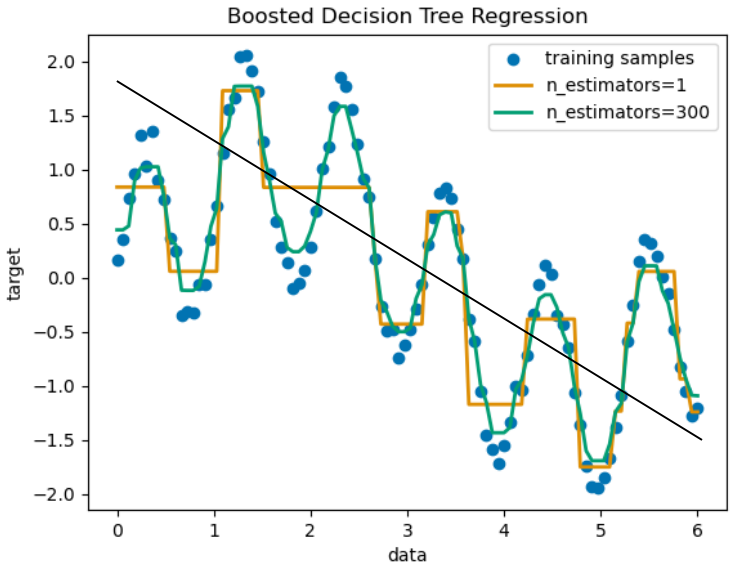
# 제10장. 회귀(回歸, regression)

- 단순/다중 회귀 분석(Simple & Multiple Linear Regression)
- 로지스틱 회귀분석(Logistic Regression)



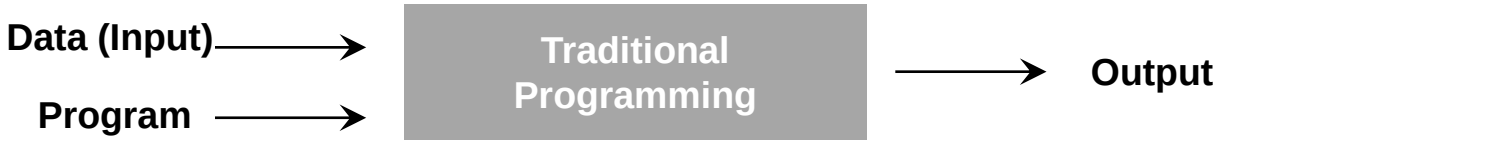
# Study Point

- 데이터, 알고리즘, 모형을 이해하고 scikit-learn 머신 러닝을 이해한다.
- 단순/다중 선형회귀 분석과 로지스틱 회귀 분석을 이해하고 실습한다.
- 이항분류와 다항분류를 이해한다.

# 머신러닝

컴퓨터에게 데이터에 존재하는 규칙성(regularities)을 발견하게 하여 새로운 데이터의 예측을 컴퓨터가 스스로 하게 하는 것 (사람이 규칙을 찾아서 문제를 해결하는 방법은 이제는 너무 어려움)

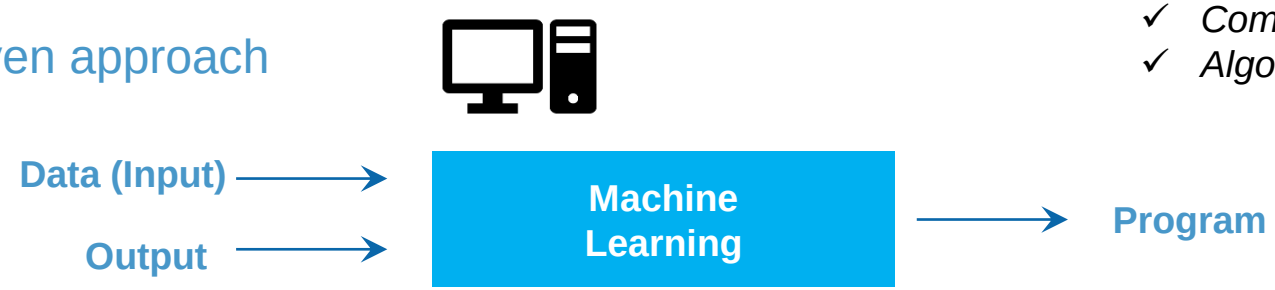
- Rule based approach




- ✓ *Big data*
- ✓ *High dimensionality*

- ✓ *Computing power*
- ✓ *Algorithm*

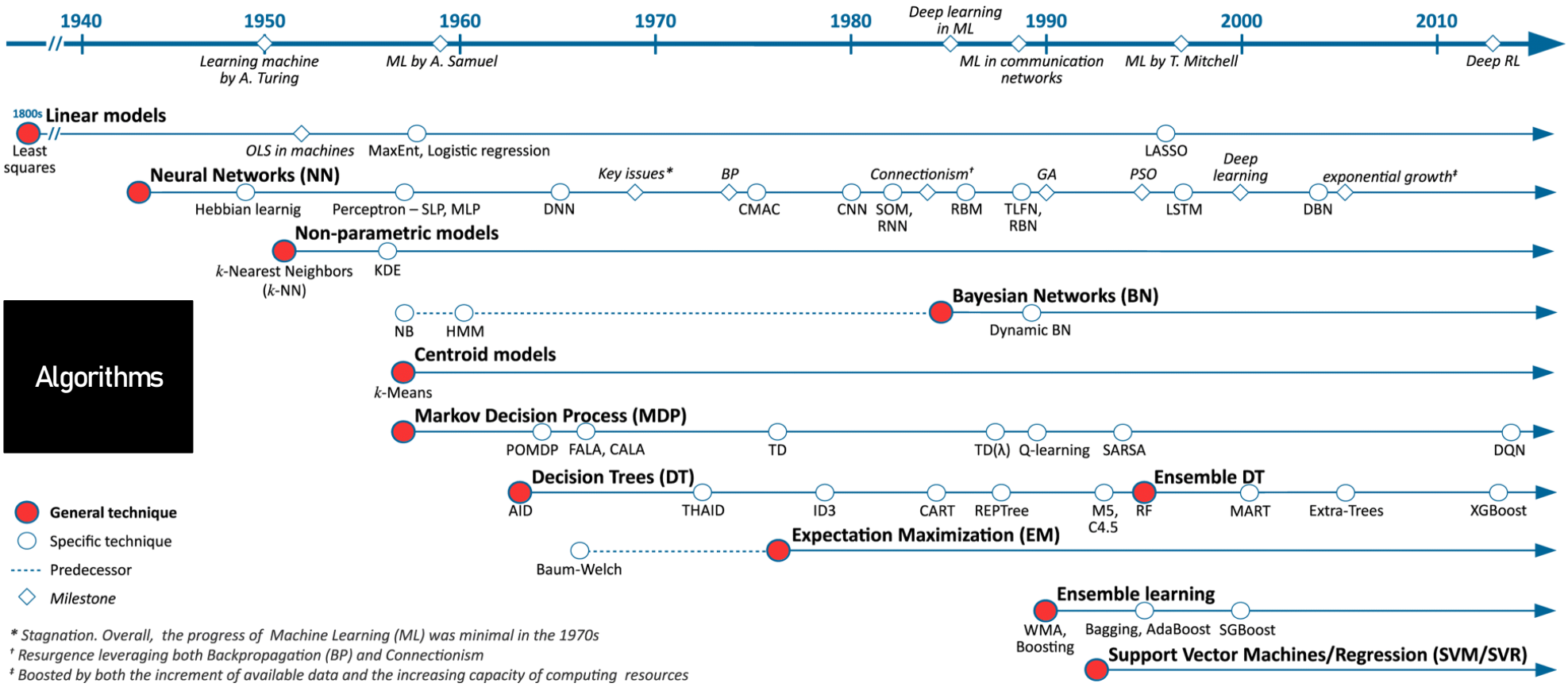
- Data driven approach



# 단순 선형 회귀생성

X가 얼만큼 변하면 Y가 어느 정도 변하는지를 추정하기 위해 y = a + bX의 선형식을 가정하고 데이터를 가장 그럴듯하게 나타내 줄 수 있는 기울기와 절편을 찾는 것이 목적

# 인공지능 알고리즘 출현 연대기

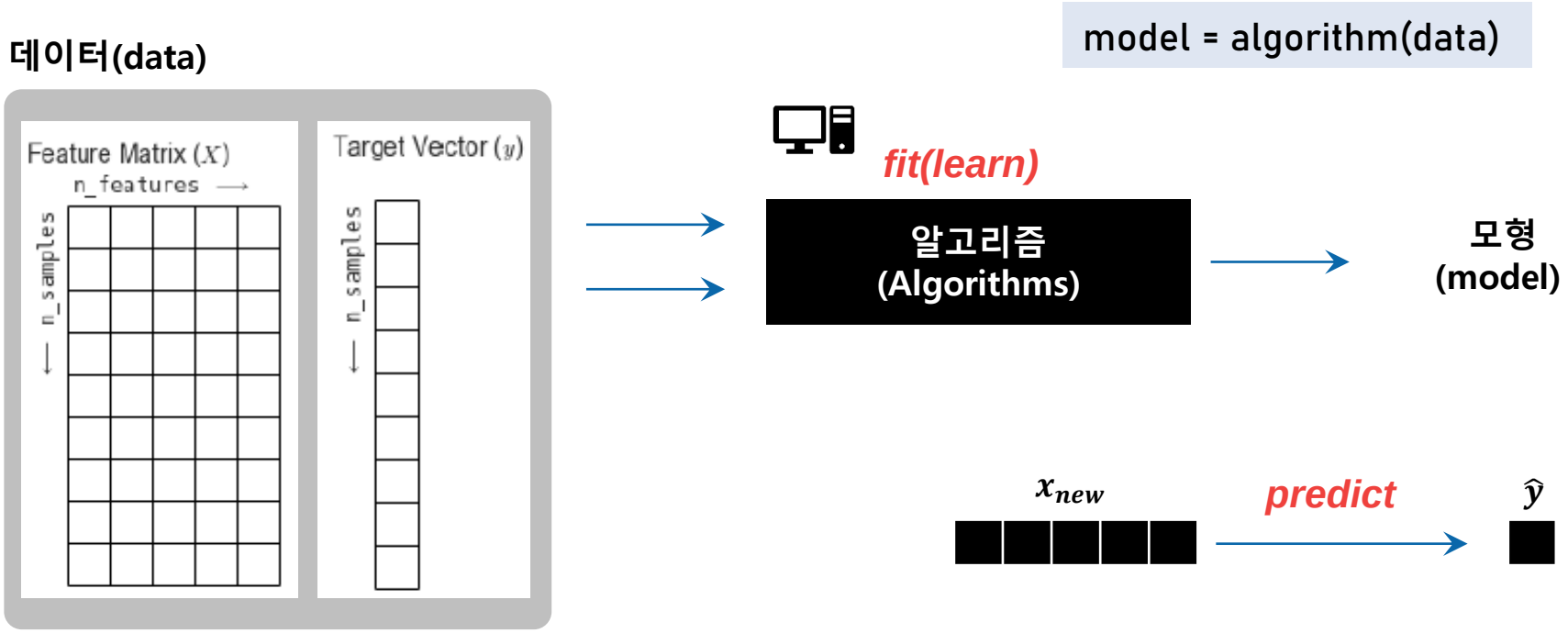



* Stagnation. Overall, the progress of Machine Learning (ML) was minimal in the 1970s
† Resurgence leveraging both Backpropagation (BP) and Connectionism
‡ Boosted by both the increment of available data and the increasing capacity of computing resources

https://towardsdatascience.com/interpreting-machine-learning-model-70fa49d20af1

# 데이터, 알고리즘, 모형

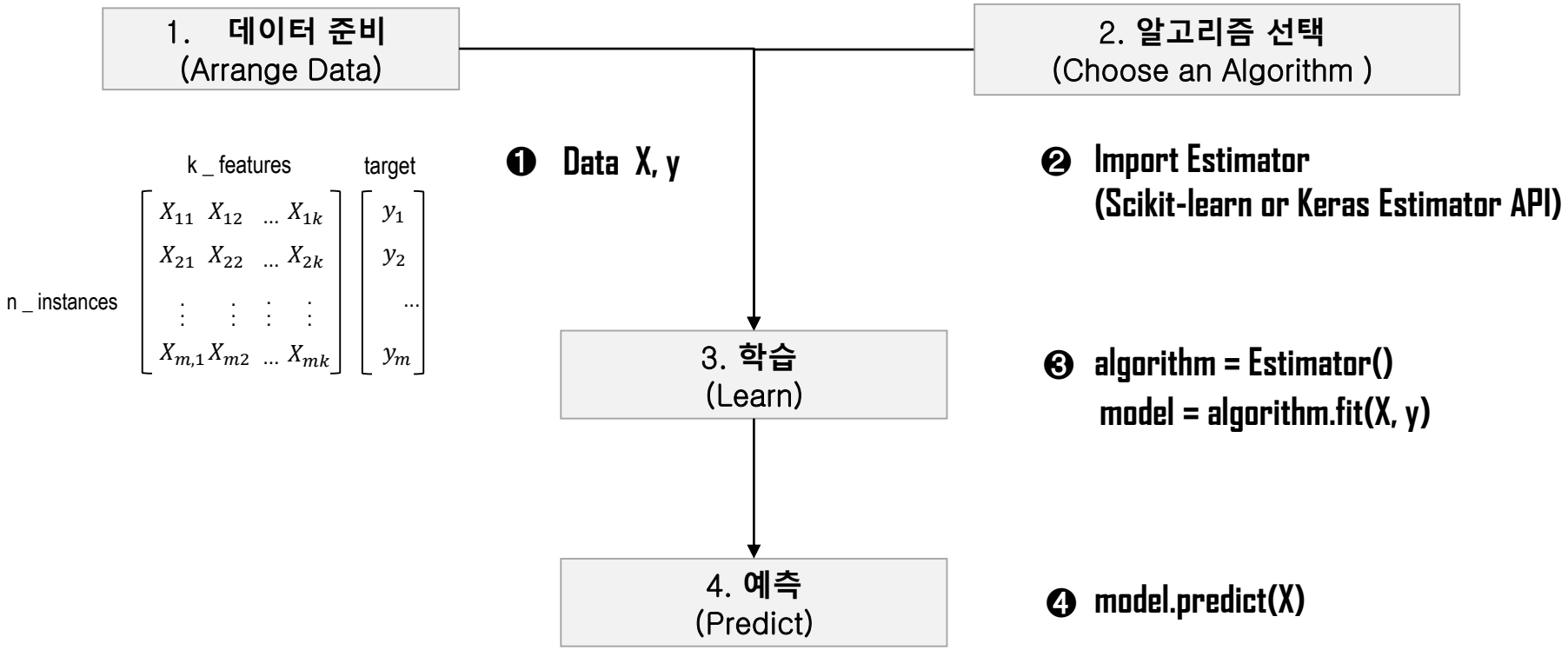
머신러닝의 목적은 과거 데이터(data)에 숨어있는 패턴을 학습하여(learn, fit) 예측하는 모형(model)을 발견하여 새로운 데이터를 예측하는 것



# 머신러닝 메커니즘(작동방식)

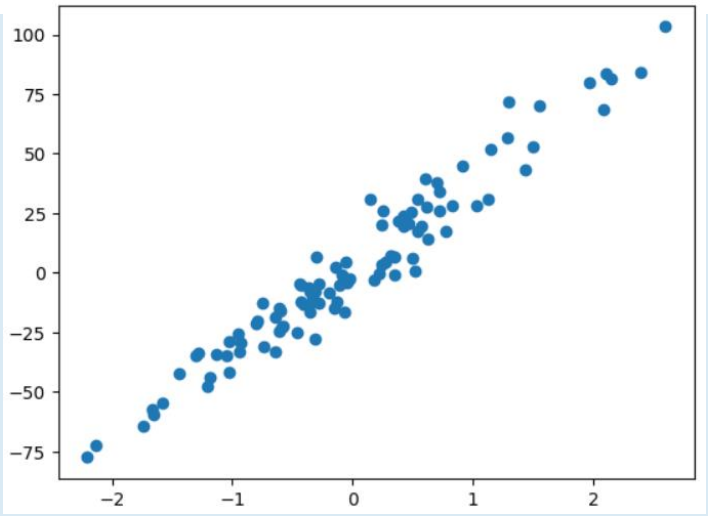
```
X = [[1,2,3], [11,12,13]]; y=[0,1]
```

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X,y)
clf.predict([[4,5,6]])
```

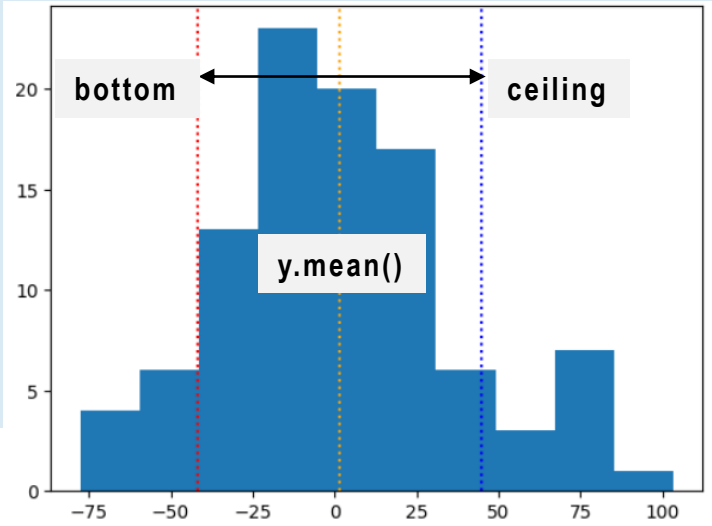


# 회귀생성

```
# Target이 실수형 인 자료
from sklearn.datasets import make_regression
X, y = make_regression(n_features=1, noise=10,
                       random_state=111)
import matplotlib.pyplot as plt
plt.scatter(X, y)
```




```
# X(특성변수)가 없다면 y를 예측하는 방법은 평균과 표준편차
y.mean()
bottom = y.mean() - 1.2*y.std()
ceiling = y.mean() + 1.2*y.std()

plt.hist(y)                                  # Histogram
plt.axvline(y.mean(), 0, 15, color='orange', linestyle='dotted')
plt.axvline(bottom, 0, 15, color='red', linestyle='dotted')
plt.axvline(ceiling, 0, 15, color='blue', linestyle='dotted')
```

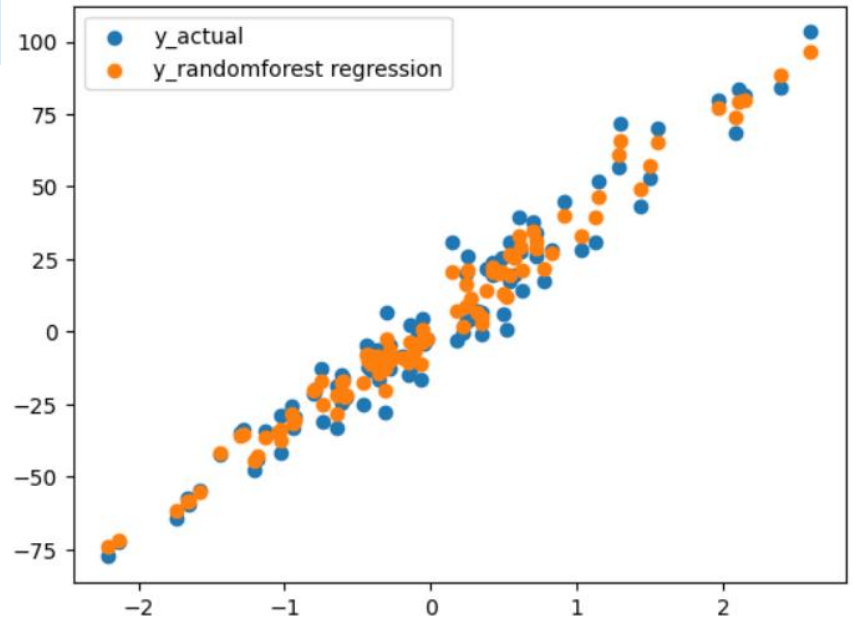


# 회귀생성

```
# 예측 1 (randomforest 회귀생성)
from sklearn.ensemble import RandomForestRegressor
regr_rf = RandomForestRegressor()
regr_rf.fit(X, y)
regr_rf.predict([[0]])
```

```
array([-2.63552691])
```

```
y_predict_rf = regr_rf.predict(X)
plt.scatter(X, y, label='y_actual')
plt.scatter(X, y_predict_rf, label='y_randomforest regression')
plt.legend()
```

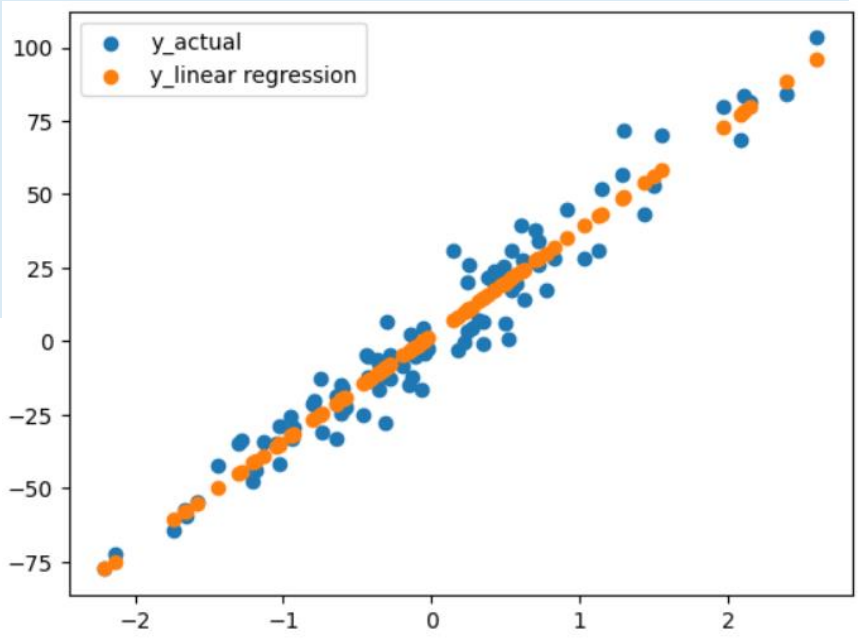


# 회귀생성

```
# 예측 2 (단순선형회귀분석)

from sklearn.linear_model import LinearRegression
regr_lr = LinearRegression().fit(X, y)
y_predict_lr = reg_lr.predict(X)
regr_lr.predict([[0]])
```

```
array([2.08555217])
```

```
plt.scatter(X, y, label='y_actual')
plt.scatter(X, y_predict_lr, label='y_linear regression')
plt.legend()
```



# 회귀생성 평가 지표

```
# MAE(평균절대오차)
# 실제 값과 예측 값의 차이(Error)를 절대값으로 변환해 평균화
# 이상치가 많을 때
from sklearn.metrics import mean_absolute_error
print('MAE_rf',mean_absolute_error(y, y_predict_rf))
print('MAE_lr',mean_absolute_error(y, y_predict_lr))
```

```
MAE_rf 3.319178718972687
MAE_lr 6.864340099043651
```

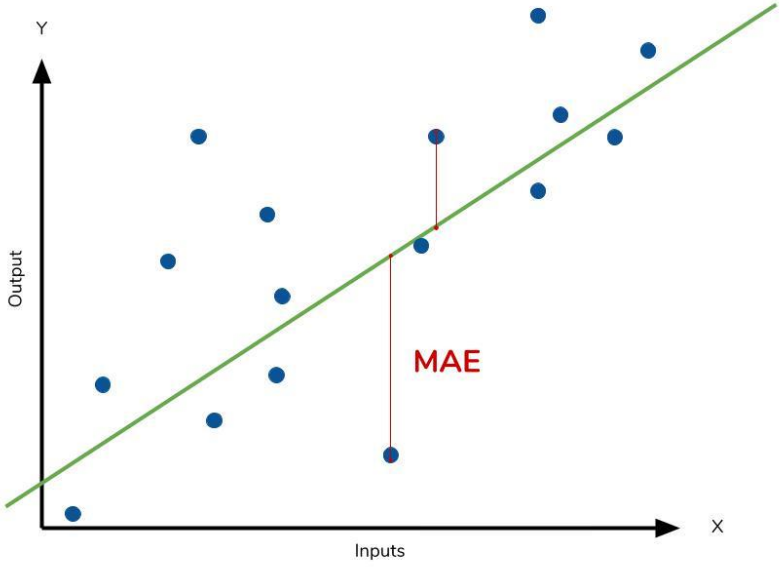
```
# MSE(평균제곱오차)
# 실제 값과 예측 값의 차이를 제곱해 평균화
# 특이값이 존재하면 수치가 많이 늘어나 이상치에 민감하다
from sklearn.metrics import mean_squared_error
print('MSE_rf',mean_squared_error(y, y_predict_rf))
print('MSE_lr',mean_squared_error(y, y_predict_lr))
```

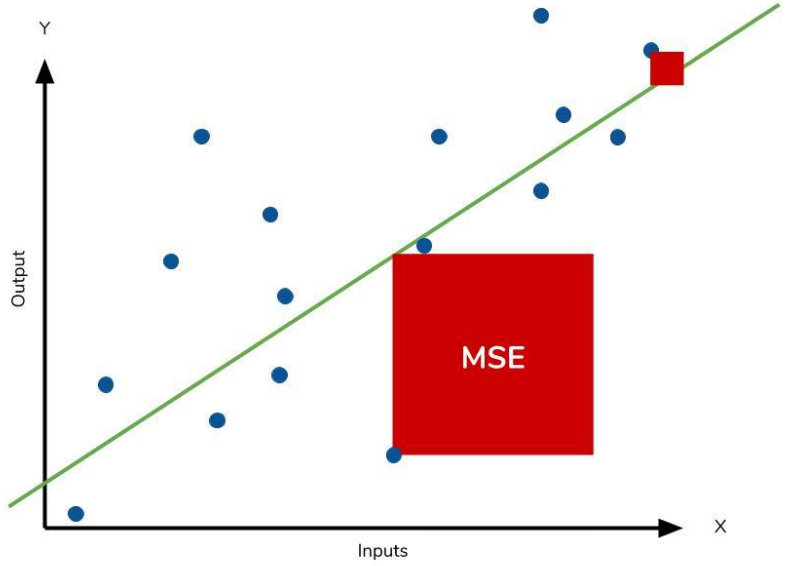
```
MSE_rf 16.696715708459532
MSE_lr 71.6356778212634
```

# 회귀생성 평가 지표_기하학적 의미

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

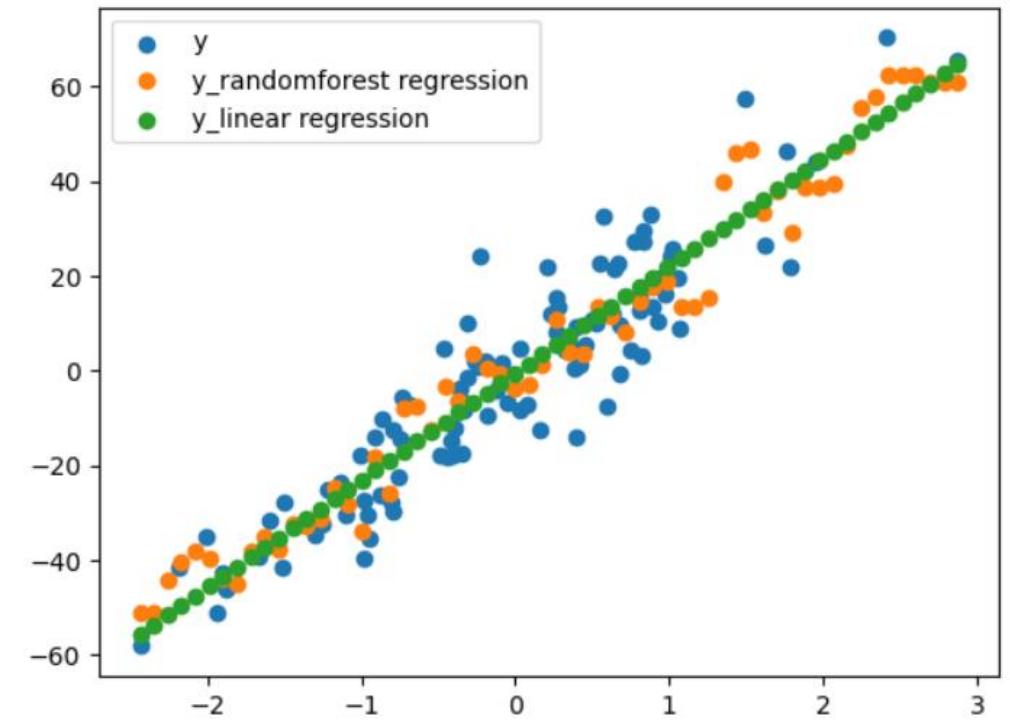
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$



# 회귀생성 시각화

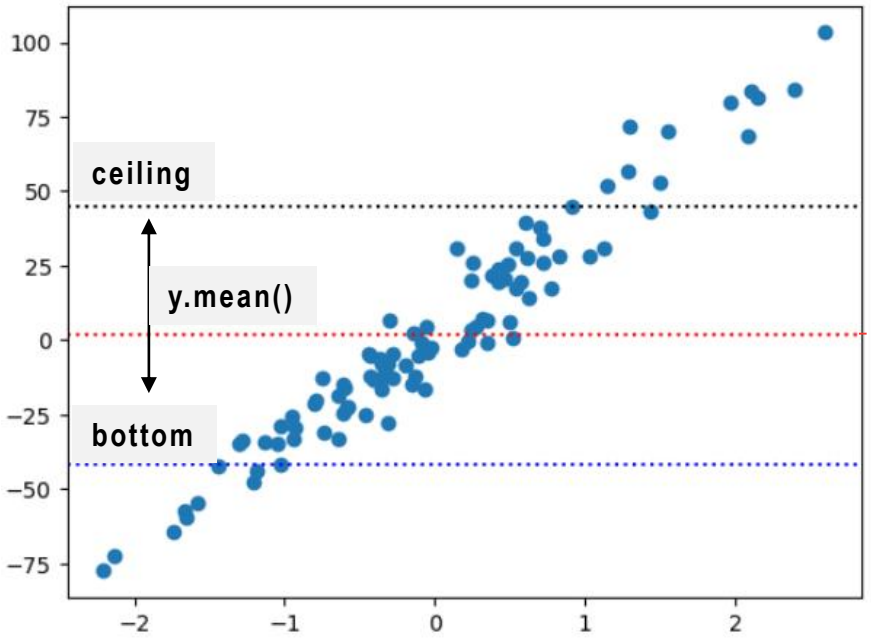
```
# 예측 1( 랜덤포레스트)과 예측 2(선형회귀)의 시각화

import numpy
x = numpy.linspace(X.min(), X.max(), 60)          # 새로운 X 변수
y_predict_rf = regr_rf.predict(x.reshape(-1,1))
y_predict_lr = regl_lr.predict(x.reshape(-1,1))

plt.scatter(X, y, label='y')
plt.scatter(x, y_predict_rf,
            label='y_randomforest regression')
plt.scatter(x, y_predict_lr,
            label='y_linear regression')
plt.legend()
```



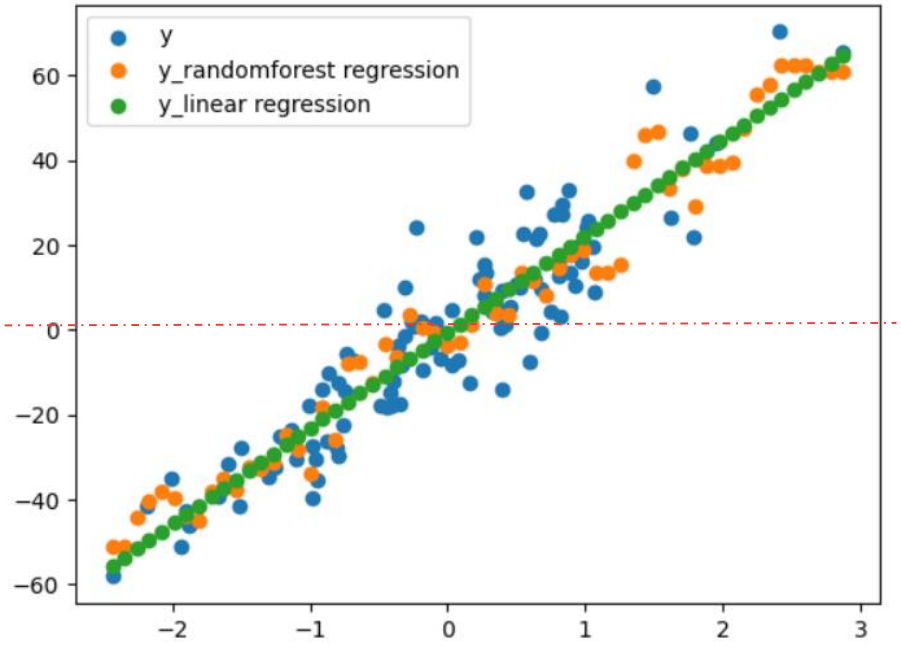
# 결정계수(coefficient of determination)

```
# y의 평균과 표준편차로만 예측
plt.scatter(X, y, label='y')
plt.axhline(y.mean(), -2, 2, color='red', linestyle='dotted')
plt.axhline(ceiling, -2, 2, color='black', linestyle='dotted')
plt.axhline(bottom, -2, 2, color='blue', linestyle='dotted')
```

y로만 y를 예측

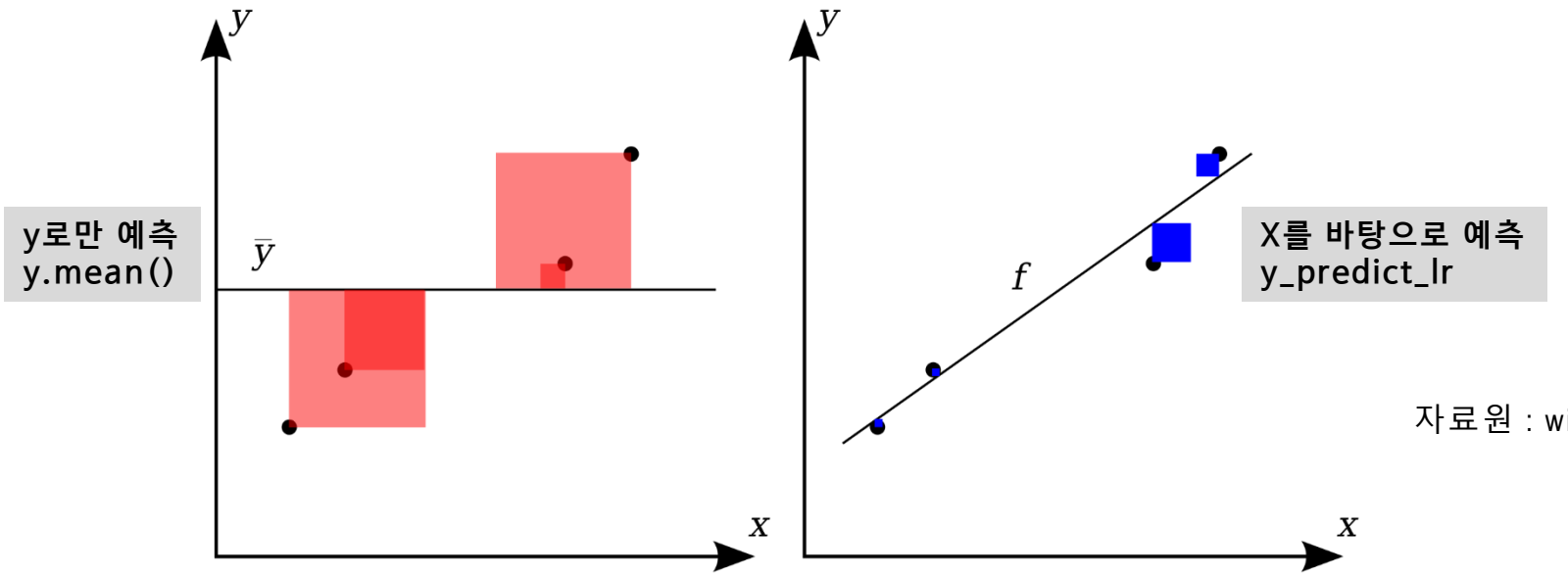



X를 바탕으로 y를 예측



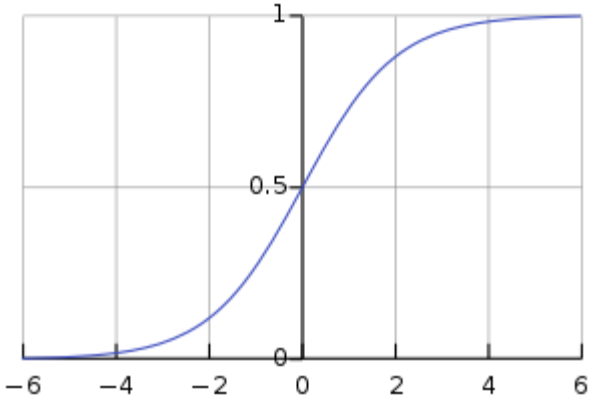
# 결정계수(coefficient of determination)

- 데이터에 얼마나 Fitting을 잘하였는지 보여주는 성능 지표로 추정한 선형 모형이 주어진 자료에 적합한 정도를 재는 척도
- 결정계수의 값은 0에서 1사이에 있으며, 종속변인(y)과 독립변인(X) 사이에 상관관계(correlation)가 높을수록 1에 가까워짐
- 즉, 결정계수가 0에 가까운 값을 가지는 선형회귀모형은 유용성이 낮은 반면, 결정계수의 값이 클수록 회귀모형의 유용성이 높다고 할 수 있음

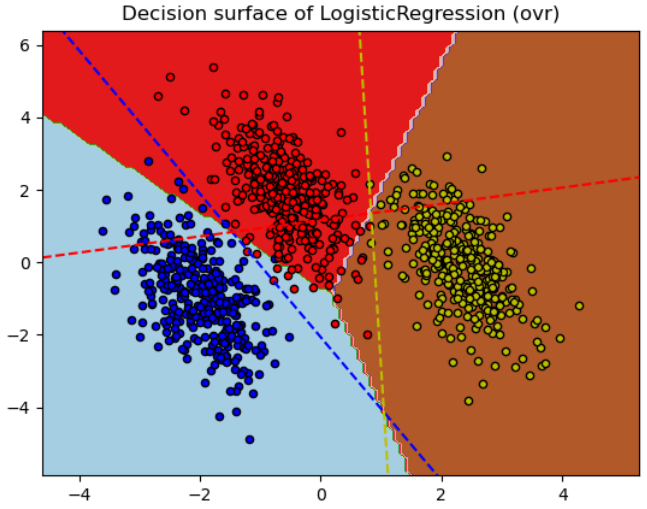



자료원 : wiki

- 단순/다중 회귀 분석(Simple & Multiple Linear Regression)
- 로지스틱 회귀분석(Logistic Regression)




$$S(x) = \frac{1}{1 + e^{-x}}$$



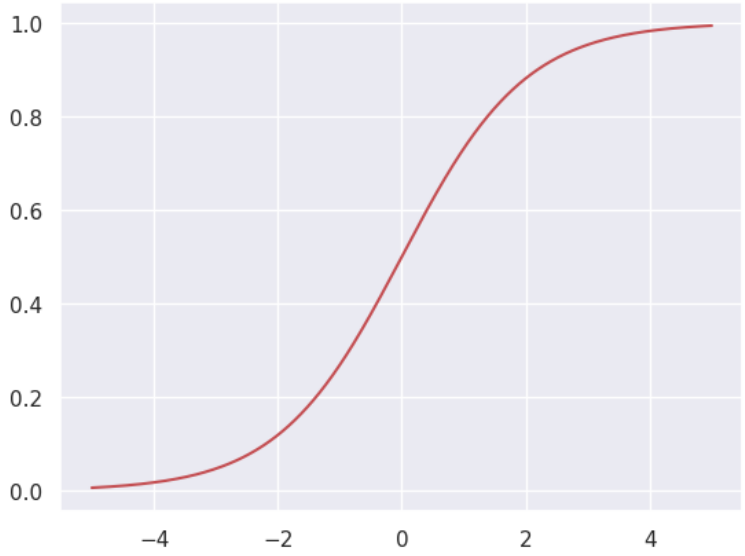
# 시그모이드 함수

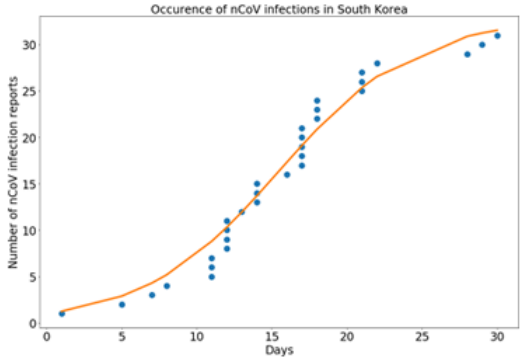
- 전염병 확산, 기술 확산, 마케팅 구전효과 등 확산 곡선(diffusion curve)형태는 Sigmoid Function 형태를 갖는다.

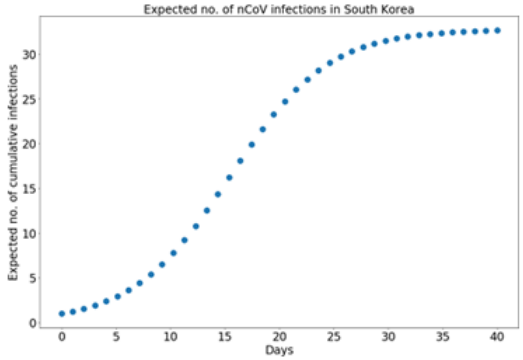
$$\text{Sigmoid}(z) = \text{Logitstic}(z) = \frac{1}{1+\exp(-z)}$$

$$z = a_0 + a_1 x_1 + a_2 x_2 + \cdots$$

```
# sigmoid Function
xx = np.linspace(-5, 5, 1000)
plt.plot(xx, 1/(1+np.exp(-xx)), 'r-', label='Sigmoid Function')
```



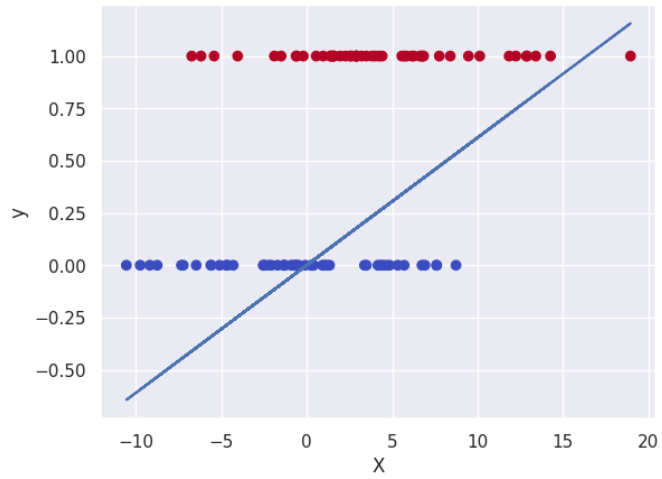



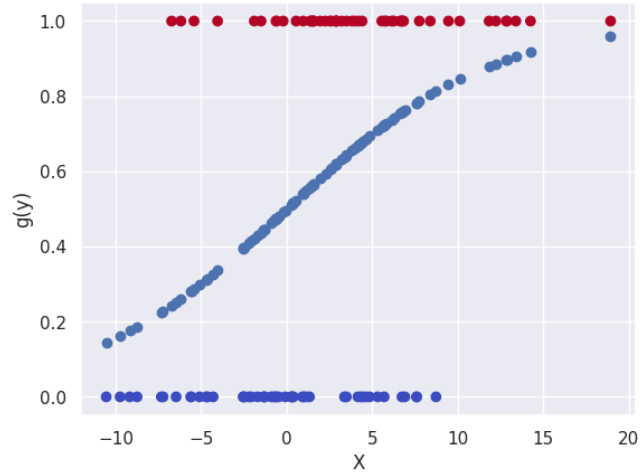




신종코로나 바이러스 확산추이 및 감염자 예측을 위한 수학적 모델링

# 로지스틱 회귀분석

```
from sklearn.datasets import make_blobs
import statsmodels.api as sm
X, y = make_blobs(100, 1, centers=2, random_state=14, cluster_std=5)  # 가상 데이터
y_predict = sm.OLS(y, X).fit().predict(X)                             #  단순선형 회귀분석
plt.scatter(X[:, 0], y, c=y, cmap='coolwarm')
plt.plot(X, y_predict); plt.ylabel('y'); plt.xlabel('X'); plt.show()

y_predict = sm.Logit(y, X).fit().predict(X)                           # 로지스틱회귀분석
plt.scatter(X[:, 0], y, c=y, cmap='coolwarm')
plt.scatter(X, y_predict); plt.ylabel('g(y)'); plt.xlabel('X');
```



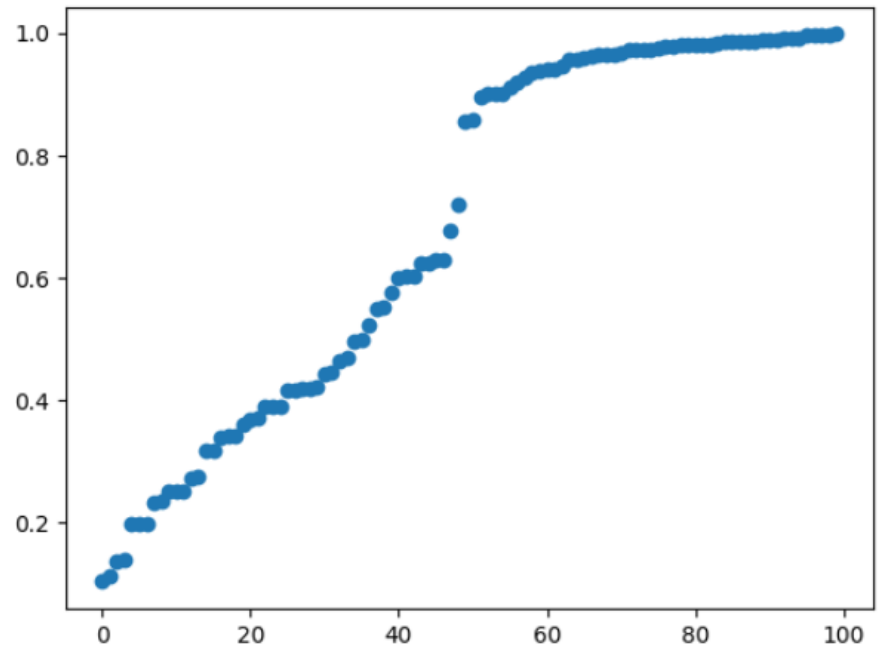



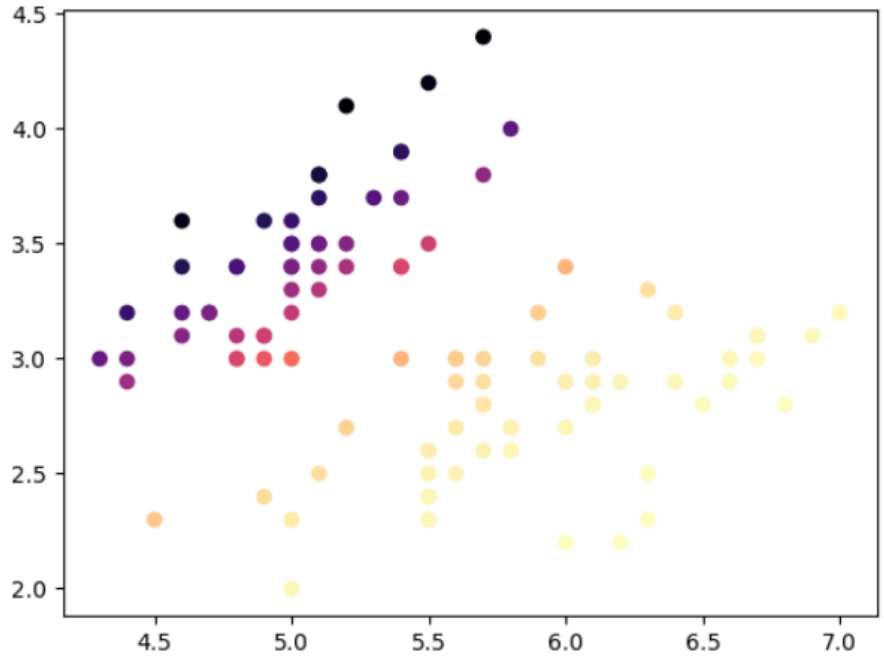
# Logistic regression – Iris binary classification

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data[:, :2]  # 2차원 그림 그리려고 2개 변수만
y = iris.target
# iris 이름이 setosa and versicolor 인 개만 이항분류를 위해
X = X[(y == 0) | (y == 1)]
y = y[(y == 0) | (y == 1)]
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='winter')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression(random_state=11, solver='saga')
lr.fit(X_train, y_train)
lr.coef_

# Sigmoid 함수의 값은 X(100,2)와 회귀식의 계수의 내적
z = X@lr.coef_.reshape(2,1)
lgr = 1 / (1 + np.exp(-z))                    # type(lgr)
```

# Logistic regression – Iris binary classification

```
np.sort(lgr.flatten())                    # 2차원 그림을 그리기 위해 순서 정열
plt.scatter(np.arange(len(lgr)), np.sort(lgr.flatten()), cmap='winter')

plt.scatter(X[:, 0], X[:, 1], c=lgr, cmap='magma')
# 채도가 낮을 수록 예측 확률의 값이 낮음
```

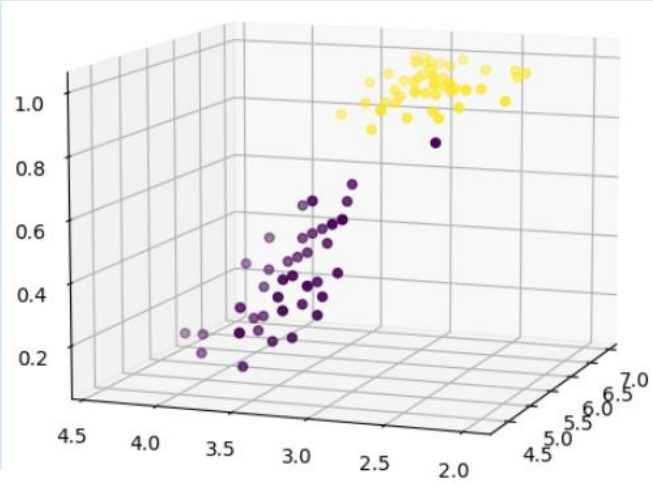


# Logistic regression – Iris binary classification

```
# 편향(bias)을 발생시키면서 컬럼 1개 추가
n = X.shape[0]
X_with_bias = np.hstack((np.ones((n, 1)), X))
X_with_bias[:10]

X_train, X_test, y_train, y_test = train_test_split(X_with_bias, y, random_sta
lr.fit(X_train, y_train)
lr.coef_
X_with_bias.shape
lr.coef_.reshape(3,1).shape
z = X_with_bias@lr.coef_.reshape(3,1)
1 / (1 + np.exp(-z))

# 3차원으로 그리기
from mpl_toolkits import mplot3d
plt.figure(figsize=(8,6))
ax = plt.axes(projection='3d')
ax.scatter3D(X[:, 0], X[:,1],1 / (1 + np.exp(-z)), c=y)
ax.view_init(10, 200)
```

```
array([[1. , 5.1, 3.5],
       [1. , 4.9, 3. ],
       [1. , 4.7, 3.2],
       [1. , 4.6, 3.1],
       [1. , 5. , 3.6],
       [1. , 5.4, 3.9],
       [1. , 4.6, 3.4],
       [1. , 5. , 3.4],
       [1. , 4.4, 2.9],
       [1. , 4.9, 3.1]])
```



# Logistic regression – Titanic binary classification

```
# Logistic regression
import seaborn as sns
df = sns.load_dataset('titanic')
y = df['survived']
X  = df.drop(columns= ['pclass', 'who', 'adult_male', 'who', 'adult_male', 'deck', 'embark_town', 'alive',
'alone'])

# 결측치 대체
X.isnull().sum()
X['age'] = X['age'].fillna(X['age'].mean())
X['embarked'] = X['embarked'].fillna(X['embarked'].mode()[0])  # X['embarked'].mode()[0]는 str

# dtypes확인하고 변경
X.info()
for col in X.columns[[1, 6]]:
    X[col] = X[col].astype('category')
X.info()

cats = [col for col in X if X[col].dtypes =='category']    # 범주형변수만 cats로 저장
```

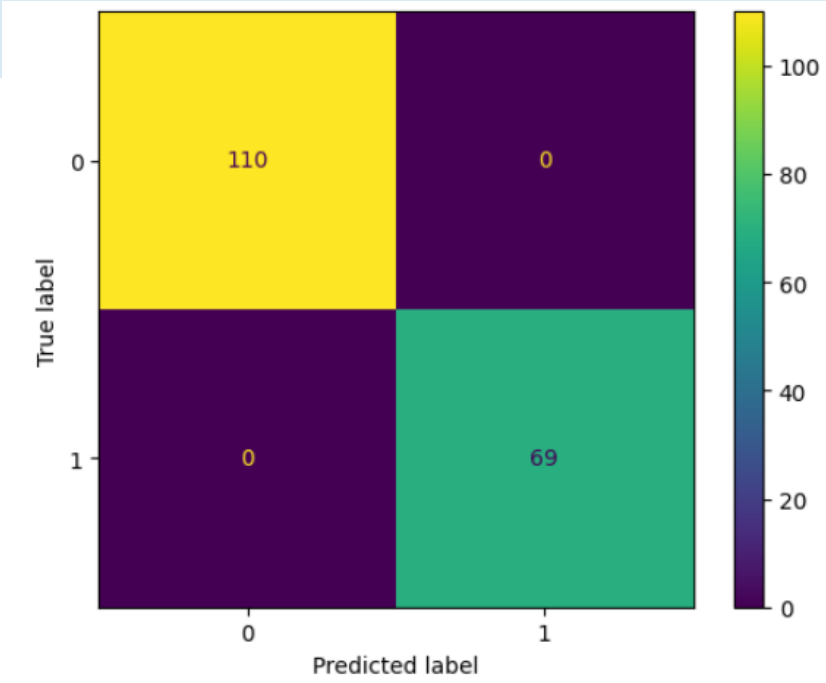
# Logistic regression – Titanic binary classification

```
# 범주형 변수 Onehot code 변환
X = pd.get_dummies(X, prefix = cats)
X.head()

# 수치형변수 자료 표준화
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X[['age', 'fare']])
scaler.transform(X[['age', 'fare']])
X.loc[:, ['age', 'fare']] = scaler.transform(X[['age', 'fare']])
X.head()

# 훈련과 시험데이터 분리후 훈련데이터 학습
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, stratify= y, random_state=11)
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression(random_state=11, solver='saga')

lr.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=11, solver='saga')
```

# Logistic regression – Titanic binary classification

```
# 예측
y_predict = lr.predict(X_test)
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_predict, labels=lr.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=lr.classes_)
disp.plot()
```



# Logistic regression – MNIST multi class 분류

```
# Logistic regression
from sklearn.datasets import load_digits
data = load_digits()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['label'] = data.target   # df['label'] = df['label'].apply(lambda x: 0 if x != 1 else x) 이항분류의 경우

X = df.drop('label', axis=1)
X = X/X.max() # 표준화
X.fillna(0, inplace=True)
y = df['label']

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, stratify= y, random_state=11)
from sklearn.multiclass import OneVsRestClassifier
base_lr= LogisticRegression(random_state=11, solver='saga')

# OneVsRestClassifier wrapper. 다중 class 분류 방식
ovr = OneVsRestClassifier(base_lr)
ovr.fit(X_train, y_train)
```

# Logistic regression

```
# confusion matrix
%matplotlib inline
y_predict = ovr.predict(X_test)
cm = confusion_matrix(y_test, y_predict, labels=ovr.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=ovr.classes_)
disp.plot()
```