

Boston Housing Analysis Report

20. 11. 23.

안중호

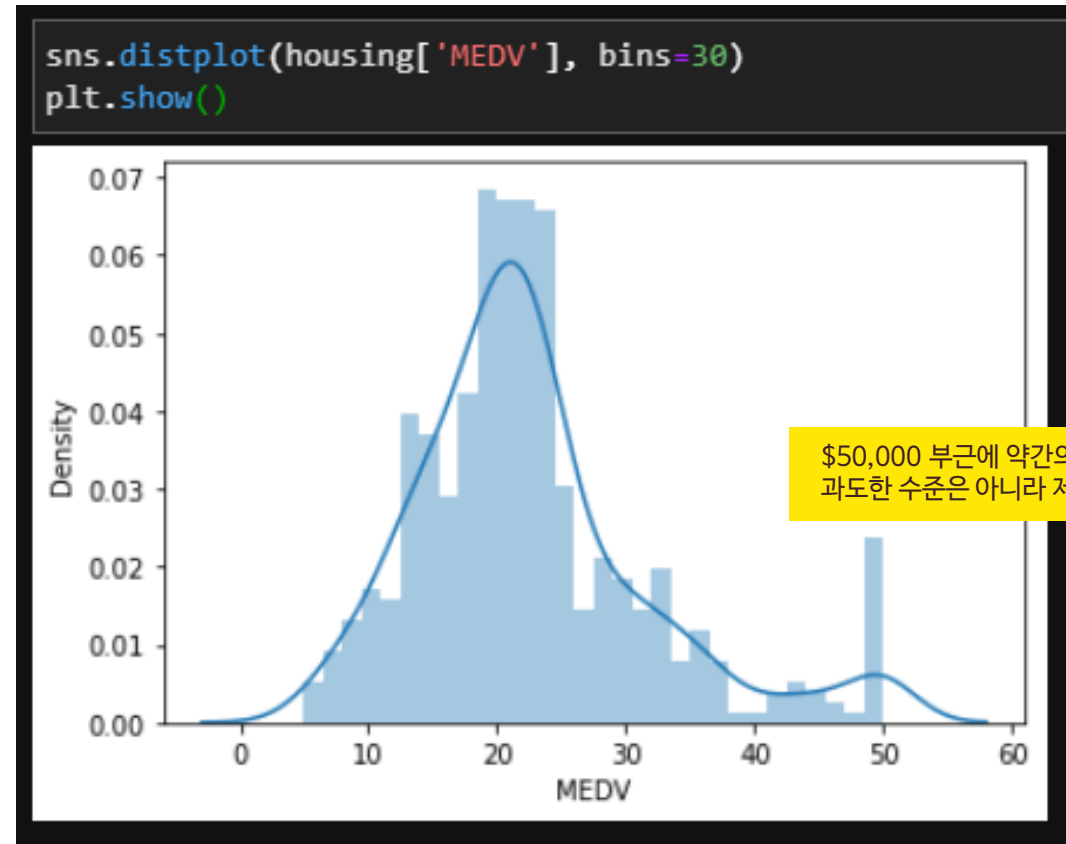
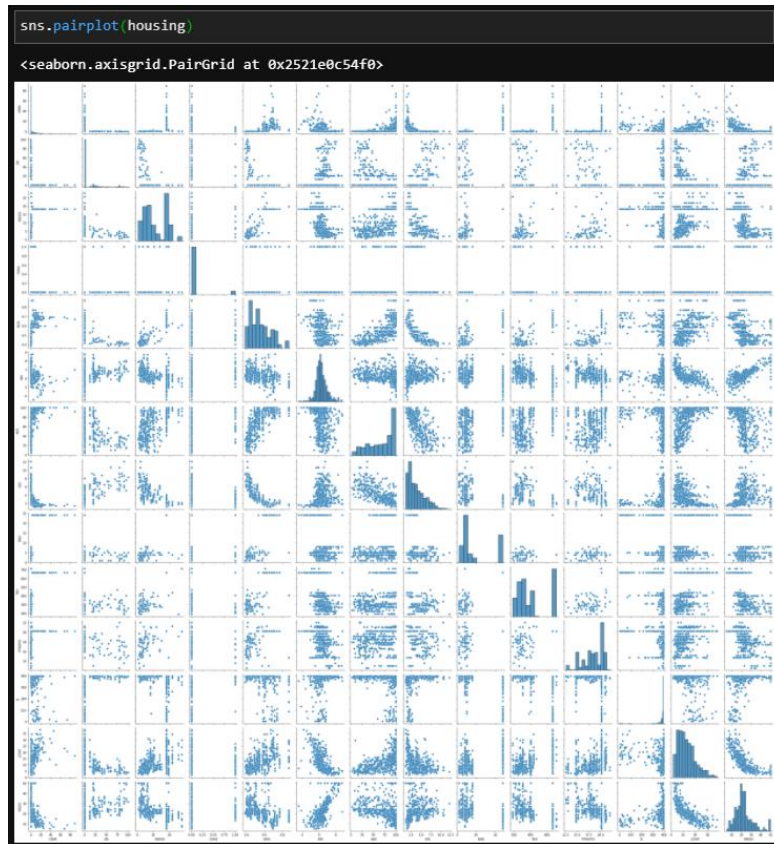
본 데이터는 총 14개의 특징이 있는 506개의 데이터로 구성되었다
보스턴 지역의 집값에 영향을 미치는 요인을 정리한 이 데이터를 기반으로
다양한 요소를 고려하며, 각 속성이 집값에 미치는 영향을 예측하고자 한다

housing.describe()														
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.6	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.1	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.7	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.9	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3	
75%	3.677082	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.9	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.9	

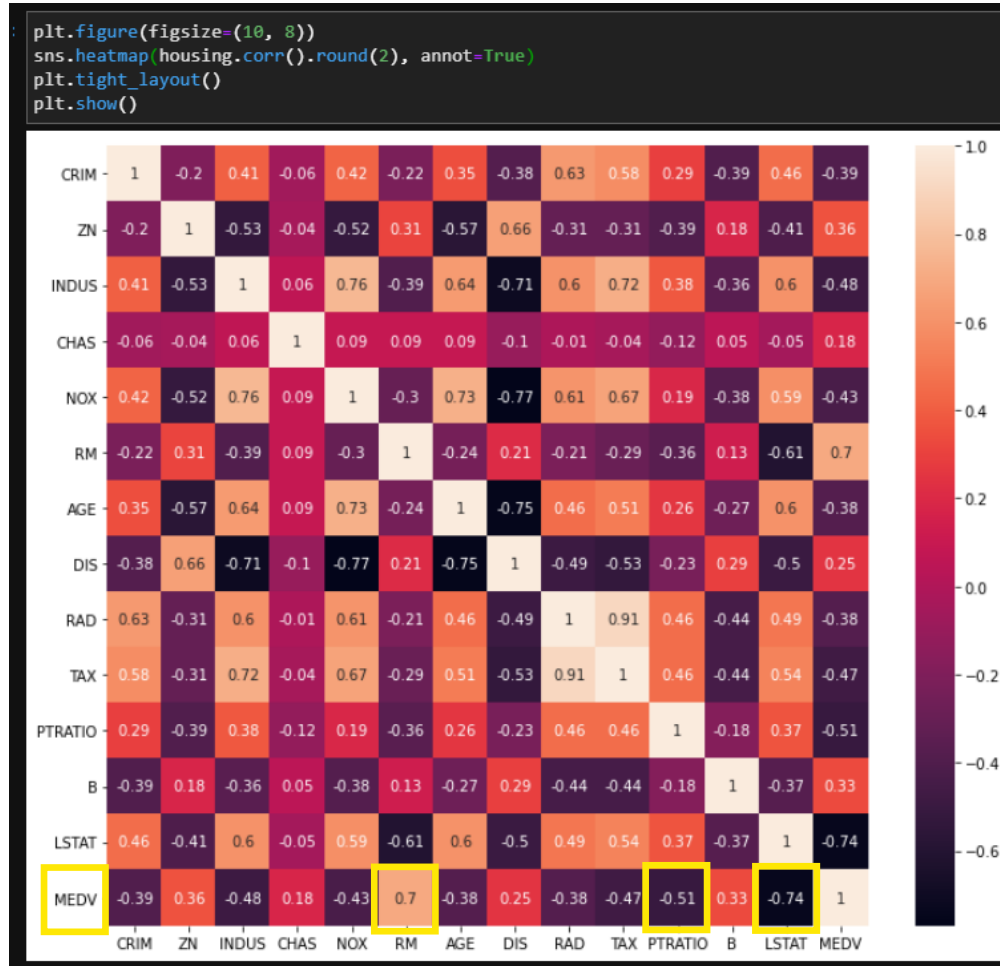
[01] CRIM	자치시(town) 별 1인당 범죄율
[02] ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03] INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04] CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05] NOX	10ppm 당 농축 일산화질소
[06] RM	주택 1가구당 평균 방의 개수
[07] AGE	1940년 이전에 건축된 소유주택의 비율
[08] DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09] RAD	방사형 도로까지의 접근성 지수
[10] TAX	10,000 달러 당 재산세를
[11] PTRATIO	자치시(town)별 학생/교사 비율
[12] B	$1000(Bk - 0.63)^2$, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13] LSTAT	모집단의 하위계층의 비율(%)
[14] MEDV	본인 소유의 주택가격(중앙값) (단위: \$1,000)

데이터 탐색 및 선형모델 적합성 검토

일부 편향성 및 변수간 약한 상호작용이 관찰되나, 심할 정도의 왜곡(Skew)은 보이지 않는다
목적변수인 주택가격은 정규분포에 가까워 선형회귀 모델을 적용하기에 적합하다
이 상태로 진행하기에 무리 없다고 판단, 변환 없이 바로 선형모델을 적용하기로 한다



주택가격과 RM(주택 1가구당 평균 방의 개수)은 강한 양의 상관관계,
LSTAT(하위 계층의 비율) 및 PTRATIO(학생-교사 비율)은 강한 음의 상관관계를 띤다
설명변수 후보들인데, 모델 설정시 어떤 기준으로 설명변수를 채택하는게 좋을까?



✓ 고려사항 1 : p-value의 존성 관련 논의

실질적 유의미성이 있다고 판단한 변수가 통계적으로 유의미하지 않다면 제외되어야 하는가?
이 오랜 질문에 2016년 미국통계학회는 p-value의 적용과 해석의 기준을 발표하였다
모델의 설명을 위해서는 선형회귀 방식이 필요하지만, 변수 선택에는 다른 식의 합의가 이루어질 수 있다

The ASA's Statement on *p*-Values: Context, Process, and Purpose

1. p-value는 데이터가 특정 통계 모형과 얼마나 상반되는지 나타낼 수 있다.
2. p-value는 연구가설이 참일 확률이나,
데이터가 무작위적 우연만으로 생성되었을 확률의 척도가 아니다.
3. 과학적 결론과 사업이나 정책적 결정이
p-value가 특정 문턱값을 넘어서는지에 의해서만 내려져서는 안 된다.
4. 적절한 추론을 위해 “완전한 보고와 투명성”이 필요하다.
5. p-value나 통계적 유의성은 “효과의 크기”나 “결과의 중요성”의 척도가 아니다.
6. p-value 그 자체만으로는 어떤 모형이나 가설에 관한 증거의 좋은 척도가 아니다.

✓ 고려사항 2: 'B' 특성 관련 이슈

흑인 비율이 주택 가격에 미치는 영향!?

데이터셋을 삭제할지 이 특성만 제거할지 의견이 분분했다(현재 deprecate 경고)
이러한 이슈를 반영, Base Model로 본 분석에서는 특성 'B'만 제외하여 사용하였다
(여러 역사적, 그리고 데이터적 의미를 담고자 다른 모델에서는 포함시켜 진행)


 [scikit-learn](#) / [scikit-learn](#)

$B = 1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town

What You Didn't Know About the Boston Housing Dataset

B Black proportion of population. At low to moderate levels of B , an increase in B should have a negative influence on housing value if Blacks are regarded as undesirable neighbors by Whites. However, market discrimination means that housing values are higher at very high levels of B . One expects, therefore, a parabolic relationship between proportion Black in a neighborhood and housing values.

1970 U. S. Census

 텐서 플로우 블로그 (Tensor ≈ Blog)

사이킷런의 `load_boston()` 함수가 삭제될 예정입니다.

B : Black proportion of population. (1970 US Census)

Waaaaaaaait a minute.

For pricing.

“B”

Just in case you've gotten this far without somehow paying attention, the column in question is called “B”:

Why not just drop “B” altogether? It sucks and the Census sucks.

1. Linear Base Model 구성 및 평가: RMSE, Variance score

Base 모형의 설명력은 68.9% (모든 변수를 사용했을 때보다 좋다!)
5 folds의 평균 RMSE(정밀도)는 5.773이다

```
y_target = housing['MEDV']
X_data = housing.drop(['MEDV', 'B'], axis=1, inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_data, y_target, test_size = 0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_train_preds = model.predict(X_train)
y_test_preds = model.predict(X_test)

# evaluating the model on training dataset
mse_train = mean_squared_error(y_train, y_train_preds)
rmse_train = np.sqrt(mse_train)

# evaluating the model on test dataset
mse_test = mean_squared_error(y_test, y_test_preds)
rmse_test = np.sqrt(mse_test)

print("The model performance for the training set")
print("-----")
print('MSE : {0:.3f}, RMSE : {1:.3F}'.format(mse_train, rmse_train))
print('Variance score : {0:.3f}'.format(r2_score(y_train, y_train_preds)))
print('\n')
print("The model performance for the test set")
print("-----")
print('MSE : {0:.3f}, RMSE : {1:.3F}'.format(mse_test, rmse_test))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_test_preds)))
```

```
The model performance for the training set
-----
MSE : 22.604, RMSE : 4.754
Variance score : 0.740
```

```
The model performance for the test set
-----
MSE : 22.778, RMSE : 4.773
Variance score : 0.689
```

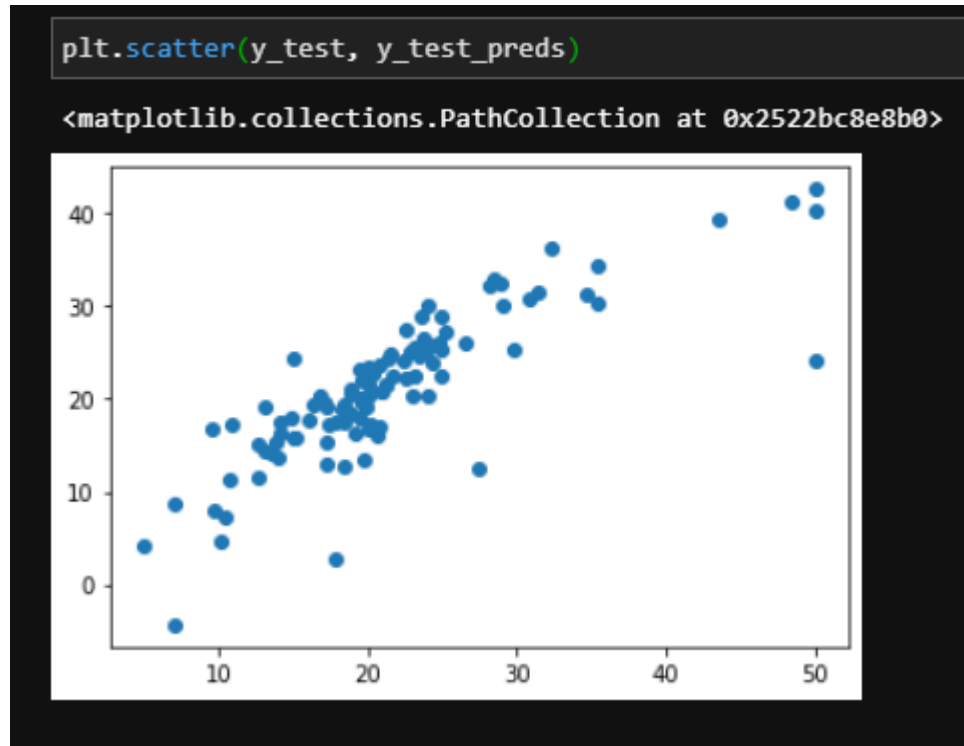
```
neg_mse_scores = cross_val_score(model, X_data, y_target, scoring='neg_mean_squared_error', cv=5)
rmse_scores = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print('5 folds의 개별 RMSE : ', np.round(rmse_scores, 2))
print('5 folds의 평균 RMSE : {0:.3f}'.format(avg_rmse))
```

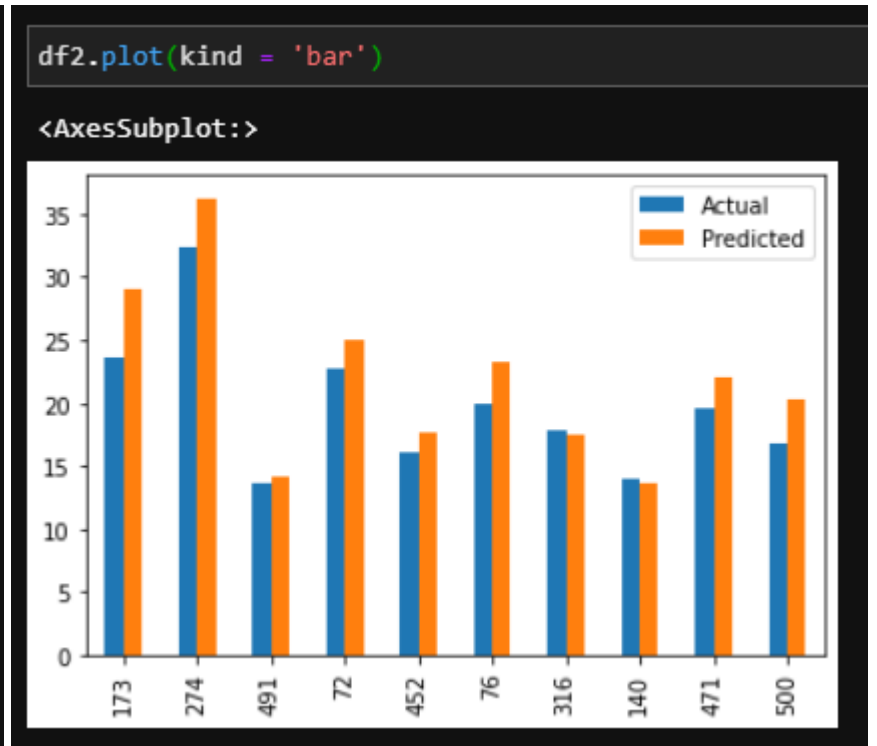
```
5 folds의 개별 RMSE : [3.54 5.14 5.86 8.78 5.54]
5 folds의 평균 RMSE : 5.773
```

