EARTHQUAKE PREDICTION MODEL USING PYTHON

Problem Definition:

The problem is to develop an earthquake prediction model using a Kaggle dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features.

Design Thinking:

1)Data Source: Choose a suitable Kaggle dataset containing earthquake data with features like date, time, latitude, longitude, depth, and magnitude.

2)Feature Exploration: Analyze and understand the distribution, correlations, and characteristics of the key features.

3)Visualization: Create a world map visualization to display earthquake frequency distribution.

4)Data Splitting: Split the dataset into a training set and a test set for model validation.

5)Model Development: Build a neural network model for earthquake magnitude prediction.

6)Training and Evaluation: Train the model on the training set and evaluate its performance on the test set.

DATASET:

This dataset includes a record of the date, time, location, depth, magnitude, and source of every earthquake with a reported magnitude 5.5 or higher since 1965.
Dataset Link:
https://www.kaggle.com/datasets/usgs/earthquake-database

PHASES INVOLVED:

1) Dataset loading
2) Data preprocessing
3) Visualizing the data on a world map
4) Split data into training and test sets
5) Model training
6) Model Evaluation

DATASET LOADING:

- Load the earthquake dataset from the file located at '/content/database.csv' into a pandas DataFrame named `data`.

```
import pandas as pd
import numpy as np

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

DATA PREPROCESSING:

1)Handling Missing Values:

- Replace missing values in the dataset with the mean of their respective columns. This is done with the `fillna` method, which fills missing values with the mean of their columns. In this case, missing values are being replaced in all columns

```
# Handling Missing Values: Replace missing values with the mean of the column
data.fillna(data.mean(), inplace=True)
```

2)Handling Outliers:

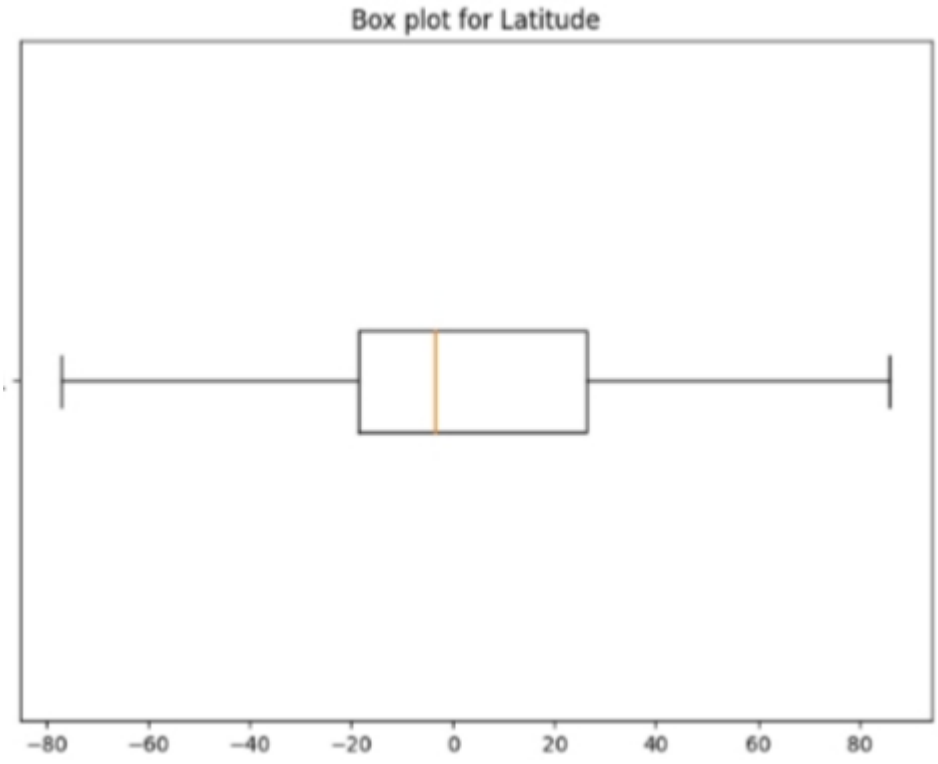
- Next, the code addresses outliers in the dataset by following these steps:
  - It defines a list of numerical columns that are relevant to the analysis.
  - For each numerical column, a box plot is created to visualize the distribution of data and identify potential outliers.
  - Outliers are identified based on the Interquartile Range (IQR) method, and data points that fall outside a specified threshold are considered outliers.
  - Outliers are removed from the dataset, creating a new DataFrame named 'outliers_removed_data' that no longer contains these extreme values.

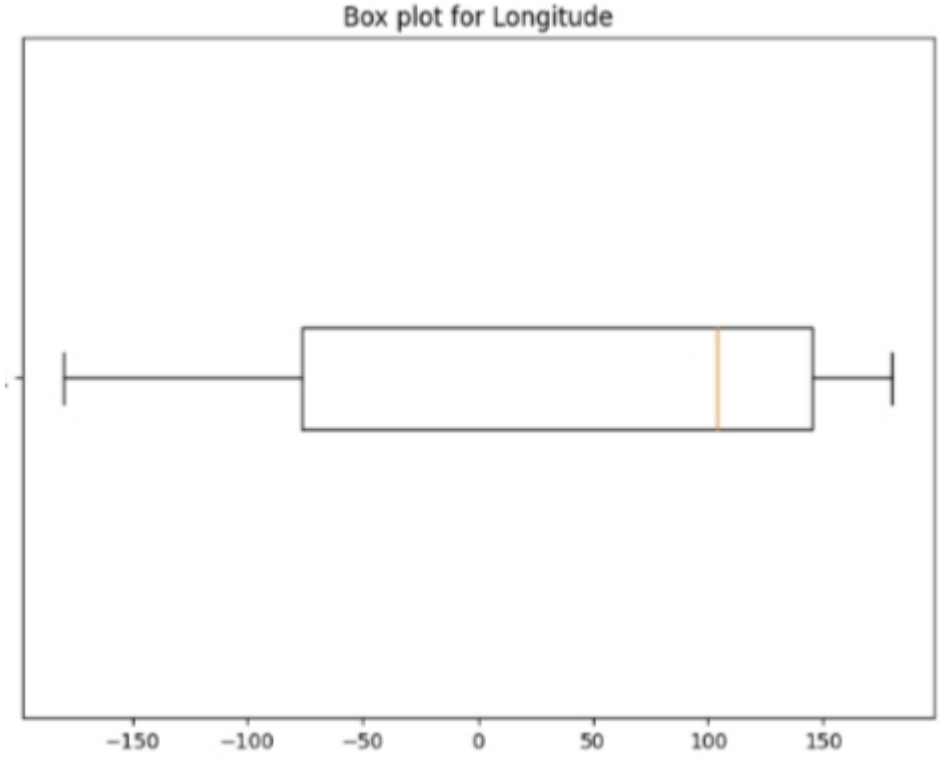
```
import matplotlib.pyplot as plt
```

```
#HANDLING OUTLIERS:
# Select columns with numerical data (excluding Date and Time)
numerical_columns = ['Latitude', 'Longitude', 'Depth', 'Depth Error', 'Depth Seismic Stations', 'Magnitude Error', 'Magnitude Seismic Stations',
'Azimuthal Gap', 'Horizontal Distance', 'Horizontal Error', 'Root Mean Square','Magnitude']
```

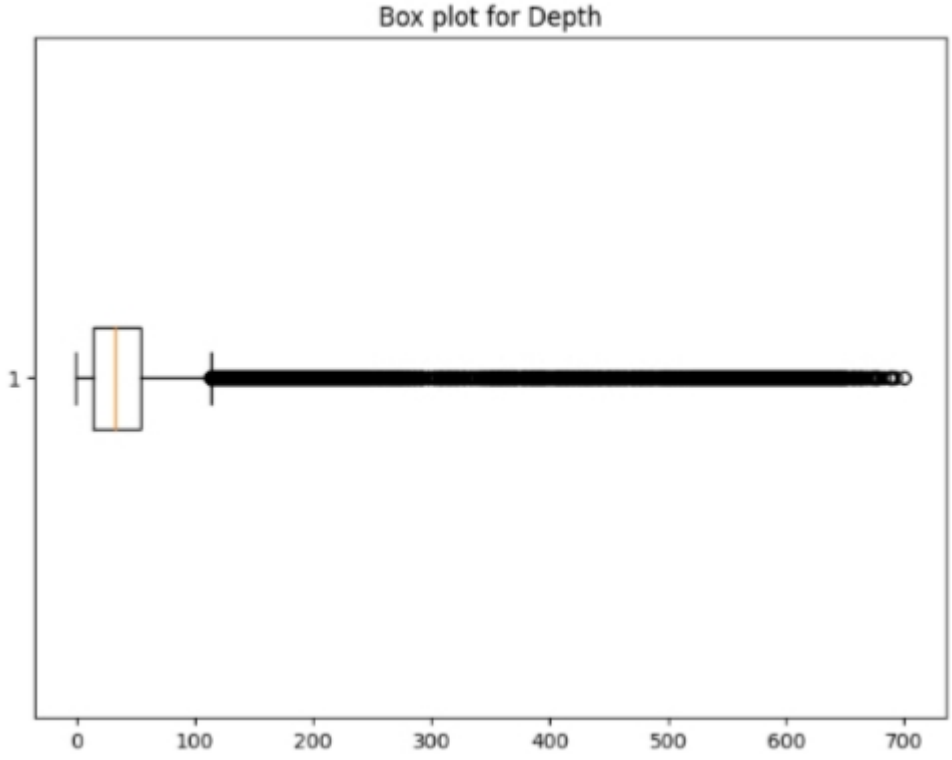
```python
# Create box plots for each numerical column to visualize outliers
outliers_removed_data = data.copy()  # Create a copy of the DataFrame to preserve the original data
for column in numerical_columns:
    plt.figure(figsize=(8, 6))
    plt.boxplot(outliers_removed_data[column], vert=False)
    plt.title(f'Box plot for {column}')
    plt.show()
    # Define a threshold for considering data points as outliers
    threshold = 1.5

    # Calculate the IQR (Interquartile Range) for the column
    Q1 = outliers_removed_data[column].quantile(0.25)
    Q3 = outliers_removed_data[column].quantile(0.75)
    IQR = Q3 - Q1

    # Identify outliers and remove them from the DataFrame
    outliers = (outliers_removed_data[column] < (Q1 - threshold * IQR)) | (outliers_removed_data[column] > (Q3 + threshold * IQR))
    outliers_removed_data = outliers_removed_data[~outliers]

# The outliers_removed_data now contains your dataset with outliers removed.
```

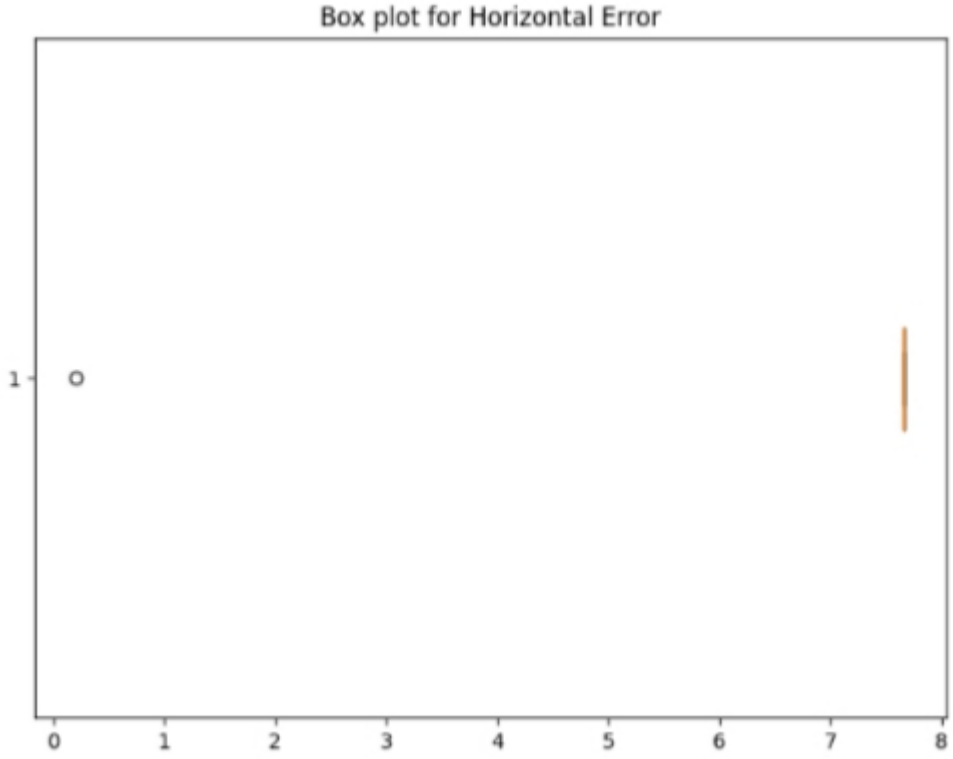
OUTPUT:

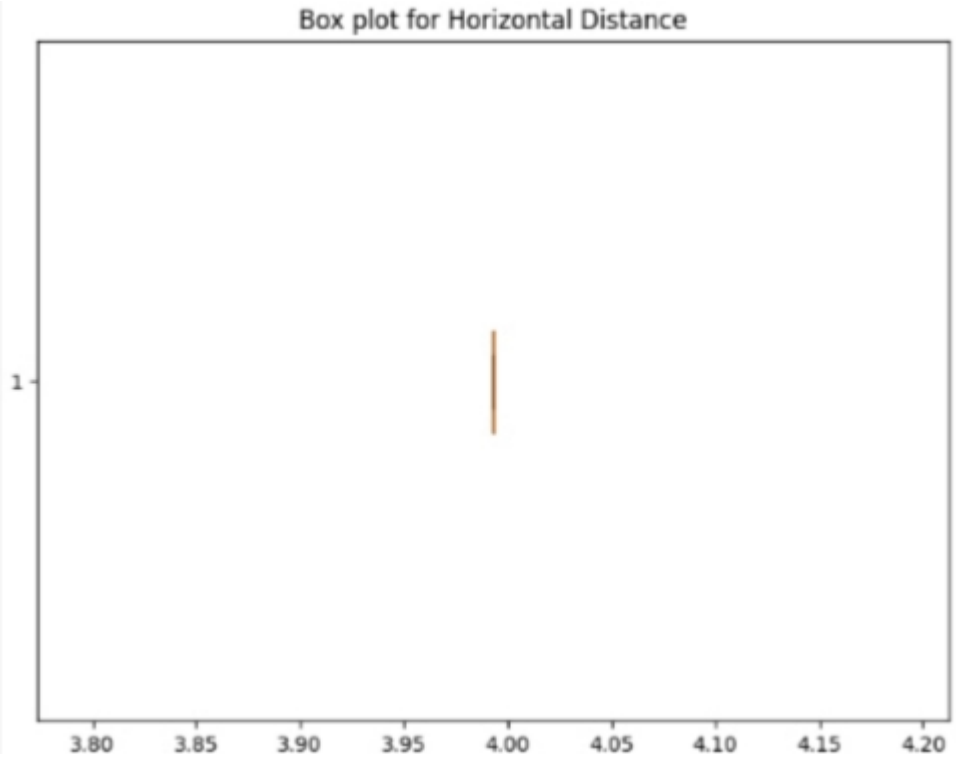


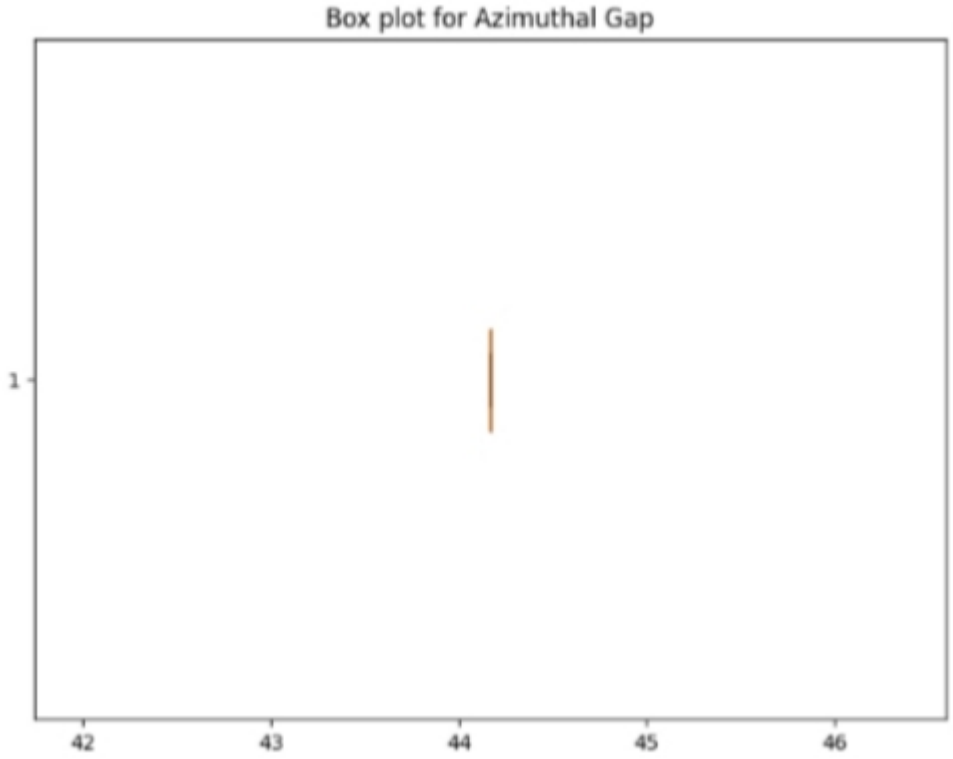


Box plot for Depth




Box plot for Horizontal Error
1
0
1
2
3
4
5
6
7
8


Box plot for Horizontal Distance
1
3.80
3.85
3.90
3.95
4.00
4.05
4.10
4.15
4.20



DATA VISUALIZATION ON A WORLD MAP:

- The code uses the Folium library to create an interactive world map.

- A Folium map is created with an initial center at latitude 0 and longitude 0, and a zoom level of 2.

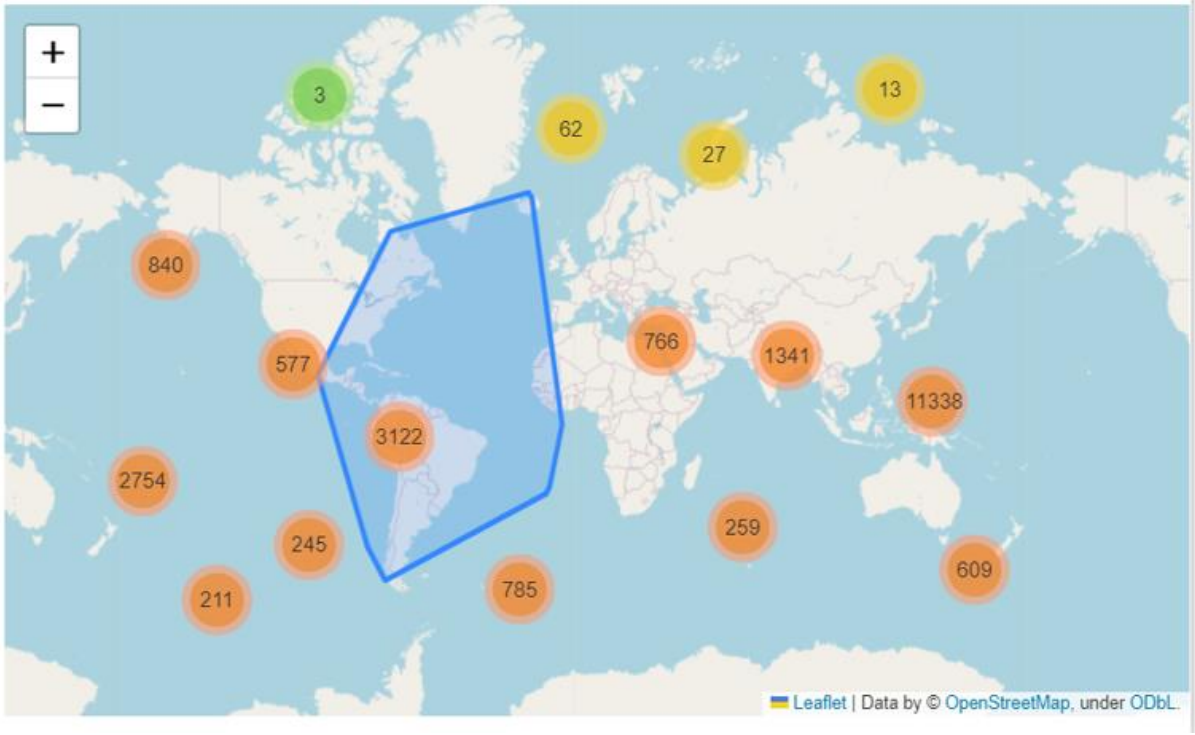
- A marker cluster is created using the MarkerCluster plugin, which groups nearby earthquake markers on the map for better performance.

- 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 folium
from folium.plugins import MarkerCluster
from IPython.display import display
```

```
#DATA VISUALIZATION ON A WORLD MAP
# 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)
```

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



SPLIT DATA INTO TRAIN AND TEST SETS:

Feature Selection and Target Variable:

- The code selects the features (X) and the target variable (y) for earthquake prediction. In this case, 'Magnitude' is

chosen as the target variable, and various seismic attributes are selected as features. These features include latitude, longitude, depth, and other relevant seismic parameters.

Standardization of Features:

- The code standardizes the selected features using the StandardScaler from Scikit-Learn. Standardization ensures that the features have a mean of 0 and a standard deviation of 1. This step is crucial for many machine learning algorithms.

Splitting the Dataset:

- The dataset is split into training and testing sets using the 'train_test_split' function from Scikit-Learn. The training set contains 80% of the data, and the testing set contains 20%. A random seed is set for reproducibility.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# Select features (X) and the target (y)
X = data[['Latitude', 'Longitude', 'Depth', 'Depth Error', 'Depth Seismic Stations',
          'Magnitude Error', 'Magnitude Seismic Stations',
          'Azimuthal Gap', 'Horizontal Distance', 'Horizontal Error', 'Root Mean Square']]
  # Replace 'Target_Column' with your target variable.Here target variable is Magnitude.
y = data['Magnitude']
```

```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Split the dataset 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)
```

MODEL TRAINING AND EVALUATION:

Random Forest Model:

- The code employs a Random Forest Regressor model for earthquake magnitude prediction.
- Missing values in the features are handled again by filling them with the mean of their respective columns.
- A Random Forest Regressor model is created with 100 decision trees for prediction. The 'random_state' parameter is set for reproducibility.

Model Training and Prediction:

- The model is trained using the training data (X_train, y_train).
- Predictions are made on the test set (X_test) to assess the model's performance.

Model Evaluation:

- Two evaluation metrics are calculated to assess the model's performance:
  - Mean Squared Error (MSE): It measures the average squared difference between the actual and predicted Magnitude values. A lower MSE indicates better predictive performance.
  - R-squared (R2) Score: It represents the proportion of variance in the target variable (Magnitude) that is predictable from the features. A higher R2 score indicates a better fit.
- The code prints these evaluation metrics, providing insights into how well the model predicts earthquake magnitudes.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
#RANDOM FOREST:
# Handle missing values by filling with the mean of the column
X.fillna(X.mean(), inplace=True)

# Create a Random Forest Regressor model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```

Output:

```
Mean Squared Error: 0.16562220379054265
R-squared (R2) Score: 0.1020595477784716
```

- The code provides the following evaluation metrics:
  - Mean Squared Error (MSE): 0.1656
  - R-squared (R2) Score: 0.1021
- These metrics indicate the performance of the Random Forest Regressor model in predicting earthquake magnitudes. The relatively high MSE and low R2 score suggest that the model may not be a strong predictor for this particular dataset.

Overall, this code covers data preprocessing, visualization, model building, and evaluation for earthquake magnitude prediction using a Random Forest Regressor. The model's performance is presented through evaluation metrics.