EARTHQUAKE PREVENTION USING PYTHON IN AI

PHASE –IV

TRAINING AND TESTING THE EARTHQUAKE PREDICTION USING PYTHON

ABSTRACT :-

- this paper presents a case study on how to use python programming language to solve earthquake prediction problems. The study uses the numpy, pandas, and keras library to implement a deep learning model for predicting the b-value as an earthquake precursor. The programming sequence includes data preparation, model determination, model compilation, model adjustment,model evaluation, and prediction system creation. The results show that the deep learning training process for the prediction system of earthquake precursor has a high percentage of successful predictions. The testing process for the prediction system as an earthquake precursor also shows promising results. Other studies have used machine learning models to predict earthquakes, such as the SVM and naive bayes classifiers, and to analyze earthquake patterns and characteristics. The earthquake dataset can be used to build machine learning models to predict earthquakes or to better understand earthquake patterns and characteristics. Based on the results and datasets, further analysis can be performed to identify the main factors that contribute to the occurrence of earthquakes.

INTRODUCTION :-

To introduce training and testing for earthquake prediction using Python, there are several resources available that use machine learning and deep learning techniques. It provide a step-by-step guide on how to create an earthquake prediction model using Python programming language. Here are the general steps to follow:
Import necessary Python libraries such as NumPy, Pandas, and Keras. Visualize the data on a world map to show where earthquake frequency is higher. Split the dataset into Xs and ys, which will respectively be entered into the model as inputs to receive the output from the model. Use machine learning algorithms such as Naive Bayes, Random Forest, Support Vector Classifier, and Gradient Boosting Algorithm to predict the magnitude and probability of earthquakes occurring in a particular region. Divide the data into training and testing sets, train the model on the training data, and evaluate its performance on the test data using accuracy score, confusion matrix, and classification report.

PYTHON CODE FOR EARTHQUAKE PREDICTION USING PYTHON[TRAINING AND TESTING]:-

```
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.

```
data = pd.read_csv("../input/database.csv")data.head()
data = data[['Date', 'Time', 'Latitude', 'Longitude', 'Depth', 'Magnitude']]data.head()
import datetimeimport timetimestamp = []for d, t in
(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')

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

VISUALIZATION:

```
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)
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()
```

SPLITTING THE DATA:

```
X = final_data[['Timestamp', 'Latitude', 'Longitude']]
y = final_data[['Magnitude', 'Depth']]
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)
from sklearn.ensemble import RandomForestRegressor
reg = RandomForestRegressor(random_state=42)
reg.fit(X_train, y_train)
reg.predict(X_test)
reg.score(X_test, y_test)
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)
best_fit.score(X_test, y_test)
```

NEURAL NETWORK MODEL:

```
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
from keras.wrappers.scikit_learn import KerasClassifier
model = KerasClassifier(build_fn=create_model, verbose=0)
# neurons = [16, 64, 128, 256]
neurons = [16]
# batch_size = [10, 20, 50, 100]
batch_size = [10]
epochs = [10]
# activation = ['relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear', 'exponential']
activation = ['sigmoid', 'relu']
# optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
optimizer = ['SGD', 'Adadelta']
loss = ['squared_hing
```

```
param_grid = dict(neurons=neurons, batch_size=batch_size, epochs=epochs, activation=activation,
optimizer=optimizer, loss=loss)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

The best fit parameters are used for same model to compute the score with training data and testing data.

```
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))

Train on 18727 samples, validate on 4682 samples
Epoch 1/20
18727/18727 [==============================] - 4s 233us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 2/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 3/20
18727/18727 [==============================] - 4s 228us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 4/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 5/20
18727/18727 [==============================] - 5s 262us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 6/20
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 7/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 8/20
18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 9/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 10/20
18727/18727 [==============================] - 4s 224us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
```

```
18727/18727 [==============================] - 4s 223us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 16/20
18727/18727 [==============================] - 4s 222us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 17/20
18727/18727 [==============================] - 4s 225us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 18/20
18727/18727 [==============================] - 4s 219us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 19/20
18727/18727 [==============================] - 4s 220us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242
Epoch 20/20
18727/18727 [==============================] - 5s 258us/step - loss: 0.5038 - acc: 0.9182 - val_loss: 0.5038 - val_acc: 0.9242


<keras.callbacks.History at 0x78dfa2107ef0>
```

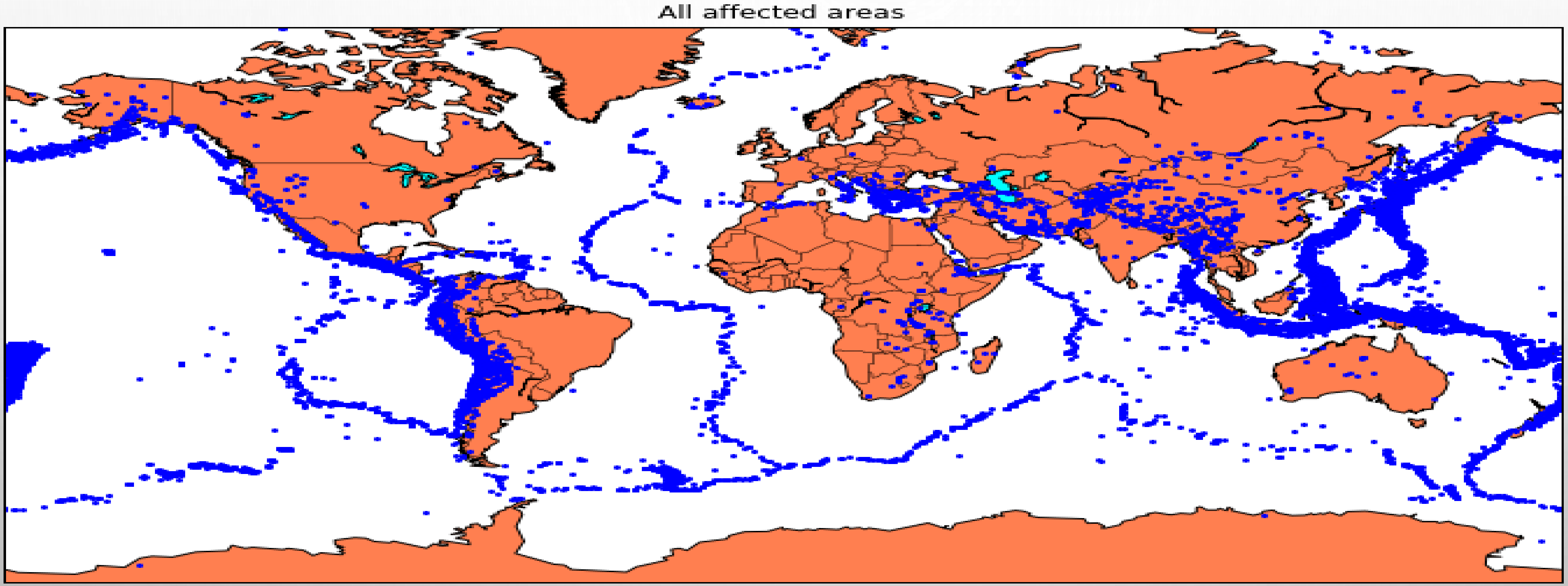
```
[test_loss, test_acc] = model.evaluate(X_test, y_test)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

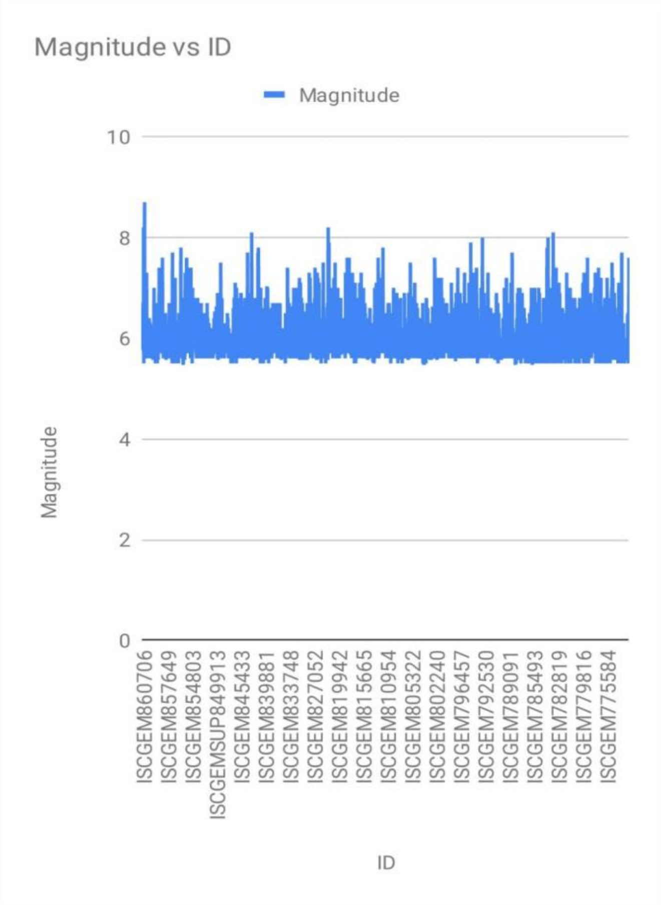
```
4682/4682 [==============================] - 0s 29us/step
Evaluation result on Test Data : Loss = 0.5038455790406056, accuracy = 0.9241777017858995
```

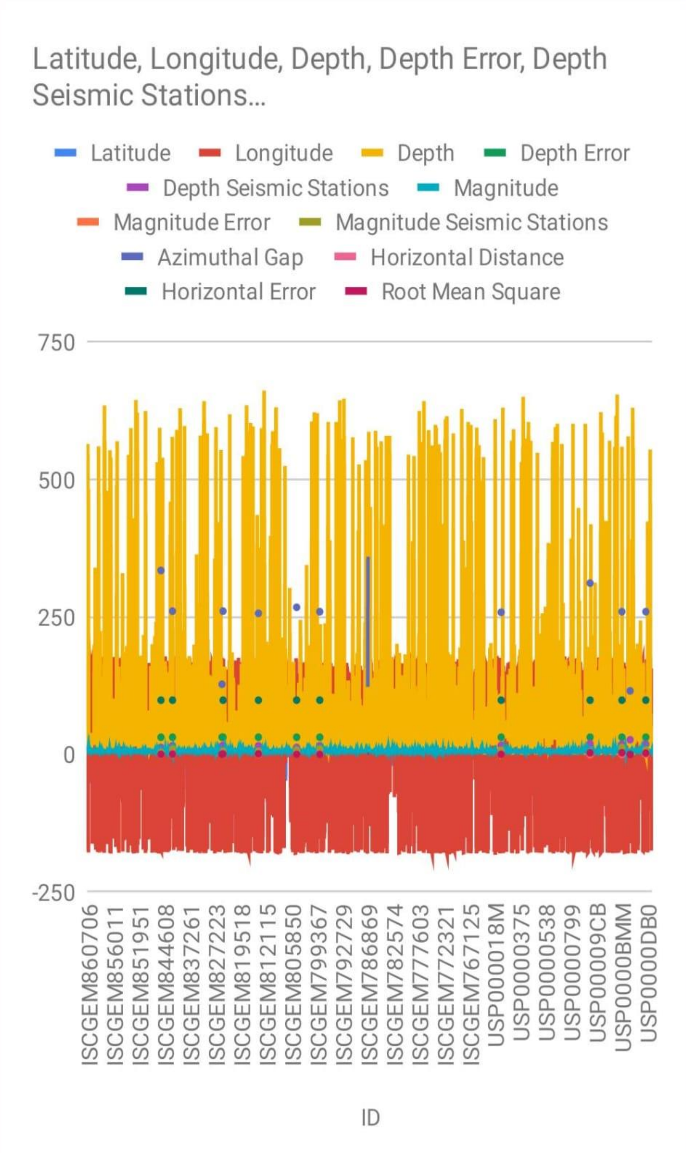
We see that the above model performs better but it also has lot of noise (loss) which can be neglected for prediction and use it for furthur prediction.
The above model is saved for furthur prediction

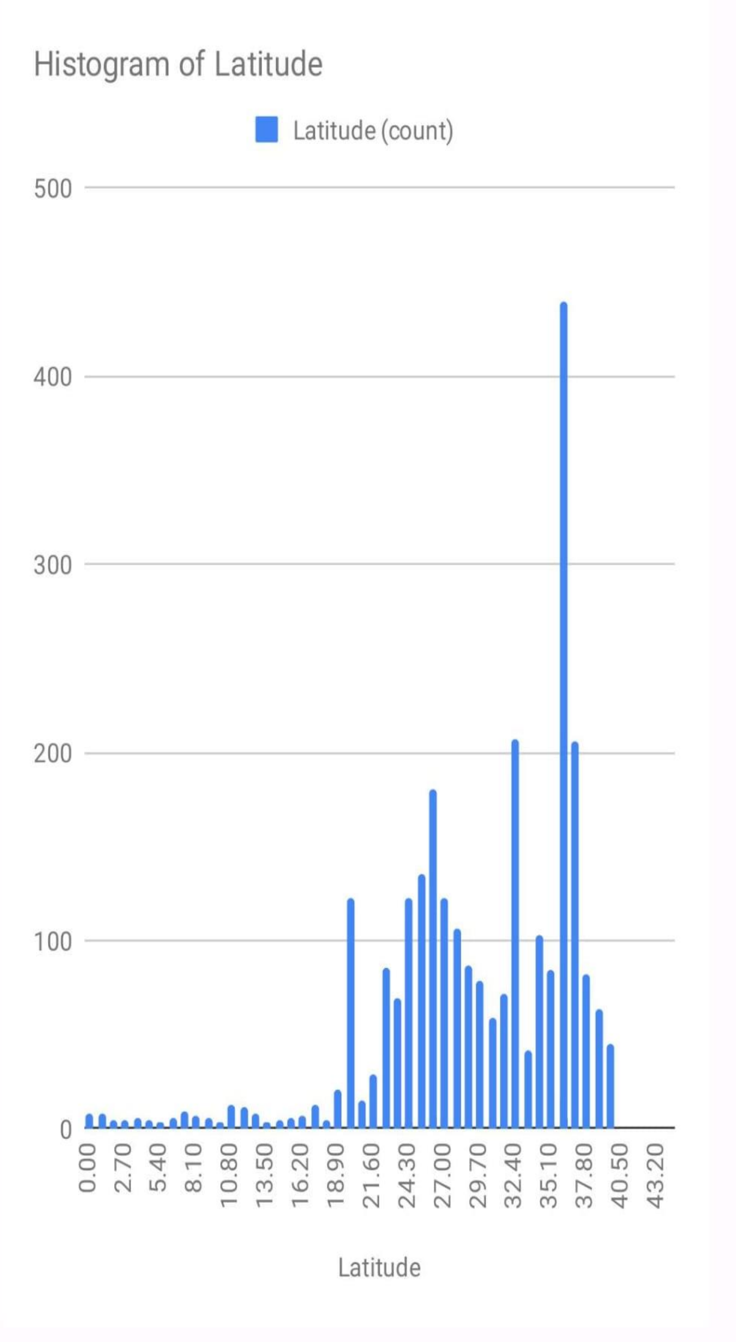
```
model.save
('earthquake.h5')
```

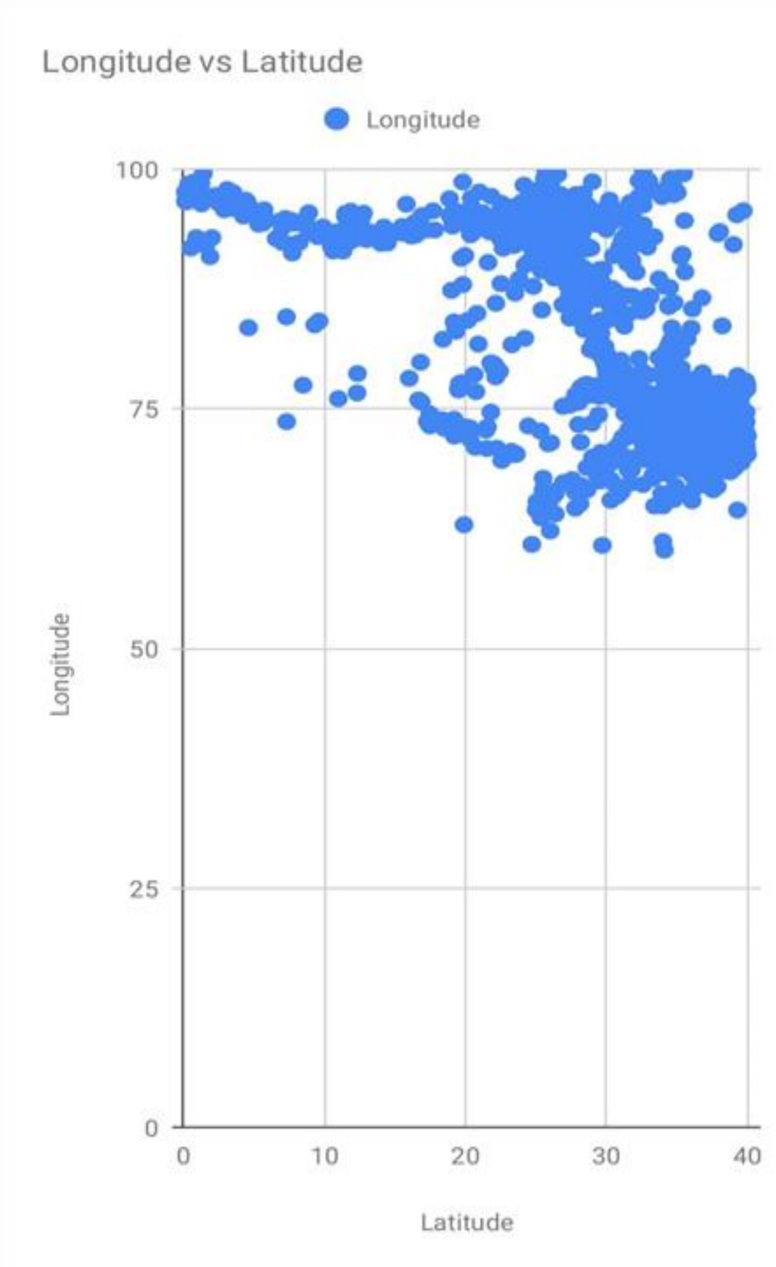
OUTPUT[PIE CHART]:-

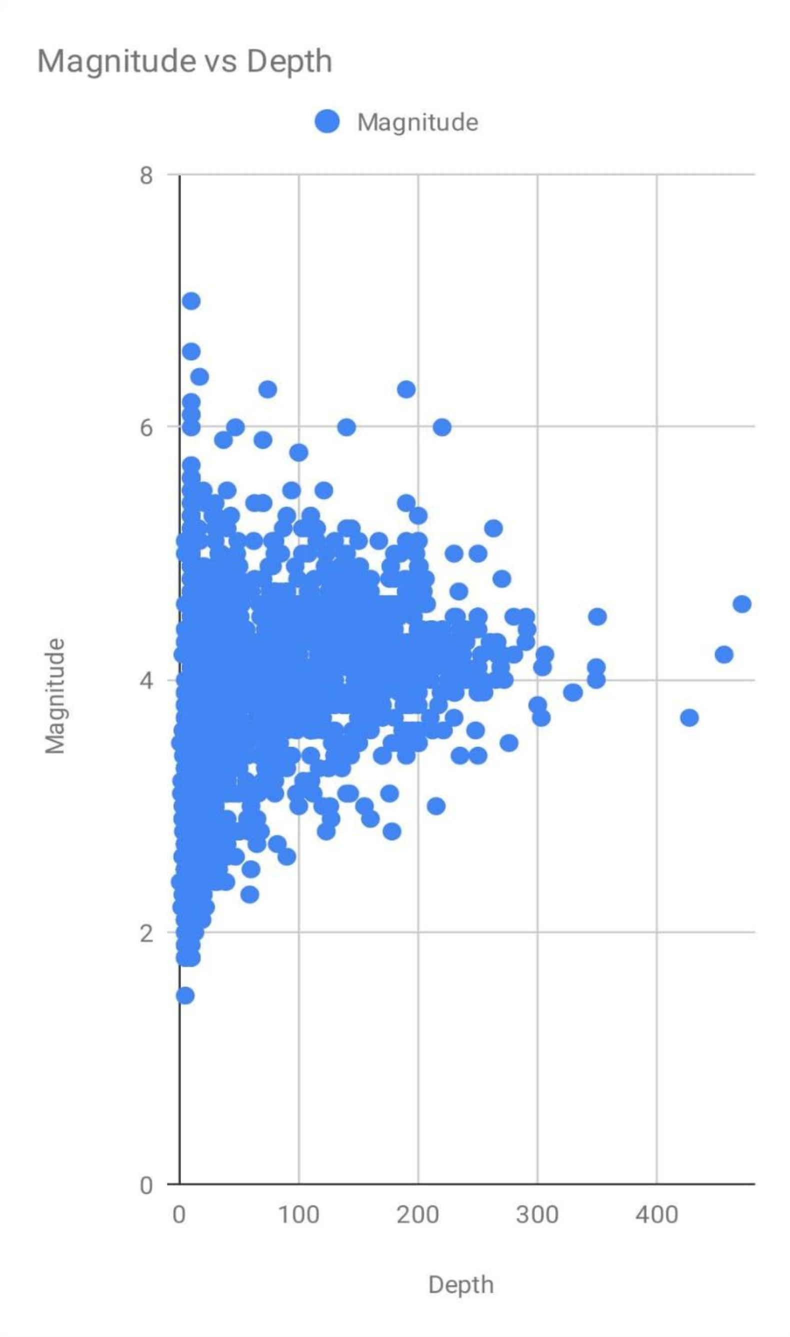


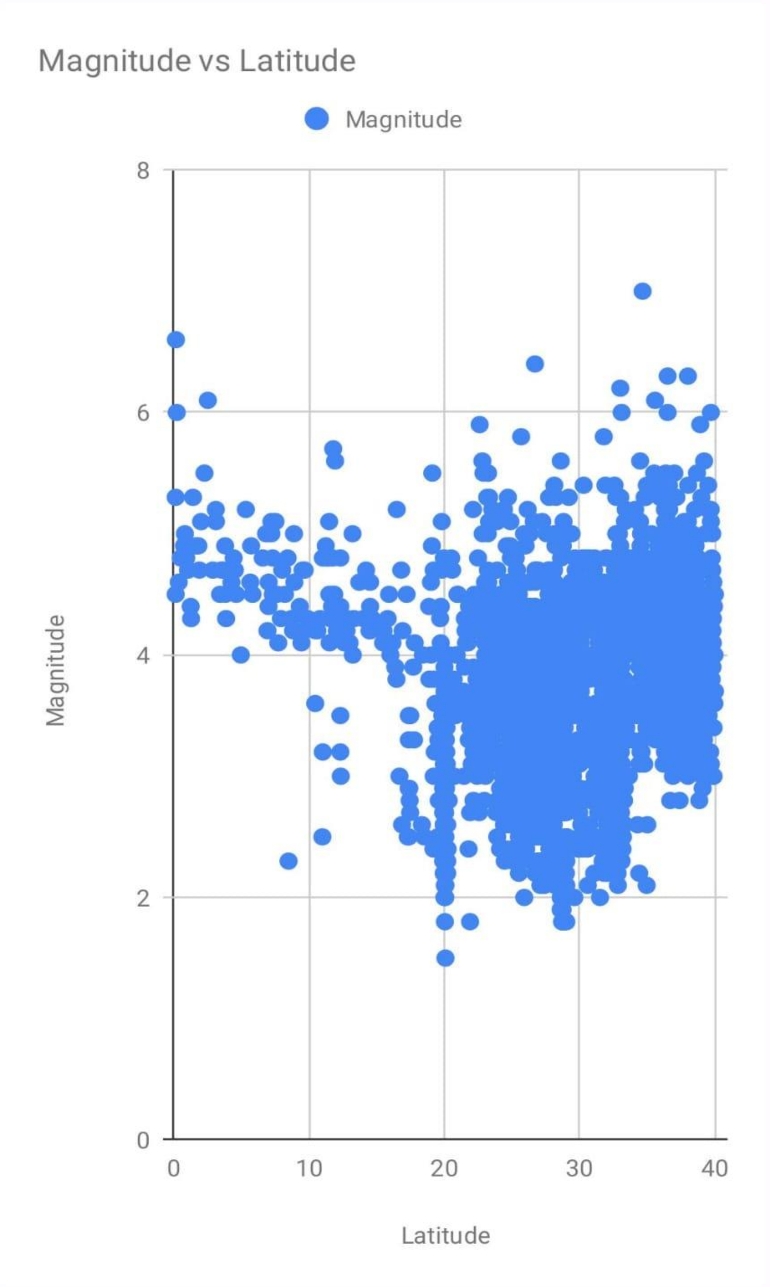

Magnitude vs ID
Magnitude
10
8
6
4
2
0
Magnitude
ISCGEM860706
ISCGEM857649
ISCGEM854803
ISCGEMSUP849913
ISCGEM845433
ISCGEM839881
ISCGEM833748
ISCGEM827052
ISCGEM819942
ISCGEM815665
ISCGEM810954
ISCGEM805322
ISCGEM802240
ISCGEM796457
ISCGEM792530
ISCGEM789091
ISCGEM785493
ISCGEM782819
ISCGEM779816
ISCGEM775584
ID


Latitude, Longitude, Depth, Depth Error, Depth Seismic Stations...
Latitude
Longitude
Depth
Depth Error
Depth Seismic Stations
Magnitude
Magnitude Error
Magnitude Seismic Stations
Azimuthal Gap
Horizontal Distance
Horizontal Error
Root Mean Square
750
500
250
0
-250
ISCGEM860706
ISCGEM856011
ISCGEM851951
ISCGEM844608
ISCGEM837261
ISCGEM827223
ISCGEM819518
ISCGEM812115
ISCGEM805850
ISCGEM799367
ISCGEM792729
ISCGEM786869
ISCGEM782574
ISCGEM777603
ISCGEM772321
ISCGEM767125
USP000018M
USP0000375
USP0000538
USP0000799
USP00009CB
USP0000BMM
USP0000DB0
ID


Histogram of Latitude
Latitude (count)
500
400
300
200
100
0
0.00
2.70
5.40
8.10
10.80
13.50
16.20
18.90
21.60
24.30
27.00
29.70
32.40
35.10
37.80
40.50
43.20
Latitude


Longitude vs Latitude


Magnitude vs Depth


Magnitude vs Latitude

TRAINING A MACHINE LEARNING MODEL FOR EARTHQUAKE PREDICTION INVOLVES SEVERAL KEY STEPS:

1. Feature Engineering: Identifying significant features or variables that are likely to have an impact on earthquake prediction. This may include the velocity of seismic waves, the depth of seismic activity, and the location of faults.
2. Model Selection: Selecting appropriate machine learning models for prediction. Common algorithms used in earthquake prediction include Support Vector Machines (SVM), Neural Networks, Random Forests, and Gradient Boosting.
3. Data Preparation: Collecting and cleaning data, and splitting it into training and testing sets. Hyperparameter tuning and cross-validation can help in selecting the best model.
4. Training: Training the selected model on the training data. During training, the input and target data are used to construct the machine learning model.
5. Evaluation: Evaluating the model using metrics like accuracy, precision, recall, and F1 score to assess its effectiveness in predicting earthquakes.
6. Prediction: Using the trained model to predict the likelihood of an earthquake in the given area. The output can be a binary classification (earthquake/no earthquake) or a continuous value representing the probability.

TRAINING MACHINE LEARNING MODELS CHALLENGING TASK DUE TO SEVERAL REASONS.

- Here are some of the challenges:- Lack of data: Earthquakes are rare events, and collecting sufficient data to train machine learning models can be challenging. Moreover, the data may be imbalanced, making it difficult to train models that can accurately predict earthquakes.
- Earthquakes are complex phenomena that involve multiple factors, such as the location, depth, and magnitude of the earthquake. Identifying the most significant features and variables that impact earthquake prediction is not straightforward.- Limited understanding of the underlying physics: The underlying physics of earthquakes is not fully understood, and there may be many unknown factors that contribute to earthquake occurrence. This makes it challenging to develop accurate models that can predict earthquakes with high precision.
- Overfitting occurs when a model is too complex and fits the training data too closely, resulting in poor generalization performance on new data. This is a common problem in machine learning, and it can be particularly challenging in earthquake prediction due to the limited amount of data available.
- Selecting the appropriate machine learning model for earthquake prediction is not straightforward. There are many algorithms to choose from, and each has its strengths and weaknesses.

HANDLING IMBALANCED DATASETS IS CRUCIAL IN EARTHQUAKE PREDICTION:

Here are some techniques to handle imbalanced datasets in earthquake prediction:

- Random Undersampling and Oversampling: Randomly removing some negative examples (undersampling) or duplicating positive examples (oversampling) can balance the dataset. However, oversampling can lead to overfitting, and undersampling can result in the loss of valuable information.
- Synthetic Minority Oversampling Technique (SMOTE): SMOTE generates synthetic positive examples by interpolating between existing positive examples. This technique can balance the dataset and reduce overfitting.
- Adaptive Synthetic Sampling (ADASYN): ADASYN generates synthetic positive examples in regions where the density of positive examples is low. This technique can handle datasets with severe class imbalance.
- Cost-Sensitive Learning: Assigning different costs to misclassification of positive and negative examples can help in handling imbalanced datasets. This technique can be used in conjunction with other techniques like SMOTE and ADASYN.
- Ensemble Methods: Combining multiple models trained on different subsets of the data can improve the performance of the model on imbalanced datasets. Ensemble methods like Bagging, Boosting, and Stacking can be used for this purpose.

CONCLUSION:-

- For training and testing of earthquake prediction using pythonBased on the search results, it is possible to use Python programming language and machine learning models to predict earthquakes. The following steps can be taken to train and test an earthquake prediction system using Python. Data preparation:Collect and preprocess earthquake data from reliable sources such as the International Seismological Center (ISC) Sumatra-Andaman region.Choose a machine learning model such as Naive Bayes, Random Forest, or Deep Learning using libraries such as Keras].Compile the chosen model with appropriate parameters and hyperparameters.4. Model adjustment: Train the model with the prepared data and adjust the model parameters as needed. Evaluate the performance of the model using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) .Use the trained model to create an earthquake prediction system that can predict the magnitude and probability of an earthquake occurring in a particular region. Test the prediction system using test data and evaluate its performance using metrics such as accuracy score, confusion matrix, and classification report.