NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

CBCPC09

OPERATING SYSTEM

PROJECT CODE

EARTHQUAKE PREDICTION

Submitted by: KUNAL 2020UEA6603
ADARSH SINGH 2020UEA6592

Course Instructor : MS. SOMIYA RANI

Importing Modules

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
raw_data = pd.read_csv("major_earthquake_database.csv")
raw_data.head(6)
```

In [3]:

```
print(raw_data.shape)
print(raw_data.columns)
```

Selecting main features from earthquake_data and creating main_data object of those features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.

In [4]:

```
main_data = raw_data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
main_data.head()
```

checking any null value and min max value of different columns

In [5]:

```
main_data.info()
```

In [6]:

```
main_data.describe()
```

scaling of date and time to unix time to fit in our model

In [7]:

```
import datetime
import time
timestamp = []
for d, t in zip(main_data['Date'],main_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('Error')
```

adding timestamp column to our data

In [8]:

```
timeStamp = pd.Series(timestamp)
main_data['Timestamp'] = timeStamp.values
final_data = main_data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'Error']
final_data.head()
```

creating a fina_data object which doesn't include date and time columns and rows in which timestamp have error value

In [9]:

```
final_data = main_data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'Error']
final_data.head()
```

Visualising data

the earthquakes from the database in visualized on to the world map which shows clear representation of the locations where frequency of the earthquake will be more.

In [10]:

```
from mpl_toolkits.basemap import Basemap

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

longitudes = final_data["Longitude"].tolist()
latitudes = final_data["Latitude"].tolist()
x,y = m(longitudes,latitudes)
```

In [11]:

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

In [ ]:

```

```

Splitting the Data

In [12]:

```
x = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
print(x.shape)
print(y.shape)
```

In [13]:

```
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, y_test.shape)
```

Training Model

Random forest regression

In [14]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

rf = RandomForestRegressor(random_state=42)
rf.fit(x_train, y_train)
rf_pred=rf.predict(x_test)
```

In [15]:

```
print('Training score',rf.score(x_train,y_train))
print('Testing score',rf.score(x_test, y_test))
```

In [ ]:

```

```

In [44]:

```
from sklearn.model_selection import GridSearchCV

parameters = {'n_estimators':[10, 20, 50, 100, 200, 500]}

grid_obj = GridSearchCV(rf, parameters)
grid_fit = grid_obj.fit(x_train, y_train)
best_fit = grid_fit.best_estimator_
gvc_pre=best_fit.predict(x_test)
```

In [46]:

```
print(best_fit.score(x_train,y_train))
print(best_fit.score(x_test, y_test))
```

function to convert input date to unix timestamp

In [32]:

```
from datetime import datetime
epoch = datetime(1970, 1, 1)

def mapdateTotime(x):
    try:
        dt = datetime.strptime(x, "%m/%d/%Y")
    except ValueError:
        dt = datetime.strptime(x, "%Y-%m-%dT%H:%M:%S.%fZ")
    diff = dt - epoch
    return diff.total_seconds()
```

In [50]:

```
latitude = float(input("Enter Latitude between -77 to 86:"))
longitude = float(input("Enter Longitude between -180 to 180:"))
date = input("Enter the date (Month/Day/Year format):")
Input_values = np.asarray([[mapdateTotime(date),latitude,longitude]],dtype=np.float32)
output_values=rf.predict(Input_values)
print('magnitude depth(km)',output_values)
```

In [51]:

```
from mpl_toolkits.basemap import Basemap

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

longitudes = longitude
latitudes = latitude
x,y = m(longitudes,latitudes)
```

In [52]:

```
fig = plt.figure(figsize=(12,10))
plt.title("on map")

m.plot(x, y, "o", markersize = 2, color = 'red')
m.drawcoastlines()
m.fillcontinents(color='yellow',lake_color='aqua')
m.drawmapboundary()
```

```
m.drawcountries()
plt.show()
```

In [ ]:

```
```