EARTHQUAKE PREDICTION MODEL USING PYTHON

AI_PHASE1

NAME: SUBASH RAJ V

REG.NO: 610821205052

Phase 1: Problem Definition and Design Thinking

Problem Definition: The problem is to develop an earthquake prediction model using a Kaggle dataset. The objective is to explore and understand the key features of earthquake data, visualize the data on a world map for a global overview, split the data for training and testing, and build a neural network model to predict earthquake magnitudes based on the given features.

Design Thinking:

1. Data Source:

The appropriate dataset containing earthquake data with features like date, time, latitude, longitude, depth, and magnitude is Kaggle dataset.

Dataset Link: https://www.kaggle.com/datasets/usgs/earthquake-database

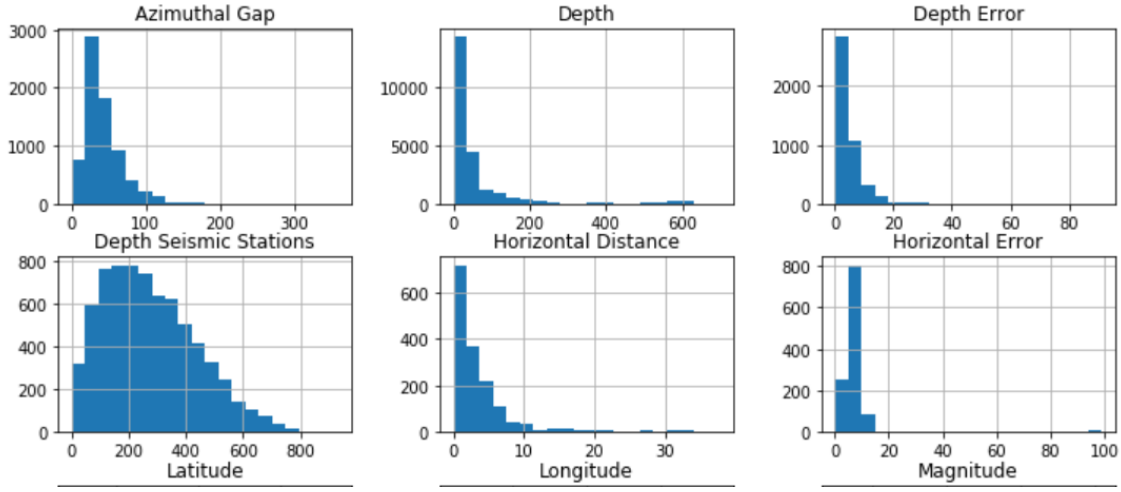
2. Feature Exploration:

To Analyze and understand the distribution, correlations, and characteristics of the key features.

Distribution: The distribution is done by the key features as histogram, the depth latitude, longitude, magnitude are as distributed as histograms. As head, describe methods are used.

```
# Plot histograms for numerical features
data.hist(bins=20, figsize=(12, 10))
plt.show()

# Density plots
sns.kdeplot(data['feature1'], shade=True)
plt.xlabel('Feature 1')
plt.ylabel('Density')
plt.show()
```

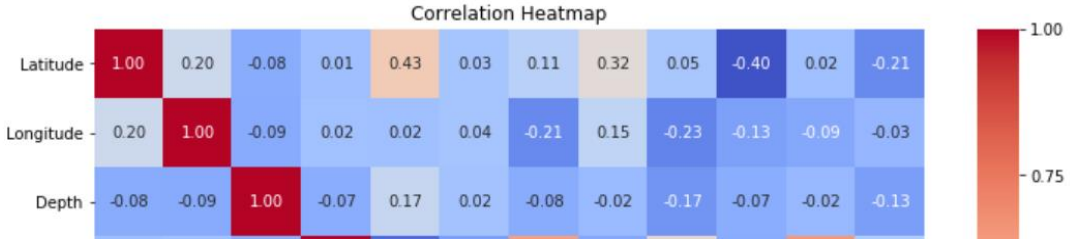


Correlations:

Exploring the correlation between features to identify any potential relationships. The correlation of the variables is done by corr() function.

```
# Calculate the correlation matrix
correlation_matrix = data.corr()

# Create a heatmap to visualize correlations
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



Characteristics of key features:

Refers to the attributes, properties, and behaviors of the variables or attributes that are considered important or influential in a dataset. Key features are those that have a significant impact on the target variable or the outcome of the analysis, and understanding their characteristics is crucial for making informed decisions in data analysis and modelling. The example of numeric features vs target variable is as key feature characteristics

```
# Scatter plots of numeric features vs. target variable
for feature in numeric_features:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(data=earthquake_data, x=feature, y='earthquake_prediction', alpha=0.5)
    plt.title(f'{feature} vs. Earthquake Prediction')
    plt.xlabel(feature)
    plt.ylabel('Earthquake Prediction')
    plt.show()
```

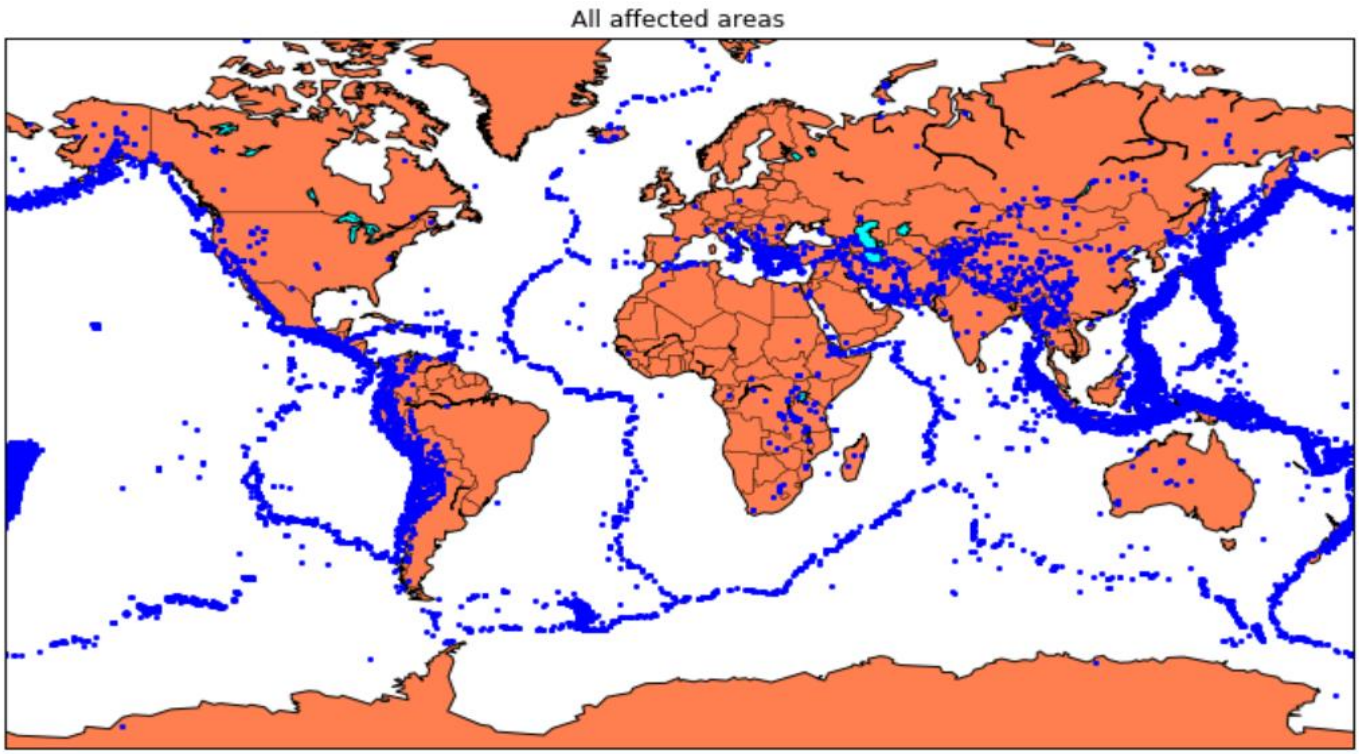
3.VISUALIZATION:

A world map visualization to display earthquake frequency distribution. Creating a world map visualization to display earthquake frequency distribution requires geographic data and a suitable library for plotting

```
from mpl_toolkits.basemap import Basemap

m = Basemap(projection='mill',llcrnrlat=-80,urcrnrlat=80, llcrnrlon=-180,urcrnrlon=180,lat_ts=20,reso

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()
```



4. Data Splitting:

Splitting the dataset into a training set and a test set for model validation. 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%.

```
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)
```

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.

```
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)
```

5. Model Development:

Building the neural network for earthquake magnitude prediction. Defining by the neural network architecture, including the number of layers, neurons per layer, and activation functions.

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer with a single neuron for magnitude prediction
])
```

The above is design code to do neural network model

```
# Scale features (standardization)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer with a single neuron for magnitude prediction
])
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae'])  # Mean squared error and
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=2)
loss, mae = model.evaluate(X_test, y_test, verbose=0)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
# Replace 'X_new' with your new data
X_new = scaler.transform(X_new)  # Scale the new data
predictions = model.predict(X_new)
```

By this code the prediction for magnitude Of earthquake can be as designed and make for the prepare for the model for the prediction of earthquake.

6. Training and Evaluation:

Training the model on the training set and evaluating its performance on the test set. Fit the chosen model to the training data. This involves setting hyperparameters and using the training data to adjust the model's internal parameters.

```
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=['ac
curacy'])
```

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

Evaluation:

Once the model is trained, evaluate its performance on the test set using appropriate evaluation metrics. Common metrics for earthquake prediction include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2).

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

the model.evaluate() method is that evaluates the x_test and y_test and the evaluation result on test data is displayed as the above code shows.