프로젝트 기반 빅데이터 서비스 솔루션 개발 전문과정

교과목명 : 분석라이브러리 활용

- 평가일 : 23.4.13
- 성명 : 임수현
- 점수 : 90

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Q1. 표준정규분포 기반의 2행 3열 배열을 랜덤하게 생성하여 크기, 자료형, 차원을 출력하세요.

In [8]:
```
a = np.random.randn(6).reshape(2,3)
print(len(a))
print(type(a))
print(np.shape(a))
```

```
2
<class 'numpy.ndarray'>
(2, 3)
```

In [251…
```
# A -3
data = np.random.randn(2,3)
print(data.shape)
print(data.dtype)
print(data.ndim)
```

```
(2, 3)
float64
2
```

Q2. arange(), reshape() 이용 1차원 2차원 3차원 배열을 아래와 같이 생성하세요.

[0 1 2 3 4 5 6 7 8 9]

[[0 1 2 3 4]
[5 6 7 8 9]]

[[[0 1 2 3 4]
[5 6 7 8 9]]]

In [17]:
```
a = np.arange(10)
b = a.reshape(2,5)
c = a.reshape(1,2,5)
print(a)
print(b)
print(c)
```

```
[0 1 2 3 4 5 6 7 8 9]
[[0 1 2 3 4]
 [5 6 7 8 9]]
[[[0 1 2 3 4]
  [5 6 7 8 9]]]
```

Q3. 1 ~ 100 까지 배열에서 3과 7의 공배수인 것만을 출력하세요.

In [20]:
```
a = np.arange(1,101)
for i in a:
    if i % 21 == 0:
        print(i)
```

```
21
42
63
84
```

Q4. 아래 3차원 배열을 생성하여 출력한 후 1차원으로 변환하여 출력하세요.(reshape() 사용)

[[[0 1 2 3 4]

[5 6 7 8 9]]

[[10 11 12 13 14]

[15 16 17 18 19]]

[[20 21 22 23 24]

[25 26 27 28 29]]]

In [239…
```
a = np.arange(30).reshape(3,2,5)
print(a)


print(a.reshape(30))
```

```
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]
```

Q5. array2d에서 인덱스를 이용해서 값을 선택하고 리스트로 아래와 같이 출력하세요.

arr2d = np.arange(1,10).reshape(3,3)

[3, 6]

[[1, 2],

[4, 5]]

```
[[1, 2, 3]
[4, 5, 6]]
```

In [29]:
```
arr2d = np.arange(1,10).reshape(3,3)
arr2d
```

Out[29]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [34]:
```
print(arr2d[0:2,2])
print(arr2d[0:2,0:2])
print(arr2d[0:2,0:3])
```

```
[3 6]
[[1 2]
 [4 5]]
[[1 2 3]
 [4 5 6]]
```

Q6. zeros_like, ones_like, full_like 함수 사용 예를 작성하세요.

In [36]:
```
a = np.zeros(9).reshape(3, 3)
print(np.zeros_like(a))
print(np.ones_like(a))
print(np.full_like(a, 5))
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
```

Q7. 10 ~ 20 사이의 정수 난수로 10행 5열 2차원 배열을 생성하고 저장한 후 다시 불러내서 출력하세요.

In [39]:
```
a = np.random.randint(10, 21, 50).reshape(10, 5)
print(a)
```

```
[[15 14 13 10 11]
 [16 13 18 12 20]
 [19 17 16 15 17]
 [16 11 20 19 14]
 [15 19 10 18 10]
 [12 14 13 18 20]
 [10 18 17 12 18]
 [13 16 20 17 18]
 [10 17 11 19 13]
 [18 12 11 12 14]]
```

In [ ]:
```
# A -2
ar = np.random.randint(10, 21, size = (10,5))
np.save("ar_test", ar)
np.load("ar_test.npy")
```

Q8. df = sns.load_dataset('titanic')로 불러와서 다음 작업을 수행한 후 출력하세요.

- 전체 칼럼중 'survived'외에 모든 칼럼을 포함한 df_x를 산출한 후 dataset/df_x.pkl로 저장한다.
- df_x.pkl을 데이터프레임 df_x 이름으로 불러온 후 앞 5개 행을 출력한다.

In [47]:
```
df = sns.load_dataset("titanic")
df_x = df.drop("survived", axis = 1)
df_x.to_pickle("df_x.pkl")
pd.read_pickle("df_x.pkl").head()
```

Out[47]:

	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embar
0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	South
1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Ch
2	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	South
3	1	female	35.0	1	0	53.1000	S	First	woman	False	C	South
4	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	South

Q9. df = sns.load_dataset('titanic')로 불러와서 deck 열에서 NaN 갯수를 계산하세요.

In [58]:
```
df.deck.value_counts(dropna = False)
```

Out[58]:
```
NaN    688
C       59
B       47
D       33
E       32
A       15
F       13
G        4
Name: deck, dtype: int64
```

Q10. Q9의 df에서 각 칼럼별 null 개수와 df 전체의 null 개수를 구하세요.

In [62]:
```
df.isnull().sum()
```

Out[62]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [63]:
```
df.isnull().sum().sum()
```

Out[63]:
```
869
```

아래 tdf 데이터프레임에서 Q11 ~ Q12 작업을 수행하세요.

In [65]:
```
import seaborn as sns
df = sns.load_dataset('titanic')
tdf = df[['survived','sex','age','class']]
tdf.head()
```

Out[65]:

	survived	sex	age	class
0	0	male	22.0	Third
1	1	female	38.0	First
2	1	female	26.0	Third
3	1	female	35.0	First
4	0	male	35.0	Third

Q11. age를 7개 카테고리로 구분하는 새로운 칼럼 'cat_age'를 생성하여 출력하세요. 단, 카테고리 구분을 수행하는 사용자 함수를 만들고 그 함수를 age 칼럼에 매핑하여 결과를 tdf1에 저장하고 출력하세요.

[카테고리]

age <= 5: cat = 'Baby'

age <= 12: cat = 'Child'

age <= 18: cat = 'Teenager'

age <= 25: cat = 'Student'

age <= 60: cat = 'Adult'

age > 60 : cat = 'Elderly'

In [66]:
```
def age_category(age):
    cat = ""
    if age <= 5: cat = 'Baby'
    elif age <= 12: cat = 'Child'
    elif age <= 18: cat = 'Teenager'
```

```
    elif age <= 25: cat = 'Student'
    elif age <= 60: cat = 'Adult'
    elif age > 60 : cat = 'Elderly'
    return cat

tdf1 = tdf.copy()
tdf1["cat_age"] = tdf1["age"].apply(lambda x: age_category(x))
tdf1
```

Out[66]:

	survived	sex	age	class	cat_age
0	0	male	22.0	Third	Student
1	1	female	38.0	First	Adult
2	1	female	26.0	Third	Adult
3	1	female	35.0	First	Adult
4	0	male	35.0	Third	Adult
...	...	...	...	...	...
886	0	male	27.0	Second	Adult
887	1	female	19.0	First	Student
888	0	female	NaN	Third	
889	1	male	26.0	First	Adult
890	0	male	32.0	Third	Adult

891 rows × 5 columns

Q12. tdf1의 sex, class 칼럼을 '_'으로 연결한 'sc'칼럼을 추가한 후 아래와 같이 출력하세요.

In [73]:
```
tdf1["sc"] = tdf1["sex"].astype(str) + "_" + tdf1["class"].astype(str)
tdf1
```

Out[73]:

	survived	sex	age	class	cat_age	sc
0	0	male	22.0	Third	Student	male_Third
1	1	female	38.0	First	Adult	female_First
2	1	female	26.0	Third	Adult	female_Third
3	1	female	35.0	First	Adult	female_First
4	0	male	35.0	Third	Adult	male_Third
...	...	...	...	...	...	...
886	0	male	27.0	Second	Adult	male_Second
887	1	female	19.0	First	Student	female_First
888	0	female	NaN	Third		female_Third
889	1	male	26.0	First	Adult	male_First
890	0	male	32.0	Third	Adult	male_Third

891 rows × 6 columns

In [ ]:
```
# A
tdf1["sc"] = tdf1[["sex","class"]].agg("_".join, axis = 1)
tdf1.head()
```

Q13. join() 메소드는 두 데이터프레임의 행 인덱스를 기준으로 결합한다. 2개의 주식데이터를 가져와서 join() 메소드로 아래와 같이 결합한 후 다음 사항을 수행하세요.

- df1과 df2의 교집합만 출력되도록 결합하여 df3에 저장하고 출력
- df3에서 중복된 칼럼을 삭제한 후 블린 인덱싱을 이용하여 eps가 3000 보다 적거나 stock_name이 이마트인 데이터를 선택하여 데이터프레임을 생성하고 df4 이름으로 저장 및 출력하세요.(단, '<' 와 '==' 를 반드시 사용해야 함)

In [240…
```
df1 = pd.read_excel('./dataset/stock price.xlsx', index_col='id')
df2 = pd.read_excel('./dataset/stock valuation.xlsx', index_col='id')
```

In [241…
```
df1.columns = ["name","value","price"]
df1
```

Out[241]:

id	name	value	price
128940	한미약품	59385.666667	421000
130960	CJ E&M	58540.666667	98900
138250	엔에스쇼핑	14558.666667	13200
139480	이마트	239230.833333	254500
142280	녹십자엠에스	468.833333	10200
145990	삼양사	82750.000000	82000
185750	종근당	40293.666667	100500
192400	쿠쿠홀딩스	179204.666667	177500
199800	툴젠	-2514.333333	115400
204210	모두투어리츠	3093.333333	3475

In [243…

```
df3 = df1.join(df2.set_index('name'), on='name', how='inner')
df3
```

Out[243]:

id	name	value	price	eps	bps	per	pbr
130960	CJ E&M	58540.666667	98900	6301.333333	54068	15.695091	1.829178
139480	이마트	239230.833333	254500	18268.166667	295780	13.931338	0.860437
145990	삼양사	82750.000000	82000	5741.000000	108090	14.283226	0.758627
185750	종근당	40293.666667	100500	3990.333333	40684	25.185866	2.470259
204210	모두투어리츠	3093.333333	3475	85.166667	5335	40.802348	0.651359

In [109…

```
df4 = df3[(df3.name == "이마트") | (df3.eps < 3000)]
df4
```

Out[109]:

id	name	value	price	eps	bps	per	pbr
139480	이마트	239230.833333	254500	18268.166667	295780	13.931338	0.860437
204210	모두투어리츠	3093.333333	3475	85.166667	5335	40.802348	0.651359

Q14. 배열 a에 대하여 3차원 자리에 2차원을 2차원 자리에 1차원을 1차원 자리에 3차원을 넣어서 변환하여 출력하세요

In [253…

```
a = np.arange(6).reshape(1,2,3)
print(a,a.shape,'₩n')
```

```
[[[0 1 2]
  [3 4 5]]] (1, 2, 3)
```

In [255…

```
a.reshape(2,3,1)
```

Out[255]:
```
array([[[0],
        [1],
        [2]],

       [[3],
        [4],
        [5]]])
```

- np.arange 함수를 사용하여 0부터 5까지의 값을 갖는 1차원 배열을 생성합니다. 이후 reshape 함수를 사용하여 이 1차원 배열을 1행 2열 3깊이를 갖는 3차원 배열로 변환
- np.transpose 함수를 사용하여 배열의 차원을 변환합니다. 이 함수는 첫 번째 인자로 배열을, 두 번째 인자로 축 순서를 지정하는 튜플을 받습니다. 여기서는 (1,2,0)이라는 튜플을 전달하여, 원래 배열에서 1번째 축을 0번째 축으로, 2번째 축을 1번째 축으로, 0번째 축을 2번째 축으로 바꿉니다.
- y는 (2,3,1) 크기의 배열이 되며, 이는 원래 배열 a의 축 순서가 (1,2,0)으로 바뀐 결과입니다.

In [ ]:

Q15. 'mpg'를 'kpl' 로 환산하여 새로운 열을 생성하고 반올림하여 소수점 아래 둘째 자리까지 처음 5개행을 출력하세요.

In [112…
```
# read_csv() 함수로 df 생성
import pandas as pd
auto_df = pd.read_csv('./dataset/auto-mpg.csv')
# 열 이름을 지정
auto_df.columns = ['mpg','cylinders','displacement','horsepower','weight',
                   'acceleration','model year','origin','name']
auto_df
```

Out[112]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

In [114…

```
auto_df["kpl"] = round(auto_df["mpg"] * 2.3523, 2)
auto_df.head()
```

Out[114]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name	k
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	42..
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	35..
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	42..
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	37.(
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	39.!

Q16. './dataset/stock-data.csv'를 데이터프레임으로 불러와서 datetime64 자료형으로 변환한 후에 년, 월, 일로 분리하고 year를 인덱스로 셋팅하여 출력하세요.

In [116…
```
df = pd.read_csv("dataSet/stock-data.csv")
```

In [117…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Date    20 non-null     object
 1   Close   20 non-null     int64 
 2   Start   20 non-null     int64 
 3   High    20 non-null     int64 
 4   Low     20 non-null     int64 
 5   Volume  20 non-null     int64 
dtypes: int64(5), object(1)
memory usage: 1.1+ KB
```

In [121…
```
df["Date"] = pd.to_datetime(df["Date"])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Date    20 non-null     datetime64[ns]
 1   Close   20 non-null     int64         
 2   Start   20 non-null     int64         
 3   High    20 non-null     int64         
 4   Low     20 non-null     int64         
 5   Volume  20 non-null     int64         
dtypes: datetime64[ns](1), int64(5)
memory usage: 1.1 KB
```

In [124…
```
df["year"] = df.Date.dt.year
df["month"] = df.Date.dt.month
df["day"] = df.Date.dt.day
df.set_index("year")
```

Out[124]:

year	Date	Close	Start	High	Low	Volume	month	day
2018	2018-07-02	10100	10850	10900	10000	137977	7	2
2018	2018-06-29	10700	10550	10900	9990	170253	6	29
2018	2018-06-28	10400	10900	10950	10150	155769	6	28
2018	2018-06-27	10900	10800	11050	10500	133548	6	27
2018	2018-06-26	10800	10900	11000	10700	63039	6	26
2018	2018-06-25	11150	11400	11450	11000	55519	6	25
2018	2018-06-22	11300	11250	11450	10750	134805	6	22
2018	2018-06-21	11200	11350	11750	11200	133002	6	21
2018	2018-06-20	11550	11200	11600	10900	308596	6	20
2018	2018-06-19	11300	11850	11950	11300	180656	6	19
2018	2018-06-18	12000	13400	13400	12000	309787	6	18
2018	2018-06-15	13400	13600	13600	12900	201376	6	15
2018	2018-06-14	13450	13200	13700	13150	347451	6	14
2018	2018-06-12	13200	12200	13300	12050	558148	6	12
2018	2018-06-11	11950	12000	12250	11950	62293	6	11
2018	2018-06-08	11950	11950	12200	11800	59258	6	8
2018	2018-06-07	11950	12200	12300	11900	49088	6	7
2018	2018-06-05	12150	11800	12250	11800	42485	6	5
2018	2018-06-04	11900	11900	12200	11700	25171	6	4
2018	2018-06-01	11900	11800	12100	11750	32062	6	1

Q17. titanic 데이터셋에서 5개 열을 선택해서 class열을 기준으로 그룹화를 수행한 후 아래와 같이 출력하였다. 다음 사항을 출력하세요.

5개 열 : ['age','sex', 'class', 'fare', 'survived']

- 그룹별 평균 출력
- 그룹별 최대값 출력

In [245…

```
import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
df1 = df[["age","sex","class","fare","survived"]]
df1["sex"].replace(["female","male"], [0,1], inplace = True)
df1["class"].replace(["Third","Second","First"],[3,2,1], inplace = True)
df1.sex = df1.sex.astype(int)
df1["class"] = df1["class"].astype(int)

df1.mean()
```

Out[245]:
```
age          29.699118
sex           0.647587
class         2.308642
fare         32.204208
survived      0.383838
dtype: float64
```

In [246…
```
df1.max()
```

Out[246]:
```
age          80.0000
sex           1.0000
class         3.0000
fare        512.3292
survived      1.0000
dtype: float64
```

In [138…
```
df1["class"].value_counts()
```

Out[138]:
```
3    491
1    216
2    184
Name: class, dtype: int64
```

In [ ]:
```
# A -4
grouped = df.groupby["class"]
print(grouped.mean())
print(grouped.max())
```

Q18. titanic 데이터셋에서 'Third'그룹만을 선택해서 group3 이름으로 저장하고 통계요약표를 출력하세요.

In [154…
```
import seaborn as sns
df = sns.load_dataset('titanic')
df.head()
```

Out[154]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	de
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	N
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	N
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	N

In [155…
```
group3 = df[df["class"] == "Third"]
group3.describe()
```

Out[155]:

	survived	pclass	age	sibsp	parch	fare
count	491.000000	491.0	355.000000	491.000000	491.000000	491.000000
mean	0.242363	3.0	25.140620	0.615071	0.393075	13.675550
std	0.428949	0.0	12.495398	1.374883	0.888861	11.778142
min	0.000000	3.0	0.420000	0.000000	0.000000	0.000000
25%	0.000000	3.0	18.000000	0.000000	0.000000	7.750000
50%	0.000000	3.0	24.000000	0.000000	0.000000	8.050000
75%	0.000000	3.0	32.000000	1.000000	0.000000	15.500000
max	1.000000	3.0	74.000000	8.000000	6.000000	69.550000

Q19. titanic 데이터셋에서 다음 전처리를 수행하세요.

1. df에서 중복 칼럼으로 고려할 수 있는 컬럼들(6개 내외)을 삭제한 후 나머지 컬럼들로 구성되는 데이터프레임을 df1 이름으로 저장 후 출력하세요.
2. df1에서 null값이 50% 이상인 칼럼을 삭제 후 df2 이름으로 저장하고 출력하세요.
3. df2에서 결측값이 있는 age 칼럼에 대해서 평균값으로 대체 처리를 수행하세요.
4. df2에서 결측값이 있는 embarked 칼럼에 대해서 앞행의 값으로 대체 처리를 수행하세요.
5. df2 문자로 되어있는 칼럼들을 레이블 인코딩 수행하여 숫자로 변환 후 df2.info()를 출력하세요

In [157…

```
df
```

Out[157]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

891 rows × 15 columns

In [161…

```
df1 = df.drop(["class","who","adult_male","embark_town","alive"], axis = 1)
df1
```

Out[161]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	alone
0	0	3	male	22.0	1	0	7.2500	S	NaN	False
1	1	1	female	38.0	1	0	71.2833	C	C	False
2	1	3	female	26.0	0	0	7.9250	S	NaN	True
3	1	1	female	35.0	1	0	53.1000	S	C	False
4	0	3	male	35.0	0	0	8.0500	S	NaN	True
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	NaN	True
887	1	1	female	19.0	0	0	30.0000	S	B	True
888	0	3	female	NaN	1	2	23.4500	S	NaN	False
889	1	1	male	26.0	0	0	30.0000	C	C	True
890	0	3	male	32.0	0	0	7.7500	Q	NaN	True

891 rows × 10 columns

In [163…

```
a = len(df1) /2
df2 = df1.dropna(thresh = a, axis=1)
df2
```

Out[163]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	alone
0	0	3	male	22.0	1	0	7.2500	S	False
1	1	1	female	38.0	1	0	71.2833	C	False
2	1	3	female	26.0	0	0	7.9250	S	True
3	1	1	female	35.0	1	0	53.1000	S	False
4	0	3	male	35.0	0	0	8.0500	S	True
...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	True
887	1	1	female	19.0	0	0	30.0000	S	True
888	0	3	female	NaN	1	2	23.4500	S	False
889	1	1	male	26.0	0	0	30.0000	C	True
890	0	3	male	32.0	0	0	7.7500	Q	True

891 rows × 9 columns

In [169…

```
df2["age"].fillna(df2["age"].mean(), inplace = True)
df2.age.isnull().sum()
```

Out[169]:

```
0
```

In [170…

```
df2.embarked.fillna(method='ffill', inplace=True)
df2.embarked.isnull().sum()
```

Out[170]:

```
0
```

In [172…
```
df2
```

Out[172]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	alone
0	0	3	male	22.000000	1	0	7.2500	S	False
1	1	1	female	38.000000	1	0	71.2833	C	False
2	1	3	female	26.000000	0	0	7.9250	S	True
3	1	1	female	35.000000	1	0	53.1000	S	False
4	0	3	male	35.000000	0	0	8.0500	S	True
...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.000000	0	0	13.0000	S	True
887	1	1	female	19.000000	0	0	30.0000	S	True
888	0	3	female	29.699118	1	2	23.4500	S	False
889	1	1	male	26.000000	0	0	30.0000	C	True
890	0	3	male	32.000000	0	0	7.7500	Q	True

891 rows × 9 columns

In [173…
```
df2 = pd.get_dummies(df2, columns=['sex',"embarked","alone"])
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   age          891 non-null    float64
 3   sibsp        891 non-null    int64
 4   parch        891 non-null    int64
 5   fare         891 non-null    float64
 6   sex_female   891 non-null    uint8
 7   sex_male     891 non-null    uint8
 8   embarked_C   891 non-null    uint8
 9   embarked_Q   891 non-null    uint8
 10  embarked_S   891 non-null    uint8
 11  alone_False  891 non-null    uint8
 12  alone_True   891 non-null    uint8
dtypes: float64(2), int64(4), uint8(7)
memory usage: 48.0 KB
```

In [ ]:
```
# A -1
from sklearn.preprocessing import LableEncoder

features = ["sex","embarked","alone"]
for feature in features:
    le = LabelEncoder()
    df2[feature] = le.fit_transform(df2[feature])
df2.info()
```

Q20. 보스톤 주택가격 데이터를 탐색한 후 가장 중요한 독립변수 2개를 선정하고 그 이유를 시각화하여 설명하세요.

In [175…
```
from sklearn.datasets import load_boston
# boston 데이타셋 로드
boston = load_boston()

# boston 데이타셋 DataFrame 변환
bostonDF = pd.DataFrame(boston.data , columns = boston.feature_names)

# boston dataset의 target array는 주택 가격임. 이를 PRICE 컬럼으로 DataFrame에 추가한
bostonDF['PRICE'] = boston.target
print('Boston 데이타셋 크기 :',bostonDF.shape)
bostonDF.head()
```

```
Boston 데이타셋 크기 : (506, 14)
```

Out[175]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
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	4.98
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	9.14
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	4.03
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	2.94
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	5.33

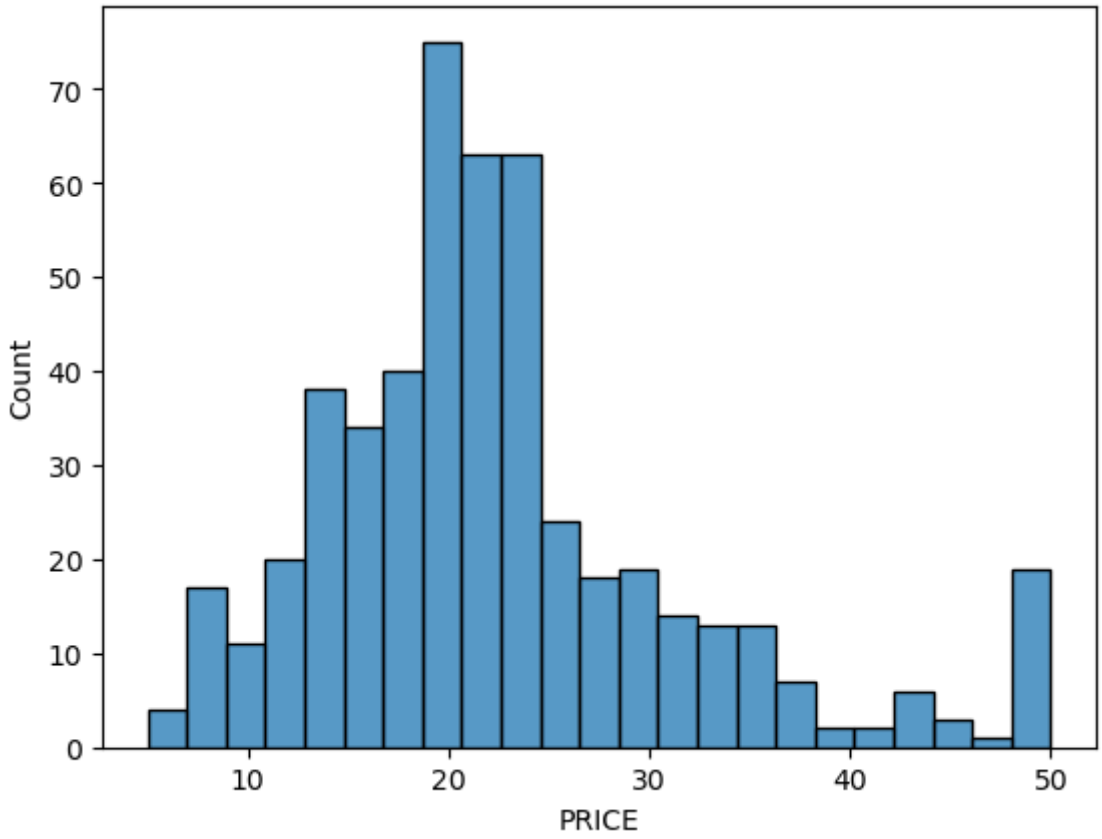
In [216…
```
bostonDF.corr()
```

Out[216]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0
PRICE	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0

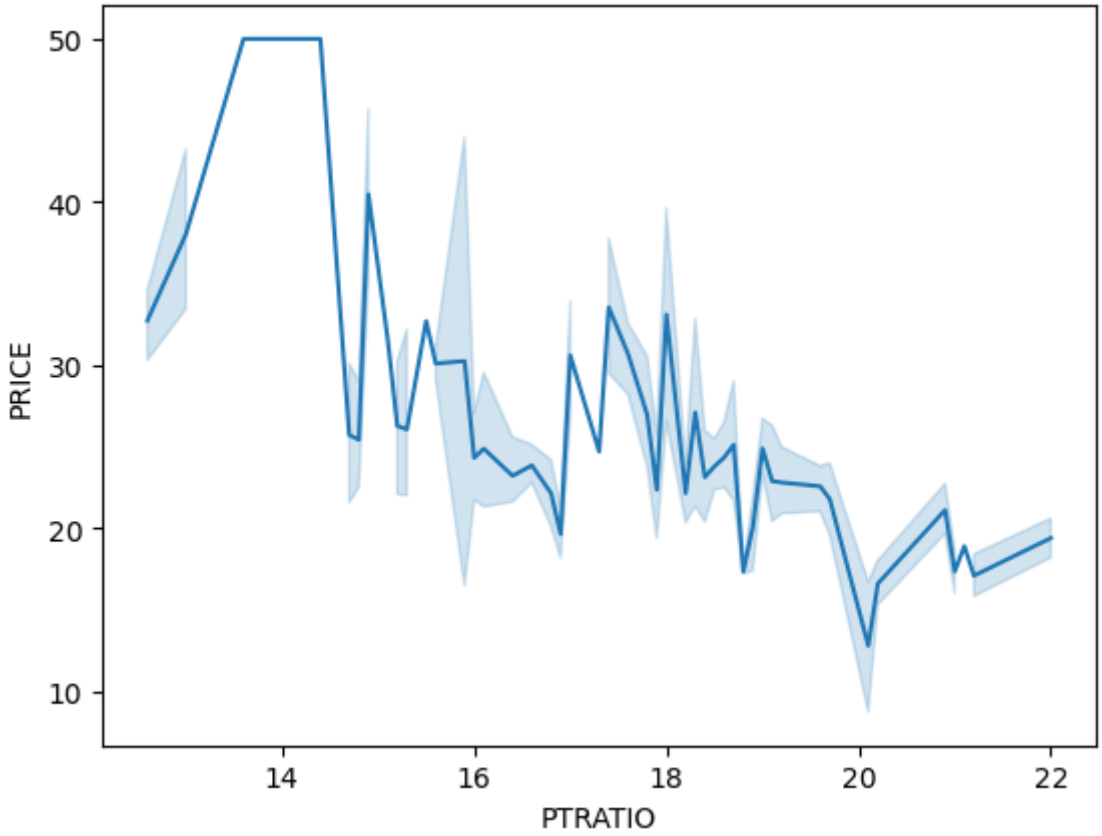
In [247…
```
sns.histplot(data = bostonDF, x = "PRICE")
```

Out[247]:
```
<AxesSubplot:xlabel='PRICE', ylabel='Count'>
```



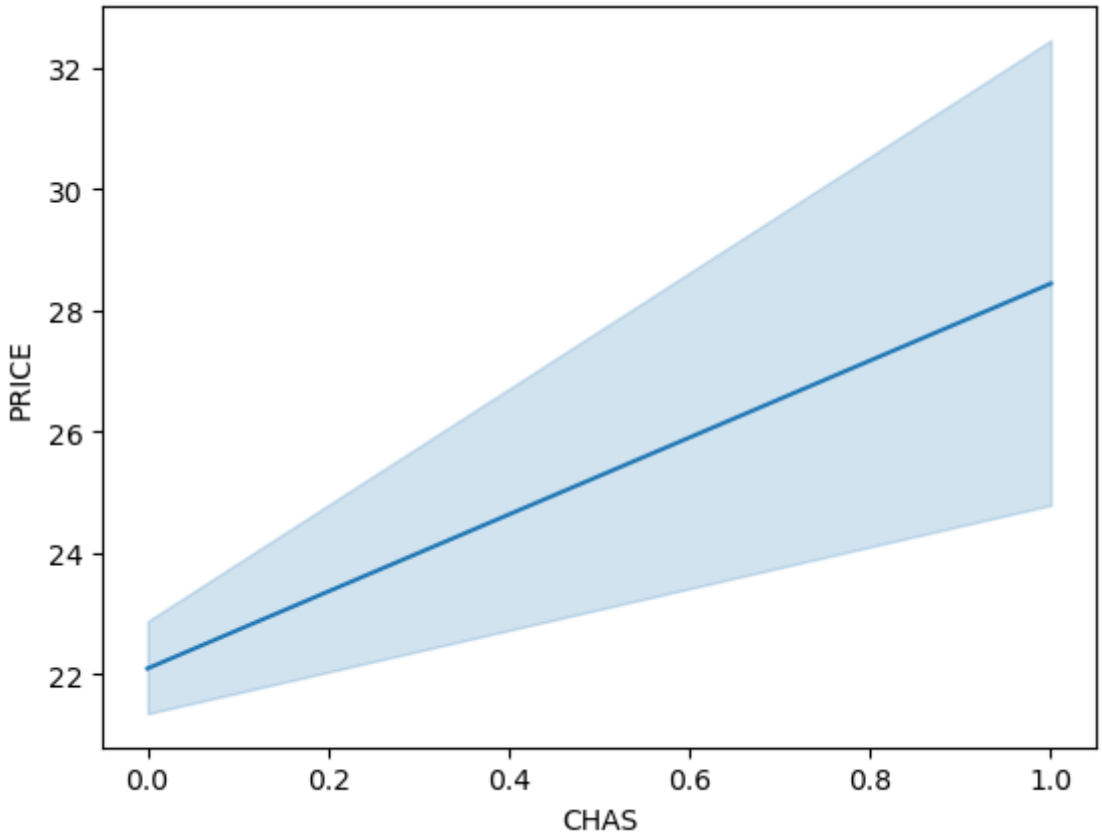
In [188…
```
sns.lineplot(data = bostonDF, x = "PTRATIO", y = "PRICE")
```

Out[188]:
```
<AxesSubplot:xlabel='PTRATIO', ylabel='PRICE'>
```



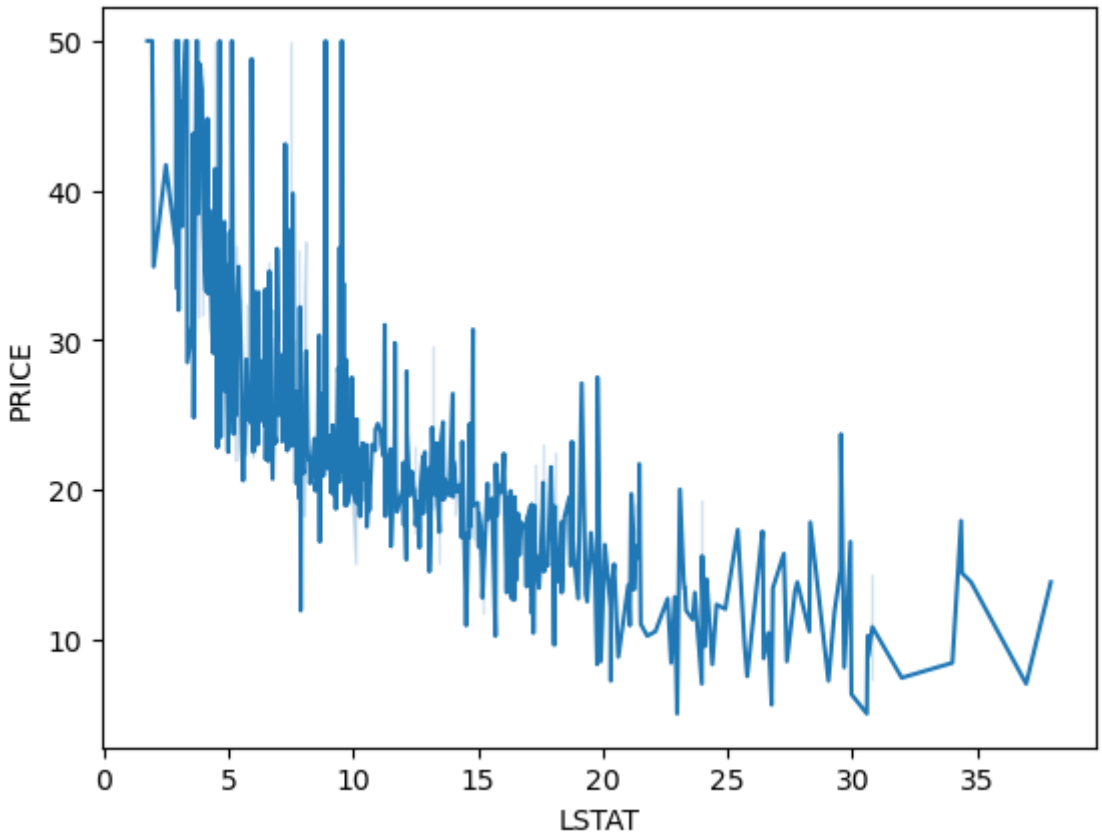
In [191…
```
sns.lineplot(data = bostonDF, x = "CHAS", y = "PRICE")
```

Out[191]:
```
<AxesSubplot:xlabel='CHAS', ylabel='PRICE'>
```



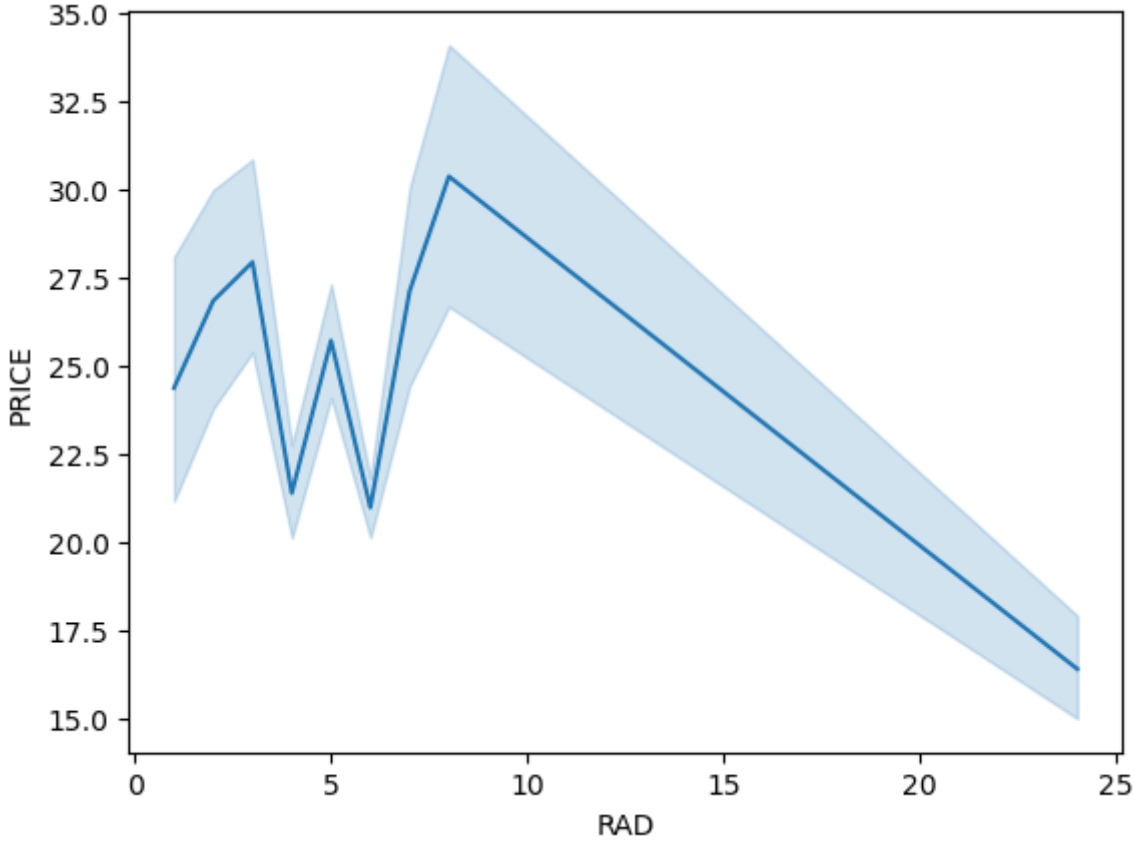
In [196…
```
sns.lineplot(data = bostonDF, x = "LSTAT", y = "PRICE")
```

Out[196]:
```
<AxesSubplot:xlabel='LSTAT', ylabel='PRICE'>
```



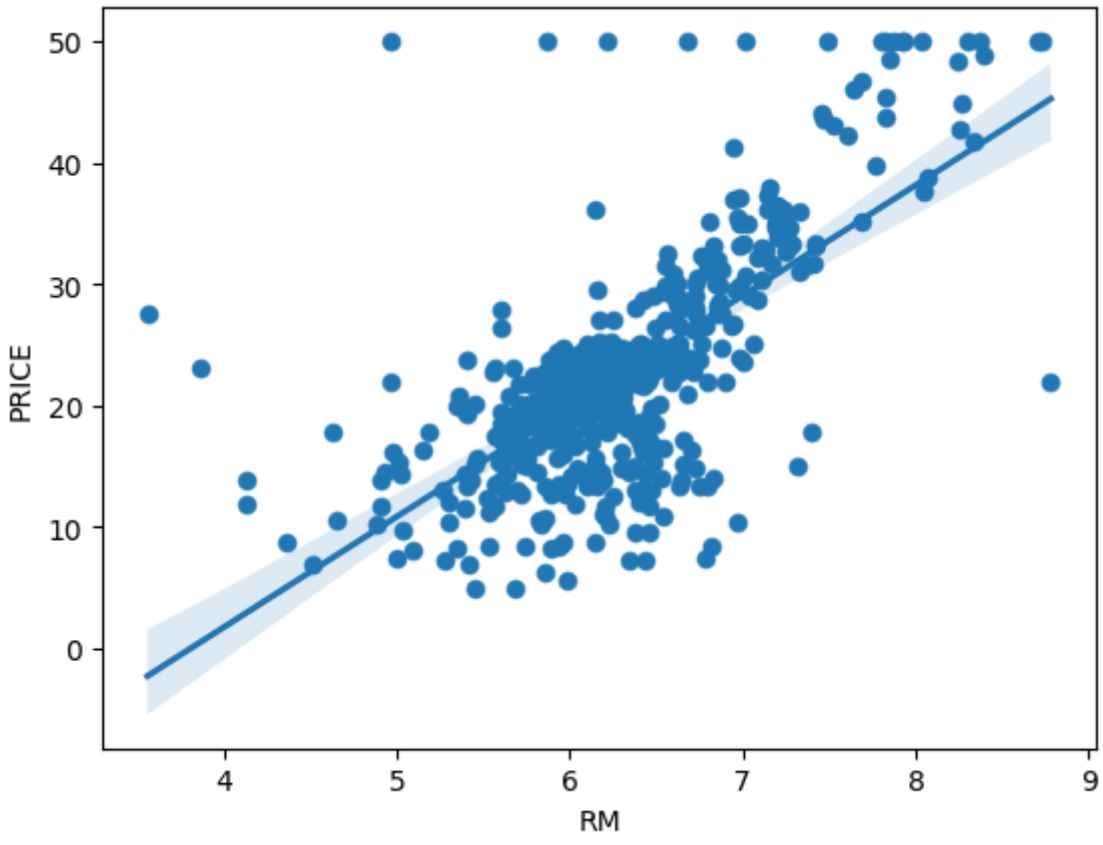
In [209…
```
sns.lineplot(data = bostonDF, x = "RAD", y = "PRICE")
```

Out[209]:
```
<AxesSubplot:xlabel='RAD', ylabel='PRICE'>
```



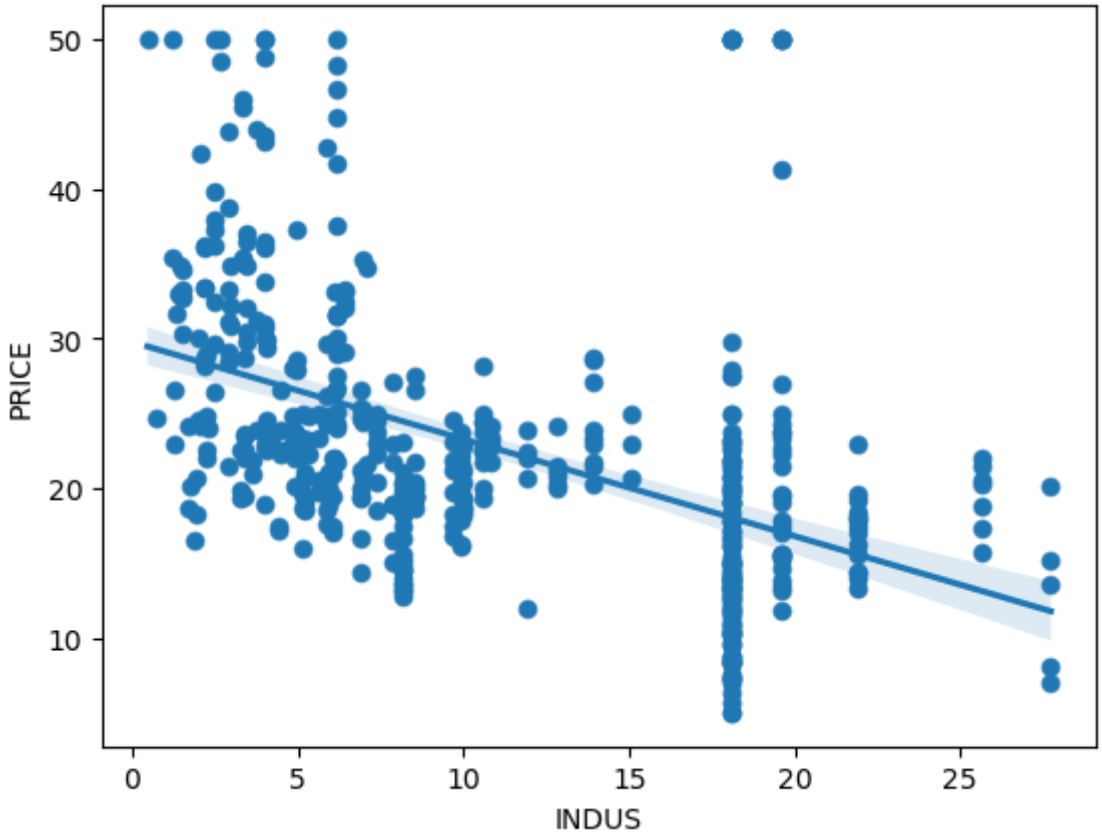
In [227…
```
sns.scatterplot(data = bostonDF, x = "RM", y = "PRICE")
sns.regplot(data = bostonDF, x = "RM", y = "PRICE")
```

Out[227]:
```
<AxesSubplot:xlabel='RM', ylabel='PRICE'>
```



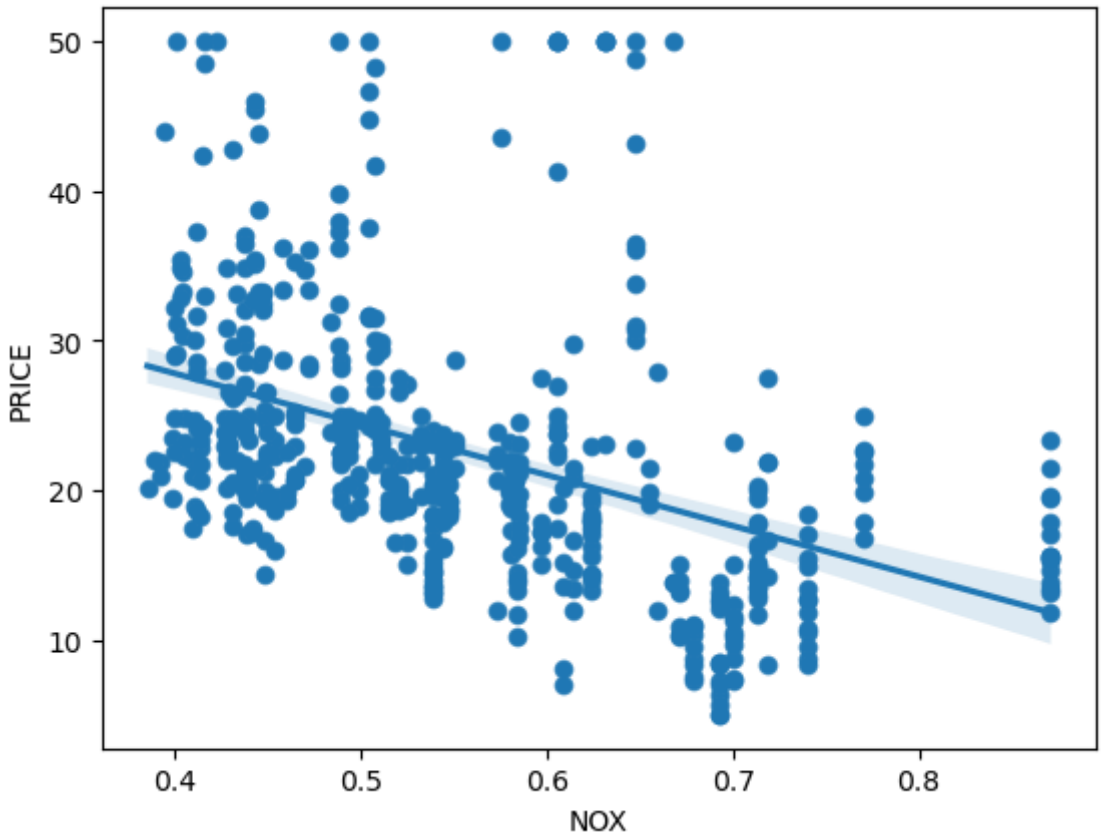
In [221…
```
sns.scatterplot(data = bostonDF, x = "INDUS", y = "PRICE")
sns.regplot(data = bostonDF, x = "INDUS", y = "PRICE")
```

Out[221]: <AxesSubplot:xlabel='INDUS', ylabel='PRICE'>



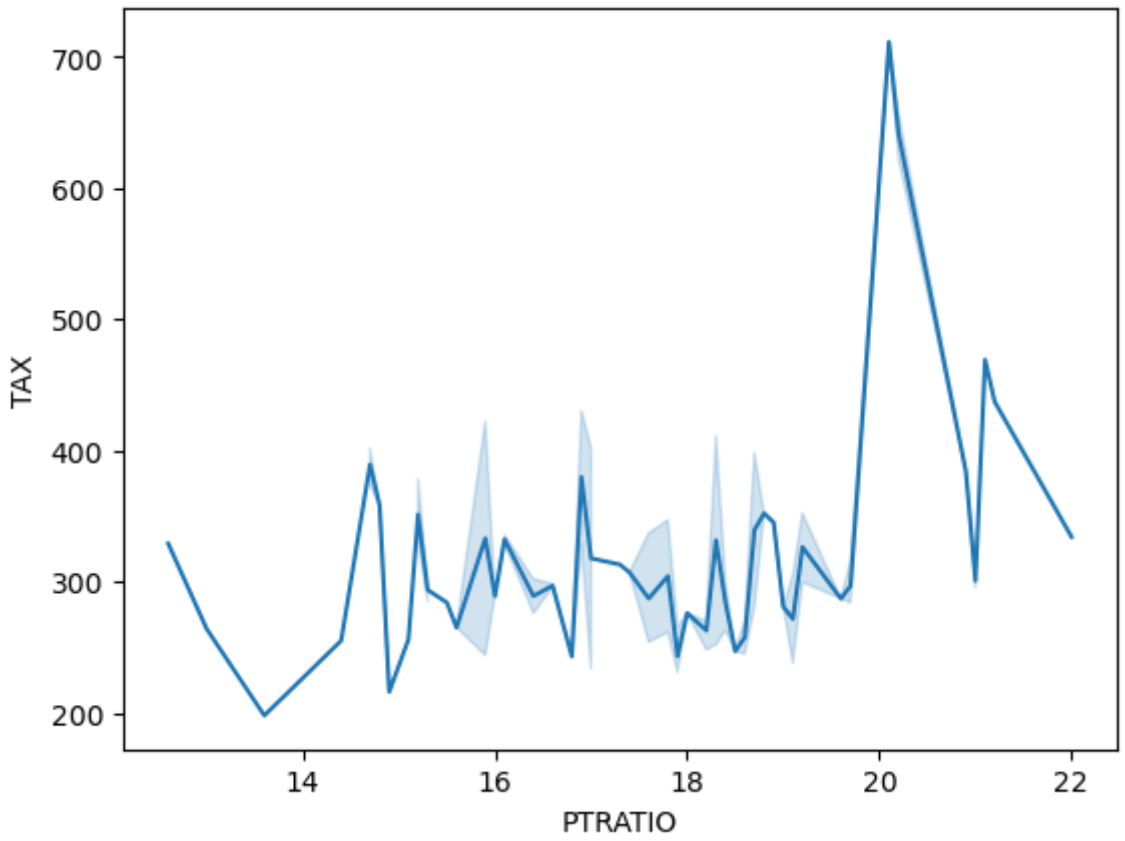
In [228…
```
sns.scatterplot(data = bostonDF, x = "NOX", y = "PRICE")
sns.regplot(data = bostonDF, x = "NOX", y = "PRICE")
```

Out[228]: <AxesSubplot:xlabel='NOX', ylabel='PRICE'>



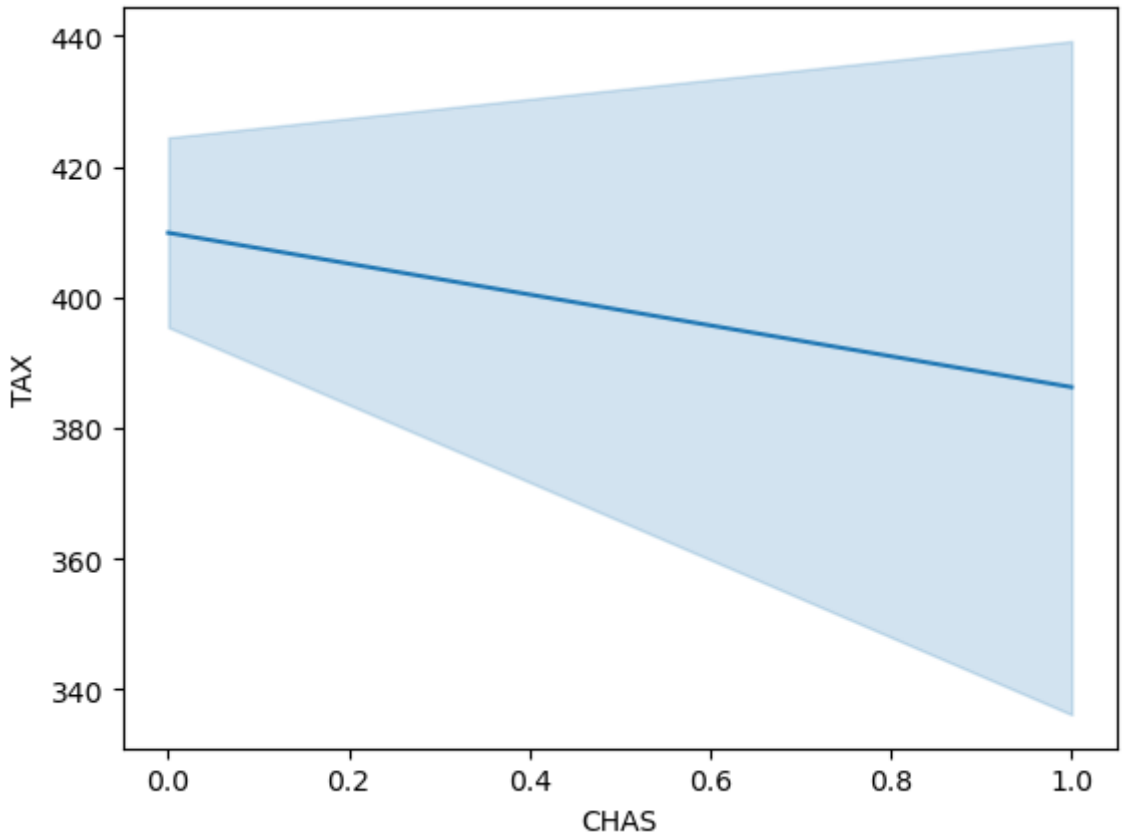
In [201…
```
sns.lineplot(data = bostonDF, x = "PTRATIO", y = "TAX")
```

Out[201]: <AxesSubplot:xlabel='PTRATIO', ylabel='TAX'>



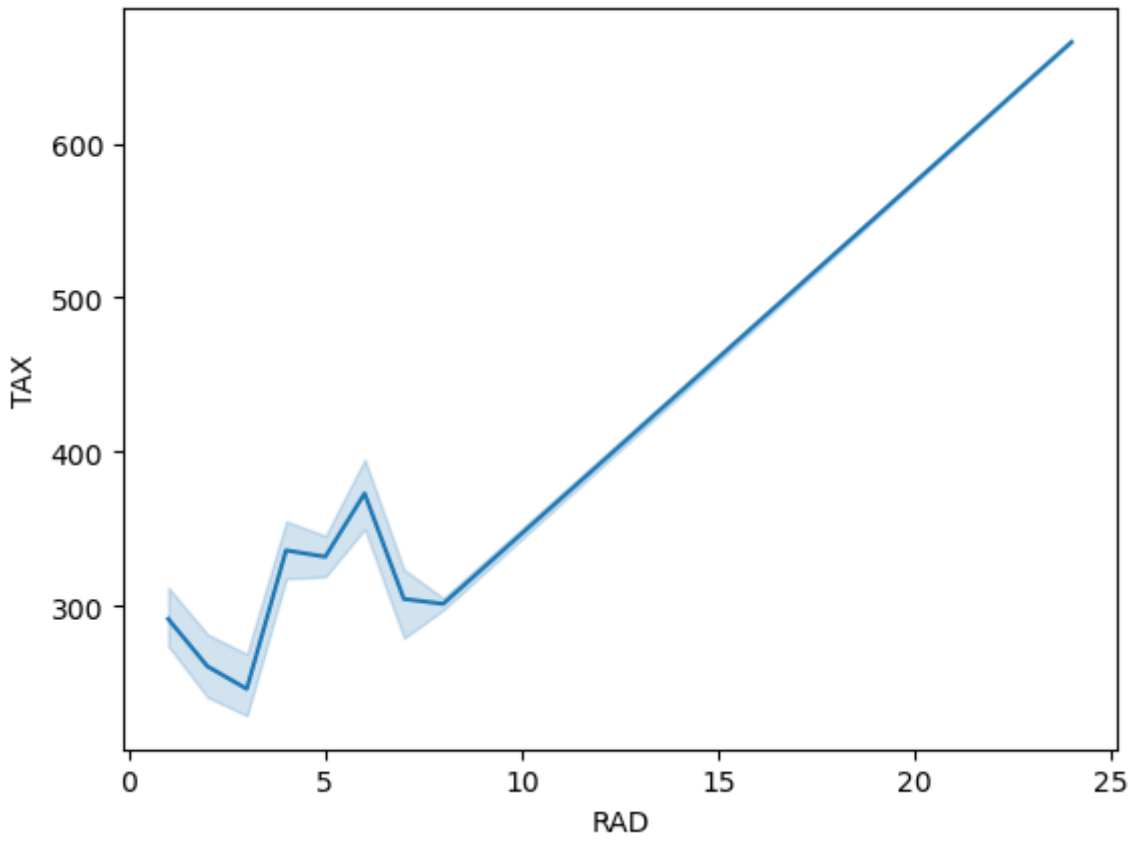
In [203…
```
sns.lineplot(data = bostonDF, x = "CHAS", y = "TAX")
```

Out[203]: <AxesSubplot:xlabel='CHAS', ylabel='TAX'>



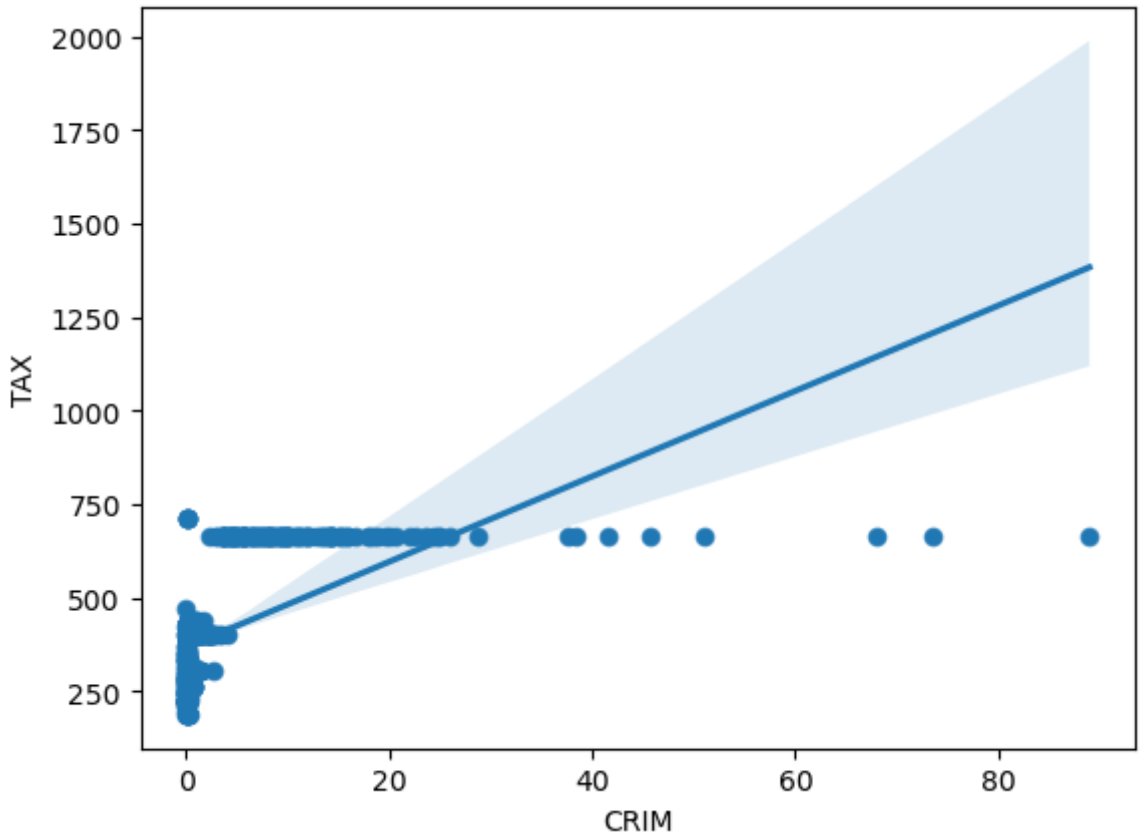
In [206…
```
sns.lineplot(data = bostonDF, x = "RAD", y = "TAX")
```

Out[206]: <AxesSubplot:xlabel='RAD', ylabel='TAX'>



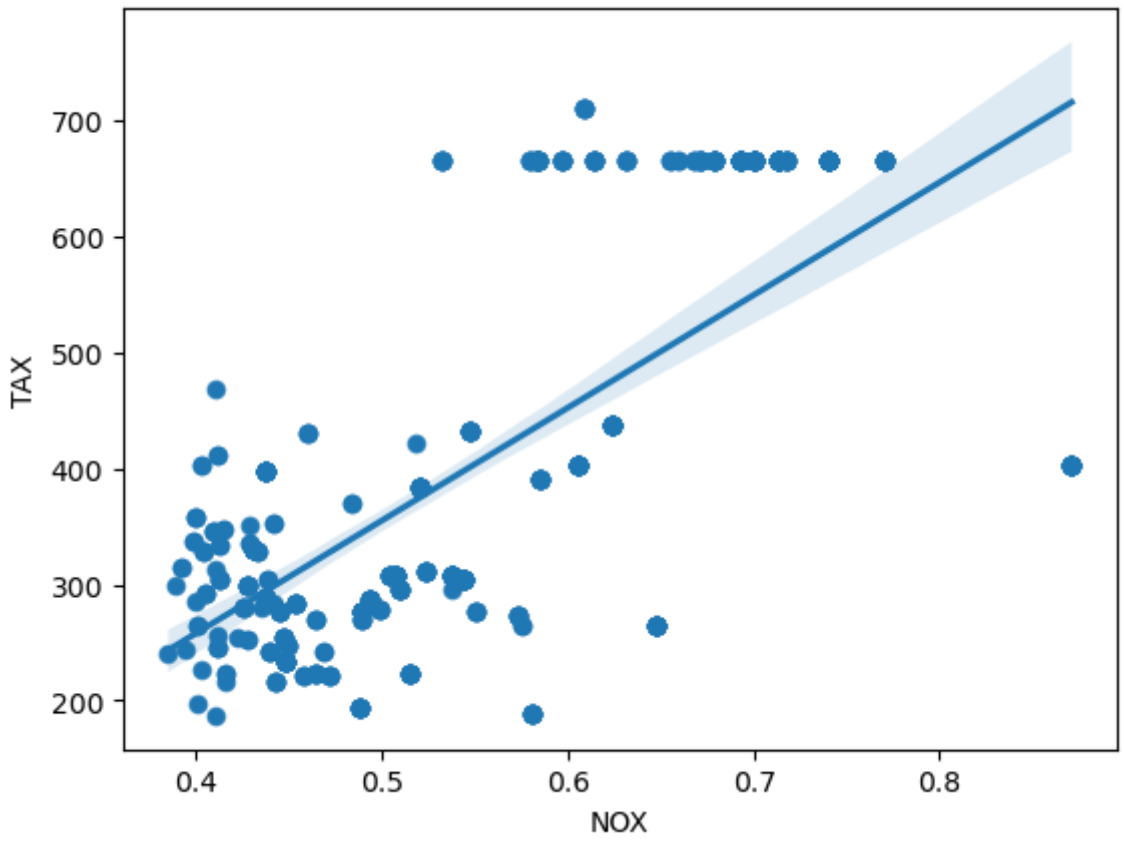
In [229…
```
sns.scatterplot(data = bostonDF, x = "CRIM", y = "TAX")
sns.regplot(data = bostonDF, x = "CRIM", y = "TAX")
```

Out[229]: <AxesSubplot:xlabel='CRIM', ylabel='TAX'>



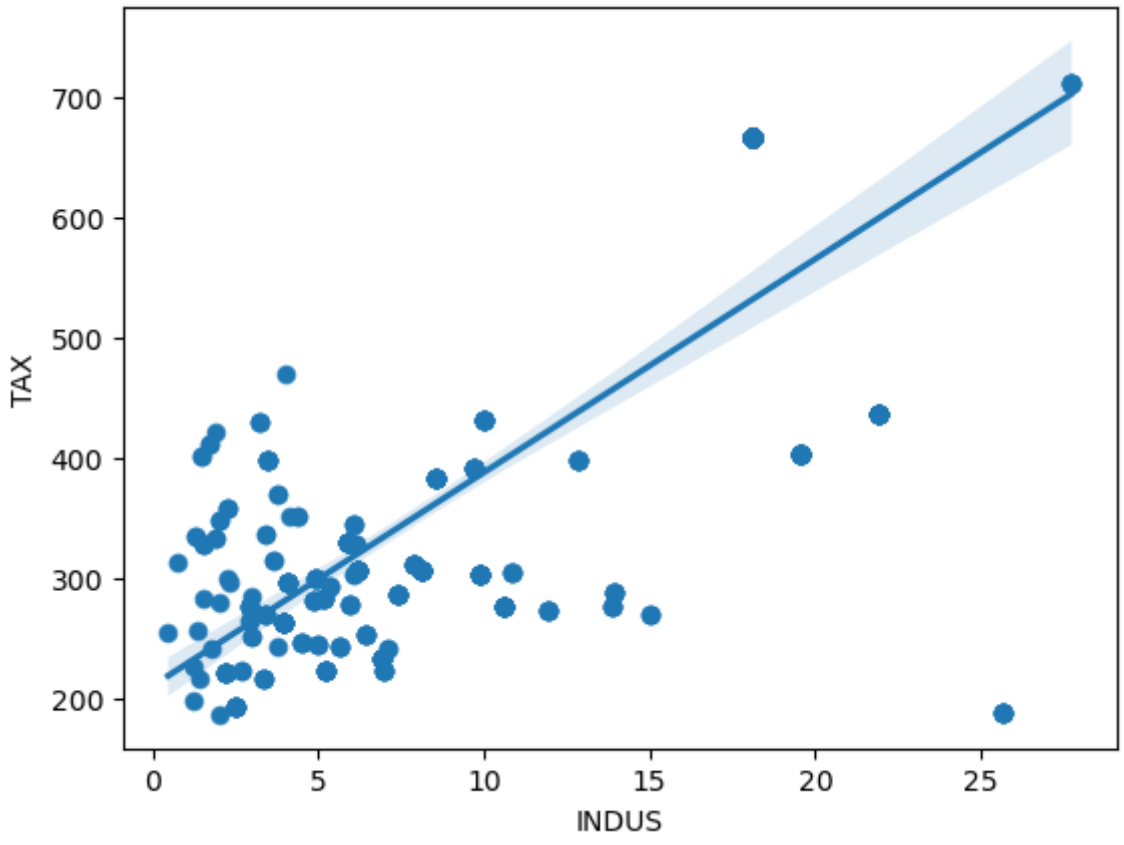
In [230…
```
sns.scatterplot(data = bostonDF, x = "NOX", y = "TAX")
sns.regplot(data = bostonDF, x = "NOX", y = "TAX")
```

Out[230]: <AxesSubplot:xlabel='NOX', ylabel='TAX'>



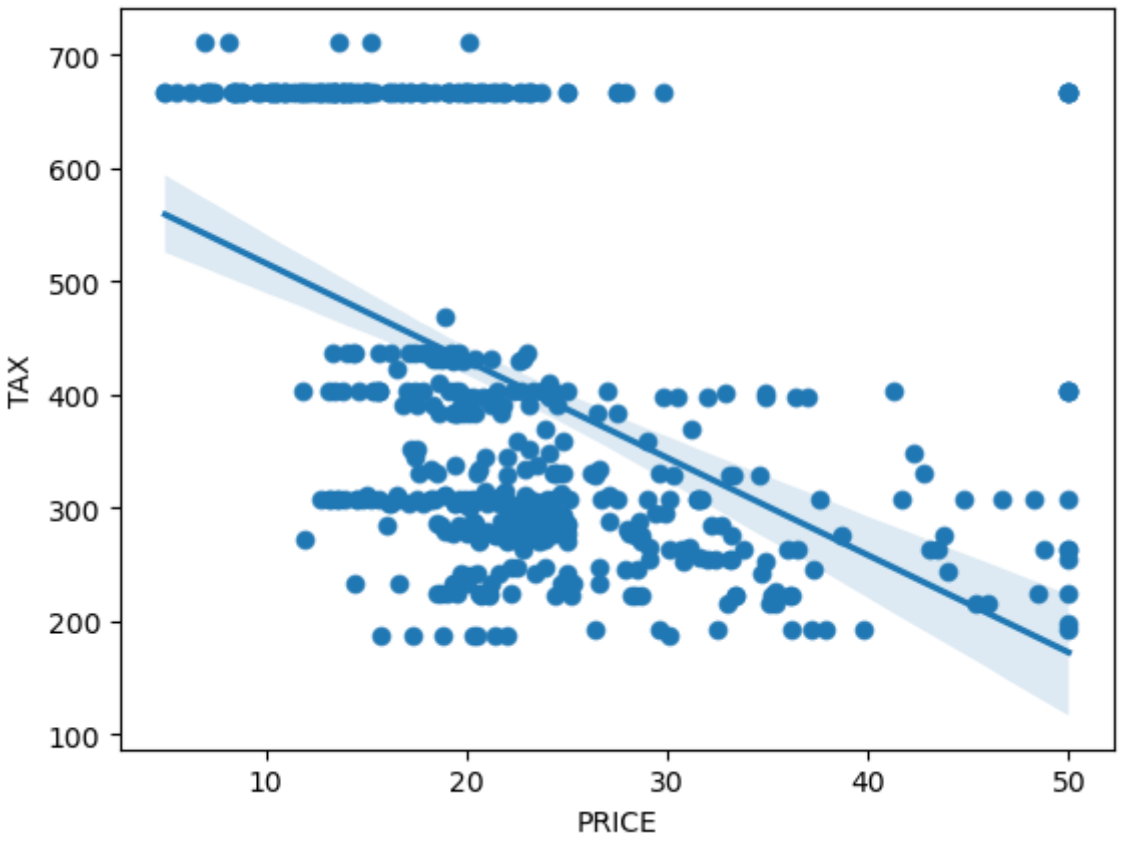
In [231…
```
sns.scatterplot(data = bostonDF, x = "INDUS", y = "TAX")
sns.regplot(data = bostonDF, x = "INDUS", y = "TAX")
```

Out[231]: <AxesSubplot:xlabel='INDUS', ylabel='TAX'>



In [232…
```
sns.scatterplot(data = bostonDF, x = "PRICE", y = "TAX")
sns.regplot(data = bostonDF, x = "PRICE", y = "TAX")
```

Out[232]: <AxesSubplot:xlabel='PRICE', ylabel='TAX'>



In [ ]: