

Machine Learning

2025. 5. 8

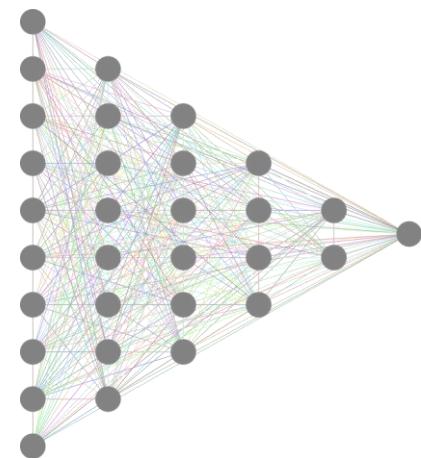
박준오

I. Machine learning

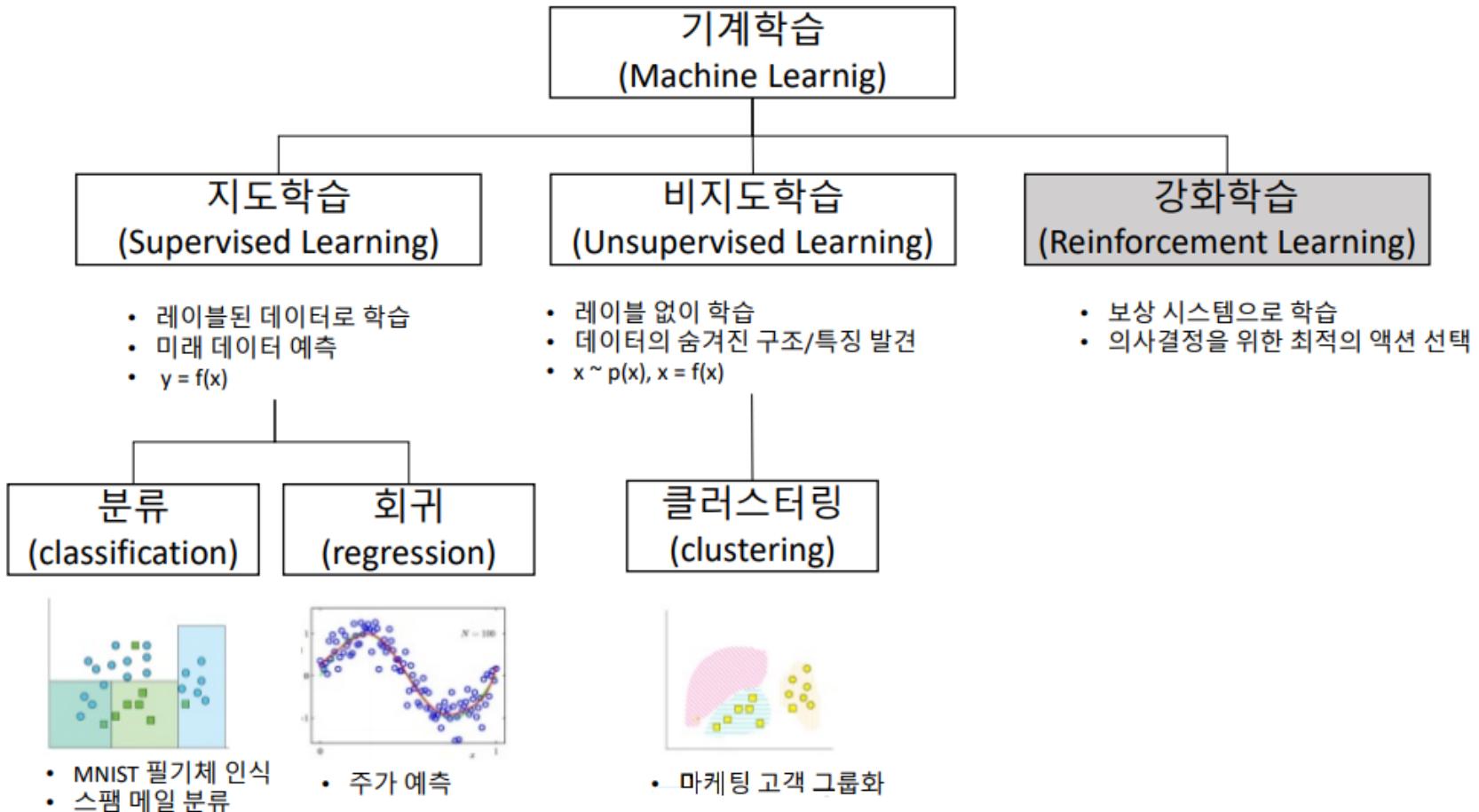
- 1. Machine Learning Intro**
- 2. Regression**
- 3. KNN/SVM**
- 4. DT/Ensemble/XGBoost**
- 5. Unsupervised Learning**

1. Machine Learning

MACHINE
LEARNING



1) Machine Learning 개요



2) ML 주요 이론

- ① 지도학습(supervised learning)
- ② 비지도학습(unsupervised learning)
- ③ 강화학습(reinforcement learning)

2) ML 주요 이론(Supervised Learning)

① 지도학습(Supervised learning)

학습데이터가 주어진 상태에서 컴퓨터를 training.

(예시) 영상 처리(개, 고양이 이미지를 training시켜, 구분할 수 있다.)

반드시 정답 데이터(Label)가 있어야 한다.

정답 데이터가 없는 경우에는 ...

=> 비지도 학습을 통해 지도 학습의

단점을 보완



2) ML 주요 이론(Supervised Learning)

① 지도학습의 종류 : 분류, 회귀

- **분류 (Classification)** : Logistic Regression/Decision Tree

주어진 데이터를 정해진 카테고리에 따라 정리하는 처리과정.
이미지 분류도 Classification 처리의 하나.

예시) 스팸메일 / 일반 메일 분류(Binary Classification)

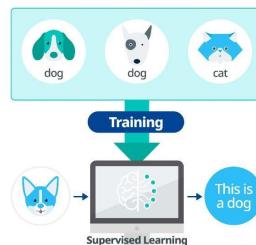
동물의 종류 분류

(강아지, 고양이, 원숭이, 돼지 등: Multi-label Classification)

- **회귀 (Regression)**

데이터들의 특징(Feature)을 기준으로 연속된 값을 예측,
특정한 패턴이나 트렌드, 경향을 예측할 때 사용

예시) 강남의 주택 평수에 따른 주택 매매가 예측



2) ML 주요 이론(Supervised Learning)

- 선형 회귀 분석(Linear Regression)

종속 변수와 한 개 이상의 독립 변수 간의 선형 관계를 모델링하는 통계적 기법.
단순 회귀(vs 다항 회귀(polynomial) vs 다중 회귀(multivariant))

2) ML 주요 이론(Supervised Learning)

- 선형 회귀 분석(Linear Regression)

종속 변수와 한 개 이상의 독립 변수 간의 선형 관계를 모델링하는 통계적 기법.
단순 회귀(vs 다항 회귀(polynomial) vs 다중 회귀(multivariant))

Simple
Linear
Regression

$$y = b_0 + b_1 x_1$$

다중 회귀

Multiple
Linear
Regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$$

다항 회귀

Polynomial
Linear
Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \dots + b_n x_1^n$$

2) ML 주요 이론(Supervised Learning)

The screenshot shows the scikit-learn API Reference page for the `LinearRegression` class. The left sidebar lists various regression models, with `LinearRegression` currently selected. The main content area displays the class definition, a brief description, parameters, and detailed explanations for each parameter.

API Reference > sklearn.linear_model > LinearRegression

LinearRegression

```
class sklearn.linear_model.LinearRegression(*, fit_intercept=True, copy_X=True, n_jobs=None, positive=False)
```

[source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients $w = (w_1, \dots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : bool, default=True
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

copy_X : bool, default=True
If True, X will be copied; else, it may be overwritten.

n_jobs : int, default=None
The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly `X` is sparse or if `positive` is set to `True`. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See [Glossary](#) for more details.

positive : bool, default=False
When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.

2) ML 주요 이론(Supervised Learning)

1. fit_intercept : bool, default=True.

True로 설정하면, 선형 모델은 데이터의 평균을 기준으로 절편을 계산하여 사용.
False로 설정하면, 절편을 사용하지 않으며, 이 경우 데이터는 이미 중심화되었다고
가정합니다(즉, 모든 특성의 평균이 0).

2. copy_X : bool, default=True

입력
데이터 x를 복사할지 여부를 결정합니다. True이면, 원본 데이터는 변경되지 않고
복사본이 사용됩니다. False로 설정하면, x가 함수에 의해 직접 수정될 수 있습니다.

3. n_jobs : int, default=None

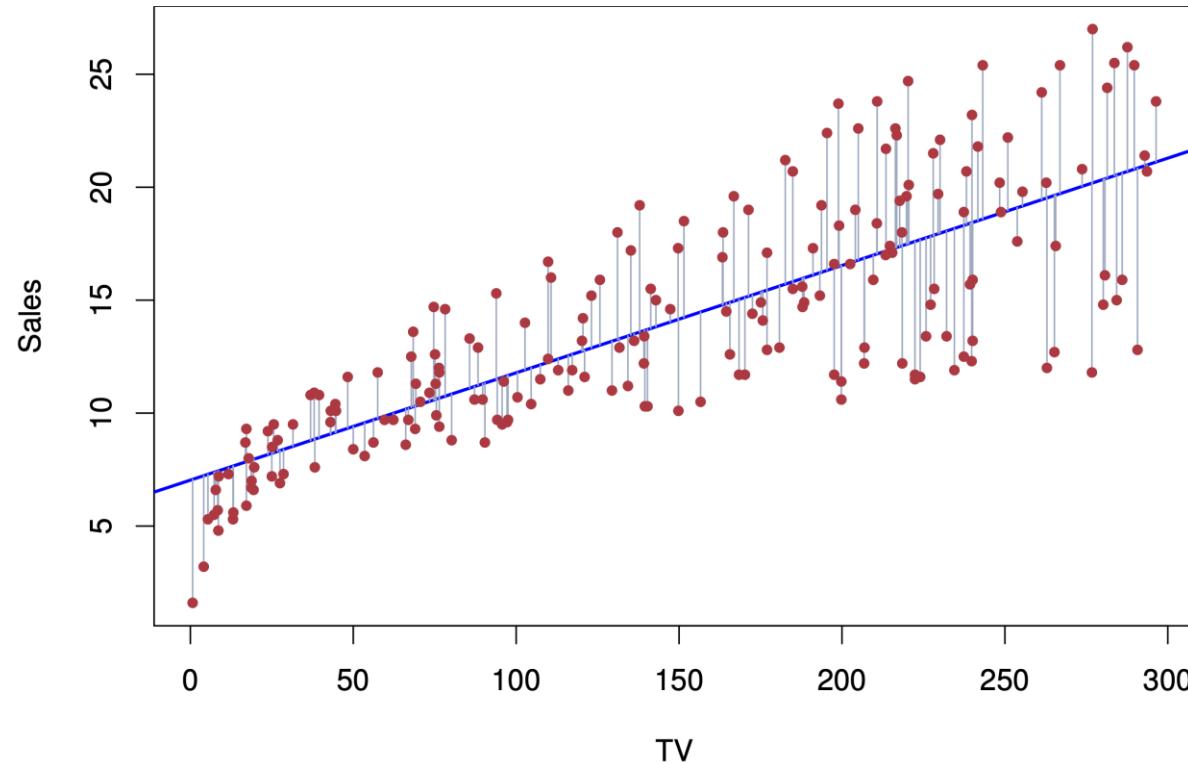
계산에 사용할 프로세서의 수, None은 1개

4. positive : bool, default=False

True로 설정하면, 모델의 계수(coefficient)가 모두 양수가 되도록 강제합니다.
이는 특정 애플리케이션에서 모든 예측 변수가 결과에 긍정적인 영향을 미쳐야
하는 경우에 유용할 수 있습니다. (모든 특성이 긍정적인 영향을 끼치는 것으로
제한하고 싶을 때 사용)

2) ML 주요 이론(Supervised Learning)

- 선형 회귀 분석(Linear Regression)

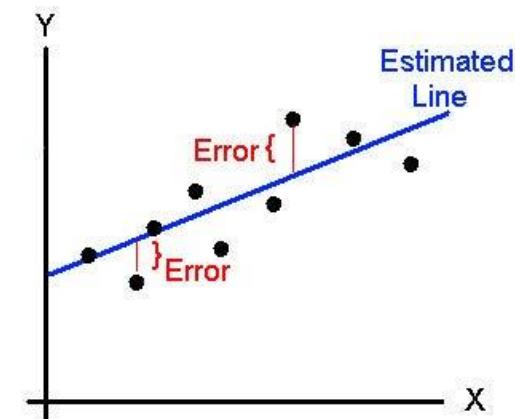


2) ML 주요 이론(Supervised Learning)

- 선형 회귀 분석(Linear Regression)

$$\hat{Y}_i = b_0 + b_1 X_i$$

Estimated (or predicted) Y value for observation i
Estimate of the regression intercept
Estimate of the regression slope
Value of X for observation i



2) ML 주요 이론(Supervised Learning)

-선형 회귀 분석(Linear Regression)

주어진 입력 변수(X)와 출력 변수(Y)의 관계를 가장 잘 설명하는
기울기와 절편을 추정하는 것이 목표.

머신러닝 : 데이터를 입력하여 평균제곱오차(MSE)를 최소화 하는 w 를 찾아가는 과정
이를 통해 새로운 입력 변수에 대한 출력 변수의 예측을 수행할 수 있다.

주요 요소 : 절편(intercept/Bias)/기울기(Slope/Coefficient)/오차항(error)

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\text{Mean } Y_i - \hat{Y}_i)^2$$

2) ML 주요 이론(Supervised Learning)

-선형 회귀 분석(Linear Regression)

*제곱을 해서 오차를 찾아가는 이유

오차를 더 뚜렷하게 확인하기 위해 절대값보다 제곱을 더 선호한다.!!
음수와 양수의 상쇄효과를 없애기 위해서 사용
(절대값도 가능하지만 오차의 크기를 강조하는 효과는 없다)

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\text{Mean Error})^{\text{Squared}}$$
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

2) ML 주요 이론(Supervised Learning)

-선형 회귀 분석(Linear Regression)

머신러닝에서의 선형회귀 ?

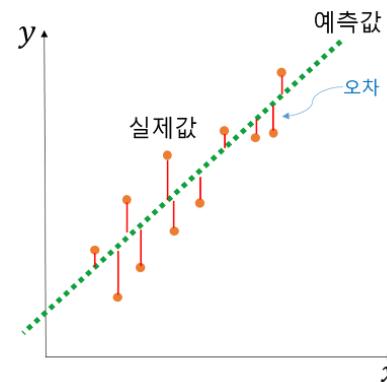
$X = [2, 3, 4]$ $Y = [6, 8, 10]$ 일때 $x = 8$ 이면 $Y = ?$ $Y = 2x + 2$ 이므로 $Y = 18$

데이터가 많아진다면....

- 1) 가설의 설정 : $H(w, b) = wx + b$ (위 자료를 만족시키는 임의의 w 와 b)
- 2) 가설 초기화 : $w=1, b=0 \Rightarrow X = [2, 3, 4] Y = [2, 3, 4] \Rightarrow \text{error} = 4, 5, 6$
error값이 가장 작아지는 손실함수 ?

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Mean Error Squared



2) ML 주요 이론(Supervised Learning)

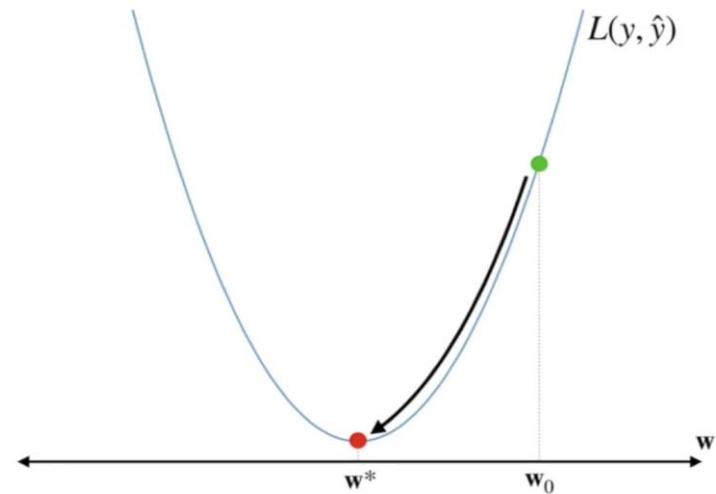
-선형 회귀 분석(Linear Regression)

1) 가설의 설정 : $H(w, b) = wx + b$ (위 자료를 만족시키는 임의의 w 와 b)

2) 가설 초기화 : $w=1, b=0 \Rightarrow X = [2, 3, 4]$ 이면 $Y = [2, 3, 4]$
 $\Rightarrow \text{error} = 4, 5, 6$

error값이 가장 작아지는 손실함수 ?

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \text{Mean} \left(Y_i - \hat{Y}_i \right)^2 \text{Error Squared}$$

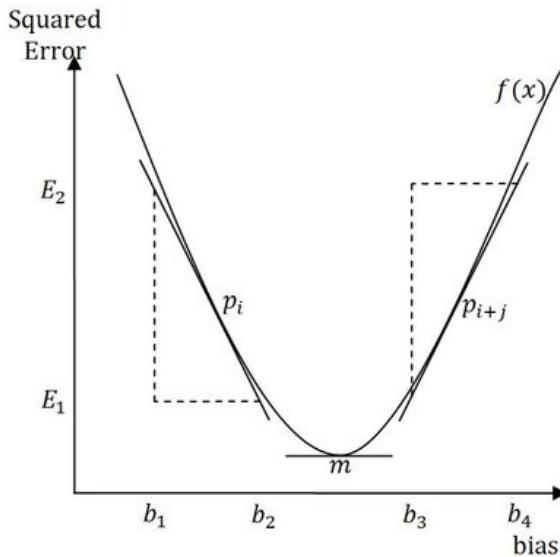
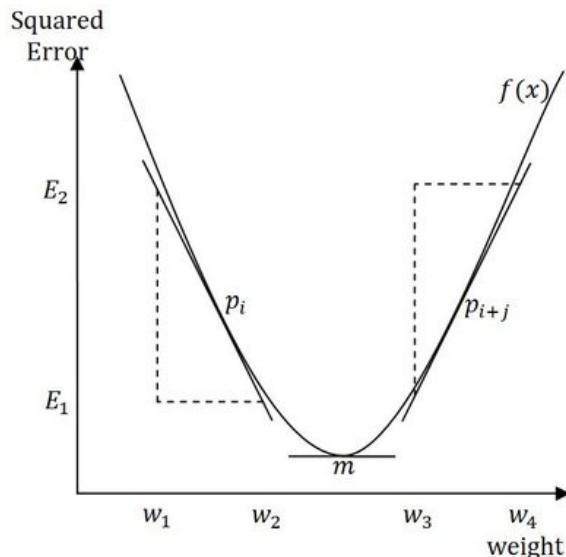


2) ML 주요 이론(Supervised Learning)

선형 회귀 분석(Linear Regression)

3) $y = wx+b$ 이므로 MSE는 $(y-y')^{**2} = (wx+b-y')^{**2}$: w 에 대한 2차 함수를 형성
 b 에 대해서도 2차 함수를 형성

MSE가 가장 작아지는 지점 : error값이 가장 작아지는 지점의 w 와 b !!
 w 와 b 의 변경을 통해 MSE가 최저(0)가 되는 지점을 찾는다 >>> 편미분 !!



2) ML 주요 이론(Supervised Learning)

선형 회귀 분석(Linear Regression)

4) w 와 b 에 대한 편미분을 통해 최적의 w , b 를 찾아간다.(편미분 후 1차함수의 기울기 !!)
 w, b 의 편미분값이 이전 w, b 에 대해 너무 커지지 않도록 학습률을 가중치를 추가 적용하며
MSE가 가장 작아지는 최적의 w , b 를 찾기 위해 반복 수행(epoch)

=> 주어진 x, y 데이터에 대해 w, b , 학습률을 변경해가며 MSE가 0에 가까운 w, b 를 찾는다

$$w = w - \eta \frac{\partial}{\partial w} MSE(w, b)$$
$$b = b - \eta \frac{\partial}{\partial b} MSE(w, b)$$

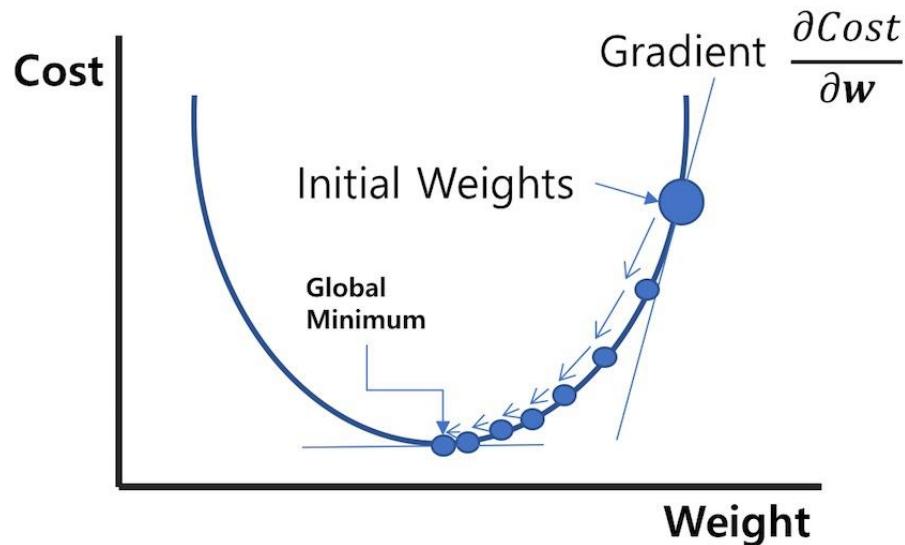
- <수식 5>. 가중치 갱신 수식
 - w : 가중치 벡터
 - 일반적으로 가중치 표시로 w 와 θ 를 사용합니다.
 - η : 학습률(Learning rate)
 - 현재 가중치의 변화량을 가중치 갱신에 적용할 때 사용하는 비율입니다.
 - b : 편차
 - 현재는 숫자인 선형회귀를 대상으로 합니다. 본 문서에서 b 는 벡터가 아닌 스칼라입니다.

2) ML 주요 이론(Supervised Learning)

선형 회귀 분석(Linear Regression)

5) w 와 b 에 대한 편미분을 통해 최적의 w , b 를 찾아간다.

=> 주어진 x, y 데이터에 대해 w, b , 학습률을 변경해가며 MSE가 0이 되는 w, b 를 찾는다



2) ML 주요 이론(Supervised Learning)

-로지스틱 회귀(Logistic Regression)

데이터가 어떤 범주에 속할 확률을 0과 1 사이의 값으로 예측하고
가능성이 더 높은 범주에 속하는 것으로 **분류(classification)**

2) ML 주요 이론(Supervised Learning)

The screenshot shows the scikit-learn API Reference page for the `LogisticRegression` class. The left sidebar lists various classes under `sklearn.linear_model`, with `LogisticRegression` selected. The main content area displays the class definition, a brief description, detailed documentation, and a "Parameters" section.

API Reference > sklearn.linear_model > LogisticRegression

LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='L2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

[source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default.** It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the [User Guide](#).

Parameters:

penalty : {‘l1’, ‘l2’, ‘elasticnet’, ‘None’}, default=‘l2’

Specify the norm of the penalty:

- `None` : no penalty is added;
- `‘l2’` : add a L2 penalty term and it is the default choice;

2) ML 주요 이론(Supervised Learning)

1. penalty: 'l1': L1 정규화, 'l2': L2 정규화
2. C: 기본(1), 정규화의 강도를 역수로 조절. 값이 작을수록 정규화가 강해집니다.
3. fit_intercept설명: 모델이 절편을 계산할지 여부를 결정합니다. 일반적으로 True(1)로 설정합니다.
4. solver설명: 최적화 문제를 해결하기 위해 사용할 알고리즘
'liblinear': 작은 데이터셋에 적합.'lbfgs'(기본값), 'sag', 'saga': 대규모 데이터셋에 적합
'newton-cg': L2 정규화에 적합.
5. max_iter설명: 최적화 알고리즘이 수렴할 때까지의 반복 횟수. 100
6. multi_class설명: 다중 클래스 분류를 위한 방법을 설정
'ovr': 이진 분류 문제로 각 클래스를 개별적으로 분류.
'multinomial': 클래스 간의 상대적 확률을 직접 모델링, auto
7. random_state설명: 솔버의 난수 발생기 시드를 설정, none

2) ML 주요 이론(Supervised Learning)

기본적으로 설정된 파라미터는

L2 규제(penalty='l2'),
중간 수준의 규제 강도(C=1.0),
'lbfgs' solver를 사용하며,
소규모 데이터셋과 이진/다중 클래스 분류에 적합하게 설정되어 있습니다.

데이터 특성(크기, 클래스 불균형 등)에 따라 파라미터를 조정하면 성능을 최적화할 수 있습니다.

분류 문제에서는

데이터가 고차원적이거나 특성이 많을 경우 과적합(overfitting)이 발생할 가능성이 높아,
L1(라쏘)이나 L2(릿지) 정규화를 통해 모델 복잡성을 제어할 필요성이 큽니다.

2) ML 주요 이론(Supervised Learning)

-로지스틱 회귀(Logistic Regression)

데이터가 어떤 범주에 속할 확률을 0과 1 사이의 값으로 예측하고 가능성이 더 높은 범주에 속하는 것으로 분류

예시) 스팸 메일 분류기.

어떤 메일을 받았을 때 : 스팸 메일일 확률이 0.5 이상이면 spam으로 분류하고, 확률이 0.5보다 작은 경우 정상 메일로 분류.

2진 분류(binary classification)

(소프트 맥스 함수를 사용할 경우 다항 로지스틱 회귀로 2개이상의 범주 예측 가능)

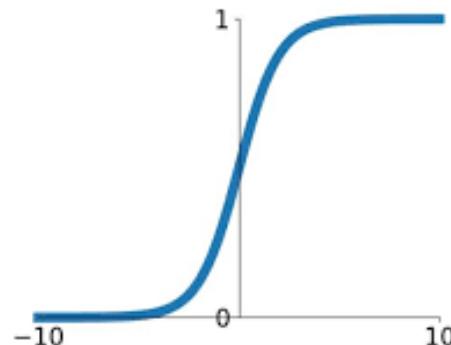
2) ML 주요 이론(Supervised Learning)

-로지스틱 회귀(Logistic Regression)

주어진 입력 변수(X)와 출력 변수(Y)의 관계를 가장 잘 설명하는 계수(가중치)와 절편을 추정하는 것이 목표. 이를 통해 새로운 입력 변수에 대한 출력 변수의 확률을 예측하고, 이진 분류를 수행할 수 있다.

시그모이드 함수가 로지스틱 회귀의 기본 활성 함수로 내장되어 있으며, 모델이 이진 분류를 수행할 때 자동으로 사용됩니다.

주요 요소 : 절편(intercept/Bias)/계수(Coefficient) 등

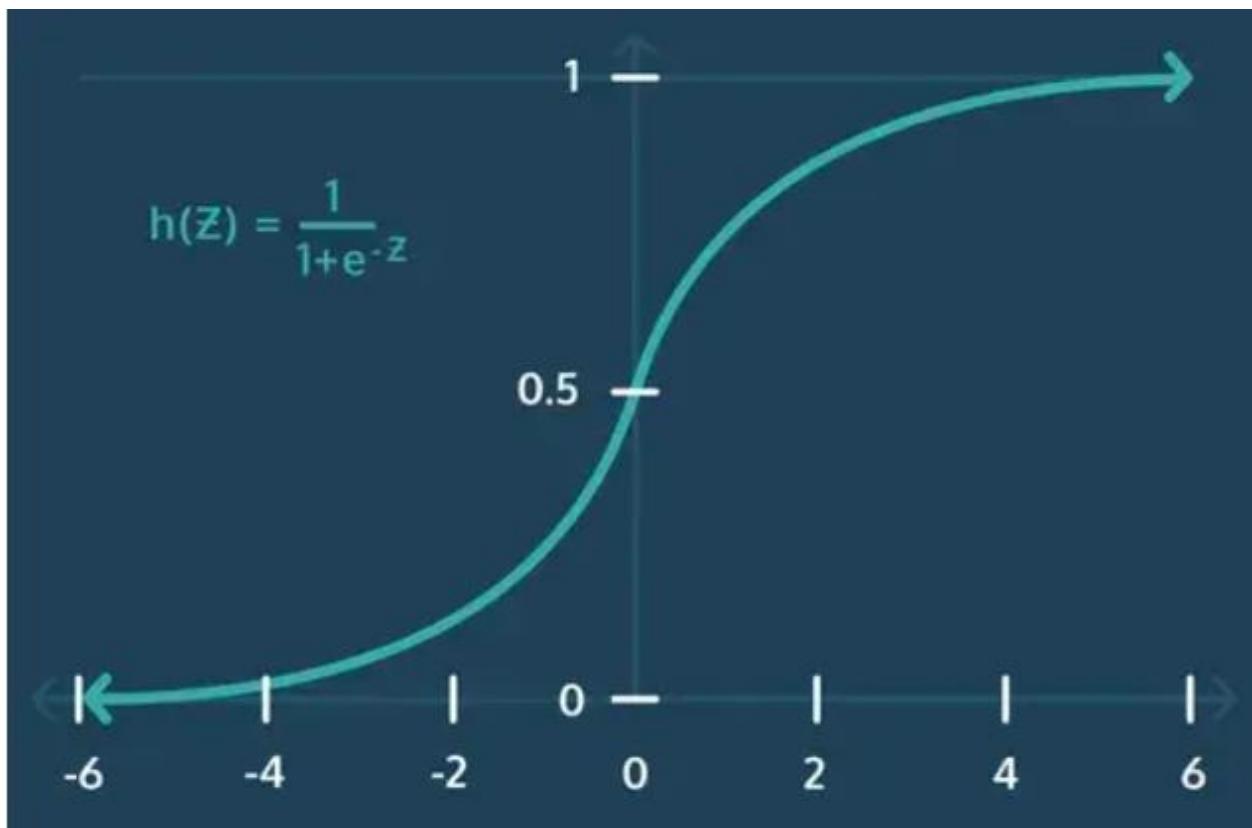


Sigmoid

2) ML 주요 이론(Supervised Learning)

- Sigmoid Function

확률을 0에서 1 사이로 커브 모양으로 나타내게 해주는 함수.
0부터 1 사이의 값으로 변환



2) ML 주요 이론(Supervised Learning)

-로지스틱 회귀(Logistic Regression)

로지스틱 회귀는 선형 모델인가 ???.

로지스틱 회귀가 선형 회귀와 구별되는 이유는 종속 변수의 변환에 있다.

로지스틱 회귀에서는 종속 변수를 이진(binary) 분류하기 위해 시그모이드 함수를 사용.(scikit learn 기준)

입력 변수의 선형 조합을 취한 후, 0과 1 사이의 값을 출력.

로지스틱 회귀는 선형 모델의 출력을 비선형 함수를 통해 변환하여 사용하는 방식으로, 비선형 문제를 해결하고 있다.

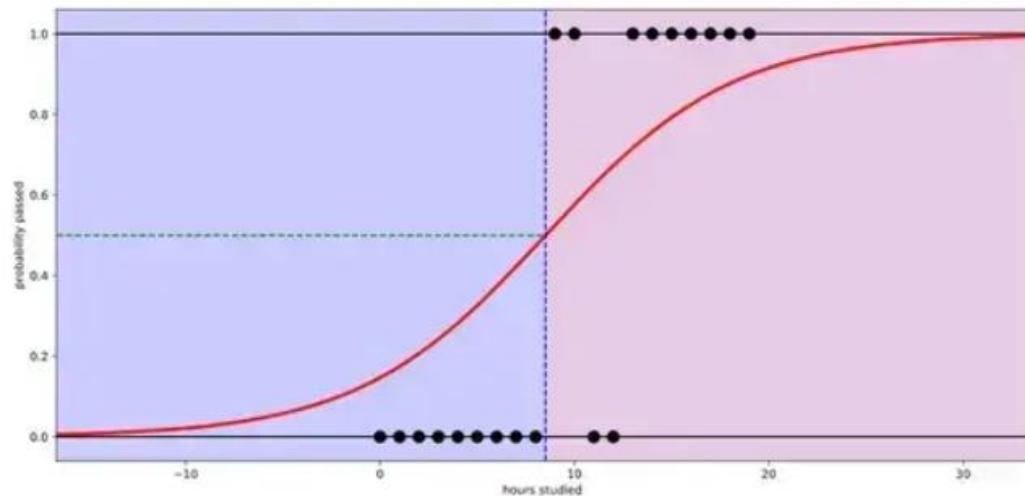
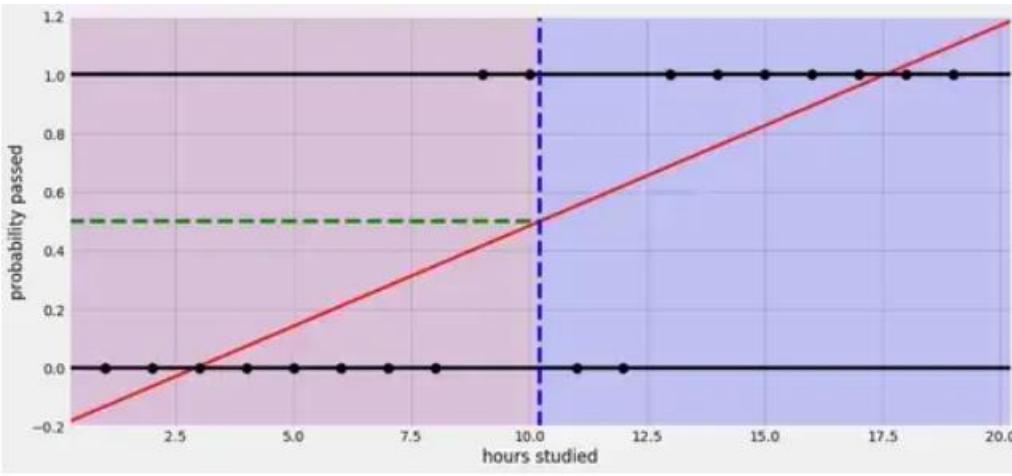
이러한 특성으로 인해 로지스틱 회귀는 비선형 모델로 간주되기도

2) ML 주요 이론(Supervised Learning)

- 선형 회귀 vs 로지스틱 회귀

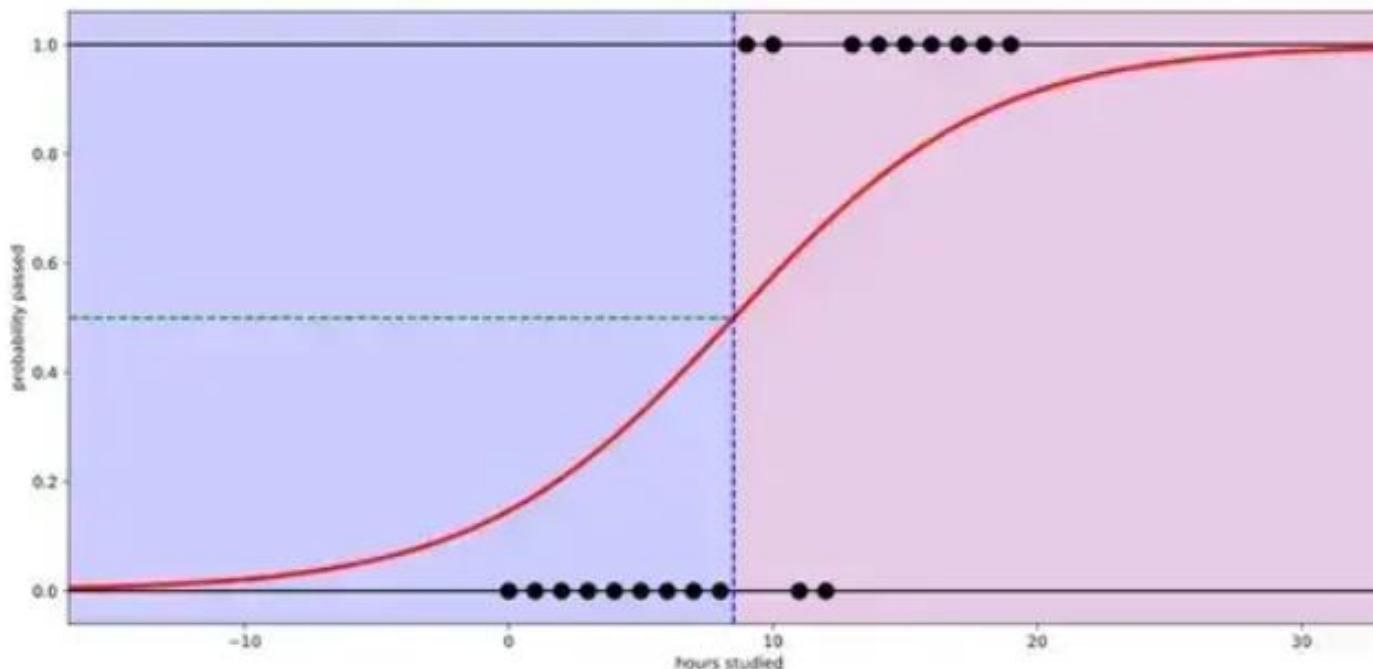
예시) 어떤 학생이 공부하는 시간에 따라 시험에 합격할 확률이
달라진다

그러나 합격 확률 1을 벗어날 수는 없다!!
(현실을 더 잘 반영하는 그래프는 ??)



2) ML 주요 이론(Supervised Learning)

- 시험에 합격할 확률이 0과 1 사이의 값으로 그려져야 합리적.
- 로지스틱 회귀의 예측 절차.
 1. 모든 속성(feature)들의 계수(coefficient)와 절편(intercept)을 0으로 초기화.
 2. sigmoid 함수에 넣어서 [0,1] 범위의 확률을 구한다.



2) ML 주요 이론(Supervised Learning)

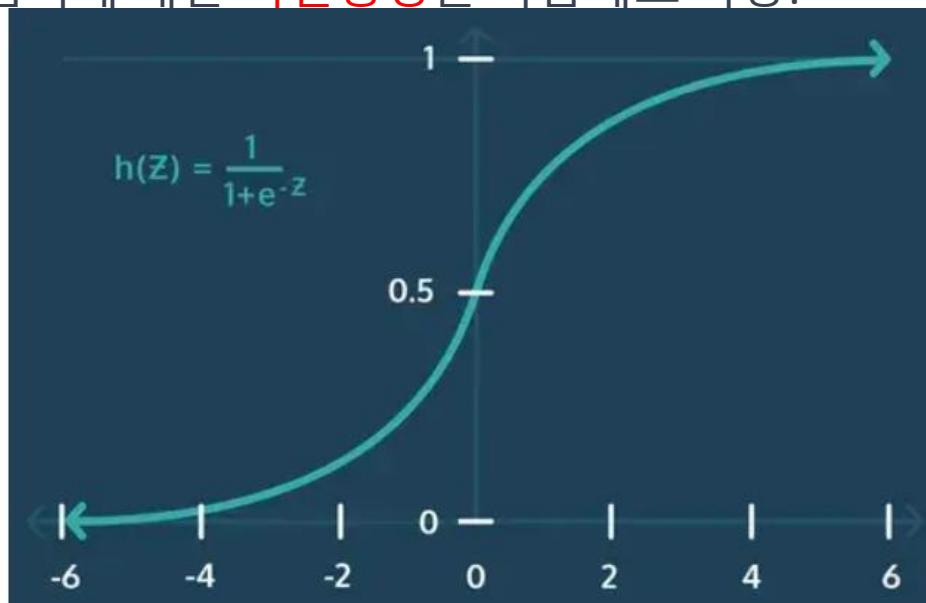
- Sigmoid Function

주로 이진 분류 문제에서 사용.

스팸 메일 여부, 암 진단 결과 예측 등 0과 1 사이의 확률 값으로 변환/분류를 수행.

신경망의 활성화 함수.

이진 분류를 위한 출력층에서 자주 사용되며,
신경망 모델이 입력에 대한 비선형성을 학습에도 사용.



2) ML 주요 이론(Supervised Learning)

- Sigmoid Function

자연 상수(e)는 수학적인 상수로서 약 2.71828로 표현되는 실수.

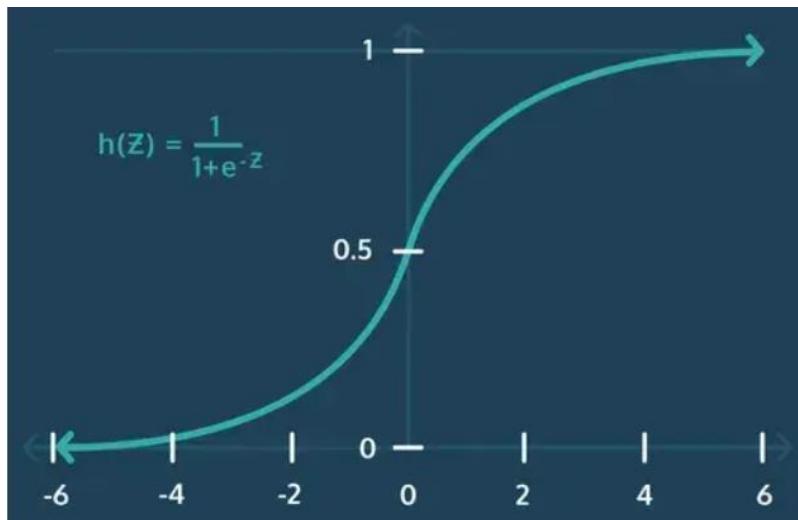
자연 상수 e 는 무리수(irrational number)로서, 무한 급수의 합으로 정의:

$$e = 1 + 1/1! + 1/2! + 1/3! + \dots$$

여기서 $n!$ 은 n 의 계승(factorial)을 의미하며, $0! = 1$ 로 정의.

무한 급수는 더해질수록 특정 값이 수렴하며, 극한값인 자연 상수 e 를 얻을 수 있다.

무리수(Irrational Number)는 정수나 분수로 표현되지 않는 실수



2) ML 주요 이론(Supervised Learning)

- Logistic Regression의 주요 파라미터 : C, penalty

1.C: 정규화 강도를 조절하는 매개변수. C 값이 작을수록 정규화가 강하게 적용되어 모델이 단순해지고 일반화 성능이 향상될 수 있다. C의 기본값은 1.0.

예시:

- C=0.1: 모델에 강한 정규화가 적용되어 단순한 모델이 선호.
- C=1.0: 모델에 중간 정도의 정규화가 적용.
- C=10.0: 모델에 약한 정규화가 적용되어 복잡한 모델이 선호.

2.penalty: 규제 유형을 지정하는 매개변수.

- 'l1': L1 규제는 모델의 가중치를 0에 가깝게 만들어 특성 선택(Feature Selection) 효과를 가지며, 모델을 희소하게 만들어준다. 즉, 일부 특성이 모델에 영향.
- 'l2': L2 규제는 모델의 가중치를 작게 만들어 모든 특성이 모델에 영향

* 로지스틱 회귀에서 정규화가 필요한 이유

과적합 방지, 다중 공선성 처리, 이상치 처리, 변수 스케일링 등

2) ML 주요 이론(Supervised Learning)

Regressions	
Simple Linear Regression	$y = b_0 + b_1x_1$
Multiple Linear Regression	$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$
Polynomial Linear Regression	$y = b_0 + b_1x_1 + b_2x_1^2 + \dots + b_nx_1^n$

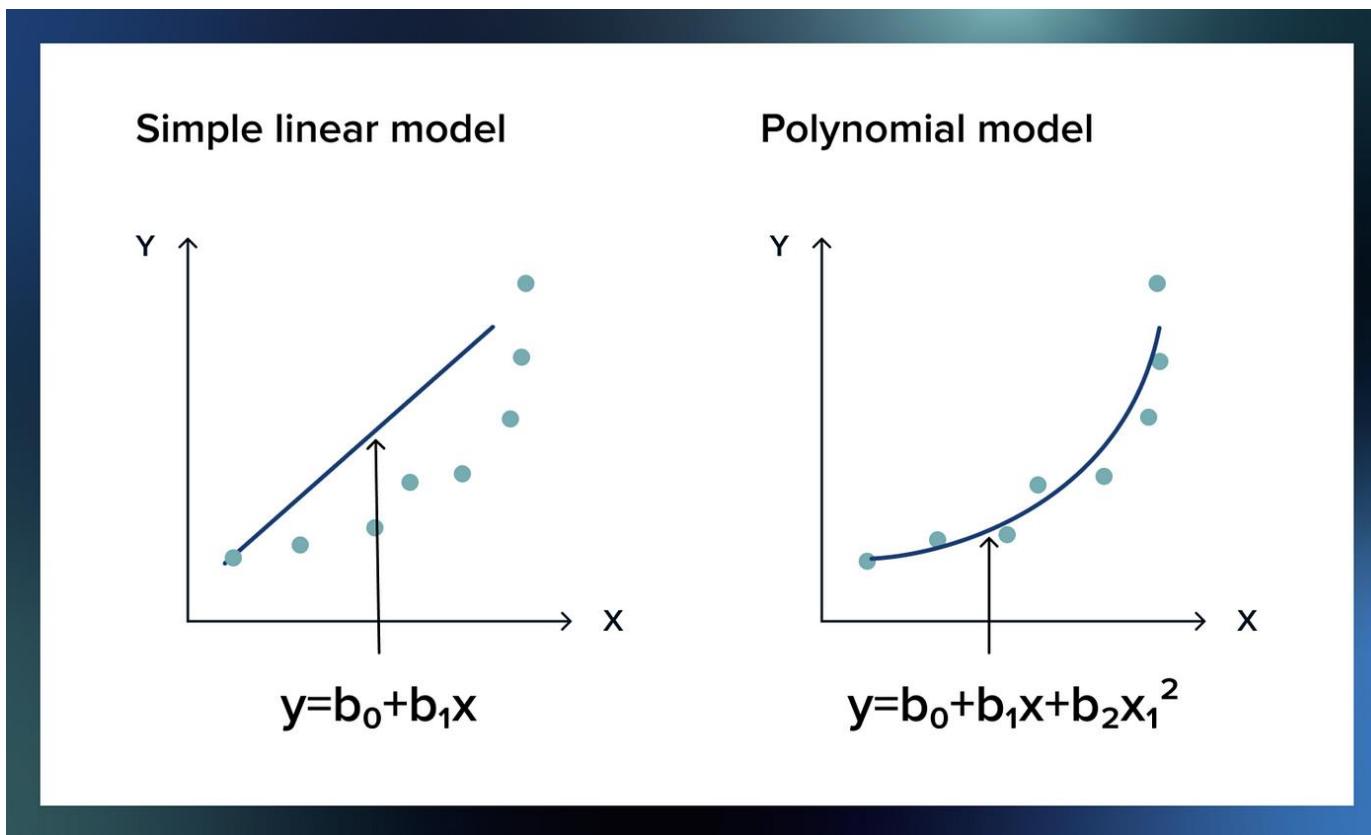
다중 회귀 (Multiple Regression)

여러 개의 독립 변수를 사용하여 종속 변수를 예측하는 회귀

다항 회귀 (Polynomial Regression)

하나 또는 여러 개의 독립 변수와 종속 변수 사이의 비선형 관계를 모델링
독립 변수의 다항식 항을 사용하는 회귀 분석 기법.

2) ML 주요 이론(Supervised Learning)



2) ML 주요 이론(Supervised Learning)

L1 정규화 vs L2 정규화

독립 변수가 2개 이상인 경우,
특히 변수 간 상관관계(다중공선성, multicollinearity 등) 문제를 완화하고
특성별 가중치를 적절히 반영하기 위해 사용

2) ML 주요 이론(Supervised Learning)

L1 정규화:

독립변수로 표현되는 특성값의 가중치 w_i 의 절댓값 합으로 페널티를 부과.

최적화 과정에서 일부 가중치를 0으로 만듦

→ 특성 선택(feature selection) 효과.

특징: 가중치가 0이 되면 해당 특성이 모델에서 제외됨.

L2 정규화:

독립변수로 표현되는 특성값의 가중치 w_i 의 제곱 합으로 페널티를 부과.

가중치를 0으로 만들지는 않지만 값을 작게 만듦

→ 모델 복잡도 감소.

특징: 모든 특성을 유지하되, 영향력을 균일하게 분산.

2) ML 주요 이론(Supervised Learning)

과적합 가능성이 높은 상황

예시 상황: 학생 시험 합격 여부(0: 불합격, 1: 합격)를 예측하는 로지스틱 회귀 모델.
데이터는 50명 학생의 기록,
특성은 공부 시간, 수면 시간, 부모님 학력, 스트레스 수준 등 20개.
특성 수가 샘플 수를 초과해 과적합 위험이 높음.

문제: 모델이 훈련 데이터의 노이즈까지 학습해,
테스트 데이터 정확도가 낮음(예: 훈련 98%, 테스트 60%).

과정: 손실 함수에 $\lambda \sum |w_i|$ 추가 (예: $\lambda=0.1$).

학습 중 가중치 조정: 예를 들어 모델이 학습하며 "공부 시간" $w_1=1.5$, "수면 시간" $w_2=0.8$, "부모님 학력" $w_3=0$, "스트레스 수준" $w_4=0$ 등 가중치를 조정하며 최적의
가중치 조합을 찾게됨

결과: 불필요한 특성("부모님 학력", "스트레스 수준")의 가중치가 0이 되어 제외.
모델은 "공부 시간", "수면 시간" 등 5개 특성만 사용.
효과: 특성 선택으로 모델 단순화, 테스트 정확도 75%로 개선.

2) ML 주요 이론(Supervised Learning)

1. 정규화가 적용된 손실 함수

정규화가 추가되면 손실 함수는 다음과 같이 변합니다:

- **기존 손실 함수:** 예를 들어, 선형 회귀의 경우 평균 제곱 오차(MSE):
$$L(w) = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$
- **정규화 추가:**
 - L1 정규화: $L_{\text{total}}(w) = L(w) + \lambda \sum |w_i|$
 - L2 정규화: $L_{\text{total}}(w) = L(w) + \lambda \sum w_i^2$ 여기서 λ 는 정규화 강도를 조절하는 하이퍼파라미터입니다.
- **효과:** 정규화 항($\lambda \sum |w_i|$ 또는 $\lambda \sum w_i^2$)은 가중치 w_i 의 크기에 따라 추가적인 페널티를 부과하므로, 전체 손실값이 커집니다.

2) ML 주요 이론(Supervised Learning)

2. 최적화 목표: 확장된 손실 함수 최소화

- 정규화가 적용된 상태에서 모델은 **확장된 손실 함수** $L_{\text{total}}(w)$ 를 최소화하는 가중치 w 를 찾습니다.
- 수학적으로:

$$\min_w [L(w) + \lambda R(w)],$$

여기서 $R(w)$ 는 L1($\sum |w_i|$) 또는 L2($\sum w_i^2$) 정규화 항.

- **왜 손실값이 커지나?**
 - 정규화 항은 가중치의 크기를 제한하기 위해 추가된 페널티.
 - 가중치가 크면 정규화 항의 값이 커져 전체 손실 L_{total} 이 증가.
 - 이를 줄이기 위해 모델은 **MSE(데이터 오차)**와 **정규화 항(모델 복잡도)** 간 균형을 맞추는 방향으로 가중치를 조정.

2) ML 주요 이론(Supervised Learning)

1. 포물면이 되는 이유

- **MSE 손실 함수의 정의:**

독립 변수가 2개(x_1, x_2)인 선형 회귀 모델에서, MSE 손실 함수는 다음과 같습니다:

$$L(w_1, w_2) = \frac{1}{n} \sum_{i=1}^n (y_i - (w_1 x_{i1} + w_2 x_{i2}))^2$$

여기서 w_1, w_2 는 가중치, y_i 는 실제 값, x_{i1}, x_{i2} 는 입력 데이터입니다.

- **2차 함수의 특성:**

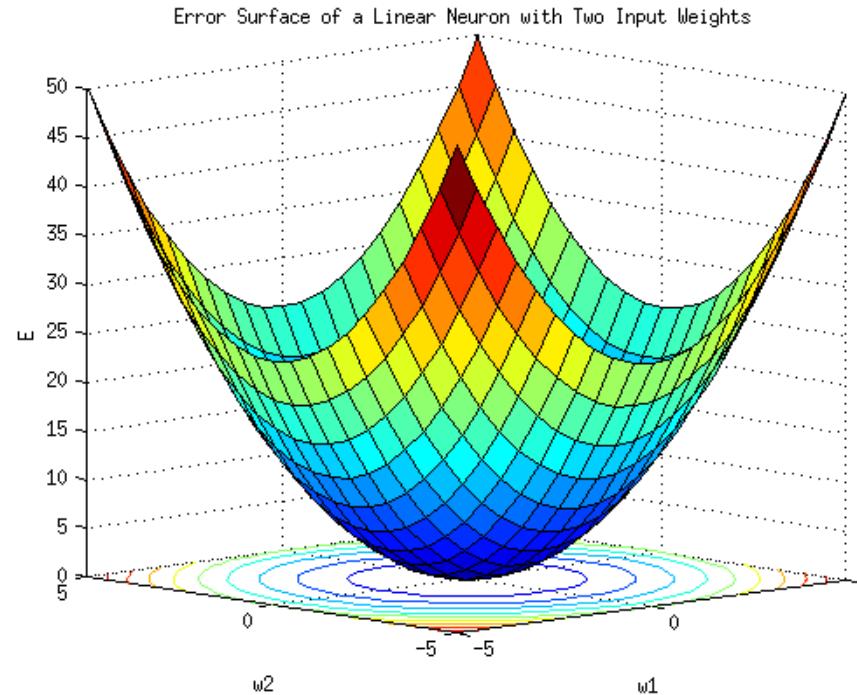
- $L(w_1, w_2)$ 는 w_1 와 w_2 에 대해 **2차 함수**입니다. 이를 전개하면:

$$L(w_1, w_2) = Aw_1^2 + Bw_2^2 + 2Cw_1w_2 + 2Dw_1 + 2Ew_2 + F$$

여기서 A, B, C, D, E, F 는 데이터에 따라 결정되는 상수(예: $A = \frac{1}{n} \sum x_{i1}^2$).

- 이 2차 다변수 함수는 3D 공간에서 **포물면(paraboloid)**을 형성합니다.

2) ML 주요 이론(Supervised Learning)



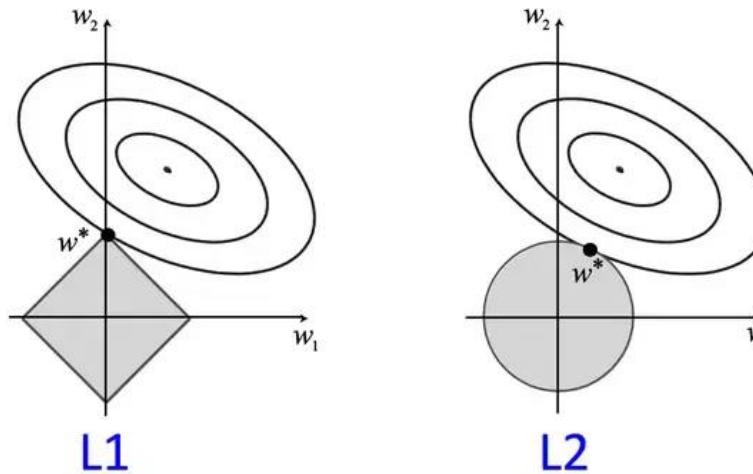
독립 변수가 2개인 경우,

MSE(Mean Squared Error) 손실 함수는 포물선(정확히는 1차원에서) 형태가 아니라
포물면(paraboloid, 3D에서) 형태를 가지며,

이를 가중치 공간(w_1, w_2)+Loss function에서 자른 등고선은 타원형

L_1, L_2 는 원의 방정식에 따라 마름모/원이 됩니다. ($|w_1| + |w_2|, w_1^2 + w_2^2 = C$)

2) ML 주요 이론(Supervised Learning)



중심이 원점(0, 0)인 경우:
 $x^2 + y^2 = r^2$

L1의 다이아몬드: $|w_1|+|w_2|$ 는 각 축에서 절댓값의 합을 제한하므로,
다이아몬드 모양(마름모)이 됩니다. 이 모양은 한 축이 0이 될 때 다른 축의 값을
극단적으로 늘릴 수 있어 특성 선택을 유도합니다.

L2의 원: $w_1^2+w_2^2$ 는 제곱 합을 제한하므로, 원형 제약을 형성합니다. 이는 가중치를
모든 방향으로 균일하게 작게 만들며, 특정 축이 0이 되는 경우를 피합니다.

2) ML 주요 이론(Supervised Learning)

다중회귀(Multiple regression) vs 다항회귀(Multi variant) :
Lasso vs Ridge

한 개 이상의 독립 변수 간의 선형 관계를 모델링하는 통계적 기법.

다중 회귀 : 변수값이 2개이상 존재하면 변수값에 대한 가중치를 조절해야 한다.

다항 회귀는 다중 회귀의 특별한 형태로,
독립 변수를 다항식 형태로 변환하여 비선형적인 관계를 모델링합니다.

라쏘(Lasso)와 **릿지(Ridge)**는
다중 회귀와 다항 회귀 모두에서 사용할 수 있으며,
모델의 복잡도를 조절하여 과적합을 방지

2) ML 주요 이론(Supervised Learning)

The screenshot shows the scikit-learn API Reference page for the `Lasso` class. The top navigation bar includes links for Install, User Guide, API (which is underlined), Examples, Community, More, and a search icon. The left sidebar lists various regression models, with `Lasso` selected. The main content area shows the `Lasso` class definition, its purpose as a Linear Model trained with L1 prior as regularizer, the optimization objective, and a note about its equivalence to the Elastic Net. It also provides links to the User Guide and detailed parameters for `alpha` and `fit_intercept`.

`class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')` [\[source\]](#)

Linear Model trained with L1 prior as regularizer (aka the Lasso).

The optimization objective for Lasso is:

$$(1 / (2 * n_samples)) * ||y - Xw||^2_2 + \alpha * ||w||_1$$

Technically the Lasso model is optimizing the same objective function as the Elastic Net with `l1_ratio=1.0` (no L2 penalty).

Read more in the [User Guide](#).

Parameters:

alpha : float, default=1.0
Constant that multiplies the L1 term, controlling regularization strength. `alpha` must be a non-negative float i.e. in `[0, inf)`.
When `alpha = 0`, the objective is equivalent to ordinary least squares, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advised. Instead, you should use the `LinearRegression` object.

fit_intercept : bool, default=True
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

2) ML 주요 이론(Supervised Learning)

The screenshot shows the scikit-learn API Reference page for the `Ridge` class. The left sidebar lists various regression models, and the main content area shows the detailed documentation for `Ridge`.

API Reference > sklearn.linear_model > Ridge

Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None) [source]
```

Linear least squares with l2 regularization.

Minimizes the objective function:

$$\|y - Xw\|^2_2 + \alpha * \|w\|^2_2$$

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

Read more in the [User Guide](#).

Parameters:

alpha : {float, ndarray of shape (n_targets,)}, default=1.0

Constant that multiplies the L2 term, controlling regularization strength. `alpha` must be a non-negative float i.e. in `[0, inf)`.

When `alpha = 0`, the objective is equivalent to ordinary least squares, solved by the [LinearRegression](#) object. For numerical reasons, using `alpha = 0` with the `Ridge` object is not advised. Instead, you should use the [LinearRegression](#) object.

- Ridge
- RidgeCV
- SGDRegressor
- ElasticNet
- ElasticNetCV
- Lars
- LarsCV
- Lasso
- LassoCV
- LassoLars
- LassoLarsCV
- LassoLarsIC
- OrthogonalMatchingPursuit
- OrthogonalMatchingPursuitCV
- ARDRegression
- BayesianRidge
- MultiTaskElasticNet
- MultiTaskElasticNetCV
- MultiTaskLasso
- MultiTaskLassoCV
- HuberRegressor
- QuantileRegressor
- RANSACRegressor
- TheilSenRegressor

2) ML 주요 이론(Supervised Learning)

-릿지 회귀(Ridge Regression)

다중 회귀 : 변수값이 2개이상 존재하면 변수값에 대한 가중치를 조절해야 한다.

(Multiple regression)

=> 복수의 정답이 존재할 수 있다.

=> 극단값의 경우 과적합의 경우가 발생(특정변수에 특화되는...)

=> 어떻게 가중치를 조절할 것인가?

=> 가중치의 제곱의 합이 특정값 이하가 되도록 규제가 필요하다 !!

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$



2) ML 주요 이론(Supervised Learning)

- 릿지 리그레션

릿지 회귀는 모든 변수들의 가중치를 동시에 축소시켜 모델이 덜 복잡하게 되도록 하여 과적합을 방지. 가능한 모든 변수를 고려!!

L2 규제를 추가하기 위해 가중치의 제곱값을 패널티로 사용.
과적합을 방지하고 모델의 일반화 성능을 향상

하이퍼파라미터인 alpha 값을 지정.
alpha 값이 클수록 규제의 강도가 커지므로, 가중치가 더 작아지도록 조정.

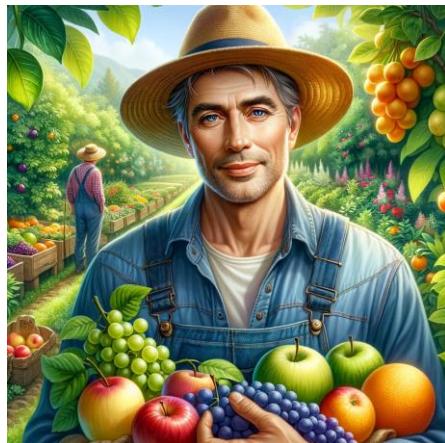
$$Ridge = \sum_{i=1}^n (y_i - y(x_i))^2 + \alpha \sum_{j=1}^p (w_j)^2$$

2) ML 주요 이론(Supervised Learning)

- 릿지 리그레션

L2 규제로 가중치의 제곱합의 패널티가 손실 함수에 추가되면 모델의 예측과 실제 값 사이의 오차가 증가하므로, Ridge 모델은 오차를 줄이기 위해 가중치 값을 조절하며 해당 변수의 영향력이 줄어든다.

Ridge 모델은 모든 변수를 고려하여 최적화하는 경향이 있으며, 과적합을 방지하고 모델의 일반화 성능을 향상시키는 효과를 가진다.



2) ML 주요 이론(Supervised Learning)

- 릿지 리그레션

기존 선형회귀에 변수값(특성값)이 출력에 영향을 미치는 가중치를 최소한으로 만드는 규제(regulation)을 부여한다.

특히 **특성값사이의 상관관계(multicollinearity)**가 주는 과적합을 방지
(사용자가 지정하는 알파값(α)이 규제의 강도를 조절)

변수 간의 상관관계가 낮은 경우에는, 모든 변수가 유용한 정보를 제공할 가능성이 있으므로, 라쏘회귀에 비교해서 릿지 회귀를 사용하는 것이 더 적절할 수 있다.

예) 캘리포니아 집값 예측 : 방의갯수와 욕실의 개수가 상관관계가 높을 수 있다.
상관관계의 중첩된 부분만큼을 감소시켜 출력이 나오도록 규제

2) ML 주요 이론(Supervised Learning)

-라쏘 회귀(Lasso Regression)

다중 회귀 : 변수값이 2개이상 존재하면 변수값에 대한 가중치를 조절해야 한다.

L1 규제를 추가하여 과적합을 방지하고 모델의 일반화 성능을 향상.

L1 규제를 추가하기 위해 가중치의 절대값을 패널티로 사용합니다

$$\text{Lasso} = \sum_{i=1}^n (y_i - y(x_i))^2 + \alpha \sum_{j=1}^p |w_j|$$



2) ML 주요 이론(Supervised Learning)

-Lasso Regression

라쏘 회귀는 변수 간의 상관관계가 높은 경우 특정 변수들의 가중치를 0으로 만들어 해당 변수들을 모델에서 완전히 제거할 수 있다. 모델의 해석력에 집중 !!

변수들 간의 상관관계를 분석하여 상관관계가 높게 나타나는 변수들 중에서 알파값을 조정하여 일부 변수들의 가중치를 0으로 만드는 알고리즘.

라쏘 회귀는 L1 규제(Regularization)를 사용하여 모델의 가중치를 조절.
L1 규제는 가중치 벡터의 절댓값의 합을 최소화하는 것으로, 작은 가중치 값을 0으로 만드는 경향이 있다.

상관관계가 높게 나타나는 변수들 중에서 알파값을 조정하면서 일부 변수들의 가중치를 0으로 만들 수 있다.



2) ML 주요 이론(Supervised Learning)

-Lasso Regression vs Ridge Regression

특징	라쏘 회귀	리지 회귀
가중치 제한	L1 규제	L2 규제
가중치 수렴 여부	0으로 수렴	0에 가까워짐
변수 선택성	변수 선택	모든 변수
희소성 유발	가능	어려움
예측 성능	일부 변수만 사용	모든 변수 사용

2) ML 주요 이론(Supervised Learning)

모델의 복잡도에 영향을 미치는 요소(model complexity)

특성(feature), 고차원 데이터(data), 모델 유형(model with parameters), 학습률(learning rates) 등 다양한 요소들

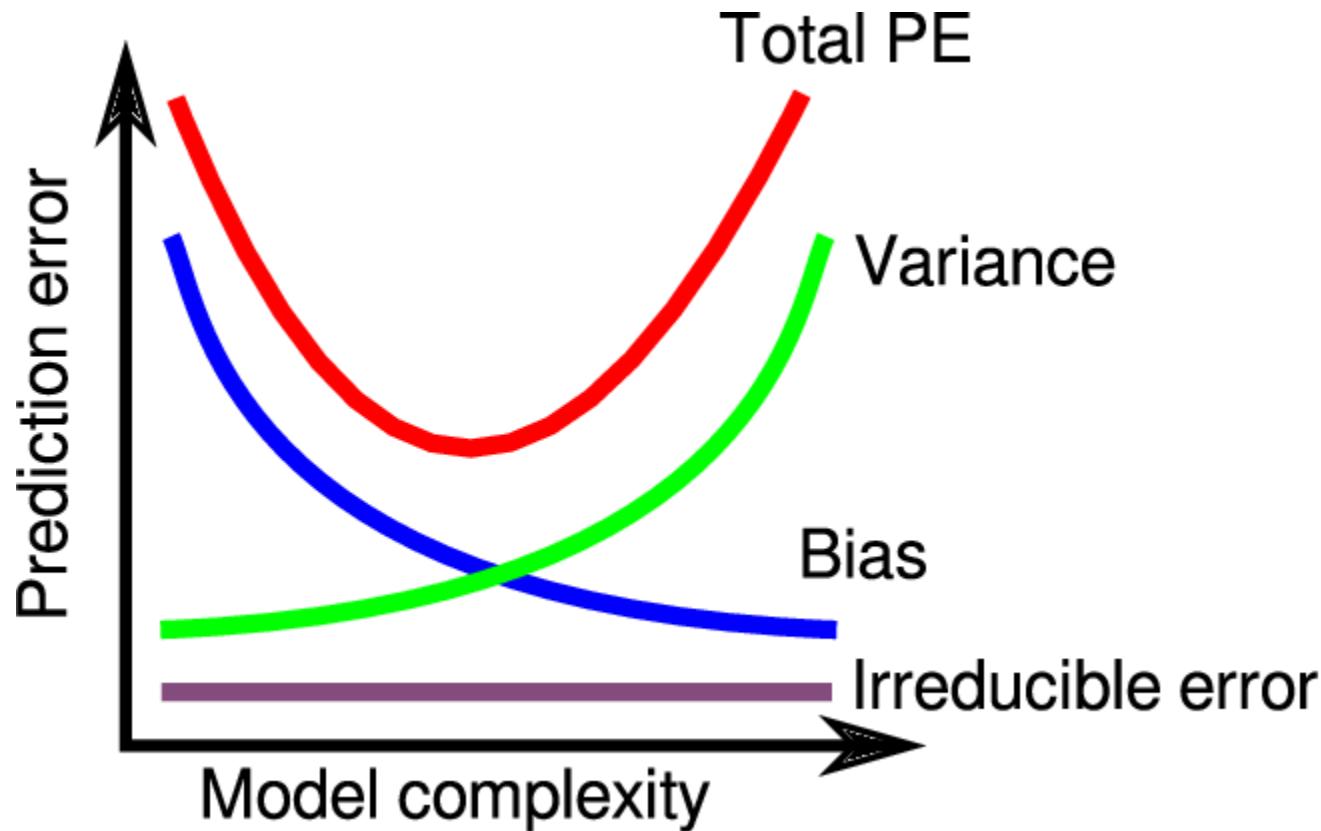
모델의 복잡도를 잘 조절하는 것이 모델 성능 향상과 과적합 방지에 중요한 역할을 합니다.

모델 복잡도는 과적합을 막기 위한 중요한 요소이며,

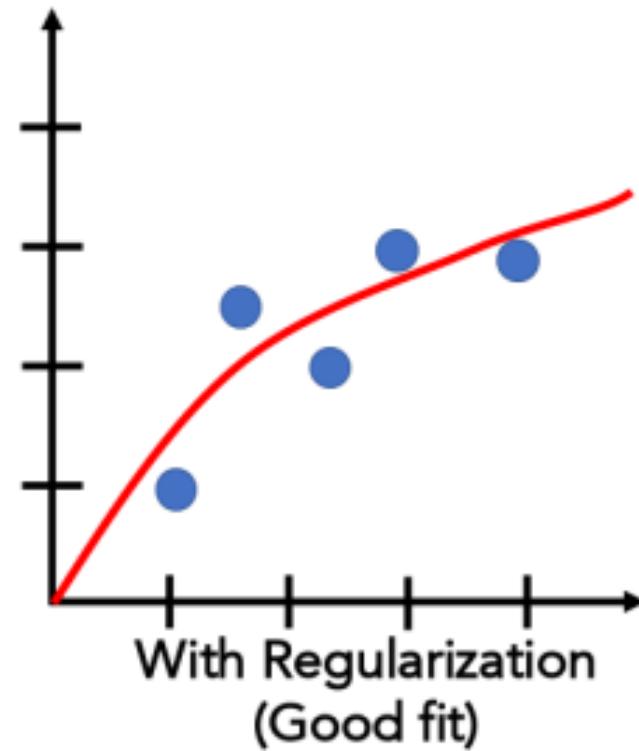
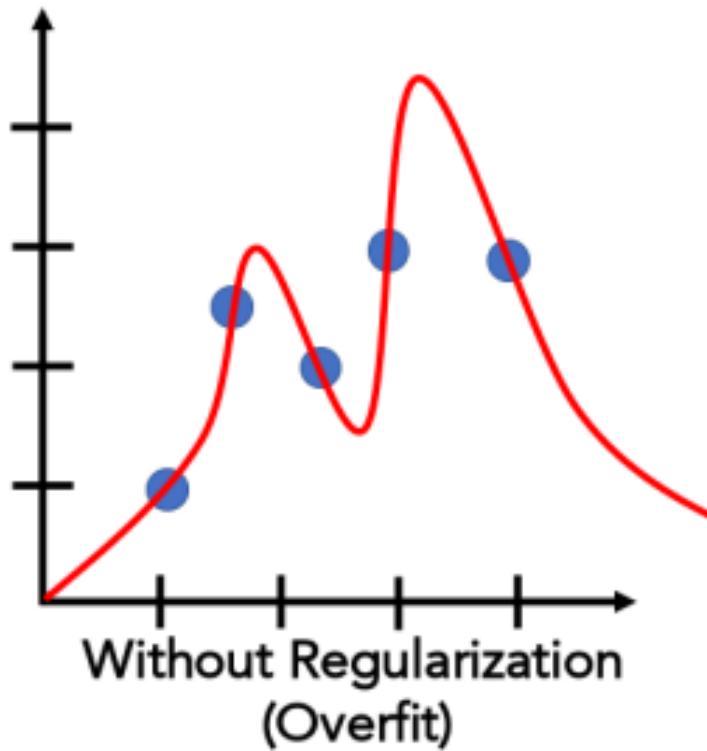
제어하는 방법으로는 정규화(regularization), 특성 선택, 학습률 조정 등이 있습니다.

2) ML 주요 이론(Supervised Learning)

The goal of regularization is to improve the overall fit



2) ML 주요 이론(Supervised Learning)



2) ML 주요 이론(Supervised Learning)

Low Bias & High Variance보다 Medium Bias & Medium Variance가 더 예측력(일반화)이 높은 모델이 된다.



2) ML 주요 이론(Supervised Learning)

-엘라스틱넷 회귀(Elastic Net Regression)

라쏘회귀와 릿지회귀를 결합하여 L1규제와 L2규제를의 장점을 결합하고 단점을 보완

The screenshot shows the scikit-learn API Reference page for the `ElasticNet` class. The URL is https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html. The page has a dark theme with yellow highlights for the class name and its methods.

Sidebar: A vertical sidebar on the left lists various regression classes: SGDRegressor, ElasticNet (highlighted in yellow), ElasticNetCV, Lars, LarsCV, Lasso, LassoCV, LassoLars, LassoLarsCV, LassoLarsIC, OrthogonalMatchingPursuit, OrthogonalMatchingPursuitCV, ARDRegression, BayesianRidge, MultiTaskElasticNet (highlighted in yellow), and MultiTaskElasticNetCV.

Breadcrumbs: The top navigation bar shows the path: Home > API Reference > `sklearn.linear_model` > `ElasticNet`.

Search: A search bar at the top right contains the text "ElasticNet".

On this page: A sidebar on the right lists the methods available for the `ElasticNet` class: fit, get_metadata_routing, get_params, path, predict, score, set_fit_request, set_params, set_score_request, and sparse_coef_. The `fit` method is highlighted in yellow.

Content: The main content area features a large yellow header "ElasticNet". Below it, the class definition is shown in code:

```
class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5,  
fit_intercept=True, precompute=False, max_iter=1000, copy_X=True, tol=0.0001,  
warm_start=False, positive=False, random_state=None, selection='cyclic') [source]
```

A brief description follows: "Linear regression with combined L1 and L2 priors as regularizer." Below that, it says "Minimizes the objective function:" and shows the mathematical formula:

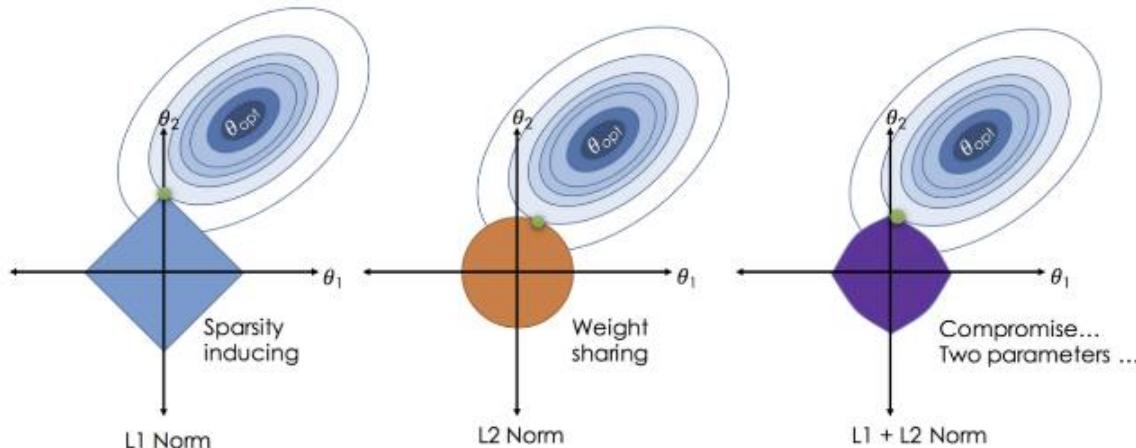
$$\frac{1}{2} \cdot n_{samples} \cdot ||y - Xw||^2 + \alpha * l1_ratio * ||w||_1 + 0.5 * \alpha * (1 - l1_ratio) * ||w||^2$$

2) ML 주요 이론(Supervised Learning)

-엘라스틱넷 회귀(Elastic Net Regression)

라쏘회귀와 릿지회귀를 결합하여 L1규제와 L2규제를의 장점을 결합하고 단점을 보완

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (y - \hat{y})^2 + r\lambda \sum_{j=1}^n |w_j| + \frac{1-r}{2}\lambda \sum_{j=1}^n w_j^2$$



2) ML 주요 이론(Supervised Learning)

-다항 회귀(Polynomial Regression)

선형 회귀의 확장된 형태,
독립 변수와 종속 변수 사이의 **비선형** 관계를 모델링. **곡선 또는 고차원**의 관계를 설명.

주요 파라미터

1. 차수(Degree): 2차 이상의 다항 회귀는 비선형 관계를 모델링. 차수가 높을수록 모델의 복잡성이 증가하므로 적절한 차수를 선택하는 것이 중요.

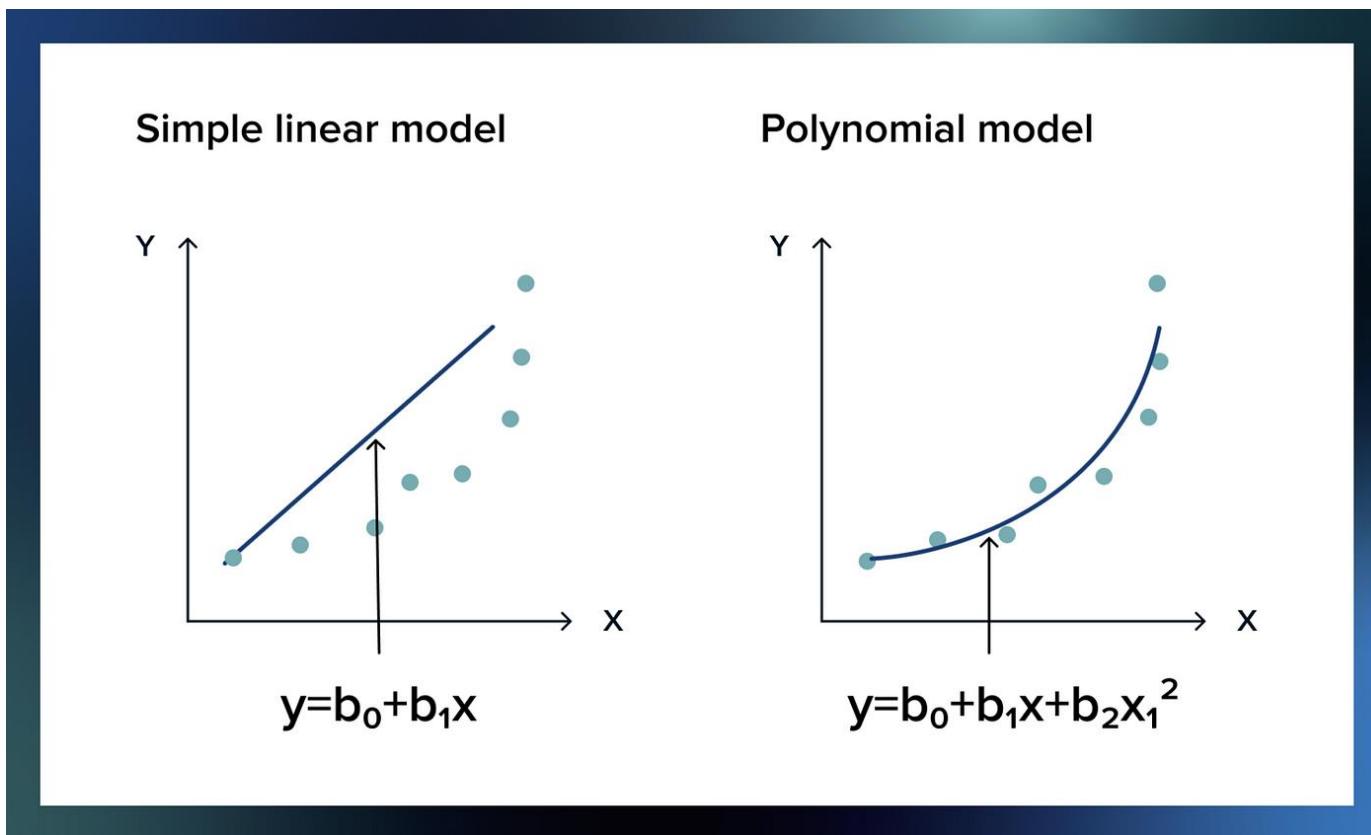
2. 상호작용(interaction): 상호작용 항은 각 독립 변수의 곱으로 표현되며, 독립 변수 간의 관계를 추가적으로 모델링할 수 있다.

3. 규제(regularization): 릿지(Ridge) 회귀나 라쏘(Lasso) 회귀와 같은 규제 방법을 사용하여 모델의 복잡성을 제어하고, 과적합을 방지할 수 있다.

2) ML 주요 이론(Supervised Learning)

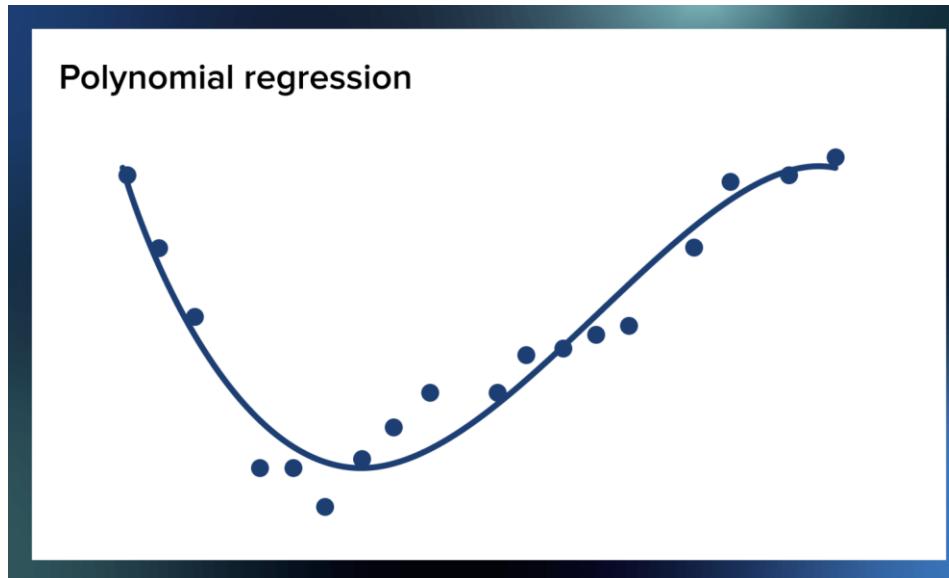
항목	다중회귀	다항회귀
독립 변수 수	여러 개 (X_1, X_2, \dots, X_n)	보통 1~2개, 거듭제곱 항 포함
관계 형태	선형 (직선적 관계)	비선형 (곡선적 관계 가능)
차원 확장	입력 변수 개수 확장	입력 변수 차수 확장
목적	다양한 요인의 영향 분석	곡선 형태의 데이터 모델링
사용 예	가격 예측, 마케팅 분석	곡선형 추세, 자연현상 모델링

2) ML 주요 이론(Supervised Learning)



2) ML 주요 이론(Supervised Learning)

-다항 회귀(Polynomial Regression)



Types of Polynomials

Linear ————— $ax + b = 0$

Quadratic ————— $ax^2 + bx + c = 0$

Cubic ————— $ax^3 + bx^2 + cx + d = 0$

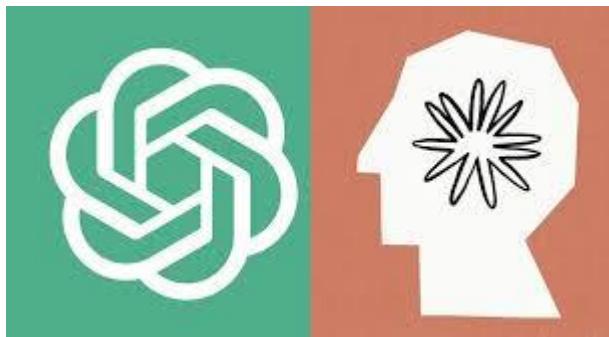
2) ML 주요 이론(Supervised Learning)

-다항 회귀(Polynomial Regression)

- 1. 비선형 관계:** 만약 캘리포니아 집값과 독립 변수들 간에 비선형적인 관계가 있다고 판단되면, 다항 회귀는 이러한 비선형성을 잡아낼 수 있는 유용한 도구가 될 수 있다.
- 2. 상호작용 효과:** 독립 변수들 간의 상호작용이 집값에 영향을 미칠 수 있는 경우, 다항 회귀를 통해 이러한 상호작용을 고려할 수 있다.
(면적과 방의 개수의 곱을 상호작용 항으로 추가 등)
- 3. 다중 공선성 해소:** 다항 회귀를 사용하여 변수들을 변환하면 상관관계가 줄어들 수 있다.
- 4. 모델 유연성:** 다항식의 차수를 조절함으로써 모델의 복잡성을 조절할 수 있으며, 데이터에 더 잘 적합시킬 수 있습니다.

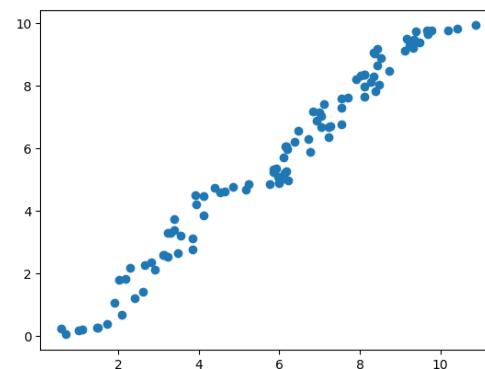
2) ML 주요 이론(Supervised Learning) 실습 1)

실습 1) 학습한 내용을 정리하고 추가적인 보완이 필요한 부분을 정리하고 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

Linear Regression 코드 실습



2) ML 주요 이론(Supervised Learning)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Generate random data for a regression problem
delta = np.random.rand(100, 1) # Random noise
X = sorted(10 * np.random.rand(100, 1) + delta) # Independent variable
Y = sorted(10 * np.random.rand(100)) # Dependent variable

# Plot the generated data
plt.plot(X, Y, 'o')
plt.show()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)

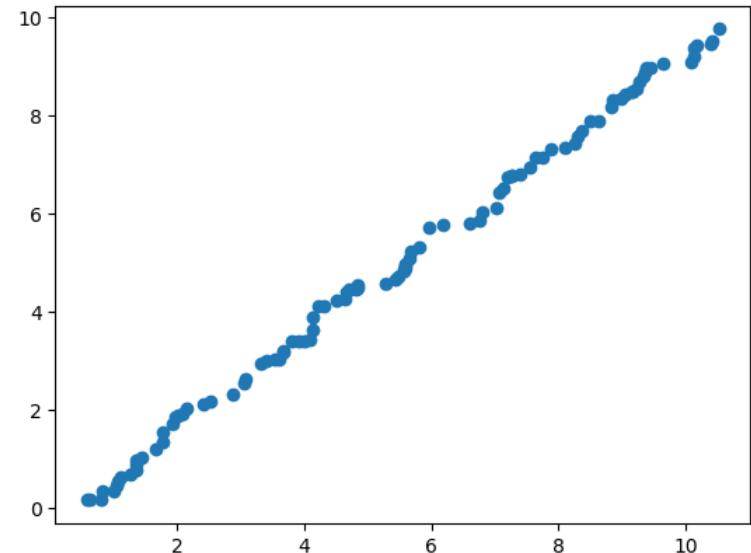
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Print the model coefficients and intercept
print("Linear regression coeff : {}".format(model.coef_))
print("Linear regression intercept : {}".format(model.intercept_))

# Evaluate the model on training and test data
print("Training Data evaluation : {}".format(model.score(X_train, y_train)))
print("Test Data evaluation : {}".format(model.score(X_test, y_test)))

# Make predictions on the test set
forecast = model.predict(X_test)

# Plot the original test data points and the regression line
plt.scatter(X_test, y_test)
plt.plot(X_test, forecast, '-b')
plt.show()
```



2) ML 주요 이론(Supervised Learning)

```
# Calculate MSE and RMSE
from sklearn.metrics import mean_squared_error
import numpy as np

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_rmse = np.sqrt(train_mse)
test_rmse = np.sqrt(test_mse)

print("Training Data MSE : {:.4f}".format(train_mse))
print("Test Data MSE : {:.4f}".format(test_mse))
print("Training Data RMSE : {:.4f}".format(train_rmse))
print("Test Data RMSE : {:.4f}".format(test_rmse))
```

```
Training Data MSE : 0.0593
Test Data MSE : 0.0924
Training Data RMSE : 0.2436
Test Data RMSE : 0.3040
```

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression(Process)

코딩 환경 구성: 코드작성에 필요한 패키지/라이브러리 업로드

```
import matplotlib.pyplot as plt, from sklearn.linear_model import LinearRegression
```

데이터 로드: numpy의 random함수로 x, y, delta 100개 생성, 정렬

```
sorted(10 * np.random.rand(100, 1)) + delta
```

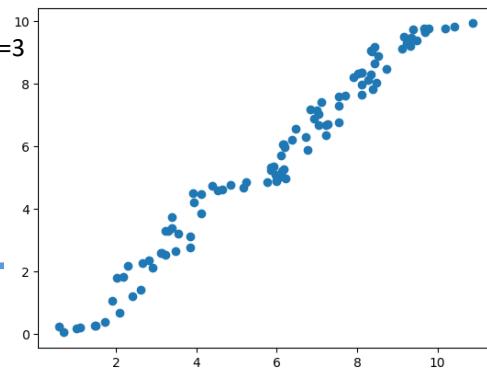
데이터 탐색(EDA): 데이터셋의 구성, 변수의 분포, 변수들 간의 상관 관계 등을 탐색. 시각화 도구
plt.plot(X, Y, 'o'), plt.show()

데이터 전처리: 학습에 적합한 형태로 가공. 변수 스케일링, 결측치 처리, 범주형 변수 인코딩 등.
StandardScaler(), train_test_split(X,Y,test_size = 0.3)

모델 학습: 예측 모델을 선택. 선형 회귀, 의사결정 트리, 랜덤 포레스트 등 다양한 모델.
model = LinearRegression() model.fit(X_train, y_train)

모델 평가: 학습된 모델의 성능을 평가. MSE, 결정 계수(R-squared) 등을 사용.
model.score(X_test, y_test)

2) ML 주요 이론(Supervised Learning)



Basic Linear Regression

Linear Regression

```
[123] 1 import numpy as np  
2 import matplotlib.pyplot as plt
```

```
[124] 1 from sklearn.linear_model import LinearRegression  
2 from sklearn.model_selection import train_test_split
```

```
1 delta = np.random.rand(100, 1) # 임의의 변동성 지정, 100행1열, 0~1  
2 X = sorted(10 * np.random.rand(100, 1)) + delta  
3 Y = sorted(10 * np.random.rand(100)) # 선형의 모습을 갖도록 정렬  
4  
5 plt.plot(X, Y, 'o')  
6 plt.show()
```

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

delta = np.random.rand(100, 1)

100행 1열의 배열을 생성하고, 각 요소에 0부터 1 사이의 임의의 값을 할당
생성된 delta는 임의의 변동성을 나타내는 값

X = sorted(10 * np.random.rand(100, 1)) + delta

0부터 10 사이의 임의의 값으로 구성된 100행 1열의 배열을 생성.
이 배열을 delta와 더한 후 정렬. X는 변동성이 추가된 후 정렬된 데이터가 된다.

Y = sorted(10 * np.random.rand(100))

0부터 10 사이의 임의의 값으로 구성된 100개의 데이터를 생성, 정렬
생성된 Y는 선형 모양을 갖는 데이터.

plt.plot(X, Y, 'o') : X와 Y의 값을 이용하여 산점도를 그리는 함수.

'o'는 데이터 포인트를 원 모양으로 표시하도록 지정.

plt.show() : 그래프를 화면에 출력.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
1 X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size = 0.3)
2 model = LinearRegression()
3 model.fit(X_train, y_train)
```

모델 객체의 생성: **model = LinearRegression()**

사용할 알고리즘 또는 모델의 종류를 선택하고 해당 알고리즘에 대한 초기화를 수행
모델의 구조와 초기화된 가중치 등을 메모리에 할당하는 과정.

fit 메서드 호출: **model.fit(X_train, y_train)**

모델 객체를 생성한 후 학습 데이터를 사용하여 모델을 학습시키는 과정
최적의 매개변수를 찾기 위해 학습을 수행

모델 객체의 생성과 fit 메서드 호출은 **연속적으로** 이루어져야 한다.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3)
```

데이터를 학습용과 테스트용으로 분할.

X는 입력 변수(features)를 나타내고, Y는 타깃 변수(target).

test_size는 테스트 데이터의 비율, 0.3은 전체 데이터 중 30%를 테스트로 사용
분할된 데이터는 각각 X_train, X_test, y_train, y_test에 저장.

model = LinearRegression():

선형 회귀 모델을 생성.

model.fit(X_train, y_train):

생성한 선형 회귀 모델에 학습 데이터인 X_train과 y_train을 적합(학습)시킨다.

모델은 주어진 학습 데이터에 대해 최적의 선형 관계를 학습.

모델은 입력 변수와 타깃 변수 간의 선형 관계를 가정하고,

학습 데이터를 기반으로 가장 적합한 회귀 계수를 찾아 model에 저장.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
1 print("Linear regression coeff : {}".format(model.coef_))
2 print("Linear regression intercept : {}".format(model.intercept_))
```

```
Linear regression coeff : [1.07019204]
Linear regression intercept : -0.7289340785209504
```

```
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.9678604483828037
test Data evaluation : 0.9848871649806696
```

계수와 절편을 구하고 모델의 설명력(결정계수)를 출력해 본다.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
print("Linear regression coeff : {}".format(model.coef_))
print("Linear regression intercept : {}".format(model.intercept_))
```

format 함수를 사용하여 출력문에 회귀 계수와 절편 값을 삽입.

"Linear regression coeff : {}"에서 {} 부분에는 model.coef_에서 얻은 회귀 계수 값이, "Linear regression intercept : {}"에서 {} 부분에는 model.intercept_에서 얻은 절편 값이 삽입된다.

각 입력 특성이 출력 변수에 미치는 영향과 모델의 기준점을 파악할 수 있다.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
print("training Data evaluation : {}".format(model.score(X_train, y_train)))
print("test Data evaluation : {}".format(model.score(X_test, y_test)))
```

model.score(X_train, y_train):

훈련 데이터 X_{train} 과 실제 정답 데이터 y_{train} 에 대한 예측 성능을 평가.

score 메서드는 모델의 결정 계수(coefficient of determination)를 반환.

결정 계수는 예측값과 실제값 사이의 총 변동 중에서 모델이 설명하는 변동의 비율
값은 0과 1 사이의 범위를 가지며, 1에 가까울수록 예측 성능이 좋다는 의미.

model.score(X_test, y_test):

테스트 데이터 X_{test} 와 실제 정답 데이터 y_{test} 에 대한 예측 성능을 평가.

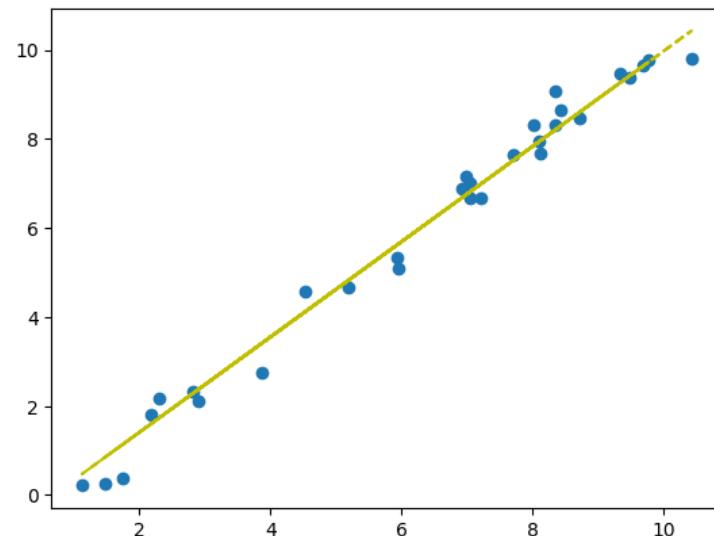
format 함수를 사용하여 출력문에 평가 점수 값을 삽입.

"training Data evaluation : {}"에서 {} 부분에는 훈련 데이터에 대한 평가 점수가, "test Data evaluation : {}"에서 {} 부분에는 테스트 데이터에 대한 평가 점수가 삽입.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
1 forecast = model.predict(X_test)
2
3 plt.scatter(X_test, y_test) #테스트 데이터에 대한 예측선
4 plt.plot(X_test, forecast, '--y')
```



테스트 데이터로 예측한 결과를 시각화해 본다

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

forecast = model.predict(X_test):

model을 사용하여 테스트 데이터 x_test에 대한 예측값을 생성.
predict 메서드는 입력 데이터에 대한 예측값을 반환.

plt.scatter(X_test, y_test):

테스트 데이터의 실제 값인 x_test와 y_test를 산점도로 시각화

plt.plot(X_test, forecast, '--y'):

테스트 데이터에 대한 예측값인 forecast를 선으로 시각화
'--y'는 선의 스타일을 나타내며, 여기서는 노란색 점선으로 지정.

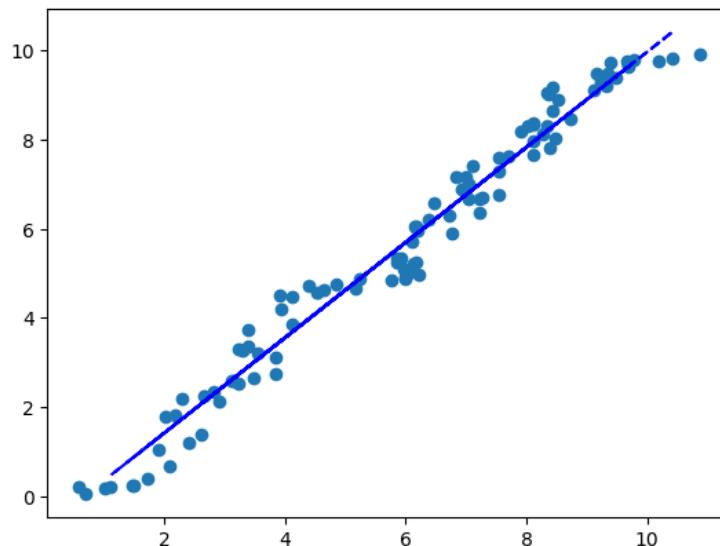
산점도와 예측 선이 함께 그려진 그래프가 생성.

모델이 테스트 데이터에 대해 얼마나 정확한 예측을 수행하는지 시각적으로 확인

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

```
1 forecast = model.predict(X_test)  
2  
3 plt.scatter(X,Y) # 전체 데이터에 대한 예측선  
4 plt.plot(X_test, forecast, '--b')
```



전체 데이터와 비교하여 예측그래프를 비교하며 오차/일반화에 대해 생각해 본다.

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

forecast = model.predict(X_test):

테스트 데이터 X_{test} 에 대한 예측값을 생성

plt.scatter(X, Y):

전체 데이터인 x 와 y 를 산점도로 시각화

plt.plot(X_test, forecast, '--b'):

전체 데이터에 대한 예측값인 $forecast$ 를 선으로 시각화

'--b'는 파란색 점선으로 지정.

전체 데이터에 대한 산점도와 예측 선이 함께 그려진 그래프가 생성

전체 데이터에 대해 얼마나 정확한 예측을 수행하는지 시각적으로 확인

2) ML 주요 이론(Supervised Learning)

Basic Linear Regression

코딩 환경 구성: 코드작성에 필요한 패키지/라이브러리 업로드

```
import matplotlib.pyplot as plt, from sklearn.linear_model import LinearRegression
```

데이터 로드: numpy의 random함수로 x, y, delta 100개 생성, 정렬

```
sorted(10 * np.random.rand(100, 1)) + delta
```

데이터 탐색(EDA): 데이터셋의 구성, 변수의 분포, 변수들 간의 상관 관계 등을 탐색. 시각화 도구
plt.plot(X, Y, 'o'), plt.show()

데이터 전처리: 학습에 적합한 형태로 가공. 변수 스케일링, 결측치 처리, 범주형 변수 인코딩 등.

```
StandardScaler(), train_test_split(X,Y,test_size = 0.3)
```

모델 학습: 예측 모델을 선택. 선형 회귀, 의사결정 트리, 랜덤 포레스트 등 다양한 모델.

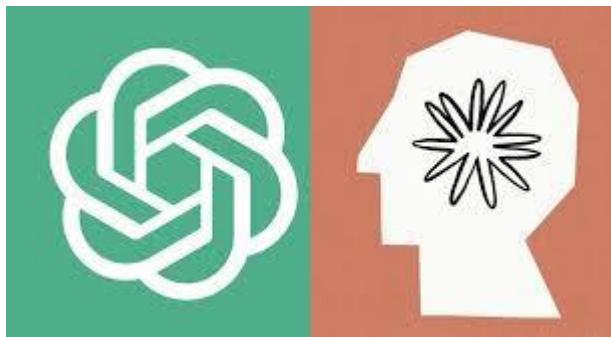
```
model = LinearRegression() model.fit(X_train, y_train)
```

모델 평가: 학습된 모델의 성능을 평가. MSE, 결정 계수(R-squared) 등을 사용.

```
model.score(X_test, y_test)
```

2) ML 주요 이론(Supervised Learning)

실습 1) 학습한 내용을 정리하고 유사한 코드를 생성하여 내용을 정리하고 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

캘리포니아 집값
예측

Linear Regression



California House Price Data Set

2) ML 주요 이론(Supervised Learning)

*Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:

- MedInc median income in block group
- HouseAge median house age in block group
- AveRooms average number of rooms per household
- AveBedrms average number of bedrooms per household
- Population block group population
- AveOccup average number of household members
- Latitude block group latitude
- Longitude block group longitude

:Missing Attribute Values: None

The target variable : the median house value for California districts, unit by \$100,000.

The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars (\$100,000).

a block group : a population of 600 to 3,000 people.

A household : a group of people residing within a home. Since the average number of rooms and bedrooms in this dataset are provided per household, these columns may take surprisingly large values for block groups with few households and many empty houses, such as vacation resorts.

California House Price Data Set

2) ML 주요 이론(Supervised Learning)

데이터셋 특성:

인스턴스 수: 20,640개

속성 수: 8개의 수치형 예측 속성과 1개의 타겟 변수

속성 정보:

MedInc: 블록 그룹 내 중간 소득

HouseAge: 블록 그룹 내 중간 주택 연령

AveRooms: 가구당 평균 방 수

AveBedrms: 가구당 평균 침실 수

Population: 블록 그룹 인구

AveOccup: 가구당 평균 거주자 수

Latitude: 블록 그룹 위도

Longitude: 블록 그룹 경도

결측치: 없음

타겟 변수: 타겟 변수는 캘리포니아 지역의 중간 집값으로, 단위는 10만 달러(\$100,000)입니다.

추가 설명:

블록 그룹: 600명에서 3,000명 사이의 인구를 가진 지역 단위입니다.

가구: 집 안에 거주하는 사람들의 그룹을 의미합니다. 이 데이터셋에서 가구당 평균 방 수와 침실 수는 제공되는데, 가구 수가 적고 빈집(예: 휴양지)이 많은 블록 그룹에서는 예상보다 큰 값을 가질 수 있습니다.

2) ML 주요 이론(Supervised Learning)

California House Price 데이터 세트 요약

주택 가격(Target Variable): 데이터셋의 주요 타깃 변수.
블록별 주택의 중간 가격(median house value).

특성(Features):

주택의 위치: 위도(latitude)와 경도(longitude)를 포함한 위치 정보.

인구 특성: 지역의 인구 수(population) 등.

주택 특성: 주택의 평균방수(avg rooms), 평균 침실 수(avg bedrooms), 세대원수(avg Occupants) 등.

지역 특성: 지역의 중간 소득(median income, 1만달러 기준) 등.

타겟 별류 : 블록별 집값(10만불 단위)

기타 : 블록(지역별 600~3000명 거주),

리조트가 포함되어 있어서 가구원수가 매우 크거나 없을 수도 있다.:

2) ML 주요 이론(Supervised Learning)

실습) 코드의 시작전에 캘리포니아 집값 데이터에 대한 description을 충분히 이해하고 공유해 주세요

2) ML 주요 이론(Supervised Learning)

데이터 로드: 데이터셋을 로드하여 분석에 활용할 수 있도록 준비.

```
fetch_california_housing()
```

```
housing_df = pd.DataFrame(housing.data, columns = housing.feature_names)
```

데이터 탐색(EDA): 데이터셋의 구성, 변수의 분포, 변수들 간의 상관 관계 등을 탐색. 시각화 도구
housing_df.head(), housing_df.describe() sns.pairplot(housing_df)

데이터 전처리: 학습에 적합한 형태로 가공. 변수 스케일링, 결측치 처리, 범주형 변수 인코딩 등.
StandardScaler()

모델 학습: 주택 가격 예측 모델을 선택. 선형 회귀, 의사결정 트리, 랜덤 포레스트 등 다양한 모델.
model = LinearRegression() model.fit(X_train, y_train)

모델 평가: 학습된 모델의 성능을 평가. MSE, 결정 계수(R-squared) 등을 사용.
model.score(X_test, y_test)

모델 튜닝: 모델의 성능을 향상시키기 위해 하이퍼파라미터를 조정, 교차 검증 등.
cross_val_score(model, housing.data, housing.target, cv=10, scoring='r2')

결과 해석: 모델의 결과를 해석하고 주택 가격에 영향을 미치는 주요 요소를 식별.

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 from sklearn.linear_model import LinearRegression  
2 from sklearn.model_selection import train_test_split  
3 import numpy as np  
4 import matplotlib.pyplot as plt  
5 import pandas as pd
```

모델링에 필요한 환경을 구성

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
3 from sklearn.datasets import fetch_california_housing  
4 housing = fetch_california_housing()  
5  
6 print(housing.keys())  
7 print(housing.DESCR)  
  
dict_keys(['data', 'target', 'frame', 'target_names', 'feature_names', 'DESCR'])  
... _california_housing_dataset:  
  
California Housing dataset
```

캘리포니아 집값 예측 데이터를 업로드하고 상세내용을
출력/분석/이해

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
from sklearn.datasets import fetch_california_housing:  
scikit-learn의 fetch_california_housing 함수 :캘리포니아 주택 데이터셋을 가져온다
```

```
housing = fetch_california_housing():  
캘리포니아 주택 데이터셋을 housing 변수에 저장.
```

```
print(housing.keys()):  
housing 데이터셋의 속성들의 키를 출력.  
데이터셋의 속성들은 딕셔너리 형태로 저장되어 있으며,  
어떤 속성들이 있는지 확인할 수 있다.
```

```
print(housing.DESCR): 데이터셋의 자세한 설명을 출력  
데이터셋의 특징과 속성들에 대한 정보를 알 수 있다.
```

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
3 housing_df = pd.DataFrame(housing.data, columns = housing.feature_names)
4 housing_df['Price'] = housing.target
5 housing_df.head()
```

	HedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price	+
0	8.33	41.00	6.98	1.02	322.00	2.56	37.88	-122.23	4.53	
1	8.30	21.00	6.24	0.97	2401.00	2.11	37.86	-122.22	3.58	
2	7.26	52.00	8.29	1.07	496.00	2.80	37.85	-122.24	3.52	
3	5.64	52.00	5.82	1.07	558.00	2.55	37.85	-122.25	3.41	
4	3.85	52.00	6.28	1.08	565.00	2.18	37.85	-122.25	3.42	

업로드된 자료를 판다스의 데이터 프레임으로 만들고
출력/분석/이해.

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

housing_df = pd.DataFrame(housing.data, columns = housing.feature_names):

housing.data를 기반으로 pandas의 DataFrame을 생성

housing.data는 캘리포니아 주택 데이터셋의 입력 특성값들을 담고 있는 배열.

columns 매개변수를 사용하여 각 열의 이름을 housing.feature_names으로 지정.

→ DataFrame의 열 이름이 주택 데이터셋의 특성 이름과 일치하게 된다.

housing_df['Price'] = housing.target:

목표 변수인 주택 가격을 Price라는 열로 추가.

housing.target은 주택 가격을 담고 있는 배열.

DataFrame의 새로운 열로 추가하면 데이터셋에 주택 가격 정보가 포함.

housing_df.head():

생성된 DataFrame의 처음 5개 행을 출력. 마지막 5개는 tail() 메서드

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 housing_df.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price	+
count	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	20640.00	
mean	3.87	28.64	5.43	1.10	1425.48	3.07	35.63	-119.57	2.07	
std	1.90	12.59	2.47	0.47	1132.46	10.39	2.14	2.00	1.15	
min	0.50	1.00	0.85	0.33	3.00	0.69	32.54	-124.35	0.15	
25%	2.56	18.00	4.44	1.01	787.00	2.43	33.93	-121.80	1.20	
50%	3.53	29.00	5.23	1.05	1166.00	2.82	34.26	-118.49	1.80	
75%	4.74	37.00	6.05	1.10	1725.00	3.28	37.71	-118.01	2.65	
max	15.00	52.00	141.91	34.07	35682.00	1243.33	41.95	-114.31	5.00	

데이터의 통계적 수치 출력/분석/이해

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

`housing_df.describe()`

`housing_df`의 요약 통계를 제공하는 메서드
각 열의 기술 통계량을 확인할 수 있다.

count: 각 열에 대한 비어 있지 않은(non-null) 값의 개수

mean: 각 열의 평균

std: 각 열의 표준 편차

min: 각 열의 최소값

25%: 각 열의 25번째 백분위수.

50%: 각 열의 중간값(median).

75%: 각 열의 75번째 백분위수.

max: 각 열의 최대값.

데이터의 분포, 중심 경향성, 분산 등을 파악하는 데 도움

예를 들어, 평균과 중간값이 크게 다르다면 해당 열의 데이터는 이상치가 있을 수 있음

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 for i, col in enumerate(housing_df.columns): # 열의 인덱스와 값을 동시에 가져온다
2     plt.figure(figsize = (8,4))
3     plt.plot(housing_df[col]) # 해당 열의 데이터로 그리기
4     plt.title(col)          # 해당 열로 제목 쓰기
5     plt.xlabel('Location') # 임의의 위치
6     plt.tight_layout()
7
```

캘리포니아 주택 데이터셋의 각 열을 시각화하는 과정.
특성별로 라인 그래프 출력/분석/이해

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

`for i, col in enumerate(housing_df.columns):`

housing_df의 열 인덱스와 해당 열의 값을 가져오기 위한 반복문
`enumerate` 함수를 사용하여 열의 인덱스와 값을 동시에 가져온다.

`plt.figure(figsize=(8, 4))`: 그림의 크기를 지정. (가로 8, 세로 4) 크기의 그림을 생성.

`plt.plot(housing_df[col])`: 해당 열의 데이터를 이용하여 선 그래프를 그리는 부분.
housing_df[col]은 해당 열의 데이터를 의미합니다.

`plt.title(col)`: 그래프의 제목을 설정

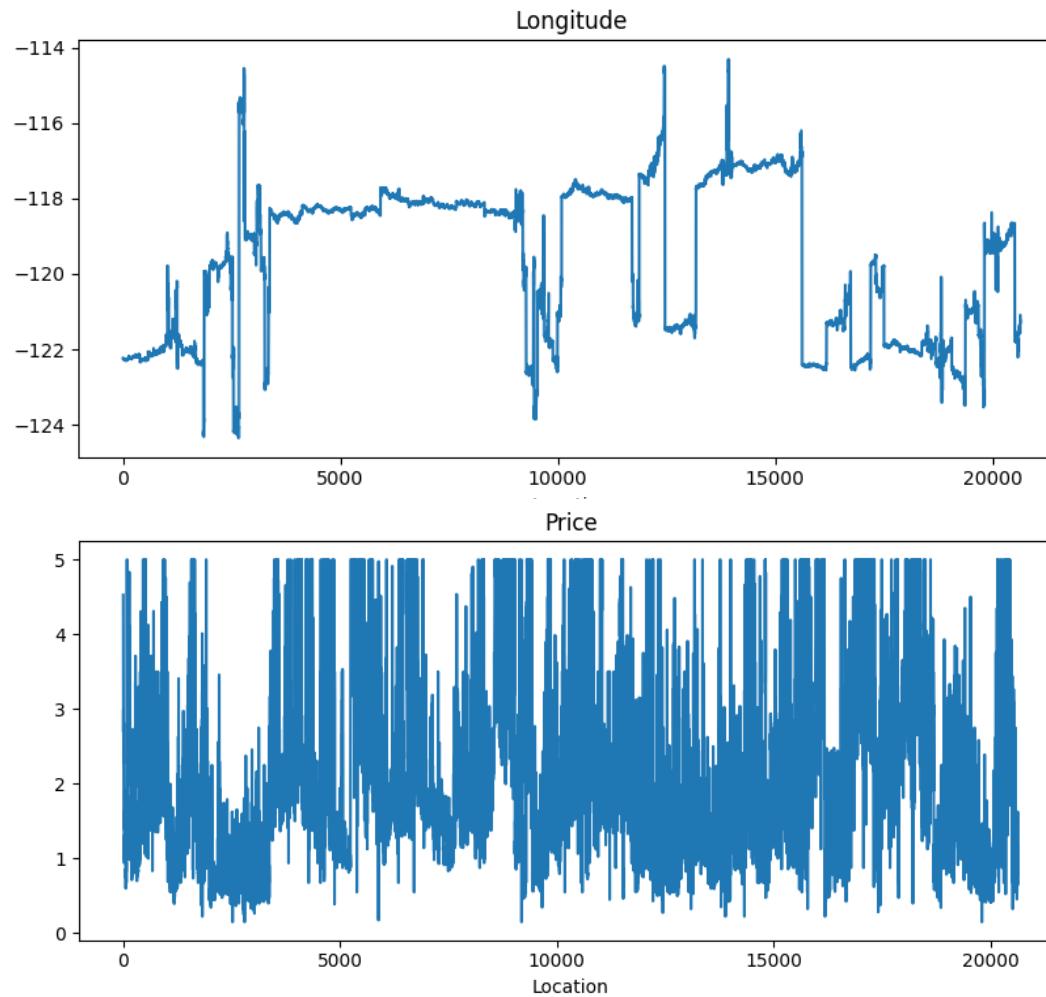
`plt.xlabel('Location')`: x축의 레이블을 설정

`plt.tight_layout()`: 그림의 여백을 자동으로 조정하여 그래프 요소들이 겹치지 않도록

.

2) ML 주요 이론(Supervised Learning)

California House Price Prediction



2) ML 주요 이론(Supervised Learning)

California House Price Prediction

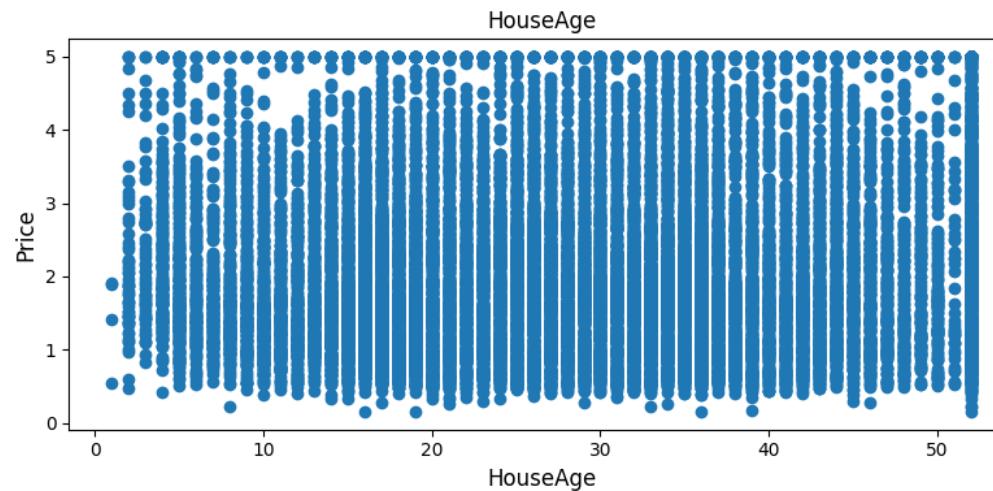
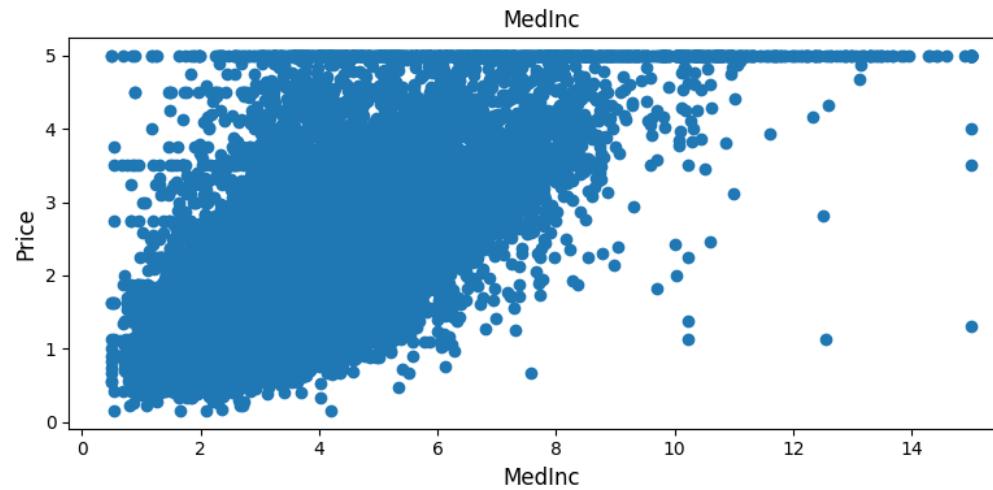
```
1 for i, col in enumerate(housing_df.columns):
2     plt.figure(figsize = (8,4))
3     plt.scatter(housing_df[col], housing_df['Price']) # 집값에 대한 열 데이터의 산포도
4     plt.title(col)
5     plt.ylabel('Price', size =12)
6     plt.xlabel(col, size =12)
7     plt.tight_layout()
```

각 특성과 집값 간의 관계를 시각화한 산포도 그래프가 생성.

각 특성과 집값 간의 선형 또는 비선형 관계를 시각적으로 출력/분석/이해

2) ML 주요 이론(Supervised Learning)

California House Price Prediction



2) ML 주요 이론(Supervised Learning)

California House Price Prediction

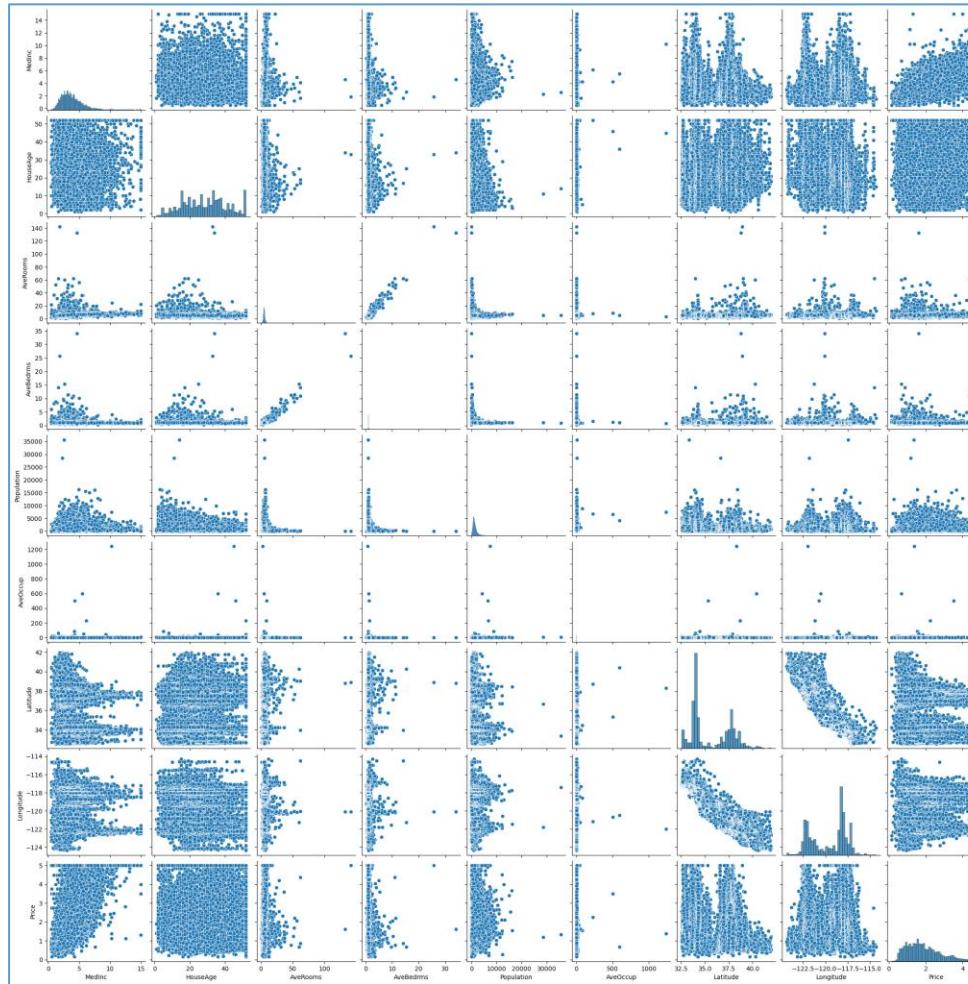
```
1 import seaborn as sns  
2  
3 sns.pairplot(housing_df);
```

housing_df의 모든 특성들 간의 쌍(pair) 관계를 시각화하는 과정.

pairplot 함수는 데이터프레임의 열들을 조합하여 쌍(pair)을 만들고,
각 쌍의 관계를 산점도(scatter plot)로 나타내어
열들 간의 상관 관계, 분포 등을 한눈에 파악할 수 있다.

2) ML 주요 이론(Supervised Learning)

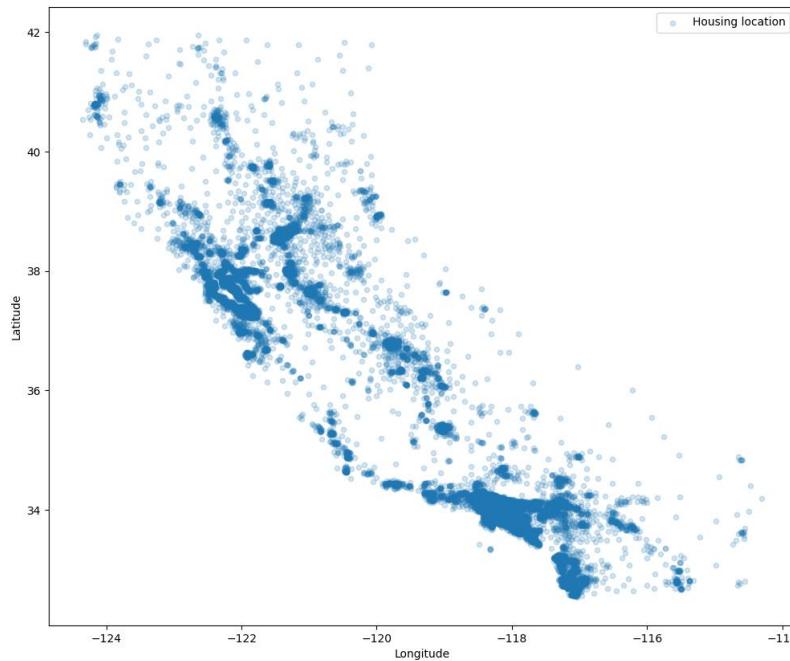
California House Price Prediction



2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 housing_df.plot(kind='scatter', x='Longitude', y='Latitude', alpha=0.2, figsize=(12,10));  
2 plt.legend(['Housing location']) # 위도 경도에 따른 집의 밀집도
```



위도, 경도에 따른 집의 밀집도를 시각화 출력/분석/이해

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 housing_df.plot(kind='scatter', x='Longitude', y='Latitude', alpha=0.2, figsize=(12,10));
2 plt.legend(['Housing location']) # 위도 경도에 따른 집의 밀집도
```

위도와 경도에 따른 집의 밀집도를 시각화.

plot 함수를 사용하여 산점도

kind='scatter'로 지정하여 산점도

x='Longitude'과 y='Latitude'로 x축과 y축에 위도와 경도를 설정.

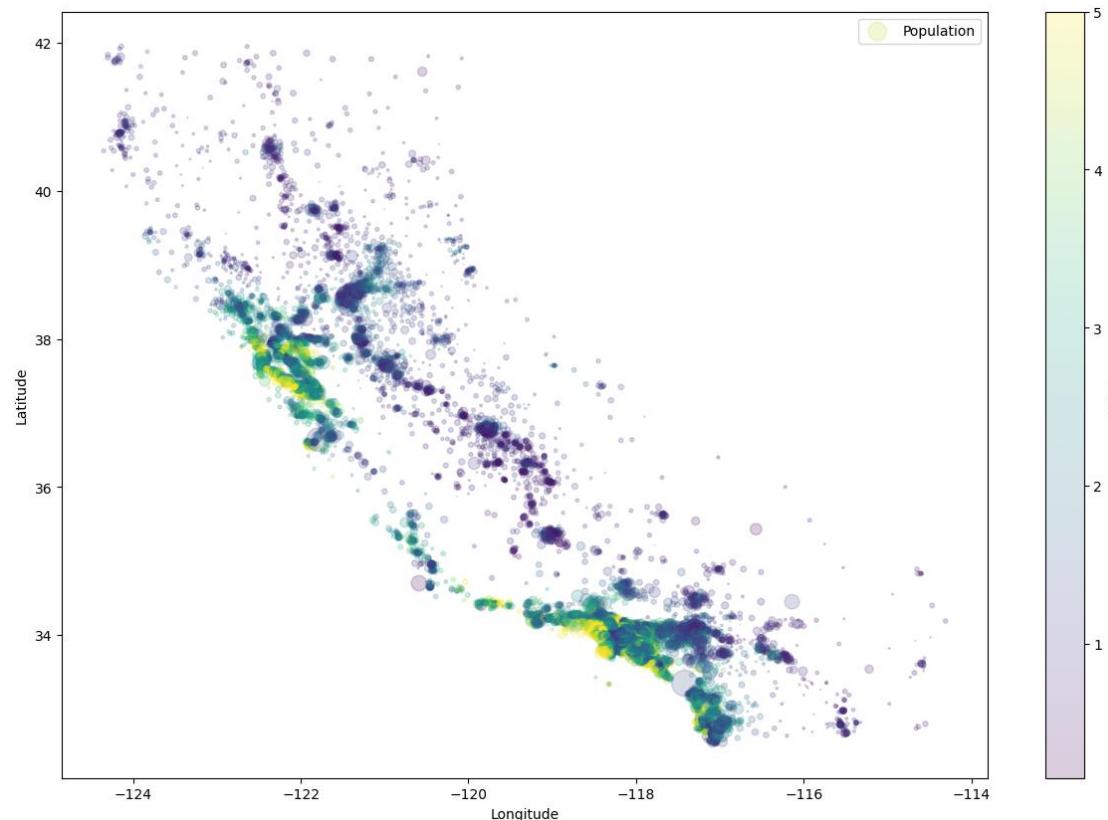
alpha=0.2는 점의 투명도

figsize=(12,10)은 그림의 크기를 지정.

plt.legend(['Housing location']) 그래프에 범례를 추가

2) ML 주요 이론(Supervised Learning)

```
1 housing_df.plot(kind='scatter', x='Longitude', y='Latitude', alpha=0.2,
2                     s=housing_df['Population']/100, label ='Population', figsize=(15,10),
3                     c='Price', cmap=plt.get_cmap('viridis'), colorbar=True);
```



인구의 분포도, 집의 밀집도와 **집값을** 비교, 출력/분석/이해 .

2) ML 주요 이론(Supervised Learning)

```
1 housing_df.plot(kind='scatter', x='Longitude', y='Latitude', alpha=0.2,
2                   s=housing_df['Population']/100, label ='Population', figsize=(15,10),
3                   c='Price', cmap=plt.get_cmap('viridis'), colorbar=True);
```

위도와 경도에 따른 집의 위치를 시각화하고,
점의 크기는 인구 밀도를 나타내며, 색상은 집값을 나타낸다(밝은 색일수록 집값이 높은 지역 !!)

s=housing_df['Population']/100

점의 크기를 조정하는 매개변수로,

housing_df의 'Population' 열 값을 100으로 나누어 크기를 조정.

인구 밀도에 따라 점의 크기가 변동.

label='Population'

범례에 표시될 라벨을 설정. 'Population'이라는 텍스트가 범례에 표시.

figsize=(15,10)은 그림의 크기를 지정.

c='Price'는 색상을 지정하는 매개변수로, 집값에 따라 점의 색상이 달라진다.

cmap=plt.get_cmap('viridis')는 색상 맵을 설정. 'viridis'는 인기있는 색상 맵 중 하나.

colorbar=True는 컬러바를 표시하는지 여부를 설정

2) ML 주요 이론(Supervised Learning)

실습) 데이터의 출력을 바탕으로 내용을 이해/분석 후 리포트를 작성하여 공유해주세요

캘리포니아 주택 가격 예측 분석 보고서

1. 서론

본 보고서는 캘리포니아 주택 가격 데이터셋에 대한 분석 결과를 제시하고, 초기 분석에서 수행된 선형 회귀 모델의 성능을 평가하며, 향후 모델 개선을 위한 잠재적인 전략을 제시하는 것을 목표로 한다. 캘리포니아 주택 시장은 경제적 영향, 정책 결정, 그리고 주택 구매자, 판매자, 투자자, 정책 입안자 등 다양한 이해관계자에게 미치는 영향으로 인해 주택 가격 예측의 중요성이 매우 크다. 본 분석은 캘리포니아 주택 가격 데이터셋에 대한 심층적인 이해를 제공하고, 주택 가격 예측 모델 개발의 기초를 마련하는 데 기여할 것이다. 보고서는 표준 데이터 분석 보고서 형식을 따르며, 데이터 설명, 탐색적 데이터 분석, 모델 구현 및 평가, 결과 해석, 논의 및 권장 사항, 그리고 결론으로 구성된다.¹

2. 데이터 설명

본 분석에 사용된 캘리포니아 주택 가격 데이터셋은 총 20,640개의 샘플로 구성되어 있으며, 각 샘플은 8개의 특징(feature)과 1개의 목표 변수(target variable)를 포함한다. 목표 변수인 MedHouseVal은 각 지역의 중위 주택 가격을 나타내며, 단위는 100,000 USD이다. 데이터셋에 포함된 8개의 특징은 초기 분석에서 구체적으로 언급되지 않았으나, 일반적으로 주택 가격 예측에 사용되는 특징으로는 위치 정보(예: 위도, 경도), 주택 관련 정보(예: 평균 방 개수, 평균 침실 개수, 주택 연식), 지역 사회 관련 정보(예: 중위 소득, 인구), 그리고 기타 관련 정보 등이 포함될 수 있다. 이러한 특징들은 주택 가격에 영향을 미치는 다양한 요인들을 포괄적으로 반영하기 위해 수집되었을 것으로 추정된다. 데이터의 출처 및 수집 방법은 명시되지 않았으나, 일반적으로 공공 데이터셋 또는 부동산 관련 플랫폼에서 수집될 수 있다.¹

California House Price Prediction(LinearRegression)

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.linear_model import LinearRegression  
2 model = LinearRegression()
```

```
1 import sklearn  
2  
3 print(sklearn.__version__)  
4
```

1.2.2

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.3)
```

모델 객체의 생성

학습용 데이터와 테스트용 데이터를 7:3으로 분리

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 model.fit(X_train, y_train)  
2 model.score(X_test, y_test)
```

```
0.6094167774061225
```

```
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))  
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))  
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.6046606132461587  
test Data evaluation : 0.6094167774061225
```

모델객체 생성 후 학습용 데이터로 학습(model.fit)

학습용 데이터로 측정한 모델의 성능과
테스트용 데이터로 측정한 모델의 성능을 출력(model.score())

Model.score() 는 결정계수를 리턴한다.

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.model_selection import cross_val_score  
2 scores = cross_val_score(model, housing.data, housing.target, cv=10, scoring='r2')  
3 print('결정계수 = {}'.format(scores))  
4 # 결정계수는 1에 가까울수록 모델을 잘 설명한다. 다중선형 분석이라 다수의 결정계수
```

```
결정계수 = [0.48254494 0.61416063 0.42274892 0.48178521 0.55705986 0.5412919  
0.47496038 0.45844938 0.48177943 0.59528796]
```

`cross_val_score ()`

지정된 모델과 데이터에 대해 교차 검증을 수행하고,
각 폴드에 대한 성능 평가 결과를 반환.

반환된 결과는 배열 형태로, 각 폴드에서의 성능 평가 지표 값들로 구성.

`model`: 평가할 모델 객체.

`housing.data`: 특성 데이터(독립 변수).

`housing.target`: 타겟 데이터(종속 변수).

`cv`: 교차 검증을 위해 데이터를 분할하는 방식. 기본값은 5,

`scoring`: 성능 평가 지표를 지정. 기본값은 `None`, '`r2`'를 지정하면 결정 계수

2) ML 주요 이론(Supervised Learning)

```
1 print('y = ' + str(model.intercept_) + ' ')
2
3 for i, c in enumerate(model.coef_):
4     print(str(c) + ' * X ' + str(i))
```

```
y = -37.267155516511664
0.43497753118309457 * X 0
0.009138816408914577 * X 1
-0.11094568873033052 * X 2
0.6408509767384429 * X 3
-5.364279376163586e-06 * X 4
-0.003799831827886535 * X 5
-0.42620137366796007 * X 6
-0.4390551929808685 * X 7
```

8개 특성값에 대한 각각의 계수를 출력해서 공식을 만들어 본다.

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.metrics import mean_squared_error, r2_score  
2  
3 y_train_predict = model.predict(X_train)  
4 rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))  
5 r2 = r2_score(y_train, y_train_predict)  
6  
7 print('rmse : {}'.format(rmse))  
8 print('R2 score: {}'.format(r2))
```

```
rmse : 0.7256154669550676  
R2 score: 0.6046606132461587
```

학습용 데이터 X_train으로 예측 데이터를 만들어 RMSE와 r2를 측정해 본다.

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.metrics import mean_squared_error, r2_score  
2  
3 y_test_predict = model.predict(X_test)  
4 rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))  
5 r2 = r2_score(y_test, y_test_predict)  
6  
7 print('rmse : {}'.format(rmse))  
8 print('R2 score: {}'.format(r2))
```

```
rmse : 0.7209360758675682  
R2 score: 0.6094167774061225
```

테스트용 데이터 X_test로 예측 데이터를 만들어 RMSE와 r2를 측정해 본다.

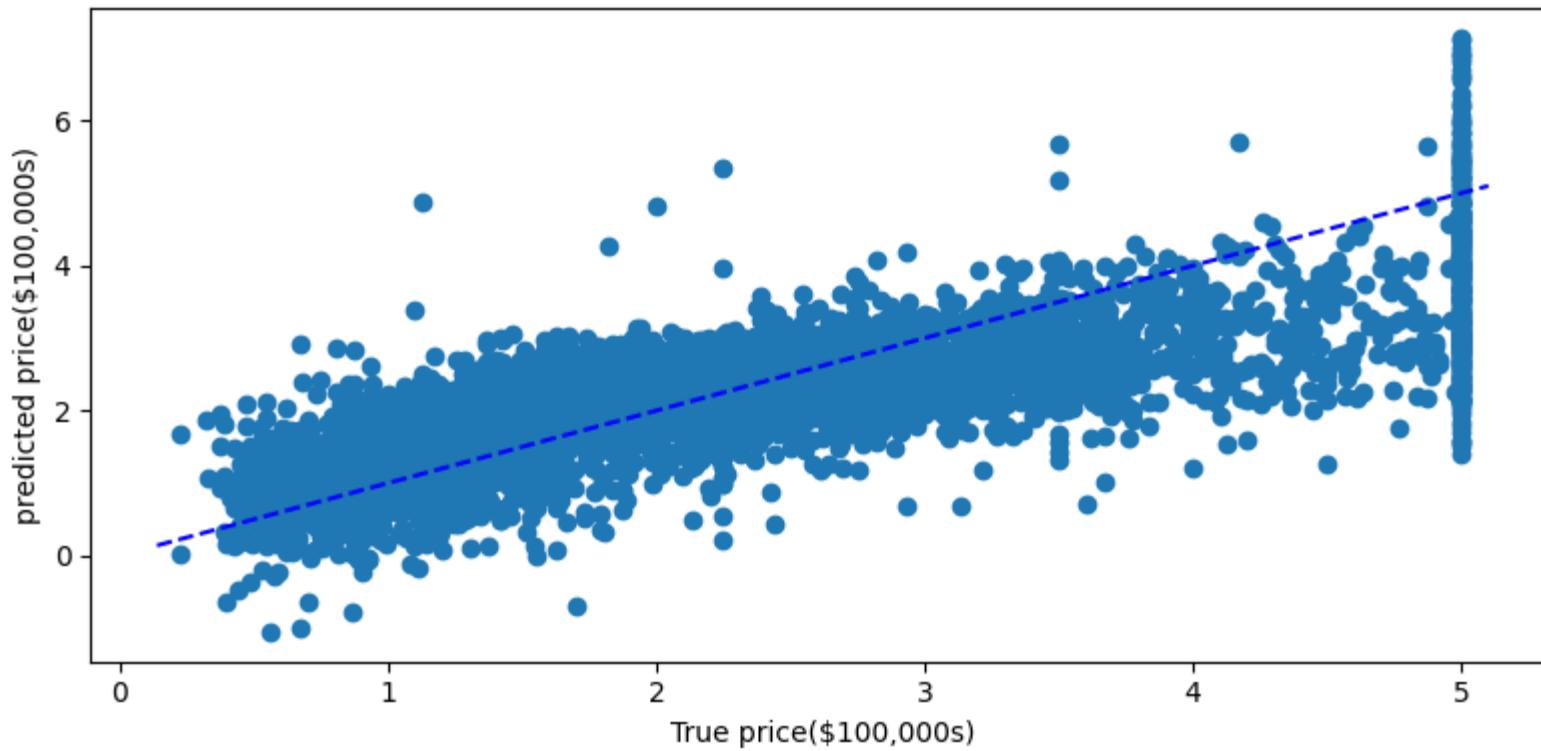
2) ML 주요 이론(Supervised Learning)

```
1 def plot_housing_prices(expected, predicted):
2     plt.figure(figsize =(8,4))
3     plt.scatter(expected, predicted)
4     plt.plot([0.14,5.1], [0.14,5.1], '--b')
5     plt.xlabel('True price($100,000s)')
6     plt.ylabel('predicted price($100,000s)')
7     plt.tight_layout()
8
9 predicted = model.predict(X_test)
10 expected = y_test
11
12 plot_housing_prices(expected, predicted)
13 print(expected, predicted)
```

```
[5.000001 1.594  0.901   ... 3.259  2.103  4.2     ] [3.70086266 1.93577971 1.54880425 ...]
```

실제 데이터(expected)와 예측 데이터를 산점도로 그리고
(plot_housing_prices) 실선 그래프로 그려본다.

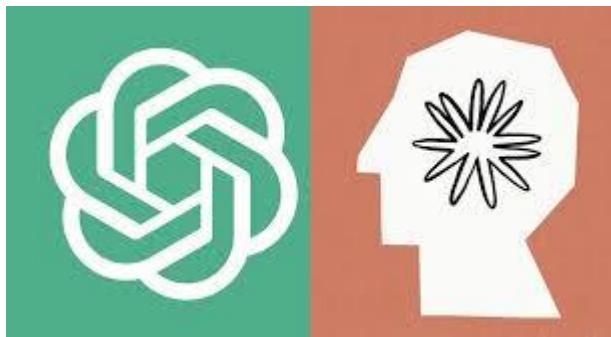
2) ML 주요 이론(Supervised Learning)



(0.14, 5.1)은 price데이터의 min, max자료를 포괄하며
실제 데이터와 예측 데이터가 정확히 일치하는 것을 가정한 선 그래프
이 그래프를 기준으로 예측치와의 차이를 살펴 볼 수 있다.

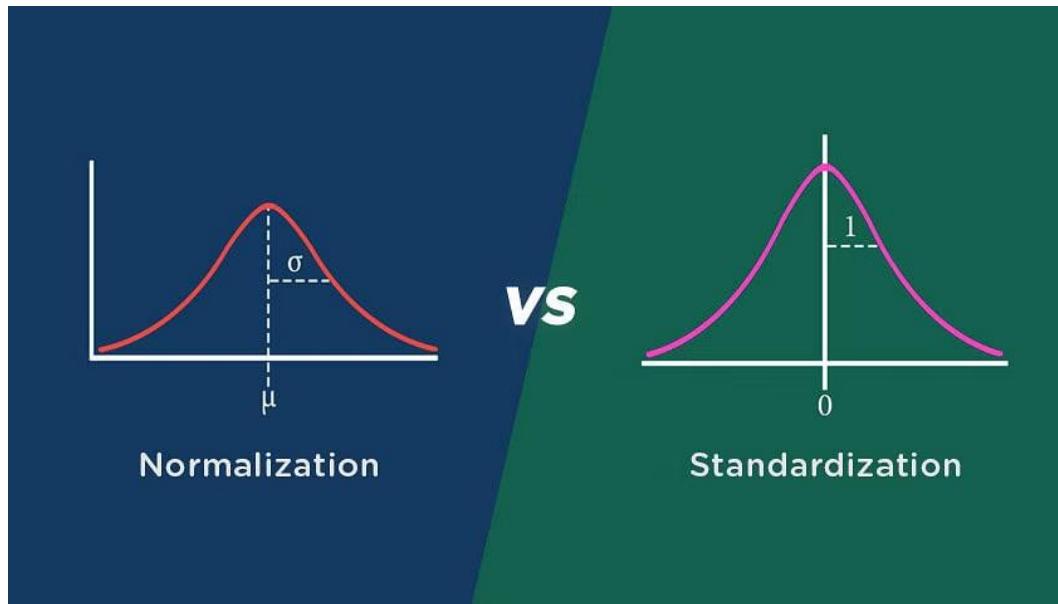
2) ML 주요 이론(Supervised Learning)

실습 1) 학습한 내용을 정리하고 유사한 코드를 생성하여 내용을 정리하고 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

California House Price Prediction
특성 데이터의 전처리
(정규화/표준화)



California House Price Prediction(Standardize/Normalize)

2) ML 주요 이론(Supervised Learning)

정규화(Normalization)

데이터의 상대적인 크기의 조정을 위해 사용.

데이터의 값 범위를 0과 1 사이로 조정하여 상대적인 위치를 파악
최소값과 최대값을 사용하여 계산.

정규화는 데이터의 값 범위가 제한되어 있을 때 유용.

표준화(Standardization)

데이터의 분포를 중심으로 모양을 유지하면서 스케일을 조정

평균이 0이고 표준편차가 1인 분포로 변환하여 데이터의 스케일을 조정

표준화는 데이터의 분포가 정규분포를 따르지 않을 때 유용하며, 이상치에 영향을 받지 않는다.

California House Price Prediction(Standardize/Normalize)

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.linear_model import LinearRegression  
2 model = LinearRegression()  
3  
4 from sklearn.model_selection import train_test_split  
5 from sklearn.preprocessing import StandardScaler  
6  
7 X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.3)
```

전처리 관련 라이브러리를 설정(standardscaler)

California House Price Prediction(Standardize/Normalize)

2) ML 주요 이론(Supervised Learning)

```
1 import pandas as pd  
2  
3 X_test_df = pd.DataFrame(X_test)  
4 pd.set_option('display.float_format', '{:.2f}'.format) # 소수점 둘째 자리까지 표시  
5 X_test_df.describe()
```

	0	1	2	3	4	5	6	7	▶
count	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00	
mean	3.87	28.51	5.46	1.10	1442.32	2.93	35.66	-119.57	
std	1.91	12.65	3.06	0.63	1121.87	0.90	2.16	2.01	
min	0.50	1.00	0.89	0.33	13.00	1.07	32.55	-124.35	
25%	2.57	18.00	4.44	1.01	784.00	2.43	33.93	-121.78	
50%	3.53	28.00	5.23	1.05	1171.00	2.82	34.26	-118.50	
75%	4.70	37.00	6.07	1.10	1757.00	3.28	37.73	-118.01	
max	15.00	52.00	141.91	34.07	16122.00	33.95	41.92	-114.31	

주어진 특성 데이터의 단위가 동일하지 않다 !!

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 scaler = StandardScaler()  
2 X_train = scaler.fit_transform(X_train)  
3 X_test = scaler.fit_transform(X_test)
```

데이터의 표준화(Standardization)를 수행

StandardScaler() : 객체를 생성하여 변수 scaler에 할당.

fit_transform() 메서드를 사용하여 훈련 데이터셋인 X_train을 표준화
훈련 데이터셋의 평균과 표준편차를 계산하고, 각 데이터 포인트를 평균으로부터의
거리에 대한 표준편차로 나누어 표준화된 값을 얻는다.

동일한 StandardScaler() 객체를 사용하여 테스트 데이터셋인 X_test도 표준화.
훈련 데이터셋의 평균과 표준편차를 사용하여 테스트 데이터셋을 표준화.

표준화된 데이터를 사용하면 각 특성의 값이 평균 0, 표준편차 1을 가지게 되어서
다른 특성들과 비교하기 쉬워지고, 머신러닝 모델 학습에 도움

2) ML 주요 이론(Supervised Learning)

```
1 import pandas as pd  
2  
3 X_test_df = pd.DataFrame(X_test)  
4 pd.set_option('display.float_format', '{:.2f}'.format) # 소수점 둘째 자리까지 표시  
5 X_test_df.describe()
```

	0	1	2	3	4	5	6	7	8
count	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00	6192.00
mean	0.00	-0.00	0.00	-0.00	0.00	0.00	0.00	-0.00	
std	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	
min	-1.76	-2.18	-1.49	-1.23	-1.27	-2.06	-1.44	-2.39	
25%	-0.68	-0.83	-0.33	-0.16	-0.59	-0.55	-0.80	-1.10	
50%	-0.18	-0.04	-0.08	-0.09	-0.24	-0.12	-0.65	0.53	
75%	0.44	0.67	0.20	-0.01	0.28	0.39	0.96	0.78	
max	5.83	1.86	44.58	52.72	13.09	34.31	2.90	2.62	

평균이 0이고 표준편차가 1인 표준 정규분포로 변형

2) ML 주요 이론(Supervised Learning)

```
[156] 1 model.fit(X_train, y_train)  
2 model.score(X_test, y_test)
```

```
0.6144127170611667
```

```
[157] 1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))  
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))  
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.6072424520097784  
test Data evaluation : 0.6144127170611667
```



Before Standardized :

training Data eval: 0.6047, test Data eval : 0.6094

Before S

표준화 전보다 결정계수가 개선 : 0.6047 > **0.6072, 0.6094** > **0.6144**

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

특성 데이터의 전처리

(상관관계 분석 후 =>
Ridge/Lasso/Elastic...)

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Correlation matrix)

```
1 # 상관 행렬 계산
2 correlation_matrix = pd.DataFrame(np.corrcoef(housing.data.T), columns=housing.feature_names, index=housing.feature_names)
3
4 # 상관 행렬 출력
5 print(correlation_matrix)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	#
MedInc	1.00	-0.12	0.33	-0.06	0.00	0.02	
HouseAge	-0.12	1.00	-0.15	-0.08	-0.30	0.01	
AveRooms	0.33	-0.15	1.00	0.85	-0.07	-0.00	
AveBedrms	-0.06	-0.08	0.85	1.00	-0.07	-0.01	
Population	0.00	-0.30	-0.07	-0.07	1.00	0.07	
AveOccup	0.02	0.01	-0.00	-0.01	0.07	1.00	
Latitude	-0.08	0.01	0.11	0.07	-0.11	0.00	
Longitude	-0.02	-0.11	-0.03	0.01	0.10	0.00	

MedInc(중간 소득)은 AveRooms(가구당 평균 방 개수)와 약한 양의 선형 관계(0.33), AveBedrms(가구당 평균 침실 개수)와는 약한 음의 선형 관계(-0.06).

HouseAge(주택의 중간 연식)은 MedInc와는 약한 음의 선형 관계(-0.12).

Population(인구)과는 약한 음의 선형 관계(-0.30).

AveRooms와 AveBedrms는 강한 양의 선형 관계(0.85).

Latitude(위도)와 Longitude(경도)는 서로 강한 음의 선형 관계(-0.92).

2) ML 주요 이론(Supervised Learning)

```
correlation_matrix = pd.DataFrame(np.corrcoef(housing.data.T),  
                                   columns=housing.feature_names,  
                                   index=housing.feature_names)
```

특성들 간의 상관관계를 계산하여 상관계수(correlation coefficient) 행렬을 생성

np.corrcoef() 함수는 주어진 데이터의 상관계수를 계산하는 함수.

housing.data.T : 특성들을 전치(transpose)하여 각 특성이 열로 표현.

np.corrcoef(housing.data.T) : 전치된 데이터셋의 특성들 간의 상관계수 행렬을 계산.

pd.DataFrame() 함수를 사용하여 계산된 상관계수 행렬을 판다스 데이터프레임으로 변환.

columns=housing.feature_names과 **index=housing.feature_names**는 열과 행의 이름을 주어진 데이터셋의 특성 이름으로 설정.

correlation_matrix는 데이터셋의 특성들 간의 상관계수를 나타내는 행렬로서,
각 특성들 간의 상관관계를 파악

2) ML 주요 이론(Supervised Learning)

상관관계 분석 후 상관관계의 처리방법

전략	설명	적용 모델	사용 도구
1. 상관관계 높은 변수 제거	상관계수 기준(예: 0.9 이상)으로 한 변수를 삭제	모든 모델에 가능	pandas .corr() + .drop()
2. 변수 결합 (파생변수 생성)	더하거나 나누거나 평균 내어 새로운 변수 생성	비선형 모델에 유리	feature engineering
3. 주성분 분석 (PCA)	상관된 변수들을 축소하여 비상관된 성분으로 압축	선형 회귀, SVM 등	sklearn.decomposition.PCA
4. 정규화 기법 (Lasso, Ridge)	L1, L2 정규화로 중요하지 않은 특성의 계수를 감소	선형 계열 모델	Ridge, Lasso
5. 트리 기반 모델 사용	상관관계에 덜 민감한 트리 기반 모델 사용	결정트리, 랜덤포레스트, XGBoost	-
6. 도메인 지식 기반 선택	도메인 지식을 활용해 중요한 변수만 수동 선택	회귀, 분류 전반	-

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

Ridge Regression

Ridge 회귀 : alpha 값

정규화(regularization)의 강도를 조절하는 하이퍼파라미터.

alpha 값은 일반적으로 0부터 무한대까지의 양수 값.

이 값이 클수록 정규화의 강도가 강해지며,
회귀 계수의 크기를 더 많이 축소시킵니다.

일반적으로 alpha 값은 로그 스케일로 지정.

0.1, 1, 10, 100과 같이 10의 거듭제곱 형태로 사용.

작은 alpha 값은 더 약한 정규화를 의미하며, 큰 alpha 값은 더 강한 정규화를 의미.

2) ML 주요 이론(Supervised Learning)

sklearn.linear_model.Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None)
```

[source]

Linear least squares with L2 regularization.

Minimizes the objective function:

Parameters:

alpha : {float, ndarray of shape (n_targets,)}, default=1.0

Constant that multiplies the L2 term, controlling regularization strength. `alpha` must be a non-negative float i.e. in `[0, inf)`.

When `alpha = 0`, the objective is equivalent to ordinary least squares, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Ridge` object is not advised. Instead, you should use the `LinearRegression` object.

If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

fit_intercept : bool, default=True

Whether to fit the intercept for this model. If set to false, no intercept will be used in calculations (i.e. `x` and `y` are expected to be centered).

copy_X : bool, default=True

If True, `X` will be copied; else, it may be overwritten.

max_iter : int, default=None

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Ridge Regression)

```
[223] 1 from sklearn.linear_model import Ridge  
2 import pandas as pd  
3  
4 from sklearn.datasets import fetch_california_housing  
5 housing = fetch_california_housing()  
6  
7 housing_df = pd.DataFrame(housing.data, columns = housing.feature_names)  
8 housing_df['Price'] = housing.target  
9 housing_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price	✎
0	8.33	41.00	6.98	1.02	322.00	2.56	37.88	-122.23	4.53	
1	8.30	21.00	6.24	0.97	2401.00	2.11	37.86	-122.22	3.58	
2	7.26	52.00	8.29	1.07	496.00	2.80	37.85	-122.24	3.52	
3	5.64	52.00	5.82	1.07	558.00	2.55	37.85	-122.25	3.41	
4	3.85	52.00	6.28	1.08	565.00	2.18	37.85	-122.25	3.42	

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Ridge)

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.2)  
  
1 model = Ridge(alpha = 10) # 0.1, 1, 10 정도의 알파값을 추천  
2 model.fit(X_train, y_train)  
  
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))  
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))  
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정  
  
training Data evaluation : 0.611311662304608  
test Data evaluation : 0.5832187763767307
```

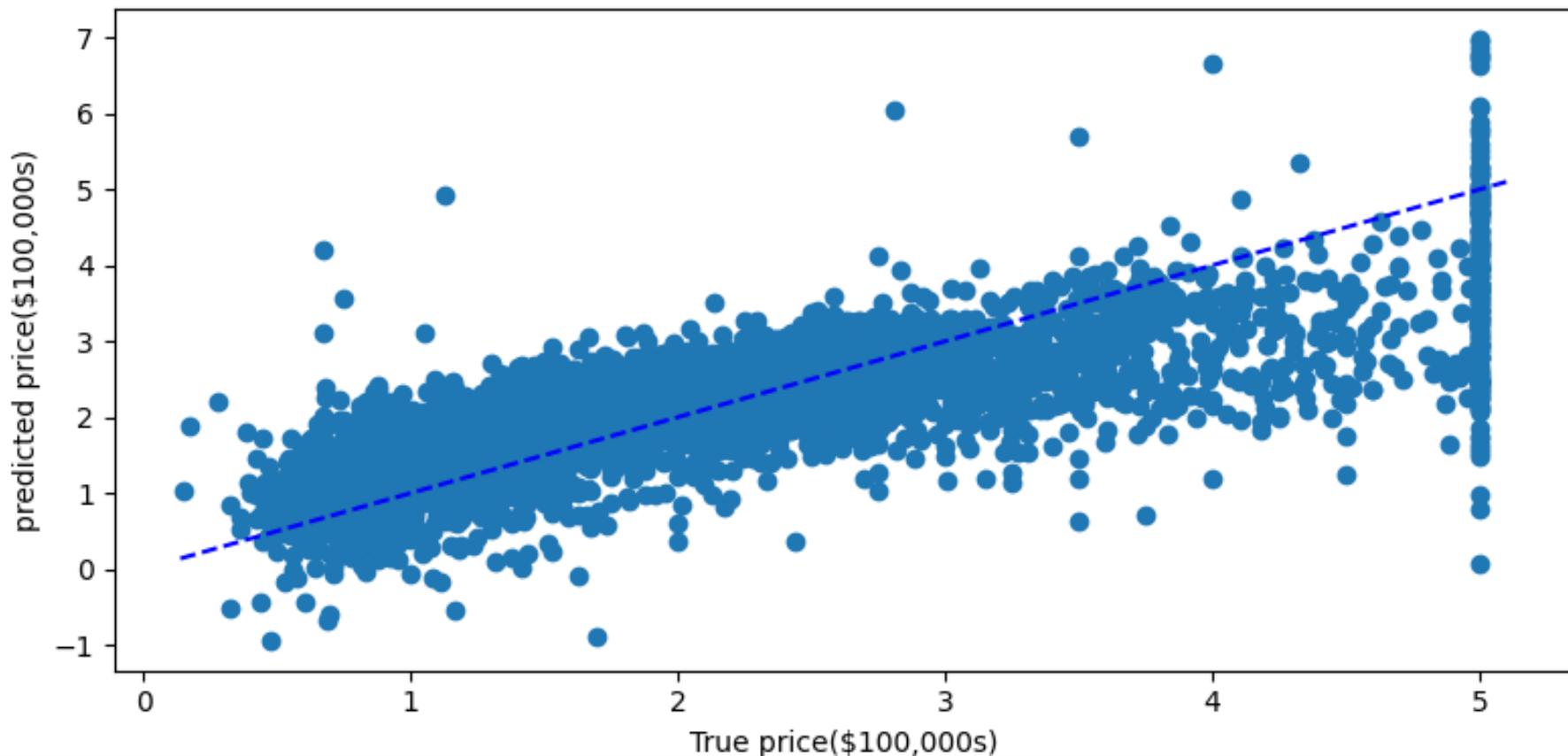
환경을 구성 후 **알파값(규제 강도)**을 조정하며 모델의 학습결과를 평가해 본다.

표준화 전보다 결정계수가 개선 : **0.6047 > 0.6072 > ?, 0.6094 > 0.6144 > ?**

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 predicted = model.predict(X_test)
2 expected = y_test
3 plot_housing_prices(expected, predicted)
```



2) ML 주요 이론(Supervised Learning)

California House Price Prediction
Lasso Regression

2) ML 주요 이론(Supervised Learning)

sklearn.linear_model.Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')
```

[source]

Linear Model trained with L1 prior as regularizer (aka the Lasso).

The optimization objective for Lasso is:

Parameters:	
alpha : float, default=1.0	Constant that multiplies the L1 term, controlling regularization strength. <code>alpha</code> must be a non-negative float i.e. in <code>[0, inf)</code> . When <code>alpha = 0</code> , the objective is equivalent to ordinary least squares, solved by the <code>LinearRegression</code> object. For numerical reasons, using <code>alpha = 0</code> with the <code>Lasso</code> object is not advised. Instead, you should use the <code>LinearRegression</code> object.
fit_intercept : bool, default=True	Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).
precompute : bool or array-like of shape (n_features, n_features), default=False	Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always <code>False</code> to preserve sparsity.
copy_X : bool, default=True	If <code>True</code> , <code>X</code> will be copied; else, it may be overwritten.
max_iter : int, default=1000	The maximum number of iterations.

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Lasso Regression)

```
[254]: 1 from sklearn.linear_model import Lasso  
2 import pandas as pd  
3  
4 from sklearn.datasets import fetch_california_housing  
5 housing = fetch_california_housing()  
6  
7 housing_df = pd.DataFrame(housing.data, columns = housing.feature_names)  
8 housing_df['Price'] = housing.target  
9 housing_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.33	41.00	6.98	1.02	322.00	2.56	37.88	-122.23	4.53
1	8.30	21.00	6.24	0.97	2401.00	2.11	37.86	-122.22	3.58
2	7.26	52.00	8.29	1.07	496.00	2.80	37.85	-122.24	3.52
3	5.64	52.00	5.82	1.07	558.00	2.55	37.85	-122.25	3.41
4	3.85	52.00	6.28	1.08	565.00	2.18	37.85	-122.25	3.42

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Lasso)

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.2)  
  
1 model = Lasso(alpha = 0.001) # 0.1, 1, 10 정도의 알파값을 추천  
2 model.fit(X_train, y_train)  
  
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))  
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))  
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정  
  
training Data evaluation : 0.604786231219316  
test Data evaluation : 0.6030881605323302
```

환경을 구성 후 알파값을 조정하며 모델의 학습결과를 평가해 본다.

표준화 전보다 결정계수가 개선 : **0.6047 > 0.6072 > ?, 0.6094 > 0.6144 > ?**

2) ML 주요 이론(Supervised Learning)

California House Price Prediction
Elastic Regression

1) Machine Learning 개요

sklearn.linear_model.ElasticNet

```
class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, precompute=False, max_iter=1000,  
copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')  
[source]
```

Linear regression with combined L1 and L2 priors as regularizer.

Minimizes the objective function:

Parameters:	alpha : float, default=1.0 Constant that multiplies the penalty terms. Defaults to 1.0. See the notes for the exact mathematical meaning of this parameter. <code>alpha = 0</code> is equivalent to an ordinary least square, solved by the <code>LinearRegression</code> object. For numerical reasons, using <code>alpha = 0</code> with the <code>Lasso</code> object is not advised. Given this, you should use the <code>LinearRegression</code> object. l1_ratio : float, default=0.5 The ElasticNet mixing parameter, with <code>0 <= l1_ratio <= 1</code> . For <code>l1_ratio = 0</code> the penalty is an L2 penalty. For <code>l1_ratio = 1</code> it is an L1 penalty. For <code>0 < l1_ratio < 1</code> , the penalty is a combination of L1 and L2. fit_intercept : bool, default=True Whether the intercept should be estimated or not. If <code>False</code> , the data is assumed to be already centered. precompute : bool or array-like of shape (n_features, n_features), default=False Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always <code>False</code> to preserve sparsity. max_iter : int, default=1000 The maximum number of iterations.
--------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Elastic Net Regression)

```
1 from sklearn.linear_model import ElasticNet  
2 from sklearn.datasets import fetch_california_housing  
3 housing = fetch_california_housing()  
4  
5 housing_df = pd.DataFrame(housing.data, columns = housing.feature_names)  
6 housing_df['Price'] = housing.target  
7 housing_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	8.33	41.00	6.98	1.02	322.00	2.56	37.88	-122.23	4.53
1	8.30	21.00	6.24	0.97	2401.00	2.11	37.86	-122.22	3.58
2	7.26	52.00	8.29	1.07	496.00	2.80	37.85	-122.24	3.52
3	5.64	52.00	5.82	1.07	558.00	2.55	37.85	-122.25	3.41
4	3.85	52.00	6.28	1.08	565.00	2.18	37.85	-122.25	3.42

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.2)
```

```
1 model = ElasticNet(alpha = 0.005, l1_ratio=0.05) # L1, L2의 가중치 설정  
2 model.fit(X_train, y_train)
```

ElasticNet

```
ElasticNet(alpha=0.005, l1_ratio=0.05)
```

```
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))  
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))  
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.6022981447564342  
test Data evaluation : 0.6193282036869008
```

환경을 구성 후 알파값, L1/L2 값을 조정하며 모델의 학습결과를 평가해

표준화 전보다 결정계수가 개선 : **0.6047 > 0.6072 > ?, 0.6094 > 0.6144 > ?**

2) ML 주요 이론(Supervised Learning)

California House Price Prediction
Polynomial Regression

2) ML 주요 이론(Supervised Learning)

sklearn.preprocessing.PolynomialFeatures

```
class sklearn.preprocessing.PolynomialFeatures(degree=2, *, interaction_only=False, include_bias=True, order='C') [source]
```

Generate polynomial and interaction features.

Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].

Parameters:

degree : int or tuple (min_degree, max_degree), default=2

If a single int is given, it specifies the maximal degree of the polynomial features. If a tuple (min_degree, max_degree) is passed, then min_degree is the minimum and max_degree is the maximum polynomial degree of the generated features. Note that min_degree=0 and min_degree=1 are equivalent as outputting the degree zero term is determined by include_bias.

interaction_only : bool, default=False

If True, only interaction features are produced: features that are products of at most degree distinct input features, i.e. terms with power of 2 or higher of the same input feature are excluded:

- included: x[0], x[1], x[0] * x[1], etc.
- excluded: x[0] ** 2, x[0] ** 2 * x[1], etc.

include_bias : bool, default=True

If True (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).

2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Polynomial Regression)

```
1 from sklearn.linear_model import LinearRegression  
2 from sklearn.preprocessing import PolynomialFeatures, StandardScaler  
3 from sklearn.pipeline import make_pipeline  
4 from sklearn.datasets import fetch_california_housing  
5  
6 housing = fetch_california_housing()  
7  
8 housing_df = pd.DataFrame(housing.data, columns = housing.feature_names)  
9 housing_df['Price'] = housing.target  
10 housing_df.head()
```

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.2)  
  
1 model = make_pipeline(PolynomialFeatures(degree = 2),  
2                         StandardScaler(),  
3                         LinearRegression())  
4  
5 model.fit(X_train, y_train)  
  
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))  
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))  
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정  
  
training Data evaluation : 0.6793644893454003  
test Data evaluation : 0.6556294473412998
```

환경을 구성 후 polynomial regression 모델의 학습결과를 평가해 본다.

표준화 전보다 결정계수가 개선 : 0.6047 > **0.6072 > ?, 0.6094 > 0.6144 > ?**

2) ML 주요 이론(Supervised Learning)

make_pipeline()

scikit-learn에서 여러 개의 변환기(transformer)와 모델(estimator)을
순차적으로 연결하여 파이프라인(Pipeline) 객체를 간단히 만드는 함수

장점	설명
코드 간결성	명시적 이름 없이 한 줄로 구성 가능
자동 이름 생성	각 스텝에 대해 이름을 자동 부여
그리드서치 연동 쉬움	GridSearchCV와 결합 시 "모델_하이퍼파라미터" 식으로 사용

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = make_pipeline(StandardScaler(),
LogisticRegression(C=1.0))

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'logisticregression__C': [0.1, 1, 10]
}

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)
```

2) ML 주요 이론(Supervised Learning)

make_pipeline()으로 만든 파이프라인 객체에서 .fit() 을 호출하면:
StandardScaler)는 fit_transform()을 자동으로 수행하고,
LogisticRegression) fit()만 수행됩니다.

```
from sklearn.pipeline import make_pipeline  
  
pipe = make_pipeline(StandardScaler(), LogisticRegression())  
pipe.fit(X_train, y_train)
```

2) ML 주요 이론(Supervised Learning)

StandardScaler

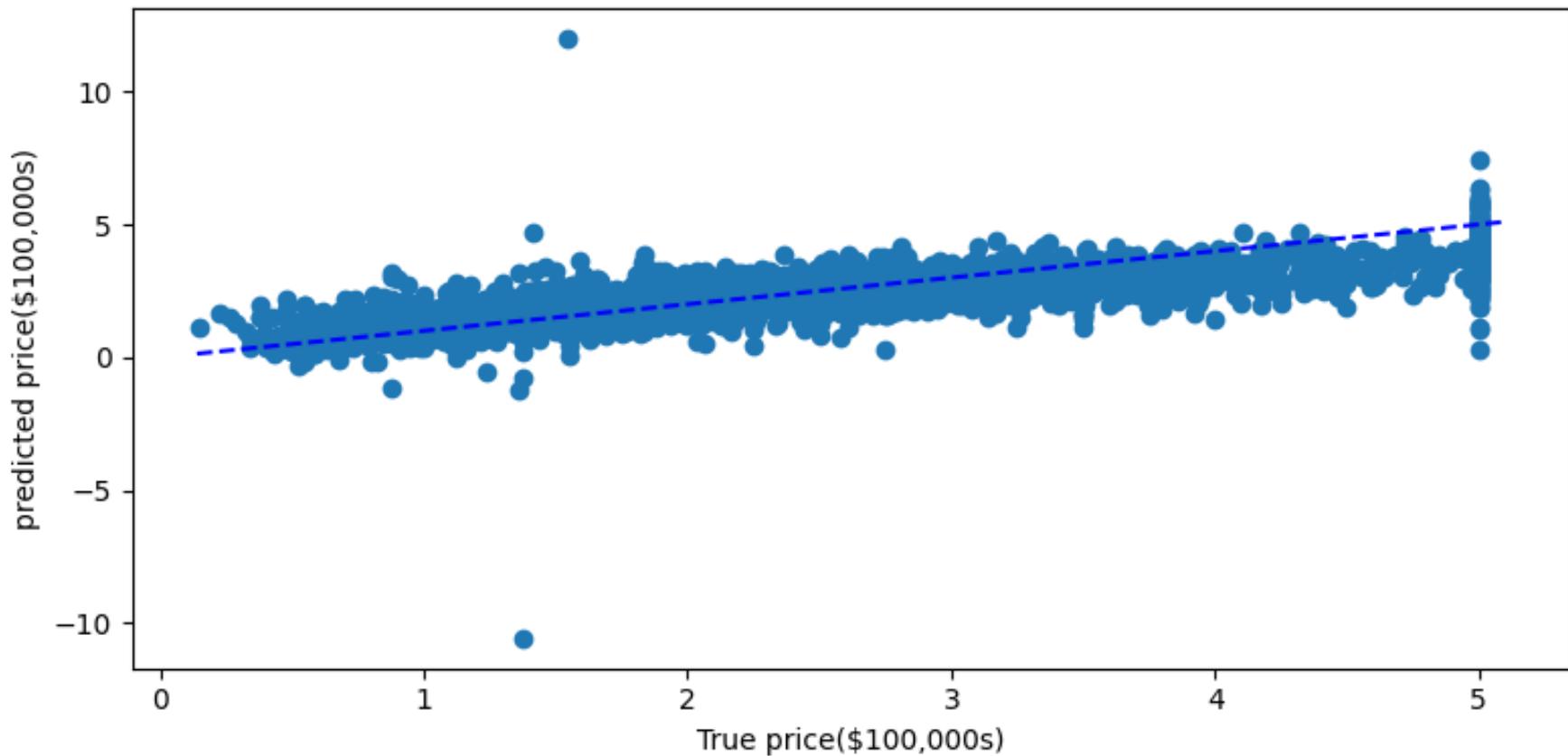
fit_transform: 데이터에 대해 표준화 파라미터(평균과 표준편차)를 학습(fit)하고, 이를 사용해 데이터를 변환(transform)
훈련 데이터를 처음 표준화할 때 사용.
데이터를 기반으로 평균과 표준편차를 계산한 뒤, 이를 적용하여 데이터를 표준화.

transform: 이미 학습된(fit으로 계산된) 평균과 표준편차를 사용해 데이터를
변환(transform)만 합니다. 새로운 파라미터를 학습하지 않음.

2) ML 주요 이론(Supervised Learning)

California House Price Prediction

```
1 predicted = model.predict(X_test)
2 expected = y_test
3 plot_housing_prices(expected, predicted)
```



2) ML 주요 이론(Supervised Learning)

California House Price Prediction(Kaggle data 심화학습)

The screenshot shows a Kaggle notebook interface. On the left, there's a sidebar with navigation links: 'kaggle' (with a menu icon), 'Create', 'Home', 'Competitions', 'Datasets', 'Models', and 'Code'. The main area displays a notebook titled 'California Housing (EDA + linear regression)' by 'MOHAMED KHALED ELSAFTY - 7MO AGO - 1,272 VIEWS'. The notebook has 58 upvotes and 8 comments. Below the title, it says 'Python · California Housing Prices'. At the bottom, there are tabs for 'Notebook' (which is selected), 'Input', 'Output', 'Logs', and 'Comments (37)'. A search bar at the top right contains the placeholder 'Search'.

2) ML 주요 이론(Supervised Learning)

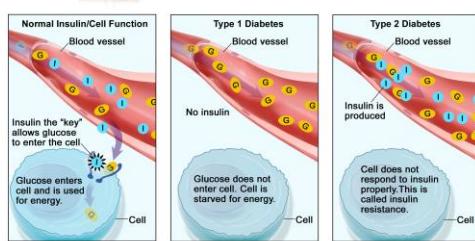
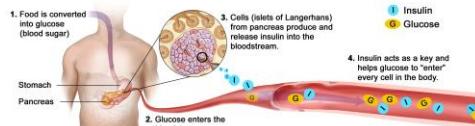
Diabetes Dataset

포도당(글루코스) 역할:

포도당은 몸의 주요 에너지원입니다. 우리가 섭취한 음식에서 포도당이 분해되어 혈액으로 흡수됩니다. 혈당: 혈액 내의 포도당 농도는 혈당으로 불립니다. 정상적인 혈당 수치는 우리 몸의 에너지를 공급하고, 세포 기능을 유지하는 데 중요합니다.

인슐린 역할:

인슐린은 췌장에서 분비되는 호르몬입니다. 인슐린은 혈당을 세포로 이동시키는 데 중요한 역할을 합니다. 세포는 이 포도당을 에너지원으로 사용합니다. 작용: 식사 후 혈당이 상승하면 췌장은 인슐린을 분비하여 혈당을 세포로 이동시켜 혈당을 낮춥니다.



2) ML 주요 이론(Supervised Learning)

Diabetes Dataset

당뇨병 환자들의 특성과 혈액 검사 결과를 포함

당뇨병 예측 모델링 및 당뇨병 관련 특성과의 상관 관계 분석 등

- age: 환자의 나이
- sex: 환자의 성별
- bmi: 체질량 지수 (체중과 키에 기반한 지수), 비만도, 18.5에서 24.9 사이인 경우 정상
- bp: 평균 혈압 (혈압 측정 값의 평균), 혈압이 높을수록 당뇨병 발병 가능성이 높다
 정상 혈압(수축기 90에서 120 mmHg, 이완기 60에서 80 mmHg)

- s1: tc (total serum cholesterol), 총 혈청 콜레스테롤 수치
- s2: ldl (low-density lipoproteins), 저밀도 지단백질 수치
- s3: hdl (high-density lipoproteins), 고밀도 지단백질 수치
- s4: tch (total cholesterol / HDL), 총 콜레스테롤 수치 / 고밀도 지단백질 수치
 총 콜레스테롤은 200 mg/dL 이하, LDL 100 mg/dL 이하, HDL 40 mg/dL 이상 정상 범위

- s5: ltg (possibly log of serum triglycerides level), 혈청 수치에 대한 로그값, 150 mg/dL 미만이 정상
- s6: glu (blood sugar level), 혈당 수치, 공복 시 70에서 100 mg/dL 정상

2) ML 주요 이론(Supervised Learning)

Diabetes Dataset

1. 공복 혈당 (Fasting Blood Glucose, FBG)

측정 방법: 8시간 이상 공복 후에 측정합니다.

정상 수치: 70-99 mg/dL (3.9-5.5 mmol/L)

당뇨 전 단계: 100-125 mg/dL (5.6-6.9 mmol/L)

당뇨병 진단: 126 mg/dL (7.0 mmol/L) 이상

2. 식후 2시간 혈당 (Postprandial Blood Glucose, PPG)

측정 방법: 식사 후 2시간 후에 측정합니다.

정상 수치: 140 mg/dL (7.8 mmol/L) 미만

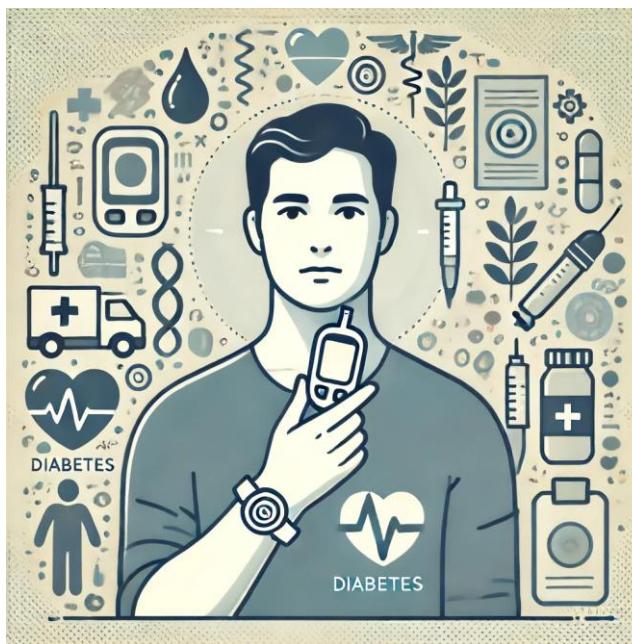
당뇨 전 단계: 140-199 mg/dL (7.8-11.0 mmol/L)

당뇨병 진단: 200 mg/dL (11.1 mmol/L) 이상



2) ML 주요 이론(Supervised Learning)

실습) Diabetes Dataset의 자료를 참조하여 관련 내용을 정리
후 공유해 주세요



2) ML 주요 이론(Supervised Learning)

Diabetes Dataset

```
1 import numpy as np  
2 import pandas as pd  
3 import matplotlib.pyplot as plt  
4 from sklearn.datasets import load_diabetes  
  
1 diabetes=load_diabetes()  
2 Dia_df=pd.DataFrame(diabetes.data, columns=diabetes.feature_names)  
3 Dia_df['degree']=diabetes.target  
4 Dia_df.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	degree
0	0.04	0.05	0.06	0.02	-0.04	-0.03	-0.04	-0.00	0.02	-0.02	151.00
1	-0.00	-0.04	-0.05	-0.03	-0.01	-0.02	0.07	-0.04	-0.07	-0.09	75.00
2	0.09	0.05	0.04	-0.01	-0.05	-0.03	-0.03	-0.00	0.00	-0.03	141.00
3	-0.09	-0.04	-0.01	-0.04	0.01	0.02	-0.04	0.03	0.02	-0.01	206.00
4	0.01	-0.04	-0.04	0.02	0.00	0.02	0.01	-0.00	-0.03	-0.05	135.00

2) ML 주요 이론(Supervised Learning)

```
1 print(diabetes.DESCR)

.. _diabetes_dataset:

Diabetes dataset
-----
Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:
 - age      age in years
 - sex
 - bmi     body mass index
 - bp      average blood pressure
 - s1      tc, total serum cholesterol
 - s2      ldl, low-density lipoproteins
 - s3      hdl, high-density lipoproteins
 - s4      tch, total cholesterol / HDL
 - s5      ltg, possibly log of serum triglycerides level
 - s6      glu, blood sugar level
```

Target:

Column 11 is
a quantitative measure of
disease progression one year
after baseline

10 features

mean centered and
scaled by the standard deviation

2) ML 주요 이론(Supervised Learning)

Diabetes Dataset(Ridge/Lasso Regression)

```
1 from sklearn.model_selection import train_test_split  
2 X_train,X_test,y_train, y_test=train_test_split(diabetes.data, diabetes.target, test_size=0.3)  
  
1 from sklearn.linear_model import Ridge, Lasso  
2  
3 RidgeRegr=Ridge(alpha=0.2)  
4 RidgeRegr.fit(X_train,y_train)  
5  
6 LassoRegr=Lasso(alpha=0.2)  
7 LassoRegr.fit(X_train,y_train)
```

분석할 데이터에 대한 충분한 이해한 후,

환경을 구성하고 알파값을 조정하며 모델의 학습결과를 평가해 본다.

2) ML 주요 이론(Supervised Learning)

Diabetes Dataset

```
1 print("training Data evaluation : {}".format(RidgeRegre.score(X_train, y_train)))
2 print("test Data evaluation : {}".format(RidgeRegre.score(X_test, y_test)))
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.5413449571499813
test Data evaluation : 0.39705453298113413
```

```
1 print("training Data evaluation : {}".format(LassoRegre.score(X_train, y_train)))
2 print("test Data evaluation : {}".format(LassoRegre.score(X_test, y_test)))
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.5389607341044169
test Data evaluation : 0.35287786926819265
```

환경을 구성 후 알파값을 조정하며 모델의 학습결과를 평가해 본다.
주어진 데이터(학습용/데이터용)에만 과적합되었는지 평가해 본다.

Diabetes Dataset

2) ML 주요 이론(Supervised Learning)

```
1 from sklearn.metrics import mean_squared_error, r2_score  
2  
3 y_test_predict = RidgeRegre.predict(X_test)  
4 rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))  
5 r2 = r2_score(y_test, y_test_predict)  
6  
7 print('Ridge 테스트데이터 예측 rmse : {}'.format(rmse))  
8 print('Ridge 테스트데이터 예측 R2 score: {}'.format(r2)) # 테스트 데이터로 예측한 성능
```

```
Ridge 테스트데이터 예측 rmse : 56.19419383758114  
Ridge 테스트데이터 예측 R2 score: 0.39705453298113413
```

```
1 from sklearn.metrics import mean_squared_error, r2_score  
2  
3 y_test_predict = LassoRegre.predict(X_test)  
4 rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))  
5 r2 = r2_score(y_test, y_test_predict)  
6  
7 print('Lasso 테스트데이터 예측 rmse : {}'.format(rmse))  
8 print('Lasso 테스트데이터 예측 R2 score: {}'.format(r2)) # 테스트 데이터로 예측한 성능
```

```
Lasso 테스트데이터 예측 rmse : 58.2164279143284  
Lasso 테스트데이터 예측 R2 score: 0.35287786926819265
```

예측한 데이터가 갖는 설명력(결정계수) 등 모델의 예측결과를 평가해 본다.

2) ML 주요 이론(Supervised Learning)

Diabetes Dataset(Polynomial Regression)

```
1 from sklearn.model_selection import train_test_split  
2 X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target, test_size=0.3)
```

```
1 from sklearn.pipeline import make_pipeline  
2 from sklearn.preprocessing import PolynomialFeatures, StandardScaler  
3 from sklearn.linear_model import LinearRegression  
4  
5 model = make_pipeline(PolynomialFeatures(degree=2),  
6                         StandardScaler(),  
7                         LinearRegression())  
8 model.fit(X_train, y_train)
```

Polynomial Regression환경을 구성 후 모델의 학습/예측 결과를 평가해 본다.

2) ML 주요 이론(Supervised Learning)

Diabetes Dataset(Polynomial Regression)

```
1 print("training Data evaluation : {}".format(model.score(X_train, y_train)))
2 print("test Data evaluation : {}".format(model.score(X_test, y_test)))
3 # score : 주어진 데이터에 대한 모델의 예측 정확성이나 설명력을 측정
```

```
training Data evaluation : 0.5667782264201318
test Data evaluation : 0.5033169770750128
```

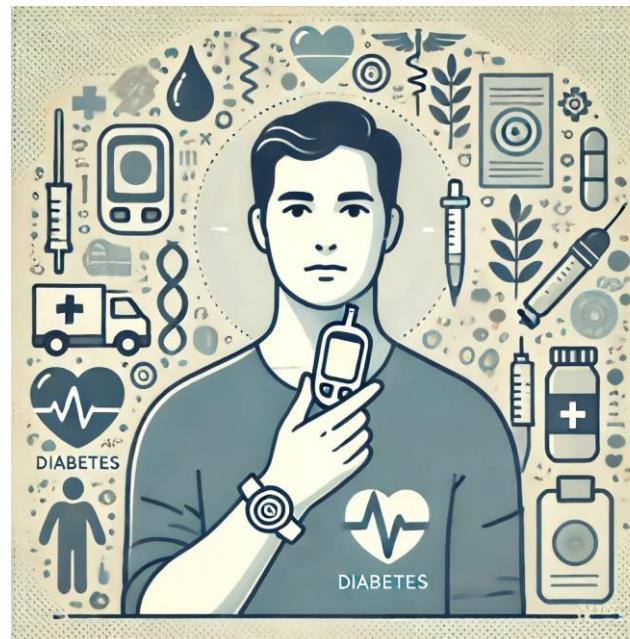
```
1 from sklearn.metrics import mean_squared_error, r2_score
2
3 y_test_predict = model.predict(X_test)
4 rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
5 r2 = r2_score(y_test, y_test_predict)
6
7 print('테스트데이터 예측 rmse : {}'.format(rmse))
8 print('테스트데이터 예측 R2 score: {}'.format(r2)) # 테스트 데이터로 예측한 성능
```

```
테스트데이터 예측 rmse : 58.74293666685849
테스트데이터 예측 R2 score: 0.5033169770750128
```

2) ML 주요 이론(Supervised Learning) 실습 1)

실습 1) Diabetes data set을 기반으로 제시된 코드를 참조하여 데이터의 분석 및 전처리를 추가한 후 분석데이터를 공유해 봅시다

실습 2) Diabetes data set을 기반으로 다양한 파라미터와 학습한 모델들을 사용하여 최적의 성능을 제시하는 모델을 학습 후 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

실습) 제시된 데이터 셋의 실험환경 구성/파라미터 설정값을 조정하며 모델의 학습결과를 평가 후 설명 리포트와 함께 공유해 봅시다.

	Multivariant Regression	After Standardized	Lasso Regression	Ridge Regression	Elastic Regression	Polynomial Regression
R2 of X train (Score())	0.6047	0.6072	0.6048	0.6113	0.6022	0.6793
R2 of X test (Score())	0.6094	0.6144	0.6031	0.5832	0.6193	0.6556

California House Price Prediction

2) ML 주요 이론(Supervised Learning)

	Multivariant Regression	After Standardized	Lasso Regression	Ridge Regression	Elastic Regression	Polynomial Regression
R2 of X train (Score())	0.6047	0.6072	0.6048	0.6113	0.6022	0.6793
R2 of X test (Score())	0.6094	0.6144	0.6031	0.5832	0.6193	0.6556

	Multivariant Regression	After Standardized	Lasso Regression (Alpha=10)	Ridge Regression (Alpha=10)	Elastic Regression (A=0.005, L1=0.05)	Polynomial Regression (degree 2, scaled)
R2 of X train (Score())	0.6017	0.6072	0.6017 0.6017 (A=0.001)	0.6017	0.6015	0.6851
R2 of X test (Score())	0.6206	0.6144	0.6153 0.6153 (A=0.001)	0.6156	0.6146	0.6625

2) ML 주요 이론(Supervised Learning)

Logistic Regression
classification

2) ML 주요 이론(Supervised Learning)

Logistic Regression

sklearn.linear_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

[source]

Parameters: `penalty : {'l1', 'l2', 'elasticnet', None}, default='l2'`

Specify the norm of the penalty:

- `None`: no penalty is added;
- `'l2'`: add a L2 penalty term and it is the default choice;
- `'l1'`: add a L1 penalty term;
- `'elasticnet'`: both L1 and L2 penalty terms are added.

C : float, default=1.0

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

1.C: 정규화 강도를 조절하는 매개변수. C 값이 작을수록 정규화가 강하게 적용되어 모델이 단순해지고 일반화 성능이 향상될 수 있다. C의 기본값은 1.0.

예시:

- C=0.1: 모델에 강한 정규화가 적용되어 단순한 모델이 선호.
- C=1.0: 모델에 중간 정도의 정규화가 적용.
- C=10.0: 모델에 약한 정규화가 적용되어 복잡한 모델이 선호.

2.penalty: 규제 유형을 지정하는 매개변수.

- 'l1': L1 규제는 모델의 가중치를 0에 가깝게 만들어 특성 선택(Feature Selection) 효과를 가지며, 모델을 희소하게 만들어 준다. 즉, 일부 특성만이 모델에 영향.
- 'l2': L2 규제는 모델의 가중치를 작게 만들어 모든 특성이 모델에 영향

2) ML 주요 이론(Supervised Learning)

Logistic Regression

로지스틱 회귀의 파라미터 C와 릿지 회귀의 알파 값

로지스틱 회귀에서 파라미터 c는 정규화 강도의 역수
모델에 적용되는 정규화의 강도를 조절.

c 값이 작을수록 강한 정규화가 적용되어 개별 데이터의 영향력을 줄일 수 있다.
c 값이 클수록 정규화 강도가 감소하며, 모델이 학습 데이터에 더 적합하게 맞출 수 있지만, 과적합의 위험성도 커진다.

릿지 회귀에서의 알파 값은 회귀 계수에 적용되는 축소(shrinkage)의 정도를 조절
알파 값이 클수록 축소의 정도가 커지고, 회귀 계수가 0에 가까워진다.
개별 변수의 영향력을 감소시키고, 다중공선성의 영향을 완화시킬 수 있다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression(Iris Data set)

붓꽃(*Iris sanguinea*)

- 늙은 아이리스
- 꽃봉오리가 마치 먹물을 머금은 붓과 같아서 '붓꽃'이라고 불리고 있습니다.
- 꽃잎의 모양과 길이에 따라 여러 가지 품종으로 나뉘어집니다.

Setosa



Vergicolor



Veriginica



붓꽃의 종류와 모양에 대한 이해를 시도해 본다

2) ML 주요 이론(Supervised Learning)

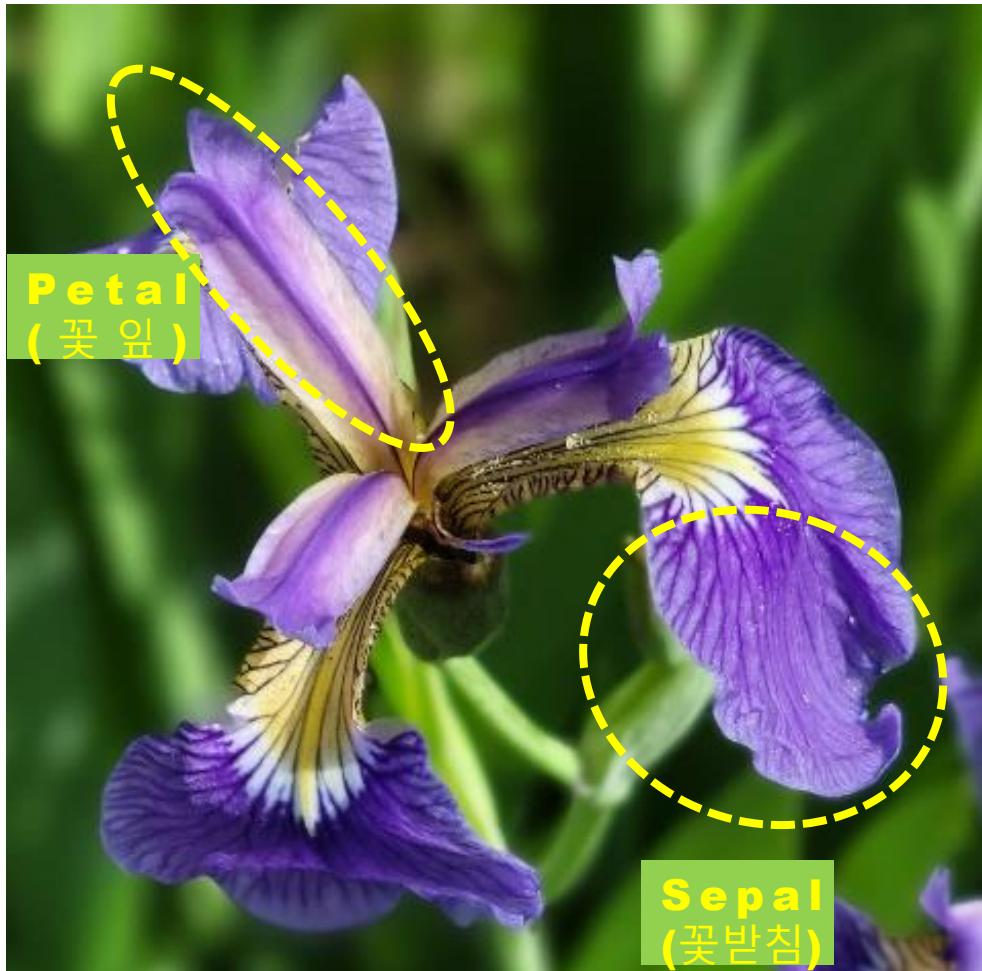
Logistic Regression



Iris(붓꽃)의 구별법은 ???

2) ML 주요 이론(Supervised Learning)

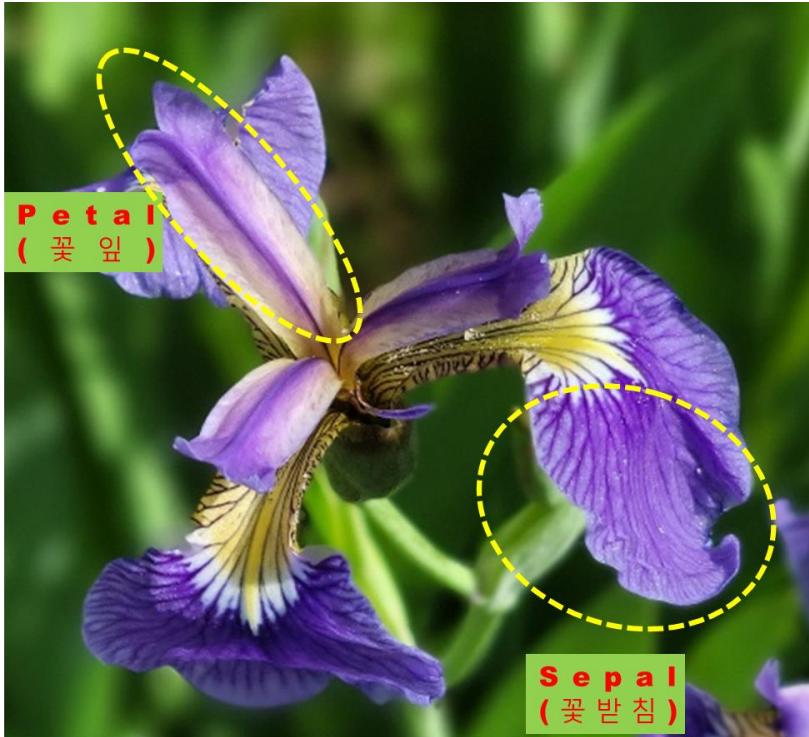
Logistic Regression



- 꽃받침의 길이 Sepal length
- 꽃받침의 폭 Sepal width
- 꽃잎의 길이 Petal length
- 꽃잎의 폭 Petal width

2) ML 주요 이론(Supervised Learning)

Logistic Regression



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3	1.4	0.1	setosa
14	4.3	3	1.1	0.1	setosa
15	5.8	4	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa

붓꽃의 4가지 특성에 따른 분류는 가능한 것인가???

2) ML 주요 이론(Supervised Learning)

Logistic Regression

The screenshot shows the scikit-learn documentation page for the `load_iris` dataset. The top navigation bar includes links for Install, User Guide, API, Examples, Community, and More. On the left, there's a sidebar with links for `sklearn.datasets.load_iris`, Examples using `sklearn.datasets.load_iris`, and a note to cite the software. The main content area has a title `sklearn.datasets.load_iris` and a code snippet `sklearn.datasets.load_iris(*, return_X_y=False, as_frame=False)`. A link to [source] is also present. Below the code snippet is a description: "Load and return the iris dataset (classification)." It is described as a "classic and very easy multi-class classification dataset". To the right of the description is a table showing dataset statistics:

Classes	3
Samples per class	50
Samples total	150
Dimensionality	4
Features	real, positive

붓꽃 데이터의 로드 방법과 데이터 소개.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 from sklearn.linear_model import LogisticRegression  
2 from sklearn.model_selection import train_test_split  
3 import pandas as pd  
4 from sklearn.datasets import load_iris
```

```
1 iris = load_iris()  
2 print(iris.DESCR)
```

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
 - sepal length in cm
 - sepal width in cm
 - petal length in cm
 - petal width in cm
 - class:
 - Iris-Setosa
 - Iris-Versicolour
 - Iris-Virginica

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)
2 species = pd.Series(iris.target, dtype='category') # 목적데이터의 시리즈 생성(카테고리type)
3 species = species.cat.rename_categories(iris.target_names) # 시리즈의 이름 변경
4 iris_df['species']= species                                # 시리즈를 iris_df에 추가
```

환경을 업로드한 자료를 데이터 프레임 형태로 구성한다.
특히 분류를 위한 카테고리 형태의 데이터 타입으로 타겟데이터를 변형한다.

목적 데이터를 카테고리 형태로 만들면 분류 모델로 사용되며, 클래스
레이블을 예측하는 작업을 수행.

카테고리 형태로 만들지 않고 학습 및 예측하는 경우에는 회귀 모델로
간주되며, 연속적인 값을 예측하는 작업을 수행.

목적 데이터의 형태는 모델의 동작과 결과에 영향을 미칠 수 있다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 iris_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  -- 
 0   sepal length (cm)    150 non-null   float64
 1   sepal width (cm)    150 non-null   float64
 2   petal length (cm)   150 non-null   float64
 3   petal width (cm)    150 non-null   float64
 4   species            150 non-null   category
dtypes: category(1), float64(4)
memory usage: 5.1 KB
```

붓꽃 데이터의 자료 정보를 확인해 본다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
[53] 1 iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	
0	5.1	3.5	1.4	0.2	setosa	
1	4.9	3.0	1.4	0.2	setosa	
2	4.7	3.2	1.3	0.2	setosa	
3	4.6	3.1	1.5	0.2	setosa	
4	5.0	3.6	1.4	0.2	setosa	

붓꽃 데이터의 실제 데이터 상위 5개를 확인해 본다.

2) ML 주요 이론(Supervised Learning)

```
[62] 1 iris_df.describe() # feature data의 통계적 정리
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

붓꽃 데이터의 통계적 분포 현황을 살펴 본다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

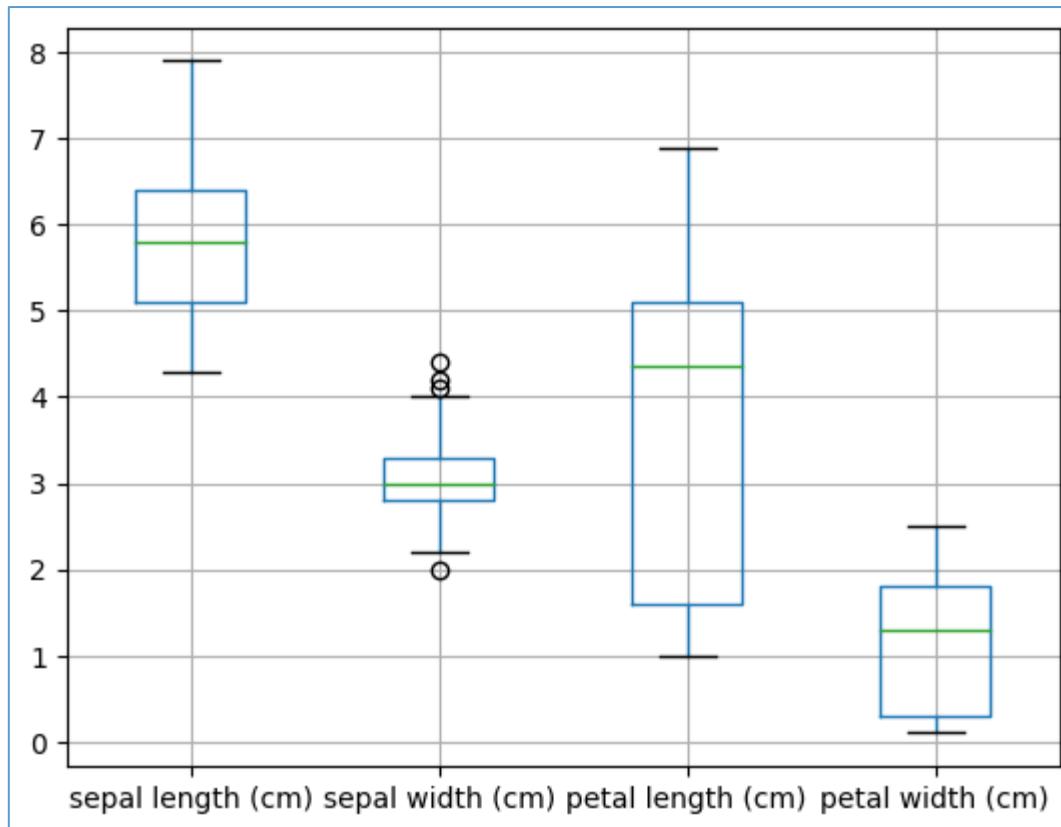
```
1 print(iris) # 데이터 프레임 작성 전의 로드된 상태의 전체 데이터의 정보를 보여준다.
```

```
[5.7, 2.8, 4.1, 1.3],  
[6.3, 3.3, 6., 2.5],  
[5.8, 2.7, 5.1, 1.9],  
[7.1, 3., 5.9, 2.1],  
[6.3, 2.9, 5.6, 1.8],  
[6.5, 3., 5.8, 2.2],  
[7.6, 3., 6.6, 2.1]]
```

붓꽃 데이터의 실제 데이터 모습을 살펴 본다

2) ML 주요 이론(Supervised Learning)

1 iris_df.boxplot()



붓꽃 데이터의 특성값의 분포를 시각적으로 확인해 본다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

박스플롯(boxplot)

데이터의 분포와 중앙값, 사분위수를 시각적으로 표현하는 그래프.

상자 (Box):

상자의 아랫부분은 25%에 해당하는 1사분위수(Q1)를 나타낸다.

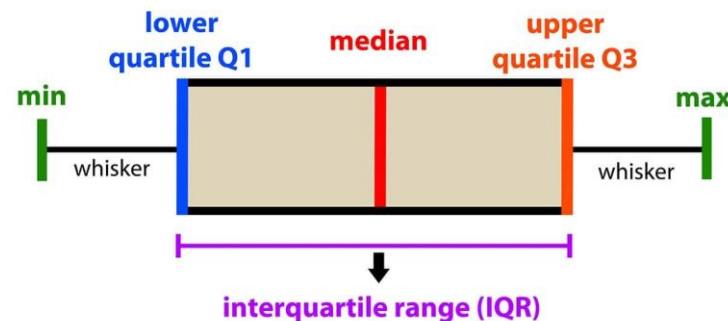
상자의 중간 부분은 50%에 해당하는 2사분위수 또는 중앙값(Q2 또는 median)을 나타냅니다. 상자의 윗부분은 75%에 해당하는 3사분위수(Q3)를 나타냅니다.

위스커 (Whiskers):

데이터의 전체 범위를 나타낸다.

일반적으로, 수염의 길이는 $1.5 * (Q3 - Q1)$ 로 정의되며, 이를 벗어나는 값은 이상치(outlier)로 간주될 수 있다.

introduction to data analysis: Box Plot



2) ML 주요 이론(Supervised Learning)

Logistic Regression

박스플롯(boxplot)

중앙값 (Median):

중앙값은 데이터를 작은 값부터 큰 값 순서로 정렬했을 때 가운데에 위치한 값.
박스플롯에서는 상자의 중간 부분에 위치한 선으로 표시.

박스플롯을 해석할 때, 주로 다음과 같은 정보를 파악할 수 있다:

데이터의 중앙값을 확인하여 데이터의 중심 경향성.

상자의 길이로 데이터의 밀집 정도를 파악.

수염을 통해 데이터의 전체 범위를 파악.

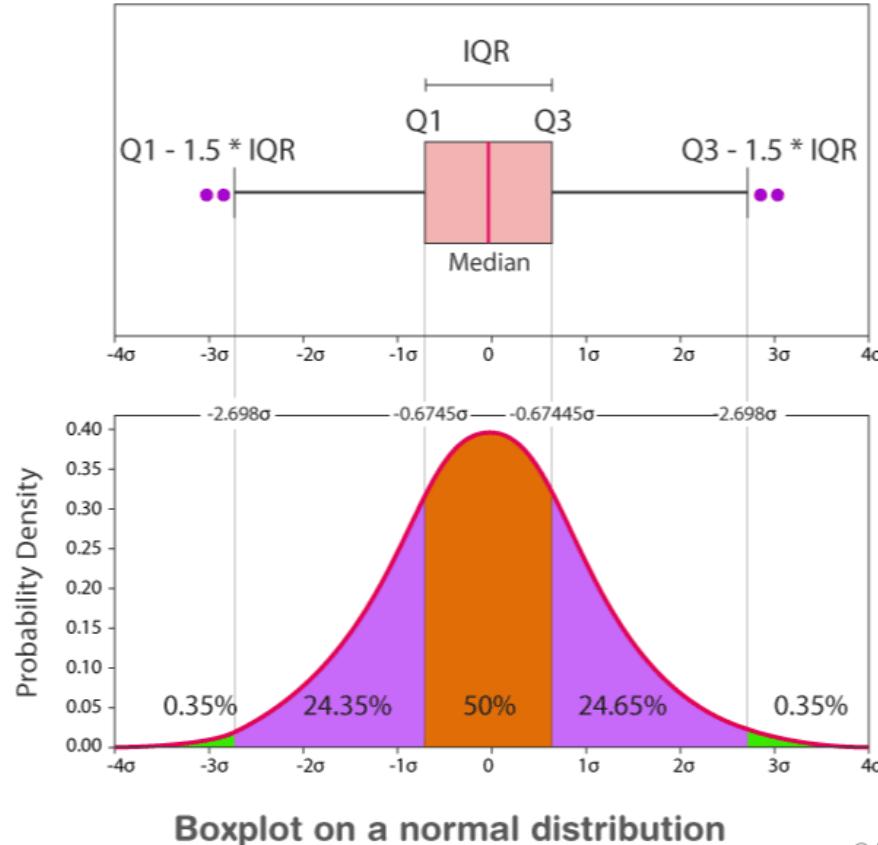
이상치를 확인하여 데이터의 특이값을 추정.

데이터의 분포와 이상치를 시각적으로 파악, 데이터의 특성과 편차를 이해

2) ML 주요 이론(Supervised Learning)

Logistic Regression

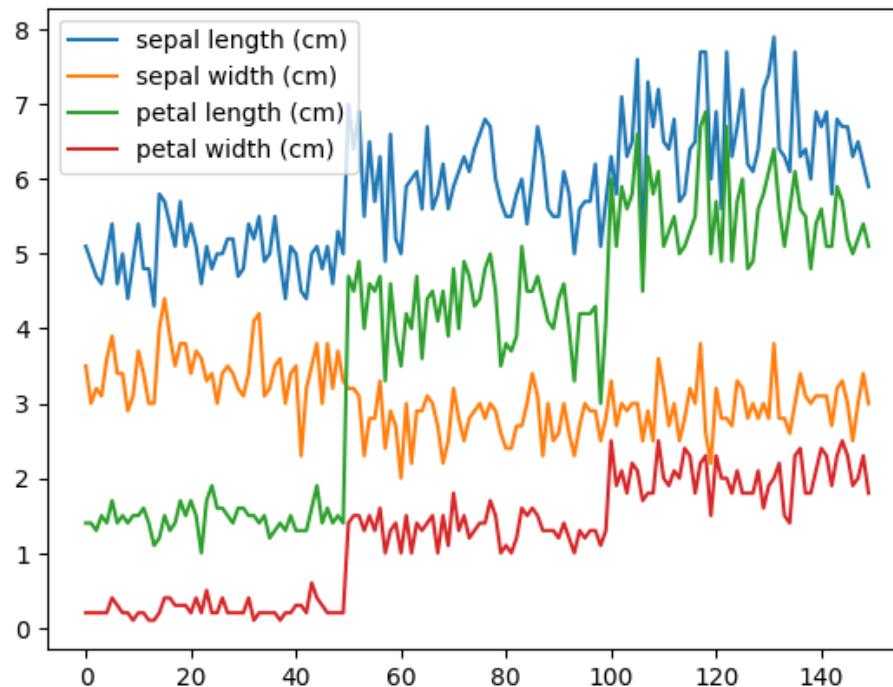
박스플롯(boxplot)



2) ML 주요 이론(Supervised Learning)

Logistic Regression

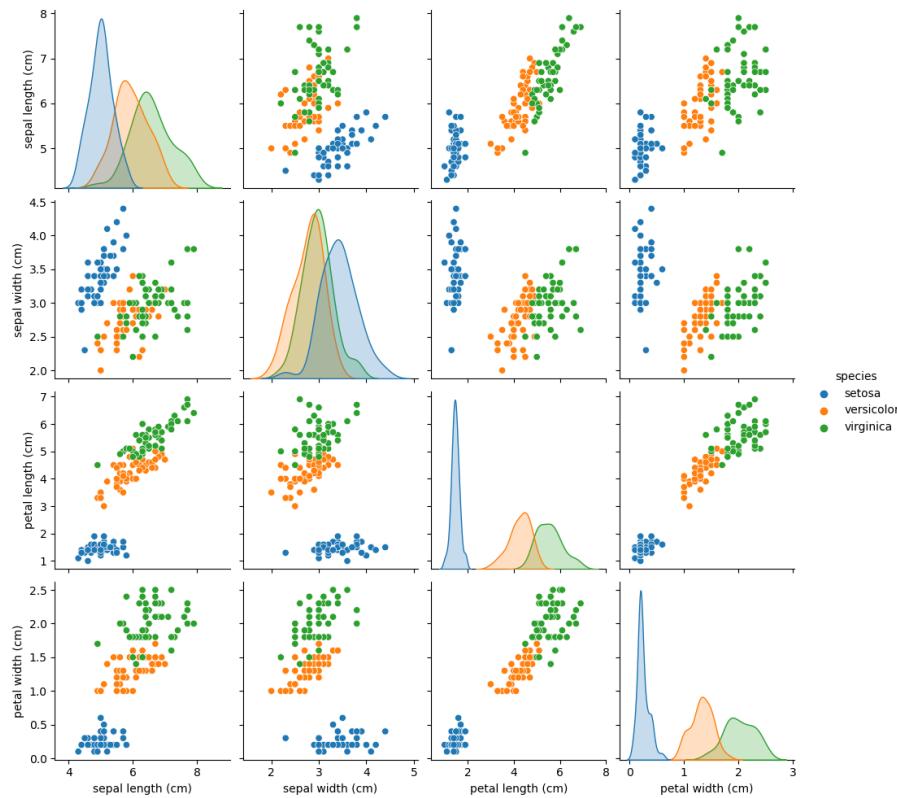
```
1 iris_df.plot()
```



붓꽃 데이터의 특성값을 라인 그래프로 시각화해 본다.

2) ML 주요 이론(Supervised Learning)

```
1 import seaborn as sns  
2  
3 sns.pairplot(iris_df, hue='species')
```



붓꽃 데이터의 특성값 별로 비교하여 상관관계를 유추해 본다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 X =iris_df.drop('species', axis=1)
2 y =iris_df['species']

8 # 학습 데이터와 테스트 데이터 분리
9 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
10
11 # 로지스틱 회귀 모델 생성 및 학습
12 logreg = LogisticRegression()
13 logreg.fit(X_train, y_train)
14
15 # 테스트 데이터로 예측
16 y_pred = logreg.predict(X_test)
17
18 # 예측 결과 평가
19 accuracy = logreg.score(X_test, y_test)
20 print("Accuracy:", accuracy)
```

Accuracy: 1.0

붓꽃 데이터의 학습 및 예측을 위해 입력특성값과 타겟값으로
준비하고 학습용 데이터와 예측용 데이터로 구분하고 예측해
보자. 정확도는 1.0이다.

2) ML 주요 이론(Supervised Learning)

1. score 메서드

사이킷런의 여러 모델 클래스에서 제공되는 메서드로, 모델이 특정 데이터셋에 대해 얼마나 잘 작동하는지 평가합니다.

사용법 **model.score(X, y)** 여기서 X는 입력 데이터, y는 실제 레이블입니다. 모델은 X에 대해 예측을 수행하고, 그 예측값을 y와 비교하여 정확도를 계산합니다.

2. accuracy_score

사이킷런의 **metrics** 모듈에서 제공되며, 예측값과 실제값을 직접 비교하여 정확도를 계산합니다. 이 함수는 모델의 predict 메서드를 사용하여 예측값을 먼저 생성한 후, 그 예측값과 실제값을 비교합니다.

사용법 **accuracy_score(y_true, y_pred)** 여기서 y_true는 실제 레이블, y_pred는 예측값입니다.

```
1 from sklearn.metrics import accuracy_score  
2  
3 accuracy_score= accuracy_score(y_test, y_pred)  
4 print('accuracy_score : ', accuracy_score)
```

```
accuracy_score : 1.0
```

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 print(y_test)  
2 print(y_pred)
```

```
73      versicolor  
18       setosa  
118     virginica  
78      versicolor  
76      versicolor  
31       setosa  
64      versicolor  
141     virginica  
68      versicolor  
82      versicolor  
110     virginica  
12       setosa  
36       setosa  
9        setosa  
19       setosa  
56      versicolor  
104     virginica  
69      versicolor  
55      versicolor  
132     virginica  
29       setosa  
127     virginica  
26       setosa  
128     virginica  
131     virginica  
145     virginica  
108     virginica  
143     virginica  
45       setosa  
30       setosa
```

```
Categories (3, object): ['setosa', 'versicolor', 'virginica']  
['versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor', 'setosa',  
 'versicolor', 'virginica', 'versicolor', 'versicolor', 'virginica', 'setosa',  
 'setosa', 'setosa', 'setosa', 'versicolor', 'virginica', 'versicolor',  
 'versicolor', 'virginica', 'setosa', 'virginica', 'setosa', 'virginica',  
 'virginica', 'virginica', 'virginica', 'virginica', 'setosa', 'setosa']
```

붓꽃 데이터의 학습 및 예측을 위해
입력특성값과 타겟값으로 준비하고
학습용 데이터와 예측용 데이터로
구분하고
예측해 본다.(정확도 : 1.0)

2) ML 주요 이론(Supervised Learning)

Logistic Regression(범주형이 아닌 수치형으로 업로드 후 학습/예측 경우)

```
1 # Iris 데이터셋 불러오기  
2 iris = load_iris()  
3  
4 # 독립 변수와 종속 변수 분리  
5 X = iris.data  
6 y = iris.target  
7  
8 # 학습 데이터와 테스트 데이터 분리  
9 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

붓꽃 데이터의 학습 및 예측을 위해 입력특성값과 타겟값으로
준비하고 학습용 데이터와 예측용 데이터로 구분

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
11 # 로지스틱 회귀 모델 생성 및 학습  
12 logreg = LogisticRegression()  
13 logreg.fit(X_train, y_train)  
14  
15 # 테스트 데이터로 예측  
16 y_pred = logreg.predict(X_test)  
17  
18 # 예측 결과 평가  
19 accuracy = logreg.score(X_test, y_test)  
20 print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

2) ML 주요 이론(Supervised Learning)

Logistic Regression

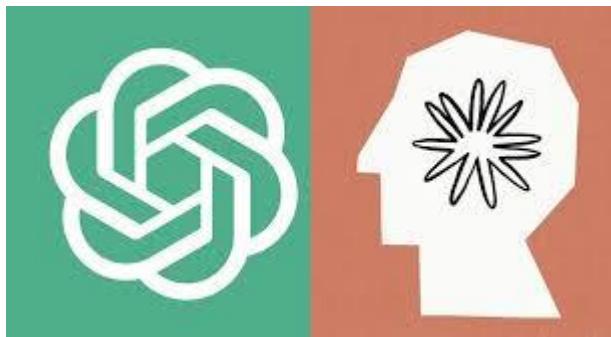
```
1 print(y_test)  
2 print(y_pred)
```

```
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]  
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```

데이터의 숫자가 많지않아서 범주형과 수치형 데이터의 목적값이 정확도에서 차이를 보이지 않았으나
데이터의 상황에 따라 결과값이 달라질 수 있다.

2) ML 주요 이론(Supervised Learning)

실습 1) 학습한 내용을 정리하고 유사한 코드를 생성하여 내용을 정리하고 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

Logistic Regression
(이진분류 vs 다중분류)

2) ML 주요 이론(Supervised Learning)

For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss. ‘liblinear’ and ‘newton-cholesky’ only handle binary classification but can be extended to handle multiclass by using OneVsRestClassifier.

The screenshot shows the scikit-learn API Reference page for the `LogisticRegression` class. The URL is https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. The page has a sidebar on the left listing various classes under `sklearn.linear_model`, including `LogisticRegression`, `LogisticRegressionCV`, `PassiveAggressiveClassifier`, `Perceptron`, `RidgeClassifier`, `RidgeClassifierCV`, `SGDClassifier`, `SGDOneClassSVM`, `LinearRegression`, `Ridge`, `RidgeCV`, `SGDRegressor`, `ElasticNet`, `ElasticNetCV`, `Lars`, `LarsCV`, `Lasso`, `LassoCV`, `LassoLars`, `LassoLarsCV`, `LassoLarsIC`, `OrthogonalMatchingPursuit`, `OrthogonalMatchingPursuitCV`, and `ARDRegression`. The main content area displays the `LogisticRegression` class definition:

```
class sklearn.linear_model.LogisticRegression(penalty='L2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
    Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)
```

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. Note that regularization is applied by default. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation, or no regularization. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the 'saga' solver.

Read more in the [User Guide](#).

Parameters:

- penalty** : {‘**l1**’, ‘**l2**’, ‘**elasticnet**’, **None**}, **default** = ‘**l2**’

Specify the norm of the penalty:

- **None** : no penalty is added;
- ‘**l2**’ : add a L2 penalty term and it is the default choice;

2) ML 주요 이론(Supervised Learning)

scikit-learn Install User Guide API Examples Community More ▾

sklearn.isotonic

sklearn.kernel_approximation

sklearn.kernel_ridge

sklearn.linear_model

sklearn.manifold

sklearn.metrics

sklearn.mixture

sklearn.model_selection

sklearn.multiclass

[OneVsOneClassifier](#)

[OneVsRestClassifier](#)

[OutputCodeClassifier](#)

sklearn.multioutput

sklearn.naive_bayes

sklearn.neighbors

sklearn.neural_network

sklearn.pipeline

sklearn.preprocessing

sklearn.random_projection

sklearn.semi_supervised

sklearn.svm

sklearn.tree

sklearn.utils

sklearn.multiclass

Multiclass learning algorithms.

- one-vs-the-rest / one-vs-all
- one-vs-one
- error correcting output codes

The estimators provided in this module are meta-estimators: they require a base estimator to be provided in their constructor. For example, it is possible to use these estimators to turn a binary classifier or a regressor into a multiclass classifier. It is also possible to use these estimators with multiclass estimators in the hope that their accuracy or runtime performance improves.

All classifiers in scikit-learn implement multiclass classification; you only need to use this module if you want to experiment with custom multiclass strategies.

The one-vs-the-rest meta-classifier also implements a `predict_proba` method, so long as such a method is implemented by the base classifier. This method returns probabilities of class membership in both the single label and multilabel case. Note that in the multilabel case, probabilities are the marginal probability that a given sample falls in the given class. As such, in the multilabel case the sum of these probabilities over all possible labels for a given sample *will not* sum to unity, as they do in the single label case.

User guide. See the [Multiclass classification](#) section for further details.

OneVsOneClassifier	One-vs-one multiclass strategy.
OneVsRestClassifier	One-vs-the-rest (OvR) multiclass strategy.
OutputCodeClassifier	(Error-Correcting) Output-Code multiclass strategy.

2) ML 주요 이론(Supervised Learning)

로지스틱 회귀의 다중 클래스 분류

Logistic Regression은 기본적으로 이진 분류(Binary Classification)에 사용하고
자동으로 멀티클래스 분류도 처리!!

다중 클래스 분류를 처리 : One-vs-Rest(OvR) 방식과 소프트맥스 회귀(Softmax Regression).

1. 이진 분류 vs 다중 클래스 분류

이진 분류에서는 한 개의 결과만을 예측합니다(예: 0 또는 1).

다중 클래스 분류에서는 여러 개의 클래스 중 하나를 선택해야 합니다.

예를 들어, 3개 이상의 클래스가 있을 경우, 각 클래스에 대한 예측을 해야 합니다.

2) ML 주요 이론(Supervised Learning)

2. 로지스틱 회귀에서 다중 클래스 처리 방법

2.1 One-vs-Rest (OvR) 방식

다중 클래스 문제를 여러 개의 이진 분류 문제로 나누는 방식

각 클래스마다 이진 분류기를 학습시키고, 각각의 이진 분류기에서 예측한 확률을 바탕으로 가장 높은 확률을 가진 클래스를 예측합니다.

예를 들어, 클래스 0 vs 클래스 1, 클래스 2,

클래스 1 vs 클래스 0, 클래스 2,

클래스 2 vs 클래스 0, 클래스 1을 학습하여 예측

2.2 소프트맥스 회귀 (Multinomial Logistic Regression)

다중 클래스 문제를 단일 모델로 처리하는 방법

모든 클래스를 동시에 분류할 수 있도록 출력층에 소프트맥스 함수를 적용

모든 클래스에 대한 확률을 출력하고, 가장 확률이 높은 클래스를 선택

2) ML 주요 이론(Supervised Learning)

3. Multiclass Classifier

MulticlassClassifier는 다중 클래스를 처리할 수 있는 모든 모델을 포괄.
LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, SVM 등
각 모델은 다중 클래스 분류를 처리하는 방식이 다릅니다.

One-vs-Rest (OvR): 각 클래스를 다른 모든 클래스로 구분하는 방식으로, 로지스틱 회귀에서 사용되는 방법입니다.

One-vs-One: 각 쌍의 클래스를 구분하는 이진 분류기를 여러 개 학습시킵니다.

OutputCodeClassifier: 각 클래스에 대해 독립적인 이진 분류기를 학습시키고, 이진 분류기의 예측 결과를 코드로 변환하여 다중 클래스 분류를 해결

Softmax (Multinomial Logistic Regression):

소프트맥스 회귀를 사용하여 다중 클래스 문제를 단일 모델로 해결합니다.

2) ML 주요 이론(Supervised Learning)

Logistic regression(binary classification)

Breast Cancer Dataset : 양성(Benign) vs 악성(Malignant)

각 샘플은 암 세포의 다양한 특성에 대한 측정값을 가지고 있다.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 데이터 로드
data = load_breast_cancer()
X, y = data.data, data.target

# 학습/테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 로지스틱 회귀 모델 생성
model = LogisticRegression(max_iter=10000) # 최대 반복 횟수 설정

# 모델 학습
model.fit(X_train, y_train)

# 예측
y_pred = model.predict(X_test)

# 정확도 계산
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
```

2) ML 주요 이론(Supervised Learning)

Logistic regression(binary classification : one vs rest)

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Iris 데이터셋 로드 (클래스 2 제외)
iris = load_iris()
X, y = iris.data, iris.target
X, y = X[y != 2], y[y != 2] # 이진 분류로 변환

# 학습 데이터와 테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 로지스틱 회귀 모델 생성
model = LogisticRegression()

# 학습
model.fit(X_train, y_train)

# 예측
y_pred = model.predict(X_test)

# 정확도 계산
accuracy = model.score(X_test, y_test)
print(f'Accuracy: {accuracy:.4f}')
```

2) ML 주요 이론(Supervised Learning)

OneVsRestClassifier + Logistic regression(multi class)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score

# 데이터 로드
iris = load_iris()
X, y = iris.data, iris.target

# 학습/테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 로지스틱 회귀 모델 생성
model = LogisticRegression(max_iter=200)

# OneVsRestClassifier 사용하여 다중 클래스 분류기 생성
ovr_model = OneVsRestClassifier(model)

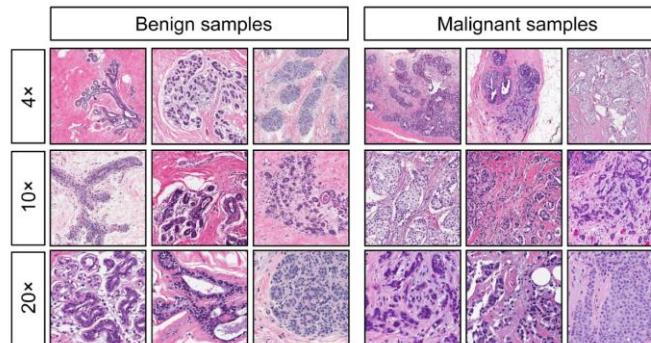
# 모델 학습
ovr_model.fit(X_train, y_train)

# 예측
y_pred = ovr_model.predict(X_test)

# 정확도 계산
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
```

2) ML 주요 이론(Supervised Learning)

Logistic Regression (Breast Cancer Data Set)



2) ML 주요 이론(Supervised Learning)

Logistic Regression(Breast Cancer Data Set)

Breast cancer 관련 특성들과
특성에 기반한 양성(benign)과 악성(malignant) 종양의 분류 정보를 포함

데이터셋에는 총 30개의 특성(feature)이 포함, 조직의 세포학적 측정값을 나타낸다.

1. 반경(radius): 조직 점의 중심부에서 암 세포까지의 평균 거리
2. 질감(texture): 그레이스케일 값의 표준 편차로서, 암 세포 영역의 픽셀값의 변동성을 나타냄
3. 둘레(perimeter): 세포들이 그려진 윤곽의 길이
4. 면적(area): 세포들이 그려진 영역의 면적
5. 부드러움(smoothness): 윤곽의 길이의 변동성
6. 연질(compactness): 윤곽의 길이의 제곱에 비례하는 암 세포의 둘레 제곱의 비율
7. 오목함(concavity): 오목한 부분의 윤곽의 정도
8. 오목점의 수(concave points): 오목한 부분의 윤곽에서 발견된 오목점의 수
9. 대칭성(symmetry): 세포들의 대칭성 정도
10. 프랙탈 차원(fractal dimension): 세포 윤곽의 굴곡 정도를 나타내는 프랙탈 차원

.....

Logistic Regression

2) ML 주요 이론(Supervised Learning)

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Class Distribution: 212 - Malignant, 357 - Benign

특성 데이터와 타겟 데이터의 구성

2) ML 주요 이론(Supervised Learning)

실습) 제시된 데이터셋(Breast Cancer)의 자료를 참조하여 특성값 등을 이해하고 정리한 자료를 공유해 주세요

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 import pandas as pd  
2 from sklearn.datasets import load_breast_cancer  
3 from sklearn.model_selection import train_test_split  
4 from sklearn.linear_model import LogisticRegression  
5 from sklearn.metrics import accuracy_score, confusion_matrix  
6  
7 # 데이터 불러오기  
8 data = load_breast_cancer()  
9 X = pd.DataFrame(data.data, columns=data.feature_names)  
10 y = data.target
```

모델링을 위한 환경을 구성하고 입력용 특성데이터와 타겟데이터 구분

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 print(data.DESCR)

.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
-----
**Data Set Characteristics:**

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:
- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
```

데이터의 구성 및 특징 등을 살펴본다

2) ML 주요 이론(Supervised Learning)

Logistic Regression

1 X.describe()

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...

8 rows x 30 columns

X 데이터의 구성 및 특징 등을 살펴본다

2) ML 주요 이론(Supervised Learning)

Logistic Regression

Y 데이터의 내용을 확인한다

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 # 데이터 분할  
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
3  
4 # 로지스틱 회귀 모델 학습  
5 model = LogisticRegression()  
6 model.fit(X_train, y_train)
```

학습용과 테스트용을 8:2로 구분하고 모델을 학습시킨다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 # 테스트 데이터로 예측 수행  
2 y_pred = model.predict(X_test)  
3  
4 # 정확도 평가  
5 accuracy = accuracy_score(y_test, y_pred)  
6 print("Accuracy: ", accuracy)
```

```
Accuracy: 0.9649122807017544
```

학습된 모델로 예측을 하고 정확도를 평가해 본다.(**96.49%**)

모델의 성능 개선을 위한 시도를 해본다.

2) ML 주요 이론(Supervised Learning)

Logistic Regression

`sklearn.model_selection.GridSearchCV`

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=np.nan, return_train_score=False) [source]
```

Parameters:

estimator : estimator object

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : dict or list of dictionaries

Dictionary with parameters names (`str`) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

cv : int, cross-validation generator or an iterable, default=None

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- `None`, to use the default 5-fold cross validation,
- `integer`, to specify the number of folds in a `(Stratified)KFold`,

2) ML 주요 이론(Supervised Learning)

Logistic Regression(모델의 성능개선 시도 :표준화/GridSearch)

```
1 from sklearn.linear_model import LogisticRegression
2 from sklearn.model_selection import GridSearchCV
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import make_pipeline
5 from sklearn.datasets import load_breast_cancer
6
7 # 데이터를 로드하고 전처리(학습데이터의 표준화).
8 X_train, y_train = load_breast_cancer(return_X_y=True)
9 scaler = StandardScaler()
10 X_train_scaled = scaler.fit_transform(X_train)
11
12 # 모델과 파라미터 그리드를 정의.
13 model = LogisticRegression()
14 param_grid = {'C': [0.1, 1, 10], 'penalty': ['l1', 'l2']}
```

모델의 성능 개선을 위한 시도 : 표준화/파라미터 튜닝,

x,y데이터만 업로드하여 데이터의 분할 없이 학습(그리드서치 이해 목적!!)

2) ML 주요 이론(Supervised Learning)

Logistic Regression(Grid Search)

```
16 # 그리드 서치를 수행.  
17 grid_search = GridSearchCV(model, param_grid, cv=5)  
18 grid_search.fit(X_train_scaled, y_train)  
19  
20 # 최적 파라미터와 최고 정확도를 출력.  
21 print("Best Hyperparameters:", grid_search.best_params_)  
22 print("Best Accuracy:", grid_search.best_score_)  
23  
24 # 테스트 데이터에 대해 모델을 평가.  
25 X_test, y_test = load_breast_cancer(return_X_y=True)  
26 X_test_scaled = scaler.transform(X_test) #테스트 입력 데이터 표준화  
27 test_accuracy = grid_search.score(X_test_scaled, y_test)  
28 print("Test Accuracy:", test_accuracy)
```

Best Hyperparameters: {'C': 1, 'penalty': 'l2'}

Best Accuracy: 0.9806862288464524

Test Accuracy: 0.9876977152899824

초기 fit.transform에서 학습된 평균/표준편차로 데이터 변환시 transform 사용

2) ML 주요 이론(Supervised Learning)

Logistic Regression

```
1 model = LogisticRegression(C=1, penalty='l2')
2 model.fit(X_train_scaled, y_train)
```

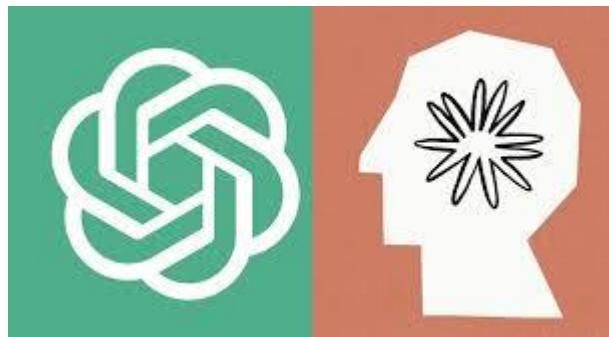
```
1 # 테스트 데이터로 예측 수행
2 y_pred = model.predict(X_test_scaled)
3
4 # 정확도 평가
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy:", accuracy)
```

```
Accuracy: 0.9876977152899824
```

특성 데이터를 표준화 시키고
최적의 파라미터를 선택하여 모델링을
개선시도(정확도 : 96.49% > **98.77%**)

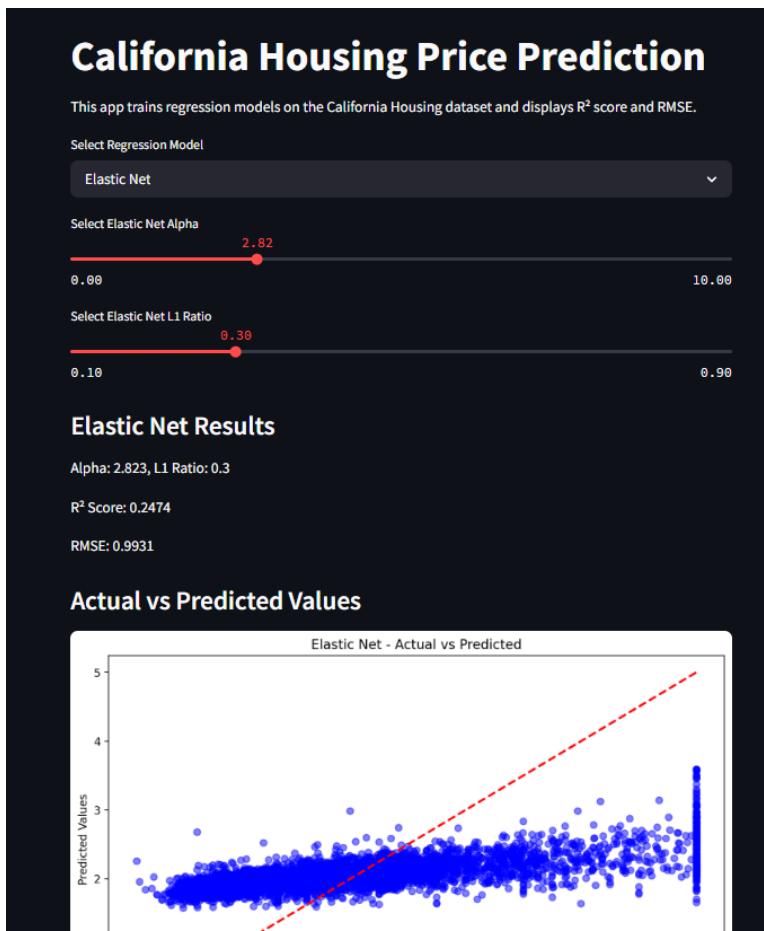
2) ML 주요 이론(Supervised Learning)

실습 1) 학습한 내용을 정리하고 정리한 내용을 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

실습) 학습한 Regression을 응용하여 스트림릿으로 모델과 파라미터의 선택 및 시각화 등을 구현하고 코드를 공유해 봅시다



2) ML 주요 이론(Supervised Learning)

```
import streamlit as st
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt

# 데이터셋 로드
housing = fetch_california_housing(as_frame=True)
data = housing.frame
X = data.drop('MedHouseVal', axis=1)
y = data['MedHouseVal']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Streamlit 앱 설정
st.title("California Housing Price Prediction")
st.write("This app trains regression models on the California Housing dataset and displays R2 score and RMSE.")

# 모델 선택
model_name = st.selectbox("Select Regression Model", ["Lasso", "Ridge", "Elastic Net", "Polynomial Regression"])

# Polynomial Features 생성 (Polynomial Regression용)
if model_name == "Polynomial Regression":
    degree = st.slider("Select Polynomial Degree", min_value=1, max_value=5, value=2)
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
else:
    X_train_poly = X_train
    X_test_poly = X_test

# 모델 및 파라미터 설정 UI
if model_name == "Lasso":
    alpha_lasso = st.slider("Select Lasso Alpha", min_value=0.001, max_value=10.0, value=1.0, step=0.001)
    model = Lasso(alpha=alpha_lasso)

elif model_name == "Ridge":
    alpha_ridge = st.slider("Select Ridge Alpha", min_value=0.001, max_value=10.0, value=1.0, step=0.001)
    model = Ridge(alpha=alpha_ridge)

elif model_name == "Elastic Net":
    alpha_elastic = st.slider("Select Elastic Net Alpha", min_value=0.001, max_value=10.0, value=1.0, step=0.001)
    l1_ratio = st.slider("Select Elastic Net L1 Ratio", min_value=0.1, max_value=0.9, value=0.5, step=0.1)
    model = ElasticNet(alpha=alpha_elastic, l1_ratio=l1_ratio)

elif model_name == "Polynomial Regression":
    model = LinearRegression()

# 모델 학습
try:
    model.fit(X_train_poly, y_train)

# 예측
    y_pred = model.predict(X_test_poly)

# 성능 평가
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# 결과 출력
    st.write(f"### {model_name} Results")
    if model_name == "Lasso":
        st.write(f"Alpha: {alpha_lasso}")
    elif model_name == "Ridge":
        st.write(f"Alpha: {alpha_ridge}")
    elif model_name == "Elastic Net":
        st.write(f"Alpha: {alpha_elastic}, L1 Ratio: ({l1_ratio})")
    elif model_name == "Polynomial Regression":
        st.write(f"Degree: (degree={degree})")
    st.write(f"R2 Score: {r2:.4f}")
    st.write(f"RMSE: {rmse:.4f}")

# 실제 값 vs 예측 값 시각화
    st.write("### Actual vs Predicted Values")
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.scatter(y_test, y_pred, color='blue', alpha=0.5)
    ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
    ax.set_xlabel('Actual Values')
    ax.set_ylabel('Predicted Values')
    ax.set_title(f'{model_name} - Actual vs Predicted')
    st.pyplot(fig)

except Exception as e:
    st.error(f"An error occurred: {str(e)}")

# 데이터셋 정보
st.write("### Dataset Info")
st.write(f"Features: ({X.columns.tolist()})")
st.write(f"Number of Samples: {len(data)}")
st.write(f"Target: Median House Value (in $100,000s)")

# 학습 버튼 (새로고침 역할)
if st.button("Retrain Model"):
    st.rerun()
```