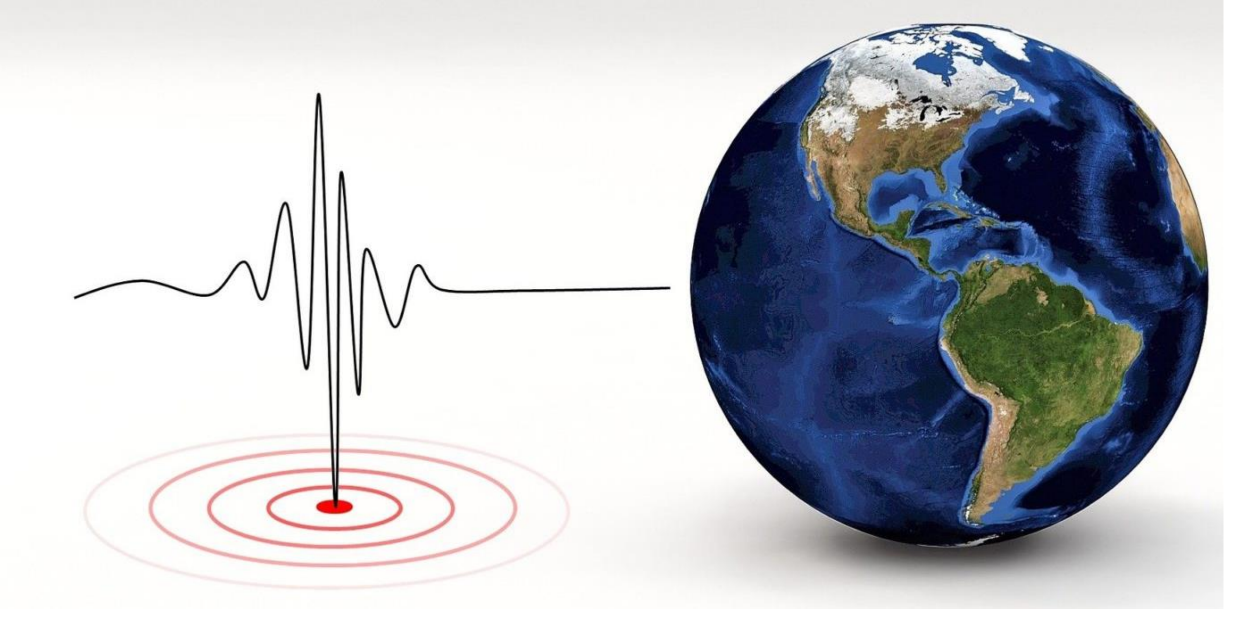
EARTHQUAKE PREDICTION MODEL USING PYTHON

Phase 5 Submission Document:

Project Title: Earthquake Prediction

Phase 5: Project Documentation & Submission

Context: In this part you will document your project and prepare it for submission.



Introduction to Earthquake Prediction

Earthquakes are natural disasters that have been a source of fascination and dread throughout human history. These sudden and often catastrophic events are the result of the Earth's tectonic plates shifting, releasing vast amounts of energy in the form of seismic waves. The ability to predict earthquakes with precision has been a long-standing scientific challenge, but researchers have made significant progress in understanding the factors that contribute to seismic activity.

The Earthquake Phenomenon:

Earthquakes occur when stress that builds up along geological faults or plate boundaries is suddenly released. This release of energy propagates as seismic waves, causing the ground to shake. The energy released in an earthquake can vary from relatively mild tremors to devastating quakes capable of causing widespread destruction and loss of life.

The Challenge of Earthquake Prediction:

The challenge of earthquake prediction lies in the inherent complexity and unpredictability of the Earth's geological processes. Earth's crust is divided into numerous tectonic plates, and their interactions, such as subduction, collision, and lateral movement, are the primary drivers of earthquakes. While scientists have a deep understanding of these processes, predicting exactly when, where, and with what intensity an earthquake will strike remains elusive.

Types of Earthquake Prediction:

1.Short-Term Prediction:

This form of prediction aims to provide warnings of imminent seismic activity, typically on the scale of days or hours. It relies on monitoring precursory signs such as ground deformation, increased seismic activity, and changes in groundwater levels. While some success has been achieved, reliable short-term prediction remains challenging.

1.Long-Term Forecasting:

Long-term forecasting focuses on estimating the probability of earthquakes occurring in specific regions over extended periods, often decades to centuries. This approach considers historical seismic data, fault characteristics, and plate tectonics. It provides a broader view of seismic hazard.

Problem Definition:

The problem is to develop an earthquake prediction model using a Kaggle Dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features .

Design Thinking Process:

1. Empathize:

1. Understand the needs and concerns of stakeholders, such as local authorities, emergency responders, and residents in earthquake-prone regions.

2. Gather data on historical earthquake occurrences, geological information, and other factors that influence earthquake prediction

2. Define:

1. Clearly define the scope of the project, including the target geographic region, time frame, and the level of accuracy or confidence required.

2. Identify the data sources and the type of model (e.g., statistical, machine learning) to be used.

3. Ideate:

1. Brainstorm potential features and data sources that could be relevant for earthquake prediction, such as fault lines, geological data, historical seismic activity, and meteorological data.

2. Consider different modeling approaches, such as logistic regression, neural networks, or time series analysis.

4. Prototype:

1. Develop a prototype in Python to test and experiment with various data sources, features, and modeling techniques.

2. Create a simple web interface or visualization to make predictions accessible to stakeholders.

5. Test:

1. Validate the model's performance using historical data and various evaluation metrics, such as accuracy, precision, and recall.

2. Collect feedback from stakeholders and adjust the model accordingly.

6. Implement:

1. Develop a production-ready version of the earthquake prediction model.

2. Set up data pipelines for real-time data ingestion and processing.

3. Ensure scalability and reliability for continuous monitoring.

7. Monitor and Improve:

1. Continuously monitor the model's performance and update it with new data.

2. Refine the model by incorporating more data sources and improving algorithms.

Phases of Development:

1. **Data Collection:**
    1. Gather historical earthquake data, geological information, and any other relevant data sources. Clean and preprocess the data.

2. **Feature Engineering:**
    1. Extract meaningful features from the data, such as fault line proximity, geological characteristics, historical seismic activity, and meteorological conditions.

3. **Model Development:**
    1. Choose an appropriate machine learning or statistical model, such as logistic regression, decision trees, or neural networks.
    2. Train the model using historical data.

4. **Evaluation:**
    1. Assess the model's performance using appropriate evaluation metrics.
    2. Adjust the model and features based on feedback and testing.

5. **Deployment:**
    1. Implement the model in a production environment.
    2. Set up real-time data ingestion and processing pipelines.
6. **Monitoring and Maintenance:**
    1. Continuously monitor the model's performance.
    2. Incorporate new data and improve the model over time.
7. **Communication:**
    1. Provide the earthquake predictions through a user-friendly interface or reporting system to stakeholders.

In earthquake prediction, the choice of dataset, data preprocessing steps, and feature exploration techniques is critical to building an effective predictive model. Here, I'll describe the typical dataset used for earthquake prediction, the essential data preprocessing steps, and some feature exploration techniques:

Dataset for Earthquake Prediction:

The dataset used for earthquake prediction typically includes various types of information. Here are some essential components:

Seismic Data:

This includes information about past seismic events, such as earthquake location (latitude and longitude), depth, magnitude, and the time of occurrence.

Geological Data:

Data related to the geological features of the region, such as fault lines, tectonic plate boundaries, and geological formations.

Environmental Data:

Data on environmental factors that may influence seismic activity, such as temperature, pressure, and rainfall.

Historical Data:

Records of past earthquakes and their characteristics are crucial for training the model.

Sensor Data:

Data from seismometers and other relevant sensors that monitor ground movement.

Data Preprocessing Steps:

Data preprocessing is a crucial step to clean and prepare the dataset for analysis. Here are some common data preprocessing steps:

Data Cleaning:

Remove missing or inconsistent data points. Ensure data consistency and correctness.

Data Transformation:

Convert data types, if necessary. For example, dates and times may need to be standardized or converted into a numerical format.

Feature Scaling:

Normalize or standardize features to ensure that they are on a consistent scale. This is particularly important when using algorithms sensitive to feature scaling, such as support vector machines or k-nearest neighbors.

Feature Engineering:

Create new features that capture relevant information. For example, you might calculate the distance between a location and nearby fault lines.

Data Splitting:

Divide the dataset into training and testing sets to assess the model's performance.

Imbalanced Data Handling: Earthquake datasets are often imbalanced, with fewer earthquake occurrences compared to non-earthquake instances.

Imbalanced Data Handling:
Earthquake datasets are often imbalanced, with fewer earthquake occurrences compared to non-earthquake instances. Techniques such as oversampling, under sampling, or synthetic data generation can address this issue.

Outlier Detection:
Identify and handle outliers that could negatively affect the model's performance.

3. Feature Exploration Techniques:

Feature exploration involves understanding the relationships and patterns in the data. Some techniques include:

Correlation Analysis:
Calculate correlations between features to identify which ones are strongly related to earthquake occurrences. High correlations can indicate significant predictive power.

Data Visualization:
Create visualizations, such as scatter plots, histograms, and heatmaps, to explore the data's distribution and relationships visually.

Dimensionality Reduction:
Use techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dimensionality of the dataset while retaining as much information as possible.

Time Series Analysis:
If you have time-related data, apply time series analysis techniques to discover temporal patterns and trends in seismic activity.

Geospatial Analysis:
Leverage geospatial analysis tools to visualize geological features, fault lines, and earthquake occurrences on maps, which can provide valuable insights.

Feature Importance Analysis:
If using machine learning models, analyze feature importances to understand which features contribute the most to the model's predictions. This can guide feature selection.

Domain Expert Collaboration:
Collaborate with domain experts, such as seismologists and geologists, to gain insights into the relevance and significance of different features. They can provide valuable domain knowledge.

Development Part 1:

- Loading
- Preprocessing

Loading DataSet:

Data loading in earthquake prediction is a crucial step in the process. It involves gathering and organizing relevant information to train and test predictive models

Downloading DataSet:

https://www.kaggle.com/datasets/usgs/earthquake-database

Importing Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns from sklearn.preprocessing
import StandardScaler from sklearn.model_selection
import train_test_split
import tensorflow as tf
```

Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically predict the earthquake datasets in CSV format, but you can adapt this code to other formats as needed.

```
data = pd.read_csv('../input/earthquake-database/database.csv')
```

```
pd.data
```

	Date	Time	Latitude	Longitude	Type
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake
...	...	...	...	...	...
23407	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake
23408	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake
23409	12/28/2016	12:38:51	36.9179	140.4262	Earthquake
23410	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake
23411	12/30/2016	20:08:28	37.3973	141.4103	Earthquake

```
pd.data.info()
```

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

Selecting Columns:

To select specific columns from the dataset, you can use the column names. For example, if you want to select the "latitude" and "longitude" columns:

```
selected_columns = earthquake_data[['latitude',
'longitude']]
```

Filtering Rows Based on Conditions:

If you want to select rows based on certain conditions, you can use boolean indexing.

```
high_magnitude_earthquakes =
earthquake_data[earthquake_data['magnitude'] > 5.0]
```

Selecting Specific Rows and Columns:

If you want to select specific rows and columns simultaneously, you can combine the two operations.

```
specific_data =
earthquake_data.loc[earthquake_data['magnitude'] > 5.0,
['latitude', 'longitude']]
```

Preprocessing Data:

Preprocessing is a crucial step in earthquake prediction as it helps clean and transform raw data into a format suitable for training machine learning models.

Handling Missing Data:

Check for missing values in your dataset and decide on a strategy to handle them. You can either remove rows with missing values, fill them using imputation techniques, or use more advanced methods depending on the nature of the missing data.

```
data= data.drop('ID', axis=1)
```

```
data.isna().sum()
```

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

```
Data= data.drop(null_columns, axis=1)
```

```
data.isna().sum()
```

```
Date                      0
Time                      0
Latitude                  0
Longitude                 0
Type                      0
Depth                     0
Magnitude                 0
Magnitude Type            3
Root Mean Square       6060
Source                    0
Location Source           0
Magnitude Source          0
Status                    0
dtype: int64
```

Feature Engineering:

Create new features that might capture important information. For example, you could calculate the distance from each earthquake to a known fault line or landmark.

```
eq_data['distance_to_fault'] = calculate_distance(eq_data['latitude'],
eq_data['longitude'])
```

```
data
```

	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	6.0	MW
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	5.8	MW
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	6.2	MW
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	5.8	MW
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	5.8	MW
...	...	...	...	...	...	...	...	...
23404	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	5.6	ML
23405	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	5.5	ML
23406	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	5.9	MWW
23407	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	6.3	MWW
23408	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	5.5	MB

```
data['Month']=data['Date'].apply(lambda x: x[0:2]) data['Year'] =
data['Date'].apply(lambda x: x[-4:])
data = data.drop('Date', axis=1)
```

```
data['Month'] = data['Month'].astype(np.int)
```

```
data[data['Year'].str.contains('Z')]
```

	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square
3378	1975-02-23T02:58:41.000Z	8.017	124.075	Earthquake	623.0	5.6	MB	1.022784
7510	1985-04-28T02:53:41.530Z	-32.998	-71.766	Earthquake	33.0	5.6	MW	1.300000
20647	2011-03-13T02:23:34.520Z	36.344	142.344	Earthquake	10.1	5.8	MWC	1.060000

```
invalid_year_indices = data[data['Year'].str.contains('Z')].index
data = data.drop(invalid_year_indices, axis=0).reset_index(drop=True)
```

```
data['Year'] = data['Year'].astype(np.int)
```

```
data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))
data = data.drop('Time', axis=1)
```

	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source
0	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM
1	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM
2	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM
3	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
4	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
...	...	...	...	...	...	...	...	...	...
23401	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN
23402	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN
23403	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US
23404	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US
23405	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US

Development Part 2:

- Visualizing the data on a world map
- Splitting it into training and testing sets

Visualizing The Data:

Data visualization is the representation of data through use of common graphics, such as charts, plots, infographics, and even animations. These visual displays of information communicate complex data relationships and data-driven insights in a way that is easy to understand.

Data Visualization in Earthquake Prediction

Visualizing data in earthquake prediction involves conveying complex information in a clear and insightful way. Here are some data visualization techniques specifically tailored for earthquake prediction:

1.Seismic Heatmaps:

1. Use a heatmap to represent the intensity of seismic activity in a geographic area.
2. Color gradients can indicate the frequency or magnitude of earthquakes.

2.Time Series with Aftershocks:

1. Create a time series plot that shows earthquake occurrences over time.
2. Highlight significant events, such as major earthquakes and aftershocks, on the timeline.

3.Magnitude vs. Depth Scatter Plots:

1. Plot earthquake magnitude against depth to identify patterns and correlations.
2. Differentiate between shallow and deep earthquakes to understand their characteristics.

4.Animated Earthquake Maps:

1. Develop animated maps that show the evolution of seismic activity over time.
2. Animations can reveal patterns and changes in earthquake distribution.

5.Fault Line Network Graphs:

1. Use network graphs to visualize relationships between fault lines, tectonic plates, and seismic events.
2. Nodes represent regions, and edges indicate connections or correlations.

6.3D Earthquake Visualization:

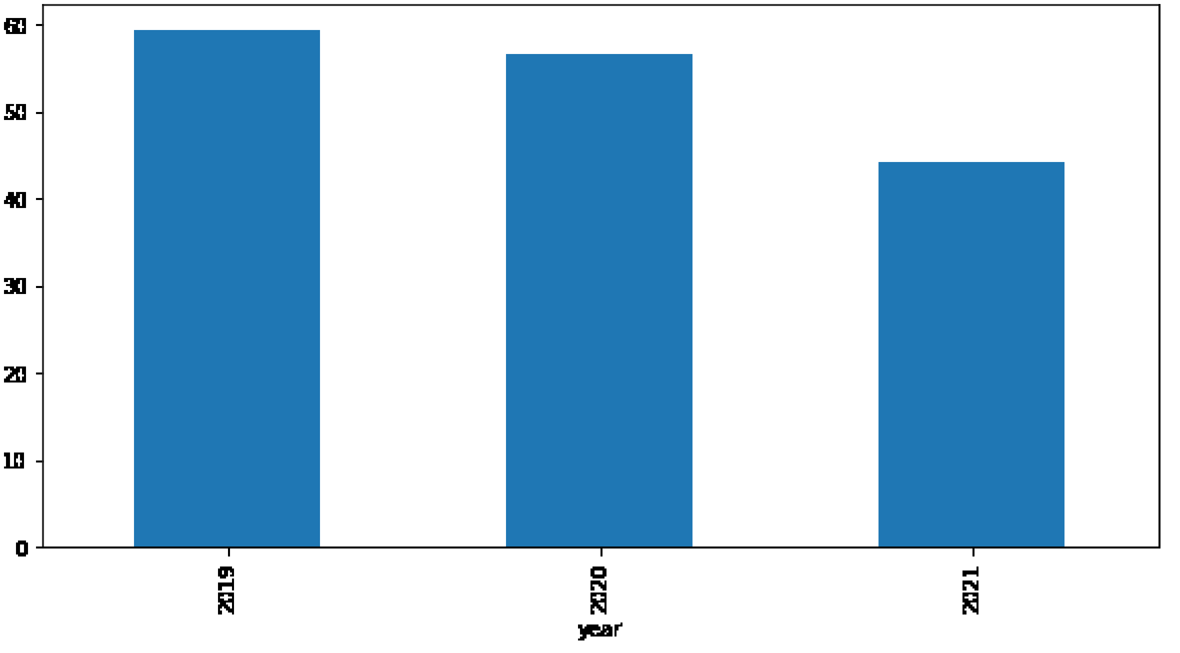
1. Represent seismic activity in a three-dimensional space, considering location, depth, and magnitude.
2. Use 3D bar charts or scatter plots for a comprehensive view.
3. visualizations to highlight relationships.

1.Real-time Earthquake Dashboard:

1. If possible, create a real-time dashboard that updates with the latest seismic data.
2. This can be valuable for monitoring ongoing seismic events.

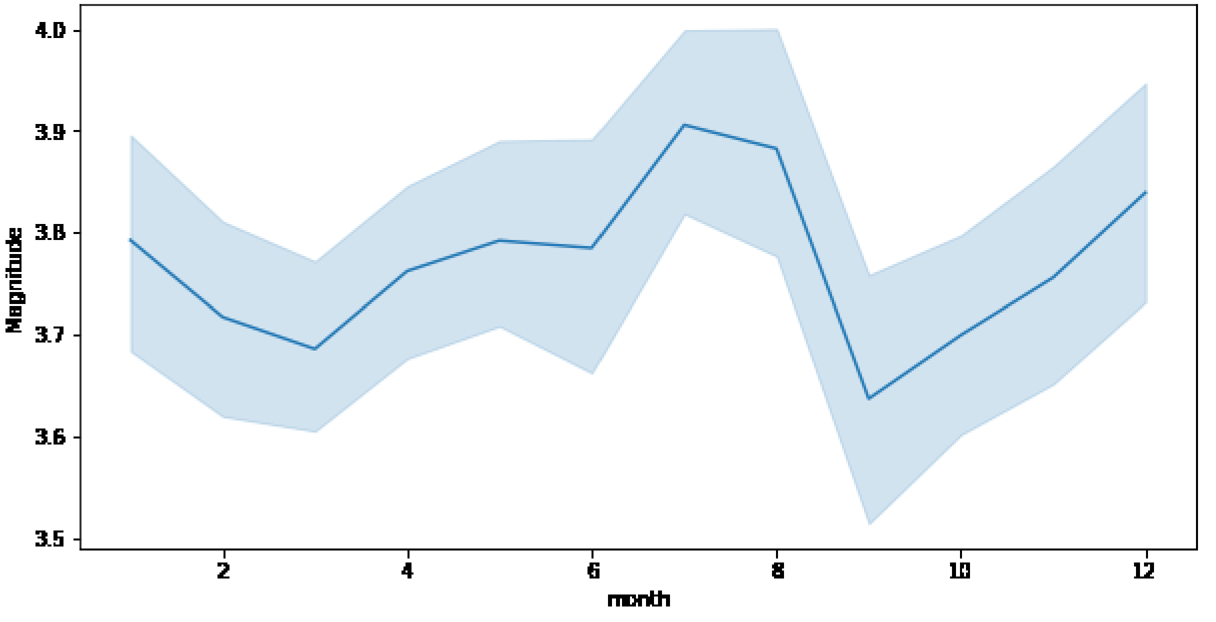
Visualizating Data:

```
plt.figure(figsize=(10, 5))
x = df.groupby('year').mean()['Depth']
x.plot.bar()
plt.show()
```



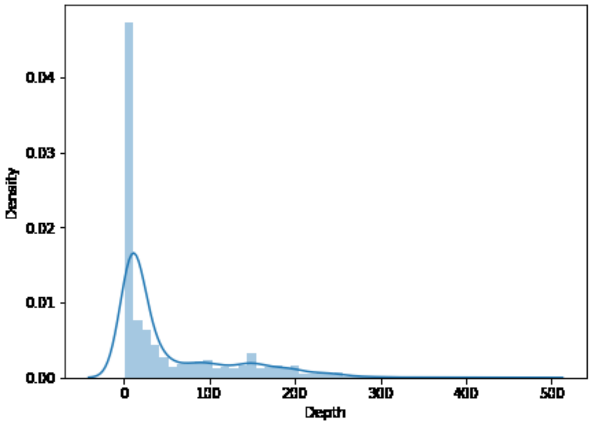
LINEPLOTS:

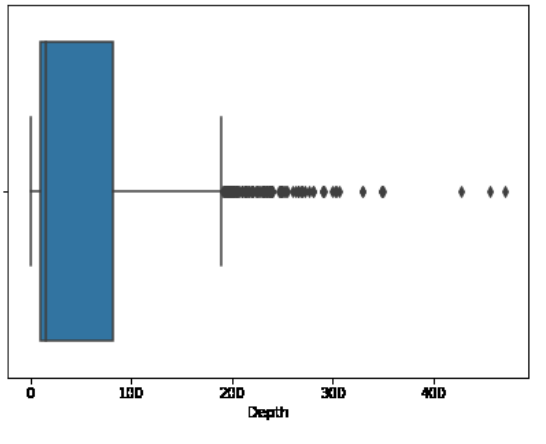
```
plt.figure(figsize=(10, 5))
sb.lineplot(data=df,
            x='month',
            y='Magnitude')
plt.show()
```



SUBPLOTS:

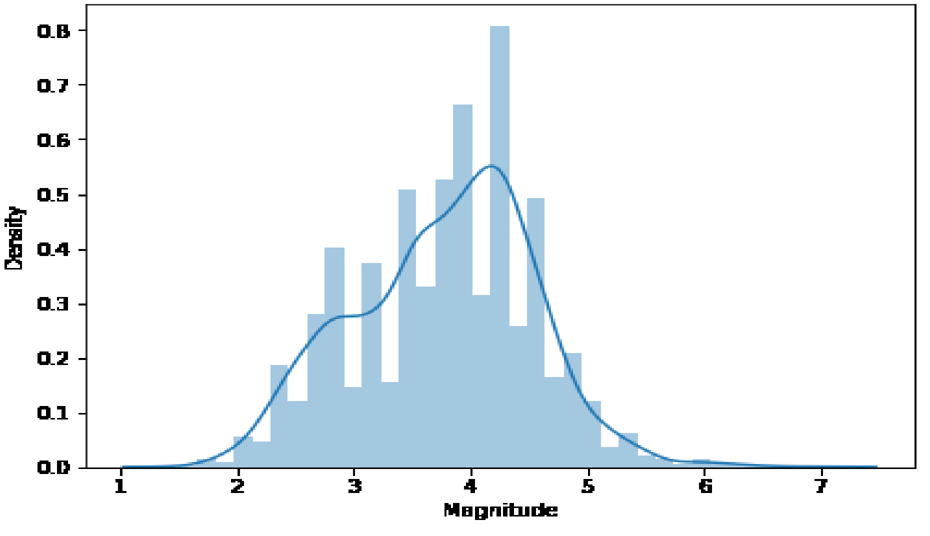
```
plt.subplots(figsize=(15, 5))

plt.subplot(1, 2, 1)
sb.distplot(df['Depth'])
plt.subplot(1, 2, 2)
sb.boxplot(df['Depth'])

plt.show()
```

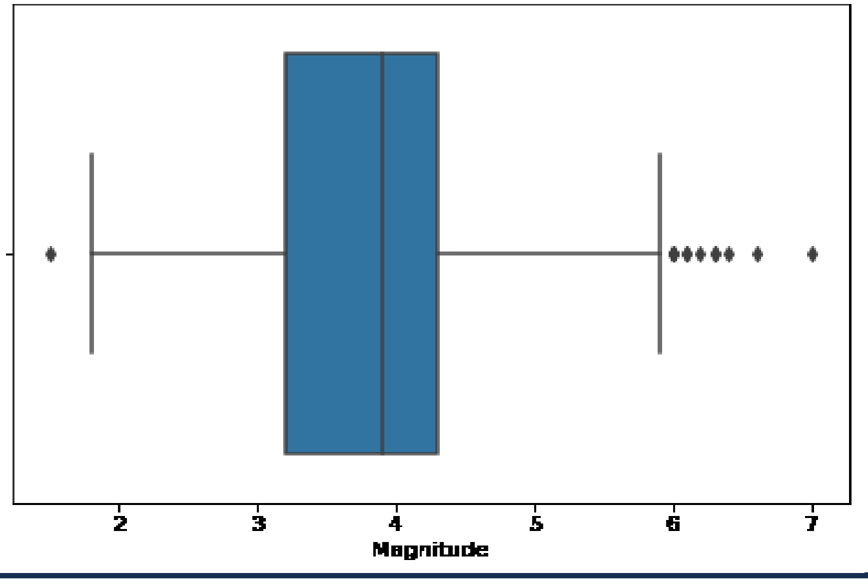




BOXPLOTS:

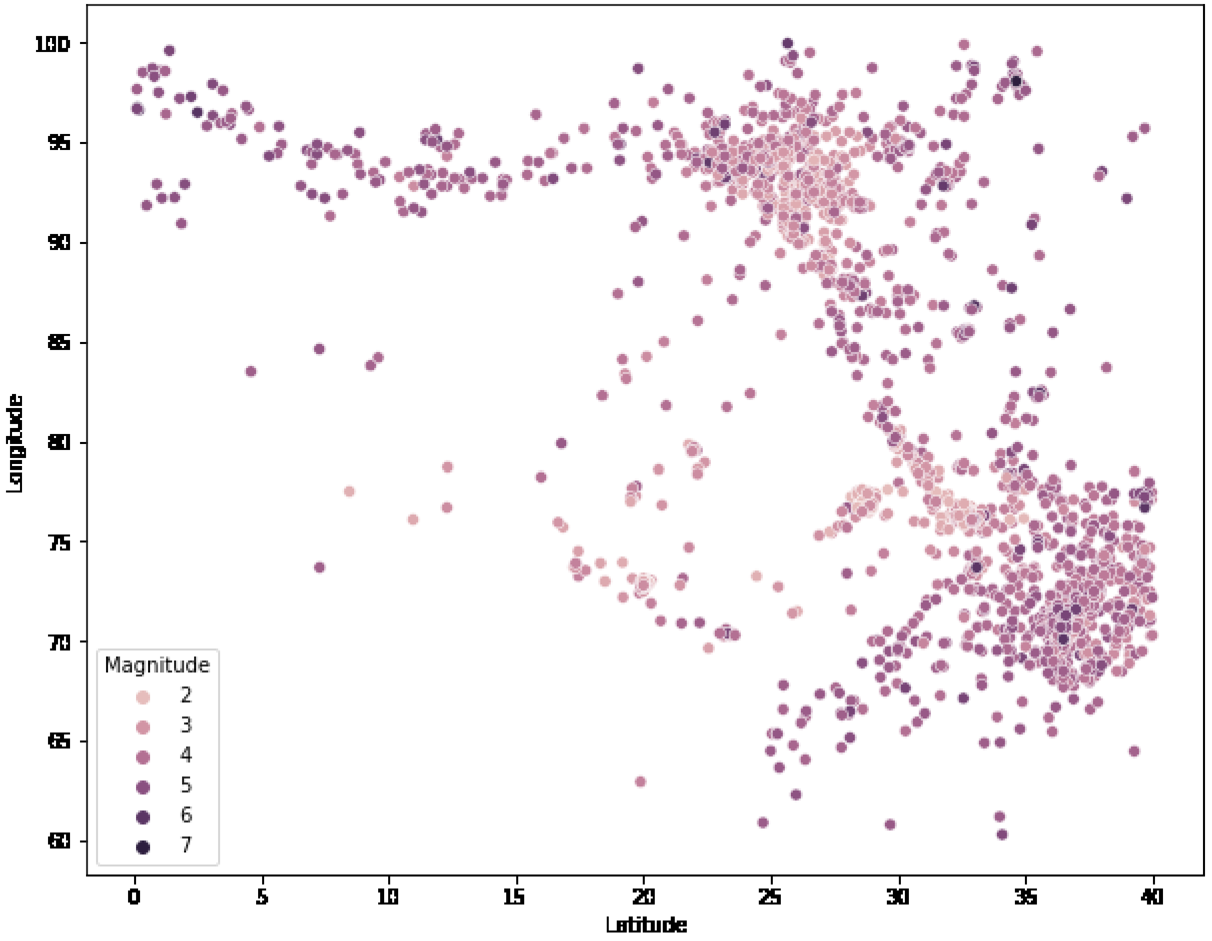
```
plt.subplots(figsize=(15, 5))

plt.subplot(1, 2, 1)
sb.distplot(df['Magnitude'])

plt.subplot(1, 2, 2)
sb.boxplot(df['Magnitude'])

plt.show()
```





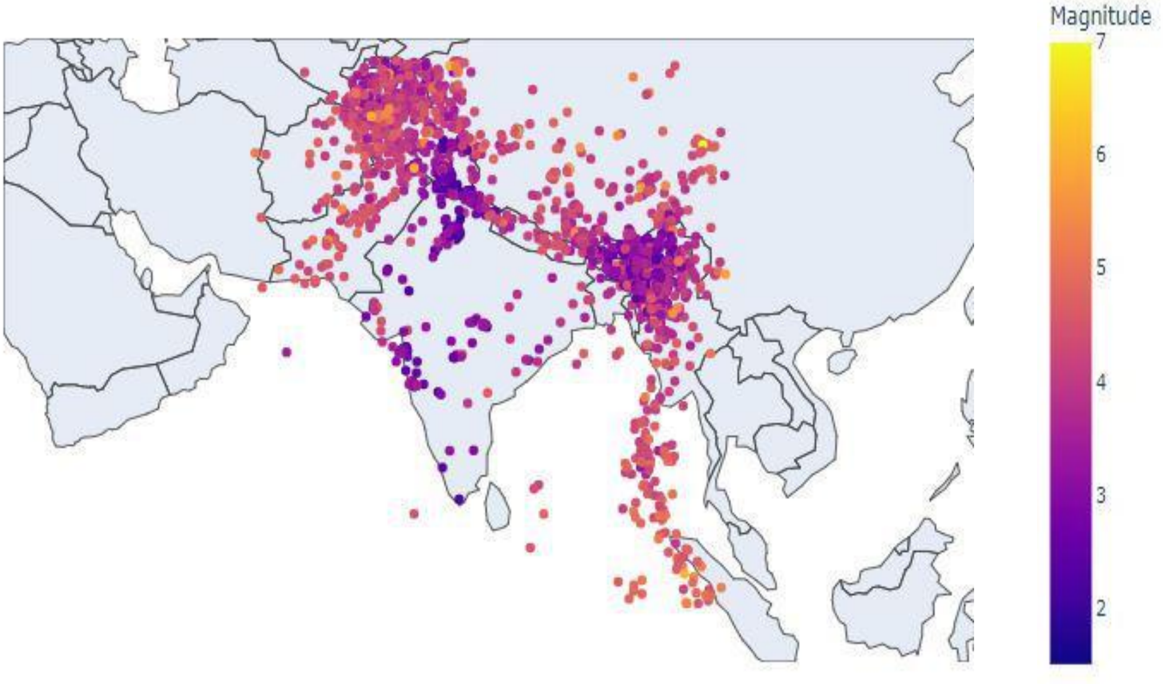
SCATTERPLOTS:

```
plt.figure(figsize=(10, 8))
sb.scatterplot(data=df,
               x='Latitude',
               y='Longitude',
               hue='Magnitude')
plt.show()
```



Now by using Plotly let's plot the latitude and the longitude data on the map to visualize which areas are more prone to earthquakes.

```
import plotly.express as px
import pandas as pd

fig = px.scatter_geo(df, lat='Latitude',
                     lon='Longitude',
                     color="Magnitude",
                     fitbounds='locations',
                     scope='asia')
fig.show()
```



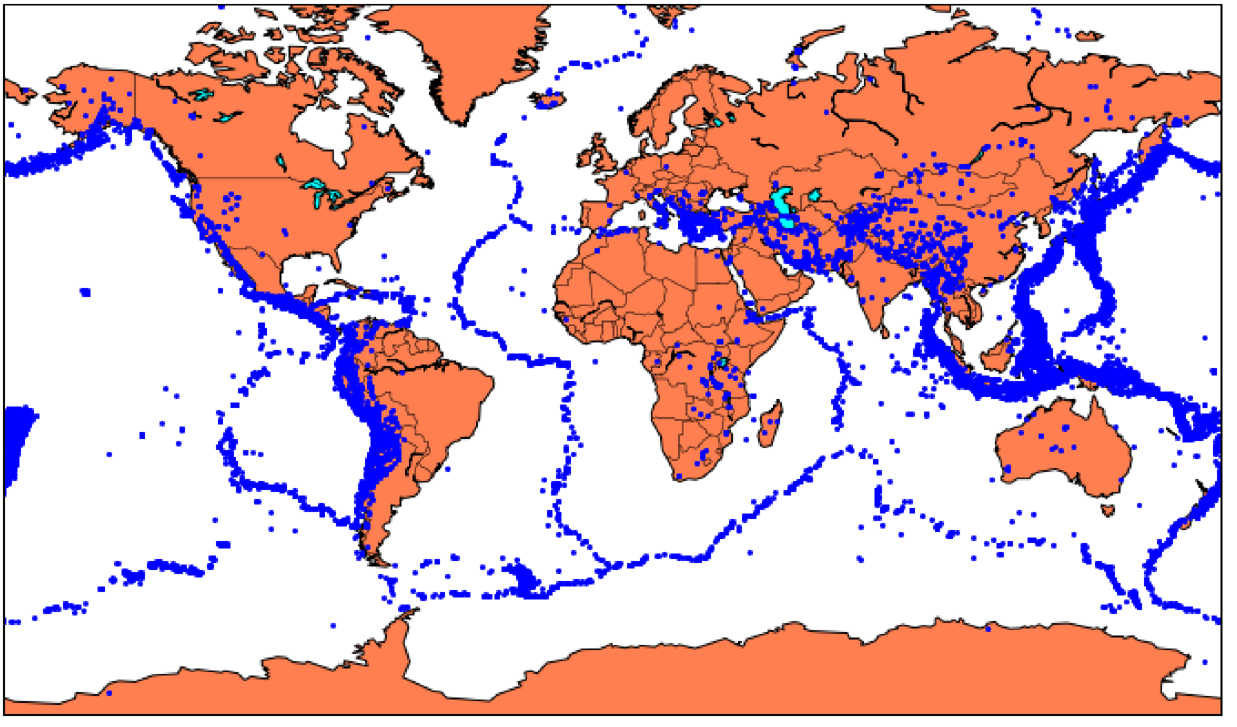
Visualization On The World Map:

```
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()
```

OUTPUT:

All affected areas



Data Spiliting:

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

```python
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
```

```python
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)
(X_train.shape, X_test.shape, y_train.shape,
X_test.shape)
```

```
(18727, 3) (4682, 3)(1237, 2) (4682,3)
```

```python
from sklearn.ensemble import RandomForestRegressor
reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

```
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]])
```

```
reg.score(X_test, y_test)
```

```
0.8614799631765803
```

```
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)
```

```
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 ]])
```

```
best_fit.score(X_test, y_test)
```

```
0.8749008584467053
```

Neural Network model

```
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.

```python
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, opti
mizer=optimizer, loss=loss)
```

```python
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))
```

```
Best: 0.666684 using {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_h
inge', 'neurons': 16, 'optimizer': 'SGD'}
0.666684 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squa
red_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss': 'squa
red_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.666684 (0.471398) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared
_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared
_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

The best fit parameters are used for same model to compute the score with training data and testing data.

```
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))
```

```
Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 4s 233us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 4s 228us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 4/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 5/20
18727/18727 [==============================] - 5s 262us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 8/20
18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 9/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 10/20
18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 11/20
18727/18727 [==============================] - 4s 221us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
```

```
Epoch 12/20
18727/18727 [==============================] - 4s 231us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 13/20
18727/18727 [==============================] - 5s 248us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 14/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 15/20
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [==============================] - 4s 225us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [==============================] - 4s 219us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 19/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [==============================] - 5s 258us/step - loss: 0.5038 - acc: 0.9182 - val_l
oss: 0.5038 - val_acc: 0.9242
```

```
Out[20]:
        <keras.callbacks.History at 0x78dfa2107ef0>
```

```
[test_loss, test_acc] = model.evaluate(X_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

```
4682/4682 [==============================] - 0s 29us/step
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

```
model.save('earthquake.h5')
```

SAMPLE PYTHON PROGRAM :

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Step 1: Load the dataset
def load_earthquake_data(file_path):
    data = pd.read_csv(file_path)  # Assuming the dataset is in CSV
format
    return data

# Step 2: Preprocess the data
def preprocess_data(data):
    # Perform any necessary preprocessing steps, such as handling
missing values, feature engineering, etc.
    # For simplicity, let's assume the dataset has features (X) and
labels (y).
```

```
    # Example: Drop rows with missing values
    data = data.dropna()

    # Example: Split the data into features (X) and labels (y)
    X = data.drop('label_column', axis=1)
    y = data['label_column']

    # Example: 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)

    # Example: Standardize features using StandardScaler
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_test_scaled, y_train, y_test

# Step 3: Train a machine learning model
def train_model(X_train, y_train):
    # Example: Use a Random Forest classifier
    model = RandomForestClassifier(n_estimators=100,
random_state=42)
    model.fit(X_train, y_train)
    return model
```

```python
# Step 4: Evaluate the model
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")

# Example usage
if __name__ == "__main__":
    file_path = "path/to/earthquake_data.csv"

    # Load data
    earthquake_data = load_earthquake_data(file_path)

    # Preprocess data
    X_train, X_test, y_train, y_test = preprocess_data(earthquake_data)

    # Train model
    trained_model = train_model(X_train, y_train)

    # Evaluate model
    evaluate_model(trained_model, X_test, y_test)
```

Creating a complete earthquake prediction program involves several steps, from loading the dataset to preprocessing and training a model. Below is a simplified Python script using popular libraries such as pandas and scikit-learn for loading and preprocessing earthquake data. Keep in mind that this is a basic example, and real-world applications might require more sophisticated techniques.

Limitations and Considerations:

It's crucial to acknowledge the limitations of earthquake prediction using machine learning models:

1.Precise earthquake prediction remains a significant scientific challenge, and machine learning models can only provide estimates or probabilities.

2.Collaboration with domain experts, such as seismologists and geologists, is essential to ensure the accuracy and relevance of the data and models used in earthquake prediction.

3.Ethical considerations must be taken into account when disseminating earthquake predictions, as false alarms can have serious consequences.

Conclusion:

In conclusion, machine learning models offer a data-driven approach to estimate the probability of earthquakes in specific regions and timeframes. While they do not provide precise predictions, they can contribute to risk assessment and preparedness efforts, ultimately enhancing our ability to mitigate the impact of seismic events. Close collaboration between data scientists and domain experts is crucial for the success of earthquake prediction projects.