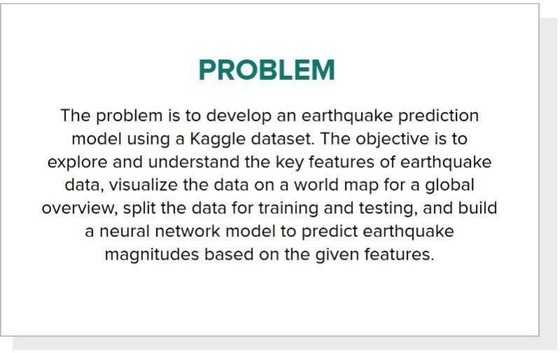
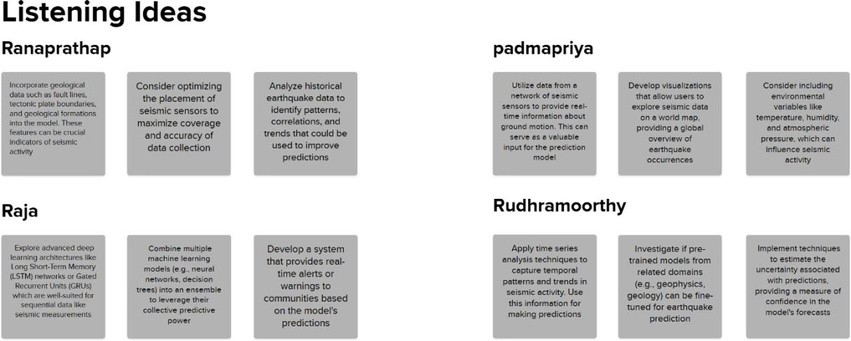
Earthquake prediction model using Python

Phase 1- Earthquake prediction model using Python

|  |  |
| --- | --- |
| Date | 30 September 2023 |
| Team ID  Name | 345  Padmapriya.E |
| Project Name | 4123-Earthquake Prediction Model using Python |





Prioritize ideas



Ideation Phase Problem Statement

Date 30 September 2023

Team ID 390

Project Name 4123- Earthquake Prediction Model using Python

**Problems Statements**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Problem Statement**  **(PS)** | **I am**  **(customer)** | **I’m trying** | **to** | **But** | **Because** | **Which makes me feel** |
| PS-1 | Prediction 1 | Provide some  aadnvdance warning based  magnitude  on monitoring ddaotma ain | | Exact time | Complexity of data history | Worry |
| PS2 | Individual 1 | uncertain  Execute and Ma prepare sufficie suddenly to pre | | y not have nt time pare | It accesses any instant of time even though the prediction in change | anxious |
| PS-3 | Prediction 2 | Avoid false alarms doing prediction | | Sometimes it false alarms | Due to the  complexity of  prediction earthquakes | Worry |
|  |  |  | |  | accuracy |  |
| PS-4 | Individual 2 | Looking for the prediction | | May become dispirited or | Execution  process | hopeless |
|  |  | announce | | complacent if |  |  |
|  |  |  | | false alarm |  |  |
|  |  |  | | occurs |  |  |
|  |  |  | | frequently |  |  |

**ProblemStatement 1**



**ProblemStatement 2**



**ProblemStatement 3**



**ProblemStatement 4**



Ideation Phase Empathize & Discover

Date 30 September 2023

Team ID 390

Project Name 4123-Earthquake Prediction Model using Python



## Project Design Phase-II

EARTHQUAKE PREDICTION MODEL

## PHASE 2- INNOVATION

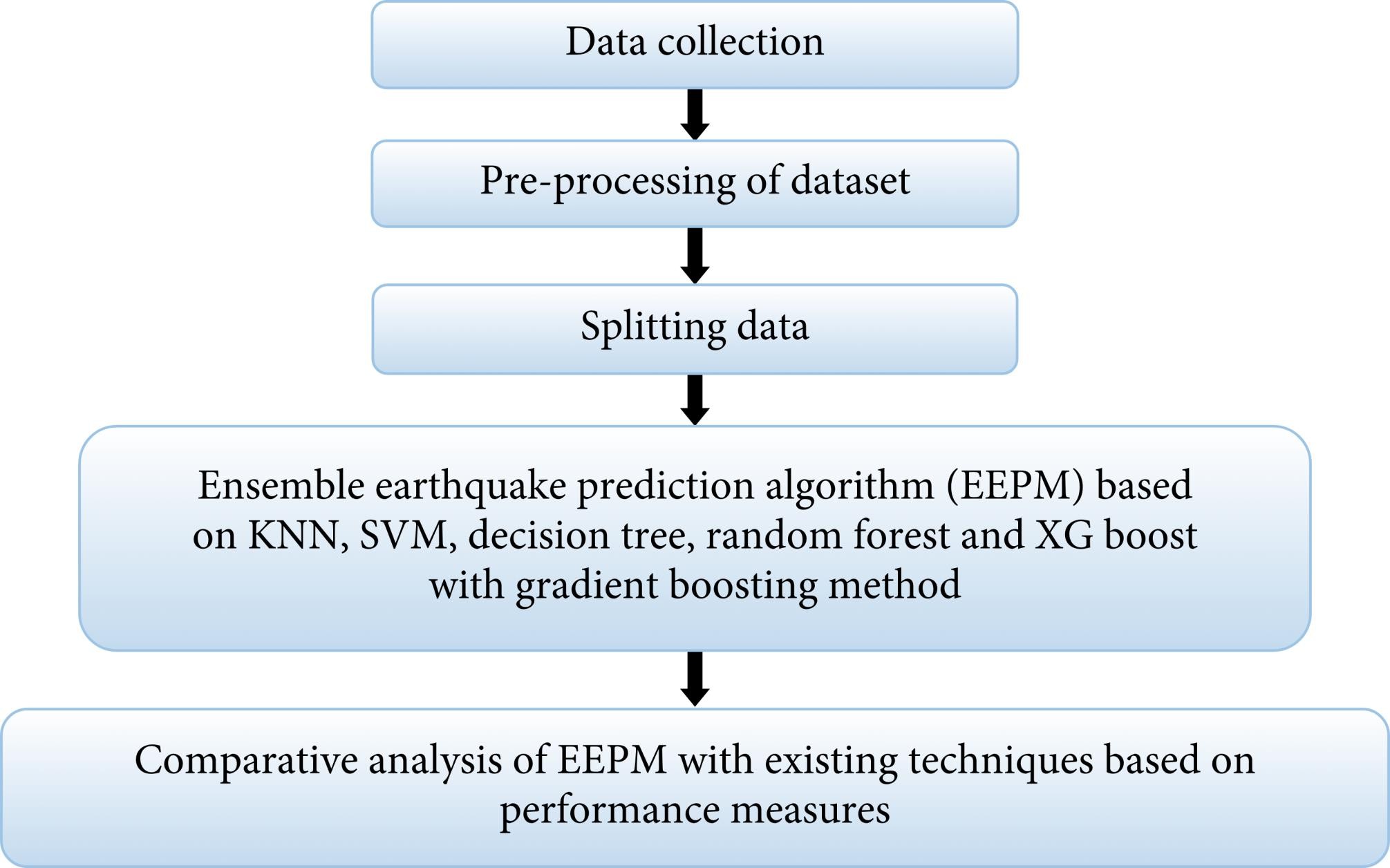
In this phase, we can explore innovative techniques such as ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.

And also excuted advanced techniques such as hyperparameter tuning to improve the prediction model's performance.

## ENSEMBLE LEARNING (BAGGING)

Ensemble learning in the context of earthquake prediction using bagging involves the use of multiple machine learning models, typically of the same type, to collectively make predictions about seismic events.

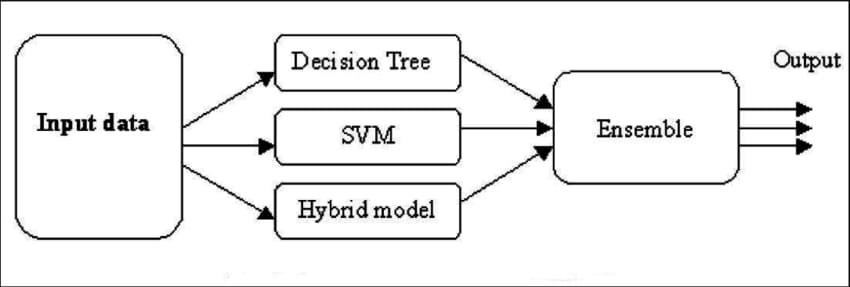
In bagging (Bootstrap Aggregating), multiple base models are trained on different subsets of the training data. Each subset is generated by sampling with replacement (bootstrapping) from the original dataset. This results in multiple diverse models, each having slightly different perspectives on the data.



In the case of earthquake prediction, ensemble learning with bagging might involve training several models (e.g., decision trees, support vector machines, etc.) on different subsets of seismic data. These models would then independently predict seismic activity. The final prediction is typically determined through some form of aggregation, such as averaging the outputs for regression tasks or using voting for classification tasks.

The advantage of using bagging in earthquake prediction lies in its ability to reduce overfitting and increase the overall stability and accuracy of predictions. By combining the outputs of multiple models trained on slightly different data, the ensemble can capture a broader range of patterns and relationships in the seismic data, potentially leading to more reliable earthquake predictions.

