In [2]:



*#In this project we are predicting POSSIBILITY(in the form of 0 or 1) & the #our model will give 1 as the outcome for earthquake will ouccur and 0 as t* **import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.preprocessing **import** LabelEncoder,StandardScaler

**from** sklearn.linear\_model **import** LinearRegression,LogisticRegression

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.svm **import** SVC

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.metrics **import** mean\_squared\_error,r2\_score,accuracy\_score,clas

In [3]:



data**=**pd.read\_csv("C://Users//ridhi//Downloads//query (1) (1).csv",encoding**=**

In [5]:

data

Out[5]: **time latitude longitude depth mag magType nst gap dmin rm**

**0** 2024-03-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -6.2159 | 146.9556 | 86.441 | 5.10 | mb | 96.0 | 67.0 | 3.175 | 0.5 |
| 64.6558 | -17.7271 | 10.000 | 4.70 | mb | 157.0 | 59.0 | 1.549 | 1.0 |
| 33.6817 | 93.1840 | 10.000 | 4.00 | mb | 25.0 | 90.0 | 4.336 | 0.7 |
| -22.2034 | -176.6137 | 150.411 | 4.50 | mb | 39.0 | 90.0 | 1.667 | 0.6 |
| 38.5875 | 70.4189 | 12.980 | 4.10 | mb | 46.0 | 114.0 | 1.395 | 0.7 |
| ... | ... | ... | ... | ... | ... | ... | ... | . |
| 18.7645 | -64.9665 | 53.710 | 3.41 | md | 11.0 | 242.0 | 0.410 | 0.2 |
| -31.2599 | 117.6740 | 7.210 | 4.60 | mb | 35.0 | 79.0 | 1.703 | 0.8 |
| -31.1853 | 117.6124 | 10.069 | 4.40 | mb | 25.0 | 92.0 | 1.766 | 0.7 |
| 2.1882 | 126.6833 | 44.615 | 4.50 | mb | 44.0 | 112.0 | 1.564 | 0.8 |
| 16.0523 | -94.9727 | 10.000 | 4.40 | mb | 34.0 | 180.0 | 2.735 | 0.6 |

18T00:18:03.554Z

**1** 2024-03-

18T00:45:47.063Z

**2** 2024-03-

18T01:25:01.175Z

**3** 2024-03-

18T01:39:31.936Z

**4** 2024-03-

18T02:10:38.894Z

**...** ...

**9709** 2024-08-

18T15:58:57.520Z

**9710** 2024-08-

18T18:05:38.347Z

**9711** 2024-08-

18T18:33:40.498Z

**9712** 2024-08-

18T18:44:00.241Z

**9713** 2024-08-

18T19:28:20.845Z

9714 rows × 22 columns



In [6]:

data.head()

Out[6]: **time latitude longitude depth mag magType nst gap dmin rms .**

**0** 2024-03-

18T00:18:03.554Z

-6.2159 146.9556 86.441 5.1 mb 96.0 67.0 3.175 0.57 .

**1** 2024-03-

18T00:45:47.063Z

64.6558 -17.7271 10.000 4.7 mb 157.0 59.0 1.549 1.08 .

**2** 2024-03-

18T01:25:01.175Z

33.6817 93.1840 10.000 4.0 mb 25.0 90.0 4.336 0.78 .

**3** 2024-03-

18T01:39:31.936Z

-22.2034 -176.6137 150.411 4.5 mb 39.0 90.0 1.667 0.60 .

**4** 2024-03-

18T02:10:38.894Z

38.5875 70.4189 12.980 4.1 mb 46.0 114.0 1.395 0.76 .

In [7]:

5 rows × 22 columns



Out[7]:

data.describe()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **latitude** | **longitude** | **depth** | **mag** | **nst** | **gap** |  |
| **count** | 9714.000000 | 9714.000000 | 9714.000000 | 9714.000000 | 9008.000000 | 9008.000000 | 8999.0 |
| **mean** | 17.824587 | -29.209848 | 66.582442 | 3.860644 | 43.302731 | 119.889833 | 2.5 |
| **std** | 30.117838 | 128.706527 | 117.616662 | 0.860700 | 38.384349 | 66.834599 | 4.3 |
| **min** | -65.277000 | -179.989500 | -1.660000 | 2.500000 | 0.000000 | 11.000000 | 0.0 |
| **25%** | -5.999875 | -155.209417 | 10.000000 | 2.952500 | 19.000000 | 69.000000 | 0.3 |
| **50%** | 19.387750 | -69.079500 | 18.102000 | 4.200000 | 31.000000 | 104.000000 | 1.1 |
| **75%** | 41.561200 | 121.731725 | 65.738000 | 4.500000 | 53.000000 | 159.000000 | 3.0 |
| **max** | 86.525000 | 179.998400 | 658.420000 | 7.400000 | 401.000000 | 355.000000 | 53.2 |



In [8]:

data.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 9714 entries, 0 to 9713

Data columns (total 22 columns):

# Column Non-Null Count Dtype

1. time 9714 non-null object
2. latitude 9714 non-null float64
3. longitude 9714 non-null float64
4. depth 9714 non-null float64
5. mag 9714 non-null float64
6. magType 9714 non-null object
7. nst 9008 non-null float64
8. gap 9008 non-null float64
9. dmin 8999 non-null float64
10. rms 9714 non-null float64
11. net 9714 non-null object
12. id 9714 non-null object
13. updated 9714 non-null object
14. place 9714 non-null object
15. type 9714 non-null object
16. horizontalError 8937 non-null float64
17. depthError 9714 non-null float64
18. magError 8953 non-null float64
19. magNst 9000 non-null float64
20. status 9714 non-null object
21. locationSource 9714 non-null object
22. magSource 9714 non-null object dtypes: float64(12), object(10)

memory usage: 1.6+ MB

In [9]:



*#some of the columns in this dataset are not beneficial for predicting our* data**=**data.drop(["id","updated","status","locationSource","magSource","depth data

