

EE 219: Large-Scale Data Mining: Models and Algorithms

# Project 4: Regression Analysis

---

Akshay Sharma (504946035)

Anoosha Sagar (605028604)

Nikhil Thakur(804946345)

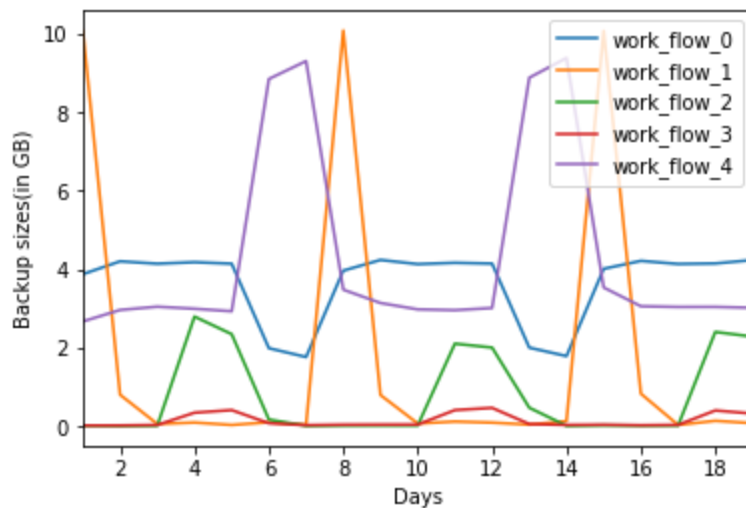
Rahul Dhavalikar (205024839)

---

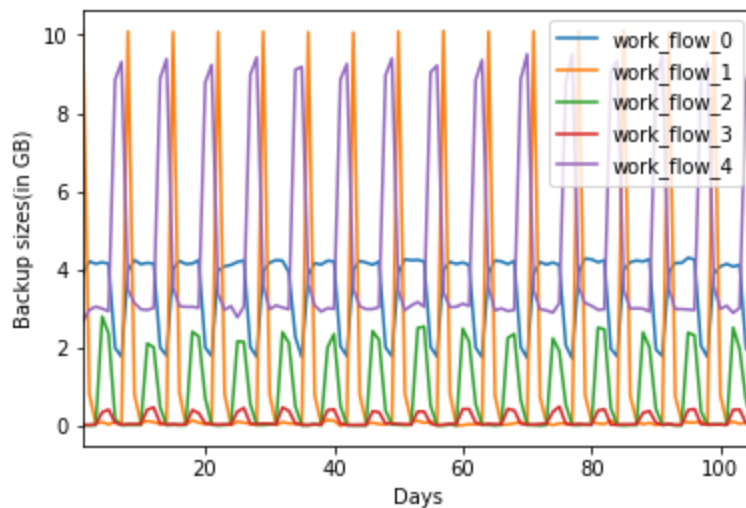
---

## I) Load the dataset

(a) For a twenty-day period (X-axis unit is day number) plot the backup sizes for all workflows (color coded on the Y-axis)



(b) Do the same plot for the first 105-day period.



Q) Can you identify any repeating patterns?

- For workflow\_0, the backup size remains almost constant during the week and falls during the weekend.

- 
- For workflow\_1, the backup size is initially high on Monday's upto around 10 GB and decreases upto mid-week. It remains negligible upto the weekend.
  - For workflow\_2, The backup size is highest on Thursday's of every week and low on the other days.
  - For workflow\_3, on an average the backup size is the lowest among all the workflows. There's a slight, negligible spike on Friday's of the week.
  - For workflow\_4, the backup size remains almost constant at around 2.5 GB upto Friday. There's a huge increase on Saturday's and Sunday's. It's highest on Sunday's at around 9 GB.

## II) Predict the backup size of a file given the other attributes

### (a) Linear Regression Model

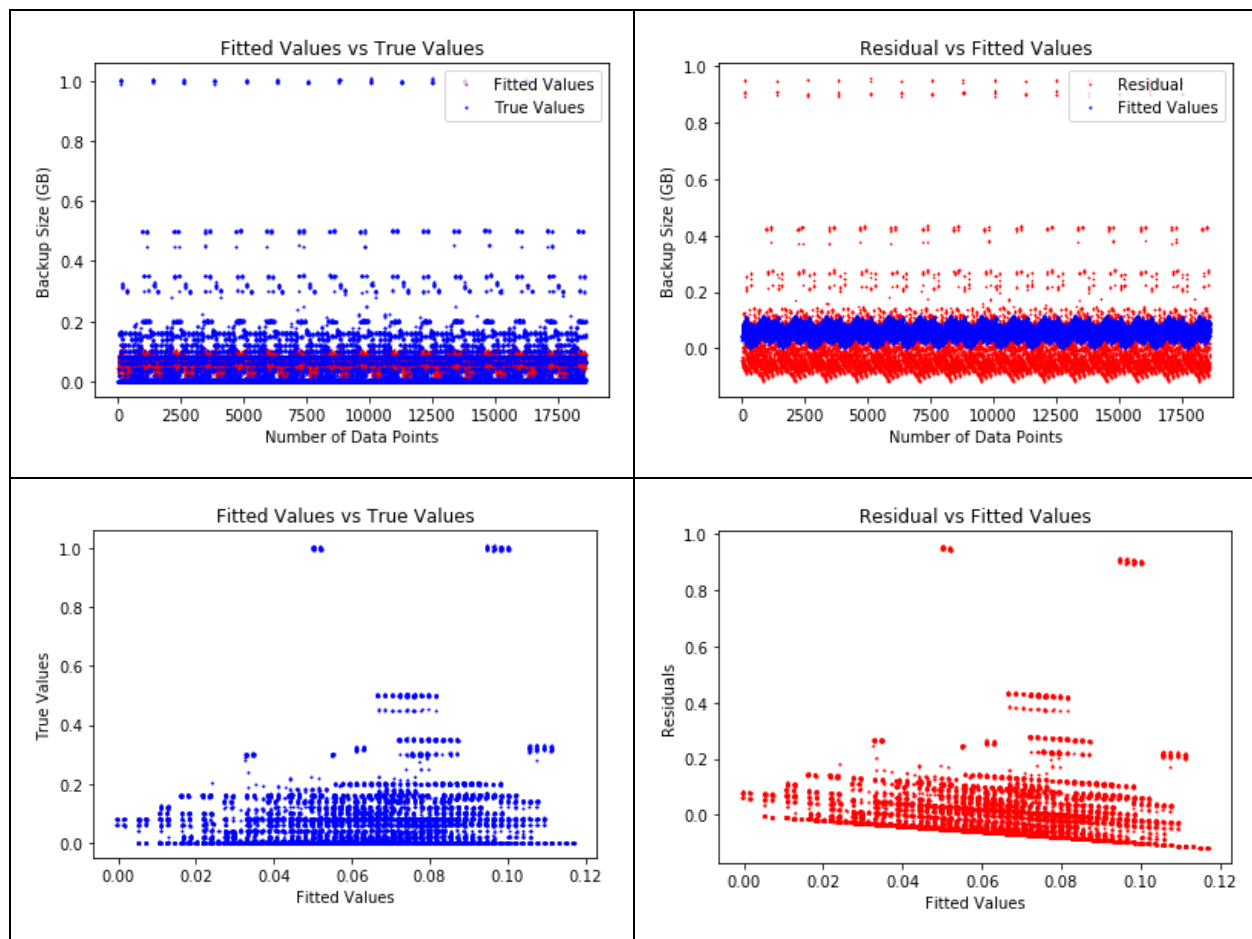
We used the '**LinearRegression**' function from the sklearn.linear\_model package for building the Linear Regression model. We have used default parameters for this task.

#### (i) Convert each categorical feature into one dimensional numerical values using scalar encoding

For scalar encoding, we have used the **LabelEncoder** from sklearn.preprocessing package.

The results for this are compiled in the table below.

<b>Training RMSE</b>	0.1018381968157108
<b>Testing RMSE</b>	0.10186627832709541

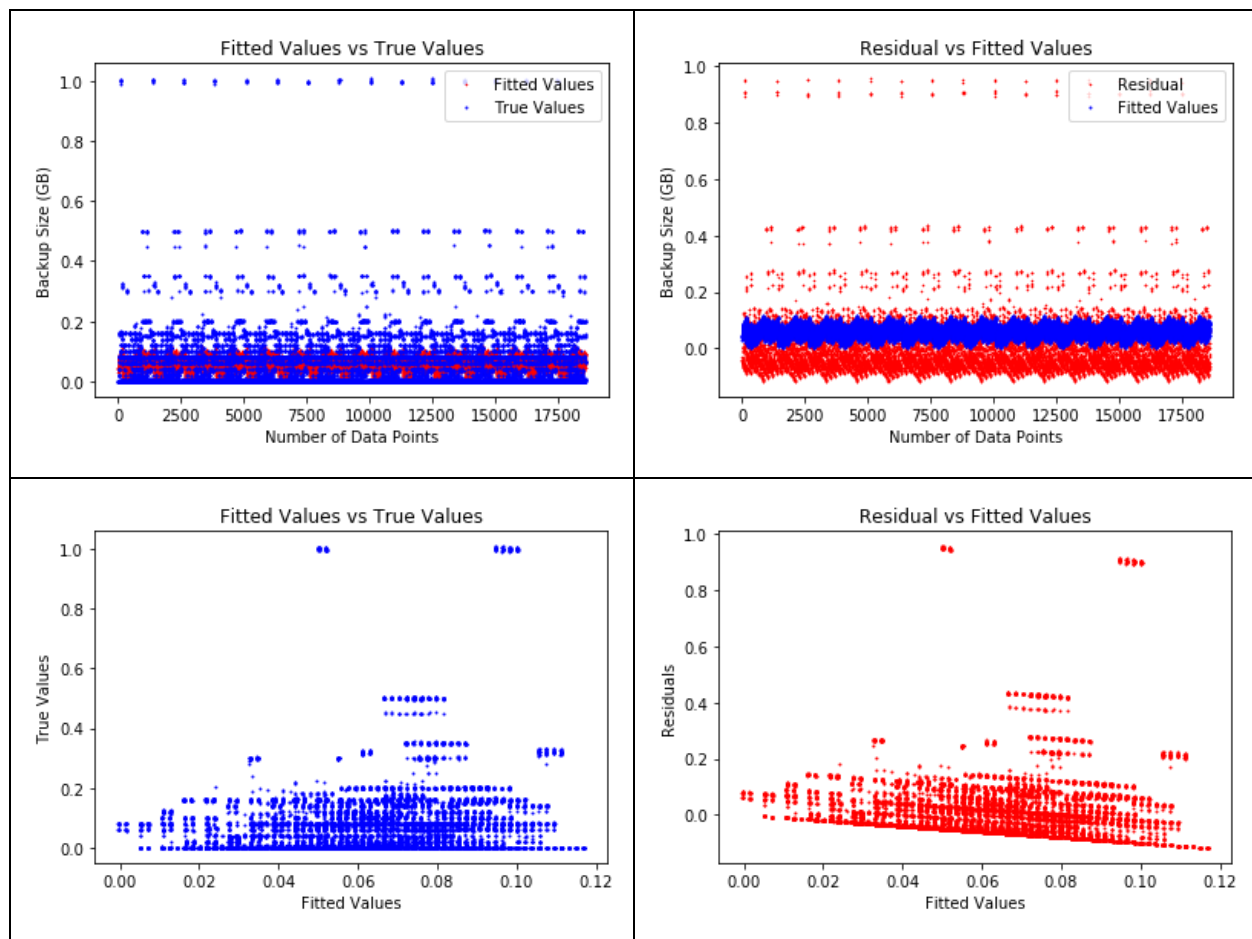


## ii) Data Preprocessing: Standardize all the numerical features

Now, we standardize features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

We have used the **StandardScaler** function from sklearn.preprocessing package for this task. The results for this are compiled in the table below.

<b>Training RMSE</b>	0.1018374764793533
<b>Testing RMSE</b>	0.10187900000264374
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>



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

After comparing these results with the previous results, we see that there's negligible change in testing RMSE after standardization.

### iii) Feature Selection

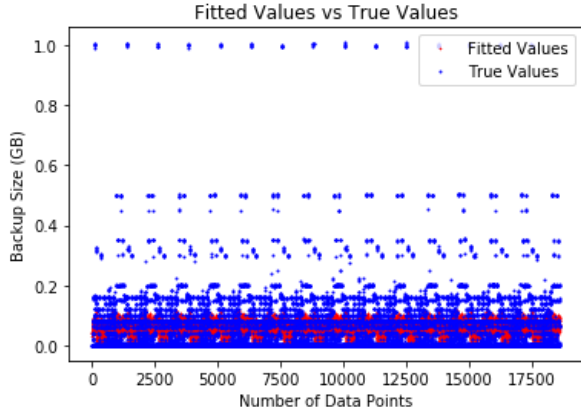
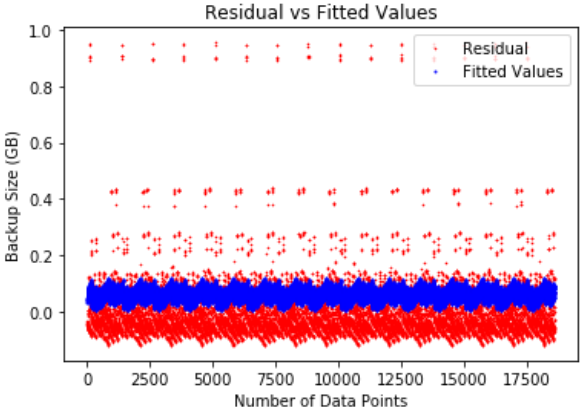
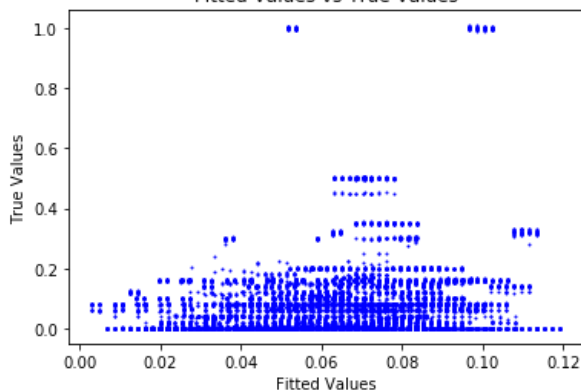
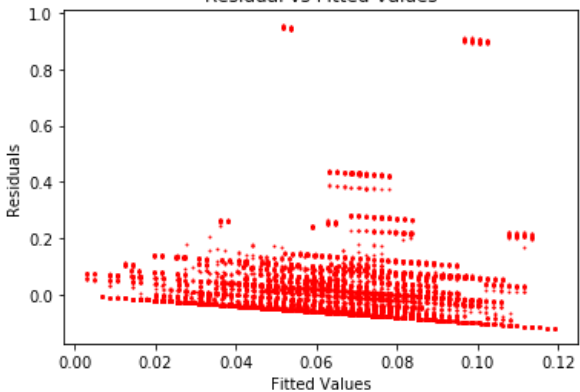
We tried to select best features using both the methods, 'F\_regression' and 'Mutual Info Regression'. We used the **SelectKBest** function from the sklearn.feature\_selection package for this task.

Best features according to 'F\_regression' were,

- Day of week
- Backup-Start Time

- File-Name

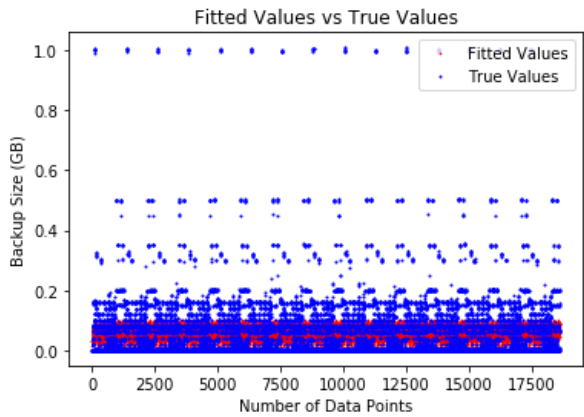
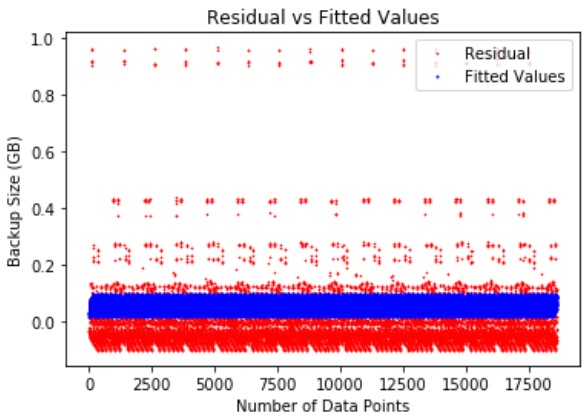
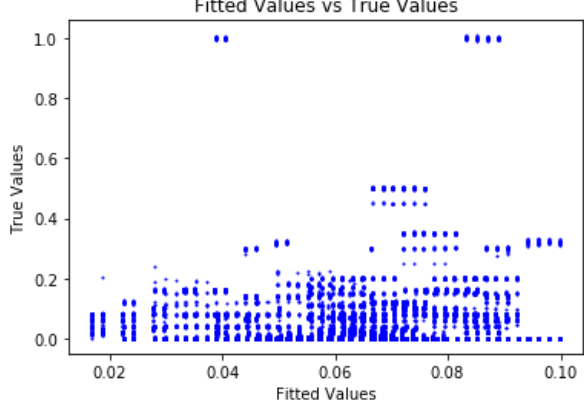
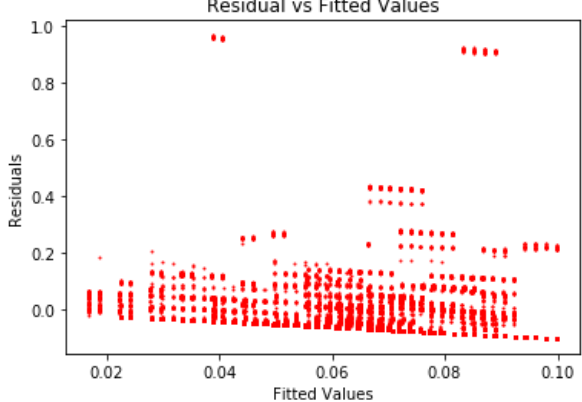
and results using the best features are as below,

