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.