IBM NAAN MUDHALVAN

Technology name: Artificial Intelligence

Project title: Earthquake prediction model using python

Phase 5: Documentation

- Clearly outline the problem statement, design thinking process, and the phases of development.
- Describe the dataset used, data preprocessing steps, and feature exploration techniques.
- Document any innovative techniques or approaches used during the development.

Dataset link:

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

Understanding the Problem Statement:

- The problem is to develop a machine learning model for the earthquake prediction using a dataset from Kaggle's repository.
- The primary objective is to explore and understand the key features in earthquake data, create visualizations for global earthquake distribution, split the data into training and testing, and build a neural network model to predict the earthquake magnitudes based on given features.

Goal:

The goal of earthquake prediction models is to contribute to the development of early warning systems that can provide real-time alerts to potentially affected areas. Such systems have the potential to save lives and reduce property damage.

PHASE 1

Design Thinking

Data Source Selection:

- The first step is to import the earthquake dataset downloaded from Kaggle.
- The dataset contains features such as date, time, latitude, longitude, depth, and magnitude

Data Preprocessing:

- Handle Missing Data: If missing data values are present in the dataset, then try to remove or amputate it.
- Data Formatting: Convert data types as needed, especially date and time features, which should be converted into datetime objects for analysis.
- Outlier Handling: Identify outliers in the dataset, which could adversely affect model performance.

Feature Exploration:

- Exploratory Data Analysis (EDA) should be conducted to understand the distribution, central tendencies, and variability of each feature.

- Identification of target variable in our dataset. The target variable in this earthquake prediction model is the earthquake magnitude.
- Calculate and visualize correlations between features and the target variable (earthquake magnitude) to identify relationships.

Visualization:

- Data visualization libraries such as matplotlib and seaborn ‘ is used to build histograms, scatter plots and correlation matrices to provide clearer understanding of the features in the dataset.
- A world map visualization depicting the frequency distribution of earthquakes globally is useful for identifying earthquake prone regions visually.

Data Splitting:

- The dataset is split into training and testing sets.
- A common practice is to allocate 80% of the data for training and 20% for testing.

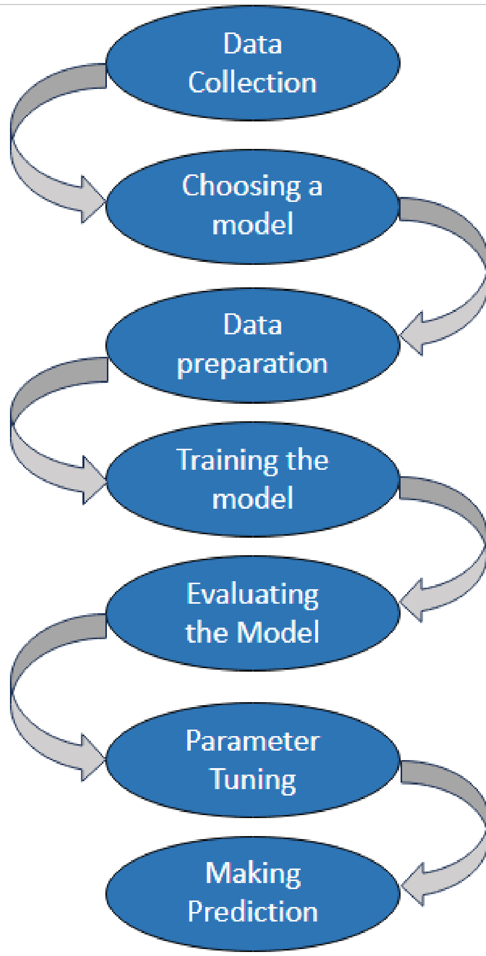
Model Development:

- Neural Networks machine learning model is used to predict the earthquake magnitudes,
- The neural network architecture should be designed by specifying the number of hidden layers, units, activation functions, and any regularization techniques (e.g., dropout) to be used.

Training and Evaluation:

- Train the neural network model using the training data and set suitable hyperparameters.
- Monitor the training process, track metrics (e.g., mean squared error, accuracy, precision, correlation matrix), and visualize training/validation loss to check for overfitting.
- Evaluate the performance of the model using appropriate evaluation metrics such as Mean Squared Error and R-squared.

Flow-Chart for Desing Thinking:



Introduction:

An earthquake prediction model using Python is a data-driven approach to predict or estimate the occurrence of earthquakes based on historical seismic data, geological information, and various machine learning techniques. In this we provide the complete model which will show the earthquake prediction model. This provides an overview of the concept and its relevance, and highlights the basic components and challenges of building such a model.

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.

PHASE II

INNOVATION

An earthquake prediction model using Python is a data-driven approach aimed at forecasting or estimating the occurrence of earthquakes based on historical seismic data, geological information, and various machine learning techniques. This will provide an overview of the concept and its relevance, highlighting the fundamental components and challenges involved in creating such a model. Here we explore innovative techniques such as ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.

Google Colab:

We have used Google Colab notebooks to perform certain tasks like data cleaning and analysis, to attain the accuracy in Earthquake prediction using Python.It's a Jupyter Notebook-based environment that offers several features and advantages. Google Colab comes pre-installed with many commonly used data science and machine learning libraries, including NumPy, Pandas, Matplotlib, TensorFlow, and PyTorch, making it convenient for data analysis and model development. You can easily connect your Google Colab environment with a GitHub repository to pull in or push out code, making version control and collaboration more efficient. You can use libraries like Matplotlib and Seaborn to create interactive visualizations within your notebooks. For a detailed walkthrough of the data cleaning and analysis process, refer to the Notebook on Google Colab, click here

Program:

```
# Importing the Libraries
import pandas as pd
import numpy as np
```

 # Loading the Dataset

```
data = pd.read_csv('database.csv')
data.head()
```

Output:

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	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	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	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	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	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic

5 rows × 21 columns

Data Analysis:

Data analysis in our Earthquake Prediction using ML project involves a meticulous exploration of seismic patterns and trends. Employing Python-based tools like NumPy and Pandas, we conducted descriptive statistics, revealing key insights into the dataset's characteristics. Visualization techniques, implemented with libraries such as Matplotlib and Seaborn, aided in uncovering spatial and temporal aspects of seismic activity. Correlation analysis provided a deeper understanding of feature relationships, guiding the model development process. The comprehensive data analysis phase contributes crucial inputs for building a robust machine learning model for earthquake prediction.

Program:

```
# Checking the Shape of the Dataset
data.shape

# Checking the Number of Entities
data.columns

# Checking Descriptive Structure of the data
data.describe()

# Checking Duplicated Rows.
data.duplicated()
```

```
# Checking the Data Information
data.info()
dataframe = pd.DataFrame(data)

# Checking Categorical and Numerical Columns
# Categorical columns
categorical_col = [col for col in dataframe.columns if
dataframe[col].dtype == 'object']
print('Categorical columns :',categorical_col)

# Numerical columns
numerical_col = [col for col in dataframe.columns if
dataframe[col].dtype != 'object']
print('Numerical columns :',numerical_col)

# Checking total number of Values in Categorical Columns
dataframe[categorical_col].nunique()

# Checking total number of Values in Numerical Columns
dataframe[numerical_col].nunique()

# Checking the Missing Values Percentage
round((dataframe.isnull().sum()/dataframe.shape[0])*100,2)
```

Output:

```
data.shape
```

```
(23412, 21)
```

```
data.columns
```

```
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')
```

```
data.describe()
```

	Latitude	Longitude	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square
count	23412.000000	23412.000000	23412.000000	4461.000000	7097.000000	23412.000000	327.000000	2564.000000	7299.000000	1604.000000	1156.000000	17352.000000
mean	1.679033	39.639961	70.767911	4.993115	275.364098	5.882531	0.071820	48.944618	44.163532	3.992660	7.662759	1.022784
std	30.113183	125.511959	122.651898	4.875184	162.141631	0.423066	0.051466	62.943106	32.141486	5.377262	10.430396	0.188545
min	-77.080000	-179.997000	-1.100000	0.000000	0.000000	5.500000	0.000000	0.000000	0.000000	0.004505	0.085000	0.000000
25%	-18.653000	-76.349750	14.522500	1.800000	146.000000	5.600000	0.046000	10.000000	24.100000	0.968750	5.300000	0.900000
50%	-3.568500	103.982000	33.000000	3.500000	255.000000	5.700000	0.059000	28.000000	36.000000	2.319500	6.700000	1.000000
75%	26.190750	145.026250	54.000000	6.300000	384.000000	6.000000	0.075500	66.000000	54.000000	4.724500	8.100000	1.130000
max	86.005000	179.998000	700.000000	91.295000	934.000000	9.100000	0.410000	821.000000	360.000000	37.874000	99.000000	3.440000

```
data.duplicated()
```

```
0        False
1        False
2        False
3        False
4        False
         ...
23407    False
23408    False
23409    False
23410    False
23411    False
Length: 23412, dtype: bool
```

```
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
```

Feature Engineering:

Feature engineering is a critical aspect of machine learning where raw data is transformed or new features are created to enhance model performance. It involves techniques like polynomial expansion, interaction terms, and domain-specific transformations to extract meaningful information. Dimensionality reduction methods, such as PCA, help manage high-dimensional data, preventing overfitting and improving model efficiency. Handling categorical variables through encoding methods ensures effective utilization of non-numeric data. Feature engineering is an iterative process, guided by continuous evaluation and refinement to build models that accurately capture underlying patterns in the data.

