

다음 발표: 전기공학과 12181350 권준혁

- 데이터 분할
- 결정트리회귀 모델
- 랜덤포레스트회귀 모델
- 교차검증
- 두 모델의 특성 중요도를 비교
- 결론: 랜덤포레스트회귀 모델이
‘중고차 가격 예측’에 더 적합한 모델임을 알 수 있다.

훈련 세트와 테스트 세트로 분할

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(data,
target, test_size = 0.2, shuffle = True, random_state = 34)
#훈련 세트(train_input, train_target)와 테스트 세트(test_input, test_target)로 나눈다.
#train_test_split함수의 여러 옵션 값들을 잘 확인해서 사용한다.
```

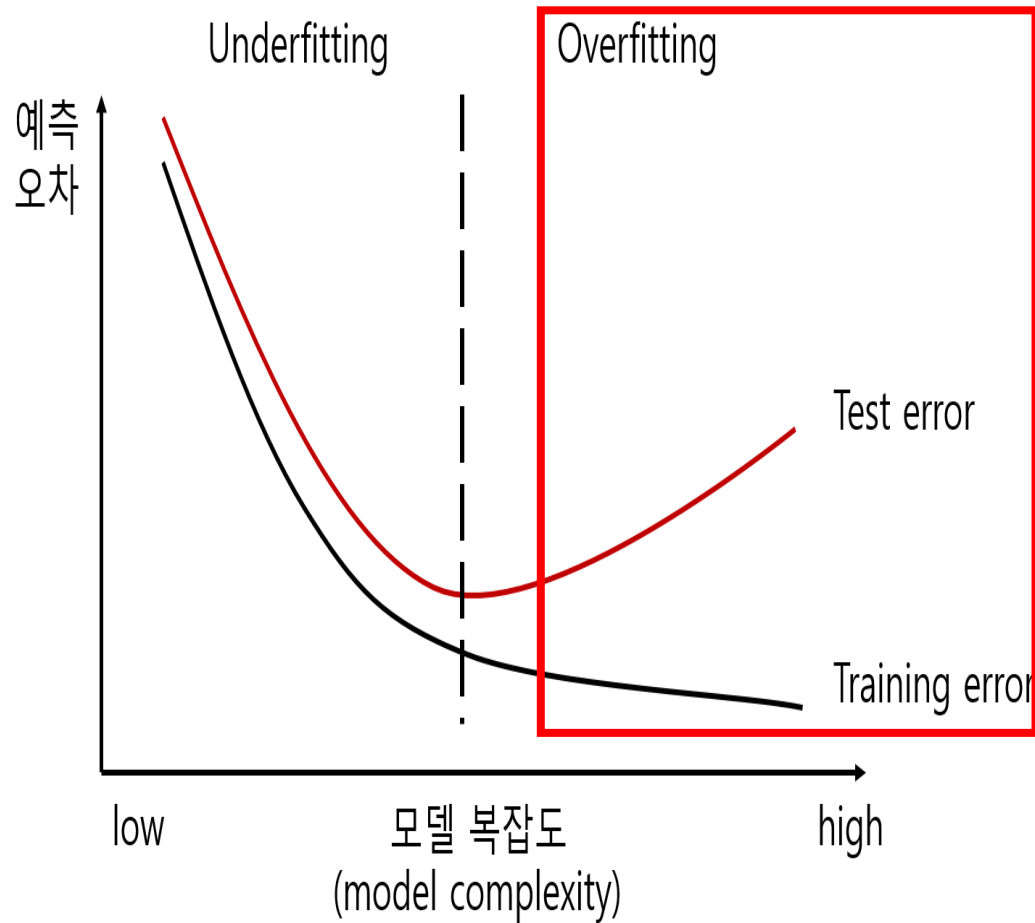
훈련 세트 :

모델 훈련(가중치 찾기)에 쓰이는 데이터 세트

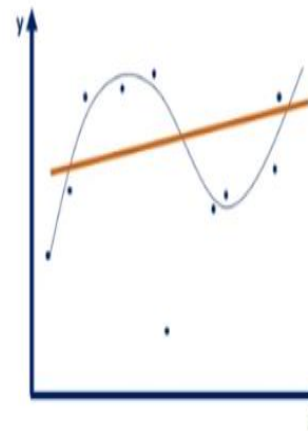
테스트 세트 :

모델의 실제 정확도를 판별하기 위해 쓰이는 데이터 세트.

오버피팅이란?



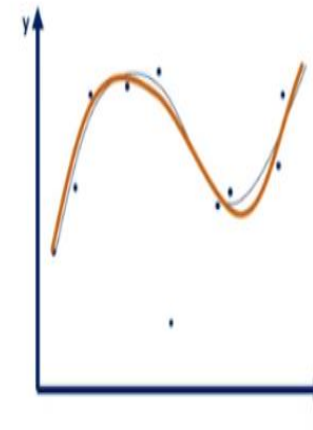
An **underfitted** model



Doesn't capture any logic

- High loss
- Low accuracy

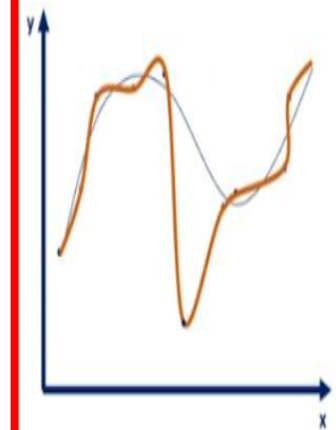
A **good** model



Captures the underlying logic of the dataset

- Low loss
- High accuracy

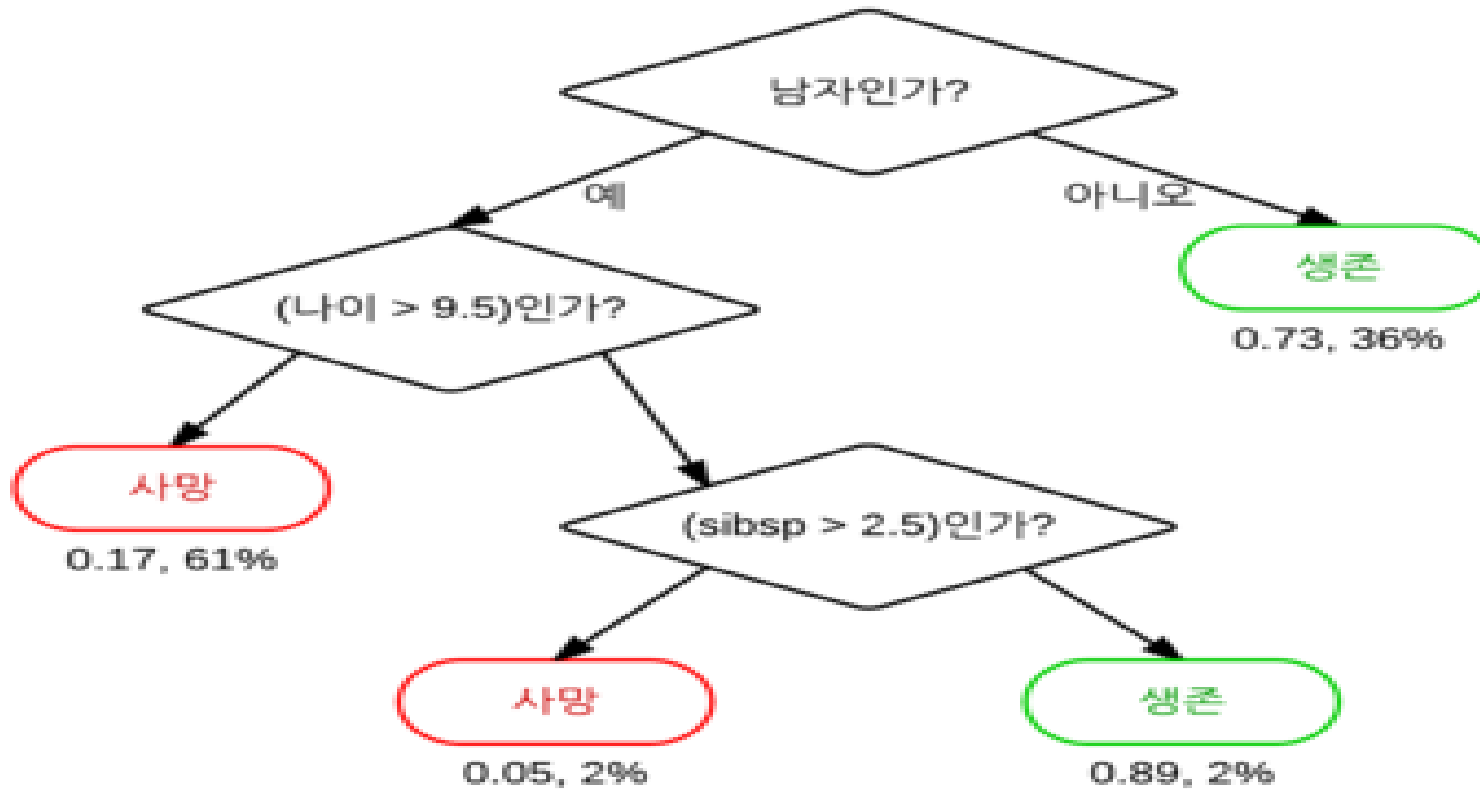
An **overfitted** model




Captures all the noise, thus "missed the point"

- Low loss
- Low accuracy

DecisionTreeRegressor



타이타닉호 탑승객의 생존 여부를 나타내는 결정 트리. 
("sibsp"는 탑승한 배우자와 자녀의 수를 의미한다.) 앞 아래의
숫자는 각각 생존 확률과 탑승객이 그 앞에 해당될 확률을 의
미한다.

DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor  
dt = DecisionTreeRegressor(random_state = 34)
```

#중고차 가격 예측과 같이, 회귀 모델에서는 분류 모델인 *DecisionTreeClassifier*가 아닌
#회귀 모델인 *DecisionTreeRegressor*를 사용해야 한다.

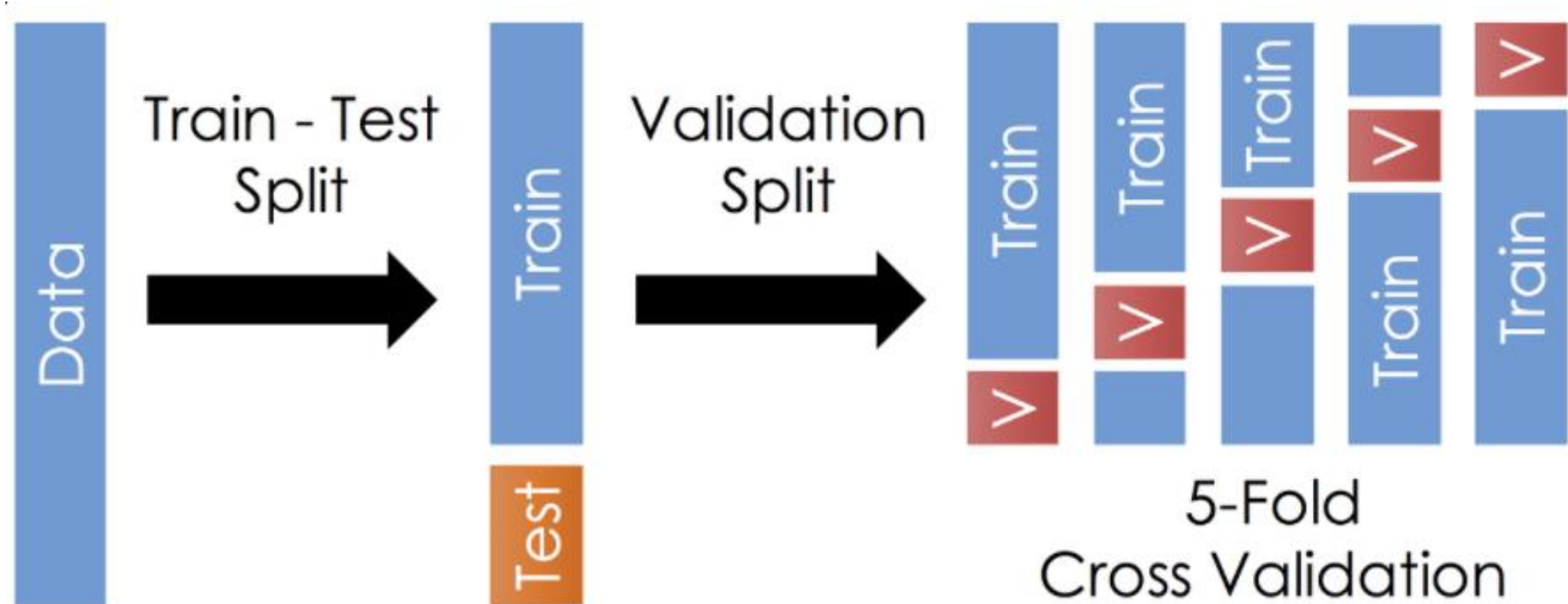
```
dt.fit(train_input, train_target)  
print(dt.score(train_input, train_target))  
print(dt.score(test_input, test_target))
```

#높은 점수가 나왔지만, 테스트 세트와 차이가 크므로 과대적합이다. 이후에 교차검증을 시도해본다.

0.999993149338792

0.8987533037919507

교차검증



DecisionTreeRegressor를 교차검증

```
from sklearn.model_selection import cross_validate
scores = cross_validate(dt, train_input, train_target)
print(np.mean(scores['test_score']))
```

#5-폴드 교차 검증을 해서 교차 검증을 5회 시도한다.

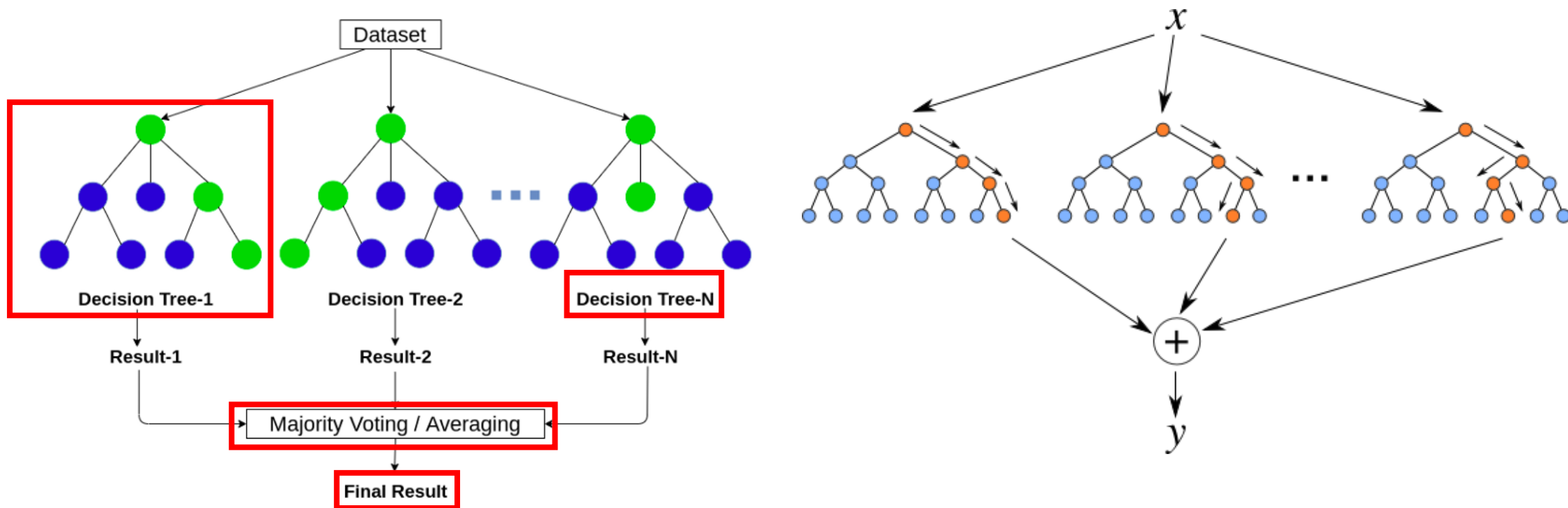
#다시 말해, 결정트리회귀를 다른 세트들로 총 5회 했을 때 정확도를 뜻한다.

0.8780421659174685

0.999993149338792

0.8987533037919507

RandomForestRegressor



결정트리회귀(DecisionTreeRegressor)를 N번
시행해서 평균을 낸 결과를 보여주는 것이
RandomForestRegressor이다.

RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
forest = RandomForestRegressor(n_jobs = -1, random_state = 34)  
forest.fit(train_input, train_target)
```

```
print(forest.score(train_input, train_target))
```

```
print(forest.score(test_input, test_target))
```

#DecisionTreeRegressor보다 테스트 세트에 대한 정확도가 높음. 즉, 일반화가 더 잘됨.

#DecisionTreeRegressor를 100회(기본값) 수행해서, 각 트리의 예측을 평균한다.

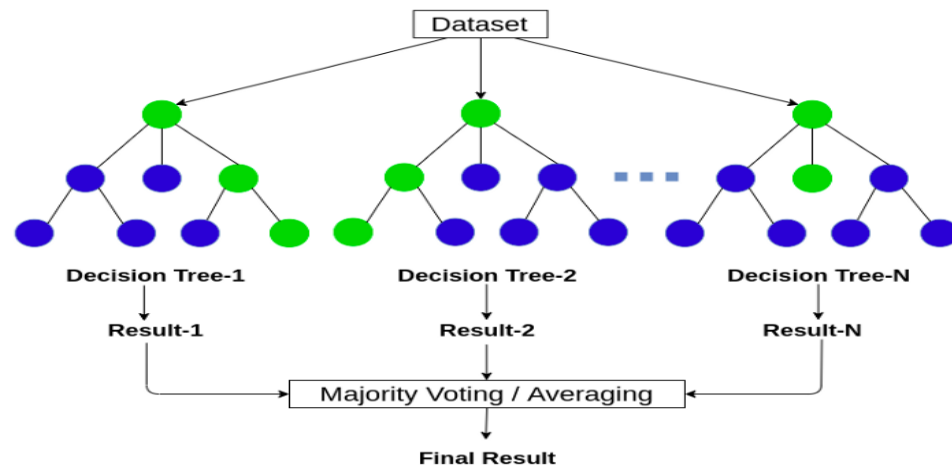
#DecisionTreeRegressor는 특성(feature)가 많을 때, 오버피팅이 되기 쉽다 이번 데이터 세트는 특성이 많아서
#오버피팅 된 것이다.

#이를 해결하기 위해 이를 해결하기 위해 RandomForestRegressor를 활용해서

#Feature의 개수를 주려 작은 결정트리를 여러개로 만들어 하나로 합치는 과정을 거쳐서 더 일반적인 예측을 한다,

```
0.9911032662018787  
0.9512260774841852
```

```
0.999993149338792  
0.8987533037919507
```



RandomForestRegressor를 교차검증

```
from sklearn.model_selection import cross_validate
import numpy as np
scores = cross_validate(forest, train_input, train_target, return_train_score = True, n_jobs = -1)
print(np.mean(scores['train_score']), np.mean(scores['test_score']))
#랜덤포레스트의 교차 검증을 수행하면, 테스트 세트로 검증할 때보다 예측도가 약간은 떨어졌지만,
#그래도 DecisionTreeRegressor보다 더 일반화됐다고 볼 수 있다.
```

```
0.9908856208199041 0.9368537449318787
```

특성 중요도 활용

| Year | Kilometers_Driven | Mileage | Engine | Power | New_Price |
|----------|-------------------|----------|----------|----------|-----------|
| 7.608374 | 11.314475 | 2.772589 | 7.686621 | 4.467440 | 2.436612 |
| 7.605890 | 11.156251 | 2.944439 | 6.905753 | 4.191169 | 2.079847 |
| 7.608871 | 10.126631 | 3.068053 | 7.311886 | 4.686750 | 2.763800 |
| 7.607381 | 10.741384 | 2.986187 | 6.990257 | 4.220243 | 2.453149 |
| 7.607878 | 11.512925 | 2.653946 | 7.669962 | 5.282630 | 4.300184 |
| ... | ... | ... | ... | ... | ... |
| 7.606885 | 11.775290 | 2.977568 | 7.286876 | 4.686289 | 2.041413 |
| 7.605392 | 11.002100 | 2.833213 | 7.311218 | 4.648186 | 2.457979 |

훈련 세트인 train_input 데이터프레임의 83개의 특성 중 Year ~ New_Price까지의 6개의 특성이 모델을 적용할 때 중요하게 작용한다. (가중치가 큰 요소들이다)

DecisionTreeRegressor의 특성 중요도

```
dt_f_i = (dt.feature_importances_)
dt_f_i[83:90] #Year~New_price
print("특성 중요도: Year, Kilometers_Driven, Mileage, Engine, Power, New_Price")
print("특성 중요도: ", dt_f_i[83:90])
```

특성 중요도: Year, Kilometers_Driven, Mileage, Engine, Power, New_Price

특성 중요도: [0.22186042 0.01431341 0.01445565 0.13736743 0.51418665 0.03783751]

RandomForestRegressor의 특성 중요도

```
f_f_i = forest.feature_importances_
```

```
f_f_i[83:90] #Year~New_price
```

#위에서 봤던 `DecisionTreeRegressor`는 `Power`의 중요도가 0.5가 넘어섰다. 중요도가 다소 치우친 것이다.

#그러나 `RandomForestRegressor`는 `Power`의 중요도가 0.147로,

#그리고 나머지 중요도가 낮았던 특성들의 중요도가 다시 상승했다.

#이는 랜덤 포레스트가 특성의 일부를 랜덤하게 선택하여 결정 트리를 훈련하기 때문이다.

#그 결과 하나의 특성에 과도하게 집중하지 않고 좀 더 많은 특성이 훈련에 기여할 기회를 얻는다.

#이는 과대적합을 줄이고 일반화 성능을 높이는 데 도움이 된다.

```
array([0.22074151, 0.01737459, 0.01609983, 0.1473653 , 0.27597517,  
       0.26893453])
```

결론

[중고차 가격 예측]을 할 때, DecisionTreeRegressor 모델보다 RandomForestRegressor 모델이 더 적합한 모델임을 알 수 있었다.

이유:

- 중고차에 대한 데이터가 83개의 특성을 포함하는데, 이렇게 특성이 많은 경우에 DecisionTreeRegressor를 사용하면 오버피팅이 될 수 있다.
- 이를 해결하기 위해, RandomForestRegressor 모델을 사용한다.

한계:

RandomForestRegressor를 사용해도 오버피팅이 생긴다.