Import Train and test Datasets and printing top ones. There are 4209 rows with 378 columns in train dataset. x0 : x8 are categorical data and y is the time taken in test bench(label data)

In [44]:

```
import numpy as np
import pandas as pd

dftrain = pd.read_csv('train.csv')
#dftrain.head(100)
#dftrainf=dftrain.drop(['y'], axis=1)
print(dftrain.head())
print(dftrain.shape)
```

```
   ID       y  X0 X1  X2 X3 X4 X5 X6 X8  ...  X375  X376  X377  X378  X
379  \
0   0  130.81   k  v  at  a  d  u  j  o  ...     0     0     1     0
0
1   6   88.53   k  t  av  e  d  y  l  o  ...     1     0     0     0
0
2   7   76.26  az  w   n  c  d  x  j  x  ...     0     0     0     0
0
3   9   80.62  az  t   n  f  d  x  l  e  ...     0     0     0     0
0
4  13   78.02  az  v   n  f  d  h  d  n  ...     0     0     0     0
0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 378 columns]
(4209, 378)
```

printing Info

In [45]:

```
dftrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

In [46]:

```
#getting y values alone

y_train = dftrain['y'].values
y_train
```

Out[46]:

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

In [48]:

```
# printing datatypes of columns, There are 368 columns of integer type and 8 object type

columns_x = [c for c in dftrain.columns if 'X' in c]

print (len(columns_x))
print (dftrain[columns_x].dtypes.value_counts())
```

```
376
int64     368
object      8
dtype: int64
```

In [50]:

```
#Input test dataset, Test data does not have any y values.

dftest = pd.read_csv('test.csv')
dftest.head()
```

Out[50]:

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

5 rows × 377 columns

In [60]:

```
#Printing Shape of test data set


dftest.shape
```

Out[60]:

```
(4209, 377)
```

In [61]:

```
#One way of working out on both test and train dataset is to concat both of them in
to a single dataset and then splitting it into train and test data.

df = pd.concat([dftrain,dftest])
```

In [62]:

```
#Printing df dataset

df.head()
```

Out[62]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X38
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	

5 rows × 378 columns

In [63]:

```
#getting shape of the dataframe combined

df.shape
```

Out[63]:

```
(8418, 378)
```

EDA of the dataset

In [64]:

```
#dropping of ID columns

df = df.drop(['ID'], axis = 1)
```

In [65]:

```
df.head()
```

Out[65]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X3
0	130.81	k	v	at	a	d	u	j	o	0	...	0	0	1	0	0	0	
1	88.53	k	t	av	e	d	y	l	o	0	...	1	0	0	0	0	0	
2	76.26	az	w	n	c	d	x	j	x	0	...	0	0	0	0	0	0	
3	80.62	az	t	n	f	d	x	l	e	0	...	0	0	0	0	0	0	
4	78.02	az	v	n	f	d	h	d	n	0	...	0	0	0	0	0	0	

5 rows × 377 columns

In [66]:

```
#check for Null Values and printing out no. of nulls for each columns. We see that
 Y has bunch of null values after concating with train data. Since test data did no
t had any Y




print(df.isnull().any().any())
print(df.shape)
print (df.isnull().sum())
```

```
True
(8418, 377)
y       4209
X0         0
X1         0
X2         0
X3         0
        ...
X380       0
X382       0
X383       0
X384       0
X385       0
Length: 377, dtype: int64
```

In [68]:

```
#df.describe().transpose()
#impute missing values with mean values for y and then printing null counts


df['y'].head()
df.y.fillna(df.y.mean(),inplace = True)
df.y.head()
print (df.isnull().sum())
```

```
y       0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 377, dtype: int64
```

In [70]:

```
#integer column analysis for unique values. since we know that X0 to X8 are categor
ical we are analyzing for all integer columns

unique_value_dict ={}
for col in df.columns:
    if col not in ["y","X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
        unique_value = str(np.sort(df[col].unique()).tolist())
        t_list = unique_value_dict.get(unique_value,[])
        t_list.append(col)
        unique_value_dict[unique_value]=t_list[:]
for unique_val,columns in unique_value_dict.items():
    print("columns containing the unique values: ", unique_val)
    print(columns)
    print("......................................")
```

