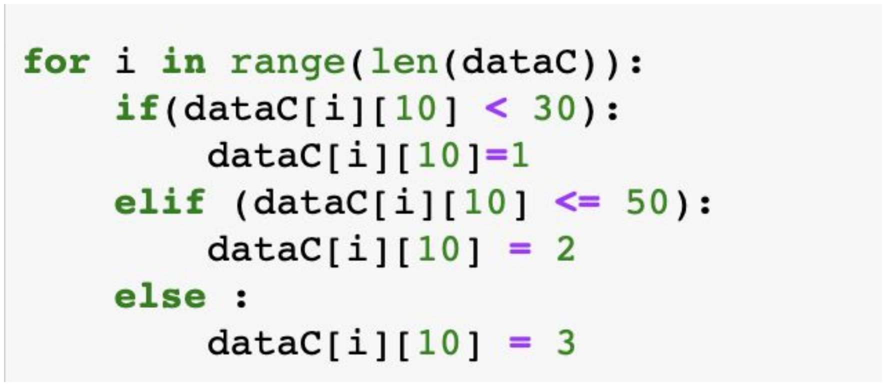
b) Standardization : First 3 numerical features



```
preprocess2 = make_column_transformer(
    (StandardScaler(copy=True, with_mean=True, with_std=True),['ft1', 'ft2','ft3']),
    (OneHotEncoder(sparse=False), ['ft4','ft5','ft6']),
    remainder='passthrough'
)
```

Figure 3.1.2 : Standardization

c) Dividing ft1 into 3 ranges and others as previous step



```
for i in range(len(dataC)):
    if(dataC[i][10] < 30):
        dataC[i][10]=1
    elif (dataC[i][10] <= 50):
        dataC[i][10] = 2
    else :
        dataC[i][10] = 3
```

Figure 3.1.3 : Split into ranges the specific ft1

Q 1. Reporting the average training RMSE and average test RMSE for 10 fold validation

Model	Train RMSE	Test RMSE
A - One hot encoding	6039.568	6081.95
B - Standardization	6039.568	6081.96
C - ft1 into ranges	6198.08	6239.96

Table 3.1.1 : Train and Test RMSE

Q 2. Plotting fitted value vs true values as scatter plots

As discussed during the discussions we use data index as X-axis, fitted values and true values as Y axis (using different colors. We can see that there is not much difference between the graphs as expected because the RMSE errors are identical. We also tried to plot the graphs on a smaller range of X-axis to check the readings properly (Graphs present in notebook submitted)

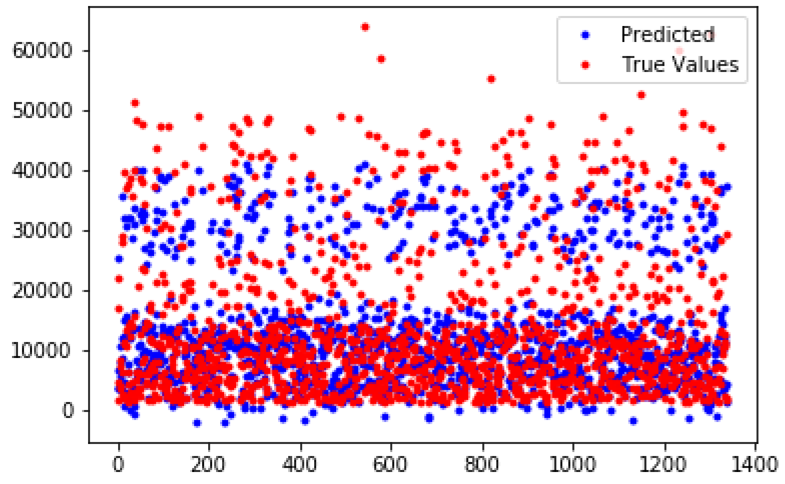


Figure 3.1.4 : Predicted and true values vs data index on the first preprocessed data

