06. Regression

- Scikit-learn 기초
- Linear Regression
- Multiple Linear Regression
- Logistic Regression

Scikit-Learn 기초

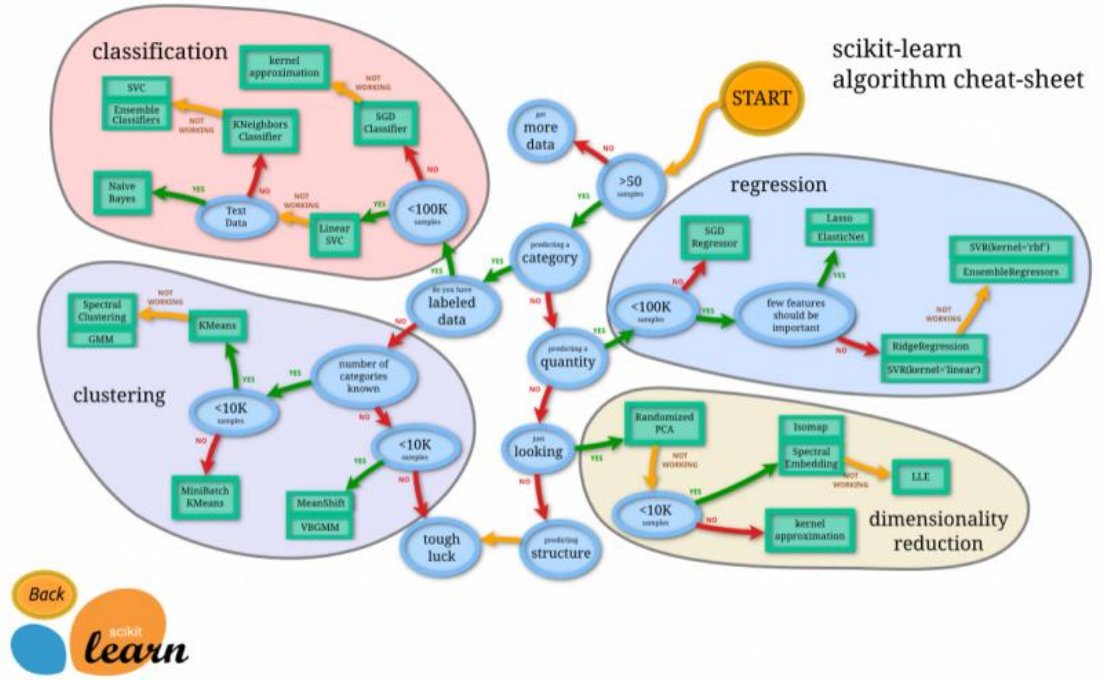
scikit-learn 이란?

- python기반의 대표적인 기계학습 라이브러리
  - scikit-learn은 오픈 소스로 공개되어 있으며, 개인, 비즈니스 관계없이 누구나 무료로 사용가능
  - 많은 기계학습 알고리즘이 구현되어 있으며, 같은 방식으로 이용이 가능
  - 기계학습을 배우기 시작할 때 적합한 라이브러리

- 설치
  - `python.exe -m pip install --upgrade pip`
  - `pip install scikit-learn`

scikit-learn의 주요기능

- **Regression**
  - SGD(stochastic gradient descent)
  - LASSO
  - Ridge, Liner SVR
  - Ensemble 등
- **Classification**
  - SGD, Linear SVC, k-NN 등
- **Clustering**
  - Kmeans, MeanShift 등
- **Dimensionality Reduction**
  - PCA 등



scikit-learn의 사용방법

- 기본적으로 다음과 같이 사용
  - 라이브러리 import
    - (예) from sklearn import svm
  - 데이터 준비(학습 또는 테스트데이터) 준비
    - sklearn의 datasets에는 샘플 데이터들이 제공되고 있음
  - 알고리즘 선택
    - 목적에 따라 알맞은 알고리즘을 선택하고 파라메터 지정
    - (예)svm.SVC(kernel='linear', C=10, gamma=0.1)
      - kernel: 커널함수(기본:rbf 즉, 가우시안 커널)
      - C: 오류허용도의 역수(기본 1.0, 값이 크면 과적합 가능성)
      - gamma: 학습데이터 의존도(값이 크면 과적합 가능성)
  - 학습
  - 테스트
  - 필요에 따라 학습결과, 정확도 등을 출력

```python
from sklearn import svm
from sklearn.metrics import accuracy_score

# prepare data (XOR)
train_data = [[0, 0], [0, 1], [1, 0], [1, 1]]
train_label = [0, 1, 1, 0]
test_data = [[0, 0], [0, 1], [1, 0], [1, 1]]
# select a learning algorithm
clf = svm.SVC(C=10, gamma=0.1) # C:오류허용도의 역수, gamma:학습data 의존도
# learning
clf.fit(train_data, train_label)
# testing
test_label = clf.predict(test_data)
# results
print(f"train_data = {train_data}, train_label = {train_label}")
print(f"test_data = {test_data}")
print(f"accuracy = {accuracy_score(train_label, test_label)}")
```

```
train_data = [[0, 0], [0, 1], [1, 0], [1, 1]], train_label = [0, 1, 1, 0]
test_data = [[0, 0], [0, 1], [1, 0], [1, 1]]
accuracy = 1.0
```

sklearn의 주요 모듈

- datasets
  - 사이킷런에 내장되어 예제로 제공하는 데이터셋
- preprocessing
  - 데이터 전처리에 필요한 다양한 가공 기능 제공
  - 문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등
- feature_selection
  - 알고리즘에 큰 영향을 미치는 특징을 우선순위대로 선택작업을 수행하는 다양한 기능 제공
- feature_extraction
  - 텍스트데이터나 이미지데이터의 벡터화된 특징을 추출하는데 사용

- **decomposition**
  - 차원 축소와 관련한 알고리즘을 지원하는 모듈
  - PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행
- **model_selection**
  - 교차 검증을 위한 학습용/테스트용 분리
  - Grid Search로 최적 파라미터 추출 등의 API 제공
- **metrics**
  - 분류, 회귀, 클러스터링, Pairwise에 대한 다양한 성능 측정방법 제공(Accuracy, Precision, Recall, ROC-AUC, RMSE 등)
- **pipeline**
  - 특징처리 등의 변환과 기계학습 알고리즘, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

기계학습 알고리즘 관련

모듈명	설명
ensemble	• 앙상블 알고리즘 제공 • 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공
linear_model	• 주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원 • SGD(Stochastic Gradient Desccent) 관련 알고리즘도 제공
naive_bayes	• 나이브 베이즈 알고리즘 제공(가우시안 NB. 다항 분포 NB 등)
neighbors	• 최근접 이웃 알고리즘 제공.(K-Nearest Neighborhood 등)
svm	• 서포트 벡터 머신 알고리즘 제공
tree	• 의사 결정 트리 알고리즘 제공
cluster	• 비지도 클러스터링 알고리즘 제공(K-평균, 계층형, DBSCAN 등)

자주 사용되는 모듈과 알고리즘

- **preprocessing 모듈**
  - **StandardScaler**
    - 평균을 제거하고 데이터를 단위 분산으로 조정
  - **MinMaxScaler**
    - 모든 feature 값이 0~1사이에 있도록 데이터를 재조정
  - **RobustScaler**
    - 아웃라이어의 영향을 최소화한 기법
    - 중간값(median)과 IQR(interquartile range:25%~75%범위의 데이터를 스케일링)을 사용

StandardScaler

```
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]
mean = [0. 0. 0.], std = [1. 1. 1.]
[[ 0.          -1.22474487  1.33630621]
 [ 1.22474487  0.          -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
```

```python
from sklearn import preprocessing
import numpy as np

data = np.array([[ 1., -1.,  2.],
                 [ 2.,  0.,  0.],
                 [ 0.,  1., -1.]])
print(data)

scaler = preprocessing.StandardScaler()
scaler.fit(data)
scaled_data = scaler.transform(data)

print(f"mean = {scaled_data.mean(axis=0)}, std = {scaled_data.std(axis=0
)}")
print(scaled_data)
```

MinMaxScaler

```
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]
mean = [0.5        0.5        0.44444444],
std = [0.40824829 0.40824829 0.41573971]
[[0.5        0.         1.        ]
 [1.         0.5        0.33333333]
 [0.         1.         0.        ]]
```

```python
from sklearn import preprocessing
import numpy as np

data = np.array([[ 1., -1.,  2.],
                 [ 2.,  0.,  0.],
                 [ 0.,  1., -1.]])
print(data)

scaler = preprocessing.MinMaxScaler()
scaler.fit(data)
scaled_data = scaler.transform(data)

print(f"mean = {scaled_data.mean(axis=0)}, std = {scaled_data.std(axis=0
)}")
print(scaled_data)
```

RobustScaler

```
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]
mean = [0.         0.         0.22222222],
std = [0.81649658 0.81649658 0.83147942]
[[ 0.         -1.          1.33333333]
 [ 1.          0.          0.        ]
 [-1.          1.         -0.66666667]]
```

```python
from sklearn import preprocessing
import numpy as np

data = np.array([[ 1., -1.,  2.],
                 [ 2.,  0.,  0.],
                 [ 0.,  1., -1.]])
print(data)

scaler = preprocessing.RobustScaler()
scaler.fit(data)
scaled_data = scaler.transform(data)

print(f"mean = {scaled_data.mean(axis=0)}, std = {scaled_data.std(axis=0
)}")
print(scaled_data)
```

metrics 모듈

- Regression에 사용되는 metric
  - **r2_score(y_true, y_pred)**
    - 일반적으로 R^2로 표시되는 계수(즉, y의 분산비율)를 계산
    - $R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n}(y_i - \bar{y}_i)^2}$
  - explained_variance_score(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average')
    - 일반적으로 분산의 차이를 계산(r2_score와 비슷함)
- Classification에 사용되는 metric
  - **accuracy_score(y_true, y_pred)**
    - 분류의 정확도를 계산
    - 예측된 값 y_pred가 y_true와 정확히 일치하는 개수의 비율의 계산
    - $accuracy(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N} 1(\hat{y} = y_i)$

```python
from sklearn.metrics import r2_score

y_true = [0, 1, 2, 3]
y_pred = [0, 2, 1, 3]
score = r2_score(y_true, y_pred)
print(score) # 0.6
```

```python
from sklearn.metrics import explained_variance_score

y_true = [0, 1, 2, 3]
y_pred = [0, 2, 1, 3]
score = explained_variance_score(y_true, y_pred)
print(score) # 0.6
```

```python
from sklearn.metrics import accuracy_score

y_true = [0, 1, 2, 3]
y_pred = [0, 2, 1, 3]
accuracy = accuracy_score(y_true, y_pred)
print(accuracy) # 0.5
accuracy = accuracy_score(y_true, y_pred, normalize=False)
print(accuracy) # 2
```

pipeline 모듈

- **Pipeline(steps)**
  - steps: (key, value)를 요소로 하는 리스트를 지정
    - key: 각 단계를 나타내는 문자열
    - value: 각 단계를 수행하는 객체
- **make_pipeline(steps)**
  - steps: 각 단계를 수행하는 객체

- **주의사항**
  - 마지막 단계를 제외한 나머지 단계는 transform(), fit_transform()이 있는 객체
  - 맨 마지막 단계는 predict()가 있는 객체

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.decomposition import PCA

pipe = Pipeline(steps=[('reduce_dim', PCA()), ('clf', SVC())])
print(pipe)
```

```
Pipeline(memory=None,
         steps=[('reduce_dim',
                 PCA(copy=True, iterated_power='auto', n_components=None,
                     random_state=None, svd_solver='auto', tol=0.0,
                     whiten=False)),
                ('clf',
                 SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
                     coef0=0.0, decision_function_shape='ovr', degree=3,
                     gamma='scale', kernel='rbf', max_iter=-1,
                     probability=False, random_state=None, shrinking=True,
                     tol=0.001, verbose=False))],
         verbose=False)
```

```
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.decomposition import PCA

pipe = make_pipeline(PCA(), SVC())
print(pipe)
```

```
Pipeline(memory=None,
         steps=[('pca',
                 PCA(copy=True, iterated_power='auto', n_components=None,
                     random_state=None, svd_solver='auto', tol=0.0,
                     whiten=False)),
                ('svc',
                 SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
                     coef0=0.0, decision_function_shape='ovr', degree=3,
                     gamma='scale', kernel='rbf', max_iter=-1,
                     probability=False, random_state=None, shrinking=True,
                     tol=0.001, verbose=False))],
         verbose=False)
```

Linear Regression

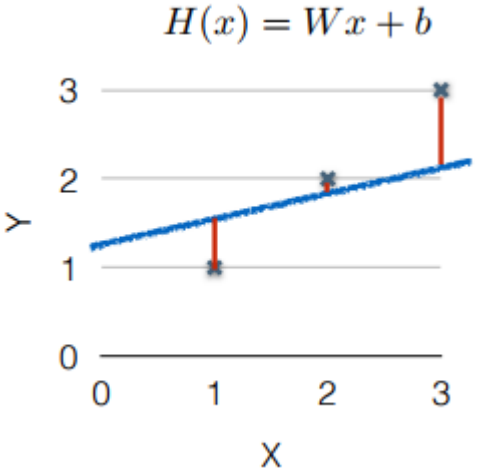
Regression이란?

- 회귀분석(Regression Analysis)
  - 2개 또는 그 이상 변수들의 의존관계를 파악함으로써 특정 변수(종속변수)의 값을 예측하는 통계학의 한 분야

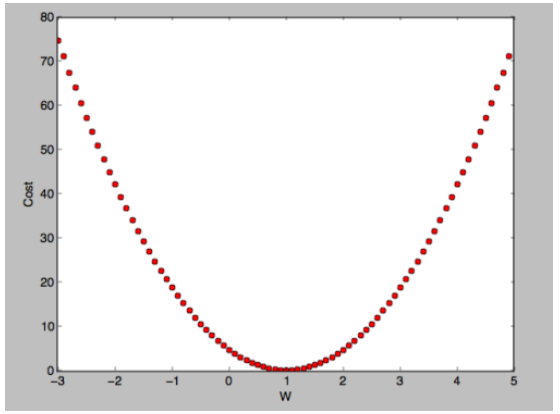
- Linear Regression Analysis(선형 회귀분석)
  - 두 변수 x, y에 대한 n개의 측정값 (x1, y1), (x2, y2), ···, (xn, yn)이 있을 때
  - 주어진 가설(hypothesis)에 대한 비용(cost)이 최소화 되도록 하는 직선을 찾는 문제

Linear Regression

- **Linear Hypothesis**
  - $H(x) = Wx + b$



- **Cost Function**
  - $cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2$
  - cost(W)함수의 모양

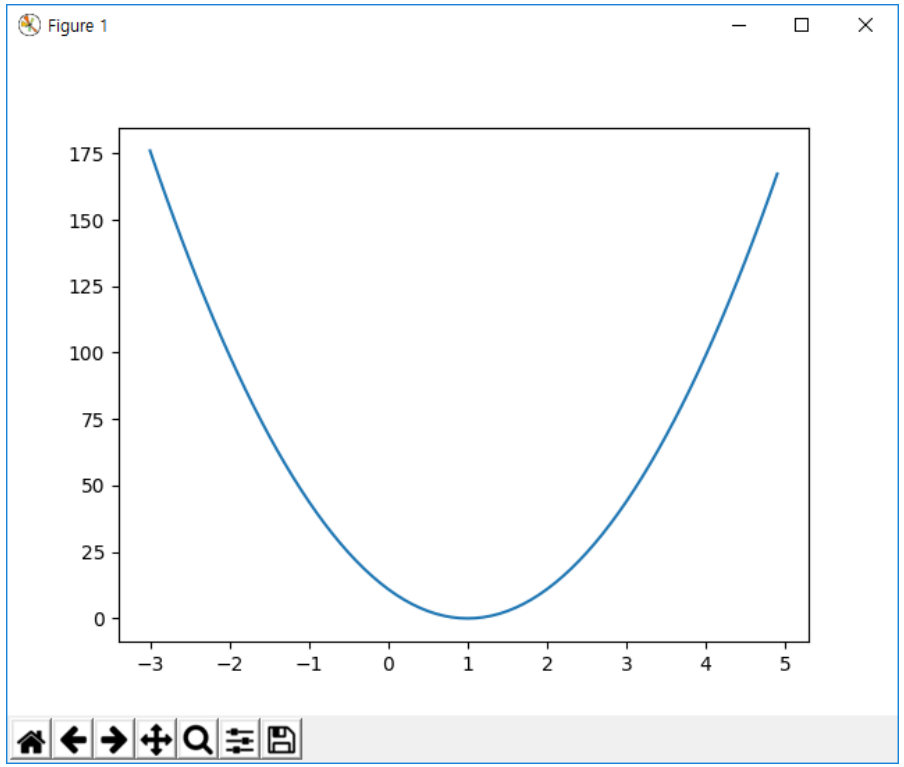


- **Linear Regression이란?**
  - cost함수 $cost(W, b)$ 를 최소화하는 W와 b를 찾는 문제

cost함수의 모양

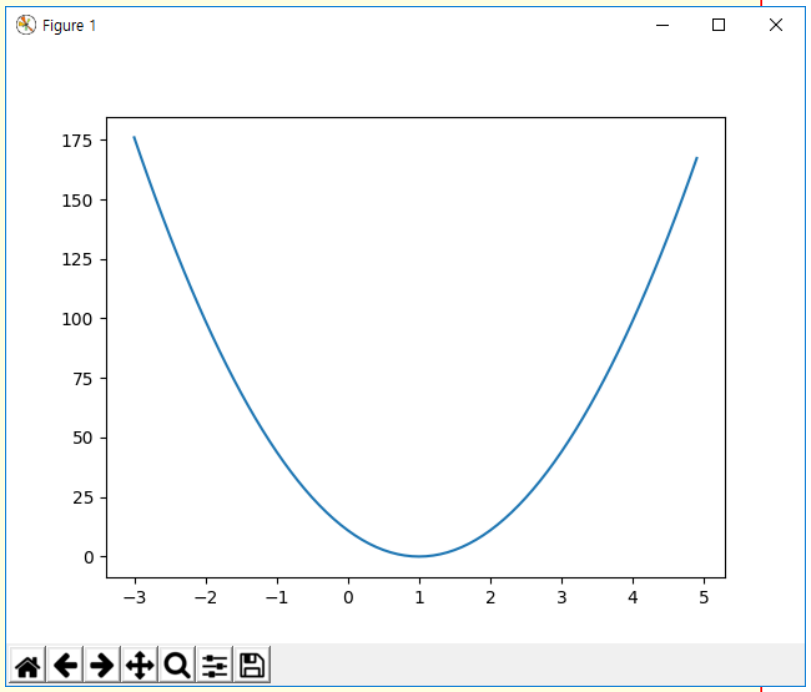
- cost함수를 단순화시켜서 생각해 보자!
  - $H(x) = Wx$
  - $cost(W) = \frac{1}{m}\sum_{i=1}^{m}\left(Wx^i - y^i\right)^2$
- 계산결과
  - x = {1, 2, 3, 4, 5}
  - y = {1, 2, 3, 4, 5}

```
w =  -3.0, c = 176.0
w =  -2.0, c =  99.0
w =  -1.0, c =  44.0
w =   0.0, c =  11.0
w =   1.0, c =   0.0
w =   2.0, c =  11.0
w =   3.0, c =  44.0
w =   4.0, c =  99.0
```

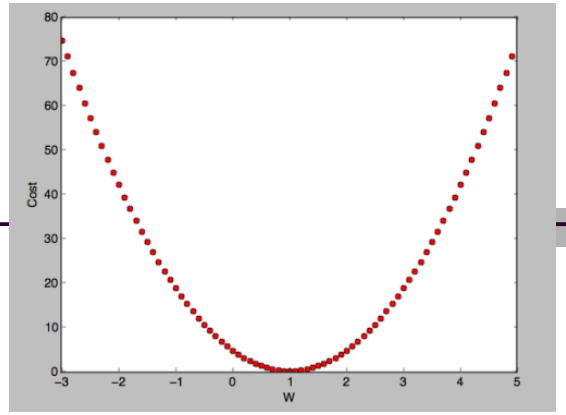


```python
import numpy as np
import matplotlib.pyplot as plt

X = np.array([1, 2, 3, 4, 5], dtype=np.float)
y = np.array([1, 2, 3, 4, 5], dtype=np.float)

def run(W):
    hypothesis = W * X
    cost = np.mean((hypothesis-y)**2)
    return cost

W_val = []
cost_val = []
for i in range(-30, 50):
    w = i * 0.1
    c = run(w)
    if i%10 == 0:
        print(f'w = {w:5.1f}, c = {c:5.1f}')
    W_val.append(w)
    cost_val.append(c)

plt.plot(W_val, cost_val)
plt.show()
```

```
w =  -3.0, c = 176.0
w =  -2.0, c =  99.0
w =  -1.0, c =  44.0
w =   0.0, c =  11.0
w =   1.0, c =   0.0
w =   2.0, c =  11.0
w =   3.0, c =  44.0
w =   4.0, c =  99.0
```



How to minimize cost?

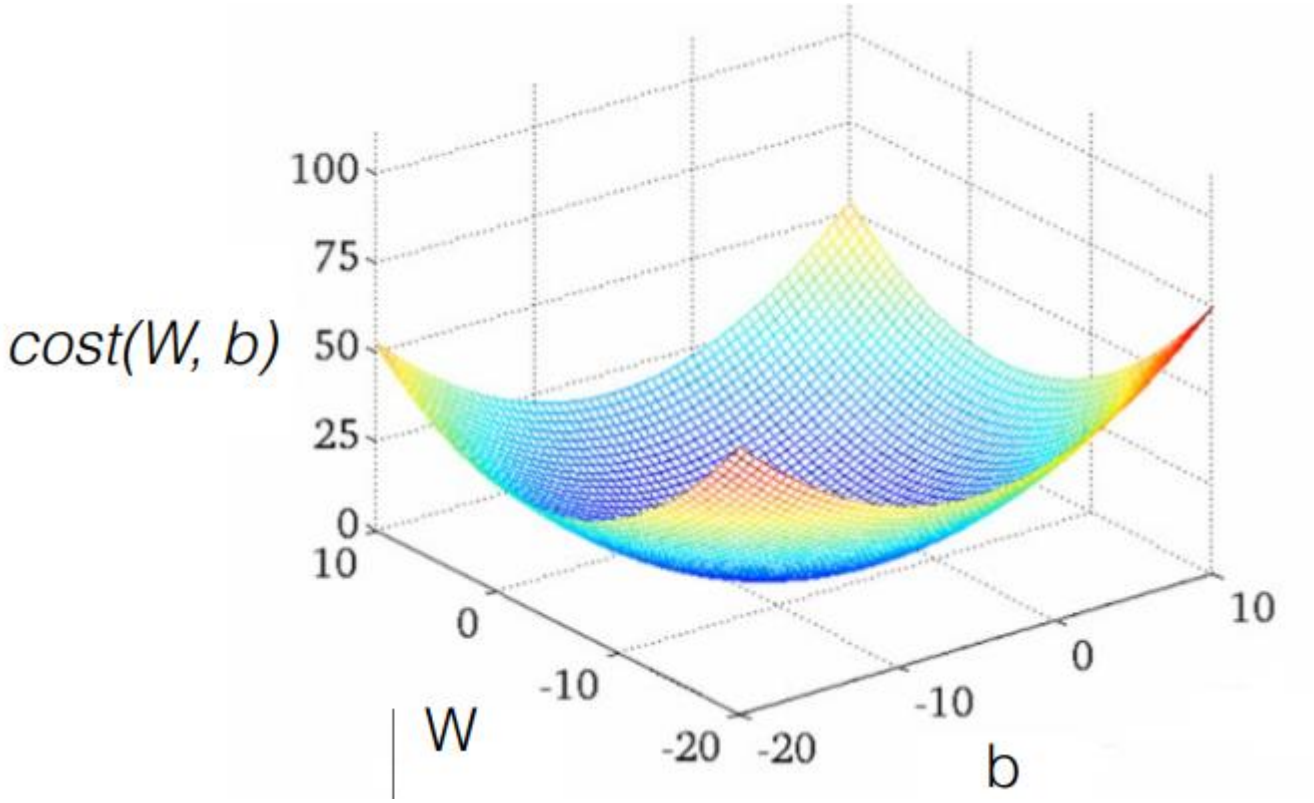


- 경사하강알고리즘을 이용
  - Gradient descent algorithm
    - 임의의 곳에서 시작하여 경사도(gradient)에 따라 W를 변경시켜가면서 cost함수의 값이 최소화되는 W를 구하는 알고리즘
  - 경사도(gradient)는 미분값
  - W값의 변화
    - $W = W - \alpha \frac{\partial}{\partial w} cost(W)$
    - α는 learning rate
- **Gradient descent algorithm**

$$W = W - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(W x^i - y^i \right) x^i$$

가중치와 편향이 고려된 경사하강법

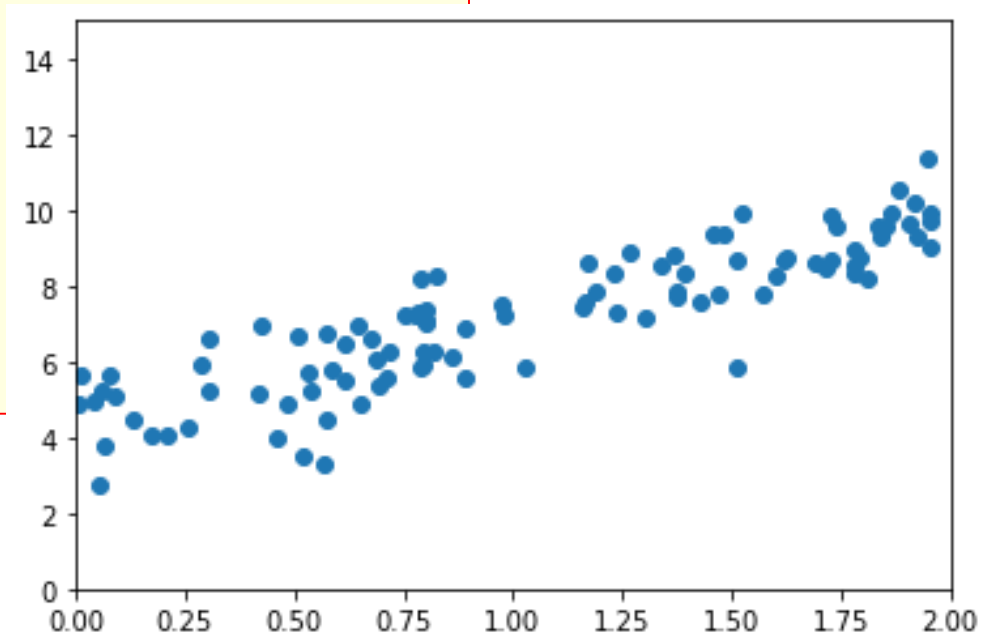
$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2$$



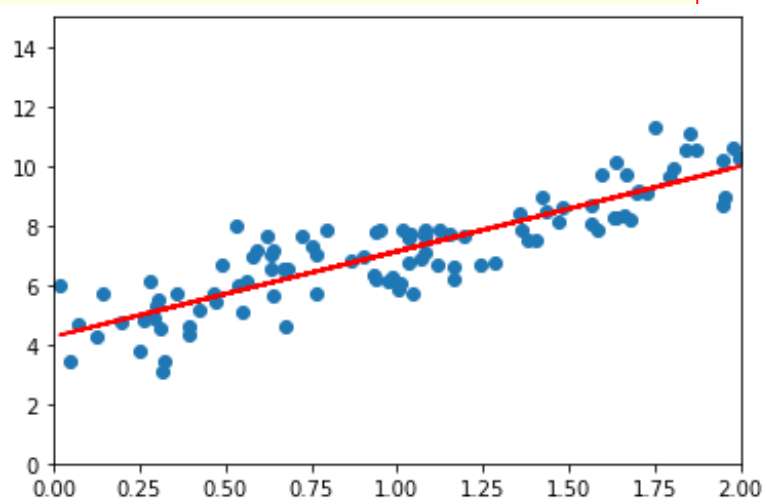
Linear Regression

- 임의의 데이터
  - 데이터가 선형회귀에 적절한지 확인

```
import numpy as np

# random data
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# plot
plt.scatter(X, y)
plt.axis([0, 2, 0, 15])
plt.show()
```



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

# random data
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# linear regression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
print(f"기울기 = {lin_reg.coef_}, y절편 = {lin_reg.intercept_}")

# plot
plt.scatter(X, y)
new_y = lin_reg.coef_ * X + lin_reg.intercept_
plt.plot(X, new_y, "r-")
plt.axis([0, 2, 0, 15])
plt.show()
```



규제가 추가된 선형 회귀

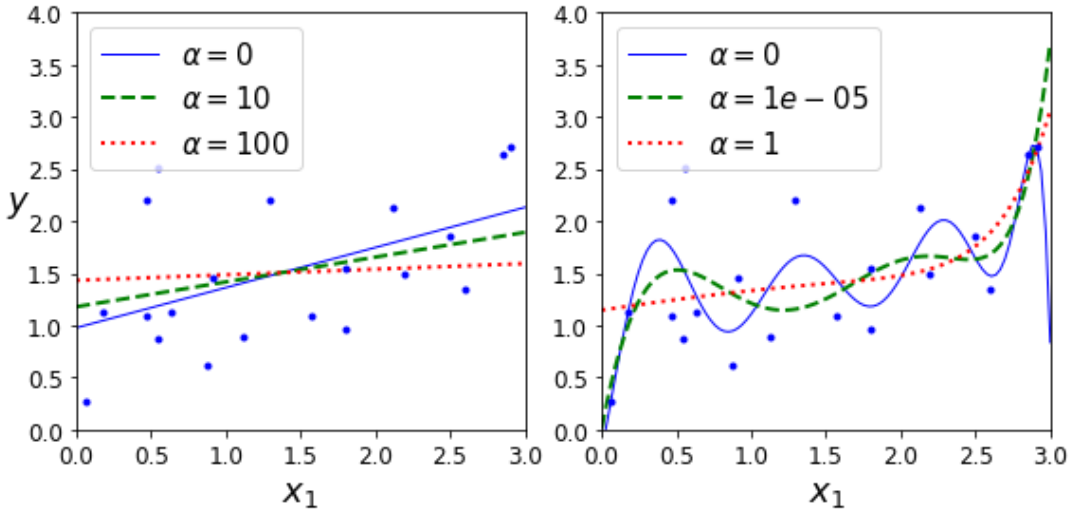
- 규제 : 과적합을 감소시키는 방법 중의 하나
- 선형회귀에서의 규제
  - 모델의 가중치를 제한하여 가중치가 가능한한 작게 유지되도록
- 종류
  - **Ridge** Regression(L2규제)

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

  - **Lasso** Regression(L1규제)

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

    - α가 0이면 선형회귀에 동일



회귀분석용 가상 데이터 만들기

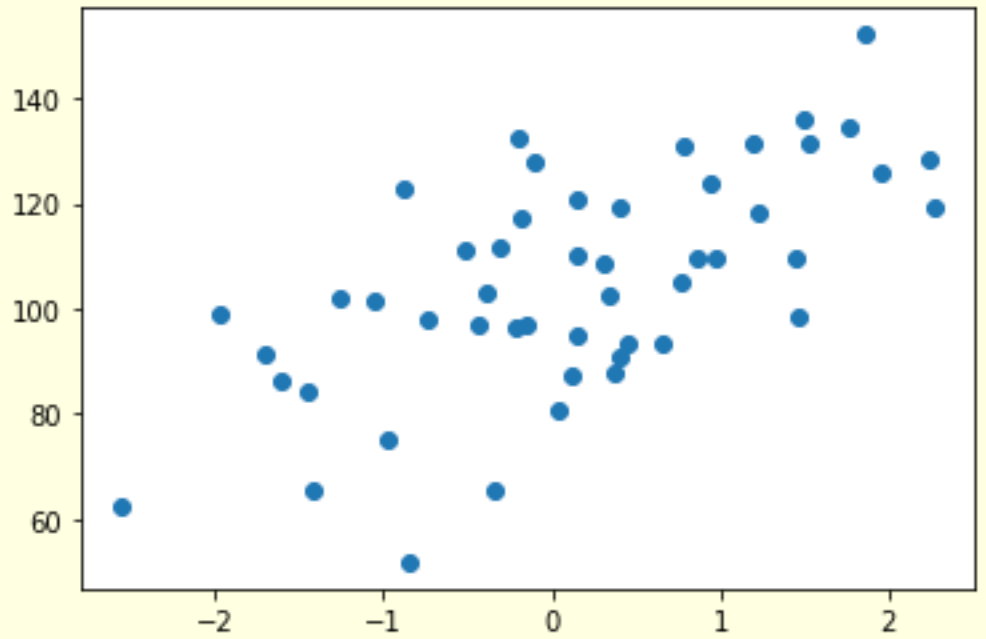
- **sckikit-learn에서는 회귀분석 데이터를 만들기 위한 함수를 제공**

```
X, y[,c] = make_regression(n_samples=100, n_features=100,
                           bias=0.0, noise=0.0,
                           coef=False, random_state=None)
```

  - n_samples : 데이터의 개수
  - n_features : 변수의 개수(차원)
  - bias : y-절편
  - niose : 잡음의 표준편차
  - coef : 계수(기울기) 출력여부(True이면 c를 반환)
  - random_state : 난수발생 시드값

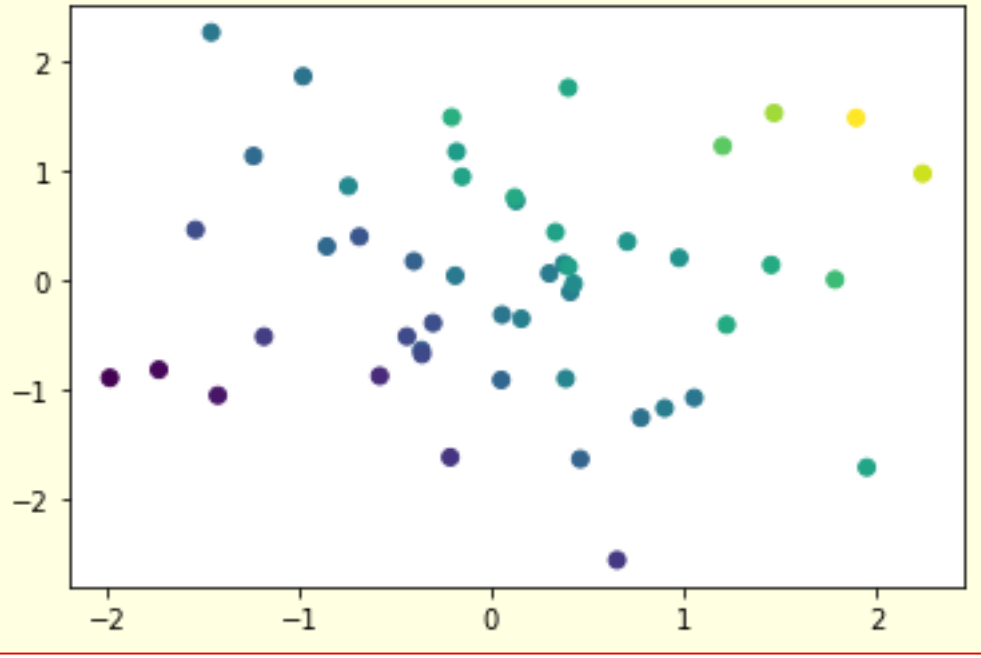
변수가 1개인 경우

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression

X, y, w = make_regression(
          n_samples=50,
          n_features=1,
          bias=100,
          noise=20,
          coef=True,
          random_state=0,
      )
plt.scatter(X, y)
plt.show()
```



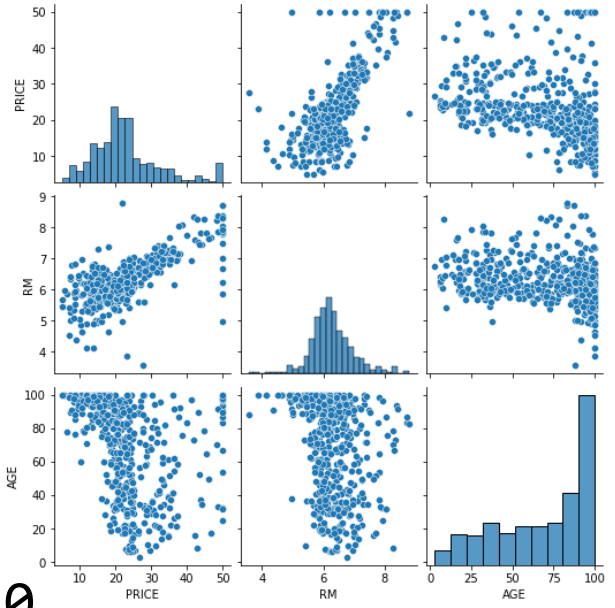
변수가 2개인 경우

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression

X, y, w = make_regression(
          n_samples=50,
          n_features=2,
          bias=100,
          noise=20,
          coef=True,
          random_state=0,
      )
plt.scatter(X, y)
plt.show()
```

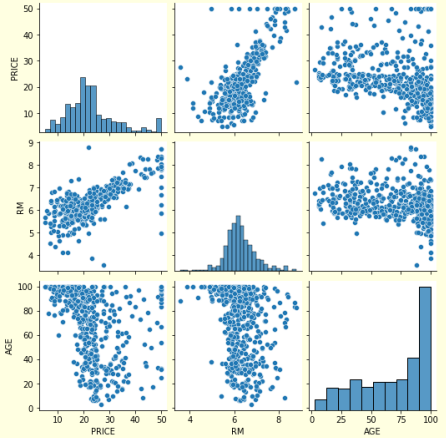


Boston 집값 예측용

- 보스턴의 506개 타운(town)의 13개 독립변수로 구성
  - CRIM: 범죄율
  - INDUS: 비소매상업지역 면적 비율
  - NOX: 일산화질소 농도
  - RM: 주택당 방 수
  - LSTAT: 인구 중 하위 계층 비율
  - B: 인구 중 흑인 비율
  - PTRATIO: 학생/교사 비율
  - ZN: 25,000 평방피트를 초과 거주지역 비율
  - CHAS: 찰스강의 경계에 위치한 경우는 1, 아니면 0
  - AGE: 1940년 이전에 건축된 주택의 비율
  - RAD: 방사형 고속도로까지의 거리
  - DIS: 직업센터의 거리
  - TAX: 재산세율



```
from sklearn.datasets import load_boston
import pandas as pd

boston = load_boston()
dfX = pd.DataFrame(boston.data, columns=boston.feature_names)
dfy = pd.DataFrame(boston.target, columns=["PRICE"])
df = pd.concat([dfX, dfy], axis=1)
df
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0					
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0					
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0					
...	...	...	...	...	...	...	...	...	...					
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0					
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0					
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0					
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0					
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0					

CRIM : 범죄율
ZN : 25,000 평방피트를 초과 거주지역 비율
INDUS : 비소매상업지역 면적 비율
CHAS : 찰스강의 경계에 위치한 경우는 1, 아니면 0
NOX : 일산화질소 농도
RM : 주택당 방 수
AGE : 1940년 이전에 건축된 주택의 비율
DIS : 직업센터의 거리
RAD : 방사형 고속도로까지의 거리
TAX : 재산세율
PTRATIO : 학생/교사 비율
B : 인구 중 흑인 비율
LSTAT : 인구 중 하위 계층 비율
PRICE : 집값

보스턴 주택 가격 예측(선형회귀)

```
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
from sklearn import metrics

boston = load_boston()
X = boston.data
y = boston.target
X_train, X_test, y_train, y_test
        = model_selection.train_test_split(X, y, test_size=0.3)

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

y_predict = lin_reg.predict(X_test)
score = metrics.r2_score(y_test, y_predict)
print(f"accuracy = {score}")
```

분류문제 : accuracy_score 사용
-> continuous is not supported
회귀문제 : r2_score 사용

```
accuracy = 0.702445591697509
```

보스턴 주택 가격 예측(Ridge 선형회귀)

```
from sklearn.datasets import load_boston
from sklearn.linear_model import Ridge
from sklearn import model_selection
from sklearn import metrics

boston = load_boston()
X = boston.data
y = boston.target
X_train, X_test, y_train, y_test
                        = model_selection.train_test_split(X, y, test_size=0.3)

lin_reg = Ridge(alpha=1.0)
lin_reg.fit(X_train, y_train)

y_predict = lin_reg.predict(X_test)
score = metrics.r2_score(y_test, y_predict)
print(f"accuracy = {score}")
```

```
accuracy = 0.7627894929117215
```

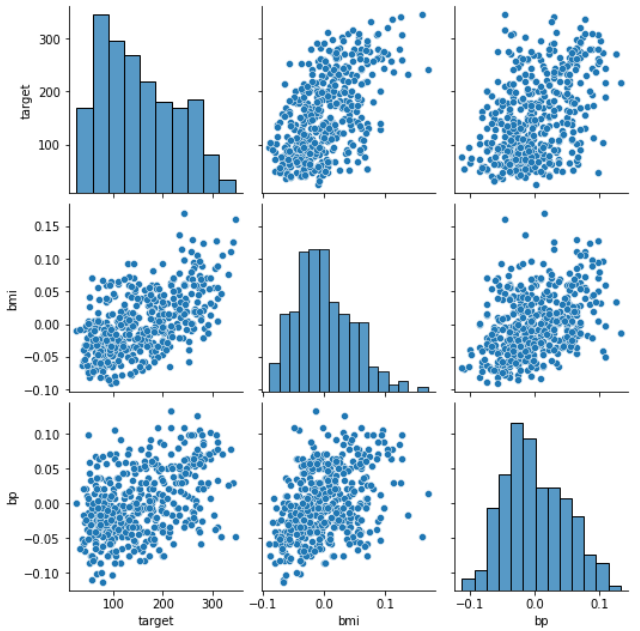
보스턴 주택 가격 예측(Lasso 선형회귀)

```
from sklearn.datasets import load_boston
from sklearn.linear_model import Lasso
from sklearn import model_selection
from sklearn import metrics

boston = load_boston()
X = boston.data
y = boston.target
X_train, X_test, y_train, y_test
                        = model_selection.train_test_split(X, y, test_size=0.3)

lin_reg = Lasso(alpha=1.0)
lin_reg.fit(X_train, y_train)

y_predict = lin_reg.predict(X_test)
score = metrics.r2_score(y_test, y_predict)
print(f"accuracy = {score}")
```

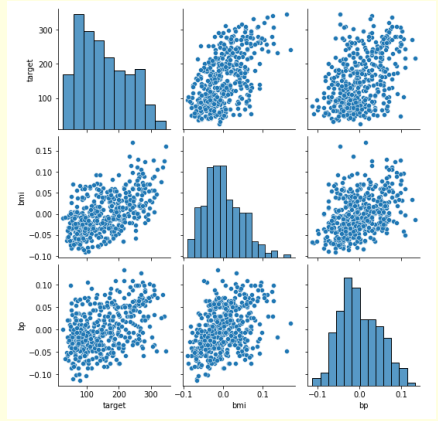
```
accuracy = 0.6872938932906716
```

당뇨병 진행도 예측용

- 442명의 당뇨병 환자를 대상으로 한 검사 결과를 나타내는 데이터
- 모든 데이터는 스케일링 되었음
  - age: 나이
  - sex: 성별
  - bmi: BMI(Body mass index)지수
  - bp: 평균혈압
  - s1~s6: 6종류의 혈액검사수치
- 종속변수는 1년 뒤에 측정한 당뇨병의 진행률



```python
from sklearn.datasets import load_diabetes
import pandas as pd
import seaborn as sns

diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df["target"] = diabetes.target
df
```



	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0
...	...	...	...	...	...	...	...	...	...	...	...
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207	178.0
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018118	0.044485	104.0
439	0.041708	0.050680	-0.015906	0.017282	-0.037344	-0.013840	-0.024993	-0.			
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.			
441	-0.045472	-0.044642	-0.073030	-0.081414	0.083740	0.027809	0.173816	-0.			

age	: 나이
sex	: 성별
bmi	: BMI(Body mass index)지수
bp	: 평균혈압
s1~s6	: 6종류의 혈액검사수치

프로젝트

- 당뇨병 진행도 예측

여러분이 해 보세요!