1. Linear Base Model 구성 및 평가: Actual vs. Predicted

이것이 앞으로 만들어갈 모델과의 비교 기준이므로 면밀히 살펴보았다
Base Model도 괜찮은 수준의 퍼포먼스를 보여주고 있다

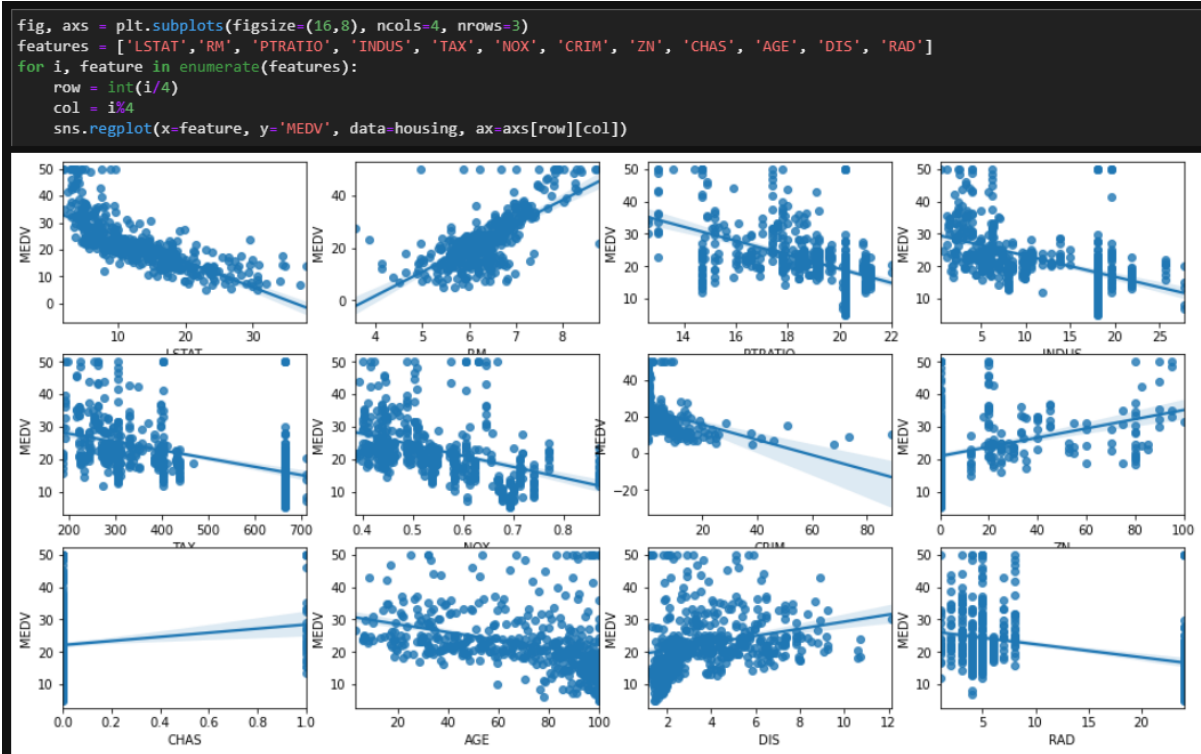


	Actual	Predicted
173	23.6	28.958215
274	32.4	36.287541
491	13.6	14.098836
72	22.8	25.033144
452	16.1	17.601968
76	20.0	23.294368
316	17.8	17.414978
140	14.0	13.688916
471	19.6	22.037299
500	16.8	20.300903



1. Linear Base Model 구성 및 평가: 시각화와 해석

선형모형에 비교적 잘 적합 되었으며, 설명도 납득할 만하다
성과와 설명력을 다잡을 수 있는 더 좋은 모델을 찾아보자



```
print('intercept : ', model.intercept_)
coeff = pd.Series(data=np.round(model.coef_, 3), index=X_data.columns)
coeff.sort_values(ascending=False)
```

intercept : 36.970469067029924

RM	4.240
CHAS	3.049
RAD	0.240
ZN	0.031
INDUS	0.025
AGE	-0.004
TAX	-0.011
CRIM	-0.127
LSTAT	-0.538
PTRATIO	-0.895
DIS	-1.466
NOX	-18.183

dtype: float64

주택당 방의 개수가 1 증가할 때마다 주택 가격은 \$4240 증가한다.

➤ 집이 크면 당연히 가격이 비싸다

찰스강변에 위치했다면 주택 가격은 \$3049 증가한다.

➤ 뷰가 좋으면 집값 높은 것은 1978년에도 그랬다

인구당 범죄율이 1씩 증가할 때마다 주택 가격은 \$127 감소한다.

➤ 우범지역의 집값이 싸다

학생-선생 비율이 1단위 증가할 때마다 주택 가격은 \$895 감소한다.

➤ 교육여건 지표이므로 타당하다

10ppm당 NO 비율이 1단위 증가할 때마다 주택 가격은 \$18183 감소한다.

➤ 일산화질소(NO)는 대기오염 지표이므로 타당하다! (공장지역 집값이 싸다)

2. 다항회귀모형 (2차): 구성 및 평가

현실의 문제는 직선이 아닌 다항의 방정식으로 표현할 때 더 나은 성능을 보일 수 있다
2차 선형 모델의 설명력은 80.7%로 개선되었다
하지만 5 folds 평균 RMSE가 5.829로 불안정하다. 과적합이 의심된다

```
from sklearn.preprocessing import PolynomialFeatures
import numpy as np

y_target = housing['MEDV']
X_data = housing.drop(['MEDV'], axis=1, inplace=False) ##

X_train, X_test, y_train, y_test = train_test_split(X_data, y_target, test_size = 0.2, random_state=42)

poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)

# fit the transformed features to Linear Regression
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)

# predicting on training data-set
y_train_preds = poly_model.predict(X_train_poly)
y_test_preds = poly_model.predict(poly_features.fit_transform(X_test))

# evaluating the model on training dataset
mse_train = mean_squared_error(y_train, y_train_preds)
rmse_train = np.sqrt(mse_train)

# evaluating the model on test dataset
mse_test = mean_squared_error(y_test, y_test_preds)
rmse_test = np.sqrt(mse_test)

print("The model performance for the training set")
print("-----")
print('MSE : {0:.3f}, RMSE : {1:.3f}'.format(mse_train, rmse_train))
print('Variance score : {0:.3f}'.format(r2_score(y_train, y_train_preds)))
print('\n')
print("The model performance for the test set")
print("-----")
print('MSE : {0:.3f}, RMSE : {1:.3f}'.format(mse_test, rmse_test))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_test_preds)))
```

The model performance for the training set

MSE : 5.314, RMSE : 2.305
Variance score : 0.939

The model performance for the test set

MSE : 14.184, RMSE : 3.766
Variance score : 0.807

```
neg_mse_scores = cross_val_score(poly_model, X_data, y_target, scoring='neg_mean_squared_error', cv=5)
rmse_scores = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)
```

```
print('5 folds의 개별 RMSE : ', np.round(rmse_scores, 2))
print('5 folds의 평균 RMSE : {0:.3f}'.format(avg_rmse))
```

5 folds의 개별 RMSE : [3.53 5.1 5.75 8.99 5.77]

5 folds의 개별 RMSE : [3.53 5.1 5.75 8.99 5.77]
5 folds의 평균 RMSE : 5.829

2. 다항 회귀 모형 (2차) : 변수의 증가와 다항변수 해석의 어려움

입력변수를 2차 다항 값으로 변환하며 해석해야 할 변수 13개가 105개로 증가했다
또한, <범죄율 * 일산화질소 농도> 항을 어떻게 설명해야 하는가?
다항 모형의 문제점은 이것이다. 분석은 했지만 해석을 하는 것이 매우 어려워진다

설명변수의 개수가 증가하는 문제

```
print('설명변수 갯수 :', len(poly_model.coef_))
print('2차 다항 회귀 계수\n', np.round(poly_model.coef_, 2))
```

설명변수 갯수 : 105
2차 다항 회귀 계수

-4.92070955e+00	7.23000000e+00	6.60000000e-01	-4.72000000e+00
3.67400000e+01	2.85260000e+02	1.78400000e+01	6.60000000e-01
-4.11000000e+00	2.66000000e+00	-1.00000000e-02	7.65000000e+00
1.30000000e-01	-2.00000000e-01	0.00000000e+00	7.00000000e-02
5.80000000e-01	2.59000000e+00	-2.03000000e+00	1.80000000e-01
-1.00000000e-02	-1.90000000e-01	2.70000000e-01	-4.00000000e-02
6.90000000e-01	-0.00000000e+00	3.00000000e-02	-0.00000000e+00
-0.00000000e+00	-6.00000000e-02	-1.38000000e+00	-0.00000000e+00
0.00000000e+00	-2.00000000e-02	-2.00000000e-02	0.00000000e+00
-0.00000000e+00	0.00000000e+00	-1.00000000e-02	5.00000000e-02
-1.00000000e-02	-2.20000000e-01	3.10000000e-01	0.00000000e+00
1.70000000e-01	-0.00000000e+00	-0.00000000e+00	-3.00000000e-02
0.00000000e+00	-2.00000000e-02	3.67400000e+01	-3.67200000e+01
-5.27000000e+00	-3.00000000e-02	-9.20000000e-01	1.30000000e-01
-1.00000000e-02	-8.40000000e-01	1.00000000e-02	-2.40000000e-01
-9.84100000e+01	-8.71000000e+00	-5.00000000e-02	1.48000000e+01
-2.08000000e+00	2.80000000e-01	-1.45000000e+01	0.00000000e+00
1.01000000e+00	1.13000000e+00	-7.00000000e-02	-3.30000000e-01
-1.80000000e-01	-1.00000000e-02	-5.60000000e-01	-1.00000000e-02

2차항 회귀계수 해석의 문제

<범죄율*학생/교사비율> 항의해석??
<하층민비율*고용중심지접근성> 항의해석??
<방의개수*찰스강변> 항의해석??
<1940년이전건축*도로접근성> 항의해석??
<일산화질소*방의개수> 항의해석??
<재산세율*비소매상업지역토지비율> 항의해석??

.
. .
. .
??????

3. 다항 회귀 모형 (3차) : 과적합 발생의 문제

3차 다항모델의 경우 엄청난 오버피팅이 확실하게 발생했다
다항 회귀는 차수가 높아질수록 학습 데이터에만 맞춘 학습이 이루어져
테스트 데이터에서는 정확도가 떨어진다. 이 경우, 해석 이전에 모델의 예측력 자체가 없다

```
poly_features = PolynomialFeatures(degree=3)
X_train_poly = poly_features.fit_transform(X_train)

# fit the transformed features to Linear Regression
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)

# predicting on training data-set
y_train_preds = poly_model.predict(X_train_poly)
y_test_preds = poly_model.predict(poly_features.fit_transform(X_test))

# evaluating the model on training dataset
mse_train = mean_squared_error(y_train, y_train_preds)
rmse_train = np.sqrt(mse_train)

# evaluating the model on test dataset
mse_test = mean_squared_error(y_test, y_test_preds)
rmse_test = np.sqrt(mse_test)

print("The model performance for the training set")
print("-----")
print('MSE : {0:.3f}, RMSE : {1:.3f}'.format(mse_train, rmse_train))
print('Variance score : {0:.3f}'.format(r2_score(y_train, y_train_preds)))
print('\n')
print("The model performance for the test set")
print("-----")
print('MSE : {0:.3f}, RMSE : {1:.3f}'.format(mse_test, rmse_test))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_test_preds)))
```

The model performance for the training set

MSE : 0.000, RMSE : 0.000
Variance score : 1.000

The model performance for the test set

MSE : 129848.063, RMSE : 360.344
Variance score : -1769.645

4. 규제 선형 모델 (Lasso) : 설명 및 채택 이유

학습 데이터의 잔차 오류 최소화와 과적합 방지의 균형을 잡는 것이 규제 모델이다
의사 결정을 돕기 위해, '집 가격에 영향을 미치는 중요한 원인에 집중' 할 목적이므로
규제 모델 중 Feature Selection 기능이 있는 Lasso 모델을 적용한다

다양한 요인을 전부 고려?



Lasso - L1 규제

W의 절대값에 대해 패널티를 부여해
영향력이 크지 않은 회귀 계수 값을 0으로 변환

Feature Selection

비용 함수 목표 = $\text{Min}(\alpha * ||W||_1)$

선택과 집중을 통해 의사 결정에 도움!



4. 규제선형모델 (Lasso): 구성 및 평가

alpha 0.07일 경우가 RMSE 5.612로 예측 성능이 가장 좋다
Base Model 5 folds의 평균 RMSE(5.773)보다 향상되었다

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet

y_target = housing['MEDV']
X_data = housing.drop(['MEDV'], axis=1, inplace=False)

X_train, X_test, y_train, y_test = train_test_split(X_data, y_target, test_size = 0.2, random_state=42)

# alpha 값에 따른 회귀 모델의 평균 RMSE를 출력하고 회귀 계수값들을 DataFrame으로 반환하는 함수
def get_linear_reg_eval(model_name, params=None, X_data_n=None, y_target_n=None, verbose=True):
    coeff_df = pd.DataFrame()
    if verbose: print('#### ', model_name, '####')
    for param in params:
        if model_name == 'Ridge': model = Ridge(alpha=param)
        elif model_name == 'Lasso': model = Lasso(alpha=param)
        elif model_name == 'ElasticNet': model = ElasticNet(alpha=param, l1_ratio=0.7)
        neg_mse_scores = cross_val_score(model, X_data_n, y_target_n, scoring='neg_mean_squared_error', cv=5)
        avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
        print('alpha {0}일때 5 폴드 세트의 평균 RMSE: {1:.3f}'.format(param, avg_rmse))
        # cross_val_score는 evaluation metric만 반환하므로 모델을 다시 학습해 회귀 계수 추출
        model.fit(X_data, y_target)
        # alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이를 DataFrame의 컬럼으로 추가
        coeff = pd.Series(data=model.coef_, index=X_data.columns)
        colname = 'alpha:' + str(param)
        coeff_df[colname] = coeff
    return coeff_df

lasso_alphas = [0.07, 0.1, 0.5, 1, 3]
coeff_lasso_df = get_linear_reg_eval('Lasso', params=lasso_alphas, X_data_n=X_data, y_target_n=y_target)
```

```
#### Lasso ####
alpha 0.07일때 5 폴드 세트의 평균 RMSE: 5.612
alpha 0.1일때 5 폴드 세트의 평균 RMSE: 5.615
alpha 0.5일때 5 폴드 세트의 평균 RMSE: 5.669
alpha 1일때 5 폴드 세트의 평균 RMSE: 5.776
alpha 3일때 5 폴드 세트의 평균 RMSE: 6.189
```

4. 규제 선형 모델 (Lasso) : 예측 결과의 해석

성능은 향상되었으나 해석은 의문이다 (Lasso는 '불필요'한 회귀계수를 급격히 감소시킨다!)
Base Model에서는 의미 있게 다루었던 NOX가 빠진 것은 납득이 어렵다
모델은 성능은 물론이고, 결과의 해석도 실체와 맞닿아 있어야 한다

```
sort_column = 'alpha:' + str(lasso_alphas[0])
coeff_lasso_df.sort_values(by=sort_column, ascending=False)
```

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
RM	3.789725	3.703202	2.498212	0.949811	0.000000
CHAS	1.434343	0.955190	0.000000	0.000000	0.000000
RAD	0.270936	0.274707	0.277451	0.264206	0.061864
ZN	0.049059	0.049211	0.049544	0.049165	0.037231
B	0.010248	0.010249	0.009469	0.008247	0.006510
NOX	-0.000000	-0.000000	-0.000000	-0.000000	0.000000
AGE	-0.011706	-0.010037	0.003604	0.020910	0.042495
TAX	-0.014290	-0.014570	-0.015442	-0.015212	-0.008602
INDUS	-0.042120	-0.036619	-0.005253	-0.000000	-0.000000
CRIM	-0.098193	-0.097894	-0.083289	-0.063437	-0.000000
LSTAT	-0.560431	-0.568769	-0.656290	-0.761115	-0.807679
PTRATIO	-0.765107	-0.770654	-0.758752	-0.722966	-0.265072
DIS	-1.176583	-1.160538	-0.936605	-0.668790	-0.000000

주택당 방의 개수가 1 증가할 때마다 주택 가격은 \$3789 증가한다.

➢ 집이 크면 당연히 가격이 비싸다

찰스강변에 위치했다면 주택 가격은 \$1434 증가한다.

➢ 뷰가 좋으면 집값 높은 것은 1978년에도 그랬다

인구당 범죄율이 1씩 증가할 때마다 주택 가격은 \$98 감소한다.

➢ 우범지역의 집값이 싸다

학생-선생 비율이 1단위 증가할 때마다 주택 가격은 \$765 감소한다.

➢ 교육여건 지표이므로 타당하다

5대 중심가와외의 가중거리가 1단위 멀어지면 주택 가격은 \$1176 감소한다.

➢ 주택 구매시 출퇴근 편의성을 고려하므로 타당하다!

그러나, 10ppm당 NO 비율의 증가는
주택 가격에 영향을 주지 않는다?
NO 비율은 주택 가격 예측에 '불필요'하다?

5. 회귀트리: Decision Tree vs. Random Forest vs. GBM vs. XGBoost vs. LightGBM

선형 회귀는 회귀 함수에서 계수의 관계를 모두 선형으로 가정한다
회귀 트리는 회귀 함수 대신 트리를 기반으로, 리프 노드 값의 평균을 통해 회귀 예측값을 계산한다
가장 성능이 좋은 GradientBoostingRegressor를 채택하였다

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

def get_model_cv_prediction(model, X_data, y_target):
    neg_mse_scores = cross_val_score(model, X_data, y_target, scoring='neg_mean_squared_error', cv=5)
    rmse_scores = np.sqrt(-1 * neg_mse_scores)
    avg_rmse = np.mean(rmse_scores)
    print('##### ', model.__class__.__name__, ' #####')
    print('5 folds의 평균 RMSE : {0:.3f} '.format(avg_rmse))

dt_reg = DecisionTreeRegressor(random_state=42, max_depth=4)
rf_reg = RandomForestRegressor(random_state=42, n_estimators=1000)
gb_reg = GradientBoostingRegressor(random_state=42, n_estimators=1000)
xgb_reg = XGBRegressor(n_estimators=1000)
lgb_reg = LGBMRegressor(n_estimators=1000)

models = [dt_reg, rf_reg, gb_reg, xgb_reg, lgb_reg]
for model in models:
    get_model_cv_prediction(model, X_data, y_target)
```

```
##### DecisionTreeRegressor #####
5 folds의 평균 RMSE : 6.070
##### RandomForestRegressor #####
5 folds의 평균 RMSE : 4.429
##### GradientBoostingRegressor #####
5 folds의 평균 RMSE : 4.229
##### XGBRegressor #####
5 folds의 평균 RMSE : 4.251
##### LGBMRegressor #####
5 folds의 평균 RMSE : 4.646
```


5. 회귀트리: Gradient Boosting 결과 설명

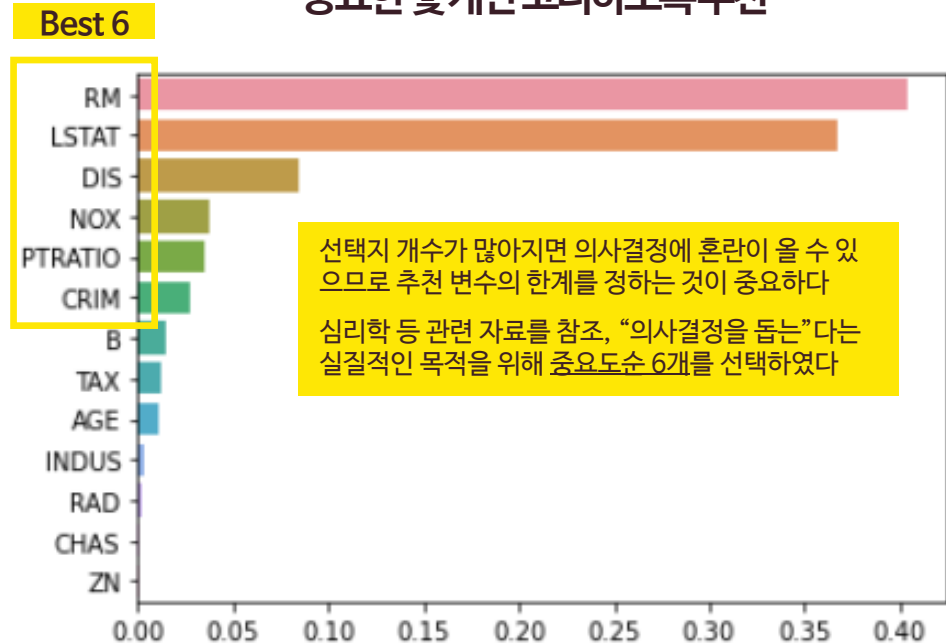
회귀 계수가 없으므로, ‘입력변수 얼마 변화당 출력변수는 얼마 변화한다’ 식의 설명이 불가능하다
대신 모델 피처별 중요도를 제공하며, 이는 ‘중요한 우선순위’를 선택하는데 유용하다

고려할 선택지가 많으면 의사결정 어려움



VS

중요한 몇 개만 고려하도록 추천



6. 분석의 결론: Boston Housing Analysis

Two-Step Regression

- ✓ 트리 모델의 feature importance 기준으로 변수를 선택하고
- ✓ 다중회귀로 반응변수의 변화가 목적변수에 미치는 영향 설명

Gradient Boost

Feature Importance

```
RM      0.404314
LSTAT   0.367514
DIS      0.084744
NOX      0.037223
PTRATIO  0.034440
CRIM     0.027594
dtype: float64
```

Linear Regression

Regression Coeff.

```
RM      4.654
CRIM     -0.099
LSTAT   -0.549
PTRATIO -0.924
DIS     -1.255
NOX    -16.625
dtype: float64
```

Two-Step Regression

- ✓ 중요한관심사를먼저알리고, 이어 그효과를 설명하는인간친화적서술방식
- ✓ p - value의불안정성, 비선형성/상호작용에 취약, 통계적유의성의한계 보완

Gradient Boost

Linear Regression

“뭐가 집값에 영향을 주나요?”

“중요한 순서대로
1가구당 방 개수,
저소득 주민 비율,
5대 고용 중심들로 부터의 거리,
일산화질소 농도,
학생-선생 비율,
그리고 범죄 발생률입니다.”

“그럼 집값은 어떻게 달라지나요?”

“가구당 방이 1개 증가할 때마다 **\$4654** 증가합니다.”
“저소득 주민의 비율이 1% 증가할 때마다 **\$549** 감소합니다.”
“5대 중심가와의 거리가 1단위 멀어지면 **\$1255** 감소합니다.”
“일산화질소 농도가 1단위 증가할 때마다 **\$16625** 감소합니다.”
“학생-선생 비율이 1단위 증가할 때마다 **\$549** 감소합니다.”
“인구당 범죄율이 1씩 증가할 때마다 **\$99** 감소합니다.”

7. 모형 검증 1: 변수와 모형의 안정성 확인

본 모델에 사용한 train, test 데이터를 csv로 제작해 통계 검증 진행한 결과
모델과 변수에 특별한 문제 없다 (유의하다라는 말을 사용하기 조심스럽다)

```
housing_model_train = pd.concat([X_train, y_train], axis=1)
housing_model_test = pd.concat([X_test, y_test], axis=1)

housing_model_train.to_csv('housing_model_train.csv', header=True, index=False)
housing_model_test.to_csv('housing_model_test.csv', header=True, index=False)
```



R

```
housing_train <- read_csv('housing_model_train.csv')
housing_test <- read_csv('housing_model_test.csv')

model <- lm(MEDV ~ RM + LSTAT + DIS + NOX + PTRATIO + CRIM, data=housing_train)
summary(model)
```

Call:
lm(formula = MEDV ~ RM + LSTAT + DIS + NOX + PTRATIO + CRIM,
data = housing_train)

Residuals:

Min	1Q	Median	3Q	Max
-12.7791	-2.6923	-0.5925	1.7640	28.9991

Coefficients:

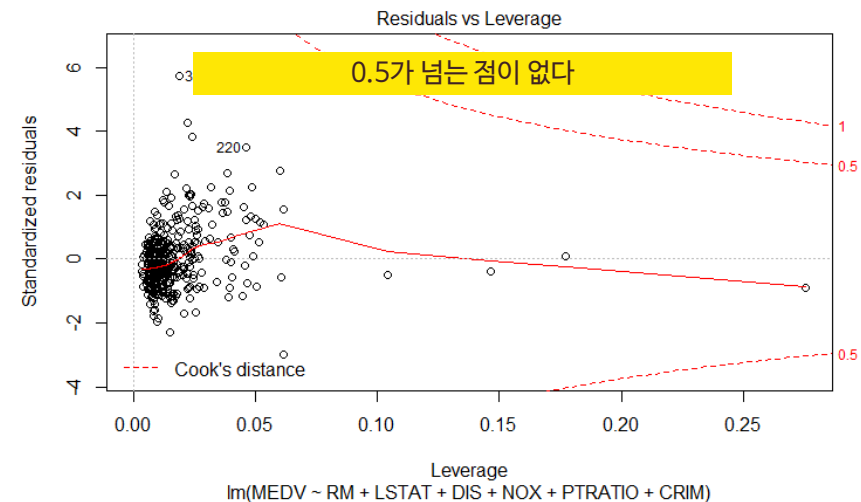
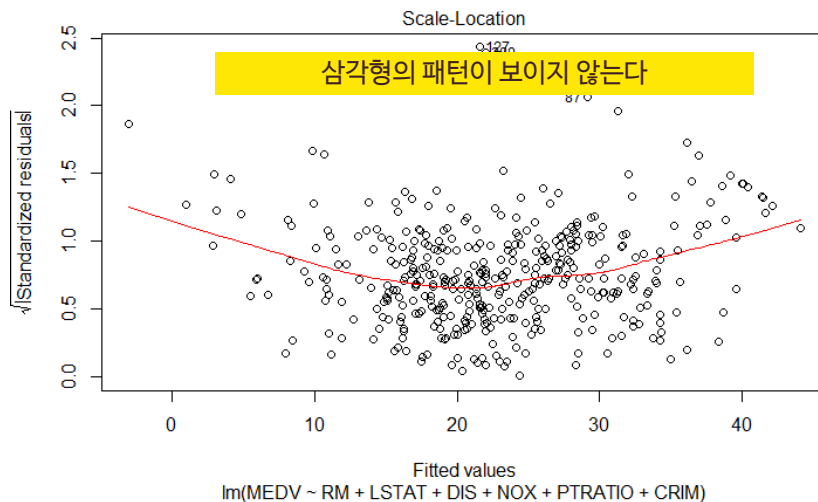
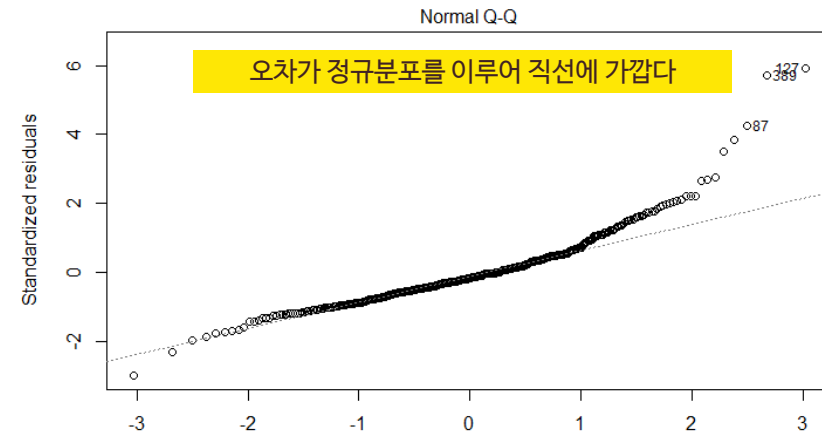
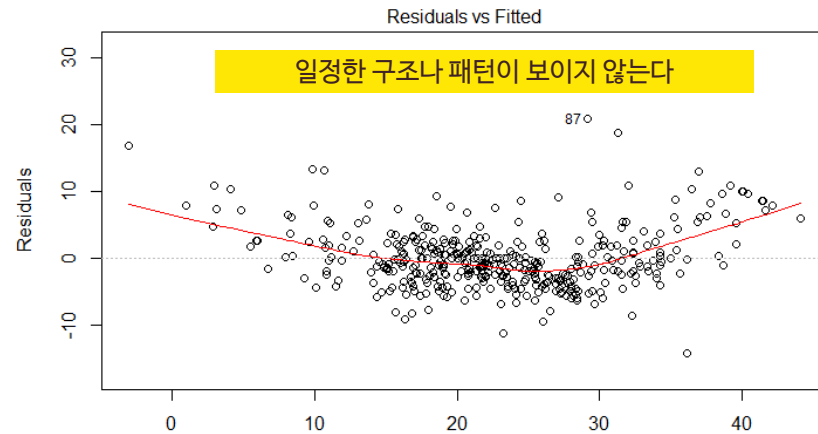
	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	33.65310	4.69205	7.172	3.62e-12	***
RM	4.57458	0.42479	10.769	< 2e-16	***
LSTAT	-0.49365	0.05254	-9.395	< 2e-16	***
DIS	-1.12125	0.17205	-6.517	2.17e-10	***
NOX	-18.30390	3.39648	-5.389	1.22e-07	***
PTRATIO	-1.03756	0.11989	-8.654	< 2e-16	***
CRIM	-0.08369	0.03191	-2.623	0.00905	**

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.572 on 398 degrees of freedom
Multiple R-squared: 0.7419, Adjusted R-squared: 0.738
F-statistic: 190.7 on 6 and 398 DF, p-value: < 2.2e-16

7. 모형 검증 2: 분석의 전제조건 확인

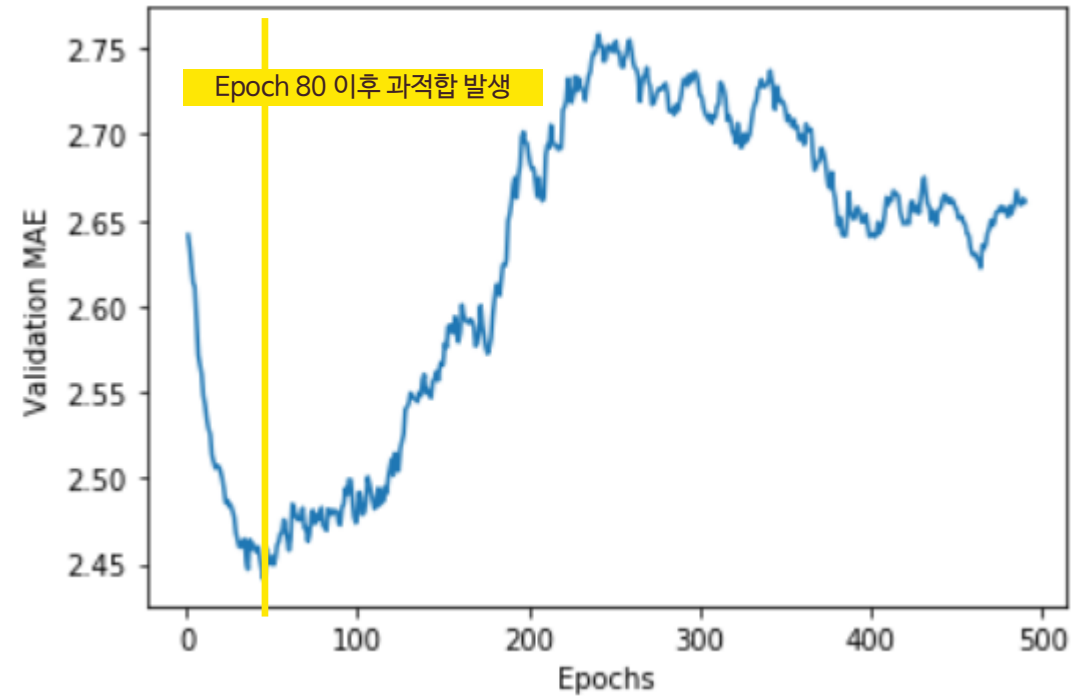
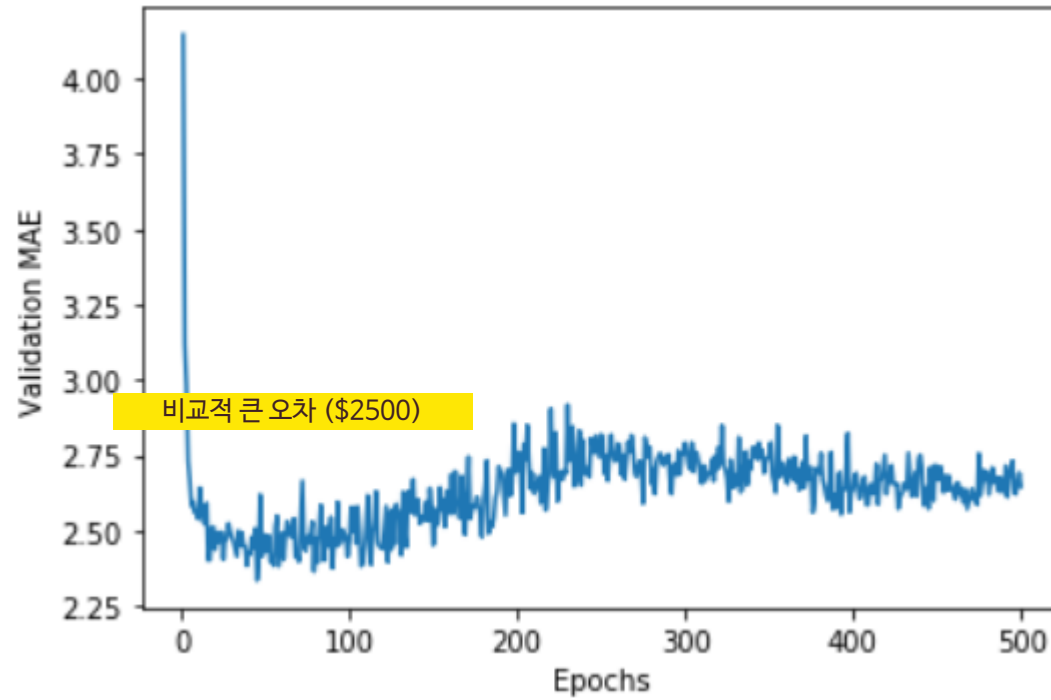
등분산성, 오차의 정규분포 가정을 만족하며
적합 값 증가에 따른 잔차의 퍼짐 및 영향점이 발견되지 않아 문제 없다



#참고: 딥러닝 모델의 적용 검토

본 분석과 별도로 딥러닝 모델을 적용해 비교한 결과
오차가 크며 성능의 개선이 없어 딥러닝의 장점이나 실익이 없었다
저차원, 소용량 데이터에는 딥러닝의 장점이 무색해진다
딥러닝을 사용하기 전에 다루고 있는 문제의 크기를 고려해야 한다

*Boston Housing은 14개의 특징, 506개 데이터셋으로 구성



(64개의 유닛을 가진 2개의 은닉층으로 네트워크 구성, 활성화 함수로 ReLU, 손실함수로 MSE 사용)

EOD