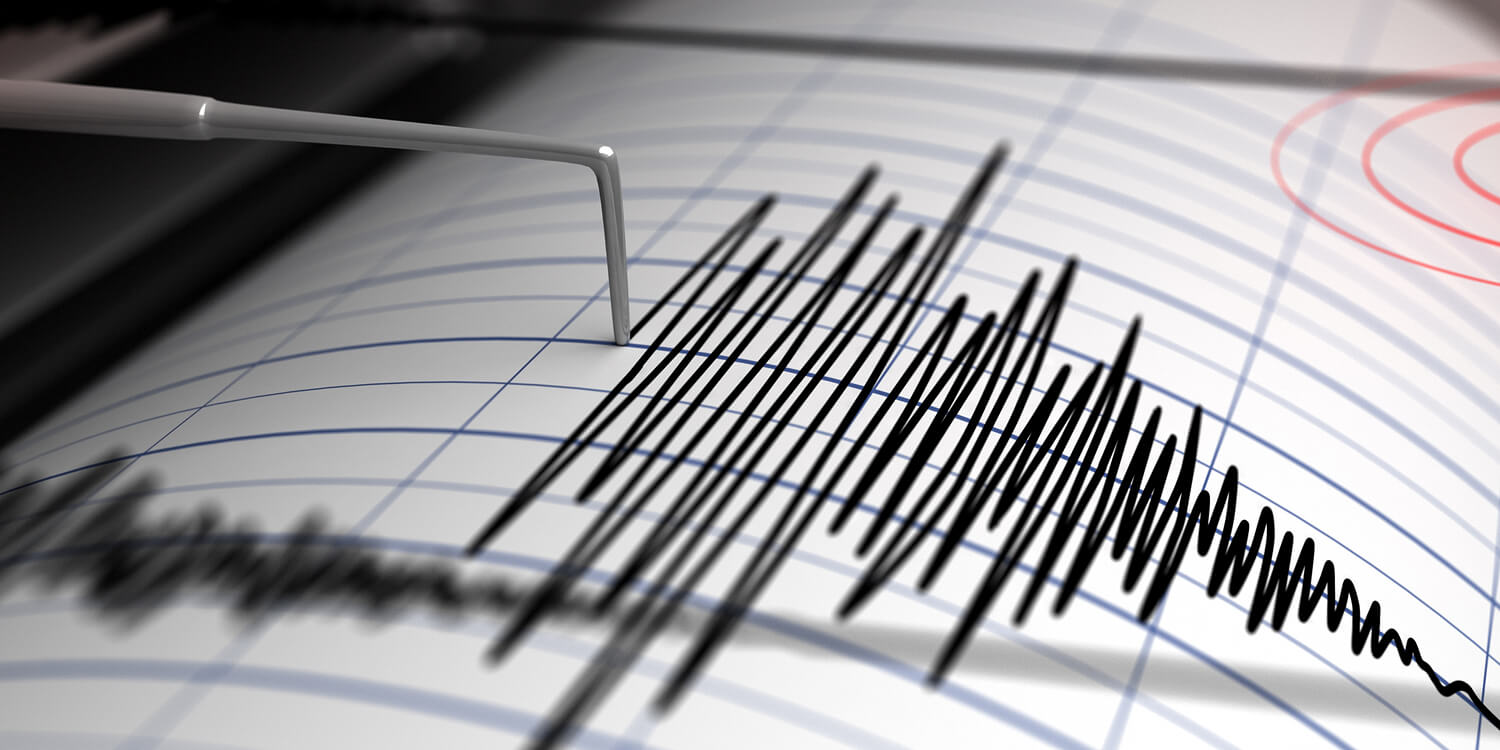
**EARTHQUAKE PREDICTION MODEL USING PYTHON**

**INTRODUCTION:**

****

**EARTHQUAKE**

Earthquake prediction is a complex and challenging endeavor that has eluded scientists for many years. While there have been significant advancements in understanding seismic activity and monitoring, predicting the exact time, location, and magnitude of earthquakes remains highly uncertain. Here are some key points to consider when discussing earthquake prediction systems:

Loading and preprocessing the dataset in Earthquake Prediction model using Python. Begin building the earthquake prediction model by loading and preprocessing the dataset. This Process involves several key steps to prepare the data for prediction and analysis. Let’s break down the steps outlined in the article.

# Loading and preprocessing the dataset in Earthquake Prediction model using Python

## Understanding the Dataset:

* Earthquake Prediction data set is an ordered collection of data.
* As we know, a collection of information obtained through observations, measurements, study, or analysis is referred to as data.
* It could include information such as facts, numbers, figures, names, or even basic descriptions of objects.

## Importing Libraries:

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

import numpy as np

import pandas as pd

import geopandas as gpd

import matplotlib.pyplot as plt

import seaborn as sns

import folium

from folium import Choropleth

from folium.plugins import HeatMap

import datetime

importos

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

## Read the dataset:

* Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.
* Use the open() function(opens a file and returns a file object as a result) to open the . data file in read-binary mode by passing the file name, and mode 'rb' as arguments to it.
* Use the read() function(reads the specified number of bytes from the file and returns them.

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

data.head()

## 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.
* Analyzing earthquake data using the matplotlib library of Python can provide valuable insights into the frequency, magnitude, and location of earthquakes, which can help in predicting and mitigating their impacts.

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)

# Feature Engineering:

* Feature Engineeringhelps to derive some valuable features from the existing ones.
* These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

## 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%.

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

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

* 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.

From sklearn.ensemble import RandomForestRegressor

reg=RandomForestRegressor(random\_state=42)

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

reg.predict(X\_test)

## Neural Network model:

* 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

# Seismic Analysis with Python:

## Data Preprocessing:

* **Date parsing:** Parsing date to dtype datetime64(ns)
* **Time Parsing:** Parsing time to dtype timedelta64
* **Adding Attributes:** ”Date\_Time ” and ” Days “

lengths = data["Date"].str.len()

lengths.value\_counts()

Data Analysis:

* Data analysts can also use Python libraries to structure large datasets and make mathematical operations more manageable.
* Pandas, a Python library, offers a data structure called data frame to effectively work with large tables of data.

Time\_series=sns.<a onclick="parent.postMessage({'referent':'.seaborn.lineplot'}, '\*')">lineplot(x=data['Date'].dt.year,y="Magnitude",data=data, color="#ffa600")

Time\_series.set\_title("Time Series Of Earthquakes Over Years", color="#58508d")

Time\_series.set\_ylabel("Magnitude", color="#58508d")

Time\_series.set\_xlabel("Date", color="#58508d")

## Geospatial Analysis:

* Earth’s crust is divided into numerous segments called the tectonic plates.
* These plates are sometimes moved in little or huge amounts due to the geothermal energy generated in the Mantle of the Earth.
* These movements are mostly seen in the tectonic boundaries and are the major cause of natural Earthquakes.

plates = list(tectonic\_plates["plate"].unique())

for plate in plates:

plate\_vals = tectonic\_plates[tectonic\_plates["plate"] == plate]

lats = plate\_vals["lat"].values

lons = plate\_vals["lon"].values

points = list(zip(lats, lons))

indexes = [None] + [<a onclick="parent.postMessage({'referent':'.kaggle.usercode.13873697.58987243.[5527,5530].i'}, '\*')">i + 1 for <a onclick="parent.postMessage({'referent':'.kaggle.usercode.13873697.58987243.[5527,5530].i'}, '\*')">i, <a onclick="parent.postMessage({'referent':'.kaggle.usercode.13873697.58987243.[5527,5530].x'}, '\*')">x in enumerate(points) if <a onclick="parent.postMessage({'referent':'.kaggle.usercode.13873697.58987243.[5527,5530].i'}, '\*')">i<len(points) - 1 and abs(<a onclick="parent.postMessage({'referent':'.kaggle.usercode.13873697.58987243.[5527,5530].x'}, '\*')">x[1] - points[<a onclick="parent.postMessage({'referent':'.kaggle.usercode.13873697.58987243.[5527,5530].i'}, '\*')">i + 1][1]) > 300] + [None]

for i in range(len(indexes) - 1):

folium.<a onclick="parent.postMessage({'referent':'.folium.vector\_layers'}, '\*')">vector\_layers.PolyLine(points[indexes[i]:indexes[i+1]], popup=plate, color="#58508d", fill=False, ).add\_to(tectonic)

The above preprocessing steps are used to predict the earthquake model using python by loading the given dataset are loaded successfully.

## Visualizing the data on a world map *And Splitting it into training and testing Sets*

**Visualization:**

Visualizing data on a world map and splitting it into training and testing sets are essential steps in data analysis and machine learning. Let's break down these tasks:

## Visualizing Data on a World Map:

**Libraries:**

You can use Python along with libraries like Pandas, Matplotlib, and Plotly for data visualization. For world maps specifically, the geopandas library is very useful as it extends Pandas to allow spatial operations.

**Steps:**

**Data Collection:** First, you need data with geographical information. For example, if you have a dataset with countries and some associated data, you can use it.

**Data Preparation:** Load your data into a Pandas DataFrame. Ensure it contains columns like country names, numerical values for visualization, and possibly latitude and longitude for precise location-based plotting.

## Map Plotting:

* Use geopandas to load a world map shapefile.
* Merge your data with the shapefile based on common keys (like country names).
* Plot the map using Matplotlib or Plotly, coloring countries based on the data values you want to visualize.
* The process of transcribing weather information onto maps, diagrams, etc.
* It usually refers specifically to decoding synoptic reports and entering those data in conventional station-model form on synoptic charts. It is done either manually or by computer.

Example code snippet using Geopandas and Matplotlib:

import geopandas as gpd

import matplotlib.pyplot as plt

# Load world map shapefile

world = gpd.read\_file(gpd.datasets.get\_path('naturalearth\_lowres'))

# Merge your data with the world map data

merged = world.set\_index('name').join(your\_data.set\_index('CountryName'))

# Plotting

fig, ax = plt.subplots(1, 1, figsize=(15, 10))

merged.plot(column='YourDataColumn', cmap='coolwarm', linewidth=0.8, ax=ax, edgecolor='0.8', legend=True)

plt.show()

## Splitting Data into Training and Testing Sets:

**Libraries:**

For splitting data into training and testing sets, you can use the train\_test\_split function from the sklearn.model\_selection module.

**Steps:**

Data Preparation: Ensure your data is cleaned and formatted properly. Convert categorical variables to numerical if needed.

**Splitting Data:**

Separate your features (X) from the target variable (y).

Use train\_test\_split to split the data into training and testing sets.

Example code snippet for splitting data:

from sklearn.model\_selection import train\_test\_split

# X contains features, y contains labels

X = your\_data.drop(columns=['target\_column'])

y = your\_data['target\_column']

# Split data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

* Now you have X\_train, X\_test, y\_train, and y\_test which you can use for training and evaluating your machine learning models.
* Make sure to replace 'YourDataColumn' with the actual column name you want to visualize, and 'target\_column' with the column name representing your target variable.

## Tools for Visualizing Data on a World Map:

**Google Maps API:**

Google Maps provides a robust API for integrating maps into web applications. You can customize the maps and add data layers on top of them.

**Leaflet:**

Leaflet is an open-source JavaScript library for interactive maps. It's lightweight, mobile-friendly, and provides a great platform for creating customized maps.

**Mapbox:**

Mapbox offers a powerful platform for customizing maps and integrating them into web and mobile applications. It provides APIs and libraries for creating visually appealing maps.

**Tableau:**

Tableau is a popular data visualization tool that supports geographical visualizations. It allows you to create interactive dashboards with maps and various data visualization components.

**D3.js:**

D3.js is a JavaScript library for creating dynamic, interactive data visualizations in web browsers. It's highly flexible and can be used to create custom map visualizations.

## Tools for Splitting Data into Training and Testing Sets:

**scikit-learn:**

Scikit-learn is a powerful Python library for machine learning. It provides the train\_test\_split function for easily splitting data into training and testing sets. Scikit-learn also offers various machine learning algorithms for model training and evaluation.

**Pandas:**

Pandas is a Python library for data manipulation and analysis. It provides powerful data structures like DataFrames, which can be used to split data into training and testing sets.

**Excel/Google Sheets:**

For smaller datasets, you can manually split data into training and testing sets using spreadsheet software like Excel or Google Sheets. While this method is not scalable for large datasets, it can be useful for educational purposes or small projects.

**NumPy:**

NumPy is a Python library for numerical computing. It can be used to perform random sampling and splitting of arrays, which is useful when dealing with numerical data in machine learning.

**RapidMiner:**

RapidMiner is a data science platform that provides a visual environment for building machine learning models. It offers various data preprocessing tools, including splitting data into training and testing sets.

When choosing a tool, consider factors such as the size of your dataset, your programming language preference, and the level of customization and interactivity you require in your visualizations.

**Visualizing Data on a World Map using Leaflet(Python):**

To visualize data on a world map using Leaflet in a Python environment, you can utilize the folium library, which allows you to create Leaflet maps easily.

Here's an example of how you can visualize data on a world map using folium in Python:

**Prerequisites:**

Make sure you have the folium library installed. You can install it using pip if you haven't already:

|  |
| --- |
| pip install folium |

**Python Code to Visualize Data on a World Map using Leaflet:**

**Python Code:**

|  |
| --- |
| import folium  # Sample data with latitude, longitude, and a value  data = [  {"name": "New York", "location": [40.7128, -74.0060], "value": 100},  {"name": "London", "location": [51.5074, -0.1278], "value": 200},  # Add more data points as needed  ]  # Create a map centered at [0, 0] with zoom level 2  world\_map = folium.Map(location=[0, 0], zoom\_start=2)  # Add markers to the map based on the data  for city in data:  folium.Marker(location=city["location"], popup=f'Value: {city["value"]}').add\_to(world\_map)  # Save the map as an HTML file  world\_map.save("world\_map.html")  # Optionally, you can also display the map in a Jupyter Notebook  # world\_map |

* In this example, the folium library is used to create a map (world\_map). The data list contains dictionaries with city names, latitude and longitude coordinates, and corresponding values.
* The folium.Marker function is used to add markers for each city on the map. The resulting map is saved as an HTML file (world\_map.html), but you can also display it directly in a Jupyter Notebook by uncommenting the world\_map line.
* Remember to customize the data list with your specific data points and their geographical coordinates. This script will generate an interactive map where you can click on markers to view the associated values.

**Splitting Data into Training and Testing Sets using scikit-learn (Python):**

* The simplest way to split the modelling dataset into training and testing sets is to assign 2/3 data points to the former and the remaining one-third to the latter.
* Therefore, we train the model using the training set and then apply the model to the test set. In this way, we can evaluate the performance of our model.

**Python Code:**

# Import necessary libraries

from sklearn.model\_selection import train\_test\_split

import pandas as pd

# Sample data: features (X) and target variable (y)

data = {

'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Feature2': [11, 12, 13, 14, 15, 16, 17, 18, 19, 20],

'Target': [0, 1, 0, 1, 1, 0, 1, 0, 1, 0]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Split data into features (X) and target variable (y)

X = df[['Feature1', 'Feature2']]

y = df['Target']

# Split data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Print the shapes of the resulting sets

print("X\_train shape:", X\_train.shape)

print("X\_test shape:", X\_test.shape)

print("y\_train shape:", y\_train.shape)

print("y\_test shape:", y\_test.shape)

In this Python example, scikit-learn's train\_test\_split function is used to split the dataset into training and testing sets. The data consists of two features ('Feature1' and 'Feature2') and a binary target variable ('Target').

The test\_size parameter determines the proportion of the dataset that will be included in the test split (in this case, 20%).

These examples demonstrate how to visualize data on a world map using Leaflet (JavaScript) and how to split data into training and testing sets using scikit-learn (Python).

**Conclusion:**

The precision and recall levels achieved for the Python model are acceptable. Perhaps not suited for alerting the general population of some area. But it can be helpful for either governments or high-risk installations (e.g. nuclear power plants) to plan and be in a better preparedness state.The above Python Program tells how we predict the Earthquake and analysis its frequency to protect citizens from harmful disasters. This program should be under Artificial Intelligence sector to predict earthquake using the python with necessary libraries.