**AI\_PHASE5**

**EARTHQUAKE PREDITION MODEL USING PYTHON PROJECT DOCUMENTATION**

**NAME: SATHISHKUMAR M**

**REG.NO: 610821205307**

**Phase 1: Problem Definition and Design**

**Thinking**

**Problem Definition:** The problem is to develop an earthquake

prediction model using a Kaggle dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features.

**Design Thinking:**

1**1. Data Source:**

The appropriate dataset containing earthquake data with features like date, time, latitude, longitude, depth, and magnitude is Kaggle dataset.

**Dataset Link:** [**https://www.kaggle.com/datasets/usgs/earthquakedatabase**](https://www.kaggle.com/datasets/usgs/earthquake-database)

**2. Feature Exploration:**

To Analyze and understand the distribution, correlations, and characteristics of the key features.

**Distribution:**

The distribution is done by the key features as histogram, the depth latitude, longitude, magnitude are as distributed as histograms. As head, describe methods are used

**Correlations:**

Exploring the correlation between features to identify any potential relationships. The correlation of the variables is done by corr() function**.**

**Characteristics of key features:**

Refers to the attributes, properties, and behaviors of the variables or attributes that are considered important or influential in a dataset. Key features are those that have a significant impact on the target variable or the outcome of the analysis, and understanding their characteristics is crucial for making informed decisions in data analysis and modelling.

**3.VISUALIZATION:**

A world map visualization to display earthquake frequency distribution. Creating a world map visualization to display earthquake frequency distribution requires geographic data and a suitable library for plotting

1. **Data Splitting:**

Splitting the dataset into a training set and a test set for model validation. Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are

Timestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and Ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

The RandomForestRegressor model to predict the outputs, we see the strange prediction from this with score above 80% which can be assumed to be best fit but not due to its predicted values

1. **Model Development:**

Building the neural network for earthquake magnitude prediction. Defining by the neural network architecture, including the number of layers, neurons per layer, and activation functions.

1. **Training and Evaluation:**

Training the model on the training set and evaluating its performance on the test set. Fit the chosen model to the training data. This involves setting hyperparameters and using the training data to adjust the model's internal parameters.

**Evaluation:**

Once the model is trained, evaluate its performance on the test set using appropriate evaluation metrics. Common metrics for earthquake prediction include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R²). the model.evaluate() method is that evaluates the x\_test and y\_test and the evaluation result on test data is displayed.

# Phase 2: Innovation

In this section need to put design into innovation to solve the problem

**HYPERPARAMETER TUNING:**

1. **Define Hyperparameters to Search:**

param\_grid = {

'model\_\_n\_estimators': [100, 200, 300],

'model\_\_max\_depth': [None, 10, 20, 30],

'model\_\_min\_samples\_split': [2, 5, 10], 'model\_\_min\_samples\_leaf': [1, 2, 4]

}

This dictionary defines the hyperparameters and their corresponding values to search for using grid search.

1. **Grid Search with Cross-Validation:**

grid\_search = GridSearchCV(estimator=pipeline, param\_grid=param\_grid, cv=5, n\_jobs=-1, verbose=2, error\_score='raise')

`GridSearchCV` is used to perform a grid search with 5-fold cross-validation to find the best hyperparameters. It utilizes the full pipeline (`pipeline`) and the hyperparameter grid (`param\_grid`).

1. **Fit the Model with the Best Hyperparameters:**

grid\_search.fit(X\_train, y\_train)

This code fits the model with the training data using the best hyperparameters found during grid search.

1. **Get the Best Hyperparameters:**

best\_params = grid\_search.best\_params\_ print("Best

Hyperparameters:", best\_params)

After grid search, this prints out the best hyperparameters that were found during the search.

1. **Use the Best Model for Prediction:**

best\_model = grid\_search.best\_estimator\_ y\_pred

= best\_model.predict(X\_test)

We use the best model (which includes the best hyperparameters) to make predictions on the testing data.

1. **Evaluate the Model's Performance:**

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

print("Accuracy:", accuracy)

Finally, the code calculates and prints the accuracy score, which is a measure of the model's performance on the test set.

**FEATURE ENGINEERING:**

1. **Creating a New Feature - Earthquake Magnitude Squared:**

X['magnitude\_squared'] = X['magnitude'] \*\* 2

* + In this step, a new feature named 'magnitude\_squared' is created in the DataFrame `X`.
  + It is calculated by squaring the values in the existing 'magnitude' feature. This can be a meaningful transformation if the relationship between the squared magnitude and the target variable is non-linear and can improve the model's performance.

1. **Splitting the Updated 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)

* + This code splits the updated DataFrame `X` and the target labels `y` into training and testing sets using the `train\_test\_split` function.
  + The `test\_size` parameter specifies that 20% of the data should be reserved for testing, and `random\_state` ensures reproducibility by fixing the random seed.

1. **Retraining the Model with Updated Features:**

best\_model.fit(X\_train, y\_train)

* + The `fit` method is called on the best model (`best\_model`) to train it using the updated training data (`X\_train` and `y\_train`).
  + This step reuses the best model found during hyperparameter tuning, which is assumed to be stored in the `best\_model` variable.

1. **Making Predictions and Evaluating:**

y\_pred = best\_model.predict(X\_test) accuracy

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

* + Here, predictions are made on the test data (`X\_test`) using the trained `best\_model`.
  + The accuracy of the model is calculated by comparing the predicted labels (`y\_pred`) with the true labels (`y\_test`) using the `accuracy\_score` function from scikit-learn.
  + The accuracy score measures the proportion of correctly predicted labels in the test set.

1. **Printing the Accuracy with Feature Engineering:**

print("Accuracy with Feature Engineering:", accuracy)

* + Finally, the code prints the accuracy of the model on the test set after feature engineering.
  + This allows you to assess whether the addition of the squared magnitude feature improved the model's predictive performance.

By following these steps, advanced techniques such as hyperparameter tuning and feature engineering to improve the prediction model's can be performed.

# Phase 3: Development Part 1

In this section building project by loading and preprocessing the dataset.

## DATA COLLECTION

**import** numpy **as** np **import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** os print(os.listdir("../input"))

['earthquake-database']

## DATA LOADING

data = pd.read\_csv("../input/earthquake-database/database.csv") data.head()

OUTPUT:

Date Time Latitude Longitude Type Depth Depth Error

\

1. 01/02/1965 13:44:18 19.246 145.616 Earthquake 131.6 NaN
2. 01/04/1965 11:29:49 1.863 127.352 Earthquake 80.0 NaN
3. 01/05/1965 18:05:58 -20.579 -173.972 Earthquake 20.0 NaN
4. 01/08/1965 18:49:43 -59.076 -23.557 Earthquake 15.0 NaN 4 01/09/1965 13:32:50 11.938 126.427 Earthquake 15.0 NaN

Depth Seismic Stations Magnitude Magnitude Type ... \ 0 NaN 6.0 MW ...

1. NaN 5.8 MW ...
2. NaN 6.2 MW ...
3. NaN 5.8 MW ...
4. NaN 5.8 MW ...

Magnitude Seismic Stations Azimuthal Gap Horizontal Distance \

1. NaN NaN NaN
2. NaN NaN NaN
3. NaN NaN NaN
4. NaN NaN NaN 4 NaN NaN NaN

Horizontal Error Root Mean Square ID Source Location Source

\

1. NaN NaN ISCGEM860706 ISCGEM ISCGEM
2. NaN NaN ISCGEM860737 ISCGEM ISCGEM
3. NaN NaN ISCGEM860762 ISCGEM ISCGEM
4. NaN NaN ISCGEM860856 ISCGEM ISCGEM 4 NaN NaN ISCGEM860890 ISCGEM ISCGEM

Magnitude Source Status

1. ISCGEM Automatic
2. ISCGEM Automatic
3. ISCGEM Automatic
4. ISCGEM Automatic
5. ISCGEM Automatic

[5 rows x 21 columns]

## QUICK OVERVIEW OF DATAS

data.head()

OUTPUT:

Date Time Latitude Longitude Type Depth Depth Error

\

1. 01/02/1965 13:44:18 19.246 145.616 Earthquake 131.6 NaN
2. 01/04/1965 11:29:49 1.863 127.352 Earthquake 80.0 NaN
3. 01/05/1965 18:05:58 -20.579 -173.972 Earthquake 20.0 NaN
4. 01/08/1965 18:49:43 -59.076 -23.557 Earthquake 15.0 NaN 4 01/09/1965 13:32:50 11.938 126.427 Earthquake 15.0 NaN

Depth Seismic Stations Magnitude Magnitude Type ... \ 0 NaN 6.0 MW ...

1. NaN 5.8 MW ...
2. NaN 6.2 MW ...
3. NaN 5.8 MW ...
4. NaN 5.8 MW ...

Magnitude Seismic Stations Azimuthal Gap Horizontal Distance \

1. NaN NaN NaN
2. NaN NaN NaN
3. NaN NaN NaN
4. NaN NaN NaN 4 NaN NaN NaN

Horizontal Error Root Mean Square ID Source Location Source

\

1. NaN NaN ISCGEM860706 ISCGEM ISCGEM
2. NaN NaN ISCGEM860737 ISCGEM ISCGEM
3. NaN NaN ISCGEM860762 ISCGEM ISCGEM
4. NaN NaN ISCGEM860856 ISCGEM ISCGEM 4 NaN NaN ISCGEM860890 ISCGEM ISCGEM

Magnitude Source Status

1. ISCGEM Automatic
2. ISCGEM Automatic
3. ISCGEM Automatic
4. ISCGEM Automatic
5. ISCGEM Automatic

[5 rows x 21 columns]

## COLUMNS NAMES

data.columns

OUTPUT:

Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error',

'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',

'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',

'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID', 'Source', 'Location Source', 'Magnitude Source', 'Status'], dtype='object')

## MAIN FEATURES OF EARTHQUAKE DATA

*# The main features from earthquake data creating a object namely, Date, Time, Latitude, Longitude, Depth, Magnitude*

data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']] data.head()

Date Time Latitude Longitude Depth Magnitude

1. 01/02/1965 13:44:18 19.246 145.616 131.6 6.0
2. 01/04/1965 11:29:49 1.863 127.352 80.0 5.8
3. 01/05/1965 18:05:58 -20.579 -173.972 20.0 6.2
4. 01/08/1965 18:49:43 -59.076 -23.557 15.0 5.8 4 01/09/1965 13:32:50 11.938 126.427 15.0 5.8

data.describe()

OUTPUT:

Latitude Longitude Depth Magnitude count 23412.000000 23412.000000 23412.000000 23412.000000 mean 1.679033 39.639961 70.767911 5.882531 std 30.113183 125.511959 122.651898 0.423066 min -77.080000 -179.997000 -1.100000 5.500000 25% -18.653000 -76.349750 14.522500 5.600000

50% -3.568500 103.982000 33.000000 5.700000 75% 26.190750 145.026250 54.000000 6.000000 max 86.005000 179.998000 700.000000 9.100000

## SCALING THE RANDOM DATA

*#Here, the data is random we need to scale according to inputs to the model. So, we convert given Date and Time to Unix time which is in seconds and a numeral. This can be easily used as input for the network we built* **import** datetime **import** time

*# Create a list to store Unix timestamps* timestamp = []

*# Iterate through the "Date" and "Time" columns* **for** d, t **in** zip(data['Date'], data['Time']): **try**:

ts = datetime.datetime.strptime(d + ' ' + t, '%m/%d/%Y %H:%M:%S') timestamp.append(time.mktime(ts.timetuple())) **except** ValueError: timestamp.append('ValueError')

*# Create a Pandas Series from the timestamp list* timeStamp = pd.Series(timestamp)

*# Add the "Timestamp" column to the DataFrame* data['Timestamp'] = timeStamp.values *# Drop the "Date" and "Time" columns*

final\_data = data.drop(['Date', 'Time'], axis=1) *# Remove rows with 'ValueError' in the "Timestamp" column* final\_data = final\_data[final\_data['Timestamp'] != 'ValueError']

*# Display the first few rows of the final dataset* final\_data.head()

OUTPUT:

Latitude Longitude Depth Magnitude Timestamp

1. 19.246 145.616 131.6 6.0 -157630542.0
2. 1.863 127.352 80.0 5.8 -157465811.0
3. -20.579 -173.972 20.0 6.2 -157355642.0
4. -59.076 -23.557 15.0 5.8 -157093817.0
5. 11.938 126.427 15.0 5.8 -157026430.0

**Phase 4: Development Part 2**

In this section buildinging the project by performing different activities like feature engineering, model training, evaluation

## VISUALIZATION

*# Extract latitude, longitude, and magnitude columns* latitude = data['Latitude'] longitude = data['Longitude'] magnitude = data['Magnitude']

*# Create a scatter plot to visualize earthquakes on a map* plt.figure(figsize=(10, 6))

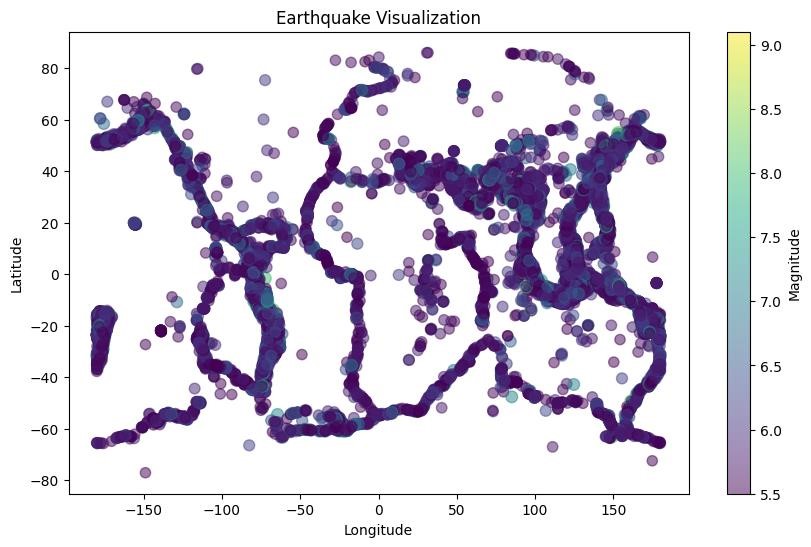
plt.scatter(longitude, latitude, c=magnitude, cmap='viridis', s=magnitude \*

10, alpha=0.5) plt.colorbar(label='Magnitude') plt.title('Earthquake Visualization') plt.xlabel('Longitude')

plt.ylabel('Latitude')

*# Show the plot* plt.show()

OUTPUT:



## FEATURE ENGINEERING

**import** numpy **as** np **import** pandas **as** pd

*# Synthetic seismic data (replace this with your real data)* data = pd.DataFrame({

'time': np.arange(0, 100, 0.1),

'acceleration': np.random.rand(1000),

*# Add more columns for other sensor data if available* })

*# Define functions for feature engineering* **def** basic\_statistics(data):

*# Calculate basic statistical features* features = {

'mean': data['acceleration'].mean(),

'std\_dev': data['acceleration'].std(),

'min': data['acceleration'].min(),

'max': data['acceleration'].max(),

}

**return** features

**def** time\_domain\_features(data): *# Calculate time domain features*

*# For example, root mean square (RMS) amplitude* rms = np.sqrt(np.mean(data['acceleration']\*\*2)) **return** {'RMS\_amplitude': rms}

**def** frequency\_domain\_features(data):

*# Calculate frequency domain features using Fourier transform* fft\_result = np.fft.fft(data['acceleration']) *# Extract amplitude and frequency information* amplitude = np.abs(fft\_result)

frequency = np.fft.fftfreq(len(fft\_result)) *# Find the dominant frequency component* dominant\_frequency = frequency[np.argmax(amplitude)] **return** {'dominant\_frequency': dominant\_frequency}

*# Apply feature engineering functions to your data* statistical\_features = basic\_statistics(data) time\_domain\_features = time\_domain\_features(data) frequency\_domain\_features = frequency\_domain\_features(data)

*# Combining all features into a single feature vector*

feature\_vector = {\*\*statistical\_features, \*\*time\_domain\_features,

\*\*frequency\_domain\_features}

*# Your feature vector is now ready for use in training your earthquake prediction model* print(feature\_vector)

OUTPUT:

{'mean': 0.49201126044541615, 'std\_dev': 0.29111111319763416, 'min':

0.0012840627090102696, 'max': 0.9997457913830645, 'RMS\_amplitude':

0.5716082705420084, 'dominant\_frequency': 0.0}

## MODEL TRAINING

X = final\_data[['Timestamp', 'Latitude', 'Longitude']] y = final\_data[['Magnitude', 'Depth']]

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.ensemble **import** RandomForestRegressor **from** sklearn.metrics **import** mean\_squared\_error

*# Splitting 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)

*# Creating a Random Forest regressor* model = RandomForestRegressor(n\_estimators=100, random\_state=42)

*# Training the model on the training data*

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

*# Making predictions on the test data* y\_pred = model.predict(X\_test)

*# Evaluating the model's performance (for regression tasks)* mse = mean\_squared\_error(y\_test, y\_pred) print("Mean Squared Error:", mse)

OUTPUT:

Mean Squared Error: 968.4488974862994

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

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

OUTPUT:

(18727, 3) (4682, 3) (18727, 2) (4682, 3)

**from** sklearn.ensemble **import** RandomForestRegressor

reg = RandomForestRegressor(random\_state=42) reg.fit(X\_train, y\_train) reg.predict(X\_test)

OUTPUT:

array([[ 5.865, 42.024], [ 5.826, 33.09 ],

[ 6.082, 39.741],

...,

[ 6.306, 23.059],

[ 5.96 , 592.283], [ 5.808, 38.222]]) reg.score(X\_test, y\_test)

OUTPUT:

0.3926671400442392

**from** sklearn.model\_selection **import** GridSearchCV

parameters = {'n\_estimators':[10, 20, 50, 100, 200, 500]}

grid\_obj = GridSearchCV(reg, parameters) grid\_fit = grid\_obj.fit(X\_train, y\_train) best\_fit = grid\_fit.best\_estimator\_ best\_fit.predict(X\_test)

OUTPUT:

array([[ 5.886 , 43.031 ], [ 5.82 , 31.3982],

[ 6.0124, 39.5216],

...,

[ 6.294 , 22.9908],

[ 5.9218, 592.385 ], [ 5.7894, 39.2764]])

## NEURAL NETWORK MODEL

**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

*# Generate synthetic seismic and geological data (replace with real data)* n\_samples = 1000 n\_features = 10

X = np.random.rand(n\_samples, n\_features)

y = np.random.randint(2, size=n\_samples) *# Binary labels (0: no earthquake, 1: earthquake)*

*# Feature engineering and preprocessing (replace with actual preprocessing steps)* scaler = StandardScaler() X = scaler.fit\_transform(X)

*# Splitting 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)

*# Building a basic feedforward neural network model* model = keras.Sequential([

keras.layers.Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)), keras.layers.Dense(64, activation='relu'),

keras.layers.Dense(1, activation='sigmoid')

])

*# Compiling the model*

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

*# Training the model*

model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

## MODEL EVALUATION

*# Evaluating the model on the test data* loss, accuracy = model.evaluate(X\_test, y\_test) print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

OUTPUT:

Epoch 1/50

25/25 [==============================] - 1s 13ms/step - loss: 0.7015 -

accuracy: 0.5000 - val\_loss: 0.7059 - val\_accuracy: 0.4400 Epoch 2/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6850 - accuracy: 0.5425 - val\_loss: 0.7107 - val\_accuracy: 0.4600 Epoch 3/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6771 -

accuracy: 0.5813 - val\_loss: 0.7109 - val\_accuracy: 0.4550 Epoch 4/50

25/25 [==============================] - 0s 8ms/step - loss: 0.6713 - accuracy: 0.5738 - val\_loss: 0.7097 - val\_accuracy: 0.4600 Epoch 5/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6646 - accuracy: 0.6075 - val\_loss: 0.7112 - val\_accuracy: 0.4400 Epoch 6/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6595 -

accuracy: 0.5938 - val\_loss: 0.7147 - val\_accuracy: 0.4550 Epoch 7/50

25/25 [==============================] - 0s 4ms/step - loss: 0.6534 -

accuracy: 0.6162 - val\_loss: 0.7126 - val\_accuracy: 0.4500 Epoch 8/50

25/25 [==============================] - 0s 4ms/step - loss: 0.6470 -

accuracy: 0.6250 - val\_loss: 0.7141 - val\_accuracy: 0.4400 Epoch 9/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6420 -

accuracy: 0.6313 - val\_loss: 0.7181 - val\_accuracy: 0.4650 Epoch 10/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6358 -

accuracy: 0.6313 - val\_loss: 0.7191 - val\_accuracy: 0.4650 Epoch 11/50

25/25 [==============================] - 0s 4ms/step - loss: 0.6259 -

accuracy: 0.6662 - val\_loss: 0.7170 - val\_accuracy: 0.4850 Epoch 12/50

25/25 [==============================] - 0s 5ms/step - loss: 0.6199 -

accuracy: 0.6712 - val\_loss: 0.7249 - val\_accuracy: 0.4850 Epoch 13/50

25/25 [==============================] - 0s 4ms/step - loss: 0.6113 -

accuracy: 0.6900 - val\_loss: 0.7219 - val\_accuracy: 0.4950 Epoch 14/50

25/25 [==============================] - 0s 4ms/step - loss: 0.6047 -

accuracy: 0.7025 - val\_loss: 0.7189 - val\_accuracy: 0.5000 Epoch 15/50

25/25 [==============================] - 0s 5ms/step - loss: 0.5968 -

accuracy: 0.7075 - val\_loss: 0.7280 - val\_accuracy: 0.4850 Epoch 16/50

25/25 [==============================] - 0s 4ms/step - loss: 0.5878 -

accuracy: 0.7262 - val\_loss: 0.7263 - val\_accuracy: 0.4700 Epoch 17/50

25/25 [==============================] - 0s 5ms/step - loss: 0.5785 -

accuracy: 0.7250 - val\_loss: 0.7319 - val\_accuracy: 0.4800 Epoch 18/50

25/25 [==============================] - 0s 5ms/step - loss: 0.5697 -

accuracy: 0.7362 - val\_loss: 0.7417 - val\_accuracy: 0.4650

Epoch 19/50

25/25 [==============================] - 0s 5ms/step - loss: 0.5598 - accuracy: 0.7538 - val\_loss: 0.7333 - val\_accuracy: 0.4900 Epoch 20/50

25/25 [==============================] - 0s 5ms/step - loss: 0.5501 -

accuracy: 0.7650 - val\_loss: 0.7402 - val\_accuracy: 0.4650 Epoch 21/50

25/25 [==============================] - 0s 5ms/step - loss: 0.5408 -

accuracy: 0.7638 - val\_loss: 0.7440 - val\_accuracy: 0.4650 Epoch 22/50

25/25 [==============================] - 0s 4ms/step - loss: 0.5300 - accuracy: 0.7763 - val\_loss: 0.7530 - val\_accuracy: 0.4700 Epoch 23/50

25/25 [==============================] - 0s 4ms/step - loss: 0.5202 - accuracy: 0.7800 - val\_loss: 0.7539 - val\_accuracy: 0.4950

