IBM NaanMudhalvan

ARTIFICIAL INTELLIGENCE

Project Title : Earthquake Prediction 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.

Workbook Link : Google Colab

Problem Definition :

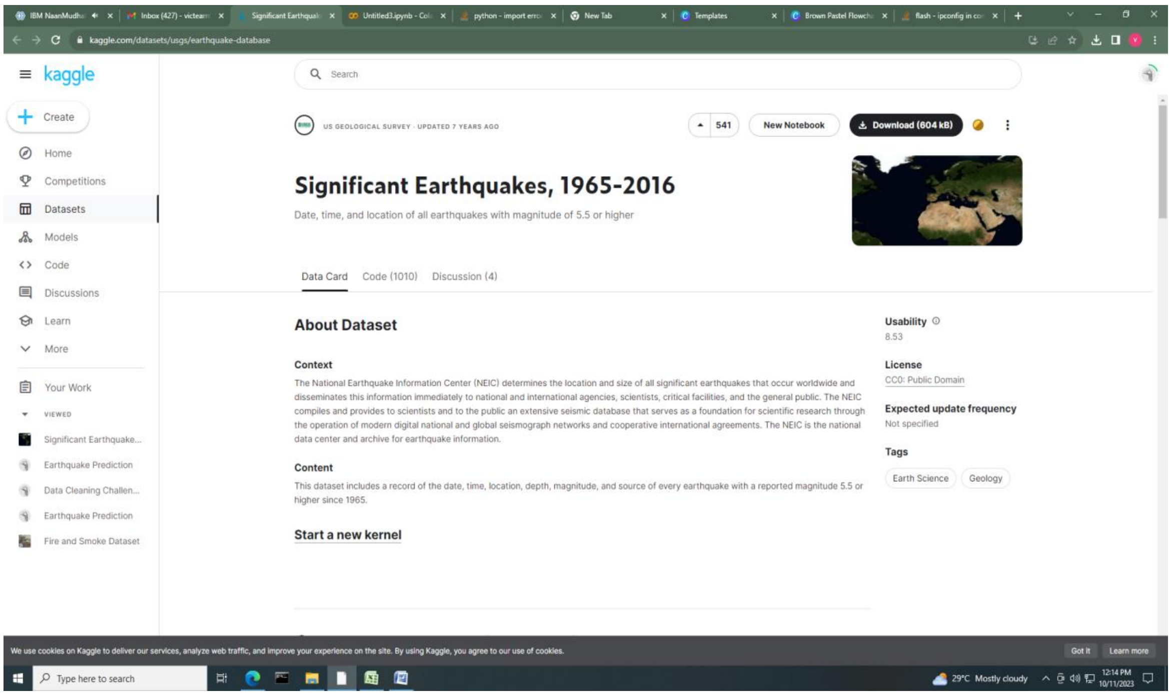
The problem at hand is to develop an earthquake prediction model using a kaggle dataset. The primary objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and ultimately construct a neural network model that can predict earthquake magnitudes based on the provided features.

DESIGN THINKING

Data Source

The first step in solving this problem is selecting a suitable kaggle dataset that contains earthquake data. This dataset should include essential features such as date, time, latitude, longitude, depth, and magnitude. The choice of the dataset is crucial as it forms the foundation of our model.

Dataset Source :



US GEOLOGICAL SURVEY · UPDATED 7 YEARS AGO

Significant Earthquakes, 1965-2016

Date, time, and location of all earthquakes with magnitude of 5.5 or higher

About Dataset

Context

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.

Start a new kernel

Usability 8.53

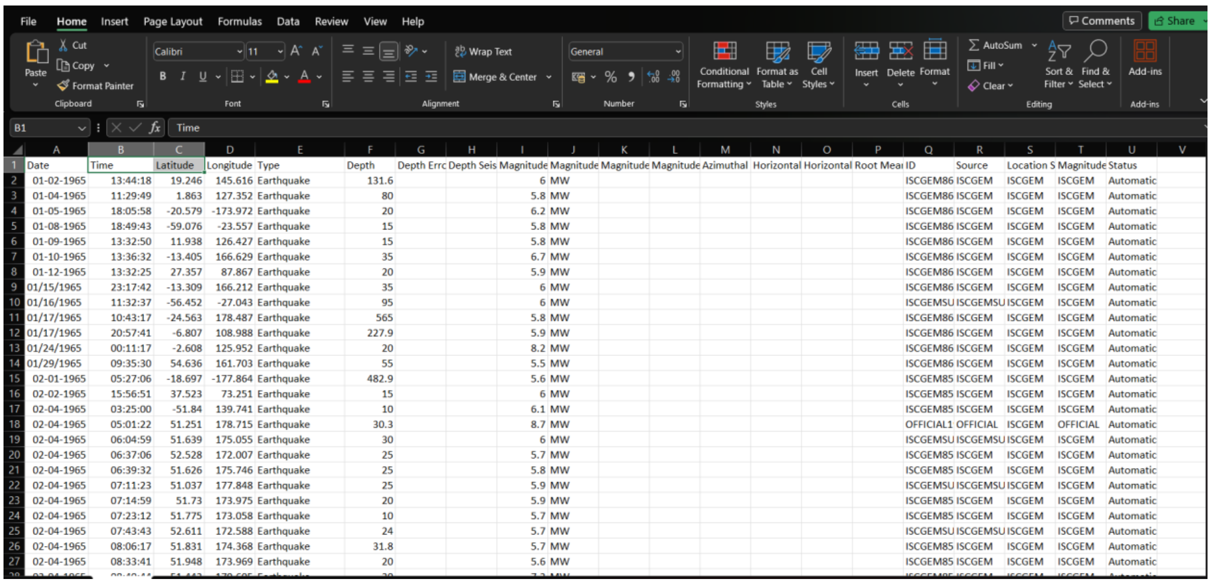
License CC0: Public Domain

Expected update frequency Not specified

Tags Earth Science, Geology

We use cookies on Kaggle to deliver our services, analyze web traffic, and improve your experience on the site. By using Kaggle, you agree to our use of cookies.

Sample Data Snapshot :



1	Date	Time	Latitude	Longitude	Type	Depth	Depth Errc	Depth Seis	Magnitude	Magnitude	Magnitude	Magnitude	Azimuthal	Horizontal	Horizontal	Root Mear	ID	Source	Location S	Magnitude	Status
2	01-02-1965	13:44:18	19.246	145.616	Earthquake	131.6			6	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01-04-1965	11:29:49	1.863	127.352	Earthquake	80			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01-05-1965	18:05:58	-20.579	-173.972	Earthquake	20			6.2	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
5	01-08-1965	18:49:43	-59.076	-23.557	Earthquake	15			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
6	01-09-1965	13:32:50	11.938	126.427	Earthquake	15			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
7	01-10-1965	13:36:32	-13.405	166.629	Earthquake	35			6.7	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
8	01-12-1965	13:32:25	27.357	87.867	Earthquake	20			5.9	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
9	01/15/1965	23:17:42	-13.309	166.212	Earthquake	35			6	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
10	01/16/1965	11:32:37	-56.452	-27.043	Earthquake	95			6	MW							ISCGEMSU	ISCGEMSU	ISCGEM	ISCGEM	Automatic
11	01/17/1965	10:43:17	-24.563	178.487	Earthquake	565			5.8	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
12	01/17/1965	20:57:41	-6.807	108.988	Earthquake	227.9			5.9	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
13	01/24/1965	00:11:17	-2.608	125.952	Earthquake	20			8.2	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
14	01/29/1965	09:35:30	54.636	161.703	Earthquake	55			5.5	MW							ISCGEM86	ISCGEM	ISCGEM	ISCGEM	Automatic
15	02-01-1965	05:27:06	-18.697	-177.864	Earthquake	482.9			5.6	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
16	02-02-1965	15:56:51	37.523	73.251	Earthquake	15			6	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
17	02-04-1965	03:25:00	-51.84	139.741	Earthquake	10			6.1	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
18	02-04-1965	05:01:22	51.251	178.715	Earthquake	30.3			8.7	MW							OFFICIAL1	OFFICIAL	ISCGEM	OFFICIAL	Automatic
19	02-04-1965	06:04:59	51.639	175.055	Earthquake	30			6	MW							ISCGEMSU	ISCGEMSU	ISCGEM	ISCGEM	Automatic
20	02-04-1965	06:37:06	52.528	172.007	Earthquake	25			5.7	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
21	02-04-1965	06:39:32	51.626	175.746	Earthquake	25			5.8	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
22	02-04-1965	07:11:23	51.037	177.848	Earthquake	25			5.9	MW							ISCGEMSU	ISCGEMSU	ISCGEM	ISCGEM	Automatic
23	02-04-1965	07:14:59	51.73	173.975	Earthquake	20			5.9	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
24	02-04-1965	07:23:12	51.775	173.058	Earthquake	10			5.7	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
25	02-04-1965	07:43:43	52.611	172.588	Earthquake	24			5.7	MW							ISCGEMSU	ISCGEMSU	ISCGEM	ISCGEM	Automatic
26	02-04-1965	08:06:17	51.831	174.368	Earthquake	31.8			5.7	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic
27	02-04-1965	08:33:41	51.948	173.969	Earthquake	20			5.6	MW							ISCGEM85	ISCGEM	ISCGEM	ISCGEM	Automatic

FEATURE EXPLORATION

Once the dataset is acquired, it's essential to dive into feature exploration. This phase involves:

1. **Data Inspection:**

Carefully examining the dataset to understand its structure, data types, and any missing values.

2. **Statistical Analysis:**

Calculating summary statistics, including mean, median, standard deviation, and quartiles for each feature. This will help us identify outliers and understand the data's distribution.

3. **Correlation Analysis:**

Investigating the correlations between features, especially between earthquake magnitude and other variables. Identifying highly correlated features can be beneficial for model development.

VISUALIZATION

Visualization plays a crucial role in gaining insights from the data. In this phase:

1. **World Map Visualization:**

Creating a world map visualization to display the geographical distribution of earthquakes. This can help identify earthquake-prone regions and patterns.

2. Time Series Plots:

Visualizing the earthquake data over time to detect any temporal trends or seasonality.

DATA SPLITTING

To evaluate our model effectively, we need to split the dataset into two subsets:

1. Training Set:

This set will be used to train our neural network model. It should contain a significant portion of the data, ensuring that the model learns from a diverse range of examples.

2. Test Set:

The test set is crucial for evaluating the model's performance. It should be separate from the training data and used to assess how well the model generalizes to unseen earthquake data.

MODEL DEVELOPMENT

In this phase, we focus on building the earthquake prediction model using a neural network. Key steps include:

1. Data Pre processing:

Preparing the data for model input, which may involve normalization, scaling, or encoding categorical variables.

2. Neural Network Architecture:

Designing the architecture of the neural network. This includes defining the number of layers, neurons, activation functions, and loss functions.

3. Model Training:

Training the neural network on the training set using appropriate optimization techniques, such as stochastic gradient descent (SGD) or Adam.

TRAINING AND EVALUATION

The final phase involves training the model and evaluating its performance:

1. Model Training:

Fit the neural network to the training data and monitor its convergence. Adjust hyper parameters as needed to optimize performance.

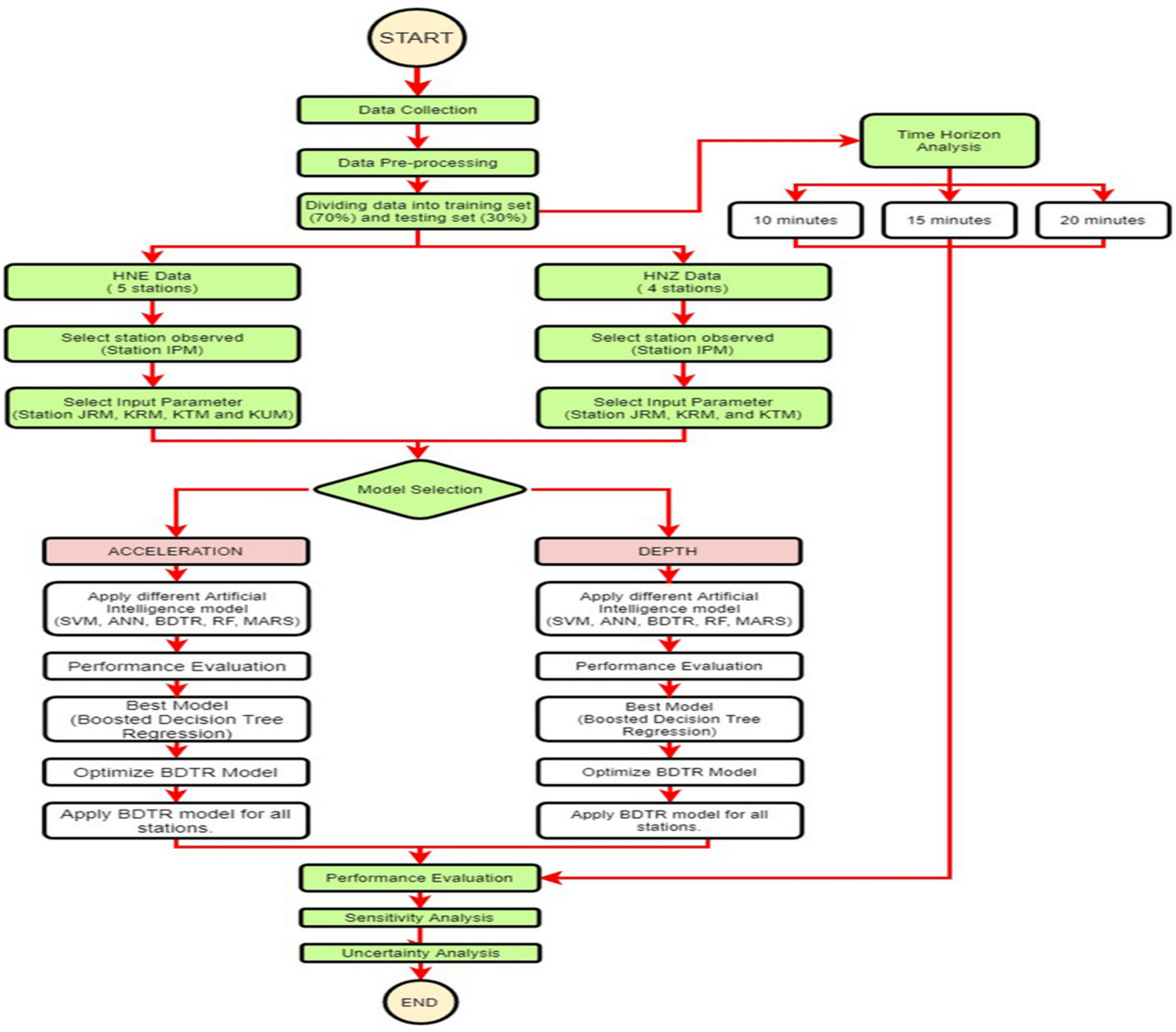
2. Model Evaluation:

Assess the model's performance on the test set using appropriate evaluation metrics, such as mean squared error (MSE) or root mean squared error (RMSE).

3. Fine-Tuning:

If the model's performance is not satisfactory, consider fine-tuning the architecture or exploring advanced techniques like hyper parameter tuning or different neural network architectures.

Flow Chart:


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

PHASE – 1

Importing the Dataset and Perform data Cleaning & Data Analysis.

INTRODUCTION

In the realm of Earthquake Prediction using Machine Learning, the initial steps of importing the dataset and conducting meticulous data cleaning are pivotal. This project begins by acquiring seismic data, a critical precursor to predictive modeling. Rigorous data cleaning techniques are then employed to ensure the dataset's integrity and reliability. Subsequently, through advanced data analysis, we aim to unveil patterns and insights crucial for developing a robust ML model capable of predicting seismic activities. This introduction sets the stage for a comprehensive exploration of earthquake prediction, emphasizing the foundational role of data import and cleaning in the ML-driven analytical process.

WORKSPACE

We’ve worked on Google Colab for the intricate task of data cleaning and analysis in Earthquake Prediction using Python. Google Colab served as a powerful and accessible platform.

Leveraging the collaborative and cloud-based features of Google Colab facilitated seamless collaboration and efficient processing of seismic datasets. The platform's integration with popular Python libraries streamlined coding and analysis workflows, enhancing productivity. For a detailed walkthrough of the data cleaning and analysis process, refer to the Notebook on Google Colab, Click Here.....

IMPORTING THE DATASET

Importing the dataset is the foundational step in our Earthquake Prediction using ML project. We seamlessly fetched seismic data from reliable sources, ensuring its accuracy and relevance. Leveraging the versatility of Python, we employed libraries like Pandas to efficiently read and organize the dataset for subsequent analysis. The chosen dataset encompasses essential seismic parameters, forming the basis for training and validating our machine learning model. The streamlined import process lays the groundwork for a comprehensive exploration into earthquake prediction methodologies.

