zlxuzgl61

October 24, 2023

0.1 EARTHQUAKE PREDICTION MODEL USING PYTHON

---

1 *Uploading_dataset*

In data analysis and machine learning, working with datasets is a fundamental task. To get started, we need to upload our dataset into our Python environment. we have uploaded a dataset using the popular pandas library. Pandas simplifies the process of working with structured data, making it an ideal choice for handling datasets in Python.

[67]:
```
!pip install pandas
import pandas as pd
dataset = pd.read_csv('bronze.csv')
print(dataset.head())    # Display the first few rows
print(dataset.info())    # Display information about columns and data types
print(dataset.describe()) # Display summary statistics
print(dataset)
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-
packages (from pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
                       time  latitude  longitude  depth  mag magType  nst  \
0  1930-12-08T08:01:02.000Z    23.261    120.277   15.0  6.3      mw  NaN
1  1930-12-03T18:51:47.000Z    18.233     96.298   10.0  7.4      mw  NaN
2  1930-12-02T07:01:30.000Z    25.854     98.356   35.0  6.2      mw  NaN
3  1930-11-28T07:32:56.000Z    18.779   -106.767   15.0  6.3      mw  NaN
4  1930-11-25T19:02:53.000Z    35.050    139.129   15.0  6.9      mw  NaN

   gap  dmin  rms  …                   updated place type  horizontalError  \
0  NaN   NaN  NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN              NaN
```

```
1  NaN   NaN  NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN              NaN
2  NaN   NaN  NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN              NaN
3  NaN   NaN  NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN              NaN
4  NaN   NaN  NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN              NaN

   depthError  magError  magNst  status  locationSource  magSource
0         NaN       NaN     NaN     NaN             NaN        NaN
1         NaN       NaN     NaN     NaN             NaN        NaN
2         NaN       NaN     NaN     NaN             NaN        NaN
3         NaN       NaN     NaN     NaN             NaN        NaN
4         NaN       NaN     NaN     NaN             NaN        NaN

[5 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 797046 entries, 0 to 797045
Data columns (total 22 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   time             797046 non-null  object
 1   latitude         797046 non-null  float64
 2   longitude        797046 non-null  float64
 3   depth            797041 non-null  float64
 4   mag              797046 non-null  float64
 5   magType          796940 non-null  object
 6   nst              421658 non-null  float64
 7   gap              470629 non-null  float64
 8   dmin             202838 non-null  float64
 9   rms              614095 non-null  float64
 10  net              797046 non-null  object
 11  id               797046 non-null  object
 12  updated          797046 non-null  object
 13  place            0 non-null       float64
 14  type             0 non-null       float64
 15  horizontalError  0 non-null       float64
 16  depthError       0 non-null       float64
 17  magError         0 non-null       float64
 18  magNst           0 non-null       float64
 19  status           0 non-null       float64
 20  locationSource   0 non-null       float64
 21  magSource        0 non-null       float64
dtypes: float64(17), object(5)
memory usage: 133.8+ MB
None
           latitude      longitude          depth            mag  \
count  797046.000000  797046.000000  797041.000000  797046.000000
mean       19.038314     -12.911692      53.726903       3.884296
std        29.219884     118.010192      99.510254       0.911611
min       -84.422000    -179.999000      -4.900000       2.500000
```

25%	-4.727000	-118.086000	10.000000	3.000000
50%	29.826000	-52.385000	23.000000	4.000000
75%	39.405000	120.892000	45.620000	4.600000
max	87.221000	180.000000	735.800000	9.500000

	nst	gap	dmin	rms	place	\
count	421658.000000	470629.000000	202838.000000	614095.000000	0.0	
mean	33.170883	141.002356	1.757801	0.763249	NaN	
std	55.796692	81.447357	3.734481	0.477974	NaN	
min	0.000000	0.000000	0.000000	0.000000	NaN	
25%	8.000000	75.000000	0.082880	0.420000	NaN	
50%	16.000000	125.000000	0.583000	0.810000	NaN	
75%	35.000000	195.900000	1.973000	1.070000	NaN	
max	934.000000	360.000000	127.420000	69.320000	NaN	

	type	horizontalError	depthError	magError	magNst	status	\
count	0.0	0.0	0.0	0.0	0.0	0.0	
mean	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	

	locationSource	magSource
count	0.0	0.0
mean	NaN	NaN
std	NaN	NaN
min	NaN	NaN
25%	NaN	NaN
50%	NaN	NaN
75%	NaN	NaN
max	NaN	NaN

	time	latitude	longitude	depth	mag	magType	\
0	1930-12-08T08:01:02.000Z	23.2610	120.2770	15.00	6.3	mw	
1	1930-12-03T18:51:47.000Z	18.2330	96.2980	10.00	7.4	mw	
2	1930-12-02T07:01:30.000Z	25.8540	98.3560	35.00	6.2	mw	
3	1930-11-28T07:32:56.000Z	18.7790	-106.7670	15.00	6.3	mw	
4	1930-11-25T19:02:53.000Z	35.0500	139.1290	15.00	6.9	mw	
…	…	…	…	…	…	…	
797041	2018-09-01T01:14:38.230Z	-30.4830	-177.9279	43.90	4.3	mb	
797042	2018-09-01T01:07:59.120Z	-10.7558	124.3621	10.00	4.0	mb	
797043	2018-09-01T01:00:13.810Z	-5.5167	147.1735	217.56	4.6	mb	
797044	2018-09-01T00:27:11.440Z	46.8819	155.6566	10.00	4.3	mb	
797045	2018-09-01T00:00:47.980Z	-55.7508	-28.3561	10.00	4.8	mb	

	nst	gap	dmin	rms	…	updated	place	type	\

```
0       NaN    NaN    NaN   NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN
1       NaN    NaN    NaN   NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN
2       NaN    NaN    NaN   NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN
3       NaN    NaN    NaN   NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN
4       NaN    NaN    NaN   NaN  …  2015-05-13T18:52:43.000Z   NaN  NaN
…     …    …    …    …  …                          …   …  …
797041  NaN  165.0  1.233  0.87  …  2018-11-07T18:37:12.040Z   NaN  NaN
797042  NaN  112.0  0.998  1.23  …  2018-11-07T18:37:12.040Z   NaN  NaN
797043  NaN  119.0  3.455  0.71  …  2018-11-07T18:37:07.040Z   NaN  NaN
797044  NaN   94.0  6.370  1.21  …  2018-11-07T18:37:12.040Z   NaN  NaN
797045  NaN   78.0  4.905  1.31  …  2018-11-07T18:37:07.040Z   NaN  NaN

        horizontalError  depthError  magError  magNst  status  locationSource  \
0                   NaN         NaN       NaN     NaN     NaN             NaN
1                   NaN         NaN       NaN     NaN     NaN             NaN
2                   NaN         NaN       NaN     NaN     NaN             NaN
3                   NaN         NaN       NaN     NaN     NaN             NaN
4                   NaN         NaN       NaN     NaN     NaN             NaN
…                 …         …       …     …     …             …
797041              NaN         NaN       NaN     NaN     NaN             NaN
797042              NaN         NaN       NaN     NaN     NaN             NaN
797043              NaN         NaN       NaN     NaN     NaN             NaN
797044              NaN         NaN       NaN     NaN     NaN             NaN
797045              NaN         NaN       NaN     NaN     NaN             NaN

        magSource
0             NaN
1             NaN
2             NaN
3             NaN
4             NaN
…           …
797041        NaN
797042        NaN
797043        NaN
797044        NaN
797045        NaN

[797046 rows x 22 columns]
```

2 *Preprocessing*

Before extracting meaningful insights or building accurate machine learning models, we must preprocess our dataset. Data preprocessing involves a series of steps to clean, transform, and structure your data for analysis.

[68]:

```
print(dataset.isnull().sum())








# Specify the columns you want to delete (e.g., 'column1', 'column2')
columns_to_delete =␣
 ↪['place','type','horizontalError','depthError','magError','magNst','status','locationSource

# Use the drop method to delete the specified columns
dataset.drop(columns=columns_to_delete, inplace=True, errors='ignore')

# The specified columns are deleted from the dataset.
print(dataset)
```

```
time                     0
latitude                 0
longitude                0
depth                    5
mag                      0
magType                106
nst                 375388
gap                 326417
dmin                594208
rms                 182951
net                      0
id                       0
updated                  0
place               797046
type                797046
horizontalError     797046
depthError          797046
magError            797046
magNst              797046
status              797046
locationSource      797046
magSource           797046
dtype: int64
                            time  latitude  longitude   depth  mag magType  \
0       1930-12-08T08:01:02.000Z   23.2610   120.2770   15.00  6.3      mw
1       1930-12-03T18:51:47.000Z   18.2330    96.2980   10.00  7.4      mw
2       1930-12-02T07:01:30.000Z   25.8540    98.3560   35.00  6.2      mw
3       1930-11-28T07:32:56.000Z   18.7790  -106.7670   15.00  6.3      mw
4       1930-11-25T19:02:53.000Z   35.0500   139.1290   15.00  6.9      mw
…                            …         …         …       …   …       …
```

```
797041  2018-09-01T01:14:38.230Z  -30.4830  -177.9279   43.90  4.3       mb
797042  2018-09-01T01:07:59.120Z  -10.7558   124.3621   10.00  4.0       mb
797043  2018-09-01T01:00:13.810Z   -5.5167   147.1735  217.56  4.6       mb
797044  2018-09-01T00:27:11.440Z   46.8819   155.6566   10.00  4.3       mb
797045  2018-09-01T00:00:47.980Z  -55.7508   -28.3561   10.00  4.8       mb

        nst    gap   dmin   rms     net           id  \
0       NaN    NaN    NaN   NaN  iscgem  iscgem907791
1       NaN    NaN    NaN   NaN  iscgem  iscgem907777
2       NaN    NaN    NaN   NaN  iscgem  iscgem907773
3       NaN    NaN    NaN   NaN  iscgem  iscgem907769
4       NaN    NaN    NaN   NaN  iscgem  iscgem907761
…      …     …     …     …      …          …
797041  NaN  165.0  1.233  0.87      us    us2000hafq
797042  NaN  112.0  0.998  1.23      us    us2000hafw
797043  NaN  119.0  3.455  0.71      us    us2000h6as
797044  NaN   94.0  6.370  1.21      us    us2000hafn
797045  NaN   78.0  4.905  1.31      us    us2000h6aq

                         updated
0       2015-05-13T18:52:43.000Z
1       2015-05-13T18:52:43.000Z
2       2015-05-13T18:52:43.000Z
3       2015-05-13T18:52:43.000Z
4       2015-05-13T18:52:43.000Z
…                              …
797041  2018-11-07T18:37:12.040Z
797042  2018-11-07T18:37:12.040Z
797043  2018-11-07T18:37:07.040Z
797044  2018-11-07T18:37:12.040Z
797045  2018-11-07T18:37:07.040Z

[797046 rows x 13 columns]
```

3 *Data Visualization*

Data visualization is a fundamental part of data analysis. It helps us to understand our dataset, identify patterns, and communicate your findings to others. In this guide, we will explore how to create informative and visually appealing data visualizations using Python, with a focus on the Matplotlib and Seaborn libraries.

[69]:
```
!pip install matplotlib seaborn
import matplotlib.pyplot as plt
import seaborn as sns

print(dataset.head())
print(dataset.describe())
```

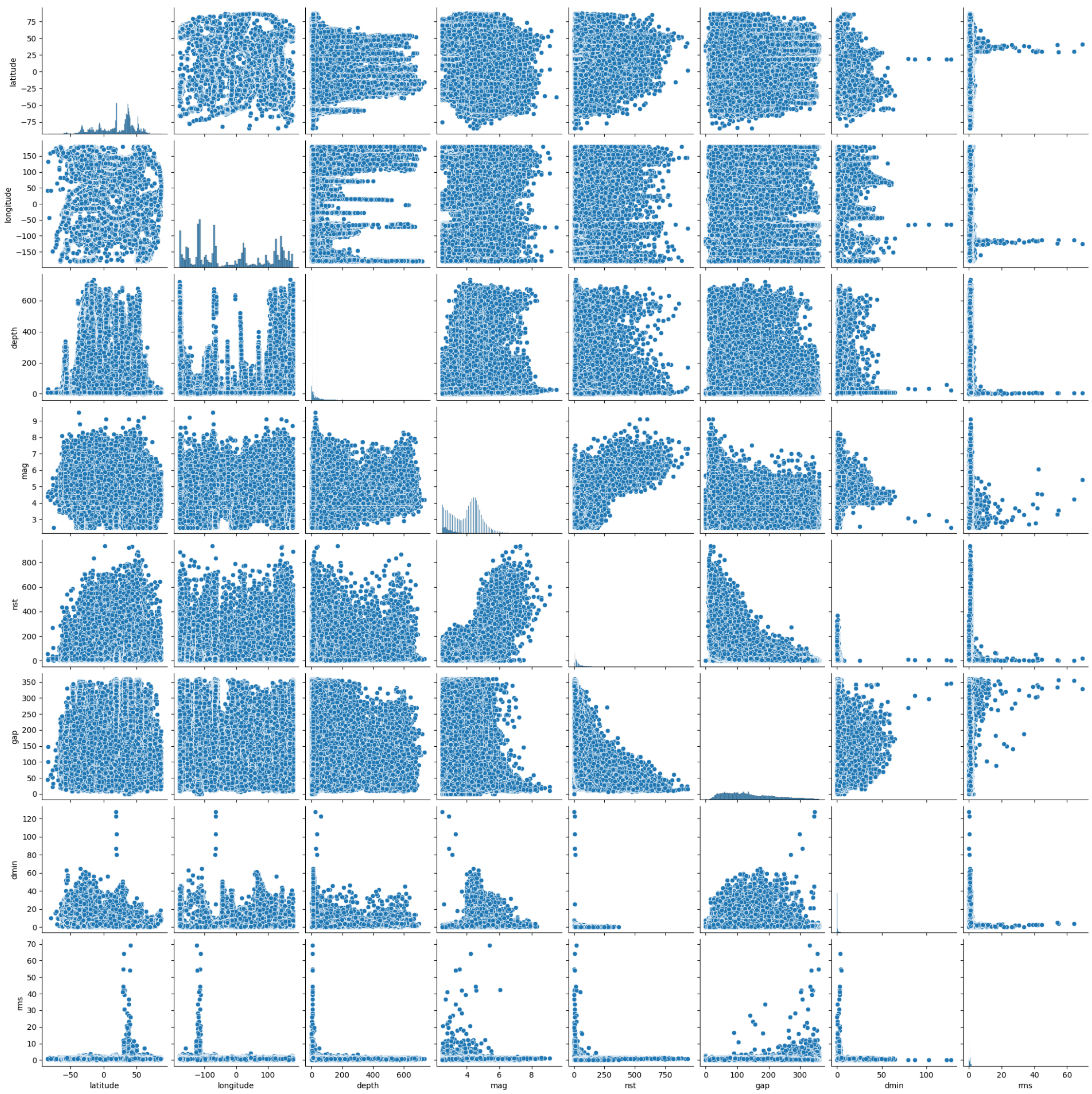
```python
print(dataset.dtypes)
sns.pairplot(dataset)
plt.show()
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-
packages (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-
packages (from seaborn) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.25->seaborn) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
                       time  latitude  longitude  depth  mag magType  nst  \
0  1930-12-08T08:01:02.000Z    23.261    120.277   15.0  6.3      mw  NaN
1  1930-12-03T18:51:47.000Z    18.233     96.298   10.0  7.4      mw  NaN
2  1930-12-02T07:01:30.000Z    25.854     98.356   35.0  6.2      mw  NaN
3  1930-11-28T07:32:56.000Z    18.779   -106.767   15.0  6.3      mw  NaN
4  1930-11-25T19:02:53.000Z    35.050    139.129   15.0  6.9      mw  NaN

   gap  dmin  rms     net            id                   updated
0  NaN   NaN  NaN  iscgem  iscgem907791  2015-05-13T18:52:43.000Z
1  NaN   NaN  NaN  iscgem  iscgem907777  2015-05-13T18:52:43.000Z
2  NaN   NaN  NaN  iscgem  iscgem907773  2015-05-13T18:52:43.000Z
3  NaN   NaN  NaN  iscgem  iscgem907769  2015-05-13T18:52:43.000Z
4  NaN   NaN  NaN  iscgem  iscgem907761  2015-05-13T18:52:43.000Z
            latitude      longitude          depth            mag  \
count  797046.000000  797046.000000  797041.000000  797046.000000
```

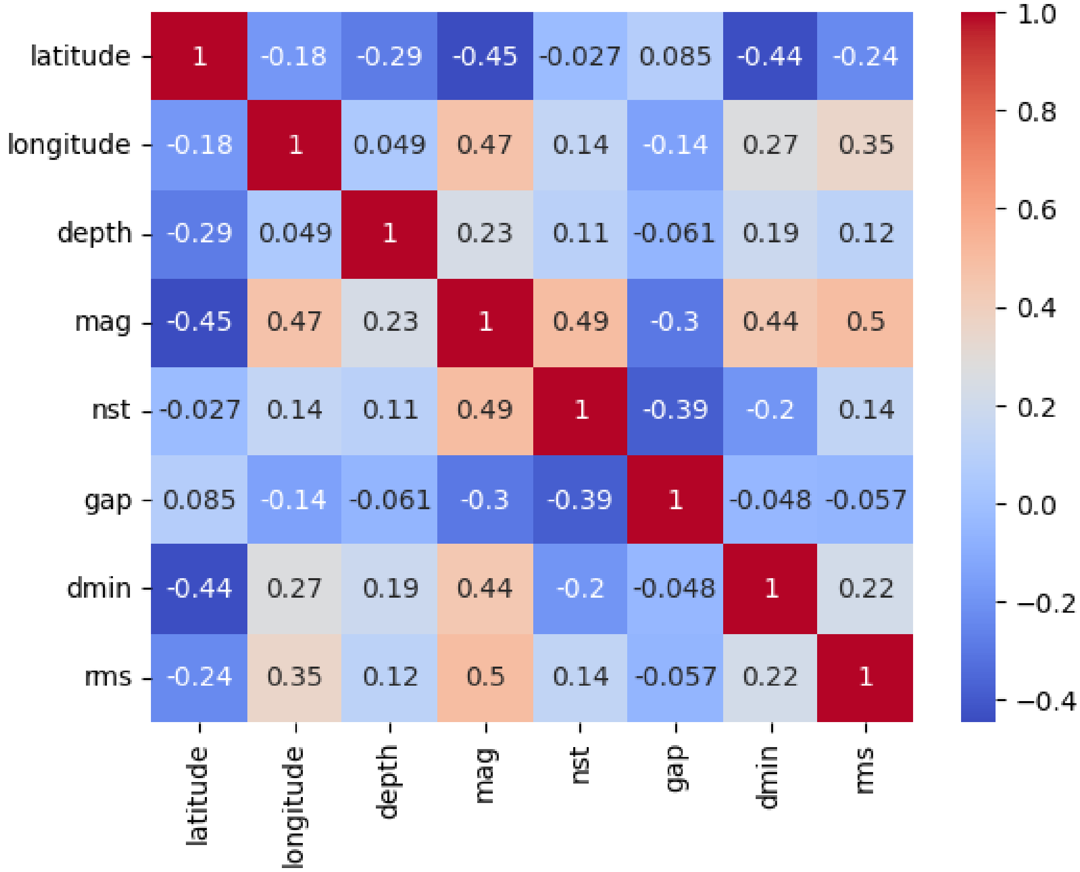
```
mean       19.038314    -12.911692     53.726903      3.884296
std        29.219884    118.010192     99.510254      0.911611
min       -84.422000   -179.999000     -4.900000      2.500000
25%        -4.727000   -118.086000     10.000000      3.000000
50%        29.826000    -52.385000     23.000000      4.000000
75%        39.405000    120.892000     45.620000      4.600000
max        87.221000    180.000000    735.800000      9.500000

                 nst           gap          dmin           rms
count  421658.000000  470629.000000  202838.000000  614095.000000
mean       33.170883     141.002356       1.757801       0.763249
std        55.796692      81.447357       3.734481       0.477974
min         0.000000       0.000000       0.000000       0.000000
25%         8.000000      75.000000       0.082880       0.420000
50%        16.000000     125.000000       0.583000       0.810000
75%        35.000000     195.900000       1.973000       1.070000
max       934.000000     360.000000     127.420000      69.320000
time           object
latitude      float64
longitude     float64
depth         float64
mag           float64
magType        object
nst           float64
gap           float64
dmin          float64
rms           float64
net            object
id             object
updated        object
dtype: object
```

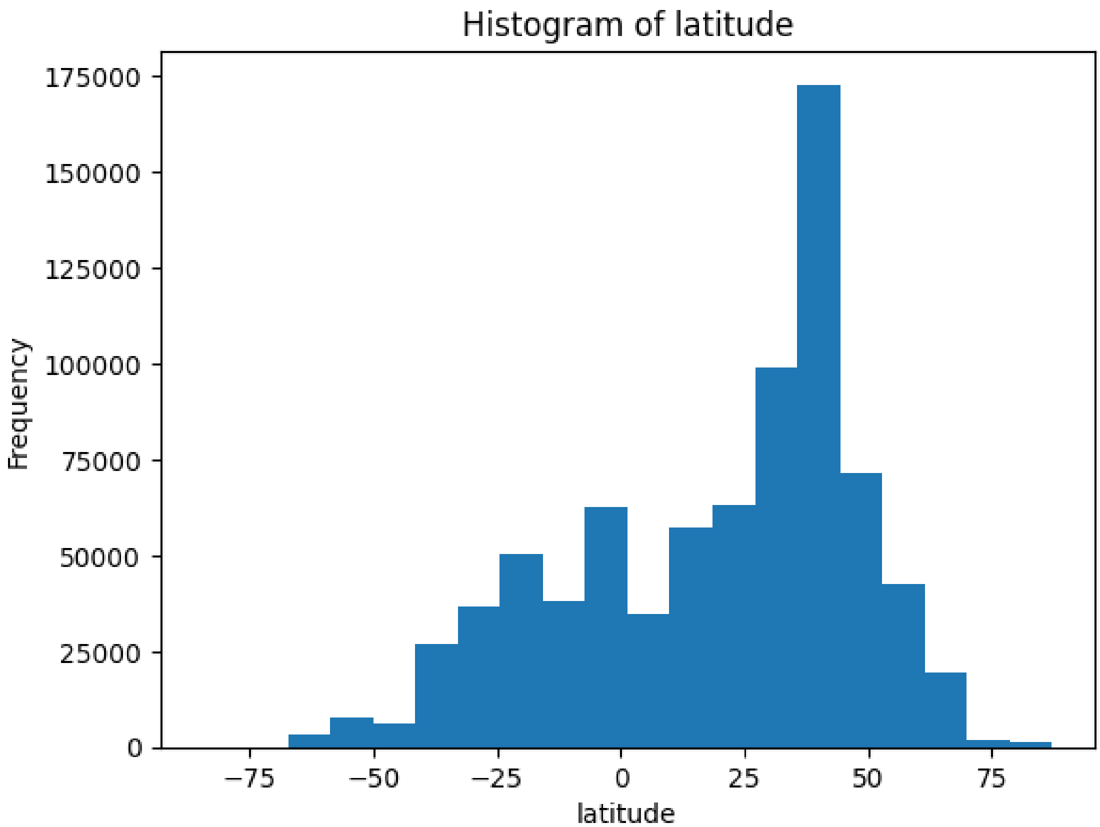


[70]:
```
sns.heatmap(dataset.corr(), annot=True, cmap="coolwarm")
plt.show()
```

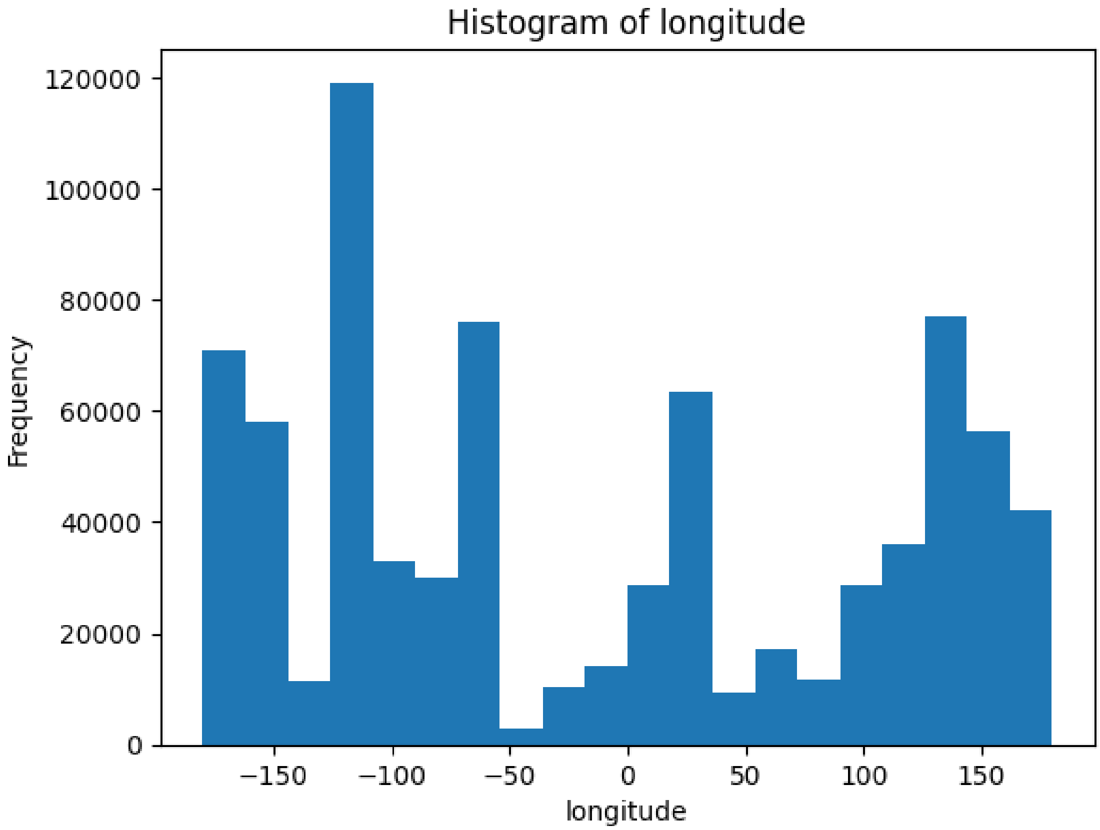
```
<ipython-input-70-26dfcbc1272d>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(dataset.corr(), annot=True, cmap="coolwarm")
```



