CHAPTER 06

Data preprocessing

01 Basics of data preprocessing

02 Strategies for data preprocessing

03 Example practice

01
Basics of data preprocessing

01 데이터 전처리의 기초

1. 데이터 전처리의 개념

- 데이터 전처리(data preprocessing) : 머신러닝 모델에 훈련 데이터를 입력하기 전에 데이터를 가공
- 넘파이나 판다스 같은 머신러닝의 핵심 도구,
  맷플롯립과 시본 같은 데이터 시각화 도구를 활용하여 실제 데이터를 정리
- 머신러닝 기초 수식

$$y = f(X)$$

  - 데이터 X를 머신러닝 함수 f( )에 넣으면 그 결과 y가 나옴
  - 데이터 X는 훈련 데이터(train data)와 테스트 데이터(test data)가 모두 같은 구조를 갖는 피쳐(feature)이어야 함

2. 데이터 품질 문제

2.1 데이터 분포의 지나친 차이

- 피쳐 간 최대값과 최소값 차이가 크게 나는 경우
- 학습에 영향을 줄 수 있기 때문에 데이터의 스케일(scale)을 맞춰줌
  - 데이터의 최댓값과 최솟값을 0에서 1 사이 값으로 바꾸거나 표준 정규분포 형태로 나타내는 등

01 데이터 전처리의 기초

2.2 기수형 데이터와 서수형 데이터

- 기수형 데이터와 서수형 데이터는 일반적으로 숫자로 표현되지 않음
- 컴퓨터가 이해할 수 있는 숫자 형태의 정보로 변형

2.3 결측치

- 결측치(missing data) : 실제로 존재하지만 데이터베이스 등에 기록되지 않는 데이터
- 해당 데이터를 빼고 모델을 돌릴 수 없기 때문에 결측치 처리 전략을 세워 데이터를 채워 넣음

2.4 이상치

- 이상치(outlier) : 극단적으로 크거나 작은 값
- 단순히 데이터 분포의 차이와는 다름
- 데이터 오기입이나 특이 현상 때문에 나타남

02
Strategies for data preprocessing

1. 결측치 처리하기 : 드롭과 채우기

- 데이터를 삭제하거나 데이터를 채움
  - 데이터가 없으면 해당 행이나 열을 삭제
  - 평균값, 최빈값, 중간값 등으로 데이터를 채움

02 데이터 전처리의 전략

In [1]:

```
import pandas as pd
import numpy as np

raw_data = {'first_name': ['Jason', np.nan, 'Tina', 'Jake',
'Amy'],
            'last_name': ['Miller', np.nan, 'Ali', 'Milner',
'Cooze'],
            'age': [42, np.nan, 36, 24, 73],
            'sex': ['m', np.nan, 'f', 'm', 'f'],
            'preTestScore': [4, np.nan, np.nan, 2, 3],
            'postTestScore': [25, np.nan, np.nan, 62, 70]}

df = pd.DataFrame(raw_data, columns = ['first_name',
'last_name', 'age', 'sex', 'preTestScore', 'postTestScore'])
df
```

Out [1]:

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

02 데이터 전처리의 전략

- 결측치를 확인할 때 isnull 함수 사용
  - NaN 값이 존재할 경우 True, 그렇지 않을 경우 False 출력

```
In [2]: df.isnull().sum() / len(df)
```

```
Out [2]: first_name        0.2
         last_name         0.2
         age               0.2
         sex               0.2
         preTestScore      0.4
         postTestScore     0.4
         dtype: float64
```

- sum 함수로 True인 경우 모두 더하고 전체 데이터 개수로 나누어 열별 데이터 결측치 비율을 구함

02 데이터 전처리의 전략

1.1 드롭

- 드롭(drop) : 결측치가 나온 열이나 행을 삭제
- dropna 사용하여 NaN이 있는 모든 데이터의 행을 없앰

In [3]:
```
df.dropna()
```

Out [3]:

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

[하나 더 알기] 드롭의 결과물 저장

- 드롭과 관련된 대부분의 명령어들은 실제 드롭한 결과를 반환하나 객체에 드롭 결과를 저장하지는 않음
- 드롭의 결과물을 저장하려면 다른 변수에 재할당
- 또는 매개변수 inplace=True 사용
  - 자체적으로 값이 변하면 이후에 해당 데이터를 불러 쓰거나 다시 코드를 실행할 때 문제가 되기 때문에 새로운 값에 복사하는 것이 좋음

In [4]:	df_no_missing = df.dropna() df_no_missing

02 데이터 전처리의 전략

- 매개변수 how로 조건에 따라 결측치를 지움
  - how에는 매개변수 'all'과 'any' 사용
  - 'all'은 행에 있는 모든 값이 NaN일 때 해당 행을 삭제
  - 'any'는 하나의 NaN만 있어도 삭제
- dropna의 기본 설정은 'any'라서 모든 결측치를 지움

In [5]:
```
df_cleaned = df.dropna(how='all')
df_cleaned
```

Out [5]:

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

02 데이터 전처리의 전략

- 열 값이 모두 NaN일 경우에는 축(axis)을 추가하여 삭제

In [6]:
```
df['location'] = np.nan
df.dropna(axis=1, how='all')
```

Out [6]:

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

- location이라는 열을 추가하여 값들을 모두 NaN으로 한 후 axis=1로 location 열만 삭제

02 데이터 전처리의 전략

- 매개변수 threshfh 데이터의 개수를 기준으로 삭제
  - thresh=1 지정하면 데이터가 한 개라도 존재하는 행은 남김
  - thresh=5 지정하면 데이터가 다섯 개 이상 있어야 남김

In [7]:
```
df.dropna(axis=0, thresh=1)
```

Out [7]:

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
2	Tina	Ali	36.0	f	NaN	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

In [8]:
```
df.dropna(thresh=5)
```

Out [8]:

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

02 데이터 전처리의 전략

1.2 채우기

- 채우기(fill) : 비어있는 값을 채움
- 일반적으로 드롭한 후에 남은 값들을 채우기 처리
- 평균, 최빈값 등 데이터의 분포를 고려해서 채움
- 함수 fillna 사용

In [9]:
```
df.fillna(0)
```

Out [9]:

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	0.0
1	0	0	0.0	0	0.0	0.0	0.0
2	Tina	Ali	36.0	f	0.0	0.0	0.0
3	Jake	Milner	24.0	m	2.0	62.0	0.0
4	Amy	Cooze	73.0	f	3.0	70.0	0.0

02 데이터 전처리의 전략

- 빈 값에 평균값을 채우려면 열 단위의 평균값을 계산하여 해당 열에만 값을 채움
  - 매개변수 inplace는 변경된 값을 리턴시키는 것이 아니고 해당 변수 자체의 값을 변경

In [10]:
```
df["preTestScore"].fillna(df["preTestScore"].mean(),
inplace=True)
df
```

Out [10]:

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
1	NaN	NaN	NaN	NaN	3.0	NaN	NaN
2	Tina	Ali	36.0	f	3.0	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

02 데이터 전처리의 전략

- 열별 분포를 고려하여 채울 수 있음
  - groupby 함수로 각 인덱스의 성별에 따라 빈칸을 채움

In [11]:	df.groupby("sex")["postTestScore"].transform("mean")
Out [11]:	0 43.5 1 NaN 2 70.0 3 43.5 4 70.0 Name: postTestScore, dtype: float64

02 데이터 전처리의 전략

- fillna 함수 안에 transform을 사용하여 인덱스를 기반으로 채울 수 있음
  - 일반적으로 쓰이는 기법

In [12]:
```
df["postTestScore"].fillna(
    df.groupby("sex")["postTestScore"].transform("mean"),
inplace=True)
df
```

Out [12]:

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	NaN
1	NaN	NaN	NaN	NaN	3.0	NaN	NaN
2	Tina	Ali	36.0	f	3.0	70.0	NaN
3	Jake	Milner	24.0	m	2.0	62.0	NaN
4	Amy	Cooze	73.0	f	3.0	70.0	NaN

2. 범주형 데이터 처리하기 : 원핫인코딩

- 원핫인코딩(one-hot encoding) : 범주형 데이터의 개수만큼 가변수(dummy variable)를 생성하여 존재 유무를 1 또는 0으로 표현
  - color라는 변수에 {Green, Blue, Yellow} 3개의 값이 있을 때
  - 3개의 가변수를 만들고 각 색상에 인덱스를 지정
    - Green의 인덱스는 0, Blue의 인덱스 1, Yellow의 인덱스는 2로 지정
  - 해당 값이면 1, 아니면 0을 입력

{Green} → [1, 0, 0]

{Blue} → [0, 1, 0]

{Yellow} → [0, 0, 1]

02 데이터 전처리의 전략

- 원핫인코딩을 적용하려면 판다스에서 제공하는 get_dummies 함수를 이용하거나 사이킷런(scikit-learn)에서 제공하는 LabelEncoder나 OneHotEncoder를 이용

In [13]:
```
edges = pd.DataFrame({'source': [0, 1, 2],
                      'target': [2, 2, 3],
                      'weight': [3, 4, 5],
                      'color': ['red', 'blue', 'blue']})
edges
```

Out [13]:

	source	target	weight	color
0	0	2	3	red
1	1	2	4	blue
2	2	3	5	blue

02 데이터 전처리의 전략

In [14]:
```
edges.dtypes
```

Out [14]:
```
source int64
target int64
weight int64
color object
dtype: object
```

In [15]:
```
pd.get_dummies(edges)
```

Out [15]:

	source	target	weight	color_blue	color_red
0	0	2	3	0	1
1	1	2	4	1	0
2	2	3	5	1	0

- get_dummies를 적용하여 범주형 데이터 color에 가변수 추가

02 데이터 전처리의 전략

In [16]:
```
pd.get_dummies(edges["color"])
```

Out [16]:

	blue	red
0	0	1
1	1	0
2	1	0

In [17]:
```
pd.get_dummies(edges[["color"]])
```

Out [17]:

	color_blue	color_red
0	0	1
1	1	0
2	1	0

- 해당 열의 color 값만 따로 추출해서 적용

02 데이터 전처리의 전략

- 필요에 따라 정수형을 객체로 변경해서 처리

	source	target	weight	color
0	0	2	3	red
1	1	2	4	blue
2	2	3	5	blue

그림 6-1 다음 코드에서 다룰 데이터

- weight는 숫자로 되어 있지만 기수형 데이터
- 데이터를 M, L, XL로 변경하여 원핫인코딩을 적용

02 데이터 전처리의 전략

In [18]:
```
weight_dict = {3:"M", 4:"L", 5:"XL"}
edges["weight_sign"] = edges["weight"].map(weight_dict)
weight_sign = pd.get_dummies(edges["weight_sign"])
weight_sign
```

Out [18]:

	L	M	XL
0	0	1	0
1	1	0	0
2	0	0	1

In [19]:
```
pd.concat([edges, weight_sign], axis=1)
```

Out [19]:

	source	target	weight	color	weight_sign	L	M	XL
0	0	2	3	red	M	0	1	0
1	1	2	4	blue	L	1	0	0
2	2	3	5	blue	XL	0	0	1

