EARTHQUAKE PREDICTION MODEL USING PYTHON

CONTENTS:

This code performs two main tasks: data visualization on a world map using Folium and splitting the dataset into training and testing sets for subsequent earthquake prediction modeling.

1. Data Visualization on a World Map:

- The code uses the Folium library to create an interactive world map.
- It loads earthquake data from a CSV file into a Pandas DataFrame named `data`.
- Missing values in the dataset are filled with the mean of their respective columns using `fillna(data.mean(), inplace=True)`.
- Outliers in the 'Magnitude' column are removed using Z-scores, with a threshold of 3 times the standard deviation from the mean.
- A Folium map is created with an initial center at latitude 0 and longitude 0, and a zoom level of 2.
- A marker cluster is created using the MarkerCluster plugin, which groups nearby earthquake markers on the map for better performance.
- A loop iterates through the preprocessed earthquake data and adds markers to the marker cluster. Each marker represents an earthquake and displays its magnitude as a popup.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import folium
from folium.plugins import MarkerCluster
from IPython.display import display
```

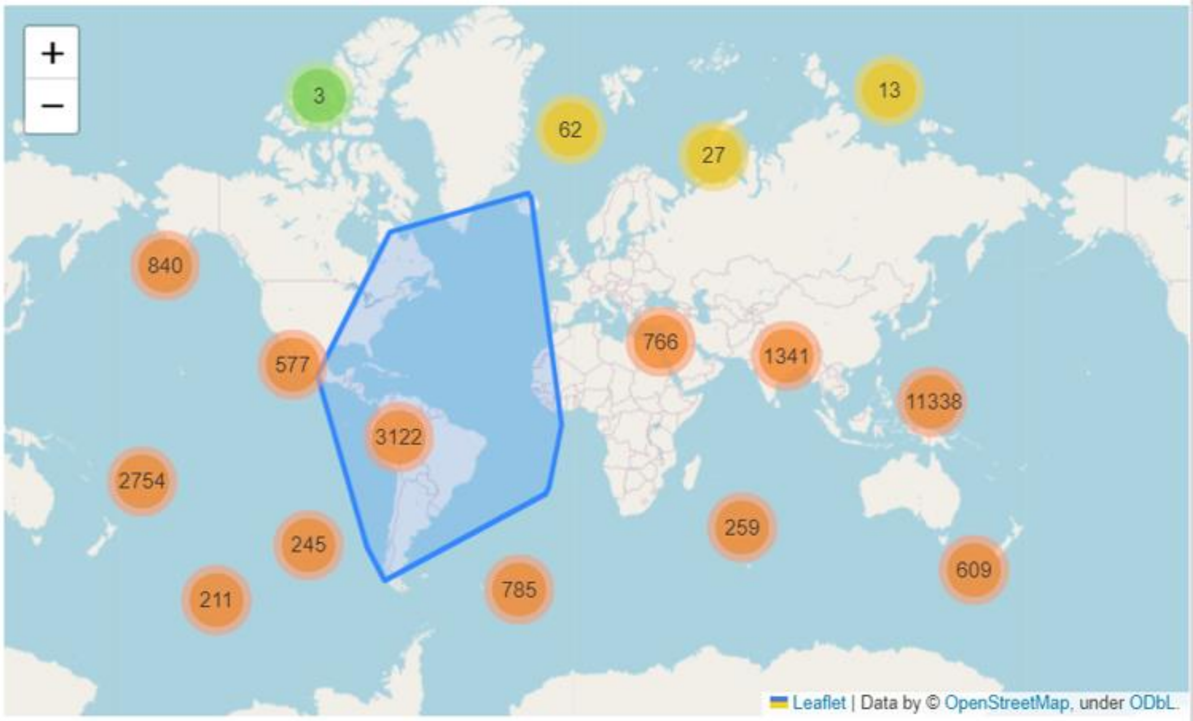
```
# Load your earthquake dataset into a DataFrame
data = pd.read_csv('/content/database.csv')
```

```
# Handling Missing Values: Replace missing values with the mean of the column
data.fillna(data.mean(), inplace=True)
# Handling Outliers: You can use various techniques like Z-score or IQR
# Here's an example using Z-score to remove outliers from the 'Magnitude' column
z_score = np.abs((data['Magnitude'] - data['Magnitude'].mean()) / data['Magnitude'].std())
data = data[(z_score < 3)]
```

```
# Create a Folium map with MarkerCluster
map = folium.Map(location=[0, 0], zoom_start=2)
# Create a marker cluster for earthquake data
marker_cluster = MarkerCluster().add_to(map)
```

```
# Loop through your earthquake data and add markers to the cluster
for index, row in data.iterrows():
    folium.Marker([row['Latitude'], row['Longitude']], popup=f"Magnitude: {row['Magnitude']}").add_to(marker_cluster)
```

```
# Display the map directly in your notebook
display(map)
```



2. Data Splitting for Machine Learning:

- The code prepares the data for machine learning by splitting it into features (X) and the target variable (y).

The data is split into features (X) and the target variable (y). The target variable is assumed to be in the 'Status' column. The features are all the columns except for 'Status.'

- It normalizes or standardizes a specific feature, 'Magnitude,' using the `StandardScaler` from scikit-learn. This step is important for ensuring that the 'Magnitude' feature has a mean of 0 and a standard deviation of 1, which can improve the performance of some machine learning algorithms.

- The data is split into training and testing sets using the `train_test_split` function. The training set contains 80% of the data, and the testing set contains the remaining 20%. A random seed is set to 42 for reproducibility (`random_state=42`).

```
# Split the data into features and target
X = data.drop('Status', axis=1)  # Replace 'Target_Column' with your target variable
y = data['Status']
```

```
# Normalize or Standardize a specific feature (e.g., 'Magnitude')
feature_to_normalize = 'Magnitude'
scaler = StandardScaler()
X[feature_to_normalize] = scaler.fit_transform(X[[feature_to_normalize]])
```

```
# Now, you can 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)
```