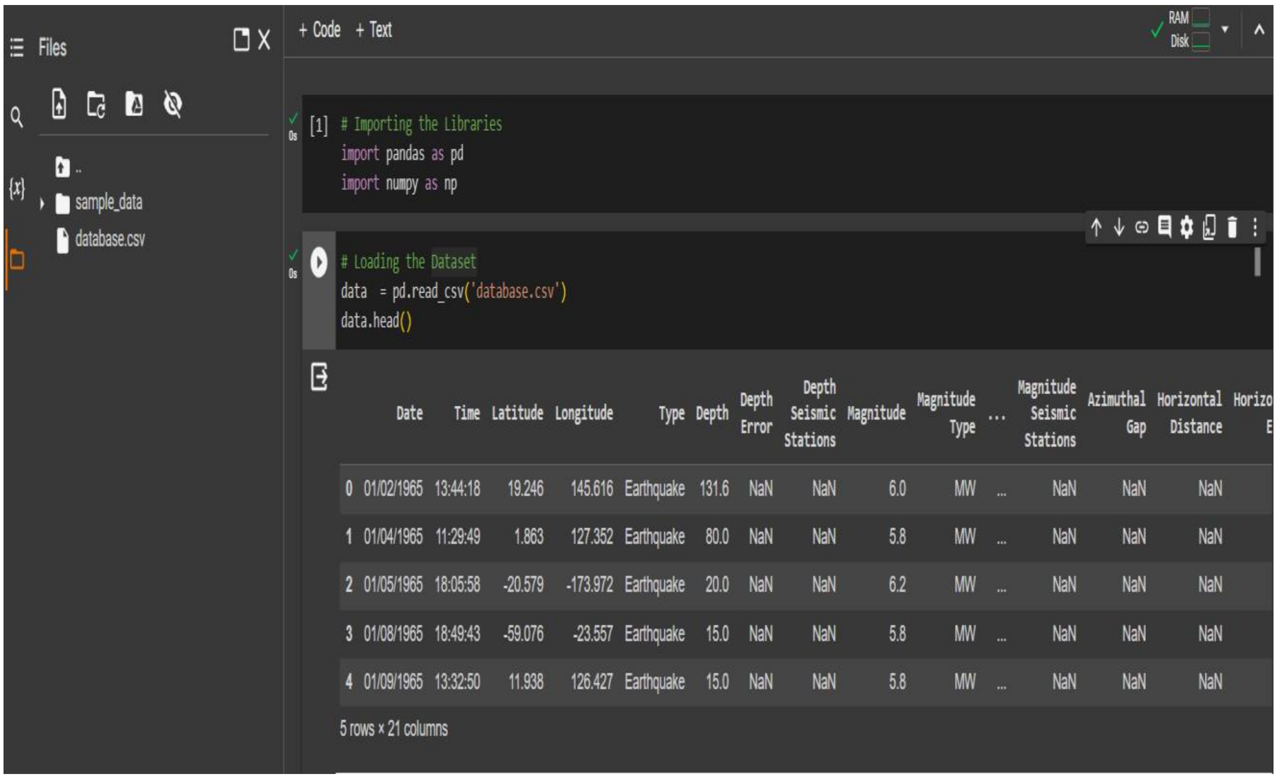
PROGRAM :

Original file is located at

https://colab.research.google.com/drive/1IHe_-veRrUX6y4RuHVhBIvX-_oGMLYa_?usp=sharing

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



# Loading the Dataset


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

OUTPUT :



```
[1] # Importing the Libraries
    import pandas as pd
    import numpy as np

# Loading the Dataset
data  = pd.read_csv('database.csv')
data.head()
```

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizo E
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	

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()

df = pd.DataFrame(data)

# *Checking Categorical and Numerical Columns*

# *Categorical columns*

cat_col = [col for col in df.columns if df[col].dtype == 'object']

print('Categorical columns :',cat_col)

# *Numerical columns*

num_col = [col for col in df.columns if df[col].dtype != 'object']

print('Numerical columns :',num_col)

# *Checking total number of Values in Categorical Columns*

df[cat_col].nunique()

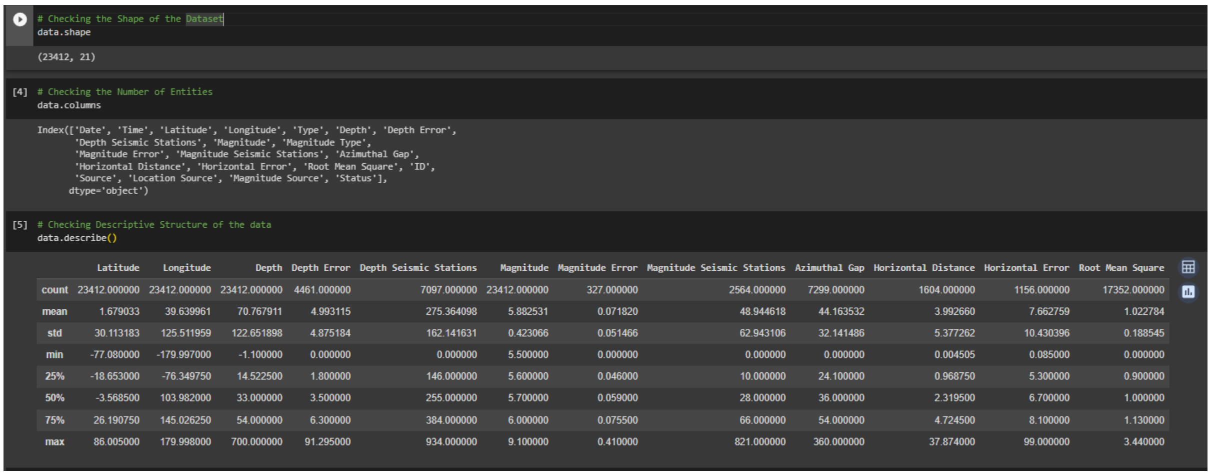
# Checking total number of Values in Numerical Columns

df[num_col].nunique()

# Checking the Missing Values Percentage

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

OUTPUT :



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

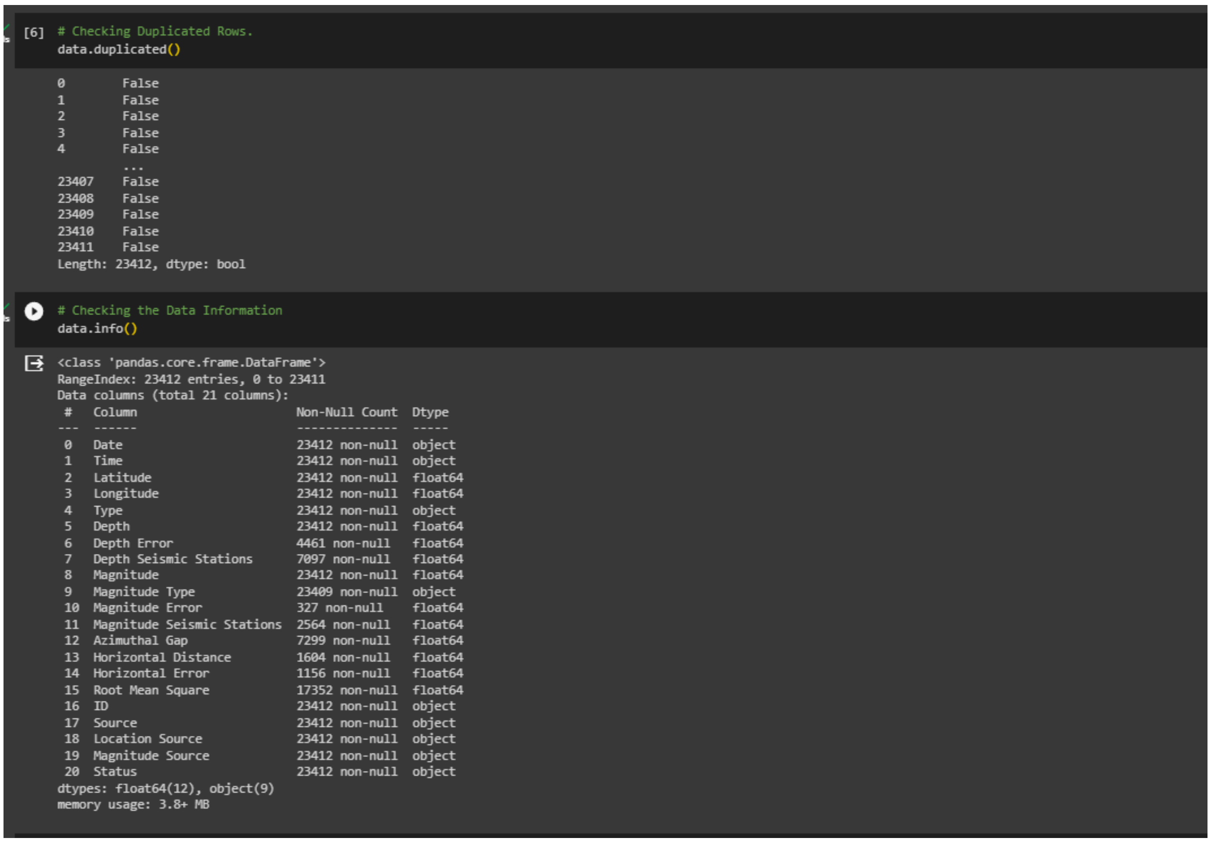
```
(23412, 21)
```

```
[4] # Checking the Number of Entities
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')
```

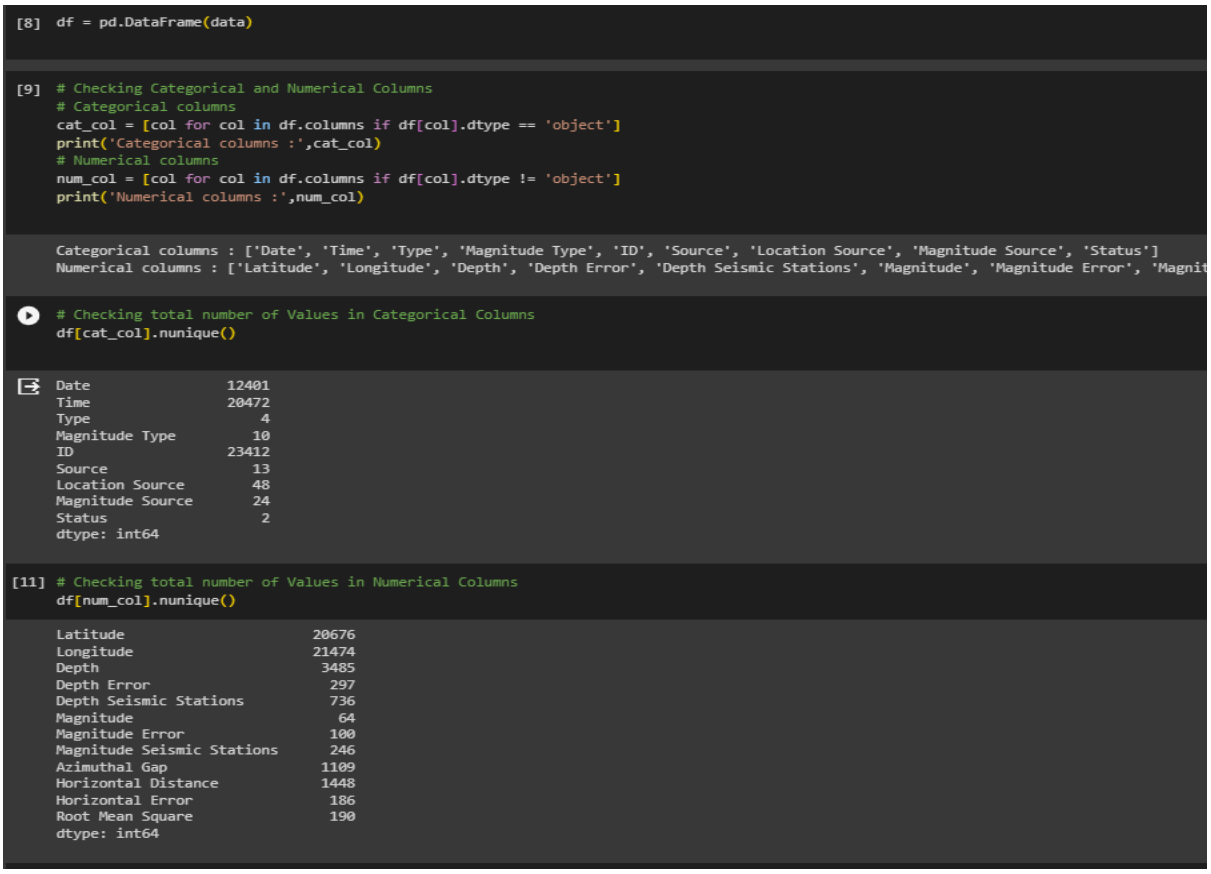
```
[5] # Checking Descriptive Structure of the data
data.describe()
```

	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



```
[6] # Checking Duplicated Rows.
    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
```

```
    # Checking the Data Information
    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
```



```
[8] df = pd.DataFrame(data)
```

```
[9] # Checking Categorical and Numerical Columns
    # Categorical columns
    cat_col = [col for col in df.columns if df[col].dtype == 'object']
    print('Categorical columns :',cat_col)
    # Numerical columns
    num_col = [col for col in df.columns if df[col].dtype != 'object']
    print('Numerical columns :',num_col)

    Categorical columns : ['Date', 'Time', 'Type', 'Magnitude Type', 'ID', 'Source', 'Location Source', 'Magnitude Source', 'Status']
    Numerical columns : ['Latitude', 'Longitude', 'Depth', 'Depth Error', 'Depth Seismic Stations', 'Magnitude', 'Magnitude Error', 'Magnit
```

```
    # Checking total number of Values in Categorical Columns
    df[cat_col].nunique()

    Date                12401
    Time                20472
    Type                    4
    Magnitude Type         10
    ID                  23412
    Source                 13
    Location Source        48
    Magnitude Source       24
    Status                  2
    dtype: int64
```

```
[11] # Checking total number of Values in Numerical Columns
     df[num_col].nunique()

    Latitude                      20676
    Longitude                     21474
    Depth                          3485
    Depth Error                     297
    Depth Seismic Stations          736
    Magnitude                        64
    Magnitude Error                 100
    Magnitude Seismic Stations      246
    Azimuthal Gap                  1109
    Horizontal Distance            1448
    Horizontal Error                186
    Root Mean Square                190
    dtype: int64
```

FEATURE ENGINEERING

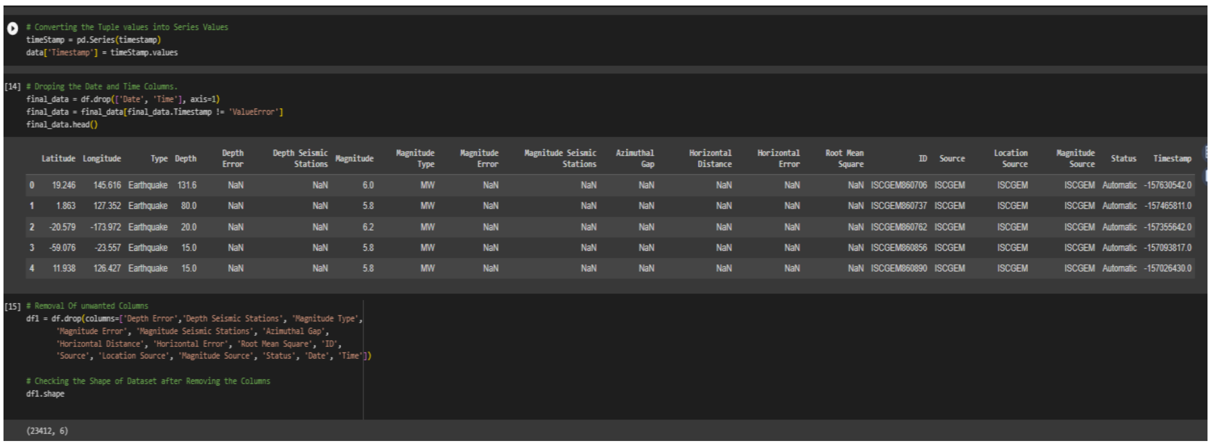
Feature engineering is a critical aspect of machine learning where raw data is transformed or new features are created to enhance model performance. It involves techniques like polynomial expansion, interaction terms, and domain-specific transformations to extract meaningful information. Dimensionality 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
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values

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

OUTPUT :



```
# Converting the Tuple values into Series Values
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

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

	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

```
[15] # Removal Of unwanted Columns
     df1 = df.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
     df1.shape

     (23412, 6)
```

DATA CLEANING

PROGRAM:

```
# Removal Of Unwanted Columns
df1 = df.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
df1.shape

df1.head(10)

# Checking Columns
df1.columns

# Checking the Missing Values Percentage
round((df1.isnull().sum()/df1.shape[0])*100,2)
```

```
# Checking the Data Information After droping
the Unwanted Columns
df1.info()

# Checking the Descriptive Structure of the
Data after the removal of Unwanted Columns
df1.describe()

# Checking Categorical and Numerical
Columns
# Categorical columns
cat_col = [col for col in df1.columns if
df1[col].dtype == 'object']
print('Categorical columns :',cat_col)

# Numerical columns
num_col = [col for col in df1.columns if
df1[col].dtype != 'object']
print('Numerical columns :',num_col)

# Checking total number of Values in
Categorical Columns
df1[cat_col].nunique()
```

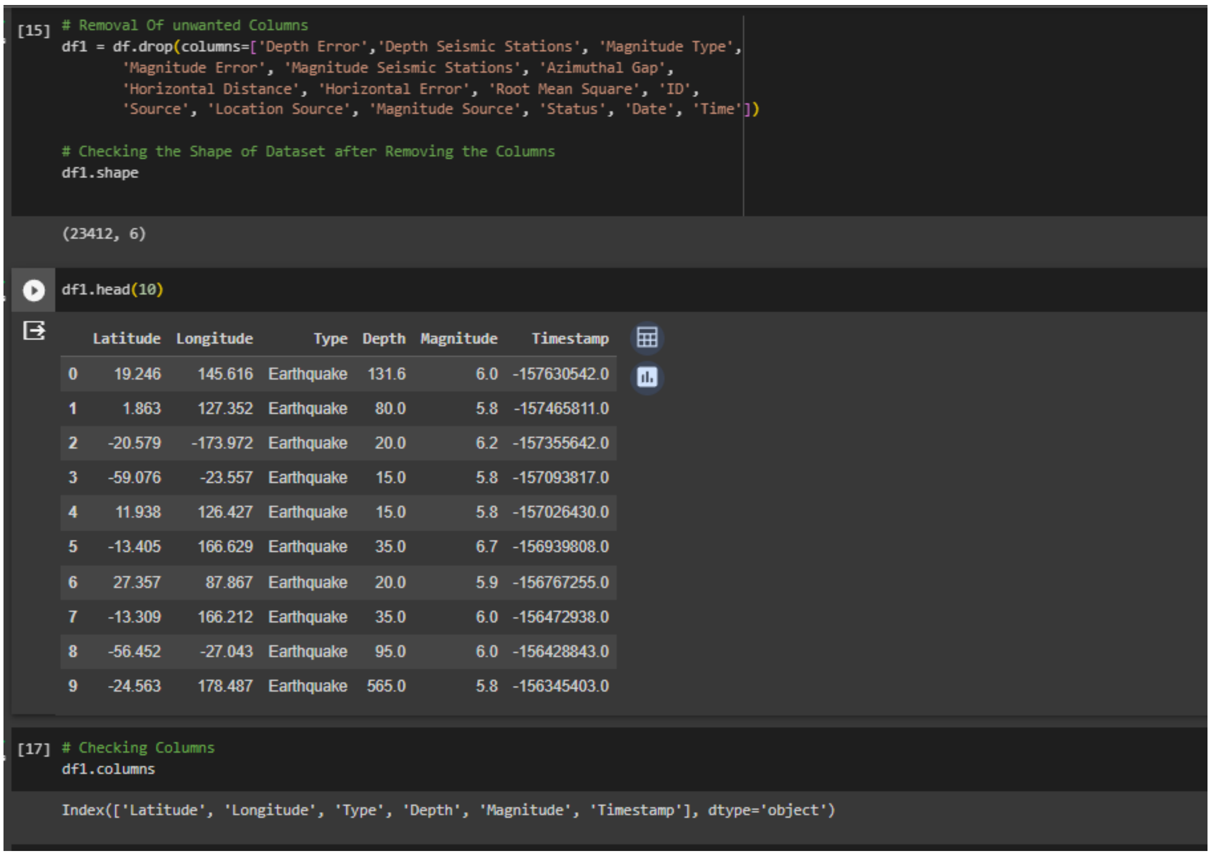
# Checking total number of Values in Numerical Columns

df[num_col].nunique()

# Let's check the null values again

df1.isnull().sum()

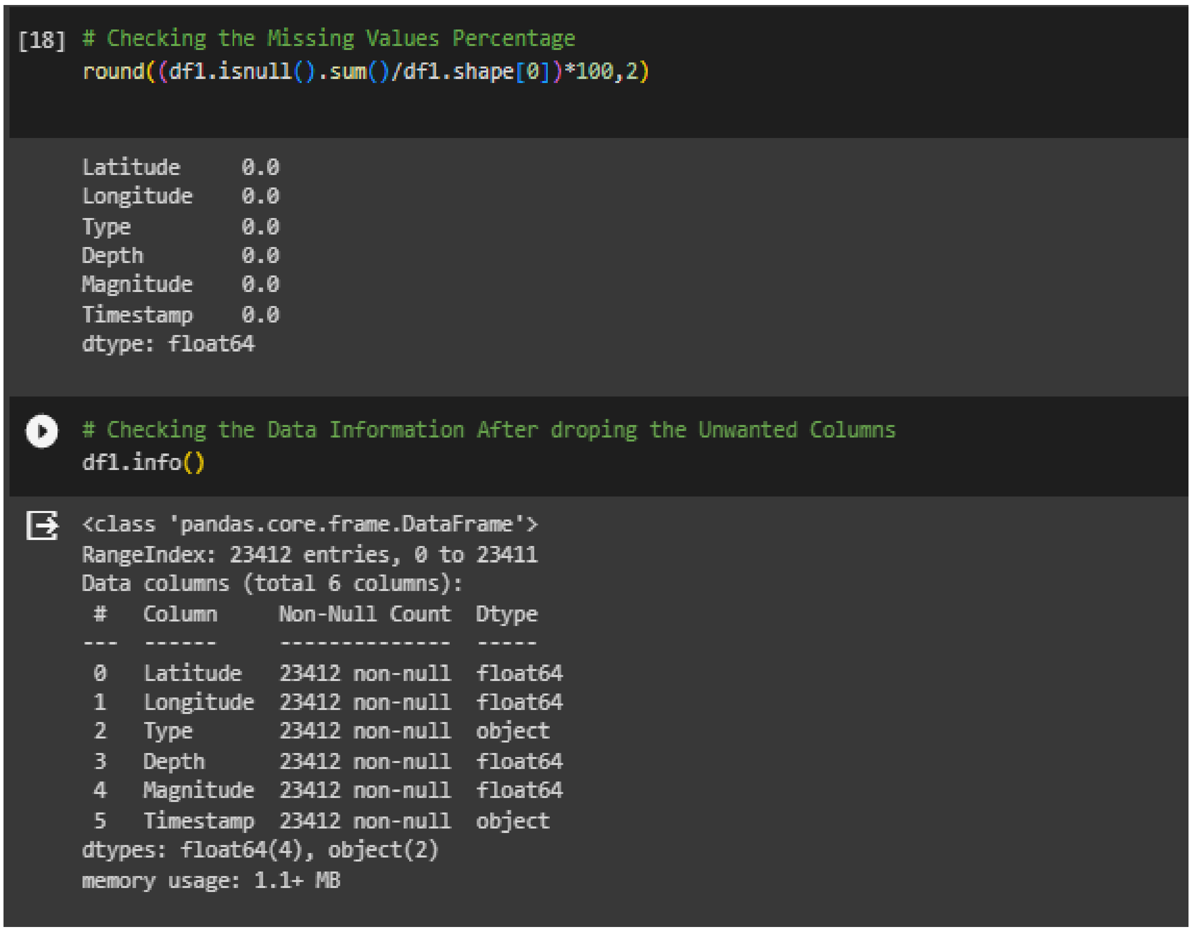
OUTPUT:



```
[15] # Removal Of unwanted Columns
     df1 = df.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
     df1.shape

     (23412, 6)
```

```
df1.head(10)

   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
5   -13.405    166.629  Earthquake   35.0        6.7  -156939808.0
6    27.357     87.867  Earthquake   20.0        5.9  -156767255.0
7   -13.309    166.212  Earthquake   35.0        6.0  -156472938.0
8   -56.452    -27.043  Earthquake   95.0        6.0  -156428843.0
9   -24.563    178.487  Earthquake  565.0        5.8  -156345403.0
```

```
[17] # Checking Columns
     df1.columns

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

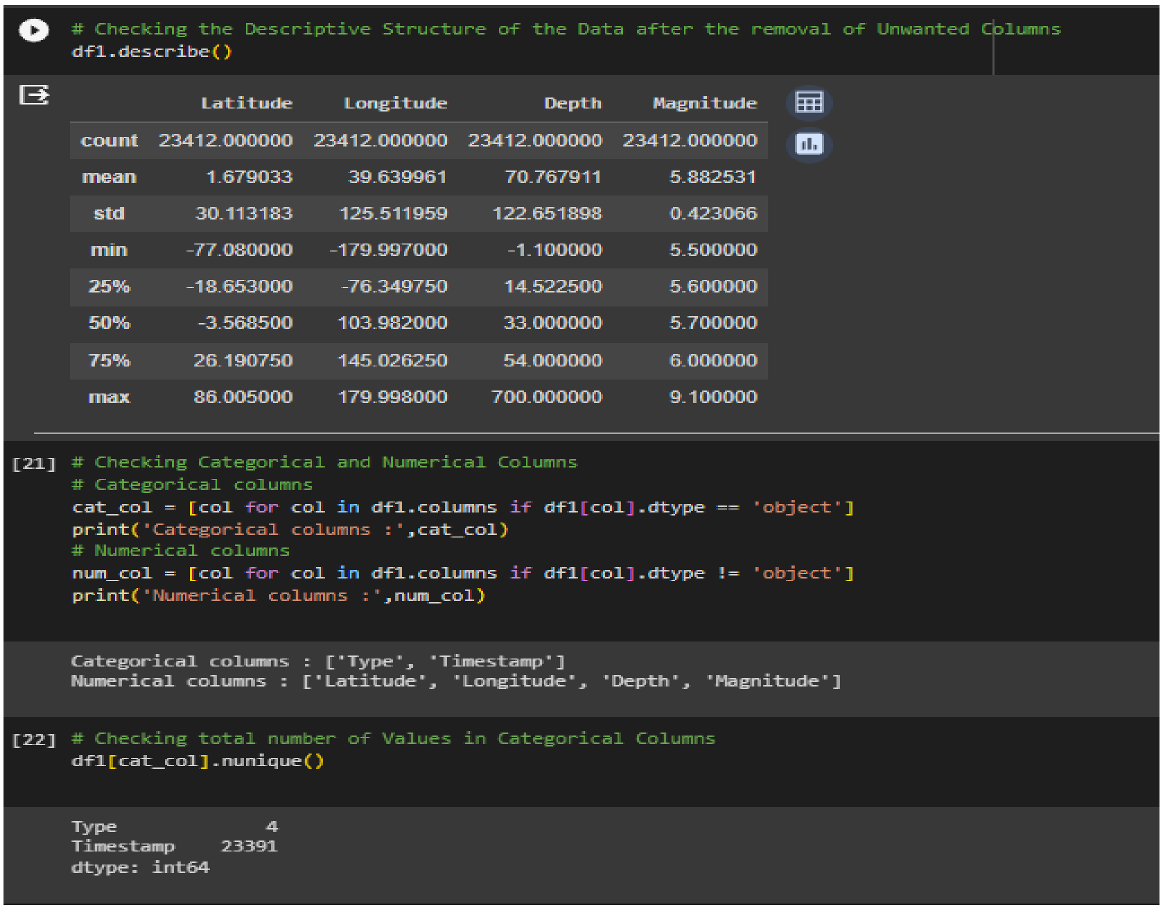


```
[18] # Checking the Missing Values Percentage
     round((df1.isnull().sum()/df1.shape[0])*100,2)
```

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

```
# Checking the Data Information After droping the Unwanted Columns
df1.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
```



```
# Checking the Descriptive Structure of the Data after the removal of Unwanted Columns
df1.describe()
```

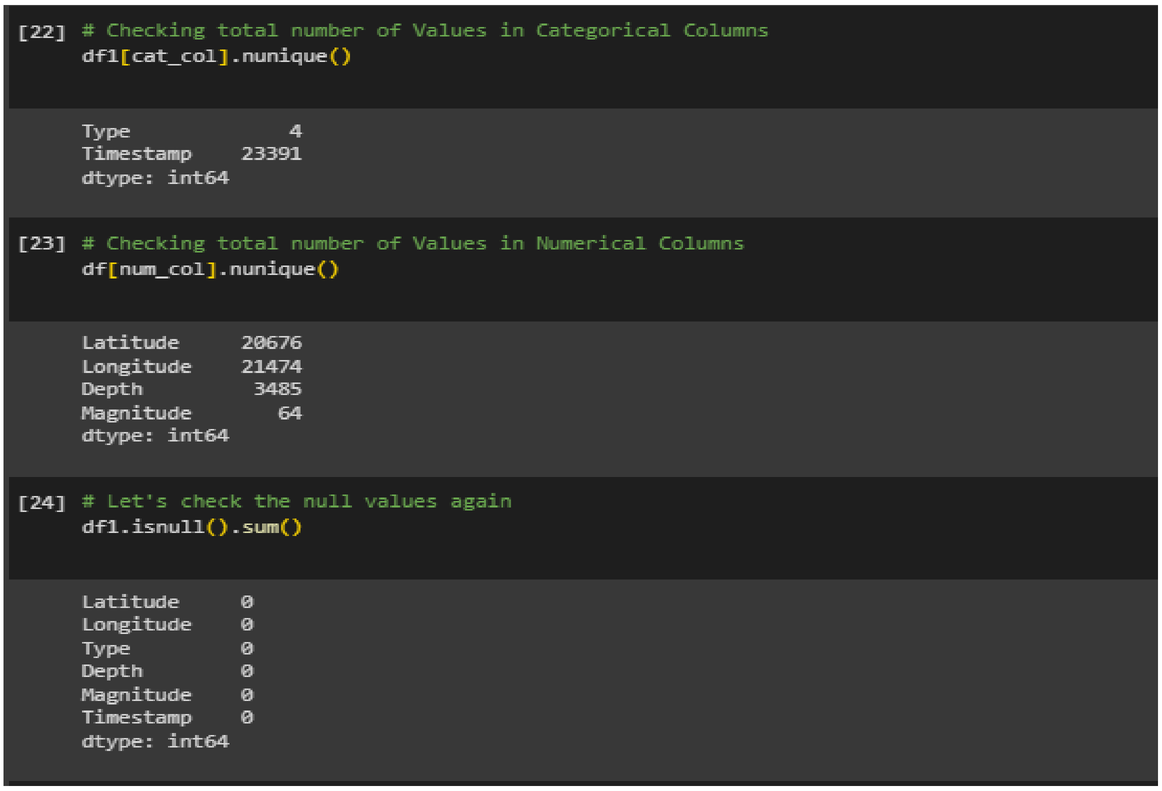
	Latitude	Longitude	Depth	Magnitude
count	23412.000000	23412.000000	23412.000000	23412.000000
mean	1.679033	39.639961	70.767911	5.882531
std	30.113183	125.511959	122.651898	0.423066
min	-77.080000	-179.997000	-1.100000	5.500000
25%	-18.653000	-76.349750	14.522500	5.600000
50%	-3.568500	103.982000	33.000000	5.700000
75%	26.190750	145.026250	54.000000	6.000000
max	86.005000	179.998000	700.000000	9.100000

```
[21] # Checking Categorical and Numerical Columns
     # Categorical columns
     cat_col = [col for col in df1.columns if df1[col].dtype == 'object']
     print('Categorical columns :',cat_col)
     # Numerical columns
     num_col = [col for col in df1.columns if df1[col].dtype != 'object']
     print('Numerical columns :',num_col)
```

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

```
[22] # Checking total number of Values in Categorical Columns
     df1[cat_col].nunique()
```

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



```
[22] # Checking total number of Values in Categorical Columns
     df1[cat_col].nunique()

     Type             4
     Timestamp    23391
     dtype: int64

[23] # Checking total number of Values in Numerical Columns
     df[num_col].nunique()

     Latitude     20676
     Longitude    21474
     Depth         3485
     Magnitude       64
     dtype: int64

[24] # Let's check the null values again
     df1.isnull().sum()

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

CONCLUSION

The process of earthquake prediction using machine learning involves meticulous data cleaning to ensure dataset reliability. Data importing combines seismic, geological, and environmental data for a comprehensive analysis. Feature engineering enhances the dataset, optimizing models for pattern recognition. Iterative refinement based on model performance fosters nuanced earthquake prediction insights. Overall, this approach, encompassing data cleaning, importing, and analysis, advances our ability to develop accurate machine learning models for mitigating the impact of seismic events.

PHASE – 2

Development Part – 1

Begin building the earthquake prediction model by loading and preprocessing the dataset

INTRODUCTION

This documentation is a guide to the preprocessing steps essential for constructing an earthquake prediction model. It covers data loading, cleaning, and exploratory analysis, providing transparency in the model-building process. The document emphasizes the rationale behind decisions, addressing challenges and nuances encountered. With a structured approach, it guides readers through feature engineering, transformations, and the crucial train-test split. Code snippets, visualizations, and examples facilitate understanding and reproducibility. Tailored for a diverse audience, from data scientists to enthusiasts, it highlights the significance of meticulous preprocessing in seismic prediction. The documentation's scope extends beyond replication, aiming to deepen comprehension of machine learning methodologies in earthquake forecasting. In 10 lines, it invites readers to explore the intricacies of preparing data for the vital task of earthquake prediction.

DATA LOADING

Data loading is the inaugural step in machine learning, essential for acquiring datasets that fuel model development. Identifying the data source, whether it be CSV files, databases, or APIs, dictates the loading approach. By integrating libraries like pandas, the process is streamlined, allowing users to efficiently manipulate and analyze data. The accompanying code snippets in the documentation showcase the programmatic loading of datasets, ensuring accessibility and ease of understanding. Versatility is emphasized, addressing various data formats such as CSV, Excel, JSON, or databases, providing adaptability to diverse structures. Robust data loading involves error handling, anticipating and managing issues like missing values or corrupted data. The documentation also offers a glimpse of the loaded data, aiding users in comprehending its structure and content. Early data cleaning initiatives may be embedded during loading, tackling issues like missing values or inconsistent formatting. Emphasizing reproducibility, the documentation guides users on how to load the data with specific parameters for consistent results. Ultimately, data loading establishes the groundwork, connecting the acquired datasets to the subsequent stages of model training in the machine learning workflow.

PREPROCESSING

Preprocessing is a pivotal stage in machine learning workflows, acting as the foundation for robust model development. It encompasses several critical steps, beginning with the loading of raw data from diverse sources, such as CSV files or databases. The process involves thorough data cleaning, addressing issues like missing values, outliers, and duplicates to ensure the quality and reliability of the dataset. Exploratory Data Analysis (EDA) is employed to gain insights into the dataset's distribution, relationships, and potential patterns, guiding subsequent preprocessing decisions. Feature engineering follows, where new features are created or existing ones are transformed to enhance the model's understanding of underlying patterns. Data normalization and scaling are crucial for ensuring that features are on a consistent scale, preventing any particular feature from dominating the model training process. Categorical variables are appropriately encoded to numerical formats, facilitating their integration into machine learning models. The dataset is then split into training and testing sets to assess the model's generalization performance accurately. Throughout this process, documentation and inline comments are incorporated, ensuring transparency and reproducibility in the preprocessing pipeline. This meticulous preprocessing paves the way for effective model training, contributing significantly to the model's overall predictive accuracy.

PROGRAM :

```
# Importing necessary 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

# Reading the dataset from the specified location

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

# Displaying the loaded dataset

data

# Providing information about the dataset, including data types and missing values

data.info()

# Dropping the 'ID' column from the dataset

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

# Identifying and dropping columns with more than 66% missing values

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

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

# Displaying the count of missing values in each column

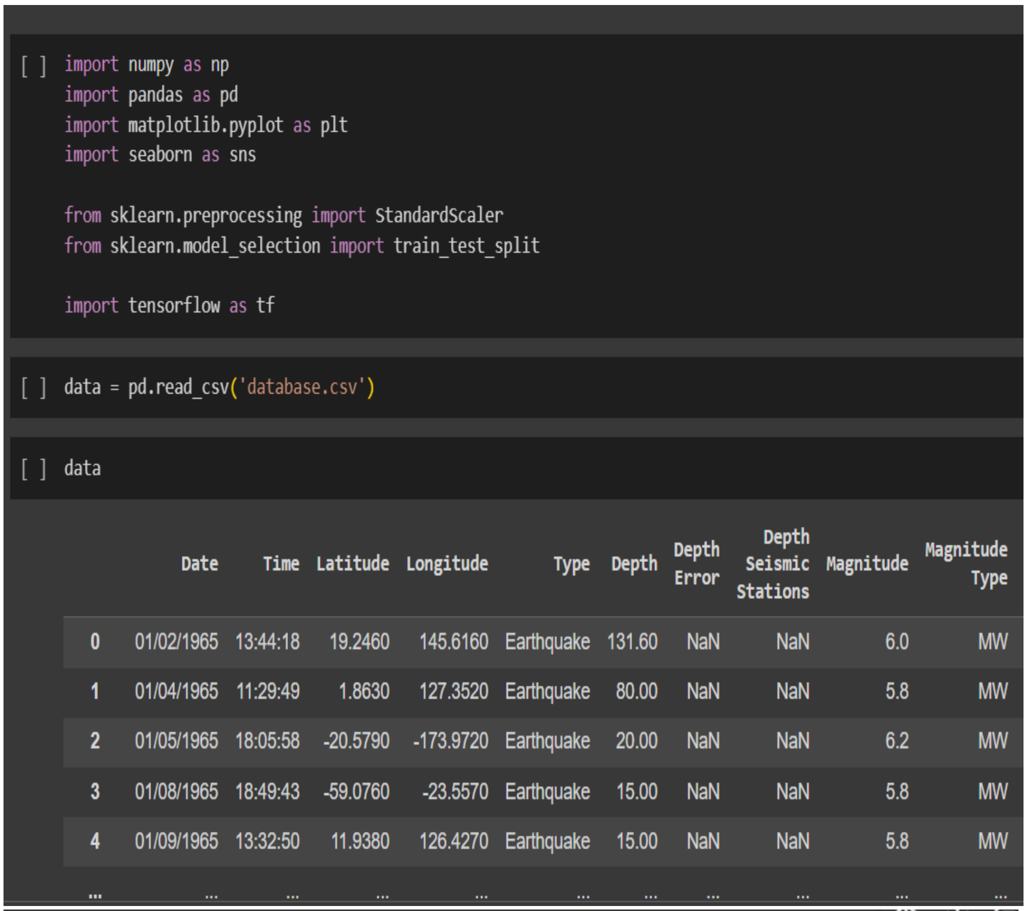
data.isna().sum()

```
# Filling missing values in the 'Root Mean Square'
column with the mean value

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


# Dropping rows with any remaining missing
values and resetting the index

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


# Confirming there are no more missing values in
the dataset

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


# Feature Engineering: 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:])


# Converting 'Month' to integer type

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


# Handling invalid 'Year' entries and converting to
integer type

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' and displaying the
modified dataset

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

data


# Displaying the shape and columns of the final
dataset

data.shape

data.columns


# Selecting relevant columns and displaying the
first few rows of the modified dataset

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

data.head()


# 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:
        # Handling cases where timestamp conversion
fails
        timestamp.append('ValueError')

# Creating a new 'Timestamp' column in the
dataset
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values

# Creating the final dataset by dropping 'Date' and
'Time' columns and removing rows with invalid
timestamps
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp !=
'ValueError']
final_data.head()
```

OUTPUT :

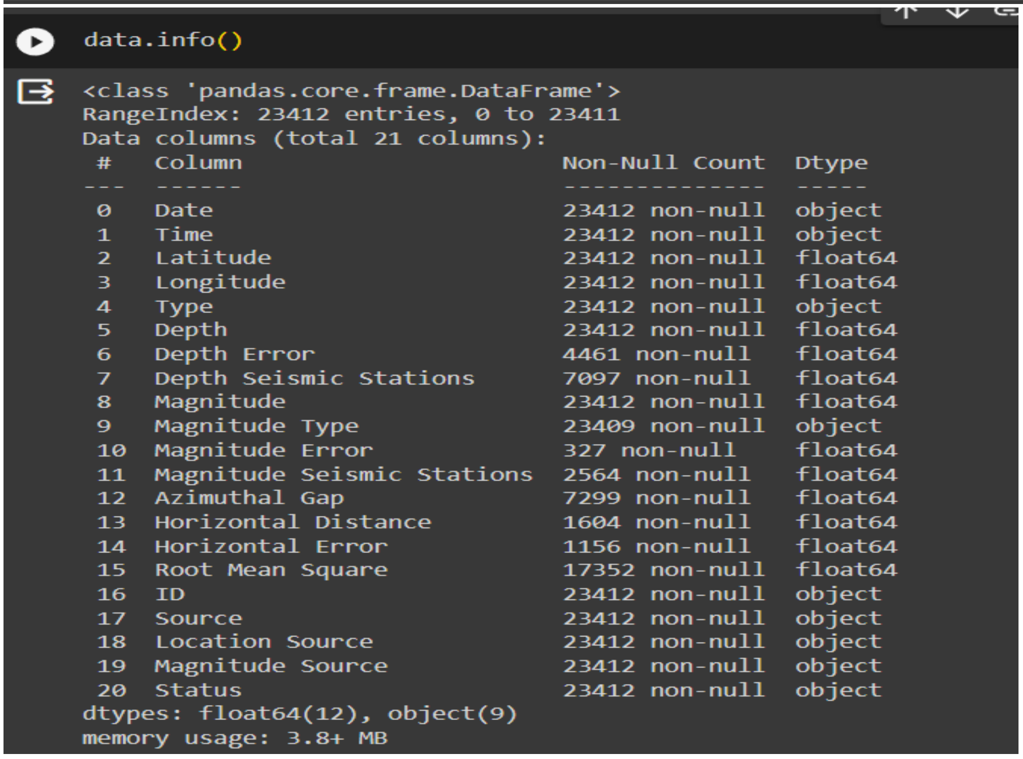


```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split

    import tensorflow as tf
```

```
[ ] data = pd.read_csv('database.csv')
```

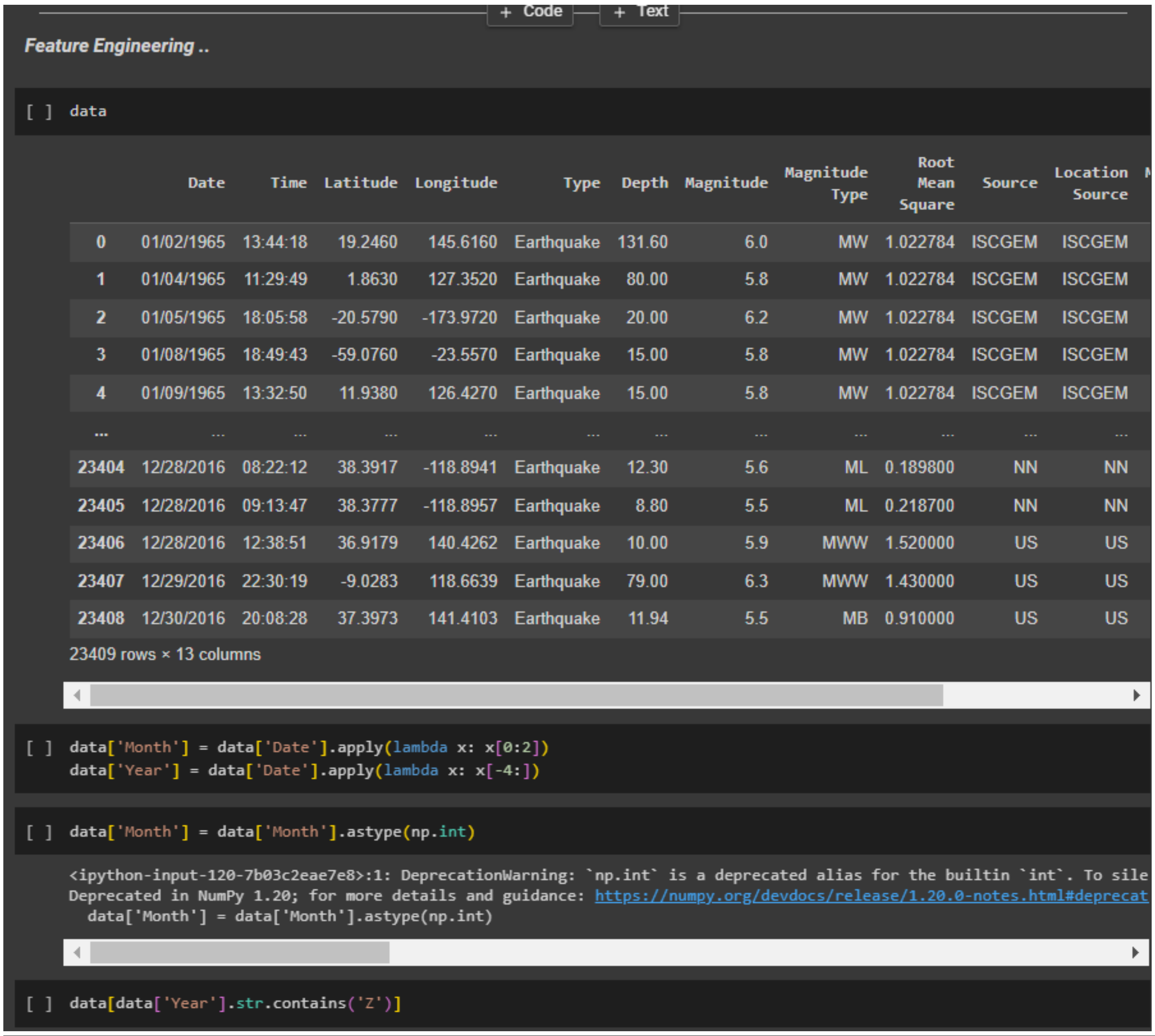
```
[ ] data
```

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW
...	...	...	...	...	...	...	...	...	...	...



```
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 ..

```
[ ] data
```

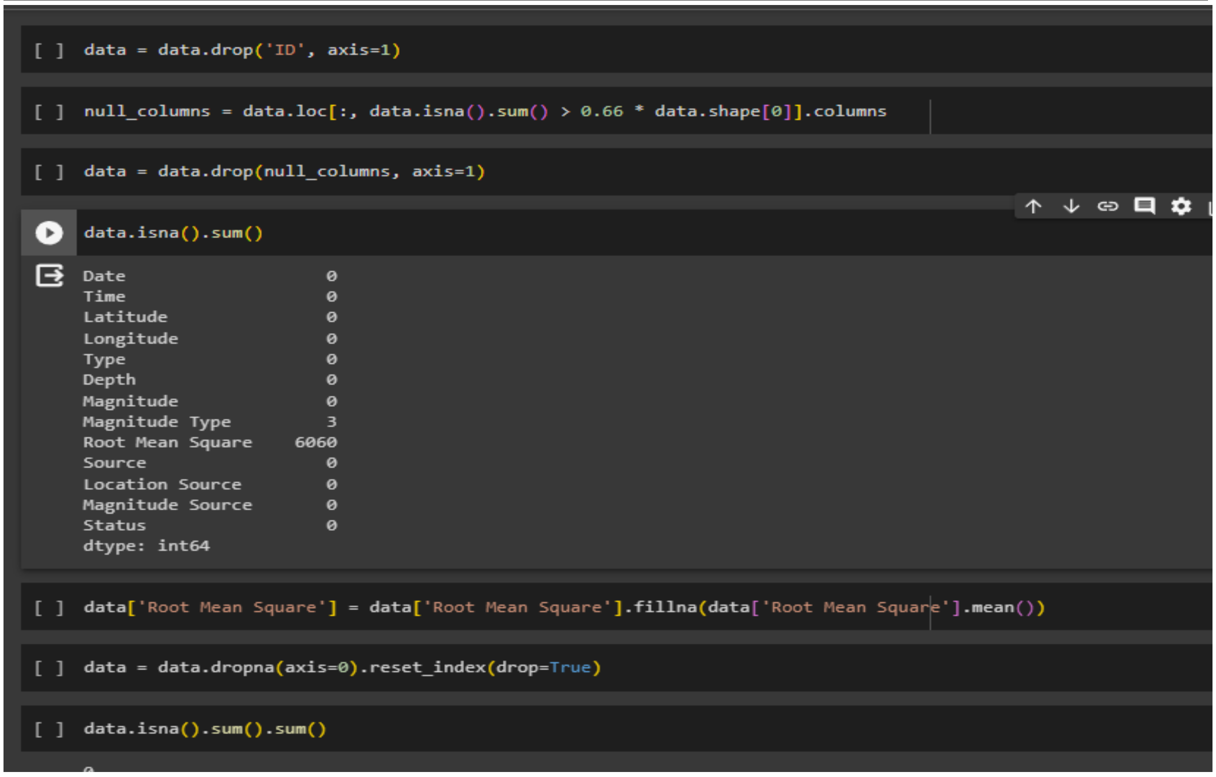
```
         Date      Time  Latitude  Longitude        Type  Depth  Magnitude  Magnitude      Root  Source  Location
                                                                                 Type      Mean            Source
                                                                                         Square
0      01/02/1965  13:44:18   19.2460   145.6160  Earthquake  131.60   6.0   MW   1.022784  ISCGEM  ISCGEM
1      01/04/1965  11:29:49    1.8630   127.3520  Earthquake   80.00   5.8   MW   1.022784  ISCGEM  ISCGEM
2      01/05/1965  18:05:58  -20.5790  -173.9720  Earthquake   20.00   6.2   MW   1.022784  ISCGEM  ISCGEM
3      01/08/1965  18:49:43  -59.0760   -23.5570  Earthquake   15.00   5.8   MW   1.022784  ISCGEM  ISCGEM
4      01/09/1965  13:32:50   11.9380   126.4270  Earthquake   15.00   5.8   MW   1.022784  ISCGEM  ISCGEM
...         ...       ...       ...        ...         ...     ...   ...  ...        ...     ...     ...
23404  12/28/2016  08:22:12   38.3917  -118.8941  Earthquake   12.30   5.6   ML   0.189800      NN      NN
23405  12/28/2016  09:13:47   38.3777  -118.8957  Earthquake    8.80   5.5   ML   0.218700      NN      NN
23406  12/28/2016  12:38:51   36.9179   140.4262  Earthquake   10.00   5.9  MWW   1.520000      US      US
23407  12/29/2016  22:30:19   -9.0283   118.6639  Earthquake   79.00   6.3  MWW   1.430000      US      US
23408  12/30/2016  20:08:28   37.3973   141.4103  Earthquake   11.94   5.5   MB   0.910000      US      US
23409 rows × 13 columns
```

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

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

```
<ipython-input-120-7b03c2eae7e8>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
  data['Month'] = data['Month'].astype(np.int)
```

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



```
[ ] 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)
```

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

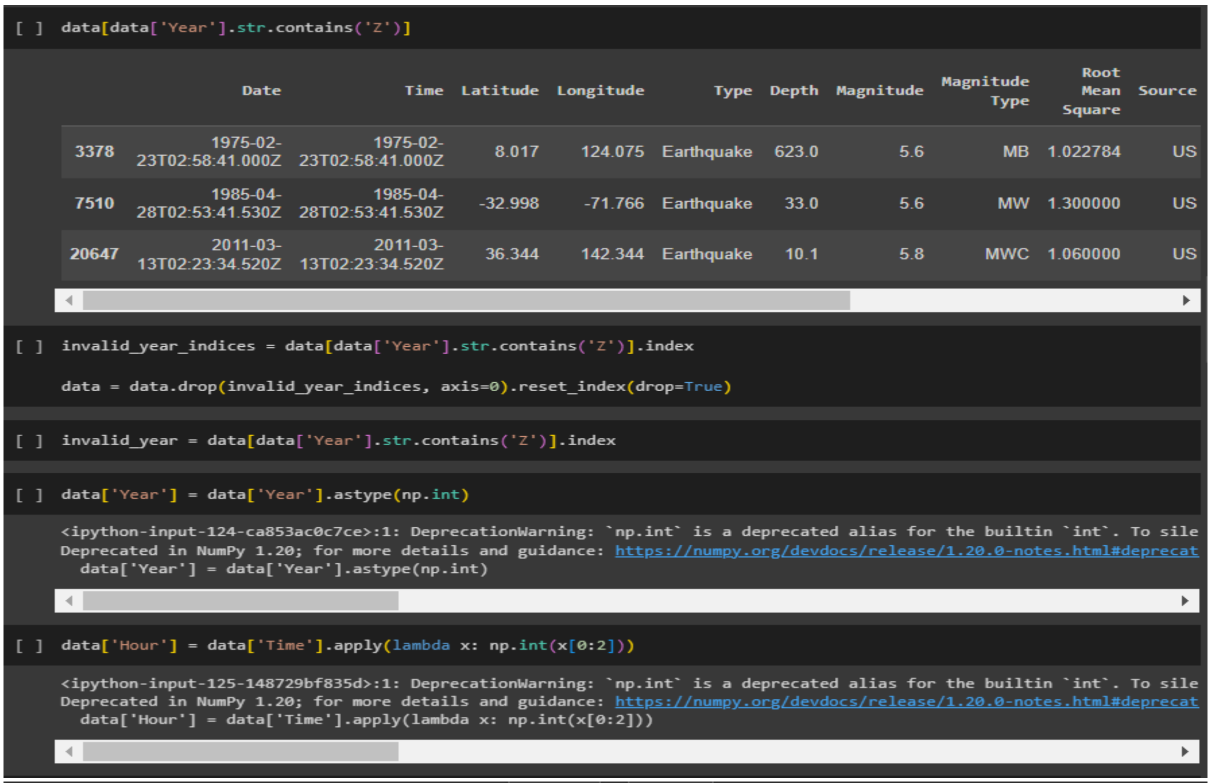
```
Date                   0
Time                   0
Latitude               0
Longitude              0
Type                   0
Depth                  0
Magnitude              0
Magnitude Type         3
Root Mean Square    6060
Source                 0
Location Source        0
Magnitude Source     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

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

```
[ ] data = data.dropna(axis=0).reset_index(drop=True)
```

```
[ ] data.isna().sum().sum()
```

```
0
```



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

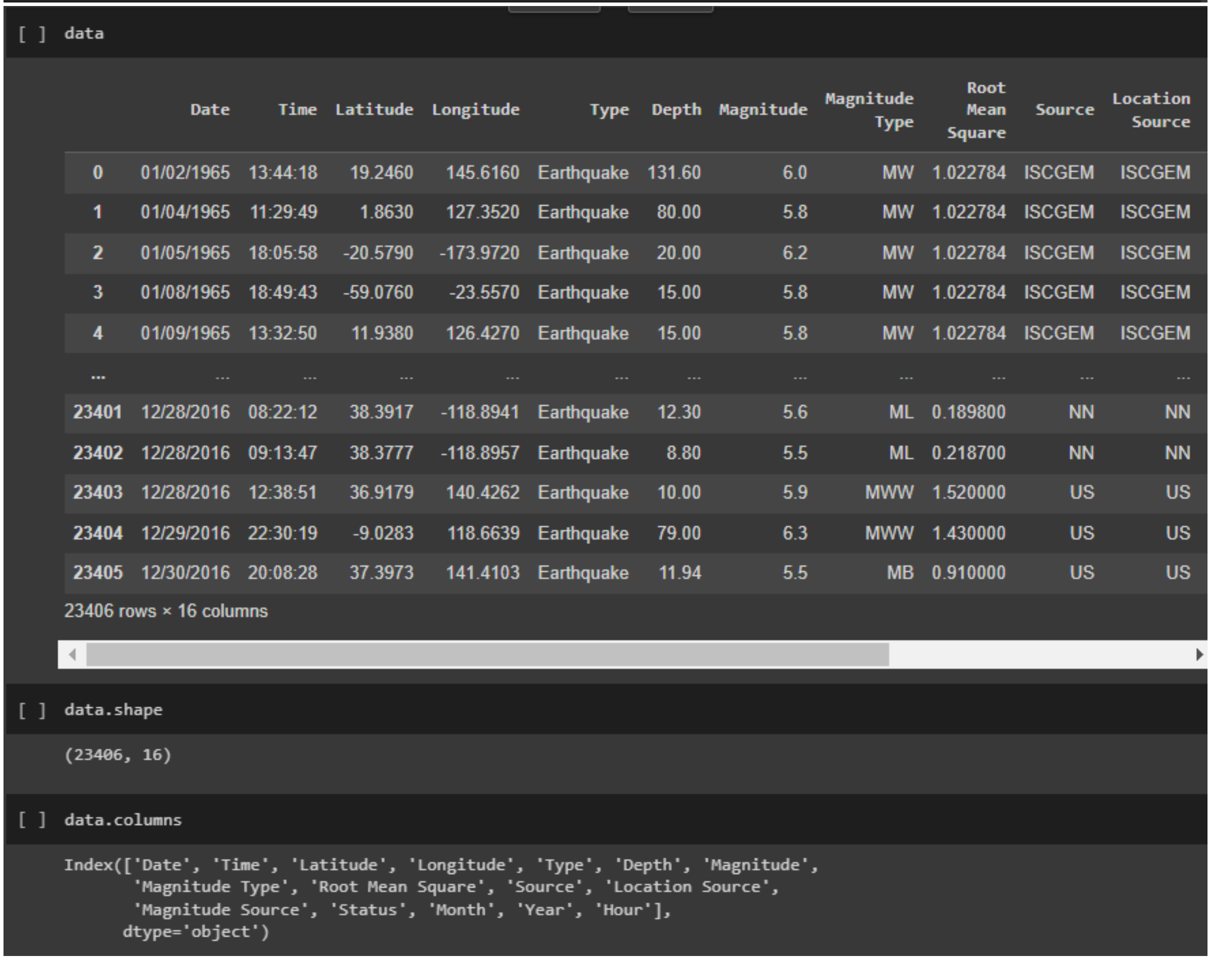
	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source
3378	1975-02-23T02:58:41.000Z	1975-02-23T02:58:41.000Z	8.017	124.075	Earthquake	623.0	5.6	MB	1.022784	US
7510	1985-04-28T02:53:41.530Z	1985-04-28T02:53:41.530Z	-32.998	-71.766	Earthquake	33.0	5.6	MW	1.300000	US
20647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

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

    data = data.drop(invalid_year_indices, axis=0).reset_index(drop=True)
```

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

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

    <ipython-input-124-ca853ac0c7ce>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
      data['Year'] = data['Year'].astype(np.int)
```

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

    <ipython-input-125-148729bf835d>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
      data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))
```



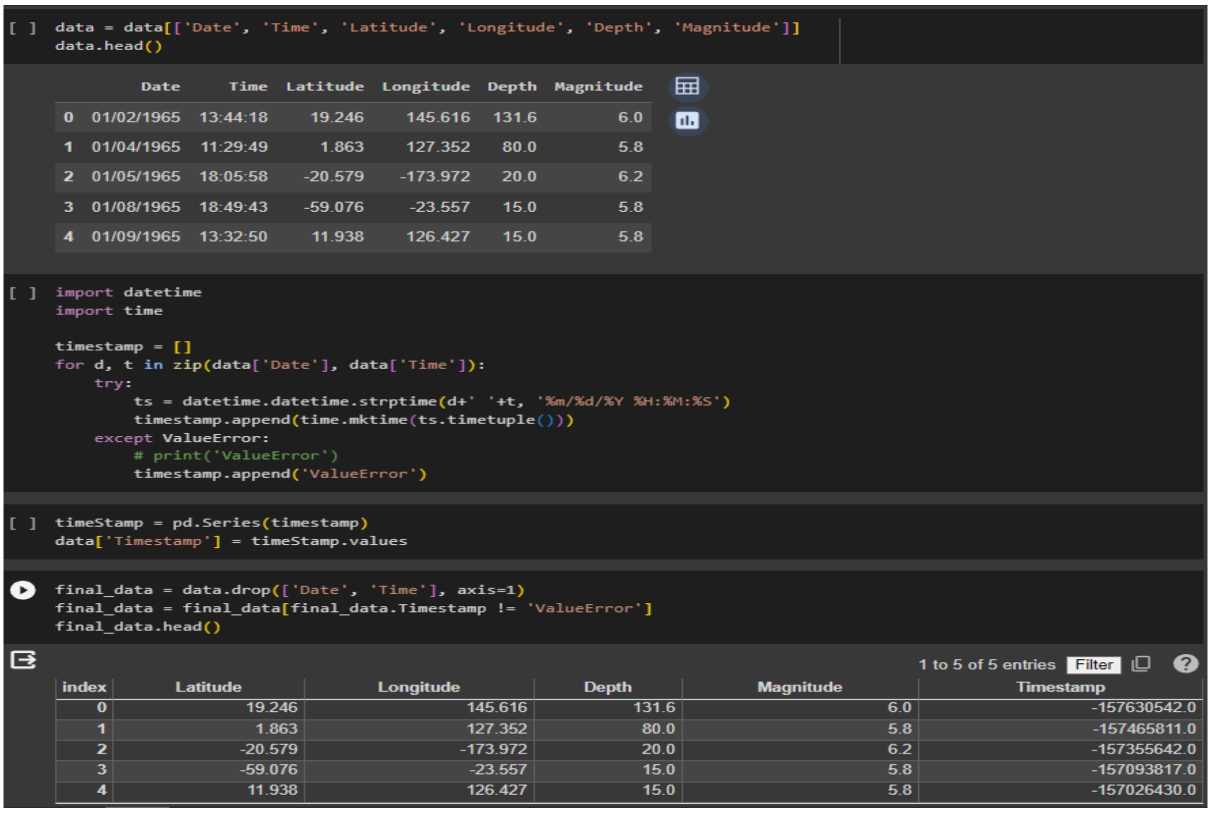
```
[ ] data
```

	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
...	...	...	...	...	...	...	...	...	...	...	...
23401	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN
23402	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN
23403	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US
23404	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US
23405	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US

23406 rows × 16 columns

```
[ ] data.shape

    (23406, 16)
```

```
[ ] data.columns

    Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Magnitude',
           'Magnitude Type', 'Root Mean Square', 'Source', 'Location Source',
           'Magnitude Source', 'Status', 'Month', 'Year', 'Hour'],
          dtype='object')
```



```
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
```

```
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')
```

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

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

```
                                                      1 to 5 of 5 entries  Filter
index  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
```

CONCLUSION

The loading and preprocessing of the earthquake dataset involved several key steps. The process began by loading the data and examining its structure, leading to the removal of the 'ID' column. Missing values were handled by dropping columns with a substantial amount of missing data and imputing the mean for the 'Root Mean Square' column. Feature engineering included extracting relevant information like 'Month', 'Year', and 'Hour' from 'Date' and 'Time'. Invalid entries in the 'Year' column were addressed. The dataset was further refined by selecting essential features and transforming 'Date' and 'Time' into a 'Timestamp' column. These steps ensure data integrity, enhance feature representation, and set the stage for constructing a robust earthquake prediction model, marking the dataset's readiness for subsequent analysis and model development.

PHASE – 3

Development Part – 2

- **Visualizing the data on the world map**
- **Splitting the dataset into Training and Testing sets.**

INTRODUCTION

In the realm of earthquake data analysis, two critical steps pave the way for robust model development: visualizing seismic events on a global scale and dividing the dataset into training and testing sets. The visualization process involves leveraging geospatial libraries like Basemap to represent earthquake occurrences worldwide, offering insights into distribution patterns and potential seismic hotspots. This spatial understanding is pivotal for informed decision-making in earthquake-prone regions. Additionally, the strategic split of the dataset into training and testing sets is essential for training machine learning models. This partitioning ensures the model's ability to generalize well to unseen data, enhancing its predictive accuracy. Together, these steps lay the groundwork for comprehensive earthquake analysis, blending geographical insights with machine learning methodologies.

DATA VISUALIZATION

Data visualization plays a crucial role in unraveling the intricate tapestry of earthquake data, offering a lens through which patterns and insights emerge. Leveraging libraries such as Matplotlib, Seaborn, and Basemap, the seismic landscape can be visually represented, providing a comprehensive view of global seismic activity. Histograms and count plots elucidate the distribution and frequency of earthquake magnitudes and types, aiding in the identification of trends. Geospatial plots, facilitated by tools like Basemap, chart the geographic coordinates of seismic events, unveiling spatial correlations and potential seismic clusters. Time-based visualizations, including yearly and monthly count plots, illuminate temporal trends and recurring patterns. Scatter plots provide a holistic view of earthquake occurrences over time, facilitating trend analysis. Such visualizations not only enhance understanding but also serve as a foundation for informed decision-making and the subsequent development of machine learning models for earthquake prediction.

DATA SPLITTING

In the journey of constructing a reliable earthquake prediction model, one indispensable phase is the strategic splitting of the dataset into training and testing sets. This division is fundamental for evaluating the model's generalization performance, providing a robust assessment of its predictive capabilities on unseen data. Through libraries like scikit-learn, the dataset is partitioned, with a portion reserved for training the model and the rest set aside for testing its predictive accuracy. The training set serves as the foundation for the model to learn underlying patterns and relationships, while the testing set serves as a benchmark for assessing its ability to make accurate predictions on new, unseen data. This meticulous separation ensures that the model's effectiveness is not solely tailored to the training data but extends to real-world scenarios, enhancing its reliability in earthquake prediction. The choice of an optimal split ratio is crucial, balancing the need for an adequately trained model with a sufficiently diverse evaluation set.

PROGRAM :

```
# Installing necessary libraries for data visualization
!pip3 install basemap

# Importing libraries for data visualization
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import seaborn as sns
sns.set(style="darkgrid")

# Displaying the minimum and maximum values of the 'Magnitude' column
print("Min Value: "+ str(data['Magnitude'].min()))
print("Max Value: " + str(data['Magnitude'].max()))

# Filtering earthquakes with magnitude greater than 8 and displaying counts by 'Location Source'
Greater_8 = data[data['Magnitude'] > 8]
```

```
Greater_8['Location Source'].value_counts()

# Similar counts for earthquakes with magnitude greater
than 7, 6, 5, and 4
Greater_7 = data[data['Magnitude'] > 7]
Greater_7['Location Source'].value_counts()
Greater_6 = data[data['Magnitude'] > 6]
Greater_6['Location Source'].value_counts()
Greater_5 = data[data['Magnitude'] > 5]
Greater_5['Location Source'].value_counts()
Greater_4 = data[data['Magnitude'] > 4]
Greater_4['Location Source'].value_counts()

# Histogram of earthquake magnitudes
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')

# Count plot of '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)")
```

```
# Function to determine marker color based on earthquake magnitude
def get_marker_color(magnitude):
    if magnitude < 6.2:
        return ('go')
    elif magnitude < 7.5:
        return ('yo')
    else:
        return ('ro')


# Basemap plot of earthquakes with different marker colors based on magnitude
plt.figure(figsize=(14,10))
eq_map = Basemap(projection='robin', resolution = 'l', lat_0=0, lon_0=-130)
eq_map.drawcoastlines()
eq_map.drawcountries()
eq_map.fillcontinents(color='gray')
eq_map.drawmapboundary()
eq_map.drawmeridians(np.arange(0, 360, 30))
lons = data['Longitude'].values
lats = data['Latitude'].values
magnitudes = data['Magnitude'].values
timestrings = data['Date'].tolist()
min_marker_size = 0.5
for lon, lat, mag in zip(lons, lats, magnitudes):
    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()
```

# Count plot of the number of earthquakes in 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="blue")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each Year')
```

# Displaying the top 5 years with the highest number of earthquakes

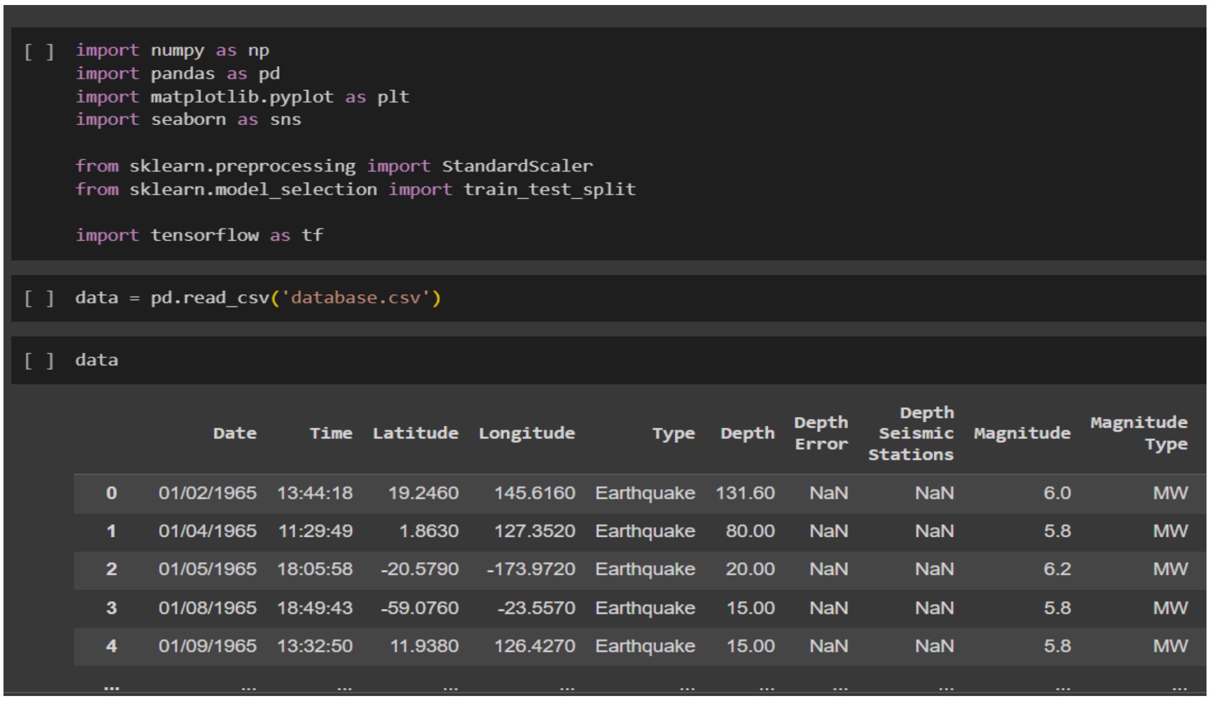
```
data['year'].value_counts()[:5]
```

```
# Count plot of the number 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')

# Displaying the top 5 months with the highest number of earthquakes
data['mon'].value_counts()[:5]

# Count plot of the number of earthquakes in each day of the month
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="orange")
```

```
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each days')

# Displaying the top 5 days of the month with the highest number of earthquakes
data['days'].value_counts()[:5]

# 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])
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 earthquake magnitudes into classes
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'

# Count plot of magnitude class distribution
sns.countplot(x='Class', data=data)
plt.ylabel('Frequency')
plt.title('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)
```

OUTPUT:

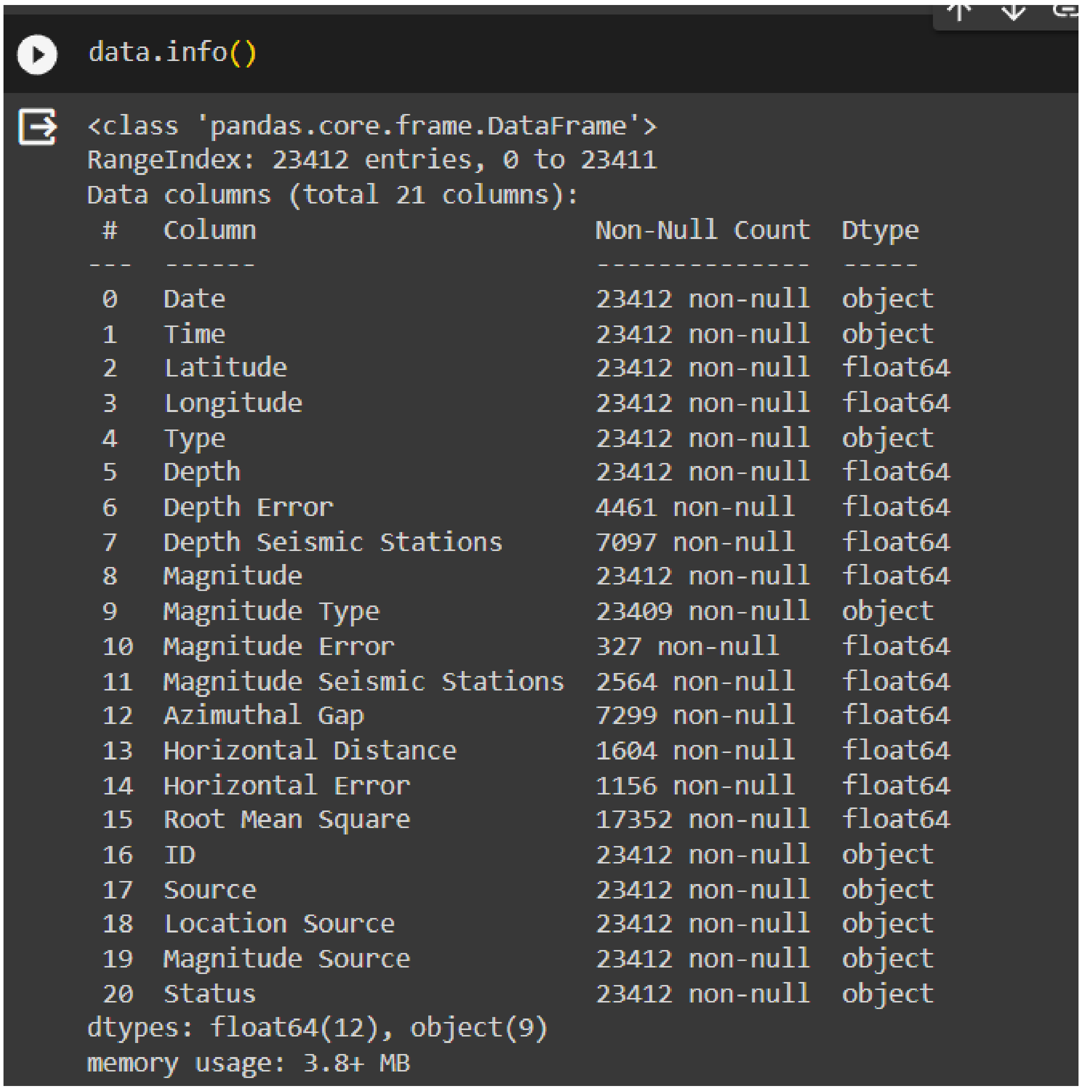


```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns

    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split

    import tensorflow as tf
```

```
[ ] data = pd.read_csv('database.csv')
```

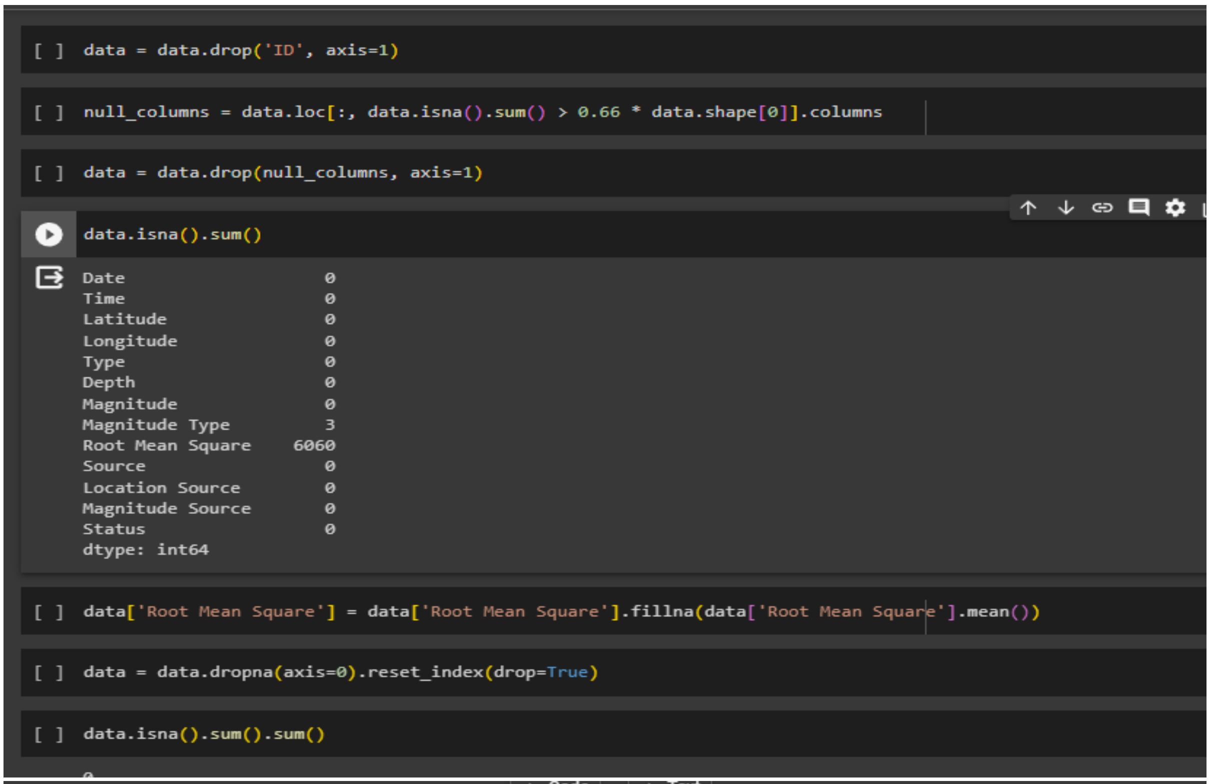
```
[ ] data
```

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW
...	...	...	...	...	...	...	...	...	...	...



```
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
```



```
[ ] 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)
```

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

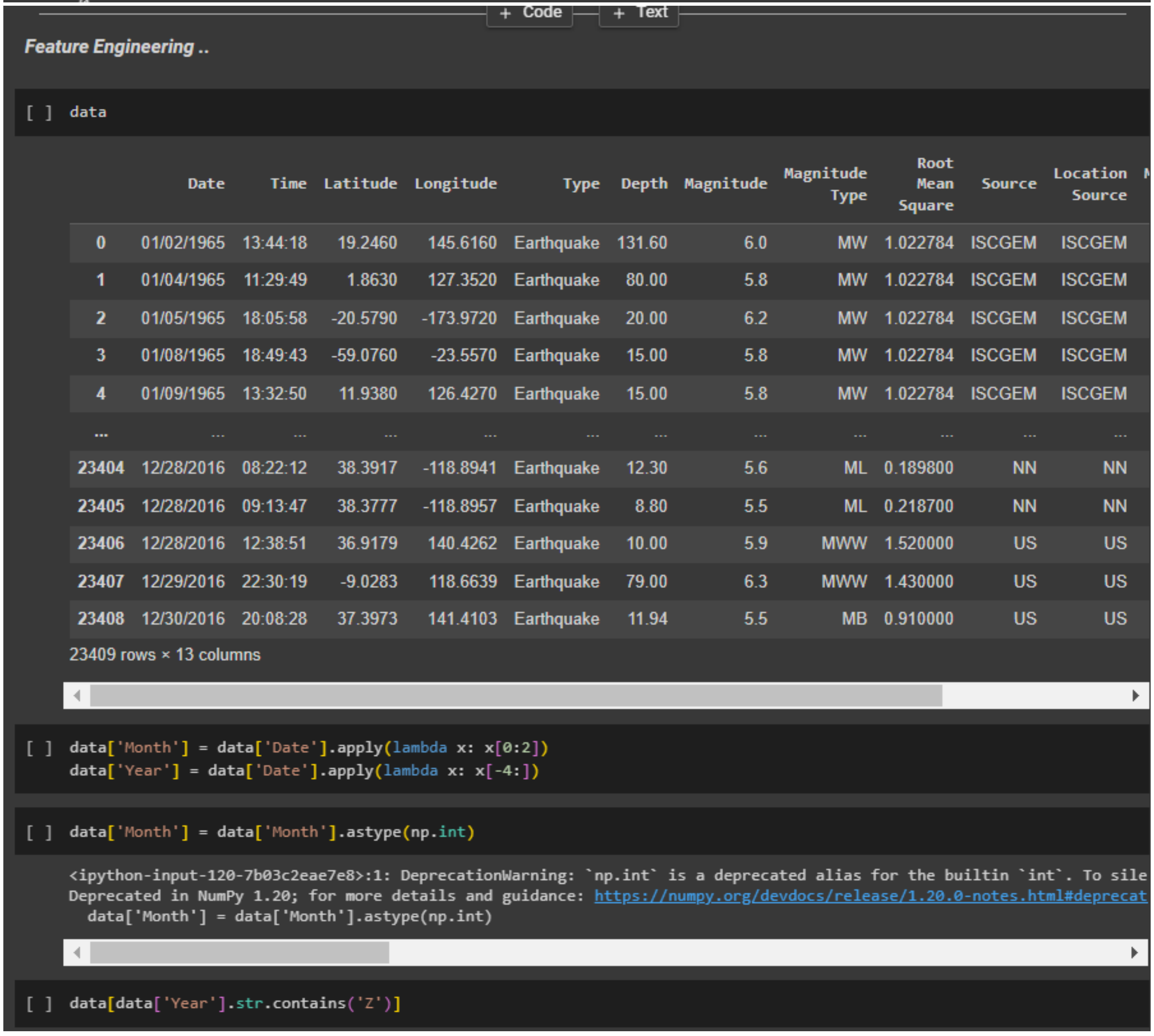
```
Date                   0
Time                   0
Latitude               0
Longitude              0
Type                   0
Depth                  0
Magnitude              0
Magnitude Type         3
Root Mean Square    6060
Source                 0
Location Source        0
Magnitude Source     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

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

```
[ ] data = data.dropna(axis=0).reset_index(drop=True)
```

```
[ ] data.isna().sum().sum()
```

```
0
```



Feature Engineering ..

```
[ ] data
```

	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
...	...	...	...	...	...	...	...	...	...	...	...
23404	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN
23405	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN
23406	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US
23407	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US
23408	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US

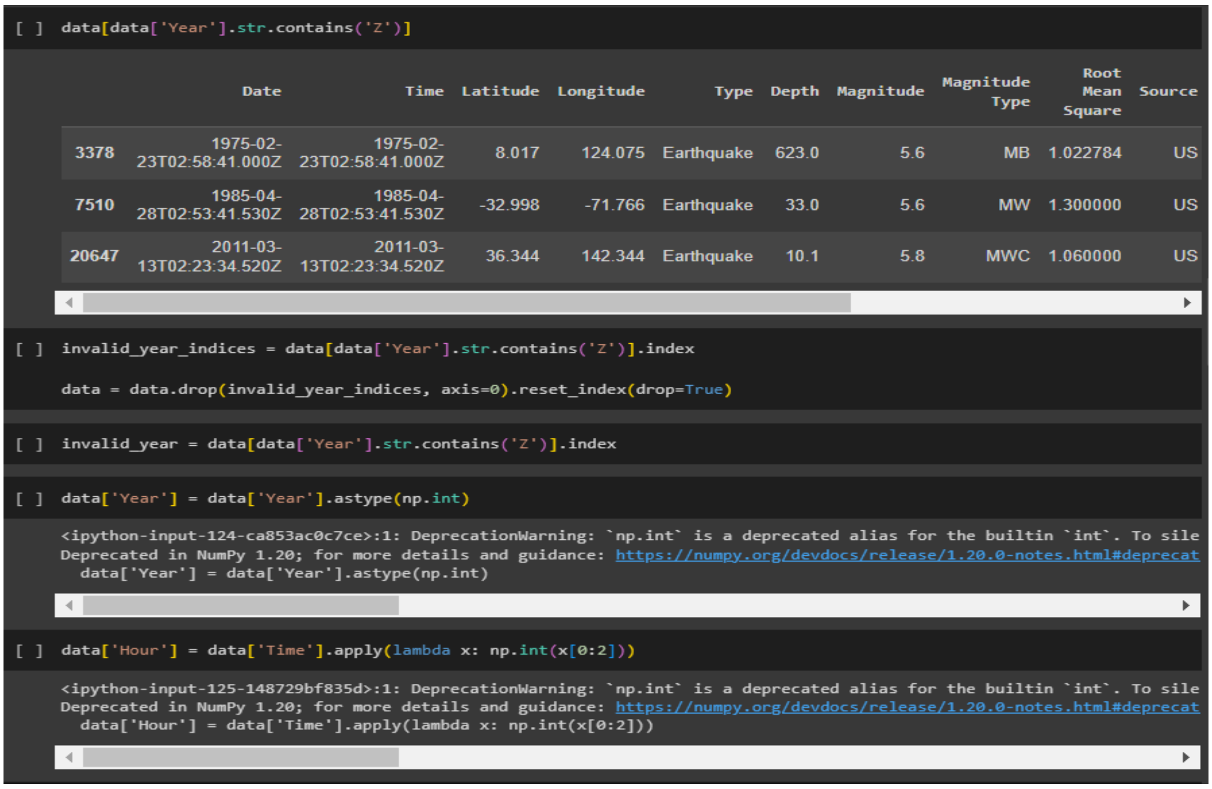
23409 rows × 13 columns

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

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

```
<ipython-input-120-7b03c2eae7e8>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
  data['Month'] = data['Month'].astype(np.int)
```

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



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

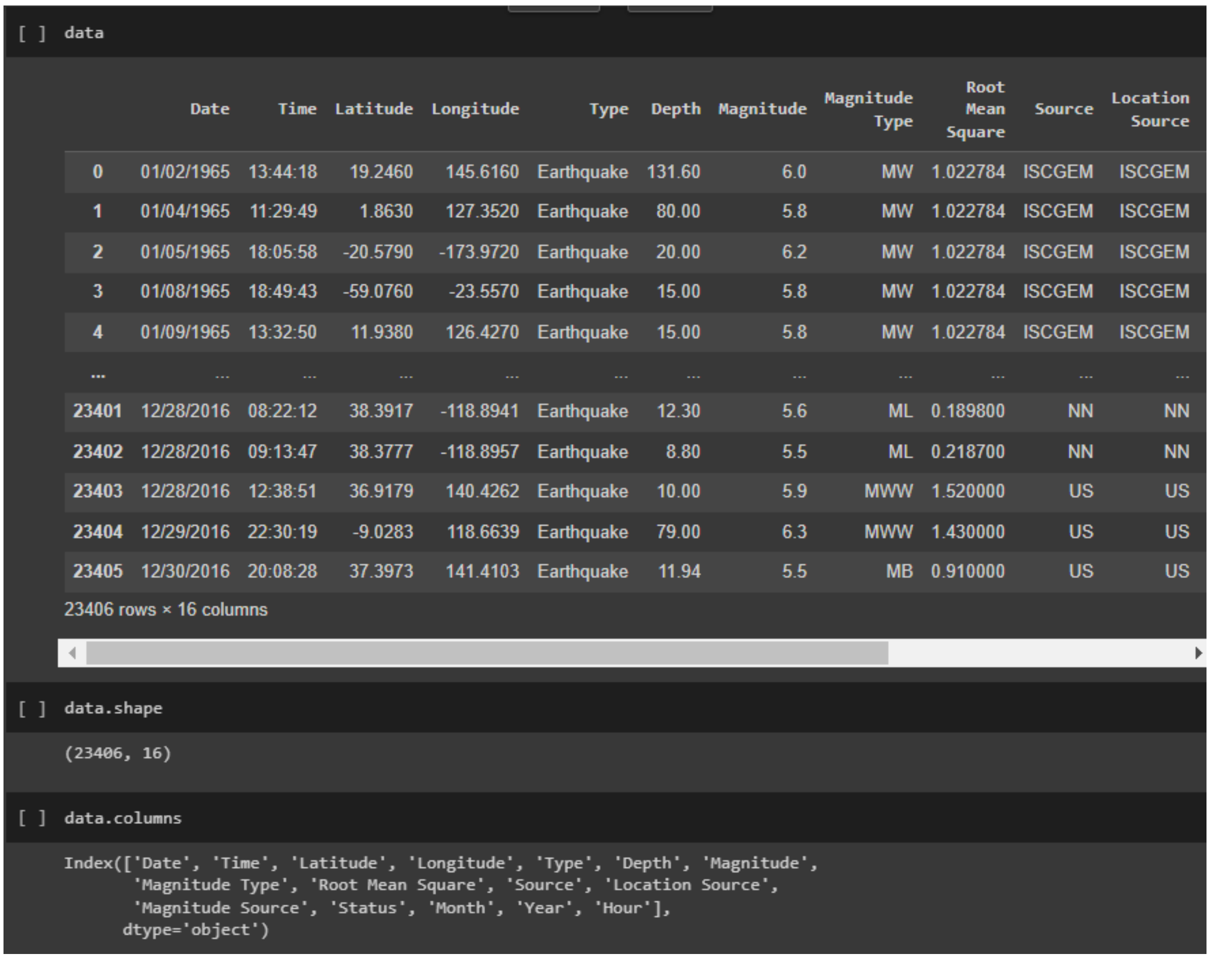
	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source
3378	1975-02-23T02:58:41.000Z	1975-02-23T02:58:41.000Z	8.017	124.075	Earthquake	623.0	5.6	MB	1.022784	US
7510	1985-04-28T02:53:41.530Z	1985-04-28T02:53:41.530Z	-32.998	-71.766	Earthquake	33.0	5.6	MW	1.300000	US
20647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

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

    data = data.drop(invalid_year_indices, axis=0).reset_index(drop=True)
```

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

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

    <ipython-input-124-ca853ac0c7ce>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
      data['Year'] = data['Year'].astype(np.int)
```

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

    <ipython-input-125-148729bf835d>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
      data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))
```



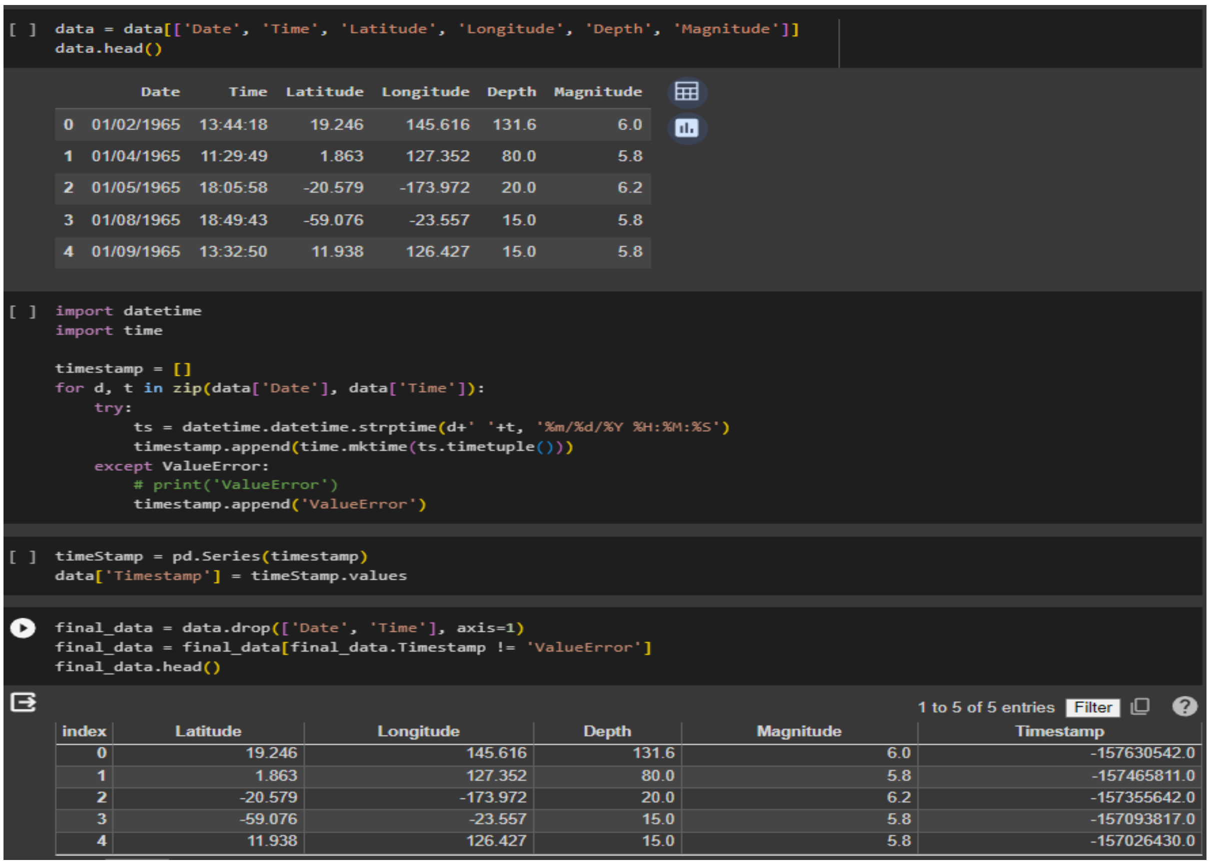
```
[ ] data
```

	Date	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM
...	...	...	...	...	...	...	...	...	...	...	...
23401	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN
23402	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN
23403	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US
23404	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US
23405	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US

23406 rows × 16 columns

```
[ ] data.shape

    (23406, 16)
```

```
[ ] data.columns

    Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Magnitude',
           'Magnitude Type', 'Root Mean Square', 'Source', 'Location Source',
           'Magnitude Source', 'Status', 'Month', 'Year', 'Hour'],
          dtype='object')
```



```
[ ] 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

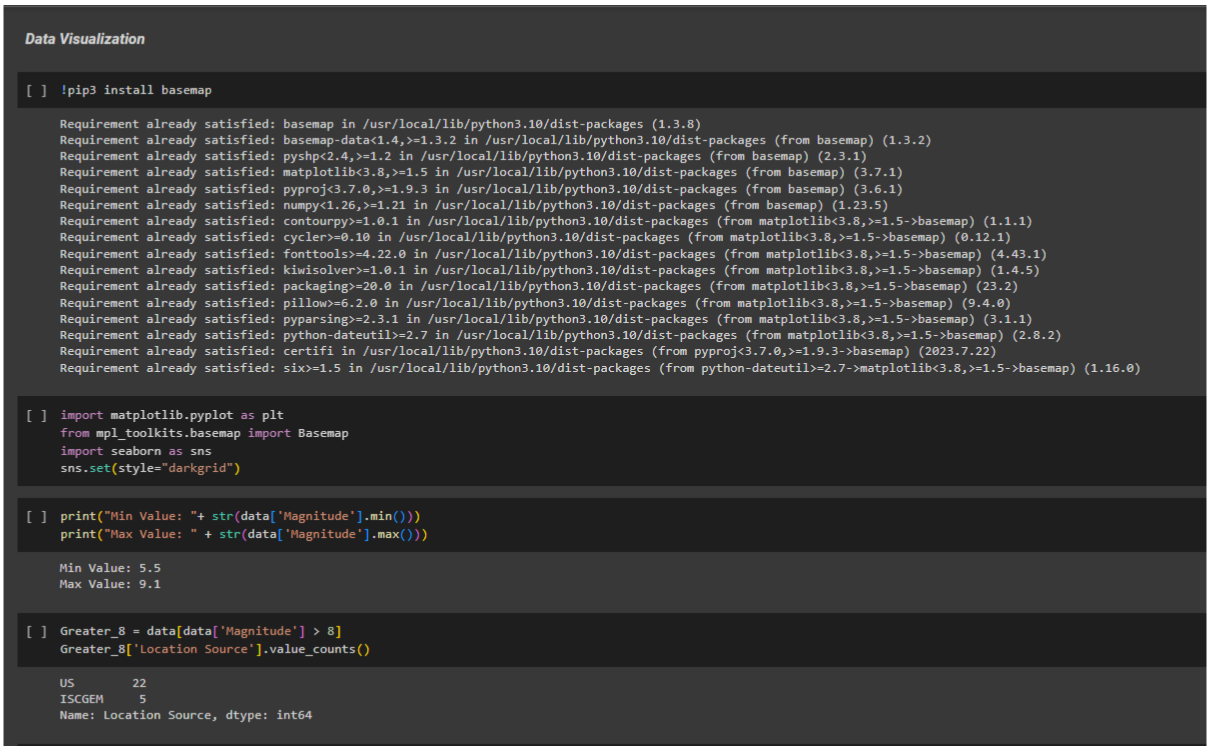
```
[ ] import datetime
    import time

    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')
```

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

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

1 to 5 of 5 entries Filter

index	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



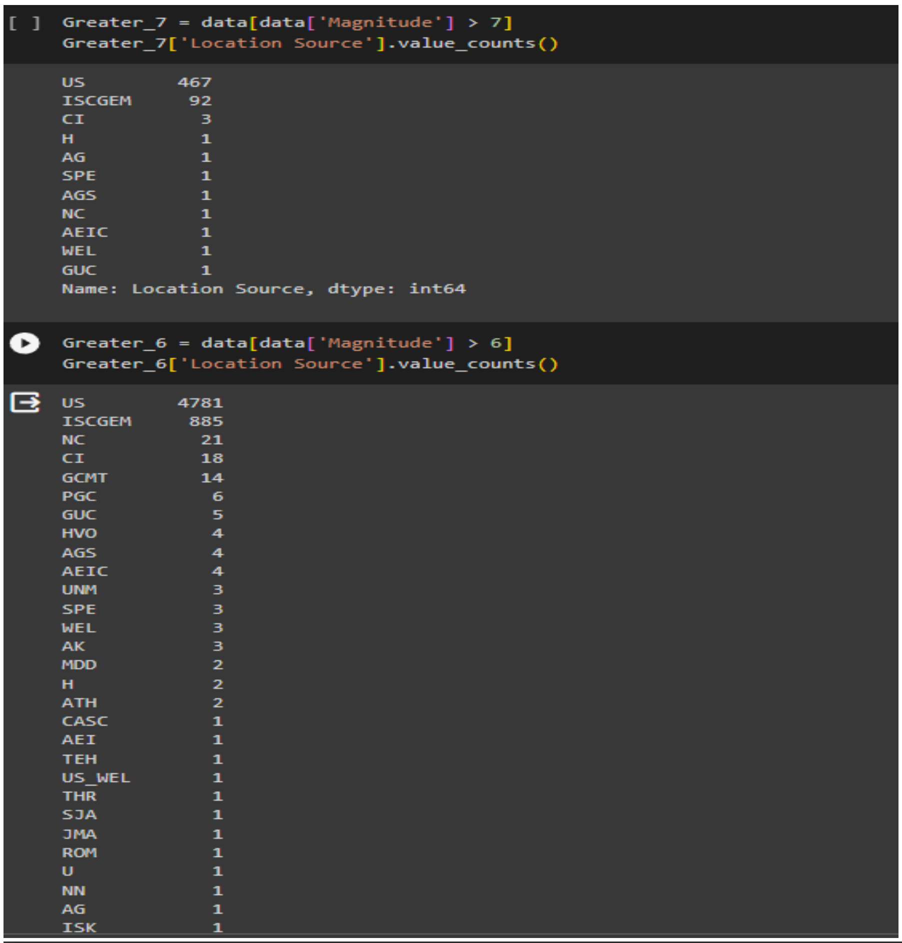
Data Visualization

```
[ ] !pip3 install basemap

    Requirement already satisfied: basemap in /usr/local/lib/python3.10/dist-packages (1.3.8)
    Requirement already satisfied: basemap-data<1.4,>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from basemap) (1.3.2)
    Requirement already satisfied: pyshp<2.4,>=1.2 in /usr/local/lib/python3.10/dist-packages (from basemap) (2.3.1)
    Requirement already satisfied: matplotlib<3.8,>=1.5 in /usr/local/lib/python3.10/dist-packages (from basemap) (3.7.1)
    Requirement already satisfied: pyproj<3.7.0,>=1.9.3 in /usr/local/lib/python3.10/dist-packages (from basemap) (3.6.1)
    Requirement already satisfied: numpy<1.26,>=1.21 in /usr/local/lib/python3.10/dist-packages (from basemap) (1.23.5)
    Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (1.1.1)
    Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (0.12.1)
    Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (4.43.1)
    Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (1.4.5)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (23.2)
    Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (9.4.0)
    Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (3.1.1)
    Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.8,>=1.5->basemap) (2.8.2)
    Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from pyproj<3.7.0,>=1.9.3->basemap) (2023.7.22)
    Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib<3.8,>=1.5->basemap) (1.16.0)
```

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

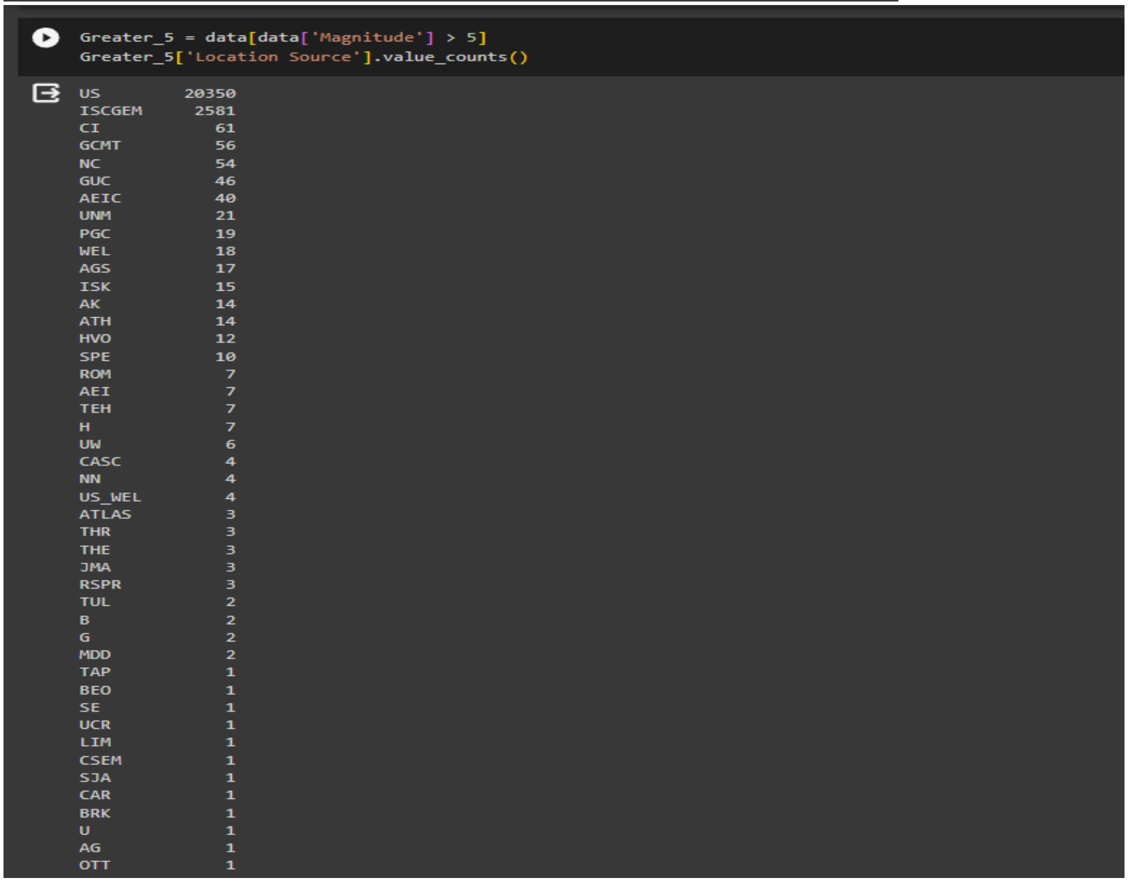
```
[ ] print("Min Value: "+ str(data['Magnitude'].min()))
    print("Max Value: " + str(data['Magnitude'].max()))

    Min Value: 5.5
    Max Value: 9.1
```

```
[ ] 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
```



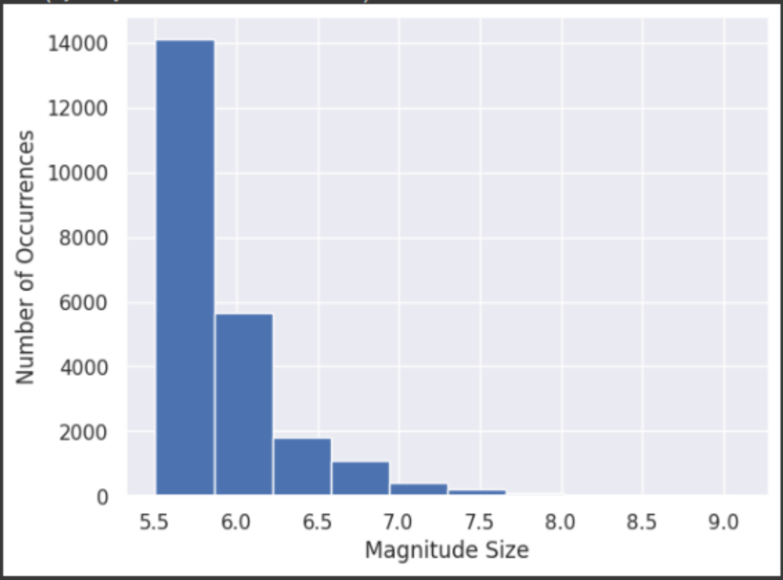
```
     Greater_5 = data[data['Magnitude'] > 5]
     Greater_5['Location Source'].value_counts()

     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
```

```
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
```

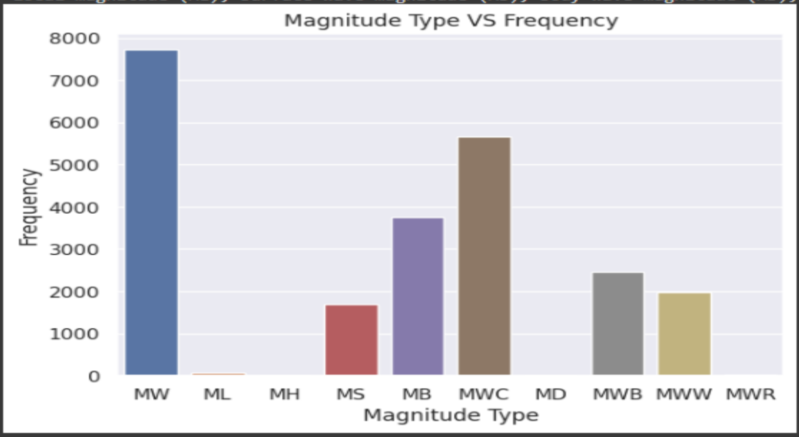
```
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')
```

```
Text(0, 0.5, 'Number of Occurrences')
```



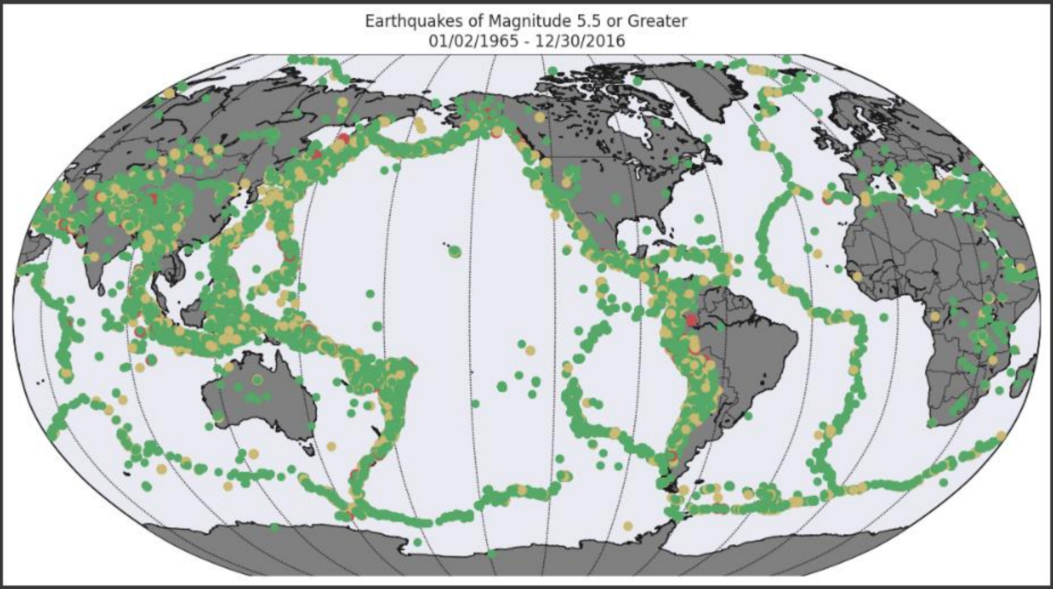
```
[ ] sns.countplot(x="Magnitude Type", data=data)
    plt.ylabel('Frequency')
    plt.title('Magnitude Type VS Frequency')
    print(" local magnitude (ML), surface-wave magnitude (Ms), body-wave magnitude (Mb), moment magnitude (Mm)")
```

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



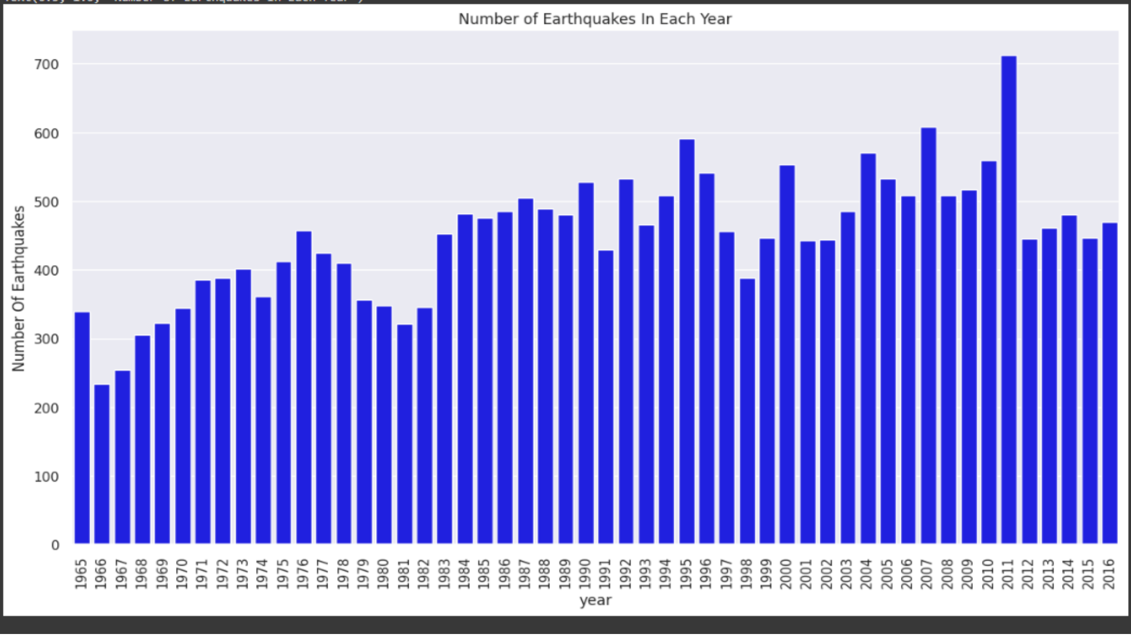
```python
def get_marker_color(magnitude):
    if magnitude < 6.2:
        return ('go')
    elif magnitude < 7.5:
        return ('yo')
    else:
        return ('ro')

plt.figure(figsize=(14,10))

eq_map = Basemap(projection='robin', resolution = 'l',
              lat_0=0, lon_0=-130)
eq_map.drawcoastlines()
eq_map.drawcountries()
eq_map.fillcontinents(color = 'gray')
eq_map.drawmapboundary()
eq_map.drawmeridians(np.arange(0, 360, 30))
```

```python
# read longitude, latitude and magnitude
lons = data['Longitude'].values
lats = data['Latitude'].values
magnitudes = data['Magnitude'].values
timestrings = data['Date'].tolist()

min_marker_size = 0.5
for lon, lat, mag in zip(lons, lats, magnitudes):
    x,y = eq_map(lon, lat)
    msize = mag # * min_marker_size
    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()
```



```
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 = "blue")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each Year')
```

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

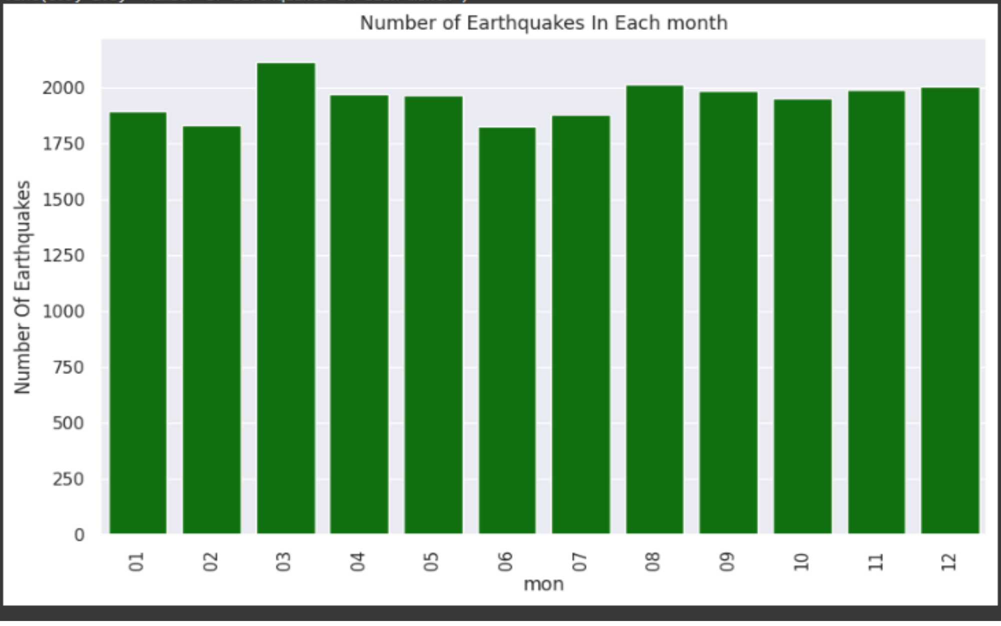


```
data['year'].value_counts()[:5]
```

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

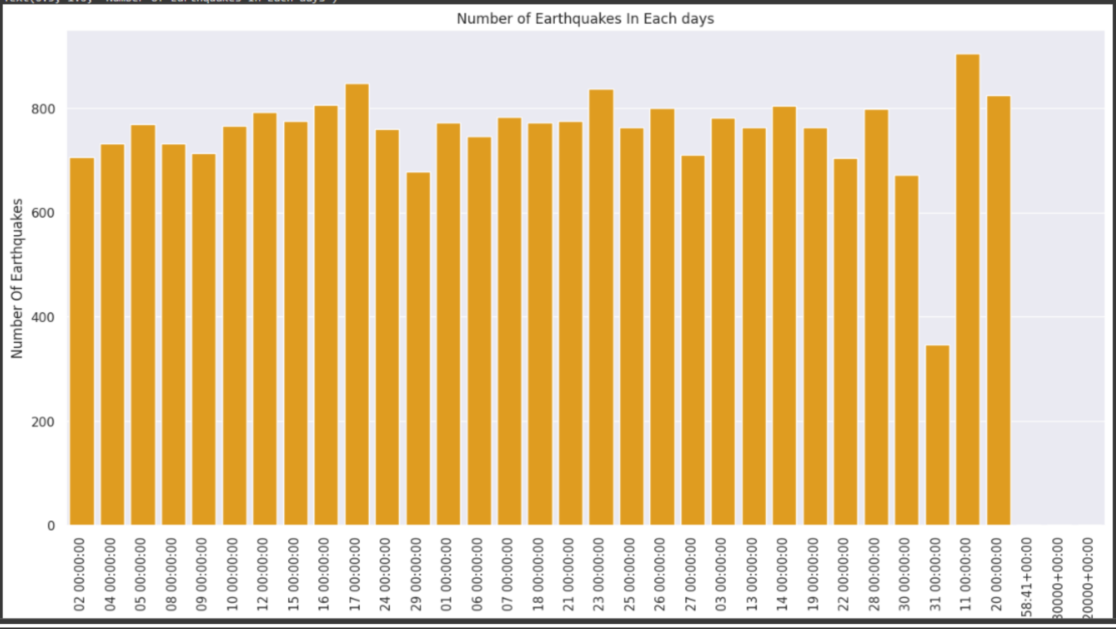
```
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['mon'] = data['date'].apply(lambda x: str(x).split('-')[1])
plt.figure(figsize=(10, 6))
sns.set(font_scale=1)
ax = sns.countplot(x="mon", data=data, color = "green")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)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

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



```python
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 = "orange")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each days')
```

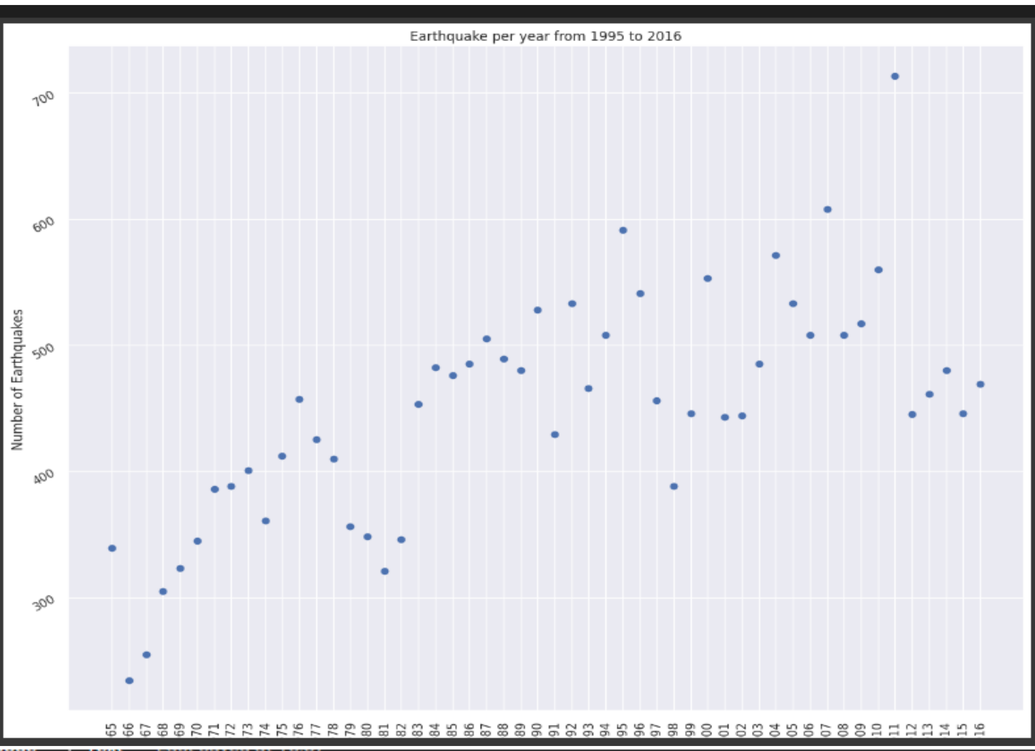
```
Text(0.5, 1.0, 'Number of Earthquakes In Each days')
```



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

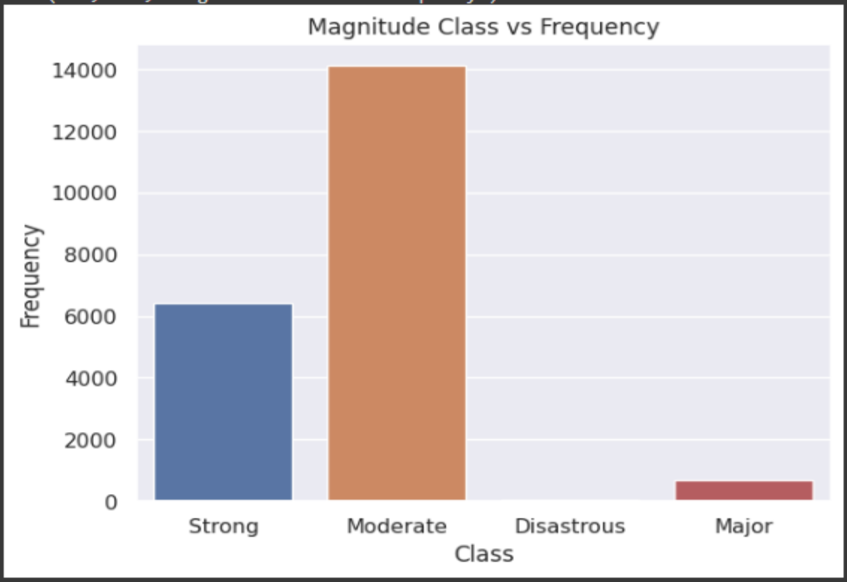
```
     11 00:00:00    905
     17 00:00:00    848
     23 00:00:00    837
     20 00:00:00    825
     16 00:00:00    807
     Name: days, dtype: int64
```

```python
[ ]  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()
```



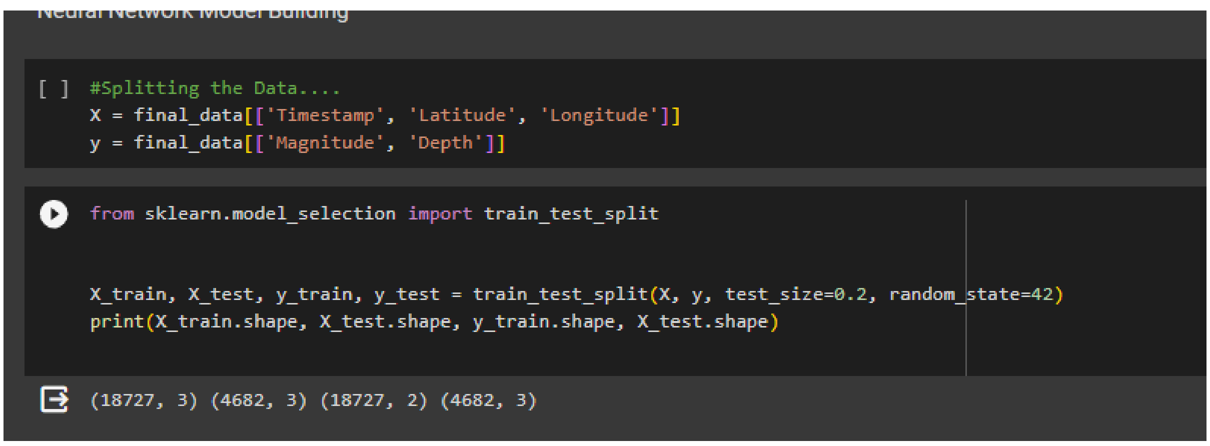
```
#Classification of magnitude types
data.loc[data['Magnitude'] >=8, 'Class'] = 'Disastrous'
data.loc[ (data['Magnitude'] >= 7) & (data['Magnitude'] < 7.9), 'Class'] = 'Major'
data.loc[ (data['Magnitude'] >= 6) & (data['Magnitude'] < 6.9), 'Class'] = 'Strong'
data.loc[ (data['Magnitude'] >= 5.5) & (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')
```



Neural Network Model Building

```
[ ] #Splitting the Data....
    X = final_data[['Timestamp', 'Latitude', 'Longitude']]
    y = final_data[['Magnitude', 'Depth']]
```



```
Neural Network Model Building

[ ] #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)
```

CONCLUSION

In conclusion, the data visualization efforts employing tools such as Basemap have provided crucial insights into the geographical distribution of earthquakes, offering a comprehensive view of seismic activities worldwide. This spatial understanding is pivotal for identifying regions prone to seismic events and informs subsequent modeling endeavors. Simultaneously, the strategic process of data splitting into training and testing sets marks a crucial preparatory phase in developing a robust earthquake prediction model. This division ensures that the model is trained on a diverse dataset, enabling it to capture underlying patterns and relationships effectively. The testing set serves as a stringent benchmark, evaluating the model's generalization capacity and predictive accuracy on previously unseen data. The combined efforts in data visualization and splitting lay a solid foundation for subsequent machine learning model development, with the goal of creating an accurate and reliable system for earthquake prediction. The integration of geographical insights and rigorous data partitioning enhances the model's adaptability and ensures its applicability in real-world scenarios.

PHASE – 4

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 DEPLOYMENT

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)
```

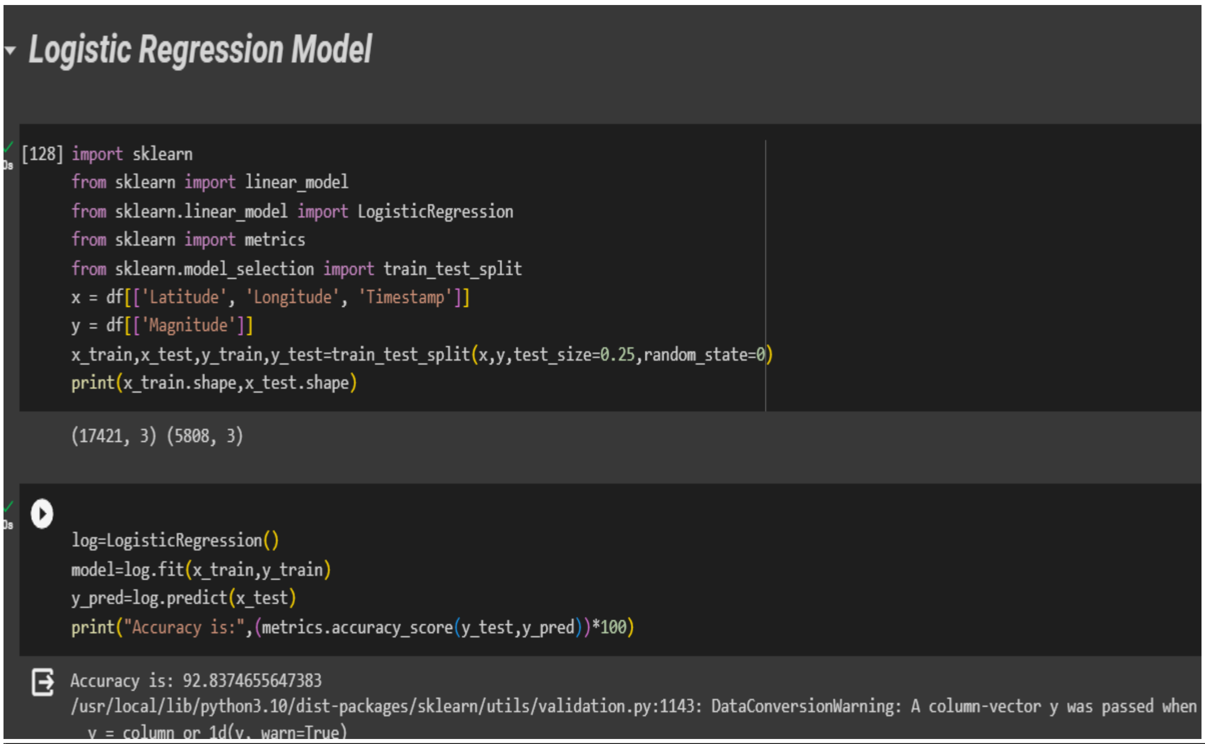
```
# Creating and training the Logistic Regression model
log = LogisticRegression()
model = log.fit(x_train, y_train)
y_pred = log.predict(x_test)

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



# Neural Network Model


# Importing necessary libraries
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 import KerasClassifier
```

```
# 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 network
model
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 hyperparameter tuning
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 parameters for 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 best
parameters

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 :



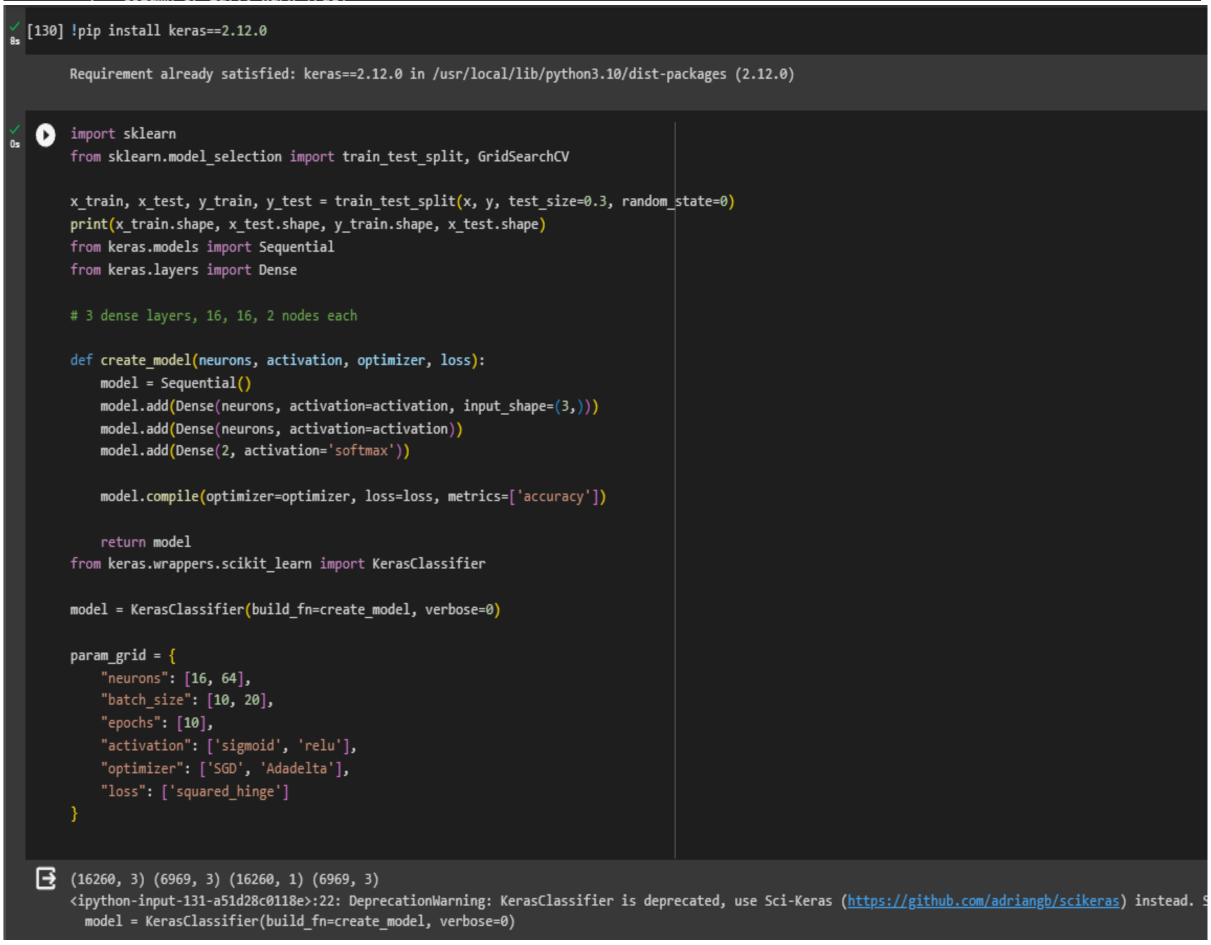
Logistic Regression Model

```
[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)
```

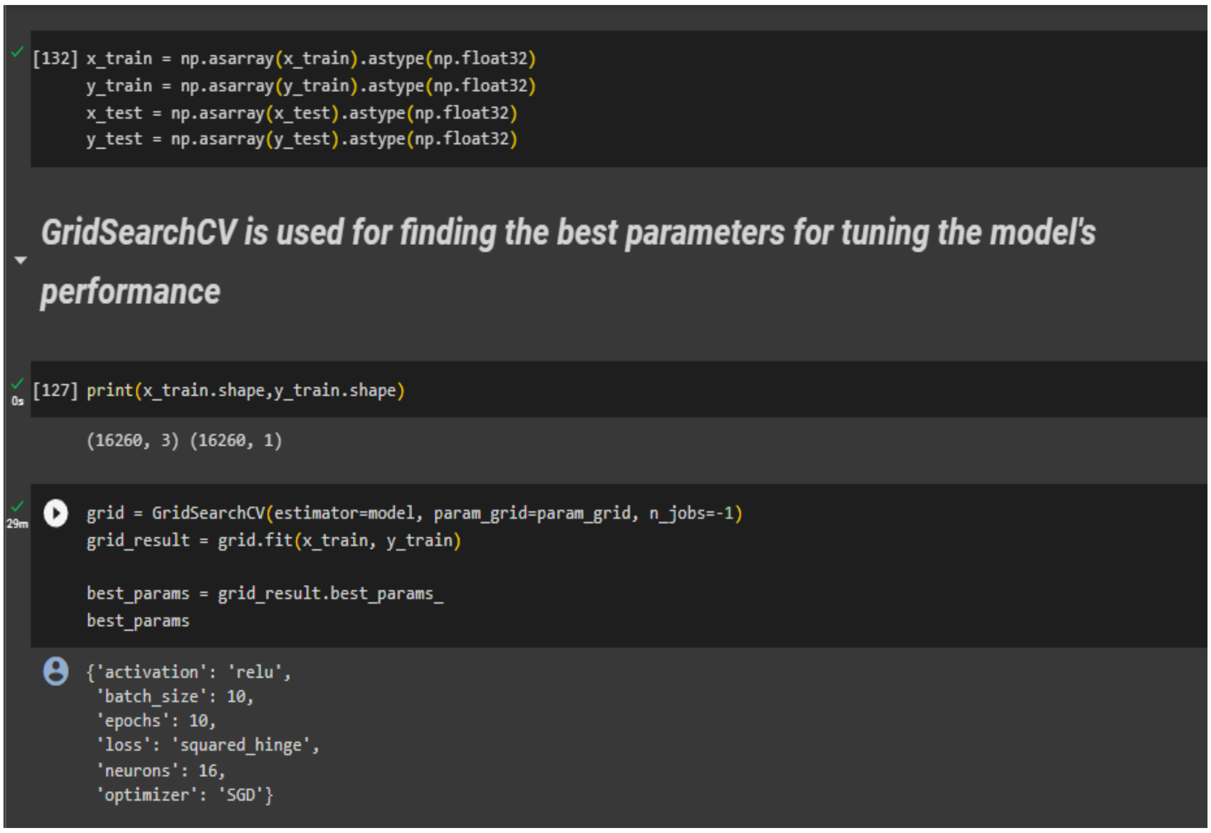


```
[130] !pip install keras==2.12.0
```

```
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.
  model = KerasClassifier(build_fn=create_model, verbose=0)
```



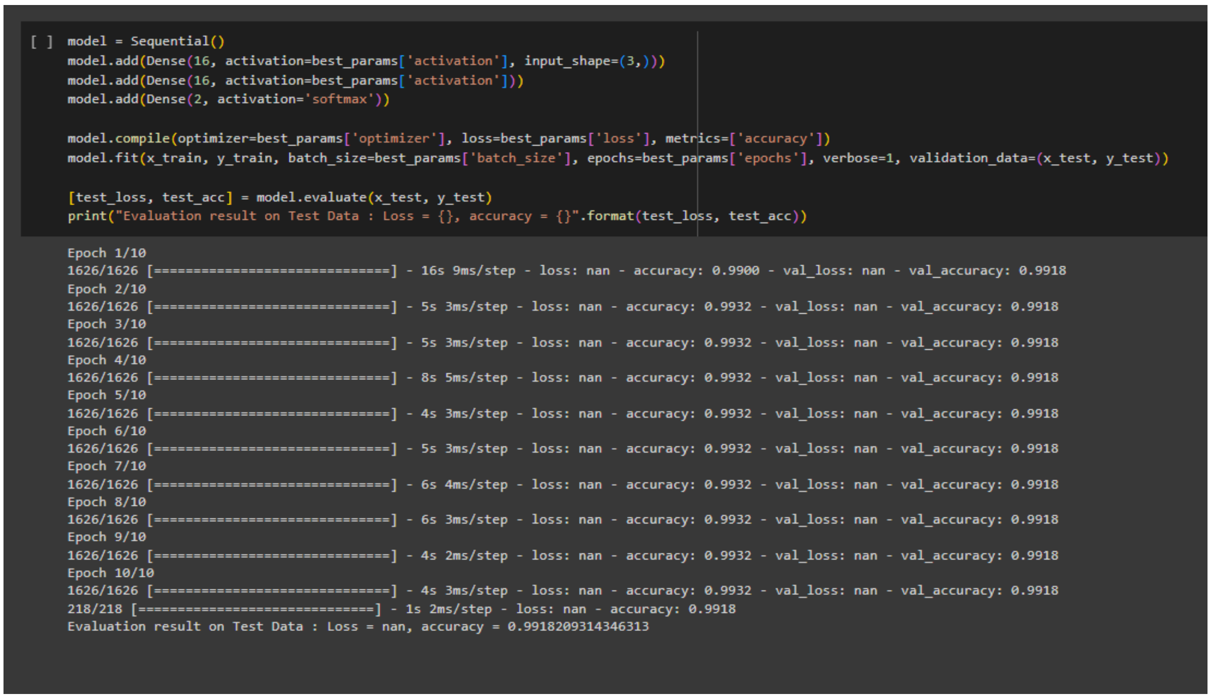
```
[132] x_train = np.asarray(x_train).astype(np.float32)
      y_train = np.asarray(y_train).astype(np.float32)
      x_test = np.asarray(x_test).astype(np.float32)
      y_test = np.asarray(y_test).astype(np.float32)
```

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 phases crafts a holistic and adaptive framework, essential for the successful integration of machine learning solutions into diverse domains.