A PROJECT REPORT ON

EARTHQUAKE PREDICTION MODEL USING PYTHON

Subject in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the guidance of

Mr. BALAJI K

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY

Approved by AICTE, New Delhi Affiliated to

Anna University, Chennai

GRT Mahalakshmi nagar, Chennai – Tirupathi Highway ,Tiruttani-631 209

PROJECT REPORT SUBMITTED BY,

NAME : SARANYA R

NM ID : au110321106047

MAIL ID : rajeshsaranya37@gmail.com

Year/sem/dept: III/V/ECE

DECLARATION

I SUDHANDIRADEVI MD hereby declare that the project report entitled creating a earthquake python is done by me under the guidance of Mr BALAJI K is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

Date of submission : 01/11/2023

Place: GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, TIRUTTANI-631209

R. Saranya.

SIGNATURE OF THE CANDIDATE

TABLE OF CONTENT:

SI.NO.	TITLE	PAGE NO
1	AIM AND ABTRACT	5
2	INTRODUCTION	7
3	PROBLEMS IN EARTHQUAKE PREDICTION	7
4	**WAYS TO FIX THOSE PROBLEMS**	9
5	**DESIGN**	11
6	**DESIGN FOR INNOVATION**	13
7	**INNOVATION IN MY PROJECT**	16
8	**BLOCKS TO ADD IN DESIGN**	19
9	**CHANGES IN DESIGN**	21

10	**DESIGN BLOCK DIAGRAM**	23
11	**LOADING AND PREPROCESSING**	24
12	**PROGRAM AND OUTPUT**	25
13	**EXECUTION STEPS**	48
14	**HOW TO OVERCOME THE CHALLENGES OF LOADING & PREPROCESSING A EARTHQUAKE PREDICTION**	49
15	**SOME COMMON DATA PREPROCESSSING**	59
16	**DATASET**	52
17	**PROGRAM AND OUTPUT**	54
18	**EXECUTION STEPS**	64

19	**CONCLUSION**	66

AIM:

1. Collecting data on past earthquakes, geological features, and other relevant factors such as seismic activity, temperature, humidity, and atmospheric pressure.

2. Preprocessing the collected data to remove noise or inconsistencies.

3. Developing machine learning models such as decision trees, random forests, and neural networks to predict earthquake likelihood and timing.

4. Training and testing the models on split data sets to evaluate accuracy and performance.

5. Optimizing the models by adjusting hyper parameters to improve accuracy and performance.

6. Validating the results by comparing predicted earthquake locations and magnitudes against actual earthquake data using statistical measures.

7. Collaborating with other researchers in the field to share data, models, and results for standardization and validation.

8. Participating in peer review processes to ensure accuracy and reliability of predictions.

9. Continuously updating and improving the system with new data and techniques.

10. Implementing the system in real-world scenarios to help prepare for and mitigate the impact of earthquake.

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 remark from seismometers with Richter magnitude scale. Ground rupture Landslides, Soil liquefaction and Tsunami are the main effects created by earthquakes. Today's earthquake warning systems used to provide regional notification of an earthquake in progress. Many methods have been already developed for predicting the time and place in which earthquakes will occur, but it did not predicted using big data analytics. This report knows that the Standard Deviation to identify next earthquake happening from tons of international geological survey data using data analysis in pandas & matplotlib framework. It's the top-level component of all the ones that you will consider in the following point current location shakes per minute. Other than above mentioned features separate pandas and matplot function implemented to analyze sheer number of earthquakes per day. Final result shows which location suffered from maximum number of shakes and priority of earthquake occurrence location and Time.

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 videos content is uploaded in YouTube update their wall in every minute. Search engines logging 600 million queries daily. There are different data centers where people can store vast amount of data, such as IBM Server, EMC Server etc. On the other hand, AWS (Amazon Web Services) provide a host of services to store, process and analyze the data at scale in a cost-effective manner. Big data term refers collection of large datasets that are distributed multidimensional and complex that it becomes difficult to processing on hand traditional data processing applications.

.

PROBLEMS IN EARTHQUAKE PREDECTION:

Especially when analyzing large datasets. If the computer or server As an AI language model, I do not have personal experience with Python or earthquake prediction. However, there are some common issues that can arise when using Python for earthquake prediction:

1. Lack of data: The accuracy of earthquake prediction depends on the amount and quality of data available. If there is not enough data or if the data is incomplete, inaccurate, or biased, the predictions may not be reliable.

2. Complexity of models: Earthquake prediction requires complex mathematical models and algorithms that may be difficult to implement in Python. This can lead to errors in the code or incorrect results.

3. Limited computational power: Earthquake prediction requires significant computational power running the Python code does not have enough processing power, the predictions may take a long time or be inaccurate.

4. Lack of standardization: There is no standard approach to earthquake prediction, and different researchers may use different methods and

models. This can make it difficult to compare and validate results across different studies.

5. Uncertainty and unpredictability: Earthquakes are inherently unpredictable, and even the most sophisticated models cannot predict with complete accuracy when or where an earthquake will occur. This can make it challenging to evaluate the effectiveness of different prediction methods and models.

WAYS TO FIX THOSE PROBLEMS:

1. Collect high-quality data: To ensure the accuracy of predictions, it is important to collect as much high-quality data as possible. This includes data on past earthquakes, geological features, and other relevant factors.

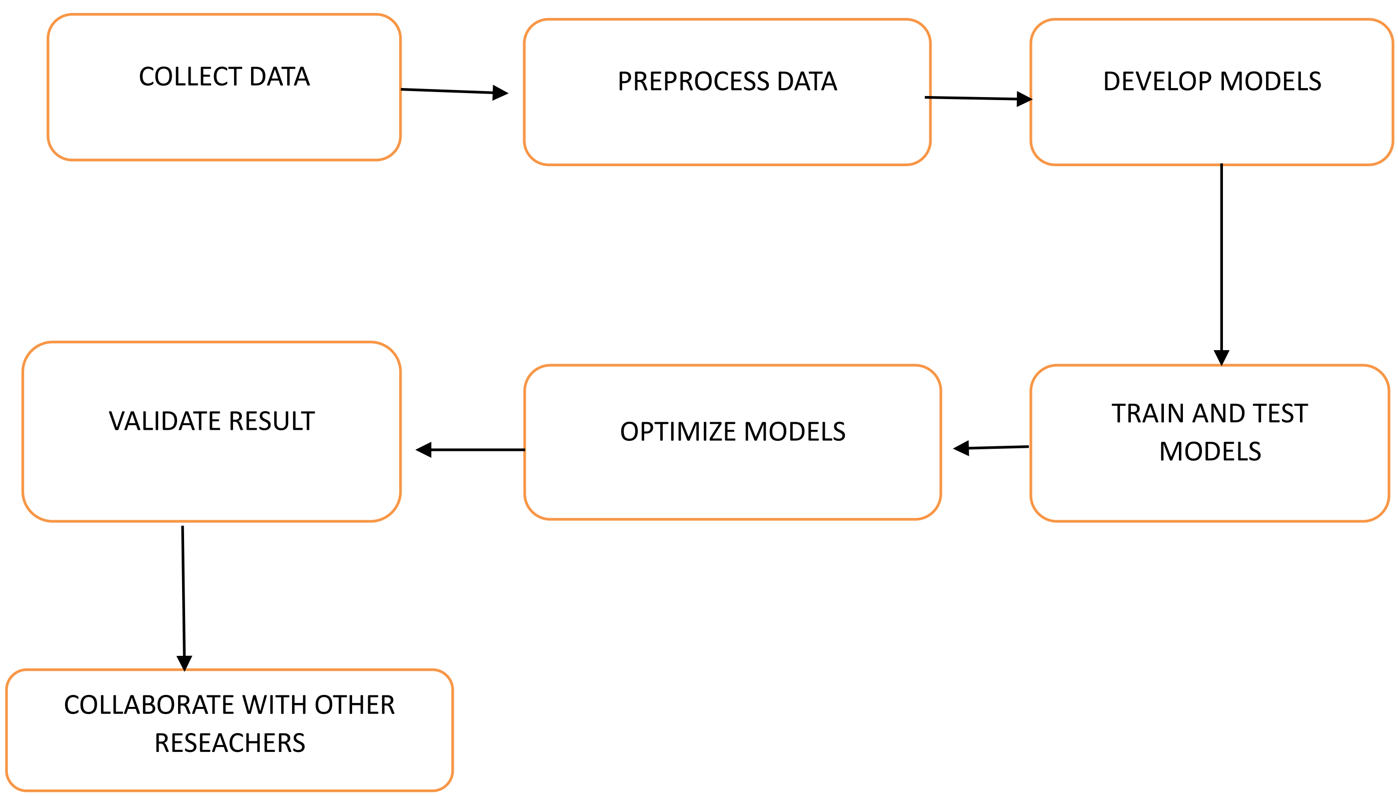
2. Develop robust models: To develop robust models, it is important to use well-established mathematical and statistical techniques. This may involve using machine learning algorithms, time series analysis, or other advanced techniques.

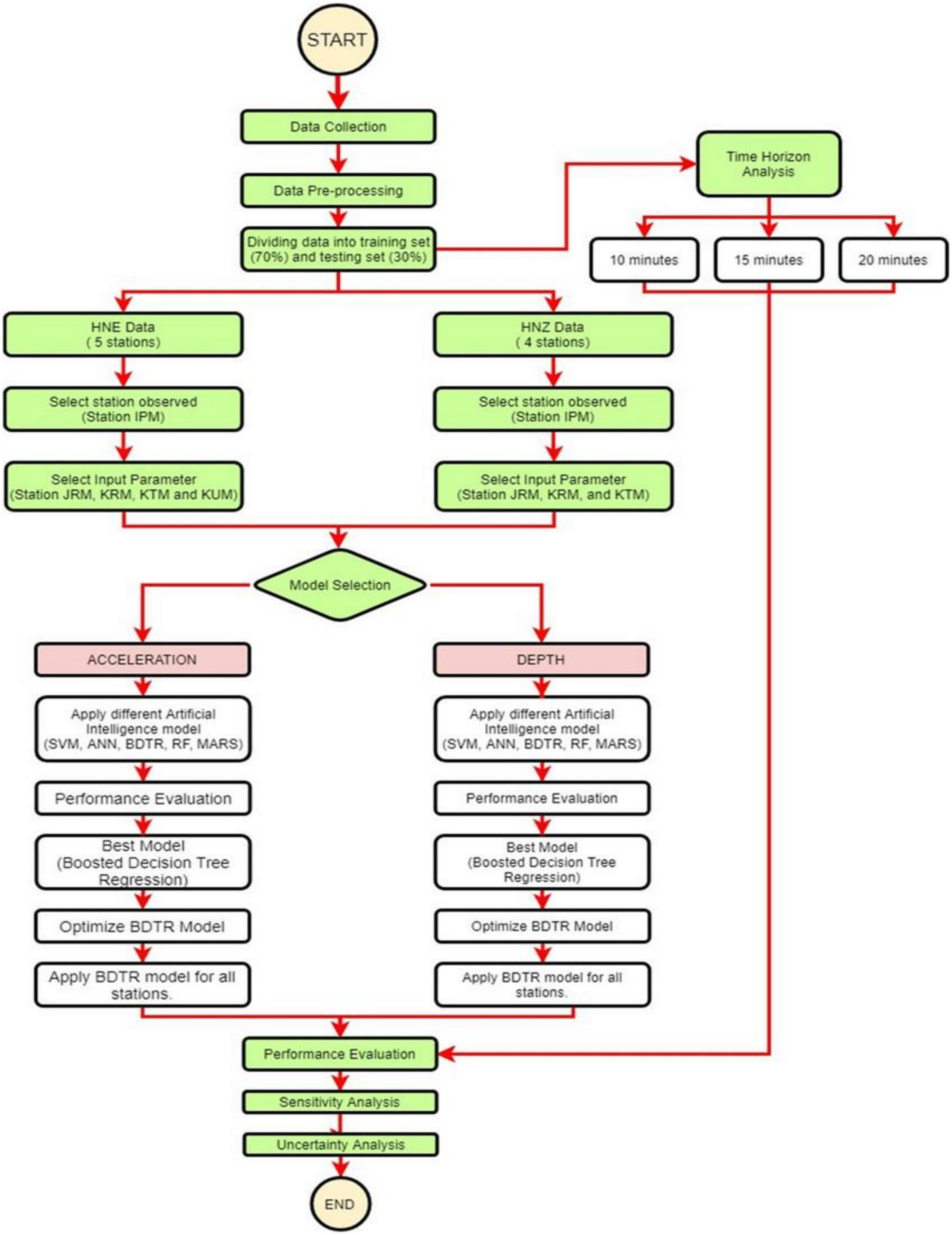
3. Optimize computational resources: To ensure that predictions are made in a timely and accurate manner, it is important to optimize computational resources. This may involve using parallel processing techniques or distributed computing methods.

4. Validate results: To ensure that predictions are accurate and reliable, it is important to validate results against real-world data. This can involve comparing predicted earthquake locations and magnitudes against actual earthquake data.

5. Collaborate with other researchers: To ensure that predictions are standardized and validated across different studies, it is important to collaborate with other researchers in the field. This can involve sharing data, models, and results, as well as participating in peer review processes.

DESIGN:




START
Data Collection
Data Pre-processing
Dividing data into training set (70%) and testing set (30%)
Time Horizon Analysis
10 minutes
15 minutes
20 minutes
HNE Data (5 stations)
HNZ Data (4 stations)
Select station observed (Station IPM)
Select station observed (Station IPM)
Select Input Parameter (Station JRM, KRM, KTM and KUM)
Select Input Parameter (Station JRM, KRM, and KTM)
Model Selection
ACCELERATION
DEPTH
Apply different Artificial Intelligence model (SVM, ANN, BDTR, RF, MARS)
Apply different Artificial Intelligence model (SVM, ANN, BDTR, RF, MARS)
Performance Evaluation
Performance Evaluation
Best Model (Boosted Decision Tree Regression)
Best Model (Boosted Decision Tree Regression)
Optimize BDTR Model
Optimize BDTR Model
Apply BDTR model for all stations.
Apply BDTR model for all stations.
Performance Evaluation
Sensitivity Analysis
Uncertainty Analysis
END

DESIGN:

Designing an earthquake prediction model using Python and artificial intelligence (AI) is a complex undertaking due to the inherent challenges in predicting earthquakes accurately. Accurate short-term earthquake prediction remains a significant scientific challenge, and most efforts focus on earthquake early warning systems or forecasting long-term seismic activity trends. Below is a high-level design for an AI-based earthquake prediction model:

1. Data Collection and Preprocessing:

Gather a comprehensive dataset of historical seismic data, including earthquake magnitudes, locations, depths, and timestamps, from sources like the United States Geological Survey (USGS) or other relevant organizations. Preprocess the data to handle missing values, outliers, and inconsistencies Convert timestamps into a consistent format and resample if necessary.

2. Feature Engineering:

Extract meaningful features from the data, including temporal trends, spatial information, geological characteristics, and environmental factors such as temperature, humidity, and atmospheric pressure. Normalize or standardize the features to ensure they have similar scales and are suitable for AI models.

3. AI Model Selection:

Choose an appropriate AI technique for your prediction task. Given the complexities of earthquake prediction, you might consider using deep learning, specifically recurrent neural networks (RNNs) or convolutional neural networks (CNNs), to capture temporal and spatial dependencies.

4. Model Architecture:

Design the model architecture to accommodate the temporal and spatial aspects of seismic data. Consider using RNNs or CNNs for time-series data and spatial analysis. Experiment with various model architectures, layer sizes, and activation functions to optimize performance.

5. Training and Validation:

Split your dataset into training, validation, and testing sets to evaluate your model's performance. Train your AI model on the training data, monitor its performance on the validation set, and use techniques like early stopping to prevent overfitting.

6. Evaluation Metrics:

Define relevant evaluation metrics for your model. For regression tasks (e.g., predicting earthquake magnitudes or times), consider using Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE). For classification tasks, metrics like accuracy, precision, recall, and F1-score can be used.

7. Hyper parameter Tuning:

Conduct hyper parameter tuning using techniques like grid search, random search, or Bayesian optimization to optimize your model’s performance.

8. Cross-Validation:

Implement cross-validation methods (e.g., k-fold cross-validation) to assess your model’s generalization performance and ensure it doesn’t overfit the training

data.

9. Real-time Data Integration:

If your goal is real-time prediction, develop a data pipeline capable of ingesting and preprocessing data from seismic sensors or other sources in real-time.

10. Model Deployment:

Deploy your trained AI model in a production environment using frameworks like Flask, Django, or Fast API for real-time predictions.

11. Continuous Learning and Ethical Considerations:

Implement mechanisms for regular model retraining and updates as new data becomes available.

INNOVATION IN MY PROJECT:

1. Advanced Deep Learning Models:

Explore state-of-the-art deep learning architectures, such as Transformer-based models, to analyse complex temporal and spatial patterns in seismic data. These models can capture long-range dependencies and may reveal previously unseen earthquake precursors.

2. Graph Neural Networks (GNNs):

Utilize GNNs to model the complex relationships between seismic sensors, fault lines, and geological features. GNNs are well-suited for data with graph structures and can help uncover hidden patterns.

3. Multi-Modal Data Fusion:

Combine data from various sources, including seismometers, GPS sensors, satellite imagery, social media, and geological data, using techniques like multi-modal fusion networks. This can provide a more comprehensive view of the Earth's dynamics.

4. Anomaly Detection:

Train anomaly detection models using unsupervised learning to identify unusual patterns or deviations in seismic data. Unusual patterns could potentially be early indicators of seismic activity.

5. Transfer Learning:

Transfer pre-trained models from related fields (e.g., climate modelling or geophysics) and fine-tune them for earthquake prediction tasks. This can leverage existing knowledge and adapt it to seismic data.

6. Geospatial Analysis:

Combine AI with geographic information systems (GIS) to analyse the spatial relationships between seismic events, fault lines, and geological features. This can provide insights into the likelihood of earthquakes in specific regions.

7. Real-time Data Processing:

Develop real-time data processing pipelines to handle continuous streams of data from sensors. This is crucial for early warning systems and rapid response to seismic events.

8. Hybrid Models:

Combine physics-based models with AI techniques. Integrating knowledge of geological processes with machine learning can lead to more accurate predictions.

9. Ethical Considerations:

Pay close attention to the ethical and societal implications of earthquake prediction. Ensure responsible communication of findings and consider the impact of false alarms.

10. Collaboration:

Collaborate with experts in geophysics, seismology, and earthquake engineering to gain domain-specific insights and validate AI models against real-world data.

11. Quantifying Uncertainty:

Develop methods to quantify and communicate uncertainty in earthquake predictions. Uncertainty estimation is critical in decision-making and risk assessment.

12. Open Data and Collaboration Platforms:

Support open data initiatives and collaborate on platforms that facilitate data sharing and collaborative research among scientists world-wide.

BLOCKS TO ADD IN DESIGN:

Innovation is a multi-faceted process, and incorporating various elements can help foster creativity and problem-solving. Here are some key “blocks” we consider

when designing for innovation:

1. User-Centric Design: Start by understanding the needs, desires, and pain points of your target audience. Design solutions that address their specific challenges.

2. Cross-Disciplinary Teams: Bring together individuals from different backgrounds and areas of expertise. This diversity can lead to fresh perspectives and unique solutions.

3. Empathy and Observation: Put yourself in the shoes of the end user. Observe their behaviours and experiences to gain deeper insights into their needs.

4. Problem Definition: Clearly define the problem you're trying to solve. A well- defined problem statement sets the foundation for a focused and effective solution.

5. Brainstorming and Ideation: Encourage open and creative thinking sessions. Generate a wide range of ideas, even seemingly unconventional ones.

6. Prototyping and Iteration: Create prototypes or mock-ups to test and refine your ideas. Iterative processes allow for continuous improvement.

7. Risk-Taking and Experimentation: Be willing to take calculated risks. Experiment with new technologies, methodologies, or approaches to find breakthrough solutions.

8. Feedback Loops: Seek feedback from various stakeholders, including end users, throughout the design process. This helps refine and validate your ideas.

9. Research and Market Analysis: Stay informed about industry trends, emerging technologies, and potential competitors. This knowledge can guide your design decisions.

10. Ethical Considerations: Ensure that your innovations align with ethical standards and societal values. Avoid potential harm or negative impacts on individuals or communities.

11. Sustainability and Environmental Impact: Consider the environmental implications of your design. Aim for solutions that are eco-friendly and sustainable.

12. Resource Allocation and Constraints: Be mindful of budget, time, and resource constraints. Efficient allocation of resources is crucial for successful implementation.

13. Collaboration and Communication: Foster a culture of collaboration within your team. Effective communication ensures that everyone is aligned and working towards a common goal.

14. Continuous Learning and Adaptation: Stay curious and open to learning. Embrace change and be willing to adapt your approach based on new information or feedback.

CHANGES IN DESIGN:

We implement changes in the design of an earthquake prediction model using Python for innovation, in consider the following things:

1. Improved Data Collection: Enhance the data collection process by incorporating real-time sensor data, satellite imagery, or any other relevant sources. This can lead to more accurate predictions.

2. Feature Engineering: Experiment with different features that could be indicative of earthquake occurrence. This might include geological, meteorological, or even social factors.

3. Advanced Machine Learning Algorithms: Explore more sophisticated algorithms like Random Forest, Support Vector Machines, or even neural networks to improve prediction accuracy.

4. Ensemble Methods: Utilize ensemble methods like bagging or boosting to combine multiple models for a more robust and accurate prediction.

5. Hyperparameter Tuning: Fine-tune the parameters of your chosen algorithms to achieve better performance. This can be done using techniques like grid search or random search.

6. Temporal Analysis: Incorporate time series analysis techniques to capture patterns and trends in seismic activity over time.

7. Spatial Analysis: Implement geospatial analysis to understand how earthquake patterns vary across different regions.

8. Probabilistic Models: Consider using probabilistic models to provide not only a prediction but also an associated probability or confidence level.

9. Feature Importance Analysis: Understand which features have the most impact on predictions. This can help in refining the feature set.

10. Model Interpretability: Ensure that the model is interpretable, so that users can understand why a certain

prediction was made. Techniques like SHAP values or LIME can be helpful.

11. Cross-Validation and Validation Sets: Implement robust validation techniques to ensure the model's generalizability and performance on unseen data.

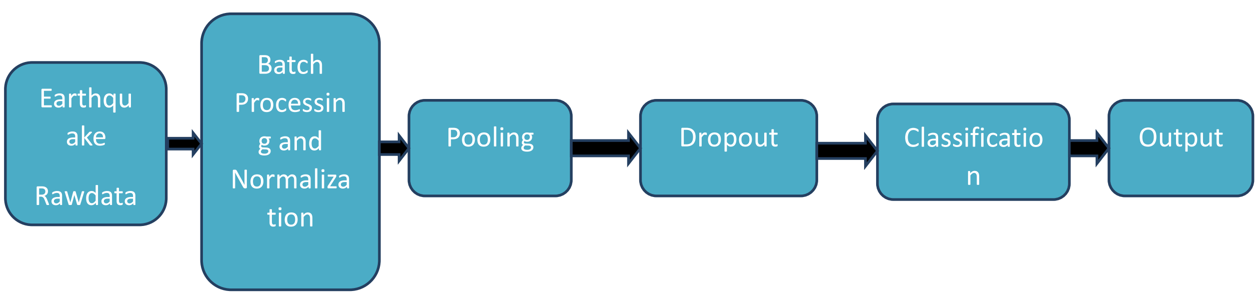
12. Incorporate External Datasets: Integrate other relevant datasets that might provide additional context or information related to seismic activity.

13. Real-time Monitoring: Develop a mechanism to continuously monitor the model's performance and retrain it as new data becomes available.

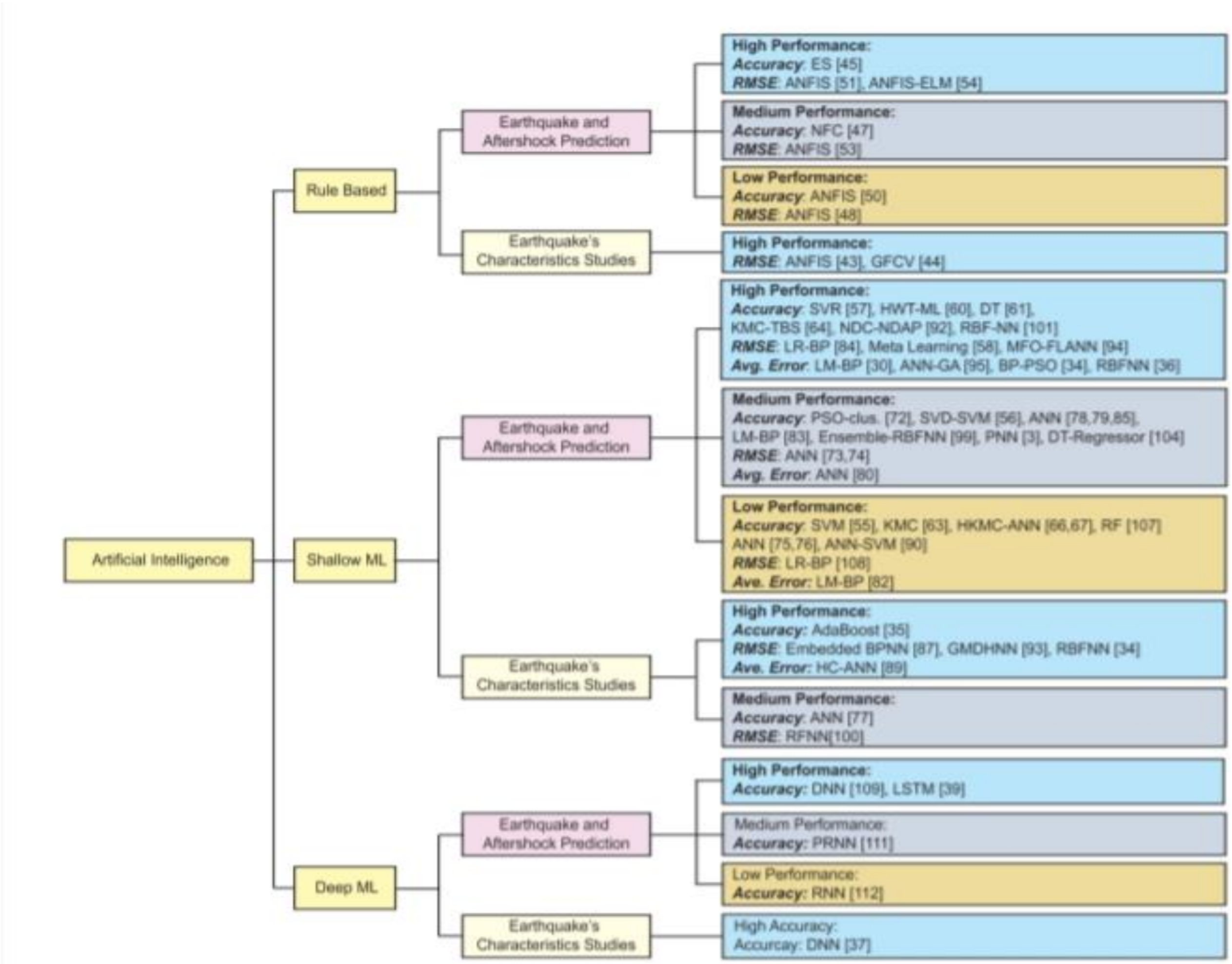
14. User-Friendly Interface: Create an intuitive interface for users to interact with the prediction model, providing clear visualizations and explanations of the predictions.

15. Ethical Considerations: Address any ethical concerns related to data privacy, bias, and potential societal impact of the predictions.

DESIGN BLOCK DIAGRAM:



DNN based Earthquake prediction model using python



LOADING AND PREPROCESSING:

Loading and preprocessing data for an earthquake prediction model in Python can be a complex task. Here are some challenges you might encounter:

1. “Data Collection”: Acquiring accurate and comprehensive earthquake data can be a challenge. You'll need to rely on sources like USGS, which provide earthquake data in various formats.

2. “Data Quality”: Earthquake data can be noisy, incomplete, or contain errors. Preprocessing may involve data cleaning and dealing with missing values.

3. “Data Volume”: Earthquake data can be vast, especially if you're working with historical records. Handling large datasets efficiently is essential.

4. “Data Format”: Earthquake data may come in various formats, such as CSV, JSON, or XML. You need to parse and convert it into a suitable format for your model.

5. “Feature Engineering”: Selecting the right features and engineering relevant ones is crucial. Geospatial and temporal data may require special treatment.

6. “Geospatial Data”: If your model involves geospatial data, you'll need to work with libraries like GeoPandas, and handle spatial data operations and transformations.

PROGRAM AND OUTPUT

In [398]:

```
import numpy as np
import pandas as pd
import requests
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from pandas.plotting import scatter_matrix
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import time
```

In [399]:

```
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/earthquake_prediction/earthquake1.csv")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [400]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24007 entries, 0 to 24006
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   id         24007 non-null  float64
 1   date       24007 non-null  object
 2   time       24007 non-null  object
 3   lat        24007 non-null  float64
 4   long       24007 non-null  float64
 5   country    24007 non-null  object
 6   city       11754 non-null  object
 7   area       12977 non-null  object
 8   direction  10062 non-null  object
 9   dist       10062 non-null  float64
 10  depth      24007 non-null  float64
 11  xm         24007 non-null  float64
 12  md         24007 non-null  float64
 13  richter    24007 non-null  float64
 14  mw         5003 non-null   float64
 15  ms         24007 non-null  float64
 16  mb         24007 non-null  float64
dtypes: float64(11), object(6)
memory usage: 3.1+ MB
```

In [401]:

```
df.describe()
```

Out[401]:

	id	lat	long	dist	depth	xm	md	richter	mw	ms	mb

	id	lat	long	dist	depth	xm	md	richter	mw	ms	mb
count	2.400700e+04	24007.000000	24007.000000	10062.000000	24007.000000	24007.000000	24007.000000	24007.000000	5003.000000	24007.000000	24007.000000
mean	1.991982e+13	37.929474	30.773229	3.175015	18.491773	4.056038	1.912346	2.196826	4.478973	0.677677	1.690561
std	2.060396e+11	2.205605	6.584596	4.715461	23.218553	0.574085	2.059780	2.081417	1.048085	1.675708	2.146108
min	1.910000e+13	29.740000	18.340000	0.100000	0.000000	3.500000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.980000e+13	36.190000	26.195000	1.400000	5.000000	3.600000	0.000000	0.000000	4.100000	0.000000	0.000000
50%	2.000000e+13	38.200000	28.350000	2.300000	10.000000	3.900000	0.000000	3.500000	4.700000	0.000000	0.000000
75%	2.010000e+13	39.360000	33.855000	3.600000	22.400000	4.400000	3.800000	4.000000	5.000000	0.000000	4.100000
max	2.020000e+13	46.350000	48.000000	95.400000	225.000000	7.900000	7.400000	7.200000	7.700000	7.900000	7.100000

In [402]:

```
df.shape
```

Out[402]:

```
(24007, 17)
```

In [403]:

```
df.head()
```

Out[403]:

	id	date	time	lat	long	country	city	area	direction	dist	depth	xm	md	richter	mw	ms	mb
0	2.000000e+13	2003.05.20	12:17:44 AM	39.04	40.38	turkey	bingol	baliklicay	west	0.1	10.0	4.1	4.1	0.0	NaN	0.0	0.0
1	2.010000e+13	2007.08.01	12:03:08 AM	40.79	30.09	turkey	kocaeli	bayraktar_izmit	west	0.1	5.2	4.0	3.8	4.0	NaN	0.0	0.0
2	1.980000e+13	1978.05.07	12:41:37 AM	38.58	27.61	turkey	manisa	hamzabeyli	south_west	0.1	0.0	3.7	0.0	0.0	NaN	0.0	3.7
3	2.000000e+13	1997.03.22	12:31:45 AM	39.47	36.44	turkey	sivas	kahvepinar_sarkisla	south_west	0.1	10.0	3.5	3.5	0.0	NaN	0.0	0.0
4	2.000000e+13	2000.04.02	12:57:38 AM	40.80	30.24	turkey	sakarya	meseli_serdivan	south_west	0.1	7.0	4.3	4.3	0.0	NaN	0.0	0.0

In [404]:

```
df.columns
```

Out[404]:

```
Index(['id', 'date', 'time', 'lat', 'long', 'country', 'city', 'area',
       'direction', 'dist', 'depth', 'xm', 'md', 'richter', 'mw', 'ms',
'mb'],
      dtype='object')
```

Data Preprocessing

In [405]:

```
df = df.drop('id',axis=1)
```

In [406]:

```
import datetime
import time

timestamp = []
for d, t in zip(df['date'], df['time']):
  ts = datetime.datetime.strptime(d+' '+t, '%Y.%m.%d %I:%M:%S %p')
  timestamp.append(time.mktime(ts.timetuple()))
timeStamp = pd.Series(timestamp)
df['Timestamp'] = timeStamp.values
final_data = df.drop(['date', 'time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
df = final_data
df.head()
```

Out[406]:

	lat	long	country	city	area	direction	dist	depth	xm	md	richter	mw	ms	mb	Timestamp
0	39.04	40.38	turkey	bingol	baliklicay	west	0.1	10.0	4.1	4.1	0.0	NaN	0.0	0.0	1.053390e+09
1	40.79	30.09	turkey	kocaeli	bayraktar_izmit	west	0.1	5.2	4.0	3.8	4.0	NaN	0.0	0.0	1.185927e+09
2	38.58	27.61	turkey	manisa	hamzabeyli	south_west	0.1	0.0	3.7	0.0	0.0	NaN	0.0	3.7	2.633497e+08

	lat	long	country	city	area	direction	dist	depth	xm	md	richter	mw	ms	mb	Timestamp
3	39.47	36.44	turkey	sivas	kahvepinar_sarkisla	south_west	0.1	10.0	3.5	3.5	0.0	NaN	0.0	0.0	8.589907e+08
4	40.80	30.24	turkey	sakarya	meseli_serdivan	south_west	0.1	7.0	4.3	4.3	0.0	NaN	0.0	0.0	9.546371e+08

In [407]:

```
df.dtypes
```

Out[407]:

```
lat            float64
long           float64
country         object
city            object
area            object
direction       object
dist           float64
depth          float64
xm             float64
md             float64
richter        float64
mw             float64
ms             float64
mb             float64
Timestamp      float64
dtype: object
```

In [408]:

```
# Data Encoding
label_encoder = preprocessing.LabelEncoder()
for col in df.columns:
    if df[col].dtype == 'object':
      label_encoder.fit(df[col])
      df[col] = label_encoder.transform(df[col])
df.dtypes
```

Out[408]:

```
lat            float64
long           float64
country          int64
city             int64
```

```
area           int64
direction      int64
dist         float64
depth        float64
xm           float64
md           float64
richter      float64
mw           float64
ms           float64
mb           float64
Timestamp    float64
dtype: object
```

In [409]:

```
df.isnull().sum()
```

Out[409]:

```
lat              0
long             0
country          0
city             0
area             0
direction        0
dist         13945
depth            0
xm               0
md               0
richter          0
mw           19004
ms               0
mb               0
Timestamp        0
dtype: int64
```

In [410]:

```
# Imputing Missing Values with Mean
si=SimpleImputer(missing_values = np.nan, strategy="mean")
si.fit(df[["dist","mw"]])
df[["dist","mw"]] = si.transform(df[["dist","mw"]])
df.isnull().sum()
```

Out[410]:

```
lat          0
long         0
country      0
city         0
area         0
direction    0
dist         0
depth        0
xm           0
md           0
```

```
richter      0
mw           0
ms           0
mb           0
Timestamp    0
dtype: int64
```

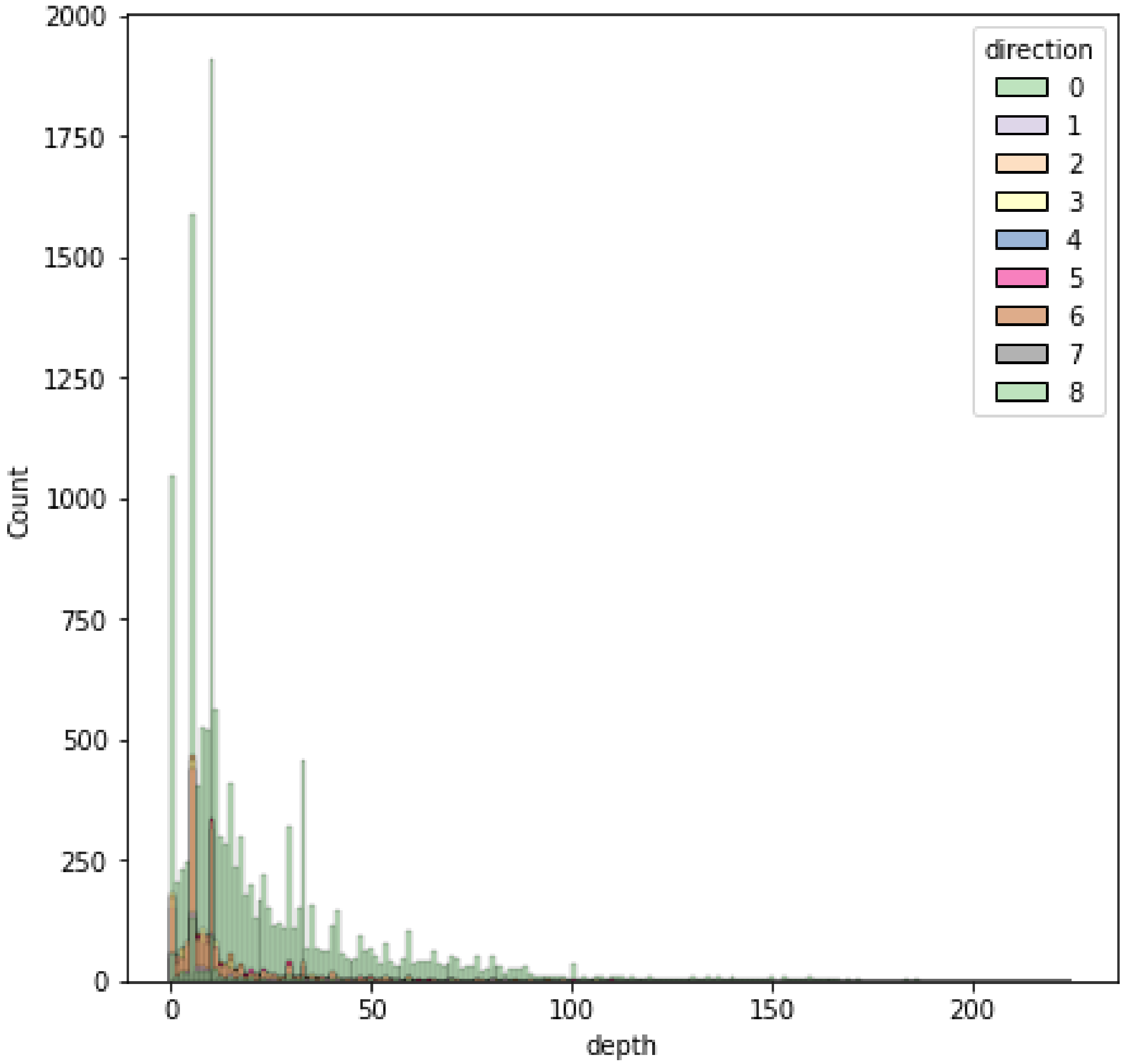
Data Visualization

In [411]:

```
import plotly.express as px
px.scatter(df, x='richter',y='xm', color="direction")
```

In [412]:

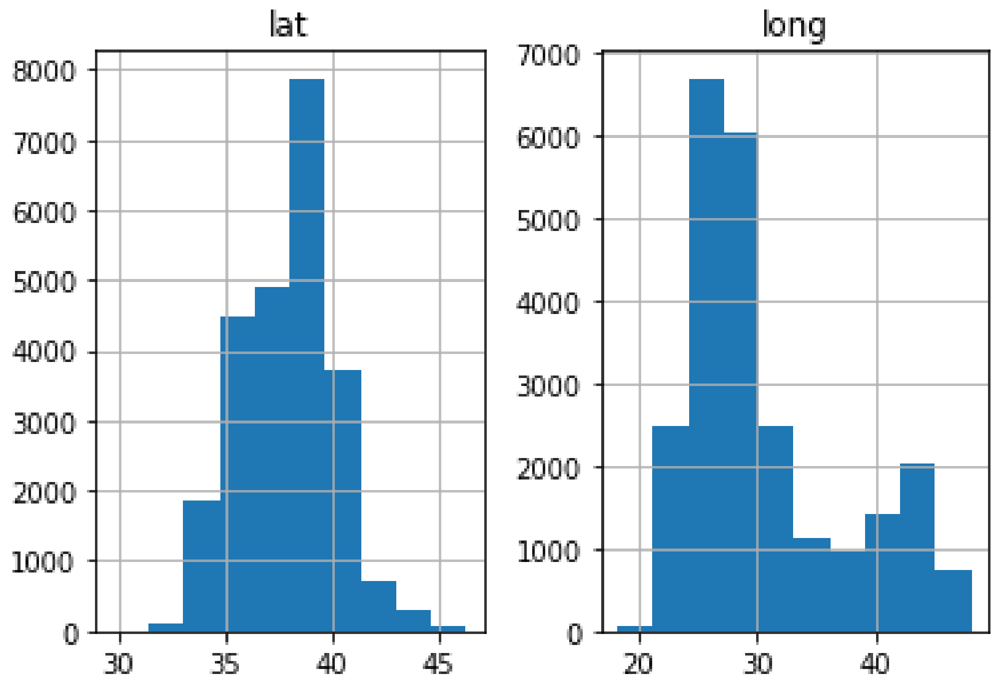
```
plt.figure(figsize=(7,7))
sns.histplot(data=df, x='depth', hue='direction',palette = 'Accent')
plt.show()
```



In [413]:

```
plt.figure(figsize=(7,7))
df[['lat','long']].hist()
plt.show()
```

```
<Figure size 504x504 with 0 Axes>
```



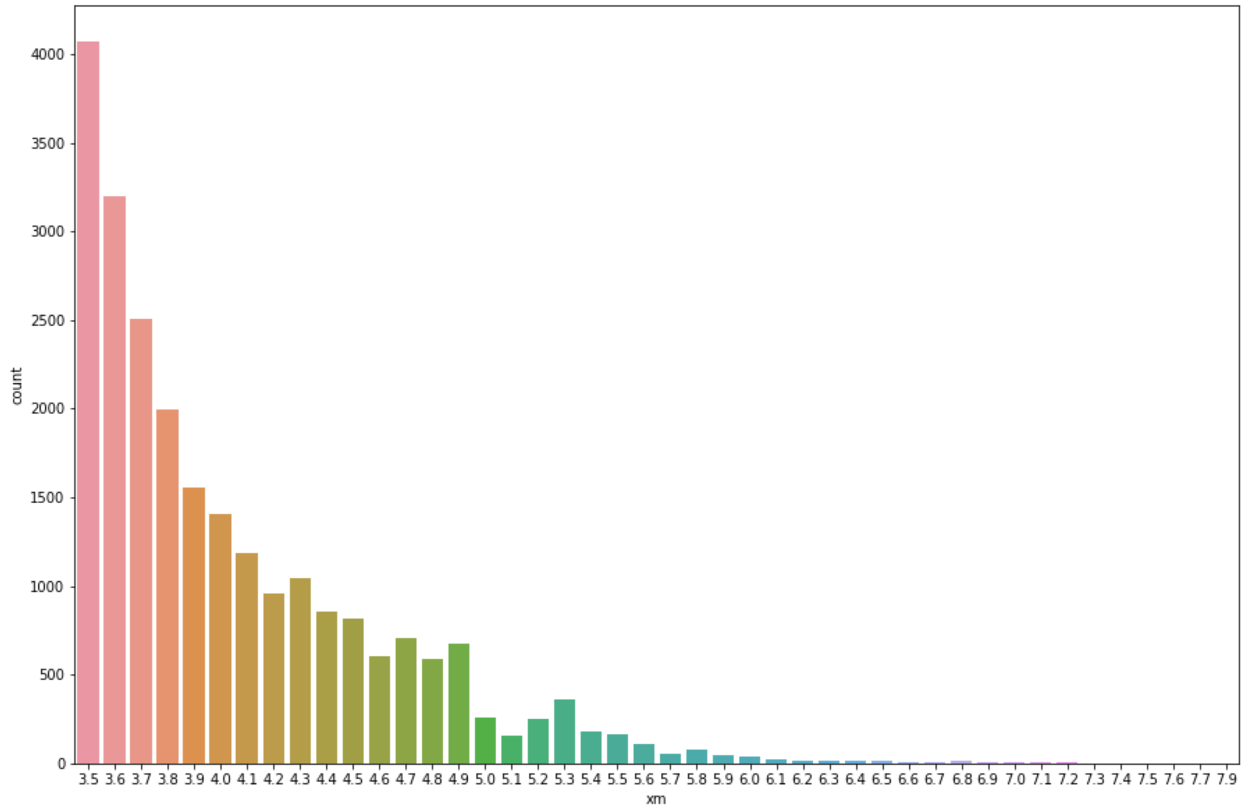
In [414]:

```
plt.figure(figsize=(15,10))
sns.countplot(df.xm)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36:
FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
```

Out[414]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d2346d400>
```



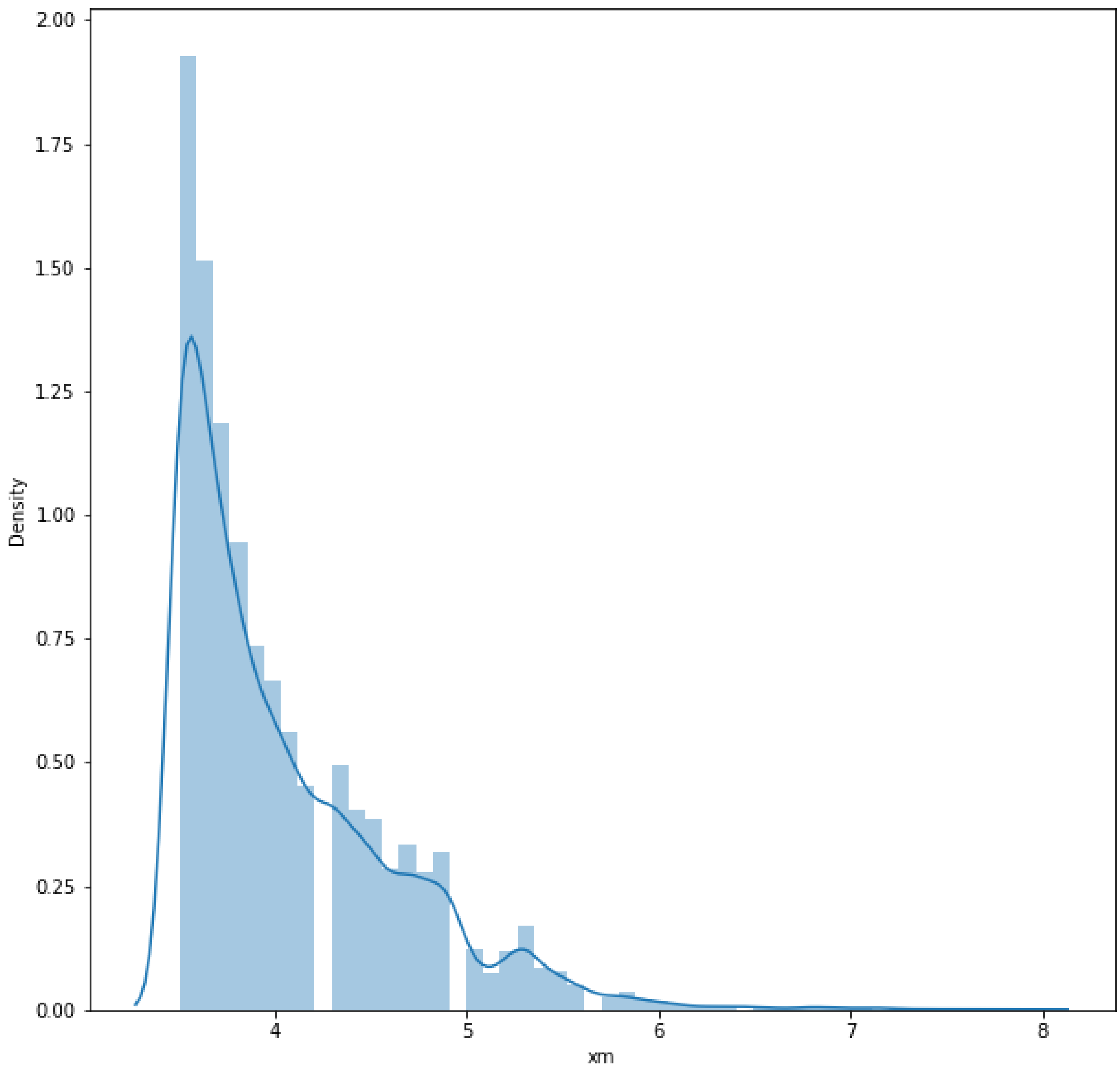
In [415]:

```
plt.figure(figsize=(10,10))
sns.distplot(df.xm)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

Out[415]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d242a4d00>
```

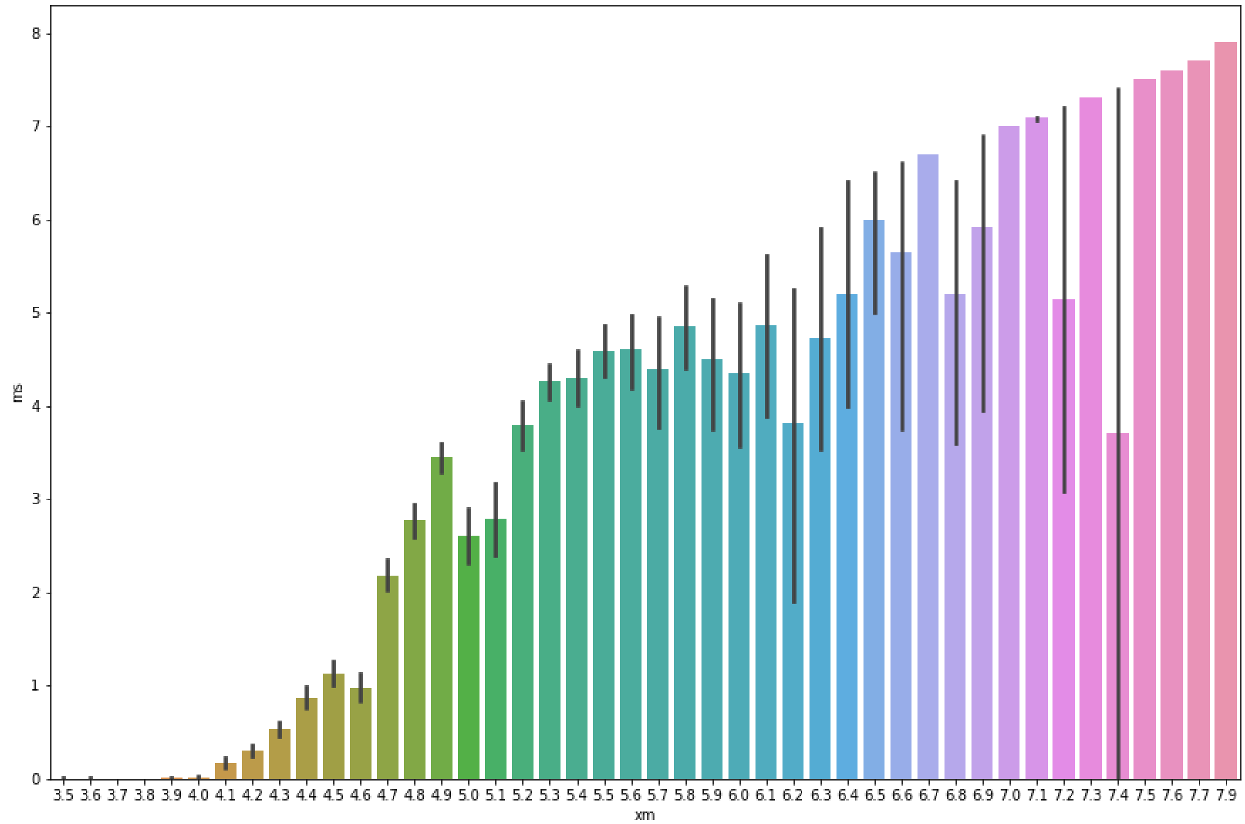


In [416]:

```
plt.figure(figsize=(15,10))
sns.barplot(x=df['xm'], y=df['ms'])
plt.xlabel('xm')
plt.ylabel('ms')
```

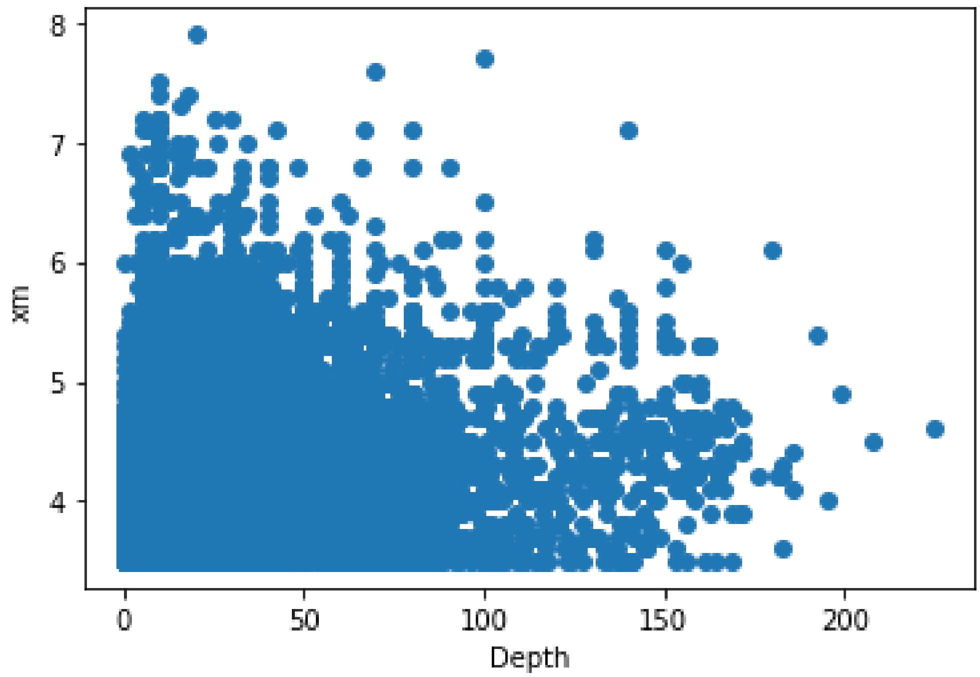
Out[416]:

```
Text(0, 0.5, 'ms')
```



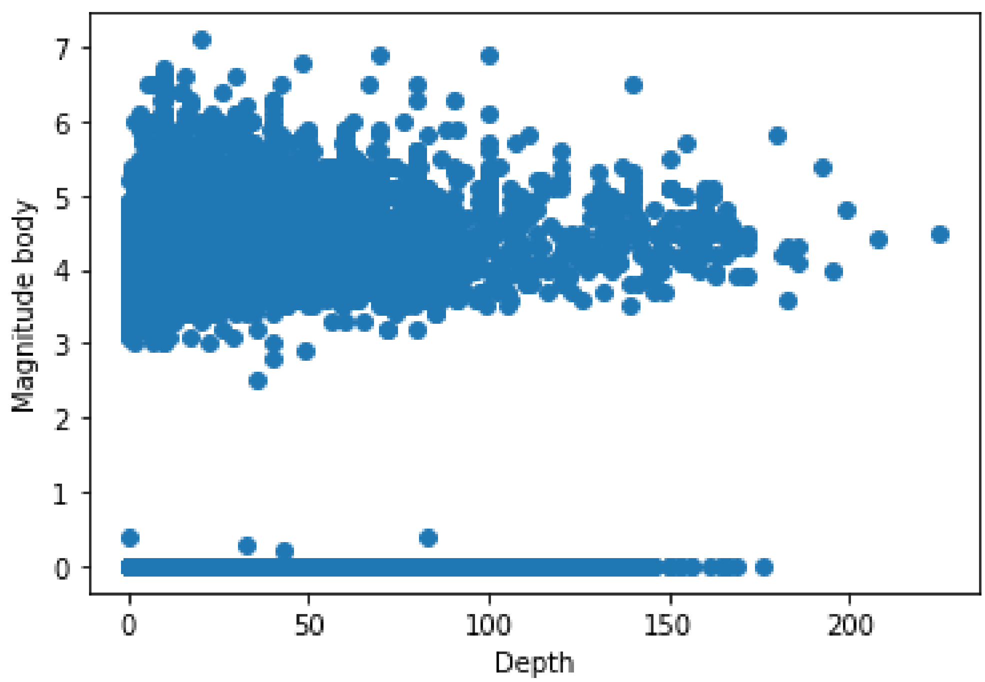
In [417]:

```
plt.scatter(df.depth, df.xm)
plt.xlabel("Depth")
plt.ylabel("xm")
plt.show()
```



In [418]:

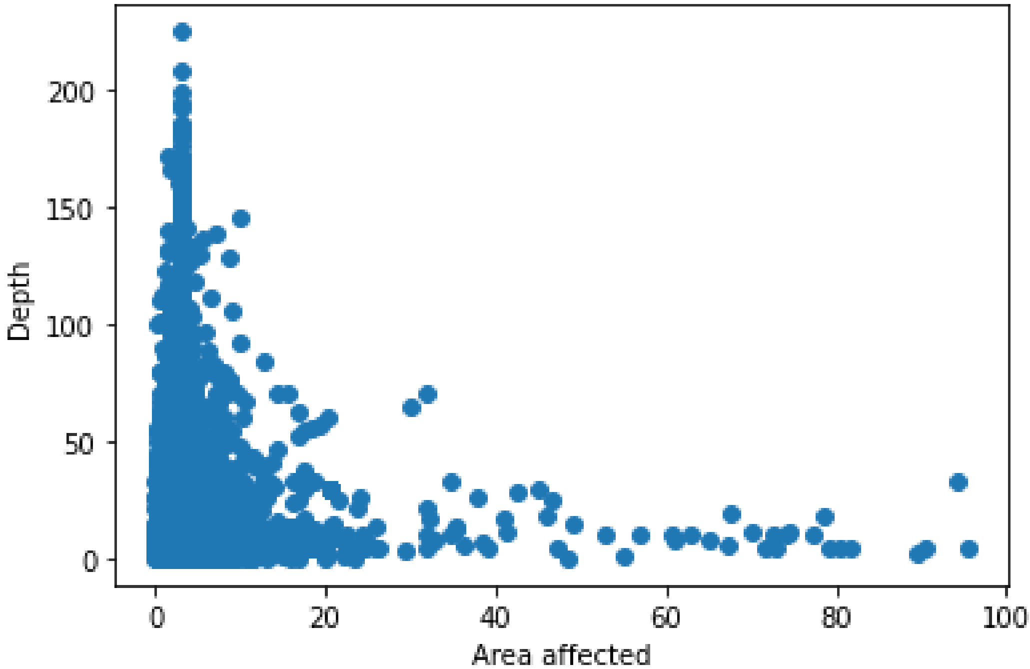
```
plt.scatter(df.depth, df.mb)
plt.xlabel("Depth")
plt.ylabel("Magnitude body")
plt.show()
```



In [419]:

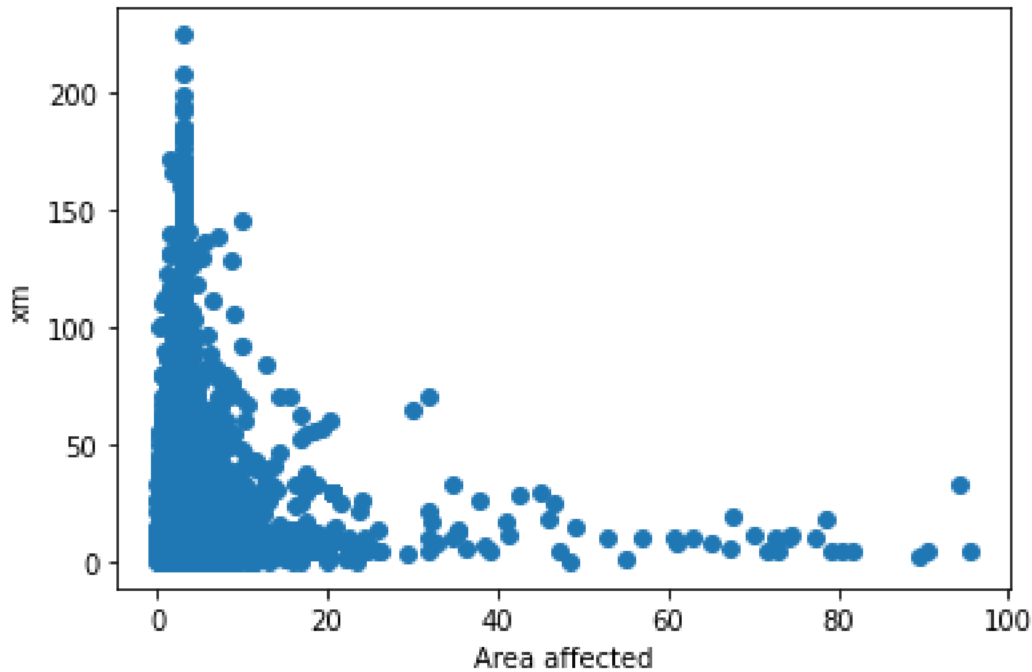
```
plt.scatter(df.dist, df.depth)
```

```
plt.xlabel("Area affected")
plt.ylabel("Depth")
plt.show()
```



In [420]:

```
plt.scatter(df.dist, df.depth)
plt.xlabel("Area affected")
plt.ylabel("xm")
plt.show()
```



Correlation between Attributes

In [421]:

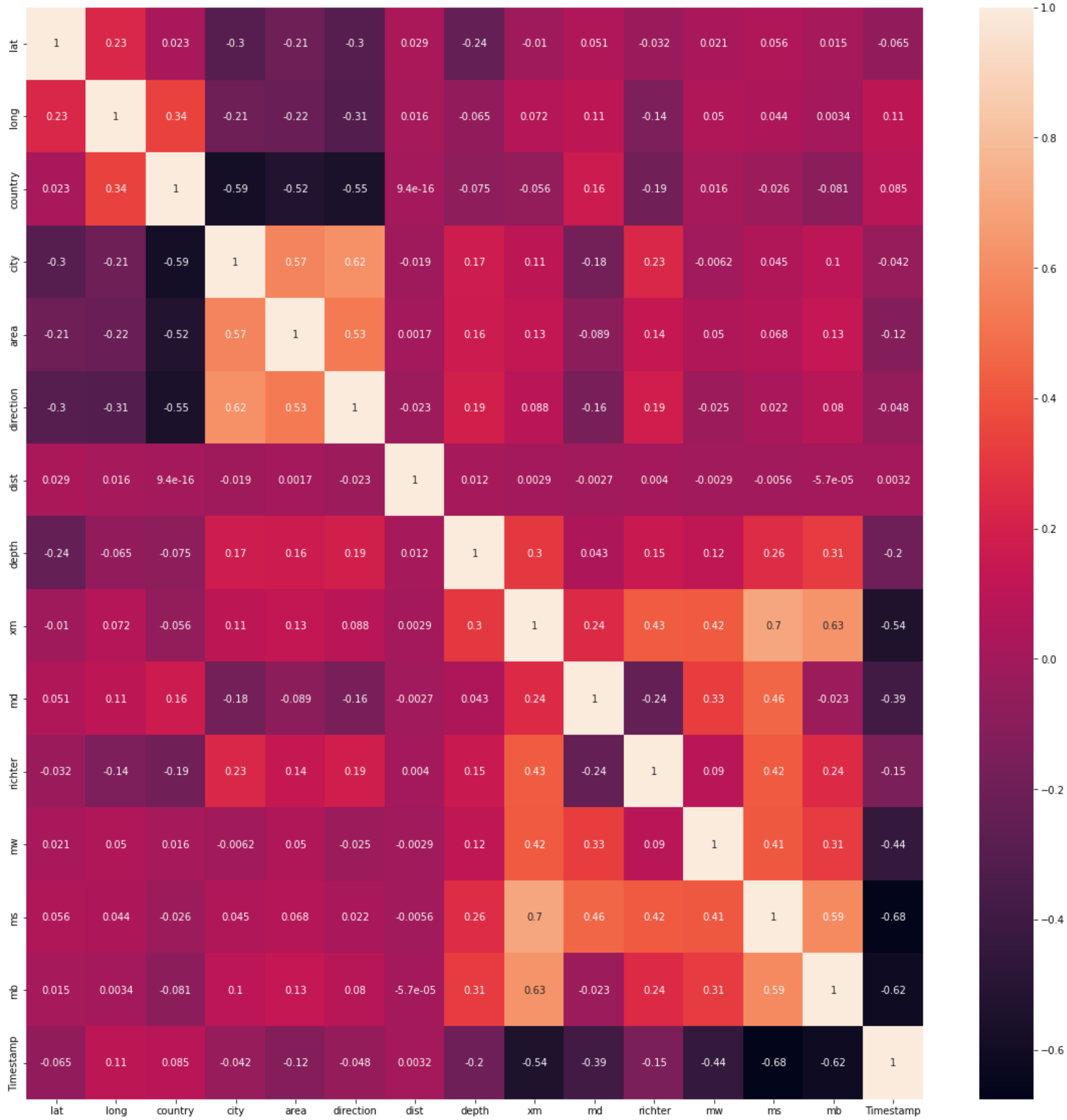
```
most_correlated = df.corr()['xm'].sort_values(ascending=False)
most_correlated
```

Out[421]:

```
xm           1.000000
ms           0.699579
mb           0.628382
richter      0.426653
mw           0.420695
depth        0.302926
md           0.241432
area         0.125275
city         0.107436
direction    0.087696
long         0.071856
dist         0.002853
lat         -0.010347
country     -0.056115
Timestamp   -0.542092
Name: xm, dtype: float64
```

In [422]:

```
plt.figure(figsize=(20,20))
dataplot=sns.heatmap(df.corr(),annot=True)
plt.show()
```



Normalization of data

In [423]:

```
# Using MinMaxScaler
scaler = preprocessing.MinMaxScaler()
d = scaler.fit_transform(df)
df = pd.DataFrame(d, columns=df.columns)
df.head()
```

Out[423]:

	lat	long	country	city	area	direction	dist	depth	xm	md	richter	mw	ms	mb	Timestamp
0	0.559904	0.743088	0.76	0.172043	0.116144	0.875	0.0	0.044444	0.136364	0.554054	0.000000	0.581685	0.0	0.000000	0.866875
1	0.665262	0.396156	0.76	0.612903	0.132306	0.875	0.0	0.023111	0.113636	0.513514	0.555556	0.581685	0.0	0.000000	0.906252
2	0.532210	0.312542	0.76	0.677419	0.459500	0.750	0.0	0.000000	0.045455	0.000000	0.000000	0.581685	0.0	0.521127	0.632149
3	0.585792	0.610249	0.76	0.870968	0.513061	0.750	0.0	0.044444	0.000000	0.472973	0.000000	0.581685	0.0	0.000000	0.809118
4	0.665864	0.401214	0.76	0.806452	0.689344	0.750	0.0	0.031111	0.181818	0.581081	0.000000	0.581685	0.0	0.000000	0.837535

Splitting the Dataset

In [424]:

```
y=np.array(df['xm'])
X=np.array(df.drop('xm',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=0.2,random_state=2)
```

Creating Models

1.Linear Regression

In [425]:

```
from sklearn.linear_model import LinearRegression
start1 = time.time()
linear=LinearRegression()
linear.fit(X_train,y_train)
ans1 = linear.predict(X_test)
end1 = time.time()
t1 = end1-start1
```

In [426]:

```
accuracy1=linear.score(X_test,y_test)
print("Accuracy of Linear Regression model is:",accuracy1)
```

```
Accuracy of Linear Regression model is: 0.63134131503029
```

In [427]:

```
from sklearn import metrics
print("Linear Regression")
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ans1))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ans1))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, ans1)))
```

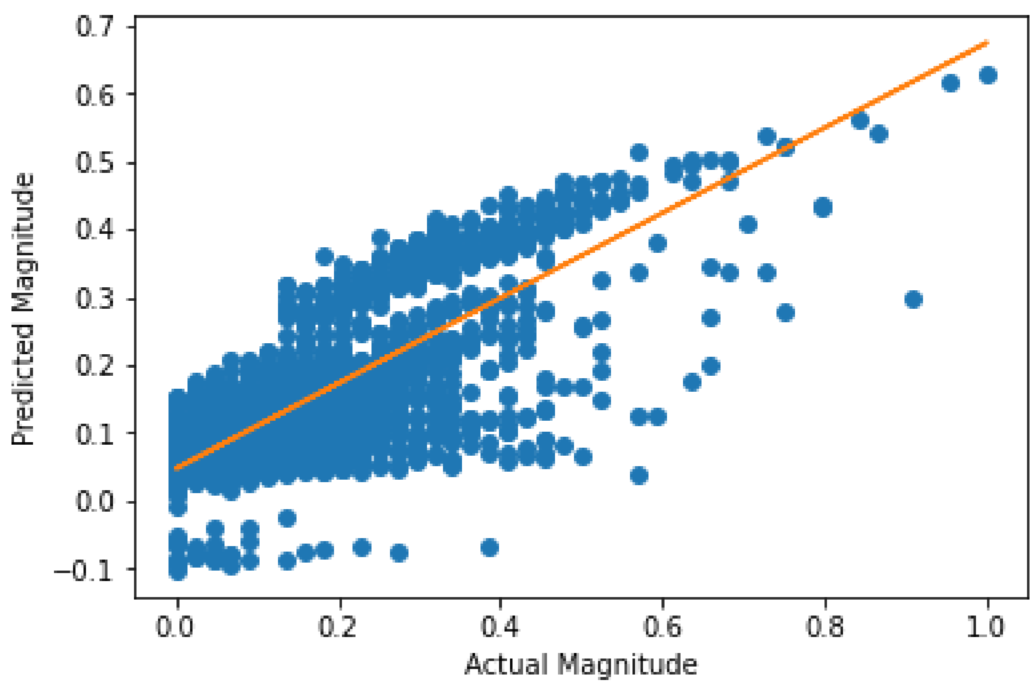
```
Linear Regression
Mean Absolute Error: 0.05878246463205686
Mean Squared Error: 0.006258271697266360
Root Mean Squared Error: 0.07910923901331854
```

In [428]:

```
plt.plot(y_test, ans1, 'o')
m, b = np.polyfit(y_test,ans1, 1)
plt.plot(y_test, m*y_test + b)
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
```

Out[428]:

```
Text(0, 0.5, 'Predicted Magnitude')
```



2.Decision Tree

In [449]:

```
from sklearn.tree import DecisionTreeRegressor
start2 = time.time()
regressor = DecisionTreeRegressor(random_state = 40)
regressor.fit(X_train,y_train)
ans2 = regressor.predict(X_test)
end2 = time.time()
t2 = end2-start2
```

In [450]:

```
accuracy2=regressor.score(X_test,y_test)
print("Accuracy of Decision Tree model is:",accuracy2)
```

```
Accuracy of Decision Tree model is: 0.9932960893884235
```

In [451]:

```
print("Decision Tree")
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ans2))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ans2))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, ans2)))
```

```
Decision Tree
Mean Absolute Error: 0.0006909999621372331
Mean Squared Error: 0.00011380416561969702
Root Mean Squared Error: 0.010667903525046383
```

3.KNN Model

In [432];

```
from sklearn.neighbors import KNeighborsRegressor
start3 = time.time()
knn = KNeighborsRegressor(n_neighbors=6)
knn.fit(X_train, y_train)
ans3 = knn.predict(X_test)
end3 = time.time()
t3 = end3-start3
```

In [433]:

```
accuracy3=knn.score(X_test,y_test)
print("Accuracy of KNN model is:",accuracy3)
```

```
Accuracy of KNN model is: 0.8457466919393031
```

In [434]:

```
print("KNN Model")
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ans3))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ans3))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, ans3)))
```

```
KNN Model
Mean Absolute Error: 0.03305598677318794
Mean Squared Error: 0.002618571462992348
Root Mean Squared Error: 0.051171979275696854
```

In [435]:

```
import random
info = {}
for i in range(10):
  k = random.randint(2,10)
  startk = time.time()
  knn = KNeighborsRegressor(n_neighbors=k)
  knn.fit(X_train, y_train)
  ans3 = knn.predict(X_test)
  endk = time.time()
  tk = endk-startk
  acc3=knn.score(X_test,y_test)
  info[k] = [acc3,tk]

for i in info:
  print("for k =",i,": accuracy =",info[i][0])
```

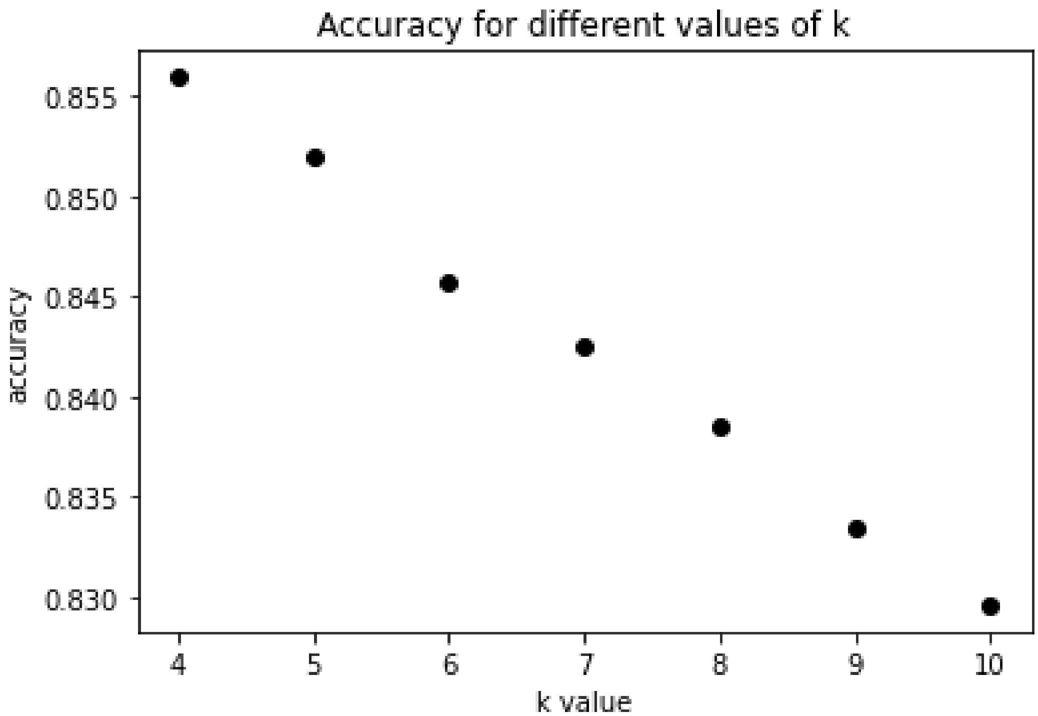
```
for k = 4 : accuracy = 0.8559118607470738
for k = 9 : accuracy = 0.8334625255508568
for k = 8 : accuracy = 0.8384577534478264
for k = 6 : accuracy = 0.8457466919393031
for k = 5 : accuracy = 0.8519381145638621
for k = 10 : accuracy = 0.8296048410841246
for k = 7 : accuracy = 0.8425261199362686
```

In [436]:

```
x = list(info.keys())
yacc = []
for i in info:
  yacc.append(info[i][0])
plt.plot(x, yacc, 'o', color='black');
plt.xlabel("k value")
plt.ylabel("accuracy");
plt.title("Accuracy for different values of k")
```

Out[436]:

```
Text(0.5, 1.0, 'Accuracy for different values of k')
```

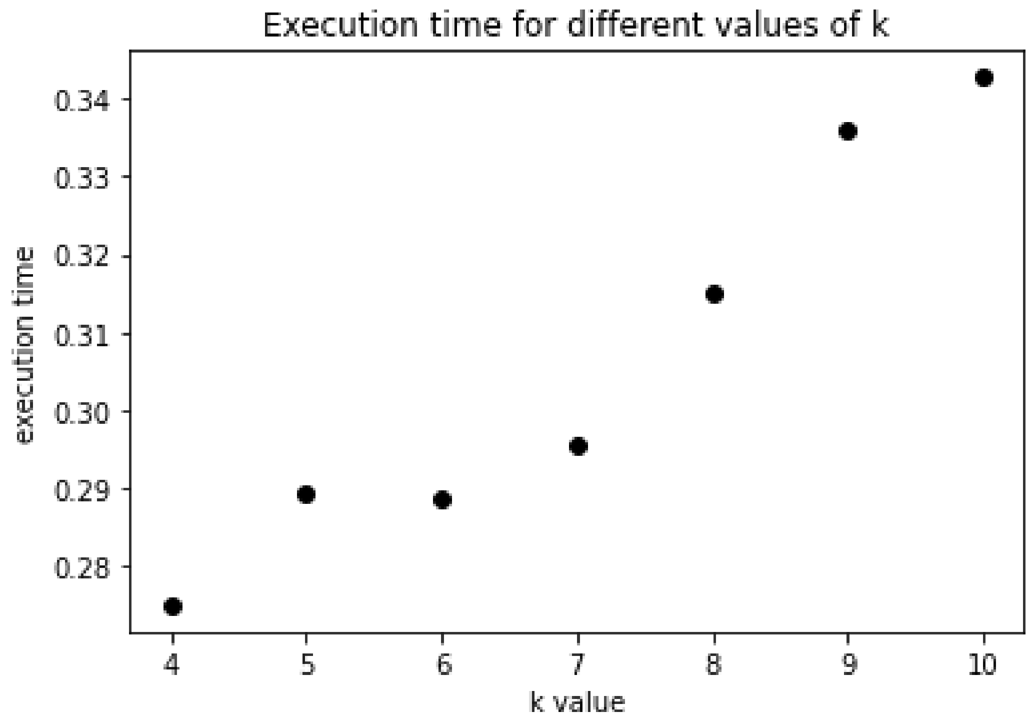


In [437]:

```
yt = []
for i in info:
  yt.append(info[i][1])
plt.plot(x, yt, 'o', color='black');
plt.xlabel("k value")
plt.ylabel("execution time");
plt.title("Execution time for different values of k")
```

Out[437]:

```
Text(0.5, 1.0, 'Execution time for different values of k')
```



Comparison Graphs

1.Accuracy

In [454]:

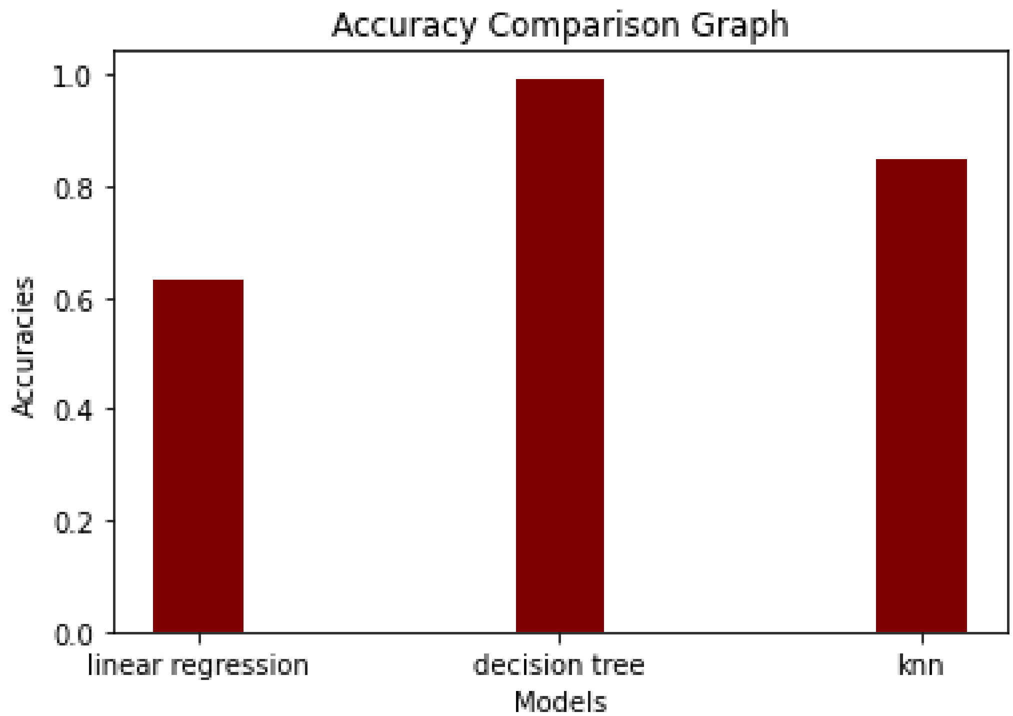
```
models = ["linear regression","decision tree","knn"]
accuracies = [accuracy1,accuracy2,accuracy3]
```

In [455]:

```
plt.bar(models, accuracies, color ='maroon',
        width = 0.25)
plt.xlabel("Models")
plt.ylabel("Accuracies")
plt.title("Accuracy Comparison Graph")
```

Out[455]:

```
Text(0.5, 1.0, 'Accuracy Comparison Graph')
```



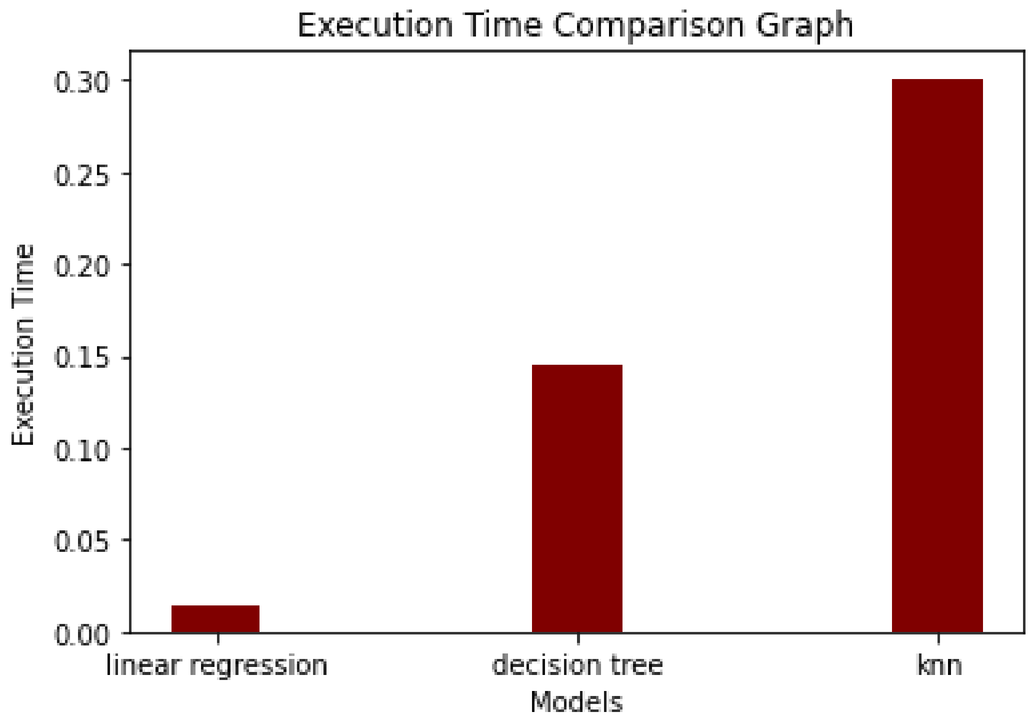
In [456]:

2.Execution Time

```
times = [t1,t2,t3]
plt.bar(models, times, color ='maroon',
        width = 0.25)
plt.xlabel("Models")
plt.ylabel("Execution Time")
plt.title("Execution Time Comparison Graph")
```

Out[456]:

```
Text(0.5, 1.0, 'Execution Time Comparison Graph')
```



EXECUTION STEPS:

1.**Open Google Colab**: Go to [Google Colab](https://colab.research.google.com/) and log in with your Google account.

2. **Create a New Notebook**:
   - Click on "New Notebook" to create a new notebook.
   - A new tab will open with an untitled notebook.

3. **Write Code**:
   - In the cells of the notebook, you can write your Python code.
   - Click on the "+" button to add a new cell.

4. **Running Code**:
   - To run a cell, you can press `Shift+Enter` or click on the "Play" button next to the cell.
   - The output of the code will appear below the cell.

5. **Saving Work**:
   - To save your work, go to "File" and choose "Save" or press `Ctrl+S` (or `Cmd+S` on Mac).

6. **Uploading Data**:
   - If your program requires external files, you can upload them using the file icon on the left sidebar.

7. **Installing Packages**:
   - If you need to install any packages, you can use `!pip install <package-name>` directly in a cell.

8. **Restarting the Kernel**:
   - If you encounter issues, you can restart the kernel by going to "Runtime" > "Restart runtime".

HOW TO OVERCOME THE CHALLENGES OF LOADING AND PREPROCESSING THE EARTHQUAKE PREDICTION:

Overcome the challenges of loading and preprocessing data for an earthquake prediction model in Python, you can follow these steps:

1. “Data Collection and Quality”:

Use reliable sources like USGS for earthquake data. Implement data validation and cleaning routines to handle missing or erroneous data.

2. “Data Volume and Format”:

Use efficient data storage formats like HDF5 or Parquet for large datasets. Utilize libraries like Pandas for data manipulation and conversion between formats.

3. “Feature Engineering”:

Collaborate with domain experts to select and engineer relevant features. Explore geospatial libraries like Geo Pandas for working with location-based data.

4. “Geospatial Data”:

Learn geospatial data manipulation techniques using Geo Pandas and other geospatial libraries. Understand coordinate reference systems (CRS) and perform necessary transformations.

5. “Time Series Data”:

Use libraries like Pandas for time series manipulation. Consider incorporating time-based features like seasonality and trends.

6. “Imbalanced Data”:

Apply techniques such as oversampling, under sampling, or Synthetic Minority Over-sampling Technique (SMOTE) to handle imbalanced data.

7. “Normalization and Scaling”:

Normalize and scale features using libraries like Scikit-Learn. Be cautious with scaling geospatial data, as simple scaling may distort distances.

SOME COMMON DATA PREPROCESSING:

Common data preprocessing tasks for building an earthquake prediction model using Python include:

1. “Data Loading”:

Import earthquake data from various sources like CSV, JSON, or databases. Use libraries like Pandas to read and organize the data.

2. “Data Cleaning”:

Handle missing values by imputing them or removing incomplete records. Detect and correct data errors or outliers that could affect model training.

3. “Feature Selection”:

Identify and select relevant features for earthquake prediction. Consider factors like geographical coordinates, depth, and magnitude.

4. “Feature Engineering”:

Create new features or transform existing ones to better represent the underlying patterns. For geospatial data, calculate distances, spatial relationships, and density metrics.

5. “Data Transformation”:

Normalize or scale features, especially if they have different scales or units. Use techniques like Min-Max scaling or Standardization.

DATA SET:

title	magnitude	date_time	cdi	mmi	alert	tsunami	sig	net	nst	dmin	Gap	magType	depth	latitude	longitude	location	continent	country
M 7.0 - 18 km SW of Malango, Solomon Islands	7	22-11-2022 02:03	8	7	green	1	768	us	117	0.509	17	mww	14	-9.7963	159.596	Malango, Solomon Islands	Oceania	Solomon Islands
M 6.9 - 204 km	6.9	18-11-2022	4	4	green	0	735	us	99	2.229	34	mww	25	-4.955	100.738	Bengkulu, Indonesia		

SW of Bengkulu, Indonesia		13:37													9			
M 7.0 -	7	#######	3	3	green	1	755	us	147	3.125	18	mww	579	-20.0508	-178.346		Oceania	Fiji
M 7.3 - 205 km ESE of Neiafu, Tonga	7.3	#######	5	5	green	1	833	us	149	1.865	21	mww	37	-19.2918	-172.129	Neiafu, Tonga		
M 6.6 -	6.6	#######	0	2	green	1	670	us	131	4.998	27	mww	624.464	-25.5948	178.278			
M 7.0 - south of the Fiji Islands	7	#######	4	3	green	1	755	us	142	4.578	26	mwb	660	-26.0442	178.381	the Fiji Islands		
M 6.8 - south of the Fiji Islands	6.8	#######	1	3	green	1	711	us	136	4.678	22	mww	630.379	-25.9678	178.363	the Fiji Islands		
M 6.7 - 60 km SSW of Boca Chica, Panama	6.7	20-10-2022 11:57	7	6	green	1	797	us	145	1.151	37	mww	20	7.6712	-82.3396	Boca Chica, Panama		Panama
M 6.8	6.8	22-09-	8	7	yello	1	11	us	17	2.13	92	mww	20	18.33	-102	Aguililla	North	Mexico

- 55 km SSW of Aguilil la, Mexic o	2022 06: 16	w	79	5	7	.913	, Mexico	America

Earthquake Prediction:

It is well known that if a disaster has happened in a region, it is likely to happen there again. Some regions really have frequent earthquakes, but this is just a comparative quantity compared to other regions. So, predicting the earthquake with Date and Time, Latitude and Longitude from previous data is not a trend which follows like other things, it is natural occurring.

Import the necessary libraries required for building the model and data analysis of the earthquakes.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


import os
print(os.listdir("../input"))
```
```
['database.csv']
```
Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.

In [2]:
```
data = pd.read_csv("../input/database.csv")
data.head()
```
Out[2]:

	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
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01/09/1965	13:32:50	11.93	126.427	Earthquak	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM86089	ISCGE	ISCGE	ISCGE	Automati

	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
			8		e												0	M	M	M	c

In [3]:
```
data.columns
```
Out[3]:
```
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')
```

Figure out the main features from earthquake data and create a object of that features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.

In [4]:
```
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()
```
Out[4]:

	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

	Date	Time	Latitude	Longitude	Depth	Magnitude
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

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

In [5]:

```
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')
        timestamp.append(time.mktime(ts.timetuple()))
    except ValueError:
        # print('ValueError')
        timestamp.append('ValueError')
```

In [6]:

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

In [7]:

```
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

Out[7]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08

	Latitude	Longitude	Depth	Magnitude	Timestamp
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08
4	11.938	126.427	15.0	5.8	-1.57026e+08

Visualization:

Here, all the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

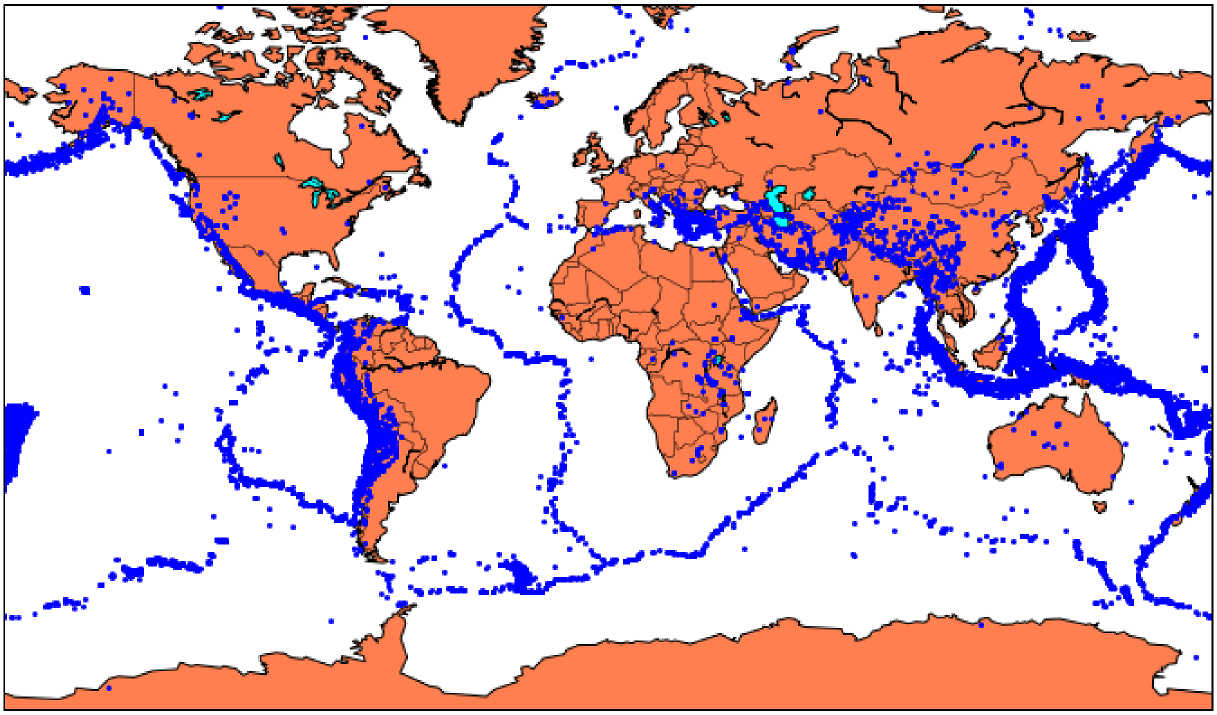
In [8]:
```
from mpl_toolkits.basemap import Basemap


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)
```

In [9]:
```
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()
```

```
/opt/conda/lib/python3.6/site-
packages/mpl_toolkits/basemap/__init__.py:1704:
MatplotlibDeprecationWarning: The axesPatch function was deprecated in
version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/opt/conda/lib/python3.6/site-
packages/mpl_toolkits/basemap/__init__.py:1707:
```

```
MatplotlibDeprecationWarning: The axesPatch function was deprecated in
version 2.1. Use Axes.patch instead.
  if limb is not ax.axesPatch:
```

All affected areas



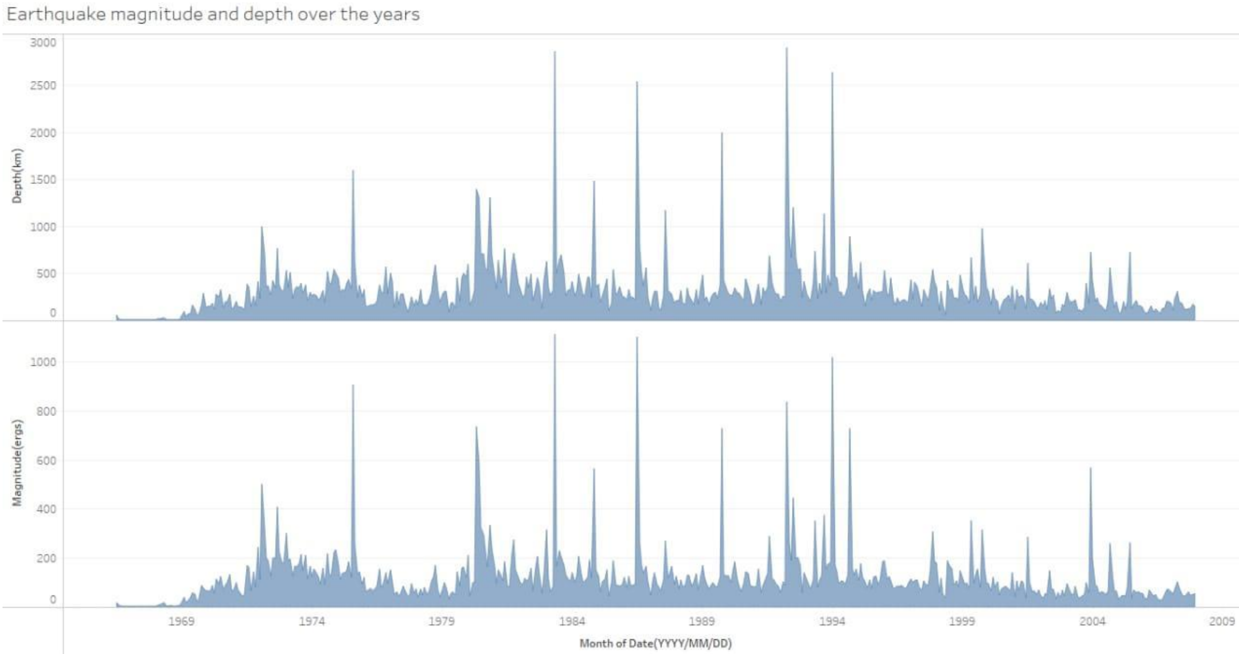
Splitting the Data

Firstly, split the data into X's and Y's which are input to the model and output of the model respectively. Here, inputs are Time stamp, Latitude and Longitude and outputs are Magnitude and Depth. Earthquake prediction: You can use this dataset to build a model that predicts when and where an earthquake might occur based on past earthquake data. You could use techniques such as time series analysis, clustering, or classification to identify patterns in the data and make predictions. Magnitude prediction: You can use this dataset to build a model that predicts the magnitude of an earthquake based on other factors such as location, depth, or the number of seismic stations that recorded the earthquake. You could use regression techniques to build this model Risk assessment: You can use this dataset to identify areas that are at higher risk of earthquakes based on historical earthquake data. You could use clustering or classification techniques to identify patterns in the data and identify areas with similar characteristics.

Anomaly detection: You can use this dataset to detect anomalies or outliers in the data, which could represent earthquakes that are unusual or unexpected. You could use techniques such as clustering or classification to identify patterns in the data and detect anomalies.

Data visualization: You can use this dataset to create visualizations of earthquake data, which could help you identify patterns and relationships in the data. You could use techniques such as scatter plots, heat maps, or geographic information system (GIS)visualize the data.th.

Split the X's and y's into train and test with validation. Training dataset contains 80% and Test dataset contain20%.



The plots of sum of Depth(km) and sum of Magnitude(ergs) for Date(YYYY/MM/DD) Month.

In [10]:
```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
```

In [11]:
```
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)
```

```
(18727, 3) (4682, 3) (18727, 2) (4682, 3)
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

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

In [12]:
```
from sklearn.ensemble import RandomForestRegressor


reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

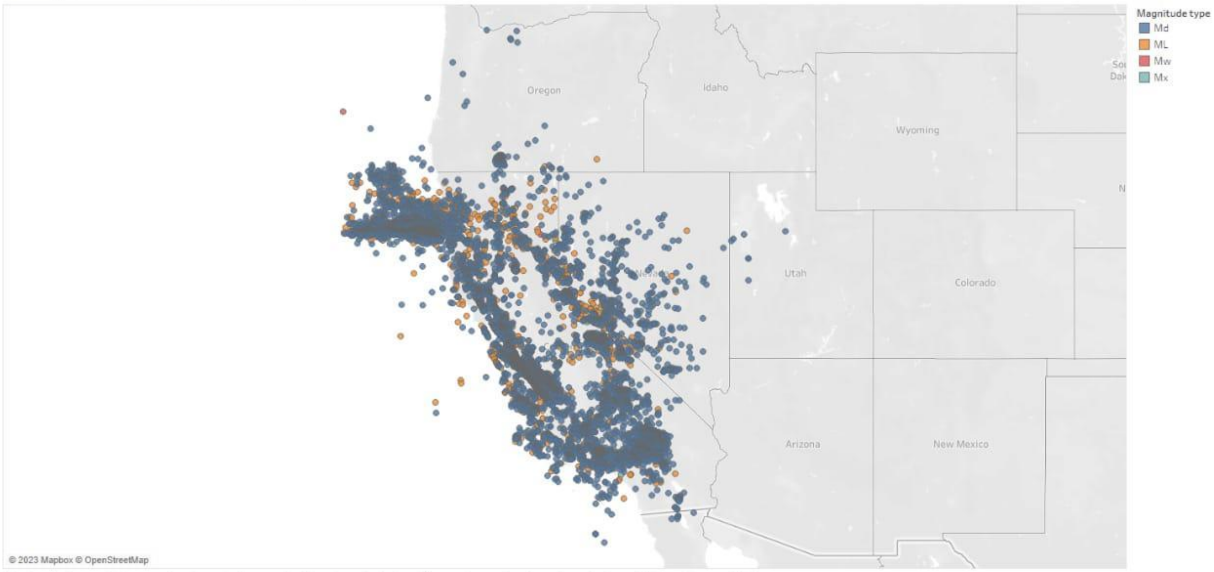
```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning:
```

```
numpy.core.umath_tests is an internal NumPy module and should not be
imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
Out[12]:
array([[  5.96,  50.97],
       [  5.88,  37.8 ],
       [  5.97,  37.6 ],
       ...,
       [  6.42,  19.9 ],
       [  5.73, 591.55],
       [  5.68,  33.61]])
In [13]:
reg.score(X_test, y_test)
Out[13]:
0.8614799631765803
In [14]:
from sklearn.model_selection import GridSearchCV


parameters = {'n_estimators':[10, 20, 50, 100, 200, 500]}


grid_obj = GridSearchCV(reg, parameters)
grid_fit = grid_obj.fit(X_train, y_train)
best_fit = grid_fit.best_estimator_
best_fit.predict(X_test)
Out[14]:
array([[  5.8888 ,  43.532  ],
       [  5.8232 ,  31.71656],
       [  6.0034 ,  39.3312 ],
       ...,
       [  6.3066 ,  23.9292 ],
       [  5.9138 , 592.151  ],
       [  5.7866 ,  38.9384 ]])
In [15]:
best_fit.score(X_test, y_test)
Out[15]:
0.8749008584467053
```

Neural Network model:

In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layers with each 16, 16, 2 nodes ReLU, ReLU and soft max as activation function.

Earthquake based on its magnitude type



Map based on Longitude (generated) and Latitude (generated). Color shows details about Magnitude type. Details are shown for Latitude(deg) and Longitude(deg).

```
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
```

```
Using TensorFlow backend.
```

In this, we define the hyperparameters with two or more options to find the best fit.

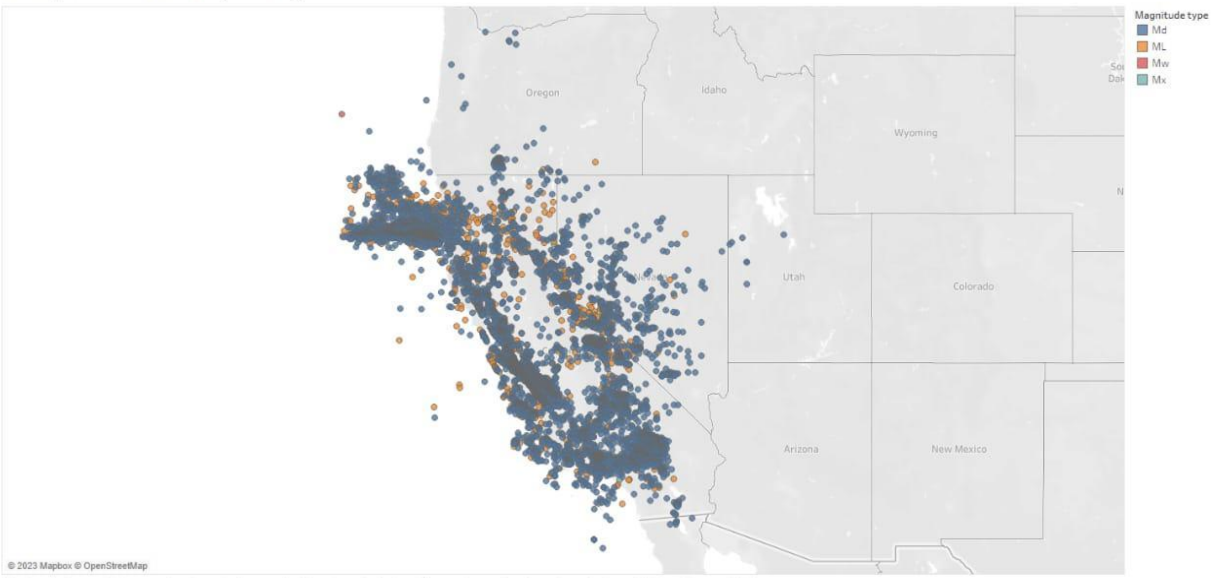
In [17]:

```
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)
```

linkcode
Here, we find the best fit of the above model and get the mean test score and standard deviation of the best fit model.

Earthquake based on its magnitude type



Map based on Longitude (generated) and Latitude (generated). Color shows details about Magnitude type. Details are shown for Latitude(deg) and Longitude(deg).

EXECUTION STEPS:

1. **Open Google Colab**: Go to [Google Colab](https://colab.research.google.com/) and log in with your Google account.

2. **Create a New Notebook**:
   - Click on "New Notebook" to create a new notebook.
   - A new tab will open with an untitled notebook.

3. **Write Code**:
   - In the cells of the notebook, you can write your Python code.
   - Click on the "+" button to add a new cell.

4. **Running Code**:
   - To run a cell, you can press `Shift+Enter` or click on the "Play" button next to the cell.
   - The output of the code will appear below the cell.

5. **Saving Work**:
   - To save your work, go to "File" and choose "Save" or press `Ctrl+S` (or `Cmd+S` on Mac).

6. **Uploading Data**:
   - If your program requires external files, you can upload them using the file icon on the left sidebar.

7. **Installing Packages**:
   - If you need to install any packages, you can use `!pip install <package-name>` directly in a cell.

8. **Restarting the Kernel**:
   - If you encounter issues, you can restart the kernel by going to "Runtime" > "Restart runtime".

CONCLUSION :

Predicting earthquakes accurately is an extremely challenging task, and it’s important to note that there is no proven method for reliably predicting earthquakes. While some research exists on statistical and machine learning models for seismic data analysis, their predictive capabilities are limited to short-term forecasts and general trends. Earthquake prediction involves complex geological and seismological factors, and it’s an active area of research but not yet fully predictable.

In Python, you can work with seismic data using libraries like ObsPy for data retrieval and processing,
and scikit-learn for building predictive models. However, any predictions made should be approached
with caution, understanding the limitations and uncertainties involved in earthquake prediction. It’s
crucial to rely on established seismic monitoring systems and follow the guidance of relevant authorities
for any necessary preparedness and response measures.