Mercedes Benz Greener Manufacturing

Business Problem

Kaggle Competition - Mercedes Benz Greener Manufacturing

The task of this competition by Daimler is to predict the time in seconds taken to pass the test for a given set of Mercedes-Benz car features, this helps the company in speedier testing and lower CO2 Emissions.

Data Acquisition

https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data

Evaluation Metric

Submissions for the competition is evaluated on R2 score. However as the metric is directly related to mse we can directly optimize mse or rmse for that matter.

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [2]:

```
import pandas as pd
import numpy as np
import string
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.cm as cm
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from prettytable import PrettyTable
import pickle
from sklearn.model_selection import RepeatedKFold,KFold
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.feature_extraction import DictVectorizer
from xgboost import plot_importance
from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDRegressor
from scipy import stats
import random
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.svm import SVR
from sklearn.decomposition import TruncatedSVD
import keras
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_validate
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

In [0]:

```
!pip install bayesian-optimization
from bayes_opt import BayesianOptimization
```

```
Collecting bayesian-optimization
  Downloading
https://files.pythonhosted.org/packages/b5/26/9842333adbb8f17bcb3d699400a8b1ccde0af0b6de8d07224e183
cdf/bayesian_optimization-1.1.0-py3-none-any.whl
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.6/dist-packages
(from bayesian-optimization) (0.22.1)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.6/dist-packages (from
bayesian-optimization) (1.17.5)
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python3.6/dist-packages (from
bayesian-optimization) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from
scikit-learn>=0.18.0->bayesian-optimization) (0.14.1)
Installing collected packages: bayesian-optimization
Successfully installed bayesian-optimization-1.1.0
```

In [0]:

```
random_seed = 3
random.seed(random_seed)
np.random.seed(random_seed)
```

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train.csv')
results=  pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-
manufacturing/sample_submission.csv')
y_train= train.y.values
```

In [0]:

```
cat_cols_0= train.columns[train.dtypes=="object"]#categorical columns
binary_cols_0= np.delete(train.columns[train.dtypes=="int64"],0)#binary columns
num_cols_0= train.columns[train.dtypes=="int64"]#numerical columns
```

Data Exploration

In [0]:

```
train.describe()
```

Out[0]:

	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

ID	y	X10	X11	X12	X13	X14	X15	X16	X17

8 rows × 370 columns

In [0]:

```
train.columns
```

Out[0]:

```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

In [0]:

```
dtype_df = train.dtypes.reset_index()
dtype_df.columns = ["Count", "Column Type"]
dtype_df.groupby("Column Type").aggregate('count').reset_index()
```

Out[0]:

	Column Type	Count
0	int64	369
1	float64	1
2	object	8

369 integer colums, 8 categorical columns

In [0]:

```
for col,k in zip(train.columns.tolist(),train.dtypes.tolist()):
  if k =='O':
    print(col +'  :  ',len(train[col].value_counts())) #cardinalities of various categorical variables
```

```
X0  :   47
X1  :   27
X2  :   44
X3  :   7
X4  :   4
X5  :   29
X6  :   12
X8  :   25
```

Checking For Missing Values

In [0]:

```
train.isnull().sum().sum() #No missing values
```

Out[0]:

```
0
```

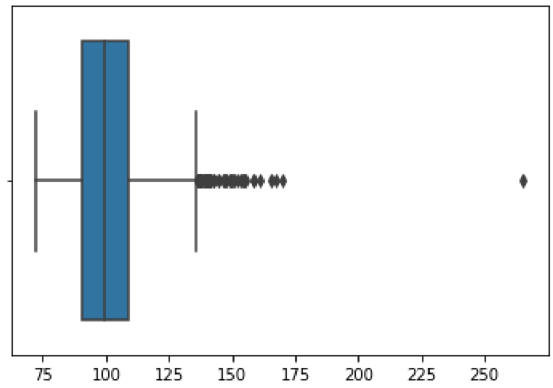
Checking for Outliers

In [0]:

```
sns.boxplot(y_train) ##presence of outliers clearly visible
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1cb3fec438>
```



Lot of Outliers atleast 20-30 of them can be seen here.

In [0]:

```
y_train.mean(),np.median(y_train)#mean being influenced by outliers
```

Out[0]:

```
(100.66931812782134, 99.15)
```

In [0]:

```
#https://stackoverflow.com/questions/22354094/pythonic-way-of-detecting-outliers-in-one-dimensiona
l-observation-data
def reject_outliers(points, thresh = 3.5):
    """detects outliers based on modified z_score computed using median"""
    if len(points.shape) == 1:
        points = points[:,None]
    median = np.median(points, axis=0)
    diff = np.sum((points - median)**2, axis=-1)
    diff = np.sqrt(diff)
    med_abs_deviation = np.median(diff)

    modified_z_score = 0.6745 * diff / med_abs_deviation

    return points[modified_z_score > thresh]
```

In [0]:

```
outliers= reject_outliers(y_train)#so everything >=146.3 can be considered an outlier
print(outliers,min(outliers))
```

```
[[146.83]
 [150.43]
 [169.91]
 [154.87]
 [147.72]
 [265.32]
 [158.53]
 [154.43]
 [149.63]
 [160.87]
 [150.89]
 [152.32]
 [167.45]
 [154.16]
 [148.94]
 [158.23]
 [153.51]
 [147.22]
 [146.3 ]
 [165.52]
 [155.62]
 [149.52]] [146.3]
```

Checking for Duplicate Rows with different Targets

In [0]:

```
full= train
full['y']= y_train
```

In [0]:

```
duplicateRowsDF = full[full[[x for x in full.columns.tolist() if x != 'ID' and x!= 'y']].duplicated
(keep= False)]
duplicateRowsDF.shape
```

Out[0]:

```
(515, 378)
```

There are identical rows with different target variable values(515 rows)

Checking for Duplicate Columns

In [0]:

```
train_train = train.T.drop_duplicates().T
train_train.shape,train.shape
```

Out[0]:

```
((4209, 322), (4209, 378))
```

56 columns which are constant throughout the trainset.

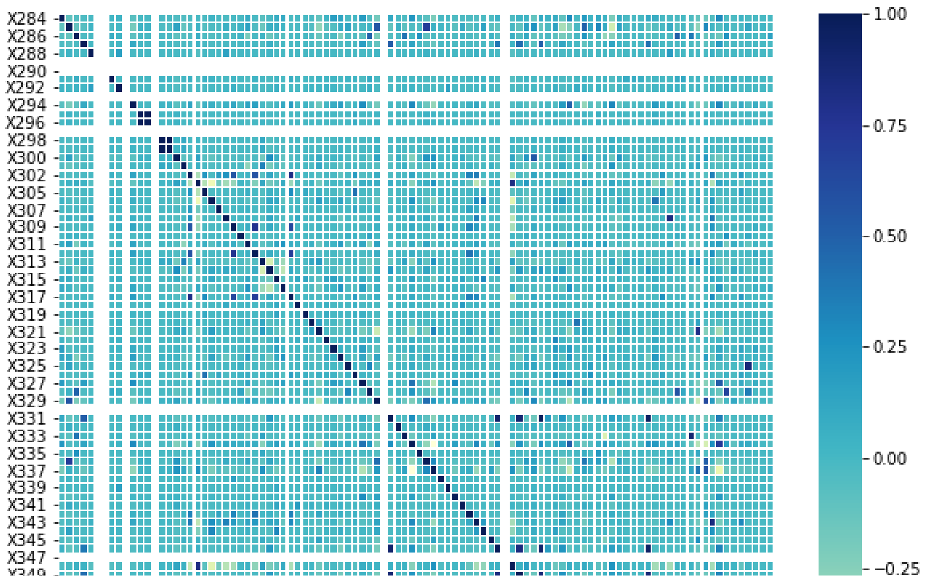
Correlation Graph

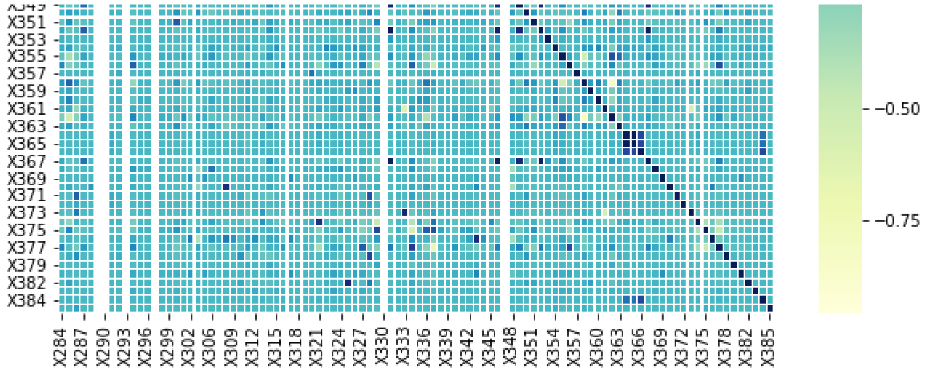
In [0]:

```
#https://www.geeksforgeeks.org/exploring-correlation-in-python/

corrmat = train.iloc[:,-100:].corr()#taking last 100 features

f, ax = plt.subplots(figsize =(10, 10))
sns.heatmap(corrmat, ax = ax, cmap ="YlGnBu", linewidths = 0.1) #There are variables with 1 correla
tion coefficient
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fad3abb8438>
```





This shows some features having high correlation with other features.

In [0]:

```
cat_cols= train.columns[train.dtypes=="object"]#categorical columns
binary_cols= np.delete(train.columns[train.dtypes=="int64"],0,-1)#binary columns
num_cols= train.columns[train.dtypes=="int64"]#numerical columns
```

Visualisation using PCA

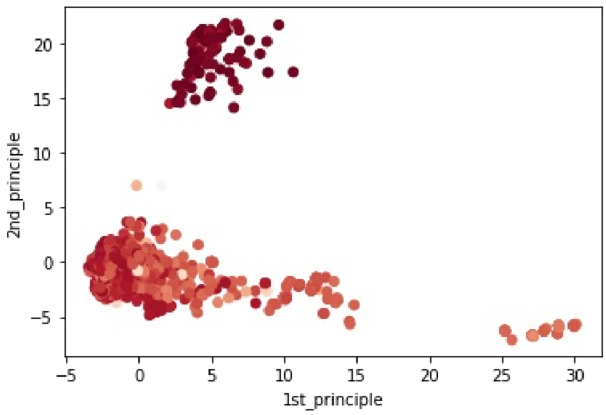
In [0]:

```
standardized_data = StandardScaler().fit_transform(train[binary_cols])
print(standardized_data.shape)
pca = decomposition.PCA()
pca.n_components = 2
pca_data = pca.fit_transform(standardized_data)
print("shape of pca_reduced.shape = ", pca_data.shape)
pca_data = np.vstack((pca_data.T, y_train)).T
```

```
(4209, 368)
shape of pca_reduced.shape =  (4209, 2)
```

In [0]:

```
pca_df = pd.DataFrame(data=pca_data, columns=("1st", "2nd", "label"))
plt.scatter(pca_df['1st'],pca_df['2nd'], c=pca_df['label'], cmap="RdBu")
plt.xlabel('1st_principle')
plt.ylabel('2nd_principle')
plt.show()
#lets check if this observation really helps
```



PCA did find some good features lets try to check the mean and std of these clusters

In [0]:

```
#This can be reason in kaggle competition many kernels used pca with 6 components
#cluster1 mean,cluster2 mean,cluster3 mean
pca_df['label'][pca_df['2nd']>14].mean(),pca_df['label'][(pca_df['2nd']<14) & (pca_df['1st']<15)].m
```

```
pca_df['label'][pca_df['2nd']>14].mean(),pca_df['label'][(pca_df['2nd']<14) & (pca_df['1st']<15)].m
ean(),pca_df['label'][(pca_df['2nd']<14) & (pca_df['1st']>15)].mean()
```

Out[0]:

```
(77.96486187845298, 101.32426029486524, 116.97744680851063)
```

In [0]:

```
#cluster1 std,cluster2 std,cluster3 std
pca_df['label'][pca_df['2nd']>14].std(),pca_df['label'][(pca_df['2nd']<14) & (pca_df['1st']<15)].st
d(),pca_df['label'][(pca_df['2nd']<14) & (pca_df['1st']>15)].std()
```

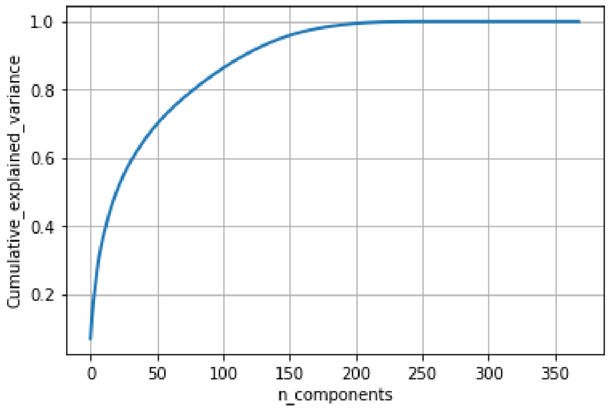
Out[0]:

```
(4.616846880686572, 11.80421638156366, 7.075507495179731)
```

Cluster 1 is very well seperated, but Cluster 2 which is fairly large is 1std away from Cluster 3.This surely can help the model but coming up with alternative ways is necessary to perform better.

In [0]:

```
pca.n_components = 369
pca_data = pca.fit_transform(standardized_data)

percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_);

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the PCA spectrum
plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



Around 95% of variance is explained with about 150 components which is half the features given.

Here these things make sense from the above observations.

1. PCA provides some useful features so including it in model will be useful.
2. Duplicate rows with different target values can be replaced with mean or median.
3. Outliers must be clipped off from the y_train, this may give overoptimistic results but atleast we can be sure that our CV and Private Leaderboard will be positively correlated.

A look at best feature Correlations on Raw Data

In [0]:

```
dic={}
for i in num_cols:
    if train[i].corr(train.y)>0.50 or train[i].corr(train.y)<-0.50:
      dic[i]=train[i].corr(train.y)
print("Important Features with there respective correlations are ",'\n','--------------------------
-------------------------------','\n',dic)
```

```
Important Features with there respective correlations are
 ---------------------------------------------------------
 {'X127': -0.5106197590551649, 'X261': 0.5887851610438137, 'X314': 0.6060052136703652}
```

Removal of Features which are uninformative

In [0]:

```
num_cols= num_cols_0
```

In [0]:

```
#Cleaning  up columns less than threshold variance.
rem_cols=[]
temp = []
for i in num_cols:
    if train[i].var()<=0.01:
        temp.append(i)
print(len(temp))
print(temp,'<0.01 variance columns')
rem_cols.extend(temp)
```

```
147
['X11', 'X15', 'X16', 'X17', 'X18', 'X21', 'X24', 'X26', 'X30', 'X33', 'X34', 'X36', 'X39', 'X40',
'X42', 'X53', 'X55', 'X59', 'X60', 'X62', 'X65', 'X67', 'X74', 'X78', 'X83', 'X86', 'X87', 'X88',
'X89', 'X90', 'X91', 'X92', 'X93', 'X94', 'X95', 'X97', 'X99', 'X102', 'X104', 'X105', 'X107', 'X1
10', 'X112', 'X122', 'X123', 'X124', 'X125', 'X145', 'X153', 'X160', 'X165', 'X167', 'X169',
'X172', 'X173', 'X183', 'X184', 'X190', 'X192', 'X199', 'X200', 'X204', 'X205', 'X207', 'X210',
'X212', 'X213', 'X214', 'X216', 'X217', 'X221', 'X227', 'X230', 'X233', 'X235', 'X236', 'X237',
'X239', 'X240', 'X242', 'X243', 'X245', 'X248', 'X249', 'X252', 'X253', 'X254', 'X257', 'X258',
'X259', 'X260', 'X262', 'X266', 'X267', 'X268', 'X269', 'X270', 'X271', 'X274', 'X277', 'X278',
'X280', 'X281', 'X282', 'X288', 'X289', 'X290', 'X292', 'X293', 'X295', 'X296', 'X297', 'X298',
'X299', 'X307', 'X308', 'X309', 'X310', 'X312', 'X317', 'X318', 'X319', 'X320', 'X323', 'X325',
'X330', 'X332', 'X335', 'X338', 'X339', 'X341', 'X344', 'X347', 'X353', 'X357', 'X364', 'X365',
'X366', 'X369', 'X370', 'X372', 'X379', 'X380', 'X382', 'X383', 'X384', 'X385'] <0.01 variance
columns
```

In [0]:

```
#removing duplicate columns and leaving the original behind.
dups=list(train.T.index[train.T.duplicated(keep= 'first')].values)
print(dups)
rem_cols.extend(dups)
```

```
['X35', 'X37', 'X39', 'X76', 'X84', 'X93', 'X94', 'X102', 'X107', 'X113', 'X119', 'X122', 'X134',
'X146', 'X147', 'X172', 'X199', 'X213', 'X214', 'X216', 'X222', 'X226', 'X227', 'X232', 'X233',
'X235', 'X239', 'X242', 'X243', 'X244', 'X245', 'X247', 'X248', 'X253', 'X254', 'X262', 'X266',
'X268', 'X279', 'X289', 'X290', 'X293', 'X296', 'X297', 'X299', 'X302', 'X320', 'X324', 'X326',
'X330', 'X347', 'X360', 'X364', 'X365', 'X382', 'X385']
```

In [0]:

```
#X4 Found to have really low variance
train.X4.value_counts()
```

Out[0]:

```
d    4205
a       2
b       1
c       1
Name: X4, dtype: int64
```

In [0]:

```
#################Removal of Uninformative Features Done Here
rem_cols= list(set(rem_cols))
rem_cols.append('X4')#only cat_col to be dropped
train= train.drop(rem_cols,axis=1)
test= test.drop(rem_cols,axis=1)
target= y_train
train.shape,test.shape
```

Out[0]:

```
((4209, 211), (4209, 210))
```

Dataset Making

LabelEncoded Dataset

In [0]:

```
#cats are label encoded here
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))


train.y= np.clip(train.y.values,0,150) #clipping of y occurs here
test.to_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_label.csv',index= Fa
lse)
train.to_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_label.csv',index=
False)
```

In [0]:

```
train.head()
```

Out[0]:

	ID	y	X0	X1	X2	X3	X5	X6	X8	X10	X12	X13	X14	X19	X20	X22	X23	X27	X28	X29	X31	X32	X38	X41	X43	X4
0	0	130.81	37	23	20	0	27	9	14	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	(
1	6	88.53	37	21	22	4	31	11	14	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	(
2	7	76.26	24	24	38	2	30	9	23	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	(
3	9	80.62	24	21	38	5	30	11	4	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	(
4	13	78.02	24	23	38	5	14	3	13	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	(

5 rows × 211 columns

Mean Encoded Dataset

In [0]:

```
##cats are mean encoded here
y_mean= train.y.mean()
for col in cat_cols:
   y=train.groupby([col]).mean()['y']
   train[col]= [y.loc[a] for a in train[col]]
   test[col]=[(y.loc[a] if a in y.index else y_mean)for a in test[col]]
train.y= np.clip(train.y.values,0,150)
test.to_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_meanenc.csv',index=
False)
train.to_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_meanenc.csv',index
= False)
```

Correlation of Categorical variables with Target

In [0]:

```
for k in cat_cols:
  print(train[k].corr(train.y),'--->',k)
```

```
0.7782581671040522 ---> X0
0.21043301114505705 ---> X1
0.4906867003496653 ---> X2
0.21598114984670597 ---> X3
0.11900439734404482 ---> X5
0.10918048655236655 ---> X6
0.17187044046225866 ---> X8
```

Feature X0 is very well correlated to the target.

OneHotEncoding Dataset

In [0]:

```
##one_hot cats are created here.
from sklearn.preprocessing import OneHotEncoder
CC= OneHotEncoder(handle_unknown='ignore',sparse= False)
train_hot= pd.DataFrame(CC.fit_transform(train[cat_cols]),columns= ['hot_'+str(x) for x in range(19
1)])
test_hot= pd.DataFrame(CC.transform(test[cat_cols]),columns= ['hot_'+str(x) for x in range(191)])

train=train.join(train_hot)
test=test.join(test_hot)
test.drop(cat_cols,axis=1,inplace= True)
train.drop(cat_cols,axis= 1,inplace= True)
print(train.shape)
train.y= np.clip(train.y.values,0,150)
test.to_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_hot.csv',index= Fals
e)
train.to_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_hot.csv',index= Fa
lse)
```

```
(4209, 395)
```

Model Preperation

CV was chosen to be RepeatedKFold with 5 folds with 3 repetitions.

