Earthquake Prediction Model using python

Name:SUBASH.K

Reg.No:513521104304

Department: CSE

Year: III

Nm ID: au513521104304

1. Mail: [subashpck@gmail.com](mailto:subashpck@gmail.com)

Phase 5 (Final Submission)

AI\_PHASE5

**1. ABSTRACT**

* An earthquake is shaking of the surface of the Earth, which caused as the result of movable plate boundary interactions .Earthquakes are measured using remarks from seismometers with Richter magnitude scale. Ground rupture, Landslides, Soil liquefaction and Tsunami are the main effects created by earthquakes.
* Today'searthquake warning systems used to provide regional notification of an earthquake in progress. Many methods have been alreadydeveloped for predicting the time and place in which earthquake will occur, but it did not predicted using big data analytics.

**Keywords**: Earthquake, Seismic waves, Seismometer, Richter magnitude scale, Tsunami, Earthquake warning systems, Big Data, Map Reduce.

1. **INTRODUCTION**

* Today, big data analytics is one of the most booming markets.

When Google search engine launched image search feature, it had indexed more than 300 million images. In every minute so

many video content are uploaded in YouTube update their wall in every minute. Search engines logging 600 million queries daily.

1. **EXISTING SYSTEM**

* The existing system addresses novel methodology to predict next earthquake. Apache hadoop is designed to run in a

distributed environment and it manages the collection of various nodes running map and reduce function. In this system data analysis performed on earthquake data in year wise and location wise.

* Earthquakes have always caused incalculable damage tostructures and properties and caused the deaths of millions of people throughout the world.

# PROPOSED SYSTEM

* In future, same Mapper and Reducer class implemented with pandas and matplatlib frame work components working effactive manner. Pandas handale with a data is esay that’s way proposed system run with effactively.
* visuvalation method is matplatlipframework working with overcome the exsisting system. data parsing &data format conversion is esay way to possible in pandas. Given query and get final result in graphically.
* The graphs compare the number of affected buildings (count) for a particular Damage Grade to their corresponding foundation type, roof type and ground floor type respectively The ratios indicate the likelihood of buildings with the given material’s ability to sustain damage against earthquakes.

# DESIGN AND IMPLEMENTATION

* This covers the technique and flow of events that were used to perform the prediction process. The prediction methodology itself is composed of three integral steps: data preprocessing, model selection and the final prediction process. In the dataset, a building was uniquely identified by 4

Attribute:

* Building Identification, District Identification, Municipality Identification, Ward Identification. These attributes were added to the training data for identifying the building damage grade.

**Feature engineering**

* The initial acoustic signal is decomposed into segments with 150000 rows per segment, which suggests that the training dataset has 4194 rows. Features are calculated as aggregations

over segments. For more details see, for example, here and here.

import numpy as npfrom sklearn.model\_selection import cross\_val\_scorefrom catboost import CatBoostRegressor # set output float precision np.set\_printoptions(precision=3)# init modelmodel = CatBoostRegressor(random\_seed=0, verbose=False)# calculate mae on foldsmae =

cross\_val\_score(model, data[best\_features], data['target'],

cv=5, scoring='neg\_mean\_absolute\_error', n\_jobs=8)# print the resultsprint('folds: {}'.format(abs(mae)))print('total:

{:.3f}'.format(np.mean(abs(mae))))

CatboostRegressor (without any tuning) trained on 15 features having highest importance score demonstrates mean average error 2.064. V

folds: [1.982 2.333 2.379 1.266 2.362] total:

2.064.

**ADVANCED TECHNIQUES**

1. Understand the Earthquake Phenomenon: Dive into the science behind earthquakes, exploring their causes, patterns, and the seismic data that holds valuable clues.
2. Model Development: Construct a machine learning model using Python that has the potential to forecast earthquake events. But we won't stop at the basics; we will consider advanced techniques that set this project

apart. 3. Advanced Techniques for Model Enhancement:

Delve into the realms of “hyperparameter tuning

**DEFNITION**

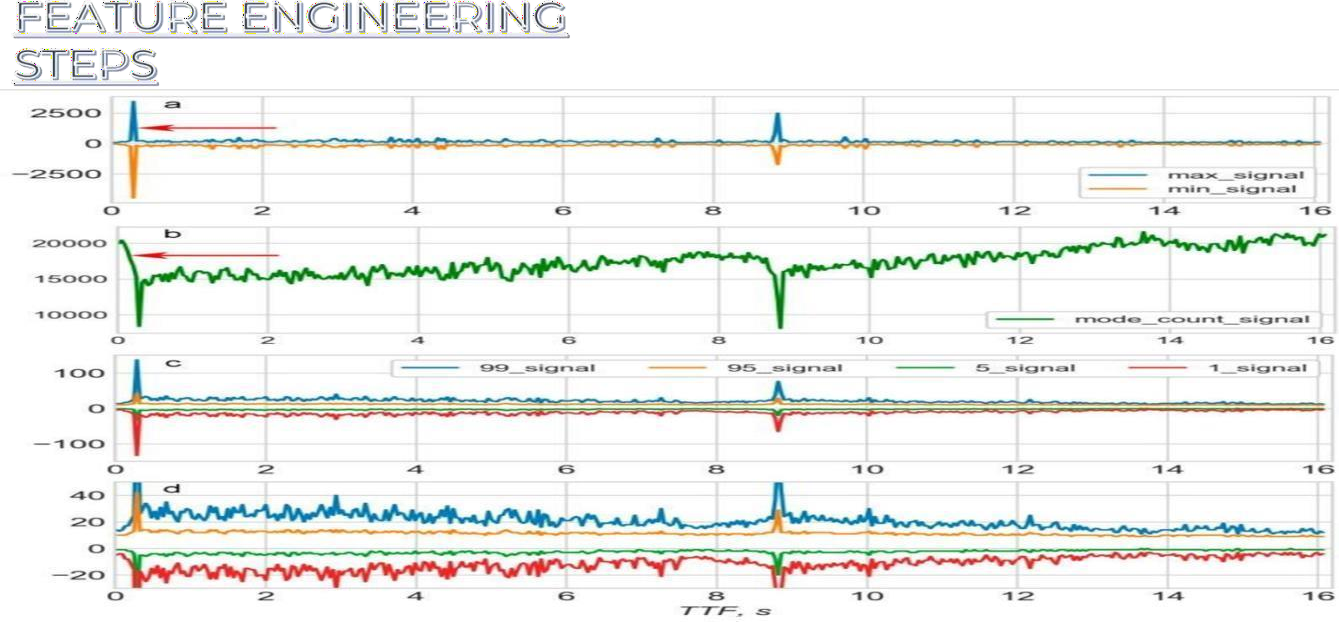
**hyperparameter tuning:**

* + 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.

**feature engineering:**

* + Feature Engineering is the process of transforming data to increase the predictive performance of machine learning models.



1. Data Collection: Obtain historical earthquake data from reliable sources, such as the United States Geological Survey (USGS) or other relevant

organizations. This data should include earthquake magnitudes, locations, depths, and timestamps.

1. Data Preprocessing: Clean and preprocess the data to remove any missing values or outliers. You may also need to convert timestamp data into a format suitable for analysis.
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.

* + temporal features: Extract temporal information,

such as the time of day, day of the week, or month, which may reveal patterns in 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

* + external data: Incorporate external data sources, such as weather data or satellite imagery, if they may have an impact on earthquake prediction.

1. 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.
2. Model Selection: Choose an appropriate machine learning or statistical model for earthquake

prediction. Common choices include decision trees, random forests, support vector machines, or deep learning models like neural networks.

1. Model Training: Train your selected model on the trainingdatausing the engineered
2. Hyperparameter Tuning: Optimize the

hyperparameters of your model using techniques like grid search or random search to improve its

1. 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).

1. Testing: Assess the model's performance on the test set to get

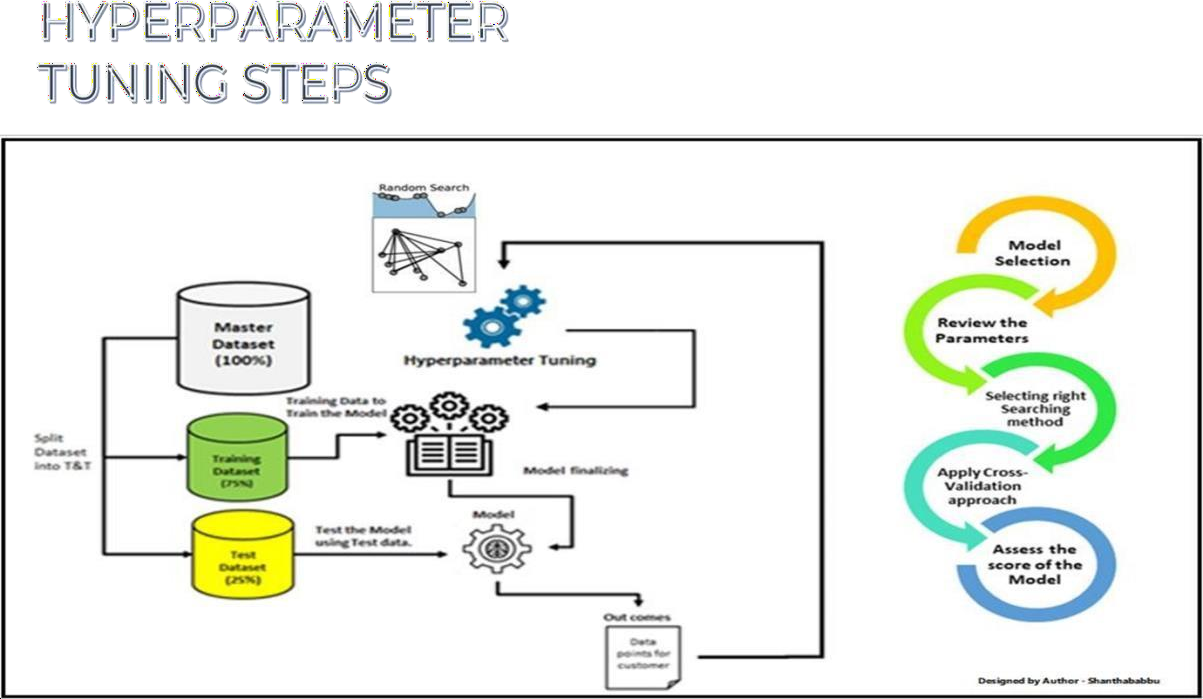
an unbiased estimate of its predictive power.

1. Deployment: If your model performs satisfactorily, you can deploy it for real-time or near-real-time earthquake prediction.

However, note that earthquake prediction is a challenging problem, and even the best models may have limited accuracy.

1. Monitoring and Maintenance: Continuously monitor and update your model as new earthquake data

becomes available to ensure its accuracy and reliability.



1. Import the necessary libraries, such as scikit- learn and any other libraries required for your specific model. Preprocess your earthquake dataset, including feature engineering, data cleaning, and splitting the data into training and validation sets.
2. Choose a Model: Select the machine learning model you want to use for earthquake prediction. Common choices include Decision Trees, Random Forest, Support Vector Machines (SVM), Gradient Boosting, or Neural Networks.
3. the Hyperparameter Grid:dictionary or a parameter grid that specifies the hyperparameters you want to tune and their

respective ranges. For example, you can define values for 'max\_depth', 'learning\_rate', 'n\_estimators', etc. Include a reasonable range of values to explore

1. Split the Data for Cross-Validation: Implement kfold cross-validation. Split your training data into multiple subsets (folds), typically using a value like k=5 or k=10. This allows you to assess your
2. Hyperparameter Search: \* Use a hyperparameter tuning technique like Grid Search or Random Search to explore different combinations of hyperparameters. Here's how to do it using scikit-learn's GridSearchCV:

# Create your model (e.g.,

DecisionTreeRegressor) model =

DecisionTreeRegressor () # Define the

hyperparameter grid param\_grid = {

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# Create GridSearchCV instance

grid\_search =

GridSearchCV(estimator=model, param\_grid=param\_grid,

scoring='neg\_mean\_squared\_error', cv=5) # Fit the model with different hyperparameter combinations grid\_search.fit(X\_train, y\_train) # Get the best

hyperparameters best\_params = grid\_search.best\_params\_

1. Evaluate Performance: After hyperparameter tuning, evaluate the model's performance on the validation set using appropriate evaluation metrics (e.g., Mean Absolute Error, Mean Squared Error).
2. Retrain the Model: Once you've found the best hyperparameters, retrain your model using the

entire training dataset (including the validation data if desired) with these optimal hyperparameters.

1. Final Evaluation: Assess the model's performance on a separate test dataset to ensure it generalizes well to new, unseen data.
2. Deploy and Monitor: If the model meets your

performance criteria, deploy it in a production or research environment. Continuously monitor its

performance and retrain as necessary with new data.

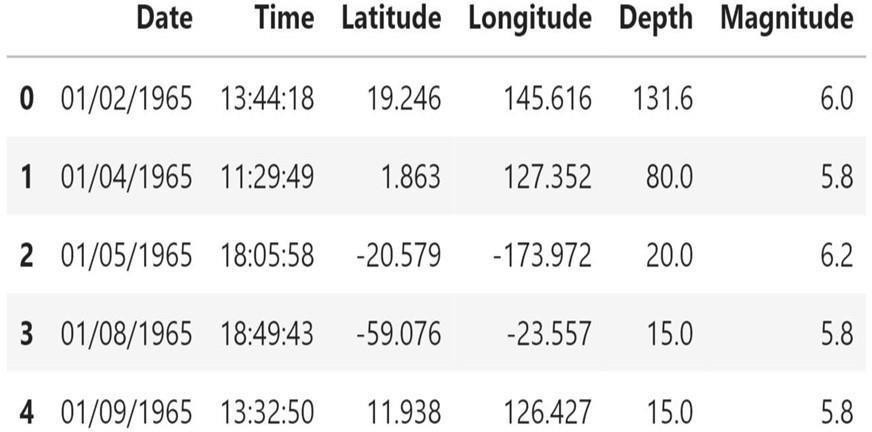
SAMPLE PROGRAM:

impo rt

num py as np impo rt

pand as aspd import matplotlib.pyplot as plt import os

output:



print(os.listdir("../input")) data=pd.read\_csv("../input/da tabase.csv") data.head()

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

**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. This is called

hyperparameter tuning. 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.

**Feature engineering:**

* Feature Engineering is the process of transforming data to increase the predictive performance of machine learning models.

1. Data Collection: Obtain historical earthquake data from reliable sources, such as the United States Geological Survey (USGS) or other relevant organizations. This data should include

earthquake magnitudes, locations, depths, and timestamps.

1. Data Preprocessing: Clean and preprocess the data to remove any missing values or outliers. You may also need to convert timestamp data into a format suitable for analysis.
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.
   * Temporal features: Extract temporal information, such as the time of day, day of the week, or month, which may reveal patterns in earthquake occurrence.
   * External data: Incorporate external data sources, such as weather data or satellite imagery, if they may have an impact on earthquake prediction.
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 Selection: Choose an appropriate machine learning or statistical model for earthquake prediction. Common choices include decision trees, random forests, support vector

machines, or deep learning models like neural networks.

* Import Libraries and Prepare Data:

1. Import the necessary libraries, such as scikit-learn and any other libraries required for your specific model. Preprocess

your earthquake dataset, including feature engineering, data cleaning, and splitting the data into training and validation sets.

1. Choose a Model: Select the machine learning model you want to use for earthquake prediction. Common choices include

Decision Trees, Random Forest, Support Vector Machines (SVM), Gradient Boosting, or Neural Networks.

1. The Hyperparameter Grid:dictionary or a parameter grid that specifies the hyperparameters you want to tune and their

respective ranges. For example, you can define values for ‘max\_depth’, ‘learning\_rate’, ‘n\_estimators’, etc. Include a reasonable range of values to explore.

1. Split the Data for Cross-Validation: Implement kfold cross- validation. Split your training data into multiple subsets (folds), typically using a value like k=5 or k=10.
2. Hyperparameter Search: \* Use a hyperparameter tuning technique like Grid Search or Random Search to explore

different combinations of hyperparameters. Here’s how to do it using scikit-learn’s GridSearchCV:

**# Create your model (e.g.,**

DecisionTreeRegressor) model =

DecisionTreeRegressor() # Define the hyperparameter grid param\_grid = {

‘max\_depth’: [None, 10, 20, 30],

‘min\_samples\_split’: [2, 5, 10],

‘min\_samples\_leaf’: [1, 2, 4]

}

# Create GridSearchCV instance

Grid\_search =

GridSearchCV(estimator=model, param\_grid=param\_grid, scoring=’neg\_mean\_squared\_error’, cv=5) # Fit the

model with different hyperparameter combinations grid\_search.fit(X\_train, y\_train) # Get the best hyperparameters best\_params =

grid\_search.best\_params\_

1. Evaluate Performance: After hyperparameter tuning, evaluate the model’s performance on the validation set using appropriate evaluation metrics (e.g., Mean Absolute Error, Mean Squared Error).
2. Retrain the Model: Once you’ve found the best

hyperparameters, retrain your model using the entire training dataset (including the validation data if desired) with these optimal hyperparameters.

1. Final Evaluation: Assess the model’s performance on a separate test dataset to ensure it generalizes well to new, unseen data.
2. Deploy and Monitor: If the model meets your performance criteria, deploy it in a production or research environment.

Continuously monitor its performance and retrain as necessary with new data.

SAMPLE PROGRAM:

Import numpy as np import pandas as

Pd import matplotlib.pyplot as plt import os

print(os.listdir(“../input”)) data=pd.read\_csv(“../input/da tabase.csv”) data.head()

Data = data[[‘Date’, ‘Time’, ‘Latitude’, ‘Longitude’, ‘Depth’,’Magnitude’]] data.head()

**Output:**

This project is not just about code and

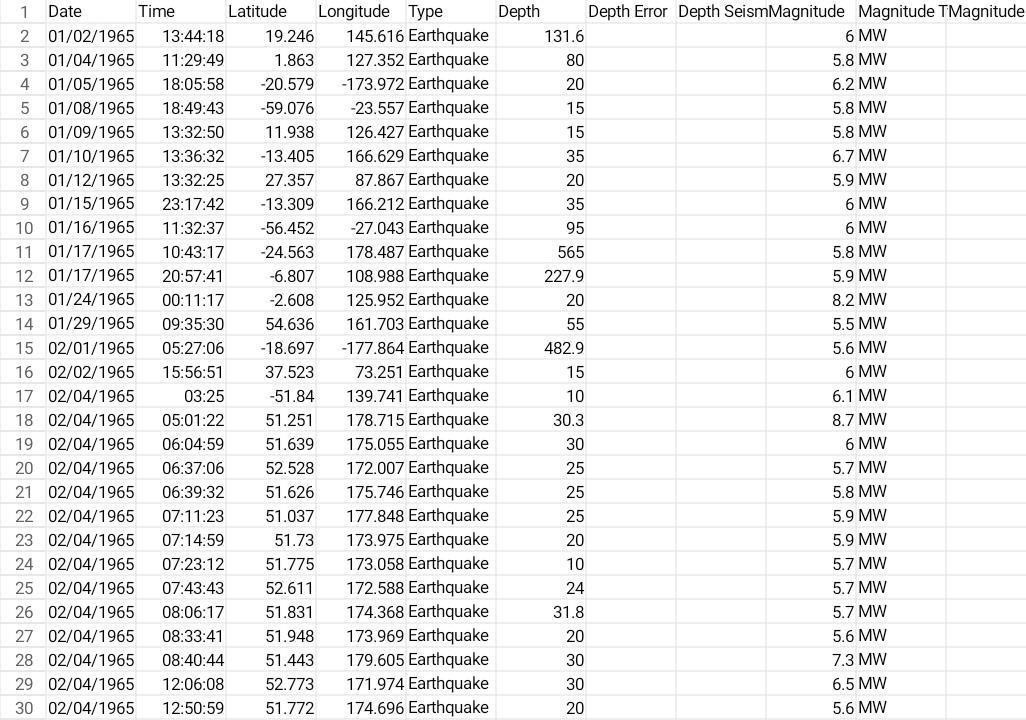
**About Dataset:**

The National Earthquake Information Center (NEIC) determines the location and size of all significant

earthquakes that occur worldwide and disseminates thise information immediately to national and international agencies, scientists, critical facilities, and the general public. international agreements. The NEIC is the national data center and archive for earthquake information.

# Content:

* This dataset includes a record of the date, time, location, depth, magnitude, and source of every earthquake with a reported magnitude 5.5 or higher since 1965.



# Necessary step to follow:

* 1. **Import Libraries:**

Start by importing the necessary libraries:

# Program:

Import pandas as pd Import numpy as np

From sklearn.model\_selection import train\_test\_split From sklearn.preprocessing import StandardScaler

# Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find

House price datasets in CSV format, but you can adapt this code to other

Formats as needed. Program:

Df = pd.read\_csv(‘ E:\USA\_Housing.csv ‘) Pd.read()

# Exploratory Data Analysis (EDA):

Perform EDA to understand your data better. This includes Checking for missing values, exploring the data’s statistics, and Visualizing it to identify patterns.

Program:

# Check for missing values Print(df.isnull().sum())

# Explore statistics Print(df.describe())

# Visualize the data (e.g., histograms, scatter plots, etc.)

# Feature Engineering:

Depending on your dataset, you may need to create new features or

Transform existing ones. This can involve one-hot encoding categorical Variables, handling date/time data, or scaling numerical features.

Program:

# Example: One-hot encoding for categorical variables

Df = pd.get\_dummies(df, columns=[‘ Avg. Area Income ‘, ‘ Avg. Area

House Age ‘])

# Split the Data:

Split your dataset into training and testing sets. This helps you evaluate

Your model’s performance later.

X = df.drop(‘price’, axis=1) # Features

Y = df[‘price’] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

Random\_state=42)

# Feature Scaling:

Apply feature scaling to normalize your data, ensuring that all Features have similar scales. Standardization (scaling to mean=0 and Std=1) is a common choice.

Program:

Scaler = StandardScaler()

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

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

# Importance of loading and processing datase:

* + 1. **Quality In, Quality Out:**

The integrity of your analysis or model depends on the quality of your data. Loading data properly ensures that you’re working with accurate and reliable information.

# Understanding Your Data:

Before diving into analysis or model building, you need a clear understanding of your data. Loading it allows you to

explore its structure, identify patterns, and gain insights into its characteristics.

# Data Cleaning:

Real-world data is often messy. Loading data lets you identify missing values, outliers, and errors that need to be addressed through cleaning processes. This is essential for meaningful and accurate results.

# Feature Engineering:

Loading data is the first step in feature engineering,

where you transform and enhance your variables to improve model performance. This can involve creating new features, scaling, or encoding categorical variables.

# Compatibility:

Different models or analysis tools may require data in

specific formats. Loading data allows you to transform it into a compatible structure, ensuring a smooth workflow.

# Efficiency:

Processing data efficiently can have a significant impact on the speed and performance of your analysis or model.

This includes tasks such as indexing, sorting, and aggregating data to streamline subsequent operations.

# Data Exploration:

Loading data allows you to visualize and explore it through various statistical and graphical methods. This

exploration is crucial for formulating hypotheses, identifying trends, and making informed decisions about further

analysis.

# Model Training:

If you’re building a machine learning model, loading and processing data is a prerequisite for training. The model’s

performance heavily depends on the quality and characteristics of the training data.

# Iterative Process:

Data loading and processing are often iterative processes. As you analyze or model, you may discover the need for additional preprocessing or adjustments. Having awell-organized and flexible data processing pipeline makes it easier to adapt.

# Reproducibility:

Properly loading and processing data contribute to the reproducibility of your work. If someone else needs to replicate your analysis or model, clear and well-documented data loading and processing steps are essential.

# Challenges involved in loading and preprocessing a earthquake prediction datasets:

**Data Volume and Size:**

* Earthquake datasets can be massive, especially if they cover long time periods and include detailed information.

Managing and loading large datasets can strain

computational resources and require efficient storage solutions.

# Data Variety:

* Earthquake data often comes in various formats,

including time- series data, geospatial data, and categorical information. Integrating and preprocessing these diverse data types can be challenging, requiring specialized techniques for each.

# Data Quality:

* Earthquake datasets may have missing or inaccurate values, outliers, or inconsistencies. Cleaning and validating the data is crucial to ensure the accuracy of predictions.

Incomplete or incorrect information can significantly impact the performance of prediction models.

# Temporal and Spatial Dependencies:

* Earthquakes exhibit temporal and spatial dependencies. Preprocessing must consider the time intervals between seismic events and the geographical relationships between data points.
* This might involve creating features that capture trends and patterns over time and space.

# Imbalanced Classes:

* The occurrence of significant earthquakes is rare compared to smaller seismic activities.
* This class imbalance can pose challenges for machine learning models, which might struggle to learn patterns associated with infrequent events. Techniques such as oversampling, undersampling, or using appropriate evaluation metrics need to be considered.

# Feature Engineering:

* Extracting meaningful features from seismic data requires domain expertise. Transforming raw sensor readings into relevant features, such as frequency components,

amplitude, and spectral characteristics, is a crucial preprocessing step for earthquake prediction.

# Normalization and Scaling:

* Different sensors and measurement units may be used in earthquake datasets. Normalizing and scaling features are essential to ensure that the model interprets all variables on a consistent scale, preventing certain features from

dominating the learning process.

# Handling Time Series Data:

* Earthquake data often involves time series information. Dealing with time-dependent patterns,

seasonality, and trends requires specialized preprocessing techniques such as time-series decomposition, lag features, or rolling statistics.

# Computational Intensity:

* Earthquake prediction models, especially those based on machine learning algorithms, can be computationally intensive. Preprocessing steps need to be optimized for

efficiency, and consideration should be given to parallel processing or distributed computing for large datasets.

# Domain-Specific Challenges:

* Understanding the geological and seismological context is crucial. Domain-specific knowledge is needed for meaningful feature selection, identifying relevant patterns, and interpreting the results. Collaborating with domain experts is often necessary.

# LOADING THE DATASET:

* Loading the dataset using machine learning is the process of bringing the data into the machine learning

environment so that it can be used to train and evaluate a model.

* The specific steps involved in loading the dataset will vary depending On the machine learning library or

framework that is being used.However, there are some general steps that are common to most Machine learning frameworks.

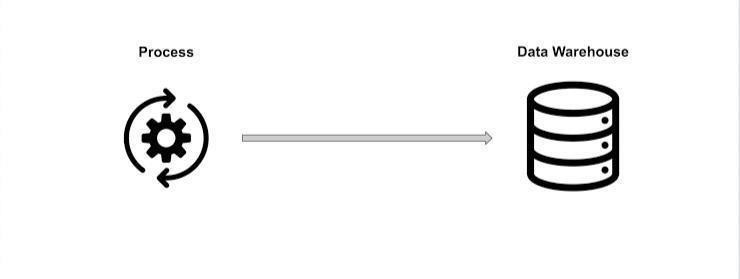
# Identify the dataset:

* The first step is to identify the dataset that you want to load.
* ThisDataset may be stored in a local file, in a database, or in a cloud storage Service.

# Load the dataset:

* Once you have identified the dataset, you need to load it into the Machine learning environment. This may involve using a built-in Function in the machine learning library, or it may involve writing your Own code.

# Preprocess the dataset:



Once the dataset is loaded into the machine learning environment, You may need to preprocess it before you can start training and Evaluating your model. This may involve cleaning the data, transforming the data into a suitable

format, and splitting the data into training and Test sets.

# Input data Pandas:

This library helps to load the data frame in a 2D array

format and has multiple functions to perform analysis tasks in one go.

# Matplotlib/Seaborn:

This library is used to draw visualizations.

Import pandas as pd

Import matplotlib.pyplot as plt Import seaborn as sb

Import warnings

Warnings.filterwarnings(‘ignore’)

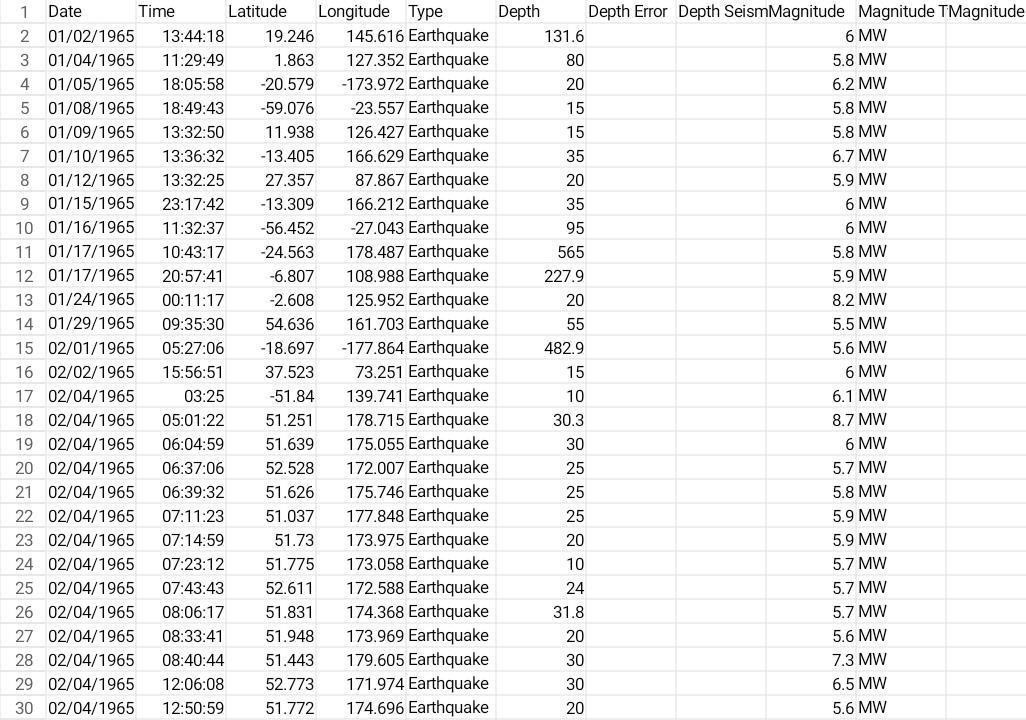
Df = pd.read\_csv(‘dataset.csv’) Df.head()

**Data exploration:**

# Data set Output:

The dataset we are using here contains data for the following columns:

* Origin time of the Earthquake
* Latitude and the longitude of the location.
* Depth – This means how much depth below the earth’s level the earthquake started.
* The magnitude of the earthquake
* Location



# PREPROCESSING DATASETS

Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.

This may involve removing errors and inconsistencies, handling Missing values, transforming the data into consistent format, and Scaling the data to a suitable range.

# Visualization:

From mpl\_toolkits.basemap import Basemap

M = Basemap(projection=’mill’,llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-

180,urcrnrlon=180,lat\_ts=20,resolution=’c’)

Longitudes = data[“Longitude”].tolist() Latitudes = data[“Latitude”].tolist()

#m = Basemap(width=12000000,height=9000000,projection=’lcc’, #resolution=None,lat\_1=80.,lat\_2=55,lat\_0=80,lon\_0=-

107.) X,y = m(longitudes,latitudes)

Fig = plt.figure(figsize=(12,10)) Plt.title(“All affected areas”)

m.plot(x, y, “o”, markersize = 2, color = ‘blue’) m.drawcoastlines()

m.fillcontinents(color=’coral’,lake\_color=’aqua’) m.drawmapboundary()

m.drawcountries() plt.show()

# Splitting The Dataset:

X = final\_data[[‘Timestamp’, ‘Latitude’, ‘Longitude’]]

Y = final\_data[[‘Magnitude’, ‘Depth’]]

From sklearn.cross\_validation 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:



We will be using the RandomForestRegressor model to predict the earthquake, here will look for its accuracy.

Reg = RandomForestRegressor(random\_state=42) Reg.fit(X\_train, y\_train)

Reg.predict(X\_test)

Reg.score(X\_test, y\_test)



# Neural Network Model:

A neural network model can be employed to forecast earthquakes by examining diverse elements and trends in

seismic data. This model harnesses the capabilities of neural networks, which draw inspiration from the neural

connections of the human brain, to analyze intricate data and reveal hidden relationships and patterns.

From keras.models import Sequential From keras.layers import Dense

Def create\_model(neurons, activation, optimizer, loss): Model = Sequential()

Model.add(Dense(neurons, activation=activation, input\_shape=(3,)))

Model.add(Dense(neurons, activation=activation)) Model.add(Dense(2, activation=’softmax’))

Model.compile(optimizer=optimizer, loss=loss, metrics=[‘accuracy’])

Return model

From keras.wrappers.scikit\_learn import KerasClassifier Model = KerasClassifier(build\_fn=create\_model, verbose=0)

# neurons = [16, 64, 128, 256]

Neurons = [16]

# batch\_size = [10, 20, 50, 100]

Batch\_size = [10]

Epochs = [10]

# activation = [‘relu’, ‘tanh’, ‘sigmoid’, ‘hard\_sigmoid’, ‘linear’, ‘exponential’]

Activation = [‘sigmoid’, ‘relu’]

# optimizer = [‘SGD’, ‘RMSprop’, ‘Adagrad’, ‘Adadelta’, ‘Adam’, ‘Adamax’, ‘Nadam’]

Optimizer = [‘SGD’, ‘Adadelta’] Loss = [‘squared\_hinge’]

Param\_grid = dict(neurons=neurons, batch\_size=batch\_size, epochs=epochs, activation=activation, optimizer=optimizer, loss=loss)

Grid = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1)

Grid\_result = grid.fit(X\_train, y\_train)

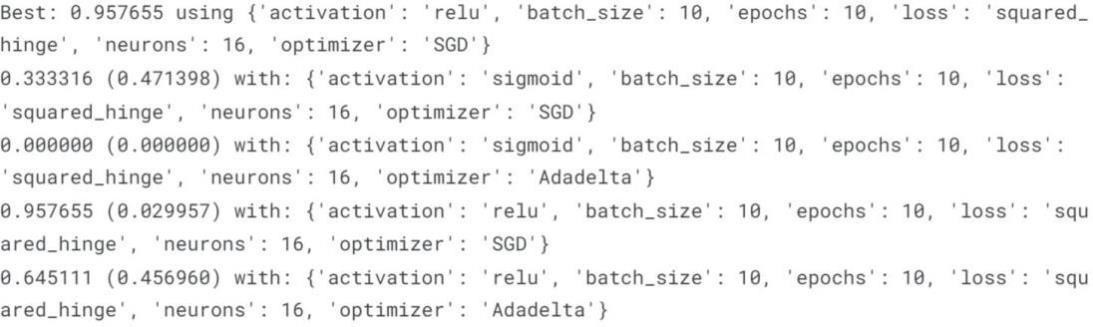
Print(“Best: %f using %s” %

(grid\_result.best\_score\_, grid\_result.best\_params\_)) Means = grid\_result.cv\_results\_[‘mean\_test\_score’]

Stds = grid\_result.cv\_results\_[‘std\_test\_score’] Params = grid\_result.cv\_results\_[‘params’]

For mean, stdev, param in zip(means, stds, params): Print(“%f (%f) with: %r” % (mean, stdev, param))

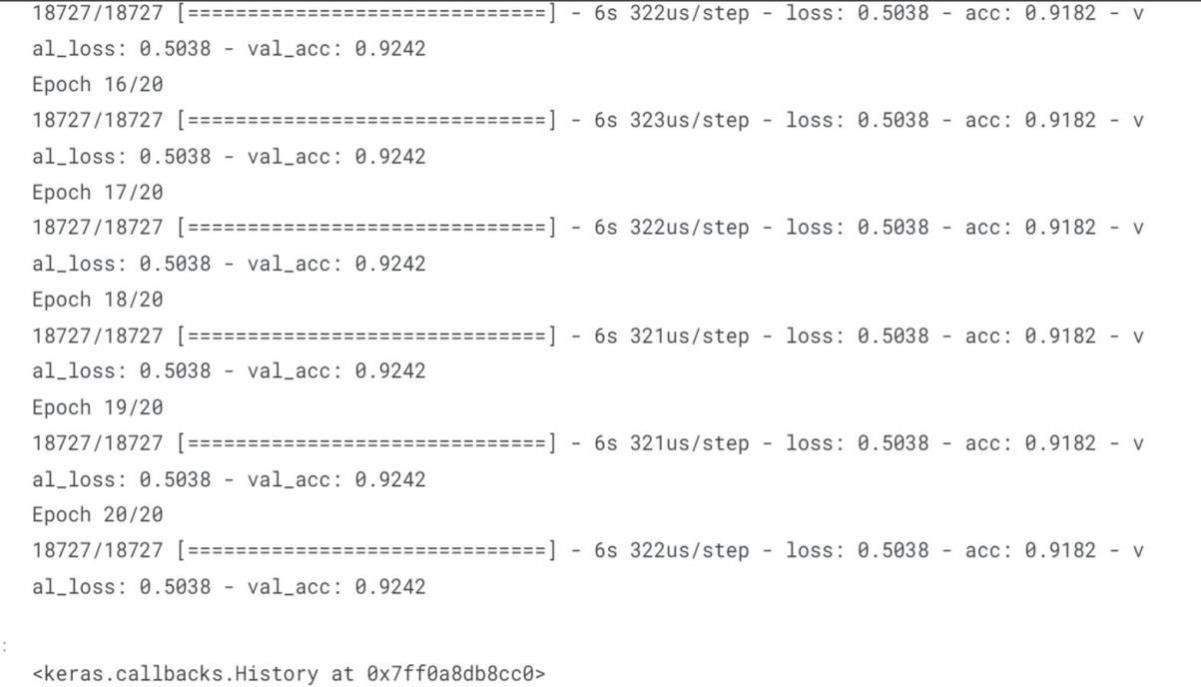
# Output:



Model = Sequential()

Model.add(Dense(16, activation=’relu’, input\_shape=(3,))) Model.add(Dense(16, activation=’relu’))

Model.add(Dense(2, activation=’softmax’))

Model.compile(optimizer=’SGD’, loss=’squared\_hinge’, metrics=[‘accuracy’])

Model. Fit(X\_train, y\_train, batch\_size=10, epochs=20, verbose=1, validation\_data=(X\_test, y\_test))

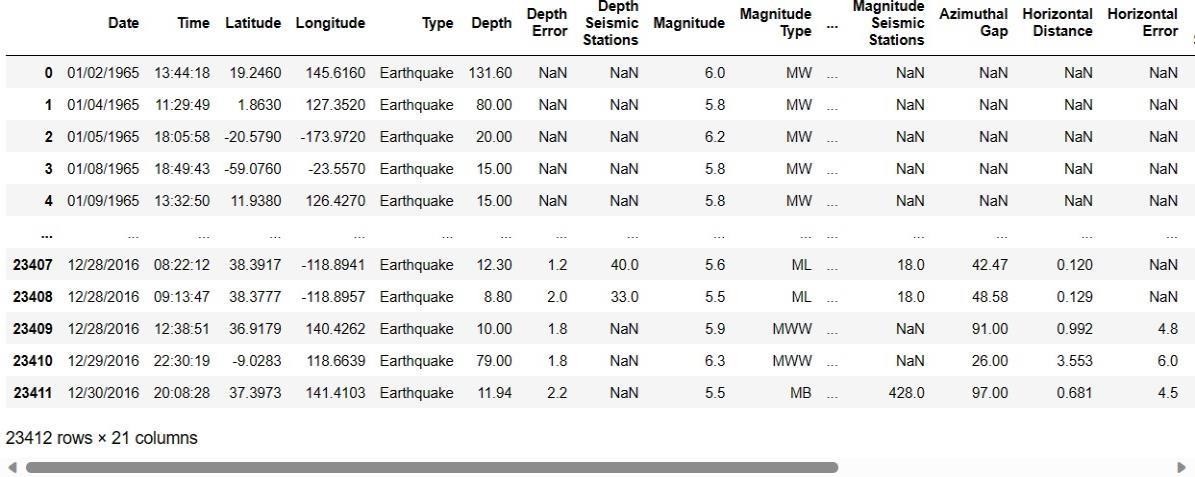
# Output:



Isn’t it amazing that we got an accuracy of 92%.

We can say the neural network is one of the best models to predict earthquakes that can be used in future.

# GIVEN DATASET :



**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.

# Prepare the data:

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

# Perform feature selection :

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

# Train the model :

There are many different machine learning algorithms that can be used for house price prediction. Some popular choices include linear regression, random forests, SVR.

# 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.

# 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.

# Features Selection :

***Checking for missing values***

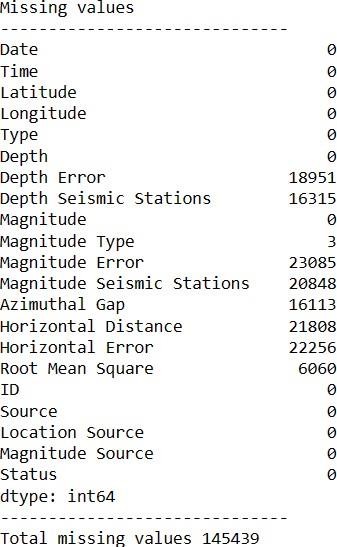
# In[1]:

print("Missing values") print("-" \*30) print(df.isna().sum()) print("- "\*30)

print("Total missing values",

df.isna().sum())

**Out[1]:**



**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, ridge 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) *# calculate R- squared score*

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)

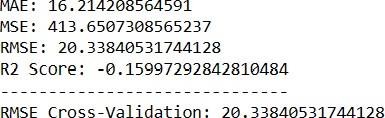
print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared) print("-"

\*30)

print("RMSE Cross-Validation:",rmse\_cross\_val)

**Out[4]:**



**Ridge Regression :**

**In[5]:**

ridge = Ridge() ridge.fit(X\_train, y\_train) predictions =

ridge.predict(X\_test)

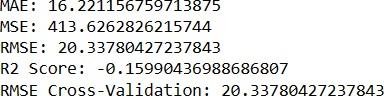
mae,mse, rmse, r\_squared,rmse\_cross\_val = evaluation(y\_test, predictions) print("MAE:",mae)

print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared)

print("RMSE Cross-Validation:",rmse\_cross\_val)

**Out[5]:**



**Lasso Regression:**

**In[6]:**

lasso = Lasso() lasso.fit(X\_train, y\_train)

predictions = lasso.predict(X\_test)

mae,mse, rmse, r\_squared,rmse\_cross\_val = evaluation(y\_test, predictions) print("MAE:",mae)

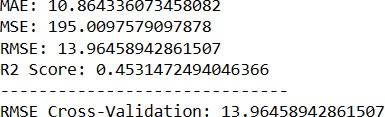
print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared) print("-"

\*30)

print("RMSE Cross-Validation:",rmse\_cross\_val)

