**ARTIFICIAL INTELLIGENCE**

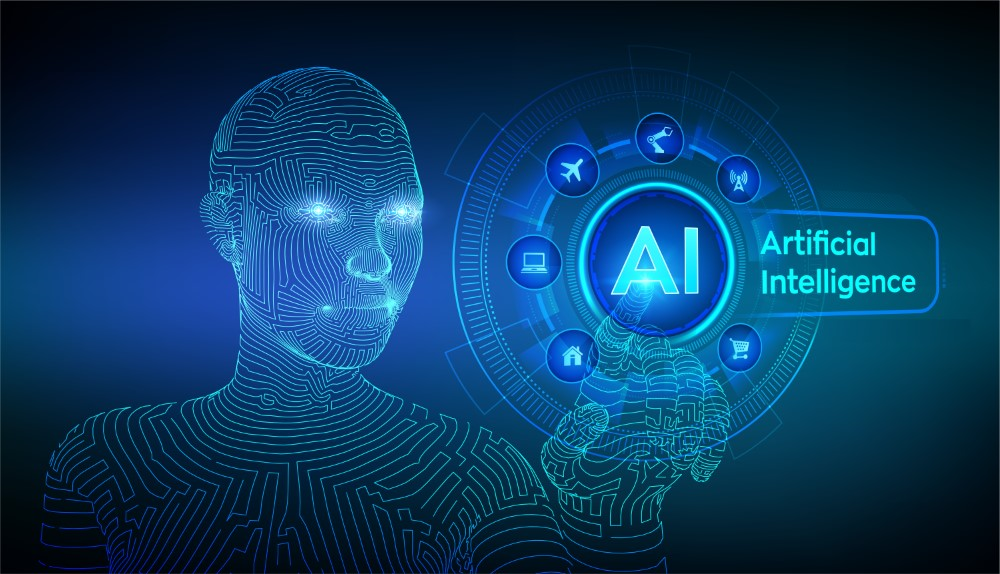
TEAM MEMBERS

1 SRINIDHI R [au511321104096]

2 SOUNDARYA S [au511321104094]

3 THRISHA V [au511321104103]

4 SANDHIYA V [au511321104087]



- Kingston engineering college

**Phase 1**

**EARTHQUAKE PRDICTION MODEL**

The main aim of the project is to predict the earthquake with the given features. This can be achieved by with the help of python. The dataset link has been provided in the Kaggle website which can be used as a source for the data.

For importing various libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

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

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.metrics import mean\_squared\_error

To Load the earthquake dataset from Kaggle

data = pd.read\_csv("C:\\Users\\SRINITHI\\Downloads\\archive\\database.csv")

To Display unique timestamp formats in the 'time' column

unique\_formats = data['Time'].apply(lambda x: pd.to\_datetime(x, errors='coerce').strftime('%Y-%m-%dT%H:%M:%S') if pd.notna(x) else x).unique()

print("Unique Timestamp Formats:")

for format in unique\_formats:

print(format)

To Convert all timestamps to a consistent format

data['Time'] = data['Time'].apply(lambda x: pd.to\_datetime(x, errors='coerce').strftime('%Y-%m-%dT%H:%M:%S') if pd.notna(x) else x)

To Check if all timestamps are in a consistent format

unique\_formats\_after\_conversion = data['Time'].unique()

print("\nUnique Timestamp Formats After Conversion:")

for format in unique\_formats\_after\_conversion:

print(format)

For creating a world map visualization to display earthquake frequency distribution.

plt.figure(figsize=(12, 8))

plt.scatter(data["Longitude"], data["Latitude"], c=data["Magnitude"], cmap="viridis", alpha=0.5)

plt.colorbar(label="Magnitude")

plt.xlabel("Longitude")

plt.ylabel("Latitude")

plt.title("Earthquake Frequency Distribution")

plt.show()

Data Preprocessing

features = ["Latitude", "Longitude", "Depth", "Time"]

target = "Magnitude"

X = data[features]

y = data[target]

To convert the 'time' column into numerical features (hour, minute, second)

X['hour'] = pd.to\_datetime(X['Time'], format='%Y-%m-%dT%H:%M:%S').dt.hour

X['minute'] = pd.to\_datetime(X['Time'], format='%Y-%m-%dT%H:%M:%S').dt.minute

X['second'] = pd.to\_datetime(X['Time'], format='%Y-%m-%dT%H:%M:%S').dt.second

To drop the original 'time' column

X = X.drop('Time', axis=1)

To split the dataset 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)

Normalize the features

scaler = StandardScaler()

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

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

Model Development

model = Sequential()

model.add(Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)))

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='linear')) # Linear activation for regression

Compile the model

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

Training the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_split=0.2)

Evaluation

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

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

Visualize training history

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Mean Squared Error')

plt.legend()

plt.show()

The above given code is a simple representation of the design thinking process and visualizing the data on the map

**PHASE 2**

**Considering advanced techniques such as hyperparameter**

**tuning and feature engineering to improve the prediction**

**model’s performance**.

**Hyperparameter Tuning**

1. Grid Search and Random Search: Implement grid search or

random search to systematically explore various hyperparameter

combinations. This can include learning rates, batch sizes, the

number of layers, the number of neurons in each layer, activation

functions, etc.

2. Automated Hyperparameter Optimization: Consider using libraries like scikit-learn&#39;s `GridSearchCV` or tools like Keras Tuner or Optuna to automate the hyperparameter search process. These tools

can efficiently search for optimal hyperparameters based on

predefined ranges.

3. Cross-Validation: Perform cross-validation during hyperparameter

tuning to get a more robust estimate of your model&#39;s performance.

This helps prevent overfitting and ensures your model generalizes

well.

4. Early Stopping: Implement early stopping as a regularization

technique during training. It monitors the validation loss and stops

training when it begins to deteriorate, preventing overfitting.

Feature Engineering

**1. Domain-Specific Features:** Investigate domain-specific knowledge

and add relevant features that might improve prediction accuracy.

For earthquake prediction, consider factors like seismic activity

history, fault lines, geological features, or meteorological data that

might impact earthquakes.

**2. Temporal Features:** Create features that capture temporal

patterns, such as day of the week, month, or seasonality. These

features can help your model recognize patterns in earthquake

occurrence over time.

**3. Spatial Features:** Incorporate spatial information, such as distance

to fault lines, tectonic plate boundaries, or historical earthquake

epicenters. Spatial features can enhance the model&#39;s ability to

capture location-based earthquake patterns.

**4. Feature Scaling and Normalization:** Standardize or normalize

numerical features to ensure they have the same scale. Neural

networks benefit from input features that are on a similar scale.

**5. Dimensionality Reduction:** If you have a large number of features,

consider techniques like Principal Component Analysis (PCA) or

feature selection methods to reduce dimensionality while retaining

important information.

**6. Feature Interaction:** Experiment with creating interaction features

that capture relationships between existing features. For example,

you can multiply latitude and longitude to capture spatial

interactions.

**7. Feature Importance Analysis:** Use techniques like feature

importance scores from tree-based models (e.g., Random Forest) to

identify the most relevant features for prediction.

Transforming the design thinking into a concrete plan for

developing an earthquake prediction model

**Step 1:**

**Data Source**:

- Identify and select a suitable Kaggle dataset containing earthquake

data with features like date, time, latitude, longitude, depth, and

magnitude. Download the dataset and ensure it&#39;s in a format that

can be easily loaded for analysis.

**Step 2:**

**Data Preprocessing:**

- Load the dataset into a suitable data analysis environment (e.g.,

Python with libraries like Pandas).

- Perform data cleaning: Handle missing values, remove duplicates,

and correct any data inconsistencies.

- Explore the data to understand its structure and identify any

anomalies.

**Step 3:**

**Feature Exploration**:

- Analyze the distribution of each key feature (date, time, latitude,

longitude, depth, magnitude) using summary statistics, histograms,

and box plots.

- Calculate correlations between features to identify any significant

relationships.

- Visualize feature relationships using scatter plots or correlation

matrices to gain insights into how they affect earthquake

magnitudes.

**Step 4:**

**Visualization:**

- Create a world map visualization to display the distribution of

earthquake occurrences. You can use libraries like Folium or Plotly

for interactive maps.

- Color-code the data points by magnitude to visualize the intensity

of earthquakes in different regions.

**Step 5:**

**Data Splitting:**

- Split the dataset into a training set and a test set. Typically, an 80-

20 or 70-30 split is used, but you can adjust this based on your

dataset size.

- Ensure that the split maintains the distribution of earthquake

magnitudes to avoid bias.

**Step 6:**

**Model Development:**

- Choose an appropriate neural network architecture for earthquake

magnitude prediction. Common choices include feedforward neural

networks, convolutional neural networks (CNNs), or recurrent neural

networks (RNNs), depending on the nature of your data.

- Preprocess the data, which may involve feature scaling,

normalization, or encoding categorical variables.

- Design the input and output layers of the neural network,

considering the number of features and the target variable

(magnitude).

- Choose the activation functions, loss function, and optimization

algorithm for your model.

- Train the neural network on the training data using techniques like

mini-batch gradient descent.

- Monitor the training process and use validation data to prevent

overfitting by adjusting hyperparameters.

