EARTHQUAKE PREDICTION MODEL USING PYTHON

ARTIFICIAL INTELLIGENCE Phase-5

Team member

210221104040:Yokesh

The study of significant earthquakes between 1965 and 2016 in the context of problem definition and design thinking could involve several aspects:

Problem Statement:

Design an earthquake prediction model that uses historical earthquake data to forecast the likelihood and magnitude of future seismic events. The goal is to provide early warnings and improve disaster preparedness in earthquake-prone regions.

Design Thinking Approach:

1. Ideate:

- Brainstorm potential solutions and approaches to earthquake prediction.

- Explore data sources, technologies, and models that can be used for prediction.

2. Prototype:

- Create a prototype of the earthquake prediction model using historical earthquake data.

- Experiment with different machine learning algorithms and data preprocessing techniques.

3. Test:

- Evaluate the performance of the prototype using appropriate metrics.

- Gather feedback from experts and stakeholders to refine the model.

4.Implement:

- Develop a functional earthquake prediction system based on the prototype.

- Integrate the model with real-time data sources for continuous monitoring.

5.Monitor:

- Continuously monitor the model's predictions and assess its accuracy.

- Make improvements as new data becomes available or as technology advances.

Phases of Development:

1. Data Collection:

- Collect historical earthquake data, including information about location, date, time, magnitude, and depth.

- Acquire relevant geological, geographical, and environmental data that might influence seismic activity.

2. Data Preprocessing:

   - Clean and preprocess the collected data, handling missing values, outliers, and inconsistencies.

   - Perform feature engineering to create relevant features for prediction.

3. Exploratory Data Analysis (EDA):

   - Conduct EDA to gain insights into the data, identify patterns, and understand the relationships between variables.

4. Model Selection:

   - Choose appropriate machine learning algorithms for earthquake prediction. Options might include regression, time series analysis, or deep learning.

5. Model Training:

   - Train the selected model on the preprocessed data, using a portion of the historical data for training.

6. Evaluation and Validation:

   - Evaluate the model's performance using metrics like Mean Squared Error, Root Mean Squared Error, or others relevant to earthquake prediction.

   - Validate the model's accuracy and reliability through cross-validation techniques.

7. Real-time Data Integration:

   - Integrate the model with real-time data sources, such as seismic sensors and satellite data, for continuous monitoring.

8. Early Warning System:

- Develop an early warning system that provides alerts to stakeholders based on the model’s predictions.

9. Deployment and Accessibility:

- Deploy the earthquake prediction model in accessible platforms or applications to ensure that the information is readily available to the public and relevant authorities.

10. Continuous Improvement:

- Continuously monitor and update the model as new data becomes available and as the technology evolves.
- Consider feedback from experts and stakeholders for enhancements.

11. Disaster Preparedness:

- Work with local communities and disaster management organizations to implement disaster preparedness and response plans based on the predictions.

Here is simple program of the model

Install necessary package first

```
pip install scikit-learn
```

Here's a basic code example:

```
#Python

import pandas as pd
```

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score



# Load your preprocessed dataset

data = pd.read_csv('earthquake_data.csv')



# Define features and target variable

X = data[['depth_km', 'longitude', 'latitude']]

y = data['magnitude']



