EE219 Project 1 – Regression Analysis

– *Shubham Mittal* (104774903), *Swati Arora* (404758379), *Anshita Mehrotra* (904743371)

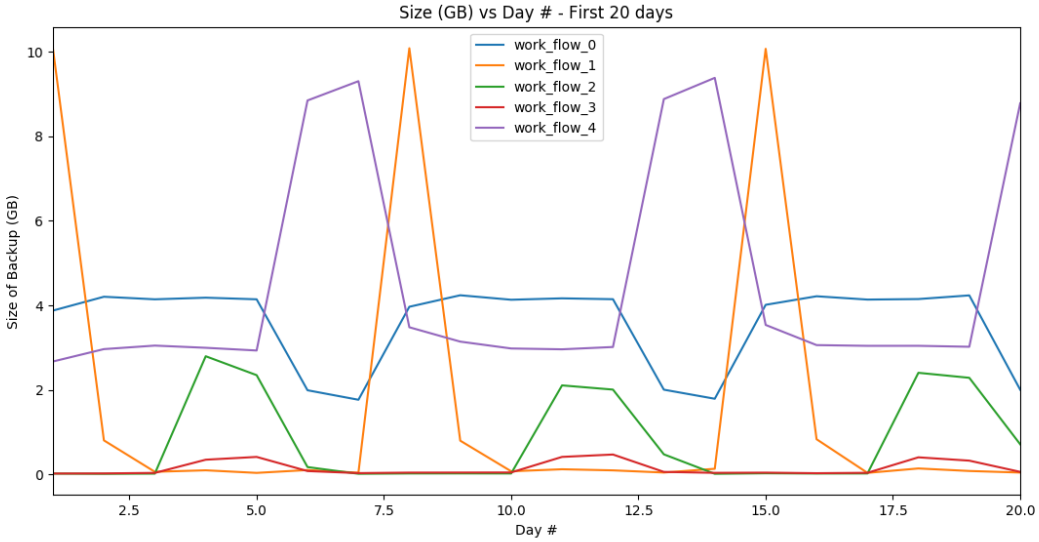
Network backup Dataset

We used Python and the associated data science libraries for this project. Since the target variable i.e. Size of Backup (GB) had small values, we normalized the data to bring the feature vector within the range of 0 – 1. Since the input dataset had categorical string values, we used both integer mapping and one-bit hot encoding convert the string values to numeric values.

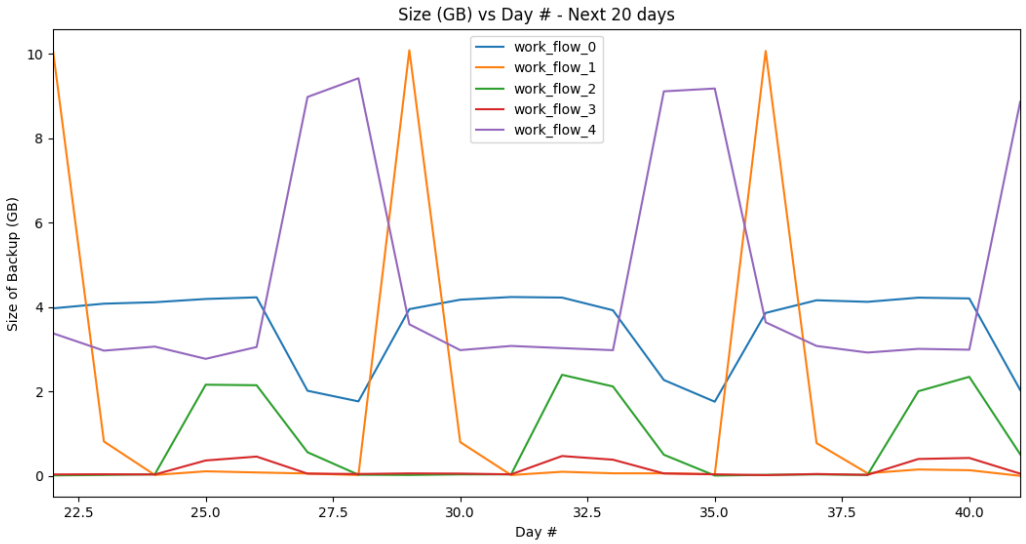
PROBLEM 1

We first computed the *Day #* using the *Week #* and *Day of Week* variables. We grouped the given dataset by workflows and computed the aggregate size of backup for each workflow grouped by *Day #* (We tried grouping by *File Name* as well, but did not see any identifiable pattern in the plots).
The plots of *Size of Backup (GB)* vs the *Day Number* are observed as follows:

a. For the first 20 days –



b. For the next 20 days –

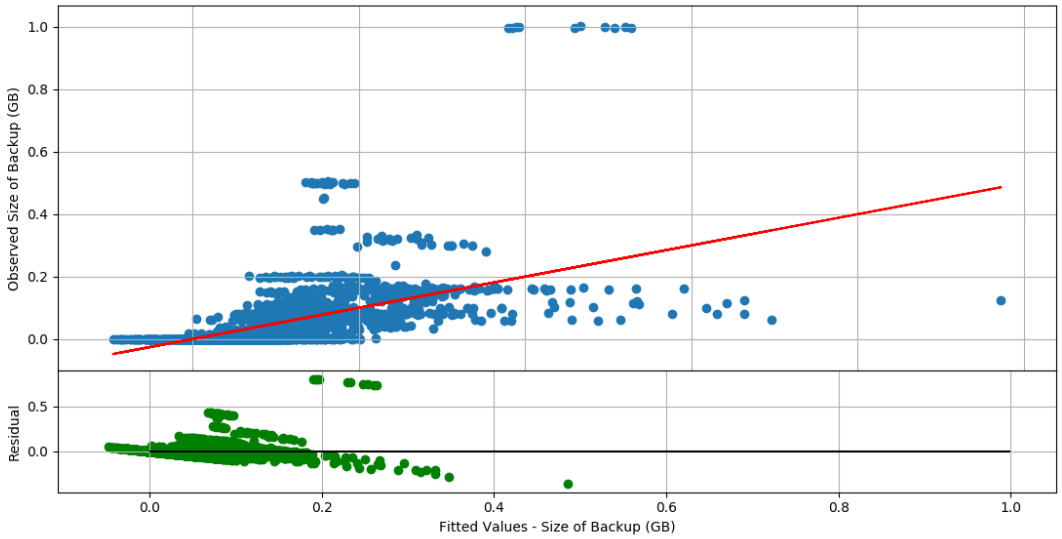


The plots above clearly show a pattern being repeated for every workflow. We observe that the Size of Backup (GB) curve repeats itself after every 7 days (approximately). The same pattern is repeated for the next 20 days as well.

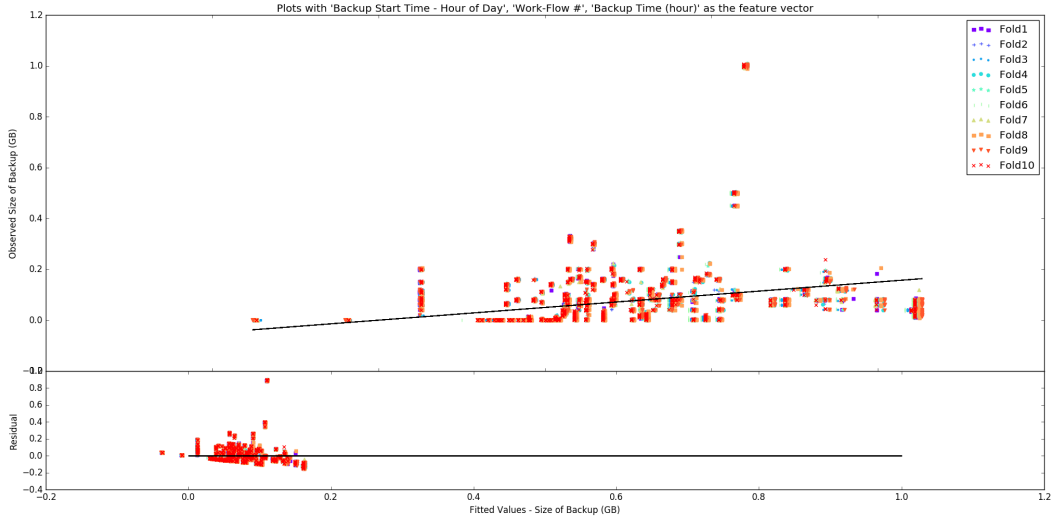
PROBLEM 2

a. Linear Regression Model:

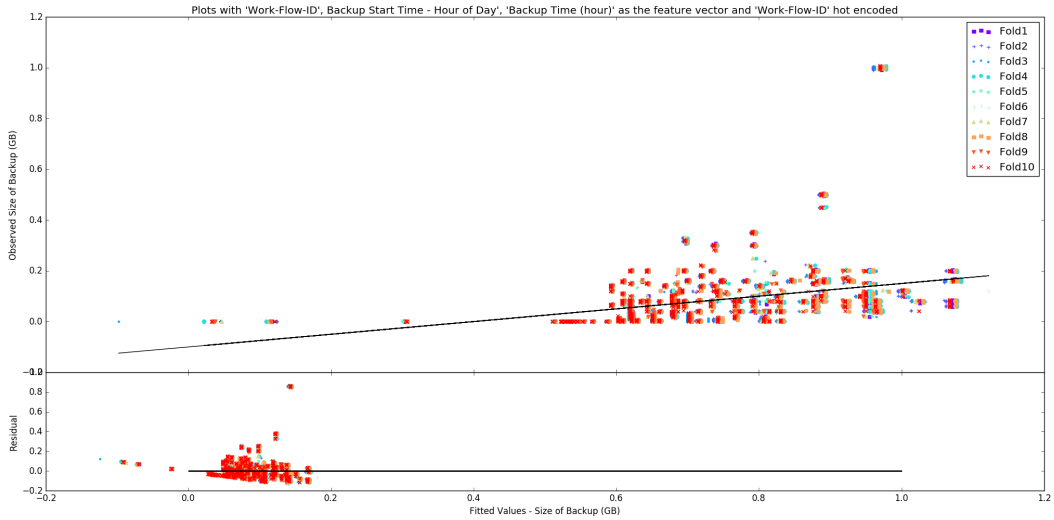
We first separate out the *Size of Backup (GB)* as the target variable and using all the other inputs as features (converted to the appropriate numeric values), we observe the spread of data in the *Observed vs Fitted plot* (below) is suboptimal. The observed root mean square error in a single fold validation is **0.0890**



We next perform a 10-Fold cross validation over the data. Before splitting the data into 10 sections, we shuffle the input data to remove any pre-existing bias. We cherry-pick the input features based on the estimated coefficients/weights of individual features obtained after fitting the model in a linear regressor. The RMSE obtained for this model is **0.09513**



In the above plot, *Work-Flow #* was one of the input feature vectors which was a numeric vector created by converting *Work-Flow-ID* using integer mapping. Instead, we not use one-bit hot encoding for the same parameter and observe a slightly better fit and a reduced error. Root mean square error for 10-fold validation with a one-bit hot encoded workflow-id is **0.0926**



Note: The above plots are made after superimposing the data points of all the 10 folds together in one graph.

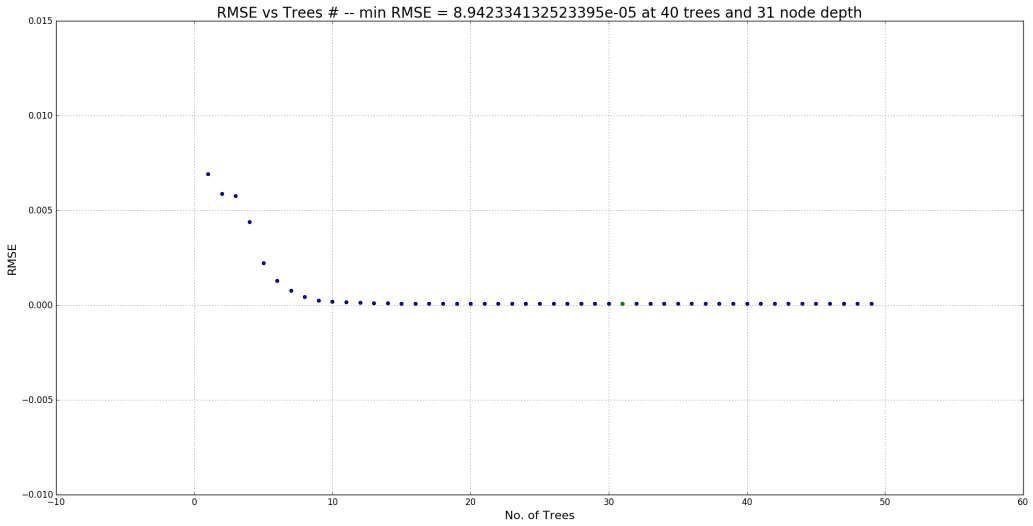
The most important feature inputs for linear regression were *Backup Start Time – Hour of Day*, *Backup Time (hour)* and *Work-Flow-ID*. We found out that these were the best inputs based on the coefficient strength/feature weight computed after fitting the model over all the features. The table below relates the features and their corresponding strengths. Any coefficient with a negative value works against the model.

Sl No.	Features	Estimated Coefficients
1.	Week #	-0.028557
2.	Day #	-0.116975
3.	Work-Flow #	0.027066
4.	File #	-0.082720
5.	Backup Time (hour)	0.908964
6.	Backup Start Time - Hour of Day	0.030172

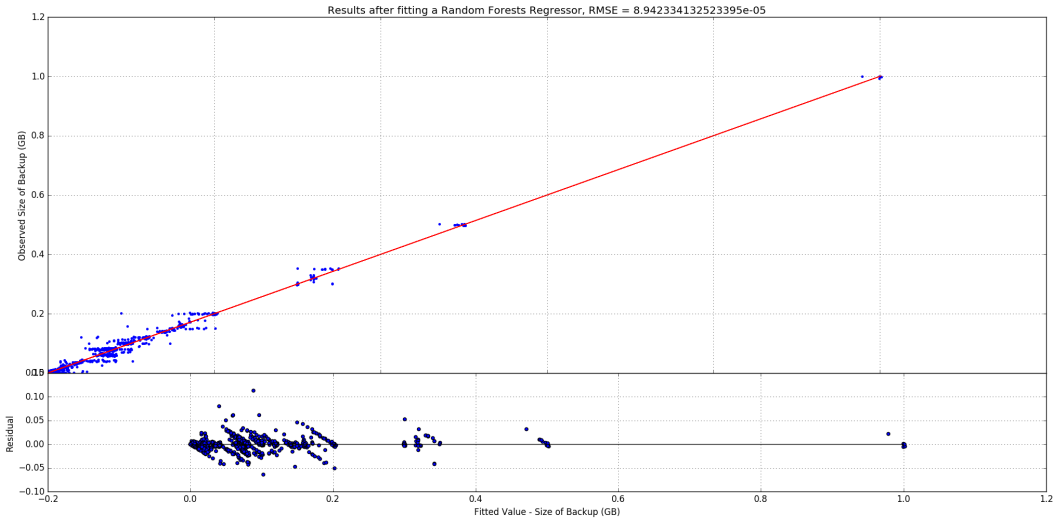
b. Random Forests Regression

After cleaning and normalizing the data, we fit a random forest regressor to the model with number of trees set to 20 and depth of each tree equal to 4. We observe the root mean square error to be **0.0044191**.

Since the efficacy of random forests highly depends on the number of tree and the tree depth, we tune both these parameters, increasing both in steps of one. We observe that the minimum root mean square error is found when we have 40 trees with a node depth of 31. After that, the error becomes almost constant and has no effect on changing the depth or the tree count. The plot observed is as follows:



The root mean square value of **8.9423e-05** indicates that random forests accurately fit the data and outperform linear regression. The observed vs fitted, plus the residual plot is shown below:



We found out that every input feature had a positive weight towards estimating the model, with the most powerful feature being the Backup Time (hour). Also, doing the hot-bit encoding resulted in a lower error than a simple integer mapping. The weights for the features are given below:

Sl No.	**Features**	**Estimated Coefficients**
1.	Week #	0.011646
2.	Work-Flow #	0.127258
3.	Backup Time (hour)	0.500383
4.	Backup Start Time – Hour of Day	0.059050
5.	Day of Week – Monday	0.156260

6.	Day of Week – Tuesday	0.013806
7.	Day of Week – Wednesday	0.004947
8.	Day of Week – Thursday	0.004682
9.	Day of Week – Friday	0.004932
10.	Day of Week – Saturday	0.060543
11.	Day of Week – Sunday	0.056495

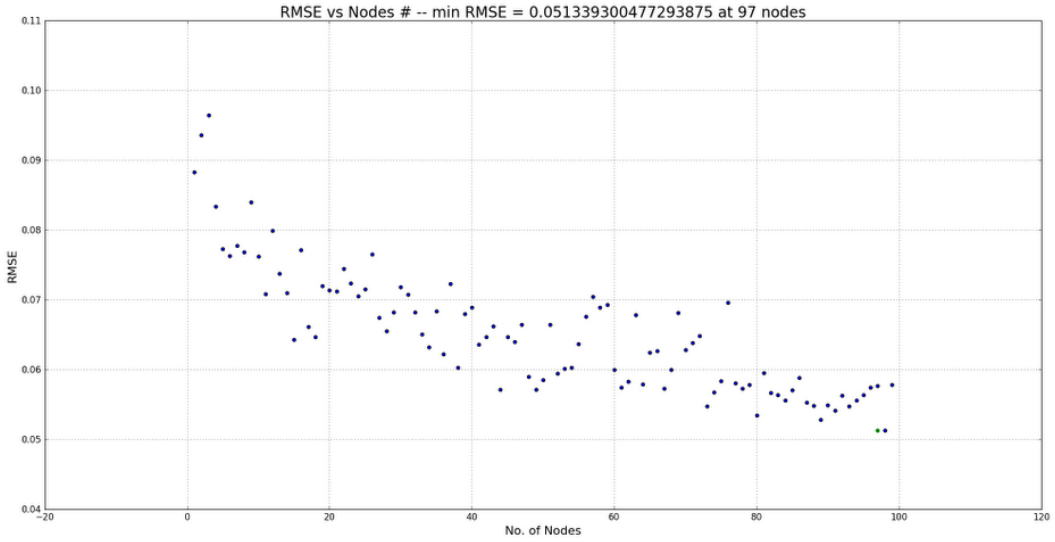
The Random Forests regressor showed interesting patterns that were not seen by linear regression. Firstly, it was able to successfully use all the input feature vectors to develop the model. Secondly it was able to accurately predict the data with a minimal root mean square error. Thirdly we observed that the model's performance can be greatly varied by tuning in the hyperparameters (like number of trees, tree depth etc.)

An interesting insight we observe is that the most important/powerful features remain the same for both the Linear Regressor as well as the Random Forests Regressor. *Backup Time (hour)* has the highest weight for both the models, followed by *Work-Flow #* and *Backup Start Time – Hour of Day*.

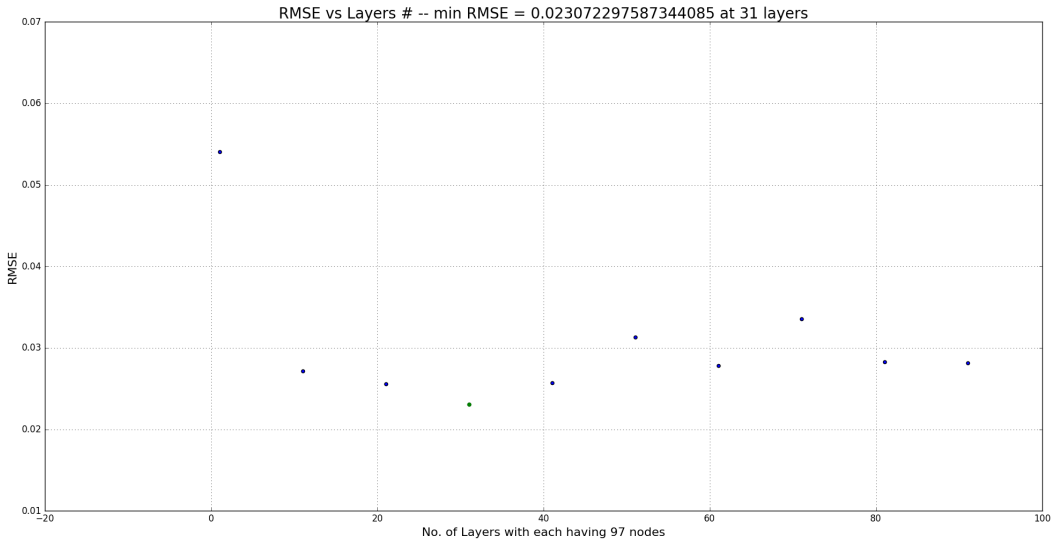
c. Neural Network Regression

Neural Networks are strong classifiers/regressors because of its inherent ability to model nonlinearities. However, its performance heavily depends on two parameters – the number of nodes in each layer and the number of layers. We first determine the optimum hyperparameters based on the minimum root mean square value achieved and then plot the residual graphs.

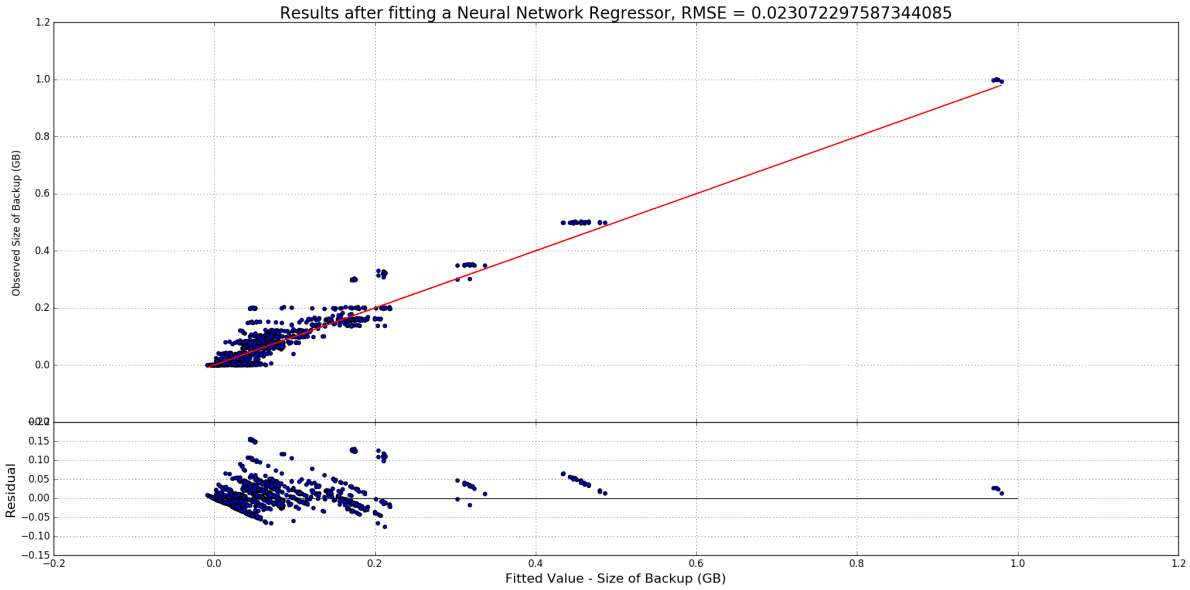
After cleaning and processing the data and using one-bit hot encoding for *Day of Week*, we first tune the number of nodes in a single layer and obtain the below plot.



Based on the plot above, the minimum root mean square error for a neural network of one layer occurs with 97 nodes and is equal to **0.05134**. We next tune the number of layers in the neural network keeping the number of layers fixed at 97 and obtain the following plot:



We observe that the root mean square error decreases as we increase the number of layers with each layer having 86 nodes. The minimum error reported is **0.02307** and in found with a model having 31 layers and 97 nodes each. The following plot represents the results obtained and shows the efficiency of a neural network to accurately model the data.



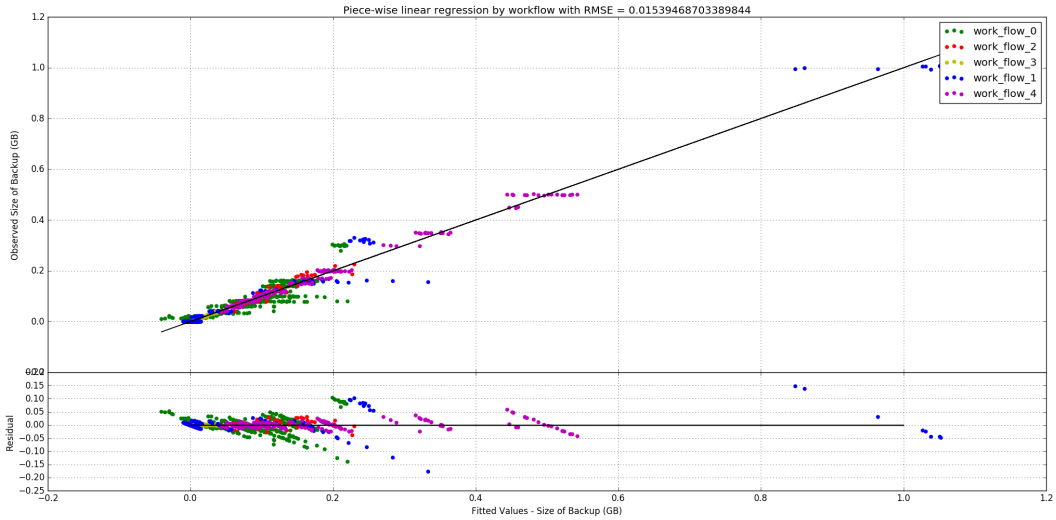
We see that although neural nets outperform linear regression (by inherently modelling the nonlinearities) they are unable to match the performance of random forests for the given data. This may be due to the few training samples and/or high dimensional categorical feature vectors. We observe

that on an average the performance of neural network regressor models improve on having more number of nodes. Given enough training data a neural net model can accurately predict all the test samples with only one layer having a huge number of nodes. However, a super wide network may result in an overfitting and thus loss of generalization.

PROBLEM 3

Piece-Wise Linear Regression

We perform piece-wise linear regression on the data grouped by workflow id. We observe that integer mapping for *Day of Week* performs better than one-bit hot encoding. This could be because of the added complexity of a higher dimensional feature vector created as a result of one-bit hot encoding.



The net root mean square error for this model is **0.0154** which is lower than a simple linear regression fit of the data where the observed RMSE was 0.0926. The reason a piece-wise fit outperforms the net linear fit is because in this case, the model is able to identify and fit on the patterns which are different for different workflows. A simplified linear regression is unable to distinguish between the patterns as all the patterns are superimposed over one another.

PART II

Boston Housing Data Set

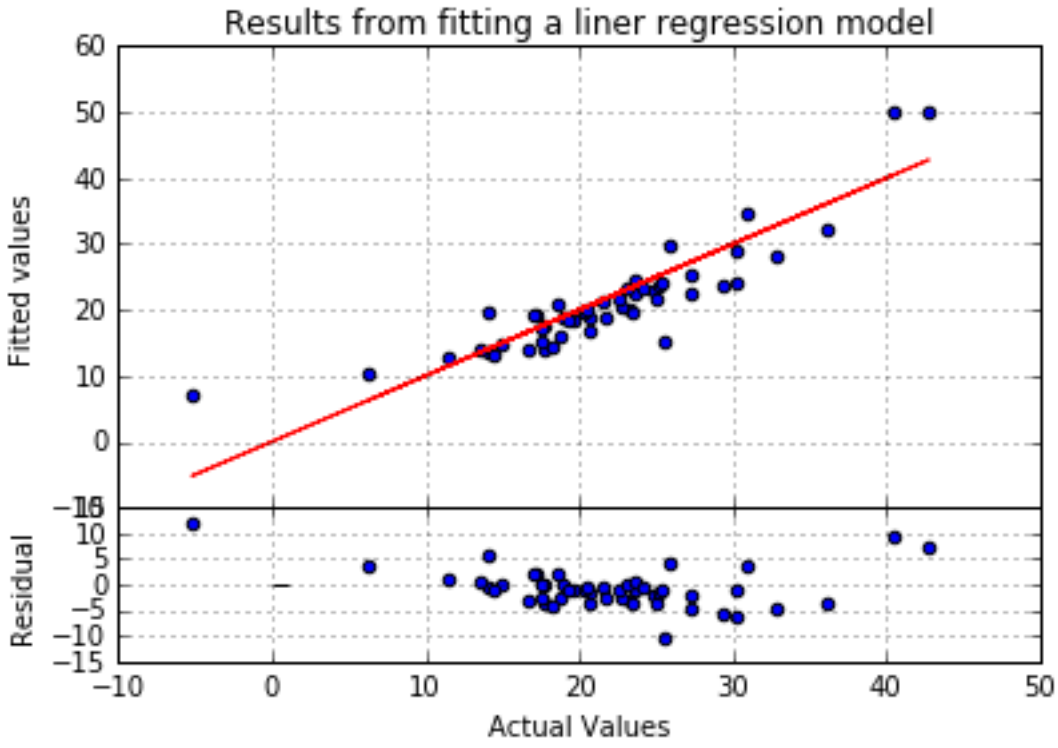
This dataset comprises of housing values in the suburbs of Boston and we have to predict Median value of owner occupied home in thousands of dollars.

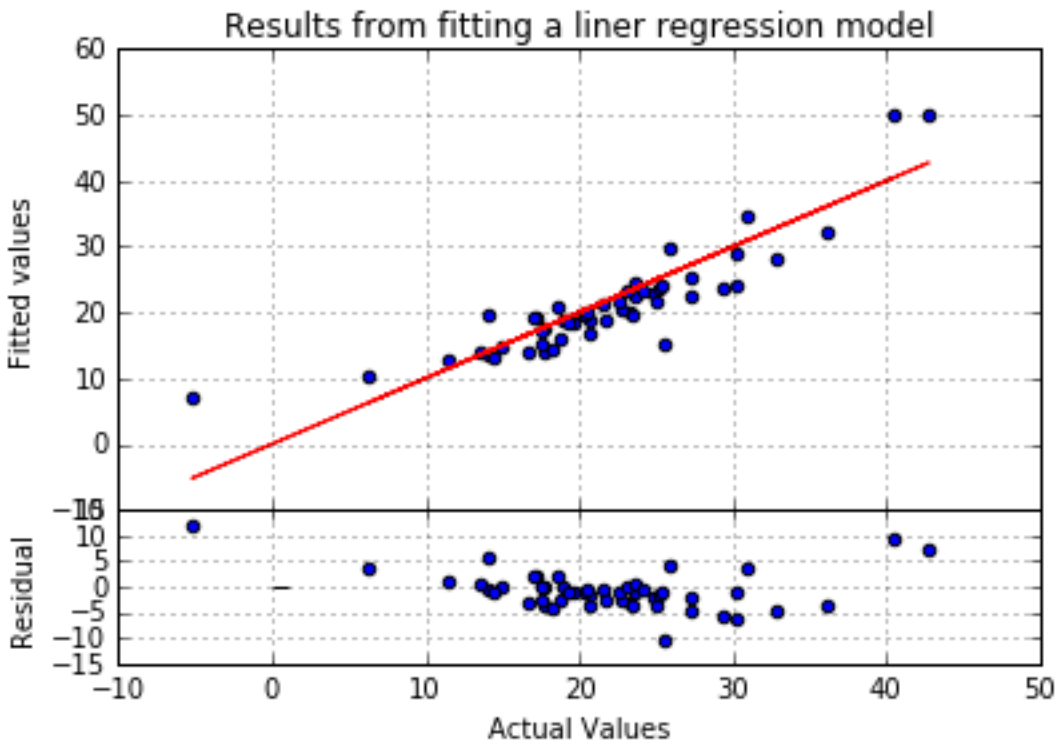
PROBLEM 4

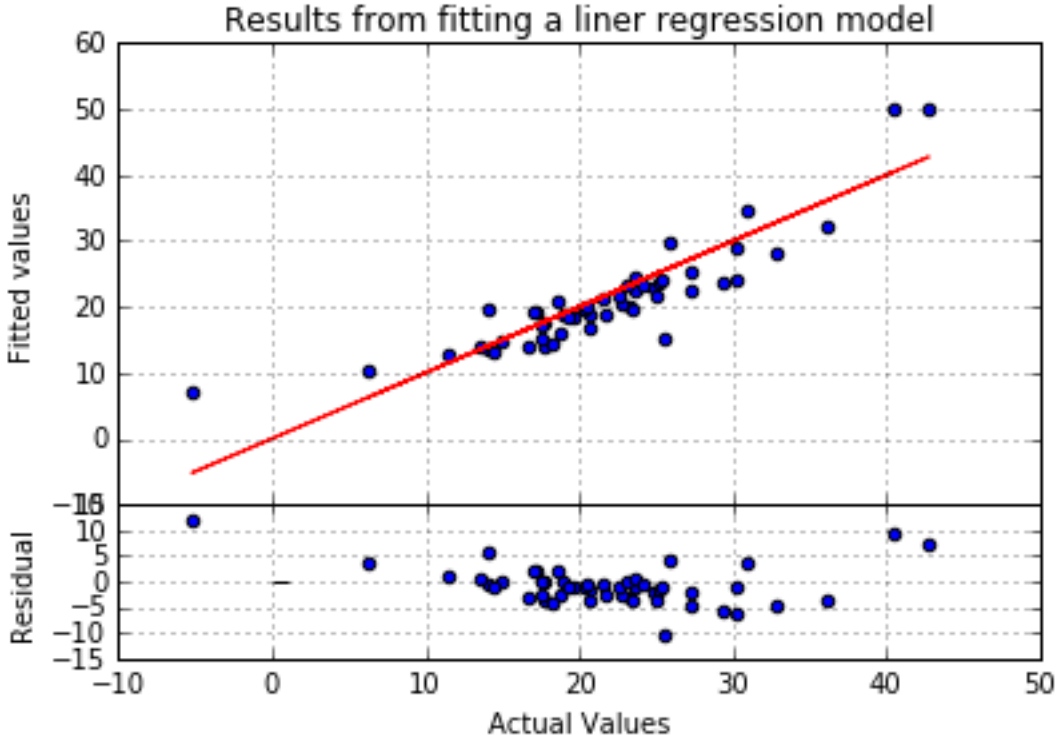
To predict the median value of owner-occupied home, we are employing linear regression model and polynomial regression model.

a. Linear Regression Model:

We first load the data and separate out the *MDV(Median Value for a house $1000's)* as the target variable and using all the other inputs as features which are converted to the appropriate numeric values. We observe the spread of data in the *Observed vs Fitted plot* (below) is suboptimal. The observed root mean square error in a single fold validation is **2.458.**




Results from fitting a liner regression model
Fitted values
Residual
Actual Values


Results from fitting a liner regression model
Fitted values
Residual
Actual Values

We next perform a 10-Fold cross validation over the data. Test and Training set split is 10% and 90% respectively. Before splitting the data into 10 sections, we shuffle the input data to remove any pre-existing bias. We cherry-pick the input features based on the estimated coefficients/weights of individual features obtained after fitting the model in a linear regressor and predict the value for test dataset. The RMSE obtained for this model is **3.888**

We also observe that some features such as ZN, NOX, RM, DIS, RAD, PTRATIO, B and LSTAT are more significant than other features in the feature space.

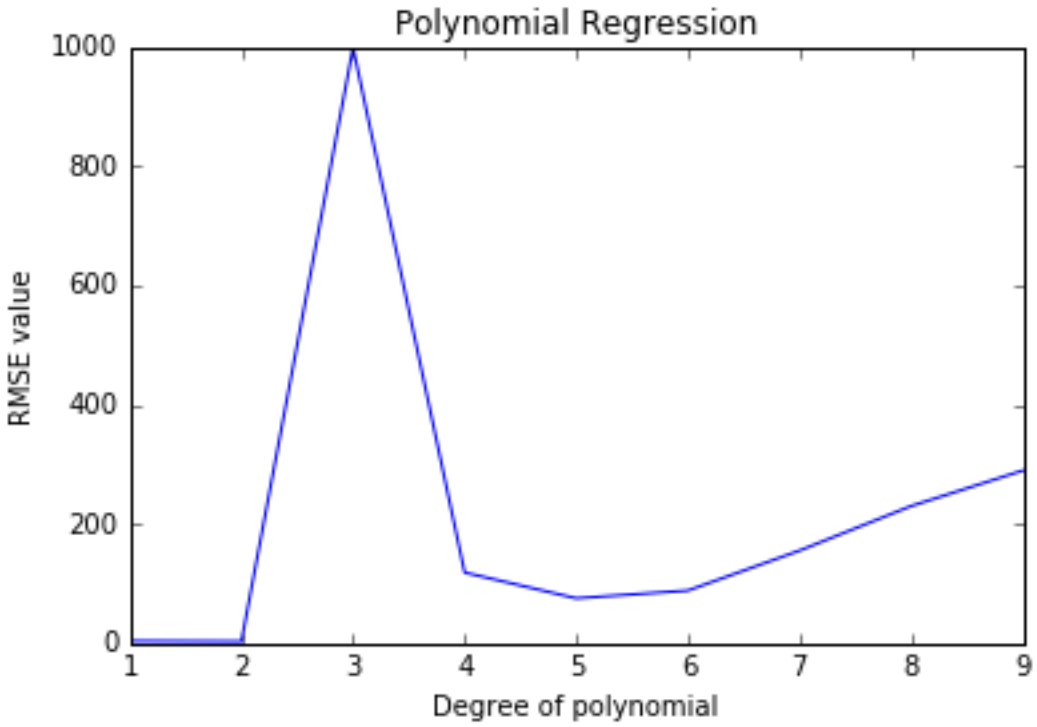
b. Polynomial Regression Model:

After loading and cleaning the data, *MDV(Median Value for a house $1000's)* is separated out as the target. We next perform pre-processing on feature set by obtaining polynomial features for the input features. Using the obtained polynomial features and training labels, we fit a linear regression model. We next perform a 10-Fold cross validation over the data. Test and Training set split is 10% and 90% respectively. Before splitting the data into 10 sections, we shuffle the input data to remove any pre-existing bias. We also obtain polynomial features for the features in test data and try to predict the unknown labels with the trained model.

This model is checked till degree 9 to obtain the optimal degree of fit. The RMSE obtained for each degree is illustrated as follows:

Degree value	RMSE
1	3.88772390585
2	3.254117836
3	998.553372066
4	118.524650282
5	75.3317118668
6	88.3090192446
7	155.253676941
8	229.934159917
9	289.458455608

We observe the spread of data in the *RMSE vs degree* (below)



As we can observe from the tabular data and the graph, RMSE value shoots after a degree of 2 thus giving me optimal degree for this data as 2.

Conclusion : Optimal degree of fit is 2 and corresponding **RMSE value is 3.254**

PROBLEM 5

To regularize the parameters and prevent overfitting of data, we are employing ridge regression model and lasso regression model. “Linear_model” library of “sklearn” is employed in Python to implement lasso and ridge regression

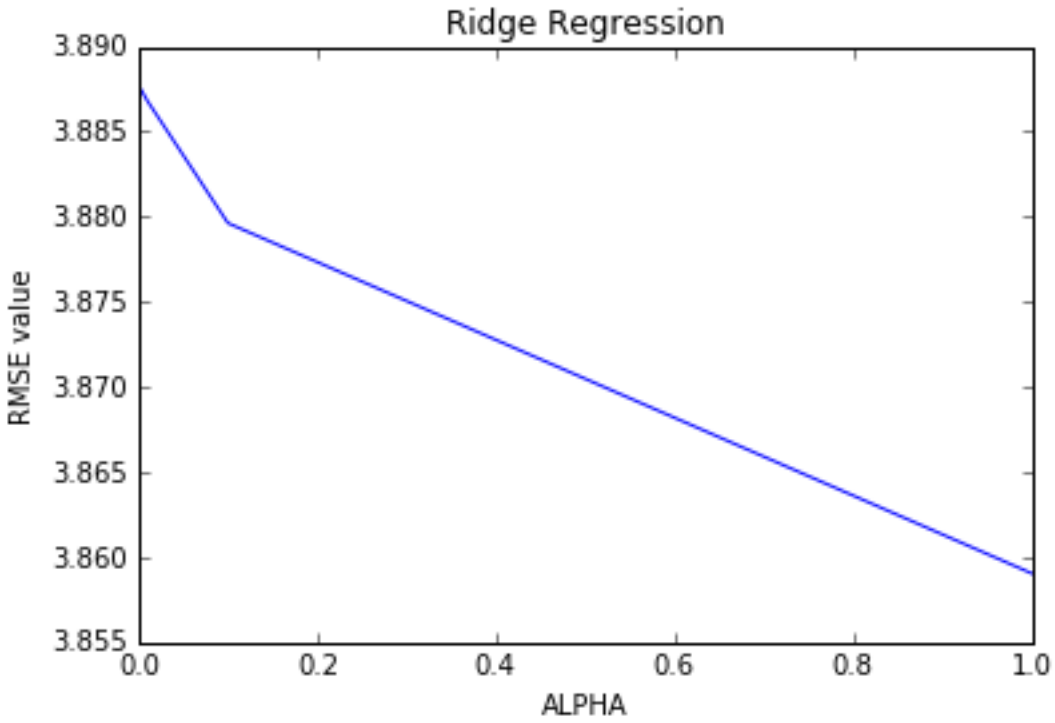
a. **Ridge Regression Model:**

We employ ridge regression which is a type of l1 regularization technique to tune the complexity parameter alpha. We first load the data and separate out the *MDV(Median Value for a house $1000’s)* as the target variable and using all the other inputs as features which are converted to the appropriate

numeric values. Then we pick an alpha from the set of values which is given and perform 10-Fold cross validation over the data. Before splitting the data into 10 sections, we shuffle the input data to remove any pre-existing bias. Above process is repeated for each value of alpha resulting in the following results:

Alpha	**RMSE**
1	3.85904839402
0.1	3.87963373235
0.01	3.88680112828
0.001	3.88763036765

We observe that there is not much difference in the RMSE values. Thus alphas do not have much effect. Though alpha value 1 gives best RMSE value but the RMSE values for alpha value 1 and 0.1 are pretty close. Minimum values of RMSE was obtained at **alpha = 1 with the value of 3.8590**



b. Lasso Regression Model:

In this question, we employ lasso regression which is a type of l2 regularization technique to tune the complexity parameter alpha. We first load the data and separate out the *MDV(Median Value for a house $1000's)* as the target variable and using all the other inputs as features which are converted to the appropriate numeric values. Then we pick an alpha from the set of values which is given and perform 10-Fold cross validation over the data. Before splitting the data into 10 sections, we shuffle the input data to remove any pre-existing bias. Above process is repeated for each value of alpha resulting in the following results:

Alpha	RMSE
1	4.33505718936
0.1	3.94828780615
0.01	3.86879191327
0.001	3.88533484981

Complexity parameter **alpha** which gives the least RMSE value and thus the optimal result is **0.01 with corresponding RMSE value as 3.8687**