IBM NaanMudhalvan
ARTIFICIAL INTELLIGENCE

Project Title : Earthquake Prediction Using Python
Phase 5 : Documentation

- Clearly outline the problem statement, design thinking process, and the phases of development.
- Describe the dataset used, data preprocessing steps, and feature exploration techniques.
- Document any innovative techniques or approaches used during the development.

Workbook Link : [Google Colab]

Problem Definition :

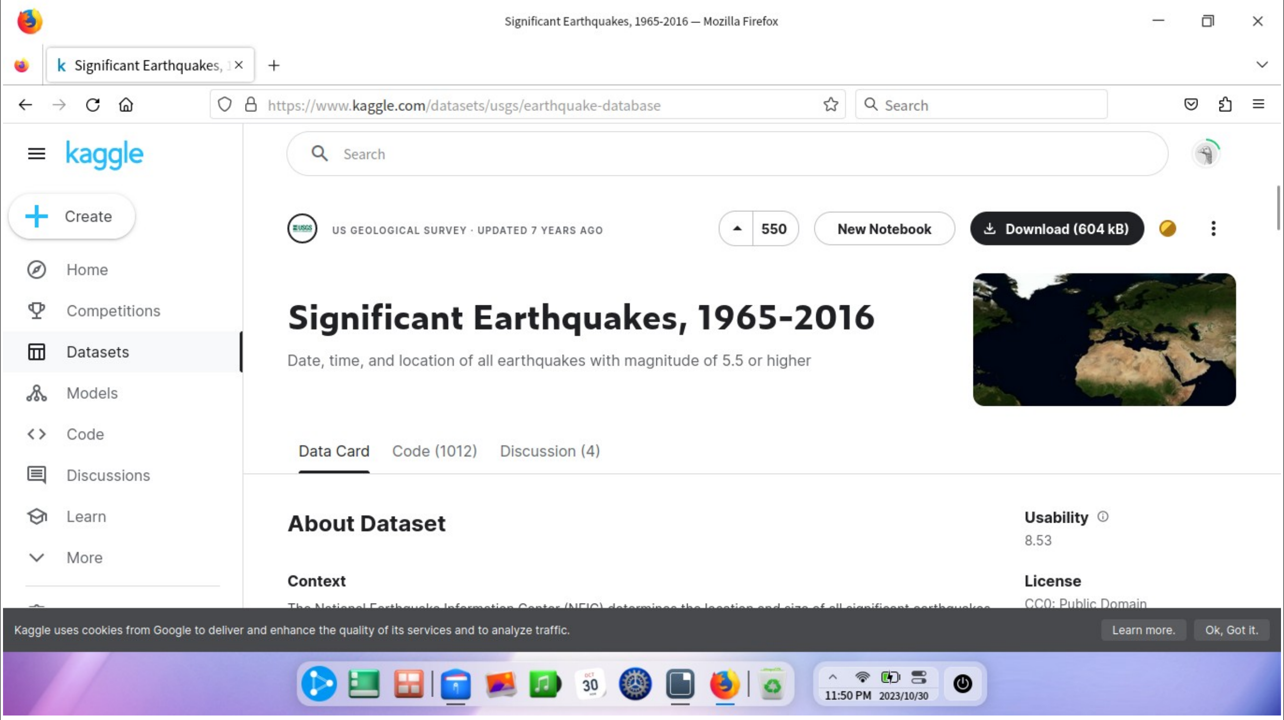
The problem at hand is to develop an earthquake prediction model using a kaggle dataset. The primary objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and ultimately construct a neural network model that can predict earthquake magnitudes based on the provided features.

DESIGN THINKING

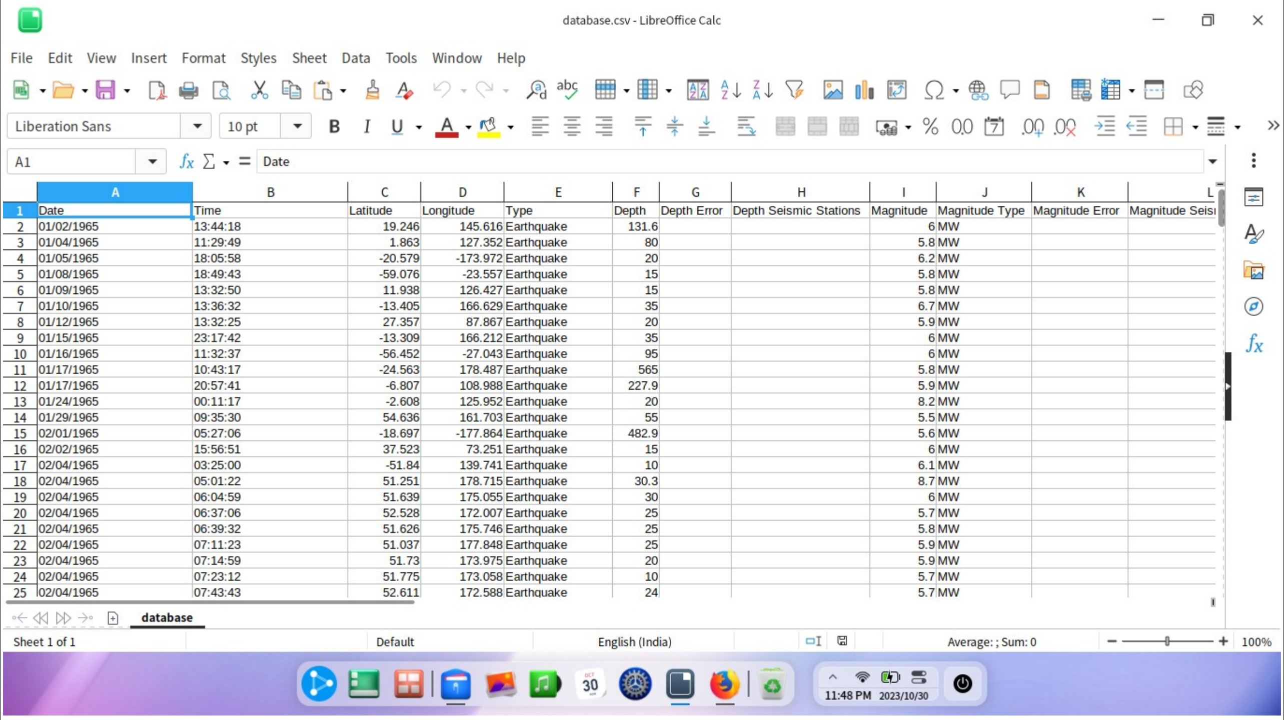
Data Source

The first step in solving this problem is selecting a suitable kaggle dataset that contains earthquake data. This dataset should include essential features such as date, time, latitude, longitude, depth, and magnitude. The choice of the dataset is crucial as it forms the foundation of our model.

Dataset Source :



Sample Dataset:



	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magnitude Seis
2	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6			6	MW		
3	01/04/1965	11:29:49	1.863	127.352	Earthquake	80			5.8	MW		
4	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20			6.2	MW		
5	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15			5.8	MW		
6	01/09/1965	13:32:50	11.938	126.427	Earthquake	15			5.8	MW		
7	01/10/1965	13:36:32	-13.405	166.629	Earthquake	35			6.7	MW		
8	01/12/1965	13:32:25	27.357	87.867	Earthquake	20			5.9	MW		
9	01/15/1965	23:17:42	-13.309	166.212	Earthquake	35			6	MW		
10	01/16/1965	11:32:37	-56.452	-27.043	Earthquake	95			6	MW		
11	01/17/1965	10:43:17	-24.563	178.487	Earthquake	565			5.8	MW		
12	01/17/1965	20:57:41	-6.807	108.988	Earthquake	227.9			5.9	MW		
13	01/24/1965	00:11:17	-2.608	125.952	Earthquake	20			8.2	MW		
14	01/29/1965	09:35:30	54.636	161.703	Earthquake	55			5.5	MW		
15	02/01/1965	05:27:06	-18.697	-177.864	Earthquake	482.9			5.6	MW		
16	02/02/1965	15:56:51	37.523	73.251	Earthquake	15			6	MW		
17	02/04/1965	03:25:00	-51.84	139.741	Earthquake	10			6.1	MW		
18	02/04/1965	05:01:22	51.251	178.715	Earthquake	30.3			8.7	MW		
19	02/04/1965	06:04:59	51.639	175.055	Earthquake	30			6	MW		
20	02/04/1965	06:37:06	52.528	172.007	Earthquake	25			5.7	MW		
21	02/04/1965	06:39:32	51.626	175.746	Earthquake	25			5.8	MW		
22	02/04/1965	07:11:23	51.037	177.848	Earthquake	25			5.9	MW		
23	02/04/1965	07:14:59	51.73	173.975	Earthquake	20			5.9	MW		
24	02/04/1965	07:23:12	51.775	173.058	Earthquake	10			5.7	MW		
25	02/04/1965	07:43:43	52.611	172.588	Earthquake	24			5.7	MW		

FEATURE EXPLORATION

Once the dataset is acquired, it's essential to dive into feature exploration. This phase involves:

1. Data Inspection:

Carefully examining the dataset to understand its structure, data types, and any missing values.

