EARTHQUAKE PREDICTION MODEL USING PYTHON

Earthquake prediction model in loading and pre-processing the dataset:

Introduction:

Machine learning has the ability to advance our knowledge of earthquakes and enable more accurate forecasting and catastrophe response. It's crucial to remember that developing accurate and dependable prediction models for earthquakes still needs more study as it is a complicated and difficult topic.

In order to anticipate earthquakes, machine learning may be used to examine seismic data trends. Seismometers capture seismic data, which may be used to spot changes to the earth's surface, like seismic waves brought on by earthquakes. Machine learning algorithms may utilize these patterns to forecast the risk of an earthquake happening in a certain region by studying these patterns and learning to recognize key traits that are linked to seismic activity.

About Dataset:

The National Earthquake Information Center (NEIC) determines the location and size of all significant earthquakes that occur worldwide and disseminates this information immediately to national and international agencies, scientists, critical facilities, and the general public. The NEIC compiles and provides to scientists and to the public an extensive seismic database that serves as a foundation for scientific research through the operation of modern digital national and global seismograph networks and cooperative

international agreements. The NEIC is the national data center and archive for earthquake information.

Content:

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

1	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seism	Magnitude	Magnitude T	Magnitude
2	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6			6	MW	
3	01/04/1965	11:29:49	1.863	127.352	Earthquake	80			5.8	MW	
4	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20			6.2	MW	
5	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15			5.8	MW	
6	01/09/1965	13:32:50	11.938	126.427	Earthquake	15			5.8	MW	
7	01/10/1965	13:36:32	-13.405	166.629	Earthquake	35			6.7	MW	
8	01/12/1965	13:32:25	27.357	87.867	Earthquake	20			5.9	MW	
9	01/15/1965	23:17:42	-13.309	166.212	Earthquake	35			6	MW	
10	01/16/1965	11:32:37	-56.452	-27.043	Earthquake	95			6	MW	
11	01/17/1965	10:43:17	-24.563	178.487	Earthquake	565			5.8	MW	
12	01/17/1965	20:57:41	-6.807	108.988	Earthquake	227.9			5.9	MW	
13	01/24/1965	00:11:17	-2.608	125.952	Earthquake	20			8.2	MW	
14	01/29/1965	09:35:30	54.636	161.703	Earthquake	55			5.5	MW	
15	02/01/1965	05:27:06	-18.697	-177.864	Earthquake	482.9			5.6	MW	
16	02/02/1965	15:56:51	37.523	73.251	Earthquake	15			6	MW	
17	02/04/1965	03:25	-51.84	139.741	Earthquake	10			6.1	MW	
18	02/04/1965	05:01:22	51.251	178.715	Earthquake	30.3			8.7	MW	
19	02/04/1965	06:04:59	51.639	175.055	Earthquake	30			6	MW	
20	02/04/1965	06:37:06	52.528	172.007	Earthquake	25			5.7	MW	
21	02/04/1965	06:39:32	51.626	175.746	Earthquake	25			5.8	MW	
22	02/04/1965	07:11:23	51.037	177.848	Earthquake	25			5.9	MW	
23	02/04/1965	07:14:59	51.73	173.975	Earthquake	20			5.9	MW	
24	02/04/1965	07:23:12	51.775	173.058	Earthquake	10			5.7	MW	
25	02/04/1965	07:43:43	52.611	172.588	Earthquake	24			5.7	MW	
26	02/04/1965	08:06:17	51.831	174.368	Earthquake	31.8			5.7	MW	
27	02/04/1965	08:33:41	51.948	173.969	Earthquake	20			5.6	MW	
28	02/04/1965	08:40:44	51.443	179.605	Earthquake	30			7.3	MW	
29	02/04/1965	12:06:08	52.773	171.974	Earthquake	30			6.5	MW	
30	02/04/1965	12:50:59	51.772	174.696	Earthquake	20			5.6	MW	

Necessary step to follow:

1.Import Libraries:

Start by importing the necessary libraries:

Program:

Import pandas as pd

Import numpy as np

From sklearn.model_selection import train_test_split

From sklearn.preprocessing import StandardScaler

2.Load the Dataset:

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

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

Formats as needed.

