ARTIFICIAL INTELLIGENCE

Earthquake prediction model using python

PHASE 5

Project Documentation

& Submission

Notebook_link:https://colab.research.google.com/drive/1nc8fHlqm4XHRlSHsoMh_EJQewPhcY5Fc?usp=sharing

Problem statement:

The problem is to develop an earthquake prediction model using a Kaggle dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for global overview, split the data for training and testing and build a neural network model to predict earthquake magnitudes based on the given features.

Understanding

Upon the given problem statement, to create an earthquake prediction model with the given dataset from Kaggle. The given dataset has many features ie, Type, longitude, latitude, etc. There is some relation between these features that can be identified and with that identified relations it is possible to predict the event of happening of a earthquake. We can move through the project with some design thinking …

Design thinking

Feature Exploration: Analyzing the given dataset to understand the relations between the features of the dataset and figure out the key features the are helpful in predicting the earthquake events.

Visualization: With the given features like longitude and latitude we can visualize how the data is being distributed among the geographical space. To visualize data we can use various python packages.

Data splitting: To make use of the data must split the data into training data and testing data. This spiltting is done by us based on the size of the dataset. The test data can be used to evaluate the model trained with training data.

Model development: To predict the magnitude of earthquake we should build a artificial neural network with many layers added to it. Each layer has its own function to do and we use different activation functions at each layer. The final layer is designed to give out the expected output.

Training and evaluation: The training can be done in many ways and once we built the model we can make use of the model for predicting the magnitude of the earthquake. The model is trained with the training data and is expected to give the magnitude of earthquake as a output.

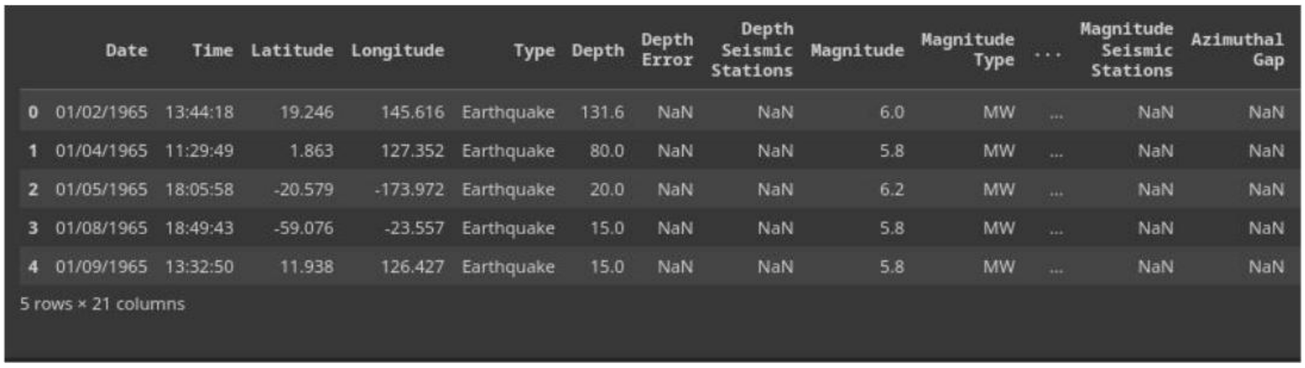
Creating a notebook:

- Open a new notebook in google colab
- Name the notebook as “Earthquake prediction”
- Connect the machine to run the cells
- Upload the dataset in the notebook

Importing the dataset

Using the pandas package we can import the dataset witha built-in function ‘read_csv()’ and passing the name of the dataset as an argument.

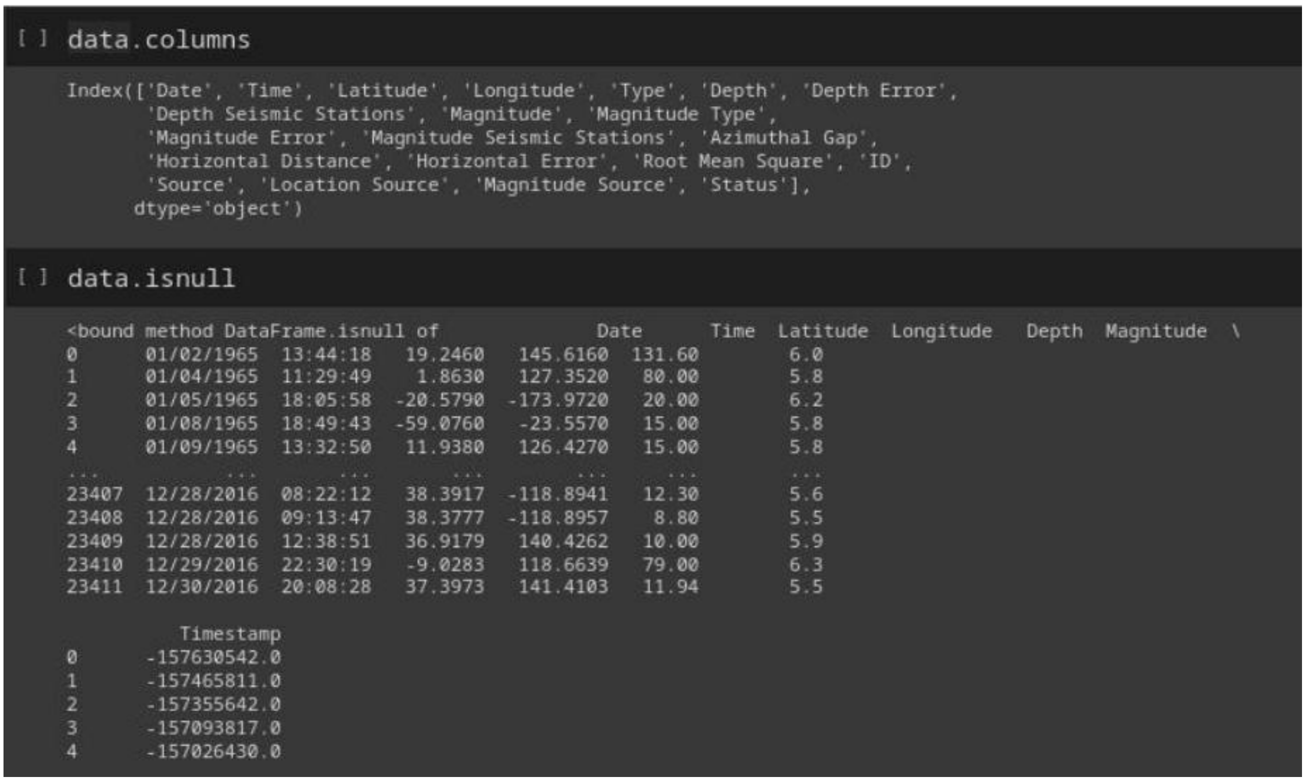
```
#Importing necessary packages
import pandas as pd
import matplotlib.pyplot as plt
import time
import datetime
#Reading the dataset using pandas
data = pd.read_csv('database.csv')
data.head()
```



```
                                                          Depth                                 Magnitude
         Date      Time  Latitude  Longitude        Type  Depth  Depth  Seismic  Magnitude  Magnitude ...  Seismic  Azimuthal
                                                                 Error  Stations                 Type     Stations       Gap
0  01/02/1965  13:44:18    19.246    145.616  Earthquake  131.6    NaN      NaN        6.0         MW ...      NaN       NaN
1  01/04/1965  11:29:49     1.863    127.352  Earthquake   80.0    NaN      NaN        5.8         MW ...      NaN       NaN
2  01/05/1965  18:05:58   -20.579   -173.972  Earthquake   20.0    NaN      NaN        6.2         MW ...      NaN       NaN
3  01/08/1965  18:49:43   -59.076    -23.557  Earthquake   15.0    NaN      NaN        5.8         MW ...      NaN       NaN
4  01/09/1965  13:32:50    11.938    126.427  Earthquake   15.0    NaN      NaN        5.8         MW ...      NaN       NaN
5 rows × 21 columns
```

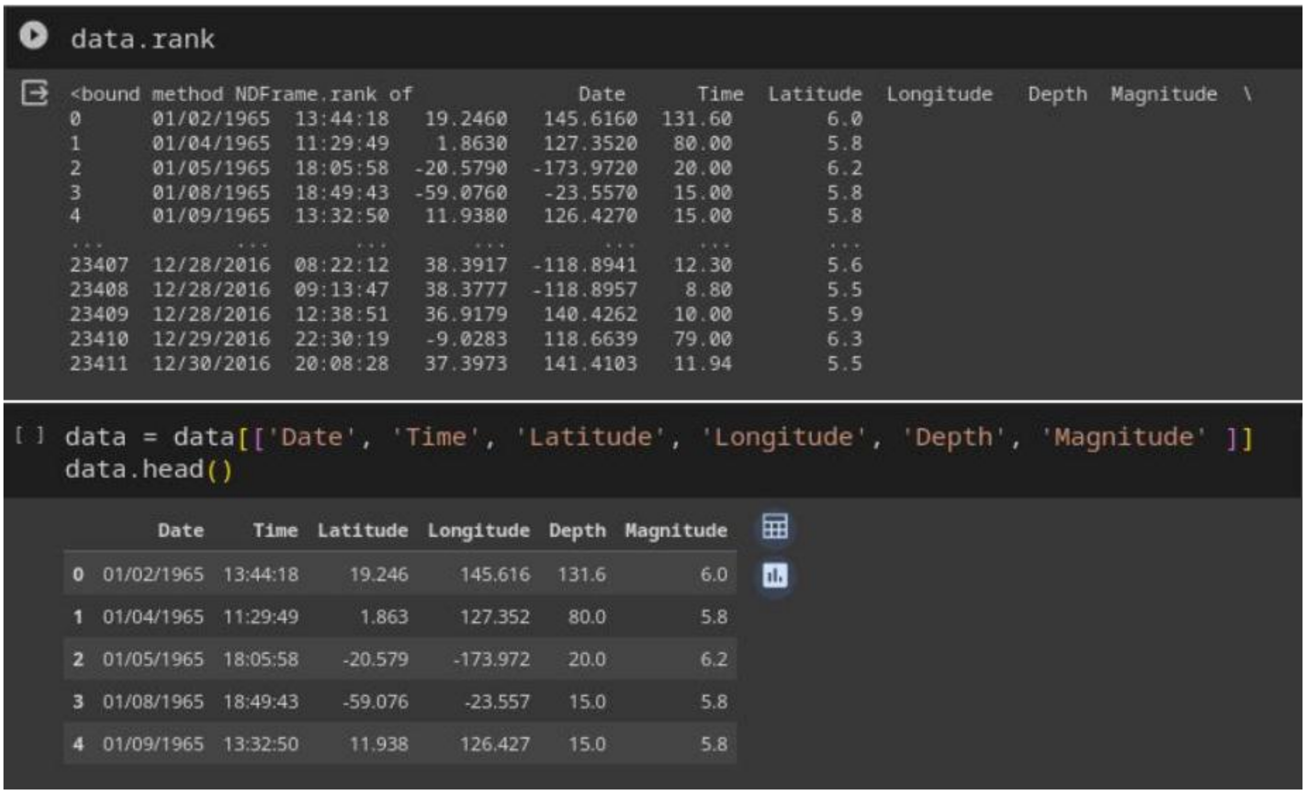
Data analysis

● Before usage of the data in the dataset we need to analyse the data.
● Firstly get to know the features that are important for the model.
● Secondly we need to know which features belongs to input and which features belongs to output



```
[ ] data.columns

    Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error',
           'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',
           'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
           'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID',
           'Source', 'Location Source', 'Magnitude Source', 'Status'],
          dtype='object')
```

```
[ ] data.isnull

    <bound method DataFrame.isnull of               Date      Time  Latitude  Longitude   Depth  Magnitude  \
    0      01/02/1965  13:44:18   19.2460   145.6160  131.60        6.0
    1      01/04/1965  11:29:49    1.8630   127.3520   80.00        5.8
    2      01/05/1965  18:05:58  -20.5790  -173.9720   20.00        6.2
    3      01/08/1965  18:49:43  -59.0760   -23.5570   15.00        5.8
    4      01/09/1965  13:32:50   11.9380   126.4270   15.00        5.8
    ...           ...       ...       ...        ...     ...        ...
    23407  12/28/2016  08:22:12   38.3917  -118.8941   12.30        5.6
    23408  12/28/2016  09:13:47   38.3777  -118.8957    8.80        5.5
    23409  12/28/2016  12:38:51   36.9179   140.4262   10.00        5.9
    23410  12/29/2016  22:30:19   -9.0283   118.6639   79.00        6.3
    23411  12/30/2016  20:08:28   37.3973   141.4103   11.94        5.5

             Timestamp
    0      -157630542.0
    1      -157465811.0
    2      -157355642.0
    3      -157093817.0
    4      -157026430.0
```



```
data.rank

    <bound method NDFrame.rank of               Date      Time  Latitude  Longitude   Depth  Magnitude  \
    0      01/02/1965  13:44:18   19.2460   145.6160  131.60        6.0
    1      01/04/1965  11:29:49    1.8630   127.3520   80.00        5.8
    2      01/05/1965  18:05:58  -20.5790  -173.9720   20.00        6.2
    3      01/08/1965  18:49:43  -59.0760   -23.5570   15.00        5.8
    4      01/09/1965  13:32:50   11.9380   126.4270   15.00        5.8
    ...           ...       ...       ...        ...     ...        ...
    23407  12/28/2016  08:22:12   38.3917  -118.8941   12.30        5.6
    23408  12/28/2016  09:13:47   38.3777  -118.8957    8.80        5.5
    23409  12/28/2016  12:38:51   36.9179   140.4262   10.00        5.9
    23410  12/29/2016  22:30:19   -9.0283   118.6639   79.00        6.3
    23411  12/30/2016  20:08:28   37.3973   141.4103   11.94        5.5
```

```
[ ] data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude' ]]
    data.head()
```

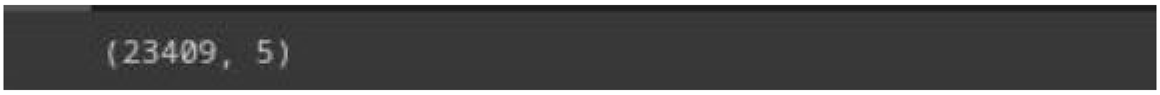
	Date	Time	Latitude	Longitude	Depth	Magnitude
0	01/02/1965	13:44:18	19.246	145.616	131.6	6.0
1	01/04/1965	11:29:49	1.863	127.352	80.0	5.8
2	01/05/1965	18:05:58	-20.579	-173.972	20.0	6.2
3	01/08/1965	18:49:43	-59.076	-23.557	15.0	5.8
4	01/09/1965	13:32:50	11.938	126.427	15.0	5.8

Feature Engineering and Data cleaning

Once the analysis and feature selection is done we can create or manipulate the features according to the need of our problem statement and output needed. Here we create a new feature called ‘Timestamp’ using two features from our dataset ie. ‘Date’ and ‘Time’. The features date and time are string objects which are needed to be converted to object datatype for our convenience.

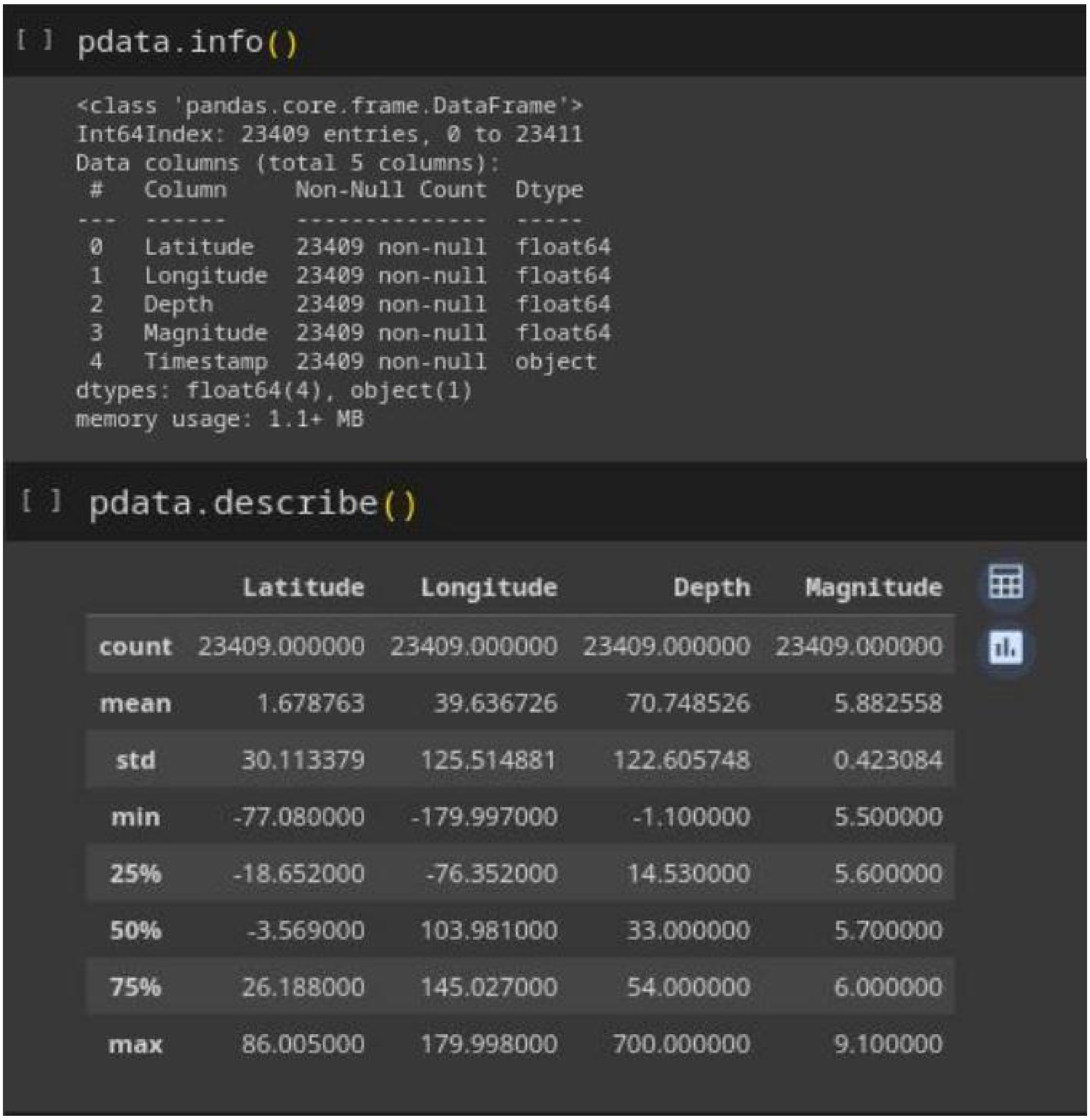
Code snippet

```
#Feature engineering
timestamp = []
for d, t in zip(data['Date'], data['Time']):
try:
ts = datetime.datetime.strptime(d+' '+t, '%m/%d/%Y %H:%M:%S')
timestamp.append(time.mktime(ts.timetuple()))
except ValueError:
timestamp.append('ValueError')#converting array into Series using pandas
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
pdata = data [data['Timestamp']!='ValueError']
#Selection of important features
pdata = pdata[['Latitude', 'Longitude', 'Depth', 'Magnitude','Timestamp']]
pdata.shape
```



```
(23409, 5)
```

Post analysis

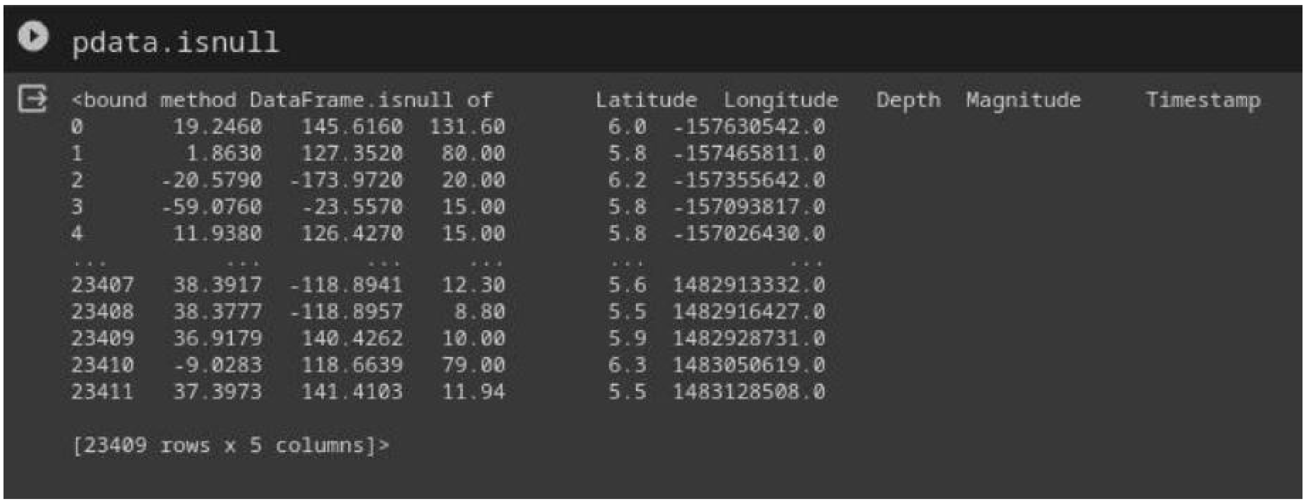


```
[ ] pdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23409 entries, 0 to 23411
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Latitude   23409 non-null  float64
 1   Longitude  23409 non-null  float64
 2   Depth      23409 non-null  float64
 3   Magnitude  23409 non-null  float64
 4   Timestamp  23409 non-null  object
dtypes: float64(4), object(1)
memory usage: 1.1+ MB
```

```
[ ] pdata.describe()
```

	Latitude	Longitude	Depth	Magnitude
count	23409.000000	23409.000000	23409.000000	23409.000000
mean	1.678763	39.636726	70.748526	5.882558
std	30.113379	125.514881	122.605748	0.423084
min	-77.080000	-179.997000	-1.100000	5.500000
25%	-18.652000	-76.352000	14.530000	5.600000
50%	-3.569000	103.981000	33.000000	5.700000
75%	26.188000	145.027000	54.000000	6.000000
max	86.005000	179.998000	700.000000	9.100000



```
pdata.isnull
```

```
<bound method DataFrame.isnull of        Latitude  Longitude   Depth  Magnitude     Timestamp
0       19.2460   145.6160  131.60        6.0  -157630542.0
1        1.8630   127.3520   80.00        5.8  -157465811.0
2      -20.5790  -173.9720   20.00        6.2  -157355642.0
3      -59.0760   -23.5570   15.00        5.8  -157093817.0
4       11.9380   126.4270   15.00        5.8  -157026430.0
...         ...        ...     ...        ...           ...
23407   38.3917  -118.8941   12.30        5.6  1482913332.0
23408   38.3777  -118.8957    8.80        5.5  1482916427.0
23409   36.9179   140.4262   10.00        5.9  1482928731.0
23410   -9.0283   118.6639   79.00        6.3  1483050619.0
23411   37.3973   141.4103   11.94        5.5  1483128508.0

[23409 rows x 5 columns]>
```

Once the data cleaning is done, post analysis is made to know about the data after the pre-processing of the data. The analysis reaveals the reduced dataset with important features in the columns and there are no known null values in the cleansed data.

Hyperparameter tuning

● Hyperparameter tuning is used to improve the model created to increase the accuracy.
● By understanding the analysis made on the dataset we can use different parameter for training the model.
● Finding the right set of hyperparameters can lead to better accuracy, faster training, and more robust models.
● The parameters like the layers, activation functions, learning rate can be adjusted to improve the performance.

params = { ‘neurons’ : [16, 32, 64, 128],
‘batch_size’ : [10, 20, 30, 40],
‘epochs’ : [25, 50, 75, 100],
‘activation’ : [‘ ‘sigmoid’, ‘relu’]
‘optimizer’ : ['SGD', 'Adadelta'],
‘loss’ : [‘mse’, ‘mae’]}

Ensembling techniques

Ensembling techniques in machine learning involve combining multiple individual models (often referred to as base models or weak learners) to create a more robust and accurate composite model. Ensembling can lead to improved predictive performance, reduced overfitting, and increased model stability.

● **Bagging:** Reduces variance by training multiple base models independently on different subsets of the training data. Commonly used with Random Forests.

● **Boosting:** Reduces bias by training a sequence of base models, where each model corrects the errors of the previous one. Includes algorithms like AdaBoost and Gradient Boosting.

● **Stacking:** Combines the predictions of multiple base models using a meta-learner to make the final prediction. Effective for leveraging diverse models.

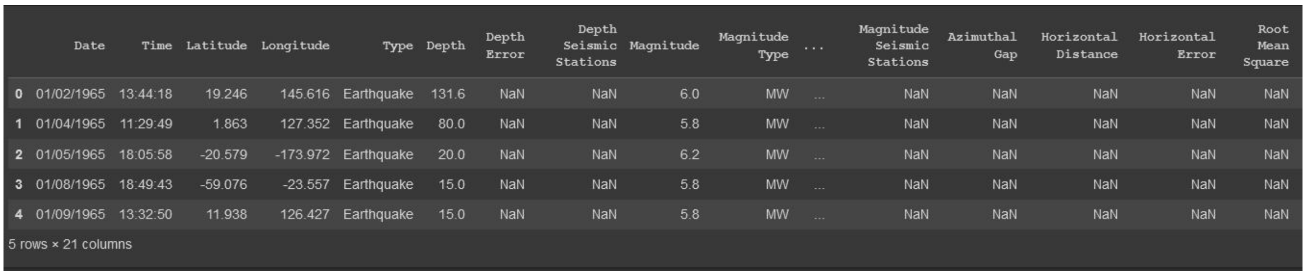
● **Voting:** Combines predictions through majority voting or averaging, often used in ensemble classifiers and regressors.

LOADING THE DATASET

import pandas as pd
data = pd.read_csv('database.csv')

Importing the dataset into a variable using built-in function in pandas package in python.

data.head()



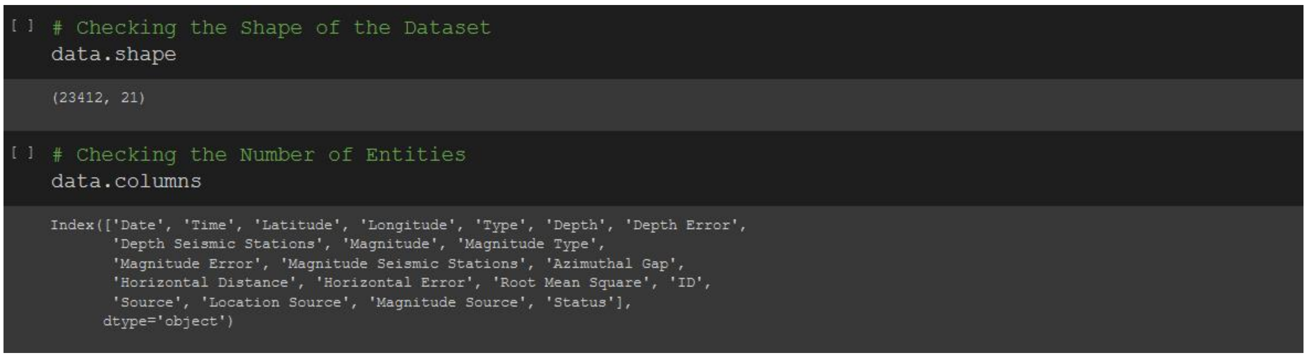
	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN	NaN
1	01/04/1965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

5 rows × 21 columns

- The data is loaded an can be used by accessing the variable give ie.data

PREPROCESSING THE DATA

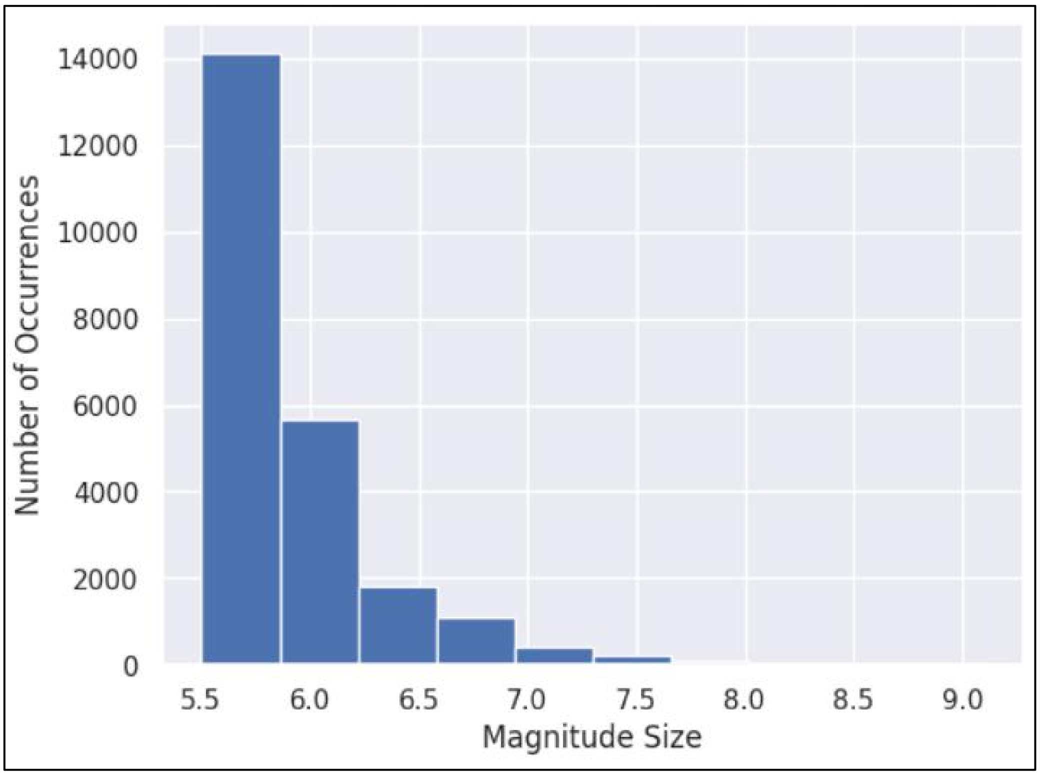
- The imported data should be preprocessed before using in to train the model.
- On analyzing the data we might come to know about the features in the dataset and the relationships between features in the dataset.



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

    (23412, 21)

[ ] # 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')
```

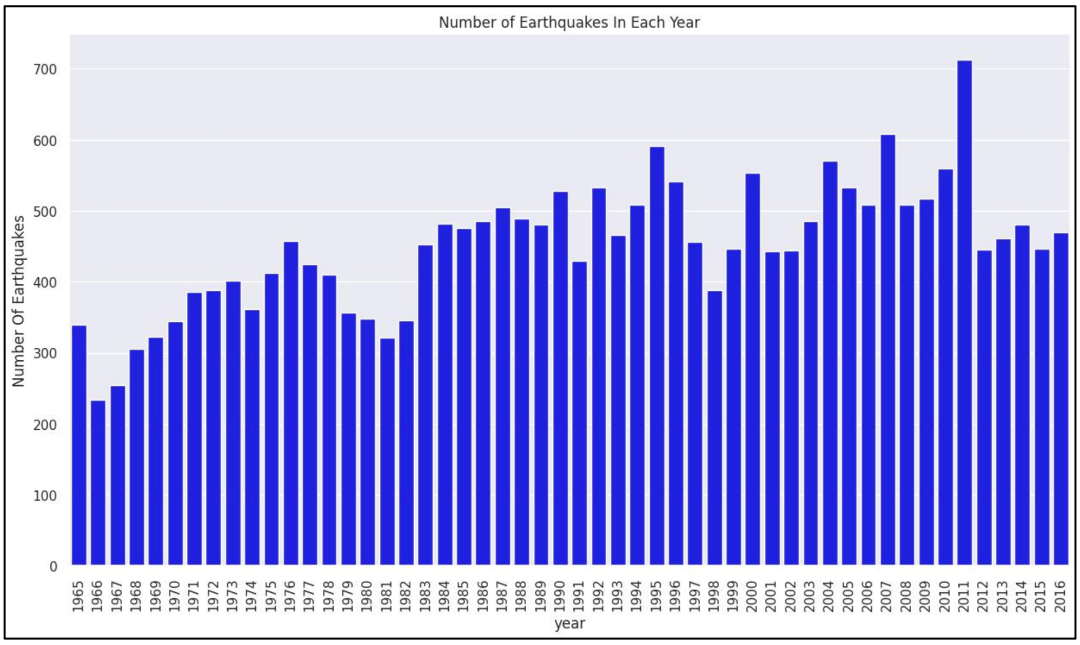
- Using the columns method we get the features in the dataset and with those details the features required for our need can be taken and used.

import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import seaborn as sns
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')



- On analysis on the magnitude feature of the dataset we get the number of occurrences that the earthquake occurred in past days.
- Using the basemap function in the mpl_toolkits package in python we can plot the occurrences of earthquake in real world map to visualize the distribution of the affected areas.
- With this we can analyse which part of the world is most affected and analysis of vulnerable areas can be spotted.

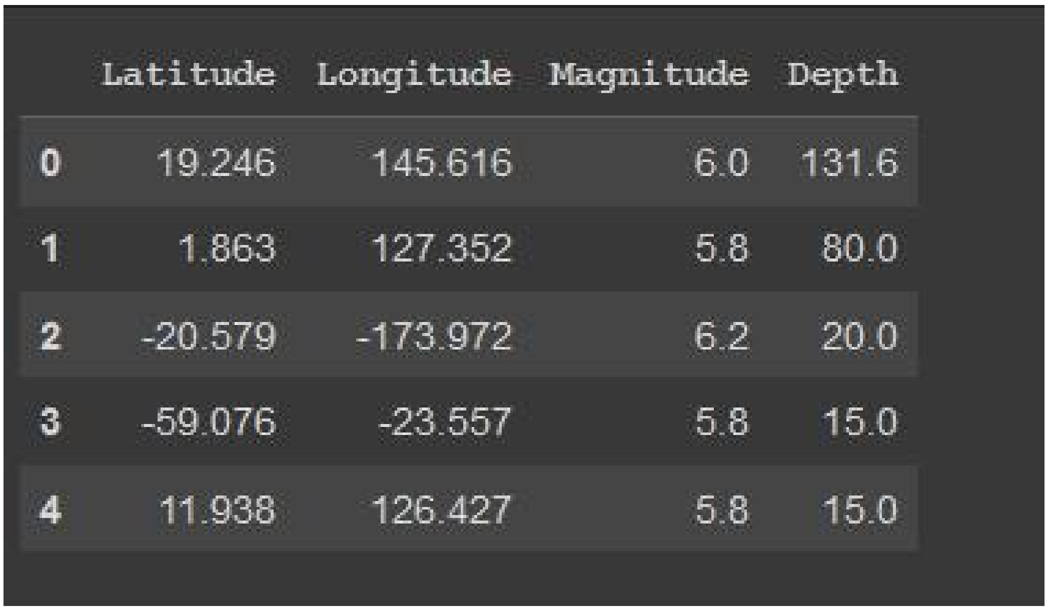
```
import datetime
data['date'] = data['Date'].apply(lambda x: pd.to_datetime(x))
data['year'] = data['date'].apply(lambda x: str(x).split('-')[0])
plt.figure(figsize=(15, 8))
sns.set(font_scale=1.0)
ax = sns.countplot(x="year", data=data, color = "blue")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)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



- Upon another analysis with the date and the earthquake occurrence we come to know that the feature date is very closely related to the the event of earthquake occurrence.

- On the analysis of the dataset given we finally get that the features [‘Latitude’,’Longitude’,’Magnitude’,’Depth’] are the important features in the dateset.

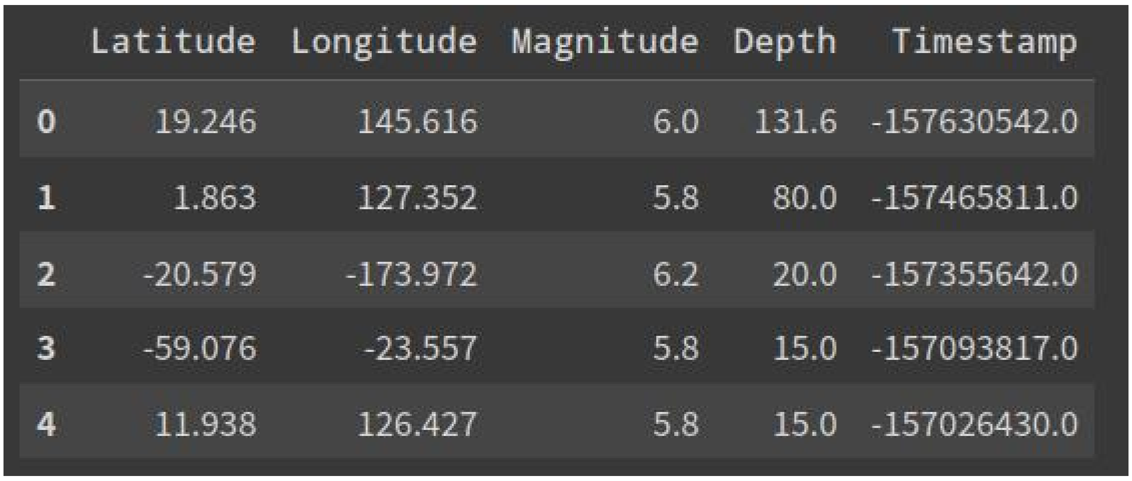
```
tailored_data = data[['Latitude','Longitude','Magnitude','Depth']]
tailored_data.head()
```



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

```
import datetime
import time
timestamp = []
for d, t in zip(data['Date'], data['Time']):
    try:
        ts=datetime.datetime.strptime(d+'    '+t,'%m/%d/%Y %H:%M:%S')
        timestamp.append(time.mktime(ts.timetuple()))
    except ValueError:
        timestamp.append('ValueError')
tailored_data['Timestamp']=pd.Series(timestamp)
tailored_data= tailored_data[tailored_data.Timestamp!='ValueError']
tailored_data.dropna()
tailored_data.head()
```

- We need to add a new feature ie.Timestamp by combining the features Date and Time.
- The Timestamp data is a object datatype and is easier for processing using neural networks.

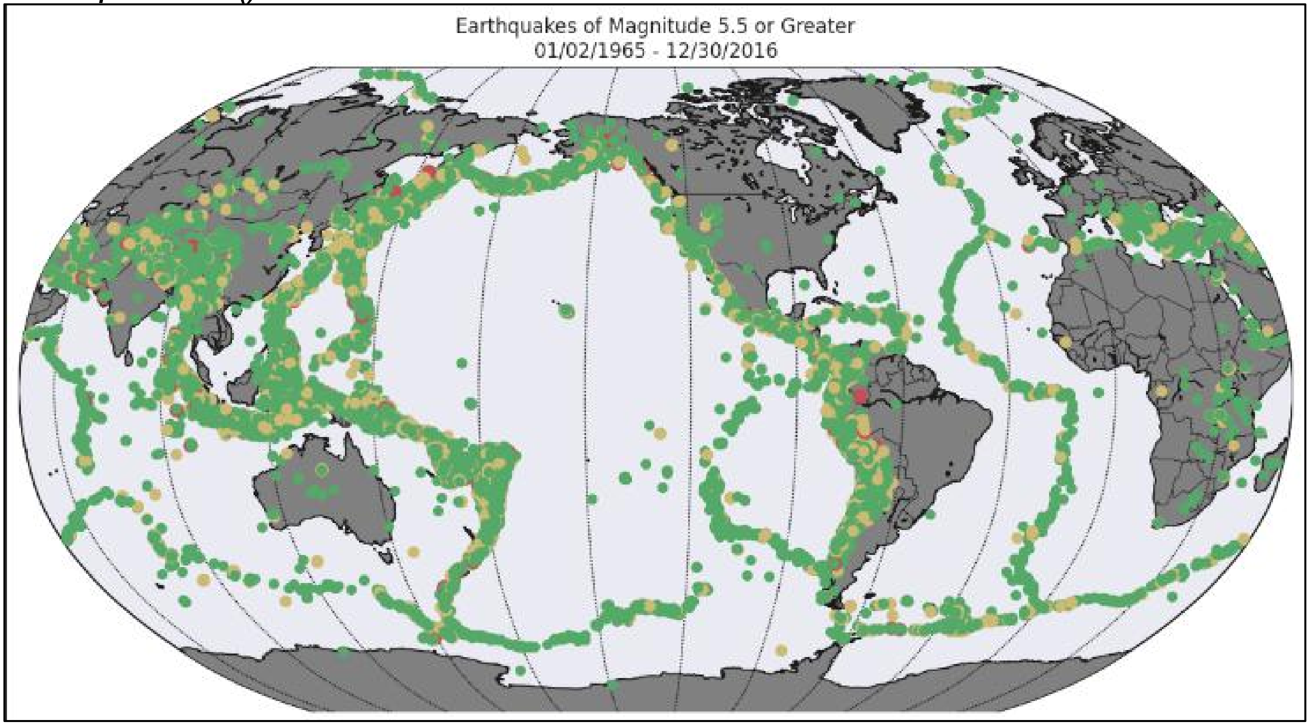


	Latitude	Longitude	Magnitude	Depth	Timestamp
0	19.246	145.616	6.0	131.6	-157630542.0
1	1.863	127.352	5.8	80.0	-157465811.0
2	-20.579	-173.972	6.2	20.0	-157355642.0
3	-59.076	-23.557	5.8	15.0	-157093817.0
4	11.938	126.427	5.8	15.0	-157026430.0

DATA VISUALIZATION

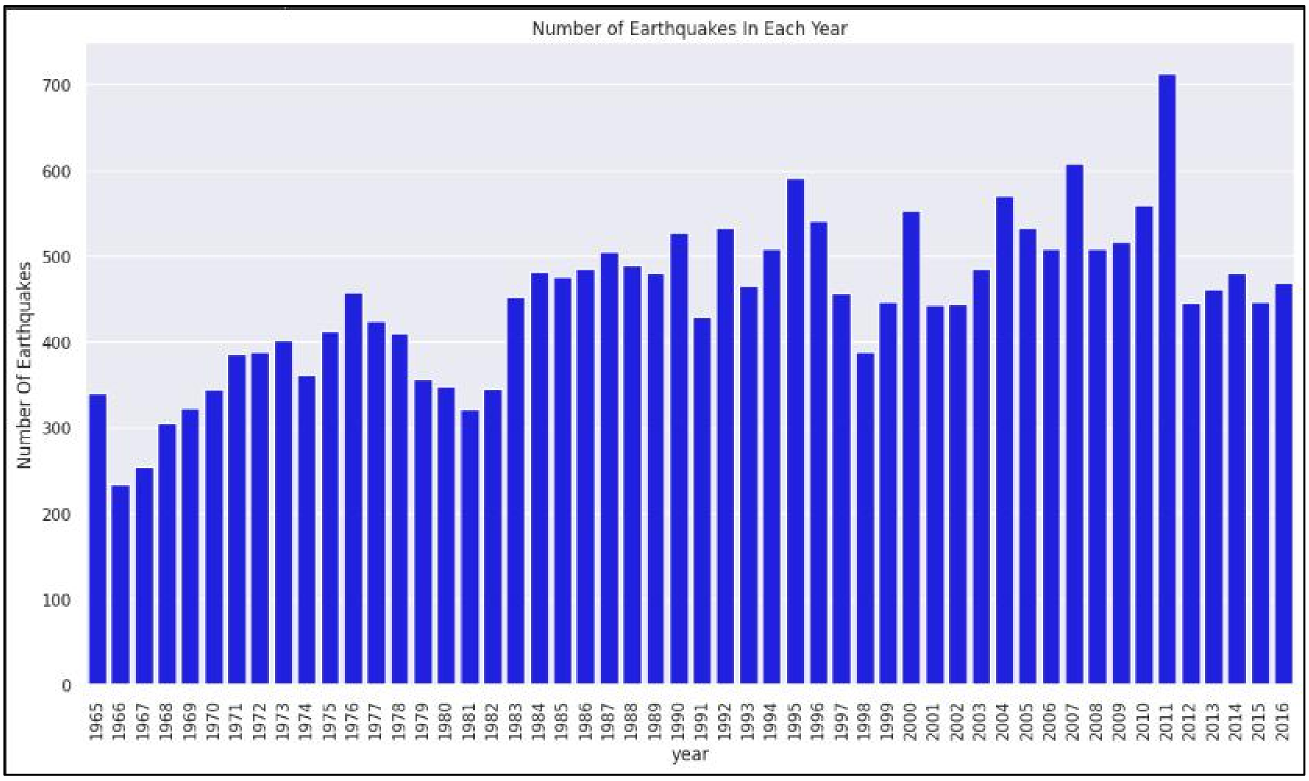
- Visualizing the data will help us to understand the dataset completely.
- With the given dataset we can explore the data by plotting the features in a bar chart or scatter plot.
- By mapping the Longitude and Latitude feature data in a world map we can easily see the distribution of the earthquake occurrence among the different countries in the world.

```
import pandas as pd
import numpy as np
data  = pd.read_csv('database.csv')
data.head()
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import seaborn as sns
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))
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()
```



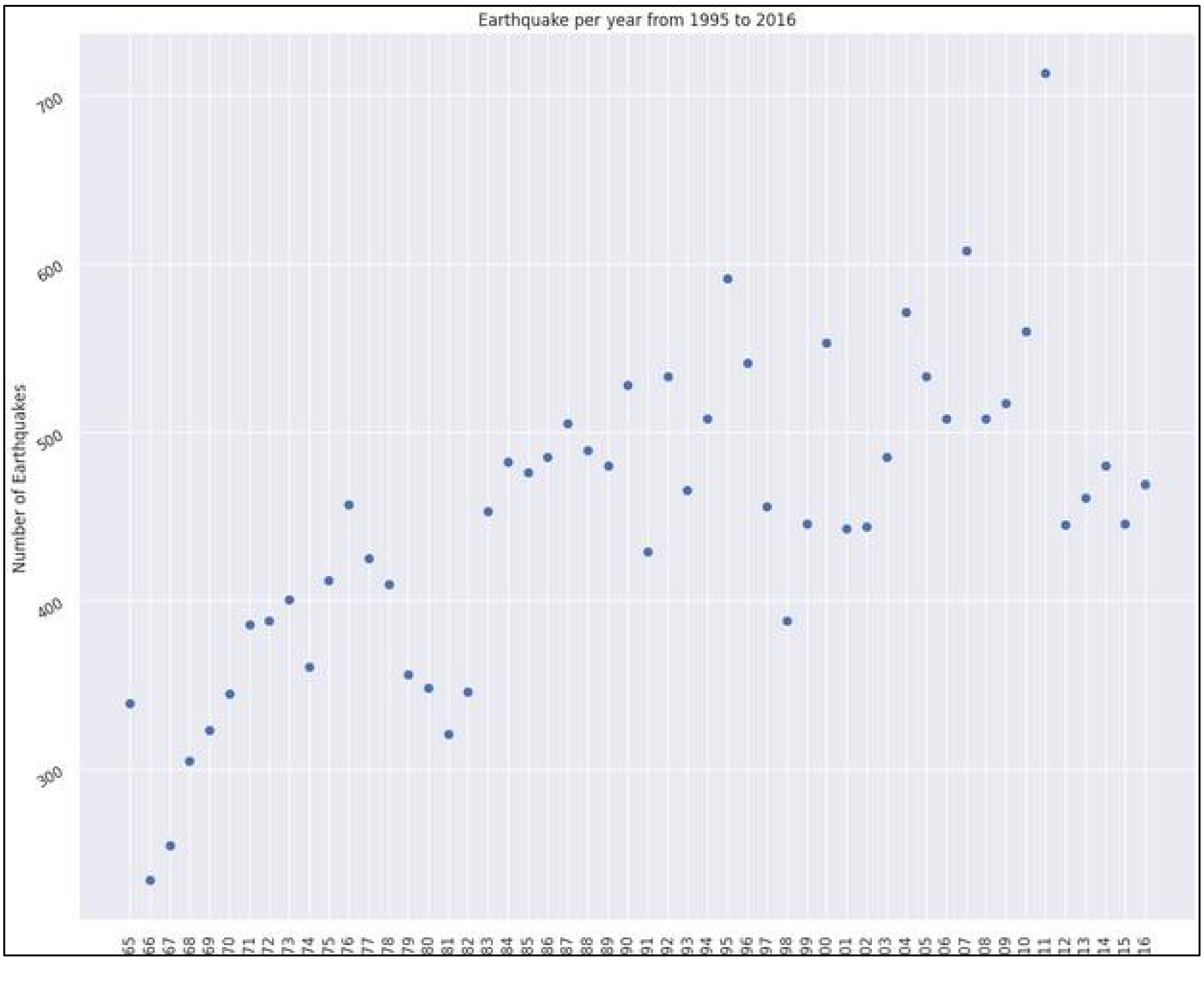
- The Date feature plays an important role in the occurences of the earthquake and plotting the bar graph for Year vs No. of occurrences shows the frequency of earthquake in each year.

```
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')
```



```
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()
```



DATA SPLITTING FOR TRAINING AND TRAINING

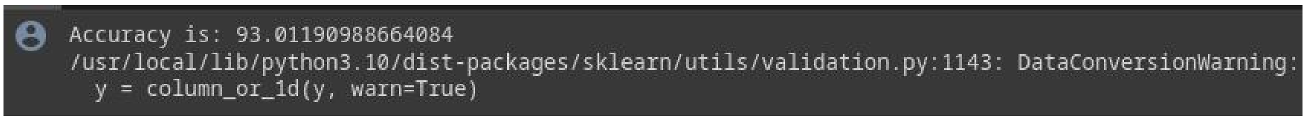
- Among the important features we need to identify the features that can be used as input for the model and the features that can be used as a target/output for the model we are going to training.
- Once the input and output features are identified the data futher neeeded to be split into training data and testing data.
- We here split 80% of the data as training data and remainig 20% data as testing data which is used for validation part in the process.

```
x = tailored_data[['Longitude','Latitude','Timestamp','Depth']]
y = tailored_data[['Magnitude']]
from sklearn.model_selection import train_test_split as tts
x_train,x_test,y_train,y_test=tts(x,y,test_size=0.2,random_state =42)
```

- Here the x variable has the features that can be used as input for the model we are going to used.
- The Y variable has the feature that is used as a target for the model ie. which is predicted by the model.
- The spliiting is done by a built-in function in the sklearn.model_selection that is called as train_test_split().
- Using random state as a paramater for function will make sure that the same split for the data is used throughout the program even we run it again.

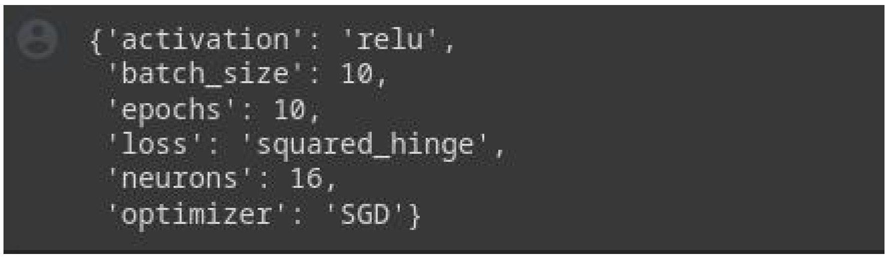
Model building and Evaluvation

```
import sklearn
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
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: 93.01190988664084
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning:
  y = column_or_1d(y, warn=True)
```

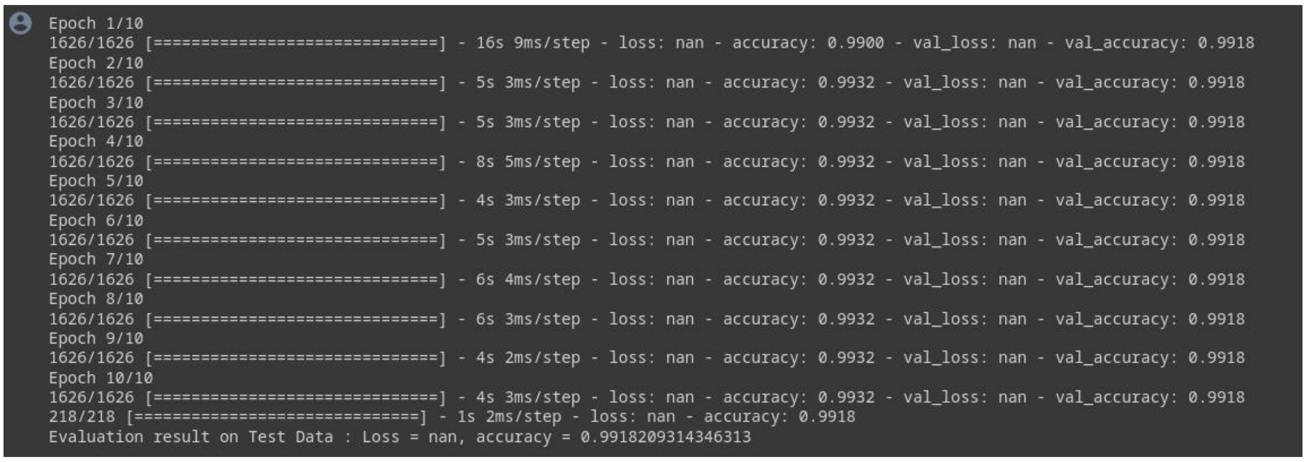
```
from keras.models import Sequential
from keras.layers import Dense
# 3 dense layers, 16, 16, 2 nodes each
def create_model(neurons, activation, optimizer, loss):
    model = Sequential()
    model.add(Dense(neurons, activation=activation, input_shape=(3,)))
    model.add(Dense(neurons, activation=activation))
    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']
}
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)
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'}
```

- Using hyperparameter tuning we get the best parameters for the model we desired to build
- We will use these best parameters to build our artificial neural network and train using those parameters.
- Then with the trained the model we evaluvate it's performance using various metrics.

```
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

Thus a good accuracy score is achieved in earthquake prediction model using python. We made use of dataset from the Kaggle platform and performed analysis on the dataset. Visualizing the dataset gave us the relation between the features in the dataset and mapping the occurrences over the world map. On inference from the dataset given we chose the features we need to train the artificial neural network. We also created a new feature named 'Timestamp' which combined the data from the 'Date' & 'Time' features from the dataset. Once the feature engineering is done we need to split the dataset into input data and output data. On the split data we again split the dataset into training data and testing data using built-in function in python. Using hyperparameter tuning we chose the best parameters for creating the model architecture. Using the build artificial neural network model we trained the model and evaluvated it successfully!