회귀분석

1. 회귀분석

변수 간의 관계를 모델링하고 예측하는 통계적 기법 중 하나로 주로 연속형 종속 변수와 1개 이상의 범주형/연속형 독립 변수 간의 관계를 파악하고 설명하는 데 사용

1) 회귀분석 유형

유형	종속 변수	독립 변수	종속 변수와 독립 변수 간 관계의 선형성
단순 선형 회귀 (simple linear regression)	1개	1개	선형
다중 선형 회귀 (multiple linear regression)	1개	2개 이상	선형
일반 선형 모형 (general linear model)	2개 이상	1개 이상	선형
비선형 회귀 (non-linear regression)	1개	1개 이상	비선형

2. 단순 선형 회귀(simple linear regression)

$$y = \beta_0 + \beta_1 x + \varepsilon_i \qquad \varepsilon_i \overset{i.i.d.}{\sim} N(0, \sigma^2)$$

- $i.i.d.$: Independent and Identically Distributed
- 회귀계수(regression coefficient)

  β_1 : 기울기(slope) , β_0 : 절편(intercept)

- ε_i : i번째 관측치 오차

*** 오차항의 4가지 가정**

$\varepsilon_i \overset{i.i.d}{\sim} N(0, \sigma^2)$

① 선형성(linearity) : 종속 변수 y와 독립 변수 x는 선형 관계에 있음

② 독립성(independence) : 각각의 오차항은 서로 독립

③ 등분산성(constant variance) : 오차항의 분산은 모두 같음

④ 정규성(normality) : 오차항은 정규분포를 따름

1) 회귀계수 추정(OLS)

- 최소제곱법(least square method)

관측된 데이터와 모델의 예측값 간의 오차의 제곱합을 최소화하여 최적의 회귀계수를 찾는 방법

$$\varepsilon_i = y_i - \beta_0 - \beta_1 x_i \qquad i = 1, 2, \cdots, n$$

오차의 제곱합 식인 $\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$을 각 회귀계수에 대해 편미분하여 그 값이 0 이 되는 $\hat{\beta}_0$과 $\hat{\beta}_1$의 값을 도출하면 아래와 같음

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{S_{xy}}{S_{xx}}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

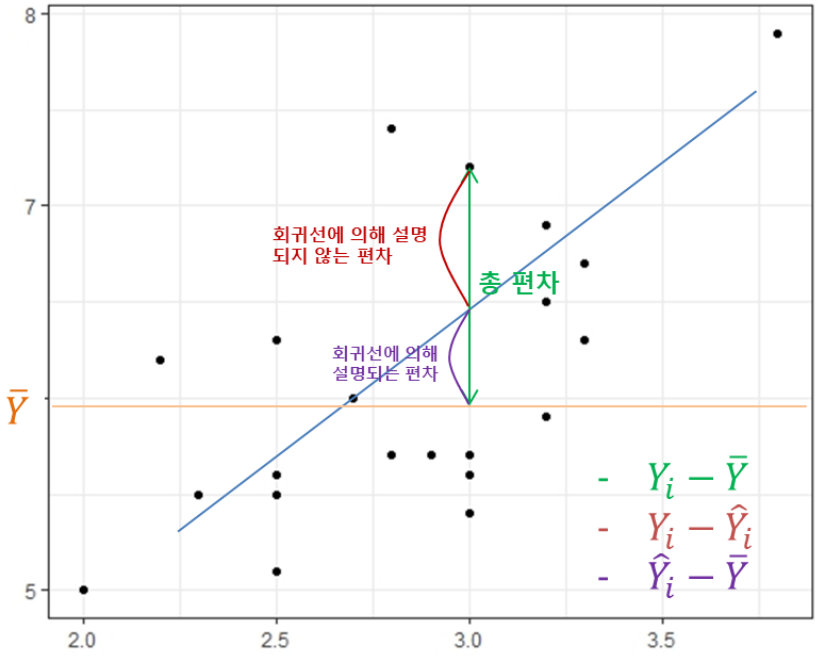
추정된 회귀계수를 통해 회귀식을 추정하면 아래와 같음

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \qquad i = 1, 2, \cdots, n$$

*** 잔차(residual)**

$$e_i = y_i - \hat{y}_i \qquad i = 1, 2, \cdots, n$$

실제 오차를 알 수 없기 때문에 잔차를 이용해 모형을 평가하고 가정을 검토함



$$\text{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \qquad \text{SSR} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \text{SSE} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$

$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$

TSS(total sum of squares) = SSR(regression sum of squares) + SSE(residual sum of squares)

[표] 유의성 검정을 위한 분산 분석 표

요인(Factor)	자유도 (Degree of Freedom)	제곱합 (Sum of Square)	평균제곱 (Mean Square)	F-value
회귀(regression)	1	SSR	$MSR = \frac{SSR}{1}$	$F = \frac{MSR}{MSE}$
잔차(Residual)	n-2	SSE	$MSE = \frac{SSE}{n-2}$	
총(Total)	n-1	SST		

검정통계량 : $F = \frac{MSR}{MSE}$ 기각역 : $F \geq F(k, n-2: \alpha)$

2) 유의성 검정

[1] 모형 유의성 검정

$$y = \beta_0 + \beta_1 x + \varepsilon_i$$

회귀식에서 기울기(β_1)가 0이면 모형이 유의하지 않음. 이를 확인하기 위해 가설검정을 하면 H_0와 H_1이 아래와 같음

$$H_0 : \beta_1 = 0 \quad \text{vs} \quad H_1 : \beta_1 \neq 0$$

통계패키지 결과 Analysis of Variance table을 얻을 수 있음

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F value	Pr>F
Model	1	SSR	MSR	F	p-value
Error	n-2	SSE	MSE		
Corrected Total	n-1	SST			

p-value < 유의수준($\alpha = 0.05$)이면 귀무가설 $H_0 : \beta_1 = 0$ 기각 $\Rightarrow$ 모형이 통계적으로 유의함

[2] 회귀계수 유의성 검정

각 변수의 중요도 및 유의성 파악 가능하며 이를 확인하기 위해 가설검정을 하면 H_0와 H_1이 아래와 같음.

$$H_0 : \beta_1 = 0 \quad \text{vs} \quad H_1 : \beta_1 \neq 0$$

검정통계량 : $T = \frac{\hat{\beta}_1 - 0}{\sqrt{\hat{\sigma}^2 / \sum_{i=1}^{n}(x_i - \bar{x})^2}}$ (단, $\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-2}$) 기각역 : $|T| \geq t_{\alpha/2}(n-2)$

통계패키지 결과 Parameter Estimates table을 얻을 수 있음

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t-value	Pr>|t|
Intercept	1	$\hat{\beta}_0$	S.E	T	p-value
x변수명	1	$\hat{\beta}_1$	S.E	T	p-value

x변수의 p-value < 유의수준($\alpha = 0.05$)이면 귀무가설 $H_0 : \beta_1 = 0$ 기각 ⇒ 회귀계수가 통계적으로 유의함

3) 회귀모형 평가

[1] 평균 절대 오차(MAE : Mean Absolute Error)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

: 실제 값과 예측 값 간의 차이를 절댓값으로 계산한 후 그 평균을 구한 값으로 오차의 크기를 고려하기 때문에 이상치에 덜 민감하며 값이 작을수록 좋은 모형임.

[2] 평균제곱근 오차(RMSE:Root Mean Square Error)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

: 오차를 제곱한 후에 평균을 구한 뒤 다시 제곱근을 취한 값으로 회귀선을 기준으로 실제값이 평균적으로 얼마나 벗어나 있는지 나타내는 값임. MAE와 달리 오차를 제곱하여 계산하므로 큰 오차에 더 많은 패널티를 부여하며 이상치(outlier)에 민감하게 반응하는 특징이 있고, 이 값이 작을수록 좋은 모형임.

*** RMSE는 y값의 단위에 영향을 받으므로 절대적인 측도는 아님**

오차의 제곱근 형태이기 때문에 실제값과 예측값의 단위와 동일한 단위를 갖게 되며 이로 인해 RMSE는 종속 변수(타겟 변수)의 단위에 영향을 받음.

[3] 결정계수(coefficient of determination)

$$R^2 = \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} = \frac{SSR}{SST}$$

: 총 편차 중에 회귀선에 의해 설명되는 편차의 비율로 1에 가까울수록 모형의 설명력이 높음

*** $R^2 = 0.8$이면 총 편차 중 80%가 모형에 의해 설명할 수 있음**

[4] 수정 결정계수

$$adj - R^2 = 1 - \frac{\frac{SSE}{n-p-1}}{\frac{SST}{n-1}}$$

n : 관측치 수 , p : 모형에 포함된 설명 변수의 수

: 수정 결정계수는 변수 수가 증가할수록 결정계수 값이 일정 수준 감소하게 변경된 모형 평가 지표로 설명 변수 수가 늘어남에 따라 증가하는 경향이 있는 결정계수의 문제점을 보완한 지표임

* R^2값과 $adj - R^2$값이 비슷하면 모델이 데이터를 잘 설명하고 있으며 과적합이나 불필요한 복잡성을 피하는 데 도움이 되는 것으로 간주.

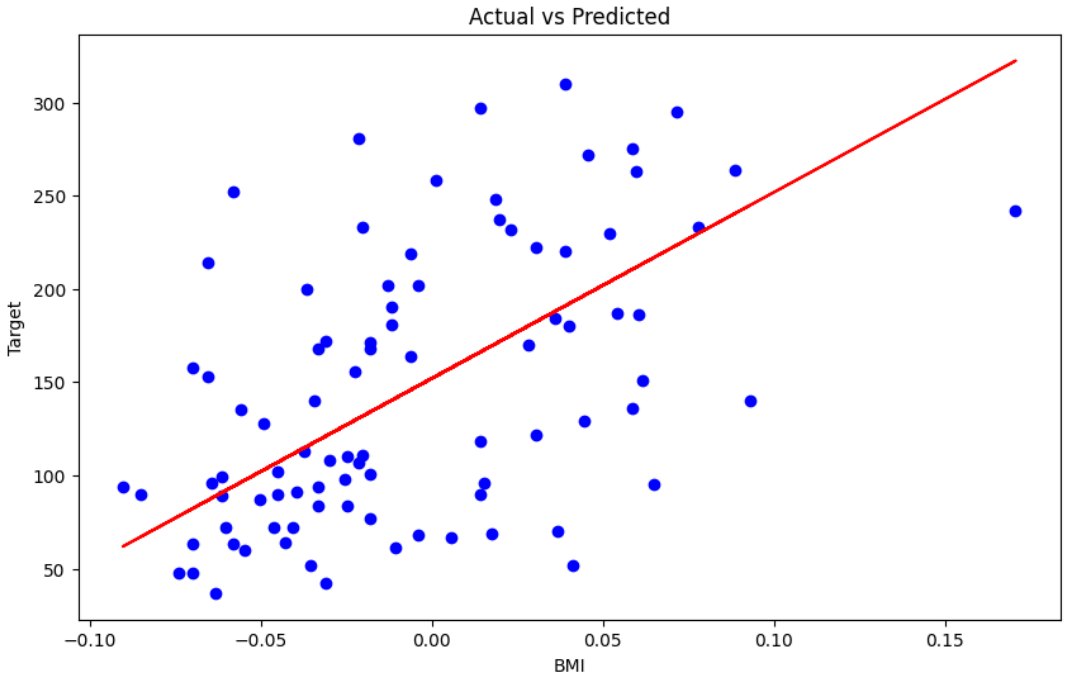
[단순 선형 회귀 실습]

```
from sklearn.datasets import load_diabetes

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import matplotlib.pyplot as plt

# 당뇨병 데이터셋 불러오기
diabetes_data = load_diabetes()

# 데이터프레임으로 변환
df_diabetes = pd.DataFrame(data=diabetes_data.data,
columns=diabetes_data.feature_names)
df_diabetes['target'] = diabetes_data.target

# 특징과 타겟 분리
X = df_diabetes[['bmi']]  # 단순선형회귀분석을 하기 위해 BMI 특징 선택
```

```
y = df_diabetes['target']

# 훈련 데이터셋과 테스트 데이터셋으로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 선형 회귀 모델 생성 및 훈련
model = LinearRegression()
model.fit(X_train, y_train)

# 테스트 데이터셋에 대한 예측
y_pred = model.predict(X_test)

# 평균 제곱 오차 계산
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# 특징에 상수항 추가 (절편)
X_train_const = sm.add_constant(X_train)    # 내부적으로 상수항을 처리하는
모델과는 달리 OLS 는 선형회귀모델이 상수항의 존재를 포착할 수 있도록 1 을 상수항으로 추가

# OLS 모델 생성 및 피팅
ols_model = sm.OLS(y_train, X_train_const)
ols_result = ols_model.fit()

# OLS summary 출력
print(ols_result.summary())

# 테스트 데이터셋과 예측 결과 시각화
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.title('Actual vs Predicted')
plt.xlabel('BMI')
plt.ylabel('Target')
plt.show()
```

```
Mean Squared Error: 4061.8259284949268
                            OLS Regression Results
==============================================================================
Dep. Variable:                 target   R-squared:                       0.366
Model:                            OLS   Adj. R-squared:                  0.364
Method:                 Least Squares   F-statistic:                     202.4
Date:                Fri, 15 Mar 2024   Prob (F-statistic):           1.40e-36
Time:                        06:42:31   Log-Likelihood:                -1958.2
No. Observations:                 353   AIC:                             3920.
Df Residuals:                     351   BIC:                             3928.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const       152.0034      3.316     45.841      0.000     145.482     158.525
bmi         998.5777     70.192     14.226      0.000     860.527    1136.628
==============================================================================
Omnibus:                        8.367   Durbin-Watson:                   1.786
Prob(Omnibus):                  0.015   Jarque-Bera (JB):                5.093
Skew:                           0.108   Prob(JB):                       0.0783
Kurtosis:                       2.453   Cond. No.                         21.2
==============================================================================
```



3. 다중 선형 회귀(multiple linear regression)

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon \qquad \varepsilon \overset{iid}{\sim} N(0, \sigma^2)$$

*** 다중 선형 회귀분석 고려사항**

① 다중공선성(multicollinearity)

: 독립변수 간의 선형 종속이 심한 경우로 차원 축소나 Ridge regression 등을 통해 해결

- VIF가 10보다 크면 다중공선성 문제가 있는 것으로 판단

② 범주형 설명 변수(qualitative predictors)

: 범주형 설명 변수를 어떻게 다룰 것인가에 관한 문제

③ 변수 선택(variable selection)

: 많은 변수로 인한 모형 과적합(overfitting) 문제를 변수 제거를 통해 해결

1) 유의성 검정

[1] 모형 유의성 검정

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

회귀식에서 기울기가 0이면 모형이 유의하지 않음. 이를 확인하기 위해 가설검정을 하면 H_0와 H_1이 아래와 같음

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0 \quad \text{vs} \quad H_1 : Some\ of\ \beta_j \neq 0 \quad (j = 1, \cdots, p)$$

[표] 유의성 검정을 위한 분산 분석 표

요인(Factor)	자유도 (Degree of Freedom)	제곱합 (Sum of Square)	평균제곱 (Mean Square)	F-value
회귀(regression)	p	SSR	$MSR = \frac{SSR}{p-1}$	$F = \frac{MSR}{MSE}$
잔차(Residual)	n-p-1	SSE	$MSE = \frac{SSE}{n-p}$	
총(Total)	n-1	SST		

검정통계량 : $F = \frac{MSR}{MSE}$ 기각역 : $F \geq F(k, n-p-1 : \alpha)$

F-test 에서 p-value < 유의수준($\alpha = 0.05$)이면 귀무가설 $H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$ 기각 ⇒ 모형이 통계적으로 유의함

[2] 회귀계수 유의성 검정

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

회귀식에서 기울기가 0이면 모형이 유의하지 않음. 이를 확인하기 위해 가설검정을 하면 H_0와 H_1이 아래와 같음

$$H_0 : \beta_j = 0 \quad \text{vs} \quad H_1 : \beta_j \neq 0 \qquad \text{검정통계량} : T = \frac{\hat{\beta}_j}{s.e.(\hat{\beta}_j)}$$

통계패키지 결과 Parameter Estimates table을 얻을 수 있음

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t-value	Pr>|t|
Intercept	1	$\hat{\beta}_0$	S.E	T	p-value
x_1변수명	1	$\hat{\beta}_1$	S.E	T	p-value
⋮	⋮	⋮	⋮	⋮	⋮
x_p변수명	1	$\hat{\beta}_p$	S.E	T	p-value

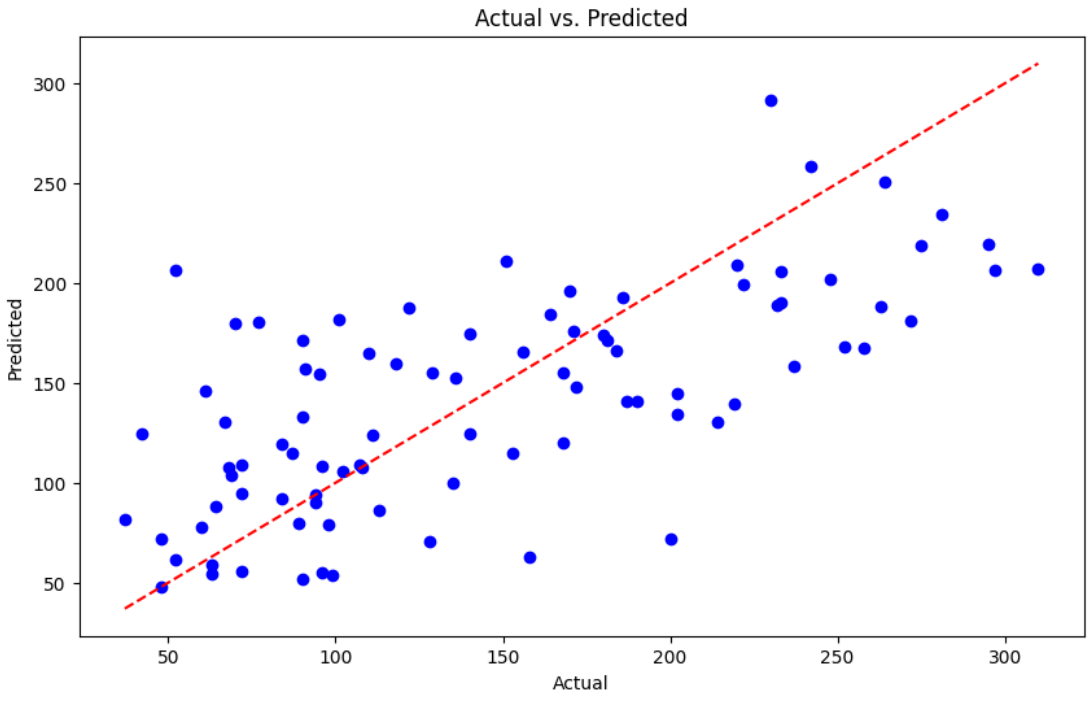
x_j변수의 p-value < 유의수준($\alpha = 0.05$)이면 귀무가설 $H_0 : \beta_j = 0$ 기각 ⇒ 회귀계수가 통계적으로 유의함

[다중 선형 회귀 코드]

```
import statsmodels.api as sm
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# 당뇨병 데이터셋 로드
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 데이터셋을 훈련 세트와 테스트 세트로 나눔
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 다중 선형 회귀 모델 초기화 및 학습
model = LinearRegression()
model.fit(X_train, y_train)

# 테스트 세트에 대한 예측
y_pred = model.predict(X_test)

# 모델 평가 (평균 제곱 오차 사용)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# OLS 요약 출력
X_train_const = sm.add_constant(X_train)  # 상수항 추가
ols_model = sm.OLS(y_train, X_train_const)
ols_results = ols_model.fit()
print(ols_results.summary())

# 테스트 세트에 대한 예측 값과 실제 값의 plot
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linestyle='--')
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
plt.title('Actual vs. Predicted')
plt.show()
```

```
Mean Squared Error: 2900.193628493482
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.528
Model:                            OLS   Adj. R-squared:                  0.514
Method:                 Least Squares   F-statistic:                     38.25
Date:                Fri, 15 Mar 2024   Prob (F-statistic):           5.41e-50
Time:                        06:51:51   Log-Likelihood:                -1906.1
No. Observations:                 353   AIC:                             3834.
Df Residuals:                     342   BIC:                             3877.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        151.3456      2.902     52.155      0.000     145.638     157.053
x1            37.9040     69.056      0.549      0.583     -97.923     173.731
x2          -241.9644     68.570     -3.529      0.000    -376.836    -107.093
x3           542.4288     76.956      7.049      0.000     391.062     693.795
x4           347.7038     71.357      4.873      0.000     207.350     488.057
x5          -931.4888    451.138     -2.065      0.040   -1818.844     -44.134
x6           518.0623    364.114      1.423      0.156    -198.122    1234.247
x7           163.4200    233.014      0.701      0.484    -294.901     621.741
x8           275.3179    185.400      1.485      0.138     -89.349     639.985
x9           736.1989    192.437      3.826      0.000     357.689    1114.709
x10           48.6707     73.435      0.663      0.508     -95.771     193.113
==============================================================================
Omnibus:                        1.457   Durbin-Watson:                   1.794
Prob(Omnibus):                  0.483   Jarque-Bera (JB):                1.412
Skew:                           0.064   Prob(JB):                        0.494
Kurtosis:                       2.718   Cond. No.                         219.
==============================================================================
```



2) 범주형 변수를 포함한 모형

범주형 변수를 dummy 변수로 변환하여 활용함

* dummy 변수

범주형 변수를 0 또는 1 의 값으로 변환한 변수

ex) $G = Male\ or\ Female$인 범주형 변수를 선형회귀모델에서 다루는 경우 (범주형 변수 1 개)

$$X = \begin{cases} 1, & if\ G = Male \\ 0, & otherwise \end{cases}$$

위와 같이 범주형 변수 G를 dummy 변수 X로 변환하여 활용함. 변환한 dummy 변수를 활용한 회귀 모델은 다음과 같음.

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- $\mu = \beta_0 + \beta_1 X$

$\Rightarrow \mu_{Female} = \beta_0 \ (\because X = 0)$, $\mu_{Female} = \beta_0 + \beta_1 \ (\because X = 1)$

- $\beta_0 = \mu_{Female}$ (Female : base line)

- $\beta_1 = \mu_{Male} - \mu_{Female}$

ex) $C = Red, Yellow\ or\ Green$, $G = Male\ or\ Female$ 인 범주형 변수를 선형회귀모델에서 다루는 경우 (범주형 변수 2 개)

$$X_1 = \begin{cases} 1, & if\ C = Yellow \\ 0, & otherwise \end{cases}$$

$$X_2 = \begin{cases} 1, & if\ C = Green \\ 0, & otherwise \end{cases}$$

$$X_3 = \begin{cases} 1, & if\ G = Female \\ 0, & otherwise \end{cases}$$

위와 같이 2 개의 범주형 변수를 dummy 변수 X_1, X_2, X_3로 변환하여 활용하며 X_1, X_2 2 개의 변수 중 하나만 고려하거나 하나만 제외하고 분석할 수 없음. 변환한 dummy 변수를 활용한 회귀 모델은 다음과 같음.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$$

- $\mu = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$

$\Rightarrow \mu_{YF} = \beta_0 + \beta_1 + \beta_3$, $\mu_{GF} = \beta_0 + \beta_2 + \beta_3$, $\mu_{RF} = \beta_0 + \beta_3$

$\mu_{YM} = \beta_0 + \beta_1$, $\mu_{GM} = \beta_0 + \beta_2$, $\mu_{RM} = \beta_0$

- $\beta_0 = \mu_{RM}$ (Red and Male : base levels)

[범주형 변수가 포함된 선형 회귀 코드]

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# 당뇨병 데이터셋 로드
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 데이터셋을 판다스 DataFrame 으로 변환
df = pd.DataFrame(X, columns=['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3',
's4', 's5', 's6'])
df['diabetes_stage'] = np.random.choice(['early', 'intermediate',
'advanced'], size=len(df))

# 범주형 변수를 더미 변수로 변환
df_dummies = pd.get_dummies(df, columns=['diabetes_stage'])

# 독립 변수와 종속 변수 분리
X = df_dummies.drop(columns=['sex'])  # 범주형 변수를 포함한 독립 변수
y = y

# 데이터셋을 훈련 세트와 테스트 세트로 나눔
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 선형 회귀 모델 초기화 및 학습
model = LinearRegression()
model.fit(X_train, y_train)

# 테스트 세트에 대한 예측
y_pred = model.predict(X_test)

# 모델 평가
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# OLS 요약 출력
X_train_const = sm.add_constant(X_train)  # 상수항 추가
ols_model = sm.OLS(y_train, X_train_const)
ols_results = ols_model.fit()
```

```
print(ols_results.summary())
Mean Squared Error: 2943.2744876403763
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.512
Model:                            OLS   Adj. R-squared:                  0.496
Method:                 Least Squares   F-statistic:                     32.55
Date:                Fri, 15 Mar 2024   Prob (F-statistic):           7.96e-47
Time:                        08:36:24   Log-Likelihood:                -1911.9
No. Observations:                 353   AIC:                             3848.
Df Residuals:                     341   BIC:                             3894.
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const                      113.3744      2.223     51.010      0.000     109.003
117.746
age                         -3.9594     69.997     -0.057      0.955    -141.639
133.720
bmi                        592.8092     77.164      7.682      0.000     441.032
744.586
bp                         309.7170     72.171      4.291      0.000     167.761
451.673
s1                        -889.9030    460.350     -1.933      0.054   -
1795.385     15.579
s2                         509.2582    371.657      1.370      0.172    -221.771
1240.288
s3                         200.5518    237.890      0.843      0.400    -267.365
668.469
s4                         187.5109    187.019      1.003      0.317    -180.346
555.368
s5                         764.0854    195.994      3.899      0.000     378.576
1149.595
s6                          40.3140     74.830      0.539      0.590    -106.873
187.501
diabetes_stage_advanced       33.4948      4.462      7.506      0.000
24.717     42.272
diabetes_stage_early          40.8731      4.244      9.630      0.000
32.524     49.222
diabetes_stage_intermediate   39.0066      4.191      9.307      0.000
30.763     47.250
==============================================================================
Omnibus:                        4.793   Durbin-Watson:                   1.806
Prob(Omnibus):                  0.091   Jarque-Bera (JB):                3.488
Skew:                           0.100   Prob(JB):                        0.175
Kurtosis:                       2.556   Cond. No.                     4.18e+15
==============================================================================
```

3) VIF (Variance Inflation Factor)

다중공선성 측정을 위한 지표 중 하나로 각 독립변수의 분산을 다른 독립변수들로부터 설명되는 분산의 비율로 계산

$$VIF(X_i) = \frac{1}{1 - R^2_{X_i}}$$

$R^2_{X_i}$: X_i를 나머지 독립변수들의 선형결합으로 설명한 결정계수

⇒ X_i와 다른 독립 변수들 간 상관관계↑ ⇒ $R^2_{X_i}$↑ ⇒ VIF↑

VIF 가 10 이상일 때 다중공선성 문제가 있는 것으로 판단하고 변수 선택, 변수 변환, 차원 축소, 릿지(Ridge) 등을 통해 해결 가능

[다중공선성 확인 실습]

```
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.datasets import load_diabetes

# 데이터셋 로드
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 데이터프레임으로 변환
df = pd.DataFrame(X, columns=['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3',
's4', 's5', 's6'])

# VIF 계산 함수
def calculate_vif(X):
    vif_data = pd.DataFrame()
    vif_data["Feature"] = X.columns
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(len(X.columns))]
    return vif_data

# VIF 계산
vif_df = calculate_vif(df)

# 결과 출력
print(vif_df)
```

```
Feature       VIF
0     age   1.217307
1     sex   1.278071
```

```
2     bmi   1.509437
3      bp   1.459428
4      s1  59.202510
5      s2  39.193370
6      s3  15.402156
7      s4   8.890986
8      s5  10.075967
9      s6   1.484623
```

4) 변수 선택 (Variable Selection)

[1] 전진 선택 (Forward Selection)

① 어떤 변수도 선택하지 않은 상태로 시작

② p-value 가 유의수준($\alpha = 0.05$)보다 작으면서 그 값이 가장 작은 변수를 선택하여 모델에 추가

③ p-value 가 유의수준보다 작은 변수가 없을 때까지 반복

[전진 선택 코드]

```
import numpy as np
import statsmodels.api as sm
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

# 데이터셋 불러오기
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 전진 선택을 위한 함수 정의
def forward_selection(X, y):
    n_features = X.shape[1]
    selected_features = []
    best_pvalue = float('inf')

    print("Step 0: No features selected")

    for i in range(n_features):
        print("\nStep", i + 1, ":")
        candidate_features = [f for f in range(n_features) if f not in
selected_features]
        best_candidate_pvalue = float('inf')
        best_candidate_feature = None
        for f in candidate_features:
            features = selected_features + [f]
```

```
            X_selected = sm.add_constant(X[:, features])
            X_train, X_test, y_train, y_test = train_test_split(X_selected,
y, test_size=0.2, random_state=42)
            model = sm.OLS(y_train, X_train)    # OLS로 선형회귀모델을 추정
            results = model.fit()
            pvalue = results.pvalues[-1]  # 상수항의 p-value
            print("Trying feature", f, "with p-value:", pvalue)
            if pvalue < best_candidate_pvalue:    # p-value가
best_candidate_pvalue보다 작으면 그 값과 변수를 best_candidate로 저장
                best_candidate_pvalue = pvalue
                best_candidate_feature = f
        if best_candidate_pvalue > 0.05:    # 가장 작은 p-value값이 유의수준
0.05보다 큰 경우 변수선택 종료
            print("All remaining features have p-values greater than 0.05.
Stopping forward selection.")
            break
        print("Selected feature", best_candidate_feature, "with p-value:",
best_candidate_pvalue)
        selected_features.append(best_candidate_feature)

    return selected_features

# 전진 선택 알고리즘 적용
selected_features = forward_selection(X, y)

# 결과 출력
print("\nSelected features:", selected_features)
```

```
Step 0: No features selected

Step 1 :
Trying feature 0 with p-value: 0.00020298294220702392
Trying feature 1 with p-value: 0.894012907513373
Trying feature 2 with p-value: 1.396737194598559e-36
Trying feature 3 with p-value: 1.498396400895689e-18
Trying feature 4 with p-value: 0.0001607301867092091
Trying feature 5 with p-value: 0.003522507470283052
Trying feature 6 with p-value: 7.561955225811275e-14
Trying feature 7 with p-value: 6.365822767753934e-17
Trying feature 8 with p-value: 1.4546354562571886e-29
Trying feature 9 with p-value: 2.6910462211899892e-14
Selected feature 2 with p-value: 1.396737194598559e-36

Step 2 :
Trying feature 0 with p-value: 0.0421765709371364
Trying feature 1 with p-value: 0.7528551895691031
Trying feature 3 with p-value: 7.474019051506183e-08
Trying feature 4 with p-value: 0.349364952671387
Trying feature 5 with p-value: 0.9368596549615127
Trying feature 6 with p-value: 1.6124779272880095e-05
Trying feature 7 with p-value: 1.4513714005724225e-05
Trying feature 8 with p-value: 1.1050001882347416e-13
Trying feature 9 with p-value: 0.00016126682389089016
Selected feature 8 with p-value: 1.1050001882347416e-13
```

```
Step 3 :
Trying feature 0 with p-value: 0.5585777616889526
Trying feature 1 with p-value: 0.23039838603985763
Trying feature 3 with p-value: 5.321308594184034e-05
Trying feature 4 with p-value: 0.0024560877473210237
Trying feature 5 with p-value: 0.07128148703721227
Trying feature 6 with p-value: 0.005541830935208539
Trying feature 7 with p-value: 0.4563404408545064
Trying feature 9 with p-value: 0.17266017658673383
Selected feature 3 with p-value: 5.321308594184034e-05

Step 4 :
Trying feature 0 with p-value: 0.7262377707571822
Trying feature 1 with p-value: 0.05634230826942985
Trying feature 4 with p-value: 0.0009634972359460144
Trying feature 5 with p-value: 0.05626402204928848
Trying feature 6 with p-value: 0.0013488484709873153
Trying feature 7 with p-value: 0.2683437785900546
Trying feature 9 with p-value: 0.5280989900787207
Selected feature 4 with p-value: 0.0009634972359460144

Step 5 :
Trying feature 0 with p-value: 0.8879172155287816
Trying feature 1 with p-value: 0.03404450869406537
Trying feature 5 with p-value: 0.00916346512982158
Trying feature 6 with p-value: 0.023177562847503202
Trying feature 7 with p-value: 0.016281709396239488
Trying feature 9 with p-value: 0.3388768536405079
Selected feature 5 with p-value: 0.00916346512982158

Step 6 :
Trying feature 0 with p-value: 0.9166990759826358
Trying feature 1 with p-value: 0.0012236671626608682
Trying feature 6 with p-value: 0.8493151727342759
Trying feature 7 with p-value: 0.5431654511232065
Trying feature 9 with p-value: 0.5624546910837185
Selected feature 1 with p-value: 0.0012236671626608682

Step 7 :
Trying feature 0 with p-value: 0.5955420748799019
Trying feature 6 with p-value: 0.7072990246748654
Trying feature 7 with p-value: 0.17367651471849296
Trying feature 9 with p-value: 0.41411946073999706
All remaining features have p-values greater than 0.05. Stopping forward
selection.

Selected features: [2, 8, 3, 4, 5, 1]
```

[2] 후진 제거 (Backward Elimination)

① 모든 변수를 포함한 전체 모델에서 시작

② p-value 가 유의수준($\alpha = 0.05$)보다 크면서 그 값이 가장 큰 변수를 선택하여 모델에서 제거

③ p-value 가 유의수준보다 큰 변수가 없을 때까지 반복

* 전진 선택보다 후진 제거가 더 자주 사용되는 이유

- 효율성

전진 선택 : 각 변수에 대해 성능을 평가하고 추가하는 과정을 반복하여 모델 구축

후진 제거 : 전체 모델에 대해 성능을 평가하여 불필요한 변수를 제거하여 모델을 구축

⇒ 변수가 많은 경우에도 비교적 효율적으로 모델 구축 가능

- 다중공선성

모든 변수를 포함한 상태에서 시작하여 다중공선성이 있는 변수를 제거하면서 모델을 구축하므로 다중공선성 고려에 유리

- 통계적 신뢰도

모든 변수를 고려한 상태에서 시작하므로 최종 모델의 신뢰도가 높을 가능성이 있음

[후진 제거 코드]

```
import numpy as np
import statsmodels.api as sm
from sklearn.datasets import load_diabetes

# 데이터셋 불러오기
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 후진 제거를 위한 함수 정의
def backward_elimination(X, y):
    n_features = X.shape[1]
    selected_features = list(range(n_features))

    print("Step 0: All features selected")

    for i in range(n_features):
        print("\nStep", i + 1, ":")
        X_selected = sm.add_constant(X[:, selected_features])
        model = sm.OLS(y, X_selected)   # OLS로 선형회귀모델 추정
        results = model.fit()
        pvalues = results.pvalues[1:]  # 첫 번째는 상수항이므로 제외
        print("P-values of features:")
        for j, pvalue in enumerate(pvalues):
            print("Trying feature", selected_features[j], "with p-value:",
pvalue)
```

```
        max_pvalue = max(pvalues)   # p-value 값 중 가장 큰 값 저장
        max_pvalue_index = np.argmax(pvalues)
        if max_pvalue > 0.05:   # 가장 큰 p-value가 유의수준 0.05보다 작으면
변수선택 종료
            print("Removing feature with max p-value:", max_pvalue)
            selected_features.pop(max_pvalue_index)
        else:
            print("All remaining features have p-values less than or equal
to 0.05. Stopping backward elimination.")
            break

    return selected_features

# 후진 제거 알고리즘 적용
selected_features = backward_elimination(X, y)

# 결과 출력
print("\nSelected features:", selected_features)
```

```
Step 0: All features selected

Step 1 :
P-values of features:
Trying feature 0 with p-value: 0.8670306337000818
Trying feature 1 with p-value: 0.00010416711927693194
Trying feature 2 with p-value: 4.296391419518744e-14
Trying feature 3 with p-value: 1.0242783922113987e-06
Trying feature 4 with p-value: 0.05794760536919818
Trying feature 5 with p-value: 0.1603902400149649
Trying feature 6 with p-value: 0.6347232557752092
Trying feature 7 with p-value: 0.27345869366068
Trying feature 8 with p-value: 1.5558990865392858e-05
Trying feature 9 with p-value: 0.305989526196422
Removing feature with max p-value: 0.8670306337000818

Step 2 :
P-values of features:
Trying feature 1 with p-value: 8.850076893232605e-05
Trying feature 2 with p-value: 3.992811898786935e-14
Trying feature 3 with p-value: 7.432272895781585e-07
Trying feature 4 with p-value: 0.058028517252595044
Trying feature 5 with p-value: 0.16163479573070638
Trying feature 6 with p-value: 0.6385632161213257
Trying feature 7 with p-value: 0.2718028267264007
Trying feature 8 with p-value: 1.5354874539696932e-05
Trying feature 9 with p-value: 0.31173206527604636
Removing feature with max p-value: 0.6385632161213257

Step 3 :
P-values of features:
Trying feature 1 with p-value: 7.920140510741005e-05
Trying feature 2 with p-value: 4.111083896537281e-14
Trying feature 3 with p-value: 7.654827997067146e-07
Trying feature 4 with p-value: 0.0025808811148499435
Trying feature 5 with p-value: 0.10967685033822673
Trying feature 7 with p-value: 0.2923202965789548
```

```
Trying feature 8 with p-value: 1.4473797121790728e-08
Trying feature 9 with p-value: 0.30401122713321616
Removing feature with max p-value: 0.30401122713321616

Step 4 :
P-values of features:
Trying feature 1 with p-value: 0.0001066131924191784
Trying feature 2 with p-value: 7.248160717363033e-15
Trying feature 3 with p-value: 1.8178874736740085e-07
Trying feature 4 with p-value: 0.0028100704962663306
Trying feature 5 with p-value: 0.11048277468242394
Trying feature 7 with p-value: 0.26191904943121136
Trying feature 8 with p-value: 6.398601734358617e-09
Removing feature with max p-value: 0.26191904943121136

Step 5 :
P-values of features:
Trying feature 1 with p-value: 0.0001758473600959283
Trying feature 2 with p-value: 6.687185418596613e-15
Trying feature 3 with p-value: 2.7868822556225574e-07
Trying feature 4 with p-value: 3.12257261439819e-06
Trying feature 5 with p-value: 0.00027230239927340837
Trying feature 8 with p-value: 1.9386354385069487e-21
All remaining features have p-values less than or equal to 0.05. Stopping
backward elimination.

Selected features: [1, 2, 3, 4, 5, 8]
```

[3] 단계적 선택 (Stepwise Selection)

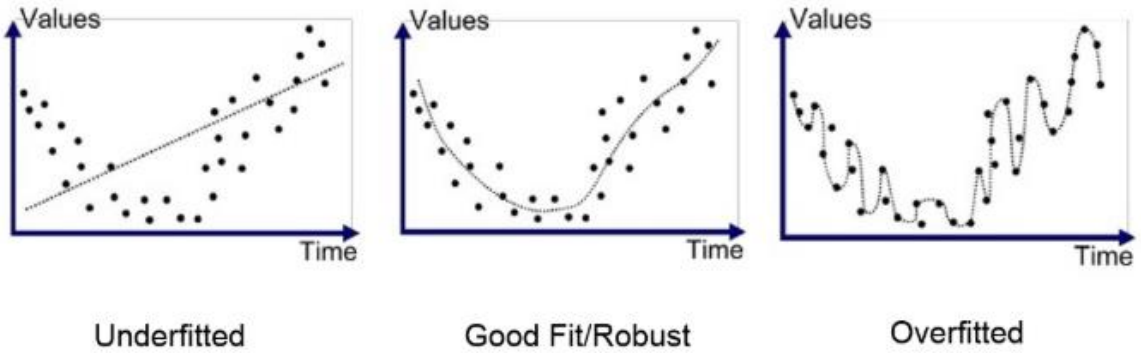
① 어떤 변수도 선택하지 않은 상태로 시작

② 전진 선택법으로 변수를 하나씩 추가

③ 후진 제거법으로 불필요한 변수를 제거

④ 더 이상 추가하거나 제거할 변수가 없을 때까지 반복

[4] Lasso, Ridge 회귀



Regularization(규제, 정규화)을 통해 다중회귀모형의 과적합을 방지하고 일반화 성능을 잃지 않도록 가중치를 제한함.

(1) Lasso 회귀

어떤 벡터 요소의 절댓값의 합인 L1-norm penalty 를 가진 선형 회귀 방법 (L1 규제 사용)

- MSE 가 최소가 되는 가중치와 편향을 찾는 동시에 가중치들의 절댓값 합이 최소가 되도록 함. 즉, 가중치의 모든 원소가 0 이 되거나 0 에 가깝도록 함. 가중치가 0 이 되는 특징의 경우 모델을 만들 때 사용되지 않기도 함. (변수 선택)

$$MSE + penalty$$
$$= MSE + \alpha \cdot L_1 - norm$$
$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{m}|W_j|$$

* 라쏘 회귀의 목적

$$\underset{w,b}{\operatorname{argmin}}\{ \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{m}|W_j| \}$$

$MSE + penalty$ 값이 최소가 되도록 하는 w(가중치)와 b(절편)값 찾기

- α값↑ ⇒ 규제↑, 가중치↓ ⇒ 과소적합(underfitting)

- α값↓ ⇒ 규제↓, 가중치↑ ⇒ 과대적합(overfitting)

[라쏘 회귀 실습]

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score

# 당뇨병 데이터셋 불러오기
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 훈련 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 라쏘 회귀 모델 생성 및 학습
```

```
alpha = 0.1  # 라쏘 회귀의 정규화 강도 조절 파라미터
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)

# 훈련된 모델을 사용하여 예측
y_pred = lasso.predict(X_test)

# 모델 평가 - R-squared
r_squared = r2_score(y_test, y_pred)
print("R-squared:", r_squared)
print("라쏘 회귀 모델의 계수:", lasso.coef_)
```

```
R-squared: 0.4718547867276227
라쏘 회귀 모델의 계수: [   0.         -152.66477923  552.69777529  303.36515791
-81.36500664
   -0.         -229.25577639    0.          447.91952518   29.64261704]
```

(2) Ridge 회귀 : 계수를 축소하여 변수의 영향력을 줄임

L2 정규화를 사용하여 회귀 계수를 추정

목적 함수에는 잔차 제곱합과 회귀 계수의 제곱합에 대한 항이 추가

L2 정규화는 회귀 계수의 크기를 제한하여 변수 간 상호작용과 다중공선성을 줄이는 효과가 있습니다. 그러나 변수 선택은 수행하지 않습니다.

어떤 벡터 요소의 제곱의 합인 L2-norm penalty 를 가진 선형 회귀 방법 (L2 규제 사용)

- MSE 가 최소가 되는 가중치와 편향을 찾는 동시에 가중치들의 제곱의 합이 최소가 되도록 함. 라쏘 회귀와는 달리 가중치가 0 에 가까워지지만 0 이 되지는 않음. (변수 선택 X) 하지만, 회귀 계수의 크기를 제한하여 변수 간 상호작용과 다중공선성을 줄이는 효과가 있음.

$$MSE + penalty$$
$$= MSE + \alpha \cdot L_2 - norm$$
$$= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{m} {W_j}^2$$

* 릿지 회귀의 목적

$$\underset{w,b}{\operatorname{argmin}}\{ \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{m} {W_j}^2 \}$$

$MSE + penalty$ 값이 최소가 되도록 하는 w(가중치)와 b(절편)값 찾기

- α값 ↑ ⇒ 규제 ↑, 가중치 ↓ ⇒ 과소적합(underfitting)

- α값 ↓ ⇒ 규제 ↓, 가중치 ↑ ⇒ 과대적합(overfitting)

[릿지 회귀 실습]

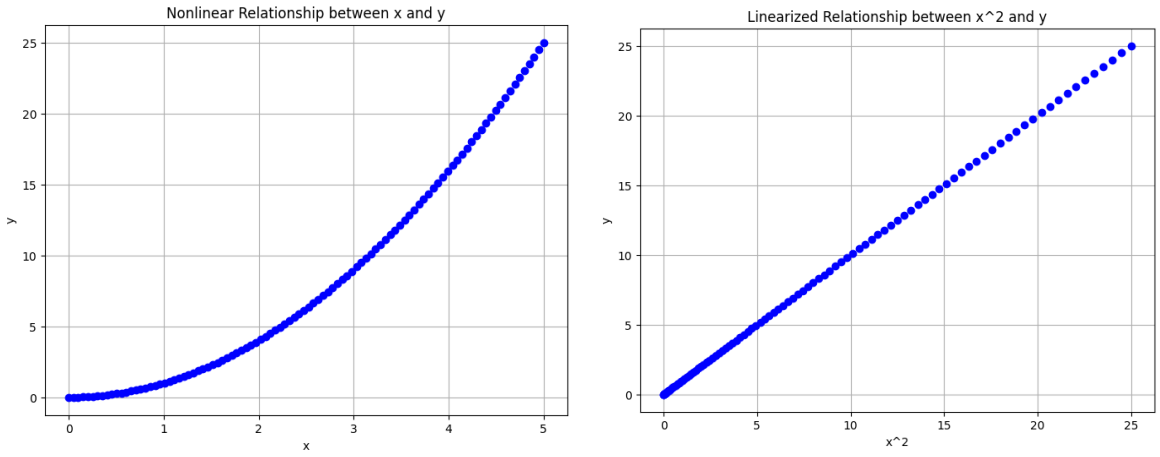
```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

# 당뇨병 데이터셋 불러오기
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 훈련 데이터와 테스트 데이터로 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 릿지 회귀 모델 생성 및 학습
alpha = 0.1  # 릿지 회귀의 정규화 강도 조절 파라미터
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)

# 훈련된 모델을 사용하여 예측
y_pred = ridge.predict(X_test)

# 모델 평가 - R-squared 로
r_squared = r2_score(y_test, y_pred)
print("R-squared:", r_squared)
print("릿지 회귀 모델의 계수:", ridge.coef_)
```

```
R-squared: 0.46085219464119265
릿지 회귀 모델의 계수: [  42.85566976 -205.49431899  505.08903304  317.0932049
-108.50026183
  -86.23673333 -190.36318008  151.70708637  392.28931896   79.9081772 ]
```

4. 회귀모형의 가정 진단

구분	진단 방법	해결 방안
선형성(linearity)	산점도, 상관계수	변수 변환, 비선형 모형 적합
독립성 (independence)	Durbin-Watson, ACF(Auto Correlation Function), 잔차 그래프	ARMA 와 같은 시계열 모형 이용
정규성(normality)	첨도와 왜도, Q-Q plot, 정규성 검정(Shapiro-Wilk or K-S test)	변수 변환, 새로운 변수 투입, 모형 수정
등분산성 (constant variance)	잔차 등분산 그래프, White test 등	변수 변환, 가중회귀분석(WLS)

1) 선형성 진단



[그림] 선형성 가정이 위배된 그래프와 제곱변환을 통해 선형성을 만족시킨 예시

① 잔차와 예측값 간의 산점도 확인

무작위적이지 않고 어떠한 패턴이 보이는 경우 선형성 가정이 위배되었다고 판단

② 종속변수와 독립변수들 간의 산점도 확인

산점도가 직선 형태를 따르지 않는 경우 선형성 가정이 위배되었다고 판단

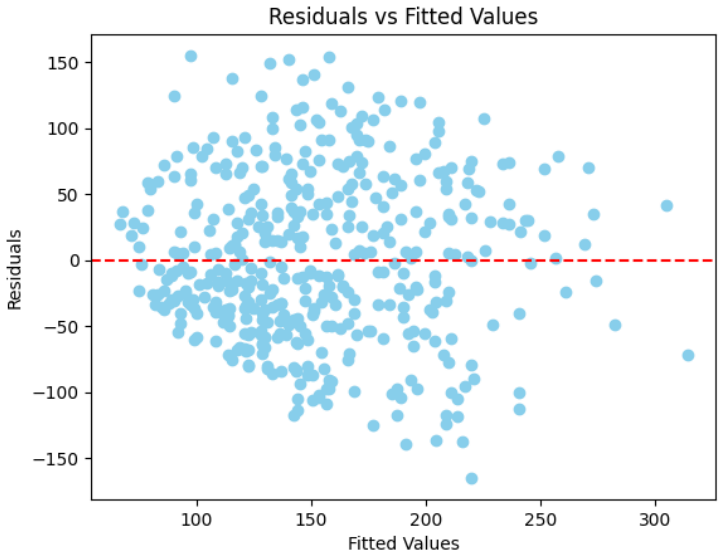
(상관계수를 이용하면 선형 관계의 정도 및 유의성은 파악 가능하지만 비선형 상관관계를 알 수 없음)

선형성 가정이 위배된 경우 (Lack of Fit) : 독립변수의 제곱 변환 등을 통해 선형성 유도 가능

[선형성 진단 및 선형성 유도 코드]

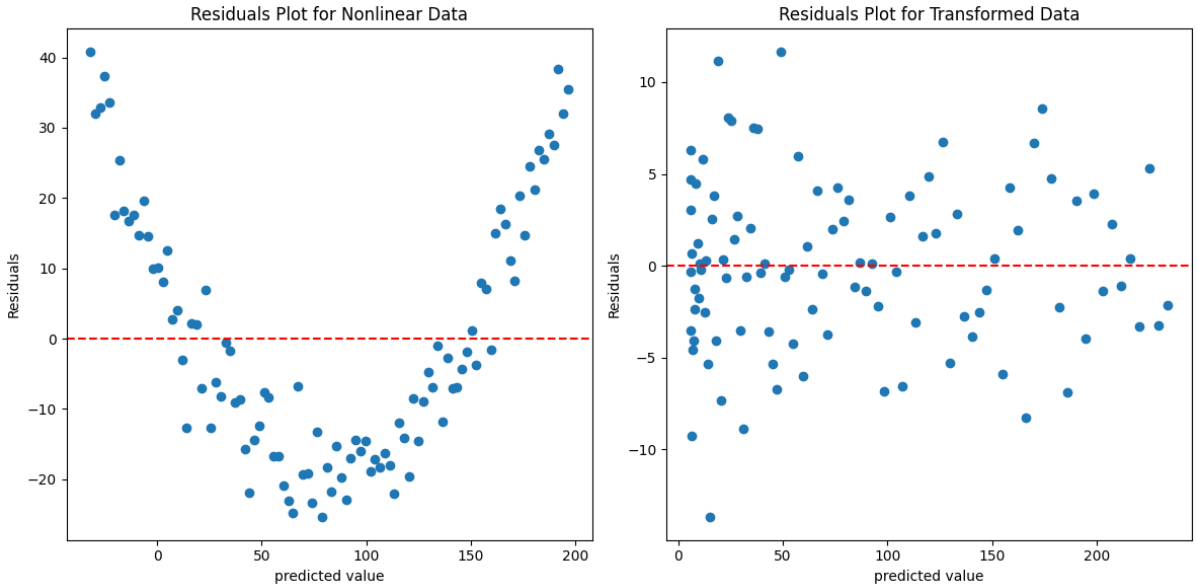
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression

# 당뇨병 데이터셋 로드
diabetes = load_diabetes()
X, y = diabetes.data[:, np.newaxis, 2], diabetes.target  # 3번째 특성 사용

# 선형 회귀 모델 피팅
model = LinearRegression()
model.fit(X, y)

# 예측값과 잔차 계산
```

```
y_pred = model.predict(X)
residuals = y - y_pred

# 잔차에 대한 산점도 그리기
plt.scatter(y_pred, residuals, color='skyblue')
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='red', linestyle='--')  # 잔차가 0 인 수평선 추가
plt.show()
```




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

# 비선형 데이터 생성 (여기서는 2 차 함수로 가정)
np.random.seed(0)
X = np.linspace(0, 10, 100)
y = 2 * X**2 + 3 * X + np.random.normal(0, 5, 100)

# 선형 회귀 모델 적합
model = LinearRegression()
model.fit(X.reshape(-1, 1), y)
y_pred = model.predict(X.reshape(-1, 1))

# 잔차 계산
residuals = y - y_pred

# 변수 변환을 통해 선형성 유도 (제곱 변환을 사용)
X_transformed = X**2
```

```
# 선형 회귀 모델 적합 (변환된 변수 사용)
model_transformed = LinearRegression()
model_transformed.fit(X_transformed.reshape(-1, 1), y)
y_pred_transformed = model_transformed.predict(X_transformed.reshape(-1,
1))

# 잔차 산점도 그리기
plt.figure(figsize=(12, 6))

# 원본 데이터의 잔차
plt.subplot(1, 2, 1)
plt.scatter(y_pred, residuals)
plt.xlabel('predicted value')
plt.ylabel('Residuals')
plt.title('Residuals Plot for Nonlinear Data')
plt.axhline(y=0, color='r', linestyle='--')  # 잔차가 0 주변에 수평선 추가

# 변수 변환 후 데이터의 잔차
plt.subplot(1, 2, 2)
plt.scatter(y_pred_transformed, residuals_transformed)
plt.xlabel('predicted value')
plt.ylabel('Residuals')
plt.title('Residuals Plot for Transformed Data')
plt.axhline(y=0, color='r', linestyle='--')  # 잔차가 0 주변에 수평선 추가

plt.tight_layout()
plt.show()
```



2) 독립성 진단

① Durbin-Watson 통계량 확인

잔차들 간의 자기상관 구조를 평가하는 데 사용되며 0 과 4 사이의 값을 가짐.

- 0 에 가까운 값 : 잔차들 간의 양의 상관관계가 있을 수 있음. (양의 자기상관)

- 2 에 가까운 값 : 잔차들이 서로 독립적임. (자기상관 없음)

- 4 에 가까운 값 : 잔차들 간의 음의 상관관계가 있을 수 있음. (음의 자기상관)

독립성 가정이 위배된 경우 : 독립 변수나 종속 변수의 로그 변환 또는 제곱근 변환 등을 통해 독립성 유도 가능

[독립성 진단 코드]

```
import numpy as np
import statsmodels.stats.stattools as sm_stat
from sklearn.datasets import load_diabetes
from scipy.stats import norm

# 당뇨병 데이터셋 로드
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# 3 번째 특성 선택
data = X[:, 2]

# Durbin-Watson 통계량 계산
dw_statistic = sm_stat.durbin_watson(data)
print("Durbin-Watson 통계량:", dw_statistic)

# Durbin-Watson 검정
def durbin_watson_test(dw_statistic):
    n = len(data)
    # 잔차의 자유도
    k = 1
    # 잔차의 표준편차
    resid_std = np.std(data)

    # Durbin-Watson 임계값 계산
    alpha = 0.05
    z_alpha_over_2 = norm.ppf(1 - alpha / 2)
    dw_crit_lower = 2 - z_alpha_over_2 * np.sqrt((2 * (k + 1)) / n)
    dw_crit_upper = 2 + z_alpha_over_2 * np.sqrt((2 * (k + 1)) / n)
```

```
    if dw_statistic < dw_crit_lower:
        return "귀무가설 기각: 양의 자기상관이 존재합니다."
    elif dw_statistic > dw_crit_upper:
        return "귀무가설 기각: 음의 자기상관이 존재합니다."
    else:
        return "귀무가설 채택: 자기상관이 존재하지 않습니다."

print(durbin_watson_test(dw_statistic))
```

```
Durbin-Watson 통계량: 2.0239082867255465
귀무가설 채택: 자기상관이 존재하지 않습니다.
```

```
import numpy as np
from scipy.stats import norm
import statsmodels.api as sm

# 독립성을 위배하는 데이터 생성
np.random.seed(0)
n = 100
x = np.random.randint(0, 10, size=n)
y = np.zeros_like(x)
for i in range(1, n):
    y[i] = 0.5 * y[i-1] + x[i] + np.random.normal()

# Durbin-Watson 통계량 계산 (차분 전)
dw_statistic_original = sm.stats.stattools.durbin_watson(y)
print("Durbin-Watson 통계량 (차분 전):", dw_statistic_original)

# Durbin-Watson 가설검정 함수 정의
def durbin_watson_hypothesis(dw_statistic, n):
    # 잔차의 자유도
    k = 1
    # Durbin-Watson 임계값 (alpha = 0.05)
    z_alpha_over_2 = norm.ppf(1 - 0.05 / 2)
    dw_crit_lower = 2 - (2 * k / n) * z_alpha_over_2 * np.sqrt((k + 1) / n)
    dw_crit_upper = 2 + (2 * k / n) * z_alpha_over_2 * np.sqrt((k + 1) / n)
    if dw_statistic < dw_crit_lower:
        return "귀무가설 기각: 양의 자기상관이 존재합니다."
    elif dw_statistic > dw_crit_upper:
        return "귀무가설 기각: 음의 자기상관이 존재합니다."
    else:
        return "귀무가설 채택: 자기상관이 존재하지 않습니다."

# Durbin-Watson 가설검정 수행 (차분 전)
print("Durbin-Watson 가설검정 (차분 전):",
durbin_watson_hypothesis(dw_statistic_original, n))
```

```
# 차분
y_diff = np.diff(y)

# Durbin-Watson 통계량 계산 (차분 후)
dw_statistic_diff = sm.stats.stattools.durbin_watson(y_diff)
print("Durbin-Watson 통계량 (차분 후):", dw_statistic_diff)

# Durbin-Watson 가설검정 수행 (차분 후)
print("Durbin-Watson 가설검정 (차분 후):",
durbin_watson_hypothesis(dw_statistic_diff, n - 1))
# 상한 값(약 2.005)보다 큰 값이라 음의 자기상관이 존재한다고 하나 2 에 매우 가까운
값이므로 무시가능한 정도
```

```
Durbin-Watson 통계량 (차분 전): 0.16537731717997373
Durbin-Watson 가설검정 (차분 전): 귀무가설 기각: 양의 자기상관이 존재합니다.
Durbin-Watson 통계량 (차분 후): 2.081200353045013
Durbin-Watson 가설검정 (차분 후): 귀무가설 기각: 음의 자기상관이 존재합니다.
```

3) 등분산성 진단

① 잔차와 예측값 간의 산점도 확인

산점도가 균등하게 분포되지 않은 경우 등분산성 가정이 위배되었다고 판단

② 바틀렛(Breusch-Pagan) 검정

잔차의 분산이 독립 변수들과 관련이 있는지 검정

⇒ p-value 가 유의수준보다 작으면 등분산성 가정이 위배되었다고 판단

등분산성 가정이 위배된 경우 (Heteroscedasticity) : Box-Cox 변환, 로그변환 등을 통해 등분산성 유도 가능

* Box-Cox transformation

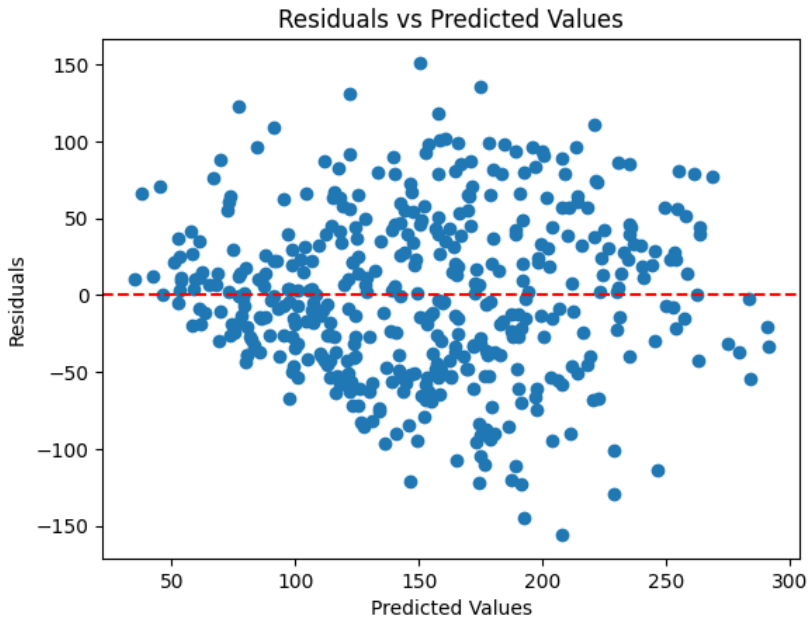
: 등분산성을 가정하는 회귀 분석 모델에서 종속 변수의 변환을 수행하는 방법으로 아래와 같이 변환 파라미터 λ에 의해 변환 형태가 달라짐

$$\begin{cases} \dfrac{y^{\lambda}-1}{\lambda} & \lambda \neq 0 \\ \log(y) & \lambda = 0 \end{cases}$$

Box-Cox transformation 으로도 등분산성 가정이 만족되지 않는 경우에는 제곱근 변환 등의 종속 변수 변환 혹은 가중 회귀(weighted regression) 등을 통해 해결 가능

[등분산성 진단 코드]

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.datasets import load_diabetes

# 데이터 불러오기
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# 모델 학습
X = sm.add_constant(X)  # 상수항 추가
model = sm.OLS(y, X).fit()

# 예측값과 잔차 계산
predicted_values = model.predict(X)
residuals = model.resid

# 잔차와 예측값 사이의 산포도 그리기
plt.scatter(predicted_values, residuals)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted Values")
plt.axhline(y=0, color='red', linestyle='--')  # 잔차가 0 인 수평선 추가
plt.show()
```



4) 정규성 진단

① 잔차 히스토그램 확인

히스토그램이 종모양을 나타내지 않는 경우 정규성 가정이 위배되었다고 판단

② Q-Q Plot(정규 확률 도표) 확인

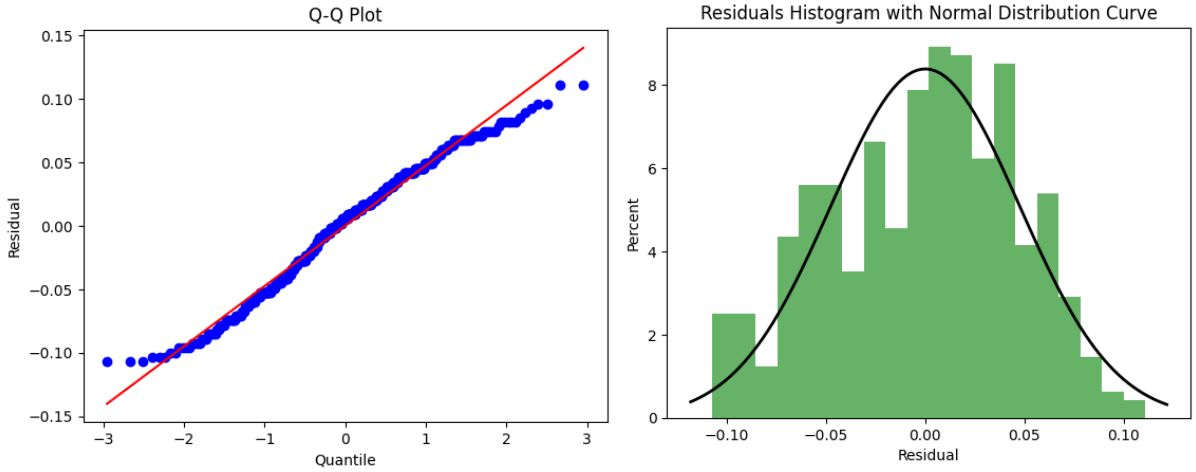
Plot 이 직선에 가깝지 않다면 정규성 가정이 위배되었다고 판단

정규성이 위배된 경우 : 로그 변환, 제곱근 변환, Box-Cox 변환 등을 통해 잔차를 변환하여 정규성 유도 가능

[정규성 진단 코드]

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.datasets import load_diabetes

# 당뇨병 데이터셋 불러오기
diabetes = load_diabetes()
data = diabetes.data[:, 0]  # 당뇨병 데이터셋의 첫 번째 열 선택 (임의로 선택)

# Q-Q plot 그리기
stats.probplot(data, dist="norm", plot=plt)
plt.title('Q-Q Plot')
plt.xlabel('Quantile')
plt.ylabel('Residual')
plt.show()

# 잔차 히스토그램 그리기
residuals = data - np.mean(data)
plt.hist(residuals, bins=20, density=True, alpha=0.6, color='g')
plt.title('Residuals Histogram with Normal Distribution Curve')
plt.xlabel('Residual')
plt.ylabel('Percent')

# 정규분포 곡선 그리기
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = stats.norm.pdf(x, np.mean(residuals), np.std(residuals))
plt.plot(x, p, 'k', linewidth=2)
```

```
plt.show()
```



5. 로지스틱 회귀 (Logistic Regression)

1) 로지스틱 회귀

종속 변수가 이항인 경우에 사용되는 분류방법론으로 종속변수를 0 과 1 사이로 산출하게 하는 로지스틱 함수를 사용하여 분류모델에서 주로 사용.

- 로지스틱 회귀모형 특징

① 회귀모형과 오차항에 대한 정규성, 등분산성, 선형성 가정이 없음

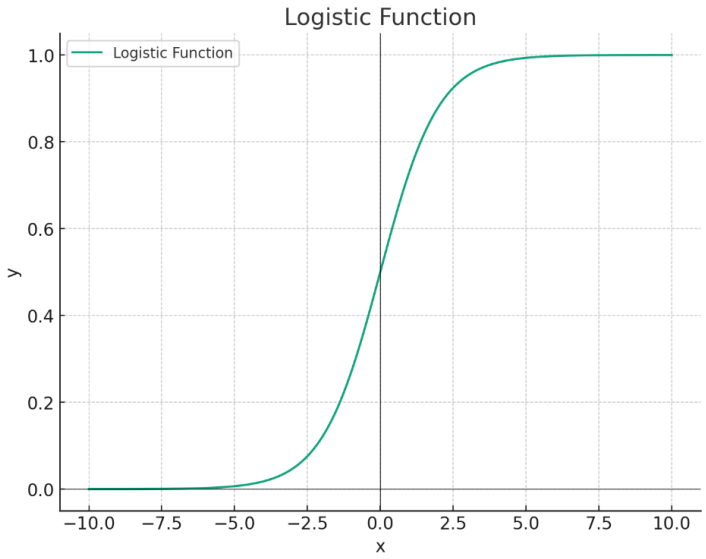
② 소표본인 경우 적합이 잘 되지 않을 수 있음

③ 설명 변수 간 척도(scale) 차이가 큰 경우 모형 접합(적합, 훈련)이 잘 되지 않음

[1] 로지스틱 함수

인수로 -∞와 ∞ 사이의 값을 받을 수 있고 결과로 0 과 1 사이의 값을 반환하는 함수로 정확하게 확률 표현이 가능

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$



[2] 오즈(odds)

성공 확률을 실패 확률로 나눈 비를 나타내는 값으로 확률에 비해 더 직관적이고 이해하기 쉬운 척도이며 0 과 ∞ 사이의 값을 가짐

$$\frac{p(x)}{1-p(x)} = e^{\beta_0+\beta_1 X}$$

[3] 로그-오즈(log-odds) or 로짓(logit)

$$\ln\frac{p(x)}{1-p(x)} = \beta_0 + \beta_1 X$$

선형 회귀와는 달리 X 가 한 단위 증가할 때 로그-오즈가 β_1만큼 증가

* 다중 로지스틱 회귀의 경우

$$\ln\frac{p(x)}{1-p(x)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

2) 회귀계수 추정 및 적합도 검정

[1] 회귀계수 추정

* 최대우도 추정법 (Maximum Likelihood Estimation, MLE)

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1-p(x_{i'}))$$

관측된 데이터가 주어진 모델 파라미터에서 나올 가능성(우도, Likelihood)을 최대화하는 모수값을 찾는 것으로 추정치 $\hat{\beta}_0$와 $\hat{\beta}_1$은 위의 함수를 최대화하도록 선택됨.

[2] 회귀계수에 대한 가설 검정

주로 Wald 검정 사용

① Wald 검정 (Wald Test)

$$H_0 : \beta_1 = 0 \quad \text{vs} \quad H_1 : \beta_1 \neq 0$$

$$\text{검정통계량} : W = \frac{(\hat{\beta}_1)^2}{SE(\hat{\beta}_1)^2}$$

$\hat{\beta}_1$: 회귀계수 추정치 , $SE(\hat{\beta}_1)^2$: 회귀계수 추정치의 표준오차

Wald Test 에서 p-value < 유의수준($\alpha = 0.05$)이면 귀무가설 $H_0 : \beta_1 = 0$ 기각 ⇒ 회귀계수가 통계적으로 유의함

② 우도비 검정 (Likelihood Ratio Test)

전체 모델(모든 예측 변수 포함)과 축소 모델(검정하고자 하는 변수를 제외한 모델) 사이의 우도 비율을 계산하여 사용하며 변수가 여러 개일 때 유용함

③ 점수 검정 (Score Test)

모델 파라미터가 귀무가설 하에 최대우도추정치 근처에 있을 때 사용하며 회귀계수 추정치 없이도 구할 수 있어 유용하지만 복잡한 모델에서는 계산이 어려움

[3] 회귀 모형에 대한 가설 검정

주로 우도비 검정 사용

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0 \quad \text{vs} \quad H_1 : Some\ of\ \beta_j \neq 0 \quad (j = 1, \cdots, p)$$

$$\text{검정통계량} : LR = -2 \ln \frac{L_0}{L_1} \sim \chi^2{}_{df}$$

L_0 : 축소 모델의 우도 , L_1 : 전체 모델의 우도, df : 전체모델과 축소 모델 간의 계수(변수) 차이

모델의 로그 우도가 높을수록 모델이 더 좋은 것으로 간주되며 모델 간 로그 우도의 차이가 클수록(검정 통계량이 클수록) 축소 모델과 전체 모델 사이에 유의한 차이가 있음을 나타내고 H_0기각.

우도비 검정에서 p-value < 유의수준($\alpha = 0.05$)이면 귀무가설 $H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$ 기각 ⇒ 회귀모형이 통계적으로 유의함.

[로지스틱 회귀 모형 검정 코드]

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy.stats import chi2

# 당뇨병 데이터셋 불러오기
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()

# 설명 변수와 목표 변수 설정
X = diabetes.data
y = (diabetes.target > 140).astype(int)  # 혈당 수치가 140 이상인 경우를 당뇨병 환자로 설정

# 상수항 추가
X = sm.add_constant(X)

# 전체 모델 피팅
model_full = sm.Logit(y, X).fit()

# 축소 모델 피팅 (ex: 첫 번째 설명 변수 제외)
X_reduced = X[:, 1:]  # 첫 번째 설명 변수를 제외한 것으로 축소 모델 생성
model_reduced = sm.Logit(y, X_reduced).fit()

# 우도비 검정 통계량 계산
LR_statistic = -2 * (model_reduced.llf - model_full.llf)

# 자유도 계산
df = model_full.df_model - model_reduced.df_model

# 유의확률 계산
p_value = 1 - chi2.cdf(LR_statistic, df)

# 유의수준 (예: 0.05) 설정
alpha = 0.05

# 유의수준과 유의확률 비교하여 가설검정 수행
if p_value < alpha:
    print("전체 모델이 축소 모델과 유의하게 다르다. (기각)")
else:
    print("전체 모델이 축소 모델과 유의하게 다르지 않다. (채택)")
```

```
Optimization terminated successfully.
         Current function value: 0.473950
         Iterations 7
Optimization terminated successfully.
```

Current function value: 0.473951

Iterations 7

전체 모델이 축소 모델과 유의하게 다르지 않다. (채택)

[로지스틱 회귀 실습 코드]

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# 데이터 불러오기
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# 이진 분류를 위해 목표 변수를 1(당뇨병 환자) 또는 0(당뇨병 환자가 아님)으로 바꿉니다.
y_binary = (y > 140).astype(int)

# 데이터를 훈련 세트와 테스트 세트로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y_binary,
test_size=0.2, random_state=42)

# 데이터 표준화
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 로지스틱 회귀 모델 학습
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# 테스트 데이터에 대한 예측
y_pred = model.predict(X_test_scaled)

# 정확도 출력
accuracy = accuracy_score(y_test, y_pred)
print("정확도:", accuracy)

# 분류 보고서 출력
print(classification_report(y_test, y_pred))
```

```
정확도: 0.7303370786516854
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        49
           1       0.69      0.72      0.71        40

    accuracy                           0.73        89
   macro avg       0.73      0.73      0.73        89
weighted avg       0.73      0.73      0.73        89
```

- accuracy(정확도) : 모델이 전체 샘플 중 올바르게 예측한 비율

- precision(정밀도) : 모델이 1 로 예측한 결과 중 실제로 1 인 샘플의 비율, 모델이 0 으로 예측한 결과 중 실제로 0 인 샘플의 비율

- recall(재현율) : 실제로 1 인 샘플 중 모델이 1 로 정확하게 예측한 비율, 실제로 0 인 샘플 중 모델이 0 으로 정확하게 예측한 비율

- F1-score : 정밀도와 재현율의 조화평균으로 1 에 가까울수록 좋은 성능

- support : 각 클래스의 실제 샘플 수