EARTHQUAKE PREDICTION MODEL USING PYTHON

TEAM MEMBER
311121205304 - MUTHEESWARAN G

Project : Earthquake prediction model using python

INTRODUCTION:

Data preprocessing is a crucial phase in data analysis and machine learning, aimed at refining raw data for effective analysis and model training. This process includes handling missing values, ensuring consistent scaling and normalization, parsing dates for temporal data, managing character encodings for accurate text interpretation, and addressing inconsistencies in data entry. By addressing these issues, data preprocessing ensures that your dataset is clean, reliable, and ready for meaningful insights and model performance improvements.

DATA PREPROCESSING:

1. Handling Missing Values:

Missing values are a common issue in datasets and can have a significant impact on your analysis or models. There are several techniques to handle missing values:

- **Deletion:** Remove rows or columns with missing values using methods like dropna in Pandas.
- **Imputation:** Fill missing values with appropriate data, such as using the mean, median, mode, or more advanced techniques like machine learning-based imputation.
- **Flagging:** Create an additional column to flag missing values.
- **Domain-Specific Handling:** Handle missing values based on domain knowledge.

2. Scaling and Normalization:

Scaling and normalization are essential when your dataset contains features with different scales or units. This ensures that all features are on a comparable scale, which can improve the performance of machine learning models.
Common techniques include:

- **Standardization:** Scaling features to have a mean of 0 and a standard deviation of 1.
- **Min-Max Scaling:** Scaling features to a specific range, often between 0 and 1.
- **Robust Scaling:** Scaling using median and interquartile range to handle outliers.

3. Parsing Data:

Parsing data involves extracting relevant information from structured or unstructured data and converting it into a structured format. This is especially important when working with datasets that contain dates, times, or complex text fields. Here are some key points to consider:

- **Parsing Dates:** When dealing with date and time data, you'll need to parse and convert it into a standardized datetime format. Python libraries like Pandas provide functions for date parsing.Consider time zone information if it's relevant to your analysis.
  Pay attention to date and time formats, as they can vary significantly
  (e.g., "YYYY-MM-DD" vs. "DD/MM/YYYY").

- **Parsing Text Data:** When working with text data, parsing may involve tokenization (splitting text into words or sentences), stemming or lemmatization (reducing words to their root form), and removing stop words (common words like "the" or "and").
  Named entity recognition (NER) can help identify entities such as names, locations, and organizations within text data.

4. Character Encodings:

Character encodings are crucial for correctly interpreting and handling text data, especially when dealing with multilingual or international datasets. Here's what you need to know:

- **Common Encodings:**UTF-8 is the most widely used character encoding and can handle characters from most languages.
  Other encodings include UTF-16, ISO-8859-1, and more.
  Ensure that your data is consistently encoded with the appropriate character encoding.

- **Decoding and Encoding:**When reading data from files or APIs, make sure to specify the correct character encoding during reading (e.g., encoding='utf-8' in Pandas read_csv).
  You may need to decode data from one encoding and then encode it in another if your analysis requires a different encoding.
- **Handling Encoding Errors:**Be aware of encoding errors. Sometimes data can be corrupted, and handling encoding errors gracefully is essential.

5. Inconsistent Data Entry:

Inconsistent data entry refers to variations or errors in data values, formats, or types. It can occur for various reasons, including manual data entry, data migration, or merging data from different sources.

Here's how to address it:

- **Data Profiling:**Begin by profiling your data to identify inconsistencies. Use tools like Pandas to check data types, unique values, and missing values.
  Look for variations in spelling, capitalization, and formatting.

- **Regular Expressions (Regex):**Regular expressions can be powerful tools for identifying and correcting inconsistencies. You can use them to search for specific patterns or validate data based on expected formats.

- **Data Cleaning Functions:**Write custom data cleaning functions to address specific inconsistencies. These functions can be used to fix typos, convert values to a consistent format, or remove outliers.

- **Visualization:**Data visualization can help identify inconsistencies, especially when dealing with numerical data. Plotting histograms and box plots can reveal outliers and unusual data patterns.

