IBM NAAN MUDHALVAN

Technology name: Artificial Intelligence

Project title: Earthquake prediction model using python

Phase 3: Development Part-1

- Loading and Preprocessing

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

Notebook link: **Google Colab**

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. We provide a model which will show the loading and pre-processing 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 Loading?

Python can be used to read data from a variety of places, including databases and files. Two file types that are often used are .txt and .csv. We can import and export files using built-in Python functionality or Python's CSV library. Data loading is the first step in machine learning, which is essential for obtaining data sets for model development. Identifying the data source, whether it is CSV files, databases, or APIs, determines the loading approach. Integrating libraries such as Pandas streamlines the process, allowing users to efficiently process and analyse the data. The accompanying code snippets in the documentation illustrate the programmatic loading of datasets and provide accessibility and ease of understanding. Versatility is emphasized by addressing various data formats such as CSV, Excel, JSON, or databases, allowing for adaptation to different structures. Robust data loading also includes error handling that anticipates and handles issues such as missing values or corrupted data.

There are five ways in loading data into a python program. They are:

1. Manually loading a file
2. Numpy Methods
3. Using np.GenFromTxt
4. Using PD.read_csv
5. Using Pickle

What is preprocessing?

Pre-processing refers to the transformations applied to our data before it is fed into the algorithm. Data preprocessing is a technique used to transform raw data into a clean data set. In other words: When the data comes from different sources, it is collected in raw format, which is not suitable for analysis.

Necessity of data preprocessing:

To achieve better results with the applied model in machine learning projects, the data must be in a suitable format. Some machine learning models require information in a specific format, for example, the Random Forest algorithm does not support null values. Therefore, in order to run the Random Forest algorithm, null values must be managed from the original raw data set.

Another aspect is that the dataset should be formatted to run more than one machine learning and deep learning algorithm in a dataset and select the best one among them.

Dataset location:

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

File location:

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

Program and Output:

#Importing libraries

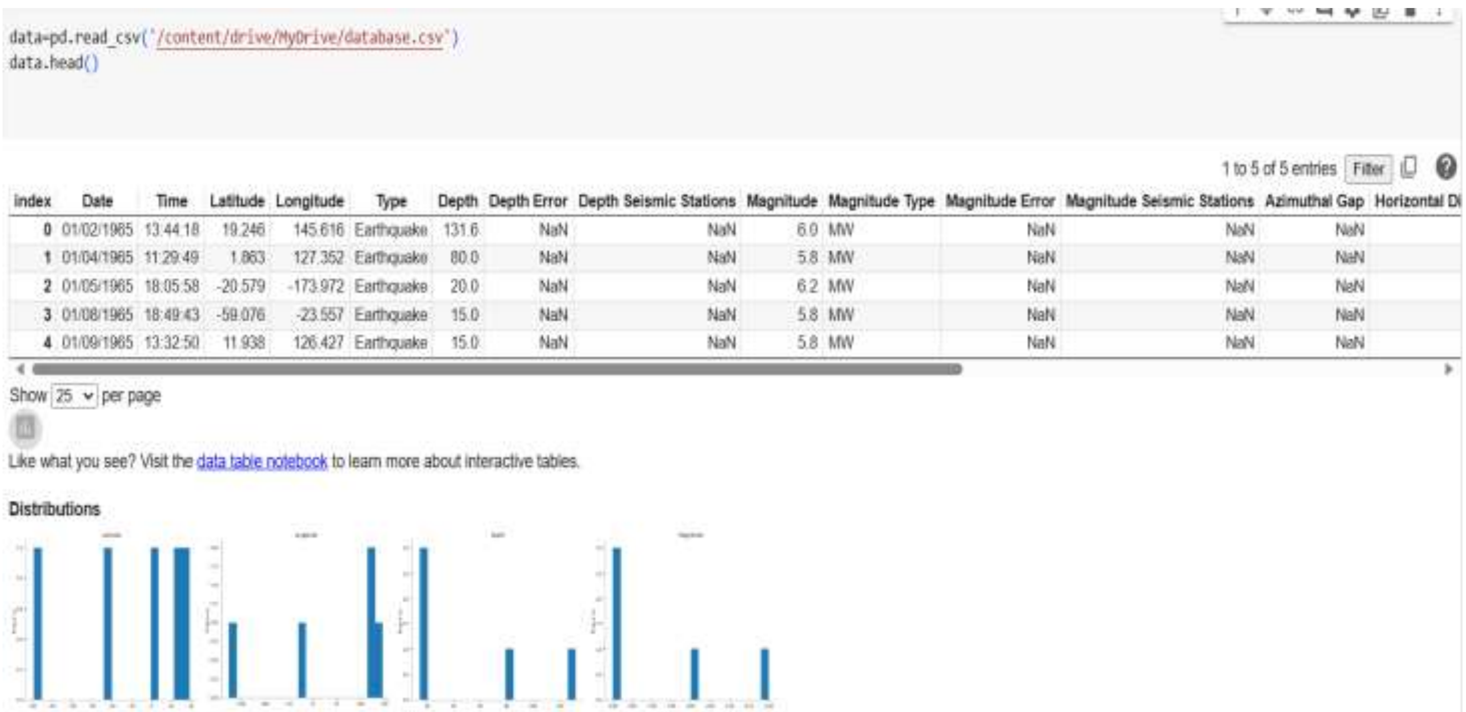
```
import pandas as pd
import numpy as np

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

 # Reading the Dataset

data=pd.read_csv('/content/drive/MyDrive/database.csv')

data.head()



```
data=pd.read_csv('/content/drive/MyDrive/database.csv')
data.head()
```

1 to 5 of 5 entries Filter

index	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Di
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	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	
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	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	
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN	NaN	

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Distributions

 # knowing the information about the dataset

data.info()

```python
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
```

#display data

data

```python
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
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

23412 rows × 21 columns

 # keeping the necessary columns and deleting unnecessary columns from the dataset

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

null_columns = data.loc[:, data.isna().sum() > 0.66 * data.shape[0]].columns

data = data.drop(null_columns, axis=1)
```

 # count of missing values

```
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
```

 # Filling missing values

data['Root Mean Square'] = data['Root Mean Square'].fillna(data['Root Mean Square'].mean())

 # deleting the columns with missing values and checking any missing values exist in the dataset

data = data.dropna(axis=0).reset_index(drop=True)

data.isna().sum().sum()

```
0
```

 # Extracting 'Month', 'Year', and 'Hour' from 'Date' and 'Time'

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

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

```
<ipython-input-39-2ccc4f5a16e1>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not mo
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  data['Month'] = data['Month'].astype(np.int)
```

 # Converting 'Month' to integer type

```
data['Month'] = data['Month'].astype(np.int)
```

```
<ipython-input-42-ec5898e41d7b>:1: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not mo
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))
```

 # converting month into integer types

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

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)
```

 # Extracting 'Hour' from 'Time'

```
data['Hour'] = data['Time'].apply(lambda x: np.int(x[0:2]))
```

```
# Displaying the shape and columns of the final dataset

data.shape

data.columns

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

```
# Converting 'Date' and 'Time' to a timestamp in seconds

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(‘Value Error’)



# Creating a new 'Timestamp' column

timeStamp = pd.Series(timestamp)

data['Timestamp'] = timeStamp.values



# creating and displaying the final dataset

final_data = data.drop(['Date', 'Time'], axis=1)

final_data = final_data[final_data.Timestamp != 'ValueError']
```

```
final_data.head()
```

	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