



파이썬 머신러닝 완벽 가이드 - 회귀



목차

1. 회귀 소개
2. 단순 선형 회귀를 통한 회귀 이해
3. 비용 최소화하기 - 경사하강법(Gradient Descent) 소개
4. 사이킷런 LinearRegression을 이용한 보스턴 주택 가격 예측
5. 다항 회귀와 과적합/과소적합 이해
6. 규제 선형 모델 - 릿지, 라쏘, 엘라스틱넷

1. 회귀 소개

a. 회귀란?

- 통계학 용어로 회귀는 여러 개의 독립변수와 한 개의 종속변수 간의 상관관계를 모델링하는 기법이다.
- 분류와 달리 예측값이 연속형 숫자 값을 가진다.
- ex) 아파트의 방 개수, 방 크기, 주변 학군 등에 따라 아파트 가격이 어떤 관계를 관 계를 가지는지 확인

y 는 종속변수(아파트 가격)
 x 는 독립변수(방 개수, 주변 학군 등)
 β 는 독립변수의 값에 영향을 미치는 회귀 계수

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon$$

- 머신러닝 관점에서 독립변수는 피쳐, 종속변수는 결정값.
- 회귀예측의 핵심은 주어진 피쳐와 결정값 데이터 기반에서 학습을 통해 최적의 회귀 계수를 찾아내는 것

b. 회귀 유형 구분

- 회귀계수의 선형/비선형 여부

회귀 계수의 결합	식
선형 회귀	$(y = \beta_0 + \beta_1 x_1 + \varepsilon)$
비선형 회귀	$(y = \alpha \ln \beta_1 x + \varepsilon /$ $y = \alpha e^{\beta_1 x} + \varepsilon /$ $y = \alpha \sin \beta_1 x + \varepsilon \dots)$

- 독립변수의 개수

독립변수 개수	식
단일 회귀	$y = \beta_0 + \beta_1 x_1 + \varepsilon$
다중 회귀	$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon$

- 종속변수의 개수

독립변수 개수	식
<u>단변량 회귀</u>	-
<u>다변량 회귀</u>	$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon_i$

c. 선형회귀 모델 종류

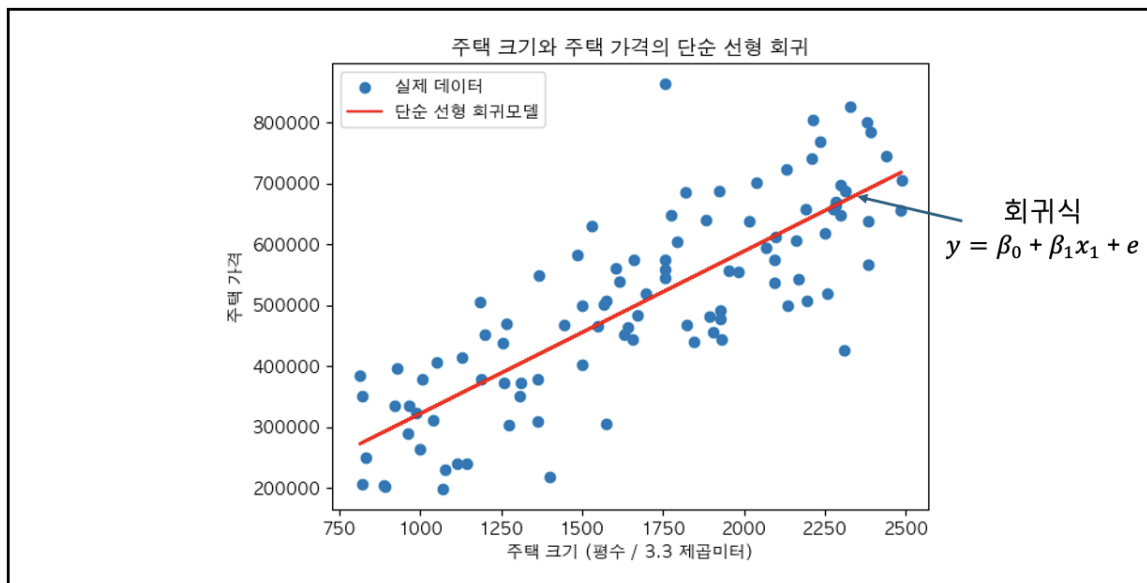
- 단순선형회귀 : 잔차제곱합(RSS - Residual Sum of Squares)를 최소화할 수 있도록 회귀 계수를 최적화하며, 규제를 적용하지 않은 모델.
- 릿지(Ridge) : 선형 회귀에 L2 규제를 추가한 회귀 모델.
 - L2 규제는 상대적으로 큰 회귀 계수 값의 예측 영향도를 감소시키기 위해서 회귀 계수값을 더 작게 만드는 규제 모델
- 라쏘(Lasso) : 선형 회귀에 L1 규제를 적용한 회귀 모델.
 - L1 규제는 예측 영향력이 낮은 피처의 회귀 계수를 0으로 만들어 회귀 예측 시 피처가 선택되지 않게 하는 규제 모델
- 엘라스틱넷(ElasticNet) : L2, L1 규제를 함께 결합 모델. 주로 피처가 많은 데이터 세트에 적용되며, L1 규제로 피처의 개수를 줄임과 동시에 L2 규제로 계수 값의 크기를 조정함.
- 로지스틱 회귀(Logistic Regression) : 선형회귀(Linear Regression)에 로지스틱 함수를 적용한 모델. 일반적으로 이진 분류뿐만 아니라 희소 영역의 분류(ex. 텍스트 분류)에서 뛰어난 예측 성능을 보임.

2. 단순 선형 회귀를 통한 회귀 이해

a. 예시 - 주택의 크기에 따라 주택 가격이 결정된다는 가정

- 그래프

- β_0 : y절편(intercept)이라고 하며, 회귀식이 y축과 만나는 점을 의미함.
- β_1 : 회귀 계수(기울기, slope)이라고 하며, 독립변수 x의 변화에 따라 종속변수 y가 반응하는 정도를 의미함.
- ε : 오차(error)이라고 하며, 모집단으로부터 추정한 회귀식으로부터 얻은 예측값과 실제 관측값의 차이를 의미함.
- e : 잔차(residual)이라고 하며, 표본(sample)으로 추정한 회귀식과 실제 관측값의 차이이자 오차의 추정치를 의미함.
- 최적의 회귀모델을 만든다는 것은 전체 데이터의 잔차(오류 값) 합이 최소가 되는 모델을 만든다는 의미이자 동시에 오류 값 합이 최소가 될 수 있는 최적의 회귀 계수를 찾는다는 의미.



b. 잔차제곱합(RSS - Residual Sum of Square)

- RSS에 대한 설명
 - 회귀모델의 비용함수.
 - 회귀모델의 잔차들의 제곱을 모두 합한 것으로 작을 수록 예측이 잘 되었다고 볼 수 있음.
 - 그만큼 예측한 값들이 관측값들에서 멀리 떨어지지 않았다는 것을 의미.

< 단순선형회귀에서의 분산분석표 >

요인	제곱합	자유도	평균 제곱	F 통계량
회귀	$SSR = \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$	1	$MSR = \frac{SSR}{1}$	$F = \frac{MSR}{MSE}$
잔차	$SSE = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$	n - 2	$MSE = \frac{SSE}{n-2}$	
전체	$SST = \sum_{i=1}^n (Y_i - \bar{Y})^2$	n - 1		

(i는 1부터 학습 데이터의 총 건수 n 까지)

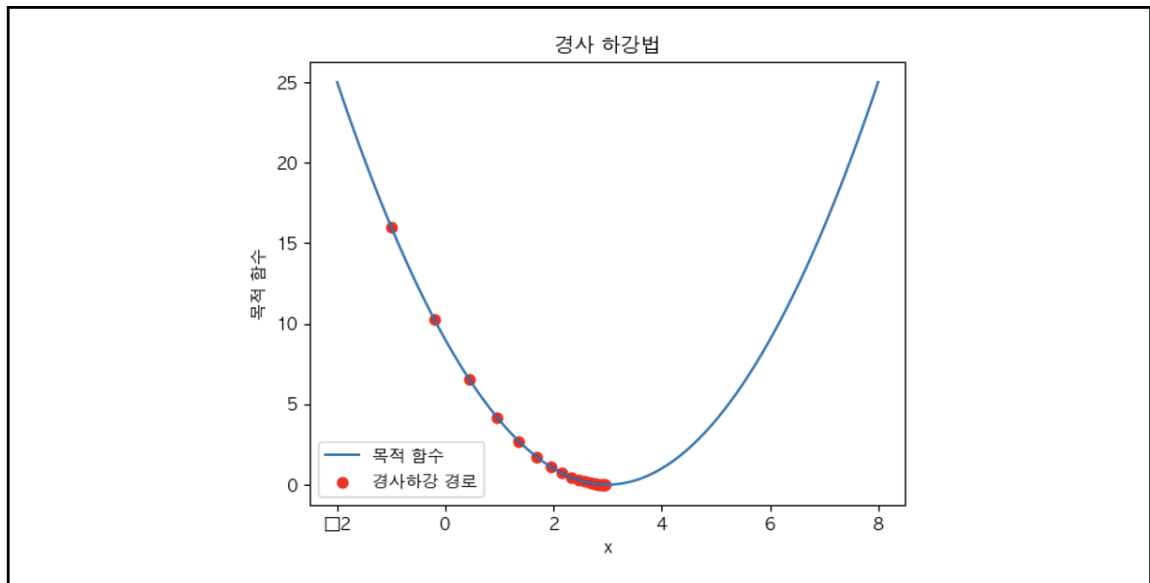
- 총 제곱합(SST)는 종속 변수의 변동을 나타내며, 회귀 모델에 대한 제곱합(SSR)은 회귀 모델에 의해 설명되는 종속 변수의 변동을, 잔차에 의한 제곱합(SSE)은 모델로 설명되지 않는 잔차의 변동을 의미.
- 자유도는 해당 제곱합이 얼마나 많은 정보를 제공하는지를 나타내는 지표.
 - 총 제곱합의 자유도는 데이터의 총 개수에서 1을 뺀 값.
 - 회귀 모델의 자유도는 사용된 설명변수의 개수입니다. 일반적으로 단순선형회귀에서는 1임.
 - 잔차의 자유도는 총 제곱합의 자유도에서 회귀 모델의 자유도를 뺀 값.
- 평균제곱은 제곱합을 자유도로 나눈 값으로, 해당 요인이 제공하는 변동의 크기를 나타냄.
- F 통계량은 회귀 모델의 적합성을 평가하는 지표로 회귀 모델이 종속 변수의 변동을 얼마나 잘 설명하는지를 나타내며, F 통계량이 크면 회귀 모델이 종속 변수의 변동을 잘 설명한다고 해석됨.

3. 비용 최소화하기 - 경사하강법(Gradient Descent) 소개

a. 경사하강법

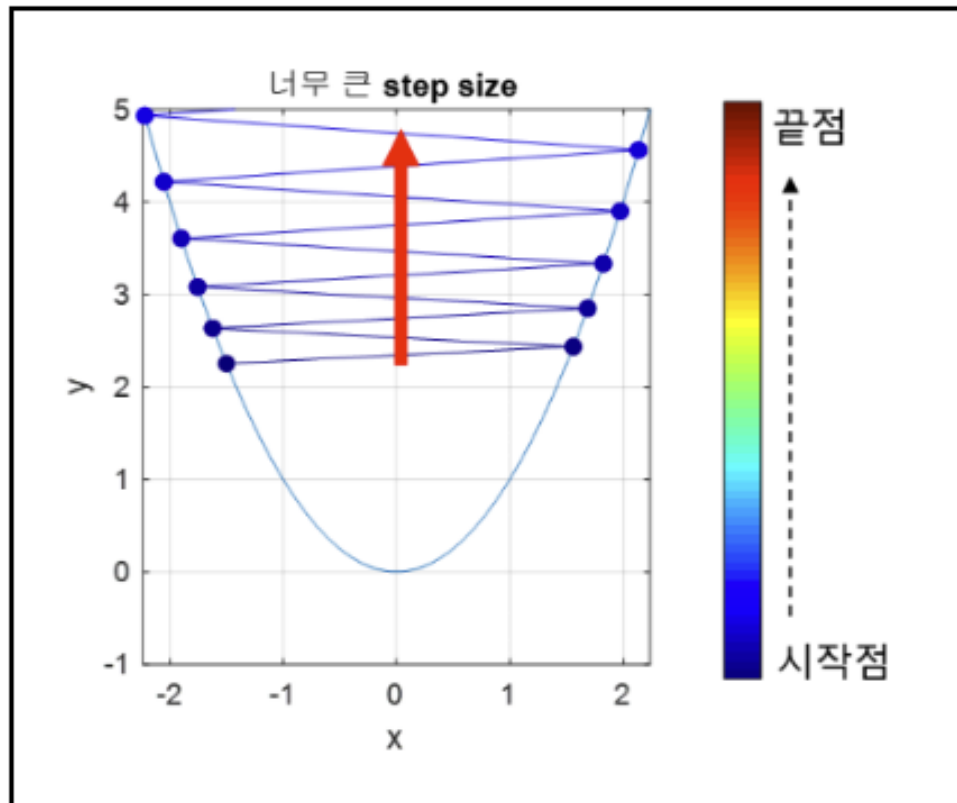
- 함수의 값이 낮아지는 방향으로 각 독립변수들의 값을 변형시키면서 함수가 최솟값을 갖도록 하는 독립변수의 값을 탐색 방법을 의미하며 일반적으로 입력된 Parameter의 검증(Validation)이 필요할 때 사용됨

- 일반적으로 경사 하강법은 함수의 최솟값을 찾아야 하는 상황에서 사용되기 때문에 인공지능의 경우 최적의 학습 패턴을 위해 자신의 파라미터(Parameter)를 검증해야 하며 검증 과정에서 손실 함수를 사용함. 검증 과정에서 손실 함수의 값이 가장 낮은 파라미터를 발견했다면 해당 파라미터가 최적의 파라미터임이 검증되는 것