2. Statistical Analysis:

Calculating summary statistics, including mean, median, standard deviation, and quartiles for each feature. This will help us identify outliers and understand the data's distribution.

3. Correlation Analysis:

Investigating the correlations between features, especially between earthquake magnitude and other variables. Identifying highly correlated features can be beneficial for model development.

VISUALIZATION

Visualization plays a crucial role in gaining insights from the data. In this phase:

1. World Map Visualization:

Creating a world map visualization to display the geographical distribution of earthquakes. This can help identify earthquake-prone regions and patterns.

2. Time Series Plots:

Visualizing the earthquake data over time to detect any temporal trends or seasonality.

DATA SPLITTING

To evaluate our model effectively, we need to split the dataset into two subsets:

1. Training Set:

This set will be used to train our neural network model. It should contain a significant portion of the data, ensuring that the model learns from a diverse range of examples.

2. Test Set:

The test set is crucial for evaluating the model's performance. It should be separate from the training data and used to assess how well the model generalizes to unseen earthquake data.

MODEL DEVELOPMENT

In this phase, we focus on building the earthquake prediction model using a neural network. Key steps include:

1. Data Pre processing:

Preparing the data for model input, which may involve normalization, scaling, or encoding categorical variables.

2. Neural Network Architecture:

Designing the architecture of the neural network. This includes defining the number of layers, neurons, activation functions, and loss functions.

3. Model Training:

Training the neural network on the training set using appropriate optimization techniques, such as stochastic gradient descent (SGD) or Adam.

TRAINING AND EVALUATION

The final phase involves training the model and evaluating its performance:

1. Model Training:

Fit the neural network to the training data and monitor its convergence. Adjust hyper parameters as needed to optimize performance.

2. Model Evaluation:

Assess the model's performance on the test set using appropriate evaluation metrics, such as mean squared error (MSE) or root mean squared error (RMSE).

3. Fine-Tuning:

If the model's performance is not satisfactory, consider fine-tuning the architecture or exploring advanced techniques like hyper parameter tuning or different neural network architectures.

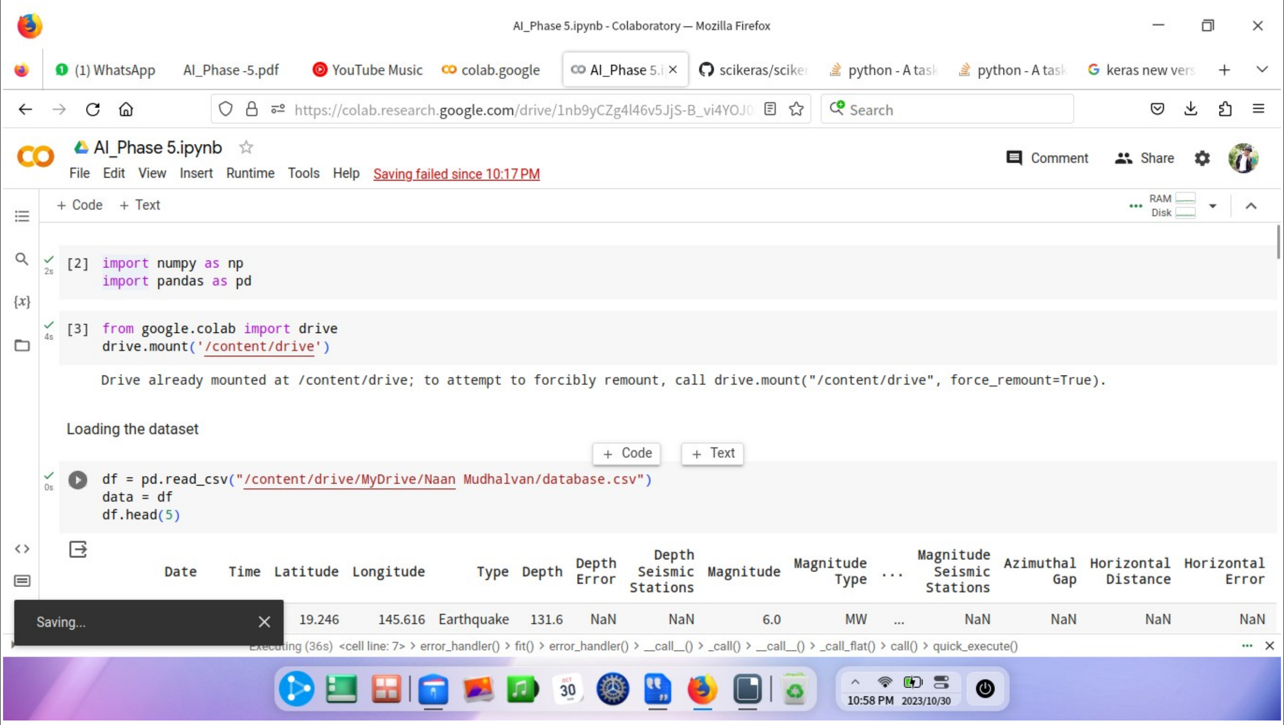
PROGRAM :

Original file is located at

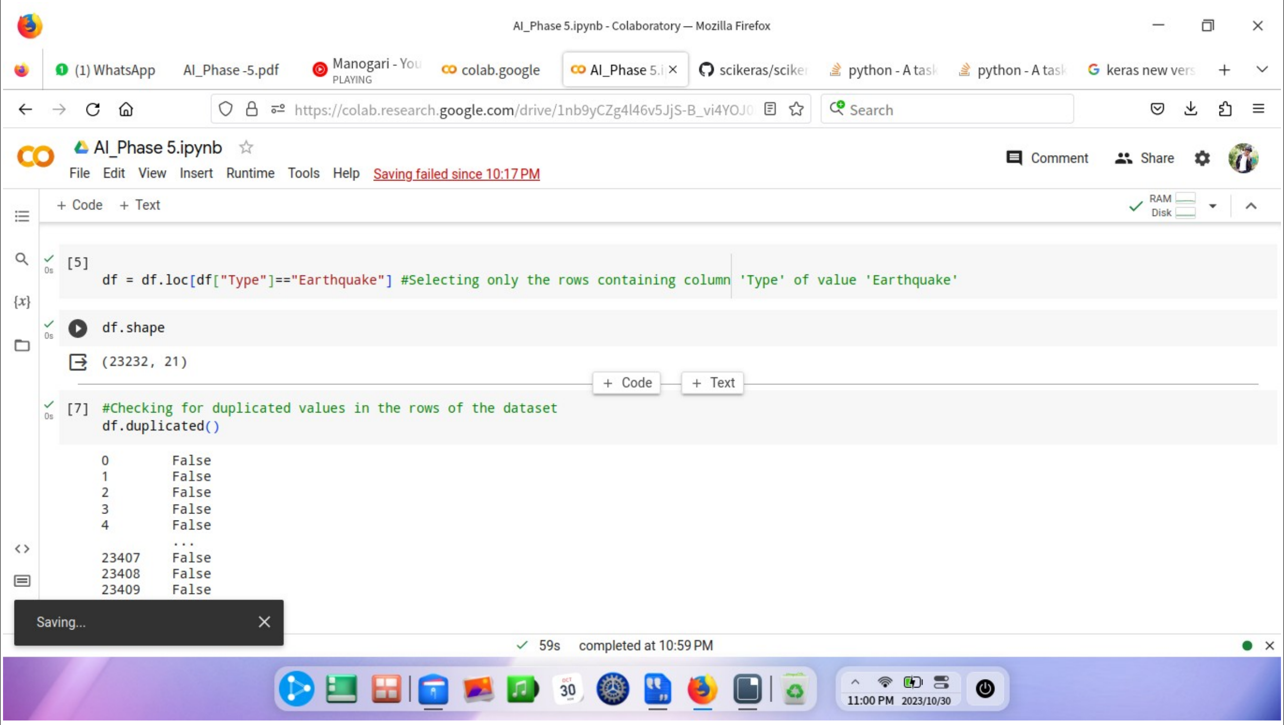
Google Colab

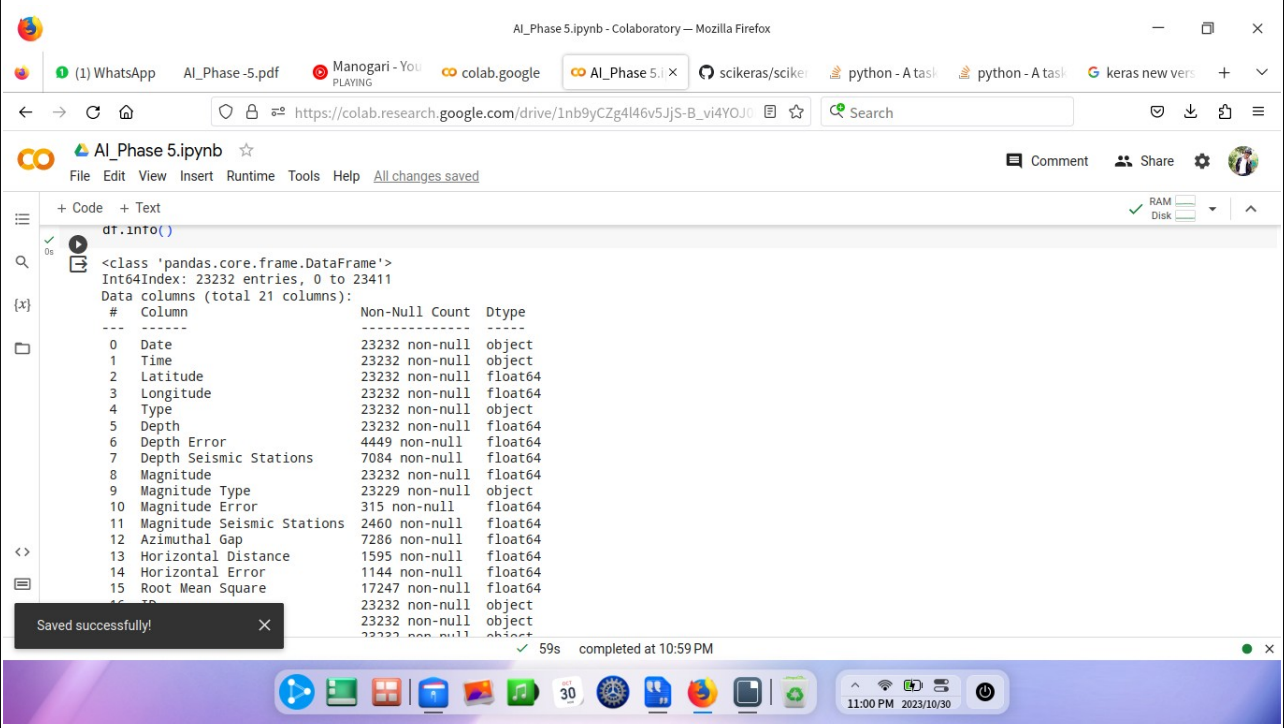
```
# Importing the Libraries
import pandas as pd
import numpy as np
# Loading the Dataset
data = pd.read_csv('database.csv')
data.head()
```

OUTPUT :




```
# Checking the Shape of the Dataset
data.shape
# Checking the Number of Entities
data.columns
# Checking Descriptive Structure of the data
data.describe()
# Checking Duplicated Rows.
data.duplicated()
# Checking the Data Information
data.info()
df = pd.DataFrame(data)
# Checking Categorical and Numerical Columns
# 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)
# Checking total number of Values in Categorical
Columns
df[cat_col].nunique()
# Checking total number of Values in Numerical
Columns
df[num_col].nunique()
# Checking the Missing Values Percentage
round((df.isnull().sum()/df.shape[0])*100,2)
```


AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox
(1) WhatsApp
AI_Phase -5.pdf
Manogari - You
PLAYING
colab.google
AI_Phase 5.i
scikeras/scike
python - A task
python - A task
keras new vers
https://colab.research.google.com/drive/1nb9yCZg4l46v5JjS-B_vi4YOJ
Search
AI_Phase 5.ipynb
File Edit View Insert Runtime Tools Help Saving failed since 10:17 PM
Comment
Share
+ Code + Text
RAM
Disk
[5]
df = df.loc[df["Type"]=="Earthquake"] #Selecting only the rows containing column 'Type' of value 'Earthquake'
df.shape
(23232, 21)
+ Code + Text
[7] #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
Saving...
59s completed at 10:59 PM
11:00 PM 2023/10/30


AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox
(1) WhatsApp
AI_Phase -5.pdf
Manogari - You
PLAYING
colab.google
AI_Phase 5.i
scikeras/scike
python - A task
python - A task
keras new vers
https://colab.research.google.com/drive/1nb9yCZg4l46v5JjS-B_vi4YOJ
Search
AI_Phase 5.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23232 entries, 0 to 23411
Data columns (total 21 columns):
Column Non-Null Count Dtype
0 Date 23232 non-null object
1 Time 23232 non-null object
2 Latitude 23232 non-null float64
3 Longitude 23232 non-null float64
4 Type 23232 non-null object
5 Depth 23232 non-null float64
6 Depth Error 4449 non-null float64
7 Depth Seismic Stations 7084 non-null float64
8 Magnitude 23232 non-null float64
9 Magnitude Type 23229 non-null object
10 Magnitude Error 315 non-null float64
11 Magnitude Seismic Stations 2460 non-null float64
12 Azimuthal Gap 7286 non-null float64
13 Horizontal Distance 1595 non-null float64
14 Horizontal Error 1144 non-null float64
15 Root Mean Square 17247 non-null float64
Saved successfully!
59s completed at 10:59 PM
11:00 PM 2023/10/30

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

	Latitude	Longitude	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Error	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square
count	23232.000000	23232.000000	23232.000000	4449.000000	7084.000000	23232.000000	315.000000	2460.000000	7286.000000	1595.000000	1144.000000	17247.000000
mean	1.386383	39.746049	71.313913	4.921323	275.796302	5.882763	0.066197	47.608943	43.807400	4.006932	6.704676	1.023518
std	29.929060	125.751903	122.968384	4.681354	161.961894	0.424032	0.039592	63.127239	30.947508	5.389051	4.629406	0.186946
min	-77.080000	-179.997000	-1.100000	0.000000	0.000000	5.500000	0.000000	0.000000	0.000000	0.004505	0.085000	0.000000
25%	-18.719500	-76.366500	15.000000	1.800000	147.000000	5.600000	0.046000	9.750000	24.100000	0.963300	5.300000	0.900000
50%	-3.680000	106.349000	33.000000	3.500000	255.000000	5.700000	0.058000	28.000000	36.000000	2.332000	6.700000	1.000000
75%	24.968500	145.290250	54.800000	6.200000	384.000000	6.000000	0.073000	62.000000	53.775000	4.730000	8.000000	1.130000
max	86.005000	179.998000	700.000000	91.295000	934.000000	9.100000	0.350000	821.000000	360.000000	37.874000	99.000000	3.220000

✓ 59s completed at 10:59 PM

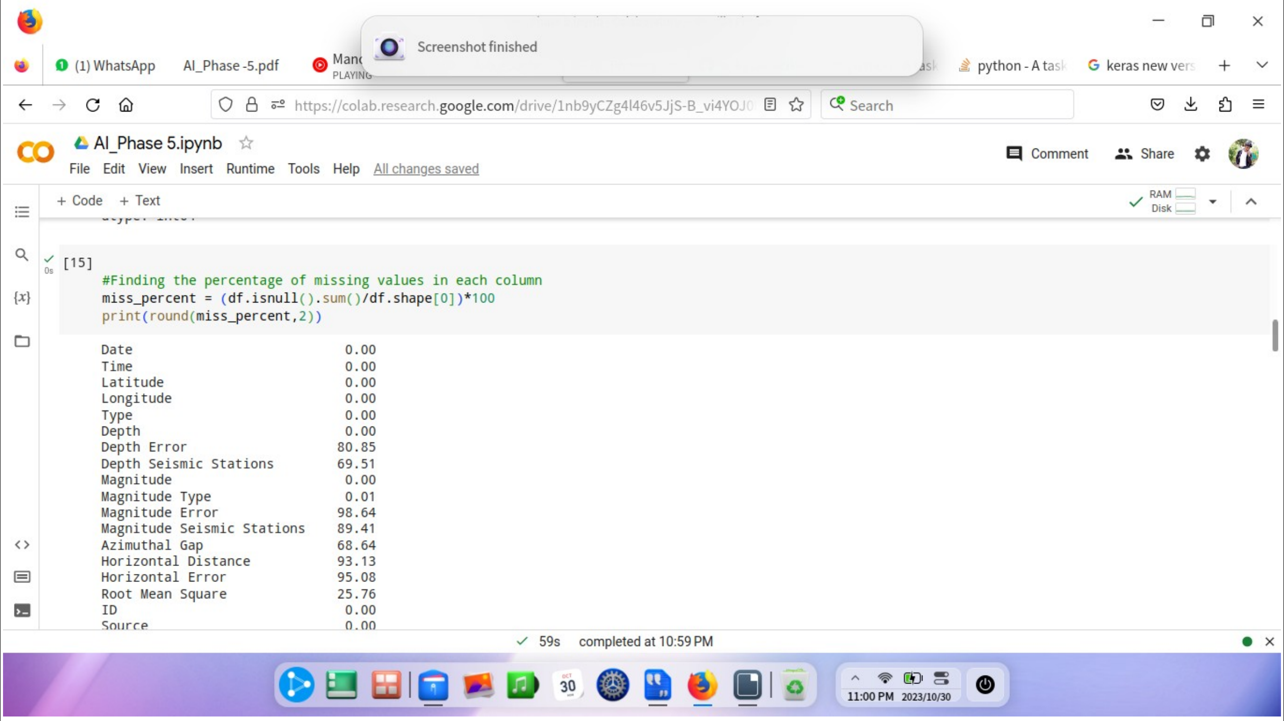
```
[11]
# Numerical columns
num_col = [col for col in df.columns if df[col].dtype != 'object']
print('Numerical columns :',num_col)
```

```
Categorical columns : ['Date', 'Time', 'Type', 'Magnitude Type', 'ID', 'Source', 'Location Source', 'Magnitude Source', 'Status']
Numerical columns : ['Latitude', 'Longitude', 'Depth', 'Depth Error', 'Depth Seismic Stations', 'Magnitude', 'Magnitude Error', 'Magnitude Seismic St
```

```
[12]
#Checking number of unique values in numerical columns
df[num_col].nunique()
```

```
Latitude                      20534
Longitude                     21312
Depth                          3482
Depth Error                     297
Depth Seismic Stations          736
Magnitude                        60
Magnitude Error                  90
Magnitude Seismic Stations      241
Azimuthal Gap                  1105
Horizontal Distance            1441
Horizontal Error                186
Root Mean Square                187
dtype: int64
```

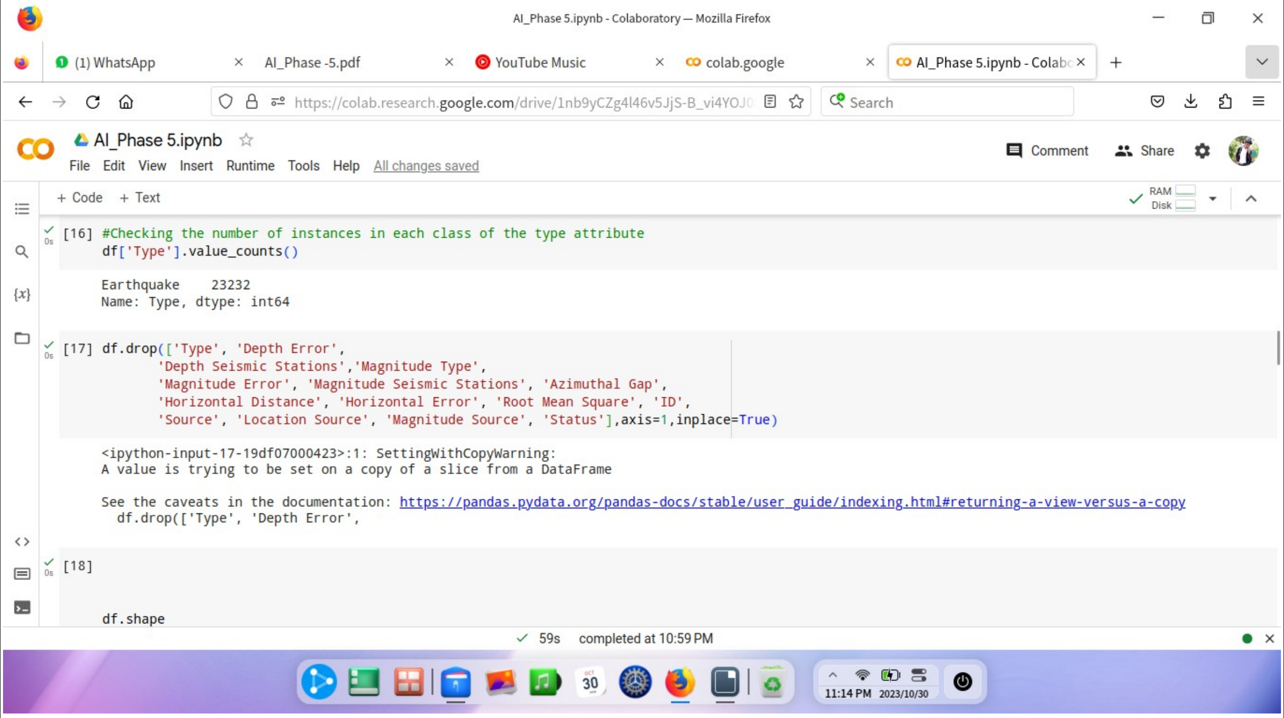
✓ 59s completed at 10:59 PM

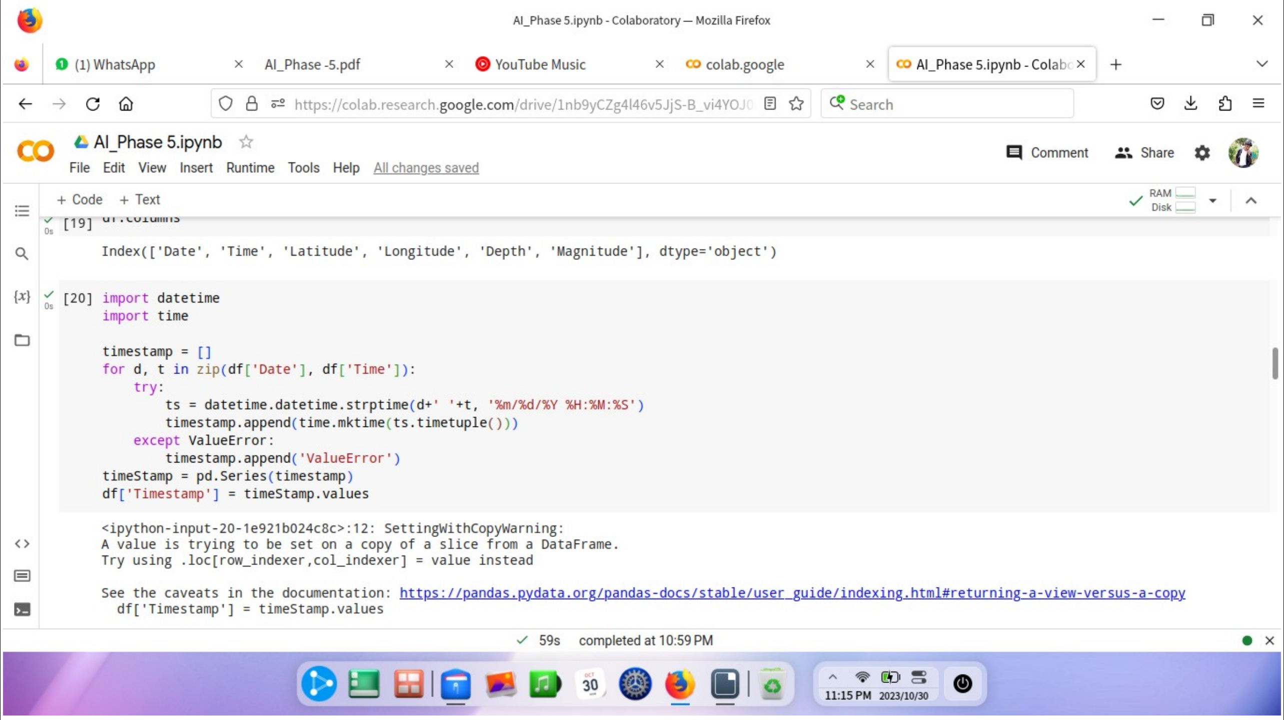



```
# Creating Timestamp Column from Data and Time
Column
import datetime
import time
timestamp = []
for d, t in zip(data['Date'], data['Time']):
try:
ts = datetime.datetime.strptime(d+' '+t,
'%m/%d/%Y %H:%M:%S')
timestamp.append(time.mktime(ts.timetuple()))
except ValueError:
# print('ValueError')
timestamp.append('ValueError')
# Converting the Tuple values into Series Values
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

```
# Droping the Date and Time Columns.
final_data = df.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp !=
'ValueError']
final_data.head()
# Removal Of Unwanted Columns
df1 = df.drop(columns=['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', 'Date', 'Time'])
# Checking the Shape of Dataset after
Removing the Columns
df1.shape
df1.head(10)
# Checking Columns
df1.columns
# Checking the Missing Values Percentage
round((df1.isnull().sum()/df1.shape[0])*100,2)
# Checking the Data Information After droping
the Unwanted Columns
df1.info()
# Checking the Descriptive Structure of the
Data after the removal of Unwanted Columns
df1.describe()
# Checking Categorical and Numerical
Columns
# Categorical columns
cat_col = [col for col in df1.columns if
df1[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in df1.columns if
df1[col].dtype != 'object']
print('Numerical columns :',num_col)
# Checking total number of Values in
Categorical Columns
df1[cat_col].nunique()
# Checking total number of Values in
Numerical Columns
df[num_col].nunique()
# Let's check the null values again
df1.isnull().sum()
```

OUTPUT:


AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox
(1) WhatsApp
AI_Phase -5.pdf
YouTube Music
colab.google
AI_Phase 5.ipynb - Colab
https://colab.research.google.com/drive/1nb9yCZg4l46v5JjS-B_vi4YOJ
Search
AI_Phase 5.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
[16] #Checking the number of instances in each class of the type attribute
df['Type'].value_counts()
Earthquake 23232
Name: Type, dtype: int64
[17] 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)
<ipython-input-17-19df07000423>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df.drop(['Type', 'Depth Error',
[18]
df.shape
59s completed at 10:59 PM
11:14 PM 2023/10/30


AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox
(1) WhatsApp
AI_Phase -5.pdf
YouTube Music
colab.google
AI_Phase 5.ipynb - Colab
https://colab.research.google.com/drive/1nb9yCZg4l46v5JjS-B_vi4YOJ
Search
AI_Phase 5.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
[19]
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude'], dtype='object')
[20] 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
<ipython-input-20-1e921b024c8c>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df['Timestamp'] = timeStamp.values
59s completed at 10:59 PM
11:15 PM 2023/10/30

AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox

AI_Phase 5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[22] df.loc[df["Magnitude"] > 7, "Magnitude"] = 1
     df.loc[df["Magnitude"] < 7, "Magnitude"] = 0
     df['Magnitude']=df['Magnitude'].astype("int64")
```

```
[23] df['Magnitude'].value_counts()

     0    23061
     7      168
     Name: Magnitude, dtype: int64
```

Data Visualization

```
[24] !pip install basemap

     Requirement already satisfied: basemap in /usr/local/lib/python3.10/dist-packages (1.3.8)
     Requirement already satisfied: basemap-data<1.4,>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from basemap) (1.3.2)
     Requirement already satisfied: pyshp<2.4,>=1.2 in /usr/local/lib/python3.10/dist-packages (from basemap) (2.3.1)
     Requirement already satisfied: matplotlib<3.8,>=1.5 in /usr/local/lib/python3.10/dist-packages (from basemap) (3.7.1)
     Requirement already satisfied: pyproj<3.7.0,>=1.9.3 in /usr/local/lib/python3.10/dist-packages (from basemap) (3.6.1)
     Requirement already satisfied: numpy<1.26,>=1.21 in /usr/local/lib/python3.10/dist-packages (from basemap) (1.23.5)
```

59s completed at 10:59 PM

11:15 PM 2023/10/30

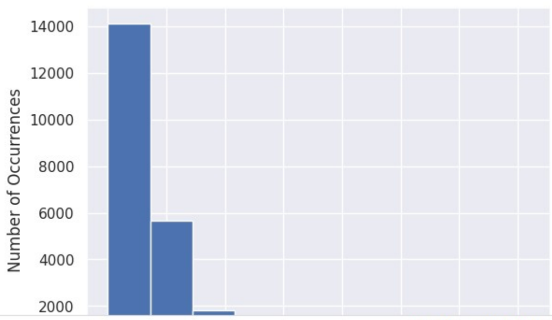
AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox

AI_Phase 5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[27] plt.hist(data['Magnitude'])
     plt.xlabel('Magnitude Size')
     plt.ylabel('Number of Occurrences')

     Text(0, 0.5, 'Number of Occurrences')
```



59s completed at 10:59 PM

11:16 PM 2023/10/30

```
[28]
Index(['Date', 'Time', 'Latitude', 'Longitude', 'Type', 'Depth', 'Depth Error',
       'Depth Seismic Stations', 'Magnitude', 'Magnitude Type',
       'Magnitude Error', 'Magnitude Seismic Stations', 'Azimuthal Gap',
       'Horizontal Distance', 'Horizontal Error', 'Root Mean Square', 'ID',
       'Source', 'Location Source', 'Magnitude Source', 'Status'],
      dtype='object')
```

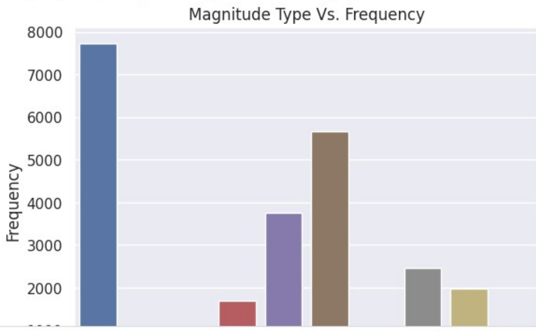
```
[29] #Earthquakes with magnitude greater than 7
     G8 = data[data['Magnitude']>7]
     G8['Location Source'].value_counts()
```

```
US        467
ISCGEM     92
CI          3
H           1
AG          1
SPE         1
AGS         1
NC          1
AEIC        1
WEL         1
GUC         1
Name: Location Source, dtype: int64
```

59s completed at 10:59 PM

```
sns.countplot(x="Magnitude Type",data = data)
[30] plt.ylabel("Frequency")
     plt.title("Magnitude Type Vs. Frequency")
```

```
Text(0.5, 1.0, 'Magnitude Type Vs. Frequency')
```



59s completed at 10:59 PM

AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox

AI_Phase 5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[31] def get_marker_color(magnitude):
         if magnitude < 6.2:
             return ('go')
         elif magnitude < 7.5:
             return ('yo')
         else:
             return ('ro')

     plt.figure(figsize=(14,10),edgecolor='w')

     map = Basemap(projection='cyl', llcrnrlat=-90, urcrnrlat=90,
                 llcrnrlon=-180, urcrnrlon=180,)
     map.drawcoastlines()
     map.drawcountries()
     map.fillcontinents(color = 'gray')
     map.drawmapboundary()
     map.drawmeridians(np.arange(0, 360, 30))
     map.drawparallels(np.arange(-90, 90, 30))

     # Reading the longitude, latitude and magnitude values from the dataset
     lons = data['Longitude'].values
     lats = data['Latitude'].values
```

✓ 59s completed at 10:59 PM

11:17 PM 2023/10/30

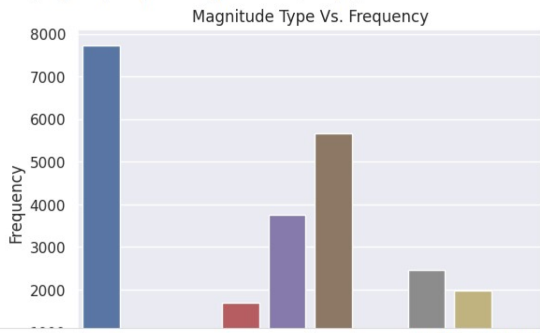
AI_Phase 5.ipynb - Colaboratory — Mozilla Firefox

AI_Phase 5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
     sns.countplot(x="Magnitude Type",data = data)
[30] plt.ylabel("Frequency")
     plt.title("Magnitude Type Vs. Frequency")
```

```
Text(0.5, 1.0, 'Magnitude Type Vs. Frequency')
```



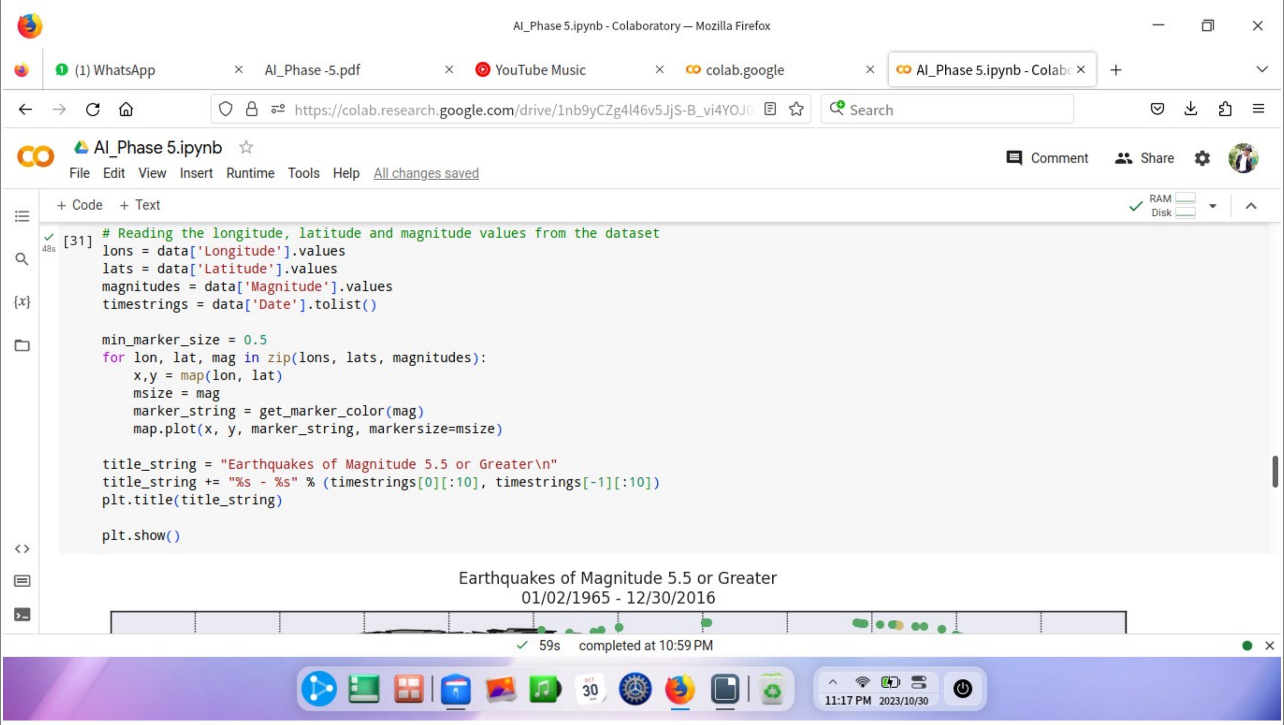
✓ 59s completed at 10:59 PM

11:16 PM 2023/10/30

PROGRAM :

```
# Importing necessary libraries
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
# Reading the dataset from the specified location
data = pd.read_csv('database.csv')
# Displaying the loaded dataset
data
# Providing information about the dataset,
including data types and missing values
data.info()
# Dropping the 'ID' column from the dataset
data = data.drop('ID', axis=1)
# Identifying and dropping columns with more than
66% missing values
null_columns = data.loc[:, data.isna().sum() > 0.66 *
data.shape[0]].columns
data = data.drop(null_columns, axis=1)
# Displaying the count of missing values in each
column
data.isna().sum()
# Filling missing values in the 'Root Mean Square'
column with the mean value
data['Root Mean Square'] = data['Root Mean
Square'].fillna(data['Root Mean Square'].mean())
# Dropping rows with any remaining missing
values and resetting the index
data = data.dropna(axis=0).reset_index(drop=True)
# Confirming there are no more missing values in
the dataset
data.isna().sum().sum()
# Feature Engineering: 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:])
# Converting 'Month' to integer type
data['Month'] = data['Month'].astype(np.int)
# Handling invalid 'Year' entries and converting to
integer type
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' and displaying the
modified dataset
```

```
data['Hour'] = data['Time'].apply(lambda x:
np.int(x[0:2]))
data
# Displaying the shape and columns of the final
dataset
data.shape
data.columns
# Selecting relevant columns and displaying the
first few rows of the modified dataset
data = data[['Date', 'Time', 'Latitude', 'Longitude',
'Depth', 'Magnitude']]
data.head()
# 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:
# Handling cases where timestamp conversion
fails
timestamp.append('ValueError')
# Creating a new 'Timestamp' column in the
dataset
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
# Creating the final dataset by dropping 'Date' and
'Time' columns and removing rows with invalid
timestamps
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp !=
'ValueError']
final_data.head()
```



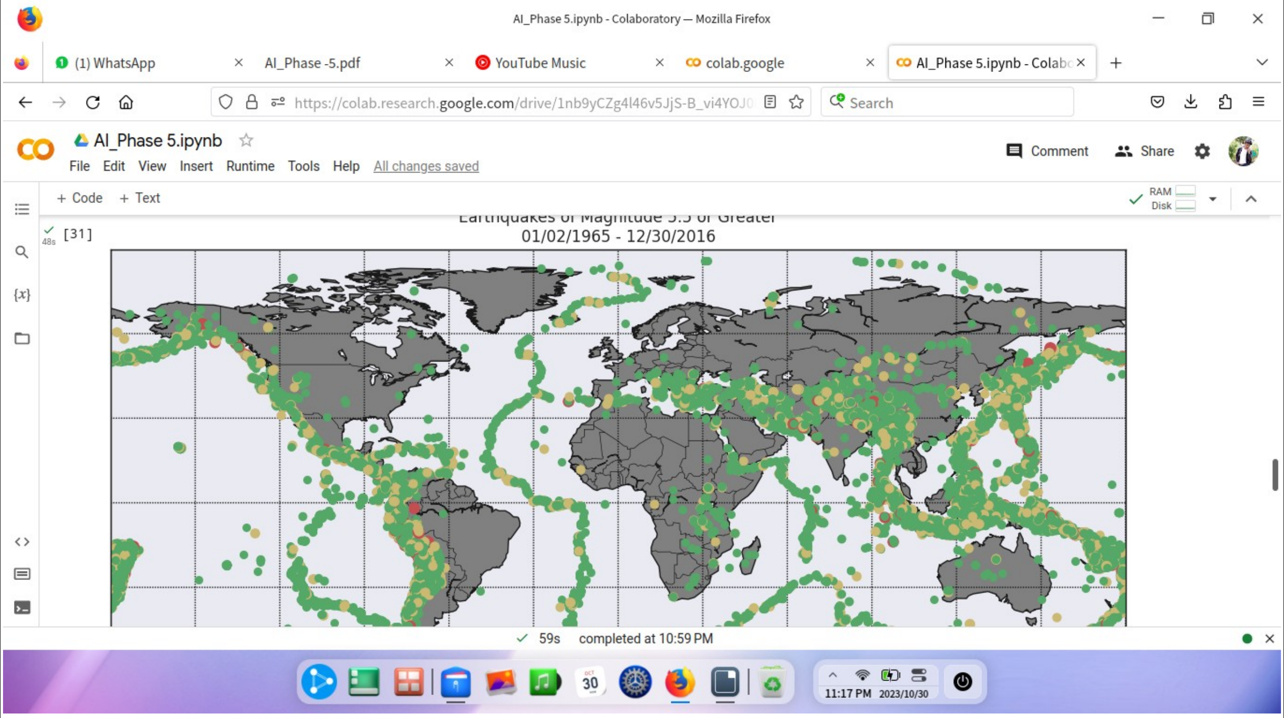

```
# Installing necessary libraries for data visualization
!pip3 install basemap
# Importing libraries for data visualization
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import seaborn as sns
sns.set(style="darkgrid")
# Displaying the minimum and maximum values of the
'Magnitude' column
print("Min Value: "+ str(data['Magnitude'].min()))
print("Max Value: " + str(data['Magnitude'].max()))
# Filtering earthquakes with magnitude greater than 8
and displaying counts by 'Location Source'
Greater_8 = data[data['Magnitude'] > 8]
Greater_8['Location Source'].value_counts()
# Similar counts for earthquakes with magnitude greater
than 7, 6, 5, and 4
Greater_7 = data[data['Magnitude'] > 7]
Greater_7['Location Source'].value_counts()
Greater_6 = data[data['Magnitude'] > 6]
Greater_6['Location Source'].value_counts()
Greater_5 = data[data['Magnitude'] > 5]
Greater_5['Location Source'].value_counts()
Greater_4 = data[data['Magnitude'] > 4]
Greater_4['Location Source'].value_counts()
# Histogram of earthquake magnitudes
plt.hist(data['Magnitude'])
plt.xlabel('Magnitude Size')
plt.ylabel('Number of Occurrences')
# Count plot of 'Magnitude Type'
```

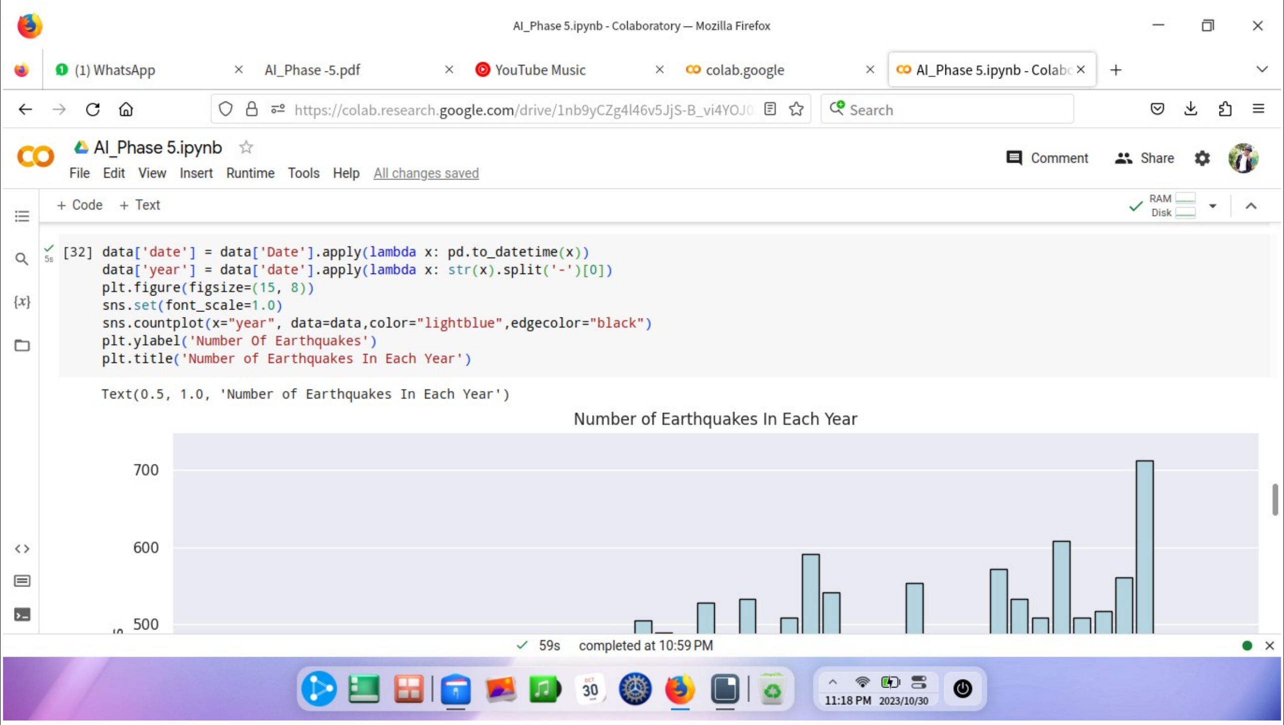
```
sns.countplot(x="Magnitude Type", data=data)
plt.ylabel('Frequency')
plt.title('Magnitude Type VS Frequency')
print(" local magnitude (ML), surface-wave magnitude
(Ms), body-wave magnitude (Mb), moment magnitude
(Mm)")
# Function to determine marker color based on
earthquake magnitude
def get_marker_color(magnitude):
if magnitude < 6.2:
return ('go')
elif magnitude < 7.5:
return ('yo')
else:
return ('ro')
# Basemap plot of earthquakes with different marker
colors based on magnitude
plt.figure(figsize=(14,10))
eq_map = Basemap(projection='robin', resolution = 'l',
lat_0=0, lon_0=-130)
eq_map.drawcoastlines()
eq_map.drawcountries()
eq_map.fillcontinents(color='gray')
eq_map.drawmapboundary()
eq_map.drawmeridians(np.arange(0, 360, 30))
lons = data['Longitude'].values
lats = data['Latitude'].values
magnitudes = data['Magnitude'].values
timestrings = data['Date'].tolist()
min_marker_size = 0.5
for lon, lat, mag in zip(lons, lats, magnitudes):
x,y = eq_map(lon, lat)
msize = mag
marker_string = get_marker_color(mag)
eq_map.plot(x, y, marker_string, markersize=msize)
title_string = "Earthquakes of Magnitude 5.5 or Greater\n"
title_string += "%s - %s" % (timestrings[0][:10],
timestrings[-1][:10])
plt.title(title_string)
plt.show()
# Count plot of the number of earthquakes in each year
import datetime
data['date'] = data['Date'].apply(lambda x:
pd.to_datetime(x))
data['year'] = data['date'].apply(lambda x: str(x).split('-')[0])
plt.figure(figsize=(15, 8))
sns.set(font_scale=1.0)
ax = sns.countplot(x="year", data=data, color="blue")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each Year')
# Displaying the top 5 years with the highest number of
```

```
earthquakes
data['year'].value_counts()[:5]
# Count plot of the number of earthquakes in each month
import datetime
data['date'] = data['Date'].apply(lambda x:
pd.to_datetime(x))
data['mon'] = data['date'].apply(lambda x: str(x).split('-')[1])
plt.figure(figsize=(10, 6))
sns.set(font_scale=1)
ax = sns.countplot(x="mon", data=data, color="green")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each month')
# Displaying the top 5 months with the highest number of
earthquakes
data['mon'].value_counts()[:5]
# Count plot of the number of earthquakes in each day of
the month
import datetime
data['date'] = data['Date'].apply(lambda x:
pd.to_datetime(x))
data['days'] = data['date'].apply(lambda x: str(x).split('-')[-
1])
plt.figure(figsize=(16, 8))
sns.set(font_scale=1.0)
ax = sns.countplot(x="days", data=data, color="orange")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.ylabel('Number Of Earthquakes')
plt.title('Number of Earthquakes In Each days')
# Displaying the top 5 days of the month with the highest
number of earthquakes
data['days'].value_counts()[:5]
# Scatter plot of the number of earthquakes per year from
1995 to 2016
x = data['year'].unique()
y = data['year'].value_counts()
count = []
for i in range(len(x)):
key = x[i]
count.append(y[key])
plt.figure(figsize=(15,12))
plt.scatter(x, count)
plt.title("Earthquake per year from 1995 to 2016")
plt.xlabel("Year")
plt.xticks(rotation=90)
plt.ylabel("Number of Earthquakes")
plt.yticks(rotation=30)
plt.show()
# Classification of earthquake magnitudes into classes
data.loc[data['Magnitude'] >= 8, 'Class'] = 'Disastrous'
data.loc[(data['Magnitude'] >= 7) & (data['Magnitude'] <
7.9), 'Class'] = 'Major'
```

