Project title - Earthquake Prediction Model using Python

Phase 2 – Innovation

Introduction:

- The data cleaning and data analysis mentioned in the phase 1 design thinking steps are implemented in this phase 2 innovation file.
- The workspace used is "Google colab".

Colab Notebook link :Notebook Link

Importing the dataset:

```
[95] #importing the required libraries
     import numpy as np
     import pandas as pd
```

```
#importing the dataset
df = pd.read_csv("/content/drive/MyDrive/database.csv")
```

```
df.head(5)
```

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

5 rows × 21 columns

Feature Engineering:

Data cleaning and analysis:

- Firstly the data is analysed before applying feature engineering.

  # Statistical information about the dataset is given by

  df.describe()

  #Printing the columns of the dataset

  df.columns

#Printing the shape of the dataset

df.shape

#Checking for duplicated values

df.duplicated()

#Printing the numerical and categorical features from the dataset

```
[71] #Printing the numerical and categorical features
     # 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', 'Magnitude
```

#Printing the number of unique values in the numerical data values

```
0s   #Checking number of unique 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
```

Printing the number of unique values in the categorical data values

```
0s   [73] #Checking number of unique 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
```

Handling missing values:

#Finding missing values in our dataset

```
[74] #Finding number of missing values in each column
     print(df.isnull().sum())

Date                             0
Time                             0
Latitude                         0
Longitude                        0
Type                             0
Depth                            0
Depth Error                  18951
Depth Seismic Stations       16315
Magnitude                        0
Magnitude Type                   3
Magnitude Error              23085
Magnitude Seismic Stations   20848
Azimuthal Gap                16113
Horizontal Distance          21808
Horizontal Error             22256
Root Mean Square              6060
ID                               0
Source                           0
Location Source                  0
Magnitude Source                 0
Status                           0
dtype: int64
```

#Finding the percentage of missing values in each column.

```
[75] #Finding the percentage of missing values in each column
     miss_percent = (df.isnull().sum()/df.shape[0])*100
     print(round(miss_percent,2))

Date                           0.00
Time                           0.00
Latitude                       0.00
Longitude                      0.00
Type                           0.00
Depth                          0.00
Depth Error                   80.95
Depth Seismic Stations        69.69
Magnitude                      0.00
Magnitude Type                 0.01
Magnitude Error               98.60
Magnitude Seismic Stations    89.05
Azimuthal Gap                 68.82
Horizontal Distance           93.15
Horizontal Error              95.06
Root Mean Square              25.88
ID                             0.00
Source                         0.00
Location Source                0.00
Magnitude Source               0.00
Status                         0.00
dtype: float64
```

Feature selection:

Since we are interested only in predicting earthquake ,we restrict our instances to contain only the type attribute with earthquake value only.

```
#Checking the number of instances in each class of the type attribute
df['Type'].value_counts()
```

```
Earthquake           23232
Nuclear Explosion      175
Explosion                4
Rock Burst               1
Name: Type, dtype: int64
```

Applying feature engineering:

```
#Selecting the rows which contains the column name "type" with value "Earthquake"
df.loc[df['Type']=="Earthquake"]
```

	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	...
...	...	...	...	...	...	...	...	...	...	...	...
23407	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	1.2	40.0	5.6	ML	...
23408	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	2.0	33.0	5.5	ML	...
23409	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	1.8	NaN	5.9	MWW	...
23410	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	1.8	NaN	6.3	MWW	...
23411	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	2.2	NaN	5.5	MB	...

23232 rows × 21 columns

Scaling down the features by dropping unwanted columns:

```
#Dropping unnecessary columns and missing values from our dataset
df.drop(['Type', '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'],axis=1,inplace=True)
```

```
[83] #Printing the shape of the dataset after applying feature engineering
print(df.shape)
```

```
(23412, 6)
```

Independent features and the target feature:

```
#Significant columns
df.columns
```

```
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude'], dtype='object')
```

Checking for null values again:

```
#Checking for null values after feature engineering
df.isnull().sum()
```

```
Date         0
Time         0
Latitude     0
Longitude    0
Depth        0
Magnitude    0
dtype: int64
```

Feature Creation technique:

- We convert given Date and Time to Unix time which is in seconds and a numeral.
- This can be easily used as input for the network we built.

```
# We convert given Date and Time to Unix time which is in seconds and a numeral.
import datetime
import time

timestamp = []
for d, t in zip(df['Date'], df['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')
timeStamp = pd.Series(timestamp)
df['Timestamp'] = timeStamp.values
```

```
[88] df.drop(['Date', 'Time'], axis=1,inplace=True)
     df = df[df.Timestamp != 'ValueError']
     print(df.head(5))
```

```
   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 Analysis after feature engineering:

```
[90] df.head(5)
```

	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

```
df.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

```
[93] df.columns

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

```
df.nunique()

Latitude      20673
Longitude     21472
Depth          3485
Magnitude        64
Timestamp     23390
dtype: int64
```

Hyperparameter Tuning:

- Number of neurons
- Activation function
- Number of hidden layers
- Batch size
- Epochs
- Optimizers
- Learning rate
- Callbacks
- Dropout layer

- The hyperparameters to tune are the number of neurons, activation function, optimizer, learning rate, batch size, and epochs.

- The second step is to tune the number of layers.
- The first hyperparameter to tune is the number of neurons in each hidden layer.
- The number of neurons should be adjusted to the solution complexity.
  For example, range is set to be from 10 to 100.
- The activation function decides how to compute the input values of a layer into output values.
- Optimizer is very important to achieve the possible highest accuracy or minimum loss. There are 7 optimizers to choose from.
- One of the hyperparameters in the optimizer is the learning rate.
- Learning rate controls the step size for a model to reach the minimum loss function.
- The number of times a whole dataset is passed through the neural network model is called an epoch.
- Suitable number of epochs between 20 to 100. An example is as follows:

```
params_nn ={
    'neurons': (10, 100),
    'activation':(0, 9),
    'optimizer':(0,7),
    'learning_rate':(0.01, 1),
    'batch_size':(200, 1000),
    'epochs':(20, 100)
}
```

Tuning the layers using regularization technique:

- Inserting regularization layers in a neural network can help prevent overfitting.
- The dropout layer, as its name suggests, randomly drops a certain number of neurons in a layer. The dropped neurons are not used anymore. The rate of how much percentage of neurons to drop is set in the dropout rate.

An example

```
params_nn2 ={
    'neurons': (10, 100),
    'activation':(0, 9),
    'optimizer':(0,7),
    'learning_rate':(0.01, 1),
    'batch_size':(200, 1000),
    'epochs':(20, 100),
    'layers1':(1,3),
    'layers2':(1,3),
    'normalization':(0,1),
    'dropout':(0,1),
    'dropout_rate':(0,0.3)
}
```

Ensembling Techniques:

- Neural network models are nonlinear and have a high variance, which can be frustrating when preparing a final model for making predictions.
- Ensemble learning combines the predictions from multiple neural network models to reduce the variance of predictions and reduce generalization error.
- K-Fold cross validation ensembling is best suited for the neural networks model. Various other techniques are as follows

- **Varying Training Data**
  - k-fold Cross-Validation Ensemble
  - Bootstrap Aggregation (bagging) Ensemble
  - Random Training Subset Ensemble
- **Varying Models**
  - Multiple Training Run Ensemble
  - Hyperparameter Tuning Ensemble
  - Snapshot Ensemble
  - Horizontal Epochs Ensemble
  - Vertical Representational Ensemble
- **Varying Combinations**
  - Model Averaging Ensemble
  - Weighted Average Ensemble
  - Stacked Generalization (stacking) Ensemble
  - Boosting Ensemble
  - Model Weight Averaging Ensemble