Chettinad
College of Engineering & Technology
Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Domain Name : Artificial Intelligence

Project Title : Earthquake Prediction Model using Python

1.	**Name of the Student (s)**
	S.No Name of the Student 1 Aarthi S 2 Pavithra R 3 Lathika M 4 Loganayaki M
2.	**Name of the Guide** : Mrs.Suganya
	Department/ Designation : CSE/AP
	Institutional Address : Chettinad College of Engineering and Technology NH-67, Karur-Trichy Highway, Puliyur CF, Karur
	Phone No. & Mobile No. : 6374527207

Title: Earthquake Prediction Model

PHASE: Development Part 2

In this part you will continue building your project.

Continue building the earthquake prediction model by:

- **Visualizing the data on a world map**
- **Splitting it into training and testing sets**

Introduction:

In the realm of geoscience and data-driven insights, the quest To foresee and mitigate the impact of seismic events has led us to Explore the intersection of Python programming and earthquake prediction. Earthquakes, unpredictable in their occurrence yet profoundly impactful, present a challenge that beckons the application of advanced computational Techniques. This project embarks on the development of an Earthquake Prediction Model using the dynamic capabilities of Python, Aiming to decipher the intricate patterns within seismic data.

The urgency of earthquake prediction lies in its potential to transform Reactive responses into proactive measures, offering vital time for Communities to prepare and respond. Python, with its rich ecosystem Of libraries and tools, provides an ideal environment for us to navigate the complexities of seismic data analysis and Machine learning model development

DATABASE:

Date	Time	Latitude	Longitude	Type	Depth	Depth Erro	Depth Seis	Magnitude
1/2/1965	13:44:18	19.246	145.616	Earthquake	131.6			6
1/4/1965	11:29:49	1.863	127.352	Earthquake	80			5.8
1/5/1965	18:05:58	-20.579	-173.972	Earthquake	20			6.2
1/8/1965	18:49:43	59.076	-23.557	Earthquake	15			5.8
1/9/1965	13:32:50	11.938	126.427	Earthquake	15			5.8
1/10/1965	13:36:32	-13.405	166.629	Earthquake	35			6.7
1/12/1965	13:32:25	27.357	87.867	Earthquake	20			5.9
1/15/1965	23:17:42	-13.309	166.212	Earthquake	35			6
1/16/1965	11:32:37	56.452	-27.043	Earthquake	95			6
1/17/1965	10:43:17	-24.563	178.487	Earthquake	565			5.8
1/17/1965	20:57:41	6.807	108.988	Earthquake	227.9			5.9
1/24/1965	0:11:17	2.608	125.952	Earthquake	20			8.2
1/29/1965	9:35:30	54.636	161.703	Earthquake	55			5.5
2/1/1965	5:27:06	-18.697	-177.864	Earthquake	482.9			5.6
2/2/1965	15:56:51	37.523	73.251	Earthquake	15			6
2/4/1965	3:25:00	-51.84	139.741	Earthquake	10			6.1
2/4/1965	5:01:22	51.251	178.715	Earthquake	30.3			8.7
2/4/1965	6:04:59	51.639	175.055	Earthquake	30			6
2/4/1965	6:37:06	52.528	172.007	Earthquake	25			5.7
2/4/1965	6:39:32	51.626	175.746	Earthquake	25			5.8
2/4/1965	7:11:23	51.037	177.848	Earthquake	25			5.9
2/4/1965	7:14:59	51.73	173.975	Earthquake	20			5.9
2/4/1965	7:23:12	51.775	173.058	Earthquake	10			5.7
2/4/1965	7:43:43	52.611	172.588	Earthquake	24			5.7
2/4/1965	8:06:17	51.831	174.368	Earthquake	31.8			5.7
2/4/1965	8:33:41	51.948	173.969	Earthquake	20			5.6
2/4/1965	8:40:44	51.443	179.605	Earthquake	30			7.3
2/4/1965	12:06:08	52.773	171.974	Earthquake	30			6.5
2/4/1965	12:50:59	51.772	174.696	Earthquake	20			5.6
2/4/1965	14:18:29	52.975	171.091	Earthquake	25			6.4
2/4/1965	15:51:25	52.99	170.874	Earthquake	25			5.8
2/4/1965	18:34:12	51.536	175.045	Earthquake	25			5.8
2/4/1965	19:44:04	13.245	44.922	Earthquake	10			5.8
2/4/1965	22:30:03	51.812	174.206	Earthquake	10			5.7
2/5/1965	6:39:50	51.762	174.841	Earthquake	25			5.7
2/5/1965	9:32:11	52.438	174.321	Earthquake	39.5			6.3
2/5/1965	13:38:47	51.946	173.84	Earthquake	30			5.7
2/5/1965	20:47:12	51.738	174.366	Earthquake	20			6
2/5/1965	22:16:02	51.487	176.558	Earthquake	30.4			5.6
2/6/1965	1:40:32	53.008	162.008	Earthquake	17.8			6.4
2/6/1965	4:02:54	52.184	175.505	Earthquake	27.7			6.2
2/6/1965	7:14:45	52.076	172.918	Earthquake	30.1			5.6
2/6/1965	12:22:28	51.744	175.213	Earthquake	37.4			5.7
2/6/1965	14:11:11	52.057	174.116	Earthquake	17.5			5.7
2/6/1965	16:50:29	53.191	161.859	Earthquake	22.5			6.3
2/6/1965	18:10:30	51.447	176.469	Earthquake	25.2			5.8

About Dataset

The National Earthquake Information Center (NEIC) determines the location and size of all significant earthquakes that occur worldwide and disseminates this information immediately to national and international agencies, scientists, critical facilities, and the general public. The NEIC compiles and provides to scientists and to the public an extensive seismic database that serves as a foundation for scientific research through the operation of modern digital national and global seismograph networks and cooperative international agreements. The NEIC is the national data center and archive for earthquake information.

Earthquake Prediction

It is well known that if a disaster has happened in a region, it is likely to happen there again. Some regions really have frequent earthquakes, but this is just a comparative

quantity compared to other regions. So, predicting the earthquake with Date and Time, Latitude and Longitude from previous data is not a trend which follows like other things, it is natural occuring.

Import the necessary libraries required for buidling the model and data analysis of the earthquakes.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
print(os.listdir("../input"))
```

['database.csv']

Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.

In [2]:

```
data = pd.read_csv("../input/database.csv")
data.head()
```

Out[2]:

Date	Time	Latitude	Longitude	Type	Depth	Depth Error	DepthSeismicStations	Magnitude	MagnitudeType	MagnitudeError	MagnitudeSeismicStations	AzimuthalGap	HorizontalDistance	HorizontalError	RootMeanSquare	ID	Source	LocationSource	MagnitudeSource	Status	
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic

Date	Time	Latitude	Longitude	Type	Depth	Depth Error	DepthSeismicStations	Magnitude	MagnitudeType	MagnitudeError	MagnitudeSeismicStations	AzimuthalGap	HorizontalDistance	HorizontalError	RootMeanSquare	ID	Source	LocationSource	MagnitudeSource	Status	
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic

Date	Time	Latitude	Longitude	Type	Depth	Depth Error	DepthSeismicStations	Magnitude	MagnitudeType	MagnitudeError	MagnitudeSeismicStations	AzimuthalGap	HorizontalDistance	HorizontalError	RootMeanSquare	ID	Source	LocationSource	MagnitudeSource	Status
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8											

In [3]:

```
data.columns
```

Out[3]:

```
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error',
       'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',
       'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
       'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID',
       'Source', 'Location Source', 'Magnitude Source', 'Status'],
      dtype='object')
```

Figure out the main features from earthquake data and create a object of that features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.

