In [0]:

```
!pip  install catboost
```

```
Collecting catboost
  Downloading
https://files.pythonhosted.org/packages/5a/8a/a867c35770291646b085e9248814eb32dbe2aa824715b08e40cd9
83e/catboost-0.15.1-cp36-none-manylinux1_x86_64.whl (61.0MB)
     |████████████████████████████████| 61.1MB 1.4MB/s
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from
catboost) (1.16.4)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catboost)
(1.12.0)
Requirement already satisfied: pandas>=0.19.1 in /usr/local/lib/python3.6/dist-packages (from
catboost) (0.24.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from catboost)
(0.10.1)
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python3.6/dist-packages
(from pandas>=0.19.1->catboost) (2.5.3)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages (from
pandas>=0.19.1->catboost) (2018.9)
Installing collected packages: catboost
Successfully installed catboost-0.15.1
```

In [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import gc
import os
import time
import logging
import datetime
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
import lightgbm as lgb
from scipy import stats
from scipy.signal import hann
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle


import lightgbm as lgb
import xgboost as xgb
```

```
import xgboost as xgb
import time
import datetime
from catboost import CatBoostRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, KFold, RepeatedKFold
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
import gc
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from scipy.signal import hilbert
from scipy.signal import hann
from scipy.signal import convolve
from scipy import stats
from sklearn.kernel_ridge import KernelRidge
from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:

```
from googleapiclient.discovery import build
import io, os
from googleapiclient.http import MediaIoBaseDownload
from google.colab import auth

auth.authenticate_user()

drive_service = build('drive', 'v3')
results = drive_service.files().list(
        q="name = 'kaggle.json'", fields="files(id)").execute()
kaggle_api_key = results.get('files', [])

filename = "/content/.kaggle/kaggle.json"
os.makedirs(os.path.dirname(filename), exist_ok=True)

request = drive_service.files().get_media(fileId=kaggle_api_key[0]['id'])
fh = io.FileIO(filename, 'wb')
downloader = MediaIoBaseDownload(fh, request)
done = False
while done is False:
    status, done = downloader.next_chunk()
    print("Download %d%%." % int(status.progress() * 100))
os.chmod(filename, 600)
```

```
Download 100%.
```

In [0]:

```
!mkdir ~/.kaggle
!cp /content/.kaggle/kaggle.json ~/.kaggle/kaggle.json
```

In [0]:

```
!cp /content/gdrive/My\ Drive/kaggle.json ~/.kaggle/kaggle.json
```

In [0]:

```
!kaggle competitions download -c LANL-Earthquake-Prediction
```

```
Downloading sample_submission.csv to /content
  0% 0.00/33.3k [00:00<?, ?B/s]
100% 33.3k/33.3k [00:00<00:00, 27.2MB/s]
Downloading test.zip to /content
 96% 233M/242M [00:01<00:00, 122MB/s]
100% 242M/242M [00:01<00:00, 143MB/s]
Downloading train.csv.zip to /content
100% 2.02G/2.03G [00:37<00:00, 52.2MB/s]
100% 2.03G/2.03G [00:37<00:00, 58.6MB/s]
```

In [0]:

```
import pandas as pd
import numpy as np
train = pd.read_csv("train.csv",dtype={'acoustic_data':np.int16,'time_to_failure':np.float32})
```

In [0]:

```
train.shape
```

Out[0]:

```
(629145480, 2)
```

In [0]:

```
rows = 150000
segments = int(np.floor(train.shape[0] / rows))
print(segments)
```

```
4194
```

In [0]:

```
X_tr = pd.DataFrame(index=range(segments), dtype=np.float64,
                        columns=['ave', 'std', 'max', 'min',
                                 'av_change_abs', 'av_change_rate', 'abs_max', 'abs_min',
                                 'std_first_50000', 'std_last_50000', 'std_first_10000', 'std_last_10
00',
                                 'avg_first_50000', 'avg_last_50000', 'avg_first_10000', 'avg_last_10
00',
                                 'min_first_50000', 'min_last_50000', 'min_first_10000', 'min_last_10
00',
                                 'max_first_50000', 'max_last_50000', 'max_first_10000', 'max_last_10
00'])
```

In [0]:

```
from sklearn.linear_model import LinearRegression
def add_trend_feature(arr, abs_values=False):
    """Fit a univariate linear regression and return the coefficient."""
    idx = np.array(range(len(arr)))
    if abs_values:
        arr = np.abs(arr)
    lr = LinearRegression()
    lr.fit(idx.reshape(-1, 1), arr)
    return lr.coef_[0]
```

In [0]:

```
def classic_sta_lta(x, length_sta, length_lta):
    sta = np.cumsum(x ** 2)
    # Convert to float
    sta = np.require(sta, dtype=np.float)
    # Copy for LTA
    lta = sta.copy()
    # Compute the STA and the LTA
    sta[length_sta:] = sta[length_sta:] - sta[:-length_sta]
    sta /= length_sta
    lta[length_lta:] = lta[length_lta:] - lta[:-length_lta]
    lta /= length_lta
    # Pad zeros
    sta[:length_lta - 1] = 0
    # Avoid division by zero by setting zero values to tiny float
    dtiny = np.finfo(0.0).tiny
    idx = lta < dtiny
    lta[idx] = dtiny
    return sta / lta
```

In [0]:

```
from joblib import Parallel, delayed
import scipy as sp
import itertools
import gc

import librosa
import pywt
```

In [0]:

```
!pip install tsfresh
```

```
Collecting tsfresh
  Downloading
https://files.pythonhosted.org/packages/7f/67/841f3620083ce2611fb6020e0f7567caaccd73a4a8d635e674abc
f9b/tsfresh-0.11.2-py2.py3-none-any.whl (119kB)
     |████████████████████████████████| 122kB 2.8MB/s
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (1.12.0)
Requirement already satisfied: requests>=2.9.1 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (2.21.0)
Requirement already satisfied: scikit-learn>=0.19.0 in /usr/local/lib/python3.6/dist-packages
(from tsfresh) (0.21.2)
Collecting pandas<=0.23.4,>=0.20.3 (from tsfresh)
  Downloading
https://files.pythonhosted.org/packages/e1/d8/feeb346d41f181e83fba45224ab14a8d8af019b48af742e047f38
cff/pandas-0.23.4-cp36-cp36m-manylinux1_x86_64.whl (8.9MB)
     |████████████████████████████████| 8.9MB 9.2MB/s
Requirement already satisfied: statsmodels>=0.8.0 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (0.9.0)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (1.16.4)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (1.3.0)
Requirement already satisfied: future>=0.16.0 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (0.16.0)
Requirement already satisfied: distributed>=1.18.3 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (1.25.3)
Requirement already satisfied: patsy>=0.4.1 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (0.5.1)
Requirement already satisfied: dask>=0.15.2 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (1.1.5)
Requirement already satisfied: tqdm>=4.10.0 in /usr/local/lib/python3.6/dist-packages (from
tsfresh) (4.28.1)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages
(from requests>=2.9.1->tsfresh) (1.24.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests>=2.9.1->tsfresh) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests>=2.9.1->tsfresh) (2019.3.9)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages
(from requests>=2.9.1->tsfresh) (3.0.4)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from
```

```
scikit-learn>=0.19.0->tsfresh) (0.13.2)
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/local/lib/python3.6/dist-packages
(from pandas<=0.23.4,>=0.20.3->tsfresh) (2.5.3)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages (from
pandas<=0.23.4,>=0.20.3->tsfresh) (2018.9)
Requirement already satisfied: tornado>=4.5.1 in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (4.5.3)
Requirement already satisfied: psutil>=5.0 in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (5.4.8)
Requirement already satisfied: sortedcontainers!=2.0.0,!=2.0.1 in /usr/local/lib/python3.6/dist-pa
ckages (from distributed>=1.18.3->tsfresh) (2.1.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (3.13)
Requirement already satisfied: cloudpickle>=0.2.2 in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (0.6.1)
Requirement already satisfied: toolz>=0.7.4 in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (0.9.0)
Requirement already satisfied: tblib in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (1.4.0)
Requirement already satisfied: click>=6.6 in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (7.0)
Requirement already satisfied: zict>=0.1.3 in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (0.1.4)
Requirement already satisfied: msgpack in /usr/local/lib/python3.6/dist-packages (from
distributed>=1.18.3->tsfresh) (0.5.6)
Requirement already satisfied: heapdict in /usr/local/lib/python3.6/dist-packages (from
zict>=0.1.3->distributed>=1.18.3->tsfresh) (1.0.0)
ERROR: google-colab 1.0.0 has requirement pandas~=0.24.0, but you'll have pandas 0.23.4 which is i
ncompatible.
Installing collected packages: pandas, tsfresh
  Found existing installation: pandas 0.24.2
    Uninstalling pandas-0.24.2:
      Successfully uninstalled pandas-0.24.2
Successfully installed pandas-0.23.4 tsfresh-0.11.2
```

In [0]:

```
from scipy import stats
from scipy.signal import hann
from tqdm import tqdm_notebook
import matplotlib.pyplot as plt
from scipy.signal import hilbert
from scipy.signal import convolve
from tsfresh.feature_extraction import feature_calculators
import librosa
import pywt
```

In [0]:

```
def maddest(d, axis=None):
    return np.mean(np.absolute(d - np.mean(d, axis)), axis)
```

In [0]:

```
def denoise_signal(x, wavelet='db4', level=1):
    coeff = pywt.wavedec(x, wavelet, mode="per")
    sigma = (1/0.6745) * maddest(coeff[-level])
    uthresh = sigma * np.sqrt(2*np.log(len(x)))
    coeff[1:] = (pywt.threshold(i, value=uthresh, mode='hard') for i in coeff[1:])

    return pywt.waverec(coeff, wavelet, mode='per')
```

In [0]:

```
def denoise_signal_simple(x, wavelet='db4', level=1):
    coeff = pywt.wavedec(x, wavelet, mode="per")
    #univeral threshold
    uthresh = 10
    coeff[1:] = (pywt.threshold(i, value=uthresh, mode='hard') for i in coeff[1:])
    # Reconstruct the signal using the thresholded coefficients
    return pywt.waverec(coeff, wavelet, mode='per')
```

In [0]:

```
np.random.seed(1337)
noise = np.random.normal(0, 0.5, 150_000)
seg = train.iloc[1*rows:1*rows+rows]
x = seg['acoustic_data'].values
y = seg['time_to_failure'].values
print(x)
x = x + noise
print(x)
x = x - np.median(x)
print(x)
x_denoised= denoise_signal_simple(x)
print(x_denoised)
```

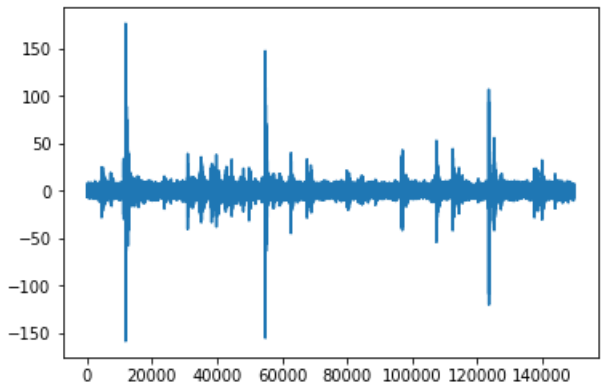
```
[5 6 8 ... 7 7 5]
[4.64840635 5.75485882 7.83909284 ... 7.43139967 7.1266968  5.02331202]
[-0.07301813  1.03343434  3.11766836 ...  2.70997519  2.40527232
  0.30188754]
[-0.01001956 -0.0101286  -0.01023926 ... -0.00972402 -0.00980837
 -0.00990623]
```

In [0]:

```
plt.plot(x)
```

Out[0]:

```
[<matplotlib.lines.Line2D at 0x7f70522b3e48>]
```

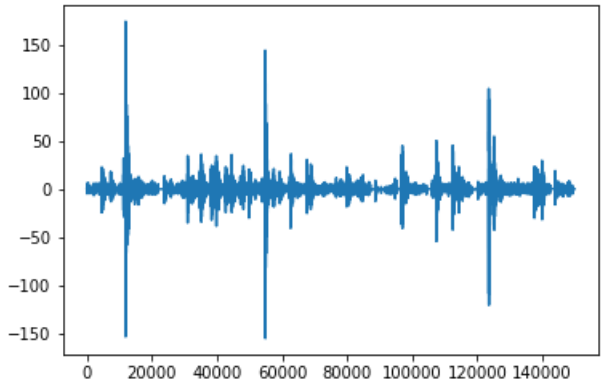


In [0]:

```
plt.plot(x_denoised)
```

Out[0]:

```
[<matplotlib.lines.Line2D at 0x7f705024c1d0>]
```



In [0]:

```
segments = int(np.floor(train.shape[0] / rows))
```

```
segments = int(np.floor(train.shape[0] / rows))

train_y = pd.DataFrame(index=range(segments), dtype=np.float64, columns=['time_to_failure'])
```

In [0]:

```
for segment in tqdm(range(segments)):
    seg = train.iloc[segment*rows:segment*rows+rows]
    train_y.loc[segment, 'time_to_failure'] = seg['time_to_failure'].values[-1]
```

```
100%|██████████| 4194/4194 [00:03<00:00, 1189.20it/s]
```

In [0]:

```
from tqdm import tqdm
from scipy.stats import kurtosis
from scipy.stats import skew


np.random.seed(1337)
noise = np.random.normal(0, 0.5, 150_000)


quake_count = 0
for segment in tqdm(range(segments)):
    seg = train.iloc[segment*rows:segment*rows+rows]
    x = seg['acoustic_data'].values


    x = x + noise

    x = x - np.median(x)




    zc = np.fft.fft(x)
    realFFT = np.real(zc)
    imagFFT = np.imag(zc)
    X_tr.loc[segment, 'Rmean'] = realFFT.mean()
    X_tr.loc[segment, 'Rstd'] = realFFT.std()
    X_tr.loc[segment, 'Rmax'] = realFFT.max()
    X_tr.loc[segment, 'Rmin'] = realFFT.min()
    X_tr.loc[segment, 'Imean'] = imagFFT.mean()
    X_tr.loc[segment, 'Istd'] = imagFFT.std()
    X_tr.loc[segment, 'Imax'] = imagFFT.max()
    X_tr.loc[segment, 'Imin'] = imagFFT.min()

    X_tr.loc[segment, 'ave'] = x.mean()
    X_tr.loc[segment, 'std'] = x.std()
    X_tr.loc[segment, 'max'] = x.max()
    X_tr.loc[segment, 'min'] = x.min()
    X_tr.loc[segment,'range'] = x.max() - x.min()
    X_tr.loc[segment,'std_to_mean'] = x.std()/x.mean()


    X_tr.loc[segment, 'av_change_abs'] = np.mean(np.diff(x))
    X_tr.loc[segment, 'av_change_rate'] = np.mean(np.nonzero((np.diff(x) / x[:-1]))[0])
    X_tr.loc[segment, 'abs_mean'] = np.abs(x).mean()
    X_tr.loc[segment, 'abs_max'] = np.abs(x).max()
    X_tr.loc[segment, 'abs_min'] = np.abs(x).min()
    X_tr.loc[segment, 'q01'] = np.quantile(x, 0.01)
    X_tr.loc[segment, 'q05'] = np.quantile(x, 0.05)
    X_tr.loc[segment, 'q10'] = np.quantile(x, 0.10)
    X_tr.loc[segment, 'q50'] = np.quantile(x, 0.50)
    X_tr.loc[segment, 'q90'] = np.quantile(x, 0.90)
    X_tr.loc[segment, 'q95'] = np.quantile(x, 0.95)
    X_tr.loc[segment, 'q99'] = np.quantile(x, 0.99)
    X_tr.loc[segment,'trend_feature'] = add_trend_feature(x)
    X_tr.loc[segment,'trend_feature_abs'] = add_trend_feature(x,abs_values=True)


    X_tr.loc[segment, 'std_first_50000'] = x[:50000].std()
```

```
    X_tr.loc[segment, 'std_last_50000'] = x[-50000:].std()
    X_tr.loc[segment, 'std_first_10000'] = x[:10000].std()
    X_tr.loc[segment, 'std_last_10000'] = x[-10000:].std()

    X_tr.loc[segment, 'avg_first_50000'] = x[:50000].mean()
    X_tr.loc[segment, 'avg_last_50000'] = x[-50000:].mean()
    X_tr.loc[segment, 'avg_first_10000'] = x[:10000].mean()
    X_tr.loc[segment, 'avg_last_10000'] = x[-10000:].mean()

    X_tr.loc[segment, 'min_first_50000'] = x[:50000].min()
    X_tr.loc[segment, 'min_last_50000'] = x[-50000:].min()
    X_tr.loc[segment, 'min_first_10000'] = x[:10000].min()
    X_tr.loc[segment, 'min_last_10000'] = x[-10000:].min()

    X_tr.loc[segment, 'max_first_50000'] = x[:50000].max()
    X_tr.loc[segment, 'max_last_50000'] = x[-50000:].max()
    X_tr.loc[segment, 'max_first_10000'] = x[:10000].max()
    X_tr.loc[segment, 'max_last_10000'] = x[-10000:].max()


    X_tr.loc[segment, 'kurt'] = kurtosis(x)
    X_tr.loc[segment, 'skew'] = skew(x)


    X_tr.loc[segment, 'Hilbert_mean'] = np.abs(hilbert(x)).mean()
    X_tr.loc[segment, 'Hann_window_mean'] = (convolve(x, hann(150), mode='same') / sum(hann(150))).
mean()
    X_tr.loc[segment, 'classic_sta_lta1_mean'] = classic_sta_lta(x, 500, 10000).mean()
    X_tr.loc[segment, 'classic_sta_lta2_mean'] = classic_sta_lta(x, 5000, 100000).mean()
    X_tr.loc[segment, 'classic_sta_lta3_mean'] = classic_sta_lta(x, 3333, 6666).mean()
    X_tr.loc[segment, 'classic_sta_lta4_mean'] = classic_sta_lta(x, 10000, 25000).mean()
    X_tr.loc[segment, 'mfcc_mean18'] = librosa.feature.mfcc(x).mean(axis=1)[18]
    X_tr.loc[segment, 'mfcc_mean14'] = librosa.feature.mfcc(x).mean(axis=1)[4]
    X_tr.loc[segment,'percentile_roll20_std_50'] = np.percentile(pd.Series(x).rolling(20).std().dro
pna().values, 50)
    X_tr.loc[segment,'num_peaks_2_denoise_simple'] =  feature_calculators.number_peaks(denoise_sign
al_simple(x), 2)

    X_tr.loc[segment,'LGBM_autocorr5'] = feature_calculators.autocorrelation(pd.Series(x), 5)
    X_tr.loc[segment,'zero_crossing']=len(np.where(np.diff(np.sign(x)))[0])

    X_tr.loc[segment, 'andersonDarling'] = 1 / (1.0 + np.exp(-10 * (stats.anderson(x)[0] - 0.3)))

    fftrhann20000 = np.sum(np.abs(np.fft.fft(np.hanning(len(x))*x)[:20000]))
    fftrhann20000_denoise = np.sum(np.abs(np.fft.fft(np.hanning(len(x))*denoise_signal(x))[:20000])
)
    fftrhann20000_diff_rate = (fftrhann20000 - fftrhann20000_denoise)/fftrhann20000

    X_tr.loc[segment,'LGBM_fftrhann20000_diff_rate'] = fftrhann20000_diff_rate
    X_tr.loc[segment, 'Moving_average_700_mean'] = pd.Series(x).rolling(window=700).mean().mean(ski
pna=True)


    X_tr.loc[segment, 'exp_Moving_average_300_mean'] = pd.Series(x).ewm(span=300).mean().mean(skipn
a=True)
    X_tr.loc[segment, 'exp_Moving_average_700_mean'] = pd.Series(x).ewm(span=700).mean().mean(skipn
a=True)
    X_tr.loc[segment, 'exp_Moving_average_1500_mean'] = pd.Series(x).ewm(span=1500).mean().mean(ski
pna=True)
    X_tr.loc[segment, 'exp_Moving_average_3000_mean'] = pd.Series(x).ewm(span=3000).mean().mean(ski
pna=True)
    X_tr.loc[segment, 'exp_Moving_average_5000_mean'] = pd.Series(x).ewm(span=5000).mean().mean(ski
pna=True)
    X_tr.loc[segment, 'exp_Moving_average_10000_mean'] = pd.Series(x).ewm(span=10000).mean().mean(s
kipna=True)
    X_tr.loc[segment, 'exp_Moving_average_30000_mean'] = pd.Series(x).ewm(span=30000).mean().mean(s
kipna=True)
    no_of_std = 3
    X_tr.loc[segment, 'MA_700MA_std_mean'] = pd.Series(x).rolling(window=700).std().mean()
    X_tr.loc[segment,'MA_700MA_BB_high_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] + no_
of_std * X_tr.loc[segment, 'MA_700MA_std_mean']).mean()
    X_tr.loc[segment,'MA_700MA_BB_low_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] - no_o
f_std * X_tr.loc[segment, 'MA_700MA_std_mean']).mean()
    X_tr.loc[segment, 'MA_400MA_std_mean'] = pd.Series(x).rolling(window=400).std().mean()
    X_tr.loc[segment,'MA_400MA_BB_high_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] + no_
of_std * X_tr.loc[segment, 'MA_400MA_std_mean']).mean()
    X_tr.loc[segment,'MA_400MA_BB_low_mean'] = (X_tr.loc[segment, 'Moving_average_700_mean'] - no_o
```

```
f_std * X_tr.loc[segment, 'MA_400MA_std_mean']).mean()
    X_tr.loc[segment, 'MA_1000MA_std_mean'] = pd.Series(x).rolling(window=1000).std().mean()

    X_tr.drop('Moving_average_700_mean', axis=1, inplace=True)

    x = pd.Series(x)



    for w in [10, 50, 100,250,500,750,1000]:

        x_roll_abs_mean = x.rolling(w).mean().dropna().values
        x_roll_mean = x.rolling(w).mean().dropna().values
        x_roll_std = x.rolling(w).std().dropna().values
        x_roll_min = x.rolling(w).min().dropna().values
        x_roll_max = x.rolling(w).max().dropna().values

        X_tr.loc[segment, 'ave_roll_std_' +str(w)] =x_roll_std.mean()
        X_tr.loc[segment, 'std_roll_std_' +str(w)] = x_roll_std.std()
        X_tr.loc[segment, 'max_roll_std_' +str(w)] = x_roll_std.max()
        X_tr.loc[segment, 'min_roll_std_' +str(w)] = x_roll_std.min()
        X_tr.loc[segment, 'q01_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.01)
        X_tr.loc[segment, 'q05_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.05)
        X_tr.loc[segment, 'q10_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.1)
        X_tr.loc[segment, 'q50_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.5)
        X_tr.loc[segment, 'q90_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.90)

        X_tr.loc[segment, 'q95_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.95)
        X_tr.loc[segment, 'q99_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.99)
        X_tr.loc[segment, 'kurtosis_roll_std' +str(w)] = kurtosis(x_roll_std)
        X_tr.loc[segment, 'skew_roll_std' +str(w)] = skew(x_roll_std)



        X_tr.loc[segment, 'ave_roll_mean_' +str(w)] =x_roll_mean.mean()
        X_tr.loc[segment, 'mean_roll_mean_' +str(w)] = x_roll_mean.std()
        X_tr.loc[segment, 'max_roll_mean_' +str(w)] = x_roll_mean.max()
        X_tr.loc[segment, 'min_roll_mean_' +str(w)] = x_roll_mean.min()
        X_tr.loc[segment, 'q01_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.01)
        X_tr.loc[segment, 'q05_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.05)
        X_tr.loc[segment, 'q10_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.1)
        X_tr.loc[segment, 'q50_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.5)
        X_tr.loc[segment, 'q90_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.9)
        X_tr.loc[segment, 'q95_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.95)
        X_tr.loc[segment, 'q99_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.99)
        X_tr.loc[segment, 'kurtosis_roll_mean' +str(w)] = kurtosis(x_roll_mean)
        X_tr.loc[segment, 'skew_roll_mean' +str(w)] = skew(x_roll_mean)



        X_tr.loc[segment, 'ave_roll_min_' +str(w)] =x_roll_min.mean()
        X_tr.loc[segment, 'min_roll_min_' +str(w)] = x_roll_min.std()
        X_tr.loc[segment, 'max_roll_min_' +str(w)] = x_roll_min.max()
        X_tr.loc[segment, 'min_roll_min_' +str(w)] = x_roll_min.min()
        X_tr.loc[segment, 'q01_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.01)
        X_tr.loc[segment, 'q05_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.05)
        X_tr.loc[segment, 'q10_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.1)
        X_tr.loc[segment, 'q50_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.5)
        X_tr.loc[segment, 'q90_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.9)
        X_tr.loc[segment, 'q95_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.95)
        X_tr.loc[segment, 'q99_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.99)
        X_tr.loc[segment, 'kurtosis_roll_min' +str(w)] = kurtosis(x_roll_min)
        X_tr.loc[segment, 'skew_roll_min' +str(w)] = skew(x_roll_min)



        X_tr.loc[segment, 'ave_roll_max_' +str(w)] =x_roll_max.mean()
        X_tr.loc[segment, 'max_roll_max_' +str(w)] = x_roll_max.std()
        X_tr.loc[segment, 'max_roll_max_' +str(w)] = x_roll_max.max()
        X_tr.loc[segment, 'max_roll_max_' +str(w)] = x_roll_max.min()
        X_tr.loc[segment, 'q01_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.01)
        X_tr.loc[segment, 'q05_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.05)
```

```
        X_tr.loc[segment, 'q10_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.1)
        X_tr.loc[segment, 'q50_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.5)
        X_tr.loc[segment, 'q90_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.9)
        X_tr.loc[segment, 'q95_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.95)
        X_tr.loc[segment, 'q99_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.99)
        X_tr.loc[segment, 'kurtosis_roll_max' +str(w)] = kurtosis(x_roll_max)
        X_tr.loc[segment, 'skew_roll_max' +str(w)] = skew(x_roll_max)
```

```
100%|██████████| 4194/4194 [2:18:40<00:00,  1.95s/it]
```

In [0]:

```
X_tr['time_to_failure'] = train_y
```

In [0]:

```
import pickle
X_tr.to_pickle('gdrive/My Drive/X_tr_LANL_noise')
```

In [0]:

```
X_tr.columns
```

Out[0]:

```
Index(['ave', 'std', 'max', 'min', 'av_change_abs', 'av_change_rate',
       'abs_max', 'abs_min', 'std_first_50000', 'std_last_50000',
       ...
       'q01_roll_max_1000', 'q05_roll_max_1000', 'q10_roll_max_1000',
       'q50_roll_max_1000', 'q90_roll_max_1000', 'q95_roll_max_1000',
       'q99_roll_max_1000', 'kurtosis_roll_max1000', 'skew_roll_max1000',
       'time_to_failure'],
      dtype='object', length=418)
```

In [0]:

```
X_tr = pd.read_pickle('gdrive/My Drive/X_tr_LANL_noise')
```

In [0]:

```
X_tr.columns
```

Out[0]:

```
Index(['ave', 'std', 'max', 'min', 'av_change_abs', 'av_change_rate',
       'abs_max', 'abs_min', 'std_first_50000', 'std_last_50000',
       ...
       'q01_roll_max_1000', 'q05_roll_max_1000', 'q10_roll_max_1000',
       'q50_roll_max_1000', 'q90_roll_max_1000', 'q95_roll_max_1000',
       'q99_roll_max_1000', 'kurtosis_roll_max1000', 'skew_roll_max1000',
       'time_to_failure'],
      dtype='object', length=418)
```

In [0]:

```
train_y = X_tr['time_to_failure']
```

In [0]:

```
X_tr = X_tr.drop(['time_to_failure'],axis=1)
```

In [0]:

```
means_dict = {}
for col in X_tr.columns:
    if X_tr[col].isnull().any():
```

```
        print(col)
        mean_value = X_tr.loc[X_tr[col] != -np.inf, col].mean()
        X_tr.loc[X_tr[col] == -np.inf, col] = mean_value
        X_tr[col] = X_tr[col].fillna(mean_value)
        means_dict[col] = mean_value
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_tr)
X_train_scaled = pd.DataFrame(scaler.transform(X_tr), columns=X_tr.columns)
```

In [0]:

```
#X_tr = X_tr.replace([np.inf, -np.inf], np.nan).dropna(axis=1)
```

In [0]:

```
submission = pd.read_csv('sample_submission.csv', index_col='seg_id')
```

In [0]:

```
X_test = pd.DataFrame(columns=X_tr.columns, dtype=np.float64, index=submission.index)
```

In [0]:

```
X_test.columns
```

Out[0]:

```
Index(['ave', 'std', 'max', 'min', 'av_change_abs', 'av_change_rate',
       'abs_max', 'abs_min', 'std_first_50000', 'std_last_50000',
       ...
       'q01_roll_max_1000', 'q05_roll_max_1000', 'q10_roll_max_1000',
       'q50_roll_max_1000', 'q90_roll_max_1000', 'q95_roll_max_1000',
       'q99_roll_max_1000', 'kurtosis_roll_max1000', 'skew_roll_max1000',
       'time_to_failure'],
      dtype='object', length=418)
```

In [0]:

```
X_test.shape
```

Out[0]:

```
(2624, 418)
```

In [0]:

```
from tqdm import  tqdm
np.random.seed(1337)
noise = np.random.normal(0, 0.5, 150_000)


for i, seg_id in enumerate(tqdm(X_test.index)):
    seg = pd.read_csv( seg_id +'.csv')

    x = seg['acoustic_data'].values

    x = x + noise

    x = x - np.median(x)




    zc = np.fft.fft(x)
    realFFT = np.real(zc)
```

```
    realFFT = np.real(zc)
    imagFFT = np.imag(zc)
    X_test.loc[seg_id, 'Rmean'] = realFFT.mean()
    X_test.loc[seg_id, 'Rstd'] = realFFT.std()
    X_test.loc[seg_id, 'Rmax'] = realFFT.max()
    X_test.loc[seg_id, 'Rmin'] = realFFT.min()
    X_test.loc[seg_id, 'Imean'] = imagFFT.mean()
    X_test.loc[seg_id, 'Istd'] = imagFFT.std()
    X_test.loc[seg_id, 'Imax'] = imagFFT.max()
    X_test.loc[seg_id, 'Imin'] = imagFFT.min()

    X_test.loc[seg_id, 'ave'] = x.mean()
    X_test.loc[seg_id, 'std'] = x.std()
    X_test.loc[seg_id, 'max'] = x.max()
    X_test.loc[seg_id, 'min'] = x.min()
    X_test.loc[seg_id,'range'] = x.max() - x.min()
    X_test.loc[seg_id,'std_to_mean'] = x.std()/x.mean()


    X_test.loc[seg_id, 'av_change_abs'] = np.mean(np.diff(x))
    X_test.loc[seg_id, 'av_change_rate'] = np.mean(np.nonzero((np.diff(x) / x[:-1]))[0])
    X_test.loc[seg_id, 'abs_mean'] = np.abs(x).mean()
    X_test.loc[seg_id, 'abs_max'] = np.abs(x).max()
    X_test.loc[seg_id, 'abs_min'] = np.abs(x).min()
    X_test.loc[seg_id, 'q01'] = np.quantile(x, 0.01)
    X_test.loc[seg_id, 'q05'] = np.quantile(x, 0.05)
    X_test.loc[seg_id, 'q10'] = np.quantile(x, 0.10)
    X_test.loc[seg_id, 'q50'] = np.quantile(x, 0.50)
    X_test.loc[seg_id, 'q90'] = np.quantile(x, 0.90)
    X_test.loc[seg_id, 'q95'] = np.quantile(x, 0.95)
    X_test.loc[seg_id, 'q99'] = np.quantile(x, 0.99)
    X_test.loc[seg_id,'trend_feature'] = add_trend_feature(x)
    X_test.loc[seg_id,'trend_feature_abs'] = add_trend_feature(x,abs_values=True)


    X_test.loc[seg_id, 'std_first_50000'] = x[:50000].std()
    X_test.loc[seg_id, 'std_last_50000'] = x[-50000:].std()
    X_test.loc[seg_id, 'std_first_10000'] = x[:10000].std()
    X_test.loc[seg_id, 'std_last_10000'] = x[-10000:].std()

    X_test.loc[seg_id, 'avg_first_50000'] = x[:50000].mean()
    X_test.loc[seg_id, 'avg_last_50000'] = x[-50000:].mean()
    X_test.loc[seg_id, 'avg_first_10000'] = x[:10000].mean()
    X_test.loc[seg_id, 'avg_last_10000'] = x[-10000:].mean()

    X_test.loc[seg_id, 'min_first_50000'] = x[:50000].min()
    X_test.loc[seg_id, 'min_last_50000'] = x[-50000:].min()
    X_test.loc[seg_id, 'min_first_10000'] = x[:10000].min()
    X_test.loc[seg_id, 'min_last_10000'] = x[-10000:].min()

    X_test.loc[seg_id, 'max_first_50000'] = x[:50000].max()
    X_test.loc[seg_id, 'max_last_50000'] = x[-50000:].max()
    X_test.loc[seg_id, 'max_first_10000'] = x[:10000].max()
    X_test.loc[seg_id, 'max_last_10000'] = x[-10000:].max()


    X_test.loc[seg_id, 'kurt'] = kurtosis(x)
    X_test.loc[seg_id, 'skew'] = skew(x)


    X_test.loc[seg_id, 'Hilbert_mean'] = np.abs(hilbert(x)).mean()
    X_test.loc[seg_id, 'Hann_window_mean'] = (convolve(x, hann(150), mode='same') / sum(hann(150)))
.mean()
    X_test.loc[seg_id, 'classic_sta_lta1_mean'] = classic_sta_lta(x, 500, 10000).mean()
    X_test.loc[seg_id, 'classic_sta_lta2_mean'] = classic_sta_lta(x, 5000, 100000).mean()
    X_test.loc[seg_id, 'classic_sta_lta3_mean'] = classic_sta_lta(x, 3333, 6666).mean()
    X_test.loc[seg_id, 'classic_sta_lta4_mean'] = classic_sta_lta(x, 10000, 25000).mean()
    X_test.loc[seg_id, 'mfcc_mean18'] = librosa.feature.mfcc(x).mean(axis=1)[18]
    X_test.loc[seg_id, 'mfcc_mean14'] = librosa.feature.mfcc(x).mean(axis=1)[4]
    X_test.loc[seg_id,'percentile_roll20_std_50'] = np.percentile(pd.Series(x).rolling(20).std().dr
opna().values, 50)
    X_test.loc[seg_id,'num_peaks_2_denoise_simple'] =  feature_calculators.number_peaks(denoise_sig
nal_simple(x), 2)

    X_test.loc[seg_id,'LGBM_autocorr5'] = feature_calculators.autocorrelation(pd.Series(x), 5)
    X_test.loc[seg_id,'zero_crossing']=len(np.where(np.diff(np.sign(x)))[0])

    X_test.loc[seg_id, 'andersonDarling'] = 1 / (1.0 + np.exp(-10 * (stats.anderson(x)[0] - 0.3)))
```

```
    X_test.loc[seg_id, 'andersonDarling'] = 1 / (1.0 + np.exp(-10 * (stats.anderson(x)[0] - 0.5)))

    fftrhann20000 = np.sum(np.abs(np.fft.fft(np.hanning(len(x))*x)[:20000]))
    fftrhann20000_denoise = np.sum(np.abs(np.fft.fft(np.hanning(len(x))*denoise_signal(x))[:20000])
)
    fftrhann20000_diff_rate = (fftrhann20000 - fftrhann20000_denoise)/fftrhann20000

    X_test.loc[seg_id,'LGBM_fftrhann20000_diff_rate'] = fftrhann20000_diff_rate
    X_test.loc[seg_id, 'Moving_average_700_mean'] = pd.Series(x).rolling(window=700).mean().mean(sk
ipna=True)


    X_test.loc[seg_id, 'exp_Moving_average_300_mean'] = pd.Series(x).ewm(span=300).mean().mean(skip
na=True)
    X_test.loc[seg_id, 'exp_Moving_average_700_mean'] = pd.Series(x).ewm(span=700).mean().mean(skip
na=True)
    X_test.loc[seg_id, 'exp_Moving_average_1500_mean'] = pd.Series(x).ewm(span=1500).mean().mean(sk
ipna=True)
    X_test.loc[seg_id, 'exp_Moving_average_3000_mean'] = pd.Series(x).ewm(span=3000).mean().mean(sk
ipna=True)
    X_test.loc[seg_id, 'exp_Moving_average_5000_mean'] = pd.Series(x).ewm(span=5000).mean().mean(sk
ipna=True)
    X_test.loc[seg_id, 'exp_Moving_average_10000_mean'] = pd.Series(x).ewm(span=10000).mean().mean(
skipna=True)
    X_test.loc[seg_id, 'exp_Moving_average_30000_mean'] = pd.Series(x).ewm(span=30000).mean().mean(
skipna=True)
    no_of_std = 3
    X_test.loc[seg_id, 'MA_700MA_std_mean'] = pd.Series(x).rolling(window=700).std().mean()
    X_test.loc[seg_id,'MA_700MA_BB_high_mean'] = (X_test.loc[seg_id, 'Moving_average_700_mean'] + n
o_of_std * X_test.loc[seg_id, 'MA_700MA_std_mean']).mean()
    X_test.loc[seg_id,'MA_700MA_BB_low_mean'] = (X_test.loc[seg_id, 'Moving_average_700_mean'] - no
_of_std * X_test.loc[seg_id, 'MA_700MA_std_mean']).mean()
    X_test.loc[seg_id, 'MA_400MA_std_mean'] = pd.Series(x).rolling(window=400).std().mean()
    X_test.loc[seg_id,'MA_400MA_BB_high_mean'] = (X_test.loc[seg_id, 'Moving_average_700_mean'] + n
o_of_std * X_test.loc[seg_id, 'MA_400MA_std_mean']).mean()
    X_test.loc[seg_id,'MA_400MA_BB_low_mean'] = (X_test.loc[seg_id, 'Moving_average_700_mean'] - no
_of_std * X_test.loc[seg_id, 'MA_400MA_std_mean']).mean()
    X_test.loc[seg_id, 'MA_1000MA_std_mean'] = pd.Series(x).rolling(window=1000).std().mean()

    X_test.drop('Moving_average_700_mean', axis=1, inplace=True)

    x = pd.Series(x)



    for w in [10, 50, 100,250,500,750,1000]:

        x_roll_abs_mean = x.rolling(w).mean().dropna().values
        x_roll_mean = x.rolling(w).mean().dropna().values
        x_roll_std = x.rolling(w).std().dropna().values
        x_roll_min = x.rolling(w).min().dropna().values
        x_roll_max = x.rolling(w).max().dropna().values

        X_test.loc[seg_id, 'ave_roll_std_' +str(w)] =x_roll_std.mean()
        X_test.loc[seg_id, 'std_roll_std_' +str(w)] = x_roll_std.std()
        X_test.loc[seg_id, 'max_roll_std_' +str(w)] = x_roll_std.max()
        X_test.loc[seg_id, 'min_roll_std_' +str(w)] = x_roll_std.min()
        X_test.loc[seg_id, 'q01_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.01)
        X_test.loc[seg_id, 'q05_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.05)
        X_test.loc[seg_id, 'q10_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.1)
        X_test.loc[seg_id, 'q50_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.5)
        X_test.loc[seg_id, 'q90_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.90)

        X_test.loc[seg_id, 'q95_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.95)
        X_test.loc[seg_id, 'q99_roll_std_' +str(w)] = np.quantile(x_roll_std, 0.99)
        X_test.loc[seg_id, 'kurtosis_roll_std' +str(w)] = kurtosis(x_roll_std)
        X_test.loc[seg_id, 'skew_roll_std' +str(w)] = skew(x_roll_std)




        X_test.loc[seg_id, 'ave_roll_mean_' +str(w)] =x_roll_mean.mean()
        X_test.loc[seg_id, 'mean_roll_mean_' +str(w)] = x_roll_mean.std()
        X_test.loc[seg_id, 'max_roll_mean_' +str(w)] = x_roll_mean.max()
        X_test.loc[seg_id, 'min_roll_mean_' +str(w)] = x_roll_mean.min()
        X_test.loc[seg_id, 'q01_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.01)
        X_test.loc[seg_id, 'q05_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.05)
```

```
        X_test.loc[seg_id, 'q10_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.1)
        X_test.loc[seg_id, 'q50_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.5)
        X_test.loc[seg_id, 'q90_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.9)
        X_test.loc[seg_id, 'q95_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.95)
        X_test.loc[seg_id, 'q99_roll_mean_' +str(w)] = np.quantile(x_roll_mean, 0.99)
        X_test.loc[seg_id, 'kurtosis_roll_mean' +str(w)] = kurtosis(x_roll_mean)
        X_test.loc[seg_id, 'skew_roll_mean' +str(w)] = skew(x_roll_mean)



        X_test.loc[seg_id, 'ave_roll_min_' +str(w)] =x_roll_min.mean()
        X_test.loc[seg_id, 'min_roll_min_' +str(w)] = x_roll_min.std()
        X_test.loc[seg_id, 'max_roll_min_' +str(w)] = x_roll_min.max()
        X_test.loc[seg_id, 'min_roll_min_' +str(w)] = x_roll_min.min()
        X_test.loc[seg_id, 'q01_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.01)
        X_test.loc[seg_id, 'q05_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.05)
        X_test.loc[seg_id, 'q10_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.1)
        X_test.loc[seg_id, 'q50_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.5)
        X_test.loc[seg_id, 'q90_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.9)
        X_test.loc[seg_id, 'q95_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.95)
        X_test.loc[seg_id, 'q99_roll_min_' +str(w)] = np.quantile(x_roll_min, 0.99)
        X_test.loc[seg_id, 'kurtosis_roll_min' +str(w)] = kurtosis(x_roll_min)
        X_test.loc[seg_id, 'skew_roll_min' +str(w)] = skew(x_roll_min)



        X_test.loc[seg_id, 'ave_roll_max_' +str(w)] =x_roll_max.mean()
        X_test.loc[seg_id, 'max_roll_max_' +str(w)] = x_roll_max.std()
        X_test.loc[seg_id, 'max_roll_max_' +str(w)] = x_roll_max.max()
        X_test.loc[seg_id, 'max_roll_max_' +str(w)] = x_roll_max.min()
        X_test.loc[seg_id, 'q01_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.01)
        X_test.loc[seg_id, 'q05_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.05)
        X_test.loc[seg_id, 'q10_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.1)
        X_test.loc[seg_id, 'q50_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.5)
        X_test.loc[seg_id, 'q90_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.9)
        X_test.loc[seg_id, 'q95_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.95)
        X_test.loc[seg_id, 'q99_roll_max_' +str(w)] = np.quantile(x_roll_max, 0.99)
        X_test.loc[seg_id, 'kurtosis_roll_max' +str(w)] = kurtosis(x_roll_max)
        X_test.loc[seg_id, 'skew_roll_max' +str(w)] = skew(x_roll_max)
```

```
100%|██████████| 2624/2624 [1:25:21<00:00,  1.87s/it]
```

In [0]:

```
X_test.to_pickle('gdrive/My Drive/X_test_LANL_noise')
```

In [0]:

```
X_test =  pd.read_pickle('gdrive/My Drive/X_test_LANL_noise')
```

In [0]:

```
X_test.columns
```

Out[0]:

```
Index(['ave', 'std', 'max', 'min', 'av_change_abs', 'av_change_rate',
       'abs_max', 'abs_min', 'std_first_50000', 'std_last_50000',
       ...
       'q01_roll_max_1000', 'q05_roll_max_1000', 'q10_roll_max_1000',
       'q50_roll_max_1000', 'q90_roll_max_1000', 'q95_roll_max_1000',
       'q99_roll_max_1000', 'kurtosis_roll_max1000', 'skew_roll_max1000',
       'time_to_failure'],
      dtype='object', length=418)
```

In [0]:

```
X_test = X_test.drop(['time_to_failure'],axis=1)
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_test_scaled = pd.DataFrame(scaler.fit_transform(X_test))
```

In [0]:

```
start = []
for i in range(0,628950001,150000):
  start.append(i)
X_tr['start'] = start
```

In [0]:

```
X_tr.head()
```

Out[0]:

	ave	std	max	min	av_change_abs	av_change_rate	abs_max	abs_min	std_first_50000	std
0	0.011842	5.126465	99.402431	-102.896793	-0.000078	74999.0	102.896793	0.000029	6.507118	3.7
1	0.005371	6.607595	176.366542	-158.580710	0.000002	74999.0	176.366542	0.000002	7.323559	5.5
2	0.016940	6.985333	135.859131	-110.359579	-0.000011	74999.0	135.859131	0.000004	6.122384	8.6
3	0.007472	6.940462	192.114333	-204.592423	0.000002	74999.0	204.592423	0.000008	6.257855	5.6
4	0.030049	7.317945	140.688803	-131.366905	-0.000004	74999.0	140.688803	0.000033	5.347053	7.7

5 rows × 419 columns

In [0]:

```
etq_meta = [
{"start":0,          "end":5656574},
{"start":5656574,    "end":50085878},
{"start":50085878,   "end":104677356},
{"start":104677356,  "end":138772453},
{"start":138772453,  "end":187641820},
{"start":187641820,  "end":218652630},
{"start":218652630,  "end":245829585},
{"start":245829585,  "end":307838917},
{"start":307838917,  "end":338276287},
{"start":338276287,  "end":375377848},
{"start":375377848,  "end":419368880},
{"start":419368880,  "end":461811623},
{"start":461811623,  "end":495800225},
{"start":495800225,  "end":528777115},
{"start":528777115,  "end":585568144},
{"start":585568144,  "end":621985673},
{"start":621985673,  "end":629145480},
]


for i, etq in enumerate(etq_meta):
    X_tr.loc[(X_tr['start'] + 150_000 >= etq["start"]) & (X_tr['start'] <= etq["end"] - 150_000), "
eq"] = i
```

In [0]:

```
X_tr = X_tr[X_tr["eq"].isin([2, 7, 0, 4, 11, 13, 9, 1, 14, 10])]
```

In [0]:

```
X_tr.head()
```

Out[0]:

	ave	std	max	min	av_change_abs	av_change_rate	abs_max	abs_min	std_first_50000	std
0	0.011842	5.126465	99.402431	-102.896793	-0.000078	74999.0	102.896793	0.000029	6.507118	3.7(
1	0.005371	6.607595	176.366542	-158.580710	0.000002	74999.0	176.366542	0.000002	7.323559	5.5
2	0.016940	6.985333	135.859131	-110.359579	-0.000011	74999.0	135.859131	0.000004	6.122384	8.6
3	0.007472	6.940462	192.114333	-204.592423	0.000002	74999.0	204.592423	0.000008	6.257855	5.6
4	0.030049	7.317945	140.688803	-131.366905	-0.000004	74999.0	140.688803	0.000033	5.347053	7.7(

5 rows × 420 columns

In [0]:

```
X_tr_allfeatures = X_tr.iloc[:,:-3]
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_tr_allfeatures)
X_tr_allfeatures_scaled = pd.DataFrame(scaler.transform(X_tr_allfeatures))
```

In [0]:

```
X_tr['time_to_failure'] = train_y
```

In [0]:

```
X_test.columns
```

Out[0]:

```
Index(['ave', 'std', 'max', 'min', 'av_change_abs', 'av_change_rate',
       'abs_max', 'abs_min', 'std_first_50000', 'std_last_50000',
       ...
       'max_roll_max_1000', 'q01_roll_max_1000', 'q05_roll_max_1000',
       'q10_roll_max_1000', 'q50_roll_max_1000', 'q90_roll_max_1000',
       'q95_roll_max_1000', 'q99_roll_max_1000', 'kurtosis_roll_max1000',
       'skew_roll_max1000'],
      dtype='object', length=417)
```

In [0]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import KFold
from numpy import random
import lightgbm as lgb


random.seed(1234)
```

```
time_to_failure = X_tr["time_to_failure"].values

train_X = X_tr_allfeatures_scaled.values
test_X = X_test_scaled.values

oof = np.zeros(len(train_X))


n_fold = 3

kf = KFold(n_splits=n_fold, shuffle=True, random_state=1337)
kf = list(kf.split(np.arange(len(X_tr))))
```

In [0]:

```
prediction = np.zeros(len(submission))

for fold_n, (train_index, valid_index) in enumerate(kf):
    print('Fold', fold_n)

    trn_data = lgb.Dataset(train_X[train_index], label=time_to_failure[train_index])
    val_data = lgb.Dataset(train_X[valid_index], label=time_to_failure[valid_index])

    params = {'num_leaves': 4,
      'min_data_in_leaf': 5,
      'objective':'fair',
      'max_depth': -1,
      'learning_rate': 0.02,
      "boosting": "gbdt",
      'boost_from_average': True,
      "feature_fraction": 0.9,
      "bagging_freq": 1,
      "bagging_fraction": 0.5,
      "bagging_seed": 0,
      "metric": 'mae',
      "verbosity": -1,
      'max_bin': 500,
      'reg_alpha': 0,
      'reg_lambda': 0,
      'seed': 0,
      'n_jobs': 1
      }

    clf = lgb.train(params, trn_data, 1000000, valid_sets = [trn_data, val_data], verbose_eval=1000
, early_stopping_rounds = 1000)

    oof[valid_index] += clf.predict(train_X[valid_index], num_iteration=clf.best_iteration)
    prediction += clf.predict(test_X, num_iteration=clf.best_iteration)

prediction /= n_fold

print('\nMAE: ', mean_absolute_error(time_to_failure, oof))
```

```
Fold 0
Training until validation scores don't improve for 1000 rounds.
[1000] training's l1: 1.18531 valid_1's l1: 2.08099
Early stopping, best iteration is:
[187] training's l1: 1.65495 valid_1's l1: 2.01166
Fold 1
Training until validation scores don't improve for 1000 rounds.
[1000] training's l1: 1.21648 valid_1's l1: 1.95255
Early stopping, best iteration is:
[122] training's l1: 1.80288 valid_1's l1: 1.87911
Fold 2
Training until validation scores don't improve for 1000 rounds.
[1000] training's l1: 1.20497 valid_1's l1: 2.06331
Early stopping, best iteration is:
[132] training's l1: 1.74556 valid_1's l1: 1.9783

MAE:  1.9563766947048582
```

In [0]:

```
prediction
```

Out[0]:

```
array([5.02782914, 5.77850313, 6.87249631, ..., 2.61524653, 2.16853869,
       9.10553079])
```

In [0]:

```
submission.time_to_failure = prediction
submission.to_csv('submission_cv.csv',index=True)
```

In [0]:

```
!kaggle competitions submit -c LANL-Earthquake-Prediction -f submission_cv.csv -m "Noise-
Submission"
```

```
100% 74.8k/74.8k [00:00<00:00, 283kB/s]
Successfully submitted to LANL Earthquake Prediction
```