

Diploma Thesis

Predict the time remaining before laboratory earthquakes (at Los Alamos National Laboratory) occur from real-time seismic data

Ruchi

March 4, 2022

Supervised by

mentors.diploma@appliedroots.com

Abstract

An important goal in seismology is the ability to accurately predict future earthquakes before they occur. It is very important for preparation of emergency personnel and disaster relief. In seismology, earthquake prediction is well defined: the identification of severity, bounded geographic region, and time window in which a quake will occur with high probability. We tried approaching earthquake prediction problem using machine learning. In particular, we applied efficient techniques from predictive machine learning and statistics to a restricted version of this problem - prediction of the time-to-failure or time-to-fault.

Contents

1 Introduction

2 Related work

3 Data collection

3.1.1 Data source and acquisitions

3.1.2 Libraries to be used

3.1.3 Performance metric

4 Data visualization

5 Feature engineering and feature selection

6 Modelling using machine learning techniques

7 Deployment

8 References

1. Introduction

Earthquake prediction is a well-studied problem. However, there is a gap between the application traditional statistics-based modelling and modern machine learning-based methods..

Earthquake prediction is considered as one of the grand challenges in earth sciences since earthquakes are catastrophic events that result in loss of life, infrastructure and have a severe impact on the economy of a country. There might be several things that might seem to be useful for earthquake prediction like planetary alignment, lunar phases, animal behaviour but all of these turn out to be of not much use in predicting future earthquakes. Researchers have applied multiple computational methods for earthquake prediction but accurate prediction of earthquake is still considered as a challenging task. Currently there is no any model exists that can predict the exact position, magnitude, frequency and time of an earthquake. Researchers have conducted several experiments on earthquake events and forecasts, leading to a variety of findings based on the factors considered. The well-known Gutenberg and Richter statistical model found a correlation between the magnitude of earthquake and frequency of earthquake Magnitude of earthquake:

The equation for the magnitude of an earthquake is $M = \log\left(\frac{I}{I_0}\right)$, where:

- M is the magnitude
- I is the intensity of the earthquake
- I_0 is the intensity of an earthquake with a magnitude of 0

This formula can be changed from logarithmic form to exponential form:

$$M = \log_{10}\left(\frac{I}{I_0}\right)$$

$$\frac{I}{I_0} = 10^M$$

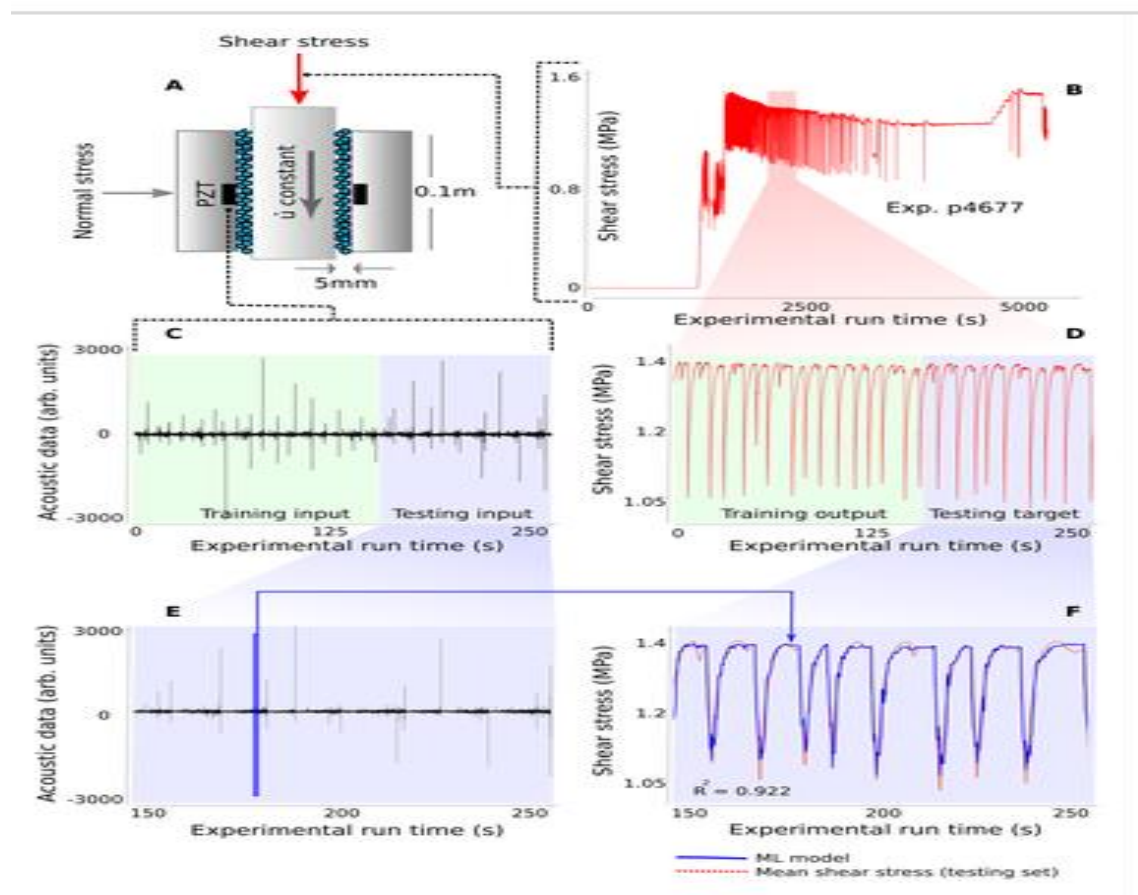
$$I = I_0 \times 10^M$$

Our intent is to design algorithms that are effective for forecasting quakes, but also to make sure that they are efficient (fast, low footprint) enough to potentially run on embedded monitoring devices in the field.

We use synthetic & real dataset released by Los Alamos National Laboratory for our project. The data is hosted at <https://www.kaggle.com/c/LANL-Earthquake-Prediction/data> and consists of simulated acoustic waveform signal.

2. Related work

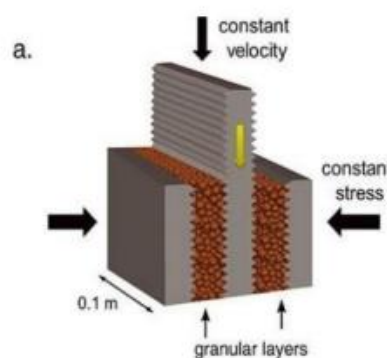
The data for this challenge comes from a classic laboratory earthquake experiments. Lot of work has already been done using this data. The idea behind characteristic, repeating earthquakes was the basis of the well-known Parkfield prediction based strictly on seismic data. Similar earthquakes occurring between 1857 and 1966 suggested a recurrence interval of 21.9 ± 3.1 years, and thus, an earthquake was expected between 1988 and 1993 but ultimately took place in 2004. With this approach, as earthquake recurrence is not constant for a given fault, event occurrence can only be inferred within large error bounds. Over the last 15 years, there has been renewed hope that progress can be made regarding forecasting owing to tremendous advances in instrumentation quality and density. These advances have led to exciting discoveries of previously unidentified slip processes that include slow slip, low frequency earthquakes and Earth tremor that occur deep in faults. These discoveries inform a new understanding of fault slip and may well lead to advances in forecasting, impending fault failure if the coupling of deep faults to the seismogenic zone can be unraveled. The advances in instrumentation sensitivity and density also provide new means to record small events that may be precursors. Acoustic/seismic precursors to failure appear to be a nearly universal phenomenon in materials.



3. Data Collection

Given seismic signals we are trying to predict the time until the onset of laboratory earthquakes. The data comes from a well-known experimental setup used to study earthquake physics.

- In lab 2 plates have been kept under pressure resulting in shear stress.
- The training data is a single, continuous segment of experimental data.
- The test data consists of a folder containing many small segments that may correspond to different experiments.
- For each seg_id in the test folder, you should predict a single time_to_failure corresponding to the time between the last row of the segment and the next laboratory earthquake. Since we have just two columns i.e Acoustic Data and time of failure given in our datasets it's very important to identify or generate new features that might help in predicting the onset of next earthquakes. This is referred as feature Engineering. It is assumed that it will help model in better prediction.



The laboratory faults fail in repetitive cycles of stick and slip that is meant to mimic the cycle of loading and failure on tectonic faults. While the experiment is considerably simplified than a fault on Earth, it shares certain physical characteristics, whose similarity, just cannot be ignored.

Data provided by the laboratory includes the following columns:

- acoustic_data - the seismic signal [int16]
- time_to_failure - the time (in seconds) until the next laboratory earthquake [float64]
- seg_id - the test segment ids for which predictions should be made (one prediction per segment)

3.1.1 Data source and acquisition: There are different ways through which data can be imported directly to google colab that can later be used for visualization. First method, since our data is readily available on kaggle, hence we can directly import data to google colab using kaggle API.

```
pip install kaggle from google.colab
import files uploaded = files.upload()
!mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600
~/.kaggle/kaggle.json !kaggle datasets download -d LANL-Earthquake-
```

Prediction Secondly, chrome extension (curlwget) can be used for directly loading the data in Notebook.

3.1.2 Libraries to use: Numpy, Pandas, sklearn, matplotlib, seaborn

```
import matplotlib.pyplot as plt          #Data Visualization
from tqdm import tqdm_notebook
import seaborn as sns
import pickle
import pywt          #PyWavelets is a free Open Source library for wavelet transforms in Python
import warnings
warnings.filterwarnings("ignore")        # Donot display warning messages.
from sklearn.preprocessing import StandardScaler    #Data Scaling
from sklearn.model_selection import GridSearchCV    #Hyperparameter optimization
from sklearn.svm import NuSVR, SVR                #Support vector Machine Model
from sklearn.kernel_ridge import KernelRidge      #kernel ridge model
from catboost import CatBoostRegressor, Pool      #Catboost Model
import xgboost as xgb                             #XgBoost Model
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from prettytable import PrettyTable
from sklearn.model_selection import KFold
```

3.1.3 **Performance metric:** There are many performance evaluator available that can be used for binary classification problems. The accuracy for prediction of earthquake is measured using the following measures:

- **Mean Absolute Error:** Mean Absolute error, as the name suggests, is the mean of all the errors obtained in each of the regressor model's predictions and the actual observation. It can mathematically be expressed as:
- **Cross validation Score:** Another performance metric that can be used is cross validation score. Cross-validation mainly used for applied machine learning to estimate a machine learning model's capability on unseen data. We will be using Mean absolute error as our performance metric to solve the given problem. MAE helps users to formulate learning problems into optimization problems. It also serves as an easy-to-understand quantifiable measurement of errors for regression problems. Mean absolute error can be used as performance metric in housing price prediction problems.
- **Advantages of MAE:** MAE unit as the output variable. Also it is most Robust to outliers.
- **Disadvantages of MAE:** The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

4. Data visualization using exploratory data analysis and feature engineering

The first and the most important part of any data science project is collecting and visualizing the data so as to get some initial intuition from it. In layman terms this can be thought of searching for a trekking destination during weekends with friends or family. We will look for nearest location, reviews, aesthetics, trek length and other important features related to different trekking options as per ones requirement. It helps in planning things ahead for a weekend getaway.

So the point is exploring the data that is already available with us, will help us in predicting what to expect in future.

Exploratory data analysis (EDA) is an approach of analyzing the datasets to summarize the main characteristics, often with the visual methods. EDA checklist typically includes following questions

1. What problem(s) are you trying to solve (or prove wrong)?
2. What kind of data do you have and how do you treat different types?
3. What's missing from the data and how do you deal with it?
4. Where are the outliers and why should you care about them?
5. How can you add, change or remove features to get more out of your data?

Importing of Data:

Our goal is to predict the timing of laboratory earthquakes using the seismic signals. Data we will be using comes from a well-known experimental setup in Los Alamos used to study earthquake physics. Dataset has 150000 datapoints. The given dataset has two features:

- 1) `acoustic_data` - the seismic signal [int16]
- 2) `time_to_failure` - the time (in seconds) until the next laboratory earthquake [float64].
- 3) `seg_id` - the test segment ids for which predictions should be made (one prediction per segment)

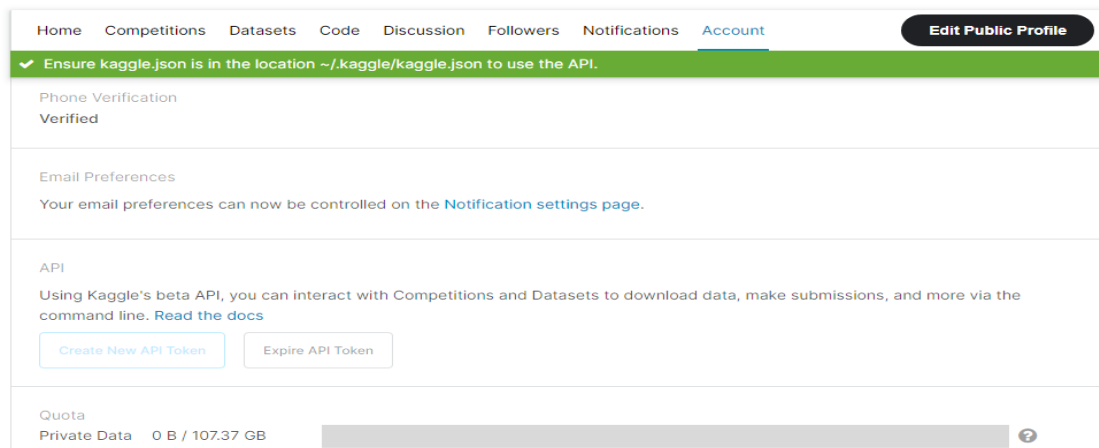
Given that data is recorded in bins of 4096 samples and within those bins seismic data is recorded at 4MHz. The seismic data is recorded using a piezoceramic sensor, which outputs a voltage upon deformation by incoming seismic waves.

Time to failure is based on a measure of fault strength (shear stress, not part of the data for the competition). When a labquake occurs this stress drops unambiguously.

The training data is a single, continuous segment of experimental data. The test data consists of a folder containing many small segments. The data within each test file is continuous, but the test files do not represent a continuous segment of the experiment; thus, the predictions cannot be assumed to follow the same regular pattern seen in the training file.

Steps to import data from kaggle directly to google colab are as follows:

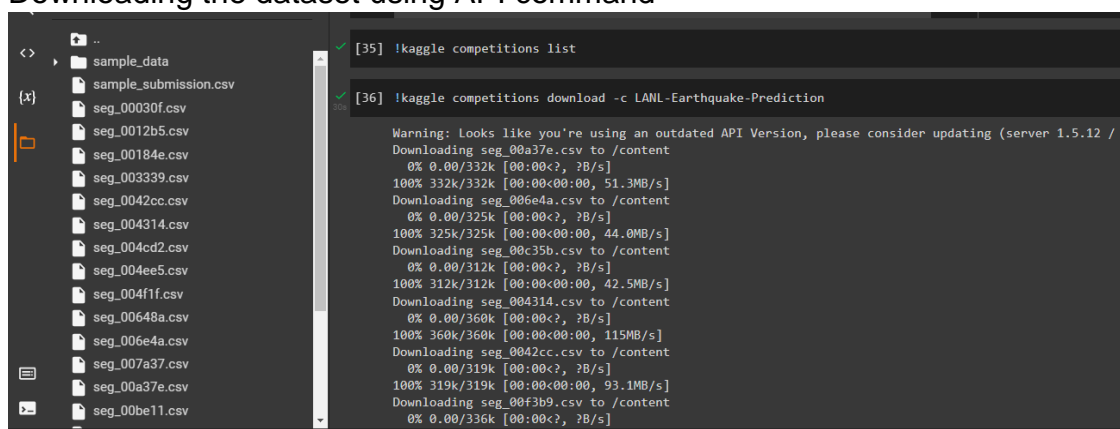
1) Creating API token in Kaggle.



2) Use of kaggle API using JSON file to download the dataset



3) Downloading the dataset using API command

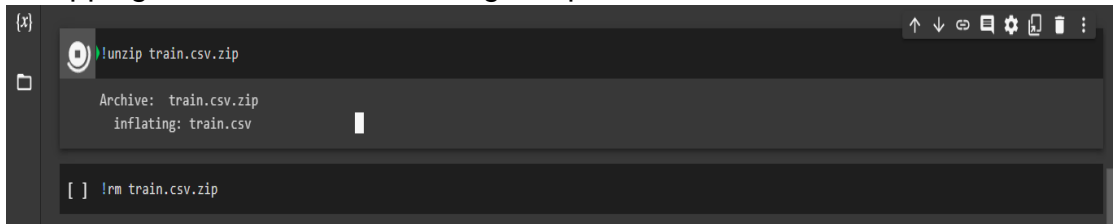


!mkdir: used to make a directory.

!chmod 600: It is used to change permission for owner. Chmod 600 means user has full read and write access. chmod 700 means user has full read write and execute access.

The exclamation mark is used for executing commands from the underlying operating system.

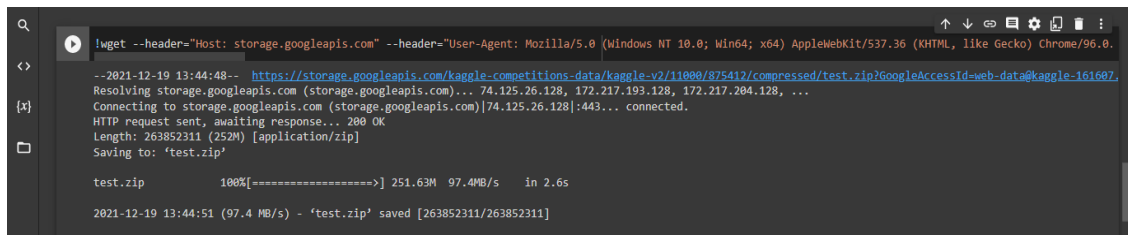
4) Unzipping of dataset and removing of zip file.



```
[x] !unzip train.csv.zip
Archive: train.csv.zip
  inflating: train.csv

[ ] !rm train.csv.zip
```

Other way of downloading the data is by using Curlwget plugin that builds a command line for 'curl/wget' tools to enable the download of data on a console only session.

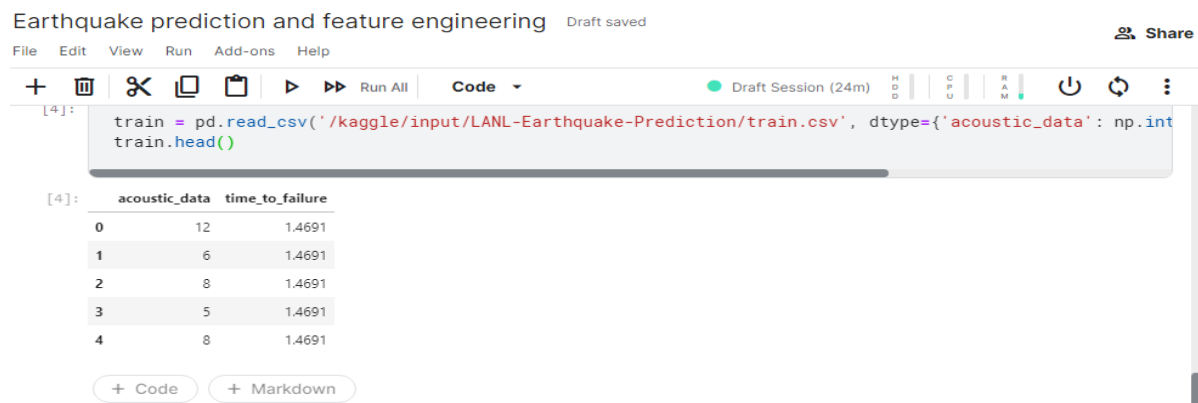


```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.
--2021-12-19 13:44:48-- https://storage.googleapis.com/kaggle-competitions-data/kaggle-v2/11000/875412/compressed/test.zip?GoogleAccessId=web-data@kaggle-161607.
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.26.128, 172.217.193.128, 172.217.204.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)[74.125.26.128]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 263852311 (252M) [application/zip]
Saving to: 'test.zip'

test.zip      100%[=====] 251.63M  97.4MB/s   in 2.6s
2021-12-19 13:44:51 (97.4 MB/s) - 'test.zip' saved [263852311/263852311]
```

Once we have our data locked and loaded to colab notebook we will move to our next step i.e. visualization:

The above steps can be used for importing data directly to the google colab but the dataset we are using is too large, hence we will be using kaggle's code editor for code execution.



Earthquake prediction and feature engineering Draft saved

File Edit View Run Add-ons Help

+ Draft Session (24m)

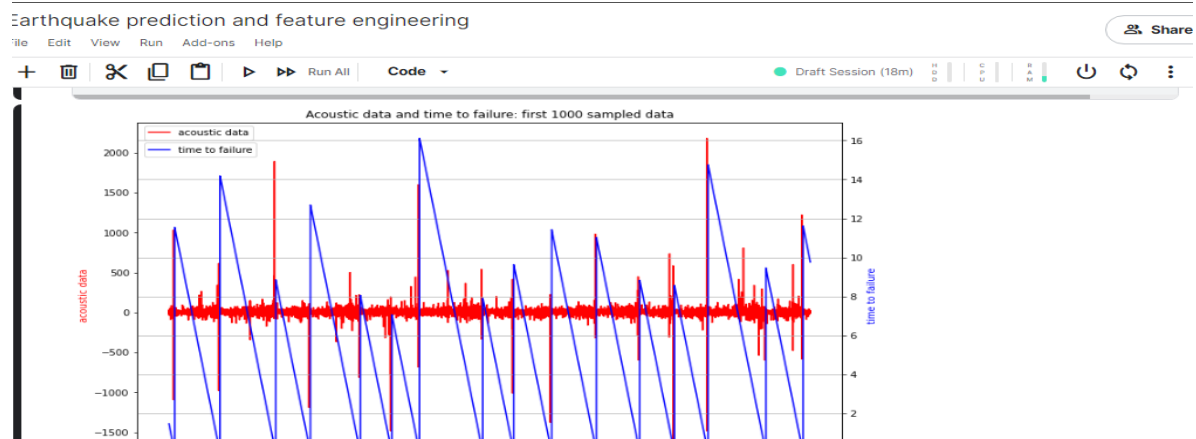
```
[4]: train = pd.read_csv('/kaggle/input/LANL-Earthquake-Prediction/train.csv', dtype={'acoustic_data': np.int
train.head()
```

	acoustic_data	time_to_failure
0	12	1.4691
1	6	1.4691
2	8	1.4691
3	5	1.4691
4	8	1.4691

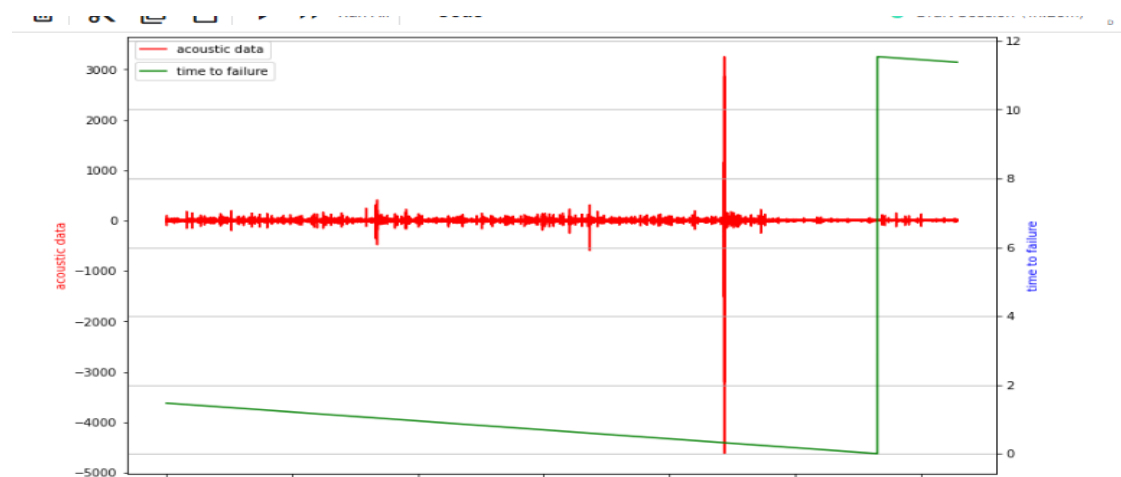
+ Code + Markdown

Data visualization:

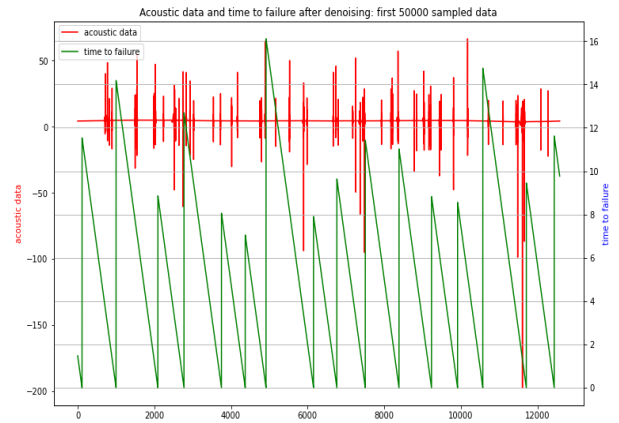
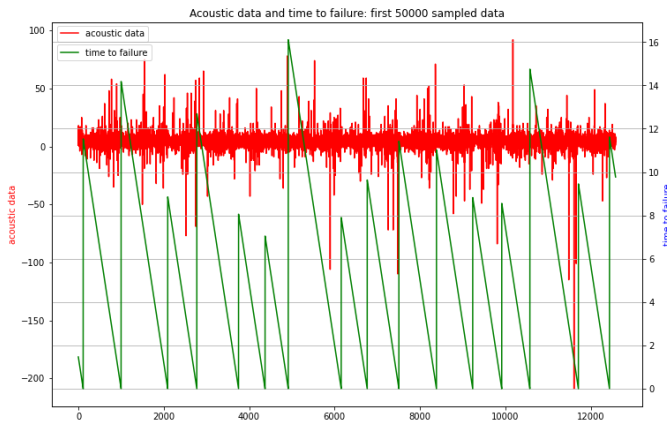
Just to get an intuition from we have plotted first 1000 datapoints and interestingly we have observed that just before the time of failure, there is a spike in acoustic data. On visualizing the complete dataset we have observed that it's a recurrent pattern. Every time just before the time of failure a spike in acoustic has been observed



The zoomed in version of the data can be seen in the figure below. Although a significant peak can be seen the figure just before the time of failure still the same cannot be concluded for the whole dataset.



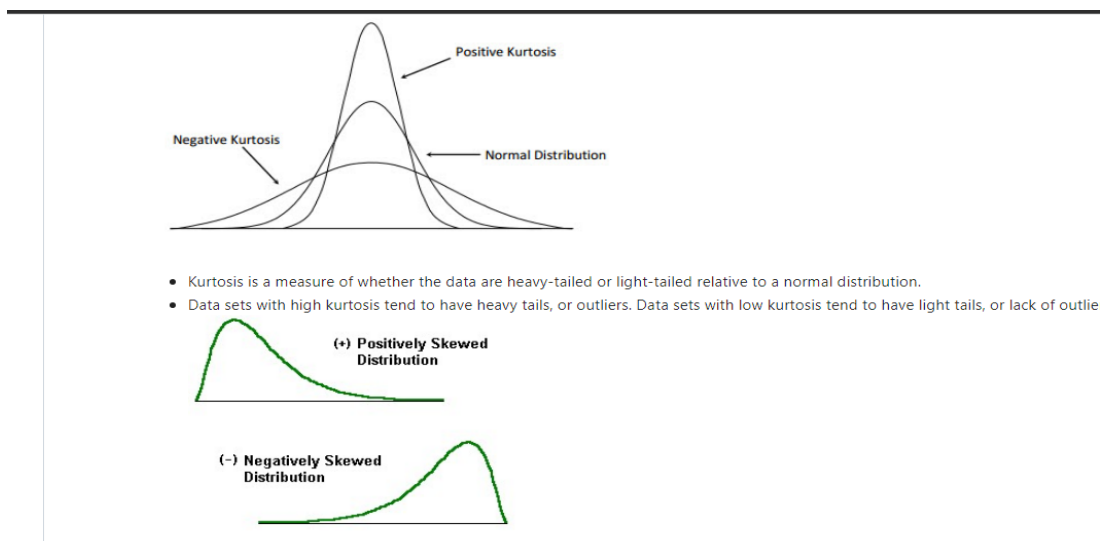
Denoising of data: In seismology, we (the people trying to measure the seismic signals) artificially generate signals called impulse signals. These impulse signals interact with the Earth's actual seismic signal (which is what we need) to produce the final signal which our seismograph picks up (this same process takes place in the laboratory simulation of an earthquake). So, the real challenge is to uncover the actual signal from mixed seismogram (which is a combination of the Earth's impulse signal and the artificial impulse signal). This is why we are trying "wavelet decomposition". The process is a sort of "deconvolution", which means it undoes the convolution process and uncovers the real Earth impulse from the mixed seismogram and the artificial impulse.



Researching through various kernels, I have decided to use wavelet decomposition for removing artificial signal and additional noises from our data.

5. Feature engineering: We are trying to get the following features for better understanding of our data.

- Aggregations: min, max, mean and std
 1. Mean gives us the average values of the dataset.
 2. Standard deviation helps in understanding the spread of data from the mean.
 3. Kurtosis values helps us in understanding the amount of outlier dataset is having. Higher the kurtosis values meaning more outliers.



- Absolute features: max, mean and std
- Quantile features: Quantiles are cut points dividing the range of a probability distribution into continuous intervals with equal probabilities.
- Rolling features
- Term Average, which is calculated as the average of consecutive elements.
- fft(fast fourier transform) A signal is information that changes over time. For example, audio, video, and voltage traces are all examples of signals. A frequency is the speed at which something repeats. For example, clocks ticks

at a frequency of one hertz (Hz), or one repetition per second. Power, in this case, just means the strength of each frequency.

Since our data has only two features so for its better interpretation we need to add more features such as mean, standard deviation, percentile values, skewness and many more, so that we can get more inferences from the data. This step can be referred as feature engineering.

So to create new features we are dividing the number of rows in train dataset to number of rows in test dataset and this is referred as segment.

For each segment we are computing min, max, average, 1,5,95 and 99 percentile values.

We can see that the mean is shifted towards higher values due to the earthquake. In addition we can see that the 25% up to 75% quartiles are looking very discrete.

Looking at the time of failure, av_change_rate, avg_max and avg_min we can't say much about it.

All the data generated using various statistical methods, we will save them in a data frame named train_df.

Earthquake prediction and feature engineering Draft saved

File Edit View Run Add-ons Help

+ [Icons] Cancel Run Code Draft Session (1h:8m)

```
[16]: train_df
```

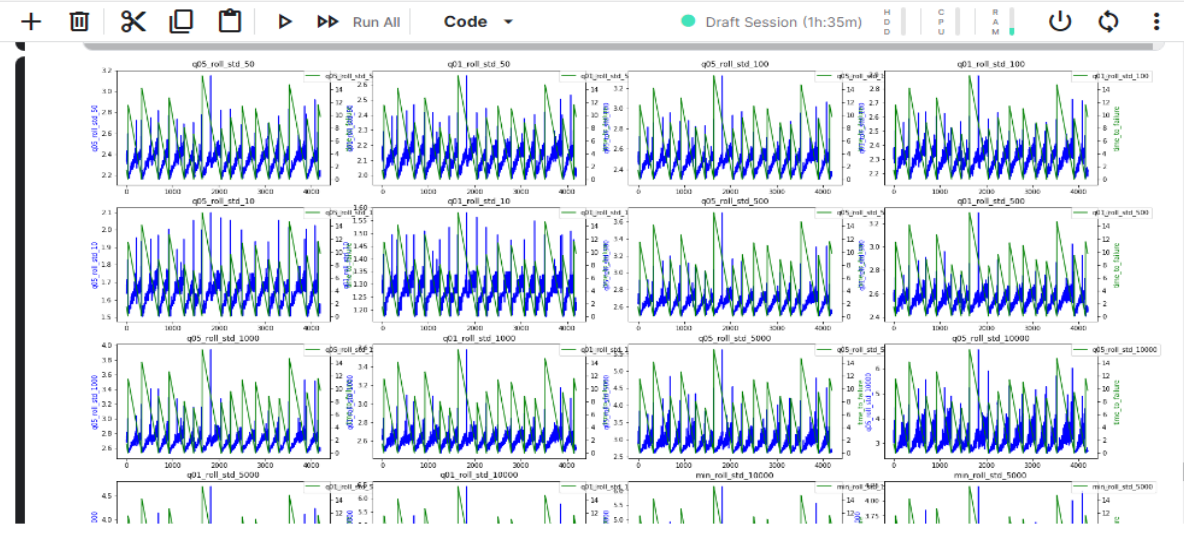
	mean	std	max	min	sum	mad	kurt	skew	med	q95	...	av_change_rate_roll_std_10000	abs_max_roll_std_10
0	4.884113	5.101106	104.0	-98.0	732617.0	3.263401	33.662481	-0.024061	5.0	11.0	...	69757.715794	11.207
1	4.725767	6.588824	181.0	-154.0	708865.0	3.574302	98.758517	0.390561	5.0	12.0	...	69813.904115	13.235
2	4.906393	6.967397	140.0	-106.0	735959.0	3.948411	33.555211	0.217391	5.0	13.0	...	70060.830746	14.344
3	4.902240	6.922305	197.0	-199.0	735336.0	3.647117	116.548172	0.757278	5.0	12.0	...	69923.983177	17.521
4	4.908720	7.301110	145.0	-126.0	736308.0	3.826052	52.977905	0.064531	5.0	12.0	...	70018.131493	16.551
...
4189	4.446347	3.721679	76.0	-61.0	666952.0	2.571151	23.956541	-0.017274	4.0	10.0	...	70042.073698	6.629
4190	4.413793	3.911331	82.0	-60.0	662069.0	2.605594	28.942916	0.161954	4.0	9.0	...	70215.391334	6.290
4191	4.607200	3.221462	63.0	-39.0	691080.0	2.394842	9.986985	0.142006	5.0	9.0	...	69997.519386	4.393
4192	4.465280	4.160361	94.0	-97.0	669792.0	2.629871	60.777372	0.029714	4.0	10.0	...	70043.438133	8.560
4193	4.518400	3.375554	64.0	-66.0	677760.0	2.455603	15.873626	-0.123669	5.0	9.0	...	70140.461922	4.892

4194 rows x 228 columns

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

```
[19]: time_to_failure
```

0	1.430797
1	1.391499
2	1.353196
3	1.313798
4	1.274400
...	...
4189	9.926798
4190	9.887500
4191	9.849197
4192	9.809799
4193	9.771496



As we can see that by plotting various statistically generated features we are unable to derive any conclusion. Hence we will try to go for feature selection.

Since our test data is present in lots of segments hence we are going to initialize them in test_df dataframe.



```
submission = pd.read_csv('/kaggle/input/LANL-Earthquake-Prediction/sample_submission.csv', index_col='seg_id')
test_df = pd.DataFrame(columns=train_df.columns, dtype=np.float64, index=submission.index)
```

+ Code

+ Markdown

```
test_path = '/kaggle/input/LANL-Earthquake-Prediction/test'
for seg_id in tqdm_notebook(test_df.index):
    seg_file = seg_id + '.csv'
    seg = pd.read_csv(os.path.join(test_path, seg_file))
    create_features(seg_id, seg, test_df, targets=False)
```

Feature selection:

Two approaches that can be used for feature selection are:

Approach1: Correlation: Features were ranked in order of absolute correlation (ignoring features where correlation had no statistical significance)

```
corrMat = train_df.corr()
corrTarget = abs(corrMat['time_to_failure'])
corrTarget = corrTarget[corrTarget > 0.5]
high_corr_features = corrTarget.index.drop(['time_to_failure'])
high_corr_features
```

+ Code

+ Markdown

```
[63]: Index(['q01_roll_std_10', 'q05_roll_std_10', 'q01_roll_std_50',
            'q05_roll_std_50', 'q01_roll_std_100', 'q05_roll_std_100',
            'min_roll_std_500', 'q01_roll_std_500', 'q05_roll_std_500',
            'min_roll_std_1000', 'q01_roll_std_1000', 'q05_roll_std_1000',
            'min_roll_std_5000', 'q01_roll_std_5000', 'q05_roll_std_5000',
            'min_roll_std_10000', 'q01_roll_std_10000', 'q05_roll_std_10000'],
           dtype='object')
```

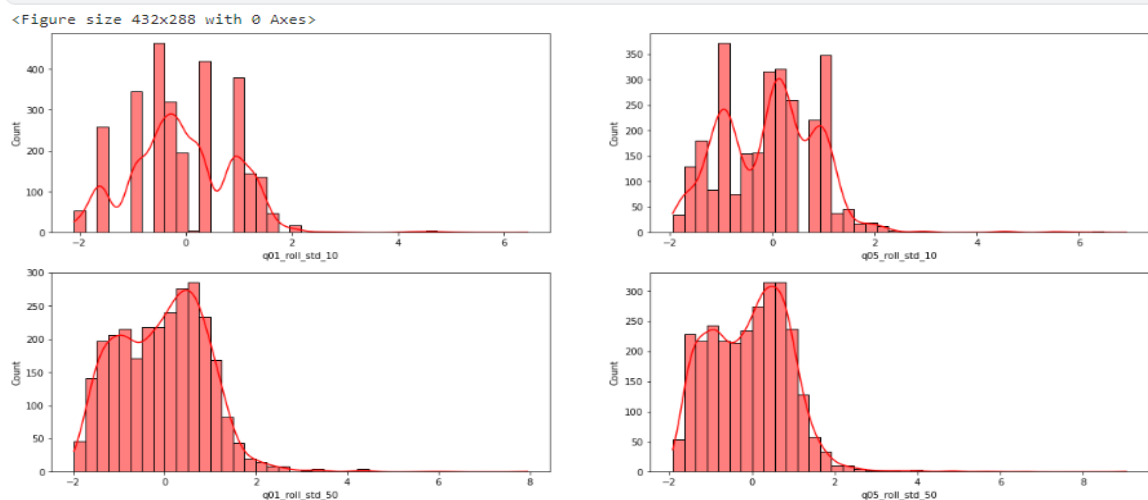
+ Code

+ Markdown

Approach2: Model based feature selection: Features were selected using the feature importance given by the Model.

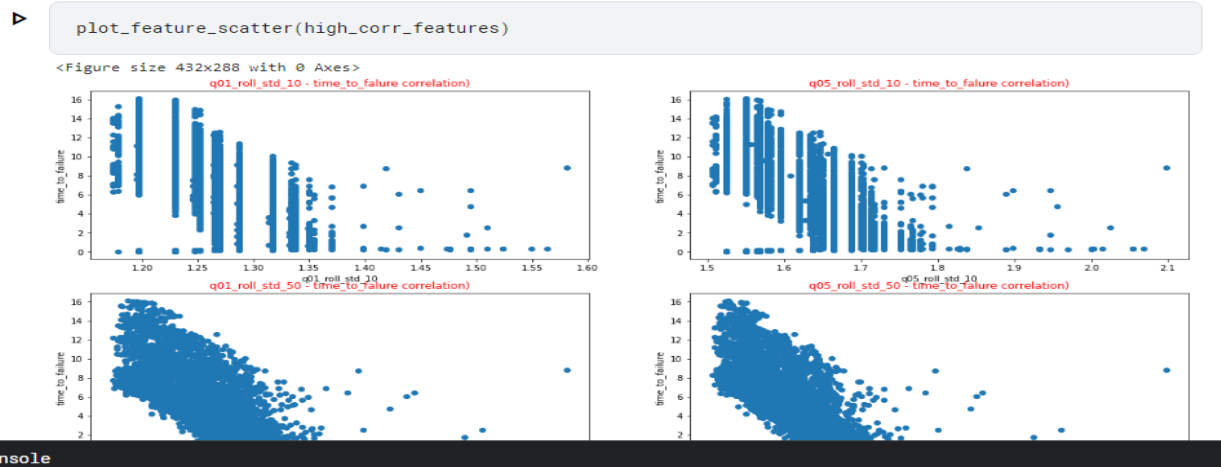
Training and model selection:

1) Histogram:



2) Scatter Plot:

We are using scatter plot to plot features that are highly correlated with time of failure. From the plot excursion can be seen for all features during time of failure.



We have calculated highly correlated features for Test dataset.

```
final_X_test = test_df[high_corr_features]
final_X_test
```

[25]:

	q01_roll_std_10	q05_roll_std_10	q01_roll_std_50	q05_roll_std_50	q01_roll_std_100	q05_roll_std_100	min_roll_std_500	q01_roll_std_500
seg_id								
seg_00030f	1.286684	1.649916	2.157143	2.381905	2.343678	2.514985	2.457079	2.561941
seg_0012b5	1.264911	1.636392	2.123964	2.335529	2.311041	2.475659	2.453978	2.575590
seg_00184e	1.269296	1.636392	2.126701	2.339021	2.312548	2.475639	2.300436	2.550708
seg_003339	1.229273	1.567021	2.042807	2.232963	2.245782	2.380476	2.344033	2.486923
seg_0042cc	1.264911	1.619328	2.104999	2.304299	2.277935	2.440070	2.396265	2.508183
...
seg_ff4236	1.286684	1.663330	2.154824	2.366863	2.332272	2.493851	2.463440	2.555124
seg_ff7478	1.264911	1.632993	2.103544	2.310932	2.301932	2.453939	2.393621	2.518780
seg_ff79d9	1.331656	1.699673	2.182397	2.425124	2.391568	2.562275	2.494014	2.653569
seg_ffbd6a	1.370320	1.766981	2.242539	2.494402	2.447510	2.623495	2.529736	2.679523
seg_ffe7cc	1.229273	1.567021	2.034498	2.215437	2.222838	2.354107	2.315683	2.456139

2624 rows × 18 columns

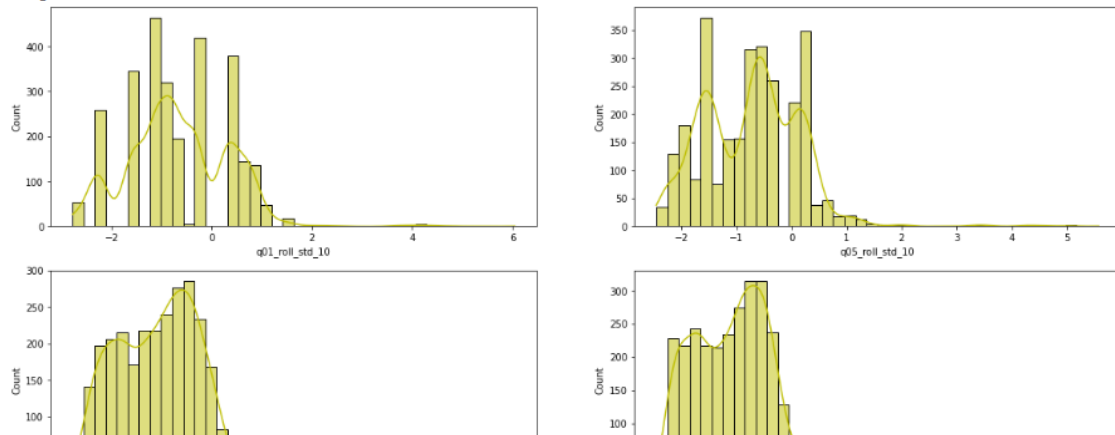
Scaling of data:

Now we will scale the features to better fit the normal distribution and will try to visualize our data again. We will scale with both train and test data.

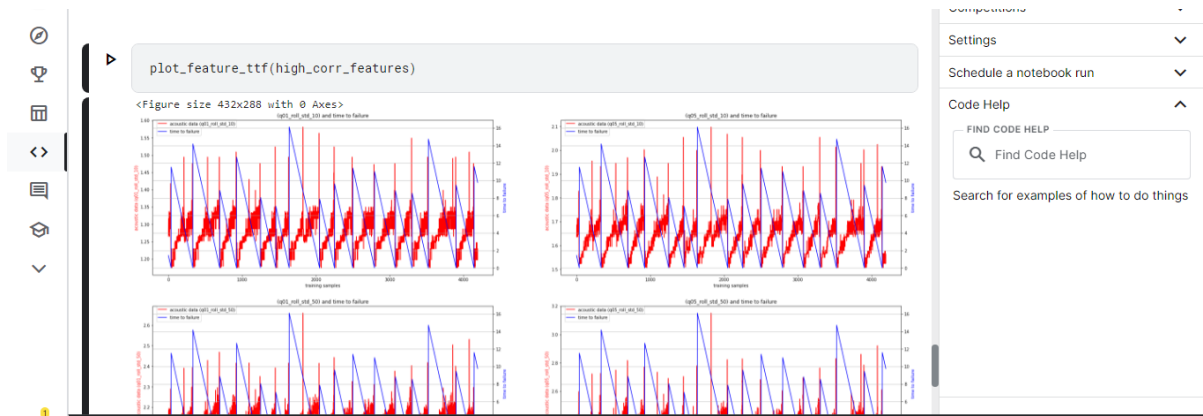
After scaling the features we have plotted the features against time of failure and found that there is a spike in the values of these features during the time of failure.

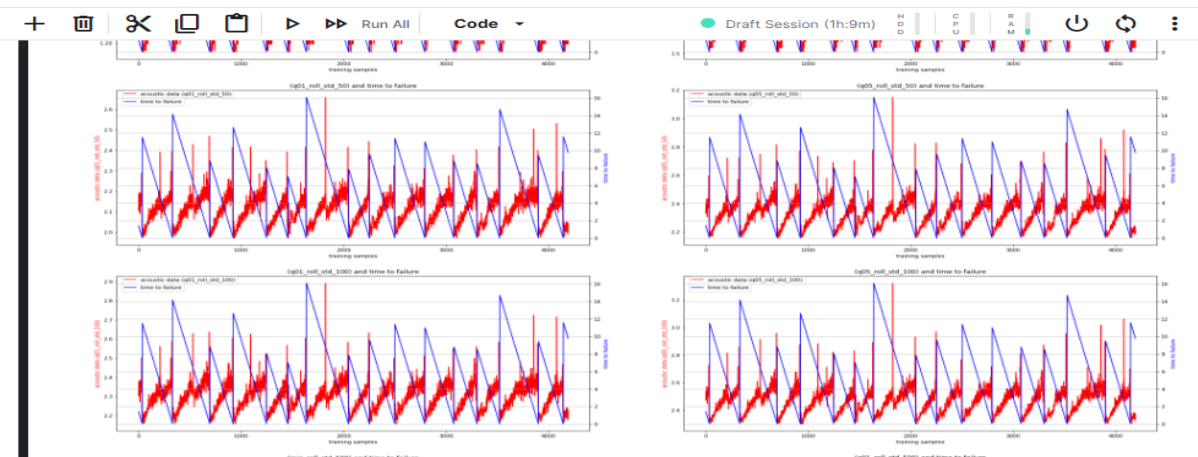
```
plot_histplot_features(high_corr_features, colors='yellow', df1=scaled_X_train)
```

<Figure size 432x288 with 0 Axes>



Once we get all the correlated features we will plot them against time of failure.





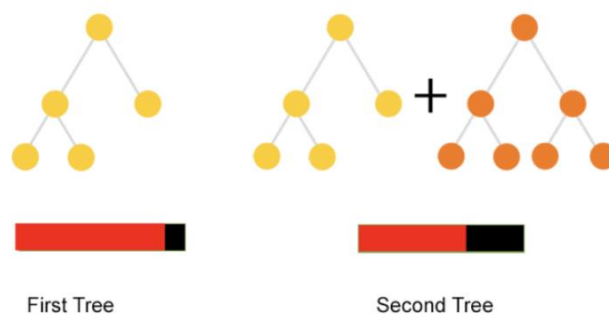
6. Modelling using machine learning techniques:

From the above plots we can see spike in the features with the during the time of failure. Now using the above features we will try to implement different models.

We tried implementing:

1) Catboost Model:

- Gradient boosting on decision trees is a form of machine learning that works by progressively training more complex models to maximize the accuracy of predictions.
- It's particularly useful for predictive models that analyze ordered (continuous) data and categorical data.
- It's one of the most efficient ways to build ensemble models. The combination of gradient boosting with decision trees provides state-of-the-art results in many applications with structured data.
- On the first iteration, the algorithm learns the first tree to reduce the training error
- This model usually has a significant error; it's not a good idea to build very big trees in boosting since they over fit the data.



```
[50]: train_pool = Pool(X_train, y_train)
      m = CatBoostRegressor(iterations=10000, loss_function='MAE', boosting_type='Ordered')
      m.fit(X_train, y_train, silent=True)
      m.best_score_

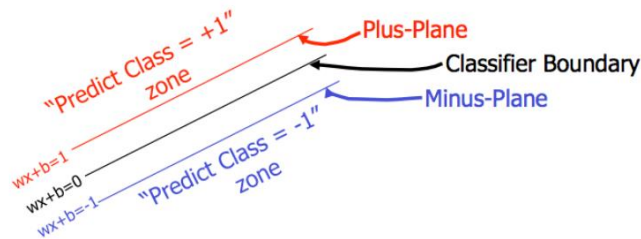
[50]: {'learn': {'MAE': 1.0722845986656637}}
```

+ Code

+ Markdown

2) Support Vector Machine + Radial Basis Function Kernel

Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification and regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.



- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

Classify as.. **+1** if $\mathbf{w} \cdot \mathbf{x} + b \geq 1$
-1 if $\mathbf{w} \cdot \mathbf{x} + b \leq -1$

• Common kernel functions for SVM

- linear $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$
- polynomial $k(\mathbf{x}_1, \mathbf{x}_2) = (\gamma \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$
- Gaussian or radial basis $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$
- sigmoid $k(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma \mathbf{x}_1 \cdot \mathbf{x}_2 + c)$

```
[36]: X_train, X_test, y_train, y_test = train_test_split(scaled_X_train, y_tr, test_size=0.33, random_state=4)
```

```
[37]: parameters = {'kernel': ('linear', 'rbf', 'poly'), 'C':[0.01, 0.1, 1, 10, 100], 'epsilon':[0.1,0.2,0.3,0.4]}
svr = SVR(gamma = 'scale')
clf = GridSearchCV(svr, parameters)
clf.fit(X_train,y_train)
clf.best_params_

[37]: {'C': 1, 'epsilon': 0.6, 'kernel': 'rbf'}
```

3) XgBoost:

XgBoost stands for xtreme gradient boosting. It is an exceptionally useful machine learning model where we don't want to sacrifice the ability to correctly classify the observation but still want a model i.e. fairly easy to understand and interpret.

[illegible]

SVR gives relatively good result on training set. However when used to predict the test data the MAE increased significantly. It may be due to over fitting of model. After training different it has been observed that Catboost Regressor model gives the best performance.

```

mytable=PrettyTable(["Model", "Best hyperparameter", "MAE"])
mytable.add_row(["SVR+RBF", "C=1, epsilon=0.6", "2.091950549780848"])
mytable.add_row(["CatBoost Regressor", "iteration=10000", "1.0722845986656637"])
mytable.add_row(["XBoost", "Kfold=5", "2.2707123057435328"])
print(mytable)

```

Model	Best hyperparameter	MAE
SVR+RBF	C=1, epsilon=0.6	2.091950549780848
CatBoost Regressor	iteration=10000	1.0722845986656637
XBoost	Kfold=5	2.2707123057435328

+ Code

+ Markdown

Hence we will be using the CatBoost Regressor model for our predictions. We have saved the model in pickle file.

```

[78]: train_pool = Pool(X_train, y_train)
      prediction = np.zeros(len(X_valid))
      m = CatBoostRegressor(iterations=10000, loss_function='MAE', boosting_type='Ordered')
      m.fit(X_train, y_train, silent=True)
      m.best_score_

```

Custom logger is already specified. Specify more than one logger at same time is not thread safe.

```

[78]: {'learn': {'MAE': 1.0722845986656637}}

```

```

[85]: joblib.dump(m, 'model.pkl')

```

```

[85]: ['model.pkl']

```

+ Code

+ Markdown

```

prediction=m.predict(final_X_test)
submission = pd.read_csv('../input/LANL-Earthquake-Prediction/sample_submission.csv')
submission['time_to_failure'] = prediction
submission.to_csv('submission_cat.csv', index=False)
submission.head()

```

```

[90]:   seg_id  time_to_failure
0  seg_00030f      1.861941
1  seg_0012b5      2.462902
2  seg_00184e      2.469592
3  seg_003339      2.459961
4  seg_0042cc      2.576124

```

+ Code

+ Markdown

7. Deployment using Heroku:

Heroku is a **cloud platform as a service** (PaaS) supporting several programming languages. One of the first cloud platforms, Heroku has been in development since June 2007, when it supported only the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go.

Heroku's simple setup makes it an ideal tool for limited budgets or businesses beginning to test opportunities in the cloud.

First we have to create a virtual environment on our local box.

```
pip install virtualenv
```

We will create a directory using make directory command (mkdir) and activate the virtual environment.

Installing all the required libraries and [packages required for the execution.

Heroku deployment: Heroku is a platform as service type model. Steps involved in Deployment via Heroku are as follows:

- 1) Requirement.txt
- 2) Procfile
- 3) Autodetect the app type
- 4) Git push and running

```
Anaconda Prompt
(base) C:\Users\hp>cd documents
(base) C:\Users\hp\Documents>cd Herokudeployment
(base) C:\Users\hp\Documents\Herokudeployment>.\venv\scripts\activate
(venv) (base) C:\Users\hp\Documents\Herokudeployment>_
```

Virtual environment will give us an empty python platform. We need to install all the necessary libraries required for the execution of our model.

```
Anaconda Prompt
(base) C:\Users\hp\Documents\Herokudeployment>.\venv\scripts\activate
(venv) (base) C:\Users\hp\Documents\Herokudeployment>pip install flask
Requirement already satisfied: flask in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (2.0.3)
Requirement already satisfied: Werkzeug>=2.0 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from flask) (2.0.3)
Requirement already satisfied: click>=7.1.2 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from flask) (8.0.4)
Requirement already satisfied: Jinja2>=3.0 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from flask) (3.0.3)
Requirement already satisfied: itsdangerous>=2.0 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from flask) (2.0.1)
Requirement already satisfied: colorama in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from click>=7.1.2->flask) (0.4.4)
Requirement already satisfied: importlib-metadata in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from click>=7.1.2->flask) (4.8.3)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from Jinja2>=3.0->flask) (2.0.1)
Requirement already satisfied: dataclasses in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from Werkzeug>=2.0->flask) (0.0)
Requirement already satisfied: typing-extensions>=3.6.4 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from importlib-metadata->click>=7.1.2->flask) (4.1.1)
Requirement already satisfied: zipp>=0.5 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from importlib-metadata->click>=7.1.2->flask) (3.6.0)
(venv) (base) C:\Users\hp\Documents\Herokudeployment>pip install gunicorn numpy pandas sklearn
Requirement already satisfied: gunicorn in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (20.1.0)
Requirement already satisfied: numpy in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (1.19.5)
Requirement already satisfied: pandas in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (1.1.5)
Requirement already satisfied: sklearn in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (0.0)
Requirement already satisfied: setuptools>=3.0 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from gunicorn) (39.0.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.2 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from pandas) (2021.3)
Requirement already satisfied: scikit-learn in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from sklearn) (0.24.2)
Requirement already satisfied: six>=1.5 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: joblib>=0.11 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from scikit-learn->sklearn) (1.1.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\hp\documents\herokudeployment\venv\lib\site-packages (from scikit-learn->sklearn) (1.5.4)
```

```

Anaconda Prompt
hint: Maybe you wanted to say 'git add .'
hint: Turn this message off by running
hint: "git config advice.addEmptyPaths false"

(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>git add .
'git' is not recognized as an internal or external command,
operable program or batch file.

(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>git add .
warning: LF will be replaced by CRLF in earthquake_prediction_and_feature_engineering_final.py.
The file will have its original line endings in your working directory

(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>git commit -m"commit_1
Author identity unknown

*** Please tell me who you are.

Run

git config --global user.email "you@example.com"
git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'hp@DESKTOP-TSUGDBH.(none)')
Logging in... done
Logged in as ruchiofficial27@gmail.com
heroku: Press any key to open up the browser to login or q to exit:
(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>q 4d84-ae6c-8ad5c54be9be?requestor=SFMyNTY.g2gDbQAAAA4xMTUuOTguMTQ1LjIyM24GAHCn01R_AMIAAVGA.
'q' is not recognized as an internal or external command,
operable program or batch file.

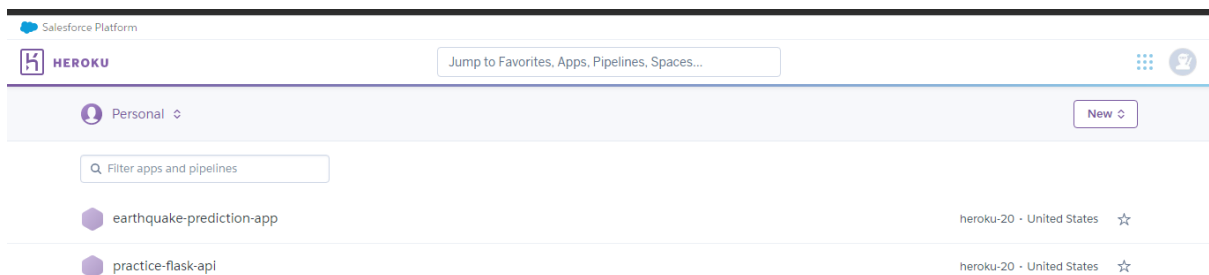
(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/07e3166b-15b4-466c-8224-2b0ea0b096f7?requestor=SFMyNTY.g2gDbQAAAA4xMTUuOTguMTQ1LjIyM24GALC6PVR_AMIAAVGA.
ek6X2Mm172BrY8d-azY6_Y-9la7D82K9rMl73F4Z9I
Logging in... done
Logged in as ruchiofficial27@gmail.com

(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>heroku git:remote -a earthquake-prediction-app
set git remote heroku to https://git.heroku.com/earthquake-prediction-app.git

(venv) (base) C:\Users\hp\Documents\HerokuDeployment\Earthquake prediction>git push heroku master

```

Then we need to signup & login to heroku.com.



Kaggle score: 1.58567

Reference:

<https://www.kaggle.com/c/LANL-Earthquake-Prediction/data>

<https://www.appliedroots.com/course/6/diploma-in-ai-ml-with-uoh>

<https://blog.griddynamics.com/xgboost-vs-catboost-vs-lightgbm-which-is-best-for-price-prediction/>

<https://www.kaggle.com/robikscube/lanl-earthquake-melspectrogram-images-fastai-nn/notebook>

<https://www.analyticsvidhya.com/blog/2021/06/visualizing-sounds-librosa/>

<https://towardsdatascience.com/a-gentle-introduction-to-exploratory-data-analysis-f11d843b8184>

<http://connor-johnson.com/2016/01/24/using-pywavelets-to-remove-high-frequency-noise/>
http://dspace.vsb.cz/bitstream/handle/10084/133114/VAN431_FEI_P1807_1801V001_2018.pdf

kaggle kernel: [LANL Earthquake Prediction : Signal Denoising | Kaggle](#)