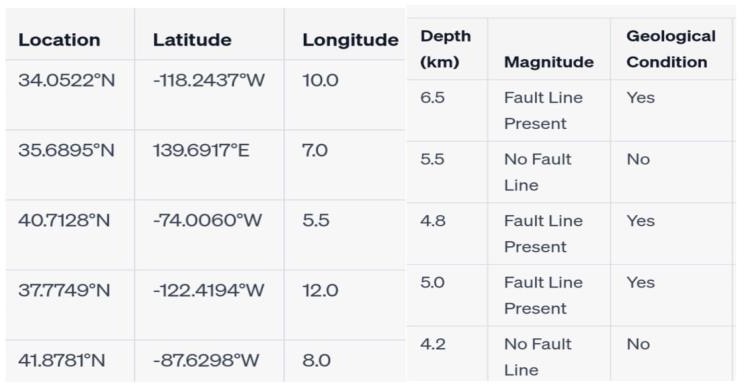
## Common Architecture of Ensemble learning:



**Synthetic Dataset**: Earthquake Prediction

## Dataset Description:

This synthetic dataset contains earthquake-related attributes for the purpose of prediction. It includes geographical coordinates, depth in kilometers, magnitude, and geological information.



## Project Details:

**Data Preprocessing:**

* Handle missing values.
* Detect and manage outliers.
* Standardize numerical features.
* Encode categorical variables (e.g., "Geological Condition").

## Feature Selection/Engineering:

* Analyze dataset to identify relevant features.
* Potentially engineer new features (e.g., spatial relationships).

## \*Model Selection:

* Utilize the Bagging ensemble method for its effectiveness in improving prediction accuracy.

## Base Model Choice:

* Employ decision trees as base models due to their capacity to handle non-linear relationships and interpretability.

example of using a Bagging ensemble with Decision Trees for earthquake prediction. We'll use the scikit-learn library in Python:

python

**# Import necessary libraries**

**from sklearn.ensemble import BaggingClassifier from sklearn.tree import DecisionTreeClassifier from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score**

**# Assume you have a dataset 'X' containing features and 'y' containing labels**

**# 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)**

**# Initialize a Decision Tree Classifier base\_classifier = DecisionTreeClassifier()**

**# Initialize a Bagging Classifier with Decision Tree as base estimator**

**bagging\_classifier = BaggingClassifier(base\_estimator=base\_classifier, n\_estimators=10, random\_state=42)**

**# Train the Bagging Classifier bagging\_classifier.fit(X\_train, y\_train)**

**# Predict using the trained model**

**y\_pred = bagging\_classifier.predict(X\_test)**

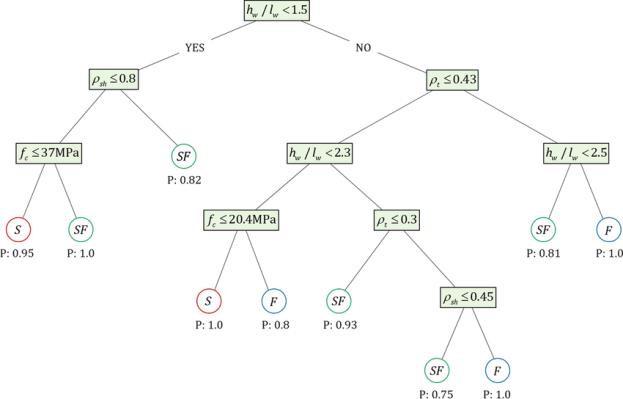
**# Calculate accuracy**

**accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy: {accuracy}")**

**In this example:**

1. We first import the necessary libraries including the BaggingClassifier, DecisionTreeClassifier, and other relevant tools.
2. We assume you have a dataset `X` with features and `y` with corresponding labels.
3. We split the data into training and testing sets.
4. We initialize a Decision Tree Classifier as the base estimator.
5. Then, we initialize a Bagging Classifier with the Decision Tree as the base estimator and specify the number of estimators (trees) in the ensemble (in this case, 10).
6. We train the Bagging Classifier on the training data.
7. Next, we use the trained model to make predictions on the test data.
8. Finally, we calculate the accuracy of the model.

**Sample decision tree(**earth quake prediction**)**:



## Model Training:

* + Divide the dataset into training and testing sets.
  + Train each base model on bootstrapped subsets of the data.

## Bagging Ensemble Creation:

* + Combine base models' predictions through techniques like averaging or voting.

## Model Evaluation:

* + Assess performance using metrics such as Mean Absolute Error (MAE) and Mean Squared Error (MSE).
  + Consider specialized metrics like precision and recall for a comprehensive evaluation.

## Hyperparameter Tuning:

* + Fine-tune hyperparameters to optimize both base models and the ensemble.

Improving prediction using hyperparameter tuning in ensemble learning, specifically bagging, involves optimizing the parameters that control the behavior of the individual base learners and the ensemble as a whole.

**Select a Base Learner:**

* + - Choose a suitable base learner (e.g., decision trees, random forests, etc.) for bagging.

**Define Hyperparameters:**

* + - Identify the hyperparameters of the chosen base learner that can be tuned. For example, in a decision tree, you might want to tune parameters like max depth, minimum samples per leaf, etc.

**Set up a Validation Set:**

* + - Split your dataset into training, validation, and test sets. The validation set is used to evaluate the performance of different hyperparameter combinations.

**Grid Search or Random Search:**

* + - Perform a hyperparameter search using techniques like grid search or random search. Grid search exhaustively tries all combinations of a predefined set of hyperparameters, while random search randomly samples combinations.

**Evaluate Performance:**

* + - For each set of hyperparameters, train the base learner on the training set and evaluate its performance on the validation set using a suitable metric (e.g., accuracy, F1-score, etc.).

**Select the Best Hyperparameters:**

* + - Identify the combination of hyperparameters that gives the best performance on the validation set.

**Train the Ensemble:**

* + - Once you have the optimal hyperparameters for the base learner, train multiple instances of the base learner with different subsets of the training data (bagging). Each base learner should be trained with a different random subset.

**Aggregate Predictions:**

* + - Combine the predictions of individual base learners. For classification tasks, this could be done through voting or averaging.

**Evaluate on Test Set:**

* + - Finally, evaluate the performance of the ensemble on the test set to get an unbiased estimate of its predictive power.

**Monitor for Overfitting:**

* + - Keep an eye out for overfitting. If the ensemble performs significantly worse on the test set compared to the validation set, you might need to revisit your hyperparameter tuning process.

Remember to iterate and refine this process as needed. It's also worth considering techniques like cross-validation and bootstrapping to further validate the performance of your ensemble.

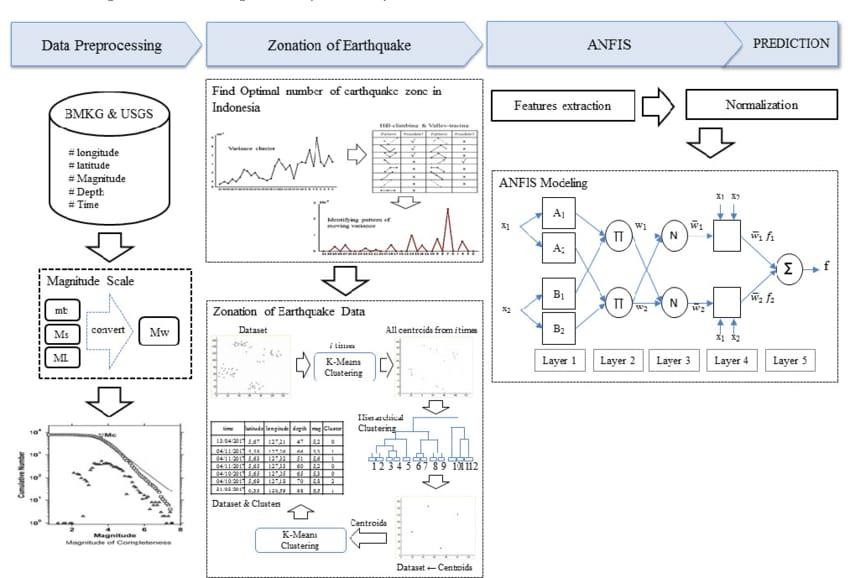
## Testing and Validation:

* + Validate model performance on a separate testing set to ensure generalizability.

## Visualization:

* + Create visualizations to display earthquake predictions.
  + Compare predictions with actual occurrences.
  + Show feature importance through visual aids.

## Control Flow for earthquake prediction:



**Conclusion:**

In conclusion, this project endeavors to construct a potent earthquake prediction model using the Bagging ensemble method. Through meticulous data preprocessing, feature engineering, and astute model selection, we aim to forge a dependable tool for earthquake prediction. The evaluation metrics will furnish valuable insights into the model's efficacy, while visualizations will serve as a vital aid in comprehending the results.This comprehensive project is designed to showcase the efficacy of the Bagging ensemble technique in earthquake prediction, with a focus on data preprocessing, model selection, and rigorous evaluation.

# ai-phase3

October 16, 2023

[2]:

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

**import matplotlib.pyplot as plt**

**import os**

print(os.listdir(r"C:\Users\91912\Desktop\AI\_Phase3"))

['database.csv']

[ ]:

*# Load your data*

[4]:

data = pd.read\_csv(r"C:\Users\91912\Desktop\AI\_Phase3/database.csv") data.head()

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

Horizontal Error Root Mean Square ID Source Location Source \

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | NaN | NaN ISCGEM860706 ISCGEM | ISCGEM |
| 1 | NaN | NaN ISCGEM860737 ISCGEM | ISCGEM |

1

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | NaN | NaN ISCGEM860762 ISCGEM | ISCGEM |
| 3 | NaN | NaN ISCGEM860856 ISCGEM | ISCGEM |
| 4 | NaN | NaN ISCGEM860890 ISCGEM | ISCGEM |

Magnitude Source Status

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

[5 rows x 21 columns]

[5]:

data.columns

1. : Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error', 'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',

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

[ ]:

*# Select relevant columns*

1. :

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [6]: | Date Time | Latitude | Longitude | Depth | Magnitude |
|  | 0 01-02-1965 13:44:18 | 19.246 | 145.616 | 131.6 | 6.0 |
|  | 1 01-04-1965 11:29:49 | 1.863 | 127.352 | 80.0 | 5.8 |
|  | 2 01-05-1965 18:05:58 | -20.579 | -173.972 | 20.0 | 6.2 |
|  | 3 01-08-1965 18:49:43 | -59.076 | -23.557 | 15.0 | 5.8 |
|  | 4 01-09-1965 13:32:50 | 11.938 | 126.427 | 15.0 | 5.8 |

[ ]:

*# Convert date and time to timestamp*

[11]:

**import datetime**

timestamp = []

**for** d, t **in** zip(data['Date'], data['Time']):

**try**:

dt = datetime.datetime.strptime(d+' '+t, '%m/**%d**/%Y %H:%M:%S') timestamp.append(dt)

**except ValueError**: timestamp.append('ValueError')

2

timeStamp = pd.Series(timestamp) data['Timestamp'] = timeStamp.values

final\_data = data.drop(['Date', 'Time'], axis=1)

final\_data = final\_data[final\_data.Timestamp != 'ValueError'] final\_data.head()

|  |  |  |
| --- | --- | --- |
| [11]: | Latitude Longitude Depth | Magnitude Timestamp |
|  | 7 -13.309 166.212 35.0 | 6.0 1965-01-15 23:17:42 |
|  | 8 -56.452 -27.043 95.0 | 6.0 1965-01-16 11:32:37 |
|  | 9 -24.563 178.487 565.0 | 5.8 1965-01-17 10:43:17 |
|  | 10 -6.807 108.988 227.9 | 5.9 1965-01-17 20:57:41 |
|  | 11 -2.608 125.952 20.0 | 8.2 1965-01-24 00:11:17 |

[ ]:

[13]:

*# Intialize Basemap*

!pip install basemap

Collecting basemap

Obtaining dependency information for basemap from https://files.pythonhosted.o rg/packages/c9/fb/0aa18cea5d108ebd35e51348ff4269b6cef10660c537e29a7ae596c5fb0b/b asemap-1.3.8-cp311-cp311-win\_amd64.whl.metadata

Downloading basemap-1.3.8-cp311-cp311-win\_amd64.whl.metadata (7.7 kB) Collecting basemap-data<1.4,>=1.3.2 (from basemap)

Downloading basemap\_data-1.3.2-py2.py3-none-any.whl (30.5 MB)

0.0/30.5 MB ? eta -: :

* 1. /30.5 MB 8.9 MB/s eta 0:00:04
  2. /30.5 MB 2.1 MB/s eta 0:00:15
  3. /30.5 MB 1.6 MB/s eta 0:00:20

0.3/30.5 MB 1.5 MB/s eta 0:00:21

0.4/30.5 MB 1.5 MB/s eta 0:00:20

* 1. /30.5 MB 1.4 MB/s eta 0:00:22
  2. /30.5 MB 1.5 MB/s eta 0:00:21
  3. /30.5 MB 1.4 MB/s eta 0:00:21

0.7/30.5 MB 1.4 MB/s eta 0:00:21

- 0.8/30.5 MB 1.3 MB/s eta 0:00:23

- 0.8/30.5 MB 1.3 MB/s eta 0:00:24

- 0.9/30.5 MB 1.3 MB/s eta 0:00:24

- 0.9/30.5 MB 1.2 MB/s eta 0:00:25

- 1.0/30.5 MB 1.2 MB/s eta 0:00:25

- 1.0/30.5 MB 1.2 MB/s eta 0:00:26

- 1.0/30.5 MB 1.2 MB/s eta 0:00:26

- 1.1/30.5 MB 1.2 MB/s eta 0:00:25

- 1.2/30.5 MB 1.2 MB/s eta 0:00:25

- 1.2/30.5 MB 1.1 MB/s eta 0:00:27

- 1.2/30.5 MB 1.1 MB/s eta 0:00:27

- 1.2/30.5 MB 1.1 MB/s eta 0:00:27

- 1.2/30.5 MB 1.1 MB/s eta 0:00:28

3

Downloading pyshp-2.3.1-py2.py3-none-any.whl (46 kB)

0.0/46.5 kB ? eta -: :

41.0/46.5 kB 2.0 MB/s eta 0:00:01

41.0/46.5 kB 2.0 MB/s eta 0:00:01

46.5/46.5 kB 333.0 kB/s eta 0:00:00

Requirement already satisfied: matplotlib<3.8,>=1.5 in c:\users\91912\anaconda3\lib\site-packages (from basemap) (3.7.2) Collecting pyproj<3.7.0,>=1.9.3 (from basemap)

Obtaining dependency information for pyproj<3.7.0,>=1.9.3 from https://files.p ythonhosted.org/packages/79/95/eb68113c5b5737c342bde1bab92705dabe69c16299c5a1226 16e50f1fbd6/pyproj-3.6.1-cp311-cp311-win\_amd64.whl.metadata

Downloading pyproj-3.6.1-cp311-cp311-win\_amd64.whl.metadata (31 kB) Requirement already satisfied: numpy<1.26,>=1.21 in c:\users\91912\anaconda3\lib\site-packages (from basemap) (1.24.3)

Requirement already satisfied: contourpy>=1.0.1 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (1.0.5)

Requirement already satisfied: cycler>=0.10 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (0.11.0)

Requirement already satisfied: fonttools>=4.22.0 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (4.25.0)

Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (1.4.4)

Requirement already satisfied: packaging>=20.0 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (23.1)

Requirement already satisfied: pillow>=6.2.0 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (9.4.0)

Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (3.0.9)

Requirement already satisfied: python-dateutil>=2.7 in c:\users\91912\anaconda3\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (2.8.2)

Requirement already satisfied: certifi in c:\users\91912\anaconda3\lib\site- packages (from pyproj<3.7.0,>=1.9.3->basemap) (2023.7.22)

Requirement already satisfied: six>=1.5 in c:\users\91912\anaconda3\lib\site- packages (from python-dateutil>=2.7->matplotlib<3.8,>=1.5->basemap) (1.16.0) Downloading basemap-1.3.8-cp311-cp311-win\_amd64.whl (486 kB)

0.0/486.8 kB ? eta -: :

112.6/486.8 kB 2.2 MB/s eta 0:00:01

194.6/486.8 kB 2.4 MB/s eta 0:00:01

286.7/486.8 kB 2.5 MB/s eta 0:00:01

430.1/486.8 kB 2.2 MB/s eta 0:00:01

24

[ ]:

[ ]:

[16]:

**from mpl\_toolkits.basemap import** Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80,␣

𝗌llcrnrlon=-180,urcrnrlon=180,lat\_ts=20,resolution='c')

longitudes = data["Longitude"].tolist() latitudes = data["Latitude"].tolist() 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()

5.0/6.1 MB 2.4 MB/s eta 0:00:01

5.2/6.1 MB 2.4 MB/s eta 0:00:01

* 1. /6.1 MB 2.4 MB/s eta 0:00:01
  2. /6.1 MB 2.4 MB/s eta 0:00:01
  3. /6.1 MB 2.4 MB/s eta 0:00:01
  4. /6.1 MB 2.4 MB/s eta 0:00:01
  5. /6.1 MB 2.4 MB/s eta 0:00:01

5.7/6.1 MB 2.4 MB/s eta 0:00:01

- 5.9/6.1 MB 2.4 MB/s eta 0:00:01 6.0/6.1 MB 2.4 MB/s eta 0:00:01

6.1/6.1 MB 2.4 MB/s eta 0:00:01

6.1/6.1 MB 2.4 MB/s eta 0:00:01

6.1/6.1 MB 2.4 MB/s eta 0:00:01

6.1/6.1 MB 2.2 MB/s eta 0:00:00

Installing collected packages: pyshp, pyproj, basemap-data, basemap

Successfully installed basemap-1.3.8 basemap-data-1.3.2 pyproj-3.6.1 pyshp-2.3.1

*# Get coordinates*

*# Create the map*

26

# ai-phase-4

October 24, 2023

[ ]:

Date:24 october 2023 Team ID:NM2023TMID345

Team Name:Proj\_227274\_team\_1

Project Name:Earthquake prediction model

[5]:

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

**import matplotlib.pyplot as plt**

**from mpl\_toolkits.basemap import** Basemap

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

[ ]:

*# Load your earthquake data into a Pandas DataFrame*

[6]:

data = pd.read\_csv(r'C:\Users\91912\Desktop\AI\_Phase3/database.csv')

1. :

data.head()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [7]: | Date Time Latitude | | Longitude | | Type | Depth Depth | Error | \ |
|  | 0 01-02-1965 13:44:18 19.246 | | 145.616 | | Earthquake | 131.6 | NaN |  |
|  | 1 01-04-1965 11:29:49 1.863 | | 127.352 | | Earthquake | 80.0 | NaN |  |
|  | 2 01-05-1965 18:05:58 -20.579 | | -173.972 | | Earthquake | 20.0 | NaN |  |
|  | 3 01-08-1965 18:49:43 -59.076 | | -23.557 | | Earthquake | 15.0 | NaN |  |
|  | 4 01-09-1965 13:32:50 11.938 | | 126.427 | | Earthquake | 15.0 | NaN |  |
| Depth Seismic Stations Magnitude Magnitude Type … \ | | | | | | | | |
| 0 | | NaN | 6.0 | | MW … | | | |
| 1 | | NaN | 5.8 | | MW … | | | |
| 2 | | NaN | 6.2 | | MW … | | | |
| 3 | | NaN | 5.8 | | MW … | | | |
| 4 | | NaN | 5.8 | | MW … | | | |
| Magnitude Seismic Stations Azimuthal Gap Horizontal Distance \ | | | | | | | | |
| 0 | | NaN |  | NaN | NaN | | | |
| 1 | | NaN |  | NaN | NaN | | | |
| 2 | | NaN |  | NaN | NaN | | | |
| 3 | | NaN |  | NaN | NaN | | | |
| 4 | | NaN |  | NaN | NaN | | | |
|  | |  | 1 |  |  | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Horizontal Error | Root Mean | Square | ID | Source | Location | Source \ |
| 0 NaN |  | NaN | ISCGEM860706 | ISCGEM |  | ISCGEM |
| 1 NaN |  | NaN | ISCGEM860737 | ISCGEM |  | ISCGEM |
| 2 NaN |  | NaN | ISCGEM860762 | ISCGEM |  | ISCGEM |
| 3 NaN |  | NaN | ISCGEM860856 | ISCGEM |  | ISCGEM |
| 4 NaN |  | NaN | ISCGEM860890 | ISCGEM |  | ISCGEM |

1. :

**import datetime import time**

timestamp = []

**for** d, t **in** zip(data['Date'], data['Time']):

**try**:

ts = datetime.datetime.strptime(d + ' ' + t, '%m/**%d**/%Y %H:%M:%S') min\_timestamp = datetime.datetime(1970, 1, 1)

max\_timestamp = datetime.datetime(2038, 1, 19)

**if** min\_timestamp <= ts <= max\_timestamp: timestamp.append(time.mktime(ts.timetuple()))

**else**:

timestamp.append('OutofRange')

**except ValueError**:

*# print('ValueError')*

timestamp.append('ValueError') data['Timestamp'] = timestamp

timeStamp = pd.Series(timestamp) data['Timestamp'] = timeStamp.values

final\_data = data.drop(['Date', 'Time'], axis=1)

final\_data = final\_data[final\_data.Timestamp != 'ValueError'] final\_data.head()

Magnitude Source Status

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

[5 rows x 21 columns]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [8]: |  | Latitude | Longitude | Type Depth Depth Error \ |
|  | 7 | -13.309 | 166.212 | Earthquake 35.0 NaN |

2

[ ]:

1. :

m = Basemap(projection='mill', llcrnrlat=-80, urcrnrlat=80, llcrnrlon=-180,␣

𝗌urcrnrlon=180, lat\_ts=20, resolution='c')

longitudes = data["Longitude"].tolist() latitudes = data["Latitude"].tolist() x, y = m(longitudes, latitudes)

\

Location Source Magnitude Source Status Timestamp

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | -56.452 | -27.043 Earthquake 95.0 | | | NaN | | |
| 9 | -24.563 | 178.487 Earthquake 565.0 | | | NaN | | |
| 10 | -6.807 | 108.988 Earthquake 227.9 | | | NaN | | |
| 11 | -2.608 | 125.952 Earthquake 20.0 | | | NaN | | |
| Depth Seismic Stations Magnitude Magnitude Type Magnitude Error | | | | | | | |
| 7 | NaN | | 6.0 | | MW | NaN | |
| 8 | NaN | | 6.0 | | MW | NaN | |
| 9 | NaN | | 5.8 | | MW | NaN | |
| 10 | NaN | | 5.9 | | MW | NaN | |
| 11 | NaN | | 8.2 | | MW | NaN | |
|  | Magnitude Seismic Stations Azimuthal | | | | Gap Horizontal | Distance | \ |
| 7 | NaN | | | | NaN | NaN |  |
| 8 | NaN | | | | NaN | NaN |  |
| 9 | NaN | | | | NaN | NaN |  |
| 10 | NaN | | | | NaN | NaN |  |
| 11 | NaN | | | | NaN | NaN |  |
| Horizontal Error Root Mean Square ID Source \ | | | | | | | |
| 7 | NaN | | NaN | ISCGEM861111 | | ISCGEM | |
| 8 | NaN | | NaN | ISCGEMSUP861125 | | ISCGEMSUP | |
| 9 | NaN | | NaN | ISCGEM861148 | | ISCGEM | |
| 10 | NaN | | NaN | ISCGEM861155 | | ISCGEM | |
| 11 | NaN | | NaN | ISCGEM861299 | | ISCGEM | |

|  |  |  |
| --- | --- | --- |
| 7 | ISCGEM | ISCGEM Automatic OutofRange |
| 8 | ISCGEM | ISCGEM Automatic OutofRange |
| 9 | ISCGEM | ISCGEM Automatic OutofRange |
| 10 | ISCGEM | ISCGEM Automatic OutofRange |
| 11 | ISCGEM | ISCGEM Automatic OutofRange |

*# Visualizing the data on a world map*

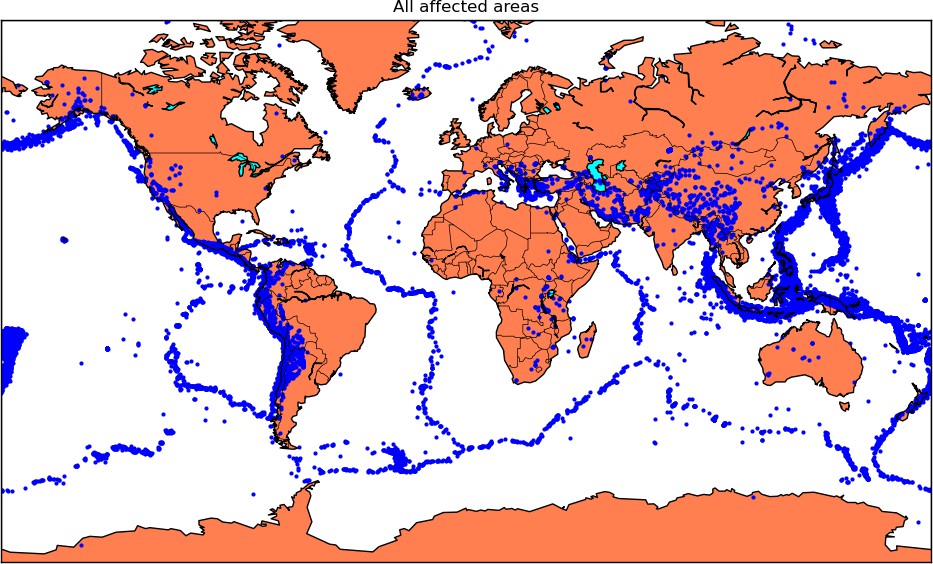
1. :

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')

3

m.drawmapboundary() m.drawcountries() plt.show()



[ ]:

*# Splitting it into training and testing sets*

[5]:

**import pandas as pd**

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

[ ]:

*# Load your earthquake data into a Pandas DataFrame*

[20]:

data = pd.read\_csv(r'C:\Users\91912\Desktop\AI\_Phase3/database.csv')

[ ]:

*# Define your feature columns and target columns*

[17]:

X = data[['Latitude', 'Longitude']] y = data[['Magnitude', 'Depth']]

[ ]:

*# Split the data into training and testing sets*

[18]:

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

𝗌random\_state=42)

[ ]:

*# Print the shapes of the training and testing sets to verify the split*

4

[19]:

print("X\_train shape:", X\_train.shape) print("X\_test shape:", X\_test.shape) print("y\_train shape:", y\_train.shape) print("y\_test shape:", y\_test.shape)

X\_train shape: (18729, 2)

X\_test shape: (4683, 2)

y\_train shape: (18729, 2)

y\_test shape: (4683, 2)

[ ]:

*# using a Decision Tree regressor for your earthquake prediction model.*

[21]:

**from sklearn.tree import** DecisionTreeRegressor

**from sklearn.metrics import** mean\_squared\_error, mean\_absolute\_error, r2\_score

[ ]:

*# Create a Decision Tree regressor*

[23]:

tree\_model = DecisionTreeRegressor(random\_state=42)

[ ]:

*# Fit the Decision Tree model to the training data*

[24]:

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

1. : DecisionTreeRegressor(random\_state=42) [ ]:

*# Make predictions on the testing data*

1. :

y\_pred\_tree = tree\_model.predict(X\_test)

[ ]:

*# Evaluate the Decision Tree model*

1. :

mse\_tree = mean\_squared\_error(y\_test, y\_pred\_tree) mae\_tree = mean\_absolute\_error(y\_test, y\_pred\_tree) r2\_tree = r2\_score(y\_test, y\_pred\_tree)

1. :

print("Decision Tree Model:") print("Mean Squared Error:", mse\_tree) print("Mean Absolute Error:", mae\_tree) print("R-squared Score:", r2\_tree)

Decision Tree Model:

Mean Squared Error: 1678.1163417214373 Mean Absolute Error: 12.115441063420867 R-squared Score: -0.03476557402029423

5