선형 회귀 (Linear Regression)

- 하나 이상의 독립 변수(X)와 **연속형 종속 변수(Y)** 간의 선형 관계를 모델링하여, 주어진 독립 변수 값으로 종속 변수 값을 예측하는 기법입니다.
- 모델은 예측값과 실제값의 차이(잔차)의 제곱합을 최소화하는 직선(또는 초평면)을 찾습니다. (최소제곱법, OLS)
- **단순 선형 회귀**: $Y = \beta_0 + \beta_1 X + \varepsilon$ (독립 변수가 1개)
- **다중 선형 회귀**: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$ (독립 변수가 여러 개)
    - Y: 종속 변수
    - X: 독립 변수
    - β_0: 절편 (intercept)
    - β_1: 기울기 (coefficient, X가 1단위 증가할 때 Y의 변화량)
    - ε: 오차항 (error term)

적용 가능한 상황

- 연속형 변수를 예측하고 싶을 때 (예: 주택 가격, 매출액, 온도 예측)
- 독립 변수가 종속 변수에 미치는 영향을 해석하고 설명하고 싶을 때 (영향의 크기 및 방향성 파악)

구현 방법

선형 회귀는 주로 `statsmodels`와 `scikit-learn` 라이브러리로 구현됩니다. 두 라이브러리는 목적이 다소 다릅니다.

- **`statsmodels`**: 통계적 추론 및 모델 해석에 중점을 둡니다. 회귀 계수의 유의성, 모델의 설명력 등 상세한 통계 리포트를 제공합니다.
- **`scikit-learn`**: 예측(Prediction)에 중점을 둡니다. 모델을 쉽게 만들고, 교차 검증, 하이퍼파라미터 튜닝 등 머신러닝 파이프라인에 통합하기 용이합니다.

선형 회귀의 4대 기본 가정

선형 회귀 모델의 결과를 신뢰하기 위해 만족해야 하는 중요한 가정들입니다. 회귀 진단 시 이 가정들이 충족되었는지 반드시 확인해야 합니다.

1. **선형성 (Linearity)**: 독립 변수와 종속 변수 간의 관계는 선형적이어야 한다.
    - **확인 방법**: 예측값(Fitted values)과 잔차(Residuals)를 산점도로 그려, 잔차들이 0을 중심으로 무작위로 흩어져 있는지 확인합니다. 패턴(예: 곡선 형태)이 보이면 선형성 가정을 위반한 것입니다.
2. **잔차의 정규성 (Normality of Residuals)**: 잔차(오차)는 정규 분포를 따라야 한다.
    - **확인 방법**: 잔차의 히스토그램, Q-Q 플롯을 확인하거나, Shapiro-Wilk 검정 등 정규성 검정을 수행합니다.
3. **잔차의 등분산성 (Homoscedasticity)**: 잔차의 분산은 모든 독립 변수 값에 대해 일정해야 한다.
    - **확인 방법**: 예측값과 잔차의 산점도에서, 잔차들이 모든 예측값 구간에서 비슷한 폭으로 흩어져 있는지 확인합니다. 깔때기 모양 등 특정 패턴을 보이면 등분산성 가정을 위반한 것입니다. (이분산성, Heteroscedasticity)

4. **잔차의 독립성 (Independence of Residuals)**: 잔차들은 서로 독립적이어야 한다. (자기상관이 없어야 한다)
    - **확인 방법**: Durbin-Watson 통계량을 확인합니다. (주로 시계열 데이터에서 중요)
        - 2에 가까우면 독립적, 0에 가까우면 양의 자기상관, 4에 가까우면 음의 자기상관을 의심합니다.

---

1. `statsmodels.OLS`

- **용도**: 모델의 통계적 의미를 상세히 분석하고 해석하는 데 사용됩니다. (R-squared, 각 계수의 t-검정 p-value, F-통계량 등)
- **주의사항**: `scikit-learn`과 달리 절편(intercept)을 자동으로 추가하지 않으므로, `sm.add_constant()` 함수를 사용하여 독립 변수 데이터에 상수항(절편)을 직접 추가해야 합니다.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm

# 예제 데이터 생성
np.random.seed(42)
X1 = np.random.rand(100) * 10
X2 = np.random.rand(100) * 5
y = 3 + 2*X1 + 5*X2 + np.random.randn(100) * 2 # Y = 3 + 2*X1 + 5*X2 + error
data = pd.DataFrame({'X1': X1, 'X2': X2, 'y': y})

X = data[['X1', 'X2']]
y = data['y']

# 1. 상수항(절편) 추가
X_with_const = sm.add_constant(X)

# 2. OLS 모델 적합
# OLS(종속변수, 독립변수)
model = sm.OLS(y, X_with_const)
results = model.fit()

# 3. 모델 결과 요약 출력
print(results.summary())
'''
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.959
Model:                            OLS   Adj. R-squared:                  0.958
Method:                 Least Squares   F-statistic:                     1143.
Date:                Fri, 10 Oct 2025   Prob (F-statistic):           3.70e-68
Time:                        15:06:50   Log-Likelihood:                -208.42
No. Observations:                 100   AIC:                             422.8
Df Residuals:                      97   BIC:                             430.6
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
```

```
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.8212      0.508      5.549      0.000       1.812       3.830
X1             1.9317      0.067     28.937      0.000       1.799       2.064
X2             5.2877      0.136     39.023      0.000       5.019       5.557
==============================================================================
Omnibus:                        6.139   Durbin-Watson:                   2.073
Prob(Omnibus):                  0.046   Jarque-Bera (JB):                5.737
Skew:                           0.456   Prob(JB):                       0.0568
Kurtosis:                       3.738   Cond. No.                         15.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
'''
```

결과 해석 방법 (`results.summary()`)

- **R-squared (결정계수)**: 모델이 종속 변수의 분산을 얼마나 설명하는지를 나타냅니다. 0~1 사이 값이며, 1에 가까울수록 설명력이 높습니다.
- **Adj. R-squared (조정된 결정계수)**: 독립 변수의 개수를 보정한 결정계수입니다. 불필요한 변수가 추가될 경우 R-squared가 증가하는 것을 방지해줍니다. 다중 회귀 모델에서 R-squared 대신 이 지표를 보는 것이 더 적합합니다.
- **F-statistic / Prob (F-statistic)**: 회귀 모델 전체의 유의성을 나타냅니다. `Prob (F-statistic)` (p-value)가 0.05보다 작으면 모델이 통계적으로 유의미하다고 할 수 있습니다.
- **coef (계수)**: 각 독립 변수의 회귀 계수(β) 값입니다.
  - `const`: 절편 (β_0)
  - `X1`, `X2`: 각 독립 변수의 기울기 (β_1, β_2). 다른 변수가 일정할 때 해당 변수가 1단위 증가할 때 종속 변수의 변화량을 의미합니다.
- **P>|t| (p-value)**: 각 회귀 계수가 통계적으로 유의미한지에 대한 p-value입니다. 0.05보다 작으면 해당 변수가 종속 변수에 유의미한 영향을 미친다고 해석할 수 있습니다.
- **[0.025 0.975]**: 회귀 계수의 95% 신뢰 구간입니다.
- **Durbin-Watson**: 잔차의 자기상관을 검정하는 통계량입니다. (1.5 ~ 2.5 사이면 독립으로 판단)

---

2. `sklearn.linear_model.LinearRegression`

- **용도**: 예측 모델을 구축하고 성능을 평가하는 데 중점을 둡니다. 머신러닝 파이프라인에 쉽게 통합할 수 있습니다.
- **주의사항**: `statsmodels`처럼 상세한 통계 리포트를 제공하지 않습니다. 계수 및 절편 값은 확인할 수 있지만, 각 계수의 유의성(p-value) 등은 직접 계산해야 합니다.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
# 예제 데이터 (위와 동일)
np.random.seed(42)
X1 = np.random.rand(100) * 10
X2 = np.random.rand(100) * 5
y = 3 + 2*X1 + 5*X2 + np.random.randn(100) * 2
data = pd.DataFrame({'X1': X1, 'X2': X2, 'y': y})

X = data[['X1', 'X2']]
y = data['y']

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

# 2. LinearRegression 모델 생성 및 학습
model = LinearRegression()
model.fit(X_train, y_train)

# 3. 모델 계수 및 절편 확인
print(f"절편 (Intercept): {model.intercept_:.4f}") # 2.5865
print(f"회귀 계수 (Coefficients): {model.coef_}")  # [1.95000442 5.3148695 ]

# 4. 테스트 데이터로 예측 수행
y_pred = model.predict(X_test)

# 5. 모델 성능 평가
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"\n테스트 데이터 성능:")
print(f"MSE (Mean Squared Error): {mse:.4f}") # 2.6657
print(f"R-squared: {r2:.4f}")                 # 0.9749
```

결과 해석 방법

- `model.intercept_`: 절편 (β_0) 값
- `model.coef_`: 각 독립 변수에 대한 회귀 계수 (β_1, β_2, ...) 배열
- **성능 평가 지표**:
  - **MSE (Mean Squared Error)**: 예측 오차의 제곱 평균. 0에 가까울수록 좋습니다.
  - **R-squared**: 결정계수. `statsmodels`의 R-squared와 동일한 의미입니다.

장단점 및 대안

장점

- **해석 용이성**: 모델이 단순하고 직관적이어서 결과를 이해하고 설명하기 쉽습니다. 각 변수가 결과에 미치는 영향을 명확히 파악할 수 있습니다.
- **빠른 속도**: 모델의 학습 및 예측 속도가 매우 빠릅니다.
- **통계적 기반**: 통계적 가정이 뒷받침되므로, 모델의 신뢰도와 유의성을 검증할 수 있습니다.

단점

- **선형성 가정**: 독립 변수와 종속 변수 간의 관계가 선형이 아닐 경우, 모델의 예측 성능이 크게 저하됩니다.
- **이상치에 민감**: 이상치(outlier)가 하나만 있어도 회귀선이 크게 왜곡될 수 있습니다.
- **다중공선성 문제**: 독립 변수들 간에 강한 상관 관계가 있으면 계수 추정이 불안정해지고 해석이 어려워집니다.

대안

- **다항 회귀 (Polynomial Regression)**: 변수 간 비선형 관계를 모델링하기 위해 독립 변수의 고차항을 추가합니다.
- **규제(Regularization) 회귀**: 다중공선성 문제를 완화하고 모델을 일반화하기 위해 사용됩니다.
  - **Ridge Regression**: 계수의 제곱합에 페널티를 부과하여 계수 크기를 줄입니다.
  - **Lasso Regression**: 계수의 절대값 합에 페널티를 부과하며, 특정 계수를 0으로 만들어 변수 선택 효과를 가집니다.
  - **ElasticNet**: Ridge와 Lasso의 페널티를 모두 사용합니다.
- **강건 회귀 (Robust Regression)**: 이상치의 영향을 줄여 안정적인 회귀 모델을 만듭니다.
- **트리 기반 모델 (Decision Tree, RandomForest 등)**: 비선형 관계, 변수 간 상호작용을 잘 잡아내며, 이상치에 상대적으로 덜 민감합니다. (단, 해석이 선형 회귀보다 복잡할 수 있습니다.)