Program:

 # Creating Timestamp Column from Data and Time Column

```
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')
```

 # Converting the Tuple values into Series Values

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

# Droping the Date and Time Columns.
final_data = dataframe.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

Output:

	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	Timestamp
0	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic	-157630542.0
1	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic	-157465811.0
2	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic	-157355642.0
3	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic	-157093817.0
4	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic	-157026430.0

Data Cleaning:

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled

Program:

```
# Removal Of Unwanted Columns
dataframe1 = dataframe.drop(columns=['Depth Error','Depth Seismic Stations', 'Magnitude Type',
'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID',
'Source', 'Location Source', 'Magnitude Source', 'Status', 'Date',
'Time'])

# Checking the Shape of Dataset after Removing the Columns
dataframe1.shape
dataframe1.head(10)

# Checking Columns
dataframe1.columns

# Checking the Missing Values Percentage
round((dataframe1.isnull().sum()/dataframe1.shape[0])*100,2)

# Checking the Data Information After droping the Unwanted Columns
dataframe1.info()
```

```
# Checking the Descriptive Structure of the Data after the removal of Unwanted Columns
dataframe1.describe()

# Checking Categorical and Numerical Columns
# Categorical columns
categorical_col = [col for col in dataframe1.columns if dataframe1[col].dtype == 'object']
print('Categorical columns :',categorical_col)

# Numerical columns
numerical_col = [col for col in dataframe1.columns if dataframe1[col].dtype != 'object']
print('Numerical columns :',numerical_col)

# Checking total number of Values in Categorical Columns
dataframe1[categorical_col].nunique()

# Checking total number of Values in Numerical Columns
dataframe[numerical_col].nunique()

# Let's check the null values again
dataframe1.isnull().sum()
```

Output:

```
(23412, 6)
```

```
dataframe1.head(5)
```

	Latitude	Longitude	Type	Depth	Magnitude	Timestamp
0	19.246	145.616	Earthquake	131.6	6.0	-157630542.0
1	1.863	127.352	Earthquake	80.0	5.8	-157465811.0
2	-20.579	-173.972	Earthquake	20.0	6.2	-157355642.0
3	-59.076	-23.557	Earthquake	15.0	5.8	-157093817.0
4	11.938	126.427	Earthquake	15.0	5.8	-157026430.0

```
dataframe1.columns
```

```
Index(['Latitude', 'Longitude', 'Type', 'Depth', 'Magnitude', 'Timestamp'], dtype='object')
```

```
round((dataframe1.isnull().sum()/dataframe1.shape[0])*100,2)
```

```
Latitude     0.0
Longitude    0.0
Type         0.0
Depth        0.0
Magnitude    0.0
Timestamp    0.0
dtype: float64
```

```
dataframe1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23412 entries, 0 to 23411
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Latitude   23412 non-null  float64
 1   Longitude  23412 non-null  float64
 2   Type       23412 non-null  object
 3   Depth      23412 non-null  float64
 4   Magnitude  23412 non-null  float64
 5   Timestamp  23412 non-null  object
dtypes: float64(4), object(2)
memory usage: 1.1+ MB
```

```
dataframe1.describe()
```

	Latitude	Longitude	Depth	Magnitude
count	23412.000000	23412.000000	23412.000000	23412.000000
mean	1.679033	39.639961	70.767911	5.882531
std	30.113183	125.511959	122.651898	0.423066
min	-77.080000	-179.997000	-1.100000	5.500000
25%	-18.653000	-76.349750	14.522500	5.600000
50%	-3.568500	103.982000	33.000000	5.700000
75%	26.190750	145.026250	54.000000	6.000000

```
categorical_col = [col for col in dataframe1.columns if dataframe1[col].dtype == 'object']
print('Categorical columns :',categorical_col)
numerical_col = [col for col in dataframe1.columns if dataframe1[col].dtype != 'object']
print('Numerical columns :',numerical_col)
```

```
Categorical columns : ['Type', 'Timestamp']
Numerical columns : ['Latitude', 'Longitude', 'Depth', 'Magnitude']
```

```
dataframe1[categorical_col].nunique()
```

```
Type             4
Timestamp    23391
dtype: int64
```

```
dataframe1[numerical_col].nunique()
```

```
Latitude     20676
Longitude    21474
Depth         3485
Magnitude       64
dtype: int64
```

```
dataframe1.isnull().sum()
```

```
Latitude     0
Longitude    0
Type         0
Depth        0
Magnitude    0
Timestamp    0
dtype: int64
```

PHASE III

Development Part-1

Loading and Preprocessing

What is Data Loading?

Python can be used to read data from a variety of places, including databases and files. Two file types that are often used are .txt and .csv. We can import and export files using built-in Python functionality or Python's CSV library. Data loading is the first step in machine learning, which is essential for obtaining data sets for model development. Identifying the data source, whether it is CSV files, databases, or APIs, determines the loading approach. Integrating libraries such as Pandas streamlines the process, allowing users to efficiently process and analyse the data. The accompanying code snippets in the documentation illustrate the programmatic loading of datasets and provide accessibility and ease of understanding. Versatility is emphasized by addressing various data formats such as CSV, Excel, JSON, or databases, allowing for adaptation to different structures. Robust data loading also includes error handling that anticipates and handles issues such as missing values or corrupted data.

There are five ways in loading data into a python program. They are:

1. Manually loading a file
2. Numpy Methods
3. Using np.GenFromTxt
4. Using PD.read_csv
5. Using Pickle

What is preprocessing?

Pre-processing refers to the transformations applied to our data before it is fed into the algorithm. Data preprocessing is a technique used to transform raw data into a clean data set. In other words: When the data comes from different sources, it is collected in raw format, which is not suitable for analysis.

Necessity of data preprocessing:

To achieve better results with the applied model in machine learning projects, the data must be in a suitable format. Some machine learning models require information in a specific format, for example, the Random Forest algorithm does not support null values. Therefore, in order to run the Random Forest algorithm, null values must be managed from the original raw data set.

Another aspect is that the dataset should be formatted to run more than one machine learning and deep learning algorithm in a dataset and select the best one among them.

Program and Output:

#Importing libraries

```
import pandas as pd
import numpy as np

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
```

 # Reading the Dataset

```
data=pd.read_csv('/content/drive/MyDrive/database.csv')

data.head()
```

```python
data=pd.read_csv('/content/drive/MyDrive/database.csv')
data.head()
```

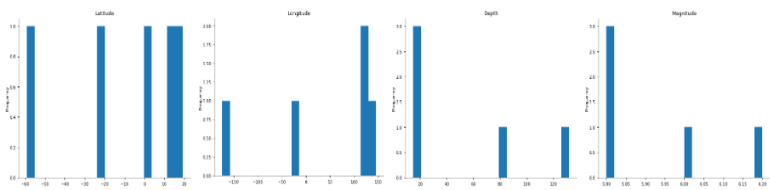
1 to 5 of 5 entries Filter

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

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Distributions



 # knowing the information about the dataset

data.info()

```
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
```

#display data

data

```
data
```

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ID
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860706
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860737
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860762
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860856
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860890
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23407	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	1.2	40.0	5.6	ML	...	18.0	42.47	0.120	NaN	0.1898	NN00570710
23408	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	2.0	33.0	5.5	ML	...	18.0	48.58	0.129	NaN	0.2187	NN00570744
23409	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	1.8	NaN	5.9	MWW	...	NaN	91.00	0.992	4.8	1.5200	US10007NAF
23410	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	1.8	NaN	6.3	MWW	...	NaN	26.00	3.553	6.0	1.4300	US10007NL0
23411	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	2.2	NaN	5.5	MB	...	428.0	97.00	0.681	4.5	0.9100	US10007NTD

23412 rows × 21 columns

 # keeping the necessary columns and deleting unnecessary columns from the dataset

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

null_columns = data.loc[:, data.isna().sum() > 0.66 *
data.shape[0]].columns

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

 # count of missing values

```
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
```

 # Filling missing values

data['Root Mean Square'] = data['Root Mean Square'].fillna(data['Root Mean Square'].mean())