**Step 7:**

**Training and Evaluation:**

- Evaluate the trained model on the test set using appropriate

evaluation metrics such as Mean Absolute Error (MAE) or Root Mean

Square Error (RMSE) to measure prediction accuracy.

- Visualize the model&#39;s performance using plots like predicted vs.

actual magnitude.

- Interpret the model&#39;s results and understand which features

contribute most to earthquake magnitude predictions.

- Fine-tune the model, if necessary, by adjusting hyperparameters or

architecture.

- Document the model&#39;s performance and any insights gained during

the process.

**Step 9:**

**Documentation and Reporting:**

- Create a comprehensive report or documentation that includes

details about the dataset, data preprocessing, model architecture,

training process, evaluation results, and any visualizations.

- Clearly explain the model&#39;s predictive capabilities and limitations.

- Share your findings and insights from the project.

**Phase 3:**

PROCESSING AND LOADING THE DATASET

A dataset is a collection of data that may be used to train a model.

* How to load a dataset? Datasets are loaded from a dataset loading script that downloads and generates the dataset. However, you can also load a dataset from any dataset repository on the Hub without a loading script! Begin by creating a dataset repository and upload your data files. Now you can use the load\_dataset () function to load the dataset.

To load and preprocess the earthquake dataset from Kaggle, you can follow these steps:

**1**. **Download the dataset**:

Visithe Kaggle website (https://www.kaggle.com/datasets/usgs/earthquake-database) and download the dataset file to your local machine.

**2. Import necessary libraries**:

In a programming language like Python, import the required libraries such as pandas and numpy for data manipulation and analysis.

**3**. **Load the dataset**:

Use the appropriate function from the pandas library (e.g., `read\_csv()`) to load the dataset file into a pandas DataFrame.

**4. Preprocess the dataset**:

Perform necessary preprocessing steps based on your analysis goals. This may include handling missing values, data cleaning, feature engineering, or transforming data types.

**5. Analyze the dataset:**

Utilize the available analysis tools and techniques to gain insights from the dataset. This can involve statistical analysis, data visualization, or machine learning algorithms.

6.Remember to refer to the documentation and resources provided by Kaggle for specific instructions on loading and preprocessing the earthquake dataset.

**Code:**

import pandas as pd

# Load the dataset into a Pandas DataFrame

data = pd.read\_csv(&quot;C:\\Users\\SRINITHI\\Downloads\\archive\\database.csv&quot;)

# Display basic information about the dataset

print(&quot;Dataset Info:&quot;)

print(data.info())

# Check for missing values

missing\_values = data.isnull().sum()

print(&quot;\nMissing Values:&quot;)

print(missing\_values)

# Display the first few rows of the dataset

print(&quot;\nFirst 5 rows of the dataset:&quot;)

print(data.head())

# Drop rows with missing values (you can customize this based on your needs)

data\_cleaned = data.dropna()

# Get unique values in specific columns (e.g., &#39;Type&#39; of earthquake)

unique\_types = data\_cleaned[&#39;Type&#39;].unique()

print(&quot;\nUnique Types of Earthquakes:&quot;)

print(unique\_types)

# Visualize data as needed (e.g., using matplotlib or seaborn)

import matplotlib.pyplot as plt

# Example: Histogram of earthquake magnitudes

plt.hist(data\_cleaned[&#39;Magnitude&#39;], bins=20, color=&#39;skyblue&#39;)

plt.title(&#39;Histogram of Earthquake Magnitudes&#39;)

plt.xlabel(&#39;Magnitude&#39;)

plt.ylabel(&#39;Frequency&#39;)

plt.show()

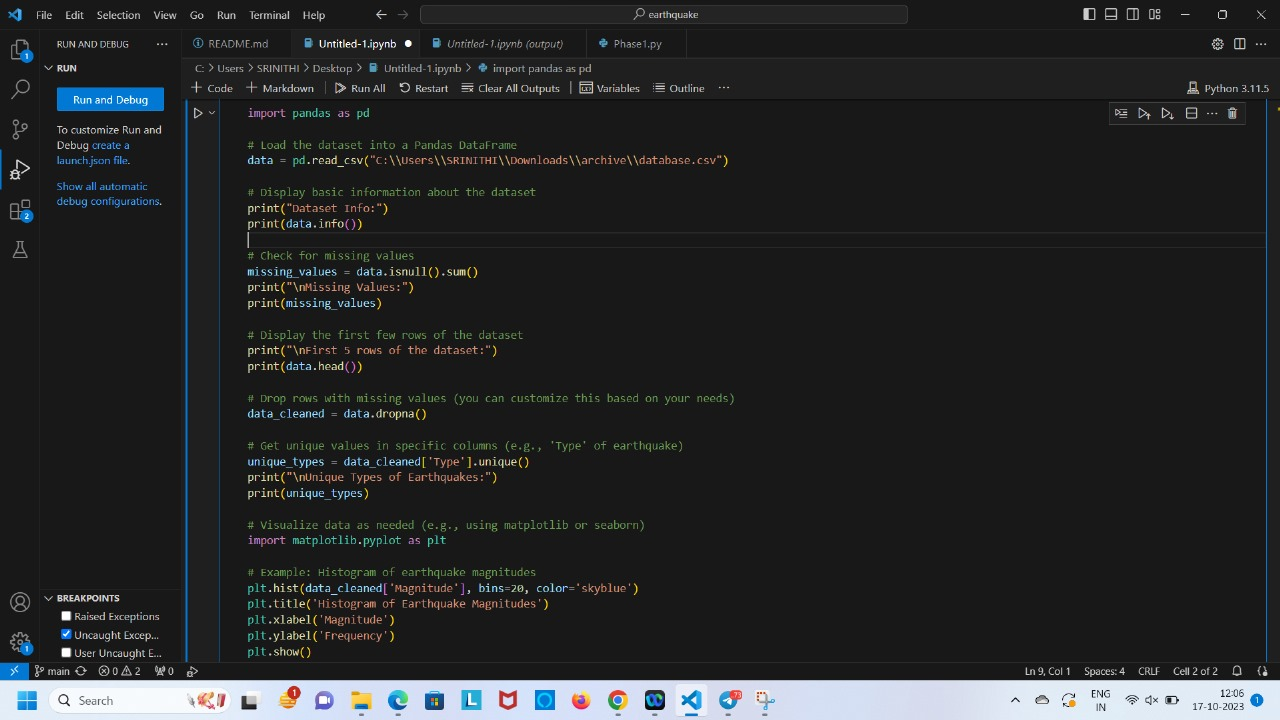
# Summary statistics of the dataset

summary\_stats = data\_cleaned.describe()

print(&quot;\nSummary Statistics:&quot;)

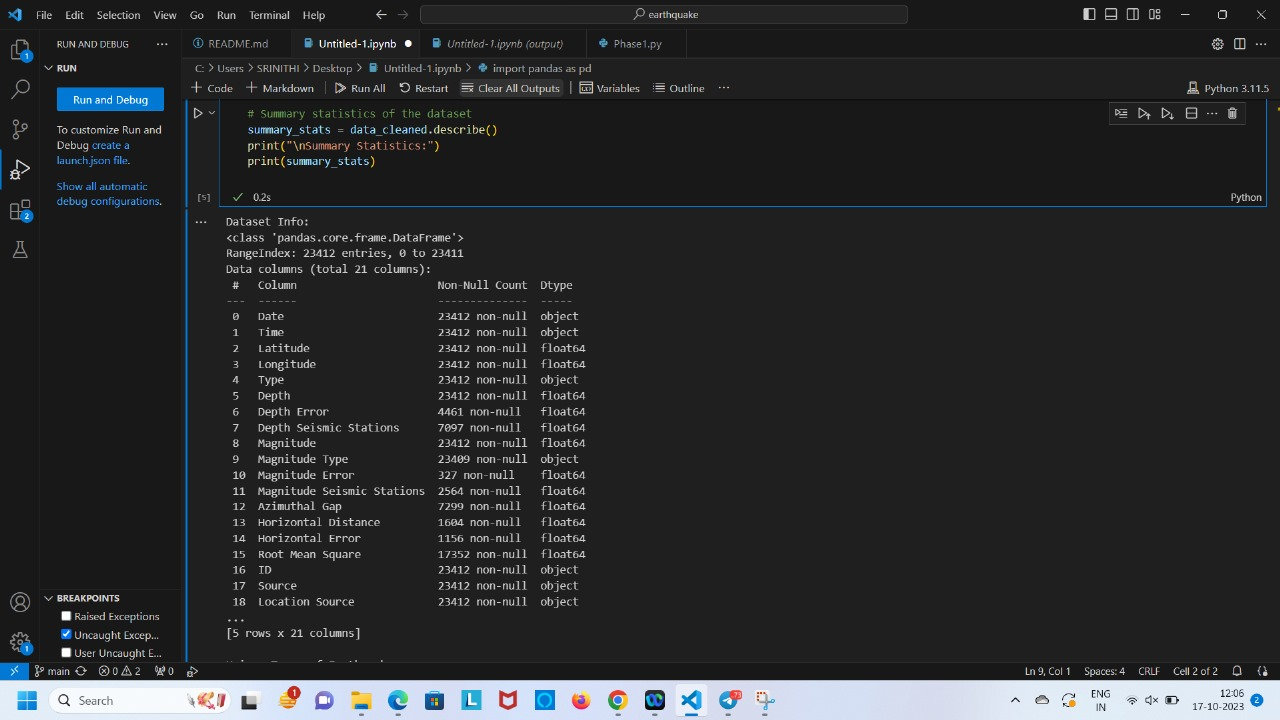
print(summary\_stats)

**Screenshot :phase 3.1**



Output:

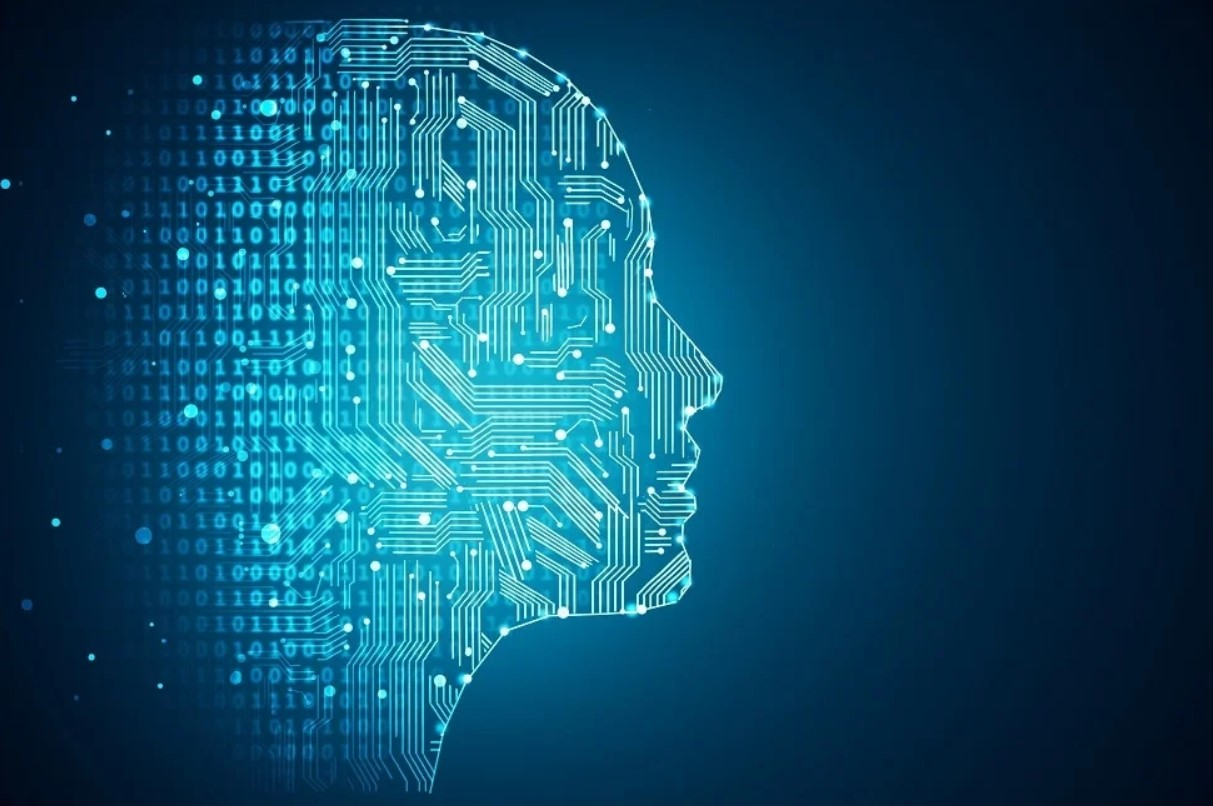
**Screenshot: phase 3.2**



**Phase 4**

**DOMAIN: ARTIFICIAL INTELLIGENCE**

**PROJECT TITLE:EARTHQUAKE PREDICTION USING PYTHON**



In this code, we've done the following:

* Loaded the preprocessed dataset.
* Selected the features (**Latitude**, **Longitude**, and **Depth**) as **X** and the target variable (**Magnitude**) as **y**.
* Split the data into training and testing sets using **train\_test\_split**.
* Created a simple linear regression model, trained it using the training data, and made predictions on the test data.
* Evaluated the model's performance using metrics like Mean Squared Error (MSE) and R-squared (R2).
* Visualized the relationship between actual and predicted magnitudes.

**Code**

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Load the preprocessed dataset

data = pd.read\_csv(&quot;C:\\Users\\SRINITHI\\Downloads\\archive

(1)\\database.csv&quot;)

# Select features (X) and target (y)

X = data[[&#39;Latitude&#39;, &#39;Longitude&#39;, &#39;Depth&#39;]]

y = data[&#39;Magnitude&#39;]

# 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)

# Create and train a linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model&#39;s performance

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(&quot;Mean Squared Error:&quot;, mse)

print(&quot;R-squared:&quot;, r2)

# Visualize the model&#39;s predictions

plt.scatter(y\_test, y\_pred, alpha=0.5)

plt.title(&#39;Actual vs. Predicted Magnitudes&#39;)

plt.xlabel(&#39;Actual Magnitude&#39;)

plt.ylabel(&#39;Predicted Magnitude&#39;)

plt.show()

import folium

# Load the preprocessed earthquake dataset

data = pd.read\_csv(&quot;C:\\Users\\SRINITHI\\Downloads\\archive

(1)\\database.csv&quot;)

# Create a map centered at a specific location (e.g., the world center)

map\_center = [0, 0]

m = folium.Map(location=map\_center, zoom\_start=2)

# Loop through the dataset and add markers for each earthquake

for index, row in data.iterrows():

folium.CircleMarker(

location=[row[&#39;Latitude&#39;], row[&#39;Longitude&#39;]],

radius=5,

color=&#39;red&#39;,

fill=True,

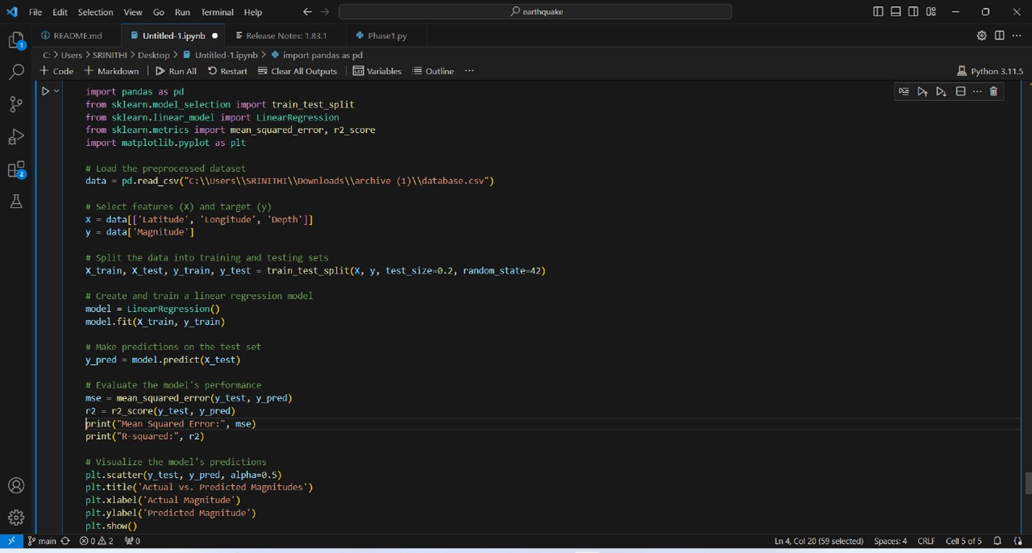
fill\_color=&#39;red&#39;,

popup=f&quot;Magnitude: {row[&#39;Magnitude&#39;]}, Depth: {row[&#39;Depth&#39;]}&quot;,

).add\_to(m)

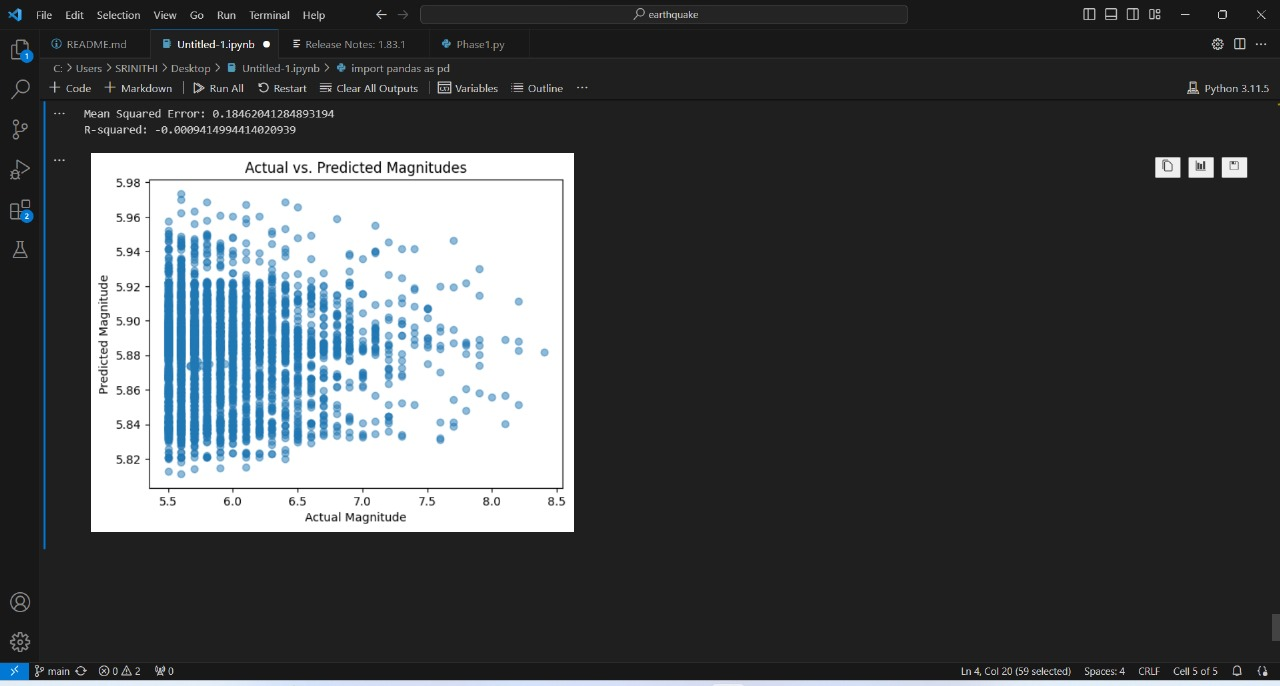
m

**Screenshot: phase 4.1**



**output**

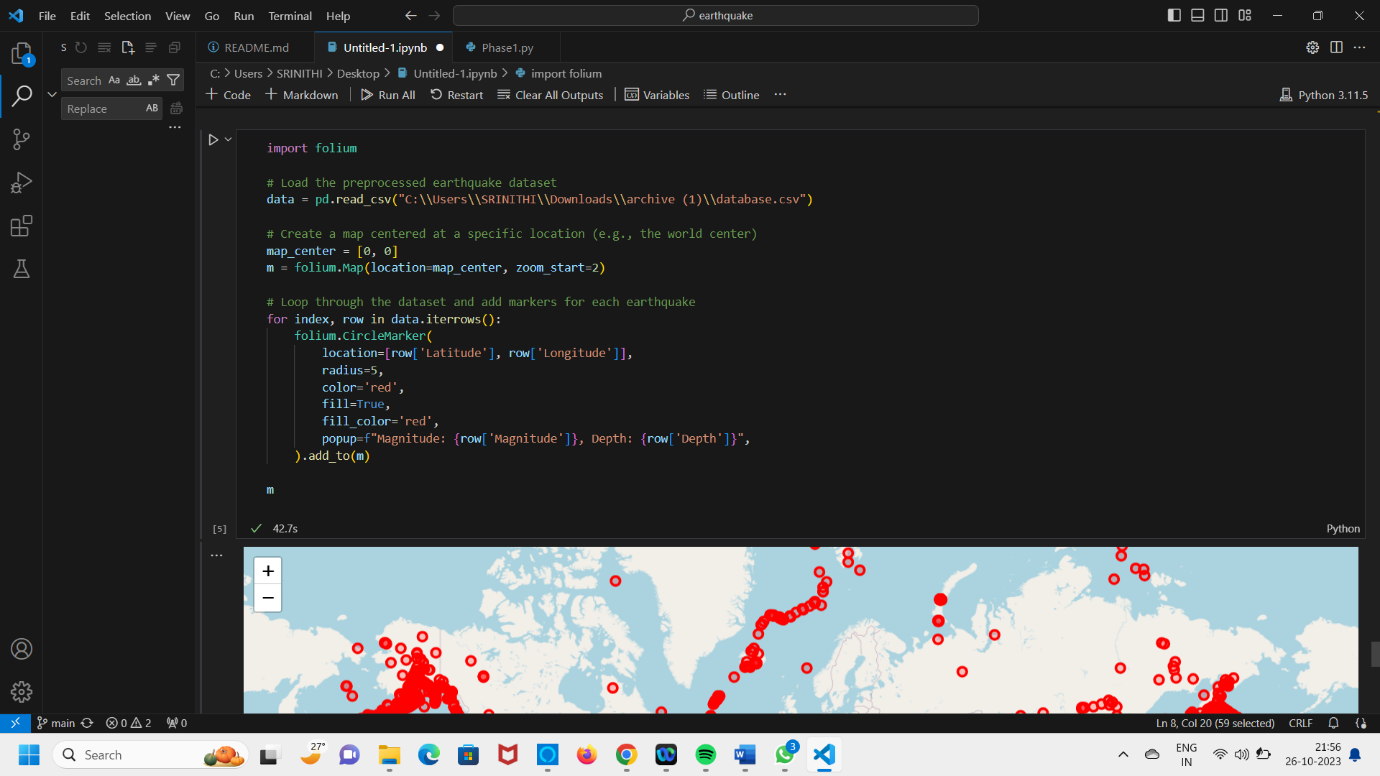
**screenshot: phase 4.2**

****

FOR VISUALISING THE DATA ON THE WORLD MAP

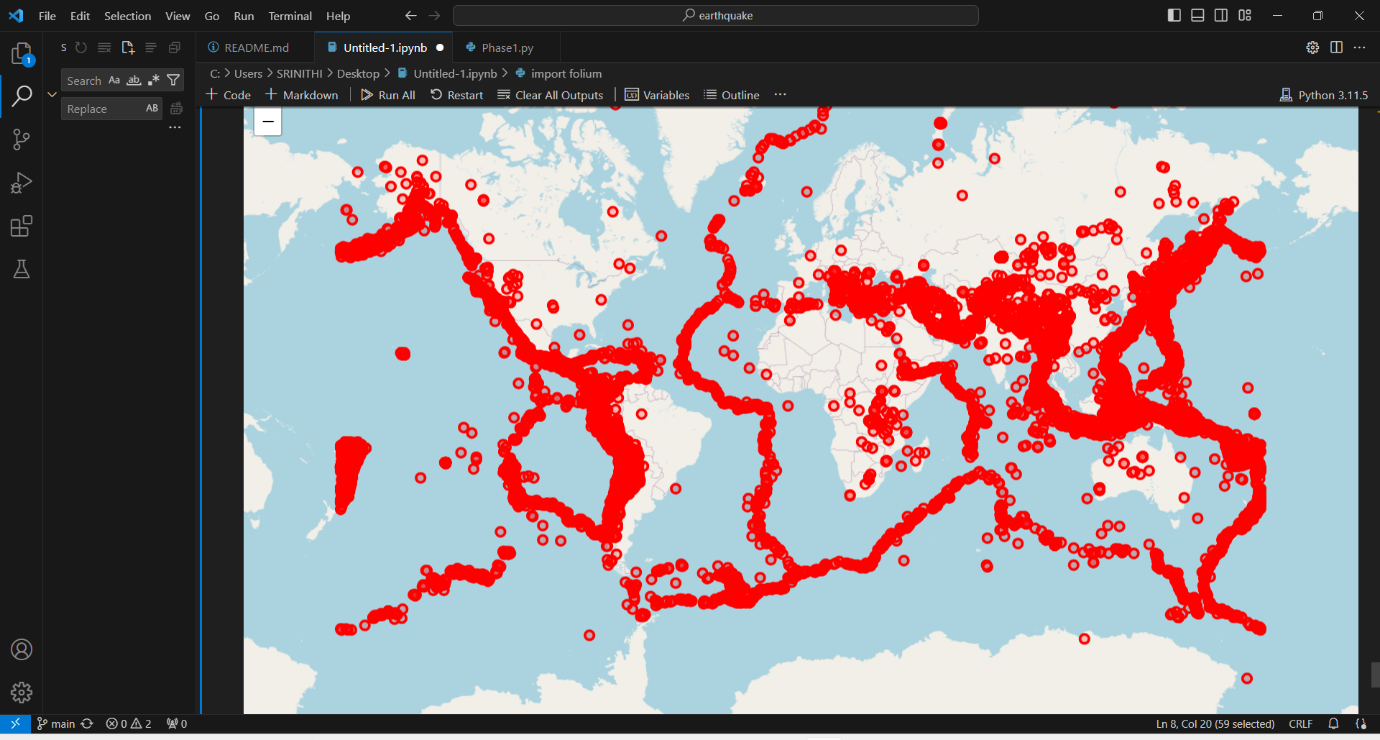
CODE:

Screenshot: phase 4.3



**output**

**screenshot: phase 4.4**

****

**CONCLUSION:**

For the given project we have defined what the problem is and using various techniques we have tried to improve the performance .We have also loaded and preprocessed the dataset , splitted the dataset into training and testing set ,we have evaluated mean squared error , visualised the dataset on the world map and we have attached screenshots of the code and the output.

SUBMITTED BY

SRINIDHI R [511321104096]

SOUNDARYA S [511321104094]

THRISHA V [511321104103]

SANDHIYA V [511321104087]

- THANK YOU