Name: JAYAMOHANRAMAN K
Email ID: jayamohanjayamohan520@gmail.com
NM ID. : au513521106008
Phase 5 : Problem Thinking and Development Part

Earthquake Prediction Model Using Python

Model Design Thinking Part

- **INTRODUCTION**
  - Earthquakes, among the most devastating natural disasters, strike with little warning, leaving communities vulnerable and in need of proactive measures.
  - In the age of data science and machine learning, we embark on a journey to harness the power of technology for early earthquake prediction.
  - earthquake prediction is a highly specialized and challenging field, and more advanced techniques and domain-specific knowledge may be required for meaningful results. Additionally, ethical considerations and expert consultation are crucial when working onsuch critical and potentially life-saving.

- **PROBLEM DESCRIPTION**
  - Earthquakes are natural disasters that can Cause significant damage to life and property.
  - The goal is to build a machine learning model that can Predict the occurrence of an earthquake based on various Features and historical earthquake data. These features may Include geographical location, depth, magnitude, time.
  - The model should be able to analyze the patterns and Trends in earthquake data and learn from the historical Occurrences to make predictions about future earthquakes.
  - To build the earthquake prediction model, you will need a Dataset containing information about past earthquakes.

- The Dataset should include features like latitude and longitude, Magnitude, depth, date and time, and any other relevant data.
- Additionally, you may want to consider incorporating real-Time data from seismic sensors to make the predictions more accurate and up-to-date.
- The objective is to develop a reliable and accurate earthquake prediction model using Python that can assist in Disaster management and preparedness efforts.

Design thinking

- Design thinking is a problem-solving approach that focuses on Understanding users' needs, generating innovative solutions,And iterating on those solutions through testing and feedback.

Data Source Selection:

- Choosing the right dataset is a critical first step. Look for a Kaggle dataset that contains comprehensive earthquake data with features like date, time, latitude, longitude, depth, and magnitude.
- Ensure that the dataset is up-to-date and relevant to your predictive modelling goals.

Visualization Enhancement:

- Enhance your world map visualization to make it more informative and interactive:
- Color-Coding: Assign different colors or markers to earthquake locations based on their magnitudes. This visual representation provides a quick understanding of the severity of earthquakes in different regions.
- Interactive Filters: Implement interactive tools that allow users to filter earthquake data by various attributes, such as depth, time range, or magnitude range.

Data Preprocessing:

- Data preprocessing is essential to ensure the quality and integrity of your dataset. This step involves several tasks:

- Handling Missing Values: Identify and address missing values in the dataset. Depending on the extent of missing data, you may choose to impute values or remove rows with missing information.
- Outlier Detection: Detect and handle outliers that could skew your analysis and model. Techniques like Z-score or IQR (Interquartile Range) can be used for outlier identification and treatment.
- Data Type Conversion: Ensure that data types are appropriate for analysis. For example, convert date and time columns to datetime objects for time series analysis.

- **Feature Exploration:**

    - Delve deep into feature exploration to gain insights into the earthquake data.
    - Distribution Analysis: Examine the statistical distribution of key features like magnitude, depth, and geographical coordinates (latitude and longitude). Histograms, box plots, and summary statistics can be helpful.
    - Correlation Analysis: Explore correlations between different features. For instance, investigate how depth correlates with magnitude or whether earthquake occurrences exhibit temporal trends.
    - Characteristics Assessment: Understand the characteristics of earthquakes in your dataset, such as the frequency of small and large earthquakes, their spatial distribution, and variations over time.

- **Exploratory Data Analysis (EDA):**

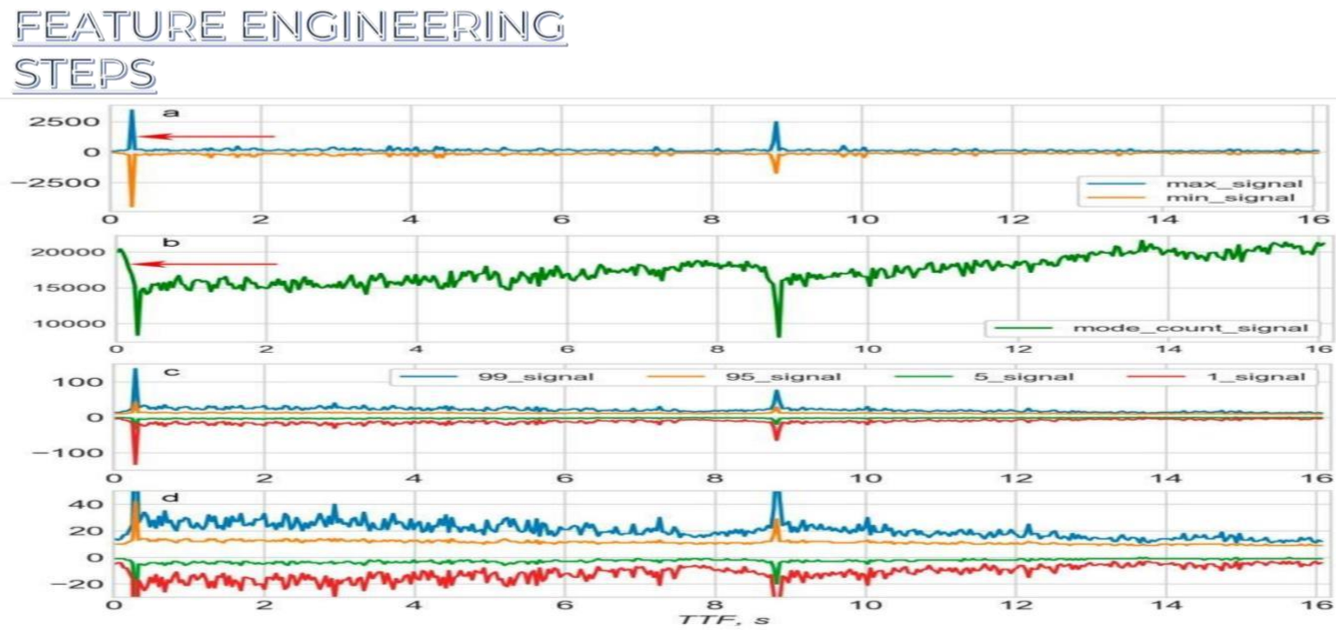
    - Dive into exploratory data analysis to uncover patterns and insights:
    - Time Series Analysis: If your dataset spans multiple years, analyze the time series data to identify trends, seasonal variations, and potential cyclical patterns in earthquake occurrences.
    - Visualization: Utilize various data visualization techniques, including line plots, bar charts, scatter plots, and heatmaps, to visualize the data and discover hidden relationships.

- **Feature Selection:**

  - Feature selection is crucial for model efficiency and interpretability:
  - Feature Importance: Use techniques like feature importance scores (e.g., from decision trees or random forests) to prioritize the most relevant features for prediction. This step can help reduce dimensionality.
  - Correlation Matrix: Create a correlation matrix to identify highly correlated features. Consider removing one of a pair of strongly correlated features to reduce multicollinearity.

- **Hyperparameter tuning:**

  - When you're training machine learning models, each dataset and model needs a different set of hyperparameters, which are a kind of variable. ○ The only way to determine these is through multiple experiments, where you pick a set of hyperparameters and run them through your model.
  - In essence, you're training your model sequentially with different sets of hyperparameters.
  - This process can be manual, or you can pick one of several automated hyperparameter tuning methods.



1. **Data Collection**: Obtain historical earthquake data from reliable sources, such as the earthquake data from kaggle or other relevant organizations. This data should include earthquake magnitudes, locations, depths, and timestamps.

2. **Feature Engineering:**

- **Spatial features:** Calculate distance or proximity to know fault lines, tectonic plate boundaries, or other geological features that may be correlated with earthquake occurrence.

- **Historical features:** Create lag features, such as earthquake occurrences in the past, to capture temporal dependencies.

- **Statisical features**: Compute statistics (mean, standard deviation, etc.) for earthquake magnitudes and depths within specific time windows or regions.

- **Geospatical features:** Utilize geographic information system (GIS) data to include features like elevation, soil type, or land use, which can affect seismic activity.

3. **Data Splitting:** Split the dataset into training, validation, and test sets. Typically, you'll use a larger portion for training and smaller portions for validation and testing.

4. **Model Evaluation**: Evaluate your model's performance on the validation set using appropriate evaluation metrics, such as mean squared error (MSE), mean absolute error (MAE), or area under the ROC curve (AUC), depending on the nature of the prediction problem (regression or classification).

5. **Deployment**: If your model performs satisfactorily, you can deploy it for real-time or nearreal-time earthquake prediction. However, note that earthquake prediction is a challenging problem, and even the best models may have limited accuracy.

6. **Monitoring and Maintenance:** Continuously monitor and update your model as new earthquake data becomes available to ensure its accuracy and reliability.

Model Development Part

1. Load data in Pandas.
2. Drop columns that aren't useful.
3. Drop rows with missing values.
4. Create dummy variables.
5. Take care of missing data.
6. Convert the data frame to NumPy.

1.Load data in Pandas:

To work on the data, you can either load the CSV in Excel or in Pandas. For the purposes of this tutorial, we'll load the CSV data in Pandas.

```
[ ]  import pandas as pd
     df = pd.read_csv("database.csv")
```

Let's take a look at the data format below:

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23412 entries, 0 to 23411
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        23412 non-null  object
 1   Time                        23412 non-null  object
 2   Latitude                    23412 non-null  float64
 3   Longitude                   23412 non-null  float64
 4   Type                        23412 non-null  object
 5   Depth                       23412 non-null  float64
 6   Depth Error                 4461 non-null   float64
 7   Depth Seismic Stations      7097 non-null   float64
 8   Magnitude                   23412 non-null  float64
 9   Magnitude Type              23409 non-null  object
 10  Magnitude Error             327 non-null    float64
 11  Magnitude Seismic Stations  2564 non-null   float64
 12  Azimuthal Gap               7299 non-null   float64
 13  Horizontal Distance         1604 non-null   float64
 14  Horizontal Error            1156 non-null   float64
 15  Root Mean Square            17352 non-null  float64
 16  ID                          23412 non-null  object
 17  Source                      23412 non-null  object
 18  Location Source             23412 non-null  object
 19  Magnitude Source            23412 non-null  object
 20  Status                      23412 non-null  object
dtypes: float64(12), object(9)
memory usage: 3.8+ MB
```

2. **Drop Columns That Aren't Useful:** Let's try to drop some of the columns which won'tcontribute much to our machine learning model. We'll start with Date and Time.

```
[ ] cols=['Date','Time']
    df=df.drop(cols, axis=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23412 entries, 0 to 23411
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Latitude                    23412 non-null  float64
 1   Longitude                   23412 non-null  float64
 2   Type                        23412 non-null  object
 3   Depth                       23412 non-null  float64
 4   Depth Error                 4461 non-null   float64
 5   Depth Seismic Stations      7097 non-null   float64
 6   Magnitude                   23412 non-null  float64
 7   Magnitude Type              23409 non-null  object
 8   Magnitude Error             327 non-null    float64
 9   Magnitude Seismic Stations  2564 non-null   float64
 10  Azimuthal Gap               7299 non-null   float64
 11  Horizontal Distance         1604 non-null   float64
 12  Horizontal Error            1156 non-null   float64
 13  Root Mean Square            17352 non-null  float64
 14  ID                          23412 non-null  object
 15  Source                      23412 non-null  object
 16  Location Source             23412 non-null  object
 17  Magnitude Source            23412 non-null  object
 18  Status                      23412 non-null  object
dtypes: float64(12), object(7)
memory usage: 3.4+ MB
```

3. **Drop Rows With Missing Values:** Next we can drop all rows in the data that have missing values (NaNs). Here's how:

```
[ ] df=df.dropna()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14 entries, 565 to 22238
Data columns (total 19 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Latitude                    14 non-null     float64
 1   Longitude                   14 non-null     float64
 2   Type                        14 non-null     object
 3   Depth                       14 non-null     float64
 4   Depth Error                 14 non-null     float64
 5   Depth Seismic Stations      14 non-null     float64
 6   Magnitude                   14 non-null     float64
 7   Magnitude Type              14 non-null     object
 8   Magnitude Error             14 non-null     float64
 9   Magnitude Seismic Stations  14 non-null     float64
 10  Azimuthal Gap               14 non-null     float64
 11  Horizontal Distance         14 non-null     float64
 12  Horizontal Error            14 non-null     float64
 13  Root Mean Square            14 non-null     float64
 14  ID                          14 non-null     object
 15  Source                      14 non-null     object
 16  Location Source             14 non-null     object
 17  Magnitude Source            14 non-null     object
 18  Status                      14 non-null     object
dtypes: float64(12), object(7)
memory usage: 2.2+ KB
```

4. Creating Dummy Variables

Instead of wasting our data, let's convert the Latitude and Longitude to columns in Pandas and drop them after conversion.

```
[ ] dummies=[]
    cols=['Latitude', 'Longitude']
    for col in cols:
      dummies.append(pd.get_dummies(df[col]))
```

```
database_dummies=pd.concat(dummies, axis=1)
```

Finally we **concatenate** to the original data frame, column-wise:

```
df=pd.concat((df,database_dummies), axis=1)
```

Now that we use converted Latitude and Longitude values into columns, we drop the redundant columns

From the data frame.

```
df=df.drop(['Latitude', 'Longitude'], axis=1)
```

Let's take a look at the new data frame:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14 entries, 565 to 22238
Data columns (total 45 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Type                        14 non-null     object
 1   Depth                       14 non-null     float64
 2   Depth Error                 14 non-null     float64
 3   Depth Seismic Stations      14 non-null     float64
 4   Magnitude                   14 non-null     float64
 5   Magnitude Type              14 non-null     object
 6   Magnitude Error             14 non-null     float64
 7   Magnitude Seismic Stations  14 non-null     float64
 8   Azimuthal Gap               14 non-null     float64
 9   Horizontal Distance         14 non-null     float64
 10  Horizontal Error            14 non-null     float64
 11  Root Mean Square            14 non-null     float64
 12  ID                          14 non-null     object
 13  Source                      14 non-null     object
 14  Location Source             14 non-null     object
 15  Magnitude Source            14 non-null     object
 16  Status                      14 non-null     object
 17  18.045                      14 non-null     uint8
 18  30.25                       14 non-null     uint8
 19  37.2315                     14 non-null     uint8
 20  37.245                      14 non-null     uint8
 21  37.2788333                  14 non-null     uint8
 22  37.2901667                  14 non-null     uint8
 23  37.2953333                  14 non-null     uint8
 24  37.2965                     14 non-null     uint8
 25  37.3005                     14 non-null     uint8
 26  37.3021667                  14 non-null     uint8
 27  37.3141667                  14 non-null     uint8
 28  38.1383333                  14 non-null     uint8
 29  41.1444                     14 non-null     uint8
 30  46.2073333                  14 non-null     uint8
```

```
 31   -122.188                    14 non-null     uint8
 32   -118.3913333                14 non-null     uint8
 33   -116.5341667                14 non-null     uint8
 34   -116.4736667                14 non-null     uint8
 35   -116.4606667                14 non-null     uint8
 36   -116.4556667                14 non-null     uint8
 37   -116.4115                   14 non-null     uint8
 38   -116.4083333                14 non-null     uint8
 39   -116.3686667                14 non-null     uint8
 40   -116.346                    14 non-null     uint8
 41   -116.3331667                14 non-null     uint8
 42   -114.8721                   14 non-null     uint8
 43   -114.8                      14 non-null     uint8
 44   -68.3509                    14 non-null     uint8
dtypes: float64(10), object(7), uint8(28)
memory usage: 2.4+ KB
```

Lets compute with interpolate() with the missing values and finding the data data of values to interpolate.

```
df['Type']=df['Type'].interpolate()
```

4. Take Care of Missing Data

Now let's observe the data columns. Notice close is now interpolated with imputed new values.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14 entries, 565 to 22238
Data columns (total 45 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Type                        14 non-null     object
 1   Depth                       14 non-null     float64
 2   Depth Error                 14 non-null     float64
 3   Depth Seismic Stations      14 non-null     float64
 4   Magnitude                   14 non-null     float64
 5   Magnitude Type              14 non-null     object
 6   Magnitude Error             14 non-null     float64
 7   Magnitude Seismic Stations  14 non-null     float64
 8   Azimuthal Gap               14 non-null     float64
 9   Horizontal Distance         14 non-null     float64
 10  Horizontal Error            14 non-null     float64
 11  Root Mean Square            14 non-null     float64
 12  ID                          14 non-null     object
 13  Source                      14 non-null     object
 14  Location Source             14 non-null     object
 15  Magnitude Source            14 non-null     object
 16  Status                      14 non-null     object
 17  18.045                      14 non-null     uint8
 18  30.25                       14 non-null     uint8
 19  37.2315                     14 non-null     uint8
 20  37.245                      14 non-null     uint8
 21  37.2788333                  14 non-null     uint8
 22  37.2901667                  14 non-null     uint8
 23  37.2953333                  14 non-null     uint8
 24  37.2965                     14 non-null     uint8
 25  37.3005                     14 non-null     uint8
 26  37.3021667                  14 non-null     uint8
 27  37.3141667                  14 non-null     uint8
 28  38.1383333                  14 non-null     uint8
 29  41.1444                     14 non-null     uint8
 30  46.2073333                  14 non-null     uint8
 31  -122.188                    14 non-null     uint8
 32  -118.3913333                14 non-null     uint8
 33  -116.5341667                14 non-null     uint8
 34  -116.4736667                14 non-null     uint8
 35  -116.4606667                14 non-null     uint8
 36  -116.4556667                14 non-null     uint8
 37  -116.4115                   14 non-null     uint8
 38  -116.4083333                14 non-null     uint8
 39  -116.3686667                14 non-null     uint8
 40  -116.346                    14 non-null     uint8
 41  -116.3331667                14 non-null     uint8
 42  -114.8721                   14 non-null     uint8
 43  -114.8                      14 non-null     uint8
 44  -68.3509                    14 non-null     uint8
dtypes: float64(10), object(7), uint8(28)
memory usage: 2.4+ KB
```

6. Convert the Data Frame to NumPy: Now that we've converted all the data to integers, it's time to prepare the data for machine learning models. This is where scikit-learn and

NumPy come into play: X= Input set with 14 attributes y = Small y output, in this case

Now we convert our data frame from Pandas to NumPy and we assign input and output:

```
X=df.values
y=df['Root Mean Square'].values
```

```
import numpy as np
X=np.delete(x, 1, axis=1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Development Part 2

GIVEN DATASET :

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23407	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	1.2	40.0	5.6	ML	...	18.0	42.47	0.120	NaN
23408	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	2.0	33.0	5.5	ML	...	18.0	48.58	0.129	NaN
23409	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	1.8	NaN	5.9	MWW	...	NaN	91.00	0.992	4.8
23410	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	1.8	NaN	6.3	MWW	...	NaN	26.00	3.553	6.0
23411	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	2.2	NaN	5.5	MB	...	428.0	97.00	0.681	4.5

23412 rows × 21 columns

Overview of the process :

The following is an overview of the process of building a earthquake prediction model used by feature selection, model training, and evaluation.

1. Prepare the data:

This includes cleaning the data, removing outliers, and handling missing values.

2. Perform feature selection :

This can be done using a variety of methods, such as correlation analysis, information gain, and recursive features elimination.

3. Train the model :

There are many different ML algorithms that can be used for earthquake prediction. Some popular algorithms are linear regression, random forests, SVR.

4. Evaluate the model :

This can be done by calculating the mean squared error(MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.

5. Deploy the model :

Once the model has been evaluating and found to be performing well, it can be deployed to production so that it can be used to predict the earthquake.

Feature Selection :

Checking for missing values

In[1]:

```
print("Missing values") print("-"

*30) print(df.isna().sum())

print("-"*30)

print("Total missing values", df.isna().sum())
```

Out[1]:

```
Missing values
------------------------------
Date                           0
Time                           0
Latitude                       0
Longitude                      0
Type                           0
Depth                          0
Depth Error                18951
Depth Seismic Stations     16315
Magnitude                      0
Magnitude Type                 3
Magnitude Error            23085
Magnitude Seismic Stations 20848
Azimuthal Gap              16113
Horizontal Distance        21808
Horizontal Error           22256
Root Mean Square            6060
ID                             0
Source                         0
Location Source                0
Magnitude Source               0
Status                         0
dtype: int64
------------------------------
Total missing values 145439
```

Model Training :

1. Choose a machine learning algorithm :

There are a number of different machine learning algorithm that can be for earthquake prediction, such as linear regression, lasso regression, decision trees, and random forests are covered.

Machine Learning Models:

In[2]:

new_row = {"Model": "Ridge", "MAE":mae, "MSE": mse,"RMSE":rmse, "R2 Score": r_squared, "RMSE(Cross-Validation)":rmse_cross_val}

models = models.append(new_row, ignore_index=True)

In[3]:

```
def evaluation(y_true, y_pred):

    # calculate MAE

    mae = mean_absolute_error(y_true, y_pred)

    # calculate MSE

    mse = mean_squared_error(y_true, y_pred)

    # calculate RMSE

    rmse = np.sqrt(mse) rmse_cross_val=np.mean(rmse)

    r_squared = r2_score(y_true, y_pred)

    # return the four metrics as a tuple

    return mae, mse, rmse, r_squared, rmse_cross_val
```

Linear Regression :

In[4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) lin_reg = LinearRegression() lin_reg.fit(X_train,
y_train) predictions = lin_reg.predict(X_test) mae, mse, rmse,
r_squared,rmse_cross_val = evaluation(y_test, predictions)
print("MAE:",mae)
print("MSE:",mse)
print("RMSE:",rmse) print("R2
Score:",r_squared) print("-"
*30)
```

print("RMSE Cross-Validation:",rmse_cross_val)

Out[4]:

```
MAE: 16.214208564591
MSE: 413.6507308565237
RMSE: 20.33840531744128
R2 Score: -0.15997292842810484
------------------------------
RMSE Cross-Validation: 20.33840531744128
```

Elastic Net:

In[7]:

elasticnet = ElasticNet()
elasticnet.fit(X_train, y_train) predictions =

elasticnet.predict(X_test) mae,mse, rmse,

r_squared,rmse_cross_val = evaluation(y_test, predictions)

print("MAE:",mae)

print("MSE:",mse)

print("RMSE:",rmse) print("R2

Score:",r_squared) print("-"

*30)

print("RMSE Cross-Validation:",rmse_cross_val)

Out[7]:

```
MAE: 10.872423700794576
MSE: 195.23917220459506
RMSE: 13.972801158128425
R2 Score: 0.4525039183247668
------------------------------
RMSE Cross-Validation: 13.972801158128425
```

Support Vector Machines:

In[8]:

svr = SVR(C=100000)

svr.fit(X_train,y_train) predictions = svr.predict(X_test)

mae,mse, rmse, r_squared,rmse_cross_val = evaluation(y_test,

predictions) print("MAE:",mae)

print("MSE:",mse)

print("RMSE:",rmse) print("R2

Score:",r_squared) print("-"

*30)

print("RMSE Cross-Validation:",rmse_cross_val)

Out[8]:

```
MAE: 60.364276908953464
MSE: 3877.4242177347583
RMSE: 62.26896673090664
R2 Score: -9.873199994813712
------------------------------
RMSE Cross-Validation: 62.26896673090664
```

Random Forest Regressor:

In[9]:

random_forest = RandomForestRegressor(n_estimators=

100) random_forest.fit(X_train, y_train)

predictions = random_forest.predict(X_test)

mae,mse, rmse, r_squared,rmse_cross_val = evaluation(y_test, predictions)
print("MAE:",mae)

```
print("MSE:",mse)

print("RMSE:",rmse) print("R2

Score:",r_squared) print("-"

*30)

print("RMSE Cross-Validation:",rmse_cross_val)
```

Out[9]:

```
MAE: 10.295796132468222
MSE: 198.72930732017593
RMSE: 14.097138267044697
R2 Score: 0.44271676711570895
------------------------------
RMSE Cross-Validation: 14.097138267044697
```

Polynomial Regression (Degree= 2):

In[10]:

```
poly_reg = PolynomialFeatures(degree =2)

X_train_2d = poly_reg.fit_transform(X_train)

X_test_2d = poly_reg.transform(X_test) lin_reg

= LinearRegression() lin_reg.fit(X_train_2d,

y_train)

predictions = lin_reg.predict(X_test_2d)

mae,mse, rmse, r_squared,rmse_cross_val = evaluation(y_test, predictions)

print("MAE:",mae)

print("MSE:",mse)

print("RMSE:",rmse)
```

print("R2 Score:",r_squared) print("-"

*30)

print("RMSE Cross-Validation:",rmse_cross_val)

Out[10]:

```
MAE: 39.11674027433722
MSE: 1563.7117106065875
RMSE: 39.5437948432695
R2 Score: -3.3850115976195143
------------------------------
RMSE Cross-Validation: 39.5437948432695
```

Model Training :

- Model training is the process of teaching a machine learning model to predict earthquake.
- Once the model is trained, it can be used to predict earthquake for new data.

1. Prepare the data.
2. Split the data into training and test sets.
3. Choose a machine learning algorithm.
4. Tune the hyperparameters of the algorithm.
5. Train the model on the training set.
6. Evaluate the model on the test set.

Split the data into train and test :

In[11]:

X = df[['Latitude', 'Longitude', 'Magnitude','Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap', 'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'Depth Error']]

Y = df['Depth']

In[12]:

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

random_state=42)

In[13]:

y_train.head()

Out[13]:

```
count    23412.000000
max        700.000000
std        122.651898
25%         14.522500
min         -1.100000
Name: Depth, dtype: float64
```

In[14]:

y_train.shape

Out[14]:

```
(18729,)
```

In[15]:

y_test.head()

Out[15]:

```
mean    70.767911
50%     33.000000
Name: Depth, dtype: float64
```

In[16]:

Y_test.shape

Out[16]:

```
(4683,)
```

Model Evaluation:

- It is the process of assessing the performance of a machine learning model on the unseen data.
- There are a number of different metrices that can be used to evaluate the performance of a earthquake prediction model.

Some of the most common metrics are:

- Mean Squared Error(MSE):
- Root Mean Squared Error(RMSE):
- Mean Absolute Error:
- R-Squared:

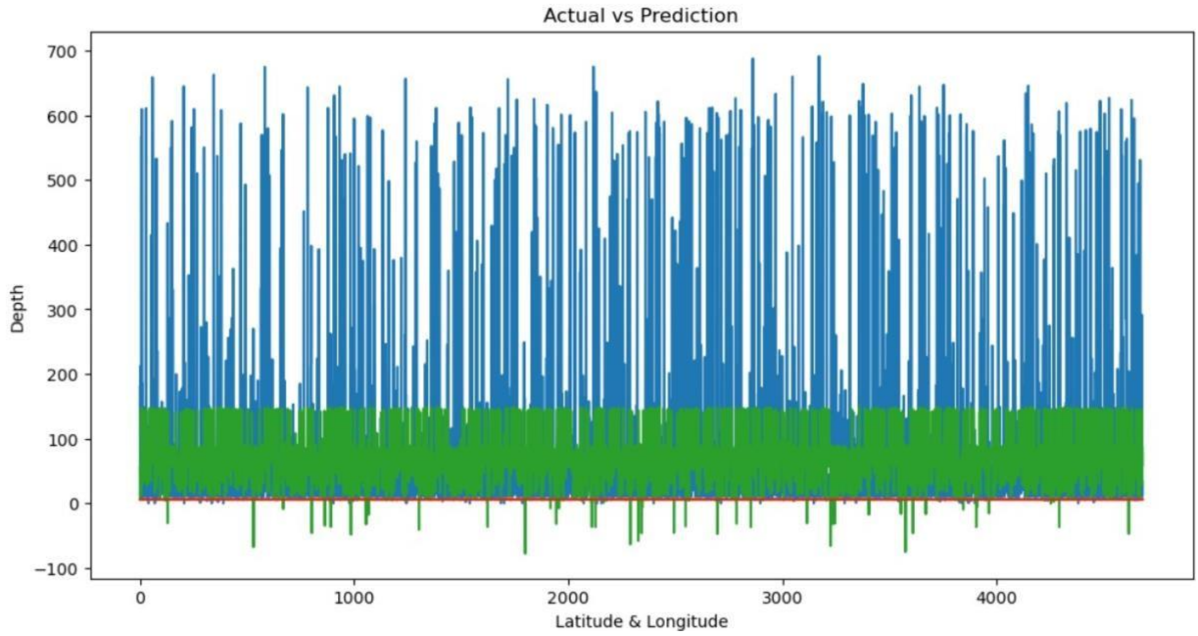
Evaluation of Predicted Data :

In[17]:

```
plt.figure(figsize=(12,6))

plt.plot(np.arange(len(y_test)), y_test)

plt.plot(np.arange(len(y_test)),predictions

) plt.xlabel("Latitude & Longitude")

plt.ylabel("Depth") plt.title("Actual vs

Prediction")
```

Out[17]:

```
Text(0.5, 1.0, 'Actual vs Prediction')
```

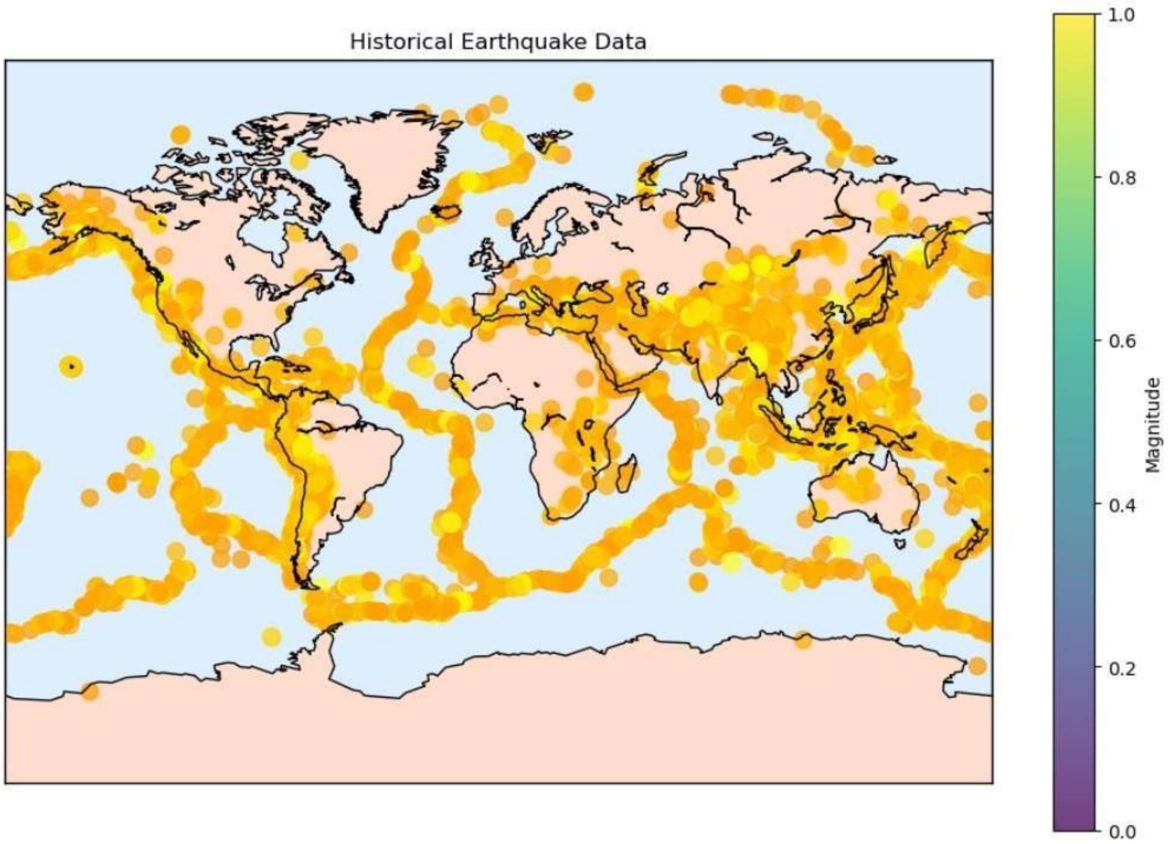


In[18]:

```
lons = df["Longitude"]

lats = df["Latitude"] mags

= df["Magnitude"]

depths = df["Depth"]

fig, ax = plt.subplots(figsize=(12,8))

m = Basemap(projection="mill", llcrnrlat=-90, urcrnrlat=90, llcrnrlon=-180,

          urcrnrlon=180, resolution="c")

m.drawcoastlines()

m.fillcontinents(color="#FFDDCC", lake_color="#DDEEFF")

m.drawmapboundary(fill_color="#DDEEFF")
x,y = m(lons, lats)

cmap = plt.get_cmap("hot")

colors = [cmap(i / max(mags)) for i in mags]
```

```
m.scatter(x, y, marker="o", c=colors, s=[i * 15 for i in mags], alpha=0.75)

plt.colorbar(label="Magnitude")  plt.title("Historical  Earthquake  Data")

plt.show()
```

Out[18]:



<u>Feature Engineering :</u>

It is a crucial aspect of predicting earthquake model using machine learning. It involves creating new features, transforming existing ones, and selecting the most relevant variables to improve the model's predictive power. Here are some feature engineering ideas for earthquake prediction.

1. Auto-recognition of diurnal periodic waveform:

These are electromagnetic disturbances (ED) that synchronize with sunrise and sunset. They can be used to filter out the background noise and focus on the anomalous signals that may precede earthquakes.

2. Higuchi Fractal Dimension:

This is a measure of the complexity or irregularity of a time series. It can be used to capture the non-linear features of ED data and quantify the degree of chaos or order in the system. A higher fractal dimension indicates a more chaotic system, which may imply a higher probability of earthquake occurrence.

3. Sliding interquartile range:

This is a robust measure of variability or dispersion in a time series. It can be used to detect outliers or spikes in ED data that may indicate seismic precursors.

4. Geo- sound:

This is the sound generated by the movement of tectonic plates or faults. It can be measured by microphones or acoustic sensors and can provide information about the stress state and deformation of the crust.

<u>Various features of perform model training :</u>

1. **Seismic waveforms:**
    - These are the signals recorded by seismometers that measure the ground motion caused by earthquakes.
    - They can be used to extract features such as amplitude, frequency, duration, phase, and polarity of the waves, which can

indicate the location, magnitude, and mechanism of the earthquake.

- Seismic waveforms can also be transformed into different domains, such as time-frequency, wavelet, or spectral, to capture more information.

2. Earthquake catalog:
    - This is a collection of historical earthquake data that includes parameters such as date, time, latitude, longitude, depth, magnitude, and fault type of each event.
    - Earthquake catalog can be used to analyze the spatial and temporal patterns of seismic activity, such as clustering, recurrence intervals, and aftershock sequences.

3. Environmental factors:

- These are the external factors that may have an impact on earthquake occurrence or detection.
- Environmental factors include parameters such as temperature, pressure, humidity, precipitation, wind speed, solar radiation, and geomagnetic field.
- Environmental factors can be measured by various sensors or instruments, such as thermometers, barometers, hygrometers, rain gauges, anemometers, pyranometers, and magnetometers.

Conclusion :

- Earthquake prediction is a challenging and important task that aims to forecast the occurrence, location, magnitude, and impact of future earthquakes based on various types of data and models.
- Earthquake prediction can help reduce the loss of life and property, improve the preparedness and resilience of communities, and advance the scientific understanding the earth's processes.

- Earthquake catalog may be biased, incomplete, or inaccurate due to different reporting standards, detection thresholds, or measurement methods.
- Earthquake models are often based on simplifying assumptions, approximations, or empirical rules that may not capture the true physics or statistics of the earthquake phenomenon.
- Earthquake prediction is not a perfect science but a continuous learning process that requires collaboration, innovation, and evaluation.
- By improving the data quality and availability, developing more realistic and robust models, enhancing the prediction accuracy and uncertainty quantification, and considering the ethical and social implications, earthquake prediction can become more feasible and beneficial for society.