IBM NaanMudhalvan

ARTIFICIAL INTELLIGENCE

Project Title : Earthquake Prediction Using Python

Phase 4 : Development Part – 2

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

Workbook Link : Google Colab

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()
```

+ Code + Text

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()
```

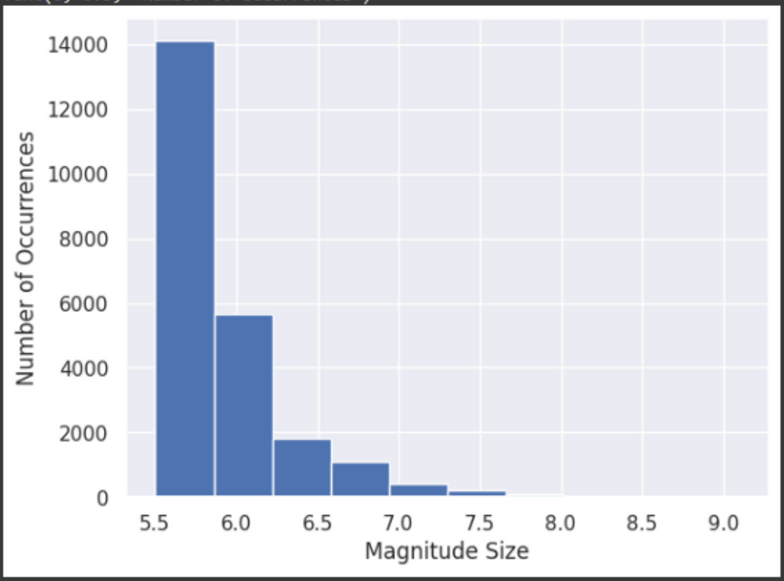
```
US          20350
ISCGEM       2581
CI             61
GCMT           56
NC             54
GUC            46
AEIC           40
UNM            21
PGC            19
WEL            18
AGS            17
ISK            15
AK             14
ATH            14
HVO            12
SPE            10
ROM             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
```

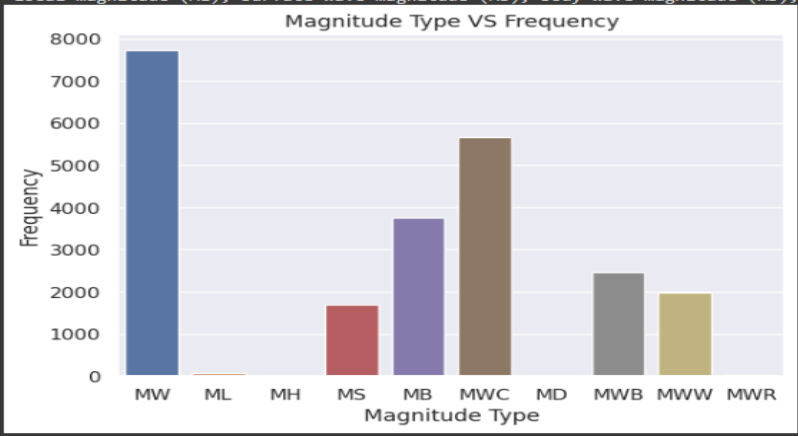
```
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')
```

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



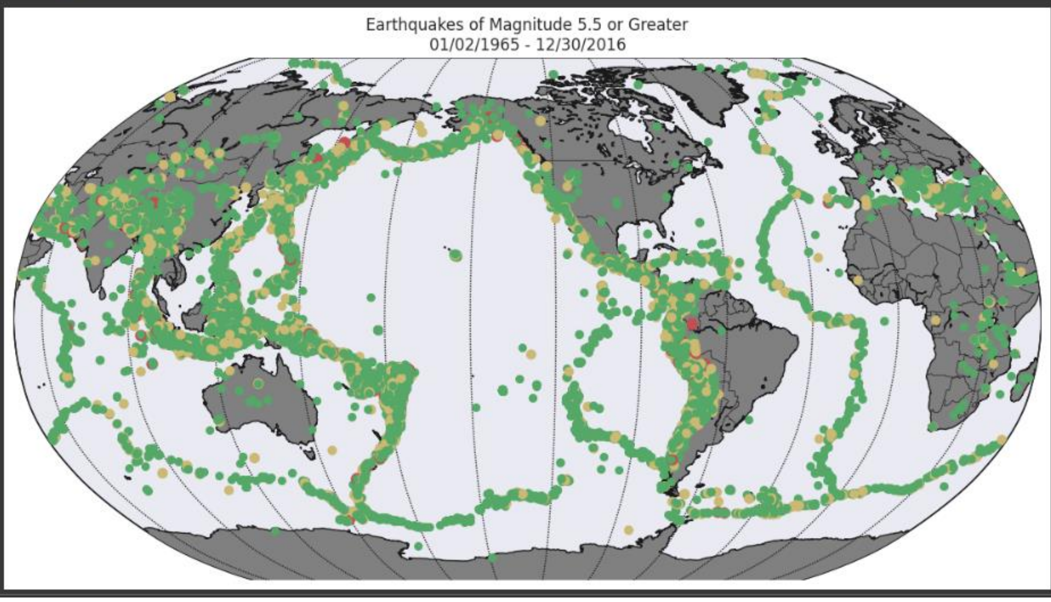

```
[ ] sns.countplot(x="Magnitude Type", data=data)
    plt.ylabel('Frequency')
    plt.title('Magnitude Type VS Frequency')
    print(" local magnitude (ML), surface-wave magnitude (Ms), body-wave magnitude (Mb), moment magnitude (Mm)")
```

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



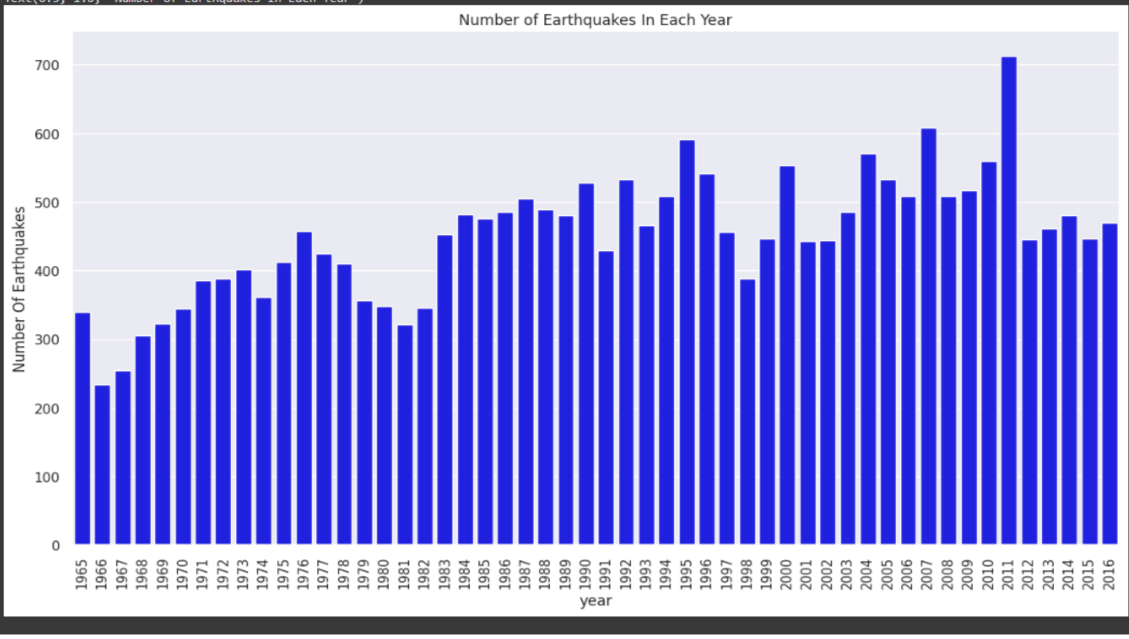

```python
def get_marker_color(magnitude):
    if magnitude < 6.2:
        return ('go')
    elif magnitude < 7.5:
        return ('yo')
    else:
        return ('ro')

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()
```



```python
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')
```

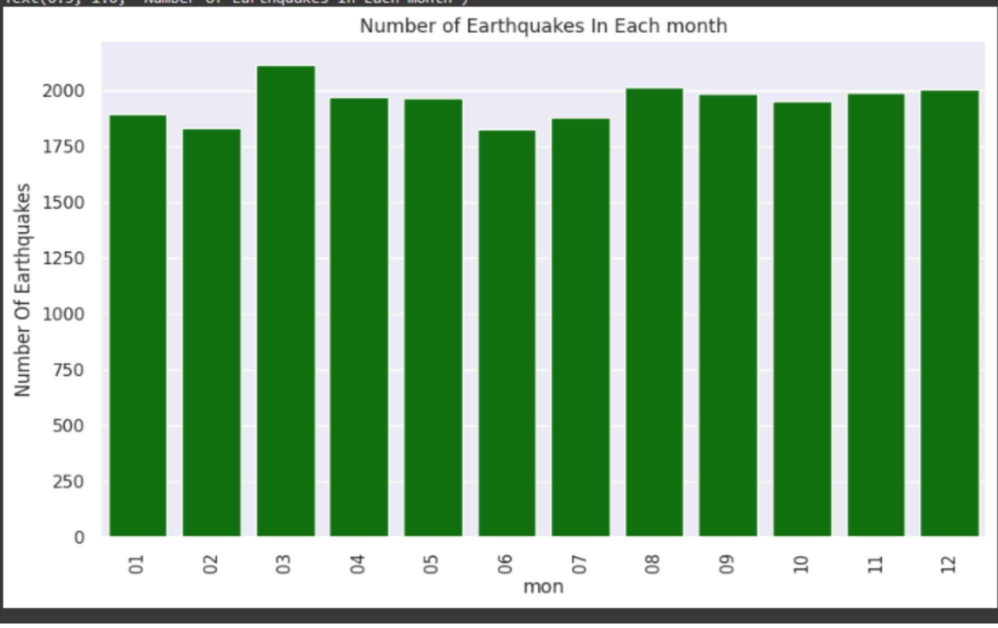



```python
data['year'].value_counts()[:5]
```

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

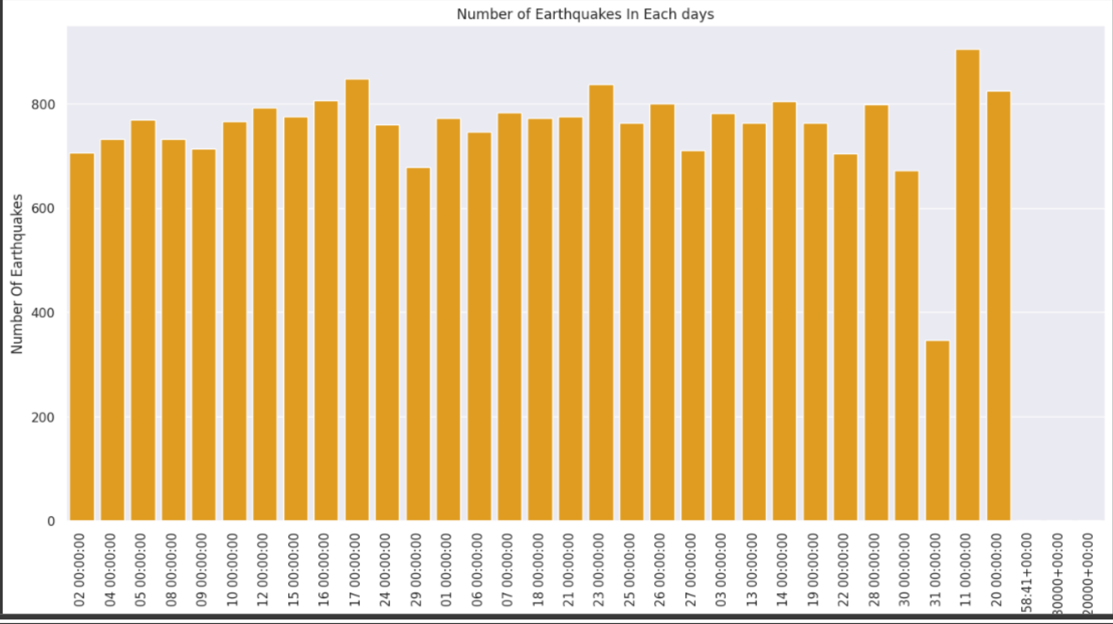
```python
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['mon'] = data['date'].apply(lambda x: str(x).split('-')[1])
plt.figure(figsize=(10, 6))
sns.set(font_scale=1)
ax = sns.countplot(x="mon", data=data, color = "green")
ax.set_xticklabels(ax.get_xticklabels(),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

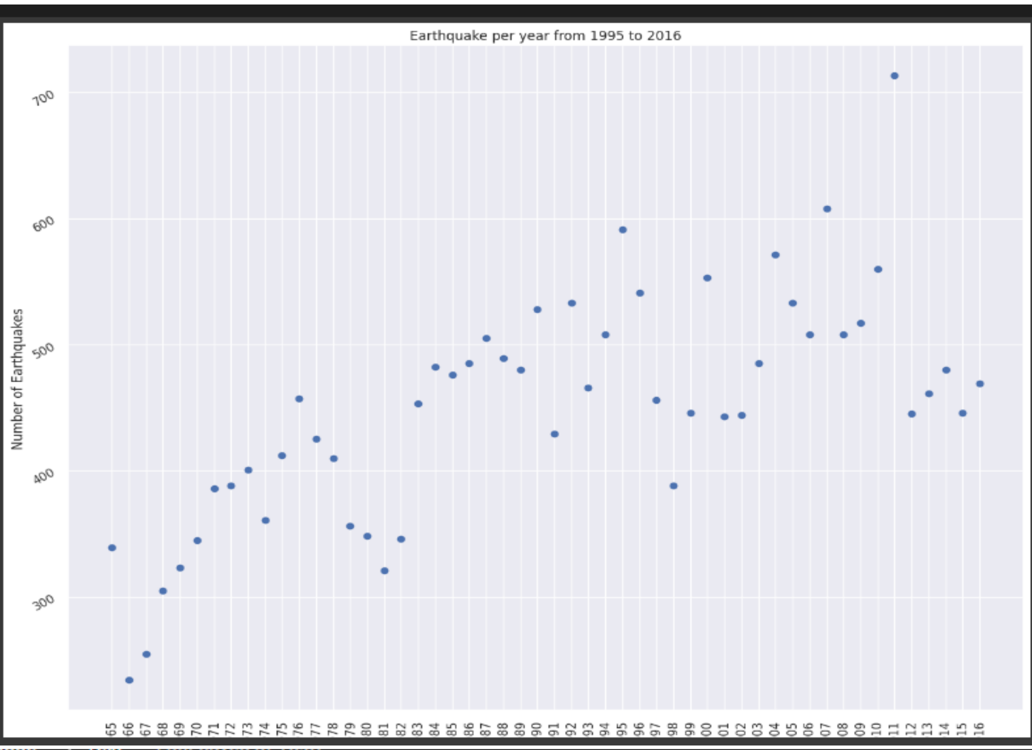
```
Text(0.5, 1.0, 'Number of Earthquakes In Each days')
```




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

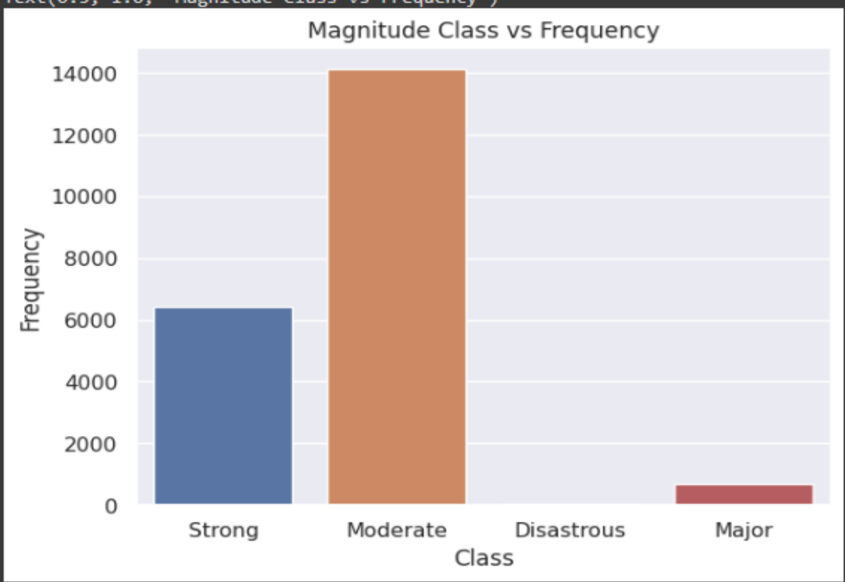
```
11 00:00:00    905
17 00:00:00    848
23 00:00:00    837
20 00:00:00    825
16 00:00:00    807
Name: days, dtype: int64
```

```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.