**GNANAMANI COLLEGE OF TECHNOLOGY**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING COLLEGE CODE : 6208**

**NAME : NANDHINI M**

**IBM REG NO : au6208211063**

**PROJECT NAME : EARTHQUAKE PREDICTION MODEL USING PYTHON**

**PHASE-3**

**DEVELOPMENT**

Developing an earthquake prediction model is a complex and challenging task, and it's important to note that reliable short-term earthquake prediction remains an unsolved problem in seismology. However, you can build models to assess seismic hazard and probability. Here are key points for developing an earthquake prediction model using Python.

**Data Collection:**

Gather seismic data, including historical earthquake records, fault data, and geological information from reputable sources like USGS Collect features like earthquake magnitudes, depths, and locations.

**Introduction:**

The goal of the project is to predict the time that an earthquake will occur in a laboratory test. The laboratory test applies shear forces to a sample of earth and rock containing a fault line. We note that these are laboratory earthquakes, not real earthquakes. The simulated earthquakes tend to occur somewhat periodically because of the test setup, but this periodicity is not perfect or guaranteed to the modeler attempting to predict the time until an earthquake in the test data provided.

The training input data consists of an acoustic signal that is over 629 million rows long. Each acoustic signal value is associated with a time-to-failure, the time when an earthquake happens. The test data consists of samples that are 150,000 samples long taken from earthquakes different from those in the test set. There is therefore likely to be a need to examine the data in 150,000 sample chunks that represent the test signals.

**Test Environment:**

Designed and run in a Python 3 Anaconda environment on a Windows 10 computer.

import scipy import matplotlib import numpy as np import pandas as pd print(sys.version) print('pandas:', pd.\_\_version\_\_) print('numpy:', np.\_\_version\_\_) print('scipy:', scipy.\_\_version\_\_)

print('matplotlib:', matplotlib.\_\_version\_\_)

**OUTPUT:**

3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)] pandas: 0.24.2 numpy: 1.16.2

scipy: 1.2.1

matplotlib: 3.0.3

**Code Setup:**

%matplotlib inline import os import time import numpy as np import pandas as pd import scipy.signal as sg from tqdm import tqdm\_notebook import matplotlib.pyplot as plt

Define some constants for data location.

DATA\_DIR = r'd:\#earthquake\data' # set for local environment!

TEST\_DIR = r'd:\#earthquake\data\test' # set for local environment!Load the data, this takes a while. There are over 629 million data rows. This data requires over 10GB of storage space on the computer's hard drive.

train\_df = pd.read\_csv(os.path.join(DATA\_DIR, 'train.csv')) print(train\_df.shape)

**OUTPUT:**

(629145480, 2)

**Exploratory Data Analysis:**

Lets validate the test files. This verifies that they all contain 150,000 samples as expected. ld = os.listdir(TEST\_DIR) sizes = np.zeros(len(ld)) for i, f in enumerate(ld):

df = pd.read\_csv(os.path.join(TEST\_DIR, f))

sizes[i] = df.shape[0]

print(np.mean(sizes)) # all were 150,000 print(np.min(sizes))

print(np.max(sizes))

**OUTPUT:**

150000.0

150000.0

150000.0

A basic time series plot of the raw data. Because of the length of the data, the plot samples every 100th data point. These plots are very common on the Kaggle site's kernels section, this one is taken from Preda (2019). Earthquakes occur when the time-to-failure signal (blue) jumps up from very near zero to a much higher value where that new higher value is the time to what is then the next quake. There appears to be a short term high amplitude oscillation very shortly before each quake. But, there also several similar such peaks that occur nearer the region centered in time between quakes. Signal noise seems to increase as time gets closer to failure, though there is also a drop after the big peak. A signal's standard deviation may prove to be a helpful predictor. The region just after the big peak may be especially hard to predict.

train\_ad\_sample\_df = train\_df['acoustic\_data'].values[::100] train\_ttf\_sample\_df = train\_df['time\_to\_failure'].values[::100]

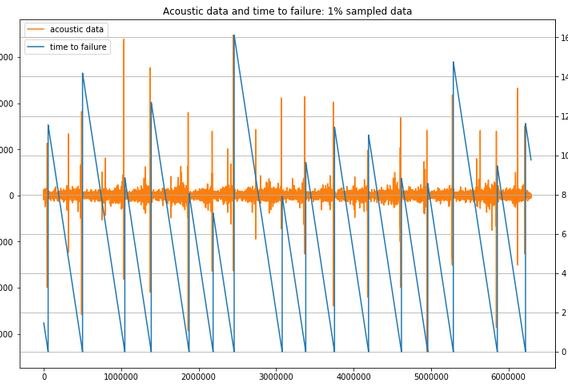
def plot\_acc\_ttf\_data(train\_ad\_sample\_df, train\_ttf\_sample\_df, title="Acoustic data and time to failure: 1% sampled data"): fig, ax1 = plt.subplots(figsize=(12, 8)) plt.title(title)

plt.plot(train\_ad\_sample\_df, color='tab:orange') ax1.set\_ylabel('acoustic data', color='tab:orange') plt.legend(['acoustic data'], loc=(0.01, 0.95)) ax2 = ax1.twinx()

plt.plot(train\_ttf\_sample\_df, color='tab:blue') ax2.set\_ylabel('time to failure', color='tab:blue') plt.legend(['time to failure'], loc=(0.01, 0.9)) plt.grid(True)

plot\_acc\_ttf\_data(train\_ad\_sample\_df, train\_ttf\_sample\_df) del train\_ad\_sample\_df del train\_ttf\_sample\_df

del train\_ad\_sample\_df



From the above plot we can see that 16 earthquakes occur in the data. The earthquakes happen when the time-to-failure reaches very nearly zero and then jumps up. There are only 15 complete time ramps that result in an earthquake and 2 incomplete time ramps. One challenge of this competition is that there are only these very few earthquakes to work with. The is a signal spike (high amplitude) just before an earthquake, but there are also signal spikes in other places that may complicate matters. While the acoustic signal is very large at over 600m rows, the very small number of actual earthquakes available will make machine learning a challenge.

# plot 150k sample slices of the training data, matches size of test data (~0.375 seconds long)

# plots signal and decreasing time to the next quake

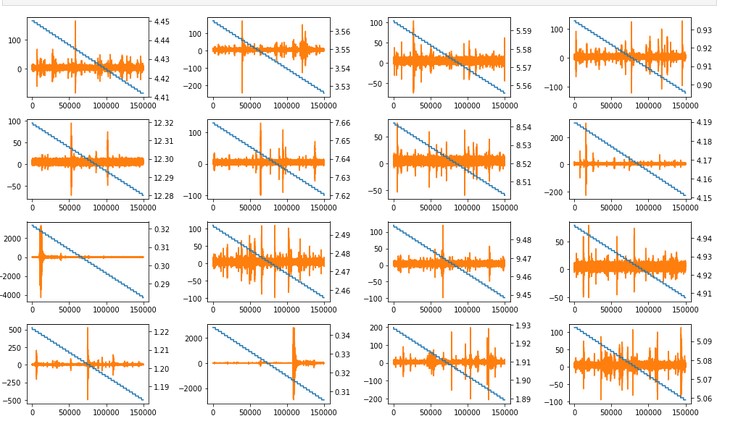
np.random.seed(2018)

rand\_idxs = np.random.randint(0, 629145480-150000, size=16, dtype=np.int32) f, axes = plt.subplots(4, 4, figsize=(14, 8)) i = 0 j = 0 for st\_idx in rand\_idxs:

ad = train\_df['acoustic\_data'].values[st\_idx: st\_idx + 150000] ttf = train\_df['time\_to\_failure'].values[st\_idx: st\_idx + 150000] axes[j][i].plot(ad, color='tab:orange') s = axes[j][i].twinx()

s.plot(ttf, color='tab:blue') i += 1 if i >= 4: i = 0

j += 1 plt.tight\_layout() plt.show()

 Find the indices for where the earthquakes occur, then plotting may be performed in the region around failure.

ttf\_diff = train\_df['time\_to\_failure'].diff() ttf\_diff = ttf\_diff.loc[ttf\_diff > 0]

print(ttf\_diff.index)

Int64Index ([ 5656574, 50085878, 104677356, 138772453, 187641820, 218652630,

245829585, 307838917, 338276287, 375377848, 419368880, 461811623, 495800225,

528777115, 585568144, 621985673], dtype='int64')

Below is a look at the signal just before failure (an earthquake). It is very interesting that the signal becomes quiet in the 150k sample slice before an earthquake. Thus, the signal spike observed in the big picture plot above must occur more than one slice (more than 150k samples) before the earthquake. These do not appear much different than plots 5 or even 8 seconds before the quake that are presented above where the time to failure ramps are shown. This looks like it will create major problems for accurate prediction. Earthquakes are deemed to have occurred where the blue line jumps up in value, representing the time to the next quake.

# plot -150,000 to +30000 samples right around the earthquake

failure\_idxs=[5656574,50085878,104677356,138772453,187641820,21865263,245829585,

307838917, 338276287, 375377848, 419368880, 461811623,

495800225, 528777115, 585568144, 621985673] f, axes = plt.subplots(4, 4, figsize=(14, 8)) i = 0 j = 0

for idx in failure\_idxs:

ad = train\_df['acoustic\_data'].values[idx - 150000: idx + 30000] ttf = train\_df['time\_to\_failure'].values[idx - 150000: idx + 30000]

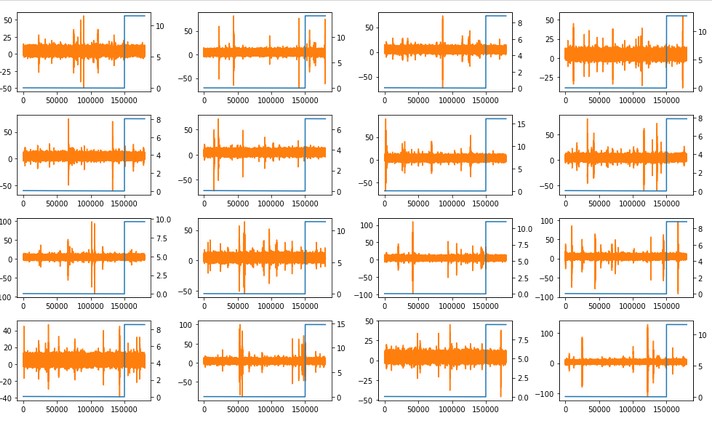
axes[j][i].plot(ad, color='tab:orange')

s = axes[j][i].twinx()

s.plot(ttf, color='tab:blue') i += 1 if i >= 4: i = 0

j += 1 plt.tight\_layout() plt.show()

del ad, ttf



By expanding the time to failure plots, it appears that the big spike in the signal before failure is remarkably consistent. The problem is that the 150k sample slices are actually very short compared to the overall time between quakes and thus these big apparently meaningful signal spikes are unlikely to be present in many training or test samples. If slicing the training data directly into 150k chunks, then only 16 of 4194 training samples (0.38%) would contain a meaningful high-valued spike. It is possible that the analysis of these spikes in some manner could add predictive capability to any small number of test samples that might contain these features, this possibility has not been explored by this author yet.

f, axes = plt.subplots(4, 4, figsize=(14, 8)) i = 0 j = 0

for idx in failure\_idxs:

ad = train\_df['acoustic\_data'].values[idx - 2000000: idx + 30000] ttf = train\_df['time\_to\_failure'].values[idx - 2000000: idx + 30000]

axes[j][i].plot(ad, color='tab:orange') axes[j][i].set\_xticklabels([])

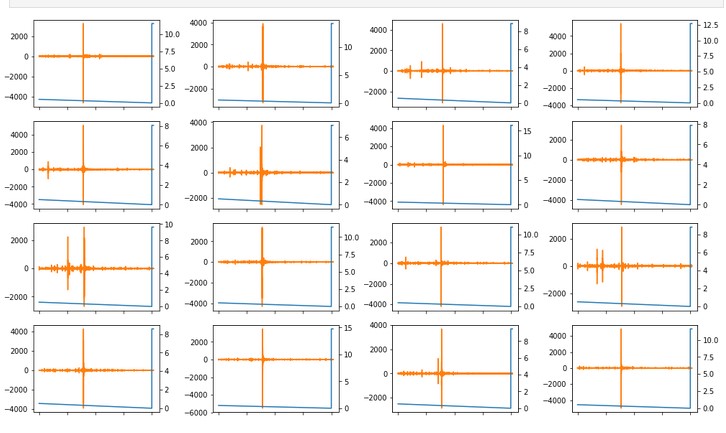
s = axes[j][i].twinx()

s.plot(ttf, color='tab:blue')

1. += 1 if i >= 4: i = 0
2. += 1

plt.tight\_layout() plt.show()

del ad, ttf



A check of the test data in the time domain is presented below, it is difficult to tell from such short signal bursts if they match the character of the training data. None of the very high valued signal spikes were caught in a partial look at the test data. However, these are rare and the existence of the spikes will be taken up again later.

# plot test signals ld = os.listdir(TEST\_DIR)

ld = ld[32:48]

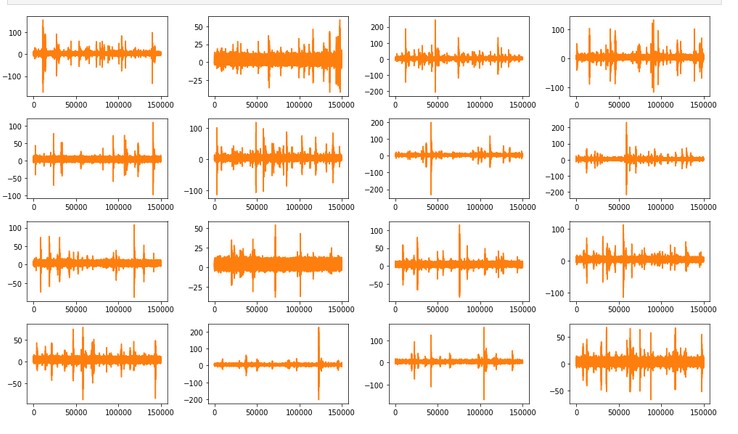
f, axes = plt.subplots(4, 4, figsize=(14, 8)) i = 0 j = 0

for sig\_file in ld:

sig = pd.read\_csv(os.path.join(TEST\_DIR, sig\_file))['acoustic\_data'] axes[j][i].plot(sig, color='tab:orange')

i += 1 if i >= 4: i = 0 j += 1 plt.tight\_layout() plt.show()

del sig



Frequency components of the signal could be very interesting to look at. This is a plot of the Fourier transform magnitude for some of the test signals. Note that there appears to be little information in the signal above the 20,000th frequency line. Noise appears to mostly disappear above the 25,000th frequency line. It is difficult to translate this to a frequency because of the signal gaps noted earlier. Still, it may be best to concentrate signal analysis on frequencies below those represented by the 20,000th frequency line. Also, there are peaks in the frequency analysis that may be valuable to collect in some manner. The DC component was eliminated for plotting purposes because it would otherwise dominate the plot and make the other frequencies hard to see. Also note that while referred to as an "FFT" in the code below, this is actually a Discreet Fourier Transform (DFT) because the signal length of 150k samples is not a number that is a power of two.

# plot frequency components of the signal

MAX\_FREQ\_IDX = 75000

signals = ld[0:12]

fig = plt.figure(figsize=(12, 5))

for i, signal in enumerate(signals): df = pd.read\_csv(os.path.join(TEST\_DIR, signal))

ad = df['acoustic\_data'].values

ad = ad - np.mean(ad) # remove DC component, otherwise it dominates the plot

b, a = sg.butter(6, Wn=20000 / 75000)

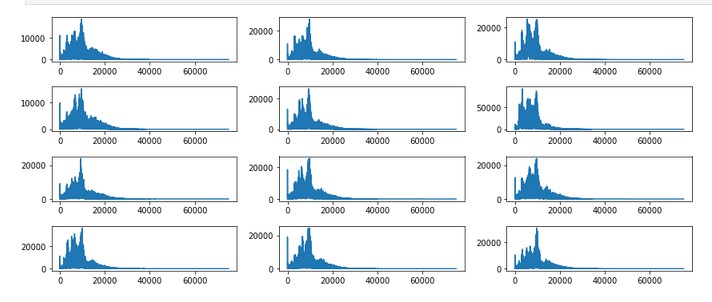
ad = sg.lfilter(b, a, ad) zc = np.fft.fft(ad)

zc = zc[:75000] # eliminate aliased portion of signal per Nyquist criteria realFFT = np.real(zc) imagFFT = np.imag(zc)

magFFT = np.sqrt(realFFT \*\* 2 + imagFFT \*\* 2) plt.subplot(4, 3, i+1)

plt.plot(magFFT, color='tab:blue') plt.tight\_layout()

plt.show()



This is a set of plots of the Fourier transform, windowed by short cosine tapers near the ends. The idea of using the window is to avoid any start up transients that might cause ringing in filters applied to the signal. It is very interesting that windowing seems to emphasize noise. This noise, however, would almost entirely be removed by a low pass or band pass filter. Windows are used to force the signal to be periodic in the time domain which reduces leakage effects at the signal endpoints in the Fourier transform.

# plot frequency components of the signal with a gentle window import warnings from scipy.signal import hann warnings.filterwarnings("ignore")

MAX\_FREQ\_IDX = 75000

ld = os.listdir(TEST\_DIR) signals = ld[0:12]

fig = plt.figure(figsize=(12, 5))

for i, signal in enumerate(signals): df = pd.read\_csv(os.path.join(TEST\_DIR, signal))

ad = df['acoustic\_data'].values

ad = ad - np.mean(ad) # remove DC component, otherwise it dominates the plot

hann\_win = sg.hanning(M=24) ad\_beg = ad[0: 12] \* hann\_win[0: 12] ad\_end = ad[-12:] \* hann\_win[-12:]

ad = np.concatenate((ad\_beg, ad[12: -12], ad\_end), axis=0)

zc = np.fft.fft(ad)

zc = zc[:75000] # eliminate aliased portion of signal per Nyquist criteria

realFFT = np.real(zc) imagFFT = np.imag(zc)

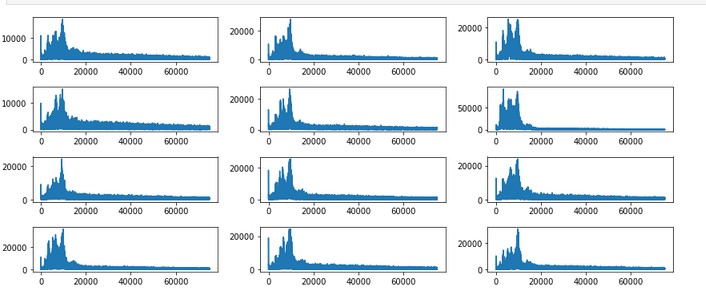
magFFT = np.sqrt(realFFT \*\* 2 + imagFFT \*\* 2)

plt.subplot(4, 3, i+1)

plt.plot(magFFT, color='tab:blue')

plt.tight\_layout()

plt.show()



Phase plots for the Fourier transform. The signal was windowed by small sections taken from a Hanning window along the edges in order to not cause an impulse-like transient in a potential filter at start-up. Phase plots show what appears to be just noise. Probably limited phase features will be all that is desired for the model, perhaps just the standard deviation.

import warnings from scipy.signal import hann warnings.filterwarnings("ignore")

MAX\_FREQ\_IDX = 75000

ld = os.listdir(TEST\_DIR) signals = ld[0:12]

fig = plt.figure(figsize=(12, 5))

for i, signal in enumerate(signals): df = pd.read\_csv(os.path.join(TEST\_DIR, signal))

ad = df['acoustic\_data'].values

ad = ad - np.mean(ad) # remove DC component, otherwise it dominates the plot

hann\_win = sg.hanning(M=24) ad\_beg = ad[0: 12] \* hann\_win[0: 12] ad\_end = ad[-12:] \* hann\_win[-12:]

ad = np.concatenate((ad\_beg, ad[12: -12], ad\_end), axis=0)

zc = np.fft.fft(ad)

zc = zc[:75000] # eliminate aliased portion of signal per Nyquist criteria

realFFT = np.real(zc) imagFFT = np.imag(zc) phzFFT = np.arctan(imagFFT / realFFT) phzFFT[phzFFT == -np.inf] = -np.pi / 2.0 phzFFT[phzFFT == np.inf] = np.pi / 2.0

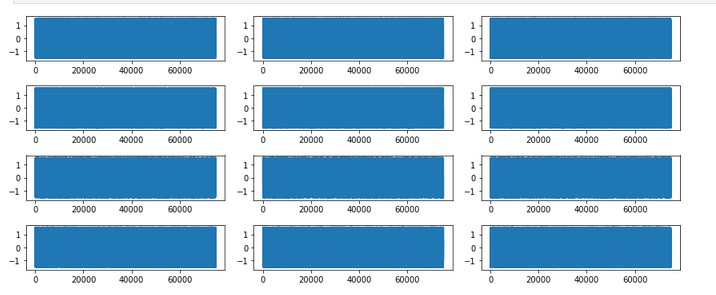
phzFFT = np.nan\_to\_num(phzFFT)

plt.subplot(4, 3, i+1)

plt.plot(phzFFT, color='tab:blue')

plt.tight\_layout()

plt.show()



From the code below, the test set "big peaks" seem to match the proportion with which they are found in the training data at 0.38% of signals (test set signals are all 150k samples long). The number of earthquakes in the training set seems to be approximately proportional to the overall size relationship between the two sets. With 16 quakes in the training set we might expect 10 quakes in the test set based on their relative sizes. As seen by the code output, there are 10 such files within the test set that exhibit the peak behavior. However, there is also concern about the possibility of bogus peaks in the test data that do not correlate well with an actual quake. The expectation of 10 possible quakes in the test set should be considered cautiously and as only an approximation

# determine is a signal contains a 'big peak' as defined by an absolute value more than 2000 ld = os.listdir(TEST\_DIR)

peaks = np.zeros(len(ld))

for i, f in enumerate(ld):

df = pd.read\_csv(os.path.join(TEST\_DIR, f)) peaks[i] = df['acoustic\_data'].abs().max()

peaks\_lg = peaks[peaks >= 2000.0] print(peaks\_lg.shape[0])

print(np.float32(peaks\_lg.shape[0]) / np.float32(peaks.shape[0]) \* 100.0) print(np.float32(2624) / np.float32(4194) \* 16)

# determine is a signal contains a 'big peak' as defined by an absolute value more than 2000 ld = os.listdir(TEST\_DIR) peaks = np.zeros(len(ld)) for i, f in enumerate(ld):

df = pd.read\_csv(os.path.join(TEST\_DIR, f)) peaks[i] = df['acoustic\_data'].abs().max()

peaks\_lg = peaks[peaks >= 2000.0] print(peaks\_lg.shape[0])

print(np.float32(peaks\_lg.shape[0]) / np.float32(peaks.shape[0]) \* 100.0) print(np.float32(2624) / np.float32(4194) \* 16)

**OUTPUT:**

10

0.38109757006168365

10.010491371154785 **Conclusions:**

It appears that this will be a difficult problem and that error is likely to be high relative to the mean time-to-failure. There do not seem to be many obvious features in the signal to correlate with the time to system failure (an earthquake). An examination of the board confirms this as even the best scores show significant error as of the date of this Jupyter notebook. The most obvious feature is the big signal spike just before failure. But, as noted, it is only present in about 0.38% of 150k sample training slices. This may make it a difficult feature to work with and 99.6% of the time the quake needs to be predicted in the absence of the major signal spike.