

Mini Project

Earthquake Prediction by Using Machine Learning

ABSTRACT: Predicting earthquakes is a critical task for minimizing human and infrastructural losses. In this project, we employ a machine learning approach to predict the magnitude of earthquakes based on various seismic features. Using a real-world dataset, we analyze key indicators, train predictive models, and evaluate performance. This documentation outlines the methodology, data insights, and technical implementation used in this research.

Keywords: Earthquake Prediction, Machine Learning, Regressors, classifiers

Dataset Information

The dataset used in this project contains seismic records from various events. It comprises 2,719 instances and 4 key features, all numeric:

Column Name	Description	Data Type
Latitude	Latitude of the earthquake's epicenter	Float
Longitude	Longitude of the earthquake's epicenter	Float
Depth	Depth at which the earthquake occurred (in km)	Float
Magnitude	Measured magnitude on the Richter scale (target)	Float

Key Characteristics

- **Total Records:** 2,719
- **Missing Values:** None
- **Target Variable:** Magnitude
- **Input Features:** Latitude, Longitude, Depth
- **Geographic Scope:** Global, with locations primarily from India and surrounding regions.

Preprocessing Performed

- Checked for missing values and confirmed data completeness.
- Normalized features to ensure model stability.
- Identified and handled outliers if any (details in later sections).

Introduction

Earthquakes are natural phenomena that occur without warning, often resulting in catastrophic consequences for people, property, and the environment. While the science of seismology has made significant progress in identifying fault lines and understanding tectonic behavior, predicting the magnitude of an earthquake remains an unsolved challenge.

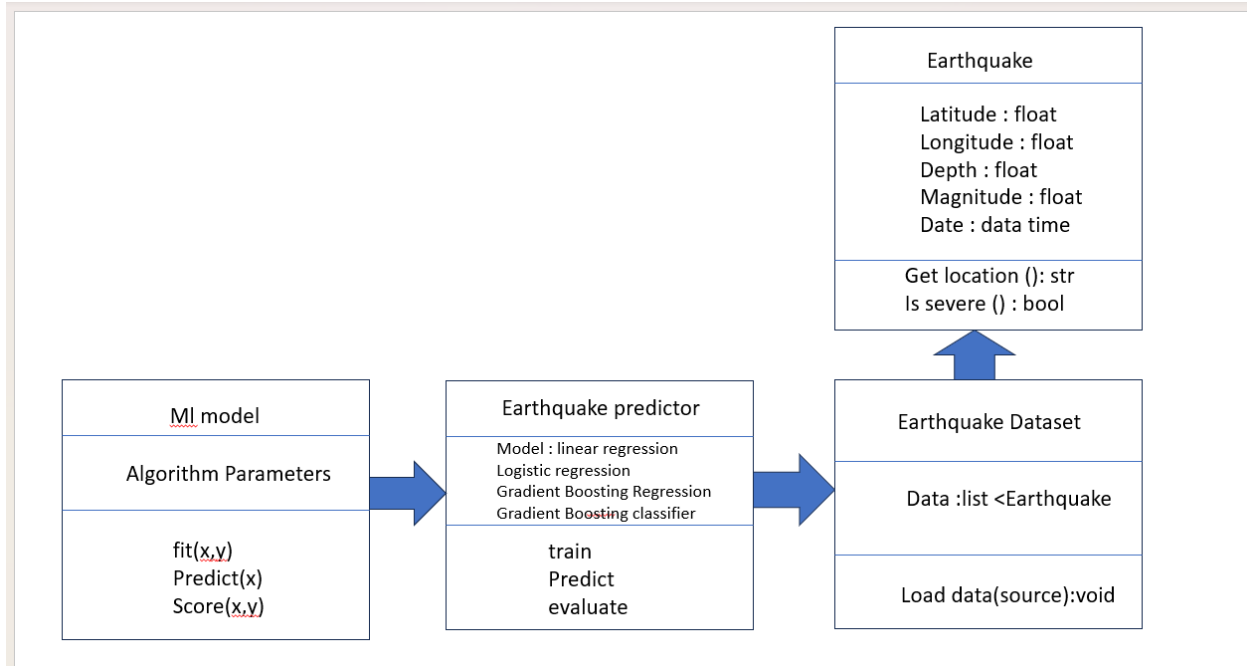
In this project, we explore how machine learning can be harnessed as a predictive tool to analyze past seismic data and estimate earthquake magnitudes based on features like geographic location and depth. Unlike traditional rule-based models, machine learning allows systems to learn from patterns hidden in the data, improving prediction accuracy over time.

The primary goal is not to predict if an earthquake will occur, but to anticipate its intensity — a crucial factor for emergency response planning, structural design, and public safety protocols. By training algorithms on historical records, we aim to build a model that learns the relationship between seismic characteristics and earthquake magnitudes.

This work represents a step toward using data science not just for reactive disaster management but as a proactive strategy for disaster risk reduction.

Class Diagram

This class diagram represents a machine learning pipeline for an Earthquake Prediction System, showing the interaction between various components and classes used in the system. Here's a theoretical explanation of each part of the diagram:



Earthquake Class

This class models a real-world earthquake event and contains the following **attributes**:

- Latitude: float – Geographical latitude of the earthquake.
- Longitude: float – Geographical longitude of the earthquake.
- Depth: float – Depth at which the earthquake occurred.
- Magnitude: float – Measured magnitude (e.g., on the Richter scale).
- Date: datetime – Date and time of the earthquake event.

Methods:

- get_location(): str – Returns the location as a string (likely combining latitude and longitude).

- `is_severe()`: bool – Determines whether the earthquake is severe based on its magnitude or other parameters.

EarthquakeDataset Class

This class handles the **collection and loading** of earthquake data.

Attributes:

- `Data`: List<Earthquake> – A list of Earthquake objects representing multiple events.

Methods:

- `load_data(source)`: void – Loads data from a specified source (e.g., a CSV file, API, or database).

MLModel Class

This is a generic machine learning model class that holds algorithm configurations and provides methods for training and evaluation.

Attributes:

- `Algorithm Parameters` – These could be hyperparameters or settings for various ML algorithms.

Methods:

- `fit(x, y)` – Trains the model on input features `x` and target labels `y`.
- `predict(x)` – Predicts outcomes based on the input `x`.
- `score(x, y)` – Evaluates the model performance on test data

EarthquakePredictor Class

This acts as a **controller** or **interface** between data and the ML model.

Attributes:

- `Model`: The class supports multiple models:

Linear Regression

Logistic Regression

Gradient Boosting Regression

Gradient Boosting Classifier

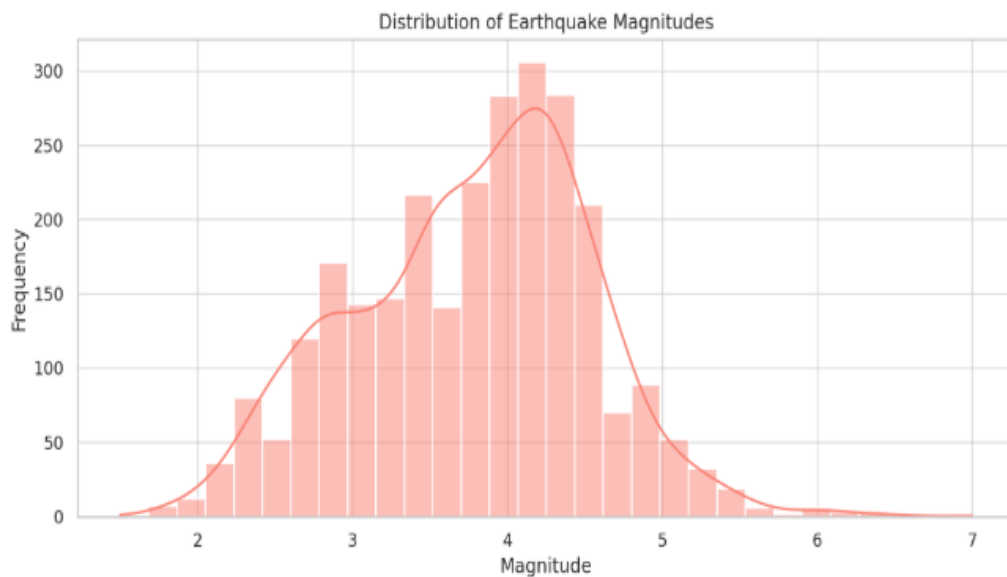
Methods:

- `train()` – Trains the model using the dataset.
- `predict()` – Uses the trained model to make predictions.
- `evaluate()` – Evaluates the model's accuracy, precision, recall, or other metrics.

Data Visualization

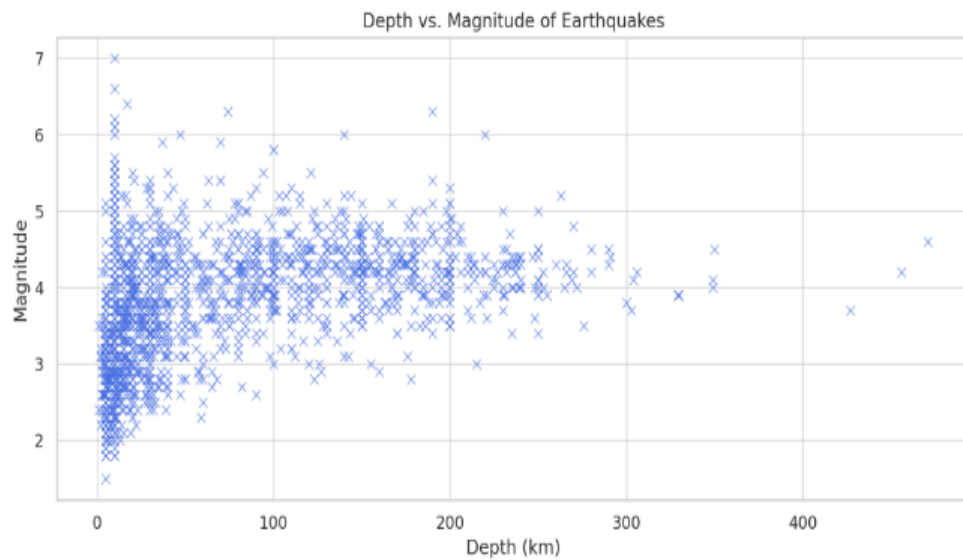
1. Distribution of Earthquake Magnitudes

- Most earthquakes in the dataset fall within the 2.0 to 4.5 range.
- The distribution is slightly right-skewed, indicating occasional occurrences of higher magnitudes.



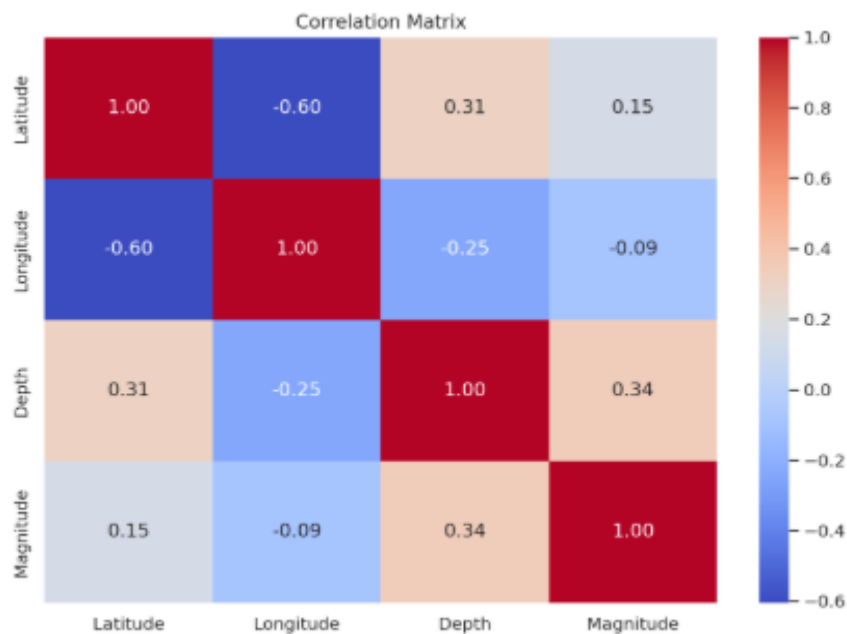
2. Depth vs Magnitude (Scatter Plot)

- There's no strong linear relationship, but many lower-magnitude quakes appear at shallow depths.
- A few deeper earthquakes have moderate magnitudes.



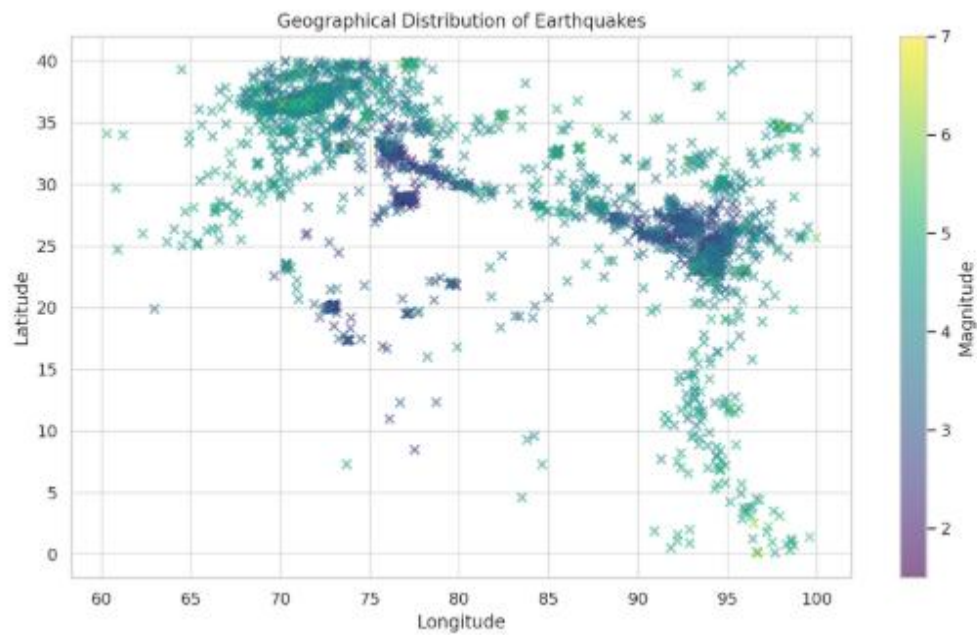
3. Correlation Heatmap

- Depth and Magnitude show a weak positive correlation.
- Latitude and Longitude show minimal correlation with magnitude, which is expected as location doesn't directly affect the energy release.



4. Geographical Distribution of Earthquakes

- Clusters of seismic activity are visible, especially in certain longitudes and latitudes (India and neighboring seismic zones).
- The color intensity indicates the magnitude — darker points represent stronger quakes.



Implementation

Here we use four algorithms

→Regression models

1.LinearRegression

2.GradientBosstingRegressor

→classification models

3.LogisticRegression

4.GradientBoostingclassifier

Source code:

#import libraries

```
import pandas as pd
```

```
import numpy as np
```

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

```
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
```

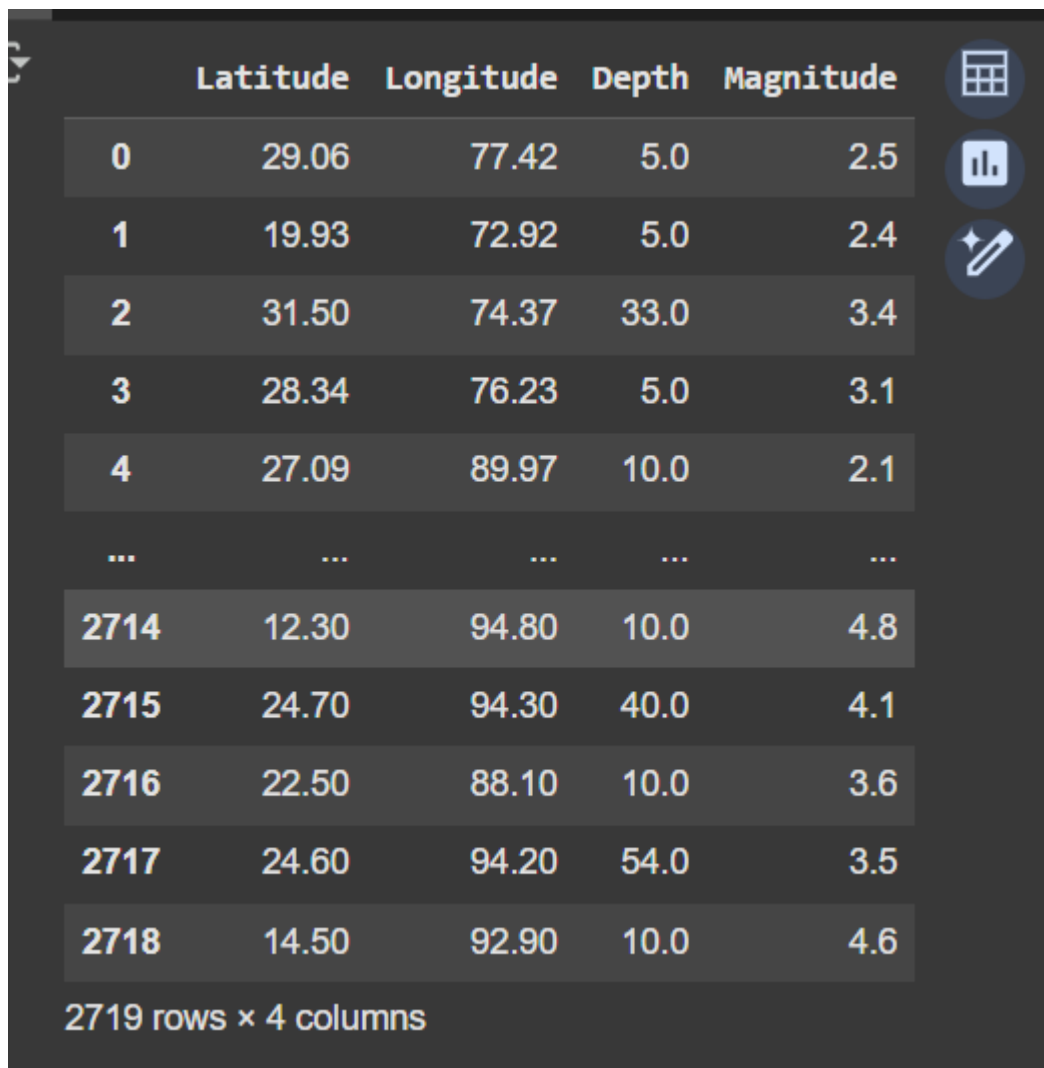
```
from sklearn.preprocessing import StandardScaler
```

#dataset

```
df=pd.read_csv("/content/dataset.csv")
```

```
df
```

output:



	Latitude	Longitude	Depth	Magnitude
0	29.06	77.42	5.0	2.5
1	19.93	72.92	5.0	2.4
2	31.50	74.37	33.0	3.4
3	28.34	76.23	5.0	3.1
4	27.09	89.97	10.0	2.1
...
2714	12.30	94.80	10.0	4.8
2715	24.70	94.30	40.0	4.1
2716	22.50	88.10	10.0	3.6
2717	24.60	94.20	54.0	3.5
2718	14.50	92.90	10.0	4.6

2719 rows × 4 columns

df.info()

output:

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2719 entries, 0 to 2718
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Latitude    2719 non-null   float64
1   Longitude    2719 non-null   float64
2   Depth        2719 non-null   float64
3   Magnitude    2719 non-null   float64
dtypes: float64(4)
memory usage: 85.1 KB

```

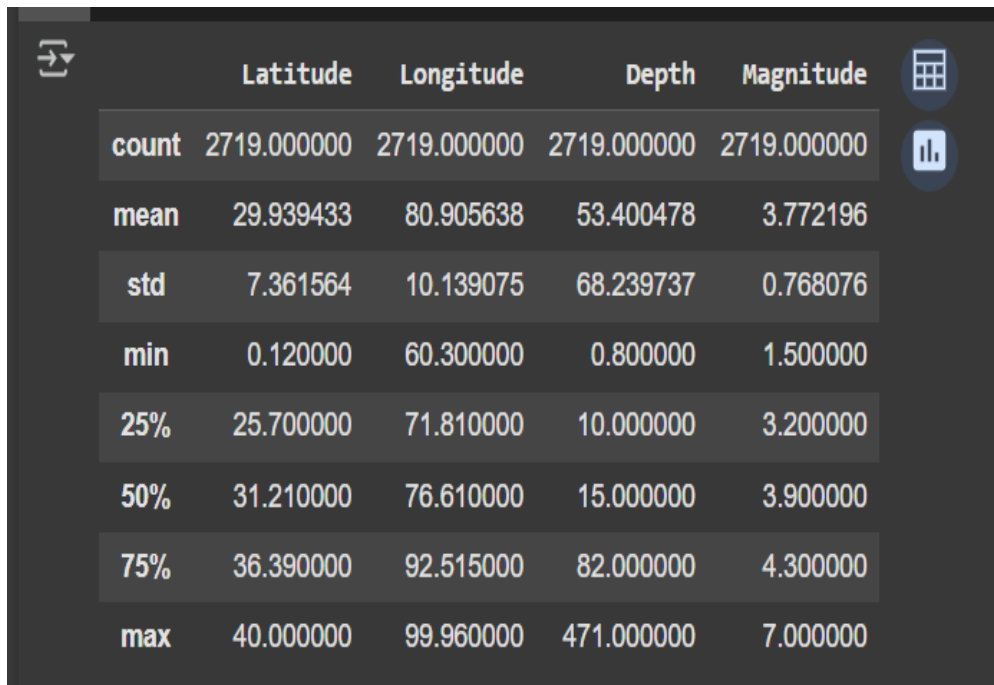
df.head(10)

output:

	Latitude	Longitude	Depth	Magnitude
0	29.06	77.42	5.0	2.5
1	19.93	72.92	5.0	2.4
2	31.50	74.37	33.0	3.4
3	28.34	76.23	5.0	3.1
4	27.09	89.97	10.0	2.1
5	38.52	73.27	115.0	5.2
6	27.90	94.20	10.0	3.0
7	26.60	92.51	28.0	3.1
8	22.88	95.95	10.0	5.5
9	37.96	72.39	160.0	4.3

df.describe()

output:

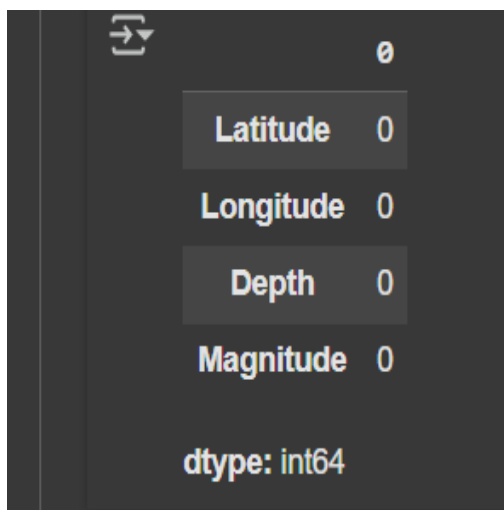


The image shows a Jupyter Notebook cell with the command `df.describe()` and its output. The output is a table with 5 columns: Latitude, Longitude, Depth, and Magnitude. The rows represent statistical measures: count, mean, std, min, 25%, 50%, 75%, and max. There are also icons for a calculator and a bar chart on the right side of the table.

	Latitude	Longitude	Depth	Magnitude
count	2719.000000	2719.000000	2719.000000	2719.000000
mean	29.939433	80.905638	53.400478	3.772196
std	7.361564	10.139075	68.239737	0.768076
min	0.120000	60.300000	0.800000	1.500000
25%	25.700000	71.810000	10.000000	3.200000
50%	31.210000	76.610000	15.000000	3.900000
75%	36.390000	92.515000	82.000000	4.300000
max	40.000000	99.960000	471.000000	7.000000

df.isnull().sum()

output:



The image shows a Jupyter Notebook cell with the command `df.isnull().sum()` and its output. The output is a table with 5 columns: Latitude, Longitude, Depth, and Magnitude. The rows represent the count of null values for each column. The dtype is int64.

	0
Latitude	0
Longitude	0
Depth	0
Magnitude	0
dtype:	int64

```
x = df[['Latitude', 'Longitude', 'Depth']]
```

```
y = df['Magnitude']
```

#split method

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=True)
```

```
print("x_train:\n",x_train)
```

output:

```
x_train:
```

	Latitude	Longitude	Depth
859	22.110	94.200	87.0
486	36.770	71.710	13.0
2229	26.200	92.900	27.0
1016	23.270	94.600	10.0
857	26.703	95.313	10.0
...
960	20.230	72.950	5.0
905	36.150	70.820	10.0
1096	36.550	70.640	143.0
235	28.720	77.030	5.0
1061	29.640	88.300	10.0

[2175 rows x 3 columns]

```
print("x_test:\n",x_test)
```

output:

```
x_test:
```

	Latitude	Longitude	Depth
1442	32.82	86.68	200.0
1152	31.21	78.54	5.0
1506	24.65	94.92	15.0
1334	36.67	70.29	212.0
2697	32.40	76.30	5.0
...
196	34.90	97.83	10.0
1397	36.52	71.08	30.0
533	26.25	95.23	80.0
1349	37.29	71.92	160.0
702	38.38	71.84	150.0

```
print("y_train:\n",y_train)
```

output:

```
y_train:
  859      5.2
  486      4.4
 2229      2.5
 1016      3.5
  857      2.5
      ...
  960      2.4
  905      4.3
 1096      4.1
  235      1.9
 1061      4.0
Name: Magnitude, Length: 2175, dtype: float64
```

```
print("y_test:\n",y_test)
```

output:

```
y_test:
 1442      4.6
 1152      2.9
 1506      3.7
 1334      3.6
 2697      2.7
      ...
  196      4.5
 1397      3.3
  533      3.2
 1349      4.3
  702      3.7
Name: Magnitude, Length: 544, dtype: float64
```

#use fit transform training data

```
scaler = StandardScaler()
```

```
x_train_scaled = scaler.fit_transform(x_train)
```

```
x_test_scaled=scaler.transform(x_test)
```

```
print("x_train_scaled:\n",x_train_scaled)
```


output:

```
x_train_scaled:
[[-0.4954332  0.90083728 -0.55403162]
 [-0.42043471  1.22868254 -0.33536588]
 [-0.22952947  0.70868627  1.37022687]
 ...
 [-0.02635176  1.4733284  -0.6269202 ]
 [-0.83769902  0.73840035 -0.6269202 ]
 [-0.13271325 -0.387763   -0.56860933]]
```

```
print("x_test_scaled:\n",x_test_scaled)
```

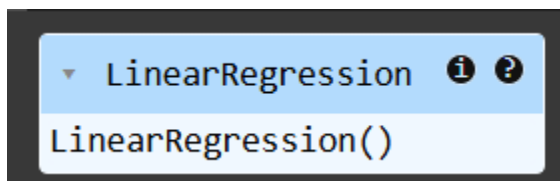
output:

```
x_test_scaled:
[[-0.8499715   1.20788269 -0.33536588]
 [ 0.58045418 -0.42341989 -0.69980878]
 [-0.20907534  1.42578589 -0.6269202 ]
 ...
 [ 1.11635245 -0.64033261 -0.6269202 ]
 [-0.18180316 -0.38677253 -0.68523106]
 [ 1.27589469 -0.34814423  0.7579628 ]]
```

Linear Regression

```
lr = LinearRegression()
```

model:



```
lr_pred = lr.predict(x_test)
```

```
print("lr_pred:\n",lr_pred)
```

output:


```
print("gbr_pred:\n",gbr_pred)
```

output:

```
lr_pred:
[4.37634437 3.60711018 3.64751607 4.3988897  3.60884627 4.28161252
 3.50802432 3.53473783 3.65025315 3.63844043 4.4663886  3.64402432
 3.66481649 4.35247988 4.15206686 3.52652242 4.18463956 3.66081508
 3.64783795 3.78978925 3.71249009 3.64019265 3.54219554 3.61355142
 3.65009312 3.76061476 3.62261721 4.08317915 3.6058949  3.62940579
 4.18727912 4.33298085 3.61616155 3.56353272 4.12834946 4.49971416
 3.70045031 3.60280374 3.59721334 3.64409564 3.79830081 4.19108702
 3.63556872 3.50817151 3.60325119 3.6772121  4.25157726 4.18428468
 3.60355358 3.64518984 3.65581338 3.64759159 3.69786227 3.67403621
 3.63298485 3.65085831 3.78769528 3.59214148 3.87004172 3.6311946
 3.62330896 3.68615084 4.35895482 3.60282764 4.28201249 3.52617742]
```

#evaluation metrics

```
mse = mean_squared_error(y_test, gbr_pred)
```

```
r2 = r2_score(y_test, gbr_pred)
```

```
print(f"Mean Squared Error: {mse:.4f}")
```

```
print(f"R2 Score: {r2:.4f}")
```

output:

```
Mean Squared Error: 0.2914
R2 Score: 0.5046
```

#plotting the data, visualizing the data

```
plt.figure(figsize=(10, 5))
```

```
plt.plot(y_test.values[:50], label='Actual', color='blue')
```

```
plt.plot(gbr_pred[:50], label='Gradient Boosting Predicted', color='green')
```

```
plt.title("Gradient Boosting Regressor - Earthquake Magnitude")
```

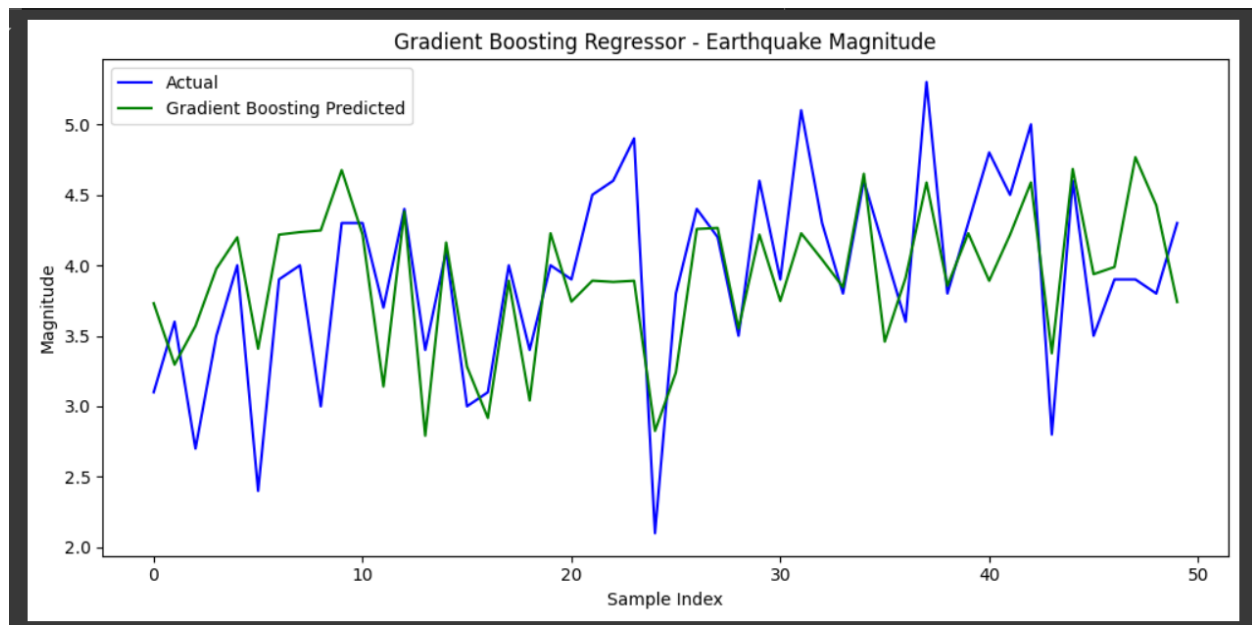
```
plt.xlabel("Sample Index")
```

```
plt.ylabel("Magnitude")
```

```
plt.legend()
```

```
plt.tight_layout()
```

```
plt.show()
```



Classification:

#import libraries

```
import pandas as pd
```

```
import numpy as np
```

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

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import seaborn as sns

#DataPreprocessing

df=pd.read_csv('/content/dataset.csv')

df

df.info()

df.describe()

df.shape

df.isnull().sum()

def magnitude_to_label(mag):

    if mag < 3.0:

return 0  # Low

    elif 3.0 <= mag <= 5.0:

        return 1  # Moderate

    else:

        return 2  # High

df['Magnitude_Label'] = df['Magnitude'].apply(magnitude_to_label)

x = df.drop(['Magnitude','Magnitude_Label'], axis=1) #x input features

y = df['Magnitude_Label'] #y target variable

#train test split method

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

print("x_train:\n",x_train)

print("x_test:\n",x_test)

#standard scaler method

scaler = StandardScaler()

```

#use fit method in training data

```
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
print("x_train_scaled:\n",x_train_scaled)
print("x_test_scaled:\n",x_test_scaled)
```

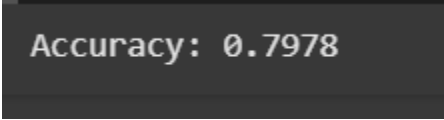
#Logistic Regression model

```
model = LogisticRegression(max_iter=1000)
model.fit(x_train_scaled, y_train)
y_pred = model.predict(x_test_scaled)
print("y_pred:\n",y_pred)
```

#accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

output:



```
Accuracy: 0.7978
```

#classification report

```
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=["Low",
"Moderate", "High"]))
```

output:

Classification Report:					
	precision	recall	f1-score	support	
Low	0.00	0.00	0.00	85	
Moderate	0.81	0.99	0.89	439	
High	0.00	0.00	0.00	20	
accuracy			0.80	544	
macro avg	0.27	0.33	0.30	544	
weighted avg	0.65	0.80	0.72	544	

#confusion matrix plotting

```

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Low", "Moderate",
"High"], yticklabels=["Low", "Moderate", "High"])

plt.xlabel("Predicted Label")

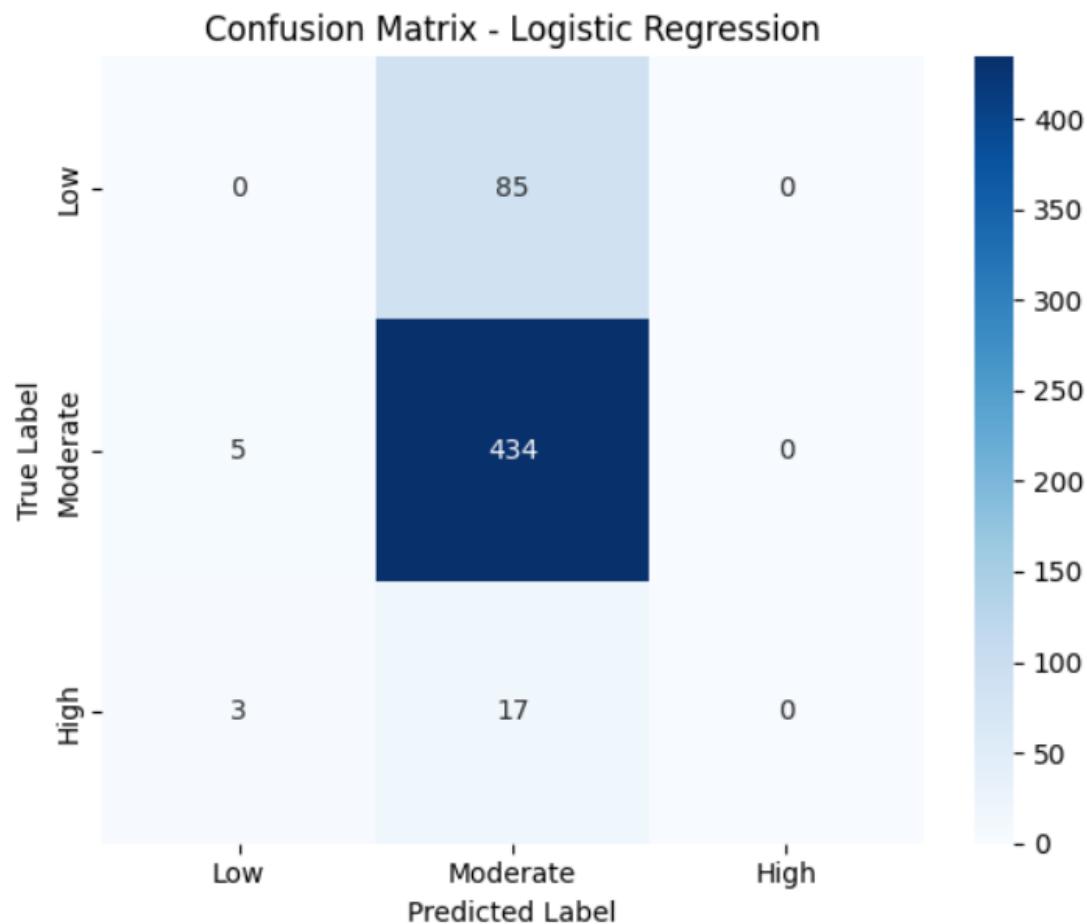
plt.ylabel("True Label")

plt.title("Confusion Matrix - Logistic Regression")

plt.tight_layout()

plt.show()

```



#Gradient Boosting Model

```
model = GradientBoostingClassifier(learning_rate=0.1, max_depth=5, random_state=42)
```

```
model.fit(X_train_scaled, y_train)
```

```
y_pred = model.predict(X_test_scaled)
```

```
print("y_pred:\n",y_pred)
```

#accuracy

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f" High Accuracy: {accuracy:.4f}")
```

output:

```
High Accuracy: 0.8382
```


#Confusion matrix

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Low", "Moderate",
"High"], yticklabels=["Low", "Moderate", "High"])

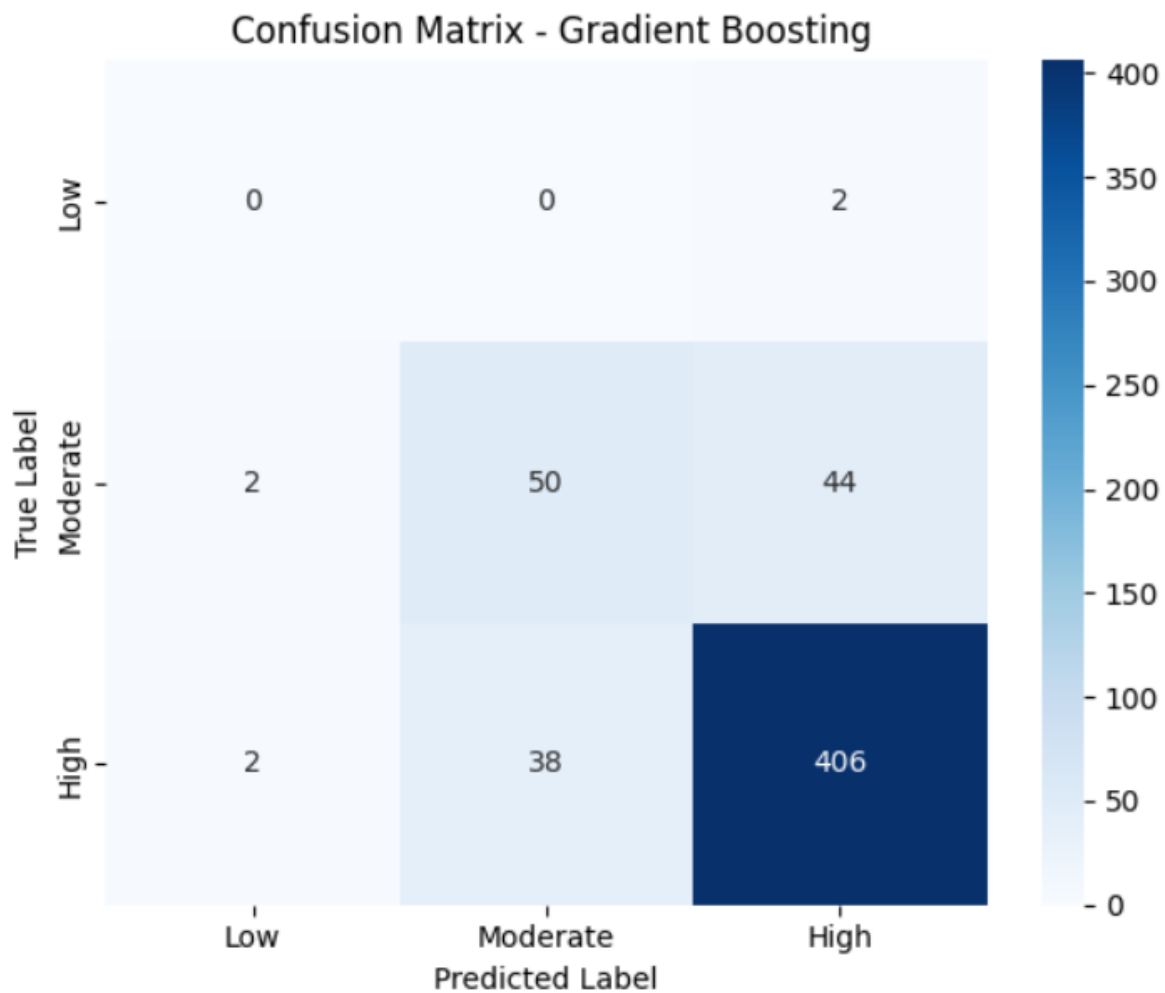
plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title(" Confusion Matrix - Gradient Boosting")

plt.tight_layout()

plt.show()
```



Conclusion

This project successfully demonstrates a machine learning-based approach to earthquake prediction. By using models like linear regression and gradient boosting, and organizing the system into modular classes, we efficiently predict earthquake severity. While prediction accuracy can improve with more data, the system lays a strong foundation for real-time seismic risk assessment