Earthquake Prediction Model using Python.

PHASE 4 Document Submission.

Prepared by,

PRADEEP KUMAR.T,

510521205028,

Bharathidasan Engineering College.

EARTHQUAKE PREDICTION

Feature Engineering:

Feature engineering is the process of selecting, transforming, or creating new features (variables or input data) from the existing raw data to improve the performance of machine learning models. It is a crucial step in the data preprocessing pipeline and plays a significant role in the success of a machine learning project. Feature engineering involves a combination of domain knowledge, creativity, and data analysis to extract relevant information and patterns from the data.

1. Geographic Features:

Latitude and Longitude:

These are essential geographical coordinates. You can use them directly or compute distances from known seismic hotspots or fault lines.

Depth:

Earthquake depth is a significant factor, and you may want to consider it as a feature.

2. Temporal Features:

Time of Day:

Split time into different segments (e.g., morning, afternoon, evening) to capture diurnal patterns.

Day of the Week: Some regions might have varying seismic activity based on the day.

Month and Season: Earthquake occurrences can be seasonal; adding these features can help.

3. Historical Features:

Rolling Statistics:

Calculate statistics (mean, standard deviation, max, min) over a specific time window to capture trends in historical earthquake activity.

Lag Features:

Include earthquake occurrences from previous time periods as lag features.

4. Geological Features:

Distance to Fault Lines:

Compute the distance between each data point and the nearest known fault line.

Tectonic Plate Boundaries:

Consider whether data points are located near tectonic plate boundaries.

5. Categorical Features:

Location Clusters: Group locations into clusters based on proximity to each other or similar geological conditions.

Hotspots: Use categorical variables to denote regions with higher seismic activity.

6. Meteorological and Environmental Data:

Incorporate weather or environmental data if they are relevant to seismic activity in your region.

7. Magnitude Features:

Calculate the average magnitude of earthquakes in the vicinity of each data point.

8. Spatial Features:

Spatial Autocorrelation: Examine if earthquake occurrences at nearby locations affect each other.

9. Density Features:

Density of Earthquakes: Calculate the number of earthquakes in a specified radius around each data point.

10. Advanced Techniques:

- Use Principal Component Analysis (PCA) to reduce the dimensionality of your data if you have a large number of features.

- Time-series features, such as autoregressive components or exponential smoothing, may be relevant depending on your data.

MODEL TRAINING:

Model training is a critical step in building an earthquake prediction model using Python. You'll need to choose an appropriate machine learning algorithm, preprocess your data, split it into training and testing sets, and then train the model. Here's a step-by-step guide:

1. Choose a Machine Learning Algorithm:

Select a machine learning algorithm suitable for your earthquake prediction problem. Some common choices include decision trees, random forests, support vector machines (SVMs), and neural networks. The choice of algorithm depends on the complexity of your data and the performance you aim to achieve.

2. Data Preprocessing:

Before training your model, you should preprocess your data to make it suitable for machine learning. This may include the following steps:

Data Cleaning:

Remove duplicates, handle missing values, and address outliers.

Feature Engineering:

Create relevant features as discussed in the previous response.

Feature Scaling:

Normalize or standardize your features to bring them to a common scale.

Data Splitting:

Split your dataset into training and testing sets to assess model performance.

3. Train-Test Split:

Divide your dataset into a training set and a testing set. Typically, a common split is 70-80% for training and 20-30% for testing. This ensures that you can evaluate your model's performance on unseen data.

4. Model Initialization:

Create an instance of your chosen machine learning model.

5. Model Training:

Train the model using your training data.

6. Model Hyperparameter Tuning:

To optimize the performance of your model, you can perform hyperparameter tuning. You can use techniques like grid search or random search to find the best hyperparameters for your model.

7. Model Evaluation:

Evaluate your model's performance on the testing data using appropriate evaluation metrics. For earthquake prediction, metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) are commonly used.

8. Visualize Results:

Visualize the model's predictions against the actual earthquake occurrences to understand how well it's performing.

9. Iterate and Refine:

If your model's performance is not satisfactory, consider iterating on the previous steps, experimenting with different algorithms, hyperparameters, or feature engineering techniques.

10. Deployment:

Once you are satisfied with your model's performance, you can deploy it for real-time prediction if necessary.

EVALUATING:

Evaluating an earthquake prediction model is a crucial step to determine its performance and effectiveness. Various evaluation metrics and techniques can be used to assess the model's performance. Here's how you can evaluate an earthquake prediction model using Python:

1. Import Libraries:

First, import the necessary Python libraries, including scikit-learn and any other libraries that you need for visualization:

2. Model Prediction:

Make predictions using your trained model on the testing data:

3. Evaluation Metrics:

a. Mean Absolute Error (MAE):

MAE represents the average absolute difference between the predicted and actual values. Lower MAE indicates better model performance.

b. Mean Squared Error (MSE):

MSE measures the average of the squared differences between predicted and actual values. Lower MSE is better.

RMSE is the square root of MSE and provides the error in the same units as the target variable.

d. R-squared (R2) Score:

R2 measures the proportion of the variance in the target variable that is predictable by the model. A higher R2 score indicates a better model fit.

4.Visualize Results:

It's often helpful to visualize the model's predictions against the actual earthquake occurrences. This can be done using scatter plots or time-series plots, depending on your data. For example, you can create a scatter plot as follows:

5. Model Interpretability:

Depending on the complexity of your model, consider using techniques like feature importance analysis to understand which features are most influential in making predictions.

6. Cross-Validation:

Perform cross-validation to assess the model's generalization performance. Cross-validation provides a more robust estimate of how well your model is likely to perform on unseen data.

7. Domain Expert Consultation:

Always consult with domain experts to ensure that the model's performance aligns with the practical requirements and expectations in the field of earthquake prediction.

8.Iterate and Refine:

If the model's performance is not satisfactory, consider fine-tuning hyperparameters, trying different algorithms, or further improving feature engineering.

PROGRAM:

Data Visulaization:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
try:
    # Load earthquake data into a Pandas DataFrame
    earthquake_data = pd.read_csv('G:/database.csv')
    # Data Inspection
    print("Data Inspection:")
    print(earthquake_data.head())
    print(earthquake_data.info())
    scaler = MinMaxScaler()
    numerical_features = ['Latitude', 'Longitude', 'Magnitude', 'Depth']
    earthquake_data[numerical_features] =
scaler.fit_transform(earthquake_data[numerical_features])
    # Data Cleaning
    # Handle missing data if needed
    earthquake_data.dropna(inplace=True)
    # Data Visualization (Optional)
```

```
    plt.scatter(earthquake_data['Longitude'],
earthquake_data['Latitude'], c=earthquake_data['Magnitude'],
cmap='viridis')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.title('Earthquake Magnitude by Location')
    plt.colorbar(label='Magnitude')
    plt.show
    # Split the data into training and testing sets
    features = earthquake_data[['Latitude', 'Longitude', 'Depth']]
    target = earthquake_data['Magnitude']
    X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)
    # Now you can build and train a machine learning model for
aftershock forecasting using X_train and y_train.
except FileNotFoundError:
    print("File not found. Please check the file path.")
except Exception as e:
    print(f"An error occurred: {str(e)}")
```

OUTPUT:

```
        Data Inspection:

        Date     Time  Latitude  Longitude       Type  Depth  Depth
Error  \
0  01/02/1965  13:44:18    19.246    145.616  Earthquake  131.6
NaN
1  01/04/1965  11:29:49     1.863    127.352  Earthquake   80.0
NaN
```

2 01/05/1965 18:05:58 -20.579 -173.972 Earthquake 20.0 NaN

3 01/08/1965 18:49:43 -59.076 -23.557 Earthquake 15.0 NaN

4 01/09/1965 13:32:50 11.938 126.427 Earthquake 15.0 NaN

	Depth	Seismic Stations	Magnitude	Magnitude Type	...	\
0		NaN	6.0	MW	...	
1		NaN	5.8	MW	...	
2		NaN	6.2	MW	...	
3		NaN	5.8	MW	...	
4		NaN	5.8	MW	...	

	Magnitude	Seismic Stations	Azimuthal Gap	Horizontal Distance	\
0		NaN	NaN	NaN	
1		NaN	NaN	NaN	
2		NaN	NaN	NaN	
3		NaN	NaN	NaN	
4		NaN	NaN	NaN	

	Horizontal Error	Root Mean Square	ID	Source	Location Source	\
0	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	
1	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	

2 NaN NaN ISCGEM860762 ISCGEM ISCGEM

3 NaN NaN ISCGEM860856 ISCGEM ISCGEM

4 NaN NaN ISCGEM860890 ISCGEM ISCGEM

	Magnitude	Source	Status
0		ISCGEM	Automatic
1		ISCGEM	Automatic
2		ISCGEM	Automatic
3		ISCGEM	Automatic
4		ISCGEM	Automatic

[5 rows x 21 columns]

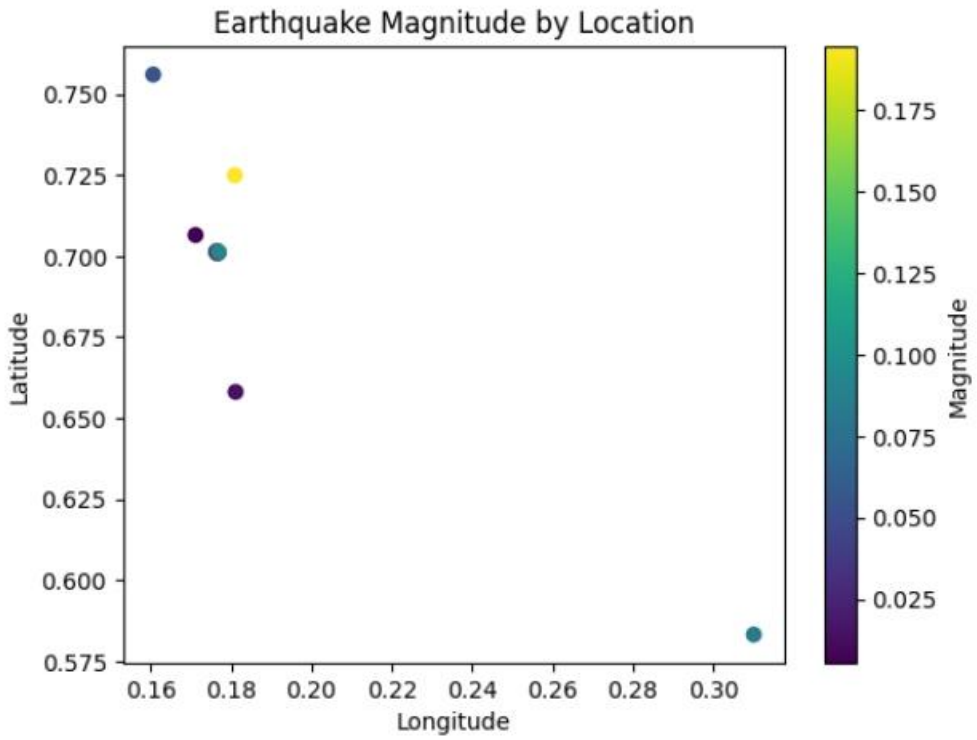
<class 'pandas.core.frame.DataFrame'>

RangeIndex: 23412 entries, 0 to 23411

Data columns (total 21 columns):

#	Column	Non-Null Count	Dtype
0	Date	23412 non-null	object
1	Time	23412 non-null	object
2	Latitude	23412 non-null	float64
3	Longitude	23412 non-null	float64
4	Type	23412 non-null	object
5	Depth	23412 non-null	float64

```
 6   Depth Error                4461 non-null   float64
 7   Depth Seismic Stations     7097 non-null   float64
 8   Magnitude                  23412 non-null  float64
 9   Magnitude Type             23409 non-null  object
 10  Magnitude Error            327 non-null    float64
 11  Magnitude Seismic Stations 2564 non-null   float64
 12  Azimuthal Gap              7299 non-null   float64
 13  Horizontal Distance        1604 non-null   float64
 14  Horizontal Error           1156 non-null   float64
 15  Root Mean Square           17352 non-null  float64
 16  ID                         23412 non-null  object
 17  Source                     23412 non-null  object
 18  Location Source            23412 non-null  object
 19  Magnitude Source           23412 non-null  object
 20  Status                     23412 non-null  object
dtypes: float64(12), object(9)
memory usage: 3.8+ MB
Non
```



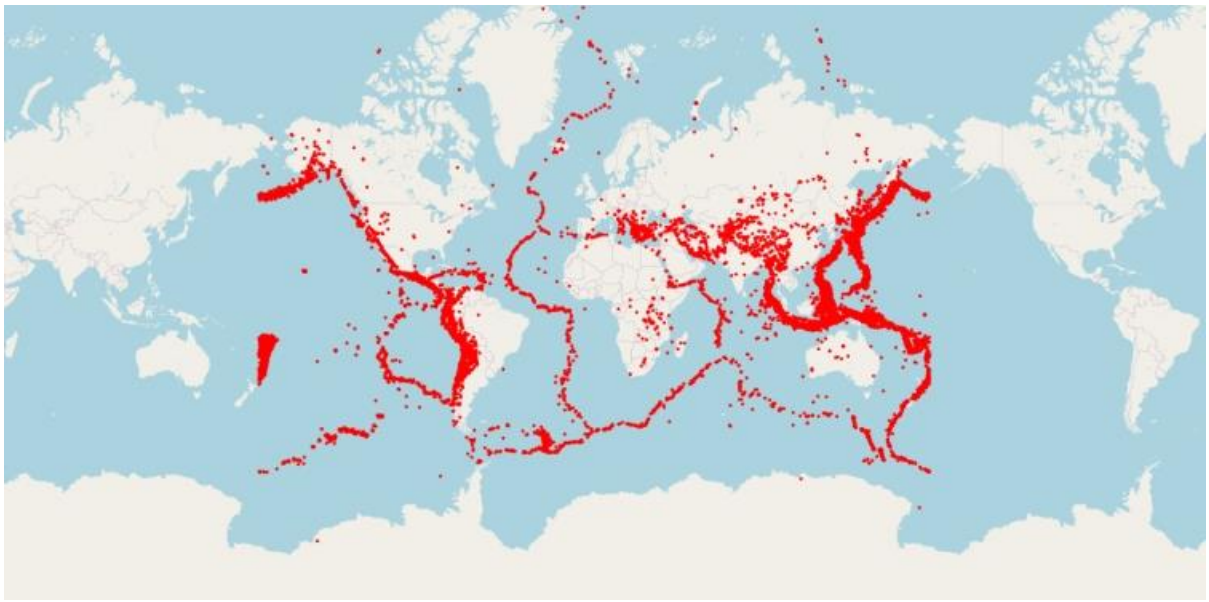
Create the Map:.

Python code:

```
import folium
# Create a base map
m = folium.Map(location=[10.124357, 78.229340], zoom_start=2)
# Add markers for earthquake locations with frequencies
for _, row in earthquake_counts.iterrows():
    folium.CircleMarker(
        location=[row['Latitude'], row['Longitude']],
        radius=row['Frequency'] / 500,  # Adjust the size based on frequency
        color='red',
        fill=True,
        fill_color='red',
```

```
        fill_opacity=0.6,
        popup=f"Frequency: {row['Frequency']}",
    ).add_to(m)
# Display the map
m.save('g:/earthquake_frequency_map.html')
# Save the map to an HTML file
```

Output:



Model Training and Evaluation:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression  # You can replace this with your chosen model
from sklearn.metrics import mean_squared_error, r2_score
# Load your earthquake dataset
```

```
df = pd.read_csv('g:/database.csv')  # Replace 'earthquake_data.csv' with your dataset file path

# Define your features (X) and target variable (y)

X = df[['Latitude', 'Longitude', 'Depth']]  # Replace these columns with your actual features

y = df['Magnitude']  # Replace with your target variable

# Split the dataset into a training set (80%) and a test set (20%)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling (optional but recommended)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Initialize and train your machine learning model (e.g., Linear Regression)

model = LinearRegression()  # You can replace this with your chosen model

model.fit(X_train_scaled, y_train)

# Make predictions on the test set

y_pred = model.predict(X_test_scaled)

# Evaluate the model's performance

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): {mse}")

print(f"R-squared (R2) Score: {r2}")
```

Output:

```
Mean Squared Error (MSE): 0.18462041284893194
```

R-squared (R2) Score: -0.00094149944140209 39

EVALUATING:

Python code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
import matplotlib.pyplot as plt
# Load your preprocessed dataset
data = pd.read_csv('G:\database.csv')
# Define your features and target variable
features = ['Latitude', 'Longitude']
target = 'Magnitude'  # Target variable indicating earthquake
magnitude
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)
# Train a regression model (Random Forest Regressor as an example)
model = RandomForestRegressor(n_estimators=100,
random_state=42)
model.fit(X_train, y_train)
# Make predictions on the test set
```

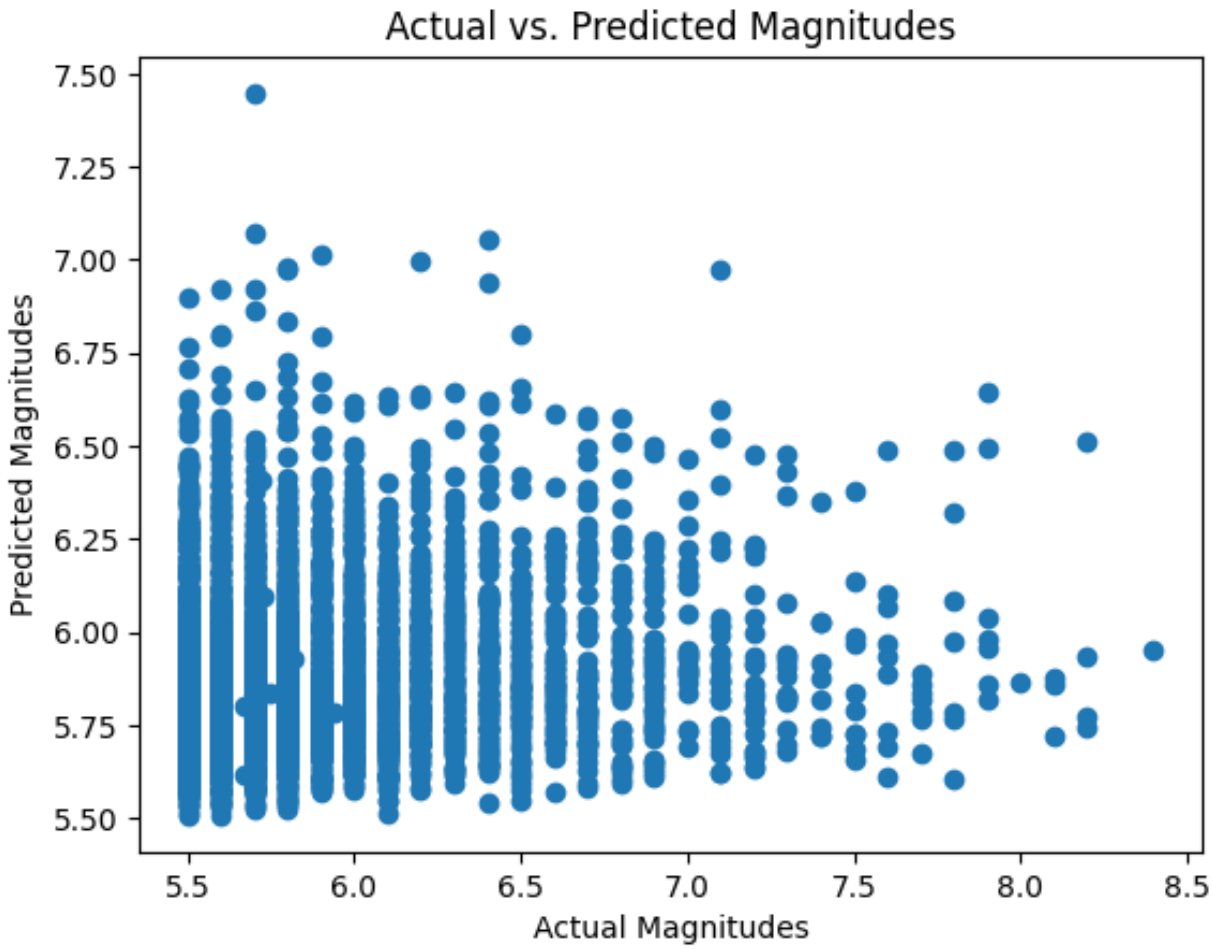
```
y_pred = model.predict(X_test)
# Evaluate the model using regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2) Score: {r2}")

# Visualize the predicted vs. actual magnitudes
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Magnitudes")
plt.ylabel("Predicted Magnitudes")
plt.title("Actual vs. Predicted Magnitudes")
plt.show()
```

OUTPUT:

```
Mean Squared Error (MSE): 0.21170610148313368
Mean Absolute Error (MAE): 0.33582664919843036
R-squared (R2) Score: -0.14778977789858927
```


Actual vs. Predicted Magnitudes
Predicted Magnitudes
Actual Magnitudes
7.50
7.25
7.00
6.75
6.50
6.25
6.00
5.75
5.50
5.5
6.0
6.5
7.0
7.5
8.0
8.5