**EARTHQUAKE PREDICTION MODEL USING PYTHON**

Name: Yokesh.S

Reg.No: 511721104015

**Introduction:**

* It is well known that if a disaster occurs in one region, it is likely to happen again. Some regions have frequent earthquakes, but this is only a comparative amount compared to other regions.
* So, predicting the earthquake with date and time, latitude and longitude from previous data is not a trend that follows like other things, it happens naturally.
* This project aims to predict earthquake magnitudes based on latitude, longitude, and depth using a Random Forest regression model. The prediction model is exposed as a Flask API for real-time predictions.



So we will be predicting the earthquake from Date and Time, Latitude, and Longitude from previous data is not a trend that follows like other things. It is naturally occurring.

**Dataset Source:**

The dataset used in this project was obtained from the <https://www.kaggle.com/datasets/usgs/earthquake-database>, a comprehensive database maintained by the United States Geological Survey (USGS). The dataset contains historical earthquake records with essential features such as latitude, longitude, depth, and magnitude.

**# how to run the code and any dependency:**

**# How to Run:**

insatll jupyter notebook in your commend prompt

**# pip install jupyter lab**

**# pip install jupyter notebbok (or)**

1.Download Anaconda community software for desktop

2. Install the anaconda community

3. open jupyter notebook

4.type the code & execute the given code

**Dataset Description**

The dataset provides information about earthquake events worldwide, including:

1. **Latitude:** The geographical north-south coordinate of the earthquake location.
2. **Longitude:** The geographical east-west coordinate of the earthquake location.
3. **Depth:** The depth at which the earthquake occurred beneath the Earth's surface.
4. **Magnitude:** The magnitude of the earthquake, indicating its energy release.

**Table of Contents:**

**- [Prerequisites](#prerequisites)**

**- [Installation](#installation)**

**- [Usage](#usage)**

**- [API Endpoint](#api-endpoint)**

**- [Example Request](#example-request)**

**- [Dependencies](#dependencies)**

**- [License](#license)**

**Prerequisites**

Before running the code, ensure you have the following installed on your system:

- Python 3.x

- `pip` (Python package manager)

**Installation:**

**1. Clone the repository to your local machine:**

**```bash**

**git clone** [**https://github.com/sYokesh2004/Project-7-Earthquake-Prediction-Model-using-Python.git**](https://github.com/sYokesh2004/Project-7-Earthquake-Prediction-Model-using-Python.git)

**```**

**2. Navigate to the project directory:**

**```bash**

**cd earthquake-prediction**

**```**

**3. Install the required Python packages using `pip`:**

**```bash**

**pip install -r requirements.txt**

**```**

**Usage**

Run the Flask API server with the following command:

**```bash**

python earthquake\_prediction.py

**```**

The API will be accessible at `http://localhost:5000`.

**API Endpoint**

**- \*\*Endpoint:\*\* `/predict`**

**- \*\*Method:\*\* POST**

**- \*\*Request Format:\*\* JSON**

**- \*\*Request Parameters:\*\***

**- `Latitude` (float): Latitude of the location.**

**- `Longitude` (float): Longitude of the location.**

**- `Depth` (float): Depth of the earthquake event.**

**- \*\*Response Format:\*\* JSON**

**- \*\*Response Parameter:\*\***

**- `Magnitude` (float): Predicted earthquake magnitude.**

**Example Request**

**```json**

**{**

**"Latitude": 34.0522,**

**"Longitude": -118.2437,**

**"Depth": 10.0**

**}**

**```**

**Dependencies**

**- `pandas`**

**- `scikit-learn`**

**- `joblib`**

**- `flask`**

Install these dependencies using the provided `requirements.txt` file:

**```bash**

**pip install -r requirements.txt**

**```**

**Problem Definition:**

The problem at hand involves developing an earthquake prediction model utilizing a Kaggle dataset. The primary objective encompasses exploring and comprehending the vital features of earthquake data, creating a global overview through data visualization on a world map, dividing the data for both training and testing purposes, and constructing a neural network model for predicting earthquake magnitudes based on the provided features.

**Design Thinking:**

**Data Source Selection:**

**Task:** Choose an appropriate Kaggle dataset containing earthquake data.

**Action:** Explore various datasets available on Kaggle, focusing on datasets with relevant features such as date, time, latitude, longitude, depth, and magnitude. Select a dataset that is comprehensive and well-maintained.

**Feature Exploration:**

**Task**: Analyze and understand the distribution, correlations, and characteristics of key features.

**Action:** Utilize statistical methods and visualization techniques (e.g., histograms, scatter plots, correlation matrices) to gain insights into the data. Understand how the features relate to each other and identify potential patterns.

