Project 3 report
Liyuan Zhao
005692591

Question 1: Standardize feature columns and prepare them for training.

Question 1

In [6]:
```
from sklearn.preprocessing import StandardScaler

#numerize all categorical data
data_copy = data.copy()
for i in categorical_features:
    cat_arr = list(data_copy[i].unique())
    len_ = len(data_copy[i].value_counts())
    data_copy[i].replace(cat_arr,[n for n in range(1, len_+1)], inplace=True)

scaler = StandardScaler()
diamond_X = data_copy.drop('price', axis=1) # remove the target
diamond_X_std = scaler.fit_transform(diamond_X)
diamond_std = scaler.fit_transform(data_copy)
diamond_y_std = diamond_std[:, -1]
```

There are three categorical features in diamond file: cut, color, and clarity. Because these categorical features are related to quality features, we are numerical encoding them. After that, I am using standard scalar to standardize features, as well as the target label, which is price.

Same process in gas file, except that we are adding year columns as a categorical feature and converting year feature to numerical feature. We are only predicting CO in this task so we are dropping the column NOX.

```
data5 = pd.read_csv('gt/gt_2015.csv')
```

In [57]:
```
data1['year'] = 2011
data2['year'] = 2012
data3['year'] = 2013
data4['year'] = 2014
data5['year'] = 2015
data = pd.concat([data1, data2,data3,data4,data5], ignore_index=True)
data['year']= data['year'].astype(str)
data.shape
```

Out[57]: (36733, 12)

Question 2: Plot a heatmap of the Pearson correlation matrix of the dataset columns.

Correlation matrix for diamond

Out[7]:

	carat	cut	color	clarity	depth	table	x	y	z	price
carat	1.000000	0.114426	-0.065386	-0.281218	0.028224	0.181618	0.975094	0.951722	0.953387	0.921591
cut	0.114426	1.000000	-0.029128	-0.118670	0.169916	0.381988	0.105361	0.105319	0.126726	0.049423
color	-0.065386	-0.029128	1.000000	0.032589	-0.001665	-0.033998	-0.055268	-0.054762	-0.055684	-0.016562
clarity	-0.281218	-0.118670	0.032589	1.000000	-0.025877	-0.133692	-0.305918	-0.297037	-0.299969	-0.140271
depth	0.028224	0.169916	-0.001665	-0.025877	1.000000	-0.295779	-0.025289	-0.029341	0.094924	-0.010648
table	0.181618	0.381988	-0.033998	-0.133692	-0.295779	1.000000	0.195344	0.183760	0.150929	0.127134
x	0.975094	0.105361	-0.055268	-0.305918	-0.025289	0.195344	1.000000	0.974701	0.970772	0.884436
y	0.951722	0.105319	-0.054762	-0.297037	-0.029341	0.183760	0.974701	1.000000	0.952006	0.865422
z	0.953387	0.126726	-0.055684	-0.299969	0.094924	0.150929	0.970772	0.952006	1.000000	0.861250
price	0.921591	0.049423	-0.016562	-0.140271	-0.010648	0.127134	0.884436	0.865422	0.861250	1.000000

Heatmap of matrix for diamond

Correlation matrix for gas

Out[8]:

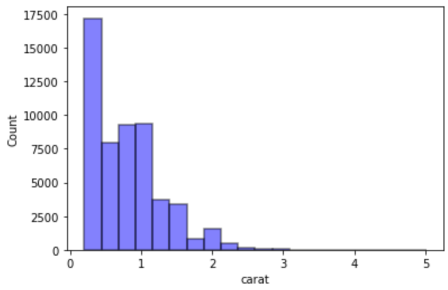
	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	year	CO
AT	1.000000	-0.406601	-0.476291	0.251974	0.045851	0.183706	0.281869	-0.091152	0.015287	0.000217	-0.174326
AP	-0.406601	1.000000	-0.015184	-0.040363	0.057533	-0.005390	-0.225601	0.118224	0.102636	0.018595	0.067050
AH	-0.476291	-0.015184	1.000000	-0.147840	-0.235153	-0.221809	0.022965	-0.137360	-0.196275	-0.177964	0.106586
AFDP	0.251974	-0.040363	-0.147840	1.000000	0.678485	0.691292	-0.466882	0.665483	0.702568	-0.254326	-0.448425
GTEP	0.045851	0.057533	-0.235153	0.678485	1.000000	0.874234	-0.699703	0.964127	0.978470	0.050643	-0.518909
TIT	0.183706	-0.005390	-0.221809	0.691292	0.874234	1.000000	-0.380862	0.910297	0.908469	-0.125743	-0.706275
TAT	0.281869	-0.225601	0.022965	-0.466882	-0.699703	-0.380862	1.000000	-0.682396	-0.706438	0.116287	0.058353
TEY	-0.091152	0.118224	-0.137360	0.665483	0.964127	0.910297	-0.682396	1.000000	0.988778	-0.029414	-0.569813
CDP	0.015287	0.102636	-0.196275	0.702568	0.978470	0.908469	-0.706438	0.988778	1.000000	-0.025415	-0.551027
year	0.000217	0.018595	-0.177964	-0.254326	0.050643	-0.125743	0.116287	-0.029414	-0.025415	1.000000	0.178450
CO	-0.174326	0.067050	0.106586	-0.448425	-0.518909	-0.706275	0.058353	-0.569813	-0.551027	0.178450	1.000000

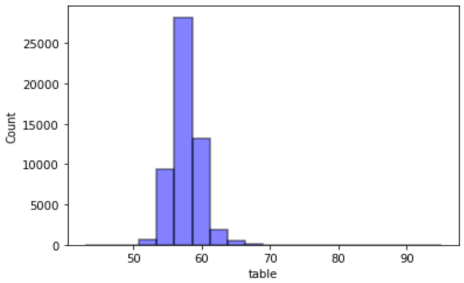
Heatmap of matrix for gas

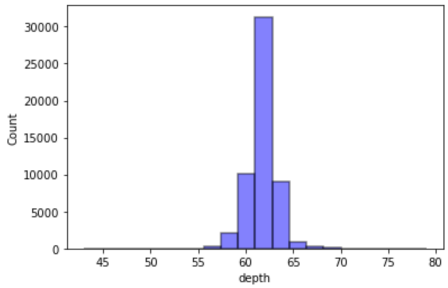
For diamond dataset, it shows that carat, x,y and z have the highest absolute correlation with the target price. It demonstrates that carat, x,y and z are most informative in terms of predicting the price of diamond

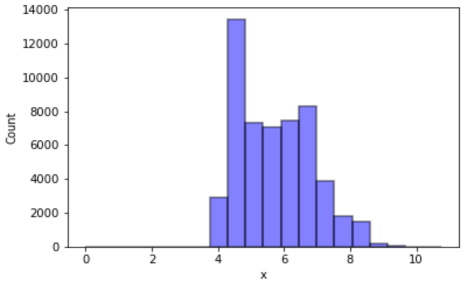
For gas (CO) dataset, it shows that year and AH have the highest absolute correlation with the target CO. However, the correlation values are slightly smaller overall. It demonstrates that year and AHare most informative in terms of predicting the value of CO.

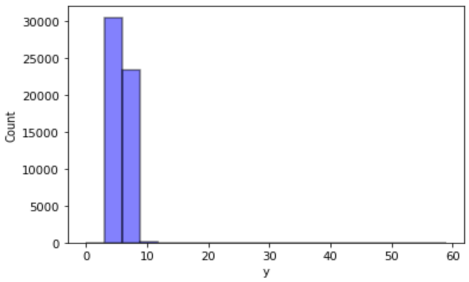
Question 3: Plot the histogram of numerical features. What preprocessing can be done if the distribution of a feature has high skewness?

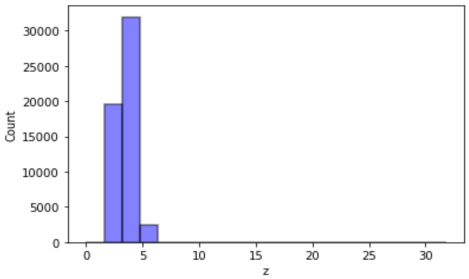


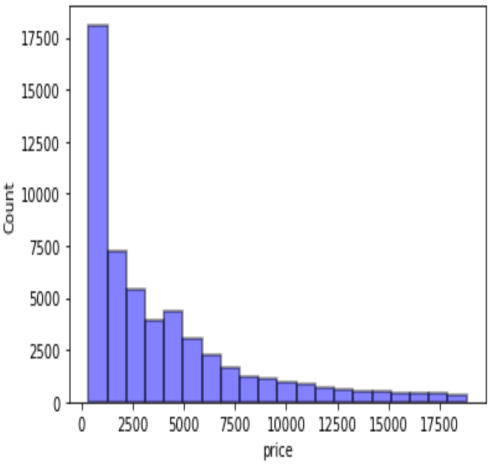




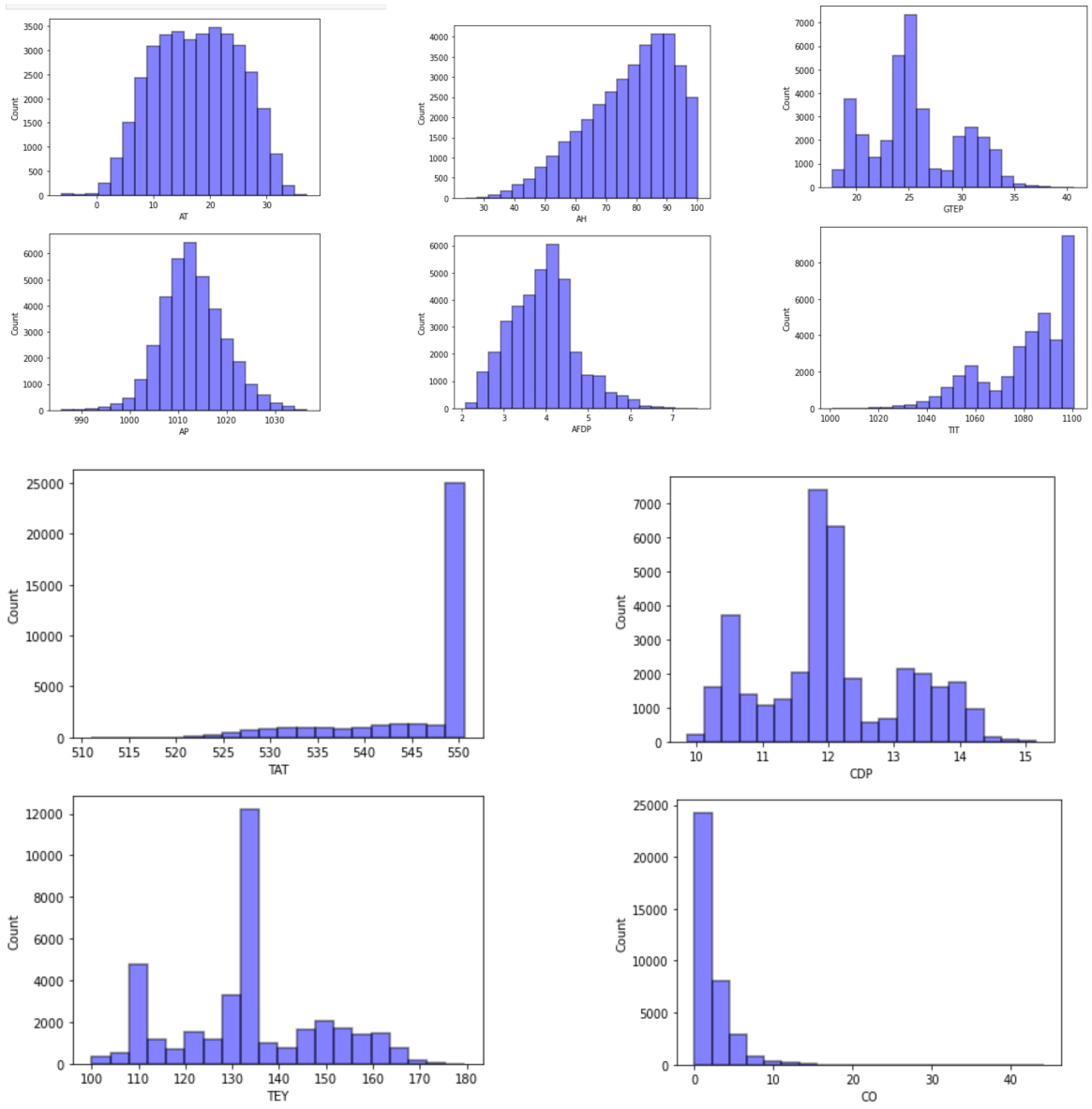








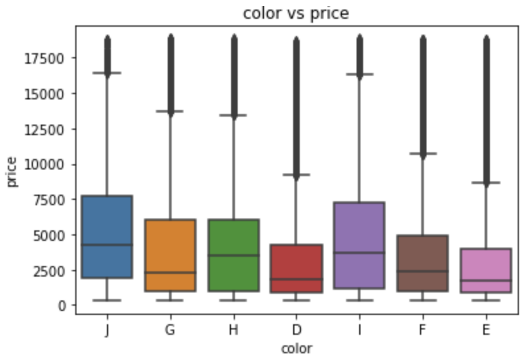
The above histograms are the diamond dataset.

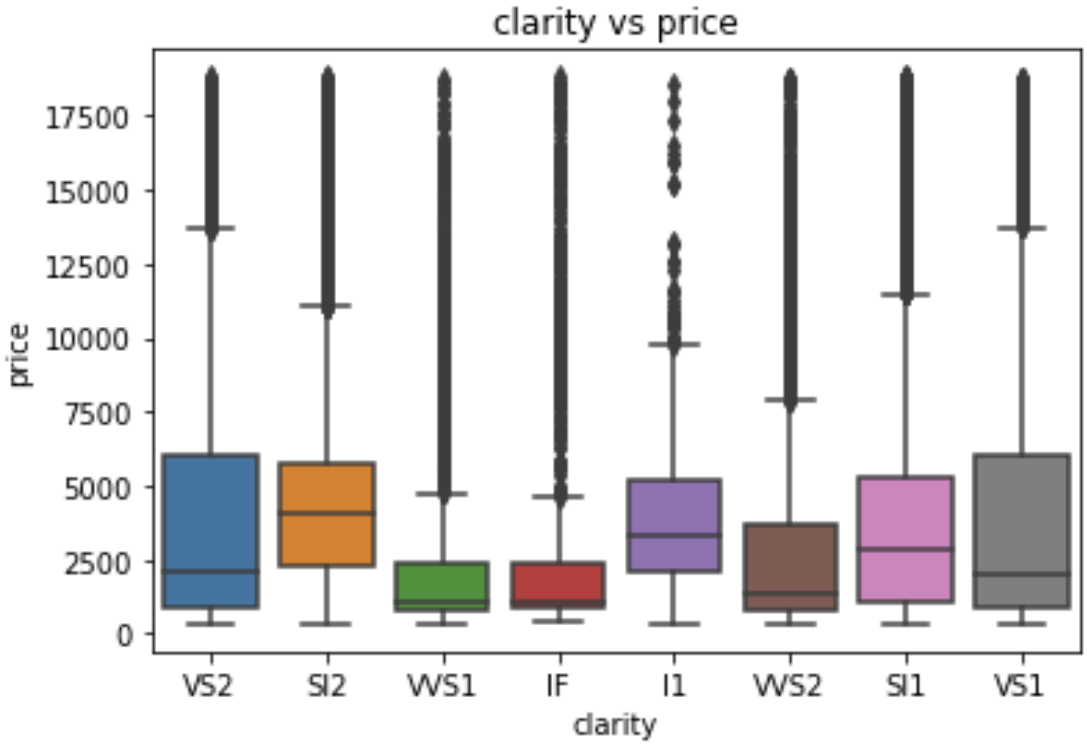


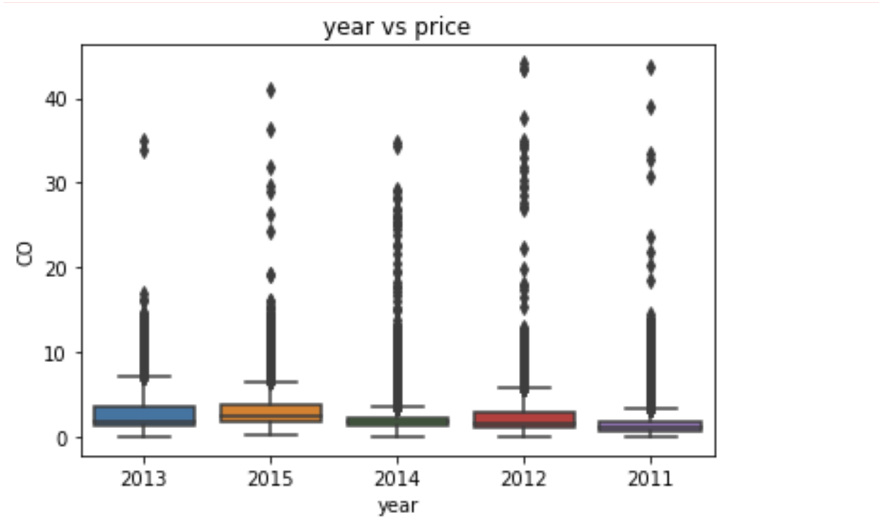
The above histograms are the gas dataset.

We should avoid high skewness of the features because it may generally cause more error if the features are not distributed evenly in regression models. We can manually remove outliers and do the scale standardization to avoid skewed distributions.

Question 4: Construct and inspect the box plot of categorical features vs target variable. What do you find?





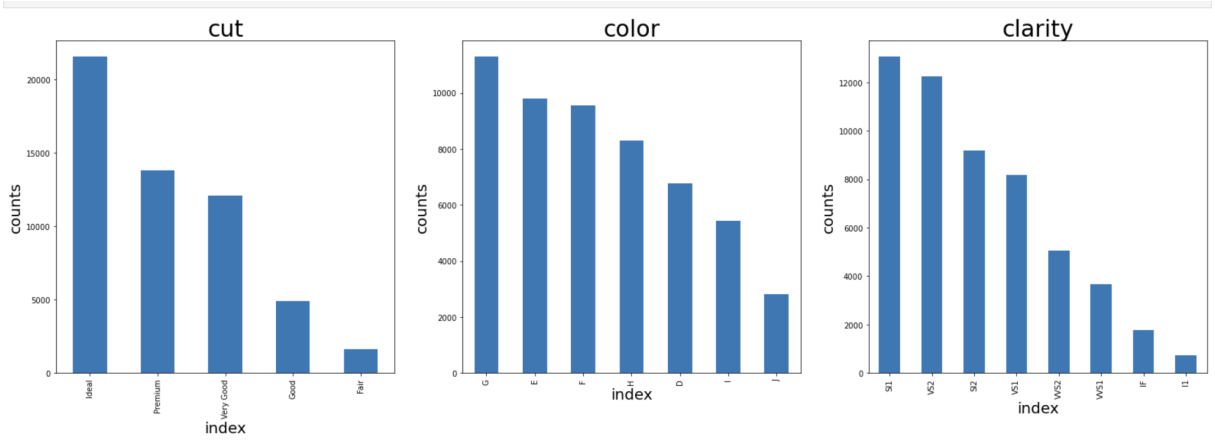


The above boxplots are for categorical features with the target label for diamond dataset and gas dataset.

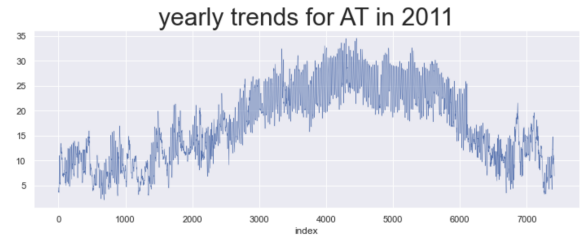
From the diamond dataset, we know that the color I has the highest price while the color E has the lowest price. The cut quality does not affect much on the price, with premium having the highest price. For clarity, WS1 and IF have the lowest effect in terms of predicting the prices, while VS2 and VS1 have the highest effects.

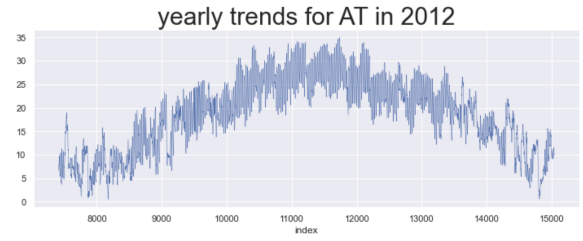
From the gas dataset, the yield of CO spreads relatively evenly in 2014, and there are relative quantities in each range. The distribution of CO yield in 2013 is relatively small, and the partial CO yield value in 2011 and 2012 can reach the highest value in 5 years.

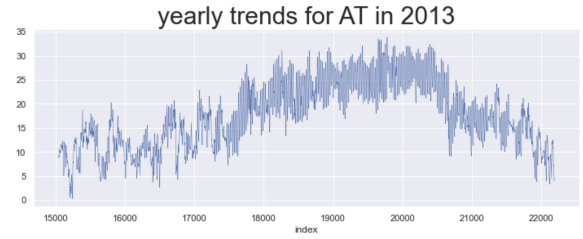
Question 5: For the Diamonds dataset, plot the counts by color, cut and clarity.

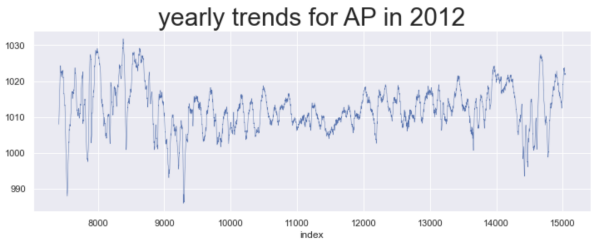


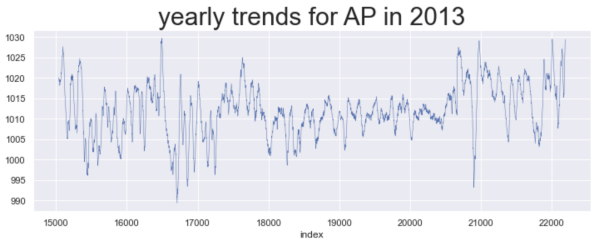
Question 6: For the Gas Emission dataset, plot the yearly trends for each feature and compare them. The data points don't have timestamps but you may assume the indeces are times.

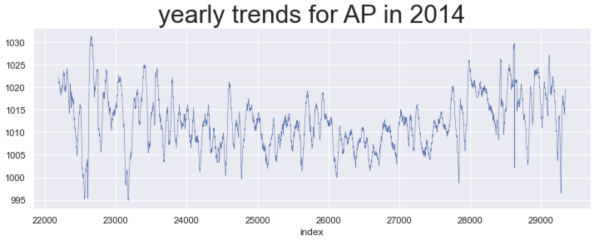


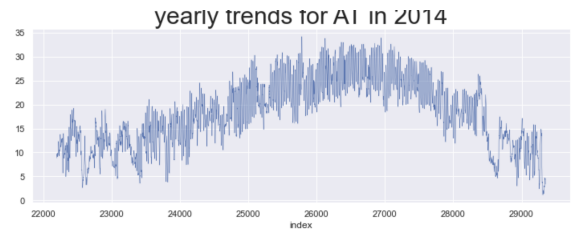


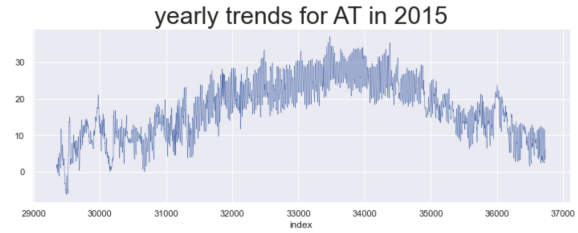


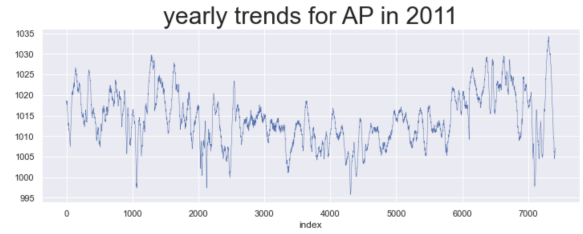


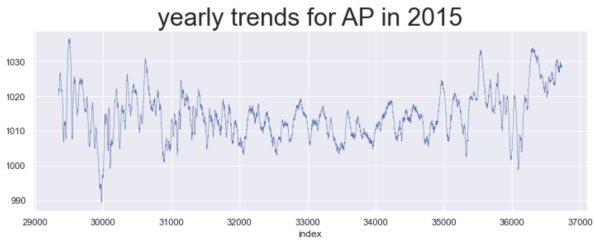


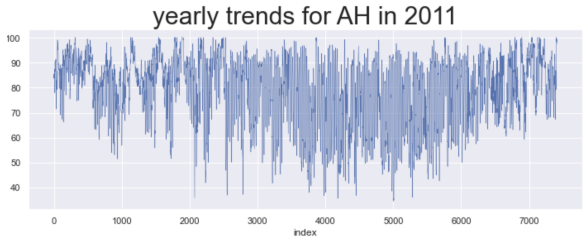


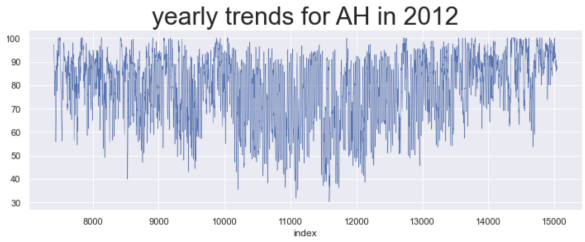


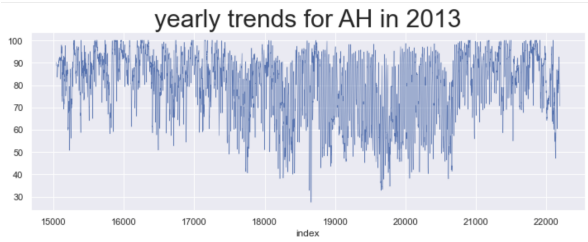


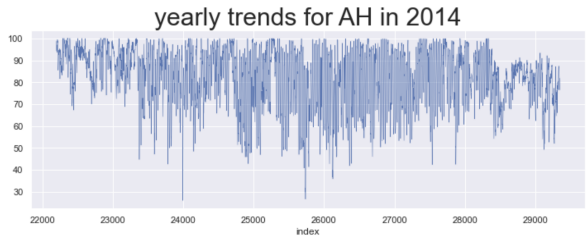


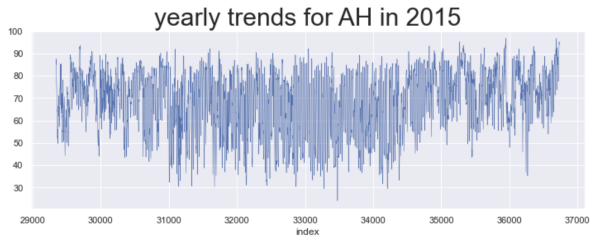


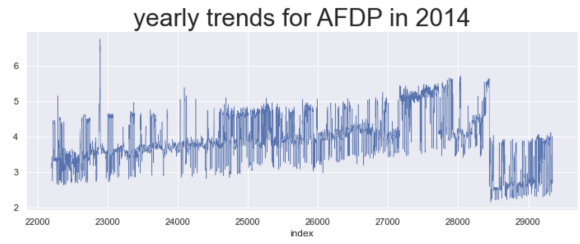


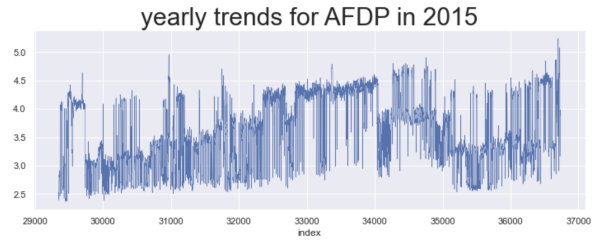


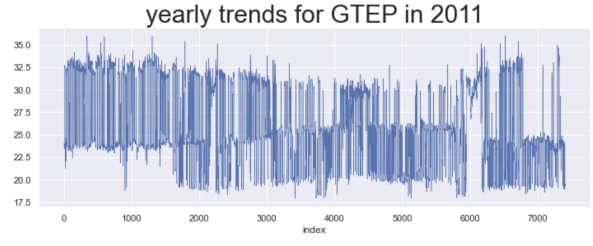

yearly trends for AH in 2013
index

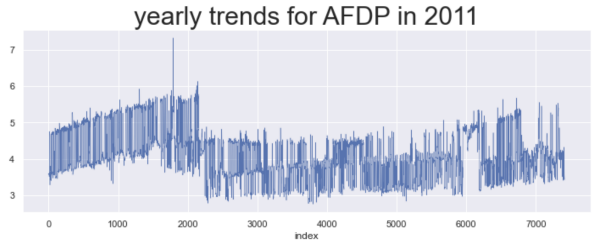

yearly trends for AH in 2014
index

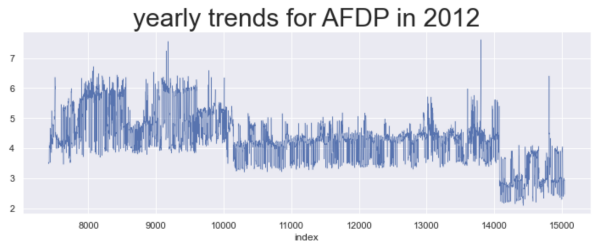

yearly trends for AH in 2015
index

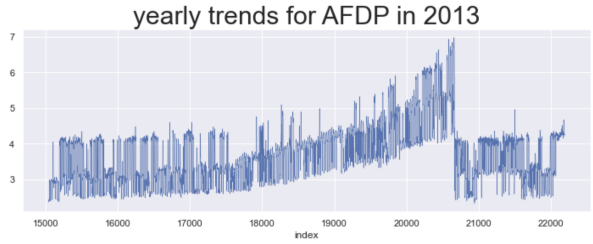

yearly trends for AFDP in 2014
index

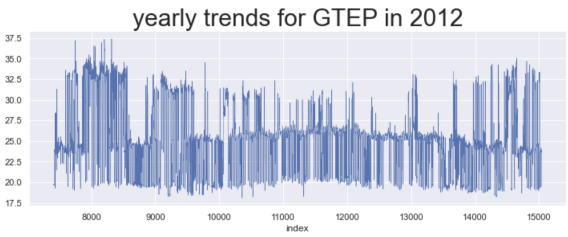

yearly trends for AFDP in 2015
index

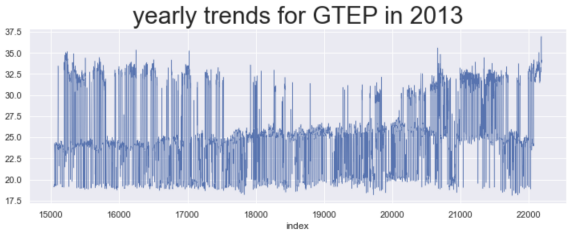

yearly trends for GTEP in 2011
index

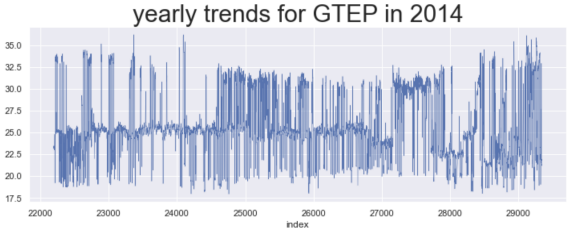

yearly trends for AFDP in 2011
index

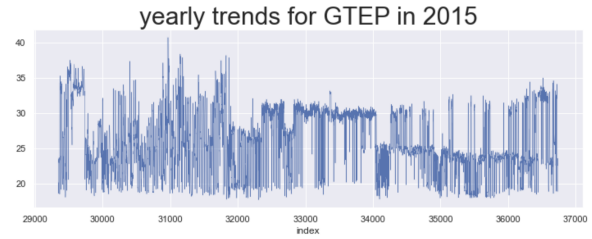

yearly trends for AFDP in 2012
index

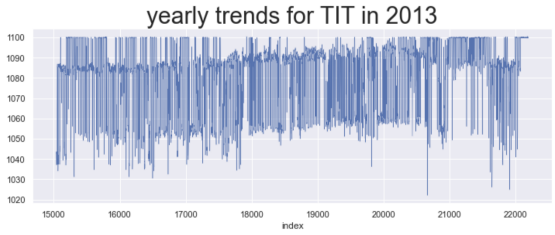

yearly trends for AFDP in 2013
index

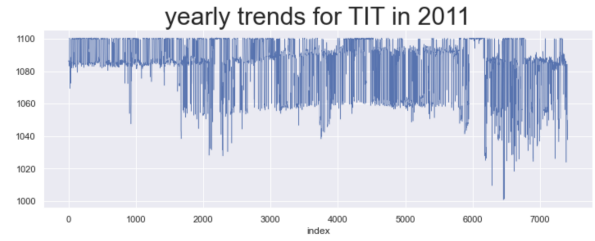

yearly trends for GTEP in 2012
index

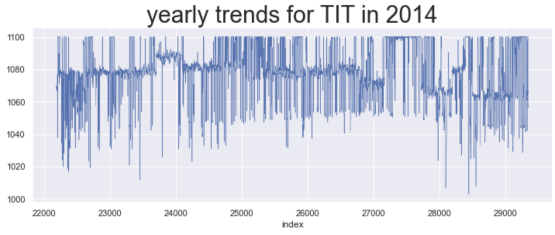

yearly trends for GTEP in 2013
index

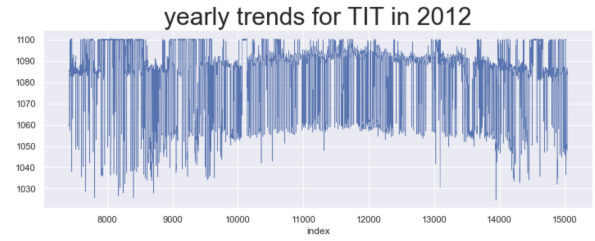

yearly trends for GTEP in 2014
index

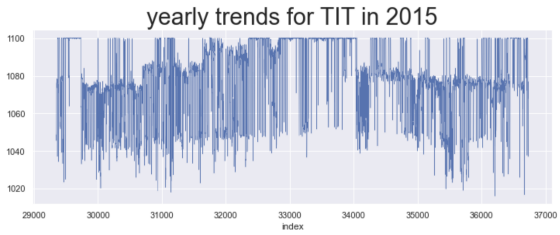

yearly trends for GTEP in 2015
index

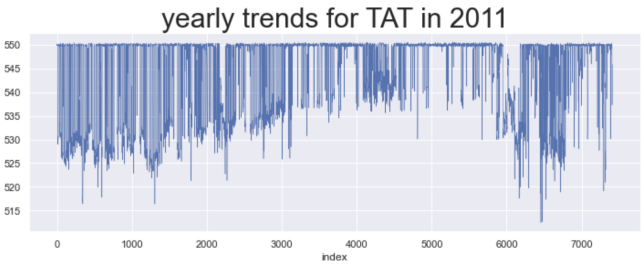

yearly trends for TIT in 2013
index

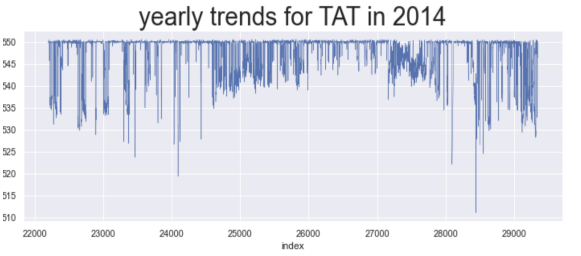

yearly trends for TIT in 2011
index

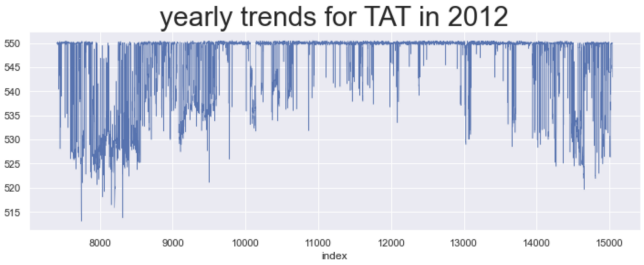

yearly trends for TIT in 2014
index

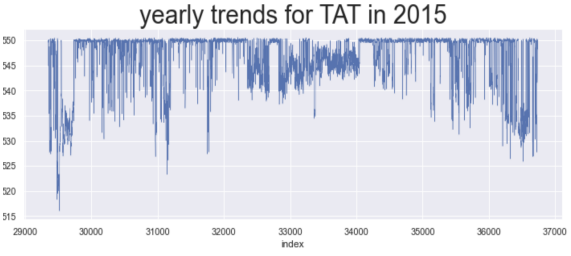

yearly trends for TIT in 2012
index

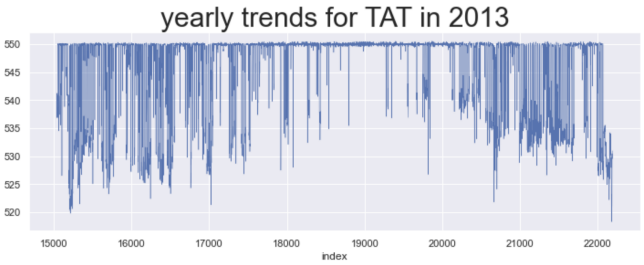

yearly trends for TIT in 2015
index

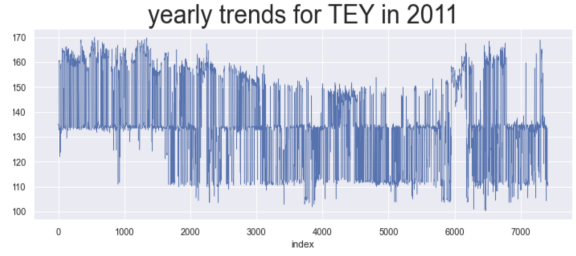

yearly trends for TAT in 2011
index

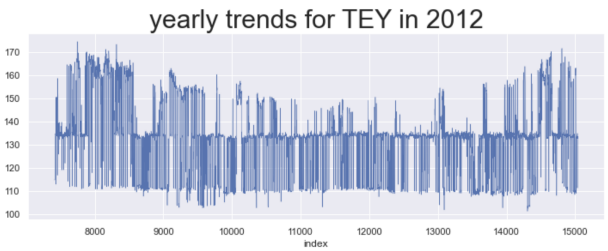

yearly trends for TAT in 2014
index

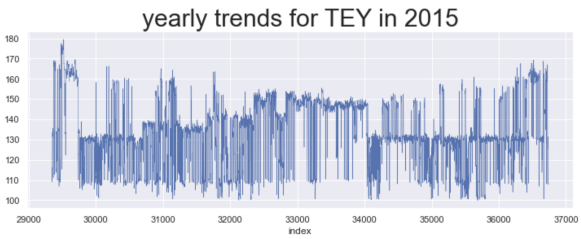

yearly trends for TAT in 2012
index

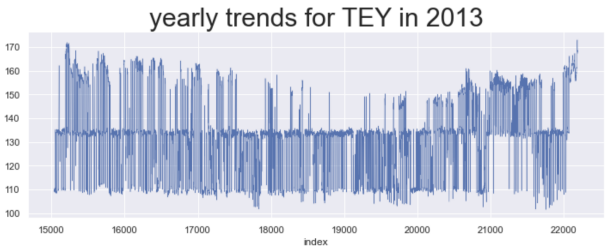

yearly trends for TAT in 2015
index

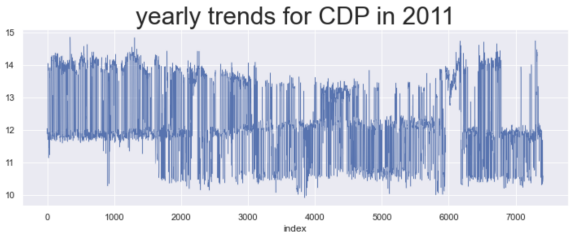

yearly trends for TAT in 2013
index

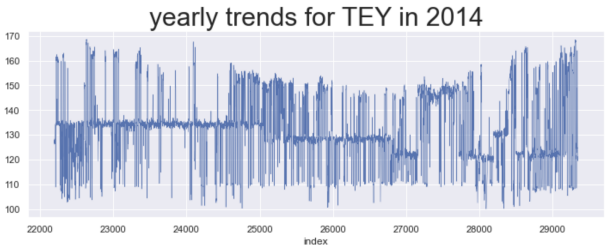

yearly trends for TEY in 2011
index

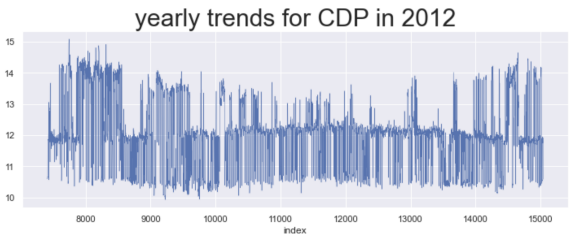

yearly trends for TEY in 2012
index

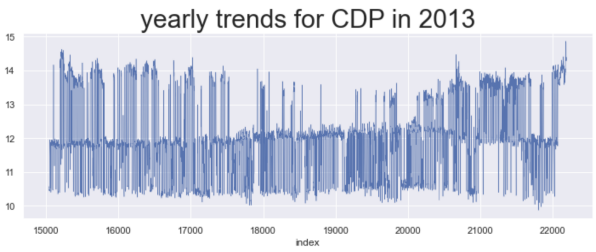

yearly trends for TEY in 2015
index

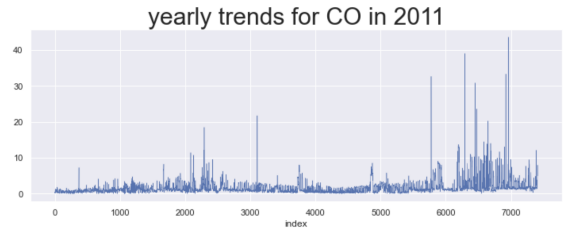

yearly trends for TEY in 2013
index

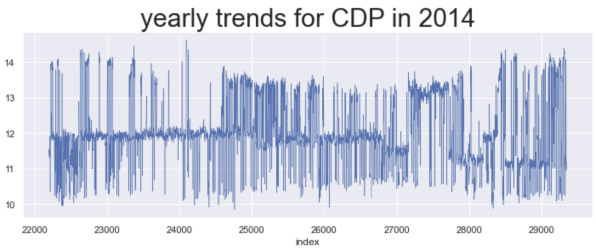

yearly trends for CDP in 2011
index

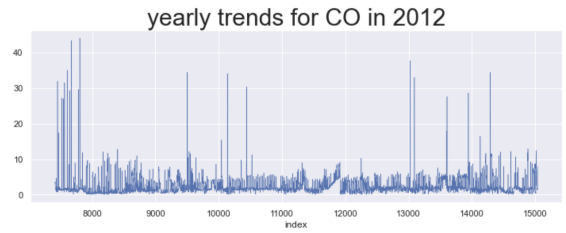

yearly trends for TEY in 2014
index

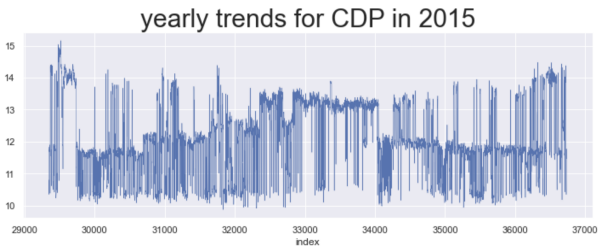

yearly trends for CDP in 2012
index

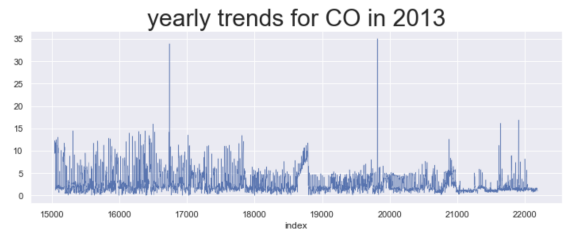

yearly trends for CDP in 2013
index

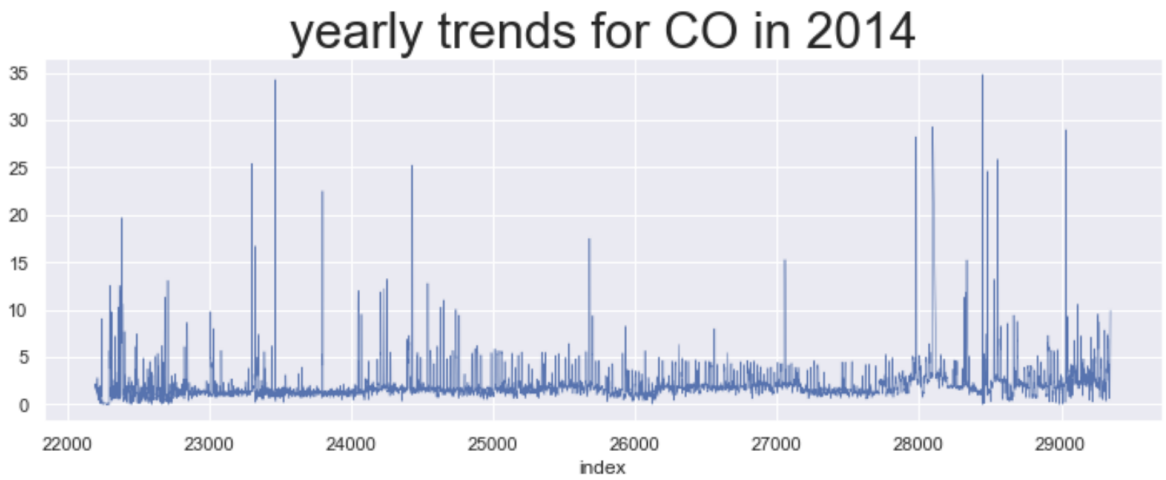

yearly trends for CO in 2011
index

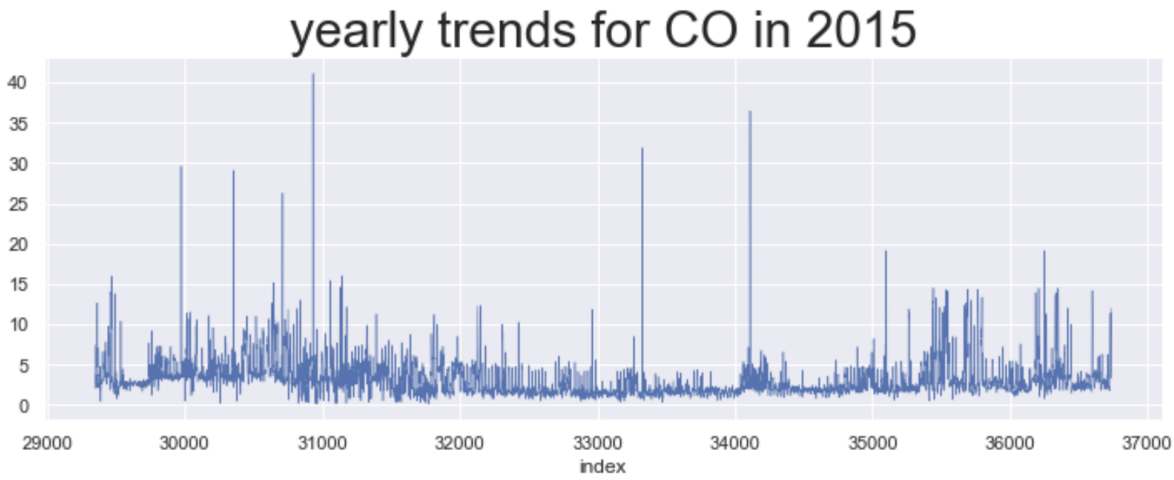

yearly trends for CDP in 2014
index


yearly trends for CO in 2012
index


yearly trends for CDP in 2015
index


yearly trends for CO in 2013
index





Question 7: How does this step affect the performance of your models in terms of test RMSE? Briefly describe your reasoning.

In [14]:
```
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import f_regression

MutualInfo_std = mutual_info_regression(diamond_X_std, diamond_y_std)
Fscore_std = f_regression(diamond_X_std, diamond_y_std)
```

In [15]:
```
diamond_X_nostd = diamond_X.to_numpy()
MutualInfo_nostd = mutual_info_regression(diamond_X_nostd, diamond_y)
Fscore_nostd = f_regression(diamond_X_nostd, diamond_y)
```

In [16]:
```
top6_MI_std = np.argsort(MutualInfo_std)[::-1][:6]
top6_FS_std = np.argsort(Fscore_std[0])[::-1][:6]
diamond_X_std_top6_MI = diamond_X.iloc[:, top6_MI_std]
diamond_X_std_top6_FS = diamond_X.iloc[:, top6_FS_std]

top6_MI_nostd = np.argsort(MutualInfo_nostd)[::-1][:6]
top6_FS_nostd = np.argsort(Fscore_nostd[0])[::-1][:6]
diamond_X_nostd_top6_MI = diamond_X.iloc[:, top6_MI_nostd]
diamond_X_nostd_top6_FS = diamond_X.iloc[:, top6_FS_nostd]
```

I am using mutual info regression and f regression for feature selection. I am picking the top 6 important features from each dataset. Therefore, I am getting rid of irrelevant or insignificant features of each dataset to perform as the functionality of reducing overfitting. Therefore, it will lower the RMSE.

The top6 selected parameters for mutual info regression is ['carat', 'y', 'x', 'z', 'clarity', 'color']

The top6 selected parameters for f regression is ['carat', 'x', 'y', 'z', 'clarity', 'table']

The top6 selected parameters for mutual info regression is ['TIT', 'TEY', 'CDP', 'GTEP', 'AFDP', 'TAT']

The top6 selected parameters for f regression is ['TIT', 'TEY', 'CDP', 'GTEP', 'AFDP', 'year']

Question 8: Explain how each regularization scheme affects the learned hypotheses.

Technique	Objective
Ordinary Least Squares	$\underset{\theta}{argmin}\ SSE$
Ridge Regression	$\underset{\theta}{argmin}\ SSE + \lambda \sum_{i=1}^{K} \theta_i^2$
Lasso Regression	$\underset{\theta}{argmin}\ SSE + \lambda \sum_{i=1}^{K} \lvert\theta_i\rvert$

For both L1 and L2 regularization, as regularization lambda increases, more weights in the learned model are set to 0, or shrink to 0. This will prevent overfitting of the training model and thus make the test accuracy higher. This is due to the fact that the Ll and L2 regularization are formulated as minimizing the weights. Thus, to prevent the regularization portion of the least squares equation from becoming too large, the optimization program will attempt to minimize the weights, ideally the weights should approach 0.

Question 9: Report your choice of the best regularization scheme along with the optimal penalty parameter and briefly explain how it can be computed.

	mean_test_score	mean_train_score	param_model	param_model__alpha	Standardize	Feature Selection
0	-0.344874	-0.358772	LinearRegression()	N/A	False	Mutual Information
1	-0.344874	-0.358772	LinearRegression()	N/A	True	Mutual Information
2	-0.344874	-0.358772	Ridge(alpha=0.001, max_iter=1000, random_state=0)	0.001	True	Mutual Information
3	-0.344875	-0.358772	Ridge(alpha=0.001, max_iter=1000, random_state=0)	0.01	True	Mutual Information
4	-0.344883	-0.358772	Ridge(alpha=0.001, max_iter=1000, random_state=0)	0.1	True	Mutual Information
5	-0.344960	-0.358772	Ridge(alpha=0.001, max_iter=1000, random_state=0)	1.0	True	Mutual Information
6	-0.345750	-0.358808	Ridge(alpha=0.001, max_iter=1000, random_state=0)	10.0	True	Mutual Information
