

# 2장



머신러닝 프로젝트  
처음부터 끝까지

# 프로젝트 진행 과정

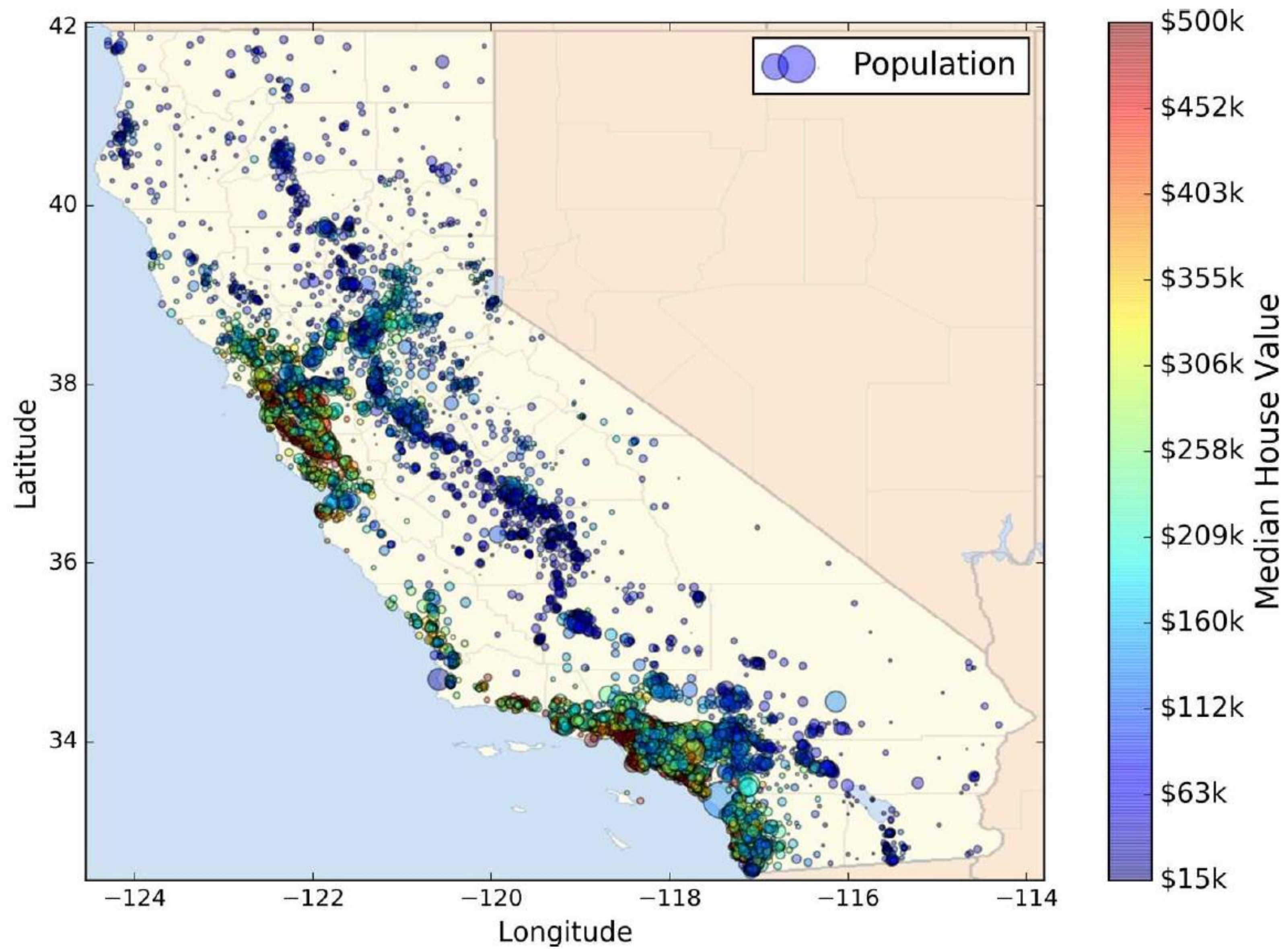
1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

# 공개 데이터 저장소

- 유명한 공개 데이터 저장소
  - UC 얼바인Irvine 머신러닝 저장소(<http://archive.ics.uci.edu/ml/>)
  - 캐글Kaggle 데이터셋(<http://www.kaggle.com/datasets>)
  - 아마존 AWS 데이터셋(<http://aws.amazon.com/ko/datasets>)
- 메타 포털(공개 데이터 저장소가 나열되어 있습니다)
  - <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 서브레딧subreddit(<http://www.reddit.com/r/datasets>)

# 캘리포니아 주택 가격 예측 데이터

- 1990년 캘리포니아 인구조사 데이터를 기반으로 작성
- 카네기 멜론Carnegie Mellon 대학교의 통계학과에서 운영 하는 StatLib 저장소(<http://lib.stat.cmu.edu/datasets/>)의 데이터를 수정한 버전을 사용함 ([http://www.dcc.fc.up.pt/~ltorgo/Regression/cal\\_housing.html](http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html))
- 블록(600~3,000명 단위)마다 인구, 중간 소득, **중간 주택 가격** 등
- ➔ 이 데이터로 모델을 학습시킴.  
➔ 측정데이터(test data)가 주어졌을 때 그 구역의 중간 주택 가격을 예측하려고 함.





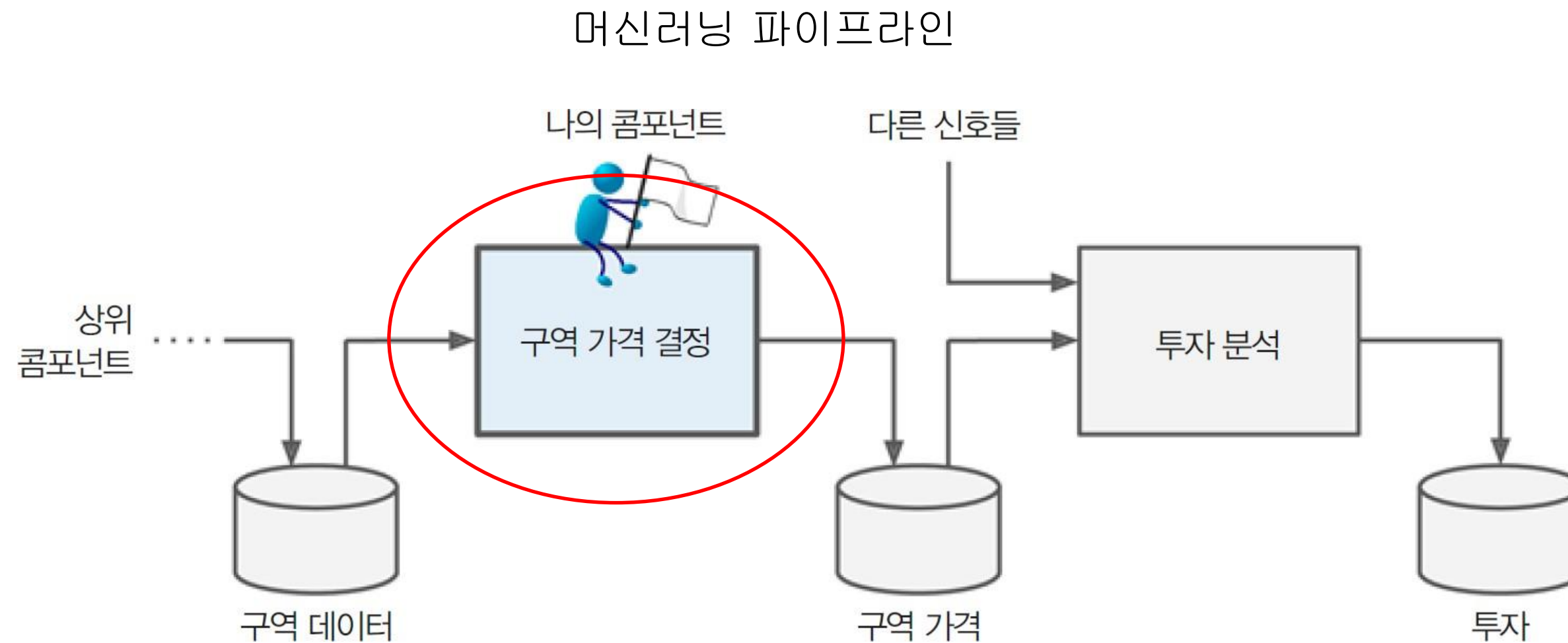
# 큰 그림 보기 Looking at the Big Picture

California census data를 이용해서 California 집 값 예측하는 프로그램 작성하자.  
전 과정을 단계별로 접근하려고 함  
부록 B “머신러닝프로젝트 체크리스트”에 자세한 항목 있음

- 문제파악 (Frame the Problem)
- 성능측정지표 선택 (Select a Performance Measure)
- 가정검사 (Check the Assumptions)

# 큰 그림 보기: 문제 파악 - 목적은?

- 목적이 무엇인지 파악 : 시스템의 구성, 알고리즘, 측정 지표, 튜닝 시간 등을 결정하기 때문에 중요함
- 주어진 문제는 “주택가격을 예측하는 머신러닝 프로그램을 작성하라. 이 시스템은 투자를 할지를 결정하는 머신러닝 시스템의 입력 중 하나로 들어갈 것이다” 이다.



# 큰 그림 보기: 문제 파악 – 기존(현재) 솔루션은?

- 현재 솔루션이 있다면 현재 해결 방법에 대한 정보를 얻을 수 있고 참고 성능으로도 사용할 수 있음.
- “현재는 전문가가 수동으로 중간 주택 가격을 추정합니다. 구역에 관한 최신 정보를 모으고 복잡한 규칙을 사용함.”
- “수동으로 추정하고 있고 오차는 약 15% 임”  
➔ 추정 오차를 줄이기 위해 머신러닝을 사용하려고 함.



# 큰 그림 보기 : 문제 파악 - 시스템 설계 방향

지도/비지도/강화학습 ?,

분류/회귀 ?,

배치/온라인 ?

- 데이터 형태 : 레이블된 샘플이 있으므로 지도 학습.
- 모델 : 연속된 값을 예측하므로 회귀 문제.  
여러 특징을 사용하므로 다변량 회귀 multivariate regression임  
(반대는 단변량 회귀 univariate regression).
- 학습방법 : 데이터는 오프라인으로 준비되어 있고 크지 않으므로 배치 학습이 적당

# 큰 그림 보기: 성능 측정 지표 선택

- 회기분석은 일반적으로 평균 제곱근 오차 사용  
RMSE(Root Mean Square Error)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

가설(hypothesis)

샘플 개수

특성

타겟(레이블)

- $R^2$  (scikit-learn 기본값)

$$R^2 = 1 - \frac{\sum_{i=1}^m (y^{(i)} - h(\mathbf{x}^{(i)}))^2}{\sum_{i=1}^m (y^{(i)} - \bar{y})^2} = 1 - \frac{\text{Residual sum of squares}}{\text{Total sum of squares}} = \frac{\text{Explained sum of squares}}{\text{Total sum of squares}} = \text{Pearson } r^2$$

선형 회귀 경우

참고 : MAE,  $l_k$  norm

- 평균 절대 오차 Mean Absolute Error(MAE)

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

- $l_k$  norm  $\|\mathbf{v}\|_k = (|v_0|^k + |v_1|^k + \dots + |v_n|^k)^{\frac{1}{k}}$

유클리디안 노름 =  $l_2$  노름 =  $\|\mathbf{v}\|_2 = \|\mathbf{v}\| = \sqrt{m} \times \mathbf{RMSE}$

맨하탄 노름 =  $l_1$  노름 =  $\|\mathbf{v}\|_1 = m \times \mathbf{MAE}$

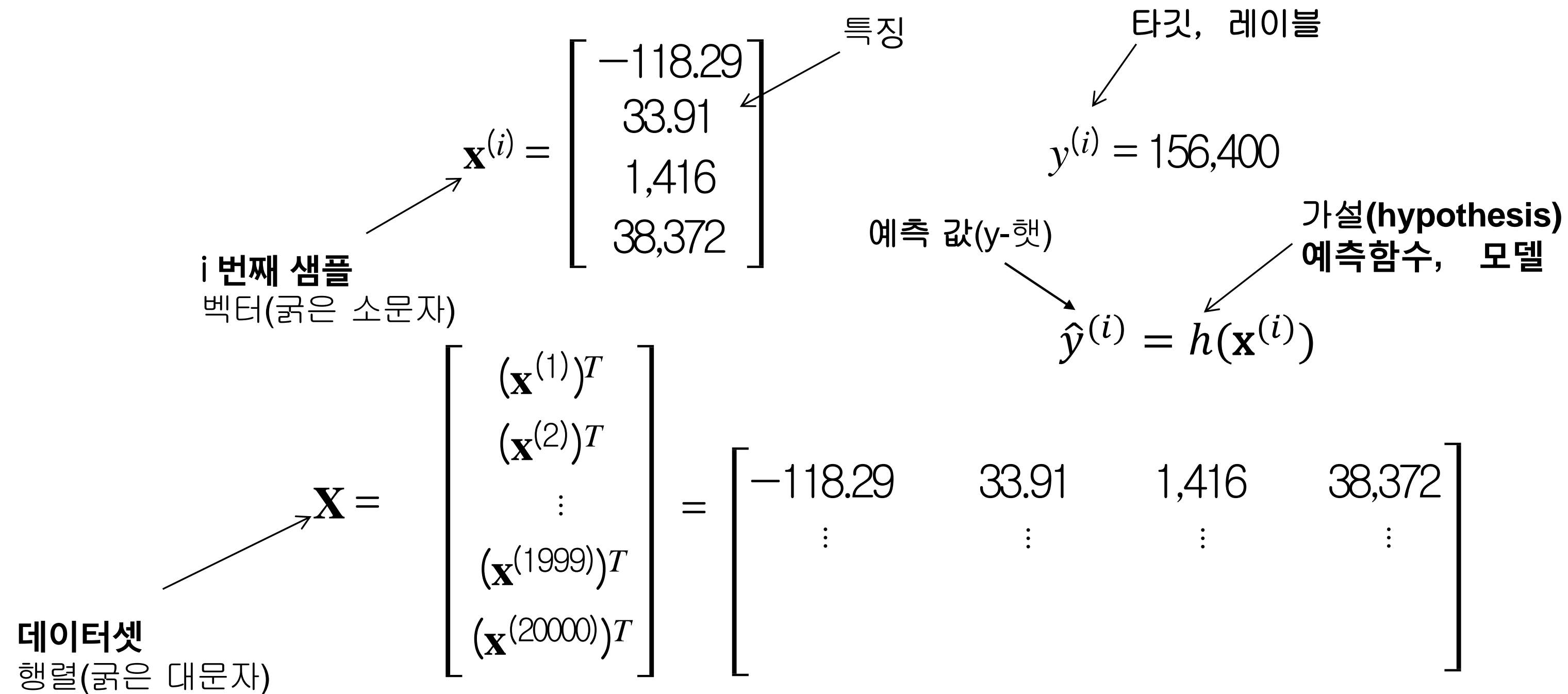
k가 클수록 큰 원소에 치우칩니다 : RMSE가 MAE 보다 이상치에 더 민감

k가 무한대이면 가장 큰 원소의 절대값

k가 0이면 벡터에서 0이 아닌 원소의 수입니다

# 참고 : 표기법

- $i$  번째 샘플의 경도가 -118.29, 위도 33.91, 주민수 1,416, 중간 소득 \$38,372  
중간 주택 가격이 \$156,400이라면,



# 큰 그림 보기: 가정 검사 Check the Assumptions

- 지금까지 세운 가정을 나열하고 검증
- 혹시 하위 시스템이 중간 주택 가격이 아니고 등급(분류 문제)을 원하지 않나요?
  - ➔ 정확한 주택 값을 예측하는 대신 (저렴, 보통, 고가)같이 분류하고자 하면?
  - ➔ 분류 문제가 됨
  - ➔ 레이블(타겟 데이터)을 다시 만들어야 함
- 너무 늦게 문제를 발견하지 않도록 주의하세요.

# 데이터 가져오기

## Get the Data

Jupyter Notebook 프로그램은

<https://github.com/ageron/handson-ml>

<https://github.com/rickiepark/handson-ml>

원 저자 코드

번역자 코드. 한글+일부 추가설명

- 데이터 다운로드
- 데이터 구조 보기 (Take a Quick Look at the Data Structure)
- 테스트세트 만들기 (Create a Test Set)



# 데이터 가져오기: 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()
```

datasets/housing 폴더 만들고  
housing.tgz 파일을 다운받고  
폴더에 압축풀음 (housing.csv만들어 짐)

```
fetch_housing_data()
```

```
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.csv 읽어 들임  
➔ pandas 객체(dataframe)가 반환됨



# 데이터 가져오기: 데이터셋 내용보기: head()

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

10개의 특성

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

housing.csv

```
longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,median_house_value,ocean_proximity  
-122.23,37.88,41.0,880.0,129.0,322.0,126.0,8.3252,452600.0,NEAR BAY  
-122.22,37.86,21.0,7099.0,1106.0,2401.0,1138.0,8.3014,358500.0,NEAR BAY  
-122.24,37.85,52.0,1467.0,190.0,496.0,177.0,7.2574,352100.0,NEAR BAY  
-122.25,37.85,52.0,1274.0,235.0,558.0,219.0,5.6431,341300.0,NEAR BAY  
-122.25,37.85,52.0,1627.0,280.0,565.0,259.0,3.8462,342200.0,NEAR BAY
```

# df.info()

```
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
```

20,640개 샘플

← 207개가 비어 있음

← 문자열 특성(범주형)

# df.value\_counts()

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

```
<1H OCEAN      9136
```

```
INLAND          6551
```

```
NEAR OCEAN      2658
```

```
NEAR BAY        2290
```

```
ISLAND           5
```

```
Name: ocean_proximity, dtype: int64
```

# df.describe()

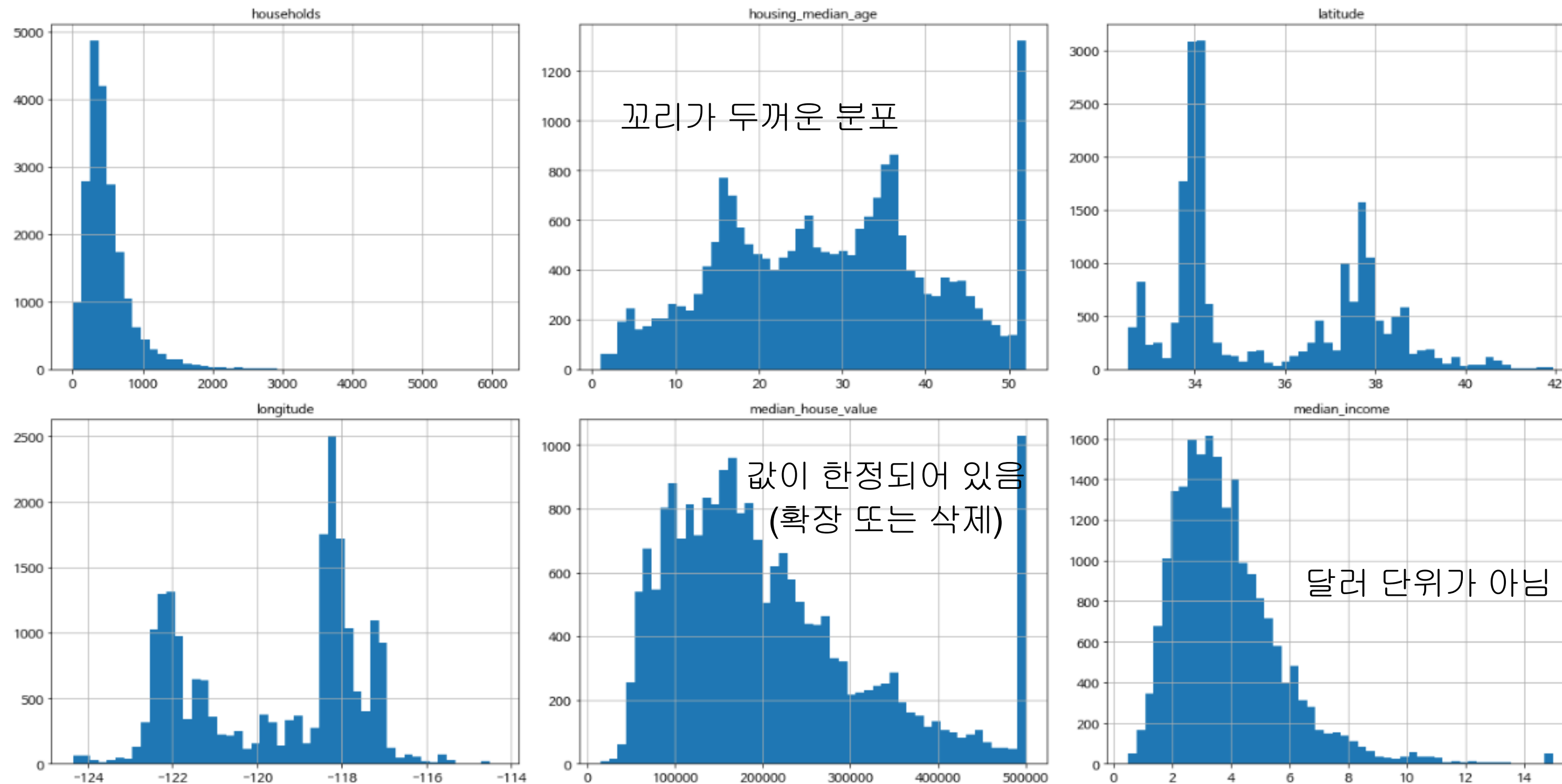
housing.describe()

← 숫자형 특성을 요약

		longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
	count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000
	mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.859617
	std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.890454
	min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900
1사분위	25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.595168
	50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.596274
3사분위	75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.741437
	max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000000

# df.hist()

`housing.hist(bins=50, figsize=(20,15))` ← 숫자형 특성의 히스토그램을 그림





# 데이터 가져오기 : Test set만들기 : numpy로 분리

- 데이터를 더 자세히 파악하기 전에 테스트 데이터를 떼어 놓아야 함
- 전체 데이터에서 너무 많은 직관을 얻으면 과대적합된 모델이 만들어짐(데이터 스누핑 data snooping 편향)

```
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]
```

단순한 방법


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

16512 train + 4128 test



# 데이터 가져오기: 앞 코드의 문제점/해결책 1

- 문제점 : 함수를 실행할 때마다 다른 테스트 세트가 만들어 집니다(데이터 스누핑 우려)
- 해결책 1 : 테스트 세트를 따로 떼어 저장하거나 고정된 seed 사용  
(예 np.random.seed(42)) → 데이터셋이 바뀌면 적용할 수 없음



```
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]
```

여기에 삽입

# 데이터 가져오기: 앞 코드의 해결책 2

- 샘플의 식별자를 해싱한 값을 기준으로 분리

```
from zlib import crc32

def test_set_check(identifier, test_ratio):    for Python 2
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]

housing_with_id = housing.reset_index()    # `index` 열이 추가된 데이터프레임이 반환됩니다.
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

우리 문제는 식별자가 없어  
"index"를 만들었음

- 이 방법은 새로운 데이터가 추가될 때 기존 데이터 뒤에 추가 되어야 함.  
➔ 실제로는 scikit-learn의 train\_test\_split() 이용하는 것이 좋음



# 데이터 가져오기: Test set만들기 : scikit-learn의 test\_train\_split()이용

이것을 사용하기 바람

다양한 배열을 넣을 수 있음  
(파이썬 리스트, 넘파이, 판다스 데이터프레임)

```
from sklearn.model_selection import train_test_split

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

train\_size 지정할 수 있음

```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	INLAND
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	INLAND
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	NEAR BAY
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	<1H OCEAN
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR OCEAN

# 고려해야 할 사항 : 샘플링 편향

- 여성이 51.3%, 남성 48.7%일 때 전체인구를 대표하는 1000명을 샘플링할 때
  - 무작위로 선택하면 편향될 수 있다 :

여성이 49%보다 적거나 54% 선택될 확률이 약 12%가 됨

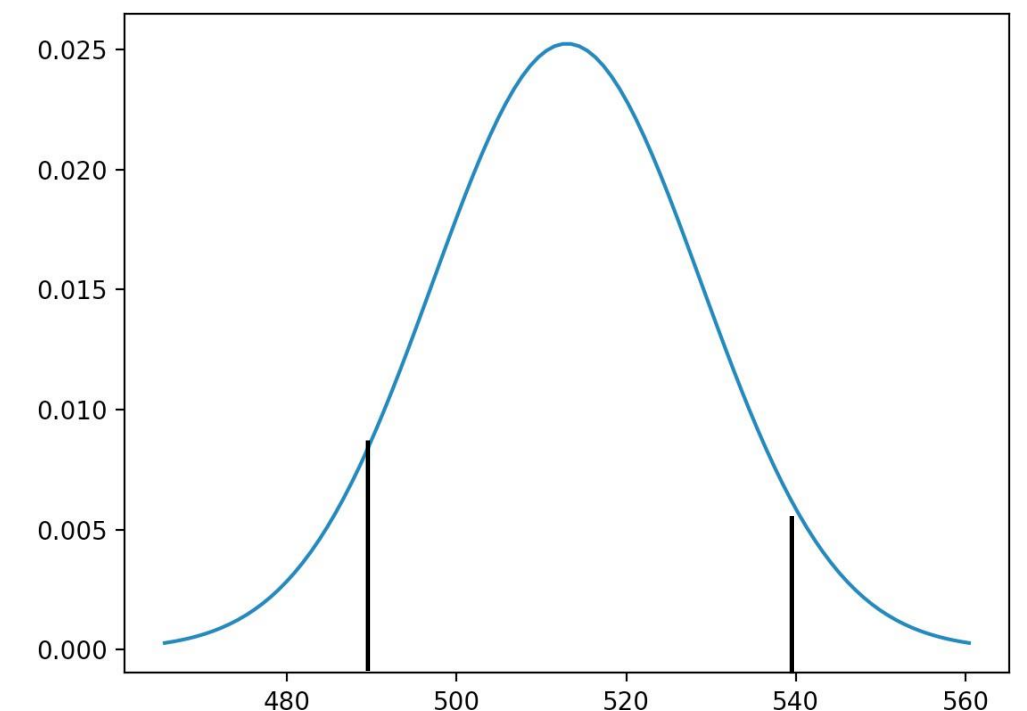
샘플 :  $n$ , 비율 :  $p$ 일 때

$n \times p \geq 10, n \times (1 - p) \geq 10$ 인 이항 분포는

$\mu = np = 513, \sigma = \sqrt{np(1-p)} = 15.8$ 인 정규분포로 근사됨

- 전체에서 여자 513명, 남자 487명을 샘플링 해야 함 : 계층적 샘플링

```
>>> from scipy.stats import norm
>>> norm.cdf(490, 513, 15.8) + (1 - norm.cdf(540, 513, 15.8))
0.11647668242572096
```



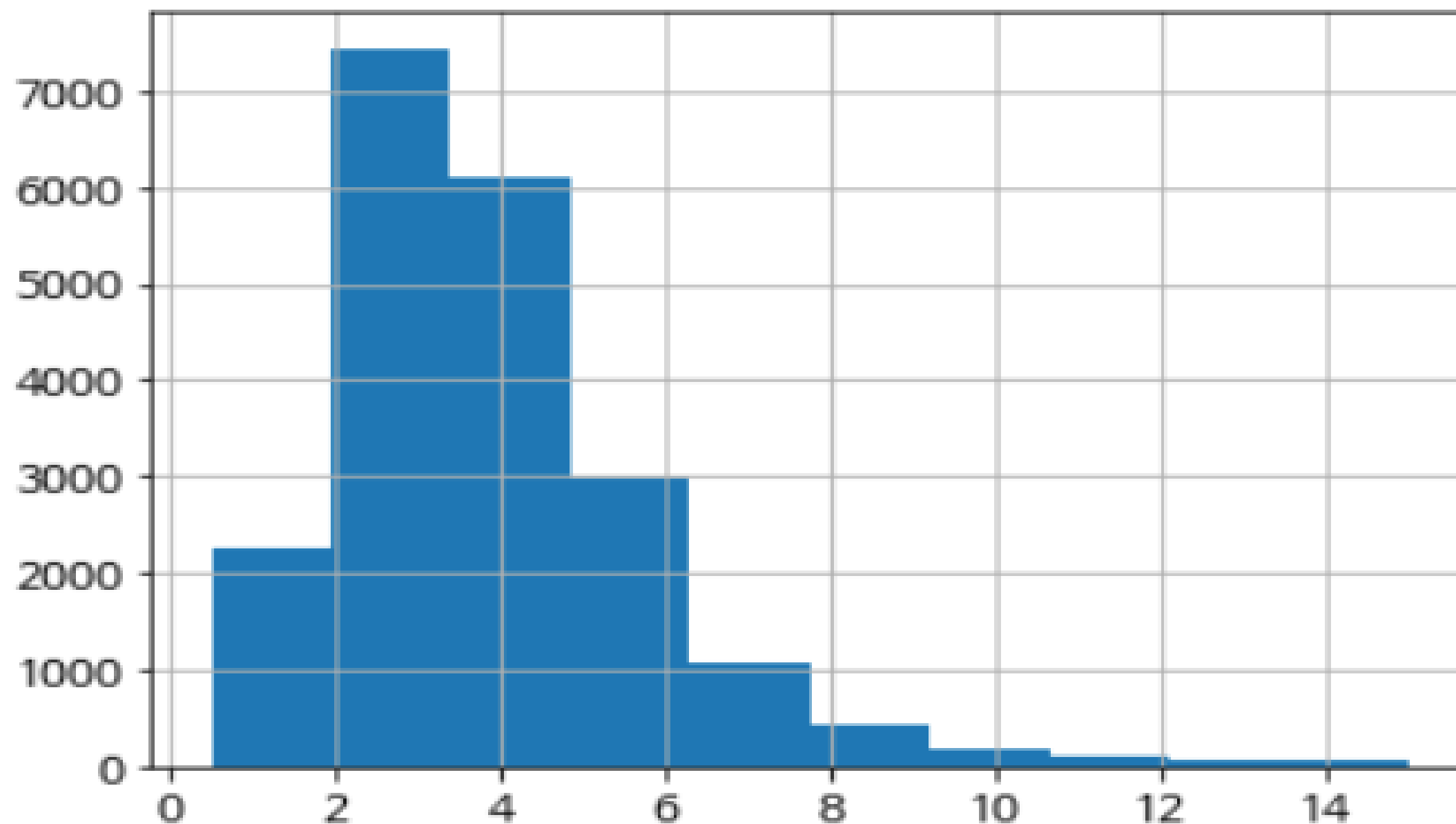
# 데이터 가져오기: 계층적 샘플링

소득(median\_income)이 중간 주택 가격을 예측하는데 중요하다고 가정함 (전문가 지식에서...)

➔ 소득 구간별로 계층적 샘플링해서 특정 구간에 샘플이 많게/적게 몰리지 않게 하자.

```
housing["median_income"].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a275f19b0>
```





# 데이터 가져오기: 계층적 샘플링2 : 소득구간 나누기

소득(median\_income)을 5개 구간으로 나누어 income\_cat에 저장하자.

```
# 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
```

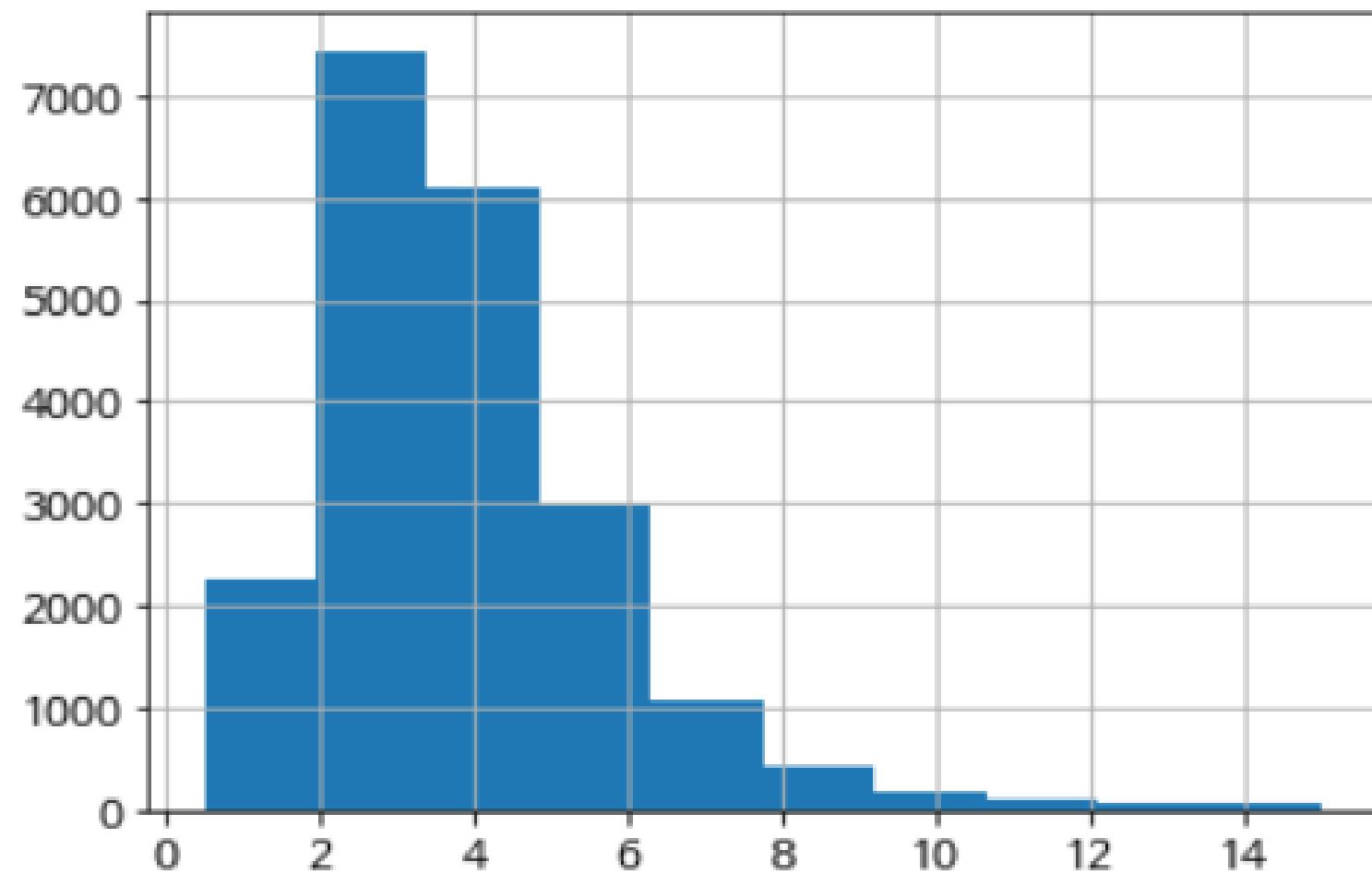
```
Name: income_cat, dtype: int64
```



# 데이터 가져오기: 계층적 샘플링3 : 히스토그램

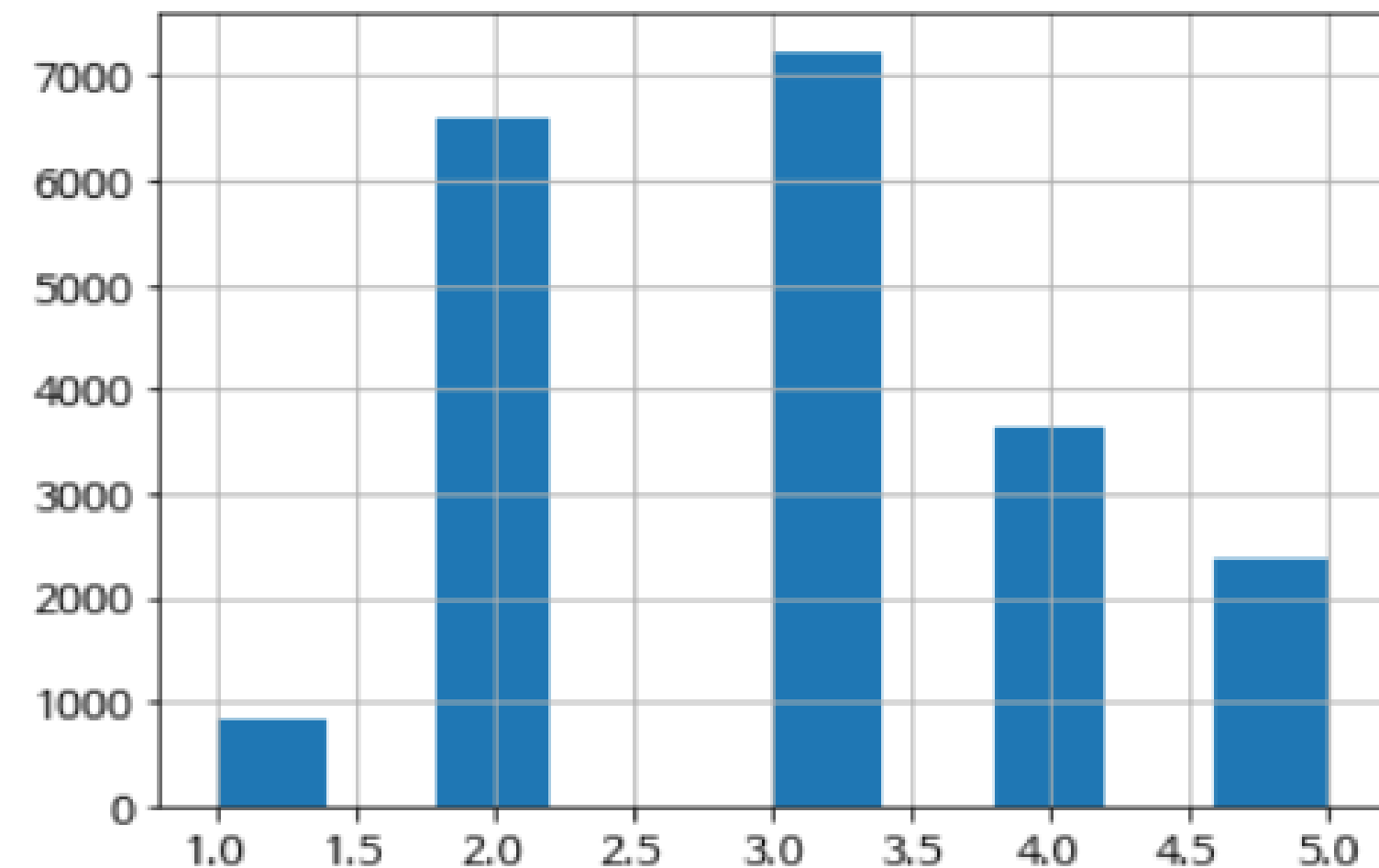
```
housing["median_income"].hist()
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x1a275f19b0>



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

<matplotlib.axes.\_subplots.AxesSubplot at 0xa25e3be48>



# 데이터 가져오기: 계층적 샘플링 4: StratifiedShuffleSplit

StratifiedKFold + ShuffleSplit (test\_size와 train\_size 매개변수 합을 1이하로 지정할 수 있음)

```
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]
```

```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

```
3.0    0.350533
2.0    0.318798
4.0    0.176357
5.0    0.114583
1.0    0.039729
Name: income_cat, dtype: float64
```

```
strat_train_set["income_cat"].value_counts() / len(strat_train_set)
```

```
3.0    0.350594
2.0    0.318859
4.0    0.176296
5.0    0.114402
1.0    0.039850
Name: income_cat, dtype: float64
```

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

```
3.0    0.350581
2.0    0.318847
4.0    0.176308
5.0    0.114438
1.0    0.039826
Name: income_cat, dtype: float64
```

# 데이터 가져오기: 계층적 샘플링 5: train\_test\_split()

이것이 사용하기 쉬움

```
strat_train_set, strat_test_set = train_test_split(housing, test_size=0.2, random_state=42,  
                                                  stratify=housing["income_cat"])
```

```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

3.0      0.350533

2.0      0.318798

4.0      0.176357

5.0      0.114583

1.0      0.039729

Name: income\_cat, dtype: float64

StratifiedShuffleSplit 사용  
(아니면 ShuffleSplit 사용)



# 샘플링 편향 비교

	전체	무작위 샘플링	계층 샘플링	무작위 샘플링 오류율	계층 샘플링 오류율
<b>1.0</b>	0.039826	0.040213	0.039738	0.973236	-0.219137
<b>2.0</b>	0.318847	0.324370	0.318876	1.732260	0.009032
<b>3.0</b>	0.350581	0.358527	0.350618	2.266446	0.010408
<b>4.0</b>	0.176308	0.167393	0.176399	-5.056334	0.051717
<b>5.0</b>	0.114438	0.109496	0.114369	-4.318374	-0.060464

# 데이터 이해를 위한 탐색과 시각화

Discover and Visualize the Data to Gain Insights

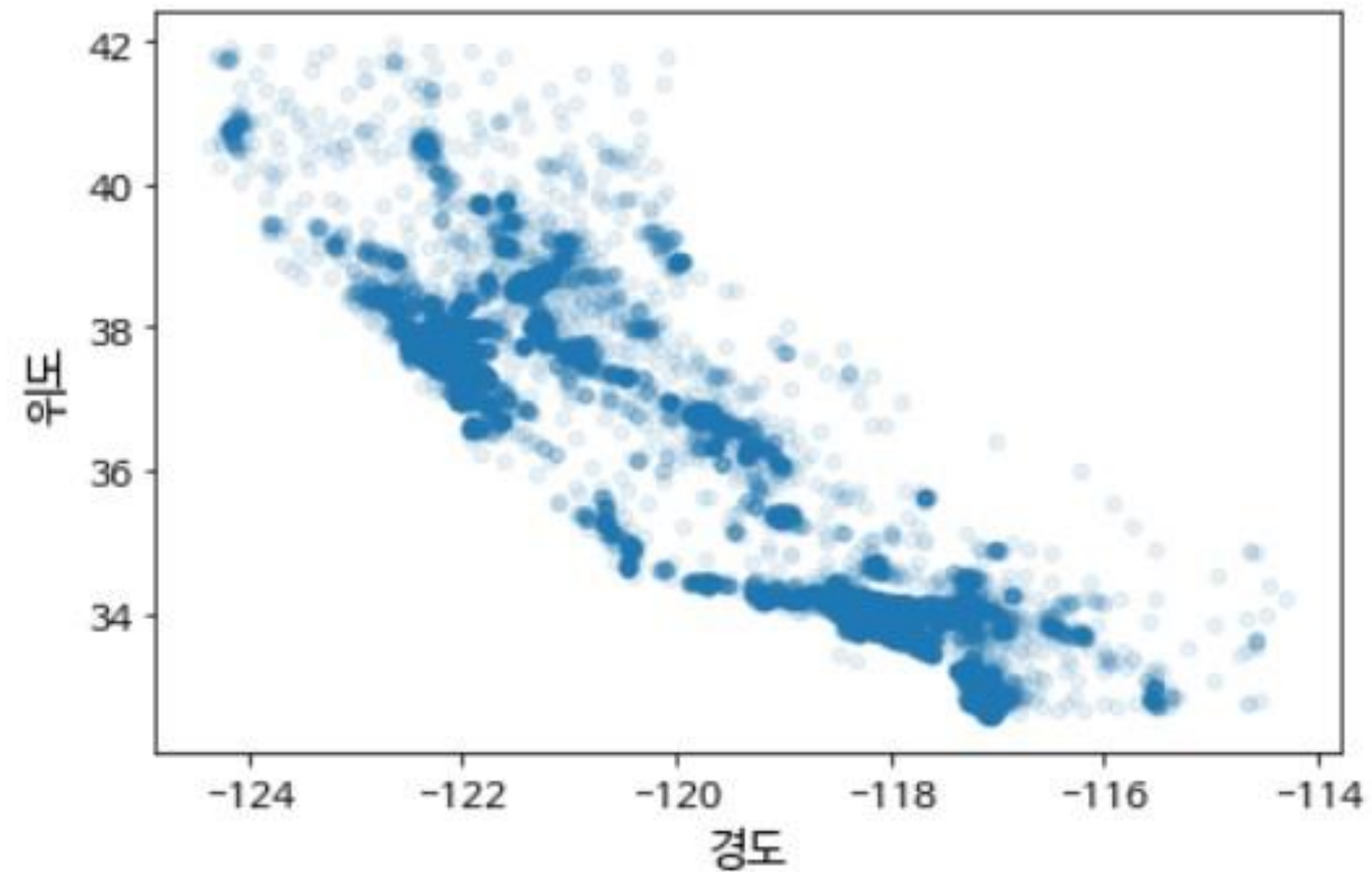
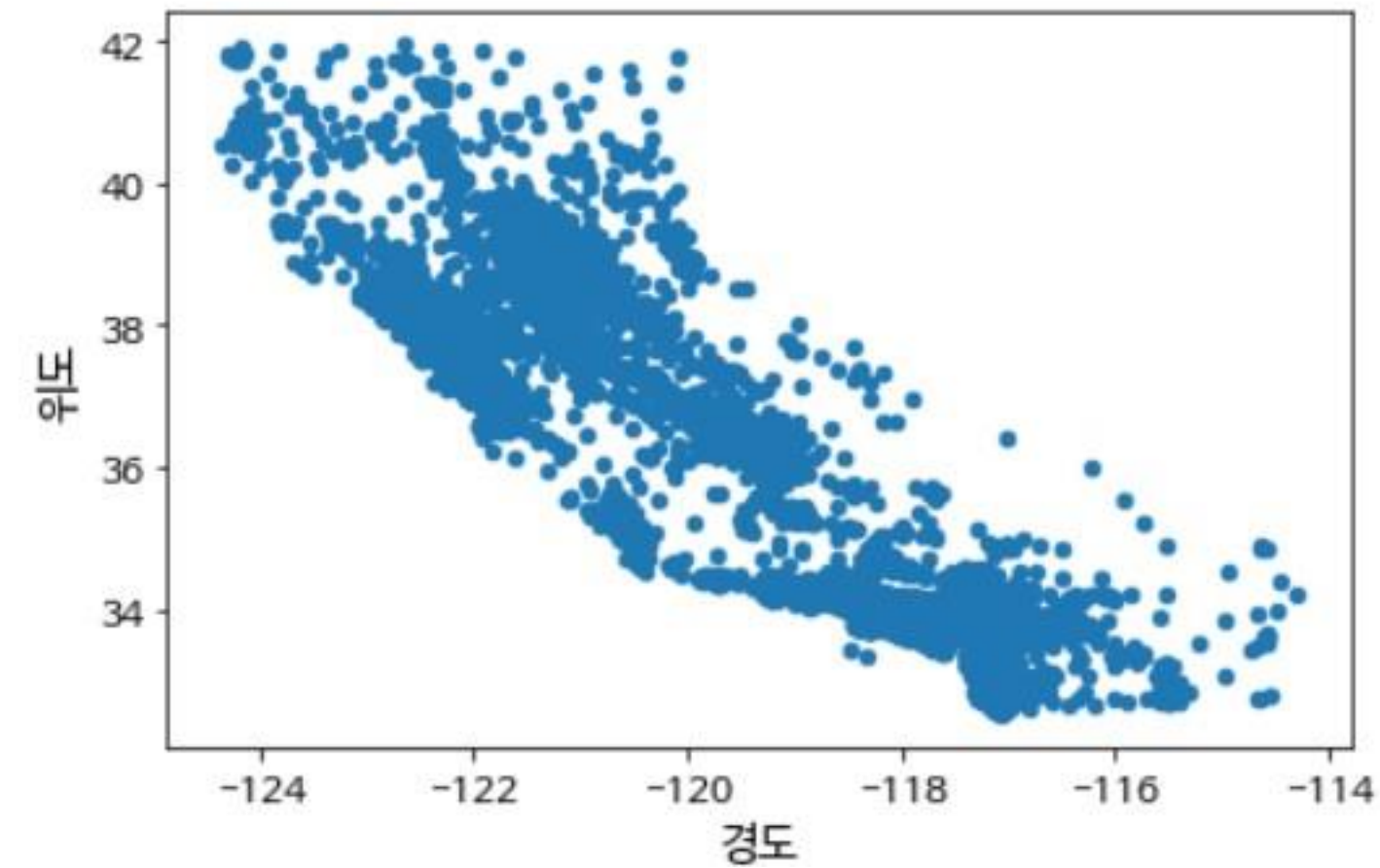
Training set에 대해서만 수행하는 것으로 가정 (Test set은 건드리지 말자)

- 시각화 (Visualizing Data)
- 상관계수 조사 (Looking for Correlations)
- 특징 조합 (Experimenting with Attribute Combinations)

# 데이터 이해: 시각화 : scatter plot

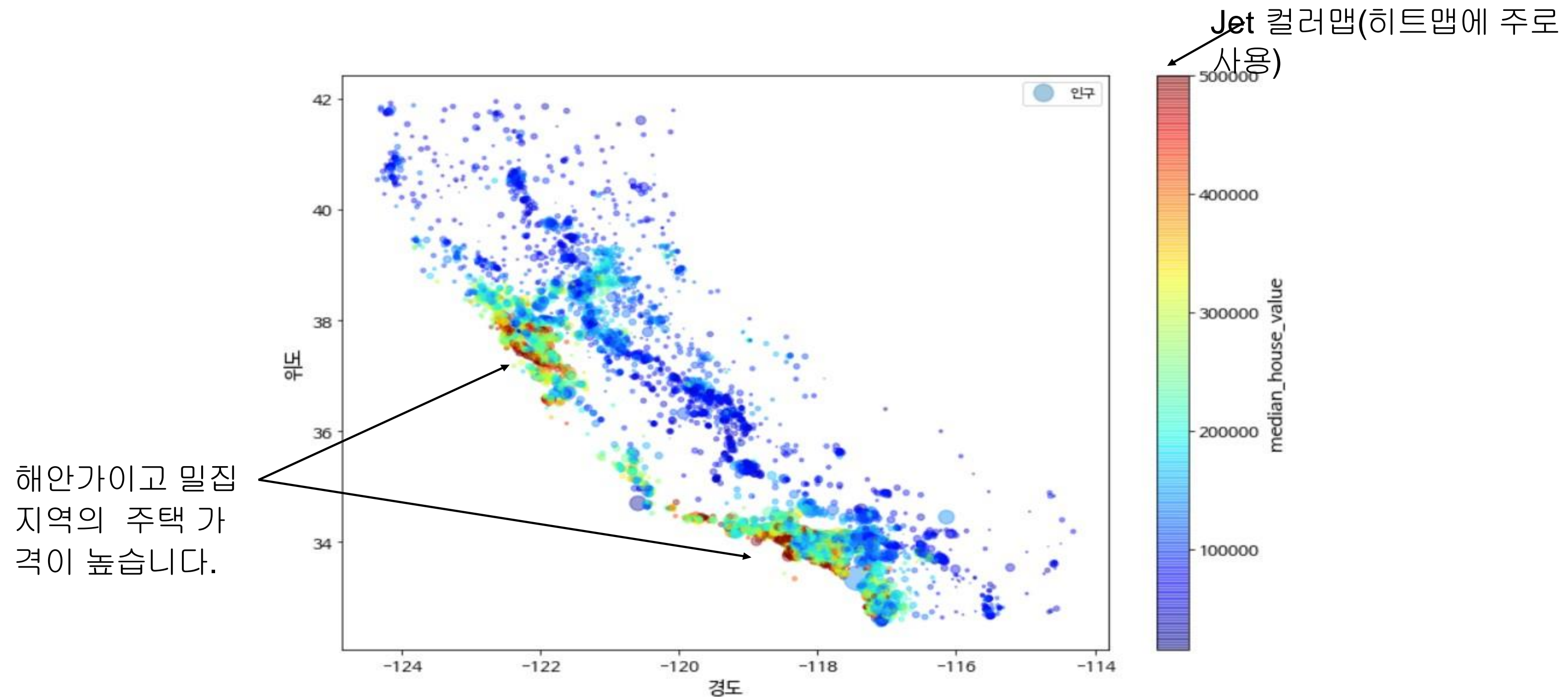
```
ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
ax.set(xlabel='경도', ylabel='위도')
save_fig("better_visualization_plot")
```

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude")
ax.set(xlabel='경도', ylabel='위도')
save_fig("bad_visualization_plot")
```





# 데이터 이해: 시각화 2 : color map



# 데이터 이해: 상관계수 : Pearson's r

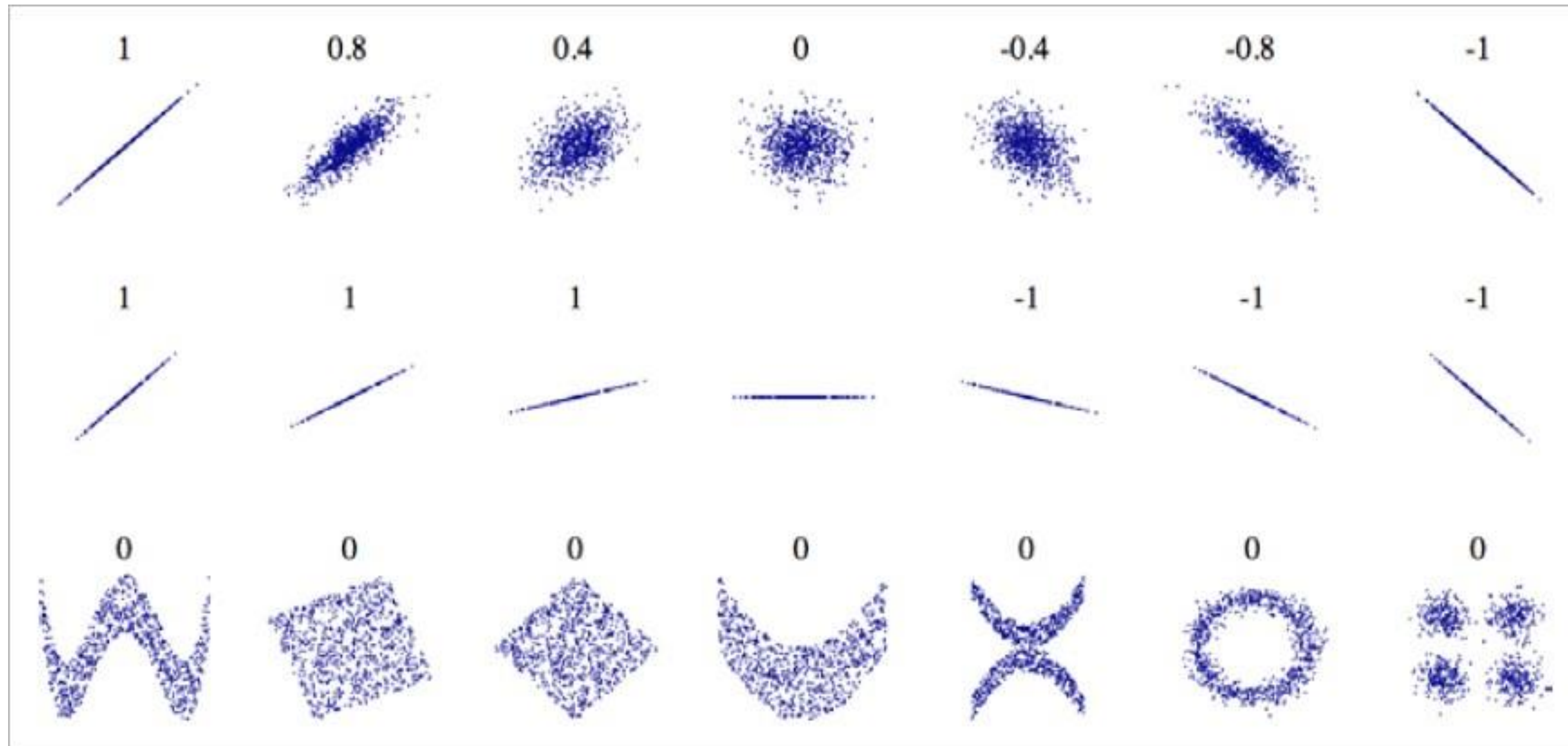
```
corr_matrix = housing.corr()
```

```
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
```

# 데이터 이해: 상관계수 2 : Pearson's r

- 선형 상관관계를 나타냄. (기울기와 상관없음)



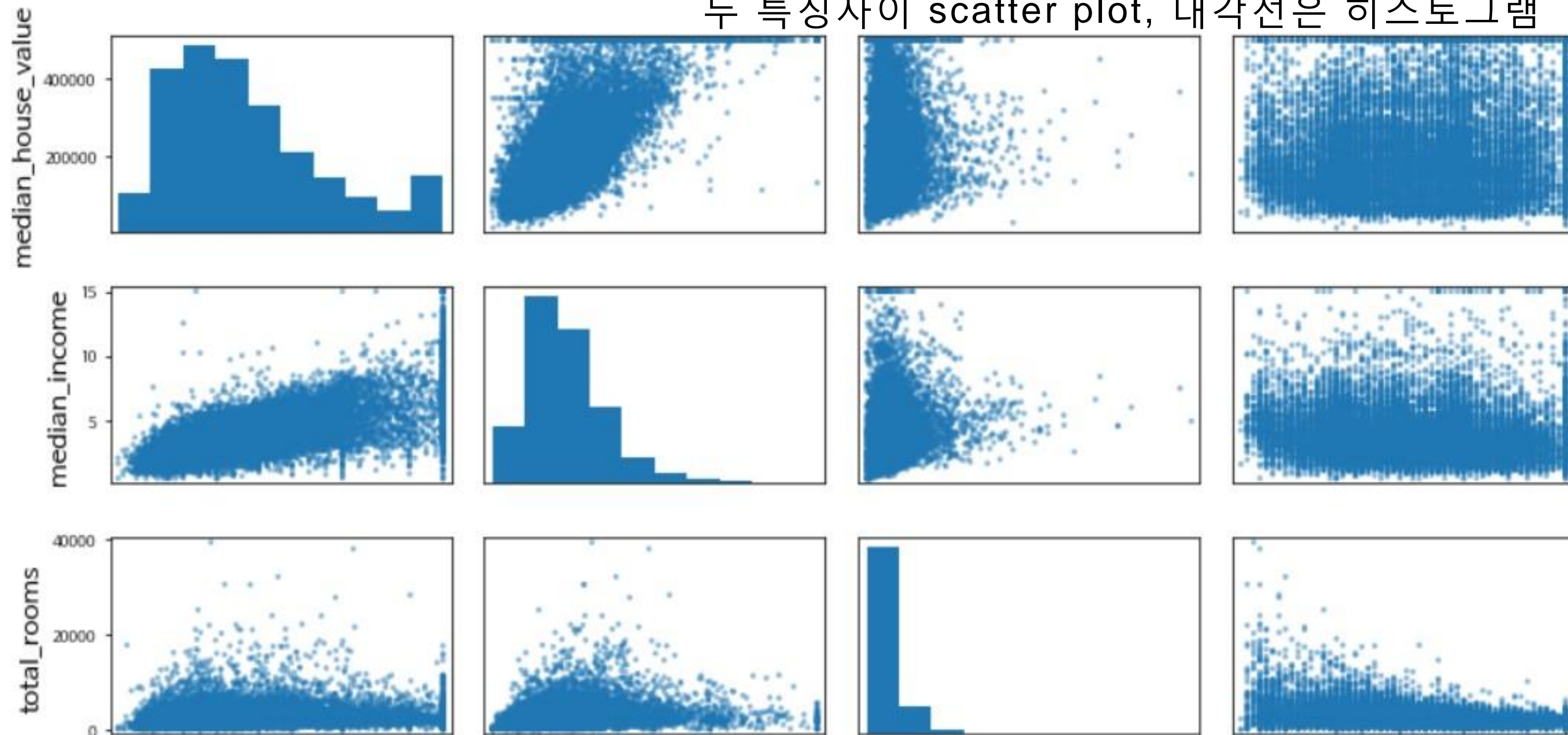


# 데이터 이해: 상관관계 시각화 : scatter\_matrix pandas

```
from pandas.plotting import scatter_matrix

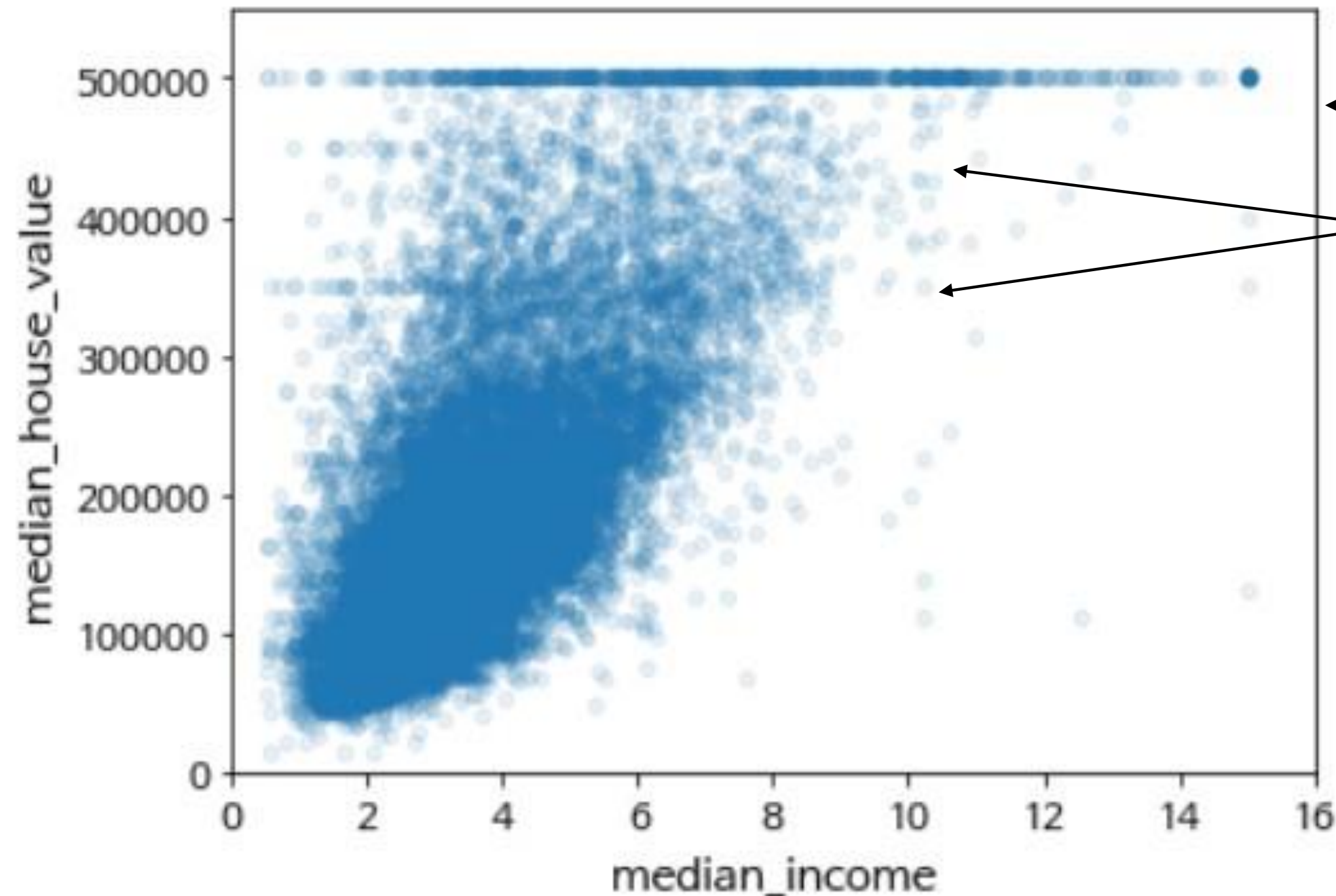
attributes = ["median_house_value", "median_income", "total_rooms",
             "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

두 특징사이 scatter plot, 대각선은 히스토그램



# 데이터 이해: 주택 가격 vs 소득 산점도 분석

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



← 가격제한 때문

← 수평선. 잘 못된 데이터 가능성

상관도 큼



# 데이터 이해: 특징 조합 실험

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

가구당 방 개수  
방당 침대수  
가구당 인원

```
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    0.146285
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
```



기존 특징보다 더 좋아 보임



기존 특징보다 더 좋아 보임

# 데이터 준비

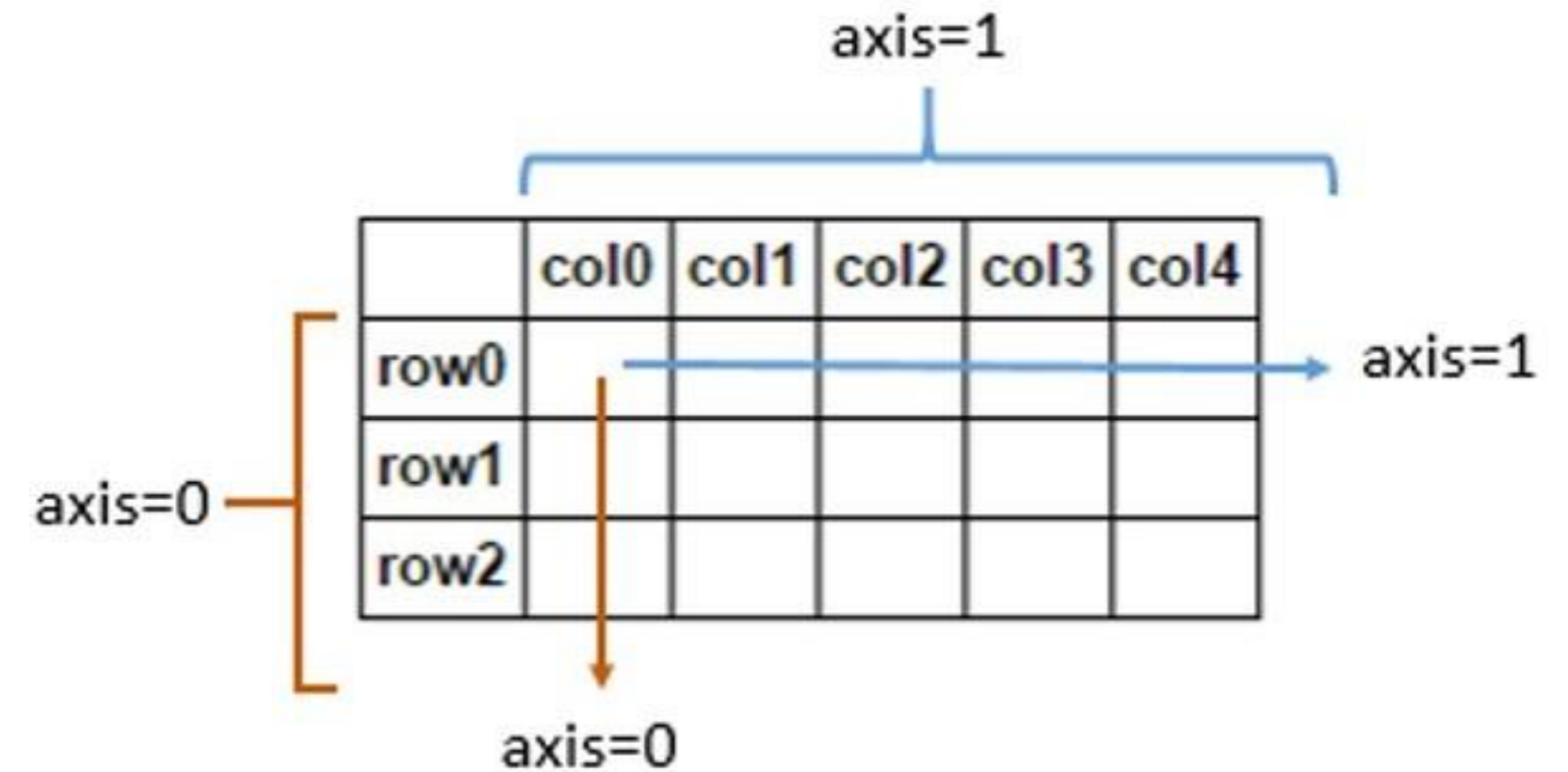
- 데이터 정제 (Data Cleaning)
- 범주형 데이터 다루기 (Handling Text and Categorical Attributes)
- 변환기 만들기 (Custom Transformers)
- 특징 스케일링 (Feature Scaling)
- 변환 파이프라인 (Transformation Pipelines)

- 함수로 만들어 놓으면 좋음

- 데이터 변환을 손쉽게 반복할 수 있음
- 다른 프로젝트에 재사용할 수 있음
- 론칭 후에 새 데이터에 적용할 때 사용함
- 최적의 조합을 찾는 데 편리함

# 데이터 준비

- 함수로 만들어 놓으면 좋음
  - 데이터 변환을 손쉽게 반복할 수 있음
  - 다른 프로젝트에 재사용할 수 있음
  - 론칭 후에 새 데이터에 적용할 때 사용함
  - 최적의 조합을 찾는 데 편리함



```
housing = strat_train_set.drop("median_house_value", axis=1) # 훈련 세트를 위해 레이블 삭제
housing_labels = strat_train_set["median_house_value"].copy()
```



# 데이터준비: 데이터 정제 1 : 샘플제거

- 옵션 1: 해당 샘플 제거,    옵션 2: 특징 삭제, 옵션 3: 대체(0, 평균, 중간값 등)

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_p
<b>4629</b>	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	2.2708	<1H
<b>6068</b>	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	5.1762	<1H
<b>17923</b>	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	4.6328	<1H
<b>13656</b>	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	1.6675	
<b>19252</b>	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	3.1662	<1H

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # 옵션 1
```

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximi
-----------	----------	--------------------	-------------	----------------	------------	------------	---------------	---------------

# 데이터준비: 데이터 정제 2 : 특징제거

```
sample_incomplete_rows.drop("total_bedrooms", axis=1) # 옵션 2
```

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	3.1662	<1H OCEAN



# 데이터준비: 데이터 정제 3 : 값 대체

```
median = housing["total_bedrooms"].median()  
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # 옵션 3  
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_p
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708	<1H
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762	<1H
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328	<1H
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675	
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662	<1H

# 데이터준비: 빈 값 대체 : scikit-learn의 Imputer

```
from sklearn.preprocessing import Imputer

imputer = Imputer(strategy="median")
```

중간값이 수치형 특성에서만 계산될 수 있기 때문에 텍스트 특성을 삭제합니다:

```
housing_num = housing.drop('ocean_proximity', axis=1)
# 다른 방법: housing_num = housing.select_dtypes(include=[np.number])
```

```
imputer.fit(housing_num)
```

```
Imputer(axis=0, copy=True, missing_values='NaN', strategy='median', verbose=0)
```

```
imputer.statistics_
```

```
array([-118.51 ,  34.26 ,  29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,  3.5409])
```

```
X = imputer.transform(housing_num)
```

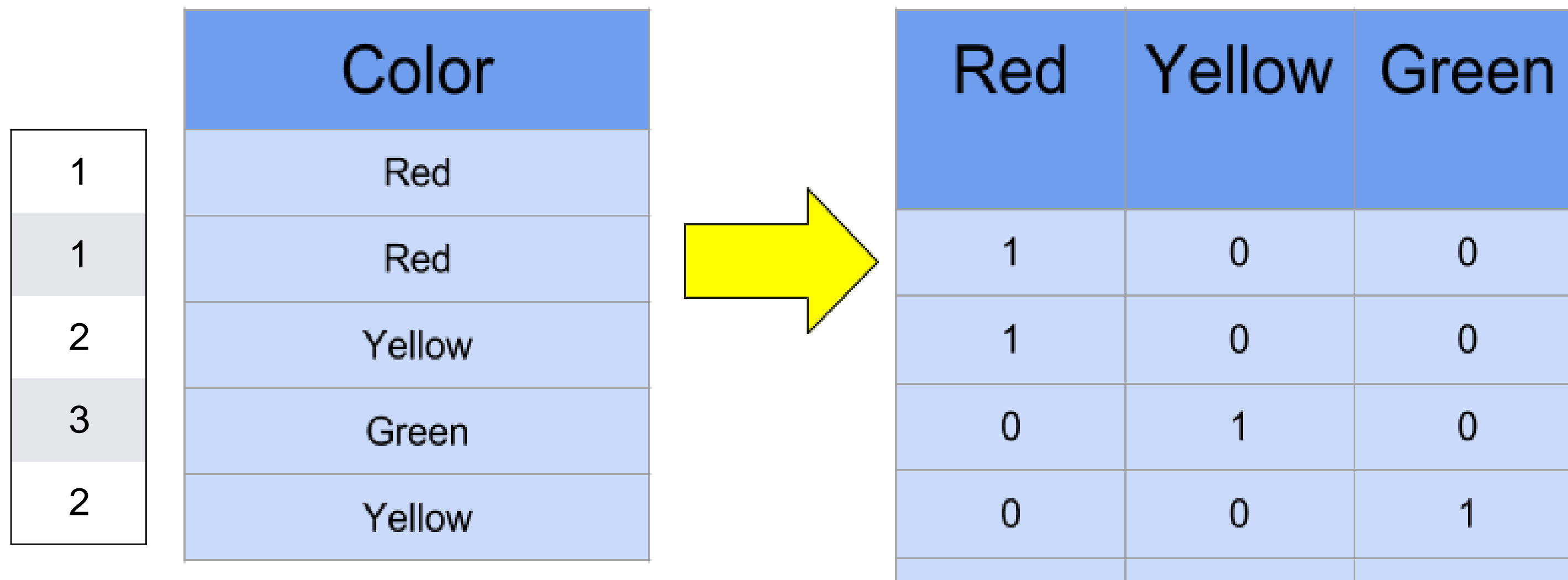


# 참고 : scikit-learn 설계 철학

- 일관성
  - 추정기: `fit(X, [y])`
  - 변환기: `transform(X)`, `fit_transform(X, [y])`
  - 예측기: `predict(X)`, `score(X, y)`
- 검사가능: 모델 파라미터(`imputer.statistics_`)와 하이퍼파라미터(`imputer.strategy`)를 공개 변수로 접근 가능
- 기본 데이터 타입으로 넘파이 배열을 사용합니다.
- 조합성: Pipeline 클래스
- 합리적 기본값: 모든 매개변수에 합리적인 기본값을 둬

# 데이터준비: 원-핫 인코딩 one-hot encoding

- 머신러닝 모델은 숫자만 처리할 수 있음. ➔ 카테고리(범주형)데이터를 숫자화 (보통 원핫 인코딩)



범주형. 숫자  
3이 1 보다 큰가?  
3이 1보다 2에 더 가까운가?

범주형. 텍스트

범주형. 원핫인코딩

# 데이터준비: 원핫인코딩 : factorize() : 범주형 → 숫자

```
housing_cat = housing['ocean_proximity']  
housing_cat.head(10)
```

```
17606    <1H OCEAN  
18632    <1H OCEAN  
14650    NEAR OCEAN  
3230      INLAND  
3555     <1H OCEAN  
19480     INLAND  
8879     <1H OCEAN  
13685     INLAND  
4937     <1H OCEAN  
4861     <1H OCEAN  
Name: ocean_proximity, dtype: object
```

```
housing_cat_encoded, housing_categories = housing_cat.factorize()  
housing_cat_encoded[:10]
```

pandas

```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0])
```

```
housing_categories
```

```
Index(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY', 'ISLAND'], dtype='object')
```

# 데이터준비: 원핫인코딩 : OneHotEncoder : 숫자 → OneHot

```
from sklearn.preprocessing import OneHotEncoder
```

```
encoder = OneHotEncoder()
```

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

2차원 배열로 변경

결과는 희소행렬

```
housing_cat_lhot.toarray()
```

희소행렬을 numpy배열로

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

(3,)

```
a = np.array([1, 2, 3])
a
```

```
array([1, 2, 3])
```

(3, 1)

```
a.reshape(-1, 1)
```

```
array([[1],
       [2],
       [3]])
```



# 데이터준비: 원핫인코딩 : OneHotEncoder : 범주형 → OneHot

scikit-learn의 LabelBinarizer사용하면 한번에 원핫으로 변환할 수 있음

```
>>> from sklearn.preprocessing import LabelBinarizer
>>> encoder = LabelBinarizer()
>>> housing_cat_1hot = encoder.fit_transform(housing_cat)
>>> housing_cat_1hot          : 결과는 numpy 배열로 나옴
array([[0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1],
       ...,
       [0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0]])
```

# 데이터준비: 원핫인코딩 : `pd.get_dummies()`

- pandas의 `get_dummies()` : 모든 문자열 특성을 원-핫 인코딩으로 변환함

```
pd.get_dummies(housing_cat).values
```

```
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]], dtype=uint8)
```

```
pd.get_dummies(housing)
```

median_income	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEAR BAY
2.7042	1	0	0	0
6.4214	1	0	0	0
2.8621	0	0	0	0
1.8839	0	1	0	0
3.0347	1	0	0	0
3.5395	0	1	0	0
8.3839	1	0	0	0
6.0000	0	1	0	0

# 데이터준비: 변환기 만들기

- `fit()`, `transform()`을 구현한 파이썬 클래스를 만들면 됨
- `TransformerMixin`을 상속하면 `fit_transform()` 메서드가 제공됨
- `BaseEstimator`를 상속하면 `get_params()`, `set_params()` 메서드가 제공됨
  - 클래스 생성자(`_init_`)에 `*args`, `**kwargs`를 사용하면 안됨
- `Pipeline` 클래스와 연계할 수 있음

# 데이터준비: 변환기 만들기 : CombinedAttributeAdder

```
from sklearn.base import BaseEstimator, TransformerMixin

# 컬럼 인덱스
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kwargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                          bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

가구당 방 개수  
가구당 인원

방당 침대수

결과는 넘파이 배열



# 데이터준비: 특징 스케일링

- 특징값 범위가 서로 다름
  - total\_number\_rooms : 6~39,320
  - median\_income : 0~15
- min-max scaling :
  - 최소값을 0, 최대값을 1이 되도록 변환
  - 잡음에 민감함 (예를들어, 백만개 중에 median\_income이 100인 잡음이 하나 있다면... )
  - scikit-learn : MinMaxScaler
- 정규화(normalization)
  - 평균 0, 분산 1 되도록 변환
  - 잡음에 강함 (예를들어, 백만개 중에 median\_income이 100인 잡음이 하나 있다면... )
  - scikit-learn : StandardScaler

# 데이터준비: 변환 파이프라인 : 전처리 단계 연결

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
num_pipeline = Pipeline([
    ('imputer', Imputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])
```

← 튜플 리스트

```
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Imputer.fit\_transform()-> CombinedAttributesAdder.fit\_transform()-> StandardScaler.fit\_transform()

```
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,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,
        -0.09586294,  0.10180567]])
```

클래스 이름의 소문자를 사용

```
from sklearn.pipeline import make_pipeline
num_pipeline2 = make_pipeline(Imputer(strategy="median"),
                              CombinedAttributesAdder(), StandardScaler())
num_pipeline2.fit_transform(housing_num)
```



# 데이터준비: 변환기 만들기 2 : DataFrameSelector

- 판다스의 데이터프레임에서 일부 컬럼을 선택하는 변환기를 만듭니다.

```
from sklearn.base import BaseEstimator, TransformerMixin

# 사이킷런이 DataFrame을 바로 사용하지 못하므로
# 수치형이나 범주형 컬럼을 선택하는 클래스를 만듭니다.
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```



# 데이터준비: 숫자와 문자열을 위한 파이프라인

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

```
num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attribs)),
    ('imputer', Imputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])
```

```
cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attribs)),
    ('cat_encoder', OneHotEncoder(sparse=False)),
])
```

num\_pipeline.steps

```
[('selector',
  DataFrameSelector(attribute_names=['longitude', 'latitude', 'housing_median_income', 'total_bedrooms', 'population', 'households', 'median_income'])),
 ('imputer',
  Imputer(axis=0, copy=True, missing_values='NaN', strategy='median')),
 ('attribs_adder', CombinedAttributesAdder(add_bedrooms_per_room=True)),
 ('std_scaler', StandardScaler(copy=True, with_mean=True, with_std=True))]
```

cat\_pipeline.steps

```
[('selector', DataFrameSelector(attribute_names=['ocean_proximity'])),
 ('cat_encoder',
  OneHotEncoder(categorical_features='all', dtype=<class 'numpy.float64'>,
    handle_unknown='error', n_values='auto', sparse=False))]
```



# 데이터준비: 두 파이프라인 연결

```
from sklearn.pipeline import FeatureUnion

full_pipeline = FeatureUnion(transformer_list=[
    ("num_pipeline", num_pipeline),
    ("cat_pipeline", cat_pipeline),
])
```

```
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)

← 10개 특성 중 ocean\_proximity 다섯 개로 늘어나고 3개 특성 추가됨

# 학습

- 모델 선택 (Model Selection)
- 학습 (Training)
- 평가 (Evaluation)

# 학습: 모델선택 및 학습 1 : 선형모델

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

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

학습

```
>>> some_data = housing.iloc[:5]  
>>> some_labels = housing_labels.iloc[:5]  
>>> some_data_prepared = full_pipeline.transform(some_data)  
>>> print("Predictions:\t", lin_reg.predict(some_data_prepared))  
Predictions:      [ 303104.   44800.  308928.  294208.  368704.]  
>>> print("Labels:\t\t", list(some_labels))  
Labels:      [359400.0, 69700.0, 302100.0, 301300.0, 351900.0]
```

예측결과 (일부샘플)  
→ 오차 큼

```
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
```

68628.19819848922

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

낮을수록 좋음

RMSE오차 (전체샘플)  
→ 오차 큼  
→ 과소적합?

# 학습: 모델선택 및 학습 2 : 결정트리

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

학습

```
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

RMSE오차

→ 오차 0

→ 과대적합 !

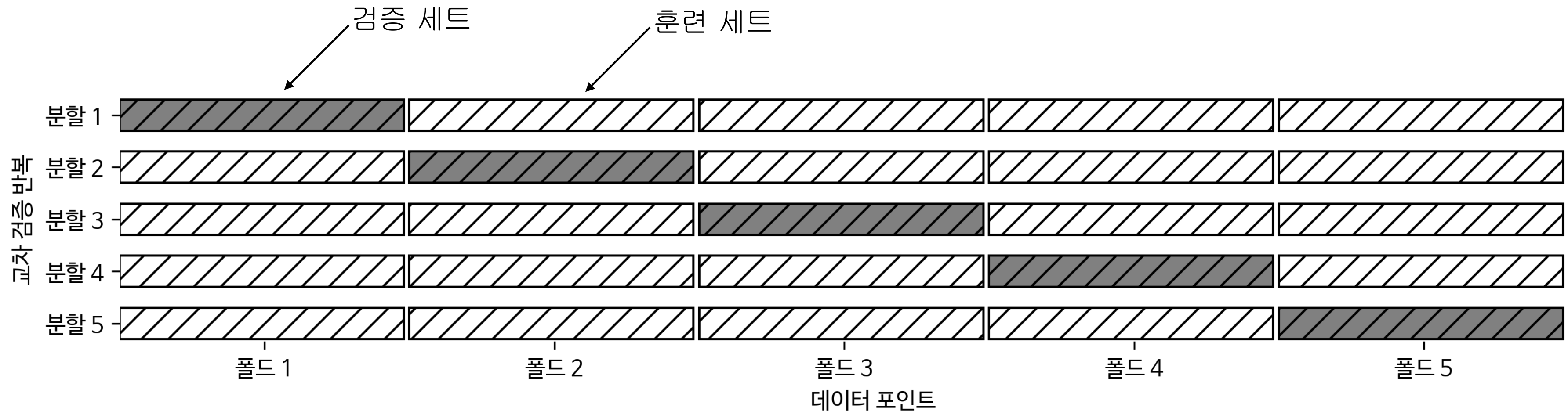
전체 샘플에 대해 훈련하고 검증한 것임

→ 훈련세트(training set)과 검증세트(test set)으로 나누어야 함



# 학습: 교차검증을 사용한 평가

- 간단한 방법 : 전체 데이터를 훈련세트와 검증세트로 나누어 (train\_test\_split사용),  
학습은 훈련세트로 검증은 검증세트로 함
- 교차검증(Cross Evaluation) : 훈련세트와 검증세트를 바꾸어 실험한 결과를 평균. 전체 데이터를 k개 집단으로 나눈 후 한 집단을 검증세트로, 나머지를 학습세트로 실험하는 과정을 모든 집단에 대해 수행하는 k-fold cross validation이 좋은 방법 (**cross\_val\_score** 사용)



# 학습: 교차검증 결과

```
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)
```

점수: [66782.73843989 66960.118071 70347.95244419 74739.57052552  
68031.13388938 71193.84183426 64969.63056405 68281.61137997  
71552.91566558 67665.10082067]  
평균: 69052.46136345083  
표준편차: 2731.6740017983466

이전결과(68628)와 차이 크지 않음

선형회귀 모델

```
from sklearn.model_selection import cross_val_score  
scores = cross_val_score(tree_reg, housing_prepared, housing_labels, scoring="neg_mean_squared_error", cv=10)  
rmse_scores = np.sqrt(-scores)  
display_scores(tree_rmse_scores)  
Scores: [ 74678.4916885 64766.2398337 69632.86942005 69166.67693232  
71486.76507766 73321.65695983 71860.04741226 71086.32691692  
76934.2726093 69060.93319262]  
Mean: 71199.4280043  
Standard deviation: 3202.70522793
```

이전결과(0)와 차이 매우 큼. 선형회기보다 나쁨

결정트리 모델

# 모델 세부 튜닝(Fine-Tune your Model)

모델이 결정되면, 최적의 모델 하이퍼파라미터를 찾아야 한다

- 그리드탐색(Grid Search)
- 랜덤탐색 (Randomized Search)
- 앙상블 방법 (Ensemble Methods)
- 최상의 모델과 오차분석 (Analyze the Best Models and their Errors)
- 테스트세트로 시스템 평가하기 (Evaluate your System on the Test Set)



# 모델 튜닝: 그리드탐색 : GridSearchCV

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # 하이퍼파라미터 12(=3×4)개의 조합을 시도합니다.
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # bootstrap은 False로 하고 6(=2×3)개의 조합을 시도합니다.
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]

forest_reg = RandomForestRegressor(random_state=42)
# 다섯 폴드에서 훈련하면 총 (12+6)*5=90번의 훈련이 일어납니다.
grid_search = GridSearchCV(forest_reg, param_grid, cv=5, scoring='neg_mean_squared_error',
                           return_train_score=True, n_jobs=-1)
grid_search.fit(housing_prepared, housing_labels)
```

3x4 + 2x3 = 12 + 6 = 18  
가지 경우

한 경우에 대해 5번 교차검증.  
12 x 5 = 60번 학습

```
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_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)
```

```
cvres = grid_search.cv_results_
```

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

결과 보기



# 모델 튜닝: 랜덤 탐색 : RandomizedSearchCV

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

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

```
forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distributions,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error',
                                random_state=42, n_jobs=-1)
rnd_search.fit(housing_prepared, housing_labels)
```

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

```
49147.15241724505 {'max_features': 7, 'n_estimators': 180}
51396.876896929905 {'max_features': 5, 'n_estimators': 15}
50797.05737322649 {'max_features': 3, 'n_estimators': 72}
50840.744513982805 {'max_features': 5, 'n_estimators': 21}
49276.17530332962 {'max_features': 7, 'n_estimators': 122}
50775.46331678437 {'max_features': 3, 'n_estimators': 75}
50681.383924974936 {'max_features': 3, 'n_estimators': 88}
49612.152530468346 {'max_features': 5, 'n_estimators': 100}
50473.01751424941 {'max_features': 3, 'n_estimators': 150}
64458.25385034794 {'max_features': 5, 'n_estimators': 2}
```

- Parameter값을 random하게 바꾸어가면서 n\_iter 번 학습
- 하이퍼파라미터 탐색공간이 커지면 GridSearch 보다 유용
- 반복횟수를 바꾸어서 탐색에 필요한 컴퓨팅 자원 조절 가능

# 모델 튜닝: 앙상블 방법

- 여러 개 모델의 모음(앙상블)을 혼합하는 방법
- 하나의 개별 인식기보다 좋은 성능
- 램덤포리스트가 대표적인 예 (7장)



# 모델 튜닝: 최상의 모델과 오차분석

- 선택된 최상의 모델에서 좋은 통찰을 얻는 경우가 많다.
- 예를들어, 특징의 중요도를 구함 → 쓸모없는 특징을 버리고 새로 학습이 가능

```
>feature_importances =  
grid_search.best_estimator_.feature_importances_  
>feature_importances  
array([ 7.14156423e-02, 6.76139189e-02,  
4.44260894e-02,  
1.66308583e-02, 1.66076861e-02, 1.82402545e-02,  
1.63458761e-02, 3.26497987e-01, 6.04365775e-02,  
1.13055290e-01, 7.79324766e-02, 1.12166442e-02,  
1.53344918e-01, 8.41308969e-05, 2.68483884e-03,  
3.46681181e-03])
```

ocean\_proximity를 원핫인코딩 한 것  
처음 특징( <1H OCEAN )만 중요  
아래 특징들은 불필요.

```
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]  
>>> cat_one_hot_attribs = list(encoder.classes_)  
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs  
>>> sorted(zip(feature_importances, attributes), reverse=True)  
[(0.32649798665134971, 'median_income'),  
(0.15334491760305854, 'INLAND'),  
(0.11305529021187399, 'pop_per_hhold'),  
(0.07793247662544775, 'bedrooms_per_room'),  
(0.071415642259275158, 'longitude'),  
(0.067613918945568688, 'latitude'),  
(0.060436577499703222, 'rooms_per_hhold'),  
(0.04442608939578685, 'housing_median_age'),  
(0.018240254462909437, 'population'),  
(0.01663085833886218, 'total_rooms'),  
(0.016607686091288865, 'total_bedrooms'),  
(0.016345876147580776, 'households'),  
(0.011216644219017424, '<1H OCEAN'),  
(0.0034668118081117387, 'NEAR OCEAN'),  
(0.0026848388432755429, 'NEAR BAY'),  
(8.4130896890070617e-05, 'ISLAND')]
```



# 모델 튜닝: 테스트 세트로 시스템 평가

- 마지막에 딱 한번만 수행함. (테스트 세트를 반복하여 사용하면 테스트 세트에 과대 적합된 모델을 만들기 때문에)
- 일반적으로, 교차검증으로 평가한 것에 비해 성능이 조금 낮음

```
final_model = grid_search.best_estimator_  
  
X_test = strat_test_set.drop("median_house_value", axis=1)  
y_test = strat_test_set["median_house_value"].copy()  
  
X_test_prepared = full_pipeline.transform(X_test)  
final_predictions = final_model.predict(X_test_prepared)  
  
final_mse = mean_squared_error(y_test, final_predictions)  
final_rmse = np.sqrt(final_mse)
```

```
final_rmse
```

```
47766.00396643308
```

# 참고 : 전체 파이프라인

- 전처리 파이프라인과 모델을 하나의 파이프라인으로 연결할 수 있음
- 그리드탐색으로 전체 과정을 자동화할 수 있음

```
full_pipeline_with_predictor = Pipeline([
    ("preparation", full_pipeline),
    ("linear", LinearRegression())
])

full_pipeline_with_predictor.fit(housing, housing_labels)
full_pipeline_with_predictor.predict(some_data)

array([210644.60459286, 317768.80697211, 210956.43331178, 59218.98886849,
       189747.55849879])
```

# 참고 : 모델 저장

- 하이퍼파라미터와 모델 파라미터를 모두 저장합니다.

```
my_model = full_pipeline_with_predictor
```

```
from sklearn.externals import joblib
joblib.dump(my_model, "my_model.pkl")
#...
my_model_loaded = joblib.load("my_model.pkl")
```



# 론칭 Launch, Monitor and Maintain

- 사용자 데이터를 입력 받는 프로그램 작성
- 실시간 성능 체크를 위한 모니터링 코드 개발 : 정해진 시간마다 시스템이 잘 동작하고 있는지를 확인하는 프로그램 필요
- 성능 평가 : 시스템 예측값들을 샘플링해서 평가 (보통 전문가가 수행)
- 입력 데이터 모니터링 코드 개발 : 사용자 입력(혹은 센서 입력일 수도 있음)이 잘 못 되었는지 확인
- 정기적인 훈련을 위한 자동화 : 새로운 추세를 반영하기 위해 주기적으로 재 훈련하도록 함. 6개월이면 너무 늦음.

감사합니다