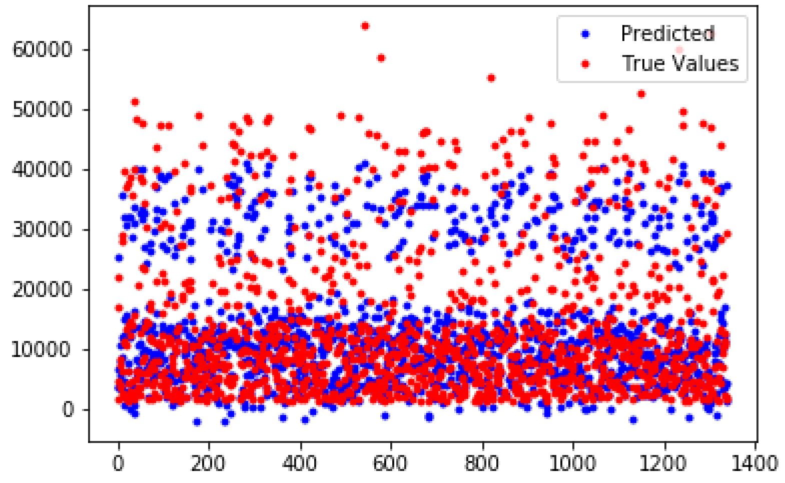


Figure 3.1.5 : Predicted and true values vs data index on the second preprocessed data

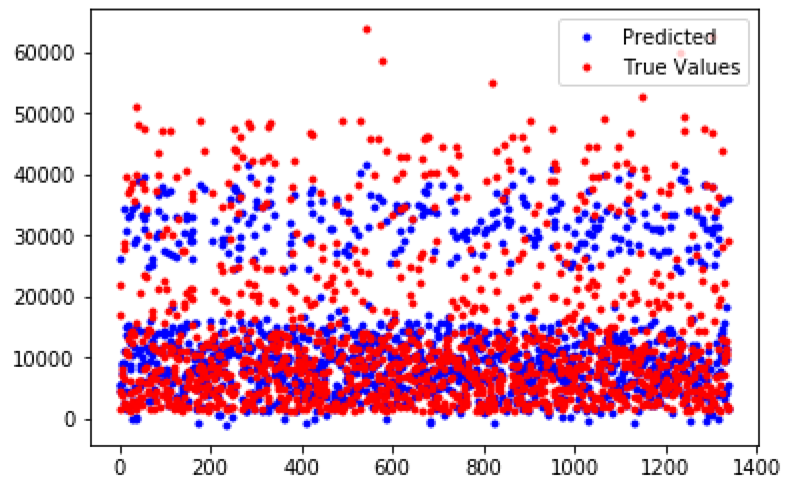


Figure 3.1.6 : Predicted and true values vs data index on the third preprocessed data

Q 3. Plotting residuals vs fitted values as scatter plots

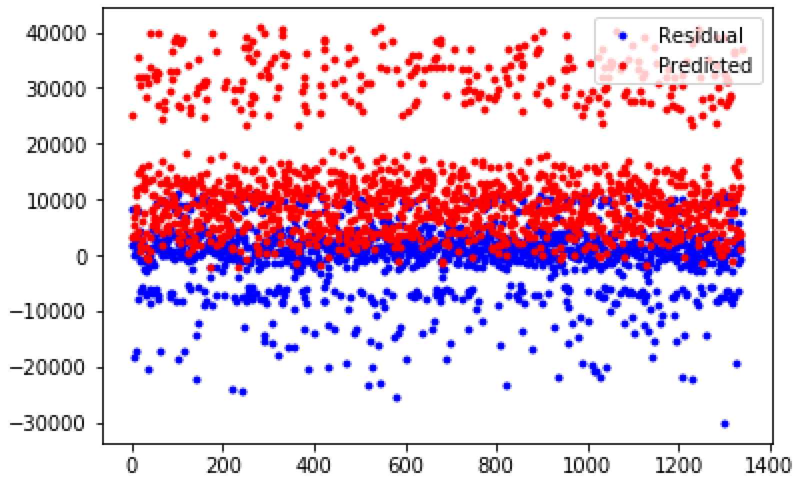


Figure 3.1.7 : Residual and fitted values vs data index on the second preprocessed data

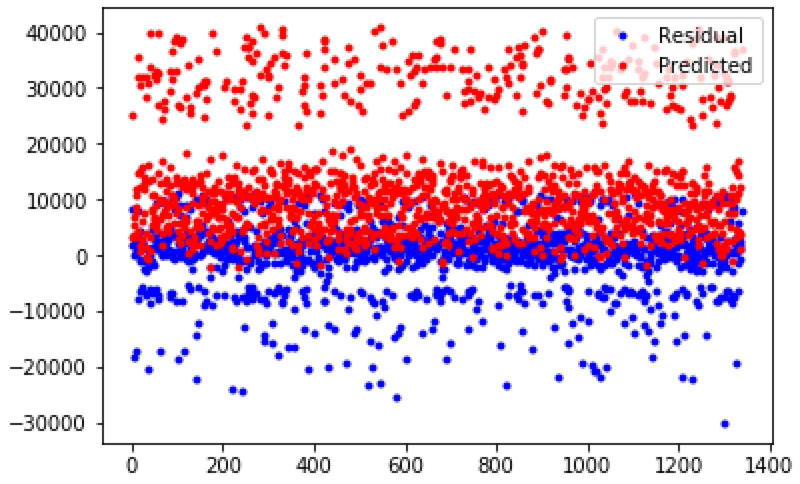


Figure 3.1.8 : Residual and fitted values vs data index on the first preprocessed data

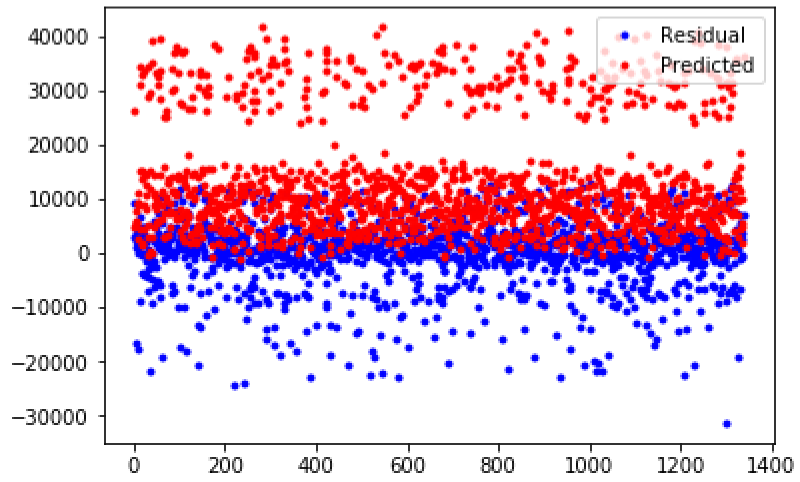


Figure 3.1.9 : Residual and fitted values vs data index on the first preprocessed data

Dataset 3 : Part 2. Correlation Exploration

a) Converting each categorical feature into numerical values and finding most important features

Using f_regression - The best features using the f-scores are ft1 and ft5 with values 131.17 and 2177.61 respectively.

Using mutual information - The best features found are are ft1 and ft5 with values 1.491 and 0.369 respectively.

It is clear from the measures that the most important features are ft1 and ft5.

b) Scatter plot charges vs ft2 (based on ft5 as color)

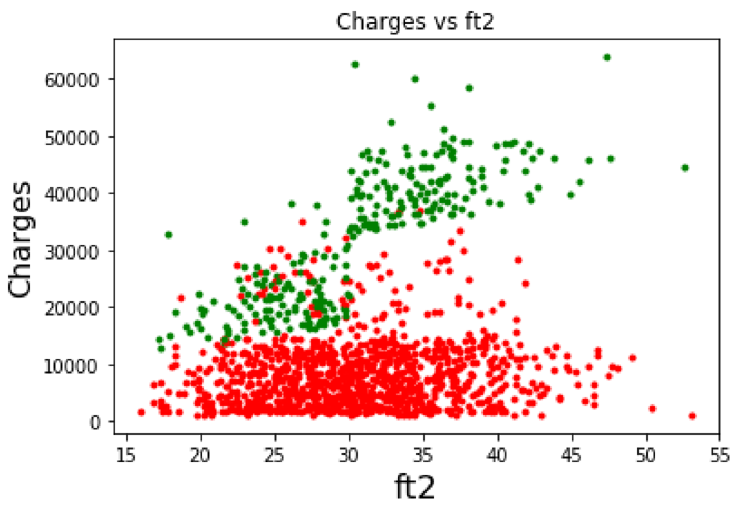


Figure 3.2.1 : Scatterplot of charges vs ft2 (Red shows ft5 = 0 and green shows ft5 = 1)

c) Scatter plot charges vs ft1 (based on ft5 as color)

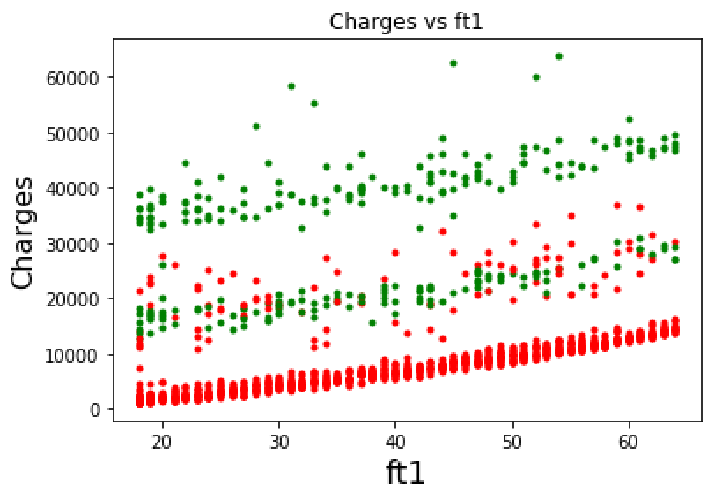


Figure 3.2.2 : Scatterplot of charges vs ft1 (Red shows ft5 = 0 and green shows ft5 = 1)

Dataset 3 : Part 3. Modify the target variable

Fitting the data with log(charges) instead of charges completely.

a) Using one preprocessing method :

I used the data from the part 3 in question one. Using standardization and one hot encoding along with range change for ft1.

After performing kfold we see that the train RMSE and test RMSE has increased considerably. The values are **8141.27** and **8204.30** respectively.

The reason is that there is a range in which it is unable to predict the correct range. As per part 2, we see that 2 features are the most significant and the upper and lower values are easily differentiable with ft5. The mid values are jumbled and using the log transformation increases this region where decision making is tough. This further degrades the performance.

Plotting the transformed y charges against the predicted values, we get the clear picture of the above reasoning. Using only first 200 points, we see that predicted values are

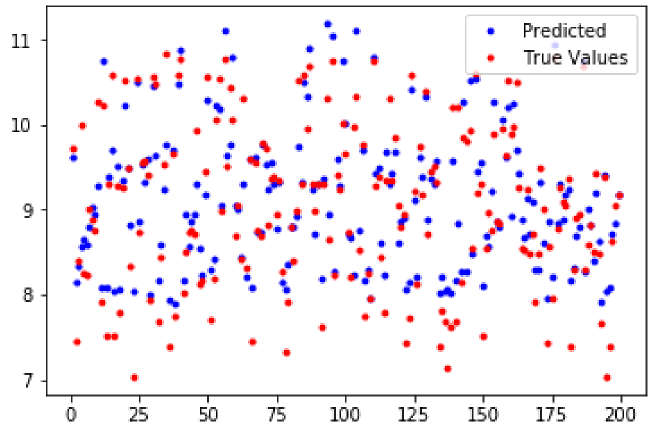


Figure 3.2.3 : Predicted and true values vs data index using Log transformation

b) Repeating correlation exploration :

Most Important features :

Using f_regression - The best features using the f-scores are ft1 and ft5 with values 515.97 and 1062.12 respectively.

Using mutual information - The best features found are are ft1 and ft5 with values 1.49 and 0.36 respectively.

It is again clear from the measures that the most important features are ft1 and ft5.

Scatter plot charges vs ft2 (Based on ft5 as color)

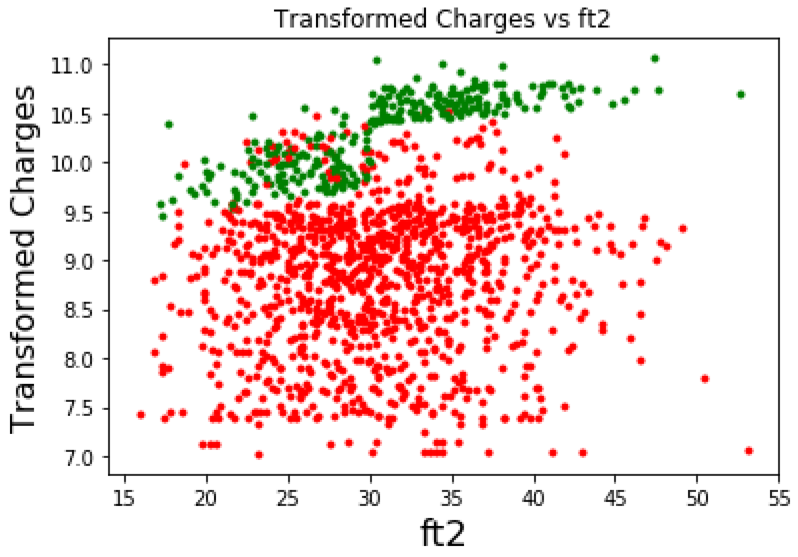


Figure 3.3.1 : Scatterplot of charges vs ft2 (Red shows ft5 = 0 and green shows ft5 = 1)

Scatter plot charges vs ft1 (Based on ft5 as color)

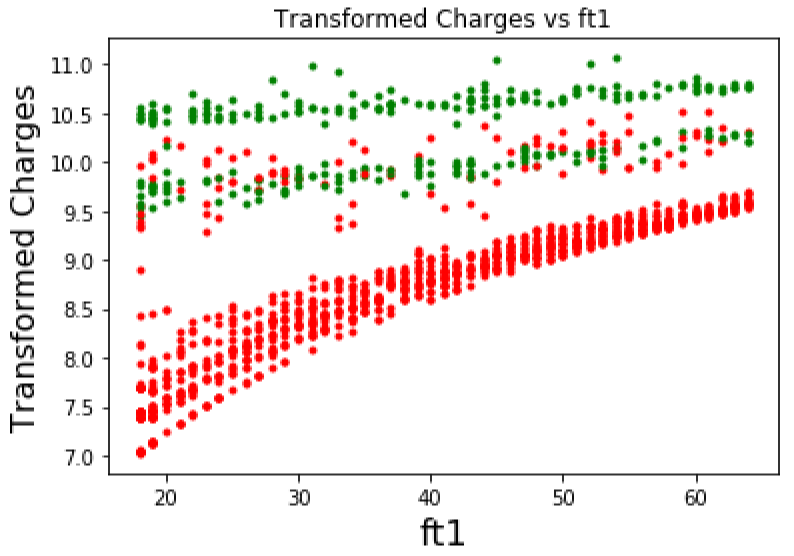


Figure 3.3.2 : Scatterplot of charges vs ft1 (Red shows ft5 = 0 and green shows ft5 = 1)

Dataset 3 : Part 4. Bonus Question

Part a.

Yes, we could improve our results by using better pre-processing. The following are the things we tried and the improvements mentioned.

We used the data generated in using one hot encoder and standardization. Then we performed polynomial transformation to preprocess this data.

The resulting RMSE train vs RMSE test values decreased considerably and come out as 4743.50 and 4889.71 respectively.

We see an improvement of 20% which is quite significant.

The next we tried a lot of permutations before grid search, using different preprocessing like Robust Scaler and Quantile Transformer.
The resulting RMSE train vs RMSE test values decreased again and come out as 4533.72 and 4690.15 respectively.

We see an overall improvement of 24% which is even better. So by just changing the preprocessing techniques we get an improvement from the one hot encoders by a margin of 25%!

Bonus Part b.

As mentioned, we tried different models to train our data on. We also tried different preprocessing as well as different parameters to check the result.

The models used for part 1 were as follows :

1. Neural Network - MLPRegressor
2. Random Forest - Random Forest Regressor
3. Decision Tree Regressor with AdaBoostRegressor
4. Gradient Boosting Regressor - Gradient Boosting Regressor

The best result I get is for the **Random Forest** with train RMSE as 1872.18.
This gives us more than **62%** improvement from the previous bonus question. The consolidated results can be found in the table below.

Consolidated Results for GridSearch on DataSet 3

Model	Train RMSE	Test RMSE
Random Forest	1872.18	4872.04
Neural Network	4073.56	4891.95
Gradient Boosting Regressor	3839.10	4554.37
Decision Tree Regressor	6405.08	6161.96

Table 3.3.1 : Best Train and Test RMSE

Conclusion

For dataset 1, we noticed that our best Random Forest (configuration: trees- 57, maximum features- 4 and maximum depth- 10) and our best Neural Network (configuration: hidden units- 600, activation function- relu) gives us similar training and testing rmse at 0.01. We see that both RF and NN are good at handling categorical data unlike Linear Regression which requires us to map the categorical data to numerical values.

In dataset 3, our car insurance predicted is heavily dependent on two features, ft1 and ft5. We use one hot encoding and scalar encoding as feature preprocessing technique to convert categorical data to numerical data. We see both perform similarly. Polynomial function, as a preprocessing step, works best with Random Forest giving us training and testing RMSE as ~1800 and ~4500 respectively. Here we notice, that unlike dataset 1, where our NN performs similar to best RF, our best NN is as good as Linear Regression and performance is not as good as RF.

The best feature pre-processing stage for KNN regression was to standardize and center each data points. For categorical features, it is to turn the features into categories. Also one hot encoding is useful for doing NN models. Also, for continuous datasets (such as dataset 2), before performing OLS or fitting any linear regression model, a constant should be added to every data point to fit the intercept.

We can say that we had majorly two different types of datasets. The first and last datasets had almost similar input categorical features while the second dataset had continuous features. We found that in the first category of datasets, neural networks and random forests perform better. So we can say that if we have categorical features as input and the dataset is not too sparse, NN and random forests work better. The second category of dataset we had was dataset 2 where features were continuous and here Ridge regression is seen to perform better than any other model. So we can say that for such dataset we will use regression.