**EARTH QUAKE PREDICTION**

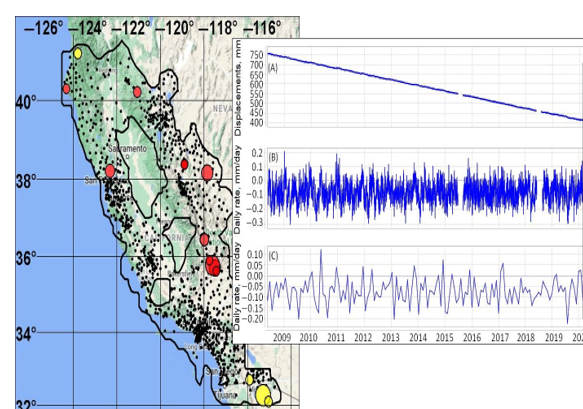
*BATCH MEMBER*

621121104012: GOWRI.K

Phase 4 Submission

***Project Title****: Earth Quake Prediction*

Topic:  The project by performing different activities like Feature engineering, Model training, evaluation.



*Earth Quake Prediction*

Earthquake prediction is a complex and challenging task, and while it's not possible to predict earthquakes with high accuracy, you can still work on earthquake-related projects, such as earthquake risk assessment or aftershock forecasting. Here's a high-level overview of how you can perform different activities like feature engineering, model training, and evaluation using Python.

***1. Data Collection and Preprocessing:***

* Obtain earthquake-related data from sources like USGS Earthquake API or various datasets available online.
* Clean the data, handle missing values, and remove outliers if necessary.

***2. Feature Engineering:***

* Create meaningful features that could potentially capture patterns in the data, such as:
* Geographic features: latitude, longitude, depth, etc.
* Time-based features: date, time of the day, month, etc.
* Seismic features: magnitude, depth, etc.
* Calculate statistical features like mean, median, standard deviation, etc., for various time windows.

***3. Data Visualization:***

* Visualize the data to gain insights and understand the relationships between different features.
* Use libraries like Matplotlib, Seaborn, or Plot for creating visualizations.

***4. Model Training:***

* Choose appropriate machine learning models for earthquake prediction, such as:
* Support Vector Machines (SVM)
* Random Forest
* Gradient Boosting Machines
* Split the data into training and testing sets using techniques like k-fold cross-validation.
* Train the models using the training data.

***5. Model Evaluation:***

* Evaluate the performance of the models using appropriate metrics, such as:
* Mean Squared Error (MSE)
* Root Mean Squared Error (RMSE)
* R-squared value
* Compare the performance of different models to select the best one.

***6. Hyperparameter Tuning:***

* Fine-tune the hyperparameters of the chosen model to improve its performance.
* Use techniques like grid search or random search to find the optimal set of hyperparameters.

***7. Testing and Deployment:***

* + Test the final model with unseen data to ensure its generalization capability.
  + Deploy the model in a suitable environment, depending on the specific requirements of the application.

Example Code (Data Collection and Preprocessing):

import pandas as pd

# Load data

data = pd.read\_csv('earthquake\_data.csv')

# Data preprocessing

# Handle missing values

data = data.dropna()

# Remove outliers

# You can use statistical techniques to identify and remove outliers

# Feature Engineering

# Create new features

data['date'] = pd.to\_datetime(data['timestamp'], unit='s')

data['month'] = data['date'].dt.month

data['hour'] = data['date'].dt.hour

# Visualize the data

# Use libraries like Matplotlib, Seaborn, or Plotly for visualization

# Model Training

# Split the data into features and target variable

X = data[['latitude', 'longitude', 'depth', 'magnitude', 'month', 'hour']]

y = data['target\_variable']

# Train the model

# Choose an appropriate model like Random Forest Regressor or Gradient Boosting Regressor

# Model Evaluation

# Evaluate the performance of the model using suitable metrics

# Hyperparameter Tuning

# Fine-tune the hyperparameters using techniques like Grid Search CV or Randomized Search CV

# Testing and Deployment

# Test the final model on unseen data and deploy it in a suitable environment

Make sure to choose the appropriate libraries and techniques based on the specific requirements of your project. Also, consider the ethical implications and limitations associated with earthquake prediction.

In [1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import os

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

['database.csv']

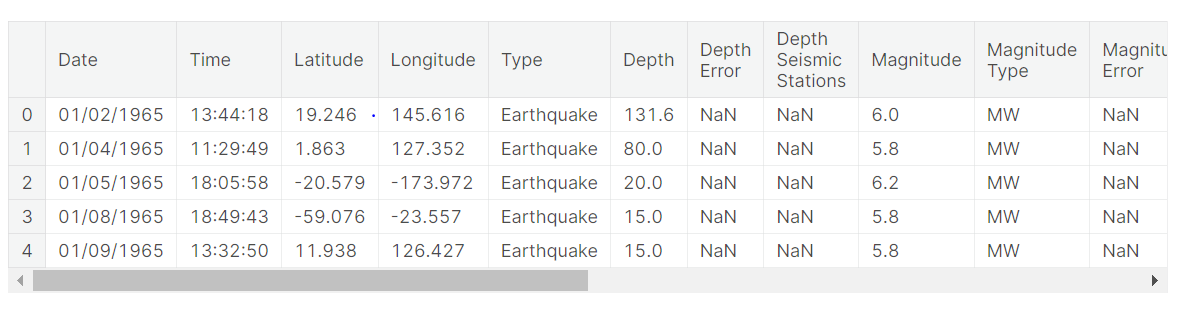
In [2]:

linkcode

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

data.head()

Out[2]:



In [3]:

data.columns

Out[3]:

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'],

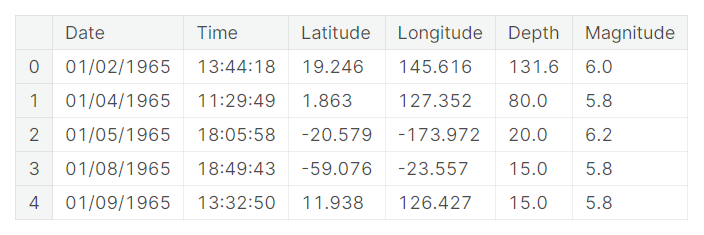
dtype='object')

In [4]:

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

data.head()

Out[4]:

 In [5]:

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')

In [6]:

timeStamp = pd.Series(timestamp)

data['Timestamp'] = timeStamp.values

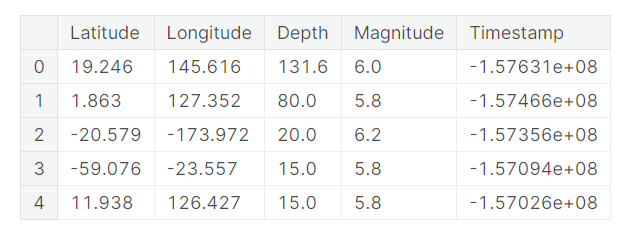
In [7]:

linkcode

final\_data = data.drop(['Date', 'Time'], axis=1)

final\_data = final\_data[final\_data.Timestamp != 'ValueError']

final\_data.head()



***Visualization:***

All the earthquakes from the database in visualized on to the world map clear representation of the locations where frequency of the earthquake will be more.

In [8]:

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)

In [9]:

linkcode

fig = plt.figure(figsize=(12,10))

plt.title("All affected areas")

m.plot(x, y, "o", markersize = 2, color = 'blue')

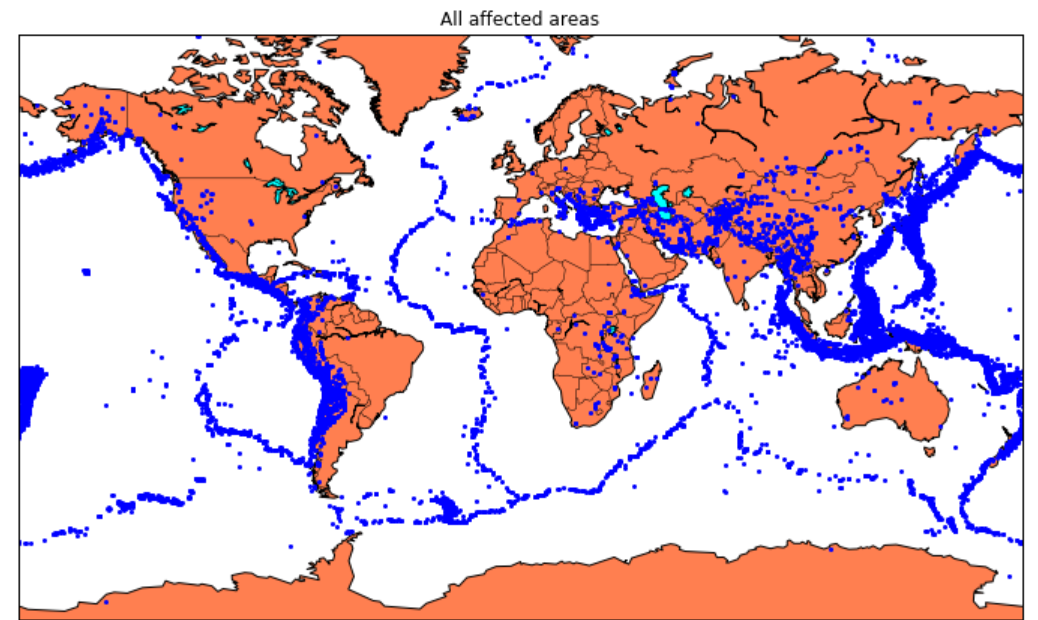
m.drawcoastlines()

m.fillcontinents(color='coral',lake\_color='aqua')

m.drawmapboundary()

m.drawcountries()

plt.show()



### ***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%.

In [10]:

X = final\_data[['Timestamp', 'Latitude', 'Longitude']]

y = final\_data[['Magnitude', 'Depth']]

In [11]:

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)

In [12]:

from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random\_state=42)

reg.fit(X\_train, y\_train)

reg.predict(X\_test)

Out[12]:

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]])

In [13]:

reg.score(X\_test, y\_test)

Out[13]:

0.8614799631765803

In [14]:

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 ]])

In [15]:

best\_fit.score(X\_test, y\_test)

Out[15]:

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.

In [16]:

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

In [17]:

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)

In [18]:

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))

In [19]:

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'])

In [20]:

linkcode

model.fit(X\_train, y\_train, batch\_size=10, epochs=20, verbose=1, validation\_data=(X\_test, y\_test))

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'])

In [20]:

linkcode

model.fit(X\_train, y\_train, batch\_size=10, epochs=20, verbose=1, validation\_data=(X\_test, y\_test))

<keras.callbacks.History at 0x7ff0a8db8cc0>

In [21]:

linkcode

model.save('earthquake.h5')