EARTHQUAKE PREDICTION USING PYTHON

Problem Definition:

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 a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features.

Design Thinking:

Data Source

- To kickstart the earthquake prediction project, the first crucial step is to identify an appropriate source of data.

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

- This suitable Kaggle dataset contains earthquake data with features like date, time, latitude, longitude, depth, and magnitude.

Feature Exploration:

Understanding key features

- In earthquake prediction, these features typically include attributes like date, time, latitude, longitude, depth, and magnitude. Understanding these features means knowing what they represent and their significance in relation to earthquake prediction.
- "magnitude" is a critical feature as it quantifies the earthquake's size and impact.

Exploratory Data Analysis (EDA)

- In earthquake prediction, we use EDA to identify trends, seasonality, or geographic clustering of earthquake occurrences. It can also help identify outliers or anomalies in the data.

Visualization:

Data Visualization for Insight

- It involves creating visual representations of earthquake-related data to gain insights and communicate findings effectively.

Earthquake Frequency Distribution

- One crucial visualization you can create is a world map that displays the distribution of earthquake occurrences across different regions. This map can help visualize the frequency and geographic patterns of earthquakes. Using color coding or markers, you

can indicate the magnitude of each earthquake event, providing a visual understanding of where stronger earthquakes are more prevalent.

Temporal Patterns

- We create time series plots or histograms to visualize temporal patterns in earthquake occurrences. This can reveal trends over time, such as whether earthquake frequency is increasing or decreasing, and if there are any seasonal or cyclical patterns.

Feature Relationships

- Creating scatter plots to see if there's a correlation between earthquake depth and magnitude. Heatmaps and correlation matrices can provide a visual representation of feature relationships, helping you identify which features might be most influential for prediction.

Outlier Detection

- Visualizations can be used to identify outliers or anomalies in the data. Box plots or violin plots can highlight data points that deviate significantly from the norm, which may require special attention during data preprocessing or model development.

Data Splitting

Training and Testing Sets

- Training Set: This portion of the data is used to train your machine learning model. The model learns patterns, relationships, and trends from this dataset. It is the foundation upon which your model is built.
- Testing Set (Validation Set): The testing set, sometimes referred to as the validation set, is used to assess the performance of your trained model. It serves as a simulation of real-world data the model hasn't seen during training. The model's predictions on this set are compared to the actual values to evaluate its accuracy and generalization capability.

Splitting Techniques

There are various techniques for splitting data:

- Random Splitting: We randomly allocate a percentage of the data (e.g., 70% for training and 30% for testing) to each set. This approach is straightforward but may not consider the data's temporal or spatial distribution.
- Stratified Splitting: In cases where we want to ensure that both training and testing sets have similar distributions of a specific feature (e.g., earthquake magnitude categories), you can use stratified splitting to maintain that balance.
- Time-Based Splitting: If our data has a temporal component, such as earthquake occurrences over time, it's essential to split it chronologically. You can use past data for training and future data for testing to simulate real-world scenarios.

Cross-Validation

In addition to a single data split, we employ cross-validation techniques, such as k-fold cross-validation. This involves dividing the data into multiple subsets (folds), repeatedly training and testing the model on different combinations of these folds, and then aggregating the results to obtain a more robust evaluation of your model's performance.

Phases of development:

Phase 1: Problem Definition and Design Thinking

Phase 2 : explore innovative techniques such as ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.

Phase 3 : Building the earthquake prediction model by loading and preprocessing the dataset.

Phase 4: Visualizing the data on a world and splitting it into training and testing sets

Phase 5: Project Documentation & Submission

Import Required Packages:

```
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	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ID	Source	Location Source	Magnitude Source	Status
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23407	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.30	1.2	40.0	5.6	ML	...	18.0	42.47	0.120	NaN	0.1898	NN00570710	NN	NN	NN	Reviewed
23408	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.80	2.0	33.0	5.5	ML	...	18.0	48.58	0.129	NaN	0.2187	NN00570744	NN	NN	NN	Reviewed
23409	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10.00	1.8	NaN	5.9	MWW	...	NaN	91.00	0.992	4.8	1.5200	US10007NAF	US	US	US	Reviewed
23410	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79.00	1.8	NaN	6.3	MWW	...	NaN	26.00	3.553	6.0	1.4300	US10007NL0	US	US	US	Reviewed
23411	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	2.2	NaN	5.5	MB	...	428.0	97.00	0.681	4.5	0.9100	US10007NTD	US	US	US	Reviewed

23412 rows × 21 columns

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
```

