EARTHQUAKE PREDICTION MODEL USING PYTHON

Phase 2 Submission Document:

Project: Earthquake Prediction

INTRODUCTION:

- ✓ Earthquake Prediction is a way of predicting the magnitude of an earthquake based on parameters such as longitude, latitude, depth, and duration magnitude, country, and depth using machine learning to give warnings of potentially damaging earthquakes early enough to allow appropriate response to the disaster, enabling people to minimize loss of life and property.

- ✓ Earthquake prediction is a complex and challenging task that involves anticipating the occurrence, magnitude, and location of seismic events before they occur.

- ✓ Earthquakes, caused by the sudden release of energy in the Earth's crust, can have devastating consequences, making prediction efforts crucial for risk mitigation and disaster preparedness.

- ✓ Earthquake prediction aims to provide advanced warning systems and insights into seismic activities, enabling communities to take precautionary measures and potentially save lives. Traditional methods involve analyzing historical seismic data and identifying patterns that may precede significant events.

- ✓ However, predicting earthquakes with high precision remains an ongoing scientific challenge.

Key Aspects of Earthquake Prediction:

1.Seismic Data Analysis:

1. Utilizing datasets that include information on earthquake occurrences, such as time, date, location (latitude and longitude), depth, and magnitude.
2. Exploring temporal and spatial patterns within the data to identify potential precursors to seismic events.

1.Feature Engineering:

1. Extracting meaningful features from the data, such as temporal trends, spatial relationships, and interactions between different parameters.
2. Incorporating domain-specific knowledge and external factors that may influence seismic activities.

2.Machine Learning Models:

1. Developing predictive models, often based on machine learning algorithms, to analyze and learn patterns from historical earthquake data.
2. Common models include neural networks, support vector machines, and decision trees.

3.Hyperparameter Tuning:

1. Fine-tuning model parameters to optimize predictive performance through techniques like grid search or Bayesian optimization.

4.Validation and Evaluation:

1. Splitting the dataset into training and testing sets to validate the model's performance.
2. Evaluating the model's accuracy, precision, recall, and other relevant metrics.

5.Challenges in Earthquake Prediction:

1. Earthquakes are inherently unpredictable due to the dynamic and complex nature of tectonic processes.
2. Limited historical data for rare, large-magnitude earthquakes makes it challenging to train accurate models.

6.Innovation and Future Directions:

1. Advancing techniques such as deep learning, ensemble methods, and integrating real-time sensor data for improved prediction capabilities.
2. Collaborative efforts between scientists, researchers, and data scientists to explore innovative solutions.

Content For Project PHASE 2:

Preprocessing of Dataset:

Data preprocessing is a crucial step in earthquake prediction, ensuring that the dataset is clean, structured, and suitable for training machine learning models. Here's a general guide for preprocessing a dataset for earthquake prediction:

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

date	depth	mag	place	latitude	longitude	depth_avg_22	depth_avg_15	depth_avg_7	mag_avg_22	mag_avg_15	ma
2020-07-14	6.70	1.58	Oklahoma	36.171483	-97.718347	6.717727	6.560000	7.100000	1.352273	1.271333	
2020-07-14	7.55	2.07	Oklahoma	36.171483	-97.718347	6.730000	6.682667	7.132857	1.372727	1.334667	
2020-07-14	7.39	1.89	Oklahoma	36.171483	-97.718347	6.747727	6.708667	6.940000	1.396818	1.377333	
2020-07-15	7.75	1.48	Oklahoma	36.171483	-97.718347	6.834545	6.764000	6.848571	1.383182	1.388667	
2020-07-15	7.81	1.50	Oklahoma	36.171483	-97.718347	6.841364	6.854667	6.964286	1.404545	1.385333	

EXAMPLE PROGRAM:

EARTHQUAKE PREDICTION:

```
# Import necessary libraries
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
from sklearn.preprocessing import StandardScaler

# Load earthquake dataset (replace 'your_dataset.csv' with the actual file)
data = pd.read_csv('your_dataset.csv')

# Preprocessing
# Assume you have 'latitude', 'longitude', 'depth', 'date', 'time' features in your
dataset
# You might need more sophisticated preprocessing based on the specific
features in your dataset
data['datetime'] = pd.to_datetime(data['date'] + ' ' + data['time'])
data['day_of_week'] = data['datetime'].dt.dayofweek
features = ['latitude', 'longitude', 'depth', 'day_of_week']
X = data[features]
y = data['magnitude']

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Model Development
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction on the test set
y_pred = model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Example Prediction (replace the values with actual test data)
sample_data = [[latitude_value, longitude_value, depth_value,
day_of_week_value]]
sample_data_scaled = scaler.transform(sample_data)
prediction = model.predict(sample_data_scaled)
print(f'Predicted Earthquake Magnitude: {prediction[0]}')
```

- Replace 'your_dataset.csv' with the actual earthquake dataset file.

- Ensure that the dataset has the necessary features like latitude, longitude, depth, date, time, and magnitude.

- The preprocessing steps here are minimal; you might need to adjust based on your dataset's characteristics.

Data Preprocessing:

```
df = df.drop('id',axis=1)
```

```
import datetime
import time

timestamp = []
for d, t in zip(df['date'], df['time']):
  ts = datetime.datetime.strptime(d+' '+t, '%Y.%m.%d %I:%M:%S %p')
  timestamp.append(time.mktime(ts.timetuple()))
timeStamp = pd.Series(timestamp)
df['Timestamp'] = timeStamp.values
final_data = df.drop(['date', 'time'], axis=1)
final_data = final_data[final_data.Timestamp != 'ValueError']
df = final_data
df.head()
```

	lat	long	country	city	area	direction	dist	depth	xm	md	ri
0	39.04	40.38	turkey	bingol	baliklicay	west	0.1	10.0	4.1	4.1	
1	40.79	30.09	turkey	kocaeli	bayraktar_izmit	west	0.1	5.2	4.0	3.8	
2	38.58	27.61	turkey	manisa	hamzabeyli	south_west	0.1	0.0	3.7	0.0	
3	39.47	36.44	turkey	sivas	kahvepinar_sarkisla	south_west	0.1	10.0	3.5	3.5	
4	40.80	30.24	turkey	sakarya	meseli_serdivan	south_west	0.1	7.0	4.3	4.3	

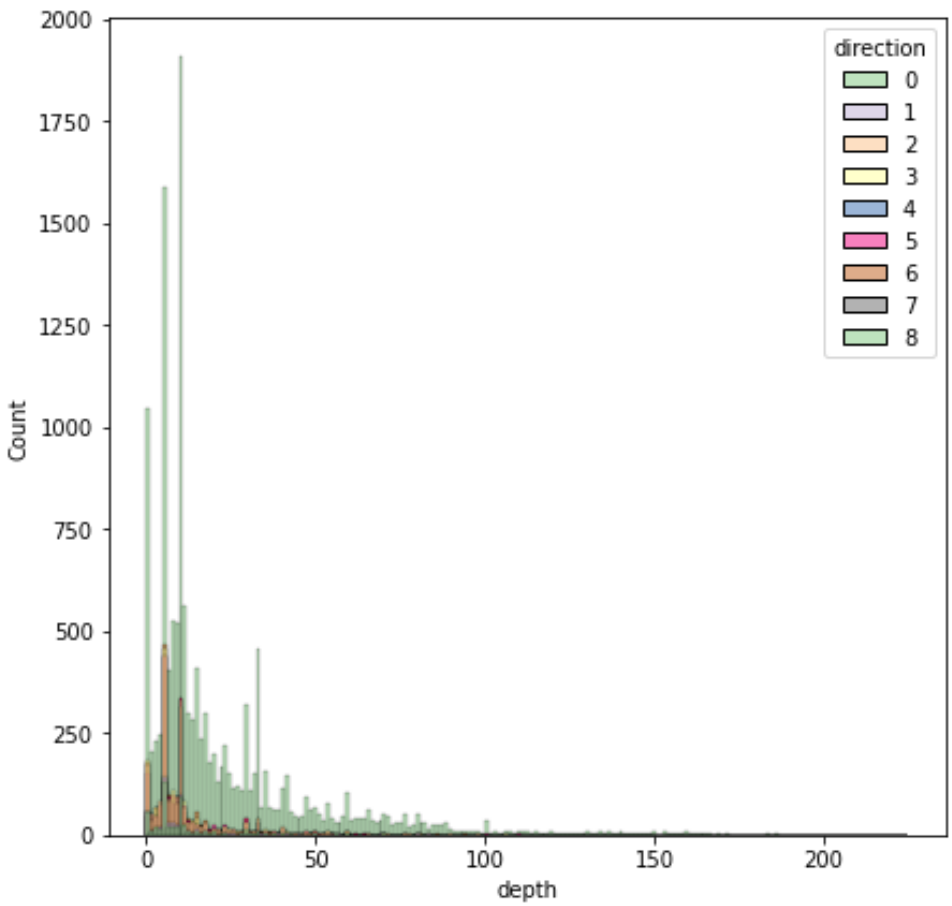
Handling Missing Values:

```
# Imputing Missing Values with Mean
si=SimpleImputer(missing_values = np.nan, strategy="mean")
si.fit(df[["dist","mw"]])
df[["dist","mw"]] = si.transform(df[["dist","mw"]])
df.isnull().sum()
```

Data Visualization:

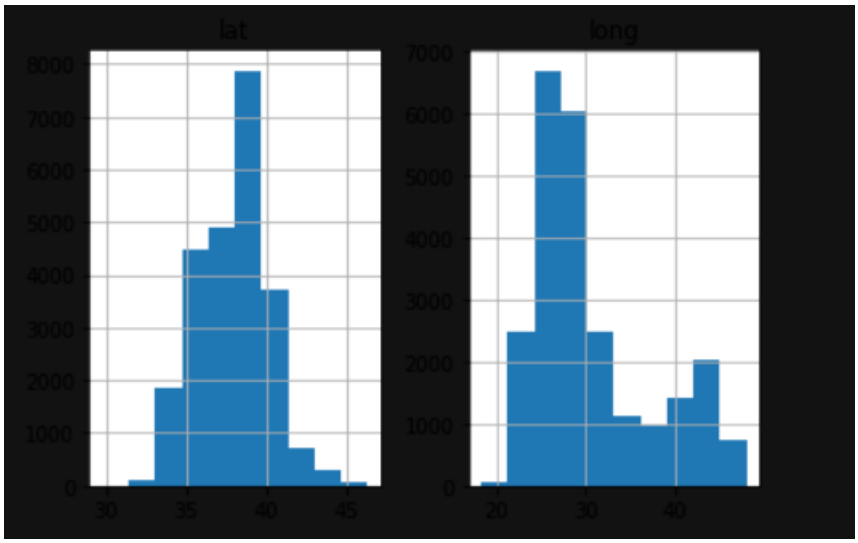
```
import plotly.express as px
px.scatter(df, x='richter',y='xm', color="direction")
```

```
plt.figure(figsize=(7,7))
sns.histplot(data=df, x='depth', hue='direction',palette = 'Accent')
plt.show()
```

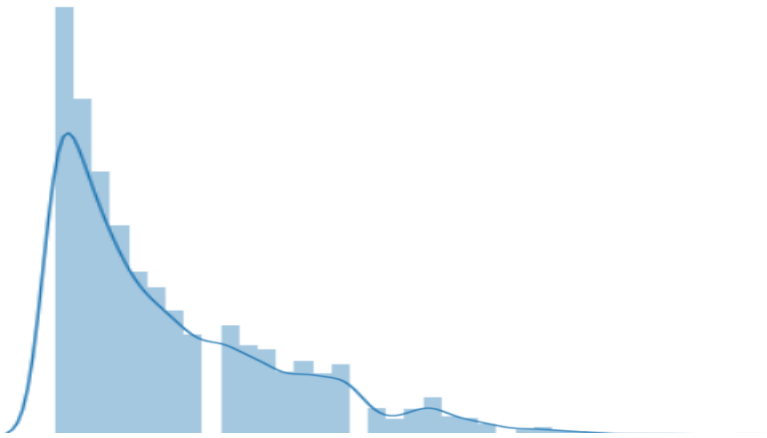


```
plt.figure(figsize=(7,7))
df[['lat','long']].hist()
plt.show()
```

```
<Figure size 504x504 with 0 Axes>
```

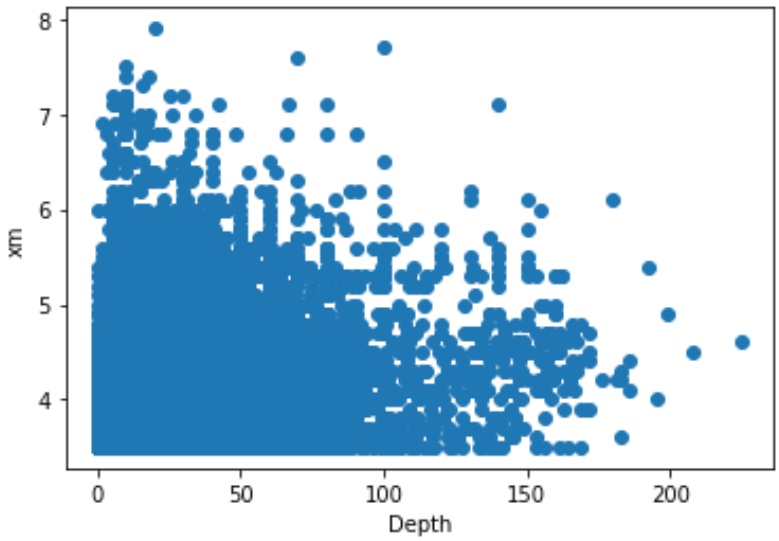



```
plt.figure(figsize=(10,10))
sns.distplot(df.xm)
```

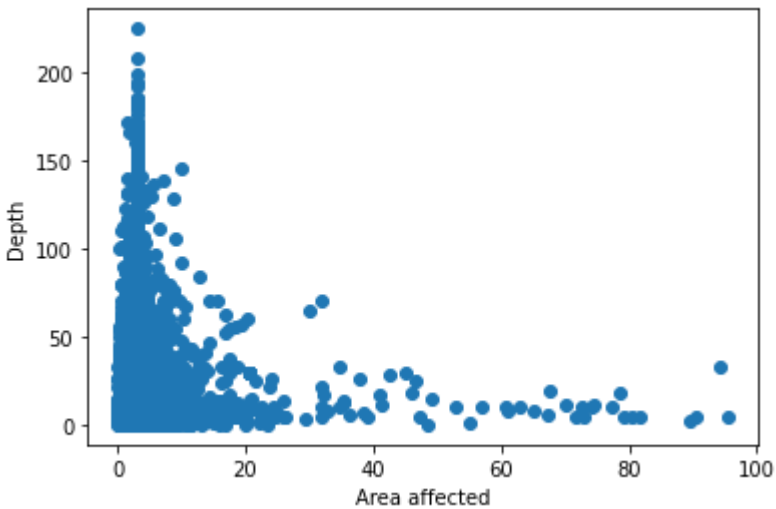


Plottings:

```
plt.scatter(df.depth, df.xm)
plt.xlabel("Depth")
plt.ylabel("xm")
plt.show()
```




```
plt.scatter(df.dist, df.depth)
plt.xlabel("Area affected")
plt.ylabel("Depth")
plt.show()
```



Splitting the Dataset

```
y=np.array(df['xm'])
X=np.array(df.drop('xm',axis=1))
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=2)
```

Creating Models

1.Linear Regression

```
from sklearn.linear_model import LinearRegression
start1 = time.time()
linear=LinearRegression()
linear.fit(X_train,y_train)
ans1 = linear.predict(X_test)
end1 = time.time()
t1 = end1-start1
```

```
accuracy1=linear.score(X_test,y_test)
print("Accuracy of Linear Regression model is:",accuracy1)
```

```
Accuracy of Linear Regression model is: 0.63134131503029
```

```
from sklearn import metrics
print("Linear Regression")
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ans1))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ans1))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, ans1)))
```

```
Linear Regression
Mean Absolute Error: 0.05878246463205686
Mean Squared Error: 0.00625827169726636
Root Mean Squared Error: 0.07910923901331854
```

2.Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
start2 = time.time()
regressor = DecisionTreeRegressor(random_state = 40)
regressor.fit(X_train,y_train)
ans2 = regressor.predict(X_test)
end2 = time.time()
t2 = end2-start2
```

```
accuracy2=regressor.score(X_test,y_test)
print("Accuracy of Decision Tree model is:",accuracy2)
```

```
Accuracy of Decision Tree model is: 0.9932960893884235
```

```
print("Decision Tree")
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ans2))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ans2))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, ans2)))
```

```
Decision Tree
Mean Absolute Error: 0.0006909999621372331
Mean Squared Error: 0.00011380416561969702
Root Mean Squared Error: 0.010667903525046383
```

3.KNN Model

```
from sklearn.neighbors import KNeighborsRegressor
start3 = time.time()
knn = KNeighborsRegressor(n_neighbors=6)
knn.fit(X_train, y_train)
ans3 = knn.predict(X_test)
end3 = time.time()
t3 = end3-start3
```

```
accuracy3=knn.score(X_test,y_test)
print("Accuracy of KNN model is:",accuracy3)
```

```
Accuracy of KNN model is: 0.8457466919393031
```

```
print("KNN Model")
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ans3))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ans3))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, ans3)))
```

```
KNN Model
Mean Absolute Error: 0.03305598677318794
Mean Squared Error: 0.002618571462992348
Root Mean Squared Error: 0.051171979275696854
```

```
x = list(info.keys())
yacc = []
for i in info:
  yacc.append(info[i][0])
plt.plot(x, yacc, 'o', color='black');
plt.xlabel("k value")
plt.ylabel("accuracy");
plt.title("Accuracy for different values of k")
```

```
Text(0.5, 1.0, 'Accuracy for different values of k')
```

CONCLUSION:

conclusion, the development of an earthquake prediction machine learning model using Python involves a systematic and multidimensional approach. Here's an elaborative summary of key aspects:

In summary, the development of an earthquake prediction model using Python is a multifaceted process that requires a combination of domain knowledge, data science expertise, and continuous improvement. Through careful preprocessing, model development, and collaboration, Python serves as a versatile tool for addressing the complexities of earthquake prediction and contributing to advancements in the field.