2.2~2.3

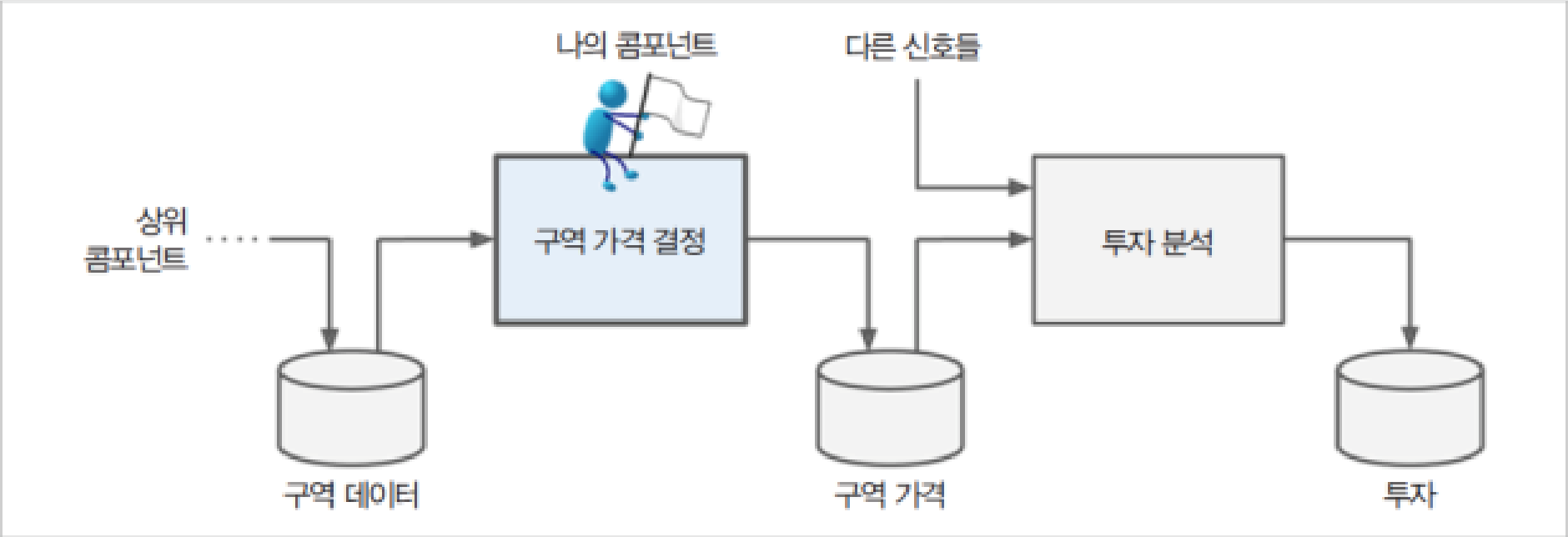
Chapter 2. 머신러닝 프로젝트 처음부터 끝까지

목차

2.2.1 문제 정의

파이프라인 : 일련의 과정을 자동화
➔ 수동적 작업의 배제로 작업의 효율성을 높인다

그림 2-2 부동산 투자를 위한 머신러닝 파이프라인



2.2.1 문제 정의

- 지도 OR 비지도 OR 강화
- 분류 OR 회귀 (다변량 회귀)
- 배치 OR 온라인 학습

2.2.1 문제 정의

TIP : 데이터가 매우 크면

1. 맵리듀스 사용 : 대용량 데이터 처리를 분산 병렬 컴퓨팅에서 처리

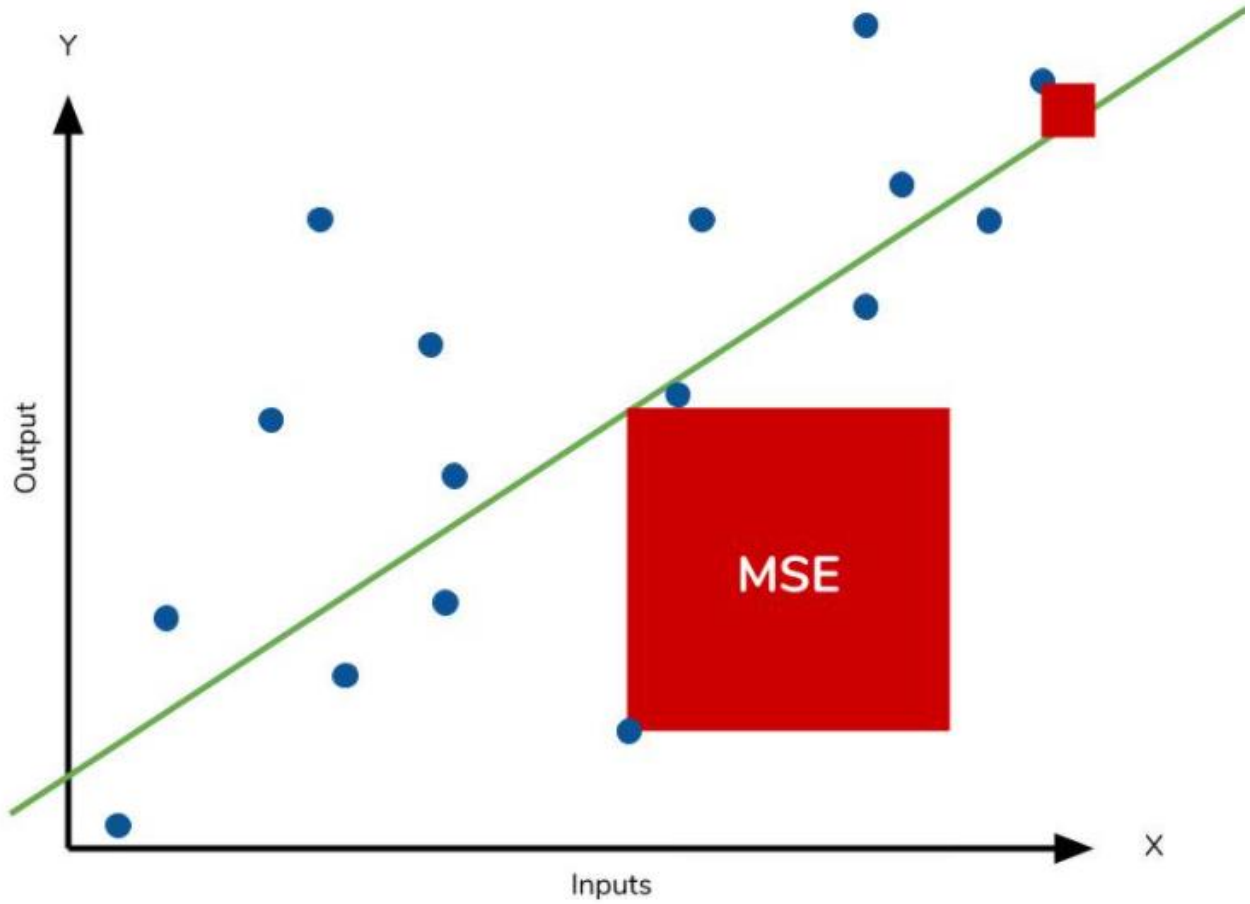
   ➔ 배치 학습을 여러 서버로 분할
2. 온라인 학습 기법 사용 : 데이터를 작게 쪼개 묶음 단위로 시스템 훈련

   ➔ 적은 비용으로 학습 가능

2.2.2 성능 측정 지표 선택

- RMSE(평균제곱근)

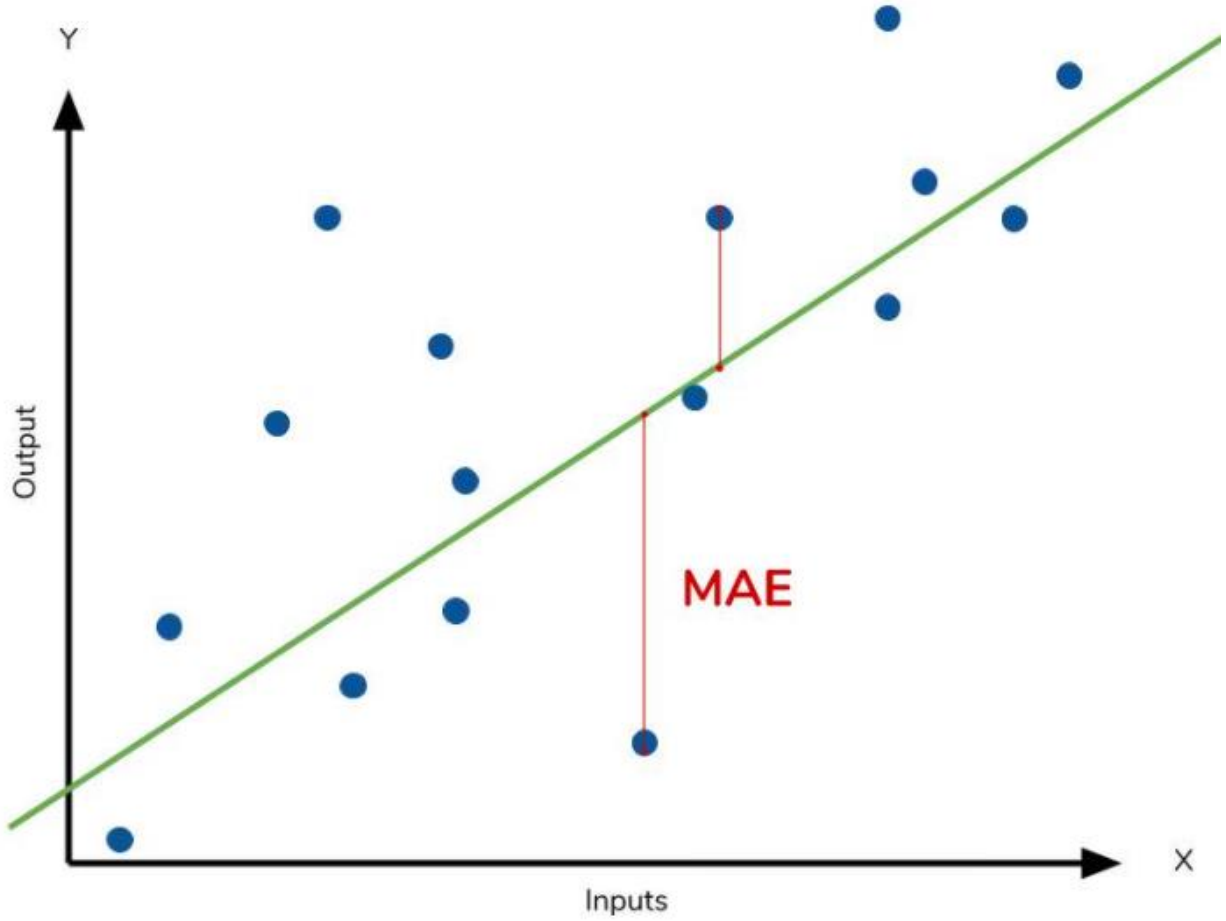
$$MSE = \frac{\sum(y - \hat{y})^2}{n}$$



2.2.2 성능 측정 지표 선택

- MAE(평균절대오차)

$$MAE = \frac{\sum |y - \hat{y}|}{n}$$



2.2.2 성능 측정 지표 선택

1. MAE

- 모델의 예측값과 실제값의 차이를 모두 더한다는 개념
- 절대값을 취하기 때문에 가장 직관적으로 알 수 있는 지표이다.
- MSE 보다 특이치에 둔감하다.

2. MSE

- 제곱을 하기 때문에 MAE와는 다르게 모델의 예측값과 실제값 차이의 면적의 합이다. 이런 차이로 특이값이 존재하면 수치가 많이 늘어난다.
- 특이치에 민감하다 (큰 이상치에 대해 크게 패널티를 주어 특이값을 인지하기 쉽다)

2.2.2 성능 측정 지표 선택

- Norm :
  ① 벡터의 길이 혹은 크기를 측정하는 방법(함수)

$$L_p = \left(\sum_i^n |x_i|^p\right)^{\frac{1}{p}}$$

  ② 상수 p의 값에 따라 L1 norm, L2 norm으로 구분

  ③ Norm 값이 클수록 큰 값의 원소에 치우치며 작은 값은 무시된다

  = 이상치가 큰 값이 있으면 그에 따라 값이 많이 휘둘린다

  (이상치 작은 값을 덜 반영하게 됨)

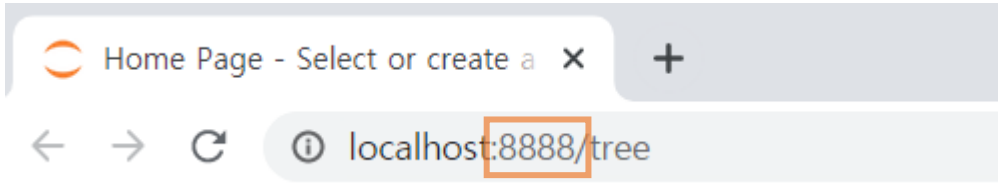
2.2.3 가정 검사

- 생략

2.3.1 작업환경 만들기

```
(base) C:₩Users₩iryun>jupyter notebook C:₩Users₩iryun
[I 21:06:03.572 NotebookApp] JupyterLab extension loaded from C:₩Users₩iryun₩Anaconda3₩lib₩site-packages₩jupyterlab
[I 21:06:03.587 NotebookApp] JupyterLab application directory is C:₩Users₩iryun₩Anaconda3₩share₩jupyter₩lab
[I 21:06:03.587 NotebookApp] Serving notebooks from local directory: C:₩Users₩iryun
[I 21:06:03.587 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 21:06:03.587 NotebookApp] http://localhost:8888/?token=cddf8fee1862f09fa59e2b46f9ac9a77a36af24a76e54a89
[I 21:06:03.587 NotebookApp]  or http://127.0.0.1:8888/?token=cddf8fee1862f09fa59e2b46f9ac9a77a36af24a76e54a89
[I 21:06:03.587 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 21:06:03.650 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/iryun/AppData/Roaming/jupyter/runtime/nbserver-5484-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=cddf8fee1862f09fa59e2b46f9ac9a77a36af24a76e54a89
     or http://127.0.0.1:8888/?token=cddf8fee1862f09fa59e2b46f9ac9a77a36af24a76e54a89
```



주피터 노트북이 127.0.0.1:8888 주소로 웹을 통해 접근하도록 설정
➔ 저 주소를 입력하기만 하면 어느 PC에서든 내 jupyter notebook을 쓸 수 있음

2.3.2 데이터 다운로드

In [1]:
```
import os
import tarfile
from six.moves import urllib
```

In [2]:
```
DOWNLOAD_ROOT="https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH=os.path.join("datasets","housing") #경로를 병합하여 새 경로 생성 ->datasets/housing
HOUSING_URL=DOWNLOAD_ROOT+"datasets/housing/housing.tgz"
```

In [3]:
```
def fetch_housing_data(housing_url=HOUSING_URL,housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path): #datasets/housing이 없으면 새로 생성
        os.makedirs(housing_path)
    tgz_path=os.path.join(housing_path,"housing.tgz") #datasets/housing 있으면 거기에 저장
    urllib.request.urlretrieve(housing_url,tgz_path) #지정 파일을 지정 경로에 저장
    housing_tgz=tarfile.open(tgz_path) #tar 파일 오픈
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

In [4]:
```
fetch_housing_data()
```

In [5]:
```
import pandas as pd

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

2.3.3 데이터 구조 훑어보기

In [7]:
```
housing=load_housing_data()
housing.head()
```

Out[7]:

	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 [8]:
```
housing.info()
```

```
<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
```

- total_bedrooms만 207개의 결측치 보유
- ocean_proximity만 object 타입 -> 데이터를 csv에서 읽었으므로 텍스트 특성일 것
  & head에서 중복된 값들이 나오는 것으로 텍스트로 된 범주형 데이터일 것

In [9]:
```
housing["ocean_proximity"].value_counts()
```

Out[9]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

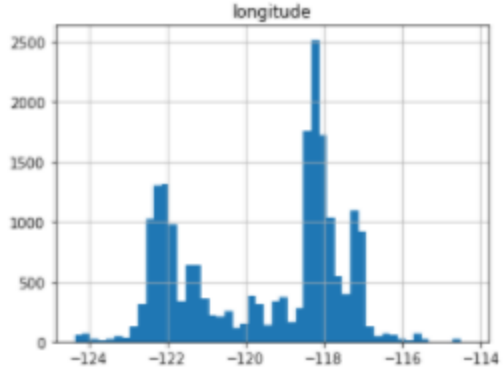
- 4개 종류의 범주형 데이터였음

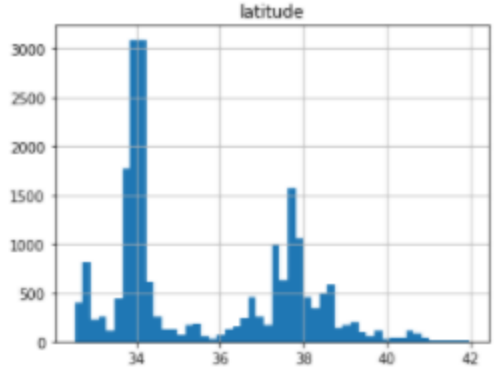
In [10]:
```
housing.describe()
```

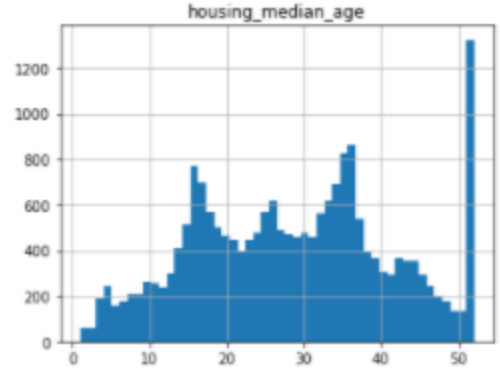
Out[10]:

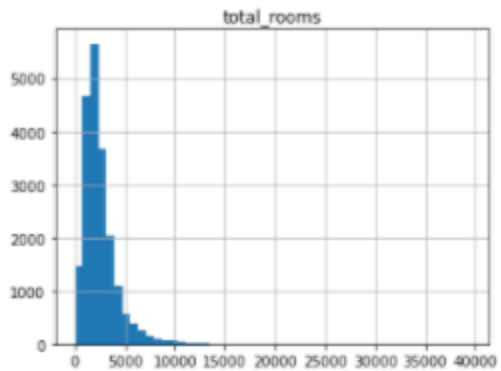
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	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

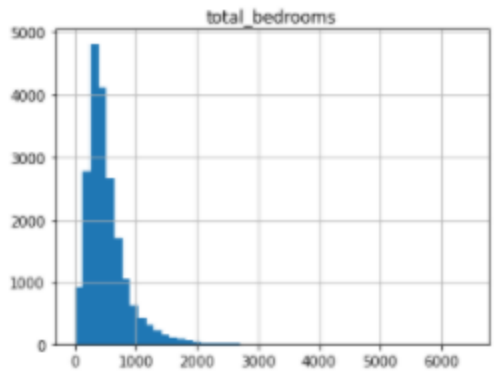
In [11]:
```
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15)) #bins: 구간 수
plt.show()
```

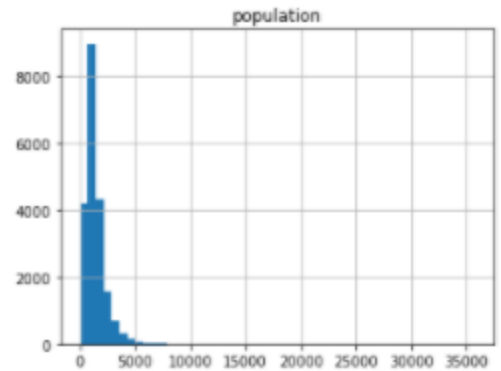


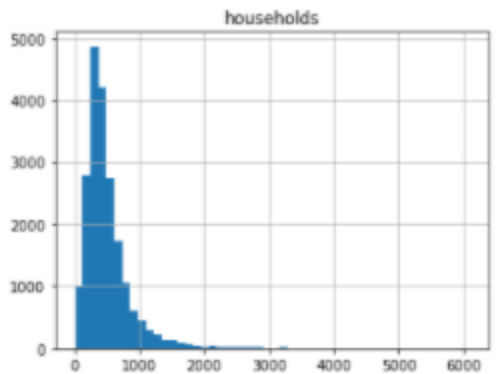


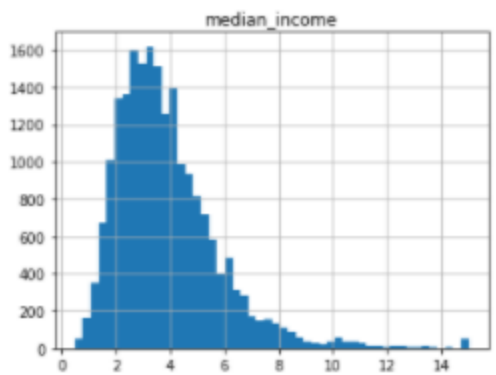


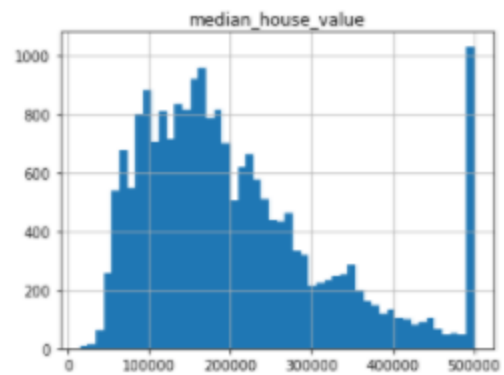












- median income의 단위 확인 필요
- housing median age, median house value는 특정 값 이상은 하나의 범주로 묶어 처리 했음
- 스케일링 필요
- 각 데이터의 왜도가 크다 (positive skewness)

2.3.4 테스트 세트 만들기

Case 1. 일정 비율 데이터를 무작위로 구분

In [12]:
```
import numpy as np

def split_train_test(data,test_ratio):
    shuffled_indices=np.random.permutation(len(data)) #shuffle과 permutation 차이
    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]
```

In [13]:
```
train_set,test_set=split_train_test(housing,0.2)
print(len(train_set),"train +",len(test_set),"test")
```

```
16512 train + 4128 test
```

- 코드 돌릴 때마다 테스트셋이 달라짐 ➔ 코드를 많이 돌리면 테스트 셋에 전체 데이터가 들어가므로 데이터 스누핑이 생길 수 있음

- 해결책

1. 테스트 셋 구분 후 저장 ➔ 다음 코드 실행때 구분된 셋 불러오기
2. 테스트 셋 불러올때마다 같은 데이터들을 불러올 수 있게 함

➔ 두 해결책의 단점 : 데이터 셋이 업데이트 되면 해당 데이터들에 대해 따로 또 처리해야됨

shuffle과 permutation의 차이

In [1]:
```
import numpy as np
```

In [2]:
```
a=[1,2,3,4,5]
b=np.random.permutation(a)
print(b)
```

```
[4 5 2 3 1]
```

In [3]:
```
print(a)
```

```
[1, 2, 3, 4, 5]
```

In [4]:
```
c=[1,2,3,4,5]
np.random.shuffle(c)
print(c)
```

```
[3, 5, 1, 2, 4]
```

- permutation : 셔플 된 array를 만들어준다
- shuffle : array를 셔플하여 반환한다

Case 2. 데이터의 고유 식별자로 구분

ex. 각 데이터 식별자의 해시값 계산해 마지막 bite 값이 51 이하면 테스트 셋

- 해시값 : 해시 함수(hash function)는 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수.
- 해시 : 해시 함수에 의해 얻어지는 값. 해시 값, 해시 코드, 해시 체크섬 또는 간단하게 해시라고 한다.

In [14]:
```
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))
    #앞서 만들었던 test_set_check 함수의 identifier를 lambda의 x로 받아와 함수 적용
    #리턴 된 test set 여부를 in_test_set으로 받아와 해당 데이터에 적용
    return data.loc[~in_test_set],data.loc[in_test_set]
```

In [15]:
```
#데이터 셋에 식별자 컬럼이 없으므로 각 행의 index를 ID로 사용
housing_with_id=housing.reset_index()
train_set,test_set=split_train_test_by_id(housing_with_id,0.2,"index")
```

In [16]:
```
#또는 불변하는 데이터들을 조합하여 ID를 만든다
housing_with_id["id"]=housing["longitude"]*1000+housing["latitude"]
train_set, test_set=split_train_test_by_id(housing_with_id,0.2,"id")
```

Case 2. 데이터의 고유 식별자로 구분

crc32(np.int64(identifier))&0xffffffff<test_ratio2*32 은 무슨 소리인가?

test_ratio * 2**32

2^32 : 16진수에서 32비트로 표현할 수 있는 정수보다 1만큼 더 큰 수
만약 0.2라면 0.2x2^32 = 51 이므로 부여한 인덱스가 51보다 낮은 데이터들을 테스트 셋으로 지정

np.int64(identifier)

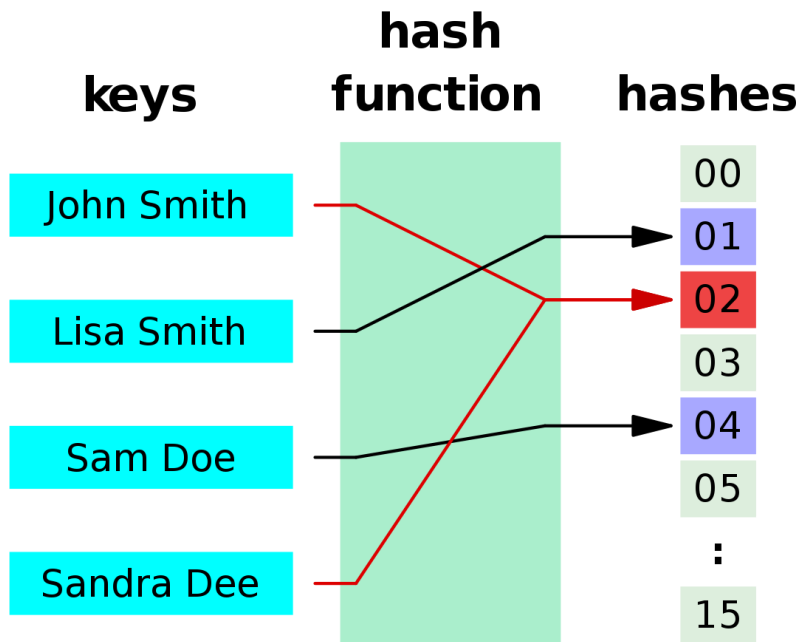
iedntifier : 각 데이터에서 index에 해당하는 정수(int)
해당 코드에서 이를 numpy.int64로 변환

crc32(np.int64(identifier))

zlib.crc32.crc32() method : 파라미터로 넘겨받은 데이터로 CRC체크섬을 생성하는 해시함수
해당 코드에서는 numpy.int64 형태의 데이터를 받아 [0,2**32-1] 범위의 정수 해시값 생성
➔ 식별자 순서와 상관 없이 랜덤한 32bit의 정수 데이터를 얻게 됨

crc32(np.int64(identifier))&0xffffffff

파이썬 버전 2와의 호환성을 위해 추가된 코드
파이썬 3.0은 crc32에서 부호 없는 32비트 정수를 반환
파이썬 2.0은 crc32에서 부호를 가지는 32비트 정수 반환
➔ 부호 비트 처리를 위해 &0xffffffff연산 실행



Case 3. 사이킷런 사용으로 구분

- train_test_split

1. 난수 초깃값 지정하는 random_state 매개변수의 존재
2. 리스트, 시리즈, 배열 등의 다양한 데이터 타입들의 행의 개수가 같다면 같은 인덱스를 기반으로 처리 가능

In [17]:
```
from sklearn.model_selection import train_test_split

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

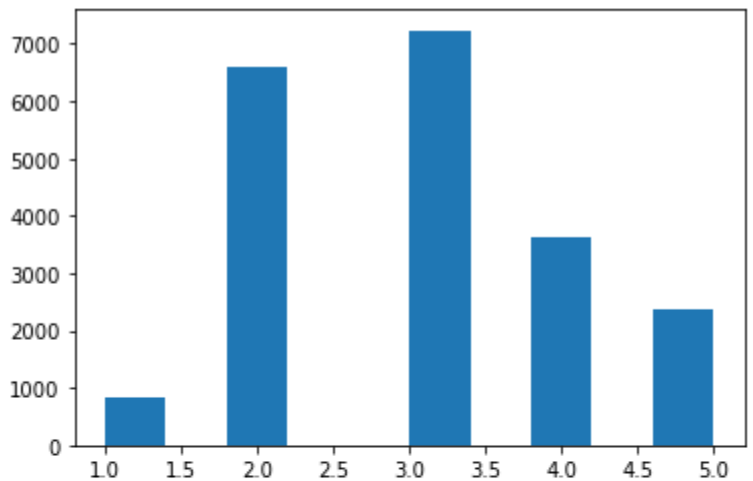
Case 4. 계층적 샘플링의 필요성

가정 : 중간 소득이 중간 주택 가격 예측에 큰 영향을 미친다.

In [18]:
```
#특정 구간으로 나눠 범주화시킨 후 해당 구간에서 일정 비율로 샘플링 한다
housing["income_cat"]=np.ceil(housing["median_income"]/1.5)
housing["income_cat"].where(housing['income_cat']<5,5.0,inplace=True)
```

In [19]:
```
plt.hist(housing["income_cat"])
```

Out[19]:
```
(array([ 822.,    0., 6581.,    0.,    0., 7236.,    0., 3639.,    0.,
        2362.]),
 array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6, 5. ]),
 <BarContainer object of 10 artists>)
```



In [20]:
```
from sklearn.model_selection import StratifiedShuffleSplit

#StratifiedShuffleSplit는 층화추출 후 인덱스를 반환한다.
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]
```

In [21]:
```
housing['income_cat'].value_counts()/len(housing)
```

Out[21]:
```
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
```

In [22]:
```
strat_train_set['income_cat'].value_counts()/len(strat_train_set)
```

Out[22]:
```
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
```

In [23]:
```
strat_test_set['income_cat'].value_counts()/len(strat_test_set)
```

Out[23]:
```
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
```