ECE219 Project 4: Regression Analysis

Firnaz Ahamed (104943091) Larsan Aro Brian Arockiam(904943186)
Hariharan Shamugavadivel(804946029) Shoban Narayan Ramesh(604741670)

March 5, 2018

1 Introduction

Regression analysis is a statistical procedure for estimating the relationship between a target variable and a set of potentially relevant variables. In this project, we explore basic regression models on a given dataset, along with basic techniques to handle overfitting; namely **cross-validation**, and **regularization**. With cross-validation, we test for over-fitting, while with regularization we penalize overly complex models.

2 Dataset

We use a **Network backup Dataset**, which is comprised of simulated traffic data on a backup system over a network. The system monitors the files residing in a destination machine and copies their changes in four hour cycles. At the end of each backup process, the size of the data moved to the destination as well as the duration it took are logged, to be used for developing prediction models. We define a workflow as 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 around 18000 data points with the following columns/variables:

1. Week index
2. Day of the week at which the file back up has started
3. Backup start time: Hour of the day
4. Workflow ID
5. File name
6. Backup size: the size of the file that is backed up in that cycle
7. Backup time: the duration of the backup procedure in hour

3 Exploratory Data Analysis

Before dwelling deep into the questions that are being asked, we explore the data and get some insights into the given dataset. Throughout this regression problem, we are trying to predict the size of the backup based on several features that we have. The features include day of the week, week number, workflow ID, file name and backup start time. Some of these features have a significant impact in predicting the backup size. We began with the **exploratory data analysis** (EDA) by plotting the size of backup as the week number, day of week and work-flow ID changes.

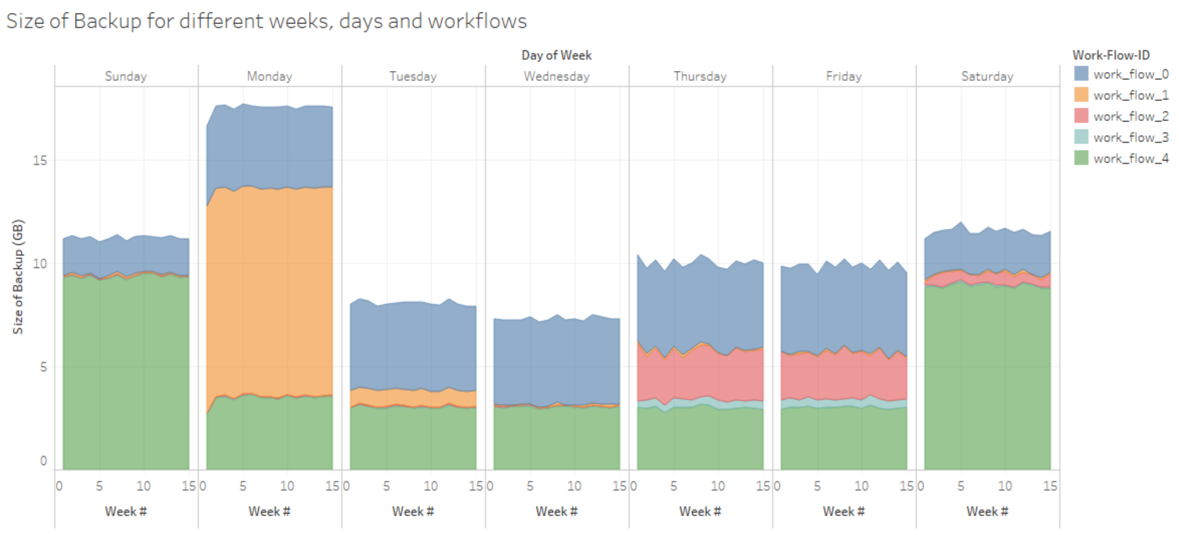

Figure 1: Size of backup for different days, weeks and workflow IDs

The x-axis shows the different days and for each day the different week number are varied in finer granularity. The work-flow ID is color coded to show the size of backup in the y-axis.

Inference The inference that can be obtained from this graph, is the significance of day and insignificance of week. As seen from this visualization, within each day's plot, the variation of the size of backup is trivially small as the week number changes. Hence, the **week number** is an **insignificant** feature in predicting the size of the backup. However, we can see visible changes between the plots of different days. For example, there is a huge increase in the size of backup between Sunday and Monday.

Similarly, we can also see the impact of day in affecting the various workflows. We can see that workflow ID 1 (orange color) does not have any significant effect in the backup size on all days except Monday. On Mondays, workflow ID 1 has a huge impact on the size of the backup. Hence the feature Workflow ID, when combined with the feature day of week, can play a pivotal role in determining the size of backup.

4 Analysis

4.1 Question 1a

Implementation In this part, we load the dataset and get an idea on type of relationships in the dataset. The backup sizes are plotted for the first 20 period. Since, we have different backup sizes on the same day for different work flows, we color code the **work flows** on Y-axis. In the pandas dataframe, we add another column in addition to Week number and Day of the week, which combined to give the Day number.

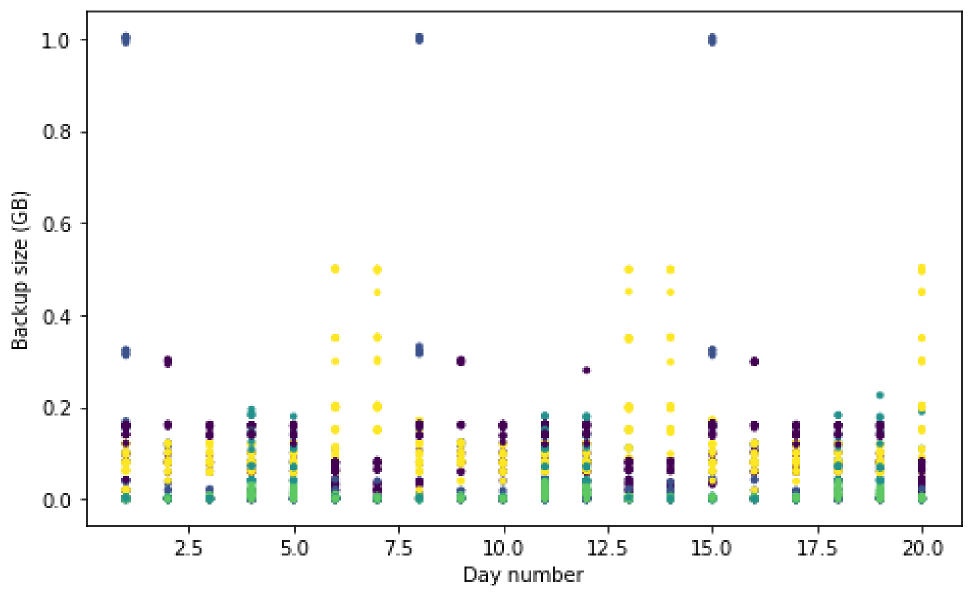

Figure 2: Plot of backup sizes for all workflows over twenty days

Inference We can observe several **repeating patterns** from the graph. For example, for a particular workflow-ID (color coded as yellow) the peak backup size is reached consistently on **weekends** - Saturdays and Sundays. The yellow peaks around 0.2-0.6GB can be observed on days 6,7 and 13,14 as well as on Day 20. Also, the highest ever backup size of 1GB occurs repeatedly on **Mondays**. Also, the green color-coded workflow-ID has the lowest backup sizes over all weekdays.

4.2 Question 1b

Implementation We repeat the same implementation as in Question 1a for the entire 105-day period.

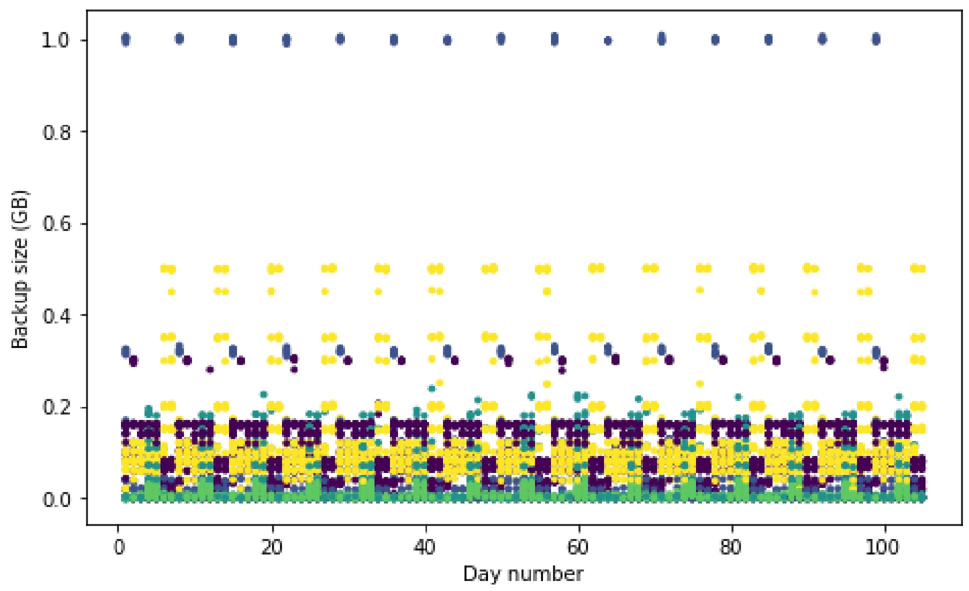

Figure 3: Plot of backup sizes for all workflows over 105 days

Inference The same repeating patterns from the **20-day** observation can be seen in this graph as well. For example, the highest ever backup sizes of 1GB still repeatedly occurs on Mondays. The other patters on weekends and weekdays still happen, however they are not clearly visible in this plot as in the 20-day plot owing to the high number of data points.

We observe that even such a simple scatter plot helps us get a quick idea of how the backup sizes vary with different days and workflow-IDs without any regression analysis. In the next tasks, a more rigorous approach towards various regression will be analysed.

4.3 Question 2a

Task In this task, we'll be performing **linear regression** on the data, trying to predict backup size of a file given other attributes in the dataset. All features like Week Number, Day of the Week, Hour of the day, Workflow ID, File name are all used while Backup time is excluded. We also will try scalar and One-Hot encoding. Throughout this section, root mean square error is taken as the metric for evaluation.

$$\text{RMSE}_{train} = \sqrt{\frac{mse_{train,1} + \cdots + mse_{train,10}}{10}} = \sqrt{\frac{sse_{train,1} + \cdots + sse_{train,10}}{10 \times (\text{\# of training data points in each fold})}}$$

$$\text{RMSE}_{test} = \sqrt{\frac{mse_{test,1} + \cdots + mse_{test,10}}{10}} = \sqrt{\frac{sse_{test,1} + \cdots + sse_{test,10}}{10 \times (\text{\# of test data points in each fold})}}$$

In our analysis, we calculate **mean squared error** for each fold (here 10-fold cross validation) and take the average and then square root it to get the train/test **RMSE**.

4.3.1 Question 2a(i)

Implementation The entire dataset is scalar encoded (i.e) convert all categorical features to numerical values. For example, the weekdays Monday to Sunday will take on values from 1 to 7. We then perform a 10-fold cross validation on the dataset and fit a linear regressor to predict the backup sizes. The results can be seen below.

	Linear Regression
Training RMSE	0.103585
Test RMSE	0.103675

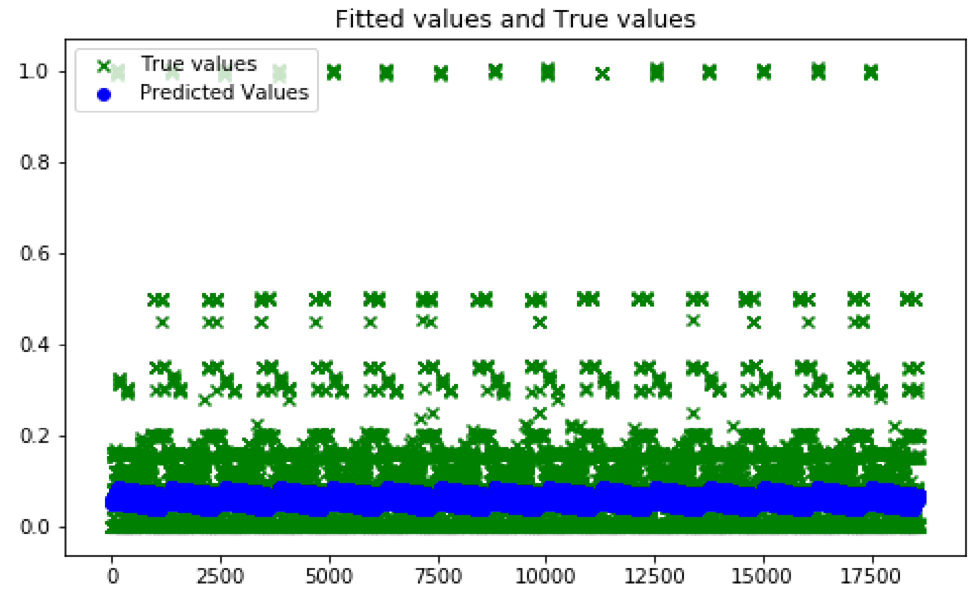

Figure 4: Plot of fitted values and true values scattered over all data points

It can be seen from the plot that the regressor tries to fit the true values as much as possible, however it's restricted to a small band between 0 - 0.2. Ideally, the range should have been higher and fit a lot of true values. This shows that while a lot of points are fit correctly, a few outliers are not predicted correctly.

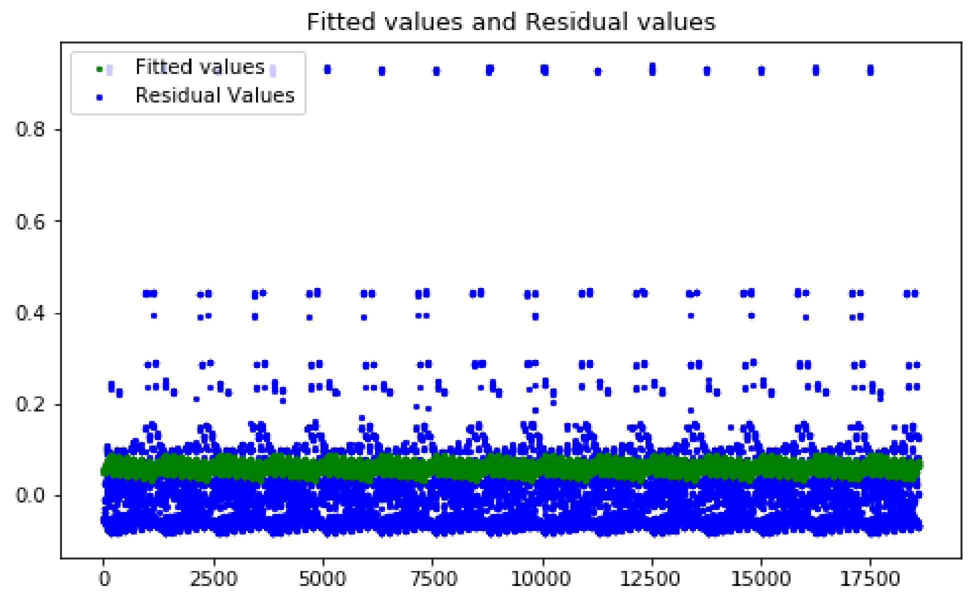

Figure 5: Plot of fitted values and residual values scattered over all data points

The above plot shows **residual values** on a very small band showing that the error is close to zero - which is a good measure of the regressor. This can also be verified in the below plot, where we plot residual values vs fitted values. The residual values on Y-axis mostly remain near zero.

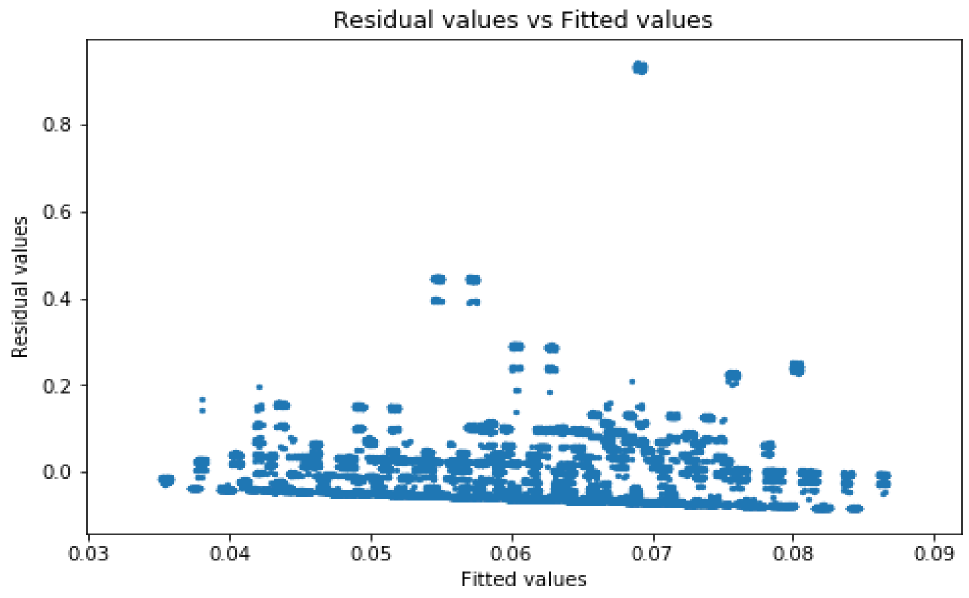

Figure 6: Plot of residual values vs fitted values for Linear Regression

4.3.2 Question 2a(ii)

Implementation The entire input is standardized and is checked on the Linear Regression. **Standardization** of a dataset is a common requirement for many estimators as they might behave badly if the individual feature do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance). We use **StandardScaler** of sklearn to set the mean to zero and variance to one. The results can be seen below.

	Linear Regression - Standardized
Training RMSE	0.103585
Test RMSE	0.103675

Assuming the features are $x = [x_1, \ldots, x_n]$, linear regression model will learn the parameters $\theta = [\theta_0, \theta_1, \ldots, \theta_n]$ such that $y = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$.

On normalizing the features, we have $\tilde{x} = [\tilde{x}_1 = \frac{x_1 - \mu_1}{\sigma_1}, \ldots, \tilde{x}_n = \frac{x_n - \mu_n}{\sigma_n}]$
Thus on learning a new linear model on the new features with parameters $\tilde{\theta} = [\tilde{\theta}_0, \tilde{\theta}_1, \ldots, \tilde{\theta}_n]$ and plugging in $\tilde{\theta}_i = \sigma_i \theta_i$ for $i = 1, \ldots, n$ and $\tilde{\theta}_0 = \theta_0 + \sum_{i=1}^{n} \mu_i \theta_i$ we get the exact same model, and thus the same accuracy. This is the reason that the standardization of features do not produce a different result than normal linear regression.

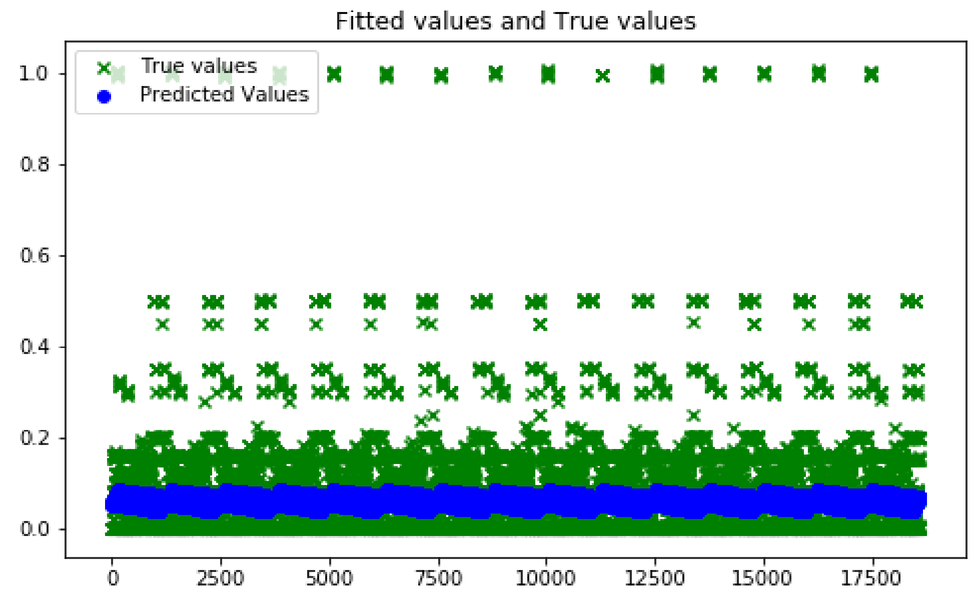

Figure 7: Plot of fitted values and true values scattered over all data points

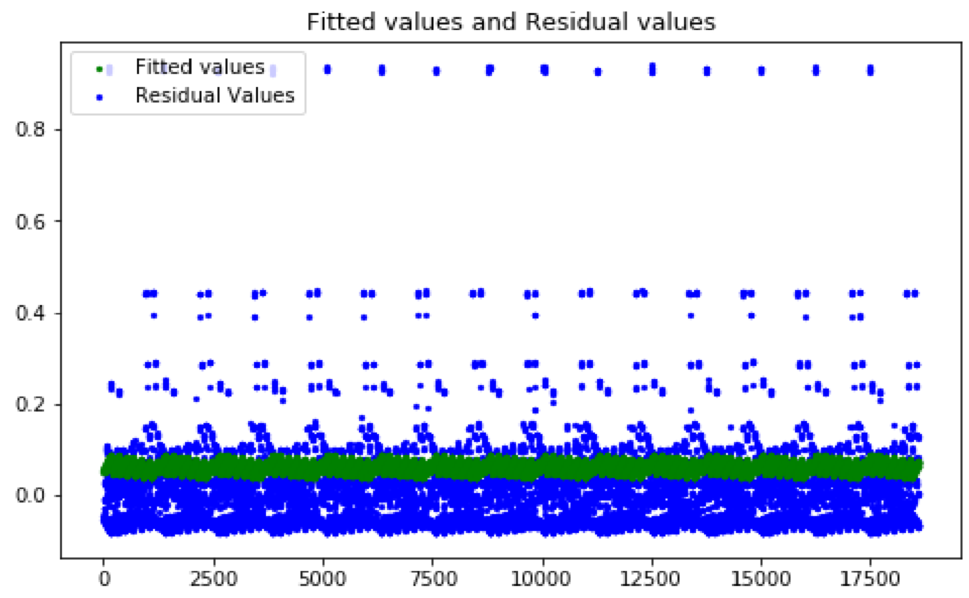

Figure 8: Plot of fitted values and residual values scattered over all data points

As inferred earlier, there is no notable difference in the plots after standardization of the data. The residuals remain close to zero while the fitted values mostly match the true values as earlier.

4.3.3 Question 2a(iii)

Implementation In this part, we use feature selection techniques to select the most important variables and train them on a new linear model. We perform **f_regression** and **mutual_info_regression** to select the three most important variables and observe train and test RMSE.

f_regression This is a scoring function to be used in a feature seletion procedure and not a free standing feature selection procedure. This is done in 2 steps:

1. The correlation between each regressor and the target is computed, that is, ((X[:, i] - mean(X[:, i])) * (y - mean_y)) / (std(X[:, i]) * std(y)).

2. It is converted to an F score then to a p-value

	Linear Regression - f_regression
Training RMSE	0.090625
Test RMSE	0.090783

We see that there is a slight improvement over the previous Linear Regression. This can be attributed to the fact that only the important features are selected for the model to learn to fit the test data. The three most important features are:

1. **Workflow_ID**
2. **Day of Week**
3. **Backup Start Time - Hour of Day**

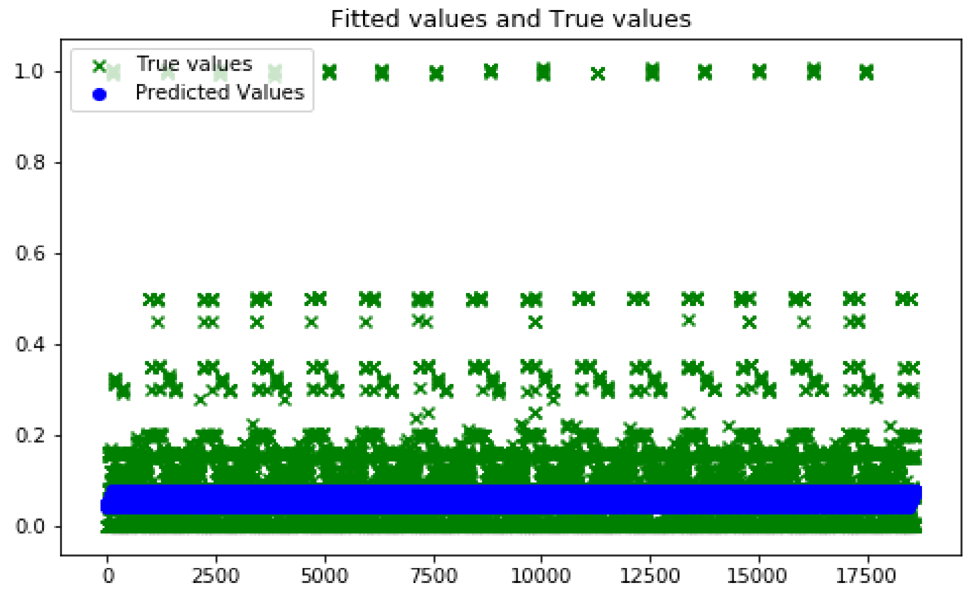

Figure 9: Plot of fitted values and true values over all data points - f_regression

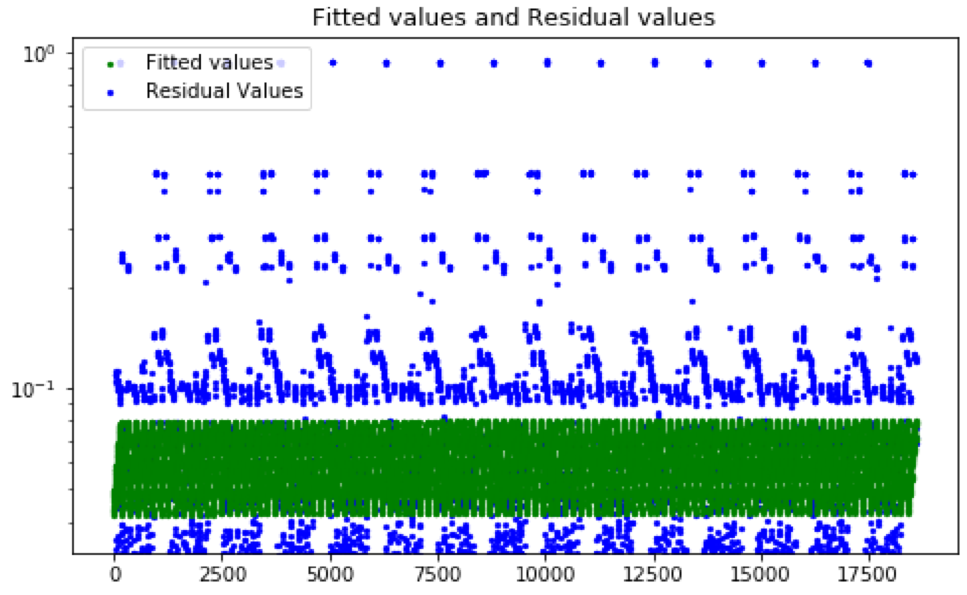

Figure 10: Plot of fitted values and residual values over all data points - f_regression

mutual_info_regression **Mutual information** between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency. The function relies on nonparametric methods based on entropy estimation from k-nearest neighbors distances.

	Linear Regression - mutual_info_regression
Training RMSE	0.090295
Test RMSE	0.090113

We see that even in this feature selection, there is a slight improvement over Vanilla Linear Regression. This can be attributed to the fact that only the important features are selected for the model to learn to fit the test data. The three most important features are:

1. Backup Start Time - Hour of Day
2. Workflow_ID
3. File_number

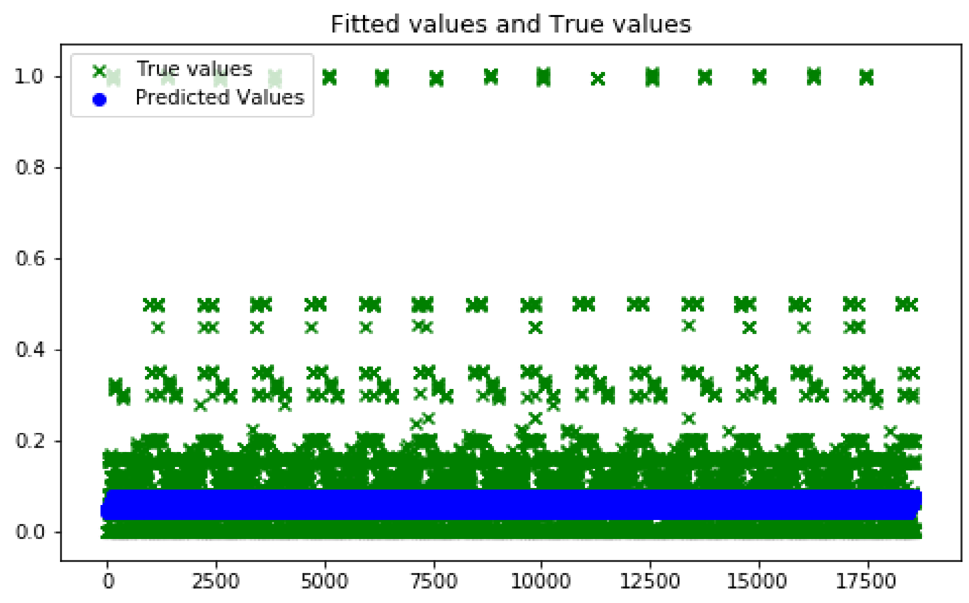

Figure 11: Plot of fitted values and true values over all data points - mutual_info_regression

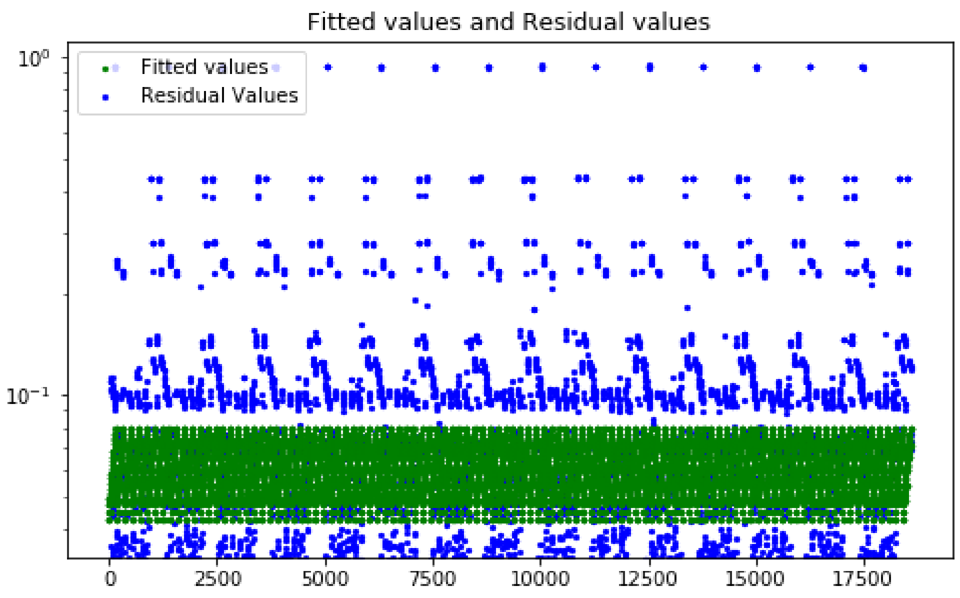

Figure 12: Plot of fitted values and residual values over all data points - mutual_info_regression

To **compare f_regression and mutual_info_regression**, the latter seems to perform better giving a lower test RMSE. This is because f_scores can capture only linear dependency between the features while mutual_info_score can capture any kind of dependency between variables.

4.3.4 Question 2a(iv)

Task In this part, we perform various feature encodings on the data - all possible combinations of scalar and one-hot encoding. We observe Linear Regression's performance on these encodings.

Implementation Scalar encoding just transforms non-numerical features to numerical features. For example, days of the week is transformed from Sunday to Saturday as 1 to 7. **One-hot encoding** converts numerical categorical features into a sparse matrix where each column corresponds to one possible value of one feature. The discrete categorical features have to be in the range [0, n_values). For each categorical variable that takes one of M values we one-hot encoding changes it as an M dimensional vector, where only one entry is 1 and the rest are 0s. Thus for the Day of the Week, Monday could be encoded as [1; 0; 0; 0; 0; 0; 0] and Friday as [0; 0; 0; 0; 0; 0; 1].

Since we have 5 features, we have 2^5 (32) combinations of scalar and one-hot encoding of features. A **powerset** method was written which could return all possible subsets of a set (list of features) and pass that to the categorical_features argument in OneHotEncoder.

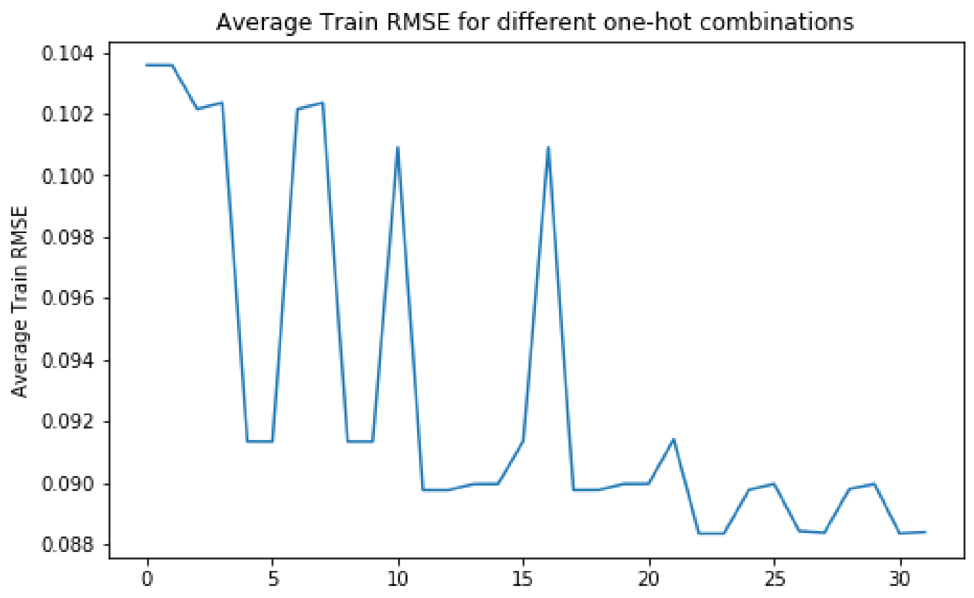

Figure 13: Plot of average train RMSE over 32 One-hot combinations

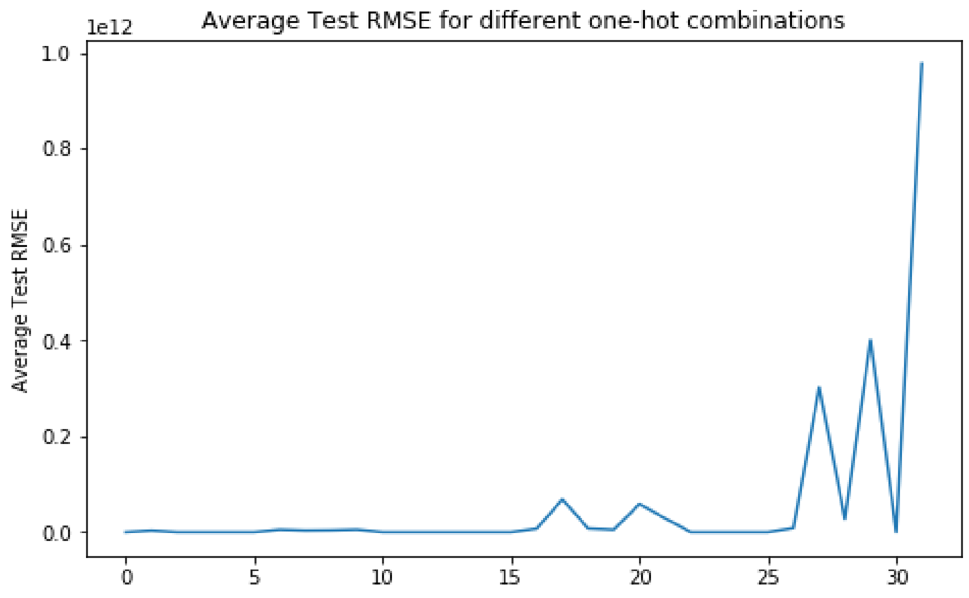

Figure 14: Plot of average test RMSE over 32 One-hot combinations

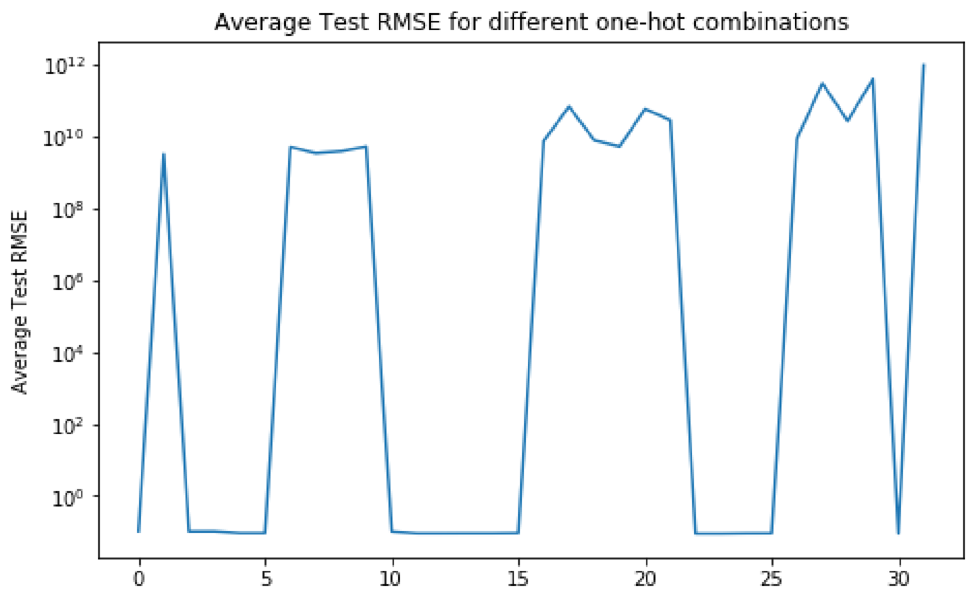

Figure 15: Plot of average test RMSE over 32 One-hot combinations (Log scale)

	Feature Encoded Regression	One-hot Combination
Lowest Training RMSE	0.088341	Day/Hour/Work-Flow-ID/File-Name
Lowest Test RMSE	0.088513	Day/Hour/Work-Flow-ID/File-Name

Inference It can be seen that different combinations of scalar/One-Hot encoding produces drastically varying results. Test RMSE of few combinations are in range of 10^{11} while others are really low ($\sim$ 0.08). The best train and test RMSE was achieved when **Day, Hour, Work Flow ID and File Name** were One-Hot encoded. This emphasizes the importance of One-Hot encoding for regression tasks and can be attributed to the fact that One-Hot encoding eliminates higher importance being attributed to features with higher categorical values. For example, Hour-1 and Hour-24 are just two different features and just because 24 is higher than 1, it should not be assumed of higher value. So, one-hot encoding allows **binarization** of these values into separate features.

Intuitive Explanation of the results : So, ideally the lowest RMSE should have been achieved when all categorical features were one-hot encoded. However, it can be seen that whenever Week Number was one-hot encoded, the RMSE is very high. As can be seen from the table below that the 16 combinations without **Week One-hot encoded** produce good results while any combination including the Week produce very bad results.

This can be due to the fact that Week Number is an unnecessary feature, which intuitively makes sense since the network activity generally depends on the day and it remains the same irrespective of which week we are in. The week number is unnecessary and since there are 15 weeks, we get a **highly sparse One-hot encoding** which could have affected the results adversely. There is more support to this statement from our previous analysis which is listed in next section.

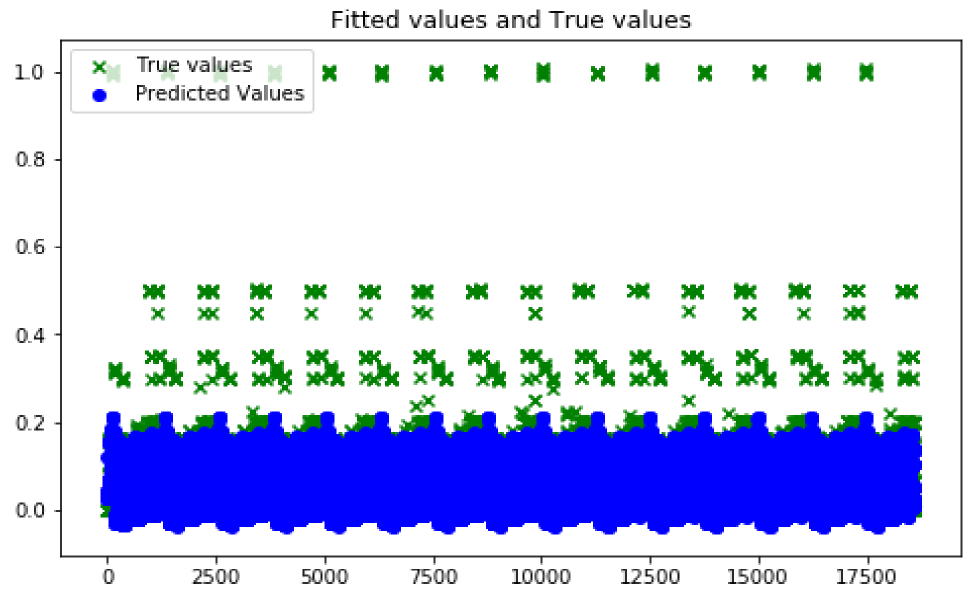

Figure 16: Plot of fitted values and true values for best One-hot combination

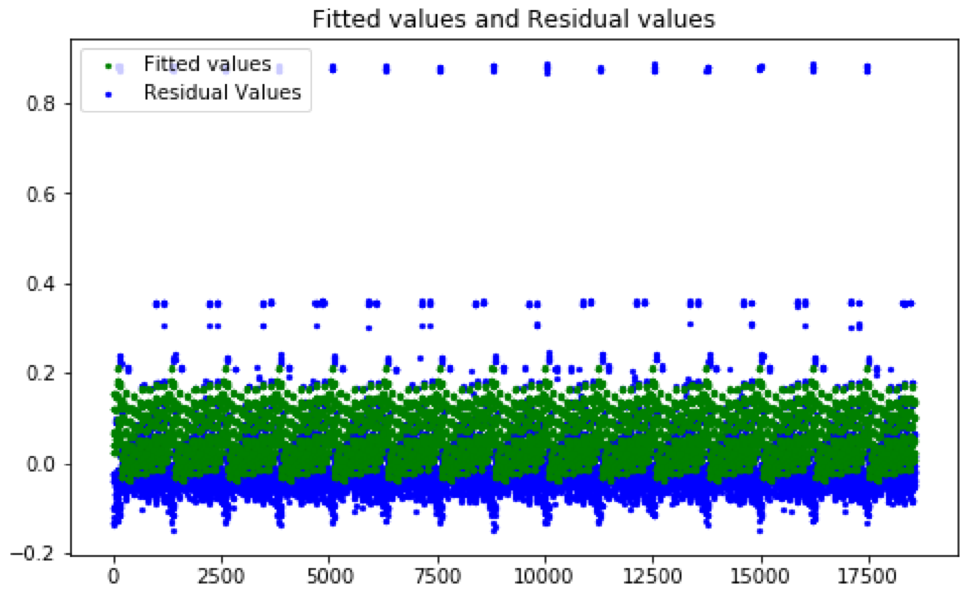

Figure 17: Plot of fitted values and residual values for best One-hot combination

It can be seen from the plot that the regressor tries to fit the true values as much as possible, however it's restricted to a small band between 0 - 0.2. The above plot shows residual values on a very small band showing that the error is close to zero - which is a good measure of the regressor. This can also be verified in the below plot, where we plot residual values vs fitted values. The residual values on Y-axis mostly remain near zero.

4.3.5 Question 2a(v)

Obvious increase in test RMSE As stated in the previous section, for all combinations with week One-hot encoded, we get obvious increases in test-RMSE. This can be due to the fact that Week Number is an unnecessary feature in the dataset. There is more support to this statement from our previous analysis as below.

1. **Exploratory Data Analysis** before Question 1a) also showed that day of the week to be more important than week number.

2. Question 2 a) iii) **feature selection** also showed that in both f_regression and mutual_info_regression, week number was not an important feature.

We can more clearly see that in the below table, where when week appears in the one-hot encoded combination, the test-RMSE increases drastically.

Test RMSE	One-hot Combination
0.088513	Day/Hour/Work-Flow-ID/File-Name
0.088517	Day/Hour/File-Name
0.088518	Day/Hour/Work-Flow-ID
0.089907	Day/Work-Flow-ID
0.089911	Day/File-Name
0.089924	Day/Work-Flow-ID/File-Name
0.090106	Hour/Work-Flow-ID/File-Name
0.090121	Hour/Work-Flow-ID
0.090124	Hour/File-Name
0.091495	Work-Flow-ID
0.091499	File-Name
0.091516	Work-Flow-ID/File-Name
0.101002	Day/Hour
0.102233	Day
0.102463	Hour
0.103676	
3263409672	Week
3453049340	Week/Hour
3898019427	Week/Work-Flow-ID
5135719350	Week/Day
5250504557	Week/File-Name
5260324848	Week/Hour/Work-Flow-ID
7427084137	Week/Day/Hour
7866949423	Week/Day/File-Name
8742182958	Week/Day/Hour/Work-Flow-ID
27271449548	Week/Day/Work-Flow-ID/File-Name
28706110043	Week/Work-Flow-ID/File-Name
58742427133	Week/Hour/File-Name
68387891908	Week/Day/Work-Flow-ID
302092246880	Week/Day/Hour/File-Name
401193928839	Week/Hour/Work-Flow-ID/File-Name
977269750531	Week/Day/Hour/Work-Flow-ID/File-Name

Fitted coefficients On taking a quick glance at the fitted coefficients of the models listed above, we find that the best models have coefficients of the order of 10^{-3} while the poor performing models have coefficients of the order of 10^{10}. This means that the poor models do not impose any constraint over the coefficients and allow them to capture noise and variance in the training data as much as possible, translating to poor generalization and in turn, very high test RMSE.

Example:

For one-hot encoding just file name, the coefficients are [[3.97401049e-02 -1.38019411e-02 -

4.03083885e-02 -5.72999472e-02 7.16701719e-02 5.86292314e-05 -2.46609448e-03 1.37283436e-03 4.21143907e-05]]

For one-hot encoding Week number and Workflow ID, the coefficients are [[5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 5.63392706e+10 -1.16243926e+09 -1.16243926e+09 -1.16243926e+09 -1.16243926e+09 -1.16243926e+09 -1.16243926e+09 -2.34389305e-03 1.38866901e-03 -1.24573708e-04]]

Controlling ill-conditioning and over-fitting We observed very significant increases in test RMSE compared to training RMSE in a few combinations. This can be attributed to **overfitting** of training data. That is the model tries to capture not only the variance of the data but also the inherent noise in it. Thus, the model will not be able to **generalize** well in the test data. In extreme cases of overfitting, the fitted curve tends to go through every single data point. Thus, when unseen new data is tested on this model, the curve will produce high test RMSE.

We also face the issue of **ill-conditioning**, where a small change in the input causes a huge change in the output. The predictions should not be sensitive to slight noises in the input and should allow room for that.

To solve the problem, we make use of regularization. **Regularization** tries to learn a very simple model that performs well. It is a constraint on the weights of the model, controlling the freedom of the weights to be very high but at the same time aimed at reducing the error.

Ridge regularizer Ridge Regularizer:

$$\min_{\beta} \|Y - X\beta\|^2 + \alpha {\|\beta\|_2}^2$$

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the **l2-norm**. So, in addition to minimizing the residual sum of squares error, we also have to minimise this L2 penalty on the weights.

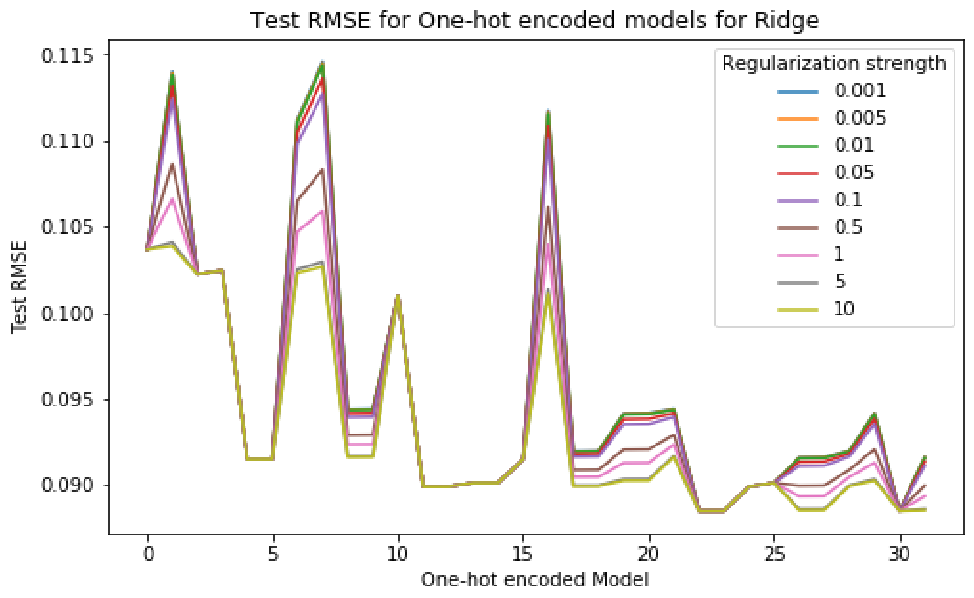

Figure 18: Test RMSE vs One-hot combinations and regularization strengths - Ridge

Table 1: Ridge regression lowest test RMSE for different regularization strengths

Regularization Strength	Test RMSE	One-hot Combination
1e-3	0.08850426065775907	Day/Hour/Work-Flow-ID/File-Name
5e-3	0.08850426047043786	Day/Hour/Work-Flow-ID/File-Name
1e-2	0.0885042602363801	Day/Hour/Work-Flow-ID/File-Name
5e-2	0.0885042583676677	Day/Hour/Work-Flow-ID/File-Name
1e-1	0.08850425604114934	Day/Hour/Work-Flow-ID/File-Name
5e-1	0.08850399605176956	Day/Hour/Work-Flow-ID/File-Name
1	0.08850423780345533	Day/Hour/Work-Flow-ID/File-Name
5	0.08850421594039007	Day/Hour/Work-Flow-ID/File-Name
10	0.08850407798481677	Day/Hour/Work-Flow-ID/File-Name

We can see from the plot that, irrespective of the regularization strength, a particular combination always gives the lowest Test-RMSE - the **Day-Hour-Work_Flow_ID-File_Name one-hot combination**. This is consistent with our results from the previous section, where we do not One-hot encode the week feature. The test-RMSE does no vary much for this particular model with respect to regularization strength. So, we choose 0.001 and this particular combination and we fit the regression model.

Table 2: Ridge Regression - Day/Hour/Work-Flow-ID one-hot encoded and alpha = 0.001

	Optimized Ridge Regression
Training RMSE	0.0883374
Test RMSE	0.08850426

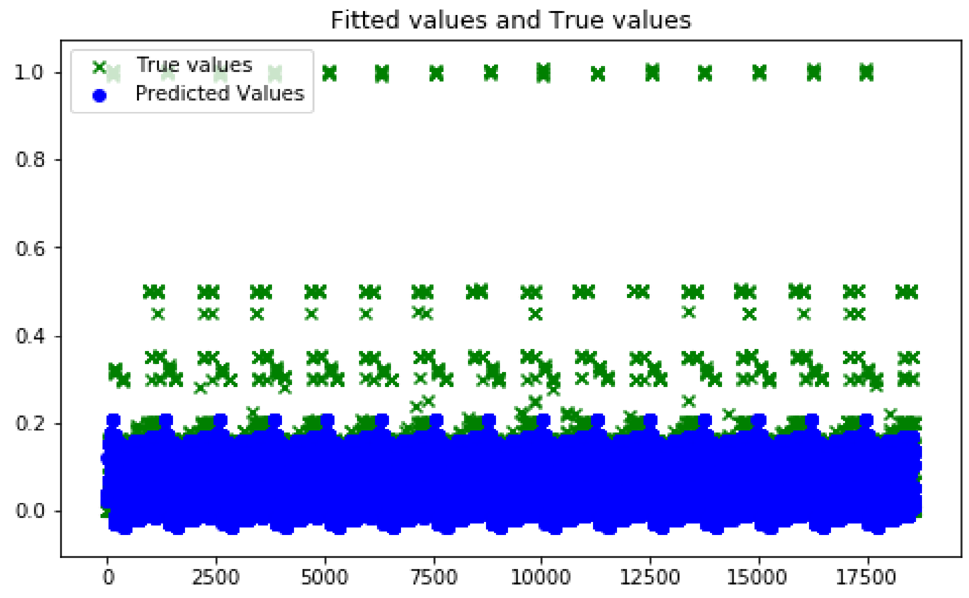

Figure 19: Plot of true values and predicted values for Optimized Ridge Regression

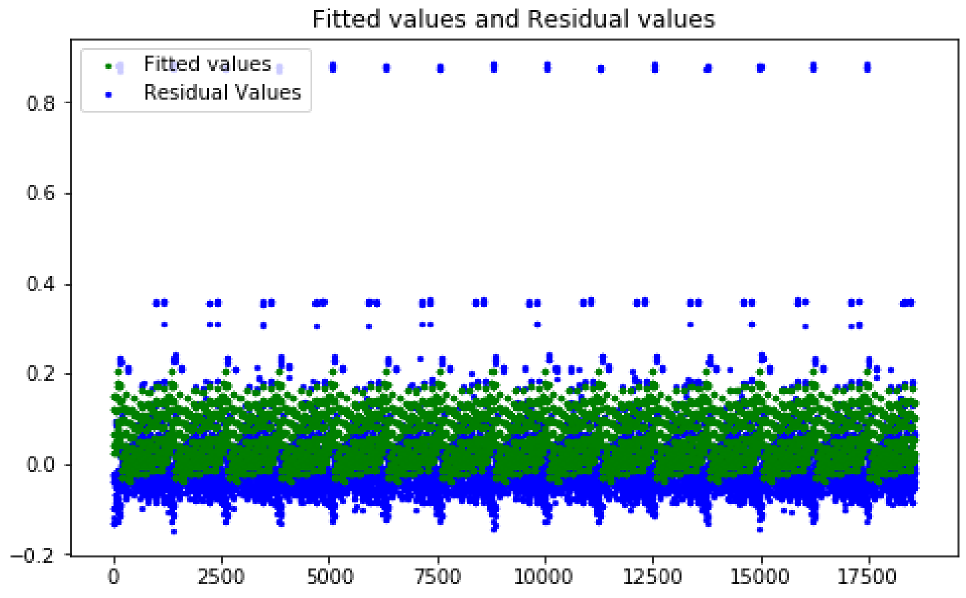

Figure 20: Plot of residual values and predicted values for Optimized Ridge Regression

We can see that we have improved the test-RMSE from our previous tasks. We also see the predicted values fit the true values better and the residuals remain close to zero.

Lasso regularizer Lasso Regularizer:

$$\min_{\beta} \|Y - X\beta\|^2 + \alpha\|\beta\|_1$$

This is a linear Model trained with **L1 prior** as regularizer (aka the Lasso). Compared to ridge, here even at small regularization strength, the coefficients reduce to zeros. This helps in better feature selection, as Lasso regularization produces **sparse results**.

Since a ridge regressor squares the error, the model will see a much larger error than the lasso regressor, so the model is much **more sensitive** to an example, and adjusts the model to minimize this error. If this example is an outlier, the model will be adjusted to minimize this single outlier case, at the expense of many other common examples, since the errors of these common examples are small compared to that single outlier case.

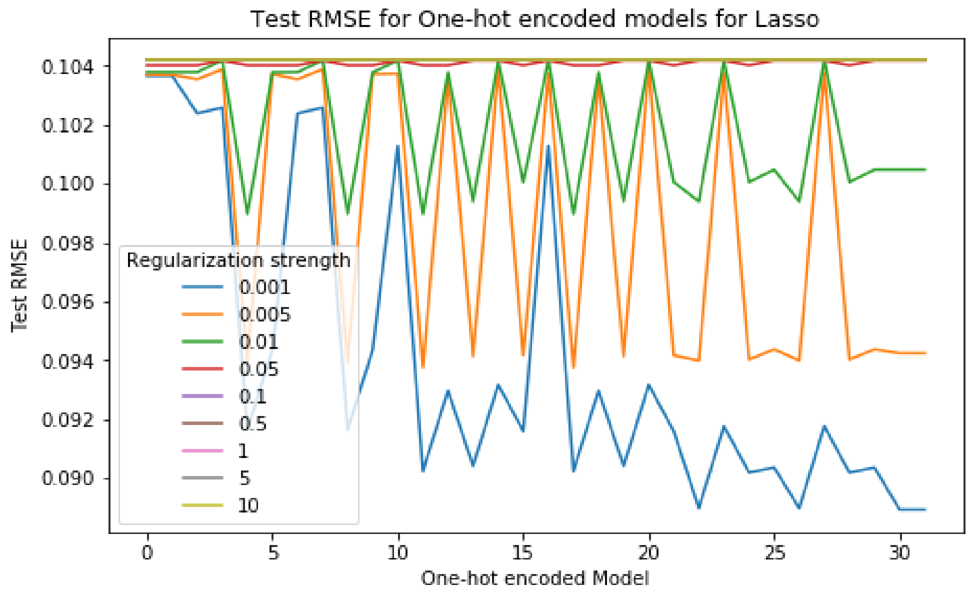

Figure 21: Test RMSE vs One-hot combinations and regularization strengths - Lasso

Table 3: Lasso regression lowest test RMSE for different regularization strengths

Regularization Strength	Test RMSE	One-hot Combination
1e-3	0.08893870957224574	Day/Hour/Work-Flow-ID/File-Name
5e-3	0.09377300396286463	Day/Work-Flow-ID
1e-2	0.09897362685888202	Work-Flow-ID
5e-2	0.10402996801999731	Day/Hour/Work-Flow-ID/File-Name
1e-1	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
5e-1	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
1	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
5	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
10	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name

We do not observe clear distinction between the lowest Test RMSE for same range of regularization strengths as we did for Ridge regression. So, we observe the lowest two range and expand the search over it as below.

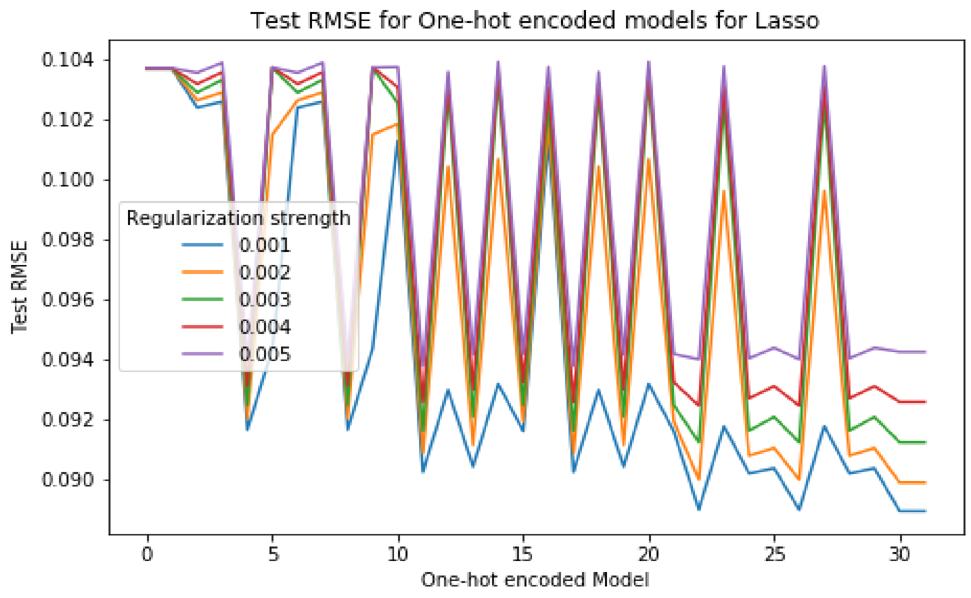

Figure 22: Test RMSE vs One-hot combinations and regularization strengths - Lasso

Table 4: Lasso regression lowest test RMSE for different regularization strengths

Regularization Strength	Test RMSE	One-hot Combination
0.001	0.08893870957224574	Day/Hour/Work-Flow-ID/File-Name
0.002	0.08988943088040094	Day/Hour/Work-Flow-ID/File-Name
0.003	0.09122951550045655	Day/Hour/Work-Flow-ID/File-Name
0.004	0.09245029652295587	Day/Hour/Work-Flow-ID/File-Name
0.005	0.09377300396286463	Day/Hour/Work-Flow-ID/File-Name

With the new range of regularization strength, we can see from the plot that, a particular combination always gives the lowest Test-RMSE - the **Day-Hour-Work_Flow_ID-File_Name one-hot combination**. This is also consistent with our results from the previous section, where we do not One-hot encode the week feature. We choose the best model from this and observe the following.

Table 5: Lasso Regression - All features one-hot encoded and alpha = 0.001

	Optimized Lasso Regression
Training RMSE	0.0887745
Test RMSE	0.0889387

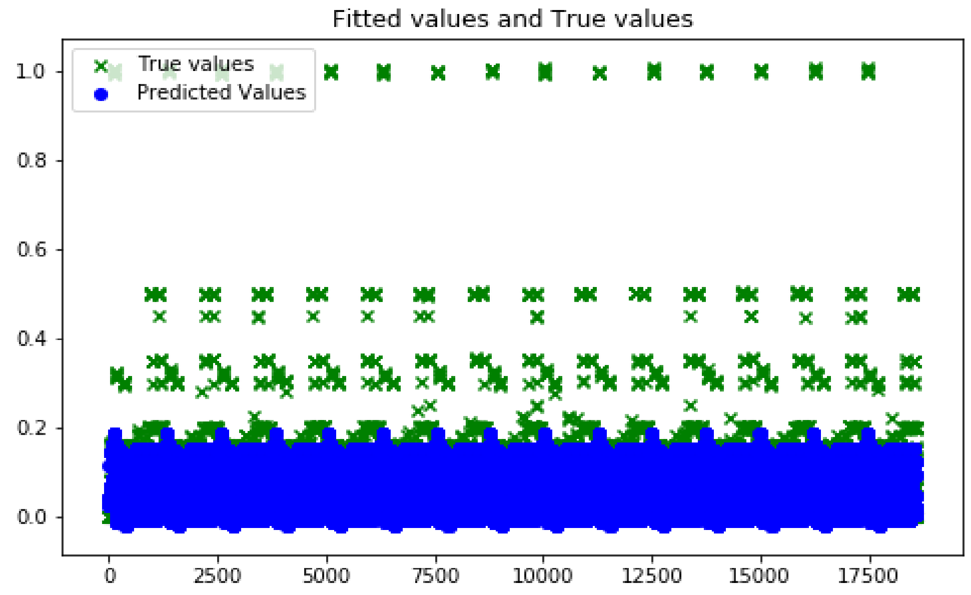

Figure 23: Plot of true values and predicted values for Optimized Lasso Regression

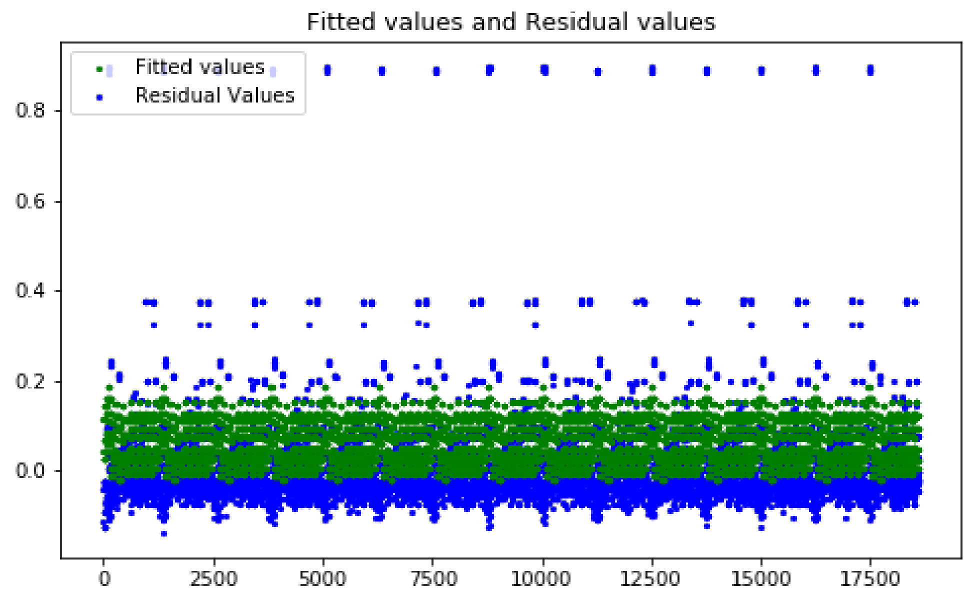

Figure 24: Plot of residual values and predicted values for Optimized Lasso Regression

ElasticNet regularizer This regularizer combines both L1 and L2 norms as prior.

Elastic Net Regularizer:

$$\min_{\beta} \|Y - X\beta\|^2 + \lambda_1\|\beta\|_1 + \lambda_2{\|\beta\|_2}^2$$

The sklearn implementation of ElasticNet does not have two different hyperparameters as λ_1 and λ_2. Rather only one argument α is used, where $\alpha = \lambda_1 + \lambda_2$. We can pass another argument l1_ratio = λ_1 / (λ_1 + λ_2).

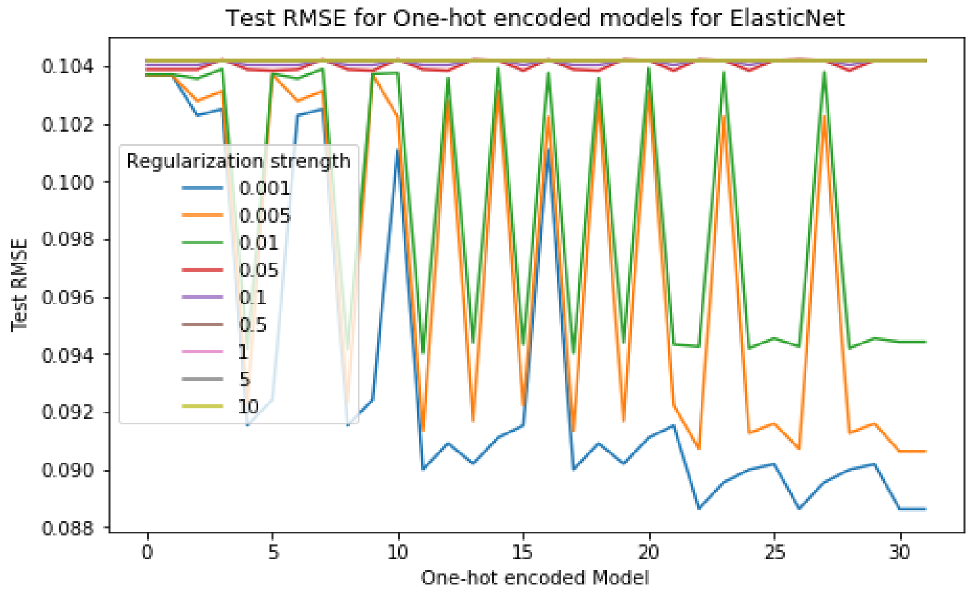

Figure 25: Test RMSE vs One-hot combinations and regularization strengths - ElasticNet

Table 6: ElasticNet regression lowest test RMSE for different regularization strengths

Regularization Strength	Test RMSE	One-hot Combination
1e-3	0.08862805625176125	Day/Hour/Work-Flow-ID/File-Name
5e-3	0.09062467585503	Day/Work-Flow-ID
1e-2	0.09403026137127211	Work-Flow-ID
5e-2	0.10383676126986398	Day/Hour/Work-Flow-ID/File-Name
1e-1	0.10403011829185733	Day/Hour/Work-Flow-ID/File-Name
5e-1	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
1	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
5	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name
10	0.10419023788574096	Day/Hour/Work-Flow-ID/File-Name

We do not observe clear distinction between the lowest Test RMSE for same range of regularization strengths as we did for Ridge regression. So, we observe the lowest two range and expand the search over it as below.

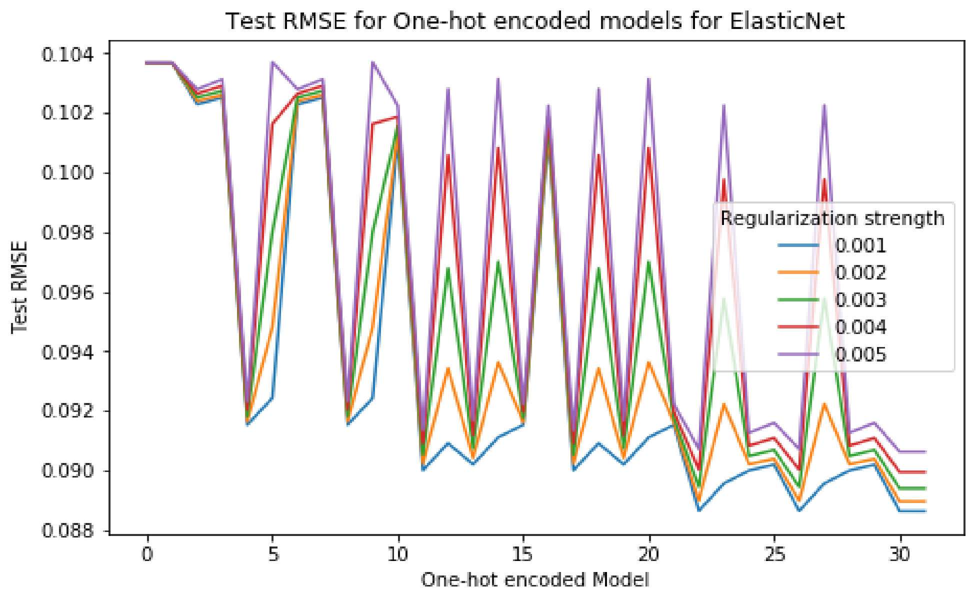

Figure 26: Test RMSE vs One-hot combinations and regularization strengths - ElasticNet

Table 7: ElasticNet regression lowest test RMSE for different regularization strengths

Regularization Strength	Test RMSE	One-hot Combination
0.001	0.08862805625176125	Day/Hour/Work-Flow-ID/File-Name
0.002	0.08895465159738973	Day/Hour/Work-Flow-ID/File-Name
0.003	0.08939179380480124	Day/Hour/Work-Flow-ID/File-Name
0.004	0.08994208185331766	Day/Hour/Work-Flow-ID/File-Name
0.005	0.09062467585503	Day/Hour/Work-Flow-ID/File-Name

With the new range of regularization strength, we can see from the plot that, a particular combination always gives the lowest Test-RMSE - the **Day-Hour-Work_Flow_ID-File_Name one-hot combination**. This is also consistent with our results from the previous section, where we do not One-hot encode the week feature. We choose the best model from this and observe the following.

Table 8: ElasticNet Regression - All features one-hot encoded and alpha = 0.001

	Optimized ElasticNet Regression
Training RMSE	0.0884675
Test RMSE	0.088628

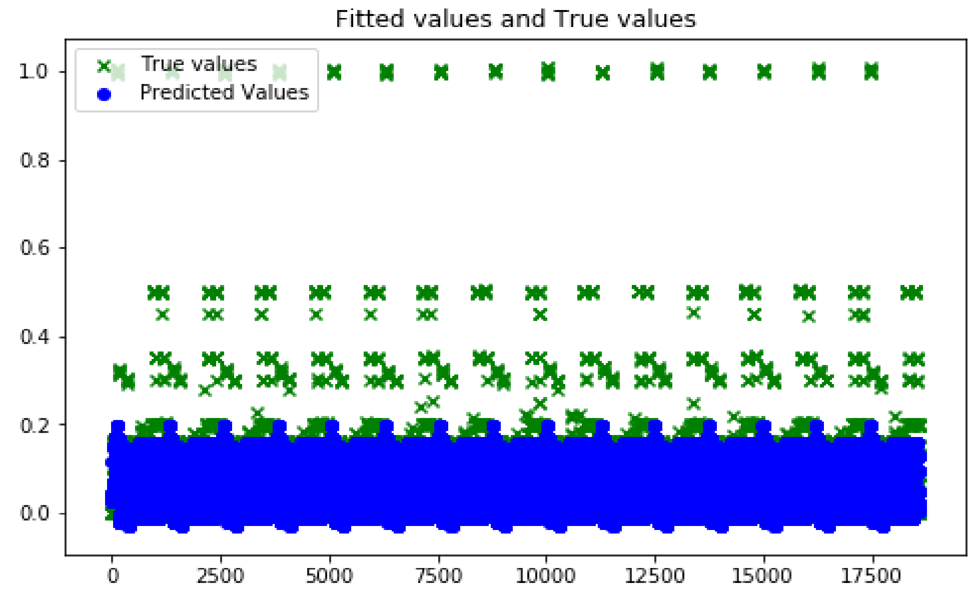

Figure 27: Plot of true values and predicted values for Optimized ElasticNet Regression

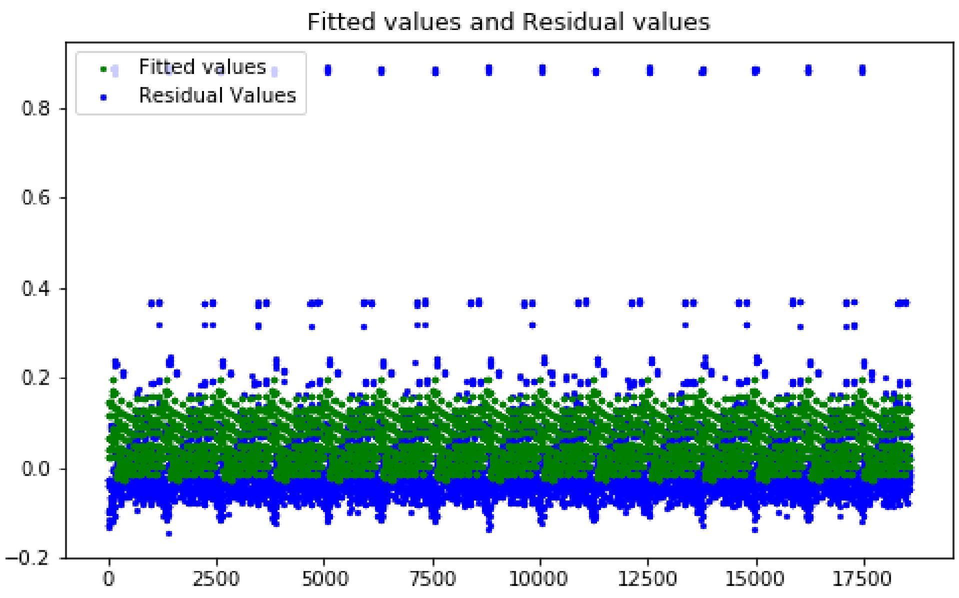

Figure 28: Plot of residual values and predicted values for Optimized ElasticNet Regression

Coefficient Analysis We try to compare the coefficients of our best model when unregularized and for each regularization. Our best model in each case was the model where we one-hot encode all features except the Week number.

Coefficients of unregularized model :

6.05042135e+09	6.05042135e+09	6.05042135e+09	6.05042135e+09
6.05042135e+09	6.05042135e+09	6.05042135e+09	-2.03384627e-02
-2.08800160e-02	7.80082145e-03	3.41419252e-02	-2.77078169e-03
2.05474225e-03	3.26621727e-02	-1.15371180e-02	-3.49185989e-02
-4.87935734e-02	6.25833069e-02	5.19420672e-03	5.22533234e-03
6.20477073e-03	5.90827793e-03	5.02648362e-03	5.10457612e-03
-2.63552643e-03	-5.09155328e-04	-2.35078792e-03	-2.06329840e-03
-3.90370589e-03	-7.24981510e-05	-5.87087809e-03	-5.72553230e-03
-5.41844088e-03	-6.88724424e-03	-5.89208677e-03	-5.12117182e-03
-8.43123713e-03	-8.54414515e-03	-7.96407781e-03	-7.36062115e-03
-8.37475463e-03	-8.11637499e-03	1.02309499e-02	1.05077399e-02
1.24364005e-02	9.48468456e-03	1.06693368e-02	9.25387524e-03
4.90897946e-05			

Coefficients of best Ridge regression model:

3.64138505e-02	-1.18710225e-02	-1.95189253e-02	-4.17270862e-03
-4.88773056e-03	2.30506821e-03	1.73146581e-03	-1.92599575e-02
-2.02878220e-02	7.24198698e-03	3.12607174e-02	-1.35512077e-03
2.40019660e-03	3.32963236e-02	-1.34516985e-02	-3.41534593e-02

```
-4.83333290e-02    6.26421634e-02    4.55810959e-03    7.05603343e-03
5.31149352e-03     6.55297179e-03    4.07940780e-03    5.73830870e-03
-7.44071687e-03    -1.47700210e-03   -3.96813418e-04   7.75030524e-05
1.42403086e-03     -5.63869989e-03   -5.68978604e-03   -5.04223589e-03
-6.33277826e-03    -6.34795763e-03   -5.05043173e-03   -5.69026628e-03
-8.21905422e-03    -8.03874992e-03   -7.67389306e-03   -8.39767718e-03
-8.31853109e-03    -7.68542169e-03   8.06882525e-03    1.16697531e-02
1.12510325e-02     9.44663170e-03    1.07675764e-02    1.14383351e-02
1.56671192e-05
```

While comparing the best unregularized model and the corresponding ridge regularized model, we can note that the high values in range of 10^9 have been completely **reduced** to range of 10^{-2}. This tells us that the regularization penalizes the coefficients while trying to find a simple enough model that generalizes well in the unseen test set.

Coefficients of best Lasso regression model:

```
3.76679914e-02     -2.06497301e-03    -1.01029899e-02    -0.00000000e+00
-0.00000000e+00    0.00000000e+00     0.00000000e+00     -1.43123599e-02
-1.45868156e-02    1.13250295e-03     2.94954180e-02     -0.00000000e+00
0.00000000e+00     4.57365672e-02     -0.00000000e+00    -2.27732673e-02
-3.93860963e-02    8.05505233e-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
5.92791331e-06
```

From the coefficients we can see that Lasso model induces a lot of zeros in the values and hence produces a **very sparse** weight matrix. This aligns well with the theoretical inference earlier. Also, it is to be noted that the high values of range 10^9 in the unregularized model have been reduced to range of 10^{-3}.

Coefficients of best Elasticnet regression model:

```
3.80905930e-02     -5.36956408e-03    -1.27192853e-02    -0.00000000e+00
-0.00000000e+00    3.94194520e-03     8.69528894e-04     -1.71510813e-02
-1.78129180e-02    4.23637117e-03     2.95173339e-02     -0.00000000e+00
0.00000000e+00     5.07923700e-02     -0.00000000e+00    -2.24928711e-02
-3.89232244e-02    8.53884148e-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
-4.48653762e-05
```

From the coefficients of ElasticNet, we can see that it produces coefficients that's intermediate to Lasso and Ridge. While it incorporates the sparsity of Lasso with zeros, the number of non-zero values is higher compared to Lasso. This is because of the ridge part in Elasticnet that results in values in range of 10^{-2}.

4.4 Question 2b

Random Forest adds robustness and stability to a model by introducing bagging. Generally, decision trees tend to be unstable where small changes in the data caused drastic variations in the model. Decision trees tended to **overfit** their data. This problem is alleviated in random forest by using ensemble learning. Random Forest can be used in both classification as well as regression. In this section of the project, we apply random forest regression to predict the size of backup.

4.4.1 Question 2b(i)

To start off, we begin by taking a random forest model that has the parameters as given in the specification. The following plots show the fitted value vs true value and the fitted value vs residuals for the random forest regression. The table specifies the training error, test error and out of bag error for the model.

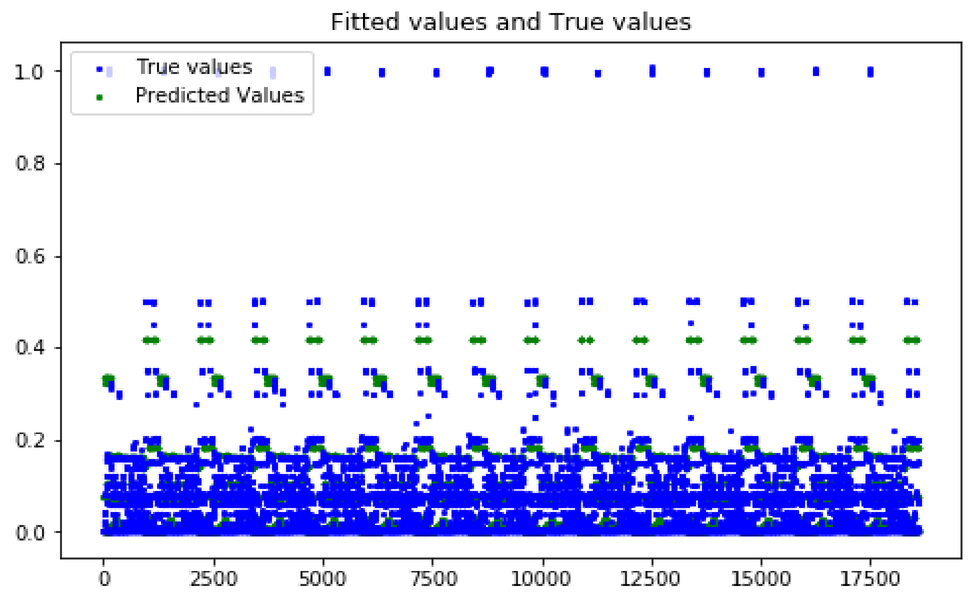

Figure 29: Plot of fitted values and true values for Random Forest Regressor

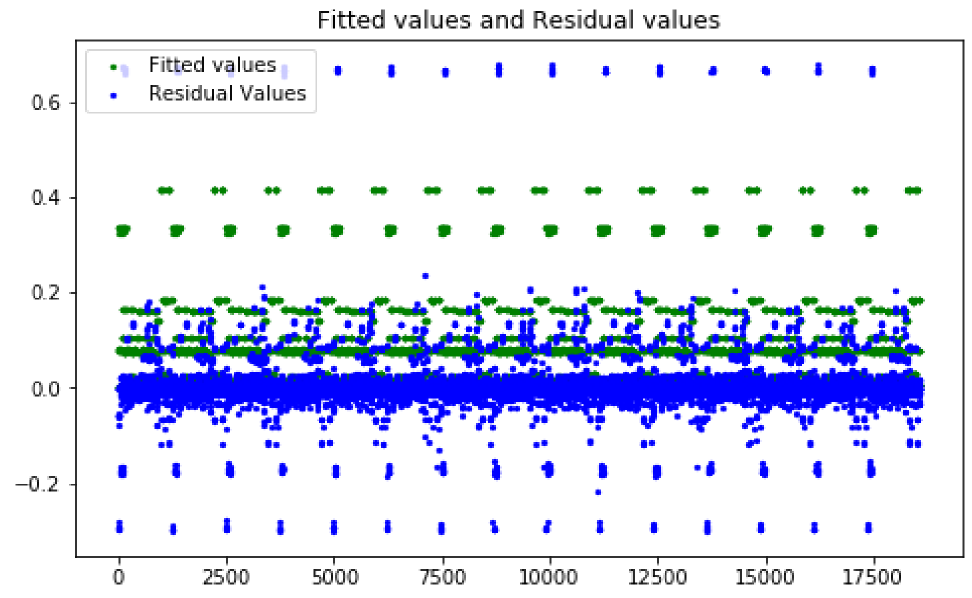

Figure 30: Plot of fitted values and residuals for Random Forest Regressor

	Random Forest Regression
Training RMSE	0.060285
Test RMSE	0.060481
Out of bag error	0.336300

Inference Firstly, the random forest regressor **performs better** than the linear regressor that we have seen earlier. This can be seen from the overlap of the true values and predicted values in the first plot, the small values of residuals in the second plot which corresponds to the blue patch in the middle and the low test RMSE of 0.06. There has been an improvement in the test error from the linear regression model. However, the test error and training error can be reduced further by optimizing the hyperparameters in the model such as maximum features, number of trees and maximum depth. In the subsequent parts, we will vary the hyperparameters to find the best parameters that will result in the lowest error.

The very small difference between the training error and test error shows that there is no overfitting. Since random forest is an **ensemble method with bagging**, overfitting is avoided and this can be seen from the small test error.

4.4.2 Question 2b(ii)

In this section, we will **vary the maximum number of features** and test error for different number of trees to determine the best value of the hyperparameters. The first plot shows the value of test RMSE for different max features and different number of trees. The x-axis corresponds to the number of trees and the different colors correspond to the different maximum number of features. Similarly, the second plot shows the **variation of out of bag error** for different max features and number of trees.

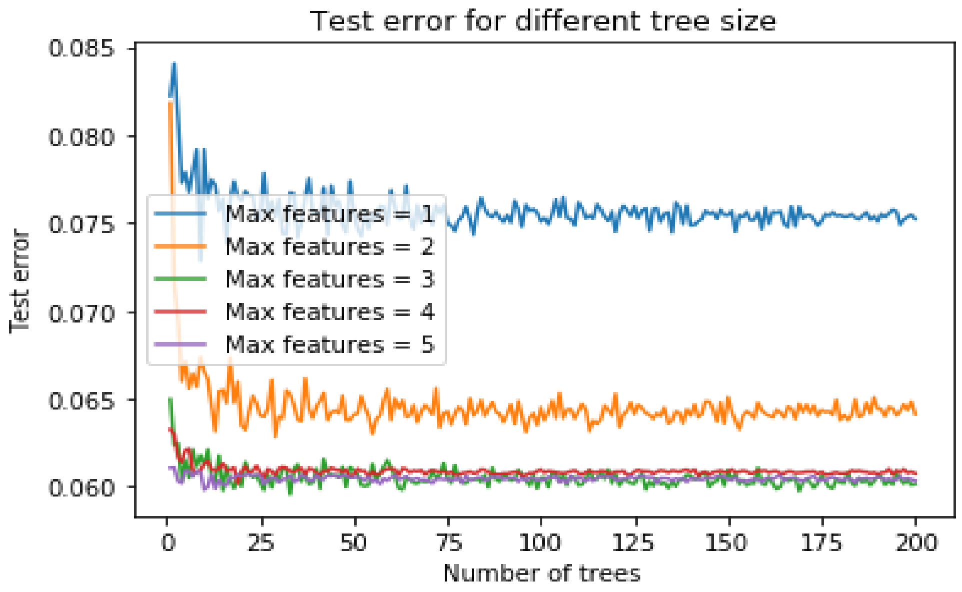

Figure 31: Plot of test error for different max features and varying tree size

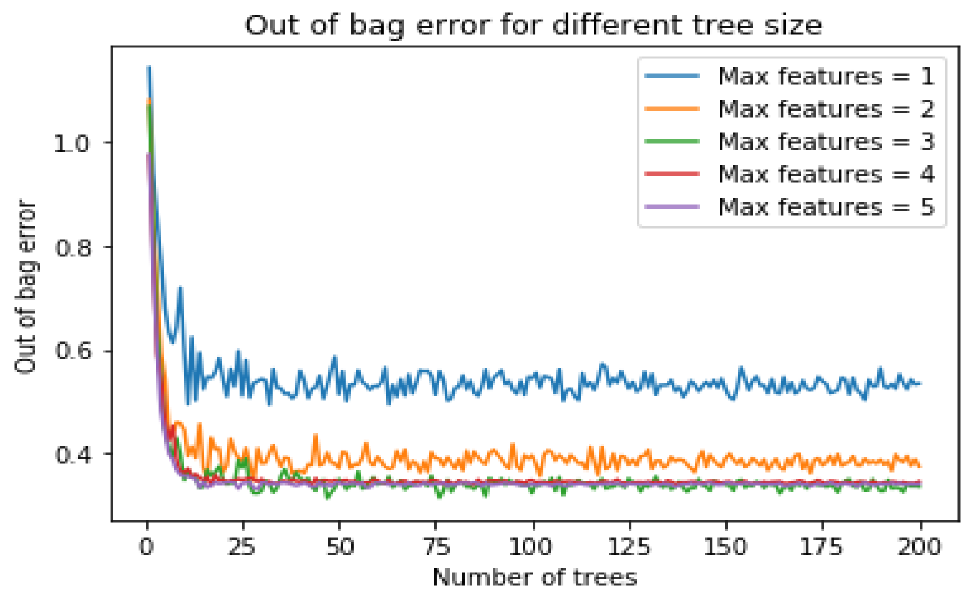

Figure 32: Plot of out of bag error for different max features and varying tree size

Inference From the first plot, we can infer the general pattern that as the maximum number of features is increased, the test error decreases. For very small number of maximum features such as 1 and 2, the value of test error is very large. For maximum features beyond 2, such as 3,4 and 5 the test error is pretty low and the lowest test error corresponds to **maximum features as 3**. This can also be seen from the second plot where the lowest out of bag error corresponds to maximum features as 3. Even for the out of bag error, the plots with max features as 1 or 2 perform very poorly as compared to all the other plots with maximum features greater than 2. Hence, a random forest regressor having maximum features greater than 2 will perform well, with the optimum number being 3.

Another significant inference from the plots above is the reduction in error as the number of trees increases and the saturation thereafter. As the number of trees increases from 1 to 10, there is an exponential decrease in both the test error and out of bag error. Beyond 10 trees, the error remains more or less the same with just minute fluctuations. This clearly shows the **effect of bagging** in reducing error and improving the **robustness** of the model. As the number of trees increases from a very small number to a larger number, the error in a single model is diminished by the averaging effect. Hence, we observe small errors beyond a certain number of tree size.

4.4.3 Question 2b(iii)

In this section, we will **vary the maximum depth** and test error for different number of trees to determine the best value of the hyperparameters. The first plot shows the value of test RMSE for different max depth and different number of trees. The x-axis corresponds to the number of trees and the different colors correspond to the different maximum depth. Similarly, the second plot shows the variation of out of bag error for different max depth and number of trees.

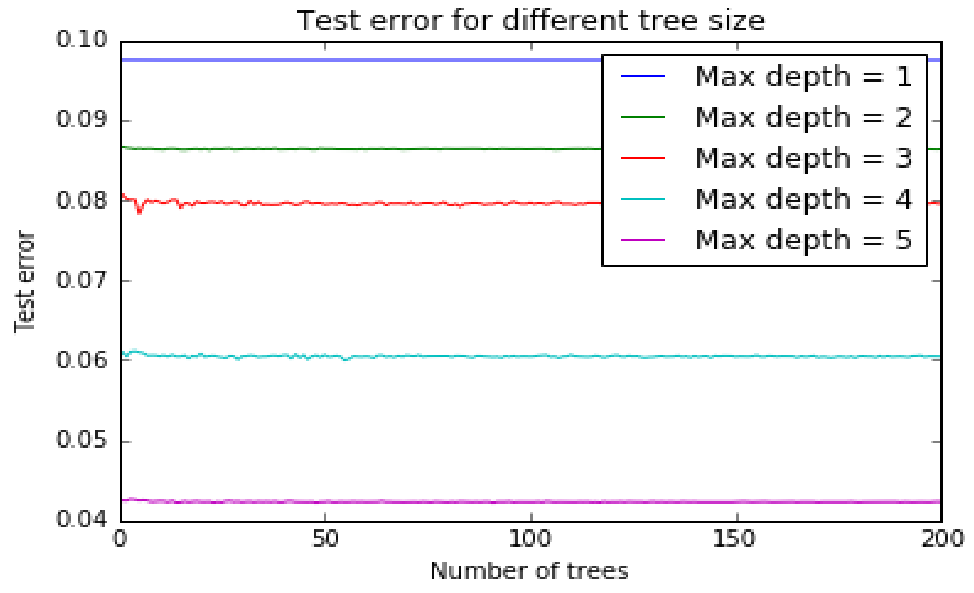

Figure 33: Plot of test error for different max depth and varying tree size

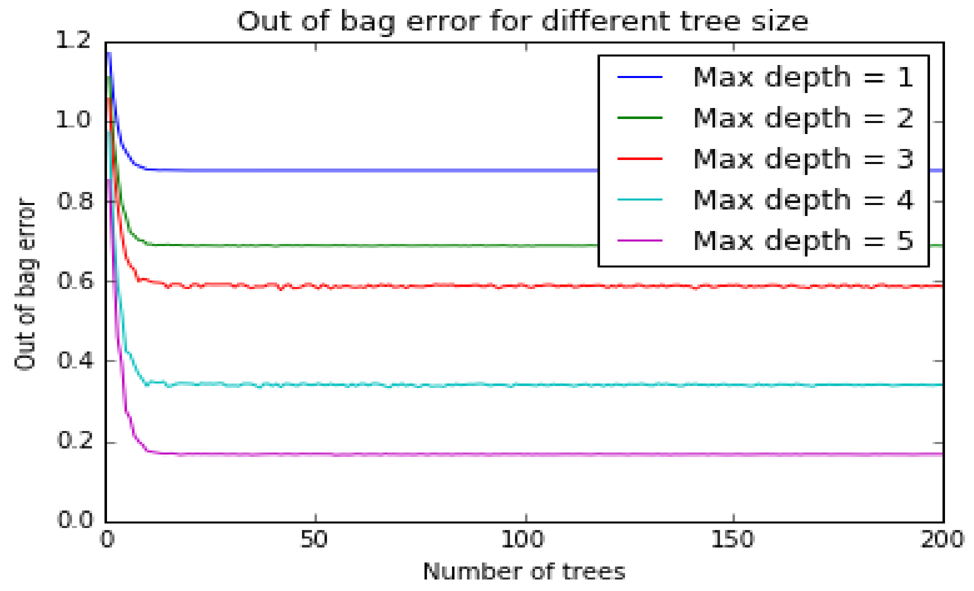

Figure 34: Plot of out of bag error for different max depth and varying tree size

Inference From the two plots above, we see that the error reduces as the maximum depth increases. There is a consistent improvement in the error as the maximum depth increases unlike the previous section where the improvement in error stopped after a certain point. Hence we realize that the **maximum depth** can be a significant factor in the performance of the random forest regressor. We find that both in terms of test error and out of bag error, the random forest regressor with depth as **5** performs the best with an test error around **0.042** and out of bag error around **0.192**.

Unlike the previous section, only the out of bag error reduces as the number of trees increase and the test error remains the same. The improvement in the out of bag error is also not as significant as the improvement seen in the previous section where we varied the maximum features. A reason for this might be that posing a constraint on the maximum depth has a **more restrictive** effect on the model and hence the models created by each of the tree in the random forest are not independent. We know that the random forest works on the assumption that the models are independent and hence taking an average of the outputs of the models help in reducing the error. However, if the models are **highly correlated** like in this case, the averaging over multiple models does not have a big impact.

4.4.4 Question 2b(iv)

The best random forest model was found using the hyperparameter search in the previous two sections. The best model corresponds to a random forest regressor with maximum depth as 5, maximum number of features as 3 and number of trees as 70. The model was fitted onto the input data and the importance of the different features are found and tabulated below.

Feature Importances				
Week number	Day of week	Backup start time	Work flow ID	File Name
0.00162685	0.32312067	0.19722801	0.19854327	0.2794812

Inference As we had earlier inferred through the **exploratory data analysis**, **feature selection** and **feature encoding**, the day of the week is the most important feature and the week number is the least important feature. This intuitively makes sense since the **network activity** generally depends on the day and it remains the same irrespective of which week we are in. By looking at the feature importance values we find that the next most important feature is the file name followed by the work flow ID and backup start time. Compared to all these features, week number has almost negligible feature importance.

4.4.5 Question 2b(v)

We take a single decision tree from the random forest regressor and visualize it and look at the various features and the criteria which are used to branch from each point in the tree.

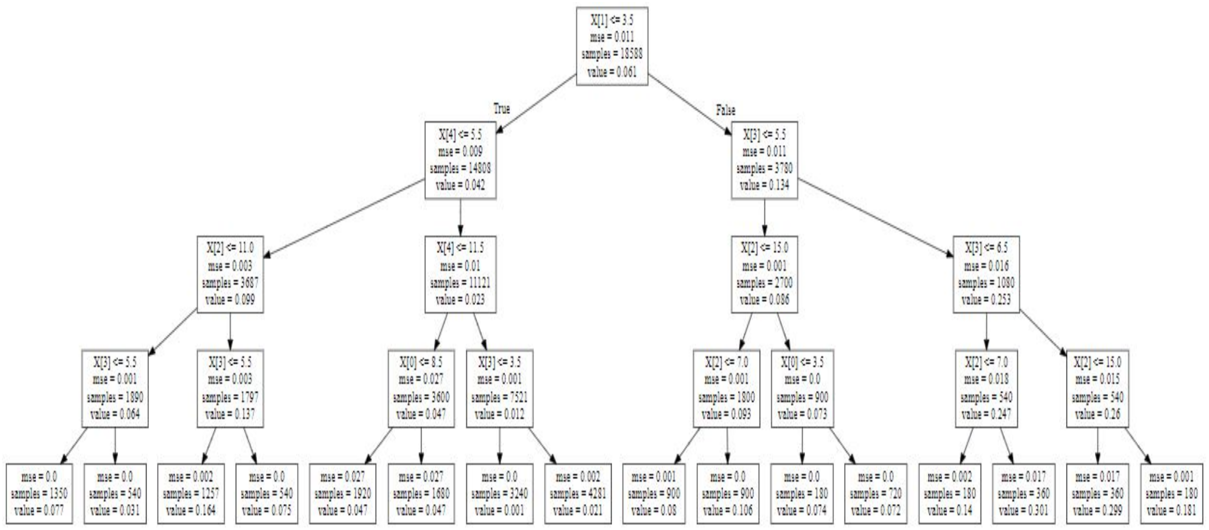

Figure 35: Visualization of a single decision tree

Inference At the **root** of the tree is the feature **'Day of week'**, which was the most important feature as we found in the previous section in the table of feature importances. This makes sense intuitively since the **most important feature** has to be used at the first step to split the tree into two branches. The criteria that is used to measure the quality of a split here is **mean square error**. By using the most important feature to split it first, we are able to minimize the mean square error by the maximum amount. The more important features are seen near the root of the tree such as day of week, file number and workflow id and the least important feature of week number is seen only near the **leaf of the tree**.

4.5 Question 2c

In this section, we implement a **neural network regression model** with one hidden layer. There are several factors such as activation function and number of hidden units that can be varied. Here, we vary the hidden layer size from 1 to 200 for the different activation functions such as **logistic, tanh and ReLU**, and plot the test RMSE as the hidden layer size varies.

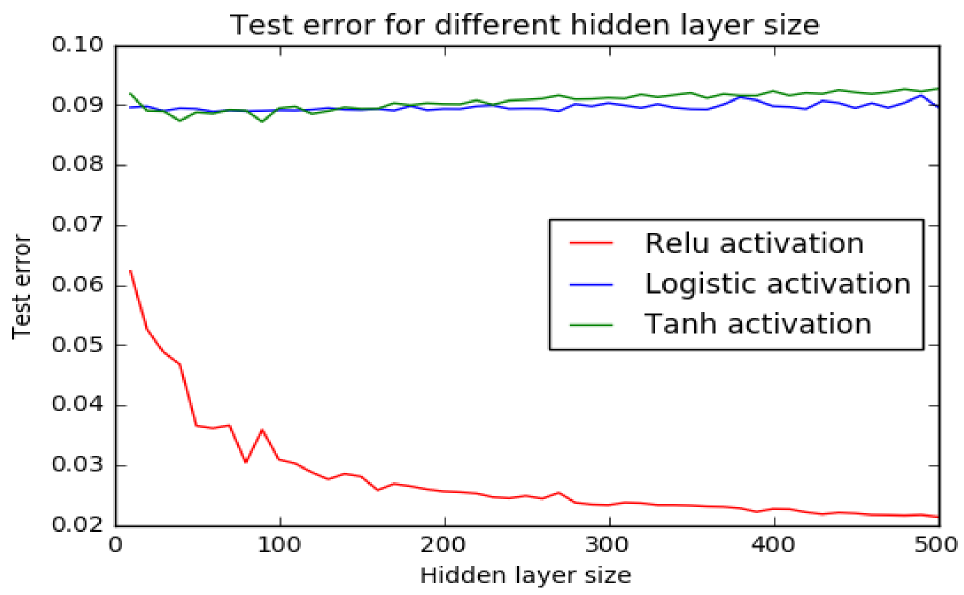

Figure 36: Plot of test error as hidden layer size varies for different activations

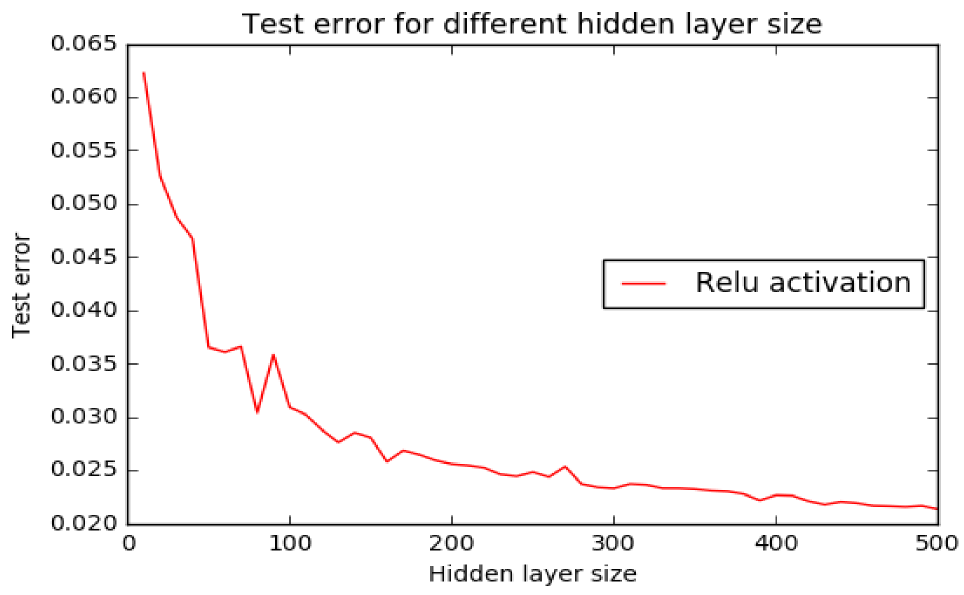

Figure 37: Plot of test error as hidden layer size varies for Relu activations

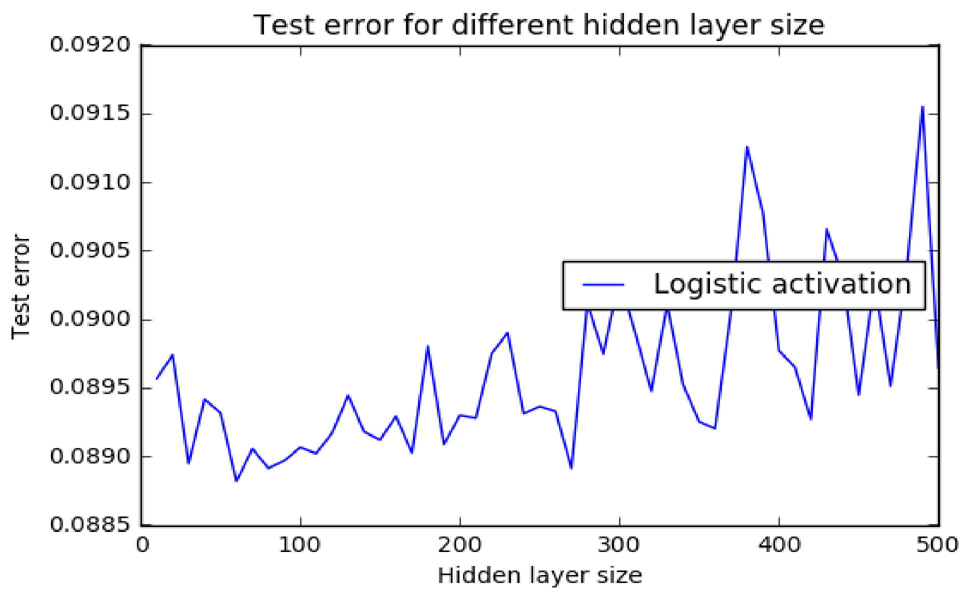

Figure 38: Plot of test error as hidden layer size varies for logistic activations

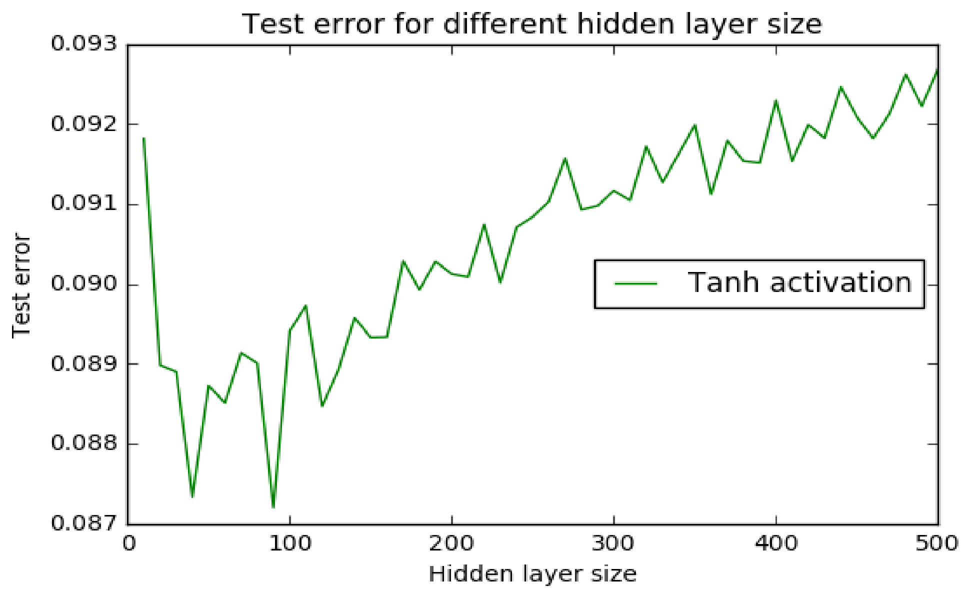

Figure 39: Plot of test error as hidden layer size varies for tanh activations

Inference There are a couple of takeaway points from the graph above. Firstly, there is an obvious improvement in performance for the **ReLU activation function** compared to the other two activations. Theoretically, we know that the ReLU function performs the best and it is the most commonly used activation function. In the case of a logistic function, there are several limitations associated with it, which results in a degraded performance. Firstly, the function **saturates** when the input is very large or very small. Therefore, the gradient becomes zero in those cases and no learning occurs (**case of vanishing gradients**). Hence, when the input is large, either in the positive or negative direction, the unit becomes ineffective in learning. Another issue is that the sigmoid unit is **not zero-centered**. This leads to **zig-zagging** during gradient descent which causes it to converge to the minima slowly.

In the case of tanh units, the issue of zig-zagging is avoided since it is **zero-centered**. However, even the tanh units suffer from the problem of **saturation** at the extremes, and hence no learning occurs at those regions. Both the tanh and sigmoid units perform poorly due to the saturation problem and this can be seen from the figure where there error with relu units keep decreasing while the other two are more or less the same.

Theoretically, as well as empirically, the ReLU activation units perform the best since they are not faced with the issue of saturation as the input becomes large. The only issue associated with ReLU units is that it is **not differentiable** at $x = 0$. However, that is addressed by using sub-gradients and it does not pose a big problem. In practice, learning with ReLU units **converge faster** compared to sigmoid and tanh, and this can also be seen from the graph above.

Best Combination The best combination that gave the lowest test error is ReLU activation unit with 500 hidden units.

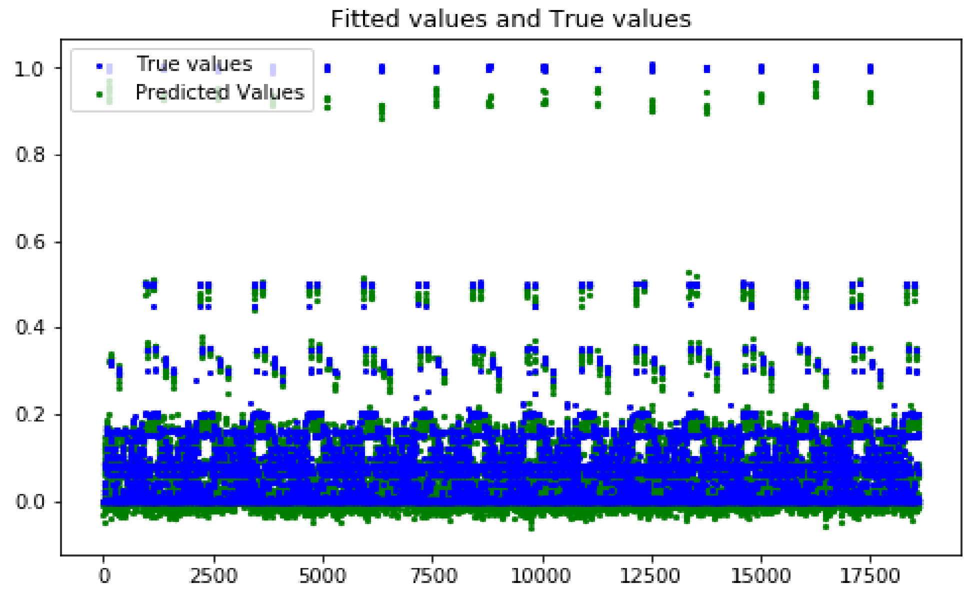

Figure 40: Plot of fitted values and true values for the best combination of neural network

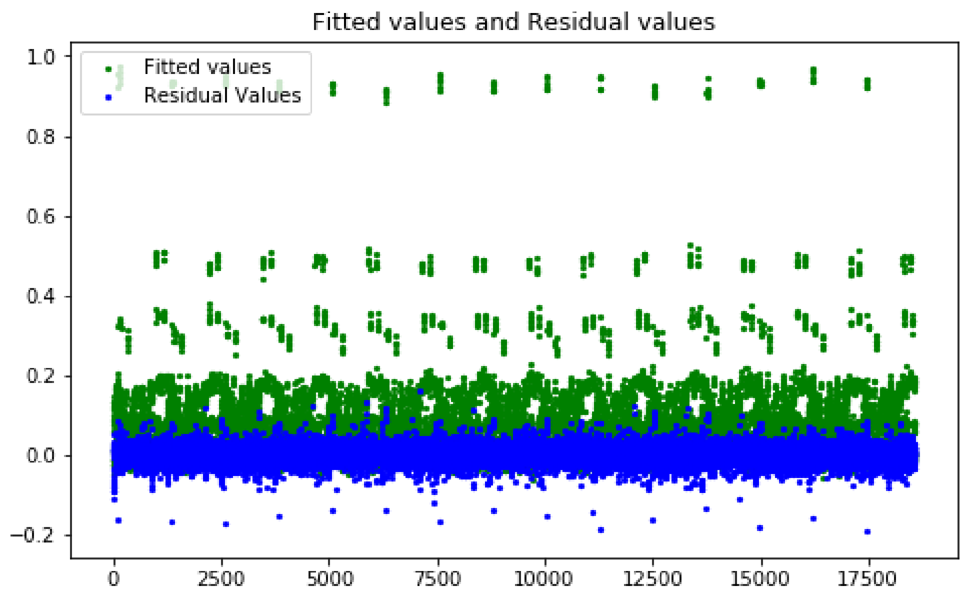

Figure 41: Plot of fitted values and residuals for the best combination of neural network

	Neural Network Regression
Training RMSE	0.012239
Test RMSE	0.024261

The graphs above show the fitted value, true value and residuals for the best combination, which is relu activation with 500 hidden units. The neural network does a good job and has a low training and testing RMSE. As seen from the graph, the residuals are also very small which correspond to the small blue patch in the middle. This shows that the regressor has done a good job in predicting the output values well.

4.6 Question 2d

In this section, we'll try to fit models on each of the workflows separately to see if there is any improvement in performance.

4.6.1 Question 2d(i)

We begin with a simple linear regression model and we separate the data based on the workflow ID. Hence, we end up with 5 subsets of data segregated based on the workflow ID. For each of these we fit a linear regression model and plot the fitted values vs true values as well as the fitted values vs residuals. For each of the 5 models, we also calculate the training and test error and report it in a table at the end.

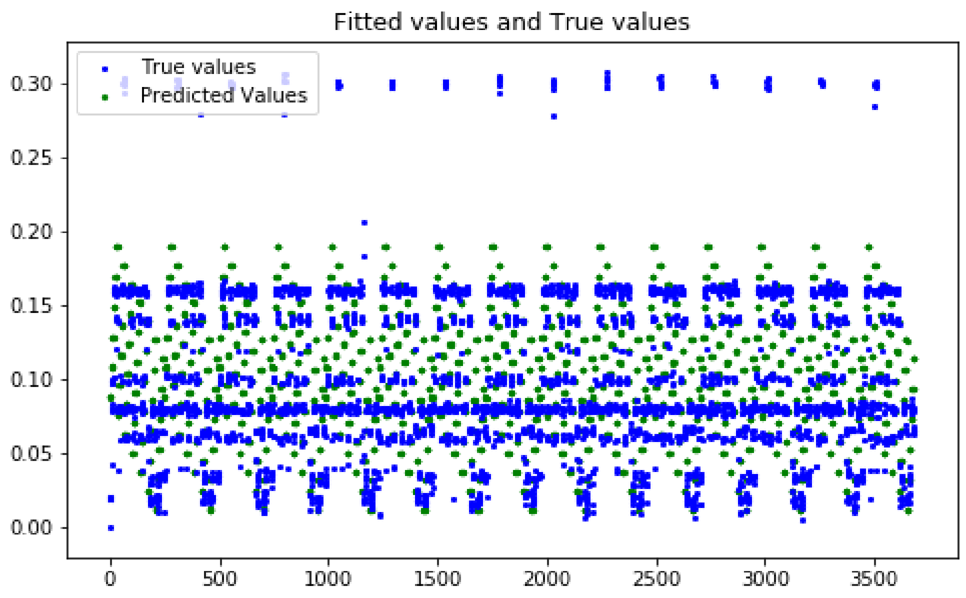

Figure 42: Plot of Fitted values and the True values for Linear Regression model for workflow ID = 0

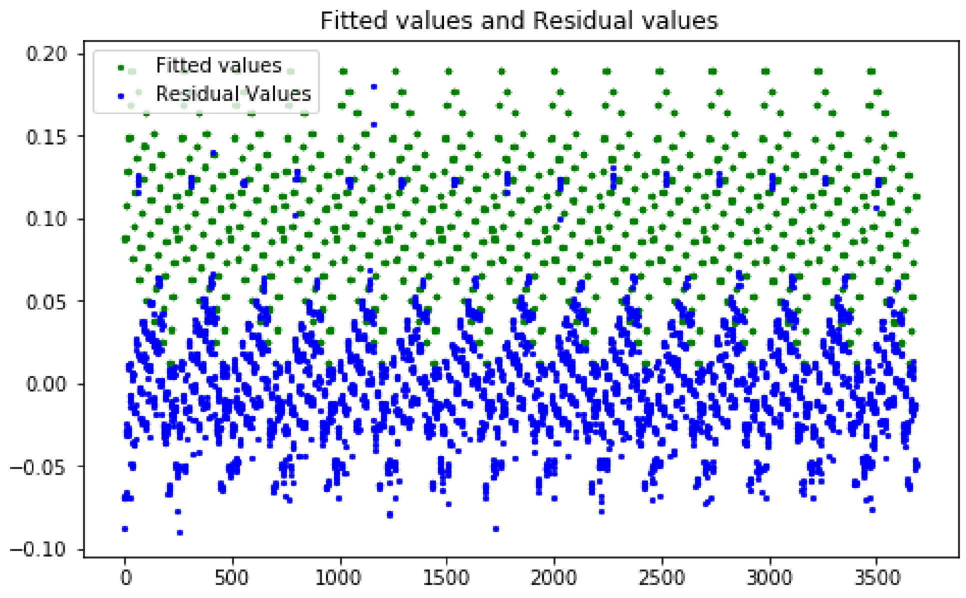

Figure 43: Plot of Fitted values and the Residual Values for the Linear Regression Model for workflow ID = 0

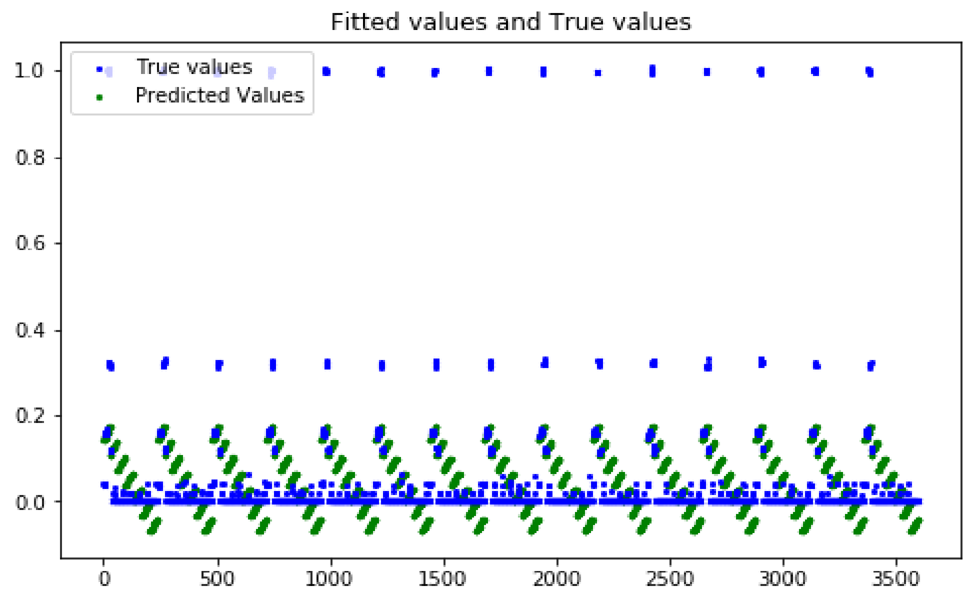

Figure 44: Plot of fitted values and True Values for the Linear Regression Model for workflow ID=1

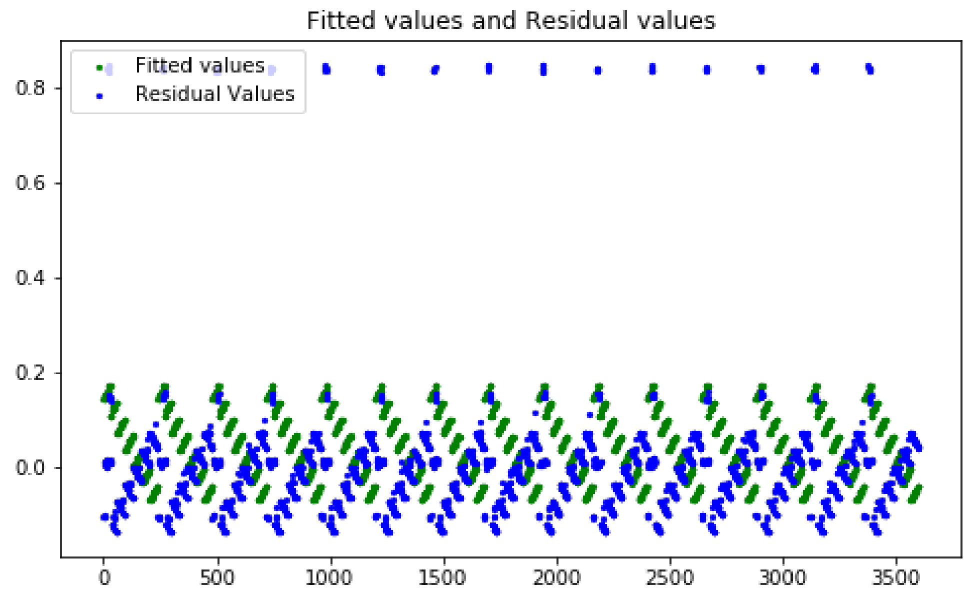

Figure 45: Plot of fitted values and Residual Values for the Linear Regression Model for workflow ID=1

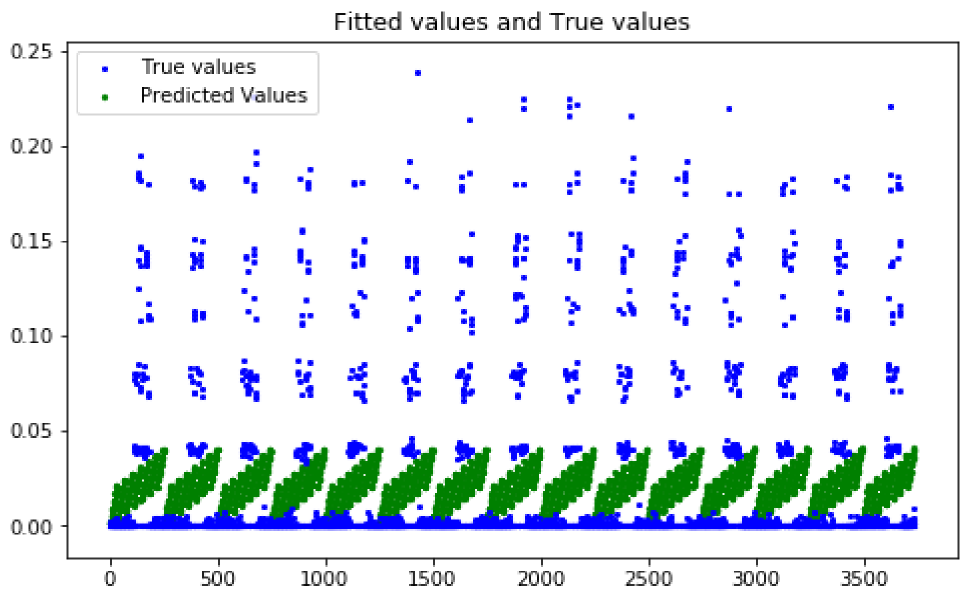

Figure 46: Plot of fitted values and True Values for the Linear Regression Model for workflow ID=2

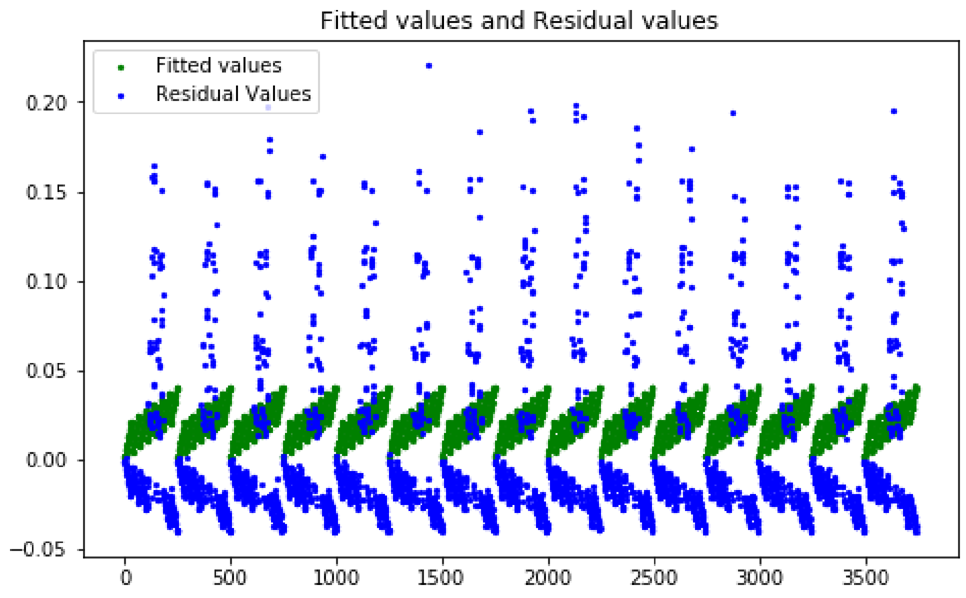

Figure 47: Plot of fitted values and Residual Values for the Linear Regression Model for workflow ID=2

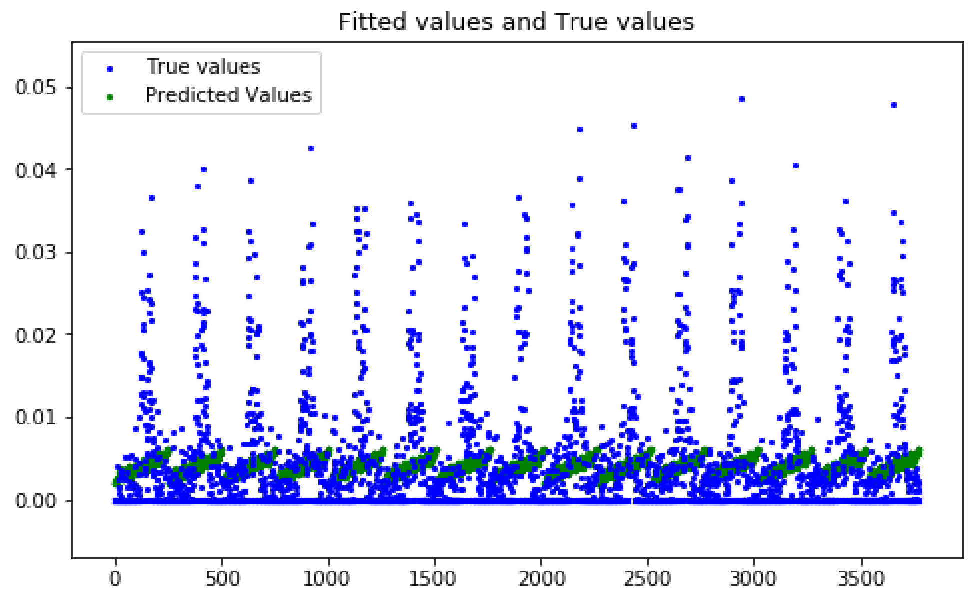

Figure 48: Plot of fitted values and True Values for the Linear Regression Model for workflow ID=3

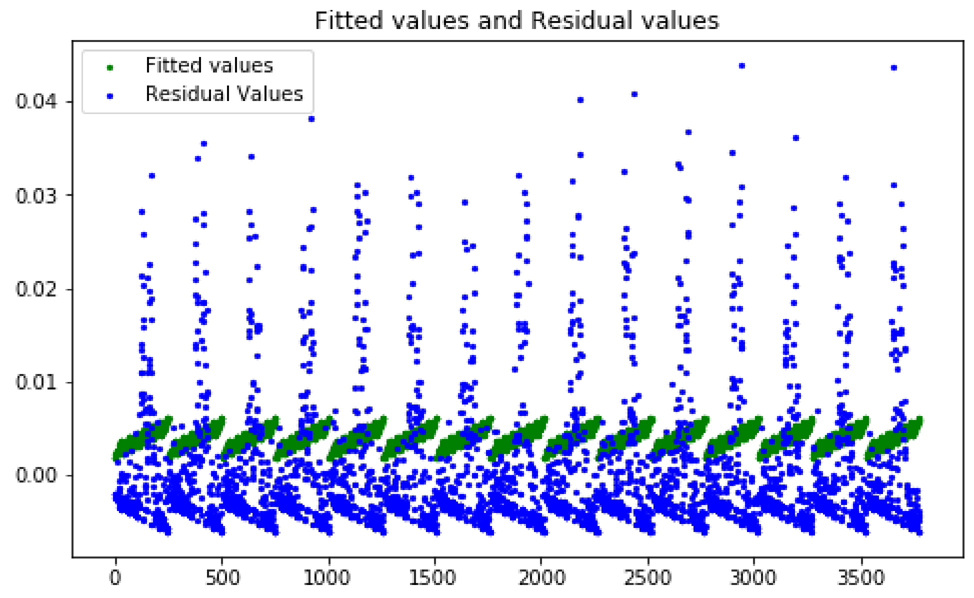

Figure 49: Plot of fitted values and Residual Values for the Linear Regression Model for workflow ID=3

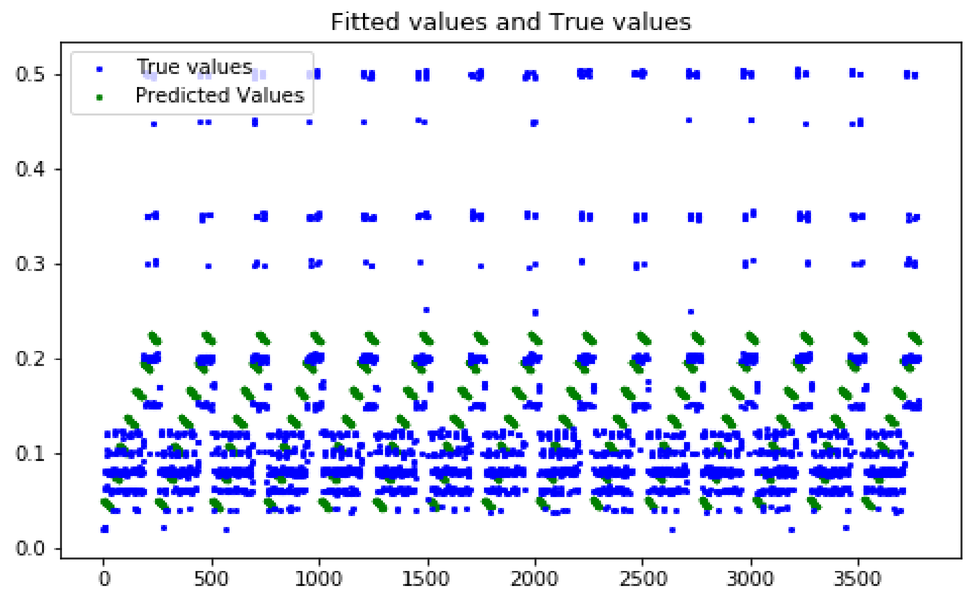

Figure 50: Plot of fitted values and True Values for the Linear Regression Model for workflow ID=4

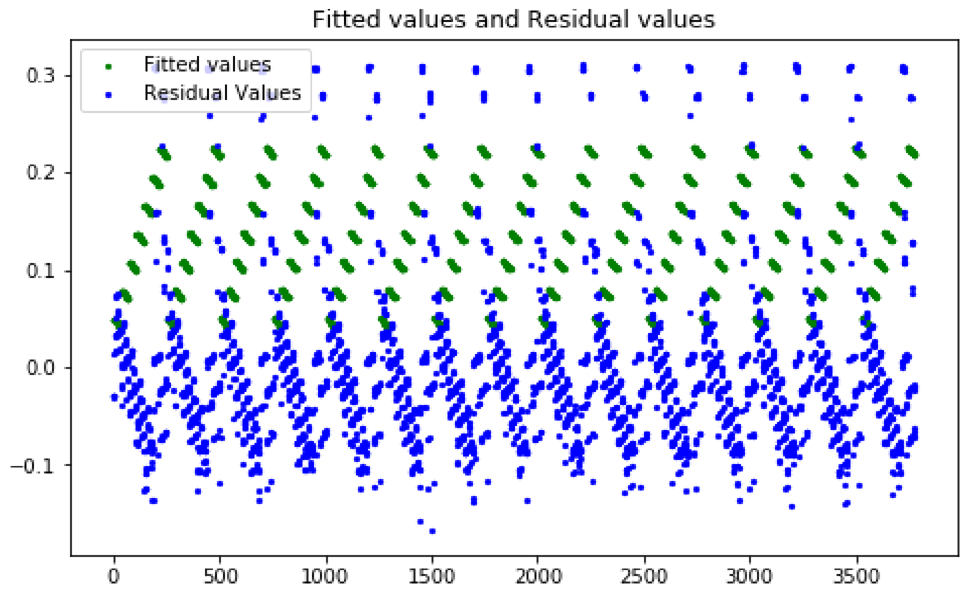

Figure 51: Plot of fitted values and Residual Values for the Linear Regression Model for workflow ID=4

Workflow ID	Training RMSE	Test RMSE
0	0.035835	0.035886
1	0.148766	0.148918
2	0.042909	0.043066
3	0.007243	0.007260
4	0.085921	0.085990

Inference From the graphs and the table, we can see that after separating the data based on the workflow ID, the **fit has improved** on most of the workflows except workflow ID 1. For some of the subsets of data such as workflow ID 0 and 2, the training and test RMSE has reduced by a huge margin and the fitted values are very close to the actual values. This can also be seen from the residual graph, where most of the residual points lie close to 0.

The information obtained from the table and graphs tell us that after splitting the data based on the workflow ID, there is a distinct pattern in some of the subsets of data such as workflow ID 0 and 2, which is exploited by the regressor to perfectly fit the data points. Whereas in the case of workflow ID 1, the data corresponding to it does not lie on a line and hence does not yield good results through the linear regression line. Probably a more complex regression model such as a polynomial function might fit better.

4.6.2 Question 2d(ii)

For each of the workflow ID, we will try to fit a **more complex regression model** using a polynomial function to see how the training and test errors vary. We have plotted the training and test errors as the polynomial degree is increased from 2 to 9 for each of the workflows separately.

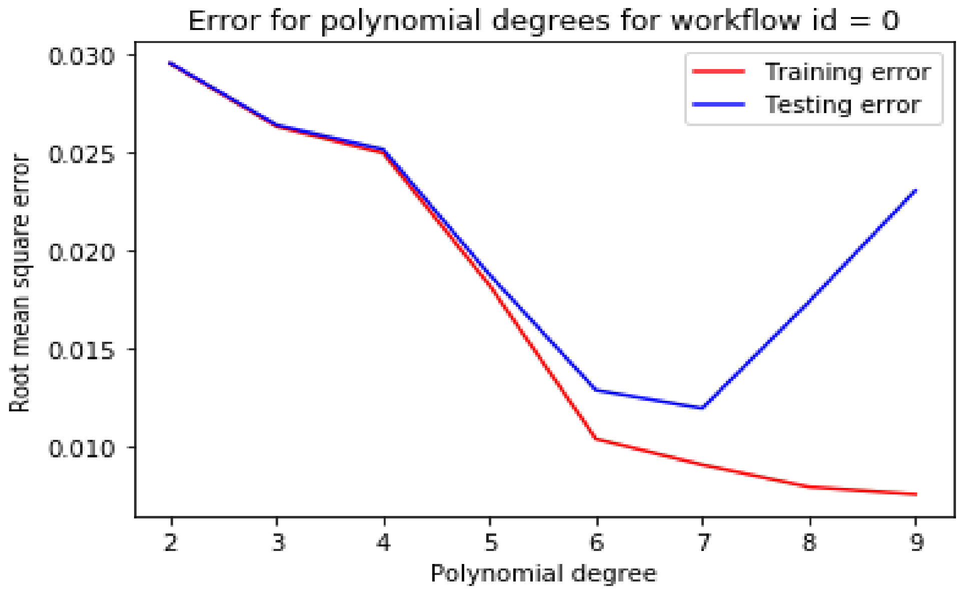

Figure 52: Plot of errors for different polynomial degrees for workflow id=0

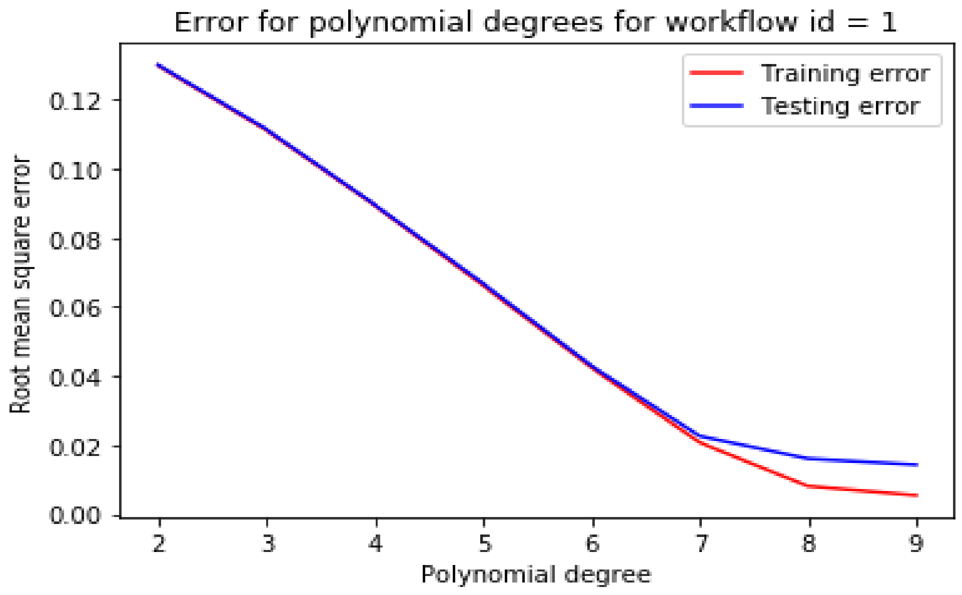

Figure 53: Plot of errors for different polynomial degrees for workflow id=1

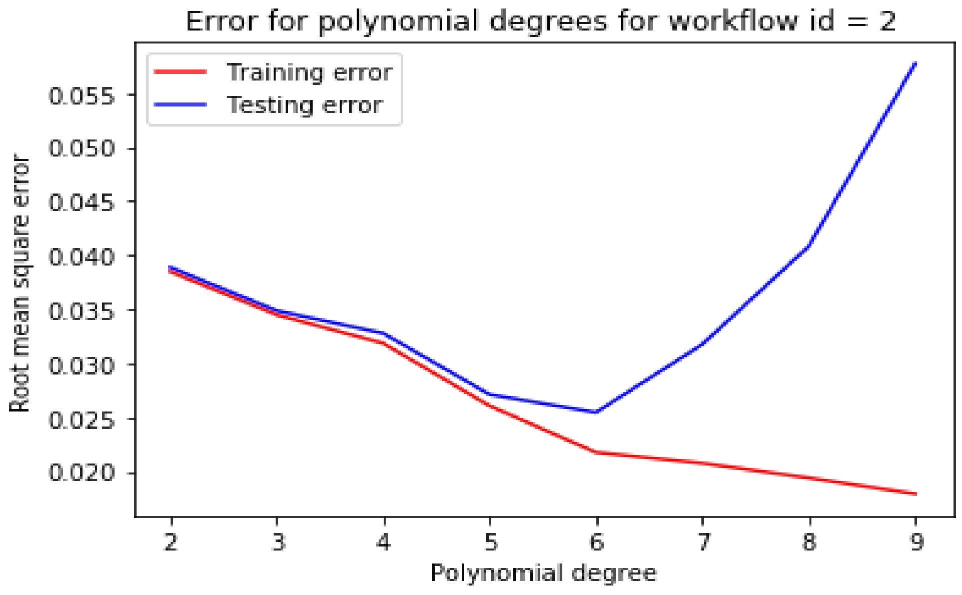

Figure 54: Plot of errors for different polynomial degrees for workflow id=2

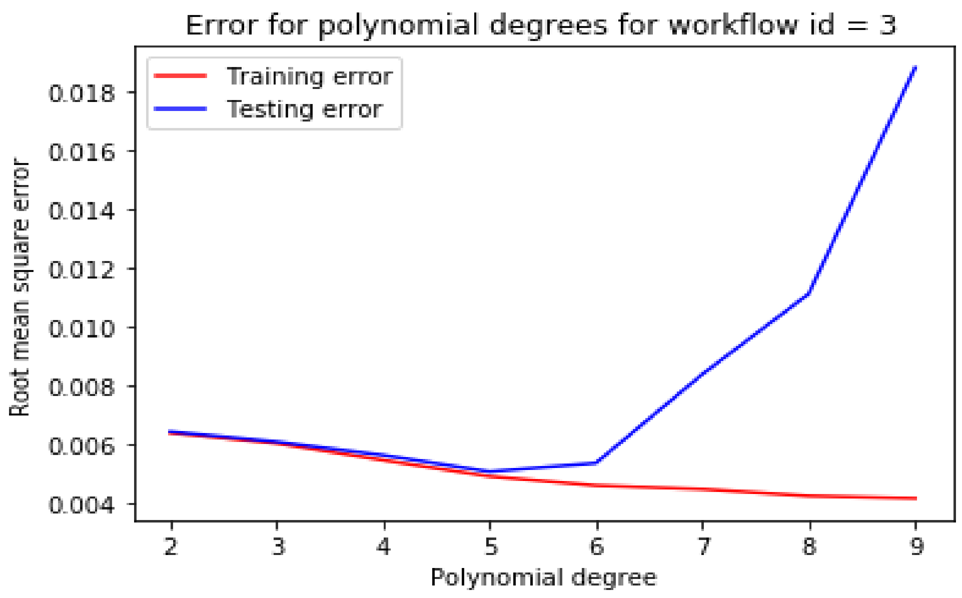

Figure 55: Plot of errors for different polynomial degrees for workflow id=3

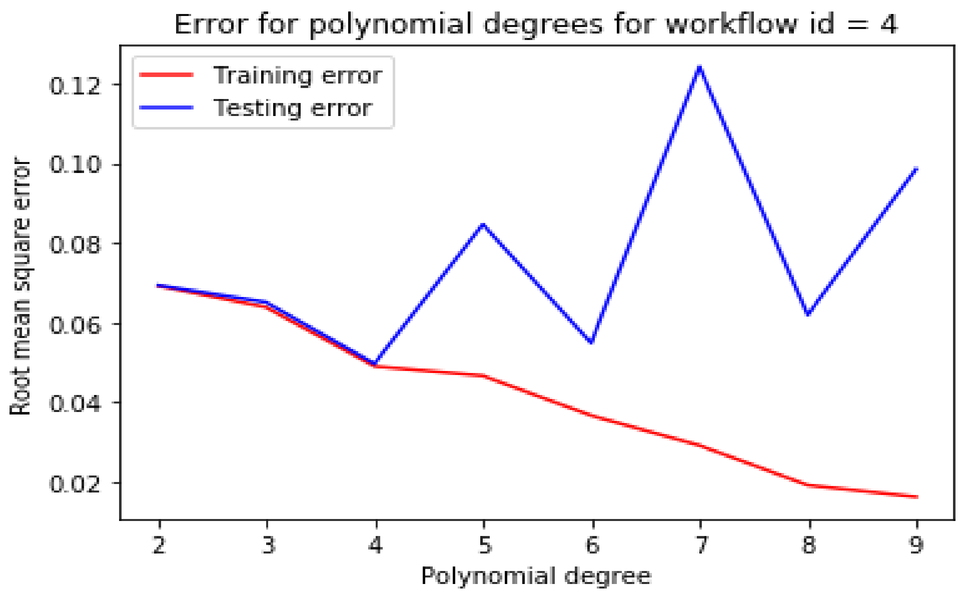

Figure 56: Plot of errors for different polynomial degrees for workflow id=4

Inference A general pattern that is observed from the plots above is that the test error increases after a certain point while the training error continues to decrease. This is due to the fact that as the polynomial degree increases, the model complexity increases and hence the model **overfits** the data. Therefore, beyond a point the model **does not generalize** resulting in a high test error and a low training error. Therefore we need to determine a polynomial degree to operate at, such that both the training error as well test error are low. From the graphs above, we find that beyond a threshold of polynomial degree 6, the generalization error of the model increases.

One of the common ways of avoiding overfitting and increased model complexity, is by using a **validation set**. Generally, the data is split into train set, test set and a validation set. The validation set helps us to determine how well the **model will generalize** and also allows us to do **hyperparameter tuning**. When a model tries to overfit, it tries to learn all the **inherent noise** along with variance in the data. When we do cross-validation, the training set changes each time and hence the noise too changes, and thus forcing the model away from overfitting.

In order to effectively exploit the entire training set for validation, k fold cross validation is done. The training set is split into k folds. Keeping one of the folds for validation, the model is trained on the other folds and validated using the one remaining fold. By repeating this using all the folds, we are effectively using the entire training set for validation and this helps to add robustness and further decrease the generalization error. Therefore by using cross validation, we are able to control the complexity of the model and reduce overfitting.

4.7 Question 2e

In this section, we implement a **K-Nearest Neighbor** regressor which will predict the value based on the predictions of the nearest neighbors. During the training process, the given datapoints are stored in memory and during the testing process, the nearest datapoints to the given test point is found and the value is predicted based on the predictions of the nearest neighbors. Firstly, we need to define the distance metric which is being used. In this case, we have used the **Euclidean distance** as the distance metric. Since distance becomes an important metric, we need to **normalize** the data before using it. We used **StandardScaler** to normalize the data before fitting the model. The following table reports the training error and test error for different values of k.

Number of neighbors	Training RMSE	Test RMSE
1	0.0	0.017636
2	0.008762	0.014933
3	0.010126	0.014295
4	0.010748	0.013828
5	0.011123	0.013586

Inference An interesting observation that is made here is the fact that the training error is zero when the number of neighbors is just 1. This can be explained by understanding the notion of training in a K Nearest Neighbor regressor. Unlike other regression technique, the training process of a KNN regressor does not create a model but rather stores the training data points in the memory. When we are evaluating the training error of the model, we are predicting on the training set once again. During this process, the **nearest training point** for each new point from the training set is that point itself. Hence, the predicted value is exactly the same as the actual value and thus the error is zero. Therefore, the entire notion of training error in a K Nearest neighbor regressor is meaningless and we just need to look at the testing error to evaluate the effectiveness of the regressor. In the following graph, we have plotted the testing error for different values of the number of neighbors as it varies from 1 to 10.

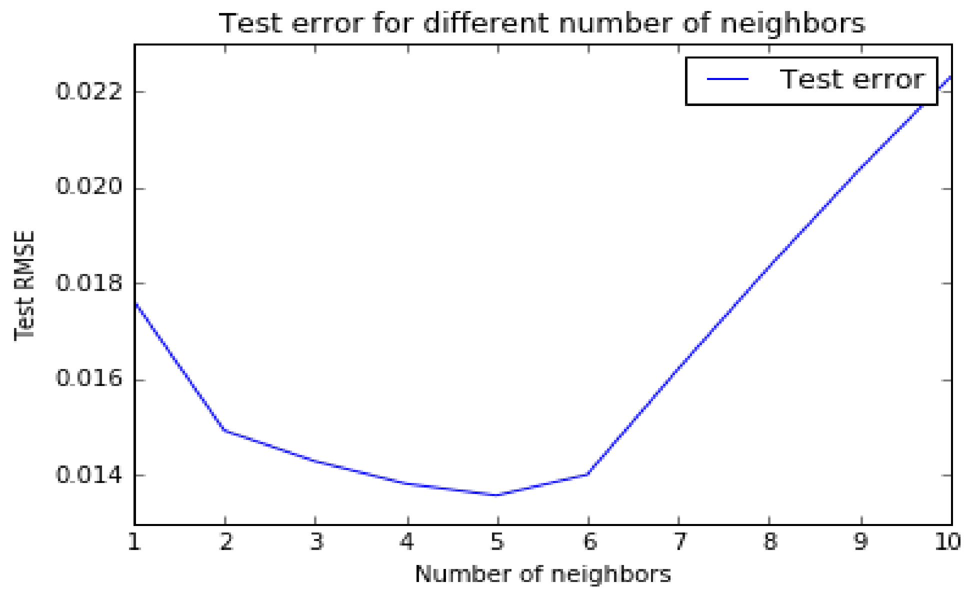

Figure 57: Plot of errors for as number of neighbors increase

Inference As seen from the plot, the test error initially decreases and then starts to increase again. The test error is minimum when the number of neighbors is 5. As the number of neighbors goes beyond 5, it starts to take into account a lot of neighbors who may actually be far off in the **high dimensional space** and hence the test error increases. Ideally, we want to have the right number of neighbors who are close by to determine the predicted value and ignore the effect of all the points which are far off. Hence, the best value of number of neighbors is **5**.

The following graphs show the plot of fitted value vs true value and fitted value vs residuals for the KNN regressor with the best parameters (k=5).

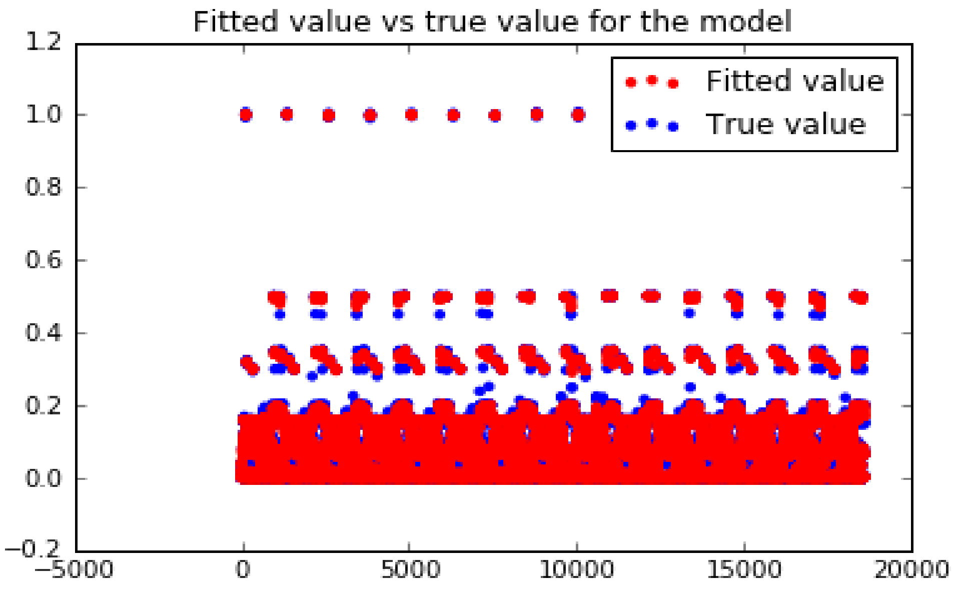

Figure 58: Plot of fitted values and true values for KNN regression

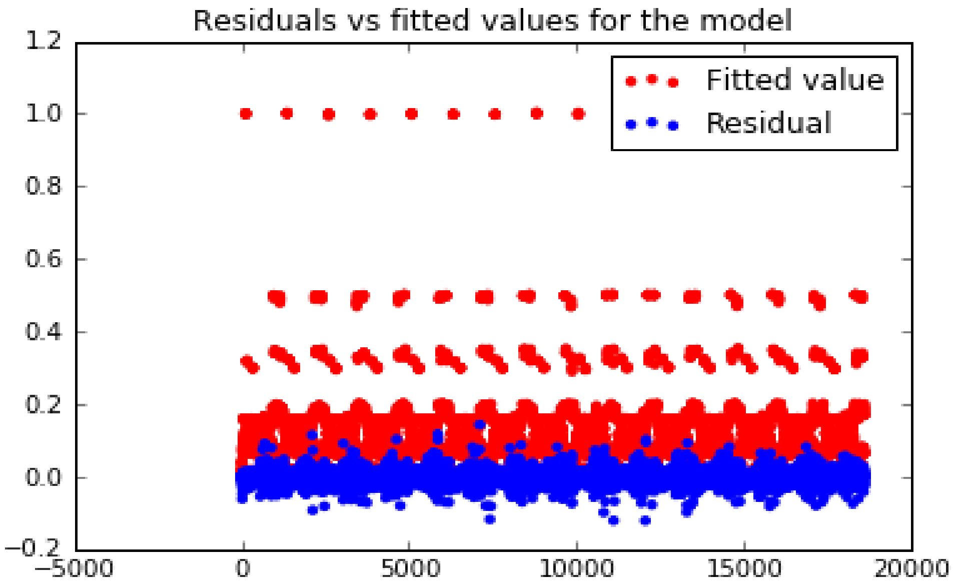

Figure 59: Plot of fitted values and residuals for KNN regression

	KNN Regression
Training RMSE	0.011123
Test RMSE	0.013586

Inference As seen from the graphs and the table, the **KNN regressor** does a very good job of predicting the output values. This regressor produces one of the lowest values for the test RMSE and fits the data well. It can also be seen from the **residual plot** where most of the residuals, corresponding to the blue points lie close to zero irrespective of the fitted points. The first plot also shows that most of the fitted data points overlap the actual data points and the regressor does exceedingly well especially after determining the best value of k through the hyperparameter search.

4.8 Question 3

Throughout this analysis, we have dealt with several regression models such as **linear** regression, linear regression with **regularization**, **polynomial** regression, **random forest** regression, **neural network** regression and **KNN** regression. The linear regression model was the simplest of all the models and thereby it had the highest test error. By introducing regularization, we were able to reduce the test error by preventing overfitting, though the improvement was not huge. By separating the dataset based on the workflow ID, we saw that the model was able to fit better. The fit was further improved by increasing the model complexity using a polynomial regression model. In order to prevent overfitting in that case, we found the best value of degree of polynomial beyond which the test error started to increase. This polynomial regression model performed really well and showed good results.

We then analyzed the random forest regression model. This model was inherently **robust** due to the **bagging** and hence it produced low test error. It was able to overcome the limitations of a single decision tree regressor by using multiple models and taking the average of the results. We were able to improve the model accuracy of this random forest regressor by determining the optimum values for the hyperparameters such as **number of trees, maximum number of features and maximum depth**. Eventually, this random forest regression model with the optimum hyperparameter setting produced very good results, achieving the lowest test RMSE compared to the previous models.

Another huge advantage with the **random forest** regression model is its capability of **handling categorical data**. In a random forest regression model, there isn't a need for any type of encoding, be it scalar or one hot. When a categorical data is one hot encoded, it increases in the dimensionality of the features and introduces sparsity in the data. The resulting sparsity causes the continuous variables to be assigned higher feature importance causing a performance degradation. One hot encoding also erases important structures in the underlying representation by splitting a single variable into multiple variables. Random Forests are able to overcome these limitations by their inherent capacity to deal with categorical data directly without the need for one hot encoding. Hence, random forest regressor can be considered as the best model for dealing with categorical data.

Neural network regressor is one of the most advanced techniques. Even a simple model with just one hidden layer with a few neurons produces such good results. Firstly, it overcomes the limitation of any linear regression model by including a **non-linear activation** function. Especially, the ReLU activation function performs exceptionally well since it does not have the **saturation problem** and this can also be verified from the analysis we did, where we compared the performance of the different activation functions.

With an increase in the number of neurons, more neurons are able to capture more features. As the number of layers increases, a more problem-specific search for features is made possible. Hence, a neural network architecture with multiple layers with several neurons has a very high capability in both classification and regression tasks. In our analysis, despite limiting the number of layers to just one, we got excellent results with a test RMSE of **0.024**. This is one of the lowest test errors among all the models.

The KNN regression model performs well on the data despite being a simple model. One of the reasons why this model works well is that we have only 5 features. Since we have only 5 features, we are operating in a 5 dimensional space. When the number of features is large, we will be in a higher dimension and this will introduce the problem called **curse of dimensionality**. In regression techniques such as KNN, where distance is calculated in the prediction process, if the number of dimensions is large, the notion of distance becomes meaningless due to the huge amount of space between the data points. Hence, when the number of features is high, KNN regression models do not perform well due to the curse of dimensionality. However, in this case since we have only 5 features, the KNN regressor does a good job of predicting the output value. Also, the input features are normalized in order to prevent a single feature from dominating the prediction result.

When the data is sparse, with a lot of features compared to the number of data points, the problem becomes underdetermined. In such cases many of the extracted features may be irrelevant. The simple linear regression model using ordinary least squares becomes inefficient and we need to use **regularization in order to handle the sparsity**. The most efficient regularization technique for handling sparse data is the **Lasso (L1) regularizer**. The Lasso regularization is able to produce sparse solutions as its output and hence can act as a feature selector. Since the **elastic net regularization** method effectively incorporates the Lasso regularization, it can also be used to handle the sparse data.

Finally, looking at the test RMSE results and the plots of the residuals and the fitted values, we can say that the **KNN regression model and neural network regression model** produce the best results. After determining the best parameters for the two models, both the models produced excellent results with a 8 time improvement in the test error from the initial linear regression model.

5 Conclusion

In this project, we have explored regression analysis using several regression models. Starting from the simple linear regression model to the more complicated neural network regression model, we have implemented and analyzed the pros and cons of each model. We also performed hyperparameter tuning for several models to determine the best parameters, to improve the model accuracy.