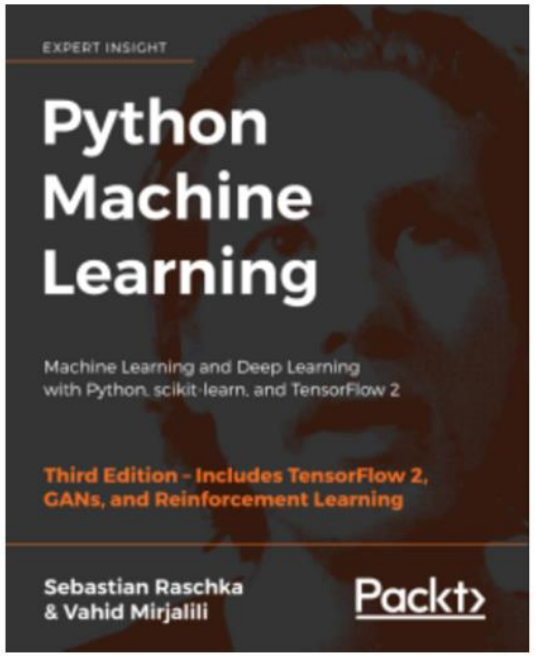
Data Preprocessing

Python Machine Learning Third Edition

Machine Learning and Deep Learning with Python, scikit-learn and Tensorflow 2

CHAPTER 4 Building Good Training Datasets – Data Preprocessing



Dealing with missing data
Identifying missing values in tabular data
Eliminating training examples or features with missing values
Imputing missing values
Categorical data encoding with pandas
Mapping ordinal features
Encoding class labels
Performing one-hot encoding on nominal features
Partitioning a dataset into separate training and test datasets
Bringing features onto the same scale

The quality of the data and the amount of useful information that it contains are key factors that determine how well a machine learning algorithm can learn

- Absolutely critical to examine and preprocess a dataset before we feed it to a learning algorithm
- Topic in this chapter
  - Removing and imputing missing values from the dataset
  - Getting categorical data into shape for machine learning algorithm

Dealing with missing data

- Crucial to take care of missing values before proceeding with further analyses
- Several practical techniques for dealing with missing values
  - ✓ By removing entries from our dataset
  - ✓ By imputing missing from other samples or features

Identifying missing values in tabular data

```
>>> import pandas as pd
>>> from io import StringIO
>>> import sys
>>> csv_data = '''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''
>>> df = pd.read_csv(StringIO(csv_data))
>>> df
      A     B     C    D
0   1.0   2.0   3.0  4.0
1   5.0   6.0   NaN  8.0
2  10.0  11.0  12.0  NaN
```

Read CSV-formatted data into a pandas DataFrame via *read_csv* function

Two missing cells were replaced by *NaN*
StringIO function allows us to read the string assigned to *csv_data* into a pandas *DataFrame*

```
>>> df.isnull().sum()
A    0
B    0
C    1
D    1
dtype: int64
```

isnull method to return a *DataFrame* with Boolean values that indicate whether a cell contains a numeric value(False) or if data is missing(True)

sum method : the number of missing values per column

➡ count the number of missing values per column

```
>>> df.values
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., nan,  8.],
       [10., 11., 12., nan]])
```

Access the NumPy array of a DataFrame via the value attribute

Eliminating training examples or features with missing values

- One of the easiest way
  - ✓ simply to remove the corresponding features (columns) or training examples (rows)

```
>>> df
     A     B     C    D
0  1.0   2.0   3.0  4.0
1  5.0   6.0   NaN  8.0
2  10.0  11.0  12.0 NaN
```

```
# remove rows that contain missing values

df.dropna(axis=0)
```

```
     A    B    C    D
0  1.0  2.0  3.0  4.0
```

```
# remove columns that contain missing values

df.dropna(axis=1)
```

```
      A     B
0   1.0   2.0
1   5.0   6.0
2  10.0  11.0
```

```
# only drop rows where all columns are NaN

df.dropna(how='all')
```

```
      A     B     C    D
0   1.0   2.0   3.0  4.0
1   5.0   6.0   NaN  8.0
2  10.0  11.0  12.0 NaN
```

```
# drop rows that have less than 3 real values

df.dropna(thresh=4)
```

```
     A    B    C    D
0  1.0  2.0  3.0  4.0
```

```
# only drop rows where NaN appear in specific columns (here: 'C')

df.dropna(subset=['C'])
```

```
      A     B     C    D
0   1.0   2.0   3.0  4.0
2  10.0  11.0  12.0 NaN
```

```
>>> #전체특성 삭제
>>> df.drop("C",axis=1)
      A     B    D
0   1.0   2.0  4.0
1   5.0   6.0  8.0
2  10.0  11.0 NaN
```

Imputing missing values

- One of the most common interpolation techniques : mean imputation

```
# impute missing values via the column mean(3rd edition)
from sklearn.impute import SimpleImputer
import numpy as np
imr=SimpleImputer(missing_values=np.nan, strategy='mean')
imr=imr.fit(df.values)
imputed_data=imr.transform(df.values)
imputed_data
```

```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

- An even more convenient way to impute missing values is by using pandas' *fillna* method

```
>>> df.fillna(df.mean())
      A     B     C    D
0   1.0   2.0   3.0  4.0
1   5.0   6.0   7.5  8.0
2  10.0  11.0  12.0  6.0
```

- *SimpleImputer* class belongs to the so-called transformer classes in scikit-learn
  - ✓ *fit* method : learn the parameters from the training data
  - ✓ *transform* method: use those parameters to transform the data

Handling categorical data

- Ordinal features
  - ✓ Categorical values that can be sorted or ordered (ex. T-shirt size)
- Nominal features
  - ✓ Don’t imply any order (ex. T-shirt color)

Categorical data encoding with pandas

```
import pandas as pd

df = pd.DataFrame([['green', 'M', 10.1, 'class1'],
                   ['red', 'L', 13.5, 'class2'],
                   ['blue', 'XL', 15.3, 'class1']])

df.columns = ['color', 'size', 'price', 'classlabel']
df
```

	color	size	price	classlabel
0	green	M	10.1	class1
1	red	L	13.5	class2
2	blue	XL	15.3	class1

```
                0   1     2       3
0   green   M  10.1  class1
1     red   L  13.5  class2
2    blue  XL  15.3  class1
```

Mapping ordinal features

- Convert the categorical string values into integers
  - ✓ Let's assume the numerical difference between features
  - ✓ XL=L+1=M+2

```
size_mapping = {'XL': 3,
                'L': 2,
                'M': 1}

df['size'] = df['size'].map(size_mapping)
df
```

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

- Reverse mapping
  - ✓ Reverse-mapping dictionary

```
inv_size_mapping = {v: k for k, v in size_mapping.items()}
df['size'].map(inv_size_mapping)
```

```
0     M
1     L
2    XL
Name: size, dtype: object
```

Encoding class labels

- Many machine learning libraries : class labels ➡ encoded as integer values

```
import numpy as np

# create a mapping dict
# to convert class labels from strings to integers
class_mapping = {label: idx for idx, label in enumerate(np.unique(df['classlabel']))}
class_mapping
```

```
{'class1': 0, 'class2': 1}
```

```
array(['class1', 'class2'], dtype=object)
```

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

✓ Use the mapping dictionary to transform the class labels into integers

```
# to convert class labels from strings to integers
df['classlabel'] = df['classlabel'].map(class_mapping)
df
```

	color	size	price	classlabel
0	green	1	10.1	0
1	red	2	13.5	1
2	blue	3	15.3	0

✓ Reverse the key-value pairs in the mapping dictionary

```
# reverse the class label mapping
inv_class_mapping = {v: k for k, v in class_mapping.items()}
df['classlabel'] = df['classlabel'].map(inv_class_mapping)
df
```

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

✓ *LabelEncode*r class in scikit-learn

```
from sklearn.preprocessing import LabelEncoder

# Label encoding with sklearn's LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

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

```
# reverse mapping
class_le.inverse_transform(y)
```

```
array(['class1', 'class2', 'class1'], dtype=object)
```

Performing one-hot encoding on nominal features

- Nominal features
  - ✓ Don't imply any order (ex. T-shirt color)

```
X = df[['color', 'size', 'price']].values

color_le = LabelEncoder()
X[:, 0] = color_le.fit_transform(X[:, 0])
X
```

```
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

  - ✓ Blue=0, green=1, red=2

- One-hot encoding

```
from sklearn.preprocessing import OneHotEncoder
X = df[['color', 'size', 'price']].values
color_ohe=OneHotEncoder()
color_ohe.fit_transform(X[:,0].reshape(-1,1)).toarray()
```

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

```
   color  size  price  classlabel
0  green     1   10.1           0
1    red     2   13.5           1
2   blue     3   15.3           0
```

- Selectively transform columns in a multi-feature array

```
from sklearn.compose import ColumnTransformer
X = df[['color', 'size', 'price']].values
c_transf=ColumnTransformer([
    ('onehot',OneHotEncoder(),[0]),
    ('nothing','passthrough',[1,2])
    ])
c_transf.fit_transform(X).astype(float)
```

```
array([[ 0. ,  1. ,  0. ,  1. , 10.1],
       [ 0. ,  0. ,  1. ,  2. , 13.5],
       [ 1. ,  0. ,  0. ,  3. , 15.3]])
```

...

Performing one-hot encoding on nominal features

- More convenient way : *get_dummies* method in pandas
  - ✓ *get_dummies* method only convert string columns and leave all other columns unchanged

```
>>> pd.get_dummies(df[['price', 'color', 'size']])
   price  size  color_blue  color_green  color_red
0   10.1     1           0            1          0
1   13.5     2           0            0          1
2   15.3     3           1            0          0
```

```
   color  size  price  classlabel
0  green     1   10.1           0
1    red     2   13.5           1
2   blue     3   15.3           0
```

  - ✓ Drop the first column by passing a *True* argument to *the drop_first* parameter

```
>>> pd.get_dummies(df[['price', 'color', 'size']], drop_first=True)
   price  size  color_green  color_red
0   10.1     1            1          0
1   13.5     2            0          1
2   15.3     3            0          0
```

  - ✓ Drop a redundant column via the *OneHotEncoder*

```
color_ohe=OneHotEncoder(categories='auto',drop='first')
c_transf=ColumnTransformer([
    ('onehot',color_ohe,[0]),
    ('nothing','passthrough',[1,2])
    ])
c_transf.fit_transform(X).astype(float)
```

```
array([[ 1. ,  0. ,  1. , 10.1],
       [ 0. ,  1. ,  2. , 13.5],
       [ 0. ,  0. ,  3. , 15.3]])
```

Partitioning a dataset into separate training and test sets

- Wine dataset

```
>>> import pandas as pd
>>> import numpy as np
>>> df_wine = pd.read_csv('https://archive.ics.uci.edu/'
                          'ml/machine-learning-databases/wine/wine.data',
                          header=None)
>>> df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                       'Alcalinity of ash', 'Magnesium', 'Total phenols',
                       'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                       'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                       'Proline']
>>> print('Class labels', np.unique(df_wine['Class label']))
Class labels [1 2 3]
>>> df_wine.head()
   Class label  Alcohol   ...    OD280/OD315 of diluted wines  Proline
0            1    14.23   ...                          3.92     1065
1            1    13.20   ...                          3.40     1050
2            1    13.16   ...                          3.17     1185
3            1    14.37   ...                          3.45     1480
4            1    13.24   ...                          2.93      735

[5 rows x 14 columns]
```

```
>>> df_wine.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class label                   178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity of ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64
 6   Total phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315 of diluted wines  178 non-null    float64
 13  Proline                       178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

- Classes 1,2,3 which refer to the different types of grape grown in the same region in Italy but derived from different wine cultivars

```
df_wine = pd.read_csv('wine.data', header=None)
```

Partitioning a dataset into separate training and test sets

- train_test_spilit function from scikit-learn's model_selection submodule

```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =    train_test_split(X, y,
                     test_size=0.3,
                     random_state=0,
                     stratify=y)
```

30% for test set

Both training and test datasets have the same class proportions as the original dataset

```
>>> df_wine.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class label                   178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity of ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64
 6   Total phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315 of diluted wines  178 non-null    float64
 13  Proline                       178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

Bringing features onto the same scale

- Feature scaling: normalization and standardization
- Normalization : min-max scaling

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{\min}}{x_{\max} - x_{\min}}$$

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

- Standardization

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

```
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

```
>>> ex = np.array([0, 1, 2, 3, 4, 5])
>>> print('standardized:', (ex - ex.mean()) / ex.std())
standardized: [-1.46385011 -0.87831007 -0.29277002  0.29277002  0.87831007  1.46385011]
>>> print('normalized:', (ex - ex.min()) / (ex.max() - ex.min()))
normalized: [0.  0.2 0.4 0.6 0.8 1. ]
```

Data Preprocessing

핸즈온 머신러닝 2판

CHAPTER 2 머신러닝 프로젝트 처음부터 끝까지
2.1 실제 데이터로 작업하기
2.2 큰 그림 보기
2.3 데이터 가져오기
2.4 데이터 이해를 위한 탐색과 시각화
2.5 머신러닝 알고리즘을 위한 데이터 준비
2.6 모델 선택과 훈련

2.1 실제 데이터로 작업하기

- 유명한 공개 데이터 저장소
  - ✓ UC 어바인 http://archive.ics.usi.edu/ml
  - ✓ Kaggle datasets http://www.kaggle.com/datasets
  - ✓ 아마존 AWS datasets https://registry.opendata.aws
- 메타 포털 (공개 데이터 저장소의 나열)
  - ✓ Data Portals http://dataportals.org
  - ✓ Open Data Monitor http://opendatamonitor.edu
  - ✓ Quandl http://quandl.com
- 인기있는 공개 데이터 저장소 나열
  - ✓ 위키백과 머신러닝 데이터셋 목록 https://goo.gl/SJHN2K
  - ✓ Quora.com https://homl.info/10
  - ✓ Datasets subredit http://www.reddit.com/r/datasets

- 캘리포니아 주택 가격(California Housing Prices) 데이터셋

머신 러닝 workflow

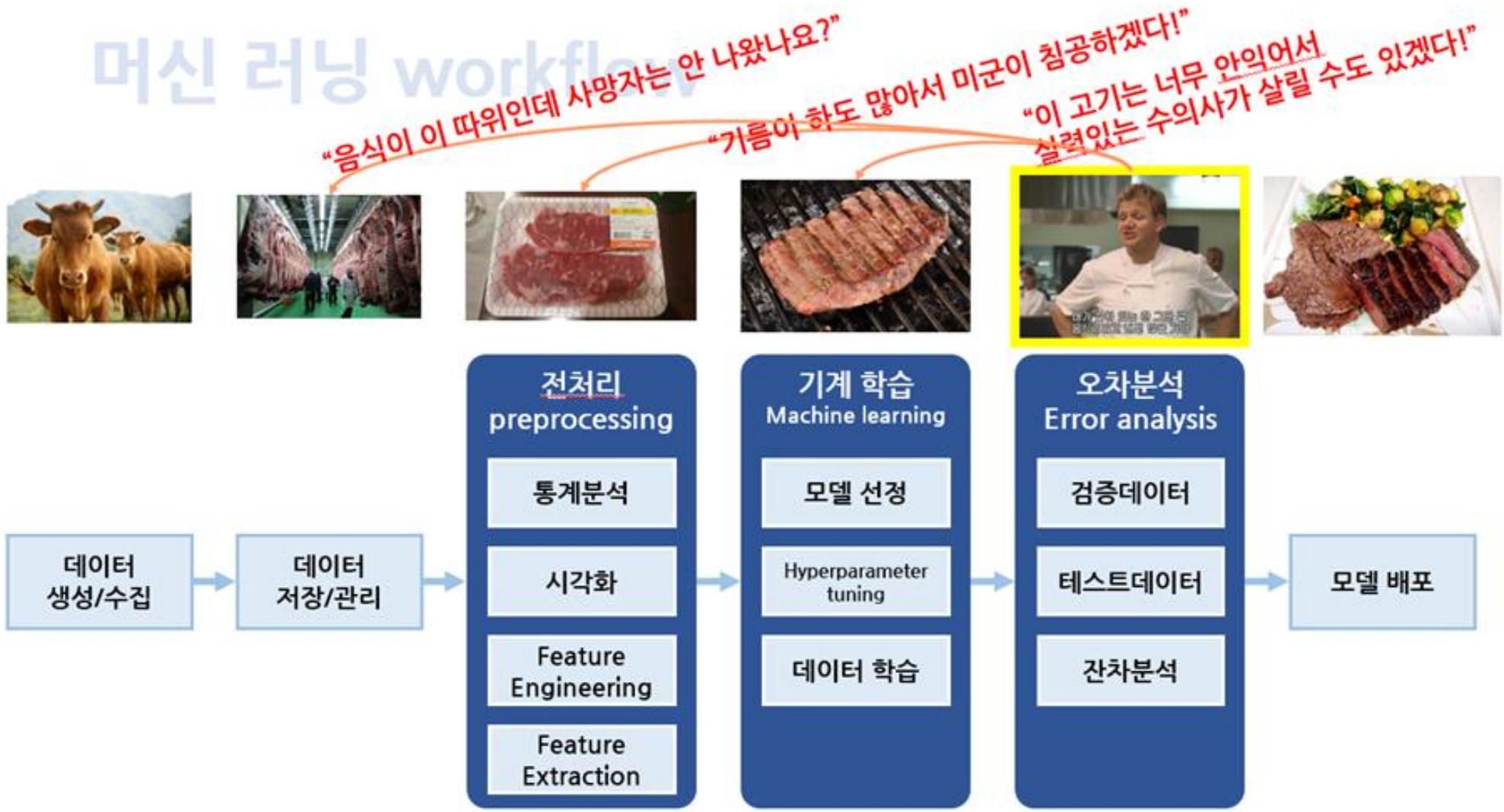





데이터 생성/수집 → 데이터 저장/관리 → **전처리 preprocessing**: 통계분석, 시각화, Feature Engineering, Feature Extraction → **기계 학습 Machine learning**: 모델 선정, Hyperparameter tuning, 데이터 학습 → **오차분석 Error analysis**: 검증데이터, 테스트데이터, 잔차분석 → 모델 배포

[KIER 이제현]

머신 러닝 workflow



[KIER 이제현]

2.2 큰 그림 보기

- 캘리포니아 인구조사 데이터 : 캘리포니아 주택가격 모델 만들기
  - ✓ Block group : 미국 인구조사국의 최소한의 지리적 단위(보통 600~3000명)
  - ✓ Block group 마다 인구, 중간소득, 중간주택가격 등
  - ✓ 이 데이터로 모델을 학습시켜 중간주택 가격을 예측

2.2.1 문제 정의

- 비즈니스의 목적이 무엇인가요?
  - ✓ 모델의 출력(구역의 중간 주택 가격 예측)으로 다른 시스템의 입력에 사용(투자 결정)
- 현재 솔루션은 어떻게 구성되어 있나요?
  - ✓ 전문가의 수동 예측, 추정치가 20%이상 벗어나기도 함
  - ✓ 중간주택가격 예측 모델을 훈련시키자! (수천 개 구역의 중간주택가격 데이터)
- 문제 정의
  - ✓ (**지도학습**,비지도학습), **다중회귀**(예측에 사용할 특성이 여러 개), **단변량 회귀**(각 구역에 하나의 값 예측)-다변량 회귀(각 구역에 여러 값 예측)

2.2.2 성능 측정 지표 선택

- 회귀 문제의 전형적인 성능지표:

Equation 2-1. Root Mean Square Error (RMSE)

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

Equation 2-2. Mean absolute error (MAE)

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

- 중간주택가격 예측: 회귀 (regression) 문제
- 카테고리 (저렴, 보통, 고가) 예측: 분류(classification) 문제

2.3 데이터 가져오기

- 데이터 파일 : datasets/housing/housing.csv
- 캘리포니아 주택가격 모델 만들기

```
import os
import tarfile
import urllib.request

HOUSING_PATH = os.path.join("datasets","housing")

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)

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

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

```
>>> housing.head()
   longitude  latitude  ...  median_house_value  ocean_proximity
0    -122.23     37.88  ...            452600.0         NEAR BAY
1    -122.22     37.86  ...            358500.0         NEAR BAY
2    -122.24     37.85  ...            352100.0         NEAR BAY
3    -122.25     37.85  ...            341300.0         NEAR BAY
4    -122.25     37.85  ...            342200.0         NEAR BAY
```

```
>>> 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
```

숫자형

```
>>> housing.describe()
          longitude      latitude  ...  median_income  median_house_value
count  20640.000000  20640.000000  ...   20640.000000        20640.000000
mean    -119.569704     35.631861  ...       3.870671       206855.816909
std        2.003532      2.135952  ...       1.899822       115395.615874
min     -124.350000     32.540000  ...       0.499900        14999.000000
25%     -121.800000     33.930000  ...       2.563400       119600.000000
50%     -118.490000     34.260000  ...       3.534800       179700.000000
75%     -118.010000     37.710000  ...       4.743250       264725.000000
max     -114.310000     41.950000  ...      15.000100       500001.000000

[8 rows x 9 columns]
```

널 값 제외

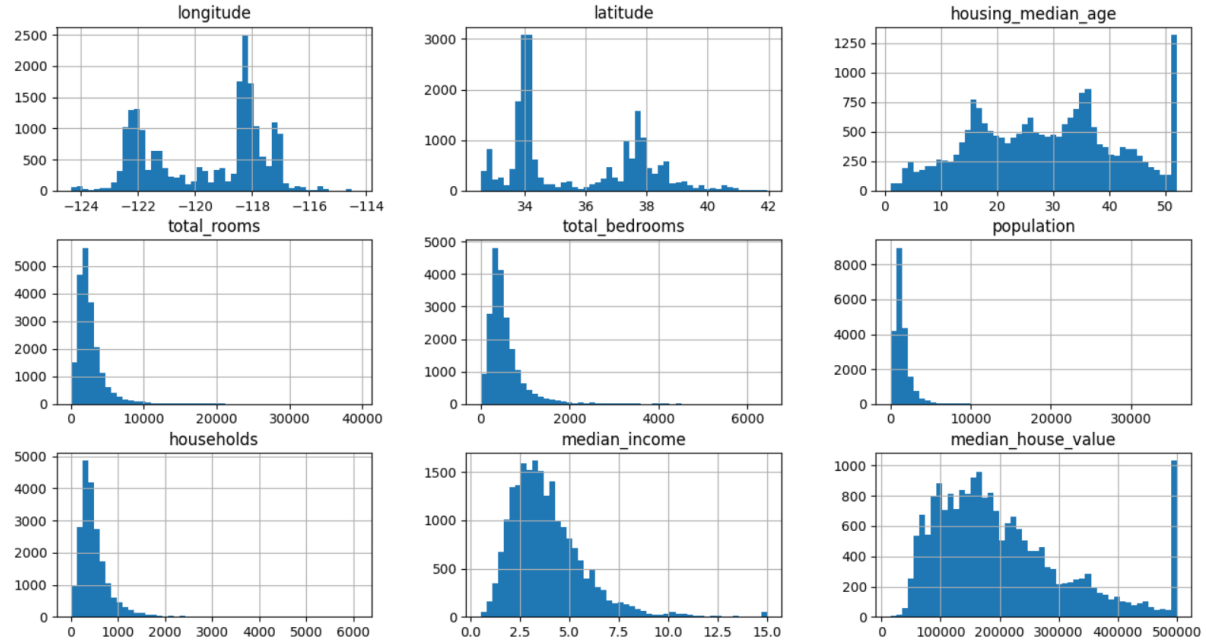
백 분위 수

전체 관측 값에서 주어진 백분율이 속하는 하위 부분의 값

데이터 형태를 빠르게 검토하는 방법: 숫자형 특성의 히스토그램

-특성 값들의 스케일이 많이 다름
-오른 쪽으로 많이 뻗어 있음

```
>>> import matplotlib.pyplot as plt
>>> housing.hist(bins=50, figsize=(20,15))
array([[<AxesSubplot:title={'center':'longitude'}>,
        <AxesSubplot:title={'center':'latitude'}>,
        <AxesSubplot:title={'center':'housing_median_age'}>],
       [<AxesSubplot:title={'center':'total_rooms'}>,
        <AxesSubplot:title={'center':'total_bedrooms'}>,
        <AxesSubplot:title={'center':'population'}>],
       [<AxesSubplot:title={'center':'households'}>,
        <AxesSubplot:title={'center':'median_income'}>,
        <AxesSubplot:title={'center':'median_house_value'}>]],
      dtype=object)
>>> plt.show()
```



15-> 15만 달러

한계 값 설정

2.3.4 테스트 세트 만들기

- 무작위로 샘플을 선택 (20% 정도)

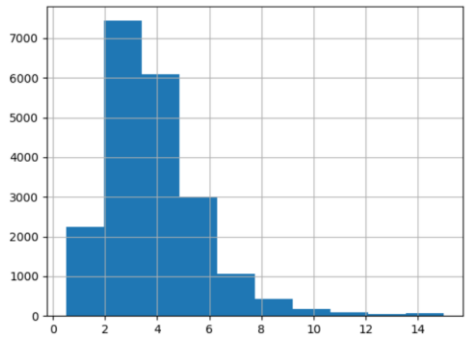
```
import numpy as np

# 예시로 만든 것입니다. 실전에서는 사이킷런의 train_test_split()를 사용하세요.₩n",
def split_train_test(data, test_ratio):
    np.random.seed(42)
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]

train_set, test_set = split_train_test(housing, 0.2)
len(train_set)
len(test_set)
```

```
>>> len(train_set)
16512
>>> len(test_set)
4128
>>>
```

```
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()
```

```
>>> test_set.head()
       longitude  latitude  ...  median_house_value  ocean_proximity
20046    -119.01     36.06  ...             47700.0           INLAND
3024     -119.46     35.14  ...             45800.0           INLAND
15663    -122.44     37.80  ...            500001.0         NEAR BAY
20484    -118.72     34.28  ...            218600.0        <1H OCEAN
9814     -121.93     36.62  ...            278000.0       NEAR OCEAN

[5 rows x 10 columns]
```

```
>>> housing["median_income"].hist()
<AxesSubplot:>
>>> plt.show()
```

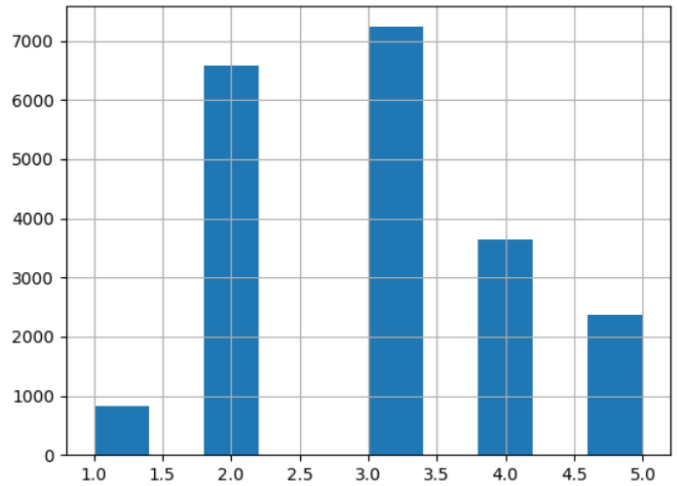


- 계층적 샘플링:
  - ✓ 계층이라는 동질의 그룹으로 나누고 테스트 세트가 전체 인구를 대표하도록 각 계층에서 올바른 수의 샘플을 추출

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf]
                               labels=[1, 2, 3, 4, 5])

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

```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

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




```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]

strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

계층 샘플링에 의한 테스트 세트가 전체 데이터 세트의 소득 카테고리 비율과 거의 같음

Income_cat 특성삭제

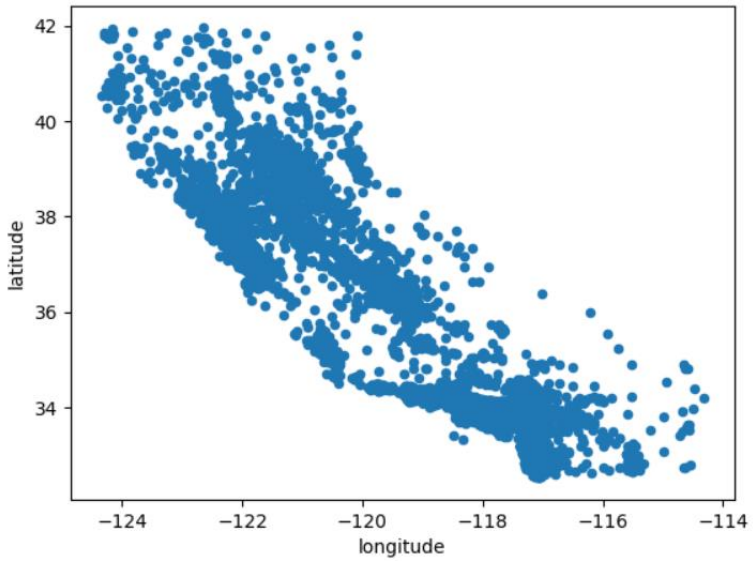
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

열 삭제

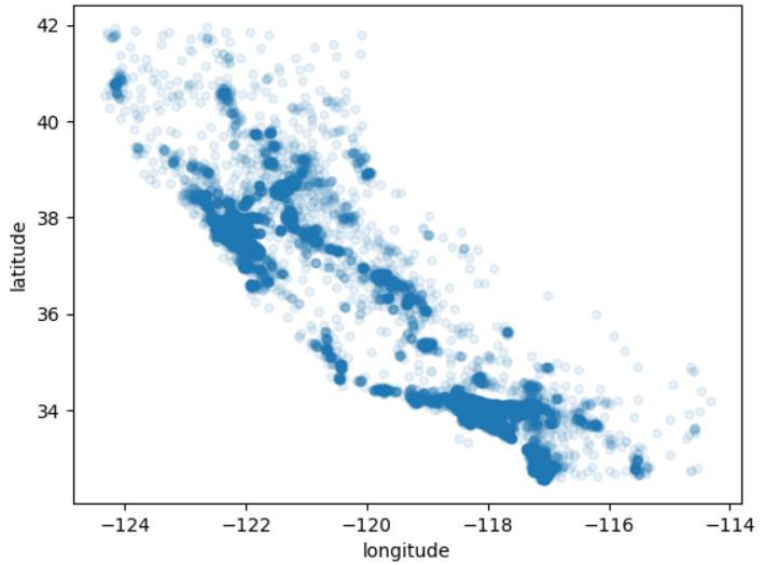
2.4 데이터 이해를 위한 탐색과 시각화

- 훈련(Training) 세트에 대해서만 탐색

```
housing = strat_train_set.copy()

housing.plot(kind="scatter", x="longitude", y="latitude")
plt.show()
```




```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
plt.show()
```

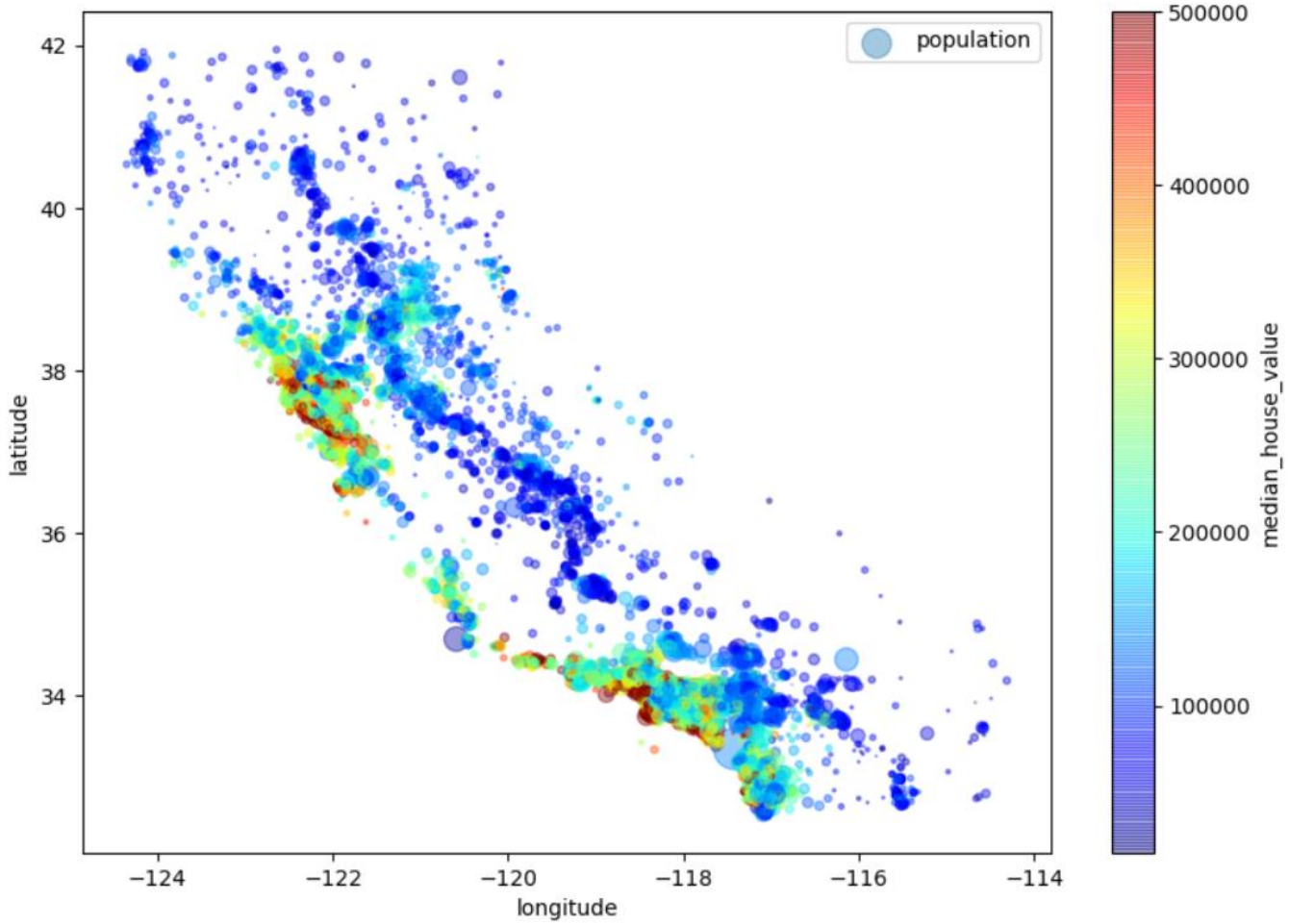


데이터가 밀집된 영역을 볼 수 있음
(Bay Area, LA 근처, San Diego 지역 밀집,
센트럴 밸리를 따라 밀집된 지역의 긴 띠)

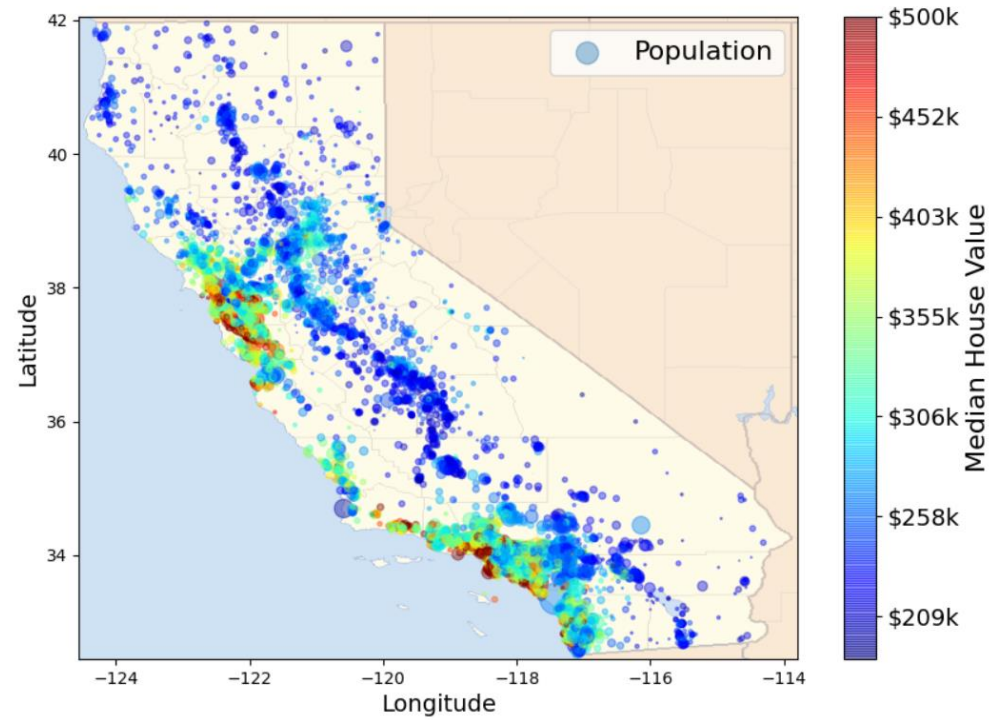
- 주택 가격

```
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()
plt.show()
```

- ✓ 원의 반지름: 구역의 인구(s)
- ✓ 색상: 가격(c)
- ✓ 컬러 맵 jet (cmap)
- ✓ 주택 가격은 지역(해변)과 인구밀도에 높은 관련



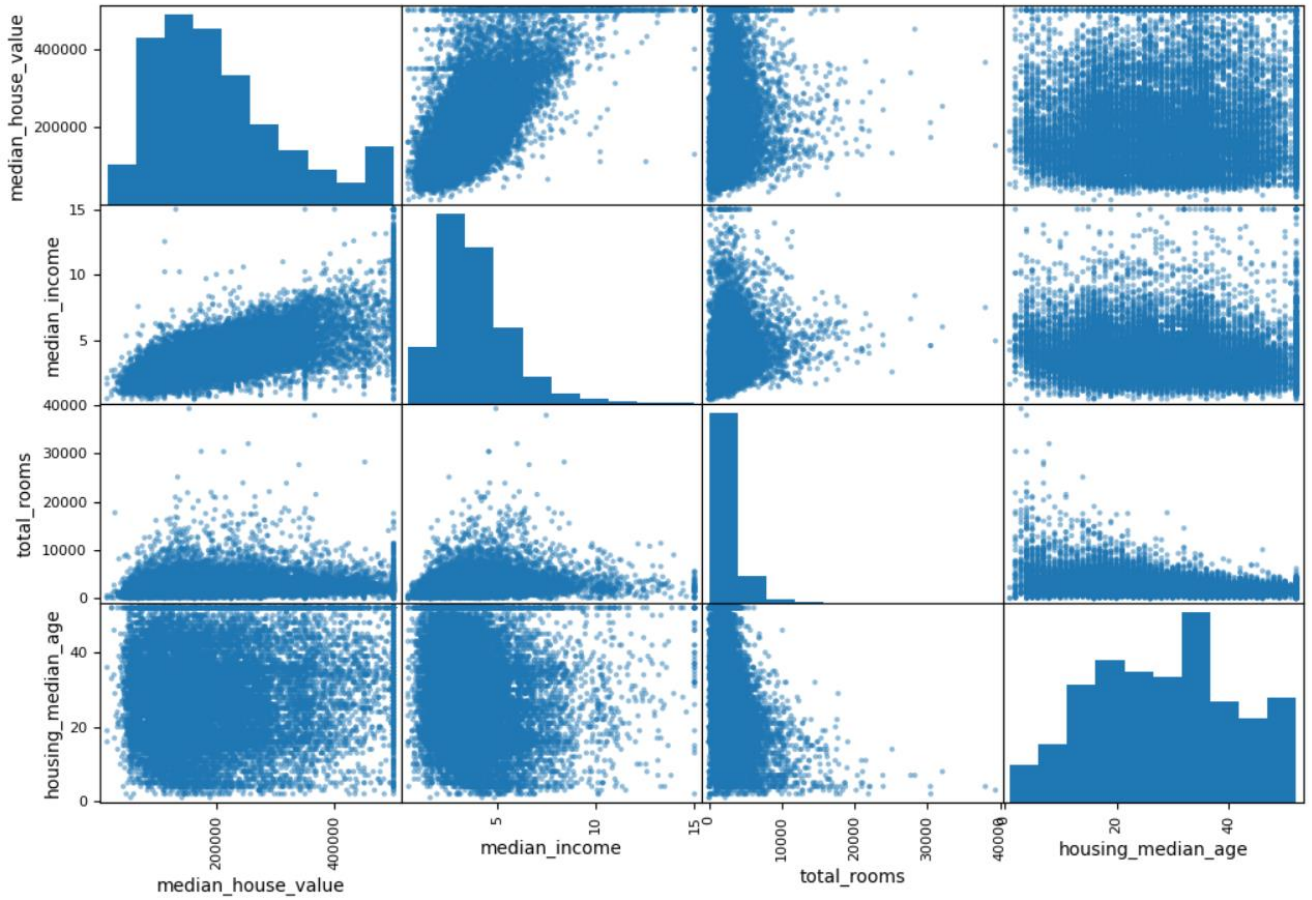
```
# Download the California image
PROJECT_ROOT_DIR = os.path.join("datasets","housing")
images_path = os.path.join(PROJECT_ROOT_DIR, "images", "end_to_end_project")
os.makedirs(images_path, exist_ok=True)
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
filename = "california.png"
print("Downloading", filename)
url = DOWNLOAD_ROOT + "images/end_to_end_project/" + filename
urllib.request.urlretrieve(url, os.path.join(images_path, filename))

import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                       s=housing['population']/100, label="Population",
                       c="median_house_value", cmap=plt.get_cmap("jet"),
                       colorbar=False, alpha=0.4,
                      )
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar(ticks=tick_values/prices.max())
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
#save_fig("california_housing_prices_plot")
```



- 상관관계 조사 -1<상관계수<1

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

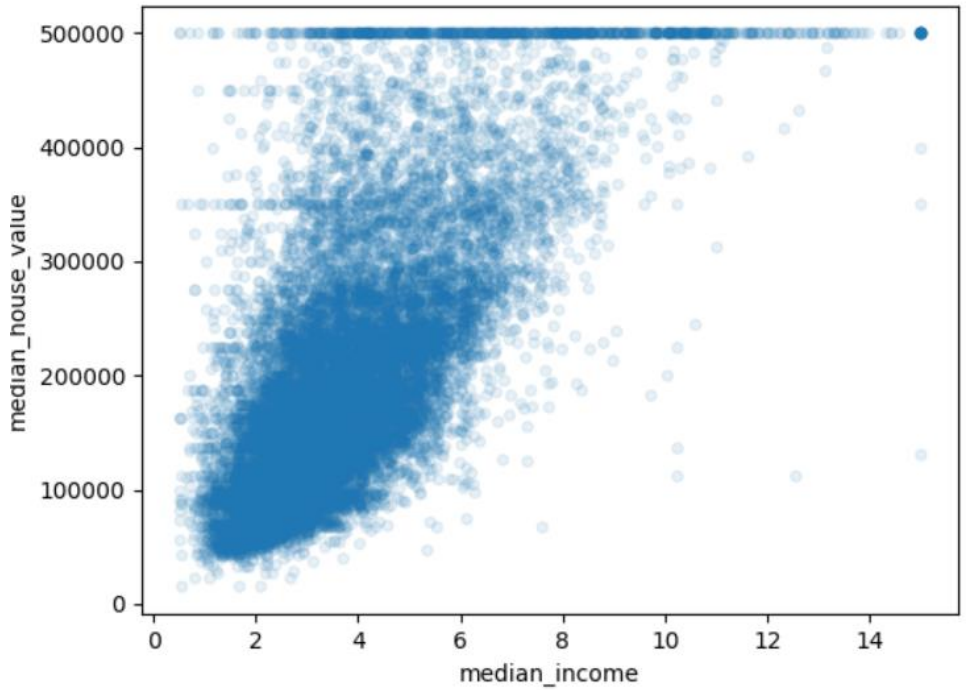
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```




```
from pandas.plotting import scatter_matrix

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

- # 상관관계 조사 -1<상관계수<1

```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
plt.show()
```



- ✓ 매우 강한 상관관계
- ✓ 가격 제한값 $500,000에서 수평선
- ✓ $450,000, $350,000, $280,000 에서도 수평선
- ✓ 해당구역 제거 하는 것이 좋음

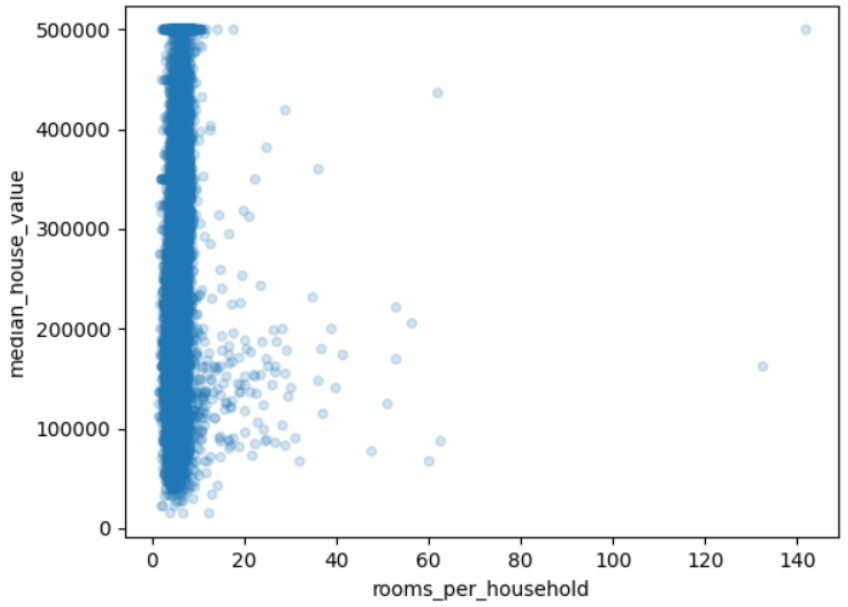
- # 특성 조합
  - ✓ 가구당 방 개수
  - ✓ 가구당 인원 등

- 특성 조합
  - ✓ 가구당 방 개수, 침실 비율, 가구당 인원 등

```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
plt.show()

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)
|
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.show()

housing.describe()
```

```
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
```

	longitude	latitude	...	bedrooms_per_room	population_per_household
count	16512.000000	16512.000000	...	16354.000000	16512.000000
mean	-119.575834	35.639577	...	0.212878	3.096437
std	2.001860	2.138058	...	0.057379	11.584826
min	-124.350000	32.540000	...	0.100000	0.692308
25%	-121.800000	33.940000	...	0.175304	2.431287
50%	-118.510000	34.260000	...	0.203031	2.817653
75%	-118.010000	37.720000	...	0.239831	3.281420
max	-114.310000	41.950000	...	1.000000	1243.333333



2.5 머신러닝을 위한 데이터 준비

- 데이터 준비: 함수를 이용한 자동화
  - ✓ 데이터 변환의 손쉬운 반복
  - ✓ 향후 사용할 변환 라이브러리의 점진적 구축
  - ✓ 실제 시스템에서 데이터 입력 전 변환에 사용
  - ✓ 데이터 변환의 시도 및 좋은 데이터 조합의 확인 편리성

- 예측 변수와 레이블의 분리

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

- 데이터 정제
  - ✓ total-_bedrooms 특성에 값이 없는 경우: 3가지 처리 방법
    - ✓ 해당구역 제거
    - ✓ 전체 특성 삭제
    - ✓ 어떤 값으로 채움(0, 평균, 중간값 등)
    - ✓ dropna(), drop(), fillna() 메서드 이용

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows

sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # 옵션 1
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # 옵션 2
median = housing["total_bedrooms"].median() # 옵션 3
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True)

sample_incomplete_rows
```

```
>>> sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
>>> sample_incomplete_rows.dropna(subset=["total_bedrooms"])
Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedr
, median_income, ocean_proximity]
Index: []
>>> sample_incomplete_rows.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5 entries, 4629 to 19252
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           5 non-null      float64
 1   latitude            5 non-null      float64
 2   housing_median_age  5 non-null      float64
 3   total_rooms         5 non-null      float64
 4   total_bedrooms      0 non-null      float64
 5   population          5 non-null      float64
 6   households          5 non-null      float64
 7   median_income       5 non-null      float64
 8   ocean_proximity     5 non-null      object
```

```
>>> sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True)
>>> sample_incomplete_rows.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5 entries, 4629 to 19252
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           5 non-null      float64
 1   latitude            5 non-null      float64
 2   housing_median_age  5 non-null      float64
 3   total_rooms         5 non-null      float64
 4   total_bedrooms      5 non-null      float64
 5   population          5 non-null      float64
 6   households          5 non-null      float64
 7   median_income       5 non-null      float64
 8   ocean_proximity     5 non-null      object
```

- 데이터 정제
  - ✓ Scikit-learn: SimpleImputer – 누락된 값을 손 쉽게 다룸

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")

housing_num = housing.drop("ocean_proximity", axis=1)
# 다른 방법: housing_num = housing.select_dtypes(include=[np.number])

imputer.fit(housing_num)
imputer.statistics_
housing_num.median().values

X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns,index=housing_num.index)

housing_tr.loc[sample_incomplete_rows.index.values]
imputer.strategy
housing_tr.head()
```

```
>>> imputer.statistics_
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.
        408.    ,    3.5409])
>>> housing_num.median().values
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.
        408.    ,    3.5409])
```

- 텍스트와 범주형 특성 다루기
  - ✓ 특성 ocean_proximity 만 텍스트 데이터
  - ✓ 텍스트에서 숫자로 바꾸기...

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

from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]

ordinal_encoder.categories_
```

```
>>> housing_cat.head(10)
       ocean_proximity
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
```

```
>>> ordinal_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
      dtype=object)]
```

```
>>> housing_cat_encoded[:10]
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

  - ✓ 원-핫 인코딩

```
from sklearn.preprocessing import OneHotEncoder

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

housing_cat_1hot.toarray()

cat_encoder.categories_
```

```
>>> housing_cat_1hot.toarray()
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

```
>>> cat_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN']
      dtype=object)]
```

- 나 만의 변환기
  - ✓ TransformerMixin 상속: fit, transform, fit_transform 자동으로 생성
  - ✓ BaseEstimator 상속: 하이퍼 파라미터 튜닝 메서드 get_params(), set_params() 얻게 됨

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # *args 또는 **kargs 없음₩n",
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # 아무것도 하지 않습니다₩n",
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_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.to_numpy())
```

- 특성 스케일링
  - ✓ 데이터 변환 중 가장 중요한 것: 특성 스케일링(feature scaling)
  - ✓ 입력 숫자 특성들의 스케일이 많이 다르면 머신러닝이 잘 동작하지 않음
  - ✓ 표준화(standardization)와 min-max 스케일링(정규화:normalization)

  - ✓ Scikit-learn: MinMaxScaler, StandardScaler

- 변환 파이프라인
  - ✓ 변환 단계의 정확한 순서에 따른 실행: pipeline class (scikit-learn)

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

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

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

- 변환 파이프라인
  - ✓ 하나의 변환기로 각 열마다 적절한 변환을 적용: ColumnTransformer(scikit-learn)
    - ✓ 수치형 열 이름의 리스트 만들기
    - ✓ 범주형 열 이름의 리스트 만들기
    - ✓ ColumnTransformer 클래스 객체 만들기
      - ✓ 튜플 리스트 (이름, 변환기, 변환기가 적용될 열 이름)

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

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

housing_prepared = full_pipeline.fit_transform(housing)
```

2.6 모델 선택과 훈련

- 문제정의 ➡ 데이터 읽기 및 탐색 ➡ 학습/테스트 세트 분리 ➡ 데이터 정제 ➡ 변환 파이프 라인 ➡ 머신러닝 모델 선택 및 훈련

- 학습세트로 학습 및 평가: 선형회귀 모델 학습

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
# 훈련 샘플 몇 개를 사용해 전체 파이프라인을 적용해 보겠습니다₩n",
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("예측:", lin_reg.predict(some_data_prepared))
print("레이블:", list(some_labels))
```

```
>>> print("예측:", lin_reg.predict(some_data_prepared))
예측: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
>>> print("레이블:", list(some_labels))
레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

```
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

from sklearn.metrics import mean_absolute_error
lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

```
>>> lin_rmse
68628.19819848922

>>> lin_mae
49439.89599001896
```