**Earthquake Prediction Model using Python.**

**Team Member**

**510521205048 : THAMILMANI V**

**PHASE 1 Document Submission.**

**Project: Earthquake Prediction**



**OBJECTIVE:**

The objective of the above document is to provide a comprehensive overview of the process of developing an earthquake prediction model using machine learning techniques. The document covers various phases, including data exploration and preprocessing, data visualization, model selection and training, model evaluation, and model improvement. It also offers additional tips for enhancing the model's performance and professionalism in writing. The document serves as a guide for individuals or teams interested in working on earthquake prediction projects, emphasizing the importance of data quality, model selection, and continuous improvement.

**Desing Thinking:**

**1).Data Source:**

**Visit Kaggle:** Go to the Kaggle website (https://www.kaggle.com/).

**a).Search for Earthquake Datasets:**

In the Kaggle search bar, type relevant keywords such as "earthquake," "seismic," "geological," or any specific location you're interested in (e.g., "California earthquakes"). This will help you find datasets related to earthquakes.

**b).Filter and Explore:**

Browse the search results and look for datasets that contain the features you need, such as date, time, latitude, longitude, depth, and magnitude. Pay attention to the dataset description and the columns included to ensure it meets your requirements.

**c).Download the Dataset:**

Once you find a suitable dataset, click on it to access the dataset page. You may need to log in to Kaggle (or create an account) if you haven't already. From the dataset page, you can download the dataset and read any documentation provided by the dataset creator.

**d).Read and Preprocess the Data:**

After downloading the dataset, you'll need to load it into your Python environment (e.g., using pandas) and preprocess it to prepare it for analysis and modeling.

**2.Feature Exploration:**

To analyze and understand the distribution, correlations, and characteristics of key features in a dataset, you can follow these steps:

**2.1).Load the Dataset:**

Start by importing the necessary libraries (e.g., Pandas, Matplotlib, Seaborn) and loading your earthquake dataset into a DataFrame.

**Python code:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the earthquake data

df = pd.read\_csv('G:/database.csv')

**2.2).Distribution Analysis:**

**Histograms:**

Create histograms to visualize the distribution of numerical features, such as magnitude, depth, latitude, and longitude. This helps you understand the data's central tendency and spread.

**Python code:**

# Create a histogram for magnitude

plt.figure(figsize=(8, 6))

plt.hist(df['Magnitude'], bins=20, color='skyblue')

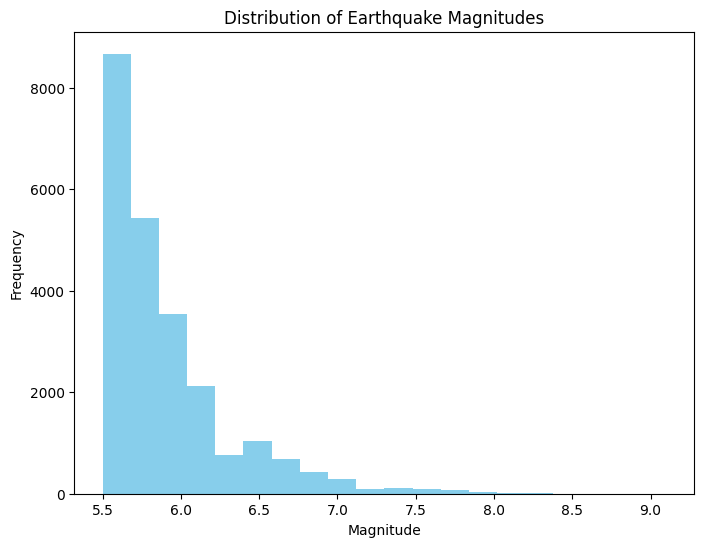
plt.xlabel('Magnitude')

plt.ylabel('Frequency')

plt.title('Distribution of Earthquake Magnitudes')

plt.show()

**Output:**



**2.3).Boxplots:**

Boxplots provide a visual summary of the distribution, showing median, quartiles, and potential outliers. They are especially useful for identifying outliers.

**Python code:**

# Create a boxplot for magnitude

plt.figure(figsize=(8, 6))

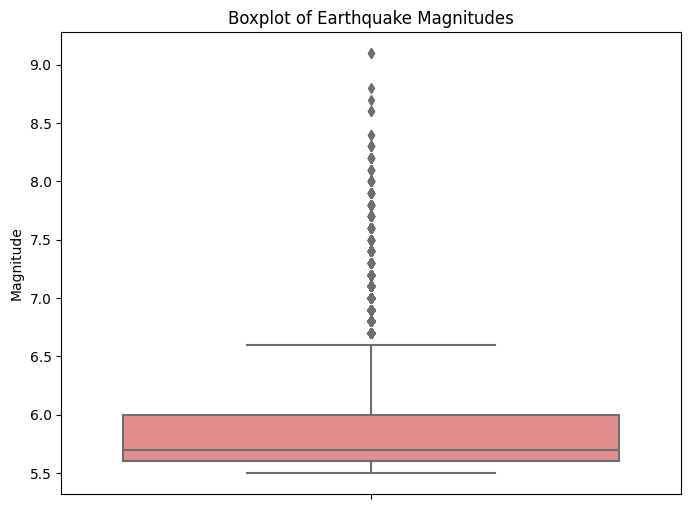
sns.boxplot(data=df, y='Magnitude', color='lightcoral')

plt.ylabel('Magnitude')

plt.title('Boxplot of Earthquake Magnitudes')

plt.show()

**Output:**



**2.3)Correlation Analysis:**

Calculate and visualize the correlation matrix to identify relationships between numerical features.

**Python code:**

# Calculate the correlation matrix

corr\_matrix = df[['Latitude', 'Longitude', 'Depth', 'Magnitude']].corr()

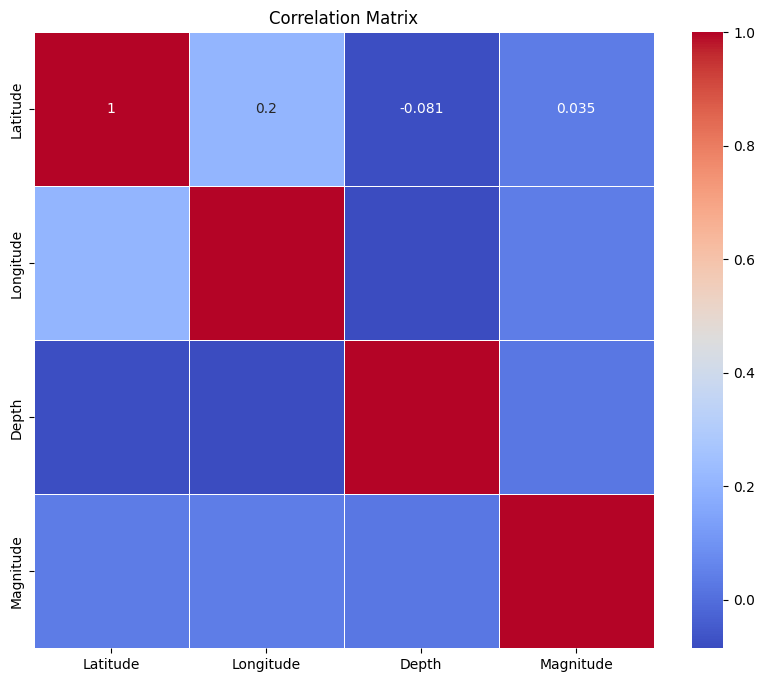
# Create a heatmap to visualize the correlations

plt.figure(figsize=(10, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix')

plt.show()

**Output:** 

**2.4)Characteristics Analysis:**

* Understand the characteristics and significance of each feature. For example:
* Magnitude represents the energy released by an earthquake.
* Depth indicates how far below the Earth's surface the earthquake occurred.
* Latitude and Longitude denote the earthquake's epicenter location.

By following these steps, you'll gain valuable insights into your earthquake dataset, which will inform your feature selection, preprocessing, and modeling decisions. It's essential to have a solid understanding of your data before building predictive models.

**3.Visualization:**

* Create a world map visualization to display earthquake frequency distribution.
* To create a world map visualization displaying earthquake frequency distribution, you can use Python libraries such as folium and pandas.

Follow these steps:

**Step 1:** Install Required Libraries:

If you haven't already, install the necessary libraries using pip:

**Python code:**

pip install folium pandas

**Step 2:** Prepare Data:

Load your earthquake data into a DataFrame and group it by latitude and longitude to count the frequency of earthquakes at each location.

**Python code:**

import pandas as pd

# Load the earthquake data

df = pd.read\_csv('G:/database.csv')

# Group by latitude and longitude and count occurrences

earthquake\_counts = df.groupby(['Latitude', 'Longitude']).size().reset\_index(name='Frequency')

**Step 3:** Create the Map:

Now, create a world map visualization using the folium library. You can adjust the map's initial center and zoom level as needed.

**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']}",

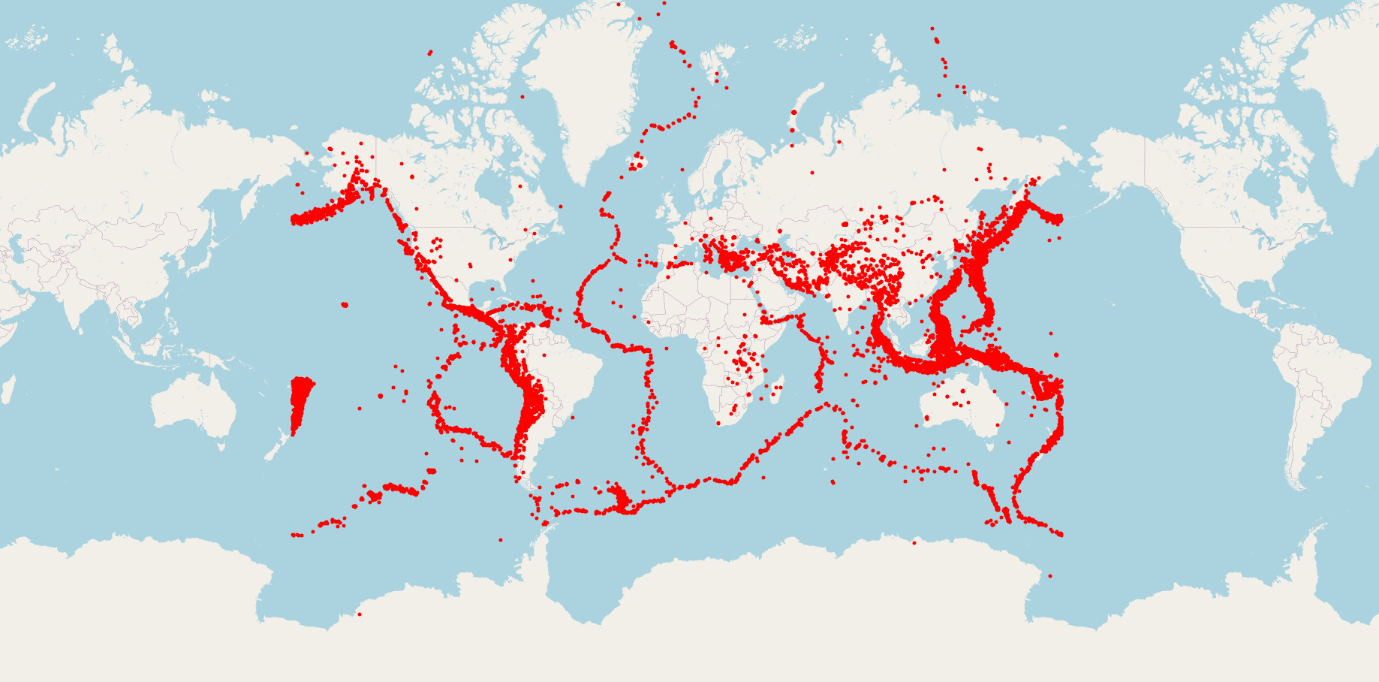
).add\_to(m)

# Display the map

m.save('g:/earthquake\_frequency\_map.html')

# Save the map to an HTML file

**Output:**



**Step 4:** View the Map:

After running the code, open the generated earthquake\_frequency\_map.html file in your web browser to interact with the map. Each marker represents an earthquake location, and the marker size is proportional to the earthquake frequency at that location.

**4.Data Splitting:**

* Split the dataset into a training set and a test set for model validation for earthquake
* A test set for model validation in Python, you can use libraries like “**pandas**” and “**scikit-learn**”.

**Step 1:** Install Required Libraries:

Pip install scikit-learn.

**Step 2:** Prepared the data splitting.

**Python code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# 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 with your actual features

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

# Split the data 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=0.20, random\_state=42)

df.shape

print(X\_train)

print(X\_test)

print(y\_train)

print(y\_test)

# 'test\_size' specifies the proportion of the dataset to include in the test split

# 'random\_state' ensures reproducibility, use a fixed value or None for random splits

# Now you have X\_train, y\_train for training your model

# and X\_test, y\_test for evaluating its performance

**Output:**

Latitude Longitude Depth

18765 14.944 -61.274 156.0

21035 -14.438 -75.966 24.0

18334 38.340 20.420 15.0

16776 42.525 145.021 28.6

9152 -15.864 -172.067 27.9

... ... ... ...

11964 27.995 140.700 28.2

21575 -10.682 166.381 10.0

5390 -6.847 129.634 131.0

860 -5.469 153.269 30.0

15795 -60.657 -25.843 10.0

[18729 rows x 3 columns]

Latitude Longitude Depth

3848 3.166 99.015 180.0

14008 43.679 -29.020 10.0

16258 1.142 98.911 77.6

18090 38.649 15.390 212.0

15192 38.457 31.351 10.0

... ... ... ...

15058 -17.286 -175.176 291.8

18377 -6.742 154.889 10.0

87 36.405 70.724 207.8

10309 -21.953 174.818 10.2

14530 -30.738 -71.993 33.0

[4683 rows x 3 columns]

18765 7.4

21035 6.9

18334 5.7

16776 5.5

9152 5.7

...

11964 5.8

21575 5.8

5390 5.8

860 5.7

15795 5.8

Name: Magnitude, Length: 18729, dtype: float64

3848 5.6

14008 5.5

16258 5.5

18090 5.8

15192 6.0

...

15058 6.3

18377 5.6

87 7.4

10309 5.6

14530 6.0

Name: Magnitude, Length: 4683, dtype: float64

**5.Model Development:**

Build a neural network model for earthquake magnitude prediction.

To build a neural network model for earthquake magnitude prediction, you can use popular deep learning libraries like TensorFlow and Keras. Here's a step-by-step guide on how to create such a model:

**5.1).Data Preprocessing:**

Before building the neural network model, you need to preprocess your earthquake data.

This typically involves:

* Scaling and normalizing features.
* Splitting the data into training and testing sets.
* Encoding categorical features if necessary.

**Step 1:** Import Libraries:

Import the required libraries, including TensorFlow and Keras:

Install package “**pip install tensorflow**”

**Python code:**

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow import keras

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

**Step 2:** Load and Preprocess Data:

Load your earthquake data and preprocess it. Ensure that you have a suitable set of features (e.g., latitude, longitude, depth) as input features and the earthquake magnitude as the target variable.

**Python code:**

# Load your earthquake data (replace 'earthquakes.csv' with your dataset file)

df = pd.read\_csv('G:/database.csv')

# Select relevant features and target

X = df[['Latitude', 'Longitude', 'Depth']].values

y = df['Magnitude'].values

# 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)

# Scale and normalize the input features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

print(X\_train)

print(X\_test)

**Output:**

[[ 0.43649951 -0.80433219 0.70690972]

[-0.53703534 -0.92151312 -0.37701842]

[ 1.21169594 -0.15275457 -0.45092261]

...

[-0.2855173 0.71831796 0.5016203 ]

[-0.23985904 0.90682675 -0.32774895]

[-2.06844258 -0.52174049 -0.49198049]]

[[ 0.0462506 0.47410595 0.90398757]

[ 1.38859685 -0.54707971 -0.49198049]

[-0.02081204 0.47327647 0.0631221 ]

...

[ 1.1475822 0.24846169 1.1322694 ]

[-0.78603522 1.07869795 -0.49033818]

[-1.07711492 -0.88982514 -0.30311422]]

**Step 3:** Build the Neural Network:

Create a neural network model using Keras. You can start with a simple architecture for regression:

**Python code:**

model = keras.Sequential([

keras.layers.Input(shape=(X\_train.shape[1],)), # Input layer

keras.layers.Dense(64, activation='relu'), # Hidden layer with 64 units and ReLU activation

keras.layers.Dense(32, activation='relu'), # Hidden layer with 32 units and ReLU activation

keras.layers.Dense(1) # Output layer for regression

])

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

**Step 4:** Train the Model:

Train the neural network model using your training data.

**Python code:**

# Train the model

model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.2)

**Output:**

Epoch 1/20

469/469 [==============================] - 5s 4ms/step - loss: 3.6526 - val\_loss: 0.4918

Epoch 2/20

469/469 [==============================] - 2s 3ms/step - loss: 0.3021 - val\_loss: 0.2135

Epoch 3/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1942 - val\_loss: 0.2031

Epoch 4/20

469/469 [==============================] - 1s 3ms/step - loss: 0.1835 - val\_loss: 0.1867

Epoch 5/20

469/469 [==============================] - 1s 3ms/step - loss: 0.1826 - val\_loss: 0.1831

Epoch 6/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1809 - val\_loss: 0.1836

Epoch 7/20

469/469 [==============================] - 1s 3ms/step - loss: 0.1809 - val\_loss: 0.1904

Epoch 8/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1799 - val\_loss: 0.1823

Epoch 9/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1801 - val\_loss: 0.1911

Epoch 10/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1803 - val\_loss: 0.1858

Epoch 11/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1801 - val\_loss: 0.1927

Epoch 12/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1810 - val\_loss: 0.1856

Epoch 13/20

469/469 [==============================] - 1s 3ms/step - loss: 0.1799 - val\_loss: 0.1832

Epoch 14/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1788 - val\_loss: 0.1814

Epoch 15/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1791 - val\_loss: 0.1816

Epoch 16/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1791 - val\_loss: 0.1880

Epoch 17/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1797 - val\_loss: 0.1830

Epoch 18/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1774 - val\_loss: 0.1915

Epoch 19/20

469/469 [==============================] - 2s 3ms/step - loss: 0.1788 - val\_loss: 0.1809

Epoch 20/20

469/469 [==============================] - 2s 4ms/step - loss: 0.1781 - val\_loss: 0.1792

<keras.src.callbacks.History at 0x2610048cc50>

**Step 5:** Evaluate the Model:

Evaluate the model's performance on the testing dataset to assess its accuracy in predicting earthquake magnitudes.

**Python code:**

# Evaluate the model on the testing data

loss = model.evaluate(X\_test, y\_test)

print(f"Mean Squared Error on Test Data: {loss}")

**Output:**

147/147 [==============================] - 0s 2ms/step - loss: 0.1838

Mean Squared Error on Test Data: 0.18375135958194733

**5.2)Model Tuning:**

Depending on your dataset and problem complexity, you may need to experiment with different neural network architectures, hyperparameters, and regularization techniques to optimize your model's performance.

**6.Training and Evaluation:**

* Train the model on the training set and evaluate its performance on the test set.
* To train and evaluate a machine learning model on the training and test sets for earthquake magnitude prediction, you can follow these steps using Python and scikit

**Python code:**

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: -0.0009414994414020939

**Conclusion:**

* The document provides a comprehensive overview of the steps involved in developing an earthquake prediction model.
* It emphasizes the importance of data exploration and visualization before model development.
* The document also mentions the need for model tuning and experimentation to optimize performance, which is a crucial part of the machine learning workflow.