7	-0.346096	-0.360126	LinearRegression()	N/A	False	F Scores
8	-0.346096	-0.360126	LinearRegression()	N/A	True	F Scores
9	-0.346096	-0.360126	Ridge(alpha=0.001, max_iter=1000, random_state=0)	0.001	True	F Scores
10	-0.346097	-0.360126	Ridge(alpha=0.001, max_iter=1000, random_state=0)	0.01	True	F Scores
11	-0.346104	-0.360126	Ridge(alpha=0.001, max_iter=1000, random_state=0)	0.1	True	F Scores
12	-0.346175	-0.360126	Ridge(alpha=0.001, max_iter=1000, random_state=0)	1.0	True	F Scores
13	-0.346906	-0.360161	Ridge(alpha=0.001, max_iter=1000, random_state=0)	10.0	True	F Scores

Out[23]:

	mean_test_score	mean_train_score	param_model	param_model__alpha	Standardize	Feature Selection
0	-0.686283	-0.668333	Lasso(alpha=0.01, max_iter=3000, random_state=0)	0.1	True	Mutual Information
1	-0.692270	-0.671415	Lasso(alpha=0.01, max_iter=3000, random_state=0)	0.01	True	F Scores
2	-0.692991	-0.663978	Lasso(alpha=0.01, max_iter=3000, random_state=0)	0.01	True	Mutual Information
3	-0.693904	-0.680845	Lasso(alpha=0.01, max_iter=3000, random_state=0)	0.1	True	F Scores
4	-0.694697	-0.666765	Ridge(max_iter=3000, random_state=0)	100.0	True	F Scores
5	-0.696901	-0.662292	Ridge(max_iter=3000, random_state=0)	100.0	True	Mutual Information
6	-0.698718	-0.662278	Lasso(alpha=0.01, max_iter=3000, random_state=0)	0.001	True	Mutual Information
7	-0.698806	-0.666455	Lasso(alpha=0.01, max_iter=3000, random_state=0)	0.001	True	F Scores
8	-0.700152	-0.662189	Ridge(max_iter=3000, random_state=0)	10.0	True	Mutual Information
9	-0.700675	-0.662187	Ridge(max_iter=3000, random_state=0)	1.0	True	Mutual Information
10	-0.700730	-0.662187	Ridge(max_iter=3000, random_state=0)	0.1	True	Mutual Information
11	-0.700736	-0.662187	Ridge(max_iter=3000, random_state=0)	0.01	True	Mutual Information
12	-0.700736	-0.662187	Ridge(max_iter=3000, random_state=0)	0.001	True	Mutual Information
13	-0.700737	-0.662187	LinearRegression()	N/A	True	Mutual Information
14	-0.700737	-0.662187	LinearRegression()	N/A	False	Mutual Information

The above charts are the top 20 performance models and results for both datasets. There is a tie performance on diamond dataset, while there is the only best performance on CO dataset, which is Ridge(alpha=0.01, max_iter=3000) with standardization.

To do parameter tuning for different types of models, I created a pipeline with data categorizing standardized and unstandardized data, and separated by mutual info regression or f regression feature selections. So there are a total of 4 combinations of datasets. I am also performing grid search on the model__alpha range from 10 ** [-3,3] to find the best alpha in this range. I am choosing cv= 10 here. FInally, I test the model performance on neg_root_mean_squared_error, which is the negative root mean square error.

The best RMSE for diamond is : 0.3449
The best RMSE for diamond is : 0.6862

Question 10: Does feature scaling play any role (in the cases with and without regularization)? Justify your answer.

For models without regularization, feature scaling **won't** make any difference to the model, since normalizing one feature will only incur changes to the corresponding coefficient and intercept, but won't affect the fitted value of target at all.

The below are the comparison for both datasets. The test and train scores do not change in terms of with and without standardization.

]:

	mean_test_score	mean_train_score	param_model	param_model__alpha	Standardize	Feature Selection
0	-0.344874	-0.358772	LinearRegression()	N/A	False	Mutual Information
1	-0.344874	-0.358772	LinearRegression()	N/A	True	Mutual Information
2	-0.346096	-0.360126	LinearRegression()	N/A	False	F Scores
3	-0.346096	-0.360126	LinearRegression()	N/A	True	F Scores

[21]:

	mean_test_score	mean_train_score	param_model	param_model__alpha	Standardize	Feature Selection
0	-0.700737	-0.662187	LinearRegression()	N/A	False	Mutual Information
1	-0.700737	-0.662187	LinearRegression()	N/A	True	Mutual Information
2	-0.701939	-0.666380	LinearRegression()	N/A	False	F Scores
3	-0.701939	-0.666380	LinearRegression()	N/A	True	F Scores

For models with regularization, scaling **plays a role** with significantly small effect to the result. Feature scaling improves the test RMSE when regularization is used. As you can see from the carts in question 9, the top models are mostly with standardization. Generally, when we are performing regularization, whether L1 or L2, the same penalty parameter has different effects for raw features and scaled features. Scaled features will cause less error in terms of regularization and thus produce smaller RMSE.

Question 11: Some linear regression packages return p-values for different features2. What is the meaning of them and how can you infer the most significant features?

The p-value for each feature tests the null hypothesis that the feature has no correlation with the target variable (probability of the weights of the feature being O) at the population level. A feature with a high p-value (> 0.05) indicates that there is insufficient evidence to find any meaningful correlation of that feature with the changes in the target-variable. On the other hand, a feature with a low p-value (<0.05) indicates that the probability of the coefficients of that particular feature being O is low, which indicates that the feature is well-suited as a salient feature.

```
carat        0.000000e+00
clarity      0.000000e+00
const       3.457529e-178
depth       1.359861e-177
color       1.631536e-173
x           6.459579e-154
table        4.256991e-70
cut          4.830197e-52
y            5.081300e-04
z            1.530477e-01
dtype: float64
```

```
const       0.000000e+00
TAT         0.000000e+00
year        0.000000e+00
TEY       9.011440e-212
AT        1.920190e-108
TIT        5.600418e-77
AH         1.990498e-12
AP         2.194446e-05
GTEP       4.378115e-03
AFDP       9.929232e-02
CDP        3.797984e-01
dtype: float64
```

The above values are the p-values for each parameter in each dataset. With the result showing that some p-values are small and close to 0, we will have the confidence to say that particular feature is significant in the linear model.

Question 12: Look up for the most salient features and interpret them.

For diamond dataset, salient features are carat, x,y and z, which have the highest absolute correlation with the target price. It demonstrates that carat, x,y and z are most informative in terms of predicting the price of diamond.
This is reasonable because carat, the weight of diamond, and the size of diamond are relevant to the price of the diamond more directly.

For CO dataset, salient features are year and AH, which have the highest absolute correlation with the target CO. However, the correlation values are slightly smaller overall. It demonstrates that year and AHare most informative in terms of predicting the value of CO.
AH stands for Ambient humidity. Based on our understanding, we cannot directly interpret this parameter for this task.

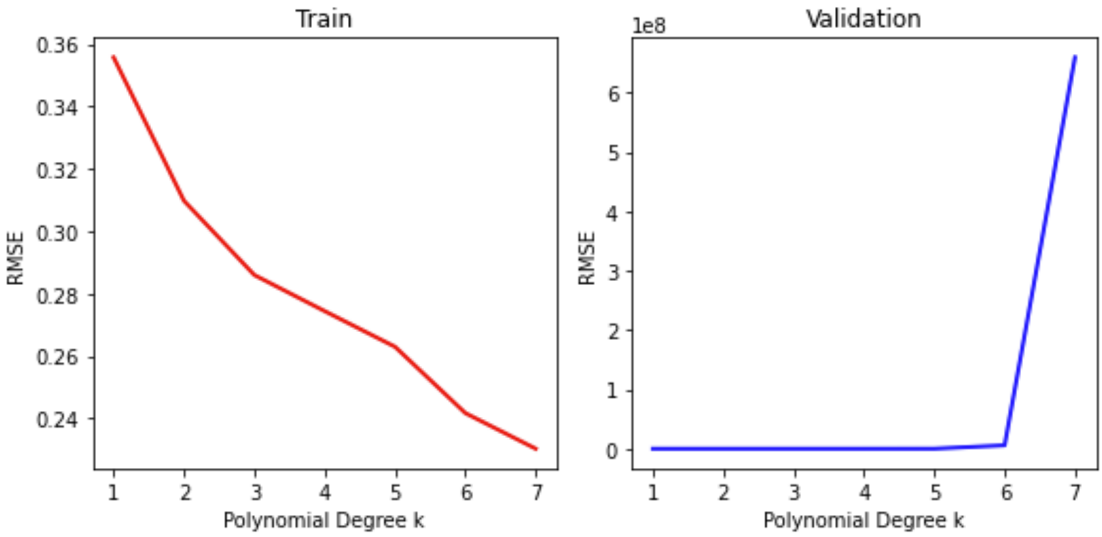
Question 13: What degree of polynomial is best? What does a very high-order polynomial imply about the fit on the training data? How do you choose this parameter?

For this question, we test the performance in terms of train and test RMSE of polynomial regression of various degrees on selected features. Since F-Score selected features performed better than MI, we selected top 6 features from F-Score. For the model, we used ridge regression to incorporate appropriate regularization term (L2) to prevent overfitting, with the penalty parameter being optimized using 7-fold grid search cross-validation.
So basically, I am using the model Ridge(alpha=0.001, random_state=0, max_iter=3000) with PolynomialFeatures(), as it's the best model from the previous part. The results for each grid search are shown below for two datasets.

The best test RMSE for diamond dataset is 0.2302 with poly degree of 7.
The best test RMSE for CO dataset is 0.4854 with poly degree of 4.

The optimal degree of polynomial we select are 3 and 4 for diamond and CO dataset, respectively. In order to avoid overfitting and make our polynomial model more efficient and interpretable at the same time, we choose k = 3 and k=4 as the optimal degree.
28]:

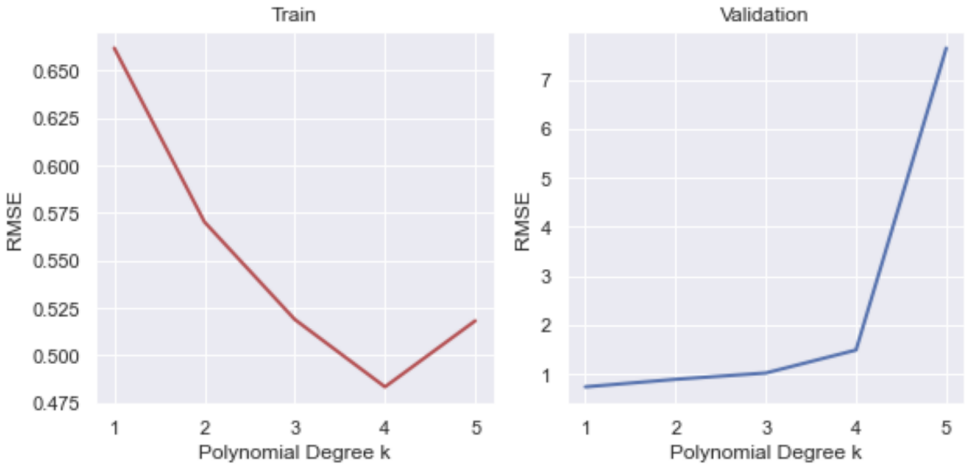
	mean_test_score	mean_train_score	param_poly_transform__degree
0	-3.699778e-01	-0.355717	1
1	-5.556096e-01	-0.309752	2
2	-3.096158e+00	-0.285952	3
3	-6.080331e+02	-0.274374	4
4	-5.357389e+04	-0.262942	5
5	-5.736250e+06	-0.241677	6
6	-6.598577e+08	-0.230263	7



[27]:

	mean_test_score	mean_train_score	param_poly_transform__degree
0	-0.700222	-0.665210	1
1	-0.686743	-0.580904	2
2	-0.660643	-0.526814	3
3	-0.700476	-0.485367	4
4	-0.994666	-0.579312	5
5	-60.451082	-52.112018	6
6	-21.647488	-12.747880	7

```
plt.show()
```



Question 14: For the diamond dataset it might make sense to craft new features such as z = x1 × x2, etc. Explain why this might make sense and check if doing so will boost accuracy.

In [32]:
```
diamond_X_cross_feature = np.prod(diamond_X[['x','y', 'z']], axis=1)
diamond_X_cross_feature = preprocessing.scale(diamond_X_cross_feature)
diamond_X_cross_concat = np.concatenate((diamond_X_standard_top6_FS, diamond_X_cross_feature.reshape(-1,1)), axis=1)
lm_cross_concat = cross_validate(Ridge(alpha=0.001, random_state=0, max_iter=3000),
                                 diamond_X_criss_concat, diamond_y_std, scoring='neg_root_mean_squared_error', cv=10)
rmse_cross_concat = np.mean(-lm_cross_concat['test_score'])
rmse_cross_concat
```

Out[32]: 0.3511888560295616

I am crossing ‘x’, ‘y’ and ‘z’ and it lowers the RMSE from around 0.36 to 0.351 using the same regressor and parameters, which is good.
It is easy to interpret since x stands of length, y stands for width and z stands for depth. So the multiplication of the three parameters get the column of the diamond, and it might show some correlation with the price

Question 15: Neural Network, Why does it do much better than linear regression?

Linear regression will do better when the underlying relationship between your variables and the response is known to be linear. However, in many cases this just doesn't hold.Neural Networks are more comprehensive and encompassing than plain linear regression, and can perform as well as Linear regressions and can do better than them when it comes to nonlinear fitting.

We can see that with linear regression, the best RMSE for diamond and CO dataset are around 0.350 and 0.700. When using the default setting of MLPRegressor, RMSE can reach as low as 0.326 and 0.642.

:

	mean_test_score	mean_train_score	param_model__hidden_layer_sizes
0	-0.326658	-0.206251	(29,)
1	-0.331476	-0.198489	(24,)
2	-0.332862	-0.203134	(21,)
3	-0.333010	-0.204021	(28,)
4	-0.334200	-0.198363	(22,)
5	-0.334625	-0.219228	(16,)
6	-0.338001	-0.208275	(12,)
7	-0.338258	-0.191707	(20,)
8	-0.338844	-0.188237	(27,)
9	-0.341802	-0.202986	(18,)

]:

	mean_test_score	mean_train_score	param_model__hidden_layer_sizes
0	-0.642203	-0.555018	(2,)
1	-0.643465	-0.495905	(12,)
2	-0.646337	-0.489360	(22,)
3	-0.648339	-0.543335	(3,)
4	-0.650583	-0.478697	(24,)
5	-0.650620	-0.486389	(23,)
6	-0.651551	-0.528140	(4,)
7	-0.653598	-0.478205	(30,)
8	-0.659801	-0.497891	(14,)
9	-0.661865	-0.522837	(6,)

Question 16-17: Adjust your network size (number of hidden neurons and depth), and weight decay as regularization. Find a good hyper-parameter set systematically. What activation function should be used for the output? You may use none.

```
time_start =time.time()

pipe_NN2 = Pipeline([
    ('model', MLPRegressor(random_state=0,max_iter=50))
])

param_grid = {
    'model__hidden_layer_sizes': [(x,) for x in np.arange(1, 31)],
    'model__alpha': [10.0**x for x in np.arange(-3,3)],
    'model__activation': ['logistic', 'tanh', 'relu']
}

grid_Q17 = GridSearchCV(pipe_NN2, param_grid=param_grid, cv=5, n_jobs=-1, verbose=False,
                                scoring='neg_root_mean_squared_error', return_train_score=True)

grid_Q17.fit(diamond_X_std, diamond_y_std)
print('\nTime to run code: {}'.format(time.time()-time_start))
```

```
time_start =time.time()

pipe_NN2 = Pipeline([
    ('model', MLPRegressor(random_state=0,max_iter=3000))
])

param_grid = {
    'model__hidden_layer_sizes': [(x,) for x in np.arange(1, 31)],
    'model__alpha': [10.0**x for x in np.arange(-3,3)],
    'model__activation': ['logistic', 'tanh', 'relu']
}

grid_Q17 = GridSearchCV(pipe_NN2, param_grid=param_grid, cv=5, n_jobs=-1, verbose=False,
                                  scoring='neg_root_mean_squared_error', return_train_score=True)

grid_Q17.fit(gas_X_std, gas_y_std)
print('\nTime to run code: {}'.format(time.time()-time_start))
```

The NN pipeline for both datasets are shown above. To avoid too much time consuming problems, I shrink the range of parameters to be tuned. As you can see the grid search cv parameter is 5 instead of 10 to save runtime.
For the pipeline, I am setting the hidden layers number from 1 to 31, range of weight decay parameter alpha with 10 ** (-3,3) and using logistics, tanh and relu as the activation layer selections.
I am also using grid search cross-validation with cv=5 for finding the best parameters, and finally return the best negative RMSE score. The results of the top 10 models are shown below.

The best RMSE for the diamond dataset is 0.318, with hidden layers of 28, alpha of 0.001 and tanh as the activation layers.
The best RMSE for the CO dataset is 0.618, with hidden layers of 1, alpha of 0.01 and tanh as the activation layers. We can see that **tanh** works generally well for these two datasets.

	mean_test_score	mean_train_score	param_model__hidden_layer_sizes	param_model__alpha	param_model__activation
0	-0.317892	-0.214216	(28,)	0.001	tanh
1	-0.318212	-0.214664	(28,)	0.01	tanh
2	-0.319766	-0.212636	(30,)	0.01	tanh
3	-0.319882	-0.212187	(30,)	0.001	tanh
4	-0.322797	-0.218046	(20,)	0.01	tanh
5	-0.323735	-0.212131	(24,)	0.001	tanh
6	-0.324001	-0.214926	(26,)	0.001	tanh
7	-0.324147	-0.213496	(24,)	0.01	tanh
8	-0.324280	-0.215375	(26,)	0.01	tanh
9	-0.326507	-0.215265	(29,)	0.1	relu

1]:

	mean_test_score	mean_train_score	param_model__hidden_layer_sizes	param_model__alpha	param_model__activation
0	-0.617994	-0.565843	(1,)	0.01	tanh
1	-0.619226	-0.565319	(1,)	0.001	tanh
2	-0.620624	-0.509685	(5,)	0.001	tanh
3	-0.621145	-0.568146	(1,)	0.1	tanh
4	-0.621323	-0.509915	(5,)	0.01	tanh
5	-0.625048	-0.503292	(13,)	0.01	logistic
6	-0.627436	-0.506502	(11,)	0.1	relu
7	-0.627503	-0.573603	(1,)	0.001	logistic
8	-0.627563	-0.496948	(12,)	0.01	logistic
9	-0.630081	-0.482953	(23,)	0.001	logistic

Question 18: What is the risk of increasing the depth of the network too far?

It will happen with overfitting problems, in situations where training error is low but validation error is high. Sometimes it will also occur with the Vanishing/exploding gradients problem, in which the gradients from the last layers will become exponentially small or large and it cannot update much information during the process of back-propagation from the final layers towards the initial layers. Moreover, it is very time-consuming to perform grid search to find an ideal hyper-parameter setting for a neural network with high depth and with many hidden layers.

Question 19: Fine-tune your model. Explain how these hyper-parameters affect the overall performance? Do some of them have regularization effect?

	mean_test_score	mean_train_score	param_model__max_features
0	-0.357264	-0.050864	0.6
1	-0.357974	-0.050313	0.7
2	-0.358487	-0.050198	0.8
3	-0.359752	-0.051906	0.5
4	-0.359950	-0.054189	0.4
5	-0.361568	-0.050220	0.9
6	-0.365354	-0.050303	1.0
7	-0.368630	-0.059295	0.3
8	-0.387314	-0.068777	0.1
9	-0.387314	-0.068777	0.2

	mean_test_score	mean_train_score	param_model__n_estimators
0	-0.356788	-0.051179	80
1	-0.356952	-0.050095	200
2	-0.357144	-0.050029	220
3	-0.357164	-0.050173	180
4	-0.357202	-0.051764	60
5	-0.357264	-0.050864	100
6	-0.357308	-0.050377	140
7	-0.357328	-0.049964	240
8	-0.357333	-0.050270	160
9	-0.357407	-0.049883	280

]:

	mean_test_score	mean_train_score	param_model__max_depth
0	-0.355723	-0.052531	22
1	-0.356682	-0.051368	25
2	-0.357235	-0.057113	19
3	-0.357261	-0.051189	28
4	-0.357432	-0.051272	30
5	-0.357823	-0.053442	21
6	-0.357961	-0.051301	29
7	-0.357989	-0.051928	23
8	-0.358147	-0.051468	24
9	-0.358153	-0.071120	16

For the diamond dataset, I am tuning the parameters by maximum number of features, number of trees and depth of tree in order. In each process, I am getting the ideal parameter with the best RMSE. The range for tuning the maximum number of features is 0-1, number of trees is 40-300 and depth of tree is 1-30. I also obtained the "Out-of-Bag Error" (OOB) for the best random forest models on diamond dataset.
The results for each process are shown above. The optimal random forest regression model after fine-tuning consists of 80 trees. The maximum number of features considered for each tree is 60% of all features, and the maximum depth of tree is 22. The best RMSE can reach 0.3557.

	mean_test_score	mean_train_score	param_model__max_features
0	-0.640253	-0.169731	0.2
1	-0.641511	-0.167622	0.3
2	-0.644678	-0.167605	0.4
3	-0.647249	-0.175309	0.1
4	-0.647917	-0.168081	0.7
5	-0.648161	-0.167657	0.5
6	-0.648998	-0.168105	0.6
7	-0.652494	-0.168731	0.8
8	-0.653785	-0.169755	0.9
9	-0.664320	-0.170601	1.0

	mean_test_score	mean_train_score	param_model__n_estimators
0	-0.630321	-0.169397	380
1	-0.630981	-0.169734	360
2	-0.631214	-0.169395	340
3	-0.631333	-0.169522	320
4	-0.631889	-0.170480	260
5	-0.631902	-0.170553	240
6	-0.632113	-0.170153	280
7	-0.632177	-0.169950	300
8	-0.632480	-0.170205	220
9	-0.633461	-0.170072	200

```
time_start =time.time()
```

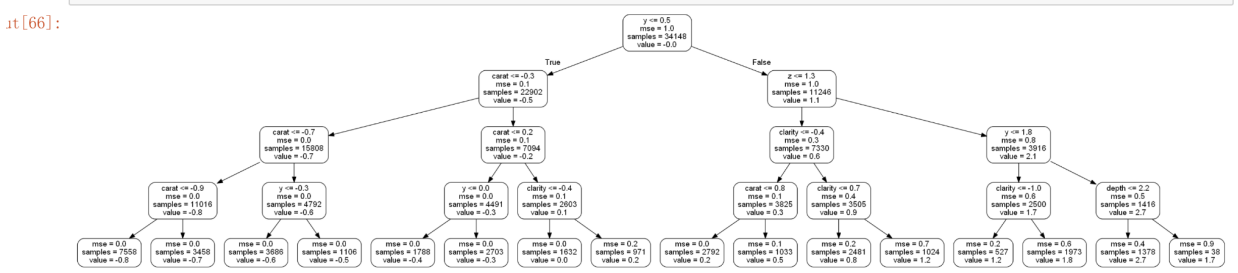
	mean_test_score	mean_train_score	param_model__max_depth
0	-0.619602	-0.379752	9
1	-0.620177	-0.352148	10
2	-0.620611	-0.409316	8
3	-0.622967	-0.441997	7
4	-0.623737	-0.326403	11
5	-0.624967	-0.476951	6
6	-0.625329	-0.282621	13
7	-0.628403	-0.303918	12
8	-0.629062	-0.263710	14
9	-0.629102	-0.247569	15

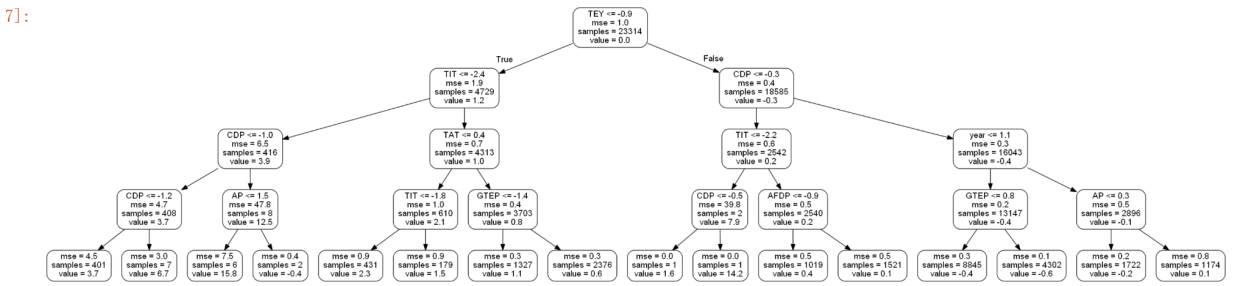
For the CO dataset, I am tuning the parameters by maximum number of features, number of trees and depth of tree in order. In each process, I am getting the ideal parameter with the best RMSE. The range for tuning the maximum number of features is 0-1, number of trees is 100-400 and depth of tree is 1-30.I also obtained the "Out-of-Bag Error" (OOB) for the best random forest models on CO dataset.
The results are shown above. The optimal random forest regression model after fine-tuning consists of 380 trees. The maximum number of features considered for each tree is 20% of all features, and the maximum depth of tree is 9. The best RMSE can reach 0.6196.

Question 20: Why does random forest perform well?

In RF, each tree along with its output target variable are both independent and identically distributed random variables as the trees are grown using a randomization technique on their individual bootstrap subsamples uncorrelated with growth of other trees. Even though one single tree may overfit, RF is a combination of many trees so that it will significantly prevent the model from overfitting.
Again, I am using cv=5 for grid-search cross validation, and the range for tuning parameters is shrunk for the purpose of saving time, so the result might not be the ideal one. We can see that the RF does not improve too much on the diamond dataset but performs well on the CO dataset. The RMSE reduces from 0.642 to 0.6196, in comparison with the best linear regression model.

Question 21: Randomly pick a tree in your random forest model (with maximum depth of 4) and plot its structure. Which feature is selected for branching at the root node? What can you infer about the importance of features? Do the important features match?





For both dataset, I am picking the parameters that result in the best RMSE. The trees are visualized shown above.

