PROJECT TITLE - EARTHQUAKE PREDICTION MODEL USING PYTHON

PHASE 5 – PROJECT DOCUMENTATION & SUBMISSION

Colab Notebook link: Click here

PROBLEM STATEMENT:

The primary objective is to explore and understand key features in earthquake data, create visualizations for global earthquake distribution, split the data into training and testing sets, and build a neural network model to predict earthquake magnitudes based on given features.

DESIGN THINKING:

Data Source Selection:

- The first step is to import the earthquake dataset downloaded from Kaggle.
- The dataset contains features such as date, time, latitude, longitude, depth, and magnitude.

Data Preprocessing:

- Handle Missing Data: If missing data values are present in the dataset , then try to remove or imputate it .
- Data Formatting: Convert data types as needed, especially date and time features, which should be converted into datetime objects for analysis.
- Outlier Handling: Identify outliers in the dataset, which could adversely affect model performance.

Feature Exploration:

- Exploratory Data Analysis (EDA) should be conducted to understand the distribution, central tendencies, and variability of each feature.
- Identification of target variable in our dataset .
- Calculate and visualize correlations between features and the target variable (earthquake magnitude) to identify relationships.

Visualization:

- Data visualization libraries such as matplotlib and seaborn is used to build histograms, scatter plots and correlation matrices to provide clearer understanding of the features in the dataset.

- A world map visualization depicting the frequency distribution of earthquakes globally is useful for identifying earthquake prone regions visually.

Data Splitting:

- The dataset is split into training and testing sets.
- A common practice is to allocate 80% of the data for training and 20% for testing .

Model Development:

- **Neural Networks** machine learning model is used to predict the earthquake magnitudes .
- The neural network architecture should be designed by specifying the number of hidden layers, units, activation functions, and any regularization techniques (e.g., dropout) to be used.

Training and Evaluation:

- Train the neural network model using the training data and set suitable hyperparameters.
- Monitor the training process, track metrics (e.g., mean squared error , accuracy , precision ,correlation matrix), and visualize training/validation loss to check for overfitting.
- Evaluate the performance of the model using appropriate evaluation metrics such as Mean Squared Error and R-squared.

PHASES OF DEVELOPMENT:

1. **Phase 1:**

   In phase 1, the problem statement and design thinking steps are clearly defined.
2. **Phase 2:**

   In this phase 2, techniques to improve model's performance such as hyperparameter tuning, feature engineering and ensembling methodologies are discussed clearly.
3. **Phase 3:**

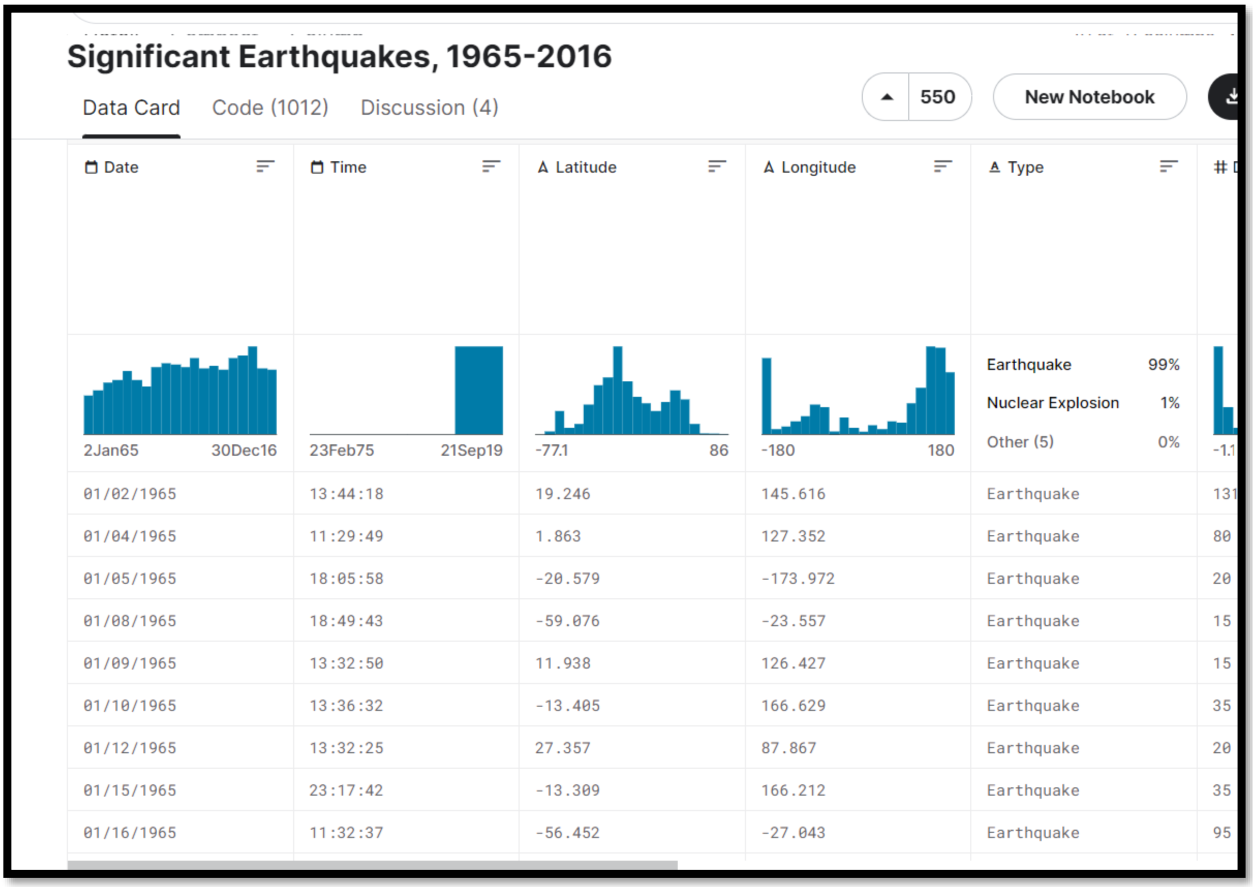
   In phase 3, the Earthquake dataset is loaded and preprocessed.
4. **Phase 4:**

   In phase 4,visualization of earthquake prone areas in a world map and neural network model building and evaluation was completed.

ABOUT THE DATASET USED:

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

- 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.
- The target feature of the neural network model is the column named "magnitude".
- In general, the dataset contained missing values in many of its columns.
- But, the beneficial fact is that those missing values containing columns are not the significant ones.
- The important features required for model building are date, time, magnitude, latitude and longitude.



DATA PRE-PROCCESSING AND FEATURE EXPLORATION:

1. **Checking for the total rows and columns:**

```
[ ]  #Finding the shape of the dataset
     df.shape

     (23412, 21)
```

2. **Checking for duplicated values in the instances of the dataset:**

```
[ ]  #Checking for duplicated values in the rows of the dataset
     df.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
```

3. **Statistical information about the dataset:**

```
[ ]  #Description about the dataset
     df.describe()
```

	Latitude	Longitude	Depth	Depth Error	Depth Seismic Stations
count	23412.000000	23412.000000	23412.000000	4461.000000	7097.000000
mean	1.679033	39.639961	70.767911	4.993115	275.364098
std	30.113183	125.511959	122.651898	4.875184	162.141631
min	-77.080000	-179.997000	-1.100000	0.000000	0.000000
25%	-18.653000	-76.349750	14.522500	1.800000	146.000000
50%	-3.568500	103.982000	33.000000	3.500000	255.000000
75%	26.190750	145.026250	54.000000	6.300000	384.000000
max	86.005000	179.998000	700.000000	91.295000	934.000000

4. Categorizing the columns based on their datatypes:

```
[ ] #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',
    Numerical columns : ['Latitude', 'Longitude', 'Depth', 'Depth Error', '
```

5. Uniqueness check in categorical columns:

```
#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
```

6. Finding number of missing values in the columns:

```
[ ] #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
```

7. Percentage of missing values:

```
#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
```

8. Checking the column named “type” for unique values:

```
#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
```

9. Selecting the rows containing only the Earthquake type:

```
#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
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN

10. Dropping unnecessary columns and handling missing values:

```
[ ] #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)

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

    (23412, 6)

[ ] #Significant columns
    df.columns

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

11. Checking for missing values after feature engineering:

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

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

12. Creating a new column called 'Timestamp' from columns 'Date' and 'Time':

```
[ ] # We convert given Date and Time to Unix time which is in seconds and a
    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
```

❖ Dropping the columns date and time after creating the column Timestamp.

```
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
```

INNOVATIVE TECHNIQUES USED:

- ✓ Through feature engineering, two features such as ‘date’, ‘time’ are converged into a single feature called ‘timestamp’.
- ✓ The above technique reduced the latency for training the neural network model and increased the performance of the model.

```
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
```

- ✓ GridSearchCV is used to find the best parameters for our neural network model.

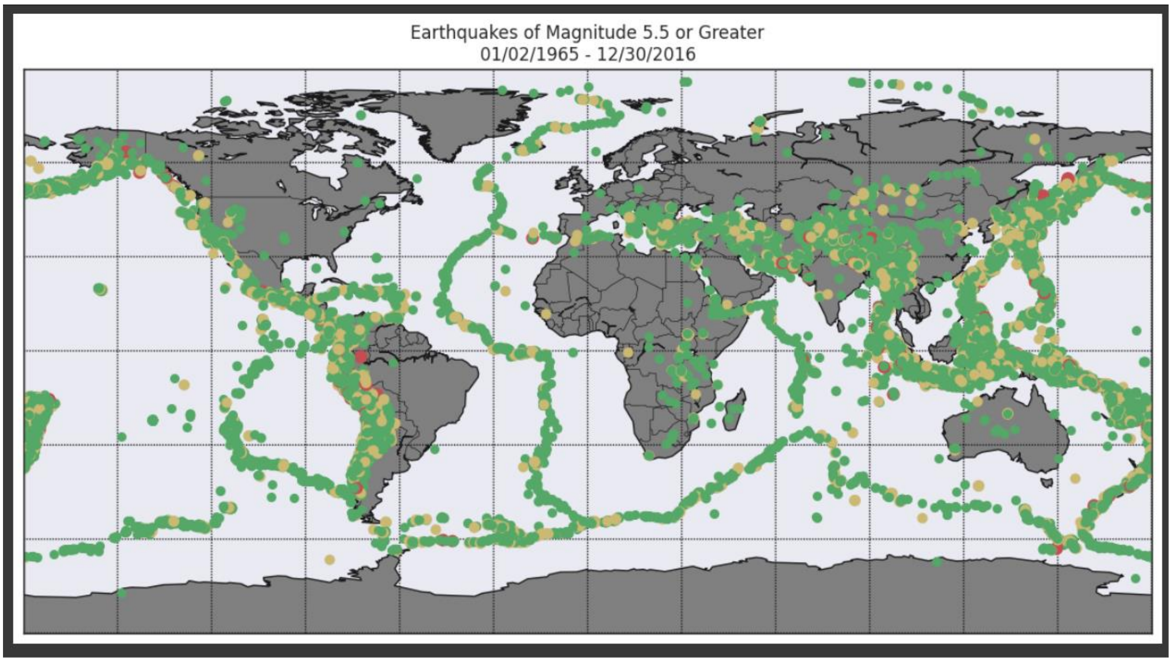
GridSearchCV is used for finding the best parameters for tuning the model's performance

```
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'}
```

- ✓ Visualizing the earthquake prone areas in the world map using the *"basemap"* toolkit of *matplotlib* library.



PERFORMANCE OF THE MODEL:

- ❖ Two models are created during this project.
- ❖ One being the Logistic Regression model and the other is the Neural Network model.
- ❖ The Neural Network model outperformed the Logistic Regression model in terms of accuracy score metrics.

Logistic Regression Model

```
[ ] 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)
```

```
[ ] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
    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
```

```
[ ] 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
```

ASSIGNMENT NOTEBOOK:

- The entire project is developed in the *Google Colab Notebook*.
- All the code files, including the data preprocessing, model training and evaluation steps are compiled in prior.

Colab Notebook link: Click here

CONCLUSION:

Our neural network model nearly scored an accuracy of 98%.The project of building an Earthquake Prediction Model has been successfully accomplished.