```
columns containing the unique values:  [0, 1]
['X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
'X20', 'X21', 'X22', 'X23', 'X24', 'X26', 'X27', 'X28', 'X29', 'X30',
'X31', 'X32', 'X33', 'X34', 'X35', 'X36', 'X37', 'X38', 'X39', 'X40',
'X41', 'X42', 'X43', 'X44', 'X45', 'X46', 'X47', 'X48', 'X49', 'X50',
'X51', 'X52', 'X53', 'X54', 'X55', 'X56', 'X57', 'X58', 'X59', 'X60',
'X61', 'X62', 'X63', 'X64', 'X65', 'X66', 'X67', 'X68', 'X69', 'X70',
'X71', 'X73', 'X74', 'X75', 'X76', 'X77', 'X78', 'X79', 'X80', 'X81',
'X82', 'X83', 'X84', 'X85', 'X86', 'X87', 'X88', 'X89', 'X90', 'X91',
'X92', 'X93', 'X94', 'X95', 'X96', 'X97', 'X98', 'X99', 'X100', 'X101',
'X102', 'X103', 'X104', 'X105', 'X106', 'X107', 'X108', 'X109', 'X110',
'X111', 'X112', 'X113', 'X114', 'X115', 'X116', 'X117', 'X118', 'X119',
'X120', 'X122', 'X123', 'X124', 'X125', 'X126', 'X127', 'X128', 'X129',
'X130', 'X131', 'X132', 'X133', 'X134', 'X135', 'X136', 'X137', 'X138',
'X139', 'X140', 'X141', 'X142', 'X143', 'X144', 'X145', 'X146', 'X147',
'X148', 'X150', 'X151', 'X152', 'X153', 'X154', 'X155', 'X156', 'X157',
'X158', 'X159', 'X160', 'X161', 'X162', 'X163', 'X164', 'X165', 'X166',
'X167', 'X168', 'X169', 'X170', 'X171', 'X172', 'X173', 'X174', 'X175',
'X176', 'X177', 'X178', 'X179', 'X180', 'X181', 'X182', 'X183', 'X184',
'X185', 'X186', 'X187', 'X189', 'X190', 'X191', 'X192', 'X194', 'X195',
'X196', 'X197', 'X198', 'X199', 'X200', 'X201', 'X202', 'X203', 'X204',
'X205', 'X206', 'X207', 'X208', 'X209', 'X210', 'X211', 'X212', 'X213',
'X214', 'X215', 'X216', 'X217', 'X218', 'X219', 'X220', 'X221', 'X222',
'X223', 'X224', 'X225', 'X226', 'X227', 'X228', 'X229', 'X230', 'X231',
'X232', 'X233', 'X234', 'X235', 'X236', 'X237', 'X238', 'X239', 'X240',
'X241', 'X242', 'X243', 'X244', 'X245', 'X246', 'X247', 'X248', 'X249',
'X250', 'X251', 'X252', 'X253', 'X254', 'X255', 'X256', 'X257', 'X258',
'X259', 'X260', 'X261', 'X262', 'X263', 'X264', 'X265', 'X266', 'X267',
'X268', 'X269', 'X270', 'X271', 'X272', 'X273', 'X274', 'X275', 'X276',
'X277', 'X278', 'X279', 'X280', 'X281', 'X282', 'X283', 'X284', 'X285',
'X286', 'X287', 'X288', 'X289', 'X290', 'X291', 'X292', 'X293', 'X294',
'X295', 'X296', 'X297', 'X298', 'X299', 'X300', 'X301', 'X302', 'X304',
'X305', 'X306', 'X307', 'X308', 'X309', 'X310', 'X311', 'X312', 'X313',
'X314', 'X315', 'X316', 'X317', 'X318', 'X319', 'X320', 'X321', 'X322',
'X323', 'X324', 'X325', 'X326', 'X327', 'X328', 'X329', 'X330', 'X331',
'X332', 'X333', 'X334', 'X335', 'X336', 'X337', 'X338', 'X339', 'X340',
'X341', 'X342', 'X343', 'X344', 'X345', 'X346', 'X347', 'X348', 'X349',
'X350', 'X351', 'X352', 'X353', 'X354', 'X355', 'X356', 'X357', 'X358',
'X359', 'X360', 'X361', 'X362', 'X363', 'X364', 'X365', 'X366', 'X367',
'X368', 'X369', 'X370', 'X371', 'X372', 'X373', 'X374', 'X375', 'X376',
'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384', 'X385']
.....................................
```

In [71]:

```
##Dropping columns if there any any unique values for a column

cols=df.columns.tolist()
for col in cols:
    if len(set(df[col].tolist()))<2:
        df=df.drop(col, axis=1)
```

In [73]:

```
#Did not find any columns to drop with unique values. Getting the shape of the data
frame

df.shape
```

Out[73]:

```
(8418, 377)
```

In [75]:

```
##Label encoding of all the categorical columns from x0 to x8 and preprocessing wit
h fit/tranform

from sklearn import preprocessing
for f in ["X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
    lbl = preprocessing.LabelEncoder()
    lbl.fit(list(df[f].values))
    df[f] = lbl.transform(list(df[f].values))
```

In [76]:

```
##Pringing dataframe top rows

df.head()
```

Out[76]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X3
0	130.81	37	23	20	0	3	27	9	14	0	...	0	0	1	0	0	0	
1	88.53	37	21	22	4	3	31	11	14	0	...	1	0	0	0	0	0	
2	76.26	24	24	38	2	3	30	9	23	0	...	0	0	0	0	0	0	
3	80.62	24	21	38	5	3	30	11	4	0	...	0	0	0	0	0	0	
4	78.02	24	23	38	5	3	14	3	13	0	...	0	0	0	0	0	0	

5 rows × 377 columns

In [77]:

```
#Seperating of y values which is a label for the dataframe

X = df.drop(columns = ['y'])
Y = df['y']
print (Y.head())
print (X.head())
```

```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
   X0  X1  X2  X3  X4  X5  X6  X8  X10  X11  ...  X375  X376  X377  X37
8  \
0  37  23  20   0   3  27   9  14    0    0  ...     0     0     1
0
1  37  21  22   4   3  31  11  14    0    0  ...     1     0     0
0
2  24  24  38   2   3  30   9  23    0    0  ...     0     0     0
0
3  24  21  38   5   3  30  11   4    0    0  ...     0     0     0
0
4  24  23  38   5   3  14   3  13    0    0  ...     0     0     0
0

   X379  X380  X382  X383  X384  X385
0     0     0     0     0     0     0
1     0     0     0     0     0     0
2     0     0     1     0     0     0
3     0     0     0     0     0     0
4     0     0     0     0     0     0

[5 rows x 376 columns]
```

In [78]:

```
print(Y.head())
```

```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

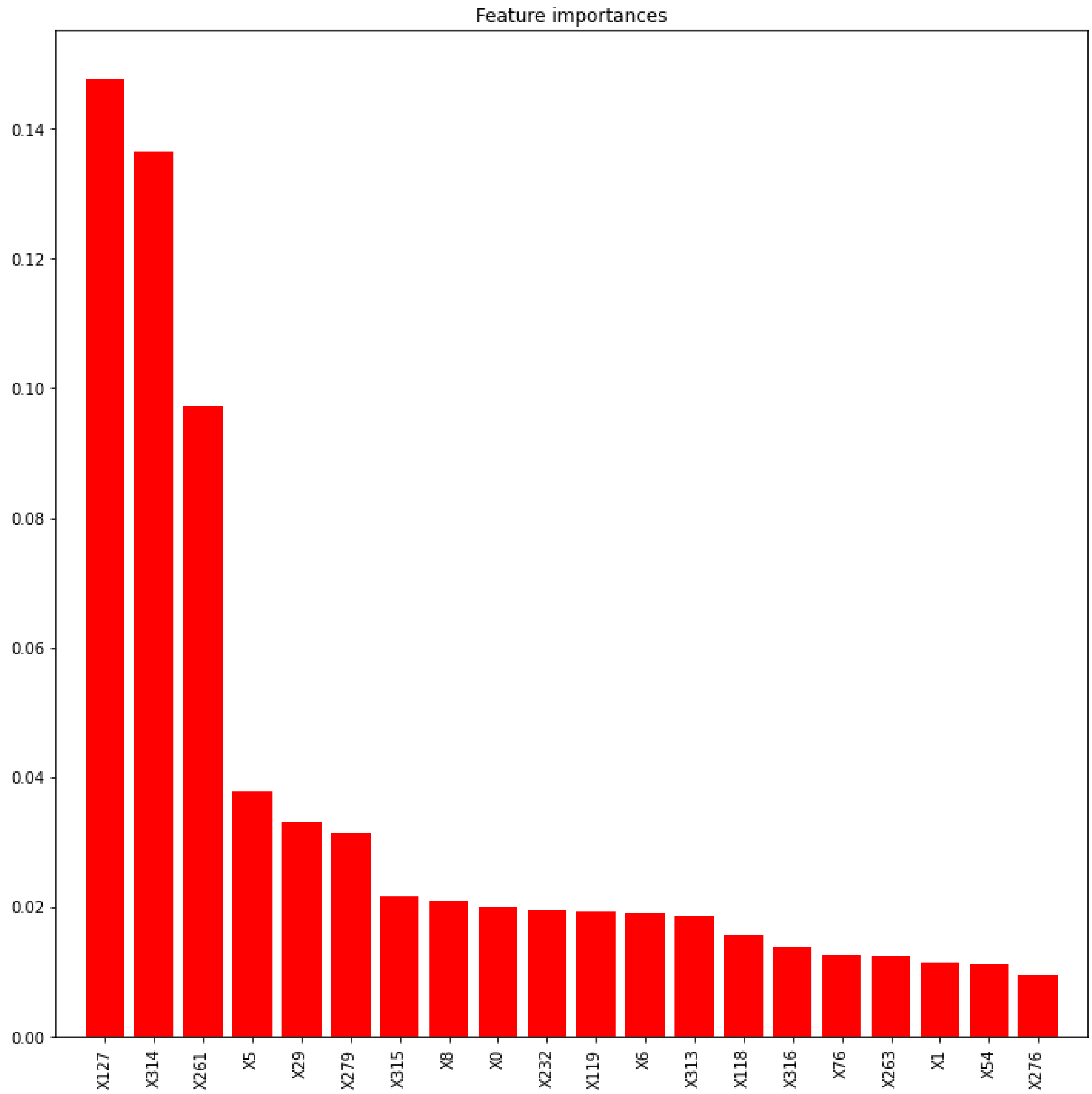
In [80]:

```
###from sklearn split data to train and test samples with 25% of randomised test sa
mple

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size = 0.25, random_stat
e = 42)

print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(6313, 376)
(6313,)
(2105, 376)
(2105,)
```

In [81]:

```
##Applying Ensemble algorithm for feature ranking and analysis


from sklearn import ensemble
model = ensemble.RandomForestRegressor(n_estimators = 200,
                                      max_depth=10, min_samples_leaf=4,
                                      max_features=0.2, n_jobs = -1,
                                      random_state=0)

model.fit(X_train,Y_train)
feat_names = X_train.columns.values

##plot the importances of all the features.

importances = model.feature_importances_


std=np.std([tree.feature_importances_ for tree in model.estimators_],axis=0)
indices=np.argsort(importances)[::-1][:20]

import matplotlib.pyplot as plt
plt.figure(figsize=(12,12))
plt.title("Feature importances")
plt.bar(range(len(indices)), importances[indices], color="r", align="center")
plt.xticks(range(len(indices)), feat_names[indices], rotation = 'vertical')
plt.xlim([-1, len(indices)])
plt.show()
```


Feature importances
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
X127
X314
X261
X5
X29
X279
X315
X8
X0
X232
X119
X6
X313
X118
X316
X76
X263
X1
X54
X276

In [82]:

```
#performing dimensionality reduction with PCA.. Consolidated dimensions to top 10

from sklearn.decomposition import PCA
n_comp = 10
pca = PCA(n_components = n_comp, random_state = 42)
pca_result_train = pca.fit_transform(X_train)
print(pca_result_train)
```

```
[[-5.00526800e+00  5.00930773e-01  7.85983299e+00 ... -1.61637176e+00
  -1.37675032e+00 -4.33249937e-01]
 [ 2.43899835e+01  1.79194876e+01 -1.26366428e+01 ...  1.74460118e+00
  -2.83426129e+00  4.83115217e+00]
 [-1.77213390e+01  2.37646428e+00  1.68032183e+00 ... -2.47428432e+00
  -8.55500500e-01  6.80067573e-01]
 ...
 [-4.74843751e+00 -4.55749153e+00 -1.55806459e+01 ...  3.25511889e-01
   5.98798381e-01  2.30510040e+00]
 [ 2.84268427e+00 -1.30766221e+01 -1.17402145e+01 ...  1.65670274e+00
  -1.78649897e+00 -3.67558178e-01]
 [-1.62506315e+01 -8.10219136e+00  5.20684109e+00 ...  1.21129905e+00
  -1.98058527e+00  9.31358495e-03]]
```

In [84]:

```
##Applying xgboost algorithm for test train split of PCA result with 80/20 split

import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train,x_valid, y_train,y_valid = train_test_split(pca_result_train,Y_train,test_s
ize = 0.2, random_state = 42)
```

In [85]:

```
df.dropna()
```

Out[85]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X
0	130.810000	37	23	20	0	3	27	9	14	0	...	0	0	1	0	0	
1	88.530000	37	21	22	4	3	31	11	14	0	...	1	0	0	0	0	
2	76.260000	24	24	38	2	3	30	9	23	0	...	0	0	0	0	0	
3	80.620000	24	21	38	5	3	30	11	4	0	...	0	0	0	0	0	
4	78.020000	24	23	38	5	3	14	3	13	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	100.669318	9	9	19	5	3	1	9	4	0	...	0	0	0	0	0	
4205	100.669318	46	1	9	3	3	1	9	24	0	...	0	1	0	0	0	
4206	100.669318	51	23	19	5	3	1	3	22	0	...	0	0	0	0	0	
4207	100.669318	10	23	19	0	3	1	2	16	0	...	0	0	1	0	0	
4208	100.669318	46	1	9	2	3	1	6	17	0	...	1	0	0	0	0	

8418 rows × 377 columns

In [86]:

```
#building final feature set with DMatrix

f_train = xgb.DMatrix(x_train, label = y_train)
f_valid = xgb.DMatrix(x_valid, label = y_valid)

#f_test = xgb.DMatrix(pca_result_test)
```

In [87]:

```
#setting params for xgb

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

In [88]:

```
#predict the score
#creating a function for the same,training for every 50 rounds

def scorer (m,w):
    labels = w.get_label()
    return 'r2', r2_score(labels, m)

final_set = [(f_train, 'train'), (f_valid, 'valid')]

P = xgb.train(params, f_train, 1000, final_set, early_stopping_rounds=50, feval=sco
rer, maximize=True, verbose_eval=10)
```

```
[01:28:12] WARNING: /workspace/src/objective/regression_obj.cu:167: re
g:linear is now deprecated in favor of reg:squarederror.
[0]	train-rmse:98.63380	valid-rmse:98.12300	train-r2:-118.4
4293	valid-r2:-131.26619
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]	train-rmse:80.75395	valid-rmse:80.24402	train-r2:-79.06
382	valid-r2:-87.45699
[20]	train-rmse:66.17490	valid-rmse:65.66124	train-r2:-52.76
442	valid-r2:-58.22772
[30]	train-rmse:54.29826	valid-rmse:53.78186	train-r2:-35.19
759	valid-r2:-38.73551
[40]	train-rmse:44.63569	valid-rmse:44.11309	train-r2:-23.46
090	valid-r2:-25.73264
[50]	train-rmse:36.78904	valid-rmse:36.25520	train-r2:-15.61
671	valid-r2:-17.05708
[60]	train-rmse:30.43295	valid-rmse:29.89357	train-r2:-10.37
092	valid-r2:-11.27616
[70]	train-rmse:25.30606	valid-rmse:24.76459	train-r2:-6.862
44	valid-r2:-7.42499
[80]	train-rmse:21.19214	valid-rmse:20.65237	train-r2:-4.513
88	valid-r2:-4.85931
[90]	train-rmse:17.91375	valid-rmse:17.37731	train-r2:-2.939
86	valid-r2:-3.14832
[100]	train-rmse:15.32852	valid-rmse:14.80437	train-r2:-1.884
75	valid-r2:-2.01083
[110]	train-rmse:13.30708	valid-rmse:12.80180	train-r2:-1.174
07	valid-r2:-1.25138
[120]	train-rmse:11.75748	valid-rmse:11.28237	train-r2:-0.697
21	valid-r2:-0.74867
[130]	train-rmse:10.58690	valid-rmse:10.15252	train-r2:-0.376
09	valid-r2:-0.41597
[140]	train-rmse:9.71275	valid-rmse:9.33356	train-r2:-0.158
22	valid-r2:-0.19675
[150]	train-rmse:9.06376	valid-rmse:8.74878	train-r2:-0.008
61	valid-r2:-0.05148
[160]	train-rmse:8.59546	valid-rmse:8.34116	train-r2:0.0929
2	valid-r2:0.04422
[170]	train-rmse:8.25674	valid-rmse:8.06883	train-r2:0.1630
0	valid-r2:0.10561
[180]	train-rmse:8.01027	valid-rmse:7.88243	train-r2:0.2122
2	valid-r2:0.14645
[190]	train-rmse:7.82932	valid-rmse:7.76039	train-r2:0.2474
1	valid-r2:0.17268
[200]	train-rmse:7.69934	valid-rmse:7.68440	train-r2:0.2721
9	valid-r2:0.18880
[210]	train-rmse:7.59400	valid-rmse:7.63432	train-r2:0.2919
7	valid-r2:0.19934
[220]	train-rmse:7.51668	valid-rmse:7.59966	train-r2:0.3063
2	valid-r2:0.20660
[230]	train-rmse:7.45579	valid-rmse:7.57918	train-r2:0.3175
1	valid-r2:0.21086
[240]	train-rmse:7.40953	valid-rmse:7.56753	train-r2:0.3259
5	valid-r2:0.21329
[250]	train-rmse:7.37368	valid-rmse:7.56115	train-r2:0.3324
```

