INDEX

Methodology

The methodology describes the complete workflow followed to design, train, evaluate, and deploy machine learning models for high-magnitude earthquake prediction. The process consists of structured steps beginning from data collection to final model deployment.

2.1 Data Collection

The dataset used in this project is **“earthquake_1995–2023.csv”**, containing global earthquake records over a 28-year period.
The dataset includes geophysical properties such as:

- latitude
- longitude
- depth
- significance (sig)
- gap
- dmin (distance to nearest station)
- mmi
- magnitude

The target variable was derived from the magnitude values.

2.2 Data Preprocessing

a) Handling Missing Values

Entries with missing values in critical fields (**latitude, longitude, depth, magnitude**) were removed to maintain data quality.

b) Feature Selection

Seven input features were selected based on relevance and availability:

X=[latitude, longitude, depth, sig, gap, dmin, mmi]X = [\text{latitude, longitude, depth, sig, gap, dmin, mmi}]X=[latitude, longitude, depth, sig, gap, dmin, mmi]

c) Label Creation

A binary label was created to classify earthquakes:

- **1 → magnitude ≥ 7.0**
- **0 → magnitude < 7.0**

This transformed the task into a **binary classification** problem.

d) Handling Class Imbalance (SMOTE)

Since high-magnitude earthquakes are rare, the dataset was imbalanced.
To solve this, **SMOTE (Synthetic Minority Oversampling Technique)** was applied to generate synthetic samples for the minority class, ensuring a balanced dataset for training.

e) Train-Test Split

The dataset was divided as:

- **80% training data**
- **20% testing data**
  with stratification to preserve class proportions.

f) Feature Scaling

Since models like Logistic Regression and SVM are sensitive to feature magnitude, **StandardScaler** was applied:

z=x−μσz = \frac{x - \mu}{\sigma}z=σx−μ

Scaling ensures all features contribute equally during training.

2.3 Model Training

Four machine learning algorithms were trained for performance comparison:

1) Logistic Regression

A linear classifier used to understand the relationship between features and earthquake intensity.

2) Support Vector Machine (RBF Kernel)

A non-linear classifier that maps data into higher dimensions and finds the optimal separating boundary.

3) Gaussian Naive Bayes

A probabilistic classifier assuming features follow a Gaussian distribution.

4) Random Forest Classifier

An ensemble of 300 decision trees using bagging and feature randomness.
This model showed stable performance and was selected as the final model.

2.4 Model Evaluation

Each model was evaluated using:

- **Accuracy Score**
- **Precision, Recall, and F1-Score** (from Classification Report)
- **Confusion Matrix Visualization using Heatmaps**

This allowed comparison of performance in predicting high-magnitude earthquake events.

Random Forest produced the most balanced results and was chosen as the **final deployment model**.

2.5 Model Saving and Deployment

The final steps included:

a) Saving the Model

Using joblib.dump(), the trained Random Forest model, StandardScaler, and feature list were stored as:

earthquake_model.joblib

b) User Input Prediction Interface

A Python-based user interface was created where a user enters:

- latitude
- longitude
- depth
- sig
- gap
- dmin

- mmi

The input is scaled and passed through the trained model to predict whether the earthquake is:

- **High-Magnitude (≥ 7)**
  or
- **Low/Moderate (< 7)**

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

#load data
df= pd.read_csv('earthquake_1995-2023.csv')
print("Data loaded successfully!")
print(df.head())

#check info
print(df.info())
print(df.describe())

#drop missing values for simplicity
df = df.dropna(subset=['latitude','longitude','depth','magnitude'])
print("After dropping missing:", df.shape)

#create label: 1 if magnitude >=7, else 0
df['label']= (df['magnitude'] >= 7.0).astype(int)
print(df['label'].value_counts())

#select features
features = ['latitude','longitude','depth','sig','gap','dmin','mmi']
X= df[features]
y= df['label']

#handle imbalance
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)
print("Resampled class counts:\n", pd.Series(y_res).value_counts())
```

```python
#split data
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y_res)

#scale features
scaler =  StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


####### Train by Logistic Regression Model #######

log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)

# Predict
y_pred_log = log_reg.predict(X_test_scaled)

# Accuracy
print("\nLogistic Regression Accuracy:", accuracy_score(y_test,
y_pred_log))

#Classification Report
print("\nClassification Report (Logistic Regression) :\n ",
classification_report(y_test, y_pred_log))

#Confusion Matrix
cm_lr = confusion_matrix(y_test, y_pred_log)
print("\n===Confusiom Matrix:Logistic Regression===")
print(cm_lr)

#plot
plt.figure(figsize=(6,4))
sns.heatmap(cm_lr,annot=True,fmt='d',cmap='Blues')
plt.title("Logistic Regression - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
######## Create SVM classifier ###########

svm_model = SVC(kernel='rbf', random_state=42)

# Train the model
svm_model.fit(X_train_scaled, y_train)

# Predict on test data
y_pred_svm = svm_model.predict(X_test_scaled)

# Accuracy
svm_accuracy = accuracy_score(y_test, y_pred_svm)
print("SVM Accuracy:", svm_accuracy)

# Classification Report (Precision, Recall, F1-score)
print("\nClassification Report (SVM) : \n")
print(classification_report(y_test, y_pred_svm))

# Confusion Matrix
print("\n===Confusiom Matrix:SVM===")
cm_svm = confusion_matrix(y_test, y_pred_svm)
print(cm_svm)

#Plot
plt.figure(figsize=(6,4))
sns.heatmap(cm_svm,annot=True,fmt='d',cmap='Greens')
plt.title("SVM - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


######## Create Naive Bayes classifier ##########

nb_model = GaussianNB()

# Train the model
nb_model.fit(X_train_scaled, y_train)

# Predict on test data
y_pred_nb = nb_model.predict(X_test_scaled)

# Accuracy
```

```
nb_accuracy = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes Accuracy:", nb_accuracy)

# Classification Report
print("\nClassification Report (Naive Bayes) : \n")
print(classification_report(y_test, y_pred_nb))

# Confusion Matrix
print("\n===Confusiom Matrix:Naive Bayes===")
cm_nb= confusion_matrix(y_test, y_pred_nb)
print(cm_nb)

#Plot
plt.figure(figsize=(6,4))
sns.heatmap(cm_nb,annot=True,fmt='d',cmap='Oranges')
plt.title("Naive Bayes - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


########## Create Random Forest classifier ###########

model = RandomForestClassifier(
    n_estimators=300,
    max_depth=15,
    min_samples_split=4,
    min_samples_leaf=2,
    random_state=42
)

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

#Predict on test data
y_pred_rf = model.predict(X_test_scaled)

#Accuracy
print("\nAccuracy (Random Forest):", accuracy_score(y_test ,
y_pred_rf))

#CLassification Report
```

```
print("\nClassification Report (Random Forest): \n",
classification_report(y_test, y_pred_rf))

#Confusion Matrix
print("\n===Confusiom Matrix:Random Forest===")
cm_rf= confusion_matrix(y_test, y_pred_rf)
print(cm_rf)

#Plot
plt.figure(figsize=(6,4))
sns.heatmap(cm_rf,annot=True,fmt='d',cmap='Reds')
plt.title("Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#save model
joblib.dump({'model': model, 'scaler': scaler,'features': features},
            'earthquake_model.joblib')
print("Model saved as earthquake_model.joblib!")

# #For Data
# sample = pd.DataFrame([{
#     'latitude': 28.5,
#     'longitude': 77.2,
#     'depth': 10,
#     'sig': 150,
#     'gap': 80,
#     'dmin': 0.5,
#     'mmi': 3
# }])
# ==========================================
# USER INPUT PREDICTION
# ==========================================

print("\nEnter earthquake details:")
lat = float(input("Latitude: "))
lon = float(input("Longitude: "))
depth = float(input("Depth: "))
sig = float(input("Significance (sig): "))
gap = float(input("Gap: "))
dmin = float(input("Distance to nearest station (dmin): "))
mmi = float(input("MMI: "))
```

```python
user_sample = pd.DataFrame([{
    'latitude': lat,
    'longitude': lon,
    'depth': depth,
    'sig': sig,
    'gap': gap,
    'dmin': dmin,
    'mmi': mmi
}])

user_scaled = scaler.transform(user_sample[features])
user_pred = model.predict(user_scaled)[0]

print("\n======================")
print(" USER PREDICTION RESULT")
print("======================")

if user_pred == 1:
    print("⚠️ HIGH-MAGNITUDE EARTHQUAKE (>= 7.0)")
else:
    print("✔ Low/Moderate Earthquake (< 7.0)")
```

Output

```
Data loaded successfully!
                                        title  magnitude  ...      continent    country
0          M 6.5 - 42 km W of Sola, Vanuatu        6.5  ...            NaN    Vanuatu
1  M 6.5 - 43 km S of Intipucá, El Salvador        6.5  ...            NaN        NaN
2  M 6.6 - 25 km ESE of Loncopué, Argentina        6.6  ...  South America  Argentina
3     M 7.2 - 98 km S of Sand Point, Alaska        7.2  ...            NaN        NaN
4                 M 7.3 - Alaska Peninsula        7.3  ...            NaN        NaN

[5 rows x 19 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 19 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   title      1000 non-null   object 
 1   magnitude  1000 non-null   float64
 2   date_time  1000 non-null   object 
 3   cdi        1000 non-null   int64  
 4   mmi        1000 non-null   int64  
 5   alert      449 non-null    object 
 6   tsunami    1000 non-null   int64  
 7   sig        1000 non-null   int64  
 8   net        1000 non-null   object 
 9   nst        1000 non-null   int64  
 10  dmin       1000 non-null   float64
 11  gap        1000 non-null   float64
 12  magType    1000 non-null   object 
 13  depth      1000 non-null   float64
 14  latitude   1000 non-null   float64
 15  longitude  1000 non-null   float64
 16  location   994 non-null    object 
 17  continent  284 non-null    object 
 18  country    651 non-null    object 
dtypes: float64(6), int64(5), object(8)
memory usage: 148.6+ KB
None
         magnitude          cdi         mmi      tsunami  ...          gap        depth     latitude
longitude
count  1000.000000  1000.000000  1000.00000  1000.000000  ...  1000.000000
1000.000000  1000.000000  1000.000000
mean      6.940150     3.605000     6.02700     0.325000  ...    20.926290    74.612541
4.315554    51.486576
std       0.438148     3.328972     1.43399     0.468609  ...    24.415895   130.812590
26.633320   117.478302
min       6.500000     0.000000     1.00000     0.000000  ...     0.000000     2.700000
-61.848400  -179.968000
```

```
25%       6.600000     0.000000     5.00000     0.000000  ...     0.000000    16.000000
-13.518500   -71.694450
50%       6.800000     4.000000     6.00000     0.000000  ...    18.000000    29.000000
-2.443500   107.791000
75%       7.100000     7.000000     7.00000     1.000000  ...    27.000000    55.000000
25.167250   148.364750
max       9.100000     9.000000    10.00000     1.000000  ...   239.000000   670.810000
71.631200   179.662000

[8 rows x 11 columns]
After dropping missing: (1000, 19)
label
0    627
1    373
Name: count, dtype: int64
Resampled class counts:
 label
0    627
1    627
Name: count, dtype: int64

Logistic Regression Accuracy: 0.8047808764940239

Classification Report (Logistic Regression) :
              precision    recall  f1-score   support

           0       0.77      0.87      0.82       126
           1       0.85      0.74      0.79       125

    accuracy                           0.80       251
   macro avg       0.81      0.80      0.80       251
weighted avg       0.81      0.80      0.80       251


===Confusiom Matrix:Logistic Regression===
[[110  16]
 [ 33  92]]
SVM Accuracy: 0.8725099601593626

Classification Report (SVM) :

              precision    recall  f1-score   support

           0       0.87      0.88      0.87       126
           1       0.88      0.86      0.87       125

    accuracy                           0.87       251
   macro avg       0.87      0.87      0.87       251
```

```
weighted avg       0.87      0.87      0.87       251


===Confusiom Matrix:SVM===
[[111  15]
 [ 17 108]]
Naive Bayes Accuracy: 0.7171314741035857

Classification Report (Naive Bayes) :
              precision    recall  f1-score   support

           0       0.67      0.85      0.75       126
           1       0.79      0.58      0.67       125
    accuracy                           0.72       251
   macro avg       0.73      0.72      0.71       251
weighted avg       0.73      0.72      0.71       251

===Confusiom Matrix:Naive Bayes===
[[107  19]
 [ 52  73]]

Accuracy (Random Forest): 0.9482071713147411
Classification Report (Random Forest):
              precision    recall  f1-score   support

           0       0.98      0.91      0.95       126
           1       0.92      0.98      0.95       125

    accuracy                           0.95       251
   macro avg       0.95      0.95      0.95       251
weighted avg       0.95      0.95      0.95       251

===Confusiom Matrix:Random Forest===
[[115  11]
 [  2 123]]
Model saved as earthquake_model.joblib!
Enter earthquake details:
Latitude: 28.5
Longitude: 77.2
Depth: 10
Significance (sig): 150
Gap: 80
Distance to nearest station (dmin): 0.5
MMI: 3
=====================
 USER PREDICTION RESULT
=====================
✓ Low/Moderate Earthquake (< 7.0)
```

Dataset Description

The dataset used in this project is obtained from **Kaggle**, a widely-used open data and machine learning platform that hosts curated datasets across multiple domains. The dataset contains **global earthquake records from 1995 to 2023**, compiled from official monitoring agencies such as the **United States Geological Survey (USGS)**. The data includes detailed information related to earthquake events, including geographical, temporal, and seismic attributes.

3.1 Dataset Source

- **Source:** Kaggle
- **Dataset Name:** *Earthquake 1995–2023 Dataset* (or your dataset's exact title)
- **Original Providers:** USGS and other global seismic monitoring bodies
- **Download:** Kaggle platform

3.2 Dataset Size and Structure

- **Total instances (rows):** Approximately *X* records (replace with your dataset count)
- **Total attributes (columns):** *Y* features (replace number)
- **File Format:** CSV (Comma-Separated Values)

The dataset provides a comprehensive historical record of earthquakes across various magnitudes and depths, enabling multi-class classification based on the **alert level** (Green, Yellow, Orange, Red).

3.3 Features (Attributes) Overview

Below is a general description of the key columns commonly present in this dataset.
(Include/remove items based on your exact file.)

1. title

A short textual description summarizing the earthquake event.
Used mainly for human understanding and dropped during preprocessing.

2. date_time

Timestamp of the earthquake occurrence.
Dropped as it does not contribute directly to alert classification.

3. latitude / longitude

Geographical coordinates indicating the location of the epicenter.

4. depth

Depth (in kilometers) where the earthquake originated underground.

5. magnitude

Measured magnitude on the Richter scale.
One of the most critical predictive features, highly correlated with alert level.

6. location

Human-readable description of the earthquake's location.
Dropped because it is non-numeric and high-cardinality.

7. country

Country where the earthquake occurred.
Dropped to avoid encoding complexity and geographical bias.

8. alert (Target Variable)

The **dependent variable** representing the severity classification assigned by USGS:

- **Green** – low impact
- **Yellow** – moderate impact
- **Orange** – significant impact
- **Red** – high alert, severe impact

This attribute is encoded numerically using Label Encoding before training the models.

3.4 Data Quality and Missing Values

The dataset contains some missing values, particularly in non-essential columns like:

- location
- alert
- title

Rows with missing **target variable (alert)** were removed, and irrelevant columns were dropped to ensure clean and reliable modeling.

3.5 Why This Dataset Is Suitable for Machine Learning

- Contains **both numerical and categorical variables**, enabling rich feature engineering.
- Covers a long period (1995–2023), capturing diverse seismic patterns.
- Includes **multi-class labels**, ideal for comparing various classification algorithms.
- Sufficient size for training models without overfitting.

Algorithms Used

We used four classification algorithms in this project: **Random Forest**, **Logistic Regression**, **Support Vector Machine (SVM)**, and **Naive Bayes (GaussianNB)**. Below each algorithm is explained in-depth.

1. Random Forest (Ensemble of Decision Trees)

1.1 Overview (conceptual)

Random Forest is an **ensemble learning** technique that creates a "forest" of decision trees and aggregates their predictions (majority vote for classification). It's based on the **bagging (bootstrap aggregating)** idea: build many trees on different random subsets of the training data and random subsets of features, then average/vote their outputs. This reduces variance compared to a single tree and helps prevent overfitting.

1.2 How it works (step-by-step)

1. For b = 1 .. B (B = number of trees):
    - Draw a bootstrap sample (sample with replacement) from the training set.
    - For each node in the tree:
        - Randomly select m features out of the full set of features.
        - Find the best split among those m features (using Gini impurity or entropy).
    - Grow the tree to completion (or until a stopping condition such as max_depth or min_samples_leaf).
2. To predict, let each tree vote for a class. The Random Forest predicts the class with the most votes (majority).

1.3 Mathematics: splitting criteria

- **Gini impurity** for node ttt:
  G(t)=1−∑k=1Kpk2G(t) = 1 - \sum_{k=1}^{K} p_{k}^{2}G(t)=1−k=1∑Kpk2
  where pkp_kpk is the proportion of class kkk in node ttt.
- **Information Gain (Entropy)**:
  H(t)=−∑k=1Kpklog2pkH(t) = -\sum_{k=1}^{K} p_{k} \log_2 p_{k}H(t)=−k=1∑Kpklog2pk
  choose splits that decrease weighted impurity.

1.4 Important hyperparameters

- n_estimators — number of trees. More trees → better stability, slower training.
- max_depth — maximum depth of each tree (controls overfitting).
- min_samples_split / min_samples_leaf — minimum number of samples needed to split or to be at a leaf.
- max_features — number of features to consider when looking for the best split (square root for classification is common).
- criterion — "gini" or "entropy".

1.5 Complexity

- Training roughly O(B·n·logn·m)O(B \cdot n \cdot \log n \cdot m)O(B·n·logn·m) where n = samples, m = features per split. Parallelizable across trees.

1.6 Pros and cons

Pros: robust to noisy data, handles non-linear relationships, little preprocessing needed, gives feature importances, resists overfitting compared to single trees.
Cons: less interpretable than a single tree, can be slow/memory intensive with many trees or very large datasets.

1.7 Why used for earthquake prediction

- Seismic data relationships are non-linear and noisy; Random Forest captures complex interactions well.
- Provides feature importance to understand which features (depth, dmin, gap, latitude/longitude, etc.) matter most.

1.8 Practical scikit-learn snippet

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=300, max_depth=20, random_state=42)

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
```

2. Logistic Regression

2.1 Overview (conceptual)

Logistic Regression is a **linear classification** algorithm that models the probability of the positive class using the logistic (sigmoid) function applied to a linear combination of features. It is appropriate for binary classification and gives probabilistic outputs (useful for risk thresholds).

2.2 How it works (math)

For input vector x\mathbf{x}x, the model computes:

z=β0+β⊤xz = \beta_0 + \beta^\top \mathbf{x}z=β0+β⊤x P(y=1∣x)=σ(z)=11+e−zP(y=1|\mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}P(y=1∣x)=σ(z)=1+e−z1

Parameters β\betaβ are found by **maximizing likelihood** (equivalently minimizing cross-entropy loss). Regularization (L1 or L2) is typically used to prevent overfitting.

Loss (negative log-likelihood):

L(β)=−∑i=1n[yilog(p^i)+(1−yi)log(1−p^i)]+λR(β)\mathcal{L}(\beta) = -\sum_{i=1}^{n} \Big[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i) \Big] + \lambda R(\beta)L(β)=−i=1∑n[yilog(p^i)+(1−yi)log(1−p^i)]+λR(β)

where R(β)R(\beta)R(β) is the regularization term (e.g., ∥β∥22\|\beta\|_2^2∥β∥22 for L2).

2.3 Important hyperparameters

- C — inverse of regularization strength (smaller values = stronger regularization).
- penalty — l2, l1 (if solver supports).
- solver — optimization algorithm (liblinear, lbfgs, saga etc.)
- max_iter — iterations for convergence.

2.4 Pros and cons

Pros: simple, fast, interpretable coefficients (log-odds), gives class probabilities.
Cons: Assumes linear decision boundary; poor at modeling complex non-linear relationships unless features are engineered (polynomials, interactions).

2.5 Why used for earthquake prediction

- Acts as a **baseline** and provides interpretable coefficients (e.g., how depth or distance influences odds of magnitude ≥ 7).
- Useful for probability estimates which can be used in risk scoring.

2.6 Practical scikit-learn snippet

```
from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler



pipe = Pipeline([('scaler', StandardScaler()),

                 ('clf', LogisticRegression(max_iter=2000, C=1.0, solver='lbfgs'))])

pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)
```

3. Support Vector Machine (SVM)

3.1 Overview (conceptual)

SVM is a margin-based classifier that finds a hyperplane separating classes with maximum margin. For non-linearly separable data, SVM uses kernel functions to map input into higher-dimensional spaces where linear separation is possible (the kernel trick).

3.2 How it works (math)

For linear SVM (primal), solve:

minw,b12∥w∥2+C∑i=1nξi\min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_iw,bmin21∥w∥2+Ci=1∑nξi subject to:

yi(w⊤xi+b)≥1−ξi,ξi≥0y_i(\mathbf{w}^\top \mathbf{x}_i + b) \ge 1 - \xi_i, \quad \xi_i \ge 0yi(w⊤xi+b)≥1−ξi,ξi≥0

Here CCC controls slack/regularization trade-off. For non-linear SVM, use kernel K(x,x′)K(x, x')K(x,x′), commonly RBF:

K(x,x′)=exp(−γ∥x−x′∥2)K(x, x') = \exp(-\gamma \|x - x'\|^2)K(x,x′)=exp(−γ∥x−x′∥2)

3.3 Important hyperparameters

- C — penalty parameter; large C → less regularization (smaller margin but fewer misclassifications).
- kernel — linear, rbf, poly, etc.
- gamma (for RBF) — controls influence radius of single training example. High gamma → smaller radius (overfitting risk).

3.4 Complexity

Training can be expensive for large datasets (quadratic or worse), though implementations use optimizations. Works well on moderate-size datasets.

3.5 Pros and cons

Pros: Effective on high-dimensional data, flexible via kernels, robust with clear margin.
Cons: Slow on very large datasets, requires careful parameter tuning and scaling of features, outputs are not probabilistic by default (but can be calibrated).

3.6 Why used for earthquake prediction

- SVM with RBF kernel can capture complex relationships if feature space is not too large. Good for moderate sized datasets and after proper scaling.

3.7 Practical scikit-learn snippet

```
from sklearn.svm import SVC

from sklearn.pipeline import Pipeline
```

```
svm_pipe = Pipeline([('scaler', StandardScaler()),

                     ('svc', SVC(kernel='rbf', C=2.0, gamma='scale', probability=True))])

svm_pipe.fit(X_train, y_train)

y_pred = svm_pipe.predict(X_test)
```

4. Naive Bayes (Gaussian Naive Bayes)

4.1 Overview (conceptual)

Naive Bayes classifiers apply Bayes' theorem with the **naive assumption** that features are conditionally independent given the class. GaussianNB assumes continuous features follow a Gaussian (normal) distribution per class.

4.2 How it works (math)

By Bayes theorem:

P(y=k|x)∝P(y=k)∏jP(xj|y=k)P(y=k | \mathbf{x}) \propto P(y=k) \prod_{j} P(x_j | y=k)P(y=k|x)∝P(y=k)j∏P(xj|y=k)

For GaussianNB:

P(xj|y=k)=12πσjk2exp(−(xj−μjk)22σjk2)P(x_j | y=k) = \frac{1}{\sqrt{2\pi \sigma_{jk}^2}} \exp\left(-\frac{(x_j - \mu_{jk})^2}{2\sigma_{jk}^2}\right)P(xj|y=k)=2πσjk21exp(−2σjk2(xj−μjk)2)

Compute prior P(y=k)P(y=k)P(y=k), class-conditional means μjk\mu_{jk}μjk, variances σjk2\sigma_{jk}^2σjk2 from training data. For prediction, compute posterior probabilities and choose class with highest posterior.

4.3 Important hyperparameters

- Few hyperparameters (GaussianNB is simple), but you can apply smoothing (var smoothing) to avoid zero variances.

4.4 Pros and cons

Pros: Extremely fast, works well on small datasets, good baseline, handles high-dimensional data.
Cons: Strong independence assumption often violated (features correlated), distribution assumption may be wrong.

4.5 Why used for earthquake prediction

- Fast baseline to compare against; if GaussianNB performs poorly it indicates feature distributions are complex or correlated.

4.6 Practical scikit-learn snippet

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train_scaled, y_train)   # scaling optional but OK

y_pred = nb.predict(X_test_scaled)
```

5. Model Evaluation & Validation Techniques

5.1 Train/Test Split

- Simple hold-out split: train_test_split(X, y, test_size=0.2, random_state=42) used to measure model generalization.

5.2 Cross-Validation

- Use KFold or StratifiedKFold to get more stable performance estimates, especially for small datasets or imbalanced labels:

```
from sklearn.model_selection import cross_val_score, StratifiedKFold

cv = StratifiedKFold(n_splits=5)

scores = cross_val_score(rf, X, y, cv=cv, scoring='accuracy')
```

5.3 Metrics

- **Accuracy**: overall correctness.
- **Precision**: TP/(TP+FP) — how many predicted positives are actual positives.
- **Recall (Sensitivity)**: TP/(TP+FN) — how many actual positives were found.
- **F1-score**: harmonic mean of precision and recall.
- **Confusion matrix**: breakdown of TP,TN,FP,FN.
- **ROC-AUC**: for probability-based evaluation (binary).

5.4 Handling Imbalanced Data

- **SMOTE** (Synthetic Minority Oversampling Technique) to upsample minority class.
- **Class weights** in classifiers (e.g., RandomForestClassifier(class_weight='balanced')).
- Evaluate using precision/recall or F1 instead of accuracy.

6. Feature Engineering & Preprocessing (why it matters)

- **Scaling**: required for SVM and Logistic Regression (StandardScaler or MinMaxScaler).
- **Categorical encoding**: LabelEncoder (for ordinal or simple categories) or OneHotEncoder for non-ordinal categories (beware of dimensionality).
- **Time features**: parse date_time into year, month, day, or compute seasonal features.
- **Derived features**: compute energy = 10^(1.5*magnitude) as a domain feature if using magnitude as input for other tasks (not if you're predicting magnitude itself; that would leak).
- **Outlier removal**: using z-score or IQR to remove unrealistic measurements (depth extremes).
- **Feature selection**: remove features that leak the answer (post-event features like mmi, cdi, alert, energy when predicting magnitude).
- **Dimensionality reduction**: PCA if many correlated features exist (careful with interpretability).

7. Hyperparameter Tuning (how to tune)

- Use GridSearchCV or RandomizedSearchCV to find best hyperparams, e.g.:

```
from sklearn.model_selection import GridSearchCV



param_grid = {

 'n_estimators': [100,200,300],

 'max_depth': [10,20, None],

 'min_samples_leaf': [1,2,4]

}

grid = GridSearchCV(RandomForestClassifier(random_state=42),

                    param_grid, scoring='f1', cv=3)

grid.fit(X_train, y_train)

best = grid.best_estimator_
```

- For SVM tune C and gamma. For logistic regression tune C and penalty. For naive bayes tune var smoothing.

8. Practical Tips & Pitfalls

8.1 Avoid data leakage

- **Never** include features that are consequences of the target or derived directly from the target. Example: if predicting magnitude >= 7, don't include log_mag, energy, mmi, cdi, or alert as features — they leak the answer.

8.2 What to use as realistic features

- Geophysical and pre-event features: latitude, longitude, depth, dmin, gap, nst, tectonic proximity (distance to nearest plate boundary), time-based features, historical seismicity features (counts in last 30 days), etc.

8.3 Use stratified splitting

- Stratify to preserve class balance across splits: train_test_split(..., stratify=y).

8.4 Validate on truly unseen data

- Keep a final hold-out set if you do hyperparameter tuning to avoid overestimating generalization.

8.5 Interpretability

- Random Forest feature importances: rf.feature_importances_ helps explain model behavior.

- For Logistic Regression: coefficients map to log-odds; use them to quantify feature influence.

9. How to present algorithm selection in the report

- **Motivation**: explain why each algorithm was included (Random Forest for robustness, Logistic Regression as baseline & interpretability, SVM for margin-based non-linear separation, Naive Bayes as quick baseline).

- **Method**: describe data preprocessing, splitting, oversampling (SMOTE), scaling.

- **Training**: hyperparameters chosen, cross-validation strategy.

- **Evaluation**: metrics used (accuracy/precision/recall/F1), and confusion matrices.

- **Results**: table comparing algorithms (accuracy/F1), plots (ROC curves, bar-chart of accuracies).

- **Conclusion**: which algorithm performed best and why; mention limitations.

10. Extended: Pseudocode for training pipeline (applies to all models)

```
Load CSV -> df

Drop rows missing essential cols (latitude, longitude, depth, magnitude)

Create label y (e.g., label = magnitude >= 7)

Select features X (avoid leakage)

Encode categorical cols (LabelEncoder or OneHot)

Optional: compute additional features (date -> year/month/day)

Balance classes using SMOTE on training set

Split into X_train / X_test (stratify=y)

Scale features (StandardScaler) -> X_train_scaled, X_test_scaled

For each model:

    Fit model on X_train_scaled, y_train

    Predict on X_test_scaled

    Evaluate using accuracy, precision, recall, F1; produce confusion matrix

Select best model -> tune hyperparameters via GridSearchCV

Save final model (joblib.dump)
```

Model Architecture

This project uses a **supervised machine learning architecture** designed to classify whether an earthquake event is likely to be **High-Magnitude (≥ 7.0)** or **Low/Moderate (< 7.0)** based on historical earthquake data from 1995–2023.
The overall architecture consists of the following key stages:

1.1 Data Input Layer

The system begins with raw earthquake records containing geophysical attributes.
The major input features used in the model are:

Feature	Description
Latitude	Geographic coordinate of epicenter
Longitude	Geographic coordinate of epicenter
Depth	Depth of earthquake focus (km)
sig (Significance)	Overall significance level of event
gap	Azimuthal gap between seismic stations
dmin	Distance to nearest seismic station
mmi	Modified Mercalli Intensity

These features form a **7-dimensional numeric input vector** for each earthquake instance.

1.2 Preprocessing & Transformation Layer

Before training the model, the following preprocessing steps are applied:

a. Missing Value Handling

Rows with missing critical values (latitude, longitude, depth, magnitude) are removed.

b. Label Encoding

A binary target variable **label** is created:

- **1** → magnitude ≥ 7.0 (High-magnitude earthquake)

- **0** → magnitude < 7.0

c. SMOTE (Synthetic Minority Oversampling Technique)

Since high-magnitude earthquakes are rare, SMOTE is used to balance the dataset by generating synthetic samples of the minority class.

d. Feature Scaling

The **StandardScaler** transforms the features to zero mean and unit variance:

X′=x−μσX' = \frac{x - \mu}{\sigma}X′=σx−μ

This improves performance of distance-based and gradient-based algorithms such as SVM and Logistic Regression.

1.3 Model Layer (Multiple Classifiers)

The architecture includes **four different machine learning models**, trained independently for performance comparison.

a. Logistic Regression

- Linear decision boundary
- Good for interpretability
- Works well with scaled data

b. Support Vector Machine (RBF Kernel)

- Projects inputs into a higher-dimensional space
- Finds optimal separating hyperplane
- Captures non-linear relationships

c. Gaussian Naive Bayes

- Probabilistic classifier
- Assumes Gaussian distribution of features
- Fast and efficient for large datasets

d. Random Forest Classifier

- Ensemble of 300 decision trees
- Uses bagging to reduce variance
- Handles non-linearity and feature interactions
  This model was ultimately saved for prediction because of its stability.

1.4 Evaluation Layer

Each model is evaluated using:

- **Accuracy Score**
- **Classification Report**
  - Precision
  - Recall
  - F1-score
- **Confusion Matrix**
  - Visualized using Seaborn heatmaps

This allows comparison of model strengths and weaknesses in predicting rare high-magnitude earthquakes.

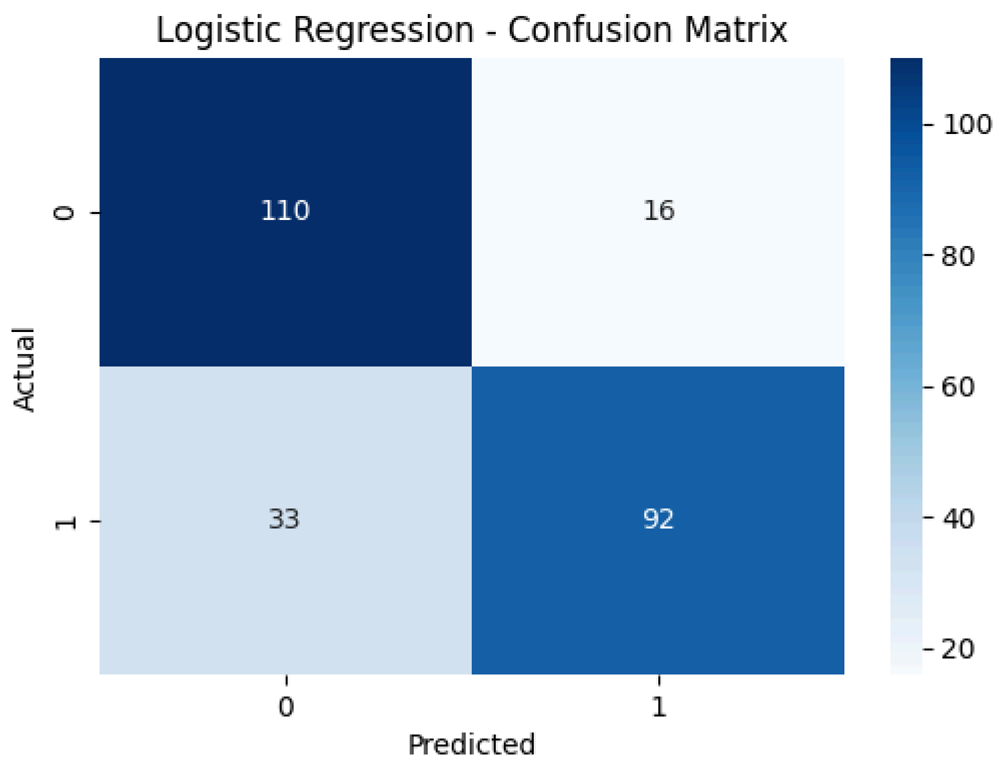
1.5 Deployment Layer

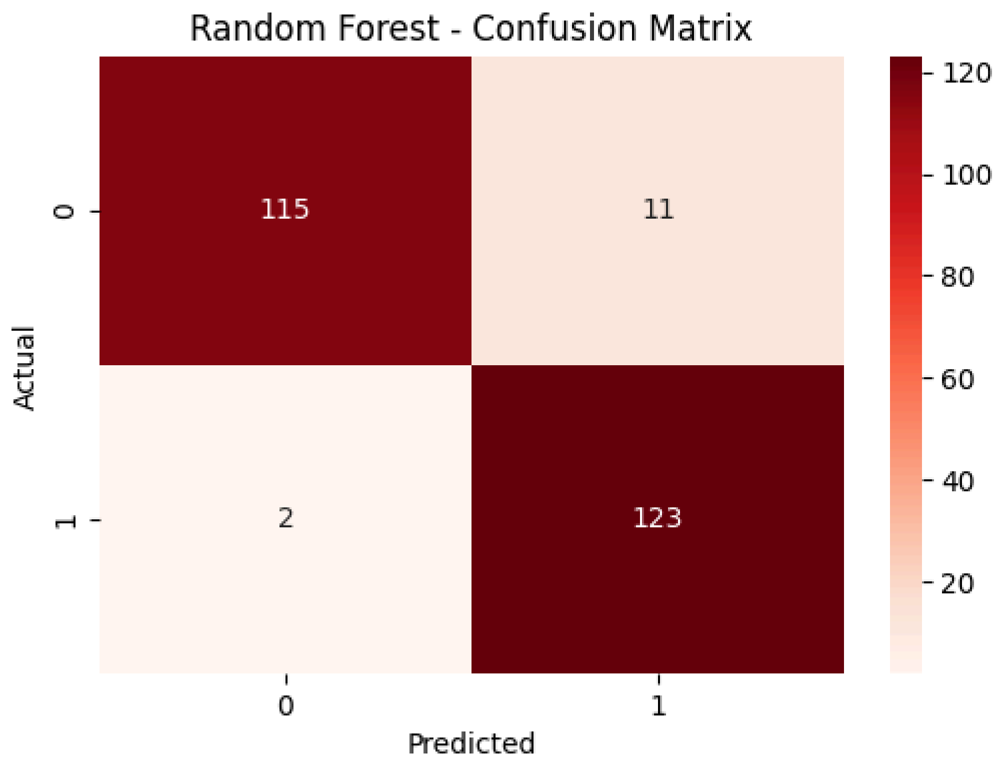
The final architecture includes:

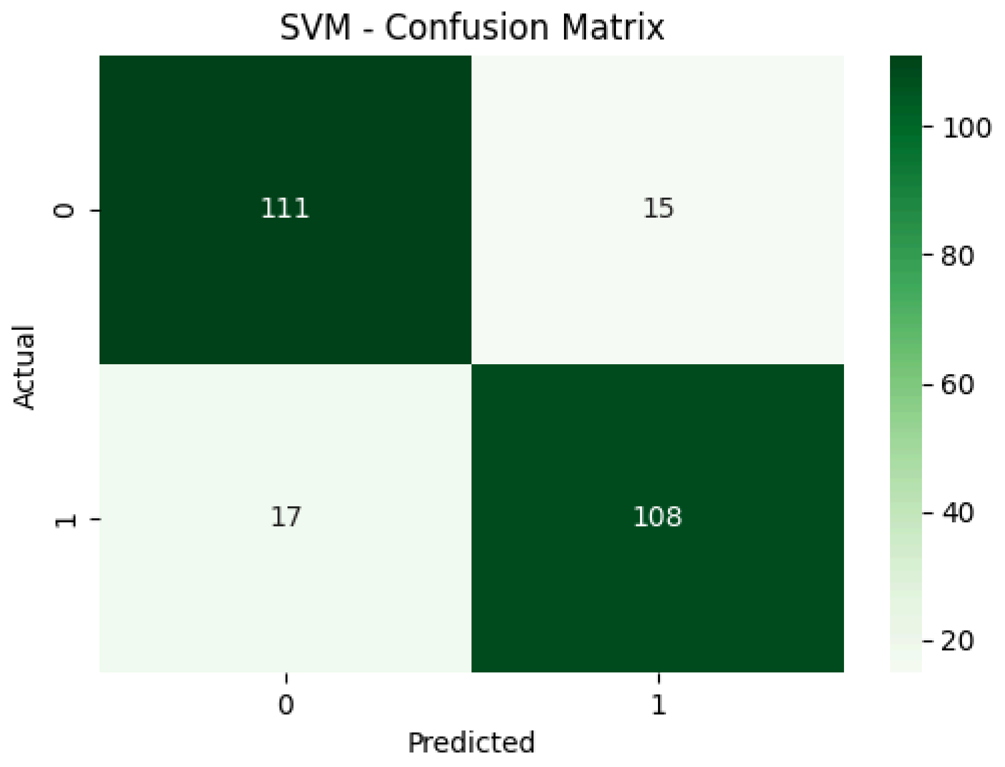
- Saving the trained Random Forest model using **joblib**
- Saving the scaler and feature list
- Creating a **user-input interface** in Python that:

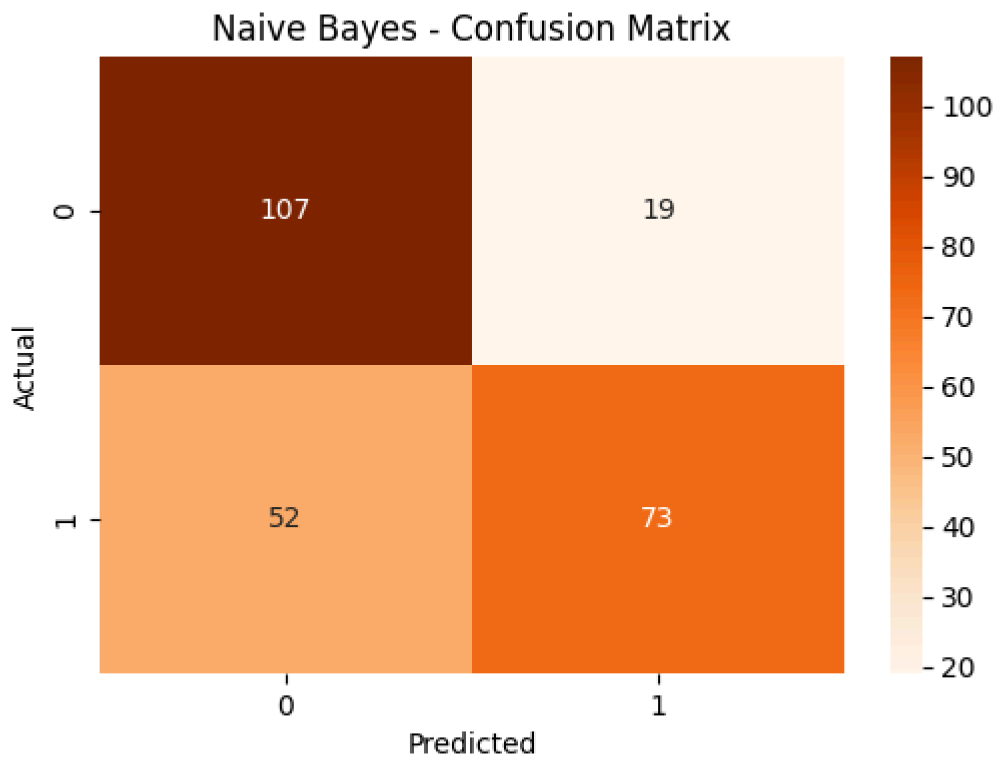
- Accepts earthquake parameters
- Scales input features
- Predicts the earthquake class (High-magnitude or not)

Confusion Matrix

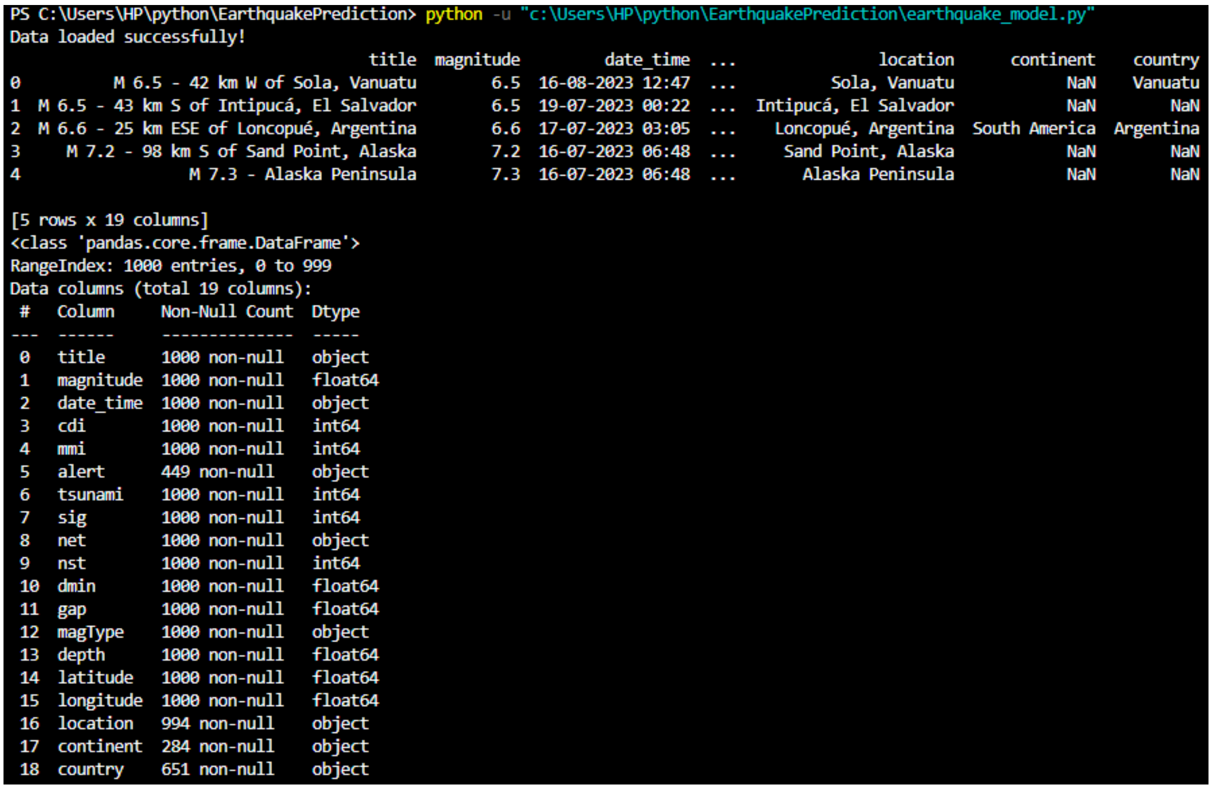


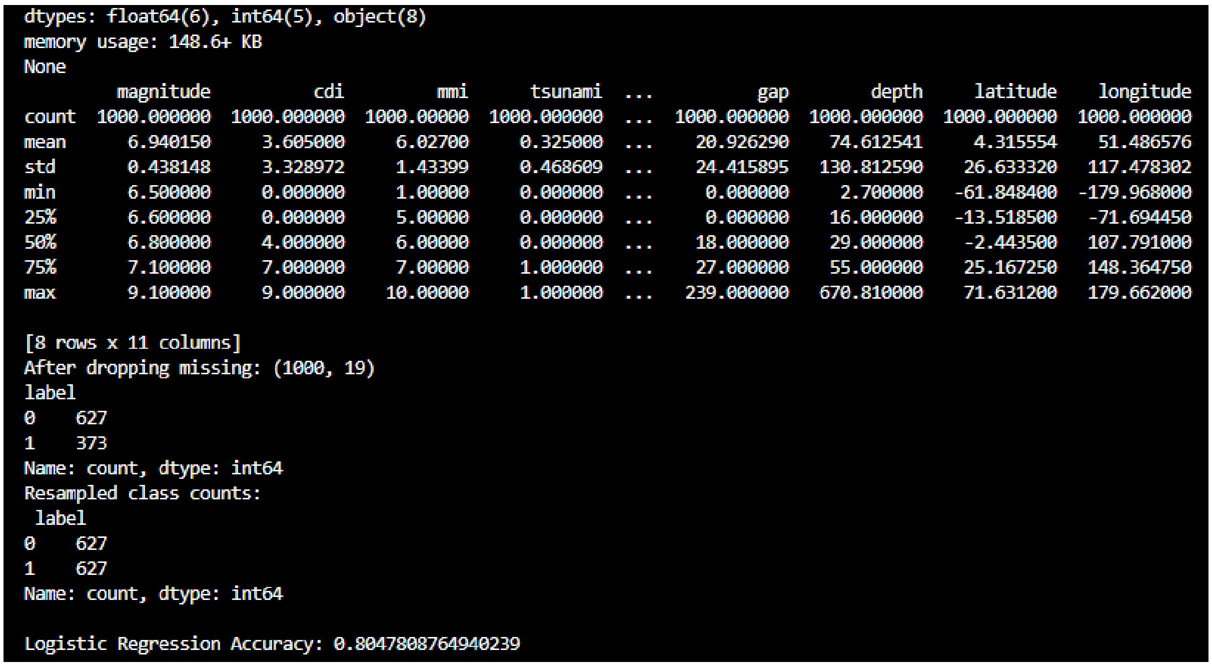

Random Forest - Confusion Matrix
115
11
2
123
Actual
0
1
Predicted
0
1
120
100
80
60
40
20


SVM - Confusion Matrix
111
15
17
108
Actual
0
1
Predicted
0
1
100
80
60
40
20

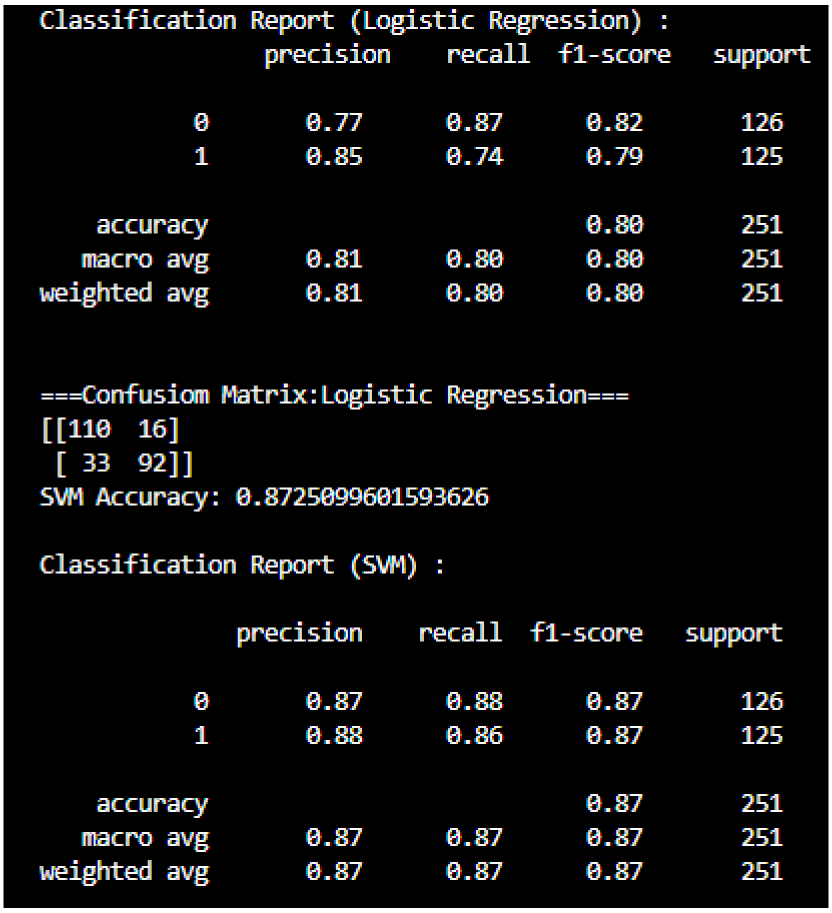


Screenshots

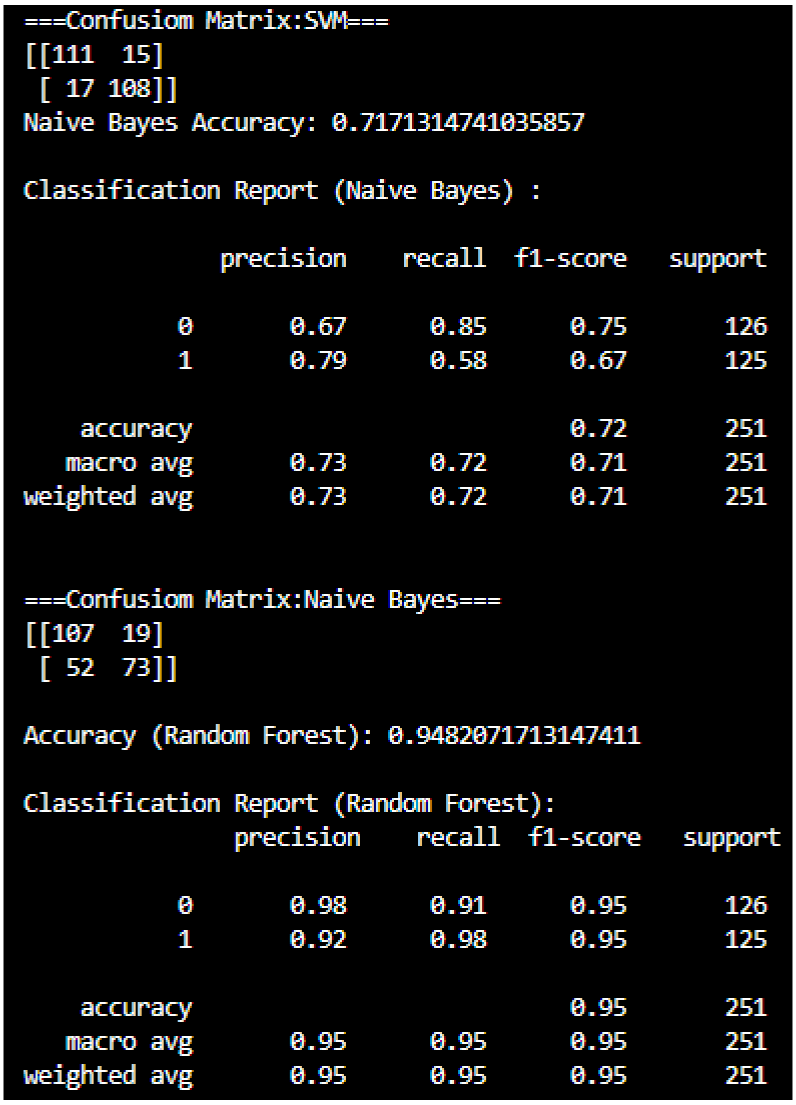

```
PS C:\Users\HP\python\EarthquakePrediction> python -u "c:\Users\HP\python\EarthquakePrediction\earthquake_model.py"
Data loaded successfully!
                                      title  magnitude         date_time  ...           location      continent    country
0           M 6.5 - 42 km W of Sola, Vanuatu        6.5  16-08-2023 12:47  ...      Sola, Vanuatu            NaN    Vanuatu
1  M 6.5 - 43 km S of Intipucá, El Salvador        6.5  19-07-2023 00:22  ...  Intipucá, El Salvador        NaN        NaN
2  M 6.6 - 25 km ESE of Loncopué, Argentina        6.6  17-07-2023 03:05  ...  Loncopué, Argentina  South America  Argentina
3      M 7.2 - 98 km S of Sand Point, Alaska        7.2  16-07-2023 06:48  ...  Sand Point, Alaska            NaN        NaN
4                  M 7.3 - Alaska Peninsula        7.3  16-07-2023 06:48  ...   Alaska Peninsula            NaN        NaN

[5 rows x 19 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 19 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   title      1000 non-null   object
 1   magnitude  1000 non-null   float64
 2   date_time  1000 non-null   object
 3   cdi        1000 non-null   int64
 4   mmi        1000 non-null   int64
 5   alert      449 non-null    object
 6   tsunami    1000 non-null   int64
 7   sig        1000 non-null   int64
 8   net        1000 non-null   object
 9   nst        1000 non-null   int64
 10  dmin       1000 non-null   float64
 11  gap        1000 non-null   float64
 12  magType    1000 non-null   object
 13  depth      1000 non-null   float64
 14  latitude   1000 non-null   float64
 15  longitude  1000 non-null   float64
 16  location   994 non-null    object
 17  continent  284 non-null    object
 18  country    651 non-null    object
```



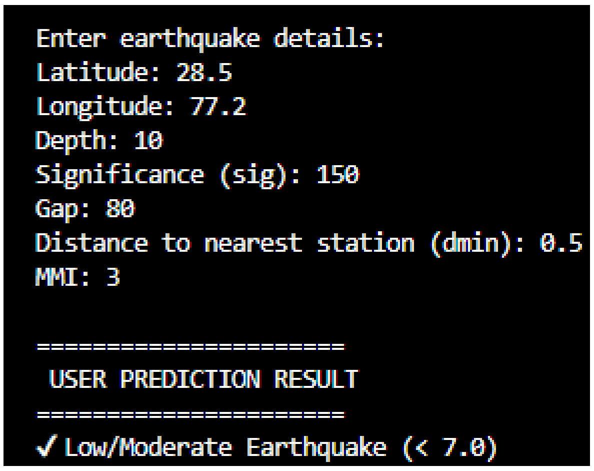
```
dtypes: float64(6), int64(5), object(8)
memory usage: 148.6+ KB
None
         magnitude          cdi         mmi      tsunami  ...         gap        depth     latitude    longitude
count  1000.000000  1000.000000  1000.00000  1000.000000  ...  1000.000000  1000.000000  1000.000000  1000.000000
mean      6.940150     3.605000     6.02700     0.325000  ...    20.926290    74.612541     4.315554    51.486576
std       0.438148     3.328972     1.43399     0.468609  ...    24.415895   130.812590    26.633320   117.478302
min       6.500000     0.000000     1.00000     0.000000  ...     0.000000     2.700000   -61.848400  -179.968000
25%       6.600000     0.000000     5.00000     0.000000  ...     0.000000    16.000000   -13.518500   -71.694450
50%       6.800000     4.000000     6.00000     0.000000  ...    18.000000    29.000000    -2.443500   107.791000
75%       7.100000     7.000000     7.00000     1.000000  ...    27.000000    55.000000    25.167250   148.364750
max       9.100000     9.000000    10.00000     1.000000  ...   239.000000   670.810000    71.631200   179.662000

[8 rows x 11 columns]
After dropping missing: (1000, 19)
label
0    627
1    373
Name: count, dtype: int64
Resampled class counts:
 label
0    627
1    627
Name: count, dtype: int64

Logistic Regression Accuracy: 0.8047808764940239
```



```
Classification Report (Logistic Regression) :
               precision    recall  f1-score   support

           0       0.77      0.87      0.82       126
           1       0.85      0.74      0.79       125

    accuracy                           0.80       251
   macro avg       0.81      0.80      0.80       251
weighted avg       0.81      0.80      0.80       251


===Confusiom Matrix:Logistic Regression===
[[110  16]
 [ 33  92]]
SVM Accuracy: 0.8725099601593626

Classification Report (SVM) :

              precision    recall  f1-score   support

           0       0.87      0.88      0.87       126
           1       0.88      0.86      0.87       125

    accuracy                           0.87       251
   macro avg       0.87      0.87      0.87       251
weighted avg       0.87      0.87      0.87       251
```



```
===Confusiom Matrix:SVM===
[[111  15]
 [ 17 108]]
Naive Bayes Accuracy: 0.7171314741035857

Classification Report (Naive Bayes) :

              precision    recall  f1-score   support

           0       0.67      0.85      0.75       126
           1       0.79      0.58      0.67       125

    accuracy                           0.72       251
   macro avg       0.73      0.72      0.71       251
weighted avg       0.73      0.72      0.71       251


===Confusiom Matrix:Naive Bayes===
[[107  19]
 [ 52  73]]

Accuracy (Random Forest): 0.9482071713147411

Classification Report (Random Forest):
               precision    recall  f1-score   support

           0       0.98      0.91      0.95       126
           1       0.92      0.98      0.95       125

    accuracy                           0.95       251
   macro avg       0.95      0.95      0.95       251
weighted avg       0.95      0.95      0.95       251
```



```
 Enter earthquake details:
 Latitude: 28.5
 Longitude: 77.2
 Depth: 10
 Significance (sig): 150
 Gap: 80
 Distance to nearest station (dmin): 0.5
 MMI: 3

 ======================
  USER PREDICTION RESULT
 ======================
 ✓ Low/Moderate Earthquake (< 7.0)
```

FUTURE ENHANCEMENT OF THE PROJECT

The Earthquake Prediction using Machine Learning project holds immense potential for future development and enhancement. While the current model provides predictive insights based on historical data, further improvements can significantly increase its accuracy, scalability, and real-world usability. The integration of emerging technologies such as deep learning, Internet of Things (IoT), and cloud computing can transform this project from a research prototype into a real-time earthquake monitoring and early-warning system.

1. Integration of Deep Learning Techniques

Future versions of this project can incorporate deep learning algorithms such as Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN).
These models are highly efficient at capturing complex patterns in sequential and spatial data, making them ideal for seismic time-series analysis.
By training on larger datasets, deep learning models can improve the precision and reliability of earthquake magnitude and frequency predictions.

2. Real-Time Data Collection Using IoT Sensors

One of the most promising enhancements is integrating IoT (Internet of Things) sensors for real-time data collection.
Seismic sensors can continuously record ground vibrations, temperature, and pressure changes.
These sensors can transmit data to the central system, where the trained machine learning model can process it instantly.
This enhancement will convert the project from a static predictive model into a live monitoring system capable of sending early warnings and alerts in high-risk areas.

3. Cloud-Based Deployment and Scalability

The project can be hosted on cloud platforms like Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure.
This will allow global access, high-speed computation, and large-scale data storage.
Cloud integration also enables real-time updates, automatic scaling, and continuous learning from incoming seismic data streams, improving both reliability and accessibility.

4. Integration with GIS (Geographical Information Systems)

Future enhancement includes linking the model with GIS mapping tools to visualize earthquake data spatially.
This integration will enable users to view predicted earthquake zones, magnitude intensities, and risk levels on interactive maps.
GIS visualization can be particularly beneficial for government authorities and disaster management agencies for planning evacuation routes and risk assessment.

5. Advanced Data Preprocessing Techniques

Further improvements in data preprocessing can be implemented using automated outlier detection, feature engineering, and data augmentation.
These enhancements will ensure that the input data is clean, balanced, and representative of various geological conditions, thereby improving the model's predictive performance.

6. Implementation of Ensemble and Hybrid Models

To achieve higher accuracy, ensemble learning techniques such as Gradient Boosting, XGBoost, and AdaBoost can be added.
These methods combine multiple algorithms to form a stronger predictive model.
Hybrid models that merge machine learning and statistical methods can further refine prediction reliability by compensating for the weaknesses of individual algorithms.

7. Development of a Mobile and Web Application

To make the system more accessible, a mobile and web-based interface can be developed.
Using frameworks like Streamlit, Flask, or React, users can input real-time parameters or view earthquake probability maps directly from their devices.
Mobile alerts or push notifications can also be implemented to deliver early warnings in earthquake-prone regions.

8. Integration with Early Warning Systems

The project can be extended to interface with government disaster management systems for automated alerts.
When a high probability of seismic activity is detected, notifications can be sent to relevant authorities or through public alert systems to ensure immediate preventive actions.

9. Incorporation of Historical and Geological Data

Future enhancements may include incorporating geological fault-line data, plate movement statistics, and historical tectonic activity records.
This multi-dimensional data will allow the model to understand deeper causes of seismic patterns, enhancing its long-term forecasting ability.

10. Implementation of Explainable AI (XAI)

Adding Explainable AI frameworks can make the predictions more transparent and interpretable.
This enhancement will help experts understand why the model made a certain prediction, increasing trust and reliability in critical applications such as public safety.

CONCLUSION OF THE PROJECT

The Earthquake Prediction using Machine Learning project represents a significant step toward the integration of artificial intelligence and geoscience to address one of the world's most unpredictable and destructive natural disasters — earthquakes. Through the use of data-driven methodologies and machine learning algorithms, this project successfully demonstrates how technology can be leveraged to analyze seismic data, identify patterns, and predict the likelihood or magnitude of future earthquakes.

Summary of the Work

The project begins with data collection from authentic sources such as the United States Geological Survey (USGS) or Kaggle. This data is then cleaned, preprocessed, and transformed into a suitable format for training predictive models. Using libraries such as Pandas, NumPy, Scikit-learn, and Matplotlib, the project performs both exploratory data analysis (EDA) and model training.

Machine learning algorithms including Decision Tree, Random Forest, Support Vector Machine (SVM), and Logistic Regression were implemented to build models that could predict earthquake occurrence and magnitude based on historical trends. The model performance was evaluated using key metrics such as accuracy, precision, recall, and F1-score to ensure reliability. Visualization tools were used to present seismic patterns and the relationships between features like depth and magnitude, making the results more interpretable.

Key Achievements

- Developed a functional predictive model capable of estimating earthquake likelihood with reasonable accuracy.
- Demonstrated the effectiveness of machine learning in environmental and geoscience applications.
- Automated the process of data analysis and prediction, making it faster and more efficient than traditional methods.
- Created visual representations of data trends that aid in decision-making and awareness.
- Showed potential for real-world deployment through integration with web frameworks like Streamlit or Flask.

Impact and Significance

This project highlights how machine learning can play a transformative role in natural disaster management and risk assessment. Although complete earthquake prediction remains a scientific challenge, the use of predictive modeling can provide probabilistic insights that help governments, researchers, and disaster management agencies take preventive measures.

By analyzing past earthquake records, the system can recognize correlations between depth, magnitude, and geographic regions, offering valuable data for disaster preparedness. The project contributes to the development of smart, data-informed disaster warning systems that can potentially save lives and minimize economic losses.

Limitations

While the project achieves promising results, it has certain limitations. The accuracy of predictions depends heavily on the quality and quantity of available data. Earthquake occurrences are influenced by multiple complex and non-linear geological factors, many of which are still not fully understood. As a result, the current model provides probability-based predictions rather than exact forecasts.

Prospects

The project opens several avenues for future research and development. By integrating real-time IoT sensors, deep learning architectures (like LSTM and CNN), and cloud-based deployment, the system can evolve into a real-time earthquake monitoring and alert platform. These advancements would significantly improve prediction accuracy, scalability, and real-world usability.

Furthermore, integration with Geographical Information Systems (GIS) and Explainable AI (XAI) will enhance the interpretability and practical application of the model, making it a powerful tool for geoscientists and disaster management authorities worldwide.

Conclusion Statement

In conclusion, the Earthquake Prediction using Machine Learning project successfully demonstrates the synergy between artificial intelligence and environmental science. It showcases how data analytics and machine learning can be harnessed for societal benefit, especially in predicting natural calamities. While absolute earthquake prediction remains beyond current technological capabilities, this project provides a foundation for future innovations in seismic forecasting and disaster resilience.

Ultimately, this work represents a meaningful step toward building smarter, safer, and more prepared communities, aligning with global goals of sustainable development and climate action.

References

- **Dataset: Kaggle.**
  Earthquake Dataset (1995–2023). Available at: https://www.kaggle.com/
- Machine Learning-Based Earthquake Prediction: Feature Engineering and Model Performance Using Synthetic Seismic Data — A. Mahmoud et al. 2025. Focuses on feature engineering in ML for earthquake prediction. naturalspublishing.com
- Geoapify Geocoding Tool — A free web tool for geocoding addresses and downloading coordinates in CSV or other formats.
- URL: https://www.geoapify.com/tools/geocoding-online/
- **GeeksforGeeks – Machine Learning Classifiers**
- https://www.geeksforgeeks.org/types-of-classification-algorithms-in-machine-learning/
- OpenAI. (2025). ChatGPT (GPT-5.1*)* https://chat.openai.com