회귀 (regression) 예측

수치형 값을 예측 (Y의 값이 연속된 수치로 표현)

예시

- 주택 가격 예측
- 매출액 예측

도큐먼트

```
!pip install scikit-learn==1.0.2
```

```
Requirement already satisfied: scikit-learn==1.0.2 in /usr/local/lib/python3.10/dist-packages (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.0.2) (1.23.5)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.0.2) (1.11.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.0.2) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.0.2) (3.2.0)
```

```
import pandas as pd
import numpy as np

np.set_printoptions(suppress=True)
```

```
from sklearn.datasets import load_boston
```

데이터 로드

```
data = load_boston()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np
```

```
        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

```
print(data['DESCR'])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/
```

```
This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Ka
```

data['data']에는 X 데이터, data['feature_names']에는 컬럼 명입니다.

```
df = pd.DataFrame(data['data'], columns=data['feature_names'])
```

Y 데이터인 price도 데이터프레임에 추가 합니다.

```
df['MEDV'] = data['target']
```

```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

컬럼 소개

속성 수 : 13

- **CRIM**: 범죄율
- **ZN**: 25,000 평방 피트 당 주거용 토지의 비율
- **INDUS**: 비소매(non-retail) 비즈니스 면적 비율
- **CHAS**: 찰스 강 더미 변수 (통로가 하천을 향하면 1; 그렇지 않으면 0)

- **NOX**: 산화 질소 농도 (천만 분의 1)
- **RM**:주거 당 평균 객실 수
- **AGE**: 1940 년 이전에 건축된 자가 소유 점유 비율
- **DIS**: 5 개의 보스턴 고용 센터까지의 가중 거리
- **RAD**: 고속도로 접근성 지수
- **TAX**: 10,000 달러 당 전체 가치 재산 세율
- **PTRATIO** 도시 별 학생-교사 비율
- **B**: 1000 (Bk-0.63) ^ 2 여기서 Bk는 도시 별 검정 비율입니다.
- **LSTAT**: 인구의 낮은 지위
- **MEDV**: 자가 주택의 중앙값 (1,000 달러 단위)

train / test 데이터를 분할 합니다.

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(df.drop('MEDV', 1), df['MEDV'])
```

```
<ipython-input-95-5fdc4024a942>:1: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.
  x_train, x_test, y_train, y_test = train_test_split(df.drop('MEDV', 1), df['MEDV'])
```

```
x_train.shape, x_test.shape
```

```
((379, 13), (127, 13))
```

```
x_train.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
73	0.19539	0.0	10.81	0.0	0.413	6.245	6.2	5.2873	4.0	305.0	19.2	377.17	7.54
461	3.69311	0.0	18.10	0.0	0.713	6.376	88.4	2.5671	24.0	666.0	20.2	391.43	14.65
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
110	0.10793	0.0	8.56	0.0	0.520	6.195	54.4	2.7778	5.0	384.0	20.9	393.49	13.00
216	0.04560	0.0	13.89	1.0	0.550	5.888	56.0	3.1121	5.0	276.0	16.4	392.80	13.51

```
y_train.head()
```

```
73     23.4
461    17.7
0      24.0
110    21.7
216    23.3
Name: MEDV, dtype: float64
```

평가 지표 만들기

MSE(Mean Squared Error)

$(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2$

예측값과 실제값의 차이에 대한 **제곱**에 대하여 평균을 낸 값

MAE (Mean Absolute Error)

$(\frac{1}{n})\sum_{i=1}^{n}|y_i - x_i|$

예측값과 실제값의 차이에 대한 **절대값**에 대하여 평균을 낸 값

RMSE (Root Mean Squared Error)

$\sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$

예측값과 실제값의 차이에 대한 **제곱**에 대하여 평균을 낸 뒤 **루트**를 씌운 값

평가 지표 만들어 보기

```
import numpy as np

pred = np.array([3, 4, 5])
actual = np.array([1, 2, 3])

def my_mse(pred, actual):
    return ((pred - actual)**2).mean()

my_mse(pred, actual)
```

```
4.0
```

```python
def my_mae(pred, actual):
    return np.abs(pred - actual).mean()
```

```python
my_mae(pred, actual)
```

```
2.0
```

```python
def my_rmse(pred, actual):
    return np.sqrt(my_mse(pred, actual))
```

```python
my_rmse(pred, actual)
```

```
2.0
```

sklearn의 평가지표 활용하기

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```python
my_mae(pred, actual), mean_absolute_error(pred, actual)
```

```
(2.0, 2.0)
```

```python
my_mse(pred, actual), mean_squared_error(pred, actual)
```

```
(4.0, 4.0)
```

모델별 성능 확인을 위한 함수

```python
import matplotlib.pyplot as plt
import seaborn as sns

my_predictions = {}

colors = ['r', 'c', 'm', 'y', 'k', 'khaki', 'teal', 'orchid', 'sandybrown',
          'greenyellow', 'dodgerblue', 'deepskyblue', 'rosybrown', 'firebrick',
          'deeppink', 'crimson', 'salmon', 'darkred', 'olivedrab', 'olive',
          'forestgreen', 'royalblue', 'indigo', 'navy', 'mediumpurple', 'chocolate',
          'gold', 'darkorange', 'seagreen', 'turquoise', 'steelblue', 'slategray',
          'peru', 'midnightblue', 'slateblue', 'dimgray', 'cadetblue', 'tomato'
         ]

def plot_predictions(name_, pred, actual):
    df = pd.DataFrame({'prediction': pred, 'actual': y_test})
    df = df.sort_values(by='actual').reset_index(drop=True)

    plt.figure(figsize=(12, 9))
    plt.scatter(df.index, df['prediction'], marker='x', color='r')
    plt.scatter(df.index, df['actual'], alpha=0.7, marker='o', color='black')
    plt.title(name_, fontsize=15)
    plt.legend(['prediction', 'actual'], fontsize=12)
    plt.show()

def mse_eval(name_, pred, actual):
    global predictions
    global colors

    plot_predictions(name_, pred, actual)

    mse = mean_squared_error(pred, actual)
    my_predictions[name_] = mse

    y_value = sorted(my_predictions.items(), key=lambda x: x[1], reverse=True)

    df = pd.DataFrame(y_value, columns=['model', 'mse'])
    print(df)
    min_ = df['mse'].min() - 10
    max_ = df['mse'].max() + 10

    length = len(df)
```

```
    plt.figure(figsize=(10, length))
    ax = plt.subplot()
    ax.set_yticks(np.arange(len(df)))
    ax.set_yticklabels(df['model'], fontsize=15)
    bars = ax.barh(np.arange(len(df)), df['mse'])

    for i, v in enumerate(df['mse']):
        idx = np.random.choice(len(colors))
        bars[i].set_color(colors[idx])
        ax.text(v + 2, i, str(round(v, 3)), color='k', fontsize=15, fontweight='bold'

    plt.title('MSE Error', fontsize=18)
    plt.xlim(min_, max_)

    plt.show()

def remove_model(name_):
    global my_predictions
    try:
        del my_predictions[name_]
    except KeyError:
        return False
    return True
```

LinearRegression

[도큐먼트]()

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression(n_jobs=-1)
```

- n_jobs: CPU코어의 사용

```
model.fit(x_train, y_train)
```

```
LinearRegression(n_jobs=-1)
```

```
pred = model.predict(x_test)

mse_eval('LinearRegression', pred, y_test)
```

규제 (Regularization)

학습이 과대적합 되는 것을 방지하고자 일종의 **penalty**를 부여하는 것

L2 규제 (L2 Regularization)

- 각 가중치 제곱의 합에 규제 강도(Regularization Strength) λ를 곱한다.
- λ를 크게 하면 가중치가 더 많이 감소되고(규제를 중요시함), λ를 작게 하면 가중치가 증가한다(규제를 중요시하지 않음).

L1 규제 (L1 Regularization)

- 가중치의 제곱의 합이 아닌 **가중치의 합**을 더한 값에 규제 강도(Regularization Strength) λ를 곱하여 오차에 더한다.
- 어떤 가중치(w)는 실제로 0이 된다. 즉, 모델에서 완전히 제외되는 특성이 생기는 것이다.

L2 규제가 L1 규제에 비해 더 안정적이라 일반적으로는 L2규제가 더 많이 사용된다

릿지(Ridge) - L2 규제

$Error = MSE + \alpha w^2$

라쏘(Lasso) - L1 규제

$Error = MSE + \alpha|w|$

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
```

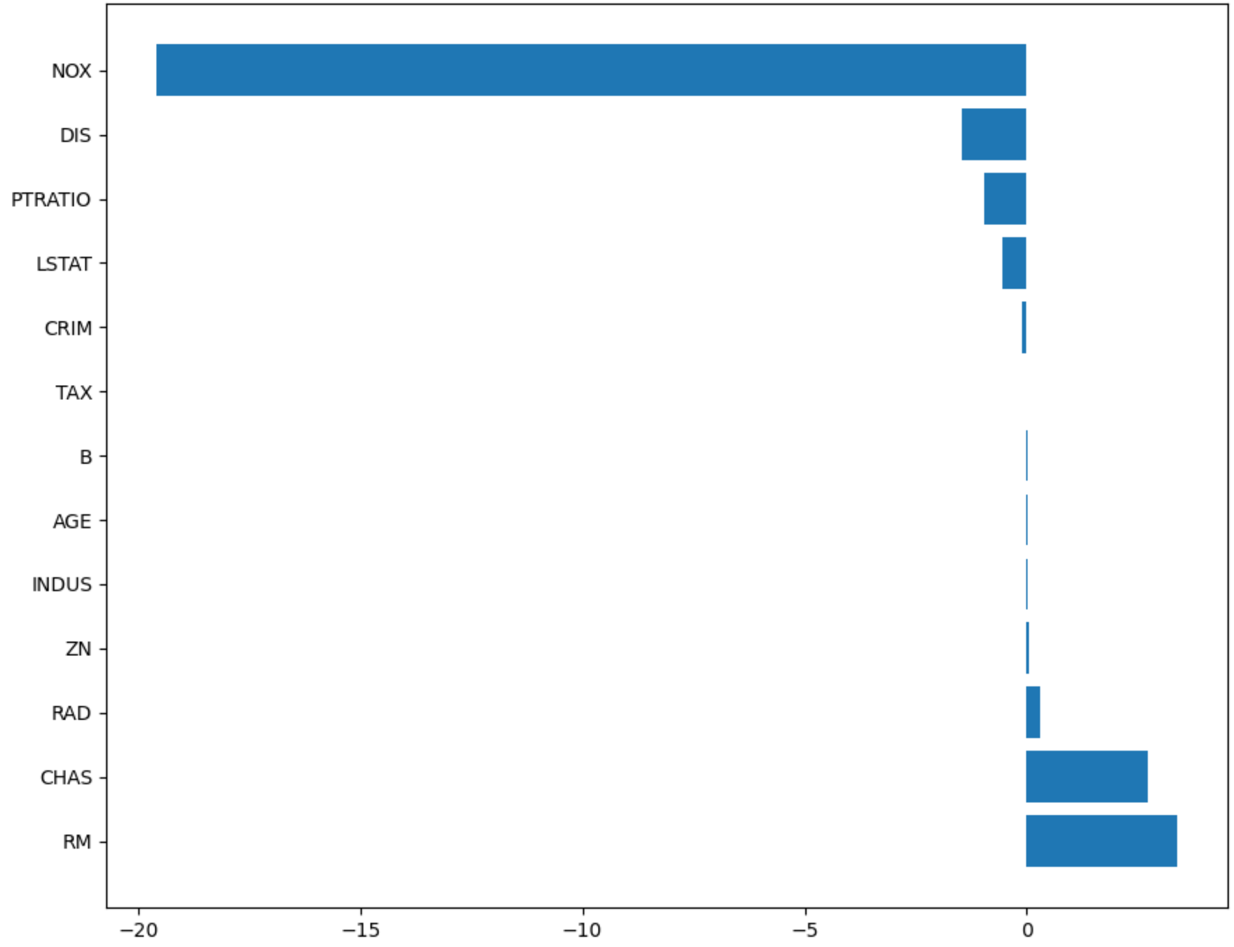
```
# 값이 커질 수록 큰 규제입니다.
alphas = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]
```

```
x_train.columns
```

```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

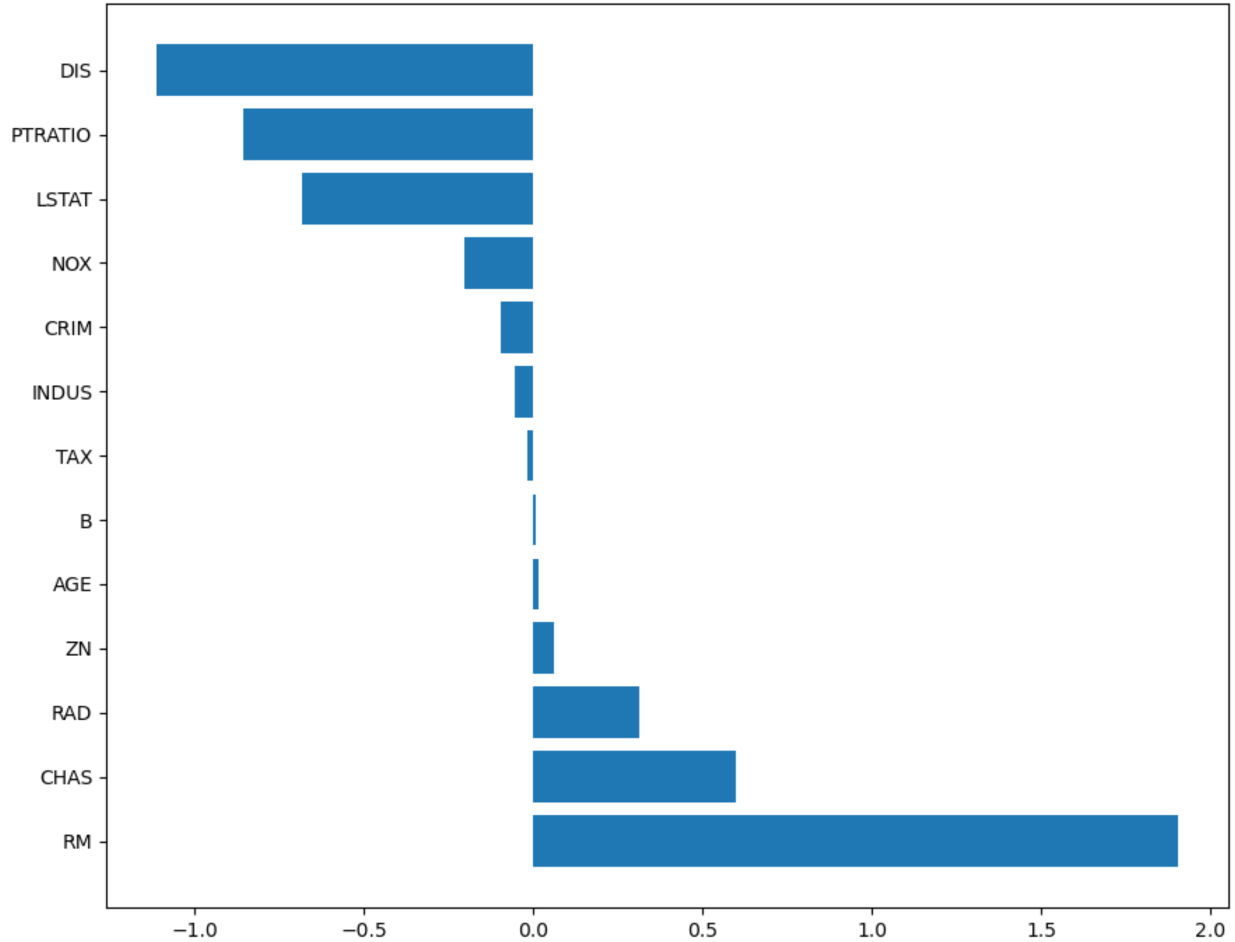
```
def plot_coef(columns, coef):
    coef_df = pd.DataFrame(list(zip(columns, coef)))
    coef_df.columns=['feature', 'coef']
    coef_df = coef_df.sort_values('coef', ascending=False).reset_index(drop=True)

    fig, ax = plt.subplots(figsize=(9, 7))
```

```
    ax.barh(np.arange(len(coef_df)), coef_df['coef'])
    idx = np.arange(len(coef_df))
    ax.set_yticks(idx)
    ax.set_yticklabels(coef_df['feature'])
    fig.tight_layout()
    plt.show()

plot_coef(x_train.columns, ridge.coef_)
```

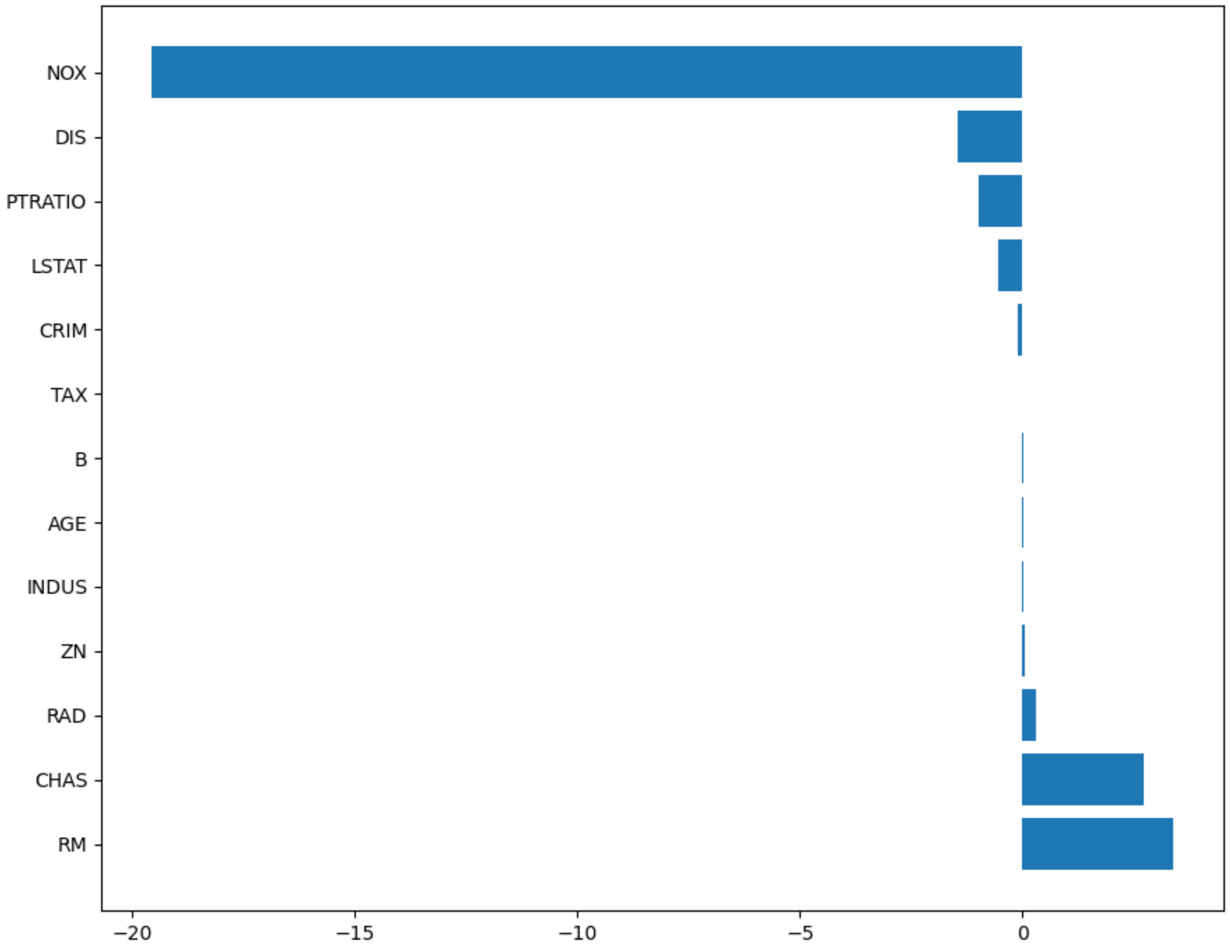


alpha 값에 따른 coef 의 차이를 확인해 봅시다

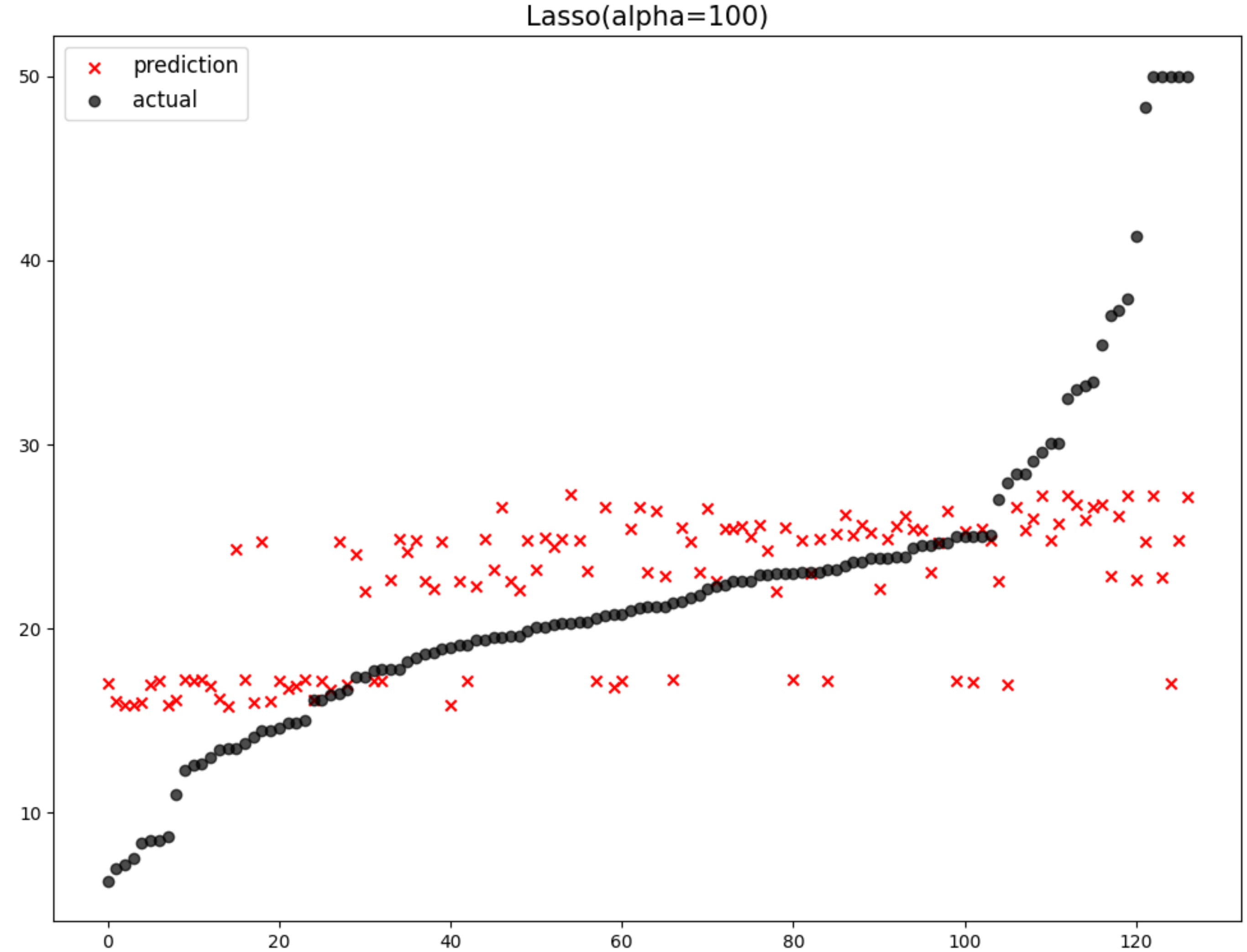
```
ridge_100 = Ridge(alpha=100)
ridge_100.fit(x_train, y_train)
ridge_pred_100 = ridge_100.predict(x_test)

ridge_001 = Ridge(alpha=0.001)
ridge_001.fit(x_train, y_train)
ridge_pred_001 = ridge_001.predict(x_test)

plot_coef(x_train.columns, ridge_100.coef_)
```



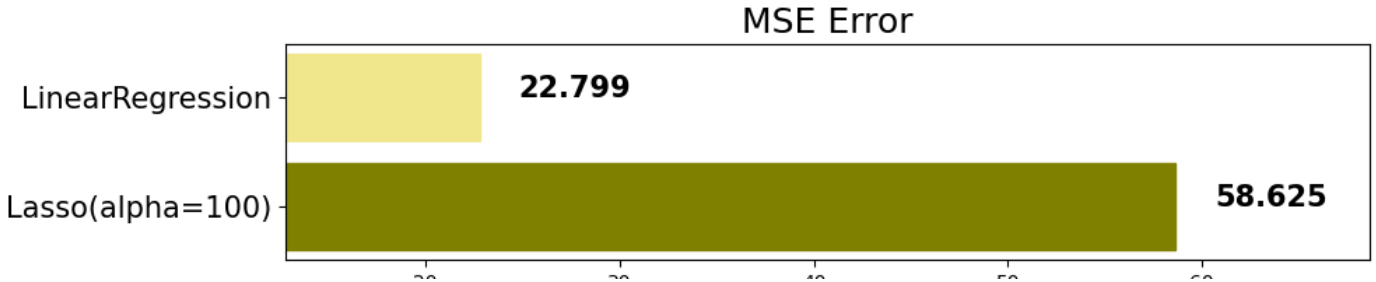

```
plot_coef(x_train.columns, ridge_001.coef_)
```

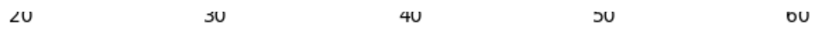


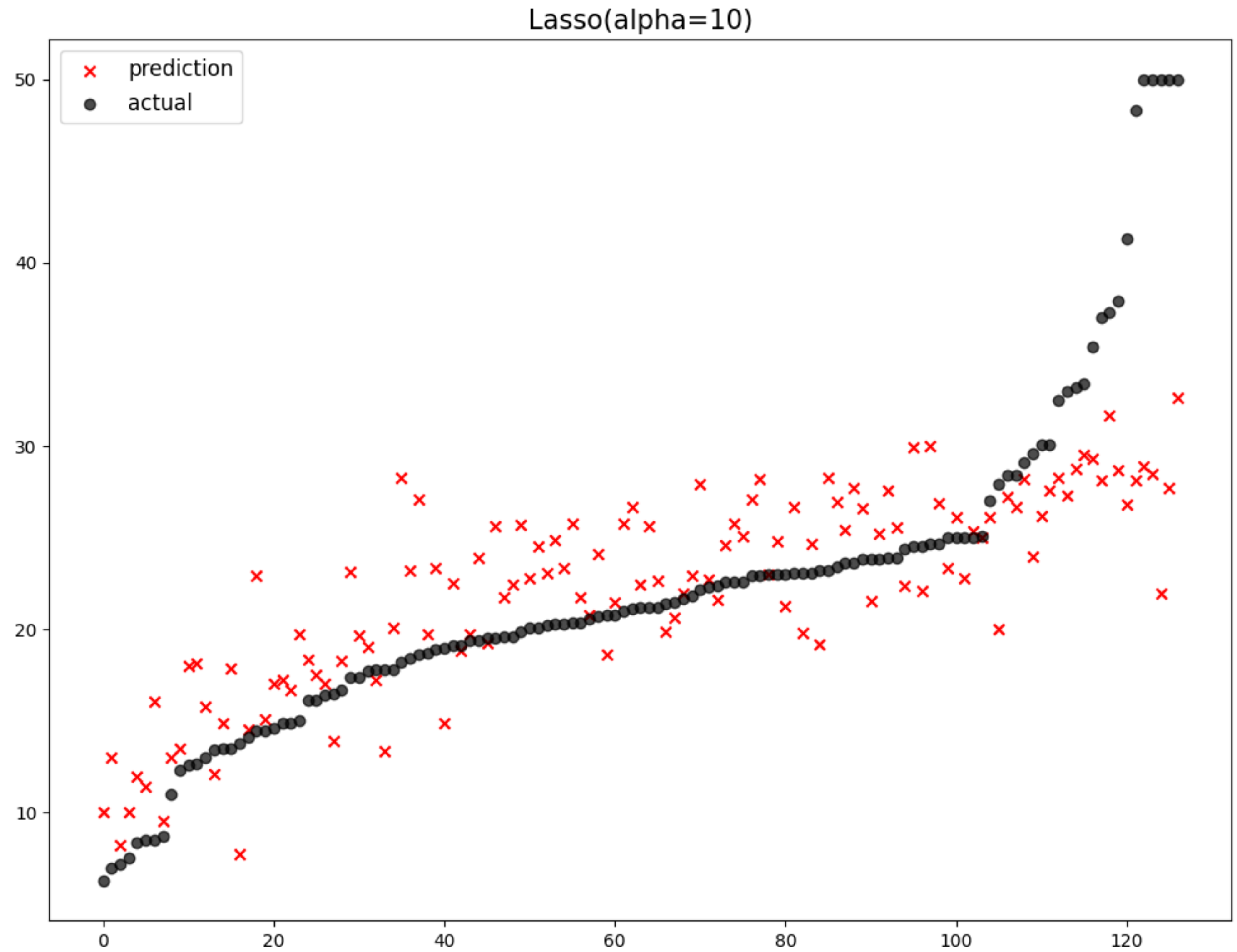

```
from sklearn.linear_model import Lasso

# 값이 커질 수록 큰 규제입니다.
alphas = [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]

for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(x_train, y_train)
    pred = lasso.predict(x_test)
    mse_eval('Lasso(alpha={})'.format(alpha), pred, y_test)
```



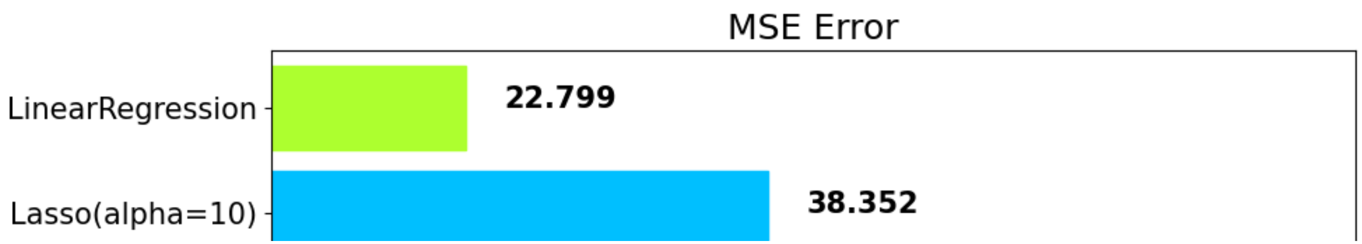
	model	mse
0	Lasso(alpha=100)	58.625379
1	LinearRegression	22.798819

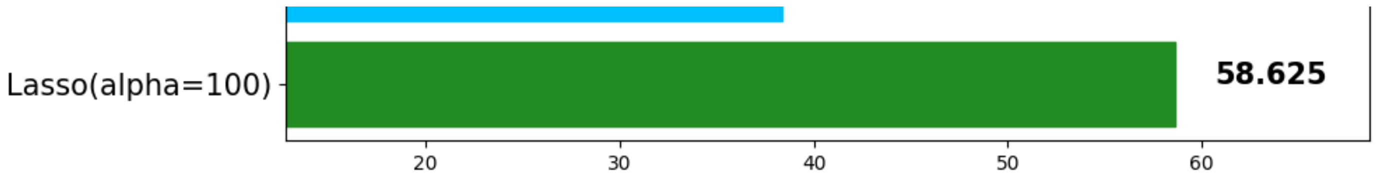


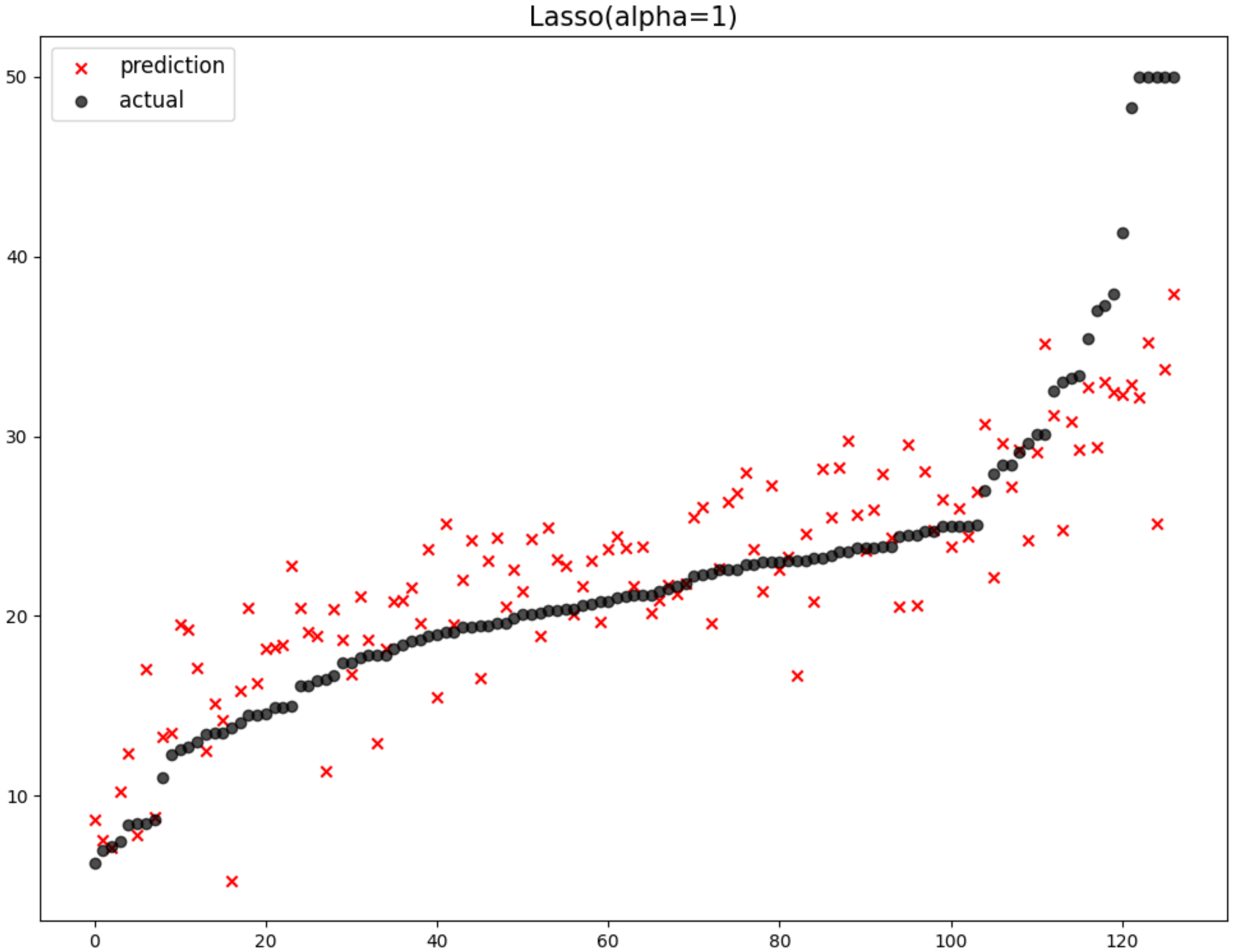




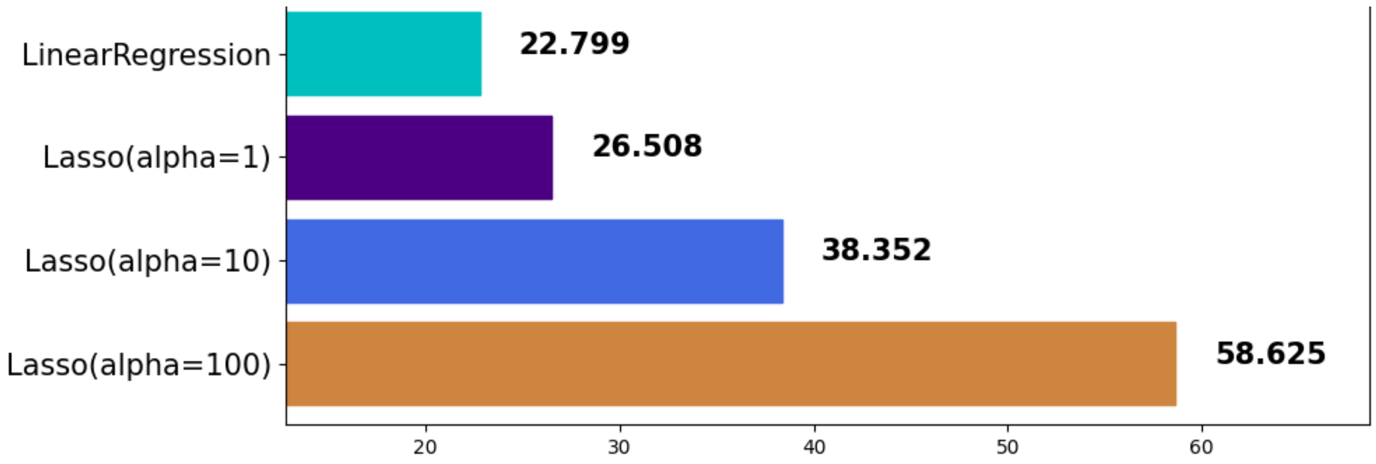
	model	mse
0	Lasso(alpha=100)	58.625379
1	Lasso(alpha=10)	38.352168
2	LinearRegression	22.798819



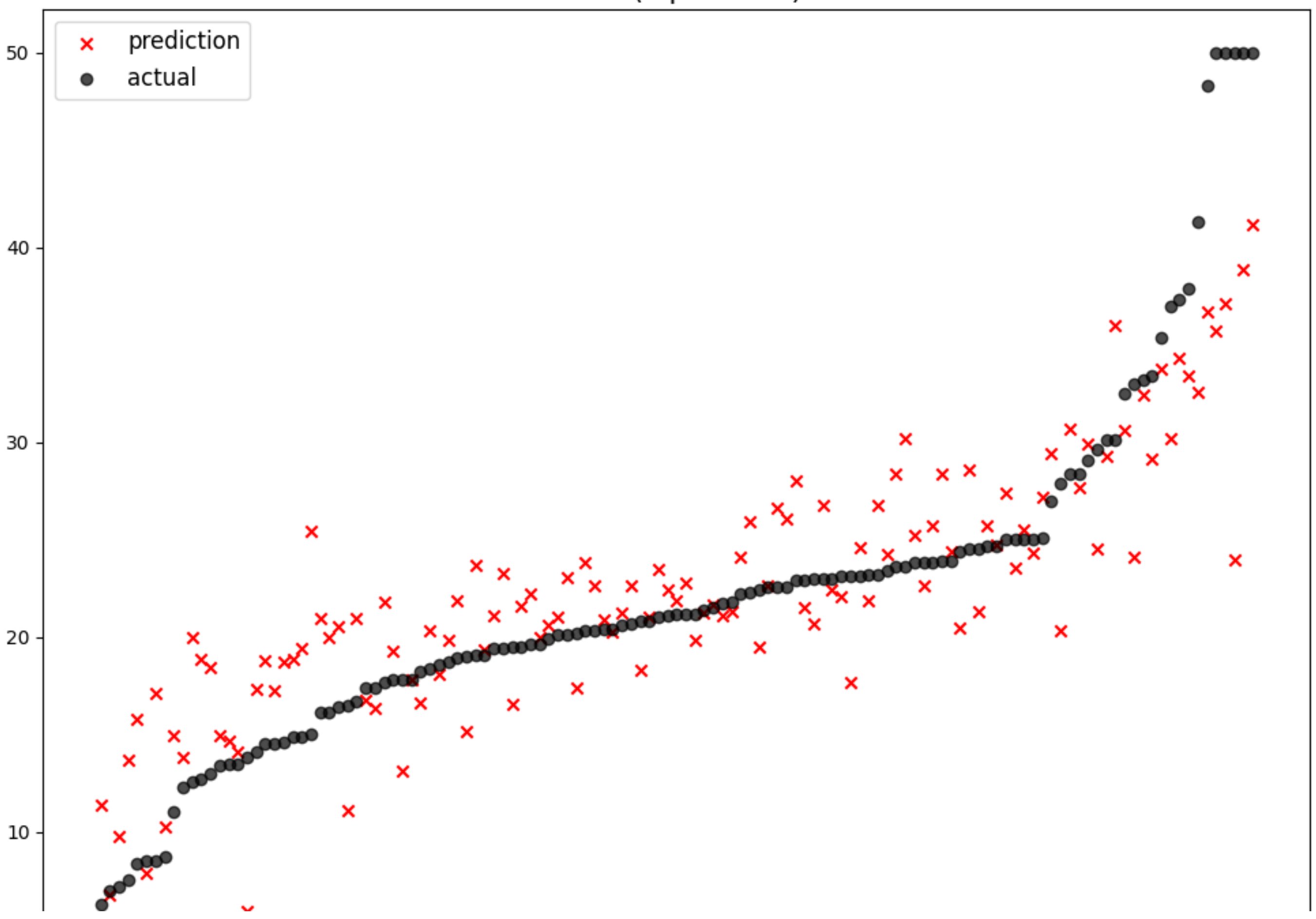


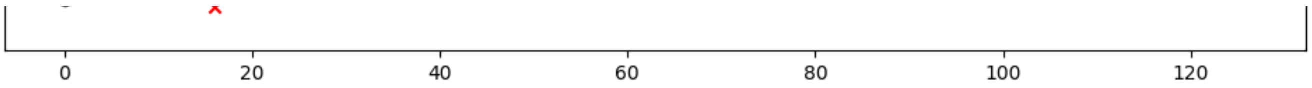


	model	mse
0	Lasso(alpha=100)	58.625379
1	Lasso(alpha=10)	38.352168
2	Lasso(alpha=1)	26.507706
3	LinearRegression	22.798819

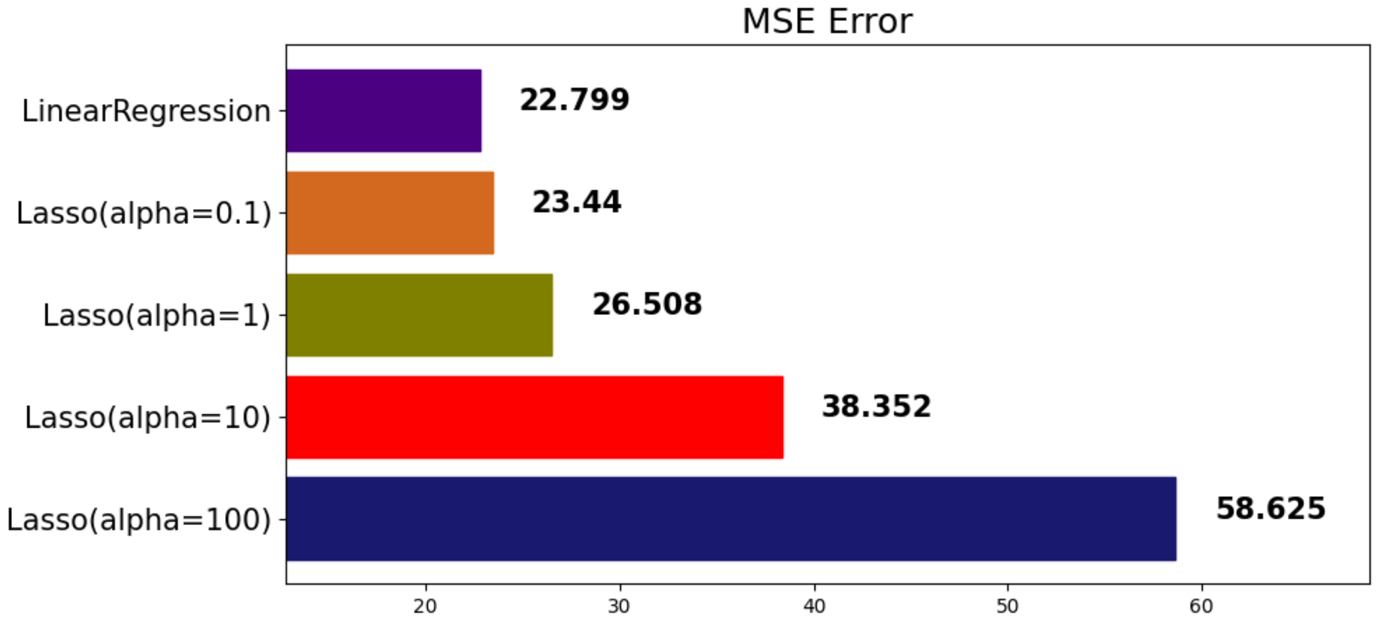


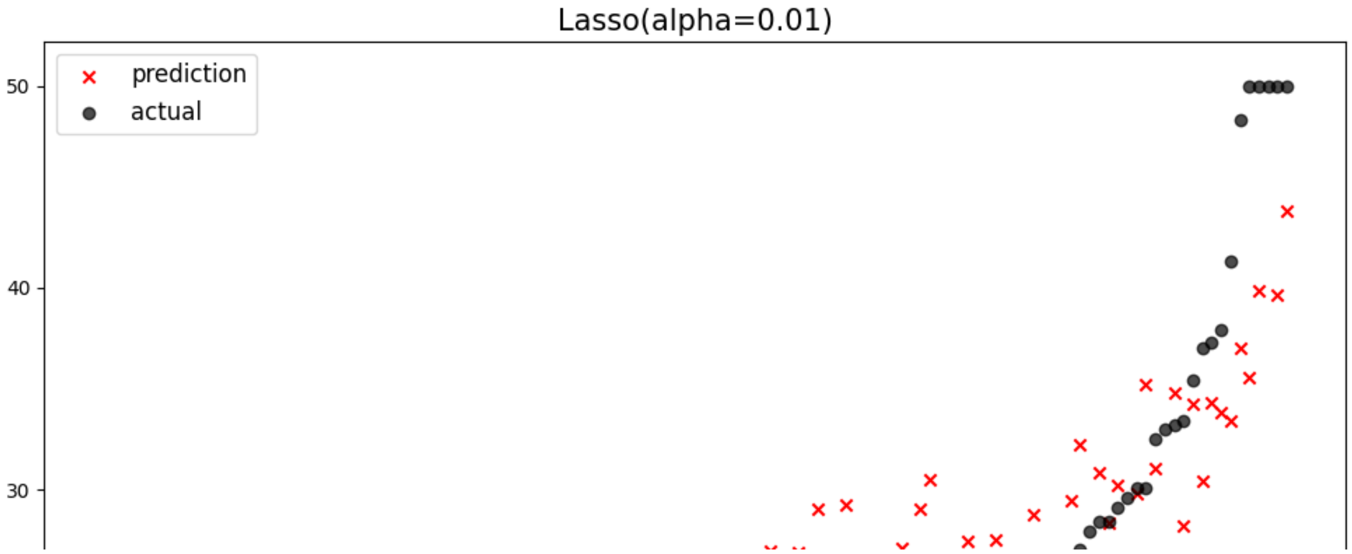
Lasso(alpha=0.1)

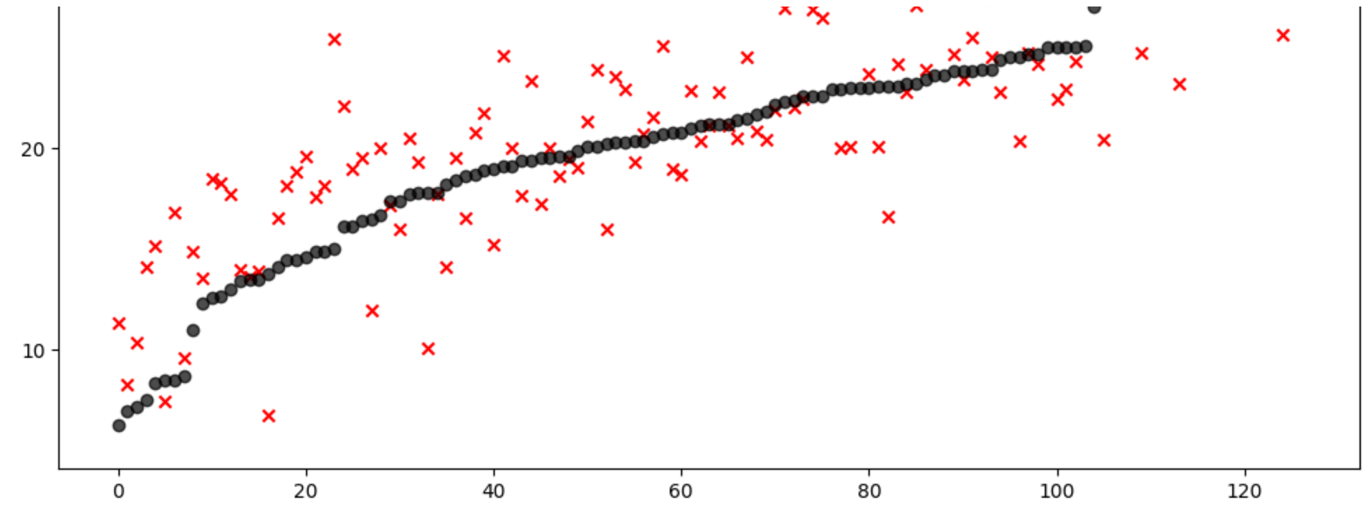




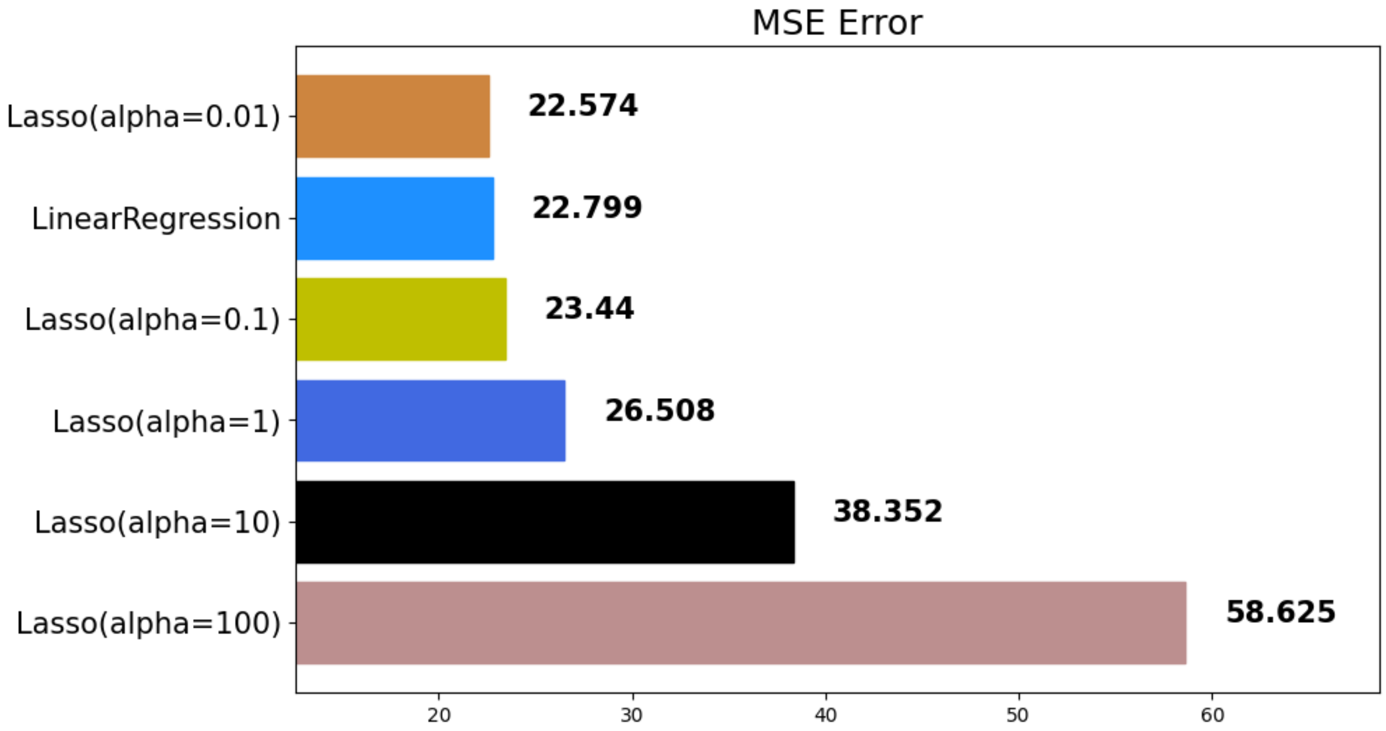
	model	mse
0	Lasso(alpha=100)	58.625379
1	Lasso(alpha=10)	38.352168
2	Lasso(alpha=1)	26.507706
3	Lasso(alpha=0.1)	23.440245
4	LinearRegression	22.798819

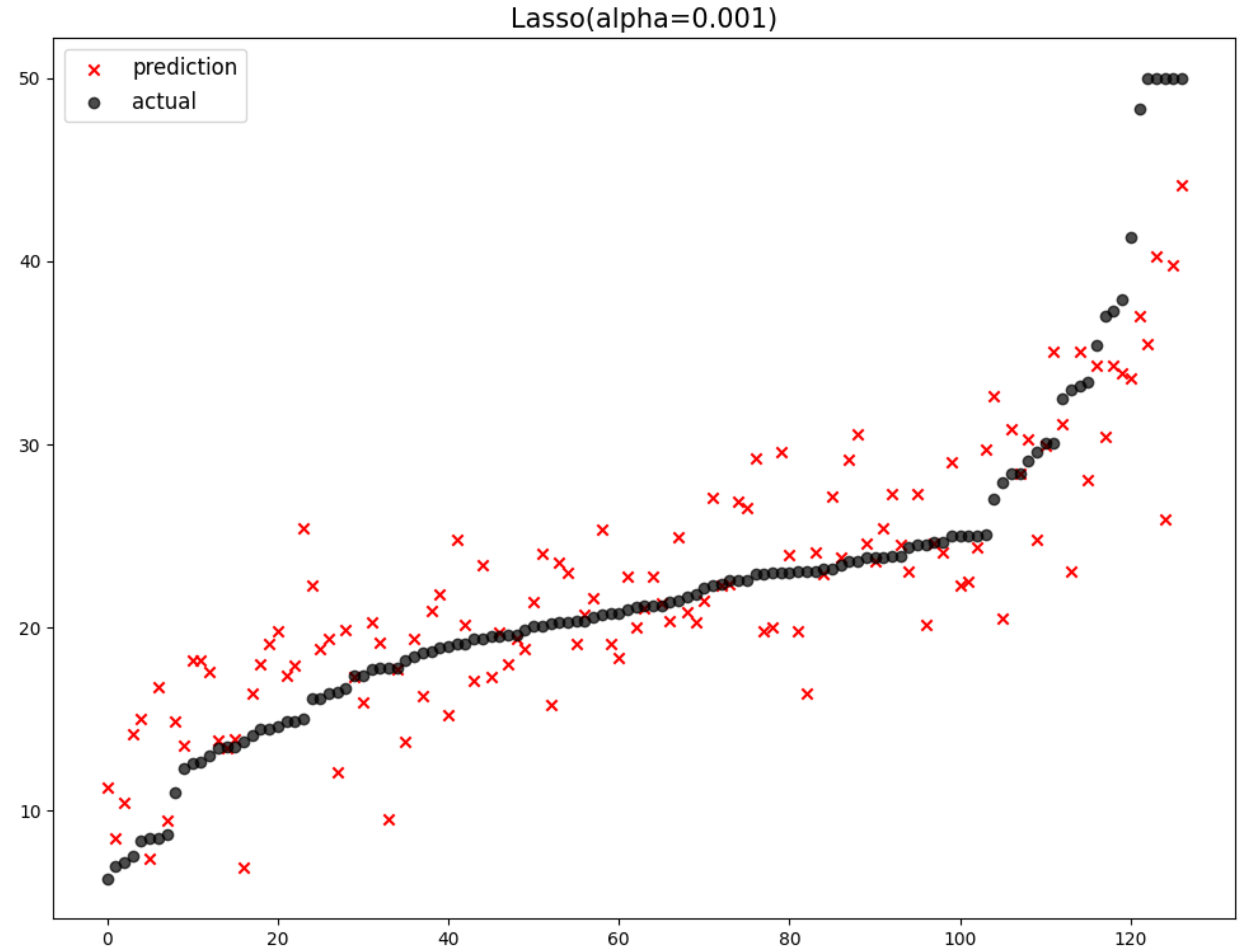




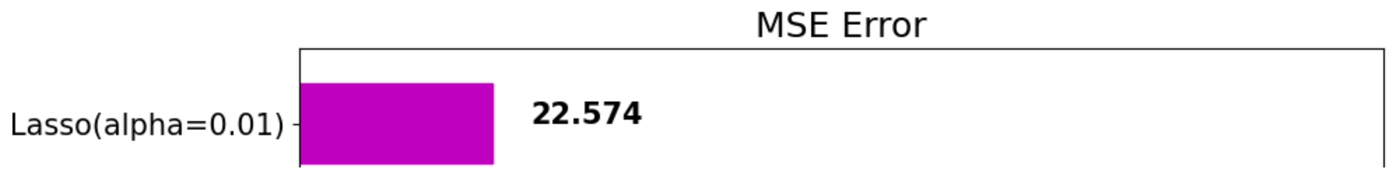


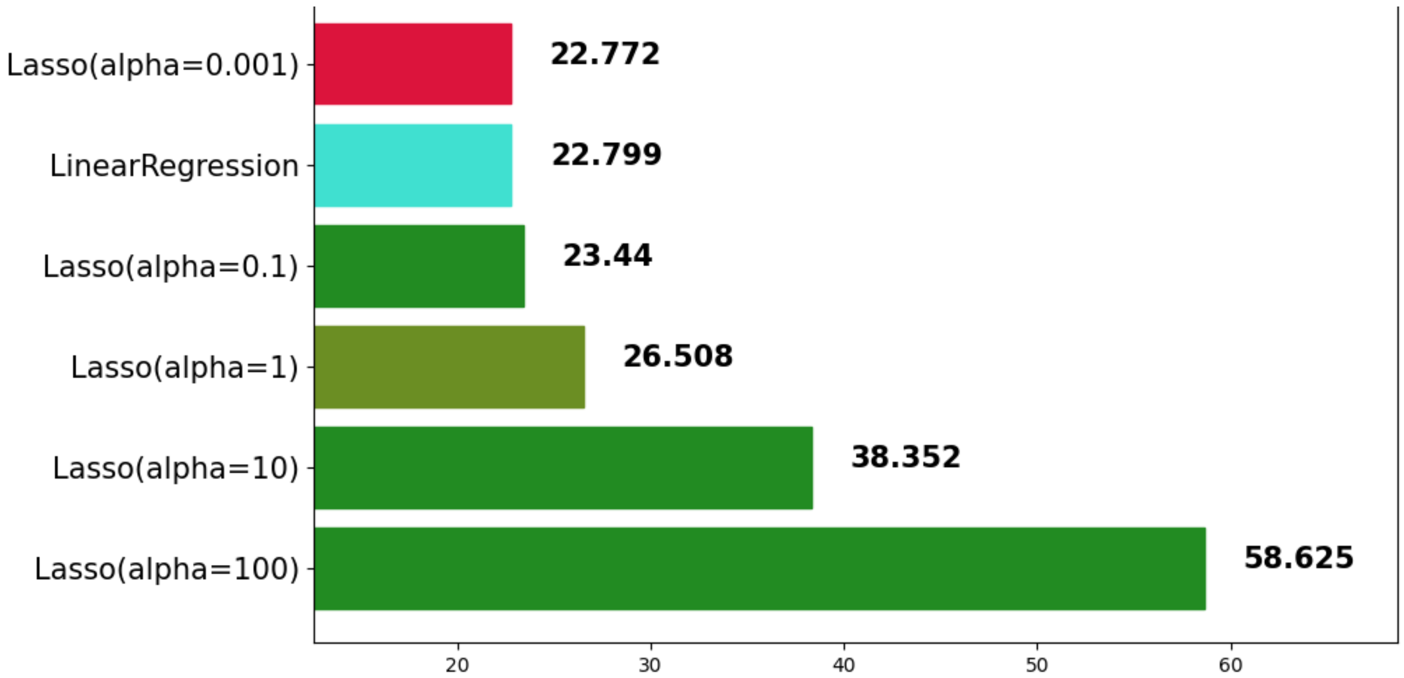
	model	mse
0	Lasso(alpha=100)	58.625379
1	Lasso(alpha=10)	38.352168
2	Lasso(alpha=1)	26.507706
3	Lasso(alpha=0.1)	23.440245
4	LinearRegression	22.798819
5	Lasso(alpha=0.01)	22.573644

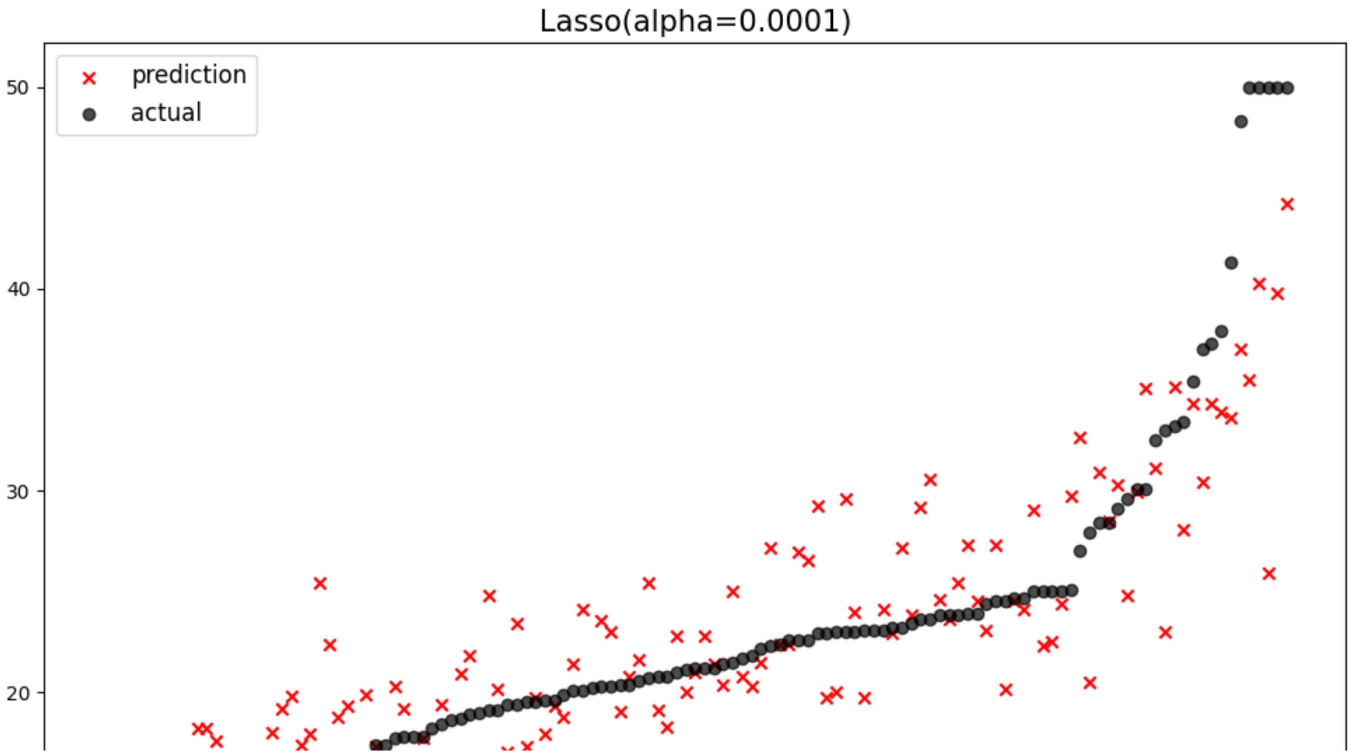




	model	mse
0	Lasso(alpha=100)	58.625379
1	Lasso(alpha=10)	38.352168
2	Lasso(alpha=1)	26.507706
3	Lasso(alpha=0.1)	23.440245
4	LinearRegression	22.798819
5	Lasso(alpha=0.001)	22.771857
6	Lasso(alpha=0.01)	22.573644




Lasso(alpha=0.001)
LinearRegression
Lasso(alpha=0.1)
Lasso(alpha=1)
Lasso(alpha=10)
Lasso(alpha=100)
22.772
22.799
23.44
26.508
38.352
58.625
20
30
40
50
60


Lasso(alpha=0.0001)
prediction
actual
50
40
30
20

```
lasso_100 = Lasso(alpha=100)
lasso_100.fit(x_train, y_train)
lasso_pred_100 = lasso_100.predict(x_test)

lasso_001 = Lasso(alpha=0.001)
lasso_001.fit(x_train, y_train)
lasso_pred_001 = lasso_001.predict(x_test)
```

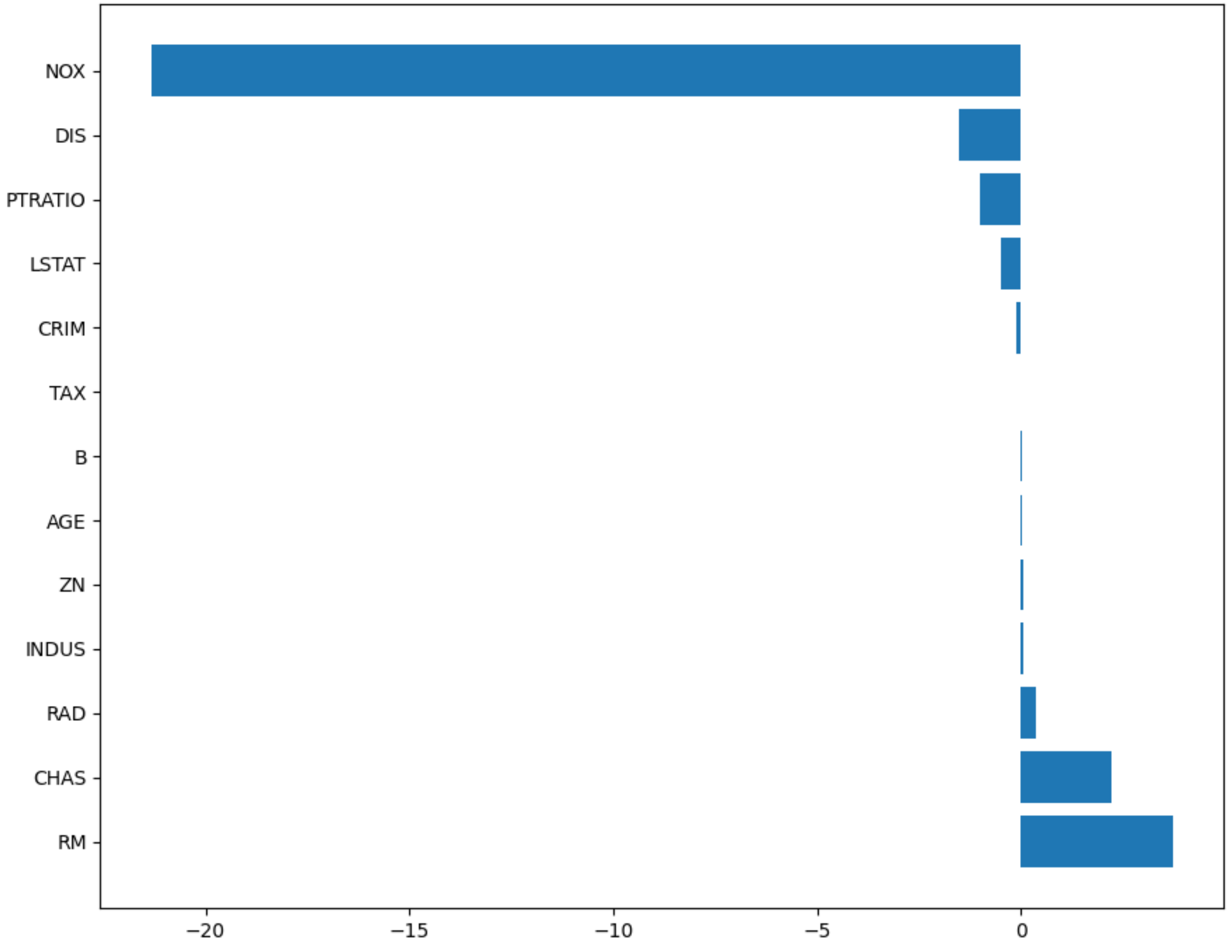
4 LinearRegression 22.798819

```
plot_coef(x_train.columns, lasso_100.coef_)
```

```
lasso_100.coef_
```

```
array([-0.        ,  0.        , -0.        ,  0.        , -0.        ,
        0.        , -0.        ,  0.        , -0.        , -0.02121177,
       -0.        ,  0.00358064, -0.        ])
```

```
plot_coef(x_train.columns, lasso_001.coef_)
```




```
lasso_001.coef_
```

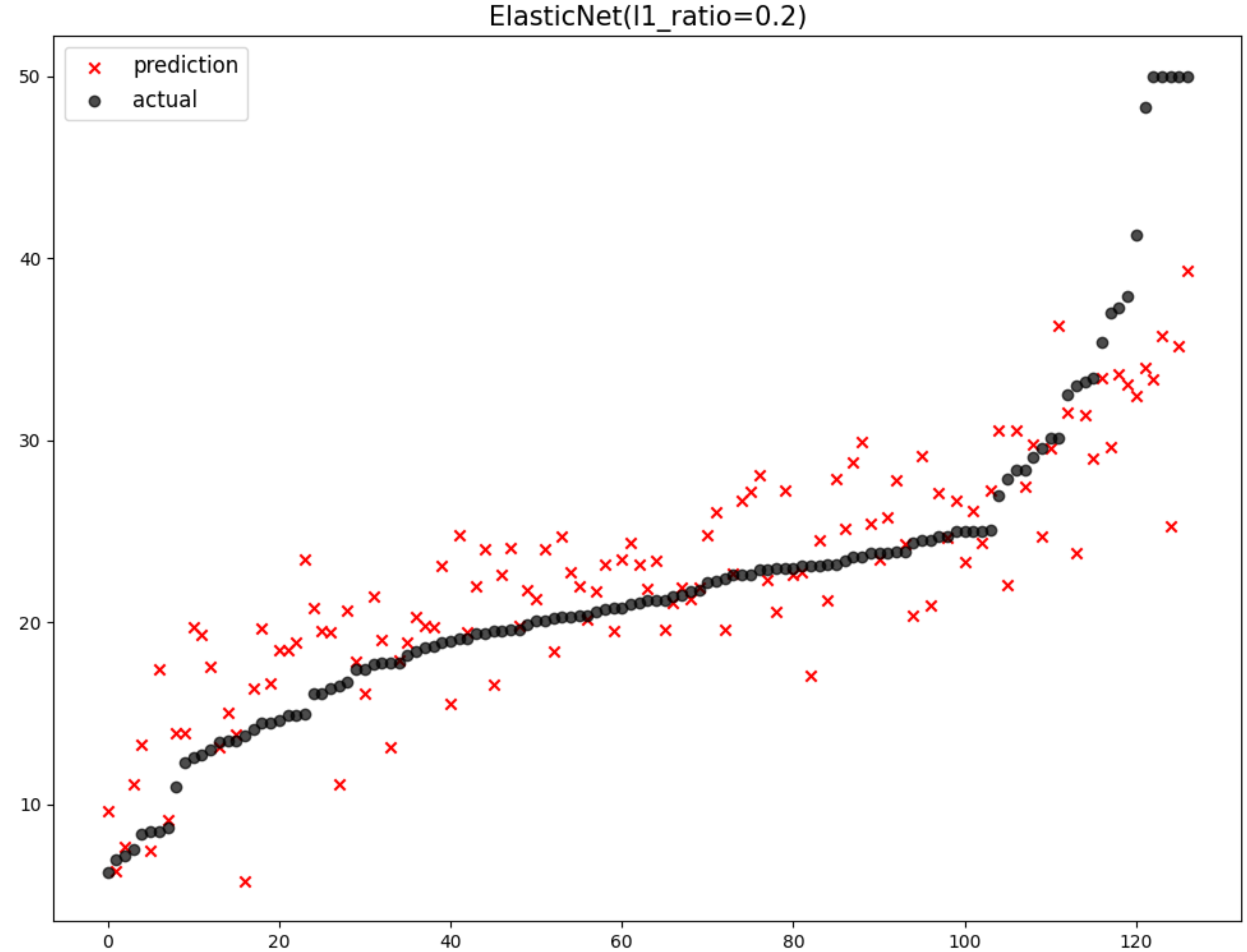
```
array([ -0.12212745,   0.05052368,   0.05271314,   2.19726572,
       -21.32373087,   3.70702032,   0.00944953,  -1.52676922,
         0.35305813,  -0.013475  ,  -1.01514003,   0.00805696,
        -0.51226653])
```

ElasticNet

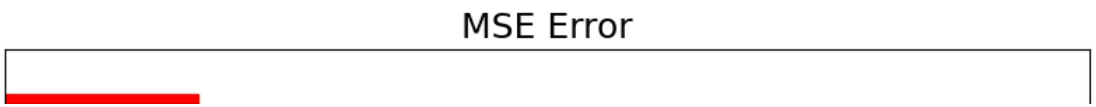
l1_ratio (default=0.5)

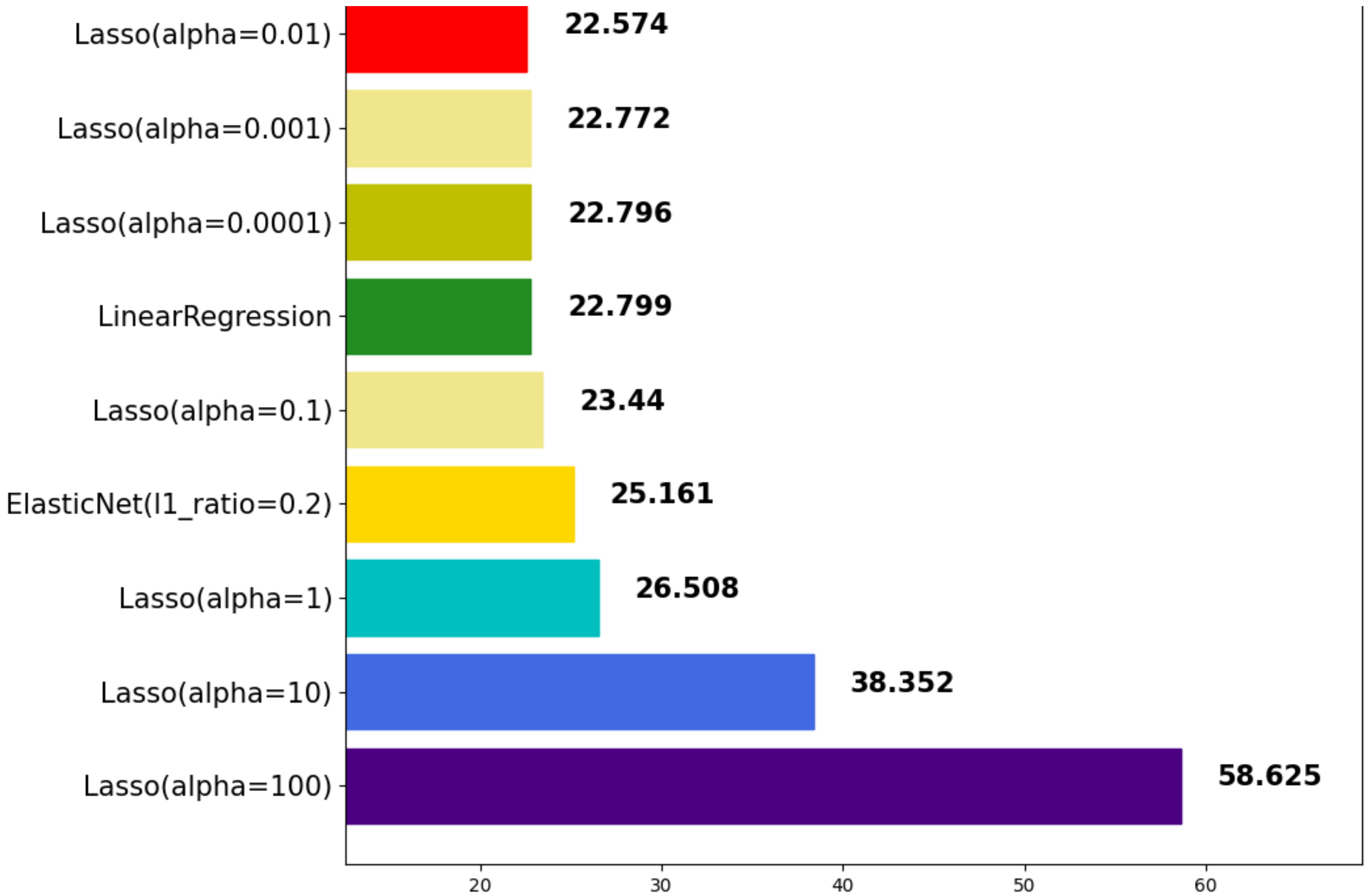
- l1_ratio = 0 (L2 규제만 사용).
- l1_ratio = 1 (L1 규제만 사용).
- 0 < l1_ratio < 1 (L1 and L2 규제의 혼합사용)

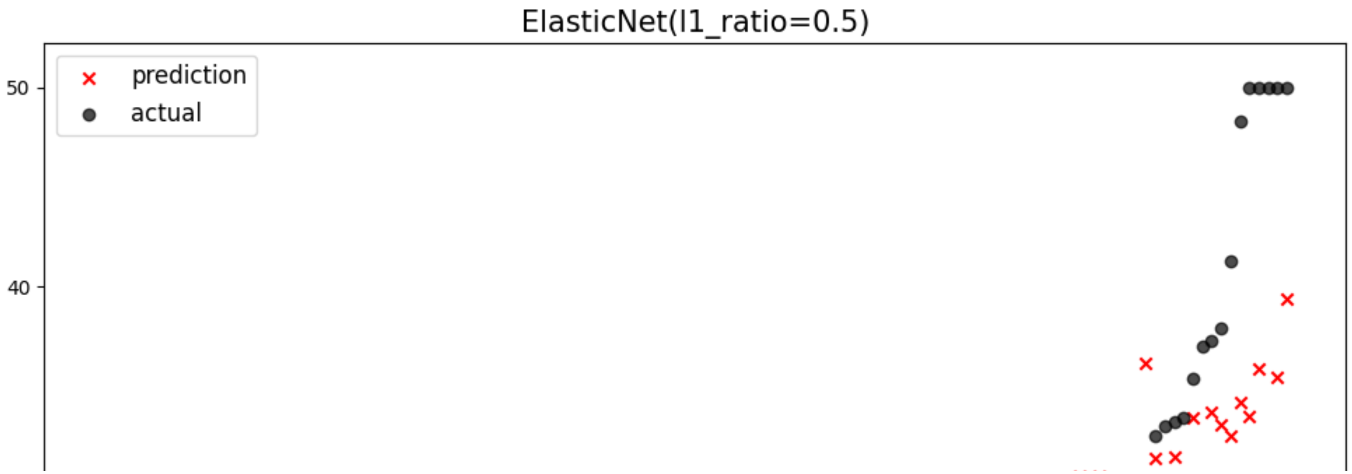
```
from sklearn.linear_model import ElasticNet

ratios = [0.2, 0.5, 0.8]

for ratio in ratios:
    elasticnet = ElasticNet(alpha=0.5, l1_ratio=ratio)
    elasticnet.fit(x_train, y_train)
    pred = elasticnet.predict(x_test)
    mse_eval('ElasticNet(l1_ratio={})'.format(ratio), pred, y_test)
```



	model	mse
0	Lasso(alpha=100)	58.625379
1	Lasso(alpha=10)	38.352168
2	Lasso(alpha=1)	26.507706
3	ElasticNet(l1_ratio=0.2)	25.161255
4	Lasso(alpha=0.1)	23.440245
5	LinearRegression	22.798819
6	Lasso(alpha=0.0001)	22.796079
7	Lasso(alpha=0.001)	22.771857
8	Lasso(alpha=0.01)	22.573644




Lasso(alpha=0.01)
22.574
Lasso(alpha=0.001)
22.772
Lasso(alpha=0.0001)
22.796
LinearRegression
22.799
Lasso(alpha=0.1)
23.44
ElasticNet(l1_ratio=0.2)
25.161
Lasso(alpha=1)
26.508
Lasso(alpha=10)
38.352
Lasso(alpha=100)
58.625
20
30
40
50
60


ElasticNet(l1_ratio=0.5)
prediction
actual
50
40