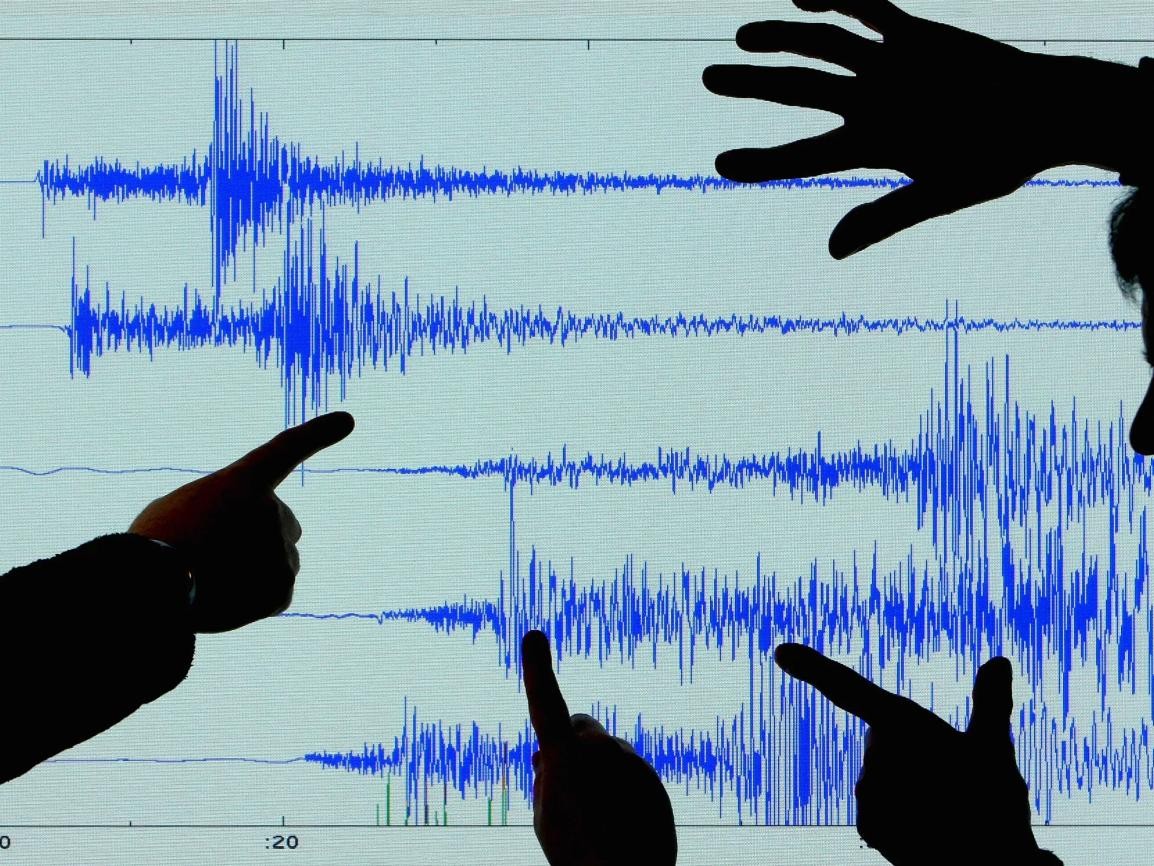
EARTHQUAKE PREDICTION MODEL USING PYTHON

Phase 3 submission

**project Title:** Earthquake prediction

**Phase 3:** Development part 1



# Introduction:

Earthquake is a natural phenomenon whose occurrence predictability is still a hot topic in academia. This is because of the destructive power it holds. In this article, we’ll learn how to analyze and visualize earthquake data with python and Matplotlib.

# Dataset:

**Necessary steps to follow:**

# Import libraries:

start by importing the necessary libraries

## program:

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score

# Load Dataset:

You can use a dataset of historical earthquake occurrences. There are various sources for such

data, including USGS (United States Geological Survey) and IRIS (Incorporated Research Institutions for Seismology). Download a suitable dataset in CSV format and load it into a Pandas DataFrame.

## program:

earthquake\_data = pd.read\_csv('earthquake\_data.csv')

# Exploratory Data Analysis(EDA):

Exploratory Data Analysis (EDA) is a crucial step in understanding and gaining insights from your earthquake prediction dataset. It involves examining the data's structure, identifying patterns, and visualizing relationships among variables.

## program:

#check for load data print(df.isnull().sum()) #Explore statistics print(df.describe())

#Visualize the data (eg. histogram,box plot ,scatter plot etc)

# Feature Engineering:

Feature engineering is a critical step in the machine learning pipeline for earthquake prediction.

It involves creating new features, transforming existing ones, and selecting the most informative variables to improve the performance of your predictive model.

def gen\_features(X): strain=[] strain.append(X.mean()) strain.append(X.std()) strain.append(X.min()) strain.append(X.kurtosis()) strain.append(X.skew())

strain.append(np.quantile(X, 0.01)) return pd.Series(strain)

dtype={'acoustic\_data': np.int16, 'time\_to\_failure': np.float64}) train = pd.read\_csv('train.csv', iterator=True, chunksize=150\_000, X\_train = pd.DataFrame()

y\_train = pd.Series() for df in train:

ch = gen\_features(df['acoustic\_data'])

X\_train = X\_train.append(ch, ignore\_index=True)

y\_train = y\_train.append(pd.Series(df['time\_to\_failure'].values[-1]))

# 5.Split the Data:

Split your dataset into training and testing sets to evaluate your model's performance.

# Progarm:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 6.Feature scaling:

Feature scaling is an important preprocessing step in many machine learning algorithms, as it helps ensure that features are on a similar scale, preventing some features from dominating the learning process. Feature scaling is particularly important when working with numerical data in earthquake prediction. Common scaling techniques include Min-Max scaling (normalization) and Standardization

(Z-score scaling).

# Program:

scaler=StandardScaler() X\_train=scaler.fit\_transform(X\_train) X\_test=scaler.transform(X\_test)

# Importance of Loading and processsing dataset:

Loading and processing the dataset is the foundation upon which the entire earthquake prediction process is built. It is critical for data quality, feature engineering, model training, evaluation, and model deployment. Careful and thoughtful data processing is essential for building effective and reliable earthquake prediction models.

# Challenges involved in loading and preprocessing Earthquake prediction dataset:

Some of these challenges are specific to the nature of earthquake data and the tools you're using.Here are the key challenges:

**Data Volume**: Earthquake datasets can be vast, and loading large datasets into memory can be

memory-intensive. You may need to consider using techniques like chunking or distributed computing to handle large datasets.

**Data Formats**: Earthquake data can come in various formats, such as CSV, JSON, or specialized formats used by seismological organizations. Reading and parsing these formats can be tricky, especially if they're not well-documented.

**Imbalanced Data**: Earthquake prediction datasets are typically imbalanced, with far more

non-earthquake instances than earthquake instances. Handling this imbalance while training a predictive model is a challenge.

**Spatial Autocorrelation**: Earthquakes tend to exhibit spatial autocorrelation, meaning that the occurrence of one earthquake can affect the likelihood of others in the same region. Accounting for this correlation can be challenging.

**Noise and Uncertainty**: Seismic data may contain noise or uncertainty due to various sources, such as instrument error or local disturbances. Distinguishing true seismic events from noise is crucial.

**Data Labeling**: Properly labeling data as earthquake occurrences or non-occurrences is challenging, particularly when dealing with small-magnitude earthquakes, foreshocks, or aftershocks. Mislabeling can lead to model biases.

**Specialized Tools**: Some tasks in earthquake prediction may require specialized tools or libraries for geospatial analysis, time-series analysis, and seismology. Integrating these tools with Python can be challenging.

**Computational Resources**: Preprocessing and modeling can be computationally intensive, especially when working with large datasets. Access to powerful computing resources may be necessary.

**Ethical Considerations**: The use of earthquake prediction models has ethical considerations, as inaccuracies in predictions can have severe consequences. Ensuring the ethical and responsible use of models is essential.

# How to overcome the challenges of loading and preprocessing a earthquake prediction Datset:

Overcoming the challenges of loading and preprocessing an earthquake prediction dataset using Python requires careful planning, appropriate tools, and specific techniques tailored to the nature of the data.

**Data Quality Assurance**:Conduct thorough data quality checks to identify and address missing values, outliers, and inconsistencies. Tools like Pandas in Python are helpful for data cleaning.

**Documentation and Version Control**:Document your preprocessing steps, code, and decisions thoroughly.Use version control tools like Git to track changes and collaborate with team members.

**Education and Collaboration**:Continuously educate yourself and collaborate with experts in seismology, geophysics, and data science to gain insights and overcome challenges.

