EARTHQUAKE PREDICTION MODEL USING PYTHON

BATCH MEMBER

TEAM-ID : Proj_212172_Team_2

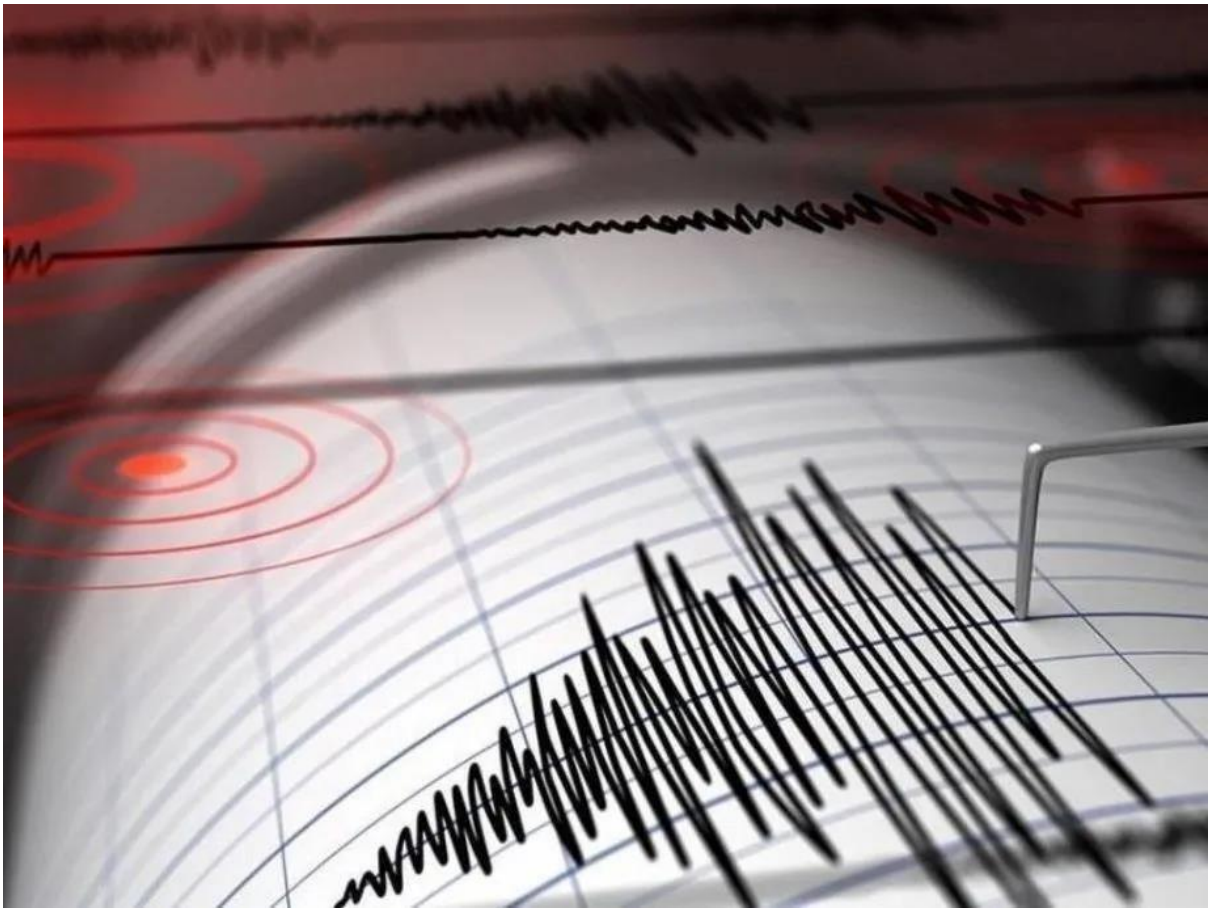
950621104097 : S.SOBIKA

Phase 3 Submission Document

Project Title: Earthquake Prediction Model Using Python

Phase 3: Development Part 1

Topic: Start building the earthquake prediction model by loading and pre-processing the dataset.



Earthquake Prediction Model Using Python

Introduction:

- Whether you're using earthquake prediction app then the value of the data will be accurate and give you the notification about it.
- The data scientist aiming to build a predictive model, the foundation of this endeavour lies in loading and preprocessing the dataset.
- This introduction will guide you through the initial steps of the process.
- We'll explore how to import essential libraries, load the earthquake dataset, and perform critical preprocessing steps.
- Data preprocessing is crucial as it helps clean, format, and prepare the data for further analysis.
- This include handling missing values, encoding categorical variables, and ensuring that the data is appropriately scaled.

Given Dataset:

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

23412 rows × 21 columns

Necessary Steps to follow:

1.Import Libraries:

Start by importing the necessary libraries:

Program:

```
Import pandas as pd

Import numpy as np

From sklearn.model_selection import train_test_split

From sklearn.preprocessing import StandardScaler
```

2.Load the Dataset:

Load the dataset into the Pandas dataframe. We can typically find earthquake prediction dataset in CSV format, but we can adapt this code to other formats as needed.

Program:

```
df = pd.read_csv('C:/Users/barat/Downloads/archive/database.csv') df
```

3.Exploratory Data Analysis(EDA):

Perform EDA to understand the data better. This includes checking for missing values, exploring the data's statistics, and visualizing it to identity patterns.

