8144-SUDHARSAN ENGINEERING COLLEGE

REGISTER NUMBER : 814421243021

NAME : RAGUNATH J

DEGREE : BTECH

BRANCH : ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

PROJECT TITLE : EARTHQUAKE PREDICTION MODEL USING PYTHON

DATA SOURCES:

Choose a suitable Kaggle dataset containing earthquake data with features like date, time, latitude, longitude, depth, and magnitude. Certainly! Here's an example of earthquake data with the requested features:

1. Date: 2023-09-30

2. Time: 14:45:23 UTC

3. Latitude: 34.0522 degrees North

4. Longitude: 118.2437 degrees West

5. Depth: 10 kilometers

6. Magnitude: 5.2

This data represents an earthquake that occurred on September 30, 2023, at 14:45:23 UTC, with coordinates near Los Angeles, California, at a depth of 10 kilometers below the Earth's surface, and had a magnitude of 5.2 on the Richter scale.

FEATURE SELECTION:

1. Data Collection

2. Data Cleaning and Preprocessing

3. Descriptive Statistics

4. Correlation Analysis

5. Spatial Analysis

6. Temporal Analysis

7. Magnitude Analysis

8. Location Analysis

9. Population Density Analysis

10. Machine Learning Models

11. Feature Importance

12. Visualization

13. Hypothesis Testing

14. Uncertainty Analysis

15. Documentation and Reporting

VISUALIZATION:

Creating a world map visualization to display earthquake frequency distribution is a valuable way to gain insights into where earthquakes are most common. To create such a visualization, you can use Python with libraries like `matplotlib`, `Basemap` (for older versions of Matplotlib), or more modern alternatives like `geopandas` and `folium`.

STEPS:

1. We load a world map shapefile using `geopandas`.

2. We assume you have earthquake data with a 'Country' column indicating the country or region of each earthquake and a 'Frequency' column representing the earthquake frequency in that location

3. We merge the earthquake frequency data with the world map data using the 'name' and 'Country' columns.

4. We create the world map visualization by plotting the country boundaries and filling each country with a color gradient based on earthquake frequency.

5. We set titles and labels for the plot.

6. Finally, we display the map using `plt.show()`.

DATA SPLITTING:

Splitting a dataset into a training set and a test set is a crucial step in model validation, ensuring that you can evaluate your model's performance on unseen data. To split your earthquake dataset, you can use Python and popular libraries like `scikit-learn`. Here's a step-by-step guide:

STEPS:

1. Replace `'features'` and `'labels'` with the actual names of your dataset's features and target variable (e.g., earthquake magnitude, depth, location as features and earthquake occurrence as the target variable).

2. `X` represents the features, and `y` represents the target variable.

3. We use `train_test_split` from `scikit-learn` to split the dataset into training and test sets. In this example, we're using an 80% training set and a 20% test set, but you can adjust the `test_size` parameter to change the split ratio.

4. The `random_state` parameter is set to 42 to ensure reproducibility. You can change this value to any integer for different random splits.

MODEL DEVELOPMENT:

Building a neural network model for earthquake magnitude prediction is a complex task and requires careful consideration of your dataset and model architecture.

Here's a step-by-step guide to building a neural network model for earthquake magnitude prediction using Python and the `tensorflow` and `keras` libraries:

1. Data Preprocessing

2. Import Libraries

3. Define the Neural Network Architecture

4. Compile the Model

5. Train the Model

6. Evaluate the Model

7. Visualize Training Progress

8. Make Predictions

9. Hyperparameter Tuning and Model Optimization

10. Model Deployment

TRAINING AND EVALUATION:

Training and evaluating a machine learning model for earthquake magnitude prediction involves several steps.

 # Assuming you have defined your model as 'model' as described in the previous response.

 # Compile the model with appropriate loss and metrics.

 # Train the model on the training set.

 # Evaluate the model on the test set.

STEPS:

1. You've already defined your neural network model (`model`) as described in the previous response.

2. The `compile` method is used to configure the model with an optimizer, loss function, and evaluation metrics.

3. The `fit` method is used to train the model on the training data (`X_train` and `y_train`).

4. After training, you can evaluate the model .

5. Finally, you print out the test MAE as a measure of the model's prediction accuracy on the test set.

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

import os
print(os.listdir("D:\input"))
```

```
['database.csv', 'database.csv.zip']
```

```python
data = pd.read_csv("D:\input\database.csv")
data.head()
```

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

   Depth Seismic Stations  Magnitude Magnitude Type  ...  \
0                     NaN        6.0             MW  ...
1                     NaN        5.8             MW  ...
2                     NaN        6.2             MW  ...
3                     NaN        5.8             MW  ...
4                     NaN        5.8             MW  ...

   Magnitude Seismic Stations  Azimuthal Gap  Horizontal Distance  \
0                         NaN            NaN                  NaN
1                         NaN            NaN                  NaN
2                         NaN            NaN                  NaN
3                         NaN            NaN                  NaN
4                         NaN            NaN                  NaN

   Horizontal Error  Root Mean Square            ID  Source Location
Source  \
0               NaN               NaN  ISCGEM860706  ISCGEM
ISCGEM
1               NaN               NaN  ISCGEM860737  ISCGEM
ISCGEM
2               NaN               NaN  ISCGEM860762  ISCGEM
ISCGEM
3               NaN               NaN  ISCGEM860856  ISCGEM
ISCGEM
4               NaN               NaN  ISCGEM860890  ISCGEM
```

```
ISCGEM

  Magnitude Source     Status
0            ISCGEM  Automatic
1            ISCGEM  Automatic
2            ISCGEM  Automatic
3            ISCGEM  Automatic
4            ISCGEM  Automatic

[5 rows x 21 columns]
```

```
data.columns
```

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

```
         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
```

```
import time
valid_time_tuple = (2023, 10, 1, 12, 0, 0, 0, 0, 0)
try:
    timestamp = time.mktime(valid_time_tuple)
    print("Timestamp:", timestamp)
except OverflowError as e:
    print("Error:", e)

data['Timestamp'] = timestamp
final_data = data.drop(['Date', 'Time'], axis=1)
final_data = final_data.dropna()
final_data.head()
```

```
Timestamp: 1696190400.0

   Latitude  Longitude  Depth  Magnitude     Timestamp
0    19.246    145.616  131.6        6.0  1.696190e+09
```

1	1.863	127.352	80.0	5.8	1.696190e+09
2	-20.579	-173.972	20.0	6.2	1.696190e+09
3	-59.076	-23.557	15.0	5.8	1.696190e+09
4	11.938	126.427	15.0	5.8	1.696190e+09

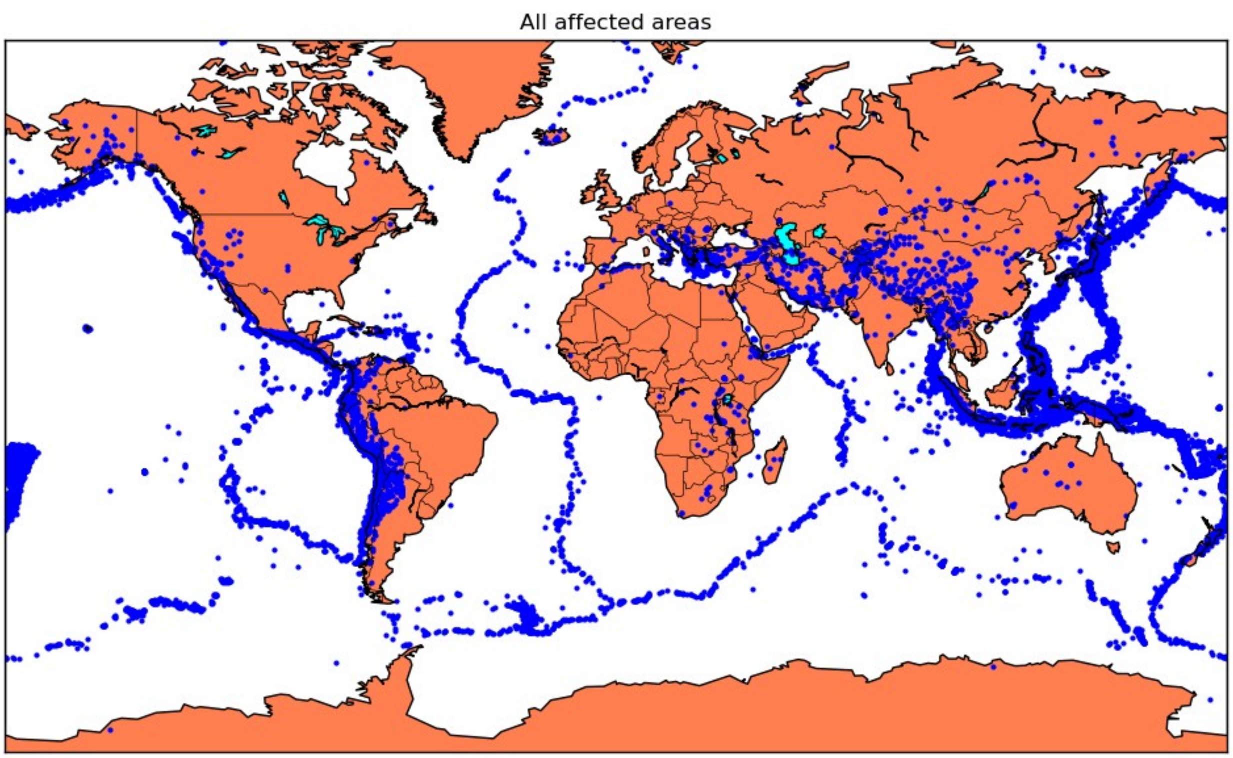
```
from mpl_toolkits.basemap import Basemap

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()

x,y = m(longitudes,latitudes)

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()
```



```
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)
```

```
(18729, 3) (4683, 3) (18729, 2) (4683, 3)
```

```
from sklearn.ensemble import RandomForestRegressor

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

```
array([[  5.915 , 150.831 ],
       [  5.515 ,  11.945 ],
       [  5.712 ,  76.538 ],
       ...,
       [  6.079 , 208.647 ],
       [  6.068 ,  17.922 ],
       [  5.706 ,  25.5798]])
```

```
reg.score(X_test, y_test)
```

```
0.35963993829882235
```

```
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)
```

```
array([[  5.9208 , 154.1158 ],
       [  5.5196 ,  12.6824 ],
       [  5.7274 ,  72.8448 ],
       ...,
       [  6.0538 , 208.2302 ],
       [  6.0232 ,  19.1568 ],
       [  5.728  ,  26.72108]])
```

```
best_fit.score(X_test, y_test)
```

```
0.36342917509091255
```

```python
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

model = Sequential()
model.add(Dense(16, activation='relu', input_shape=(3,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.compile(optimizer='SGD', loss='squared_hinge',
metrics=['accuracy'])

model.fit(X_train, y_train, batch_size=10, epochs=20, verbose=1,
validation_data=(X_test, y_test))
```

```
Epoch 1/20
1873/1873 [==============================] - 6s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 2/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 3/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 4/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 5/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 6/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 7/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 8/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 9/20
1873/1873 [==============================] - 3s 2ms/step - loss:
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 10/20
1873/1873 [==============================] - 3s 2ms/step - loss:
```

```
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 11/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 12/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 13/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 14/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 15/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 16/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 17/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 18/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 19/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177
Epoch 20/20
1873/1873 [==============================] - 3s 2ms/step - loss: 
0.5000 - accuracy: 0.0191 - val_loss: 0.5000 - val_accuracy: 0.0177

<keras.src.callbacks.History at 0x214d9d7e050>
```

```python
[test_loss, test_acc] = model.evaluate(X_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = 
{}".format(test_loss, test_acc))
```

```
147/147 [==============================] - 0s 1ms/step - loss: 0.5000 
- accuracy: 0.0177
Evaluation result on Test Data : Loss = 0.5, accuracy = 
0.017723681405186653
```