<b>Training RMSE</b>	0.10187202291663523
<b>Testing RMSE</b>	0.10189277430914348
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	
	

Best features according to 'Mutual\_Info\_regression' were,

- Backup-Start Time
- Work-Flow-ID
- File Name

and results using the best features are as below,

<b>Training RMSE</b>	0.10246762590008582
<b>Testing RMSE</b>	0.10250615345693079
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	
	

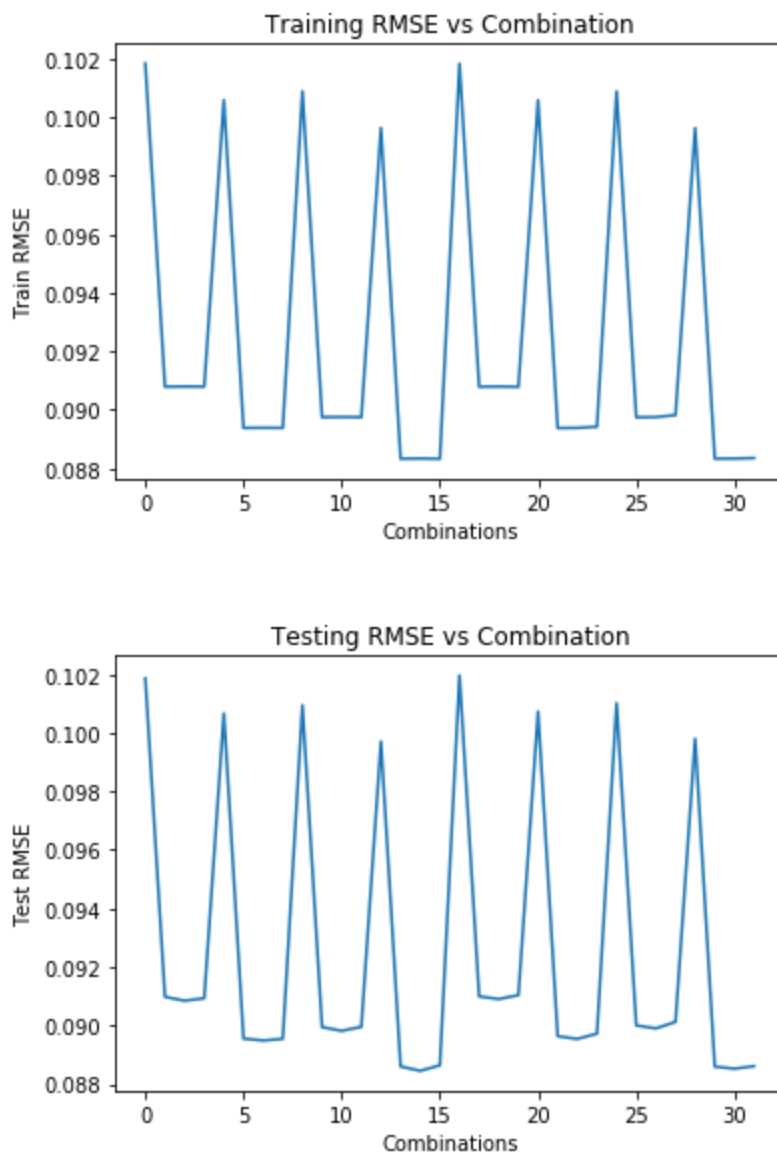
Among F\_regression and Mutual Info Regression, we get slightly better testing rmse with F\_regression. The performance remains almost the same even after using the best features.

---

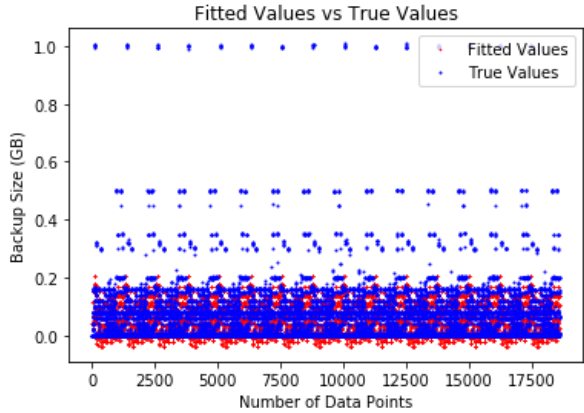
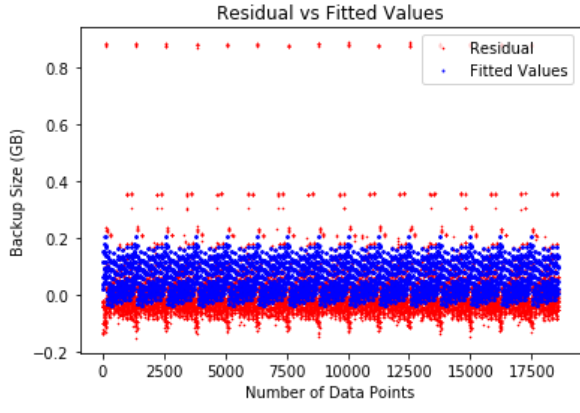
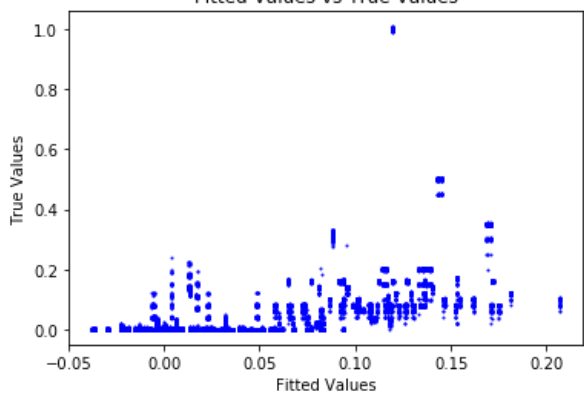
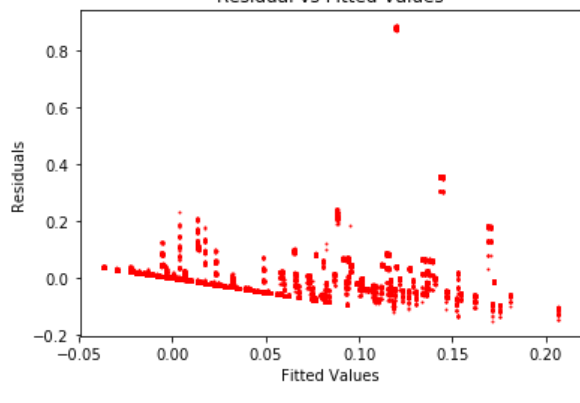
#### iv) Feature Encoding

There are 32 possible combinations of encoding the five categorical variables. We have given below the plot of Training and Testing RMSE for all the 32 combinations. The best combination we get is as below - 14 (01110) where the columns with 1 are one hot encoded. So best combination is when the Day of week, Backup-Start Time & Work-Flow-ID columns are one-hot encoded.

We have then plotted the plots for the best combination.





<b>Training RMSE</b>	0.08834016633474057
<b>Testing RMSE</b>	0.0884528308663724
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	
	

**Q) Which combinations achieve best performance? Can you provide an intuitive explanation?**

The best performance is achieved by the combination which has the Day of week, Backup-Start Time & Work-Flow-ID columns one-hot encoded. These columns are also one of the most important features as reported by the `f_regression` and `mutual info regression`. In some other cases we see similar trends that when majority of the columns are one-hot

---

encoded we get better results. This is because LabelEncoder converts data to have ordinal relationships, which basically means that the regression algorithm will assume that Sunday > Saturday > Friday which is not true in our case. One hot encoding transforms categorical features to a format that works better with classification and regression algorithms.

So in the testRMSE vs combinations graph, when more features are one-hot encoded we see a dip in the graph and when less features are one-hot encoded we see a peak in the graph.

### **v) Controlling ill-conditioning and over-fitting**

**Q) You should have found obvious increases in test RMSE compared to training RMSE in some combinations, can you explain why this happens?**

The reason for increase in test RMSE as compared train RMSE is overfitting. The model is trying to memorize the data exactly as it is and hence is not performing well on the test set.

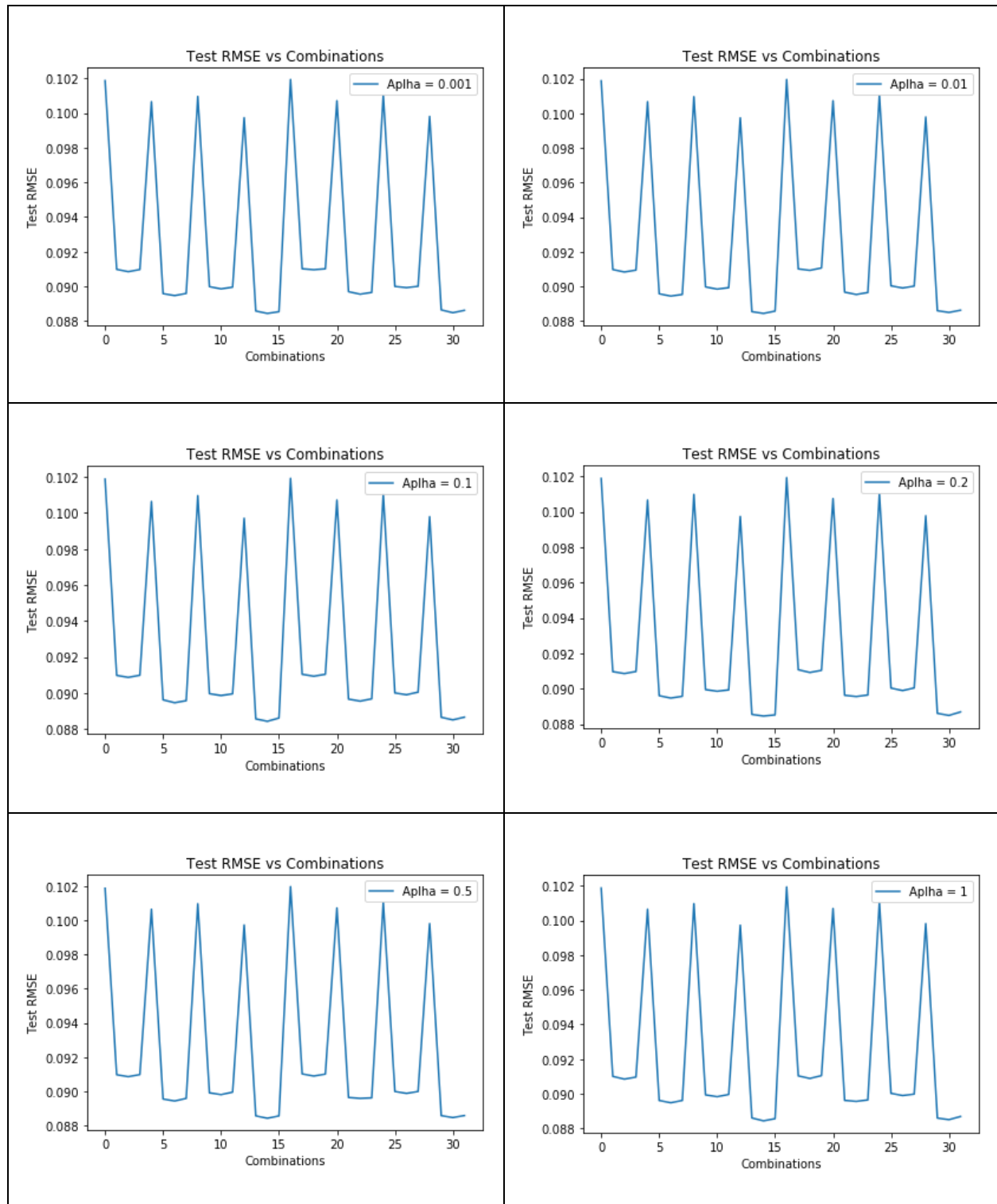
For solving it ,we use the following regularizers,

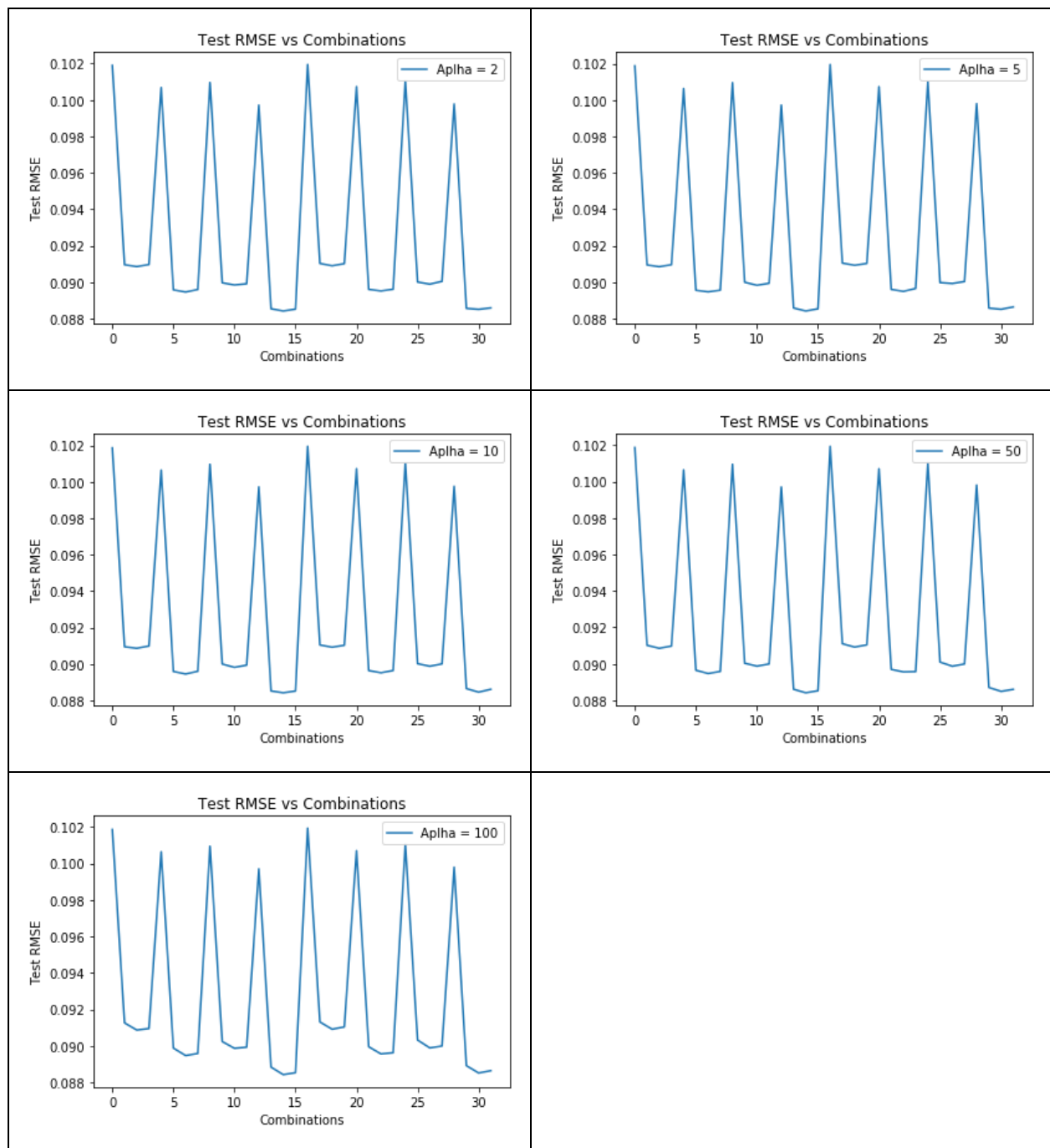
- **Ridge Regularizer** - Linear least squares with  $L_2$  regularization.
- **Lasso Regularizer** - Linear Model trained with  $L_1$  prior as regularizer
- **ElasticNet Regularizer** - Linear regression with combined  $L_1$  and  $L_2$  priors as regularizer.

We use this models from the sklearn.linear\_model package.

We run each of the regularizers for each alpha for each of 32 combinations and report the best combination.

## Using Ridge Regularizer





**Best Alpha: 50**

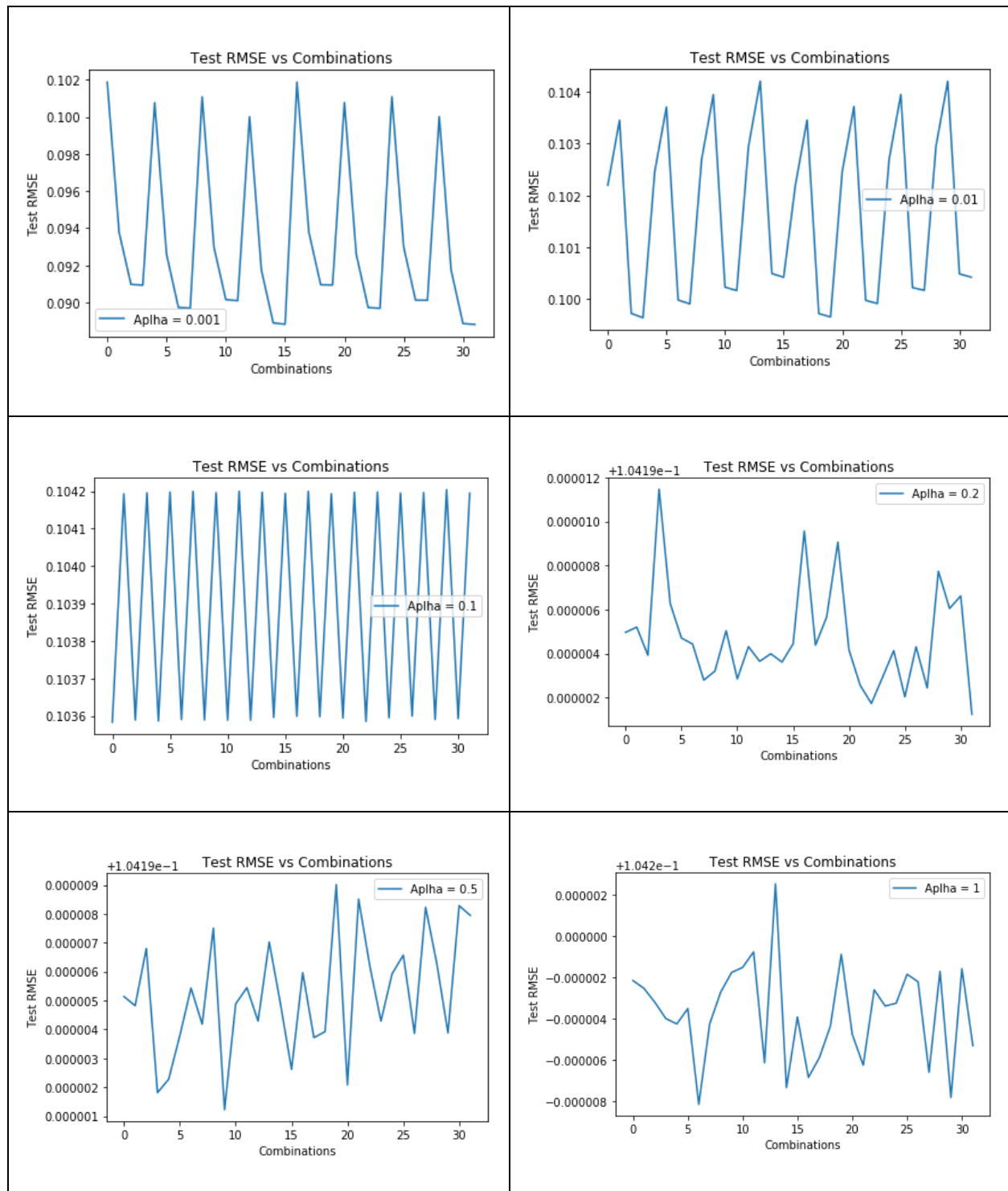
**Best Combination: 15 - refers to the following features,**

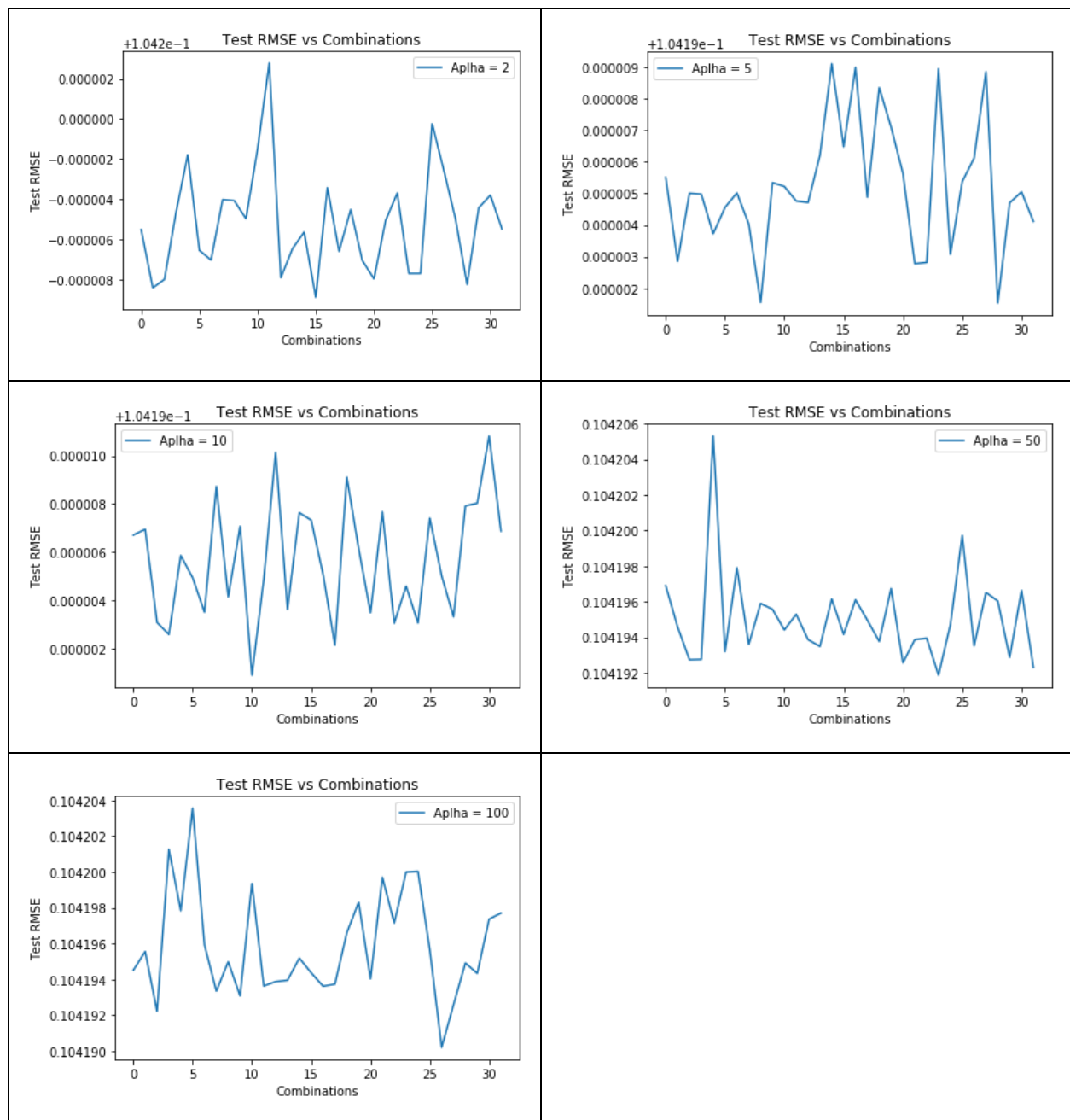
15 = (01111) where the columns with 1 are one hot encoded. So best combination is when the below columns are one-hot encoded.

- Day of week
- Backup-Start Time
- Work-Flow-ID
- File Name

<b>Training RMSE</b>	0.08834611203264654
<b>Testing RMSE</b>	0.08843264762881899
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>

## Using Lasso Regularizer



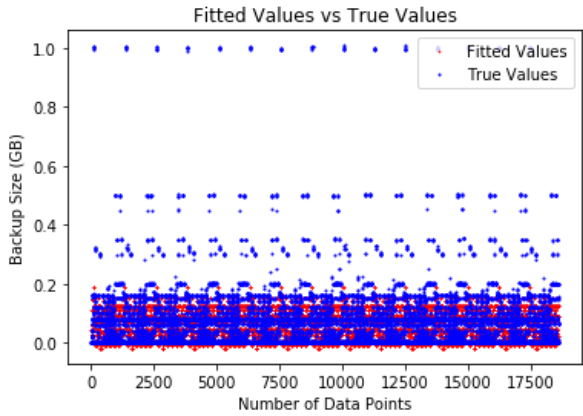
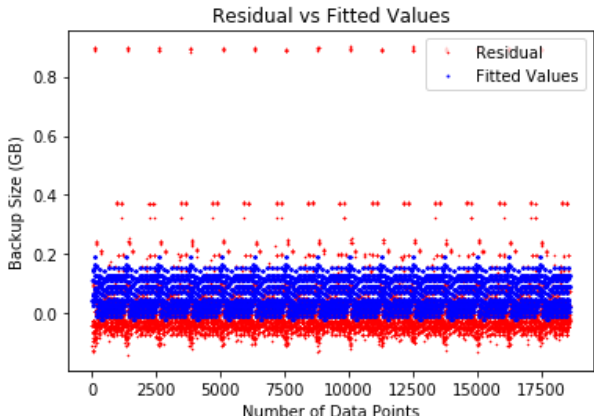
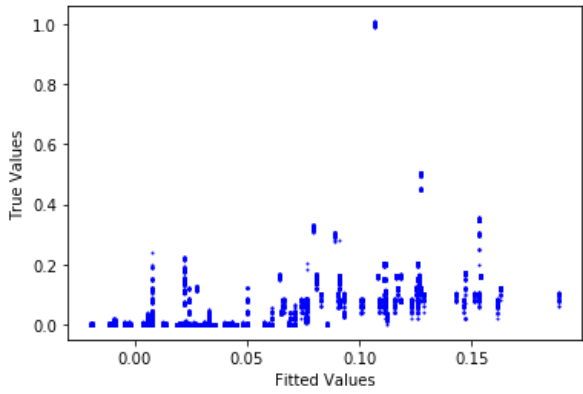
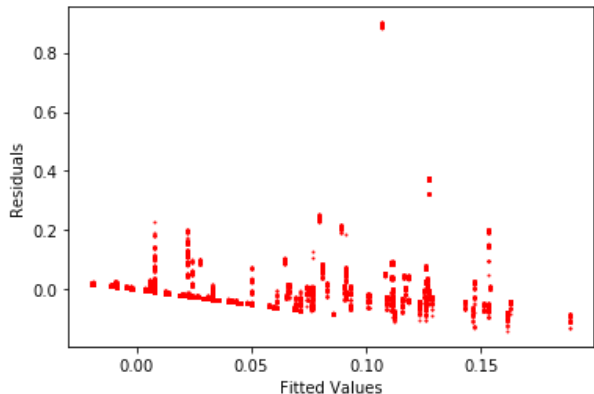


**Best Alpha: 0.001**  
**Best Combination: 14**

14 = (01110) where the columns with 1 are one hot encoded. So best combination is when the below columns are one-hot encoded.

- Day of week
- Backup-Start Time
- Work-Flow-ID

And the results for this best combination are as follows.

<b>Training RMSE</b>	0.08877872361488477
<b>Testing RMSE</b>	0.08884607126090166
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	
	

## Using Elastic Net Regularizer

There are many cases for different values for alpha and L-ratio. The lowest Test RMSE is got at alpha = 0.001 and L-ratio = 0.01. The graph of all the combinations vs the Test RMSE is



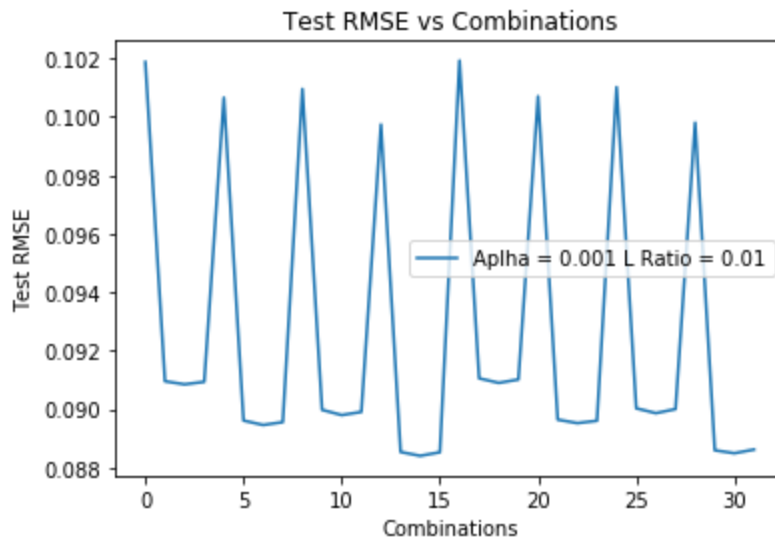
---

plotted below. On observing this graph we see that the best combination of feature encoding is combination 14

14 = (01110) where the columns with 1 are one hot encoded. So best combination is when the below columns are one-hot encoded.

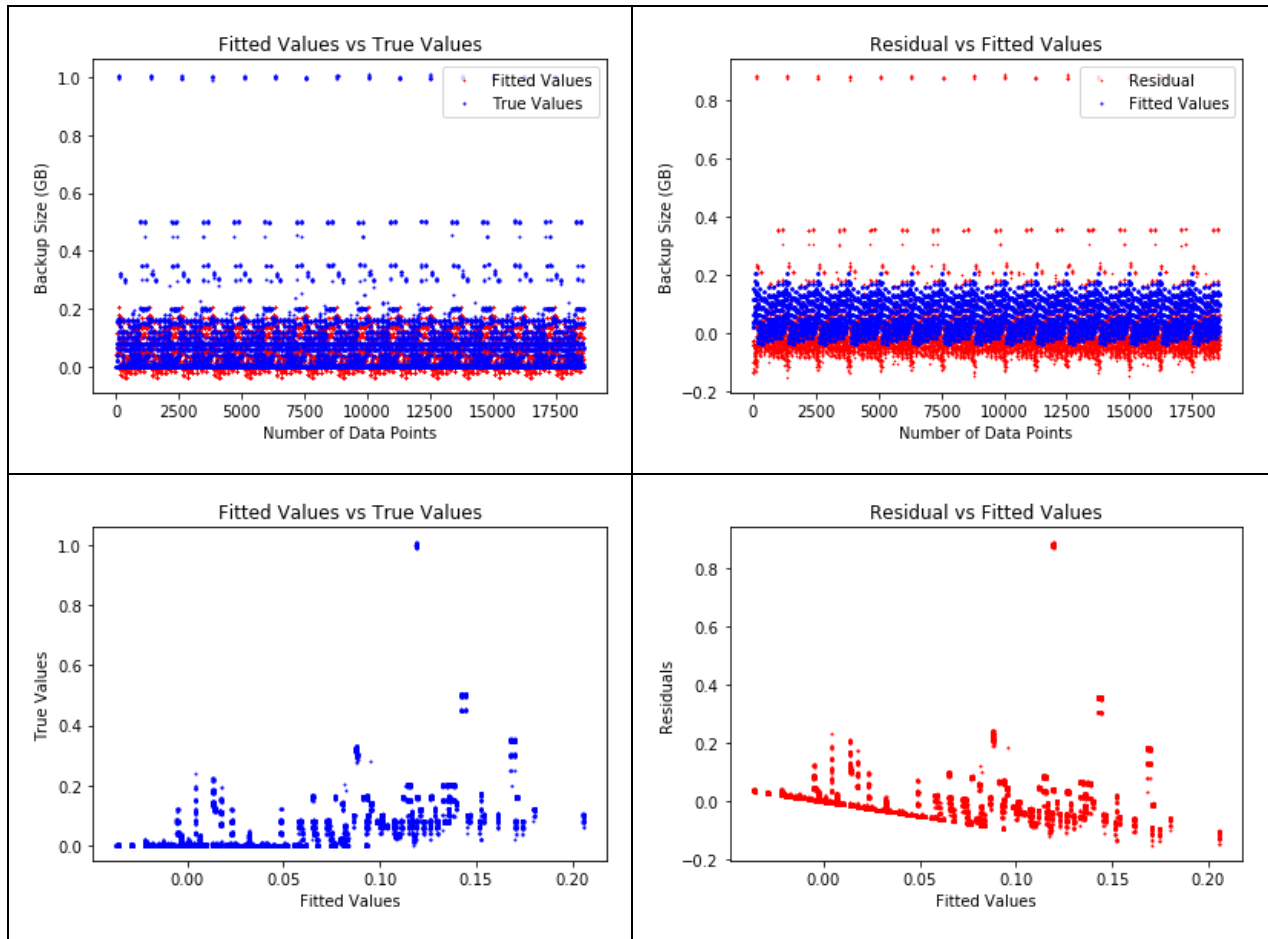
- Day of week
- Backup-Start Time
- Work-Flow-ID

And the results for this best combination are as follows.



**Results for Alpha: 0.001 , Lratio: 0.01**  
**Best Combination: 14**

Training RMSE	0.08834219045934692
Testing RMSE	0.08843024945120317
Fitted values vs True values	Residuals vs Fitted values



The coefficients for the original model are as below,

#### Unregularized Model:

[ 1.76468470e+10, 1.76468470e+10, 1.76468470e+10, 1.76468470e+10,  
 1.76468470e+10, 1.76468470e+10, 1.76468470e+10, 1.76468470e+10,  
 1.76468470e+10, 1.76468470e+10, 1.76468470e+10, -6.43495217e+09,  
 -6.43495217e+09, -6.43495217e+09, -6.43495217e+09,  
 -6.43495217e+09, -6.43495217e+09, -3.59756909e+11, -3.59756909e+11,  
 -3.59756909e+11, -3.59756909e+11, -3.59756909e+11,  
 9.77362447e+10, 4.44081078e+10, 3.49429401e+11, 3.04586190e+10,  
 6.83186214e+10, 7.02588794e+09, 7.02588794e+09, 6.03540248e+10,  
 6.03540248e+10, -2.44667268e+11, -2.44667268e+11, -2.44667268e+11,  
 -2.44667268e+11, -2.44667268e+11, -2.44667268e+11, 7.43035136e+10,  
 7.43035136e+10, 7.02588794e+09, 7.43035136e+10, 7.43035136e+10,

---

7.43035136e+10, 7.43035136e+10, 3.64435112e+10, 3.64435112e+10,  
3.64435112e+10, 3.64435112e+10, 3.64435112e+10, 3.64435112e+10,  
7.02588794e+09, 7.02588794e+09, 7.02588794e+09, 6.03540248e+10,  
6.03540248e+10, 6.03540248e+10, 6.03540248e+10]

The unregularized model has very high coefficients of the order of  $10^{10}$

The coefficients for all the 3 models are as below,

**Ridge Regularizer:**

[-7.59429660e-04, -3.83365373e-05, 3.36953666e-04, -3.36613364e-04,  
3.16547870e-04, -2.05725282e-04, 7.76638907e-06, 7.91876068e-04,  
1.21694115e-04, 1.19487866e-04, -5.47226447e-04, 6.93885030e-04,  
-1.77627770e-04, -1.07999633e-05, -3.12451981e-04, -5.59670306e-03,  
3.85773695e-02, 3.21649081e-03, 1.46837553e-03, -5.15394786e-03,  
-1.26118864e-02, -1.98996986e-02, -1.98905508e-02, -2.07121511e-02,  
7.66858823e-03, 3.29308502e-02, -1.97795774e-03, 1.98122116e-03,  
3.23917798e-02, -1.19181244e-02, -3.40688896e-02, -4.82299982e-02,  
6.18252324e-02, 4.73376745e-03, 5.88864855e-03, -2.24234816e-03,  
-1.93965813e-03, -6.36862302e-03, -5.98398342e-03, -5.54548160e-03,  
-6.53342797e-03, -4.20333517e-03, -5.43403840e-03, -8.18717736e-03,  
-7.88414290e-03, 5.58443780e-03, -7.82959915e-03, -8.16278970e-03,  
-8.20782157e-03, -7.95846752e-03, 1.00634167e-02, 1.05853947e-02,  
1.07609576e-02, 9.77311117e-03, 1.03717122e-02, 1.02706399e-02,  
5.55993344e-03, 5.19875509e-03, 5.42623743e-03, -1.63007015e-03,  
-2.07653853e-03, -1.88387651e-03, -2.14563287e-03]

Compared to the unregularized model, model with ridge regularizer has much lower coefficients in the range of  $10^{-3}$

**Lasso Regularizer:**

[-0. , -0. , 0. , -0. , 0. ,  
-0. , 0. , 0. , 0. , 0. ,  
-0. , 0. , -0. , -0. , -0. ,  
-0. , 0.03551031, 0. , 0. , -0. ,

---

```
-0.0021173 , -0.01013452, -0.0143399 , -0.01480008, 0.00185475,  
0.02751801, -0.    , 0.    , 0.04725556, -0.    ,  
-0.02138538, -0.03814218, 0.08187752, 0.    , 0.    ,  
-0.    , 0.    , -0.    , -0.    , -0.    ,  
-0.    , -0.    , -0.    , -0.    , -0.    ,  
0.    , -0.    , -0.    , -0.    , -0.    ,  
0.    , 0.    , 0.    , 0.    , 0.    ,  
0.    , 0.    , 0.    , 0.    , 0.    ,  
-0.    , 0.    , -0.    ]
```

Compared to the unregularized model, model with lasso regularizer has much lower coefficients and are also sparse(with many zeroes)

#### **ElasticNet Regularizer:**

```
[-6.12966212e-04, -0.00000000e+00, 2.15189235e-04, -1.80065161e-04,  
1.94737612e-04, -4.54012864e-05, 0.00000000e+00, 6.81518799e-04,  
0.00000000e+00, 0.00000000e+00, -3.96005464e-04, 5.81585405e-04,  
-1.66324078e-05, 0.00000000e+00, -1.55352592e-04, -4.13800278e-03,  
4.04292942e-02, 4.64458066e-03, 2.87608419e-03, -3.68651976e-03,  
-1.12363272e-02, -1.86037890e-02, -2.00687017e-02, -2.09069711e-02,  
7.64812519e-03, 3.31624137e-02, -1.96191995e-03, 1.90096582e-03,  
3.99985586e-02, -1.15288180e-02, -3.65344637e-02, -5.36604278e-02,  
7.14724336e-02, -0.00000000e+00, 8.43653166e-04, -8.97633787e-05,  
-0.00000000e+00, -1.65507960e-03, -1.25271000e-03, -7.92056059e-04,  
-1.82842996e-03, 3.77417704e-05, -6.75693656e-04, -6.80038742e-04,  
-3.62232852e-04, 5.24344769e-04, -3.05027961e-04, -6.54451543e-04,  
-7.01673926e-04, -4.40164389e-04, 3.23690275e-03, 3.78432433e-03,  
3.96844927e-03, 2.93246980e-03, 3.56024840e-03, 3.45425725e-03,  
4.98672542e-04, 1.19475565e-04, 3.58503623e-04, 0.00000000e+00,  
-0.00000000e+00, -0.00000000e+00, -0.00000000e+00]
```

Compared to the unregularized model, model with elasticnet regularizer has much lower coefficients in the range of  $10^{-3}$

## (b) Random forest regression

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. We have used the **RandomForestRegressor** provided in sklearn package. We have varied some of the key parameters of this model in the later parts. Some details about these parameters are as follows.

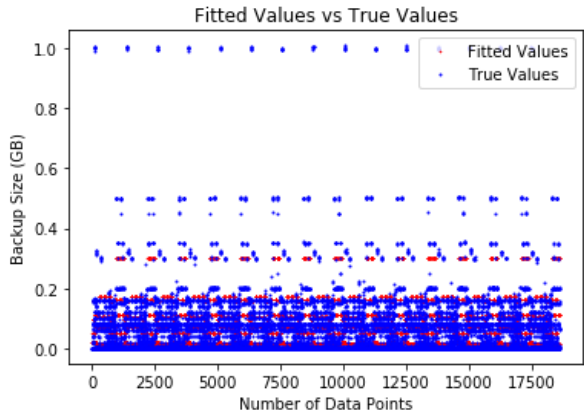
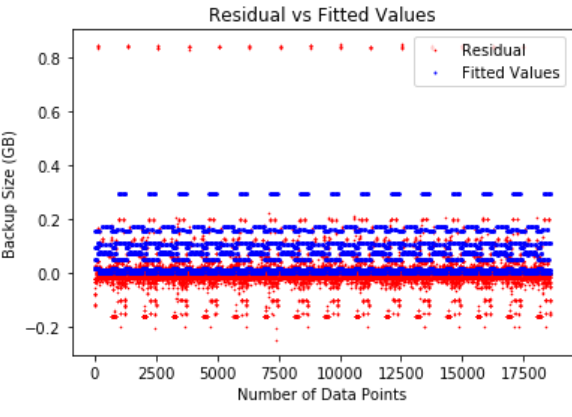
**n\_estimators** : The number of trees in the forest.

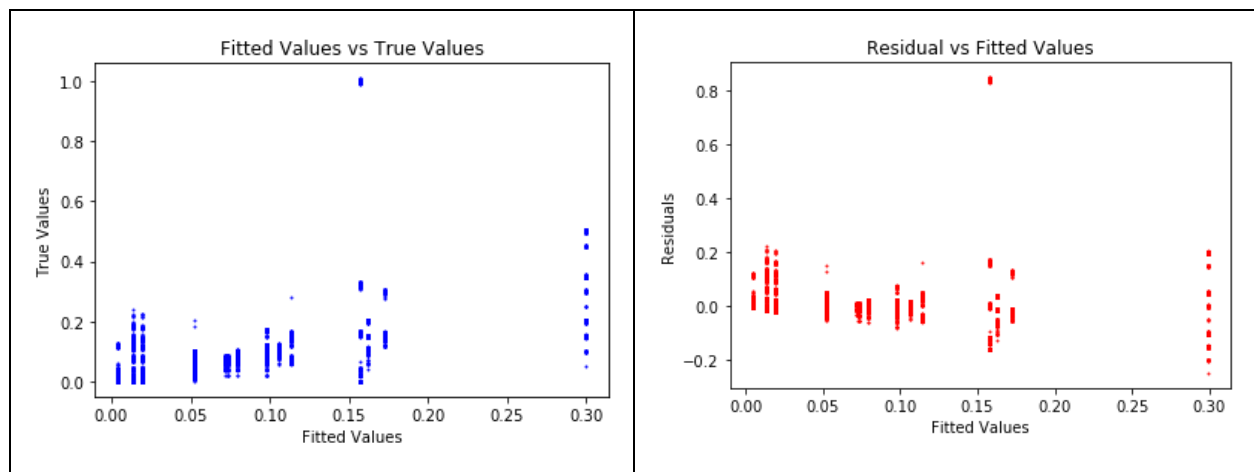
**max\_features** : The number of features to consider when looking for the best split: If int, then consider max\_features features at each split.

**max\_depth** : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

### (i) Initial Model

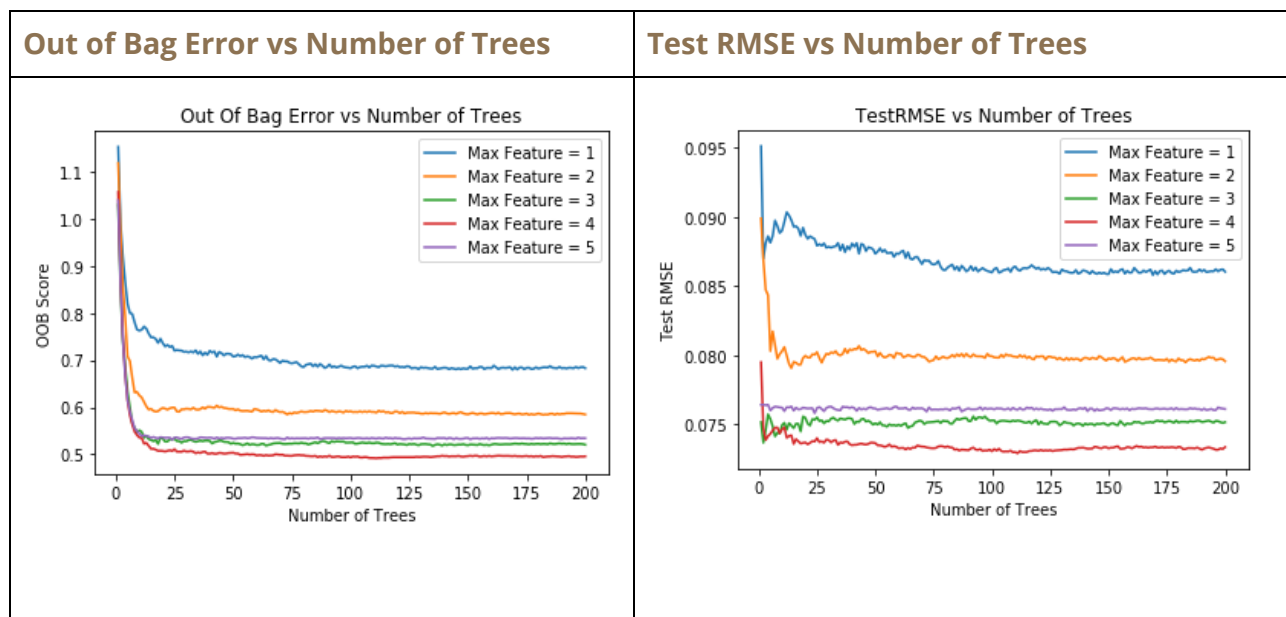
In the table below we first explore a basic model without optimizations.

<b>Training RMSE</b>	0.07602492442266995
<b>Testing RMSE</b>	0.07611235229518903
<b>Out of Bag Error</b>	0.5344337250689708
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	

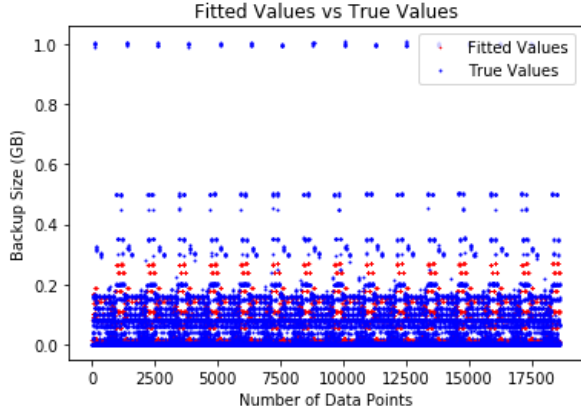
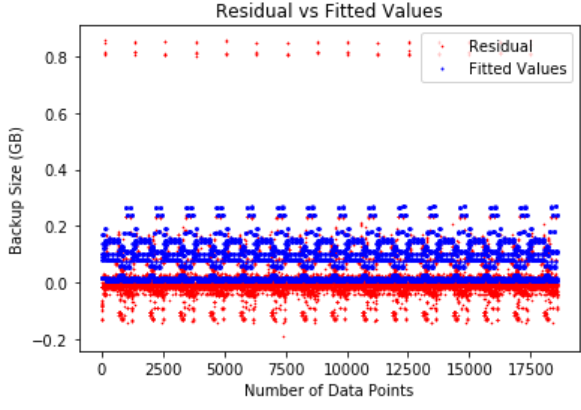
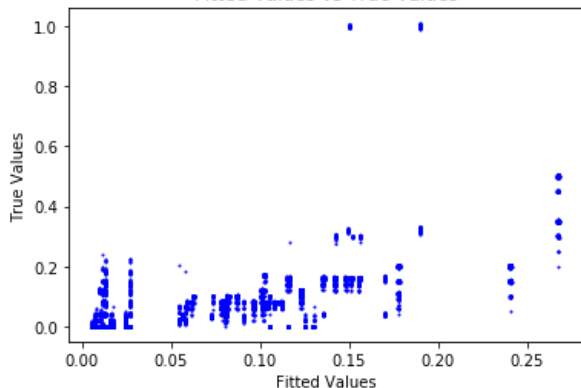
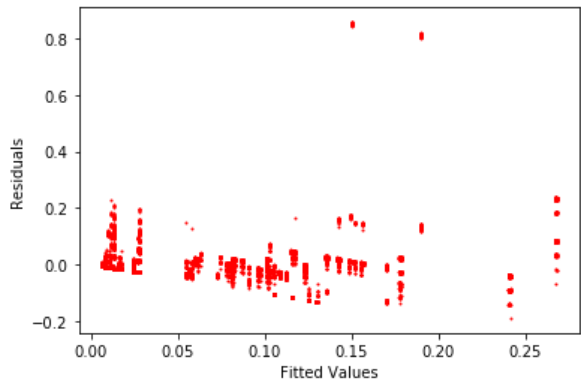


## (ii) Sweep over number of trees and maximum number of features

In this part, we have kept the max depth constant(=4) and varied the number of trees and maximum features. As seen in the graphs below, as the number of trees increases the testRMSE and OOB error keep on steadily decreasing. We have used the parameter combination that gives the lowest testRMSE as the best combination.



**Best Model with Max Depth = 4 (constant):**  
**Max features = 4**  
**Number of trees = 111**

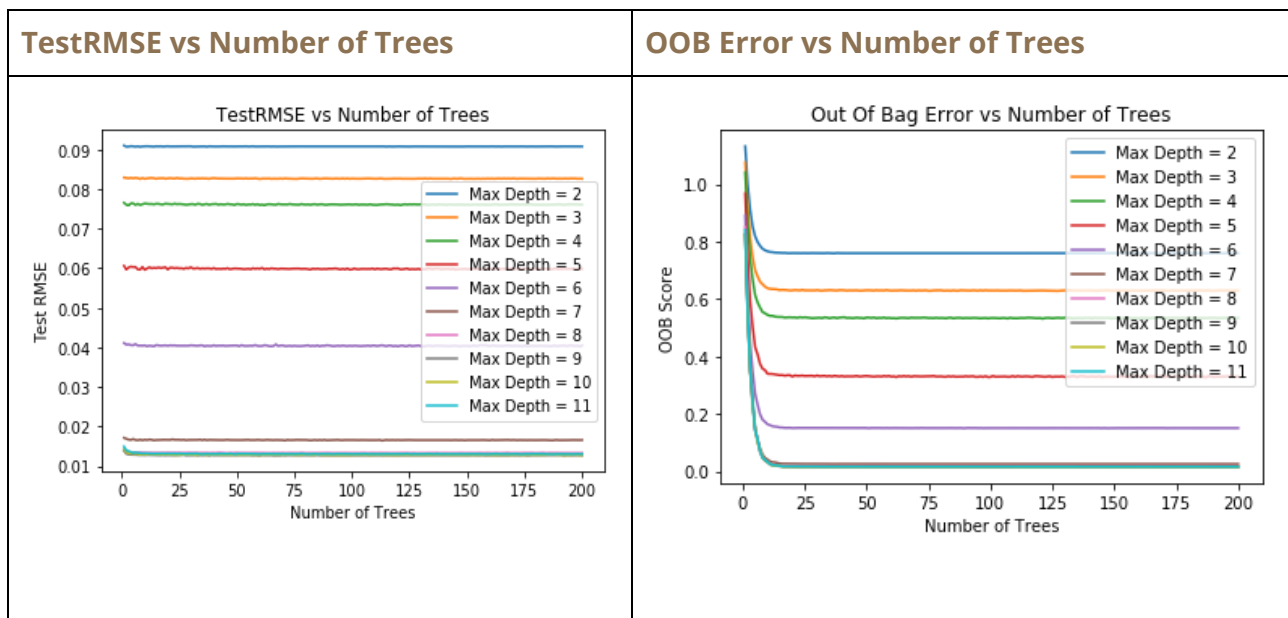
<b>Training RMSE</b>	0.07283580579538636
<b>Testing RMSE</b>	0.07291689560179437
<b>Out of Bag Error</b>	0.4917337249389283
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	
	

### (iii) Sweep over number of trees and max depth

#### Q) What parameters would you pick to achieve the best performance?

The performance of random forest regressor mainly depends on 3 parameters - the number of trees, max features and max depth of the trees. In this task we have chosen to keep max features as constant and have varied the number of trees and the max depth.

On choosing these parameters, the test RMSE has indeed decreased even further as compared to the previous part. We even get a much lower OOB error. Below are the results for this experiment.



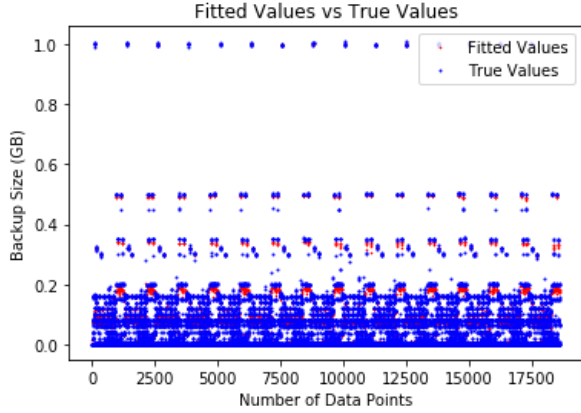
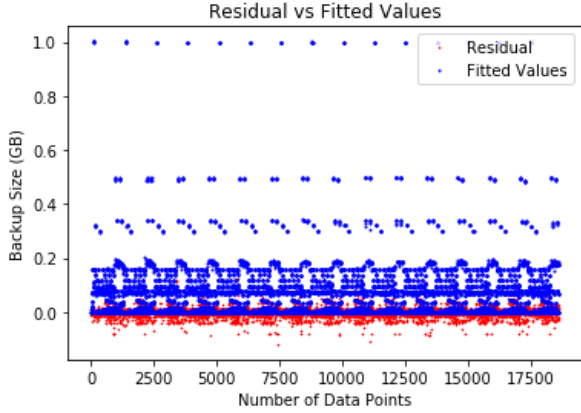
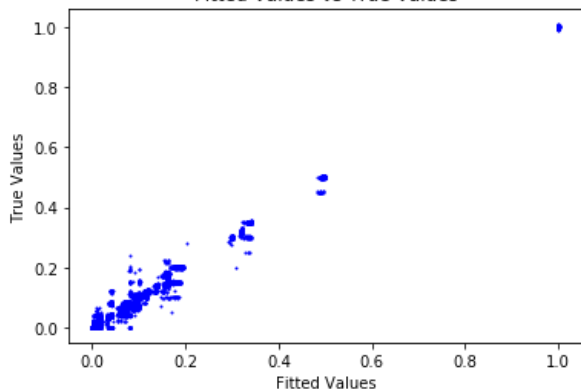
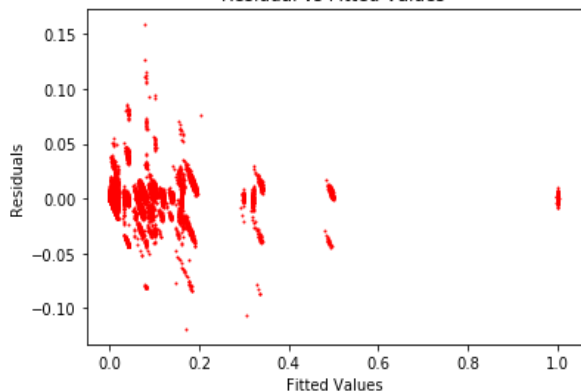
**Best Model with max features = 5 (constant)**

**Max Depth = 9**

**Number of trees = 60**

Training RMSE	0.011626450906815763
---------------	----------------------

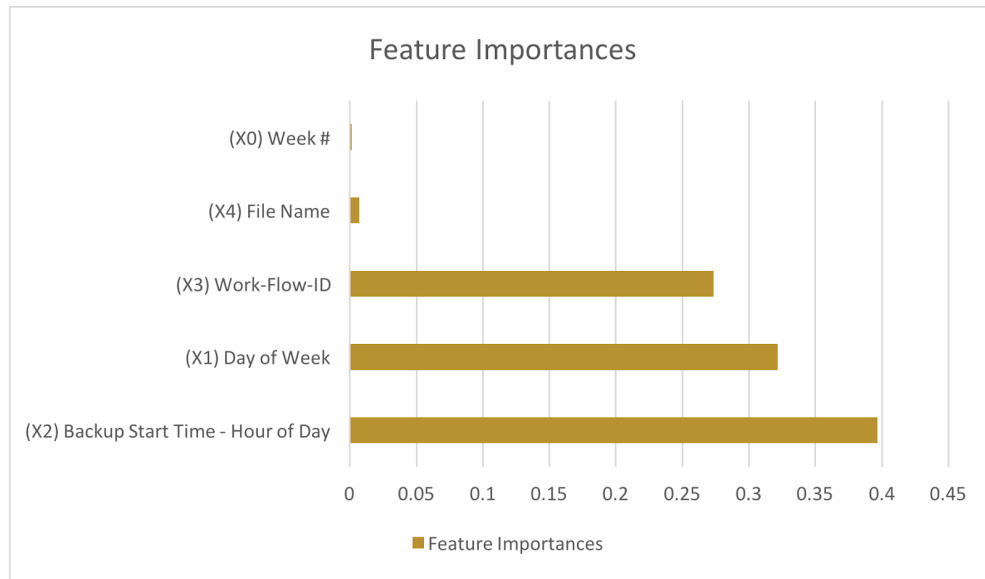


<b>Testing RMSE</b>	0.012628837803137626
<b>Out of Bag Error</b>	0.014896763588996753
<b>Fitted values vs True values</b>	<b>Residuals vs Fitted values</b>
	
	

#### (iv) Feature importances from the best random forest regression

We find the feature importances from the above best random forest regression

<b>(X2) Backup Start Time - Hour of Day</b>	0.39641834
<b>(X1) Day of Week</b>	0.32155655
<b>(X3) Work-Flow-ID</b>	0.27336781
<b>(X4) File Name</b>	0.00720733
<b>(X0) Week #</b>	0.00144996



### (v) Visualize your decision trees

For visualization, we have to keep max\_depth as 4.

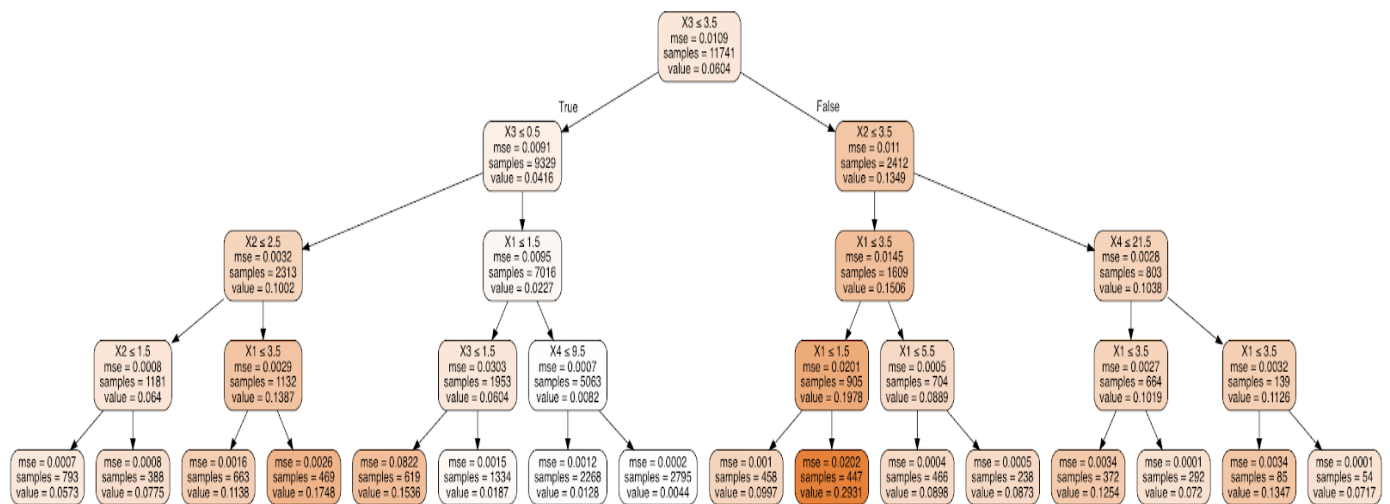
For max depth=4, the best combination is as per part 2(b)(ii) which is

Number of trees = 111, Max features = 4

### Feature importances for this combination

<b>(X3) Work-Flow-ID</b>	4.78777945e-01
<b>(X1) Day of Week</b>	3.44087885e-01
<b>(X2) Backup Start Time - Hour of Day</b>	9.01154596e-02
<b>(X4) File Name</b>	8.69428369e-02
<b>(X0) Week #</b>	7.58731376e-05

### Tree visualization for this combination



**Q) Which is the root node in this decision tree? Is it the most important feature according to the feature importance reported by the regressor?**

The root node in the above decision tree is for the decision on parameter X3 i.e.

Work-Flow-ID. The most important feature according to the feature importance reported by the regressor is Work-Flow-ID as well. Thus, we get consistent results through both the methods. With the most important feature as the root node, we get the best separation of data at the root node.

## (c) Neural network Regression model

For this task we have used the **MLPRegressor** from sklearn.

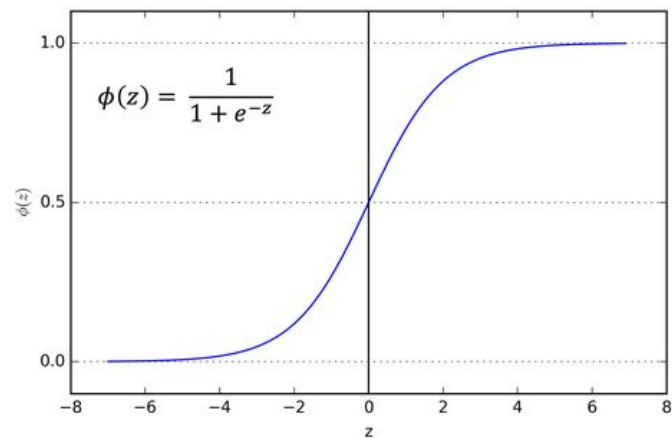
MLPRegressor trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting.

The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

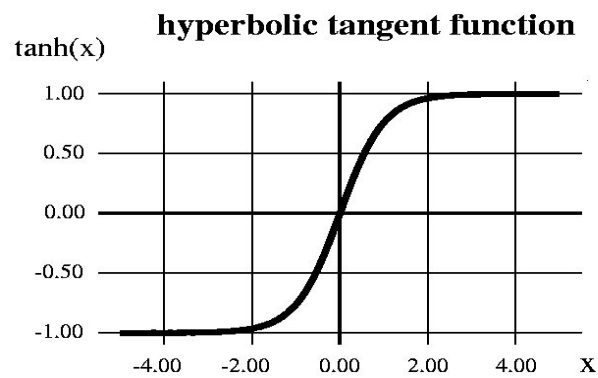
---

So we have experimented with both - the 'lbfgs' solver and 'adam' solver for this task. We will also be experimenting with 3 activation functions.

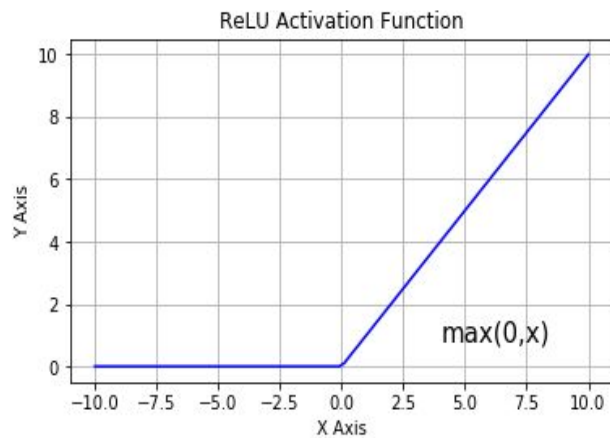
Logistic Function:



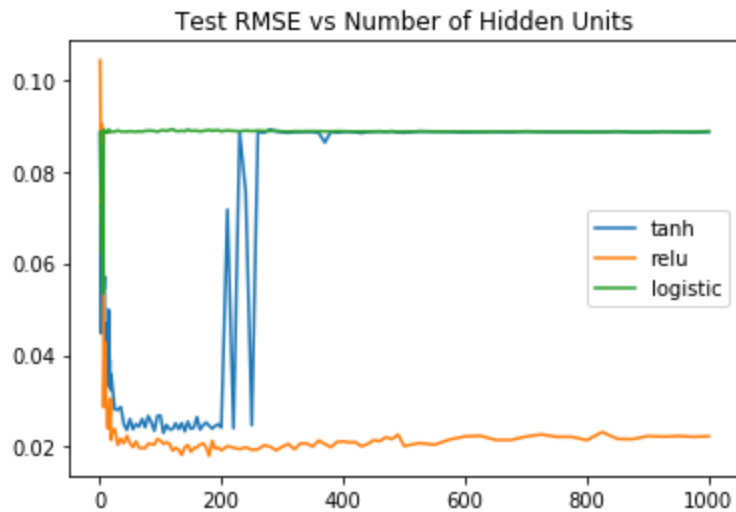
Tanh Function:



Relu Function:

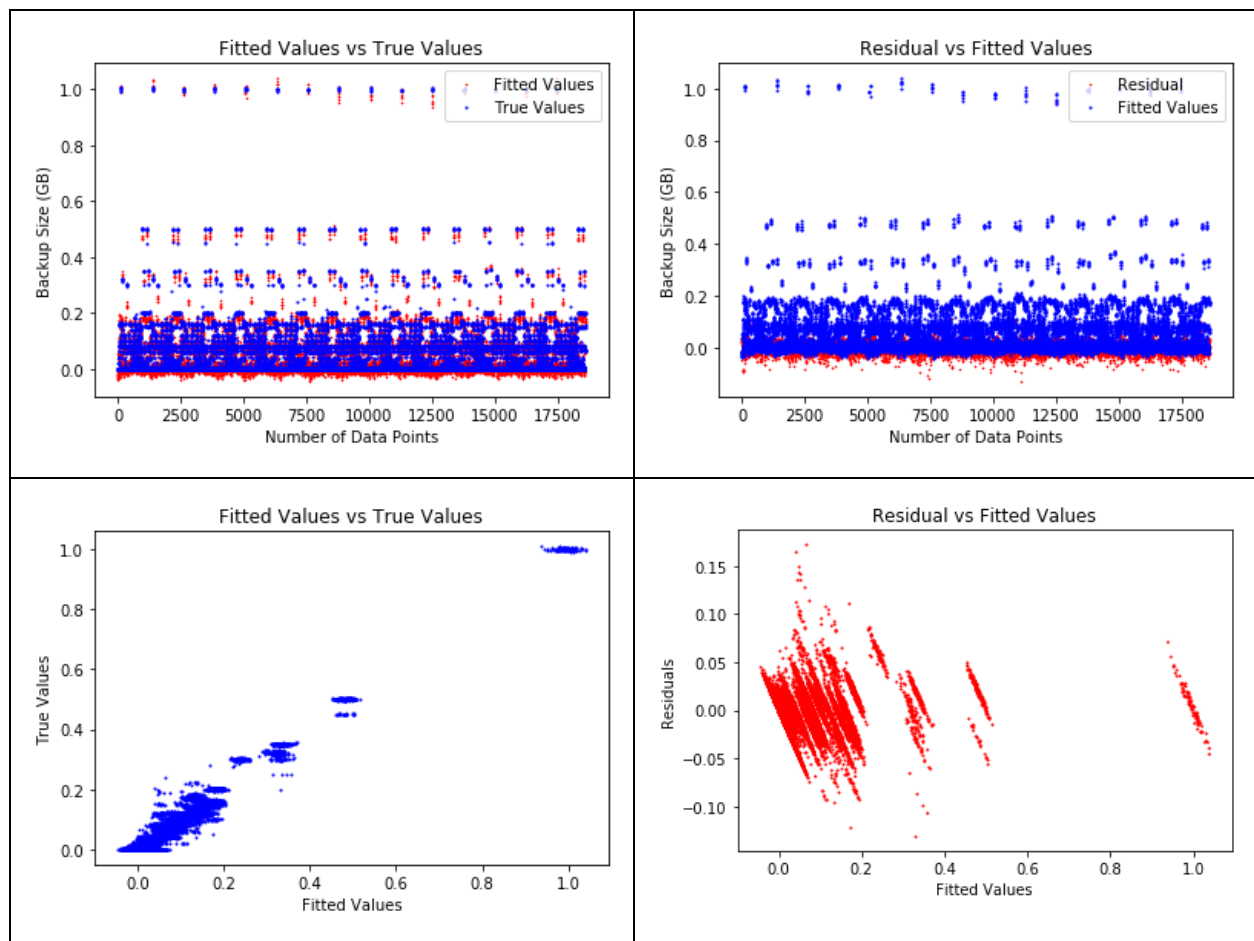


## Ibfgs Solver

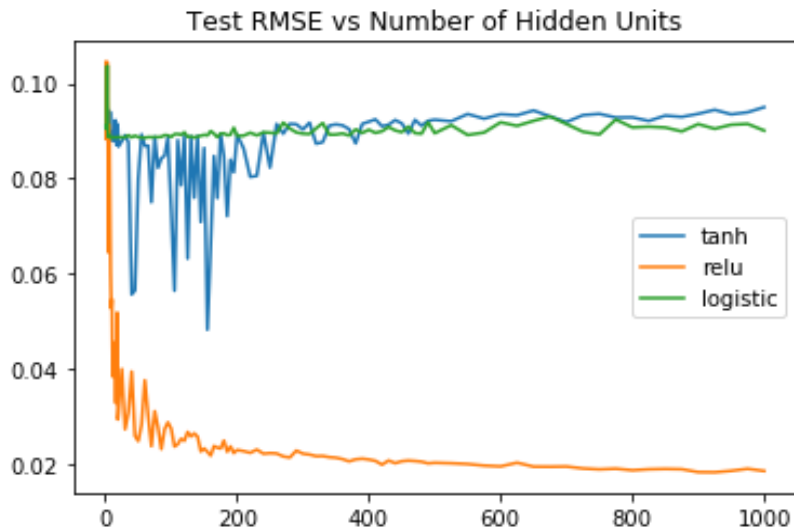


We can see that Relu performs the best among the 3 activation functions in terms of the Test RMSE.

<b>Best Combination</b>	relu with 180 hidden units
<b>Training RMSE</b>	0.017505487354359587
<b>Testing RMSE</b>	0.018471557023900253

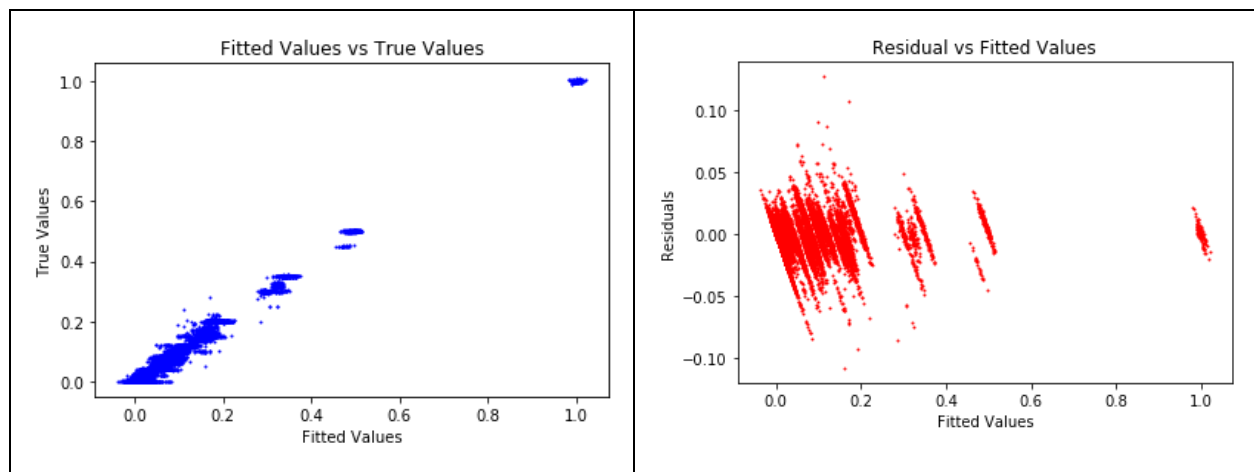


## Results with Adam Solver



We can see that Relu performs the best among the 3 activation functions in terms of the Test RMSE.

<b>Best Combination</b>	Relu with 925 hidden units
<b>Training RMSE</b>	0.013295878780614102
<b>Testing RMSE</b>	0.018379574776719924
<p>Fitted Values vs True Values</p>	<p>Residual vs Fitted Values</p>



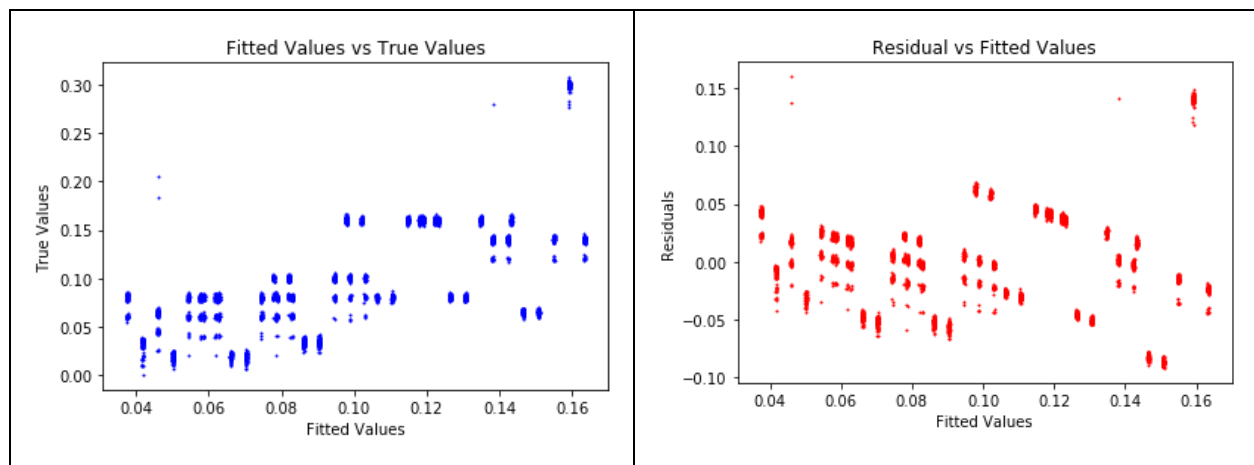
## (d) Predict for each of the workflows separately

### (i) Linear Regression for each workflow separately

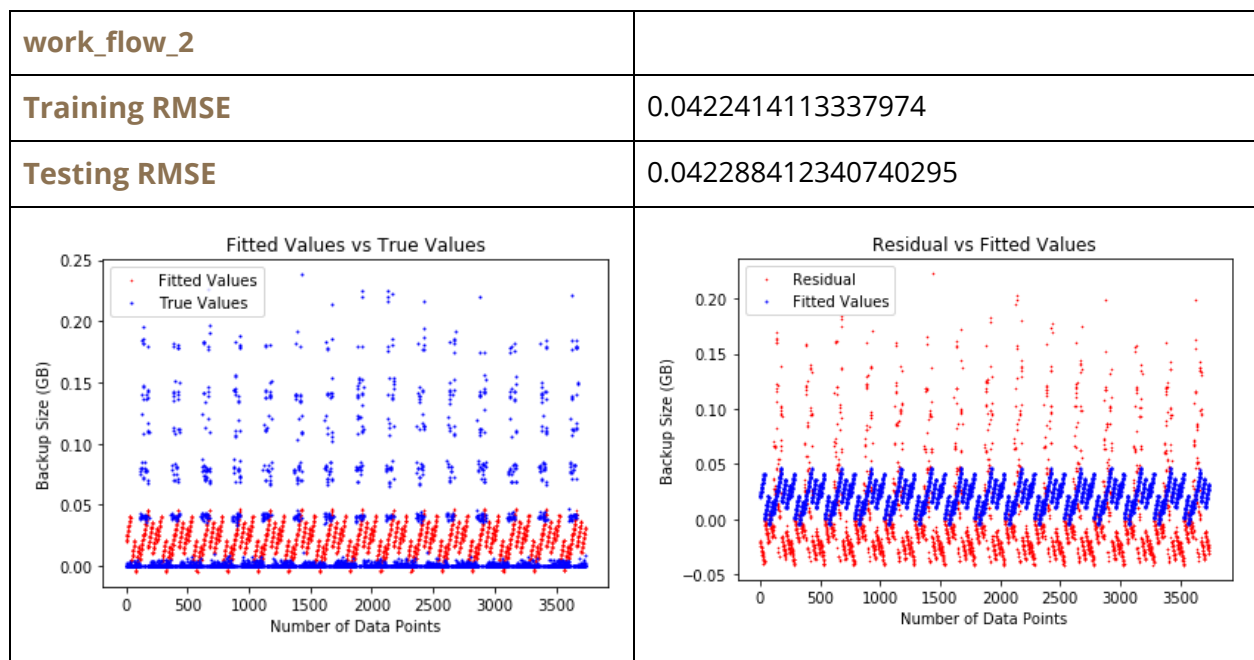
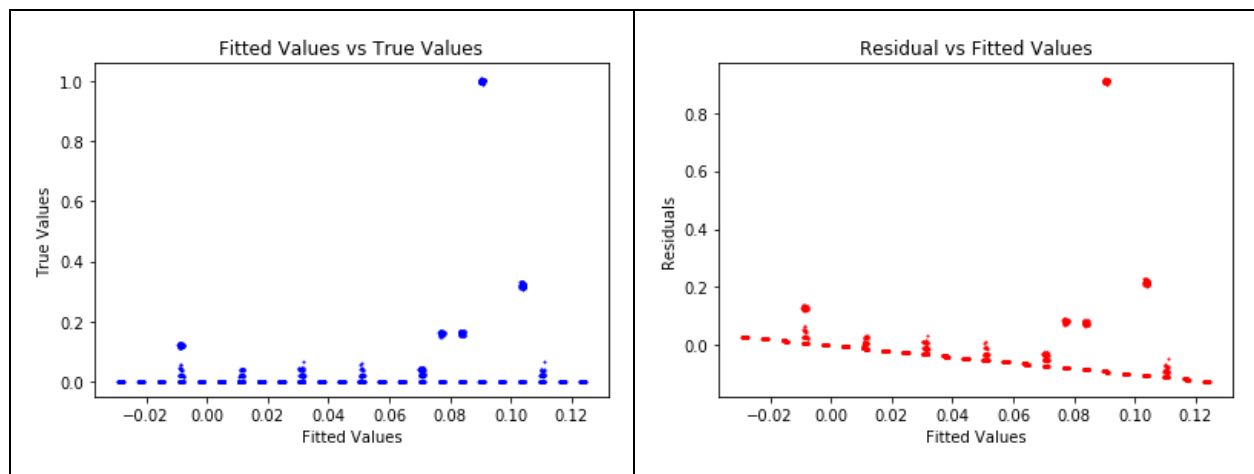
In this task we have used Linear Regression for each workflow separately to see how the data is fit for each workflow. This will give us a better insight into the data. The results for each workflow are given below.

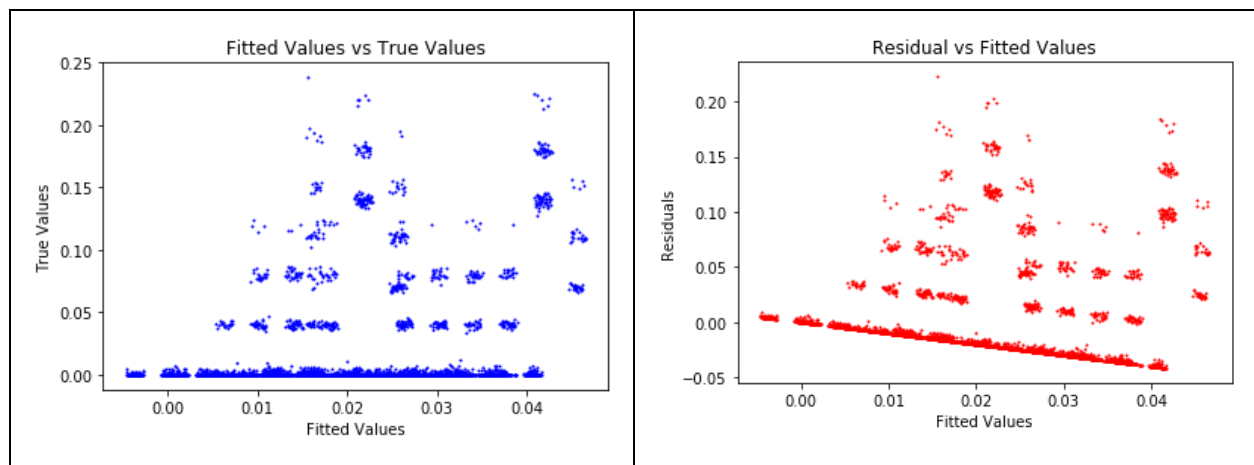
<b>work_flow_0</b>	
<b>Training RMSE</b>	0.04298421289948907
<b>Testing RMSE</b>	0.04304082938571748

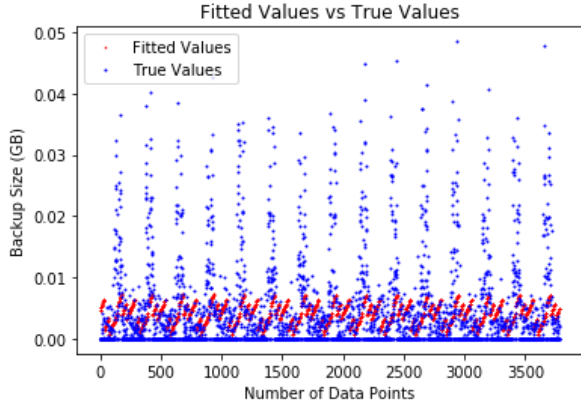
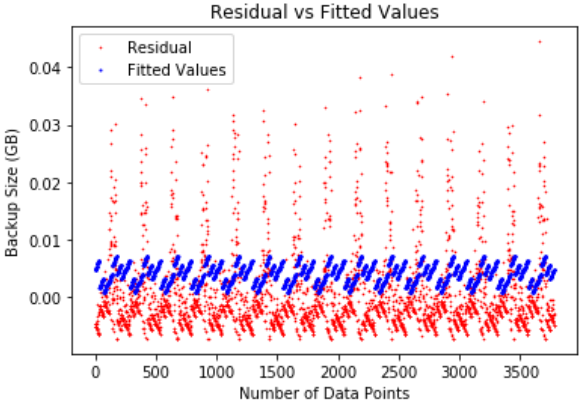
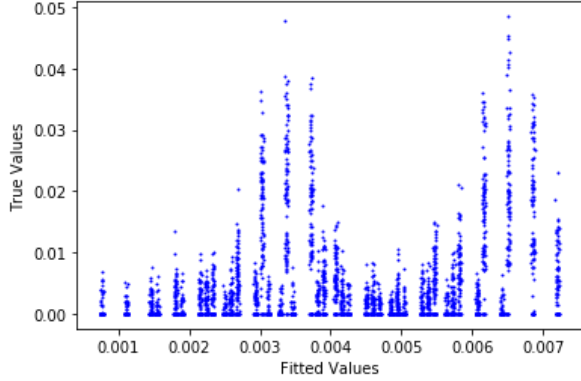
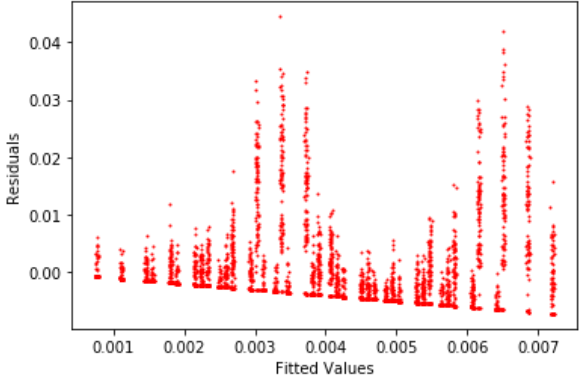


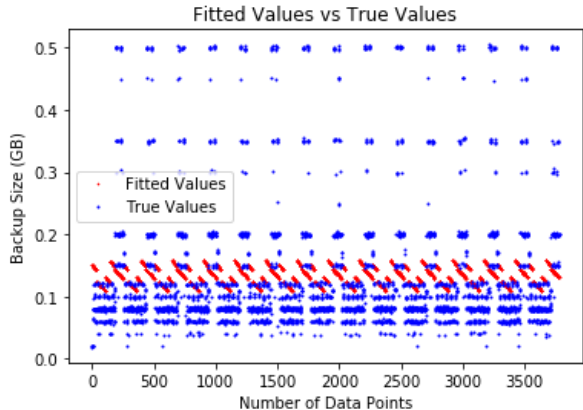
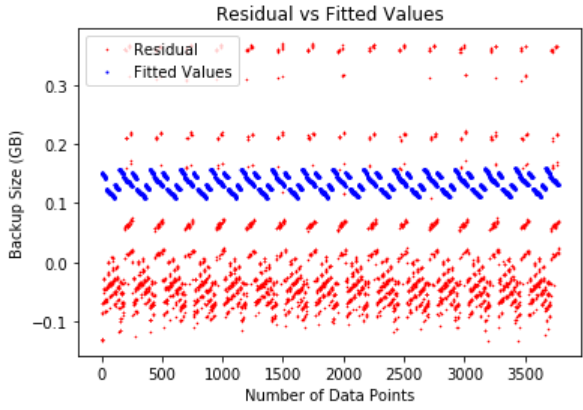
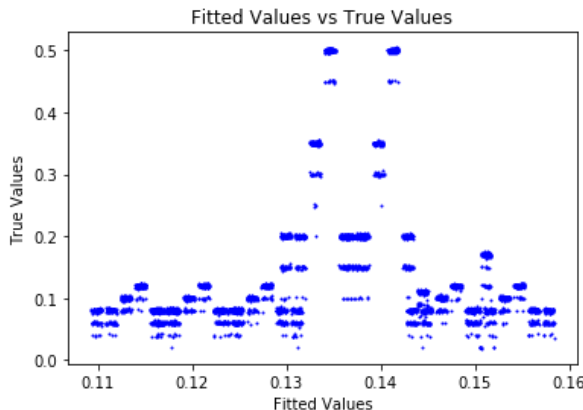
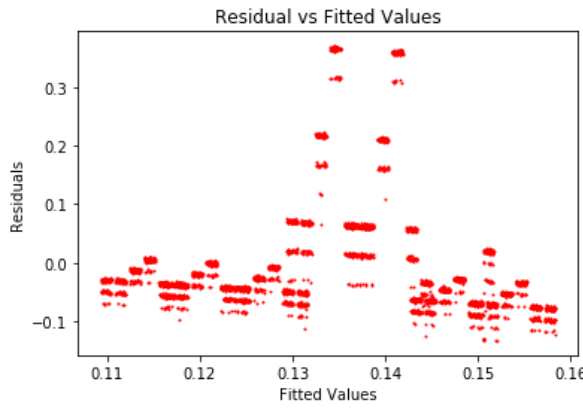


<b>work_flow_1</b>	
<b>Training RMSE</b>	0.15963884341674872
<b>Testing RMSE</b>	0.15982377944443607
<p>This scatter plot shows 'Backup Size (GB)' on the y-axis (0.0 to 1.0) and 'Number of Data Points' on the x-axis (0 to 3500). It contains two series: 'Fitted Values' (red dots) and 'True Values' (blue dots). The 'True Values' are clustered at 0.0, 0.2, and 0.3 GB, while 'Fitted Values' are clustered at 0.0 and 0.1 GB.</p>	<p>This scatter plot shows 'Backup Size (GB)' on the y-axis (0.0 to 0.8) and 'Number of Data Points' on the x-axis (0 to 3500). It contains two series: 'Residual' (red dots) and 'Fitted Values' (blue dots). The 'Fitted Values' are clustered at 0.0, 0.1, and 0.2 GB, while the 'Residual' values are scattered between 0.0 and 0.2 GB.</p>





<b>work_flow_3</b>	
<b>Training RMSE</b>	0.007117168892377209
<b>Testing RMSE</b>	0.007130278194550298
 <p><b>Fitted Values vs True Values</b></p> <p>This plot shows Backup Size (GB) (y-axis, 0.00 to 0.05) versus Number of Data Points (x-axis, 0 to 3500). It compares Fitted Values (red dots) and True Values (blue dots). The fitted values are generally lower than the true values.</p>	 <p><b>Residual vs Fitted Values</b></p> <p>This plot shows Backup Size (GB) (y-axis, 0.00 to 0.04) versus Number of Data Points (x-axis, 0 to 3500). It compares Residuals (red dots) and Fitted Values (blue dots). The residuals are mostly positive, indicating the model tends to underpredict.</p>
 <p><b>Fitted Values vs True Values</b></p> <p>This zoomed-in plot shows True Values (y-axis, 0.00 to 0.05) versus Fitted Values (x-axis, 0.001 to 0.007). It compares Fitted Values (red dots) and True Values (blue dots). The fitted values are generally lower than the true values.</p>	 <p><b>Residual vs Fitted Values</b></p> <p>This zoomed-in plot shows Residuals (y-axis, 0.00 to 0.04) versus Fitted Values (x-axis, 0.001 to 0.007). The residuals are mostly positive, indicating the model tends to underpredict.</p>

<b>work_flow_4</b>	
<b>Training RMSE</b>	0.10301788553573978
<b>Testing RMSE</b>	0.1030957786754087
	
	

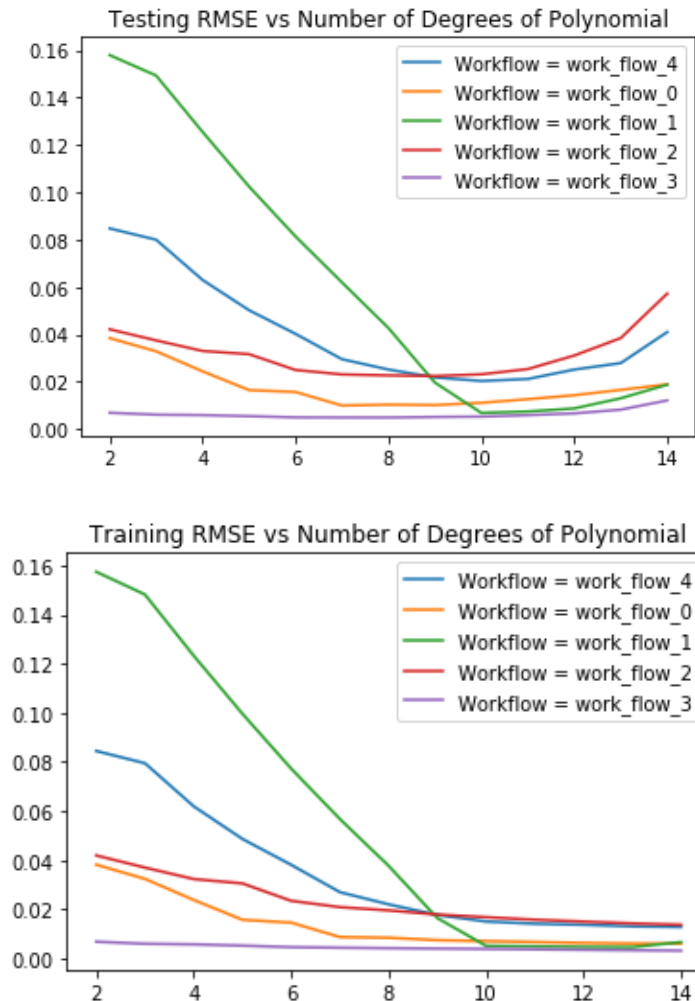
### Q) Explain if the fit is improved?

We observe that in most of the workflows the test RMSE is lower than the earlier model. The reason maybe because of the presence of the large amount of outliers, the model is not able to fit a very good function as opposed to the case when we run the model separately on each workflow.

---

## (ii) Polynomial functions

In this task we have used polynomial functions of variable degree for the variables.



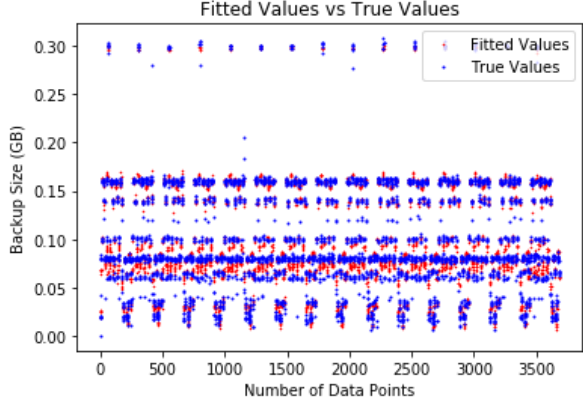
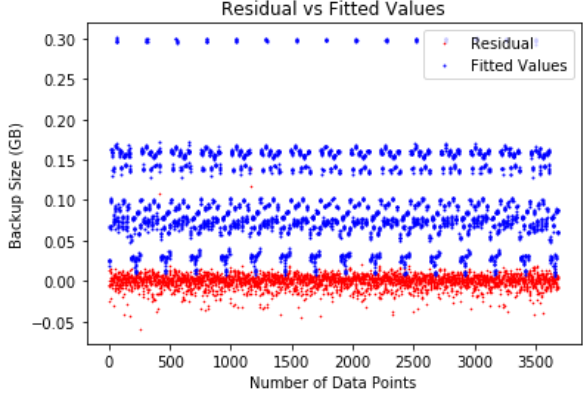
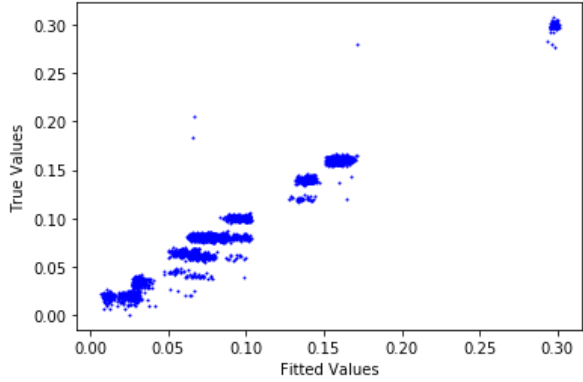
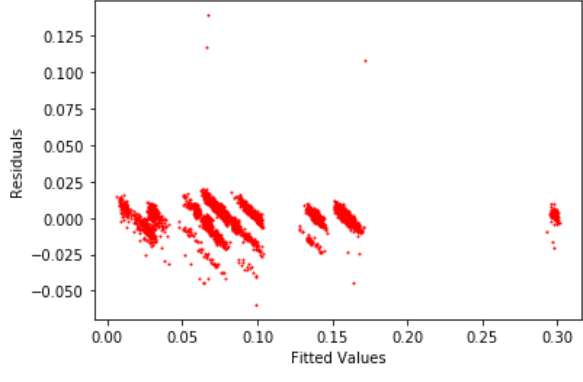
**Q) Can you find a threshold on the degree of the fitted polynomial beyond which the generalization error of your model gets worse?**

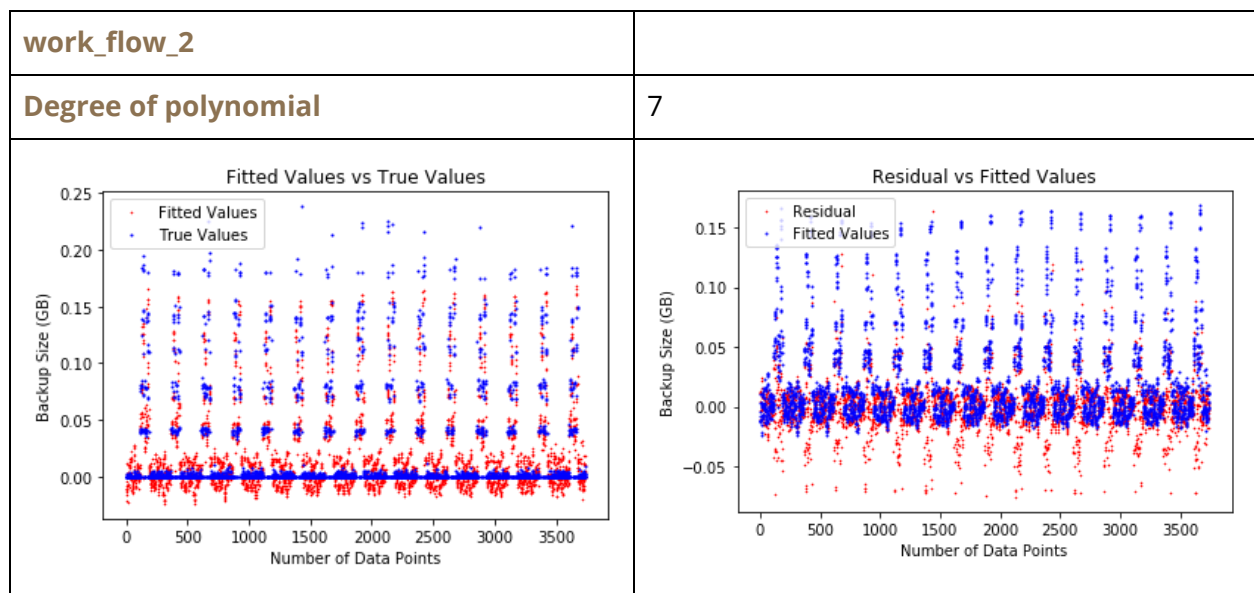
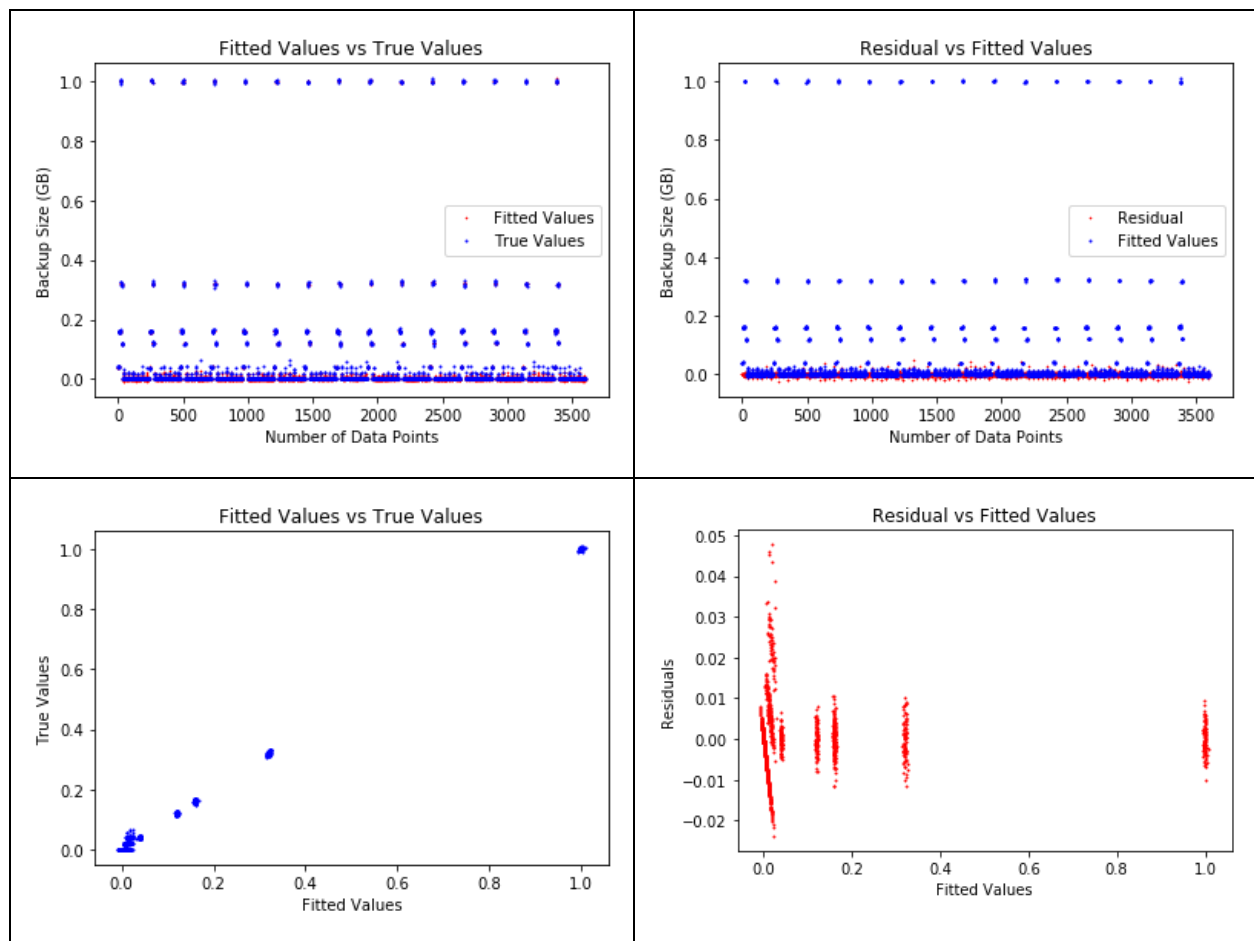
As the data is not linearly separable, we get better results by using polynomial functions for our data. As the degree of the polynomial is increased, the curve can better fit the data thus lowering the test and train RMSE.

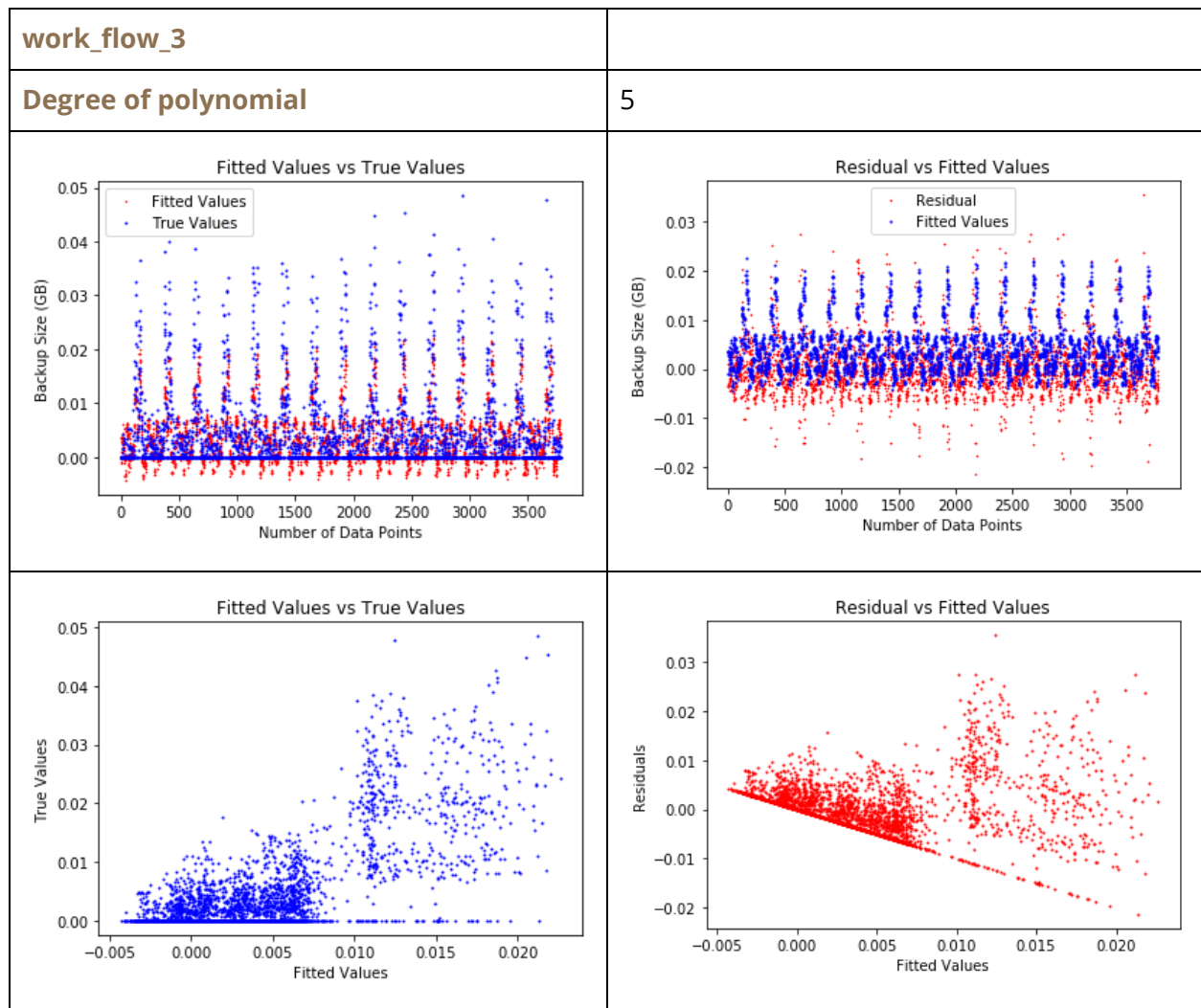
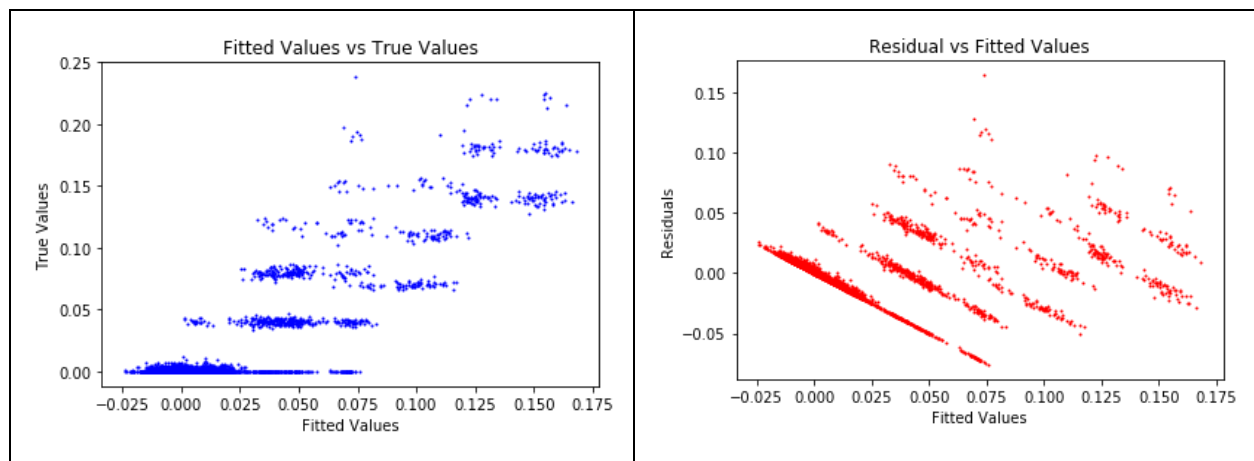
However, on observing the above graphs, for each workflow we see that after a certain degree of polynomial, the testRMSE actually starts to increase even though the trainRMSE is

decreasing. This is a classic example of overfitting where a curve of higher degree polynomial overfits the data resulting in decreased training but increase testing error.

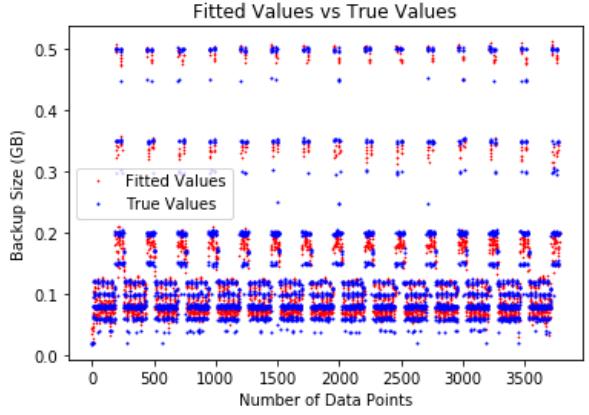
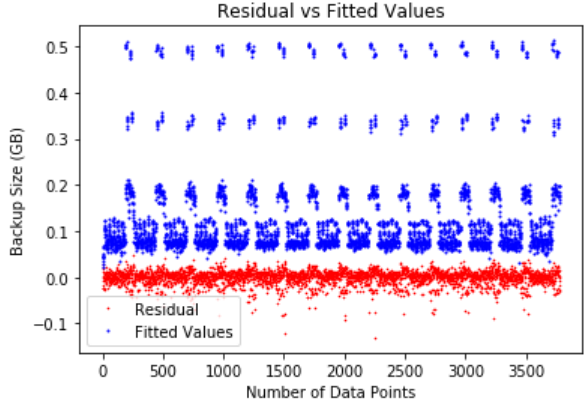
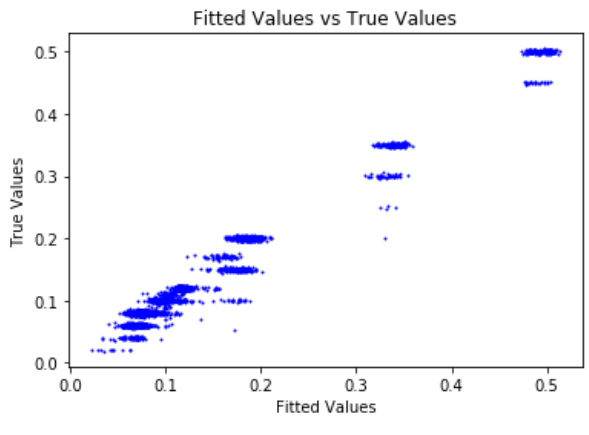
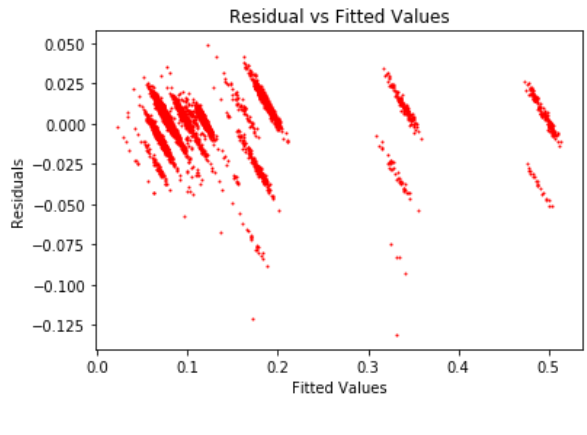
At a certain degree of polynomial for each workflow, we have found the fit of the data for that degree as shown below for each workflow.

<b>work_flow_0</b>	
<b>Degree of polynomial</b>	7
	
	
<b>work_flow_1</b>	
<b>Degree of polynomial</b>	10







work_flow_4	
Degree of polynomial	10
	
	

**Q) Can you explain how cross validation helps controlling the complexity of your model?**

A typical ML model generally suffers from high bias and/or high variance problem. In case of high variance the performance of the algorithm largely depends on the input data and hence it varies a lot with different data if it has a high variance. High bias basically limits the ability of the model to learn complex functions. Cross validation helps to counter these 2 problems upto extent. In each fold of the Cross validation, the model is exposed to slightly different data, taken as a subset from the original dataset thus trying to control the variance and bias upto some extent and also help in reducing overfitting. Doing a cross validation helps to reduce the complexity of the model thereby reducing the overfitting!!

---

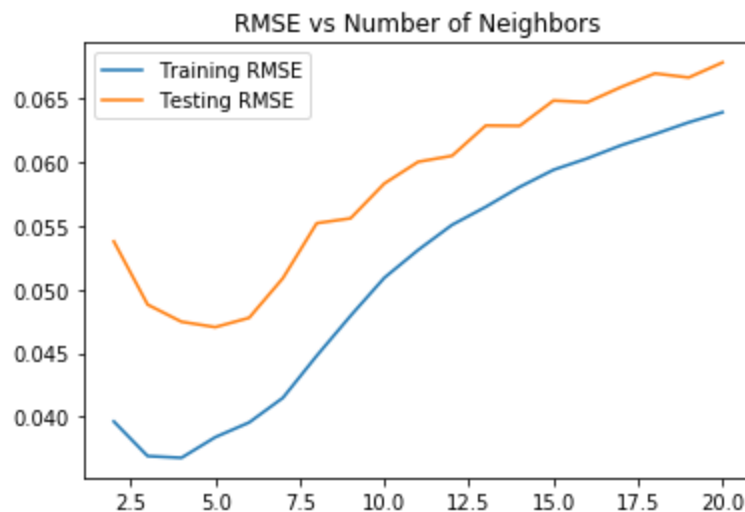
## (e) k-nearest neighbor regression

For this task we have used the **KNeighborsRegressor** from the sklearn package.

It is a regression based on k-nearest neighbors. The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

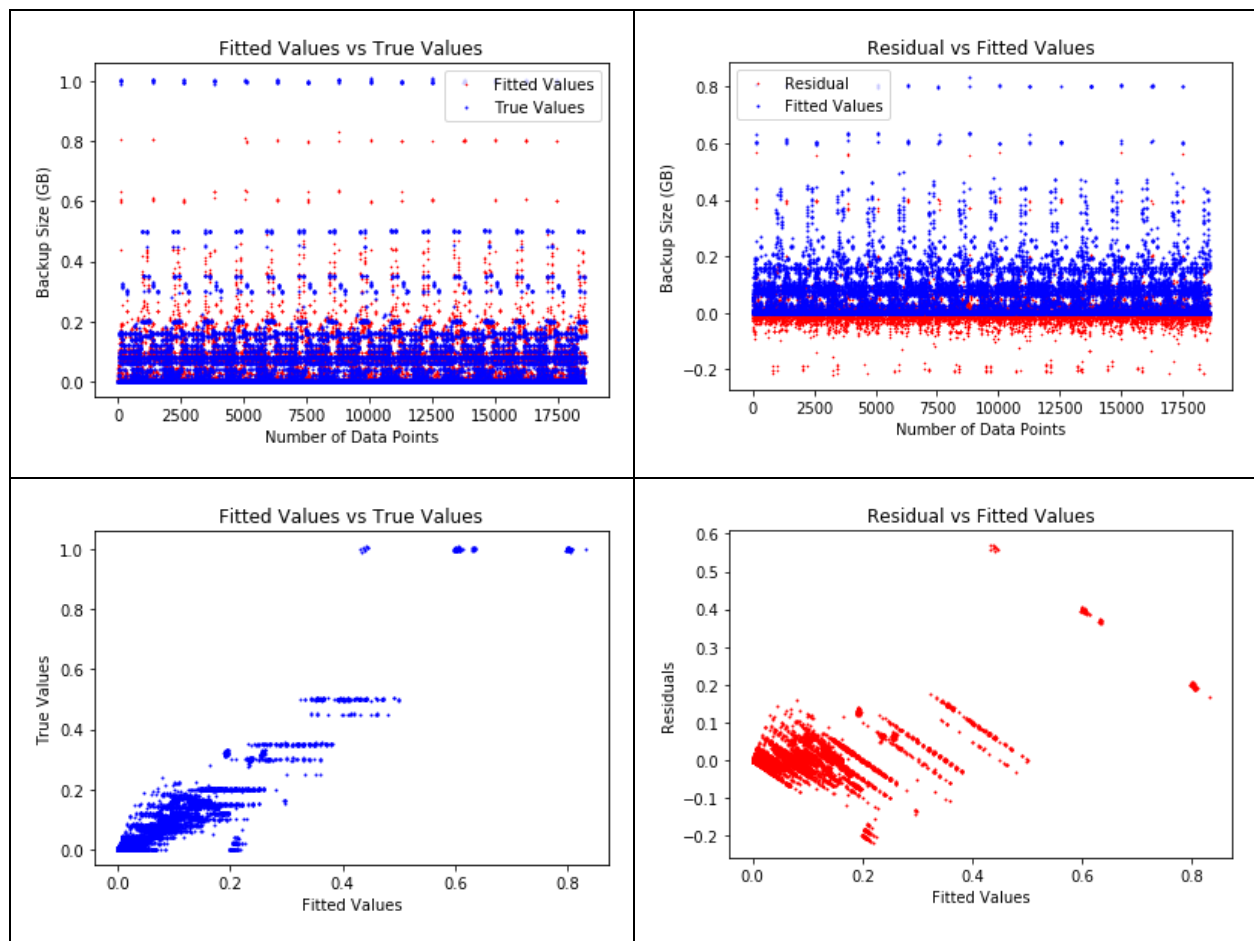
### With label encoding

We have used scalar encoding for the data before passing it to the Knn regressor. The parameter which we have varied for the Knn regressor is the number of neighbors. We sweep through the number of neighbors to find the best parameter.



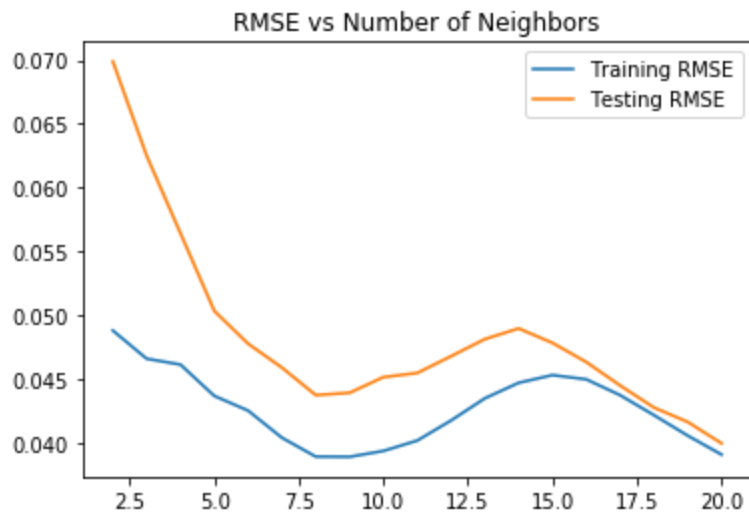
We see that after a point(here k=5), the testing and training RMSE start to increase and continue to increase till k=20.

Best Parameter	For 5 Neighbors
Training RMSE	0.03855601248919903
Testing RMSE	0.04745824054008073

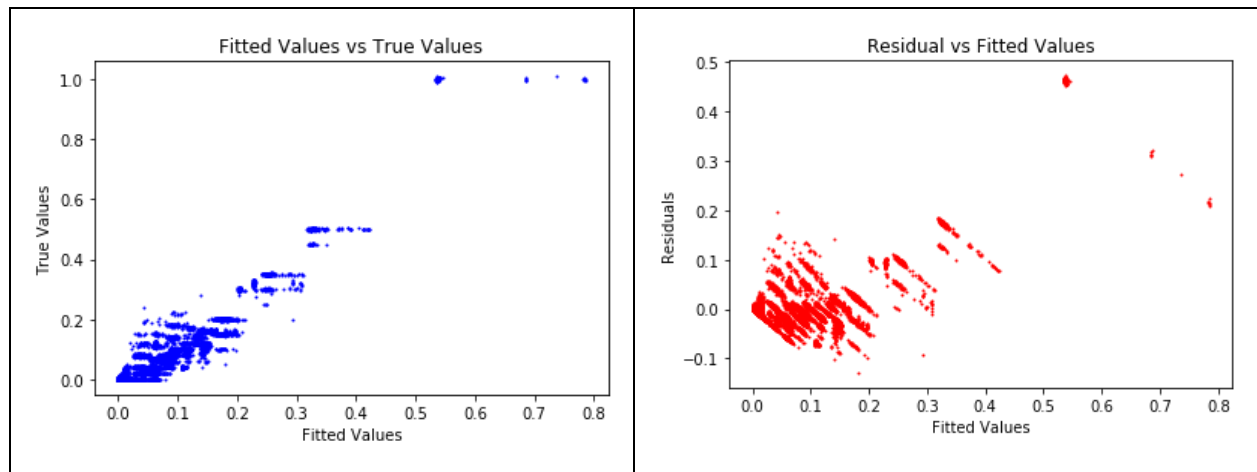


### With one hot encoding

Here we have one-hot encoded the data before passing to the Knn regressor. We get slightly better results in this case than the previous case. This is because any learner based on standard distance metrics (such as k-nearest neighbors) between samples will get confused without one-hot encoding.



<b>Best Parameter</b>	For 20 Neighbors
<b>Training RMSE</b>	0.03909469600746857
<b>Testing RMSE</b>	0.03973063629505254

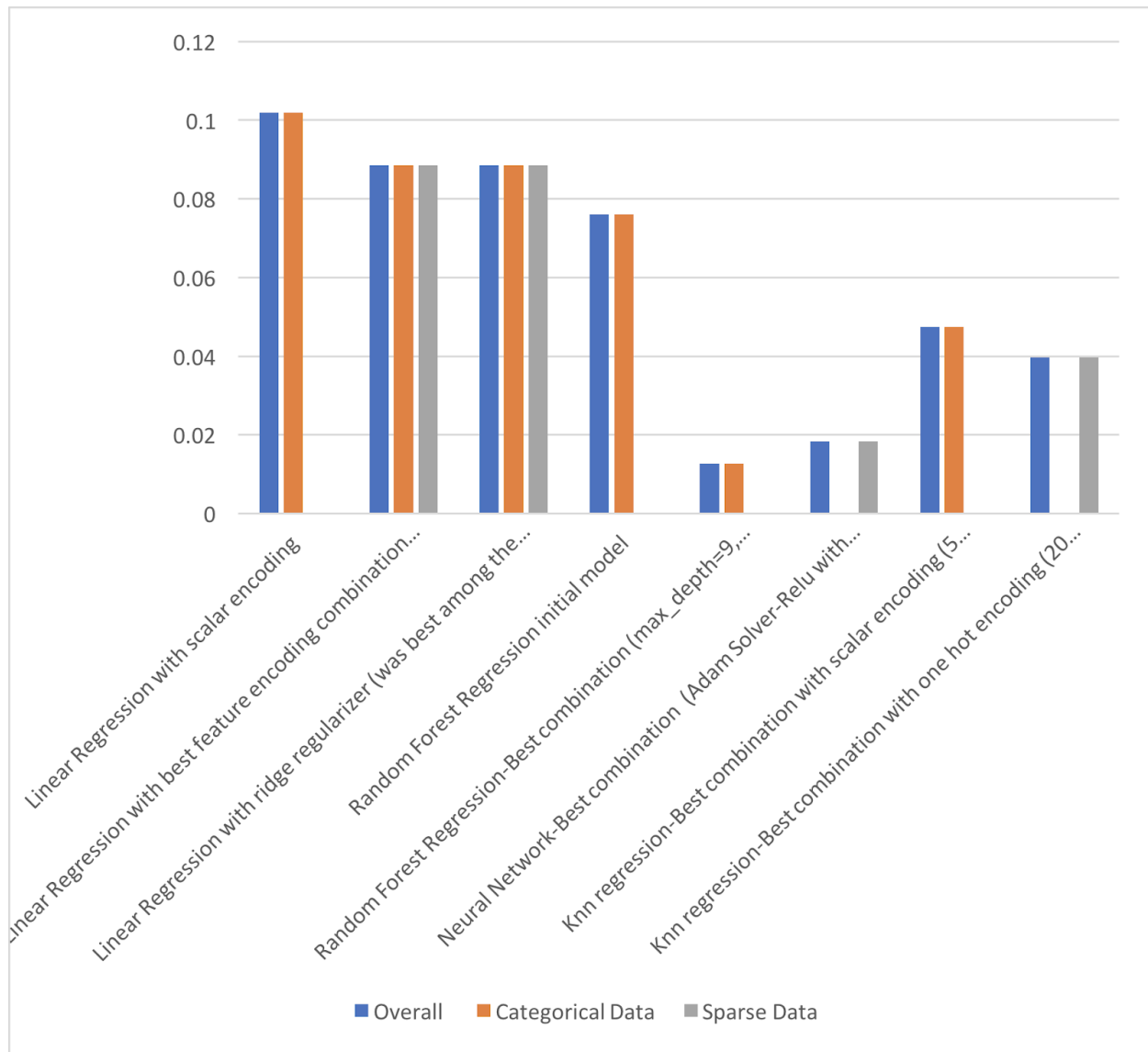


---

### III) Comparison and Analysis

We will first get the results from all the models into a single table for better comparison and analysis. We even visualize these results to get a better understanding.

Model	Details	Test RMSE
Linear regression	with scalar encoding	0.10186627832709541
Linear regression	with best feature encoding combination (scalar + one hot encoding)	0.0884528308663724
Linear regression	with ridge regularizer (was best among the 3 regularizers)	0.08843264762881899
Random Forest Regressor	initial model	0.07611235229518903
Random Forest Regressor	Best combination (max_depth=9, n_estimators=60, max_features=5)	0.012628837803137626
Neural Network	Best combination (Adam Solver-Relu with 925 hidden units)	0.018379574776719924
Knn regressor	Best combination with scalar encoding (5 neighbors)	0.04745824054008073
Knn regressor	Best combination with one hot encoding (20 neighbors)	0.03973063629505254



- Among all the models, we observed that the Random Forest regressor with optimal parameters gives the lowest test RMSE
- For categorical data, the best performer was again the Random forest regressor. This is expected as Decision tree models can handle categorical variables without one-hot encoding them and multiple such decision trees forming a forest can predict the data with better accuracy resulting in low RMSE.
- Amongst the one-hot encoded methods of linear regression, knn regression and neural networks, the neural network model is the best performer. One-hot encoded

---

data is basically sparse data, hence we can conclude that the neural network model can handle sparse data very well.