Programs:

```
# Check for missing values

Print(df.isnull().sum())

#Explore statistics Print(df.describe())

#Visulaize the data
```

(eg., histogram, scatter plots, etc.)

4.Feature Engineering:

Depending on the dataset, we may need to create new features or transform existing ones. This can involve one- hot encoding categorical variables, handling date/time data, or scaling numerical features.

Program:

```
# One-hot encoding for categorical variables df =

pd.get_dummies(df, columns=['Latitude','Longitude'])
```

5.Split the Data:

Split the dataset into training and testing sets. This helps us to evaluate the model's performance later.

Program:

```
X = df.drop('Depth',axis=1) y=df['Depth']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

6.Feature Scaling:

Apply feature scaling to normalize the data, ensuring that all features have similar scales.

Program:

```
scaler = StandardScaler()

X_train= scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Importance of loading and processing dataset:

Loading and preprocessing the dataset is an important first step in building any model. However, it is especially important for earthquake prediction model, as earthquake prediction datasets are often complex and noisy.

By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

Challenges involved in loading and preprocessing an earthquake prediction dataset:

There are a number of challenges involved in loading and preprocessing a earthquake prediction dataset, including:

- **Handling missing values:**

  Earthquake prediction dataset often contain missing values, which can be due to a variety of factors, such as human error or incomplete data collection.

- **Scaling the features:**

  It is often helpful to scale the features before training a machine learning model. This can help to improve the performance of the model and make it more robust to outliers.

- **Splitting the dataset into training and testing sets:**

  Once the data has been pre-processed, we need to split the dataset into training and testing sets. The training set will be used to train the model, and the testing set will be used to evaluate the performance of the model on unseen data.

Program:

1.Importing the needed libraries:

import pandas as pd import numpy as np import

seaborn as sns import matplotlib.pyplot as plt from

sklearn.model_selection import train_test_split from

sklearn.compose import ColumnTransformer from

sklearn.preprocessing import StandardScaler from

sklearn.preprocessing import OneHotEncoder from

sklearn.pipeline import Pipeline

Loading Dataset:

df = pd.read_csv('C:/Users/barat/Downloads/archive/database.csv')

Data Exploration:

Dataset:

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error
0	01/02/1965	13:44:18	19.2460	145.6160	Earthquake	131.60	NaN	NaN	6.0	MW	...	NaN	NaN	NaN	NaN
1	01/04/1965	11:29:49	1.8630	127.3520	Earthquake	80.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
2	01/05/1965	18:05:58	-20.5790	-173.9720	Earthquake	20.00	NaN	NaN	6.2	MW	...	NaN	NaN	NaN	NaN
3	01/08/1965	18:49:43	-59.0760	-23.5570	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
4	01/09/1965	13:32:50	11.9380	126.4270	Earthquake	15.00	NaN	NaN	5.8	MW	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

23412 rows × 21 columns

2.Preprocessing the dataset:

- Removing the errors.

- Cleaning, transforming and integrating data.
- Handling missing values.
- Transforming the data into a consistent format.

Visualisation and pre-processing of data:

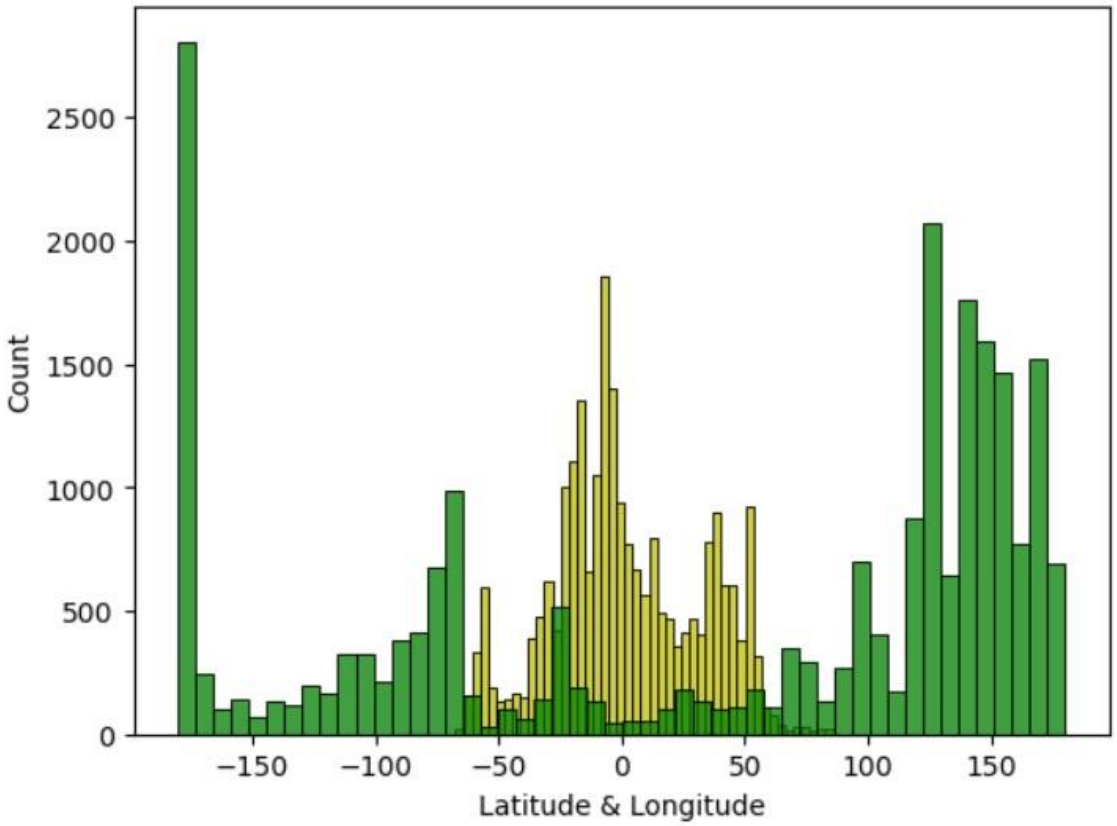
In[1]: sns.histplot(df,x = 'Latitude', bins=50,

color='y') sns.histplot(df,x = 'Longitude', bins=50,

color='g') plt.xlabel('Latitude & Longitude')

Out[1]:

```
Text(0.5, 0, 'Latitude & Longitude')
```

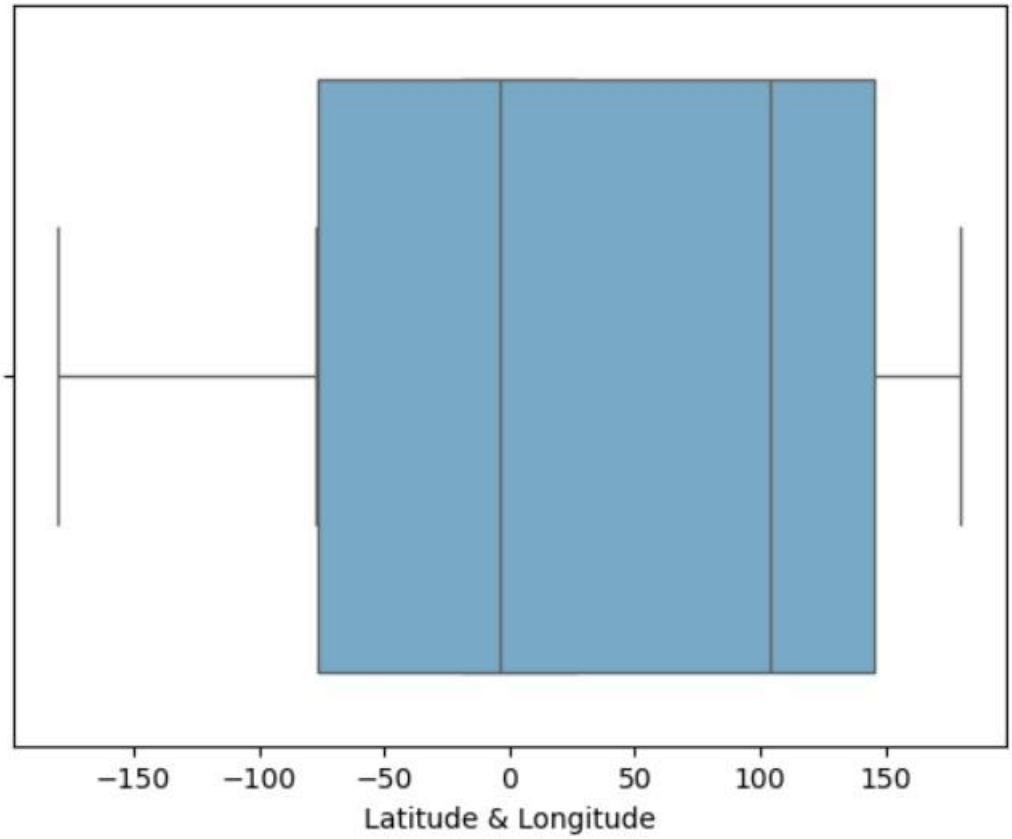


In[2]:

```
sns.boxplot(df,x='Latitude',palette='Blues')
sns.boxplot(df,x='Longitude',palette='Blues') plt.xlabel('Latitude
& Longitude')
```

Out[2]:

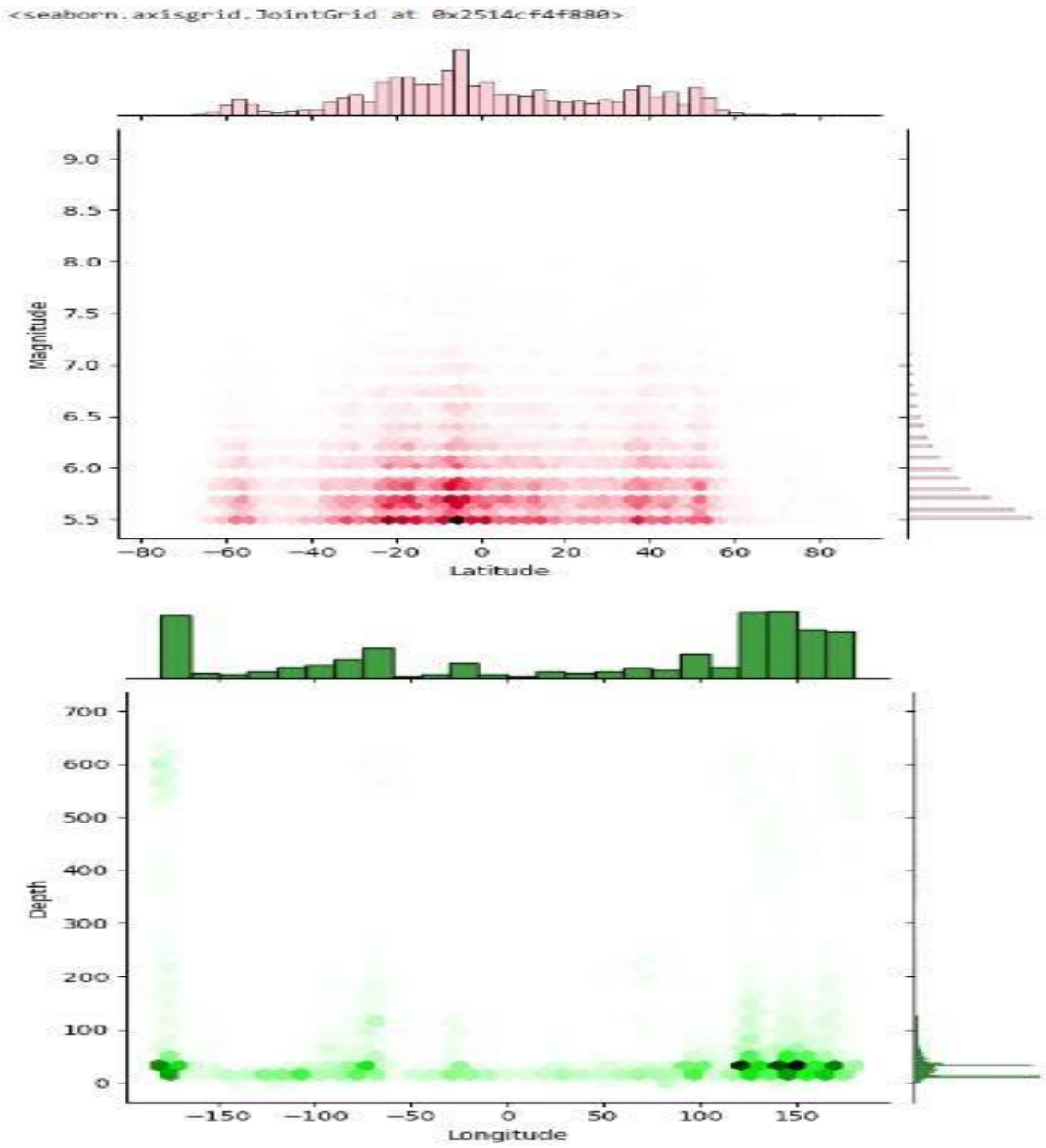
```
Text(0.5, 0, 'Latitude & Longitude')
```



In[3]:

```
sns.jointplot(df,x='Latitude', y='Magnitude', kind='hex', color='pink')
sns.jointplot(df,x='Longitude', y='Depth', kind='hex', color='green')
```

Out[3]:

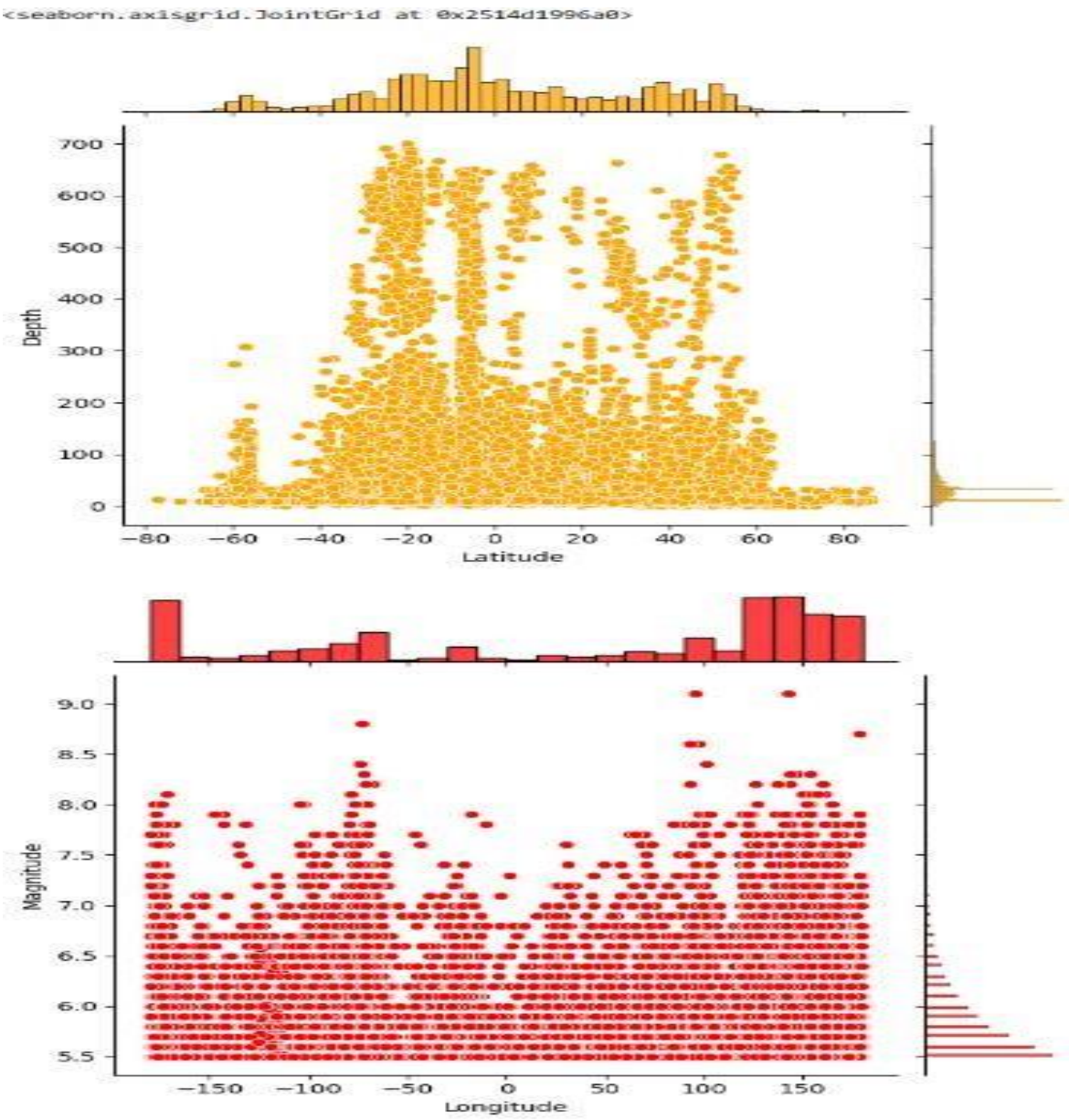


In[4]:

sns.jointplot(df,x='Latitude', y='Depth', color='orange')

sns.jointplot(df,x='Longitude', y='Magnitude', color='red')

Out[4]:

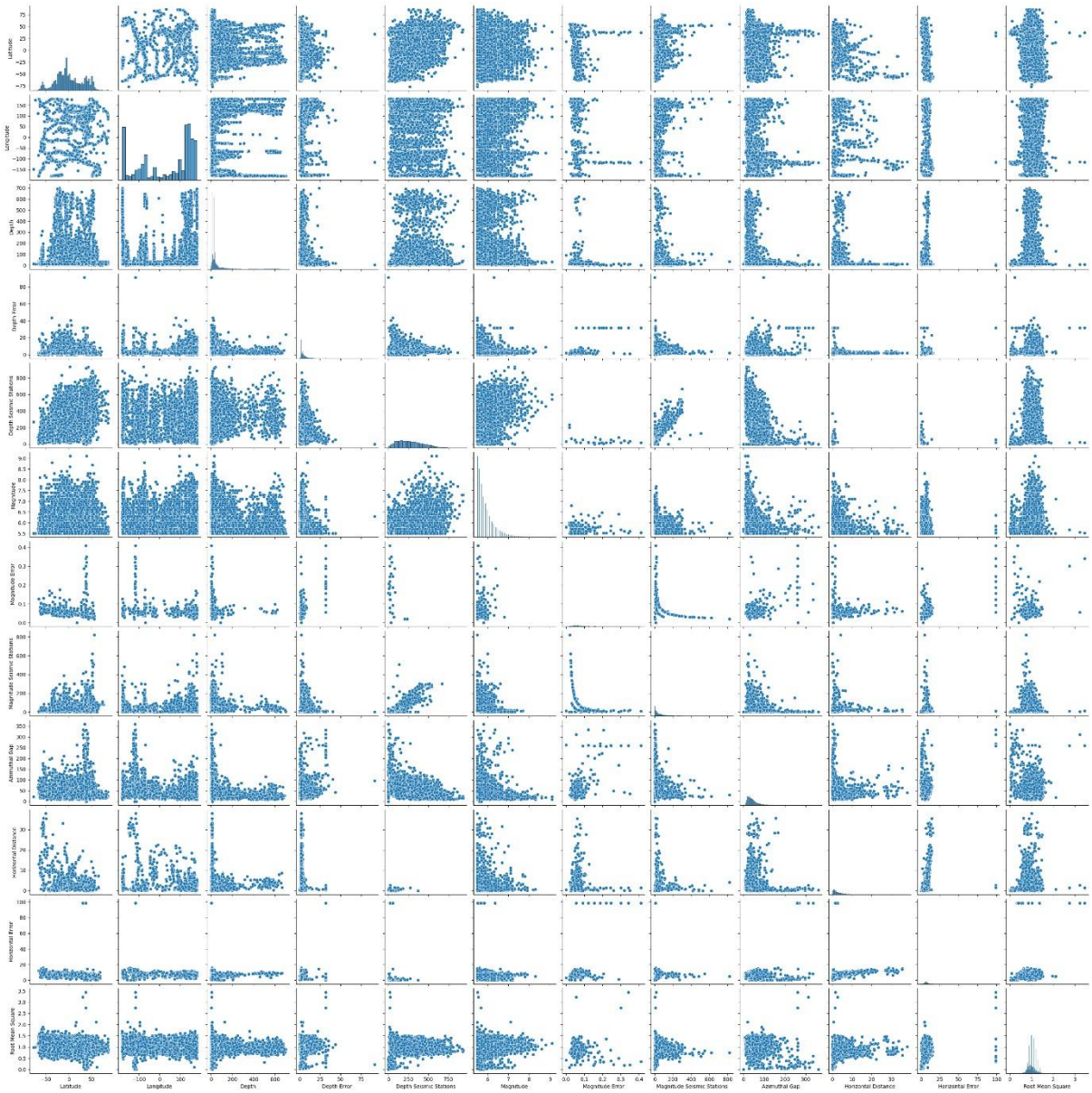


In[5]:

plt.figure(figsize=(6,8))

sns.pairplot(df) plt.show()

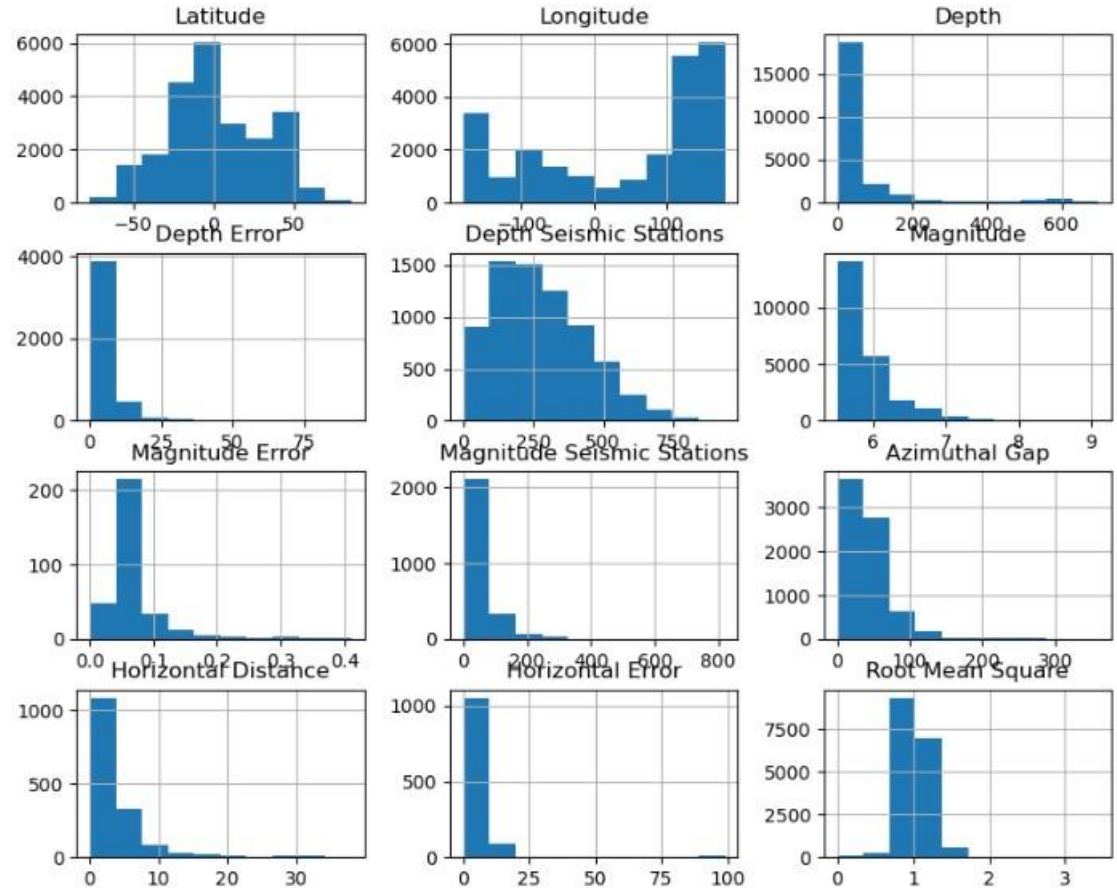
Out[5]:



In[6]: df.hist(figsize=(10,8))

Out[6]:

```
array([[<AxesSubplot: title={'center': 'Latitude'}>,
        <AxesSubplot: title={'center': 'Longitude'}>,
        <AxesSubplot: title={'center': 'Depth'}>],
       [<AxesSubplot: title={'center': 'Depth Error'}>,
        <AxesSubplot: title={'center': 'Depth Seismic Stations'}>,
        <AxesSubplot: title={'center': 'Magnitude'}>],
       [<AxesSubplot: title={'center': 'Magnitude Error'}>,
        <AxesSubplot: title={'center': 'Magnitude Seismic Stations'}>,
        <AxesSubplot: title={'center': 'Azimuthal Gap'}>],
       [<AxesSubplot: title={'center': 'Horizontal Distance'}>,
        <AxesSubplot: title={'center': 'Horizontal Error'}>,
        <AxesSubplot: title={'center': 'Root Mean Square'}>]],
      dtype=object)
```



In[7]:

df.corr(numeric_only= True)

Out[7]:

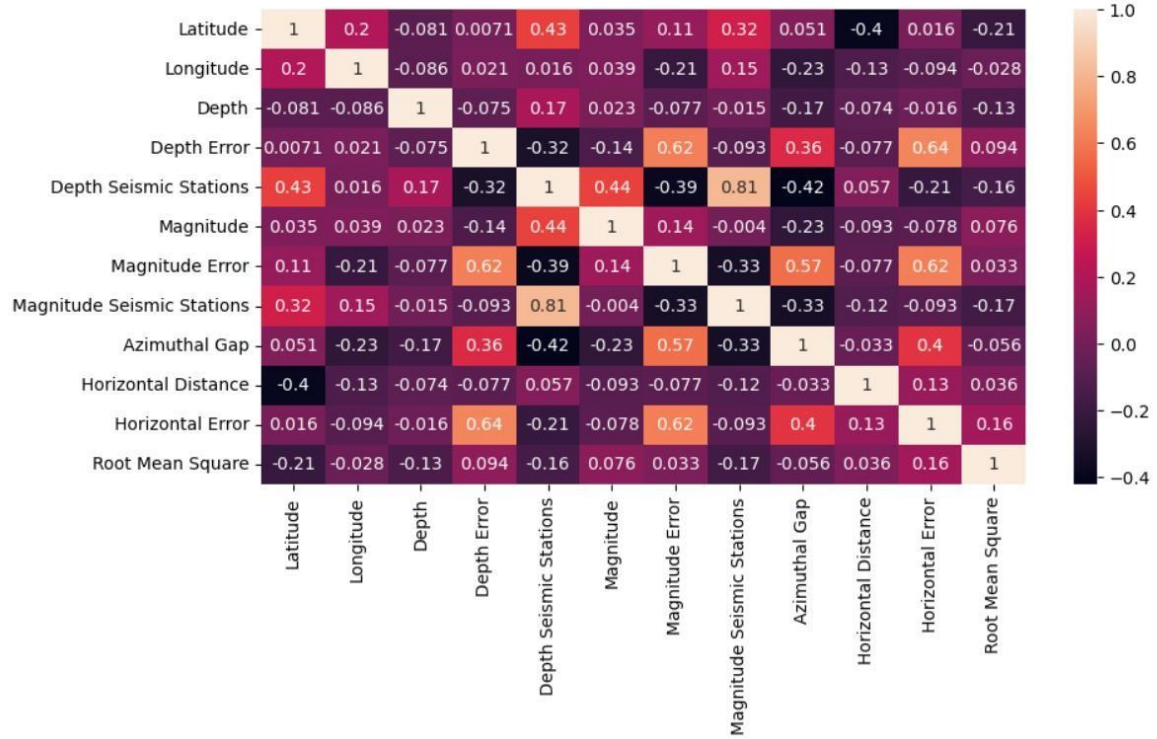
	Latitude	Longitude	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square
Latitude	1.000000	0.203546	-0.081020	0.007080	0.433815	0.034987	0.113208	0.315075	0.050794	-0.396768	0.015625	-0.214762
Longitude	0.203546	1.000000	-0.085861	0.020552	0.015924	0.038579	-0.214609	0.148510	-0.233097	-0.131313	-0.093827	-0.028061
Depth	-0.081020	-0.085861	1.000000	-0.074609	0.174663	0.023457	-0.076918	-0.015254	-0.171162	-0.073832	-0.016467	-0.134002
Depth Error	0.007080	0.020552	-0.074609	1.000000	-0.320579	-0.135880	0.618254	-0.093292	0.357704	-0.077423	0.644593	0.094398
Depth Seismic Stations	0.433815	0.015924	0.174663	-0.320579	1.000000	0.440582	-0.385993	0.813374	-0.420556	0.056619	-0.214959	-0.158620
Magnitude	0.034987	0.038579	0.023457	-0.135880	0.440582	1.000000	0.135573	-0.003972	-0.233579	-0.092609	-0.078406	0.075865
Magnitude Error	0.113208	-0.214609	-0.076918	0.618254	-0.385993	0.135573	1.000000	-0.334062	0.567411	-0.076744	0.617721	0.032616
Magnitude Seismic Stations	0.315075	0.148510	-0.015254	-0.093292	0.813374	-0.003972	-0.334062	1.000000	-0.334864	-0.117606	-0.093143	-0.167473
Azimuthal Gap	0.050794	-0.233097	-0.171162	0.357704	-0.420556	-0.233579	0.567411	-0.334864	1.000000	-0.033482	0.396450	-0.056217
Horizontal Distance	-0.396768	-0.131313	-0.073832	-0.077423	0.056619	-0.092609	-0.076744	-0.117606	-0.033482	1.000000	0.126877	0.035778
Horizontal Error	0.015625	-0.093827	-0.016467	0.644593	-0.214959	-0.078406	0.617721	-0.093143	0.396450	0.126877	1.000000	0.157842
Root Mean Square	-0.214762	-0.028061	-0.134002	0.094398	-0.158620	0.075865	0.032616	-0.167473	-0.056217	0.035778	0.157842	1.000000

In[8]: plt.figure(figsize=(10,5))

sns.heatmap(df.corr(numeric_only= True),annot = True)

plt.show()

Out[8]:



Some common data preprocessing tasks include:

- **Data Cleaning:**
  - Identifying and correcting errors.
  - Removing duplicate records.
  - Filling in missing values.
- **Data Transformation:**
  - Converting the data into a format.
  - Converting categorical data to numerical data.
  - Scaling the data to a suitable range.

Program:

```
import pandas as pd import seaborn as sns import

matplotlib.pyplot as plt from

sklearn.model_selection import train_test_split from

sklearn.compose import ColumnTransformer from

sklearn.preprocessing import StandardScaler  from

sklearn.preprocessing import OneHotEncoder from

sklearn.pipeline import Pipeline
```

Load the dataset:

In[1]:

```
df = pd.read_csv('C:/Users/barat/Downloads/archive/database.csv')
```

Exploratory Data Analysis:

In[2]:

```
missing_values = df.isnull().sum() print(missing_values)
```

Feature Engineering:

In[3]: description =

df.describe()

print(description)

Data Splitting:

In[4]:

X = df.drop('Depth',axis=1) y=df['Depth']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

random_state=42) categorical_cols = ['Longitude'] transformers=[

('num', StandardScaler(),['Magnitude','Date']),

('cat',OneHotEncoder(),categorical_cols)

]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

random_state=42) print(f"X_train shape:{X_train.shape}")

print(f"X_test shape:{X_test.shape}") print(f"y_train

shape:{y_train.shape}") print(f"y_test shape:{y_test.shape}")

Preprocessing and feature sealing using pipeline: model

= Pipeline(

[

('preprocessor',preprocessor),

])

X_train = model.fit_transform(X_train) X_test

= model.transform(X_test)

Print("PreProcessing Complete!!!")

Output:

1.Checking for missing values:

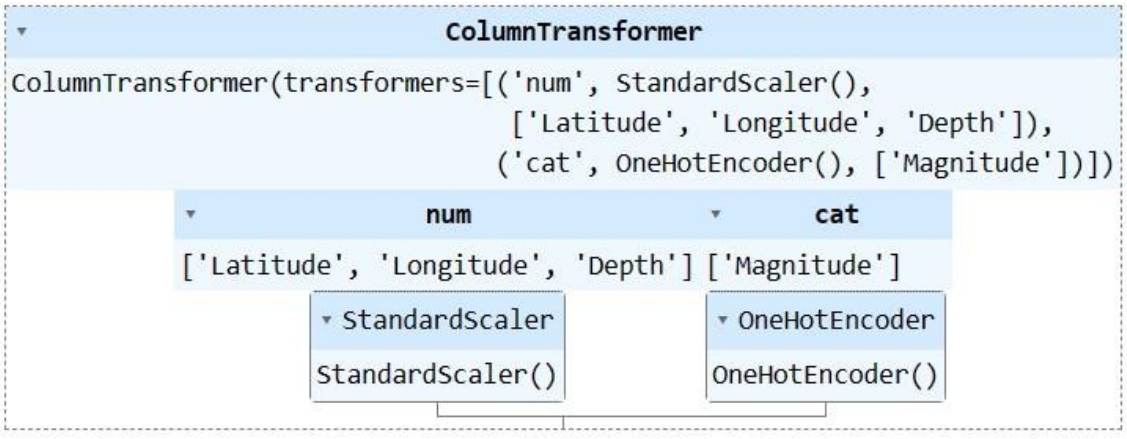
```
Date                            0
Time                            0
Latitude                        0
Longitude                       0
Type                            0
Depth                           0
Depth Error                 18951
Depth Seismic Stations      16315
Magnitude                       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
```

2.Descriptive Statistics:

	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square
count	23412.000000	4461.000000	7097.000000	23412.000000	327.000000	2564.000000	7299.000000	1604.000000	1156.000000	17352.000000
mean	70.767911	4.993115	275.364098	5.882531	0.071820	48.944618	44.163532	3.992660	7.662759	1.022784
std	122.651898	4.875184	162.141631	0.423066	0.051466	62.943106	32.141486	5.377262	10.430396	0.188545
min	-1.100000	0.000000	0.000000	5.500000	0.000000	0.000000	0.000000	0.004505	0.085000	0.000000
25%	14.522500	1.800000	146.000000	5.600000	0.046000	10.000000	24.100000	0.968750	5.300000	0.900000
50%	33.000000	3.500000	255.000000	5.700000	0.059000	28.000000	36.000000	2.319500	6.700000	1.000000
75%	54.000000	6.300000	384.000000	6.000000	0.075500	66.000000	54.000000	4.724500	8.100000	1.130000
max	700.000000	91.295000	934.000000	9.100000	0.410000	821.000000	360.000000	37.874000	99.000000	3.440000

3.Data Splitting:

```
X_train shape:(18729, 42168)
X_test shape:(4683, 42168)
y_train shape:(18729,)
y_test shape:(4683,)
```



PreProcessing Complete!!!

Conclusion:

- Understanding the data's structure, and may potential issues through exploratory data analysis (EDA) is essential for informed decision-making.

- Data preprocessing emerged as a pivot aspect of this process. It involves cleaning, transforming, and refining the dataset to ensure that it aligns with the requirements of machine learning algorithms.

- With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a house price prediction model.