Diabetes Dataset

<모델 개요>

- "Diavetes Dataset"은 442명의 당뇨병 환자에 대한 진단 측정치로 구성되어 있으며, regression models를 위한 dataset이다.

- 9개의 features를 통해 당뇨병의 진행 정도를 예측하는 모델을 만들자.

<Dataset 특징>

샘플 수: 442 / feature 수: 10 / target: 1년 후 질병(당뇨병) 진행의 정량적 측정치

<Features>

1. age:나이

2. sex:성별

3. bmi:체질량 지수(Body Mass Index)

4. bp:평균 혈압(Average Blood Pressure)

5. s1:혈청(Serum) 측정치 1

6. s2:혈청 측정치 2

7. s3:혈청 측정치 3

8. s4:혈청 측정치 4

9. s5:혈청 측정치 5

10. s6:혈청 측정치 6

<데이터 형태>

모든 feature는 standardization(표준화)되어 있음(mean=0, std=1)

<scikit-learn으로 데이터 로드>

```
[1]
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# seaborn 패키지 = matplotlib에 기반을 둠. 그래프 스타일을 간편하게 커스터마이즈
# 데이터의 관계, 분포, 범주형 표현에서 강력함.
# matplotlib에서 구현하기 복잡한 시각화 작업을 더 쉽게 작성 가능함.

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
```

```
[2]
# 데이터 로드
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target
columns = diabetes.feature_names
data = pd.DataFrame(X, columns=columns)
data['target'] = y
```

당뇨병 진행 정도를 예측하는 회귀 모델을 만드는 절차

1. 데이터 분석 및 탐색(EDA)

- 데이터의 분포, 상관관계 등을 살펴보며 각 feature가 target에 어떤 영향을 미치는지 파악.
- pandas, matplotlib, seaborn을 활용해 EDA(Exploratory Data Analysis)를 진행하고, 주요한 feature와 target 간의 관계를 시각하여 data set에 대한 이해 마련.

<주어진 dataset의 결측치 여부, 데이터 타입, 기초 통계 요약 확인>

```
[3]
# 데이터 기본 정보 확인 - 결측치 여부, 데
# 데이터 구조 확인
print(data.info())

# 통계 요약
print(data.describe())
```

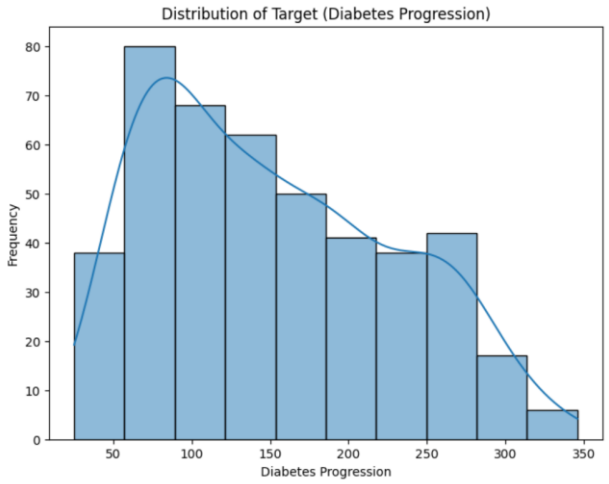
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
None
```

	age	sex	bmi	bp	s1
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02
mean	-2.511817e-19	1.230790e-17	-2.245564e-16	-4.797570e-17	-1.381499e-17
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02
min	-1.072256e-01	-4.464164e-02	-9.027530e-02	-1.123988e-01	-1.267807e-01
25%	-3.729927e-02	-4.464164e-02	-3.422907e-02	-3.665608e-02	-3.424784e-02
50%	5.383060e-03	-4.464164e-02	-7.283766e-03	-5.670422e-03	-4.320866e-03
75%	3.807591e-02	5.068012e-02	3.124802e-02	3.564379e-02	2.835801e-02
max	1.107267e-01	5.068012e-02	1.705552e-01	1.320436e-01	1.539137e-01

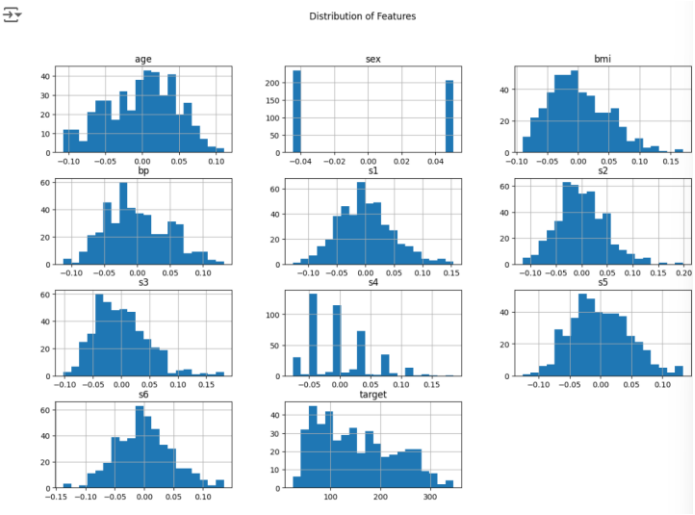
	s2	s3	s4	s5	s6
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02
mean	3.918434e-17	-5.777179e-18	-9.042540e-18	9.293722e-17	1.130318e-17
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02
min	-1.156131e-01	-1.023071e-01	-7.639450e-02	-1.260971e-01	-1.377672e-01
25%	-3.035840e-02	-3.511716e-02	-3.949338e-02	-3.324559e-02	-3.317903e-02
50%	-3.819065e-03	-6.584468e-03	-2.592262e-03	-1.947171e-03	-1.077698e-03
75%	2.984439e-02	2.931150e-02	3.430886e-02	3.243232e-02	2.791705e-02
max	1.987880e-01	1.811791e-01	1.852344e-01	1.335973e-01	1.356118e-01

	target
count	442.000000
mean	152.133484
std	77.093005
min	25.000000
25%	87.000000
50%	140.500000
75%	211.500000
max	346.000000

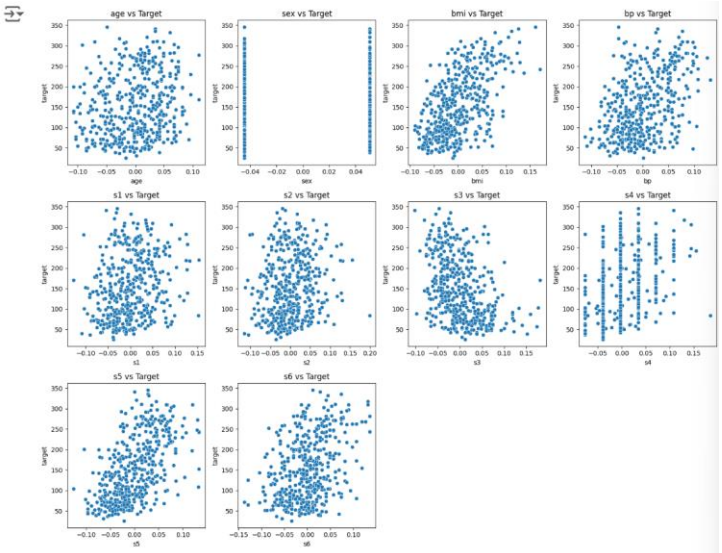
<seaborn 패키지로 구현한 시각화를 통해 target 분포 확인>



<각 feature의 분포를 히스토그램으로 표현 및 파악>



<target과 feature 관계 파악을 위한 scatter plot 시각화>



2. 데이터 분할

- 전체 데이터를 학습용(Train)과 검증용(Test)으로 나눔. 보통 80:20 또는 70:30 비율로 나누며, train_test_split을 사용.
- 데이터셋을 분할하여 모델 성능을 검증할 때 편향되지 않도록 함. => **랜덤 분할**과 **층화 분할**(Stratified Sampling) 두 가지 접근법을 주로 사용

---

1. 랜덤 분할 (Random Split)

- train_test_split 함수를 사용해 데이터를 무작위로 분할.
- 데이터를 학습/테스트 세트로 분할할 때 고르게 분배 -> 특정 패턴이 한쪽에 쏠리는 것을 방지.

여기서 random_state는 결과 재현성을 위해 설정, 동일한 숫자를 설정하면 이후에도 같은 방식으로 데이터를 분할. 랜덤 분할은 전체 데이터에 균일하게 분포된 경우에 적합. 하지만 **데이터가 비대칭적으로 분포된 경우에는 층화 분할을 고려**해야 함.

---

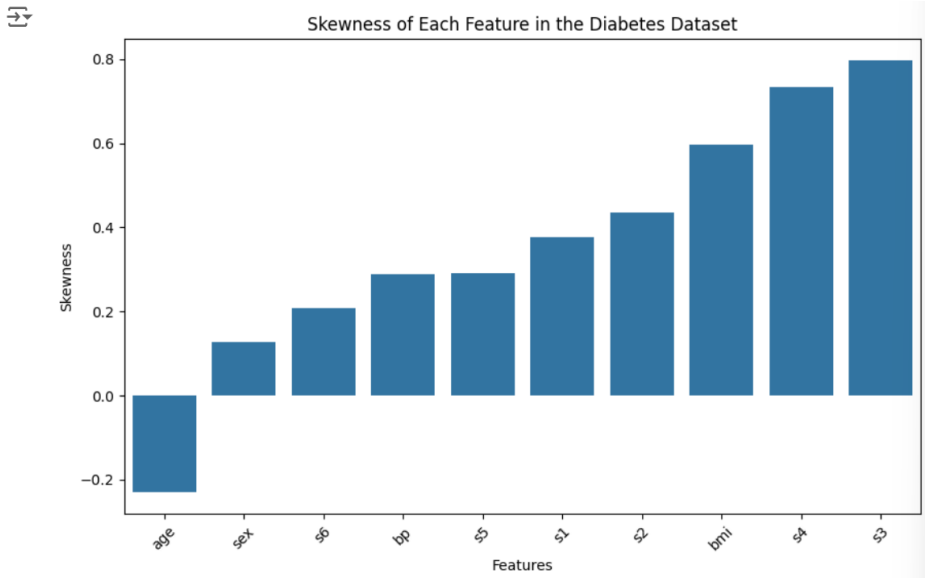
2. 계층화 분할 (Stratified Sampling)

- **데이터의 비율 유지한 채** 데이터를 분할. **일반적으로 classfication 문제에서 사용**, 회귀 문제에서도 데이터의 범주를 나누어 층화할 수 있음.
- 예를 들어, 타겟 변수가 몇몇 구간에 집중된 경우, 구간을 나누고 비율을 유지하면서 데이터를 분할 -> 학습/테스트 세트에 동일한 패턴을 포함시킬 수 있음.
- train_test_split 함수에서 "stratify 매개변수"를 사용하여 층화 분할

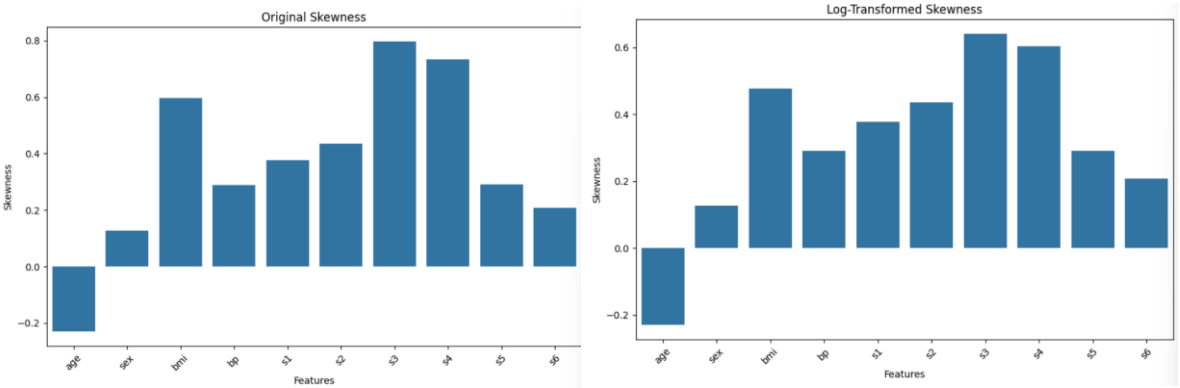
주어진 Diabetes Dataset의 분포를 확인해보았을 때, right tail skewness(오른쪽으로 치우쳐진 비대칭적) 분포를 가진다는 것을 확인. 이때 right tail skewness란 data 값이 평균값보다 큰 경우가 많은 상황. 따라서 각 feature는 비대칭적 분포를 가진다고 판단함.

이에 따라 위 논리대로라면 Diabetes Dataset은 층화분할을 고려할수도 있으나, Diabetes Dataset은 classification이 아닌 regression 형태의 데이터 셋을 가지기 때문에 계층화 분

할이 적절한 분할방법이 아니라고 판단(target이 연속적이기 때문에 계층 분할보다 랜덤 분할이 더 적절하다고 판단함). 따라서 주어진 비대칭적 분포 데이터 셋을 최대한 대칭적 분포로 변환하여 랜덤 분할하는 방법으로 학습을 진행함.



=> Right tail Skewness를 가지는 Diabetes Dataset



=> Skewness가 0.5 이상인 feature를 찾아 로그변환(log1p) 방식으로 대칭적 분포 유도

```python
# 데이터 분할하기
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes

# Diabetes Dataset 로드
diabetes_data = load_diabetes()
X = diabetes_data.data
y = diabetes_data.target

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

# 분할 결과 출력
print("Train set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
Train set size: (353, 10)
Test set size: (89, 10)
```

=> 최종 분할

3. 모델 선택 및 학습

[1]회귀 모델 중에서 **선형 회귀(Linear Regression)**, **라쏘 회귀(Lasso Regression)**, **릿지 회귀(Ridge Regression)** 중 적절한 모델을 탐색함.

1. 선형 회귀 (Linear Regression)

- **적합한 경우**: 데이터가 **다중공선성** 문제가 적고, 모든 feature가 예측에 있어 유의미한 영향을 줄 때.
- **장점**: 해석이 쉬워서 각 feature가 결과에 미치는 영향을 명확하게 파악 가능.
- **단점**: feature 간 강한 상관관계(다중공선성)가 있거나, 데이터에 노이즈가 많을 경우 성능 낮음. 과적합이 발생할 가능성 존재.

=> 데이터가 작고 feature의 상관관계가 낮다면 기본 선형 회귀를 시도.

2. 릿지 회귀 (Ridge Regression-L2 norm)

- **적합한 경우**: **다중공선성 문제**가 존재할 때. 선형 회귀와 달리 **L2 정규화**(제곱 페널티)를 통해 특정 feature의 가중치를 줄이므로, 모델이 전체 feature를 사용하면서도 과적합을 줄이는 효과가 있음.
- **장점**: 과적합 방지에 효과적, feature 간 상관관계가 있어도 안정적인 예측 가능.
- **단점**: 모든 feature를 사용, 예측에 불필요한 feature도 영향을 줌.

=> 다중공선성 문제를 줄이면서도 feature의 정보를 모두 활용하고자 할 때 추천.

Diabetes Dataset의 경우 feature들이 표준화되어 있고

다중공선성이 있을 가능성이 있으므로 적합한 선택일 수 있음.

3. 라쏘 회귀 (Lasso Regression-L1 norm)

- **적합한 경우**: 예측에 **영향을 미치지 않는 feature**들을 제거하고, 중요한 feature만 선택하고자 할 때. **L1 정규화**(절대값 페널티)를 통해 특정 feature의 계수를 0으로 만들어 자동으로 feature 선택을 수행.
- **장점**: 불필요한 feature를 제거해 모델을 단순화. 과적합 방지와 feature 선택을 동시에!
- **단점**: 중요한 feature의 가중치도 과하게 축소할 수 있어 성능이 떨어질 수 있음.

=> feature가 많아 차원 축소가 필요하거나, 가장 중요한 feature에 집중하려 할 때.

Diabetes Dataset의 경우 feature 수가 많지 않음,

성능 향상을 위해 시도해볼 수는 있겠다 정도.

.

[2]다중공선성(Multicollinearity) 문제 고려

: **feature들 간의 상관관계가 매우 높아** 독립 변수들이 서로 종속적인 관계를 가지게 되는 상황. 이는 회귀 모델의 해석을 어렵게 하고, 예측 성능에 부정적인 영향을 줌.

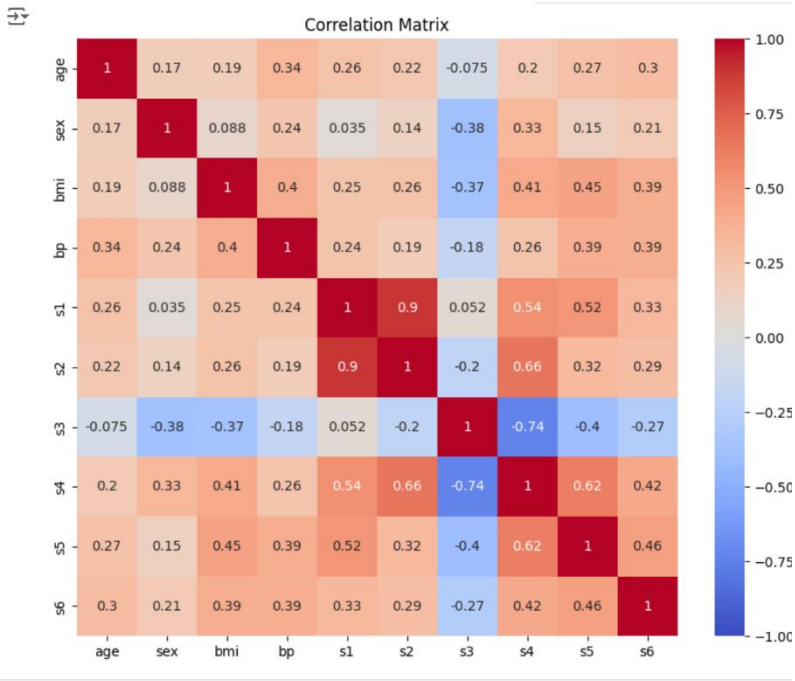
다중공선성 존재 판단 지표

1. 상관행렬 (Correlation Matrix) 확인

- 각 feature 간의 상관 계수를 계산하여 나타낸 행렬. 다중공선성 문제는 feature들 간의 상관 계수가 매우 클 때 나타나기 때문.
- 상관 계수가 0.8 이상이거나 -0.8 이하인 feature 쌍이 있다면, 그 두 feature는 강한 상관관계를 가지고 있음을 의미 -> 다중공선성의 잠재적 원인.

2. 분산 팽창 계수 (Variance Inflation Factor, VIF)

- VIF: 각 feature가 다른 feature들과 얼마나 상관되어 있는지를 수치로 나타낸 것.
- VIF 값이 높을수록 다중공선성이 높다. 일반적으로 특정 feature의 VIF가 10을 초과 -> 해당 feature는 다른 feature들과 강한 상관관계가 있으며 다중공선성 문제가 존재할 가능성이 높음.



=>상관행렬

: feature s1과 s2는 0.9의 높은 상관계수를 가짐 -> 다중공선성 문제가 존재할 가능성이 큼.

	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

=>분산 팽창 계수

: feature s1, s2, s3, s5는 다중공선성 문제를 가지고 있을 확률이 높음.

=>다중공선성 문제를 가지고 있을 확률이 높은 Diabete dateset

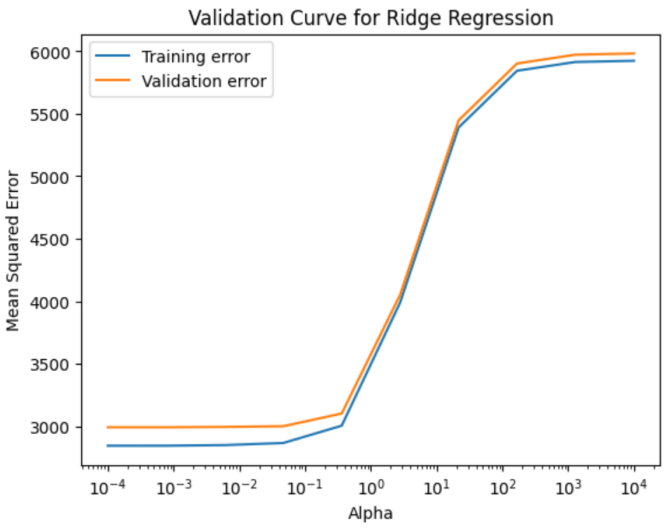
사용이 적합하다고 판단한 모델

- **릿지 회귀**. 주어진 Diabetes Dataset은 다중공선성 문제가 있을 수 있어 모든 feature의 영향을 유지하면서 과적합을 줄이는 릿지 회귀가 적합한 선택.
- 선형 회귀와 달리 릿지 회귀는 L2 정규화(제곱 페널티-L2 norm)를 통해 특정 feature 의 가중치를 줄이므로, 모델이 전체 feature 를 사용하면서도 과적합을 줄이는 효과가 있기 때문임.

4. 하이퍼파라미터 튜닝

1) Validation Curve를 활용해 여러 α값에 따른 학습 및 검증 오류 변화를 시각화

-> 특정 α 범위에서 검증 오류(Train, validation)가 최소화되는 부분 관찰가능



Train, Validation error(검증 오류) 모두 α값이 작을수록 낮음. 검증 오류가 낮을수록 over-fit 발생도 낮아짐.

a>=10일 때, 오류 급격히 증가(=underfit). 지나치게 큰 α값은 모델이 데이터를 충분히 학습하지 못하도록 함. 10^-4 ~ 10^-1에서 비교적 안정적인 학습이 가능함.

- **알파(**규제 강도(regularization strength) 파라미터) **값의 역할**
    - (릿지, 랏소 모두) α=0에 가까울수록 linear regression model과 유사한 결과값 도출.
- **릿지 회귀 (Ridge Regression)**
    - α값이 커질수록 모든 가중치에 대한 제약이 강해짐=가중치가 더 작아짐.
    - **가중치에 대한 제약이 강해져** 모델이 단순해지고 over-fit 가능성 줄어듦. 지나치게 큰 알파는 **under-fit**을 유도함.
- **라쏘 회귀 (Lasso Regression)**
    - α값이 커질수록 가중치의 절대값 감소=일부 feature 가중치를 0으로 설정.
    - 많은 특성이 0이 되어 모델이 단순해짐(불필요한 특성 제거). 지나치게 큰 **알파**는 중요한 특성도 함께 제거하여 under-fit을 유도함.

2) GridSearchCV 또는 RandomizedSearchCV로 하이퍼파라미터를 최적화. 릿지 회귀의 경우 규제 강도(regularization strength) 파라미터를 조정.

```python
# GridSearchCV로 최적의 Ridge 모델 알파 값을 greedy하게 탐색
from sklearn.model_selection import GridSearchCV

# Ridge 모델에 대한 알파 값 범위 설정
ridge_alphas = {'alpha': [0.1, 0.3, 0.5, 0.7, 0.9]}
ridge_search = GridSearchCV(
    Ridge(),
    ridge_alphas,
    cv=5,
    scoring=['neg_mean_squared_error', 'r2'],
    refit='neg_mean_squared_error',  # 'neg_mean_squared_error'를 기준으로 최적 모델 선택
    return_train_score=True
)
ridge_search.fit(X, y)

print("Best alpha for Ridge based on MSE:", ridge_search.best_params_)
print("Best Ridge model MSE:", -ridge_search.best_score_)
print("Corresponding Ridge model R^2:", ridge_search.cv_results_['mean_test_r2'][ridge_search.best_index_])
```

```
Best alpha for Ridge based on MSE: {'alpha': 0.1}
Best Ridge model MSE: 3006.7057011496754
Corresponding Ridge model R^2: 0.47988210231953665
```

=> 최적의 알파 값을 찾기 위해 GridSearchCV로 greed하게 모델 학습을 실행하여 가장 낮은 MSE 값을 기준으로 최적의 알파 값 탐색 과정을 거침.

5. 모델 성능 평가

1) 회귀 모델의 성능을 평가하기 위해, **MSE(Mean Squared Error)**, **RMSE(Root Mean Squared Error)**, **R^2 Score** 지표 사용.

- **그러나,** MSE가 낮으면 모델이 예측과 실제 값 간 오차가 적다는 것을 의미하지

만, 반드시 **일반화 성능**이 우수하다는 보장을 갖는 것은 아님

- **과적합 문제**

  MSE를 지나치게 낮추면 훈련 데이터에 대해 과적합 발생 가능함. 과적합 된 모델은 새로운 데이터에 대한 성능이 떨어질 수 있음. 과적합 방지를 위해 교차 검증을 통해 평가하거나 **릿지 회귀**와 **라쏘 회귀**와 같은 정규화 기법을 사용함.

- 따라서 다른 지표도 함께 확인.

  R^2 점수(결정 계수, Coefficient of Determination): 회귀 모델이 **타겟 변수의 변동성을 얼마나 잘 설명하는지**를 나타내는 지표. MSE가 낮아도 모델의 예측성능을 명확히 표현함.

  $R^2 = 1$: 모델이 모든 데이터를 완벽하게 설명하는 경우

  $0 < R^2 < 1$: 모델이 타겟 변수의 변동성 일부를 설명. 1에 가까울수록 예측력이 높은 모델.

  $R^2 = 0$: 모델이 타겟 변수의 변동성을 전혀 설명하지 못하는 경우. 즉, 모델이 예측하는 값이 단순히 평균을 예측하는 것과 동일한 상태.

  $R^2 < 0$: 모델이 평균보다도 설명력이 낮은 경우. 실제 데이터와 예측 데이터 간의 차이가 너무 크거나, 모델이 타겟과의 관계를 잘못 학습한 경우.

- MSE는 데이터의 스케일(단위)에 영향을 크게 받음. 값이 즉, 큰 타겟 변수를 사용할 경우 MSE가 커짐.

  ⇨ **따라서 RMSE**(Root Mean Squared Error)-'스케일에 덜 민감', **MAE**(Mean Absolute Error)-'아웃라이어에 덜 민감'을 고려함.

```
Linear Regression
MSE: 3403.8877929322553
MAE: 47.62636803032458
R2 Score: 0.3908118647948903

Ridge Regression (Best alpha: 0.1)
MSE: 3356.280253757379
MAE: 47.214176271755726
R2 Score: 0.39933210687568577

Lasso Regression (Best alpha: 0.1)
MSE: 3361.8191859700573
MAE: 47.535983985729175
R2 Score: 0.39834081339278204
```

=> 최적 alpha 값에 대한 선형, 릿지, 랏소 모델의 MSE, MAE, R^2 점수 출력하기.

2) Cross-Validation(교차 검증)을 통해 모델의 일반화 성능을 검토

K-Fold 교차 검증과 Stratified K-Fold

- **데이터 편향을 최소화하고 모델의 일반화 성능을 평가하기 위해 K-Fold Cross-Validation을 사용.**
- **데이터를 K개의 폴드(데이터셋)로 나누어 K번 반복해서 모델을 평가하는 방법으로, 데이터의 편향을 줄이면서 성능을 검증.**
- **회귀 문제에서도 Stratified K-Fold를 사용, 이를 통해 각 폴드에 타겟의 분포를 고르게 유지할 수 있음.**

```python
# Ridge Regression 모델을 사용하여 K-Fold 교차 검증
# 데이터를 여러 번(폴드 수만큼) 나누어 각각의 폴드에서 모델을 학습 및 평가
# 최종적으로 각 폴드에서 평가된 점수를 평균 내어 모델의 성능을 확인.

import numpy as np
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.datasets import load_diabetes

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

# K-Fold 설정
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 데이터를 5개의 폴드로 분할
# shuffle=True: 데이터를 무작위로 섞어 분할. 편향을 줄이는 데 도움
# random_state=42: 재현성을 위해 설정. 고정된 시드 -> 같은 결과
# 모델 초기화
model = Ridge(alpha=0.1)

# 각 폴드의 MSE와 R^2 점수를 저장할 리스트
mse_scores = []
mae_scores = []
r2_scores = []
```

```python
# K-Fold 설정
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 데이터를 5개의 폴드로 분할
# shuffle=True: 데이터를 무작위로 섞어 분할. 편향을 줄이는 데 도움
# random_state=42: 재현성을 위해 설정. 고정된 시드 -> 같은 결과
# 모델 초기화
model = Ridge(alpha=0.1)

# 각 폴드의 MSE와 R^2 점수를 저장할 리스트
mse_scores = []
mae_scores = []
r2_scores = []

# K-Fold 교차 검증
for train_index, test_index in kf.split(X):
    # kf.split(X)를 통해 각 폴드의 train dataset, test dataset의 index를 구함
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # 모델 학습
    model.fit(X_train, y_train)

    # 예측 with test dataset
    y_pred = model.predict(X_test)

    # MSE와 R^2 점수 계산
    mse_scores.append(mean_squared_error(y_test, y_pred))
    mae_scores.append(mean_absolute_error(y_test, y_pred))
    r2_scores.append(r2_score(y_test, y_pred))


# 평균 MSE와 R^2 점수 출력
print("Average MSE from K-Fold Cross-Validation:", np.mean(mse_scores))
print("Average MAE from K-Fold Cross-Validation:", np.mean(mae_scores))
print("Average R^2 Score from K-Fold Cross-Validation:", np.mean(r2_scores))
```

```
Average MSE from K-Fold Cross-Validation: 3013.810607770459
Average MAE from K-Fold Cross-Validation: 44.49645464858337
Average R^2 Score from K-Fold Cross-Validation: 0.4790759615259203
```