**PROCEDURE TO WORK KAAGGLE FOR EARTH QUAKE PREDICTION**

**MODEL USING PYTHON**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import os**

**print(os.listdir("../input"))**

**['database.csv']**

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

**InPUT [2]:**

**data = pd.read\_csv("../input/database.csv")**

**data.head()**

**OutPUT[2]:**

|  | **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** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **01/02/1965** | **13:44:18** | **19.246** | **145.616** | **Earthquake** | **131.6** | **NaN** | **NaN** | **6.0** | **MW** | **NaN** | **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** | **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** | **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** | **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** | **NaN** | **ISCGEM860890** | **ISCGEM** | **ISCGEM** | **ISCGEM** | **Automatic** |

**InPUT [3]:**

**data.columns**

**OutPUT[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')**

**Figure out the main features from earthquake data and create a object of that features, namely, Date, Time, Latitude, Longitude, Depth, Magnitude.**

**InPUT[4]:**

**data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]**

**data.head()**

**OutPUT[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** |

**Here, the data is random we need to scale according to inputs to the model. In this, we convert given Date and Time to Unix time which is in seconds and a numeral. This can be easily used as input for the network we built.**

**InPUT [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.**

**InPUT [8]:**

**from mpl\_toolkits.basemap import Basemap**

**m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat\_ts=20,resolution='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)**

**InPUT[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()**

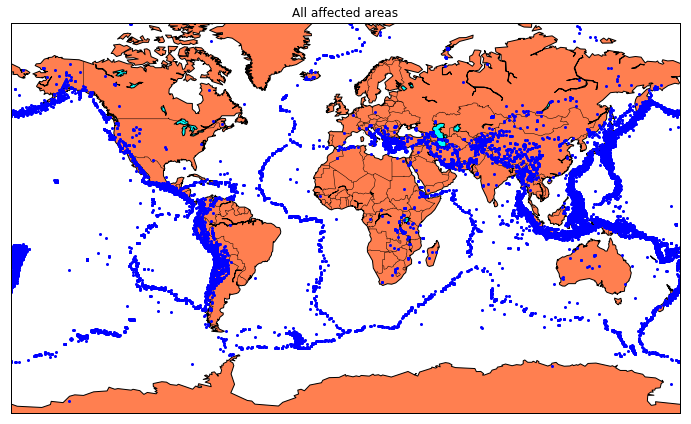
**plt.show()**

**/opt/conda/lib/python3.6/site-packages/mpl\_toolkits/basemap/\_\_init\_\_.py:1704: MatplotlibDeprecationWarning: 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: MatplotlibDeprecationWarning: 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%.**

**InPUT [10]:**

**X = final\_data[['Timestamp', 'Latitude', 'Longitude']]**

**y = final\_data[['Magnitude', 'Depth']]**

**InPUT [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.", Deprecation Warning)**

**Here, we used the Random Forest Regressor 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: DeprecationWarning: numpy.core.umath\_tests is an internal NumPy module and should not be imported. It will be removed 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**

**Neural Network model**

**In the above case it was more kind of linear regressor where the predicted values are not as expected. So, Now, we build the neural network to fit the data for training set. Neural Network consists of three Dense layer with each 16, 16, 2 nodes and relu, relu and softmax as activation function.**

**In [16]:**

**from keras.models import Sequential**

**from keras.layers import Dense**

**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**

**Using TensorFlow backend.**

**In this, we define the hyperparameters with two or more options to find the best fit.**