

# 핸즈온 머신러닝

2장. 머신러닝 프로젝트 처음부터 끝까지

박해선(옮긴이)

[haesun.park@tensorflow.blog](mailto:haesun.park@tensorflow.blog)  
<https://tensorflow.blog>

# 프로젝트 진행 과정

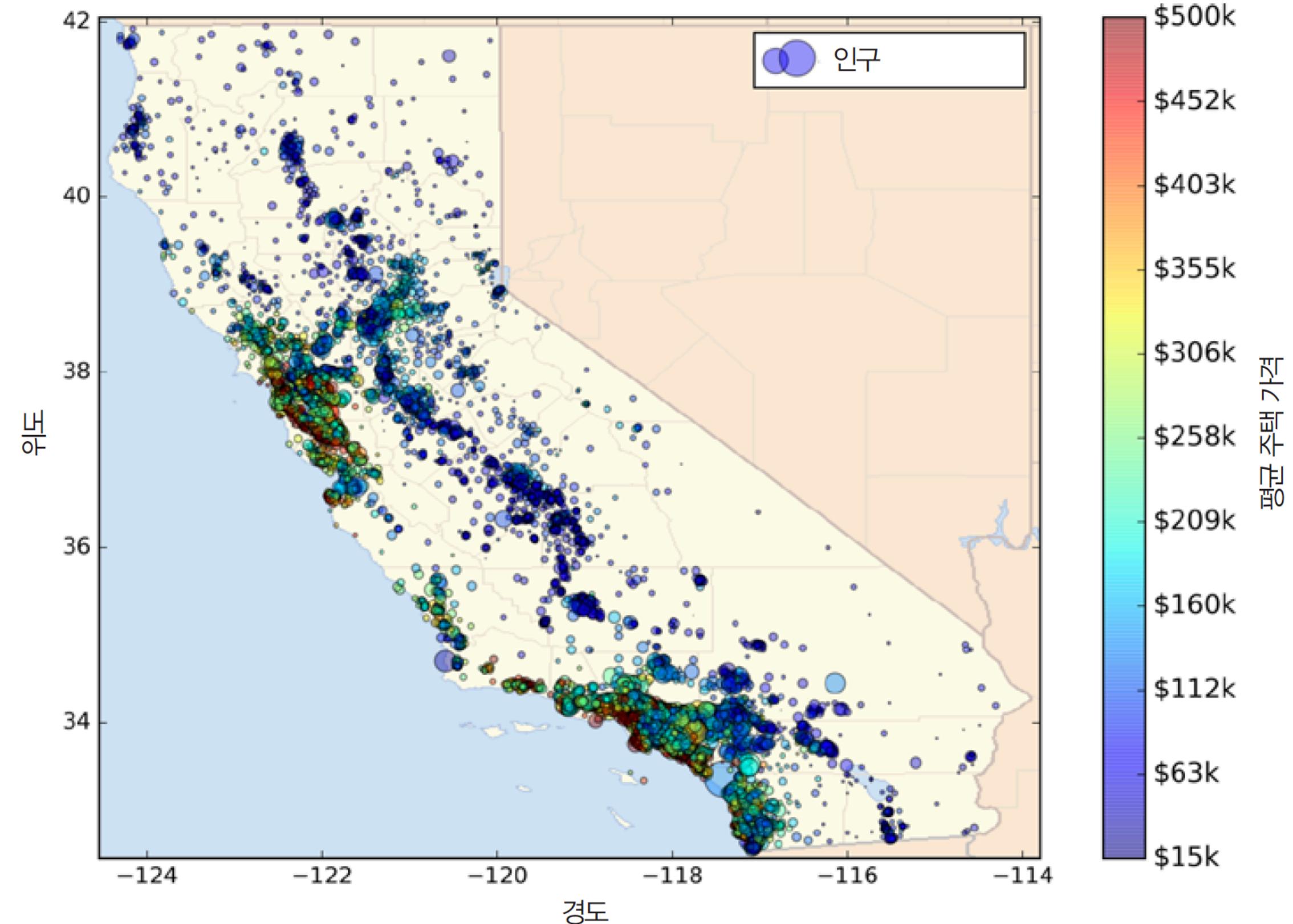
- 큰 그림을 봅니다.
- 데이터를 구합니다.
- 데이터로부터 통찰을 얻기 위해 탐색하고 시각화합니다.
- 머신러닝 알고리즘을 위해 데이터를 준비합니다.
- 모델을 선택하고 훈련시킵니다.
- 모델을 상세하게 조정합니다.
- 솔루션을 제시합니다.
- 시스템을 론칭하고 모니터링하고 유지 보수합니다.

# 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 서브레딧<sup>subreddit</sup> (<http://www.reddit.com/r/datasets>)

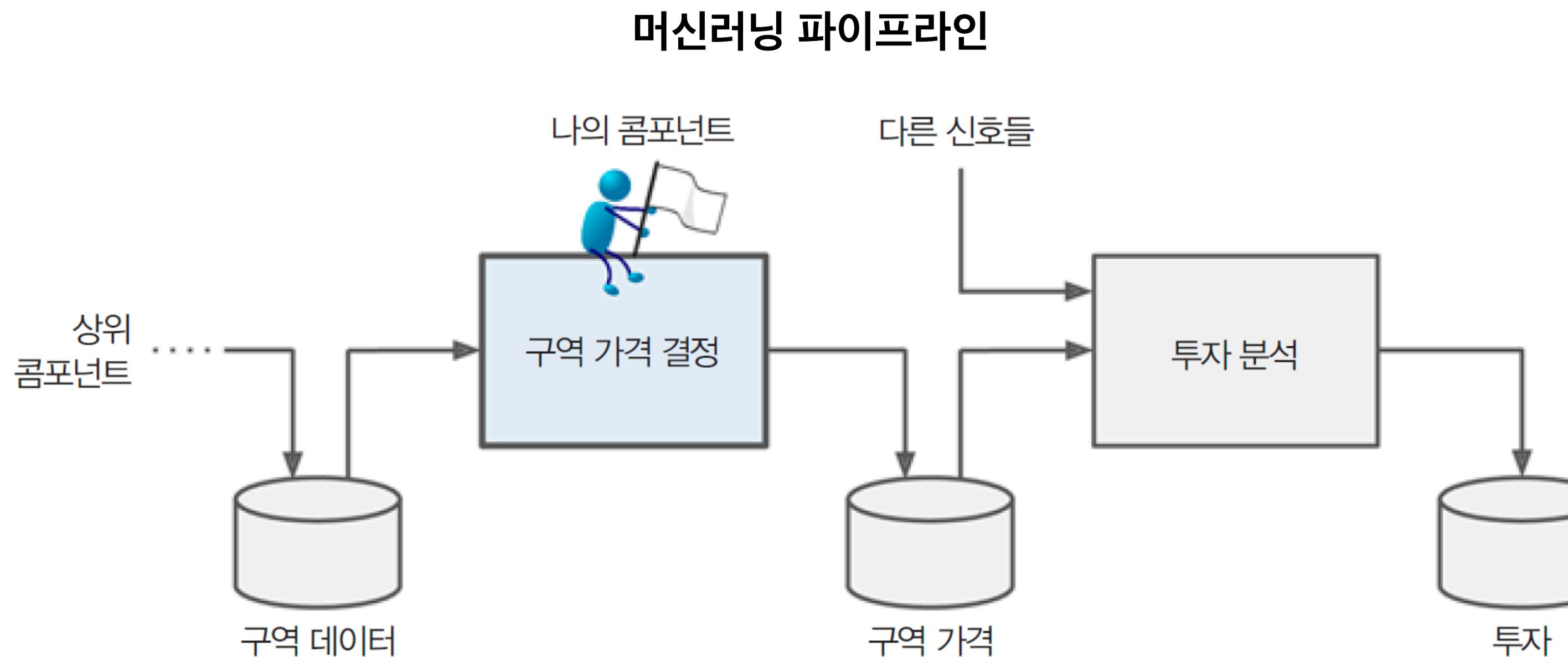
# 캘리포니아 주택 가격 데이터셋

- 1990년 캘리포니아 인구조사 데이터를 기반으로 합니다.
- 카네기 멜론Carnegie Mellon 대학교의 통계학과에서 운영하는 StatLib 저장소(<http://lib.stat.cmu.edu/datasets/>)의 데이터를 수정한 버전을 사용합니다 ([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
- 머신러닝 체크리스트(부록 B)



# 비즈니스의 목적은?

- 시스템의 구성, 알고리즘, 측정 지표, 튜닝 시간 등을 결정하기 때문에 중요합니다.
- “중간 주택 가격 예측이 다른 신호와 함께 **투자 결정**을 내리는 머신러닝 시스템의 입력으로 사용됩니다.”



# 현재 솔루션은?

- 현재 솔루션이 있다면 현재 해결 방법에 대한 정보를 얻을 수 있고 참고 성능으로도 사용할 수 있습니다.
- “현재는 전문가가 수동으로 중간 주택 가격을 추정합니다. 구역에 관한 최신 정보를 모으고 복잡한 규칙을 사용합니다.”
- “수동으로 추정한 값은 10% 이상 벗어날 때가 많습니다.”
- 문제 정의: 지도/비지도/강화학습, 분류/회귀, 배치/온라인?

# 문제 정의

- 레이블된 샘플이 있으므로 지도 학습입니다.
- 연속된 값을 예측하므로 회귀 문제입니다. 특히 여러 특성을 사용하므로 다변량 회귀 multivariate regression입니다(반대는 단변량 회귀 univariate regression).
- 데이터는 오프라인으로 준비되어 있고 크지 않으므로 배치 학습이 적당합니다.

# 성능 측정 지표

- 평균 제곱근 오차 Root Mean Square Error(RMSE)

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

가설(hypothesis)  
샘플 개수      특성      타깃 or 레이블

- R<sup>2</sup> (scikit-learn 기본값)

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

when 선형 회귀

# 다른 지표는?

- 평균 절대 오차 Mean Absolute Error(MAE)

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

- 노름 Norm

$$\|\nu\|_k = (\|v_0\|^k + \|v_1\|^k + \cdots + \|v_n\|^k)^{\frac{1}{k}}$$

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

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

$k$ 가 클수록 큰 원소에 치우칩니다 : RMSE가 MAE 보다 이상치에 더 민감

$k$ 가 무한대이면 가장 큰 원소의 절대값

$k$ 가 0이면 벡터에서 0이 아닌 원소의 수입니다

# 표기법

- 어떤 구역의 경도가 -118.29, 위도 33.91, 주민수 1,416, 중간 소득 \$38,372이고 중간 주택 가격이 \$156,400이라면,

$$\mathbf{X}^{(i)} = \begin{bmatrix} -118.29 \\ 33.91 \\ 1,416 \\ 38,372 \end{bmatrix}$$

벡터(굵은 소문자)

행렬(굵은 대문자)

특성(n)

타깃 or 레이블

예측 값(y-햇)

가설(hypothesis)

$$\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$$
$$\mathbf{X}^{(i)} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(1999)})^T \\ (\mathbf{x}^{(20000)})^T \end{bmatrix} = \begin{bmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

# 가정 검사

- 지금까지 세운 가정을 나열하고 검증하세요.
- 혹시 하위 시스템이 중간 주택 가격이 아니고 등급(분류 문제)을 원하지 않나요?
- 너무 늦게 문제를 발견하지 않도록 주의하세요.

# 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") ← 함수를 만들어 사용합니다
    urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()

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)
```

윈도우 환경을 위해

# 데이터셋의 처음 다섯행

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

10개의 특성

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	341300.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	342200.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

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  
-122.24,37.85,52.0,1467.0,190.0,496.0,177.0,7.2574,341300.0,NEAR BAY  
-122.25,37.85,52.0,1274.0,235.0,558.0,219.0,5.6431,342200.0,NEAR BAY  
-122.25,37.85,52.0,1627.0,280.0,565.0,259.0,3.8462,342200.0,NEAR BAY
```

# df.info()

```
20,640개 샘플  
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 ← 207개가 비어 있음  
 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 ← 문자열 특성(범주형)  
 dtypes: float64(9), object(1)  
 memory usage: 1.6+ MB
```

# 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

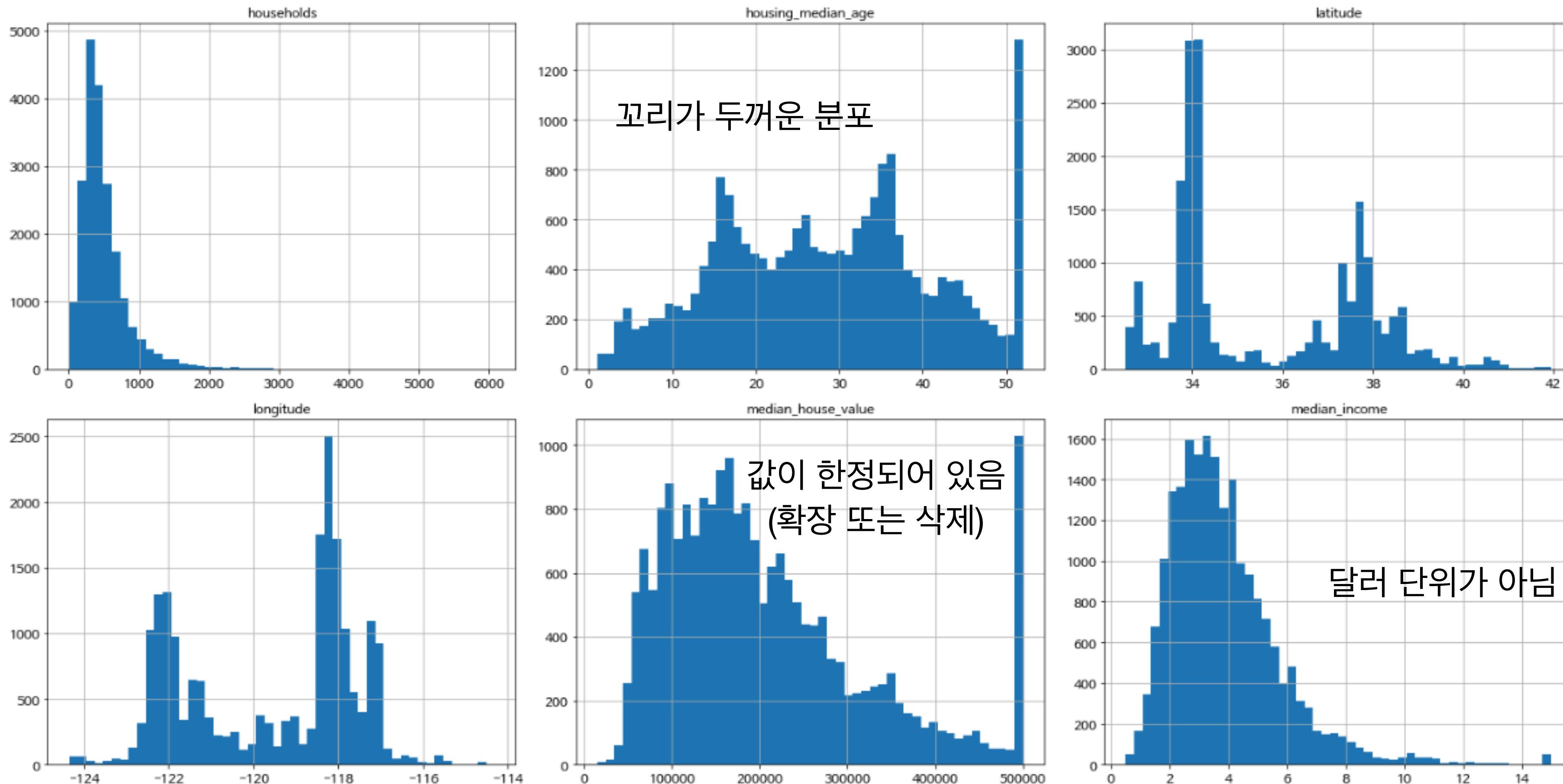
← 범주형을 확인합니다

# df.describe()

housing.describe() ← 숫자형 특성을 요약합니다									
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_in	
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.81	1.000000
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.89	1.000000
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.49	0.000000
1사분위	<b>25%</b>	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.50
	<b>50%</b>	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.50
3사분위	<b>75%</b>	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.70
	<b>max</b>	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.00

# df.hist()

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



# 테스트 데이터

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

단순한 방법

```
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

# split\_train\_test의 문제점

- 함수를 실행할 때마다 다른 테스트 세트가 만들어 집니다(데이터 스누핑 우려)
- 해결책
  - 테스트 세트를 따로 떼어 저장하거나 np.random.seed() 함수 사용
  - 업데이트된 데이터셋을 적용하기 어려움
- 샘플의 식별자를 해싱한 값을 기준으로 분리

# test\_set\_check

```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    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]
```

for Python 2

# train\_set, 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")
```

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]  
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

test\_set.head()

새로운 데이터는 행 끝에 추가되고 이전 행은 삭제되지 않아야 합니다

	index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_
8	8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	
10	10	-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	
11	11	-122.26	37.85	52.0	3503.0	752.0	1504.0	734.0	
12	12	-122.26	37.85	52.0	2491.0	474.0	1098.0	468.0	
13	13	-122.26	37.84	52.0	696.0	191.0	345.0	174.0	

범주형 특성의 카테고리 분포를 잘 유지하고 있을까요?

# with scikit-learn

여러개의 배열을 넣을 수 있습니다  
(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)
```

```
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

# 샘플링 편향

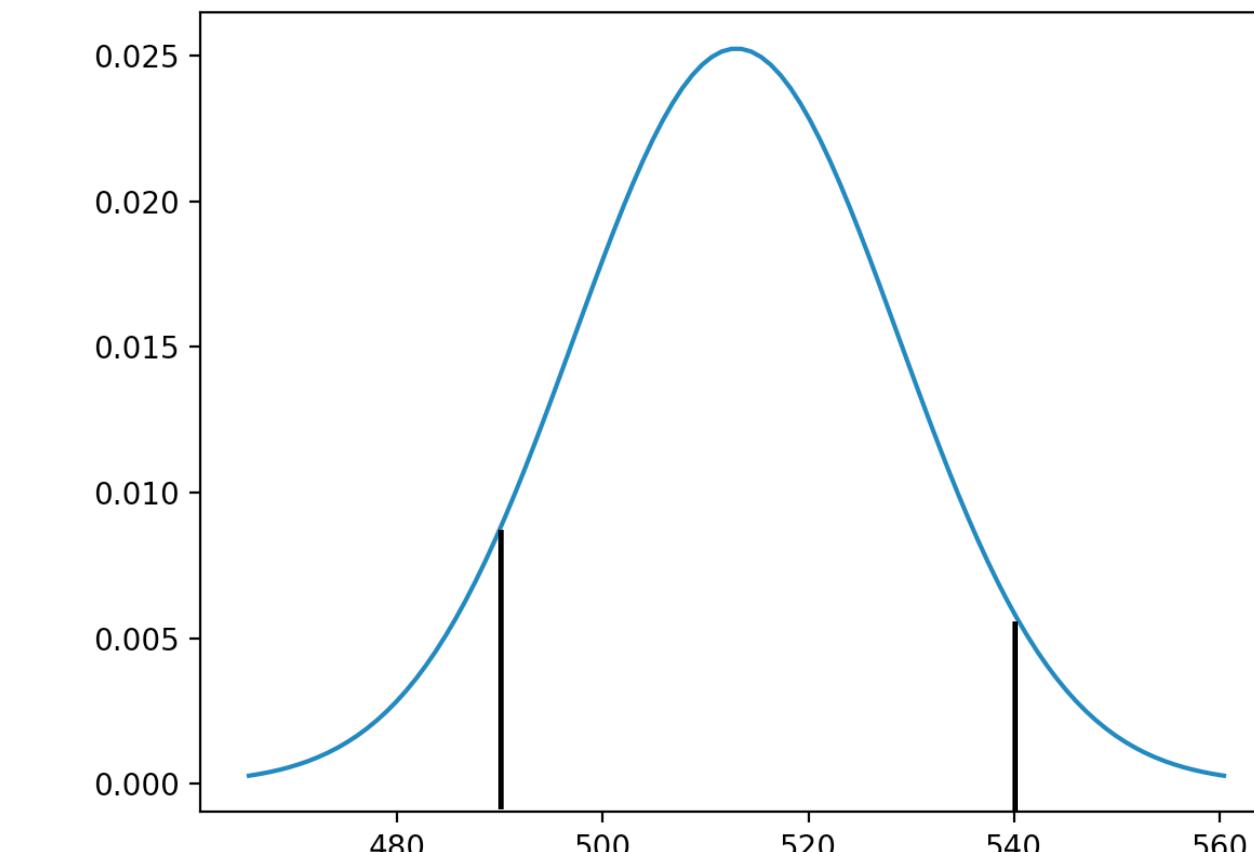
- 1,000명 중 여성이 48.7%일 때 무작위로 선택하면
  - 49%보다 적거나 54% 많은 여성이 테스트 세트에 들어갈 확률이 약 12%입니다

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 근사합니다.

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

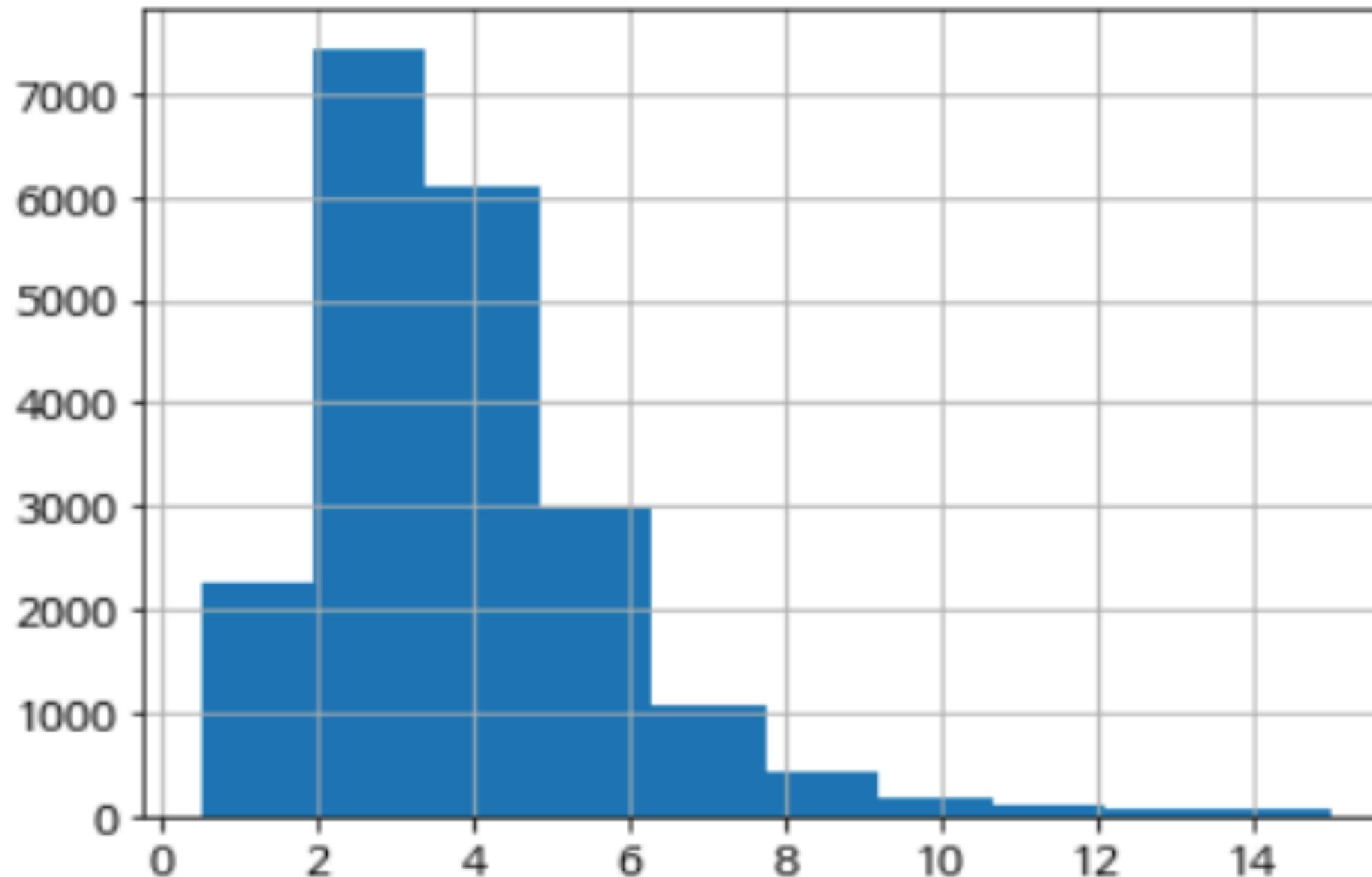


# media\_income

중간소득이 중간 주택 가격을 예측하는 데 중요하다고 가정합니다.

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

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



# income\_cat

```
# 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)
```

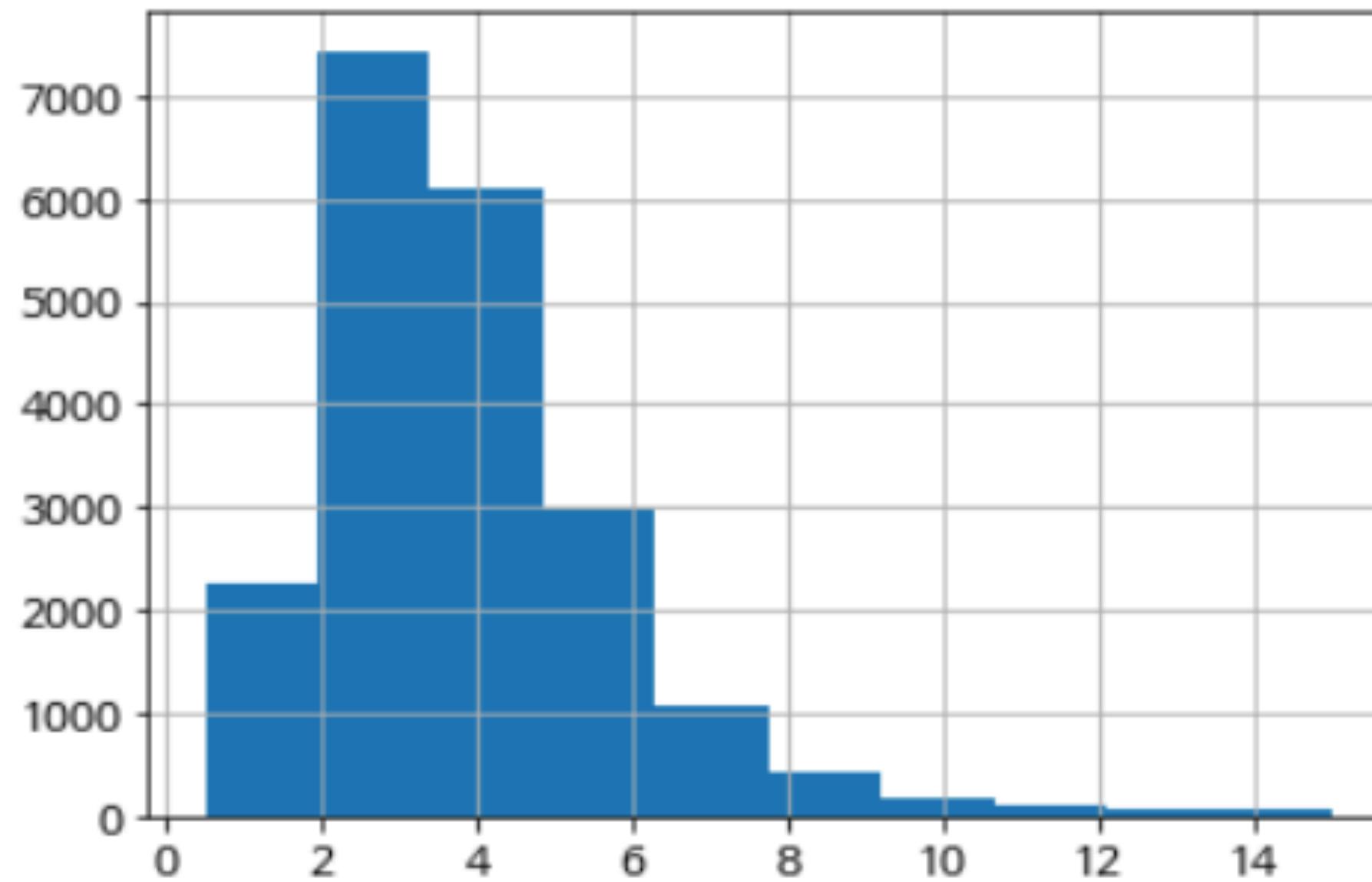
```
housing["income_cat"].value_counts()
```

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

# income\_cat histogram

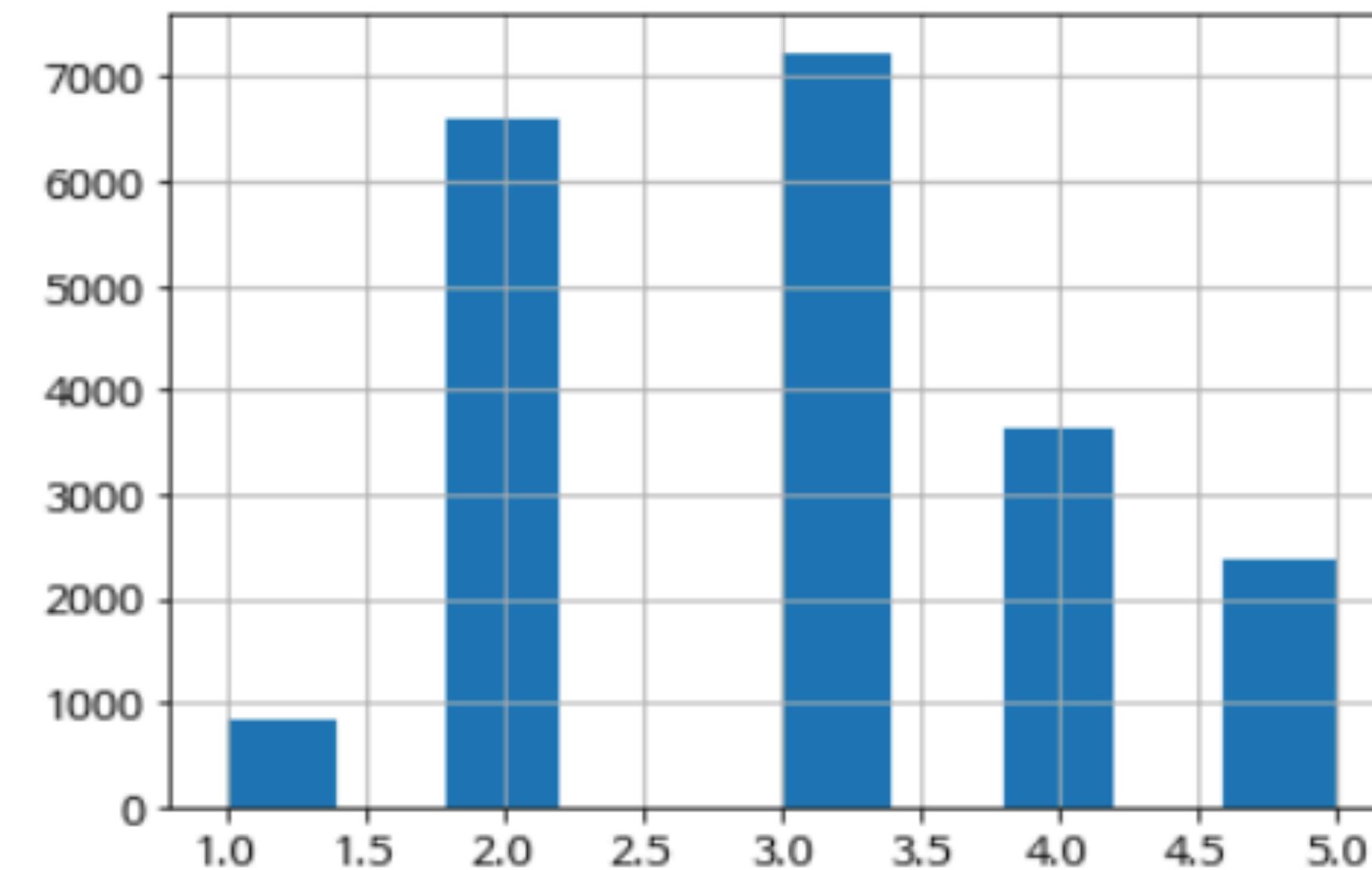
```
housing["median_income"].hist()
```

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



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

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



# StratifiedShuffleSplit

- StratifiedKFold + ShuffleSplit
- test\_size와 train\_size 매개변수 합을 1이하로 지정할 수 있습니다.

```
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
```

# with train\_test\_split

```
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
```

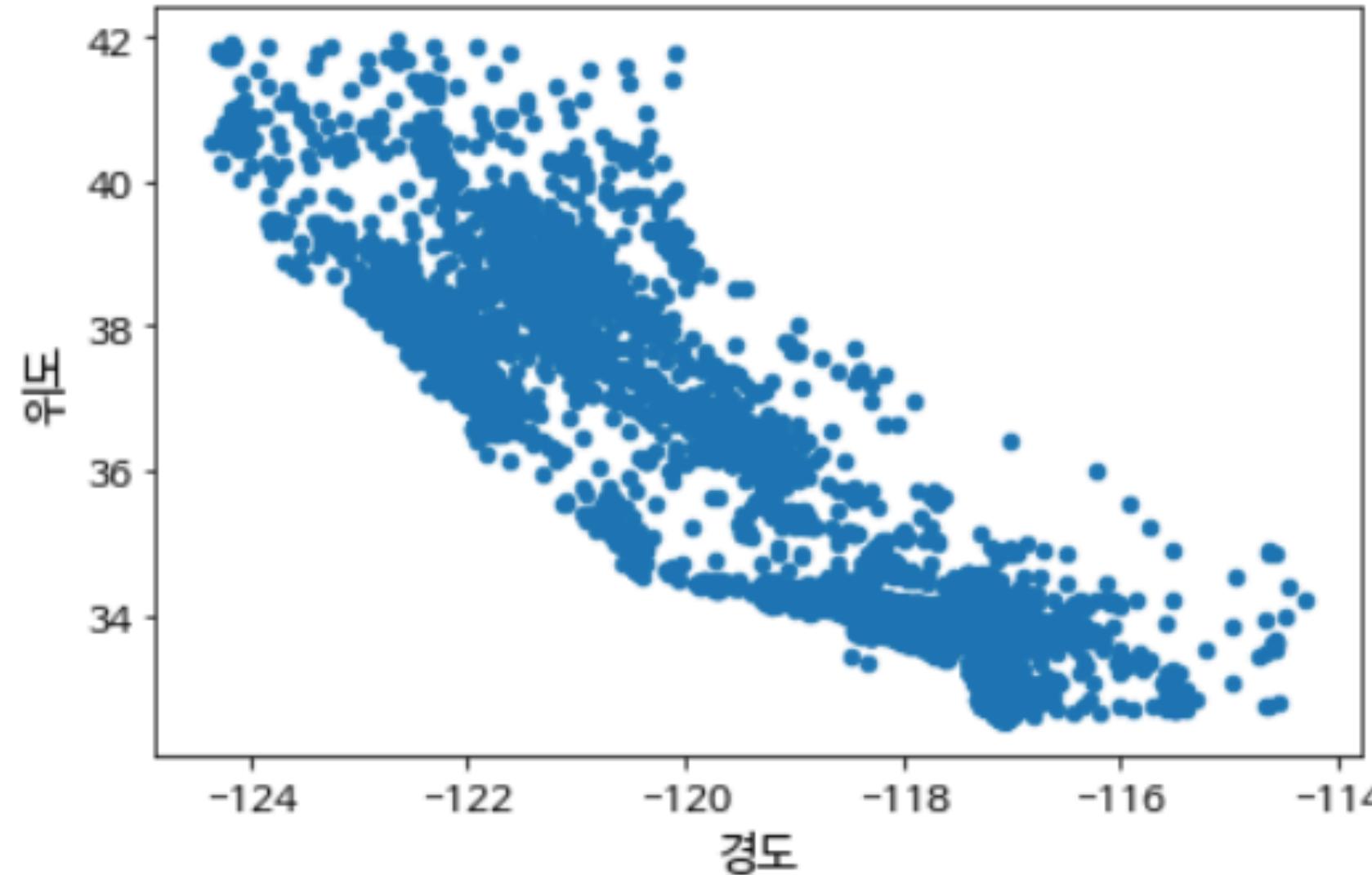
StratifiedShuffleSplit 사용  
(아니면 ShuffleSplit 사용)

# 샘플링 편향 비교

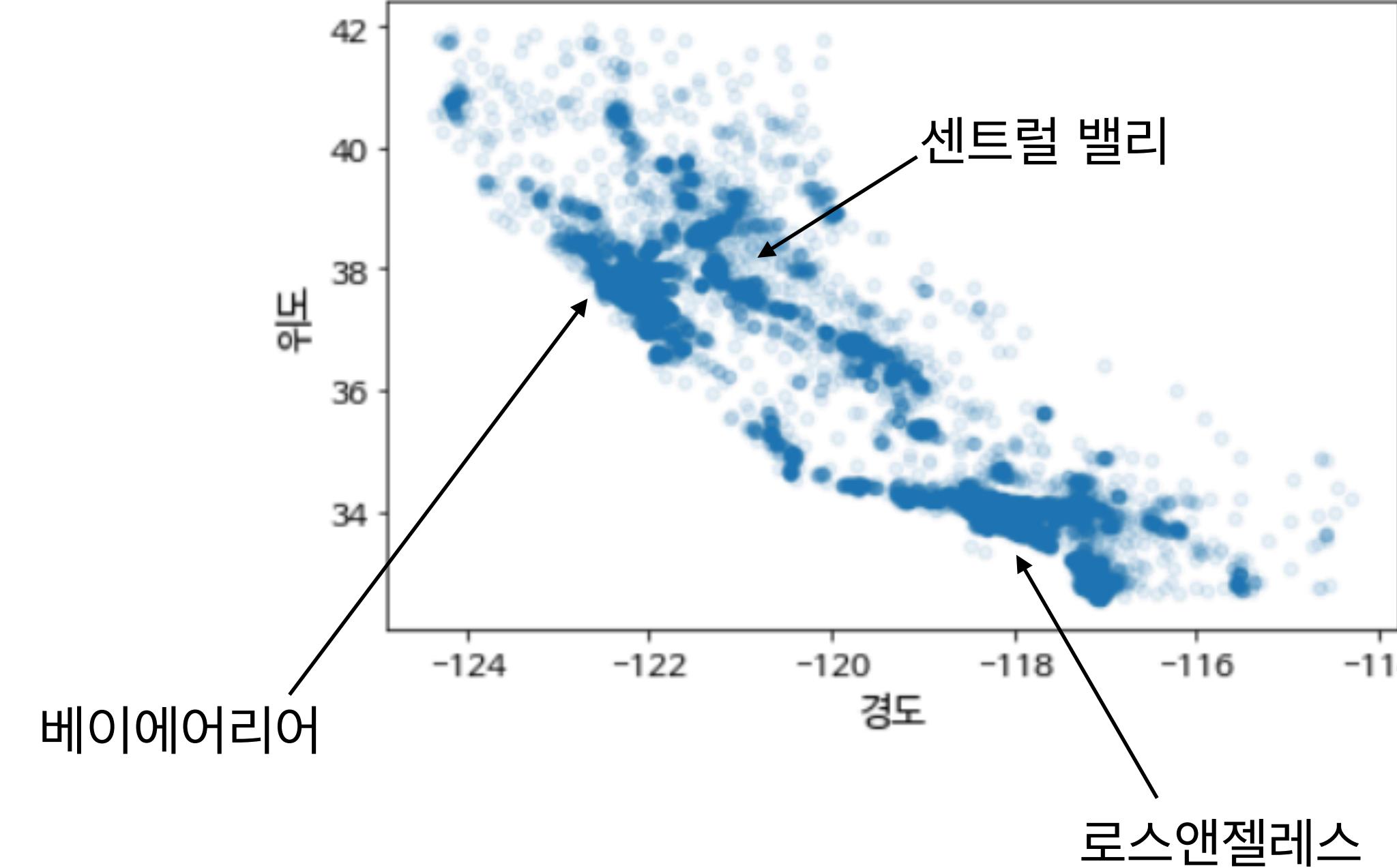
	전체	무작위 샘플링	계층 샘플링	무작위 샘플링 오류율	계층 샘플링 오류율
1.0	0.039826	0.040213	0.039738	0.973236	-0.219137
2.0	0.318847	0.324370	0.318876	1.732260	0.009032
3.0	0.350581	0.358527	0.350618	2.266446	0.010408
4.0	0.176308	0.167393	0.176399	-5.056334	0.051717
5.0	0.114438	0.109496	0.114369	-4.318374	-0.060464

# scatter plot

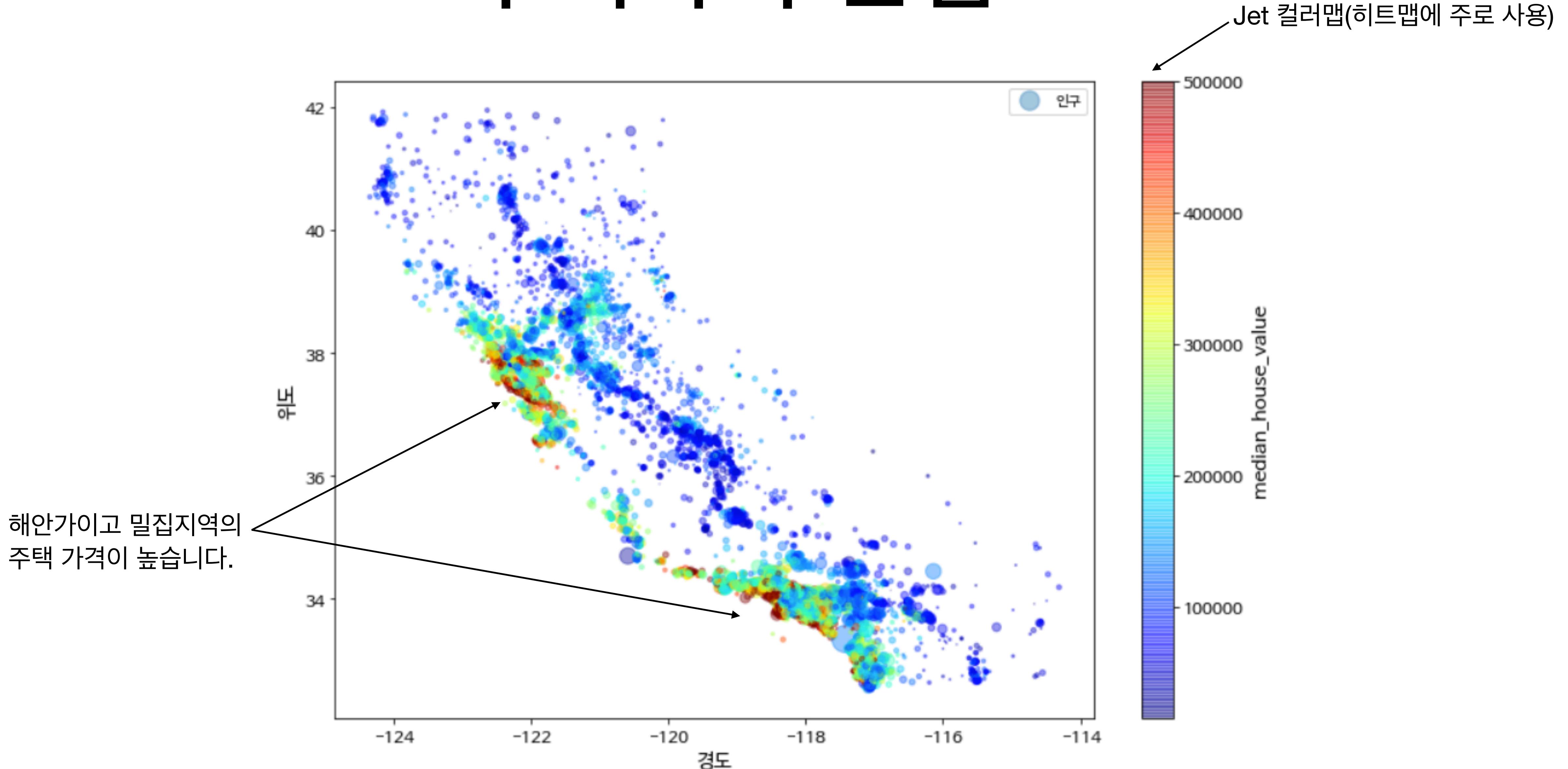
```
ax = housing.plot(kind="scatter", x="longitude", y="latitude")
ax.set(xlabel='경도', ylabel='위도')
save_fig("bad_visualization_plot")
```



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

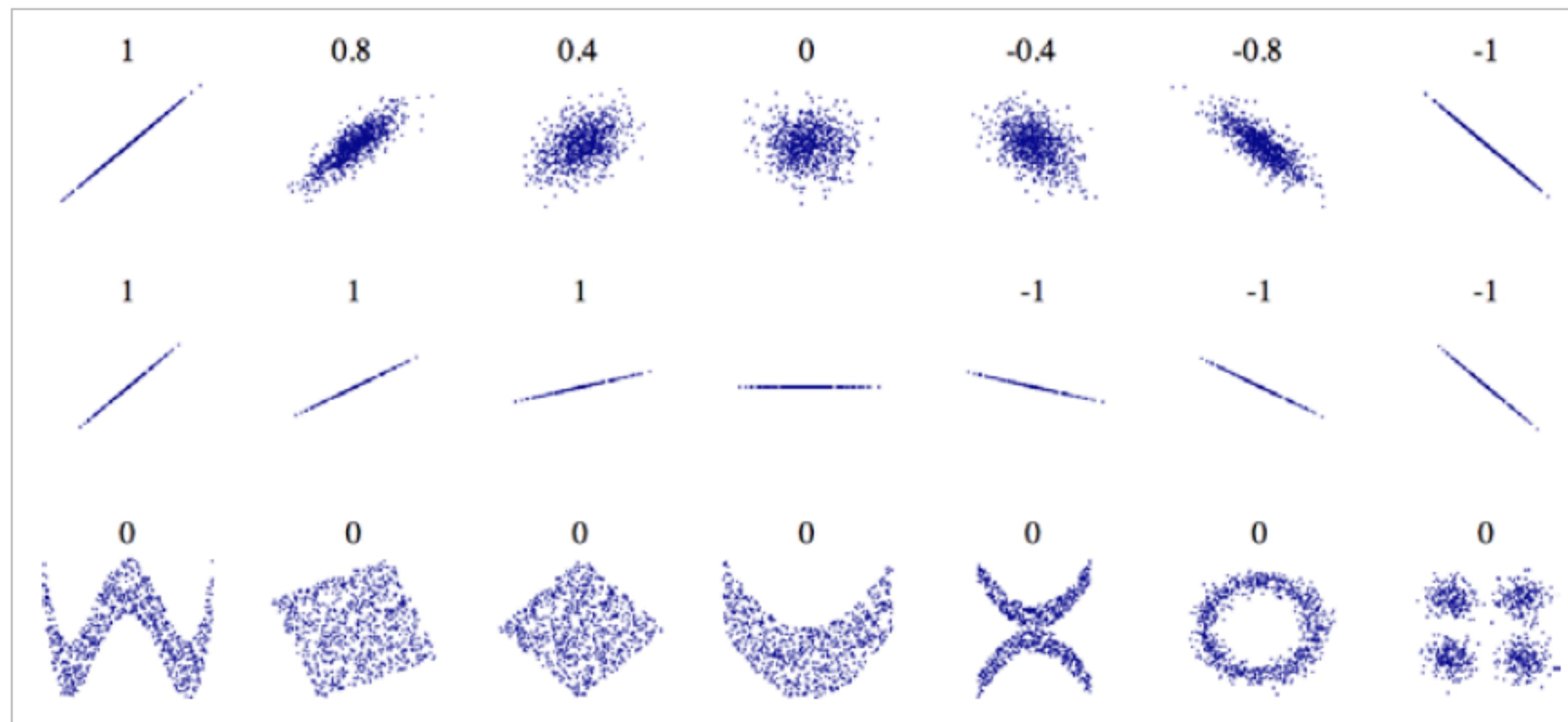


# 주택가격 산점도



# Pearson's r

- 선형 상관관계를 나타내며 기울기와 상관없습니다.



# df.corr()

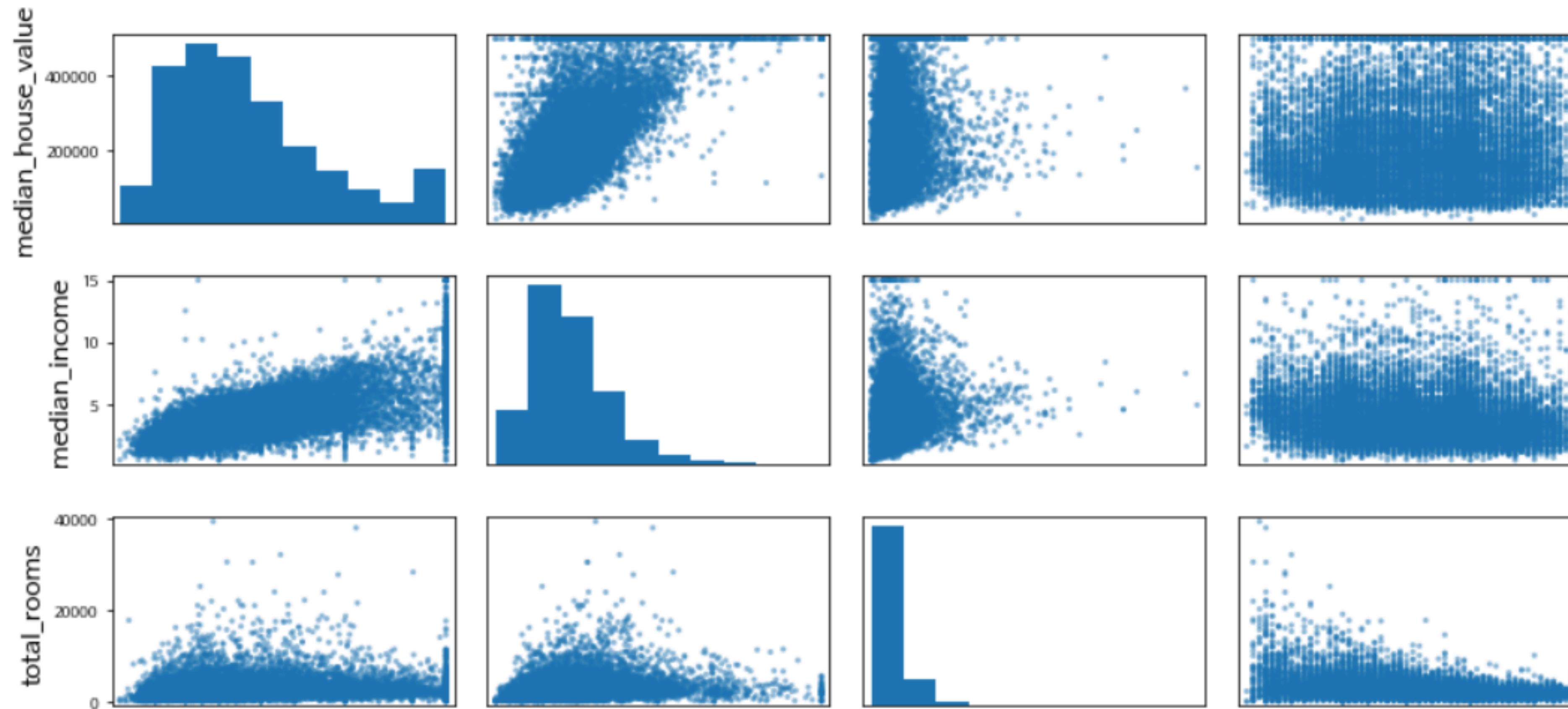
```
corr_matrix = housing.corr()
```

```
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
```

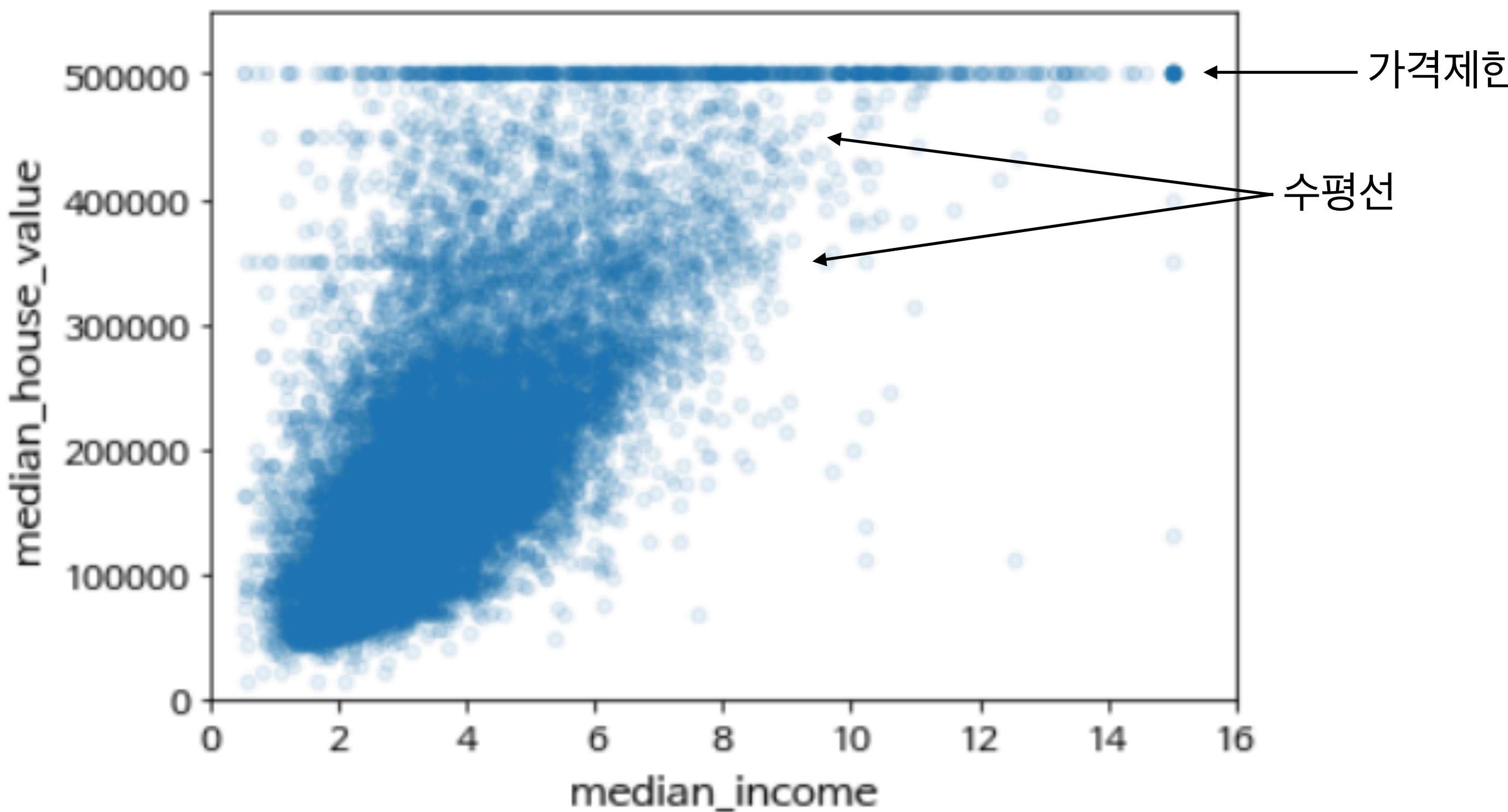
# scatter\_matrix

```
from pandas.plotting import scatter_matrix  
  
attributes = ["median_house_value", "median_income", "total_rooms",  
              "housing_median_age"]  
scatter_matrix(housing[attributes], figsize=(12, 8)) ← diagonal='hist'(default) or 'kde'  
save_fig("scatter_matrix_plot")
```



# 중간 주택 가격 vs 중간 소득

```
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")
```



# 특성 조합

```
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" ]
```

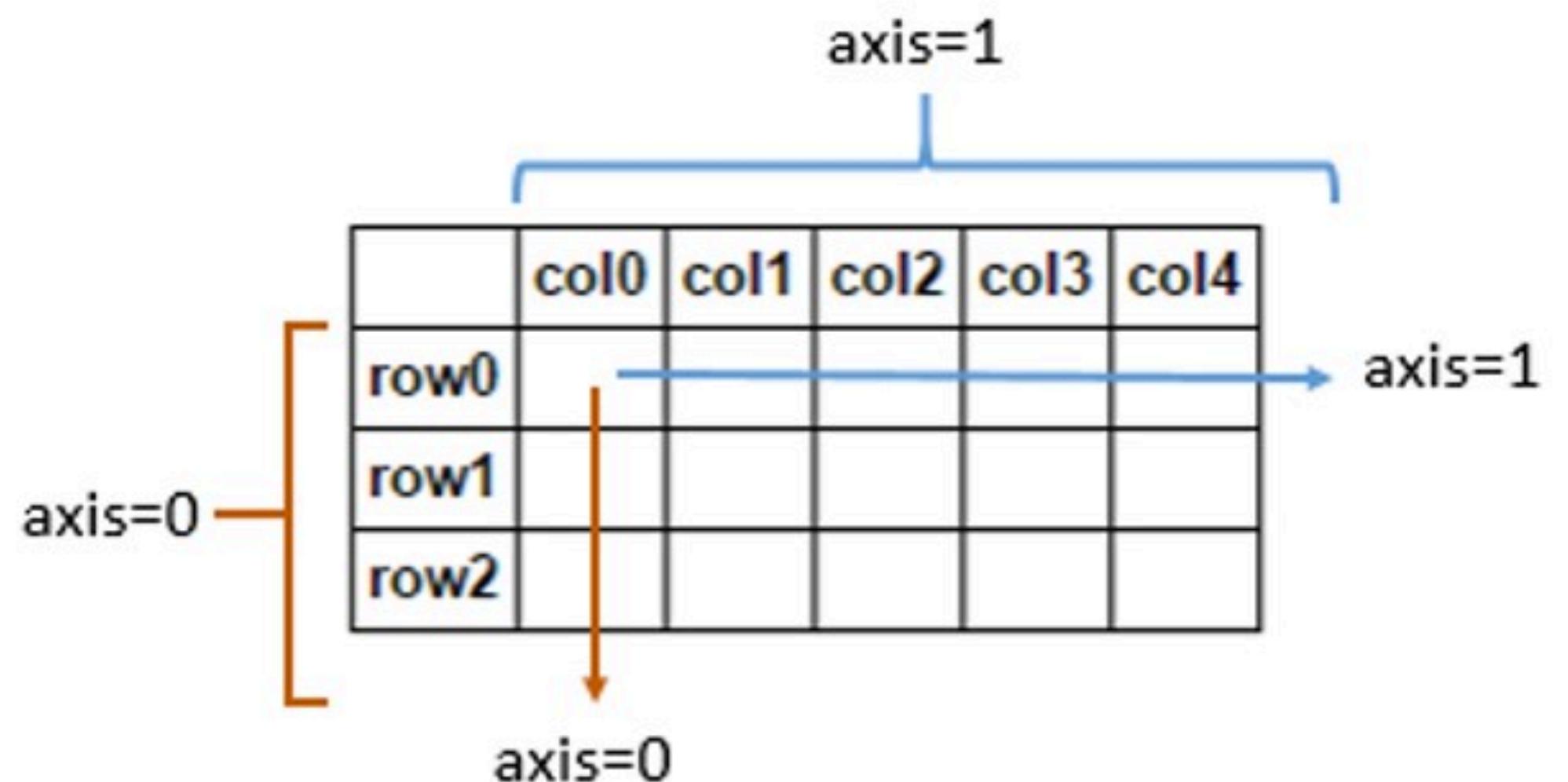
가구당 방 개수  
방당 침대수  
가구당 인원

```
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

# 데이터 준비를 위한 함수

- 데이터 변환을 손쉽게 반복할 수 있습니다.
- 다른 프로젝트에 재사용할 수 있습니다.
- 론칭 후에 새 데이터에 적용할 때 사용합니다.
- 최적의 조합을 찾는데 편리합니다.



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

# 빠진 값이 있는 total\_bedrooms

- 옵션 1: 해당 구역 제거, 옵션 2: 전체 특성 삭제, 옵션 3: 대체(0, 평균, 중간값 등)
- 옵션 1

```
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
4629	-118.30	34.07		18.0	3759.0	NaN	3296.0	1462.0	2.2708 <1H
6068	-117.86	34.01		16.0	4632.0	NaN	3038.0	727.0	5.1762 <1H
17923	-121.97	37.35		30.0	1955.0	NaN	999.0	386.0	4.6328 <1H
13656	-117.30	34.05		6.0	2155.0	NaN	1039.0	391.0	1.6675
19252	-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
-----------	----------	--------------------	-------------	----------------	------------	------------	---------------	---------------

# 옵션 2: 특성 삭제

```
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

# 옵션 3: 대체

저장

```
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

# with scikit-learn

하이퍼파라미터(클래스 매개변수)

```
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\_ ← 모델 파라미터(인스턴스 변수)

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

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

# scikit-learn Design

- 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 가능합니다.
- 클래스 남용 방지: 기본 데이터 타입으로 넘파이 배열을 사용합니다.
- 조합성: Pipeline 클래스
- 합리적 기본값: 모든 매개변수에 합리적인 기본값을 둡니다.

# 원-핫 인코딩 one-hot encoding

- 머신러닝 모델은 숫자만 처리할 수 있습니다.

The diagram illustrates the process of one-hot encoding. On the left, a table labeled "Color" contains five rows of data. The first column is numerical (1, 1, 2, 3, 2), and the second column is categorical (Red, Red, Yellow, Green, Yellow). A large yellow arrow points from this table to the right, indicating the transformation. On the right, another table shows the resulting one-hot encoding. It has three columns: "Red", "Yellow", and "Green". The rows correspond to the data in the first table. The "Red" column has values [1, 1, 0, 0, 0], the "Yellow" column has values [0, 0, 1, 0, 1], and the "Green" column has values [0, 0, 0, 1, 0].

Color	
1	Red
1	Red
2	Yellow
3	Green
2	Yellow

	Red	Yellow	Green
1	1	0	0
1	1	0	0
2	0	1	0
3	0	0	1
2	0	1	0

# ocean\_proximity 변환

- 범주형(텍스트) 데이터를 변환하는 방법(원-핫 인코딩one-hot encoding)
  - df.factorize() –> sklearn.preprocessing.OneHotEncoder
  - pd.get\_dummies()
  - scikit-learn's new OneHotEncoder (v0.20 before year-end)
- LabelEncoder는 텍스트 범주 y를 위한 정수 인코딩 <–> OrdinalEncoder는 텍스트 범주 X를 위한 정수 인코딩 (v0.20)

# df.factorize()

```
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]
```

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

```
housing_categories
```

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

# OneHotEncoder

2차원 배열로 변경

```
from sklearn.preprocessing import OneHotEncoder  
  
encoder = OneHotEncoder()  
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
```

```
housing_cat_1hot.toarray()
```

```
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]])
```

# pd.get\_dummies()

- 모든 문자열 특성을 원-핫 인코딩으로 변환합니다.

```
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

# new OneHotEncoder

- scikit-learn 0.20 버전에 추가될 예정(sklearn.preprocessing.OneHotEncoder)

```
from future_encoders import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)

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.]])
```

# 나만의 변환기

- Pipeline 클래스와 연계할 수 있음.
- fit()-self 반환, transform(), fit\_transform()을 구현한 파이썬 클래스 (duck typing)
- TransformerMixin을 상속하면 fit\_transform() 메서드가 제공됩니다.
- BaseEstimator를 상속하면 get\_params(), set\_params() 메서드가 제공됩니다.
- 클래스 생성자(\_\_init\_\_)에 \*args, \*\*kargs를 사용하면 안됩니다.

# CombinedAttributesAddr

```
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 **kargs
        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)
```

합리적 기본값

가구당 방 개수

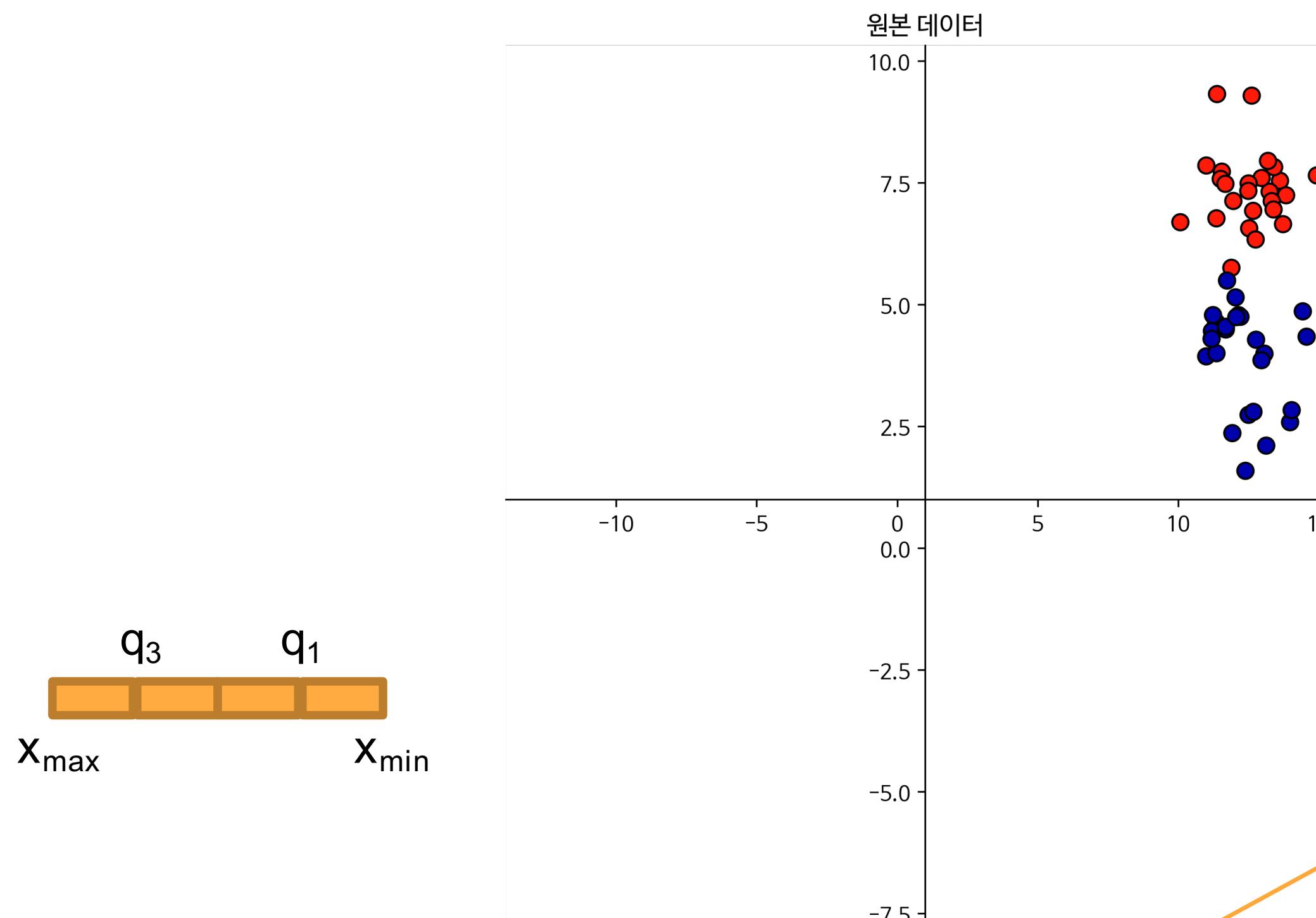
가구당 인원

방당 침대수

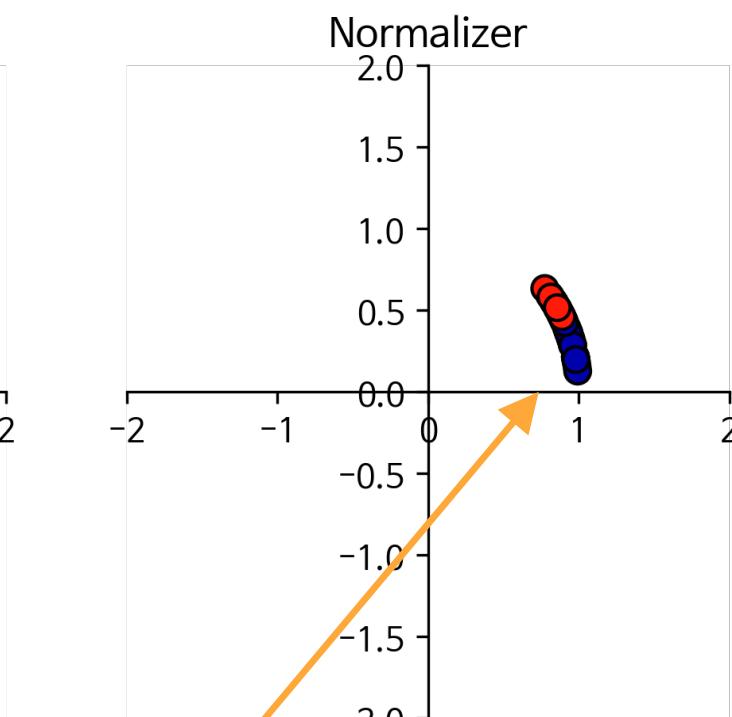
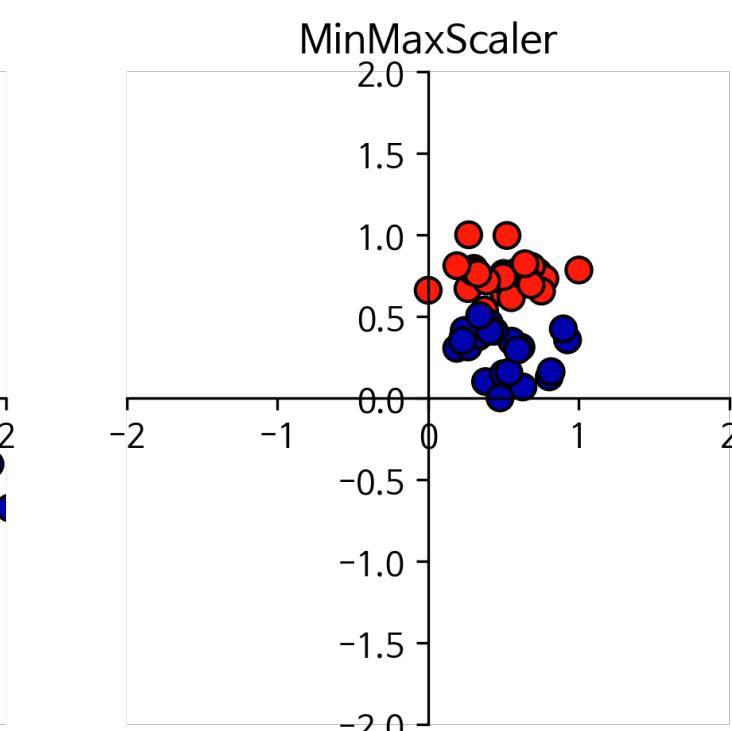
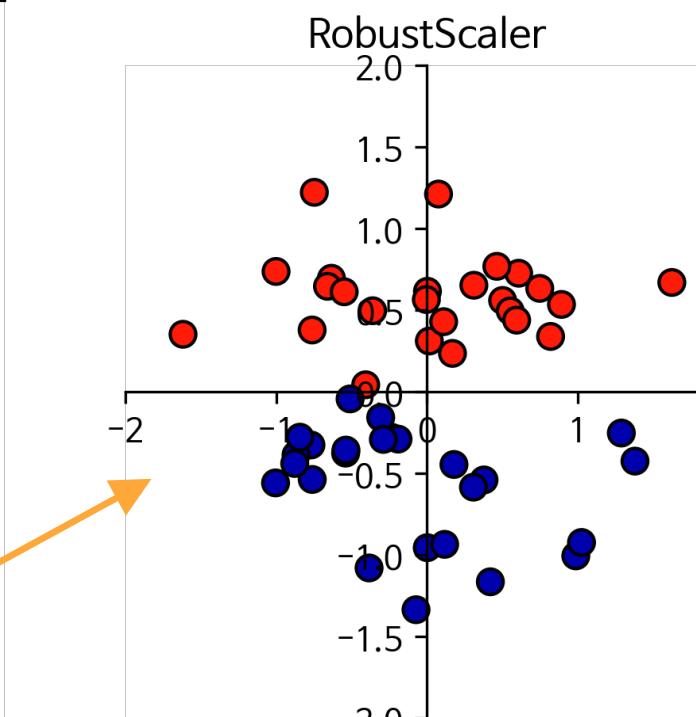
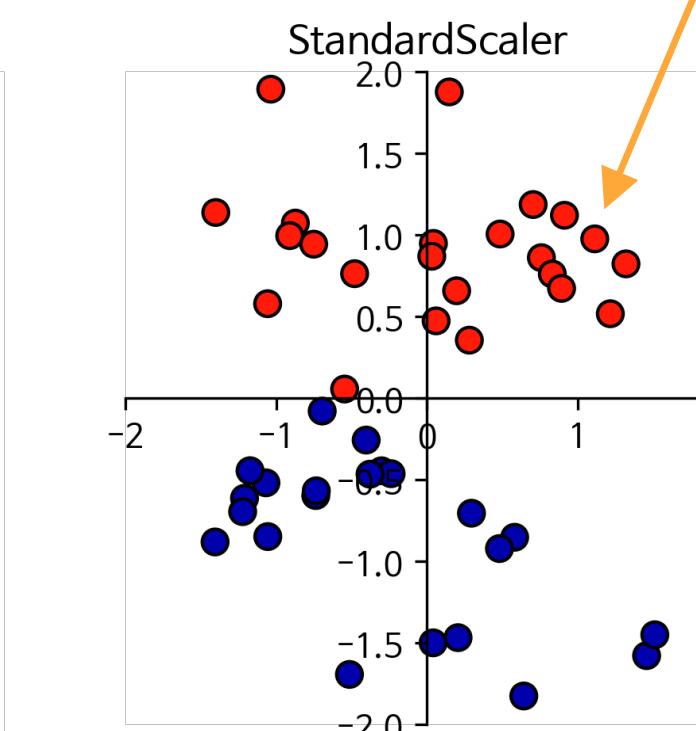
넘파이

# 특성 스케일링

$$\frac{x - \bar{x}}{\sigma} \quad \text{표준점수, z-점수: 평균 } 0, \text{ 분산 } 1$$



$\frac{x - q_2}{q_3 - q_1}$  사분위값 사용, 이상치에 덜 민감



2차원일 때는 원  
(3차원은 구)

$$\frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

모든 특성이  
0과 1사이에 위치  
이상치에 민감

Normalizer(norm='l2')  
다른 방법과 달리  
행(포인트) 별로 적용  
(유clidean 거리를 1로 만듦)

$$\frac{X}{\sqrt{x_1^2 + x_2^2}}$$

# 전처리 단계 연결

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

num_pipeline = Pipeline([
    ('imputer', Imputer(strategy="median")),
    ('atribbs_adder', CombinedAttributesAdder()), ←———— 튜플 리스트
    ('std_scaler', StandardScaler()),
])
housing_num_tr = num_pipeline.fit_transform(housing_num)           Imputer.fit_transform()->
                                                                    CombinedAttributesAddr.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)
```

# 나만의 변환기 2

- 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
```

# 숫자와 문자열을 위한 파이프라인

컬럼 이름 리스트

```
num_attrs = list(housing_num)
cat_attrs = ["ocean_proximity"]

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

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

v0.20 버전

num\_pipeline.steps

```
[('selector',
  DataFrameSelector(attribute_names=['longitude', 'latitude', 'rooms', '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))]
```

# 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

# 모델 선택

- 선형 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)
```

```
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

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

낮을수록 좋음

housing\_labels.mean(), housing\_labels.std()

```
lin_reg.score(housing_prepared, housing_labels)
```

(206990.9207243217, 115703.01483031621)

0.6481624842804428

R<sup>2</sup> 높을수록 좋음. 과소적합

# 모델 선택

- 결정 트리

```
from sklearn.tree import DecisionTreeRegressor  
  
tree_reg = DecisionTreeRegressor(random_state=42)  
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

```
tree_reg.score(housing_prepared, housing_predictions)
```

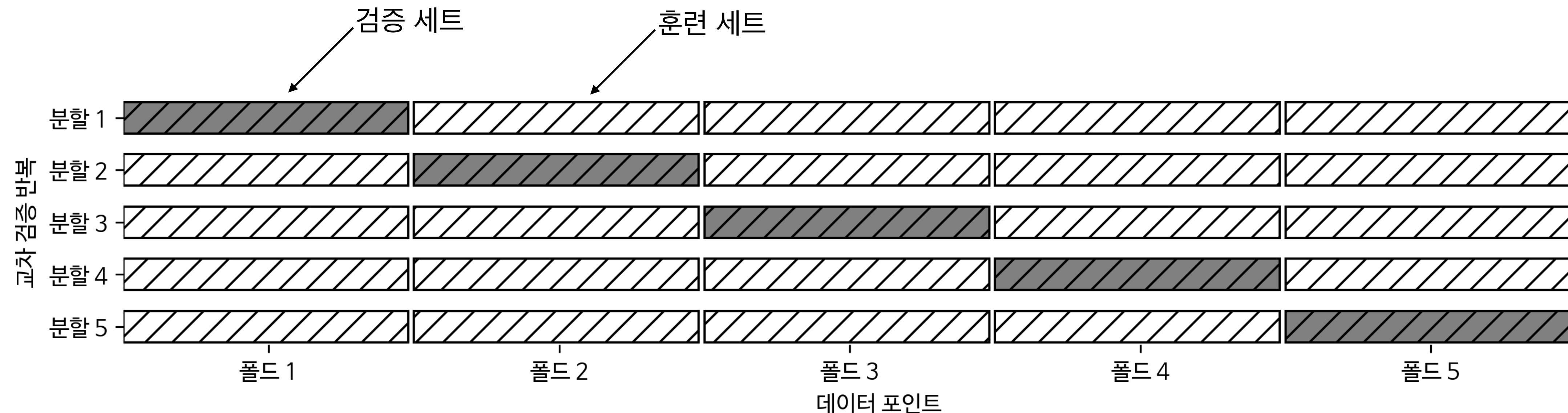
1.0



R<sup>2</sup> 과대적합

# 교차검증을 사용한 평가

- 훈련 세트 중 일부를 사용하여 검증에 사용합니다(검증 세트 혹은 개발 세트)



# cross\_val\_score

- 정확한 모델 평가 점수

```
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 ← 68628

표준편차: 2731.6740017983466

```
lin_r2_scores = cross_val_score(lin_reg, housing_prepared, housing_labels, cv=10)  
display_scores(lin_r2_scores)
```

점수: [0.62810772 0.67366246 0.62569782 0.59558993 0.65206084 0.63573031  
0.68070897 0.66309703 0.6262189 0.64803579]  
평균: 0.6428909748300279 ← 0.648  
표준편차: 0.02444728212205088

# 랜덤포레스트

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(random_state=42)
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
```

21941.911027380233

```
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)
```

점수: [51650.94405471 48920.80645498 52979.16096752 54412.74042021  
50861.29381163 56488.55699727 51866.90120786 49752.24599537  
55399.50713191 53309.74548294]  
평균: 52564.19025244012 ← 69052 ~ 0  
표준편차: 2301.873803919754

# 모델 튜닝

- 모든 모델을 시도해 볼 수는 없습니다.
- 가능성 있는 몇 개의 모델을 고른 뒤 최적의 하이퍼파라미터를 찾아야 합니다.

```
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)
```

# 최적 모델

```
grid_search.best_params_
```

```
{'max_features': 8, 'n_estimators': 30} -----> 46694
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features=8, max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=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=42,
                      verbose=0, warm_start=False)
```

그리드서치에서 테스트한 하이퍼파라미터 조합의 점수를 확인합니다:

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

```
63647.85444595992 {'max_features': 2, 'n_estimators': 3}
55611.50159876327 {'max_features': 2, 'n_estimators': 10}
53370.06407363344 {'max_features': 2, 'n_estimators': 30}
60959.138858487866 {'max_features': 4, 'n_estimators': 3}
52740.58416665252 {'max_features': 4, 'n_estimators': 10}
50374.14214614731 {'max_features': 4, 'n_estimators': 30}
58661.2866461823 {'max_features': 6, 'n_estimators': 3}
52000.073070776026 {'max_features': 6, 'n_estimators': 10}
```

# 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)
```

rvs() 메서드를 지원해야 합니다.

```
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}
```

# 테스트 세트 평가

- 마지막에 딱 한번만 수행합니다.
- 테스트 세트를 반복하여 사용하면 테스트 세트에 과대적합된 모델을 만듭니다.

```
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

# 전체 파이프라인

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

```
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])
```

# 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

- 실시간 성능 체크를 위한 모니터링 코드 개발
- 사람의 성능 평가(크라우드 소싱 등)
- 입력 데이터 모니터링 코드 개발
- 정기적인 훈련을 위한 자동화
- 생각보다 전체 프로세스에 투여되는 시간이 많습니다!

**감사합니다**