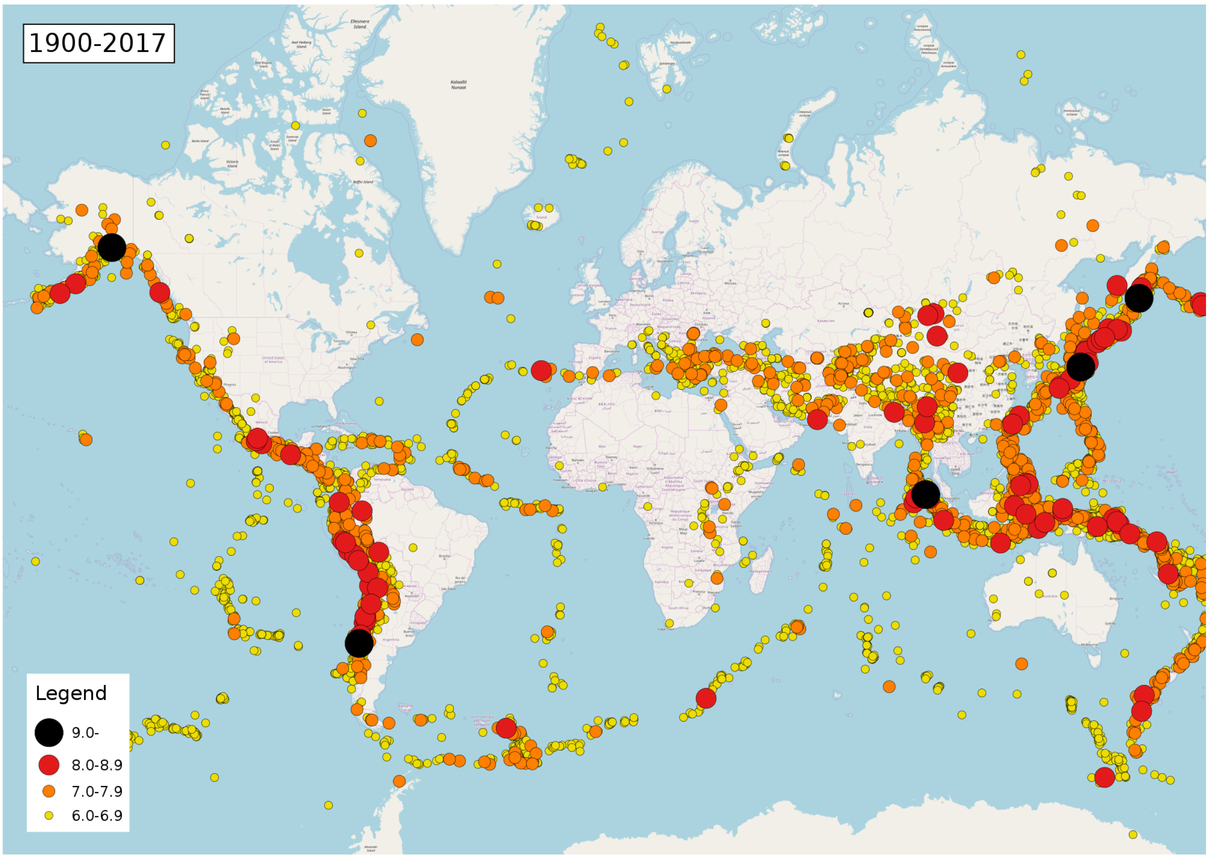
PHASE 3: DEVELOPMENT PART 1

PHASE 3: SUBMISSION DOCUMENT

EARTHQUAKE PREDICTION MODEL USING PYTHON



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:

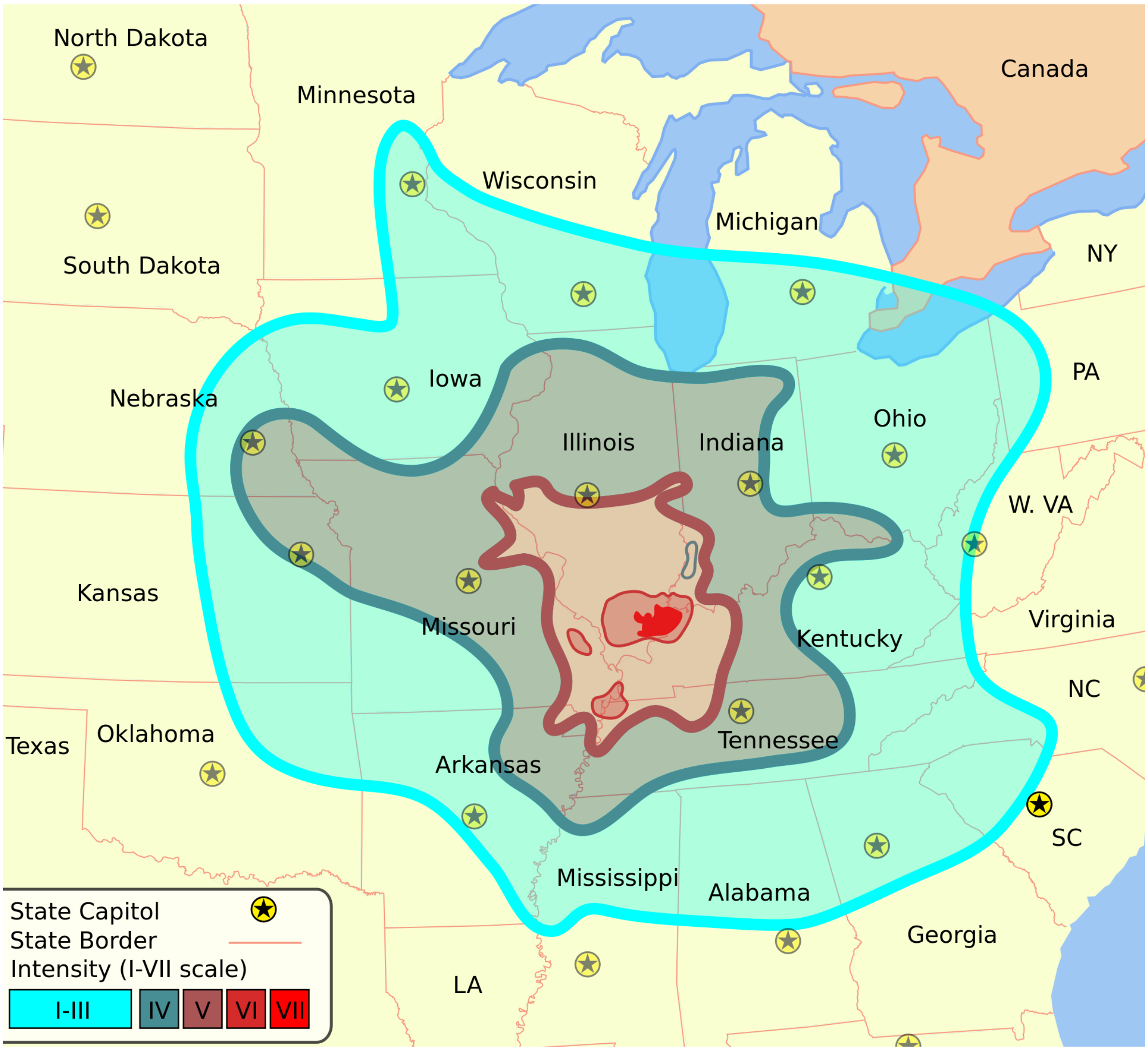
Train the machine learning model using historical earthquake data.

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()
```

	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

```
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)
```

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 ]])
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
```

```python
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.