 # deleting the columns with missing values and checking any missing values exist in the dataset

data = data.dropna(axis=0).reset_index(drop=True)

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

```
0
```

 # Extracting 'Month', 'Year', and 'Hour' from 'Date' and 'Time'

data['Month'] = data['Date'].apply(lambda x: x[0:2])

data['Year'] = data['Date'].apply(lambda x: x[-4:])

```
<ipython-input-39-2ccc4f5a16e1>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not mo
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  data['Month'] = data['Month'].astype(np.int)
```

 # Converting 'Month' to integer type

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

```
<ipython-input-42-ec5898e41d7b>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not moc
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))
```

 # converting month into integer types

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

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)

 # Extracting 'Hour' from 'Time'

data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))

 # Displaying the shape and columns of the final dataset

data.shape

data.columns

data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]

data.head()

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

 # Converting 'Date' and 'Time' to a timestamp in seconds

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:

        timestamp.append(‘Value Error’)



# Creating a new 'Timestamp' column

timeStamp = pd.Series(timestamp)

data['Timestamp'] = timeStamp.values



# creating and displaying the final dataset

final_data = data.drop(['Date', 'Time'], axis=1)

final_data = final_data[final_data.Timestamp != 'ValueError']
```

```
final_data.head()
```

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-157630542.0
1	1.863	127.352	80.0	5.8	-157465811.0
2	-20.579	-173.972	20.0	6.2	-157355642.0
3	-59.076	-23.557	15.0	5.8	-157093817.0
4	11.938	126.427	15.0	5.8	-157026430.0

PHASE IV

Development Part-2

<u>Visualizing the data on the world map</u>

<u>Splitting the dataset into Training and Testing sets</u>

What is Data Visualization?

The importance of data visualization is simple: it helps people see, interact with, and better understand data. Whether simple or complex, the right visualization can get everyone on the same page, regardless of their skill level.

There is probably no industry that doesn't benefit from making data more understandable. Every STEM field benefits from understanding data— - as do fields in government, finance, marketing, history, consumer products, services, education, sports, and so on.

While we'll wax poetic about data visualization time and time again (you are on the Tableau website, after all), there are undeniably practical, real-life applications. And because visualization is so widely used, it's also one of the most useful professional skills you can develop. The better you can communicate your arguments visually, whether in a dashboard or a slide deck, the better you can use that information. The concept of the citizen data scientist is on the rise. Skills are changing to meet a data-driven world. It is becoming increasingly important for professionals to use data to make decisions and tell stories using visuals where data drives the who, what, when, where, and how.

While traditional education typically draws a clear line between creative storytelling and technical analysis, the modern profession also values those who can mediate between the two: Data visualization is right in the middle between analysis and visual storytelling.

What is Data splitting?

Scikit-learn alias sklearn is the most useful and robust library for machine learning in Python. The scikit-learn library provides us with the model_selection module in which we have the splitter function train_test_split()

Syntax:

```
train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
```

Parameters:

1. *arrays: inputs such as lists, arrays, data frames, or matrices
2. test_size: this is a float value whose value ranges between 0.0 and 1.0. it represents the proportion of our test size. its default value is none.

3. train_size: this is a float value whose value ranges between 0.0 and 1.0. it represents the proportion of our train size. its default value is none.

4. random_state: this parameter is used to control the shuffling applied to the data before applying the split. it acts as a seed.
5. shuffle: This parameter is used to shuffle the data before splitting. Its default value is true.
6. stratify: This parameter is used to split the data in a stratified fashion.

Dataset location:

/content/drive/MyDrive/dataset/database.csv

File location:

https://colab.research.google.com/drive/1X09h3D1-JiF4Bkfu6bNfKIIkkArOjJtd?usp=sharing

Program and Output:

#installing the basemap into colab

```
!pip install basemap
```

#Importing the necessary libraries

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import seaborn as sns
sns.set(style='darkgrid')
```

#Checking the minimum and maximum of magnitude

```
print("Min Value: "+ str(data['Magnitude'].min()))
print("Max Value: "+ str(data['Magnitude'].max()))
```

```
Min Value: 5.5
Max Value: 9.1
```

#magnitudes greater than 8 to 4

```
Greater_8 = data[data['Magnitude'] > 8]
Greater_8['Location Source'].value_counts()
```

```
US         22
ISCGEM      5
Name: Location Source, dtype: int64
```

```
Greater_7 = data[data['Magnitude'] > 7]
Greater_7['Location Source'].value_counts()
```

```
US        467
ISCGEM     92
CI          3
H           1
AG          1
SPE         1
AGS         1
NC          1
AEIC        1
WEL         1
GUC         1
Name: Location Source, dtype: int64
```

Greater_6 = data[data['Magnitude'] > 6]
Greater_6['Location Source'].value_counts()

```
US        4781
ISCGEM     885
NC          21
CI          18
GCMT        14
PGC          6
GUC          5
HVO          4
AGS          4
AEIC         4
UNM          3
SPE          3
WEL          3
AK           3
MDD          2
H            2
ATH          2
CASC         1
AEI          1
TEH          1
US_WEL       1
THR          1
SJA          1
JMA          1
ROM          1
U            1
NN           1
AG           1
ISK          1
UW           1
BOU          1
Name: Location Source, dtype: int64
```

```
Greater_5 = data[data['Magnitude'] > 5]
Greater_5['Location Source'].value_counts()
```

```
US          20350
ISCGEM       2581
CI             61
GCMT           56
NC             54
GUC            46
AEIC           40
UNM            21
PGC            19
WEL            18
AGS            17
ISK            15
AK             14
ATH            14
HVO            12
SPE            10
ROM             7
AEI             7
TEH             7
H               7
UW              6
CASC            4
NN              4
US_WEL          4
ATLAS           3
THR             3
THE             3
JMA             3
RSPR            3
TUL             2
B               2
G               2
MDD             2
TAP             1
BEO             1
SE              1
UCR             1
LIM             1
CSEM            1
SJA             1
CAR             1
BRK             1
U               1
AG              1
OTT             1
SLC             1
BOU             1
PR              1
Name: Location Source, dtype: int64
```

```
Greater_4 = data[data['Magnitude'] > 4]
```

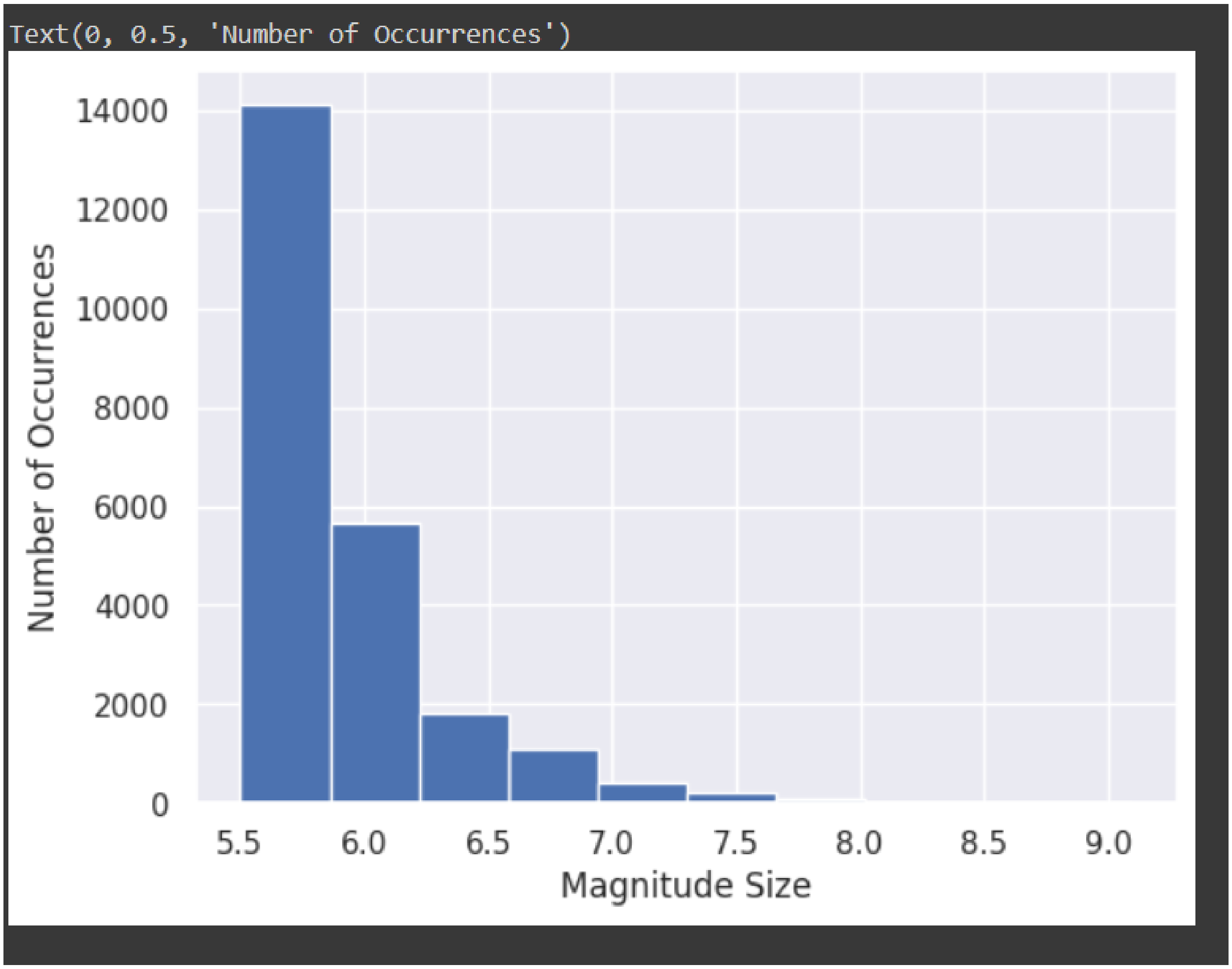
Greater_4['Location Source'].value_counts()

```
US          20350
ISCGEM       2581
CI             61
GCMT           56
NC             54
GUC            46
AEIC           40
UNM            21
PGC            19
WEL            18
AGS            17
ISK            15
AK             14
ATH            14
HVO            12
SPE            10
ROM             7
AEI             7
TEH             7
H               7
UW              6
CASC            4
NN              4
US_WEL          4
ATLAS           3
THR             3
THE             3
JMA             3
RSPR            3
TUL             2
B               2
G               2
```

```
MDD             2
TAP             1
BEO             1
SE              1
UCR             1
LIM             1
CSEM            1
SJA             1
CAR             1
BRK             1
U               1
AG              1
OTT             1
SLC             1
BOU             1
PR              1
Name: Location Source, dtype: int64
```

#plotting the histogram for magnitude

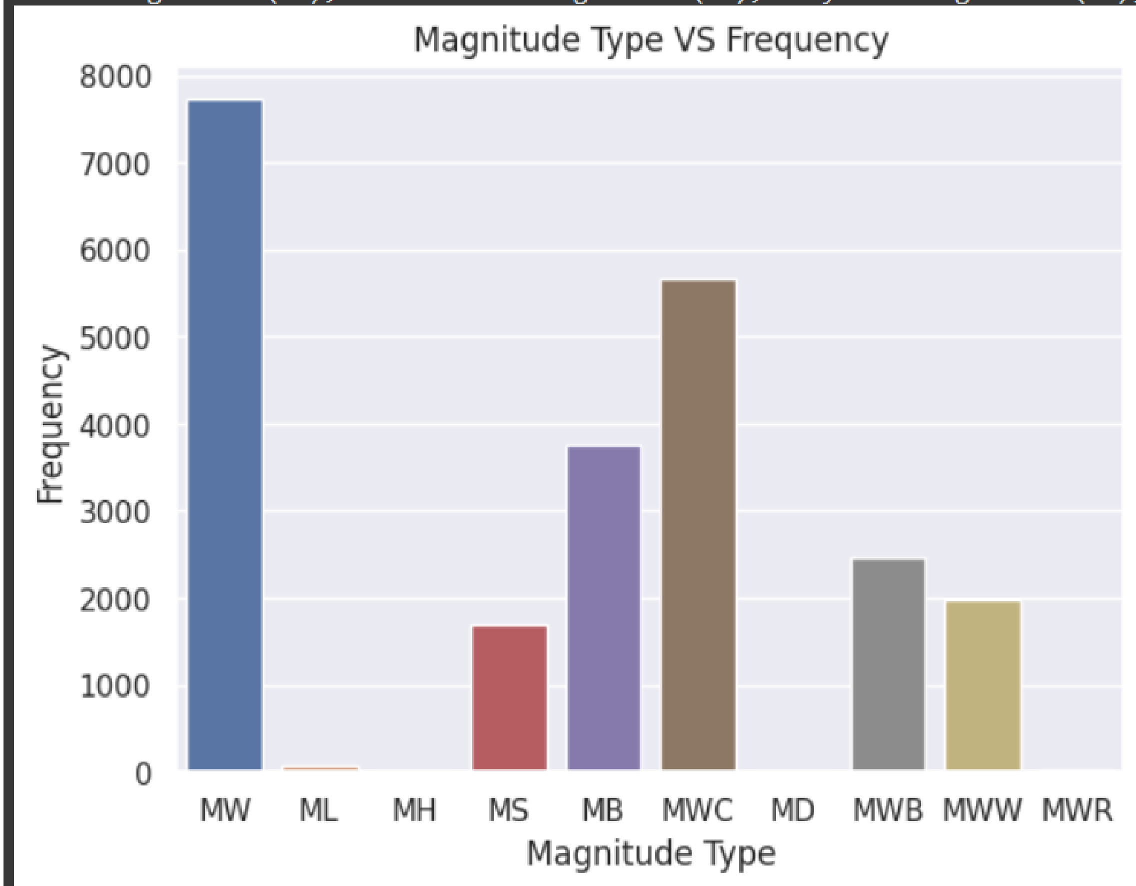
```
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')
```



#plotting the count-plot for magnitude type

```
sns.countplot(x="Magnitude Type", data=data)
plt.ylabel('Frequency')
plt.title('Magnitude Type VS Frequency')
print(" local magnitude (ML), surface-wave magnitude (Ms), body-wave magnitude (Mb), moment magnitude (Mm)")
```

local magnitude (ML), surface-wave magnitude (Ms), body-wave magnitude (Mb), moment magnitude (Mm)



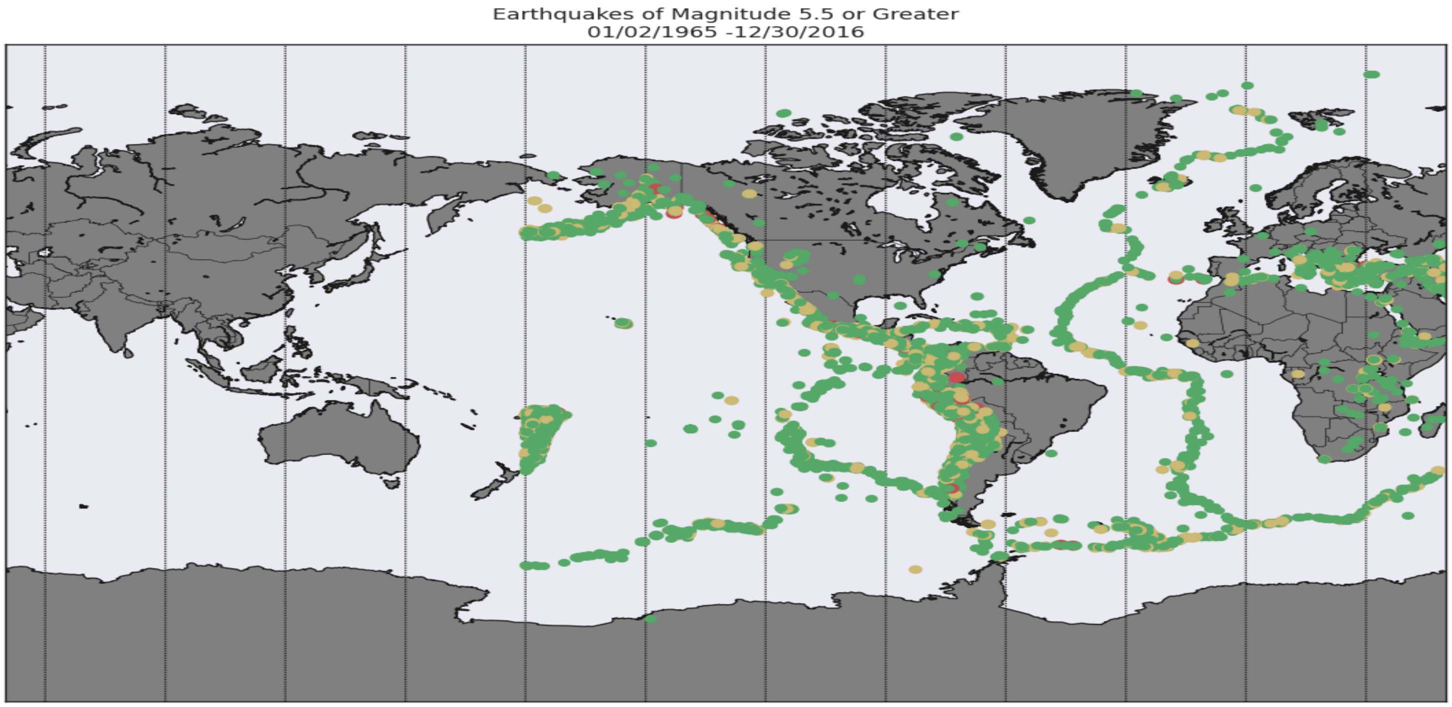
#marking the magnitude

```
from numpy.ma.core import make_mask
from matplotlib.axes import ma
def get_marker_color(magnitude):
  if magnitude < 6.2:
    return ('go')
  elif magnitude < 7.5:
    return ('yo')
  else:
    return('ro')
```

#marking different colours of magnitude on basemap plot

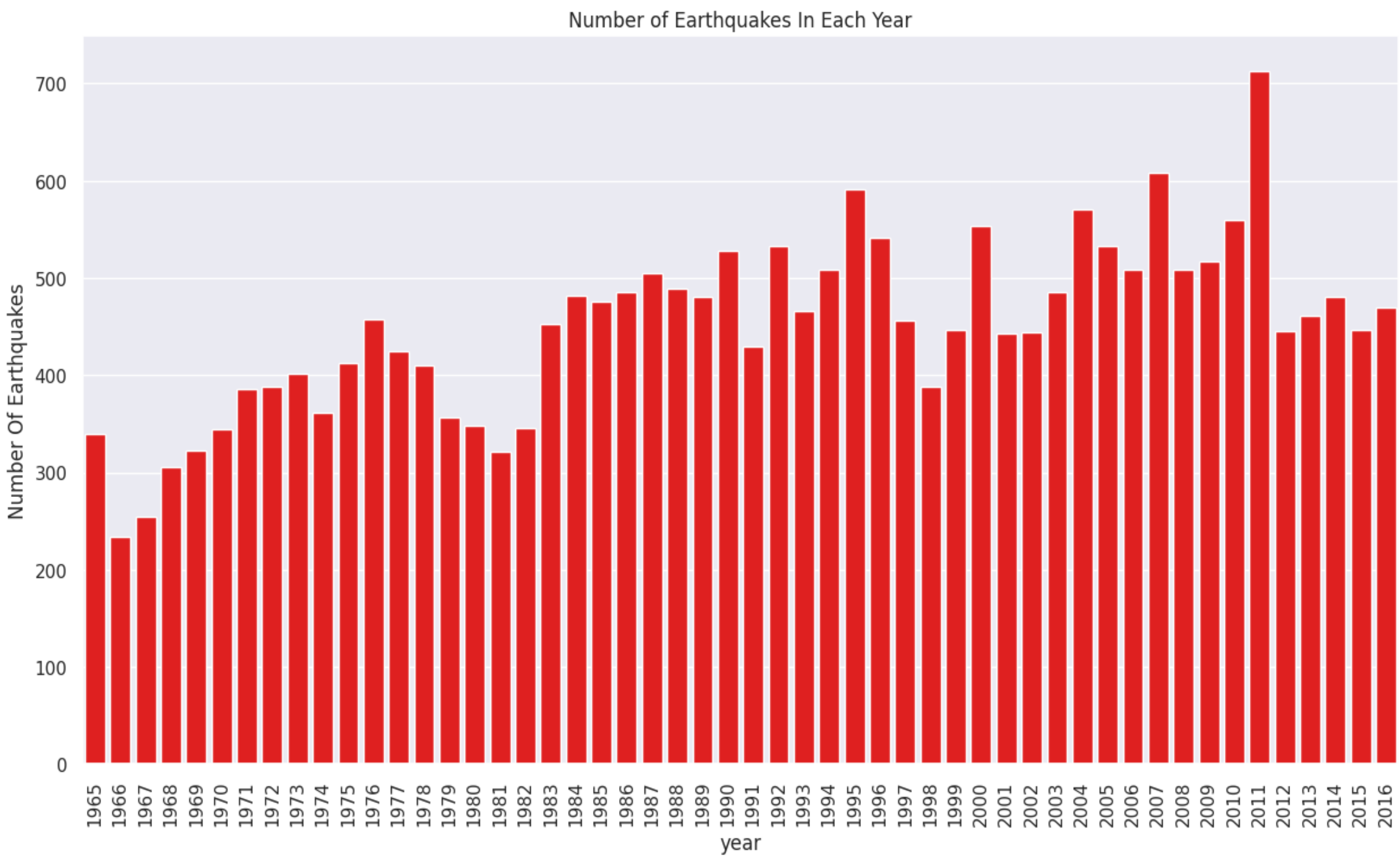
```
plt.figure(figsize=(14,10))
eq_map = Basemap(projection='mill', resolution = 'l', lat_0 = 0, lon_0 = -130)
```

```
eq_map.drawcoastlines()
eq_map.drawcountries()
eq_map.fillcontinents(color = 'grey')
eq_map.drawmapboundary()
eq_map.drawmeridians(np.arange(0, 360, 30))

#latitude, longitude and magnitude

lons = data['Longitude'].values
lats = data['Latitude'].values
mags = data['Magnitude'].values
timestrings = data['Date'].tolist()

min_marker_size = 0.5
for lon, lat, mag in zip(lons, lats, mags):
  x,y = eq_map(lon, lat)
  msize = mag
  marker_string = get_marker_color(mag)
  eq_map.plot(x,y, marker_string, markersize = msize)

title_string = "Earthquakes of Magnitude 5.5 or Greater\n"
title_string += "%s -%s" % (timestrings[0][:10], timestrings[-1][:10])
plt.title(title_string)

plt.show()
```



#countplot for no. of earthquakes each year

```
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['year'] = data['date'].apply(lambda x: str(x).split('-')[0])
plt.figure(figsize=(15, 8))
sns.set(font_scale=1.0)
ax = sns.countplot(x="year", data=data, color = "red")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each Year')
```



#Ranking the highest no. earthquakes in each year

data['year'].value_counts()[:5]

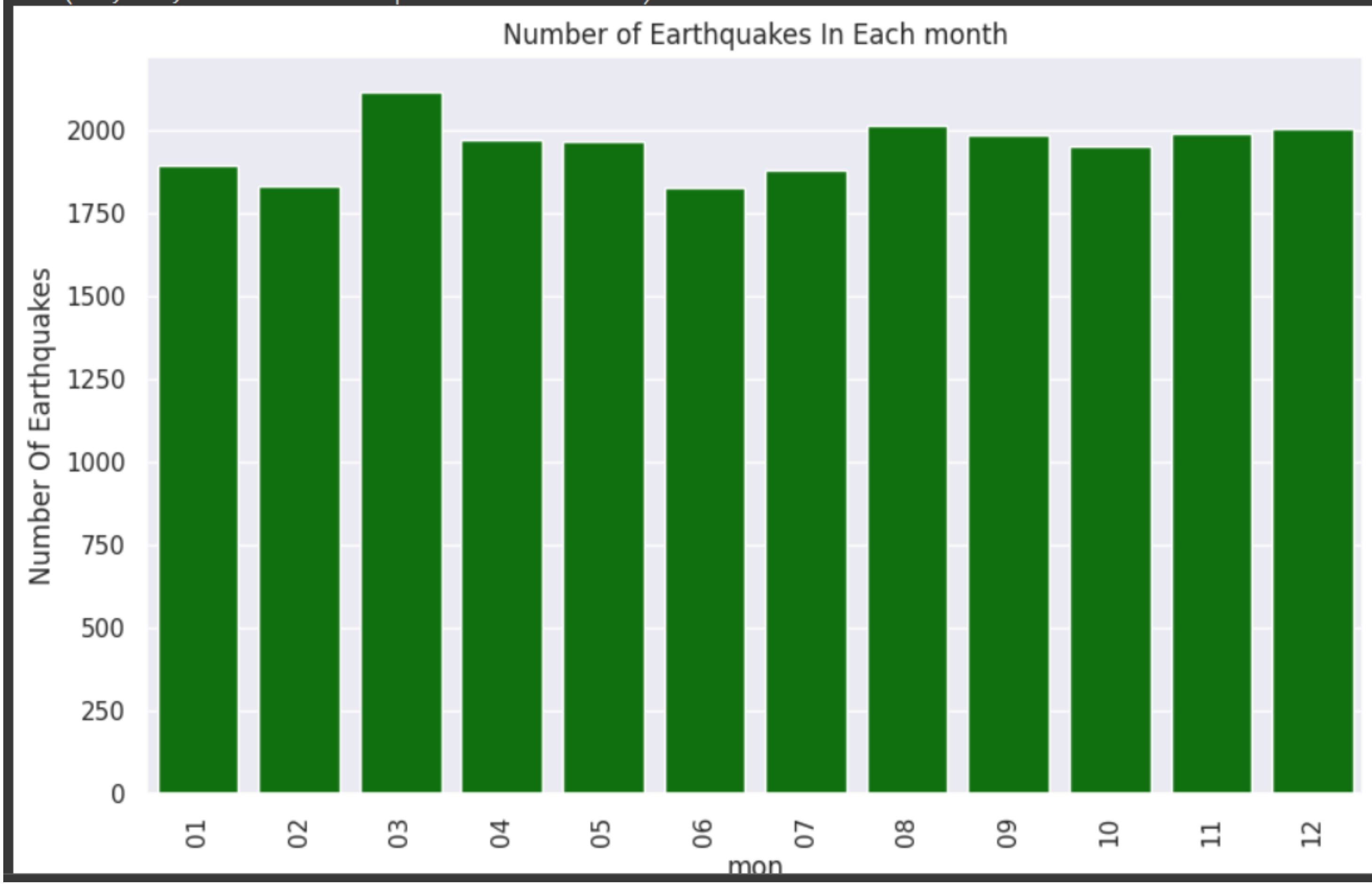
```
2011    713
2007    608
1995    591
2004    571
2010    560
Name: year, dtype: int64
```

#countplot for no. of earthquakes in each month

```
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['mon'] = data['date'].apply(lambda x: str(x).split('-')[1])
plt.figure(figsize=(10, 6))
sns.set(font_scale=1)
ax = sns.countplot(x="mon", data=data, color = "green")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each month')
```

```
Text(0.5, 1.0, 'Number of Earthquakes In Each month')
```



#Ranking the highest no. earthquakes in each months

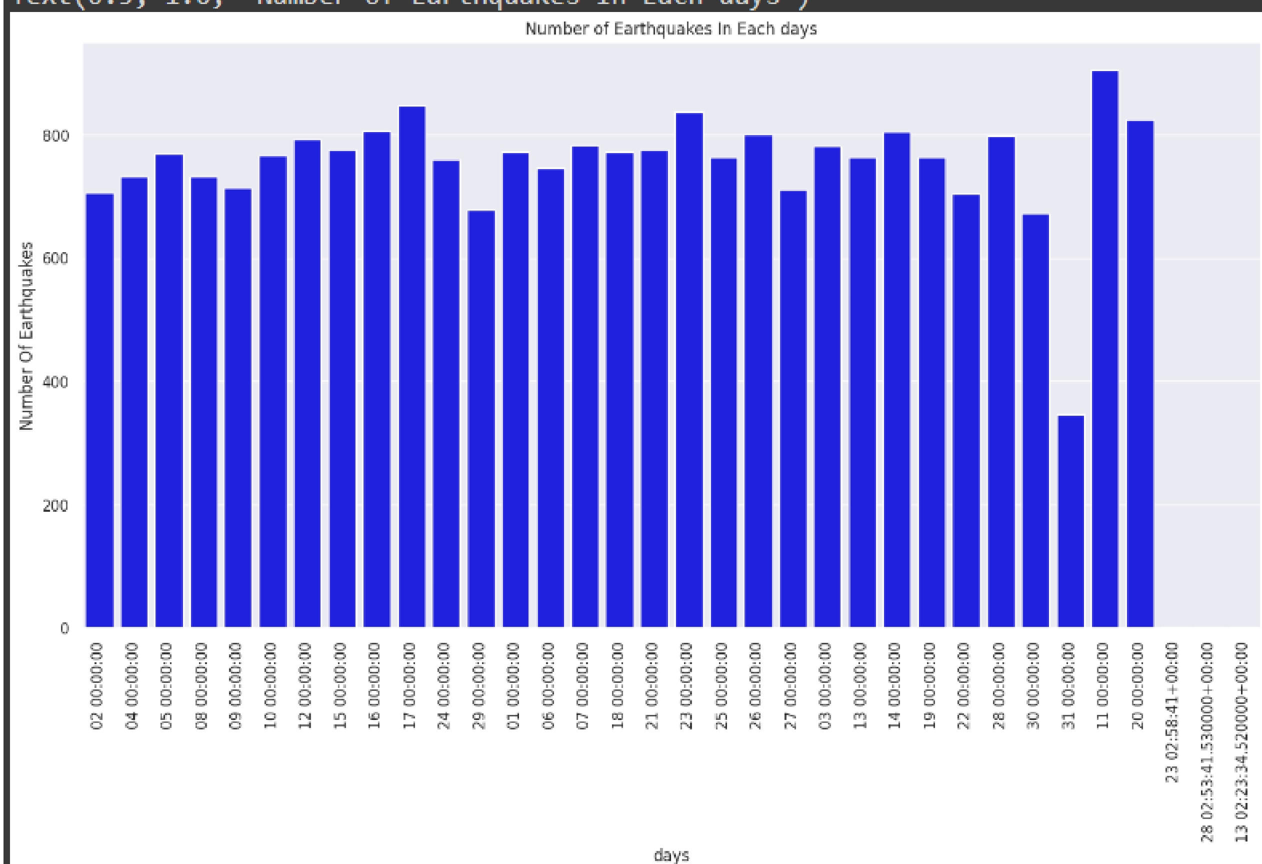
```
data['mon'].value_counts()[:5]
```

```
03    2114
08    2014
12    2001
11    1987
09    1985
Name: mon, dtype: int64
```

#countplot for no. of earthquakes in each day

```
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['days'] = data['date'].apply(lambda x: str(x).split('-')[-1])
plt.figure(figsize=(16, 8))
sns.set(font_scale=1.0)
ax = sns.countplot(x="days", data=data, color = "blue")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each days')
```

```
Text(0.5, 1.0, 'Number of Earthquakes In Each days')
```



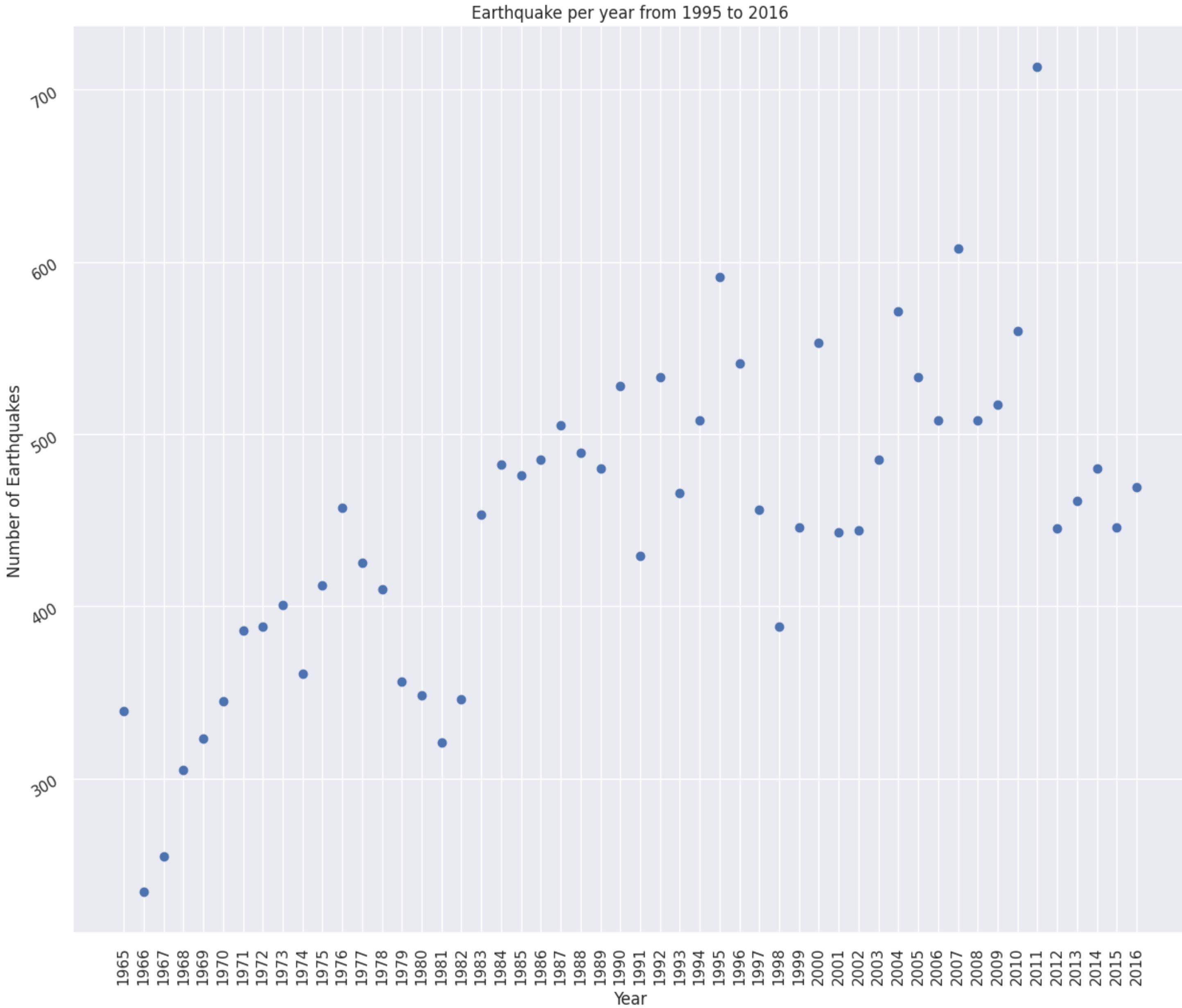
#Ranking the highest no. earthquakes in each day

```
data['days'].value_counts()[:5]
```

```
11 00:00:00    905
17 00:00:00    848
23 00:00:00    837
20 00:00:00    825
16 00:00:00    807
Name: days, dtype: int64
```

 # Scatter plot of the number of earthquakes per year from 1995 to 2016

```
x = data['year'].unique()
y = data['year'].value_counts()

count = []
for i in range(len(x)):
    key = x[i]
    count.append(y[key])

#Scatter Plot
plt.figure(figsize =(15,12))

plt.scatter(x,count)
plt.title("Earthquake per year from 1995 to 2016")
plt.xlabel("Year")
plt.xticks(rotation=90)
plt.ylabel("Number of Earthquakes")
plt.yticks(rotation=30)
plt.show()
```

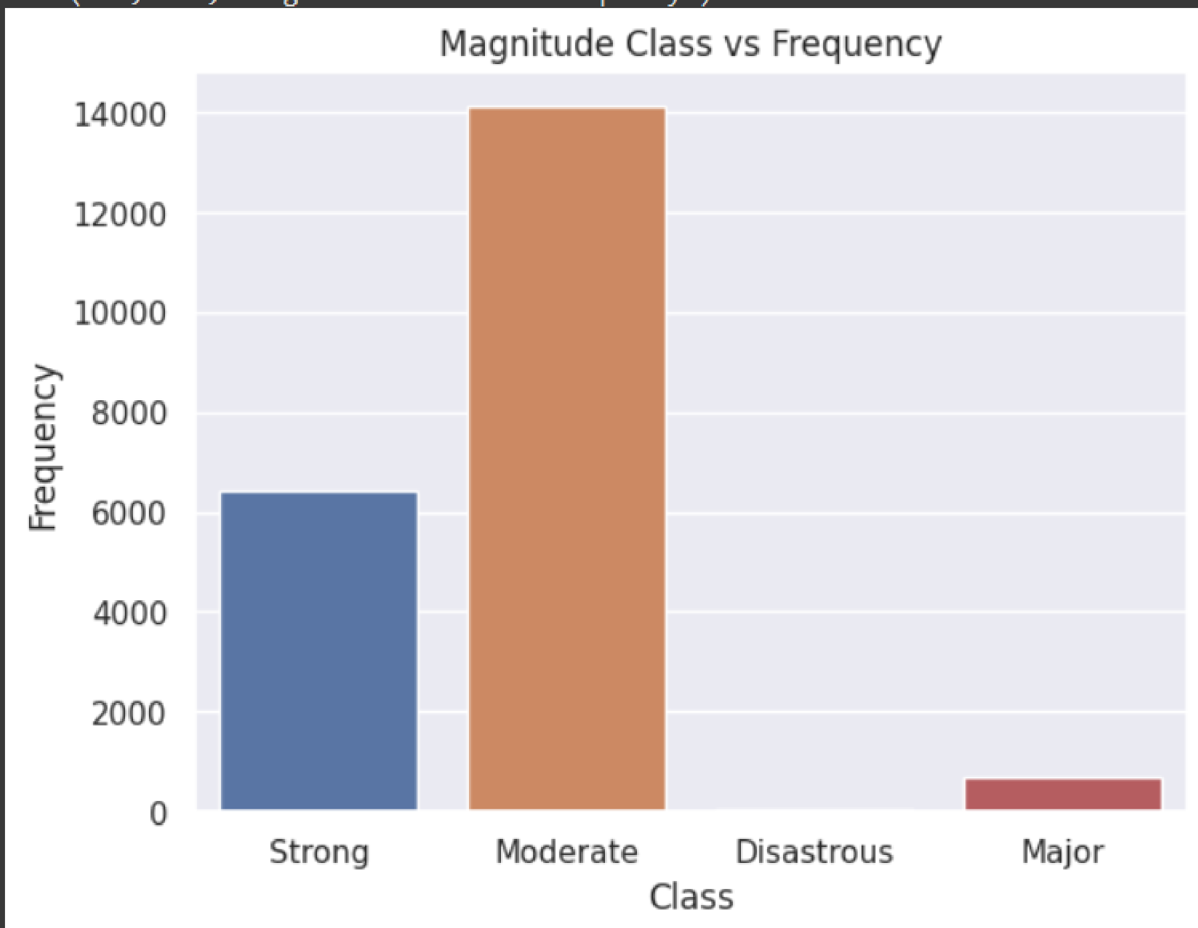


#Classification of magnitude types

data.loc[data['Magnitude'] >=8, 'Class'] = 'Disastrous'
data.loc[(data['Magnitude'] >= 7) & (data['Magnitude'] < 7.9), 'Class'] = 'Major'
data.loc[(data['Magnitude'] >= 6) & (data['Magnitude'] < 6.9), 'Class'] = 'Strong'
data.loc[(data['Magnitude'] >= 5.5) & (data['Magnitude'] < 5.9), 'Class'] = 'Moderate'

```
# Magnitude Class distribution

sns.countplot(x='Class', data=data)
plt.ylabel('Frequency')
plt.title('Magnitude Class vs Frequency')
```

```
Text(0.5, 1.0, 'Magnitude Class vs Frequency')
```



#splitting the data…

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

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=42)
print(X_train.shape, X_test.shape, y_train.shape,  X_test.shape)
```

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

PHASE V

Building model

Model Selection

In the model selection phase of a machine learning project, the crucial task is to identify the most appropriate algorithm for the given problem and dataset. This phase involves a systematic exploration of various models to find the one that best fits the data and achieves the desired predictive performance. Researchers and data scientists evaluate a spectrum of algorithms, ranging from classic approaches like linear regression to sophisticated techniques such as support vector machines or neural networks. The choice often depends on the nature of the problem, the characteristics of the data, and the trade-off between model complexity and interpretability. Hyperparameter tuning further refines the selected model, optimizing its performance. Model selection is an iterative process, guided by cross-validation techniques and performance metrics tailored to the specific problem, ensuring that the chosen model generalizes well to unseen data. A thorough understanding of the data and problem domain is crucial during this phase, empowering practitioners to make informed decisions that lay the foundation for a successful machine learning solution.

Model Training

Model training is a critical phase in machine learning where the selected algorithm learns patterns and relationships from the provided data. During this process, the model is exposed to a labeled training dataset, and it adjusts its internal parameters to minimize the difference between its predictions and the actual outcomes. This optimization is often performed using techniques like gradient descent, where the algorithm iteratively refines its parameters. The training dataset is typically divided into batches to efficiently process large volumes of data. The model's performance is continuously assessed using a loss function, which quantifies the disparity between predicted and actual values. Hyperparameter tuning is often performed at this stage to optimize the model's configuration. The ultimate goal of model training is to create a well-generalized model that can make accurate predictions on new, unseen data. Regularization techniques are frequently employed to prevent overfitting, ensuring the model's adaptability to diverse datasets. Upon successful training, the model is ready for evaluation and, eventually, deployment in real-world applications.

Model Evaluation

Model evaluation is a critical phase in the machine learning lifecycle, determining the effectiveness of a trained model. Metrics such as accuracy, precision, recall, and F1 score offer insights into its performance. These metrics quantify the model's ability to make correct predictions and handle class imbalances. Additionally, techniques like cross-validation assess its robustness across different subsets of data. A well-evaluated model strikes a balance between bias and variance, avoiding overfitting or underfitting. Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) provide a holistic view of a model's discriminative power, especially in binary classification tasks. Understanding the model's strengths and weaknesses through evaluation guides further refinements, ensuring its reliability when deployed in real-world scenarios. Continuous monitoring and validation against unseen data are essential to maintain its efficacy over time. Comprehensive documentation of the evaluation process enhances transparency, facilitating collaboration and informed decision-making in model selection and deployment

Hyperparameter Tuning

Hyperparameter tuning is a crucial step in optimizing the performance of a machine learning model. It involves systematically adjusting the hyperparameters, which are configuration settings external to the model itself, to enhance its predictive capabilities. This process aims to strike a balance between underfitting and overfitting, ensuring the model generalizes well to new, unseen data. Common techniques for hyperparameter tuning include grid search and randomized search, where different combinations of hyperparameter values are explored. The choice of hyperparameters, such as learning rates or regularization strengths, profoundly influences a model's effectiveness. Fine-tuning these parameters requires a delicate trade-off, often involving iterative experimentation and validation. Successful hyperparameter tuning can significantly improve a model's accuracy and robustness, contributing to its overall effectiveness in real-world applications. As models become more complex, the importance of thoughtful hyperparameter selection continues to grow, making it a critical aspect of the machine learning model development process.

Model Development

Model deployment is a critical phase in the machine learning life cycle, marking the transition from development to practical application. Once a model has been trained and validated, deployment involves integrating it into a production environment for real-time use. The deployment process includes optimizing the model for efficiency, ensuring compatibility with the target system, and establishing a reliable and scalable infrastructure. It is crucial to monitor the deployed model's performance in real-world scenarios and implement mechanisms for continuous improvement. Security considerations, such as data privacy and model robustness, should be addressed during deployment to mitigate potential risks. Comprehensive documentation of the deployment process facilitates seamless collaboration and maintenance. Overall, effective model deployment is essential for translating machine learning innovations into tangible, impactful solutions within various domains.

Program:

 # Logistic Regression Model

 # Importing necessary libraries

import sklearn from sklearn import linear_model

from sklearn.linear_model import LogisticRegression

from sklearn import metrics

from sklearn.model_selection import train_test_split

 # Selecting features and target variable

x = df[['Latitude', 'Longitude', 'Timestamp']]y =

df[['Magnitude']]

 # Splitting the dataset into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.3, random_state=0)

print(x_train.shape, x_test.shape, y_train.shape,x_test.shape)

 # Creating and training the Logistic Regressionmodel

log = LogisticRegression()

model = log.fit(x_train, y_train)

y_pred = log.predict(x_test)

```
# Evaluating the model's accuracy
print("Accuracy is:", (metrics.accuracy_score(y_test,y_pred)) * 100)

# Neural Network Model

import sklearn
from sklearn.model_selection import train_test_split,
GridSearchCV
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn importKerasClassifier

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.3,
random_state=0)
print(x_train.shape,x_test.shape,      y_train.shape,x_test.shape)


# Defining a function to create a neural networkmodel
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


# Creating a KerasClassifier
model = KerasClassifier(build_fn=create_model, verbose=0)
# Defining a parameter grid for hyperparametertuning
param_grid = { "neurons": [16, 64],
"batch_size": [10, 20],"epochs":
[10],"activation": ['sigmoid', 'relu'],
"optimizer": ['SGD', 'Adadelta'],"loss":
['squared_hinge']
}

# Converting data to numpy arrays


x_train = np.asarray(x_train).astype(np.float32)y_train =
np.asarray(y_train).astype(np.float32)x_test =
np.asarray(x_test).astype(np.float32)
y_test = np.asarray(y_test).astype(np.float32)
# Using GridSearchCV to find the best parametersfor the model
grid=GridSearchCV(estimator=model,param_grid=param_grid,
n_jobs=-1)
grid_result = grid.fit(x_train, y_train)


# Retrieving the best parameters
best_params = grid_result.best_params_
```

```
# Creating and training the final model with the bestparameters

model = Sequential()

model.add(Dense(16, activation=best_params['activation'],
input_shape=(3,)))

model.add(Dense(16, activation=best_params['activation']))

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

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

model.fit(x_train, y_train, batch_size=best_params['batch_size'],
epochs=best_params['epochs'],

verbose=1,validation_data=(x_test, y_test))


# Evaluating the final model on the test set

[test_loss, test_acc] = model.evaluate(x_test, y_test)

print("Evaluation result on Test Data: Loss = {},accuracy =
{}".format(test_loss, test_acc))
```

OUTPUT :

Logistic Regression Model

```
[128] import sklearn
      from sklearn import linear_model
      from sklearn.linear_model import LogisticRegression
      from sklearn import metrics
      from sklearn.model_selection import train_test_split
      x = df[['Latitude', 'Longitude', 'Timestamp']]
      y = df[['Magnitude']]
      x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
      print(x_train.shape,x_test.shape)
```

```
(17421, 3) (5808, 3)
```

```
log=LogisticRegression()
model=log.fit(x_train,y_train)
y_pred=log.predict(x_test)
print("Accuracy is:",(metrics.accuracy_score(y_test,y_pred))*100)
```

```
Accuracy is: 92.8374655647383
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when
  y = column_or_1d(y, warn=True)
```

```
[130] !pip install keras==2.12.0
```

```
Requirement already satisfied: keras==2.12.0 in /usr/local/lib/python3.10/dist-packages (2.12.0)
```

```
import sklearn
from sklearn.model_selection import train_test_split, GridSearchCV

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
print(x_train.shape, x_test.shape, y_train.shape, x_test.shape)
from keras.models import Sequential
from keras.layers import Dense

# 3 dense layers, 16, 16, 2 nodes each

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)

param_grid = {
    "neurons": [16, 64],
    "batch_size": [10, 20],
    "epochs": [10],
    "activation": ['sigmoid', 'relu'],
    "optimizer": ['SGD', 'Adadelta'],
    "loss": ['squared_hinge']
}
```

```
(16260, 3) (6969, 3) (16260, 1) (6969, 3)
<ipython-input-131-a51d28c0118e>:22: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. S
  model = KerasClassifier(build_fn=create_model, verbose=0)
```

```
[132] x_train = np.asarray(x_train).astype(np.float32)
      y_train = np.asarray(y_train).astype(np.float32)
      x_test = np.asarray(x_test).astype(np.float32)
      y_test = np.asarray(y_test).astype(np.float32)
```

GridSearchCV is used for finding the best parameters for tuning the model's performance

```
[127] print(x_train.shape,y_train.shape)

      (16260, 3) (16260, 1)
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)

best_params = grid_result.best_params_
best_params
```

```
{'activation': 'relu',
 'batch_size': 10,
 'epochs': 10,
 'loss': 'squared_hinge',
 'neurons': 16,
 'optimizer': 'SGD'}
```

```
[ ] model = Sequential()
    model.add(Dense(16, activation=best_params['activation'], input_shape=(3,)))
    model.add(Dense(16, activation=best_params['activation']))
    model.add(Dense(2, activation='softmax'))

    model.compile(optimizer=best_params['optimizer'], loss=best_params['loss'], metrics=['accuracy'])
    model.fit(x_train, y_train, batch_size=best_params['batch_size'], epochs=best_params['epochs'], verbose=1, validation_data=(x_test, y_test))

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

```
Epoch 1/10
1626/1626 [==============================] - 16s 9ms/step - loss: nan - accuracy: 0.9900 - val_loss: nan - val_accuracy: 0.9918
Epoch 2/10
1626/1626 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 3/10
1626/1626 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 4/10
1626/1626 [==============================] - 8s 5ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 5/10
1626/1626 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 6/10
1626/1626 [==============================] - 5s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 7/10
1626/1626 [==============================] - 6s 4ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 8/10
1626/1626 [==============================] - 6s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 9/10
1626/1626 [==============================] - 4s 2ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
Epoch 10/10
1626/1626 [==============================] - 4s 3ms/step - loss: nan - accuracy: 0.9932 - val_loss: nan - val_accuracy: 0.9918
218/218 [==============================] - 1s 2ms/step - loss: nan - accuracy: 0.9918
Evaluation result on Test Data : Loss = nan, accuracy = 0.9918209314346313
```

Conclusion

In conclusion, the development of a machine learning model is a multifaceted journey that encompasses problem definition, data collection, preprocessing, exploratory data analysis, and feature engineering. The thoughtful selection of an appropriate model, meticulous training, and rigorous evaluation are pivotal to achieving robust predictive performance. The iterative processes of hyperparameter tuning and deployment usher the model into real-world applications. Continuous monitoring and maintenance ensure its relevance and effectiveness over time. Documentation stands as a beacon, illuminating the path taken, aiding collaboration, and facilitating future enhancements. In this dynamic landscape, the synergy of these phase crafts a holistic and adaptive framework, essential for the successful integration of machine learning solutions into diverse domains