(a)Linear Regression

i. First convert each categorical feature into one dimensional numerical values using scalar encoding (e.g. Monday to Sunday can be mapped to 1-7), and then directly use them to fit a basic linear regression model.

First, we used the library, “Pandas” to load the “.csv” file as an object, and then replaced all the texts into numbers. After this, we could retrieve X and Y. X is the feature matrix, and Y is the vector consisted of supervised labels. The algorithm is shown below.

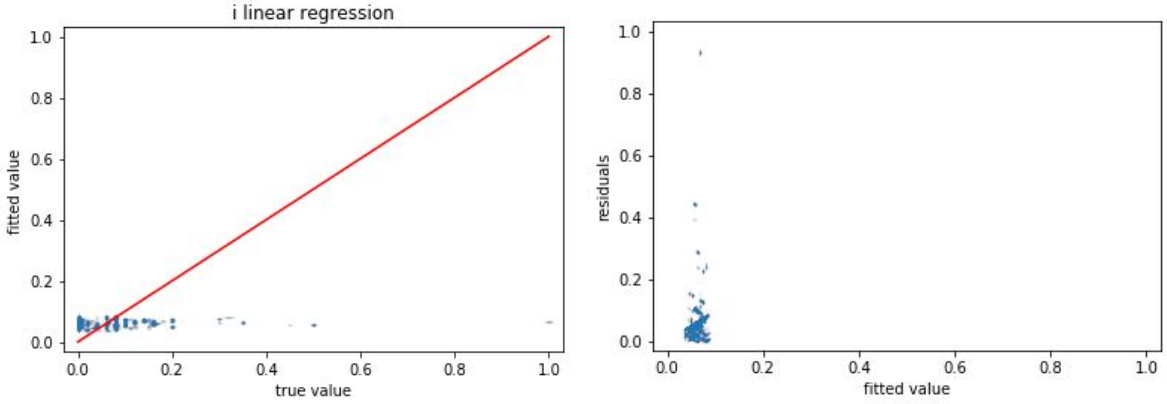
```
""" ====== Change to scalar ====== """

content = pd.read_csv("network_backup_dataset.csv")
content = content.replace({'Day of Week': {'Monday' : 0, 'Tuesday' : 1, 'Wednesday' : 2 ,
                                   'Thursday' : 3, 'Friday' : 4,
                                   'Saturday' : 5, 'Sunday' : 6 }})

WorkFlow_list = sorted(pd.unique(content['Work-Flow-ID']))
File_list     = sorted(pd.unique(content['File Name']))
num_WorkFlow  = len(WorkFlow_list)
num_File      = len(File_list)

for i in np.arange(num_WorkFlow):
    content = content.replace({'Work-Flow-ID': {'work_flow_%d'%i:i}})
for i in np.arange(num_File):
    content = content.replace({'File Name': {'File_%d'%i:i}})

content_all_np = content.values
X = content_all_np[: , 0:5]
Y = content_all_np[: ,  5 ]
```

Then, we splitted the X, Y into 10 folds and did the linear regression using cross validation. In each fold-set, 9 of folds were for training, and the last one was for testing. The training errors and the testing errors in each 10 validation were recorded. Also, we have the average training errors and the testing errors. The result is provided below.

```
Test_RMSE  fold: 1  =  0.10671805208020818
Train_RMSE fold: 1  =  0.10324315757258783
Test_RMSE  fold: 2  =  0.10018461437874714
Train_RMSE fold: 2  =  0.1039667779154282
Test_RMSE  fold: 3  =  0.10684977389075649
```

```
Train_RMSE fold: 3  =  0.10322579905716478
Test_RMSE  fold: 4  =  0.10036709167474589
Train_RMSE fold: 4  =  0.10394642919472956
Test_RMSE  fold: 5  =  0.10711585430913634
Train_RMSE fold: 5  =  0.10319511299709448
Test_RMSE  fold: 6  =  0.10044533653426363
Train_RMSE fold: 6  =  0.10393838251334332
Test_RMSE  fold: 7  =  0.10705026885111442
Train_RMSE fold: 7  =  0.10320263161789271
Test_RMSE  fold: 8  =  0.10046664550261591
Train_RMSE fold: 8  =  0.10393638595781884
Test_RMSE  fold: 9  =  0.10707418585670946
Train_RMSE fold: 9  =  0.10320098997210213
Test_RMSE  fold: 10  =  0.09994712085738942
Train_RMSE fold: 10  =  0.10399160058704292

Average Test_RMSE  =  0.1036218943935687
Average Train_RMSE =  0.10358472673852046
```

We plot the fitted value vs true value scattered over the data points, and the residuals values vs fitted values. It fitted very bad.

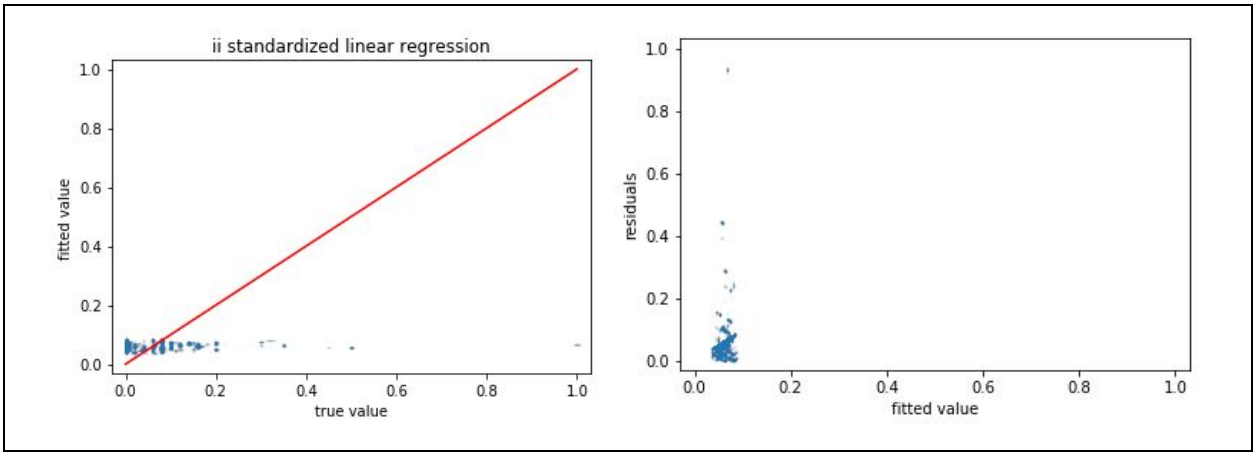


ii. Data Preprocessing: Standardize (see the Useful Functions Section) all these numerical features, then fit and test the model. How does the fitting result change as shown in the plots?

In this part, we standardized the features before feeding them into the regressor. Not surprising, the results were the same. Because by standardizing the input, we only result different interceptions in the parameter, the other parts of the regression line will be the same.

```
Test_RMSE  fold: 1  =  0.10671805208020818
Train_RMSE fold: 1  =  0.10324315757258784
Test_RMSE  fold: 2  =  0.10018461437874714
Train_RMSE fold: 2  =  0.1039667779154282
Test_RMSE  fold: 3  =  0.10684977389075649
Train_RMSE fold: 3  =  0.10322579905716478
Test_RMSE  fold: 4  =  0.1003670916747459
Train_RMSE fold: 4  =  0.10394642919472956
Test_RMSE  fold: 5  =  0.10711585430913634
Train_RMSE fold: 5  =  0.10319511299709448
Test_RMSE  fold: 6  =  0.10044533653426362
Train_RMSE fold: 6  =  0.10393838251334332
Test_RMSE  fold: 7  =  0.10705026885111442
Train_RMSE fold: 7  =  0.10320263161789271
Test_RMSE  fold: 8  =  0.1004666455026159
Train_RMSE fold: 8  =  0.10393638595781884
Test_RMSE  fold: 9  =  0.10707418585670946
Train_RMSE fold: 9  =  0.10320098997210213
Test_RMSE  fold: 10  =  0.09994712085738942
Train_RMSE fold: 10  =  0.10399160058704292

Average Test_RMSE  =  0.10362189439356868
Averagerall Train_RMSE =  0.10358472673852046
```

We plot the fitted value vs true value scattered over the data points, and the residuals values vs fitted values. It fitted very bad,too.

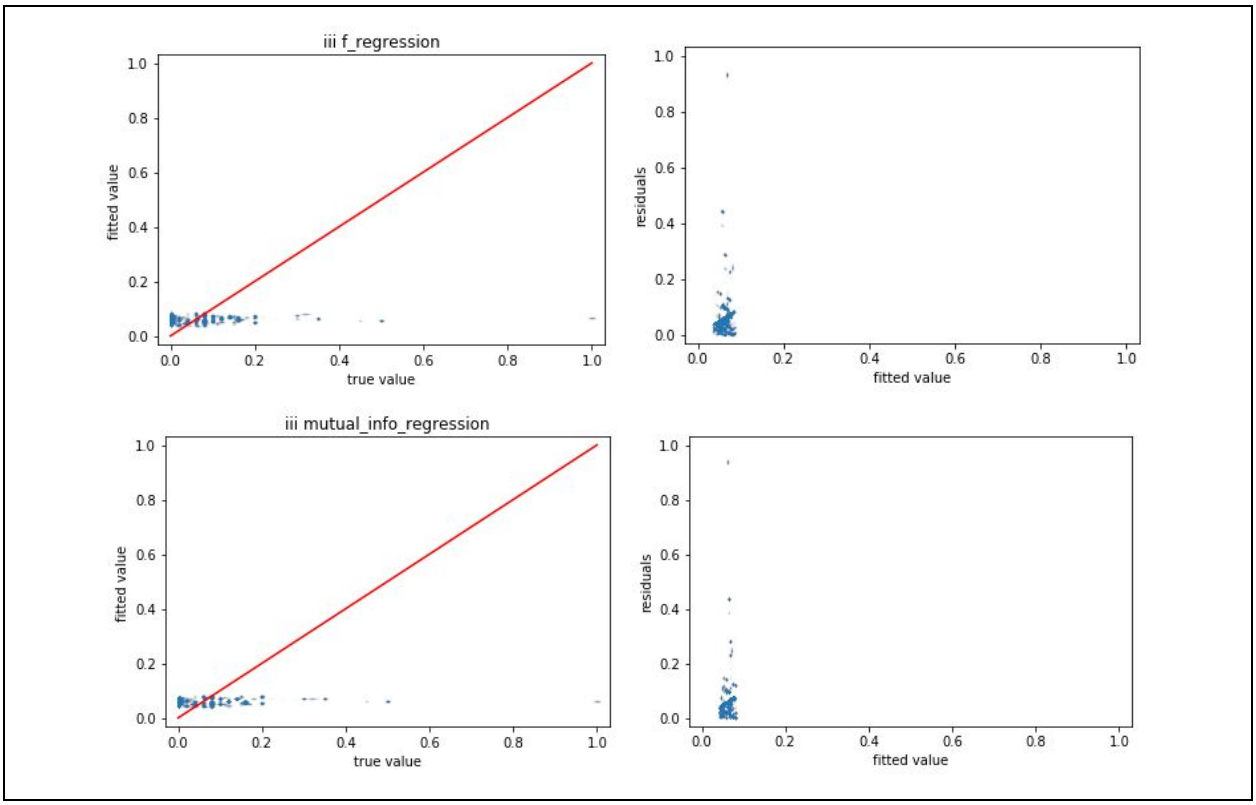


iii Feature Selection: Use f regression and mutual information regression measure to select three most important variables respectively. Report thet three most important variables you find. Use those three most important variables to train a new linear model, does the performance improve?

In this part, we have tried the f_regression and mutual_info_regression. The importances of each features via the two methods are shown below. By using the f_regression's features, the result improved a little, but for the mutual_info_regression, it got worse a little. We can infer that, although some features seems more important based on the measures. These less important features still play some rules. Furthermore, we just assume the system is linear, which may not be true.

f_regression	mutual_info_regression
0 5.60569e-05 1 0.257504 2 1 3 0.173401 4 0.167971	0 0.00235085 1 0.304276 2 0.385706 3 1 4 0.998778
best features: 123	best features: 234
Test_RMSE = 0.10361671302936296	Test_RMSE = 0.10371576817323373

We plot the fitted value vs true value scattered over the data points, and the residuals values vs fitted values. Still they fitted very bad.


iii f_regression
fitted value
true value
residuals
fitted value
iii mutual_info_regression
fitted value
true value
residuals
fitted value

iv Feature Encoding: As explained in the preceding discussions, there are 32 possible combinations of encoding the five categorical variables. Plot the average training RMSE and test RMSE for each combination (in range 1 to 32). Which combinations achieve best performance? Can you provide an intuitive explanation?

In this part, we introduced the “Hot-One-Encoding.” Instead of changing all the “scalar features” into “hot-one features,” each of the features can be assigned to either of the two methods. And then, we tested the performance via the 32 combinations. To achieve the combination, we used the binary mapping. The function is defined below. The “hybrid features” and the “combination indices,” like “01001” or “00010” were generated. “0” stands for “one-hot feature,” and “1” stands for the “scalar feature.”

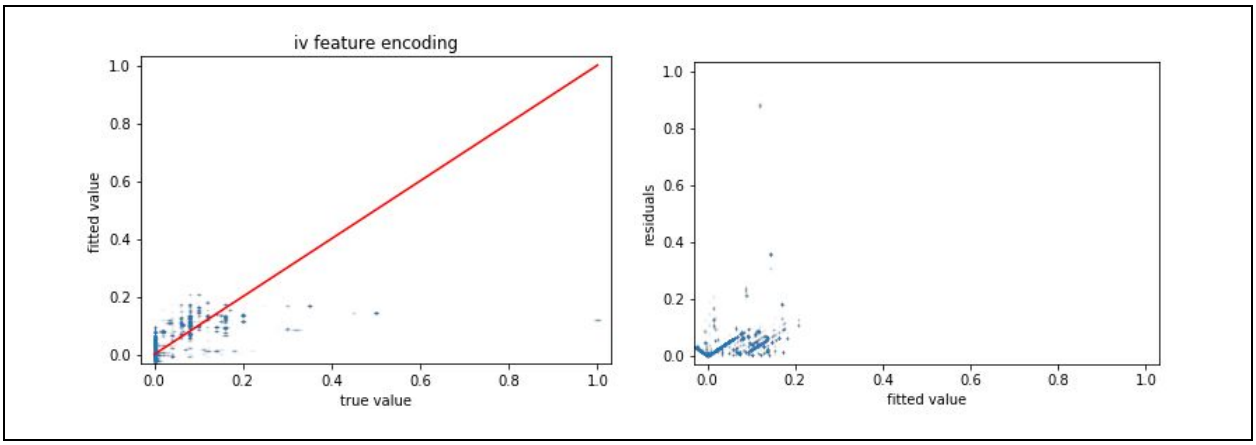
```
def hot32comb(X):

    # the 32 combinations, split X into 5 columns
    enc1 =  preprocessing.OneHotEncoder(sparse = False)
    num_sample = X.shape[0]

    X0 = X[:,0].reshape((num_sample,1))
    X1 = X[:,1].reshape((num_sample,1))
    X2 = X[:,2].reshape((num_sample,1))
    X3 = X[:,3].reshape((num_sample,1))
    X4 = X[:,4].reshape((num_sample,1))

    X_Hot0 = enc1.fit_transform(X0)
    X_Hot1 = enc1.fit_transform(X1)
    X_Hot2 = enc1.fit_transform(X2)
    X_Hot3 = enc1.fit_transform(X3)
    X_Hot4 = enc1.fit_transform(X4)

    X_Very_Hot = []
    Hot_comb   = []

    for i in np.arange(32):

        a = str(bin(i))[2:].rjust(5,'0')
        H0 = X0*int(a[0]) + X_Hot0*(1-int(a[0]))
        H1 = X1*int(a[1]) + X_Hot1*(1-int(a[1]))
        H2 = X2*int(a[2]) + X_Hot2*(1-int(a[2]))
        H3 = X3*int(a[3]) + X_Hot3*(1-int(a[3]))
        H4 = X4*int(a[4]) + X_Hot4*(1-int(a[4]))

        X_Very_Hot_i = np.concatenate((H0,H1,H2,H3,H4),axis =1)
        X_Very_Hot.append(X_Very_Hot_i)
```

```
        Hot_comb.append(a)

    print("X_Very_Hot generated")
    return X_Very_Hot,Hot_comb
```

And then, we fed the “transformed features” into the k-fold linear regression again. The result is shown below. We can see some extremely large errors like all the combinations start with “0”, i.e., choosing “one-hot” for the “week” feature. Therefore, we should not transform this feature into “one-hot” in the future. The combination with the smallest error was 10010, which the error was 0.088362727980649. We can see that the Hot-One-Encoding worked.

'00000': 14716760786.520405	'10000': 0.08837466435725168
'00001': 575317902.5219147	'10001': 0.08837111476723367
'00010': 5012899713.035536	'10010': 0.088362727980649
'00011': 468456632.43596303	'10011': 0.10092722002382389
'00100': 4864851846.814598	'10100': 0.0897723051994113
'00101': 1616078044.79514	'10101': 0.08977048513320296
'00110': 5116633690.021553	'10110': 0.0897525769902775
'00111': 3333279793.5643406	'10111': 0.10221283916093334
'01000': 8662784496.061806	'11000': 0.08997766960853824
'01001': 346006938.40653944	'11001': 0.08997579432600858
'01010': 8162010831.54614	'11010': 0.08998532873712582
'01011': 861225978.4243702	'11011': 0.10243109433708475
'01100': 5516894073.0442295	'11100': 0.09134626622102046
'01101': 444689199.8642498	'11101': 0.09133173770620374
'01110': 5652055026.767418	'11110': 0.09132864969688585
'01111': 2236463060.6911	'11111': 0.10359812722230885

At last, we use the best parameters to run over all dataset and plotted the following two figures. The one with fitted value against true value scattered over the data points and the one with residuals values against fitted values. It fitted much better than before.


iv feature encoding
fitted value
true value
residuals
fitted value
0.0
0.2
0.4
0.6
0.8
1.0

v. Controlling ill-conditioning and over-fiting: You should have found obvious increases in test RMSE compared to training RMSE in some combinations, can you explain why this happens? Observe those fitted coefficients. To solve this problem, you can try the following regularizations with suitable parameters.

In this section, we were still using the linear model. However, we introduced the regulations to prevent overfitting. Three regression with different regulators were used. 1. Ridge regression. 2. Lasso regression. 3. The combination of the previous two, the Elastic Net Regresssion. In this part, we also transformed the features into 32 combinations of “scalar” and “one-hot.” All of them were fed into the regressors.

We wanted to use the “linear_model.ElasticNet” to do all ridge, lasso and elastic. However, when we set l1_ratio = 0, the algorithm encoutered an error. Therefore, we just performed them in 3 ways.

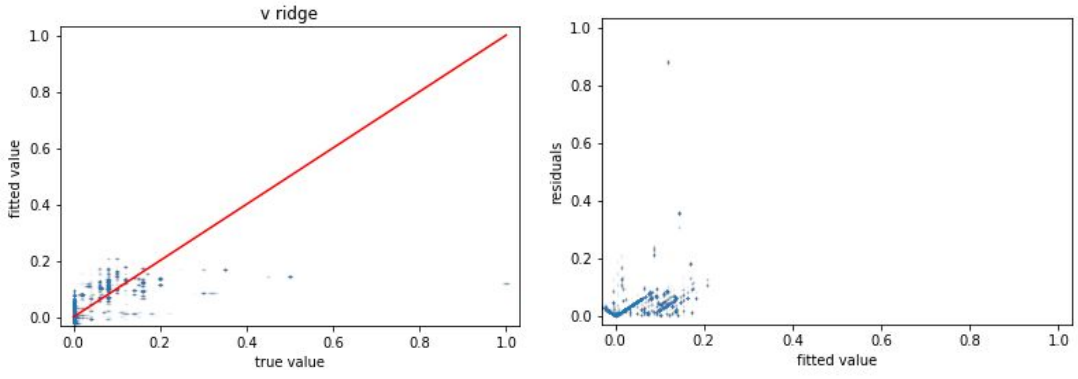
```
def hot32acc_best(X_Very_Hot,Hot_comb,Y,reg_model,alpha,l1_ratio):
    all_test_RMSE_i = 0
    best_RMSE       = 0
    best_comb       = 0
    for i in np.arange(32):
        X_in = X_Very_Hot[i]
        if reg_model == 'ridg':
            _ ,_ , all_test_RMSE_i = kfold_ridg(X_in,Y,alpha)
        elif reg_model == 'lass':
            _ ,_ , all_test_RMSE_i = kfold_lass(X_in,Y,alpha)
        elif reg_model == 'elst':
            _ ,_ , all_test_RMSE_i = kfold_elst(X_in,Y,alpha,l1_ratio)
        else:
            print("Wrong Input")
            break
        name = Hot_comb[i]
        if i ==0:
            best_RMSE = all_test_RMSE_i
            best_comb = name
        if all_test_RMSE_i < best_RMSE:
            best_RMSE = all_test_RMSE_i
            best_comb = name
    return best_RMSE,best_comb
```

=== 1. Ridge ===

The alpha's that we have tested were: [0.1,0.3,0.5,1,3,5,10,20,40,80,120]. The result is shown below. We found that when alpha = 5, we got the smallest error, which was 0.08836773802910462. The combination was "10001." The algorithm tend to choose hot-one features.

	0	1	2
0	0.1	10001	0.0883678
1	0.3	10001	0.0883678
2	0.5	10001	0.0883678
3	1	10001	0.0883678
4	3	10001	0.0883677
5	5	10001	0.0883677
6	10	10001	0.0883678
7	20	10001	0.0883682
8	40	10001	0.0883701
9	80	10000	0.0883759
10	120	10000	0.0883855

Visualize fitted values vs true values and residual vs fitted values. It fitted good.



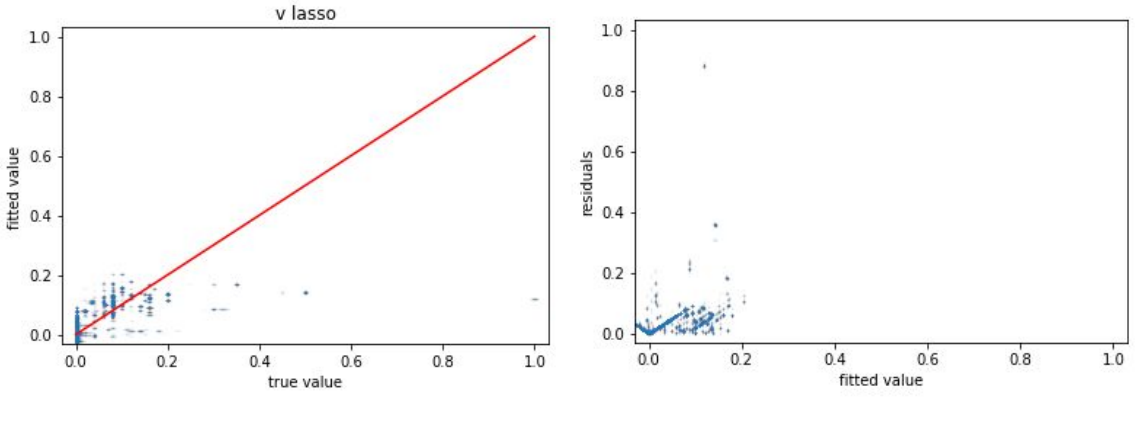
=== 2. Lasso ===

The alpha's that we have tested were: [0.0001,0.001,0.002,0.005,0.01,0.1,0.3,0.5,1,3,5,10,20,40,80,120]. The result is shown below. We found that when alpha = 0.0001, we got the smallest error, which was 0.08837160165492905. combination was "00000." surprisedly, by choosing all "one-hot features," we got an acceptable result with Lasso. Actually, we've tried "0.00001," but our python release runtime error. Back one thousand steps to say, if the regulation parameter is too small, we would rather not use it.

	0	1	2
0	0.0001	0	0.0883716
1	0.001	0	0.0888033
2	0.002	0	0.0897528
3	0.005	101	0.0936564
4	0.01	101	0.0988953

	0	1	2
0	0.1	0	0.10414
1	0.3	0	0.10414
2	0.5	0	0.10414
3	1	0	0.10414
4	3	0	0.10414
5	5	0	0.10414
6	10	0	0.10414
7	20	0	0.10414
8	40	0	0.10414
9	80	0	0.10414
10	120	0	0.10414

Visualize fitted values vs true values and residual vs fitted values. It fitted good.



=== 3. Elastic ===

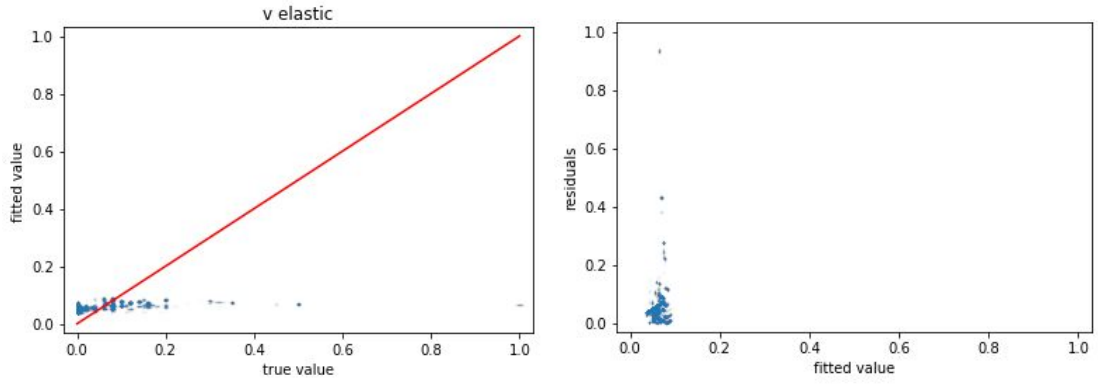
In this part, the instead of choosing lambda1 and lambda2, we chose alpha = lambda1+lambda2 and the l1_ratio. The alpha's and l1_ratio's we have tested were:

alphas's	[0.5,5.0,20.0,80.0,120.0,200.0]
l1_ratio's	[0.01,0.1,0.3,0.5,0.7,0.9,0.99]

The result is shown below. The rows are the alpha's while the columns are the l1_ratio's. We can see that once the coefficient of Lasso increased to certain amount. The error went to a certain value. When alpha=0.01 and l1_ratio=0.01, we got the best result.

	0	1	2	3	4	5	6	7
0	0	0.01	0.1	0.3	0.5	0.7	0.9	0.99
1	0.5	0.10021	0.10398	0.10414	0.10414	0.10414	0.10414	0.10414
2	5	0.103982	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414
3	20	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414
4	80	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414
5	120	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414
6	200	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414	0.10414

Visualize fitted values vs true values and residual vs fitted values. It fitted bad.



In the experiment above, we can see that the ridge regression was a better regressor than the others of this system.

(d) Predict the Backup size for each of the workflows separately.

i. Using linear regression model.

In this part, we predicted the outputs respect to the workflows separately. And again, the combination of “scalar” and “one-hot” among the 4 features still were still applied in this linear regression part. The result is shown below. Note that the errors of work_flow 0,2,3 were low. That meant these work_flow’s were more linear than the others.

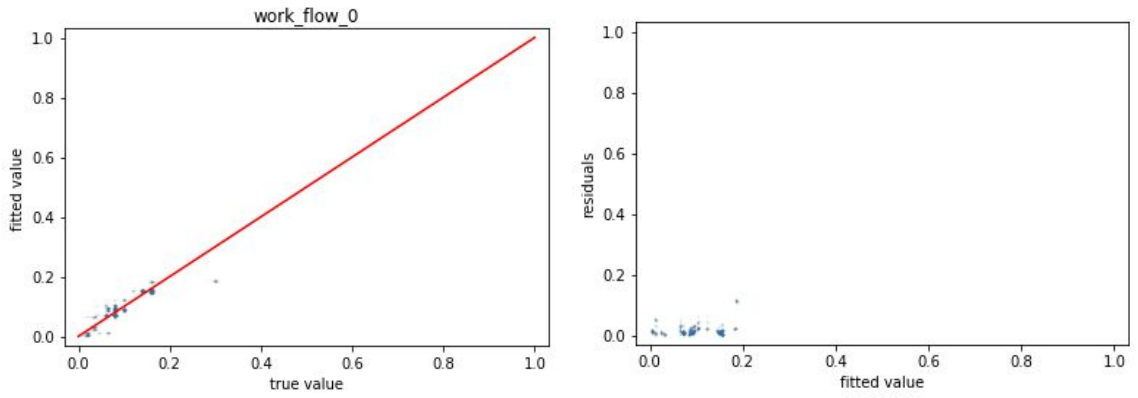
Test errors

Key	Type	Size	Value
work_flow_0	float64	1	0.025518351854589373
work_flow_1	float64	1	0.11239622690453732
work_flow_2	float64	1	0.030898544174410524
work_flow_3	float64	1	0.005271988971669496
work_flow_4	float64	1	0.054196071085722254

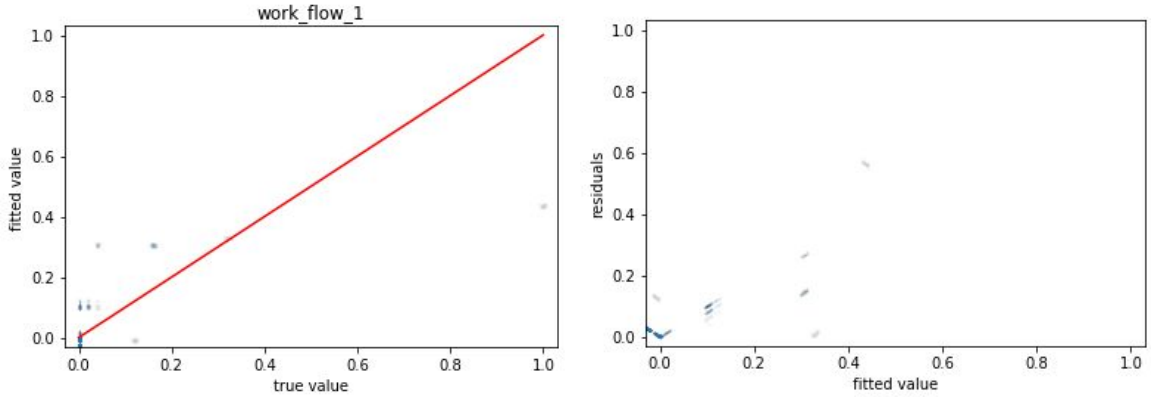
Train errors

Key	Type	Size	Value
work_flow_0	float64	1	0.025519732378102705
work_flow_1	float64	1	0.1134477906908699
work_flow_2	float64	1	0.03096898087332387
work_flow_3	float64	1	0.005300703767936993
work_flow_4	float64	1	0.054038041259421785

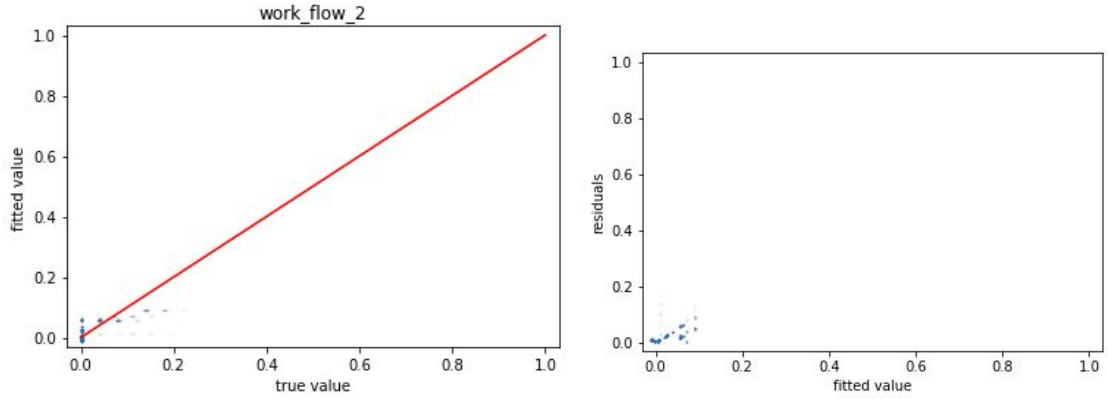
Visualize fitted values vs true values and residual vs fitted values.



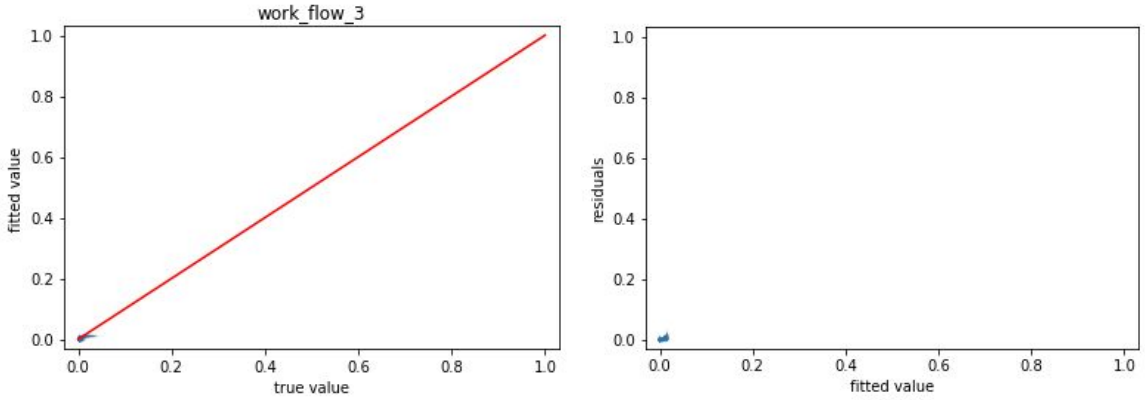
Very good fitting



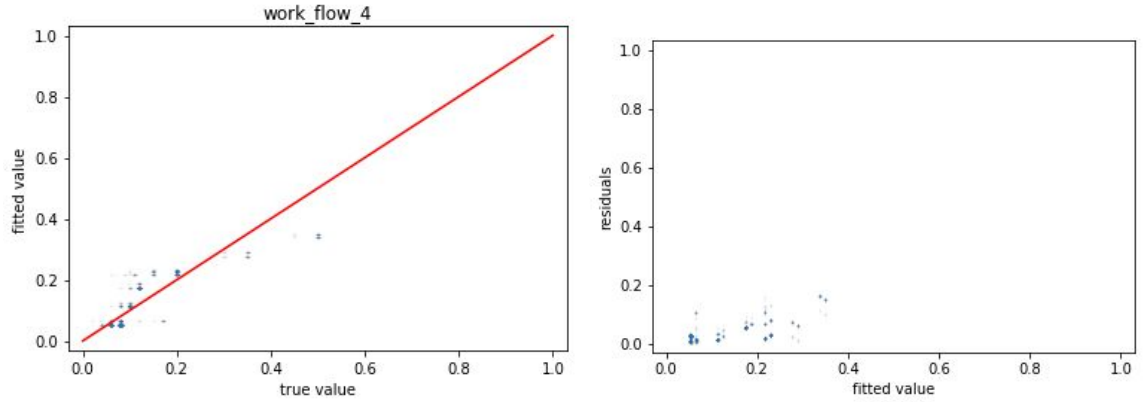
Trend is OK, but not very good fitting.



Bad fitting



Bad fitting, but they squeezed together. Therefore result good RMSE.



Good fitting

ii. Using polynomial regression model.

In this part, instead of linear regression, we want to use higher order polynomials to do the regression. Because the one-hot vectors are all “0” and “1,” it is not meaningful to do the power. Therefore, we did not do any one-hot encoding in this part. Also, higher order model is prone to overfit, so we need to apply regulation. We have found that the lasso is a bad method in this part, nor the elastic. Hence, we only utilized the ridge regulator.

We tried the order from 1 to 11 with the 5 work_flow’s. The best results of them are shown below.

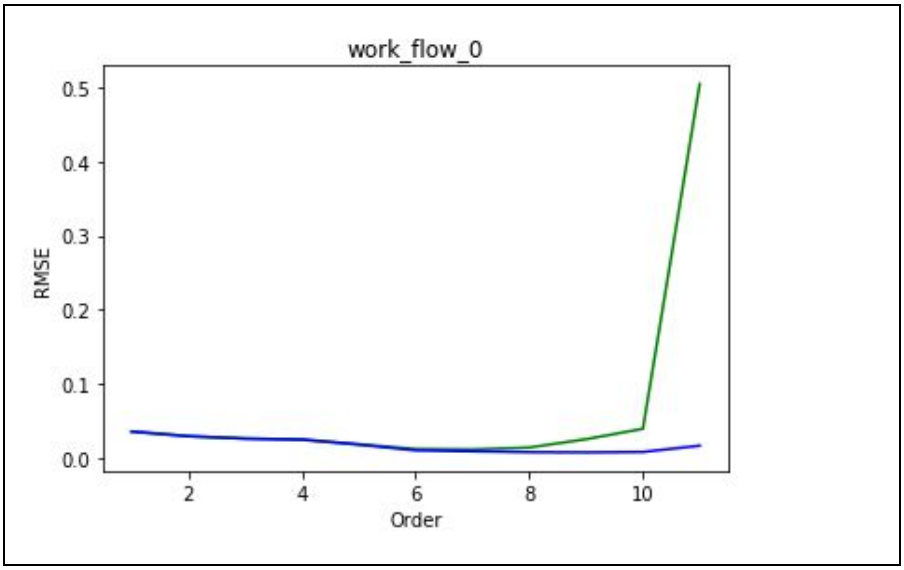
```
RMSE_ridge generated wf =  0
Best test error =  0.011638935096083375
Avg train error =  0.009154225947658873
order =  7.0

RMSE_ridge generated wf =  1
Best test error =  0.01180625812820955
Avg train error =  0.005679696713554399
order =  9.0

RMSE_ridge generated wf =  2
Best test error =  0.024872236813356548
Avg train error =  0.021790168227113504
order =  6.0

RMSE_ridge generated wf =  3
Best test error =  0.005016268656012702
Avg train error =  0.004900196414962753
order =  5.0

RMSE_ridge generated wf =  4
Best test error =  0.02989399232124109
Avg train error =  0.026401115912749618
order =  6.0
```

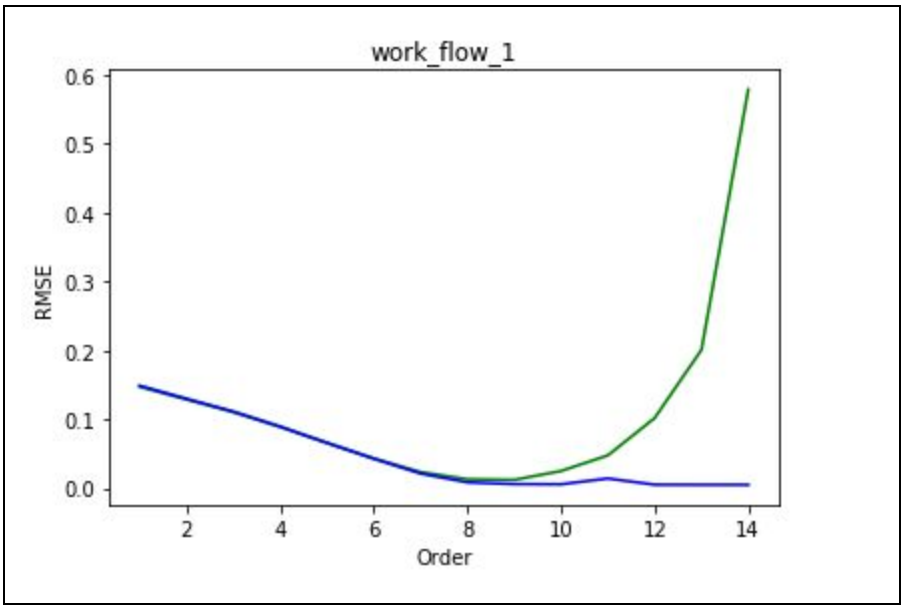
We plotted the RMSE's of test and train vs order.

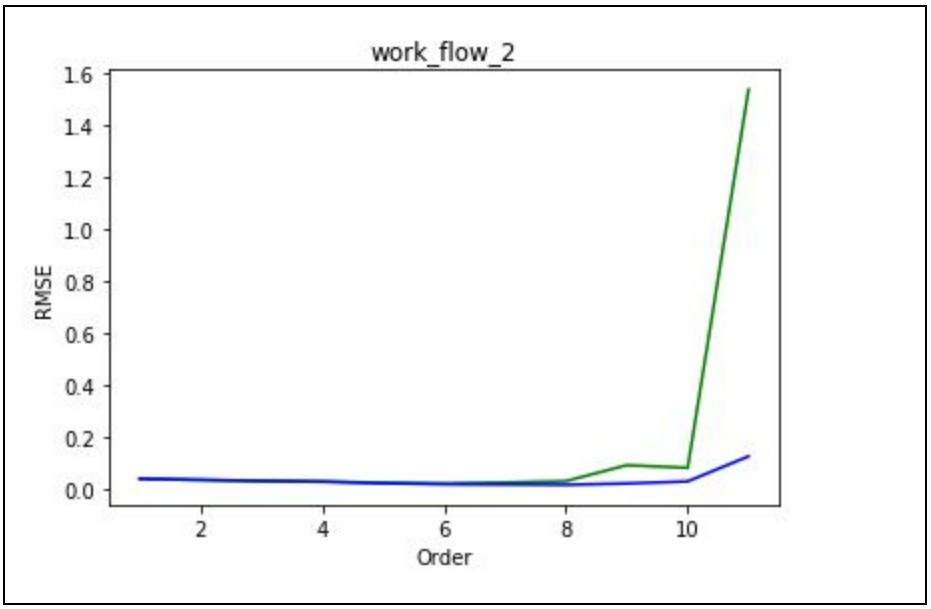
We know that work_flow 0,2,3,4 have low RMSE, and we assume them to be more linear like. Therefore, we can see they have acceptable test errors while the order were small. On the contrary, work_flow 1 had very large RMSE during the low order. Therefore, this system is very not linear. Furthermore, we can see in work_flow 0,2,3,4 that the test RMSE went very high around the order of 10. These were the overfitting. By increasing the regulation parameter, this phonenum should reduce. However, we can not rely on the regulation parameter. We should choose a right order. For work_flow1, we tested on more order to see the thredshold of overfitting.
The thredshold of them were: [11,14,11,10,11]

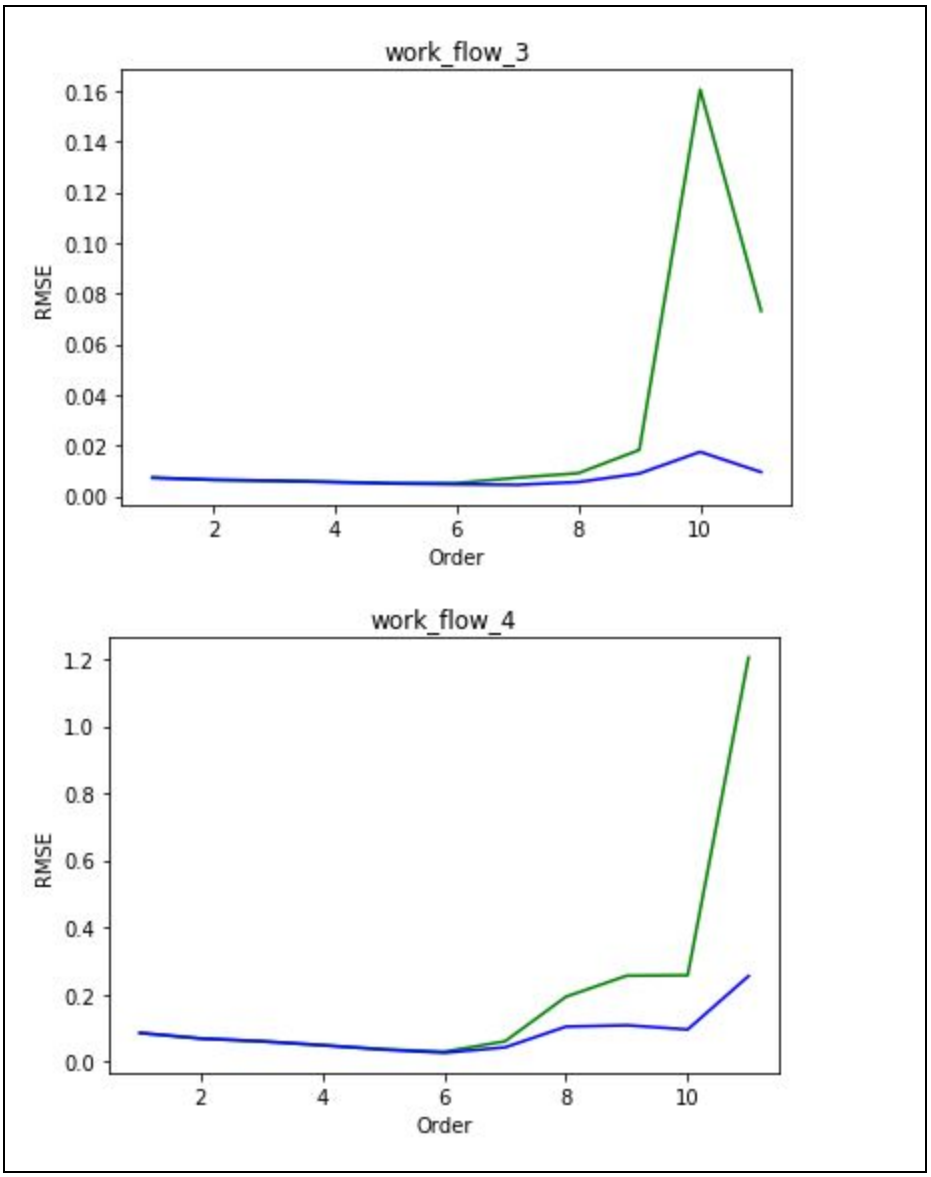
The cross validation helps controlling the complexity by providing us the right hyperparameters such as the polynomial order. By taking out one fold as testing, or say validating fold, we can get an unbiased RMSE. To be even, we did this 10 times and made mean.

The green curves are the test RMSE, the blue curves are train RMSE

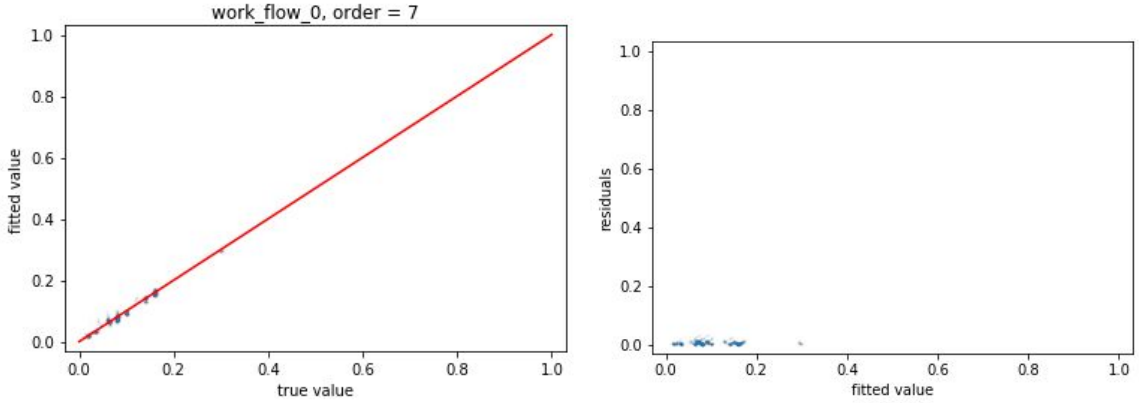



work_flow_1
RMSE
Order
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2
4
6
8
10
12
14

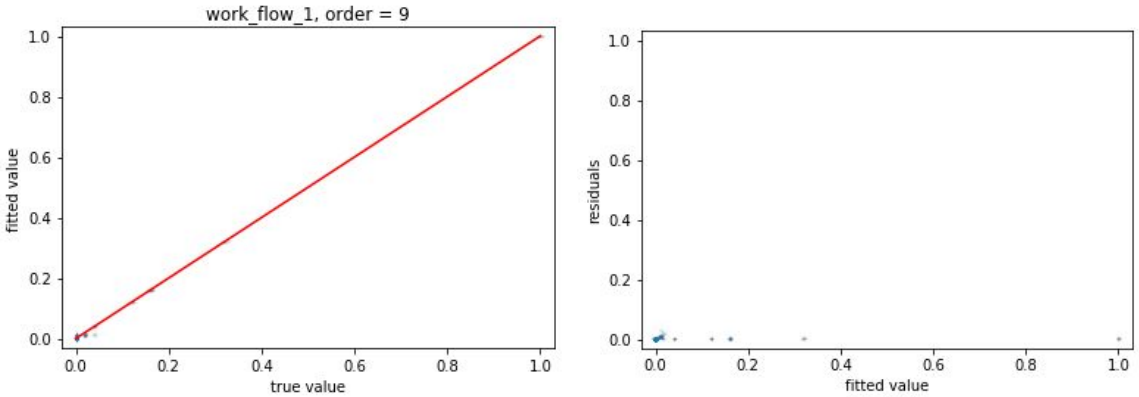

work_flow_2
RMSE
Order
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
2
4
6
8
10


work_flow_3
RMSE
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2
4
6
8
10
Order
work_flow_4
RMSE
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
Order

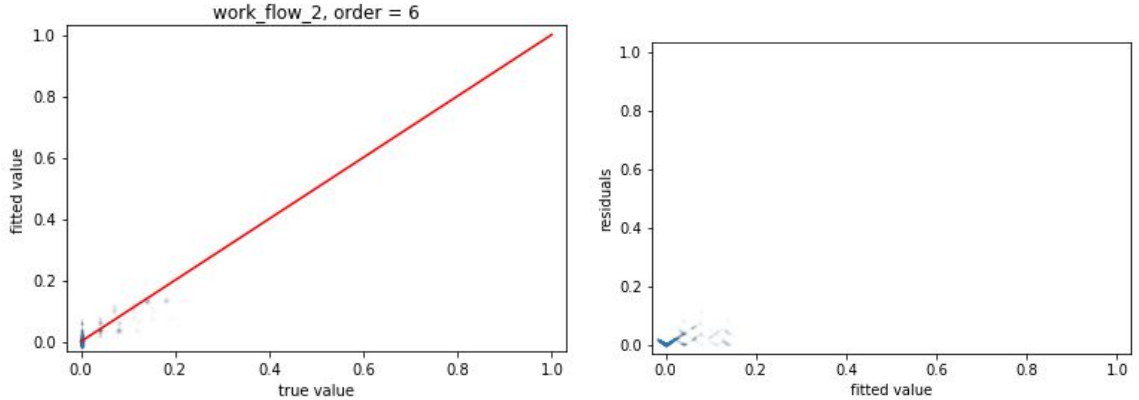
Visualize fitted values vs true values and residual vs fitted values.



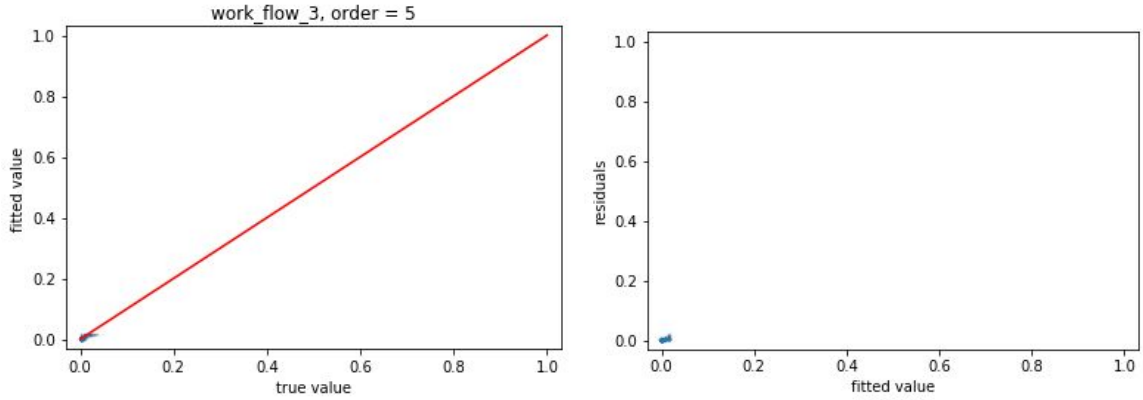
Very good fitting



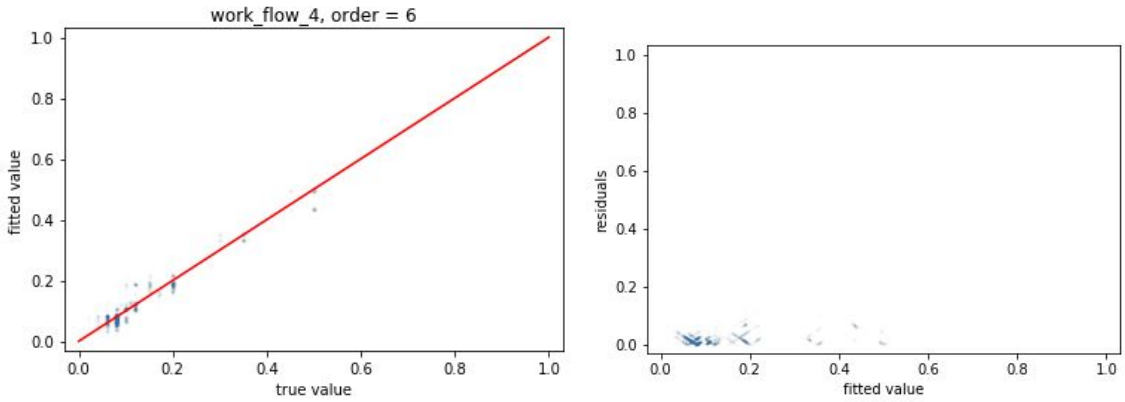
Very good fitting



Good fitting



Bad fitting, but they squeezed together. Therefore result good RMSE.



Good fitting