In [0]:

```
cv= RepeatedKFold(n_splits= 5,n_repeats=3,random_state= random_seed)
```

Baseline

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_label.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_label.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
cat_cols= [i for i in cat_cols_0 if i in train.columns] ##categorical columns
binary_cols= [i for i in binary_cols_0 if i in train.columns] #binary columns
num_cols= [i for i in num_cols_0 if i in train.columns ]#numerical columns
```

In [0]:

```
#https://www.kaggle.com/hakeem/stacked-then-averaged-models-0-5697
def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

# Add decomposed components: PCA / ICA etc.
n_comp = 12
ids_test = test.ID.values
# PCA
pca = PCA(n_components=n_comp, random_state=random_seed)

SS=StandardScaler()
pca_train= SS.fit_transform(train[num_cols])
pca_test= SS.transform(test[num_cols])
pca2_results_train = pca.fit_transform(pca_train)
pca2_results_test = pca.transform(pca_test)

# Append decomposition components to datasets
for i in range(1, n_comp+1):
    train['pca_' + str(i)] = pca2_results_train[:, i-1]
    test['pca_' + str(i)] = pca2_results_test[:, i-1]

# Prepare data
X = np.array(train)
y = y_train
y_mean = np.mean(y)

X_test = np.array(test)

print('X.shape = ' + str(X.shape) + ', y.shape = ' + str(y.shape))
print('X_test.shape = ' + str(X.shape))

params = {}
params['n_trees'] = 500
params['objective'] = 'reg:linear'
params['eta'] = 0.005
params['max_depth'] = 3
params['subsample'] = 0.95
params['base_score'] = y_mean
params['silent'] = 1
params['n_thread']= -1
#params['colsample_bytree']= .9
xgb_r2_seal1 = []
test_preds_buf = []
d_test = xgb.DMatrix(X_test)
```

```
X.shape = (4209, 222), y.shape = (4209,)
X_test.shape = (4209, 222)
```

In [0]:

```
fold_i = 0
for train_index, test_index in cv.split(X):
    print('Fold #' + str(fold_i))
    x_train, x_valid, y_train, y_valid = X[train_index], X[test_index], y[train_index], y[test_inde
x]

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

    print('XGB: Evaluating model')
    eval_set = [(x_train, y_train), (x_valid, y_valid)]
    watchlist = [(d_train, 'train'), (d_valid, 'valid')]

    model = xgb.train(params, d_train, 1000, watchlist, early_stopping_rounds=50, \
        feval=xgb_r2_score, maximize=True, verbose_eval=100)

    p = model.predict(d_valid)
    r2 = r2_score(y_valid, p)
    xgb_r2_seal1.append(r2)
    print('R2 = ' + str(r2))
```

```python
    test_preds_buf.append(model.predict(d_test))

    fold_i += 1

print('XGB Mean R2 = ' + str(np.mean(xgb_r2_seal1)) + ' +/- ' + str(np.std(xgb_r2_seal1)))

print('XGB: Train on full dataset and predicting on test')
d_train = xgb.DMatrix(X, label=y)
watchlist = [(d_train, 'train')]
model = xgb.train(params, d_train, 700, watchlist, feval=xgb_r2_score, \
    maximize=True, verbose_eval=100)

p_test = model.predict(d_test)

test_preds_buf = np.array(test_preds_buf).T
test_preds_buf = np.concatenate((test_preds_buf, p_test.reshape((len(p_test),1))), axis=1)

subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = np.mean(test_preds_buf, axis=1)
subm.to_csv('xgb_pca12_15fold_16mdls.csv', index=False)
```

```
Fold #0
XGB: Evaluating model
[0] train-rmse:12.3509 valid-rmse:12.0628 train-r2:0.005999 valid-r2:0.005397
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.
[100] train-rmse:9.68992 valid-rmse:9.5898 train-r2:0.388175 valid-r2:0.371406
[200] train-rmse:8.48776 valid-rmse:8.52492 train-r2:0.530568 valid-r2:0.503257
[300] train-rmse:7.98084 valid-rmse:8.11952 train-r2:0.584966 valid-r2:0.549378
[400] train-rmse:7.75116 valid-rmse:7.97747 train-r2:0.608511 valid-r2:0.565007
[500] train-rmse:7.62515 valid-rmse:7.93269 train-r2:0.621137 valid-r2:0.569877
[600] train-rmse:7.54012 valid-rmse:7.91998 train-r2:0.629539 valid-r2:0.571254
Stopping. Best iteration:
[640] train-rmse:7.51233 valid-rmse:7.91823 train-r2:0.632264 valid-r2:0.571444

R2 = 0.5710785808510108
Fold #1
XGB: Evaluating model
[0] train-rmse:12.3073 valid-rmse:12.2392 train-r2:0.005966 valid-r2:0.005873
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.
[100] train-rmse:9.69424 valid-rmse:9.60363 train-r2:0.38326 valid-r2:0.387926
[200] train-rmse:8.51408 valid-rmse:8.4484 train-r2:0.524281 valid-r2:0.526323
[300] train-rmse:8.01881 valid-rmse:7.99773 train-r2:0.578017 valid-r2:0.57551
[400] train-rmse:7.79705 valid-rmse:7.82869 train-r2:0.601035 valid-r2:0.593265
[500] train-rmse:7.67615 valid-rmse:7.76956 train-r2:0.613311 valid-r2:0.599386
[600] train-rmse:7.59854 valid-rmse:7.74955 train-r2:0.621091 valid-r2:0.601447
[700] train-rmse:7.53456 valid-rmse:7.74688 train-r2:0.627445 valid-r2:0.601721
Stopping. Best iteration:
[707] train-rmse:7.53074 valid-rmse:7.74641 train-r2:0.627822 valid-r2:0.60177

R2 = 0.60156437389248
Fold #2
XGB: Evaluating model
[0] train-rmse:12.2127 valid-rmse:12.6139 train-r2:0.005983 valid-r2:0.005311
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.
[100] train-rmse:9.60346 valid-rmse:10.037 train-r2:0.385351 valid-r2:0.3702
[200] train-rmse:8.42594 valid-rmse:8.89891 train-r2:0.526841 valid-r2:0.504932
[300] train-rmse:7.93222 valid-rmse:8.4309 train-r2:0.580665 valid-r2:0.555635
[400] train-rmse:7.7126 valid-rmse:8.24176 train-r2:0.603564 valid-r2:0.57535
[500] train-rmse:7.58856 valid-rmse:8.16573 train-r2:0.616212 valid-r2:0.583149
[600] train-rmse:7.50704 valid-rmse:8.12359 train-r2:0.624415 valid-r2:0.587439
[700] train-rmse:7.44268 valid-rmse:8.10936 train-r2:0.630826 valid-r2:0.588884
[800] train-rmse:7.38532 valid-rmse:8.10844 train-r2:0.636495 valid-r2:0.588977
Stopping. Best iteration:
[782] train-rmse:7.39516 valid-rmse:8.1068 train-r2:0.635525 valid-r2:0.589143

R2 = 0.5889381996129521
Fold #3
XGB: Evaluating model
[0] train-rmse:12.314 valid-rmse:12.2127 train-r2:0.005889 valid-r2:0.005462
```

```
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.
[100] train-rmse:9.723 valid-rmse:9.50695 train-r2:0.380223 valid-r2:0.397325
[200] train-rmse:8.55888 valid-rmse:8.30096 train-r2:0.519748 valid-r2:0.54053
[300] train-rmse:8.07478 valid-rmse:7.81603 train-r2:0.57254 valid-r2:0.592645
[400] train-rmse:7.86293 valid-rmse:7.62187 train-r2:0.594675 valid-r2:0.612632
[500] train-rmse:7.74741 valid-rmse:7.54259 train-r2:0.606497 valid-r2:0.620648
[600] train-rmse:7.66734 valid-rmse:7.51069 train-r2:0.614589 valid-r2:0.623851
[700] train-rmse:7.59777 valid-rmse:7.49025 train-r2:0.621551 valid-r2:0.625895
[800] train-rmse:7.53905 valid-rmse:7.48007 train-r2:0.627378 valid-r2:0.626912
[900] train-rmse:7.4866 valid-rmse:7.47389 train-r2:0.632545 valid-r2:0.627528
Stopping. Best iteration:
[901] train-rmse:7.48581 valid-rmse:7.47377 train-r2:0.632622 valid-r2:0.62754

R2 = 0.6274182456170972
Fold #4
XGB: Evaluating model
[0] train-rmse:12.284 valid-rmse:12.3349 train-r2:0.005911 valid-r2:0.00485
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.
[100] train-rmse:9.67905 valid-rmse:9.78124 train-r2:0.382816 valid-r2:0.374248
[200] train-rmse:8.50276 valid-rmse:8.61995 train-r2:0.523712 valid-r2:0.514014
[300] train-rmse:8.01258 valid-rmse:8.12963 train-r2:0.577045 valid-r2:0.567729
[400] train-rmse:7.80333 valid-rmse:7.91004 train-r2:0.598847 valid-r2:0.590766
[500] train-rmse:7.68583 valid-rmse:7.8167 train-r2:0.610837 valid-r2:0.600368
[600] train-rmse:7.60048 valid-rmse:7.77492 train-r2:0.619432 valid-r2:0.604628
[700] train-rmse:7.53637 valid-rmse:7.75947 train-r2:0.625826 valid-r2:0.606197
[800] train-rmse:7.48076 valid-rmse:7.74586 train-r2:0.631328 valid-r2:0.607578
[900] train-rmse:7.42941 valid-rmse:7.74113 train-r2:0.636371 valid-r2:0.608057
[999] train-rmse:7.38267 valid-rmse:7.7371 train-r2:0.640932 valid-r2:0.608465
R2 = 0.6084651675649255
Fold #5
XGB: Evaluating model
[0] train-rmse:12.2175 valid-rmse:12.5966 train-r2:0.005968 valid-r2:0.0057
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.
[100] train-rmse:9.63302 valid-rmse:10.0184 train-r2:0.382043 valid-r2:0.371059
[200] train-rmse:8.46928 valid-rmse:8.83221 train-r2:0.522332 valid-r2:0.511181
[300] train-rmse:7.97968 valid-rmse:8.32724 train-r2:0.575963 valid-r2:0.565478
[400] train-rmse:7.7614 valid-rmse:8.11402 train-r2:0.598844 valid-r2:0.587446
[500] train-rmse:7.64645 valid-rmse:8.01351 train-r2:0.610639 valid-r2:0.597603
[600] train-rmse:7.56863 valid-rmse:7.96312 train-r2:0.618523 valid-r2:0.602648
[700] train-rmse:7.50538 valid-rmse:7.9432 train-r2:0.624872 valid-r2:0.604634
[800] train-rmse:7.45066 valid-rmse:7.93235 train-r2:0.630322 valid-r2:0.605713
[900] train-rmse:7.4022 valid-rmse:7.92521 train-r2:0.635116 valid-r2:0.606422
[999] train-rmse:7.35583 valid-rmse:7.92241 train-r2:0.639673 valid-r2:0.606701
R2 = 0.6067007775198987
Fold #6
XGB: Evaluating model
[0] train-rmse:12.3082 valid-rmse:12.2362 train-r2:0.005945 valid-r2:0.004884
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.
[100] train-rmse:9.6676 valid-rmse:9.68474 train-r2:0.386721 valid-r2:0.37662
[200] train-rmse:8.47336 valid-rmse:8.5823 train-r2:0.52888 valid-r2:0.510464
[300] train-rmse:7.97133 valid-rmse:8.15665 train-r2:0.583052 valid-r2:0.557818
[400] train-rmse:7.74944 valid-rmse:7.99553 train-r2:0.605941 valid-r2:0.575115
[500] train-rmse:7.62934 valid-rmse:7.94433 train-r2:0.618061 valid-r2:0.580538
[600] train-rmse:7.53911 valid-rmse:7.93471 train-r2:0.627042 valid-r2:0.581554
Stopping. Best iteration:
[634] train-rmse:7.5097 valid-rmse:7.93378 train-r2:0.629946 valid-r2:0.581652

R2 = 0.5812543076056167
Fold #7
XGB: Evaluating model
[0] train-rmse:12.3552 valid-rmse:12.0441 train-r2:0.005933 valid-r2:0.006128
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.
[100] train-rmse:9.75764 valid-rmse:9.32006 train-r2:0.379983 valid-r2:0.404856
[200] train-rmse:8.59138 valid-rmse:8.1069 train-r2:0.519338 valid-r2:0.549708
[300] train-rmse:8.10417 valid-rmse:7.62493 train-r2:0.572307 valid-r2:0.601658
[400] train-rmse:7.88408 valid-rmse:7.43952 train-r2:0.595222 valid-r2:0.620795
[500] train-rmse:7.76241 valid-rmse:7.3683 train-r2:0.607619 valid-r2:0.62802
```

```
[600] train-rmse:7.68187 valid-rmse:7.33635 train-r2:0.61572 valid-r2:0.631239
[700] train-rmse:7.61771 valid-rmse:7.3256 train-r2:0.622112 valid-r2:0.632319
[800] train-rmse:7.56239 valid-rmse:7.31944 train-r2:0.627581 valid-r2:0.632937
[900] train-rmse:7.50889 valid-rmse:7.3167 train-r2:0.632831 valid-r2:0.633212
Stopping. Best iteration:
[924] train-rmse:7.49643 valid-rmse:7.31583 train-r2:0.634049 valid-r2:0.633299

R2 = 0.633100803530456
Fold #8
XGB: Evaluating model
[0] train-rmse:12.1642 valid-rmse:12.8003 train-r2:0.005963 valid-r2:0.004828
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.
[100] train-rmse:9.56887 valid-rmse:10.2088 train-r2:0.384881 valid-r2:0.366997
[200] train-rmse:8.39869 valid-rmse:9.03328 train-r2:0.526128 valid-r2:0.504383
[300] train-rmse:7.91093 valid-rmse:8.53931 train-r2:0.579571 valid-r2:0.557106
[400] train-rmse:7.69807 valid-rmse:8.33053 train-r2:0.601891 valid-r2:0.578498
[500] train-rmse:7.57724 valid-rmse:8.24542 train-r2:0.61429 valid-r2:0.587067
[600] train-rmse:7.49254 valid-rmse:8.20507 train-r2:0.622865 valid-r2:0.591098
[700] train-rmse:7.42518 valid-rmse:8.18486 train-r2:0.629616 valid-r2:0.59311
[800] train-rmse:7.37043 valid-rmse:8.17573 train-r2:0.635057 valid-r2:0.594017
[900] train-rmse:7.31994 valid-rmse:8.17222 train-r2:0.640041 valid-r2:0.594365
Stopping. Best iteration:
[867] train-rmse:7.33653 valid-rmse:8.17182 train-r2:0.638407 valid-r2:0.594406

R2 = 0.5943769871984499
Fold #9
XGB: Evaluating model
[0] train-rmse:12.4221 valid-rmse:11.7645 train-r2:0.006007 valid-r2:0.005639
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.
[100] train-rmse:9.76239 valid-rmse:9.26333 train-r2:0.38609 valid-r2:0.383499
[200] train-rmse:8.56069 valid-rmse:8.18938 train-r2:0.527926 valid-r2:0.51816
[300] train-rmse:8.05729 valid-rmse:7.78058 train-r2:0.581813 valid-r2:0.565065
[400] train-rmse:7.83718 valid-rmse:7.61955 train-r2:0.604349 valid-r2:0.582882
[500] train-rmse:7.72236 valid-rmse:7.56344 train-r2:0.615857 valid-r2:0.589003
[600] train-rmse:7.64564 valid-rmse:7.53085 train-r2:0.623452 valid-r2:0.592537
[700] train-rmse:7.58252 valid-rmse:7.52443 train-r2:0.629644 valid-r2:0.593231
Stopping. Best iteration:
[711] train-rmse:7.57629 valid-rmse:7.52338 train-r2:0.630251 valid-r2:0.593345

R2 = 0.5930718577981611
Fold #10
XGB: Evaluating model
[0] train-rmse:12.24 valid-rmse:12.5085 train-r2:0.005998 valid-r2:0.005681
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.
[100] train-rmse:9.62262 valid-rmse:9.98336 train-r2:0.385658 valid-r2:0.366614
[200] train-rmse:8.44074 valid-rmse:8.8388 train-r2:0.527302 valid-r2:0.50352
[300] train-rmse:7.94529 valid-rmse:8.3592 train-r2:0.581164 valid-r2:0.555937
[400] train-rmse:7.72561 valid-rmse:8.15113 train-r2:0.604005 valid-r2:0.577769
[500] train-rmse:7.60754 valid-rmse:8.06173 train-r2:0.616017 valid-r2:0.586979
[600] train-rmse:7.53273 valid-rmse:8.0133 train-r2:0.623532 valid-r2:0.591927
[700] train-rmse:7.46792 valid-rmse:7.99187 train-r2:0.629982 valid-r2:0.594107
[800] train-rmse:7.41187 valid-rmse:7.98031 train-r2:0.635516 valid-r2:0.59528
[900] train-rmse:7.35876 valid-rmse:7.97429 train-r2:0.640721 valid-r2:0.59589
Stopping. Best iteration:
[890] train-rmse:7.36398 valid-rmse:7.97395 train-r2:0.640211 valid-r2:0.595925

R2 = 0.595779890830896
Fold #11
XGB: Evaluating model
[0] train-rmse:12.1852 valid-rmse:12.7188 train-r2:0.006077 valid-r2:0.004993
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.
[100] train-rmse:9.53788 valid-rmse:10.2399 train-r2:0.391038 valid-r2:0.355052
[200] train-rmse:8.33973 valid-rmse:9.16631 train-r2:0.534423 valid-r2:0.483196
[300] train-rmse:7.83609 valid-rmse:8.74811 train-r2:0.588959 valid-r2:0.529278
[400] train-rmse:7.61317 valid-rmse:8.59089 train-r2:0.612013 valid-r2:0.546046
[500] train-rmse:7.49207 valid-rmse:8.53034 train-r2:0.624258 valid-r2:0.552422
[600] train-rmse:7.40367 valid-rmse:8.51815 train-r2:0.633072 valid-r2:0.5537
Stopping. Best iteration:
[645] train-rmse:7.37 valid-rmse:8.51578 train-r2:0.636402 valid-r2:0.553949
```

```
R2 = 0.5538135431540314
Fold #12
XGB: Evaluating model
[0] train-rmse:12.3296 valid-rmse:12.1488 train-r2:0.005936 valid-r2:0.004393
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.
[100] train-rmse:9.69766 valid-rmse:9.59275 train-r2:0.38503 valid-r2:0.379268
[200] train-rmse:8.51149 valid-rmse:8.46469 train-r2:0.52627 valid-r2:0.516674
[300] train-rmse:8.01533 valid-rmse:8.0151 train-r2:0.579891 valid-r2:0.566653
[400] train-rmse:7.80208 valid-rmse:7.82739 train-r2:0.601948 valid-r2:0.586713
[500] train-rmse:7.68494 valid-rmse:7.76081 train-r2:0.61381 valid-r2:0.593714
[600] train-rmse:7.60238 valid-rmse:7.73157 train-r2:0.622064 valid-r2:0.59677
[700] train-rmse:7.53985 valid-rmse:7.7127 train-r2:0.628255 valid-r2:0.598735
[800] train-rmse:7.48644 valid-rmse:7.70588 train-r2:0.633503 valid-r2:0.599445
Stopping. Best iteration:
[841] train-rmse:7.46565 valid-rmse:7.70458 train-r2:0.635536 valid-r2:0.599581

R2 = 0.5994461161814348
Fold #13
XGB: Evaluating model
[0] train-rmse:12.249 valid-rmse:12.4728 train-r2:0.005897 valid-r2:0.006091
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.
[100] train-rmse:9.68791 valid-rmse:9.72424 train-r2:0.378141 valid-r2:0.395871
[200] train-rmse:8.53746 valid-rmse:8.45479 train-r2:0.517063 valid-r2:0.543307
[300] train-rmse:8.05587 valid-rmse:7.9093 train-r2:0.570011 valid-r2:0.600337
[400] train-rmse:7.83844 valid-rmse:7.69163 train-r2:0.592909 valid-r2:0.622031
[500] train-rmse:7.71636 valid-rmse:7.60337 train-r2:0.605491 valid-r2:0.630656
[600] train-rmse:7.63842 valid-rmse:7.56181 train-r2:0.613421 valid-r2:0.634683
[700] train-rmse:7.57727 valid-rmse:7.54273 train-r2:0.619585 valid-r2:0.636524
[800] train-rmse:7.52635 valid-rmse:7.53256 train-r2:0.624681 valid-r2:0.637504
[900] train-rmse:7.47702 valid-rmse:7.52844 train-r2:0.629584 valid-r2:0.6379
[999] train-rmse:7.42874 valid-rmse:7.52628 train-r2:0.634353 valid-r2:0.638108
R2 = 0.6381075400531017
Fold #14
XGB: Evaluating model
[0] train-rmse:12.4632 valid-rmse:11.589 train-r2:0.005952 valid-r2:0.006376
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.
[100] train-rmse:9.83863 valid-rmse:8.96531 train-r2:0.380533 valid-r2:0.405352
[200] train-rmse:8.65655 valid-rmse:7.81588 train-r2:0.520445 valid-r2:0.548056
[300] train-rmse:8.15849 valid-rmse:7.37881 train-r2:0.57404 valid-r2:0.597188
[400] train-rmse:7.93061 valid-rmse:7.23054 train-r2:0.597503 valid-r2:0.613215
[500] train-rmse:7.80423 valid-rmse:7.17841 train-r2:0.610229 valid-r2:0.618772
[600] train-rmse:7.7217 valid-rmse:7.16038 train-r2:0.61843 valid-r2:0.620684
[700] train-rmse:7.65417 valid-rmse:7.1622 train-r2:0.625074 valid-r2:0.620492
Stopping. Best iteration:
[654] train-rmse:7.68426 valid-rmse:7.15698 train-r2:0.622121 valid-r2:0.621045

R2 = 0.6204555651407138
XGB Mean R2 = 0.600904797103415 +/- 0.02215987402178757
XGB: Train on full dataset and predicting on test
[0] train-rmse:12.2938 train-r2:0.005993
[100] train-rmse:9.67984 train-r2:0.383758
[200] train-rmse:8.50574 train-r2:0.524184
[300] train-rmse:8.01605 train-r2:0.577394
[400] train-rmse:7.8037 train-r2:0.599486
[500] train-rmse:7.69273 train-r2:0.610797
[600] train-rmse:7.62106 train-r2:0.618015
[699] train-rmse:7.56359 train-r2:0.623754
```

LB(.54638,56166)CV:(.6009)

SD of CV also produces as estimate as how well our folds are responding to these showing clipping of outliers clearly works.

LR with OneHotEncoding Done Here

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_hot.csv')
```

```
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_hot.csv')
results=  pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-
manufacturing/sample_submission.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
#using onehot on categorical columns,leaving binary columns as they are already normalized.
from sklearn.preprocessing import MinMaxScaler
ids_test= test.ID.values
SS=MinMaxScaler()
train['ID']=SS.fit_transform(train.ID.values.reshape(-1,1))
test['ID']= SS.transform(test.ID.values.reshape(-1,1))

x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {'alpha':[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1,1e1,1e2,1e3],'loss':['squared_loss']}
model= SGDRegressor(random_state= random_seed,penalty= 'l1')

clf = GridSearchCV(model, parameters, cv=cv, scoring='r2',verbose=1,return_train_score=True)
clf.fit(X, y)

print(clf.best_params_,clf.best_score_)
```

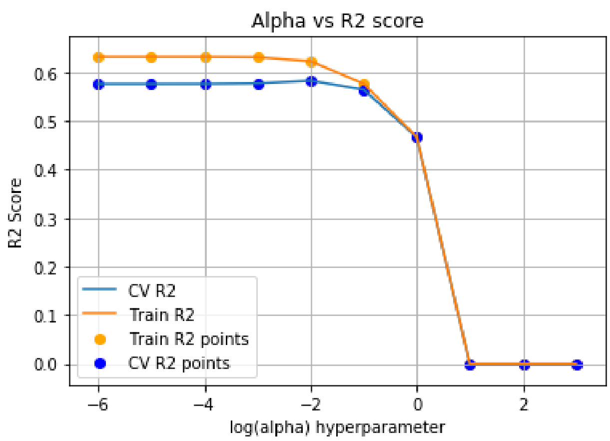
```
(4209, 394) X.shape
Fitting 15 folds for each of 10 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:  1.3min finished
```

```
{'alpha': 0.01, 'loss': 'squared_loss'} 0.5830251985845702
```

In [0]:

```
cv_r1= clf.cv_results_['mean_train_score']
cv_r2 = clf.cv_results_['mean_test_score']
plt.plot(np.log10(parameters['alpha']),cv_r2,label= 'CV R2')
plt.title('Alpha vs R2 score')
plt.xlabel('log(alpha) hyperparameter')
plt.ylabel('R2 Score')
plt.plot(np.log10(parameters['alpha']),cv_r1,label= 'Train R2')
plt.scatter(np.log10(parameters['alpha']),cv_r1,label= 'Train R2 points',color= 'orange')
plt.scatter(np.log10(parameters['alpha']),cv_r2,label= 'CV R2 points',color= 'blue')
plt.grid()
plt.legend()
plt.show()
```



In [0]:

```
model= SGDRegressor(**clf.best_params_,random_state= random_seed)
model.fit(X,y)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = model.predict(x_test)
subm.to_csv('lr_5folds.csv', index=False)
```

LB(0.53780,0.53253) cv:.58302

Little t-test to check Importance of ID Feature

In [0]:

```
cv1=RepeatedKFold(n_splits= 5,n_repeats= 50,random_state= random_seed)#250 folds
lr_id=cross_val_score(model,X,y,scoring='r2',cv= cv1,verbose=1,n_jobs=1)
lr_idless=cross_val_score(model,X[:,1:],y,scoring='r2',cv= cv1,verbose=1,n_jobs=1)#data with ID col
umn removed
```

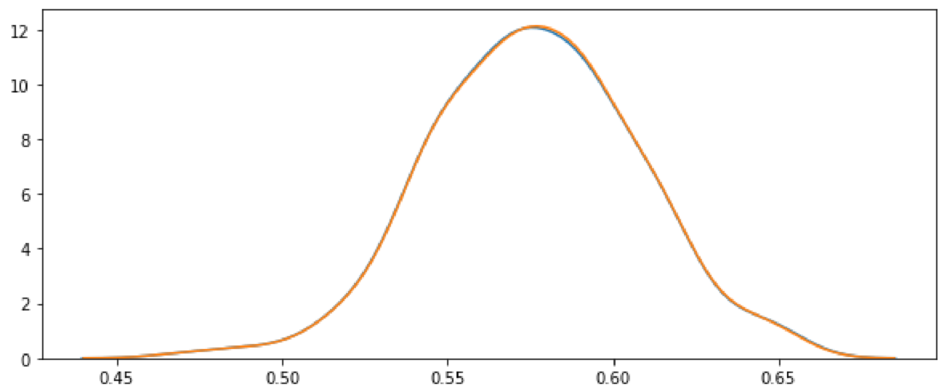
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 250 out of 250 | elapsed:  1.3min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 250 out of 250 | elapsed:  1.3min finished
```

In [0]:

```
#https://machinelearningmastery.com/parametric-ue,axis=1)statistical-significance-tests-in-python/
plt.figure(figsize=(10,4))
sns.distplot(lr_id,hist= False)
sns.distplot(lr_idless,hist= False)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6026e24780>
```



The gaussian distribution shows that t-test can be used for statistical significance testing.Lets try to check the null hypothesis that cv with ID and without ID are from same distribution.

In [0]:

```
#Now as the samples are not independent we have to use scipy.ttest_rel or Paired students t_test d
istribution.
stats.ttest_rel(lr_id,lr_idless)
```

Out[0]:

```
Ttest_relResult(statistic=-1.7934487843033402, pvalue=0.07411461054366326)
```

This test concludes that with the training setup and CV that we have used ID and Not using them doesent make any difference. Here we go out of what has been mentioned in kaggle kernels that ID Provided significant improvement as pvalue >.05 null hypothesis cant be rejected. The reson can be that model was complex enough to make use of other features that ID becomes redundant.

Mean Encoding for LR Done here

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_meanenc.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_meanenc.csv'
)

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
ids_test= test.ID.values
SS=MinMaxScaler()
train.iloc[:,:8]=SS.fit_transform(train.iloc[:,:8])#Id feature with all the cat variables are resc
aled
test.iloc[:,:8]= SS.transform(test.iloc[:,:8])
x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {'alpha':[1e-1,1,1e1,1e-2,1e-3,1e-4],'loss':['squared_loss'],'penalty':['l1','l2']}
model= SGDRegressor(random_state= random_seed,)

clf = GridSearchCV(model, parameters, cv=cv, scoring='r2',verbose=1)
clf.fit(X, y)

print(clf.best_params_,clf.best_score_)
```

```
(4209, 210) X.shape
Fitting 15 folds for each of 12 candidates, totalling 180 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 180 out of 180 | elapsed:   47.3s finished
```

```
{'alpha': 0.01, 'loss': 'squared_loss', 'penalty': 'l1'} 0.5838024669319191
```

In [0]:

```
model= SGDRegressor(**clf.best_params_,random_state= random_seed)
model.fit(X,y)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = model.predict(x_test)
subm.to_csv('lr_5folds_meanenc.csv', index=False)
```

LB(0.53792,0.54382)CV:.58380

MeanEncoding Done Here for RandomForestRegressor

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_meanenc.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_meanenc.csv'
)

y_train= train.y.values
targets= y_train.copy()
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()

x_test= np.array(test)
X= np.array(train)
y= targets
```

```
print(X.shape,'X.shape')
parameters= {"n_estimators":[600],
              "max_depth": [4],#[3,4,5,6,7,8,9,10],#list(range(2,10)),#4 is the best
              "min_samples_leaf": [5],#[1,2,3,4,5,6],#[3,4,5,6,7],
              "max_features": [.95],
             'min_impurity_decrease':[1e-2],#[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]
              }
model= RandomForestRegressor(n_jobs=1,random_state= random_seed)

clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
```

```
(4209, 210) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   42.9s finished
```

```
{'max_depth': 4, 'max_features': 0.95, 'min_impurity_decrease': 0.01, 'min_samples_leaf': 5,
'n_estimators': 600}
```

In [0]:

```
rf_tar= RandomForestRegressor(**clf.best_params_,random_state= random_seed,oob_score= True)
rf_tar.fit(X,y)
print('oob_score: ',rf_tar.oob_score_)
cv_score=cross_val_score(rf_tar,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = rf_tar.predict(x_test)
subm.to_csv('rf_5folds_meanenc.csv', index=False)
```

```
oob_score:  0.6054807310827848
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

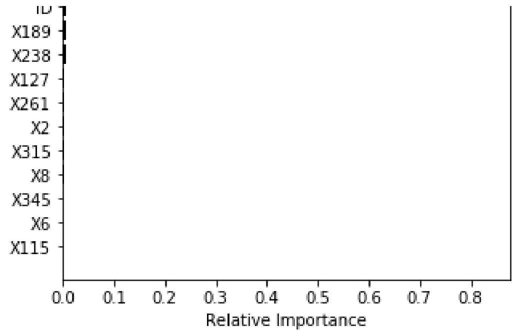
```
0.6055514389987438  +/-  0.022463458244656155
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  2.6min finished
```

LB(0.54897,0.55864),CV:.6055

In [0]:

```
features = train.columns
importances = rf_tar.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



MeanEncoding Using XGBoost

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_meanenc.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_meanenc.csv'
)

y_train= train.y.values
targets= y_train.copy()
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
cv3= KFold(5,True,random_seed)
parameters= {'learning_rate': [0.05],
               'subsample': [.72],#[.9,.8,.7,.6,.5,1],
               'colsample_bytree': [.72],#[.9,.8,.7,.6,.5,1],#[.8],#[0.8,.85]
               'min_child_weight':[10],#[10,20,30,50,100,150,200], #[1,5,10],#[110,120,130]
               'max_depth': [2],#[2,4,6,10],
               'n_estimators':[151],
               'verbosity':[1],
             'gamma':[.01],#[1e-2,1e-3,1e-4,0,.1,.2,.3,.4,.5,1,3,5,10],
             'reg_alpha':[1],#[1e-5,1e-3,1e-1,1,1e1,1e2]

             }
model= xgb.XGBRegressor(n_jobs=1,random_state= random_seed,verbosity=1,silent=True)
clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    7.6s finished
```

```
{'colsample_bytree': 0.72, 'gamma': 0.01, 'learning_rate': 0.05, 'max_depth': 2,
'min_child_weight': 10, 'n_estimators': 151, 'reg_alpha': 1, 'subsample': 0.72, 'verbosity': 1}
```

In [0]:

```
xgb_tar= xgb.XGBRegressor(**clf.best_params_,random_state= random_seed,silent=True)
X_lab= pd.DataFrame(X,columns= train.columns)
x_test_lab=pd.DataFrame(x_test,columns= train.columns)
xgb_tar.fit(X_lab,y)
cv_score=cross_val_score(xgb_tar,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = xgb_tar.predict(x_test_lab)
subm.to_csv('xgb_meanenc.csv', index=False)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

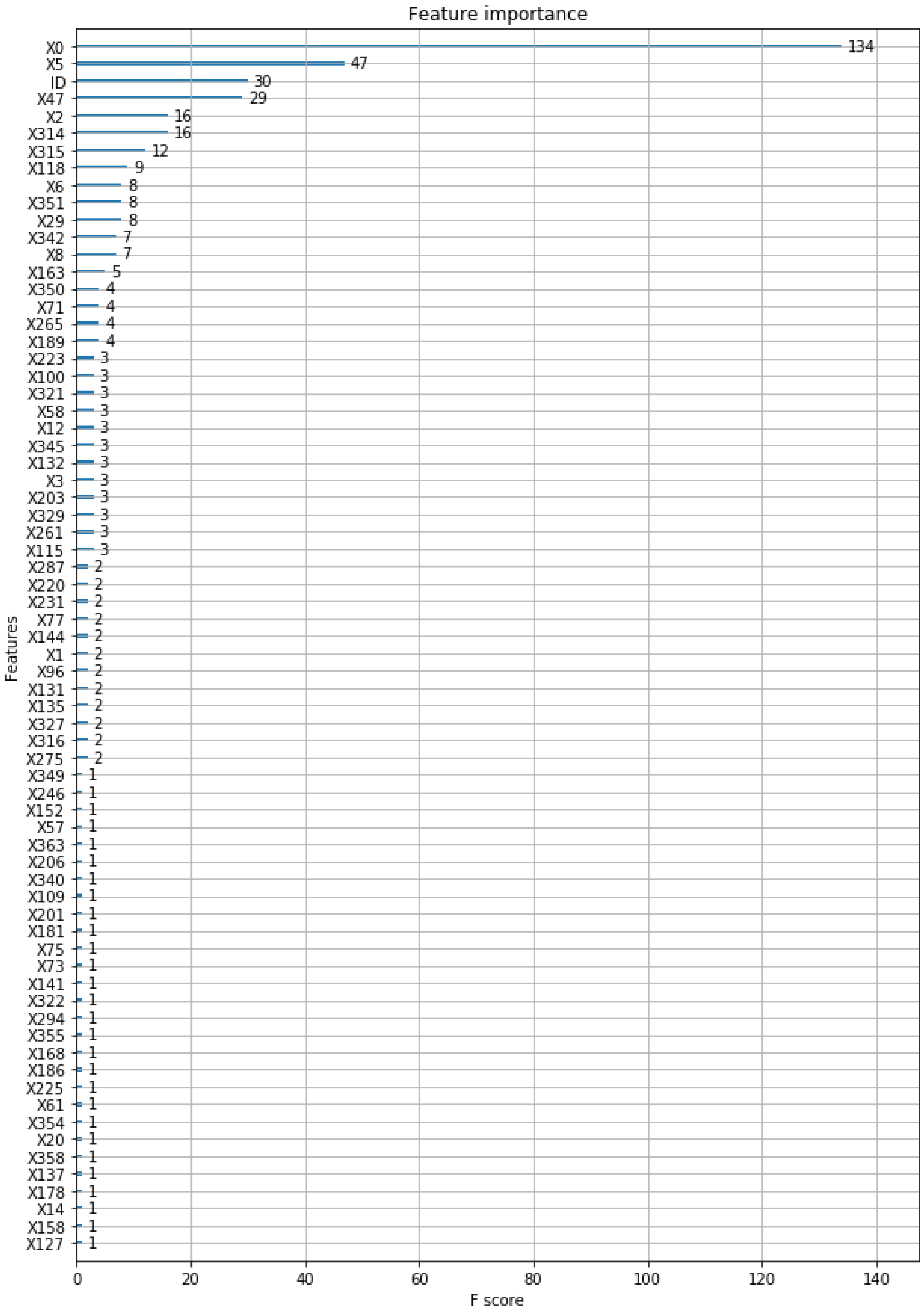
```
0.6081670903478663  +/-  0.02161144780988898
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   27.3s finished
```

LB(0.54832,0.55701)CV:.60816

In [0]:

```
from xgboost import plot_importance
fig, ax = plt.subplots(figsize=(10, 15))
plot_importance(xgb_mean,max_num_features= 70, ax=ax)
plt.show()
```



MeanEncoding on ExtraTreesRegressor

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()
```

```
x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {"n_estimators":[750],#range(700,1500,50),
              "max_depth": [4],#[3,4,5,6,7,8,9,10],#list(range(2,10)),#4 is the best
              "min_samples_leaf": [10],#[3,4,5,6,7],
              "max_features": [.95],#[.95],
             'min_impurity_decrease':[1e-4],#[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]
              }
model= ExtraTreesRegressor(n_jobs=1,random_state= random_seed)

clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                        verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
```

```
(4209, 210) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

```
{'max_depth': 4, 'max_features': 0.95, 'min_impurity_decrease': 0.0001, 'min_samples_leaf': 10, 'n
_estimators': 750}
```

In [0]:

```
et_tar= ExtraTreesRegressor(**clf.best_params_,random_state= random_seed,oob_score= True,bootstrap
= True)
et_tar.fit(X,y)
print('oob_score: ',et_tar.oob_score_)
cv_score=cross_val_score(et_tar,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = et_tar.predict(x_test)
subm.to_csv('et_5folds_meanenc.csv', index=False)
```

```
oob_score:  0.6030986205377533
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6009667986911981  +/-  0.021531916778920388
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  2.4min finished
```

LB(0.54831,0.55545),CV:.60096

Random Forest Regressor LabelEncoding

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_label.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_label.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()
```

```
x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {"n_estimators":[600],
              "max_depth": [4],#[3,4,5,6,7,8,9,10],#list(range(2,10)),#4 is the best
              "min_samples_leaf": [5],#[1,2,3,4,5,6],#[3,4,5,6,7],
              "max_features": [.95],
             'min_impurity_decrease':[1e-2],#[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]
              }
model= RandomForestRegressor(n_jobs=1,random_state= random_seed)

clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                             verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
```

```
(4209, 210) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   41.0s finished
```

```
{'max_depth': 4, 'max_features': 0.95, 'min_impurity_decrease': 0.01, 'min_samples_leaf': 5,
'n_estimators': 600}
```

In [0]:

```
rf_label= RandomForestRegressor(**clf.best_params_,random_state= random_seed,oob_score= True)
rf_label.fit(X,y)
print('oob_score: ',rf_label.oob_score_)
cv_score=cross_val_score(rf_label,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = rf_label.predict(x_test)
subm.to_csv('rf_5folds_label.csv', index=False)
```

```
oob_score:  0.6022263656601465
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

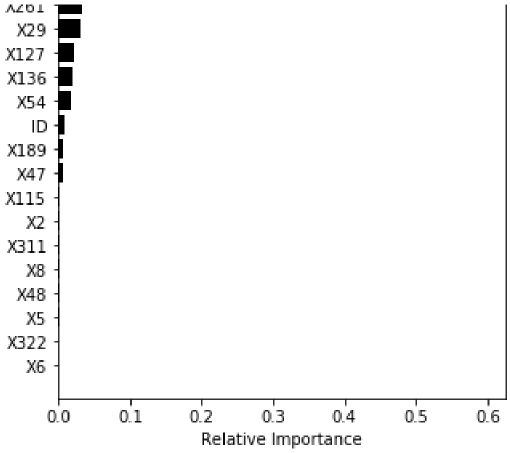
```
0.6023986685575103  +/-  0.02119851269797542
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  2.5min finished
```

LB(0.54986,0.55686),CV:.6023

In [0]:

```
features = train.columns
importances = rf_label.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Xgboost with LabelEncoding

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()

x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {'learning_rate': [0.05],
               'subsample': [.72],#[.9,.8,.7,.6,.5,1],
               'colsample_bytree': [.72],#[.9,.8,.7,.6,.5,1],#[.8],#[0.8,.85]
               'min_child_weight':[10],#[10,20,30,50,100,150,200], #[1,5,10],#[110,120,130]
               'max_depth': [2],#[2,4,6,10],
               'n_estimators':[151],
               'verbosity':[1],
             'gamma':[.01],#[1e-2,1e-3,1e-4,0,.1,.2,.3,.4,.5,1,3,5,10],
             'reg_alpha':[1],#[1e-5,1e-3,1e-1,1,1e1,1e2]

             }
model= xgb.XGBRegressor(n_jobs=1,random_state= random_seed,verbosity=1,silent=True)
clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
```

```
(4209, 210) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    7.0s finished
```

```
{'colsample_bytree': 0.72, 'gamma': 0.01, 'learning_rate': 0.05, 'max_depth': 2,
'min_child_weight': 10, 'n_estimators': 151, 'reg_alpha': 1, 'subsample': 0.72, 'verbosity': 1}
```

In [0]:

```
xgb_lab= xgb.XGBRegressor(**clf.best_params_,random_state= random_seed,silent=True)
X_lab= pd.DataFrame(X,columns= train.columns)
x_test_lab=pd.DataFrame(x_test,columns= train.columns)
xgb_lab.fit(X_lab,y)
cv_score=cross_val_score(xgb_lab,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = xgb_lab.predict(x_test_lab)
subm.to_csv('xgb_meanenc.csv', index=False)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6042210150371721  +/-  0.021417573272027906
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   26.6s finished
```

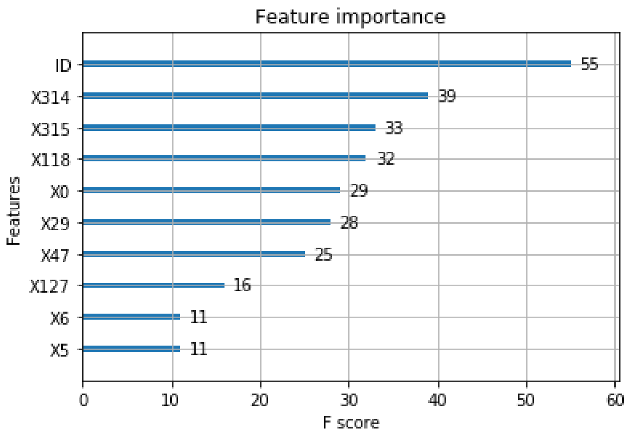
LB(0.55034,0.55555)CV:.60422

In [0]:

```
plot_importance(xgb_lab,max_num_features= 10)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f650ac2dfd0>
```



ExtraTrees Regressor LabelEncoding

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()

x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {"n_estimators":[350],#range(50,600,50),
              "max_depth": [4],#[3,4,5,6,7,8,9,10],#list(range(2,10)),#4 is the best
              "min_samples_leaf": [10],#[3,4,5,6,7],
              "max_features": [.95],#[.95],
             'min_impurity_decrease':[1e-4],#[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]
              }
model= ExtraTreesRegressor(n_jobs=1,random_state= random_seed)

clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
```

```
(4209, 210) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   27.9s finished
```

```
{'max_depth': 4, 'max_features': 0.95, 'min_impurity_decrease': 0.0001, 'min_samples_leaf': 10, 'n
_estimators': 350}
```

In [0]:

```
et_lab= ExtraTreesRegressor(**clf.best_params_,random_state= random_seed,oob_score= True,bootstrap
= True)
et_lab.fit(X,y)
print('oob_score: ',et_lab.oob_score_)
cv_score=cross_val_score(et_lab,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = et_lab.predict(x_test)
subm.to_csv('et_5folds_label.csv', index=False)
```

```
oob_score:  0.6010326127840766
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.5986308922513477  +/-  0.021121052286242285
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.1min finished
```

LB(0.54703,0.55379),CV:.59863

Stacking all the models trained on LabelEncoded dataset

In [0]:

```
ridge= Ridge(random_state=random_seed,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(rf_label, xgb_lab,et_lab),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=cv)

cv_score=cross_val_score(stack,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
print(cv_score.mean(),' +/- ',cv_score.std())
stack.fit(X,y)
ids_test= test.ID
y_pred = stack.predict(x_test)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = y_pred
subm.to_csv('submission_xgb_rf_stack_ridge_label.csv', index=False)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 43.7min finished
```

```
0.6052152518324975  +/-  0.02131608341482325
```

LB(.55226,.55790),CV:.60521

Using Feature Interactions

In [0]:

```
#https://www.kaggle.com/qqgeogor/some-feature-engineering
from scipy.stats import spearmanr
sum_cols = []
for c in binary_cols:
    score = (spearmanr(y,train[c]))
    if score[0]>=0.2 and score[0]<=0.3:
        print(c,score)
        sum_cols.append(c)

train['sum_row_2_to_3'] = train.drop('ID', axis=1)[sum_cols].sum(axis=1)
test['sum_row_2_to_3'] = test.drop('ID', axis=1)[sum_cols].sum(axis=1)
```

```
X14 SpearmanrResult(correlation=0.23271473847676327, pvalue=7.203142355334819e-53)
X48 SpearmanrResult(correlation=0.20357136242667562, pvalue=1.2996316630463163e-40)
```

```
X48 SpearmanrResult(correlation=0.2035713624266756?, pvalue=1.2996316630463163e-40)
X51 SpearmanrResult(correlation=0.2611121293074829, pvalue=1.4523427512271017e-66)
X66 SpearmanrResult(correlation=0.21315968127467158, pvalue=1.8967188268459342e-44)
X118 SpearmanrResult(correlation=0.27090253163947037, pvalue=1.0602732360517293e-71)
X126 SpearmanrResult(correlation=0.2417200374417423, pvalue=5.07364560585800le-57)
X130 SpearmanrResult(correlation=0.23415409276582105, pvalue=1.6054566396772044e-53)
X179 SpearmanrResult(correlation=0.2366639405029322, pvalue=1.1432996184544922e-54)
X191 SpearmanrResult(correlation=0.2296580095236195, pvalue=1.687603773776457e-51)
X198 SpearmanrResult(correlation=0.2055358155294325, pvalue=2.2052011207271959e-41)
X223 SpearmanrResult(correlation=0.22158801608519174, pvalue=5.595741666819109e-48)
X224 SpearmanrResult(correlation=0.2202985852333516, pvalue=1.9842662148088542e-47)
X251 SpearmanrResult(correlation=0.23130280662901975, pvalue=3.109618518431174e-52)
X264 SpearmanrResult(correlation=0.24384866402424946, pvalue=4.990798176113577e-58)
X275 SpearmanrResult(correlation=0.27139287140038404, pvalue=5.786853161764286e-72)
X306 SpearmanrResult(correlation=0.20025584540217214, pvalue=2.4888178471384066e-39)
X311 SpearmanrResult(correlation=0.20604979188717262, pvalue=1.3822106096801595e-41)
X315 SpearmanrResult(correlation=0.2003369580762672, pvalue=2.3168337750087892e-39)
```

In [0]:

```python
print('feature 2')
sum_cols = []
for c in binary_cols:
    score = (spearmanr(y,train[c]))
    if score[0]>=0.1 and score[0]<=0.2:
        print(c,score)
        sum_cols.append(c)

train['sum_row_1_to_2'] = train.drop('ID', axis=1)[sum_cols].sum(axis=1)
test['sum_row_1_to_2'] = test.drop('ID', axis=1)[sum_cols].sum(axis=1)
```

```
feature 2
X47 SpearmanrResult(correlation=0.12128833484321079, pvalue=2.8886024870257605e-15)
X52 SpearmanrResult(correlation=0.1973661460541078, pvalue=3.12675244238439e-38)
X64 SpearmanrResult(correlation=0.1018128914475367, pvalue=3.578824433646211e-11)
X68 SpearmanrResult(correlation=0.1620006747077727, pvalue=3.8136608908043135e-26)
X71 SpearmanrResult(correlation=0.1488980228079688, pvalue=2.697699163541339e-22)
X75 SpearmanrResult(correlation=0.15457622673680788, pvalue=6.364261721190012e-24)
X85 SpearmanrResult(correlation=0.12812999370557837, pvalue=7.144875915703804e-17)
X96 SpearmanrResult(correlation=0.15027972692024613, pvalue=1.09846238933138e-22)
X150 SpearmanrResult(correlation=0.158286007669749, pvalue=5.090255968583112e-25)
X151 SpearmanrResult(correlation=0.10641844916400506, pvalue=4.462529252730155e-12)
X155 SpearmanrResult(correlation=0.1324120072574912, pvalue=6.357836342267669e-18)
X156 SpearmanrResult(correlation=0.15733524644002178, pvalue=9.782389408056896e-25)
X170 SpearmanrResult(correlation=0.18496198665489488, pvalue=1.0467972693778774e-33)
X180 SpearmanrResult(correlation=0.13961195349211947, pvalue=9.076232006156846e-20)
X187 SpearmanrResult(correlation=0.17533512536462786, pvalue=2.0784884314473318e-30)
X197 SpearmanrResult(correlation=0.10093443281576443, pvalue=5.26917067879295e-11)
X208 SpearmanrResult(correlation=0.10268676629017812, pvalue=2.427729630970363e-11)
X228 SpearmanrResult(correlation=0.13851145288704508, pvalue=1.763550391559232e-19)
X241 SpearmanrResult(correlation=0.12792750298267436, pvalue=7.995002119385666e-17)
X255 SpearmanrResult(correlation=0.12673703162225913, pvalue=1.542732133835433e-16)
X300 SpearmanrResult(correlation=0.1898497322521831, pvalue=1.8800384633817425e-35)
X331 SpearmanrResult(correlation=0.11129622821365086, pvalue=4.457032549198639e-13)
X336 SpearmanrResult(correlation=0.10845626975293331, pvalue=1.725695040238013e-12)
X343 SpearmanrResult(correlation=0.14082256066685175, pvalue=4.343890516376526e-20)
X346 SpearmanrResult(correlation=0.10633459482991026, pvalue=4.638682505000038e-12)
X349 SpearmanrResult(correlation=0.10711713753912025, pvalue=3.2283458226849445e-12)
X352 SpearmanrResult(correlation=0.10868172714744195, pvalue=1.5518127537038296e-12)
X354 SpearmanrResult(correlation=0.13550704330465846, pvalue=1.0523423958299538e-18)
X355 SpearmanrResult(correlation=0.13494346470740545, pvalue=1.4647314937577317e-18)
X363 SpearmanrResult(correlation=0.140751755338967, pvalue=4.5360080897504667e-20)
X367 SpearmanrResult(correlation=0.11210964584749873, pvalue=3.0052862540951983e-13)
X368 SpearmanrResult(correlation=0.10153251624707284, pvalue=4.0505682892088874e-11)
X376 SpearmanrResult(correlation=0.13212760762123826, pvalue=7.484679814830494e-18)
```

In [0]:

```python
print('feature 3')
sum_cols = []
for c in binary_cols:
    score = (spearmanr(y,train[c]))
    if score[0]>=0.05 and score[0]<=0.1:
        print(c,score)
        sum_cols.append(c)
```

```
train['sum_row_05_to_1'] = train.drop('ID', axis=1)[sum_cols].sum(axis=1)
test['sum_row_05_to_1'] = test.drop('ID', axis=1)[sum_cols].sum(axis=1)
```

```
feature 3
X12 SpearmanrResult(correlation=0.08803722235600211, pvalue=1.0590641386771599e-08)
X13 SpearmanrResult(correlation=0.05170775007267667 4, pvalue=0.0007911225065559363)
X44 SpearmanrResult(correlation=0.09077275830914208, pvalue=3.6465001055743447e-09)
X69 SpearmanrResult(correlation=0.08971114020696344, pvalue=5.536034403876071e-09)
X82 SpearmanrResult(correlation=0.05364350728508881 4, pvalue=0.0004982552551932894)
X109 SpearmanrResult(correlation=0.0745623704890028, pvalue=1.281036881645429e-06)
X131 SpearmanrResult(correlation=0.07308723350648445, pvalue=2.068575956661907e-06)
X142 SpearmanrResult(correlation=0.08659647934328635, pvalue=1.8336344983701157e-08)
X163 SpearmanrResult(correlation=0.06583065125948522, pvalue=1.9178121894735077e-05)
X171 SpearmanrResult(correlation=0.08929405917188903, pvalue=6.5143701262795555e-09)
X176 SpearmanrResult(correlation=0.05596658896258349, pvalue=0.0002804684439862748)
X177 SpearmanrResult(correlation=0.05371222547262586, pvalue=0.0004900093282330147)
X189 SpearmanrResult(correlation=0.08218439546249769, pvalue=9.330036351108397e-08)
X211 SpearmanrResult(correlation=0.06813896741092876, pvalue=9.668012242365568e-06)
X219 SpearmanrResult(correlation=0.07935224746508104, pvalue=2.5400892891807954e-07)
X225 SpearmanrResult(correlation=0.05269253420717179, pvalue=0.0006264789958319144)
X238 SpearmanrResult(correlation=0.07812144065596013, pvalue=3.884742270359681e-07)
X283 SpearmanrResult(correlation=0.05843083105502538, pvalue=0.00014888585859330333)
X285 SpearmanrResult(correlation=0.08213315400676739, pvalue=9.503464542426387e-08)
X329 SpearmanrResult(correlation=0.06186839070436732, pvalue=5.907406368270677e-05)
X334 SpearmanrResult(correlation=0.06401628564243005, pvalue=3.235819253228418e-05)
X351 SpearmanrResult(correlation=0.09477077714758957, pvalue=7.253009815587749e-10)
X377 SpearmanrResult(correlation=0.05786229631946555, pvalue=0.0001726831680043041)
```

Interactions Included

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()

x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {"n_estimators":[600],
              "max_depth": [4],#[3,4,5,6,7,8,9,10],#list(range(2,10)),#4 is the best
              "min_samples_leaf": [5],#[1,2,3,4,5,6],#[3,4,5,6,7],
              "max_features": [.95],
             'min_impurity_decrease':[1e-2],#[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]
              }
model= RandomForestRegressor(n_jobs=1,random_state= random_seed)

clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
rf_label= RandomForestRegressor(**clf.best_params_,random_state= random_seed,oob_score= True)
rf_label.fit(X,y)
print('oob_score: ',rf_label.oob_score_)
```

```
(4209, 213) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   43.8s finished
```

```
{'max_depth': 4, 'max_features': 0.95, 'min_impurity_decrease': 0.01, 'min_samples_leaf': 5,
'n_estimators': 600}
oob_score:  0.6021561663055823
```

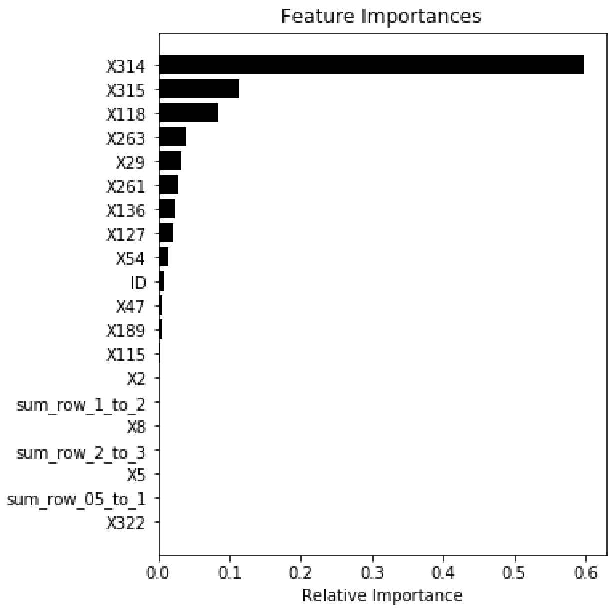
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6022266021009817  +/-  0.021248052108807296
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  2.6min finished
```

In [0]:

```
features = train.columns
importances = rf_label.feature_importances_
indices = (np.argsort(importances))[-20:]
plt.figure(figsize=(5,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='k', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



new features found have got a place in the feature importance chart

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()

x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {'learning_rate': [0.05],
               'subsample': [.72],#[.9,.8,.7,.6,.5,1],
               'colsample_bytree': [.72],#[.9,.8,.7,.6,.5,1],#[.8],#[0.8,.85]
               'min_child_weight':[10],#[10,20,30,50,100,150,200], #[1,5,10],#[110,120,130]
               'max_depth': [2],#[2,4,6,10],
               'n_estimators':[151],
               'verbosity':[1],
             'gamma':[.01],#[1e-2,1e-3,1e-4,0,.1,.2,.3,.4,.5,1,3,5,10],
             'reg_alpha':[1],#[1e-5,1e-3,1e-1,1,1e1,1e2]

             }
model= xgb.XGBRegressor(n_jobs=1,random_state= random_seed,verbosity=1,silent=True)
clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                          verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
xgb_lab= xgb.XGBRegressor(**clf.best_params_,random_state= random_seed,silent=True)
X_lab= pd.DataFrame(X,columns= train.columns)
x_test_lab=pd.DataFrame(x_test,columns= train.columns)
xgb_lab.fit(X_lab,y)
```

```
(4209, 213) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    7.7s finished
```

```
{'colsample_bytree': 0.72, 'gamma': 0.01, 'learning_rate': 0.05, 'max_depth': 2,
'min_child_weight': 10, 'n_estimators': 151, 'reg_alpha': 1, 'subsample': 0.72, 'verbosity': 1}
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6033413557227743  +/-  0.02117461279342622
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   26.8s finished
```

In [0]:

```
cv3= KFold(5,True,random_seed)
y_mean= y_train.mean()

x_test= np.array(test)
X= np.array(train)
y= targets
print(X.shape,'X.shape')
parameters= {"n_estimators":[350],#range(50,600,50),
              "max_depth": [4],#[3,4,5,6,7,8,9,10],#list(range(2,10)),#4 is the best
              "min_samples_leaf": [10],#[3,4,5,6,7],
              "max_features": [.95],#[.95],
             'min_impurity_decrease':[1e-4],#[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]
              }
model= ExtraTreesRegressor(n_jobs=1,random_state= random_seed)

clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(X, y)

ids_test= test.ID.values
print(clf.best_params_)
et_lab= ExtraTreesRegressor(**clf.best_params_,random_state= random_seed,oob_score= True,bootstrap
= True)
et_lab.fit(X,y)
print('oob_score: ',et_lab.oob_score_)
```

```
(4209, 213) X.shape
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   28.3s finished
```

```
{'max_depth': 4, 'max_features': 0.95, 'min_impurity_decrease': 0.0001, 'min_samples_leaf': 10, 'n
_estimators': 350}
oob_score:  0.6008159716945323
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.5986564267097183  +/-  0.021069775384546185
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.1min finished
```

In [0]:

```
ridge= Ridge(random_state=random_seed,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(rf_label, xgb_lab,et_lab),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=cv)

cv_score=cross_val_score(stack,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
print(cv_score.mean(),' +/- ',cv_score.std())
```

```
stack.fit(X,y)
ids_test= test.ID
y_pred = stack.predict(x_test)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = y_pred
subm.to_csv('submission_xgb_rf_stack_ridge_label.csv', index=False)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed: 45.1min finished
```

```
0.6044614316044898  +/-  0.021252431513073353
```

LB(.55196,.55743),CV:.60446

cv got worse along with other feature interactions than before, so including the best model without any feature interactions for inference.

SVR WITH TSVD

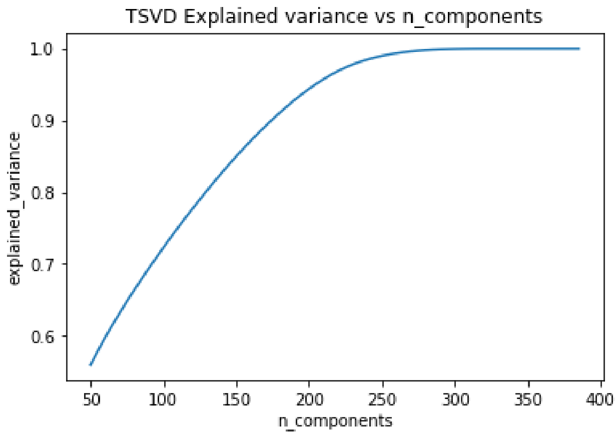
In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_hot.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_hot.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
print(train.shape)
```

```
(4209, 394)
```

In [0]:

```
x=[]
y=[]
SS= StandardScaler()
svd_train=SS.fit_transform(train)
svd_test= SS.transform(test)
for i in range(50,390,5):
  tsvd= TruncatedSVD(i,random_state= random_seed)
  _= tsvd.fit_transform(svd_train)
  x.append(i)
  y.append(sum(tsvd.explained_variance_ratio_))
plt.plot(x,y)
plt.ylabel('explained_variance')
plt.xlabel('n_components')
plt.title('TSVD Explained variance vs n_components')
tsvd= TruncatedSVD(230,random_state= random_seed)#elbow between 250,200
svd_train= tsvd.fit_transform(svd_train)
svd_test= tsvd.transform(svd_test)
```



In [0]:

```
for c in [.04]:#[.01,.02,.03,.04,.05,.06,.1,]:#[1e-3,1e-2,1e-1,1]:
  svr= SGDRegressor(loss= 'epsilon_insensitive',alpha= c,penalty= 'elasticnet',random_state= random
_seed)
  print(c)
  cv_score=cross_val_score(svr,svd_train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
  print(cv_score.mean(),' +/- ',cv_score.std())
  svr.fit(svd_train,targets)
  ids_test= test.ID
  y_pred = svr.predict(svd_test)
  subm = pd.DataFrame()
  subm['ID'] = ids_test
  subm['y'] = y_pred
  subm.to_csv('svr_tsvd.csv', index=False)
```

```
0.04
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:   10.4s finished
```

```
0.5567327921982091  +/-  0.02028322537034436
```

LB(.5021,.5157),CV:.5567

Kernel SVM

In [0]:

```
cv3= KFold(5,True,random_seed)
model= SVR()
parameters={'kernel':['poly'],'degree':[2,3,4,5],'C':[1e-1,1,1e1,1e2,1e3]}
clf = GridSearchCV(model, parameters, cv=cv3, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(svd_train, targets)
print(clf.best_score_,clf.best_params_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 10.4min finished
```

```
0.5192986687896454 {'C': 10.0, 'degree': 2, 'kernel': 'poly'}
```

In [0]:

```
from sklearn.svm import SVR

cv3= KFold(5,True,random_seed)
model= SVR()
parameters={'kernel':['rbf'],'C':[1e-3,1e-2,1e-1,1,1e1,1e2,1e3]}
clf = GridSearchCV(model, parameters, cv=cv, scoring='r2',
                         verbose=1,return_train_score= True,n_jobs=-1)
clf.fit(svd_train, targets)
print(clf.best_score_)
```

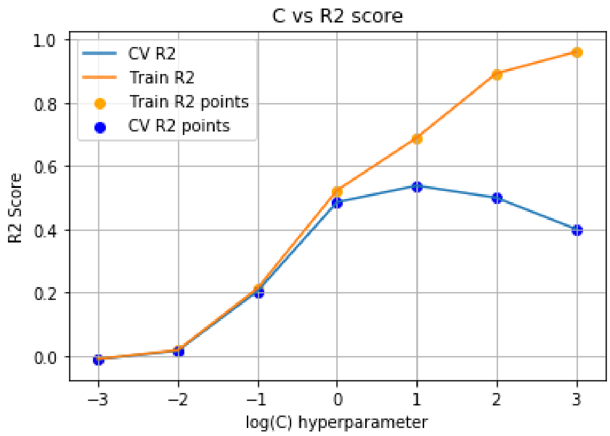
```
Fitting 15 folds for each of 7 candidates, totalling 105 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 105 out of 105 | elapsed: 10.9min finished
```

```
0.5368236466098156
```

In [0]:

```
cv_r1= clf.cv_results_['mean_train_score']
cv_r2 = clf.cv_results_['mean_test_score']
plt.plot(np.log10(parameters['C']),cv_r2,label= 'CV R2')
plt.title('C vs R2 score')
plt.xlabel('log(C) hyperparameter')
plt.ylabel('R2 Score')
plt.plot(np.log10(parameters['C']),cv_r1,label= 'Train R2')
plt.scatter(np.log10(parameters['C']),cv_r1,label= 'Train R2 points',color= 'orange')
plt.scatter(np.log10(parameters['C']),cv_r2,label= 'CV R2 points',color= 'blue')
plt.grid()
plt.legend()
plt.show()
```



In [0]:

```
svr= SVR(**{'C':10,'kernel':'rbf'})##using rbf
```

In [0]:

```
cv_score=cross_val_score(svr,svd_train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
print(cv_score.mean(),' +/- ',cv_score.std())
svr.fit(svd_train,targets)
ids_test= test.ID
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:   57.2s finished
```

```
0.5368236466098156  +/-  0.02336082835882139
```

Linear SVM worked better than kernels SVM.

SVR WITH SELECTKBEST

In [0]:

```
#Centering of data done before fitting to a  LinearModel
SS= StandardScaler()
ss_train=SS.fit_transform(train)
ss_test= SS.transform(test)
skbest= SelectKBest(f_regression,k=230)
skbest_train=skbest.fit_transform(ss_train,targets)
skbest_test=skbest.transform(ss_test)
svr= SGDRegressor(loss= 'epsilon_insensitive',alpha= .04,penalty= 'elasticnet',random_state= random
_seed)
svr.fit(skbest_train,targets)
cv_score=cross_val_score(svr,skbest_train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
print(cv_score.mean(),' +/- ',cv_score.std())
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
```

```
0.5633626352641173  +/-  0.02062498588462952
```

```
[Parallel(n_jobs=-1)]: Done  15 out of  15 | elapsed:    9.3s finished
```

In [0]:

```
ids_test= test.ID
y_pred = svr.predict(skbest_test)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = y_pred
subm.to_csv('skbest_150.csv', index=False)
```

LB(0.50797,0.51673),CV:.5639

Bayesian Optimisation

Label Encoding

RandomForest BayesianTuning

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_label.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_label.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
#https://www.kaggle.com/btyuhas/bayesian-optimization-with-xgboost
def rf_evaluate(n_estimators,max_depth,min_samples_leaf,max_features,min_impurity_decrease):
  params={
      'n_estimators':int(n_estimators),
          'max_depth':int(max_depth),
          'min_samples_leaf':int(min_samples_leaf),
          'min_impurity_decrease':min_impurity_decrease,
          'max_features':max_features
          }
  rf_label= RandomForestRegressor(**params)
  cv_score=cross_val_score(rf_label,train,y_train,scoring='r2',cv= cv3,verbose=1,n_jobs=1)
  return cv_score.mean()
```

In [0]:

```
rf_bo= BayesianOptimization(rf_evaluate,{'n_estimators':(550,650),
          'max_depth':(1,5),
          'min_samples_leaf':(1,7),
          'min_impurity_decrease':(.001,1),
          'max_features':(.5,1)
          })
rf_bo.maximize(init_points=10, n_iter=50, acq='ei')
```

```
|   iter    |  target   | max_depth | max_fe... | min_im... | min_sa... | n_esti... |
-------------------------------------------------------------------------------------
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  1        |  0.5985   |  4.435    |  0.6865   |  0.5557   |  1.367    |  617.7    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   21.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  2         |  0.5965   |  3.487    |  0.5462   |  0.07859  |  4.351    |  578.8    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  3         |  0.3906   |  1.377    |  0.7204   |  0.8453   |  4.818    |  638.2    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   21.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  4         |  0.5587   |  2.553    |  0.8821   |  0.9798   |  2.795    |  558.1    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  5         |  0.5975   |  3.71     |  0.5985   |  0.1878   |  1.793    |  630.1    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  6         |  0.3888   |  1.823    |  0.9155   |  0.08812  |  3.893    |  621.7    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   21.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  7         |  0.5604   |  2.816    |  0.7506   |  0.191    |  3.589    |  606.5    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   39.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  8         |  0.5972   |  4.639    |  0.9006   |  0.6443   |  1.855    |  600.5    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  9         |  0.3889   |  1.892    |  0.8648   |  0.8162   |  6.142    |  635.9    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.8s finished
```

```
|  10        |  0.3986   |  1.94     |  0.5018   |  0.2839   |  4.127    |  555.8    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.9s finished
```

```
|  11        |  0.6019   |  4.763    |  0.5455   |  0.006589 |  1.112    |  649.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.6s finished
```

```
|  12        |  0.3893   |  1.0      |  1.0      |  1.0      |  1.0      |  570.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.3s finished
```

```
|  13        |  0.5958   |  5.0      |  1.0      |  1.0      |  7.0      |  566.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   42.9s finished
```

```
|  14        |  0.596    |  5.0      |  1.0      |  1.0      |  7.0      |  590.8    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.7s finished
```

```
|  15        |  0.5997   |  4.951    |  0.7691   |  0.456    |  1.085    |  551.7    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.1s finished
```

```
|  16        |  0.6037   |  5.0      |  0.5      |  0.001    |  1.0      |  562.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   46.0s finished
```

```
|  17        |  0.5958   |  5.0      |  1.0      |  1.0      |  7.0      |  650.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   41.7s finished
```

```
|  18        |  0.5953   |  5.0      |  1.0      |  1.0      |  1.0      |  584.9    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.2s finished
```

```
|  19        |  0.6023   |  5.0      |  0.5      |  0.001    |  7.0      |  609.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.9s finished
```

```
|  20        |  0.6034   |  5.0      |  0.5      |  0.001    |  1.0      |  638.8    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.7s finished
```

```
|  21        |  0.5959   |  5.0      |  1.0      |  1.0      |  7.0      |  579.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.5s finished
```

```
|  22        |  0.3993   |  1.0      |  0.5      |  0.001    |  1.0      |  591.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   57.1s finished
```

```
|  23        |  0.6006   |  5.0      |  1.0      |  0.001    |  1.0      |  609.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   45.1s finished
```

```
|  24        |  0.5955   |  5.0      |  1.0      |  1.0      |  1.0      |  626.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.3s finished
```

```
|  25        |  0.6029   |  5.0      |  0.5      |  0.001    |  7.0      |  600.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   38.0s finished
```

```
|  26        |  0.596    |  4.975    |  0.9703   |  0.9464   |  6.898    |  556.9    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.6s finished
```

```
|  27        |  0.5952   |  5.0       |  1.0       |  1.0       |  1.0       |  557.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.3s finished
```

```
|  28        |  0.5971   |  4.726     |  0.814     |  0.7847    |  6.196     |  550.1     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.5s finished
```

```
|  29        |  0.3997   |  1.0       |  0.5       |  1.0       |  1.0       |  650.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   11.9s finished
```

```
|  30        |  0.3903   |  1.074     |  0.7483    |  0.975     |  6.985     |  583.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.4s finished
```

```
|  31        |  0.5962   |  5.0       |  0.5       |  1.0       |  1.0       |  576.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.7s finished
```

```
|  32        |  0.6026   |  5.0       |  0.5       |  0.001     |  7.0       |  644.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.0s finished
```

```
|  33        |  0.6026   |  5.0       |  0.5       |  0.001     |  6.047     |  584.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   23.8s finished
```

```
|  34        |  0.5958   |  5.0       |  0.5       |  1.0       |  4.259     |  561.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   55.7s finished
```

```
|  35        |  0.6012   |  5.0       |  1.0       |  0.001     |  1.0       |  579.8     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.9s finished
```

```
|  36        |  0.6025   |  5.0       |  0.5       |  0.001     |  7.0       |  627.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.7s finished
```

```
|  37        |  0.6036   |  5.0       |  0.5       |  0.001     |  1.0       |  593.3     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.1s finished
```

```
|  38        |  0.5955   |  5.0       |  0.5       |  1.0       |  7.0       |  616.6     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.0s finished
```

```
|   39        |   0.6024   |   5.0       |   0.5       |   0.001     |   7.0       |   573.6     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.5s finished
```

```
|   40        |   0.3892   |   1.0       |   1.0       |   1.0       |   1.0       |   550.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.9s finished
```

```
|   41        |   0.6037   |   5.0       |   0.5       |   0.001     |   1.0       |   632.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   48.3s finished
```

```
|   42        |   0.5954   |   5.0       |   1.0       |   1.0       |   1.0       |   644.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.0s finished
```

```
|   43        |   0.5962   |   5.0       |   0.5       |   1.0       |   1.0       |   612.1     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.8s finished
```

```
|   44        |   0.5954   |   5.0       |   1.0       |   1.0       |   1.0       |   550.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.0s finished
```

```
|   45        |   0.6032   |   5.0       |   0.5       |   0.001     |   1.0       |   622.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   44.0s finished
```

```
|   46        |   0.5958   |   5.0       |   1.0       |   1.0       |   5.334     |   605.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.6s finished
```

```
|   47        |   0.6013   |   5.0       |   0.5       |   0.001     |   3.892     |   614.7     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.9s finished
```

```
|   48        |   0.6034   |   5.0       |   0.5       |   0.001     |   1.0       |   604.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   52.6s finished
```

```
|   49        |   0.6029   |   5.0       |   1.0       |   0.001     |   7.0       |   561.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   54.2s finished
```

```
|   50        |   0.6002   |   5.0       |   1.0       |   0.001     |   3.759     |   574.7     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.5s finished
```

```
|  51        |  0.5957   |  5.0      |  0.5      |  1.0      |  4.239    |  630.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

```
|  52        |  0.6016   |  5.0      |  1.0      |  0.001    |  4.183    |  647.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.4s finished
```

```
|  53        |  0.6029   |  5.0      |  1.0      |  0.001    |  7.0      |  550.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.5s finished
```

```
|  54        |  0.596    |  5.0      |  0.5      |  1.0      |  4.505    |  595.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.3s finished
```

```
|  55        |  0.603    |  5.0      |  0.5      |  0.001    |  1.0      |  644.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.6s finished
```

```
|  56        |  0.389    |  1.063    |  0.9292   |  0.05271  |  1.065    |  562.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.2s finished
```

```
|  57        |  0.5958   |  5.0      |  0.5      |  1.0      |  3.456    |  650.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.2s finished
```

```
|  58        |  0.3892   |  1.221    |  0.9858   |  0.4283   |  6.881    |  600.7    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   55.0s finished
```

```
|  59        |  0.6014   |  5.0      |  1.0      |  0.001    |  4.114    |  590.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  60        |  0.6029   |  5.0      |  1.0      |  0.001    |  7.0      |  639.1    |
=====================================================================================
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

In [0]:

```
rf_bo.max
```

Out[0]:

```
{'params': {'max_depth': 5.0,
  'max_features': 0.5,
  'min_impurity_decrease': 0.001,
  'min_samples_leaf': 1.0,
  'n_estimators': 632.1540758277154},
 'target': 0.6037271160271158}
```

CV improved by .014 than the manually tuned model.

In [0]:

```
rf_label= RandomForestRegressor(**{'max_depth': 5,'max_features': 0.5,'min_impurity_decrease': 0.00
1,'min_samples_leaf': 1,'n_estimators': 632},random_state= random_seed,oob_score= True)
rf_label.fit(train,targets)
print('oob_score: ',rf_label.oob_score_)
cv_score=cross_val_score(rf_label,train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = rf_label.predict(test)
subm.to_csv('rf_5folds_label.csv', index=False)
```

```
oob_score:  0.604051459344535
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6032145685770464  +/-  0.021977434185266955
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.8min finished
```

LB(0.55001,0.55685),CV:.6032

XGBoost Bayesian Tuning

In [0]:

```
def
xgb_evaluate(n_estimators,max_depth,subsample,colsample_bytree,gamma,reg_alpha,min_child_weight):
  params={
      'n_estimators':int(n_estimators),
          'max_depth':int(max_depth),
          'min_child_weight':int(min_child_weight),
          'gamma':gamma,
          'subsample':subsample,
          'colsample_bytree':colsample_bytree,
          'reg_alpha':reg_alpha
          }
  xgb_label= xgb.XGBRegressor(**params,silent= True,random_state= random_seed,learning_rate= .05)
  cv_score=cross_val_score(xgb_label,train,y_train,scoring='r2',cv= cv3,verbose=1,n_jobs=1)
  return cv_score.mean()
```

In [0]:

```
xgb_bo= BayesianOptimization(xgb_evaluate,{
               'subsample': (.5,1),
               'colsample_bytree': (.5,1),
               'min_child_weight':(1,10),
               'max_depth': (1,8),
               'n_estimators':(180,230),
             'gamma':(.001,100),
             'reg_alpha':(.001,100)
          })
xgb_bo.maximize(init_points=10, n_iter=50, acq='ei')
```

```
|   iter    |  target   | colsam... |   gamma   | max_depth | min_ch... | n_esti... | reg_alpha | s
ubsample |
-------------------------------------------------------------------------------------------------
------
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  1        |  0.5892   |  0.6336   |  12.6     |  6.086    |  4.385    |  227.2    |  21.19    |
.7971    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  2         |  0.6027    |  0.8023    |  60.97     |  2.006     |  4.175     |  226.7     |  15.42     |
.8826    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.9s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  3         |  0.6018    |  0.5166    |  71.72     |  3.029     |  3.501     |  203.7     |  70.0      |
.6677    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  4         |  0.5989    |  0.6265    |  64.81     |  5.567     |  2.728     |  219.9     |  67.55     |
.8275    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  5         |  0.5938    |  0.5846    |  1.26      |  7.146     |  7.544     |  228.4     |  62.83     |
.5677    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   16.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  6         |  0.6003    |  0.6394    |  35.39     |  4.78      |  1.178     |  207.8     |  46.5      |
.8968    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   11.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  7         |  0.604     |  0.6346    |  21.29     |  2.384     |  5.235     |  195.2     |  58.0      |
.6401    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  8         |  0.6027    |  0.7818    |  23.79     |  3.765     |  8.63      |  181.0     |  47.13     |
.969     |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  9         |  0.5784    |  0.5693    |  89.69     |  1.576     |  2.885     |  182.3     |  40.92     |
.5005    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.1s finished
```

```
|  10        |  0.5858    |  0.8783    |  40.8      |  7.852     |  1.908     |  216.4     |  23.76     |
.9637    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.6s finished
```

```
|  11        |  0.5835    |  0.9763    |  47.78     |  1.214     |  9.072     |  227.5     |  53.51     |
.8326    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.5s finished
```

```
|  12        |  0.6019   |  0.6102   |  5.624    |  2.512    |  1.314    |  180.4    |  99.25    |
.5681   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.1s finished
```

```
|  13        |  0.6034   |  0.7698   |  97.62    |  2.857    |  2.117    |  229.7    |  96.42    |
.6963   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   11.9s finished
```

```
|  14        |  0.6019   |  0.7695   |  94.67    |  2.482    |  1.525    |  229.7    |  6.43     |
.9599   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.7s finished
```

```
|  15        |  0.5787   |  0.6772   |  8.024    |  1.526    |  1.097    |  181.2    |  1.452    |
.7751   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.6s finished
```

```
|  16        |  0.6035   |  0.5097   |  4.408    |  2.019    |  2.597    |  182.1    |  59.97    |
.9472   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.6s finished
```

```
|  17        |  0.6023   |  0.6405   |  42.99    |  3.603    |  1.121    |  185.7    |  99.18    |
.6969   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.5s finished
```

```
|  18        |  0.5776   |  0.7595   |  37.93    |  1.188    |  2.061    |  182.2    |  57.28    |
.6758   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.2s finished
```

```
|  19        |  0.6002   |  0.7327   |  93.33    |  7.907    |  3.65     |  202.2    |  98.12    |
.9228   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.2s finished
```

```
|  20        |  0.5794   |  0.5706   |  0.9547   |  1.487    |  9.848    |  188.6    |  33.21    |
.5738   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   18.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  21        |  0.5982   |  0.5291   |  16.23    |  7.943    |  7.948    |  186.7    |  84.5     |
.9055   |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.5s finished
```

```
|  22        |  0.6036   |  0.7815   |  98.67    |  2.387    |  4.221    |  227.4    |  97.73    |
.879    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.4s finished
```

```
|  23       |  0.5791   |  0.6099   |  83.43    |  1.09     |  9.286    |  186.8    |  0.3443   |
.815     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.9s finished
```

```
|  24       |  0.5985   |  0.7248   |  18.52    |  5.707    |  1.159    |  216.4    |  99.97    |
.9176    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.6s finished
```

```
|  25       |  0.5816   |  0.6158   |  97.66    |  1.545    |  9.121    |  220.7    |  52.51    |
.8081    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.7s finished
```

```
|  26       |  0.5787   |  0.5554   |  80.12    |  1.028    |  1.504    |  209.9    |  96.79    |
.5943    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.2s finished
```

```
|  27       |  0.5938   |  0.9736   |  79.33    |  5.752    |  9.236    |  228.9    |  0.3057   |
.98      |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.9s finished
```

```
|  28       |  0.5993   |  0.7212   |  99.3     |  7.777    |  9.722    |  181.6    |  80.46    |
.6987    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.3s finished
```

```
|  29       |  0.5834   |  0.5986   |  76.26    |  1.949    |  1.467    |  229.1    |  5.231    |
.9654    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.6s finished
```

```
|  30       |  0.5929   |  0.8395   |  95.83    |  7.63     |  9.743    |  226.4    |  22.84    |
.9391    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   19.4s finished
```

```
|  31       |  0.6005   |  0.7881   |  99.7     |  3.608    |  9.611    |  227.2    |  5.122    |
.7105    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   36.4s finished
```

```
|  32       |  0.5926   |  0.8082   |  0.7876   |  7.801    |  2.292    |  206.1    |  68.74    |
.5303    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.4s finished
```

```
|  33       |  0.5846   |  0.7228   |  37.49    |  1.396    |  8.855    |  229.2    |  4.313    |
```

```
.8347   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.4s finished
```

```
|  34       |  0.5921   |  0.6071   |  60.34    |  7.433    |  9.581    |  226.1    |  25.84    |
.9443   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.1s finished
```

```
|  35       |  0.5843   |  0.8471   |  7.616    |  1.06     |  1.261    |  229.1    |  38.99    |
.7884   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   36.6s finished
```

```
|  36       |  0.5963   |  0.8913   |  99.31    |  7.551    |  1.456    |  193.4    |  72.55    |
.6073   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.9s finished
```

```
|  37       |  0.6009   |  0.593    |  98.65    |  4.465    |  1.144    |  181.1    |  98.01    |
.5624   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.0s finished
```

```
|  38       |  0.5807   |  0.8441   |  0.7711   |  7.243    |  8.532    |  227.2    |  0.8451   |
.7384   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.7s finished
```

```
|  39       |  0.5815   |  0.8872   |  30.68    |  7.528    |  9.641    |  180.6    |  0.1315   |
.6032   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.8s finished
```

```
|  40       |  0.5994   |  0.8675   |  1.143    |  5.097    |  8.111    |  229.0    |  99.9     |
.933    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   17.4s finished
```

```
|  41       |  0.6013   |  0.575    |  45.94    |  7.828    |  9.336    |  180.5    |  98.13    |
.9436   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   45.5s finished
```

```
|  42       |  0.5946   |  0.9551   |  96.79    |  7.672    |  7.981    |  228.7    |  96.6     |
.5024   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.4s finished
```

```
|  43       |  0.5984   |  0.8057   |  98.84    |  7.275    |  1.127    |  223.7    |  99.44    |
.7457   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.5s finished
```

```
|  44        |  0.5938   |  0.9473   |  8.323    |  7.414    |  3.426    |  181.9    |  55.68    |
.843    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.2s finished
```

```
|  45        |  0.5772   |  0.6087   |  10.69    |  1.208    |  9.807    |  196.3    |  98.6     |
.9554   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   39.3s finished
```

```
|  46        |  0.5925   |  0.9551   |  7.255    |  7.795    |  3.528    |  230.0    |  87.1     |
.8332   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.5s finished
```

```
|  47        |  0.5839   |  0.8432   |  61.96    |  1.007    |  8.901    |  210.5    |  17.95    |
.538    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.4s finished
```

```
|  48        |  0.5907   |  0.86     |  99.72    |  7.679    |  1.781    |  180.3    |  9.675    |
.718    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.4s finished
```

```
|  49        |  0.6044   |  0.9109   |  99.26    |  2.084    |  2.074    |  227.9    |  83.82    |
.7265   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.8s finished
```

```
|  50        |  0.6028   |  0.5492   |  99.06    |  6.724    |  6.54     |  189.0    |  99.15    |
.9442   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.7s finished
```

```
|  51        |  0.5996   |  0.9357   |  82.71    |  6.297    |  3.647    |  180.7    |  94.98    |
.8733   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   18.3s finished
```

```
|  52        |  0.599    |  0.5323   |  16.45    |  7.839    |  2.743    |  182.1    |  99.84    |
.8448   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   17.1s finished
```

```
|  53        |  0.6002   |  0.5719   |  51.32    |  6.665    |  1.188    |  198.8    |  84.13    |
.9762   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.8s finished
```

```
|  54        |  0.5761   |  0.8306   |  9.145    |  1.49     |  6.036    |  180.4    |  73.93    |
```

```
.897    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.3s finished
```

```
|  55        |  0.6021   |  0.5123   |  23.03    |  3.06     |  2.216    |  183.7    |  29.93    |
.8721    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   34.3s finished
```

```
|  56        |  0.5935   |  0.786    |  2.71     |  7.475    |  1.569    |  213.8    |  99.32    |
.6773    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   18.8s finished
```

```
|  57        |  0.5974   |  0.5327   |  29.3     |  7.832    |  9.723    |  207.5    |  73.58    |
.9775    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   22.1s finished
```

```
|  58        |  0.5935   |  0.7351   |  97.86    |  5.654    |  3.035    |  229.9    |  12.27    |
.9663    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.3s finished
```

```
|  59        |  0.5784   |  0.5286   |  14.97    |  1.607    |  2.017    |  194.4    |  51.69    |
.8085    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  60        |  0.5956   |  0.5618   |  35.54    |  7.531    |  9.408    |  182.1    |  27.43    |
.9801    |
===================================================================================================
======
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   17.2s finished
```

In [0]:

```
xgb_bo.max
```

Out[0]:

```
{'params': {'colsample_bytree': 0.910880732284358,
  'gamma': 99.25580698505343,
  'max_depth': 2.08381908163798,
  'min_child_weight': 2.073601162878024,
  'n_estimators': 227.9262239473573,
  'reg_alpha': 83.82477076460415,
  'subsample': 0.7265495851767015},
 'target': 0.6044287226660563}
```

In [0]:

```
xgb_lab= xgb.XGBRegressor(**{'colsample_bytree': 0.910880732284358,
  'gamma': 99.25580698505343,
  'max_depth': 2,
  'min_child_weight': 2,
  'n_estimators': 227.,
  'reg_alpha': 83.82477076460415,
  'subsample': 0.7265495851767015},random_state= random_seed,silent=True)
xgb_lab.fit(train,targets)
cv_score=cross_val_score(xgb_lab,train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=1)
```

```
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = xgb_lab.predict(test)
subm.to_csv('xgb_meanenc.csv', index=False)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6002236276093992  +/-  0.021061742036034777
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   41.4s finished
```

LB(0.54865,0.54894),CV:.60022

ExtraTrees Bayesian Tuning

In [0]:

```
def et_evaluate(n_estimators,max_depth,min_samples_leaf,max_features,min_impurity_decrease):
  params={
      'n_estimators':int(n_estimators),
          'max_depth':int(max_depth),
          'min_samples_leaf':int(min_samples_leaf),
          'min_impurity_decrease':min_impurity_decrease,
          'max_features':max_features
          }
  et_label= ExtraTreesRegressor(**params,random_state= random_seed)
  cv_score=cross_val_score(et_label,train,y_train,scoring='r2',cv= cv3,verbose=1,n_jobs=1)
  return cv_score.mean()
```

In [0]:

```
et_bo= BayesianOptimization(et_evaluate,{'n_estimators':(550,650),
          'max_depth':(1,5),
          'min_samples_leaf':(1,7),
          'min_impurity_decrease':(.001,1),
          'max_features':(.5,1)
          })
et_bo.maximize(init_points=10, n_iter=50, acq='ei')
```

```
|   iter    |  target   | max_depth | max_fe... | min_im... | min_sa... | n_esti... |
-------------------------------------------------------------------------------------
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  1        |  0.5992   |  4.185    |  0.6748   |  0.4425   |  2.53     |  602.4    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  2        |  0.5957   |  3.368    |  0.787    |  0.5012   |  6.157    |  636.2    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  3        |  0.3883   |  1.845    |  0.8563   |  0.07371  |  4.911    |  594.1    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  4        |  0.389    |  1.992    |  0.9417   |  0.9656   |  2.672    |  569.0    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.2s finished
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

| 5 | 0.3884 | 1.226 | 0.8027 | 0.2493 | 5.282 | 619.3 |

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

| 6 | 0.596 | 4.149 | 0.5498 | 0.8136 | 3.648 | 581.1 |

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   19.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

| 7 | 0.5553 | 2.77 | 0.6679 | 0.0408 | 5.918 | 612.2 |

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

| 8 | 0.3936 | 1.155 | 0.6397 | 0.2847 | 5.678 | 568.3 |

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

| 9 | 0.5521 | 2.228 | 0.7094 | 0.3868 | 5.991 | 622.5 |

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.3s finished
```

| 10 | 0.597 | 3.041 | 0.7069 | 0.05671 | 3.982 | 563.1 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.4s finished
```

| 11 | 0.6017 | 5.0 | 0.5 | 0.001 | 7.0 | 550.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.0s finished
```

| 12 | 0.5957 | 5.0 | 0.5 | 1.0 | 7.0 | 626.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   17.1s finished
```

| 13 | 0.3892 | 1.0 | 1.0 | 0.001 | 1.0 | 650.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.1s finished
```

| 14 | 0.6017 | 5.0 | 0.5 | 0.001 | 7.0 | 606.4 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.5s finished
```

| 15 | 0.4008 | 1.0 | 0.5 | 0.001 | 1.0 | 552.5 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.4s finished
```

| 16 | 0.6017 | 5.0 | 0.5 | 0.001 | 7.0 | 559.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.7s finished
```

```
|   17        |   0.4008  |   1.0     |   0.5     |   0.001   |   1.0     |   632.5   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.9s finished

|   18        |   0.602   |   5.0     |   0.5     |   0.001   |   1.0     |   586.0   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   16.7s finished

|   19        |   0.3892  |   1.0     |   1.0     |   0.001   |   7.0     |   630.5   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.0s finished

|   20        |   0.6017  |   5.0     |   0.5     |   0.001   |   7.0     |   643.8   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.1s finished

|   21        |   0.602   |   5.0     |   0.5     |   0.001   |   1.0     |   609.4   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.7s finished

|   22        |   0.5962  |   5.0     |   0.5     |   1.0     |   1.0     |   640.3   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.6s finished

|   23        |   0.6017  |   5.0     |   0.5     |   0.001   |   7.0     |   620.7   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   56.3s finished

|   24        |   0.5949  |   5.0     |   1.0     |   0.001   |   1.0     |   561.1   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.5s finished

|   25        |   0.6018  |   5.0     |   0.5     |   0.001   |   7.0     |   583.1   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   47.9s finished

|   26        |   0.5948  |   5.0     |   1.0     |   1.0     |   6.522   |   639.2   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.2s finished

|   27        |   0.5957  |   5.0     |   0.5     |   1.0     |   5.795   |   609.9   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.0s finished

|   28        |   0.5997  |   5.0     |   0.5     |   0.001   |   4.055   |   637.3   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.9s finished

|   29        |   0.4007  |   1.0     |   0.5     |   1.0     |   7.0     |   642.3   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   28.1s finished
```

| 30 | 0.5957 | 5.0 | 0.5 | 1.0 | 7.0 | 650.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.0s finished
```

| 31 | 0.602 | 5.0 | 0.5 | 0.001 | 1.0 | 580.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.4s finished
```

| 32 | 0.602 | 5.0 | 0.5 | 0.001 | 1.0 | 604.8 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.6s finished
```

| 33 | 0.5962 | 5.0 | 0.5 | 1.0 | 1.0 | 624.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.9s finished
```

| 34 | 0.6021 | 5.0 | 0.5 | 0.001 | 1.0 | 650.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   42.7s finished
```

| 35 | 0.5948 | 5.0 | 1.0 | 1.0 | 5.254 | 563.4 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.8s finished
```

| 36 | 0.5957 | 5.0 | 0.5 | 1.0 | 5.028 | 623.6 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.3s finished
```

| 37 | 0.4012 | 1.0 | 0.5 | 1.0 | 1.0 | 604.8 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

| 38 | 0.5949 | 5.0 | 1.0 | 0.001 | 1.0 | 598.4 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   46.2s finished
```

| 39 | 0.5948 | 5.0 | 1.0 | 1.0 | 3.323 | 583.1 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.4s finished
```

| 40 | 0.3892 | 1.0 | 1.0 | 1.0 | 7.0 | 550.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.1s finished
```

| 41 | 0.5997 | 5.0 | 0.5 | 0.001 | 4.097 | 562.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   34.1s finished
```

```
|  42        |  0.5997   |  5.0       |  0.5       |  0.001     |  3.437     |  602.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   44.4s finished
```

```
|  43        |  0.5948   |  4.189     |  1.0       |  1.0       |  3.922     |  559.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

```
|  44        |  0.5959   |  5.0       |  1.0       |  0.001     |  4.696     |  609.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.9s finished
```

```
|  45        |  0.5963   |  4.193     |  0.5168    |  0.7429    |  6.896     |  636.3     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.1min finished
```

```
|  46        |  0.5949   |  5.0       |  1.0       |  0.001     |  1.0       |  645.3     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   46.5s finished
```

```
|  47        |  0.5948   |  5.0       |  1.0       |  1.0       |  1.0       |  601.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   48.5s finished
```

```
|  48        |  0.5948   |  5.0       |  1.0       |  1.0       |  1.0       |  634.1     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   47.4s finished
```

```
|  49        |  0.6018   |  4.934     |  0.9841    |  0.2432    |  6.209     |  579.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   34.9s finished
```

```
|  50        |  0.6017   |  5.0       |  0.5       |  0.001     |  7.0       |  611.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   49.8s finished
```

```
|  51        |  0.6011   |  4.488     |  0.9016    |  0.07362   |  6.964     |  624.5     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   41.6s finished
```

```
|  52        |  0.5997   |  4.984     |  0.6923    |  0.05288   |  5.058     |  647.6     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.1min finished
```

```
|  53        |  0.6011   |  5.0       |  1.0       |  0.001     |  7.0       |  650.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.7s finished
```

```
|  54        |  0.5997   |  5.0      |  0.5      |  0.001    |  4.16     |  581.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.0s finished
```

```
|  55        |  0.5956   |  4.143    |  0.5      |  1.0      |  7.0      |  623.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.9s finished
```

```
|  56        |  0.5961   |  4.964    |  0.7042   |  0.7613   |  6.963    |  580.9    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.8s finished
```

```
|  57        |  0.5265   |  2.646    |  0.963    |  0.1142   |  1.21     |  582.8    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   38.6s finished
```

```
|  58        |  0.5931   |  3.954    |  1.0      |  0.001    |  5.55     |  561.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   36.8s finished
```

```
|  59        |  0.5997   |  5.0      |  0.5      |  0.001    |  3.364    |  626.6    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  60        |  0.5541   |  2.187    |  0.6568   |  0.9883   |  1.595    |  561.5    |
=========================================================================================
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   19.8s finished
```

In [0]:

```
et_bo.max
```

Out[0]:

```
{'params': {'max_depth': 5.0,
  'max_features': 0.5,
  'min_impurity_decrease': 0.001,
  'min_samples_leaf': 1.0,
  'n_estimators': 650.0},
 'target': 0.6020747888106721}
```

In [0]:

```
et_lab= ExtraTreesRegressor(**{'max_depth': 5,
  'max_features': 0.5,
  'min_impurity_decrease': 0.001,
  'min_samples_leaf': 1,
  'n_estimators': 650},random_state= random_seed,oob_score= True,bootstrap= True)
et_lab.fit(train,targets)
print('oob_score: ',et_lab.oob_score_)
cv_score=cross_val_score(et_lab,train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = et_lab.predict(test)
subm.to_csv('et_5folds_label.csv', index=False)
```

```
oob_score:  0.6025695090342889
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6012477051379436  +/-  0.021889636974332664
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.5min finished
```

LB(0.54710,0.55228),CV:.6012

In [0]:

```
train.shape
```

Out[0]:

```
(4209, 210)
```

Stacking the Bayesian Tuned Models

In [0]:

```
ridge= Ridge(random_state=random_seed,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(rf_label, xgb_lab,et_lab),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=cv)

'''cv_score=cross_val_score(stack,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
print(cv_score.mean(),' +/- ',cv_score.std())'''
X= np.array(train)
y= targets
x_test= np.array(test)
stack.fit(X,y)
ids_test= test.ID
y_pred = stack.predict(x_test)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = y_pred
subm.to_csv('submission_xgb_rf_stack_ridge_label.csv', index=False)
```

LB(0.55211,0.55630)

MeanEncoding

RandomForest BayesianTuning

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_meanenc.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_meanenc.csv'
)

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
cv3= KFold(5,True,random_seed)
def rf_evaluate(n_estimators,max_depth,min_samples_leaf,max_features,min_impurity_decrease):
  params={
      'n_estimators':int(n_estimators),
          'max_depth':int(max_depth),
```

```
            'min_samples_leaf':int(min_samples_leaf),
            'min_impurity_decrease':min_impurity_decrease,
            'max_features':max_features
            }
    rf_label= RandomForestRegressor(**params)
    cv_score=cross_val_score(rf_label,train,y_train,scoring='r2',cv= cv3,verbose=1,n_jobs=1)
    return cv_score.mean()
```

In [12]:

```
rf_bo= BayesianOptimization(rf_evaluate,{'n_estimators':(550,650),
            'max_depth':(1,5),
            'min_samples_leaf':(1,7),
            'min_impurity_decrease':(.001,1),
            'max_features':(.5,1)
            })
rf_bo.maximize(init_points=10, n_iter=50, acq='ei')
```

```
|   iter    |  target   | max_depth | max_fe... | min_im... | min_sa... | n_esti... |
-------------------------------------------------------------------------------------

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  1        |  0.5926   |  2.367    |  0.7997   |  0.03869  |  5.589    |  579.3    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  2        |  0.5989   |  4.046    |  0.8648   |  0.7502   |  2.946    |  560.3    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   43.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  3        |  0.6045   |  4.168    |  0.9995   |  0.1842   |  3.764    |  577.0    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  4        |  0.5987   |  4.326    |  0.7789   |  0.7738   |  6.066    |  616.3    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  5        |  0.4553   |  1.312    |  0.6555   |  0.5867   |  4.518    |  572.1    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   16.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  6        |  0.5871   |  2.414    |  0.5563   |  0.7508   |  5.452    |  602.0    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  7        |  0.5995   |  4.403    |  0.6687   |  0.6872   |  6.463    |  610.7    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  8        |  0.5993   |  3.169    |  0.6846   |  0.6125   |  5.459    |  615.1    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|   9         |   0.602   |   3.293   |   0.8999  |   0.201   |   3.885   |   622.7   |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.0s finished
```

```
|   10        |   0.602   |   3.517   |   0.9533  |   0.1389  |   1.36    |   606.6   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.0s finished
```

```
|   11        |   0.6053  |   5.0     |   1.0     |   0.001   |   7.0     |   550.0   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.5s finished
```

```
|   12        |   0.4476  |   1.0     |   0.5     |   0.001   |   1.0     |   650.0   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   54.2s finished
```

```
|   13        |   0.6051  |   5.0     |   1.0     |   0.001   |   7.0     |   586.9   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.2s finished
```

```
|   14        |   0.6053  |   5.0     |   1.0     |   0.001   |   1.0     |   550.0   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.9s finished
```

```
|   15        |   0.6058  |   5.0     |   0.5     |   0.001   |   1.0     |   583.3   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.6s finished
```

```
|   16        |   0.6045  |   5.0     |   1.0     |   0.001   |   3.637   |   557.4   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   36.2s finished
```

```
|   17        |   0.5973  |   5.0     |   1.0     |   1.0     |   7.0     |   580.4   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   58.7s finished
```

```
|   18        |   0.605   |   5.0     |   1.0     |   0.001   |   7.0     |   636.6   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.5s finished
```

```
|   19        |   0.5969  |   5.0     |   1.0     |   1.0     |   1.0     |   559.7   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.1s finished
```

```
|   20        |   0.4709  |   1.0     |   1.0     |   1.0     |   2.958   |   550.0   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   16.1s finished
```

```
|   21        |   0.4712  |   1.0     |   1.0     |   0.001   |   7.0     |   629.9   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.4s finished
```

| 22 | 0.6055 | 5.0 | 0.5 | 0.001 | 1.0 | 634.8 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.5s finished
```

| 23 | 0.6059 | 5.0 | 0.5 | 0.001 | 1.0 | 620.1 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   41.1s finished
```

| 24 | 0.5972 | 5.0 | 1.0 | 1.0 | 7.0 | 650.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.1s finished
```

| 25 | 0.6049 | 5.0 | 0.5 | 0.001 | 6.245 | 562.5 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.3s finished
```

| 26 | 0.4711 | 1.0 | 1.0 | 0.001 | 6.926 | 607.1 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   23.2s finished
```

| 27 | 0.5963 | 5.0 | 0.5 | 1.0 | 2.917 | 600.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   23.4s finished
```

| 28 | 0.5962 | 5.0 | 0.5 | 1.0 | 2.31 | 611.6 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.7s finished
```

| 29 | 0.605 | 5.0 | 0.5 | 0.001 | 7.0 | 643.5 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.3s finished
```

| 30 | 0.6047 | 5.0 | 0.5 | 0.001 | 4.227 | 580.6 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.1s finished
```

| 31 | 0.4712 | 1.0 | 1.0 | 0.001 | 1.0 | 589.9 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.5s finished
```

| 32 | 0.6049 | 5.0 | 0.5 | 0.001 | 7.0 | 601.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   54.6s finished
```

| 33 | 0.6051 | 5.0 | 1.0 | 0.001 | 1.0 | 579.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.0s finished
```

| 34 | 0.5972 | 5.0 | 1.0 | 1.0 | 7.0 | 559.4 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.9s finished
```

| 35 | 0.4487 | 1.0 | 0.5 | 1.0 | 1.0 | 620.4 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.2s finished
```

| 36 | 0.605 | 5.0 | 0.5 | 0.001 | 4.68 | 625.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   56.9s finished
```

| 37 | 0.6049 | 5.0 | 1.0 | 0.001 | 4.336 | 614.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

| 38 | 0.6055 | 5.0 | 1.0 | 0.001 | 1.0 | 643.3 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   23.2s finished
```

| 39 | 0.5963 | 5.0 | 0.5 | 1.0 | 3.818 | 605.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   58.1s finished
```

| 40 | 0.6044 | 5.0 | 1.0 | 0.001 | 3.512 | 622.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   22.6s finished
```

| 41 | 0.6009 | 3.028 | 0.5382 | 0.09153 | 6.81 | 621.3 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.5s finished
```

| 42 | 0.6059 | 5.0 | 0.5 | 0.001 | 1.0 | 650.0 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.1s finished
```

| 43 | 0.5963 | 5.0 | 0.5 | 1.0 | 4.184 | 639.4 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   52.1s finished
```

| 44 | 0.6041 | 5.0 | 1.0 | 0.001 | 3.562 | 561.9 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.2s finished
```

| 45 | 0.6061 | 5.0 | 0.5 | 0.001 | 1.0 | 605.2 |

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   22.3s finished
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   22.5s finished
```

```
|  46        |  0.5962   |  4.79     |  0.5      |  1.0      |  7.0      |  595.6    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.6s finished
```

```
|  47        |  0.5965   |  5.0      |  0.5      |  1.0      |  3.91     |  550.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.2s finished
```

```
|  48        |  0.6041   |  4.097    |  0.5      |  0.001    |  4.667    |  617.8    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   58.2s finished
```

```
|  49        |  0.6054   |  5.0      |  1.0      |  0.001    |  1.0      |  627.8    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   53.0s finished
```

```
|  50        |  0.6054   |  5.0      |  1.0      |  0.001    |  7.0      |  577.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   59.8s finished
```

```
|  51        |  0.6049   |  5.0      |  1.0      |  0.001    |  4.352    |  649.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.0s finished
```

```
|  52        |  0.6047   |  5.0      |  0.5      |  0.001    |  2.316    |  625.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   44.7s finished
```

```
|  53        |  0.6051   |  4.92     |  1.0      |  0.001    |  4.222    |  582.9    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.5s finished
```

```
|  54        |  0.6044   |  4.849    |  0.5848   |  0.0394   |  4.317    |  560.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.9s finished
```

```
|  55        |  0.6046   |  4.192    |  0.6838   |  0.06154  |  4.439    |  600.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.3s finished
```

```
|  56        |  0.6039   |  4.952    |  0.7966   |  0.006261 |  2.744    |  637.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.9s finished
```

```
|  57        |  0.5967   |  5.0      |  1.0      |  1.0      |  2.799    |  648.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.4s finished
```

```
|  58        |  0.5967   |  4.04     |  0.6081   |  0.953    |  6.559    |  599.9    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.7s finished
```

```
|  59        |  0.6007   |  3.936    |  0.5122   |  0.06401  |  5.271    |  612.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  60        |  0.6009   |  3.406    |  0.5298   |  0.246    |  1.098    |  603.3    |
=====================================================================================
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   21.1s finished
```

In [13]:

```
rf_bo.max
```

Out[13]:

```
{'params': {'max_depth': 5.0,
  'max_features': 0.5,
  'min_impurity_decrease': 0.001,
  'min_samples_leaf': 1.0,
  'n_estimators': 605.153190194469},
 'target': 0.6060533028225061}
```

In [21]:

```
ids_test=test.ID
rf_label= RandomForestRegressor(**{'max_depth': 5,'max_features': 0.5,'min_impurity_decrease': 0.00
1,'min_samples_leaf': 1,'n_estimators': 605},random_state= random_seed,oob_score= True)
rf_label.fit(train,targets)
print('oob_score: ',rf_label.oob_score_)
cv_score=cross_val_score(rf_label,train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = rf_label.predict(test)
subm.to_csv('rf_5folds_meanenc.csv', index=False)
```

```
oob_score:  0.6059308454243664
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6056875307382426  +/-  0.022821727761309956
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.7min finished
```

LB(0.54846,0.55850),CV:.6056

XGBoost Bayesian Tuning

In [0]:

```
def
xgb_evaluate(n_estimators,max_depth,subsample,colsample_bytree,gamma,reg_alpha,min_child_weight):
  params={
      'n_estimators':int(n_estimators),
          'max_depth':int(max_depth),
          'min_child_weight':int(min_child_weight),
          'gamma':gamma,
          'subsample':subsample,
          'colsample_bytree':colsample_bytree,
          'reg_alpha':reg_alpha
          }
```

```
    xgb_label= xgb.XGBRegressor(**params,silent= True,random_state= random_seed,learning_rate= .05)
    cv_score=cross_val_score(xgb_label,train,y_train,scoring='r2',cv= cv3,verbose=1,n_jobs=1)
    return cv_score.mean()
```

In [15]:

```
xgb_bo= BayesianOptimization(xgb_evaluate,{
               'subsample': (.5,1),
               'colsample_bytree': (.5,1),
               'min_child_weight':(1,10),
               'max_depth': (1,8),
               'n_estimators':(180,230),
             'gamma':(.001,100),
             'reg_alpha':(.001,100)
          })
xgb_bo.maximize(init_points=10, n_iter=50, acq='ei')
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|   iter    |  target   | colsam... |   gamma   | max_depth | min_ch... | n_esti... | reg_alpha | s
ubsample |
-------------------------------------------------------------------------------------------------
------

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  1        |  0.6046   |  0.5662   |  52.69    |  5.867    |  9.282    |  184.7    |  82.22    |
.7814    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  2        |  0.6      |  0.6887   |  55.18    |  7.549    |  3.163    |  188.5    |  68.69    |
.7124    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  3        |  0.6059   |  0.9435   |  16.39    |  3.122    |  2.225    |  195.5    |  20.63    |
.6013    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  4        |  0.6069   |  0.515    |  23.13    |  2.29     |  7.699    |  184.9    |  60.24    |
.5523    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  5        |  0.6066   |  0.6506   |  93.84    |  1.791    |  4.822    |  223.2    |  81.86    |
.5456    |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   20.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  6        |  0.6017   |  0.7706   |  82.71    |  5.48     |  3.06     |  213.7    |  46.89    |
.931     |

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   16.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  7        |  0.603    |  0.5437   |  50.25    |  6.433    |  3.308    |  204.4    |  86.99    |
.9644    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   41.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  8        |  0.5887   |  0.9786   |  54.67    |  6.005    |  7.364    |  227.1    |  5.058    |
.6258    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  9        |  0.5983   |  0.9585   |  52.78    |  6.236    |  9.755    |  207.1    |  15.17    |
.9968    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.1s finished
```

```
|  10       |  0.6066   |  0.5097   |  33.49    |  1.546    |  3.087    |  192.9    |  6.906    |
.9042    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.6s finished
```

```
|  11       |  0.608    |  0.9038   |  99.72    |  1.39     |  3.416    |  181.5    |  14.05    |
.561     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.7s finished
```

```
|  12       |  0.6059   |  0.7583   |  96.56    |  1.107    |  8.995    |  199.4    |  93.8     |
.513     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.5s finished
```

```
|  13       |  0.6076   |  0.8642   |  6.38     |  1.453    |  7.543    |  183.4    |  3.692    |
.8132    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.9s finished
```

```
|  14       |  0.6068   |  0.8423   |  42.19    |  1.066    |  1.427    |  180.5    |  24.02    |
.9003    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.5s finished
```

```
|  15       |  0.6049   |  0.5298   |  0.8529   |  1.268    |  9.915    |  207.6    |  97.39    |
.5431    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.4s finished
```

```
|  16       |  0.6075   |  0.5258   |  2.322    |  2.127    |  3.888    |  180.6    |  6.788    |
.9866    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.9s finished
```

```
|  17       |  0.6059   |  0.5177   |  51.58    |  1.221    |  9.942    |  214.0    |  74.77    |
.7994    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.0s finished
```

```
|  18       |  0.6072   |  0.7632   |  1.048    |  1.262    |  4.942    |  180.9    |  10.53    |
.7749    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.5s finished
```

```
|  19       |  0.6068   |  0.519    |  0.2915   |  1.224    |  7.897    |  181.1    |  40.42    |
.5932    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.2s finished
```

```
|  20       |  0.6063   |  0.573    |  97.93    |  1.369    |  9.831    |  182.8    |  29.62    |
.7782    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.5s finished
```

```
|  21       |  0.6075   |  0.8662   |  22.7     |  1.488    |  3.37     |  180.1    |  9.583    |
.5554    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.1s finished
```

```
|  22       |  0.6056   |  0.5171   |  94.09    |  1.048    |  4.042    |  222.3    |  98.09    |
.8149    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.7s finished
```

```
|  23       |  0.6073   |  0.7252   |  19.07    |  1.093    |  2.002    |  180.1    |  28.33    |
.7802    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.4s finished
```

```
|  24       |  0.6071   |  0.7244   |  99.02    |  1.03     |  1.891    |  185.5    |  44.13    |
.7318    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.7s finished
```

```
|  25       |  0.6073   |  0.9524   |  26.29    |  1.141    |  4.988    |  180.7    |  4.516    |
.8408    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.3s finished
```

```
|  26       |  0.6068   |  0.8816   |  4.699    |  1.388    |  1.022    |  181.7    |  13.18    |
.8407    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.2s finished
```

```
|  27       |  0.6076   |  0.8655   |  2.17     |  1.113    |  6.966    |  181.2    |  11.78    |
.6194    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.2s finished
```

```
|  28       |  0.6074   |  0.8399   |  99.66    |  1.531    |  3.448    |  181.7    |  1.48     |
.7912    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.5s finished
```

```
|  29       |  0.6066   |  0.5713   |  28.95    |  1.076    |  9.345    |  181.0    |  40.04    |
.6583   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.8s finished
```

```
|  30       |  0.6071   |  0.6884   |  98.73    |  1.412    |  1.567    |  180.7    |  16.23    |
.7429   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.7s finished
```

```
|  31       |  0.6067   |  0.9243   |  62.92    |  1.013    |  7.904    |  200.9    |  89.28    |
.7481   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.8s finished
```

```
|  32       |  0.6065   |  0.799    |  97.45    |  1.097    |  1.238    |  181.3    |  3.496    |
.913    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.0s finished
```

```
|  33       |  0.6071   |  0.8138   |  26.45    |  1.035    |  1.045    |  180.9    |  23.65    |
.8328   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.0s finished
```

```
|  34       |  0.607    |  0.5527   |  19.69    |  2.06     |  2.891    |  181.3    |  0.1469   |
.5057   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.8s finished
```

```
|  35       |  0.6074   |  0.6465   |  5.743    |  1.666    |  4.286    |  180.2    |  0.6132   |
.68     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.0s finished
```

```
|  36       |  0.6078   |  0.9892   |  2.963    |  1.03     |  5.869    |  181.0    |  10.31    |
.808    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.7s finished
```

```
|  37       |  0.6066   |  0.7578   |  0.3356   |  1.189    |  9.655    |  180.1    |  25.43    |
.9303   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.5s finished
```

```
|  38       |  0.6066   |  0.5054   |  62.5     |  1.211    |  8.979    |  180.1    |  15.48    |
.5501   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.4s finished
```

```
|  39       |  0.6072   |  0.6794   |  6.706    |  1.061    |  9.861    |  181.3    |  8.962    |
```

```
.9046   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.0s finished
```

```
|  40       |  0.6063   |  0.8077   |  47.96    |  1.047    |  1.093    |  200.7    |  89.99    |
.5883   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.7s finished
```

```
|  41       |  0.6061   |  0.7633   |  99.94    |  1.076    |  2.008    |  187.9    |  56.61    |
.984    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.7s finished
```

```
|  42       |  0.6067   |  0.5832   |  6.048    |  1.074    |  1.571    |  185.3    |  7.731    |
.7379   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.6s finished
```

```
|  43       |  0.6067   |  0.626    |  22.0     |  1.06     |  1.169    |  181.2    |  12.38    |
.7667   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.6s finished
```

```
|  44       |  0.6079   |  0.8278   |  3.706    |  1.385    |  9.658    |  180.8    |  0.2024   |
.7597   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.4s finished
```

```
|  45       |  0.6067   |  0.5467   |  53.4     |  1.02     |  1.752    |  180.1    |  11.16    |
.6988   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    6.1s finished
```

```
|  46       |  0.6063   |  0.6095   |  10.71    |  1.315    |  1.425    |  180.0    |  0.8726   |
.8769   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.6s finished
```

```
|  47       |  0.6072   |  0.7974   |  3.477    |  1.277    |  1.667    |  229.3    |  81.57    |
.9098   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.4s finished
```

```
|  48       |  0.6071   |  0.8451   |  0.8441   |  1.209    |  3.193    |  225.7    |  99.23    |
.7831   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.2s finished
```

```
|  49       |  0.6066   |  0.8026   |  30.13    |  1.325    |  9.634    |  224.5    |  95.26    |
.5657   |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.4s finished

|  50       |  0.6076   |  0.7517   |  4.008    |  1.034    |  2.317    |  228.3    |  85.34    |
.6672   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.4s finished

|  51       |  0.6059   |  0.5151   |  16.63    |  1.17     |  2.069    |  225.5    |  95.56    |
.7719   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.2s finished

|  52       |  0.6074   |  0.8583   |  1.821    |  1.187    |  9.14     |  181.5    |  0.9646   |
.8986   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.0s finished

|  53       |  0.6071   |  0.8719   |  98.44    |  1.261    |  9.308    |  180.4    |  24.51    |
.9311   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.3s finished

|  54       |  0.6066   |  0.8026   |  46.49    |  1.179    |  9.792    |  227.9    |  97.91    |
.5695   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.4s finished

|  55       |  0.6059   |  0.5493   |  1.053    |  1.096    |  8.237    |  230.0    |  98.77    |
.5912   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    7.9s finished

|  56       |  0.6072   |  0.6088   |  5.057    |  1.056    |  2.454    |  200.0    |  45.33    |
.5321   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   10.0s finished

|  57       |  0.6076   |  0.9466   |  0.3838   |  1.001    |  7.255    |  204.7    |  57.83    |
.5502   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.4s finished

|  58       |  0.6079   |  0.7683   |  3.635    |  1.053    |  1.05     |  193.0    |  43.32    |
.5613   |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.1s finished

|  59       |  0.608    |  0.9588   |  5.396    |  1.095    |  5.173    |  182.3    |  2.449    |
.532    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  60       |  0.608    |  0.9608   |  96.86    |  1.033    |  9.877    |  181.1    |  13.91    |
.5168   |
=====================================================================================================
```

```
==========================================================================================
======
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.1s finished
```

In [16]:

```
xgb_bo.max
```

Out[16]:

```
{'params': {'colsample_bytree': 0.9037809011527773,
  'gamma': 99.71989153287436,
  'max_depth': 1.3897314547237558,
  'min_child_weight': 3.4161279924807255,
  'n_estimators': 181.54045840649545,
  'reg_alpha': 14.045841492319456,
  'subsample': 0.5609960527163034},
 'target': 0.6080095379748612}
```

In [22]:

```
xgb_lab= xgb.XGBRegressor(**{'colsample_bytree': 0.9037809011527773,
  'gamma': 99.71989153287436,
  'max_depth': 1,
  'min_child_weight': 3,
  'n_estimators': 181,
  'reg_alpha': 14.045841492319456,
  'subsample': 0.5609960527163034},random_state= random_seed,silent=True)
xgb_lab.fit(train,targets)
cv_score=cross_val_score(xgb_lab,train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = xgb_lab.predict(test)
subm.to_csv('xgb_meanenc.csv', index=False)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6070954901435992  +/-  0.021917588798338678
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   25.6s finished
```

LB(0.54578,0.55596),CV:.6070

ExtraTrees Bayesian Tuning

In [0]:

```
def et_evaluate(n_estimators,max_depth,min_samples_leaf,max_features,min_impurity_decrease):
  params={
      'n_estimators':int(n_estimators),
          'max_depth':int(max_depth),
          'min_samples_leaf':int(min_samples_leaf),
          'min_impurity_decrease':min_impurity_decrease,
          'max_features':max_features
          }
  et_label= ExtraTreesRegressor(**params,random_state= random_seed)
  cv_score=cross_val_score(et_label,train,y_train,scoring='r2',cv= cv3,verbose=1,n_jobs=1)
  return cv_score.mean()
```

In [18]:

```
et_bo= BayesianOptimization(et_evaluate,{'n_estimators':(550,650),
          'max_depth':(1,5),
          'min_samples_leaf':(1,7),
          'min_impurity_decrease':(.001,1),
          'max_features':(.5,1)
          })
```

```
et_bo.maximize(init_points=10, n_iter=50, acq='ei')
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|   iter    |  target   | max_depth | max_fe... | min_im... | min_sa... | n_esti... |
-------------------------------------------------------------------------------------
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  1        |  0.5976   |  4.368    |  0.7927   |  0.6947   |  5.324    |  565.9    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  2        |  0.5974   |  3.054    |  0.8543   |  0.1196   |  5.931    |  618.5    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   45.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  3        |  0.5998   |  4.382    |  0.9693   |  0.5367   |  6.943    |  635.6    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   38.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  4        |  0.6032   |  4.992    |  0.7609   |  0.1714   |  6.895    |  628.4    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  5        |  0.4428   |  1.04     |  0.9258   |  0.2377   |  1.064    |  630.3    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  6        |  0.5527   |  2.642    |  0.9011   |  0.5342   |  6.072    |  615.7    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   50.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  7        |  0.6008   |  4.162    |  0.9856   |  0.1582   |  2.821    |  649.4    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   12.9s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  8        |  0.4352   |  1.994    |  0.7736   |  0.3029   |  1.609    |  634.9    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  9        |  0.4276   |  1.482    |  0.5272   |  0.2989   |  5.713    |  598.5    |
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.9s finished
```

```
|  10       |  0.5978   |  3.893    |  0.6774   |  0.3032   |  4.646    |  570.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.9s finished
```

```
|  11       |  0.4461   |  1.0      |  1.0      |  0.001    |  7.0      |  650.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   13.5s finished
```

```
|  12        |  0.4472   |  1.0      |  1.0      |  0.001    |  1.0      |  550.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   45.4s finished
```

```
|  13        |  0.5957   |  5.0      |  1.0      |  1.0      |  7.0      |  647.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   55.4s finished
```

```
|  14        |  0.5999   |  5.0      |  1.0      |  0.001    |  1.0      |  567.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.2s finished
```

```
|  15        |  0.602    |  4.852    |  0.983    |  0.01178  |  1.087    |  649.6    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   14.1s finished
```

```
|  16        |  0.4471   |  1.0      |  1.0      |  1.0      |  1.0      |  567.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   56.3s finished
```

```
|  17        |  0.6034   |  5.0      |  1.0      |  0.001    |  7.0      |  580.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   28.7s finished
```

```
|  18        |  0.6032   |  5.0      |  0.5      |  0.001    |  7.0      |  550.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.8s finished
```

```
|  19        |  0.6032   |  5.0      |  0.5      |  0.001    |  7.0      |  570.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   56.7s finished
```

```
|  20        |  0.5998   |  5.0      |  1.0      |  0.001    |  1.0      |  582.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   40.5s finished
```

```
|  21        |  0.5957   |  5.0      |  1.0      |  1.0      |  3.094    |  574.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

```
|  22        |  0.5998   |  5.0      |  1.0      |  0.001    |  1.0      |  618.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   48.2s finished
```

```
|  23        |  0.6031   |  4.966    |  0.9678   |  0.1054   |  6.301    |  620.1    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished
```

```
|  24        |  0.6034    |  5.0       |  1.0       |  0.001     |  7.0       |  638.8     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   46.6s finished
```

```
|  25        |  0.5957    |  5.0       |  1.0       |  1.0       |  3.468     |  650.0     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   56.0s finished
```

```
|  26        |  0.5988    |  5.0       |  1.0       |  0.001     |  4.273     |  568.3     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.8s finished
```

```
|  27        |  0.6022    |  5.0       |  0.5       |  0.001     |  2.617     |  557.4     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.9s finished
```

```
|  28        |  0.426     |  1.0       |  0.5       |  0.001     |  7.0       |  581.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.7s finished
```

```
|  29        |  0.6038    |  5.0       |  0.5       |  0.001     |  1.0       |  594.1     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   31.4s finished
```

```
|  30        |  0.6011    |  5.0       |  0.5       |  0.001     |  4.944     |  586.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   22.8s finished
```

```
|  31        |  0.5962    |  5.0       |  0.5       |  1.0       |  7.0       |  558.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.0s finished
```

```
|  32        |  0.6011    |  5.0       |  0.5       |  0.001     |  3.6       |  646.6     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.0s finished
```

```
|  33        |  0.5962    |  5.0       |  0.5       |  1.0       |  2.854     |  565.2     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   15.7s finished
```

```
|  34        |  0.4475    |  1.0       |  1.0       |  1.0       |  7.0       |  623.9     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.4s finished
```

```
|  35        |  0.6038    |  5.0       |  0.5       |  0.001     |  1.0       |  605.6     |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.5s finished
```

```
|  36        |  0.6032   |  5.0      |  0.5      |  0.001    |  7.0      |  633.1    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.4s finished

|  37        |  0.6013   |  4.936    |  0.6226   |  0.02934  |  3.777    |  575.5    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   50.3s finished

|  38        |  0.5999   |  4.3      |  1.0      |  0.001    |  4.051    |  618.1    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   22.6s finished

|  39        |  0.5965   |  5.0      |  0.5      |  1.0      |  1.0      |  550.0    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    9.3s finished

|  40        |  0.4275   |  1.0      |  0.5      |  0.001    |  1.0      |  615.1    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.0s finished

|  41        |  0.6032   |  5.0      |  0.5      |  0.001    |  7.0      |  602.1    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished

|  42        |  0.5998   |  5.0      |  1.0      |  0.001    |  1.0      |  625.2    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   32.7s finished

|  43        |  0.6032   |  5.0      |  0.5      |  0.001    |  7.0      |  616.4    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  1.0min finished

|  44        |  0.6034   |  5.0      |  1.0      |  0.001    |  7.0      |  631.9    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.4s finished

|  45        |  0.6029   |  4.987    |  0.606    |  0.1642   |  1.794    |  599.4    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   44.3s finished

|  46        |  0.6039   |  5.0      |  0.7314   |  0.001    |  7.0      |  594.7    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   46.8s finished

|  47        |  0.5957   |  4.248    |  1.0      |  1.0      |  1.797    |  647.7    |

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   33.4s finished

|  48        |  0.6022   |  5.0      |  0.5      |  0.001    |  2.783    |  622.6    |
```

```
|  48        |  0.6022   |  5.0      |  0.5      |  0.001    |  2.783    |  622.6    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   26.2s finished
```

```
|  49        |  0.6022   |  4.986    |  0.5329   |  0.06959  |  6.795    |  563.7    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   35.9s finished
```

```
|  50        |  0.5961   |  4.889    |  0.7622   |  0.8854   |  6.984    |  636.0    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.6s finished
```

```
|  51        |  0.6021   |  5.0      |  0.5      |  0.001    |  2.425    |  570.9    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   41.0s finished
```

```
|  52        |  0.6006   |  4.955    |  0.9108   |  0.04719  |  3.977    |  552.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   45.8s finished
```

```
|  53        |  0.6003   |  4.967    |  0.9332   |  0.05143  |  4.479    |  603.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   58.7s finished
```

```
|  54        |  0.5998   |  5.0      |  1.0      |  0.001    |  1.549    |  589.4    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.9s finished
```

```
|  55        |  0.6009   |  4.702    |  0.9945   |  0.2422   |  3.839    |  647.5    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   34.1s finished
```

```
|  56        |  0.5962   |  4.699    |  0.6828   |  0.958    |  6.87     |  644.6    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   24.1s finished
```

```
|  57        |  0.5964   |  4.981    |  0.5213   |  0.7639   |  5.676    |  577.7    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   47.4s finished
```

```
|  58        |  0.602    |  4.999    |  0.9626   |  0.01122  |  1.03     |  597.2    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   30.2s finished
```

```
|  59        |  0.5996   |  3.988    |  0.7372   |  0.001353 |  6.923    |  617.3    |
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
|  60        |  0.5964   |  4.987    |  0.6005   |  0.9998   |  5.981    |  593.6    |
=========================================================================================
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   27.6s finished
```

In [19]:

```
et_bo.max
```

Out[19]:

```
{'params': {'max_depth': 5.0,
  'max_features': 0.7314395497956648,
  'min_impurity_decrease': 0.001,
  'min_samples_leaf': 7.0,
  'n_estimators': 594.714569291968},
 'target': 0.6039282280667301}
```

In [23]:

```
et_lab= ExtraTreesRegressor(**{'max_depth': 5,
  'max_features': 0.7314395497956648,
  'min_impurity_decrease': 0.001,
  'min_samples_leaf': 7,
  'n_estimators': 594},random_state= random_seed,oob_score= True,bootstrap= True)
et_lab.fit(train,targets)
print('oob_score: ',et_lab.oob_score_)
cv_score=cross_val_score(et_lab,train,targets,scoring='r2',cv= cv,verbose=1,n_jobs=1)
print(cv_score.mean(),' +/- ',cv_score.std())
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = et_lab.predict(test)
subm.to_csv('et_5folds_meanenc.csv', index=False)
```

```
oob_score:  0.6037204363544807
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
0.6027958466070058  +/-  0.021437728345114402
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.7min finished
```

LB(0.54888,0.55583),CV:.6027

Stacking Bayesian Tuned Models

In [0]:

```
ridge= Ridge(random_state=random_seed,fit_intercept= False,alpha=0)
stack = StackingCVRegressor(regressors=(rf_label, xgb_lab,et_lab),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=cv)

cv_score=cross_val_score(stack,X,y,scoring='r2',cv= cv,verbose=1,n_jobs=-1)
print(cv_score.mean(),' +/- ',cv_score.std())
X= np.array(train)
y= targets
x_test= np.array(test)
stack.fit(X,y)
ids_test= test.ID
y_pred = stack.predict(x_test)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = y_pred
subm.to_csv('submission_xgb_rf_stack_ridge_meanenc.csv', index=False)
```

LB(0.54879,0.55873)

Simple DeepLearning

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_hot.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_hot.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
SS= StandardScaler()
ss_train=SS.fit_transform(train)
ss_test= SS.transform(test)
```

In [0]:

```
import keras.backend as K

def r2(y_true, y_pred):
  SS_res =  K.sum(K.square( y_true-y_pred ))
  SS_tot = K.sum(K.square( y_true - K.mean(y_true)))
  return ( 1 - SS_res/(SS_tot + K.epsilon()))
```

In [0]:

```
from keras.layers import Dense, Dropout, BatchNormalization, Activation
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

In [0]:

```
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=15,
        verbose=1),

    ModelCheckpoint(
        'model.h5',
        monitor='val_loss',
        save_best_only=True,
        verbose=0)
]
```

In [0]:

```
#https://github.com/GKarmakar/RegressionUsingNN/blob/master/RegressionUsingNeuralNetwork.ipynb
#https://www.kaggle.com/frednavruzov/baseline-to-start-with-keras-lb-0-55
cv3= KFold(5,True,random_seed)
def simple_net(input_dims=394,act_func='relu',batch_size=20,epochs=30,dropout= .3,init='normal'):
  model = Sequential()
  #input layer
  model.add(Dense(input_dims, input_dim=input_dims,kernel_regularizer = 'l2',
                  kernel_initializer = init,))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Dropout(dropout))
  # hidden layers
  model.add(Dense(input_dims,kernel_regularizer = 'l2',
                  kernel_initializer = init,))
  model.add(BatchNormalization())
  model.add(Activation(act_func))
  model.add(Dropout(dropout))

  model.add(Dense(input_dims//2,kernel_regularizer = 'l2',
                  kernel_initializer = init))
  model.add(BatchNormalization())
  model.add(Activation(act_func))
  model.add(Dropout(dropout))
  model.add(Dense(input_dims//4, activation=act_func,kernel_regularizer = 'l2',
                  kernel_initializer = init))
```

```
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error',
                  optimizer='adam',metrics=[r2])
    return model
```

In [0]:

```
def visualize_learning_curve(history):
  # list all data in history
  print(history.history.keys())
  # summarize history for accuracy
  plt.plot(history.history['r2'][:])
  plt.plot(history.history['val_r2'][:])
  plt.title('model r2')
  plt.ylabel('R2')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper right')
  plt.show()
  # summarize history for loss
  plt.plot(history.history['loss'][:])
  plt.plot(history.history['val_loss'][:])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper right')
  plt.show()
```

In [0]:

```
def train_nn(X,y,act_func='tanh',batch_size=20,epochs=30,dropout= .3,init='normal'):
  net=KerasRegressor(build_fn= simple_net,epochs=epochs,batch_size=batch_size,verbose=1,init= init,
dropout=dropout,act_func= act_func)
  cv_score=cross_validate(net,X,y,scoring='r2',cv= cv3,verbose=0,return_train_score=True)
  print('Test R2: ',cv_score['test_score'].mean(),'Train R2: ',cv_score['train_score'].mean())
  return net
```

In [0]:

```
def coordinate(X,y):
  estimator = train_nn(X, y)
  history = estimator.fit(X, y, validation_split=0.2,
                          callbacks=callbacks,
                          verbose=1)
  visualize_learning_curve(history)
  return estimator
```

In [0]:

```
def predict(X,model):
  subm = pd.DataFrame()
  subm['ID'] =test.ID
  subm['y'] = model.predict(ss_test)
  subm.to_csv('NN.csv', index=False)
```

In [39]:

```
model=coordinate(ss_train,y_train)
predict(ss_test,model)
```

```
Epoch 1/30
3367/3367 [==============================] - 26s 8ms/step - loss: 8515.5914 - r2: -64.4362
Epoch 2/30
3367/3367 [==============================] - 3s 1ms/step - loss: 3916.9261 - r2: -28.6934
Epoch 3/30
3367/3367 [==============================] - 3s 1ms/step - loss: 1979.5502 - r2: -14.2376
Epoch 4/30
3367/3367 [==============================] - 3s 1ms/step - loss: 1050.3632 - r2: -6.8811
Epoch 5/30
3367/3367 [==============================] - 3s 1ms/step - loss: 571.8356 - r2: -3.1734
Epoch 6/30
3367/3367 [==============================] - 3s 1ms/step - loss: 327.7656 - r2: -1.2676
```

```
Epoch 7/30
3367/3367 [==============================] - 3s 994us/step - loss: 224.3935 - r2: -0.4904
Epoch 8/30
3367/3367 [==============================] - 3s 1ms/step - loss: 183.9367 - r2: -0.1745
Epoch 9/30
3367/3367 [==============================] - 3s 1ms/step - loss: 163.3458 - r2: -0.0351
Epoch 10/30
3367/3367 [==============================] - 3s 1ms/step - loss: 150.0476 - r2: 0.0531
Epoch 11/30
3367/3367 [==============================] - 3s 1ms/step - loss: 147.4296 - r2: 0.0635
Epoch 12/30
3367/3367 [==============================] - 3s 1ms/step - loss: 144.6273 - r2: 0.0893
Epoch 13/30
3367/3367 [==============================] - 3s 1ms/step - loss: 141.2322 - r2: 0.1027
Epoch 14/30
3367/3367 [==============================] - 4s 1ms/step - loss: 130.1913 - r2: 0.1978
Epoch 15/30
3367/3367 [==============================] - 4s 1ms/step - loss: 110.3492 - r2: 0.3447
Epoch 16/30
3367/3367 [==============================] - 3s 1ms/step - loss: 96.9133 - r2: 0.4475
Epoch 17/30
3367/3367 [==============================] - 3s 1ms/step - loss: 90.6750 - r2: 0.4877
Epoch 18/30
3367/3367 [==============================] - 4s 1ms/step - loss: 85.6620 - r2: 0.5013
Epoch 19/30
3367/3367 [==============================] - 4s 1ms/step - loss: 101.1352 - r2: 0.3810
Epoch 20/30
3367/3367 [==============================] - 3s 1ms/step - loss: 83.2798 - r2: 0.5428
Epoch 21/30
3367/3367 [==============================] - 4s 1ms/step - loss: 79.5521 - r2: 0.5603
Epoch 22/30
3367/3367 [==============================] - 4s 1ms/step - loss: 77.9325 - r2: 0.5657
Epoch 23/30
3367/3367 [==============================] - 4s 1ms/step - loss: 75.3524 - r2: 0.5826
Epoch 24/30
3367/3367 [==============================] - 4s 1ms/step - loss: 72.9777 - r2: 0.6013
Epoch 25/30
3367/3367 [==============================] - 4s 1ms/step - loss: 71.4064 - r2: 0.6171
Epoch 26/30
3367/3367 [==============================] - 3s 1ms/step - loss: 71.4893 - r2: 0.6102
Epoch 27/30
3367/3367 [==============================] - 4s 1ms/step - loss: 69.8228 - r2: 0.6213
Epoch 28/30
3367/3367 [==============================] - 4s 1ms/step - loss: 69.3109 - r2: 0.6125
Epoch 29/30
3367/3367 [==============================] - 3s 1ms/step - loss: 67.7538 - r2: 0.6324
Epoch 30/30
3367/3367 [==============================] - 3s 1ms/step - loss: 67.1110 - r2: 0.6316
842/842 [==============================] - 10s 12ms/step
3367/3367 [==============================] - 2s 463us/step
Epoch 1/30
3367/3367 [==============================] - 26s 8ms/step - loss: 8751.1529 - r2: -68.8466
Epoch 2/30
3367/3367 [==============================] - 4s 1ms/step - loss: 4424.9088 - r2: -32.8068
Epoch 3/30
3367/3367 [==============================] - 3s 1ms/step - loss: 1779.0366 - r2: -12.5719
Epoch 4/30
3367/3367 [==============================] - 4s 1ms/step - loss: 873.2336 - r2: -5.4628
Epoch 5/30
3367/3367 [==============================] - 4s 1ms/step - loss: 450.5973 - r2: -2.2676
Epoch 6/30
3367/3367 [==============================] - 3s 1ms/step - loss: 266.8097 - r2: -0.8226
Epoch 7/30
3367/3367 [==============================] - 4s 1ms/step - loss: 196.7855 - r2: -0.2842
Epoch 8/30
3367/3367 [==============================] - 4s 1ms/step - loss: 168.8520 - r2: -0.0769
Epoch 9/30
3367/3367 [==============================] - 4s 1ms/step - loss: 150.7507 - r2: 0.0352
Epoch 10/30
3367/3367 [==============================] - 3s 1ms/step - loss: 143.0735 - r2: 0.0912
Epoch 11/30
3367/3367 [==============================] - 4s 1ms/step - loss: 148.8341 - r2: 0.0533
Epoch 12/30
3367/3367 [==============================] - 4s 1ms/step - loss: 137.3620 - r2: 0.1409
Epoch 13/30
3367/3367 [==============================] - 4s 1ms/step - loss: 129.4968 - r2: 0.1892
Epoch 14/30
```

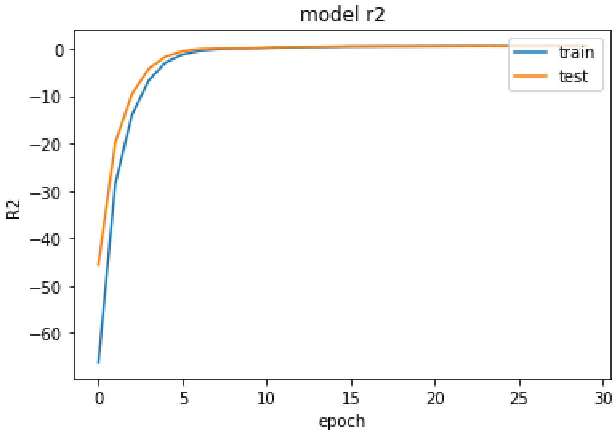
```
3367/3367 [==============================] - 4s 1ms/step - loss: 116.2159 - r2: 0.2763
Epoch 15/30
3367/3367 [==============================] - 4s 1ms/step - loss: 101.2555 - r2: 0.4020
Epoch 16/30
3367/3367 [==============================] - 4s 1ms/step - loss: 90.9139 - r2: 0.4733
Epoch 17/30
3367/3367 [==============================] - 4s 1ms/step - loss: 84.8745 - r2: 0.5137
Epoch 18/30
3367/3367 [==============================] - 4s 1ms/step - loss: 82.0231 - r2: 0.5395
Epoch 19/30
3367/3367 [==============================] - 4s 1ms/step - loss: 78.8118 - r2: 0.5453
Epoch 20/30
3367/3367 [==============================] - 4s 1ms/step - loss: 77.0596 - r2: 0.5609
Epoch 21/30
3367/3367 [==============================] - 4s 1ms/step - loss: 74.9461 - r2: 0.5754
Epoch 22/30
3367/3367 [==============================] - 4s 1ms/step - loss: 73.8421 - r2: 0.5741
Epoch 23/30
3367/3367 [==============================] - 4s 1ms/step - loss: 71.7142 - r2: 0.5978
Epoch 24/30
3367/3367 [==============================] - 4s 1ms/step - loss: 69.4208 - r2: 0.6057
Epoch 25/30
3367/3367 [==============================] - 4s 1ms/step - loss: 68.8991 - r2: 0.6132
Epoch 26/30
3367/3367 [==============================] - 4s 1ms/step - loss: 69.1336 - r2: 0.6131
Epoch 27/30
3367/3367 [==============================] - 4s 1ms/step - loss: 69.6268 - r2: 0.6131
Epoch 28/30
3367/3367 [==============================] - 4s 1ms/step - loss: 68.5625 - r2: 0.6175
Epoch 29/30
3367/3367 [==============================] - 4s 1ms/step - loss: 67.6354 - r2: 0.6143
Epoch 30/30
3367/3367 [==============================] - 4s 1ms/step - loss: 66.6751 - r2: 0.6245
842/842 [==============================] - 9s 11ms/step
3367/3367 [==============================] - 1s 414us/step
Epoch 1/30
3367/3367 [==============================] - 26s 8ms/step - loss: 8522.6159 - r2: -69.7455
Epoch 2/30
3367/3367 [==============================] - 4s 1ms/step - loss: 3857.9594 - r2: -29.7001
Epoch 3/30
3367/3367 [==============================] - 4s 1ms/step - loss: 1883.1718 - r2: -13.5898
Epoch 4/30
3367/3367 [==============================] - 4s 1ms/step - loss: 982.8776 - r2: -6.4134
Epoch 5/30
3367/3367 [==============================] - 4s 1ms/step - loss: 522.3584 - r2: -2.8739
Epoch 6/30
3367/3367 [==============================] - 4s 1ms/step - loss: 304.2967 - r2: -1.1380
Epoch 7/30
3367/3367 [==============================] - 4s 1ms/step - loss: 212.7118 - r2: -0.4224
Epoch 8/30
3367/3367 [==============================] - 4s 1ms/step - loss: 176.5441 - r2: -0.1539
Epoch 9/30
3367/3367 [==============================] - 4s 1ms/step - loss: 165.2015 - r2: -0.0637
Epoch 10/30
3367/3367 [==============================] - 4s 1ms/step - loss: 151.2701 - r2: 0.0099
Epoch 11/30
3367/3367 [==============================] - 4s 1ms/step - loss: 141.8774 - r2: 0.0719
Epoch 12/30
3367/3367 [==============================] - 4s 1ms/step - loss: 139.1489 - r2: 0.1194
Epoch 13/30
3367/3367 [==============================] - 4s 1ms/step - loss: 144.7026 - r2: 0.0581
Epoch 14/30
3367/3367 [==============================] - 4s 1ms/step - loss: 134.4913 - r2: 0.1363
Epoch 15/30
3367/3367 [==============================] - 4s 1ms/step - loss: 110.4688 - r2: 0.3273
Epoch 16/30
3367/3367 [==============================] - 4s 1ms/step - loss: 98.9470 - r2: 0.4292
Epoch 17/30
3367/3367 [==============================] - 4s 1ms/step - loss: 90.5203 - r2: 0.4858
Epoch 18/30
3367/3367 [==============================] - 4s 1ms/step - loss: 87.4536 - r2: 0.4992
Epoch 19/30
3367/3367 [==============================] - 4s 1ms/step - loss: 84.4835 - r2: 0.5183
Epoch 20/30
3367/3367 [==============================] - 4s 1ms/step - loss: 82.0521 - r2: 0.5360
Epoch 21/30
3367/3367 [==============================] - 4s 1ms/step - loss: 79.7849 - r2: 0.5498
```

