**회귀분석**

--- 회귀분석에서 젤 먼저 해야 할 것은 plot 그려서 회귀분석에 적합할 지 확인

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 | 종속변수 | 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)

- : Independent and Identically Distributed random variable

- 회귀계수(regression coefficient)

: 기울기(slope) , : 절편(intercept)

- : 번째 관측치 오차

**\* 오차항의 4가지 가정**

① 선형성(linearity) : 종속 변수 와 독립변수 는 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)

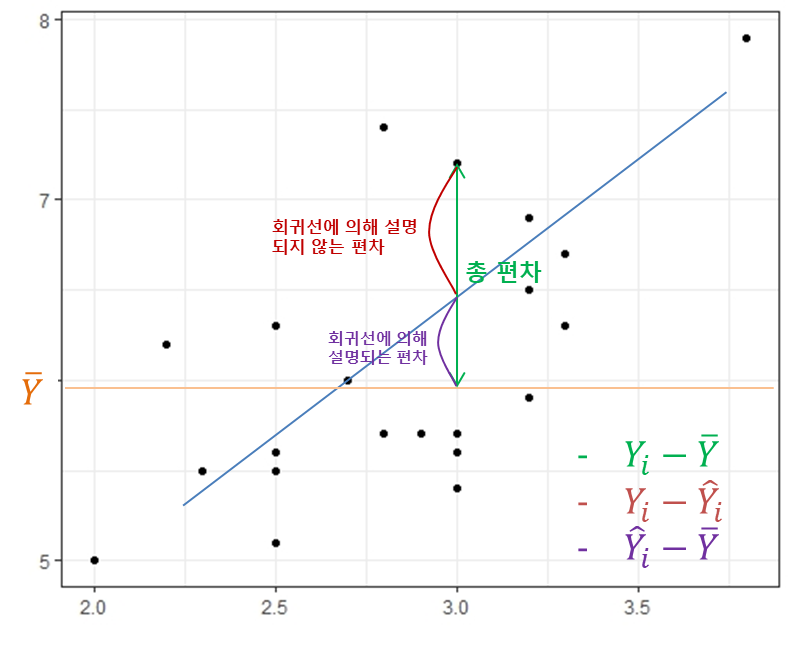
- 최소제곱법(least square method)

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

오차의 제곱합 식인 을 각 회귀계수에 대해 편미분하여 그 값이 0이 되는 과 의 값을 도출하면 아래와 같음

추정된 회귀계수를 통해 회귀식을 추정하면 아래와 같음

**\* 잔차(residual)**

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

TSS = SSR = SSE =

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 |  |  |
| 잔차(Residual) | n-2 | SSE |  |  |
| 총(Total) | n-1 | SST |  |  |

검정통계량 : 기각역 :

2) 유의성 검정

[1] 모형 유의성 검정

회귀식에서 기울기()가 0이면 모형이 유의하지 않음. 이를 확인하기 위해 가설검정을 하면 귀무가설과 채택가설이 아래와 같음

vs

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()이면 귀무가설 기각 ⇒ 모형이 통계적으로 유의함

[2] 회귀계수 유의성 검정

각 변수의 중요도 및 유의성 파악 가능하며 이를 확인하기 위해 가설검정을 하면 귀무가설과 채택가설이 아래와 같음

vs

검정통계량 : (단, ) 기각역 :

통계패키지 결과 Parameter Estimates table을 얻을 수 있음

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter Estimates | | | | | |
| Variable | DF | Parameter Estimate | Standard Error | t-value | Pr>|t| |
| Intercept | 1 |  | S.E | T | p-value |
| 변수명 | 1 |  | S.E | T | p-value |

변수의 p-value < 유의수준()이면 귀무가설 기각 ⇒ 회귀계수가 통계적으로 유의함

3) 회귀모형 평가

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

: 실제 값과 예측 값 간의 차이를 절대값으로 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)

: 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 값의 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.