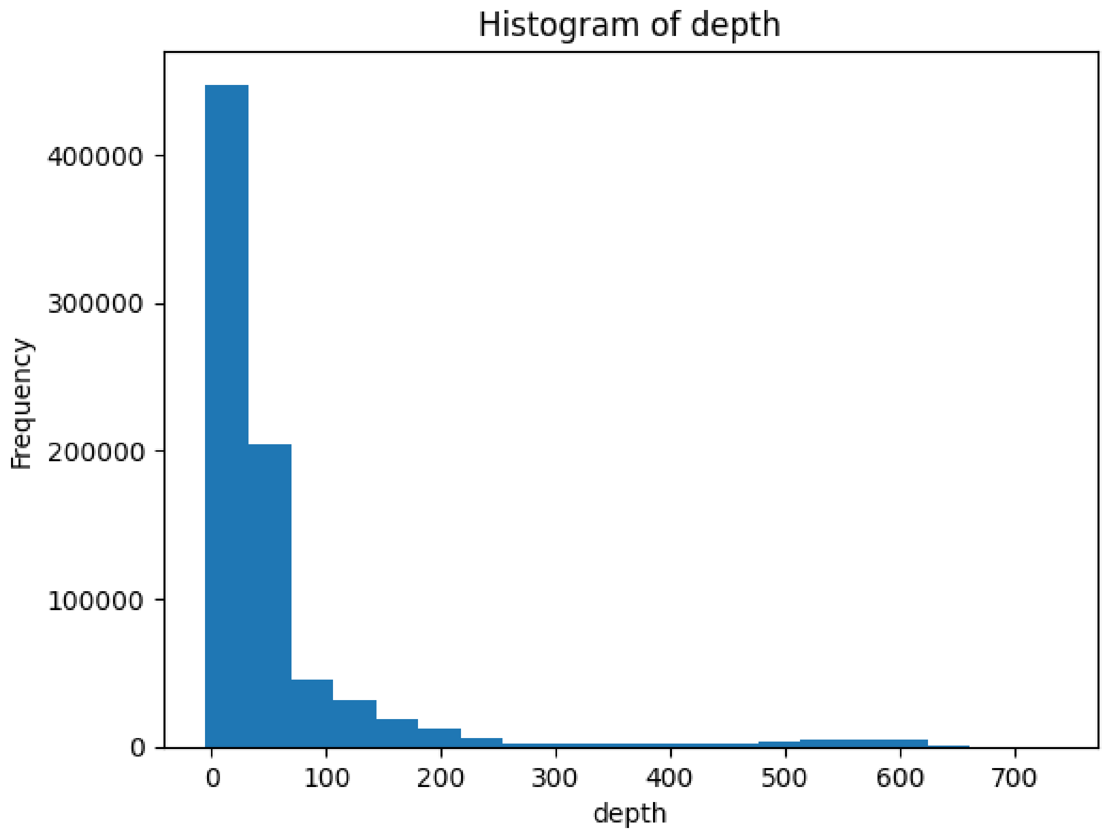

```
[71]: for column in dataset.select_dtypes(include=['int', 'float']):
          plt.hist(dataset[column], bins=20)
          plt.xlabel(column)
          plt.ylabel("Frequency")
          plt.title(f"Histogram of {column}")
          plt.show()
```

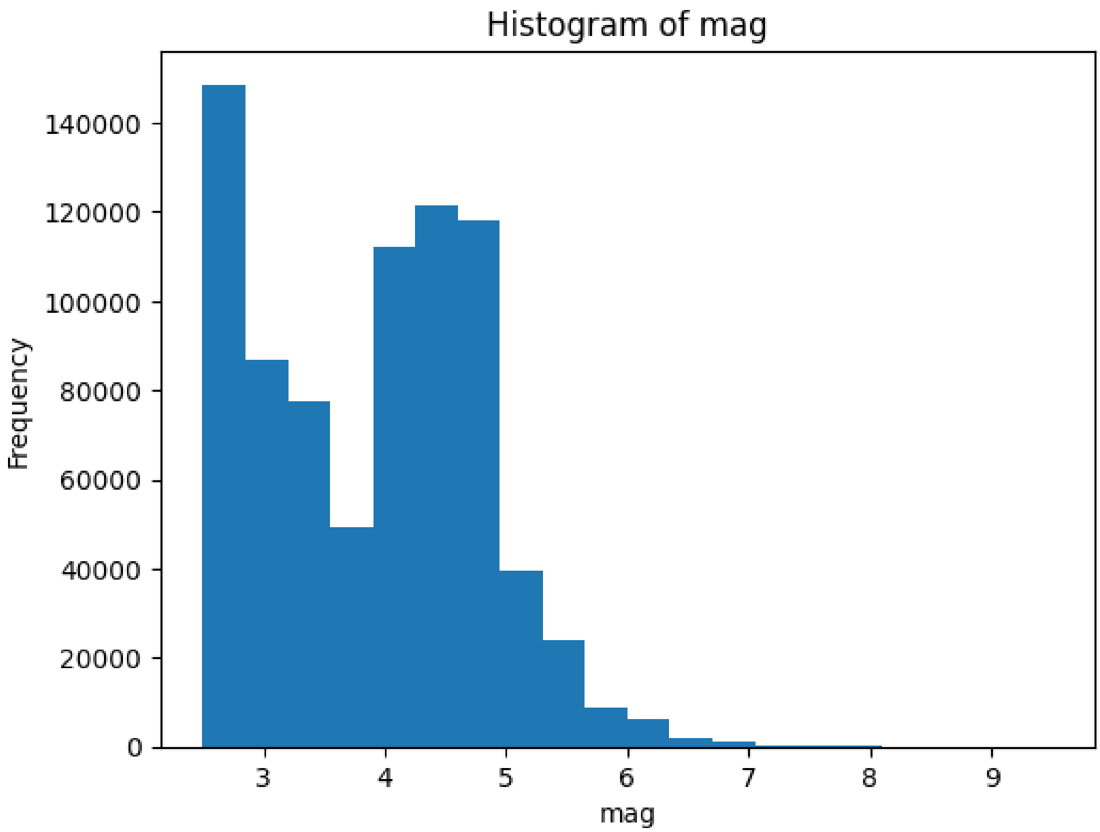


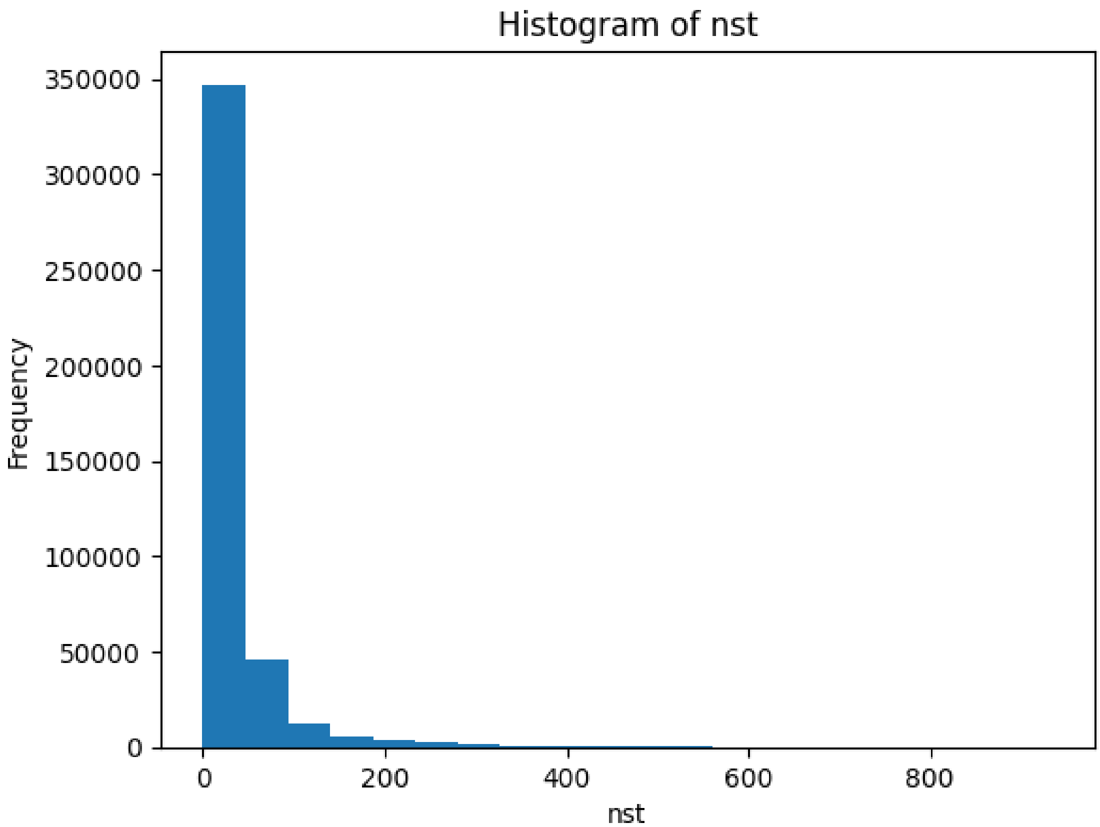
Histogram of latitude
Frequency
175000
150000
125000
100000
75000
50000
25000
0
−75
−50
−25
0
25
50
75
latitude

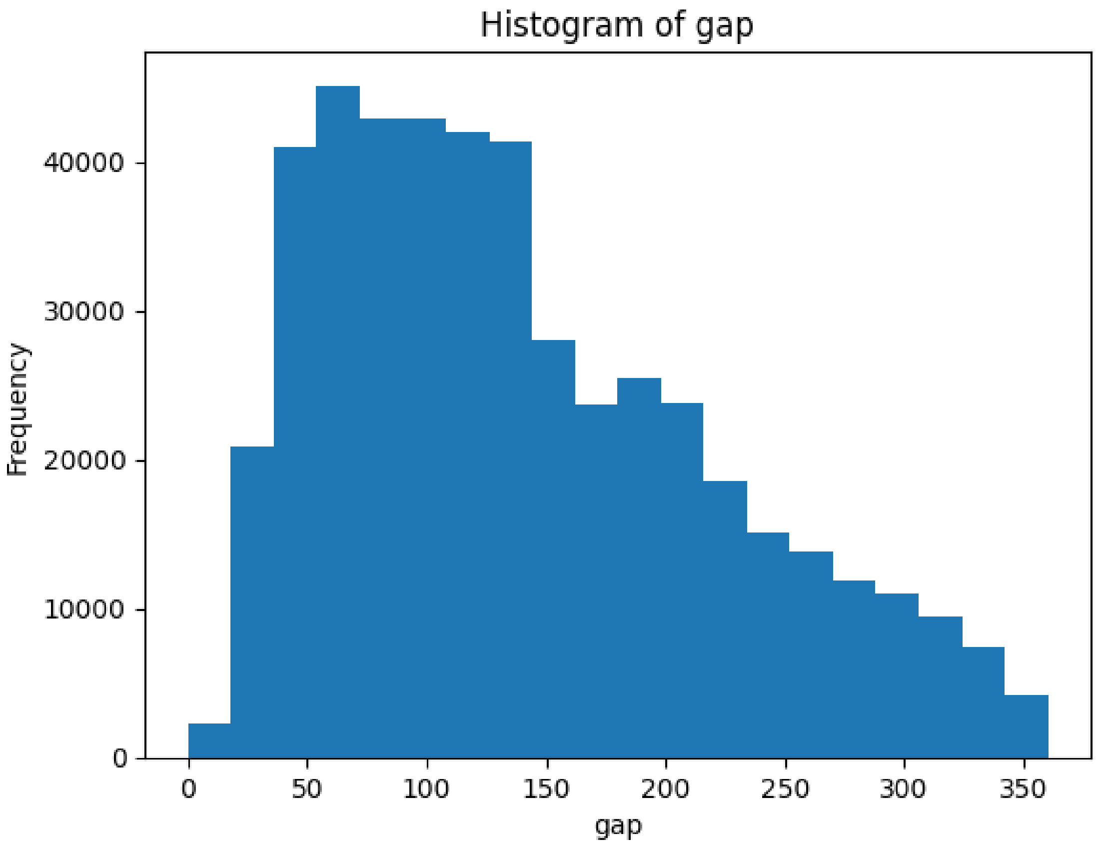


Histogram of longitude
120000
100000
80000
60000
40000
20000
0
Frequency
−150
−100
−50
0
50
100
150
longitude

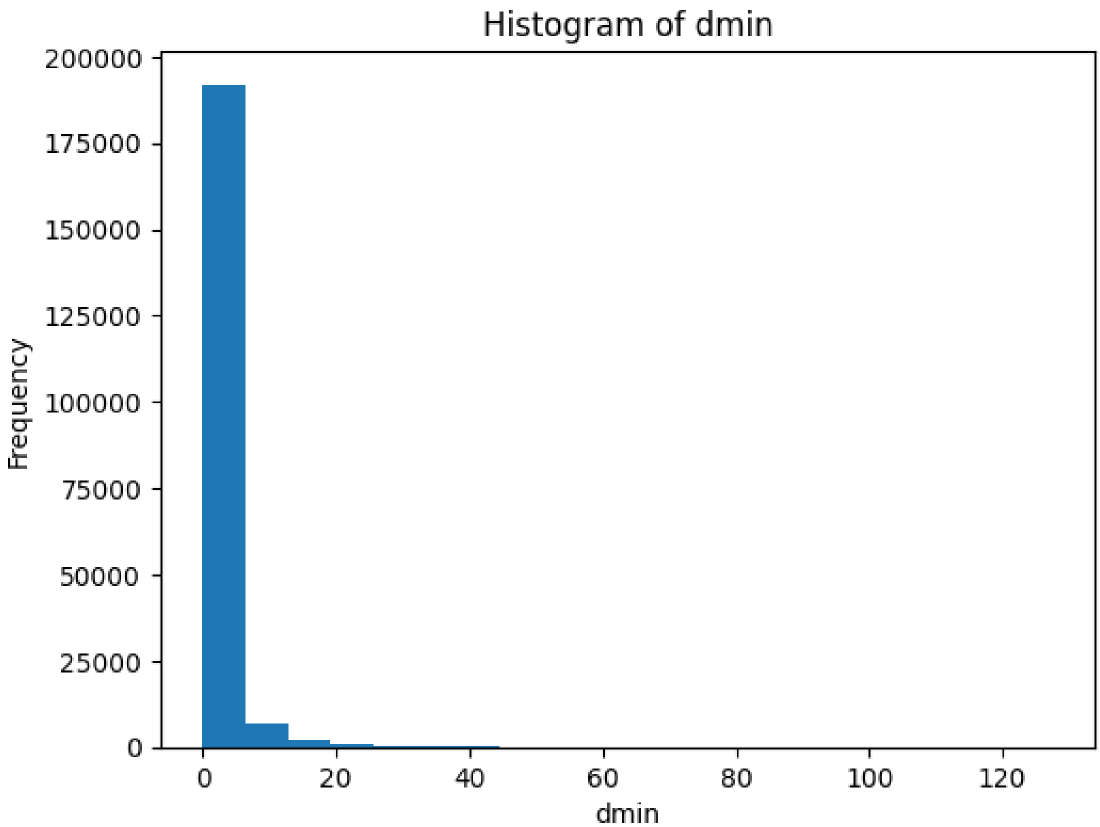

Histogram of depth
Frequency
400000
300000
200000
100000
0
0
100
200
300
400
500
600
700
depth


Histogram of mag
Frequency
0
20000
40000
60000
80000
100000
120000
140000
3
4
5
6
7
8
9
mag

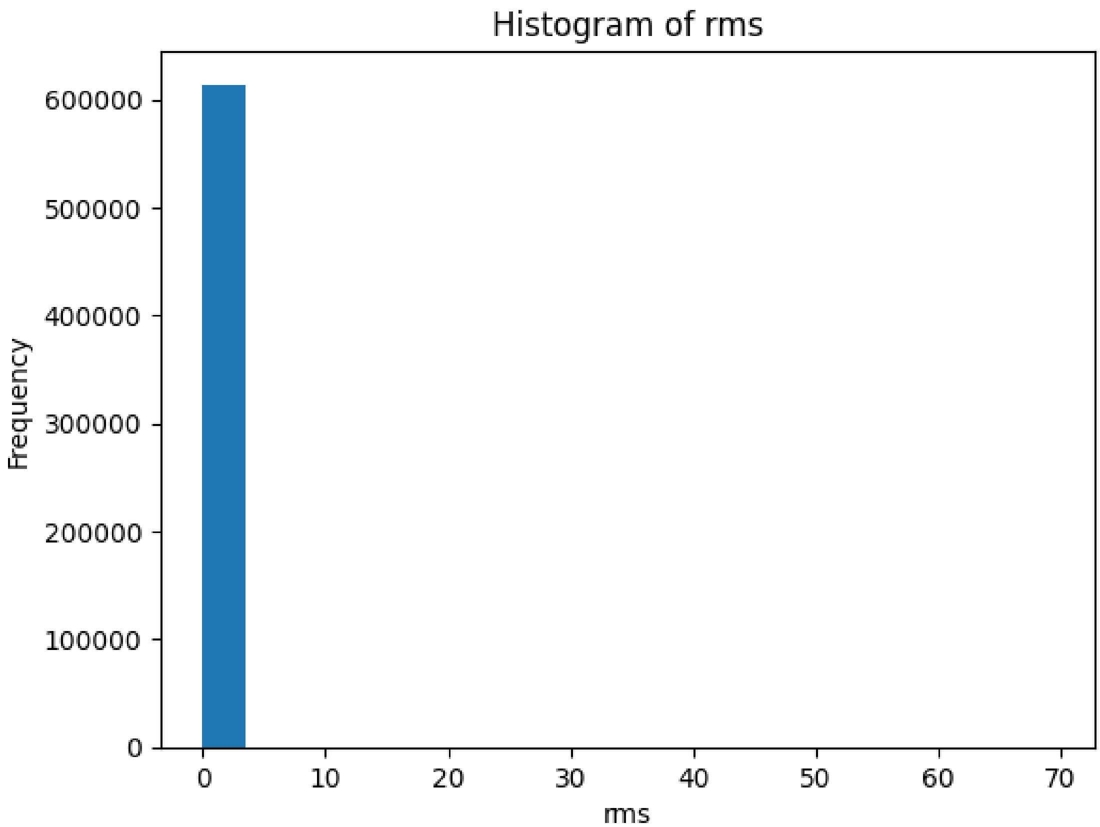

Histogram of nst
Frequency
350000
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800
nst



Histogram of gap
Frequency
gap
0
10000
20000
30000
40000
0
50
100
150
200
250
300
350



Histogram of dmin
Frequency
dmin
200000
175000
150000
125000
100000
75000
50000
25000
0
0
20
40
60
80
100
120



4 *Splitting as train and test data*

In the field of machine learning, building accurate and reliable models is essential. To achieve this, we need to carefully divide our dataset into two distinct parts: the training set and the test set. This process is known as train-test split, and it plays a fundamental role in model development and evaluation.

The Importance of Train-Test Split:

The primary goal of any machine learning model is to generalize well to unseen data. In other words, the model should not only perform excellently on the data it was trained on but also on new, unseen data. Train-test split helps us achieve this goal by providing a way to assess how well our model is likely to perform in real-world scenarios.

Understanding the Train and Test Sets:

Training Set: This is the portion of your dataset that you use to train your machine learning model. The model learns patterns and relationships within the training set, making it capable of making predictions.

Test Set: The test set is used to evaluate the model's performance. It's a separate portion of the dataset that the model has never seen during training. This helps you gauge how well the model generalizes to new, unseen data.

[74]:
```
from sklearn.model_selection import train_test_split
import numpy as np


dataset.
 ↪drop(columns=['time','magType','net','id','updated','nst','gap','dmin','rms'],␣
 ↪inplace=True, errors='ignore')
# Specify your features (X) and target variable (y)
dataset=dataset.dropna()
X = dataset.drop('mag', axis=1)  # Features (exclude the target column)
y = dataset['mag']  # Target variable


# Split the data into a training set (80%) and a test set (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print(X_train.head())


# The 'test_size' parameter controls the size of the test set, and␣
 ↪'random_state' sets a seed for randomization.
```

```
        latitude   longitude    depth
562370  -3.471000  100.588000   35.000
611385  31.998667 -115.158333    6.014
17499   15.573000  -92.536000  124.800
705970  33.812500  141.700100   10.000
76980   39.333000   20.372000   33.000
```

[75]:
```
X_train = np.asarray(X_train).astype(np.float32)
y_train = np.asarray(y_train).astype(np.float32)
```

5 *Model Development*

Neural networks, inspired by the structure of the human brain, have revolutionized the field of machine learning and artificial intelligence. These models, composed of interconnected artificial neurons, have shown remarkable performance in tasks like image recognition, natural language processing, and much more. In this guide, we will walk you through the process of developing neural network models.

Understanding Neural Networks:

1. **Neurons**: In a neural network, each neuron (or node) processes information and passes it to the next layer. Input features are fed into the input layer, and the network's output is derived from the final layer.

2. **Hidden Layers**: In addition to the input and output layers, neural networks often contain one or more hidden layers. These layers perform complex calculations, enabling the network to capture intricate patterns in the data.

Steps in Model Development:

1. **Data Preparation**:
   - Collect and preprocess your data. Ensure it's in a format suitable for training a neural network.
   - Split your dataset into training and testing sets for model evaluation.
2. **Model Architecture**:
   - Decide the structure of your neural network. This includes the number of layers, the number of neurons in each layer, and the activation functions.
   - Select the appropriate loss function for your specific task (e.g., mean squared error for regression or categorical cross-entropy for classification).
3. **Training**:
   - Use an optimizer (e.g., Adam or stochastic gradient descent) to minimize the loss and adjust the model's weights.
   - Define the number of training epochs and batch size. Experiment with these hyperparameters to achieve the best performance.
4. **Evaluation**:
   - Assess your model's performance using metrics such as accuracy, mean squared error, precision, recall, or F1 score, depending on your task.
   - Validate the model on a separate test set to gauge its ability to generalize to new data.
5. **Fine-tuning and Optimization**:
   - Based on the evaluation results, fine-tune your model. This may involve adjusting hyperparameters, adding regularization techniques, or altering the architecture.
6. **Deployment**:
   - Once satisfied with your model's performance, deploy it in a production environment. This can involve creating an API, integrating it into a web application, or using it in a real-time system.

Developing neural network models is an iterative and creative process. The performance of your model depends not only on the network's architecture but also on your data, preprocessing steps, and hyperparameter choices. Continuously experiment, evaluate, and refine your model to achieve the best possible results in your machine learning tasks.

[84]:
```
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split

# Load and preprocess your dataset
# Assuming you have features X and target variable y
# You can use the code for train-test split mentioned earlier

# Define your neural network architecture
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
```

```
    keras.layers.Dense(1)  # Adjust the number of units for your specific task
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')  # Choose an␣
 ↪appropriate loss function
print(X_train)
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)  # You can adjust the␣
 ↪number of epochs and batch size

# Evaluate the model on the test set
test_loss = model.evaluate(X_test, y_test)

# Make predictions
predictions = model.predict(X_test)
print(predictions)
```

```
[[  -3.471     100.588      35.      ]
 [  31.998667 -115.15833     6.014   ]
 [  15.573     -92.536     124.8     ]
 …
 [  20.325      95.03       33.      ]
 [  60.204    -152.4238     88.4     ]
 [  37.4945   -118.37067     3.186   ]]
Epoch 1/10
19926/19926 [==============================] - 43s 2ms/step - loss: 0.5437
Epoch 2/10
19926/19926 [==============================] - 46s 2ms/step - loss: 0.3592
Epoch 3/10
19926/19926 [==============================] - 46s 2ms/step - loss: 0.3403
Epoch 4/10
19926/19926 [==============================] - 54s 3ms/step - loss: 0.3344
Epoch 5/10
19926/19926 [==============================] - 43s 2ms/step - loss: 0.3306
Epoch 6/10
19926/19926 [==============================] - 38s 2ms/step - loss: 0.3281
Epoch 7/10
19926/19926 [==============================] - 37s 2ms/step - loss: 0.3264
Epoch 8/10
19926/19926 [==============================] - 38s 2ms/step - loss: 0.3247
Epoch 9/10
19926/19926 [==============================] - 39s 2ms/step - loss: 0.3266
Epoch 10/10
19926/19926 [==============================] - 37s 2ms/step - loss: 0.3247
4982/4982 [==============================] - 7s 1ms/step - loss: 0.3220
4982/4982 [==============================] - 8s 2ms/step
```

```
[[2.804029 ]
 [2.9881215]
 [3.091743 ]
 …
 [4.65536  ]
 [4.734812 ]
 [2.9705603]]
```

6 *Manual Prediction*

So, we've successfully trained a machine learning model, and now it's time to put it to work. In this guide, we'll walk you through the process of making manual predictions using your pre-developed model. Whether you're predicting real estate prices, classifying images, or solving complex problems, this guide will help you understand how to leverage your model for specific predictions.

Prerequisites:

Before you can make manual predictions, ensure that you have the following in place:

1. **Trained Model**: You should have a pre-trained machine learning model ready to use.
2. **Data Preprocessing**: Make sure that you've preprocessed your input data in the same way as you did during model training.

Steps for Manual Predictions:

1. **Load the Pre-trained Model**:
    - Start by loading the pre-trained model into your development environment. This might involve using libraries like `joblib` for scikit-learn models or `tf.keras.models.load_model` for TensorFlow/Keras models.
2. **Collect Input Data**:
    - Depending on your use case, you might collect input data through user input, a file, or an API call. This input data should be structured in the same way as the data used during model training.
3. **Preprocess the Input Data**:
    - Data preprocessing is often a crucial step in making predictions. Ensure that the input data is transformed and prepared in the same way it was before during model training. This may involve scaling, encoding, and feature engineering.
4. **Make Predictions**:
    - Pass the preprocessed input data to your pre-trained model using the `predict` method. The model will return a prediction based on the input features.
5. **Interpret the Prediction**:
    - Depending on your problem, the model may return a numerical value, a class label, or some other kind of output. Interpret the prediction according to the context of your application.

6. **Display the Result**:

    - Show the prediction to the user, store it in a database, or use it for further decision-making, depending on the application's requirements.

Making manual predictions with your pre-trained machine learning model allows you to harness the power of your hard work and bring it to real-world use cases. By following these steps and ensuring consistency with your preprocessing and model loading, you can leverage your model effectively for specific predictions in your applications.

[86]:
```python
# Collect user input (assumes a simple numerical feature)
latitude = float(input("Enter latitude: "))
longitude = float(input("Enter longitude: "))
depth = float(input("Enter depth: "))
    # Create a list or NumPy array with the input features
input_features = [latitude,longitude,depth]

    # Perform any necessary preprocessing on the input (e.g., scaling, encoding)

    # Make predictions
prediction = model.predict([input_features])




# Display the prediction
print("Predicted value: ", prediction)

if(prediction<6.0):
  print(" low to moderate earthquake occured")
else:
  print(" strong to catastrophic earthquake occured")
```

```
Enter latitude: 44
Enter longitude: -210
Enter depth: 124
1/1 [==============================] - 0s 54ms/step
Predicted value:  [[3.467468]]
 low to moderate earthquake occured
```