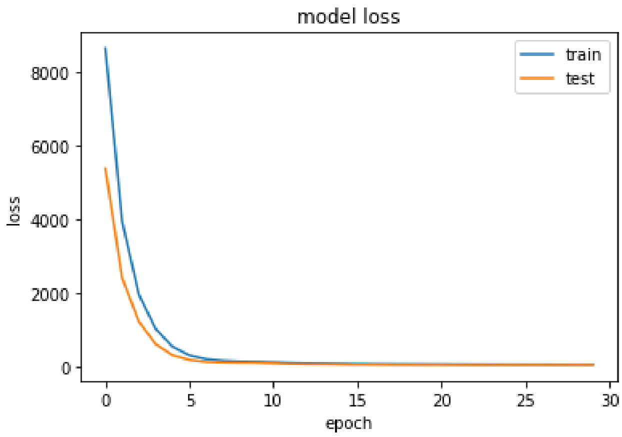
```
Epoch 22/30
3367/3367 [==============================] - 4s 1ms/step - loss: 77.1599 - r2: 0.5636
Epoch 23/30
3367/3367 [==============================] - 4s 1ms/step - loss: 75.0243 - r2: 0.5716
Epoch 24/30
3367/3367 [==============================] - 4s 1ms/step - loss: 72.3552 - r2: 0.5978
Epoch 25/30
3367/3367 [==============================] - 4s 1ms/step - loss: 72.0596 - r2: 0.5986
Epoch 26/30
3367/3367 [==============================] - 4s 1ms/step - loss: 70.7953 - r2: 0.5966
Epoch 27/30
3367/3367 [==============================] - 4s 1ms/step - loss: 68.1002 - r2: 0.6277
Epoch 28/30
3367/3367 [==============================] - 4s 1ms/step - loss: 70.7098 - r2: 0.6004
Epoch 29/30
3367/3367 [==============================] - 4s 1ms/step - loss: 69.5539 - r2: 0.5985
Epoch 30/30
3367/3367 [==============================] - 4s 1ms/step - loss: 67.5843 - r2: 0.6239
842/842 [==============================] - 9s 11ms/step
3367/3367 [==============================] - 2s 501us/step
Epoch 1/30
3367/3367 [==============================] - 27s 8ms/step - loss: 8804.8117 - r2: -67.9466
Epoch 2/30
3367/3367 [==============================] - 4s 1ms/step - loss: 4516.9677 - r2: -34.2627
Epoch 3/30
3367/3367 [==============================] - 4s 1ms/step - loss: 1928.3286 - r2: -13.8308
Epoch 4/30
3367/3367 [==============================] - 4s 1ms/step - loss: 971.9577 - r2: -6.2450
Epoch 5/30
3367/3367 [==============================] - 4s 1ms/step - loss: 506.0744 - r2: -2.6771
Epoch 6/30
3367/3367 [==============================] - 4s 1ms/step - loss: 294.0682 - r2: -1.0327
Epoch 7/30
3367/3367 [==============================] - 4s 1ms/step - loss: 207.5099 - r2: -0.3529
Epoch 8/30
3367/3367 [==============================] - 4s 1ms/step - loss: 176.5518 - r2: -0.1326
Epoch 9/30
3367/3367 [==============================] - 4s 1ms/step - loss: 166.9854 - r2: -0.0606
Epoch 10/30
3367/3367 [==============================] - 4s 1ms/step - loss: 155.6320 - r2: 0.0097
Epoch 11/30
3367/3367 [==============================] - 4s 1ms/step - loss: 142.9391 - r2: 0.0909
Epoch 12/30
3367/3367 [==============================] - 4s 1ms/step - loss: 143.5113 - r2: 0.0920
Epoch 13/30
3367/3367 [==============================] - 4s 1ms/step - loss: 133.6202 - r2: 0.1622
Epoch 14/30
3367/3367 [==============================] - 4s 1ms/step - loss: 128.6977 - r2: 0.1935
Epoch 15/30
3367/3367 [==============================] - 4s 1ms/step - loss: 115.6219 - r2: 0.2898
Epoch 16/30
3367/3367 [==============================] - 4s 1ms/step - loss: 103.4312 - r2: 0.3828
Epoch 17/30
3367/3367 [==============================] - 4s 1ms/step - loss: 91.6362 - r2: 0.4689
Epoch 18/30
3367/3367 [==============================] - 4s 1ms/step - loss: 85.0296 - r2: 0.5176
Epoch 19/30
3367/3367 [==============================] - 4s 1ms/step - loss: 82.5686 - r2: 0.5352
Epoch 20/30
3367/3367 [==============================] - 4s 1ms/step - loss: 80.4238 - r2: 0.5366
Epoch 21/30
3367/3367 [==============================] - 4s 1ms/step - loss: 77.3241 - r2: 0.5654
Epoch 22/30
3367/3367 [==============================] - 4s 1ms/step - loss: 74.3979 - r2: 0.5928
Epoch 23/30
3367/3367 [==============================] - 4s 1ms/step - loss: 73.3339 - r2: 0.5883
Epoch 24/30
3367/3367 [==============================] - 4s 1ms/step - loss: 71.4029 - r2: 0.6017
Epoch 25/30
3367/3367 [==============================] - 4s 1ms/step - loss: 71.6969 - r2: 0.5924
Epoch 26/30
3367/3367 [==============================] - 4s 1ms/step - loss: 70.0902 - r2: 0.6003
Epoch 27/30
3367/3367 [==============================] - 4s 1ms/step - loss: 68.8446 - r2: 0.6038
Epoch 28/30
3367/3367 [==============================] - 4s 1ms/step - loss: 68.0339 - r2: 0.6159
Epoch 29/30
```

```
3367/3367 [==============================] - 4s 1ms/step - loss: 69.4057 - r2: 0.6024
Epoch 30/30
3367/3367 [==============================] - 4s 1ms/step - loss: 66.3652 - r2: 0.6244
842/842 [==============================] - 10s 12ms/step
3367/3367 [==============================] - 2s 499us/step
Epoch 1/30
3368/3368 [==============================] - 27s 8ms/step - loss: 8732.5961 - r2: -68.4656
Epoch 2/30
3368/3368 [==============================] - 4s 1ms/step - loss: 5259.7661 - r2: -39.6488
Epoch 3/30
3368/3368 [==============================] - 4s 1ms/step - loss: 1912.9765 - r2: -13.8544
Epoch 4/30
3368/3368 [==============================] - 4s 1ms/step - loss: 911.5284 - r2: -5.8551
Epoch 5/30
3368/3368 [==============================] - 4s 1ms/step - loss: 459.8965 - r2: -2.2939
Epoch 6/30
3368/3368 [==============================] - 4s 1ms/step - loss: 268.3910 - r2: -0.8428
Epoch 7/30
3368/3368 [==============================] - 4s 1ms/step - loss: 196.1335 - r2: -0.2830
Epoch 8/30
3368/3368 [==============================] - 4s 1ms/step - loss: 169.8800 - r2: -0.0754
Epoch 9/30
3368/3368 [==============================] - 4s 1ms/step - loss: 151.0981 - r2: 0.0327
Epoch 10/30
3368/3368 [==============================] - 4s 1ms/step - loss: 140.9902 - r2: 0.0976
Epoch 11/30
3368/3368 [==============================] - 4s 1ms/step - loss: 138.8921 - r2: 0.1124
Epoch 12/30
3368/3368 [==============================] - 4s 1ms/step - loss: 139.3974 - r2: 0.1026
Epoch 13/30
3368/3368 [==============================] - 4s 1ms/step - loss: 134.7139 - r2: 0.1304
Epoch 14/30
3368/3368 [==============================] - 4s 1ms/step - loss: 116.2870 - r2: 0.2862
Epoch 15/30
3368/3368 [==============================] - 4s 1ms/step - loss: 100.9647 - r2: 0.3930
Epoch 16/30
3368/3368 [==============================] - 4s 1ms/step - loss: 93.8049 - r2: 0.4488
Epoch 17/30
3368/3368 [==============================] - 4s 1ms/step - loss: 88.1588 - r2: 0.4960
Epoch 18/30
3368/3368 [==============================] - 4s 1ms/step - loss: 84.8202 - r2: 0.5095
Epoch 19/30
3368/3368 [==============================] - 3s 1ms/step - loss: 80.8129 - r2: 0.5438
Epoch 20/30
3368/3368 [==============================] - 4s 1ms/step - loss: 78.4245 - r2: 0.5482
Epoch 21/30
3368/3368 [==============================] - 4s 1ms/step - loss: 74.3458 - r2: 0.5904
Epoch 22/30
3368/3368 [==============================] - 4s 1ms/step - loss: 74.7302 - r2: 0.5665
Epoch 23/30
3368/3368 [==============================] - 4s 1ms/step - loss: 73.7042 - r2: 0.5793
Epoch 24/30
3368/3368 [==============================] - 4s 1ms/step - loss: 78.3491 - r2: 0.5346
Epoch 25/30
3368/3368 [==============================] - 4s 1ms/step - loss: 71.9548 - r2: 0.6054
Epoch 26/30
3368/3368 [==============================] - 4s 1ms/step - loss: 73.0597 - r2: 0.5852
Epoch 27/30
3368/3368 [==============================] - 4s 1ms/step - loss: 74.0047 - r2: 0.5825
Epoch 28/30
3368/3368 [==============================] - 4s 1ms/step - loss: 70.2945 - r2: 0.6052
Epoch 29/30
3368/3368 [==============================] - 4s 1ms/step - loss: 69.6984 - r2: 0.6138
Epoch 30/30
3368/3368 [==============================] - 4s 1ms/step - loss: 69.4786 - r2: 0.5984
841/841 [==============================] - 10s 11ms/step
3368/3368 [==============================] - 2s 567us/step
Test R2:  0.5835123439619243 Train R2:  0.6786805666969773
Train on 3367 samples, validate on 842 samples
Epoch 1/30
3367/3367 [==============================] - 31s 9ms/step - loss: 8655.6619 - r2: -66.2429 - val_l
oss: 5392.6170 - val_r2: -45.4981
Epoch 2/30
3367/3367 [==============================] - 4s 1ms/step - loss: 3956.3348 - r2: -28.5247 - val_lo
ss: 2417.6144 - val_r2: -19.7820
Epoch 3/30
3367/3367 [==============================] - 4s 1ms/step - loss: 1965.6233 - r2: -13.9184 - val_lo
```

```
ss: 1232.6026 - val_r2: -9.4711
Epoch 4/30
3367/3367 [==============================] - 4s 1ms/step - loss: 1035.1458 - r2: -6.6174 - val_los
s: 621.2586 - val_r2: -4.1608
Epoch 5/30
3367/3367 [==============================] - 4s 1ms/step - loss: 555.0844 - r2: -2.8705 - val_loss
: 325.0117 - val_r2: -1.5952
Epoch 6/30
3367/3367 [==============================] - 4s 1ms/step - loss: 323.0361 - r2: -1.1841 - val_loss
: 198.7169 - val_r2: -0.5080
Epoch 7/30
3367/3367 [==============================] - 4s 1ms/step - loss: 219.6666 - r2: -0.4147 - val_loss
: 143.1042 - val_r2: -0.0432
Epoch 8/30
3367/3367 [==============================] - 4s 1ms/step - loss: 177.1124 - r2: -0.1055 - val_loss
: 128.3134 - val_r2: 0.0767
Epoch 9/30
3367/3367 [==============================] - 4s 1ms/step - loss: 156.4448 - r2: 0.0275 - val_loss:
122.0014 - val_r2: 0.1223
Epoch 10/30
3367/3367 [==============================] - 4s 1ms/step - loss: 149.2903 - r2: 0.0674 - val_loss:
119.3981 - val_r2: 0.1438
Epoch 11/30
3367/3367 [==============================] - 4s 1ms/step - loss: 136.5954 - r2: 0.1712 - val_loss:
105.4801 - val_r2: 0.2591
Epoch 12/30
3367/3367 [==============================] - 4s 1ms/step - loss: 122.3972 - r2: 0.2720 - val_loss:
90.7439 - val_r2: 0.3832
Epoch 13/30
3367/3367 [==============================] - 4s 1ms/step - loss: 110.6474 - r2: 0.3515 - val_loss:
83.2029 - val_r2: 0.4452
Epoch 14/30
3367/3367 [==============================] - 4s 1ms/step - loss: 105.3282 - r2: 0.3842 - val_loss:
81.1730 - val_r2: 0.4604
Epoch 15/30
3367/3367 [==============================] - 4s 1ms/step - loss: 97.2078 - r2: 0.4443 - val_loss:
73.6457 - val_r2: 0.5208
Epoch 16/30
3367/3367 [==============================] - 4s 1ms/step - loss: 90.8968 - r2: 0.4933 - val_loss:
68.7038 - val_r2: 0.5593
Epoch 17/30
3367/3367 [==============================] - 4s 1ms/step - loss: 87.2534 - r2: 0.5164 - val_loss:
68.4745 - val_r2: 0.5585
Epoch 18/30
3367/3367 [==============================] - 4s 1ms/step - loss: 84.7609 - r2: 0.5291 - val_loss:
65.6601 - val_r2: 0.5808
Epoch 19/30
3367/3367 [==============================] - 4s 1ms/step - loss: 83.3305 - r2: 0.5342 - val_loss:
63.9680 - val_r2: 0.5909
Epoch 20/30
3367/3367 [==============================] - 4s 1ms/step - loss: 81.6102 - r2: 0.5484 - val_loss:
64.5950 - val_r2: 0.5859
Epoch 21/30
3367/3367 [==============================] - 4s 1ms/step - loss: 79.3739 - r2: 0.5685 - val_loss:
64.9844 - val_r2: 0.5814
Epoch 22/30
3367/3367 [==============================] - 4s 1ms/step - loss: 77.0034 - r2: 0.5808 - val_loss:
62.1905 - val_r2: 0.6015
Epoch 23/30
3367/3367 [==============================] - 4s 1ms/step - loss: 74.8749 - r2: 0.5847 - val_loss:
60.7176 - val_r2: 0.6115
Epoch 24/30
3367/3367 [==============================] - 4s 1ms/step - loss: 73.2484 - r2: 0.6003 - val_loss:
61.7137 - val_r2: 0.6016
Epoch 25/30
3367/3367 [==============================] - 4s 1ms/step - loss: 73.2196 - r2: 0.5939 - val_loss:
62.6146 - val_r2: 0.5947
Epoch 26/30
3367/3367 [==============================] - 4s 1ms/step - loss: 71.8043 - r2: 0.6029 - val_loss:
63.0882 - val_r2: 0.5878
Epoch 27/30
3367/3367 [==============================] - 4s 1ms/step - loss: 71.9443 - r2: 0.5988 - val_loss:
63.4113 - val_r2: 0.5845
Epoch 28/30
3367/3367 [==============================] - 4s 1ms/step - loss: 70.5043 - r2: 0.6088 - val_loss:
61.8914 - val_r2: 0.5947
Epoch 29/30
```

```
3367/3367 [==============================] - 4s 1ms/step - loss: 70.1237 - r2: 0.6151 - val_loss:
66.2796 - val_r2: 0.5573
Epoch 30/30
3367/3367 [==============================] - 4s 1ms/step - loss: 70.3179 - r2: 0.6104 - val_loss:
63.1493 - val_r2: 0.5840
dict_keys(['val_loss', 'val_r2', 'loss', 'r2'])
```






```
4209/4209 [==============================] - 11s 3ms/step
```

LB(0.52588,0.52741),CV:.5835

GridSearch for NN

In [0]:

```
#https://stackoverflow.com/questions/48390601/explicitly-specifying-test-train-sets-in-
gridsearchcv
from sklearn.model_selection import PredefinedSplit
train_indices = np.full((3368,), -1, dtype=int)
test_indices = np.full((841,), 0, dtype=int)
test_fold = np.append(train_indices, test_indices)
#.2 split
np.random.shuffle(test_fold)#shuffling of array done here
ps = PredefinedSplit(test_fold)
```

In [24]:

```
net=KerasRegressor(build_fn= simple_net,verbose=False,epochs=30)

dropout_rate = [0.1, 0.2, 0.3, 0.4, ]
init = ['glorot_uniform', 'normal', 'uniform']
act=['tanh','relu']
batches = [5, 10, 20]
param_grid = dict(dropout=dropout_rate,
                  batch_size=batches,
                  act_func= act,
                  init=init)
```

```python
grid = GridSearchCV(estimator=net, param_grid=param_grid,cv=ps,verbose=2,n_jobs=-1,scoring='r2')
grid_result = grid.fit(ss_train, y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 1 folds for each of 72 candidates, totalling 72 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning:
A worker stopped while some jobs were given to the executor. This can be caused by a too short wor
ker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 58.1min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed: 105.6min finished
```

```
Best: 0.565752 using {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.4, 'init':
'glorot_uniform'}
0.556514 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.1, 'init':
'glorot_uniform'}
0.557950 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.1, 'init': 'normal'}
0.550767 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.1, 'init': 'uniform'}
0.563773 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.2, 'init':
'glorot_uniform'}
0.555549 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.2, 'init': 'normal'}
0.551606 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.2, 'init': 'uniform'}
0.550896 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.3, 'init':
'glorot_uniform'}
0.562173 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.3, 'init': 'normal'}
0.552852 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.3, 'init': 'uniform'}
0.565752 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.4, 'init':
'glorot_uniform'}
0.553614 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.4, 'init': 'normal'}
0.552952 (0.000000) with: {'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.4, 'init': 'uniform'}
0.545485 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.1, 'init':
'glorot_uniform'}
0.505039 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.1, 'init': 'normal'}
0.553780 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.1, 'init':
'uniform'}
0.549157 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.2, 'init':
'glorot_uniform'}
0.547299 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.2, 'init': 'normal'}
0.545869 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.2, 'init':
'uniform'}
0.547616 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.3, 'init':
'glorot_uniform'}
0.555956 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.3, 'init': 'normal'}
0.548747 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.3, 'init':
'uniform'}
0.552946 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.4, 'init':
'glorot_uniform'}
0.550035 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.4, 'init': 'normal'}
0.550389 (0.000000) with: {'act_func': 'tanh', 'batch_size': 10, 'dropout': 0.4, 'init':
'uniform'}
0.555831 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.1, 'init':
'glorot_uniform'}
0.539495 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.1, 'init': 'normal'}
0.548627 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.1, 'init':
'uniform'}
0.541174 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.2, 'init':
'glorot_uniform'}
0.558481 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.2, 'init': 'normal'}
0.497519 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.2, 'init':
'uniform'}
0.559311 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.3, 'init':
'glorot_uniform'}
0.554451 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.3, 'init': 'normal'}
0.553339 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.3, 'init':
'uniform'}
0.550692 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.4, 'init':
'glorot_uniform'}
```

```
0.557567 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.4, 'init': 'normal'}
0.544810 (0.000000) with: {'act_func': 'tanh', 'batch_size': 20, 'dropout': 0.4, 'init':
'uniform'}
0.423350 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.1, 'init':
'glorot_uniform'}
0.368515 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.1, 'init': 'normal'}
0.466682 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.1, 'init': 'uniform'}
0.291897 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.2, 'init':
'glorot_uniform'}
0.474281 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.2, 'init': 'normal'}
0.131836 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.2, 'init': 'uniform'}
0.474855 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.3, 'init':
'glorot_uniform'}
0.425045 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.3, 'init': 'normal'}
0.482757 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.3, 'init': 'uniform'}
0.517481 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.4, 'init':
'glorot_uniform'}
0.390409 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.4, 'init': 'normal'}
0.485513 (0.000000) with: {'act_func': 'relu', 'batch_size': 5, 'dropout': 0.4, 'init': 'uniform'}
0.365072 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.1, 'init':
'glorot_uniform'}
0.400959 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.1, 'init': 'normal'}
0.348765 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.1, 'init':
'uniform'}
0.436931 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.2, 'init':
'glorot_uniform'}
0.436363 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.2, 'init': 'normal'}
0.472437 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.2, 'init':
'uniform'}
0.517868 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.3, 'init':
'glorot_uniform'}
0.474401 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.3, 'init': 'normal'}
0.485187 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.3, 'init':
'uniform'}
0.527972 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.4, 'init':
'glorot_uniform'}
0.496651 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.4, 'init': 'normal'}
0.418151 (0.000000) with: {'act_func': 'relu', 'batch_size': 10, 'dropout': 0.4, 'init':
'uniform'}
0.377919 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.1, 'init':
'glorot_uniform'}
0.467957 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.1, 'init': 'normal'}
0.273387 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.1, 'init':
'uniform'}
0.433345 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.2, 'init':
'glorot_uniform'}
0.418988 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.2, 'init': 'normal'}
0.454751 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.2, 'init':
'uniform'}
0.429461 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.3, 'init':
'glorot_uniform'}
0.457475 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.3, 'init': 'normal'}
0.502086 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.3, 'init':
'uniform'}
0.527533 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.4, 'init':
'glorot_uniform'}
0.422391 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.4, 'init': 'normal'}
0.392919 (0.000000) with: {'act_func': 'relu', 'batch_size': 20, 'dropout': 0.4, 'init':
'uniform'}
```

Fitting the best NN model according to Gridsearch

In [0]:

```
#gridsearch best params
best={'act_func': 'tanh', 'batch_size': 5, 'dropout': 0.4, 'init': 'glorot_uniform'}
net=KerasRegressor(build_fn= simple_net,verbose=False,epochs=30,**best)
net.fit(ss_train,y_train)
subm = pd.DataFrame()
subm['ID'] =test.ID
subm['y'] = net.predict(ss_test)
subm.to_csv('NN_final.csv', index=False)
```

LB(0.53706 0.55288) CV: 5657

Storing the Best Model So Far

In [0]:

```
pkl_filename = "/content/drive/My Drive/mercedes-benz-greener-manufacturing/pickle_model.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(stack, file)
```

Inference

In [0]:

```
pkl_filename = "/content/drive/My Drive/mercedes-benz-greener-manufacturing/pickle_model.pkl"
with open(pkl_filename, 'rb') as file:
    pickle_model = pickle.load(file)
```

```
[14:36:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[14:36:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
```

In [0]:

```
test= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/test_label.csv')
train= pd.read_csv('/content/drive/My Drive/mercedes-benz-greener-manufacturing/train_label.csv')

y_train= train.y.values
targets= y_train
train.drop(['y'],inplace= True,axis=1)
```

In [0]:

```
x_test= np.array(test)
ids_test= test.ID
y_pred = pickle_model.predict(x_test)
subm = pd.DataFrame()
subm['ID'] = ids_test
subm['y'] = y_pred
subm.to_csv('pickle_submission.csv', index=False)
```

Conclusion

In [34]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Categorical_encoding",'CV', "PublicLB","PrivateLB"]
x.add_row(["XGB_pca",'label',.5972, .5615,.5463])
x.add_row(["LinearReg",'one_hot',.5830, .5325,.5378])
x.add_row(["LinearReg",'mean/tar',.5838, .5438,.5379])
x.add_row(["RandomForest",'mean/tar',.6055, .5586,.5489])
x.add_row(["XGboost",'mean/tar',.6081, .5570,.5483])
x.add_row(["ExtraTrees",'mean/tar',.6009, .5554,.5483])
x.add_row(["RandomForest",'label',.6023, .5568,.5498])
x.add_row(["XGboost",'label',.6042, .5555,.5503])
x.add_row(["ExtraTrees",'label',.5986, .5537,.5470])
x.add_row(["***Stacked(et,rf,xgb)***",'label',.6052, .5579,.5522])
x.add_row(["Stack+interactions(et,rf,xgb)",'label',.6044, .5573,.5519])
x.add_row(['SVR+TSVD','one_hot',.5567,.5157,.5021])
x.add_row(['SVR+selectKBest','one_hot',.5639,.5167,.5079])
x.add_row(['NeuralNet','one_hot',.5657,0.5528,0.5370])
print(x)
```

```
+-------------------------------+----------------------+--------+----------+-----------+
|             Model             | Categorical_encoding |   CV   | PublicLB | PrivateLB |
+-------------------------------+----------------------+--------+----------+-----------+
|            XGB_pca            |        label         | 0.5972 |  0.5615  |   0.5463  |
```

LinearReg	one_hot	0.583	0.5325	0.5378
LinearReg	mean/tar	0.5838	0.5438	0.5379
RandomForest	mean/tar	0.6055	0.5586	0.5489
XGboost	mean/tar	0.6081	0.557	0.5483
ExtraTrees	mean/tar	0.6009	0.5554	0.5483
RandomForest	label	0.6023	0.5568	0.5498
XGboost	label	0.6042	0.5555	0.5503
ExtraTrees	label	0.5986	0.5537	0.547
Stacked(et,rf,xgb)	label	0.6052	0.5579	0.5522
Stack+interactions(et,rf,xgb)	label	0.6044	0.5573	0.5519
SVR+TSVD	one_hot	0.5567	0.5157	0.5021
SVR+selectKBest	one_hot	0.5639	0.5167	0.5079
NeuralNet	one_hot	0.5657	0.5528	0.537

Overview of the Project

DataExploration observations

369columns ,4209 rows,no missing values present There are 20-25 outliers,duplicate rows,along with columns present.Correlation between some features is really high,Pca seems to be of some Importance.

DataPreperation

Removal of features with <.01 variance,clipping of outliers to 150,dataset with label encoding ,one-hot,mean encoding of categorical variables done.

Modelling

CV used was of 15 folds with 5 fold,3 repeats random_seed with 3 used throughout for reproducibility.

1. Baseline was with 12 pca components appended to label encoded datasetwith xgboost on top, averaging of 15 cv models is done along with one trained on whole dataset.
2. LinearRegression with one-hot encoding and mean enconding done. t-test was used to check on cv folds whether ID feature really made any statistical significance on CV folds or not. Found out it doesen't.
3. Mean encoding was done on RandomForestRegressor,XGBRegressor,ExtraTreesRegressor with their respective hyperparameter tuning.
4. Label encoding dataset used with all the three models with tuning(RF,XGB,ExtraTrees)
5. Stacking done on above tree based models with a ridge Regressor on top.
6. New features are tried based on their Spearman correlation with the target variable.
7. Linear SVR and kernel SVR are modelled on one_hot encoding, here to reduce dimensionality TSVD and SelectKBest, both were tried out.
8. Bayesian Optimization library has been used to further finely tune the tree based models.
9. A Deeplearning model with a simple architecture has been created on the one-hot encoded dataset.
10. Storing the best model of all the above.
11. Checking for reproducibility of results.
12. Conclusion with pretty table with PublicLB,PrivateLB,CV scores of above mentioned models.

In [0]:

```

```