Project 4
Regression Analysis

Shivani Soman - 304946159
Maithili Bhide - 104943331
Satya Vasanth Reddy - 405029459

Introduction

In this project we will be performing Regression Analysis on the Network Backup Dataset. Weexplore regression models such as Linear Regression, Random Forest, Neural Network and K-Nearest neighbors to predict our target variable - Backup size. Our goal is to estimate the relationship between a target variable and a set of potentially relevant variables. We also employ various techniques such as cross-validation and regularization. Cross-validation tests for overfitting while regularization penalizes more complex models. We also compare different models and analyse how they behave on the give dataset and draw various conclusions

Dataset

For this project we are using a Network backup Dataset, which is comprised of simulated traffic data on a backup system over a network. The system monitors the files and runs backup cycles every 4 hours that log the size of the data moved to the destination as well as the duration it took. A workflow is a task that backs up data from a group of files, which have similar patterns of change in terms of size over time. The dataset has the following features -

1. Week index: Week number
2. Day of week: Categorical value (Monday-Sunday) indicating when backup took place
3. Backup start time: Hour of the day
4. Workflow ID: Categorical value (wok_flow_0 - work_flow_4)
5. File Name: Categorical value (file_0 - file_29)
6. Backup size: Size of the file that is backed up in that cycle in GB
7. Backup time: Duration of the backup procedure in hour

Through the course of the questions in section 2, we will be working on modifications of the categorical and numeric features 1-5. We will use scaler encoded features, one-hot-encoded features and different combinations of scalar and one-hot-encoded features to predict the Backup size.

1. Plots of backup sizes against day number for all workflows

For this section we have calculated the day number for each datapoint in the dataset and introduced that as another feature ‘Day #’. To calculate the day number, we are using the features ‘Week #’ and ‘Day of Week’ as follows -

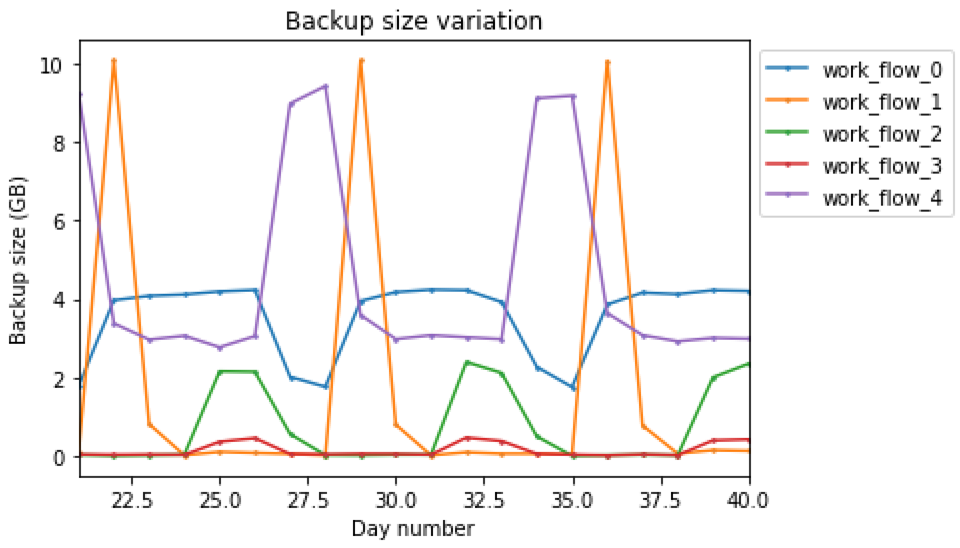
```
dataset['Day #'] = (dataset['Week #'] - 1) * 7 + encode_day_names(dataset['Day of Week'])
```

Implementation

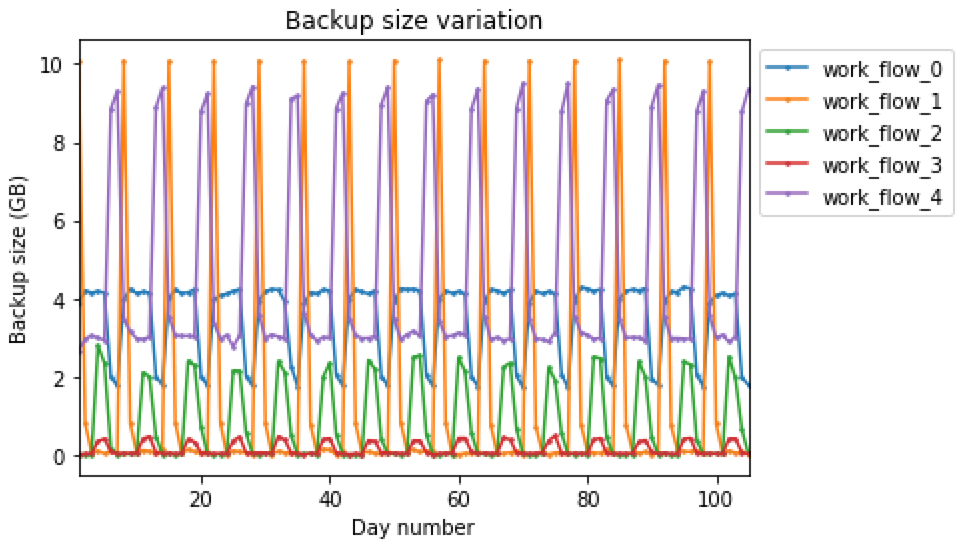
The function encode_day_names() returns the numerical value for Day of Week where the values Monday - Sunday are encoded as 1 - 7.

To get the plots for this section, we are grouping all the data points by their Workflow ID and then summing up the Backup size for all files in that particular workflow.

For the first plot, we are selecting a 20 day period between day 21 - day 40 instead of just selecting the first 20 days to better visualize the peaks for each workflow.



For the second plot, we are selecting the first 105 day period and plotting a similar graph.



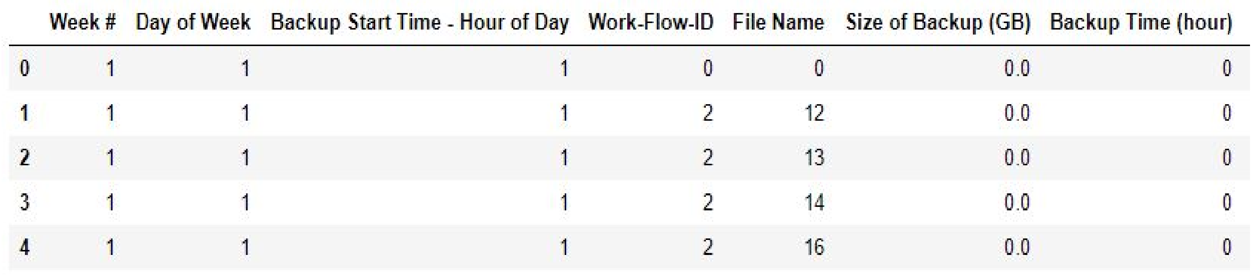
From the above graphs we can see that each of the workflows have a repeating pattern that repeats over a 7 day period, that is every week. From the first graph we can also infer that the Backup size for workflows 1 and 4 is significantly higher than that for the other workflows.

2.a Linear Regression

Part i)

Implementation:

In order to perform linear regression, we first converted all the categorical variables into numerical features using scaler encoding. The variables that underwent scaler encoding were 'Workflow ID','Day of Week' and 'File Name'. For example, the variable 'Day of Week' was converted to numerical values of [1,2...7] corresponding to values of Monday to Sunday. The following table shows the first five entries of our dataset after carrying out scaler encoding.



	Week #	Day of Week	Backup Start Time - Hour of Day	Work-Flow-ID	File Name	Size of Backup (GB)	Backup Time (hour)
0	1	1	1	0	0	0.0	0
1	1	1	1	2	12	0.0	0
2	1	1	1	2	13	0.0	0
3	1	1	1	2	14	0.0	0
4	1	1	1	2	16	0.0	0

We used all these features except 'Backup Time' as candidate features to predict the variable 'Size of Backup'.

Just to carry out an initial observation on linear regression, we first fit the basic linear regression model without using any cross validation approaches using a split of 90% training and 10% testing data. We achieved the following results.

Coefficients	-4.46137250e-05	-2.40527826e-03	1.38306408e-03	-4.14520552e-04	4.85929375e-04
Intercept	0.049413880522				
Test RMSE	0.106503866998				

We then carried out 10 fold cross-validation on the entire dataset for better evaluation of the performance of the regressor. We also calculated the Training and Testing RMSE in each one of the ten folds as well as Average Training and Testing RMSE over all of the folds. The average RMSE values were calculated by taking the average of the Mean Squared Error (MSE) over the ten folds and then taking the square root of the average MSE. We used the following formula -

$$\mathrm{RMSE}_{train} = \sqrt{\frac{mse_{train,1} + \cdots + mse_{train,10}}{10}}$$

We obtained the following Training and Testing RMSE values -

Fold	**Training RMSE**	**Testing RMSE**
1	0.103265021252	0.106503866998
2	0.103428926322	0.105029409306
3	0.101891490799	0.117793845398
4	0.104325747394	0.0967244993663
5	0.104118712604	0.0987136373113
6	0.104606731957	0.0939770759809
7	0.103710941139	0.102543127434

8	0.102680720217	0.111465416023
9	0.103622312889	0.103306632317
10	0.104198598987	0.0979498611003
Average	**0.103587856885**	**0.103627896737**

Analysis:

As observed from the values, the average Test RMSE obtained from cross-validation is lesser than the one achieved earlier without cross-validation. This is because we averaged over all different values obtained from splitting the dataset into ten parts, wherein some sections observed high Test RMSE whereas in other sections the RMSE was considerably lower. This shows that cross-validation provides a more complete performance evaluation of the regressor as compared to just splitting the dataset once.

We then used linear regression to train and test the model on the entire dataset of scaler encoded values. We calculated residual values of each datapoint in the dataset for better understanding of the performance.

The residual values were calculated by subtracting the fitted values from the actual values as follows -

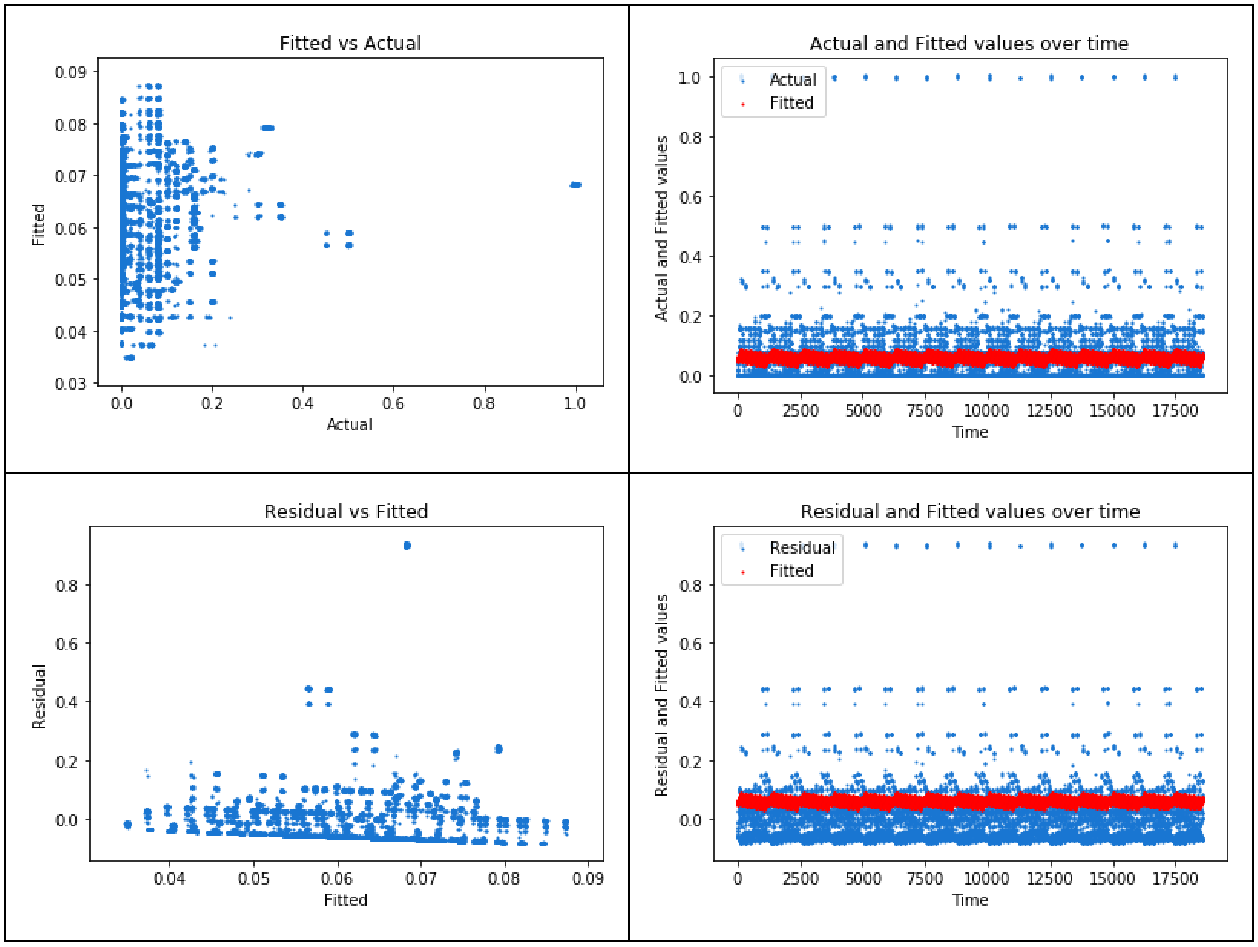
$$\textbf{\textit{Residual}}_i = \textbf{\textit{Actual}}_i - \textbf{\textit{Fitted}}_i$$

Where,
i - index of datapoint
Actual_i - True value of datapoint i
Fitted_i - Predicted value of datapoint i

We also plotted four graphs in total -

1. Fitted Values (y-axis) vs Actual Values (x-axis)
2. Residual Values (y-axis) vs Fitted Values (x-axis)
3. Fitted & Actual Values (y-axis) vs Time/Data Entries (x-axis)
4. Residual & Fitted Values (y-axis) vs Time/Data Entries (x-axis)

The four graphs we obtained are as follows -



Analysis:

From the graphs we can observe that as the data is not linear, linear regression has very poor performance on this dataset. For better performance, the residual values should be as close to zero as possible but as the graphs show, residual values are pretty high.

Part ii)

In part ii) we performed data preprocessing on the dataset by standardizing all the numerical features. We used the StandardScaler library in sklearn.preprocessing for this step. Standardization centers the data before scaling and then scales it to unit variations. This is useful in cases if a feature has a variance that is of a higher magnitude than others, it might dominate the objective function and make the estimator unable to learn from other features correctly. In this section, we performed standardization on the five features and not on the target variable.

Implementation:

We first standardized the entire dataset and then fit the basic linear regression model on the standardized data without using any cross validation approaches using a split of 90% training and 10% testing data. We achieved the following results.

Coefficients	-0.00019274	-0.00479403	0.00944936	-0.00058665	0.00420832
Intercept	0.0607646798087				
Test RMSE	0.106503866998				

Analysis

From the table above, we can see obvious changes in the coefficient and intercept values obtained after standardization as compared to the values obtained in part i). Standardization makes the units of the coefficients of regression same, so that it is much easier to compare them.

However, no change can be observed in the test RMSE obtained before and after standardization. This shows that standardization does not improve or decrease the performance in case of linear regression.

We then carried out 10 fold cross-validation by standardizing the data and then fitting and testing the model. The Training and Testing RMSE values we obtained in each fold as well as the average Training and Testing RMSE were identical to the results obtained in part i). The average Training and Testing RMSE values are shown below.

Average Training RMSE	**Average Test RMSE**
0.103587856885	0.103627896737

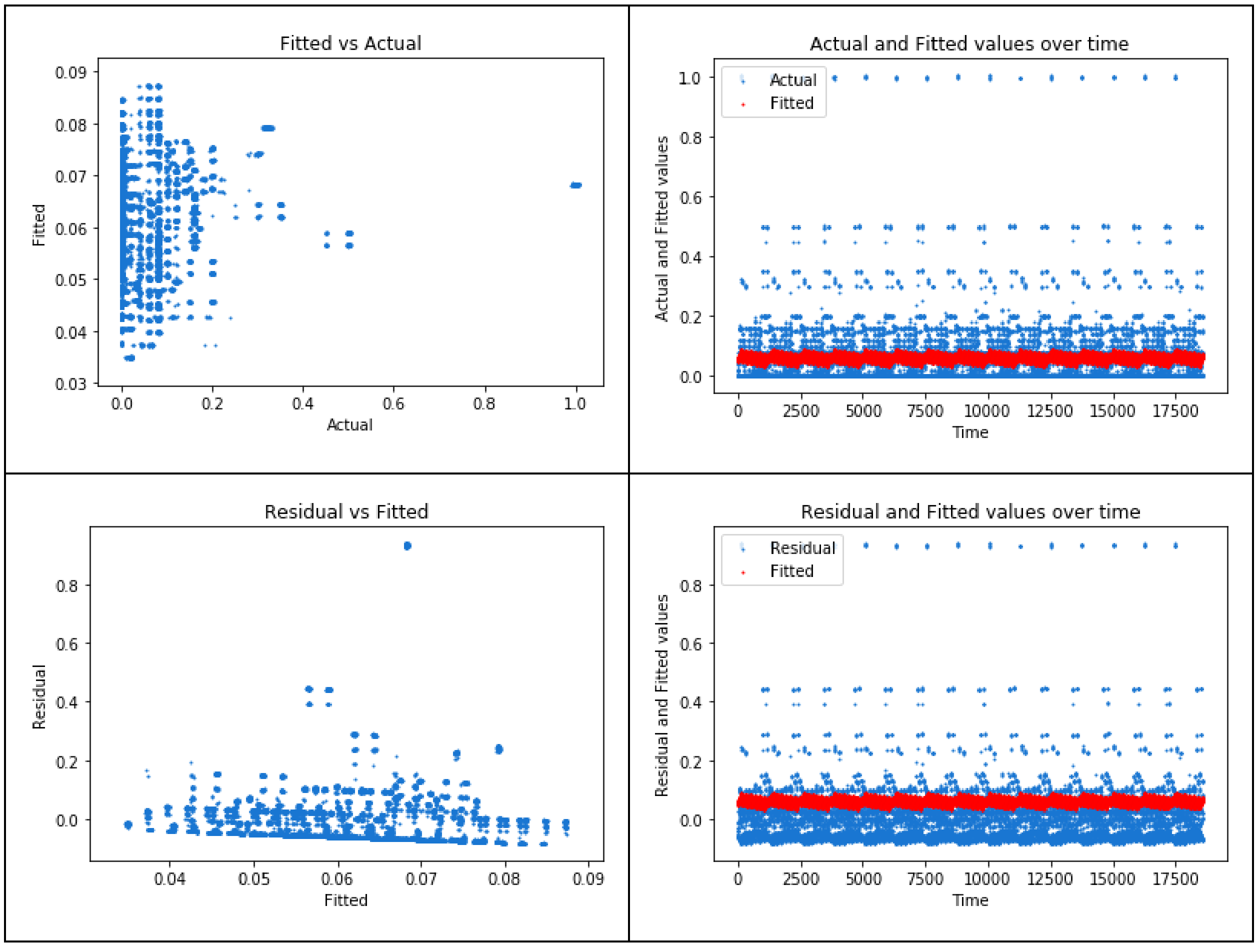
This shows that the accuracy of the model does not change after standardization. Considering a model -

$$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\epsilon$$

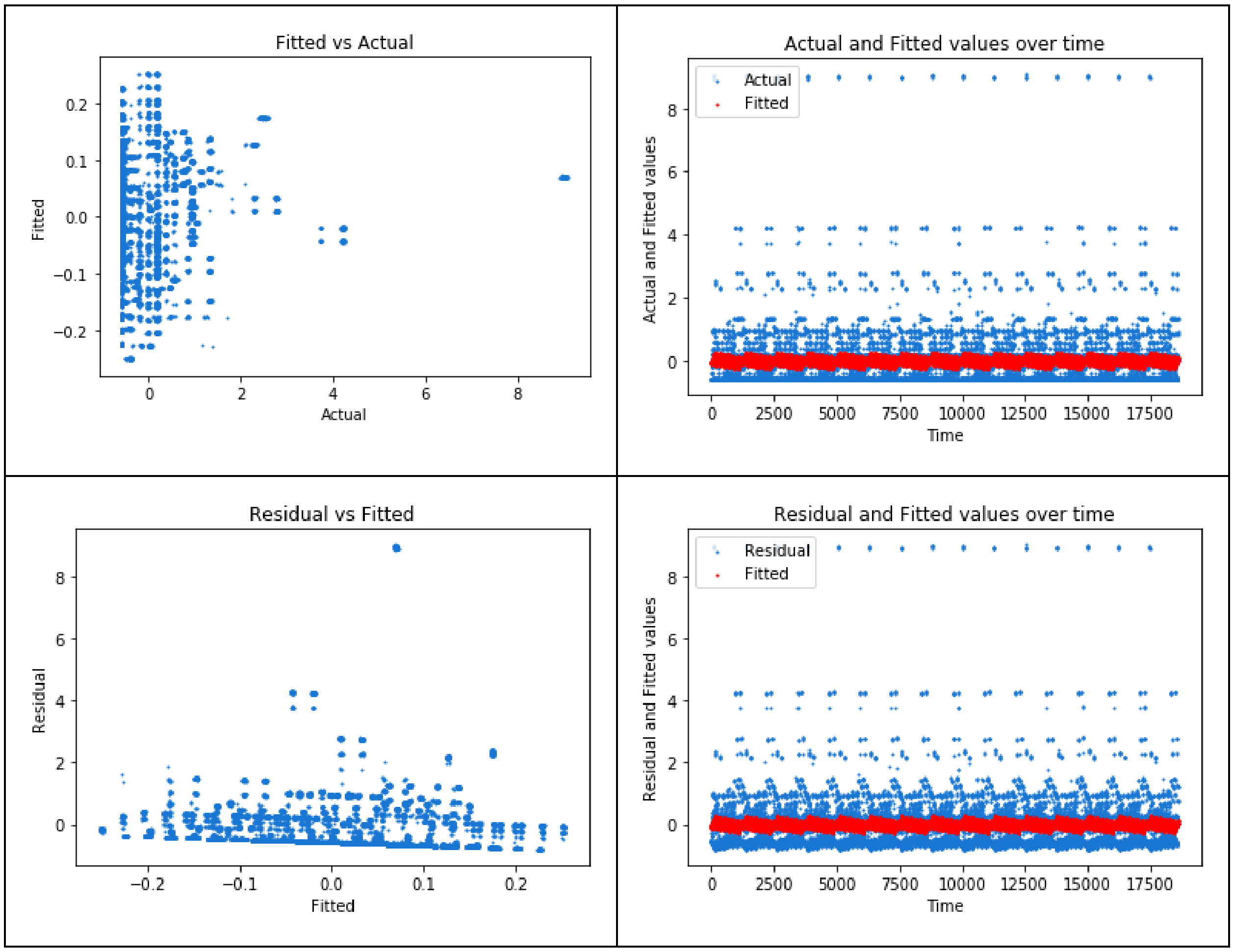
The values of β0,β1,etc. Correspond to the slopes of the fitting surface i.e. how much the surface changes when you change x1,x2, etc. by one unit. Scaling the values of x1,x2, etc. correspondingly scales the slopes and does not have any effect on the least squares estimator.

How does the fitting result change as shown in the plots?

We then fit the model again on the entire standardized dataset and plotted the four graphs as mentioned in part i). The graphs showed no difference from the ones obtained in part i). Also, we haven't standardized the target variable so there is also no change in the true values of the data points.



If we do carry out standardization on the target variable, the plots are as follows -



As you can see from the plots, standardizing the target variable only causes the values to have zero mean and unit variance, but does not change the performance of the regressor.

Part iii)

In this section, we performed feature selection to select the top three features that give the best score. We used f_regression and mutual information regression measures as evaluation metrics for this purpose. We also used the SelectKBest() function from the sklearn.feature_selection library. It is used to remove all but the top k best scoring features from the dataset.

Metric : f_regression

We first used the metric f_regression to find the features with the top scores in the dataset. We obtained the following f_regression metric scores for each of the features -

Week #	Day of Week	Backup Start Time - Hour of Day	Workflow ID	File Name
8.45006257e-03	3.88163798e+01	1.50740934e+02	2.61386654e+01	2.53200943e+01

From the table, we can observe that the top three features obtained from f_regression are : 'Day of Week', 'Backup Start Time - Hour of Day' and 'Workflow ID'. The least important feature found is 'Week #', with a drastically low score as compared to others, which shows that 'Week #' is not statistically significant in predicting target values.

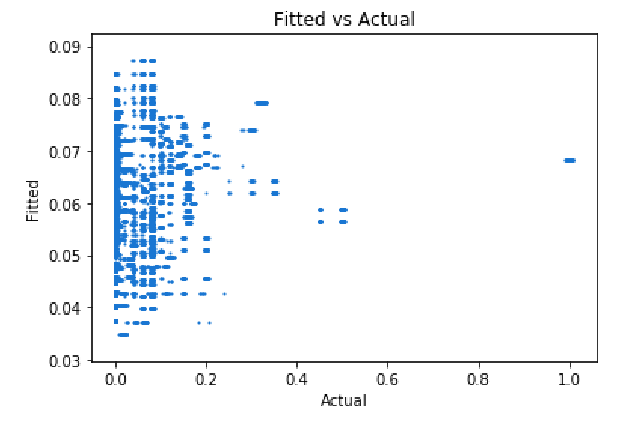
Does the performance improve?

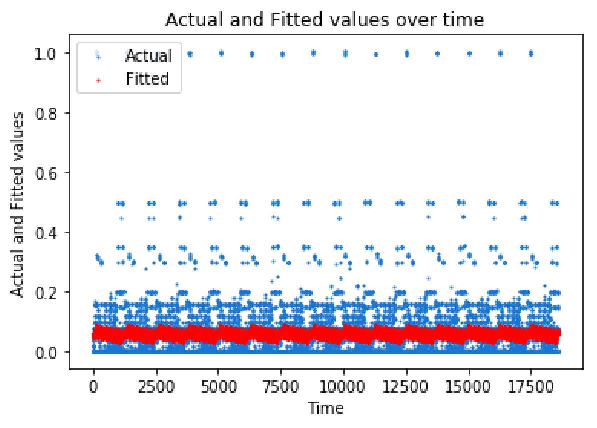
We then performed 10 fold cross validation on the transformed dataset which included only the top three features and calculated the average Training and Testing RMSE obtained over the 10 folds. The RMSE values are as follows-

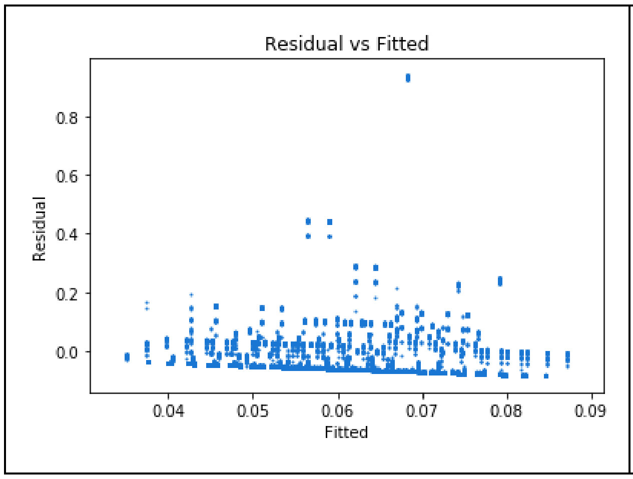
Average Training RMSE	Average Test RMSE
0.103588792478	0.103611218666

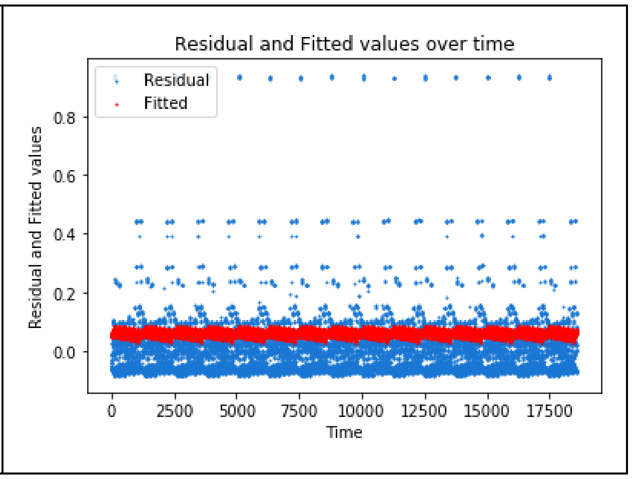
From the table we observe a slight increase in the Training RMSE as compared to the earlier sections, however the Test RMSE does have a slight decrease in value.

We then fitted and tested the regressor on the entire dataset with the top features given by f_regression and plotted the four plots as explained in the earlier sections.









From the plots we observe no significant improvement in the performance from the regressor, as is evident from the RMSE values.

Metric : Mutual Information Regression

We then used the mutual_info_regression metric in order to again find the top 3 features in the dataset. We obtained the following mutual_info_regression metric scores for each of the five candidate features -

Week #	Day of Week	Backup Start Time - Hour of Day	Workflow ID	File Name
0.00313339	0.2271637	0.23895027	0.27668999	0.43091928

As seen from the scores, the top three features from mutual_info_regession scores are 'Backup Start Time - Hour of Day', 'Workflow ID' and 'File Name'. Again, we see that 'Week #' has the least score using mutual_info_regression as well.

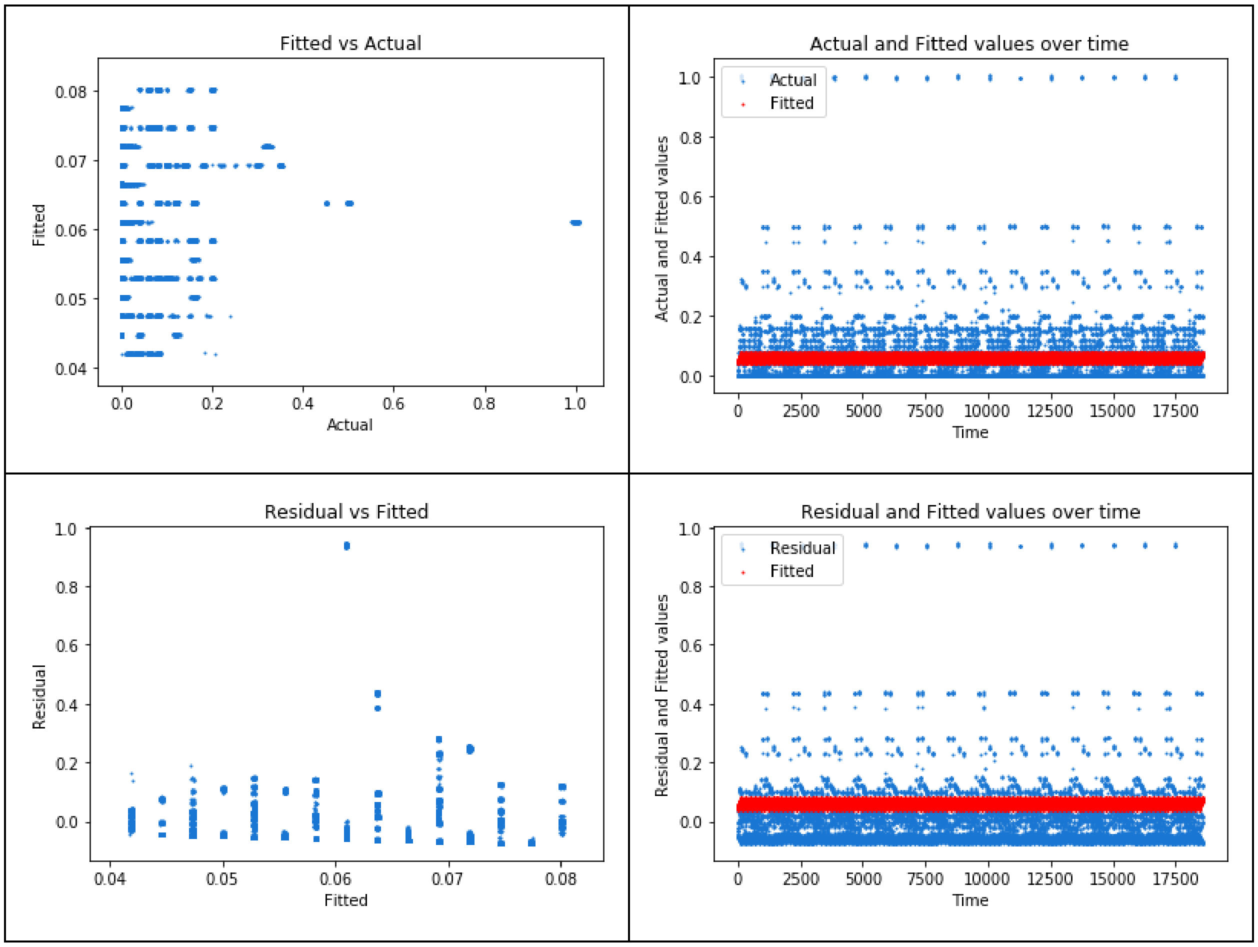
Does the performance improve?

We then performed 10 fold cross validation on the transformed dataset which included only the top three features mentioned above and calculated the average Training and Testing RMSE obtained over the 10 folds. The RMSE values are as follows-

Average Training RMSE	Average Test RMSE
0.103697228859	0.103720626763

Unfortunately, we observe slight increases in both Training and Testing RMSE as compared to the earlier sections.

We then fitted and tested the regressor on the entire dataset with the top features given by mutual_info_regression and plotted the four plots as explained in the earlier sections.



From the plots, we do not observe any sort of significant improvement in the performance of the regressor after considering only the top three features given by the mutual_info_regression score.

Part iv)

In this part, we used One Hot Encoding on the categorical variables in addition to scaler encoding. For each categorical variable that takes one of M values we encoded it as an M dimensional vector, where only one entry is 1 and the rest are 0's. Thus for the Day of the Week, Monday is encoded as [1, 0, 0, 0, 0, 0, 0] and Sunday as [0, 0, 0, 0, 0, 0, 1].

We considered 32 different combinations where in each combination, a subset of the features were One Hot Encoded while the rest were scaler encoded.

While performing the analysis for this part, we found that the Test RMSE in some cases shoots up in the range of e+10 to e+12 exponential. This was found to be true in cases where the feature 'Week #' was One Hot Encoded. But on further inspection we found that this was not the case when the parameter 'shuffle' in the Kfold() method was set to 'True'. Only when the parameter 'shuffle' was set to False, i.e. the dataset was not shuffled before 10 fold cross-validation, did we observe such high values in the Test RMSE. We present our findings for both the cases below -

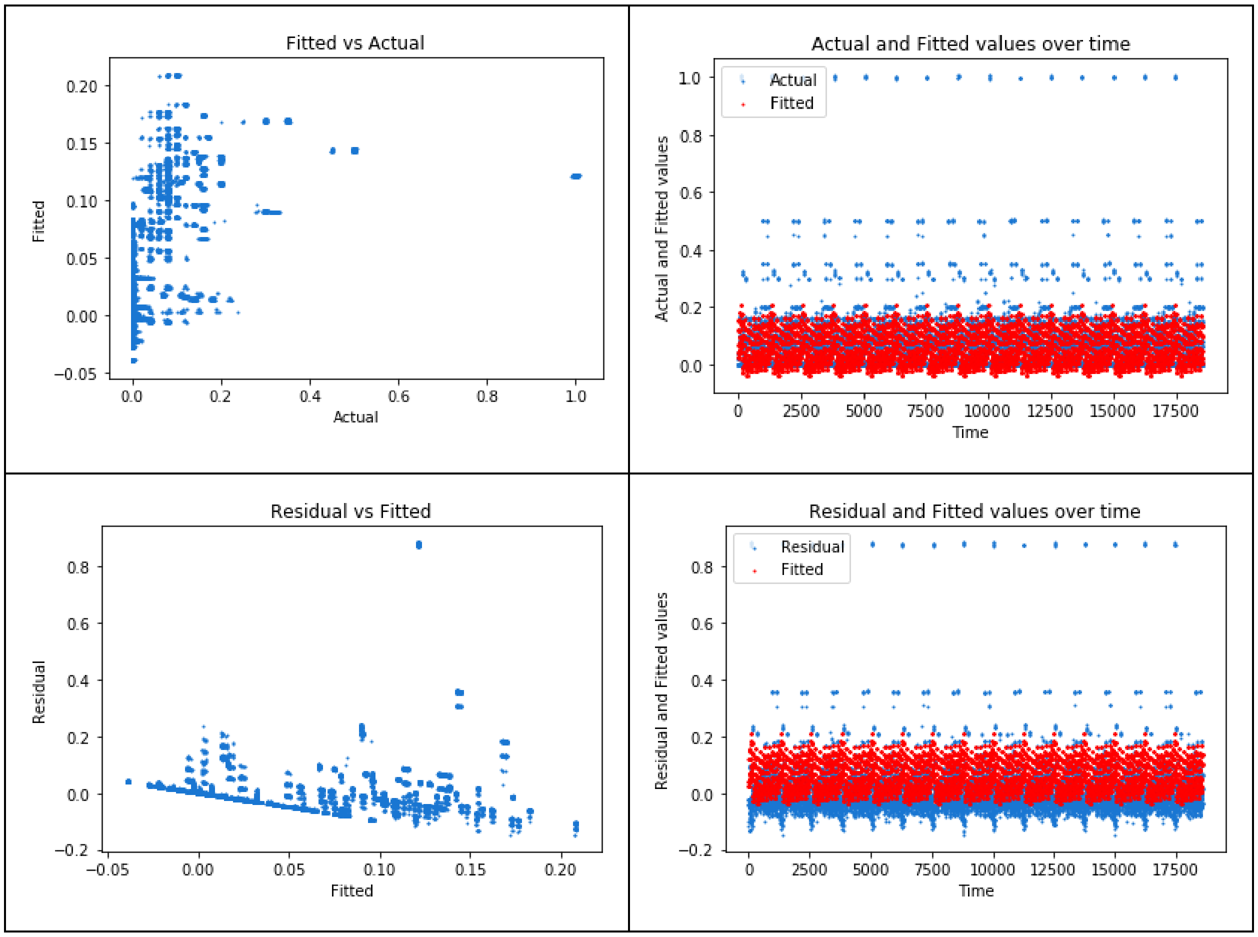
With shuffle = False

In this step, we kept shuffle = False in our KFold() cross-validation function and calculated the training and testing RMSE values averaged over all ten folds for a particular combination of One hot encoding and scaler encoding. We found the following results with the least Test RMSE-

One Hot Encoded Features	Day of Week, Backup Start Time - Hour of Day, File Name
Scaler Encoded Features	Week #, Workflow ID
Training RMSE	0.0883380479166
Testing RMSE	0.0885064915829

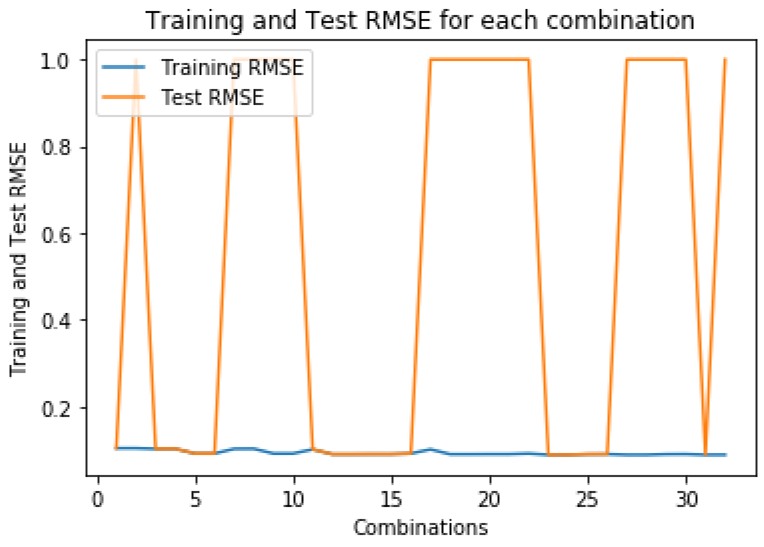
The Training and Testing RMSE obtained is lower than all the earlier sections. This shows that One Hot Encoding certain categorical features definitely helps improve performance of linear regression.

We then encoded the entire dataset using the best combination mentioned in the table above and fitted and tested the regressor on it. We plot four graphs depicting the performance of the regressor as mentioned in the sections above.



The graphs prove that there is a definite improvement in the performance of the regressor.

We also plotted a graph of the Training and Testing RMSE obtained in all of the 32 combinations. The following graph contains the Training and Testing RMSE values plotted on the y-axis with the different combination numbers on the x-axis.



As is evident from the graph, in certain combinations, the Test RMSE is very high. In order to plot the graph effectively, we have capped all values greater than 1 to 1. But as our code output shows, all those values are in the range of e+10 to e+12. All those specific combinations had the feature 'Week #' One Hot Encoded.

With shuffle = True

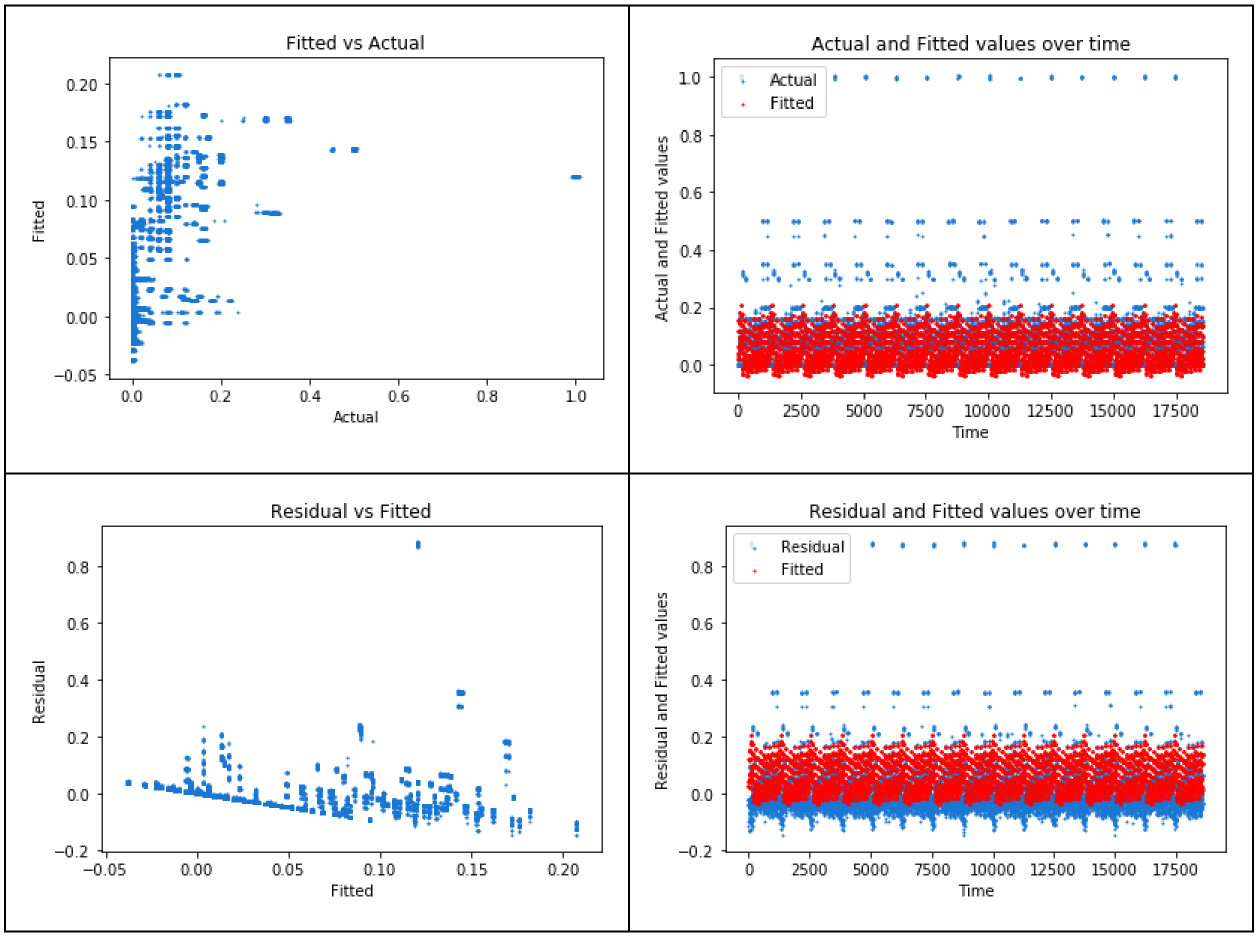
Which combinations achieve best performance?

Then, we kept shuffle = True in our KFold() cross-validation function and calculated the training and testing RMSE values averaged over all ten folds for a particular combination of One hot encoding and scaler encoding. We found the following results with the least Test RMSE-

One Hot Encoded Features	Day of Week, Backup Start Time - Hour of Day, Workflow ID
Scaler Encoded Features	Week #, File Name
Training RMSE	0.0883427584382
Testing RMSE	0.0884431647976

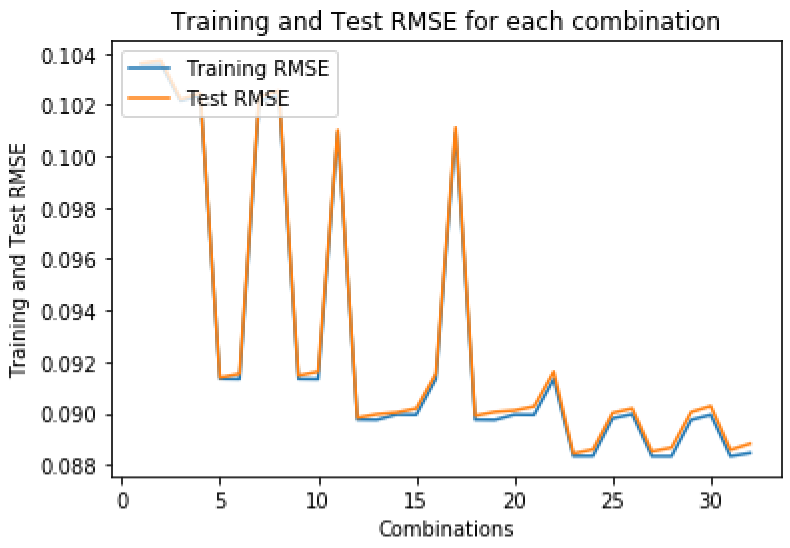
The only difference in this case was that instead of File Name, the feature Workflow ID was one hot encoded. This gave an even better result than the earlier section with the least Training and Testing RMSE obtained.

We then encoded the entire dataset using the best combination mentioned in the table above and fitted and tested the regressor on it. We plot four graphs depicting the performance of the regressor as mentioned in the sections above.



The graphs obtained are not that much different than the ones in the last section, but from the RMSE values we know that the performance was improved.

We then plotted the same graph of the Training and Testing RMSE obtained in all of the 32 combinations. The following graph contains the Training and Testing RMSE values plotted on the y-axis with the different combination numbers on the x-axis.



From the graph you can see that there are no RMSE values greater than 0.104. Shuffling the data before cross-validation helped in reducing the high Test RMSE values obtained in certain combinations. This may be the case since the data points were ordered by 'Week #' and during cross-validation certain 'Week #' values must have not been included in the testing set, leading to unusually high Test RMSE.

Part v)

In this part we used different types of regularization methods to prevent ill-conditioning and overfitting. We used three types of regularization, namely, Ridge Regularization, Lasso Regularization and Elastic Net Regularization.

Reason for high Test RMSE

Looking at some of the coefficients in the past sections, we found that the coefficients tended to be in the ranges of e+11 in either direction of zero, which can lead to overfitting. Using these large values, the model is trying to exactly fit the data, which leads to lower Training RMSE but extremely high Testing RMSE.

Regularization helps to mitigate this issue by adding a penalty for coefficients that are too large. In each of the three types of regularizations, we found the best hyperparameter values for each combination of One hot and scaler encoding. We used GridSearchCV() in order to find the parameter with the least RMSE score.

Ridge Regression

We first used Ridge() from sklearn.linear_model to try and minimize the error caused by overfitting. It uses a loss function that is the linear least squares function and regularization is given by the l2-norm.

We used Ridge regression to find the best combination and the best value of parameter 'alpha' which corresponds to the regularization strength. We chose alpha values such as [0.0001,0.001,0.01,0.1,1,10,100,1000,10000] and obtained the following results -

One Hot Encoded Features	Day of Week, Backup Start Time - Hour of Day, Workflow ID
Scaler Encoded Features	Week #, File Name
Alpha Value	10
Testing RMSE	0.0885040092843
Coefficient Values	3.91627621e-02, -1.28132184e-02, -2.01924610e-02 ,-5.22906083e-03, -5.68395769e-03, 3.26492675e-03, 1.49100903e-03, -2.01413694e-02, -2.09906028e-02, 7.76661954e-03, 3.33486570e-02, -1.98564497e-03, 2.00234054e-03, 3.89679066e-02, -1.36046079e-02, -4.00945986e-02 -5.71782631e-02, 7.19095630e-02, 1.12276439e-05, 7.06996262e-05

As you can see, the coefficient values are much smaller than obtained before and RMSE score with regularization is close to the RMSE score without regularization. But in the cases where we observed really high RMSE values, using regularization would definitely help.

The alpha values specify the regularization strength, where larger values imply stronger regularization. In some of the combinations, a really high value of alpha such as 10000 gave the least RMSE whereas in other cases an alpha of 1 was the best.

Lasso Regression

We then used Lasso() from sklearn.linear_model for regularization. It uses l1 prior for regularization. Similar to ridge, in this case also we chose alpha values from [0.0001,0.001,0.01,0.1,1,10,100,1000,10000] and obtained the following results -

One Hot Encoded Features	Day of Week, Backup Start Time - Hour of Day, Workflow ID, Week #, File Name
Scaler Encoded Features	-
Alpha Value	0.0001
Testing RMSE	0.0885082624122
Coefficient Values	-0.00000000e+00 -0.00000000e+00 0.00000000e+00 -0.00000000e+00 0.00000000e+00 -0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 -0.00000000e+00 0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 4.37513766e-02 -6.96907942e-03 -1.44485670e-02 -0.00000000e+00 -0.00000000e+00 7.71880392e-03 5.93830959e-03 -2.10345963e-02 -2.18506978e-02 5.78177598e-03 3.14450311e-02 -2.79293029e-03 2.96261921e-06 5.17787530e-02 -0.00000000e+00 -2.56788220e-02 -4.23859230e-02 8.64849397e-02 -0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 -0.00000000e+00 0.00000000e+00 -0.00000000e+00 -0.00000000e+00 0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00 -0.00000000e+00 0.00000000e+00 0.00000000e+00

Lasso gave the best performance when all the features were One Hot encoded. The testing RMSE obtained was similar to the value obtained by ridge regression. In lasso, a lot of the coefficient values were set to zero. Ridge regression shrinks all regression coefficients towards zero while the lasso tends to give a set of zero regression coefficients and leads to a sparse solution. On increasing the value of alpha from 0.0001, the coefficient matrix became more and more sparse until all the coefficient values were zero.

Elastic Net Regression

We then used ElasticNet() from sklearn.linear_model for regularization. It uses both l1 and l2 norms for regularization. We optimized over two parameters, alpha and l1_ratio. Where l1_ratio is the elastic net mixing parameter, i.e. an l1_ratio = 0 corresponds to l2 norm and l1_ratio = 1 corresponds to l1 norm. We chose l1_ratio values from [.1, .5, .7, .9, .95, .99, 1] and alpha values from [0.0001,0.001,0.01,0.1,1,10,100,1000,10000] and obtained the following results-

One Hot Encoded Features	Day of Week, Backup Start Time - Hour of Day, Workflow ID
Scaler Encoded Features	Week #, File Name
Alpha Value	0.0001
L1 Ratio	0.1
Testing RMSE	0.0885041347576
Coefficient Values	4.44408782e-02 -7.55621693e-03 -1.49633759e-02 -0.00000000e+00 -4.10950189e-04 8.42870975e-03 6.64932445e-03 -2.07574009e-02 -2.16077887e-02 7.10663053e-03 3.27563959e-02 -2.54650999e-03 1.32681863e-03 5.24174310e-02 -0.00000000e+00 -2.63462105e-02 -4.32777307e-02 8.61851759e-02 1.06818909e-05 3.91963985e-05

The best result was obtained by one hot encoding the same features as in ridge regression. It gave the least test RMSE when l1 ratio was only 0.1, i.e more weightage was given to l2 norm.

Keeping l1_ratio = 0.5 as constant and changing the alpha values gave results similar to Lasso with increasing number of coefficients being set to zero until every coefficient was zero. Keeping alpha fixed at 0.0001 and changing l1_ratio values led to the coefficients being scaled down closer and closer to zero.

Comparing coefficient values

The following table shows all the coefficients for the best combination obtained in the unregularized section with shuffle = True, i.e. with Day of Week, Backup Start Time - Hour of Day, Workflow ID as the One Hot Encoded features. We found the best parameters for each regularized regression method by optimizing over alpha and l1_ratio values which gave the least RMSE.

Unregularized Best Model	1.01577410e+11 1.01577410e+11 1.01577410e+11 1.01577410e+11 1.01577410e+11 1.01577410e+11 1.01577410e+11 -1.98613126e+09 -1.98613126e+09 -1.98613126e+09 -1.98613126e+09 -1.98613126e+09 -1.98613126e+09 1.27733569e+09 1.27733569e+09 1.27733569e+09 1.27733569e+09 1.27733569e+09 9.67979431e-05 8.05854797e-05
Ridge Regularized Model	3.91627621e-02 -1.28132184e-02 -2.01924610e-02 -5.22906083e-03

Alpha = 10	-5.68395769e-03 3.26492675e-03 1.49100903e-03 -2.01413694e-02 -2.09906028e-02 7.76661954e-03 3.33486570e-02 -1.98564497e-03 2.00234054e-03 3.89679066e-02 -1.36046079e-02 -4.00945986e-02 -5.71782631e-02 7.19095630e-02 1.12276439e-05 7.06996262e-05
Lasso Regularized Model Alpha = 0.0001	4.37514542e-02 -6.96874077e-03 -1.44485415e-02 -0.00000000e+00 -0.00000000e+00 7.71883647e-03 5.93834862e-03 -2.10326338e-02 -2.18486329e-02 5.78376088e-03 3.14470120e-02 -2.79076352e-03 4.90377695e-06 5.11006421e-02 -0.00000000e+00 -2.50092883e-02 -4.10421342e-02 8.85036715e-02 5.87970677e-06 -1.12785699e-04

Elastic Net Regularized Model
Alpha = 0.0001
L1_ratio = 0.1
	4.44408782e-02 -7.55621693e-03 -1.49633759e-02 -0.00000000e+00 -4.10950189e-04 8.42870975e-03 6.64932445e-03 -2.07574009e-02 -2.16077887e-02 7.10663053e-03 3.27563959e-02 -2.54650999e-03 1.32681863e-03 5.24174310e-02 -0.00000000e+00 -2.63462105e-02 -4.32777307e-02 8.61851759e-02 1.06818909e-05 3.91963985e-05

Comparing the coefficients obtained from these regularized models with the unregularized models we can see that, these coefficients are much closer to zero than the ones obtained earlier. Regularization helps in reducing high values of the coefficients by scaling them closer to zero and penalizing higher values. From the results obtained earlier, we can see that this helps reduce RMSE and overall improves performance.

2.b Random Forest regression model

The random forest model can handle categorical variables however for the sake of consistency we will be using the scaler encoded features Week #, Day of Week, Backup start time, Workflow ID and File Name to predict the Backup size. The Week # and Backup start time are already numeric features taking values from 1 - 15 and 1 - 24 respectively.

Day of Week, Workflow ID and File Name are encoded to take the values 1 - 7 corresponding to Monday - Sunday, 0 - 4 corresponding to each workflow and 0 - 29 corresponding to each filename.

Parameters of the random forest regressor -

1. n_estimators: The number of trees in the forest.
2. max_features: The number of features to consider when looking for the best split
3. max_depth: The maximum depth of the tree.
4. bootstrap: Whether bootstrap samples are used when building trees.
5. oob_score: Whether to use out-of-bag samples to estimate the R^2 on unseen data.

Attributes of the random forest regressor we consider -

1. oob_score_: Value of the OOB Score (max = 1, higher the better)
2. feature_importances_: The feature importances for each feature in the data (the higher, the more important the feature).

Random forest is an averaging algorithm based on randomized decision trees. By specifying ‘n_estimators’ we indicate the number of decision trees that we will be using to build the random forest model. When we set ‘bootstrap’ to ‘True’ each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. We know that decision trees are sensitive to the specific data on which they are trained. Bagging or bootstrapping in random forest reduces our concern about the individual trees in the forest overfitting to training data. Random forest changes the algorithm for the way that the sub-trees are learned so that the resulting predictions from all of the subtrees have less correlation. Combining predictions from multiple models in this ensemble gives us a better regression model. Out-of-bag samples are samples of training data points that were left behind when creating the bootstrap sample.

Part i)

In this part we will create a random forest model with the following initial parameters -

Number of trees: 20

Depth of each tree: 4

Bootstrap: True

Maximum number of features: 5

The average training and test RMSE for this initial model calculated by 10 fold cross validation

Average Training RMSE	**Average Test RMSE**
0.0604832481886	0.0607882493747

From this result we can see that the average training and test RMSE values are sufficiently low even for our initial model with no parameter tuning. This is because Random Forest is an ensemble method and we are using Bootstrap samples.

Out of Bag Error

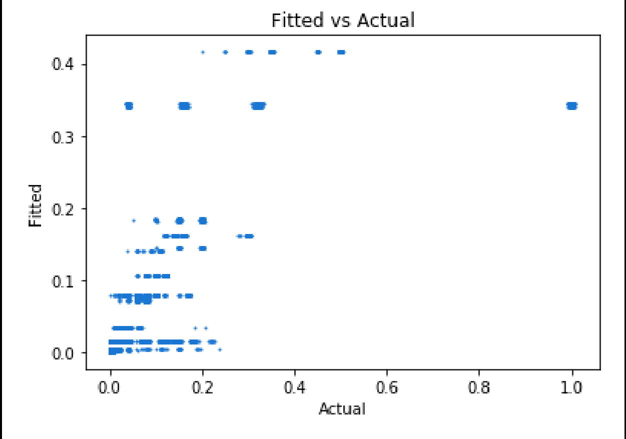
Out-of-bag (OOB) error is a method of measuring the prediction error of random forests. For each bootstrap sample taken from the training data, there will be samples left behind that were not included. These samples are called Out-Of-Bag samples or OOB. The performance of each model on its left out samples when averaged can provide an estimated accuracy of the bagged models. This estimated performance is often called the OOB estimate of performance.

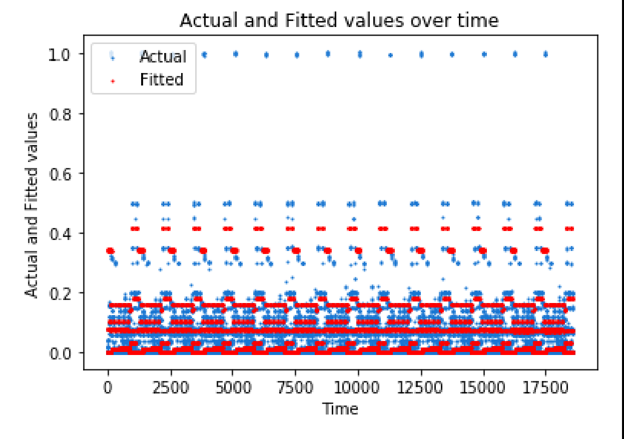
We can use oob_score_ attribute of our random forest model to get the out of bag R2 score. It is also called the coefficient of determination and is the proportion of the variance in the dependent variable that is predictable from the independent variable(s). The best possible oob_score_ is 1.0 and lower values indicate worse performance. The OOB error can then be calculated as 1 - oob_score_.

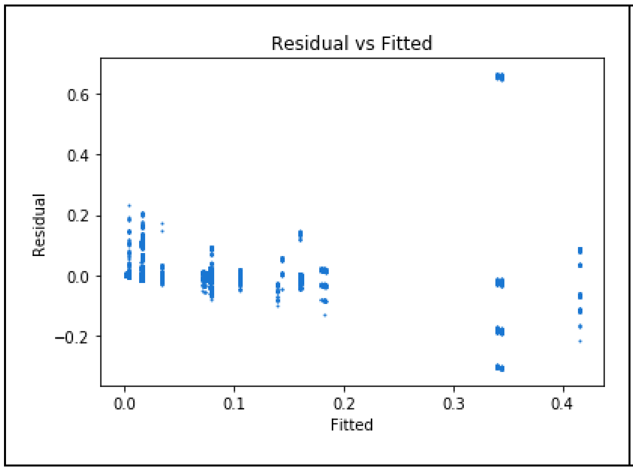
OOB Score	0.656630223607
OOB Error	0.343369776393

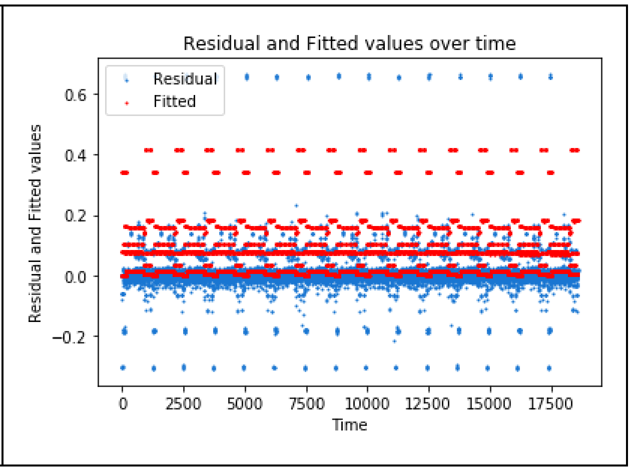
Plots

The actual values are the true labels of our data points and the fitted values are the predicted labels. Residuals are calculated by subtracting the fitted values from the actual values. Thus, residuals closer to 0 indicate a better model fit for the data.







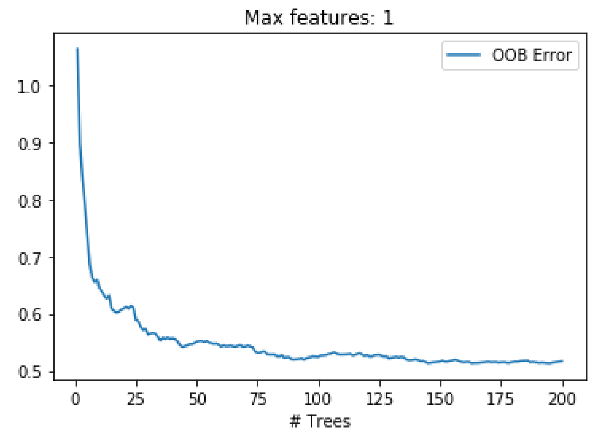


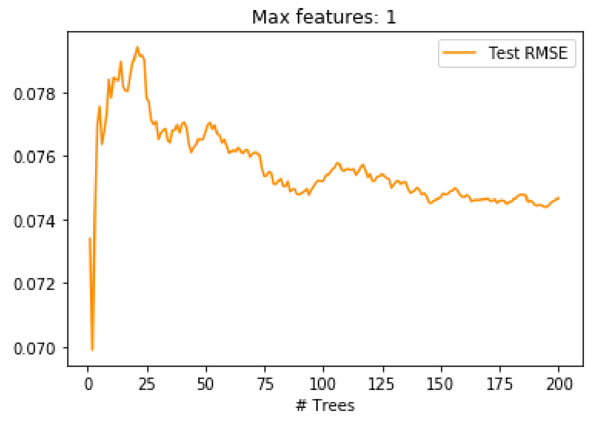
From the plots above we can see that the random forest model does a better job of fitting our non-linear data than the linear regression model in the previous section. In the further sections after parameter tuning, we will get a better visualization of the best random forest model.

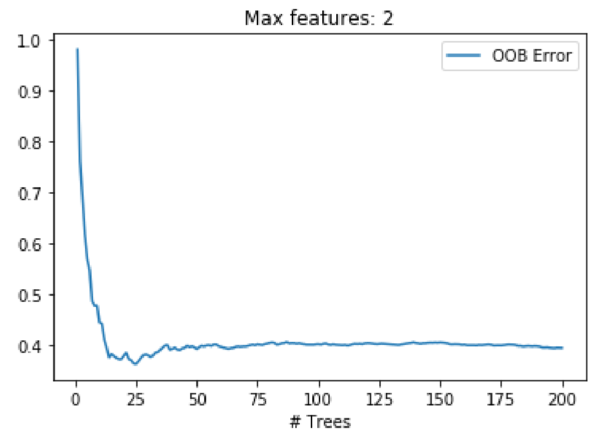
Part ii)

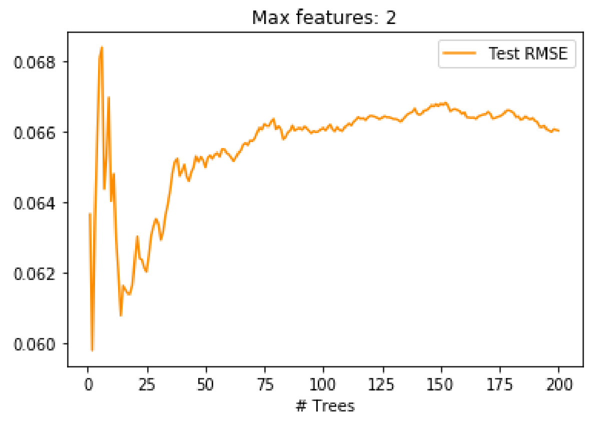
In this part we sweep over number of trees from 1 to 200 and maximum number of features from 1 to 5. For each value of max_features we plot the OOB error against number of trees and average Test RMSE against number of trees.

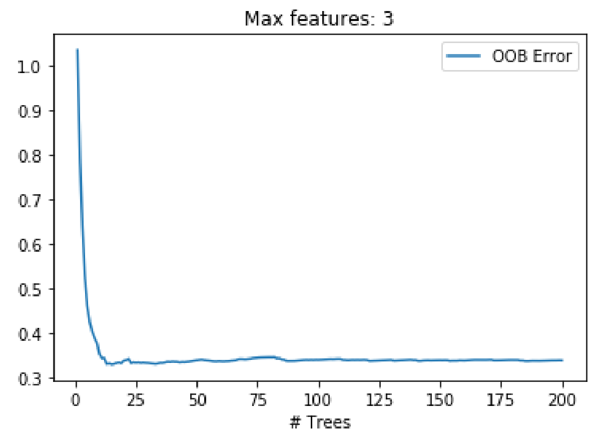
Max Features	**Minimum OOB Error**	**Minimum average Test RMSE**
1	0.512762136272, #195	0.0698928915494, #2
2	0.36287171125, #25	0.0597898399214, #2
3	0.329047038553, #15	0.0581903685889, #13
4	0.343816448266, #156	0.0608522551228, #24
5	0.340599524776, #154	0.0604685995758, #3

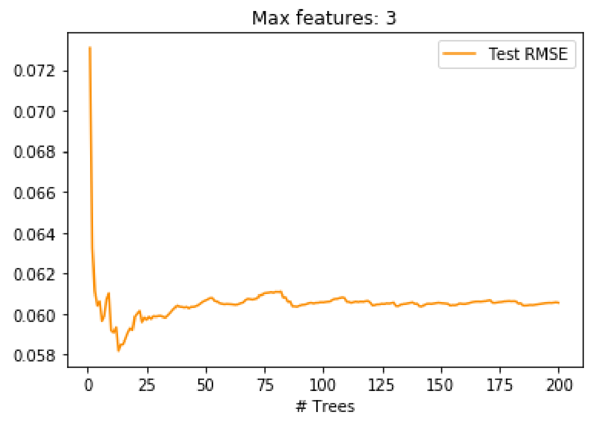

Max features: 1
OOB Error
1.0
0.9
0.8
0.7
0.6
0.5
0
25
50
75
100
125
150
175
200
# Trees

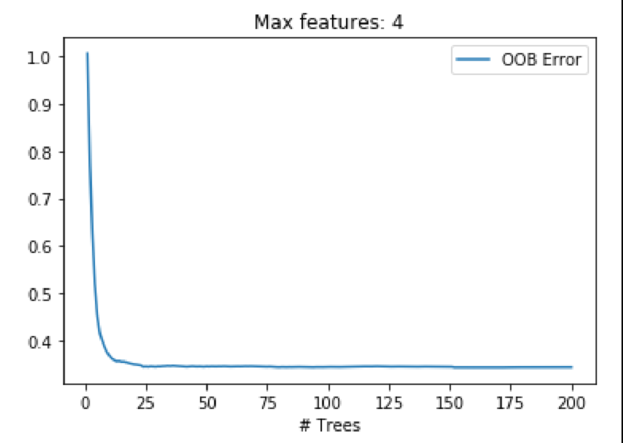

Max features: 1
Test RMSE
0.078
0.076
0.074
0.072
0.070
0
25
50
75
100
125
150
175
200
# Trees

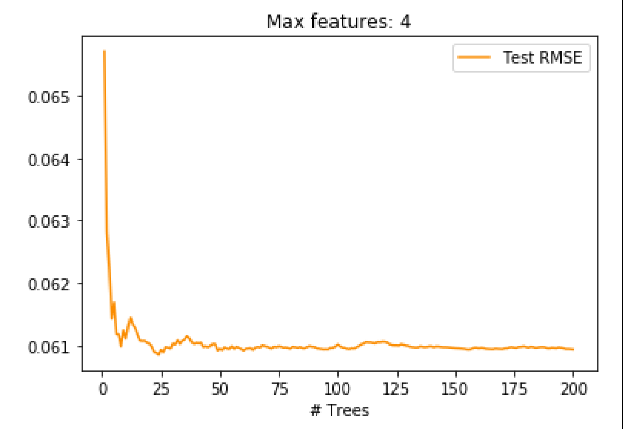

Max features: 2
OOB Error
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
25
50
75
100
125
150
175
200
# Trees

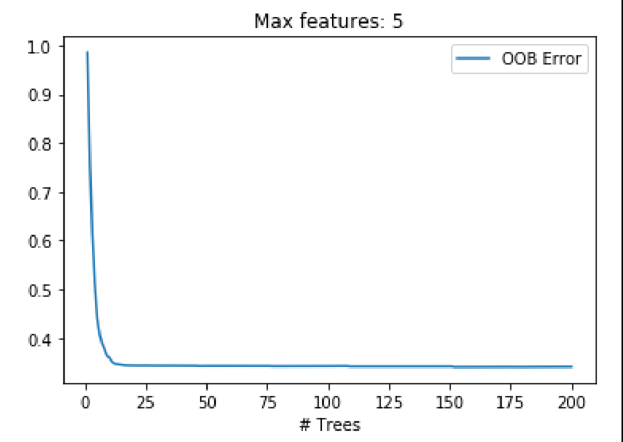

Max features: 2
Test RMSE
0.068
0.066
0.064
0.062
0.060
0
25
50
75
100
125
150
175
200
# Trees

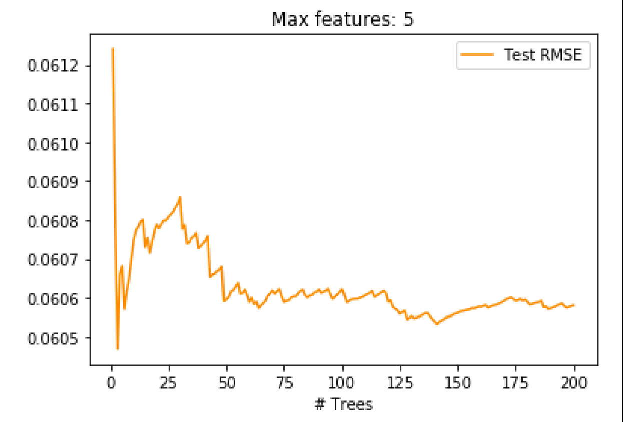

Max features: 3
OOB Error
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
25
50
75
100
125
150
175
200
# Trees


Max features: 3
Test RMSE
0.072
0.070
0.068
0.066
0.064
0.062
0.060
0.058
0
25
50
75
100
125
150
175
200
# Trees









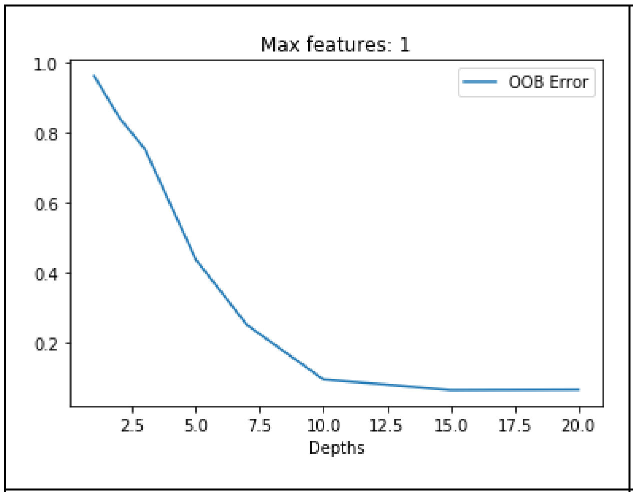
From the above figures we can see that the average Test RMSE has a consistently lowest value when max_features = 4 and it increases for max_features = 5 indicating a possible overfitting to the training data when max_features = 5. The Test RMSE is calculated by 10 fold cross validation. From the graph of the OOB error we can see that the elbow is at # Trees = 13 after which the OOB error flattens out. This gives us the best value of number of trees, ‘n_estimators’ to be 13 for all future parts. Thus we can see that for our network backup dataset, a random forest model which is an ensemble of 13 decision trees and considers only 4 of the features instead of 5 is sufficient for getting a good accuracy. In the further parts, we will be analyzing the feature importance of the 5 features we have been using for training and will determine which of the 4 features our model uses.

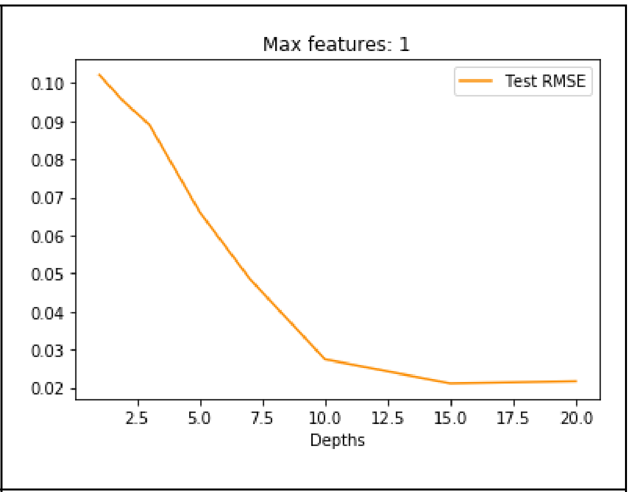
Part iii)

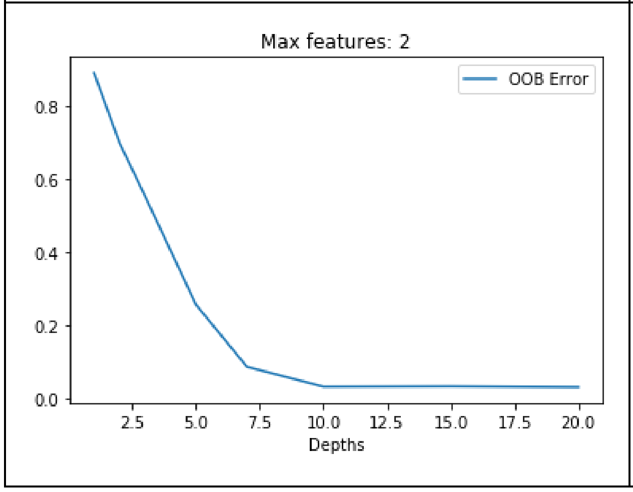
In this part we have experimented on the maximum depth of the tree parameter of the random forest regressor. In order to do this we have kept the value of ‘n_estimators’ constant at 13 as determined from the previous part.

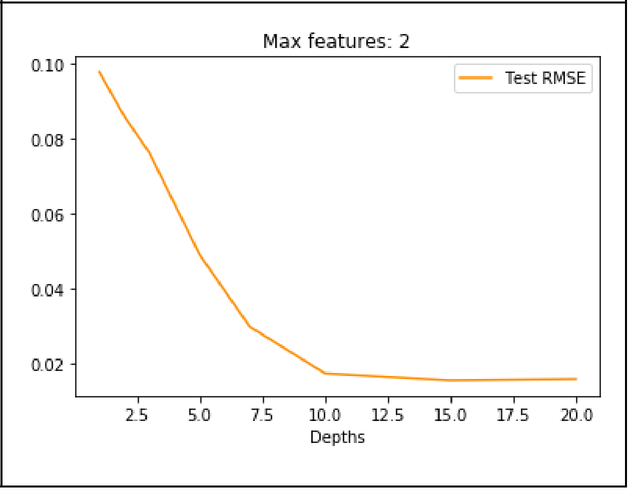
We sweep over the values of 'max_depth' in depths = [1,2,3, 5,7,10,15,20, None] and maximum number of features from 1 to 5. For each value of max_features we plot the OOB error against depths and average Test RMSE against depths.

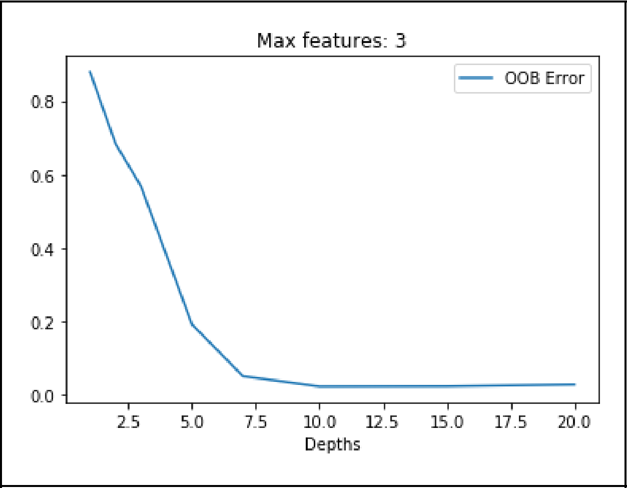
Max Features	Minimum OOB Error	Minimum average Test RMSE
1	0.0603454204542, #None	0.0205093268258, #None
2	0.0317060281505, #20	0.0155427840928, #15
3	0.0238662143006, #10	0.0135374886247, #10
4	0.018546384693, #10	0.0129456923991, #10
5	0.018428083607, #10	0.0129387656467, #10

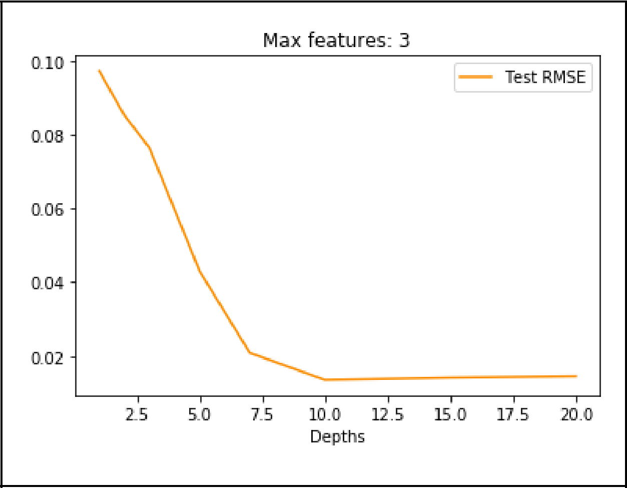


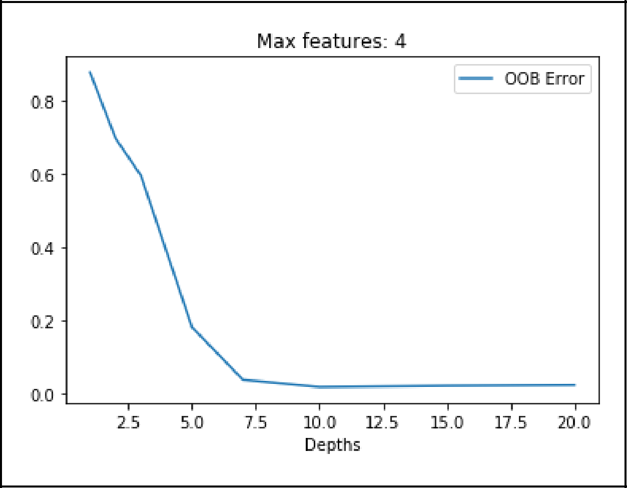


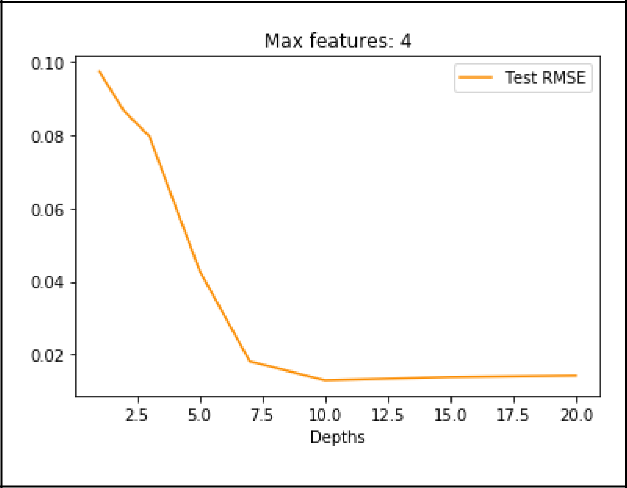


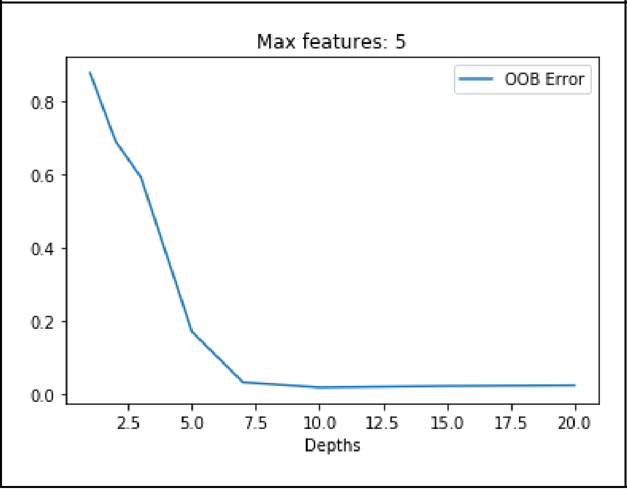


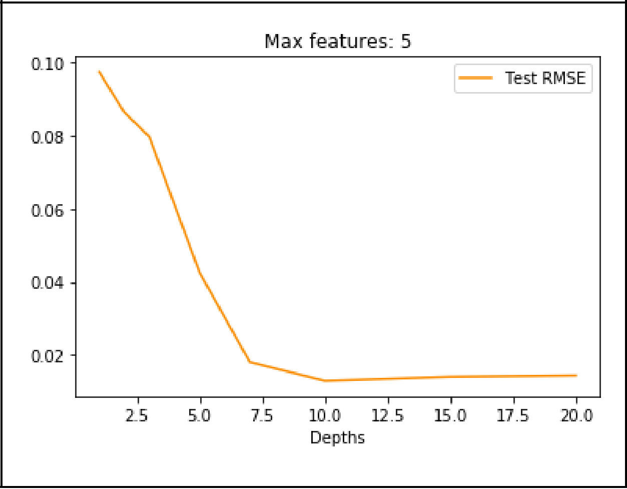












From these graphs we can see that for n_estimators = 13, the average Test RMSE is lowest at max_depth = 10 after which it flattens out. Thus for our best model random forest regressor we will be using max_depth = 10 and n_estimators = 13, max_features (as determined in Part ii). From the minimum values of the OOB Error and average test RMSE, we can also see that the results are best for max_depth = 10 and not for max_depth = None when we are considering more than 1 feature for the predictions. Thus by using more features we are able to restrict the depth of our decision trees to 10 in order to get good accuracy.

Part iv)

For this part we are creating the best random forest model as determined from the results obtained for the previous parts. Our best random forest model has the following parameters -

Number of trees: 13

Depth of each tree: 10

Bootstrap: True

Maximum number of features: 4

The average training and test RMSE of the Best model calculated by 10 fold cross validation

Average Training RMSE	**Average Test RMSE**
0.0111352376946	0.0129456923991

Feature Importance

As the Bagged decision trees are constructed, we can calculate how much the error function drops for a variable at each split point. These drops in error can be averaged across all decision trees and output to provide an estimate of the importance of each input variable. The greater the drop when the variable was chosen, the greater the importance. For our regression task 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.

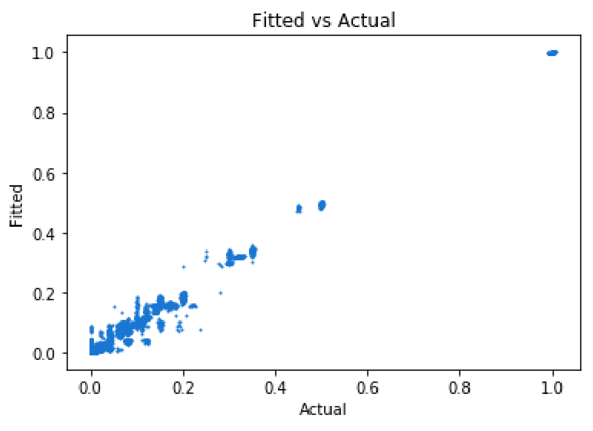
The following table lists out the features along with their feature importance in decreasing order. Thus for our best model, the best feature is 'Backup Start Time - Hour of Day' while the worst is 'Week #' for making a branching decision. Hence we can say that our best model for random forest ignores the 'Week #' feature while constructing decision trees when max_features is set to 4.

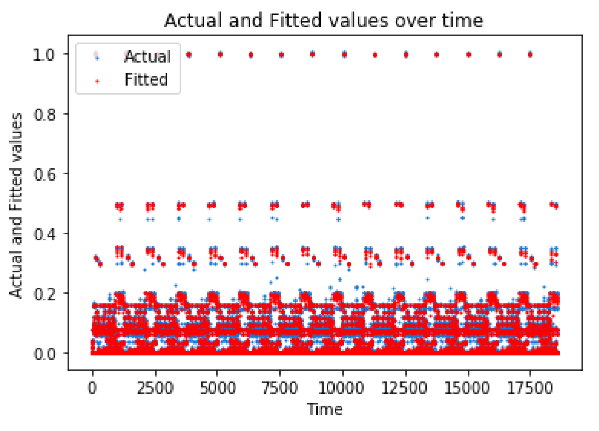
Feature	**Feature Importance**
Backup Start Time - Hour of Day	0.39779999999999999
Day of Week	0.2248

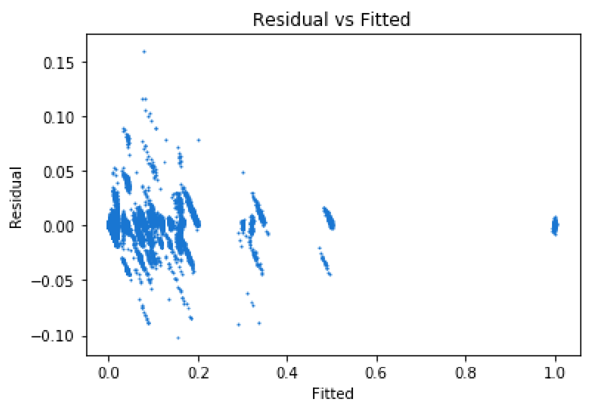
File Name	0.1981
Work-Flow-ID	0.1762
Week #	0.0030999999999999999

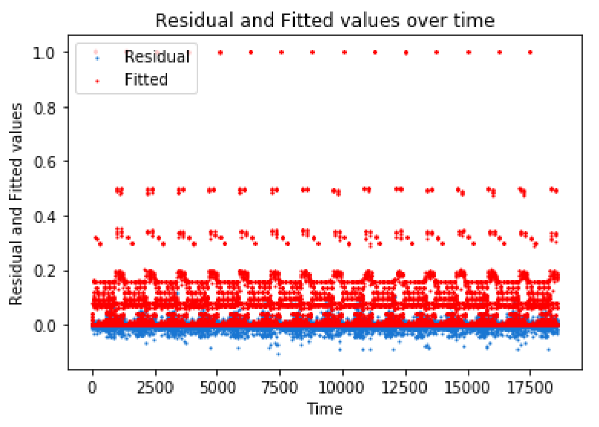
Plots

The actual values are the true labels of our data points and the fitted values are the predicted labels. Residuals are calculated by subtracting the fitted values from the actual values. Thus, residuals closer to 0 indicate a better model fit for the data.









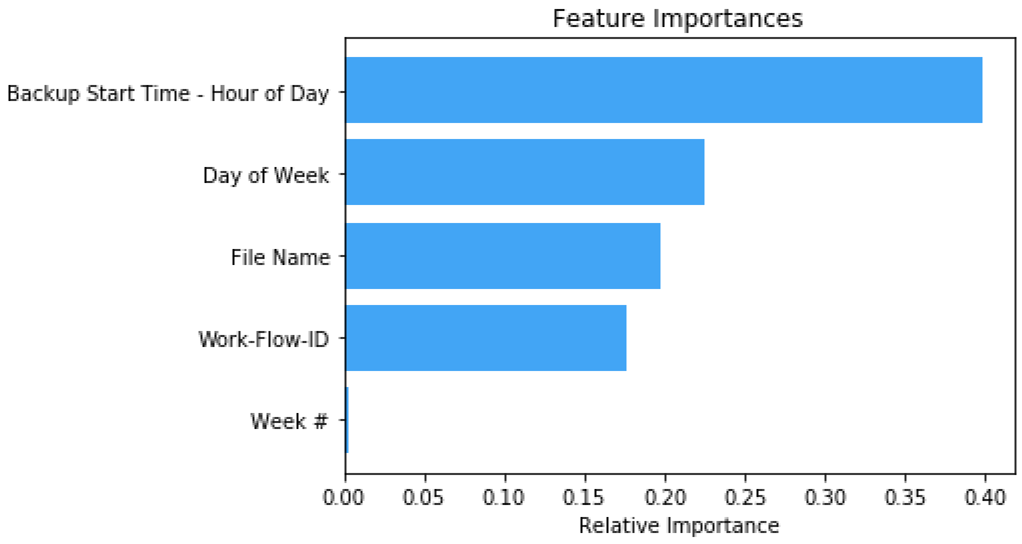
By contrasting these graphs with the ones obtained for the initial model in Part i, we can clearly see that this model fits the non-linear data much better. The plot of actual and fitted values over time has a lot of coinciding points indicating this better fit. The plot of actual vs fitted values gives us points along the diagonal axis as opposed to the scattered points seen in the initial model. Similarly, the residual and fitted values over time plot shows that the residual points are aggregated at 0 indicating that our true labels and predicted labels coincide for most of the data points.

Part v)

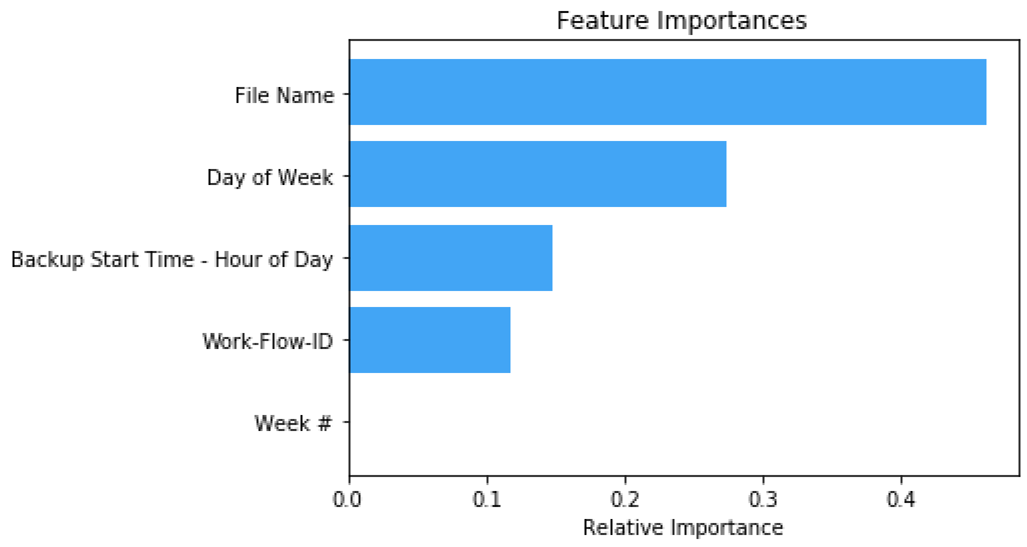
We know that the best model we obtained in Part iv has a max_depth = 10. By restricting the max_depth to 4 in this section, we can see from the below bar charts that the feature importance of the models has changed. This effect is the result of different features being used for making the branching decisions when the depth of the tree is restricted. For visualizing the decision tree of depth 4 from the random forest model that we are using for this section, we have used the GraphViz package in Python.

The bar charts below help us visualize the relative importance of different features for the best model and for the random forest model with depth restricted to 4.

Visualizing feature importance for best model in Part iv -

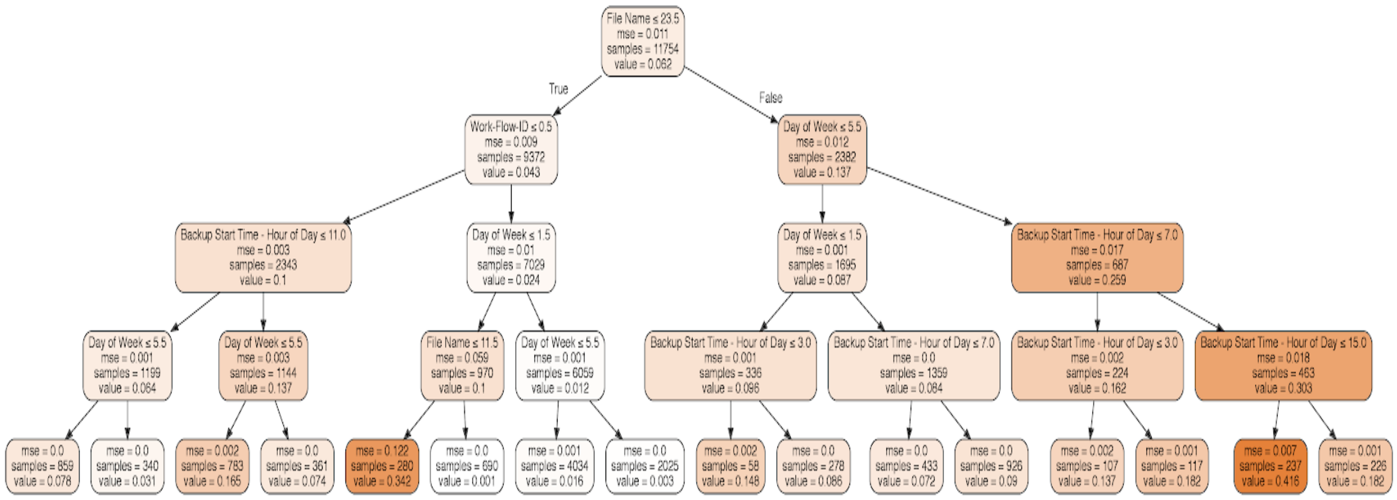


Visualizing feature importance for random forest estimator with depth restricted to 4 -



Visualizing decision tree for random forest estimator with depth restricted to 4 -

The following figure shows us the 4th estimator from the 13 tree random forest model with max_depth = 4 that we constructed. The root of the tree is the feature with maximum feature importance as visualized from the bar chart above. The reason this root does not match with the root of the best model random forest regressor is that in this case we had to restrict the depth to 4 while our best model had a depth of 10. In the process of visualizing the feature importance, we noticed that the maximum feature importance feature for each decision tree in the forest is different however the root of that particular tree is always the feature with the highest relative feature importance. This is because the metric 'feature_importance_' is calculated relative to a specific dataset, i.e. it is calculated for the data specific to that particular decision tree (estimator) in our random forest model.



2.c Neural Network Regressor

Implementation

We are using MLPRegressor from the sklearn package to perform this task. This model optimizes the squared-loss using LBFGS or stochastic gradient descent. MLPRegressor trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting. This implementation works with data represented as dense and sparse numpy arrays of floating point values.

As asked, we have created a model with one layer and the size varying from 1 to 200. For each size of the model, we performed the Kfold cross validation. We computed the MSE(mean squared error) for each fold and using that, we reported the average test and train RMSE using the below equation

$$\text{RMSE}_{train} = \sqrt{\frac{mse_{train,1} + \cdots + mse_{train,10}}{10}}$$

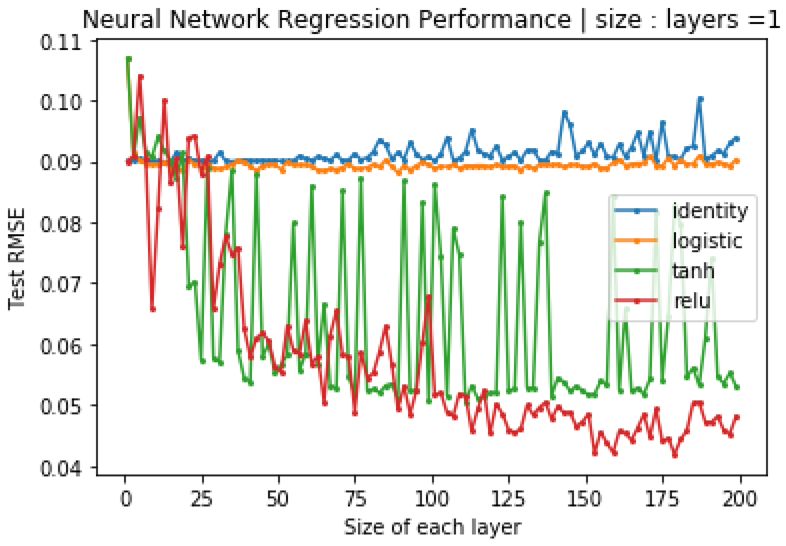
We performed the above analysis for 4 different types of activation algorithms i.e [‘identity’, ‘logistic’, ‘relu’, ‘tanh’]

Analysis:

The following is the minimum avg. Test RMSE observed for a Neural Network with a hidden layer and size varying from 1 to 200 neurons.

Activation Algorithm	*Avg. Test RMSE*
Identity	0.08862
Logistic	0.08687
Tanh	0.05325
Relu	0.04297

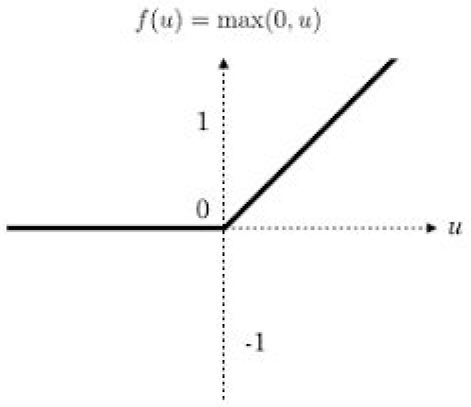
The figure below shows how the average test RMSE varies for changing layer size. The trend for identity and logistic seem to be almost same and are unaffected by the layer size. But both tanh and relu have a decreasing trend with spiky graph.



<u>Best Combination is "relu" with layer size = 1777</u>

We also tried to see how the trend looks for each activation algorithm when the number of hidden layers are increased. As expected, The trends for all the activation algorithms remained same as previous but the important observation we made is that the least RMSE is pretty quickly achieved(80) compared to the above case. The more the number of layers, the faster the RMSE decreases.

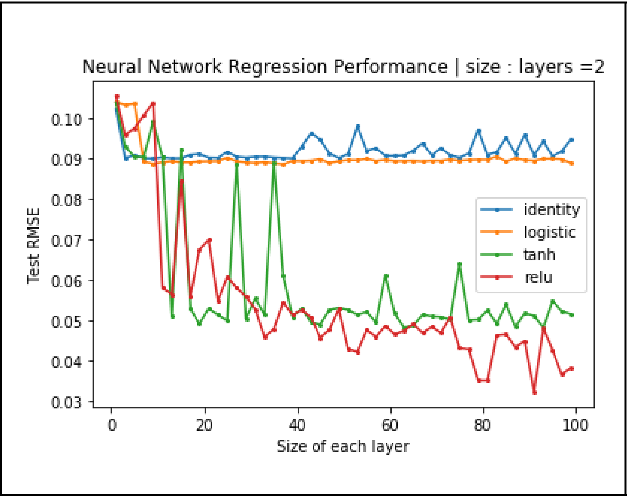
The definition of a ReLU is h=max(0,a) where a=Wx+b.

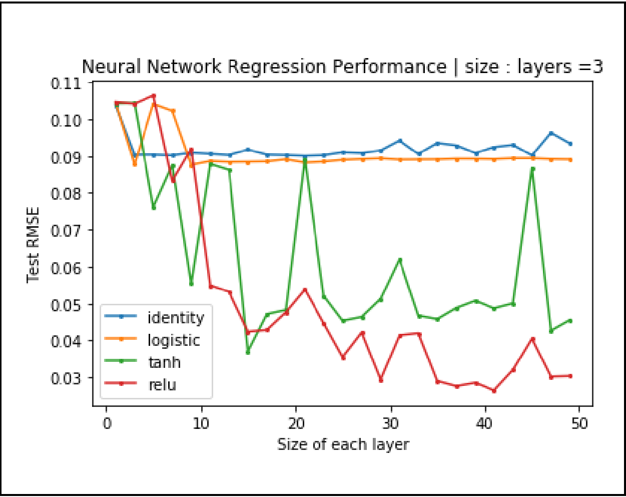


Two major benefits of ReLUs are:

1. **Sparsity** : Sparsity arises when a≤0. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.
2. **Reduced likelihood of vanishing gradient**: This arises when a>0. In this section the gradient has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLUs results in faster learning.

In the below graphs, we used a 2 and 3 layer neural networks to perform the regression task. Even though the trend is same, the neural network with 3 layers achieves the minimum Avg. RMSE much faster(~40) compared to the one with 2 layers (~90).





Also, we wanted to check if neural networks perform well in case of scalar encoding. The RMSEs(~0.08) in this case are much higher compared to the model with one-hot encoding(0.035). We can infer that in this case neural networks doesn't perform well for categorical data.

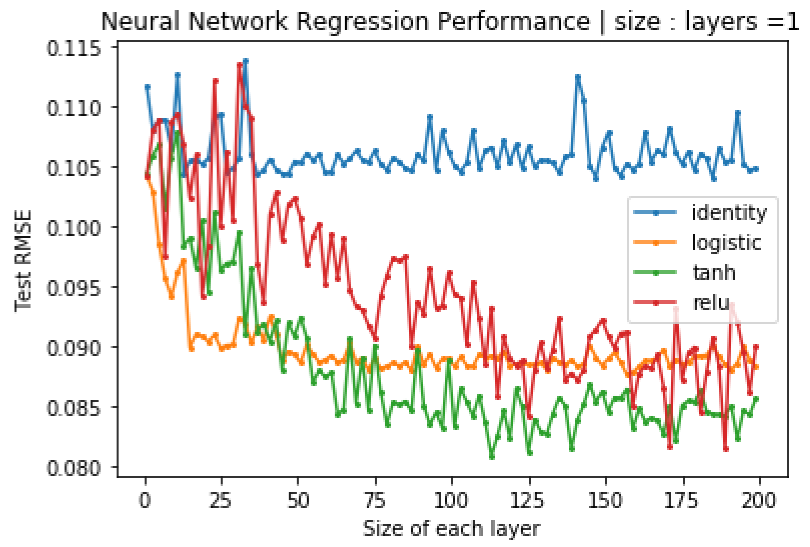


Figure: Neural Network Regressor with Scalar encoding

2.d Backupsize Prediction for Workflows Separately

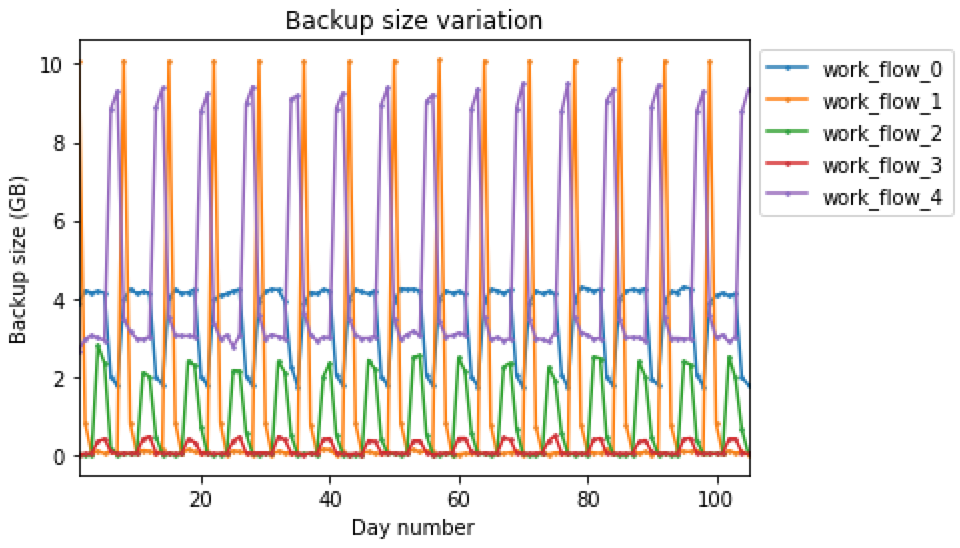
2.d i Linear Regression Model

To perform the task of prediction for workflows separately, we did split the data frame based on the Workflow_ID column values. We maintain the separated data frames in a list and perform the required operations like splitting the features and labels, performing encoding using comprehensive list operations

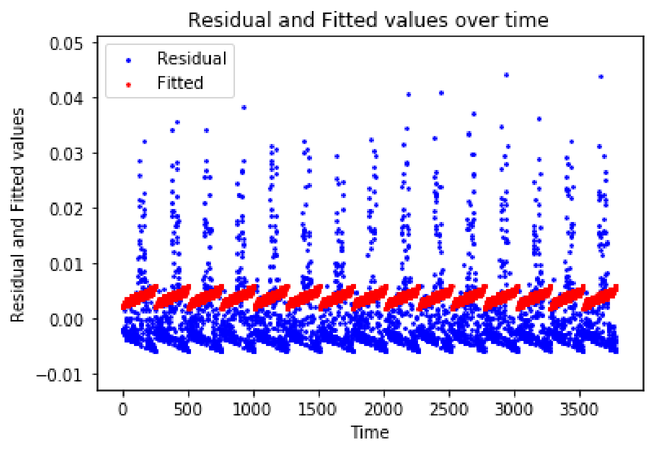
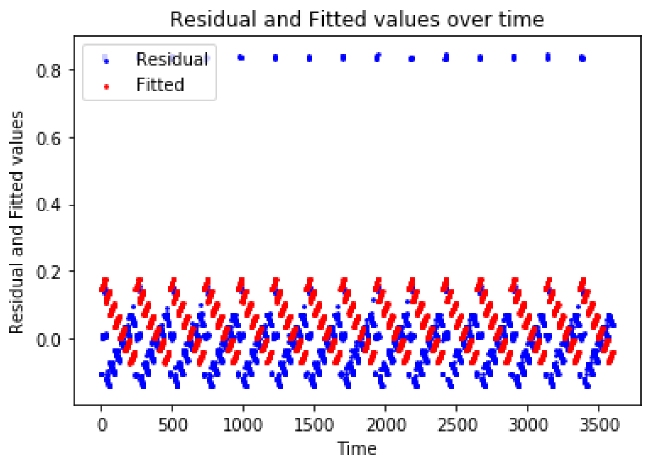
Workflow ID	*Avg. Test RMSE*	*Avg. Train RMSE*
1	0.1489	0.1487
0	0.03587	0.03583
3	0.00725	0.00724
2	0.04297	0.04291
4	0.08601	0.08591

Analysis:

The fit is improved for all the workflows except Workflow 1. The test RMSE values are very different for each of the work flows. We weren't expecting this as the purpose is similar for all the workflows i.e. backing up the data periodically. But we then tried to analyze the reason for this variance. In the figure below from Question 1, where we plot the backup size Vs time for each of the workflows for 105 days, all the workflows are different. One striking difference can be observed from Workflow 1 and Workflow 3. The range (difference between the min and max backup size) is least in case of Workflow 3(.3 ~) and highest in case of Workflow 1 (10 ~).



Since we are using the linear regression model to backup size, it is clearly evident that the fitted linear line will deviate much from the original graph and will also have a higher slope in case of Workflow 1. In case of Workflow 3, the range is much less and the fitted line will also have lesser slope relatively and gives less RMSE. It is clearly evident from the Residual and Fitted values over time. The magnitude of residual values are not greater than 0.05 in case of Workflow 3 and they even go up to 0.8 fro Workflow 1. Hence the RMSE is very high for Workflow1.

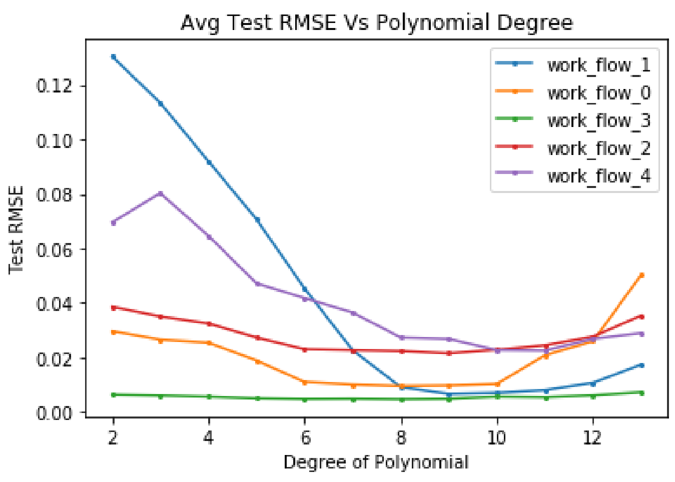
Workflow 3	**Workflow 1**
	

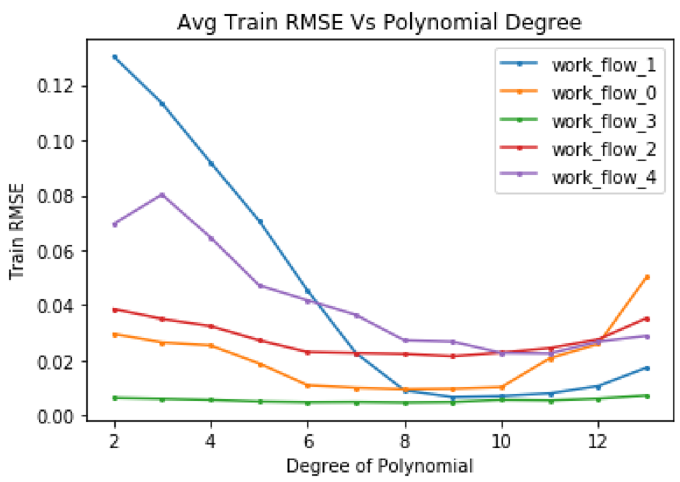
The same argument applies to other workflows also. If we order the rest of the workflows according to the range of the values, Workflow 1 is followed by Workflow 4 and Workflow 0 and Workflow 2 have similar range. The same pattern can be observed in the RMSEs table. It is important to note that range is not a strict measure to reason the RMSEs but in this scenario, given the pattern of the data, it best explains the observed variances in RMSEs.

2.d i Polynomial Curve Fitting

Implementation:

We then performed the prediction task by fitting a more complex regression function to the data. The following graphs show the Avg. Train and Test RMSEs for varying polynomial degree from 2 to 14. The ***analyse_poly_lin_reg_separate*** function takes in the range of polynomial degrees and performs Kfold for each workflow for each polynomial degree. The minimum RMSE and the degree at which the minimum occurs is stored for each workflow. There is also a provision to see the plots of the best fitted model for each workflow.





Analysis:

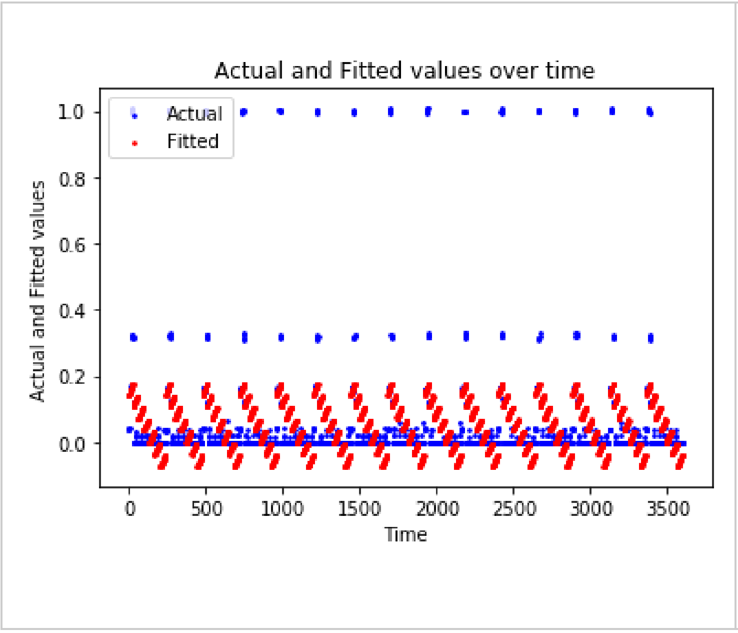
In the above plots, both the training and test RMSEs look similar. After a particular degree, there isn't either much change in the RMSE or the RMSE increases. The trend in different for different workflows. As we already mentioned from the plot of Backup Size Vs Time, Workflow 3 have little variance from min and max (smaller range of the plot). So it is easier to fit such simple graphs at smaller degrees. Hence, the RMSEs for Workflow 3 doesn't change much even though the polynomial degree increases. This can be contrasted with that of Workflow 1 that has much larger range relatively and hence needs a bigger polynomial to fit. So the RMSE plot keeps decreasing rapidly until 9 and slightly increases later on.

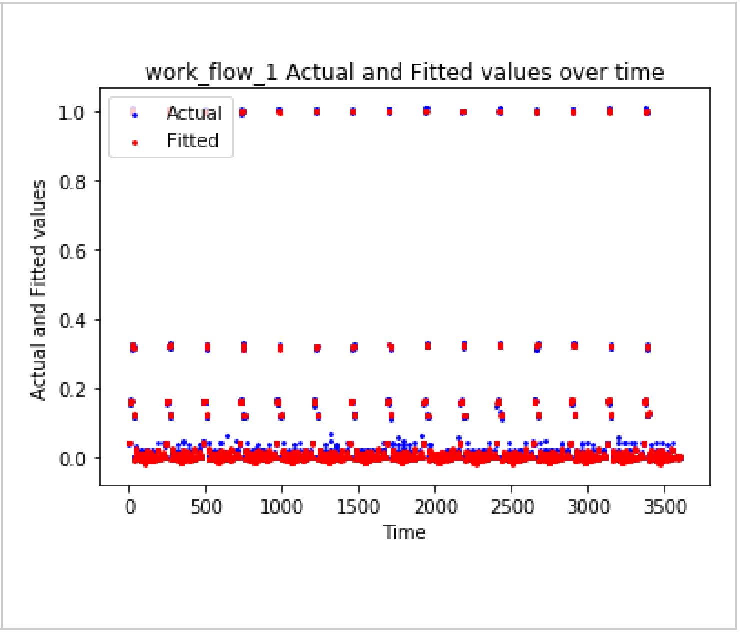
The below is a table showing the min. Average RMSE values and also the polynomial degree at which the minimum occured. The last column of the table is added in order to compare with the results of normal linear regression. There is a drastic increase in the performance in case of polynomial fitting. The results of every workflow show significant decrease in the RMSE values. This is obvious as the workflow data is not at all linear. There is more than a 10-fold improvement in case of Workflow 1 but the others have only 50% decrease. But the end RMSE values are almost similar for all workflows. This is only because Linear Regression isn't a good choice for the dataset given

Workflow ID	*Min. Avg. Test RMSE (Polynomial)*	*Threshold Polynomial Degree*	*Avg. Test RMSE (Linear Regression)*
1	0.00706	9	*0.1489*
0	0.00973	8	*0.03587*
3	0.00487	8	*0.00725*
2	0.02281	9	*0.04297*
4	0.02254	11	*0.08601*

To give an idea of how the predicted values look different for the 2 models, we plotted the actual and fitted values as a scatter plot for both models. It can be clearly seen that the predicted values match the actual values much better in case of the Polynomial fitting supporting our observation of decreased RMSE.

Linear Regression Model	**Polynomial fitting (Degree = 9)**





How does cross-validation improve?

Cross validation evaluation method is better than residuals because residuals do not measure how well the learner will do when predictions are made on completely new data. The advantage of Kfold cross validation method is it doesn't matter how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. The variance of the resulting estimate is reduced as k is increased. The main disadvantage of this method is that the training algorithm has to be run k times, which means it takes k times as much computation to make an evaluation.

2 e. KNN Regression

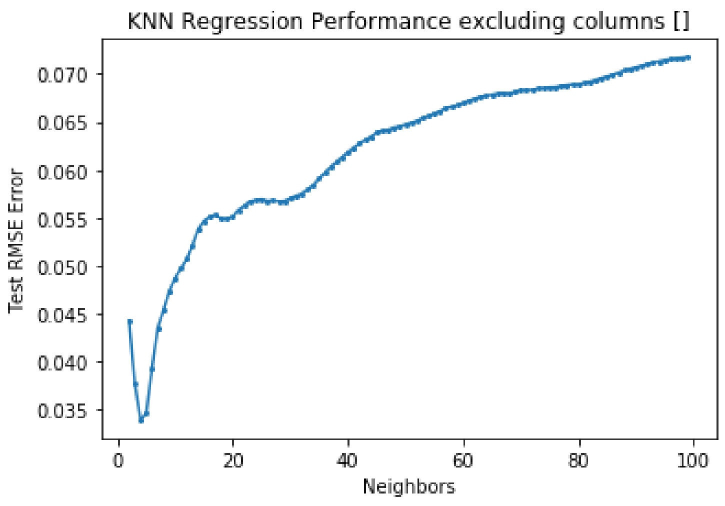
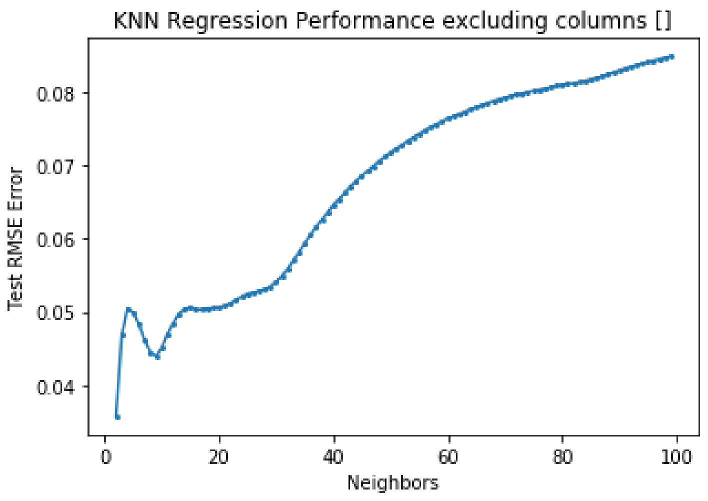
Implementation:

For KNN regression, we are using sklearn.KNeighborsRegressor to model the data and perform prediction. Our analyze_knn() takes the dataset, neighbor_range, remove_cols, encoding as parameters. The neighbor_range is the list of neighbor ranges over which we perform KNeighborsRegression. We also performed some more analysis by dropping few columns given by remove_cols argument. The encoding parameter decides whether we perform scalar or one-hot-encoding of the data set.

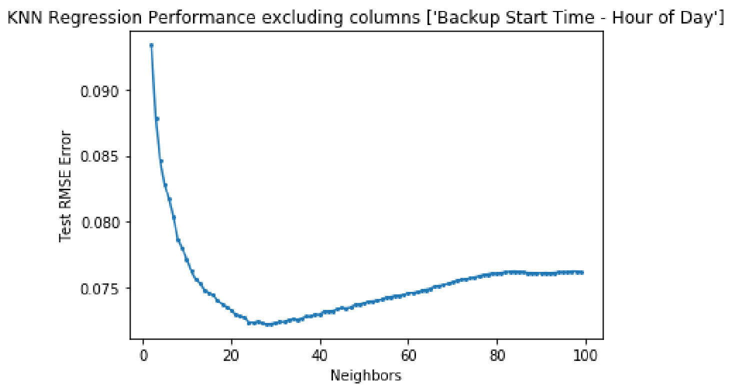
Analysis:

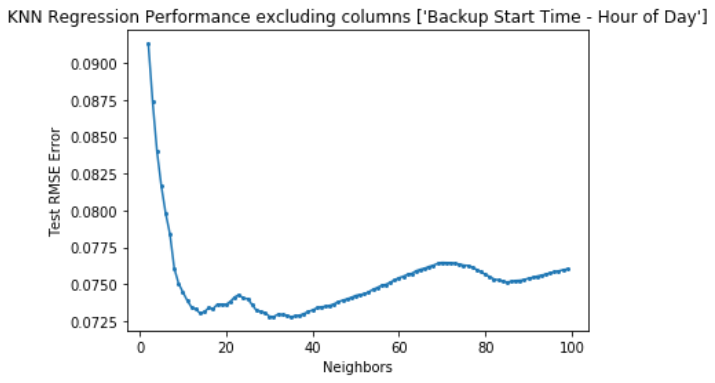
Model	Min. Average Test RMSE
KNN with Scalar Encoding	0.03393 (4 neighbors)
KNN with one-hot-encoding	0.03576 (2 neighbors)

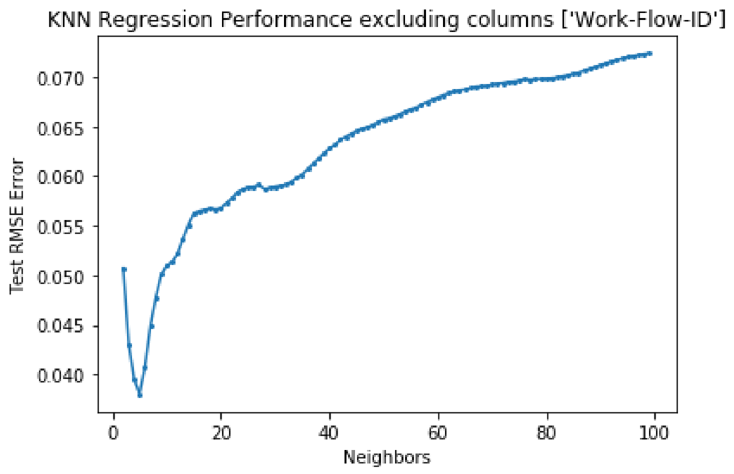
We observed that one-hot-encoding takes much computational resources as the dimension of the data is much higher compared to the scalar encoding

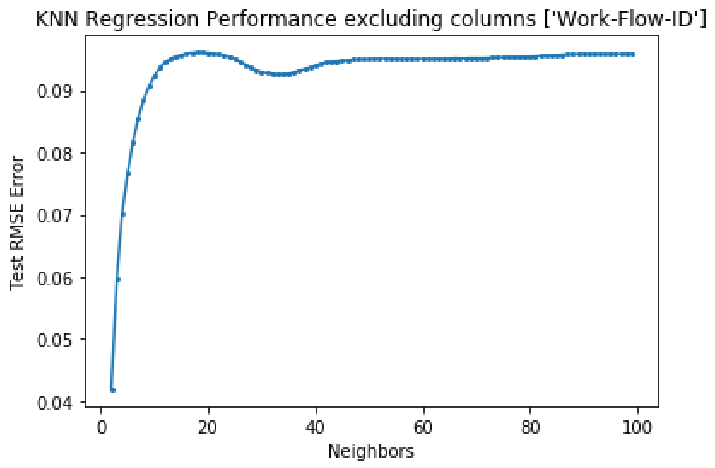
Scalar Encoding	One Hot Encoding
The plots below show the variation of avg. Test RMSE for scalar(left) and one hot encoding(right). There is a similar trend observed after the neighbors increase beyond 10. Since we have different data patterns for different workflows, after a certain number of neighbors, the prediction task uses data points that are farther from the current one that is predicted which might include data points from other workflows too. This leads to a gradual increase in the RMSE values. As for the initial part of the plots, the trend is different because the optimal number of neighbors depends on the features that are used. In case of scalar encoding, the optimal number of neighbors is 4.	
	

The plots below shows the variation when we exclude particular columns from the data. We did this to analyze how important a column is. In case of our earlier sections, we found that the column 'Backup Start Time' is important in almost all the models. It is clearly evident if we see this graph. The RMSE is way beyond the optimal RMSE observed earlier. Also since the key feature is removed from the dataset, it takes adequate number of neighbors to arrive at an appropriate prediction task. The optimal number of neighbors is 20 here.

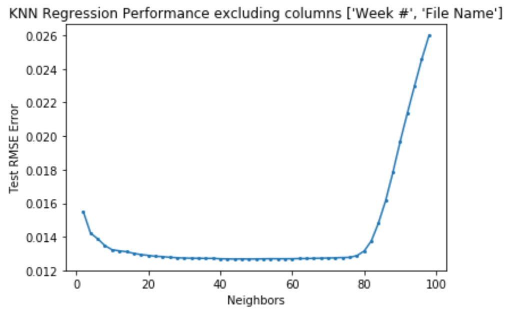


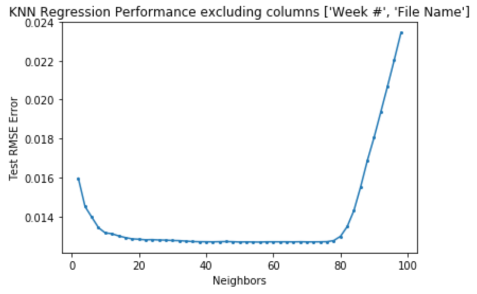






The below is a simple sanity check to see if we get the same trend when the categorical data columns are dropped and we do scalar(left) and one hot encoding(right)





3 Comparison of Models

Summary of test RMSEs

Model	Variation	Min. Avg. Test RMSE
Linear Regression	Initial	0.106503866998
	10 fold CV	0.103627896737
	Standardized data	0.106503866998
	10 fold CV Standardized data	0.103627896737
	f_regression	0.103611218666
	Mutual information	0.103720626763
	One hot + Scaler, shuffle = F	0.0885064915829
	One hot + Scaler, shuffle = T	0.0884431647976
	Ridge Regression	0.0885040092843
	Lasso Regression	0.0885082624122
	Elastic Net	0.0885041347576
Random Forest	Initial	0.0607882493747
	Best model (n_estimators =	0.0129456923991

	13, max_features = 4, max_depth = 10)	
Neural Network	1 layer (each size 200)	0.04192
	2 layers (each size 100)	0.03187
	3 layers (each size 50)	0.02431
	1 layer (each size 200) scalar	0.08717
KNN	Best (Neighbor = 4) scalar	0.03393
	Best (Neighbors = 2) one-hot	0.03576

Analysis

- Amongst these models, random forest is capable of handling categorical data. Random forest models require almost no input preparation as they can handle binary features, categorical features, numerical features.
- As the dataset is not linear, linear regression performs the worst out of all the three regressors.
- Even after standardizing the data or regularization, there is not much improvement in the performance of linear regression.
- Although regularization does help in reducing the RMSE slightly, it is not enough to beat the Random Forest regressor or Neural Networks.
- The best overall performance is obtained for the random forest regressor. The parameters for this best model are n_estimators = 13, max_features = 4, max_depth = 10.
- This is because the random forest algorithm does implicit feature selection for each decision tree in the forest by means of the feature importance attribute.
- Thus, each decision tree in this ensemble model is highly optimized with very little external parameter tuning. This is also the reason why our initial random forest model performs so much better than the linear regression models.
- By tweaking the parameters just a little bit we were even able to beat the RMSE of Neural Networks.
- An additional advantage to the random forest model is that it does not take as much time as neural networks to train and test.
- The neural networks performed bad when the data set is scalar encoded. In this case, neural networks are best suited for sparse data than categorical data.

- It is not fair to compare polynomial regression with the current models as the data is trained on the workflows separately. But it is important to note that the results in that case are much better compared to any other models.
- If we order them by the computational complexity, Linear Regression requires much less resources whereas Neural Network regressor consumes the most of all the models we tested for the prediction task.