```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```

Out[4]:

	Date	Time	Latitude	Longitude	Depth	Magnitude
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0
1	01/04/1965	11:29:49	1.863	127.352	80.0	5.8
2	01/05/1965	18:05:58	-20.579	-173.972	20.0	6.2
3	01/08/1965	18:49:43	-59.076	-23.557	15.0	5.8
4	01/09/1965	13:32:50	11.938	126.427	15.0	5.8

Here, the data is random we need to scale according to inputs to the model. In this, we convert given Date and Time to Unix time which is in seconds and a numeral. This can be easily used as input for the network we built.

In:[5]

```
import datetime
import time
```

```
timestamp = []
for d, t in zip(data['Date'], data['Time']):
    try:
        ts = datetime.datetime.strptime(d+' '+t, '%m/%d/%Y %H:%M:%S')
        timestamp.append(time.mktime(ts.timetuple()))
    except ValueError:
        # print('ValueError')
        timestamp.append('ValueError')
```

in:[6]

```
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

In [7]:

linkcode

```
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

Out[7]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08

4	11.938	126.427	15.0	5.8	-1.57026e+08

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08
4	11.938	126.427	15.0	5.8	

Visualization

Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

In [8]:

linkcode

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
```

```
        #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)
in[9]:
 fig = plt.figure(figsize=(12,10))
 plt.title("All affected areas")
 m.plot(x, y, "o", markersize = 2, color = 'blue')
 m.drawcoastlines()
 m.fillcontinents(color='coral',lake_color='aqua')
 m.drawmapboundary()
 m.drawcountries()
 plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py:1704: Matpl
otlibDeprecationWarning: The axesPatch function was deprecated in version 2.1. U
se Axes.patch instead.
  limb = ax.axesPatch
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py:1707: Matpl
otlibDeprecationWarning: The axesPatch function was deprecated in version 2.1. U
se Axes.patch instead.
  if limb is not ax.axesPatch:
```

Splitting the Data

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

```
In [10]:
 X = final_data[['Timestamp', 'Latitude', 'Longitude']]
 y = final_data[['Magnitude', 'Depth']]
In [11]:
 from sklearn.cross_validation import train_test_split

 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
 42)
 print(X_train.shape, X_test.shape, y_train.shape, X_test.shape)
(18727, 3) (4682, 3) (18727, 2) (4682, 3)
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: Deprecation
Warning: This module was deprecated in version 0.18 in favor of the model_selectio
n module into which all the refactored classes and functions are moved. Also note
```

that the interface of the new CV iterators are different from that of this module. Thi
s module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Here, we used the RandomForestRegressor model to predict the outputs, we see the strange prediction from this with score above 80% which can be assumed to be best fit but not due to its predicted values.

In[12]:

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:29: De
precationWarning: numpy.core.umath_tests is an internal NumPy module and shoul
d not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

Out[12]:

```
array([[  5.96,  50.97],
       [  5.88,  37.8 ],
       [  5.97,  37.6 ],
       ...,
       [  6.42,  19.9 ],
       [  5.73, 591.55],
       [  5.68,  33.61]])
```

In [13]:

```
reg.score(X_test, y_test)
```

Out[13]:

```
0.8614799631765803
```

In [14]:

```
from sklearn.model_selection import GridSearchCV

parameters = {'n_estimators':[10, 20, 50, 100, 200, 500]}

grid_obj = GridSearchCV(reg, parameters)
grid_fit = grid_obj.fit(X_train, y_train)
best_fit = grid_fit.best_estimator_
best_fit.predict(X_test)
```

Out[14]:

```
array([[  5.8888 ,  43.532  ],
       [  5.8232 ,  31.71656],
       [6.0034 ,  39.3312 ],
       ...,
```

```
       [  6.3066 ,  23.9292 ],
       [  5.9138 , 592.151  ],
       [  5.7866 ,  38.9384 ]])
```

In [15]:

```
 best_fit.score(X_test, y_test)
```

Out[15]:

```
0.8749008584467053
```

Neural Network model

In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.

In [16]:

```
from keras.models import Sequential
from keras.layers import Dense

def create_model(neurons, activation, optimizer, loss):
    model = Sequential()
    model.add(Dense(neurons, activation=activation, input_shape=(3,)))
    model.add(Dense(neurons, activation=activation))
    model.add(Dense(2, activation='softmax'))

    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

    return model
```

```
Using TensorFlow backend.
```

In this, we define the hyperparameters with two or more options to find the best fit.

In [17]:

```
from keras.wrappers.scikit_learn import KerasClassifier

model = KerasClassifier(build_fn=create_model, verbose=0)

# neurons = [16, 64, 128, 256]
neurons = [16]
# batch_size = [10, 20, 50, 100]
batch_size = [10]
epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'exponential']
```

```
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hinge']

param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, activa
tion=activation, optimizer=optimizer, loss=loss)
```

Here, we find the best fit of the above model and get the mean test score and standard deviation of the best fit model.

In[18]:

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
   print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.666684 using {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'sq
uared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.666684 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'los
s': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'los
s': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.666684 (0.471398) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 's
quared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 's
quared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

The best fit parameters are used for same model to compute the score with training data and testing data.

The best fit parameters are used for same model to compute the score with training data and testing data.

In [19]:

```
model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(3,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='SGD', loss='squared_hinge', metrics=['accuracy'])
```

In [20]:

```
model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_data=(
X_test, y_test))
```

```
Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 4s 233us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 4s 228us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 4/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 5/20
18727/18727 [==============================] - 5s 262us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 8/20
18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 9/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 10/20
18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 11/20
18727/18727 [==============================] - 4s 221us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 12/20
18727/18727 [==============================] - 4s 231us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 13/20
18727/18727 [==============================] - 5s 248us/step - loss: 0.5038 - a
cc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
```

```
Epoch 14/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 15/20
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [==============================] - 4s 225us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [==============================] - 4s 219us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 19/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [==============================] - 5s 258us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
```

Out[20]:

```
<keras.callbacks.History at 0x78dfa2107ef0>
```

In [21]:

```python
[test_loss, test_acc] = model.evaluate(X_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

```
4682/4682 [==============================] - 0s 29us/step
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

linkcode

We see that the above model performs better but it also has lot of noise (loss) which can be neglected for prediction and use it for furthur prediction.

The above model is saved for furthur prediction.

In [22]:

```python
model.save('earthquake.h5')
```

Content

This dataset includes a record of the date, time, location, depth, magnitude, and source of every earthquake with a reported magnitude 5.5 or higher since 1965.

Data Quality Matters: Emphasize the importance of high-quality data. The accuracy of your predictions is directly tied to the reliability and completeness of your dataset.

Feature Selection is Key: Highlight the significance of choosing the right features for your model. Features like historical seismic activity, fault lines, and geological data play a critical role in improving prediction accuracy.

Algorithm Selection: Discuss the algorithms you experimented with and the reasons for choosing a particular one Machine learning algorithms like Random Forest, Support Vector Machines, or neural networks might yield different results, so justify your choice.

Evaluation Metrics: Discuss the evaluation metrics used to assess your model's performance. Common metrics include precision, recall, and F1 score. Explain why you chose these metrics and their implications for earthquake prediction.

Challenges and Limitations: Acknowledge the challenges faced during the project. Whether it's the scarcity of labeled data, computational limitations, or the inherent unpredictability of earthquakes, being transparent about limitations is crucial.

Future Work: Suggest potential improvements and future directions for the project. This could involve incorporating more advanced machine learning techniques, leveraging real-time data, or collaborating with domain experts.

Community Collaboration: Emphasize the collaborative nature of earthquake prediction. Encourage collaboration with the scientific community, as addressing such complex challenges often requires a collective effort