Research Project Name:

Feature Engineering using Signal Processing to predict DTC and DTS values

In this project we aim to find out the effect of Wavelet Transformation a signal Processing technique on DTC and DTS velocities prediction

Timeline for the project:

- DTC and DTS prediction using normal log data
- Wavelet Transforamtion is performed on the log data
- DTC and DtS prediction using transformed data

1) DTC and DTS prediction usign XG Boost Regression model

This will include the following steps:

- Imports
- Data exploration and Prepration
- Model Development
- Model Prediction and evaluation

Data is taken from Synthethic Sonic Log Curves Generation Contest

Description fo the data:

- CAL - Caliper, unit in Inch,
- CNC - Neutron, unit in dec
- GR - Gamma Ray, unit in API
- HRD - Deep Resisitivity, unit in Ohm per meter,
- HRM - Medium Resistivity, unit in Ohm per meter,
- PE - Photo-electric Factor, unit in Barn,
- ZDEN - Density, unit in Gram per cubit meter,
- DTC - Compressional Travel-time, unit in nanosecond per foot,
- DTS - Shear Travel-time, unit in nanosecond per foot,

Importing libraries and dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score

import xgboost as xgb
```

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv("test.csv")
```

```
print(train_df.shape,test_df.shape)
```

```
(30143, 9) (11088, 7)
```

```
train_df.describe()
```

	CAL	CNC	GR	HRD	HRM	PE
count	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000
mean	-8.394576	-23.692615	38.959845	3.977690	1.547299	-17.446739
std	129.970219	157.142679	108.504554	365.112753	456.908969	149.083136
min	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000
25%	8.058350	0.122800	17.248750	0.717700	0.712050	0.053100
50%	8.625000	0.193600	36.821800	1.623000	1.628100	4.941500
75%	9.048850	0.337150	58.346150	3.158300	3.280600	7.856650
max	21.064200	3490.158200	1470.253400	10000.000000	60467.761700	28.106400

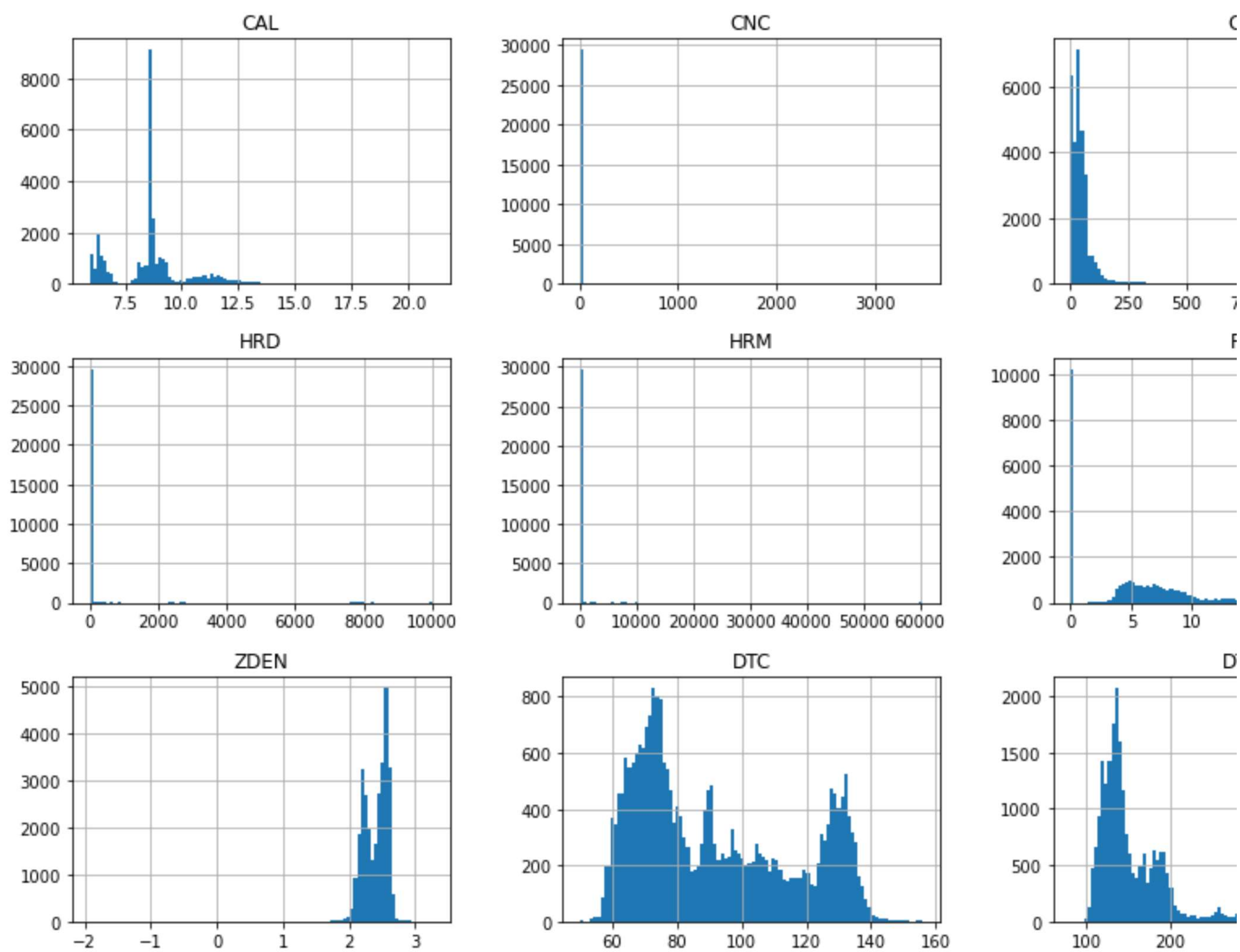
```
test_df.describe()
```

	CAL	CNC	GR	HRD	HRM	PE
count	11088.000000	11088.000000	11088.000000	11088.000000	11088.000000	11088.000000
mean	8.634049	0.158501	28.966414	4.028372	106.752210	7.353522
std	0.044064	0.091298	43.648163	7.198112	2374.620246	1.239075
min	8.500000	0.009800	0.852000	0.083900	0.102700	4.760800

```
df= train_df.copy()
```

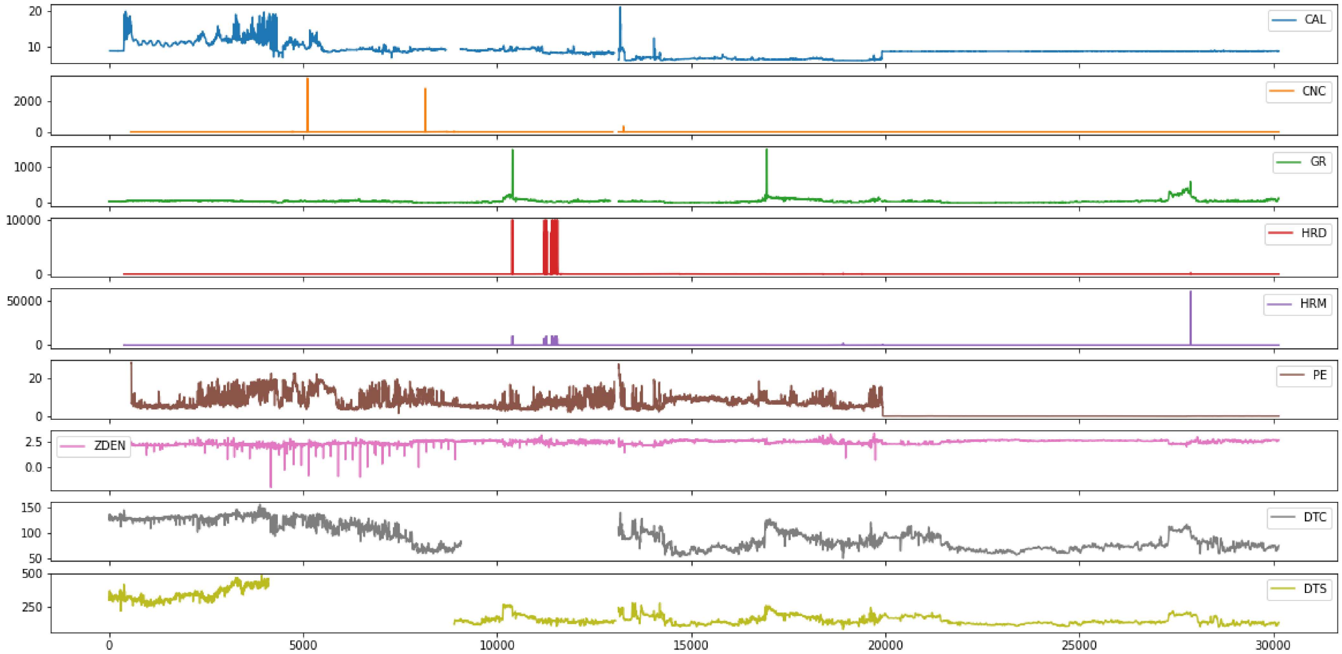
```
# Replace value -999 ( missing value indicators ) as NA
df.replace(['-999', -999], np.nan, inplace=True)
```

```
histdf = df.hist(bins=100,figsize=(15,10))
```




```
df.plot(subplots=True,figsize=(20,10))
```

```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>], dtype=object)
```



Making all the negative values as null values

```
df['ZDEN'][df['ZDEN']<0] = np.nan
df['GR'][df['GR']<0] = np.nan
df['CNC'][df['CNC']<0] = np.nan
df['PE'][df['PE']<0] = np.nan
```

Making all the outliers as null values

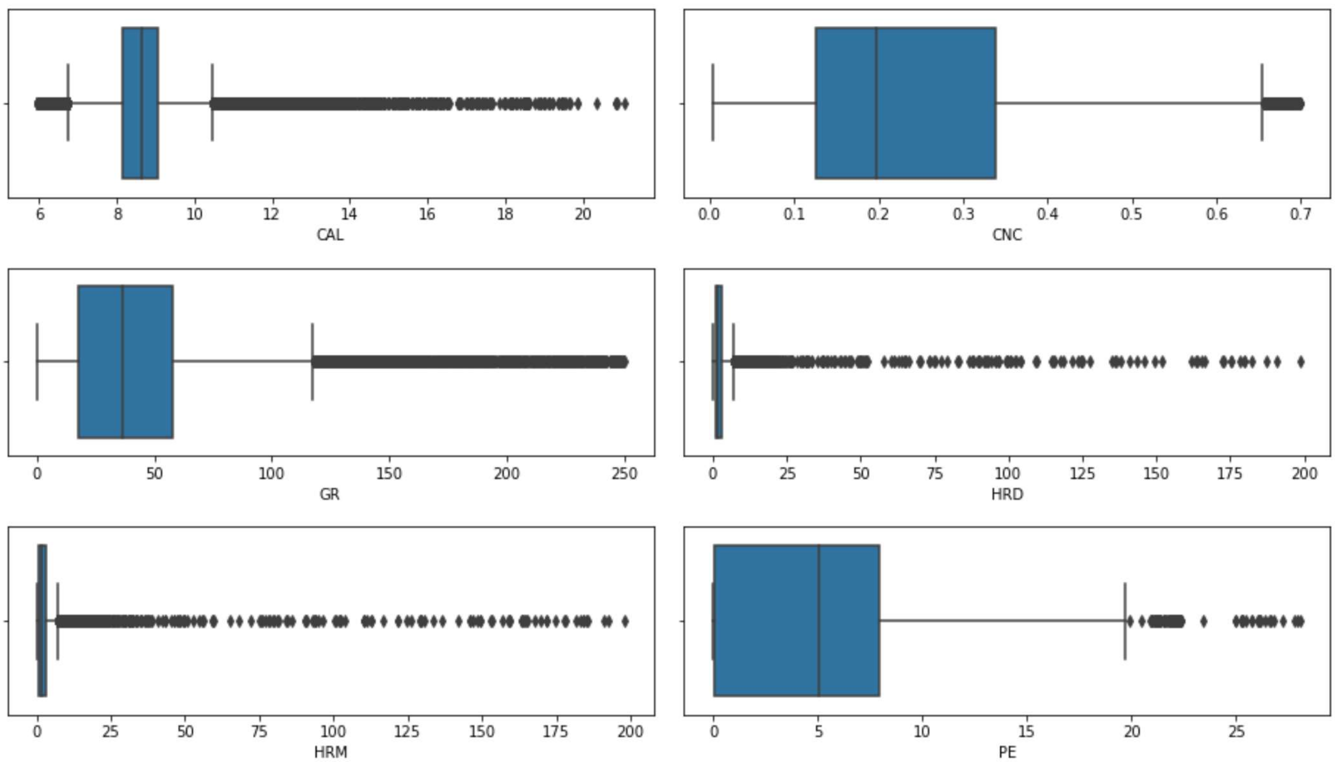
```
# GR
df['GR'][(df['GR']>250)] = np.nan
# CNC
df['CNC'][df['CNC']>0.7] = np.nan
```

```
# HRM & HRD
df['HRD'][df['HRD']>200] = np.nan
df['HRM'][df['HRM']>200] = np.nan
```

Plotting a box plot for visualising the data

```
plt.figure(figsize=(13,10))
plt.subplot(4,2,1)
sns.boxplot(df['CAL'])

plt.subplot(4,2,2)
sns.boxplot(df['CNC'])

plt.subplot(4,2,3)
sns.boxplot(df['GR'])

plt.subplot(4,2,4)
sns.boxplot(df['HRD'])

plt.subplot(4,2,5)
sns.boxplot(df['HRM'])

plt.subplot(4,2,6)
sns.boxplot(df['PE'])

plt.subplot(4,2,7)
sns.boxplot(df['ZDEN'])

plt.tight_layout(1.7)
plt.show()
```

```
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
<ipython-input-12-8851a3c48e82>:23: MatplotlibDeprecationWarning: Passing the pad parame
  plt.tight_layout(1.7)
```



Transforming the data

```
df['HRM'] = df['HRM'].apply(lambda x:np.log(x))
df['HRD'] = df['HRD'].apply(lambda x:np.log(x))
```

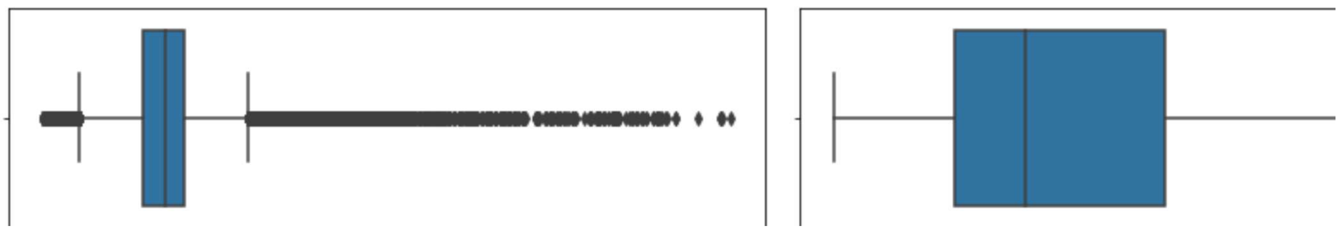
ZDEN

Plotting box plot after transformation

```
plt.figure(figsize=(13,10))
plt.subplot(4,2,1)
sns.boxplot(df['CAL'])

plt.subplot(4,2,2)
sns.boxplot(df['CNC'])
```

```
plt.subplot(4,2,3)
sns.boxplot(df['GR'])

plt.subplot(4,2,4)
sns.boxplot(df['HRD'])

plt.subplot(4,2,5)
sns.boxplot(df['HRM'])

plt.subplot(4,2,6)
sns.boxplot(df['PE'])

plt.subplot(4,2,7)
sns.boxplot(df['ZDEN'])

plt.tight_layout(1.7)
plt.show()
```

```
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
C:\Users\khank\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
  warnings.warn(
<ipython-input-14-8851a3c48e82>:23: MatplotlibDeprecationWarning: Passing the pad parame
  plt.tight_layout(1.7)
```



Finding Correlations

[ ] ↳ *2 cells hidden*

Predictive Power Score

[ ] ↳ *3 cells hidden*

Model Development

[ ] ↳ *9 cells hidden*

We will use XB Boost model with tuned Hyperparameters for prediction

[ ] ↳ *8 cells hidden*

Predictions for test data

[ ] ↳ *18 cells hidden*

Using PyCaret

[ ] ↳ *12 cells hidden*

2) Wavelet Transformation

[ ] ↳ *10 cells hidden*

First we will do everything for our train data

[ ] ↳ *6 cells hidden*

db4 wavelet transformation

[ ] ↳ *18 cells hidden*

Predictions for test data

[ ] ↳ *26 cells hidden*

Trying wavelet on Caliper log for dts prediction

[ ] ↳ *10 cells hidden*

Final result plots

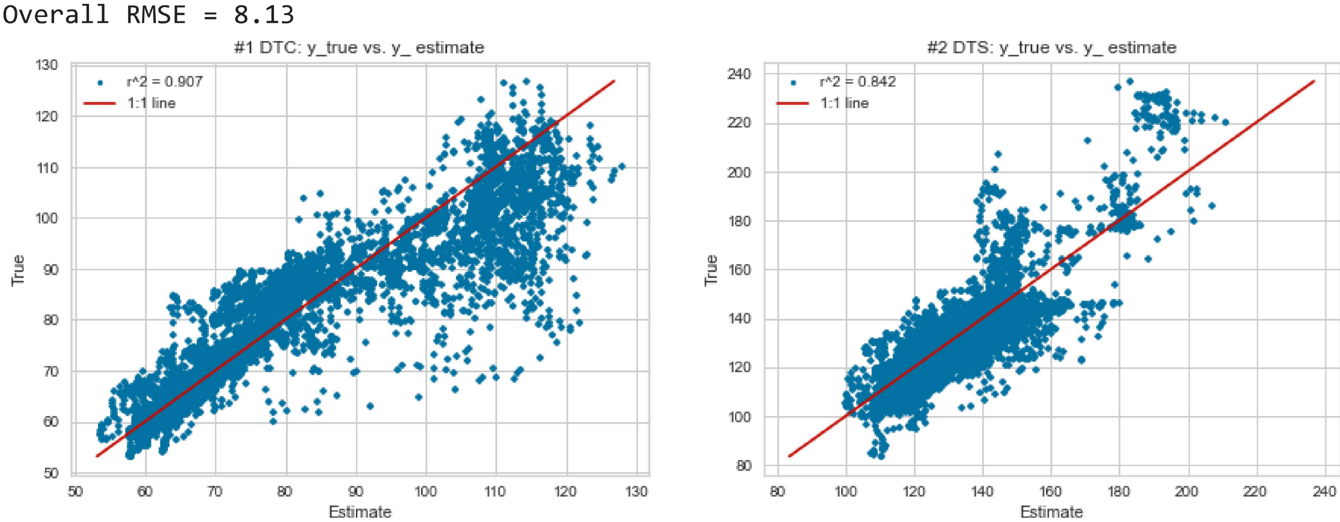
```
# check the accuracy of predicted data and plot the result
#print('Combined r2 score is:', '{:.5f}'.format((r2_score(y_real, y_predict))))
dtc_real = reals_wave[:, 0]
dtc_pred = preds_wave[:, 0]
dts_real = testing_data_cal['DTS  ']
dts_pred = final_test_df_dts_cal['DTS']
print('DTC:', '{:.5f}'.format(np.sqrt(r2_score(dtc_real, dtc_pred))))
print('DTS:', '{:.5f}'.format(np.sqrt(r2_score(dts_real, dts_pred))))
plt.subplots(nrows=2, ncols=2, figsize=(16,10))
plt.subplot(2, 2, 1)
plt.plot(reals_wave[:, 0])
plt.plot(preds_wave[:, 0])
plt.legend(['True', 'Predicted'])
plt.xlabel('Sample')
plt.ylabel('DTC')
plt.title('DTC Prediction Comparison')

plt.subplot(2, 2, 2)
```

```
plt.plot(testing_data_cal['DTS  '])
plt.plot(final_test_df_dts_cal['DTS'])
plt.legend(['True', 'Predicted'])
plt.xlabel('Sample')
plt.ylabel('DTS')
plt.title('DTS Prediction Comparison')

plt.subplot(2, 2, 3)
plt.scatter(reals_wave[:, 0], preds_wave[:, 0])
plt.xlabel('Real Value')
plt.ylabel('Predicted Value')
plt.title('DTC Prediction Comparison')

plt.subplot(2, 2, 4)
plt.scatter(testing_data_cal['DTS  '], final_test_df_dts_cal['DTS'])
plt.xlabel('Real Value')
plt.ylabel('Predicted Value')
plt.title('DTS Prediction Comparison')

plt.show()
```

```
DTC: 0.90677
DTS: 0.84168
```


```
# Plot results:
plt.figure(figsize=(15,5))
i = 0
plt.subplot(1,2,i+1)
plt.plot(preds_wave[:,i], reals_wave[:,i], '.', label = 'r^2 = %.3f' % (np.sqrt(r2_score(real
plt.plot([reals_wave[:,i].min(),reals_wave[:,i].max()],[reals_wave[:,i].min(),reals_wave[:,i]
plt.title('#1 DTC: y_true vs. y_ estimate'); plt.xlabel('Estimate'); plt.ylabel('True')
plt.legend()
i += 1
plt.subplot(1,2,i+1)
plt.plot(final_test_df_dts_cal['DTS'], testing_data_cal['DTS  '], '.', label = 'r^2 = %.3f' %
plt.plot([testing_data_cal['DTS  '].min(),testing_data_cal['DTS  '].max()],[testing_data_cal[
plt.title('#2 DTS: y_true vs. y_ estimate'); plt.xlabel('Estimate'); plt.ylabel('True')
plt.legend()

MSE_0 = mean_squared_error(reals_wave[:,0], preds_wave[:,0]);
RMSE_0 = np.sqrt(mean_squared_error(reals_wave[:,0], preds_wave[:,0]));
MSE_1 = mean_squared_error(testing_data_cal['DTS  '], final_test_df_dts_cal['DTS']);
RMSE_1 = np.sqrt(mean_squared_error(testing_data_cal['DTS  '], final_test_df_dts_cal['DTS']))
print('RMSE of test data (#1 DTC): %.2f' %(RMSE_0))
print('RMSE of test data (#2 DTS): %.2f' %(RMSE_1))
print('Overall RMSE = %.2f' %np.sqrt((MSE_0+MSE_1)/2))
```

```
RMSE of test data (#1 DTC): 6.11
RMSE of test data (#2 DTS): 9.75
Overall RMSE = 8.13
```




```
meandtc= np.sqrt(r2_score(dtc_real, dtc_pred))
meandts= np.sqrt(r2_score(dts_real, dts_pred))
```

```
test_list= [meandtc,meandts]
sum = 0
for ele in test_list:
  sum += ele
res = sum / len(test_list)


print("Combined r2 after wavelet transformation is :",res)
```

```
Combined r2 after wavelet transformation is : 0.8742239294641188
```