KUCC - 머신러닝 & 딥러닝 입문

머신러닝&딥러닝 세션

2주차

| 머신러닝 훈련 (1)

참고 사항

| 본 교안은 '핸즈온 머신러닝' 내용을 참고하고, 정리한 것입니다.

| GitHub 코드 링크는 아래와 같으며, 더 많은 공부를 원하시는 분들은 해당 링크에서 다른 내용에 대해 깊게 공부할 수 있습니다.

| Icon 사용 출처 : https://www.Flaticon.com/kr

[핸즈온 머신러닝 깃허브] https://github.com/ExcelsiorCJH/Hands-On-ML

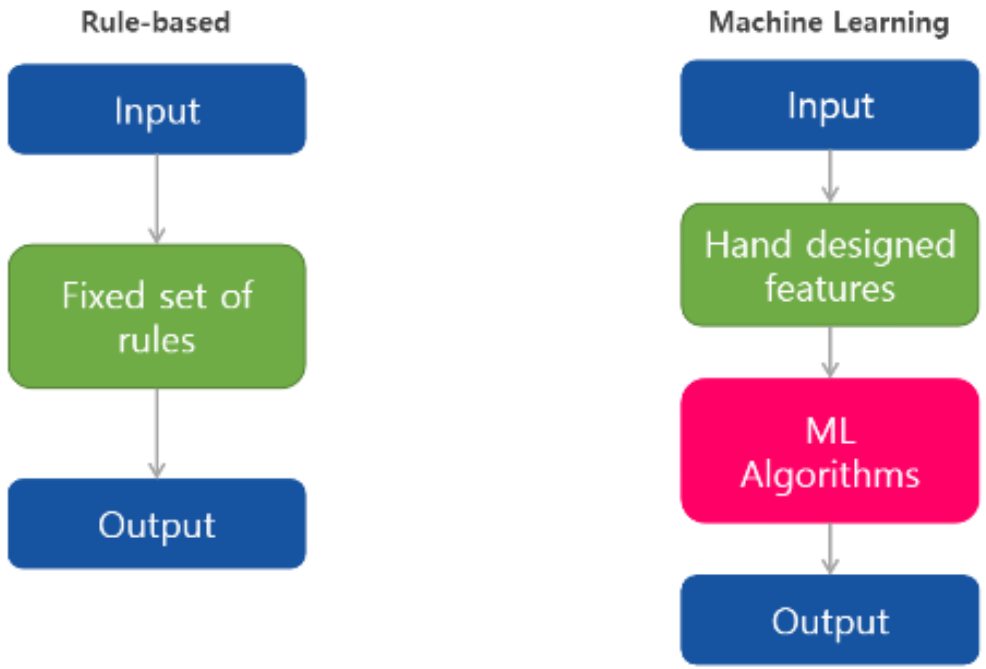
[Ratsgo님 깃허브] https://ratsgo.github.io/blog/

목차

머신 러닝의 개념

1.2 왜 머신러닝을 사용하는가?

- 머신러닝 모델을 통해 코드를 간단하고 더 잘 수행할 수 있도록 함
- 머신러닝 기법으로 복잡한 문제를 해결할 수 있음
- 새로운 데이터에 대해 유동적으로 적응할 수 있음



머신 러닝의 개념

1.3 머신 러닝의 시스템과 종류

| 사람의 감독 하에 훈련하는 것인가?
▶ 지도/비지도/준 지도/강화 학습

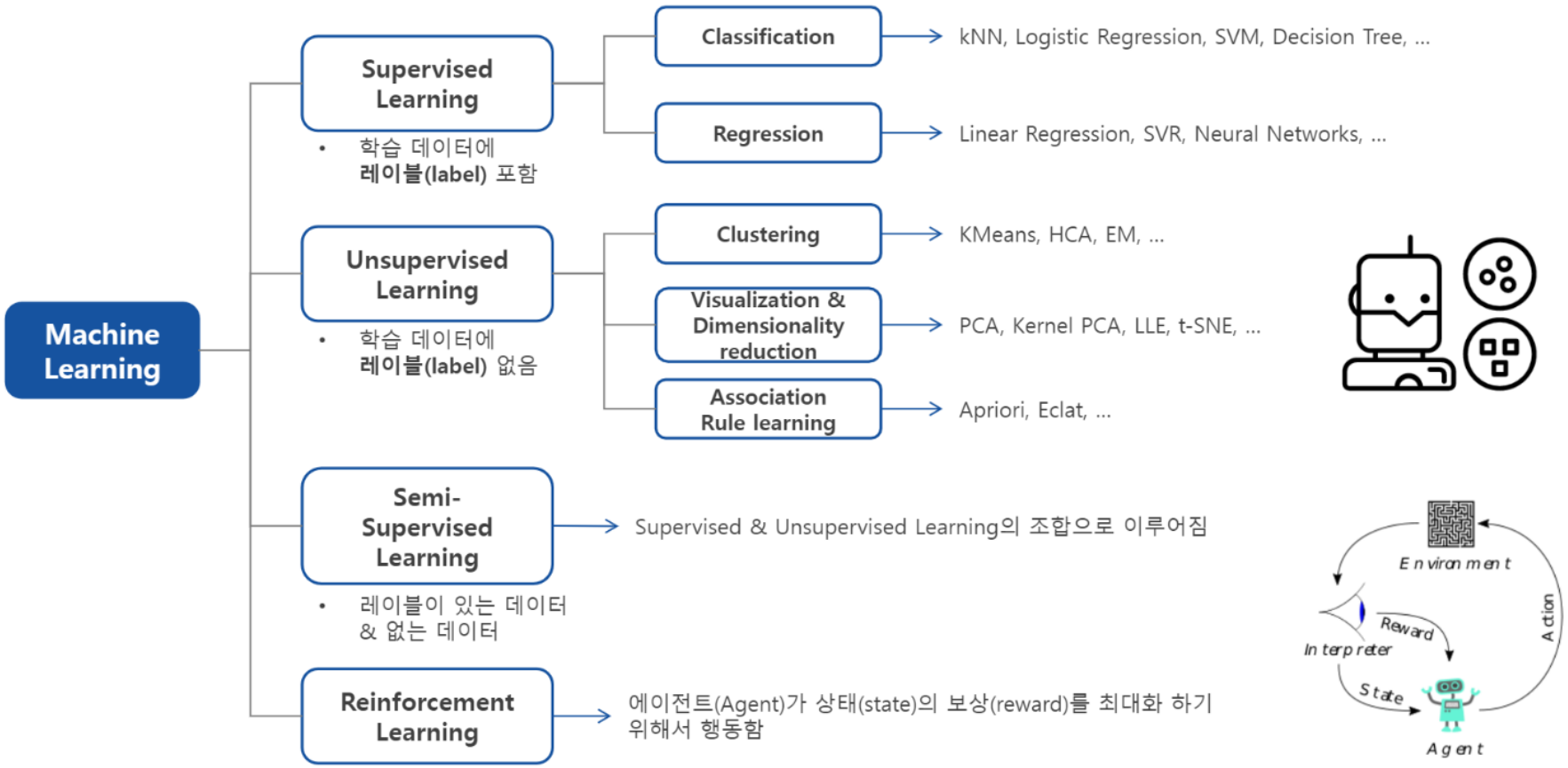
| 실시간으로 점진적인 학습을 하는 것인지
▶ 온라인/배치 학습

| 단순한 데이터 비교인지, 패턴을 발견하는 모델을 만드는 지
▶ 사례/모델 기반 학습

머신 러닝의 개념

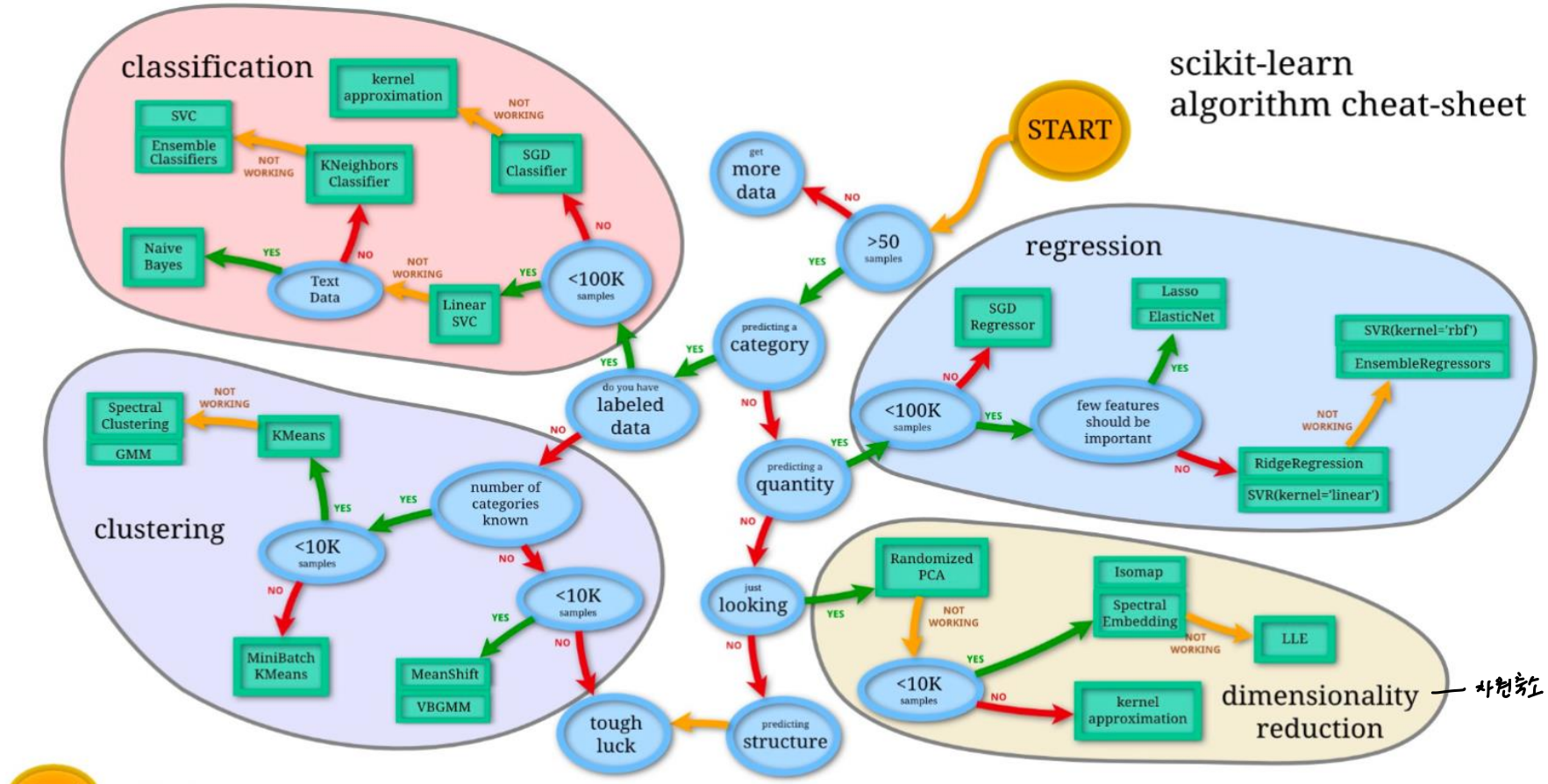
1.3.1 지도 학습과 비지도 학습

- '학습하는 동안의 감독 형태나 정보량'에 따라 머신러닝 종류를 분류



머신 러닝의 개념

<머신러닝의 종류>



차원축소

머신 러닝의 개념

1.3.2 배치 학습과 온라인 학습

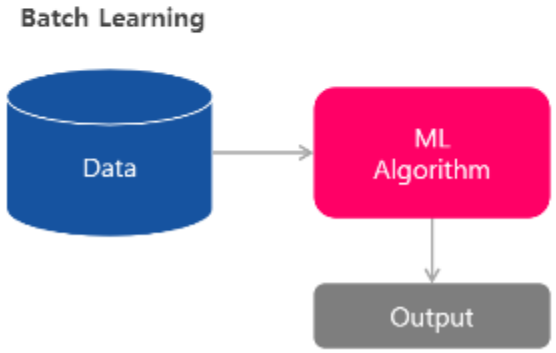
- 입력 데이터의 스트림stream 으로 부터 점진적으로 학습할 수 있는지 여부로 분류

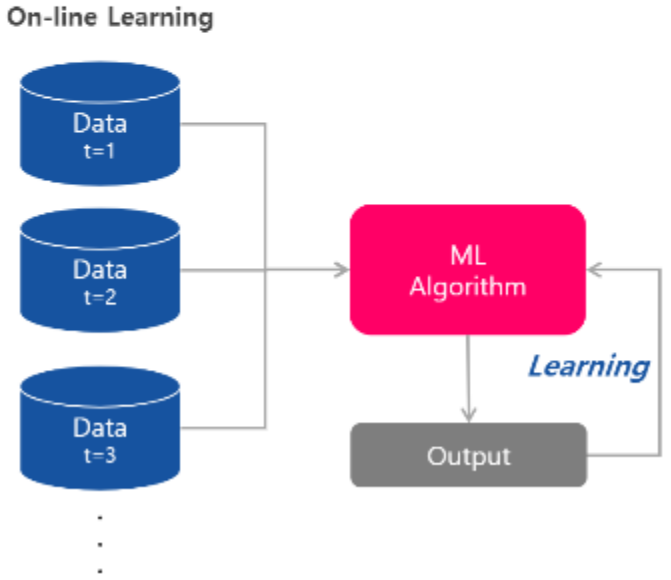
배치 학습 (Batch learning) : 점진적으로 학습X

- 점진적으로 학습할 수 없으며, 데이터를 모두 사용해 훈련 시킴
- 주로 오프라인에서 수행 → ***offline learning***

온라인 학습 (Online learning) : 점진적으로 학습

- 데이터를 순차적 또는 작은 묶음 단위(미니 배치)로 학습
- 빠른 변화에 적응해야 하는 시스템에 적합(ex. 주식가격)





머신 러닝의 개념

1.3.3 사례 기반 학습과 모델 기반 학습

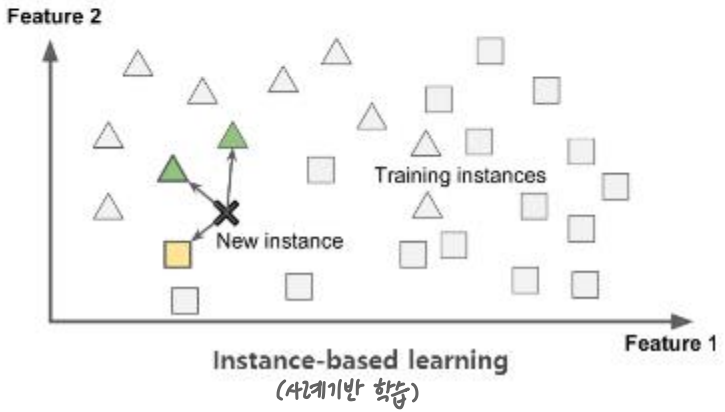
- 어떻게 **일반화** 되는가에 따라 분류
- **일반화**란 학습데이터가 아닌 새로운 데이터를 예측하는데 전반적으로 잘 예측할 수 있도록 하는 것을 말함

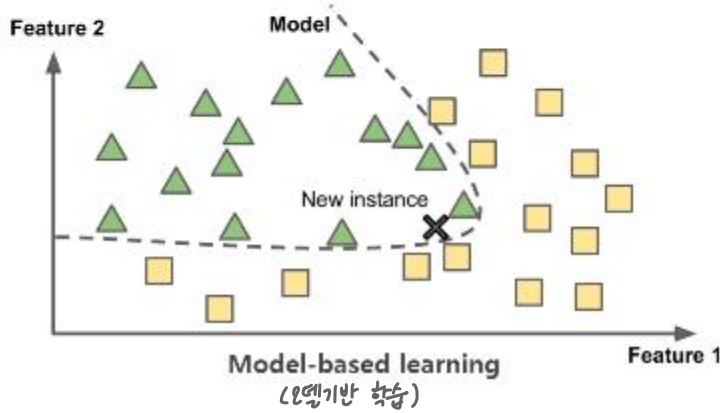
① 사례 기반 학습 (Instance-based learning)

- 이미 알고 있는 데이터와 새로운 데이터 간의 유사한 정도를 통해 예측하는 방법
- 대표적인 알고리즘으로는 k-NN(k-Nearest Neighbors)가 있음

② 모델 기반 학습 (Model-based learning)

- 학습 데이터로부터 일반화 할 수 있는 모델을 만들어 **예측**하는 방법





머신 러닝의 개념

1.4 머신 러닝의 주요 도전 과제 : Data

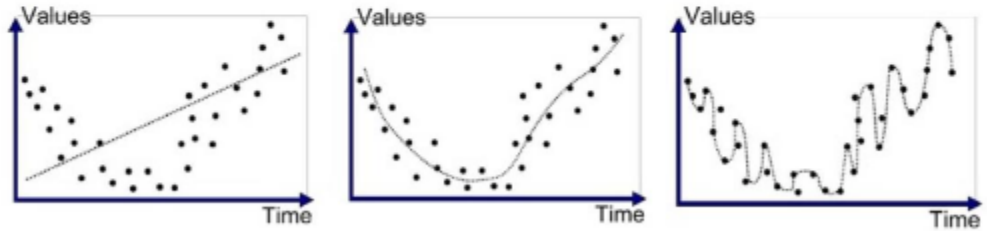
- 머신 러닝 Algorithm이 잘 작동하기 위해서는 Data가 많아야 함
  - ▶ 빅데이터 기반의 학습
- 일반화가 잘 되려면 새로운 데이터를 학습데이터가 잘 대표하여야 함
  - ▶ 학습데이터의 정규성/보편성/등분산성
- 학습 데이터 정제에 많은 시간을 투자해야 함
  - ▶ 학습 데이터에 존재하는 에러(이상치, 영향치, 노이즈) 제거 필요
- 학습에 사용할 좋은 특성을 찾는 것이 중요 (by using Feature Engineering)
  - ▶ 특성 선택 (feature selection) : 가지고 있는 특성 중 유용한 특성 선택
  - ▶ 특성 추출 (feature extraction) : 특성을 결합하여 유용한 특성 생성

머신 러닝의 개념

1.4 머신러닝의 주요 도전 과제 - 알고리즘

- 머신러닝 알고리즘이 학습데이터에 **과대적합(overfitting)** 되지 않아야 한다.
  - 파라미터 수가 적은 모델을 선택(고차원 다항 모델 보다 선형모델)
  - 학습 데이터에서 특성 수를 줄이거나, 모델에 제약을 가함
  - 학습 데이터를 더 많이 모은다.
  - 학습 데이터의 노이즈를 줄인다.

- 또한, **과소적합(underfitting)** 되지 않아야 한다.
  - 파라미터가 더 많은 모델을 선택
  - 더 좋은 특성을 추가
  - 모델의 제약을 줄임



Underfitted | Good Fit/Robust | Overfitted

머신 러닝의 개념

1.5 정리

| 머신 러닝은 명시적인 규칙을 코딩하는 것이 아니다. 기계가 데이터로부터 학습하여, 특정 작업을 잘 하도록 만드는 것이다.

| 머신 러닝은 지도/비지도 학습, 배치/온라인 학습, 사례/모델 기반 학습으로 분류 가능하다.

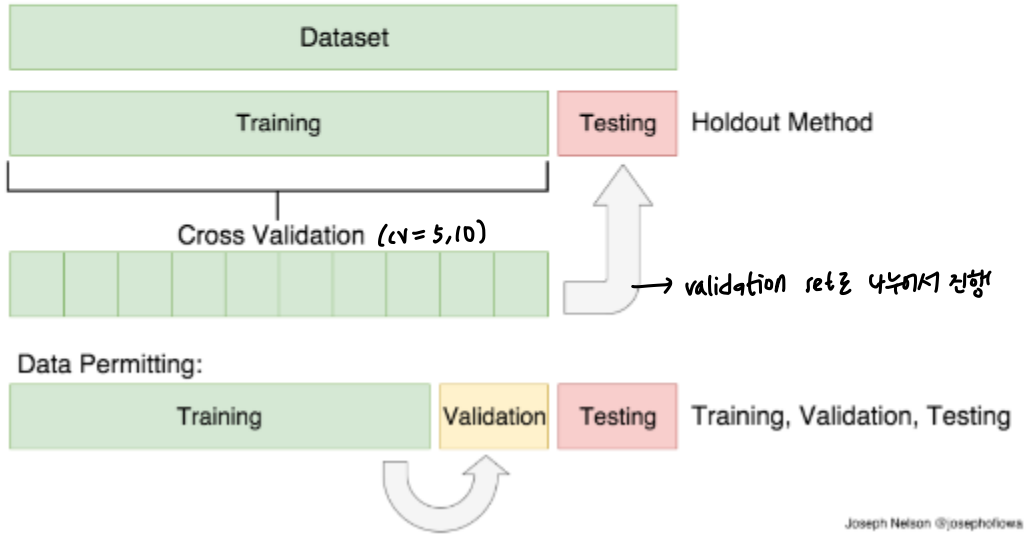
| 머신 러닝은 훈련(학습) 세트에 데이터를 모아 학습 알고리즘에 넣어준다.
▶ 모델 기반 : 훈련 세트에 모델을 맞추기 위해 파라미터 조정
▶ 사례 기반 : 학습 데이터를 기억하고, 새로운 데이터를 일반화하기 위해 유사도 측정

| 훈련 세트가 많고, 훈련 세트가 새로운 데이터를 대표할 수 있으며, 노이즈가 별로 없는 데이터로 학습을 진행해야 함.

머신 러닝의 개념

1.6 테스트와 검증

- **학습 세트**(Train Set): 머신러닝 모델을 학습할 때 사용하는 데이터 셋 — 학습
- **검증 세트**(Validation Set): 모델학습에 필요한 하이퍼파라미터를 찾기위해 사용하는 데이터 셋 — 하이퍼파라미터 찾기
- **테스트 세트**(Test Set): 학습된 모델을 평가하기 위한 데이터 셋 — 평가



캘리포니아 주택 가격 예측!

1.5 정리

| 블록 그룹마다 중간소득/인구/중간 주택 가격 有

| 파이프라인 : 데이터 처리 컴포넌트의 연속

- 각 컴포넌트는 데이터를 추출해 목적에 맞게 처리

- 결과를 다른 컴포넌트나 저장소에 보내는 일련의 과정

1

데이터 가져오기(해킹?)

기본 Library Import

```
In [1]: import pandas as pd
        import numpy as np
        import os

        np.random.seed(42)

        import matplotlib.pyplot as plt
        import platform

        %matplotlib inline
```

라이브러리 설명

numpy
- ▶ 행렬(Matrix) 연산 관련 라이브러리
- ▶ 선형대수 구현 라이브러리!

pandas
- ▶ 데이터프레임(Table) 관련 라이브러리

matplotlib
- ▶ 시각화 관련 라이브러리(그래프)

os, platform…
- ▶ 운영체제, 시스템 정보 관련 라이브러리

기본 Library Import

시각화 위해 한글적용코드

```
path = 'c:/windows/Fonts/malgun.ttf'

from matplotlib import font_manager, rc

if platform.system() == 'Darwin':
    rc('font', family = 'AppleGothic')
elif platform.system() == 'Windows':
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
else:
    print('unknown system...sorry~~~')

plt.rcParams['axes.unicode_minus'] = False
```

mac 한정

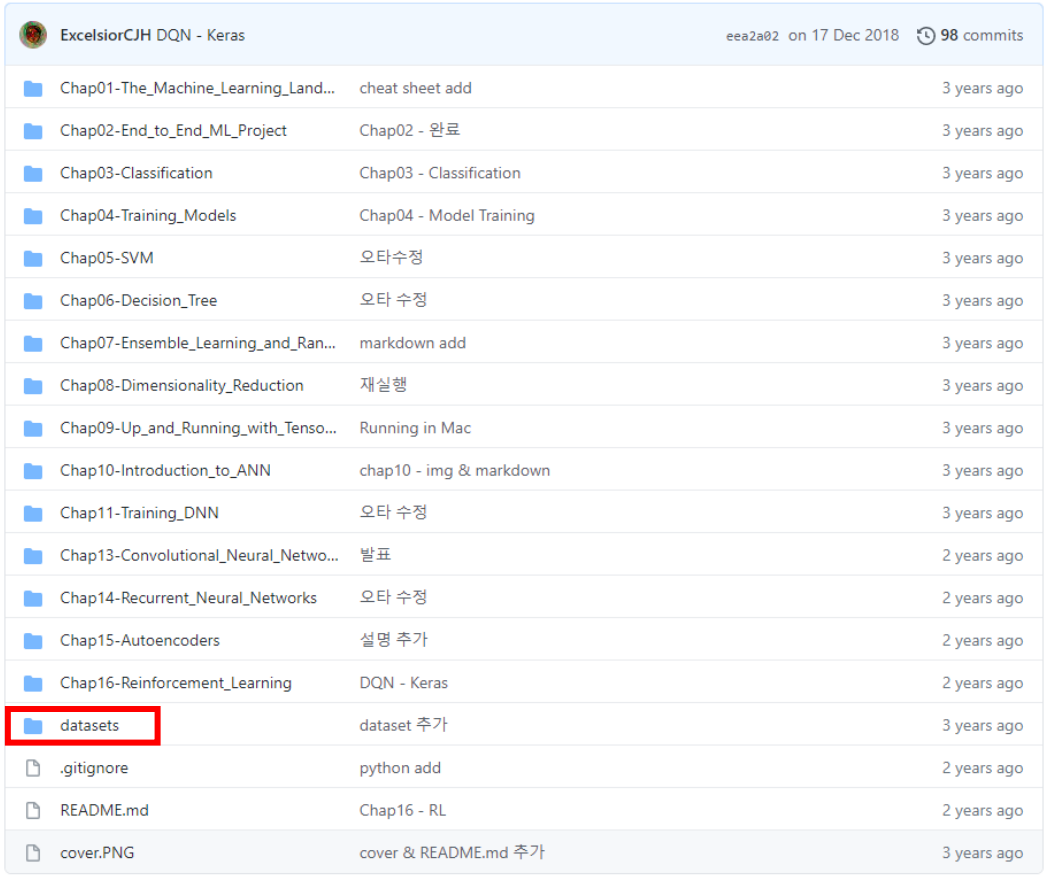
윈도우 한정

mac/윈도우가 아닌 경우

한글 적용 코드

Matplotlib에서 한글을 사용하려면 폰트 지정을 해줘야 합니다.

데이터 가져오기



ExcelsiorCJH DQN - Keras — eea2a02 on 17 Dec 2018 — 98 commits

Chap01-The_Machine_Learning_Land...	cheat sheet add	3 years ago
Chap02-End_to_End_ML_Project	Chap02 - 완료	3 years ago
Chap03-Classification	Chap03 - Classification	3 years ago
Chap04-Training_Models	Chap04 - Model Training	3 years ago
Chap05-SVM	오타수정	3 years ago
Chap06-Decision_Tree	오타 수정	3 years ago
Chap07-Ensemble_Learning_and_Ran...	markdown add	3 years ago
Chap08-Dimensionality_Reduction	재실행	3 years ago
Chap09-Up_and_Running_with_Tenso...	Running in Mac	3 years ago
Chap10-Introduction_to_ANN	chap10 - img & markdown	3 years ago
Chap11-Training_DNN	오타 수정	3 years ago
Chap13-Convolutional_Neural_Netwo...	발표	2 years ago
Chap14-Recurrent_Neural_Networks	오타 수정	2 years ago
Chap15-Autoencoders	설명 추가	2 years ago
Chap16-Reinforcement_Learning	DQN - Keras	2 years ago
datasets	dataset 추가	3 years ago
.gitignore	python add	2 years ago
README.md	Chap16 - RL	2 years ago
cover.PNG	cover & README.md 추가	3 years ago

데이터 가져오기

master ▾ Hands-On-ML / datasets / housing / Go to file

ExcelsiorCJH dataset 추가 c63a6dd on 26 Jul 2018 History

..		
README.md	dataset 추가	3 years ago
housing.csv	dataset 추가	3 years ago
housing.tgz	dataset 추가	3 years ago

California Housing

Source

This dataset is a modified version of the California Housing dataset available from Luís Torgo's page (University of Porto). Luís Torgo obtained it from the StatLib repository (which is closed now). The dataset may also be downloaded from StatLib mirrors.

This dataset appeared in a 1997 paper titled *Sparse Spatial Autoregressions* by Pace, R. Kelley and Ronald Barry, published in the *Statistics and Probability Letters* journal. They built it using the 1990 California census data. It contains one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

Tweaks

The dataset in this directory is almost identical to the original, with two differences:

- 207 values were randomly removed from the `total_bedrooms` column, so we can discuss what to do with missing data.
- An additional categorical attribute called `ocean_proximity` was added, indicating (very roughly) whether each block group is near the ocean, near the Bay area, inland or on an island. This allows discussing what to do with categorical data.

Note that the block groups are called "districts" in the Jupyter notebooks, simply because in some contexts the name "block group" was confusing.

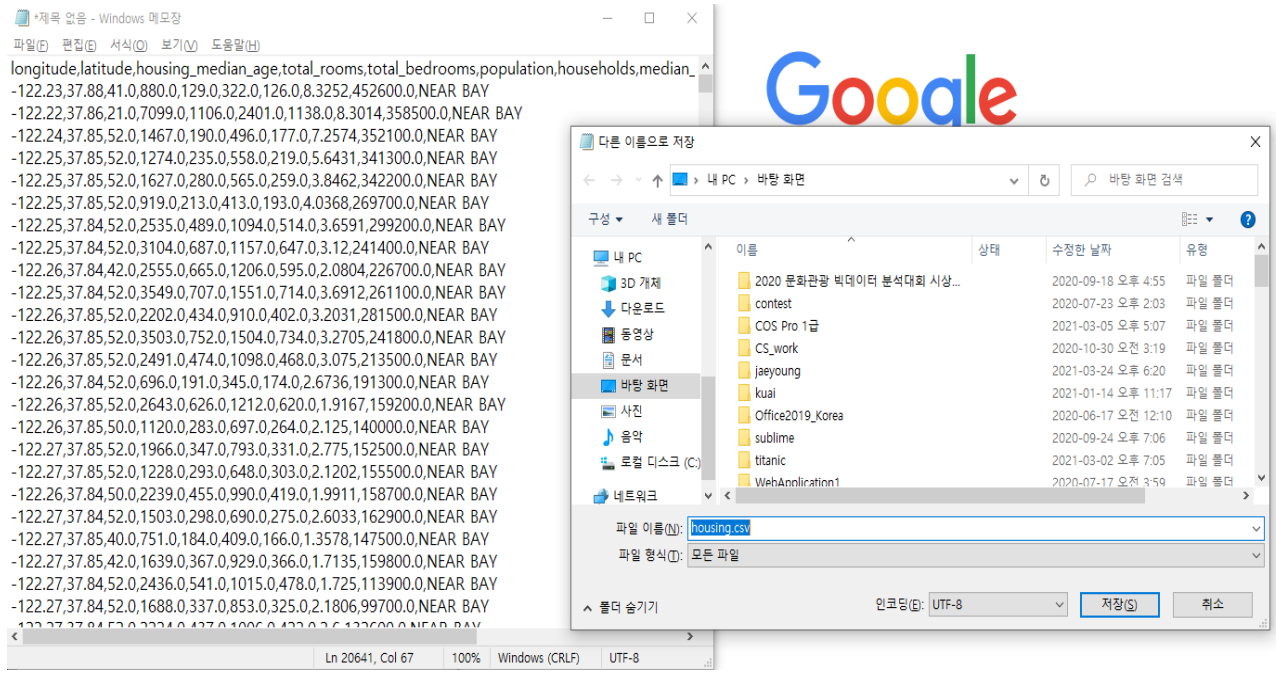
Data description

```
>>> 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

>>> housing["ocean_proximity"].value_counts()
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64

>>> housing.describe()
          longitude      latitude  housing_median_age   total_rooms  \
count  16513.000000  16513.000000        16513.000000  16513.000000
mean    -119.575972     35.639693           28.652335   2622.347605
std        2.002048      2.138279           12.576306   2138.559393
min     -124.350000     32.540000            1.000000      6.000000
25%     -121.800000     33.940000           18.000000   1442.000000
50%     -118.510000     34.260000           29.000000   2119.000000
75%     -118.010000     37.720000           37.000000   3141.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income
count    16355.000000  16513.000000  16513.000000   16513.000000
mean       534.885112   1419.525465    496.975050       3.875651
std        412.716467   1115.715084    375.737945       1.905088
min          2.000000      3.000000      2.000000       0.499900
25%        295.000000    784.000000    278.000000       2.566800
50%        433.000000   1164.000000    408.000000       3.541400
75%        644.000000   1718.000000    602.000000       4.745000
max       6210.000000  35682.000000   5358.000000      15.000100
```

데이터 가져오기

```
longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,median_house_value,ocean_pro
ity
-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
-122.25,37.85,52.0,919.0,213.0,413.0,193.0,4.0368,269700.0,NEAR BAY
-122.25,37.84,52.0,2535.0,489.0,1094.0,514.0,3.6591,299200.0,NEAR BAY
-122.25,37.84,52.0,3104.0,687.0,1157.0,647.0,3.12,241400.0,NEAR BAY
-122.26,37.84,42.0,2555.0,665.0,1206.0,595.0,2.0804,226700.0,NEAR BAY
-122.25,37.84,52.0,3549.0,707.0,1551.0,714.0,3.6912,261100.0,NEAR BAY
-122.26,37.85,52.0,2202.0,434.0,910.0,402.0,3.2031,281500.0,NEAR BAY
-122.26,37.85,52.0,3503.0,752.0,1504.0,734.0,3.2705,241800.0,NEAR BAY
-122.26,37.85,52.0,2491.0,474.0,1098.0,468.0,3.075,213500.0,NEAR BAY
-122.26,37.84,52.0,696.0,191.0,345.0,174.0,2.6736,191300.0,NEAR BAY
-122.26,37.85,52.0,2643.0,626.0,1212.0,620.0,1.9167,159200.0,NEAR BAY
-122.26,37.85,50.0,1120.0,283.0,697.0,264.0,2.125,140000.0,NEAR BAY
-122.27,37.85,52.0,1966.0,347.0,793.0,331.0,2.775,152500.0,NEAR BAY
-122.27,37.85,52.0,1228.0,293.0,648.0,303.0,2.1202,155500.0,NEAR BAY
-122.26,37.84,50.0,2239.0,455.0,990.0,419.0,1.9911,158700.0,NEAR BAY
-122.27,37.84,52.0,1503.0,298.0,690.0,275.0,2.6033,162900.0,NEAR BAY
-122.27,37.85,40.0,751.0,184.0,409.0,166.0,1.3578,147500.0,NEAR BAY
-122.27,37.85,42.0,1639.0,367.0,929.0,366.0,1.7135,159800.0,NEAR BAY
-122.27,37.84,52.0,2436.0,541.0,1015.0,478.0,1.725,113900.0,NEAR BAY
-122.27,37.84,52.0,1688.0,337.0,853.0,325.0,2.1806,99700.0,NEAR BAY
-122.27,37.84,52.0,2224.0,437.0,1006.0,422.0,2.6,132600.0,NEAR BAY
-122.28,37.85,41.0,535.0,123.0,317.0,119.0,2.4038,107500.0,NEAR BAY
-122.28,37.85,49.0,1130.0,244.0,607.0,239.0,2.4597,93800.0,NEAR BAY
-122.28,37.85,52.0,1898.0,421.0,1102.0,397.0,1.808,105500.0,NEAR BAY
-122.28,37.84,50.0,2082.0,492.0,1131.0,473.0,1.6424,108900.0,NEAR BAY
-122.28,37.84,52.0,729.0,160.0,395.0,155.0,1.6875,132000.0,NEAR BAY
-122.28,37.84,49.0,1916.0,447.0,863.0,378.0,1.9274,122300.0,NEAR BAY
-122.28,37.84,52.0,2153.0,481.0,1168.0,441.0,1.9615,115200.0,NEAR BAY
-122.27,37.84,48.0,1922.0,409.0,1026.0,335.0,1.7969,110400.0,NEAR BAY
-122.27,37.83,49.0,1655.0,366.0,754.0,329.0,1.375,104900.0,NEAR BAY
-122.27,37.83,51.0,2665.0,574.0,1258.0,536.0,2.7303,109700.0,NEAR BAY
-122.27,37.83,49.0,1215.0,282.0,570.0,264.0,1.4861,97200.0,NEAR BAY
-122.27,37.83,48.0,1798.0,432.0,987.0,374.0,1.0972,104500.0,NEAR BAY
-122.28,37.83,52.0,1511.0,390.0,901.0,403.0,1.4103,103900.0,NEAR BAY
-122.26,37.83,52.0,1470.0,330.0,689.0,309.0,3.48,191400.0,NEAR BAY
-122.26,37.83,52.0,2432.0,715.0,1377.0,696.0,2.5898,176000.0,NEAR BAY
-122.26,37.83,52.0,1665.0,419.0,946.0,395.0,2.0978,155400.0,NEAR BAY
-122.26,37.83,51.0,936.0,311.0,517.0,249.0,1.2852,150000.0,NEAR BAY
-122.26,37.84,49.0,713.0,202.0,462.0,189.0,1.025,118800.0,NEAR BAY
-122.26,37.84,52.0,950.0,202.0,467.0,198.0,3.9643,188800.0,NEAR BAY
-122.26,37.83,52.0,1443.0,311.0,660.0,292.0,3.0125,184400.0,NEAR BAY
-122.26,37.83,52.0,1656.0,420.0,718.0,382.0,2.6768,182300.0,NEAR BAY
-122.26,37.83,50.0,1125.0,322.0,616.0,304.0,2.026,142500.0,NEAR BAY
-122.27,37.82,43.0,1007.0,312.0,558.0,253.0,1.7348,137500.0,NEAR BAY
-122.26,37.82,40.0,624.0,195.0,423.0,160.0,0.9506,187500.0,NEAR BAY
-122.27,37.82,40.0,946.0,375.0,700.0,352.0,1.775,112500.0,NEAR BAY
```

ctrl+A



데이터 가져오기

```
In [2]: housing = pd.read_csv('./housin
        housing.head()
```

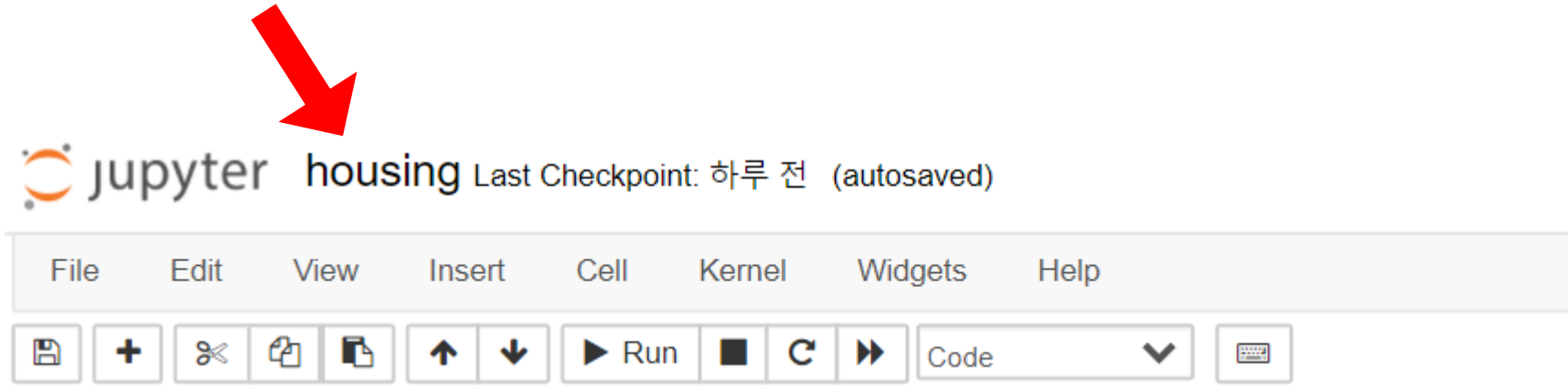
Out[2]:

	longitude	latitude	housi	tal_bedrooms	population		median_house_value	ocean_proximity
0	-122.23	37.88		129.0			452600.0	NEAR BAY
1	-122.22	37.86					358500.0	NEAR BAY
2	-122.24	37.85				.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.			5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0		.0	3.8462	342200.0	NEAR BAY

pandas 라이브러리의 러왔습니다.

- 주어진 데이터의 상위 코드
- read_csv 함수 내부에는 일의 경로가 지정되어야 합니다.

데이터 가져오기


jupyter
housing Last Checkpoint: 하루 전 (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

데이터 가져오기

In [4]:
```
import os
import tarfile
from six.moves import urllib

Download_root = "https://raw.githubusercontent.com/ExcelsiorCJH/Hands-On-ML/master/"
Housing_path = "datasets/housing"
Housing_url = Download_root + Housing_path + "/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()    .
```

In [5]:
```
def load_housing_data(housing_path = Housing_path):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [6]:
```
fetch_housing_data()
housing = load_housing_data()
housing.head()
```

데이터 가져오기

```
In [6]: fetch_housing_data()
housing = load_housing_data()
housing.head()
```

Out[6]:

	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	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

pandas 라이브러리의 read_csv 함수를 활용하여 csv파일을 불러왔습니다.

| 주어진 데이터의 상위 5개 Attribute를 보여주는 코드 : *dataframe.head()*

| read_csv 함수 내부에는 불러올 csv파일의 경로가 지정되어야 합니다.

2

데이터 통찰/시각화

데이터 시각화

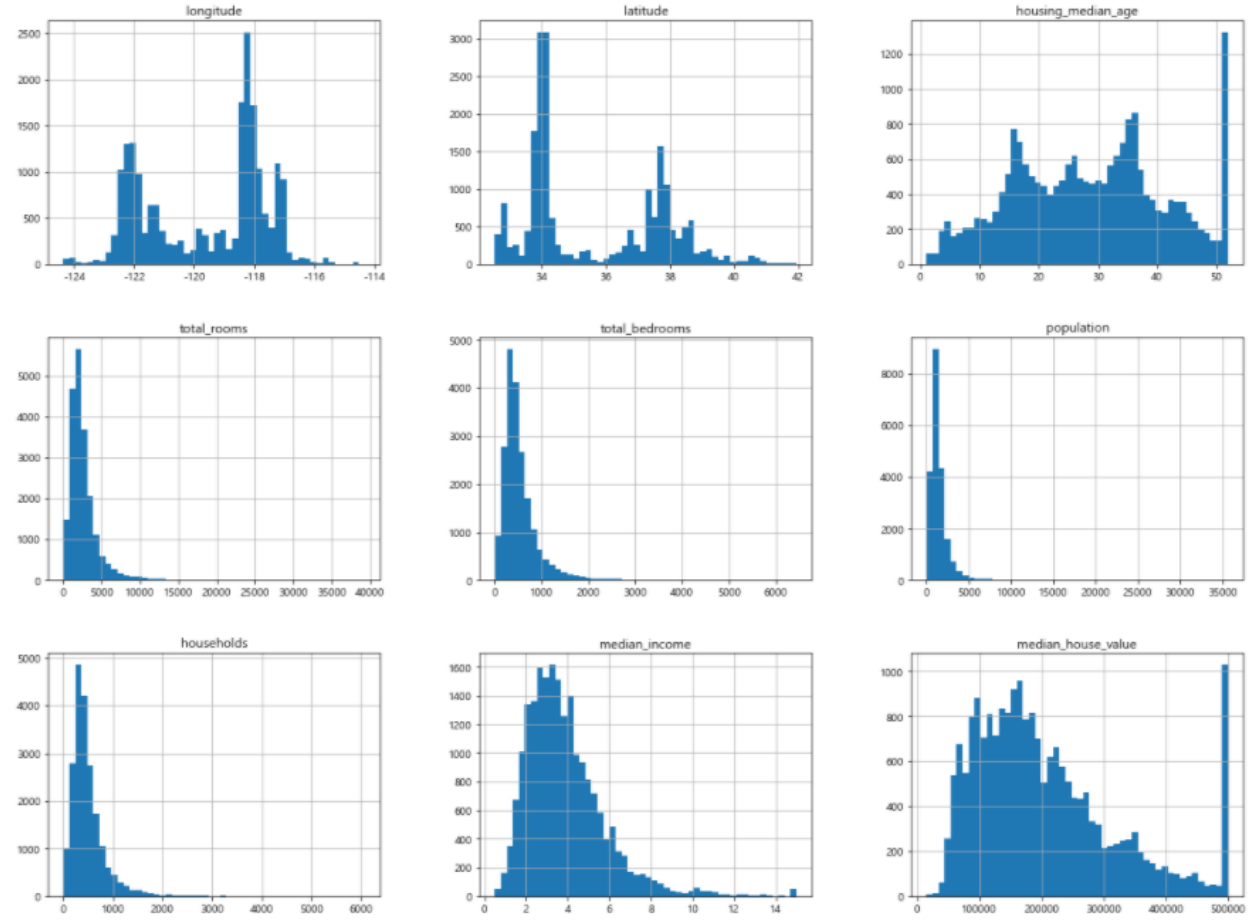
In [7]:
```
display(housing.info())
display(housing.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB

None
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
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	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

데이터 시각화

```
In [8]: housing.hist(bins = 50, figsize = (20,15))
        plt.show()
```



데이터 시각화

히스토그램 정리

| 중간 소득 특성이 US달러가 아님!

→ 중간 소득이 최대 15달러일수는 없다!

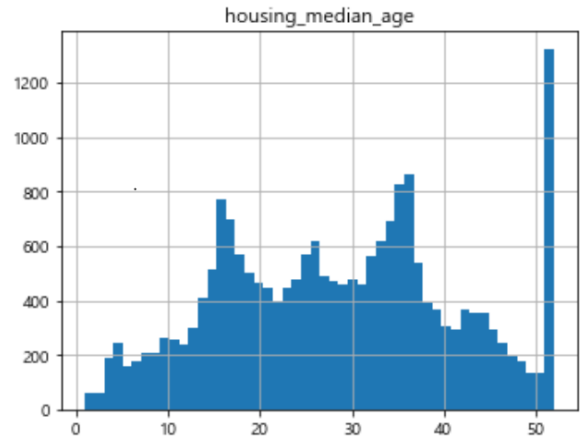
| (중요!) 이미 Scaling 된 결과라는 판단이 필요합니다.

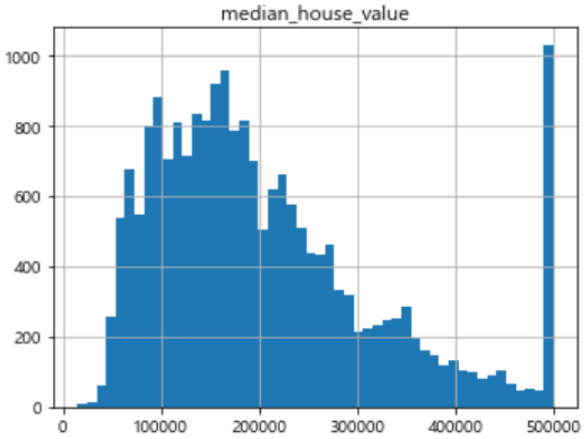
최소값을 0, 최대값을 15로 Scaling을 했네요

| 어느정도 정규성을 보장하고 있는 데이터라는 것을 이해합시다.

머신 러닝에서 전처리된 데이터를 다루는 경우가 흔하고, 이것이 문제가 되지는 않습니다. 다만, 데이터가 어떻게 계산되었는지는 반드시 이해합시다.

데이터 시각화





히스토그램 정리

| 중간 주택 연도/중간주택가격은 최대값, 최솟값을 한정

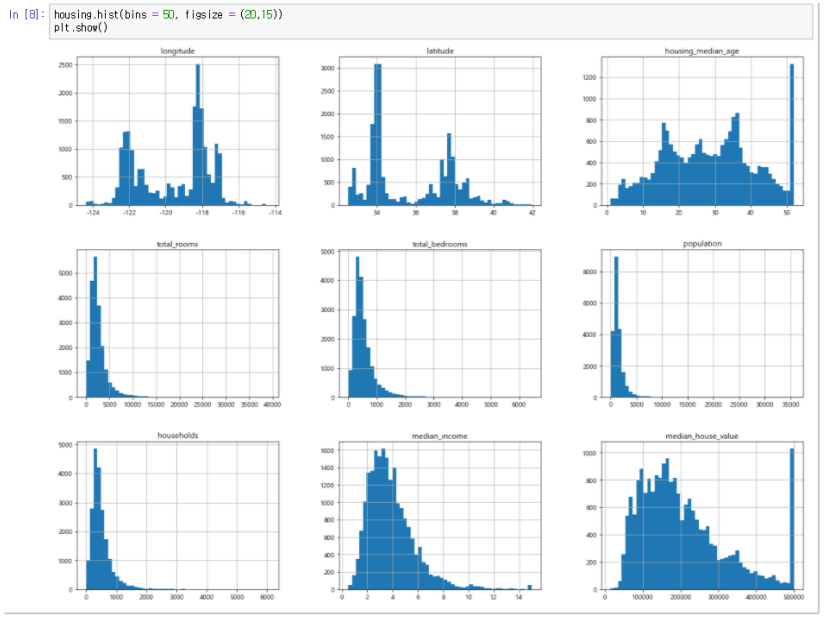
| (중요!) '중간주택가격'은 타깃 속성!

| 머신 러닝 학습 시 '50만 달러 초과 값'에 대한 예측은 나쁠 것

→ 한계값 밖 구역에 대한 정확한 값을 찾는다 (불가능)
→ 훈련 세트에서 이런 구역을 제거 (가능)

데이터 시각화

```
In [8]: housing.hist(bins = 50, figsize = (20,15))
        plt.show()
```



히스토그램 정리

| 전체적으로 첨도와 왜도가 상이합니다.

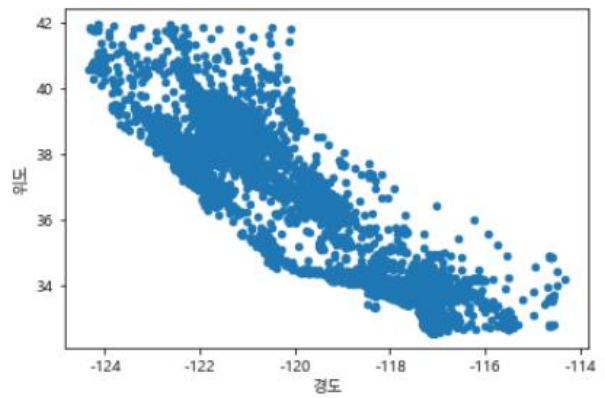
→ 이런 분포의 차이는 머신러닝 알고리즘에서 패턴을 찾기 어렵게 합니다.
→ 정규화/표준화가 필요합니다.

| 특성들의 스케일이 많이 다릅니다.

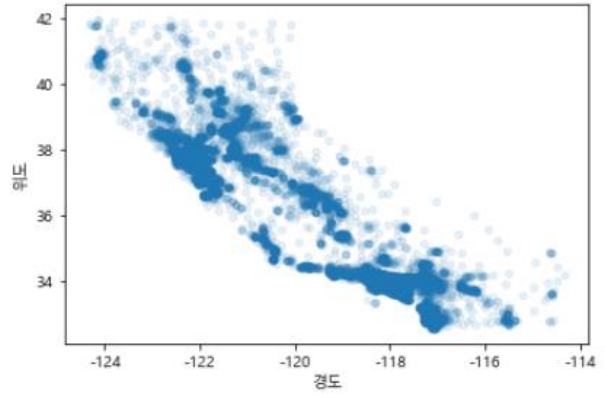
→ 각각의 서로 다른 특성들을 동일하게 Scaling할 필요가 있습니다.

데이터 시각화

In [10]:
```
ax = housing.plot(kind='scatter', x='longitude', y='latitude')
ax.set(xlabel='경도', ylabel='위도');
```



In [11]:
```
ax = housing.plot(kind='scatter', x='longitude', y='latitude',
ax.set(xlabel='경도', ylabel='위도');
```

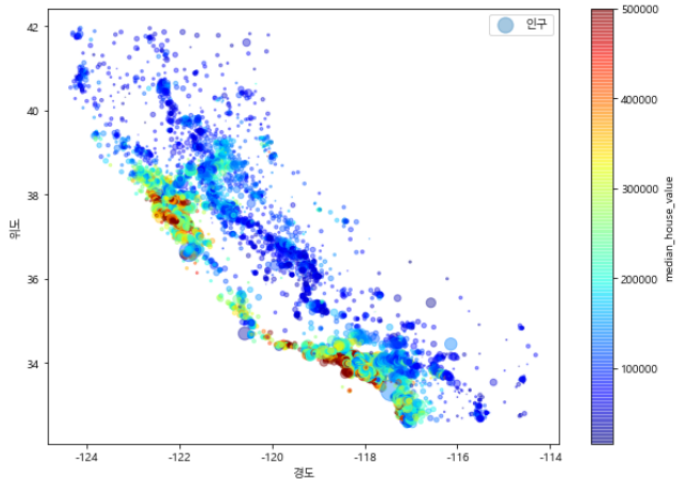


데이터 시각화

	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	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

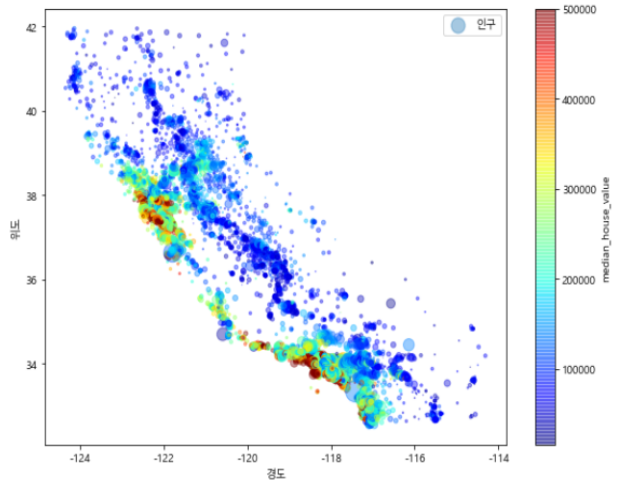
In [13]:
```
ax = housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
                  s = housing['population']/100, label='인구',figsize=(10,7),
                  c = 'median_house_value', cmap=plt.get_cmap("jet"), colorbar=True, sharex=False)

ax.set(xlabel='경도', ylabel='위도')
plt.legend()
```

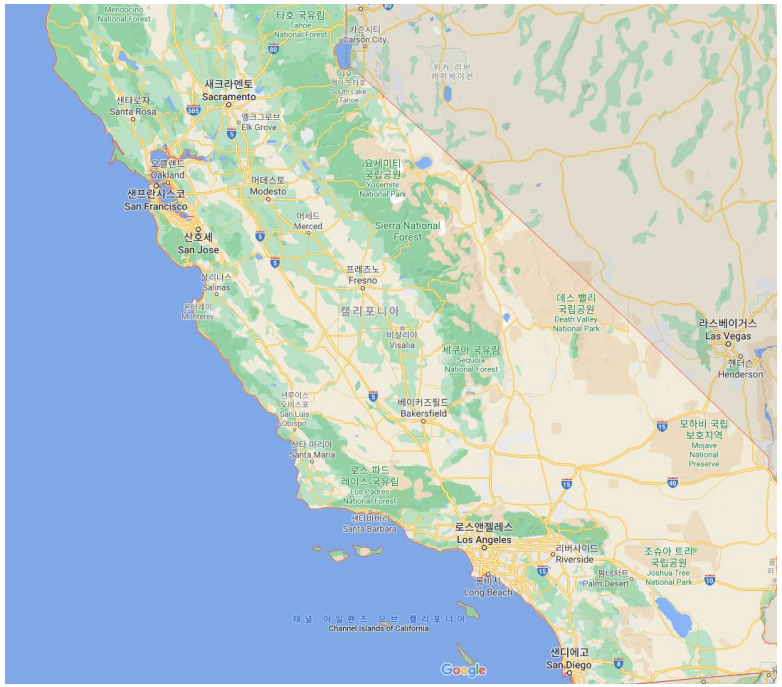
Out[13]: <matplotlib.legend.Legend at 0x1d60a300bc8>



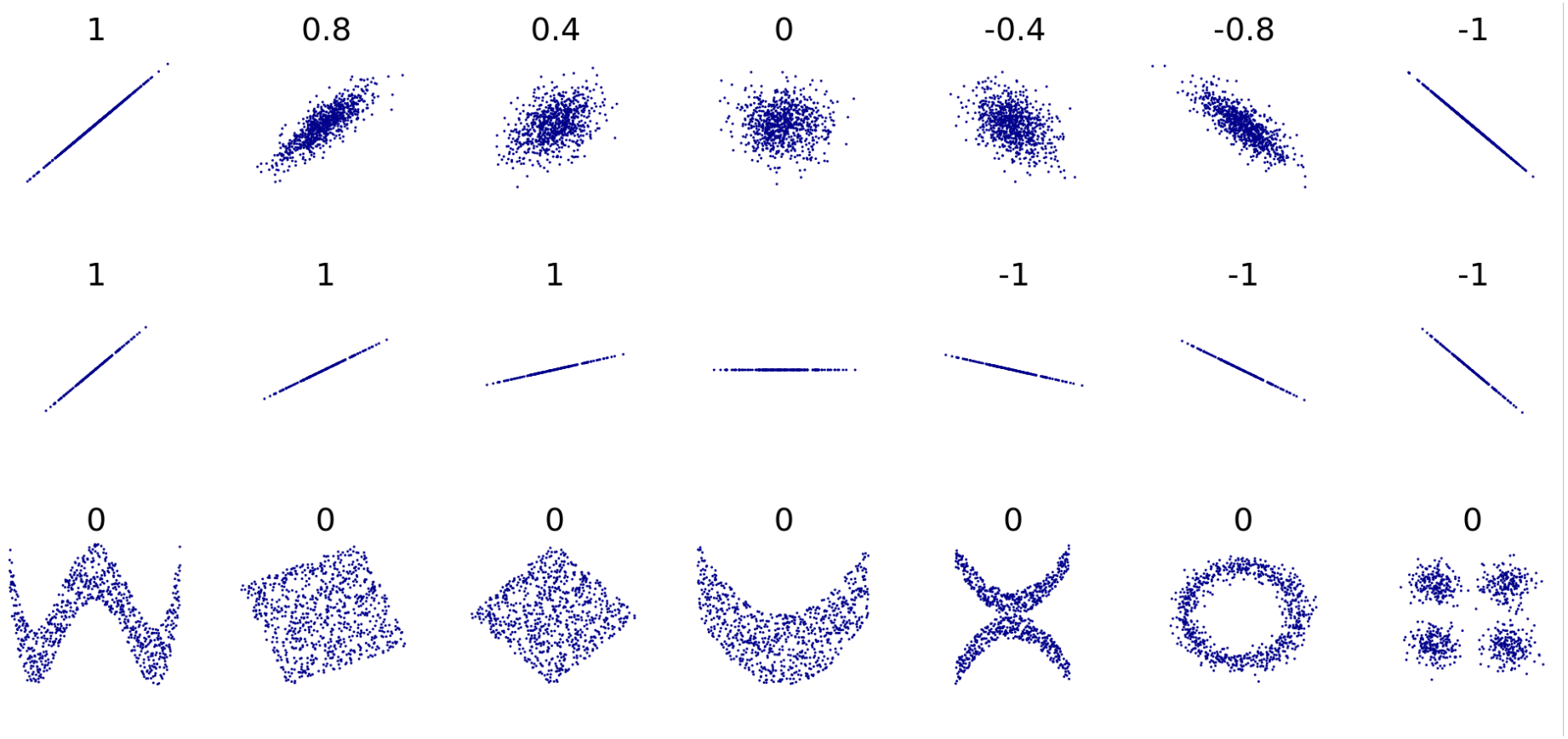
데이터 통찰

```
In [13]: ax = housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
                           s = housing['population']/100, label='인구',figsize=(10,7),
                           c = 'median_house_value', cmap=plt.get_cmap("jet"), colorbar=True, sharex=False)

         ax.set(xlabel='경도', ylabel='위도')
         plt.legend()

Out[13]: <matplotlib.legend.Legend at 0x1d60a300bc8>
```





데이터 통찰



데이터 통찰

In [14]:
```
corr_matrix = housing.corr()
corr_matrix
```

Out[14]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608	0.099773	0.055310	-0.015176	-0.045967
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983	-0.108785	-0.071035	-0.079809	-0.144160
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451	-0.296244	-0.302916	-0.119034	0.105623
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380	0.857126	0.918484	0.198050	0.134153
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000	0.877747	0.979728	-0.007723	0.049686
population	0.099773	-0.108785	-0.296244	0.857126	0.877747	1.000000	0.907222	0.004834	-0.024650
households	0.055310	-0.071035	-0.302916	0.918484	0.979728	0.907222	1.000000	0.013033	0.065843
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723	0.004834	0.013033	1.000000	0.688075
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686	-0.024650	0.065843	0.688075	1.000000

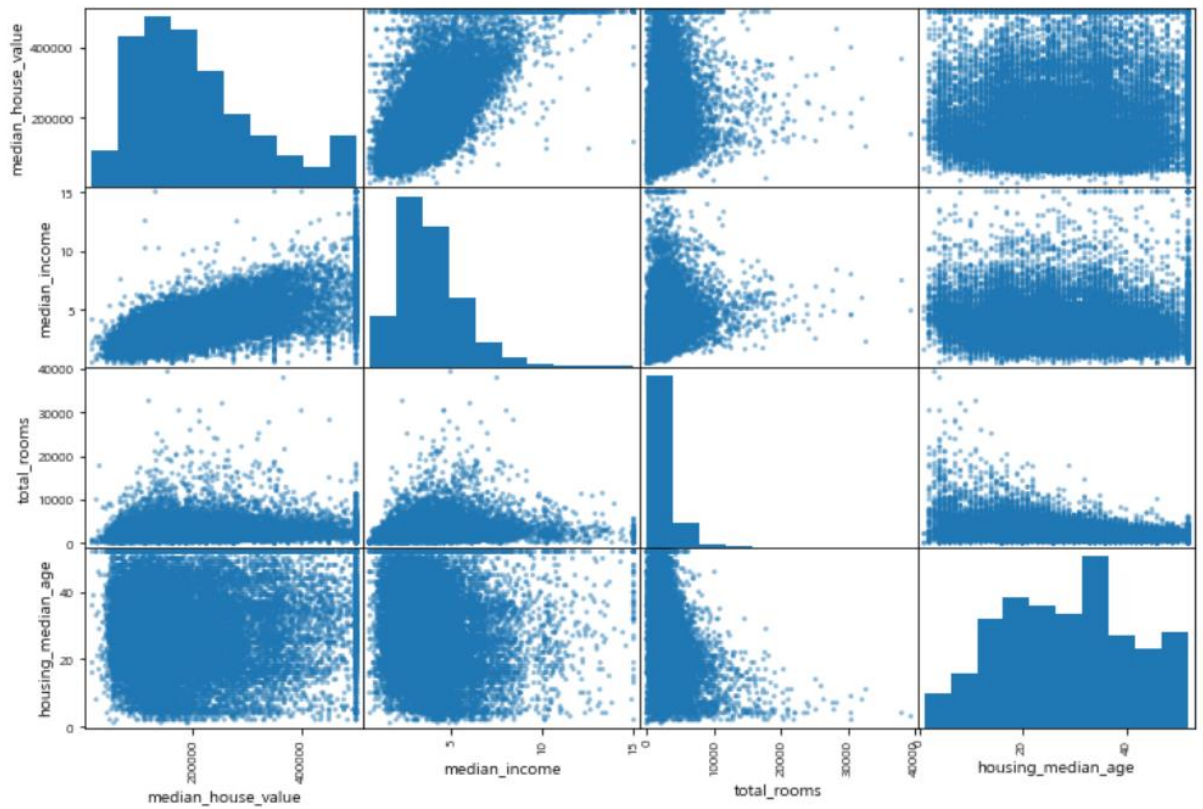
데이터 통찰

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

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

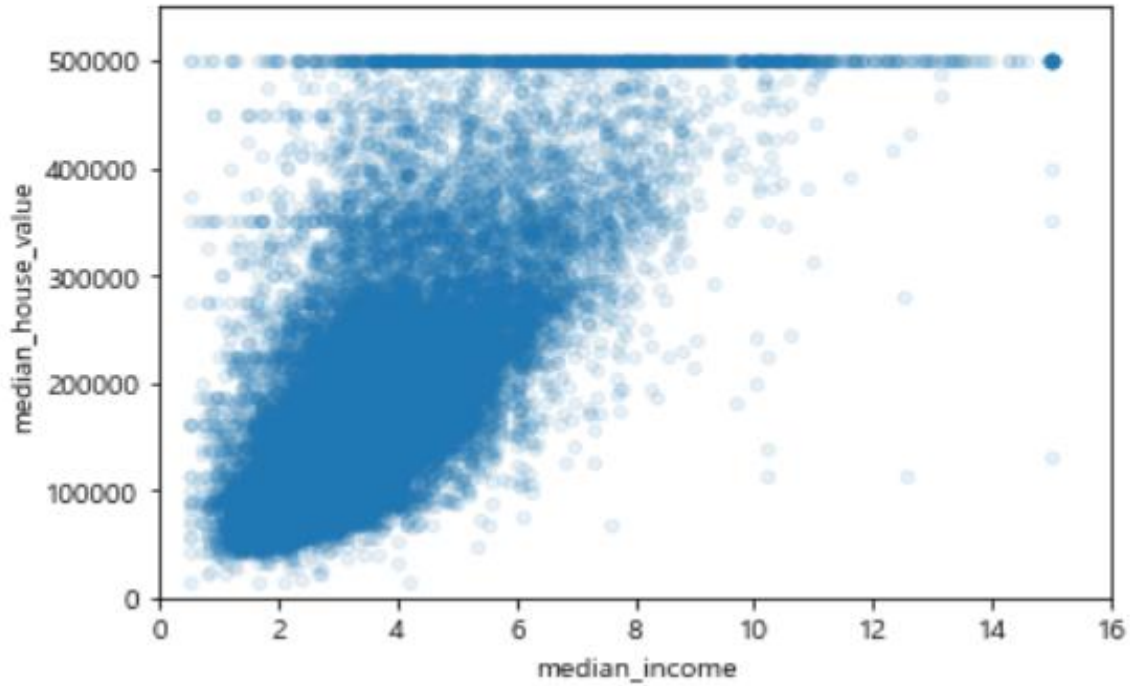
데이터 통찰

In [16]:
```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]

scatter_matrix(housing[attributes], figsize=(12,8));
```



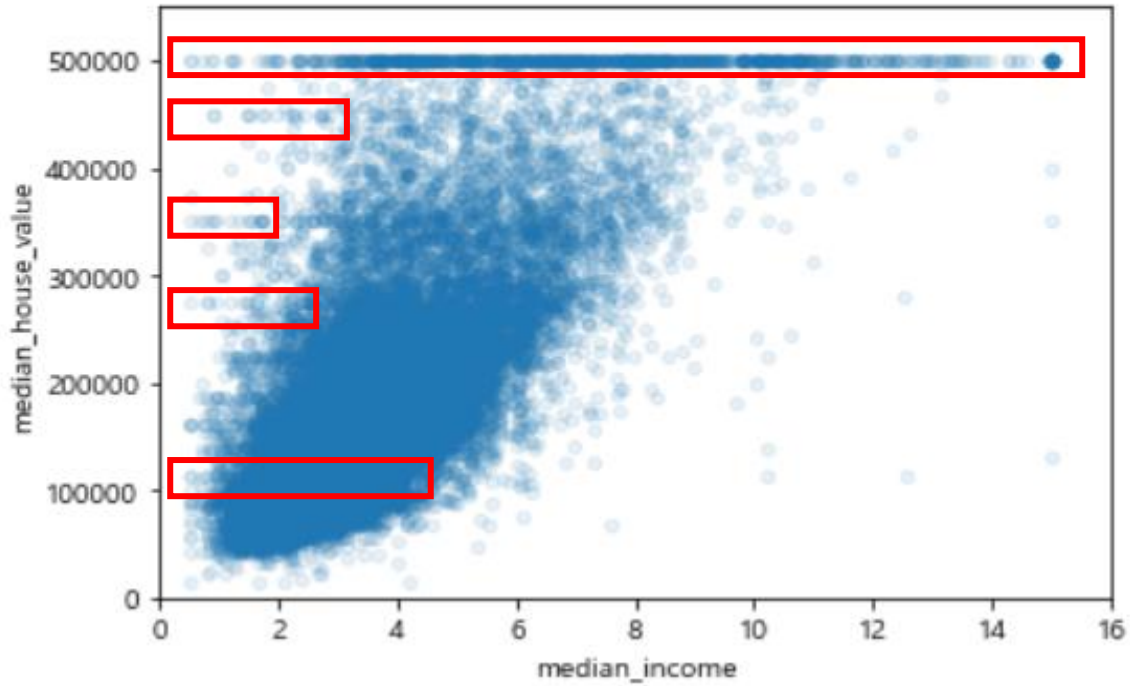
데이터 통찰

In [17]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000]);
```



데이터 통찰

```
In [17]: housing.plot(kind="scatter", x="median_income", y="median_house_value",
                      alpha=0.1)
         plt.axis([0, 16, 0, 550000]);
```



데이터 통찰

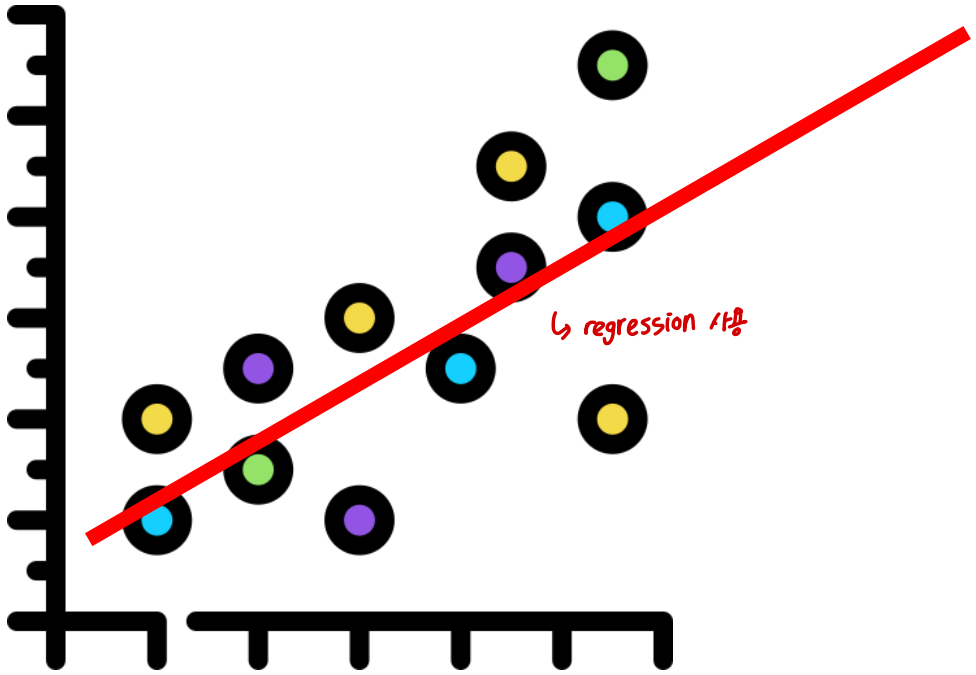
In [18]:
```
housing['rooms_per_household'] = housing['total_rooms'] / housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms'] / housing['total_rooms']
housing['population_per_household'] = housing['population'] / housing['households']

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

Out[18]:
```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

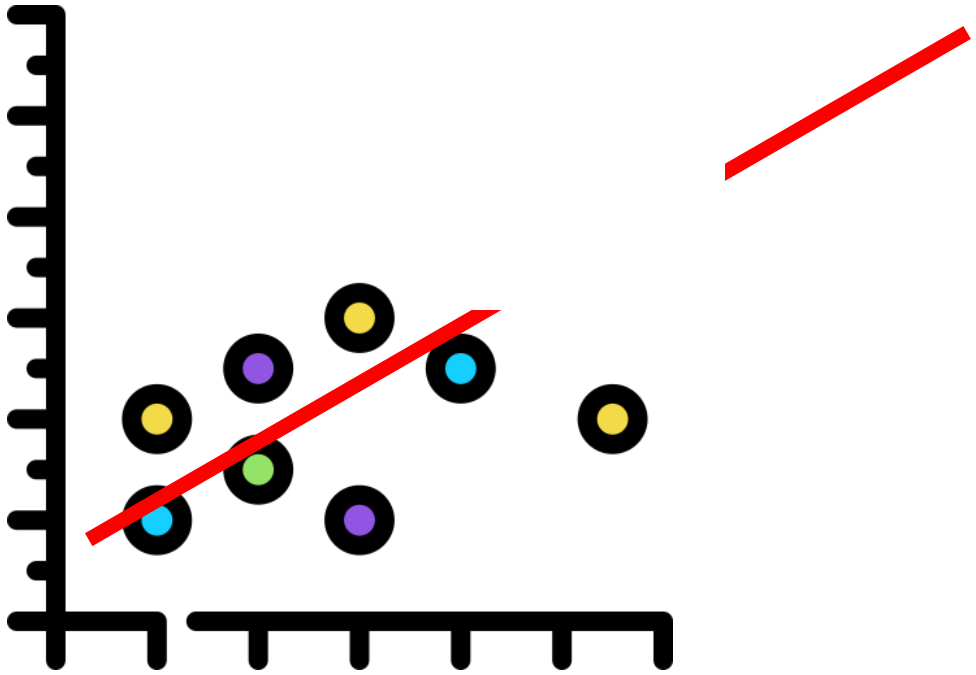
3

데이터 훈련/평가

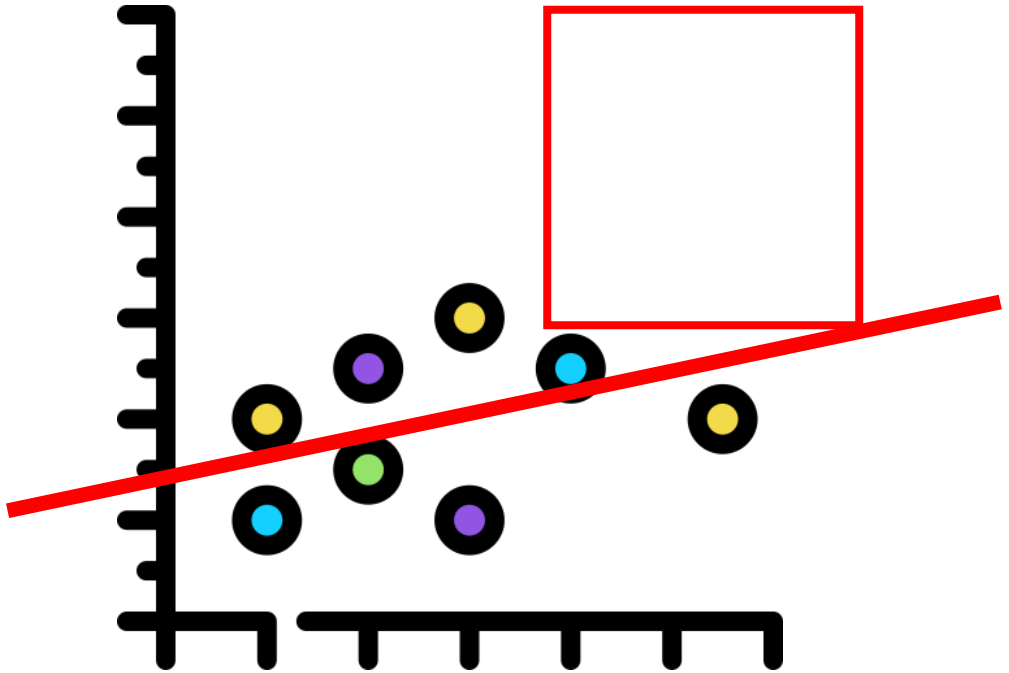
데이터 훈련



데이터 훈련



데이터 훈련



데이터 훈련

데이터를 더 깊게 들여다 보기 전에 Test Set을 따로 떼어놓아야 합니다.

↳ 과소적합 (underfitting)의 위험존재

그리고 Test Set을 절대 들여다보면 안됩니다!

| WHY?

→테스트 셋에서 겉으로 드러난 어떤 패턴에 속는다.
→특정 머신러닝 모델만 낙관적 편향에 따라 선택.
→일반화했을 때, 기대한 성능이 나오지 않을 것 (데이터 스누핑 발생).

데이터 훈련

Q

올바른 Training Set이란 무엇일까요?

어떻게 올바른 Training Set을 만들죠?

데이터 훈련 - 전처리

▶ 계층적 샘플링

In [19]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
print(len(train_set), "train +", len(test_set), "test")
```

```
16512 train + 4128 test
```

In [18]:
```
test_set
```

Out[18]:

	longitude	latitude	housing_median_age	total_rooms	...	ocean_proximity	rooms_per_household	bedrooms_per_room	population_per_household
20046	-119.01	36.06	25.0	1505.0	...	INLAND	4.1...	NaN	3.8...
3024	-119.46	35.14	30.0	2943.0	...	INLAND	5.0...	NaN	2.6...
15663	-122.44	37.80	52.0	3830.0	...	NEA...	3.9...	NaN	1.3...
20484	-118.72	34.28	17.0	3051.0	...	<1H...	6.1...	NaN	3.4...
9814	-121.93	36.62	34.0	2351.0	...	NEA...	5.4...	NaN	2.4...
...	...	...	...	...	...	...	...	...	...
15362	-117.22	33.36	16.0	3165.0	...	<1H...	7.0...	0.1...	2.9...
16623	-120.83	35.36	28.0	4323.0	...	NEA...	6.1...	0.2...	2.3...
18086	-122.05	37.31	25.0	4111.0	...	<1H...	7.2...	0.1...	2.7...
2144	-119.76	36.77	36.0	2507.0	...	INLAND	5.2...	0.1...	2.5...
3665	-118.37	34.22	17.0	1787.0	...	<1H...	3.9...	0.2...	3.7...

4128 rows × 13 columns

데이터 훈련 - 전처리

▶ 계층적 샘플링

In [19]:
```
train_set
```

Out[19]:

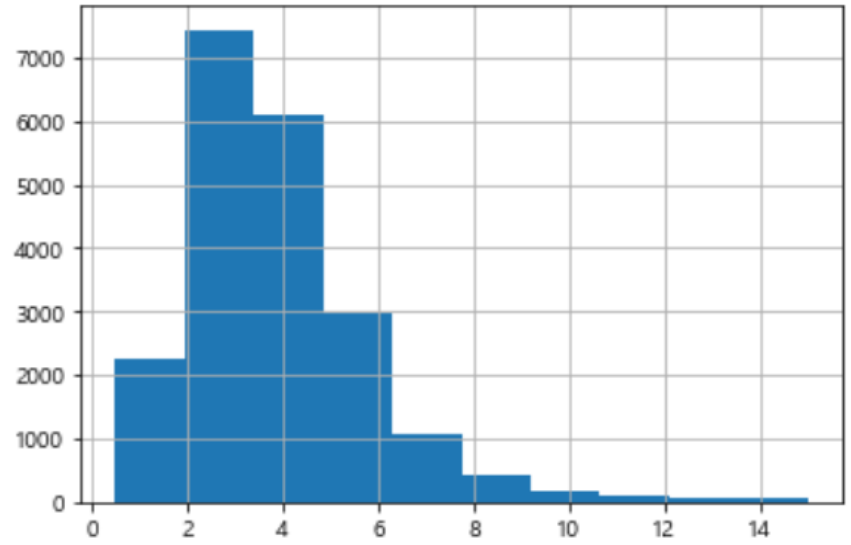
	longitude	latitude	housing_median_age	total_rooms	...	ocean_proximity	rooms_per_household	bedrooms_per_room	population_per_household
14196	-117.03	32.71	33.0	3126.0	...	NEA...	5.0...	0.2...	3.6...
8267	-118.16	33.77	49.0	3382.0	...	NEA...	4.4...	0.2...	1.7...
17445	-120.48	34.66	4.0	1897.0	...	NEA...	5.6...	0.1...	2.7...
14265	-117.11	32.69	36.0	1421.0	...	NEA...	4.0...	0.2...	3.9...
2271	-119.80	36.78	43.0	2382.0	...	INLAND	6.2...	0.1...	2.3...
...	...	...	...	...	...	...	...	...	...
11284	-117.96	33.78	35.0	1330.0	...	<1H...	6.1...	0.1...	3.0...
11964	-117.43	34.02	33.0	3084.0	...	INLAND	6.8...	0.1...	3.9...
5390	-118.38	34.03	36.0	2101.0	...	<1H...	3.9...	0.2...	3.3...
860	-121.96	37.58	15.0	3575.0	...	<1H...	6.3...	0.1...	3.1...
15795	-122.42	37.77	52.0	4226.0	...	NEA...	3.4...	0.3...	2.1...

16512 rows × 13 columns

데이터 훈련 - 전처리

▶ 계층적 샘플링

```
In [20]: housing['median_income'].hist();
```



히스토그램 정리

| 정규성을 띄는 데이터 특성인 중간 소득을 살펴봅시다!

→ 소득 별 빈도가 상이합니다.

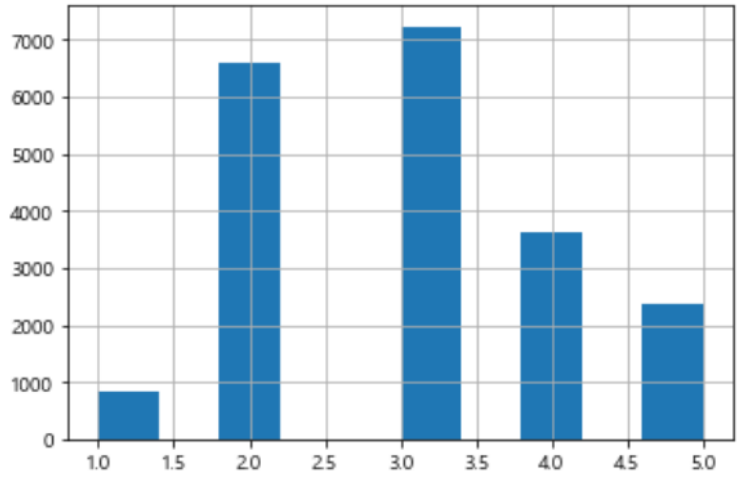
→ 이러한 상황에서 훈련 셋 80%, 테스트 셋 20%를 아무렇게나 뽑는 것은 옳지 않습니다.

| 계층별로 트레이닝, 테스트 셋을 뽑도록 합시다!

데이터 훈련 - 전처리

▶ 계층적 샘플링

```
In [23]: housing['income_cat'] = np.ceil(housing['median_income'] / 1.5)
         housing['income_cat'].where(housing['income_cat'] < 5, 5.0, inplace=True)

In [24]: housing['income_cat'].value_counts()

Out[24]: 3.0    7236
         2.0    6581
         4.0    3639
         5.0    2362
         1.0     822
         Name: income_cat, dtype: int64

In [25]: housing['income_cat'].hist();
```



히스토그램 정리

| 데이터를 5계층으로 분류했습니다.

→ 계층 별 비중에 따라 Sampling을 진행합시다!

데이터 훈련 - 전처리

▶ 계층적 샘플링

비슷한 비중으로 Sampling 되었습니다!

In [29]:
```
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]
```

```
display(housing['income_cat'].value_counts()/len(housing))
print('='*20)
display(strat_test_set['income_cat'].value_counts()/len(strat_test_set))
```

```
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

==================

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
```

데이터 훈련 - 전처리

▶ 계층적 샘플링

In [32]:
```
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
```

Out[32]:

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

테이블 정리

| 계층 별 비중의 오차 Table입니다.

→ Rand. %error > Strat. %error

→ 계층화하고 데이터를 선택하는 것의 일반화 성능이 더 좋을것입니다.

데이터 훈련 - 전처리

▶ 계층적 샘플링

In [35]:
```
for set_ in (strat_train_set, strat_test_set, housing):
    set_.drop("income_cat", axis=1, inplace=True)

housing.columns
```

Out[35]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'rooms_per_household',
       'bedrooms_per_room', 'population_per_household'],
      dtype='object')
```

| 더 이상 쓸모가 없는 'Income_cat' 특성을 제거합시다.

데이터 훈련 - 전처리

▶ NULL값 제거

In [57]:
```
housing = strat_train_set.drop(columns=['median_house_value']) #train_X
housing_labels = strat_train_set['median_house_value'].copy() #train_Y
```

In [59]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 17606 to 15775
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   longitude                 16512 non-null  float64
 1   latitude                  16512 non-null  float64
 2   housing_median_age        16512 non-null  float64
 3   total_rooms               16512 non-null  float64
 4   total_bedrooms            16354 non-null  float64
 5   population                16512 non-null  float64
 6   households                16512 non-null  float64
 7   median_income             16512 non-null  float64
 8   ocean_proximity           16512 non-null  object
 9   rooms_per_household       16512 non-null  float64
 10  bedrooms_per_room         16354 non-null  float64
 11  population_per_household  16512 non-null  float64
dtypes: float64(11), object(1)
memory usage: 1.6+ MB
```

데이터 훈련 - 전처리

▶ NULL값 제거

```
In [74]: housing[housing['bedrooms_per_room'].isna()==True]
```

Out[74]:

	longitude	latitude	housing_median_age	total_rooms	...	ocean_proximity	ooms_per_household	bedrooms_per_room	population_per_household
4629	-118.30	34.07	18.0	3759.0	...	<1H...	2.5...	NaN	2.2...
6068	-117.86	34.01	16.0	4632.0	...	<1H...	6.3...	NaN	4.1...
17923	-121.97	37.35	30.0	1955.0	...	<1H...	5.0...	NaN	2.5...
13656	-117.30	34.05	6.0	2155.0	...	INLAND	5.5...	NaN	2.6...
19252	-122.79	38.48	7.0	6837.0	...	<1H...	4.8...	NaN	2.4...
...	...	...	...	...	...	...	...	...	...
3376	-118.28	34.25	29.0	2559.0	...	<1H...	3.3...	NaN	2.4...
4691	-118.37	34.07	50.0	2519.0	...	<1H...	4.8...	NaN	2.1...
6052	-117.76	34.04	34.0	1914.0	...	INLAND	5.8...	NaN	4.7...
17198	-119.75	34.45	6.0	2864.0	...	NEA...	4.7...	NaN	2.3...
4738	-118.38	34.05	49.0	702.0	...	<1H...	3.7...	NaN	2.4...

158 rows × 12 columns

제거?　　대치?

데이터 훈련 - 전처리

▶ NULL값 제거

In [76]:
```
from sklearn.impute import SimpleImputer

housing_num = housing.drop('ocean_proximity', axis=1)
```

In [77]:
```
imputer = SimpleImputer(strategy='median')
imputer.fit(housing_num)
```

Out[77]:
```
SimpleImputer(strategy='median')
```

In [81]:
```
print('imputer.statistics_ :', list(map(lambda x: "%.2f"%x, imputer.statistics_)))
print('housing_num.median().values :', list(map(lambda x: "%.2f"%x, housing_num.median().values)))
```

```
imputer.statistics_ : ['-118.51', '34.26', '29.00', '2119.50', '433.00', '1164.00', '408.00', '3.54', '5.23', '0.20', '2.82']
housing_num.median().values : ['-118.51', '34.26', '29.00', '2119.50', '433.00', '1164.00', '408.00', '3.54', '5.23', '0.20', '2.82']
```

데이터 훈련 - 전처리

▶ NULL값 제거

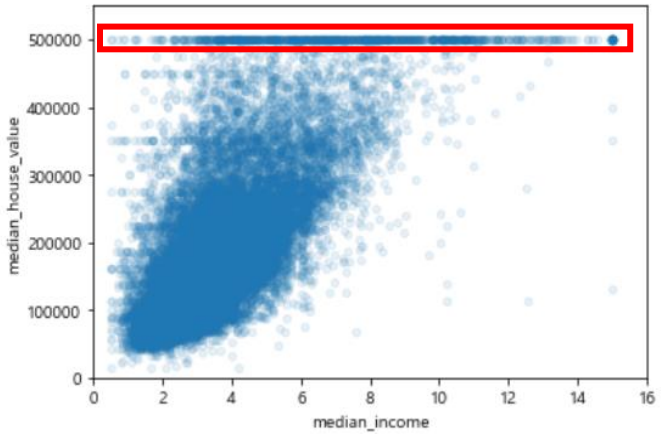
In [83]:
```
X = imputer.transform(housing_num)

housing_tr = pd.DataFrame(X, columns = housing_num.columns, index = housing.index.values)
```

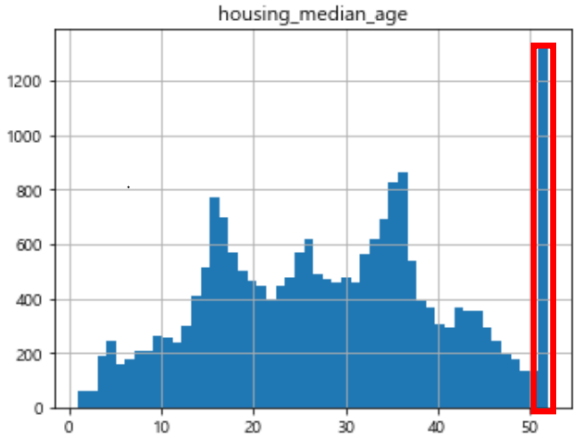
In [84]:
```
housing_tr.info()
```

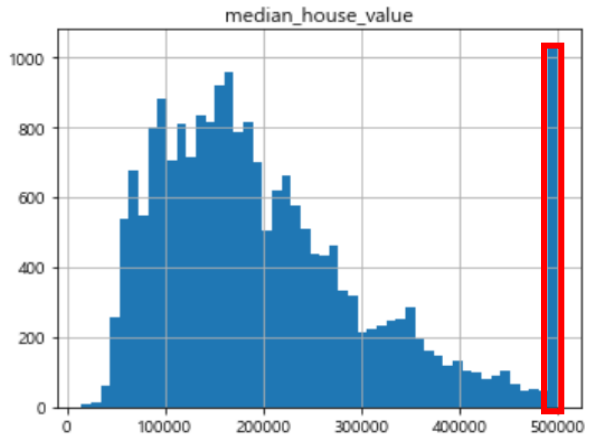
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 17606 to 15775
Data columns (total 11 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   longitude                 16512 non-null  float64
 1   latitude                  16512 non-null  float64
 2   housing_median_age        16512 non-null  float64
 3   total_rooms               16512 non-null  float64
 4   total_bedrooms            16512 non-null  float64
 5   population                16512 non-null  float64
 6   households                16512 non-null  float64
 7   median_income             16512 non-null  float64
 8   rooms_per_household       16512 non-null  float64
 9   bedrooms_per_room         16512 non-null  float64
 10  population_per_household  16512 non-null  float64
dtypes: float64(11)
memory usage: 1.5 MB
```

데이터 훈련 - 전처리

▶ 이상치 제거



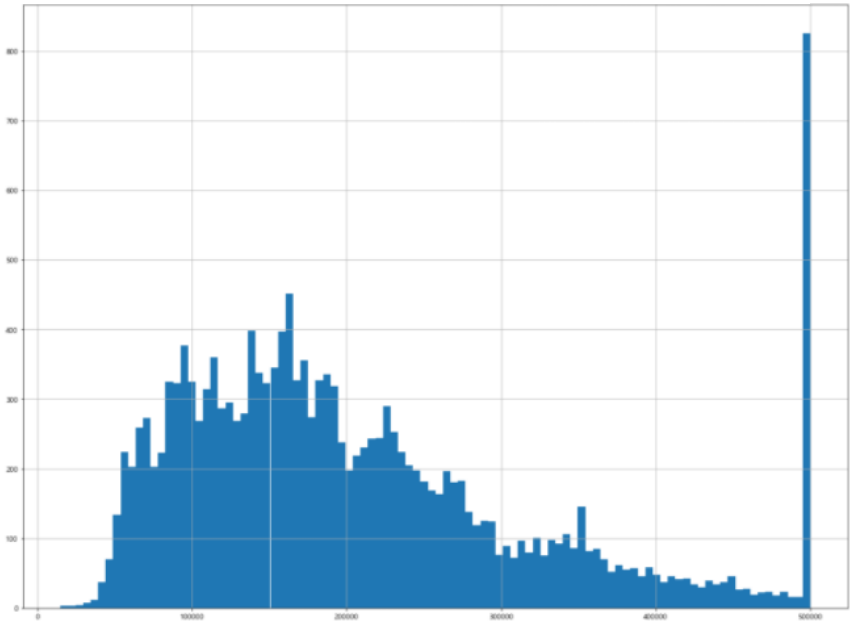




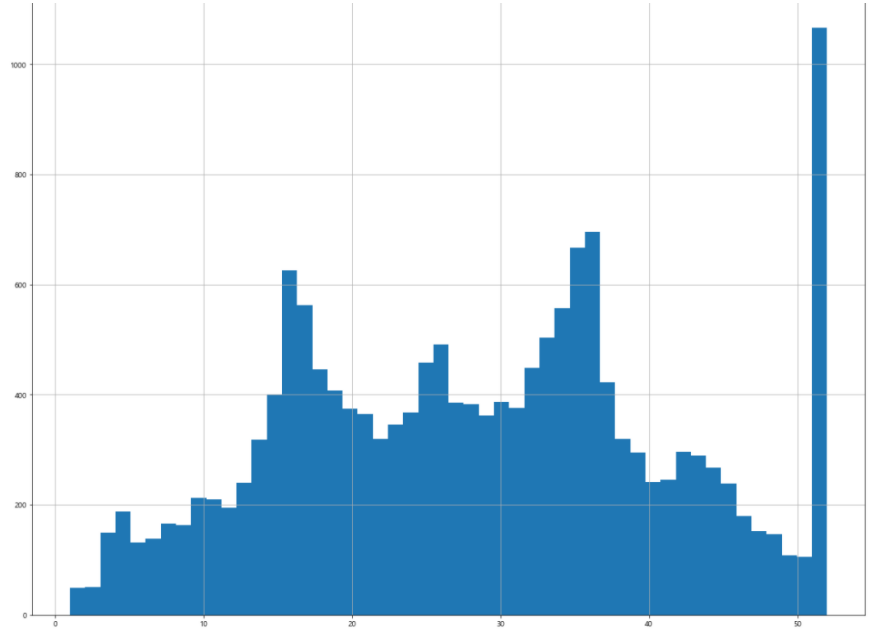
데이터 훈련 - 전처리

▶ 이상치 제거

In [63]:
```
housing_labels.hist(bins = 100, figsize = (20,15))
plt.show()
```



In [70]:
```
housing_tr['housing_median_age'].hist(bins = 50, figsize = (20,15))
plt.show()
```



데이터 훈련 - 전처리

▶ 이상치 제거

```
In [83]: housing_tr[housing_tr['housing_median_age']==max(housing_tr['housing_median_age'])]
```

Out[83]:

	longitude	latitude	housing_median_age	total_rooms	...	median_income	rooms_per_household	bedrooms_per_room	population_per_household
8879	-118.50	34.04	52.0	2233.0	...	8.3839	8.0...	0.1...	2.7...
16121	-122.46	37.79	52.0	2059.0	...	3.7419	5.1...	0.2...	2.4...
2651	-124.10	40.47	52.0	1196.0	...	3.5345	4.5...	0.1...	3.6...
16131	-122.48	37.79	52.0	4683.0	...	4.1148	4.8...	0.2...	2.3...
533	-122.27	37.78	52.0	1222.0	...	3.7708	4.6...	0.2...	2.3...
...	...	...	...	...	...	...	...	...	...
15670	-122.44	37.80	52.0	1006.0	...	2.7717	3.9...	0.2...	1.7...
11878	-117.38	33.99	52.0	1797.0	...	2.7054	5.7...	0.1...	2.8...
108	-122.24	37.82	52.0	3481.0	...	3.9000	4.8...	0.2...	2.0...
5364	-118.42	34.04	52.0	1358.0	...	5.6454	5.0...	0.2...	2.1...
15775	-122.45	37.77	52.0	3095.0	...	3.5750	4.8...	0.2...	1.9...

1027 rows × 11 columns

데이터 훈련 - 전처리

▶ 이상치 제거

```
In [84]: housing_labels[housing_labels.values==max(housing_labels.values)]

Out[84]: 8879     500...
         4861     500...
         15614    500...
         18086    500...
         16121    500...
                   ...
         8667     500...
         5306     500...
         18067    500...
         5364     500...
         15775    500...
         Name: median_house_value, Length: 786, dtype: float64
```

데이터 훈련 - 전처리

▶ 이상치 제거

In [82]:
```
X_index = housing_tr[housing_tr['housing_median_age']==max(housing_tr['housing_median_age'])].index

y_index = housing_labels[housing_labels.values==max(housing_labels.values)].index

print("Variable_index - Target_index :₩t",len(set(X_index) - set(y_index)))

print("Target_index - Variable_index :₩t",len(set(y_index)-set(X_index)))

print("Union of Variable_index & Target_index :", len(set(y_index).union(set(X_index))))
```

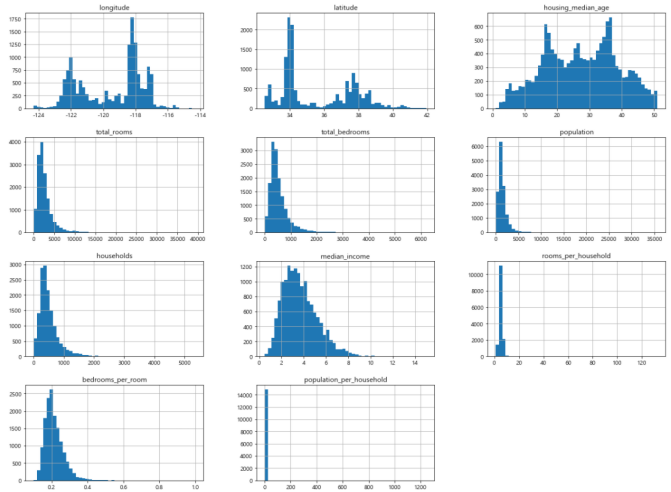
```
Variable_index - Target_index :  887
Target_index - Variable_index :  646
Union of Variable_index & Target_index : 1673
```

데이터 훈련 - 전처리

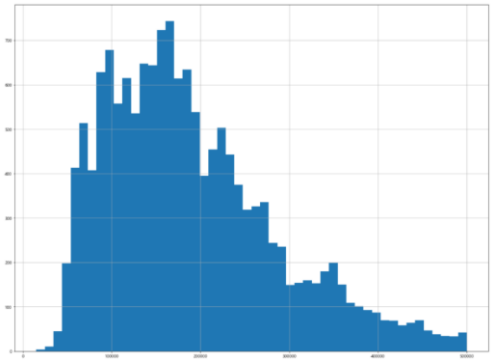
▶ 이상치 제거

In [92]:
```
tmp_x = housing_tr.drop(index = list(X_union_y))
tmp_y = housing_labels.drop(index = list(X_union_y))

tmp_x.hist(bins = 50, figsize = (20,15))
plt.show()

tmp_y.hist(bins = 50, figsize = (20,15))
plt.show()
```

X - variables

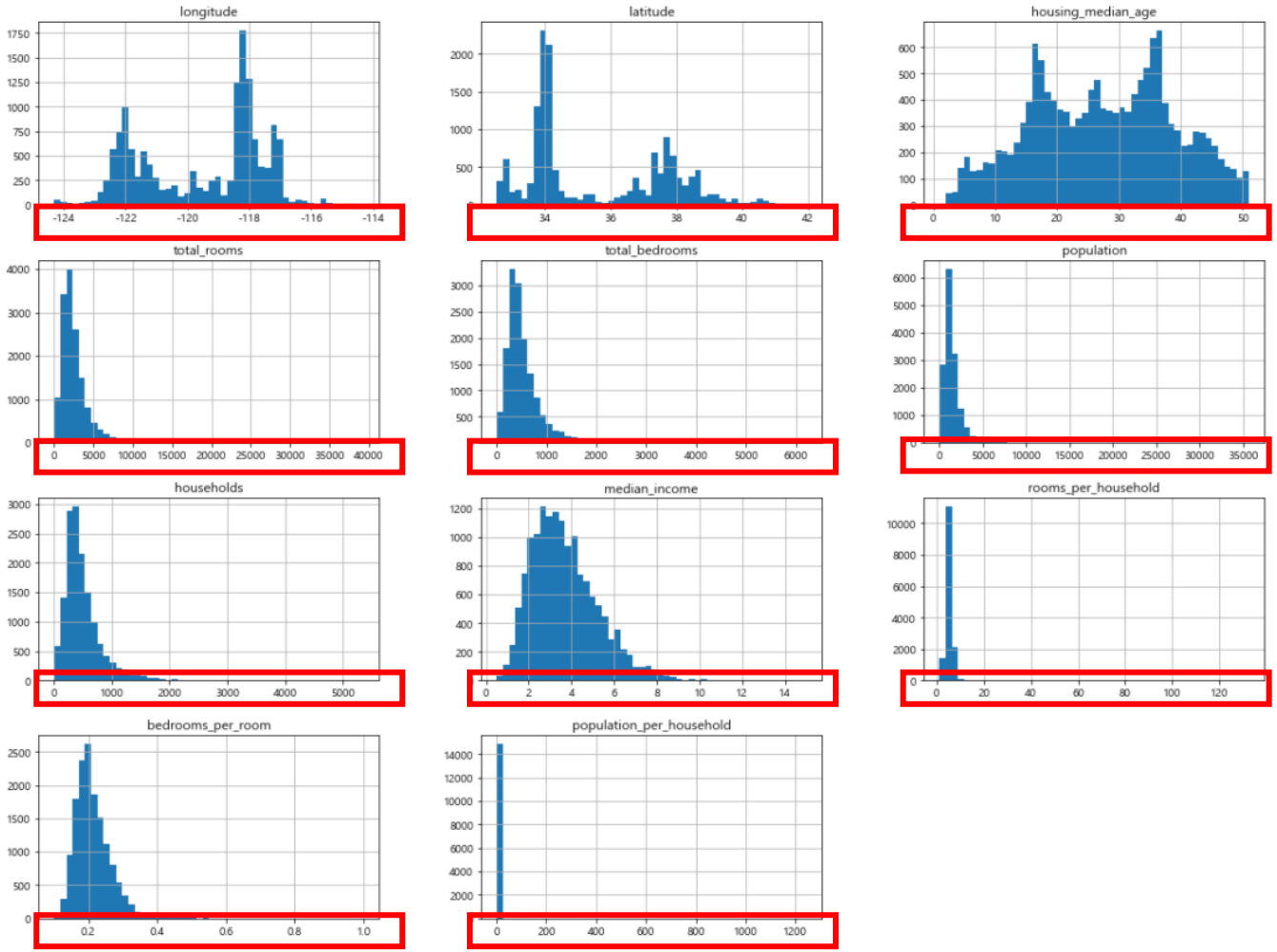


y - target



데이터 훈련 - 전처리

▶ 특성 스케일링



데이터 훈련 - 전처리

▶ 특성 스케일링

정규화

In [102]:
```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10],[1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))
print("max of data = ", scaler.data_max_)
print("min_of data = ", scaler.data_min_)
print("data range = ", scaler.data_range_)
print("<data scaling> ₩n", scaler.transform(data))
print("[2, 2] minmax scaling ", scaler.transform([[2,2]]))
```

```
MinMaxScaler()
max of data =  [ 1. 18.]
min_of data =  [-1.  2.]
data range =  [ 2. 16.]
<data scaling>
 [[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
[2, 2] minmax scaling  [[1.5 0. ]]
```

표준화

In [104]:
```
from sklearn.preprocessing import StandardScaler

data = [[-1, 2], [-0.5, 6], [0, 10],[1, 18]]
scaler = StandardScaler()
print(scaler.fit(data))
print("mean = ", scaler.mean_)
print("variance = ", scaler.var_)
print("z-score ₩n", scaler.transform(data))
print("[2, 2] standard scaling ", scaler.transform([[2,2]]))
```

```
StandardScaler()
mean =  [-0.125  9.   ]
variance =  [ 0.546875 35.      ]
z-score
 [[-1.18321596 -1.18321596]
 [-0.50709255 -0.50709255]
 [ 0.16903085  0.16903085]
 [ 1.52127766  1.52127766]]
[2, 2] standard scaling  [[ 2.87352447 -1.18321596]]
```

데이터 훈련 - 전처리

▶ 특성 스케일링

In [58]:
```
scaler = StandardScaler()
scaler.fit(tmp_x)
tmp_prepared = scaler.transform(tmp_x)

prepared = pd.DataFrame(tmp_prepared, columns = tmp_x.columns, index = tmp_x.index)
prepared.head()
```

Out[58]:

	longitude	latitude	housing_median_age	total_rooms	...	median_income	rooms_per_household	bedrooms_per_room	population_per_household
17606	-1.2...	0.7...	0.9...	-0.4...	...	-0.6...	-0.3...	0.1...	-0.0...
18632	-1.2...	0.6...	-1.1...	-0.9...	...	1.7...	0.2...	-1.0...	-0.0...
14650	1.1...	-1.3...	0.3...	-0.3...	...	-0.5...	-0.4...	0.5...	-0.0...
3230	-0.0...	0.3...	-0.1...	-0.3...	...	-1.1...	-0.0...	-0.2...	0.0...
3555	0.4...	-0.6...	-0.8...	1.8...	...	-0.4...	-0.3...	0.3...	-0.0...

5 rows × 11 columns

데이터 훈련 - 전처리

▶ 범주형 데이터 처리

```
In [55]: housing['ocean_proximity']
```

```
Out[55]: 17606     <1H...
         18632     <1H...
         14650     NEA...
         3230      INLAND
         3555      <1H...
                    ...
         6563      INLAND
         12053     INLAND
         13908     INLAND
         11159     <1H...
         15775     NEA...
         Name: ocean_proximity, Length: 16512, dtype: object
```

```
In [54]: housing_cat = pd.get_dummies(housing['ocean_proximity'])
         housing_cat = housing_cat.loc[tmp_x.index]
         housing_cat
```

Out[54]:

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
17606	1	0	0	0	0
18632	1	0	0	0	0
14650	0	0	0	0	1
3230	0	1	0	0	0
3555	1	0	0	0	0
...	...	...	...	...	...
7364	1	0	0	0	0
6563	0	1	0	0	0
12053	0	1	0	0	0
13908	0	1	0	0	0
11159	1	0	0	0	0

14839 rows × 5 columns

데이터 훈련 - 전처리

▶ 범주형 데이터 처리

In [57]:
```
housing = pd.concat([prepared, housing_cat], axis=1)
housing
```

Out[57]:

	longitude	latitude	housing_median_age	total_rooms	...	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
17606	-1.2...	0.7...	0.9...	-0.4...	...	0	0	0	0
18632	-1.2...	0.6...	-1.1...	-0.9...	...	0	0	0	0
14650	1.1...	-1.3...	0.3...	-0.3...	...	0	0	0	1
3230	-0.0...	0.3...	-0.1...	-0.3...	...	1	0	0	0
3555	0.4...	-0.6...	-0.8...	1.8...	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...
7364	0.6...	-0.7...	1.4...	-0.8...	...	0	0	0	0
6563	0.6...	-0.6...	1.6...	-0.6...	...	1	0	0	0
12053	0.9...	-0.7...	1.1...	-0.6...	...	1	0	0	0
13908	1.5...	-0.7...	-1.5...	1.0...	...	1	0	0	0
11159	0.7...	-0.8...	0.3...	-0.3...	...	0	0	0	0

14839 rows × 16 columns

데이터 훈련 - 훈련

▶ 선형 회귀 모델

In [62]:
```
from sklearn.linear_model import LinearRegression

housing_prepared = np.array(housing)
housing_labels = tmp_y

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

Out[62]:
```
LinearRegression()
```

In [64]:
```
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)
print('RMSE : ', lin_rmse)
```

```
RMSE :  58244.36758810236
```

데이터 훈련 - 훈련

▶ 의사결정나무 모델

In [66]:
```
from sklearn.tree import DecisionTreeRegressor

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

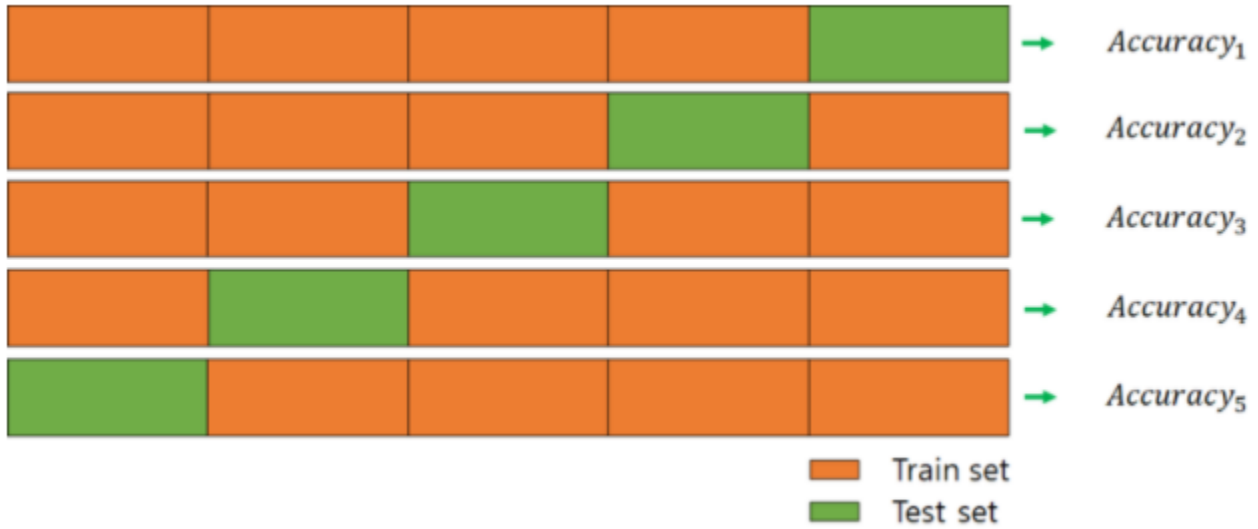
Out[66]:
```
DecisionTreeRegressor()
```

In [67]:
```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
print('RMSE : ', tree_rmse)
```

```
RMSE :  0.0
```

데이터 훈련 - 훈련

- Cross Validation



데이터 훈련 - 훈련

▶ Cross Validation

In [69]:
```
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)
tree_rmse_scores = np.sqrt(-scores)
```

In [70]:
```
def display_scores(scores):
    print("점수 :", scores)
    print("평균 :", scores.mean())
    print("표준편차 :", scores.std())

display_scores(tree_rmse_scores)
```

```
점수 : [62454.41054262 63025.77955203 63832.92408127 63777.66473357
 65482.47275832 65252.96917991 60402.82013775 61480.54712836
 64392.03096883 65620.55714711]
평균 : 63572.21762297725
표준편차 : 1655.9051509908227
```

CrossValidation - Tree model

In [71]:
```
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 : [56722.17148984 58053.7705809  57735.81527587 64063.65140457
 58965.20314997 59435.21574818 54764.97078541 58147.54954935
 60177.86191379 57146.94472927]
평균 : 58521.31546271383
표준편차 : 2338.912842506291
```

CrossValidation – Linear Regression model

데이터 훈련 - 훈련

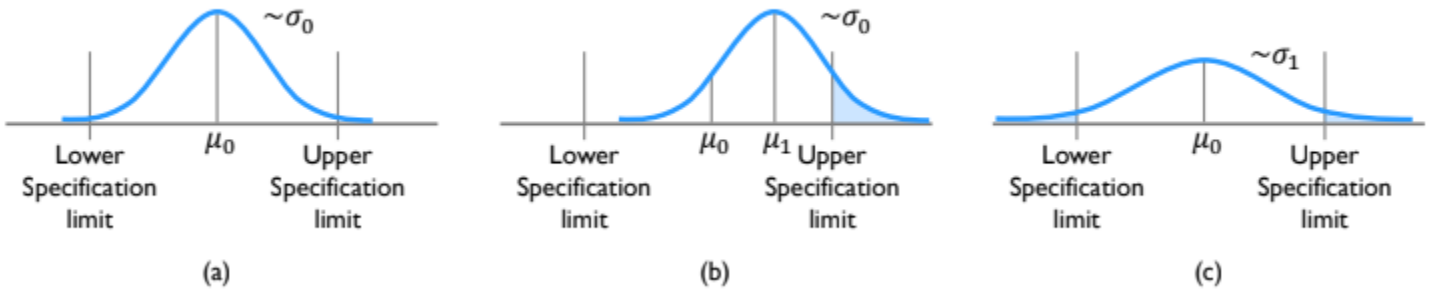
▶ Cross Validation

In [70]:
```
def display_scores(scores):
    print("점수 :", scores)
    print("평균 :", scores.mean())
    print("표준편차 :", scores.std())

display_scores(tree_rmse_scores)
```

```
점수 : [62454.41054262 63025.77955203 63832.92408127 63777.66473357
 65482.47275832 65252.96917991 60402.82013775 61480.54712836
 64392.03096883 65620.55714711]
평균 : 63572.21762297725
표준편차 : 1655.9051509908227
```

In [71]:
```
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 : [56722.17148984 58053.7705809  57735.81527587 64063.65140457
 58965.20314997 59435.21574818 54764.97078541 58147.54954935
 60177.86191379 57146.94472927]
평균 : 58521.31546271383
표준편차 : 2338.912842506291
```



어떤 알고리즘이 더 좋을까요?

Test Set과의 비교

다음시간에…

Q&A