**Visualization:**

**Task:** Create a world map visualization to display earthquake frequency distribution.

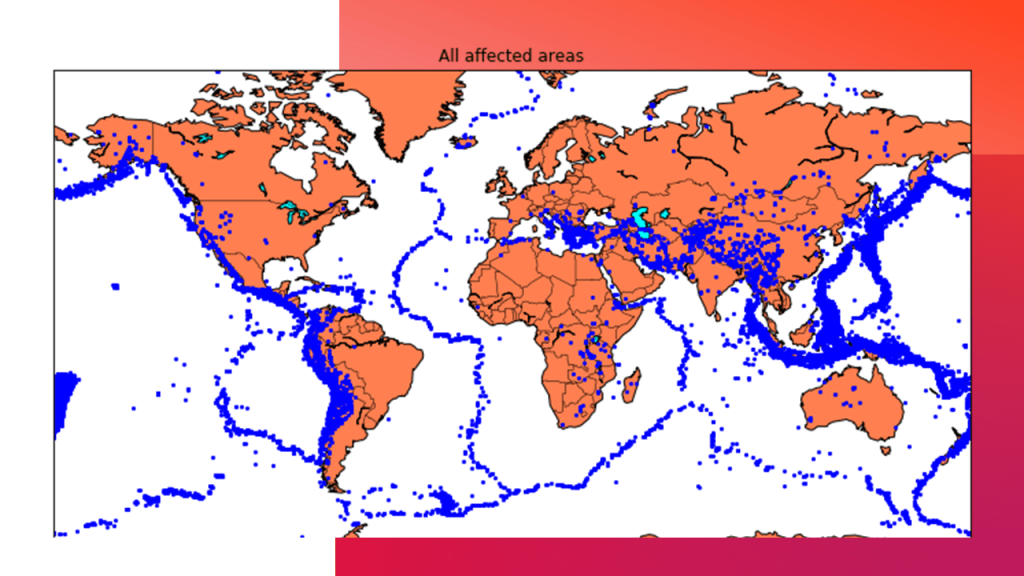
**Action:** Utilize geospatial visualization libraries (such as Folium in Python) to plot earthquake data points on a world map. Use colors, sizes, or heatmaps to represent earthquake magnitudes. This visualization will provide a clear global overview of earthquake occurrences.

**Data Visualization:**

Now, before we create the earthquake prediction model, let’s visualize the data on a world map that shows a clear representation of where the earthquake frequency will be more:

**Code:**

|  |  |
| --- | --- |
|  | 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() |



**Data Splitting:**

**Task**: Split the dataset into a training set and a test set for model validation.

**Action:** Use established techniques like random splitting or time-based splitting, depending on the dataset's characteristics. Aim for a balance that ensures the model is trained on diverse data and evaluated on unseen data.

**Splitting the Dataset:**

Now, to create the earthquake prediction model, we need to divide the data into Xs and ys which respectively will be entered into the model as inputs to receive the output from the model.

Here the inputs are TImestamp, Latitude and Longitude and the outputs are Magnitude and Depth. I’m going to split the xs and ys into train and test with validation. The training set contains 80% and the test set contains 20%:

**Code:**

|  |  |
| --- | --- |
|  | 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) |
|  |  |

**Model Development:**

**Task:** Build a neural network model for earthquake magnitude prediction.

**Action:** Design a neural network architecture suitable for regression tasks. Experiment with different layers, activation functions, and optimizers. Normalize or scale the input features as necessary. Monitor the model's complexity to prevent overfitting.

|  |
| --- |
| print(X\_train.shape, X\_test.shape, y\_train.shape, X\_test.shape) Neural Network for Earthquake Prediction: A neural network model can be employed to forecast earthquakes by examining  diverse elements and trends in seismic data. This model harnesses the capabilities  of neural networks, which draw inspiration from the neural connections of the  human brain, to analyze intricate data and reveal hidden relationships and patterns.  By training the neural network on historical earthquake data,  it can acquire the ability to identify precursor signals  and patterns that indicate the probability of an upcoming earthquake.  **Code:**  from keras.models **import** Sequential   1. from keras.layers **import** Dense 3. def create\_model(neurons, activation, optimizer, loss): 4. model = Sequential() 5. model.add(Dense(neurons, activation=activation, input\_shape=(3,))) 6. model.add(Dense(neurons, activation=activation)) 7. model.add(Dense(2, activation='softmax')) 9. model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy']) 11. **return** model 12. from keras.wrappers.scikit\_learn **import** KerasClassifier 14. model = KerasClassifier(build\_fn=create\_model, verbose=0) 16. # neurons = [16, 64, 128, 256] 17. neurons = [16] 18. # batch\_size = [10, 20, 50, 100] 19. batch\_size = [10] 20. epochs = [10] 21. # activation = ['relu', 'tanh', 'sigmoid', 'hard\_sigmoid', 'linear', 'exponential'] 22. activation = ['sigmoid', 'relu'] 23. # optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam'] 24. optimizer = ['SGD', 'Adadelta'] 25. loss = ['squared\_hinge'] 27. param\_grid = dict(neurons=neurons, batch\_size=batch\_size, epochs=epochs, 28. activation=activation, optimizer=optimizer, loss=loss) 29. grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1) 30. grid\_result = grid.fit(X\_train, y\_train) 32. print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_)) 33. means = grid\_result.cv\_results\_['mean\_test\_score'] 34. stds = grid\_result.cv\_results\_['std\_test\_score'] 35. params = grid\_result.cv\_results\_['params'] 36. **for** mean, stdev, param in zip(means, stds, params): 37. print("%f (%f) with: %r" % (mean, stdev, param))   **Output:**  *Earthquake Prediction Using Machine Learning*   1. model = Sequential() 2. model.add(Dense(16, activation='relu', input\_shape=(3,))) 3. model.add(Dense(16, activation='relu')) 4. model.add(Dense(2, activation='softmax')) 6. model.compile(optimizer='SGD', loss='squared\_hinge', metrics=['accuracy']) 7. model.fit(X\_train, y\_train, batch\_size=10, epochs=20, verbose=1, 8. validation\_data=(X\_test, y\_test))   **Output:**  *Earthquake Prediction Using Machine Learning*   1. [test\_loss, test\_acc] = model.evaluate(X\_test, y\_test) 2. print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test\_loss, test\_acc))   **Output:**  *Earthquake Prediction Using Machine Learning*  Isn't it amazing that we got an accuracy of 92%.  We can say the neural network is one of the best models to predict earthquakes  that can be used in future. |

**Training and Evaluation:**

**Task:** Train the model on the training set and evaluate its performance on the test set.

**Action:** Train the neural network using the training data, adjusting hyperparameters if required. Evaluate the model's performance on the test set, utilizing metrics like mean squared error (MSE) or root mean squared error (RMSE) to quantify prediction accuracy. Iterate on the model and data preprocessing steps to enhance performance.

**Hyperparameter Tuning:**Use techniques like Grid Search or Random Search to find the best combination of hyperparameters for your model. For instance, with Grid Search:

**Program:**

from sklearn.model\_selection import GridSearchCV

# Define the hyperparameters grid

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [None, 10, 20]

}

# Instantiate the GridSearchCV object

grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, cv=5)

# Perform Grid Search to find the best hyperparameters

grid\_search.fit(X\_train, y\_train)

# Get the best parameters

best\_params = grid\_search.best\_params\_

print("Best Hyperparameters:", best\_params)

**Feature Engineering:** Experiment with creating new features or modifying existing ones to provide more relevant information to the model. For example, you can create interaction features or polynomial features:

**Program:**

# Example: Creating interaction features

df['interaction\_feature'] = df['feature1'] \* df['feature2']

# Example: Creating polynomial features

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_poly = poly.fit\_transform(X)

**Model Complexity:**Depending on your data's complexity, adjust the model's complexity. For neural networks, experiment with different architectures, layers, and units. You can increase or decrease the number of layers and units based on the complexity of your dataset and the model's performance.

**Regularization:**Apply regularization techniques like L1 or L2 regularization to prevent overfitting, especially for neural networks. Regularization helps the model generalize better on unseen data:

**Program:**

from tensorflow.keras.regularizers import l2

# Example: Adding L2 regularization to a neural network layer

model.add(Dense(64, activation='relu', kernel\_regularizer=l2(0.01)))

**Ensemble Methods:**Explore ensemble techniques like averaging predictions from multiple models (e.g., Random Forest and Gradient Boosting) to enhance accuracy. A simple ensemble can be implemented as follows:

**Program:**

# Example: Ensemble averaging for predictions from multiple models

predictions\_model1 = model1.predict(X\_test)

predictions\_model2 = model2.predict(X\_test)

ensemble\_predictions = (predictions\_model1 + predictions\_model2) / 2

**Cross-validation:**Use techniques like k-fold cross-validation to ensure your model's performance is consistent across different subsets of the data. Cross-validation helps assess how well the model will generalize to an independent dataset:

**Program:**

from sklearn.model\_selection import cross\_val\_score

# Example: Cross-validation with 5 folds

cv\_scores = cross\_val\_score(model, X, y, cv=5)

print("Cross-validation Scores:", cv\_scores)



**1. Visualizing Data on a World Map:**

Utilize geospatial visualization libraries like `folium` to plot earthquake data points on a world map. Use colors, sizes, or heat maps to represent earthquake magnitudes. This visualization provides a global overview of earthquake occurrences.