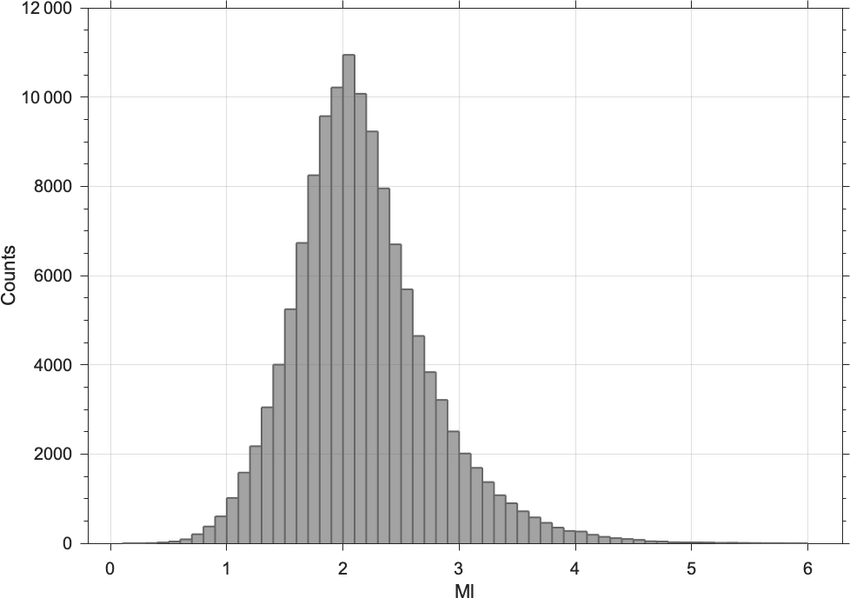
**Test and Validate**:Test preprocessing steps on small subsets of data to validate that they work as expected before applying them to the entire dataset.Cross-validate models to ensure their robustness and performance.

# visualization and preprocessing data:

earthquake\_data['magnitude'].hist(bins=2000) plt.xlabel('Magnitude')

plt.ylabel('Frequency')

plt.title('Histogram of Earthquake Magnitude') plt.show()



# Box plot:

sns.boxplot(x='will\_occur', y='depth', data=earthquake\_data) plt.xlabel('Earthquake Occurrence')

plt.ylabel('Depth')

plt.title('Depth vs. Earthquake Occurrence') plt.show()

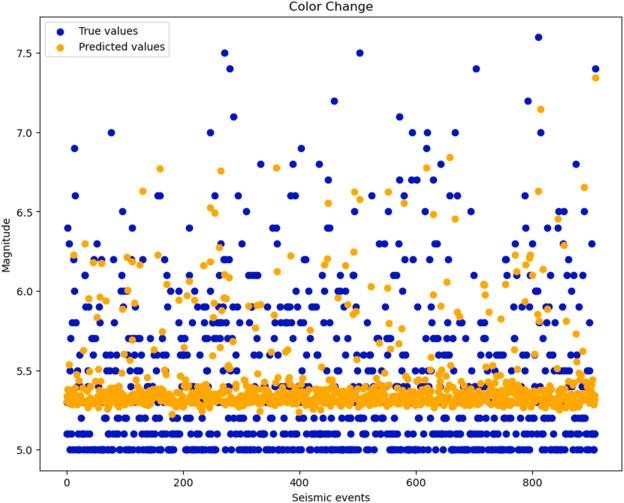


# Scatter Plot:

plt.scatter(earthquake\_data['longitude'], earthquake\_data['latitude],c=earthquake\_data['will\_occur'], cmap='coolwarm')

plt.xlabel('Longitude') plt.ylabel('Latitude')

