Submitted by Dr. Thyagaraju G S

Problem Statement

DESCRIPTION

Reduce the time a Mercedes-Benz spends on the test bench.

Problem Statement Scenario: Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams. To ensure the safety and reliability of every unique car configuration before they hit the road, the company's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach. You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.

Following actions should be performed:

1. If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

2. Check for null and unique values for test and train sets.

3. Apply label encoder.

4. Perform dimensionality reduction.

5. Predict your test_df values using XGBoost.

Importing Necessary Libraries and Packages

In [7]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
import time
import operator
# for dimensionality reduction
from sklearn.decomposition import PCA
```

Loading the Data

In [8]:

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

df_train = train
df_test = test
print('Size of train set: {} rows and {} columns'.format(*df_train.shape))
```

```
print('Size of test set : {} rows and {} columns'.format(*df_test.shape))
```

```
Size of train set: 4209 rows and 378 columns
Size of test set : 4209 rows and 377 columns
```

In [3]:

```
train.head()
```

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [4]:

```
train.describe()
```

Out[4]:

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.007603
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 370 columns

In [5]:

```
test.head()
```

Out[5]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

In [6]:

```
test.describe()
```

Out[6]:

	ID	X10	X11	X12	X13	X14	X15	X16	X17	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.00
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.000713	0.002613	0.008791	0.01
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.026691	0.051061	0.093357	0.10
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.00
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

8 rows × 369 columns

In [11]:

```
# Collect the Y values into an array
# seperate the y from the data as we will use this to learn as
# the prediction output
y_train = df_train['y'].values
```

In [13]:

```
y_train[0:10]
```

Out[13]:

```
array([130.81,  88.53,  76.26,  80.62,  78.02,  92.93, 128.76,  91.91,
       108.67, 126.99])
```

In [14]:

```
df_train['y'].values
```

Out[14]:

```
array([130.81,  88.53,  76.26, ..., 109.22,  87.48, 110.85])
```

1. If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

The variance will be zero if all the values are same. so we need to check if the min value in the col is equal to the max value in the col

In [15]:

```
# If any feature has the same value in each row, it is considered an unhelpful feature.
unhelpful_features = []
for feature in train:
    if max(train[feature]) == min(train[feature]):
        print(feature)
        unhelpful_features.append(feature)
```

```
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
```

In [16]:

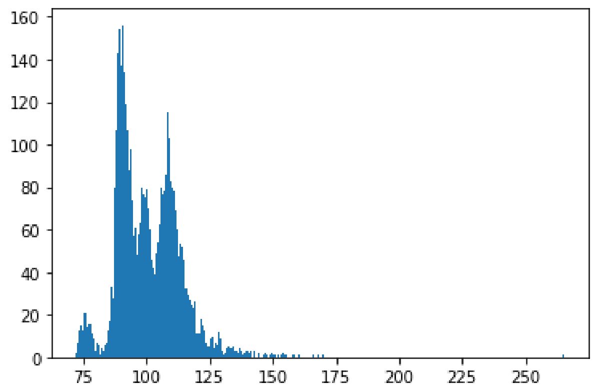
```python
# If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
#train.var(axis=0).head(10)
#train.var(axis=0) > 0

train.drop(train.var()[train.var() > 0].index.values, axis=1).head()
```

Out[16]:

	X0	X1	X2	X3	X4	X5	X6	X8	X11	X93	X107	X233	X235	X268	X289	X290	X293	X297	X330	X347
0	k	v	at	a	d	u	j	o	0	0	0	0	0	0	0	0	0	0	0	0
1	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	0	0	0	0	0
2	az	w	n	c	d	x	j	x	0	0	0	0	0	0	0	0	0	0	0	0
3	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0	0	0	0
4	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0	0	0	0

In [17]:

```python
# To detect any outliers, plot the y values.
plt.hist(train.y, bins = 300)
plt.show()
```



In [18]:

```python
print(sum(train.isnull().values, 0))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0]
```

