2-3. 상관 관계 분석

```
#상관 관계 행렬
df_corr=df.corr()

#히트맵
plt.figure(figsize=(10,10))
sns.set(font_scale=0.8)
sns.heatmap(df_corr, annot=True, cbar=False)
plt.show()
```

결과창

corr()을 이용해 숫자 데이터를 갖는 변수 간의 상관 계수를 계산했다. heatmap()을 사용해 시각화하였으며, 이미지 크기는 (10,10), 폰트스케일은 0.8로 설정했다.

```
[10] #Target 변수와 상관관계가 높은 순으로 출력
     corr_order=df.corr().loc[:'LSTAT', 'Target'].abs().sort_values(ascending=False)
     corr_order
```

```
LSTAT      0.737663
RM         0.695360
PTRATIO    0.507787
INDUS      0.483725
TAX        0.468536
NOX        0.427321
CRIM       0.388305
RAD        0.381626
AGE        0.376955
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
Name: Target, dtype: float64
```

상관관계 높은 순으로 출력

목표변수 Target과 상관 계수가 높은 순서대로 출력을 했다. abs()로 상관 계수의 값을 모두 양수로 변경해주었고, sort_values()에 ascending=False 옵션을 적용해 상관 계수 값을 기준으로 내림차순 정렬을 했다.

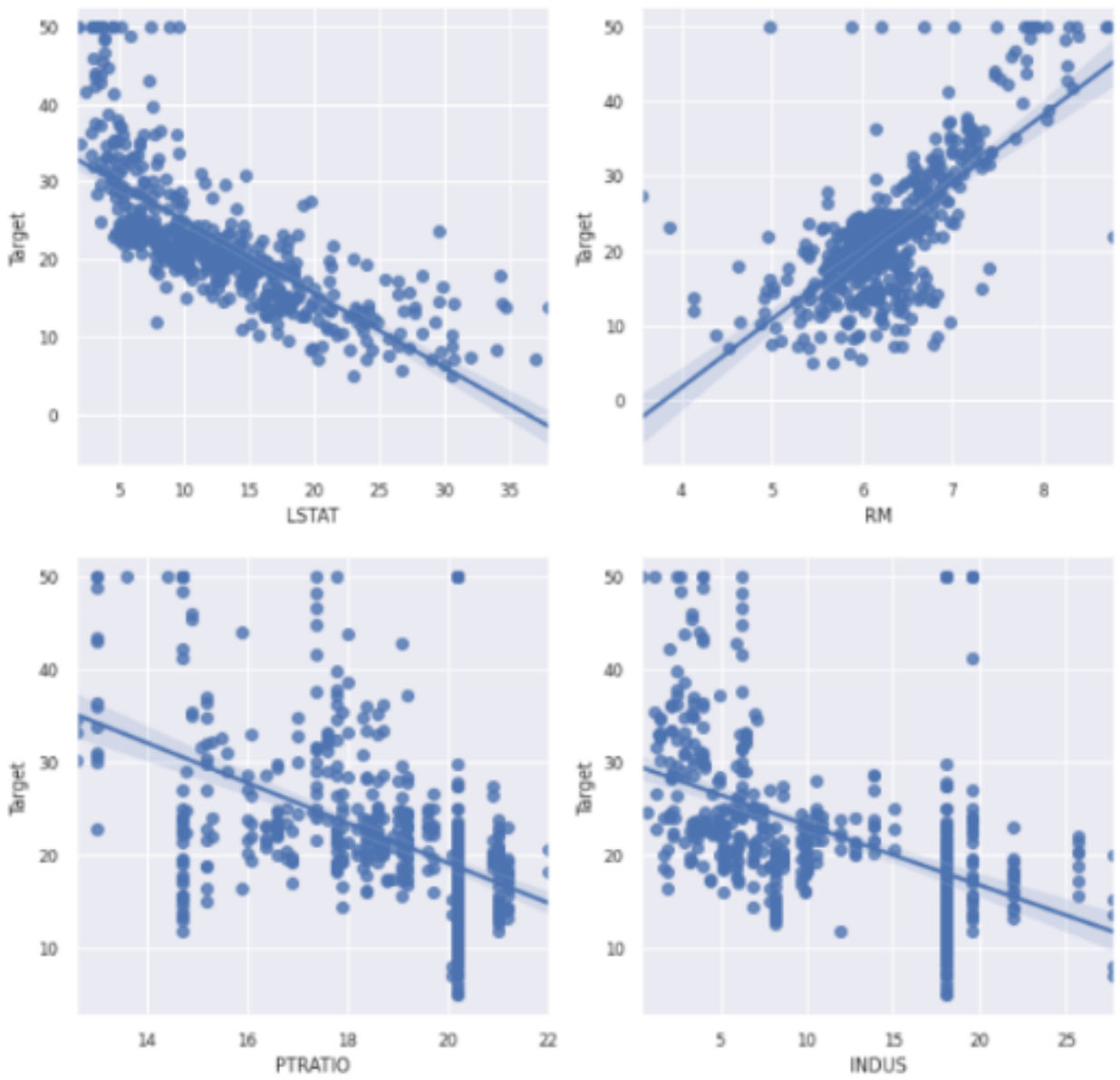
```
[11] #시각화로 분석할 피처 선택 추출
     plot_cols=['Target', 'LSTAT', 'RM', 'PTRATIO', 'INDUS']
     plot_df=df.loc[:, plot_cols]
     plot_df.head()
```

	Target	LSTAT	RM	PTRATIO	INDUS
0	24.0	4.98	6.575	15.3	2.31
1	21.6	9.14	6.421	17.8	7.07
2	34.7	4.03	7.185	17.8	7.07
3	33.4	2.94	6.998	18.7	2.18
4	36.2	5.33	7.147	18.7	2.18

피처 선택 추출

이젠 데이터분포를 살펴보자! 가장 좋은 방법은 그래프를 그려 시각화 하는 것이 좋다. Target 변수와 함께, 상관 계수 top4 LSTAT, RM, PTRATIO, INDUS를 추출했다.

```
#regplot으로 선형회귀선 표시
plt.figure(figsize=(10,10))
for idx, col in enumerate(plot_cols[1:]):
  ax1=plt.subplot(2, 2, idx+1)
  sns.regplot(x=col, y=plot_cols[0], data=plot_df, ax=ax1)
plt.show()
```

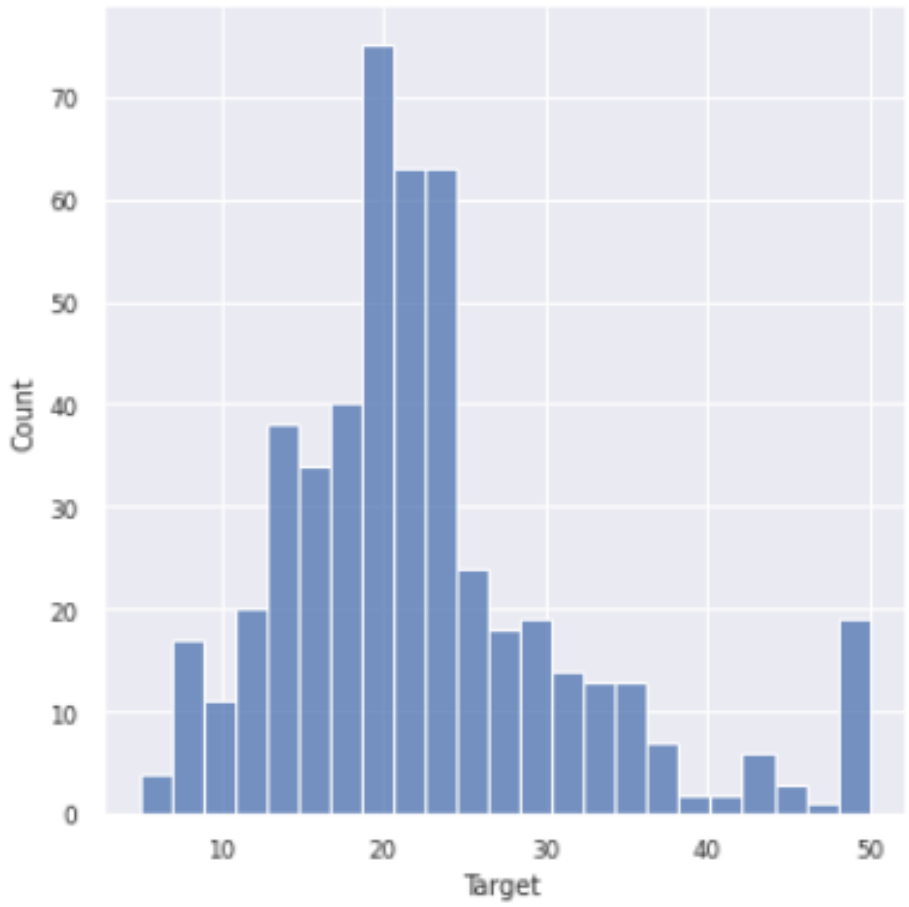


결과 창

~~사본~~ regplot 함수로 선형 회귀선을 산점도에 표시할 수 있다. x변수에는 target을 제외한 4개 피처를, y변수에는 targe
시본
t을 입력한다. subplot()으로 2x2 격자 프레임을 만들어주었다. LSTAT와 RM의 선형관계가 뚜렷하게 나타난다.

2-4. 목표 변수(Target 열) - 주택 가격

```
[13]  #Target 데이터 분포
      sns.displot(x='Target', kind='hist', data=df) #hist: 히스토그램으로 출력
      plt.show()
```



Target 데이터분포

Target 열의 주택가격 데이터분포를 displot()으로 그렸다. Target 열은 20을 중신으로 정규분포 형태를 보이며, 최대값에 해당하는 50에 많은 데이터가 분포되어 있는 특이점을 보인다.

3. 데이터 전처리

3-1. 피처 스케일링

각 피처(열)의 데이터 크기에 따른 상대적 영향력의 차이를 제거하기 위해, 피처의 크기를 비슷한 수준으로 맞춰주는 작업이 필요하다. 이 과정을 피처 스케일링(Feature Scaling)이라고 한다.

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()

df_scaled=df.iloc[:, :-1]   #마지막열임을 나타내는 -1은 포함하지 않음
scaler.fit(df_scaled)
df_scaled=scaler.transform(df_scaled)

#스케일링 변환된 값을 데이터프레임에 반영
df.iloc[:, :-1]=df_scaled[:, :]
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Target
0	0.000000	0.18	0.067815	0.0	0.314815	0.577505	0.641607	0.269203	0.000000	0.208015	0.287234	1.000000	0.089680	24.0
1	0.000236	0.00	0.242302	0.0	0.172840	0.547998	0.782698	0.348962	0.043478	0.104962	0.553191	1.000000	0.204470	21.6
2	0.000236	0.00	0.242302	0.0	0.172840	0.694386	0.599382	0.348962	0.043478	0.104962	0.553191	0.989737	0.063466	34.7
3	0.000293	0.00	0.063050	0.0	0.150206	0.658555	0.441813	0.448545	0.086957	0.066794	0.648936	0.994276	0.033389	33.4
4	0.000705	0.00	0.063050	0.0	0.150206	0.687105	0.528321	0.448545	0.086957	0.066794	0.648936	1.000000	0.099338	36.2

결과창

위는 사이킷런의 MinMaxScaler를 활용한 정규화방법이다. 목표변수 Target을 제외한 나머지 13개 열의 데이터를 iloc 인덱서로 추출한다. 이 데이터를 MinMaxScaler 인스턴스 객체에 fit()으로 각 열의 데이터를 최소값 0, 최대값 1 사이로 변환하는 식을 학습한다. transform()을 사용하면 학습한 변환식을 실제로 적용해 데이터를 정규화 변환한다.

3-2. 학습 데이터와 테스트 데이터 분할

```
#학습데이터와 테스트데이터 분할
from sklearn.model_selection import train_test_split
x_data=df.loc[:, ['LSTAT', 'RM']]
y_data=df.loc[:, 'Target']
x_train, x_test, y_train, y_test=train_test_split(x_data,
                                                  y_data,
                                                  test_size=0.2,
                                                  shuffle=True,
                                                  random_state=12)
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(404, 2) (404,)
(102, 2) (102,)
```

결과창

Target과 상관 계수가 가장 큰 LSTAT와 RM을 학습데이터 x_data로 선택한다. 그리고 506개의 주택 샘플 중 20%를 모델 평가에 사용한다. train_test_split()의 test_size 옵션에 0.2(20%)를 입력하면, 404개의 학습데이터 x_train, y_train과 102개의 테스트 데이터 x_test, y_test로 분할된다.

4. 베이스라인 모델-선형 회귀

```
#선형 회귀 모형
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train, y_train)

print("회귀계수(기울기):", np.round(lr.coef_, 1)) #np: numpy, coef_: 피처에 대한 회귀 계수 값
print("상수항(절편):", np.round(lr.intercept_, 1))  #intercept_: 상수항(절편) 값
```

```
회귀계수(기울기): [-23.2  25.4]
상수항(절편): 16.3
```

결과창

선형 회귀 모델의 예측력은 꽤 좋다. LinearRegression 클래스 객체를 생성하고, fit 메소드에 학습데이터인 x_train. y_train을 입력하면 선형 회귀식을 찾는다.

선형 회귀 모델의 coef_ 속성으로부터 각 피처에 대한 회귀 계수 값을 얻고 intercept_ 속성에서 상수항(절편)값을 얻는다. 이를 통해 LSTAT에 대한 회귀 계수는 -23.2이고, RM에 대한 회귀 계수는 25.4 임을 알 수 있다.

5. 모델 성능 평가

예측값과 실제값의 오차는 꽤 있는 편이다.

잔차(residuals): 실제값(정답)과 예측값의 차이로, 값이 작을 수록 모델의 성능이 좋다.

모델 성능을 평가할 때에는 수치화된 성능 지표를 사용한다. 회귀모델의 성능을 평가하는 지표는 MAE, MSE, RMSE 등이 있다. 사이킷런의 metrics 모듈에는 다양한 성능 평가 지표가 정의 되어 있다.

구분	설명
MAE(Mean Absolute Error)	실제값과 예측값의 차이 잔차의 절대값을 평균한 값
MSE(Mean Squared Error)	실제값과 예측값의 차이 잔차의 제곱을 평균한 값
RMSE(Root Mean Squared Error)	MSE의 제곱근

```
#성능 평가
from sklearn.metrics import mean_squared_error
y_train_pred=lr.predict(x_train)

train_mse=mean_squared_error(y_train, y_train_pred) #훈련 데이터의 평가 점수
print("Train MSE:%.4f" % train_mse)

test_mse=mean_squared_error(y_test, y_test_pred)
print("Test MSE:%.4f" % test_mse)
```

```
Train MSE:30.8042
Test MSE:29.5065
```

결과창

mean_squared_error(): 테스트오차(Test MSE)를 계산하는 함수

평가 지표 함수에 테스트 데이터의 실제값(y_test)와 예측값(y_test_pred)를 입력한다. 비교를 위해 훈련 데이터의 평가 점수인 Train MSE도 계산했다.

결과를 보면 Train MSE는 30.8, Test MSE는 29.5의 결과가 나왔다.