Machine Learning with Python

Life is too short, You need Python

실습 내용

- Grid Search로 Decision Tree 알고리즘 모델을 튜닝합니다.

1.환경 준비

- 기본 라이브러리와 대상 데이터를 가져와 이후 과정을 준비합니다.

In [1]:
```
# 라이브러리 불러오기
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings(action='ignore')
%config InlineBackend.figure_format = 'retina'
```

In [2]:
```
# 데이터 읽어오기
path = 'https://raw.githubusercontent.com/jangrae/csv/master/boston.csv'
data = pd.read_csv(path)
```

2.데이터 이해

- 분석할 데이터를 충분히 이해할 수 있도록 다양한 탐색 과정을 수행합니다.

In [3]:
```
# 상위 몇 개 행 확인
data.head()
```

Out[3]:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

데이터 설명

- crim: 자치시(Town)별 1인당 범죄율
- zn: 25,000 평방피트를 초과하는 거주지역 비율
- indus: 비소매상업지역이 점유하고 있는 토지 비율
- chas: 찰스강에 대한 더미 변수 (= 1 강 경계에 위치; 0 나머지)
- nox: 10ppm당 농축 일산화질소
- rm: 주택 1가구당 평균 방 개수
- age: 1940년 이전에 건축된 소유주택 비율
- dis: 5개 보스턴 직업센터까지 접근성 지수
- rad: 방사형 도로까지의 접근성 지수
- tax: 10,000달러 당 재산세율
- ptratio: 자치시(Town)별 학생/교사 비율
- black: 1000(Bk - 0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 의미
- lstat: 모집단 하위 계층의 비율(%)
- medv: 본인 소유 주택가격(중앙값) (단위:$1,000)

In [4]:
```
# 기술통계 확인
data.describe()
```

Out[4]:

	crim	zn	indus	chas	nox	rm	age	dis
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500

3.데이터 준비

- 전처리 과정을 통해 머신러닝 알고리즘에 사용할 수 있는 형태의 데이터를 준비합니다.

1) x, y 분리

In [5]:
```
# target 확인
target = 'medv'

# 데이터 분리
x = data.drop(target, axis=1)
y = data.loc[:, target]
```

2) 학습용, 평가용 데이터 분리

In [6]:
```
# 모듈 불러오기
from sklearn.model_selection import train_test_split

# 데이터 분리
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)
```

4.성능 예측

- k-Fold Cross Validation을 사용해 모델의 성능을 예측합니다.

In [23]:
```
# 불러오기
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, r2_score
```

In [24]:
```
# 선언하기
model_dt = DecisionTreeRegressor()
```

In [25]:
```
# 성능예측
cv_score = cross_val_score(model_dt ,x_train, y_train, cv=10)
```

In [26]:
```
# 결과확인
print(cv_score.mean())
```

```
0.7170849956213846
```

5.모델 튜닝

- Grid Search로 튜닝을 진행합니다.

1) 모델 튜닝

- 성능을 확인할 파라미터를 딕셔너리 형태로 선언합니다.

- 기존 모델을 기본으로 GridSearchCV 알고리즘을 사용하는 모델을 선언합니다.
- 다음 정보를 최종 모델에 파라미터로 전달합니다.
  - 기본 모델 이름
  - 파라미터 변수
  - cv: K-Fold 분할 개수(기본값=5)
  - n_iter: 시도 횟수(기본값=10)
  - scoring: 평가 방법

In [27]:
```
# 파라미터 선언
  # max_depth: 1~50
param = {'max_depth': range(1, 51)}
```

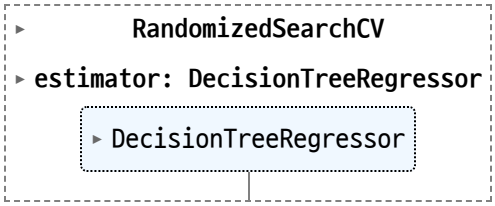
- 다음 두 가지 모델을 선언합니다.
  - 기본 모델: 기본 알고리즘을 사용하는 튜닝 대상 모델
  - 최종 모델: RandomizedSearchCV 알고리즘을 사용하는 모델
- 다음 정보를 최종 모델에 파라미터로 전달합니다.
  - 기본 모델 이름
  - 파라미터 변수
  - cv: K-Fold 분할 개수(기본값=5)
  - scoring: 평가 방법

In [28]:
```
# 선언하기
from sklearn.model_selection import RandomizedSearchCV

# Random Search 선언
  # cv=5
  # scoring='r2'
model = RandomizedSearchCV(model_dt,   # 튜닝할 기본 모델
                           param,      # 테스트 대상 매개변수 범위
                           cv=5,       # K-Fold cv 개수
                           scoring='r2' # 평가지표 (회귀여서 r2)
                           )
```

In [29]:
```
# 학습하기
model.fit(x_train, y_train)
```

Out[29]:



2) 결과 확인

- model.cv*results* 속성에 성능 테스트와 관련된 많은 정보가 포함되어 있습니다.
- 이 중 중요한 정보를만 추출해서 확인합니다.
- 다음 3가지는 꼭 기억해야 합니다.
  - model.cv*results*['mean_test_score']: 테스트로 얻은 성능
  - model.best*params*: 최적의 파라미터

- model.best*score*: 최고의 성능

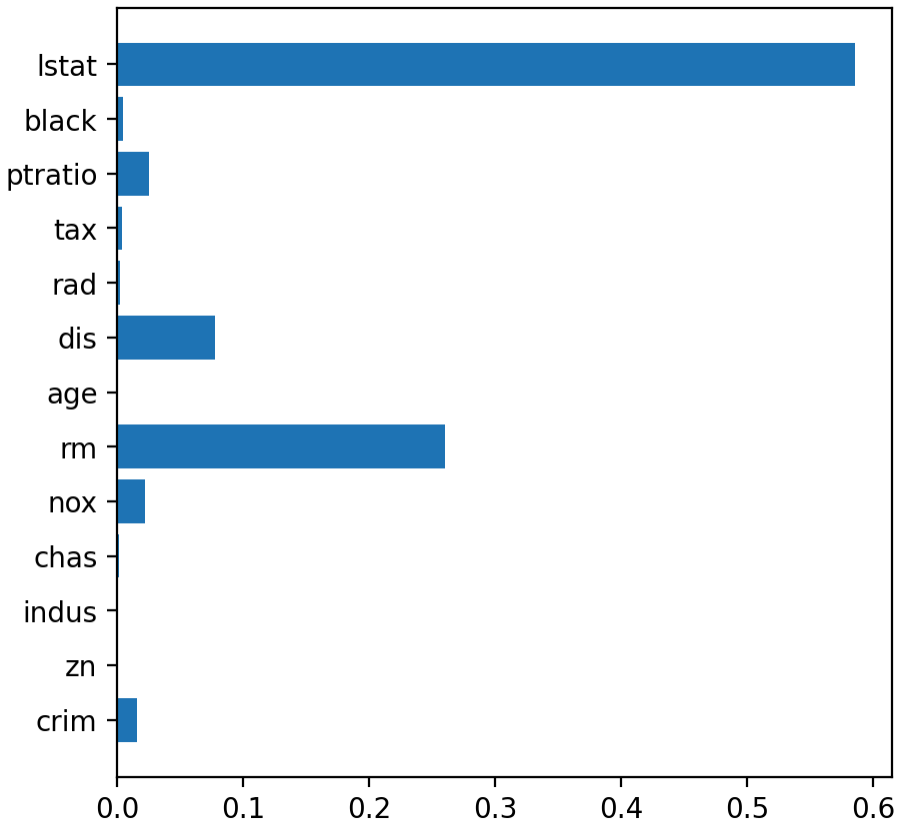
In [30]:
```
# 중요 정보 확인
print('=' * 80)
print(model.cv_results_['mean_test_score'])
print('-' * 80)
print('최적파라미터:', model.best_params_)
print('-' * 80)
print('최고성능:', model.best_score_)
print('=' * 80)
```

```
================================================================================
[0.57894062 0.73898056 0.68251134 0.72418408 0.7476187  0.7123853
 0.71225231 0.72109005 0.71933683 0.71598079]
--------------------------------------------------------------------------------
최적파라미터: {'max_depth': 5}
--------------------------------------------------------------------------------
최고성능: 0.74761870079763
================================================================================
```

3) 변수 중요도

- model.best*estimator* 모델의 변수 중요도를 확인합니다.

In [31]:
```
# 변수 중요도
plt.figure(figsize=(5, 5))
plt.barh(y=list(x), width=model.best_estimator_.feature_importances_)
plt.show()
```



6.성능 평가

- 학습을 통해 예상한 성능과 실제 평가에 따른 성능은 차이가 있을 수 있습니다.
- 예선전에서 성적이 좋았다고 본선에서도 성적이 좋다고 보장할 수는 없겠지요?

In [32]:
```
# 예측하기
y_pred = model.predict(x_test)
```

In [33]:
```
# 평가하기
print('MAE:', mean_absolute_error(y_test, y_pred))
print('R2-Score:', r2_score(y_test, y_pred))
```

```
MAE: 2.7128606133767392
R2-Score: 0.8627140711202218
```

In [ ]:
```

```