EARTHQUAKE PREDICTION MODEL USING PYTHON

Data Source: Choose a suitable Kaggle dataset containing earthquake data with features like date, time, latitude, longitude, depth, and Magnitude

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

Feature exploration : Analyzing an earthquake dataset involves exploring the distribution, correlations, and characteristics of key features. The date and time data allow us to identify temporal patterns in earthquake occurrence. Latitude and longitude provide spatial information, enabling the mapping of earthquake hotspots. The depth of earthquakes can reveal whether they originate within the Earth's crust or deeper, influencing their potential for damage. Magnitude, often measured using the Richter scale, quantifies the earthquake's strength. Correlations can be examined, for instance, between magnitude and depth, helping to understand seismic behavior. In summary, this dataset combines temporal, spatial, and physical characteristics to provide insights into earthquake patterns and their potential impacts.

Visualization : 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.

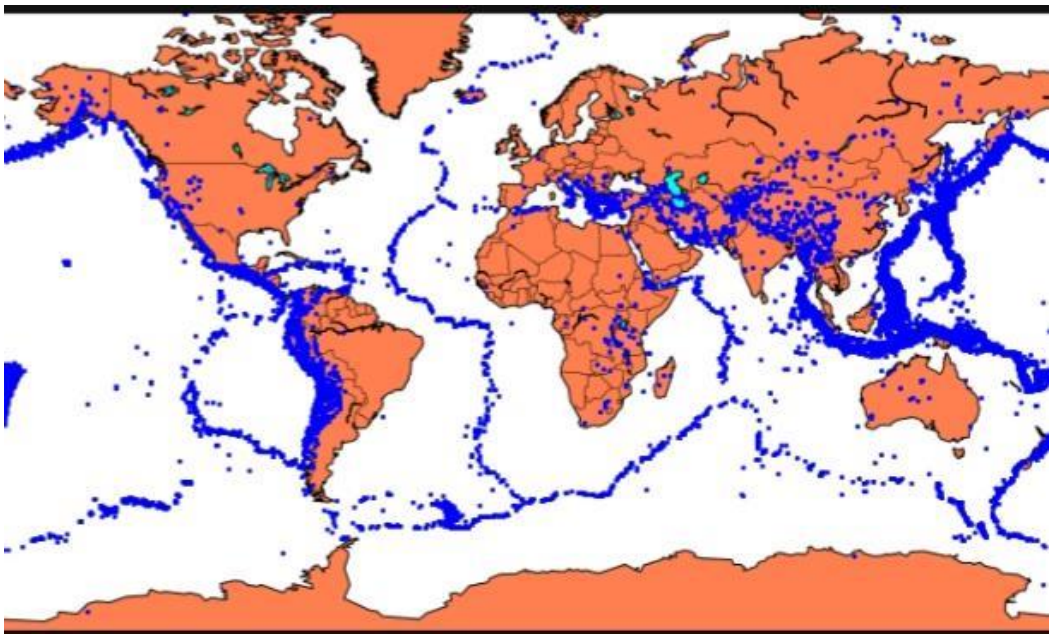
Program:

```
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)
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().drawcountries()
plt.show()
```



Splitting 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%.

Program:

```
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)
```

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.

Model development: 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.

Program:

```
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
```

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

Program:

```
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)
```

Conclusion: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.