# Split the data into training and testing sets

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

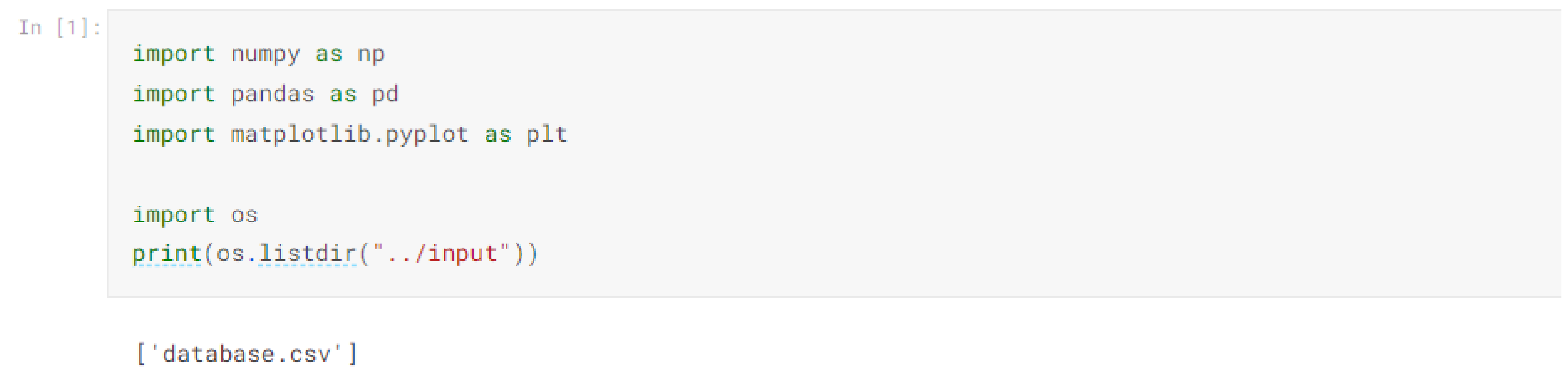
```
# Create a linear regression model

model = LinearRegression()



# Train the model

model.fit(X_train, y_train)



# Make predictions on the test set

y_pred = model.predict(X_test)



# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)
```

print(f"Mean Squared Error: {mse}")

print(f"R-squared (R2) Score: {r2}")

Let’s begin building our Earthquake prediction model using python

1.Import the necessary libraries required for buidling the model and data analysis of the earthquakes.

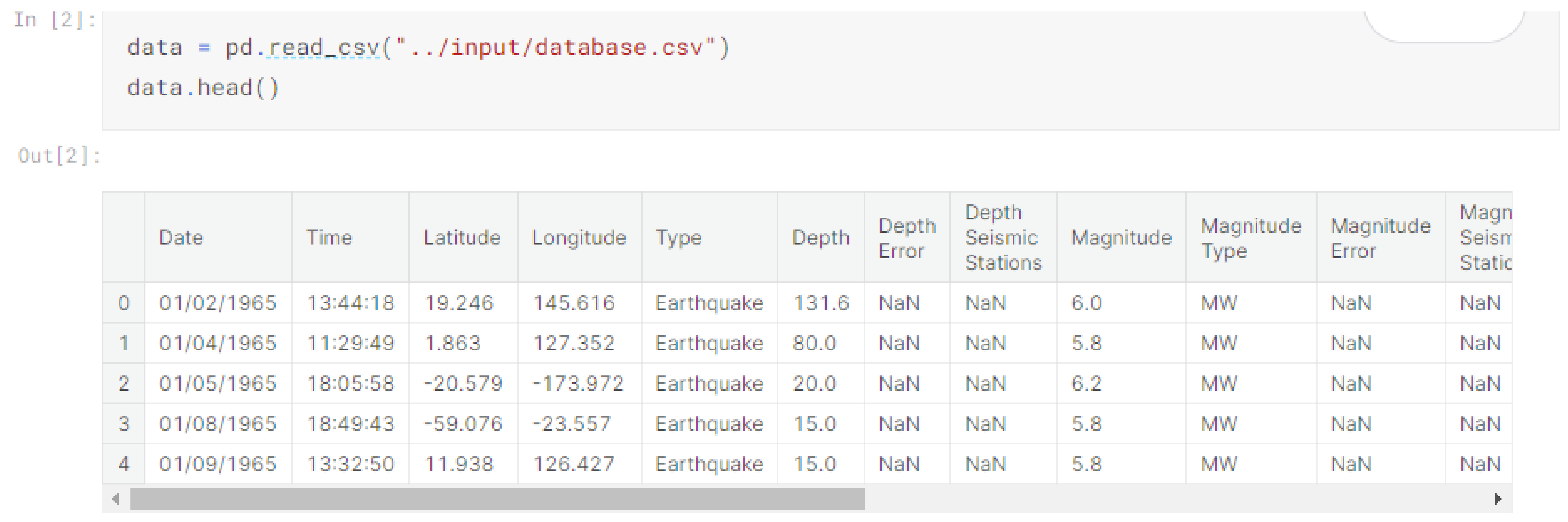


```
In [1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
print(os.listdir("../input"))
```

```
['database.csv']
```

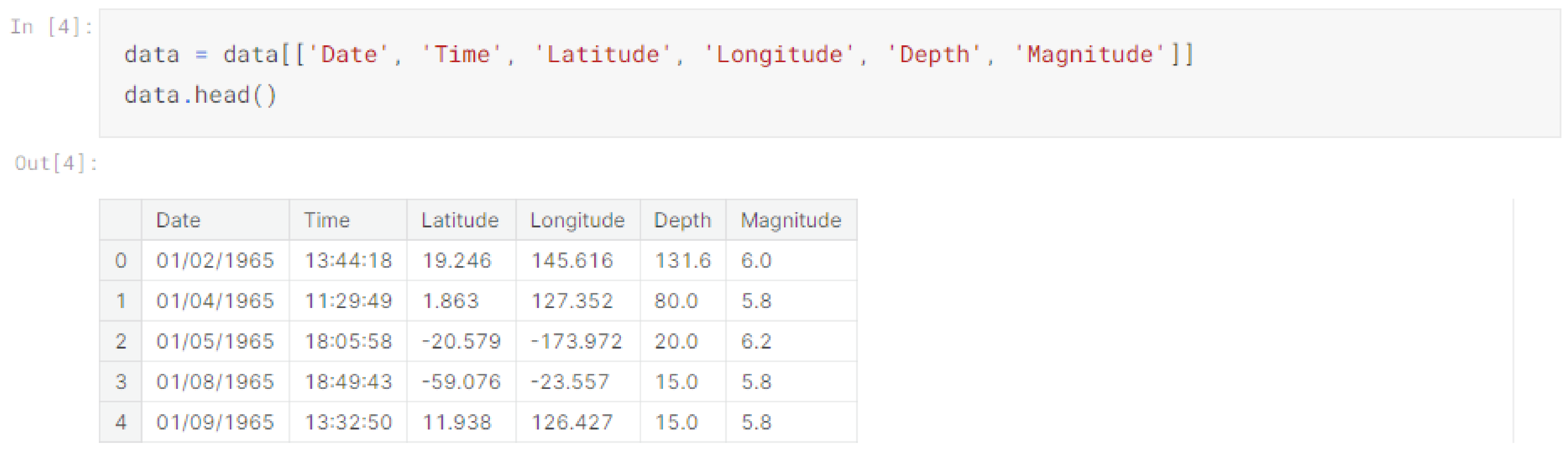
Data loading

Read the data from csv and also columns which are necessary for the model and the column which needs to be predicted.



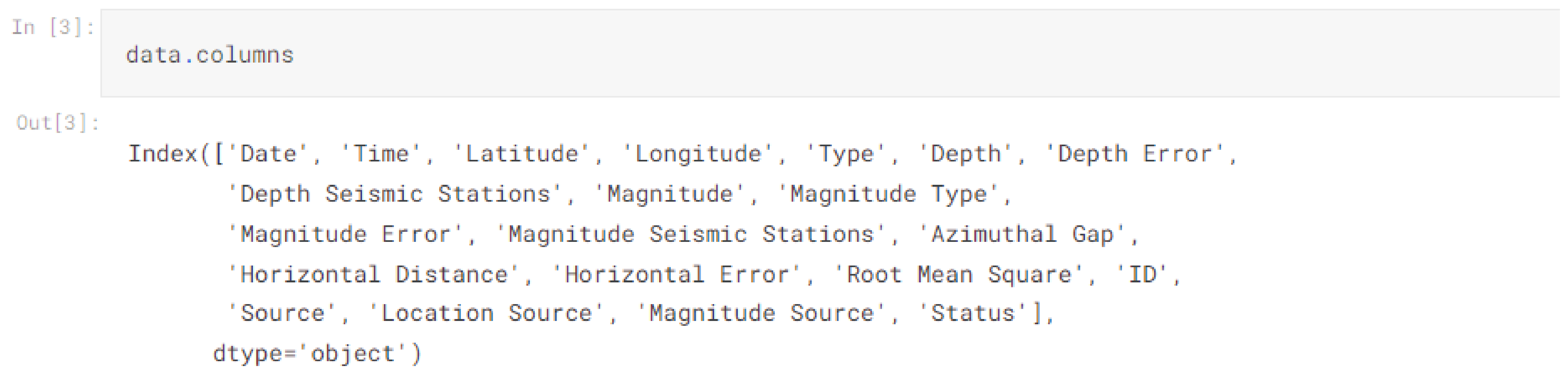
```
In [2]:
data = pd.read_csv("../input/database.csv")
data.head()

Out[2]:
```

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	Magnitude Error	Magn Seism Static
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	NaN	NaN	6.0	MW	NaN	NaN
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80.0	NaN	NaN	5.8	MW	NaN	NaN
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20.0	NaN	NaN	6.2	MW	NaN	NaN
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15.0	NaN	NaN	5.8	MW	NaN	NaN



```
In [4]:
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]
data.head()

Out[4]:
```

	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



```
In [3]:
data.columns

Out[3]:
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')
```

Data preprocessing

2.Set Timestamps: If your dataset provides date and time information, it’s essential to set timestamps to indicate when each earthquake event occurred. This can be useful for time-series analysis.

```
In [5]:
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')
```

```
In [6]:
timeStamp = pd.Series(timestamp)
data['Timestamp'] = timeStamp.values
```

```
In [7]:
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
final_data.head()
```

Out[7]:

	Latitude	Longitude	Depth	Magnitude	Timestamp
0	19.246	145.616	131.6	6.0	-1.57631e+08
1	1.863	127.352	80.0	5.8	-1.57466e+08
2	-20.579	-173.972	20.0	6.2	-1.57356e+08
3	-59.076	-23.557	15.0	5.8	-1.57094e+08
4	11.938	126.427	15.0	5.8	-1.57026e+08

Visualization

Here, all 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.

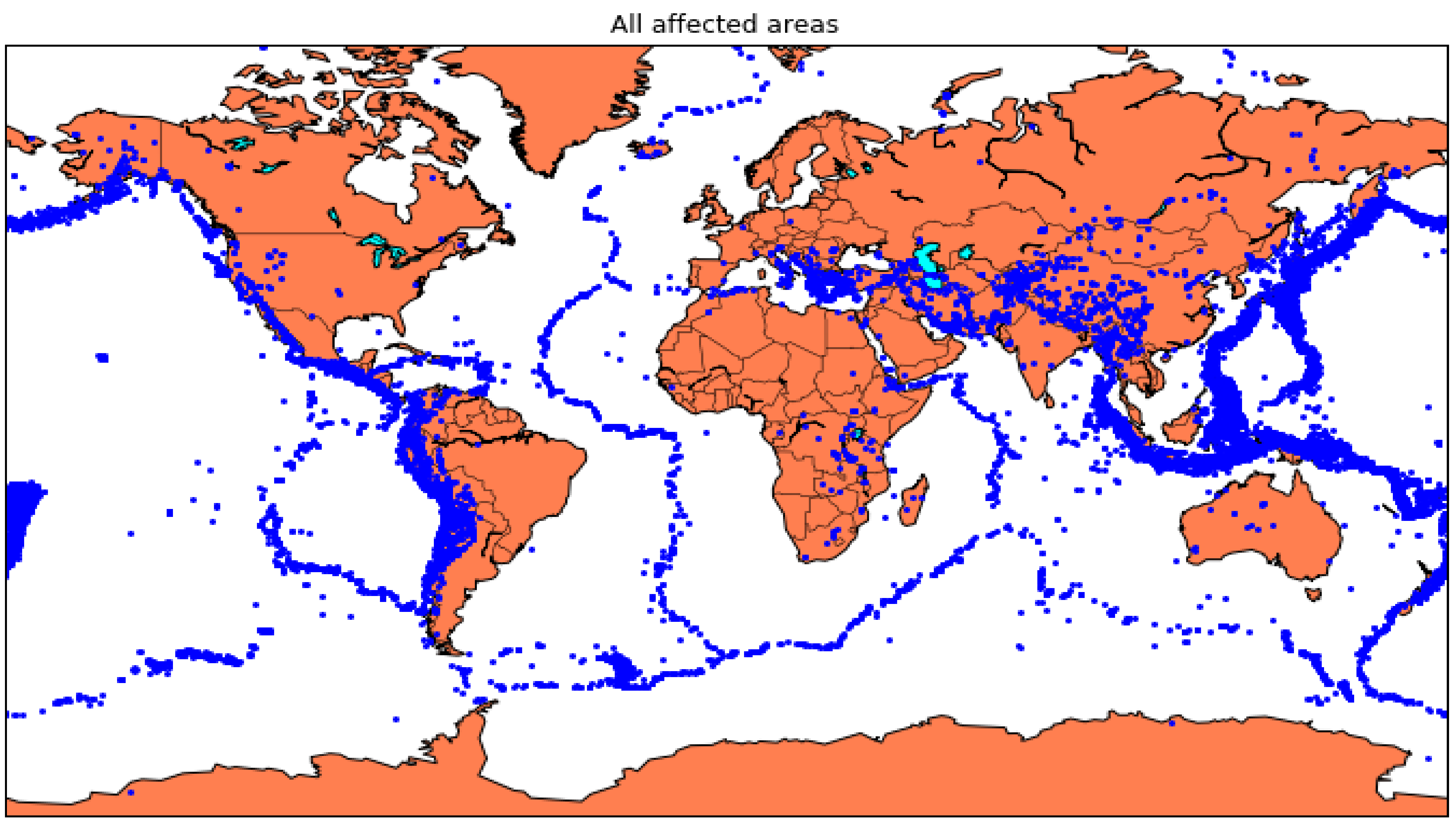
In [8]:

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,res
olution='c')

longitudes = data["Longitude"].tolist()
latitudes = data["Latitude"].tolist()
#m = Basemap(width=12000000,height=9000000,projection='lcc',
            #resolution=None,lat_1=80.,lat_2=55,lat_0=80,lon_0=-107.)
x,y = m(longitudes,latitudes)
```

In [9]:

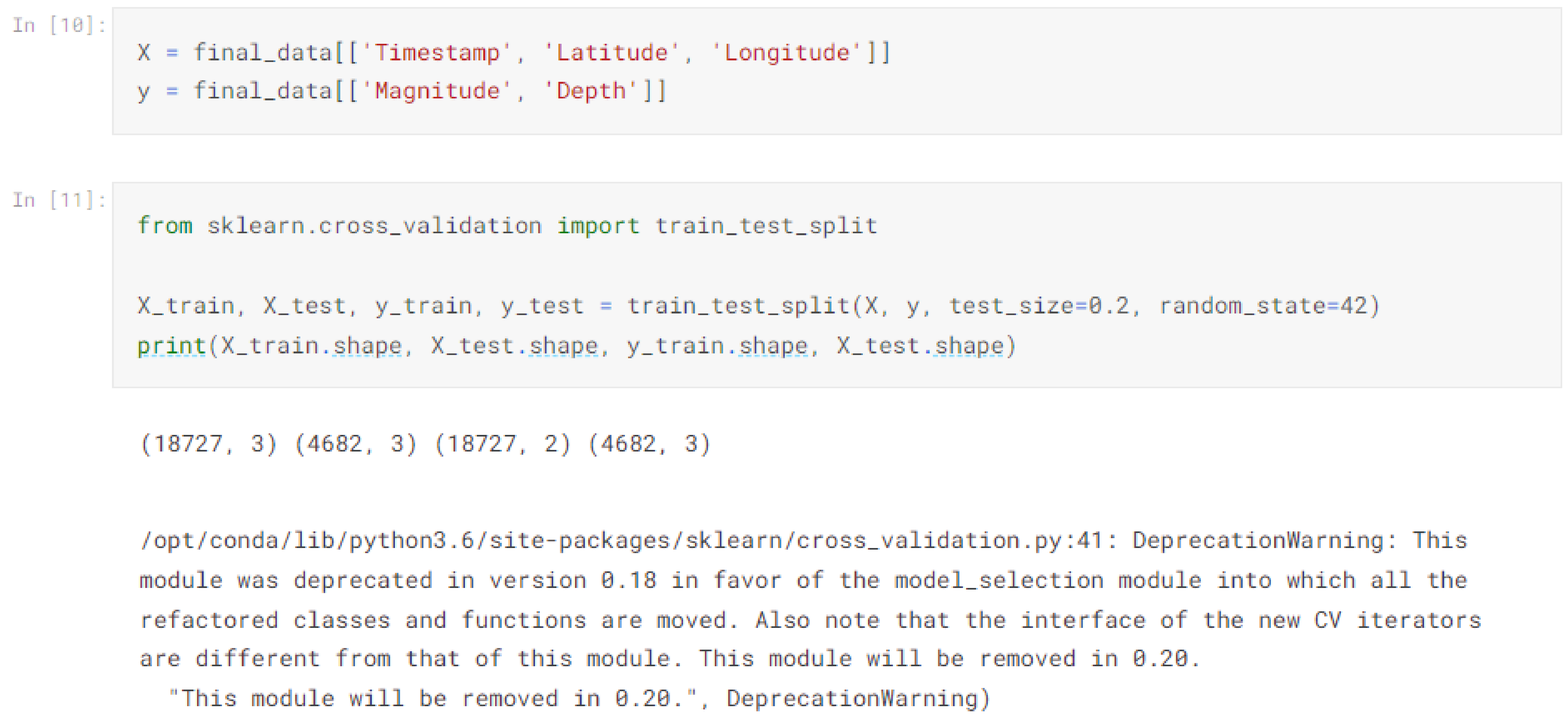
```
fig = plt.figure(figsize=(12,10))
plt.title("All affected areas")
m.plot(x, y, "o", markersize = 2, color = 'blue')
m.drawcoastlines()
m.fillcontinents(color='coral',lake_color='aqua')
m.drawmapboundary()
m.drawcountries()
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py:1704: MatplotlibDeprecat
ionWarning: The axesPatch function was deprecated in version 2.1. Use Axes.patch instead.
  limb = ax.axesPatch
/opt/conda/lib/python3.6/site-packages/mpl_toolkits/basemap/__init__.py:1707: MatplotlibDeprecat
ionWarning: The axesPatch function was deprecated in version 2.1. Use Axes.patch instead.
  if limb is not ax.axesPatch:
```



Splitting the Data:

Firstly, split the data into Xs and ys which are input to the model and output of the model respectively. Here, inputs are TImestamp, Latitude and Longitude and outputs are Magnitude and Depth. Split the Xs and ys into train and test with validation. Training dataset contains 80% and Test dataset contains 20%.

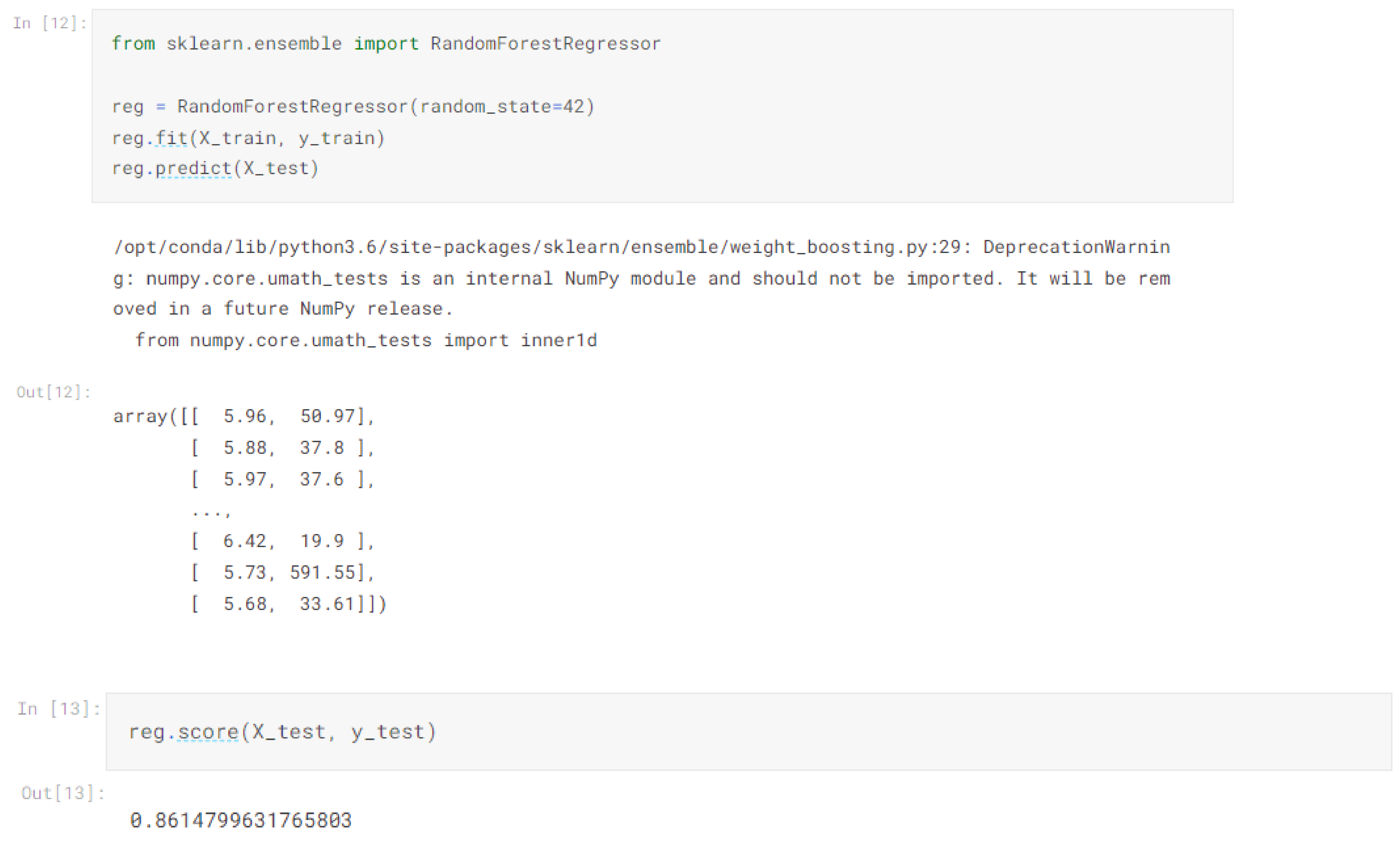