- **Domain Knowledge:**Consult with domain experts to understand the context of the data and the possible sources of inconsistencies. This knowledge can be invaluable in addressing data quality issues.

- **Data Validation Rules:**Define data validation rules or constraints for your dataset and use them to validate the data. For example, you can create rules to check that values fall within an expected range or format.

Properly handling parsing, character encodings, and inconsistent data is essential for ensuring that your dataset is clean, accurate, and reliable. It's a critical step in preparing your data for meaningful analysis or machine learning applications.

CODE:

```
#Data Preprocessing includes
'''
1)Handling missing values
2)Scaling and normalization
3)Parsing dates
4)Character encodings
5)Inconsistent Data Entry'''

# modules we'll use
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import datetime

# read data
earthquakes = pd.read_csv("database.csv")

# set seed for reproducibility
np.random.seed(0)

# print the first 5 rows of the data
earthquakes.head()
```

Out[2]:

	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	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01-04-1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01-05-1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01-08-1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01-09-1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN	NaN	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic

5 rows × 21 columns

```
#The data contains Missing values
'''HANDLING MISSING VALUES'''

#The profile of the missing values
missing_values = earthquakes.isnull().sum()
print("MISSING VALUES")
print(missing_values)
```

```
MISSING VALUES
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
```

#Dropping rows and columns that contain NaN values
earthquakes.dropna(axis=0, inplace=True)
earthquakes.dropna(axis=1, inplace=True)
earthquakes.sample(10)

Out[4]:

	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
3673	10/28/1975	14:30:00	37.290167	-116.411500	Nuclear Explosion	1.30	31.61	19.0	5.67	ML	...	8.0	260.0	1.472000	99.000	2.75	CI3006257	CI	CI	CI	Reviewed
3307	11/22/1974	16:25:34	30.250000	-114.800000	Earthquake	6.00	31.61	20.0	5.55	ML	...	4.0	312.0	2.641000	99.000	3.22	CI3319062	CI	CI	CI	Reviewed
1532	03/26/1970	19:00:01	37.300500	-116.534167	Nuclear Explosion	1.20	31.61	16.0	5.54	ML	...	10.0	260.0	1.377000	99.000	0.35	CI3325031	CI	CI	CI	Reviewed
5631	09-07-1980	04:36:38	38.138333	-118.391333	Earthquake	6.00	31.61	5.0	5.52	ML	...	12.0	333.0	1.090000	8.020	0.33	CI9735242	CI	CI	CI	Reviewed
1129	12/19/1968	16:30:01	37.231500	-116.473667	Nuclear Explosion	1.40	31.61	16.0	5.52	ML	...	6.0	257.0	1.415000	99.000	1.36	CI3342181	CI	CI	CI	Reviewed
22238	05/28/2014	21:15:07	18.045000	-68.350900	Earthquake	90.00	2.00	31.0	5.80	ML	...	12.0	262.8	0.467124	2.900	0.41	PR14148004	PR	PR	PR	Reviewed
3754	01-03-1976	19:15:01	37.296500	-116.333167	Nuclear Explosion	1.50	31.61	20.0	5.84	ML	...	8.0	261.0	1.534000	99.000	0.44	CI3001652	CI	CI	CI	Reviewed
897	04/26/1968	15:00:02	37.295333	-116.455667	Nuclear Explosion	1.20	31.61	17.0	5.63	ML	...	6.0	261.0	1.438000	99.000	1.03	CI3342128	CI	CI	CI	Reviewed
3516	06/26/1975	12:30:01	37.278833	-116.368667	Nuclear Explosion	6.00	31.61	19.0	5.52	ML	...	13.0	260.0	1.504000	99.000	3.44	CI12328563	CI	CI	CI	Reviewed
5523	05/18/1980	15:32:11	46.207333	-122.188000	Earthquake	1.51	0.56	18.0	5.70	MD	...	1.0	62.0	0.008296	0.682	0.22	UW10084803	UW	UW	UW	Reviewed

#The data column is not parsed
"'PARSING DATES"'

 # print the first few rows of the date column
print(earthquakes['Date'].head())

```
565     12/20/1966
897     04/26/1968
1129    12/19/1968
1380    09/16/1969
1532    03/26/1970
Name: Date, dtype: object
```