Out[9]: **time latitude longitude depth mag magType nst gap dmin rm**

**0** 2024-03-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -6.2159 | 146.9556 | 86.441 | 5.10 | mb | 96.0 | 67.0 | 3.175 | 0.5 |
| 64.6558 | -17.7271 | 10.000 | 4.70 | mb | 157.0 | 59.0 | 1.549 | 1.0 |
| 33.6817 | 93.1840 | 10.000 | 4.00 | mb | 25.0 | 90.0 | 4.336 | 0.7 |
| -22.2034 | -176.6137 | 150.411 | 4.50 | mb | 39.0 | 90.0 | 1.667 | 0.6 |
| 38.5875 | 70.4189 | 12.980 | 4.10 | mb | 46.0 | 114.0 | 1.395 | 0.7 |
| ... | ... | ... | ... | ... | ... | ... | ... | . |
| 18.7645 | -64.9665 | 53.710 | 3.41 | md | 11.0 | 242.0 | 0.410 | 0.2 |
| -31.2599 | 117.6740 | 7.210 | 4.60 | mb | 35.0 | 79.0 | 1.703 | 0.8 |
| -31.1853 | 117.6124 | 10.069 | 4.40 | mb | 25.0 | 92.0 | 1.766 | 0.7 |
| 2.1882 | 126.6833 | 44.615 | 4.50 | mb | 44.0 | 112.0 | 1.564 | 0.8 |
| 16.0523 | -94.9727 | 10.000 | 4.40 | mb | 34.0 | 180.0 | 2.735 | 0.6 |

18T00:18:03.554Z

**1** 2024-03-

18T00:45:47.063Z

**2** 2024-03-

18T01:25:01.175Z

**3** 2024-03-

18T01:39:31.936Z

**4** 2024-03-

18T02:10:38.894Z

**...** ...

**9709** 2024-08-

18T15:58:57.520Z

**9710** 2024-08-

18T18:05:38.347Z

**9711** 2024-08-

18T18:33:40.498Z

**9712** 2024-08-

18T18:44:00.241Z

**9713** 2024-08-

18T19:28:20.845Z

9714 rows × 12 columns



In [10]:



*#adding a column earthquake which has value as 0 when magnitude of earthqua*

data['earthquake']**=**np.where(data['mag']**>=**3,1,0) data

Out[10]: **time latitude longitude depth mag magType nst gap dmin rm**

**0** 2024-03-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -6.2159 | 146.9556 | 86.441 | 5.10 | mb | 96.0 | 67.0 | 3.175 | 0.5 |
| 64.6558 | -17.7271 | 10.000 | 4.70 | mb | 157.0 | 59.0 | 1.549 | 1.0 |
| 33.6817 | 93.1840 | 10.000 | 4.00 | mb | 25.0 | 90.0 | 4.336 | 0.7 |
| -22.2034 | -176.6137 | 150.411 | 4.50 | mb | 39.0 | 90.0 | 1.667 | 0.6 |
| 38.5875 | 70.4189 | 12.980 | 4.10 | mb | 46.0 | 114.0 | 1.395 | 0.7 |
| ... | ... | ... | ... | ... | ... | ... | ... | . |
| 18.7645 | -64.9665 | 53.710 | 3.41 | md | 11.0 | 242.0 | 0.410 | 0.2 |
| -31.2599 | 117.6740 | 7.210 | 4.60 | mb | 35.0 | 79.0 | 1.703 | 0.8 |
| -31.1853 | 117.6124 | 10.069 | 4.40 | mb | 25.0 | 92.0 | 1.766 | 0.7 |
| 2.1882 | 126.6833 | 44.615 | 4.50 | mb | 44.0 | 112.0 | 1.564 | 0.8 |
| 16.0523 | -94.9727 | 10.000 | 4.40 | mb | 34.0 | 180.0 | 2.735 | 0.6 |

18T00:18:03.554Z

**1** 2024-03-

18T00:45:47.063Z

**2** 2024-03-

18T01:25:01.175Z

**3** 2024-03-

18T01:39:31.936Z

**4** 2024-03-

18T02:10:38.894Z

**...** ...

**9709** 2024-08-

18T15:58:57.520Z

**9710** 2024-08-

18T18:05:38.347Z

**9711** 2024-08-

18T18:33:40.498Z

**9712** 2024-08-

18T18:44:00.241Z

**9713** 2024-08-

18T19:28:20.845Z

9714 rows × 13 columns



In [11]:



data['earthquake'].value\_counts()

Out[11]: earthquake

1 7250

0 2464

Name: count, dtype: int64

In [12]:

data.isnull()

Out[12]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **time** | **latitude** | **longitude** | **depth** | **mag** | **magType** | **nst** | **gap** | **dmin** | **rms** | **net** | **pla** |
| **0** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **1** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **2** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **3** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **4** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **9709** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **9710** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **9711** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **9712** | False | False | False | False | False | False | False | False | False | False | False | Fa |
| **9713** | False | False | False | False | False | False | False | False | False | False | False | Fa |

9714 rows × 13 columns



In [13]:



*#finding the no of null values in each column*

data.isnull().sum()

|  |  |  |
| --- | --- | --- |
| Out[13]: | time | 0 |
|  | latitude | 0 |
|  | longitude | 0 |
|  | depth | 0 |
|  | mag | 0 |
|  | magType | 0 |
|  | nst | 706 |
|  | gap | 706 |
|  | dmin | 715 |
|  | rms | 0 |
|  | net | 0 |
|  | place | 0 |
|  | earthquake | 0 |
|  | dtype: int64 |  |

In [14]:



*#nst,gap,dmin are the columns with the null values. #finding the datatype of each of them*

print("nst has dtype ",data["nst"].dtype) print("gap has dtype ",data["gap"].dtype) print("dmin has dtype ",data["dmin"].dtype)

nst has dtype float64 gap has dtype float64 dmin has dtype float64

In [15]:



*#since all of the columns having null values are numeric*

*#therefore replacing the null values with the mean of each of the columns*

data["nst"].fillna(data["nst"].mean(),inplace**=True**) data["gap"].fillna(data["gap"].mean(),inplace**=True**) data["dmin"].fillna(data["dmin"].mean(),inplace**=True**)

In [16]:



print(data["nst"].isnull().sum()," ",data["gap"].isnull().sum()," ",data["d

0 0 0

In [17]:



*#working with the dates*

*#converting the data type of date\_time column from object to datetime*

data['time']**=**pd.to\_datetime(data['time']) data["month"]**=**data["time"].dt.month

*#dropping the time column from the dataset*

data**=**data.drop("time",axis**=**1) data

Out[17]: **latitude longitude depth mag magType nst gap dmin rms net place**

**0** -6.2159 146.9556 86.441 5.10 mb 96.0 67.0 3.175 0.57 us

**1** 64.6558 -17.7271 10.000 4.70 mb 157.0 59.0 1.549 1.08 us

**2** 33.6817 93.1840 10.000 4.00 mb 25.0 90.0 4.336 0.78 us

**3** -22.2034 -176.6137 150.411 4.50 mb 39.0 90.0 1.667 0.60 us

**4** 38.5875 70.4189 12.980 4.10 mb 46.0 114.0 1.395 0.76 us

56 km N of Lae, Papua New Guinea

115 km S

of Akureyri, Iceland

266 km NNE of Nagqu, China

177 km SW of Houma, Tonga

49 km S of Rasht, Tajikistan

**...** ... ... ... ... ... ... ... ... ... ... ...

46 km N

of Charlotte

**9709** 18.7645 -64.9665 53.710 3.41 md 11.0 242.0 0.410 0.20 pr

**9710** -31.2599 117.6740 7.210 4.60 mb 35.0 79.0 1.703 0.87 us

**9711** -31.1853 117.6124 10.069 4.40 mb 25.0 92.0 1.766 0.73 us

**9712** 2.1882 126.6833 44.615 4.50 mb 44.0 112.0 1.564 0.86 us

Amalie,

U.S.

Virgin Islands

62 km WNW of Merredin, Australia

71 km WNW of Merredin, Australia

156 km WNW of Tobelo,

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | Indonesia  17 km S of San |
| **9713** | 16.0523 | -94.9727 | 10.000 | 4.40 | mb | 34.0 | 180.0 | 2.735 | 0.68 | us | Mateo |
|  |  |  |  |  |  |  |  |  |  |  | del Mar, |
|  |  |  |  |  |  |  |  |  |  |  | Mexico |

9714 rows × 13 columns



In [18]:



*#extracting the exact places which are affected*

data['place'] **=** data['place'].apply(**lambda** x: x.split(', ')[1] **if** ', ' **in** x data['place']

Out[18]: 0 Papua New Guinea

1. Iceland
2. China
3. Tonga
4. Tajikistan

...

|  |  |  |
| --- | --- | --- |
| 9709 | U.S. | Virgin Islands |
| 9710 |  | Australia |
| 9711 |  | Australia |
| 9712 |  | Indonesia |
| 9713 |  | Mexico |
| Name: | place, | Length: 9714, dtype: object |

In [19]:

*#visualising our dataset*

data["place"].value\_counts()

Out[19]: place

Alaska 1759

Indonesia 575

Hawaii 469

Puerto Rico 442

CA 419

...

Grenada 1

Mozambique 1

Nunavut 1

Norwegian Sea 1

Saint Lucia 1

Name: count, Length: 240, dtype: int64

In [20]:



series3**=**data["mag"].value\_counts() x2**=**np.array(series3.index)

y2**=**series3.values

print("datatype of x2 ",type(x2)) print("datatype of y2 ",type(y2)) print("\n")

fig**=**plt.figure(figsize**=**(5,5)) plt.hist(x2)

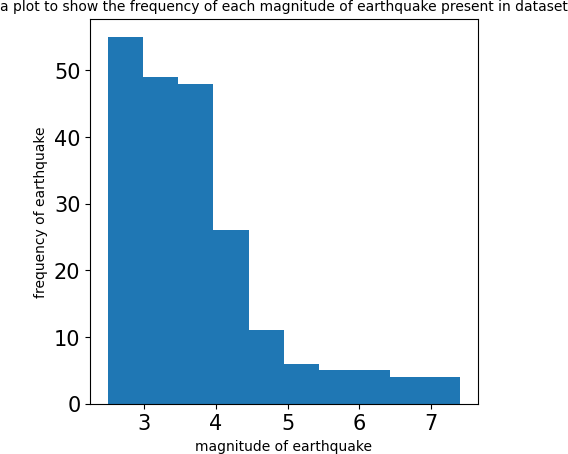
plt.xticks(fontsize**=**15) plt.yticks(fontsize**=**15)

plt.ylabel("frequency of earthquake",fontsize**=**10)

plt.title("a plot to show the frequency of each magnitude of earthquake pre plt.xlabel("magnitude of earthquake",fontsize**=**10)

plt.show()

datatype of x2 <class 'numpy.ndarray'> datatype of y2 <class 'numpy.ndarray'>



In [21]:



series2**=**data["place"].value\_counts() x1**=**np.array(series2.index)

y1**=**series2.values

print("datatype of x1 ",type(x1)) print("datatype of y1 ",type(y1)) print("\n")

fig**=**plt.figure(figsize**=**(30,70)) plt.barh(x1,y1)

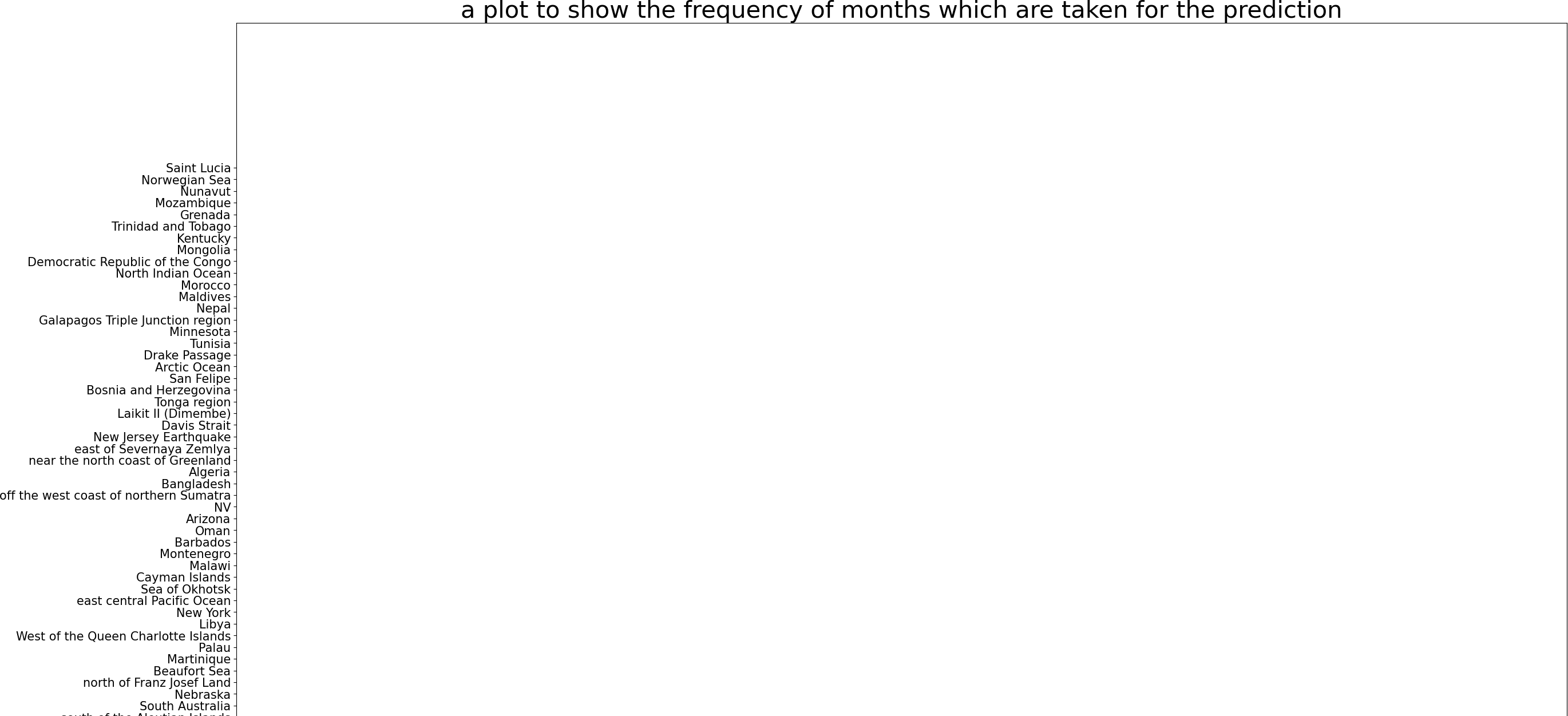
plt.xticks(fontsize**=**30) plt.yticks(fontsize**=**15)

plt.ylabel("name of place",fontsize**=**30)

plt.title("a plot to show the frequency of months which are taken for the p plt.xlabel("frequency",fontsize**=**30)

plt.show()

datatype of x1 <class 'numpy.ndarray'> datatype of y1 <class 'numpy.ndarray'>



In [22]:

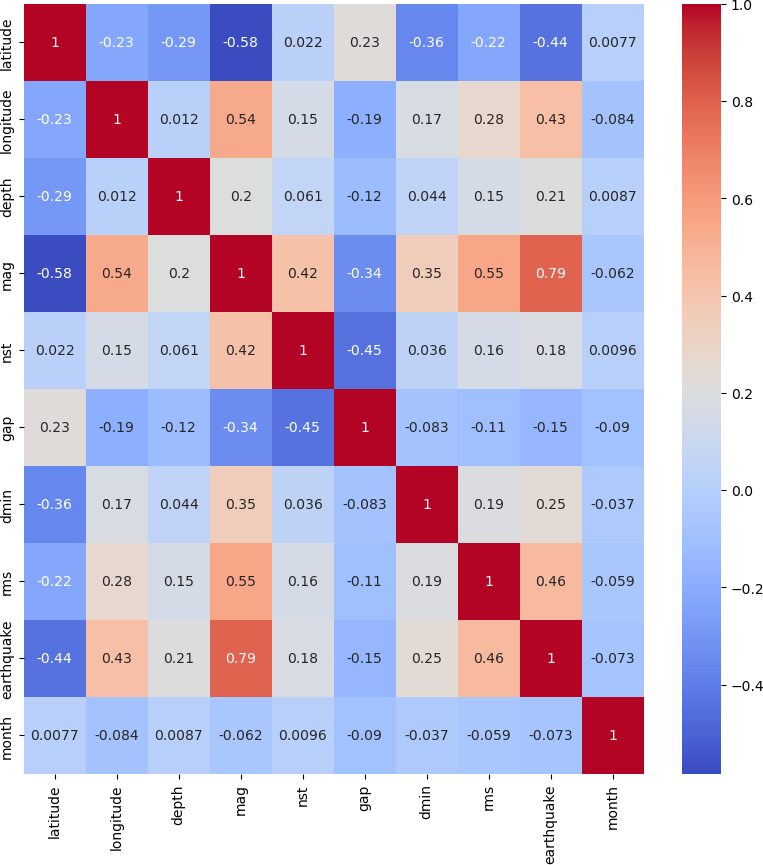


*#visualising correlations*

fig**=**plt.figure(figsize**=**(10,10))

correlation\_matrix**=**data.corr(numeric\_only**=True**)

sns.heatmap(correlation\_matrix,annot**=True**,cmap**=**'coolwarm') plt.show()



In [23]:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| data |  | | | | | | | | | | |
|  | **latitude** | **longitude** | **depth** | **mag** | **magType** | **nst** | **gap** | **dmin** | **rms** | **net** | **place** |
| **0** | -6.2159 | 146.9556 | 86.441 | 5.10 | mb | 96.0 | 67.0 | 3.175 | 0.57 | us | Papua New |
|  |  |  |  |  |  |  |  |  |  |  | Guinea |
| **1** | 64.6558 | -17.7271 | 10.000 | 4.70 | mb | 157.0 | 59.0 | 1.549 | 1.08 | us | Iceland |
| **2** | 33.6817 | 93.1840 | 10.000 | 4.00 | mb | 25.0 | 90.0 | 4.336 | 0.78 | us | China |
| **3** | -22.2034 | -176.6137 | 150.411 | 4.50 | mb | 39.0 | 90.0 | 1.667 | 0.60 | us | Tonga |
| **4** | 38.5875 | 70.4189 | 12.980 | 4.10 | mb | 46.0 | 114.0 | 1.395 | 0.76 | us | Tajikistan |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9709** | 18.7645 | -64.9665 | 53.710 | 3.41 | md | 11.0 | 242.0 | 0.410 | 0.20 | pr | U.S.  Virgin |
|  |  |  |  |  |  |  |  |  |  |  | Islands |
| **9710** | -31.2599 | 117.6740 | 7.210 | 4.60 | mb | 35.0 | 79.0 | 1.703 | 0.87 | us | Australia |
| **9711** | -31.1853 | 117.6124 | 10.069 | 4.40 | mb | 25.0 | 92.0 | 1.766 | 0.73 | us | Australia |
| **9712** | 2.1882 | 126.6833 | 44.615 | 4.50 | mb | 44.0 | 112.0 | 1.564 | 0.86 | us | Indonesia |
| **9713** | 16.0523 | -94.9727 | 10.000 | 4.40 | mb | 34.0 | 180.0 | 2.735 | 0.68 | us | Mexico |

Out[23]:

9714 rows × 13 columns



In [24]:

data.columns

Out[24]: Index(['latitude', 'longitude', 'depth', 'mag', 'magType', 'nst', 'gap',

'dmin', 'rms', 'net', 'place', 'earthquake', 'month'], dtype='object')

In [25]:



data**=**data[['month','latitude', 'longitude', 'depth', 'magType', 'nst', 'gap

In [26]:

data.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 9714 entries, 0 to 9713

Data columns (total 13 columns):

# Column Non-Null Count Dtype

* 1. month 9714 non-null int32
  2. latitude 9714 non-null float64
  3. longitude 9714 non-null float64
  4. depth 9714 non-null float64
  5. magType 9714 non-null object
  6. nst 9714 non-null float64
  7. gap 9714 non-null float64
  8. dmin 9714 non-null float64
  9. rms 9714 non-null float64
  10. net 9714 non-null object
  11. place 9714 non-null object
  12. earthquake 9714 non-null int32
  13. mag 9714 non-null float64 dtypes: float64(8), int32(2), object(3) memory usage: 910.8+ KB

In [27]:



*#Encoding the categorical data ie converting datatypes of those columns whi #using label encoding*

label\_encoders**=**{}

categorical\_columns**=**['net','magType','place',]

**for** column **in** categorical\_columns:

label\_encoders[column]**=**LabelEncoder()

data[column]**=**label\_encoders[column].fit\_transform(data[column])

data

Out[27]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **month** | **latitude** | **longitude** | **depth** | **magType** | **nst** | **gap** | **dmin** | **rms** | **net** | **place** | **e** |
| **0** | 3 | -6.2159 | 146.9556 | 86.441 | 0 | 96.0 | 67.0 | 3.175 | 0.57 | 11 | 137 |  |
| **1** | 3 | 64.6558 | -17.7271 | 10.000 | 0 | 157.0 | 59.0 | 1.549 | 1.08 | 11 | 68 |  |
| **2** | 3 | 33.6817 | 93.1840 | 10.000 | 0 | 25.0 | 90.0 | 4.336 | 0.78 | 11 | 36 |  |
| **3** | 3 | -22.2034 | -176.6137 | 150.411 | 0 | 39.0 | 90.0 | 1.667 | 0.60 | 11 | 179 |  |
| **4** | 3 | 38.5875 | 70.4189 | 12.980 | 0 | 46.0 | 114.0 | 1.395 | 0.76 | 11 | 174 |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **9709** | 8 | 18.7645 | -64.9665 | 53.710 | 2 | 11.0 | 242.0 | 0.410 | 0.20 | 8 | 186 |  |
| **9710** | 8 | -31.2599 | 117.6740 | 7.210 | 0 | 35.0 | 79.0 | 1.703 | 0.87 | 11 | 15 |  |
| **9711** | 8 | -31.1853 | 117.6124 | 10.069 | 0 | 25.0 | 92.0 | 1.766 | 0.73 | 11 | 15 |  |
| **9712** | 8 | 2.1882 | 126.6833 | 44.615 | 0 | 44.0 | 112.0 | 1.564 | 0.86 | 11 | 74 |  |
| **9713** | 8 | 16.0523 | -94.9727 | 10.000 | 0 | 34.0 | 180.0 | 2.735 | 0.68 | 11 | 100 |  |

9714 rows × 13 columns

In [28]:



*#training our model and predicting the outcome*

X**=**data.iloc[:,:**-**2].values Y**=**data.iloc[:,**-**2].values

In [29]:

Y

Out[29]: array([1, 1, 1, ..., 1, 1, 1])

In [30]:



*#feature engineering #feature selection*

**from** sklearn.feature\_selection **import** SelectKBest,f\_classif selector**=**SelectKBest(f\_classif,k**=**11)

X\_new**=**selector.fit\_transform(X,Y) print(X\_new)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [[ | 3. | -6.2159 | 146.9556 ... 0.57 | 11. | 137. | ] |
| [ | 3. | 64.6558 | -17.7271 ... 1.08 | 11. | 68. | ] |
| [ | 3. | 33.6817 | 93.184 ... 0.78 | 11. | 36. | ] |
| ... |  |  |  |  |  |  |
| [ | 8. | -31.1853 | 117.6124 ... 0.73 | 11. | 15. | ] |
| [ | 8. | 2.1882 | 126.6833 ... 0.86 | 11. | 74. | ] |
| [ | 8. | 16.0523 | -94.9727 ... 0.68 | 11. | 100. | ]] |

In [31]:



*#Feature scaling*

scaler**=**StandardScaler()

scaled\_features**=**scaler.fit\_transform(X\_new) scaled\_features

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Out[31]: | array([[-1.65444944, | | -0.79825532, | 1.36880794, | ..., | -0.02499242, |
|  | 0.51854647, | | 0.65863436], |  |  |  |
|  | [-1.65444944, | | 1.55501283, | 0.08922111, | ..., | 1.75592468, |
|  | 0.51854647, | | -0.32900333], |  |  |  |
|  | [-1.65444944, | | 0.52652949, | 0.95100187, | ..., | 0.70832639, |
|  | 0.51854647, | | -0.7870382 ], |  |  |  |
|  | ..., | |  |  |  |  |
| [ | | 1.84022784, | -1.62735485, | 1.14081087, | ..., | 0.53372667, |
|  | | 0.51854647, | -1.08762358], |  |  |  |
| [ | | 1.84022784, | -0.51920034, | 1.21129189, | ..., | 0.98768593, |
|  | | 0.51854647, | -0.24312179], |  |  |  |
| [ | | 1.84022784, | -0.05884811, -0.51097826, | | ..., | 0.35912695, |
|  | | 0.51854647, | 0.12903154]]) | |  |  |

In [32]:



*#Splitting the dataset*

X\_train,X\_test,Y\_train,Y\_test**=**train\_test\_split(scaled\_features,Y,test\_size**=**

*#CLASSIFICATION ALGORITHMS*

classifiers**=**{

'Logistic Regression':LogisticRegression(), 'Support Vector Machine ':SVC(),

'Decision Tree':DecisionTreeClassifier(), 'Random Forest':RandomForestClassifier(), 'Naive Bayes':GaussianNB(),

'K Nearest Neighbour':KNeighborsClassifier()

}

In [33]:



*#Training and evaluating classifiers*

**from** sklearn.metrics **import** confusion\_matrix,ConfusionMatrixDisplay results**=**{}

**for** name,clf **in** classifiers.items(): clf.fit(X\_train,Y\_train)

Y\_pred**=**clf.predict(X\_test)

cm**=**confusion\_matrix(Y\_test,Y\_pred)

print(f"Confusion matrix for {name} is \n",cm) accuracy**=**accuracy\_score(Y\_test,Y\_pred)

results[name]**=**accuracy

print(f"{name} has accuracy of {accuracy**\***100:.2f} ")

print(classification\_report(Y\_test,Y\_pred,zero\_division**=**1)) print("\n\n")

Confusion matrix for Logistic Regression is [[ 667 151]

[ 291 2129]]

Logistic Regression has accuracy of 86.35

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.70 | 0.82 | 0.75 | 818 |
| 1 | 0.93 | 0.88 | 0.91 | 2420 |
| accuracy |  |  | 0.86 | 3238 |
| macro avg | 0.82 | 0.85 | 0.83 | 3238 |
| weighted avg | 0.87 | 0.86 | 0.87 | 3238 |

Confusion matrix for Support Vector Machine is [[ 780 38]

[ 337 2083]]

Support Vector Machine has accuracy of 88.42

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.70 | 0.95 | 0.81 | 818 |
| 1 | 0.98 | 0.86 | 0.92 | 2420 |
| accuracy |  |  | 0.88 | 3238 |
| macro avg | 0.84 | 0.91 | 0.86 | 3238 |
| weighted avg | 0.91 | 0.88 | 0.89 | 3238 |

Confusion matrix for Decision Tree is [[ 608 210]

[ 216 2204]]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decision Tree | has accuracy  precision | of 86.84  recall | f1-score | support |
| 0 | 0.74 | 0.74 | 0.74 | 818 |
| 1 | 0.91 | 0.91 | 0.91 | 2420 |
| accuracy |  |  | 0.87 | 3238 |
| macro avg | 0.83 | 0.83 | 0.83 | 3238 |
| weighted avg | 0.87 | 0.87 | 0.87 | 3238 |

Confusion matrix for Random Forest is [[ 720 98]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [ 249 2171]]  Random Forest | has accuracy precision | of 89.28  recall | f1-score | support |
| 0 | 0.74 | 0.88 | 0.81 | 818 |
| 1 | 0.96 | 0.90 | 0.93 | 2420 |
| accuracy |  |  | 0.89 | 3238 |
| macro avg | 0.85 | 0.89 | 0.87 | 3238 |
| weighted avg | 0.90 | 0.89 | 0.90 | 3238 |

In [34]:



*#finding the best classifier*

best\_classifier**=**max(results,key**=**results.get)

print("best classifier is ",best\_classifier," with an accuracy of ",results

Confusion matrix for Naive Bayes is [[ 771 47]

[ 434 1986]]

|  |  |  |  |
| --- | --- | --- | --- |
| Naive Bayes has accuracy  precision | of 85.15  recall | f1-score | support |
| 0 0.64 | 0.94 | 0.76 | 818 |
| 1 0.98 | 0.82 | 0.89 | 2420 |
| accuracy |  | 0.85 | 3238 |
| macro avg 0.81 | 0.88 | 0.83 | 3238 |
| weighted avg 0.89 | 0.85 | 0.86 | 3238 |

Confusion matrix for K Nearest Neighbour is [[ 673 145]