Diamond: RandomForestRegressor(max_depth=4, max_features=0.6, n_estimators=80,
oob_score=True, random_state=0)

CO: RandomForestRegressor(max_depth=4, max_features=0.2, n_estimators=380,
oob_score=True, random_state=0)

For the diamond dataset, feature branching at the root node is y. We can conclude that the most significant feature in this randomly picked tree is y. More generally, any feature that appears at the top node of the tree structure tends to be important.
In a decision tree, the features closer to the root node are more salient and significant than the features near the leaf nodes. The top features (in order) for this RF are:
Level 1: 'y'.
Level 2: 'carat' and 'z'.
Level 3: 'carat', carat', 'clarity' and 'y'

For the gas dataset, feature branching at the root node is TEY. We can conclude that the most significant feature in this randomly picked tree is TEY.
Level 1: 'TEY'.
Level 2: 'TIT' and 'CDP'.
Level 3: 'CDP', TAT', 'TIT' and 'year'

Question 22-24: Read the documentation of LightGBM and CatBoost and experiment on the picked dataset to determine the important hyperparameters along with a proper search space for the tuning of these parameters. Apply Bayesian optimization using skopt.BayesSearchCV from scikit-optmize to search good hyperparameter combinations in your search space. Report the best hyperparameter found and the corresponding RMSE, for both algorithms. Interpret the effect of the hyperparameters using the Bayesian optimization results: Which of them helps with performance? Which helps with regularization (shrinks the generalization gap)? Which affects the fitting efficiency?

I am applying on the diamond dataset.
Below is LightGBM Pipeline with fine tuning.

```
time_start =time.time()

opt = BayesSearchCV(
    lgb.LGBMRegressor(random_state=0, n_jobs=-1),
    {
        'boosting_type': ['gbdt', 'dart','rf'],
        'num_leaves': np.arange(10,3000,100),
        'max_depth': [2**i for i in range(5)],
        'n_estimators': np.arange(10,4000,100),
        'reg_alpha': [10.0**x for x in np.arange(-4,2)],
        'reg_lambda': [10.0**x for x in np.arange(-4,2)],
        'subsample': np.arange(0.1,1,0.1),
        'subsample_freq': np.arange(0,20,5),
        'min_split_gain': [10.0**x for x in np.arange(-4,0)]
    },
    n_iter=10,
    cv=5,
    n_jobs=-1,
    verbose=False,
    random_state=0,
    scoring = 'neg_root_mean_squared_error',
    return_train_score = True
)

_ = opt.fit(diamond_X_std, diamond_y_std)

print('\nTime to run code: {}'.format(time.time()-time_start))
```

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with faster training speed, higher efficiency and in use of lower memory usage.
Here are the parameters used for LightGBM.

```
'boosting_type': ['gbdt', 'dart','rf'],
'num_leaves': np.arange(10,3000,100),
'max_depth': [2**i for i in range(5)],
'n_estimators': np.arange(10,4000,100),
'reg_alpha': [10.0**x for x in np.arange(-4,2)],
'reg_lambda': [10.0**x for x in np.arange(-4,2)],
'subsample': np.arange(0.1,1,0.1),
'subsample_freq': np.arange(0,20,5),
'min_split_gain': [10.0**x for x in np.arange(-4,0)]
```

boosting_type:
‘gbdt’: traditional Gradient Boosting Decision Tree.
‘dart’: Dropouts meet Multiple Additive Regression Trees.
‘rf’: Random Forest.

GBDT is the most widely known boosted decision tree algorithm due to its accuracy, reliability and efficiency without requiring considerable effort for hyperparameter tuning. GBDT treats individual decision trees within an ensemble as weak learners, with the first tree aiming to fit the feature set to target variables and the succeeding trees aiming to reduce the residual error.
DART solves the overspecialization problem in GBDT by introducing dropout, which is used in neural networks to drop weights for regularization and improving generalization.
RF is the random forest.

num_leaves:
Maximum tree leaves for base learners.
Max_depth:
Maximum tree depth for base learners, <=0 means no limit.
N_estimators:
Number of boosted trees to fit.
reg_alpha:
L1 regularization term on weights
reg_lambda:
L2 regularization term on weights.
subsample:
Subsample ratio of the training instance.
subsample_freq:
Frequency of subsample, <=0 means no enable.
Min_split_gain:
Minimum loss reduction required to make a further partition on a leaf node of the tree.

After introducing and tuning the range of parameters, I am applying grid-search cross validation with cv=5 (in order to save computing time) and find out the best RMSE score.

Below is Chartboost Pipeline with fine tuning.

```
time_start =time.time()

optcatLG = BayesSearchCV(
    CatBoostRegressor(random_state=0,verbose=1,thread_count=-1,bootstrap_type='Bayesian'),
    {
        'colsample_bylevel': np.arange(0.1,1,0.1),
        'num_trees': np.arange(10,1000,100),
        'l2_leaf_reg': [10.0**x for x in np.arange(-4,5)],
        'num_leaves': np.arange(10,3000,100),
        'max_depth': [2**i for i in range(5)],
        'bagging_temperature': np.arange(0.1,10,1),
        'grow_policy': ['Lossguide'],

    },
    n_iter=10,
    cv=5,
    n_jobs=-1,
    verbose=1,
    random_state=0,
    scoring = 'neg_root_mean_squared_error',
    return_train_score = True
)

_ = optcatLG.fit(diamond_X_std, diamond_y_std)

print('\nTime to run code: {}'.format(time.time()-time_start))
```

CatBoost is a high-performance open source library for gradient boosting on decision trees. Here are the parameters used for Catboost.

'colsample_bylevel': np.arange(0.1,1,0.1),
'num_trees': np.arange(10,1000,100),
'l2_leaf_reg': [10.0**x for x in np.arange(-4,5)],
'num_leaves': np.arange(10,3000,100),
'max_depth': [2**i for i in range(5)],
'bagging_temperature': np.arange(0.1,10,1),
'grow_policy': ['Lossguide'],

colsample_bylevel:
Random subspace method. The percentage of features to use at each split selection, when features are selected over again at random.
Num_trees:
The maximum number of trees that can be built when solving machine learning problems.

When using other parameters that limit the number of iterations, the final number of trees may be less than the number specified in this parameter.

l2_leaf_reg:

Coefficient at the L2 regularization term of the cost function.

Num_leaves:

The maximum number of leafs in the resulting tree. Can be used only with the Lossguide growing policy.

max_depth:

Depth of the tree.

bagging_temperature:

Defines the settings of the Bayesian bootstrap. It is used by default in classification and regression modes.

grow_policy:

The tree growing policy. Defines how to perform greedy tree construction

Grow policy specifies how the trees will be generated from the leaves. Unlike LightGBM, Catboost grows a balanced tree such that at each level, the feature-split pair that minimizes the loss function is selected and utilized for all level nodes. **Lossguide** can be used as Leaf by leaf growth strategy: on each iteration, non-terminal leaf with the best loss improvement is split.

After introducing and tuning the range of parameters, I am applying grid-search cross validation with cv=5 and iteration=10 (in order to save computing time) and find out the best RMSE score.

The results for both boosters are shown below.

	mean_test_score	mean_train_score	param_boosting_type	param_num_leaves	param_max_depth	param_n_estimators	param_reg_alpha	param_reg_lambda	p
0	-0.275816	-0.101435	gbdt	210	4	3110	1.0	1.0	
1	-0.288591	-0.092936	dart	2410	8	1810	0.1	1.0	
2	-0.290269	-0.115269	dart	1310	16	3610	0.1	0.01	
3	-0.328401	-0.150652	gbdt	210	4	110	0.001	0.001	
4	-0.333888	-0.108777	gbdt	810	16	2310	0.001	1.0	
5	-0.342693	-0.189913	dart	510	2	1110	0.001	0.0001	
6	-0.361240	-0.171567	gbdt	2510	8	2510	10.0	0.001	
7	-0.416695	-0.172692	rf	1010	16	1910	0.001	1.0	
8	-0.419286	-0.246905	gbdt	710	1	710	0.01	0.1	
9	-0.428706	-0.186893	rf	710	8	710	1.0	0.01	

	mean_test_score	mean_train_score	param_colsample_bylevel	param_num_trees	param_l2_leaf_reg	param_num_leaves	param_max_depth	param_bagging_temperature
0	-0.297628	-0.079872	0.7	810	1.0	2010	8	1.1
1	-0.345533	-0.198718	0.9	710	10.0	510	2	2.1
2	-0.379675	-0.194619	0.8	310	1000.0	1610	4	1.1
3	-0.380590	-0.172944	0.6	110	10.0	2810	8	5.1
4	-0.390453	-0.227208	0.9	810	1.0	810	2	8.1
5	-0.428061	-0.262136	0.7	410	0.001	2510	2	9.1
6	-0.457969	-0.288250	0.1	610	0.01	1010	2	5.1
7	-0.458560	-0.285227	0.5	710	100.0	2210	1	3.1
8	-0.558371	-0.366991	0.5	110	0.0001	710	1	5.1
9	-0.801117	-0.773296	0.3	10	0.01	710	2	2.1

For LightGBM:
Best parameters: OrderedDict([('boosting_type', 'gbdt'), ('max_depth', 4), ('min_split_gain', 0.001), ('n_estimators', 3110), ('num_leaves', 210), ('reg_alpha', 1.0), ('reg_lambda', 1.0), ('subsample', 0.5), ('subsample_freq', 10)]) ,
Test RMSE: **-0.2758162942730318**
Train RMSE: -0.24690464927487055

For Catboost:
Best parameters: OrderedDict([('bagging_temperature', 1.1), ('colsample_bylevel', 0.7000000000000001), ('grow_policy', 'Lossguide'), ('l2_leaf_reg', 1.0), ('max_depth', 8), ('num_leaves', 2010), ('num_trees', 810)]) ,
Test RMSE: -0.2976281893895917
Train RMSE: -0.7732956858001354

Question 25: Perform 10-fold cross-validation and measure average RMSE errors for training and validation sets. Why is the training RMSE different from that of validation set?

Cross validation has been done for all the models and pipelines. Please refer the previous parts Training and testing RMSE can have huge differences as we increase the model complexity, the training RMSE will decrease, but the testing RMSE may not. This is because it may happen with an overfitting problem. In general, as training RMSE decreases monotonically, validation RMSE tends to decrease first and then increase, as the model only works well in the training set but not the validation set. The goal of the parameter tuning process is to find the optimal model structure that gives us the minimal validation/test RMSE.

Question 26: For random forest model, measure “Out-of-Bag Error” (OOB) as well. Explain what OOB error and R2 score means given this link.

OOB parameter is also set in the previous part.
For this part, we are testing on the best RF model for each dataset:

```
RandomForestRegressor(max_depth=22, max_features=0.6, n_estimators=80,
                      oob_score=True, random_state=0)

RandomForestRegressor(max_depth=9, max_features=0.2, n_estimators=380,
                      oob_score=True, random_state=0)
```

Optimal RF Regression Model for diamond dataset:
R^2 score: 0.9970,
OOB score: 0.9798.
OptimalRF Regression Model for gas dataset:
R^2 score: 0.7789,
OOB score: 0.7093.

The OOB score is used in Bootstrapping, which means that it is not using full datasets to train the model. It shares the same mathematical formula as R2 score. The Random Forest calculates it internally using only the training data. The R2 score tells you how successfully your model accounts for the intrinsic variation in the data. It is a statistical description of how the samples fit along the model. Both scores predict the generalizability of the model.