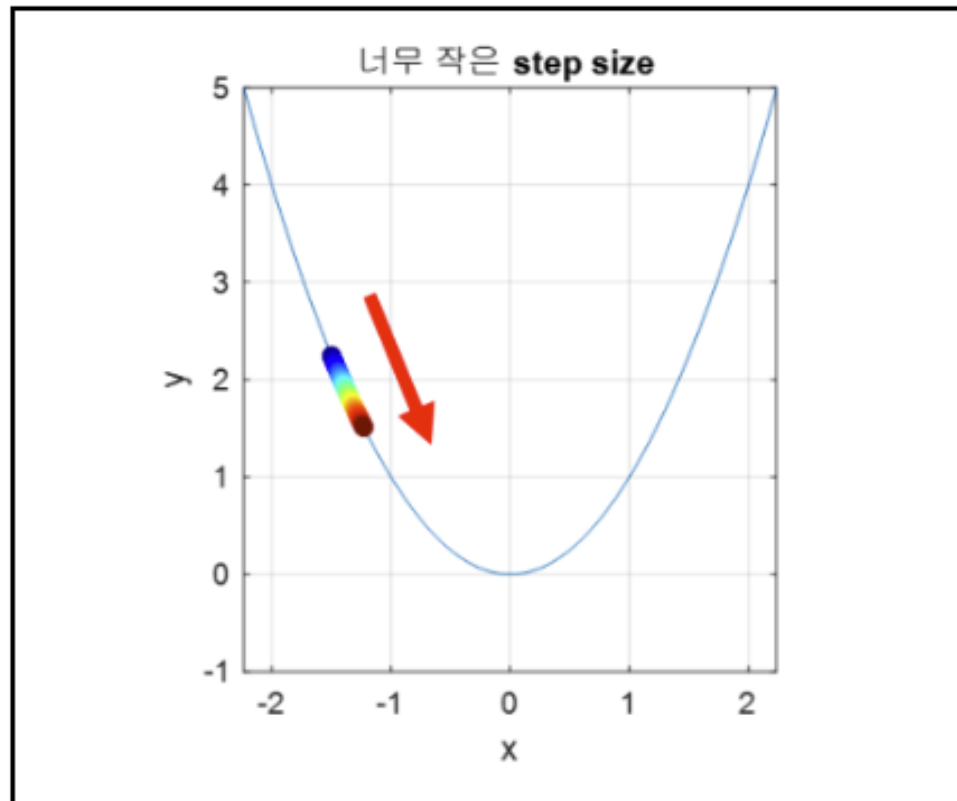


b. 경사하강법의 문제점

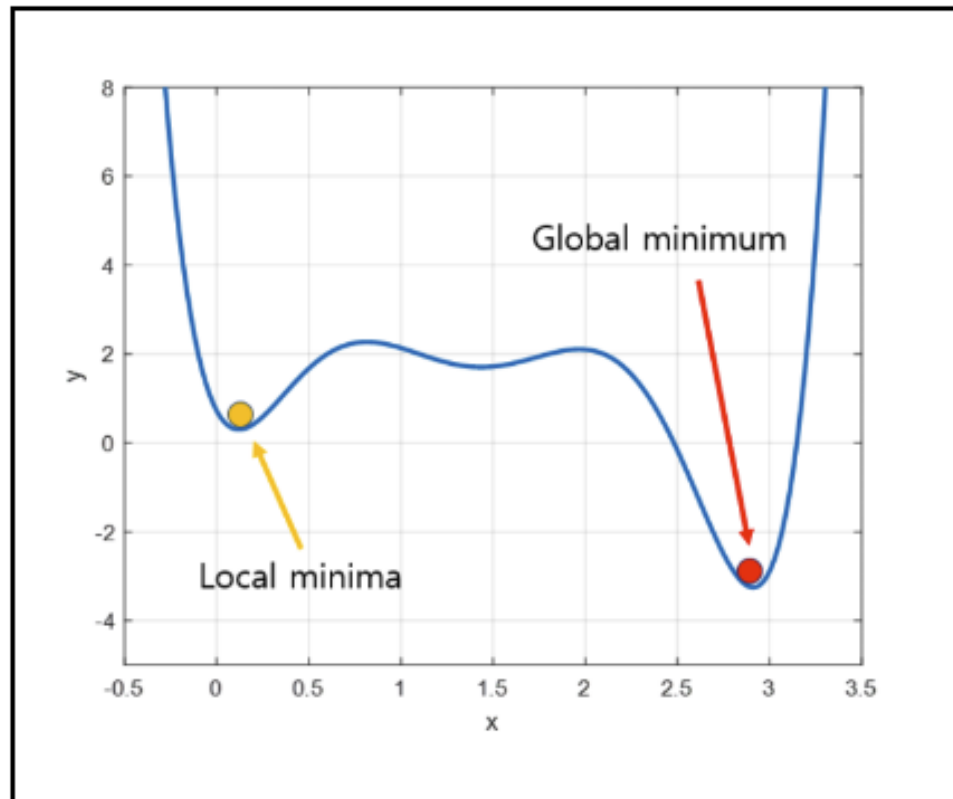
- Step size 선정 : Step size를 과도하게 설정한 경우
 - Step size를 크게 잡은 경우, 한 번에 이동하는 크기가 커지게 되므로 빠르게 수렴할 수 있다는 장점이 있음.
 - 하지만 과도하게 설정하게 되면 손실 함수의 최솟값을 계산하기 어려워질 수 있으며 함수의 값이 커지는 방향으로 최적화가 진행될 수 있으므로 주의해야함



- Step size 선정 : Step size를 너무 작게 설정한 경우
 - Step size가 너무 작게 설정된 경우 발산하지 않을 수 있다는 점이 있지만, 최적의 파라미터를 탐색할 때 소요되는 시간이 다소 오래 걸릴 수 있다는 단점이 존재함



- Local minima(지역 극솟값) 문제
 - 실제로 최적의 파라미터를 찾기 위해 Global minimum을 찾아야 하지만, 경사 하강법의 특성상 알고리즘이 시작되는 파라미터 위치가 랜덤이므로 특정한 경우에서 Local minima에 빠지게 되어 헤어 나오지 못하는 상황이 발생할 수 있음



4. 사이킷런 LinearRegression을 이용한 보스턴 주택 가격 예측

a. 회귀 평가 지표

- 아래 표 외에, 로그를 적용한 MSLE(Mean Squared Log Error)와 RMSLE(Root Mean Squared Log Error)도 사용함

평가지표	설명	수식
MAE	Mean Absolute Error이며, 실제값과 예측값의 차이를 절댓값으로 변환해 평균한 것	$MAE = \frac{1}{n} \sum_{i=1}^n Y_i - \hat{Y}_i $
MSE	Mean Squared Error이며, 실제값과 예측값의 차이를 제곱해 평균한 것	$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$
RMSE	Root Mean Squared Error이며, MSE 값이 오류의 제곱을 구하므로 실제 오류 평균보다 더 커지는 특성이 있으므로 MSE에 루트를 씌운 것	$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2}$
R^2	분산 기반으로 예측 성능을 평가. 실제 값의 분산대비 예측값의 분산 비율을 지표로 하며, 1에 가까울수록 예측 정확도가 높음	$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$

b. LinearRegression을 이용해 보스턴 주택 가격 회귀 구현

- 데이터 불러오기

```
from sklearn.datasets import load_boston

# 데이터 셋 로드
boston = load_boston()

# boston 데이터 셋 DataFrame 변환
boston_df = pd.DataFrame(boston.data,
                          columns=boston.feature_name)

# boston 데이터 셋의 target 배열은 주택가격으로 "PRICE" 컬럼 추가
boston_df["PRICE"] = boston.target
print("Boston 데이터 셋 크기 : ", boston_df.shape)
boston_df.head()
```

Boston 데이터 셋 크기 : (506, 14)

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

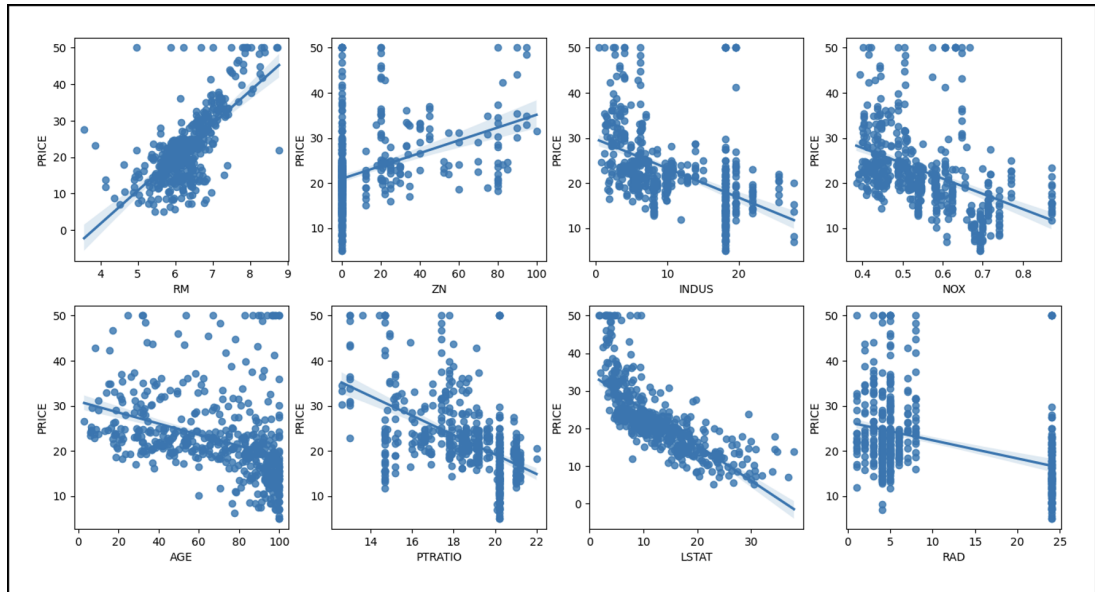
- 독립변수와 종속변수 간의 상관관계 시각화하기
 - RM(방 개수)는 PRICE(가격)과의 가장 큰 양의 상관관계를 가지고 있으므로 방의 개수가 많으면 많을수록 주택가격이 증가한다는 의미.
 - LSTAT(하위 계층의 비율)은 PRICE(가격)과의 가장 큰 음의 상관관계를 가지고 있으므로 하위 계층의 비율이 적을수록 주택가격이 증가한다는 의미.

```
# 2개의 행과 4개의 열을 가진 subplots를 이용, axs는 4*2개의 ax
fig, axs = plt.subplots(figsize=(16,8), ncols=4, nrows=2)
lm_features = ["RM", "ZN", "INDUS", "NOX", "AGE", "PTRATIO",
               "DIS", "RAD", "TAX", "B", "LSTAT", "PRICE"]
for i, feature in enumerate(lm_features):
```

```

row = int(i/4)
col = i%4
# 사본의 regplot을 이용해 산점도와 선형 회귀 직선을 함께 표현
sns.regplot(x=feature, y="PRICE", data=boston_df, a