```
6       valid-r2:0.21461
[260]   train-rmse:7.34255      valid-rmse:7.55569      train-r2:0.3380
9       valid-r2:0.21575
[270]   train-rmse:7.31035      valid-rmse:7.55072      train-r2:0.3438
8       valid-r2:0.21678
[280]   train-rmse:7.28544      valid-rmse:7.54890      train-r2:0.3483
4       valid-r2:0.21716
[290]   train-rmse:7.26522      valid-rmse:7.54605      train-r2:0.3519
5       valid-r2:0.21775
[300]   train-rmse:7.24864      valid-rmse:7.54549      train-r2:0.3549
1       valid-r2:0.21787
[310]   train-rmse:7.23163      valid-rmse:7.54479      train-r2:0.3579
3       valid-r2:0.21801
[320]   train-rmse:7.21352      valid-rmse:7.54366      train-r2:0.3611
4       valid-r2:0.21824
[330]   train-rmse:7.19800      valid-rmse:7.54392      train-r2:0.3638
9       valid-r2:0.21819
[340]   train-rmse:7.18353      valid-rmse:7.54336      train-r2:0.3664
5       valid-r2:0.21831
[350]   train-rmse:7.16932      valid-rmse:7.54330      train-r2:0.3689
5       valid-r2:0.21832
[360]   train-rmse:7.15299      valid-rmse:7.54524      train-r2:0.3718
2       valid-r2:0.21792
[370]   train-rmse:7.13840      valid-rmse:7.54582      train-r2:0.3743
8       valid-r2:0.21780
Stopping. Best iteration:
[326]   train-rmse:7.20404      valid-rmse:7.54249      train-r2:0.3628
2       valid-r2:0.21849
```

The above results indicates that r2 score has been improving for every 50 rounds and the best round is at 326 with 0.21849 and also rmse values has been imporving for evey iteration.

In [89]:

```
##predicting P algorithm on f_Valid and printing out top 5 vals

p_test = P.predict(f_valid)
p_test[:5]
```

Out[89]:

```
array([104.032364,  97.90249 , 106.1731   , 100.45344 ,  98.16179 ],
      dtype=float32)
```

In [90]:

```
#Converting to a Dataframe and printing out predictions for bench time of mercedes
 car testing line.

Predicted_Data = pd.DataFrame()
Predicted_Data['y'] = p_test
Predicted_Data.head(50)
```

Out[90]:

	y
0	104.032364
1	97.902489
2	106.173103
3	100.453438
4	98.161789
5	101.739326
6	100.072372
7	96.974182
8	105.002441
9	96.100266
10	98.048866
11	99.294830
12	104.769356
13	105.089523
14	104.151176
15	98.237175
16	91.736694
17	99.824348
18	96.261726
19	92.708221
20	98.501198
21	98.448921
22	104.934738
23	98.215034
24	109.141129
25	97.751320
26	102.907219
27	98.099167
28	98.863754
29	98.384857
30	96.962685
31	103.172012
32	101.288124

	y
33	99.453850
34	97.381752
35	100.832855
36	100.745087
37	97.898903
38	97.706001
39	94.843491
40	107.599785
41	98.990822
42	102.191399
43	101.400589
44	99.122757
45	98.913658
46	98.754494
47	87.114380
48	104.354904
49	98.609795

In [ ]:

```

```