EE 219 Project 1: Regression Analysis

Winter 2017

January 30th, 2017

Arunav Singh (304760844)
John Moon (204774912)
Jiawen Yu (204753330)

Network Backup Dataset

1) Plot the actual copy sizes of all the files on a time period of 20 days for each workflow.

The plots of the five workflows (0-4) are shown below.

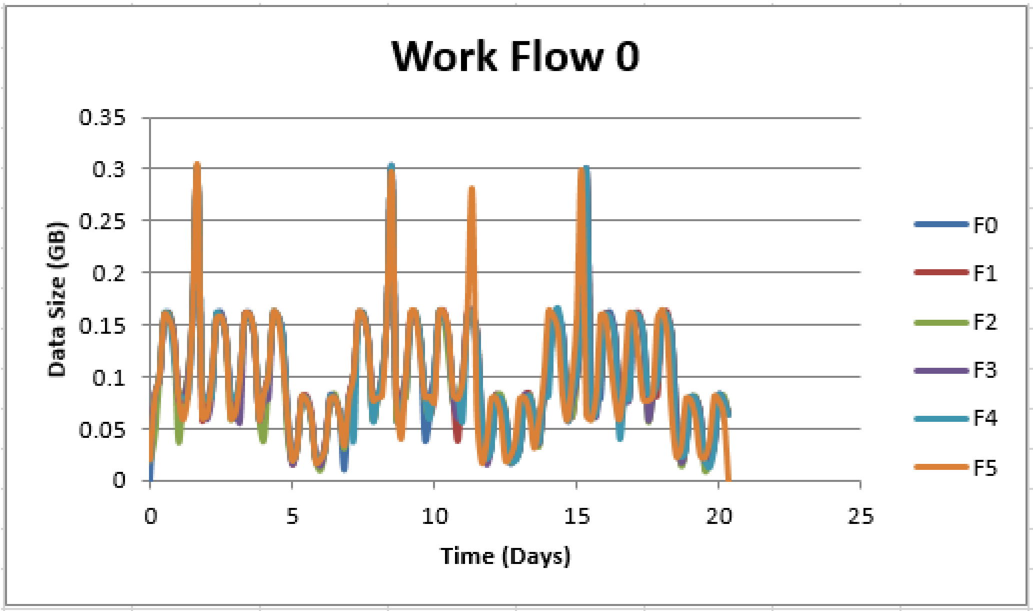


Figure 1: Plot of time versus backup data size for work-flow-0

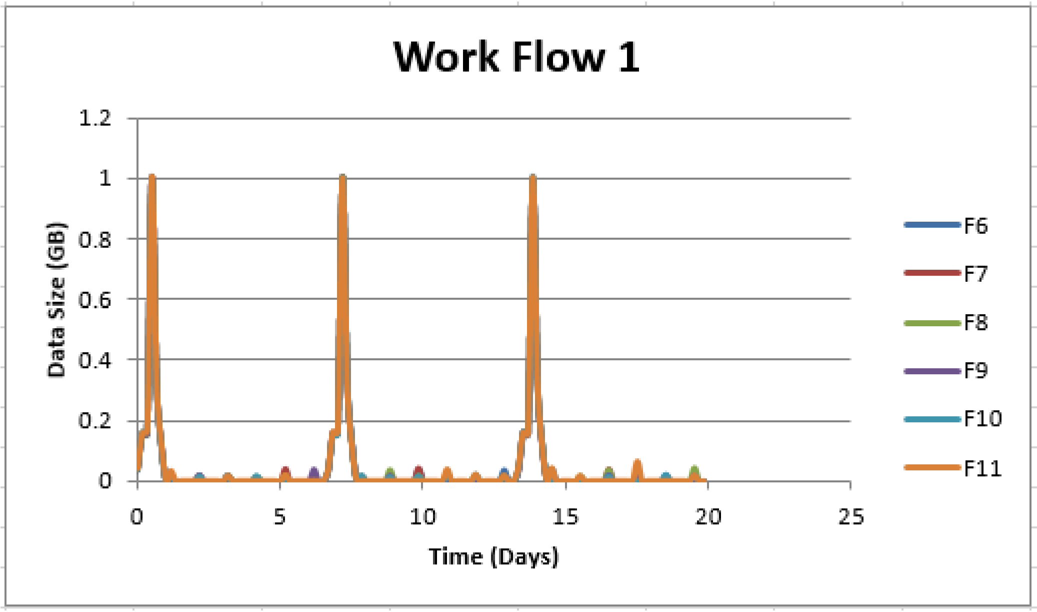


Figure 2: Plot of time versus backup data size for work-flow-1

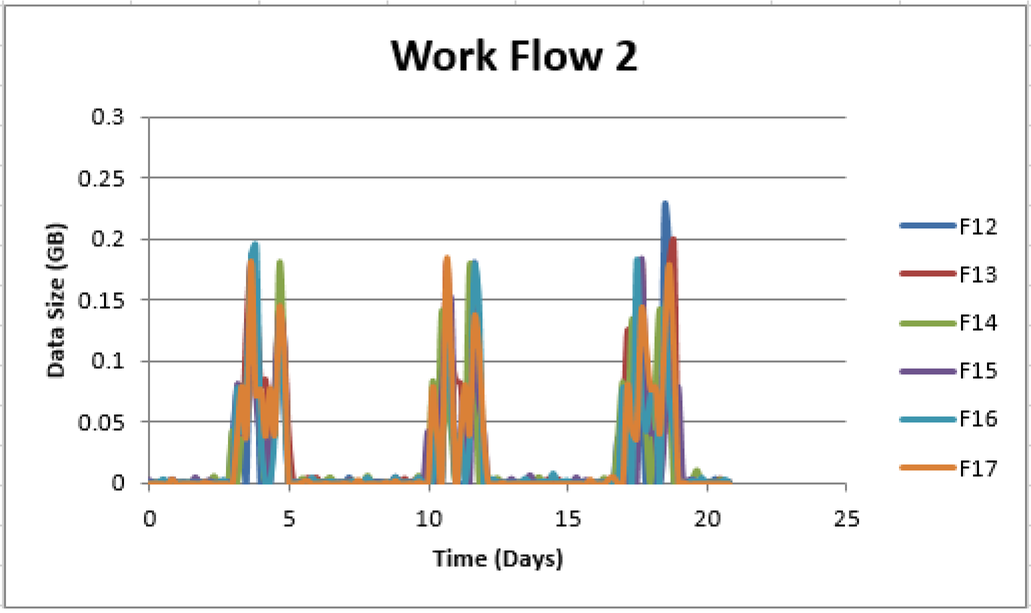


Figure 3: Plot of time versus backup data size for work-flow-2

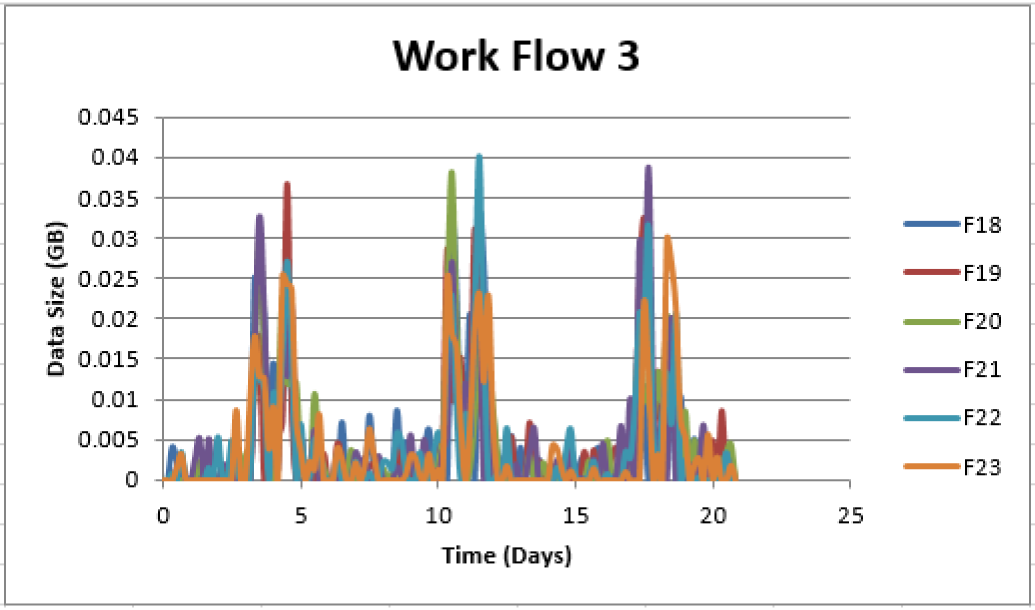


Figure 4: Plot of time versus backup data size for work-flow-3

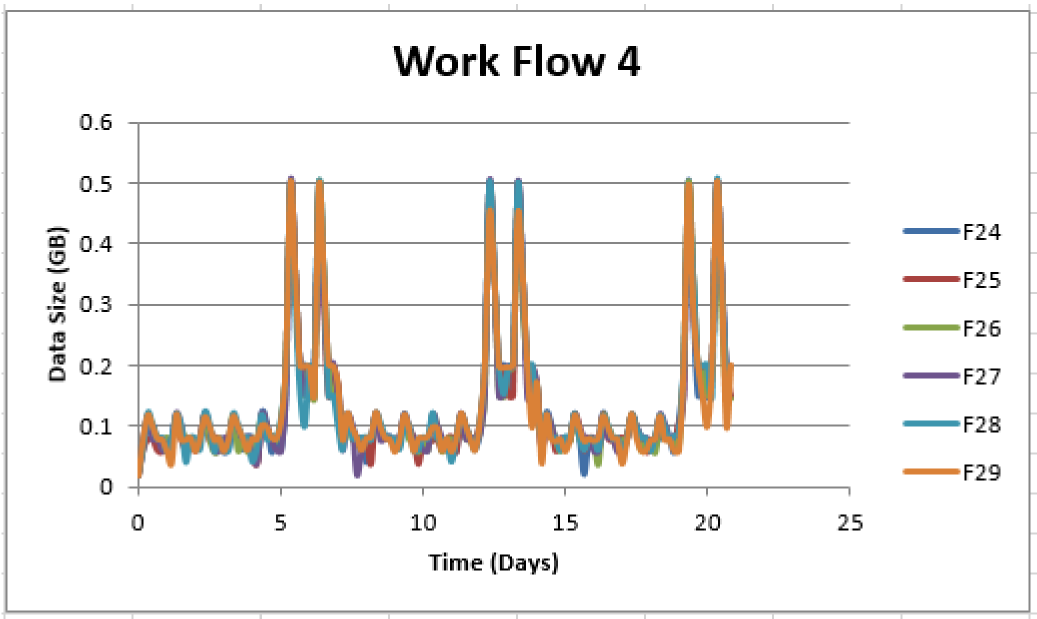


Figure 5: Plot of time versus backup data size for work-flow-4

Can you identify any repeating patterns?

Let us start with work-flow-0. The backup data size follows a sinusoidal shape, with peaks during the middle of the day, and valleys during the beginning and end of the day. The magnitude of the sinusoid decreases for about two days corresponding to the weekend (Saturday and Sunday). On the 2nd, 9th, 12th, and 16th days, there was a sharp spike in backup data size which corresponds to the Tuesday of each week (File 5 exhibited a spike on the 12th but no others did so we can assume an error of some sort or an abnormality or it happens every other week).

For work-flow-1 the pattern is obvious; the size of the backup data is extremely small except for Monday of each week.

For work-flow-2, the pattern resembles a mountain with 2 major peaks that occur sometime around the Thursday and Friday of each week. There is hardly any backup data during the other days.

For work-flow-3, the pattern resembles the pattern in work-flow-2 but with more relative backup data during the non-peak days (the days besides Thursday and Friday of each week). The size of the backup data is also much smaller compared to work-flow-2.

For work-flow-4, the pattern is nearly the opposite of work-flow-1; small sinusoidal behavior during the weekdays and larger magnitude peaks during the weekend.

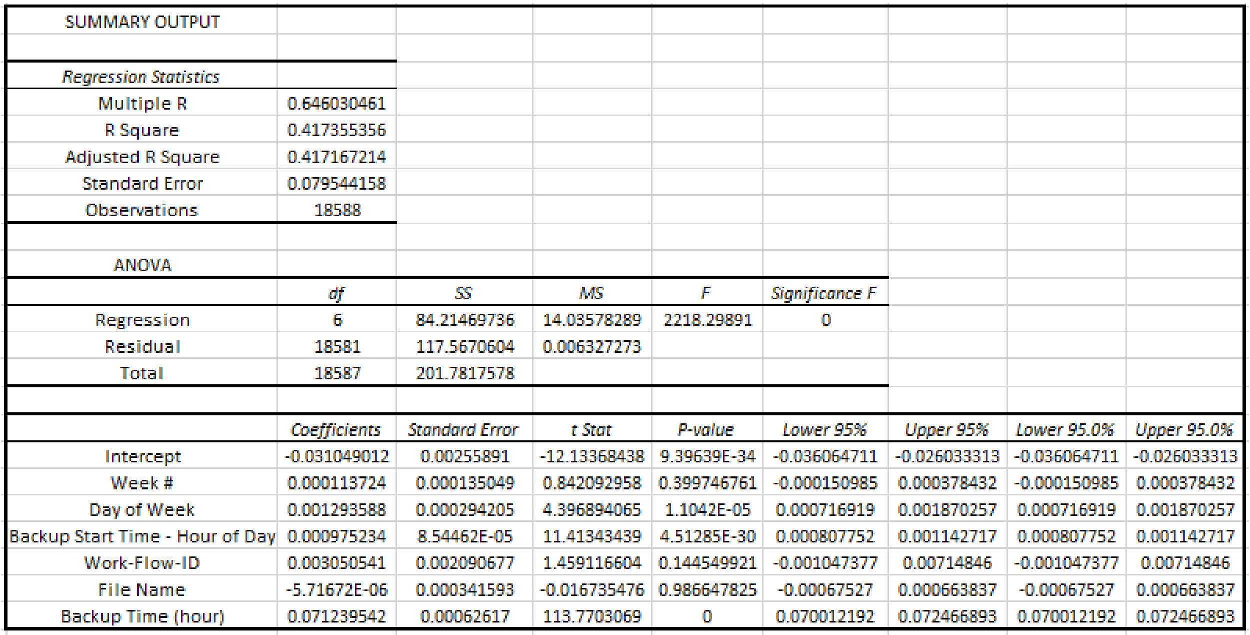
The procedure to generate the plots were done with MATLAB (sorting and filtering the data) and Excel (plotting the data). Those codes and files have been included in the zip if desired to look at.

2a) Fit a linear regression model with backup size as the target. Use least squares as the penalty. Use 10-Fold Cross-Validation.

This was implemented in MATLAB. Please refer to “traintest.m” for the code. The code is well documented and explained. A summary of the code and its functionality is provided below.

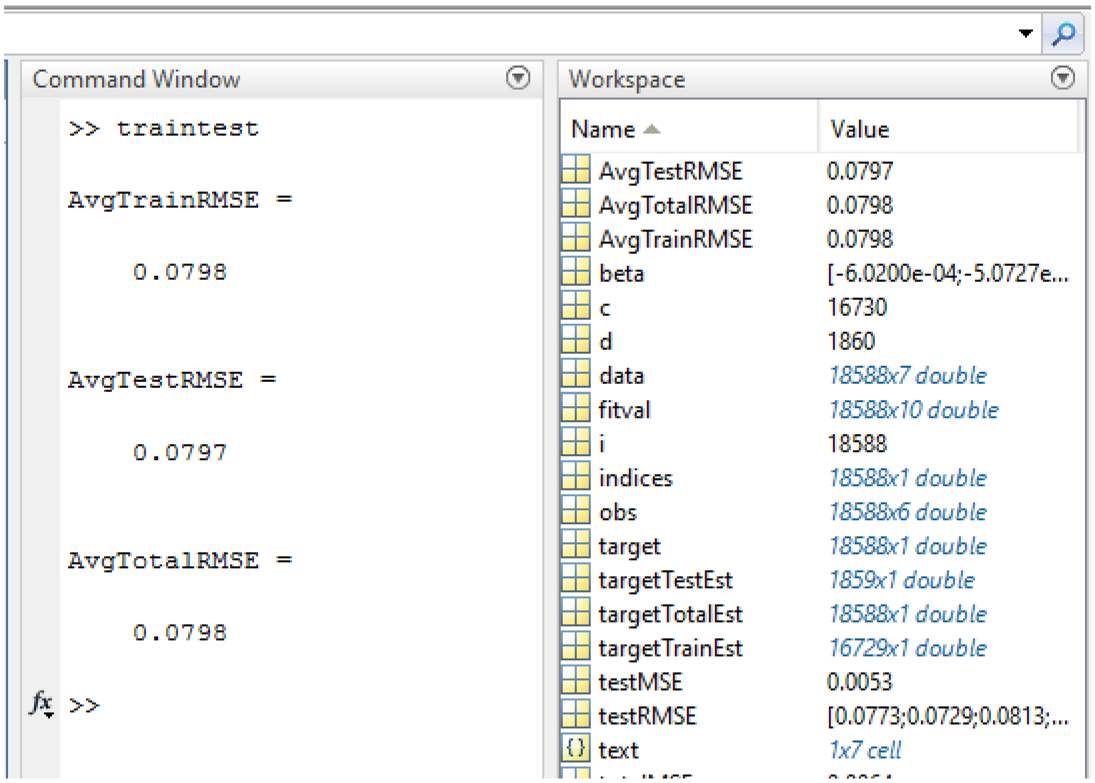
1. First, we needed to encode the text observations as a number. We decided to encode Monday – Sunday to the numbers 1 – 7 (Monday being 1, Tuesday being 2, etc.). Next, we encoded the workflow number to the number plus 1 (to avoid 0s) (work_flow_0 being 1, work_flow_1 being 2, etc.). The same was done to the file number (adding one).
2. Next, we read in the data using xlsread. After, we create an indices vector; this vector contains the numbers 1 through 10. The number indicates which fold that corresponding observation belongs to. This was done using crossvalind.
3. After separating the data into observations and targets, we train the model 10 times, each time using a different fold for testing. The output of the code displays the average training data RMSE, average testing RMSE, and average total RMSE.

Analyze the statistics of the different variables and report RMSE.



SUMMARY OUTPUT								
Regression Statistics								
Multiple R	0.646030461							
R Square	0.417355356							
Adjusted R Square	0.417167214							
Standard Error	0.079544158							
Observations	18588							
ANOVA								
	df	*SS*	*MS*	*F*	*Significance F*			
Regression	6	84.21469736	14.03578289	2218.29891	0			
Residual	18581	117.5670604	0.006327273					
Total	18587	201.7817578						
	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	-0.031049012	0.00255891	-12.13368438	9.39639E-34	-0.036064711	-0.026033313	-0.036064711	-0.026033313
Week #	0.000113724	0.000135049	0.842092958	0.399746761	-0.000150985	0.000378432	-0.000150985	0.000378432
Day of Week	0.001293588	0.000294205	4.396894065	1.1042E-05	0.000716919	0.001870257	0.000716919	0.001870257
Backup Start Time - Hour of Day	0.000975234	8.54462E-05	11.41343439	4.51285E-30	0.000807752	0.001142717	0.000807752	0.001142717
Work-Flow-ID	0.003050541	0.002090677	1.459116604	0.144549921	-0.001047377	0.00714846	-0.001047377	0.00714846
File Name	-5.71672E-06	0.000341593	-0.016735476	0.986647825	-0.00067527	0.000663837	-0.00067527	0.000663837
Backup Time (hour)	0.071239542	0.00062617	113.7703069	0	0.070012192	0.072466893	0.070012192	0.072466893

Figure 6: Statistical data derived from the model



```
>> traintest

AvgTrainRMSE =

    0.0798


AvgTestRMSE =

    0.0797


AvgTotalRMSE =

    0.0798

>>
```

Name	Value
AvgTestRMSE	0.0797
AvgTotalRMSE	0.0798
AvgTrainRMSE	0.0798
beta	[-6.0200e-04;-5.0727e...
c	16730
d	1860
data	*18588x7 double*
fitval	*18588x10 double*
i	18588
indices	*18588x1 double*
obs	*18588x6 double*
target	*18588x1 double*
targetTestEst	*1859x1 double*
targetTotalEst	*18588x1 double*
targetTrainEst	*16729x1 double*
testMSE	0.0053
testRMSE	[0.0773;0.0729;0.0813;...
text	*1x7 cell*

Figure 7: RMSE values for the training, testing, and total dataset

We use the standard of a p-value less than 0.05 results in rejecting the null hypothesis. From *Figure 6* we can see that the p-values less than 0.05 are the intercept, day of week, backup start time, and backup time. For the backup time, the p-value is zero because of the nature of the data; the time is 0 when the size is 0, but when the time is 1, the size has a large range (similarly with 2 and up). The intercept will always have a large p-value as it is the error correcting factor.

These results are intuitive; the day of week (as pointed out from the work-flow pattern analysis) is very important in determining the size of the backup data. Similarly, with backup start time, for something like work-flow-0, the middle of the day had higher numbers than the beginning and end (sinusoidal and peaking behaviors). We can look at the relative impact of the others by ascending p-value; the workflow id had a relatively small p-value at around 0.1 which is reasonable as shown with the patterns. The week number was at around 0.3 due to certain work-flows having weekly patterns, and file number close to 1 since all the files exhibited the same pattern under a certain workflow and thus didn't really influence the size of the backup data.

Evaluate your model using the plots "Fitted and Actual vs Time" and "Residuals vs Fitted".

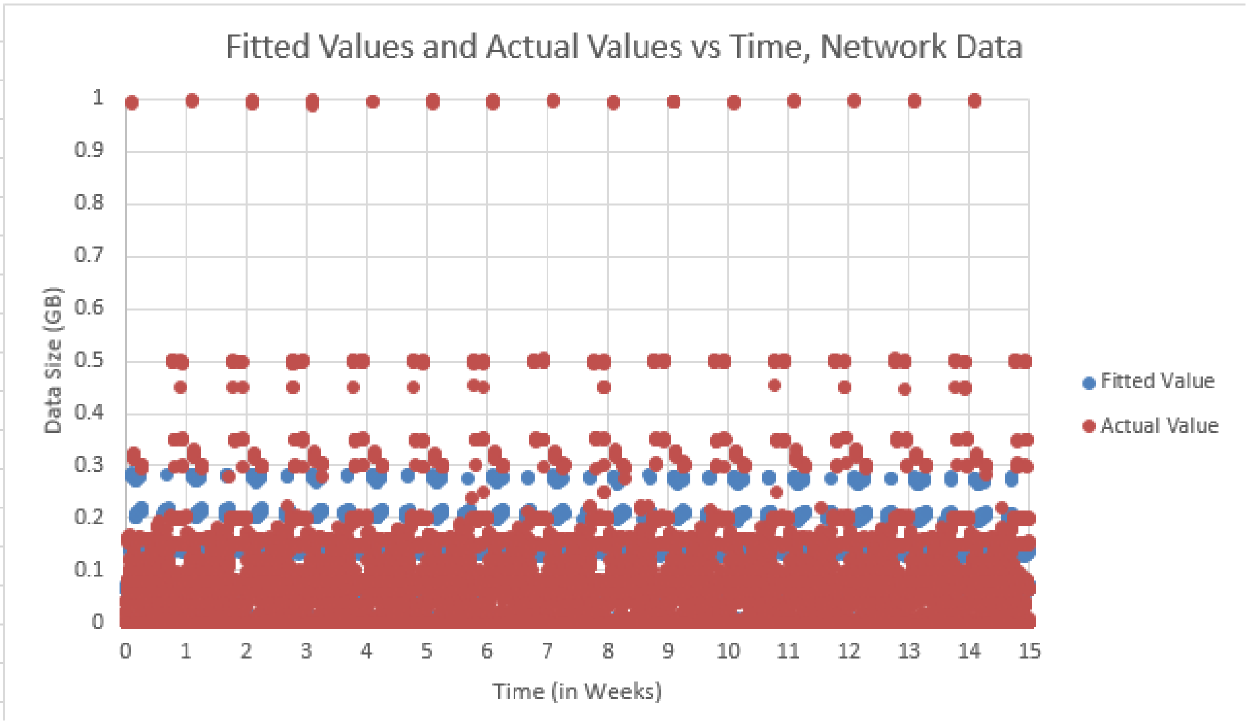


Figure 8: Fitted Values and Actual Values vs Time

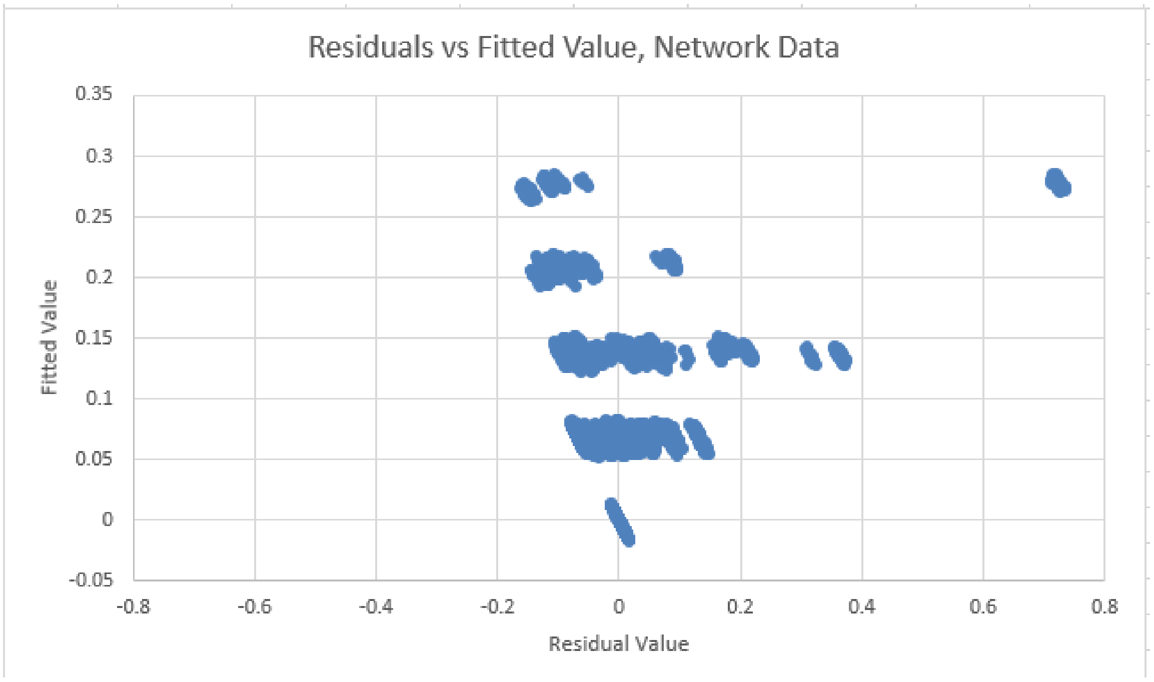


Figure 9: Residuals vs Fitted Value

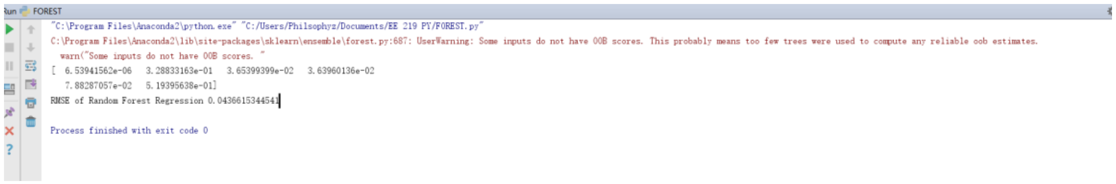
Let us look at *Figure 8*. We can see for small backup data, our model is doing well (the dots overlap so much that one cannot see the sea of blue dots under the sea of red dots). However, when the backup data gets larger, we cannot estimate those values well due to the linear nature of the data; most it is clustered around lower values (roughly around 0.25 GB). Thus, to minimize the error function, our model attempts to fit most the data instead of accommodating for the outliers.

Figure 9 shows us how centered our residuals are around zero. In an ideal world, we would have a straight vertical line at zero, telling us that no matter the value of the fitted data, we have a perfect estimate of the actual data. However, we can see a trend towards that zero line minus a group of outliers when the data size is big (around 1GB, as discussed before).

Thus, our model does a good job of fitting the majority in the data, and performs worse when the actual data is an outlier. This is expected due to the linear nature of the model.

2b) Use a random forest regression model. Use 20 trees and depth 4. Tune parameters and report the best RMSE you can get. Compare the performance with the LR model. Which features are more important? Can you identify patterns from part 1?

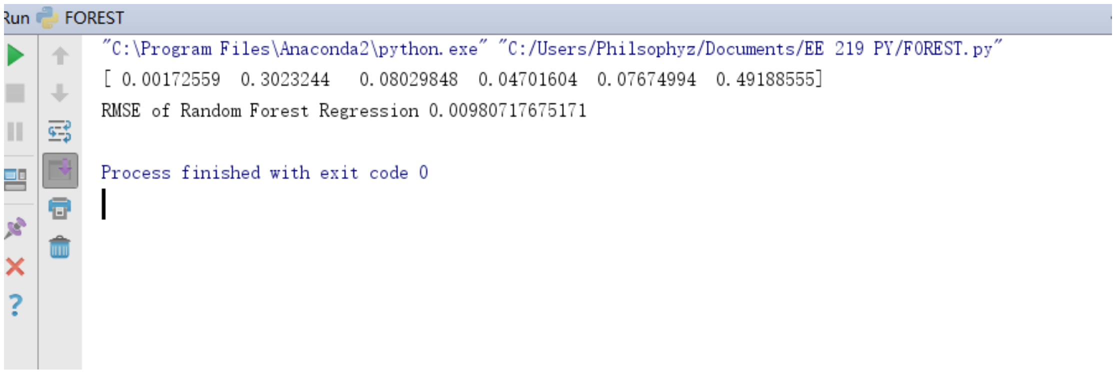
We started with the given initial values: 20 trees and depth of 4; with maximum features set to auto, the initial RMSE was 0.043. This was done in python (refer to “FOREST.py”).



```
"C:\Program Files\Anaconda2\python.exe" "C:/Users/Philsophyz/Documents/EE 219 PY/FOREST.py"
C:\Program Files\Anaconda2\lib\site-packages\sklearn\ensemble\forest.py:687: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
[  6.53941562e-06   3.28833163e-01   3.65399399e-02   3.63960136e-02
   7.88287057e-02   5.19395638e-01]
RMSE of Random Forest Regression 0.0436615344541

Process finished with exit code 0
```

Figure A: RMSE of 20/4

After tuning the number of trees and depth, the best RMSE we could get was 0.0098 with number of trees being 34 and a depth of 11.



```
"C:\Program Files\Anaconda2\python.exe" "C:/Users/Philsophyz/Documents/EE 219 PY/FOREST.py"
[ 0.00172559  0.3023244   0.08029848  0.04701604  0.07674994  0.49188555]
RMSE of Random Forest Regression 0.00980717675171

Process finished with exit code 0
```

Figure B: RMSE of 34/11

The importance of each variable is shown in the table below (the higher value, the more important that variable is).

Week#	**Day of week**	**Backup start time**	**Work flow id**	**File name**	**Backup time (hour)**
0.00172559	0.3023244	0.08029848	0.04701604	0.07674994	0.49188555

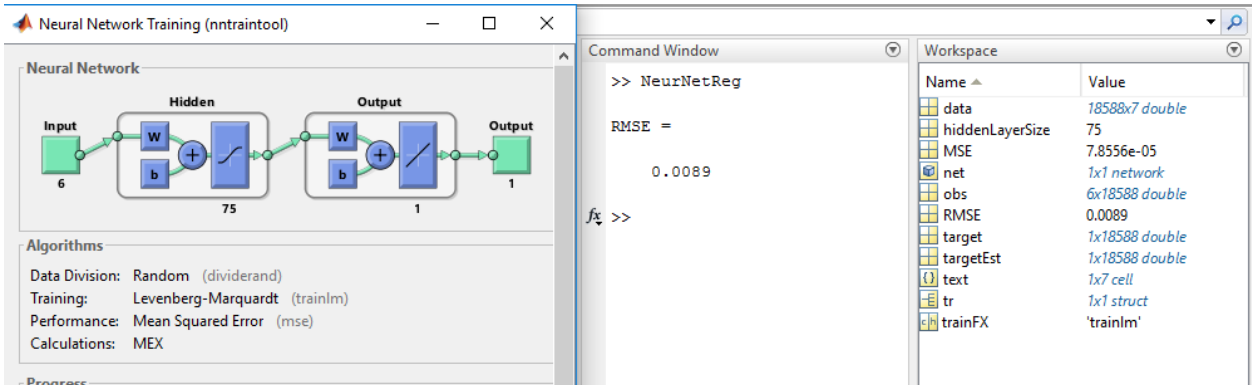
Figure C: Importance of variables

The performance of the random forest algorithm was much better than the LR model; RMSE of 0.07 (LM) vs 0.009 (Forest). The patterns from part one (statistical importance) were generally the same; backup time (hour) prevailed again as one of the most important factors, along with the day of the week.

2c) Use a neural network regression model. Explain the parameters of the network and their effect on RMSE.

This was implemented in MATLAB. Please refer to "NeurNetReg.m" for the code. The code is well documented and explained. A summary of the code and its functionality is provided below.

1. Data is read in using xlsread. Data is separated into observations and targets. Some transposition is done.
2. We specify the hidden layer size and training function to be used. The network is generated.
3. We set the train/test ratio to 90/10 as specified in the project.
4. We train our network using the data. Outputs are provided below.



Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

```
>> NeurNetReg

RMSE =

    0.0089

>>
```

Name	Value
data	18588x7 double
hiddenLayerSize	75
MSE	7.8556e-05
net	1x1 network
obs	6x18588 double
RMSE	0.0089
target	1x18588 double
targetEst	1x18588 double
text	1x7 cell
tr	1x1 struct
trainFX	'trainlm'

Figure 10: Neural Network training and testing in MATLAB

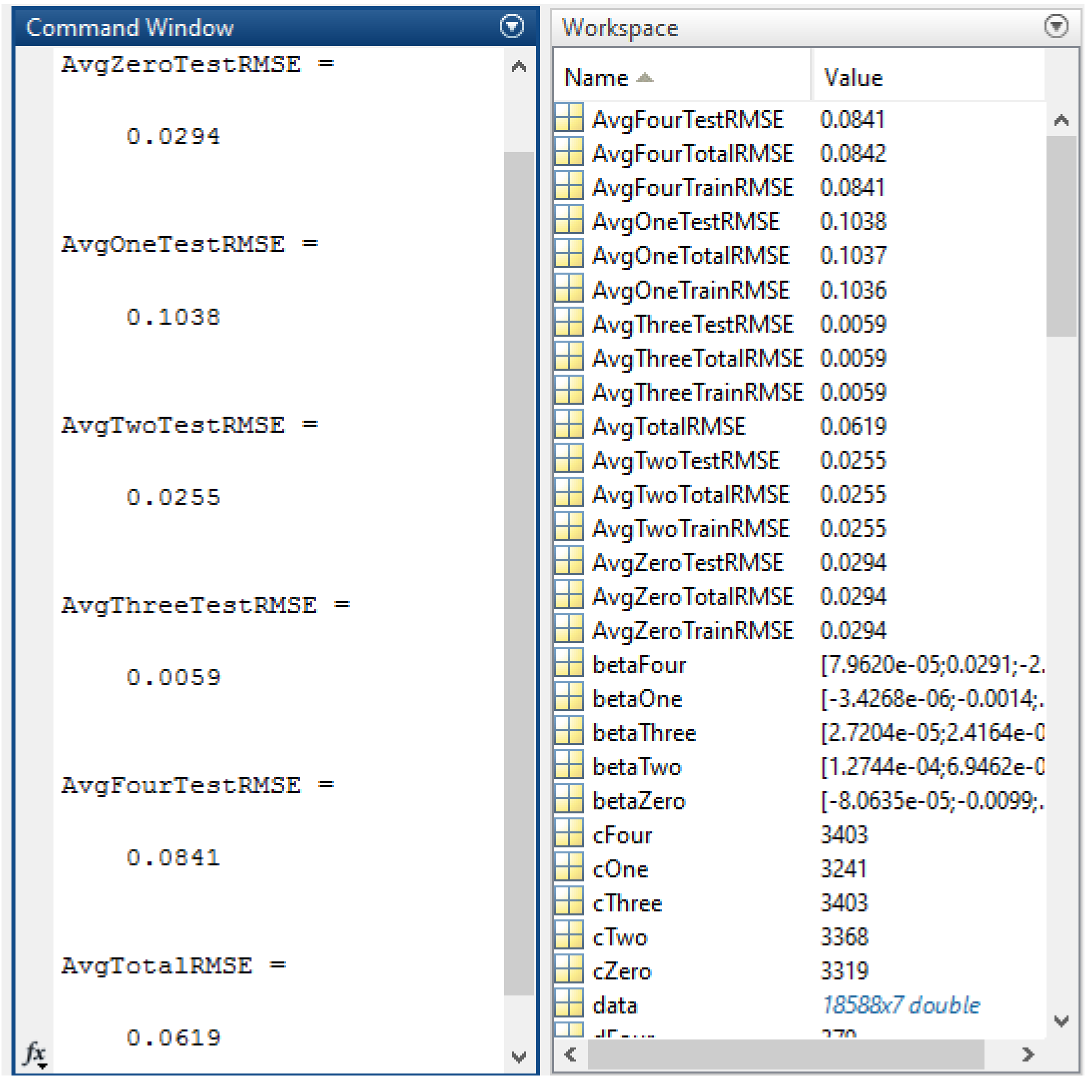
The key parameters of this implementation are the hidden layer size, training function, and the training/testing ratio. Hidden layer size is arguably the most important parameter of a neural network; the performance (and correspondingly reduction in RMSE) increases as the number of layers increase. The tradeoff is the computational and runtime cost; in our code, we chose a layer size of 75 due to minimal improvements in RMSE with an increased number of layers past that point. The next parameter is the training function; different literature on different training methods apply to different datasets. For our application, we chose LM since it is the default method for MSE. However, upon further analysis, a more suited method could be found that reduces our RMSE more than the LM method. Finally, the train/test ratio affects the generality of the model; fine tuning this to get the best RMSE differs from dataset to dataset. Since we were told to use 90/10 that is what this implementation uses.

NOTE: You will need the MATLAB neural network toolbox to run this code; you can obtain a free 30-day trial on their website.

3) Predict the backup size of the workflows separately. Explain if the fit is improved.

This was implemented in MATLAB. Please refer to “byWorkFlow.m” for the code. The code is well documented and explained. A summary of the code and its functionality is provided below.

1. This code functions the same as “traintest.m” but now applied to each workflow separately. Thus, we have 3 average RMSE (train, test, total) values for each workflow and an overall average RMSE (applying the piecewise function to the total dataset).
2. For brevity, only the average test RMSEs are printed (along with the overall average).



```
AvgZeroTestRMSE =

    0.0294


AvgOneTestRMSE =

    0.1038


AvgTwoTestRMSE =

    0.0255


AvgThreeTestRMSE =

    0.0059


AvgFourTestRMSE =

    0.0841


AvgTotalRMSE =

    0.0619
```

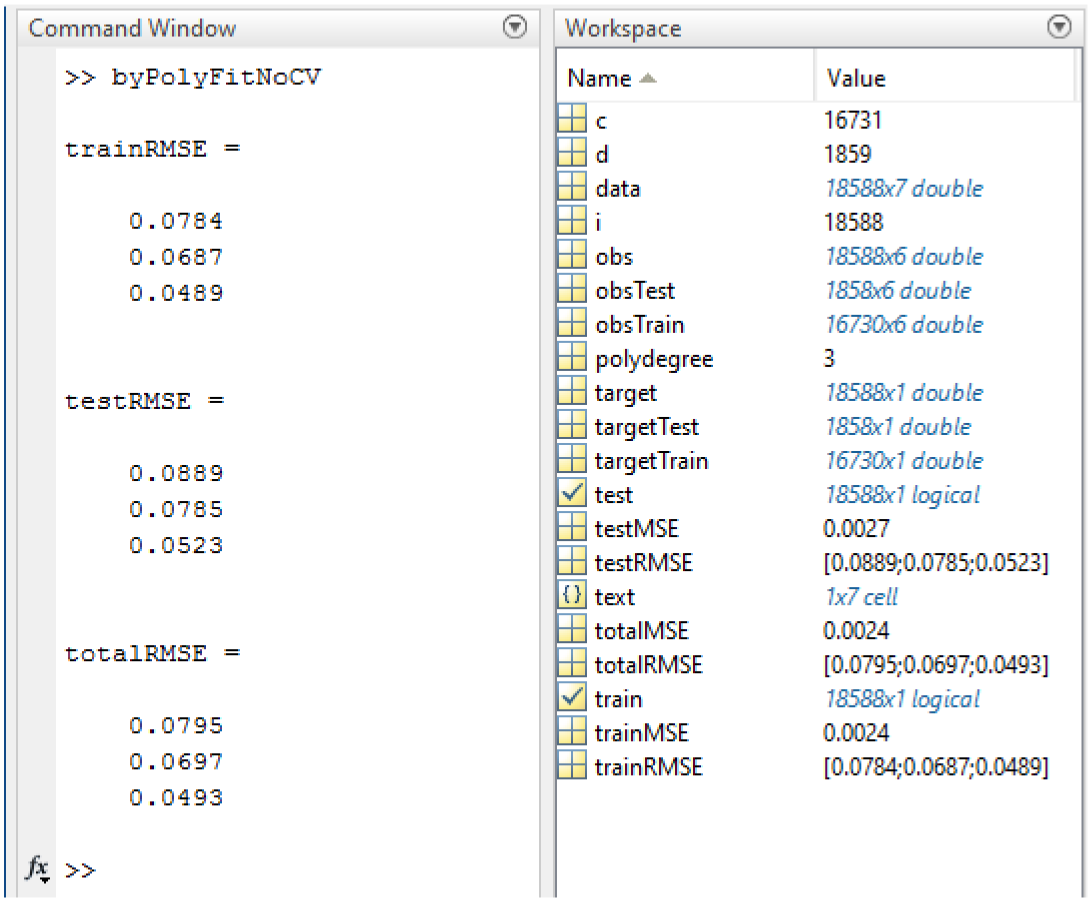
Name	Value
AvgFourTestRMSE	0.0841
AvgFourTotalRMSE	0.0842
AvgFourTrainRMSE	0.0841
AvgOneTestRMSE	0.1038
AvgOneTotalRMSE	0.1037
AvgOneTrainRMSE	0.1036
AvgThreeTestRMSE	0.0059
AvgThreeTotalRMSE	0.0059
AvgThreeTrainRMSE	0.0059
AvgTotalRMSE	0.0619
AvgTwoTestRMSE	0.0255
AvgTwoTotalRMSE	0.0255
AvgTwoTrainRMSE	0.0255
AvgZeroTestRMSE	0.0294
AvgZeroTotalRMSE	0.0294
AvgZeroTrainRMSE	0.0294
betaFour	[7.9620e-05;0.0291;-2.
betaOne	[-3.4268e-06;-0.0014;.
betaThree	[2.7204e-05;2.4164e-0
betaTwo	[1.2744e-04;6.9462e-0
betaZero	[-8.0635e-05;-0.0099;.
cFour	3403
cOne	3241
cThree	3403
cTwo	3368
cZero	3319
data	18588x7 double

Figure 11: Average RMSE values for each work flow, and the overall average RMSE of the data with the piecewise function as the model

With this piecewise model the overall fit is improved (lower average total RMSE). This is because we are now effectively modeling each workflow separately, and since each workflow has a distinct pattern, we can better predict the data given the workflow and applying the piecewise parameter that corresponds to that workflow. This is opposed to before, when we were trying to model the chaos of all the workflows combined with only one set of weights. Now, we can use a different set of weights depending on the workflow id and obtain a better prediction, thus leading to a lower RMSE.

Fit a polynomial function and do the following: 1) plot of degree vs RMSE for a fixed training and testing set, and 2) same plot but with average RMSE and using 10-Fold CV. How does CV help with controlling complexity?

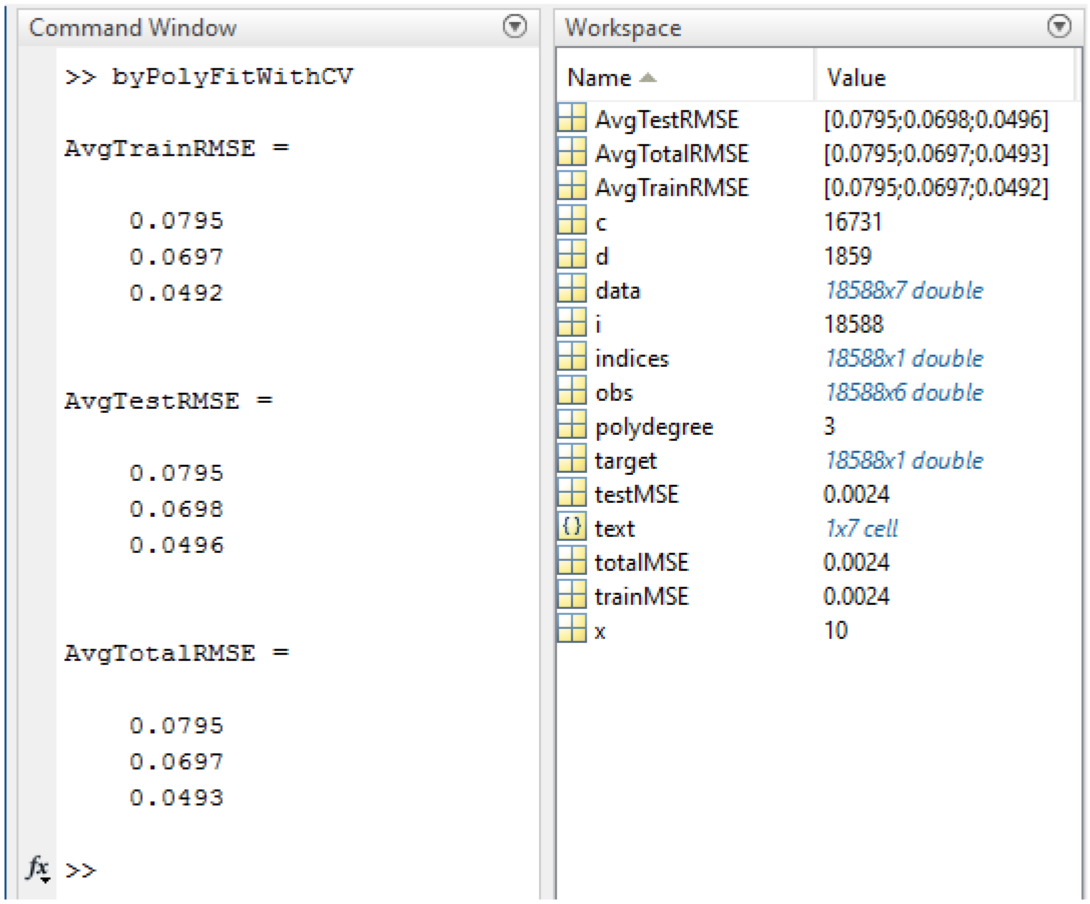
Refer to codes “byPolyFitNoCV.m” and “byPolyFitWithCV.m”. For testing purposes, we have set the range of degrees to be small. These codes employ “polyfitn”, a toolbox that is free to download online.



```
>> byPolyFitNoCV

trainRMSE =

    0.0784
    0.0687
    0.0489


testRMSE =

    0.0889
    0.0785
    0.0523


totalRMSE =

    0.0795
    0.0697
    0.0493

>>
```

Name	Value
c	16731
d	1859
data	18588x7 double
i	18588
obs	18588x6 double
obsTest	1858x6 double
obsTrain	16730x6 double
polydegree	3
target	18588x1 double
targetTest	1858x1 double
targetTrain	16730x1 double
test	18588x1 logical
testMSE	0.0027
testRMSE	[0.0889;0.0785;0.0523]
text	1x7 cell
totalMSE	0.0024
totalRMSE	[0.0795;0.0697;0.0493]
train	18588x1 logical
trainMSE	0.0024
trainRMSE	[0.0784;0.0687;0.0489]

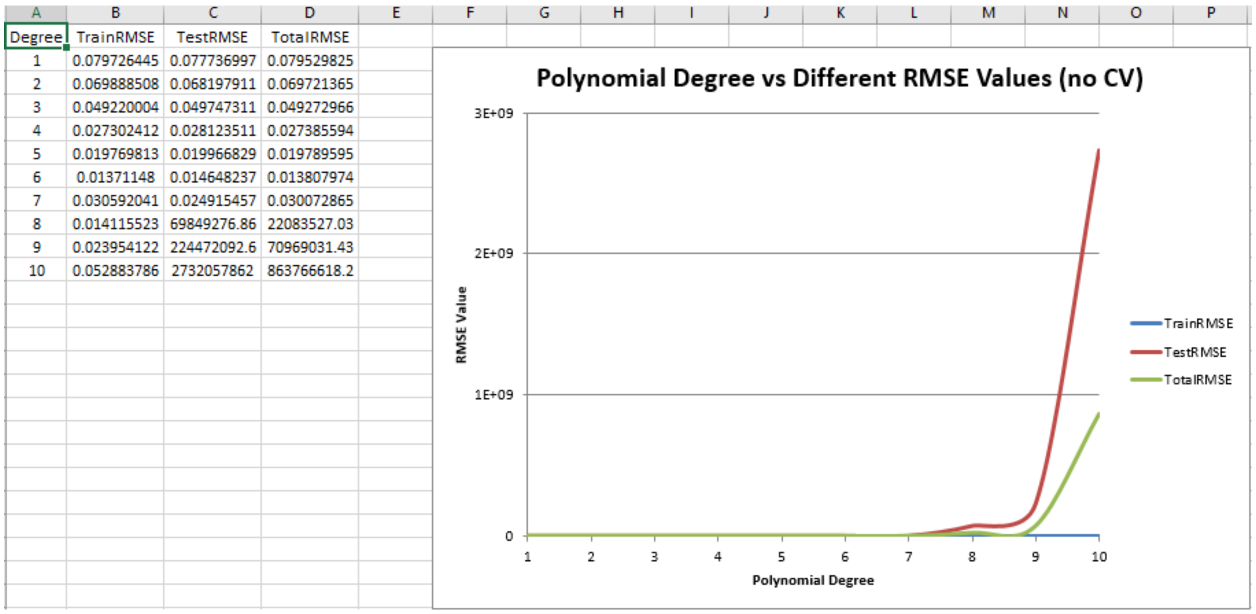
Figure 12: Fixed training and testing dataset, degrees from 1-3



```
>> byPolyFitWithCV

AvgTrainRMSE =

    0.0795
    0.0697
    0.0492


AvgTestRMSE =

    0.0795
    0.0698
    0.0496


AvgTotalRMSE =

    0.0795
    0.0697
    0.0493

>>
```

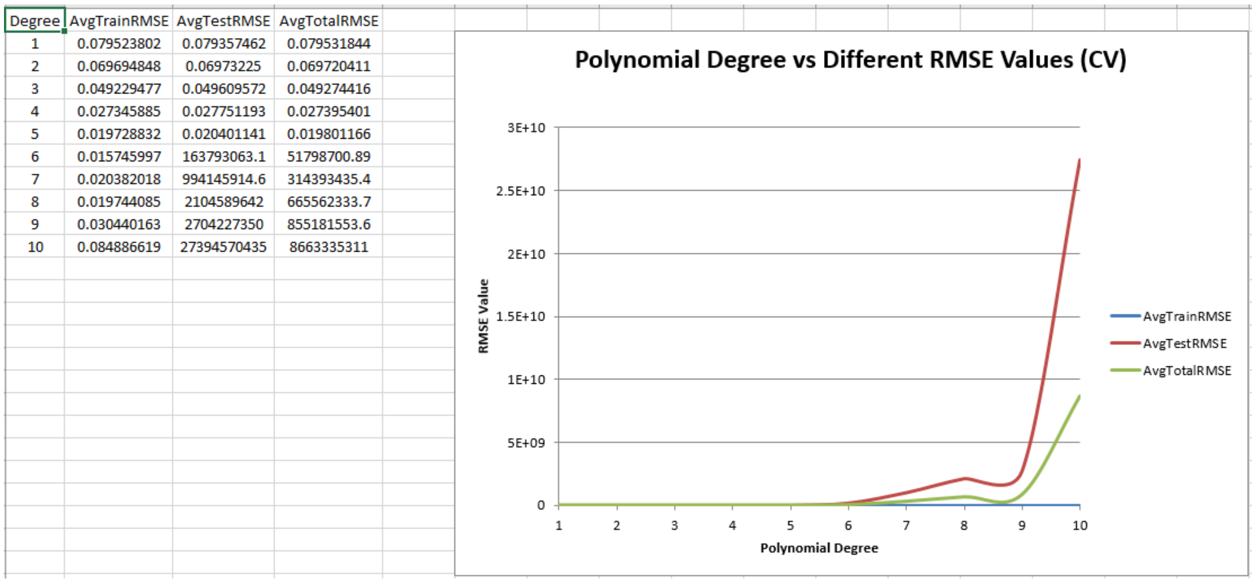
Name	Value
AvgTestRMSE	[0.0795;0.0698;0.0496]
AvgTotalRMSE	[0.0795;0.0697;0.0493]
AvgTrainRMSE	[0.0795;0.0697;0.0492]
c	16731
d	1859
data	18588x7 double
i	18588
indices	18588x1 double
obs	18588x6 double
polydegree	3
target	18588x1 double
testMSE	0.0024
text	1x7 cell
totalMSE	0.0024
trainMSE	0.0024
x	10

Figure 13: Using 10-Fold CV, degrees from 1-3



Degree	TrainRMSE	TestRMSE	TotalRMSE
1	0.079726445	0.077736997	0.079529825
2	0.069888508	0.068197911	0.069721365
3	0.049220004	0.049747311	0.049272966
4	0.027302412	0.028123511	0.027385594
5	0.019769813	0.019966829	0.019789595
6	0.01371148	0.014648237	0.013807974
7	0.030592041	0.024915457	0.030072865
8	0.014115523	69849276.86	22083527.03
9	0.023954122	224472092.6	70969031.43
10	0.052883786	2732057862	863766618.2

Figure 14: Degree vs RMSE, no CV, degrees 1-10



Degree	AvgTrainRMSE	AvgTestRMSE	AvgTotalRMSE
1	0.079523802	0.079357462	0.079531844
2	0.069694848	0.06973225	0.069720411
3	0.049229477	0.049609572	0.049274416
4	0.027345885	0.027751193	0.027395401
5	0.019728832	0.020401141	0.019801166
6	0.015745997	163793063.1	51798700.89
7	0.020382018	994145914.6	314393435.4
8	0.019744085	2104589642	665562333.7
9	0.030440163	2704227350	855181553.6
10	0.084886619	27394570435	8663335311

Figure 15: Degree vs average RMSE, with CV, degrees 1-10

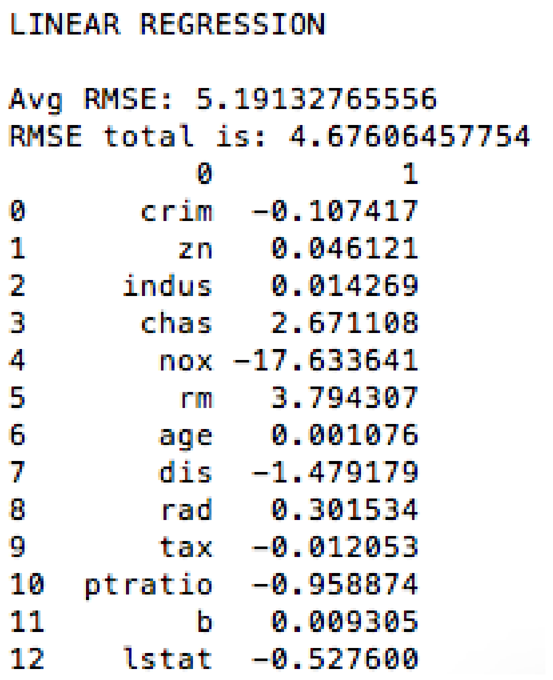
Looking at *Figure 14* we see the test and total average RMSEs spike with a degree of 8, with no CV. However, in *Figure 15*, we can see the spike occur at a degree of 6, with CV. Cross validation controls the complexity of the model by ensuring we do not overfit the data by checking it 10 times with 10 different testing sets for each polynomial degree. Had we not used CV, we would have assumed an order of 6 (where the minimum RMSE occurs) but CV tells us that the RMSE spikes at a degree of 6. Thus, we can regulate the complexity of our model and would select a degree of 5.

4) Fit a LR model with MEDV as the target, perform 10-Fold CV, and analyze the significance of different variables with the statistics.

NOTE: In the main function of BosHousing.py , only uncomment and run one line at a time. Instructions for each are below and also commented in .py file.

We implemented Linear Regression in python and used 10-fold cross validation in order to validate our linear fit.

Running the function call LinearRegs(‘housing_data.csv’) in the main function gave us the following output:



```
LINEAR REGRESSION

Avg RMSE: 5.19132765556
RMSE total is: 4.67606457754
           0          1
0       crim  -0.107417
1         zn   0.046121
2      indus   0.014269
3       chas   2.671108
4        nox -17.633641
5         rm   3.794307
6        age   0.001076
7        dis  -1.479179
8        rad   0.301534
9        tax  -0.012053
10   ptratio  -0.958874
11         b   0.009305
12     lstat  -0.527600
```

Figure 16: Linear Regression

We used Excel to generate useful statistics for the data as shown in *Figure 17*. We can see here that the variables that had p-values that were too high, such as INDUS (proportion of non-retail business acres per town) and AGE (proportion of owner-occupied units built prior to 1940) have little significance in our data. The other variables that did have much more significant in our data allow us to make interesting inferences about our data.

CRIM: The per capita crime rate by town had a great impact on housing prices as it makes logical sense that houses in more dangerous areas would be less desired.

CHAS: Distance from the Charles River also impacts housing prices as one can argue that being close to the river or having river views can increase the property value.

RAD: Having access to industrial highways affects how accessible different parts of the city are to people located in certain areas.

TAX: We can easily link property tax rates per $10,000 to a homes value.

PTRATIO: We can infer that a low pupil-teacher ratio is more desirable and thus will probably be lower around homes with higher values.

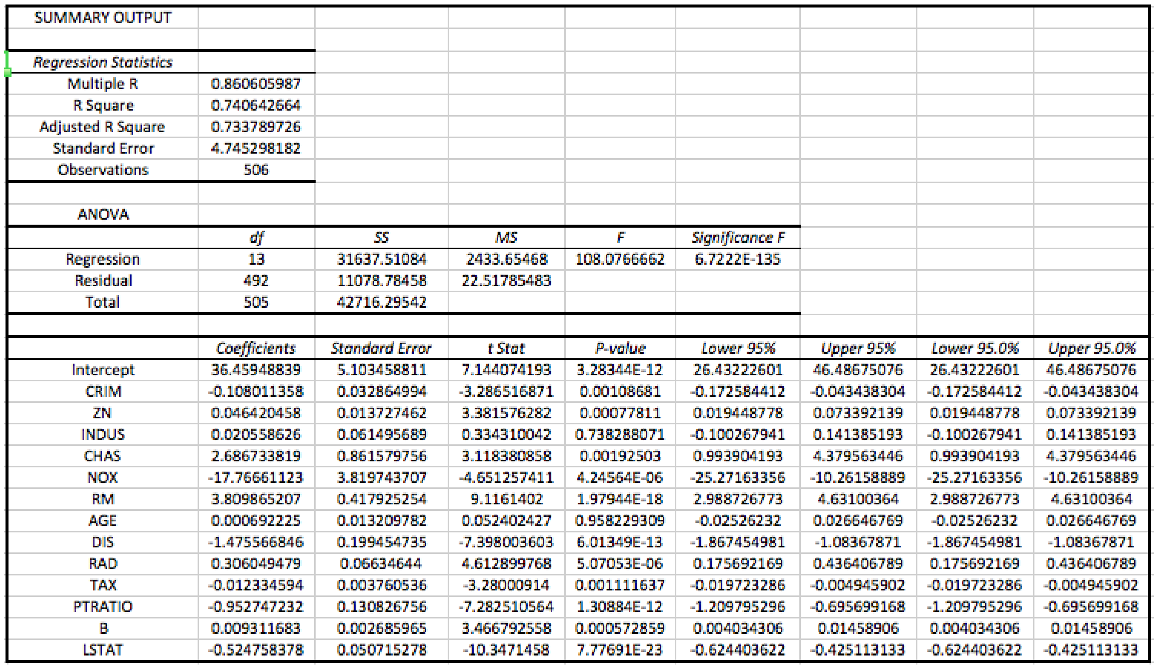
LSTAT: The higher percentage of a lower status population will likely give rise to homes with lower values by definition.

RM: The average number of rooms per dwelling likely increases with higher-valued homes.

B: The proportion of blacks seems to have to an impact on housing prices.

ZN: The proportion of residential land zoned for lots over 25,000 sq. ft. are most likely reserved for homes that are much more massive and higher-valued than the average Boston home.

NOX: NOx emissions are known to be deleterious to our health and can also be seen to make certain homes less desirable and hence impact their values.



SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.860605987
R Square	0.740642664
Adjusted R Square	0.733789726
Standard Error	4.745298182
Observations	506

ANOVA

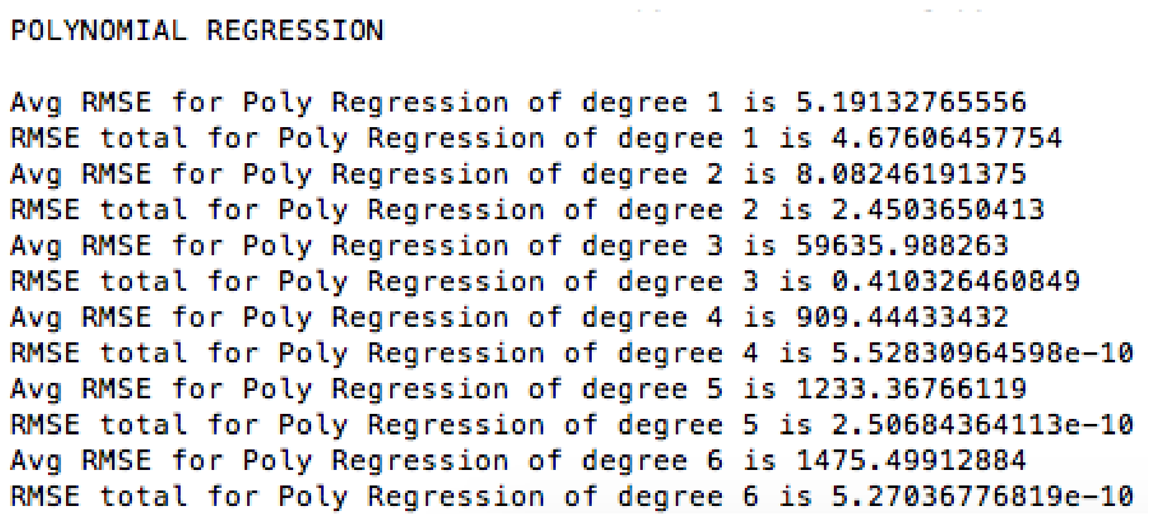
	df	*SS*	*MS*	*F*	*Significance F*
Regression	13	31637.51084	2433.65468	108.0766662	6.7222E-135
Residual	492	11078.78458	22.51785483		
Total	505	42716.29542			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	36.45948839	5.103458811	7.144074193	3.28344E-12	26.43222601	46.48675076	26.43222601	46.48675076
CRIM	-0.108011358	0.032864994	-3.286516871	0.00108681	-0.172584412	-0.043438304	-0.172584412	-0.043438304
ZN	0.046420458	0.013727462	3.381576282	0.00077811	0.019448778	0.073392139	0.019448778	0.073392139
INDUS	0.020558626	0.061495689	0.334310042	0.738288071	-0.100267941	0.141385193	-0.100267941	0.141385193
CHAS	2.686733819	0.861579756	3.118380858	0.00192503	0.993904193	4.379563446	0.993904193	4.379563446
NOX	-17.76661123	3.819743707	-4.651257411	4.24564E-06	-25.27163356	-10.26158889	-25.27163356	-10.26158889
RM	3.809865207	0.417925254	9.1161402	1.97944E-18	2.988726773	4.63100364	2.988726773	4.63100364
AGE	0.000692225	0.013209782	0.052402427	0.958229309	-0.02526232	0.026646769	-0.02526232	0.026646769
DIS	-1.475566846	0.199454735	-7.398003603	6.01349E-13	-1.867454981	-1.08367871	-1.867454981	-1.08367871
RAD	0.306049479	0.06634644	4.612899768	5.07053E-06	0.175692169	0.436406789	0.175692169	0.436406789
TAX	-0.012334594	0.003760536	-3.28000914	0.001111637	-0.019723286	-0.004945902	-0.019723286	-0.004945902
PTRATIO	-0.952747232	0.130826756	-7.282510564	1.30884E-12	-1.209795296	-0.695699168	-1.209795296	-0.695699168
B	0.009311683	0.002685965	3.466792558	0.000572859	0.004034306	0.01458906	0.004034306	0.01458906
LSTAT	-0.524758378	0.050715278	-10.3471458	7.77691E-23	-0.624403622	-0.425113133	-0.624403622	-0.425113133

Figure 17: Statistics for Coefficients

We found the total RMSE = 4.676.

We also implemented Polynomial Regression using python and found some interesting cases. We found that as we increase the degree of the polynomial, we found the total RMSE to decrease as expected. However upon using the 10-fold cross validation we saw that overfitting of the data had occurred. *Figure 18* compares the average RMSE values of the cross validation with the total RMSE values of the dataset.



```
POLYNOMIAL REGRESSION

Avg RMSE for Poly Regression of degree 1 is 5.19132765556
RMSE total for Poly Regression of degree 1 is 4.67606457754
Avg RMSE for Poly Regression of degree 2 is 8.08246191375
RMSE total for Poly Regression of degree 2 is 2.4503650413
Avg RMSE for Poly Regression of degree 3 is 59635.988263
RMSE total for Poly Regression of degree 3 is 0.410326460849
Avg RMSE for Poly Regression of degree 4 is 909.44433432
RMSE total for Poly Regression of degree 4 is 5.52830964598e-10
Avg RMSE for Poly Regression of degree 5 is 1233.36766119
RMSE total for Poly Regression of degree 5 is 2.50684364113e-10
Avg RMSE for Poly Regression of degree 6 is 1475.49912884
RMSE total for Poly Regression of degree 6 is 5.27036776819e-10
```

Figure 18: Polynomial Regression

Here we can see that even though the total RMSE dercreased from degree one to two, the average RMSE for the 10-fold cross validation had increased. This trend continued on from degree 2 to degree 3 as well and we can see the spike in the average RMSE. We see that a polynomial regression of degree 1 is in fact the optimal case as it has the least average RMSE. And since polynomial regression of degree 1 is equivalent to the linear case, we achieved the same average and total RMSE's.

Plot the same curves from part 2 (Fitted and Actual vs Samples and Residuals vs Fitted).

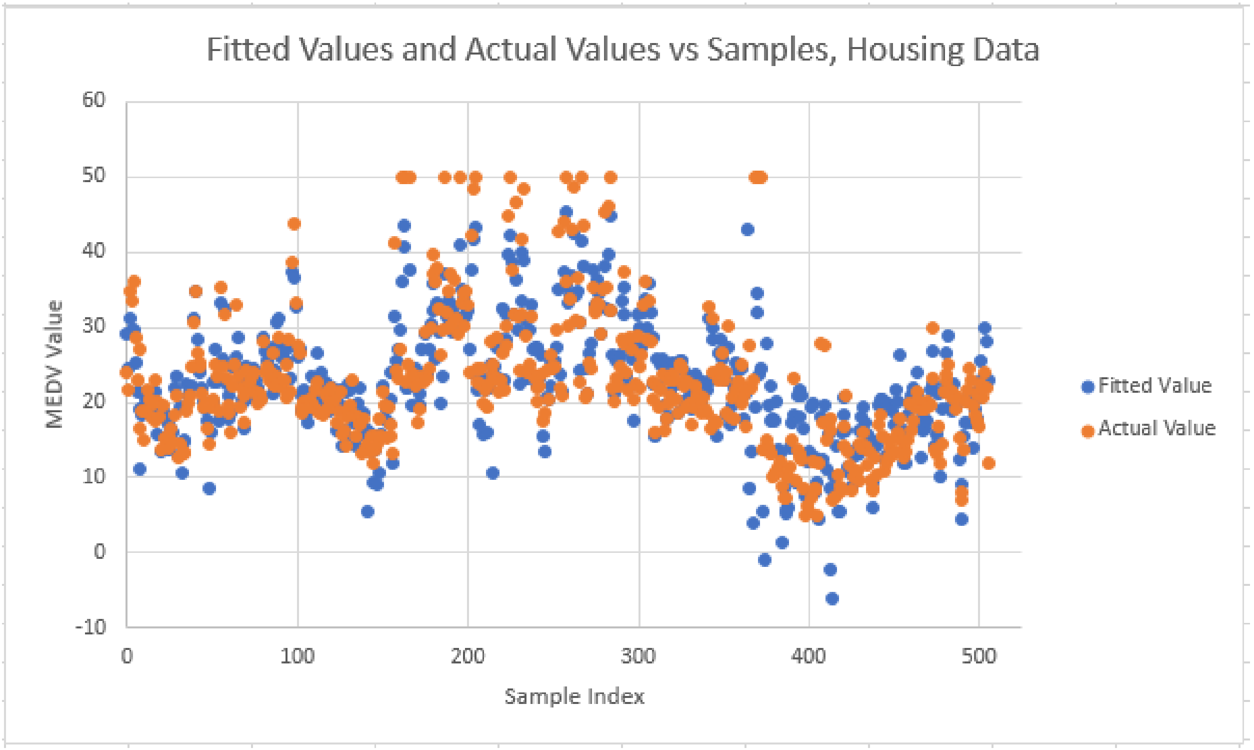


Figure 19: Fitted Values and Actual Values vs Samples

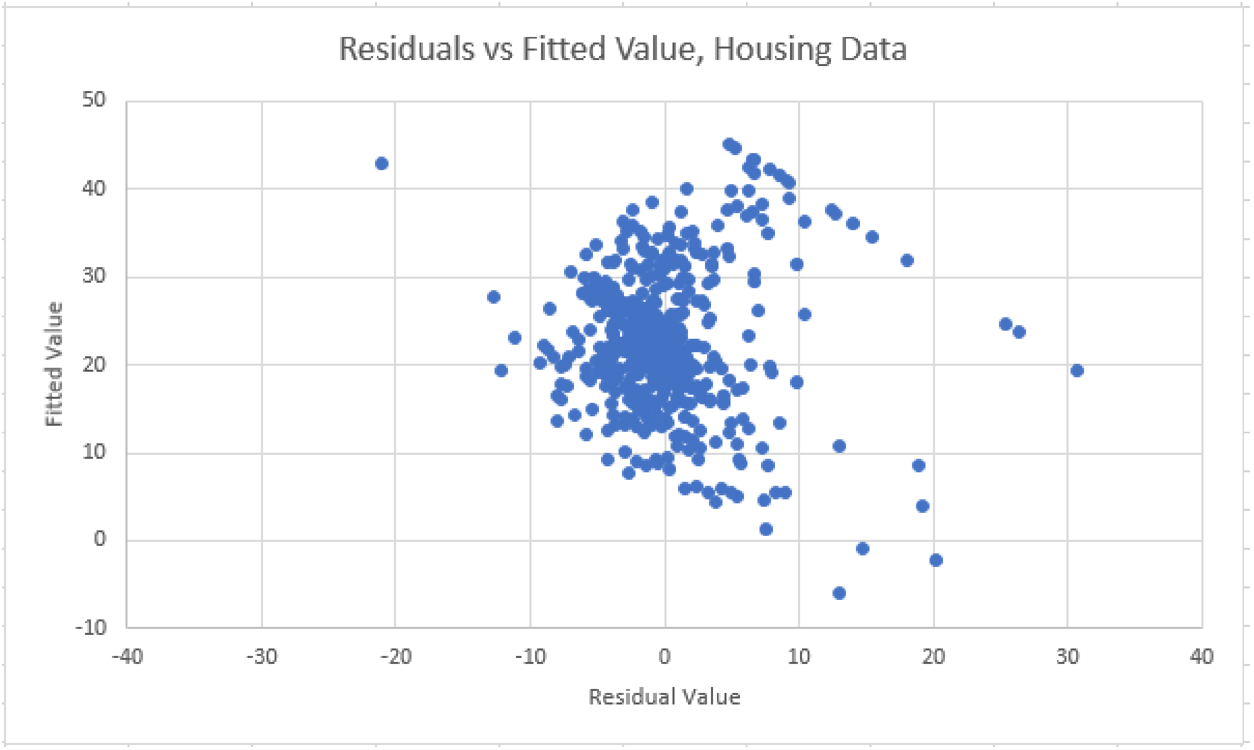


Figure 20: Residuals vs Fitted Value

As before, we can see from *Figure 19* that our model does a fair job of fitting the data, and from *Figure 20* we can see that our residuals are centered around the zero line and thus generally does a good job of fitting the data.

Repeat the steps for fitting a polynomial and plot the same graphs.

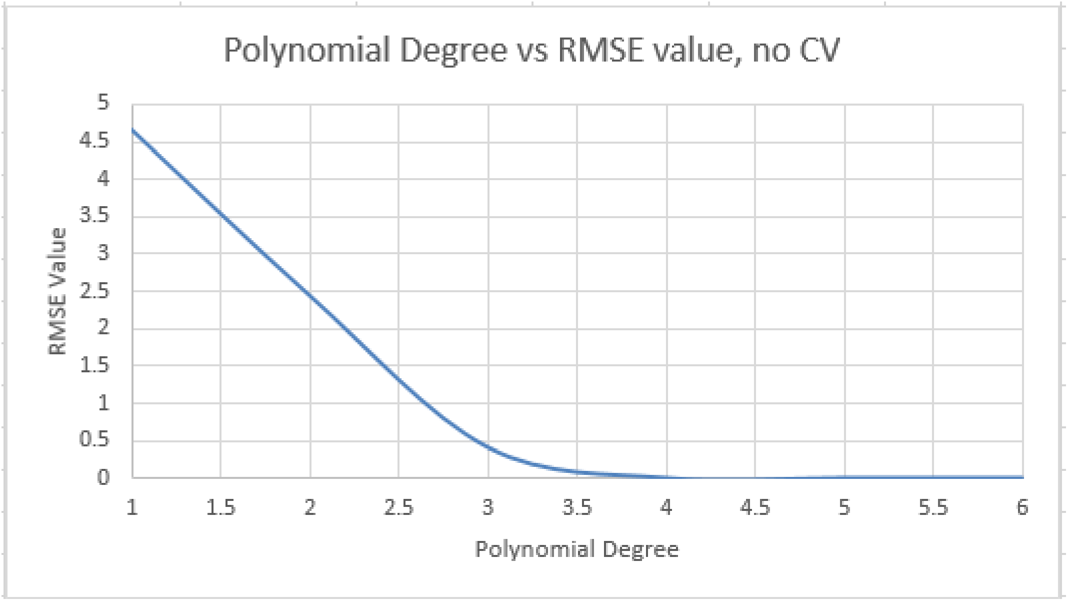


Figure 21: Polynomial Degree vs RMSE values, no CV

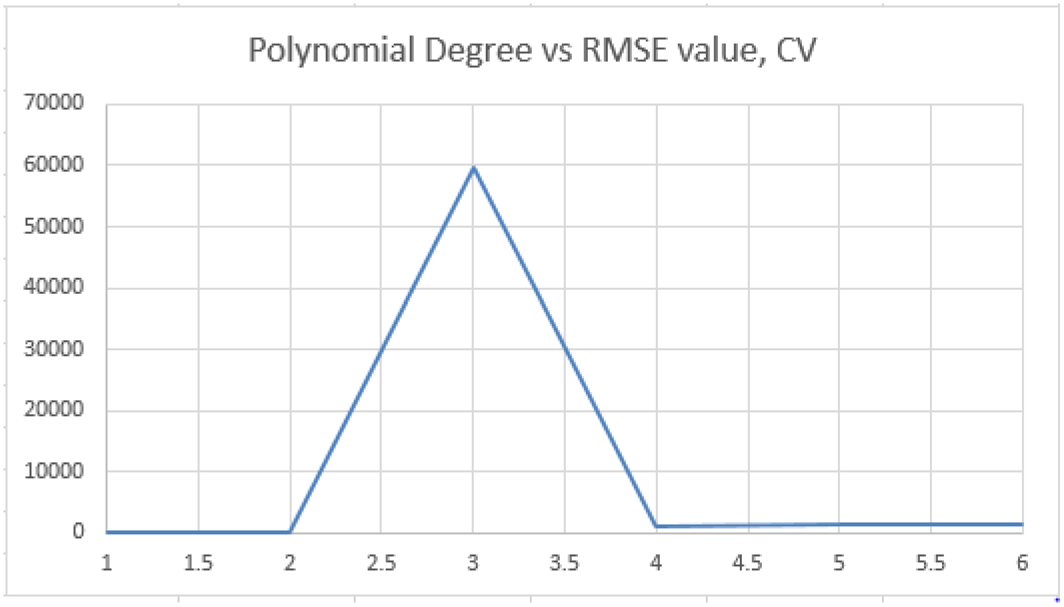


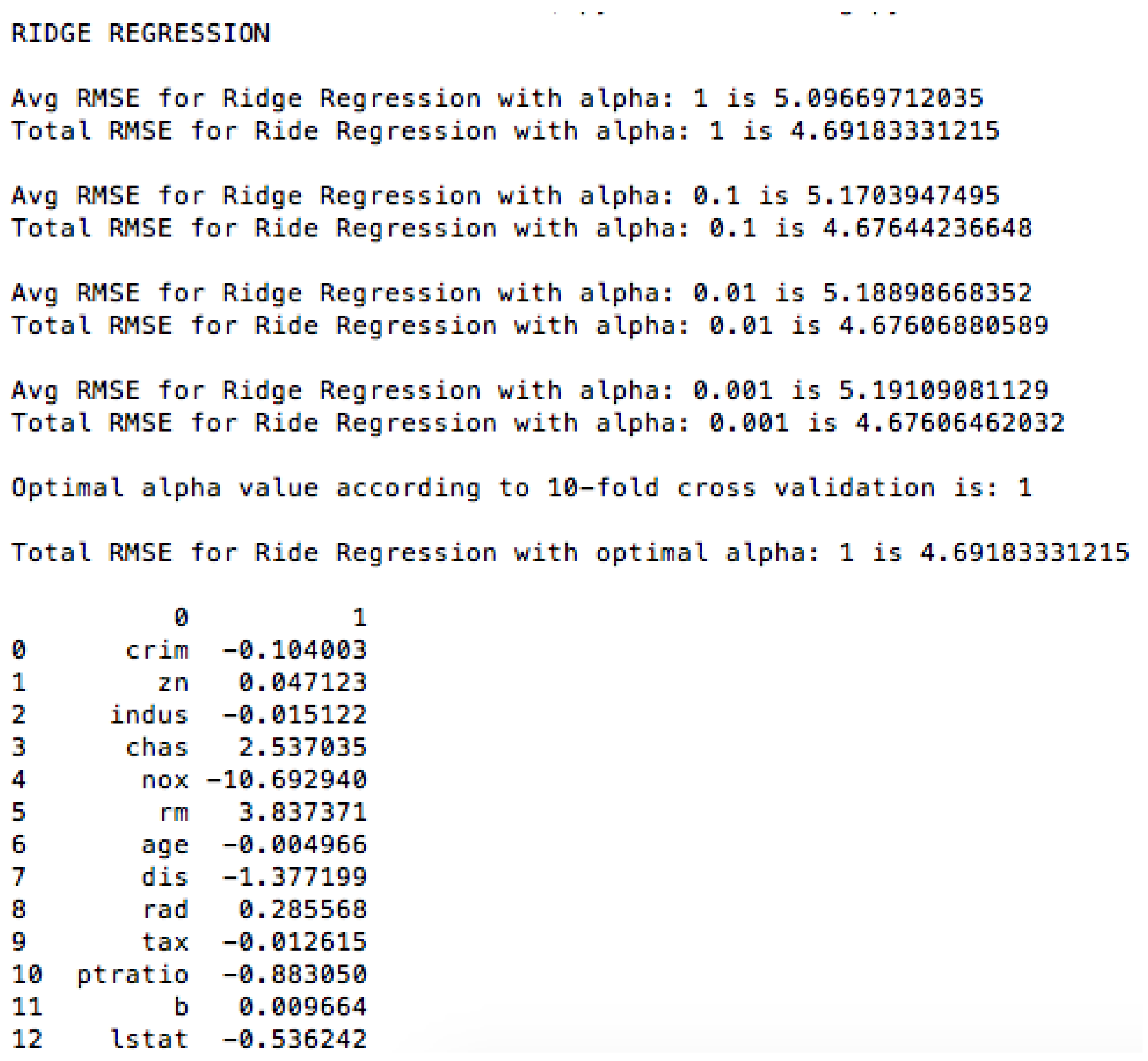
Figure 22: Polynomial Degree vs average RMSE values, CV

Again, we can see the effects of CV, we would select a degree of 2.

5a) Tune alpha and report best RMSE via 10-Fold CV. Compare values; use ridge.

We implemented Ridge Regression in python and used 10-fold cross validation to find the optimal alpha value per the average RMSE value calculated during the cross validation.

Running the function call **RidgeRegs(‘housing_data.csv’)** in the main function gave us the following output:



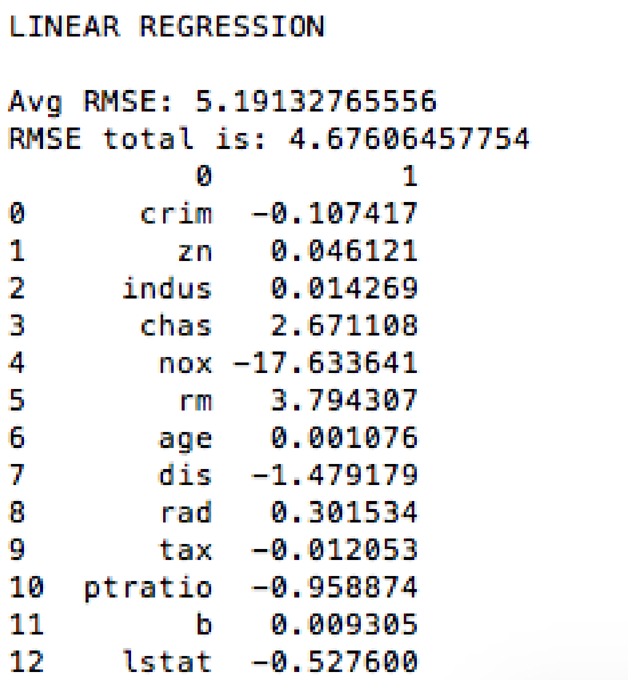
```
RIDGE REGRESSION

Avg RMSE for Ridge Regression with alpha: 1 is 5.09669712035
Total RMSE for Ride Regression with alpha: 1 is 4.69183331215

Avg RMSE for Ridge Regression with alpha: 0.1 is 5.1703947495
Total RMSE for Ride Regression with alpha: 0.1 is 4.67644236648

Avg RMSE for Ridge Regression with alpha: 0.01 is 5.18898668352
Total RMSE for Ride Regression with alpha: 0.01 is 4.67606880589

Avg RMSE for Ridge Regression with alpha: 0.001 is 5.19109081129
Total RMSE for Ride Regression with alpha: 0.001 is 4.67606462032

Optimal alpha value according to 10-fold cross validation is: 1

Total RMSE for Ride Regression with optimal alpha: 1 is 4.69183331215

          0          1
0      crim  -0.104003
1        zn   0.047123
2     indus  -0.015122
3      chas   2.537035
4       nox -10.692940
5        rm   3.837371
6       age  -0.004966
7       dis  -1.377199
8       rad   0.285568
9       tax  -0.012615
10  ptratio  -0.883050
11        b   0.009664
12    lstat  -0.536242
```

Figure 23: Ridge Regression

Here we can see that as we decreased the alpha value, the Avg RMSE value for the 10-fold cross validation decreased thus giving us an optimal alpha value of **$\alpha = 1$.**

Using this alpha, we then generated the above coefficients.

An un-regularized model would represent an l_2 or l_1 regularization with an alpha value of **$\alpha = 0$**, or simply just linear regression. The output below shows the coefficients from said un-regularized model.



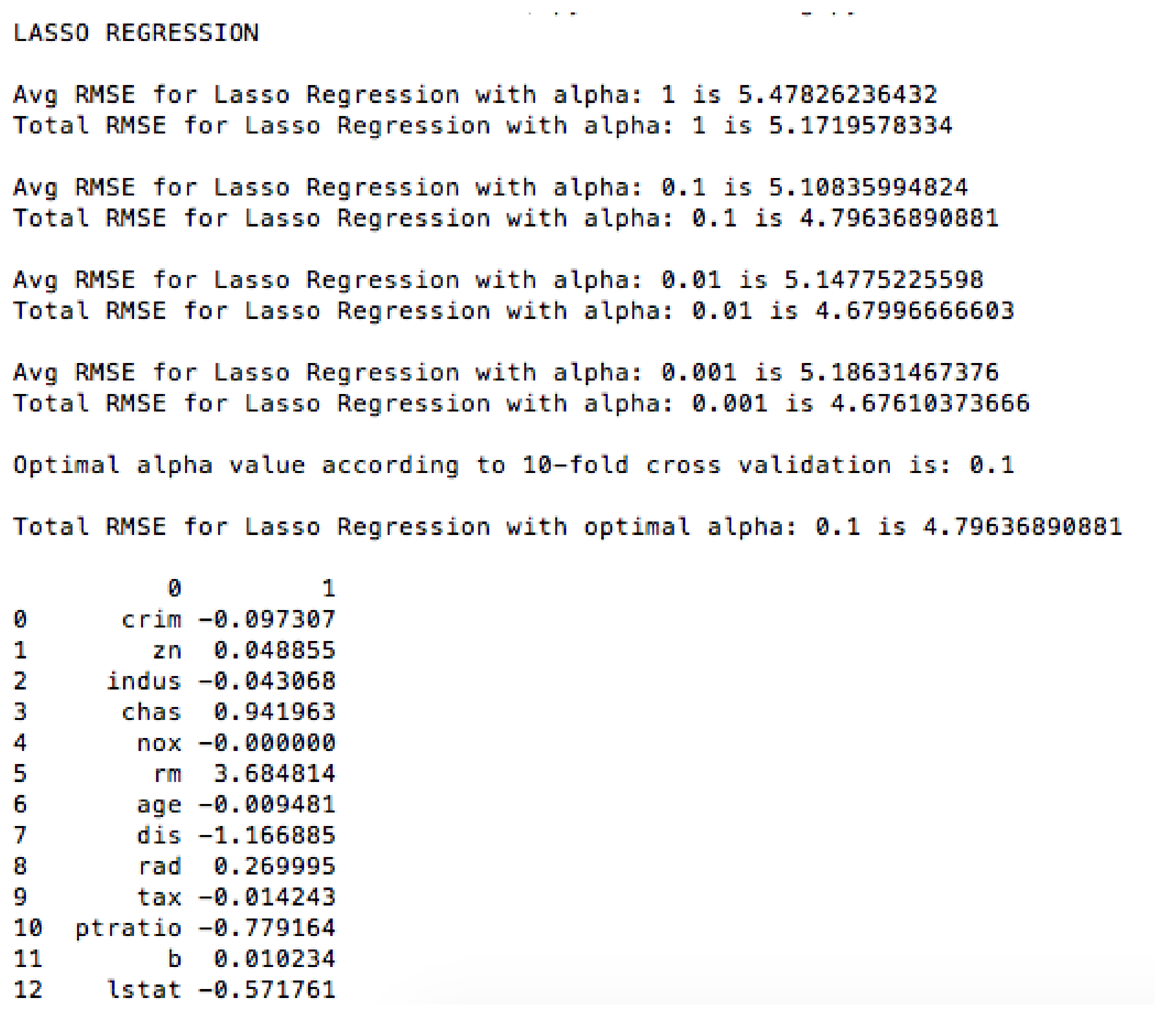
```
LINEAR REGRESSION

Avg RMSE: 5.19132765556
RMSE total is: 4.67606457754
           0          1
0       crim  -0.107417
1         zn   0.046121
2      indus   0.014269
3       chas   2.671108
4        nox -17.633641
5         rm   3.794307
6        age   0.001076
7        dis  -1.479179
8        rad   0.301534
9        tax  -0.012053
10   ptratio  -0.958874
11         b   0.009305
12     lstat  -0.527600
```

Figure 24: Linear Regression

5b) Tune alpha and report best RMSE as the previous step for lasso regularization.

And lastly we used the same process for Lasso regression in order to find the optimal alpha value according to the average RMSE generated from the 10-fold cross validation.

Running the function call **LassoRegs('housing_data.csv)** in the main function gave us the following output:

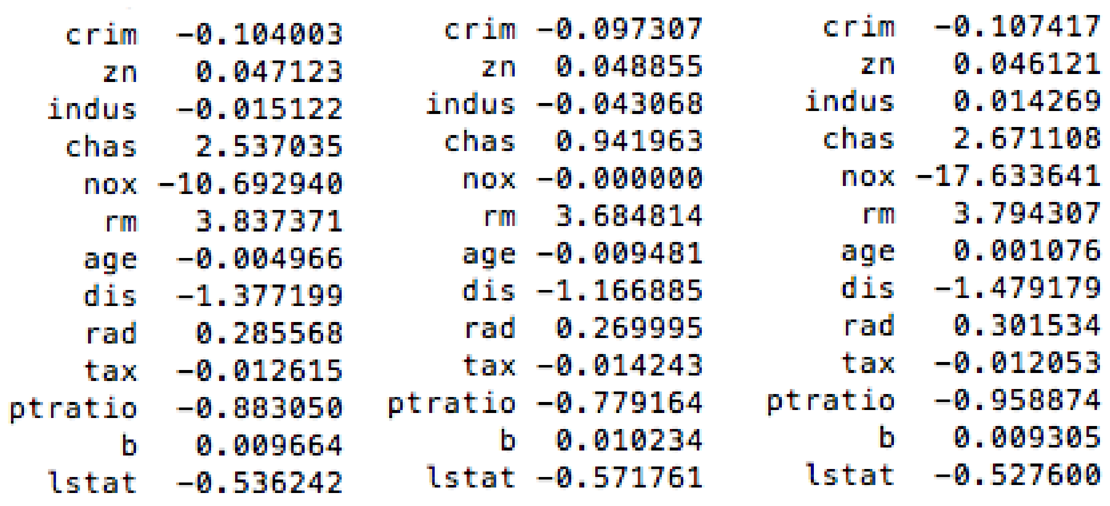


```
LASSO REGRESSION

Avg RMSE for Lasso Regression with alpha: 1 is 5.47826236432
Total RMSE for Lasso Regression with alpha: 1 is 5.1719578334

Avg RMSE for Lasso Regression with alpha: 0.1 is 5.10835994824
Total RMSE for Lasso Regression with alpha: 0.1 is 4.79636890881

Avg RMSE for Lasso Regression with alpha: 0.01 is 5.14775225598
Total RMSE for Lasso Regression with alpha: 0.01 is 4.67996666603

Avg RMSE for Lasso Regression with alpha: 0.001 is 5.18631467376
Total RMSE for Lasso Regression with alpha: 0.001 is 4.67610373666

Optimal alpha value according to 10-fold cross validation is: 0.1

Total RMSE for Lasso Regression with optimal alpha: 0.1 is 4.79636890881

          0         1
0      crim -0.097307
1        zn  0.048855
2     indus -0.043068
3      chas  0.941963
4       nox -0.000000
5        rm  3.684814
6       age -0.009481
7       dis -1.166885
8       rad  0.269995
9       tax -0.014243
10  ptratio -0.779164
11        b  0.010234
12    lstat -0.571761
```

Figure 25: Lasso Regression

Here we can see that the optimal value of alpha that we achieved in the given range was an alpha of **α = 0.1.**

In *Figure 26* we compare the optimal coefficients for Ridge, Lasso, and Linear (Un regularized) Regression. This was done using the **RidgeVLassoVUnreg('housing_data.csv')** function call.



```
    crim   -0.104003      crim -0.097307      crim   -0.107417
      zn    0.047123        zn  0.048855        zn    0.046121
   indus   -0.015122     indus -0.043068     indus    0.014269
    chas    2.537035      chas  0.941963      chas    2.671108
     nox  -10.692940       nox -0.000000       nox  -17.633641
      rm    3.837371        rm  3.684814        rm    3.794307
     age   -0.004966       age -0.009481       age    0.001076
     dis   -1.377199       dis -1.166885       dis   -1.479179
     rad    0.285568       rad  0.269995       rad    0.301534
     tax   -0.012615       tax -0.014243       tax   -0.012053
 ptratio   -0.883050   ptratio -0.779164   ptratio   -0.958874
       b    0.009664         b  0.010234         b    0.009305
   lstat   -0.536242     lstat -0.571761     lstat   -0.527600
```

Figure 26: Optimal coefficients of Ridge (left), Lasso (middle), and Linear (right) Regressions

Across each of the three regressions, we do not see drastic differences in the magnitudes of the coefficients except in a few cases. The Lasso Regression has a much smaller coefficient related to the nitric oxides concentration (parts per 10 million) then the other two.   The main difference across the three methods are the following: Ridge Regression is l_2 regularization in which we add the squared, two-norm of beta with some coefficient alpha while Lasso Regression is l_1 regularization in which we add the one-norm of beta with some coefficient alpha.