```



- 선형회귀모델 학습,예측,평가

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_ab

# 독립변수, 종속변수 생성
y_target = boston_df["PRICE"]
X_data = boston_df.drop(["PRICE"], axis=1, inplace=False)

# 훈련, 학습 데이터셋 분리
X_train, X_test, y_train, y_test = train_test_split(X_data, y_target,
                                                    random_state=42)

# 선형회귀 학습/예측/평가 수행
lr = LinearRegression()
lr.fit(X_train, y_train)

```

```

y_pred = lr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("MSE : {0:.3f}, MAE : {1:.3f}, RMSE : {2:.3f}".format(mse, mae, rmse))
print("r2_score : {0:.3f}".format(r2))

```

```

MSE : 17.297, MAE : 3.214, RMSE : 4.159
r2_score : 0.757

```

```

# 절편과 회귀계수 출력
print("절편 값 : ", lr.intercept_)
print("회귀 계수 값 : ", np.round(lr.coef_,1))

```

```

절편 값 : 40.9955951721644
회귀 계수 값 : [ -0.1  0.1  0.   3.  -19.8  3.4  0.  -1.7  0.4 -0.  -0.9  0.
 -0.6]

```

```

coef = pd.Series(lr.coef_, index=boston_df.columns[:-1])
coef_sorted = coef.sort_values(ascending=True)

# 회귀계수 시각화
plt.figure(figsize=(10, 6))
ax = coef_sorted.plot(kind='barh')
plt.title('Linear Regression Coefficients')
plt.xlabel('Features')
plt.ylabel('Coefficient Magnitude')
plt.xlim(-20, 5)

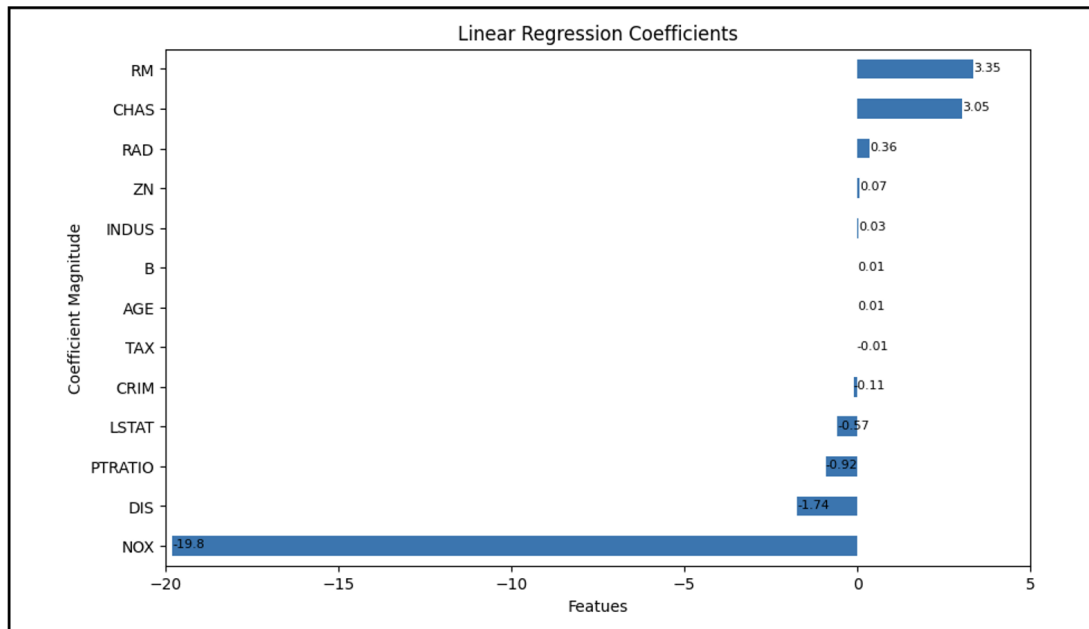
```

```

for i in ax.patches:
    plt.text(i.get_width(), i.get_y() + 0.2,
             str(round(i.get_width(), 2)), fontsize=8, color='black')

plt.show

```



- 교차 검증
 - "cv=5" : 파라미터를 활용해 5개의 폴드 세트를 만들 수 있음
 - MSE score가 낮을수록 좋은 모델
 - RMSE score는 제공하지 않으므로 MSE score에 -1을 곱한 후, 제곱근으로 계산을 함

```

from sklearn.model_selection import cross_val_score

# cross_val_score()로 5 폴드 세트로 MSE를 구한 뒤, 이를 기반으로
neg_mse_scores = cross_val_score(lr, X_data, y_target,

rmse_scores = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

```

```
# cross_val_score(scoring="neg_mean_squared_error")로 받
print(" 5 folds 의 개별 Negative MSE scores : ", np.round
print(" 5 folds 의 개별 RMSE scores : ", np.round(rmse_sc
print(" 5 folds 의 평균 RMSE : {0:.3f}".format(avg_rmse))
```

```
5 folds 의 개별 Negative MSE scores : [-12.46 -26.05 -33.07 -80.76 -33.31]
5 folds 의 개별 RMSE scores : [3.53 5.1 5.75 8.99 5.77]
5 folds 의 평균 RMSE : 5.829
```

5. 다항 회귀와 과적합/과소적합 이해

a. 다항 회귀 이해

- 다항 회귀 : 독립변수가 한 개나 1차식이 아닌 2차, 3차 방정식과 같은 다항식으로 표현되는 것을 말함.
- 다항 회귀를 비선형 회귀로 혼동하기 쉽지만, 다항 회귀는 선형 회귀라는 점.
 - 선형/비선형 회귀를 나누는 기준은 회귀 계수가 선형/비선형인지에 따른 것이
지 독립변수의 선형/비선형 여부와는 무관함.

< 2차항 회귀식 >

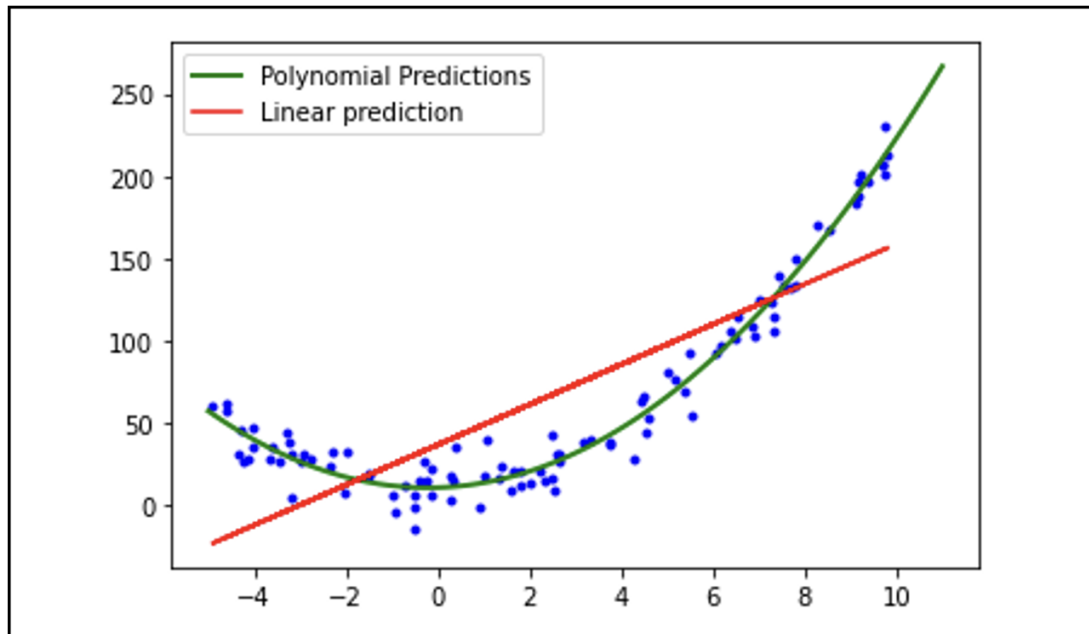
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1 x_2 + \beta_5 x_2^2$$

새로운 변수인 $z = [x_1, x_2, x_1^2, x_1 x_2, x_2^2]$ 활용한다면,

$$y = \beta_0 + \beta_1 z_1 + \beta_2 z_2 + \beta_3 z_3 + \beta_4 z_4 + \beta_5 z_5$$

위와 같이 표현할 수 있기 때문에 다항회귀는 선형회귀임

- 데이터 셋이 곡선형태를 가지게 된다면, 독립변수와 종속변수의 관계를 직선보
다는 다항 회귀의 곡선으로 표현한 것이 더 예측 성능이 높음.



- 단항값을 2차 다항값으로 변환

