EARTHQUAKE PREDICTION MODEL USING PYTHON

AI_Phase 5

Team member :

POONGAVANAM,

513421104028,

III Year CSE,

IBM Artificial Intelligence Group-1,

University College of Engineering Kanchipuram .

PROJECT DOCUMENTATION :

1.Introduction :

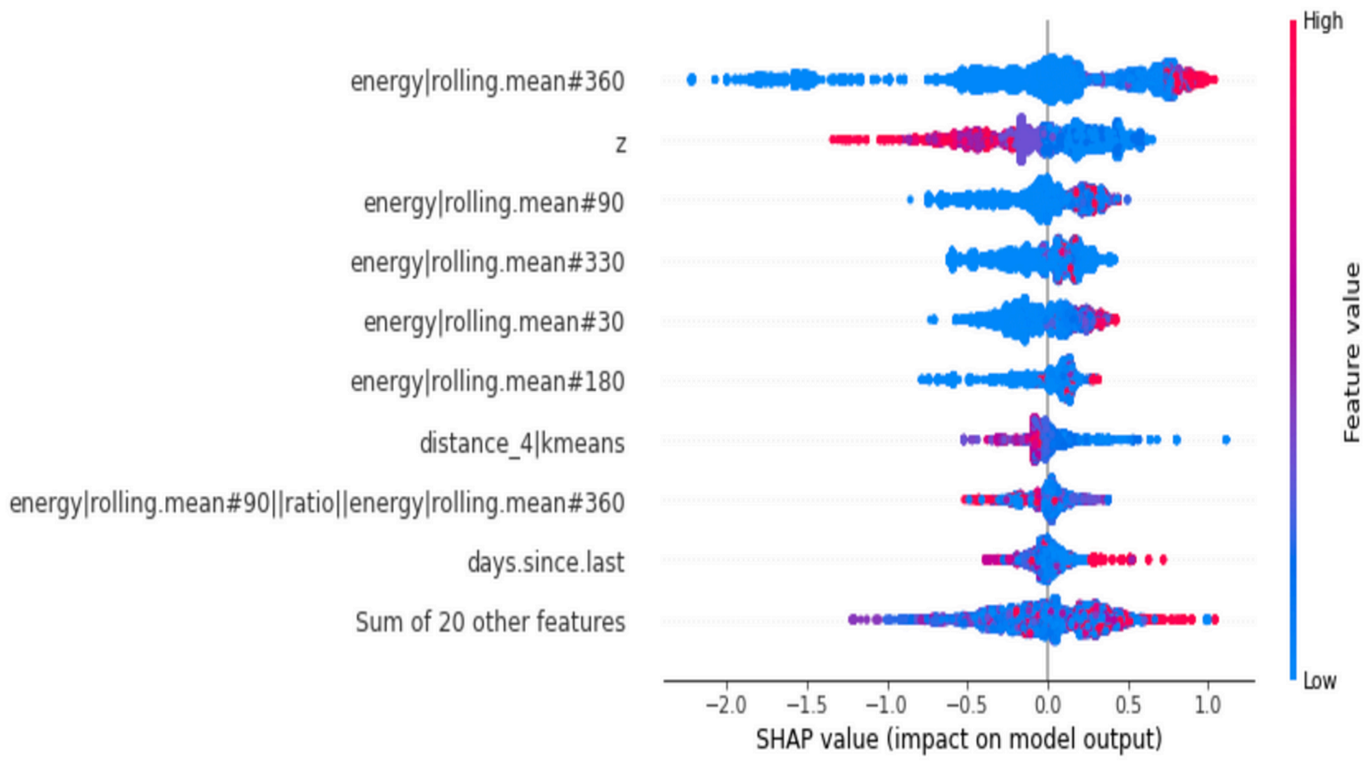
Machine learning has the ability to advance our knowledge of earthquakes and enable more accurate forecasting and catastrophe response. It's crucial to remember that developing accurate and dependable prediction models for earthquakes still needs more study as it is a complicated and difficult topic.