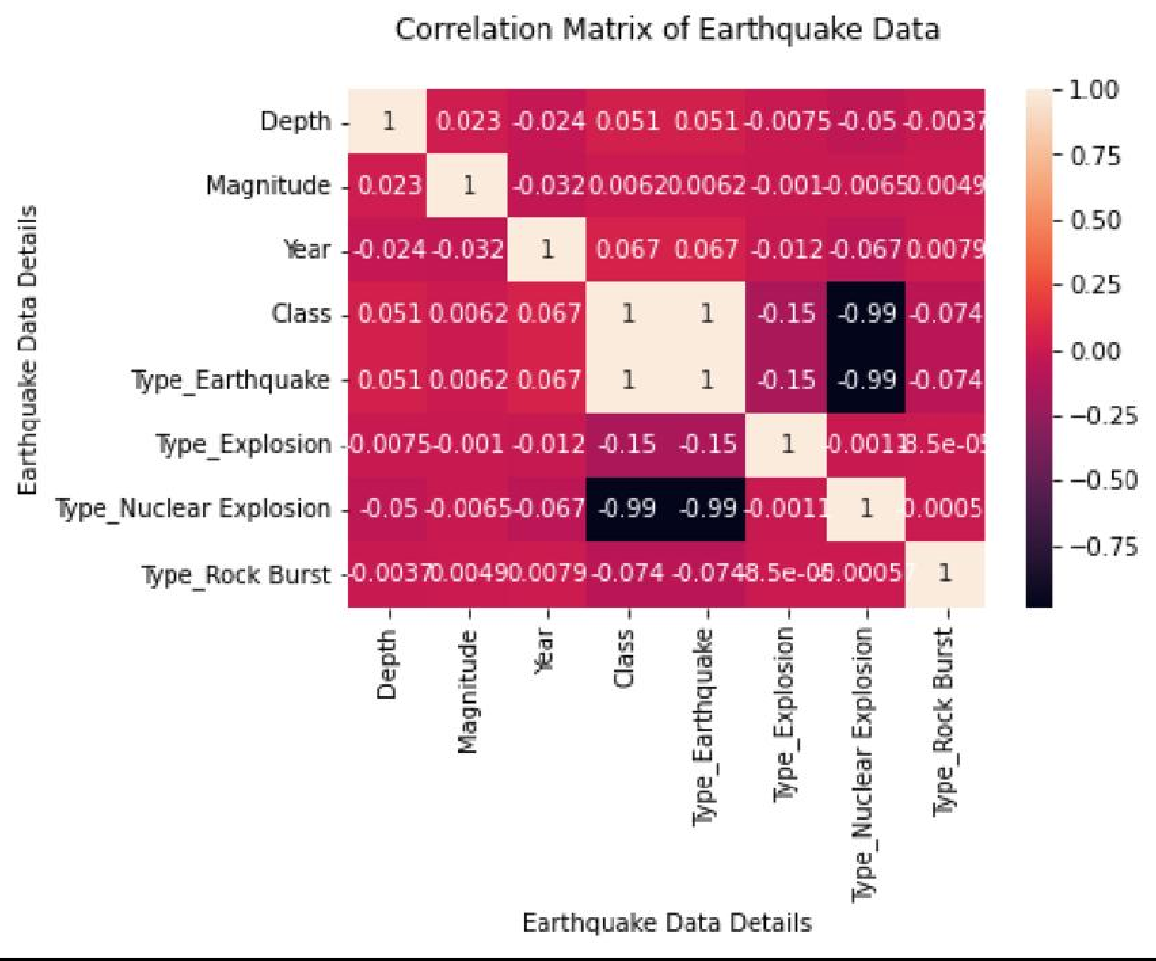
plt.title('Earthquake Occurrence on Geospatial Coordinates') plt.show()



# visualization correlation:

correlation\_matrix = earthquake\_data.corr() sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm') plt.title('Correlation Matrix')

plt.show()



**Program:**

import numpy as np

from import pandas as pd sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score

# step 1:Load the dataset

earthquake\_data = pd.read\_csv('earthquake\_data.csv')

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | latitude | longitude | depth | magnitude | timestamp |
| 0 | 19.246 | 145.616 | 131.6 | 6 | -1.58E+08 |
| 1 | 1.863 | 127.352 | 80 | 5.8 | -1.57E+08 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | -20.579 | -173.972 | 20 | 6.2 | -1.57E+08 |
| 3 | -59.076 | -23.557 | 15 | 5.8 | -1.57E+08 |
| 4 | 11.938 | 126.427 | 15 | 5.8 | -1.57E+08 |

# Step 2:Explore data analysis

train = pd.read\_csv('train.csv', nrows=6000000, dtype={'acoustic\_data' : np.int16, 'time\_to\_failure':np.float64})

# Step 3:Feature Engineering

def gen\_features(X):

strain=[] strain.append(X.mean()) strain.append(X.std()) strain.append(X.min()) strain.append(X.kurtosis()) strain.append(X.skew())

strain.append(np.quantile(X, 0.01)) return pd.Series(strain)

dtype={'acoustic\_data': np.int16, 'time\_to\_failure': np.float64}) train = pd.read\_csv('train.csv', iterator=True, chunksize=150\_000, X\_train = pd.DataFrame()

y\_train = pd.Series() for df in train:

ch = gen\_features(df['acoustic\_data'])

X\_train = X\_train.append(ch, ignore\_index=True)

y\_train = y\_train.append(pd.Series(df['time\_to\_failure'].values[-1]))

# Step 4:Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Output:

**Exploratory Data Analysis:**

|  |  |  |
| --- | --- | --- |
| 0 | 12 | 1.4691 |
| 1 | 6 | 1.4691 |
| 2 | 8 | 1.4681 |
| 3 | 5 | 1.4691 |
| 4 | 8 | 1.4691 |
| 5 | 8 | 1.4691 |
| 6 | 9 | 1.4691 |
| 7 | 7 | 1.4691 |
| 8 | 5 | 1.4691 |
| 9 | 3 | 1.4691 |

# Feature Engineering:

