In [1]:
```
import numpy as np
import pandas as pd
```

Pythonic missing data

In [3]:
```
var=np.array([1,None,3,4])
var
```

Out[3]:
```
array([1, None, 3, 4], dtype=object)
```

Missing numerical data

In [9]:
```
var1=np.array([1,np.nan,3,4])
var1.dtype
```

Out[9]:
```
dtype('float64')
```

In [10]:
```
1+np.nan
```

Out[10]:
```
nan
```

In [11]:
```
0*np.nan
```

Out[11]:
```
nan
```

In [13]:
```
var1.sum(),var1.min(),var1.max()
```

Out[13]:
```
(nan, nan, nan)
```

In [15]:
```
np.nansum(var1),np.nanmin(var1),np.nanmax(var1)
```

Out[15]:
```
(8.0, 1.0, 4.0)
```

NaN and None in Pandas

In [16]:
```
pd.Series([1,np.nan,2,None])
```

Out[16]:
```
0    1.0
1    NaN
2    2.0
3    NaN
dtype: float64
```

In [19]:
```
x=pd.Series(range(2),dtype=int)
x
```

Out[19]:
```
0    0
1    1
dtype: int32
```

In [20]:
```
x[0]=None
x
```

Out[20]:
```
0    NaN
1    1.0
dtype: float64
```

In [21]:
```
data=pd.Series([1,np.nan,'hello',None])
```

In [22]:
```
data
```

Out[22]:
```
0        1
1      NaN
2    hello
3     None
dtype: object
```

In [23]:
```
data.isnull()
```

Out[23]:
```
0    False
1     True
2    False
3     True
dtype: bool
```

In [25]:
```
data[data.notnull()]
```

Out[25]:
```
0        1
2    hello
dtype: object
```

In [26]:
```
data.dropna()
```

Out[26]:
```
0        1
2    hello
dtype: object
```

In [28]:
```
df=pd.DataFrame([[1,np.nan,2],
                 [2,3,5],
                 [np.nan,4,6]])
df
```

Out[28]:

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	NaN	4.0	6

In [29]:
```
df.dropna()
```

Out[29]:

	0	1	2
1	2.0	3.0	5

In [30]:
```
df.dropna(axis='columns')
```

Out[30]:

	2
0	2
1	5
2	6

In [32]:
```
df[3]=np.nan
df
```

Out[32]:

	0	1	2	3
0	1.0	NaN	2	NaN
1	2.0	3.0	5	NaN
2	NaN	4.0	6	NaN

In [33]:
```
df.dropna(axis='columns',how='all')
```

Out[33]:

	0	1	2
0	1.0	NaN	2
1	2.0	3.0	5
2	NaN	4.0	6

In [34]:
```
df.dropna(axis='rows',)
```

Out[34]:

0	1	2	3

In [42]:
```
df.dropna(axis='rows',thresh=3)
```

Out[42]:

	0	1	2	3
1	2.0	3.0	5	NaN

In [43]:
```
data=pd.Series([1,np.nan,'hello',None],index=list('abcd'))
data
```

Out[43]:
```
a        1
b      NaN
c    hello
d     None
dtype: object
```

In [44]:
```
data.fillna(0)
```

Out[44]:
```
a        1
b        0
c    hello
d        0
dtype: object
```

In [45]:
```
data.fillna(method='ffill')
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_7644\1844443866.py:1: FutureWar
ning: Series.fillna with 'method' is deprecated and will raise in a future
version. Use obj.ffill() or obj.bfill() instead.
  data.fillna(method='ffill')
```

Out[45]:
```
a        1
b        1
c    hello
d    hello
dtype: object
```

In [46]:
```
data.fillna(method='bfill')
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_7644\1397359497.py:1: FutureWar
ning: Series.fillna with 'method' is deprecated and will raise in a future
version. Use obj.ffill() or obj.bfill() instead.
  data.fillna(method='bfill')
```

Out[46]:
```
a        1
b    hello
c    hello
d     None
dtype: object
```

In [47]:
```
df
```

Out[47]:

	0	1	2	3
0	1.0	NaN	2	NaN
1	2.0	3.0	5	NaN
2	NaN	4.0	6	NaN

In [48]:
```
df.fillna(method='ffill',axis=1)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_7644\2272395643.py:1: FutureWar
ning: DataFrame.fillna with 'method' is deprecated and will raise in a fut
ure version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill',axis=1)
```

Out[48]:

	0	1	2	3
0	1.0	1.0	2.0	2.0
1	2.0	3.0	5.0	5.0
2	NaN	4.0	6.0	6.0

import Boston Housing Price data-set

In [49]:
```
from sklearn.datasets import load_boston
```

In [51]:
```
boston=load_boston()
boston.data.shape
```

Out[51]:
```
(506, 13)
```

In [55]:
```
x=boston.data
y=boston.target
columns=boston.feature_names

# creater dataframes

boston_df=pd.DataFrame(boston.data)
boston_df.columns=columns
boston_df.head(7)
```

Out[55]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	

In [58]:
```
boston_df.columns
```

Out[58]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'T
AX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

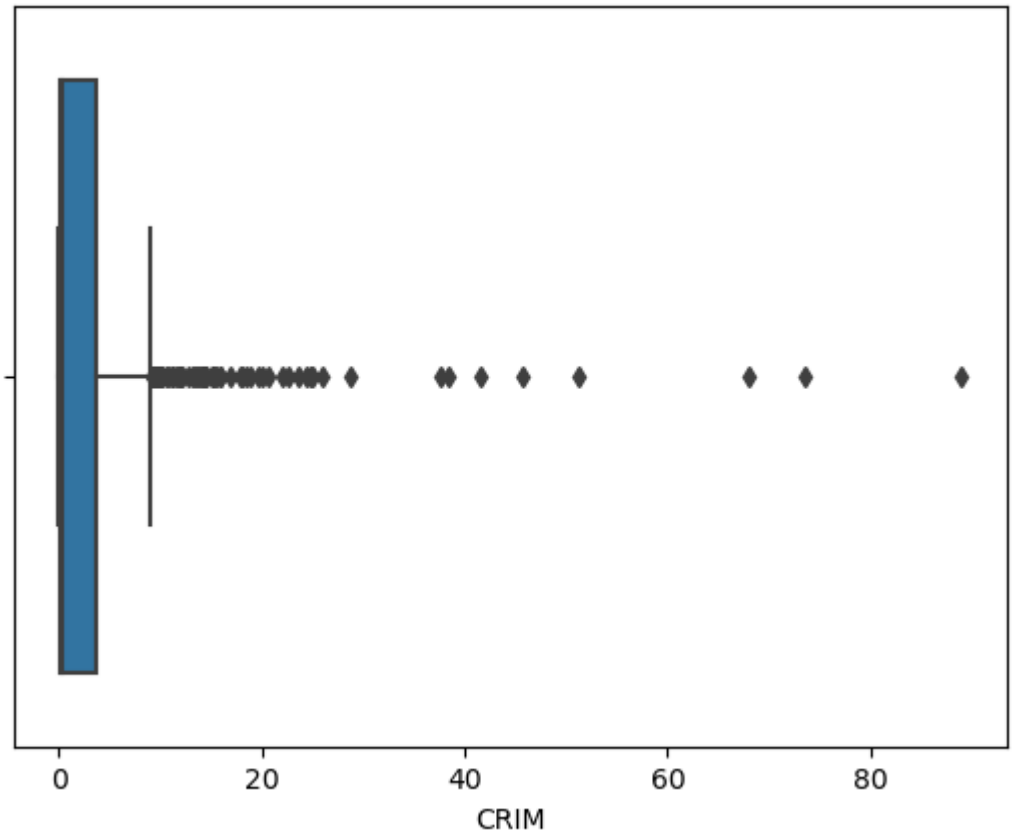
Method to detecd Outliers

1.Box Plot

In [59]:
```
import seaborn as sns
sns.boxplot(x=boston_df['CRIM'])
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\seaborn\_core.py:1225: FutureWar
ning: is_categorical_dtype is deprecated and will be removed in a future v
ersion. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

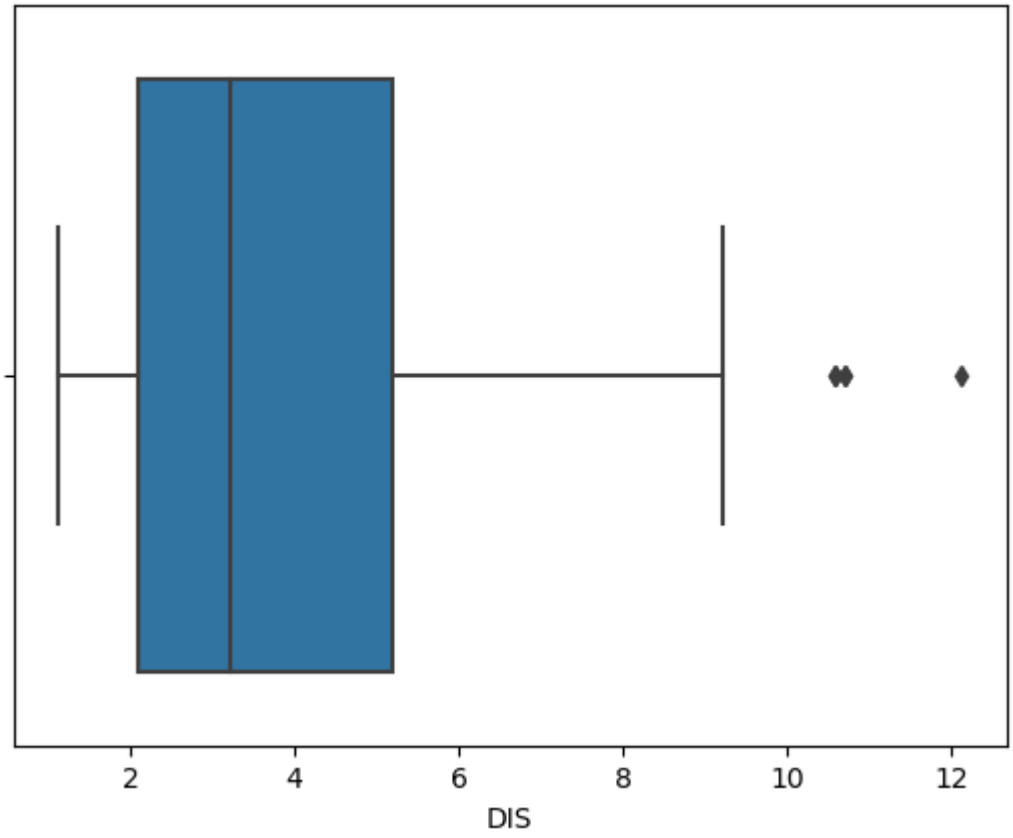
Out[59]: <AxesSubplot:xlabel='CRIM'>



In [60]:
```
sns.boxplot(x=boston_df['DIS'])
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\seaborn\_core.py:1225: FutureWar
ning: is_categorical_dtype is deprecated and will be removed in a future v
ersion. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

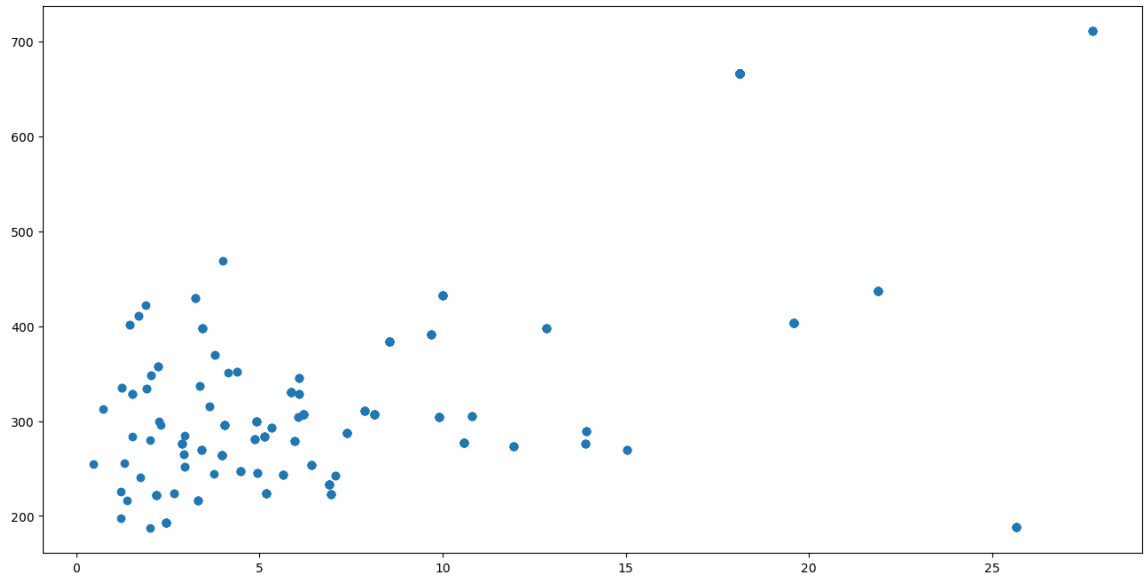
Out[60]:
```
<AxesSubplot:xlabel='DIS'>
```



2.Scatter Plot

In [62]:
```
import matplotlib.pyplot as plt

fig,ax=plt.subplots(figsize=(16,8))
ax.scatter(boston_df['INDUS'],boston_df['TAX'])
plt.show()
```



3.Z-score

In [63]:
```
from scipy import stats
import numpy as np

z=np.abs(stats.zscore(boston_df))
print(z)
```

```
         CRIM        ZN     INDUS      CHAS       NOX        RM       AGE
\
0    0.419782  0.284830  1.287909  0.272599  0.144217  0.413672  0.120013
1    0.417339  0.487722  0.593381  0.272599  0.740262  0.194274  0.367166
2    0.417342  0.487722  0.593381  0.272599  0.740262  1.282714  0.265812
3    0.416750  0.487722  1.306878  0.272599  0.835284  1.016303  0.809889
4    0.412482  0.487722  1.306878  0.272599  0.835284  1.228577  0.511180
..        ...       ...       ...       ...       ...       ...       ...
501  0.413229  0.487722  0.115738  0.272599  0.158124  0.439316  0.018673
502  0.415249  0.487722  0.115738  0.272599  0.158124  0.234548  0.288933
503  0.413447  0.487722  0.115738  0.272599  0.158124  0.984960  0.797449
504  0.407764  0.487722  0.115738  0.272599  0.158124  0.725672  0.736996
505  0.415000  0.487722  0.115738  0.272599  0.158124  0.362767  0.434732

          DIS       RAD       TAX   PTRATIO         B     LSTAT
0    0.140214  0.982843  0.666608  1.459000  0.441052  1.075562
1    0.557160  0.867883  0.987329  0.303094  0.441052  0.492439
2    0.557160  0.867883  0.987329  0.303094  0.396427  1.208727
3    1.077737  0.752922  1.106115  0.113032  0.416163  1.361517
4    1.077737  0.752922  1.106115  0.113032  0.441052  1.026501
..        ...       ...       ...       ...       ...       ...
501  0.625796  0.982843  0.803212  1.176466  0.387217  0.418147
502  0.716639  0.982843  0.803212  1.176466  0.441052  0.500850
503  0.773684  0.982843  0.803212  1.176466  0.441052  0.983048
504  0.668437  0.982843  0.803212  1.176466  0.403225  0.865302
505  0.613246  0.982843  0.803212  1.176466  0.441052  0.669058

[506 rows x 13 columns]
```

its difficult to find the outliers in above data so lets try any define a threshold to identify an outlier

In [64]:
```
threshold=3
print(np.where(z>3))
```

```
(array([ 55,  56,  57, 102, 141, 142, 152, 154, 155, 160, 162, 163, 199,
       200, 201, 202, 203, 204, 208, 209, 210, 211, 212, 216, 218, 219,
       220, 221, 222, 225, 234, 236, 256, 257, 262, 269, 273, 274, 276,
       277, 282, 283, 283, 284, 347, 351, 352, 353, 353, 354, 355, 356,
       357, 358, 363, 364, 364, 365, 367, 369, 370, 372, 373, 374, 374,
       380, 398, 404, 405, 406, 410, 410, 411, 412, 412, 414, 414, 415,
       416, 418, 418, 419, 423, 424, 425, 426, 427, 427, 429, 431, 436,
       437, 438, 445, 450, 454, 455, 456, 457, 466], dtype=int64), array([
1,  1,  1, 11, 12,  3,  3,  3,  3,  3,  3,  3,  1,  1,  1,  1,  1,
        1,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  5,  3,  3,  1,  5,
        5,  3,  3,  3,  3,  3,  3,  1,  3,  1,  1,  7,  7,  1,  7,  7,  7,
        3,  3,  3,  3,  3,  5,  5,  5,  3,  3,  3, 12,  5, 12,  0,  0,  0,
        0,  5,  0, 11, 11, 11, 12,  0, 12, 11, 11,  0, 11, 11, 11, 11, 11,
       11,  0, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11],
      dtype=int64))
```

In [ ]:
```
print(z[55][1])
```

4.IQR

In [70]:
```
Q1=boston_df.quantile(0.25)
Q3=boston_df.quantile(0.75)
IQR=Q3-Q1
print(IQR)
```

```
CRIM         3.595038
ZN          12.500000
INDUS       12.910000
CHAS         0.000000
NOX          0.175000
RM           0.738000
AGE         49.050000
DIS          3.088250
RAD         20.000000
TAX        387.000000
PTRATIO      2.800000
B           20.847500
LSTAT       10.005000
dtype: float64
```

Removing Outlier

1.Z-score

In [71]:
```
boston_df=boston_df[(z<3).all(axis=1)]
```

In [72]:
```
boston_df.shape
```

Out[72]:
```
(415, 13)
```

2.IQR

In [73]:
```
boston_df_out=boston_df[~((boston_df<(Q1-1.5*IQR))|(boston_df>(Q3+1.5*IQR))
```

In [75]:
```
boston_df_out.shape
```

Out[75]:
```
(415, 13)
```

In [ ]:
```

```