[ 260 2160]]

|  |  |  |
| --- | --- | --- |
| K Nearest Neighbour has  precision | accuracy of 87.49  recall f1-score | support |
| 0 0.72 | 0.82 0.77 | 818 |
| 1 0.94 | 0.89 0.91 | 2420 |
| accuracy | 0.87 | 3238 |
| macro avg 0.83 | 0.86 0.84 | 3238 |
| weighted avg 0.88 | 0.87 0.88 | 3238 |

best classifier is Random Forest with an accuracy of 0.8928350833848054

In [35]:



fpr,tpr,thresholds**=**roc\_curve(Y\_test,Y\_pred)

*#calculating the auc*

roc\_auc**=**auc(fpr,tpr)

*#plot the roc curve*

plt.figure(figsize**=**(7,7))

plt.plot(fpr,tpr,color**=**'darkorange',lw**=**2,label**=**f'ROC curve(AUC area={roc\_au plt.plot([0,1],[0,1],color**=**'navy',lw**=**2,linestyle**=**'--')

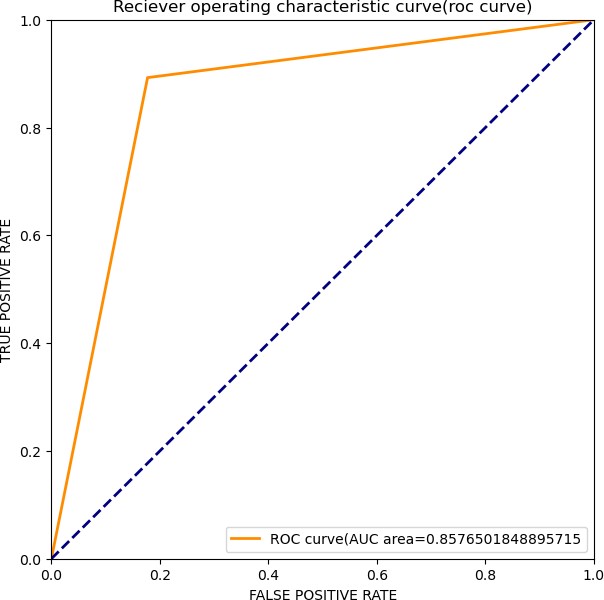
plt.xlim([0.0,1.0])

plt.ylim([0.0,1.0])

plt.xlabel('FALSE POSITIVE RATE') plt.ylabel('TRUE POSITIVE RATE')

plt.title('Reciever operating characteristic curve(roc curve)') plt.legend(loc**=**'best')

plt.show()



In [36]:



*#as our dataset is huge but we have made a simplified version of decision t*

**from** sklearn.tree **import** plot\_tree model **=** DecisionTreeClassifier(

max\_depth**=**5, *# Limit the depth of the tree*

min\_samples\_split**=**10, *# Minimum number of samples required to spli* min\_samples\_leaf**=**5, *# Minimum number of samples required to be a* random\_state**=**42

)

model.fit(X\_train, Y\_train)

*# Print the number of features and classes to verify*

print("Number of features in model:", model.n\_features\_in\_) print("Number of classes in model:", len(model.classes\_))

*# Plot the decision tree*

plt.figure(figsize**=**(20,10))

plot\_tree(model, feature\_names**=**['month','latitude','longitude','depth','mag plt.title("Simplified Decision Tree Visualization")

plt.show()

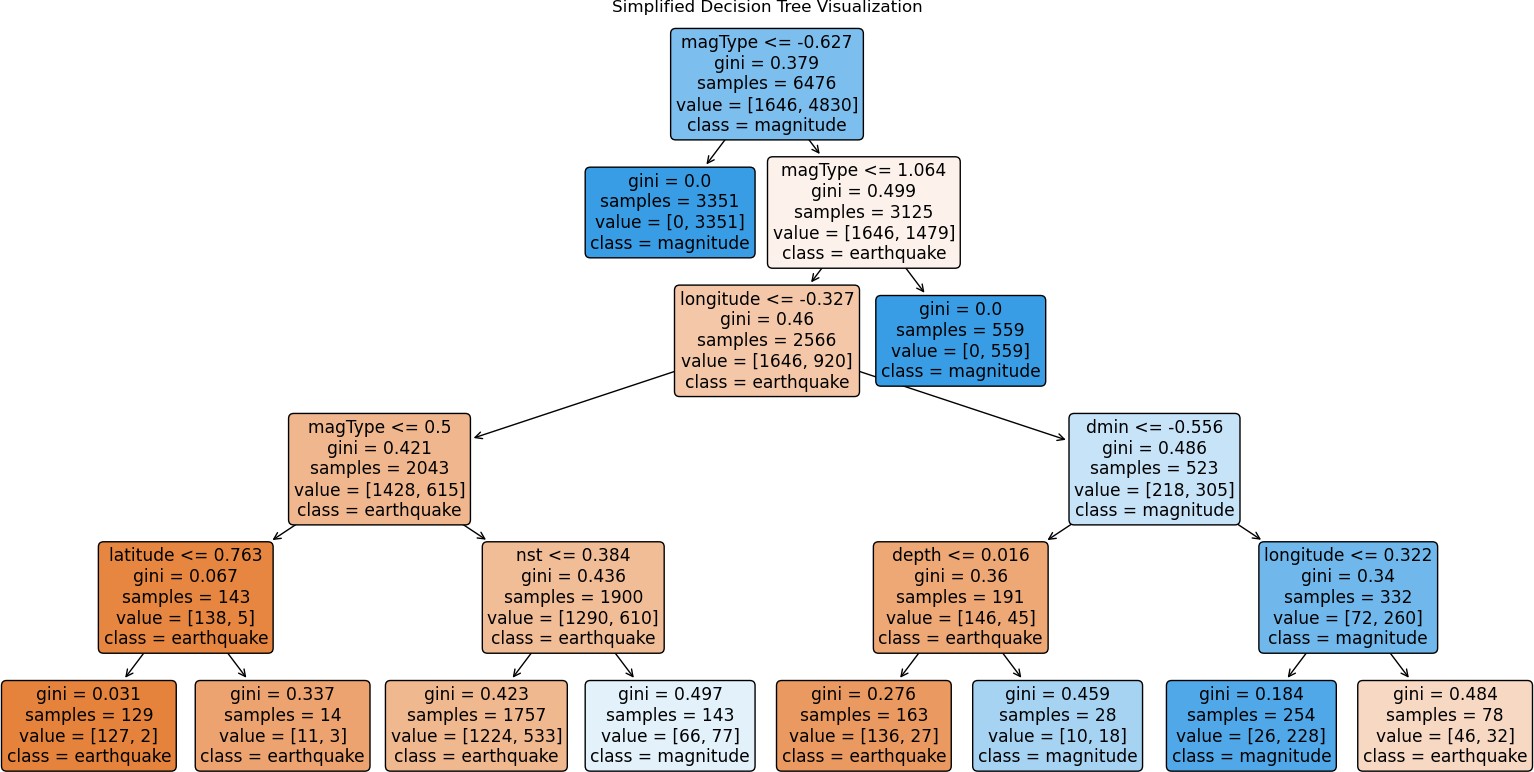
In [37]:



**import** pickle

**with** open("earthquakePrediction.pkl","wb") **as** file1: pickle.dump(best\_classifier,file1)

Number of features in model: 11 Number of classes in model: 2



In [ ]:

In [38]:



*#predicting the magnitude of earthquake*

x**=**data.iloc[:,:**-**1].values y**=**data.iloc[:,**-**1].values

x\_train,x\_test,y\_train,y\_test**=**train\_test\_split(x,y,test\_size**=**1**/**3,random\_sta

In [39]:



*#feature engineering #feature selection*

**from** sklearn.feature\_selection **import** SelectKBest,f\_classif selector1**=**SelectKBest(f\_classif,k**=**12)

x\_new**=**selector1.fit\_transform(x,y) print(x\_new)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [[ | 3. | -6.2159 | 146.9556 ... 11. | 137. | 1. | ] |
| [ | 3. | 64.6558 | -17.7271 ... 11. | 68. | 1. | ] |
| [ | 3. | 33.6817 | 93.184 ... 11. | 36. | 1. | ] |
| ... |  |  |  |  |  |  |
| [ | 8. | -31.1853 | 117.6124 ... 11. | 15. | 1. | ] |
| [ | 8. | 2.1882 | 126.6833 ... 11. | 74. | 1. | ] |
| [ | 8. | 16.0523 | -94.9727 ... 11. | 100. | 1. | ]] |

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\feature\_selection\\_univ ariate\_selection.py:113: RuntimeWarning: divide by zero encountered in div ide

f = msb / msw

In [40]:



*#Feature scaling*

scaler1**=**StandardScaler()

scaled\_features1**=**scaler.fit\_transform(x\_new) scaled\_features1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[40]: | array([[-1.65444944, | -0.79825532, | 1.36880794, | ..., | 0.51854647, |
|  | 0.65863436, | 0.5829769 ], |  |  |  |
|  | [-1.65444944, | 1.55501283, | 0.08922111, | ..., | 0.51854647, |
|  | -0.32900333, | 0.5829769 ], |  |  |  |
|  | [-1.65444944, | 0.52652949, | 0.95100187, | ..., | 0.51854647, |
|  | -0.7870382 , | 0.5829769 ], |  |  |  |
|  | ..., |  |  |  |  |
|  | [ 1.84022784, | -1.62735485, | 1.14081087, | ..., | 0.51854647, |
|  | -1.08762358, | 0.5829769 ], |  |  |  |
|  | [ 1.84022784, | -0.51920034, | 1.21129189, | ..., | 0.51854647, |
|  | -0.24312179, | 0.5829769 ], |  |  |  |
| [ 1.84022784, | | -0.05884811, -0.51097826, | | ..., | 0.51854647, |
| 0.12903154, | | 0.5829769 ]]) | |  |  |

In [41]:



*#Splitting the dataset*

x\_train,x\_test,y\_train,y\_test**=**train\_test\_split(x,y,test\_size**=**1**/**3,random\_sta

In [42]:



*#here multilinear regression is used.*

regressor**=**LinearRegression()

In [43]:

regressor.fit(x\_train,y\_train)

Out[43]: LinearRegression()

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**

**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [44]:



y\_predictAll**=**regressor.predict(x) y\_predictAll

Out[44]: array([4.83855039, 4.81685316, 4.1326252 , ..., 4.59060059, 4.52350276,

4.04888751])

In [45]:



*#finding the regression coefficients*

regressor.coef\_

Out[45]: array([ 3.18811908e-03, -7.50369430e-03, 8.50414311e-04, -2.68181482e-04, 1.54327518e-03, 6.52682306e-03, -8.06352869e-04, 1.30715081e-02,

5.12739137e-01, 3.32658213e-02, 2.19984185e-04, 7.93414443e-0

1])

In [46]:

*#finding the intercepts*

regressor.intercept\_

Out[46]: 2.5847635202107133

In [47]:



*#model evaluation*

*#printing the r squared*

print("r squared ",r2\_score(y,y\_predictAll)**\***100)

r squared 87.06575124792693

In [48]:



*#printing the mean squared error*

print("mean squared error is ",mean\_squared\_error(y,y\_predictAll))

mean squared error is 0.09580773489963246

In [49]:

regressor.score(x,y)**\***100

Out[49]: 87.06575124792693

In [50]:



regressor.score(x\_train,y\_train)**\***100

Out[50]: 87.18469261285573

In [51]:



regressor.score(x\_test,y\_test)**\***100

Out[51]: 86.82240448046048

In [52]:



*#as our model gives a score of 86.82 % score on the testing data which is v*

In [53]:



**with** open("magnitudePrediction.pkl","wb") **as** file2: pickle.dump(regressor,file2)

In [ ]:

In [ ]:

In [ ]: