IBM NAAN MUDHALVAN

Technology name: Artificial Intelligence

Project title: Earthquake prediction model using python

Phase 4: Development Part-2

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

Dataset link:

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

Introduction:

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

About Dataset

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

Content

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

What is Data Visualization?

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

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

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

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

What is Data splitting?

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

Syntax:

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

Parameters:

1. *arrays: inputs such as lists, arrays, data frames, or matrices
2. test_size: this is a float value whose value ranges between 0.0 and 1.0. it represents the proportion of our test size. its default value is none.
3. train_size: this is a float value whose value ranges between 0.0 and 1.0. it represents the proportion of our train size. its default value is none.
4. random_state: this parameter is used to control the shuffling applied to the data before applying the split. it acts as a seed.
5. shuffle: This parameter is used to shuffle the data before splitting. Its default value is true.
6. stratify: This parameter is used to split the data in a stratified fashion.

Dataset location:

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

File location:

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

Program and Output:

#installing the basemap into colab

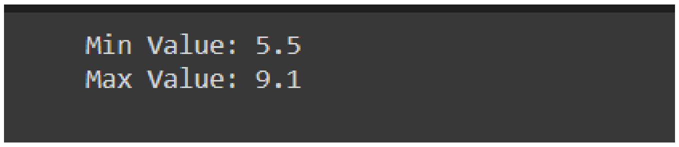
!pip install basemap

#Importing the necessary libraries

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

#Checking the minimum and maximum of magnitude

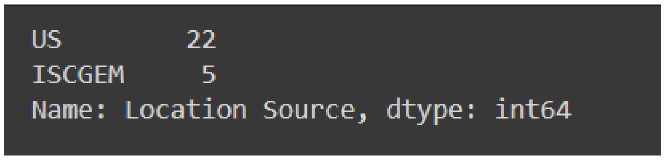
```
print("Min Value: "+ str(data['Magnitude'].min()))
print("Max Value: "+ str(data['Magnitude'].max()))
```



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

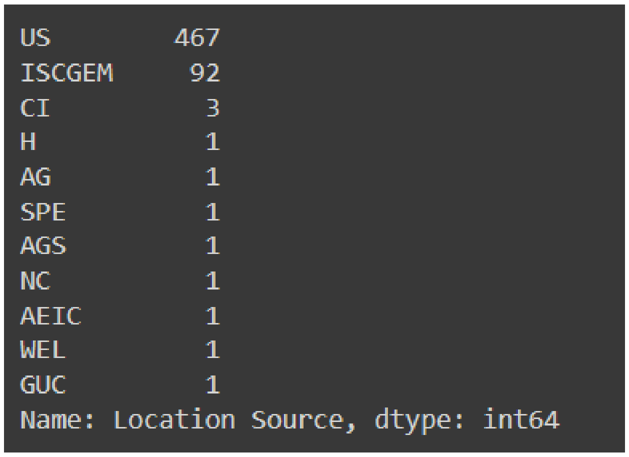
#magnitudes greater than 8 to 4

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



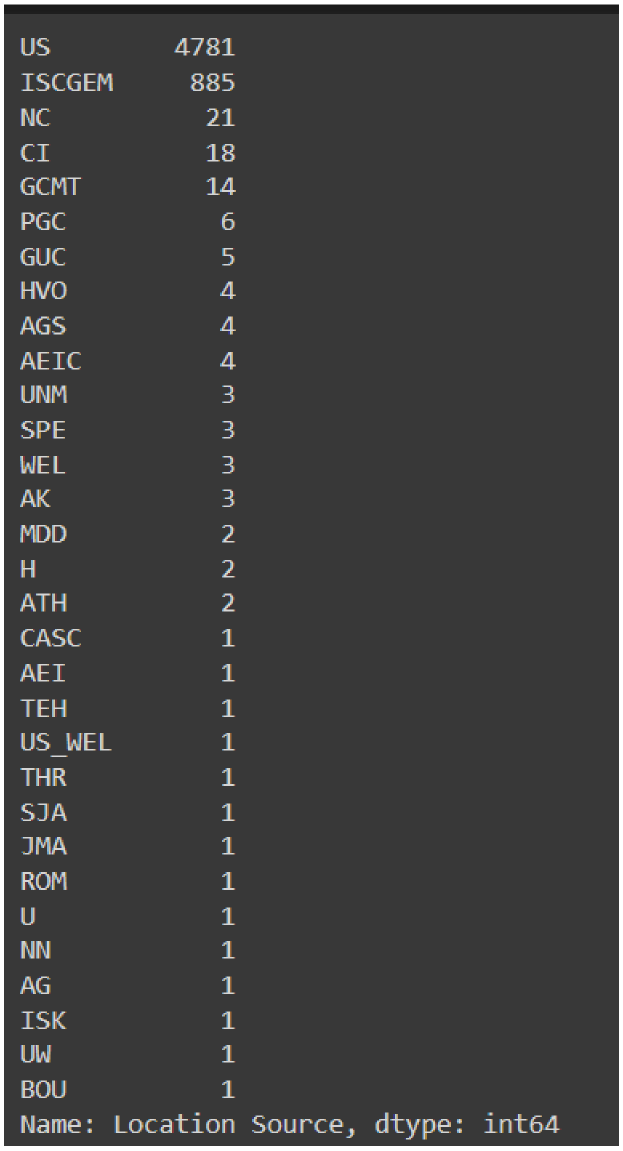
```
US          22
ISCGEM       5
Name: Location Source, dtype: int64
```

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



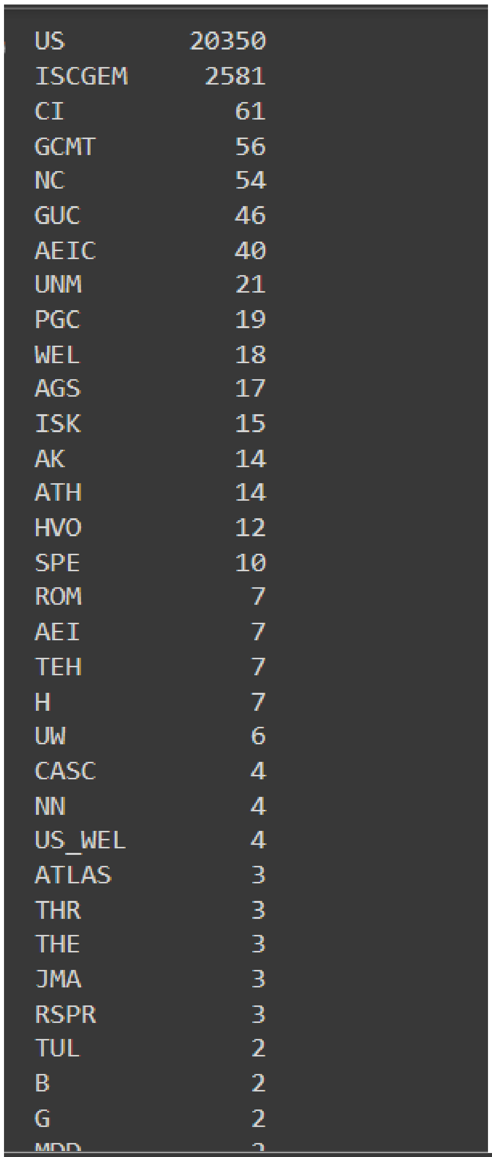
```
US         467
ISCGEM      92
CI           3
H            1
AG           1
SPE          1
AGS          1
NC           1
AEIC         1
WEL          1
GUC          1
Name: Location Source, dtype: int64
```

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

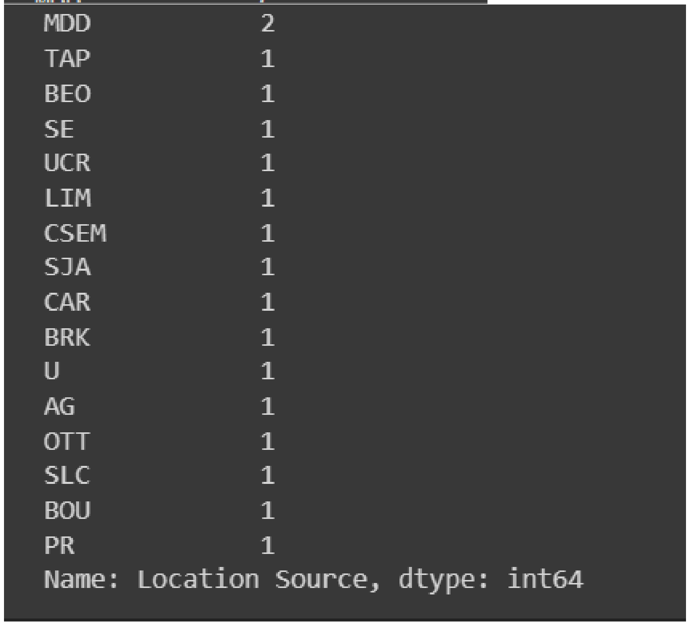


```
US         4781
ISCGEM      885
NC           21
CI           18
GCMT         14
PGC           6
GUC           5
HVO           4
AGS           4
AEIC          4
UNM           3
SPE           3
WEL           3
AK            3
MDD           2
H             2
ATH           2
CASC          1
AEI     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

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

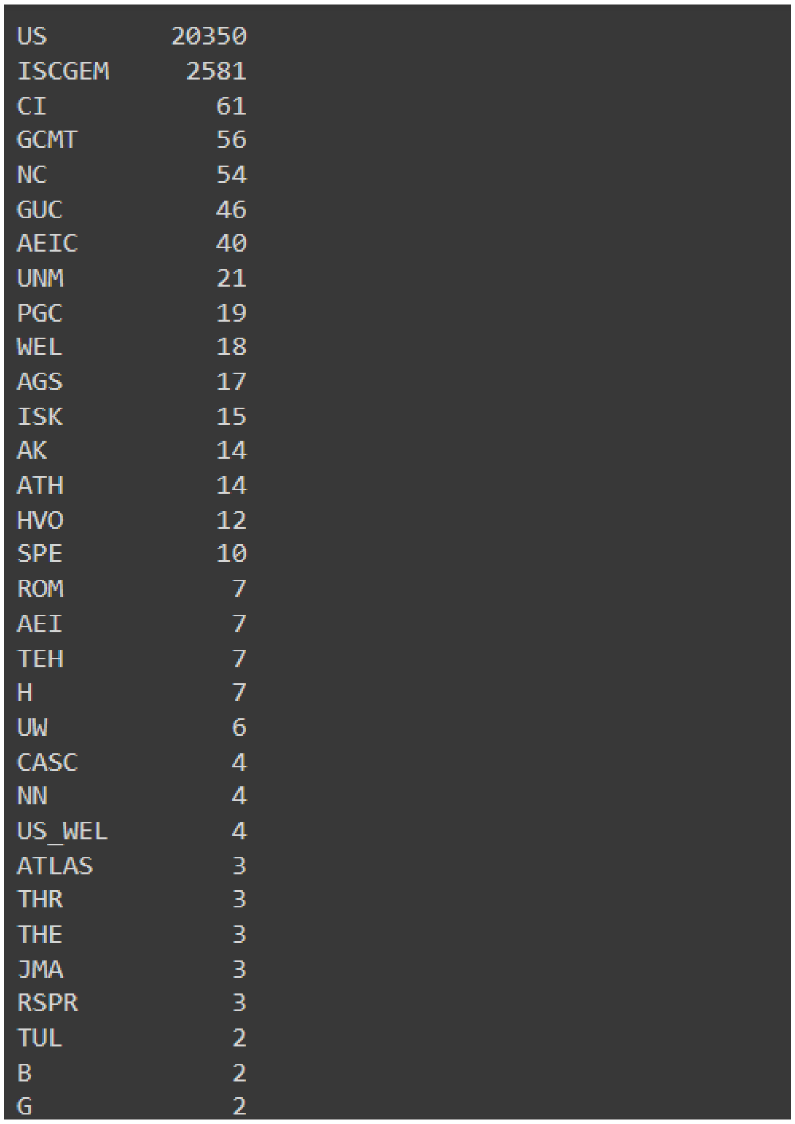


```
US        20350
ISCGEM     2581
CI           61
GCMT         56
NC           54
GUC          46
AEIC         40
UNM          21
PGC          19
WEL          18
AGS          17
ISK          15
AK           14
ATH          14
HVO          12
SPE          10
ROM           7
AEI           7
TEH     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

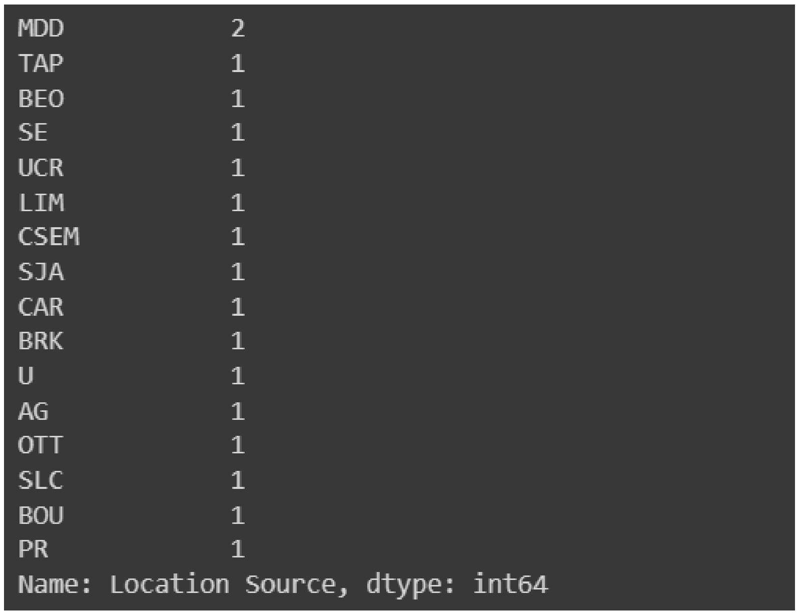


```
MDD           2
TAP           1
BEO           1
SE            1
UCR           1
LIM           1
CSEM          1
SJA           1
CAR           1
BRK           1
U             1
AG            1
OTT           1
SLC           1
BOU           1
PR            1
Name: Location Source, dtype: int64
```

```
Greater_4 = data[data['Magnitude'] > 4]
Greater_4['Location Source'].value_counts()
```



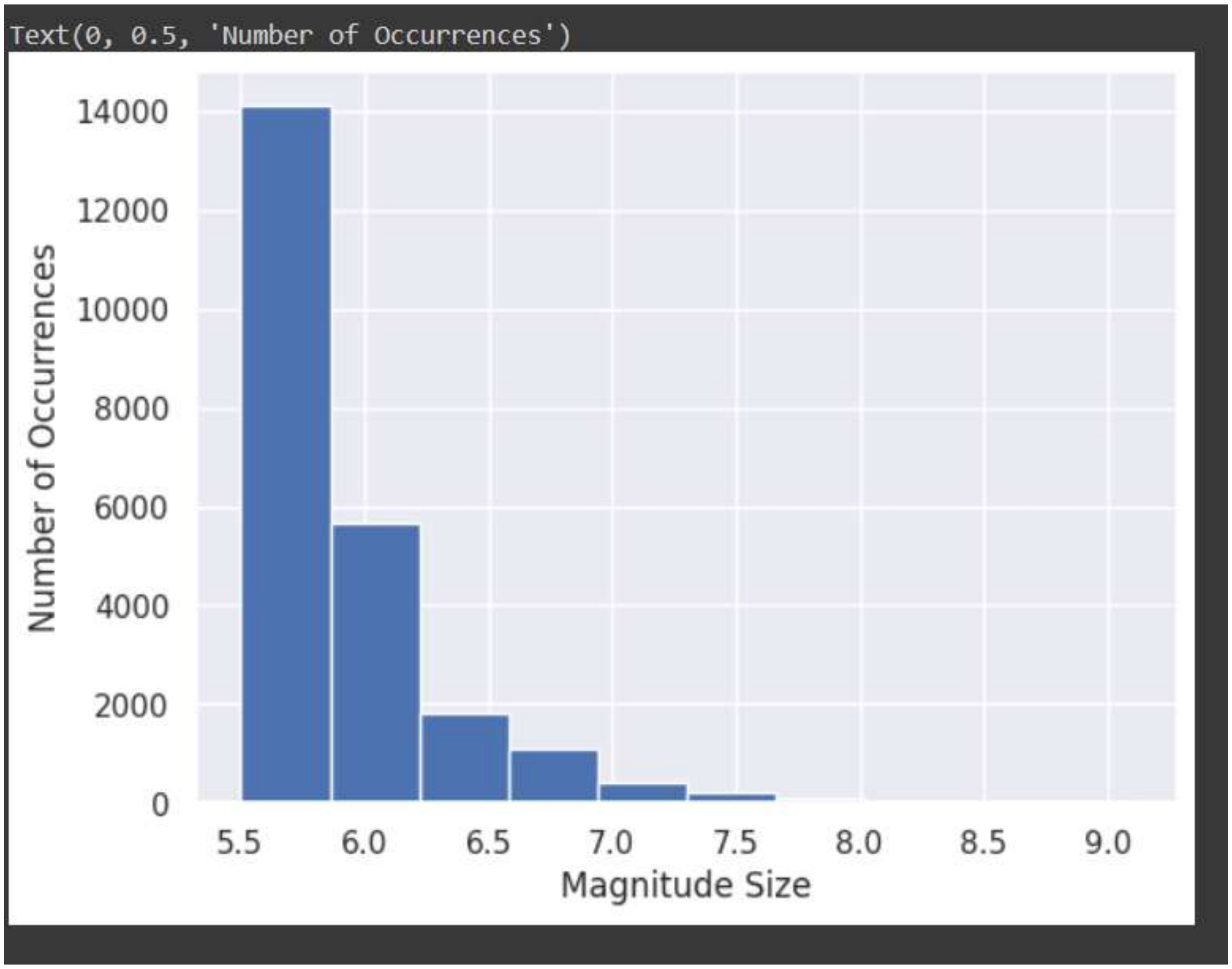
```
US        20350
ISCGEM     2581
CI           61
GCMT         56
NC           54
GUC          46
AEIC         40
UNM          21
PGC          19
WEL          18
AGS          17
ISK          15
AK           14
ATH          14
HVO          12
SPE          10
ROM           7
AEI           7
TEH     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



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

#plotting the histogram for magnitude

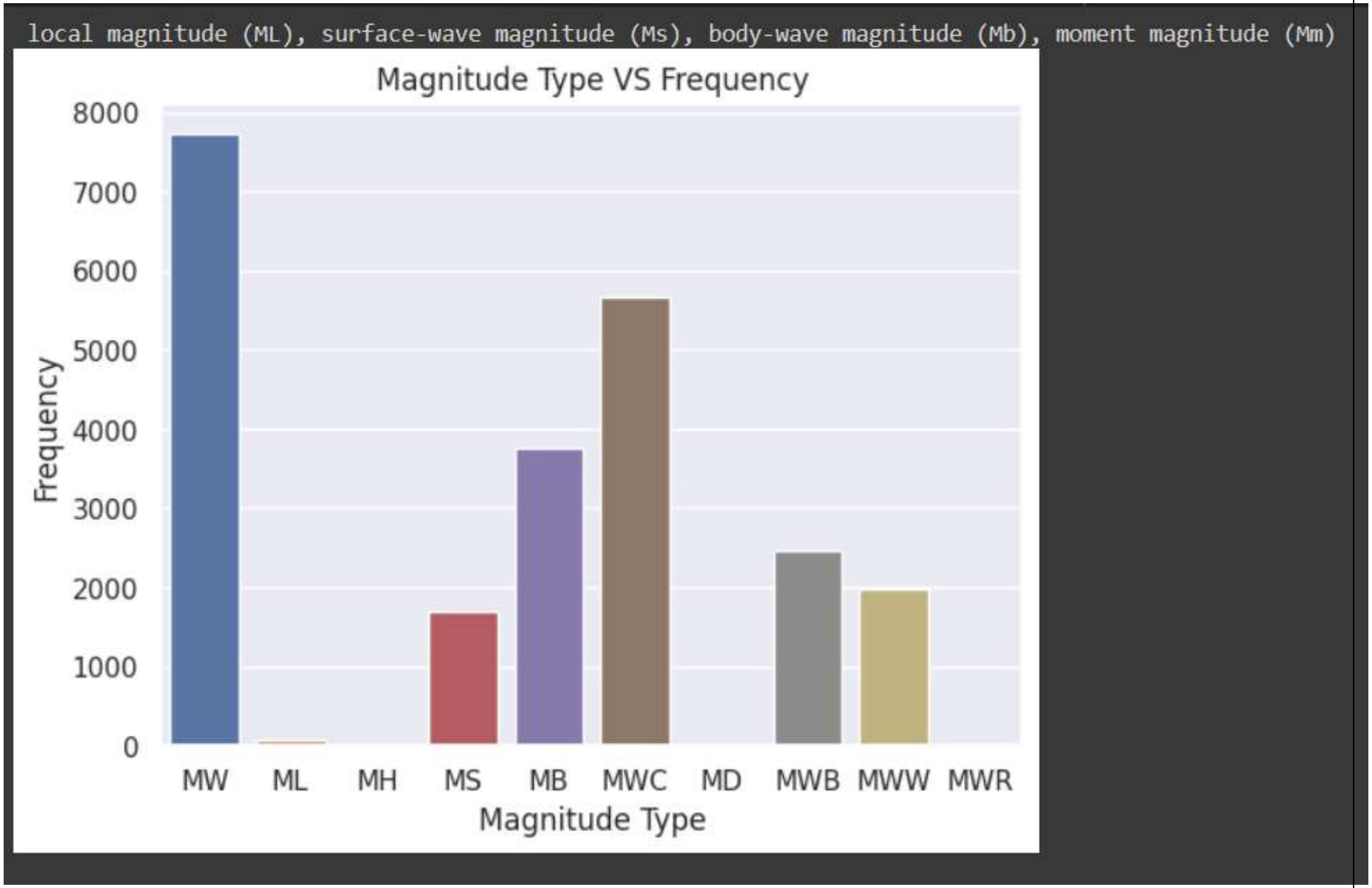
```
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')
```



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

#plotting the count-plot for magnitude type

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


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

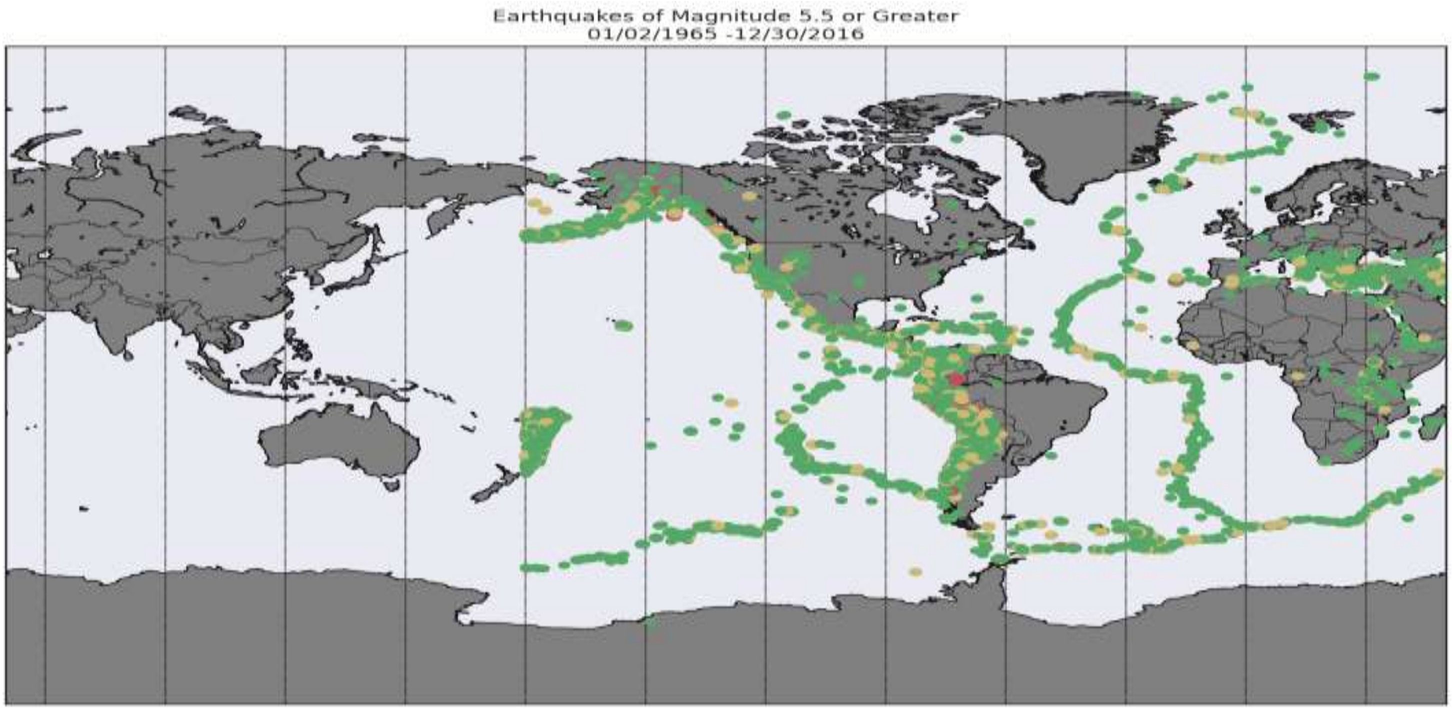
#marking the magnitude

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

#marking different colours of magnitude on basemap plot

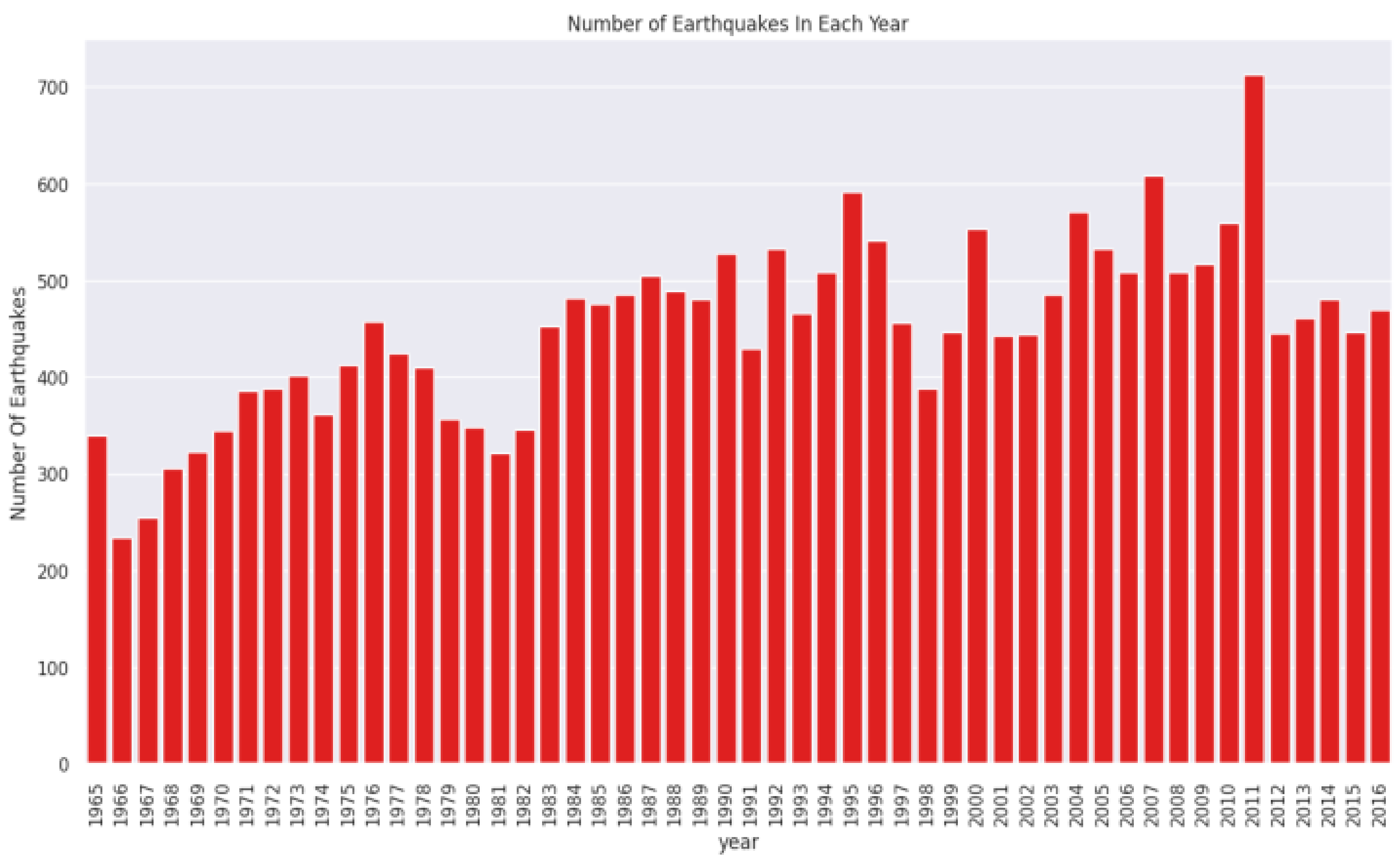
```
plt.figure(figsize=(14,10))
eq_map = Basemap(projection='mill', resolution = 'l', lat_0 = 0, lon_0 = -130)
```

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

#latitude, longitude and magnitude

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

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

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

plt.show()
```



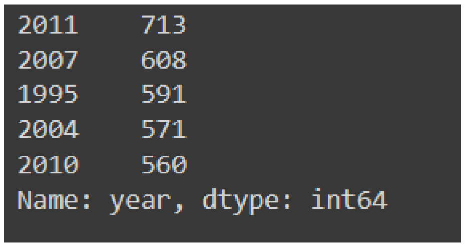
#countplot for no. of earthquakes each year

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



#Ranking the highest no. earthquakes in each year

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

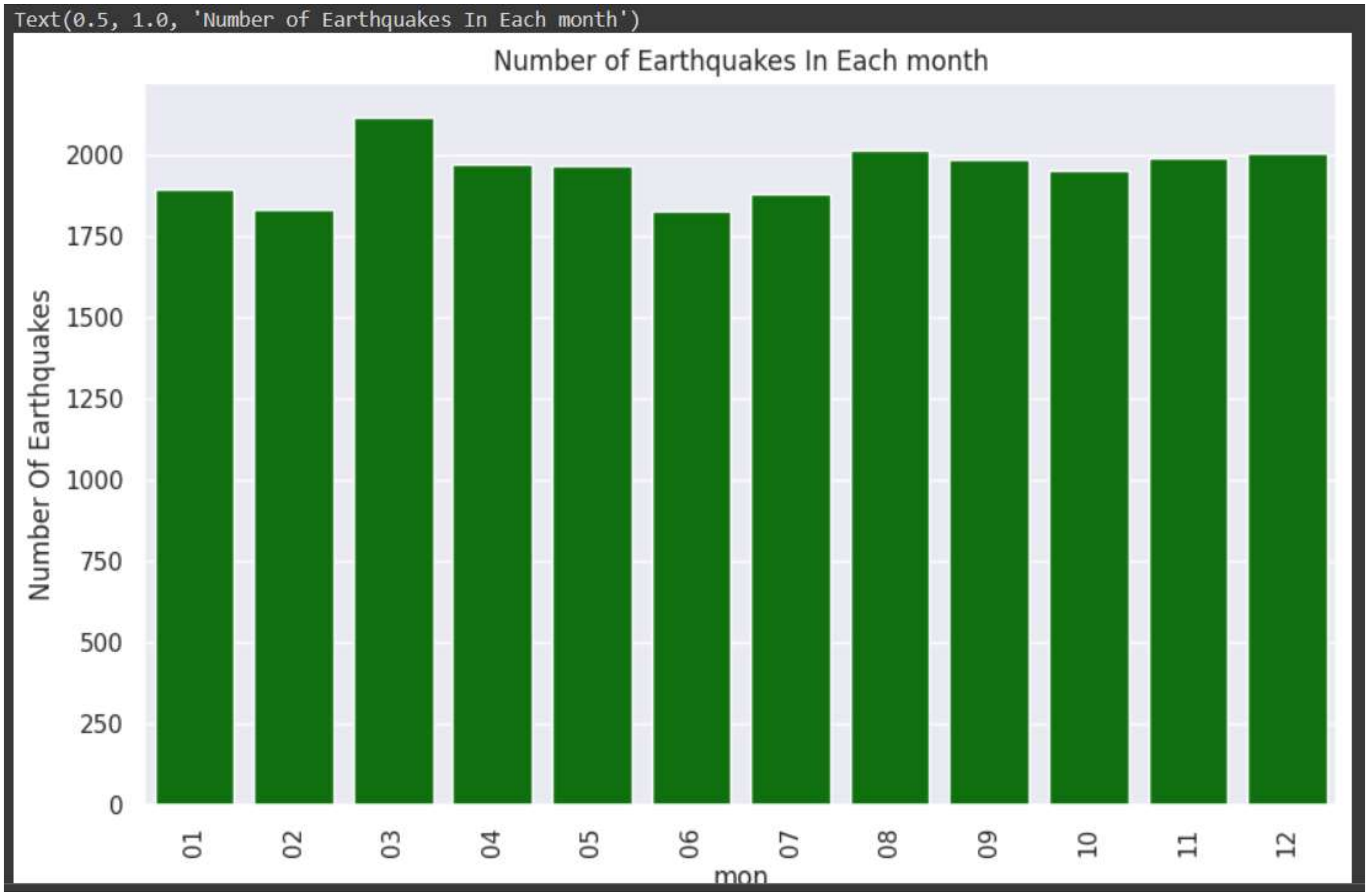


```
2011    713
2007    608
1995    591
2004    571
2010    560
Name: year, dtype: int64
```

#countplot for no. of earthquakes in each month

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

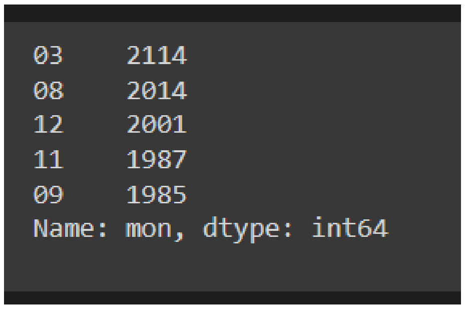
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each month')



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

#Ranking the highest no. earthquakes in each months

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

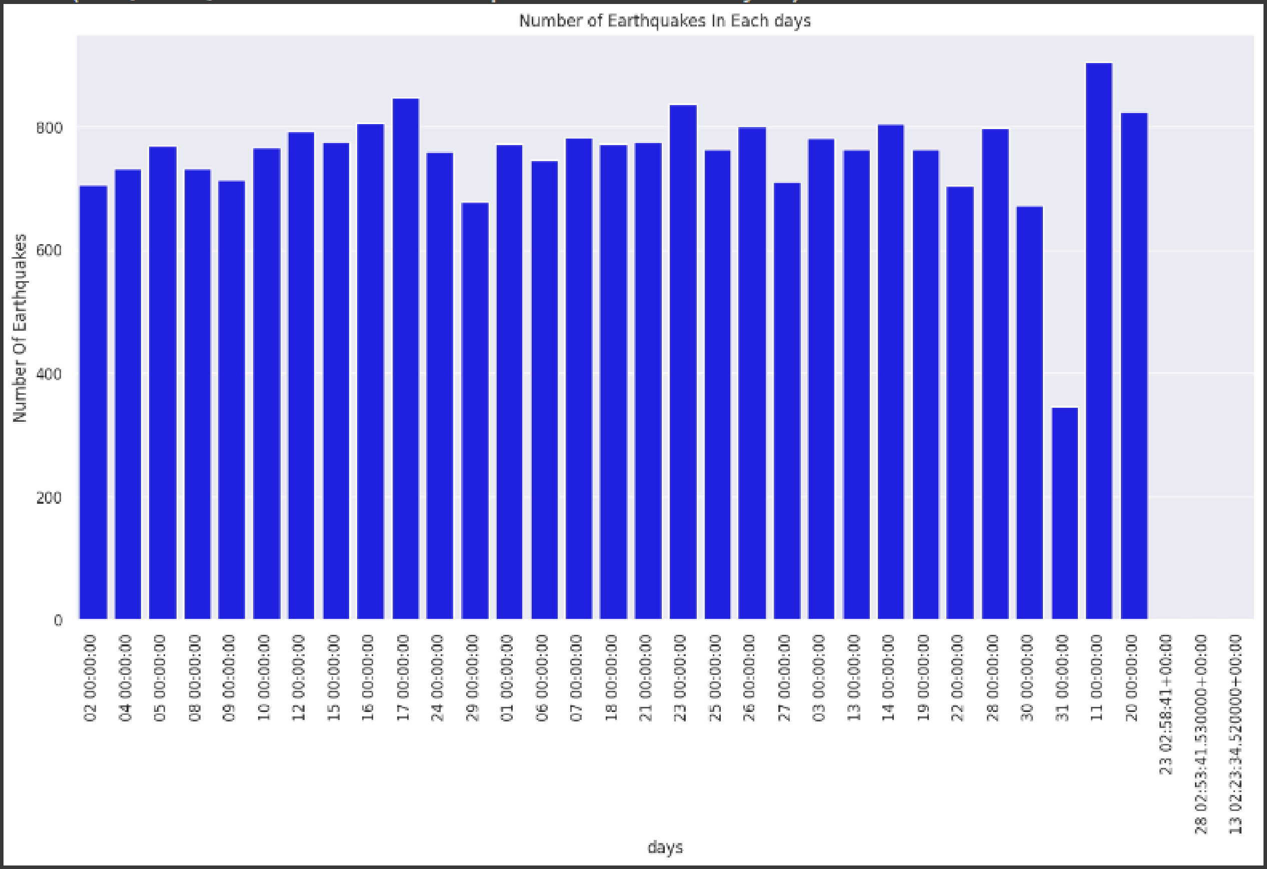


```
03    2114
08    2014
12    2001
11    1987
09    1985
Name: mon, dtype: int64
```

#countplot for no. of earthquakes in each day

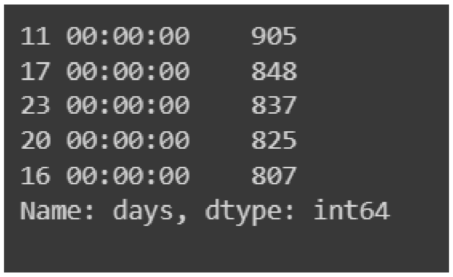
```
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['days'] = data['date'].apply(lambda x: str(x).split('-')[-1])
plt.figure(figsize=(16, 8))
sns.set(font_scale=1.0)
ax = sns.countplot(x="days", data=data, color = "blue")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)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

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



#Ranking the highest no. earthquakes in each day

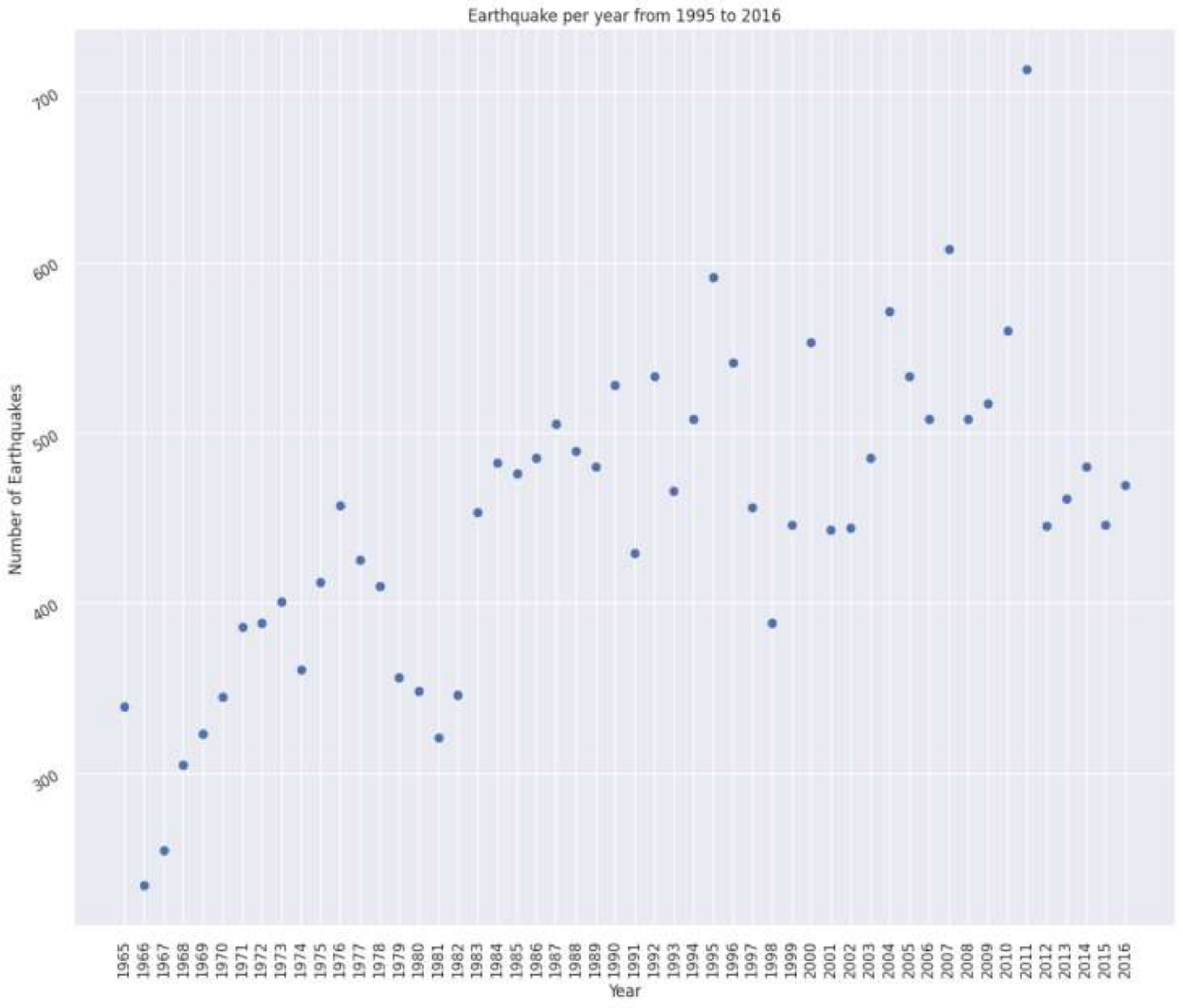
```
data['days'].value_counts()[:5]
```



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

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

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

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

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

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

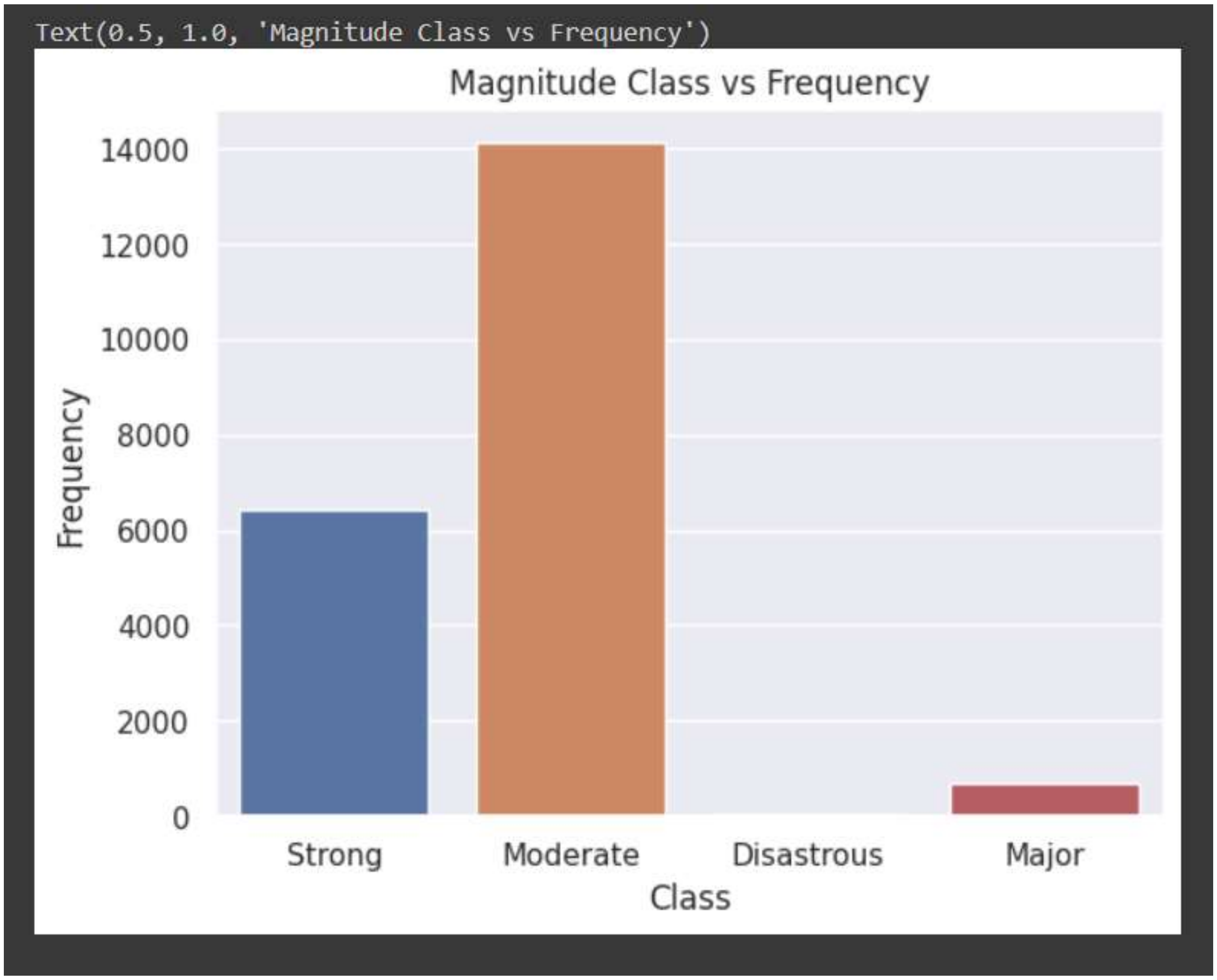


#Classification of magnitude types

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

# Magnitude Class distribution

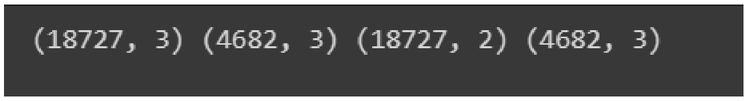
```
sns.countplot(x='Class', data=data)
plt.ylabel('Frequency')
plt.title('Magnitude Class vs Frequency')
```



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

#splitting the data…

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

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(X_train.shape, X_test.shape, y_train.shape,  X_test.shape)
```



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