ARTIFICIAL INTELLIGENCE- GROUP 5

PROJECT: EARTHQUAKE PREDICTION MODEL USING PYTHON

PHASE 5: PROJECT DOCUMENTATION & SUBMISSION

SUBMITTED BY-

MOHAMED SALI HESHAM.A:963321104035

INTRODUCTION:

The project on developing an earthquake prediction model using Python involves the application of machine learning algorithms to predict earthquakes. This is considered one of the great unsolved problems in earth sciences.The project leverages the power of Python, a popular language in the field of data science, due to its simplicity and the availability of numerous scientific and numerical libraries. The model aims to predict the magnitude and probability of an earthquake occurring in a particular region based on historical data.The data used for this project is typically collected from seismic monitoring stations. With the increase in technology, many seismic monitoring stations have increased, allowing us to use machine learning and other data-driven methods to predict earthquakes.The earthquake prediction model is not a trend that follows like other things, it happens naturally. Therefore, predicting the earthquake with date and time, latitude and longitude from previous data is a challenging task.This project will take you through how to create a model for the task of Earthquake Prediction using Machine Learning and the Python programming language. It's an exciting journey that combines data science, machine learning, and real-world impact

Objective: The objective of an Earthquake Prediction Model using Python is to develop a system that can forecast the occurrence, location, magnitude, and possibly the timing of future earthquakes. This involves analyzing historical seismic data, geological features, and other relevant parameters to identify patterns and trends that might indicate imminent seismic activity. The ultimate goal is to provide early warning systems to help mitigate potential damages and save lives. Keep in mind that while there have been efforts to develop earthquake prediction models, accurately predicting earthquakes remains a highly complex and uncertain task due to the unpredictable nature of seismic events.

Design Thinking Process:

Data Source Selection: Choose an appropriate dataset containing earthquake data.

Feature Exploration: Analyze and understand the distribution, correlations, and characteristics of key features.

Visualization: Create a world map visualization to display earthquake frequency distribution.

Data Splitting: Split the dataset into a training set and a test set for model validation.

Development Phases:

Data Preparation: Import necessary Python libraries and load the dataset. Convert the given date and time to Unix time which can be used as an entry for the network.

Model Determination: Design a neural network architecture suitable for regression tasks.

Model Compilation: Experiment with different layers, activation functions, and optimizers.

Model Adjustment: Normalize or scale the input features as necessary. Monitor the model's complexity to prevent overfitting.

Model Evaluation: Train the model on the training set and evaluate its performance on the test set.

Prediction System Creation: Develop a system that can use the trained model to make predictions on new data

Data Collection Process: The data for this project is typically collected from seismic monitoring stations. The dataset should contain relevant features such as date, time, latitude, longitude, depth, and magnitude. You can find such datasets on platforms like Kaggle

Data Visualization: Create a world map visualization to display earthquake frequency distribution.Action Utilize geospatial visualization libraries (such as Folium in Python) to plot earthquake data points on a world map. Use colors, sizes, or heatmaps to represent earthquake magnitudes. This visualization will provide a clear global overview of earthquake occurrences.

Let us see how the insights from the analysis can help website owners improve user

The insights from the analysis of an earthquake prediction model can be very valuable for website owners, especially those who operate in the field of geology, disaster management, insurance, and public safety. Here's how these insights can help improve user experience:

Personalized Alerts: Based on the user's location data (with their consent), website owners can provide personalized earthquake alerts. This can help users take necessary precautions in advance.

Interactive Visualizations: Insights from the model can be used to create interactive visualizations (like heatmaps) showing earthquake-prone areas. This can help users understand the data better.

Educational Content: The insights can be used to generate educational content about earthquakes, their causes, effects, and safety measures. This can engage users and encourage them to visit the website more often.

Community Engagement: Website owners can create forums or discussion boards where users can discuss the predictions made by the model. This can foster a sense of community among users.

Real-time Updates: If the model is set up to run regularly, it could provide predicted structural damage following an earthquake in near-real-time. This real-time update feature could significantly improve user experience by providing timely and relevant information

Data Source:

Direct Link: https://www.kaggle.com/datasets/usgs/earthquake-database

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

```
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.

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

	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
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
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
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic

```
data.columns
```

```
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Er ror',
       '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, D ate, Time, Latitude, Longitude, Depth, Magnitude

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

	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 gi ven Date and Time to Unix time which is in seconds and a numeral. This can be easily used as in put for the network we built.

```
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')
```

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

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

	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

Feature Exploration:

For feature exploration in an Earthquake Prediction Model using Python, you'll want to consider various types of data that could potentially influence seismic activity. Here are some feature categories you might want to explore:

- **Geological Features**:
- Geographical coordinates (latitude, longitude)
- Depth of the Earth's crust at the location
- Type of geological formations (e.g., fault lines, tectonic plate boundaries)

- **Seismic Activity History**:
- Historical seismic events in the region (magnitude, date, time)

- Frequency of past earthquakes

- **Geophysical Data**:
- Gravitational anomalies
- Magnetic field data

- **Climate and Environmental Factors**:
- Temperature, humidity, and atmospheric pressure
- Rainfall and water levels
- Changes in groundwater levels

- **Infrastructure and Urbanization**:
- Proximity to urban centers or populated areas
- Density of buildings and infrastructure

- **Topographical Information**:
- Elevation above sea level
- Slope and aspect of the terrain

- **Remote Sensing Data**:
- Satellite imagery for land cover, vegetation, and land use

- **Time Series Data**:
- Changes over time in any of the above features

- **Social and Economic Indicators**:
- Population density
- Socio-economic factors of the region

- **Sensor Data**:
- Data from seismometers, accelerometers, and other monitoring devices

```
def gen_features(X):
    fe = []
    fe.append(X.mean()) fe.append(X.std()) fe.append(X.min())
   fe.append(X.max()) fe.append(X.kurtosis()) fe.append(X.skew())
   fe.append(np.quantile(X,0.01)) fe.append(np.quantile(X,0.05))
   fe.append(np.quantile(X,0.95)) fe.append(np.quantile(X,0.99))
   fe.append(np.abs(X).max()) fe.append(np.abs(X).mean())
   fe.append(np.abs(X).std())
    return pd.Series(fe)
   #Lets read the training set again now in chunks and append features
   train = pd.read_csv('../input/train.csv', iterator=True, chunksize=150_000, dtype
   ={'acoustic_data': np.int16, 'time_to_failure': np.float64})
   X_train = pd.DataFrame()
   y_train = pd.Series() for df
   in train:
    ch = gen_features(df['acoustic_data'])
    X_train = X_train.append(ch, ignore_index=True) y_train =
   y_train.append(pd.Series(df['time_to_failure'].values[-1]))
   X_train.head(10) #Let's check the training dataframe
```

OUTPUT:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	4.884113	5.101106	-98.0	104.0	33.662481	-0.024061	-8.0	-2.0	11.0	18.0	104.0	5.576567	4.333325
1	4.725767	6.588824	-154.0	181.0	98.758517	0.390561	-11.0	-2.0	12.0	21.0	181.0	5.734167	5.732777
2	4.906393	6.967397	-106.0	140.0	33.555211	0.217391	-15.0	-3.0	13.0	26.0	140.0	6.152647	5.895945
3	4.902240	6.922305	-199.0	197.0	116.548172	0.757278	-12.0	-2.0	12.0	22.0	199.0	5.933960	6.061214
4	4.908720	7.301110	-126.0	145.0	52.977905	0.064531	-15.0	-2.0	12.0	26.0	145.0	6.110587	6.329485
5	4.913513	5.434111	-144.0	142.0	50.215147	-0.100697	-10.0	-2.0	12.0	19.0	144.0	5.695167	4.608383
6	4.855660	5.687823	-78.0	120.0	23.173004	0.208810	-12.0	-2.0	12.0	21.0	120.0	5.791007	4.732118
7	4.505427	5.854512	-134.0	139.0	52.388738	-0.176333	-11.0	-2.0	11.0	20.0	139.0	5.415000	5.025126
8	4.717833	7.789643	-156.0	168.0	65.360261	-0.160166	-16.0	-3.0	13.0	26.0	168.0	6.152273	6.714605
9	4.730960	6.890459	-126.0	152.0	53.760207	0.150779	-14.0	-3.0	12.0	23.0	152.0	5.925120	5.895191

Data Visualization:

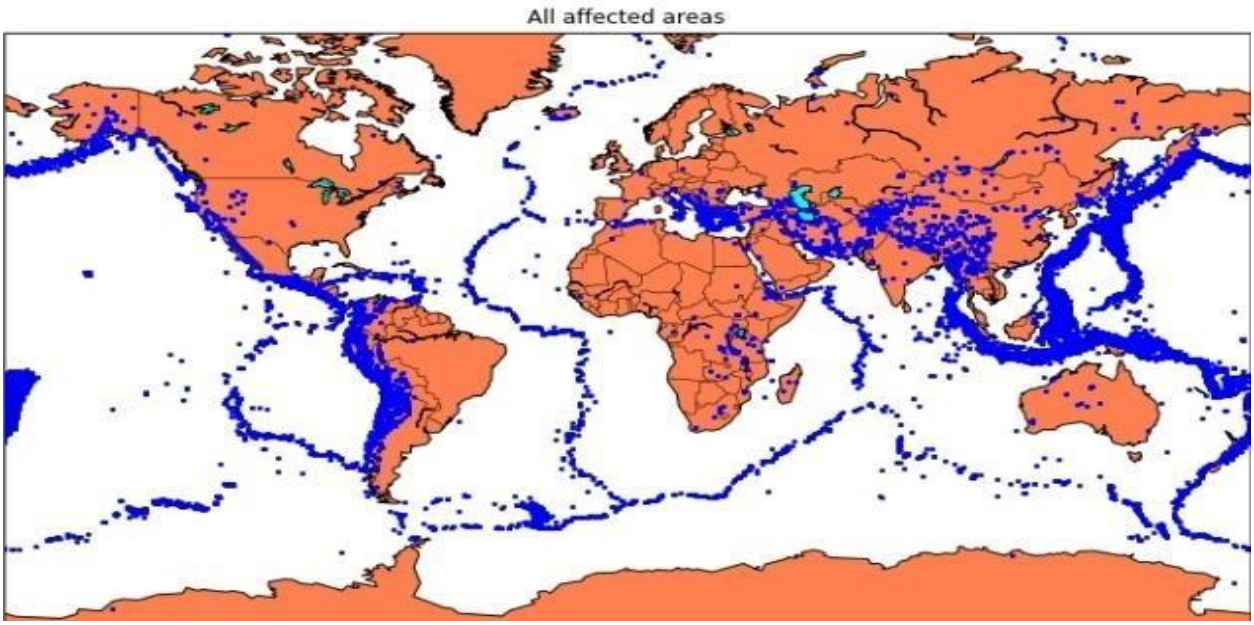
Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,u
rcrnrlon=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)

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()
```



Data Splitting:
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%.

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']] y =
final_data[['Magnitude', 'Depth']]
```

```
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, r
andom_state=42)
print(X_train.shape, X_test.shape, y_train.shape, X_test.shape)
```

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.

```
from sklearn.ensemble import RandomForestRegressor

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

```
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]])
```

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

```
0.8614799631765803
```

```
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)
```

```
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 ]])
```

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

```
0.8749008584467053
```

Neural Network:

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

```
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

```
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', 'expon ential']
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, a
ctivation=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.

```
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_par ams_))
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: 0.957655 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10
, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.333316 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epo chs': 10,
'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'} 0.000000 (0.000000) with:
{'activation': 'sigmoid', 'batch_size': 10, 'epo chs': 10, 'loss': 'squared_hinge',
'neurons': 16, 'optimizer': 'Adadelta'}
0.957655 (0.029957) with: {'activation': 'relu', 'batch_size': 10, 'epochs
': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.645111 (0.456960) with: {'activation': 'relu', 'batch_size': 10, 'epochs
': 10, 'loss': 'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

Training and Evaluation:

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

```
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'])
```

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

```
Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 6s 330us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 6s 320us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 6s 320us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 4/20
18727/18727 [==============================] - 6s 322us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 5/20
18727/18727 [==============================] - 6s 321us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 6s 323us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 6s 322us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 8/20
18727/18727 [==============================] - 6s 321us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 9/20
18727/18727 [==============================] - 6s 322us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
```

Epoch 10/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 11/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 12/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 13/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 14/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 15/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [==============================] - 6s 323us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 19/20

```
18727/18727 [==============================] - 6s 321us/step - loss:
0.503
8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [==============================] - 6s 322us/step - loss:
0.503 8 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
 <keras.callbacks.History at 0x7ff0a8db8cc0>
```

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

```
 4682/4682 [==============================] - 0s 39us/step

Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9
241777017858995
```

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.

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

Conclusion :

Predicting earthquakes is a complex and challenging task, and it's important to note that no reliable method for short-term earthquake prediction currently exists. However, you can use Python for seismic data analysis and research. In conclusion.Python can be a valuable tool for analyzing seismic data, studying earthquake patterns, and conducting research in seismology.Machine learning and deep learning techniques can be applied to identify earthquake patterns and anomalies, but these methods are still in the early stages of development.It's important to rely on established earthquake monitoring agencies and early warning systems for safety and preparedness rather than attempting prediction on your own.Earthquake safety and preparedness are crucial, so make sure to be informed about earthquake risks in your area and have an emergency plan in place.