머신러닝 실습

머신러닝 기반 빅데이터 엔지니어링 과정

빅데이터 X Campus (단국대학교)

2018.08

데이터사이언스 학과 이성신 석사과정

주요 단계

- 큰 그림 그려보기
- 데이터를 구하기
- 데이터로부터 통찰을 얻기 위해 탐색하고 시각화하기
- 머신러닝 알고리즘을 위해 데이터를 준비
- 모델을 선택하고 훈련
- 모델을 상세하게 조정
- 솔루션을 제시
- 시스템을 론칭하고 모니터링하며 유지 보수하기

내용출처: 핸즈온 머신러닝

공개 데이터셋 종류

- 공개 데이터 저장소
  - UC 얼바인 머신러닝 저장소: http://archive.ics.uci.edu/ml/
  - 캐글 데이터셋: http://www.kaggle.com/datasets
  - 아마존 AWS 데이터셋: http://aws.amazon.com/ko/datasets
- 메타 포털(공개 데이터 저장소가 나열되어 있음)
  - http://dataportals.org/
  - http://opendatamonitor.eu/
  - http://quandl.com
- 인기 있는 공개 데이터 저장소가 나열되어 있는 다른 페이지
  - 위키백과 머신러닝 데이터셋 목록: https://goo.gl/SJHN2k
  - Quora.com 질문: http://goo.gl/zDR78y
  - 데이터셋 서브레딧: http://www.reddit.com/r/datasets

내용출처: 핸즈온 머신러닝

데이터 분석

- statLib 저장소에 있는 캘리포니아 주택가격 데이터셋을 사용하여 데이터를 분석

- 간단한 함수를 만들어서 데이터셋을 다운로드

- 함수를 만들어서 데이터셋을 다운로드 할 경우 데이터가 정기적으로 변경되면 최근 데이터가 필요할 때마다 스크립트를 실행하면 되니 유용함!!

- 또는 데이터를 내려받는 일을 자동화하면 여러 기기에 데이터셋을 설치해야 할 때도 편리

내용출처: 핸즈온 머신러닝

데이터 다운로드

- Fetch_housing_data()를 호출하면 코드를 실행하는 작업공간에 datasets/housing 디렉터리를 만듬
- Housing.tgz 파일을 내려받고 같은 디렉터리에 압축을 풀어 housing.csv 파일을 만듬

데이터 다운로드

```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

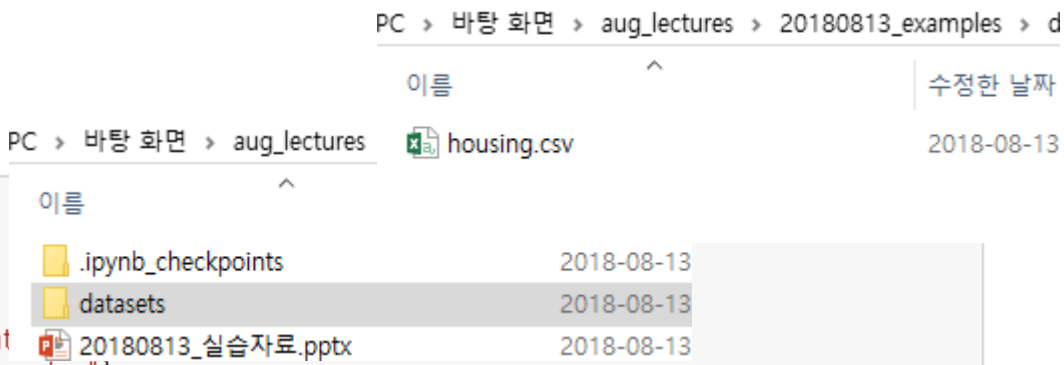
```
fetch_housing_data()
```

내용출처: 핸즈온 머신러닝

데이터 다운로드

- Fetch_housing_data()를 호출하면 코드를 실행하는 작업공간에 datasets/housing 디렉터리를 만듬
- Housing.tgz 파일을 내려받고 같은 디렉터리에 압축을 풀어 housing.csv 파일을 만듬

데이터 다운로드




```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercont
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

```
fetch_housing_data()
```

내용출처: 핸즈온 머신러닝

데이터 다운로드

- 다운로드가 안될경우
- 강의자료실에서 20180813_실습 자료를 다운로드 후 >> datasets >> housing.csv 데이터셋 위치를 HOUSING_PATH 에다가 입력

if not download:

```
1  HOUSING_PATH = r'C:₩Users₩user₩Desktop₩aug_lectures₩20180813_examples₩datasets'
```

데이터 읽어오기

- Pandas 를 사용하여 데이터를 읽어오기
- 데이터를 읽는 부분도 함수로 만들기
  - load_housing_data()
- 이 함수는 모든 데이터를 담은 pandas의 DataFrame 객체를 반환

데이터 읽어오기

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

```
housing = load_housing_data()
housing.head()
```

내용출처: 핸즈온 머신러닝

데이터 구조 훑어보기

- DataFame의 head() 메서드를 사용해서 처음 다섯 행을 확인

```
housing = load_housing_data()
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_ho
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	

- 각 행은 하나의 구역을 의미함
- 특성(column)은 longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value, ocean_proximity 등 10개가 존재

내용출처: 핸즈온 머신러닝

데이터 구조 훑어보기

- DataFrame의 info 메서드는 데이터에 대한 간략한 설명과 특히 전체 행 수, 각 특성의 데이터 타입과 null이 아닌 값의 개수를 확인 할 수 있음

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude           20640 non-null float64
latitude            20640 non-null float64
housing_median_age  20640 non-null float64
total_rooms         20640 non-null float64
total_bedrooms      20433 non-null float64
population          20640 non-null float64
households          20640 non-null float64
median_income       20640 non-null float64
median_house_value  20640 non-null float64
ocean_proximity     20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

total_bedrooms 특성은 20433개만 값이 존재. 207개의 구역은 값이 없는것을 의미(null)

내용출처: 핸즈온 머신러닝

데이터 구조 훑어보기

- DataFrame의 info 메서드는 데이터에 대한 간략한 설명과 특히 전체 행 수, 각 특성의 데이터 타입과 null이 아닌 값의 개수를 확인 할 수 있음

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude           20640 non-null float64
latitude            20640 non-null float64
housing_median_age  20640 non-null float64
total_rooms         20640 non-null float64
total_bedrooms      20433 non-null float64
population          20640 non-null float64
households          20640 non-null float64
median_income       20640 non-null float64
median_house_value  20640 non-null float64
ocean_proximity     20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

ocean_proximity 필드만 빼고 모든 특성은 float64형(숫자형)

내용출처: 핸즈온 머신러닝

데이터 구조 훑어보기

- head() 함수를 사용해서 확인해봤을때 열의 값이 반복되는 것으로 보아서 이 특성은 아마도 범주형(categorical) 일 것임.

ısing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
```

값이 반복

내용출처: 핸즈온 머신러닝

데이터 구조 훑어보기

- 어떤 카테고리가 있고 각 카테고리마다 얼마나 많은 구역이 있는지 value_counts()메서드로 확인

```
housing["ocean_proximity"].value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

내용출처: 핸즈온 머신러닝

데이터 구조 훑어보기

- describe() 메서드를 사용하여 숫자형 특성의 요약 정보를 확인

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.

백분위수

내용출처: 핸즈온 머신러닝

히스토그램 그려보기

- 데이터의 형태를 빠르게 검토하는 다른 방법은 각 숫자형 특성을 히스토그램으로 그려보는 것
- 특성마다 따로 히스토그램을 그릴 수도 있고 전체 데이터셋에 대해 hist() 메서드를 호출하면 모든 숫자형 특성에 대한 히스토그램을 출력

히스토그램 그려보기

```
1 %matplotlib inline
2 import matplotlib.pyplot as plt
3 housing.hist(bins=50, figsize=(20,15))
4 plt.savefig("attribute_histogram_plots")
5 plt.show()
```

아래 보이는 figure 가 .ipynb 코드가 실행되는 위치(폴더)에 저장됨

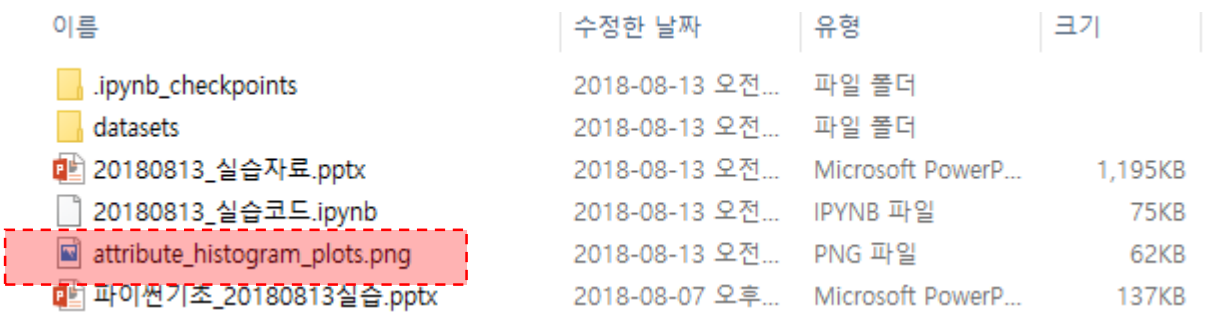
내용출처: 핸즈온 머신러닝

히스토그램 그려보기

- 데이터의 형태를 빠르게 검토하는 다른 방법은 각 숫자형 특성을 히스토그램으로 그려보는 것
- 특성마다 따로 히스토그램을 그릴 수도 있고 전체 데이터셋에 대해 hist() 메서드를 호출하면 모든 숫자형 특성에 대한 히스토그램을 출력

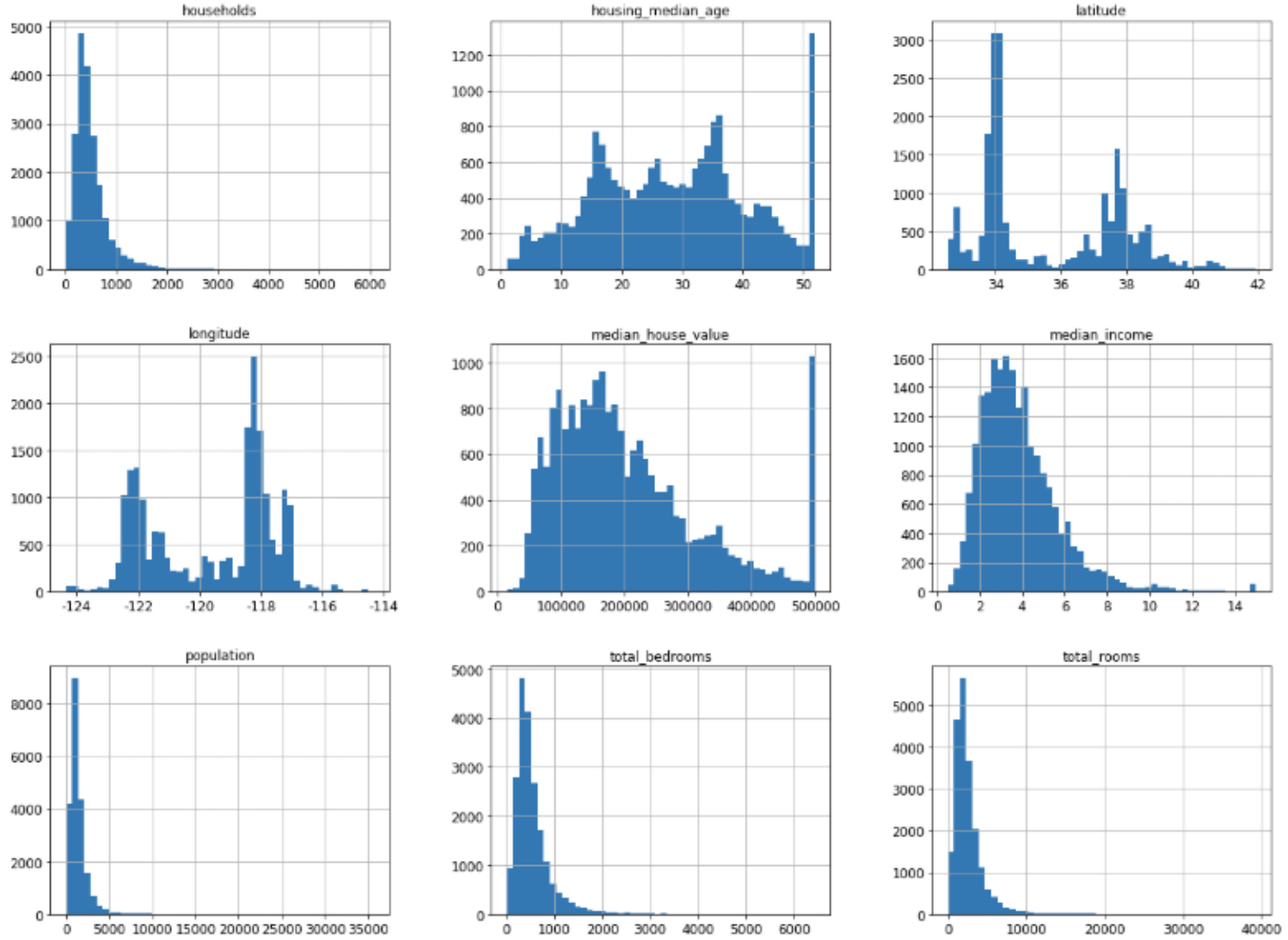
히스토그램 그려보기

```
1 %matplotlib inline
2 import matplotlib.pyplot as plt
3 housing.hist(bins=50, figsize=(20,15))
4 plt.savefig("attribute_histogram_plots")
5 plt.show()
```



내용출처: 핸즈온 머신러닝

히스토그램 그려보기



내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 임의의 난수를 발생하여 특정 %만큼 분리하기
- 이 방법 외에 사이킷런에 train_test_split() 또는 StraightKFold() 함수를 사용해서 만들 수 있음 (이 방법을 더 선호)

```
# 일관된 출력을 위해 유사난수 초기화
np.random.seed(42)
```

```
import numpy as np

# 예시를 위해서 만든 것입니다. 사이킷런에는 train_test_split() 함수가 있습니다
def split_train_test
    shuffled_indices
    test_set_size =
    test_indices = s
    train_indices =
    return data.ilo
```

```
train_set, test_set
print(len(train_set)
```

```
16512 train + 4128 t
```

사람들은 난수 초기값으로 42를 자주 사용. 이 숫자는 '삶, 우주, 그리고 모든 것에 대한 궁극적인 질문의 해답' 인 것 외에는 특별한 의미는 없음. 이 숫자는 더글러스 애덤스의 소설 '은하수를 여행하는 히치하이커를 위한 안내서' 에서 슈퍼컴퓨터인 깊은생각이 이 질문에 대해 750만 년 동안 계산하여 내놓은 답 ㅋㅋ

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 임의의 난수를 발생하여 특정 %만큼 분리하기
- 이 방법 외에 사이킷런에 train_test_split() 또는 StraightKFold() 함수를 사용해서 만들 수 있음 (이 방법을 더 선호)

```
# 일관된 출력을 위해 유사난수 초기화
np.random.seed(42)
```

```
import numpy as np

# 예시를 위해서 만든 것입니다. 사이킷런에는 train_test_split() 함수가 있습니다.
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

```
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

```
16512 train + 4128 test
```

전체 데이터셋중 20%만큼 테스트 데이터셋으로 분리

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 사이킷런 함수를 사용한 학습과 테스트 데이터세트 만들기
- 무작위 샘플링 방식: train_test_split
  - Random seed 는 train_test_split 함수의 option으로 입력가능

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- BUT !! train_test_split 함수는 데이터셋이 충분히 크면 상관없지만 데이터가 작을 경우 문제가 생깁니다.

- 예를 들어 전체 인구를 대표할 수 있는 1,000명을 선택하여 설문조사를 한다고 할때, 전체 인구의 51.3%가 여성이고 48.7%가 남성이라면, 잘 구성된 설문조사는 샘플에서도 이 비율을 유지해야합니다.

- 즉 여성은 513명, 남성은 487명이어야 하는데, 이를 **계층적 샘플링** (Stratified sampling) 이라고 합니다.

- 기본 무작위 샘플링을 사용하면 49%보다 적거나 54%보다 많은 여성이 테스트 세트에 들어갈 확률이 12%이기 때문에 설문조사 결과를 크게 편향시키게 될 수 있습니다.

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 이제부터는…!
- 여러분이 부동산 회사에 막 고용된 데이터 과학자라고 가정하고 데이터셋을 분석해봅시다.
- 부동산 전문가가 중간 소득이 중간 주택 가격을 예측하는 데 매우 중요하다고 이야기해주었다고 가정해봅시다.
- 이 경우 테스트 세트가 전체 데이터셋에 있는 여러 소득 카테고리를 잘 대표해야 합니다.

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 중간 소득이 연속적인 숫자형 특성이므로 소득에 대한 카테고리 특성을 만들어야 합니다.
- 중간 소득 대부분은 $20,000~$50,000 사이에 모여 있지만 일부는 $60,000를 넘기도 합니다.
- 계층별로 데이터셋에는 충분한 샘플 수가 있어야 합니다. 그렇지 않으면 계층의 중요도를 추정하는데 편향이 발생할 것입니다.
- 이 말은 너무 많은 계층으로 나누면 안된다는 뜻이고 각 계층이 충분히 커야 합니다.

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 다음 코드는 중간 소득을 1.5로 나누고(소득의 카테고리 수를 제한하기 위해), ceil 함수를 사용하여 반올림해서 소득 카테고리 특성을 만들고 (이산적인 카테고리를 만들기 위해), 5보다 큰 카테고리를 5로 합칩니다.

```
# 소득 카테고리 개수를 제한하기 위해 1.5로 나눕니다.
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
# 5 이상은 5로 레이블합니다.
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

```
housing["income_cat"].value_counts()
```

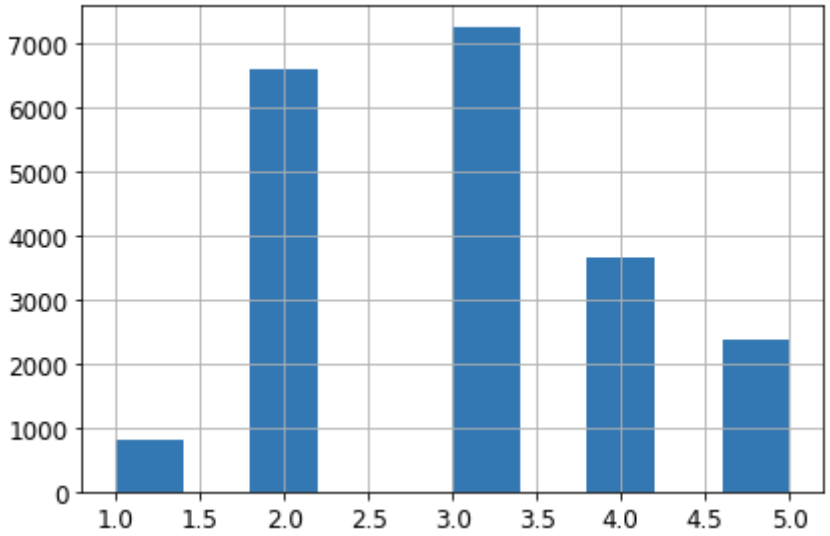
```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: income_cat, dtype: int64
```

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 이 소득 카테고리 히스토그램을 만들어서 확인해봅시다!

```
housing["income_cat"].hist()
save_fig('income_category_hist')
```



내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 이제 소득 카테고리를 기반으로 계층 샘플링을 할 준비가 되었습니다.
- 사이킷런의 StratifiedShuffleSplit() 함수를 사용하여 계층 샘플링을 합니다.

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 전체 데이터셋과 계층 샘플링으로 만든 테스트 세트에서 소득 카테고리 비율을 비교해봅시다.

```
from sklearn.model_selection import train_test_split
def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] / compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] / compare_props["Overall"] - 100
compare_props
```

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 계층 샘플링을 사용해서 만든 테스트 세트가 전체 데이터셋에 있는 소득 카테고리의 비율과 거의 같습니다.
- 반면 일반 무작위 샘플링으로 만든 테스트 세트는 비율이 많이 달라졌습니다.

	Overall	Random	Stratified	Rand. %error	Strat. %error
1.0	0.039826	0.040213	0.039729	0.973236	-0.243309
2.0	0.318847	0.324370	0.318798	1.732260	-0.015195
3.0	0.350581	0.358527	0.350533	2.266446	-0.013820
4.0	0.176308	0.167393	0.176357	-5.056334	0.027480
5.0	0.114438	0.109496	0.114583	-4.318374	0.127011

내용출처: 핸즈온 머신러닝

테스트 데이터 세트 만들기

- 이제 income_cat 특성을 삭제해서 데이터를 원래 상태로 되돌립시다!

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

내용출처: 핸즈온 머신러닝

데이터 시각화

- 지금까지는 다뤄야할 데이터의 종류를 파악하기 위해 데이터를 간단하게 살펴보았습니다. 이제 조금 더 깊이 파악해보겠습니다.
- 먼저 테스트 세트를 떼어놓고 훈련 세트에 대해서만 탐색을 하겠습니다.
- 훈련 세트를 손상시키지 않기 위해 복사본을 만들어서 사용합니다.

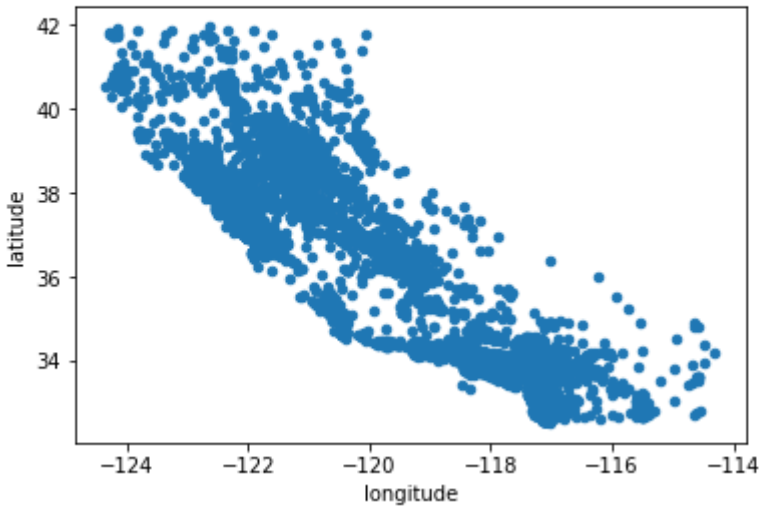
```
housing = strat_train_set.copy()
```

내용출처: 핸즈온 머신러닝

데이터 시각화

- 지리정보(위도와 경도)가 있으니 모든 구역을 산점도로 만들어서 데이터를 시각화 해봅시다.

```
1 ax = housing.plot(kind="scatter", x="longitude", y="latitude")
2 ax.set(xlabel='longitude', ylabel='latitude')
3 plt.savefig("bad_visualization_plot")
```

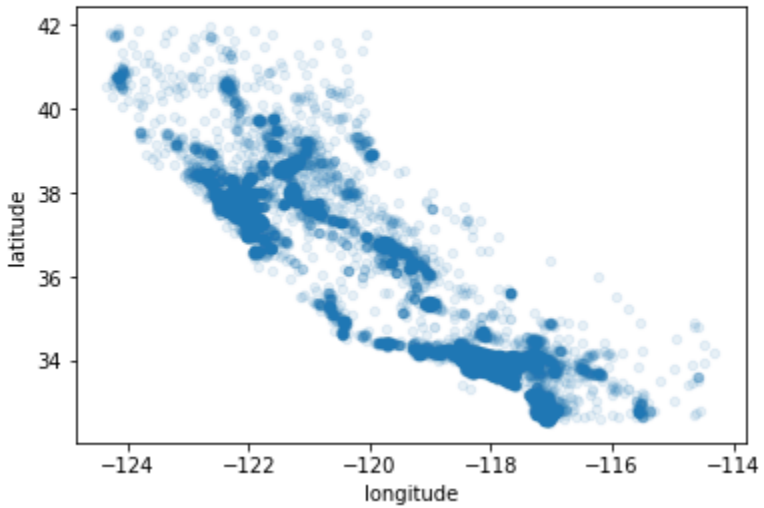


내용출처: 핸즈온 머신러닝

데이터 시각화

- 이 그림에서 특정 패턴을 찾기 힘들기 때문에 alpha 옵션을 0.1로 주어서 데이터 포인트가 밀집된 영역을 잘 표시해보기.

```
1 ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
2 ax.set(xlabel='longitude', ylabel='latitude')
3 plt.savefig("better_visualization_plot")
```



내용출처: 핸즈온 머신러닝

데이터 시각화

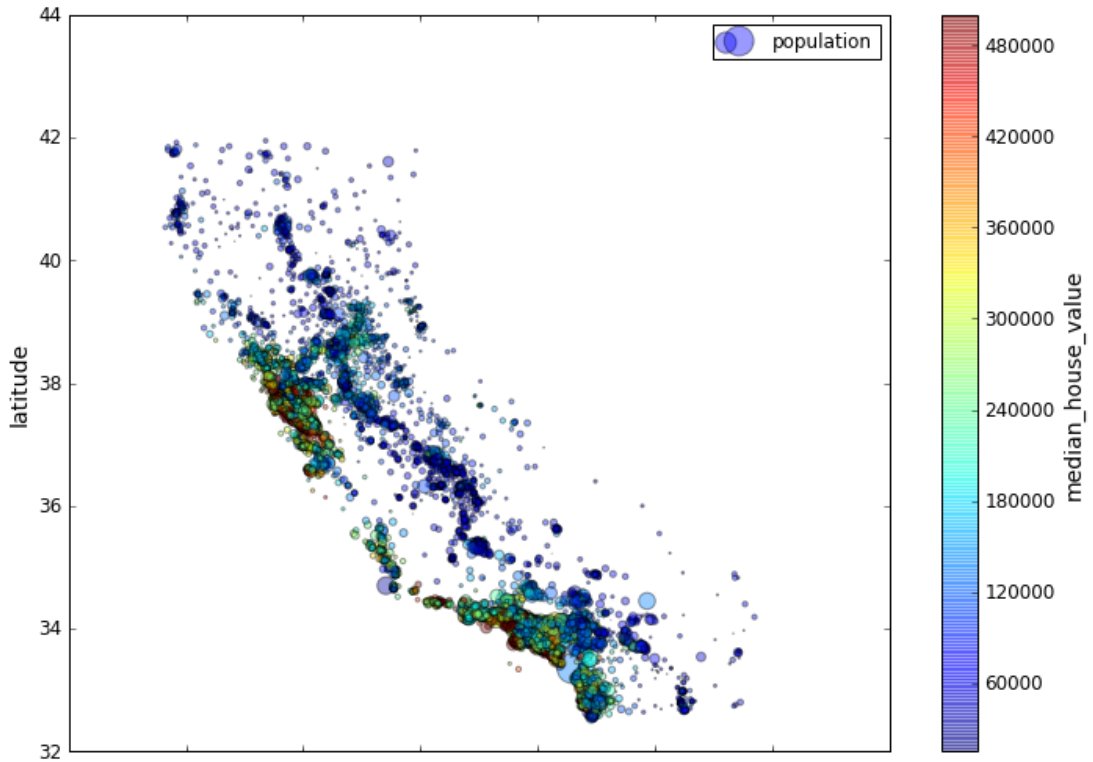
- 이제 주택가격을 나타내 보겠습니다.
- 원의 반지름은 구역의 인구를 나타내고(매개변수 s), 색깔은 가격을 나타냅니다.(매개변수 c).
- 여기에서는 미리 정의된 컬러맵 중 파란색(낮은 가격)에서 빨간색(높은 가격)까지 범위를 가지는 jet을 사용합니다. (매개변수 cmap)

내용출처: 핸즈온 머신러닝

데이터 시각화

```
housing.plot(kind="scatter", x="longitude", y="latitude",
    s=housing['population']/100, label="population",
    c="median_house_value", cmap=plt.get_cmap("jet"),
    colorbar=True, alpha=0.4, figsize=(10,7),
)
plt.legend()
save_fig("housing_prices_scatterplot")
plt.show()
```

```
Saving figure housing_prices_scatterplot
```



내용출처: 핸즈온 머신러닝

데이터 시각화

- 데이터셋이 너무 크지 않으므로 모든 특성 간의 표준 상관계수(standard correlation coefficient, 피어슨의 r 이라고도 부름)를 corr() 메서드를 이용해서 쉽게 계산해보기
- 상관관계의 범위는 -1부터 1까지.
- 1에 가까우면 강한 양의 상관관계를 가진다는 뜻

```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

내용출처: 핸즈온 머신러닝

데이터 시각화

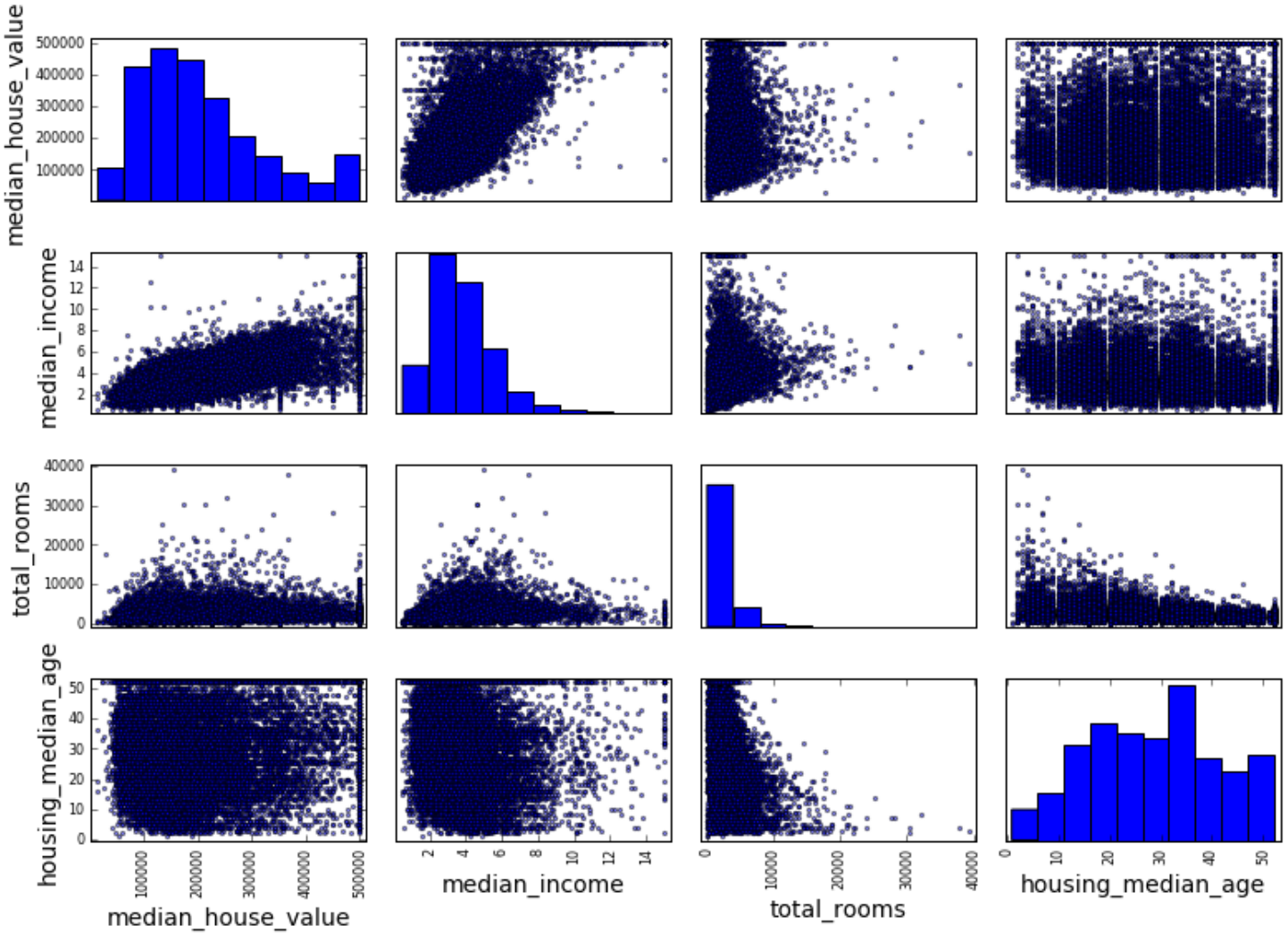
- 특성 사이의 상관관계를 확인하는 다른 방법은 숫자형 특성 사이에 산점도를 그려주는 pandas의 scatter_matrix 함수를 사용하는 것
- 아래 예시는 중간 주택 가격과 상관관계가 높아 보이는 특성 몇 개만 나타낸 그림

```
from pandas.tools.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(11, 8))
save_fig("scatter_matrix_plot")
plt.show()
```

내용출처: 핸즈온 머신러닝

데이터 시각화

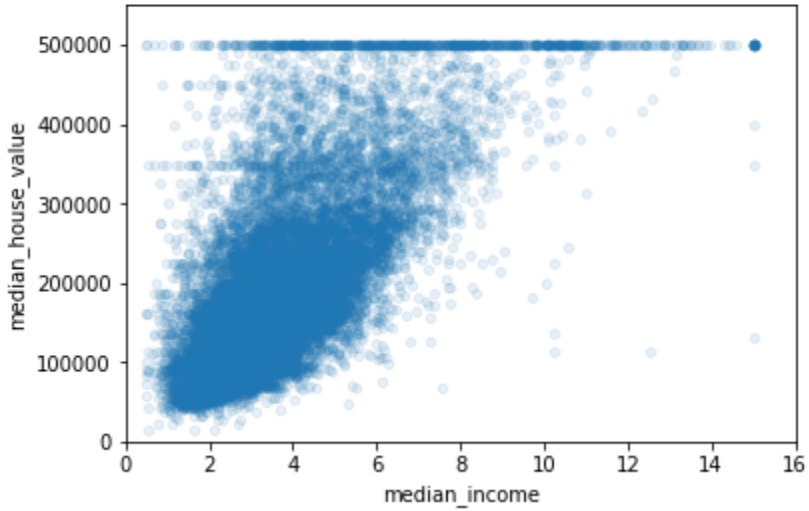


내용출처: 핸즈온 머신러닝

데이터 시각화

- 중간 주택 가격(median_house_value)을 예측하는 데 가장 유용할 것 같은 특성은 중간 소득(median_income)이므로 상관관계 산점도를 확대해 보겠습니다.

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
plt.savefig("income_vs_house_value_scatterplot")
```



내용출처: 핸즈온 머신러닝

실험

- 머신러닝 알고리즘용 데이터를 실제로 준비하기 전에 마지막으로 해볼 수 있는 것은 여러 특성의 조합을 시도해보는 것

- 예를 들어 특정 구역의 방 개수는 얼마나 많은 가구수가 있는지 모른다면 그다지 유용하지 않음.

- 진짜 필요한 것은 가구당 방의 개수!!

- 유의미한 데이터셋을 만들어봅시다.

내용출처: 핸즈온 머신러닝

실험

- 새로운 bedrooms_per_room 특성은 전체 방 개수나 침대 개수보다 중간 주택 가격과의 상관관계가 훨씬 높습니다!!

```
housing["rooms_per_household"] = housing["total_rooms"] / housing["population"]
housing["bedrooms_per_room"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["population_per_household"] = housing["population"] / housing["households"]
```

```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

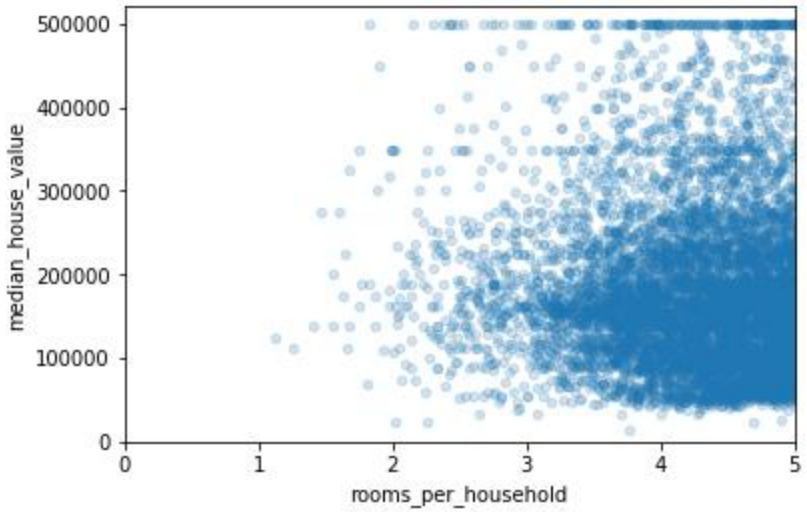
```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.199429
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

내용출처: 핸즈온 머신러닝

데이터 시각화

- page 38과는 다른 형태의 데이터셋

```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



머신러닝 알고리즘을 위한 데이터 준비

- 머신러닝 알고리즘을 위한 데이터를 만드는 함수를 정의

- 함수로 정의해야 하는 이유
  - 어떤 데이터셋에 대해서도 데이터 변환을 손쉽게 반복할 수 있음
  - 향후 프로젝트에 사용할 수 있는 변환 라이브러리를 점진적으로 구축하게 됨
  - 실제 시스템에서 알고리즘에 새 데이터를 주입하기 전에 변환시키는 데 이 함수를 사용할 수 있음
  - 여러 가지 데이터 변환을 쉽게 시도해볼 수 있고 어떤 조합이 가장 좋은지 확인하는 데 편리함

내용출처: 핸즈온 머신러닝

머신러닝 알고리즘을 위한 데이터 준비

- 훈련 세트로 복원하고(strat_train_set을 다시한번 복사)
- 예측 변수와 타깃 값에 같은 변형을 적용하지 않기 위해 예측 변수와 레이블을 분리
  - drop() 함수는 데이터 복사본을 만들며 start_train_set에는 영향을 주지 않음

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

내용출처: 핸즈온 머신러닝

누락된 데이터 처리

- 머신러닝 알고리즘에 사용되는 데이터는 누락된 값을 처리하고 사용해야 하기때문에, DataFrame의 dropna(), drop(), fillna() 메서드를 이용해서 간단하게 처리.

- dropna() 함수 사용

```
housing_copy.dropna(subset=["total_bedrooms"])    # option 1
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
18086	-122.05	37.31	25.0	4111.0	538.0	1585.0	568.0	9.2298
16718	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285
13600	-117.25	34.16	37.0	1709.0	278.0	744.0	274.0	3.7188

내용출처: 핸즈온 머신러닝

누락된 데이터 처리

- drop() 함수사용

```
housing_copy = housing.copy().iloc[21:24]
housing_copy.drop("total_bedrooms", axis=1)       # option 2
```

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	ocean_proximity
18086	-122.05	37.31	25.0	4111.0	1585.0	568.0	9.2298	<1H OCEAN
16718	-120.66	35.49	17.0	4422.0	2307.0	885.0	2.8285	<1H OCEAN
13600	-117.25	34.16	37.0	1709.0	744.0	274.0	3.7188	INLAND

- fillna() 함수 사용

```
housing_copy = housing.copy().iloc[21:24]
median = housing_copy["total_bedrooms"].median()
housing_copy["total_bedrooms"].fillna(median, inplace=True) # option 3
housing_copy
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
18086	-122.05	37.31	25.0	4111.0	538.0	1585.0	568.0	9.2298
16718	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285
13600	-117.25	34.16	37.0	1709.0	278.0	744.0	274.0	3.7188

내용출처: 핸즈온 머신러닝

누락된 데이터 처리

- 사이킷런의 Imputer는 누락된 값을 손쉽게 다루도록 해줌
- 누락된 값을 특성의 중간값으로 대체한다고 지정하여 Imputer의 객체를 생성
- 텍스트 특성인 ocean_proximity를 제외한 모든 수치형 특성에 대해 imputer를 적용함
- imputer 객체의 fit() 메서드를 사용해 훈련 데이터에 적용

```
from sklearn.preprocessing import Imputer

imputer = Imputer(strategy='median')
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
housing_tr.iloc[21:24]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
21	-122.05	37.31	25.0	4111.0	538.0	1585.0	568.0	9.2298
22	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285
23	-117.25	34.16	37.0	1709.0	278.0	744.0	274.0	3.7188

내용출처: 핸즈온 머신러닝

누락된 데이터 처리

- 사이킷런의 Imputer는 누락된 값을 손쉽게 다루도록 해줌
- 누락된 값을 특성의 중간값으로 대체한다고 지정하여 Imputer의 객체를 생성
- 텍스트 특성인 ocean_proximity를 제외한 모든 수치형 특성에 대해 imputer를 적용함
- imputer 객체의 fit() 메서드를 사용해 훈련 데이터에 적용

```
from sklearn.preprocessing import Imputer

imputer = Imputer(strategy='median')
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
housing_tr.iloc[21:24]
```

imputer의 transform 메서드의 반환결과는 numpy 이기 때문에 이를 pandas 의 DataFrame 형태로 되돌리기

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
21	-122.05	37.31	25.0	4111.0	538.0	1585.0	568.0	9.2298
22	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285
23	-117.25	34.16	37.0	1709.0	278.0	744.0	274.0	3.7188

내용출처: 핸즈온 머신러닝

누락된 데이터 처리

- 사이킷런의 Imputer는 누락된 값을 손쉽게 다루도록 해줌
- 누락된 값을 특성의 중간값으로 대체한다고 지정하여 Imputer의 객체를 생성
- 텍스트 특성인 ocean_proximity를 제외한 모든 수치형 특성에 대해 imputer를 적용함
- imputer 객체의 fit() 메서드를 사용해 훈련 데이터에 적용

```
from sklearn.preprocessing import Imputer

imputer = Imputer(strategy='median')
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
housing_tr.iloc[21:24]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
21	-122.05	37.31	25.0	4111.0	538.0	1585.0	568.0	9.2298
22	-120.66	35.49	17.0	4422.0	945.0	2307.0	885.0	2.8285
23	-117.25	34.16	37.0	1709.0	278.0	744.0	274.0	3.7188

내용출처: 핸즈온 머신러닝

텍스트와 범주형 특성 다루기

- 대부분의 머신러닝 알고리즘은 숫자형 데이터를 다루므로 이 카테고리를 텍스트에서 숫자로 바꾸도록 하겠습니다.
- 사이킷런에서 LabelEncoder함수를 사용하여 텍스트형 데이터를 숫자형으로 바꿔보겠습니다.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
housing_cat = housing["ocean_proximity"]
housing_cat_encoded = encoder.fit_transform(housing_cat)
housing_cat_encoded
```

```
array([0, 0, 4, ..., 1, 0, 3])
```

```
print(encoder.classes_)
```

```
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

내용출처: 핸즈온 머신러닝

원-핫 인코딩

- 이 표현 방식의 문제는 머신러닝 알고리즘이 가까이 있는 두 값이 떨어져 있는 두 값보다 더 비슷하다고 생각한다는 점
- 예를 들어서 카테고리 0과 1보다 카테고리 0과 4가 더 비슷
- 이러한 문제는 일반적으로 카테고리별 이진 특성을 만들어서 해결
- 카테고리가 <1H OCEAN 일때는 한 특성이 1이고 그외 특성은 0, 카테고리가 INLAND 일 때는 다른 한 특성이 1이되고 그 외 특성은 0으로
- 한 특성만 1이고(핫) 나머지는 0이므로 이를 원-핫 인코딩(one-hot encoding)이라고 부름

내용출처: 핸즈온 머신러닝

원-핫 인코딩

- 사이킷런은 숫자로 된 범주형 값을 원-핫 벡터로 바꿔주는 OneHotEncoder를 제공함

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot
```

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

```
housing_cat_1hot.toarray()
```

```
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.],
       ...,
       [ 0.,  1.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.]])
```

내용출처: 핸즈온 머신러닝

특성 스케일링

- 데이터에 적용할 가장 중요한 변환 중 하나가 특성 스케일링(feature scaling) 입니다.

- 모든 특성의 범위를 같도록 만들어주는 방법으로 min-max 스케일링(정규화라고도 불림) 과 표준화(standardization)가 널리 사용됩니다.

- min-max 스케일링은 0~1 범위에 들도록 값을 이동하고 스케일을 조정하면 됨.

- 사이킷런에는 이에 해당하는 MinMaxScaler 함수를 제공
  - 0~1 사이를 원하지 않는다면 feature_range 매개변수로 범위를 변경할 수 있음

내용출처: 핸즈온 머신러닝

특성 스케일링

- 표준화는 데이터에서 먼저 평균을 뺀 후 표준편차로 나누어 결과 분포의 분산이 1이 되도록 함.

- min-max 스케일링과는 달리 표준화는 범위의 상한과 하한이 없어 어떤 알고리즘에서는 문제가 될 수 있음.(신경망에서는 종종 입력값의 범위를 0~1사이로 기대함)

- 그러나 표준화는 이상치에 영향을 덜 받음

- 사이킷런에서는 표준화를 위한 StandardScaler 함수를 제공

내용출처: 핸즈온 머신러닝

변환 파이프라인

- 앞서 보았듯이 변환 단계가 많으며 정확한 순서대로 실행되어야 합니다.

- 다행히 사이킷런에는 연속된 변환을 순서대로 처리할 수 있도록 도와주는 Pipeline 클래스가 있습니다.

내용출처: 핸즈온 머신러닝

변환 파이프라인

수치 전처리를 위한 파이프라인 만들기

```
1  from  sklearn.pipeline import Pipeline
2  from sklearn.preprocessing import StandardScaler
3
4  num_pipeline = Pipeline([
5          ('imputer', Imputer(strategy="median")),
6          ('attribs_adder', CombinedAttributesAdder()),
7          ('std_scaler', StandardScaler()),
8      ])
9
10 housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
1  housing_num_tr
```

```
array([[-1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895, ..., -0.22852947,
        -0.09586294,  0.10180567]])
```

ColumnTransformer를 이용한 방법

```
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

```
housing_prepared
```

```
array([[-1.15604281,  0.77194962,  0.74333089, ...,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, ...,  0.        ,
         0.        ,  1.        ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, ...,  0.        ,
         1.        ,  0.        ]])
```

```
housing_prepared.shape
```

```
(16512, 16)
```

모델 선택과 훈련

- 선형회귀 모델

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
# let's try the full pipeline on a few training instances
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = preparation_pipeline.transform(some_data)

print("Predictions:₩t", lin_reg.predict(some_data_prepared))
print("Labels:₩t₩t", list(some_labels))
```

```
Predictions:    [ 210644.60459286  317768.80697211  210956.43331178   59218.98886849
  189747.55849879]
Labels:         [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

내용출처: 핸즈온 머신러닝

모델 선택과 훈련

- RMSE측정

```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
68628.198198489219
```

- 대부분 구역의 중간 주택 가격은 $120,000 ~ $ 265,000사이인데, 예측 오차가 $68,628인것은 매우 만족스럽지 못한 결과

- 이는 모델이 훈련 데이터에 과소적합된 사례

내용출처: 핸즈온 머신러닝

모델 선택과 훈련

- 결정트리 + 선형회귀 모델

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
0.0
```

- rmse 가 0? 이것은 모델이 데이터에 너무 심하게 과대적합된 것

- 모델이 론칭할 준비가 되기 전까지 테스트 세트는 사용하지 않으려 하므로 훈련 세트의 일부분으로 훈련을 하고 다른 일부분은 모델 검증에 사용해야 합니다. >> 교차검증

내용출처: 핸즈온 머신러닝

교차검증

- 사이킷런의 교차검증 함수를 사용
- 훈련 세트를 폴드(fold)라 불리는 10개의 서브셋으로 무작위로 분할
- 그런 다음 결정트리 모델을 10번 훈련하고 평가하는데, 매번 다른 폴드를 선택해 평가에 사용하고 나머지 9개 폴드는 훈련에 사용
- 10개의 평가점수가 담긴 배열이 결과가 됨

내용출처: 핸즈온 머신러닝

모델 선택과 훈련

```
from sklearn.model_selection import cross_val_score

tree_scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                              scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-tree_scores)
```

```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [ 69316.02634772  65498.84994772  71404.25935862  69098.46240168
  70580.30735263  75540.88413124  69717.93143674  70428.42648461
  75888.17618283  68976.12268448]
Mean: 70644.9446328
Standard deviation: 2938.93789263
```

- 결정트리가 이전보다는 좋아보이지 않음 ㅠㅠ
- 교차검증으로 모델의 성능을 추정하는것 뿐만 아니라 이 추정이 얼마나 정확한지(즉 표준편차)를 측정할 수 있음.
- 결정트리 점수가 대략 71.379 +- 2,458 사이

내용출처: 핸즈온 머신러닝

모델 선택과 훈련

- 선형회귀 모델의 점수 계산

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [ 66760.97371572  66962.61914244  70349.94853401  74757.02629506
  68031.13388938  71193.84183426  64968.13706527  68261.95557897
  71527.64217874  67665.10082067]
Mean: 69047.8379055
Standard deviation: 2735.51074287
```

내용출처: 핸즈온 머신러닝

모델 선택과 훈련

- 랜덤 포레스트 회귀모델
  - 랜덤 포레스트와 선형회귀 모델을 모아서 하나의 모델을 만듬
  - = 앙상블 학습이라고 하며, 머신러닝 알고리즘의 성능을 극대화하는 방법 중 하나

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
22252.738943108321
```

내용출처: 핸즈온 머신러닝

모델 선택과 훈련

- 랜덤 포레스트 회귀의 점수 계산

```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [ 52869.23106834  49189.93801195  51726.73647871  54995.98190463
  50979.93079904  55978.43765914  52283.7609046   51001.92227546
  54447.35786983  53389.94422283]
Mean: 52686.3241195
Standard deviation: 1971.26547795
```

```
scores = cross_val_score(lin_reg, housing_prepared, housing_labels, scoring="neg_mean_squared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

```
count       10.000000
mean     69047.837905
std       2883.481504
min      64968.137065
25%      67138.239562
50%      68146.544734
75%      70982.868509
max      74757.026295
dtype: float64
```

내용출처: 핸즈온 머신러닝

모델 세부튜닝

- 그리드 탐색
  - 만족할 만한 하이퍼파라미터 조합을 찾을 때까지 수동으로 하이퍼파라미터를 조정하는 것
  - 하지만 이는 매우 지루한 작업이고 많은 경우의 수를 탐색하기에는 시간이 부족
  - 대신 **사이킷런의 GridSearchCV를 사용**하면 좋음
  - 탐색하고자 하는 하이퍼파라미터와 시도해볼 값을 지정하기만 하면 됨

- 다음 코드는 RandomForestRegressor에 대한 최적의 하이퍼파라미터 좋바을 탐색함

내용출처: 핸즈온 머신러닝

모델 세부튜닝

- param_grid 설정에 따라 사이킷런이 첫번째 dict()에 있는 n_estimators와 max_features 하이퍼파라미터의 조합인 3x4 = 12개를 평가
- 그 다음 두번째 dict()에 있는 하이퍼파라미터의 조합인 2x3=6 개를 시도
- 하지만 두번째는 bootstrap 하이퍼파라미터를 True가아니라 False로 설정
- 모든 파라미터를 합하면 18개 조합을 탐색하고, 각각 모델을 5번 훈련시킴(5번의 교차검증을 사용하기 때문, cv=5)
  - 다시 말해서 전체 훈련 횟수는 18x5=90이 됨

내용출처: 핸즈온 머신러닝

모델 세부튜닝(꽤 시간걸림.....)

```
from sklearn.model_selection import GridSearchCV

param_grid = [
        {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
        {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
    ]

forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(housing_prepared, housing_labels)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid=[{'max_features': [2, 4, 6, 8], 'n_estimators': [3, 10, 30]}, {'bootstrap': [False], 'max_feature
s': [2, 3, 4], 'n_estimators': [3, 10]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='neg_mean_squared_error', verbose=0)
```

내용출처: 핸즈온 머신러닝

모델 세부 튜닝

- 훈련 시간은 꽤 오래 걸리지만, 다음과 같이 최적의 조합 , 최적의 추정기를 얻을수 있음

```
grid_search.best_params_
```

```
{'max_features': 6, 'n_estimators': 30}
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=6, max_leaf_nodes=None, min_impurity_split=1e-07,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

내용출처: 핸즈온 머신러닝

모델 세부 튜닝

- 각 하이퍼파라미터 별 평가 점수도 확인 가능

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
64912.0351358 {'max_features': 2, 'n_estimators': 3}
55535.2786524 {'max_features': 2, 'n_estimators': 10}
52940.2696165 {'max_features': 2, 'n_estimators': 30}
60384.0908354 {'max_features': 4, 'n_estimators': 3}
52709.9199934 {'max_features': 4, 'n_estimators': 10}
50503.5985321 {'max_features': 4, 'n_estimators': 30}
59058.1153485 {'max_features': 6, 'n_estimators': 3}
52172.0292957 {'max_features': 6, 'n_estimators': 10}
49958.9555932 {'max_features': 6, 'n_estimators': 30}
59122.260006 {'max_features': 8, 'n_estimators': 3}
52441.5896087 {'max_features': 8, 'n_estimators': 10}
50041.4899416 {'max_features': 8, 'n_estimators': 30}
62371.1221202 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54572.2557534 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59634.0533132 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52456.0883904 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58825.665239 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
52012.9945396 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

내용출처: 핸즈온 머신러닝

모델 세부 튜닝

- 랜덤 탐색
  - 그리드 탐색 방법은 이전 예제와 같이 비교적 적은 수의 조합을 탐구할 때 괜찮습니다.
  - 하지만 하이퍼파라미터 탐색 공간이 커지면 RandomizedSearchCV를 사용하는 편이 더 좋습니다.
  - RandomizedSearchCV는 GridSearchCV와 거의 같은 방식으로 사용하지만 가능한 모든 조합을 시도하는 대신 각 반복마다 하이퍼파라미터에 임의의 수를 대입하여 지정한 횟수만큼 평가합니다.
    - 이 방법의 장점은 랜덤 탐색을 1,000회 반복하도록 실행하면 하이퍼파라미터마다 각기 다른 1,000개의 값을 탐색합니다.
    - 단순히 반복 횟수를 조절하는 것만으로 하이퍼파라미터 탐색에 투입할 컴퓨팅 자원을 제어할 수 있습니다.

내용출처: 핸즈온 머신러닝

모델 세부 튜닝(이건 시간이 더걸림....ㅠ)


```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor()
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error')
rnd_search.fit(housing_prepared, housing_labels)
```

```
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False),
          fit_params={}, iid=True, n_iter=10, n_jobs=1,
          param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fd4fe486470
>, 'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fd4fe486cc0>},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='neg_mean_squared_error',
          verbose=0)
```

내용출처: 핸즈온 머신러닝

모델 세부 튜닝

```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
50239.6442738 {'max_features': 3, 'n_estimators': 121}
50307.8432326 {'max_features': 3, 'n_estimators': 187}
49185.0150532 {'max_features': 6, 'n_estimators': 88}
49133.3305418 {'max_features': 5, 'n_estimators': 137}
49021.6318804 {'max_features': 7, 'n_estimators': 197}
49636.8878839 {'max_features': 6, 'n_estimators': 39}
52273.457854 {'max_features': 2, 'n_estimators': 50}
54413.8506712 {'max_features': 1, 'n_estimators': 184}
51953.3364641 {'max_features': 2, 'n_estimators': 71}
49174.1414792 {'max_features': 6, 'n_estimators': 140}
```

내용출처: 핸즈온 머신러닝