```
In [10]:
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
```

```
In [11]:
from sklearn.cross_validation 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)
```

```
(18727, 3) (4682, 3) (18727, 2) (4682, 3)

/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This
module was deprecated in version 0.18 in favor of the model_selection module into which all the
refactored classes and functions are moved. Also note that the interface of the new CV iterators
are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Here, we used the **RandomForestRegressor** model to predict the outputs, we see the strange prediction from this with score above 80% which can be assumed to be best fit but not due to its predicted values.



```
In [12]:
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarnin
g: numpy.core.umath_tests is an internal NumPy module and should not be imported. It will be rem
oved in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

```
Out[12]:
array([[  5.96,  50.97],
       [  5.88,  37.8 ],
       [  5.97,  37.6 ],
       ...,
       [  6.42,  19.9 ],
       [  5.73, 591.55],
       [  5.68,  33.61]])
```

```
In [13]:
reg.score(X_test, y_test)
```

```
Out[13]:
0.8614799631765803
```

In [14]:
```
from sklearn.model_selection import GridSearchCV

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

grid_obj = GridSearchCV(reg, parameters)
grid_fit = grid_obj.fit(X_train, y_train)
best_fit = grid_fit.best_estimator_
best_fit.predict(X_test)
```

Out[14]:
```
array([[  5.8888 ,  43.532  ],
       [  5.8232 ,  31.71656],
       [  6.0034 ,  39.3312 ],
       ...,
       [  6.3066 ,  23.9292 ],
       [  5.9138 , 592.151  ],
       [  5.7866 ,  38.9384 ]])
```

In [15]:
```
best_fit.score(X_test, y_test)
```

Out[15]:
```
0.8749008584467053
```