유클리디안 거리(제곱)랑 맨해튼 거리(절대값)에 관해서 공부하렴

[3] 결정계수(coefficient of determination)

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

**\* 이면 총 편차 중 80%가 모형에 의해 설명할 수 있음**

[3] 수정 결정계수

: 관측치 수 , : 모형에 포함된 설명 변수의 수

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

Rsquare랑 adjRsquare값이 비슷하면 왜 좋냐

[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 OLS는 선형회귀모델이 상수항의 존재를 포착할 수 있도록 1을 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

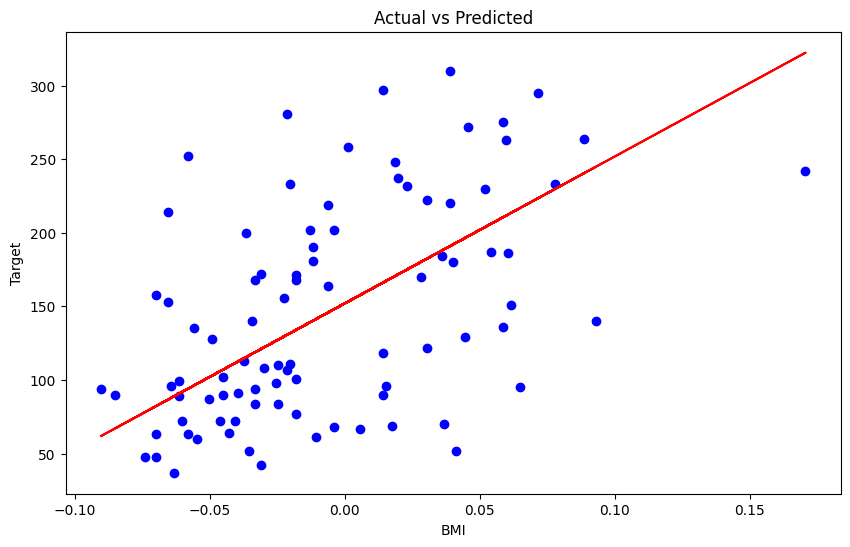
===========================================================================

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)

**\* 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

회귀식에서 기울기가 0이면 모형이 유의하지 않음. 이를 확인하기 위해 가설검정을 하면 귀무가설과 채택가설이 아래와 같음

vs

[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 |  |  |
| 잔차(Residual) | n-p-1 | SSE |  |  |
| 총(Total) | n-1 | SST |  |  |

검정통계량 : 기각역 :

F-test에서 p-value < 유의수준()이면 귀무가설 기각 ⇒ 모형이 통계적으로 유의함

[다중 선형 회귀 실습]

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

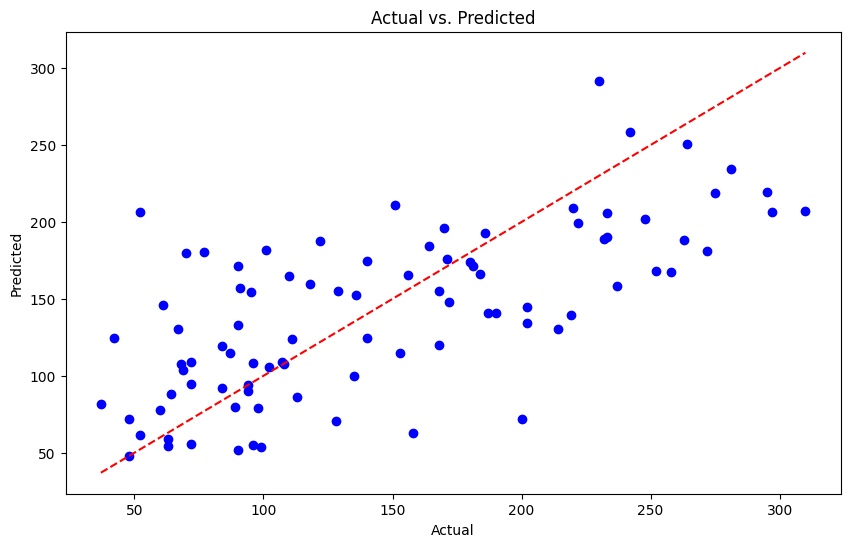
===========================================================================

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 1의 값으로 변환한 변수

ex) 인 범주형 변수를 선형회귀모델에서 다루는 경우 (범주형 변수 1개)

위와 같이 범주형 변수 를 dummy 변수 로 변환하여 활용함. 변환한 dummy 변수를 활용한 회귀 모델은 다음과 같음.

-

⇒ ( (

- (Female : base line)

-

ex) 인 범주형 변수를 선형회귀모델에서 다루는 경우 (범주형 변수 2개)

위와 같이 2개의 범주형 변수를 dummy 변수 , ,로 변환하여 활용하며 이 3개의 변수 중 하나만 고려하거나 하나만 제외하고 분석할 수 없음. 변환한 dummy변수를 활용한 회귀 모델은 다음과 같음.

-

⇒ , ,

, ,

- (Red and Male : base levels)

[범주형 변수가 포함된 선형 회귀 실습]

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, r2\_score

# 당뇨병 데이터셋 로드

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target

# 데이터셋을 판다스 DataFrame으로 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 (평균 제곱 오차와 R-Squared 값 사용)

mse = mean\_squared\_error(y\_test, y\_pred)

r\_squared = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R-Squared:", r\_squared)

# 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

R-Squared: 0.444471463380707

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

를 나머지 독립변수들의 선형결합으로 설명한 결정계수

⇒ 와 다른 독립 변수들 간 상관관계↑ ⇒ ↑ ⇒ VIF↑

VIF가 10이상일 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-value가 유의수준()보다 작으면서 그 값이 가장 작은 변수를 선택하여 모델에 추가

③ p-value가 유의수준보다 작은 변수가 없을 때까지 반복

[전진 선택 실습]

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-value가 유의수준()보다 크면서 그 값이 가장 큰 변수를 선택하여 모델에서 제거

③ p-value가 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 실습]

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]

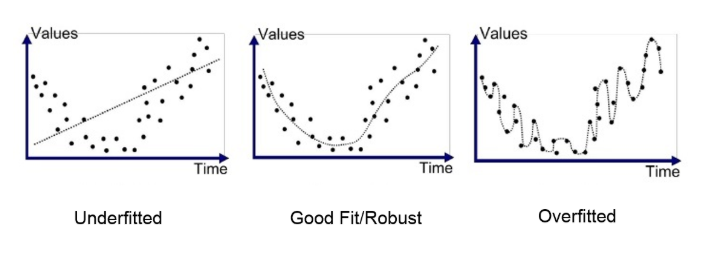
[3] 단계적 선택 (Stepwise Selection)

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 penalty를 가진 선형 회귀 방법 (L1 규제 사용)

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

\* 라쏘 회귀의 목적

값이 최소가 되도록 하는 (가중치)와 (절편)값 찾기

- 값↑ ⇒ 규제↑, 가중치↓ ⇒ 과소적합(underfitting)

- 값↓ ⇒ 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 penalty를 가진 선형 회귀 방법 (L2 규제 사용)

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

\* 릿지 회귀의 목적

값이 최소가 되도록 하는 (가중치)와 (절편)값 찾기

- 값↑ ⇒ 규제↑, 가중치↓ ⇒ 과소적합(underfitting)

- 값↓ ⇒ 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-squared로

r\_squared = r2\_score(y\_test, y\_pred)

print("R-squared:", r\_squared)

print("릿지 회귀 모델의 계수:", ridge.coef\_)

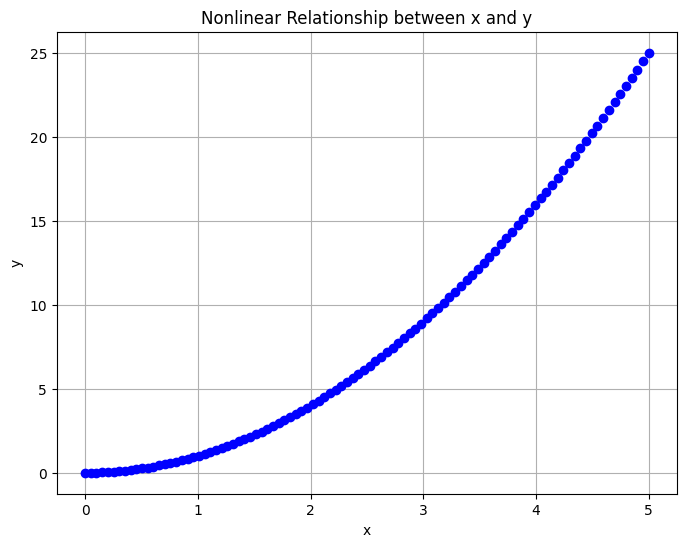
R-squared: 0.46085219464119265

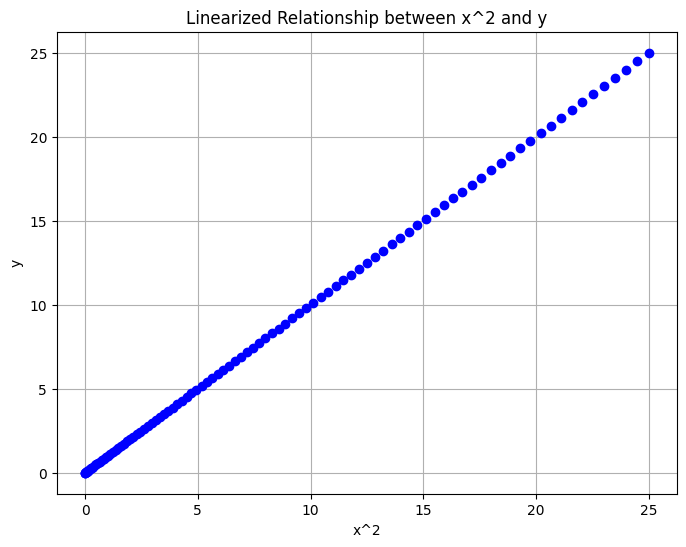
릿지 회귀 모델의 계수: [ 42.85566976 -205.49431899 505.08903304 317.0932049 -108.50026183

-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 | ARMA와 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')

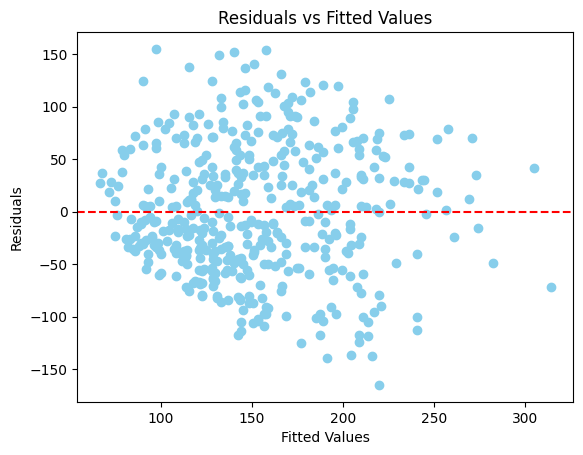
plt.title('Residuals vs Fitted Values')

plt.xlabel('Fitted Values')

plt.ylabel('Residuals')

plt.axhline(y=0, color='red', linestyle='--')  # 잔차가 0인 수평선 추가

plt.show()



import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# 비선형 데이터 생성 (여기서는 2차 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 계산 (변환된 변수 사용)

residuals\_transformed = y - y\_pred\_transformed

# 잔차 산점도 그리기

plt.figure(figsize=(12, 6))

# 원본 데이터의 잔차

plt.subplot(1, 2, 1)

plt.scatter(X, residuals)

plt.xlabel('Feature')

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(X\_transformed, residuals\_transformed)

plt.xlabel('Transformed Feature')

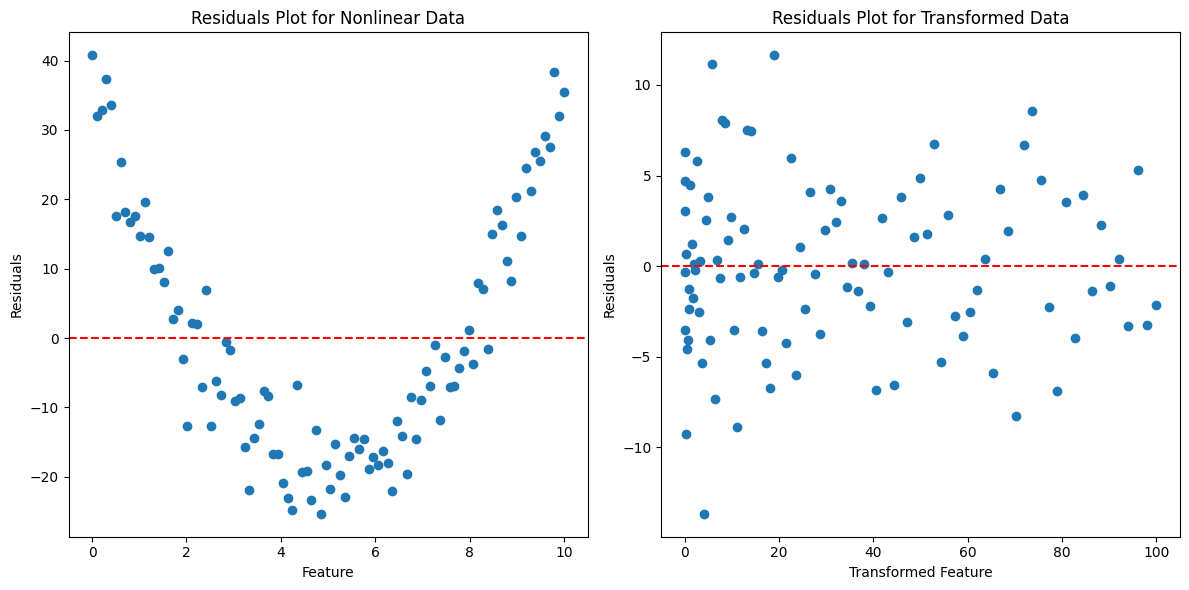
plt.ylabel('Residuals')

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 0과 4 사이의 값을 가짐.

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

- 2에 가까운 값 : 잔차들이 서로 독립적임. (자기상관 없음)

- 4에 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

--- 차분을 왜 하냐 (자기상관 때문?)

[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

# 단순선형회귀를 위해 3번째 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 2에 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-value가 유의수준보다 작으면 등분산성 가정이 위배되었다고 판단

등분산성 가정이 위배된 경우 (Heteroscedasticity) : Box-Cox 변환을 통해 등분산성 유도 가능

\* Box-Cox transformation

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

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

[등분산성 진단 코드]

---가중회귀 추가 (GLS)

4) 정규성 진단

① 잔차 히스토그램 확인

히스토그램이 종모양을 나타내지 않는 경우 정규성 가정이 위배되었다고 판단

② Q-Q Plot(정규 확률 도표) 확인

Plot이 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 및 정규성 유도 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()

# Shapiro-Wilk 테스트를 사용하여 정규성 검정

statistic, p\_value = stats.shapiro(data)

# 결과 출력

print("Shapiro-Wilk Test 결과")

print("통계량 (Test Statistic):", statistic)

print("p-value:", p\_value)

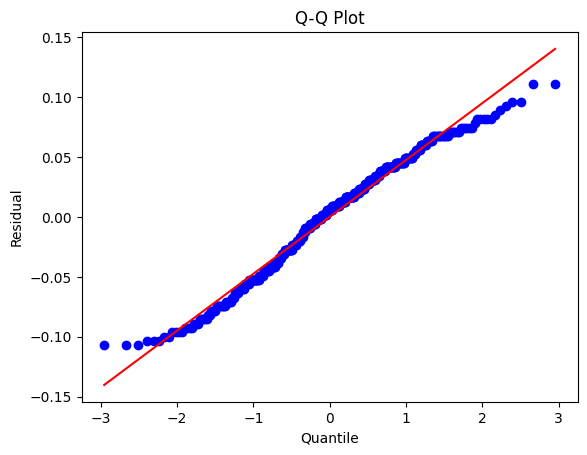
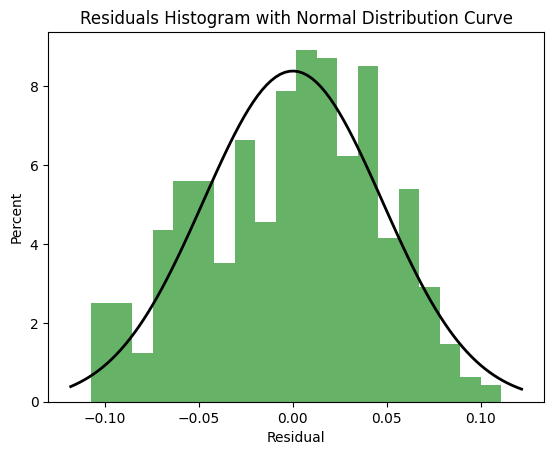
# p-value를 이용한 결과 해석

alpha = 0.05

if p\_value > alpha:

    print("데이터는 정규분포를 따릅니다 (귀무가설을 기각할 수 없음)")

else:

    print("데이터는 정규분포를 따르지 않습니다 (귀무가설 기각)")

Shapiro-Wilk Test 결과

통계량 (Test Statistic): 0.9823583960533142

p-value: 3.27964844473172e-05

데이터는 정규분포를 따르지 않습니다 (귀무가설 기각)

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

import seaborn as sns

# Iris 데이터셋 로드

iris = load\_iris()

data = iris.data[:, 0]  # Iris 데이터셋에서 첫 번째 특성을 사용

# QQ plot과 잔차 히스토그램 그리기

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# QQ plot 그리기

stats.probplot(data, dist="norm", plot=ax1)

ax1.set\_title('Q-Q Plot')

ax1.set\_xlabel('Theoretical Quantiles')

ax1.set\_ylabel('Sample Quantiles')

# 잔차 히스토그램 그리기

residuals = data - np.mean(data)

sns.histplot(residuals, kde=True, ax=ax2)

ax2.set\_title('Residuals Histogram')

ax2.set\_xlabel('Residuals')

ax2.set\_ylabel('Frequency')

plt.show()

# Shapiro-Wilk 검정을 사용한 정규성 검정

statistic, p\_value = stats.shapiro(data)

print("Shapiro-Wilk 검정 결과:")

print(f"Statistic: {statistic}, p-value: {p\_value}")

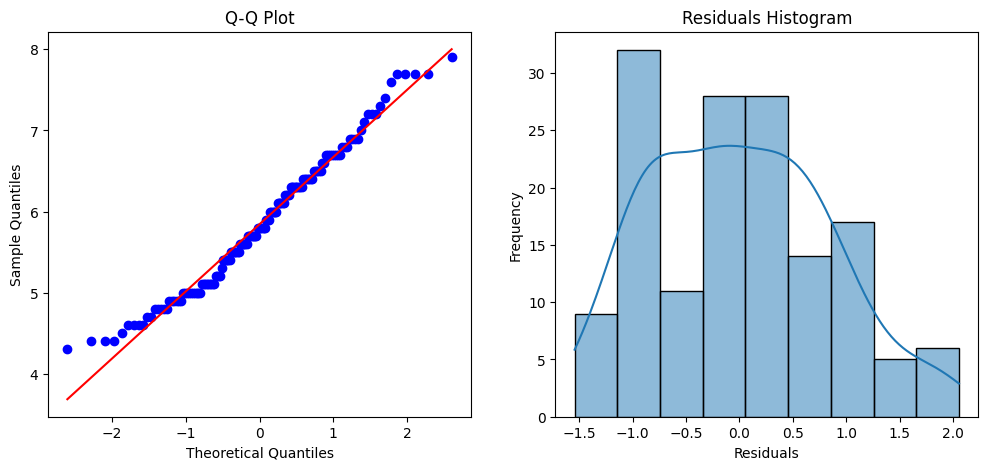
# p-value가 유의수준(일반적으로 0.05)보다 작으면 정규성을 기각할 수 있음

if p\_value < 0.05:

    print("데이터가 정규 분포를 따르지 않음")

else:

    print("데이터가 정규 분포를 따름")



Shapiro-Wilk 검정 결과:

Statistic: 0.9760899543762207, p-value: 0.01018026564270258

데이터가 정규 분포를 따르지 않음

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from scipy.stats import boxcox

# Iris 데이터셋 로드

iris = load\_iris()

data = iris.data[:, 0]  # Iris 데이터셋에서 첫 번째 특성을 사용

# Box-Cox 변환

transformed\_data, lambda\_val = boxcox(data)

# Box-Cox 변환 시 사용된 lambda 값 출력

print("Lambda 값:", lambda\_val)

# QQ plot 그리기

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

stats.probplot(transformed\_data, dist="norm", plot=axes[0])

axes[0].set\_title('Q-Q Plot after Box-Cox Transformation')

axes[0].set\_xlabel('Theoretical Quantiles')

axes[0].set\_ylabel('Sample Quantiles')

# 잔차 히스토그램 그리기

axes[1].hist(transformed\_data, bins=20, density=True, color='skyblue', edgecolor='black') # 밀도 플로팅을 위해 density=True 설정

axes[1].set\_title('Histogram of Transformed Data with Kernel Density Estimation')

axes[1].set\_xlabel('Transformed Values')

axes[1].set\_ylabel('Density')

# 밀도 추정 곡선 추가

xmin, xmax = axes[1].get\_xlim()

x = np.linspace(xmin, xmax, 100)

p = stats.norm.pdf(x, np.mean(transformed\_data), np.std(transformed\_data))

axes[1].plot(x, p, 'k', linewidth=2)

plt.show()

# Shapiro-Wilk 검정을 사용한 정규성 검정

statistic, p\_value = stats.shapiro(transformed\_data)

print("Shapiro-Wilk 검정 결과:")

print(f"Statistic: {statistic}, p-value: {p\_value}")

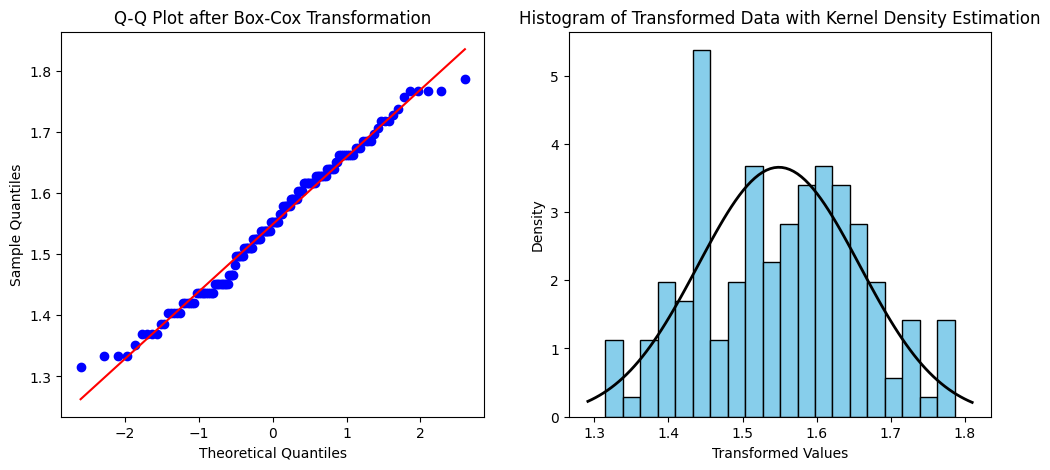
# p-value가 유의수준(일반적으로 0.05)보다 작으면 정규성을 기각할 수 있음

if p\_value < 0.05:

    print("데이터가 정규 분포를 따르지 않음")

else:

    print("데이터가 정규 분포를 따름")

Lambda 값: -0.14475080273816873

Shapiro-Wilk 검정 결과:

Statistic: 0.9825820326805115, p-value: 0.05463974177837372

데이터가 정규 분포를 따름

5. 로지스틱 회귀 (Logistic Regression)

1) 로지스틱 회귀

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

- 로지스틱 회귀모형 특징

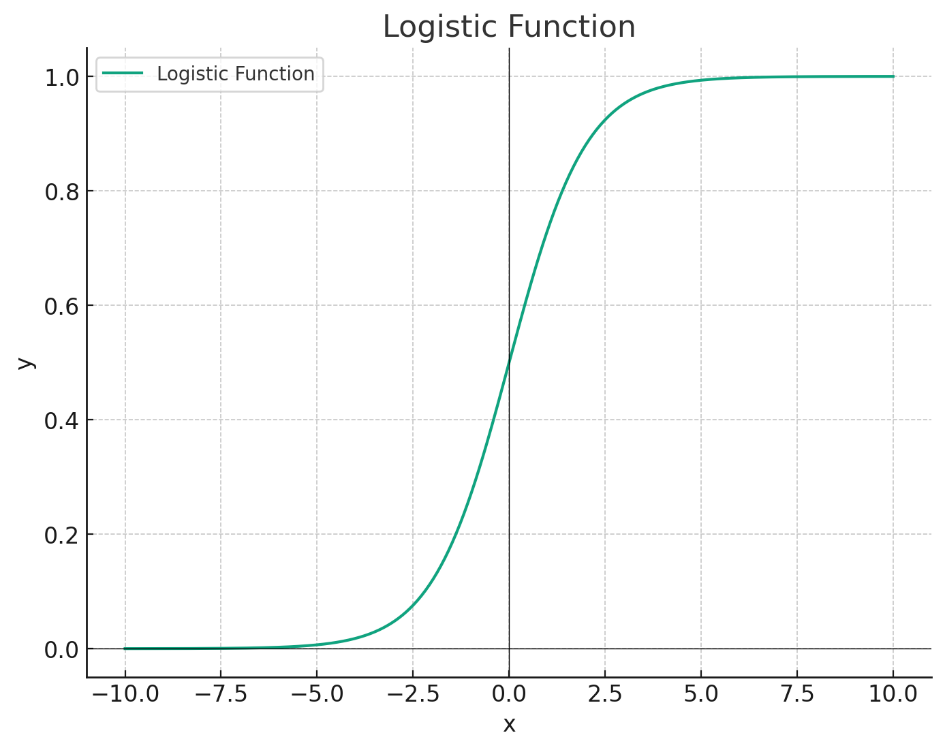
① 회귀모형과 오차항에 대한 정규성, 등분산성, 선형성 가정이 없음

② 소표본인 경우 적합이 잘 되지 않을 수 있음

③ 설명 변수 간 척도(scale) 차이가 큰 경우 모형 접(적?)합이 잘 되지 않음

[1] 로지스틱 함수

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



[2] 오즈(odds)

성공 확률을 실패 확률로 나눈 비를 나타내는 값으로 0과 ∞ 사이의 값을 가짐

[3] 로그-오즈(log-odds) or 로짓(logit)

선형 회귀와는 달리 X가 한 단위 증가할 때 로그-오즈가 만큼 증가

\* 다중 로지스틱 회귀의 경우

--- 오즈가 성능 평가 때문인지 베타 추정 때문인지 수식 설명 필요

2) 회귀계수 추정 및 적합도 검정

[1] 회귀계수 추정

--- MLE가 뭔지 수식이랑 다 공부해

\* 최대우도 추정법 (Maximum Likelihood Estimation, MLE) \*\*\*\*\*\*\*\*\*

관측된 데이터가 주어진 모델 파라미터에서 나올 가능성(우도, Likelihood)을 최대화하는 모수값을 찾는 것으로 추정치 와 은 위의 함수를 최대화하도록 선택됨.

[2] 회귀계수에 대한 가설 검정

주로 Wald 검정 사용

① Wald 검정 (Wald Test)

vs

검정통계량 :

: 회귀계수 추정치 , : 회귀계수 추정치의 표준오차

Wald Test에서 p-value < 유의수준()이면 귀무가설 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

카이 쓴다는 거 찾아봐

주로 우도비 검정 사용

vs

: 축소 모델의 우도 , : 전체 모델의 우도

우도비 검정에서 p-value < 유의수준()이면 귀무가설 기각 ⇒ 회귀모형이 통계적으로 유의함

[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 1로 예측한 결과 중 실제로 1인 샘플의 비율, 모델이 0으로 예측한 결과 중 실제로 0인 샘플의 비율

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

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

- support : 각 클래스의 실제 샘플 수

검정통계량으로 T와 F를 회귀에서 왜 보여주는지