- 데이터를 원핫인코딩 형태로 변경한 후 필요에 따라 병합이나 연결로 두 가지의 데이터를 합침

3. 범주형 데이터로 변환하여 처리하기 : 바인딩

- 바인딩(binding) : 연속형 데이터를 범주형 데이터로 변환

In [20]:

```
raw_data = {'regiment': ['Nighthawks', 'Nighthawks',
'Nighthawks', 'Nighthawks', 'Dragoons', 'Dragoons',
'Dragoons', 'Dragoons', 'Scouts', 'Scouts', 'Scouts',
'Scouts'],
        'company': ['1st', '1st', '2nd', '2nd', '1st',
'1st', '2nd', '2nd','1st', '1st', '2nd', '2nd'],
        'name': ['Miller', 'Jacobson', 'Ali', 'Milner',
'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger',
'Riani', 'Ali'],
        'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3,
2, 3],
        'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57,
62, 70, 62, 70]}

df = pd.DataFrame(raw_data, columns = ['regiment',
'company', 'name', 'preTestScore', 'postTestScore'])
df
```

02 데이터 전처리의 전략

Out [20]:

	regiment	company	name	preTestScore	postTestScore
0	Nighthawks	1st	Miller	4	25
1	Nighthawks	1st	Jacobson	24	94
2	Nighthawks	2nd	Ali	31	57
3	Nighthawks	2nd	Milner	2	62
4	Dragoons	1st	Cooze	3	70
5	Dragoons	1st	Jacon	4	25
6	Dragoons	2nd	Ryaner	24	94
7	Dragoons	2nd	Sone	31	57
8	Scouts	1st	Sloan	2	62
9	Scouts	1st	Piger	3	70
10	Scouts	2nd	Riani	2	62
11	Scouts	2nd	Ali	3	70

02 데이터 전처리의 전략

- postTestScore에 대한 학점을 측정하는 코드를 작성
  - 데이터 범위를 구분 : 0~25, 25~50, 50~75, 75~100으로 구분
  - 함수 cut 사용
    - bins 리스트에 구간의 시작 값 끝 값을 넣고 구간의 이름을 리스트로 나열
      bins의 원소는 5개이고 group_names는 4개
    - cut 함수로 나눌 시리즈 객체와 구간, 구간의 이름을 넣어주면 해당 값을 바인딩하여 표시해줌

02 데이터 전처리의 전략

In [21]:
```
bins = [0, 25, 50, 75, 100] # bins 정의(0-25, 25-50, 60-75, 75-100)
group_names = ['Low', 'Okay', 'Good', 'Great']
categories = pd.cut(
    df['postTestScore'], bins, labels=group_names)
categories
```

Out [21]:
```
0 Low
1 Great
2 Good
3 Good
4 Good
5 Low
6 Great
7 Good
8 Good
9 Good
10 Good
11 Good
Name: postTestScore, dtype: category
Categories (4, object): ['Low' < 'Okay' < 'Good'
< 'Great']
```

4. 데이터의 크기 맞추기 : 피쳐 스케일링

- 스케일링(scaling) : 데이터 간 범위를 맞춤
  - 몸무게와 키를 하나의 모델에 넣으면 데이터의 범위가 훨씬 넓어져 키가 몸무게에 비해 모델에 과다하게 영향을 줌
- x1과 x2의 변수 범위가 다를 때 하나의 변수 범위로 통일시켜 처리

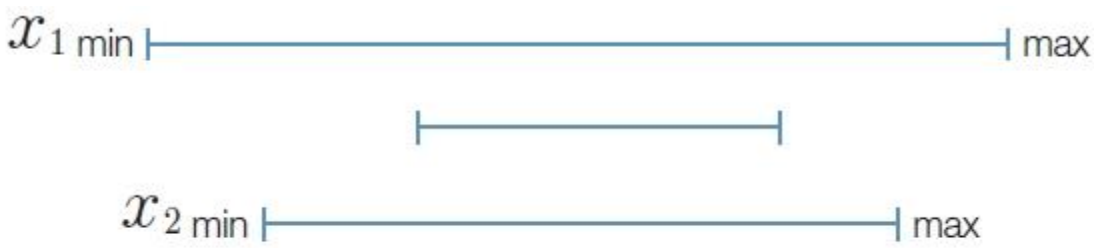

그림 6-2 x_1과 x_2의 변수 범위의 차이

02 데이터 전처리의 전략

- 최솟값-최댓값 정규화(min-max normalization) :
  최솟값과 최댓값을 기준으로 0에서 1, 또는 0에서 지정 값까지로 값의 크기를 변화시킴

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}(new_{\max} - new_{\min}) - new_{\max}$$

  - x는 처리하고자 하는 열, x_i는 이 열의 하나의 값,
    max(x)는 해당 열의 최댓값, min(x)는 해당 열의 최솟값
  - new_max와 new_min은 새롭게 지정되는 값의 최댓값 또는 최솟값

02 데이터 전처리의 전략

- z-스코어 정규화(z-score normalization) :
  기존 값을 표준 정규분포값으로 변환하여 처리

$$z = \frac{x_i - \mu}{\sigma}$$

  - μ는 x 열의 평균값이고 σ는 표준편차
  - 통계학 시간에 배우는 수식과 동일

02 데이터 전처리의 전략

In [22]:
```
df = pd.DataFrame(
    {'A':[14.00,90.20,90.95,96.27,91.21],
     'B':[103.02,107.26,110.35,114.23,114.68],
     'C':['big','small','big','small','small']})

df
```

Out [22]:

	A	B	C
0	14.00	103.02	big
1	90.20	107.26	small
2	90.95	110.35	big
3	96.27	114.23	small
4	91.21	114.68	small

02 데이터 전처리의 전략

- 스케일링할 때는 브로드캐스팅 개념으로 스칼라 값(평균값, 최댓값, 최솟값)과 벡터(열) 값 간 연산

In [23]:	df["A"] - df["A"].min()
Out [23]:	0 0.00 1 76.20 2 76.95 3 82.27 4 77.21 Name: A, dtype: float64

02 데이터 전처리의 전략

- 최솟값-최댓값 정규화 방법에서 최댓값과 최솟값을 따로 구하지 않고 코드로 수식을 나타낼 수 있음

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

In [24]:	(df["A"] - df["A"].min()) / (df["A"].max() - df["A"].min())
Out [24]:	0 0.000000 1 0.926219 2 0.935335 3 1.000000 4 0.938495 Name: A, dtype: float64

02 데이터 전처리의 전략

- z-스코어 정규화 수식 역시 코드로 나타낼 수 있음

In [25]:	`( df["B"] - df["B"].mean() ) / (df["B"].std())`

Out [25]:	`0  -1.405250`
	`1  -0.540230`
	`2  0.090174`
	`3  0.881749`
	`4  0.973556`
	`Name: B, dtype: float64`

03
Example practice

03 Example practice

In [7]:
```
# cut() 함수 사용하여 임의로 구간화
df1.insert(3, 'BMI_bin2', 0) # 구간화용 빈 컬럼 생성

df1['BMI_bin2'] = pd.cut(df1.BMI, bins=[0, 20, 30, 40, 50, 60, 70, 95]
                         , labels=['a', 'b', 'c', 'd', 'e', 'f', 'g'])

df1.head()
```

Out [7]:

	HeartDisease	BMI	BMI_bin	BMI_bin2	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabet
0	No	16.60	a	a	Yes	No	No	3.0	30.0	No	Female	55-59	White	Y
1	No	20.34	b	b	No	No	Yes	0.0	0.0	No	Female	80 or older	White	I
2	No	26.58	b	b	Yes	No	No	20.0	30.0	No	Male	65-69	White	Y
3	No	24.21	b	b	No	No	No	0.0	0.0	No	Female	75-79	White	I
4	No	23.71	b	b	No	No	No	28.0	0.0	Yes	Female	40-44	White	I

In [8]:
```
# BMI_bin2 구간 별 관측치 수 집계

df1.BMI_bin2.value_counts().to_frame().style.background_gradient(cmap='winter')
```

Out [8]:

	BMI_bin2
b	202548
c	86198
a	14699
d	13839
e	2019
f	363
g	129