**Out[6]:**



**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)

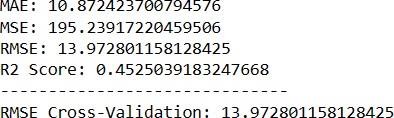
print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared) print("-"

\*30)

print("RMSE Cross-Validation:",rmse\_cross\_val)

**Out[7]:**



**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)

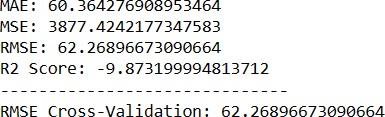
print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared) print("-"

\*30)

print("RMSE Cross-Validation:",rmse\_cross\_val)

**Out[8]:**



**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)

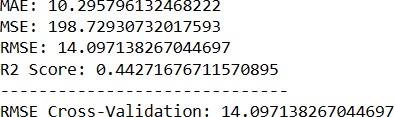
print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared) print("-"

\*30)

print("RMSE Cross-Validation:",rmse\_cross\_val)

**Out[9]:**



**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)

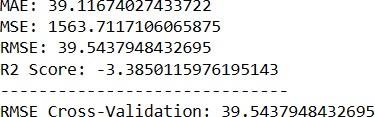
print("MSE:",mse)

print("RMSE:",rmse) print("R2 Score:",r\_squared)

print("-" \*30)

print("RMSE Cross-Validation:",rmse\_cross\_val)

# Out[10]:



**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']

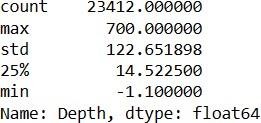
# In[12]:

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]:



**In[14]:**

y\_train.shape

# Out[14]:



**In[15]:**

y\_test.head()

# Out[15]:



**In[16]:**

Y\_test.shape

# Out[16]:



**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):

This metric measures the average squared difference between the predicted and actual earthquake.

**Root Mean Squared Error(RMSE):** This metric is the square root of the MSE. **Mean Absolute Error:**

This metric measures the average absolute difference between the predicted and actual earthquake.

# R-Squared:

This metric measures how well the model explains the variation in the actual earthquake happened.

# 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)),predi ctions) plt.xlabel("Latitude & Longitude") plt.ylabel("Depth") plt.title("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")

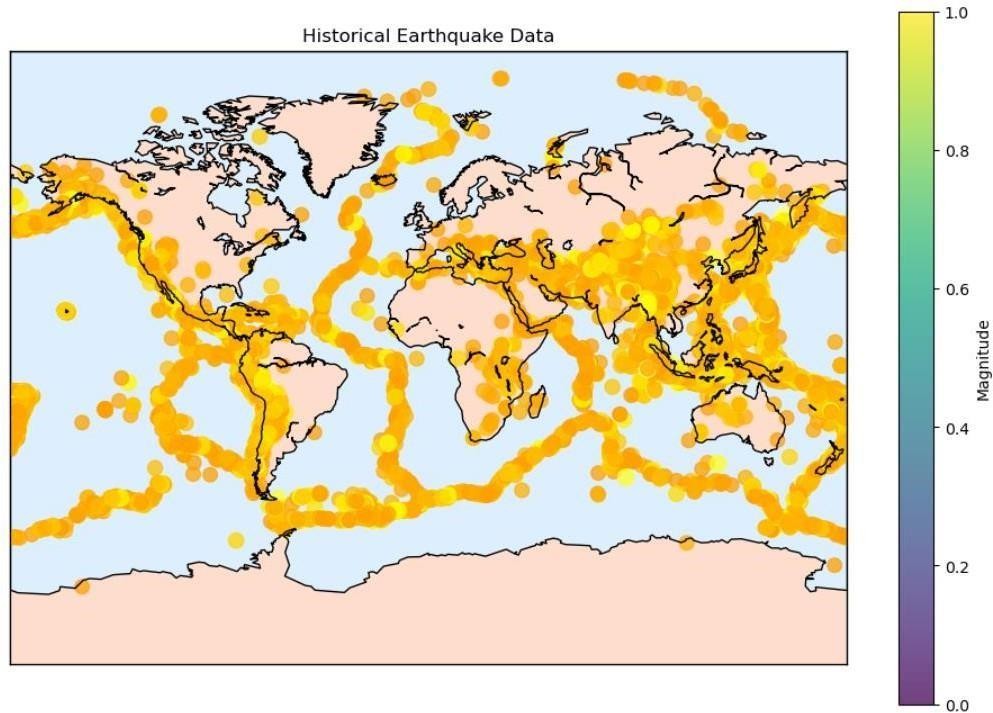
m.drawmapboundary(fill\_color="#DDEEFF") x,y = m(lons, lats)

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]:



**In[19]:**

print(r2\_score(y\_test, predictions)) print(mean\_absolute\_error(y\_test,predictions)) print(mean\_squared\_error(y\_test,predictions))

# Out[19]:



**Features Engineering :**

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.

# 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.

1. **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.

# 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.

1. **Gutenberg-Richter Law:**

This is a statistical law that relates the frequency and magnitude of earthquakes in a given region. It can be used to estimate the probability of occurrence and the expected magnitude of future earthquakes based on historical seismic events.

1. **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.

**Various features of perform model training :**

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.

1. **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. Earthquake catalog can also be used to estimate the probability and expected magnitude of future earthquakes based on statistical models, such as the Gutenberg-Richter law or the Poisson distribution.

1. **Geological features:**

These are the characteristics of the earth’s crust and mantle that affect the generation and propagation of seismic waves. Geological features include parameters such as rock type, density, porosity, permeability, elasticity, viscosity, and stress state. Geological features can be derived from various sources, such as borehole logs, geophysical surveys, or satellite

imagery. Geological features can be used to model the structure and dynamics of the earth’s interior and to simulate the ground motion at specific locations or regions.

1. **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. Environmental factors can be used to identify potential precursors or anomalies that may indicate seismic activity or to filter out noise or interference in seismic data.

**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 of the earth’s processes.
  + However, earthquake prediction is also subject to many uncertainties, limitations, and ethical issues that need to be addressed.
  + Earthquake data is often noisy, incomplete, inconsistent, or unreliable.
  + Earthquake catalog may be biased, incomplete, or inaccurate due to different reporting standards, detection thresholds, or measurement methods.