```
data.loc[(data['Magnitude'] >= 6) & (data['Magnitude'] <
6.9), 'Class'] = 'Strong'
data.loc[(data['Magnitude'] >= 5.5) & (data['Magnitude'] <
5.9), 'Class'] = 'Moderate'
# Count plot of magnitude class distribution
sns.countplot(x='Class', data=data)
plt.ylabel('Frequency')
plt.title('Magnitude Class vs Frequency')
#Splitting the Data....
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape,
X_test.shape)
```

OUTPUT:





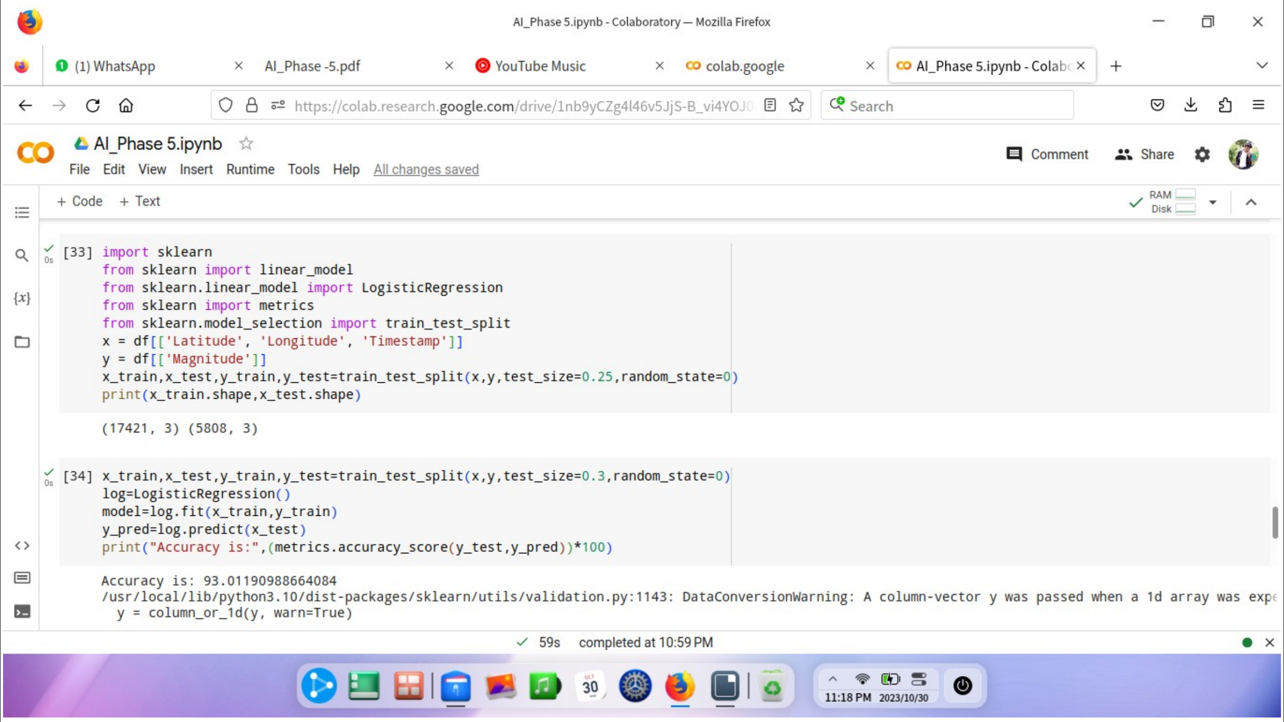
PROGRAM :

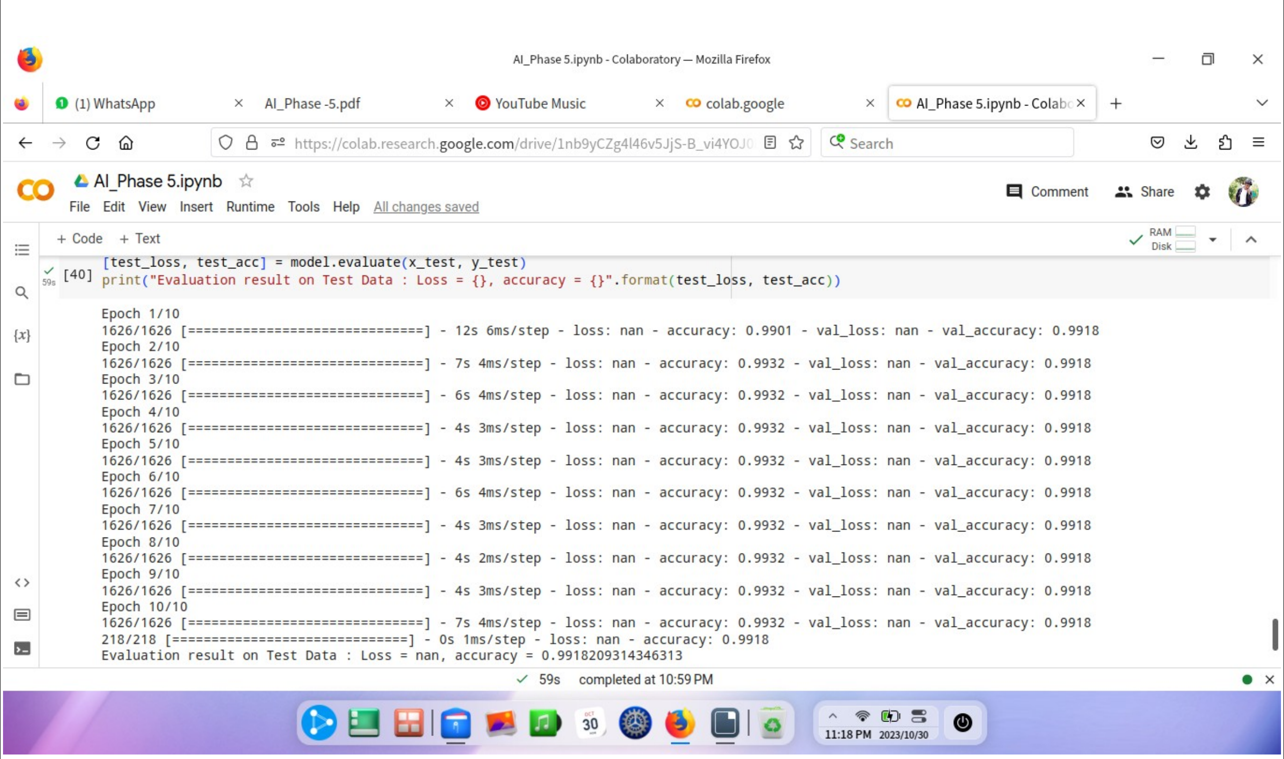
```
# Logistic Regression Model
# Importing necessary libraries
import sklearn
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
# Selecting features and target variable
x = df[['Latitude', 'Longitude', 'Timestamp']]
y = df[['Magnitude']]
# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=0)
print(x_train.shape, x_test.shape, y_train.shape,
x_test.shape)
# Creating and training the Logistic Regression
model
log = LogisticRegression()
model = log.fit(x_train, y_train)
y_pred = log.predict(x_test)
# Evaluating the model's accuracy
print("Accuracy is:", (metrics.accuracy_score(y_test,
y_pred)) * 100)
# Neural Network Model
# Importing necessary libraries
import sklearn
from sklearn.model_selection import train_test_split,
GridSearchCV
import numpy as np
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import
KerasClassifier
# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=0)
print(x_train.shape, x_test.shape, y_train.shape,
x_test.shape)
# Defining a function to create a neural network
model
def create_model(neurons, activation, optimizer,
loss):
model = Sequential()
model.add(Dense(neurons, activation=activation,
input_shape=(3,)))
model.add(Dense(neurons, activation=activation))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer=optimizer, loss=loss,
metrics=['accuracy'])
return model
# Creating a KerasClassifier
model = KerasClassifier(build_fn=create_model,
verbose=0)
# Defining a parameter grid for hyperparameter
tuning
param_grid = {
"neurons": [16, 64],
"batch_size": [10, 20],
"epochs": [10],
"activation": ['sigmoid', 'relu'],
"optimizer": ['SGD', 'Adadelta'],
"loss": ['squared_hinge']
}
# Converting data to numpy arrays
x_train = np.asarray(x_train).astype(np.float32)
y_train = np.asarray(y_train).astype(np.float32)
x_test = np.asarray(x_test).astype(np.float32)
y_test = np.asarray(y_test).astype(np.float32)
# Using GridSearchCV to find the best parameters
for the model
grid = GridSearchCV(estimator=model,
param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)
# Retrieving the best parameters
best_params = grid_result.best_params_
# Creating and training the final model with the best
parameters
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))
# Evaluating the final model on the test set
[test_loss, test_acc] = model.evaluate(x_test, y_test)
print("Evaluation result on Test Data: Loss = {},
accuracy = {}".format(test_loss, test_acc))
```

OUTPUT :





CONCLUSION:

In conclusion, the development of a machine learning model is a multifaceted journey that encompasses problem definition, data collection, preprocessing, exploratory data analysis, and feature engineering. The thoughtful selection of an appropriate model, meticulous training, and rigorous evaluation are pivotal to achieving robust predictive performance.