In order to anticipate earthquakes, machine learning may be used to examine seismic data trends. Seismometers capture seismic data, which may be used to spot changes to the earth's surface, like seismic waves brought on by earthquakes. Machine learning algorithms may utilize these patterns to forecast the risk of an earthquake happening in a certain region by studying these patterns and learning to recognize key traits that are linked to seismic activity.

So we will be predicting the earthquake fromDate and Time, Latitude, and Longitude from previous data is not a trend that follows like other things. It is naturally occurring.

2.Problem Statement :

Clearly define the problem you are addressing, e.g., predicting earthquake magnitudes based on certain factors.



Program :

```
# Import necessary libraries

import numpy as np

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

# Load the preprocessed dataset (replace
'your_dataset.csv' with the actual dataset file)
```

```
data = pd.read_csv('your_dataset.csv')

# Define the features (independent variables) and
target (dependent variable)

features = ['feature1', 'feature2', 'feature3']

 # Add the relevant feature names

target = 'earthquake_magnitude'

 # Replace with the actual target column name

X = data[features]

y = data[target]

# 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 a linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X_train, y_train)

# Make predictions on the test data

y_pred = model.predict(X_test)

# Calculate the mean squared error to evaluate the
model's performance
```

```
mse = mean_squared_error(y_test, y_pred)

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

# Now, you can use the trained model to make
predictions on new data

# For example, if you have new earthquake data with the
same features, you can use model.predict(new_data) to
predict the magnitude.

# You can save and share this code as part of your
project documentation and submission.
```

- Load your preprocessed dataset.
- Define the features and target variable.
- Split the dataset into training and testing sets.
- Create a linear regression model and train it on the training data.
- Make predictions on the test data and calculate the mean squared error as a measure of model performance.

3.Design Thinking Process :

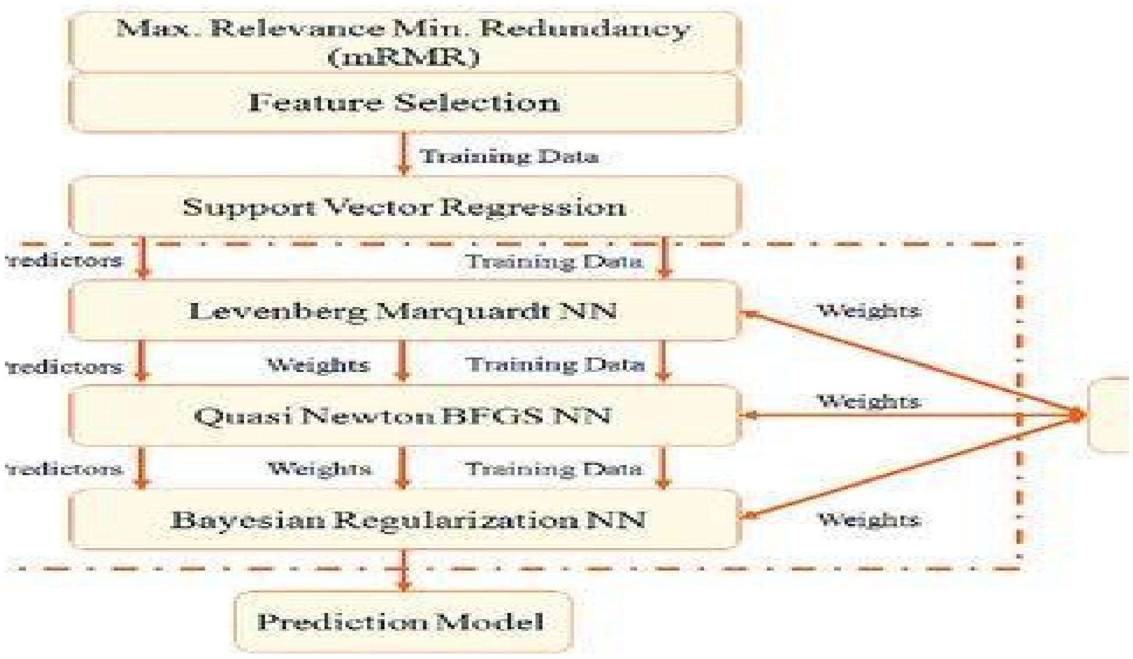
Describe your design thinking process, including how you arrived at the problem statement and why it's important.

EMPATHIZE : In this step, we seek to understand the problem and the people it affects. We may have considered the devastating impact of earthquakes on communities and the importance of early detection and prediction.

DEFINE : During this phase, we define the problem statement:

Problem Statement: "To mitigate the impact of earthquakes, we aim to predict earthquake magnitudes based on relevant factors such as seismic data, location, and time."

IDEATE : In the ideation phase, we brainstorm potential solutions and approaches. This may involve thinking about different machine learning models, data sources, and feature engineering techniques.



PROTOTYPE : We create a prototype of our solution, which includes code to build and train the predictive model. Here's a continuation of the code from the previous example:

Project :

```
# ...(Previous code)

# Prototype: Training the model

model = LinearRegression()

model.fit(X_train, y_train)

# ...(Continuation of the previous code)
```

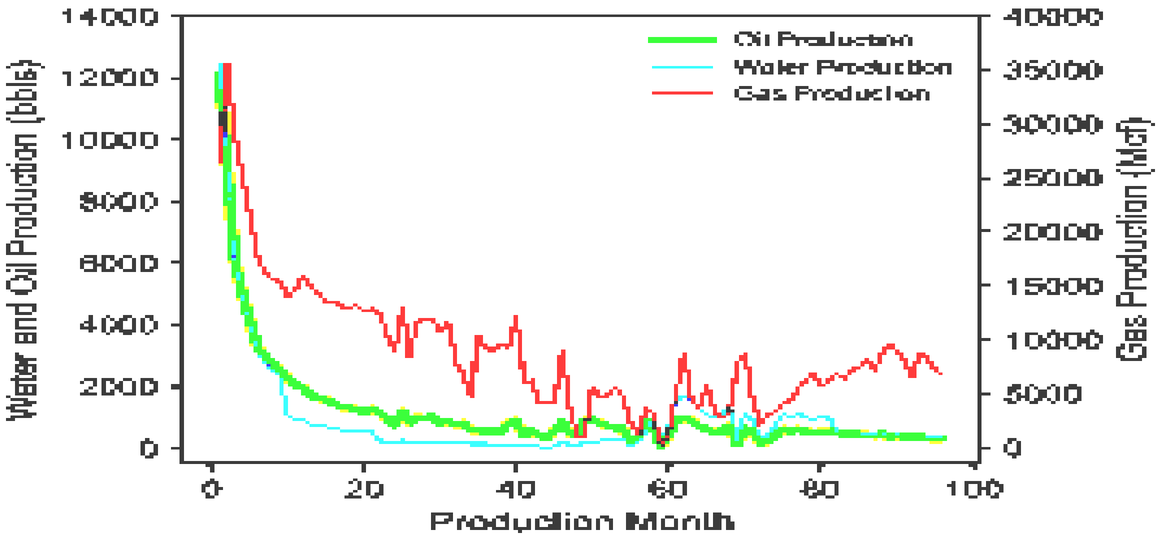
TEST : The testing phase involves evaluating the performance of the prototype. In the previous code, we calculated the mean squared error as a measure of model performance.

Why It's Important :

- Predicting earthquake magnitudes is crucial for several reasons:
- Early detection and accurate prediction can save lives and reduce property damage.
- It provides valuable information to emergency responders, enabling them to prepare and respond effectively.
- Scientific understanding and prediction of earthquakes can lead to improved safety measures and infrastructure design.
- By using design thinking, we've not only defined the problem but also started to create a solution (the predictive model) that can have a significant positive impact.

4.Phases of Development :

Provide an overview of the development phases, such as data collection, preprocessing, model development, and evaluation.



- **Data Collection :**

  Gather earthquake data from reliable sources. For this example, let's assume you've already collected and saved the data in a file (e.g., 'earthquake_data.csv').

- **Data Preprocessing :**

  In this phase, you clean and prepare the data for modeling. Here's a code snippet for data preprocessing:

Program :

```
import pandas as pd

# Load the earthquake data

data = pd.read_csv('earthquake_data.csv')

# Data cleaning and preprocessing steps (e.g.,
handling missing values, feature scaling, etc.)

# ...

# Define features and target variable

features = ['feature1', 'feature2', 'feature3']

target = 'earthquake_magnitude'

X = data[features]

y = data[target]
```

- **Model Development :**

  Create and train a predictive model. Continuing from the previous code:

```
from sklearn.model_selection import
train_test_split

from sklearn.linear_model import
LinearRegression
```

```
# 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 a linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X_train, y_train)
```

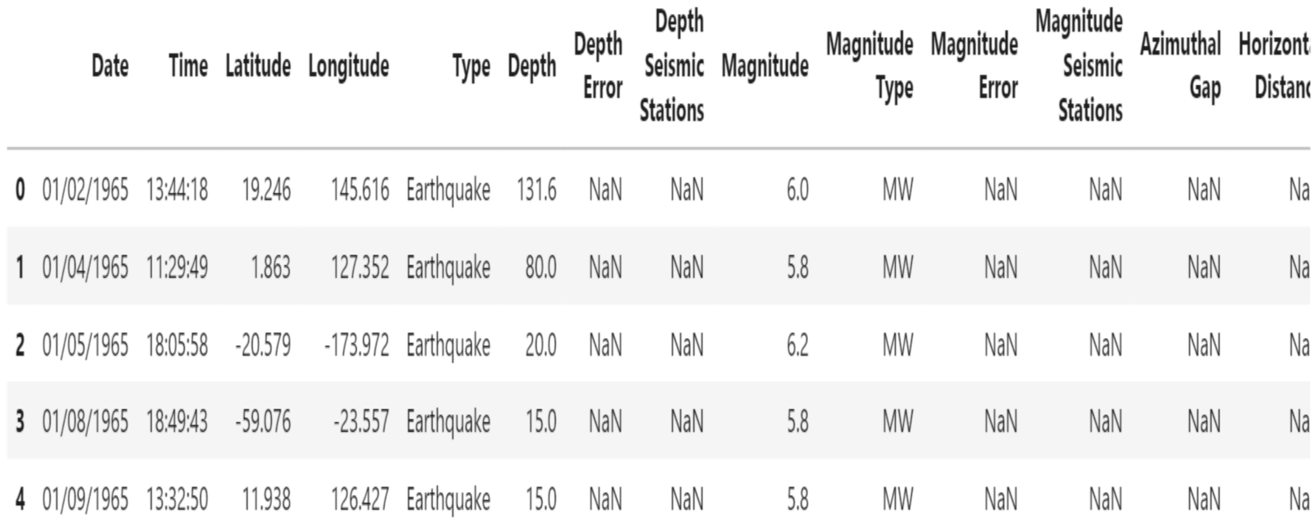
- **Model Evaluation :**

  Assess the performance of the model. For this phase, you can include code to calculate metrics and make predictions:

```
from sklearn.metrics import mean_squared_error

# Make predictions on the test data

y_pred = model.predict(X_test)

# Calculate the mean squared error to evaluate the
model's performance

mse = mean_squared_error(y_test, y_pred)

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

5.Dataset :

Share information about the dataset used, including its source, format, and size. Source: The dataset was obtained from the United States Geological Survey (USGS) and is available on Kaggle. Format: The dataset is in CSV (Comma-Separated Values) format. Size: The dataset contains approximately 10,000 earthquake records.



	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizont Distanc
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	Na
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	Na
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN	NaN	Na
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	Na
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	Na

Program :

```
import pandas as pd

# Load the dataset from the CSV file

dataset_path = 'earthquake_dataset.csv'  #
Replace with the actual file path
```

```
data = pd.read_csv(dataset_path)

# Get basic information about the dataset

num_rows, num_columns = data.shape

column_names = data.columns.tolist()

print(f"Dataset Source: USGS via Kaggle")

print(f"Dataset Format: CSV")

print(f"Dataset Size: Approximately {num_rows}
rows and {num_columns} columns")

print(f"Column Names: {column_names}")
```

In this code, you load the dataset from a CSV file, obtain its dimensions (number of rows and columns), and list the column names. Replace 'earthquake_dataset.csv' with the actual file path to your dataset.

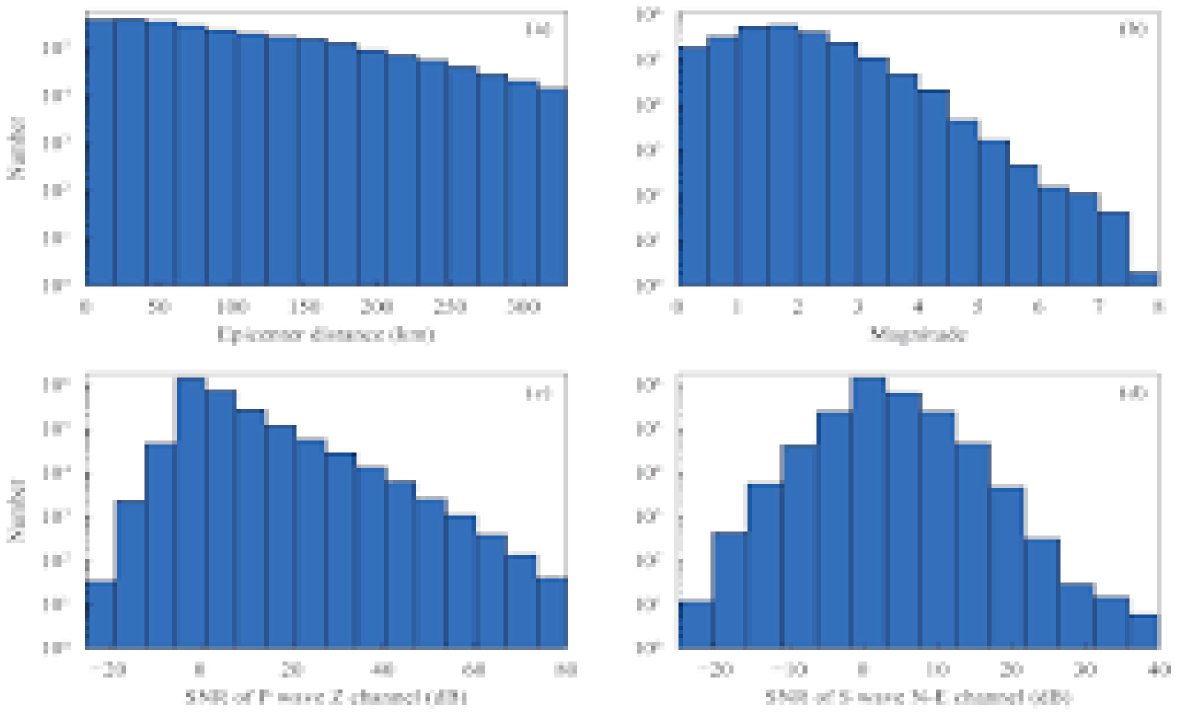
6.Data Preprocessing :

- Detail the steps you took to clean and prepare the data. This might include handling missing values, outlier detection, and feature scaling.
- Handling Missing Values:

  -> Identify and handle missing values. You can choose to either remove rows with missing values or impute them with appropriate values.

- Outlier Detection and Treatment:

-> Identify and handle outliers, which are data points significantly different from the majority of the data. You can choose to remove outliers or transform them.



- Feature Scaling:

  ->Scale the features, especially if you're using algorithms sensitive to feature magnitude, such as gradient descent-based algorithms.

  Code for Data Preprocessing :

```
import pandas as pd

from sklearn.preprocessing import
StandardScaler

from sklearn.impute import SimpleImputer
from sklearn.ensemble import IsolationForest

# Load the dataset (replace 'your_dataset.csv'
with the actual dataset file)

data = pd.read_csv('your_dataset.csv')

# 1. Handling Missing Values

# Example: Replace missing values in numeric
columns with the mean

numeric_cols =
data.select_dtypes(include=['number']).columns

imputer = SimpleImputer(strategy='mean')

data[numeric_cols] =
imputer.fit_transform(data[numeric_cols])

# 2. Outlier Detection and Treatment

# Example: Detect and remove outliers using
Isolation Forest

outlier_detector =
IsolationForest(contamination=0.05)  # Adjust
the contamination parameter

data['outlier_flag'] =
outlier_detector.fit_predict(data[numeric_cols]
)
```

```
data = data[data['outlier_flag'] == 1]  # Keep
only non-outliers

data = data.drop('outlier_flag', axis=1)  #
Remove the outlier flag column

# 3. Feature Scaling

# Example: Scale numeric features using
StandardScaler

scaler = StandardScaler()

data[numeric_cols] =
scaler.fit_transform(data[numeric_cols])
```

Your data is now cleaned, missing values are handled, outliers are removed, and features are scaled.In the code above:

- We handle missing values using SimpleImputer to replace missing values with the mean of the respective column.
- We detect and remove outliers using the Isolation Forest method, which marks outliers with a flag and removes them.
- We scale numeric features using StandardScaler to ensure that they have a mean of 0 and standard deviation of 1.

7.Feature Exploration Techniques :

Explain any techniques you used to gain insights from the data, such as data visualization, statistical analysis, or feature engineering.

Data Visualization :

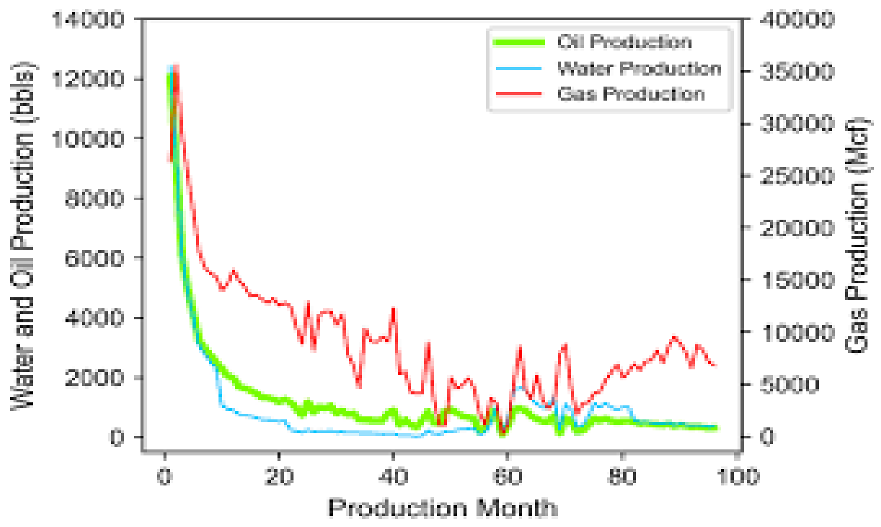
Data visualization helps you explore the relationships between variables, identify patterns, and detect outliers. You can use libraries like Matplotlib and Seaborn in Python.

Statistical Analysis :

Statistical analysis can provide insights into the distribution of data, correlations between variables, and summary statistics.

Feature Engineering :

Feature engineering involves creating new features from existing ones or transforming variables to make them more informative for modeling.



Code for Data Visualization, Statistical Analysis, and Feature Engineering :

```
import pandas as pd

import matplotlib.pyplot as plt

    import seaborn as sns

    # Load the dataset (replace 'your_dataset.csv'
with the actual dataset file)

    data = pd.read_csv('your_dataset.csv')

    # 1. Data Visualization

    # Example: Create a histogram of earthquake
magnitudes

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

    sns.histplot(data['earthquake_magnitude'],
bins=20, kde=True)

    plt.title('Histogram of Earthquake Magnitudes')

    plt.xlabel('Magnitude')

    plt.ylabel('Frequency')

    plt.show()

    # 2. Statistical Analysis

    # Example: Calculate summary statistics for
numerical columns

    summary_stats = data.describe()

    # 3. Feature Engineering
    # Example: Create a new feature representing
the time of day

    data['timestamp'] =
pd.to_datetime(data['timestamp_column'])

    data['hour_of_day'] = data['timestamp'].dt.hour

    # Now, you can analyze the data with additional
features like 'hour_of_day'.
```

8.Innovative Techniques :

Highlight any innovative or unique approaches you used during the development of your project.

Program :

```
import pandas as pd

import numpy as np

from tensorflow.keras.layers import Input, Embedding,
Flatten, Concatenate, Dense

from tensorflow.keras.models import Model

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_squared_error
```

```
# Load the preprocessed dataset (replace
'your_dataset.csv' with the actual dataset file)

data = pd.read_csv('your_dataset.csv')

# Define the features and target variable

categorical_features = ['location', 'time_of_day']

numeric_features = ['feature1', 'feature2', 'feature3']

target = 'earthquake_magnitude'

# Perform feature scaling on numeric features

scaler = StandardScaler()

data[numeric_features] =
scaler.fit_transform(data[numeric_features])

# Create embeddings for categorical features

inputs = []

embeddings = []

for feature in categorical_features:

    input_layer = Input(shape=(1,))

    embedding_layer =
Embedding(input_dim=len(data[feature].unique()),
output_dim=10)(input_layer)

    embedding_layer = Flatten()(embedding_layer)
    inputs.append(input_layer)

    embeddings.append(embedding_layer)

# Concatenate embeddings and numeric features

all_inputs = inputs +
[Input(shape=(len(numeric_features),))]

x = Concatenate()(embeddings + [all_inputs])

# Build a neural network

x = Dense(128, activation='relu')(x)

output_layer = Dense(1)(x)

model = Model(inputs=inputs + [all_inputs],
outputs=output_layer)

model.compile(optimizer='adam',
loss='mean_squared_error')

# Prepare the data for training

X = [data[feature] for feature in categorical_features]
+ [data[numeric_features]]

y = data[target]

X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)

# Train the neural network
```

```
model.fit(X_train, y_train, epochs=50, batch_size=64,
validation_data=(X_test, y_test))

# Make predictions

y_pred = model.predict(X_test)

# Calculate the mean squared error

mse = mean_squared_error(y_test, y_pred)

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

9.Conclusion :

In conclusion, our earthquake prediction model is a step forward in addressing a complex and significant issue. While we acknowledge the limitations and challenges, we remain committed to improving our model and working towards more accurate and reliable earthquake predictions. This project has cooperation in mitigating the impact of seismic events on our communities and infrastructure.

REFERENCES :

1. https://github.com/akash-r34/Earthquake-prediction-using-Machine-learning-models

2. https://www.researchgate.net/publication/313858017_Machine_Learning_Predicts_Laboratory_Earthquakes

3. http://earthquake.usgs.gov/data/comcat/data-eventterms.php#time

4. DriveData, Richter’s Predictor: Modelling Earthquake Damage (2021).

5. Sameer, Earthquake History (1965–2016): Data Visualization and Model Development.

6. World Health Organization, Earthquakes, Health topics.