+ Code + Markdown

Preprocessing

data.isna().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
Source                         0
Location Source                0
Magnitude Source               0
Status                         0
dtype: int64
```

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()
```

```
[19…  0
```

```
data['Month'] = data['Date'].apply(lambda x: x[0:2])
data['Year'] = data['Date'].apply(lambda x: x[-4:])

data = data.drop('Date', axis=1)
data['Month'] = data['Month'].astype(np.int)
data[data['Year'].str.contains('Z')]
```

	Time	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source	Magnitude Source	Status	Month	Year
3378	1975-02-23T02:58:41.000Z	8.017	124.075	Earthquake	623.0	5.6	MB	1.022784	US	US	US	Reviewed	19	000Z
7510	1985-04-28T02:53:41.530Z	-32.998	-71.766	Earthquake	33.0	5.6	MW	1.300000	US	US	HRV	Reviewed	19	530Z
20647	2011-03-13T02:23:34.520Z	36.344	142.344	Earthquake	10.1	5.8	MWC	1.060000	US	US	GCMT	Reviewed	20	520Z

```
invalid_year_indices = data[data['Year'].str.contains('Z')].index

data = data.drop(invalid_year_indices, axis=0).reset_index(drop=True)
data['Year'] = data['Year'].astype(np.int)
data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))

data = data.drop('Time', axis=1)
data
```

	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source	Magnitude Source	Status	Month	Year	Hour
0	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	13
1	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	11
2	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	18
3	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	18
4	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	Automatic	1	1965	13
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
23401	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN	NN	Reviewed	12	2016	8
23402	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN	NN	Reviewed	12	2016	9
23403	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US	US	Reviewed	12	2016	12
23404	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US	US	Reviewed	12	2016	22
23405	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US	US	Reviewed	12	2016	20

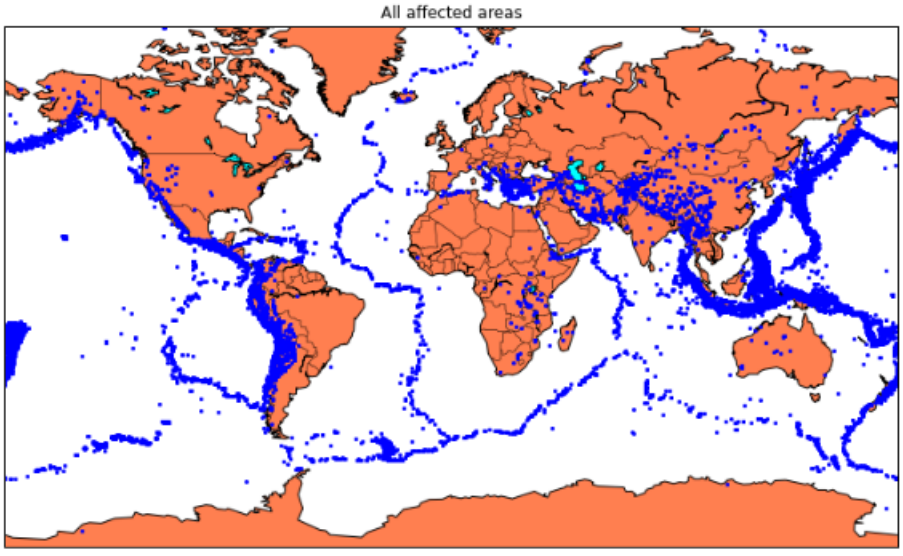
23406 rows × 14 columns

```
data['Status'].unique()
```

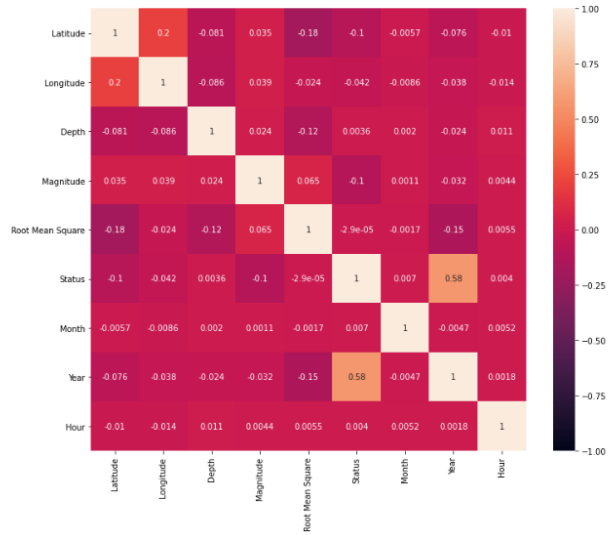
```
data['Status'] = data['Status'].apply(lambda x: 1 if x == 'Reviewed' else 0)
```

Visualization

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-
180,urcrnrlon=180,lat_ts=20,resolution='c')

longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
            #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)

fig = plt.figure(figsize=(12,10))
plt.title("All affected areas")
m.plot(x, y, "o", markersize = 2, color = 'blue')
m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()
```




```
numeric_columns = [column for column in data.columns if data.dtypes[column] != 'object']
corr = data[numeric_columns].corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, vmin=-1.0, vmax=1.0)
plt.show()
```



Encoding

data

[36...

	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	Root Mean Square	Source	Location Source	Magnitude Source	Status	Month	Year	Hour
0	19.2460	145.6160	Earthquake	131.60	6.0	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	0	1	1965	13
1	1.8630	127.3520	Earthquake	80.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	0	1	1965	11
2	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	0	1	1965	18
3	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	0	1	1965	18
4	11.9380	126.4270	Earthquake	15.00	5.8	MW	1.022784	ISCGEM	ISCGEM	ISCGEM	0	1	1965	13
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23401	38.3917	-118.8941	Earthquake	12.30	5.6	ML	0.189800	NN	NN	NN	1	12	2016	8
23402	38.3777	-118.8957	Earthquake	8.80	5.5	ML	0.218700	NN	NN	NN	1	12	2016	9
23403	36.9179	140.4262	Earthquake	10.00	5.9	MWW	1.520000	US	US	US	1	12	2016	12
23404	-9.0283	118.6639	Earthquake	79.00	6.3	MWW	1.430000	US	US	US	1	12	2016	22
23405	37.3973	141.4103	Earthquake	11.94	5.5	MB	0.910000	US	US	US	1	12	2016	20

23406 rows × 14 columns

```
data['Type'].unique()
def onehot_encode(df, columns, prefixes):
    df = df.copy()
    for column, prefix in zip(columns, prefixes):
        dummies = pd.get_dummies(df[column], prefix=prefix)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(column, axis=1)
    return df
data = onehot_encode(
    data,
    ['Type', 'Magnitude Type', 'Source', 'Location Source', 'Magnitude Source'],
    ['t', 'mt', 's', 'ls', 'ms']
)
```

data

	Latitude	Longitude	Depth	Magnitude	Root Mean Square	Status	Month	Year	Hour	t_Earthquake	...	ms_NN	ms_OFFICIAL	ms_PAR	ms_PGC	ms_PR	ms_SE	ms_US	ms_US_GCMT	ms_US_PGC	ms_UW
0	19.2460	145.6160	131.60	6.0	1.022784	0	1	1965	13	1	...	0	0	0	0	0	0	0	0	0	0
1	1.8630	127.3520	80.00	5.8	1.022784	0	1	1965	11	1	...	0	0	0	0	0	0	0	0	0	0
2	-20.5790	-173.9720	20.00	6.2	1.022784	0	1	1965	18	1	...	0	0	0	0	0	0	0	0	0	0
3	-59.0760	-23.5570	15.00	5.8	1.022784	0	1	1965	18	1	...	0	0	0	0	0	0	0	0	0	0
4	11.9380	126.4270	15.00	5.8	1.022784	0	1	1965	13	1	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23401	38.3917	-118.8941	12.30	5.6	0.189800	1	12	2016	8	1	...	1	0	0	0	0	0	0	0	0	0
23402	38.3777	-118.8957	8.80	5.5	0.218700	1	12	2016	9	1	...	1	0	0	0	0	0	0	0	0	0
23403	36.9179	140.4262	10.00	5.9	1.520000	1	12	2016	12	1	...	0	0	0	0	0	0	1	0	0	0
23404	-9.0283	118.6639	79.00	6.3	1.430000	1	12	2016	22	1	...	0	0	0	0	0	0	1	0	0	0
23405	37.3973	141.4103	11.94	5.5	0.910000	1	12	2016	20	1	...	0	0	0	0	0	0	1	0	0	0

23406 rows × 105 columns

Splitting and Scaling

```
y = data.loc[:, 'Status']
X = data.drop('Status', axis=1)
scaler = StandardScaler()

X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=56)
```

Modeling and Training

```
y.mean()
```

```
[49…  0.88737930445185
```

```
inputs = tf.keras.Input(shape=(104,))
x = tf.keras.layers.Dense(64, activation='relu')(inputs)
x = tf.keras.layers.Dense(64, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, outputs)


model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[tf.keras.metrics.AUC(name='auc')]
)


batch_size = 32
epochs = 30

history = model.fit(
    X_train,
```

```
    y_train,
    validation_split=0.2,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=[tf.keras.callbacks.ReduceLROnPlateau()],
    verbose=0
)
```

Results

model.evaluate(X_test, y_test)

```
220/220 [==============================] - 0s 884us/step - loss: 0.0015 - auc: 0.9999
[47]: [0.001497176825068891, 0.9999196529388428]
```

len(y_test)

```
[48]: 7022
```