#Notice the difference in how the dtype is shown in below two cases
```
print(earthquakes['Date'].dtype)
print()
earthquakes['Date'].dtype
```
```
object

dtype('O')
```

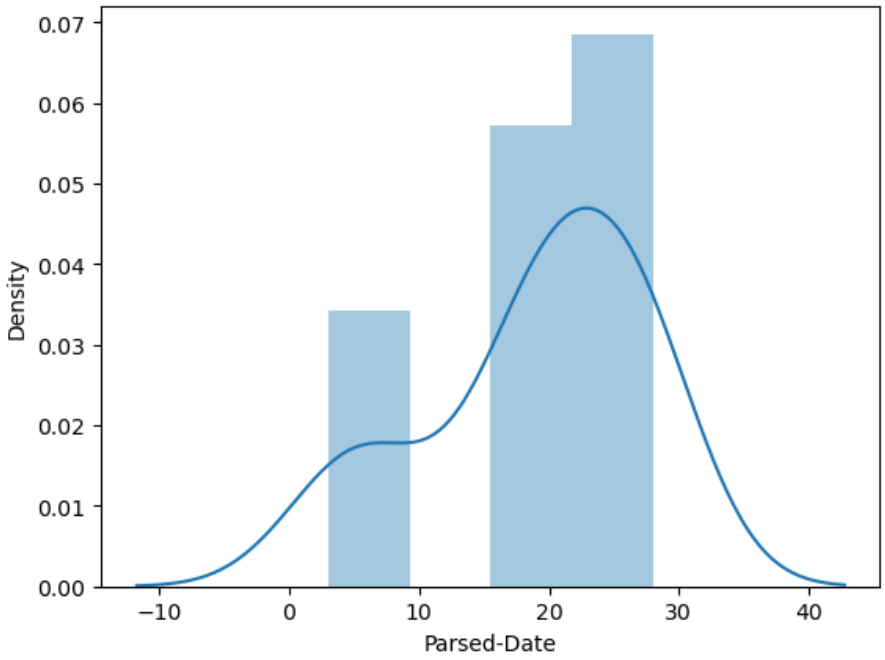
 # using infer_datetime_format as the explicit format failed due to data not in consistent form
```
earthquakes['Parsed-Date'] = pd.to_datetime(earthquakes['Date'], infer_datetime_format=True)
```

 # Check the date in new column : Parsed-Date
```
earthquakes['Parsed-Date'].head()
```
```
565    1966-12-20
897    1968-04-26
1129   1968-12-19
1380   1969-09-16
1532   1970-03-26
Name: Parsed-Date, dtype: datetime64[ns]
```

 # print sample data for Parsed-Date
```
earthquakes['Parsed-Date'].sample(10)
```

```
2723    1973-06-06
1129    1968-12-19
1380    1969-09-16
1532    1970-03-26
5631    1980-09-07
565     1966-12-20
3754    1976-01-03
22238   2014-05-28
3516    1975-06-26
3307    1974-11-22
Name: Parsed-Date, dtype: datetime64[ns]
```

```
pip install pytz
import pandas as pd
import pytz

# Assuming datetimes are originally in UTC
utc = pytz.UTC

# Convert the 'Parsed-Date' column to datetime format
earthquakes['Parsed-Date'] = pd.to_datetime(earthquakes['Parsed-Date'])

# Make sure the datetime column is timezone-aware (UTC)
earthquakes['Parsed-Date'] = earthquakes['Parsed-Date'].apply(lambda x: x if x.tzinfo else
utc.localize(x))

# Now, use the .dt accessor to extract date components
day_of_month_earthquakes = earthquakes['Parsed-Date'].dt.day
month_of_earthquakes = earthquakes['Parsed-Date'].dt.month
year_of_earthquakes = earthquakes['Parsed-Date'].dt.year

#data visualization
day_of_month_earthquakes = earthquakes['Parsed-Date'].dt.day

day_of_month_earthquakes = day_of_month_earthquakes.dropna()

sns.distplot(day_of_month_earthquakes)
```



Hence the loading and Preprocessing of the dataset has been completed.