

## ch02 앙상블 기법- RandomForest(3)

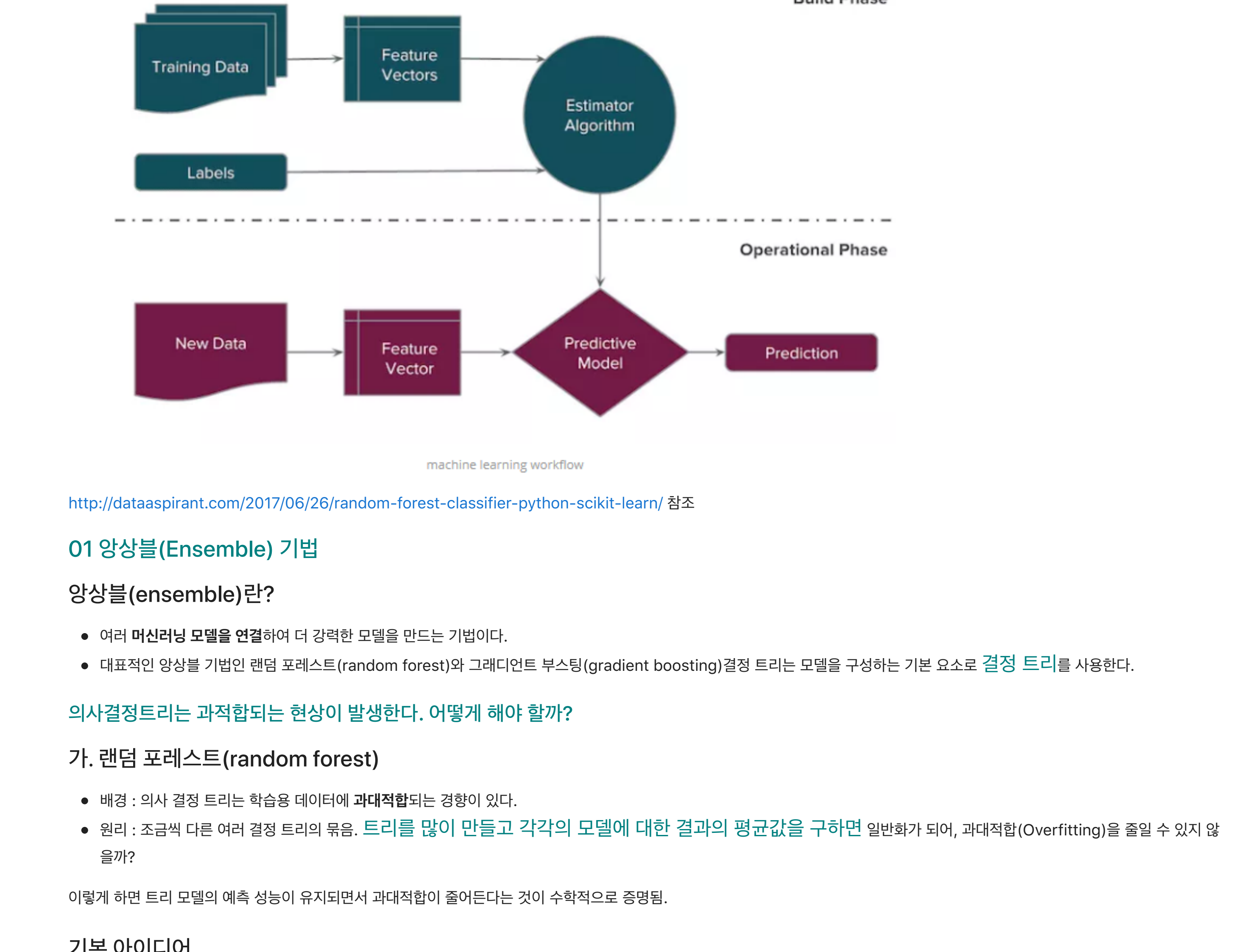
### 학습 내용

1. 랜덤 포레스트의 배경과 원리에 대해 알아본다.
02. 랜덤 포레스트의 파라미터에 대해 알아본다.
03. 트리에서 사용하는 변수의 중요도에 대해서 체크해보고 알아본다.
04. 의사결정트리와 선형회귀를 그래프를 통해 확인하고 알아본다.

### 목차

#### 01 앙상블 기법

##### 02 집값 데이터를 활용한 랜덤 포레스트 분석



<http://dataspirant.com/2017/06/26/random-forest-classifier-python-scikit-learn/> 참조

## 01 앙상블(Ensemble) 기법

### 앙상블(ensemble)란?

- 여러 머신러닝 모델을 연결하여 더 강력한 모델을 만드는 기법이다.
- 대표적인 앙상블 기법인 랜덤 포레스트(random forest)와 그레디언트 부스팅(gradient boosting)결정 트리는 모델을 구성하는 기본 요소로 **결정 트리**를 사용한다.

### 의사결정트리는 과적합되는 현상이 발생한다. 어떻게 해야 할까?

#### 가. 랜덤 포레스트(random forest)

- 배경: 의사 결정 트리는 학습용 데이터에 **과대적합**되는 경향이 있다.
- 원리: 조금씩 다른 여러 결정 트리의 묶음, **트리를 많이 만들고 각각의 모델에 대한 결과의 평균값을 구하면** 일반화가 되어, 과대적합(Overfitting)을 줄일 수 있지 않을까?

이렇게 하면 트리 모델의 예측 성능이 유지되면서 과대적합이 줄어든다는 것이 수학적으로 증명됨.

#### 기본 아이디어

- (1) 만들어지는 각각의 트리는 타깃 예측을 잘해야 한다. 그리고 다른 트리와 구별되어야 한다.
- (2) 랜덤 포레스트는 각각의 트리 성격이 달라지도록 트리 생성 시에 **데이터 샘플링의 무작위성**을 주입한다.
- (3) 트리를 랜덤하게 만드는 방법은 2가지
  - **데이터 포인터를 무작위로 선택하는 방법**
  - **분할 테스트(노드 데이터 조건) feature(특성)을 무작위로 선택** </pre>

#### 나. 랜덤 포레스트 구축

- 01 생성할 **트리의 개수(n\_estimators)**를 선택한다.
- 02 **부트 스트랩 샘플(bootstrap sample)**을 생성
  - n\_samples개의 데이터 포인터 중에서
  - 무작위로 데이터 n\_samples 횟수만큼 반복 추출. (**중복 추출** 될 수 있음)
- 중복 추출로 인해 (대략 1/3정도) 누락될 수 있다. 또는 중복된 데이터가 있을 수 있다.
- 03 생성된 데이터 셋으로 결정 트리를 만든다.
  - 단, 여기서 특성(feature)는 **무작위로 선택된 특성** 중에서 최선의 테스트(조건)를 고른다. (max\_features 매개변수로 몇 개의 **feature**를 고를지는 선택이 가능하
  - 다.)
    - max\_feature 값을 크게 하면 랜덤 포레스트의 트리들은 매우 비슷해지고, 가장 두드러진 특성을 이용해 데이터에 잘 맞춰진다.
    - max\_feature를 낮추면 랜덤 포레스트 트리들은 많이 달라지고 각 트리는 데이터에 맞추기 위해 깊이가 깊어진다.
- 04 결과적으로 부트스트랩 샘플링은 랜덤 포레스트의 트리가 **조금씩 다른 데이터셋을 이용해** 만들어지도록 한다.
  - 각각의 트리는 **전체 특성(feature)의 일부만을 사용**한다. </pre>

## 02 집값 데이터를 활용한 랜덤 포레스트 분석

### 5개의 랜덤 포레스트 모델을 생성

```
In [ ]: from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
```

캐글 코리아 2차 대회 데이터 셋 데이터

- <https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr/data>

항목명	의미	값(가)
id	집을 구분하는 번호	
date	집을 구매한 날짜	
price	집의 가격(Target variable)	
bedrooms	침실의 수	
bathrooms	화장실의 수	
sqft_living	주거 공간의 평방 피트(면적)	
sqft_lot	부지의 평방 피트(면적)	
floors	층의 총 수	
waterfront	집의 전방에 강이 흐르는지 여부 (a.k.a 리버뷰)	
view	집이 얼마나 좋아 보이는지의 정도	
condition	집의 전반적인 상태	
grade	King County grading 시스템 기준으로 매긴 집의 등급	
sqft_above	지하실을 제외한 평방 피트(면적)	
sqft_basement	지하실의 평방 피트(면적)	
yr_built	지어진 년도	
yr_renovated	집을 재건축한 년도	
zipcode	우편번호	
lat	위도	
long	경도	
sqft_living15	2015년 기준 주거 공간의 평방 피트(면적, 집을 재건축했다면, 변화가 있을 수 있음)	
sqft_lot15	2015년 기준 부지의 평방 피트(면적, 집을 재건축했다면, 변화가 있을 수 있음)	

```
In [ ]: import pandas as pd

train = pd.read_csv("house_train.csv")
test = pd.read_csv("house_test.csv")
```

```
In [ ]: train.columns
```

```
Out[ ]: Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

- 예측하고자 하는 값(target)이 price

```
In [ ]: train.head()
```

```
Out[ ]:   id      date      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  ...  grade  sqft_above  sqft_basement  yr_built  yr_renov  
```

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renov
0	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	
1	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	
2	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	
3	20140627T000000	257500.0	3	2.25	1715	6819	2.0	0	0	...	7	1715	0	1995	
4	20150115T000000	291850.0	3	1.50	1060	9711	1.0	0	0	...	7	1060	0	1963	

5 rows × 21 columns

```
In [ ]: X_all = train.drop(['price'], axis=1) # 열 기준 컬럼 삭제
y = train['price']

print(type(X_all), type(y))
print(X_all.shape, y.shape)
```

```
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
(15035, 20) (15035,)
```

```
In [ ]: X_all.columns
```

```
Out[ ]: Index(['id', 'date', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```
In [ ]: X_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15035 entries, 0 to 15034
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  --
0   id                    15035 non-null   int64
1   date                 15035 non-null   object
2   bedrooms             15035 non-null   int64
3   bathrooms            15035 non-null   float64
4   sqft_living          15035 non-null   int64
5   sqft_lot             15035 non-null   int64
6   floors               15035 non-null   float64
7   waterfront           15035 non-null   int64
8   view                 15035 non-null   int64
9   condition            15035 non-null   int64
10  grade                15035 non-null   int64
11  sqft_above           15035 non-null   int64
12  sqft_basement        15035 non-null   int64
13  yr_built             15035 non-null   int64
14  yr_renovated         15035 non-null   int64
15  zipcode              15035 non-null   int64
16  lat                  15035 non-null   float64
17  long                 15035 non-null   float64
18  sqft_living15        15035 non-null   int64
19  sqft_lot15          15035 non-null   int64
dtypes: float64(4), int64(15), object(1)
memory usage: 2.3+ MB
```

```
In [ ]: train.corr()
```

```
Out[ ]:   id      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  condition  grade  sqft_above  sqft_basement  yr_built  yr_renov
0   id      1.000000  0.020899  0.010520  0.104030  0.041725  -0.034077  0.182848  -0.011775  -0.024360  -0.101618  0.078622  -0.073086  0.1955
1   price  -0.020899  1.000000  0.323672  0.525479  0.702899  0.096793  0.262588  0.265738  0.400806  0.039740  0.667211  0.608577  0.0
2   bedrooms  0.010520  0.323672  1.000000  0.530548  0.596974  0.033475  0.189532  -0.004819  0.085703  0.034885  0.375286  0.494867  0.0
3   bathrooms  0.104030  0.525479  0.530548  1.000000  0.755853  0.089308  0.508649  0.075452  0.187488  -0.125907  0.666278  0.688255  0.2
4   sqft_living  0.041725  0.702899  0.596974  0.755853  1.000000  0.176500  0.363193  0.108137  0.282821  -0.054213  0.762543  0.878736  0.0
5   sqft_lot   -0.034077  0.096793  0.033475  0.089308  0.176500  1.000000  0.001535  0.025584  0.080441  -0.002099  0.119906  0.186242  0.0
6   floors     0.182848  0.262588  0.034885  0.508649  0.363193  0.001535  1.000000  0.031159  0.034511  -0.261016  0.462598  0.529476  -0.2
7   waterfront -0.011775  0.265738  -0.004819  0.075452  0.108137  0.025584  0.031159  1.000000  0.389669  0.016113  0.088061  0.081968  0.0
8   view       -0.024360  0.400806  0.085703  0.187488  0.282821  0.080441  0.034511  0.389669  1.000000  0.045255  0.247924  0.172693  0.2
9   condition -0.101618  0.039740  0.034885  -0.125907  -0.054213  -0.002099  -0.261016  0.016113  0.045255  1.000000  -0.143599  -0.152856  0.0
10  grade      0.078622  0.667211  0.375286  0.666278  0.762543  0.119906  0.462598  0.088061  0.247924  -0.143599  1.000000  0.759240  0.0
11  sqft_above  0.073086  0.608577  0.494867  0.688255  0.878736  0.186242  0.529476  0.081968  0.172693  -0.152856  0.759240  1.000000  -0.0
12  sqft_basement -0.050634  0.322218  0.315183  0.282642  0.434017  0.017818  -0.239350  0.071576  0.265880  0.175064  0.162657  -0.048623  1.0
13  yr_built    0.202477  0.047290  0.158799  0.503964  0.319927  0.058686  0.490436  -0.026523  -0.061670  -0.366590  0.440608  0.422431  -0.0
14  yr_renovated -0.029810  0.140808  0.022729  0.065423  0.064893  -0.001451  0.009752  0.104168  0.107605  -0.062342  0.030155  0.031441  0.0
15  zipcode     -0.005761  -0.051498  -0.162081  -0.207500  -0.200745  -0.127709  -0.059107  0.028632  0.089247  0.001106  -0.186541  -0.258474  0.0
16  lat         0.002588  0.301604  -0.011190  0.018110  0.051609  -0.082234  0.049004  -0.014772  0.050285  -0.014772  0.121319  0.001074  0.0
17  long        0.014767  0.023547  -0.135802  0.227669  0.245429  0.227451  0.126983  -0.037922  -0.073511  -0.110742  0.202130  0.347226  -0.0
18  sqft_living15 0.029248  0.586419  0.407394  0.573541  0.760271  0.147562  0.287125  0.091810  0.278267  -0.091407  0.715321  0.737795  0.0
19  sqft_lot15  -0.032269  0.086384  0.027242  0.088120  0.184176  0.728458  -0.010287  0.028255  0.076501  -0.003873  0.119734  0.194226  0.0
```

### Price와의 상관계수 확인

```
In [ ]: train.corr()[['price']]
```

```
Out[ ]:   id      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  condition  grade  sqft_above  sqft_basement  yr_built  yr_renov
0   id      1.000000  0.020899  0.010520  0.104030  0.041725  -0.034077  0.182848  -0.011775  -0.024360  -0.101618  0.078622  -0.073086  0.1955
1   price  -0.020899  1.000000  0.323672  0.525479  0.702899  0.096793  0.262588  0.265738  0.400806  0.039740  0.667211  0.608577  0.0
2   bedrooms  0.010520  0.323672  1.000000  0.530548  0.596974  0.033475  0.189532  -0.004819  0.085703  0.034885  0.375286  0.494867  0.0
3   bathrooms  0.104030  0.525479  0.530548  1.000000  0.755853  0.089308  0.508649  0.075452  0.187488  -0.125907  0.666278  0.688255  0.2
4   sqft_living  0.041725  0.702899  0.596974  0.755853  1.000000  0.176500  0.363193  0.108137  0.282821  -0.054213  0.762543  0.878736  0.0
5   sqft_lot   -0.034077  0.096793  0.033475  0.089308  0.176500  1.000000  0.001535  0.025584  0.080441  -0.002099  0.119906  0.186242  0.0
6   floors     0.182848  0.262588  0.034885  0.508649  0.363193  0.001535  1.000000  0.031159  0.034511  -0.261016  0.462598  0.529476  -0.2
7   waterfront -0.011775  0.265738  -0.004819  0.075452  0.108137  0.025584  0.031159  1.000000  0.389669  0.016113  0.088061  0.081968  0.0
8   view       -0.024360  0.400806  0.085703  0.187488  0.282821  0.080441  0.034511  0.389669  1.000000  0.045255  0.247924  0.172693  0.2
9   condition -0.101618  0.039740  0.034885  -0.125907  -0.054213  -0.002099  -0.261016  0.016113  0.045255  1.000000  -0.143599  -0.152856  0.0
10  grade      0.078622  0.667211  0.375286  0.666278  0.762543  0.119906  0.462598  0.088061  0.247924  -0.143599  1.000000  0.759240  0.0
11  sqft_above  0.073086  0.608577  0.494867  0.688255  0.878736  0.186242  0.529476  0.081968  0.172693  -0.152856  0.759240  1.000000  -0.0
12  sqft_basement -0.050634  0.322218  0.315183  0.282642  0.434017  0.017818  -0.239350  0.071576  0.265880  0.175064  0.162657  -0.048623  1.0
13  yr_built    0.202477  0.047290  0.158799  0.503964  0.319927  0.058686  0.490436  -0.026523  -0.061670  -0.366590  0.440608  0.422431  -0.0
14  yr_renovated -0.029810  0.140808  0.022729  0.065423  0.064893  -0.001451  0.009752  0.104168  0.107605  -0.062342  0.030155  0.031441  0.0
15  zipcode     -0.005761  -0.051498  -0.162081  -0.207500  -0.200745  -0.127709  -0.059107  0.028632  0.089247  0.001106  -0.186541  -0.258474  0.0
16  lat         0.002588  0.301604  -0.011190  0.018110  0.051609  -0.082234  0.049004  -0.014772  0.050285  -0.014772  0.121319  0.001074  0.0
17  long        0.014767  0.023547  -0.135802  0.227669  0.245429  0.227451  0.126983  -0.037922  -0.073511  -0.110742  0.202130  0.347226  -0.0
18  sqft_living15 0.029248  0.586419  0.407394  0.573541  0.760271  0.147562  0.287125  0.091810  0.278267  -0.091407  0.715321  0.737795  0.0
19  sqft_lot15  -0.032269  0.086384  0.027242  0.088120  0.184176  0.728458  -0.010287  0.028255  0.076501  -0.003873  0.119734  0.194226  0.0
```

### 3개의 특징을 선택 후, 데이터 전처리

- 정규화(normalization): 정규화는 값의 범위를 0~1 사이로 만들어준다.
  - Min-Max Normalization: 모든 feature에 대해 각각의 최소값 0, 최대값 1로 나눠 값을 0~1 사이로 변환.
    - (X-MIN)/(MAX-MIN) -> 이상치의 영향을 받을 수 있다.
- 표준화(standardization): 표준화는 데이터가 정규 분포를 따른다는 가정하에 수행. 데이터를 평균은 0, 표준편차는 1로 만들어준다.
  - 이상치 문제를 피할 수 있다.

```
In [ ]: sel = ['sqft_living', 'sqft_lot', 'bedrooms'] # 'bedrooms', 'bathrooms',
X = X_all[sel]
y = train['price']

nor_X = MinMaxScaler().fit_transform(X) # 입력 데이터 정규화
print('정규화 : ', nor_X.shape)
```

정규화 : (15035, 3) (15035,)

```
In [ ]: # 정규화 데이터 사용
X_train, X_test, y_train, y_test = train_test_split(nor_X, y,
                                                    random_state=42)

# 정규화 데이터 사용 안함.
# X_train, X_test, y_train, y_test = train_test_split(X, y,
#                                                    random_state=42)
```

```
In [ ]: model = RandomForestRegressor(n_estimators=5, random_state=2) # 5개의 트리
print("학습용 데이터 셋, 테스트용 데이터 셋")
print("모델.score(X_train, y_train)")
print("모델.score(X_test, y_test)")

RandomForestRegressor(n_estimators=5, random_state=2)
학습용 데이터 셋, 테스트용 데이터 셋
0.8922137121180719
0.37937640288308927
```

```
In [ ]: # 학습된 랜덤포레스트의 트리 모델
print(model.estimator_.end="\n\n")
print(model.score, end="\n\n")
print(model.base_estimator) # 기본 모델

[DecisionTreeRegressor(max_features='auto', random_state=1872583848), DecisionTreeRegressor(max_features='auto', random_state=7949214
87), DecisionTreeRegressor(max_features='auto', random_state=111352301), DecisionTreeRegressor(max_features='auto', random_state=1853
453896), DecisionTreeRegressor(max_features='auto', random_state=212398710)]

<bound method RegressorMixin.score of RandomForestRegressor(n_estimators=5, random_state=2)>

DecisionTreeRegressor()
```

```
In [ ]: print("부트스트랩 : {}".format(model.bootstrap))
print("노드 분할 기준 : {}".format(model.criterion)) # 회귀 모델의 경우, 분할 기준은 mse(Mean Squared Error)
```

부트스트랩 : True  
노드 분할 기준 : mse

```
In [ ]: # 사용된 피쳐 중요도와 피쳐의 개수
print(model.feature_importances_)
print(model.n_features_)

[0.69818654 0.25989234 0.04192112]
3
```

```
In [ ]: # model : 모델
# n_features : feature(변수의 개수)
# feature_names : 특징의 이름
def plot_feature_important_up(model, n_features, feature_names):
    imp = model.feature_importances_
    plt.barh(range(n_features), imp, align='center') # 그래프(가로 막대 그래프)
    plt.xticks(np.arange(n_features), feature_names) # y축의 축의 값
    plt.xlabel("feature importance") # y축 라벨(제목)
    plt.ylabel("feature") # y축 제목
    plt.ylim(-1, n_features) # y축의 범위 지정
```

```
In [ ]: feature_names = sel # 선택된 피쳐의 이름
n_features = X.shape[1] # 선택된 피쳐의 개수
plot_feature_important_up(model, n_features, feature_names) # 피쳐의 중요도 확인
```



### 5개의 트리

- 다섯 개의 트리가 만든 결정 경계는 완벽하게 다르다.
- 랜덤 포레스트는 개개의 트리는는 덜 과대적합된다.
- 실제로는 매우 많은 트리(수백, 수천개)를 사용하여 더 부드러운 경계가 만들어짐.

```
In [ ]: %time
X_train, X_test, y_train, y_test = train_test_split(nor_X, y, random_state=42)
model_5 = RandomForestRegressor(n_estimators=5, random_state=2) # 5개의 트리
model_5.fit(X_train, y_train)
```

CPU times: user 156 ms, sys: 6.49 ms, total: 163 ms  
Wall time: 215 ms

```
Out[ ]: RandomForestRegressor(n_estimators=5, random_state=2)
```

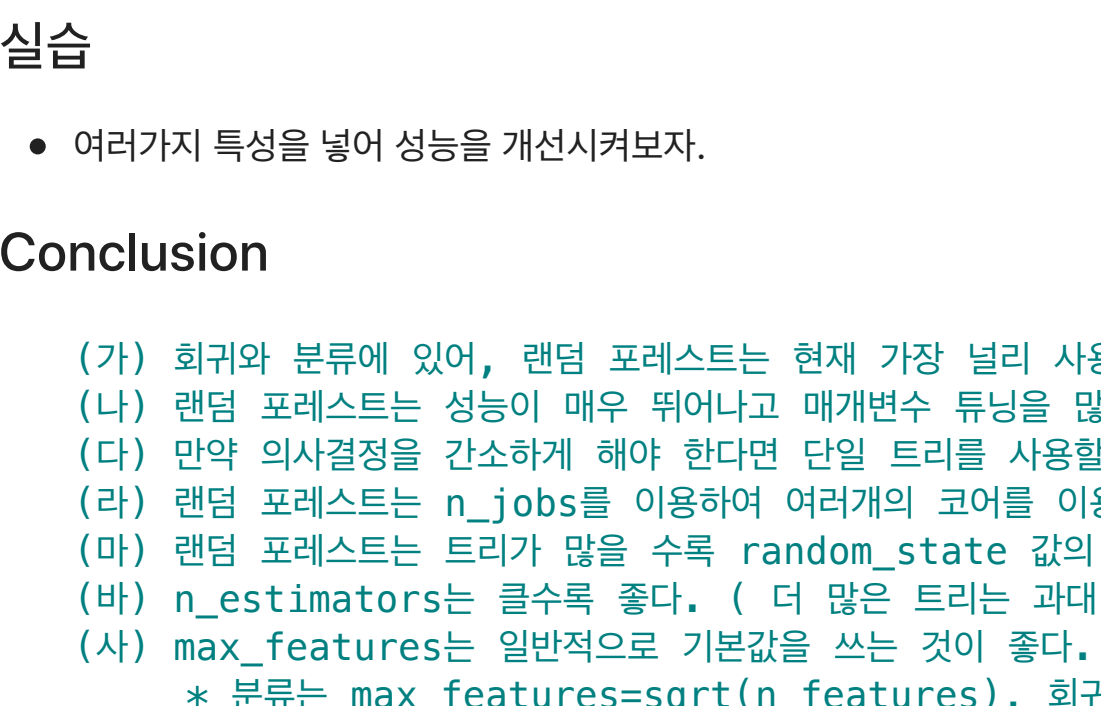
```
In [ ]: model_5
```

```
Out[ ]: RandomForestRegressor(n_estimators=5, random_state=2)
```

```
In [ ]: print(model_5.score(X_train, y_train)) # 학습용 데이터의 결정계수 확인
print(model_5.score(X_test, y_test)) # 테스트용 데이터의 결정계수 확인

0.89221371211808399
0.37937640288308927
```

```
In [ ]: n_features = X.shape[1]
plot_feature_important_up(model, n_features, feature_names)
```



### 5개의 모델 각각에 대한 정확도 평가

```
In [ ]: for model in model_5.estimators_:
    model.fit(X_train, y_train)
    print("훈련 세트 정확도 : {:.3f}".format(model.score(X_train, y_train)))
    print("테스트 세트 정확도 : {:.3f}".format(model.score(X_test, y_test)))
```

훈련 세트 정확도 : 0.977  
테스트 세트 정확도 : 0.094  
훈련 세트 정확도 : 0.997  
테스트 세트 정확도 : 0.056  
훈련 세트 정확도 : 0.997  
테스트 세트 정확도 : 0.087  
훈련 세트 정확도 : 0.997  
테스트 세트 정확도 : 0.059  
훈련 세트 정확도 : 0.997  
테스트 세트 정확도 : 0.069

### 100개 모델에 각각에 대한 정확도 평가 - 10개 단위 출력으로 확인