Program:

Df = pd.read_csv(‘ E:\USA_Housing.csv ‘)

Pd.read()

3. Exploratory Data Analysis (EDA):

Perform EDA to understand your data better. This includes

Checking for missing values, exploring the data’s statistics, and

Visualizing it to identify patterns.

Program:

Check for missing values

Print(df.isnull().sum())

Explore statistics

Print(df.describe())

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

4. Feature Engineering:

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

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

Program:

```
# Example: One-hot encoding for categorical variables

Df = pd.get_dummies(df, columns=[‘ Avg. Area Income ‘, ‘ Avg. Area

House Age ‘])
```

5. Split the Data:

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

Your model’s performance later.

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

Y = df[‘price’] # Target variable

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

Random_state=42)
```

6. Feature Scaling:

Apply feature scaling to normalize your data, ensuring that all

Features have similar scales. Standardization (scaling to mean=0 and

Std=1) is a common choice.

Program:

```
Scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

<u>Importance of loading and processing datase:</u>

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

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

2. **Understanding Your Data:**

Before diving into analysis or model building, you need a clear understanding of your data. Loading it allows you to explore its structure, identify patterns, and gain insights into its characteristics.

3. **Data Cleaning:**

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

4. **Feature Engineering:**

Loading data is the first step in feature engineering, where you transform and enhance your variables to improve model performance. This can involve creating new features, scaling, or encoding categorical variables.

5. **Compatibility:**

Different models or analysis tools may require data in specific formats. Loading data allows you to transform it into a compatible structure, ensuring a smooth workflow.

6. **Efficiency:**

Processing data efficiently can have a significant impact on the speed and performance of your analysis or model. This includes tasks such as indexing, sorting, and aggregating data to streamline subsequent operations.

7. **Data Exploration:**

Loading data allows you to visualize and explore it through various statistical and graphical methods. This exploration is crucial for formulating hypotheses, identifying trends, and making informed decisions about further analysis.

8. **Model Training:**

If you're building a machine learning model, loading and processing data is a prerequisite for training. The model's performance heavily depends on the quality and characteristics of the training data.

9. **Iterative Process:**

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

10. **Reproducibility:**

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

Challenges involved in loading and preprocessing a earthquake prediction datasets:

Data Volume and Size:

Earthquake datasets can be massive, especially if they cover long time periods and include detailed information. Managing and loading large datasets can strain computational resources and require efficient storage solutions.

Data Variety:

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

Data Quality:

Earthquake datasets may have missing or inaccurate values, outliers, or inconsistencies. Cleaning and validating the data is crucial to ensure the accuracy of predictions. Incomplete or incorrect information can significantly impact the performance of prediction models.

Temporal and Spatial Dependencies:

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

Imbalanced Classes:

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

Feature Engineering:

Extracting meaningful features from seismic data requires domain expertise. Transforming raw sensor readings into relevant features, such as frequency components, amplitude, and spectral characteristics, is a crucial preprocessing step for earthquake prediction.

Normalization and Scaling:

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

Handling Time Series Data:

Earthquake data often involves time series information. Dealing with time-dependent patterns, seasonality, and trends requires specialized preprocessing techniques such as time-series decomposition, lag features, or rolling statistics.

Computational Intensity:

Earthquake prediction models, especially those based on machine learning algorithms, can be computationally intensive. Preprocessing steps need to be optimized for efficiency, and consideration should be given to parallel processing or distributed computing for large datasets.

Domain-Specific Challenges:

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

LOADING THE DATASET:

Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.

The specific steps involved in loading the dataset will vary depending On the machine learning library or framework that is being used.However, there are some general steps that are common to most Machine learning frameworks.

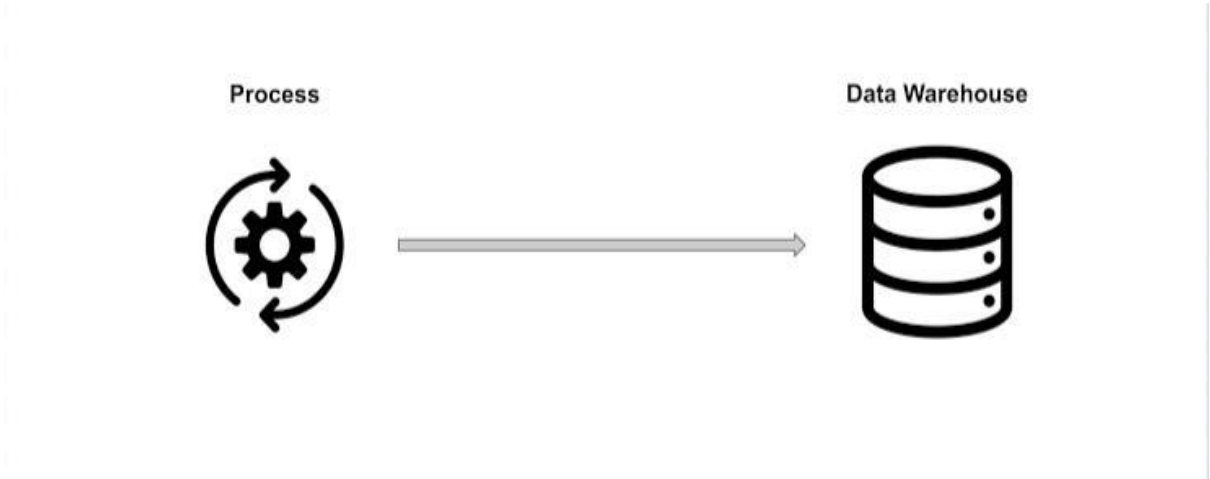
Identify the dataset:

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

Load the dataset:

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

Preprocess the dataset:



Once the dataset is loaded into the machine learning environment, You may need to preprocess it before you can start

training and Evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and Test sets.

Input data

Pandas:

This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.

Matplotlib/Seaborn:

This library is used to draw visualizations.

```
Import pandas as pd
Import matplotlib.pyplot as plt
Import seaborn as sb
 Import warnings
Warnings.filterwarnings('ignore')

Df = pd.read_csv('dataset.csv')
Df.head()
```

Data exploration:

Data set

Output:

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

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

1	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seism	Magnitude	Magnitude T	Magnitude
2	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6			6	MW	
3	01/04/1965	11:29:49	1.863	127.352	Earthquake	80			5.8	MW	
4	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20			6.2	MW	
5	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15			5.8	MW	
6	01/09/1965	13:32:50	11.938	126.427	Earthquake	15			5.8	MW	
7	01/10/1965	13:36:32	-13.405	166.629	Earthquake	35			6.7	MW	
8	01/12/1965	13:32:25	27.357	87.867	Earthquake	20			5.9	MW	
9	01/15/1965	23:17:42	-13.309	166.212	Earthquake	35			6	MW	
10	01/16/1965	11:32:37	-56.452	-27.043	Earthquake	95			6	MW	
11	01/17/1965	10:43:17	-24.563	178.487	Earthquake	565			5.8	MW	
12	01/17/1965	20:57:41	-6.807	108.988	Earthquake	227.9			5.9	MW	
13	01/24/1965	00:11:17	-2.608	125.952	Earthquake	20			8.2	MW	
14	01/29/1965	09:35:30	54.636	161.703	Earthquake	55			5.5	MW	
15	02/01/1965	05:27:06	-18.697	-177.864	Earthquake	482.9			5.6	MW	
16	02/02/1965	15:56:51	37.523	73.251	Earthquake	15			6	MW	
17	02/04/1965	03:25	-51.84	139.741	Earthquake	10			6.1	MW	
18	02/04/1965	05:01:22	51.251	178.715	Earthquake	30.3			8.7	MW	
19	02/04/1965	06:04:59	51.639	175.055	Earthquake	30			6	MW	
20	02/04/1965	06:37:06	52.528	172.007	Earthquake	25			5.7	MW	
21	02/04/1965	06:39:32	51.626	175.746	Earthquake	25			5.8	MW	
22	02/04/1965	07:11:23	51.037	177.848	Earthquake	25			5.9	MW	
23	02/04/1965	07:14:59	51.73	173.975	Earthquake	20			5.9	MW	
24	02/04/1965	07:23:12	51.775	173.058	Earthquake	10			5.7	MW	
25	02/04/1965	07:43:43	52.611	172.588	Earthquake	24			5.7	MW	
26	02/04/1965	08:06:17	51.831	174.368	Earthquake	31.8			5.7	MW	
27	02/04/1965	08:33:41	51.948	173.969	Earthquake	20			5.6	MW	
28	02/04/1965	08:40:44	51.443	179.605	Earthquake	30			7.3	MW	
29	02/04/1965	12:06:08	52.773	171.974	Earthquake	30			6.5	MW	
30	02/04/1965	12:50:59	51.772	174.696	Earthquake	20			5.6	MW	

PREPROCESSING DATASETS

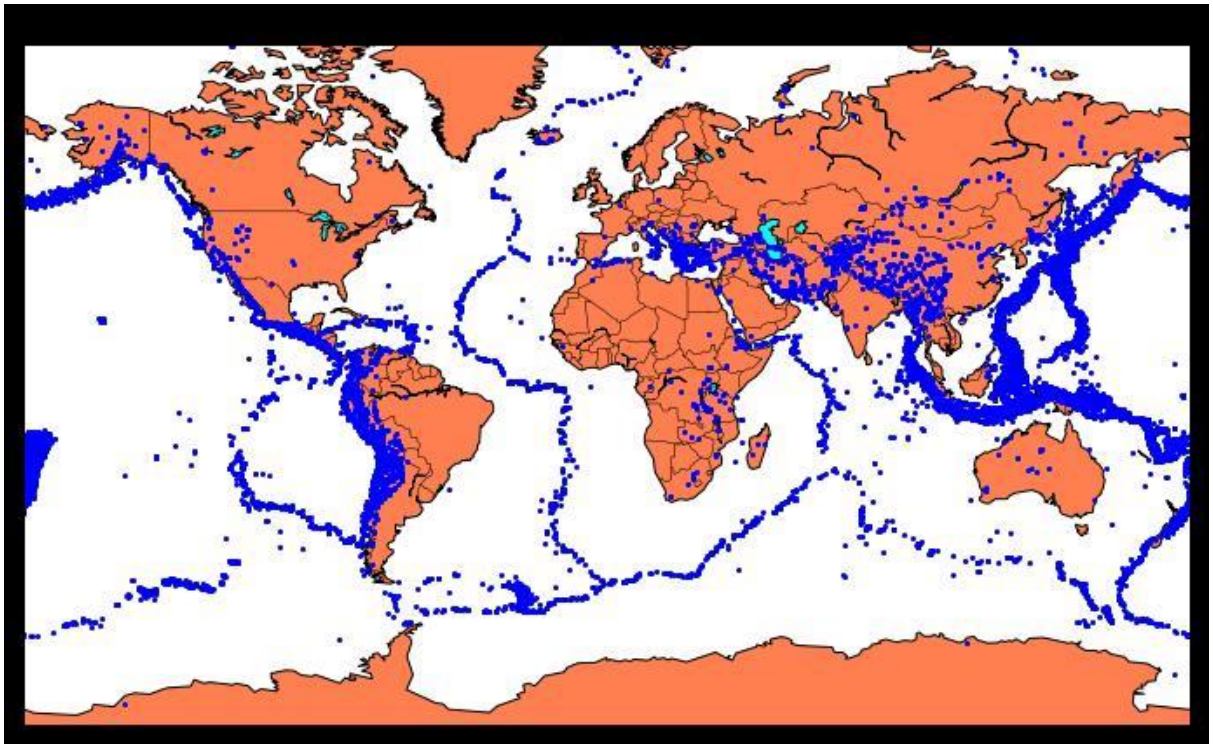
Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.

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

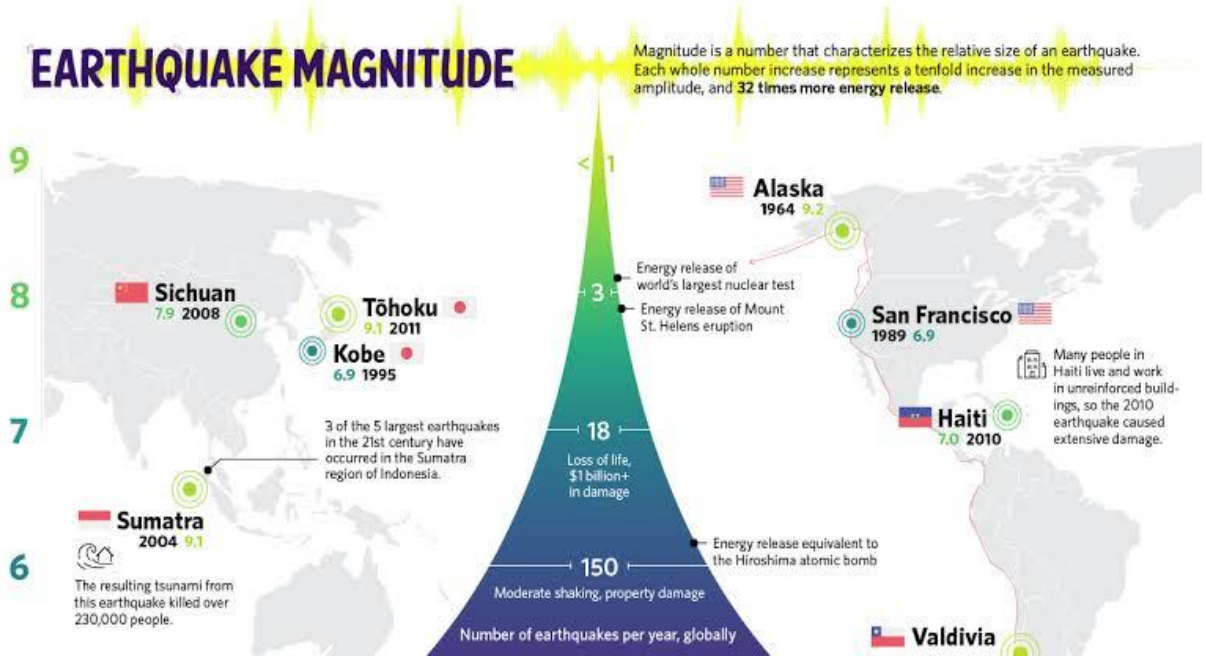
Visualization:

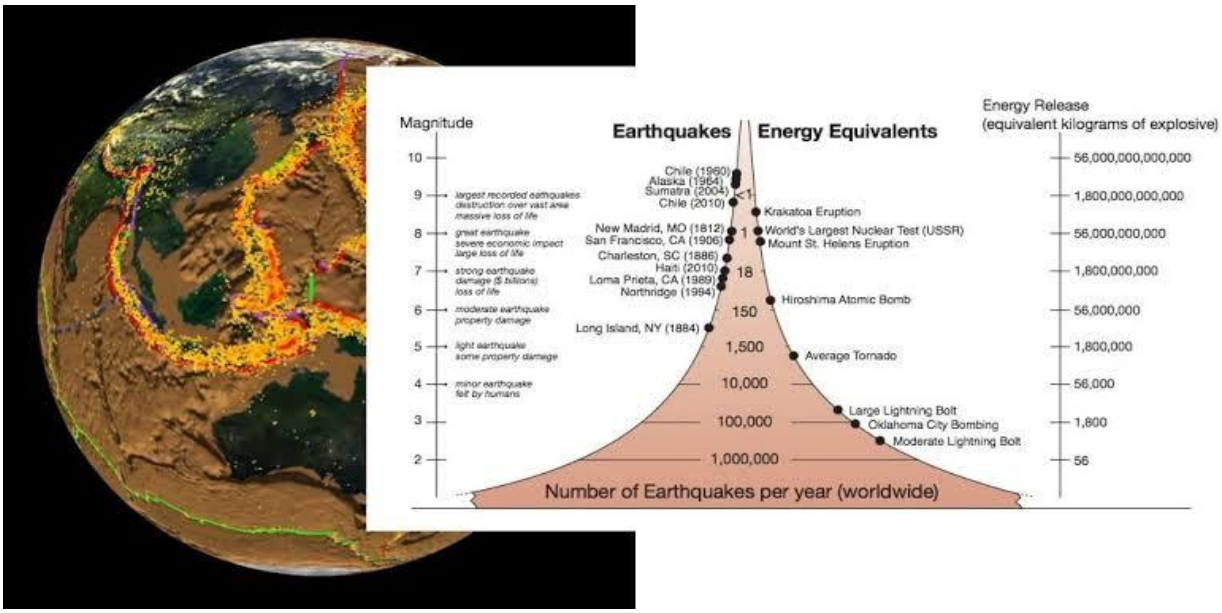
```
From mpl_toolkits.basemap import Basemap

M = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,resolution='c')

Longitudes = data["Longitude"].tolist()
Latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
        #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
X,y = m(longitudes,latitudes)

Fig = plt.figure(figsize=(12,10))
Plt.title("All affected areas")
m.plot(x, y, "o", markersize = 2, color = 'blue')
m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
```

```
m.drawcountries()
plt.show()
```



We have located on the world map where earthquakes happened in the last few years.





Splitting The Dataset:

X = final_data[[‘Timestamp’, ‘Latitude’, ‘Longitude’]]

Y = final_data[[‘Magnitude’, ‘Depth’]]

```
From sklearn.cross_validation import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

Output:

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

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

```
Reg = RandomForestRegressor(random_state=42)

Reg.fit(X_train, y_train)

Reg.predict(X_test)

Reg.score(X_test, y_test)
```

```
0.86147996317658O3
```

Neural Network Model:

A neural network model can be employed to forecast earthquakes by examining diverse elements and trends in seismic data. This model harnesses the capabilities of neural networks, which draw inspiration from the neural connections of the human brain, to analyze intricate data and reveal hidden relationships and patterns. By training the neural network on historical earthquake data, it can acquire the ability to identify precursor signals and patterns that indicate the probability of an upcoming earthquake.

```
From keras.models import Sequential

From keras.layers import Dense


Def create_model(neurons, activation, optimizer, loss):

   Model = Sequential()

   Model.add(Dense(neurons, activation=activation, input_shape=(3,)))

   Model.add(Dense(neurons, activation=activation))

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


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


   Return model


From keras.wrappers.scikit_learn import KerasClassifier
```

```
Model = KerasClassifier(build_fn=create_model, verbose=0)

# neurons = [16, 64, 128, 256]
Neurons = [16]
# batch_size = [10, 20, 50, 100]
Batch_size = [10]
Epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'exponential']
Activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
Optimizer = ['SGD', 'Adadelta']
Loss = ['squared_hinge']

Param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, activation=activation, optimizer=optimizer, loss=loss)

Grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
Grid_result = grid.fit(X_train, y_train)

Print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

Means = grid_result.cv_results_[‘mean_test_score’]

Stds = grid_result.cv_results_[‘std_test_score’]

Params = grid_result.cv_results_[‘params’]

For mean, stdev, param in zip(means, stds, params):

Print(“%f (%f) with: %r” % (mean, stdev, param))

Output:

```
Best: 0.957655 using {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squared_
hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.333316 (0.471398) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss':
'squared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.000000 (0.000000) with: {'activation': 'sigmoid', 'batch_size': 10, 'epochs': 10, 'loss':
'squared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
0.957655 (0.029957) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squ
ared_hinge', 'neurons': 16, 'optimizer': 'SGD'}
0.645111 (0.456960) with: {'activation': 'relu', 'batch_size': 10, 'epochs': 10, 'loss': 'squ
ared_hinge', 'neurons': 16, 'optimizer': 'Adadelta'}
```

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’])

```
18727/18727 [==============================] - 6s 322us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [==============================] - 6s 323us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 19/20
18727/18727 [==============================] - 6s 321us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [==============================] - 6s 322us/step - loss: 0.5038 - acc: 0.9182 - v
al_loss: 0.5038 - val_acc: 0.9242

<keras.callbacks.History at 0x7ff0a8db8cc0>
```

Model. Fit(X_train, y_train, batch_size=10, epochs=20, verbose=1, validation_data=(X_test, y_test))

Output:

```
4682/4682 [==============================] - 0s 39us/step
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

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

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

Conclusion:

Understanding earthquakes and effectively responding to them remains a complex and challenging task, even with the latest technological advancements. However, leveraging the capabilities of machine learning can greatly enhance our comprehension of seismic events. By employing machine learning techniques to analyze seismic data, we can uncover valuable insights and patterns that contribute to a deeper understanding of earthquakes. These insights can subsequently inform more effective strategies for mitigating risks and responding to seismic events.

As we head towards the future, we might see new technologies that will precisely predict the place and time of the earthquake that will happen.