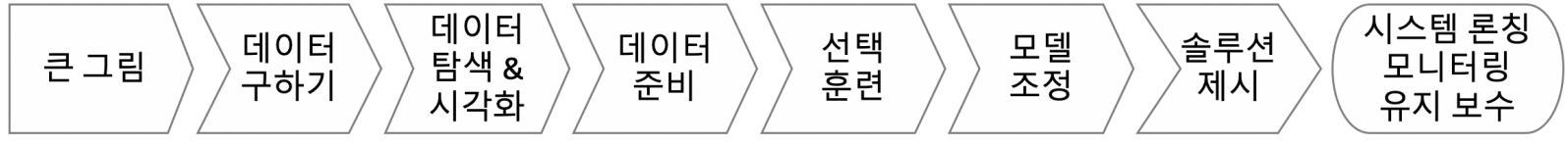
2장 머신러닝 프로젝트 처음부터 끝까지

- 주택 가격을 예측하는 회귀 작업을 살펴보면서 선형 회귀, 결정 트리, 랜덤 포레스트 등 여러 알고리즘 학습



2.1 실제 데이터로 작업하기

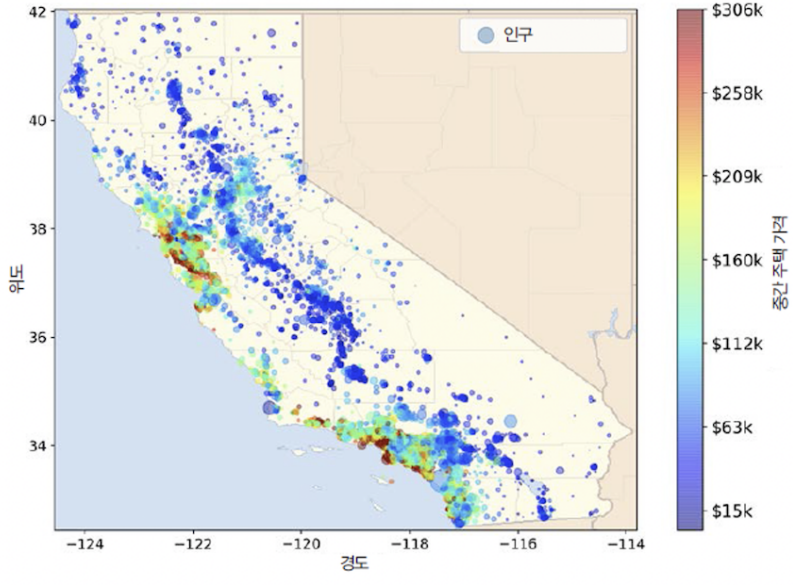
- 유명한 공개 데이터 저장소
    - UC 얼바인Irvine 머신러닝 저장소(http://archive.ics.uci.edu/ml (http://archive.ics.uci.edu/ml))
    - 캐글Kaggle 데이터셋(http://www.kaggle.com/datasets (http://www.kaggle.com/datasets))
    - 아마존 AWS 데이터셋(https://registry.opendata.aws (https://registry.opendata.aws))
- 메타 포털(공개 데이터 저장소가 나열)
    - 데이터 포털Data Portals(http://dataportals.org (http://dataportals.org))
    - 오픈 데이터 모니터Open Data Monitor(http://opendatamonitor.eu (http://opendatamonitor.eu))
    - 퀀들Quandl(http://quandl.com (http://quandl.com))
- 인기 있는 공개 데이터 저장소가 나열되어 있는 다른 페이지
    - 위키백과 머신러닝 데이터셋 목록(https://goo.gl/SJHN2k (https://goo.gl/SJHN2k))
    - Quora.com(https://homl.info/10 (https://homl.info/10))
    - 데이터셋 서브레딧subreddit(http://www.reddit.com/r/datasets (http://www.reddit.com/r/datasets))

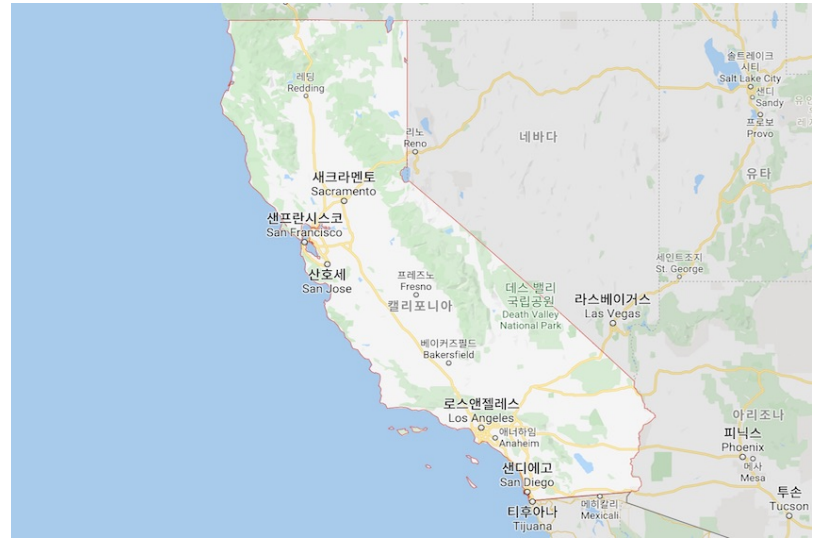
2.2 큰 그림 보기

- 주어진 데이터: 미국 캘리포니아 인구조사 데이터
    - 특성: 구역(block)별 인구, 중간 소득, 경도, 위도 등
    - 레이블: 중간 주택 가격
- 목표: 캘리포니아 주택가격 모델 구현

문제 정의

- 지도 학습(supervised learning): 조사된 주택가격 레이블 활용 훈련
- 회귀(regression): 중간 주택 가격 예측
    - 다중 회귀(multiple regression): 여러 특성을 활용한 예측
    - 단변량 회귀(univariate regression): 구역마다 하나의 가격만 예측
- 배치 학습(batch learning): 빠르게 변하는 데이터에 적응할 필요가 없음


인구
위도
경도
중간 주택 가격
42
40
38
36
34
−124
−122
−120
−118
−116
−114
$306k
$258k
$209k
$160k
$112k
$63k
$15k


레딩
Redding
리노
Reno
네바다
솔트레이크 시티
Salt Lake City
샌디
Sandy
프로보
Provo
유타
새크라멘토
Sacramento
샌프란시스코
San Francisco
산호세
San Jose
프레즈노
Fresno
캘리포니아
데스 밸리 국립공원
Death Valley National Park
라스베이거스
Las Vegas
세인트조지
St. George
베이커즈필드
Bakersfield
로스앤젤레스
Los Angeles
애너하임
Anaheim
샌디에고
San Diego
티후아나
Tijuana
메히칼리
Mexicali
아리조나
피닉스
Phoenix
메사
Mesa
투손
Tucson

2.3 데이터 구하기

작업환경 만들기

- 추천: 구글 코랩, Docker 등 활용
- 직접 개발환경 설정하기는 교재 2.3.1절 참조

데이터 다운로드

- StaLib 저장소에 있는 데이터 활용
- 저자가 교육 목적으로 일부 특성 제외 및 범주형 특성 추가

데이터 구조 훑어보기

pandas의 데이터프레임 활용

- `head()`,`info()`,`describe()`,`hist()` 등을 사용하여 데이터 구조 훑어보기

head() 메서드 활용 결과

```
In [5]: housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
0	-122.23	37.88	41.0	880.0	129.0	322.(
1	-122.22	37.86	21.0	7099.0	1106.0	2401.(
2	-122.24	37.85	52.0	1467.0	190.0	496.(
3	-122.25	37.85	52.0	1274.0	235.0	558.(
4	-122.25	37.85	52.0	1627.0	280.0	565.(

`info()` 메서드 활용 결과

1. 총 **샘플링 개수**: 20,640
    - 캘리포니아를 20,640구역으로 나위어 조사한 인구조사
    - 구역 크기:8 600 ~ 3,000명
2. 구역별로 경도, 위도, 중간 주택 연도, 해안 근접도 등 총 10개의 조사 항목
3. '해안 근접도'는 범주형 데이터이고 나머지는 수치형 데이터.
4. '방의 총 개수'의 경우 누락된 데이터인 207개의 null 값 존재

범주형 데이터 탐색

- '해안 근접도'는 5개의 범주로 구분
    - <1H OCEAN: 해안에서 1시간 이내
    - INLAND: 내륙
    - NEAR OCEAN: 해안 근처
    - NEAR BAY: Bay Area라 불리는 샌프란시스코 도시 중심 지역
    - ISLAND: 섬

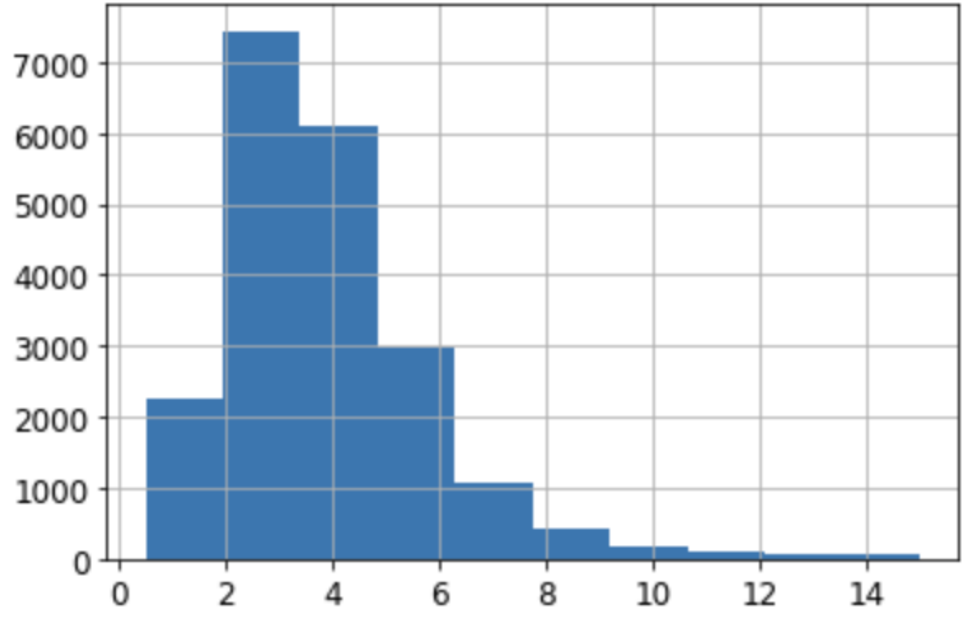
테스트 세트 만들기

- 모델 학습 시작 이전
    - 준비된 데이터셋을 훈련 세트과 테스트 세트로 구분
- 테스트 세트에 포함된 데이터는 미리 분석하지 말 것.
    - 미리 분석 시 **데이터 스누핑 편향**을 범할 가능성이 높아짐
    - 미리 보면서 알아낸 직관이 학습 모델 설정에 영향을 미칠 수 있음
    - 테스트 세트 크기: 전체 데이터 셋의 20%
- 훈련 세트와 데이터 세트를 구분하는 방식에 따라 결과가 조금씩 달라짐
    - 무작위 샘플링 vs. 계층적 샘플링
- 여기서는 계층적 샘플링 활용

계층적 샘플링

- 계층: 동질 그룹
- 테스트 세트: 전체 계층을 대표하도록 각 계층별로 적절한 샘플 추출
    - 계층 기준 예제: 소득
- 소득의 범주: 계층별로 충분한 크기의 샘플이 포함되도록 지정
    - 학습 과정에서 편향이 발생하지 않도록 하기 위해
    - 특정 소득 구간에 포함된 샘플이 과하게 적거나 많으면 해당 계층의 중요도가 과대 혹은 과소 평가될 것

- 전체 데이터셋의 중간 소득 히스토그램 활용



- 대부분 구역의 중간 소득이 **1.5~6.0**(15,000~60,000$) 사이
- 소득 구간을 아래 숫자를 기준으로 5개로 구분

