**PHASE 3: DEVELOPMENT PART 1 PHASE 3: SUBMISSION DOCUMENT**

**EARTHQUAKE PREDICTION MODEL**

**USING PYTHON**

**TEAM MEMBERS :**

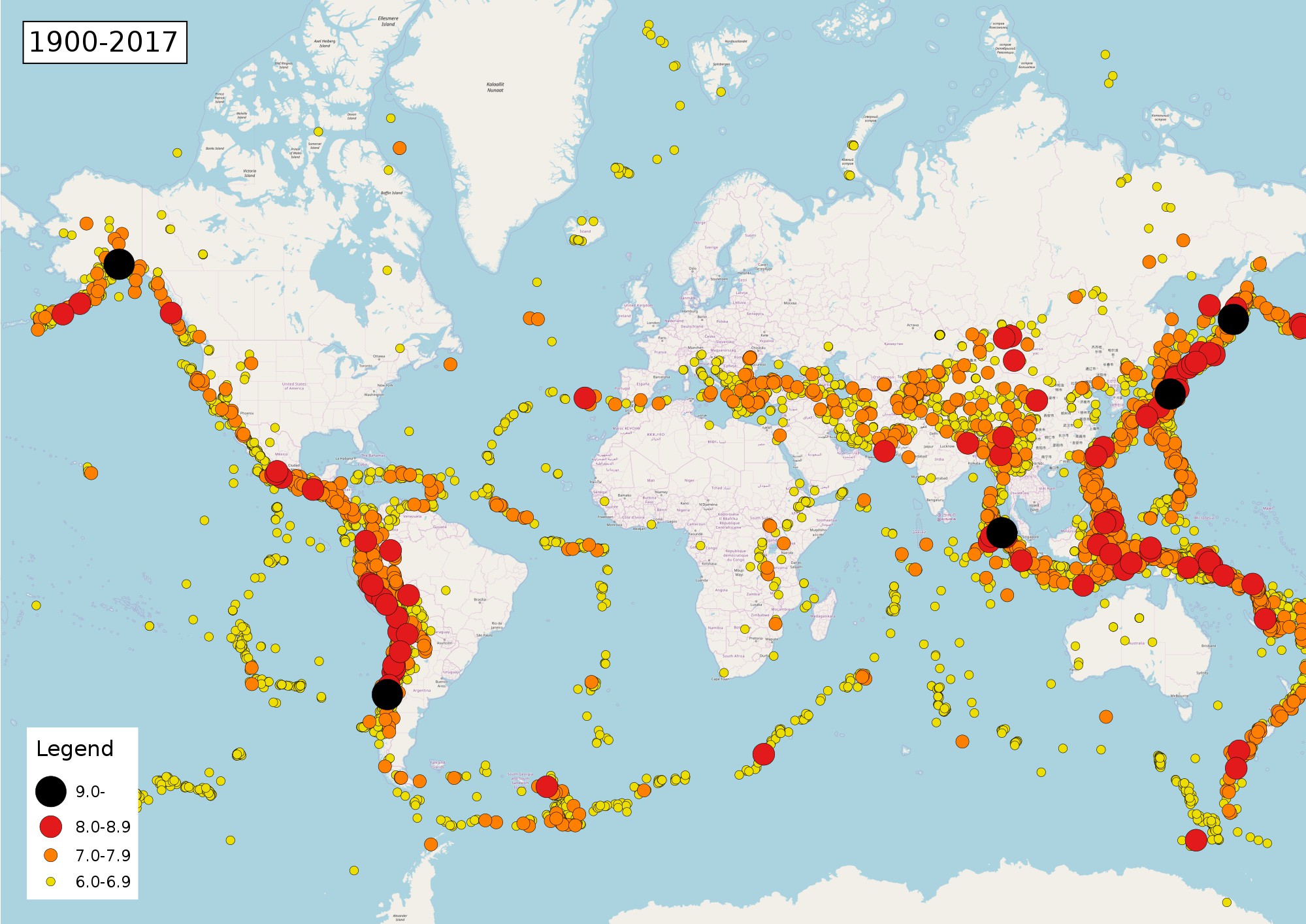
SARAVANA P 113321106086

SARATH KUMAR M 113321106085

YEKOLLA KESAVA 113321106118

PULA GNANA CHAITANYA 113321106071

VEMURU VINAY KUMAR REDDY 113321106113



**INTRODUCTION:**

**Developing an earthquake prediction model is a complex task that involves the analysis of various geophysical data and machine learning techniques.**

**While I can provide a high-level overview of the process using Python, it's essential to note that earthquake prediction remains a challenging and ongoing research topic, and no model can accurately predict earthquakes with certainty.**

**Here's a general outline of how you might approach this using Python:**

# MACHINE LEARNING MODELS:

**Choose appropriate machine learning algorithms, such as random forests, support vector machines, or neural networks.**

**Split the data into training and testing sets**

**for model evaluation.**

# DEPLOYMENT:

**If the model proves effective,it can be deployed in real time systems to provide warnings and help in disaster preparedness.**

# CONTINUOUS MONITORING:

**Continuous monitor and update the model with new data to ensure its effectiveness in predicting earthquakes.**

# MODEL EVALUATION:

**Evaluate the model's performance using metrics like Mean Absolute Error (MAE) or Mean Squared Error (MSE).**

**Perform cross-validation to assess the**

**model's generalisation.**

# MODEL TRAINING:

**Train the machine learning model using historical earthquake data.**

**By using the machine learning library of**

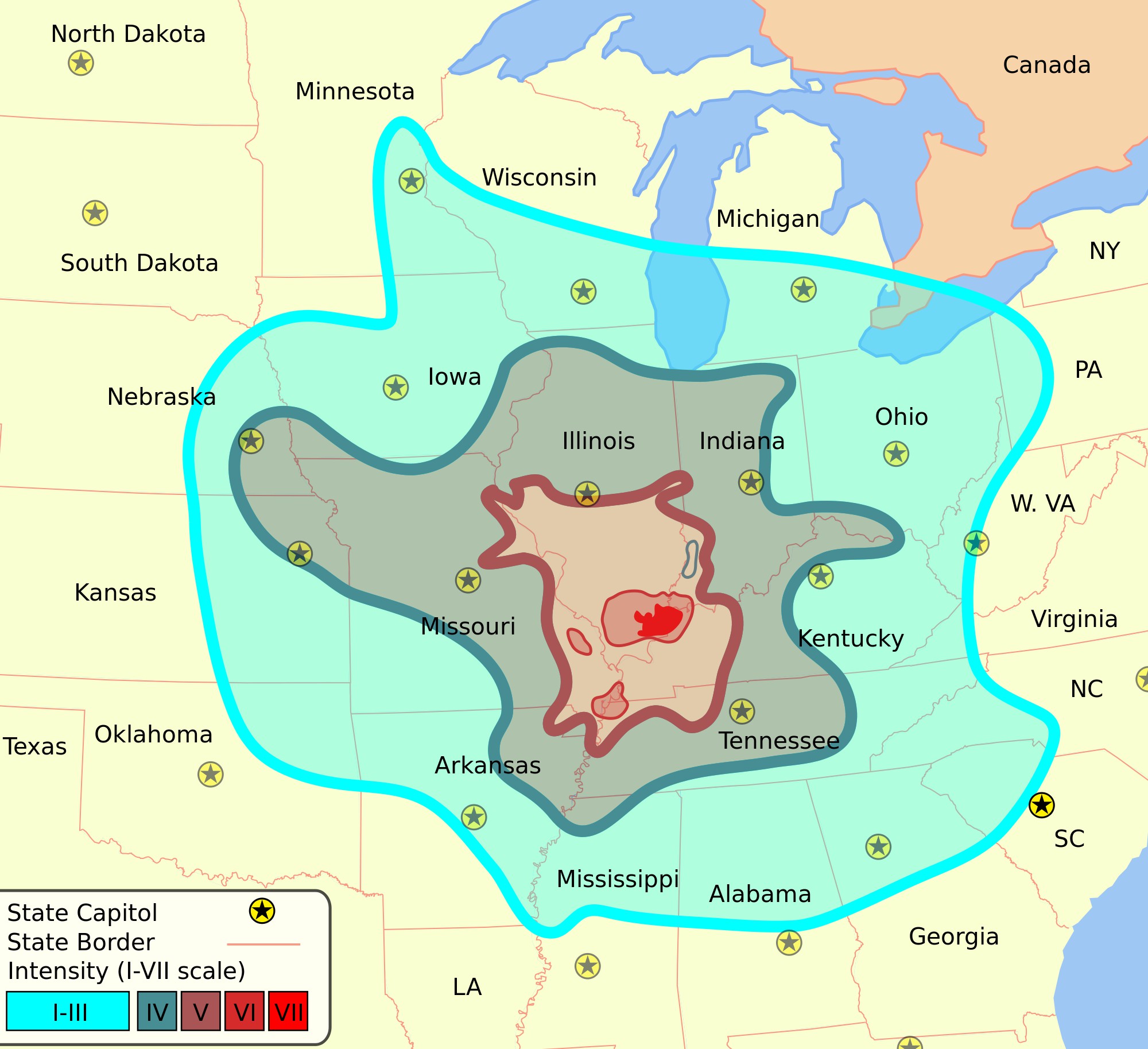
**Spark we build the regression models on the training data set.**

# DATA COLLECTION:

**Gather seismic data, including historical earthquake records, fault line data, and geological information.**

**Consider using APIs like the USGS**

**Earthquake Catalog API to obtain earthquake data.**



# PROGRAM CODE:

**import numpy as np import pandas as pd**

**import matplotlib.pyplot as plt**

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

['database.csv']

data = pd.read\_csv("../input/database.csv")

data.head()

data.columnsIndex(['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')

data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]

data.head()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()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Latitu de** | **Longitu de** | **Dept h** | **Magnitu de** | **Timestam p** |
| **0** | **19.24**  **6** | **145.616** | **131.**  **6** | **6.0** | **-1.57631e+ 08** |
| **1** | **1.863** | **127.352** | **80.0** | **5.8** | **-1.57466e+ 08** |
| **2** | **-20.57**  **9** | **-173.97**  **2** | **20.0** | **6.2** | **-1.57356e+ 08** |
| **3** | **-59.07**  **6** | **-23.557** | **15.0** | **5.8** | **-1.57094e+ 08** |
| **4** | **11.93**  **8** | **126.427** | **15.0** | **5.8** | **-1.57026e+ 08** |

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()

m.drawcountries()

plt.show()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)

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)

**Out[13]:**

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 ],

**Out[14]:**

|  |  |  |  |
| --- | --- | --- | --- |
| **[** | **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)

**Out[15]:**

0.8749008584467053

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

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)

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)) 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, validation\_data=(X\_test, y\_test)) [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

model.save('earthquake.h5')

# CONCLUSION:

**It's crucial to emphasise that earthquake prediction is a challenging problem, and most efforts focus on earthquake monitoring and early warning systems rather than precise prediction.**