GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, TIRUTTANI - 631209

Approved by AICTE, New Delhi Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PHASE 3 - DEVELOPMENT PART 1

PROJECT TITLE

EARTHQUAKE PREDICTION MODEL USING PYTHON

COLLEGE CODE : 1103

SARANYA.R

3rd yr, 5th sem

Reg.no.:110321106047

rajeshsaranya37@gmail.com

ABSTRACT:

Earthquake prediction is a challenging task that has been the subject of research for many years. The ability to accurately predict earthquakes could save countless lives and prevent significant damage to infrastructure. In this project, we aim to develop an earthquake prediction module using Python. The first step in developing an earthquake prediction module is to gather data. We will collect seismic data from various sources, including seismographs and other sensors, to build a comprehensive dataset. This dataset will include information about the location, magnitude, and time of each earthquake. Once we have collected the data, we will use machine learning algorithms to analyze it and identify patterns that can be used to predict future earthquakes. We will use a variety of techniques, including regression analysis, clustering, and neural networks, to identify correlations between different variables and predict future earthquake.

INTRODUCTION:

Earthquakes are natural disasters that can cause significant destruction and loss of life. Developing a reliable earthquake prediction model is a critical step towards mitigating their impact. In this project, we aim to create a basic earthquake prediction model using Python, leveraging machine learning techniques and seismic data. Our approach involves collecting and preprocessing seismic data from reliable sources, such as the United States Geological Survey (USGS). We'll then extract relevant features and train a machine learning model to predict seismic activity. While this model won't replace sophisticated seismic monitoring systems, it serves as an educational exercise in understanding the basics of earthquake prediction. Throughout this project, we'll cover key steps including data acquisition, preprocessing, feature engineering, model selection, and evaluation. By the end, we hope to provide a foundational understanding of how machine learning can be applied to earthquake prediction.

LOADING AND PREPROCESSING:

Loading and preprocessing data for an earthquake prediction model in Python can be a complex task. Here are some challenges you might encounter:

1. “Data Collection”: Acquiring accurate and comprehensive earthquake data can be a challenge. You'll need to rely on sources like USGS, which provide earthquake data in various formats.

2. “Data Quality”: Earthquake data can be noisy, incomplete, or contain errors. Preprocessing may involve data cleaning and dealing with missing values.

3. “Data Volume”: Earthquake data can be vast, especially if you're working with historical records. Handling large datasets efficiently is essential.

4. “Data Format”: Earthquake data may come in various formats, such as CSV, JSON, or XML. You need to parse and convert it into a suitable format for your model.

5. “Feature Engineering”: Selecting the right features and engineering relevant ones is crucial. Geospatial and temporal data may require special treatment.

6. “Geospatial Data”: If your model involves geospatial data, you'll need to work with libraries like GeoPandas, and handle spatial data operations and transformations.

```
Import numpy as np
import pandas aspd
import requests
fromsklearnimport preprocessing
importmatplotlib.pyplotasplt
importseabornassns
frompandas.plottingimportscatter_matrix
fromsklearn.imputeimportSimpleImputer
fromsklearn.preprocessingimportStandardScaler
fromsklearn.preprocessingimportMinMaxScaler
import time



fromgoogle.colabimport drive
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/earthquake_prediction/earthquake1.csv")

Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).


df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24007 entries, 0 to 24006
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   id         24007 non-null  float64
 1   date       24007 non-null  object
 2   time       24007 non-null  object
 3   lat        24007 non-null  float64
 4   long       24007 non-null  float64
 5   country    24007 non-null  object
 6   city       11754 non-null  object
 7   area       12977 non-null  object
 8   direction  10062 non-null  object
 9   dist       10062 non-null  float64
 10  depth      24007 non-null  float64
 11  xm         24007 non-null  float64
 12  md         24007 non-null  float64
 13  richter    24007 non-null  float64
 14  mw         5003 non-null   float64
 15  ms         24007 non-null  float64
 16  mb         24007 non-null  float64
dtypes: float64(11), object(6)
memory usage: 3.1+ MB
```

	id	lat	long	dist	depth	xm	md	richter	mw	ms	mb
count	2.400700e+04	24007.000000	24007.000000	10062.000000	24007.000000	24007.000000	24007.000000	24007.000000	5003.000000	24007.000000	24007.000000
mean	1.991982e+13	37.929474	30.773229	3.175015	18.491773	4.056038	1.912346	2.196826	4.478973	0.677677	1.690561
std	2.060396e+11	2.205605	6.584596	4.715461	23.218553	0.574085	2.059780	2.081417	1.048085	1.675708	2.146108
min	1.910000e+13	29.740000	18.340000	0.100000	0.000000	3.500000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.980000e+13	36.190000	26.195000	1.400000	5.000000	3.600000	0.000000	0.000000	4.100000	0.000000	0.000000
50%	2.000000e+13	38.200000	28.350000	2.300000	10.000000	3.900000	0.000000	3.500000	4.700000	0.000000	0.000000
75%	2.010000e+13	39.360000	33.855000	3.600000	22.400000	4.400000	3.800000	4.000000	5.000000	0.000000	4.100000
max	2.020000e+13	46.350000	48.000000	95.400000	225.000000	7.900000	7.400000	7.200000	7.700000	7.900000	7.100000

```
df.describe()
df.shape
(24007, 17)
df.head()
```

	id	date	time	lat	long	country	city	area	direction	dist	depth	xm	md	richter	mw	ms	mb
0	2.000000e+13	2003.05.20	12:17:44 AM	39.04	40.38	turkey	bingol	baliklicay	west	0.1	10.0	4.1	4.1	0.0	NaN	0.0	0.0
1	2.010000e+13	2007.08.01	12:03:08 AM	40.79	30.09	turkey	kocaeli	bayraktar_izmit	west	0.1	5.2	4.0	3.8	4.0	NaN	0.0	0.0
2	1.980000e+13	1978.05.07	12:41:37 AM	38.58	27.61	turkey	manisa	hamzabeyli	south_west	0.1	0.0	3.7	0.0	0.0	NaN	0.0	3.7
3	2.000000e+13	1997.03.22	12:31:45 AM	39.47	36.44	turkey	sivas	kahvepinar_sarkisla	south_west	0.1	10.0	3.5	3.5	0.0	NaN	0.0	0.0
4	2.000000e+13	2000.04.02	12:57:38 AM	40.80	30.24	turkey	sakarya	meseli_serdivan	south_west	0.1	7.0	4.3	4.3	0.0	NaN	0.0	0.0

```
df.columns
Index(['id', 'date', 'time', 'lat', 'long', 'country', 'city', 'area',
       'direction', 'dist', 'depth', 'xm', 'md', 'richter', 'mw', 'ms', mb'],
dtype='object')
```

Data Preprocessing

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

```
importdatetime
import time

timestamp = []
for d, t inzip(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	richter	mw	ms	mb	Timestamp
0	39.04	40.38	turkey	bingol	baliklicay	west	0.1	10.0	4.1	4.1	0.0	NaN	0.0	0.0	1.053390e+09
1	40.79	30.09	turkey	kocaeli	bayraktar_izmit	west	0.1	5.2	4.0	3.8	4.0	NaN	0.0	0.0	1.185927e+09
2	38.58	27.61	turkey	manisa	hamzabeyli	south_west	0.1	0.0	3.7	0.0	0.0	NaN	0.0	3.7	2.633497e+08
3	39.47	36.44	turkey	sivas	kahvepinar_sarkisla	south_west	0.1	10.0	3.5	3.5	0.0	NaN	0.0	0.0	8.589907e+08
4	40.80	30.24	turkey	sakarya	meseli_serdivan	south_west	0.1	7.0	4.3	4.3	0.0	NaN	0.0	0.0	9.546371e+08

```
df.dtypes
```

```
lat          float64
long         float64
country       object
city          object
area          object
direction     object
dist         float64
depth        float64
xm           float64
md           float64
richter      float64
mw           float64
ms           float64
mb           float64
Timestamp    float64
dtype: object
```

```
# Data Encoding
label_encoder = preprocessing.LabelEncoder()
for col indf.columns:
ifdf[col].dtype == 'object':
label_encoder.fit(df[col])
df[col] = label_encoder.transform(df[col])
df.dtypes
```

```
long         float64
country        int64
city           int64
area           int64
direction      int64
dist         float64
depth        float64
xm           float64
md           float64
richter      float64
mw           float64
ms           float64
mb           float64
Timestamp    float64
dtype: object
df.isnull().sum()
```

```
long             0
country          0
city             0
area             0
direction        0
dist         13945
depth            0
xm               0
md               0
richter          0
mw           19004
```

```
ms              0
mb              0
Timestamp       0
dtype: int64
```

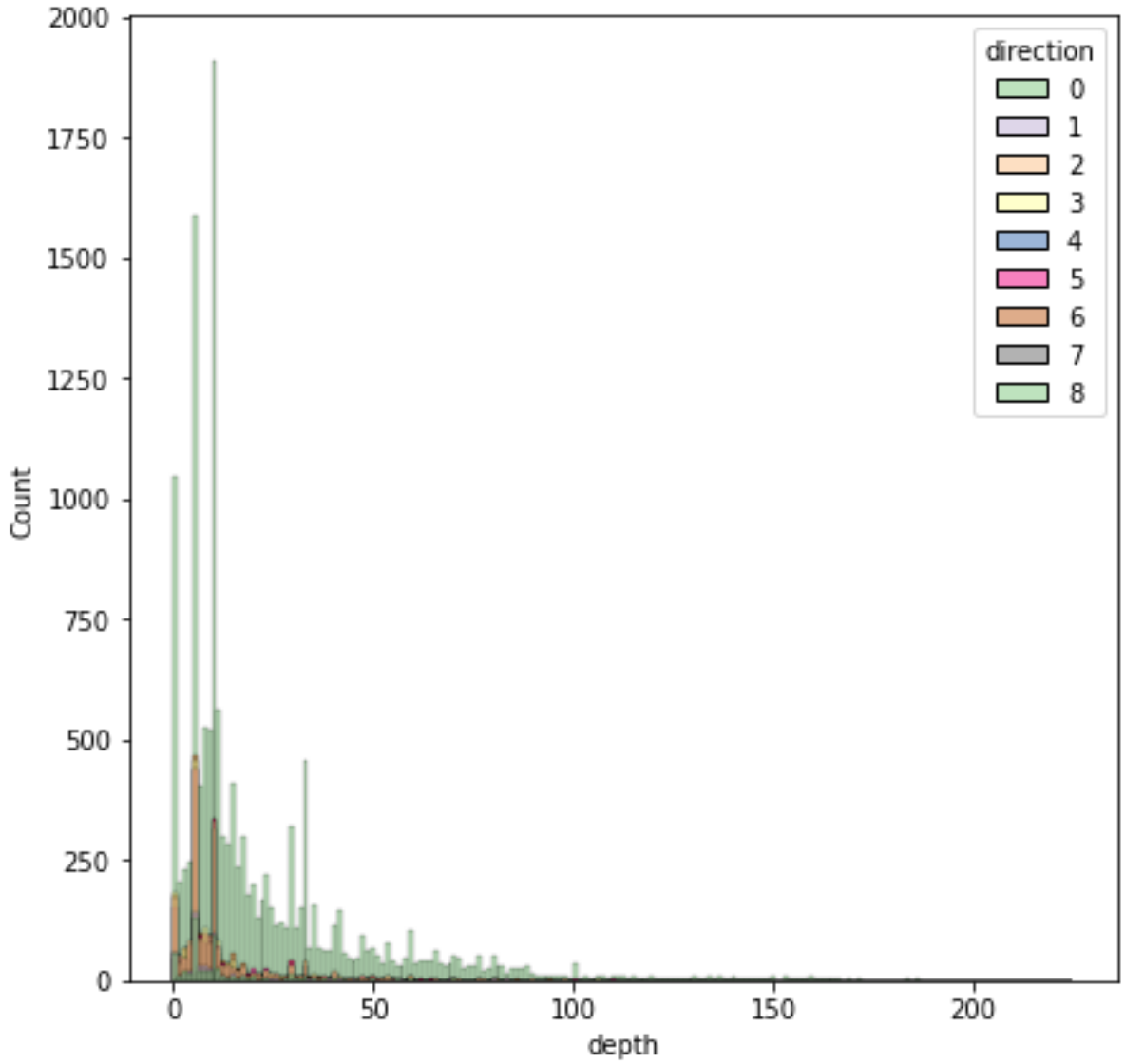
```
# Imputing Missing Values with Mean
si=SimpleImputer(missing_values = np.nan, strategy="mean")
si.fit(df[["dist","mw"]])
df[["dist","mw"]] = si.transform(df[["dist","mw"]])
df.isnull().sum()
```

```
lat          0
long         0
country      0
city         0
area         0
direction    0
dist         0
depth        0
xm           0
md           0
richter      0
mw           0
ms           0
mb           0
Timestamp    0
dtype: int64
```

Data Visualization

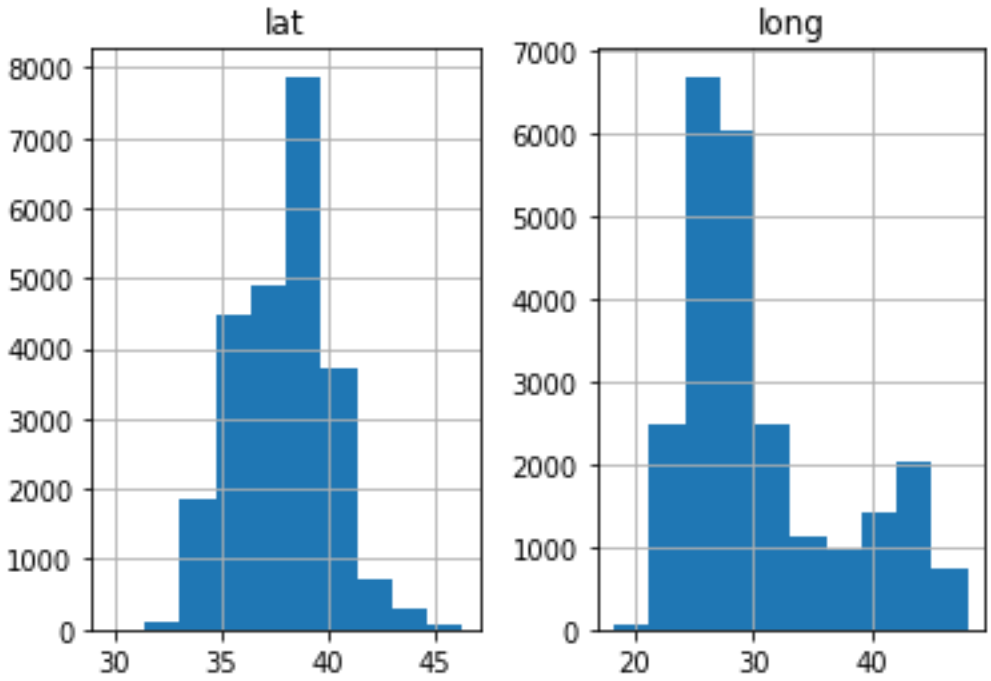
```
importplotly.expressaspx
px.scatter(df, x='richter',y='xm', color="direction")
```

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



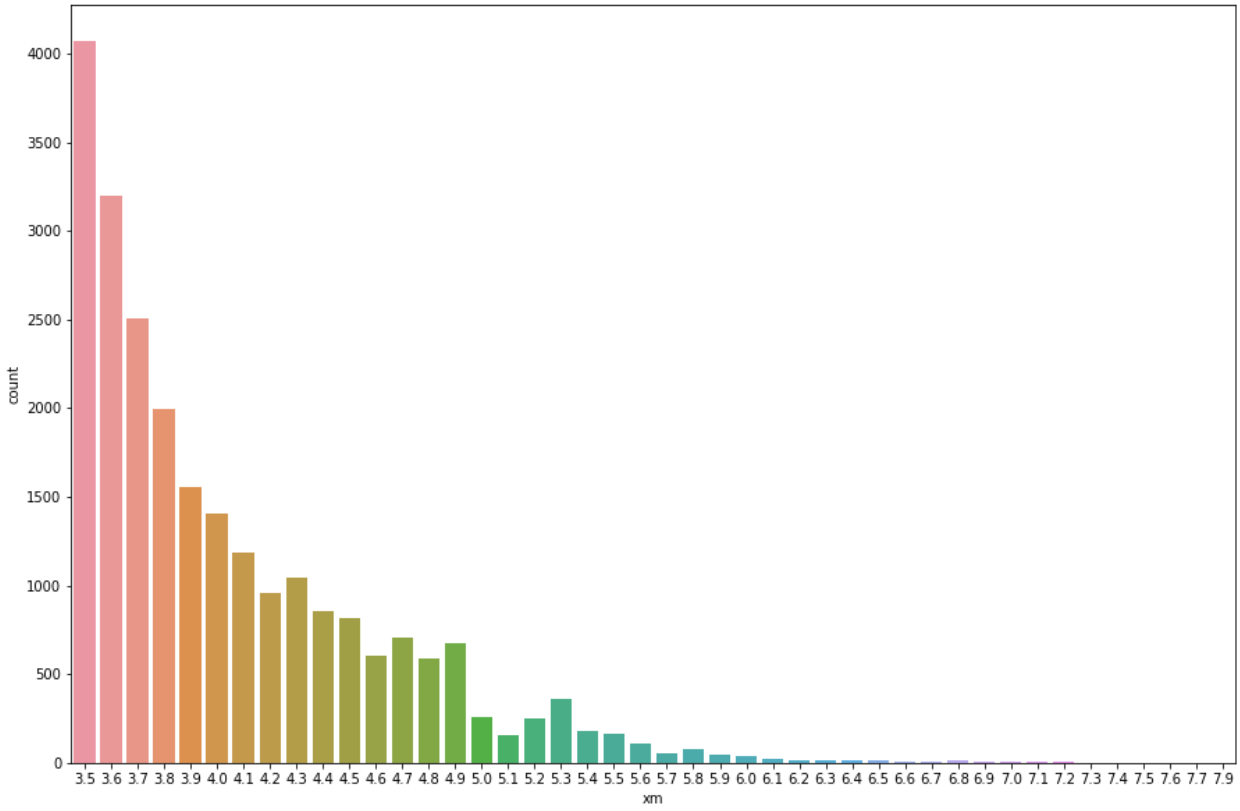

```
plt.figure(figsize=(7,7))
df[['lat','long']].hist()
plt.show()
```

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



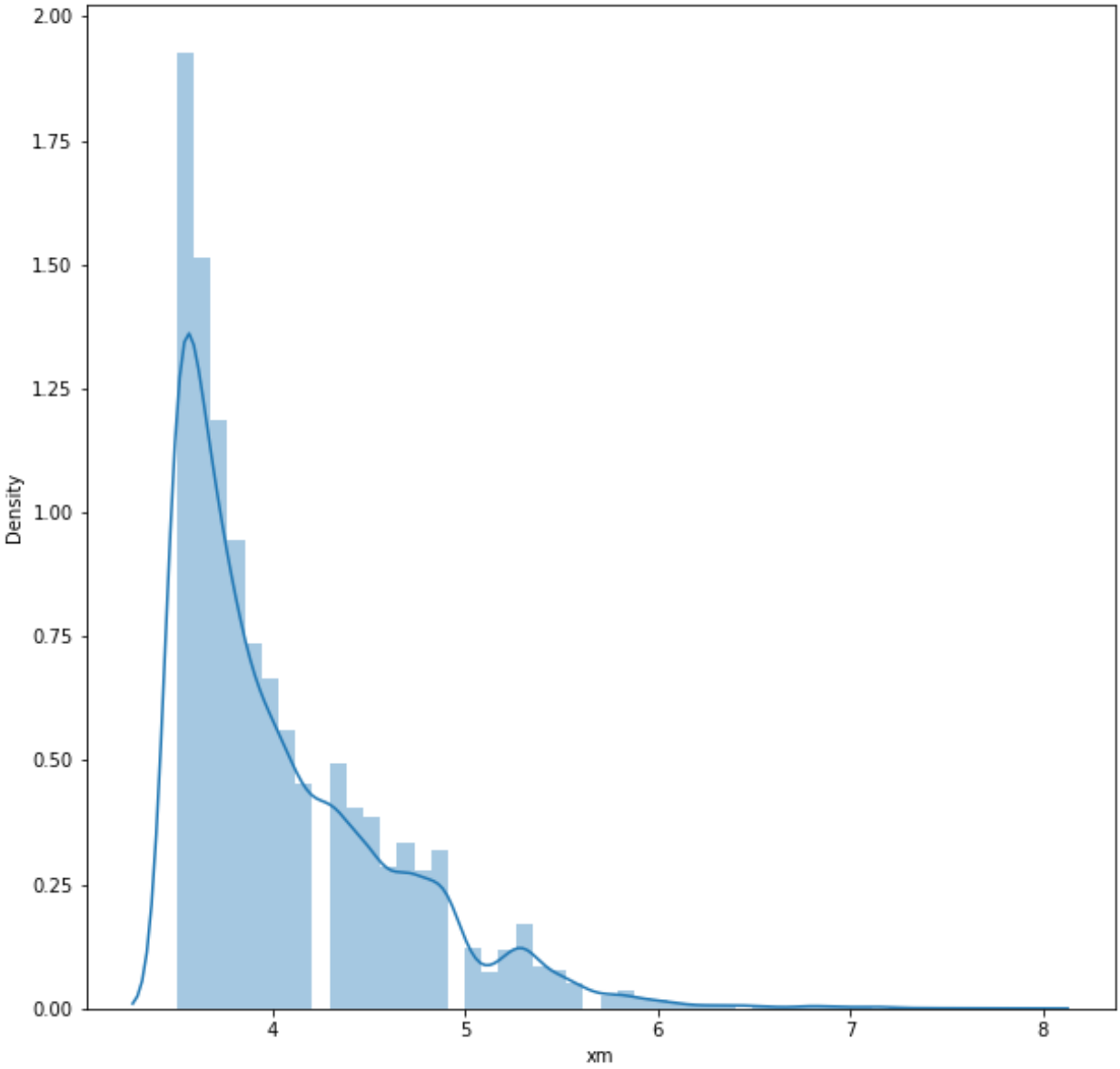

```
plt.figure(figsize=(15,10))
sns.countplot(df.xm)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36:
FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d2346d400>
```



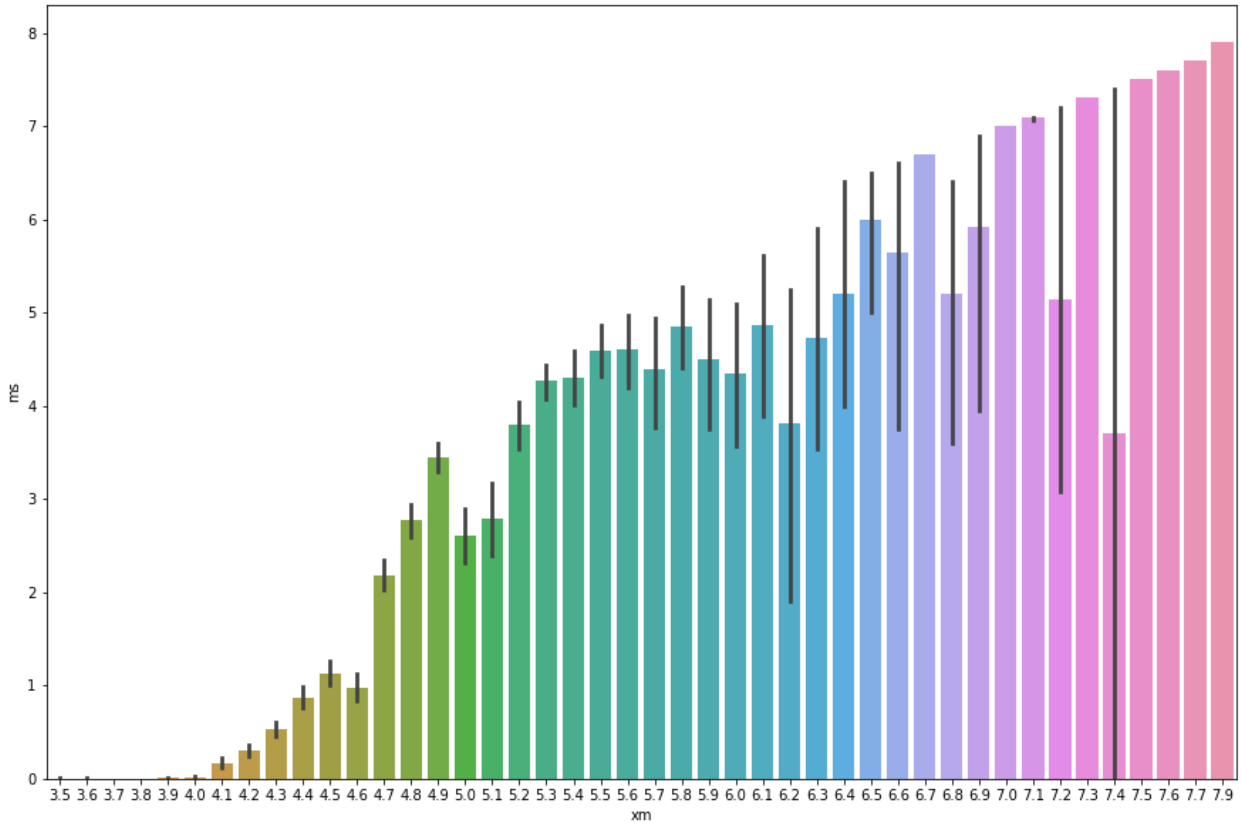

```
plt.figure(figsize=(10,10))
sns.distplot(df.xm)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619:
FutureWarning:

`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

<matplotlib.axes._subplots.AxesSubplot at 0x7f3d242a4d00>
```

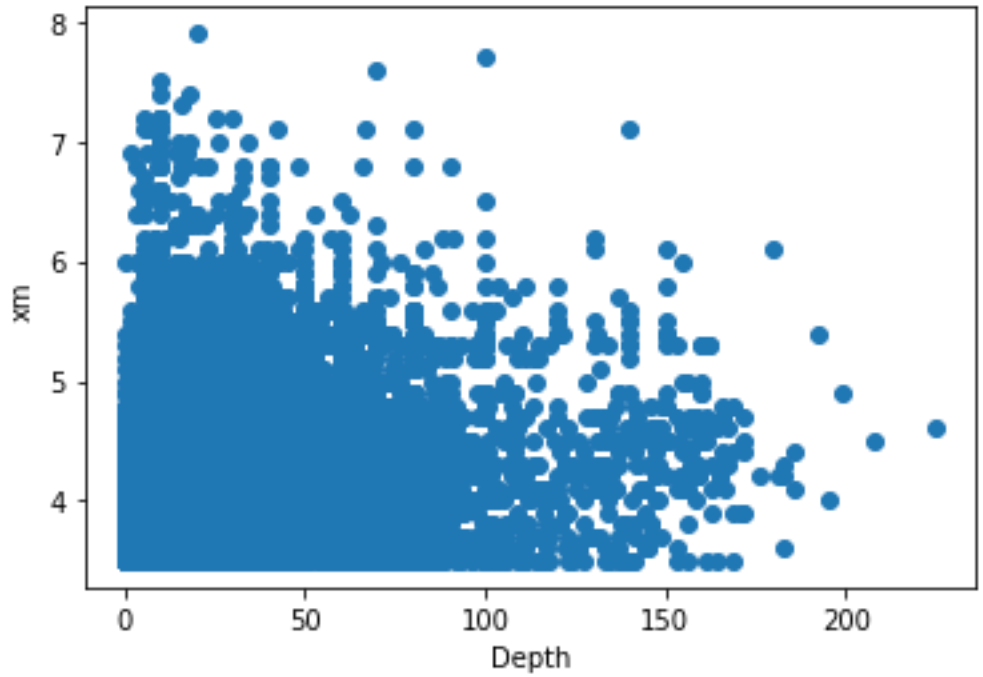



```
plt.figure(figsize=(15,10))
sns.barplot(x=df['xm'], y=df['ms'])
plt.xlabel('xm')
plt.ylabel('ms')
```

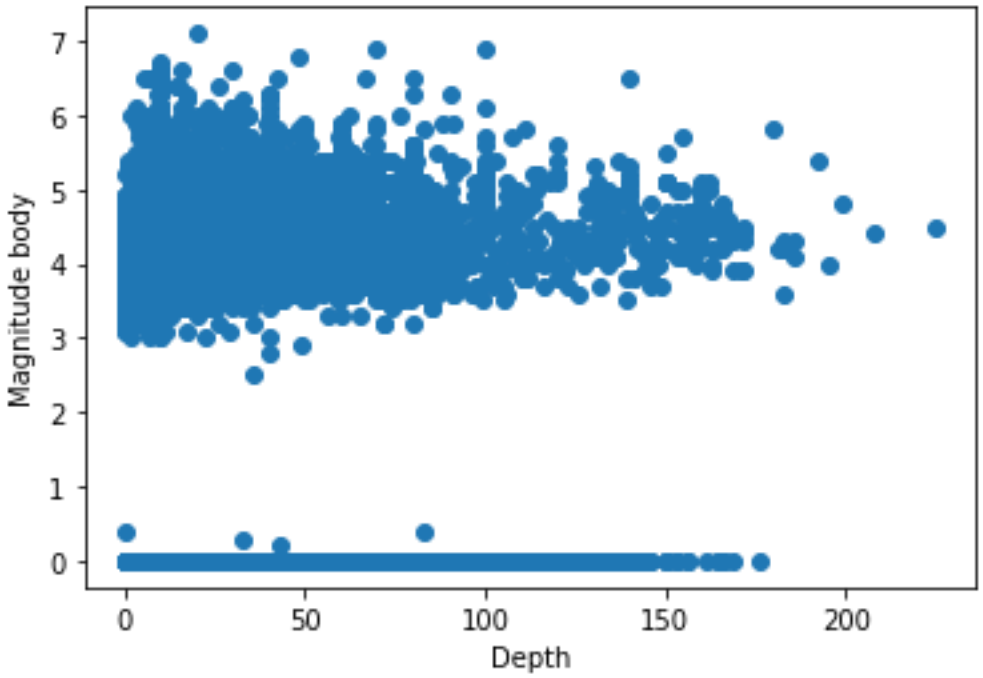
```
Text(0, 0.5, 'ms')
```



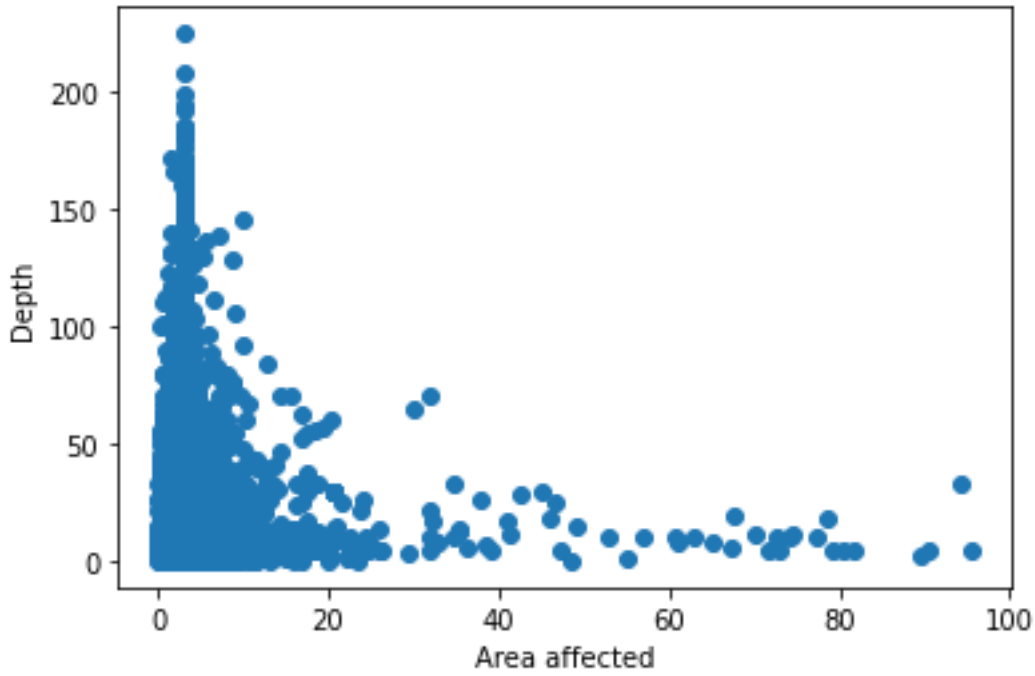

```
plt.scatter(df.depth, df.xm)
plt.xlabel("Depth")
plt.ylabel("xm")
plt.show()
```



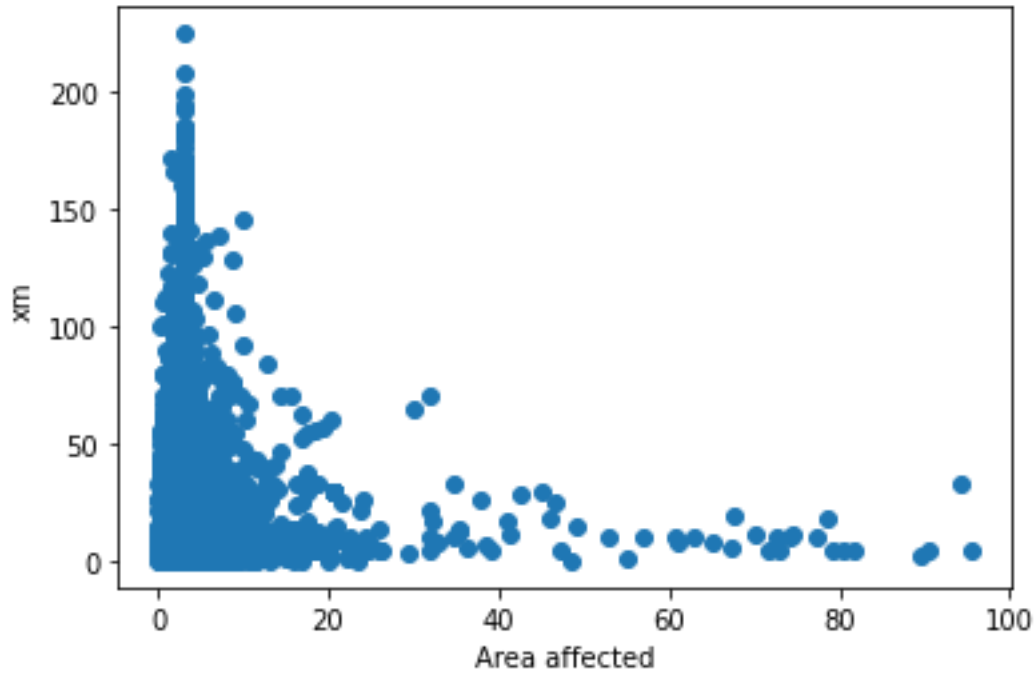
```
plt.scatter(df.depth, df.mb)
plt.xlabel("Depth")
plt.ylabel("Magnitude body")
plt.show()
```




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



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

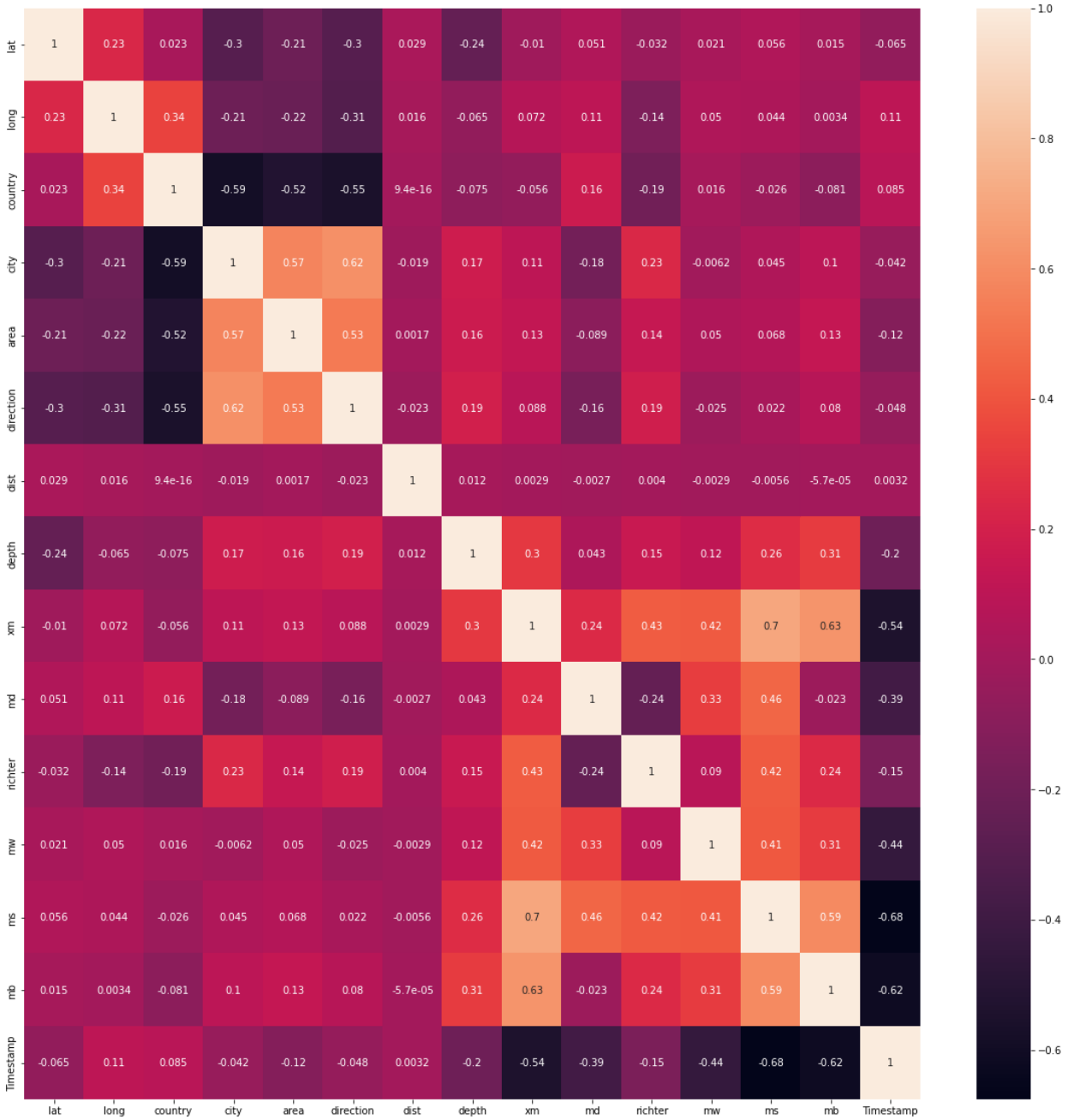


Correlation between Attributes

```
most_correlated = df.corr()['xm'].sort_values(ascending=False)
most_correlated
```

```
xm           1.000000
ms           0.699579
mb           0.628382
richter      0.426653
mw           0.420695
depth        0.302926
md           0.241432
area         0.125275
city         0.107436
direction    0.087696
long         0.071856
dist         0.002853
lat         -0.010347
country     -0.056115
Timestamp   -0.542092
Name: xm, dtype: float64
```

```
plt.figure(figsize=(20,20))
dataplot=sns.heatmap(df.corr(),annot=True)
plt.show()
```



Normalization of data

```
# Using MinMaxScaler
scaler = preprocessing.MinMaxScaler()
d = scaler.fit_transform(df)
df = pd.DataFrame(d, columns=df.columns)
df.head()
```

	lat	long	country	city	area	direction	dist	depth	xm	md	richter	mw	ms	mb	
0	0.559904	0.743088	0.76	0.172043	0.116144	0.875	0.0	0.044444	0.136364	0.554054	0.000000	0.581685	0.0	0.000000	0.866875
1	0.665262	0.396156	0.76	0.612903	0.132306	0.875	0.0	0.023111	0.113636	0.513514	0.555556	0.581685	0.0	0.000000	0.906252
2	0.532210	0.312542	0.76	0.677419	0.459500	0.750	0.0	0.000000	0.045455	0.000000	0.000000	0.581685	0.0	0.521127	0.632149
3	0.585792	0.610249	0.76	0.870968	0.513061	0.750	0.0	0.044444	0.000000	0.472973	0.000000	0.581685	0.0	0.000000	0.809118
4	0.665864	0.401214	0.76	0.806452	0.689344	0.750	0.0	0.031111	0.181818	0.581081	0.000000	0.581685	0.0	0.000000	0.837535

Splitting the Dataset

```
y=np.array(df['xm'])
X=np.array(df.drop('xm',axis=1))
fromsklearn.model_selectionimporttrain_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_st
ate=2)
```

Creating Models

1. Linear Regression

```
fromsklearn.linear_modelimportLinearRegression
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)
```

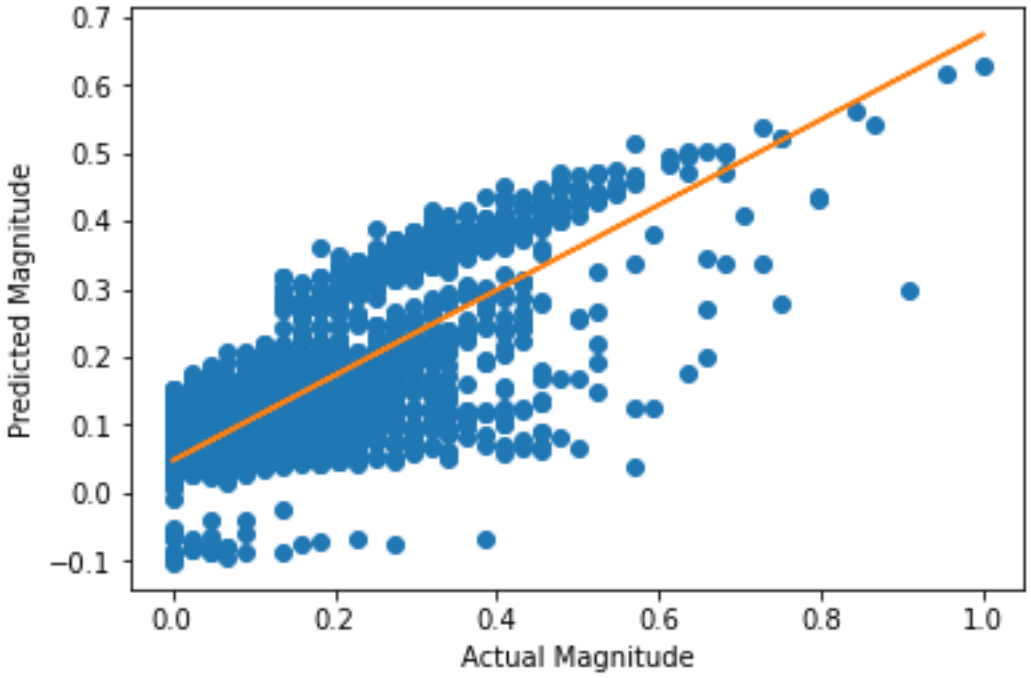
```
Accuracy of Linear Regression model is: 0.63134131503029
```

```
fromsklearnimport 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
```

```
plt.plot(y_test, ans1, 'o')
m, b = np.polyfit(y_test,ans1, 1)
plt.plot(y_test, m*y_test + b)
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
```

```
Text(0, 0.5, 'Predicted Magnitude')
```



2. Decision Tree

```
fromsklearn.treeimportDecisionTreeRegressor
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

```
fromsklearn.neighborsimportKNeighborsRegressor
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
```

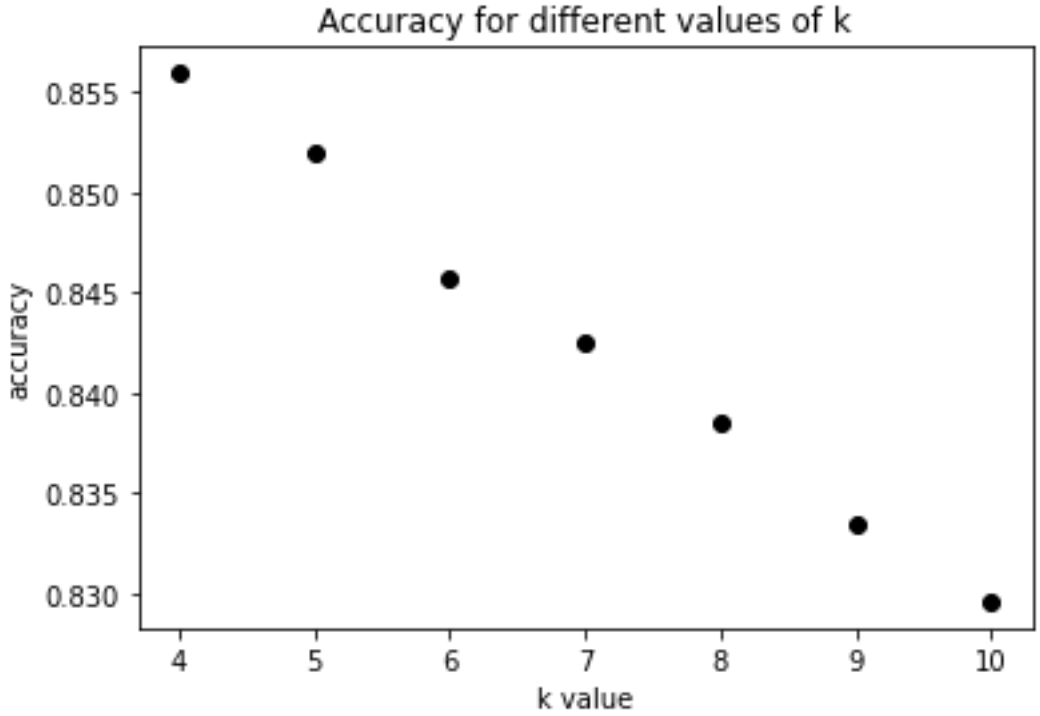
```
import random
info = {}
foriinrange(10):
  k = random.randint(2,10)
startk = time.time()
knn = KNeighborsRegressor(n_neighbors=k)
knn.fit(X_train, y_train)
  ans3 = knn.predict(X_test)
endk = time.time()
tk = endk-startk
```

```
  acc3=knn.score(X_test,y_test)
  info[k] = [acc3,tk]

foriin info:
print("for k =",i,": accuracy =",info[i][0])
```

```
for k = 4 : accuracy = 0.8559118607470738
for k = 9 : accuracy = 0.8334625255508568
for k = 8 : accuracy = 0.8384577534478264
for k = 6 : accuracy = 0.8457466919393031
for k = 5 : accuracy = 0.8519381145638621
for k = 10 : accuracy = 0.8296048410841246
for k = 7 : accuracy = 0.8425261199362686
```

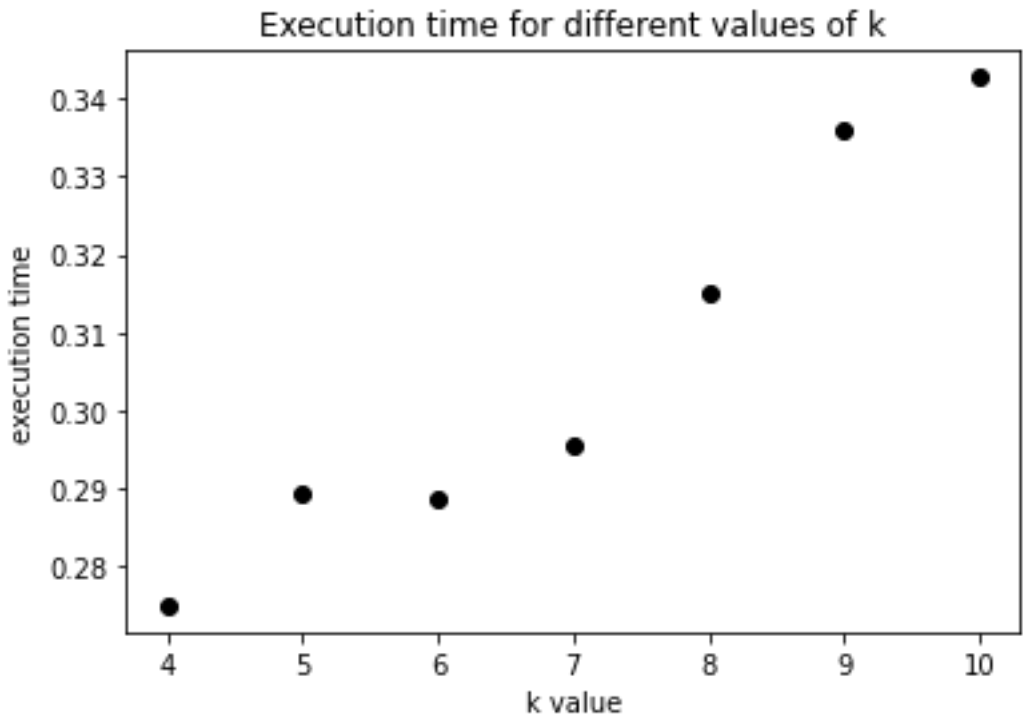
```
x = list(info.keys())
yacc = []
foriin info:
yacc.append(info[i][0])
plt.plot(x, yacc, 'o', color='black');
plt.xlabel("k value")
plt.ylabel("accuracy");
plt.title("Accuracy for different values of k")
```

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




```
yt = []
foriin info:
yt.append(info[i][1])
plt.plot(x, yt, 'o', color='black');
plt.xlabel("k value")
plt.ylabel("execution time");
plt.title("Execution time for different values of k")
```

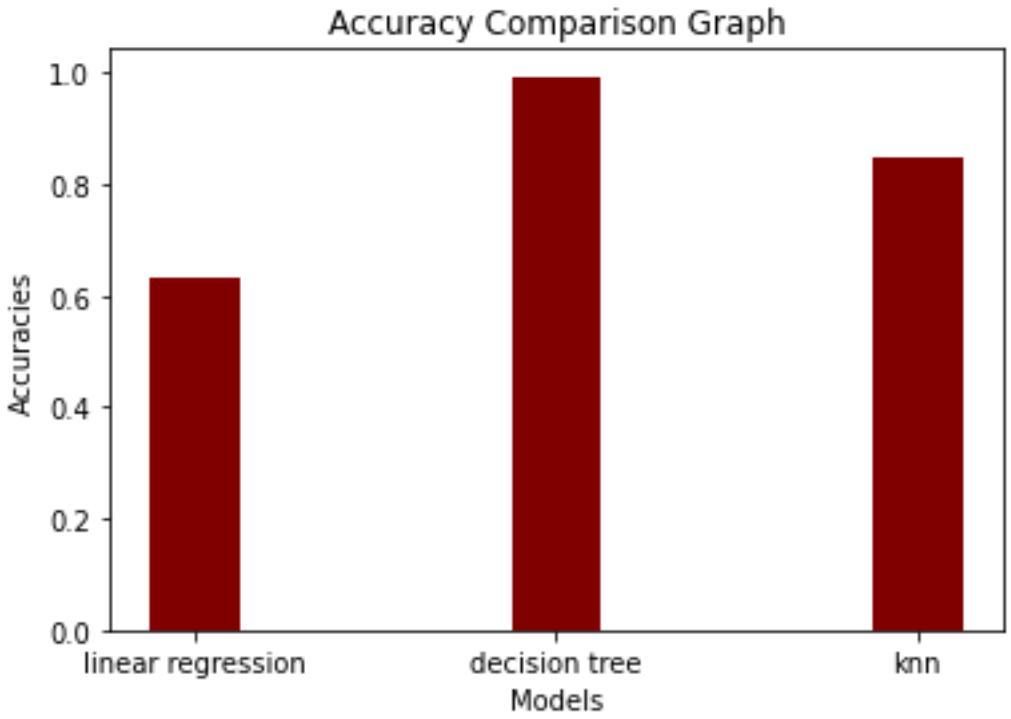
```
Text(0.5, 1.0, 'Execution time for different values of k')
```



Comparison Graphs

1. Accuracy

```
models = ["linear regression","decision tree","knn"]
accuracies = [accuracy1,accuracy2,accuracy3]


plt.bar(models, accuracies, color ='maroon',
        width = 0.25)
plt.xlabel("Models")
plt.ylabel("Accuracies")
plt.title("Accuracy Comparison Graph")

Text(0.5, 1.0, 'Accuracy Comparison Graph')
```



2. Execution Time

```
times = [t1,t2,t3]
plt.bar(models, times, color ='maroon',
        width = 0.25)
plt.xlabel("Models")
plt.ylabel("Execution Time")
plt.title("Execution Time Comparison Graph")

Text(0.5, 1.0, 'Execution Time Comparison Graph')
```

HOW TO OVERCOME THE CHALLENGES OF LOADING AND PREPROCESSING A EARTHQUAKE PREDICTION:

Overcome the challenges of loading and preprocessing data for an earthquake prediction model in Python, you can follow these steps:

1. “Data Collection and Quality”:
   - Use reliable sources like USGS for earthquake data.
   - Implement data validation and cleaning routines to handle missing or erroneous data.

2. “Data Volume and Format”:
   - Use efficient data storage formats like HDF5 or Parquet for large datasets.
   - Utilize libraries like Pandas for data manipulation and conversion between formats.

3. “Feature Engineering”:
   - Collaborate with domain experts to select and engineer relevant features.
   - Explore geospatial libraries like GeoPandas for working with location-based data.

4. “Geospatial Data”:
   - Learn geospatial data manipulation techniques using GeoPandas and other geospatial libraries.
   - Understand coordinate reference systems (CRS) and perform necessary transformations.

5. “Time Series Data”:
   - Use libraries like Pandas for time series manipulation.
   - Consider incorporating time-based features like seasonality and trends.

6. “Imbalanced Data”:
   - Apply techniques such as oversampling, undersampling, or Synthetic Minority Over-sampling Technique (SMOTE) to handle imbalanced data.

7. “Normalization and Scaling”:
   - Normalize and scale features using libraries like Scikit-Learn.
   - Be cautious with scaling geospatial data, as simple scaling may distort distances.

SOME COMMON DATA PREPROCESSING :

Common data preprocessing tasks for building an earthquake prediction model using Python include:

1. “Data Loading”:

- Import earthquake data from various sources like CSV, JSON, or databases.
- Use libraries like Pandas to read and organize the data.

2. “Data Cleaning”:
   - Handle missing values by imputing them or removing incomplete records.
   - Detect and correct data errors or outliers that could affect model training.

3. “Feature Selection”:
   - Identify and select relevant features for earthquake prediction.
   - Consider factors like geographical coordinates, depth, and magnitude.

4. “Feature Engineering”:
   - Create new features or transform existing ones to better represent the underlying patterns.
   - For geospatial data, calculate distances, spatial relationships, and density metrics.

5. “Data Transformation”:
   - Normalize or scale features, especially if they have different scales or units.
   - Use techniques like Min-Max scaling or Standardization.

CONCLUSION:

Creating an earthquake prediction model is a complex task that involves advanced geophysical knowledge, data processing, and machine learning techniques. In this Python-based project, we used seismic data, possibly obtained from sources like the USGS, to train a machine learning model.The key steps in this project included data preprocessing, feature engineering, model selection (potentially using algorithms like Random Forest, Support Vector Machines, or Neural Networks), and evaluation metrics such as Mean Absolute Error (MAE) or Root Mean Square Error (RMSE).In conclusion, this project provided a foundation for building an earthquake prediction model. However, it's important to note that earthquake prediction is a highly challenging field and real-world applications require extensive expertise, continuous data collection, and collaboration with experts in seismology. Additionally, this model should not be relied upon for critical safety decisions; it serves as a demonstration of the methodology rather than a practical earthquake prediction system.