```
from sklearn.preprocessing import PolynomialFeatures

# 다항식으로 변환한 단항식 생성, [[0, 1], [2, 3]]의 2x2 행렬 생성
X = np.arange(4).reshape(2,2)
print("1차 단항식 계수 피쳐:\n", X)

# degree = 2인 2차 다항식으로 변환하기 위해 PolynomialFeatures
poly = PolynomialFeatures(degree=2)
poly.fit(X)
poly_trans = poly.transform(X)
print("변환된 2차 다항식 계수 피쳐:\n", poly_trans)
```

1차 단항식 계수 피쳐:

```
[[0 1]
```

```
[2 3]]
```

변환된 2차 다항식 계수 피쳐:

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

```
[1. 2. 3. 4. 6. 9.]]
```

- 단항값을 2, 3차 다항값으로 변환하는 예제

```
from sklearn.preprocessing import PolynomialFeatures

# 다항식으로 변환한 단항식 생성, [[0, 1], [2, 3]]의 2x2 행렬 생성
X = np.arange(4).reshape(2,2)
print("1차 단항식 계수 피처:\n", X)

# degree = 2인 2차 다항식으로 변환하기 위해 PolynomialFeatures
poly = PolynomialFeatures(degree=2)
poly.fit(X)
poly_trans = poly.transform(X)
print("변환된 2차 다항식 계수 피처:\n", poly_trans)
```

1차 단항식 계수 피처:

$\begin{bmatrix} 0 & 1 \end{bmatrix}$

$\begin{bmatrix} 2 & 3 \end{bmatrix}$

변환된 2차 다항식 계수 피처:

$\begin{bmatrix} 1. & 0. & 1. & 0. & 0. & 1. \end{bmatrix}$

$\begin{bmatrix} 1. & 2. & 3. & 4. & 6. & 9. \end{bmatrix}$

첫번째 입력 단항 계수 피처 $[x_1 = 0, x_2 = 1]$ 를

$[1, x_1 = 0, x_2 = 1, x_1^2 = 0, x_1x_2 = 0, x^2 = 1]$ 로 변환됨.

두번째 입력 단항 계수 피처 $[x_1 = 2, x_2 = 3]$ 를

$[1, x_1 = 2, x_2 = 3, x_1^2 = 4, x_1x_2 = 5, x^2 = 9]$ 로 변환됨.


```

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

def polynomial_func(X):
    y = 1 + 2*X[:,0] + 3*X[:,0]**2 + 4*X[:,1]**3
    return y

X = np.arange(4).reshape(2,2)
y = polynomial_func(X)

# 3차 다항식 변환
poly_trans = PolynomialFeatures(degree=3).fit_transform
print("3차 다항식 계수 feature: \n", poly_trans)

# Linear Regression에 3차 다항식 계수 feature와 3차 다항식 결과
model = LinearRegression()
model.fit(poly_trans, y)

print("3차 다항식 회귀계수:\n", np.round(model.coef_, 2))

```

```

3차 다항식 계수 feature:
[[ 1.  0.  1.  0.  0.  1.  0.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.  8. 12. 18. 27.]]
3차 다항식 회귀계수:
[[0.    0.    0.    0.01 0.01 0.02 0.02 0.03 0.04 0.06]
 [0.    0.14 0.14 0.27 0.41 0.55 0.55 0.82 1.23 1.78]]


```

b. 다항 회귀를 이용한 과소적합 및 과적합 이해

- 다항 회귀는 다항식의 차수가 높아질수록 매우 복잡한 피쳐 간의 관계까지 모델링이 가능하지만, 학습 데이터에만 너무 맞춘 학습이 이뤄져서 과적합 문제가 발생함.
- 예제) 다항식의 코사인 그래프

Underfitting vs. Overfitting

This example demonstrates the problems of underfitting and overfitting and how we can use linear regression with polynomial features to

 https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html



```

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

def true_fun(X):
    return np.cos(1.5 * np.pi * X)

np.random.seed(0)

n_samples = 30
degrees = [1, 4, 15]

X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1

# 다항회귀의 차수를 1, 4, 15로 각각 변화시키면서 비교합니다.
plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    # 개별 degree별로 Polynomial 변환합니다.
    polynomial_features = PolynomialFeatures(degree=deg

    linear_regression = LinearRegression()
    pipeline = Pipeline(
        [
            ("polynomial_features", polynomial_features),
            ("linear_regression", linear_regression),
        ]
    )
    pipeline.fit(X[:, np.newaxis], y)

    # 교차 검증으로 다항 회귀를 평가합니다.
    scores = cross_val_score(

```

```

        pipeline, X[:, np.newaxis], y,
        scoring="neg_mean_squared_error", cv=10
    )

    # Pipeline을 구성하는 세부 객체를 접근하는 named_steps["객체"]
    coef = pipeline.named_steps['linear_regression'].coef_
    print("\nDegree {0} 회귀계수는 {1} 입니다.".format(degree, coef))

    print("Degree {0} MSE는 {1} 입니다.".format(degrees[i], -scores[i]))

    # 0~1 까지 테스트 데이터셋을 100개로 나눠 예측을 수행합니다
    # 테스트 데이터 세트에 회귀 예측을 수행하고 예측 곡선과 실제 곡선을 비교합니다
    X_test = np.linspace(0, 1, 100)
    # 예측값 곡선
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Predicted")
    # 실제값 곡선
    plt.plot(X_test, true_fun(X_test), label="True function")
    plt.scatter(X, y, edgecolor="b", s=20, label="Sample")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title(
        "Degree {} \nMSE = {:.2e} (+/- {:.2e})".format(
            degrees[i], -scores.mean(), scores.std()
        )
    )
plt.show()

```

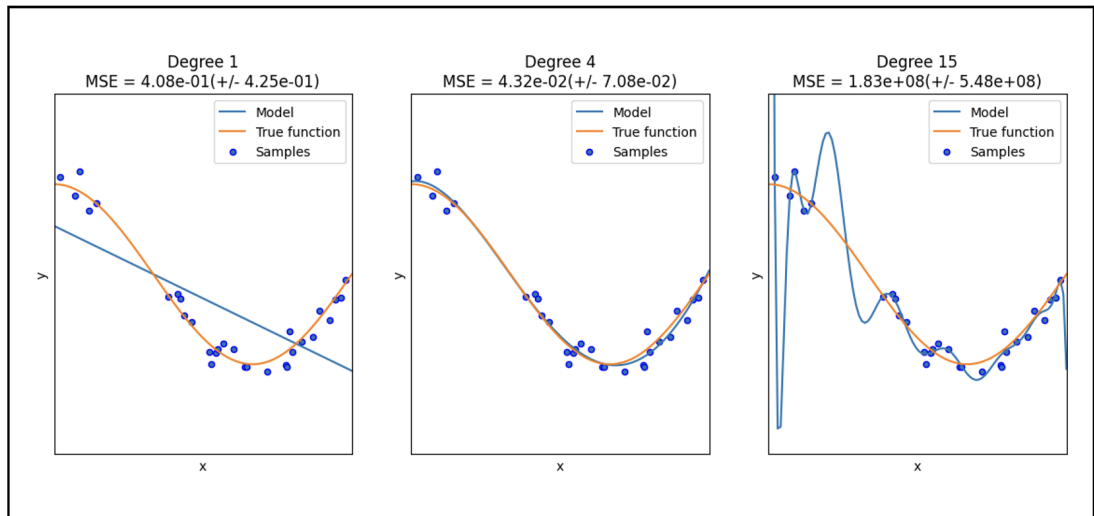
```

Degree 1 회귀계수는 [-1.61] 입니다.
Degree 1 MSE는 0.40772896250986845 입니다.

Degree 4 회귀계수는 [ 0.47 -17.79 23.59 -7.26] 입니다.
Degree 4 MSE는 0.043208749872317834 입니다.

Degree 15 회귀계수는 [-2.98293000e+03  1.03899500e+05 -1.87416382e+06  2.03716564e+07
-1.44873571e+08  7.09316967e+08 -2.47066414e+09  6.24562771e+09
-1.15676855e+10  1.56895436e+10 -1.54006546e+10  1.06457645e+10
-4.91379382e+09  1.35920182e+09 -1.70381489e+08] 입니다.
Degree 15 MSE는 182663732.56281167 입니다.

```

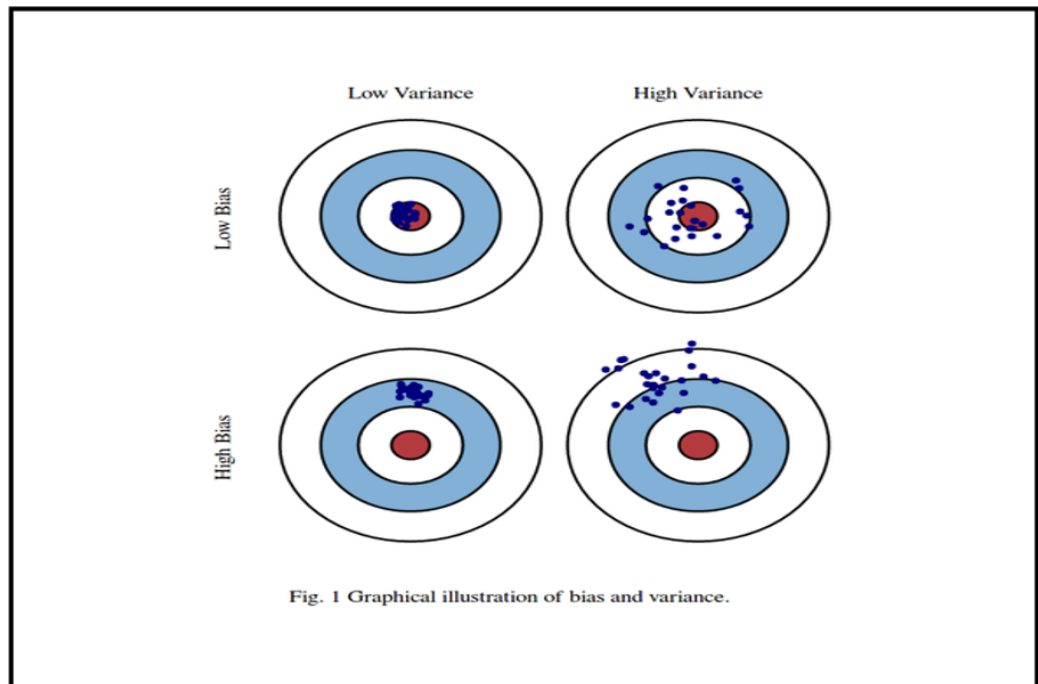


- Degree 1 예측곡선 : 직선으로서 단순선형회귀와 같고, 해당 예측 곡선은 학습 데이터의 패턴을 제대로 반영하지 못하고 있는 과소적합 모델이 되었음. MSE 값은 0.41
- Degree 4 예측곡선 : 실제 데이터 세트와 유사한 모습이며, 변동하는 잡음까지 예측하지 못했지만, 학습데이터 세트를 비교적 잘 반영해 코사인 곡선 기반으로 테스트 데이터를 잘 예측한 곡선을 가진 모델이 되었음. MSE 값은 0.04로 가장 뛰어난 예측 성능을 나타냄.
- Degree 15 예측곡선 : 데이터 세트의 변동 잡음값까지 지나치게 반영한 결과, 예측 곡선이 학습 데이터 세트만 정확히 예측하고, 테스트 값의 실제 곡선과는 완전히 다른 형태의 예측곡선이 만들어졌음.

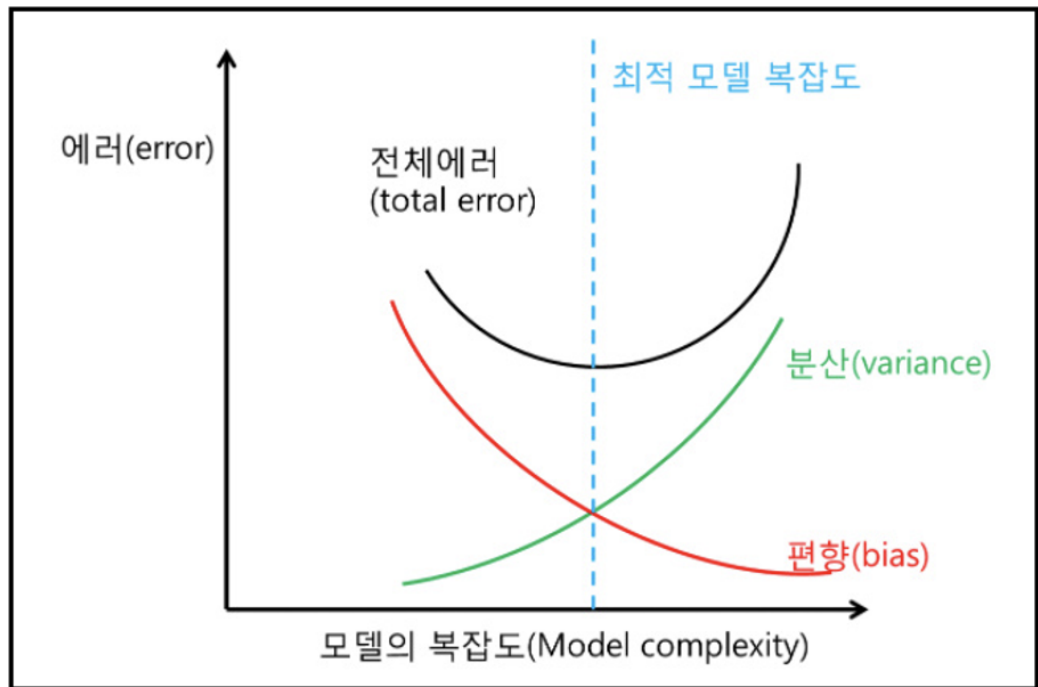
c. 편향-분산 트레이드오프(Bias-Variance Trade off)

- 편향(Bias)
 - 예측값과 실제값의 차에 대한 평균으로 예측값이 실제값과 얼마나 떨어져 있는지를 의미
 - 편향이 크면 과소적합을 야기시키며, 지나치게 단순한 모델로 인한 error가 발생
 - 모델에 편향이 크다는 것은 그 모델이 뭔가 중요한 요소를 놓치고 있다는 뜻
- 분산(Variance)
 - 다양한 데이터셋에 대하여 예측값이 얼마나 변화할 수 있는지에 대한 양을 의미
 - 분산이 크면 과대 적합을 야기시키며, 훈련 데이터에 지나치게 적합시키려는 모델로 인한 error 발생

- 분산이 큰 모델은 훈련 데이터에 지나치게 적합을 시켜 일반화가 되지 않은 모델을 의미



- 편향-분산 트레이드오프
 - 편향과 분산은 트레이드오프 관계를 가져 한쪽이 커지면 다른 한쪽은 작아지는 관계임.
 - 편향이 높으면 분산이 낮아지고(과소적합), 반대로 분산이 높으면 편향이 낮아짐(과적합)
 - 편향이 분산이 서로 트레이드오프를 이루면서 오류 Cost값이 최대로 낮아지는 모델을 구축하는 것이 가장 효율적인 머신러닝 예측모델을 만드는 방법



6. 규제 선형 모델 - 릿지, 라쏘, 엘라스틱넷

a. 규제 선형 모델의 개요

- 비용함수는 학습 데이터의 잔차 오류 값을 최소화 하는 RSS(잔차 제곱합) 최소화 방법과 과적합을 방지하기 위해 회귀 계수 값이 커지지 않도록 하는 방법이 서로 균형을 이뤄야 함.

$$\text{비용 함수 목표} = \text{Min}(\text{RSS}(W) + \alpha * ||W||_2^2)$$

↑

학습데이터 잔차
오류 최소화

↑

회귀계수
크기제어

- $\alpha = 0$ 인 경우는 W 가 커도 회귀계수 크기제어 부분이 0이 되어 비용함수는 $\text{Min}(\text{RSS}(W))$
- $\alpha = \text{무한대}$ 인 경우는 회귀계수 크기제어 부분이 무한이 되어 비용함수는 W 를 0에 가깝게 최소화 해야함
- α 를 0에서부터 지속적으로 값을 증가시키면 회귀 계수 값의 크기를 감소시킬 수 있으므로 비용함수에 α 값으로 패널티를 부여해 회귀 계수 값의 크기를 감

소시켜 과적합을 개선 하는 방식을 규제(Regularization)라고 부름.

b. 릿지 회귀(Ridge Regression)

- 릿지 회귀란 L2 규제를 적용한 회귀 방식
- L2 규제는 비용 함수에서의 W를 제공하여 패널티를 부여하는 방식

< 릿지 회귀 >

$$J(\theta) = MSE(\theta) + \alpha * \frac{1}{2} * \sum_{i=1}^n \theta_i^2$$

◦ 릿지 모델 수행

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score

# alpha = 10으로 설정해 릿지 회귀 수행
ridge = Ridge(alpha=10)
neg_mse_scores = cross_val_score(ridge, X_data, y_target)

rmse_scores = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print(" 5 folds 의 개별 Negative MSE scores : ", np.round(neg_mse_scores, 2))
print(" 5 folds 의 개별 RMSE scores : ", np.round(rmse_scores, 2))
print(" 5 folds 의 평균 RMSE : {0:.3f}".format(avg_rmse))
```

```
5 folds 의 개별 Negative MSE scores : [-11.422 -24.294 -28.144 -74.599 -28.517]
5 folds 의 개별 RMSE scores : [3.38  4.929 5.305 8.637 5.34 ]
5 folds 의 평균 RMSE : 5.518
```

```
# 릿지에 사용될 alpha 파라미터의 값을 정의
alphas = [0, 0.1, 1, 10, 100]

# alphas list 값을 반복하면서 alpha에 따른 평균 rmse를 구함.
for alpha in alphas:
    ridge = Ridge(alpha=alpha)

    # cross_val_score를 이용해 5 폴드의 평균 RMSE를 계산
    neg_mse_scores = cross_val_score(ridge, X_data, y_t

    avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
    print('alpha {0} 일 때, 5 folds의 평균 RMSE : {1:.3f}')
```

```
alpha 0 일 때, 5 folds의 평균 RMSE : 5.829
alpha 0.1 일 때, 5 folds의 평균 RMSE : 5.788
alpha 1 일 때, 5 folds의 평균 RMSE : 5.653
alpha 10 일 때, 5 folds의 평균 RMSE : 5.518
alpha 100 일 때, 5 folds의 평균 RMSE : 5.330
```

- alpha 값에 따른 회귀 계수 값 시각화

```
# alpha에 따른 회귀 계수 값을 시각화하기 위해 5개의 열로 된 맷플로
fig, axs = plt.subplots(figsize=(18,6), nrows=1, ncols=5)

# 각 alpha에 따른 회귀계수 값을 데이터로 저장하기 위한 DataFrame
coef_df = pd.DataFrame()

# alphas 리스트 값을 차례로 입력해 회귀계수 값 시각화 및 데이터 저장
for pos, alpha in enumerate(alphas):
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_data, y_target)
    # alpha에 따른 피쳐별로 회귀 계수를 Series로 변환하고 이를 DataFrame에 저장
    coef = pd.Series(data=ridge.coef_, index=X_data.columns)
    colname = "alpha:" + str(alpha)
    coef_df[colname] = coef
    # 막대그래프로 각 alpha 값에서의 회귀 계수를 시각화. 회귀 계수
```



```

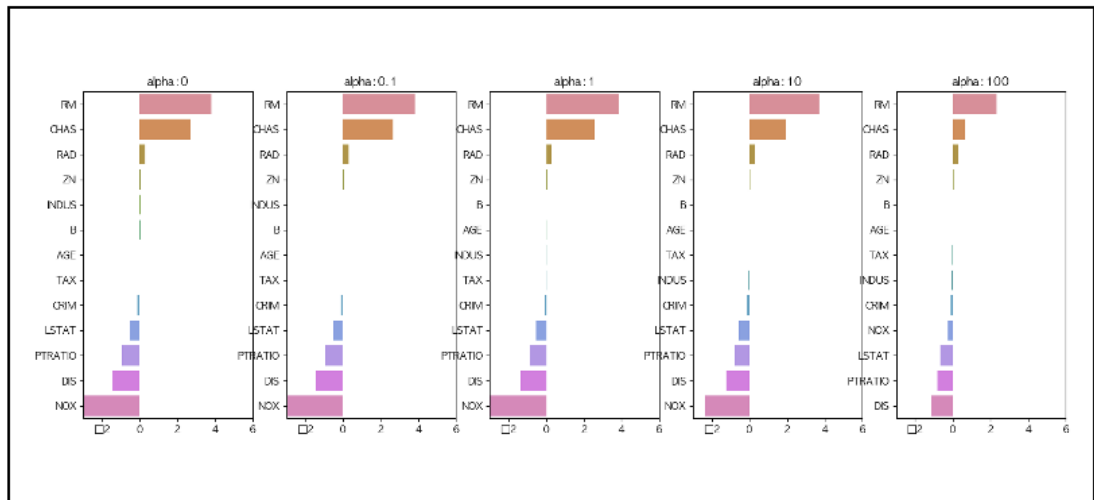
coef = coef.sort_values(ascending=False)
axs[pos].set_title(colname)
axs[pos].set_xlim(-3,6)
sns.barplot(x=coef.values, y=coef.index, ax=axs[pos]

```

```

# for문 바깥에서 맷플롯립의 show 호출 및 alpha에 따른 피쳐별 회귀
plt.show()

```



```

ridge_alphas = [0,0.1,1,10,100]
sort_column = "alpha:"+str(ridge_alphas[0])
coef_df.sort_values(by=sort_column, ascending=False)

```

	alpha:0	alpha:0.1	alpha:1	alpha:10	alpha:100
RM	3.809865	3.818233	3.854000	3.702272	2.334536
CHAS	2.686734	2.670019	2.552393	1.952021	0.638335
RAD	0.306049	0.303515	0.290142	0.279596	0.315358
ZN	0.046420	0.046572	0.047443	0.049579	0.054496
INDUS	0.020559	0.015999	-0.008805	-0.042962	-0.052826
B	0.009312	0.009368	0.009673	0.010037	0.009393
AGE	0.000692	-0.000269	-0.005415	-0.010707	0.001212
TAX	-0.012335	-0.012421	-0.012912	-0.013993	-0.015856
CRIM	-0.108011	-0.107474	-0.104595	-0.101435	-0.102202
LSTAT	-0.524758	-0.525966	-0.533343	-0.559366	-0.660764
PTRATIO	-0.952747	-0.940759	-0.876074	-0.797945	-0.829218
DIS	-1.475567	-1.459626	-1.372654	-1.248808	-1.153390
NOX	-17.766611	-16.684645	-10.777015	-2.371619	-0.262847

c. 라쏘 회귀(Lasso Regression)

- 라쏘 회귀란 L1 규제를 적용한 회귀 방식
- L1 규제는 비용 함수에서 W의 절댓값에 대해 패널티를 부여하는 방식으로 불필요한 회귀 계수를 급격하게 감소시켜 0으로 만들고 제거함

< 라쏘 회귀 >

$$J(\theta) = MSE(\theta) + \alpha * \frac{1}{2} * \sum_{i=1}^n |\theta_i|$$

◦ 라쏘모델 수행

- alpha가 0.7일때, 가장 좋은 평균 RMSE를 보여줌
- alpha의 크기를 증가함에 따라 일부 피처의 회귀 계수가 아예 0으로 바뀌고 있음

```
lasso_alphas = [0.07, 0.1, 0.5, 1, 3]
coef_lasso_df = get_linear_reg_eval("Lasso", params=las
```

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

```
# 반환된 coef_lasso_df를 첫번째 컬럼순으로 내림차순 정렬해 회귀계
sort_column = "alpha:"+str(lasso_alphas[0])
coef_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

c. 엘라스틱넷 회귀(ElasticNet)

- 엘라스틱넷 회귀는 L2 규제와 L1 규제를 결합한 회귀이며, 라쏘 회귀가 서로 상관관계가 높은 피쳐들의 경우에 이들 중에서 중요 피쳐만을 선택하고 다른 피쳐들은 모두 회귀 계수를 0으로 만드는 성향이 강함
- 특히 이러한 성향으로 인해 alpha 값에 따라 회귀 계수의 값이 급격히 변동할 수도 있는데, 엘라스틱넷 회귀는 이를 L2 규제를 라쏘 회귀에 추가한 것

- 반대로 엘라스틱넷 회귀의 단점은 L1과 L2 규제가 결합된 규제로 인해 수행시간이 상대적으로 오래 걸린다는 것

< 엘라스틱넷 회귀 >

$$J(\theta) = MSE(\theta) + r * \alpha * \sum_{i=1}^n \theta_i^2 + \alpha * \frac{1-r}{2} * \sum_{i=1}^n |\theta_i|$$

```
# 엘라스틱넷에 사용될 alpha 파라미터 값들을 정의하고 get_linear_reg_eval
# l1_ratio는 0.7로 고정
elastic_alphas = [0.07, 0.1, 0.5, 1, 3]
coef_elastic_df = get_linear_reg_eval("ElasticNet", param_grid={"alpha": elastic_alphas},
                                     X_data_n=X_data, y_data_n=y_data)
```

```
##### ElasticNet #####
alpha 0.07 일 때 5폴드 세트의 평균 RMSE: 5.542
alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 5.526
alpha 0.5 일 때 5폴드 세트의 평균 RMSE: 5.467
alpha 1 일 때 5폴드 세트의 평균 RMSE: 5.597
alpha 3 일 때 5폴드 세트의 평균 RMSE: 6.068
```

```
# 반환된 coef_elastic_df를 첫번째 컬럼순으로 내림차순 정렬해 회귀
sort_column = "alpha:"+str(elastic_alphas[0])
coef_elastic_df.sort_values(by=sort_column, ascending=False)
```

	alpha:0.07	alpha:0.1	alpha:0.5	alpha:1	alpha:3
RM	3.574162	3.414154	1.918419	0.938789	0.000000
CHAS	1.330724	0.979706	0.000000	0.000000	0.000000
RAD	0.278880	0.283443	0.300761	0.289299	0.146846
ZN	0.050107	0.050617	0.052878	0.052136	0.038268
B	0.010122	0.010067	0.009114	0.008320	0.007020
AGE	-0.010116	-0.008276	0.007760	0.020348	0.043446
TAX	-0.014522	-0.014814	-0.016046	-0.016218	-0.011417
INDUS	-0.044855	-0.042719	-0.023252	-0.000000	-0.000000
CRIM	-0.099468	-0.099213	-0.089070	-0.073577	-0.019058
NOX	-0.175072	-0.000000	-0.000000	-0.000000	-0.000000
LSTAT	-0.574822	-0.587702	-0.693861	-0.760457	-0.800368
PTRATIO	-0.779498	-0.784725	-0.790969	-0.738672	-0.423065
DIS	-1.189438	-1.173647	-0.975902	-0.725174	-0.031208

- alpha =0.5일때 RMSE가 5.467로 가장 좋은 예측 성능을 보이고 있음
- alpha값에 따른 피쳐들의 회귀 계수들 값이 라쏘보다는 상대적으로 0이 되는 값이 적음 알수 있음

d. 선형 회귀 모델을 위한 데이터 변환

- 데이터 변환
 - 선형회귀모델과 같은 선형 모델은 일반적으로 피쳐와 타깃값 간에 선형의 관계가 있다고 가정하고, 이러한 최적의 선형함수를 찾아내 결과값을 예측함.
 - 또한 선형회귀모델은 피쳐값과 타깃값의 분포가 정규분포(즉, 평균을 중심으로 종 모양으로 데이터 값이 분포된 형태) 형태를 매우 선호함.
 - 특히 타깃값의 경우 정규 분포형태가 아니라 특정값의 분포가 치우친 왜곡(Skew)된 형태의 분포도일 경우 예측 성능에 부정적인 영향을 미칠 가능성이 높음.
 - 피쳐값 역시 결정값보다는 덜하지만 왜곡된 분포도로 인해 예측 성능에 부정적인 영향을 미칠 수 있음.
- 데이터 변환 종류

- StandardScaler 클래스 : 평균이 0 분산이 1인 표준 정규분포를 가진 데이터 세트로 변환함. 이상치에 매우 민감하며, 회귀보다 분류에 유용함
- MinMaxScaler 클래스 : 최솟값이 0이고, 최댓값이 1인 값으로 정규화 수행함. 이상치에 매우 민감하며, 분류보다 회귀에 유용함
- 로그변환 : log 함수를 적용하여 보다 정규분포에 가까운 형태로 값이 분포되도록 변환함. 주로 많이 사용되는 변환 방법임
- 데이터 변환 적용예제
 - 최댓값/최솟값 정규화로 피쳐 데이터세트를 변경해도 성능상의 개선은 없었음
 - 표준정규분포로 일차 변환후 2차 다항식 변환을 했을때 $\alpha=100$ 에서 4.634로 성능이 개선됐으며, 최댓값/최솟값 정규화로 일차 변환 후 2차 다항식 변환을 했을 때 $\alpha=1$ 에서 4.323으로 성능이 개선되었음.
 - 반면에 로그 변환을 보면 α 가 0.1, 1, 10인 경우에 모두 좋은 성능 향상이 있음을 알 수 있음.

```
# Ridge의 alpha값을 다르게 적용하고 다양한 데이터 변환 방법에 따른 RM
alphas = [0.1, 1, 10, 100]

# 5개 방식으로 변환. 먼저 원본 그대로, 표준정규 분포, 표준정규 분포+다항식
# 최대/최소 정규화, 최대/최소 정규화 + 다항식 특성, 로그변환
scale_methods = [(None, None), ("Standard", None), ("Standard", 2),
                  ("MinMax", None), ("MinMax", 2), ("Log", None)]
for scale_method in scale_methods:
    X_data_scaled = get_scaled_data(method=scale_method[0],
                                     input_data=X_data)
    print("\n## 변환 유형:{0}, Polynomial Degree:{1}".format(scale_method[1], 2))
    get_linear_reg_eval("Ridge", params=alphas, X_data_n=X_data_scaled,
                        y_target_n=y_target, verbose=False,
```

<pre>## 변환 유형:None, Polynomial Degree:None alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 5.788 alpha 1 일 때 5폴드 세트의 평균 RMSE: 5.653 alpha 10 일 때 5폴드 세트의 평균 RMSE: 5.518 alpha 100 일 때 5폴드 세트의 평균 RMSE: 5.330</pre>	<pre>## 변환 유형:MinMax, Polynomial Degree:None alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 5.764 alpha 1 일 때 5폴드 세트의 평균 RMSE: 5.465 alpha 10 일 때 5폴드 세트의 평균 RMSE: 5.754 alpha 100 일 때 5폴드 세트의 평균 RMSE: 7.635</pre>
<pre>## 변환 유형:Standard, Polynomial Degree:None alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 5.826 alpha 1 일 때 5폴드 세트의 평균 RMSE: 5.803 alpha 10 일 때 5폴드 세트의 평균 RMSE: 5.637 alpha 100 일 때 5폴드 세트의 평균 RMSE: 5.421</pre>	<pre>## 변환 유형:MinMax, Polynomial Degree:2 alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 5.298 alpha 1 일 때 5폴드 세트의 평균 RMSE: 4.323 alpha 10 일 때 5폴드 세트의 평균 RMSE: 5.185 alpha 100 일 때 5폴드 세트의 평균 RMSE: 6.538</pre>
<pre>## 변환 유형:Standard, Polynomial Degree:2 alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 8.827 alpha 1 일 때 5폴드 세트의 평균 RMSE: 6.871 alpha 10 일 때 5폴드 세트의 평균 RMSE: 5.485 alpha 100 일 때 5폴드 세트의 평균 RMSE: 4.634</pre>	<pre>## 변환 유형:Log, Polynomial Degree:None alpha 0.1 일 때 5폴드 세트의 평균 RMSE: 4.770 alpha 1 일 때 5폴드 세트의 평균 RMSE: 4.676 alpha 10 일 때 5폴드 세트의 평균 RMSE: 4.836 alpha 100 일 때 5폴드 세트의 평균 RMSE: 6.241</pre>