Epoch 24/50

25/25 [==============================] - 0s 4ms/step - loss: 0.5113 -

accuracy: 0.8000 - val\_loss: 0.7617 - val\_accuracy: 0.4850 Epoch 25/50

25/25 [==============================] - 0s 4ms/step - loss: 0.5043 -

accuracy: 0.7900 - val\_loss: 0.7582 - val\_accuracy: 0.4850 Epoch 26/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4944 -

accuracy: 0.7937 - val\_loss: 0.7634 - val\_accuracy: 0.4950 Epoch 27/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4820 -

accuracy: 0.8175 - val\_loss: 0.7699 - val\_accuracy: 0.4800 Epoch 28/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4719 -

accuracy: 0.8150 - val\_loss: 0.7832 - val\_accuracy: 0.4750 Epoch 29/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4603 -

accuracy: 0.8263 - val\_loss: 0.7937 - val\_accuracy: 0.4800 Epoch 30/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4600 -

accuracy: 0.8250 - val\_loss: 0.7819 - val\_accuracy: 0.5100 Epoch 31/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4462 -

accuracy: 0.8225 - val\_loss: 0.8214 - val\_accuracy: 0.4800 Epoch 32/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4316 -

accuracy: 0.8450 - val\_loss: 0.8068 - val\_accuracy: 0.4800 Epoch 33/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4133 -

accuracy: 0.8587 - val\_loss: 0.8268 - val\_accuracy: 0.4600 Epoch 34/50

25/25 [==============================] - 0s 4ms/step - loss: 0.4042 -

accuracy: 0.8675 - val\_loss: 0.8109 - val\_accuracy: 0.4750 Epoch 35/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3955 - accuracy: 0.8775 - val\_loss: 0.8383 - val\_accuracy: 0.4850 Epoch 36/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3863 - accuracy: 0.8900 - val\_loss: 0.8515 - val\_accuracy: 0.5000 Epoch 37/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3807 - accuracy: 0.8725 - val\_loss: 0.8455 - val\_accuracy: 0.5000 Epoch 38/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3745 - accuracy: 0.8863 - val\_loss: 0.8584 - val\_accuracy: 0.5000

Epoch 39/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3564 - accuracy: 0.9000 - val\_loss: 0.8744 - val\_accuracy: 0.4750 Epoch 40/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3484 - accuracy: 0.9137 - val\_loss: 0.8772 - val\_accuracy: 0.4850 Epoch 41/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3338 -

accuracy: 0.9000 - val\_loss: 0.8788 - val\_accuracy: 0.5050 Epoch 42/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3304 -

accuracy: 0.9000 - val\_loss: 0.9459 - val\_accuracy: 0.4600 Epoch 43/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3270 -

accuracy: 0.8988 - val\_loss: 0.9118 - val\_accuracy: 0.4950 Epoch 44/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3149 -

accuracy: 0.9275 - val\_loss: 0.9395 - val\_accuracy: 0.4400 Epoch 45/50

25/25 [==============================] - 0s 4ms/step - loss: 0.3010 -

accuracy: 0.9237 - val\_loss: 0.9541 - val\_accuracy: 0.4600 Epoch 46/50

25/25 [==============================] - 0s 4ms/step - loss: 0.2908 -

accuracy: 0.9262 - val\_loss: 0.9331 - val\_accuracy: 0.4650 Epoch 47/50

25/25 [==============================] - 0s 4ms/step - loss: 0.2813 -

accuracy: 0.9350 - val\_loss: 0.9438 - val\_accuracy: 0.4900 Epoch 48/50

25/25 [==============================] - 0s 4ms/step - loss: 0.2721 -

accuracy: 0.9425 - val\_loss: 0.9695 - val\_accuracy: 0.4850 Epoch 49/50

25/25 [==============================] - 0s 4ms/step - loss: 0.2704 -

accuracy: 0.9337 - val\_loss: 1.0072 - val\_accuracy: 0.4450 Epoch 50/50

25/25 [==============================] - 0s 4ms/step - loss: 0.2602 - accuracy: 0.9425 - val\_loss: 0.9908 - val\_accuracy: 0.4700

7/7 [==============================] - 0s 3ms/step - loss: 0.9908 - accuracy:

0.4700

Test Loss: 0.9907826781272888, Test Accuracy: 0.4699999988079071

**CONCLUSION OF DOCUMENT:**

Thus the entire Earthquake prediction Model using python has been builded with the essential techniques such as Loading the dataset, preprocessing the datatset, and other included technical steps such as feature engineering, Model training, Model evaluation and various features are explained and builded Successfully.