2.Check for null and unique values for test and train sets

In [21]:

```python
# remove columns ID and Y from the data as they are not used for learning
usable_columns = list(set(df_train.columns) - set(['ID', 'y']))
y_train = df_train['y'].values
id_test = df_test['ID'].values

x_train = df_train[usable_columns]
```

```
x_test = df_test[usable_columns]
```

In [22]:

```
# Check for null and unique values for test and train sets

def check_missing_values(df):
    if df.isnull().any().any():
        print("There are missing values in the dataframe")
    else:
        print("There are no missing values in the dataframe")
check_missing_values(x_train)
check_missing_values(x_test)
```

```
There are no missing values in the dataframe
There are no missing values in the dataframe
```

3. Apply Label Encoder

In [24]:

```
## Importing the data as some data-dimensions is lost on execution of the below
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

df_train = train
df_test = test
print('Size of train set: {} rows and {} columns'.format(*df_train.shape))
print('Size of test set : {} rows and {} columns'.format(*df_test.shape))
print('Size of train set: {} rows and {} columns'.format(*train.shape))
print('Size of test set : {} rows and {} columns'.format(*test.shape))
```

```
Size of train set: 4209 rows and 378 columns
Size of test set : 4209 rows and 377 columns
Size of train set: 4209 rows and 378 columns
Size of test set : 4209 rows and 377 columns
```

In [25]:

```
# removing the outlier
train = train.loc[train['y'] < 170, :]

# seperating label and features
y_train = train['y']
train = train.drop('y', axis=1)

from sklearn.preprocessing import LabelEncoder

# label encoding the categorical features for dimension reduction
for c in train.columns:
    if train[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(train[c].values) + list(test[c].values))
        train[c] = lbl.transform(list(train[c].values))
        test[c] = lbl.transform(list(test[c].values))
```

In [26]:

```
train.head()
```

Out[26]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	37	23	20	0	3	27	9	14	0	...	0	0	1	0	0	0	0	0	0	0
1	6	37	21	22	4	3	31	11	14	0	...	1	0	0	0	0	0	0	0	0	0
2	7	24	24	38	2	3	30	9	23	0	...	0	0	0	0	0	0	1	0	0	0
3	9	24	21	38	5	3	30	11	4	0	...	0	0	0	0	0	0	0	0	0	0

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
3	9	24	21	38	5	3	30	11	4	0	...	0	0	0	0	0	0	0	0	0	0
4	13	24	23	38	5	3	14	3	13	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

In [27]:

```
test.head()
```

Out[27]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	24	23	38	5	3	26	0	22	0	...	0	0	0	1	0	0	0	0	0	0
1	2	46	3	9	0	3	9	6	24	0	...	0	0	1	0	0	0	0	0	0	0
2	3	24	23	19	5	3	0	9	9	0	...	0	0	0	1	0	0	0	0	0	0
3	4	24	13	38	5	3	32	11	13	0	...	0	0	0	1	0	0	0	0	0	0
4	5	49	20	19	2	3	31	8	12	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

4 . Perform Dimensionality Reduction

In [28]:

```
## **** Principal Component Analysis [PCA] ***
from sklearn.decomposition import PCA
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(train)
pca2_results_test = pca.transform(test)

dim_reds = list()
train_pca = pd.DataFrame()
test_pca = pd.DataFrame()

## Multiple Analysis results to list()
for i in range(1, n_comp + 1):
    train_pca['pca_' + str(i)] = pca2_results_train[:, i - 1]
    test_pca['pca_' + str(i)] = pca2_results_test[:, i - 1]
```

In [29]:

```
train_pca.head()
```

Out[29]:

	pca_1	pca_2	pca_3	pca_4	pca_5	pca_6	pca_7	pca_8	pca_9	pca_10	pca_11	pca_12
0	-4206.496934	-0.004169	-0.040700	13.251230	-4.374623	-21.241295	-2.758000	4.104086	1.642392	-0.498231	1.866237	0.592316
1	-4200.488489	-0.062182	1.778654	11.436857	-5.129672	-25.175338	-4.510954	-0.480798	0.936312	-0.624534	-0.038862	0.905201
2	-4199.490590	16.463287	13.810357	11.673968	-15.110398	-23.034331	-2.236057	1.170535	1.703934	-0.433577	-0.181609	-0.997103
3	-4197.491940	16.415288	14.793977	7.402465	3.435350	-25.485340	-4.359994	-1.889173	2.224015	0.216951	-0.638662	-1.367747
4	-4193.528315	16.833193	14.091917	10.236943	-3.105061	-8.539142	3.715471	-1.739574	2.176641	1.286413	-0.716414	-1.617840

In [30]:

```
test_pca.head()
```

Out[30]:

	pca_1	pca_2	pca_3	pca_4	pca_5	pca_6	pca_7	pca_8	pca_9	pca_10	pca_11	pca_12
0	-4205.500089	16.395861	13.630763	10.987925	-13.528819	-19.304659	6.717590	-1.621970	2.410903	1.402041	-0.525819	-1.622523
1	-4204.544020	-15.484363	-9.304810	-3.277414	-12.167708	-1.884763	0.906581	4.025849	1.994823	-0.420344	0.913304	0.310413
2	-4203.559045	12.739930	-4.123913	10.589703	3.726840	4.664174	-2.595748	-0.997856	0.487645	0.389143	0.058323	-0.936454
3	-4202.489529	14.423706	14.338755	0.316159	-6.226689	-26.207857	-3.988227	-2.536012	2.172712	-0.016570	-1.067269	-1.831161
4	-4201.489262	-12.161069	1.704583	12.975329	-2.594001	-25.353281	-1.511267	1.219026	-2.950836	-1.281744	0.214800	0.520281

5. Predict your test_df values using xgboost

In [31]:

```
print('Size of df_train set: {} rows and {} columns'.format(*df_train.shape))
print('Size of df_test set : {} rows and {} columns'.format(*df_test.shape))
print('Size of train set: {} rows and {} columns'.format(*train.shape))
print('Size of test set : {} rows and {} columns'.format(*test.shape))
```

```
Size of df_train set: 4209 rows and 378 columns
Size of df_test set : 4209 rows and 377 columns
Size of train set: 4208 rows and 377 columns
Size of test set : 4209 rows and 377 columns
```

In [32]:

```
df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")
print('Size of train set: {} rows and {} columns'.format(*df_train.shape))
print('Size of test set : {} rows and {} columns'.format(*df_test.shape))

#df_train = train
#df_test = test

usable_columns = list(set(df_train.columns) - set(['ID', 'y']))

y_train = df_train['y'].values
id_test = df_test['ID'].values

x_train = df_train[usable_columns]
x_test = df_test[usable_columns]

for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality == 1:
        x_train.drop(column, axis=1) # Column with only one value is useless so we drop it
        x_test.drop(column, axis=1)
    if cardinality > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train[column] = x_train[column].apply(mapper)
        x_test[column] = x_test[column].apply(mapper)

x_train.head()
```

```
Size of train set: 4209 rows and 378 columns
Size of test set : 4209 rows and 377 columns
```

```
C:\Users\thyagaraj\Anaconda3\lib\site-packages\ipykernel_launcher.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
C:\Users\thyagaraj\Anaconda3\lib\site-packages\ipykernel_launcher.py:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

Out[32]:

	X355	X292	X26	X295	X34	X156	X199	X133	X31	X210	...	X189	X250	X123	X124	X117	X313	X342	X168	X55	X203
0	0	0	0	0	0	1	0	0	1	0	...	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	1	0	...	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	...	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	0	...	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1	0	...	0	1	0	0	0	0	0	0	0	0

5 rows × 376 columns

In [34]:

```python
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2, random_state=4242)

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
d_test = xgb.DMatrix(x_test)

params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4

def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

clf = xgb.train(params, d_train, 1000, watchlist, early_stopping_rounds=50, feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[22:13:48] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[0] train-rmse:99.1397 valid-rmse:98.2538 train-r2:-58.3426 valid-r2:-67.6247
Multiple eval metrics have been passed: 'valid-r2' will be used for early stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10] train-rmse:81.1832 valid-rmse:80.2714 train-r2:-38.7928 valid-r2:-44.804
[20] train-rmse:66.541 valid-rmse:65.5967 train-r2:-25.7332 valid-r2:-29.5876
[30] train-rmse:54.6149 valid-rmse:53.6305 train-r2:-17.0092 valid-r2:-19.4459
[40] train-rmse:44.9172 valid-rmse:43.8842 train-r2:-11.1814 valid-r2:-12.6899
[50] train-rmse:37.0508 valid-rmse:35.9587 train-r2:-7.28831 valid-r2:-8.19158
[60] train-rmse:30.6913 valid-rmse:29.5289 train-r2:-4.68723 valid-r2:-5.19837
[70] train-rmse:25.5745 valid-rmse:24.3342 train-r2:-2.949 valid-r2:-3.20936
[80] train-rmse:21.4844 valid-rmse:20.1622 train-r2:-1.78687 valid-r2:-1.88973
[90] train-rmse:18.2427 valid-rmse:16.8438 train-r2:-1.00933 valid-r2:-1.01679
[100] train-rmse:15.7022 valid-rmse:14.2284 train-r2:-0.488656 valid-r2:-0.439108
[110] train-rmse:13.7342 valid-rmse:12.1909 train-r2:-0.138886 valid-r2:-0.056463
[120] train-rmse:12.2363 valid-rmse:10.6423 train-r2:0.095993 valid-r2:0.194898
[130] train-rmse:11.1155 valid-rmse:9.49446 train-r2:0.25401 valid-r2:0.3592
[140] train-rmse:10.2883 valid-rmse:8.67372 train-r2:0.360921 valid-r2:0.465199
[150] train-rmse:9.68714 valid-rmse:8.08748 train-r2:0.433418 valid-r2:0.535047
[160] train-rmse:9.25779 valid-rmse:7.68539 train-r2:0.482529 valid-r2:0.580131
[170] train-rmse:8.94881 valid-rmse:7.42527 train-r2:0.516494 valid-r2:0.608072
[180] train-rmse:8.7327 valid-rmse:7.25853 train-r2:0.539565 valid-r2:0.625476
[190] train-rmse:8.57747 valid-rmse:7.15432 train-r2:0.555789 valid-r2:0.636153
[200] train-rmse:8.4698 valid-rmse:7.0953 train-r2:0.566871 valid-r2:0.642132
[210] train-rmse:8.39489 valid-rmse:7.06257 train-r2:0.574498 valid-r2:0.645425
[220] train-rmse:8.33634 valid-rmse:7.04415 train-r2:0.580413 valid-r2:0.647272
```

```
[230] train-rmse:8.29296 valid-rmse:7.03716 train-r2:0.584768 valid-r2:0.647972
[240] train-rmse:8.26163 valid-rmse:7.03778 train-r2:0.5879 valid-r2:0.64791
[250] train-rmse:8.23418 valid-rmse:7.04304 train-r2:0.590634 valid-r2:0.647384
[260] train-rmse:8.21284 valid-rmse:7.04926 train-r2:0.592753 valid-r2:0.64676
[270] train-rmse:8.1958 valid-rmse:7.05766 train-r2:0.594441 valid-r2:0.645919
[280] train-rmse:8.17342 valid-rmse:7.07043 train-r2:0.596653 valid-r2:0.644636
Stopping. Best iteration:
[238] train-rmse:8.26719 valid-rmse:7.03552 train-r2:0.587345 valid-r2:0.648136
```

In [35]:

```
p_test = clf.predict(d_test)

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.head()
```

Out[35]:

	ID	y
0	1	89.715477
1	2	105.261513
2	3	90.130981
3	4	77.643867
4	5	111.152771

Submitted by Dr. Thyagaraju G S

Date : November 1st 2019

In [ ]: