

Chap2. 머신러닝 프로젝트 처음부터 끝까지

Seolyoung Jeong, Ph.D.

경북대학교 IT 대학

가상의 프로젝트 진행

2.2 큰그림 보기

- 2.2.1 문제정의
- 2.2.2 성능 측정 지표 선택
- 2.2.3 가정 검사

2.3 데이터 가져오기

- 2.3.1 작업 환경 만들기
- 2.3.2 데이터 다운로드
- 2.3.3 데이터 구조 훑어보기
- 2.3.4 테스트 세트 만들기

2.4 데이터 탐색 및 시각화

- 2.4.1 지리적 데이터 시각화
- 2.4.2 상관관계 조사
- 2.4.3 특성 조합으로 실험

2.5 데이터 준비

- 2.5.1 데이터 정제
- 2.5.2 텍스트와 범주형 특성 다루기
- 2.5.3 나만의 변환기
- 2.5.4 특성 스케일링
- 2.5.5 변환 파이프라인

2.6 모델 선택 및 훈련

- 2.6.1 훈련 세트에서 훈련하고 평가하기
- 2.6.2 교차 검증을 사용한 평가

2.7 모델 세부 튜닝

- 2.7.1 그리드 탐색
- 2.7.2 랜덤 탐색
- 2.7.3 앙상블 방법
- 2.7.4 최상의 모델과 오차 분석
- 2.7.5 테스트 세트로 시스템 평가하기

◆ 솔루션 제시

2.8 시스템 론칭, 모니터링, 유지보수

2.2 큰그림 보기

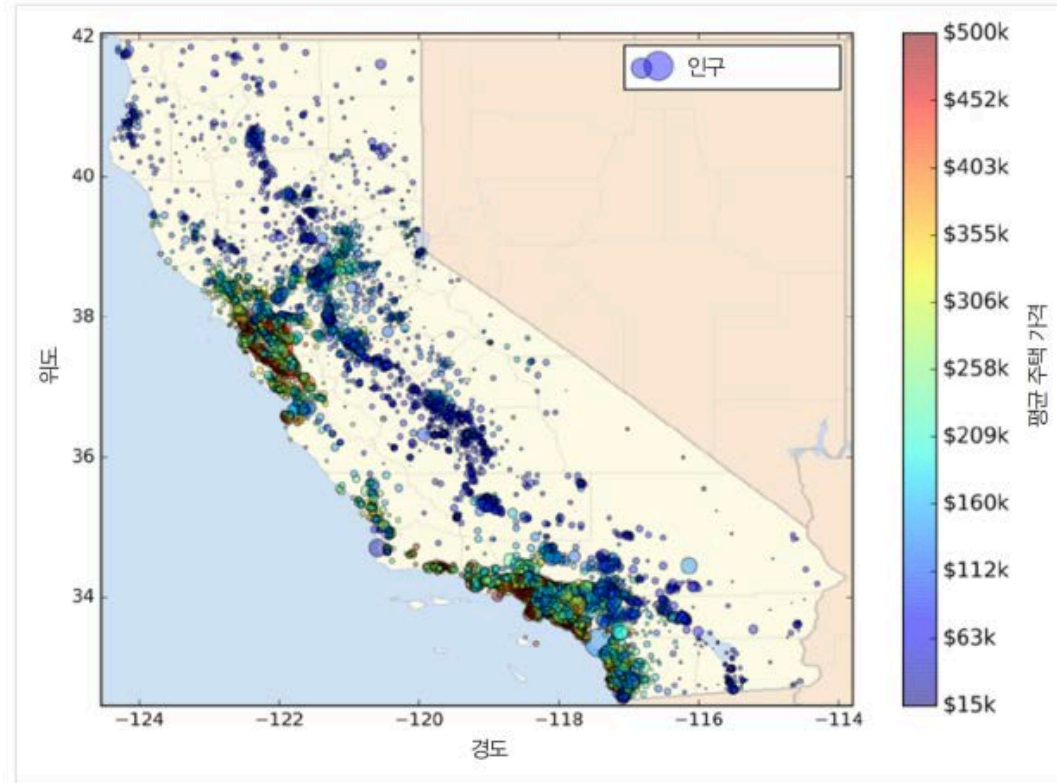
◆ 프로젝트 목적

- 캘리포니아 인구조사 데이터를 사용
- **캘리포니아 주택 가격 모델 작성**

◆ 캘리포니아 인구조사 데이터

- 캘리포니아 블록 그룹별 (600~3,000명 인구)
- 인구, 중간 소득, 중간 주택 가격 포함

그림 2-1 캘리포니아 주택 가격



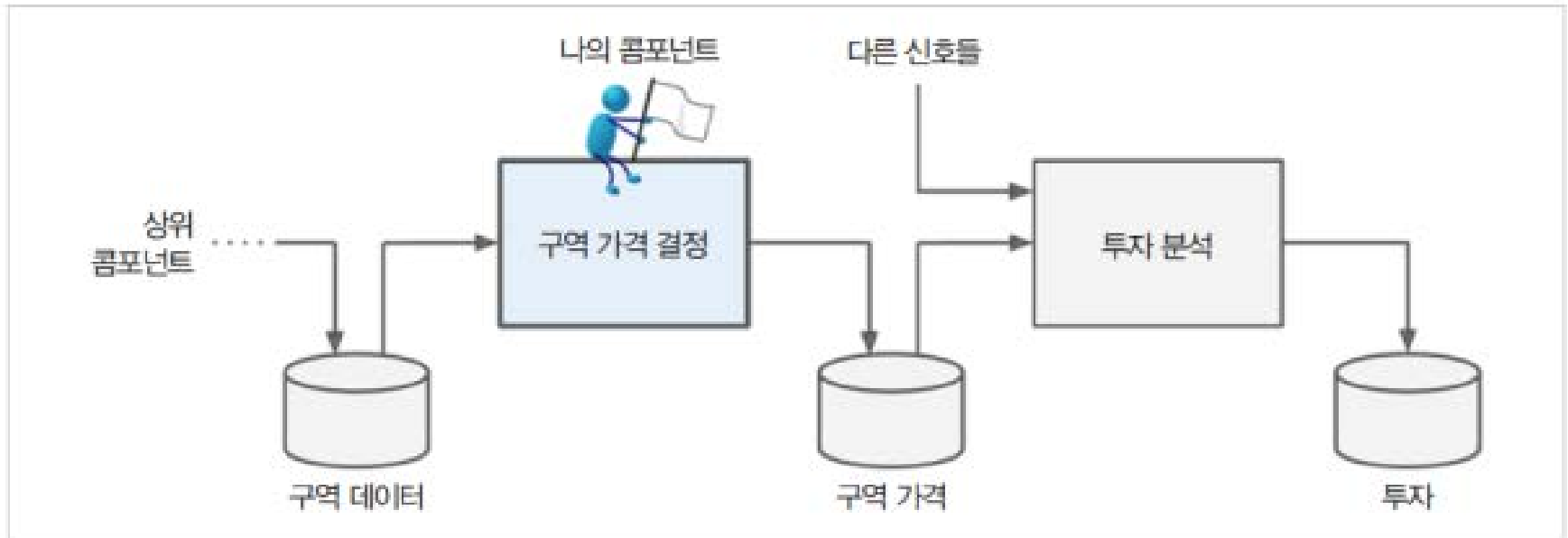
2.2.1 문제 정의 - 비즈니스 목적

◆ 비즈니스 목적 파악

- 시스템 구성, 알고리즘, 측정 지표, 튜닝 시간 결정

◆ 중간 주택 가격 예측 출력 → 투자 결정 머신러닝 시스템 입력

그림 2-2 부동산 투자를 위한 머신러닝 파이프라인



2.2.1 문제 정의

◆ 현재 솔루션?

- 전문가가 수동으로 추정 (10%이상 오류)

◆ 문제 정의

- 지도/비지도/강화학습 ?
- 분류/회귀학습 ?
- 배치/온라인학습 ?

- 레이블된 샘플 있음 → 지도학습
- 연속된 값을 예측 → 회귀학습 (여러 특성 : multivariate regression)
- 오프라인, 비교적 느리고, 크지 않은 데이터 → 배치학습

2.2.2 성능 측정 지표

◆ 전형적인 회귀문제 성능 지표

- 평균 제곱근 오차 (Root Mean Square Error : RMSE)
 - 오차가 커질 수록 RMSE 값은 커짐 (예측에 많은 오류가 있음)

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

- 평균 절대 오차 (Mean Absolute Error)
 - 이상치로 보이는 경우가 많은 경우 사용

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

- \mathbf{X} : 데이터셋에 있는 모든 샘플의 모든 특성값을 포함하는 행렬
- h : 시스템의 예측 함수 (가설 : hypothesis)
- m : 데이터셋에 있는 샘플 수
- $y^{(i)}$: 샘플 $\mathbf{x}^{(i)}$ 에 대한 예측값

2.2.2 성능 측정 지표

◆ 예측값과 타깃값 사이의 거리를 재는 방법 (norm)

- Euclidian norm : RMSE
- Manhattan norm : MAE

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

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

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

- k가 클수록 큰 원소에 치우치며 작은 값은 무시
- RMSE가 MAE보다 이상치에 더 민감
- 이상치가 매우 드물면 RMSE가 잘 맞아 일반적으로 널리 사용
(예: 종 모양 분포의 양 끝단)

표기법

◆ 어떤 구역의 값

- 경도 -118.29, 위도 33.91
- 주민수 1,416명
- 중간소득 \$38,372
- 중간 주택 가격 \$156,400

$$\mathbf{x}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1,416 \\ 38,372 \end{pmatrix} \quad y^{(1)} = 156,400$$

◆ 데이터셋의 모든 샘플 값

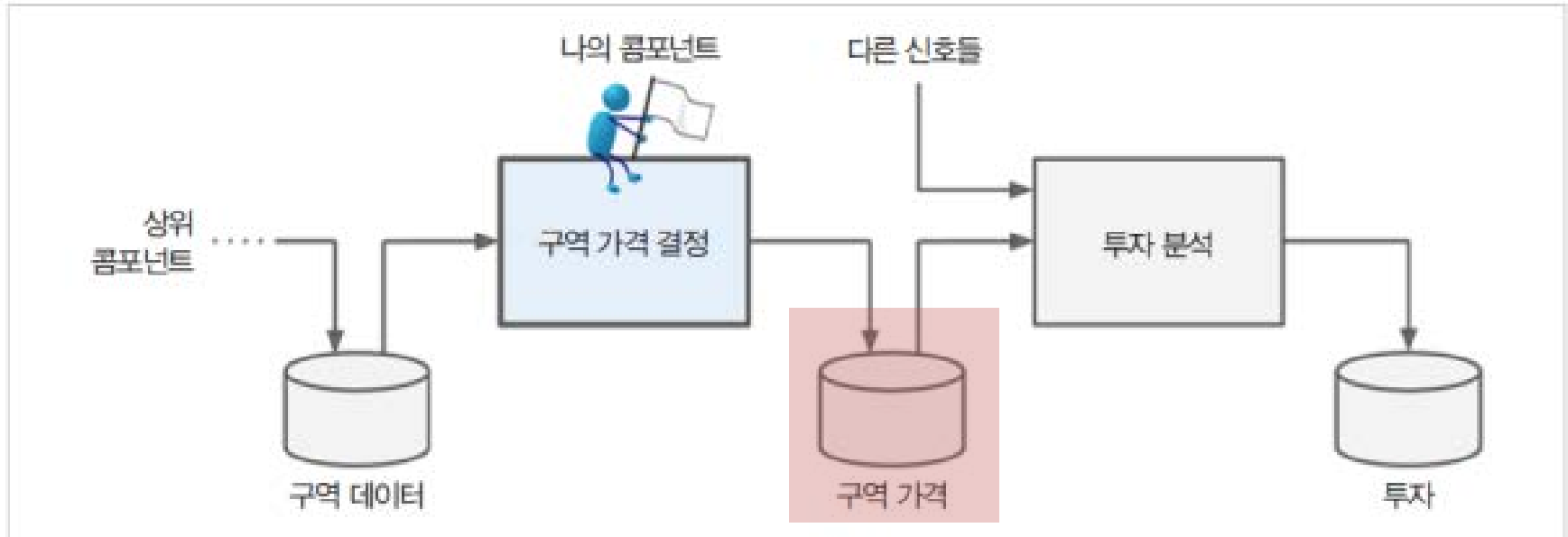
$$X = \begin{pmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(1999)})^T \\ (\mathbf{x}^{(2000)})^T \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

◆ 하나의 샘플에 대한 예측값

$$\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$$

2.2.3 가정 검사

◆ 지금까지 만든 가정을 나열하고 검사



- 만약, 중간 주택 가격이 아닌 등급(저렴/보통/고가)을 원한다면?
→ 회귀가 아닌, 분류작업이 필요

◆ 너무 늦게 문제를 발견하지 않도록 주의...

2.3.1 작업환경 만들기

◆ 필요한 python package

- numpy, pandas, matplotlib
- jupyter notebook
- scikit-learn
- tensorflow
- ...

◆ Anaconda package

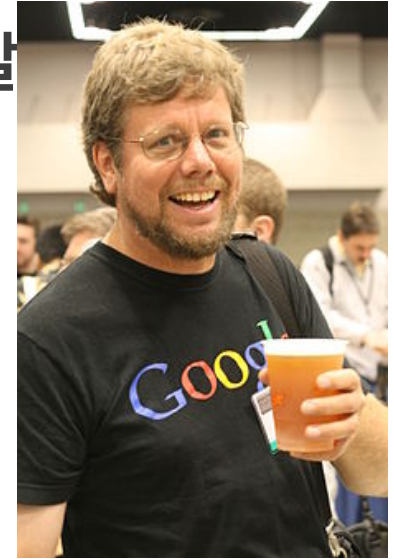
- 300개 이상 모듈 포함

◆ Anaconda 설치 후 업데이트 확인 할 것

- Anaconda Prompt
 - > conda update scikit-learn
 - > conda update pnadas

파이썬(Python)이란?

- ◆ 1991년 귀도 반 로섬(Guido Van Rossum)이 발
- ◆ 플랫폼 독립적인 인터프리터 언어이며,
- ◆ 객체 지향적, 동적 타이핑 대화형 언어
- ◆ 처음 C언어로 구현되었음



Guido Van Rossum



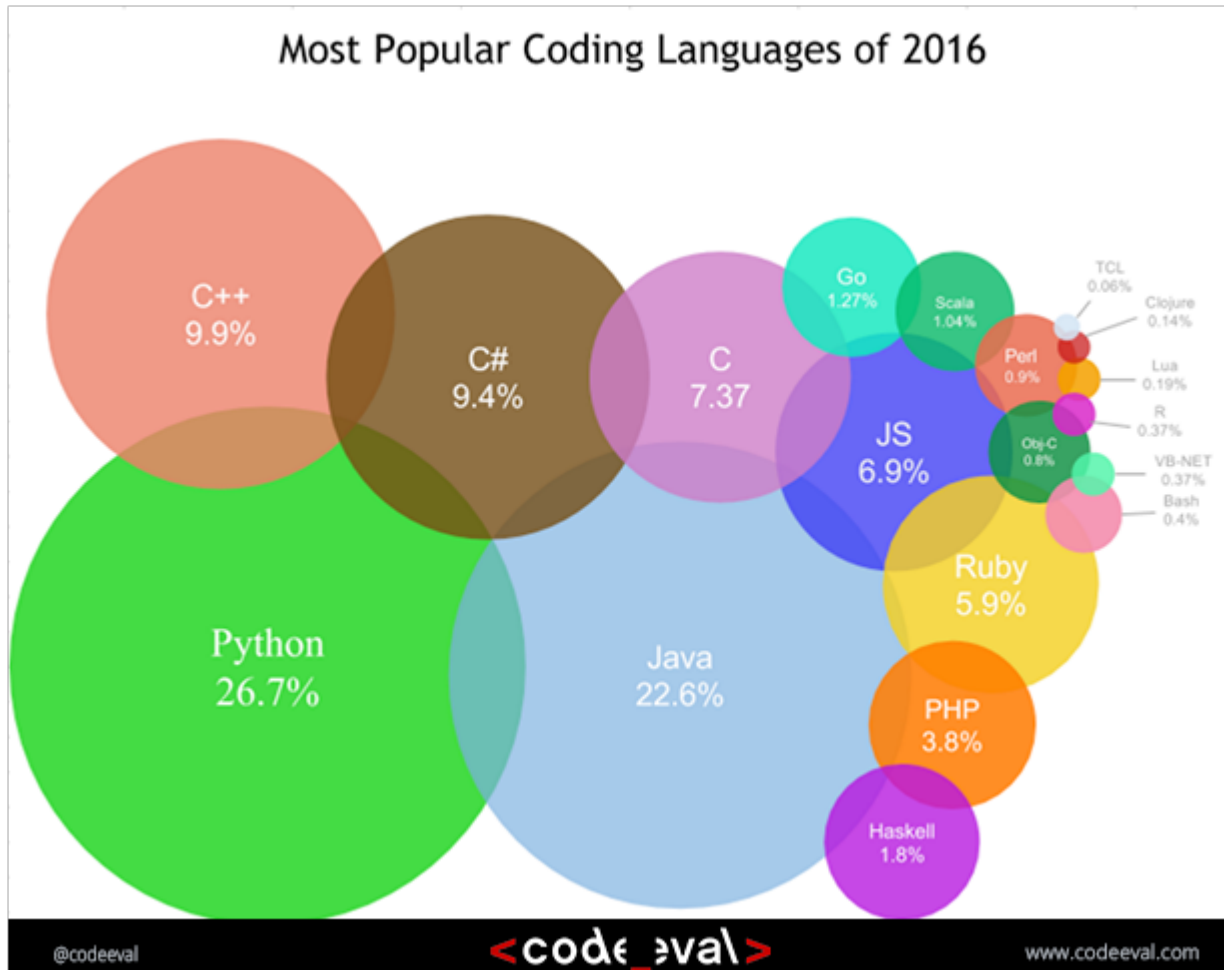
Python Logo

파이썬은 ‘피톤’이라는 이름으로 알려진, 고대 그리스 신화에 나오는 거대한 뱀의 이름. 피톤은 Python을 고대 그리스어로 읽은 것이며, 영어를 그대로 읽으면 ‘파이선’이 됨.

사실, 파이썬이라는 이름은 파이썬을 만든 귀도 반 로섬(Guido van Rossum)이 자신이 좋아하는 영국 코미디 프로인 ‘몬티 파이선의 날아다니는 서커스(Monty Python’s Flying Circus)’에서 따왔다고 함. 물론 여기서의 파이선도 피톤을 의미.

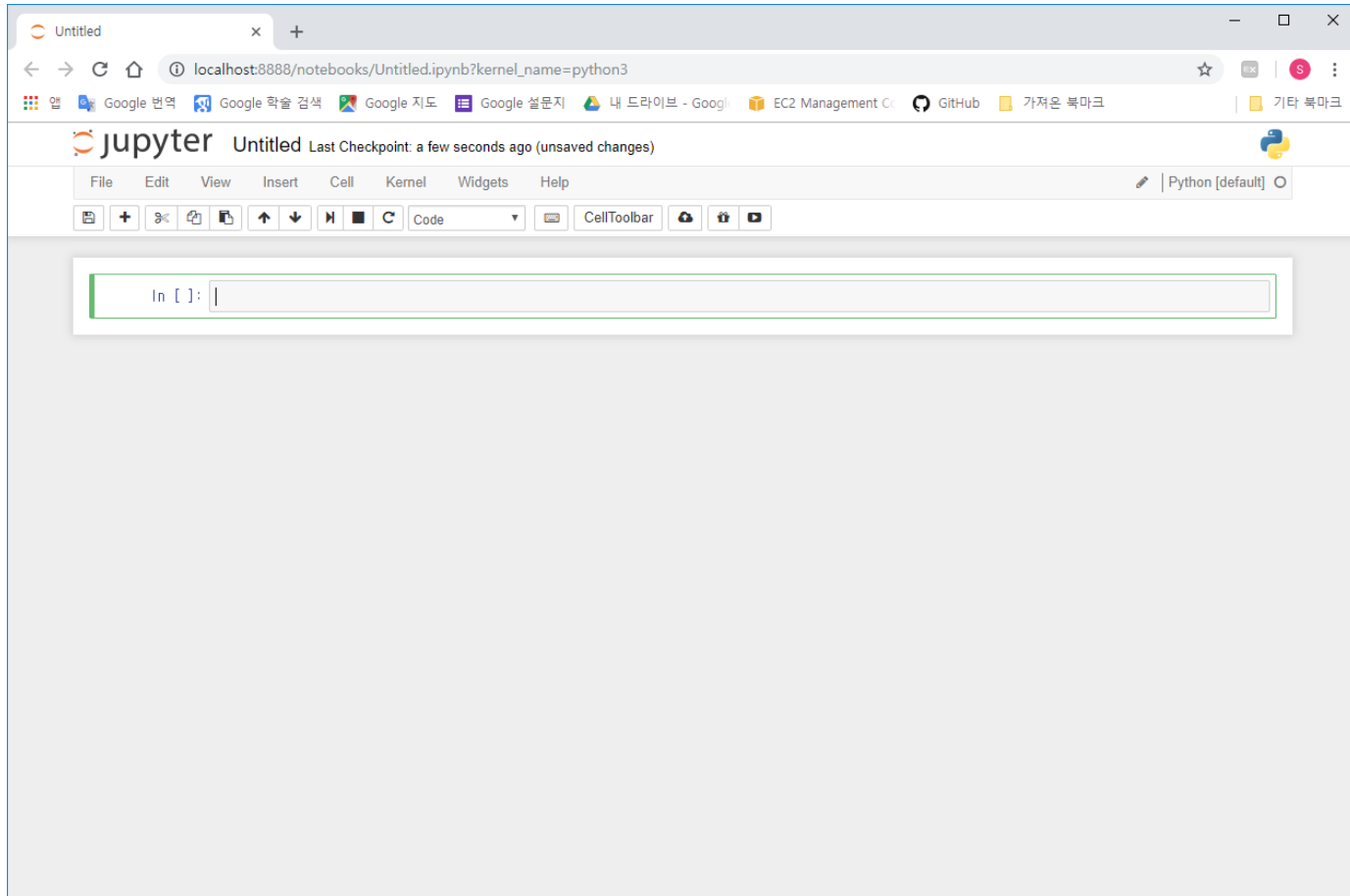
왜 파이썬을 배우는가?

◆ 가장 많이 사용되고 있는 언어 (2016년)



Jupyter Notebook

- 웹브라우저 기반 툴 (<http://localhost:8888>)
- Python 외에 Ruby, R, JavaScript와 같은 다른 언어도 사용 가능



2.3.2 데이터 다운로드

- ◆ download/datasets/housing/ 폴더에
- ◆ housing.tgz 압축 파일 다운로드
- ◆ 압축해제 → housing.csv 파일

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

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

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

```
fetch_housing_data()
```

pandas로 csv 파일 읽기

◆ 10개의 특성

```
In [5]: 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)
```

```
In [6]: 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
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR

2.3.3 데이터 구조 훑어보기

◆ housing.info()

- 총 20,640개 샘플
- total_bedrooms 특성은 20,433개만 널값이 아님 (207개 비어 있음)
- ocean_proximity 필드 데이터 타입
 - object (문자열), 카테고리 (예: NEAR BAY)

In [7]: 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: 1.5+ MB
```


◆ ocean_proximity 필드 내 카테고리 확인

```
In [8]: housing["ocean_proximity"].value_counts()
```

```
Out [8]: <1H OCEAN      9136  
INLAND          6551  
NEAR OCEAN      2658  
NEAR BAY        2290  
ISLAND           5  
Name: ocean_proximity, dtype: int64
```

◆ 숫자형 특성 요약 정보

```
In [9]: housing.describe()
```

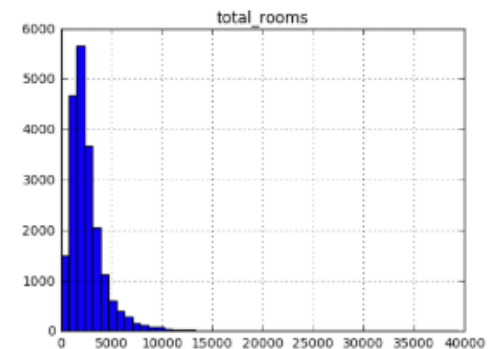
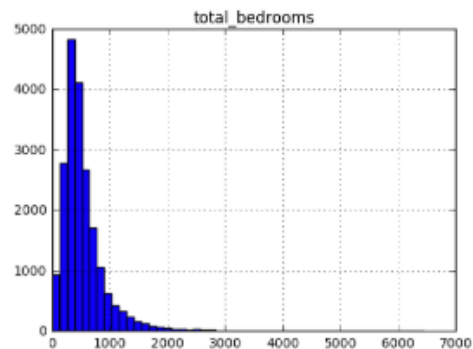
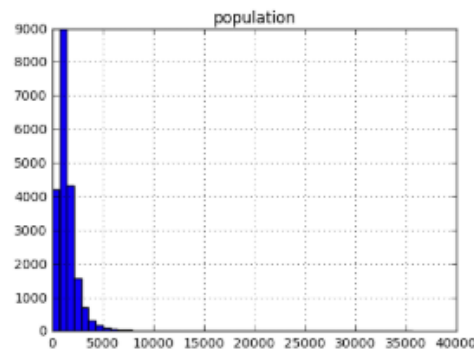
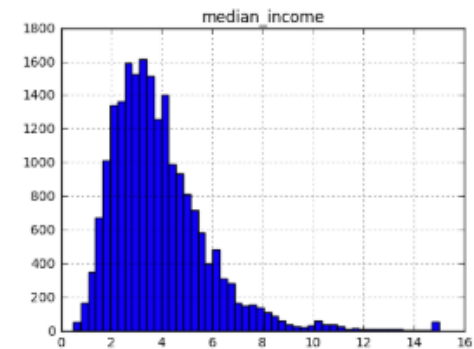
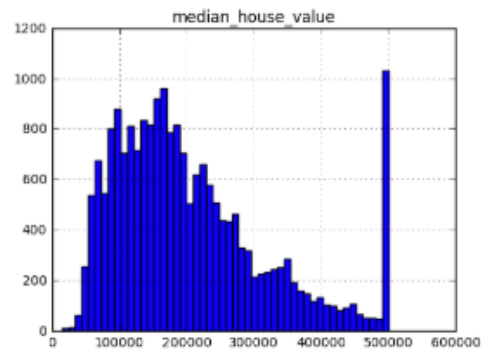
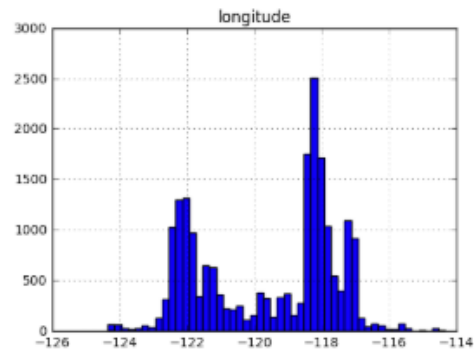
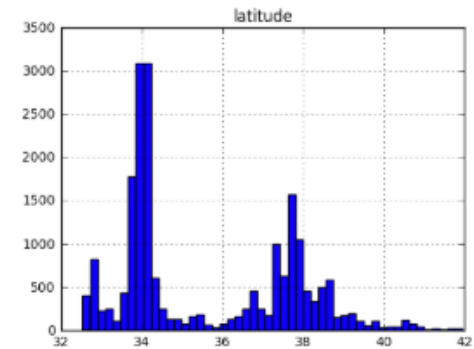
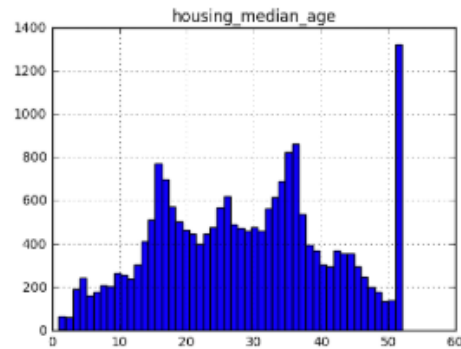
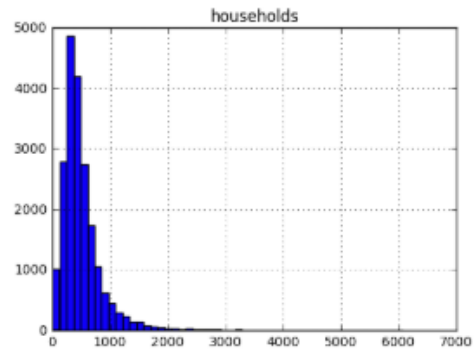
```
C:\Users\snowflower\Anaconda3\lib\site-packages\numpy\lib\function_base.py:3834: RuntimeWarning: Invalid value encountered in percentile  
RuntimeWarning)
```

```
Out [9]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.00
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.8
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.6
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.00
25%	-121.800000	33.930000	18.000000	1447.750000	NaN	787.000000	280.000000	2.563400	119600.0
50%	-118.490000	34.260000	29.000000	2127.000000	NaN	1166.000000	409.000000	3.534800	179700.0
75%	-118.010000	37.710000	37.000000	3148.000000	NaN	1725.000000	605.000000	4.743250	264725.0
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.0

matplotlib 히스토그램 그리기

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



2.3.4 테스트 세트 만들기

◆ 데이터로부터 테스트 세트 분리

- 전체 데이터에서 너무 많은 직관을 얻으면 과대적합된 모델이 생성될 가능성이 있음 (data snooping 편향)

◆ 무작위로 샘플을 선택해서 데이터셋의 20% 정도 분리

◆ 프로그램 재실행 시 이전의 데이터셋 불러와야...

- 이전 세트 저장 후 불러오기
- 동일한 난수 인덱스 생성하기

◆ 데이터셋 업데이트 후에도 적용 가능해야...

→ 식별자를 사용하여 테스트 세트로 결정

scikit-learn 테스트 세트 분리 함수

◆ 사이킷런 `train_test_split()` 함수

- 난수 초기값 지정 가능
- 동일 인덱스 기반 분리 가능

여러개의 배열을 넣을 수 있습니다
(파이썬 리스트, 넘파이, 판다스 데이터프레임)

`train_size`도 지정할 수 있습니다

```
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

◆ 계층적 샘플링

- 샘플에서도 전체 비율 유지해야 함

◆ 중간 주택 가격 예측 시 중간 소득이 중요

- 소득 카테고리 특성 중요
- 중간 소득 히스토그램 : 대부분 \$20,000~\$50,000 일부 \$60,000
- 계층별로 데이터셋에 충분한 샘플수가 필요

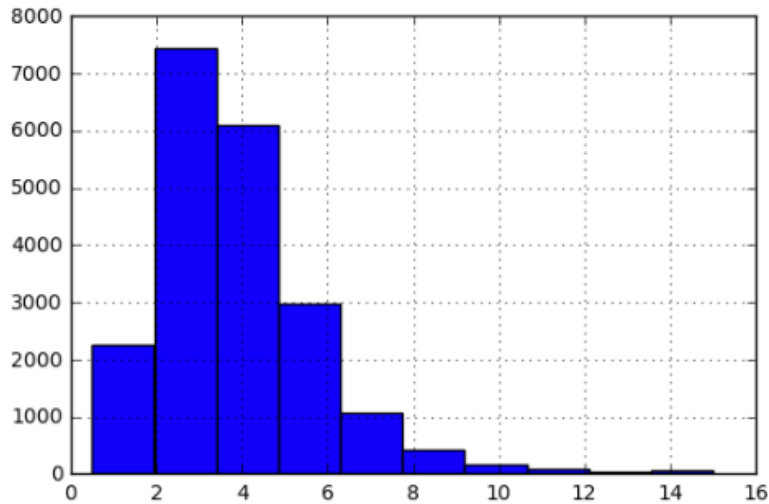


소득 카테고리 “income_cat”

- ◆ 소득 카테고리 개수 제한 : 1.5로 나눔
- ◆ 5이상은 5로 레이블

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

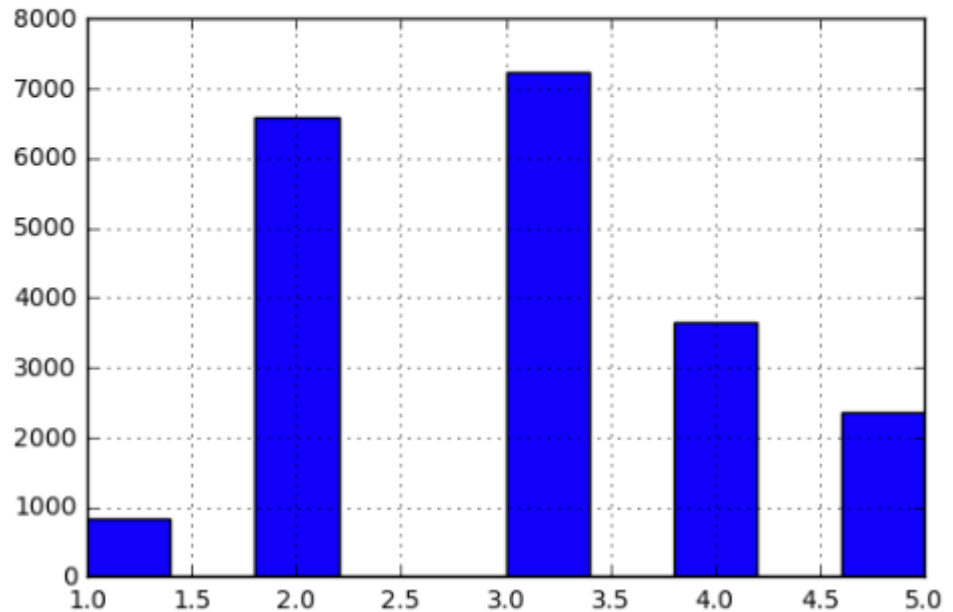
<matplotlib.axes._subplots.AxesSubplot at 0x8eb72b0>



```
import numpy as np
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

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

<matplotlib.axes._subplots.AxesSubplot at 0x678b9d0>



소득 카테고리 기반 계층 샘플링

◆ scikit-learn의 StratifiedShuffleSplit

- StratifiedKFold의 계층 샘플링 + ShuffleSplit의 랜덤 샘플링
- test_size와 train_size 매개변수 합을 1 이하로 지정 가능

```
In [22]: 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]
```

```
In [23]: housing["income_cat"].value_counts() / len(housing)
```

```
Out[23]: 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
```

2.4 데이터 이해를 위한 탐색과 시각화

◆ train_set 복사본 생성 후 사용

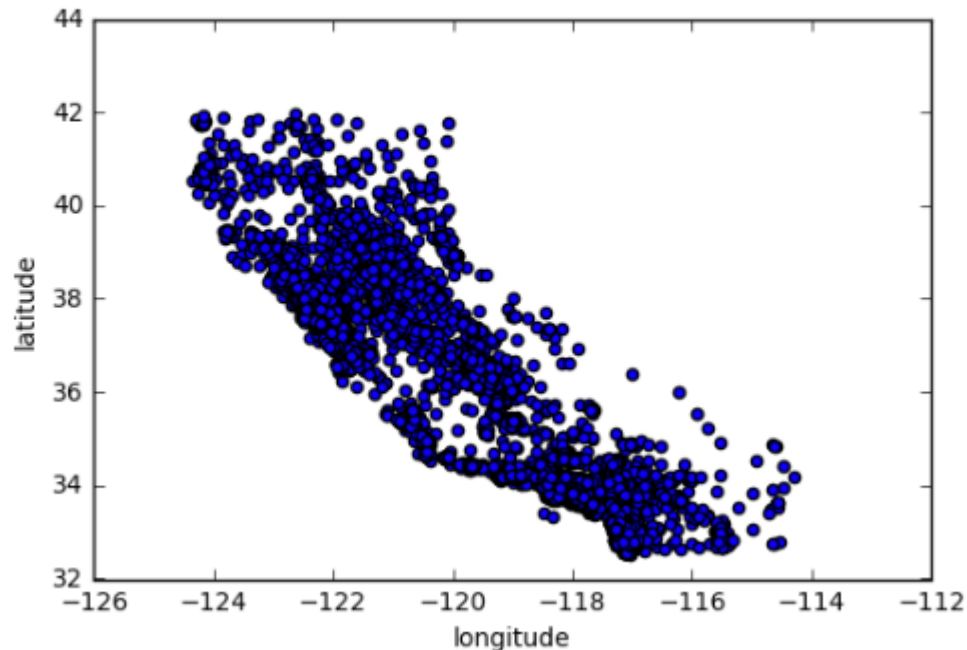
- `housing = strat_train_set.copy()`

◆ 2.4.1 지리적 데이터 시각화

- 모든 구역 산점도

```
In [26]: housing.plot(kind="scatter", x="longitude", y="latitude")
```

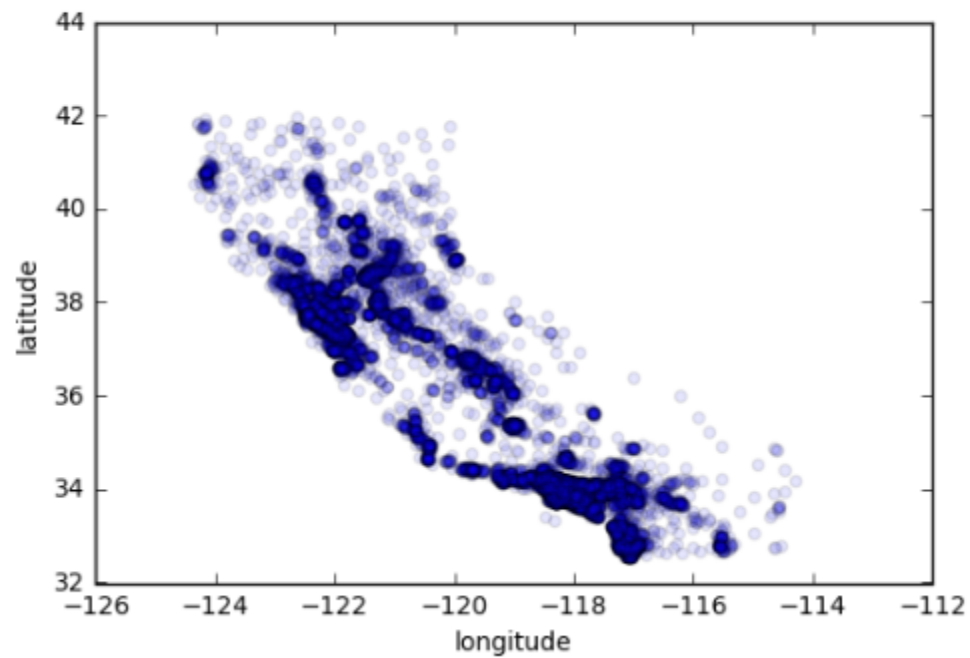
```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0xad2f250>
```



- 데이터 포인트가 밀집된 영역 : alpha option = 0.1

```
In [27]: housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

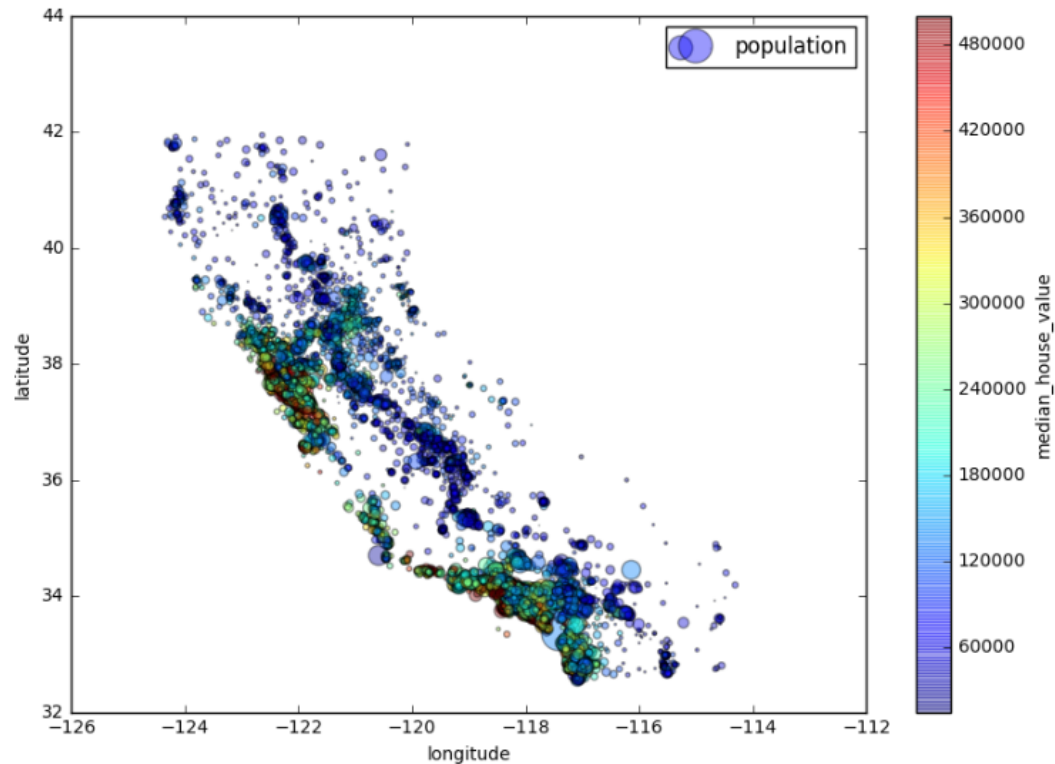
```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0xad73270>
```



- 캘리포니아 주택 가격
 - 원의 반지름 : 구역의 인구 (s)
 - 색깔 : 가격 (c)

```
In [28]: housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,  
s=housing["population"]/100, label="population", figsize=(10,7),  
c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True, sharex=False  
)  
plt.legend()
```

Out [28]: <matplotlib.legend.Legend at 0xa241dd0>



2.4.2 상관관계 조사

◆ 표준 상관관계수 (standard correlation coefficient)

- Pearson's r
- corr() 함수 이용

```
In [29]: corr_matrix = housing.corr()  
corr_matrix["median_house_value"].sort_values(ascending=False)  
  
Out [29]: 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
```

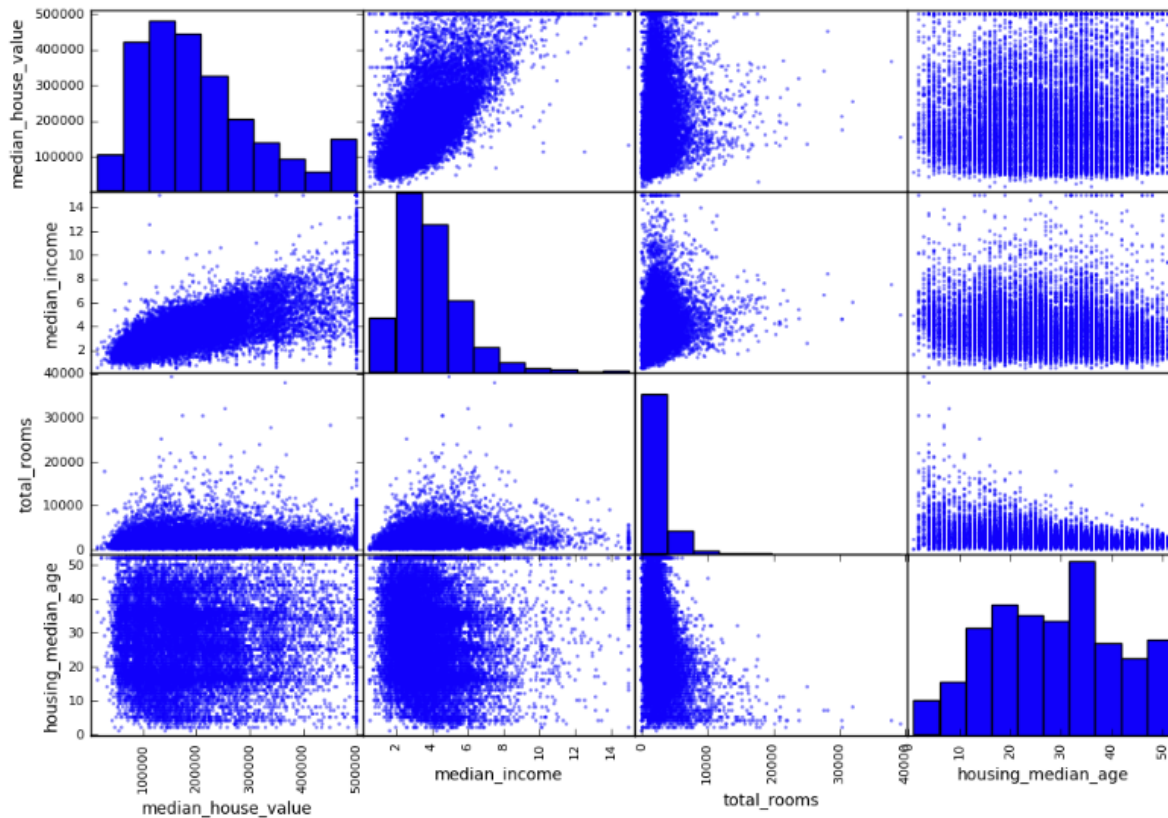
- 상관관계 범위 : -1 ~ 1
- 1에 가까우면 강한 양의 상관관계
 - 예) 중간 소득이 올라갈 때 주택 가격 증가
- -1에 가까우면 강한 음의 상관관계
 - 예) 위도가 커질수록(북쪽) 주택 가격이 조금씩 감소
- 0에 가까우면 선형적인 상관관계 없음

pandas – scatter matrix 함수

```
In [22]: from pandas.plotting import scatter_matrix
```

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

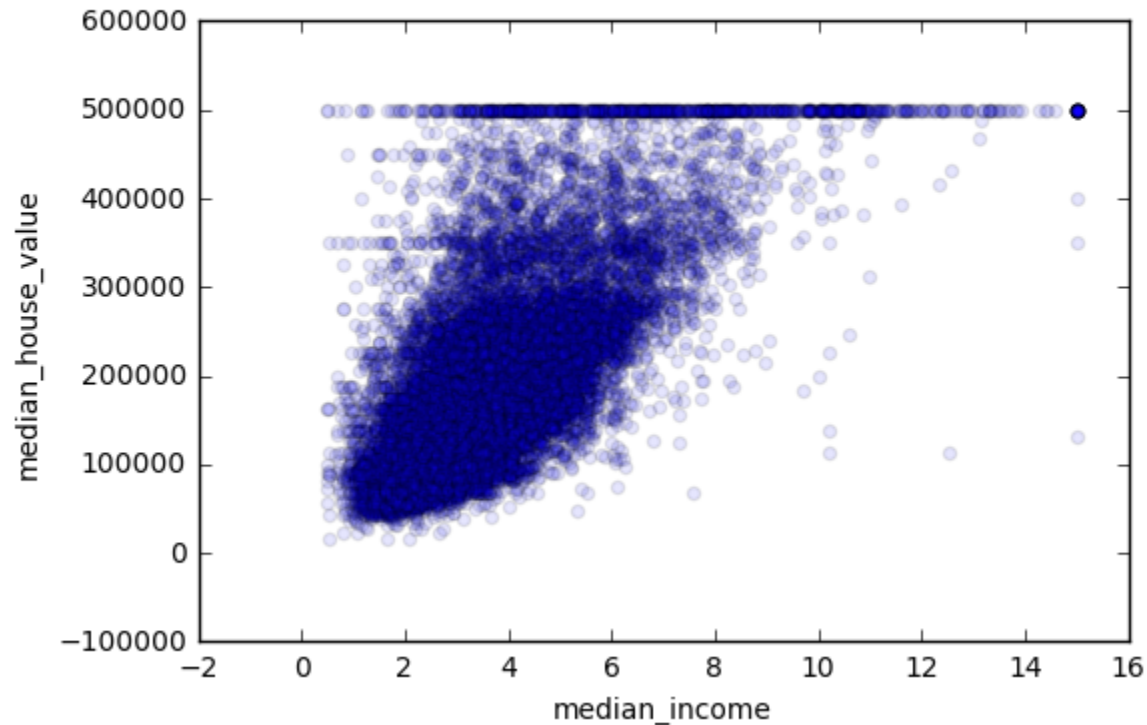
```
Out [22]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x08157690>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BA31030>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BA38170>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0B007330>],  
[<matplotlib.axes._subplots.AxesSubplot object at 0x0B00F590>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BE2B630>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BE36C50>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BB81930>],  
[<matplotlib.axes._subplots.AxesSubplot object at 0x0BB8AF30>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BED6C30>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BEE6910>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BF2DF30>],  
[<matplotlib.axes._subplots.AxesSubplot object at 0x0BF3BCF0>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BF89250>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BF91810>,  
<matplotlib.axes._subplots.AxesSubplot object at 0x0BFD0650>]], dtype=object)
```



중간 소득 vs. 중간 주택 가격

```
In [23]: housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
```

```
Out [23]: <matplotlib.axes._subplots.AxesSubplot at 0xc0e7bd0>
```



2.4.3 특성 조합

◆ 머신러닝 알고리즘 적용 전 확인

- 이상한 데이터 확인 → data refining 필요
- 상관관계 확인
- 여러 특성 조합 시도
 - 예) 방 개수 – 가구수
 - 예) 방 개수 – 침대 개수
 - 예) 가구 당 인원

```
In [24]: 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)
```

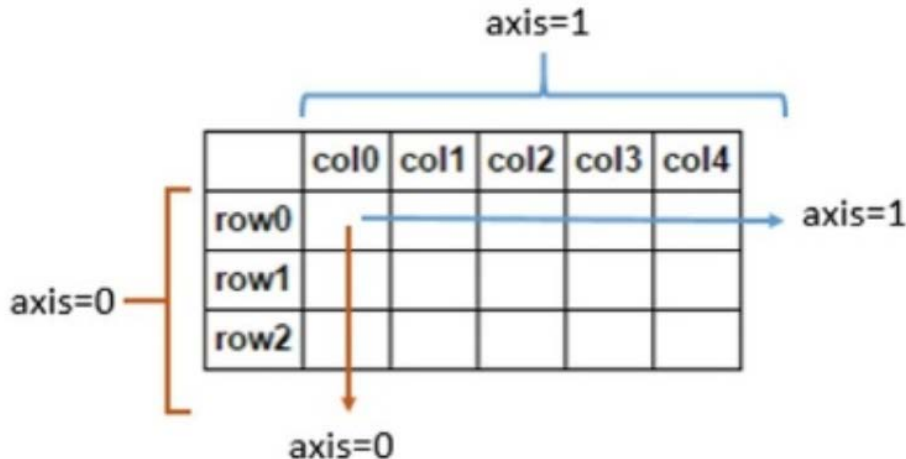
```
Out [24]: median_house_value      1.000000
median_income      0.687160
rooms_per_household 0.146285
total_rooms        0.135097
housing_median_age  0.114110
households         0.064506
total_bedrooms     0.047689
population_per_household -0.021985
population         -0.026920
longitude          -0.047432
latitude           -0.142724
bedrooms_per_room  -0.259984
Name: median_house_value, dtype: float64
```

2.5 머신러닝 알고리즘을 위한 데이터 준비

◆ 데이터 준비 자동화

- 데이터 변환 손쉽게 반복
- 다른 프로젝트에 재사용
- 론칭 후 새 데이터에 적용 시 사용
- 최적의 조합을 찾는데 편리

◆ train set 위해 lable 삭제 (axis=1)



2.5.1 데이터 정제

◆ 누락된 특성 처리 함수

- 예) total_bedrooms
- 옵션 1: 해당 구역 제거
- 옵션 2: 전체 특성 삭제
- 옵션 3: 어떤 값을 채우기 (0, 평균, 중간값 등)

◆ scikit-learn Imputer 함수

- 누락된 값 처리

◆ scikit-learn

- 일관성 : 추정기(fit함수), 변환기(transform함수), 예측기(predict, score)
- 검사 기능 : 모델 파라미터와 하이퍼파라미터를 공개 변수로 접근
- 클래스 남용 방지 : 기본 데이터 타입으로 numpy 배열 사용
- 조합성 : 기존의 구성요소 최대한 재사용 (파이프라인 클래스)
- 합리적인 기본값 : 모든 매개변수에 합리적인 기본값 지정

2.5.2 텍스트와 범주형 특성 다루기

◆ “ocean_proximity” 카테고리 텍스트 → 숫자로 변형

- 각 카테고리를 다른 정수값으로 매핑 : pandas의 factorize() 함수

```
In [37]: housing_cat = housing["ocean_proximity"]  
housing_cat.head(10)
```

```
Out [37]: 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
```

```
In [38]: housing_cat_encoded, housing_categories = housing_cat.factorize()  
housing_cat_encoded[:10]
```

```
Out [38]: array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0], dtype=int32)
```

```
In [39]: housing_categories
```

```
Out [39]: Index(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY', 'ISLAND'], dtype='object')
```

- one-hot encoding

scikit-learn OneHotEncoder

- ◆ 숫자로 된 범주형 값을 one-hot vector로 변환
- ◆ `fit_transform()` ← 2차원 배열 (reshape으로 2차원으로 변형)
- ◆ 출력 형태 : SciPy의 sparse matrix
 - '0'이 아닌 원소의 위치만 저장

```
In [40]: from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot
```

```
Out [40]: <16512x5 sparse matrix of type '<class 'numpy.float64'>'
          with 16512 stored elements in Compressed Sparse Row format>
```

```
In [41]: housing_cat_1hot.toarray()
```

```
Out [41]: 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.]])
```

- ◆ numpy 배열로 변형 : `toarray()` 함수

2.5.3 나만의 변환기

◆ scikit-learn은 유용한 변환기를 많이 제공

```
In [48]: from sklearn.base import BaseEstimator, TransformerMixin

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

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

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

```
In [49]: housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household", "population_per_household"])
housing_extra_attribs.head()
```

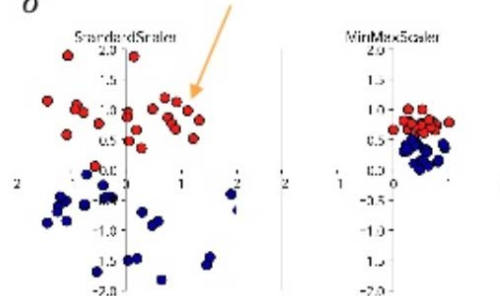
Out [49]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	rooms_per
0	-121.89	37.29	38	1568	351	710	339	2.7042	<1H OCEAN	4.62537
1	-121.93	37.05	14	679	108	306	113	6.4214	<1H OCEAN	6.00885
2	-117.2	32.77	31	1952	471	936	462	2.8621	NEAR OCEAN	4.22511
3	-119.61	36.31	25	1847	371	1460	353	1.8839	INLAND	5.23229
4	-118.59	34.23	17	6592	1525	4459	1463	3.0347	<1H OCEAN	4.50581

2.5.4 특성 스케일링(feature scaling)

- ◆ 머신러닝 알고리즘은 입력 숫자 특성들의 scale이 많이 다르면 잘 작동하지 않음
 - 예) 전체 방 개수 범위 6~39,320 / 중간소득범위 0~15
- ◆ 특성의 범위를 같도록...
 - min-max scaling : 정규화 (normalization)
 - 0~1 범위에 들도록 값 이동 및 스케일 조정
 - 데이터-최소값 / 최대값-최소값
 - scikit-learn MinMaxScaler 변환기
 - 표준화 (standardization)
 - 평균을 뺀 후, 표준편차로 나누어 결과 분포의 분산이 1이 되도록 함
 - scikit-learn StandardScaler 변환기

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



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

모든 특성이
0과 1사이에 위치
이상치에 민감

2.5.5 변환 파이프라인

◆ Pipeline 클래스

- scikit-learn에서 연속된 변환을 순서대로 처리

◆ 숫자 특성 처리 파이프라인

```
In [50]: from sklearn.pipeline import Pipeline
         from sklearn.preprocessing import StandardScaler

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

         housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
In [51]: housing_num_tr
```

```
Out [51]: array([[ -1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
        [ -1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
        [  1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
        ...,
        [  1.58648943, -0.72478134, -1.56295222, ...,  0.3469342 ,
        -0.03055414, -0.52177644],
        [  0.78221312, -0.85106801,  0.18664186, ...,  0.02499488,
         0.06150916, -0.30340741],
        [-1.43579109,  0.99645926,  1.85670895, ..., -0.22852947,
        -0.09586294,  0.10180567]])
```

◆ pandas의 dataframe 처리하는 변환기 작성

- DataFrameSelector 클래스 : 나머지는 버리고, 필요한 특성을 선택하여 데이터프레임을 numpy 배열로 변경 (수치형만 다루는 파이프라인)

```
In [52]: from sklearn.base import BaseEstimator, TransformerMixin

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
```

```
In [53]: num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

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

cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attribs)),
    ('cat_encoder', CategoricalEncoder(encoding="onehot-dense")),
])
```

- 수치형 파이프라인 / 범주형 파이프라인

◆ 두 파이프라인 연결

```
In [54]: from sklearn.pipeline import FeatureUnion
```

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

```
In [55]: housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

```
Out [55]: 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.          ]])
```

```
In [60]: housing_prepared.shape
```

```
Out [60]: (16512, 16)
```

2.6 모델 선택과 훈련

◆ 2.6.1 훈련 세트에서 훈련하고 평가하기

◆ 모델선택1) 선형 모델

```
In [56]: from sklearn.linear_model import LinearRegression
```

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

```
Out [56]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
In [57]: some_data = housing.iloc[:5]  
some_labels = housing_labels.iloc[:5]  
some_data_prepared = full_pipeline.transform(some_data)
```

```
In [58]: print("예측: ", lin_reg.predict(some_data_prepared))
```

```
예측: [ 210644.60459286  317768.80697211  210956.43331178  59218.98886849  
189747.55849879]
```

```
In [59]: print("레이블: ", list(some_labels))
```

```
레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```


RMSE 측정

◆ scikit-learn의 mean_square_error() 함수 사용

```
In [61]: 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
```

```
Out [61]: 68628.198198489234
```

```
In [62]: housing_labels.mean(), housing_labels.std()
```

```
Out [62]: (206990.9207243217, 115703.01483031521)
```

```
In [63]: lin_reg.score(housing_prepared, housing_labels)
```

```
Out [63]: 0.64816248428044276
```

- RMSE 예측 오차 : \$68,628 → 낮을수록 좋음
- score 높을수록 좋음 → 과소적합 (underfitting)
- 모델이 train data에 과소적합된 사례
 - 모델이 너무 단순해서 데이터의 내재된 구조를 학습하지 못할 때
 - 해결방법 1) 파라미터가 더 많은 **강력한 모델** 선택 → **먼저 시도**
 - 해결방법 2) 학습 알고리즘에 더 좋은 특성 제공
 - 해결방법 3) 모델 규제 감소 (→ 이 예제에서는 규제 사용 안함)

모델 선택2) 결정 트리 (6장)

◆ DesionTreeRegressor

```
In [64]: from sklearn.tree import DecisionTreeRegressor
```

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

```
Out [64]: DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,  
                                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,  
                                presort=False, random_state=None, splitter='best')
```

```
In [65]: housing_predictions = tree_reg.predict(housing_prepared)  
tree_mse = mean_squared_error(housing_labels, housing_predictions)  
tree_rmse = np.sqrt(tree_mse)  
tree_rmse
```

```
Out [65]: 0.0
```

```
In [66]: tree_reg.score(housing_prepared, housing_predictions)
```

```
Out [66]: 1.0
```

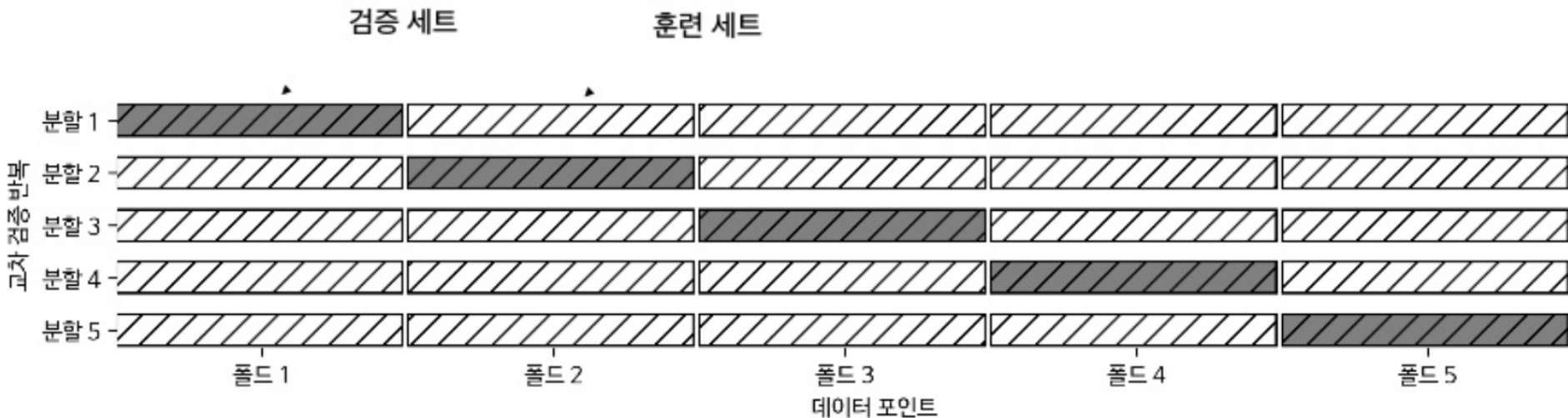
- 오차 = 0.0
- score = 1.0 → 과대적합 (overfitting)
 - model이 train data에 너무 잘 맞지만, 일반성이 떨어짐

2.6.2 교차 검증을 사용한 평가

◆ train set 중 일부를 사용하여 검증에 사용

◆ scikit-learn cross-validation

- fold : subset
- 훈련 세트를 10개의 폴드(서브셋)으로 분할
- 결정트리모델을 10번 훈련하고 평가
(매번 다른 폴드를 선택해서 평가, 나머지 9개 폴드는 훈련에 사용)
- 결과) 10개의 평가 점수가 담긴 배열



◆ 결정 트리 검증 결과 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)

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

display_scores(tree_rmse_scores)
```

```
Scores: [ 68109.06259568  66943.63356662  71594.75323241  68942.80518278
  70918.0003004   75081.18897418  70626.41498532  71763.18783985
  78783.35316833  70158.1998105 ]
Mean: 71292.0599656
Standard deviation: 3274.42334379
```

- 평균 : 71,292
- 표준편차 : 3,274

◆ 회귀 모델 검증 결과 In [70]:

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                              scoring="neg_mean_squared_error", cv=10)

lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [ 66782.73843989  66960.118071   70347.95244419  74739.57052552
  68031.13388938  71193.84183426  64969.63056405  68281.61137997
  71552.91566558  67665.10082067]
Mean: 69052.4613635
Standard deviation: 2731.6740018
```

- 평균 : 69,052
- 표준편차 : 2,731

RandomForestRegressor (7장)

◆ 무작위로 특성을 선택해서 많은 DecisionTree를 생성하고, 그 예측을 평균 낸

- 앙상블 학습 : 여러 다른 모델을 모아서 하나의 모델을 만드는 것
- 머신러닝 알고리즘 성능 극대화 방법 중 하나

```
In [71]: from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared, housing_labels)
```

```
Out [71]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                                max_features='auto', 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=10, n_jobs=1,
                                oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
In [72]: housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
Out [72]: 22534.651251015624
```

```
In [73]: from sklearn.model_selection import cross_val_score

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

```
Scores: [ 53018.62573497  49748.6765257  51080.6823411  55480.57393783
          51220.62786958  55532.7434547  50918.77758198  50700.22329812
          55377.35557488  52712.46753021]
Mean: 52579.0753849
Standard deviation: 2087.4053737
```

2.7 모델 세부 튜닝

- ◆ 가능성 있는 2~5개 정도의 모델을 선정하여 저장해 두면 편리함
- ◆ 최적의 하이퍼파라미터를 찾아야 함
- ◆ 가장 단순한 방법 → 만족할 만한 하이퍼파라미터 조합을 찾을 때까지 수동으로 조정

2.7.1 그리드 탐색

◆ scikit-learn의 GridSearchCV 사용

- 탐색하고자 하는 하이퍼파라미터와 시도값 지정
- 가능한 모든 하이퍼파라미터 조합에 대해 교차 검증을 사용해 평가

In [74]: `from sklearn.model_selection import GridSearchCV`

```
param_grid = [  
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},  
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},  
]  
forest_reg = RandomForestRegressor()  
  
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,  
                           scoring='neg_mean_squared_error',  
                           return_train_score=True)  
  
grid_search.fit(housing_prepared, housing_labels)
```

Out [74]: `GridSearchCV(cv=5, error_score='raise',
 estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
 max_features='auto', 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=10, n_jobs=1,
 oob_score=False, random_state=None, verbose=0, warm_start=False),
 fit_params=None, iid=True, n_jobs=1,
 param_grid=[{'max_features': [2, 4, 6, 8], 'n_estimators': [3, 10, 30]}, {'bootstrap': [False], 'max_features': [2, 3, 4], 'n_estimators': [3, 10]}],
 pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
 scoring='neg_mean_squared_error', verbose=0)`

```
In [75]: grid_search.best_params_
```

```
Out [75]: {'max_features': 6, 'n_estimators': 30}
```

```
In [76]: grid_search.best_estimator_
```

```
Out [76]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                                max_features=6, 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=None,
                                verbose=0, warm_start=False)
```

```
In [77]: cvres = grid_search.cv_results_
         for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
             print(np.sqrt(-mean_score), params)
```

```
64223.2850771 {'max_features': 2, 'n_estimators': 3}
55407.8687859 {'max_features': 2, 'n_estimators': 10}
52932.3550944 {'max_features': 2, 'n_estimators': 30}
60222.8024295 {'max_features': 4, 'n_estimators': 3}
52951.7955765 {'max_features': 4, 'n_estimators': 10}
50280.7716783 {'max_features': 4, 'n_estimators': 30}
59048.4337212 {'max_features': 6, 'n_estimators': 3}
52588.478215 {'max_features': 6, 'n_estimators': 10}
50031.7461754 {'max_features': 6, 'n_estimators': 30}
58077.5052279 {'max_features': 8, 'n_estimators': 3}
51545.0350056 {'max_features': 8, 'n_estimators': 10}
50083.4490232 {'max_features': 8, 'n_estimators': 30}
63065.7241353 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54091.1788418 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59798.9622518 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52314.2649346 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
59226.5630142 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51963.370483 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```


◆ 2.7.1 그리드 탐색

- 비교적 적은 수의 조합 탐구에 적합
- 가능한 모든 조합을 시도
- 하이퍼파라미터마다 몇 개의 값만 탐색

◆ 2.7.2 랜덤 탐색

- 탐색 공간이 커지면 랜덤 탐색 방식이 유용
- 각 반복마다 하이퍼파라미터에 임의의 수를 대입하여, 지정한 횟수만큼 평가
- 하이퍼파라미터마다 각기 다른 값 탐색
- 단순히 반복 횟수를 조절하는 것만으로 하이퍼파라미터 탐색에 투입할 컴퓨팅 자원 제어 가능

◆ 2.7.3 앙상블 방법

- 단일 모델을 연결하여 모델의 그룹으로 만듦
- 예) 결정 트리의 앙상블 → 랜덤 포레스트
- 7장

2.7.2 랜덤 탐색

```
In [80]: 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)
```

```
Out [80]: RandomizedSearchCV(cv=5, error_score='raise',
    estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
    max_features='auto', 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=10, n_jobs=1,
    oob_score=False, random_state=42, verbose=0, warm_start=False),
    fit_params=None, iid=True, n_iter=10, n_jobs=-1,
    param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x0BEBE110>, 'n_estimators': <scipy.st
ats._distn_infrastructure.rv_frozen object at 0x0BEBE6F0>},
    pre_dispatch='2*n_jobs', random_state=42, refit=True,
    return_train_score=True, scoring='neg_mean_squared_error',
    verbose=0)
```

```
In [81]: cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49147.1524172 {'max_features': 7, 'n_estimators': 180}
51396.8768969 {'max_features': 5, 'n_estimators': 15}
50797.0573732 {'max_features': 3, 'n_estimators': 72}
50840.744514 {'max_features': 5, 'n_estimators': 21}
49276.1753033 {'max_features': 7, 'n_estimators': 122}
50775.4633168 {'max_features': 3, 'n_estimators': 75}
50681.383925 {'max_features': 3, 'n_estimators': 88}
49612.1525305 {'max_features': 5, 'n_estimators': 100}
50473.0175142 {'max_features': 3, 'n_estimators': 150}
64458.2538503 {'max_features': 5, 'n_estimators': 2}
```

테스트 세트 평가

◆ 마지막에 한번 수행

- test set → predictor, label 데이터
- full pipeline 사용하여 데이터 변환
- test set에서 최종 모델 평가

```
In [82]: 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)
```

```
In [83]: final_rmse
```

```
Out [83]: 48403.473415816981
```

- 최종 RMSE 오차 확인

2.8 론칭, 모니터링, 시스템 유지보수

◆ 전처리와 예측을 포함한 파이프라인

- preparation + linear model

```
In [84]: 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)

Out [84]: array([ 210644.60459286,  317768.80697211,  210956.43331178,
                   59218.98886849,  189747.55849879])
```

◆ 모델 저장

- 하이퍼파라미터, 모델 파라미터 모두 저장

```
In [85]: my_model = full_pipeline_with_predictor

In [86]: from sklearn.externals import joblib
          joblib.dump(my_model, "my_model.pkl") # DIFF
          #...
          my_model_loaded = joblib.load("my_model.pkl") # DIFF
```

◆ 론칭

- 실시간 성능 체크를 위한 모니터링 코드 개발
- 분석가의 성능 평가 (예: 해당 분야의 전문가)
- 입력 데이터 모니터링 코드 개발
- 정기적 훈련을 위한 자동화

◆ Further Chapters...

Part I	Part II
3장 분류	(9장 텐서플로 시작하기)
4장 모델 훈련	10장 인공 신경망 소개
5장 서포트 벡터 머신	11장 심층 신경망 훈련
6장 결정 트리	12장 다중 머신과 장치를 위한 분산 텐서플로
7장 앙상블 학습과 랜덤 포레스트	13장 합성곱 신경망
8장 차원 축소	14장 순환 신경망
	...

Any Questions... Just Ask!