```
[0, 1.5, 3.0, 4.6, 6.0, np,inf]
```

계층 샘플링과 무작위 샘플링 비교

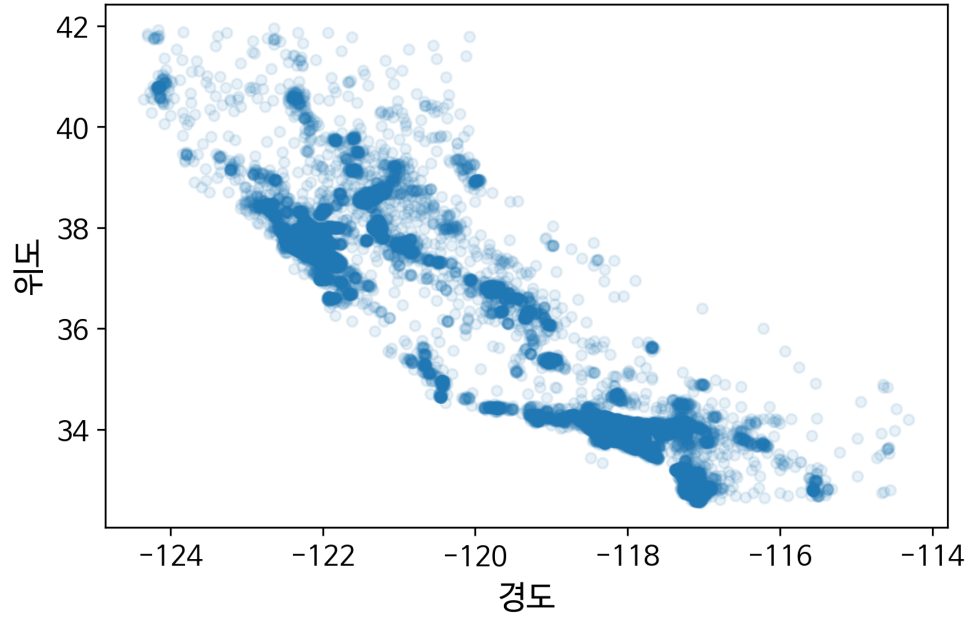
	전체	계층 샘플링	무작위 샘플링	무작위 샘플링 오류율	계층 샘플링 오류율
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

2.4 데이터 이해를 위한 탐색과 시각화

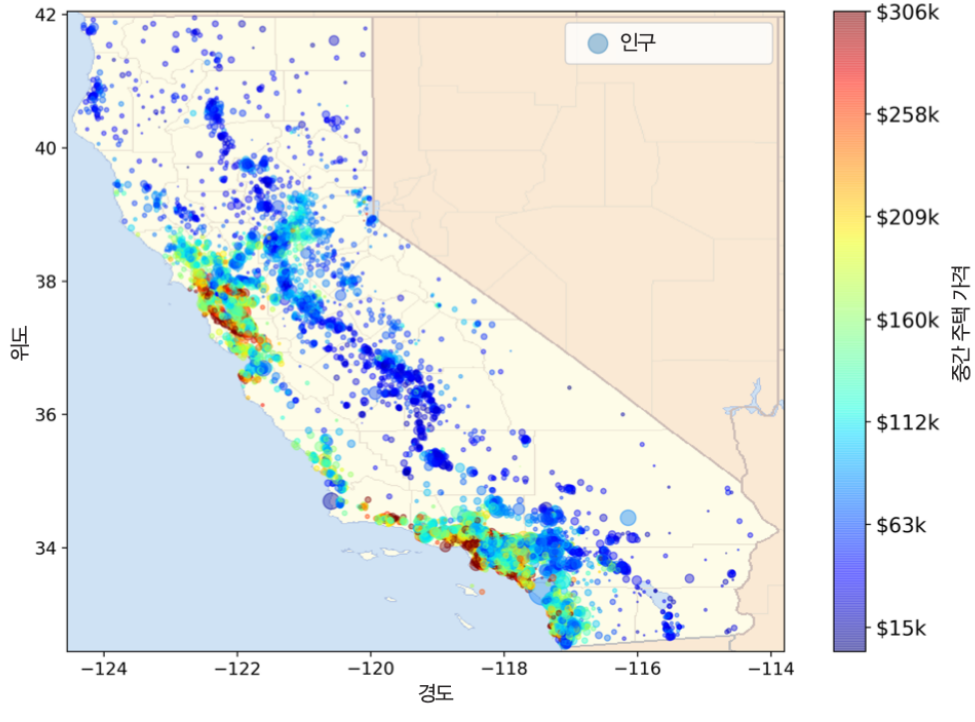
- 테스트 세트를 제외한 훈련 세트에 대해서만 시각화를 이용하여 탐색

지리적 데이터 시각화

- 구역이 집결된 지역과 그렇지 않은 지역 구분 가능
- 샌프란시스코의 베이 에어리어, LA, 샌디에고 등 밀집된 지역 확인 가능



- 주택 가격이 해안 근접도, 인구 밀도와 관련이 큼
- 해안 근접도: 위치에 따라 다르게 작용
    - 대도시 근처: 해안 근처 주택 가격이 상대적 높음
    - 북부 캘리포니아 지역: 높지 않음



상관관계 조사

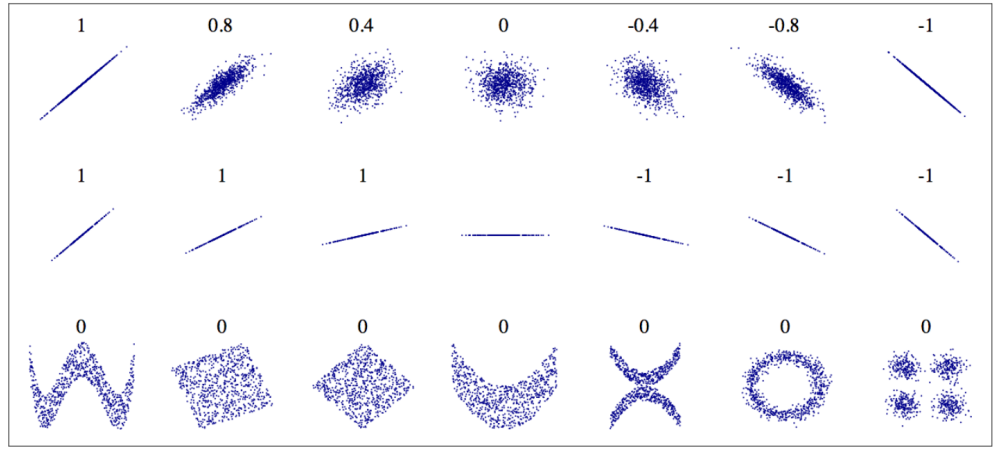
- 중간 주택 가격 특성과 다른 특성 사이의 상관관계: 상관계수 활용

```
In [39]: corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
Out[39]: 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
```

상관계수의 특징

- 상관계수: $[-1, 1]$ 구간의 값
- 1에 가까울 수록: 강한 양의 선형 상관관계
- -1에 가까울 수록: 강한 음의 선형 상관관계
- 0에 가까울 수록: 매우 약한 선형 상관관계



주의사항

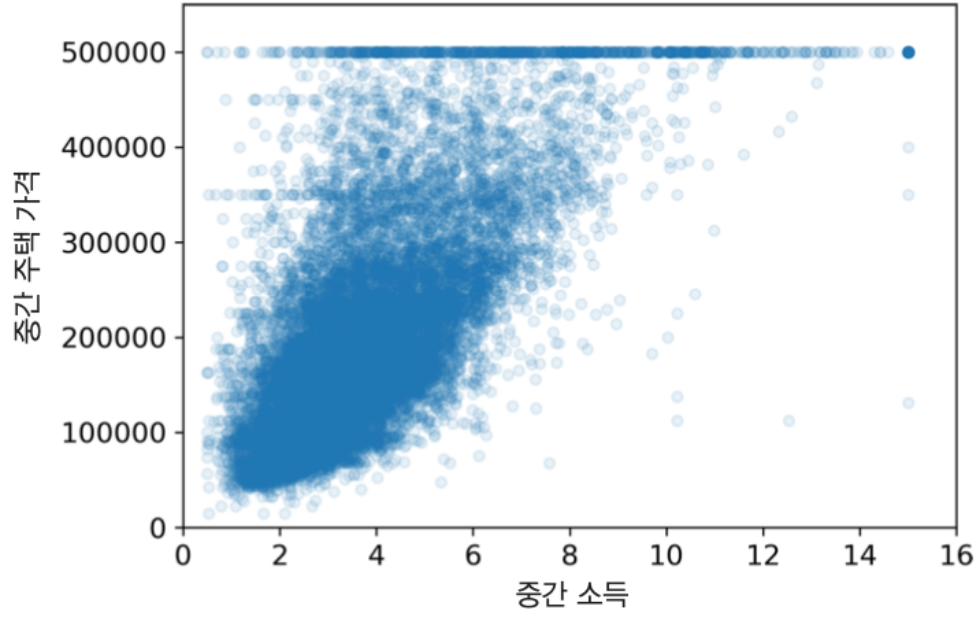
- 상관계수가 0에 가까울 때: 선형 관계가 거의 없다는 의미이지, 아무런 관계가 없다는 의미는 아님
- 상관계수는 기울기와 아무 연관 없음

상관계수를 통해 확인할 수 있는 정보

- 중간 주택 가격과 중간 소득의 상관계수가 0.68로 가장 높음
    - 중간 소득이 올라가면 중간 주택 가격도 상승하는 경향이 있음

중간 주택 가격과 중간 소득의 관계

- 산점도 활용
    - 점들이 너무 넓게 퍼져 있음. 완벽한 선형관계와 거리 멂.
    - 50만 달러 수평선: 가격 제한
    - 35만, 28만, 그 아래 정도에서도 수평선 존재
    - 이상한 형태를 학습하지 않도록 해당 구역을 제거하는 것이 좋음.



특성 조합 실험

- 구역별 방의 총 개수와 침실의 총 개수 대신 아래 특성이 보다 유용함
    - '가구당 방 수'(rooms for household)
    - '방 하나당 침실 수'(bedrooms for room)
    - '가구당 인원'(population per household)

- 특성별 상관계수 다시 확인
    - 중간 주택 가격과 방 하나당 침실 개수: -0.26
    - 방 하나당 침실 개수가 적을 수록 주택 가격이 상승하는 경향

2.5 머신러닝 알고리즘을 위한 데이터 준비

데이터 전처리와 변환 파이프라인

- 데이터 전처리(data preprocessing): 모델 학습을 효율적으로 진행하기 위해 주어진 데이터를 변환시키는 것
- 수치형 데이터와 범주형 데이터에 대해 다른 변환과정을 사용

- 수치형 데이터 전처리 과정
    - 데이터 정제
    - 조합 특성 추가
    - 특성 스케일링

- 범주형 데이터 전처리 과정
    - 원-핫-인코딩(one-hot-encoding)

- 수치형 데이터 전처리 과정에 사용된 세 가지 변환과정은 파이프라인을 이용하여 자동화

- 이후 원-핫-인코딩 변화과정으로 변환된 범주형 데이터와 결합하면서 전처리 과정이 완료됨

사이킷런 API 활용

- 변환과정 중에서 '조합 특성 추가' 과정을 제외한 나머지 과정은 사이킷런에서 제공하는 관련 API를 활용
- '조합 특성 추가' 과정도 다른 사이킷런 API와 호환이 되는 방식으로 사용자가 직접 구현하는 방법을 설명
- 사이킷런에서 제공하는 API는 일관되고 단순한 인터페이스를 제공
- 이 성질을 이용하여 '조합 특성 추가' 과정을 지원하는 API를 구현하면 사이킷런의 다른 API와 자동으로 호환

일관성

- 모든 사이킷런의 API는 일관되고 단순한 인터페이스를 제공
- 대표적인 API는 세 가지

- 추정기(estimator)
    - 데이터셋을 기반으로 특정 모델 파라미터들을 추정하는 클래스의 객체이며, fit() 메서드가 이 기능을 수행
    - fit()메서드의 리턴값: 생성된 모델 파라미터를 인스턴스 속성으로 갖는 self
        - 특정 속성이 업데이트된 객체 자신이 리턴값

- 변환기(transformer):
    - 데이터셋을 변환하는 추정기이며, transform()이 이 기능을 수행
        - 변환기는 fit()과 transform() 모두 포함되어 있어야 함
    - fit() 메서드에 의해 학습된 파라미터를 이용하여 데이터셋을 변환한다.
    - 모든 변화기는 fit() 메서드와 transform() 메서드를 연속해서 호출하는 fit_transform() 메서드를 함께 제공

- 예측기(predictor): 데이터셋의 특정 특성에 대한 예측을 하는 추정기
    - 주어진 데이터셋과 관련된 값을 예측하는 기능을 제공하는 추정기이며, predict() 메서드가 이 기능을 수행
        - fit()과 predict() 메서드가 포함되어 있어야 함
    - predict() 메서드가 추정한 값의 성능을 측정하는 score() 메서드도 포함
    - 일부 예측기는 추정치의 신뢰도를 평가하는 기능도 제공

수치형 데이터 전치러 과정 1: 데이터 정제

- 누락된 특성값 존재 경우
    - 해당 값 또는 특성을 먼저 처리하고 모델 학습 진행
- `total_bedrooms` 특성에 207개 구역에 대한 값이 null로 채워져 있음, 즉, 일부 구역에 대한 정보가 누락됨.

- null 값 처리 방법
    - 해당 구역 제거
    - 전체 특성 삭제
    - 특정 값으로 채우기
        - 0, 평균값, 중앙값 등
        - 책에서는 중앙값으로 채움.

텍스트와 범주형 특성 다루기: 원-핫 인코딩

- 범주형 입력 특성인 해안 근접도(ocean_proximity)를 수치형 데이터로 변환

```
In [65]: ordinal_encoder.categories_
```

```
Out[65]: [array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OC
         EAN'],
               dtype=object)]
```

단순 수치화의 문제점

- 해안 근접도는 단순히 구분을 위해 사용
- 해안에 근접하고 있다 해서 주택 가격이 기본적으로 더 비싼 것은 아니라는 의미
- 반면에 수치화된 값들은 크기를 비교할 수 있는 숫자
- 따라서 모델 학습 과정에서 숫자들의 크기 때문에 잘못된 학습이 이루어질 수 있다.

원-핫 인코딩(one-hot encoding)

- 수치화된 범주들 사이의 크기 비교를 피하기 위해 더미(dummy) 특성을 추가하여 활용
- 생성되는 더미 특성은 사용된 범주를 사용
- 예를 들어, 해안 근접도 특성 대신에 다섯 개의 범주 전부를 새로운 특성으로 추가 -> 각각의 특성 값을 아래와 같이 정함
    - 해당 카테고리의 특성값: 1
    - 나머지 카테고리의 특성값: 0

In [68]:
```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[68]:
```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

수치형 데이터 전처리 과정 2: 조합 특성 추가

- 특성 추가를 위해 변환기 클래스를 **직접 정의**
- 앞서 살펴본 다음 세 가지 특성을 자동으로 추가하는 변환기 클래스를 정의
    - 가구당 방 개수(rooms for household)
    - 방 하나당 침실 개수(bedrooms for room)
    - 가구당 인원(population per household)
- 변환기 클래스를 선언하기 위해서는 fit() 메서드와 transform() 메서드만 정의하면 됨
- 주의: fit() 메서드의 리턴값은 self

CombinedAttributesAdder 변환기 클래스 선언

- **init**() 메서드: 방 하나당 침실 개수 속성을 추가할지 여부를 확인
- fit() 메서드: 계산해야 하는 파라미터가 없음
    - 아무 일도 할 필요 없이 바로 self를 리턴
- transform() 메서드: 넘파이 어레이를 입력받아 속성을 추가한 어레이를 반환

```
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True):
        ...

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        ...
```

수치형 데이터 전처리 과정 3: 특성 스케일링

- 머신러닝 알고리즘은 입력 데이터셋의 특성값들의 스케일(범위)이 다르면 제대로 작동하지 않음
- 특성에 따라 다루는 숫자의 크기가 다를 때 통일된 스케일링이 필요
- **주의**: 타깃(레이블)에 대한 스케일링은 하지 않음

min-max 스케일링

- **정규화**(normalization)라고도 불림
- 특성값 x를 $\frac{x-min}{max-min}$로 변환
- 변환 결과: **0에서 1** 사이
- 이상치에 매우 민감
    - 이상치가 매우 **크면 분모가 매우 커져서** 변환된 값이 **0 근처**에 몰림

표준화(standardization)

- 특성값 x 를 $\frac{x-\mu}{\sigma}$ 로 변환
    - μ: 특성값들의 **평균**값
    - σ: 특성값들의 **표준편차**
- 결과: 변환된 데이터들이 **표준정규분포**를 이룸
    - 이상치에 상대적으로 영향을 덜 받음.

주의사항

- 모든 변환기의 fit() 메서드는 훈련 데이터에 대해서만 적용
- transform() 메서드는 모든 데이터에 대해 적용
    - 훈련 세트를 이용하여 필요한 파라미터를 확인한 후 그 값들을 이용하여 전체 데이터셋트를 변환
    - 예를 들어, 따로 떼어놓은 테스트 데이터들은 훈련 데이터를 이용하여 확인된 값들을 이용하여 특성 스케일링을 진행

변환 파이프라인

- 모든 전처리 단계를 정확한 순서대로 연속적으로 진행되어야 함
- 사이킷런의 Pipeline 클래스가 이 기능을 지원
- 수치형 데이터 전처리 과정을 파이프라인으로 묶은 방법

pipeline 클래스 활용

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

- Pipeline의 인스턴스 생성에 필요한 인자는 추정기의 이름과 추정기의 쌍으로 이루어진 튜플들의 리스트
- 마지막 추정기를 제외하면 모두 변환기 즉, fit_transform() 메서드를 포함하고 있어야 함
- 생성된 파이프라인 객체의 fit() 메서드를 호출하면, 마지막 단계 이전까지는 해당 변환기의 fit_transform() 메소드가 연속해서 호출되며, 최종적으로 마지막 추정기의 fit() 메서드가 호출

ColumnTransformer 클래스

- 사이킷런의 `ColumnTransformer` 클래스를 이용하여 특성별로 지정된 전처리를 처리할 수 있도록 지정 가능
    - 수치형 특성: `num_pipeline` 변환기
    - 범주형 특성: `OneHotEncoder` 변환기

```
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

2.6 모델 선택과 훈련

- 전처리 후 두 요소를 결정해야함
    - 학습 모델
    - 회귀 모델 성능 측정 지표
- 목표: 구역별 중간 주택 가격 예측 모델
- 학습 모델: 회귀 모델
- 회귀 모델 성능 측정 지표: 평균 제곱근 오차(RMSE)

회귀 모델 성능 측정 지표

평균 제곱근 오차(root mean square error, RMSE):

- RMSE는 유클리디안 노름 또는 ℓ_2 노름으로도 불림

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=0}^{m-1}(h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

- $\mathbf{X}$: (평가대상) 데이터셋 전체 샘플들의 특성값들로 구성된 행렬, 레이블(타겟) 제외
  - m: 데이터셋 $\mathbf{X}$의 크기
  - $\mathbf{x}^{(i)}$: i 번째 샘플의 전체 특성값 벡터. 레이블(타겟) 제외
  - $y^{(i)}$: i 번째 샘플의 레이블
  - h: 예측 함수
  - $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$: i번째 샘플에 대한 예측 값

- **주의**
  - 인덱스 i가 책에서와는 달리 0부터 시작함.
  - 이유는 파이썬 넘파이 어레이, 데이터프레임에 사용되는 인덱스 개념과 통일시키기 위해서임.

평균 절대 오차(mean absolute error, MAE):

- MAE는 맨해튼 노름 또는 ℓ_1 노름으로도 불림
- 이상치가 많은 경우 활용

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=0}^{m-1} | h(\mathbf{x}^{(i)}) - y^{(i)} |$$

훈련 세트에서 훈련하고 평가하기

- 지금까지 한 일
    - 훈련 세트와 테스트 세트로 분류
    - 머신러닝 알고리즘에 주입할 데이터를 자동으로 정제하고 준비하기 위해 변환 파이프라인 작성
- 이제 할 일
    - 모델 선택 후 훈련시키기
    - 예제: 선형 회귀, 결정트리 회귀

선형 회귀 모델(4장)

- 사이킷런의 선형 회귀 모델은 **`LinearRegression`** 예측기 클래스가 제공

- 훈련:

```
from sklearn.linear_model import LinearRegression

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

- 예측

```
lin_reg.predict(housing_prepared))
```

결정트리 회귀 모델(6장)

- 결정 트리 모델은 데이터에서 복잡한 비선형 관계를 학습할 때 사용
- 사이킷런의 **`DecisionTreeRegressor`** 예측기가 결정 트리 회귀 모델을 생성

- 훈련:

```
from sklearn.tree import DecisionTreeRegressor

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

- 예측:

```
housing_predictions = tree_reg.predict(housing_prepared)
```

교차 검증을 사용한 평가

k-겹 교차 검증

- 훈련 세트를 **폴드**(fold)라 불리는 k-개의 부분 집합으로 무작위로 분할
- 총 k 번 지정된 모델을 훈련
    - 훈련할 때마다 매번 다른 하나의 폴드를 평가에 사용
    - 다른 (k-1) 개의 폴드를 이용해 훈련
- 최종적으로 k 번의 평가 결과가 담긴 배열 생성

예제: 결정 트리 모델 교차 검증

- k = 10으로 설정

```
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)
```

- **주의**
    - 효용함수: k-겹 교차 검증의 모델 학습 과정에서 성능을 측정할 때 높을 수록 좋은 효용 함수 활용
    - RMSE의 음숫값을 이용하여 훈련되는 모델 평가
        - `scoring="neg_mean_squared_error"`
    - 교차 검증 결과 평가를 위해 다시 음숫값(`-scores`)을 사용해야 함.

예제: 선형 회귀 모델 교차 검증

- k = 10으로 설정

```
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)
```

앙상블 학습(7장)

- 여러 다른 모델을 모아서 하나의 모델을 만드는 기법
- 즉, 교차 검증을 일반화 시킨 모델 학습법임.
- 머신러닝 알고리즘의 성능을 극대화는 방법 중 하나

- 앙상블 학습 예제: 랜덤 포레스트
    - 특성을 무작위로 선택해서 많은 결정 트리를 만들고 그 예측을 평균 내는 모델
    - 사이킷런의 `RandomForestRegressor` 사용법은 기본적으로 동일함.

- 훈련

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

- 예측

```
housing_predictions = forest_reg.predict(housing_prepared)
```

2.7 모델 세부 튜닝

- 살펴 본 모델 중에서 **랜덤 포레스트** 모델의 성능이 가장 좋았음
- 가능성이 높은 모델을 선정한 후에 **모델 세부 설정을 튜닝**해야함
- 튜닝을 위한 세 가지 방식
    - **그리드 탐색**
    - **랜덤 탐색**
    - **앙상블 방법**

그리드 탐색

- 지정한 하이퍼파라미터의 모든 조합을 교차검증하여 최선의 하이퍼파라미터 조합 찾기
- 사이킷런의 `GridSearchCV` 활용

- 예제: **랜덤 포레스트 모델에 대한 최적 조합**을 찾기
  - 총 $(3 \times 4 + 2 \times 3 = 18)$ 가지 조합 확인
  - 5-겹 교차검증(cv=5)이므로, 총 $(18 \times 5 = 90)$번 훈련함.

```
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(random_state=42)
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)
```

랜덤 탐색

- 그리드 탐색은 적은 수의 조합을 실험해볼 때 유용
- 조합의 수가 커지거나, 설정된 탐색 공간이 커지면 랜덤 탐색이 효율적
    - 설정값이 연속적인 값을 다루는 경우 랜덤 탐색이 유용
- 사이킷런의 `RandomizedSearchCV` 추정기가 랜덤 탐색을 지원

- 예제
  - n_iter=10: 랜덤 탐색이 총 10회 진행
    - n_estimators와 max_features 값을 지정된 구간에서 무작위 선택
  - cv=5: 5-겹 교차검증. 따라서 랜덤 포레스트 학습이 $(10 \times 5 = 50)$번 이루어짐

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

param_distribs = {
        '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_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error'
, random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

앙상블 방법

- 결정 트리 모델 하나보다 랜덤 포레스트처럼 모델의 그룹이 보다 좋은 성능을 낼 수 있음.
- 또한 최고 성능을 보이는 서로 다른 개별 모델을 조합하면 보다 좋은 성능을 얻을 수 있음
- 7장에서 자세히 다룸

최상의 모델과 오차 분석

- 그리드 탐색과 랜덤 탐색 등을 통해 얻어진 최상의 모델을 분석해서 문제에 대한 좋은 통창을 얻을 수 있음
- 예를 들어, 최상의 랜덤 포레스트 모델에서 사용된 특성들의 중요도를 확인하여 일부 특성을 제외할 수 있음.
    - 중간 소득(median income)과 INLAND(내륙, 해안 근접도)가 가장 중요한 특성으로 확인됨
    - 해안 근접도의 다른 네 가지 특성은 별로 중요하지 않음

테스트 셋으로 최상의 모델 평가하기

1. 최상의 모델 확인

```
final_model = grid_search.best_estimator_
```

2. 테스트 세트 전처리

    - 전처리 파이프라인의 `transform()` 메서드를 직접 활용
    - **주의**: `fit()` 메서드는 전혀 사용하지 않음

3. 최상의 모델을 이용하여 예측하기

4. 최상의 모델 평가 및 론칭