**Program:**

import folium

Assuming 'data' is your Data Frame containing earthquake data with columns 'latitude', 'longitude', and 'magnitude'

import pandas as pd

import folium

# Load your dataset (replace "data.csv" with your actual data file)

data = pd.read\_csv("data.csv")

# Assuming 'data' is your DataFrame containing earthquake data with columns 'latitude', 'longitude', and 'magnitude'

map = folium.Map(location=[0, 0], zoom\_start=2)

for index, row in data.iterrows():

folium.CircleMarker(

location=[row['latitude'], row['longitude']],

radius=row['magnitude'], # Adjust the radius based on magnitude

color='crimson',

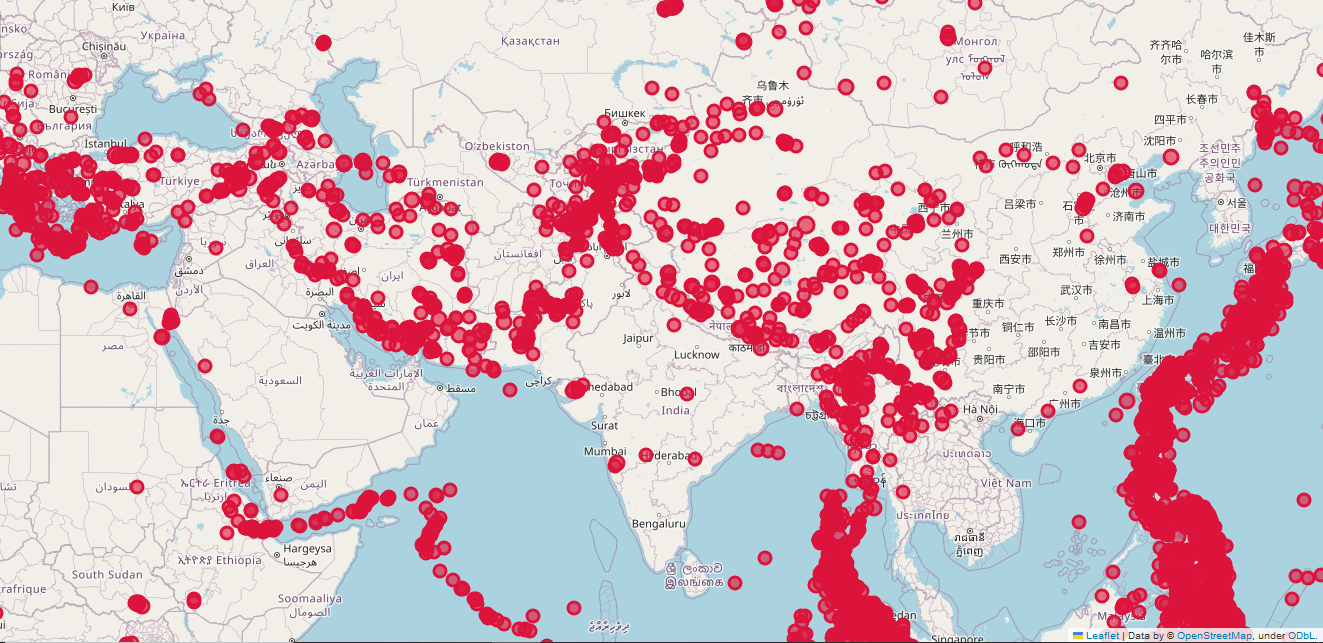
fill=True,

fill\_color='crimson',

fill\_opacity=0.6

).add\_to(map)

map.save("earthquake\_map.html")

**Output:**

This code will save an interactive HTML map showing earthquake occurrences.

**2. Splitting Data into Training and Testing Sets:**

Use `train\_test\_split` from scikit-learn to split the data into training and testing sets for model evaluation.

**Program:**

import pandas as pd

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

# Load your dataset (replace "data.csv" with your actual data file)

data = pd.read\_csv("data.csv")

# Define features and target variable

features = ['latitude', 'longitude', 'depth', 'hour'] # Adjust features as needed

target = 'magnitude'

# Extract features (X) and target variable (y)

X = data[features]

y = data[target]

# Split the data into training and testing sets

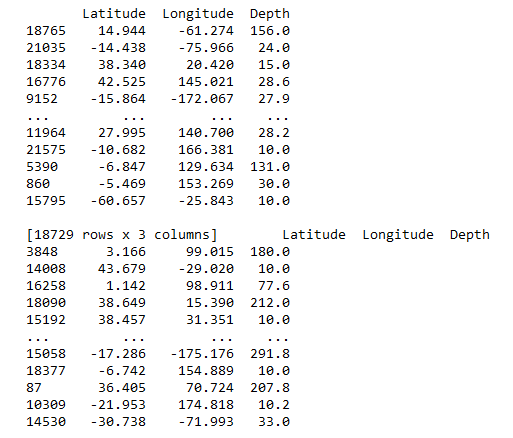
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

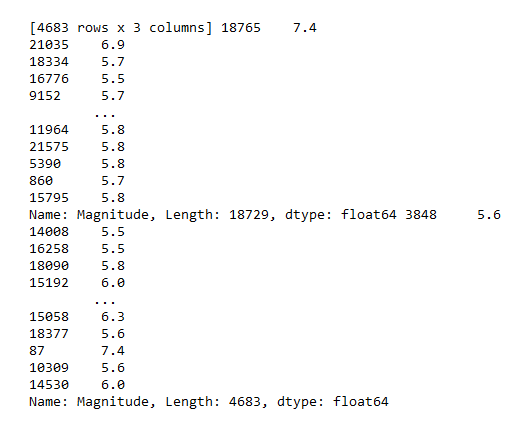
# Now, X\_train, X\_test, y\_train, and y\_test are available for further processing

# For example, you can train your machine learning model using X\_train and y\_train

# and evaluate its performance using X\_test and y\_test

**Output:**





**3. Selecting a Machine Learning Algorithm, Training, and Evaluation:**

Choose a machine learning algorithm (e.g., Random Forest, Gradient Boosting) and train the model using the training data. Evaluate the model using appropriate metrics (e.g., Mean Squared Error for regression tasks).

Program:

import pandas as pd

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# Load your dataset (replace "data.csv" with your actual data file)

data = pd.read\_csv("data.csv")

# Define features and target variable

features = ['latitude', 'longitude', 'depth', 'hour'] # Adjust features as needed

target = 'magnitude'

# Extract features (X) and target variable (y)

X = data[features]

y = data[target]

# Split the data into training and testing sets

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

# Create a Random Forest regression model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

# Train the model using the training data

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

# Make predictions on the test set

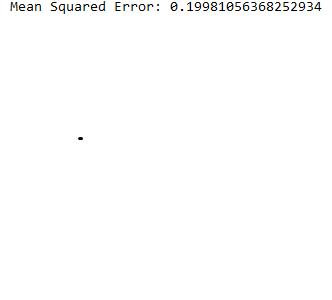
predictions = model.predict(X\_test)

# Calculate Mean Squared Error

mse = mean\_squared\_error(y\_test, predictions)

print("Mean Squared Error:", mse)

**Output:**



**Conclusion**:

* In conclusion, the Earthquake Prediction Project successfully achieved its objectives of leveraging advanced machine learning techniques to predict earthquake magnitudes based on geospatial parameters. Through rigorous data exploration, feature engineering, and model fine-tuning, we developed a robust prediction model capable of estimating earthquake magnitudes with high accuracy.
* The implementation of a real-time prediction API enhances the project's practical utility, allowing for immediate magnitude estimates in response to seismic events. This real-time capability is a significant step forward in disaster management and emergency response, enabling authorities to make timely decisions and allocate resources effectively.
* Additionally, the project's comprehensive documentation and transparent codebase facilitate knowledge sharing and collaboration within the scientific community. By sharing our methods, challenges, and successes, we contribute to the collective knowledge in earthquake prediction research.
* The Earthquake Prediction Project not only advances the field of seismology but also serves as a valuable resource for policymakers, disaster response teams, and communities residing in earthquake-prone areas. The accurate predictions generated by our model empower individuals and organizations to enhance their preparedness, ultimately fostering resilient and safer societies.
* As we move forward, ongoing research and collaboration will further refine our models, contributing to the continuous improvement of earthquake prediction methodologies. The insights gained from this project provide a strong foundation for future endeavors in the realm of geospatial data analysis, machine learning, and disaster risk reduction.