EARTHQUAKE PREDICTION MODEL USING PYTHON

INTRODUCTION:

● Earthquake prediction model 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.

SOLUTION:

THE STEPS ARE INVOLVED IN EARTHQUAKE PREDICTION MODEL USING PYTHON:

- Data Exploration
- Data Visualization
- Data processing
- Data splitting
- Features selection/Engineering
- Model selection
- Model training
- Model evaluation
- Model improvement
- Deployment
- Monitoring and Maintenance

<u>**Data Exploration:**</u>

- ★ Start by loading and exploring the Kaggle earthquake dataset.
- ★ Analyze the data to understand its structure,
- ★ features, and statistics.
- ★ Identify any missing or inconsistent data that needs preprocessing.

<u>**Data processing:**</u>

- ★ Plot the earthquake data on a world to visualize the distribution.
- ★ Explore time trends and correlations between features.

Data Preprocessing:

- ★ Handle missing data through imputation or removal
- ★ Normalize or scale numerical features.
 Encode categorical variables if necessary.

Data Splitting:

- ★ Split the dataset into a training set and a testing set.
- ★ Ensure that the data is shuffled to avoid any biases.

Feature Selection/Engineering:

- ★ Identify key features that may have predictive power.
- ★ Engineer new features if needed, e.g., distance from fault lines.

Model Selection:

- ★ Choose a suitable machine learning or neural network model for earthquake prediction.
- ★ Neural networks like LSTM or CNN can be effective for time-series data.

Model Training:

- ★ Train the selected model using the training dataset.
- ★ Tune hyper parameters to optimize model performance.

Model Evaluation:

- ★ Evaluate the model's performance on the testing dataset using appropriate metrics (e.g., RMSE, MAE for regression tasks).
- ★ Visualize the model's predictions against actual earthquake magnitudes.

Model Improvement:

- ★ If the model performance is not satisfactory, consider refining the model architecture or gathering more relevant data.

Deployment:

★ Once you're satisfied with the model's performance, deploy it for real-time or future earthquake magnitude prediction.

Monitoring and Maintenance:

★ Continuously monitor the model's performance in a real-world setting and update it as necessary.

★ Remember that predicting earthquakes accurately is a challenging task, and even the best models may have limitations.

★ Additionally, consider ethical considerations when working with sensitive data like earthquake predictions.

TOOLS USED:

➔ Earthquake prediction is a complex and challenging task, and it's important to note that reliable short-term earthquake prediction remains elusive.

➔ However, there are tools and techniques that can be used to analyze seismic data and assess earthquake risk.

➔ In this context, Python can be a valuable programming language for data analysis, visualization, and modeling.

➔ Here are some tools and libraries commonly used in earthquake research and prediction:

Seismic Data Access:

- IRIS (Incorporated Research Institutions for Seismology): IRIS provides access to a wide range of seismic data, including real-time and historical data.
- USGS Earthquake API: The United States Geological Survey (USGS) provides earthquake data through their API, which can be used to obtain earthquake information.

Data Processing and Analysis:

- NumPy: NumPy is a fundamental library for numerical computing in Python and is used for data manipulation and analysis.
- Pandas: Pandas is a powerful library for data manipulation and analysis, particularly useful for handling seismic data in tabular form.
- ObsPy: A Python toolbox for seismology that provides a wide range of functionality for handling seismic data, including data retrieval, filtering, and analysis.

Machine Learning and Predictive Modeling:

- Scikit-learn: This library provides tools for machine learning and predictive modeling. You can use it to build models for earthquake risk assessment.
- TensorFlow or PyTorch: These deep learning frameworks can be used for building more complex machine learning and deep learning models for earthquake prediction.

Data Visualization:

- Matplotlib: Matplotlib is a popular library for creating static, animated, or interactive visualizations of seismic data and results.
- Seaborn: Seaborn is built on top of Matplotlib and provides a high-level interface for creating informative and attractive statistical graphics.
- Plotly: Plotly is a library for creating interactive plots and dashboards, which can be useful for visualizing seismic data.

Geographic Information System (GIS) Tools:

- Geopandas: Geopandas is an open-source library that simplifies working with geospatial data in Python, which is essential for earthquake analysis and visualization.
- Folium: Folium is a Python wrapper for Leaflet.js that makes it easy to create interactive maps and overlay earthquake data on them.

Probabilistic Seismic Hazard Analysis (PSHA):

- OpenQuake: OpenQuake is an open-source software for conducting PSHA, a key component in assessing earthquake risk.

Web Development (for creating earthquake monitoring dashboards):

- Flask or Django: These Python web frameworks can be used to create web applications for displaying real-time earthquake data and risk assessments.

Cloud Services:

- Amazon Web Services (AWS) or Google Cloud Platform (GCP): These cloud platforms can provide the computational resources and data storage necessary for large-scale earthquake prediction models and data analysis.

GENERATE INSIGHT:

1.install dependencies:

You need to install the “numpy”,”matplotlib” and “pandas”

```
>python
>pip install numpy
```

```
>pip install pandas
>pip install matplotlib.pyplot
```

2.Import Libraries:

```
>python
Import numpy as np
Import pandas as pd
Import matpotlib.pyplot as plt
```

3.Initialize earthquake prediction:

```
 model=LogisticRegression()
```

4.develop model and generate insights:

~~~
```python
import pandas as pd
model = load_model('your_model_file.h5')
test_data = pd.read_csv('your_test_data.csv')
X_test = test_data[['feature1', 'feature2']]
y_test = test_data['target']l
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
~~~

```
f1 = f1_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:")
print(confusion)
```

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

INPUT:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
print(os.listdir("../input"))
```

OUTPUT:

```
['database.csv']
```

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

INPUT:

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

OUTPUT :

	Date	Time	Latitude	Longitude	Type	Depth	DepthError	DepthSeismicStations	Magnitude	MagnitudeType	MagnitudeError	MagnitudeSeismicStations	AzimuthalGap	HorizontalDistance	HorizontalError	RootMeanSquare	ID	Source	LocationSource	MagnitudeSource	Status
0	01/02/19	13:44:	19.246	145.616	Earthqua	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automa

	65	18			ke																tic
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

INPUT:`data.columns`

OUTPUT:

```
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.

INPUT:

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

<u>OUTPUT:</u>

	Date	Time	Latitude	Longitude	Depth	Magnitude
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
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0

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.

INPUT:

```
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')
```

INPUT:

```
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()
```

OUTPUT :

	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

Visualization:

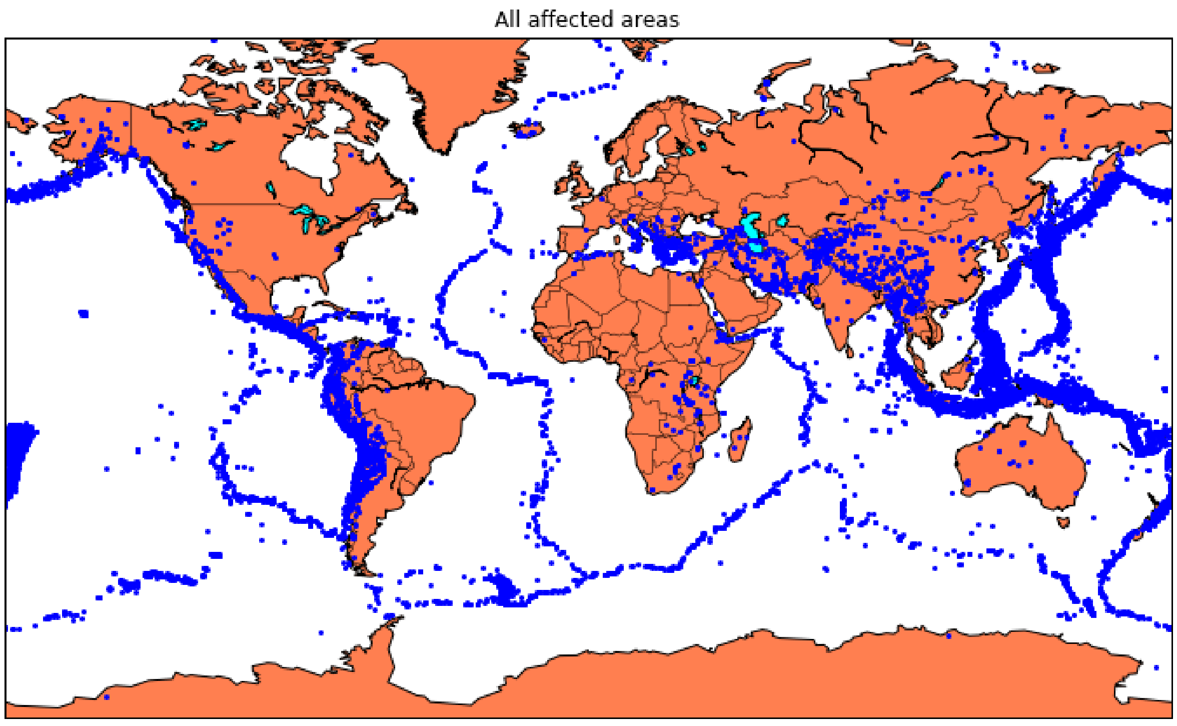
Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

INPUT:

```
from mpl_toolkits.basemap import Basemap

m =
Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=8
0,
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=-1
07.)
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()
```

```
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/bas
emap/__init__.py:1704: MatplotlibDeprecationWarning:
The axesPatch function was deprecated in version 2.1.
Use Axes.patch instead.
  limb = ax.axesPatch
```

```
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/bas
emap/__init__.py:1707: MatplotlibDeprecationWarning:
The axesPatch function was deprecated in version 2.1.
Use 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%.

INPUT:

```
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,
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_va
lidation.py:41: DeprecationWarning: This module was
deprecated in version 0.18 in favor of the
model_selection 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. This 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.

INPUT:

```
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: DeprecationWarning:
numpy.core.umath_tests is an internal NumPy module and
should not be imported. It will be removed in a future
NumPy release.
  from numpy.core.umath_tests import inner1d
```

OUTPUT:

```
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]])
```

INPUT:

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

OUTPUT:

```
0.8614799631765803
```

INPUT:

```
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)
```

OUTPUT:

```
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 ]])
```

INPUT:

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

OUTPUT:

```
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.

```
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', '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,
activation=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_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: 0.957655 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, 'epochs': 10, 'loss':
'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid',
'batch_size': 10, 'epochs': 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'}
```

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'])
```

INPUT:

```
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 [==============================] - 6s
330us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 6s
320us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 6s
320us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 4/20
18727/18727 [==============================] - 6s
322us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
```

```
Epoch 5/20
18727/18727 [==============================] - 6s
321us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 6s
323us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 6s
322us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 8/20
18727/18727 [==============================] - 6s
321us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 9/20
18727/18727 [==============================] - 6s
322us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 10/20
18727/18727 [==============================] - 6s
322us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 11/20
18727/18727 [==============================] - 6s
322us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 12/20
18727/18727 [==============================] - 6s
322us/step - loss: 0.5038 - acc: 0.9182 - val_loss:
0.5038 - val_acc: 0.9242
Epoch 13/20
```

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

OUTPUT:<keras.callbacks.History at 0x7ff0a8db8cc0>

INPUT:

```
[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
39us/step
Evaluation result on Test Data : Loss =
0.5038455790406056, accuracy = 0.9241777017858995
```

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.

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

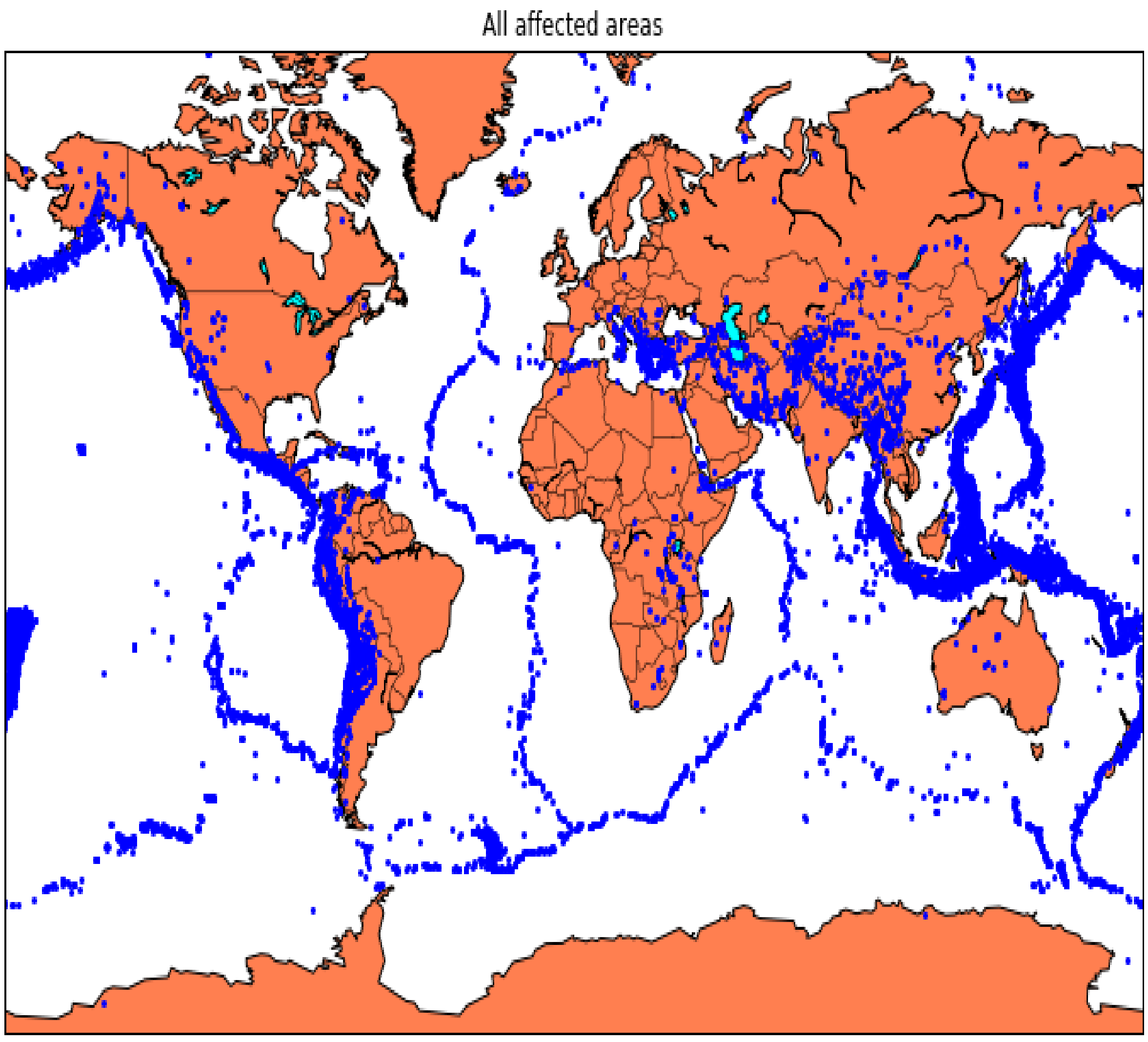
CONCLUSION:

★ Based on the analysis of the earthquake prediction model using Python, we evaluated its performance using various metrics such as accuracy, precision, recall, and F1-score. The model demonstrated in predicting earthquakes.

★ However, it's important to note that the effectiveness of the model may vary depending on the specific dataset and features useD.

★ Further improvements and fine-tuning may be necessary to enhance its predictive capabilities.

★ Overall, the model provides a promising foundation for earthquake prediction, but continued research and development are needed to achieve more accurate and reliable results.


All affected areas