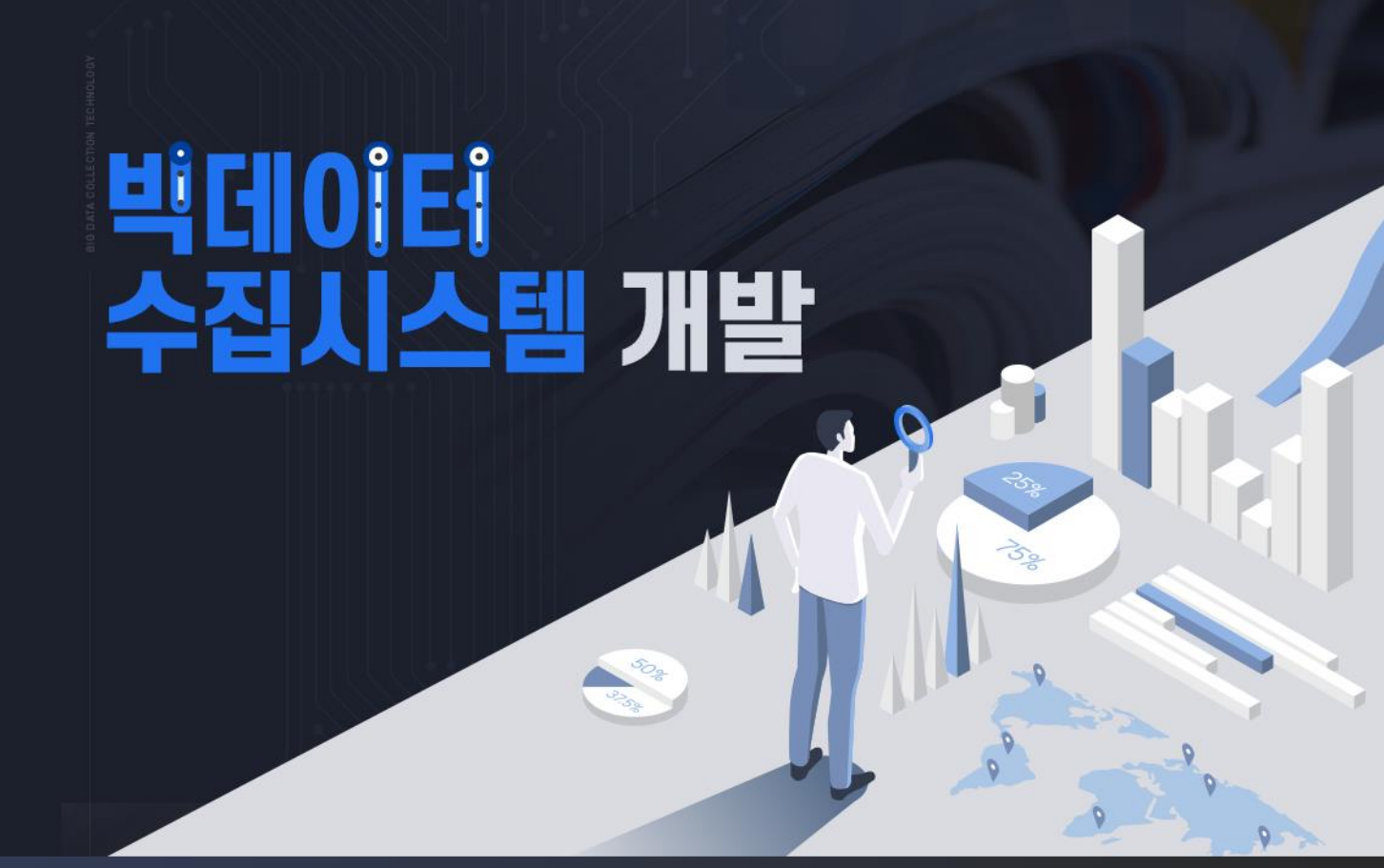


# 데이터 정제와 변환

한국기술교육대학교
온라인평생교육원

## 학습목표

- 입력된 데이터의 누락치, 이상치의 제거와 대체를 통해 올바른 분석 결과를 얻을 수 있다.
- 데이터를 통계 분석에 적절하도록 변환할 수 있다.

## 학습내용

- 데이터 정제
- 데이터 변환

# 데이터 정제

## 1 누락 데이터 처리

- 머신러닝, 딥러닝에서 데이터의 누락은 전체 예측 결과에 많은 영향을 미침

누락된 데이터 / 중복된 데이터 / 이상치 데이터

예측력이 높고 신뢰성 있는 결과를 얻기 위해
바른 처리 필요

- DataFrame에는 데이터가 누락될 경우 NaN(Not a Number)으로 표기됨

데이터 분석 시 보통 데이터의 크기는 수십만 건~수백만 건

엑셀 등의 프로그램에서 눈으로 오류가 있는 데이터를 검증하는 것은 거의 불가능에 가까움

파이썬에서 제공하는 라이브러리, 함수 등을 이용해 누락되거나 특정 범위를 벗어나는 데이터에 대한 처리 필요

# 데이터 정제

## 1 누락 데이터 처리

### 1 데이터 삭제

#### 1 Isnull 함수

- 데이터가 NaN이면 True, 아니면 False를 반환함
- Numpy 연산은 스칼라 연산이 아닌 벡터 연산을 함

```
import pandas as pd
import numpy as np

#NaN은 Numpy를 사용해 직접 입력 가능
s = pd.Series([1,2,3,4,np.nan])
print( s.isnull() )
```

```
0    False
1    False
2    False
3    False
4     True
```

# 데이터 정제

## 1 누락 데이터 처리

### 1 데이터 삭제

#### 2 Sum 함수

- 인자들의 합계를 반환함
- Isnull 함수 사용 후 결과값에 Sum 함수 사용
  ➡ True 개수 반환

```
import pandas as pd
import numpy as np

#NaN은 Numpy를 사용해 직접 입력 가능
s = pd.Series([1,2,3,4,np.nan])
print( s.isnull() )
print( "NaN 값 개수 ",  s.isnull().sum() )
```

```
NaN 값 개수  1
```

# 데이터 정제

## 1 누락 데이터 처리

### 1 데이터 삭제

#### 3 Dropna 함수

DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)[source]

subset에서 지정한 필드에 NaN 값이 있는 행 삭제 → inplace=True로 지정하면 자신의 데이터가 수정됨 → 필드 지정 방법은 list 형태

subset=['필드1', '필드2']

| | |
|---|---|
| thresh | ▪ int 값이 필드에서 부여한 개수만큼 NaN이 있는 행·열 삭제 |
| how | ▪ any, all any : 행이나 컬럼값 중에 하나라도 있으면 행이나 열 삭제<br>▪ All : 행이나 컬럼값에 모두 NaN이 있을 때 행·컬럼 삭제 |
| axis | ▪ 0은 행, 1은 열을 지칭<br>▪ 삭제할 방향을 지시 |

# 데이터 정제

## 1 누락 데이터 처리

### 1 데이터 삭제

#### 4 reset_index 함수

- DataFrame 인덱스를 재지정
- 새로 필드를 추가하도록 되어 있음
  ➡ drop 옵션을 True로 지정

```
import pandas as pd
import numpy as np

#NaN은 Numpy를 사용해 직접 입력 가능
s = pd.Series([1,2,3,4,np.nan])

data = s.dropna(how='any', axis=0)
print(data)

data = data.reset_index(drop=True)
```

# 데이터 정제

## 1 누락 데이터 처리

### 1 데이터 삭제

#### 5 예제

```
#파일명 : exam11_1.py

import pandas as pd

data = pd.read_csv("./data/data.csv")

print("컬럼명 : ", data.columns)
print("인덱스 : ", data.index)

print( data.info() )

#누락된 데이터가 있는지 확인
print( data['height'].value_counts(dropna=False))
print( data['height'].isnull())   #NaN이면 True, 아니면 False 반환
print( data['height'].notnull())

#Null값인 것에 대한 개수 확인
print( data['height'].isnull().sum(axis=0))

#NaN값을 thresh개 가진 열 모두 삭제
data_thresh = data.dropna(axis=1, thresh=3)
```

(계속)

# 데이터 정제

## 1 누락 데이터 처리

### 1 데이터 삭제

#### 5 예제

```
#합계
print ( data['height'].sum() )

print("데이터 개수")
print(data.shape)
data = data.dropna(subset=['height'], how='any', axis=0)
print("삭제 후 데이터 개수")
print(data.shape)

data = data.dropna(subset=['weight'], how='any', axis=0)
print("삭제 후 데이터 개수")
print(data.shape)

#인덱스 다시 부여
data = data.reset_index(drop=True)
print(data)
```

# 데이터 정제

## 1 누락 데이터 처리

### 2 누락 데이터 치환

- 누락 데이터 삭제
  ➡ 실제 데이터 수집 시 누락이 많아 행·열을 삭제할 경우 제대로 분석이 어려운 경우 발생

**문제 해결 방법**

- 행이나 컬럼 전체를 삭제하는 것보다 대체값을 바꾸는 게 좋음
- 대체값은 보통 평균값 또는 중간값을 사용
- 최댓값과 최솟값 편차가 클 경우에는 평균값보다는 중간값이 집단을 대표하는 경우가 더 많음

# 데이터 정제

## 1 누락 데이터 처리

### 2 누락 데이터 치환

```
#파일명 : exam11_2.py
#누락 데이터 치환

import pandas as pd

data = pd.read_csv("./data/data.csv")

#누락된 데이터가 있는지 확인
print( data['height'].value_counts(dropna=False))
print( data['height'].isnull()) #NaN이면 True, 아니면 False 반환
print( data['height'].notnull())

#null값인 것에 대한 개수 확인
print("height 필드 Nan 개
수 : ", data['height'].isnull().sum(axis=0))
print("weight 필드 Nan 개
수 : ", data['weight'].isnull().sum(axis=0))


print("누락값 대체 후 ----------------")
#누락값 대체
mean_height = data['height'].mean()
mean_weight = data['weight'].mean()
```

# 데이터 정제

## 1 누락 데이터 처리

### 2 누락 데이터 치환

- fillna 함수 NaN 값을 특정 값으로 대체
- inplace 옵션은 True를 입력해야 값이 바뀜

```
data['height'].fillna( mean_height, inplace=True)
data['weight'].fillna( mean_weight, inplace=True)

print("height 필드 Nan 개수 : ",
data['height'].isnull().sum(axis=0))
print("weight 필드 Nan 개수 : ",
data['weight'].isnull().sum(axis=0))

print( data )
```

```
weight 필드 Nan 개수 :  2
누락값 대체 후 -----------------
height 필드 Nan 개수 :  0
weight 필드 Nan 개수 :  0
   name      height      weight
0    A1  180.000000   92.000000
1    A2  176.000000   70.000000
2    A3  175.000000   65.000000
3    A4  172.000000   64.000000
4    A5  168.000000   67.107143
5    A6  175.000000   74.000000
6    A7  177.000000   68.000000
7    A8  172.000000   65.000000
8    A9  177.888889   77.000000
9   A10  173.000000   59.000000
10  A11  174.000000   66.000000
11  A12  177.000000   63.000000
12  A13  179.000000   65.000000
```

```
13  A14  181.000000   72.000000
14  A15  189.000000   79.000000
15  A16  193.000000   84.000000
16  A17  177.888889   34.000000
17  A18  172.000000  160.000000
18  A19  179.000000   84.000000
19  A20  169.000000   64.000000
20  A21  167.000000   56.000000
21  A22  230.000000   12.000000
22  A23  171.000000   15.000000
23  A24  177.888889   80.000000
24  A25  187.000000   78.000000
25  A26  176.000000   67.107143
26  A27  175.000000   59.000000
27  A28  173.000000   62.000000
28  A29  169.000000   53.000000
29  A30  174.000000   59.000000
```

# 데이터 정제

## 2 중복 데이터 처리

- Pandas 라이브러리는 데이터 수집 중 중복된 데이터가 있을 경우 쉽게 중복을 제거할 수 있는 함수를 제공

```
Dataframe객체['필드명'].duplicated()
```

- 중복되지 않았을 경우 False, 중복될 경우 True를 반환
- 두 개 이상의 데이터가 중복되어 있을 때 첫 번째 데이터는 False, 그 뒤에 중복되어 나타나는 데이터 항목은 True로 출력됨

```
Dataframe객체.drop_duplicates()
```

- 중복된 필드가 있는 모든 행이 삭제됨

```
Dataframe객체.drop_duplicates(subset=[필드명])
```

- subset 필드를 별도로 지정하면 그 필드 내의 데이터가 중복될 경우 모든 행이 삭제됨

# 데이터 정제

## 2 중복 데이터 처리

```
#파일명 : exam11_3.py
#중복 데이터 제거

import pandas as pd

data = {
    'passenger_code':['A101', 'A102', 'A103', 'A101', 'A104', '
                                A101', 'A103'],
    'target':['광주', '서울', '부산', '광주', '대구', '광주', '부산'],
    'price':[25000, 27000, 45000, 25000, 35000, 27000,
             45000]
}

df = pd.DataFrame(data)

print( df )

print("중복된 데이터 ")
col = df['passenger_code'].duplicated()  #중복된 데이터 확인
print( col )

#중복 제거 시 모든 데이터가 일치된 것만 제거
df2 = df.drop_duplicates()
print( df2 )
```

(계속)

# 데이터 정제

## 2 중복 데이터 처리

```
print("특정 컬럼값이 중복일 때 제거하기 -----")
df3 = df.drop_duplicates(subset=['passenger_code'])
print( df3 )


print("두 개의 컬럼값이 중복일 때 제거하기 -----")
df3 = df.drop_duplicates(subset=['passenger_code',
'target'])
print( df3 )
```

```
중복된 데이터
0    False
1    False
2    False
3     True
4    False
5     True
6     True
Name: passenger_code, dtype: bool
  passenger_code target  price
0           A101     광주  25000
1           A102     서울  27000
2           A103     부산  45000
4           A104     대구  35000
5           A101     광주  27000
특정 컬럼값이 중복일때 제거하기 -----
  passenger_code target  price
0           A101     광주  25000
1           A102     서울  27000
2           A103     부산  45000
4           A104     대구  35000
두개의 컬럼값이 중복일때 제거하기 -----
  passenger_code target  price
0           A101     광주  25000
1           A102     서울  27000
2           A103     부산  45000
4           A104     대구  35000
```

# 데이터 변환

## 1 정규화 및 단위 전환

머신러닝, 딥러닝에서 데이터의 단위가 너무 다를 경우
전체 예측 평가에 영향을 미칠 수 있음

일부 알고리즘에서는 데이터값을 평균 0, 분산 1이 되도록
만들어줘야 함

**정규화**

### 정규화 수식

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

(정규화하고자 하는 값 - 데이터값 중 최솟값)
(데이터값 중 최댓값 - 데이터값 중 최솟값)

| 사이킷런 (Scikit-learn) 라이브러리 | ▪ 정규화를 지원하는 클래스도 있음 |
|---|---|

# 데이터 변환

## 1 정규화 및 단위 전환

- 우리나라에서 사용하는 단위와 유럽이나 미국 등 기타 국가들이 사용하는 단위가 달라 데이터를 보기 어려운 경우가 있음

**무게의 단위**

| 〈우리나라〉 | 〈일부 국가〉 |
| --- | --- |
| 킬로그램 (kg) | 파운드 (lb) |

- 단위를 일치시키는 것이 데이터를 분석할 때 편리

# 데이터 변환

## 1 정규화 및 단위 전환

```
#파일명 : exam11_4.py
#데이터 표준화

import pandas as pd

data = pd.read_csv('./data/auto-mpg.csv')
print(data.info())
print(data.head())

#컬럼명 변경
data.columns=['mpg', 'cyl', 'disp', 'power', 'weight', 'acce', 'model']
print(data.head())
```

**정규화**

- 데이터 분석에서 아주 중요한 작업
- 정규화 작업을 해줘야만 하는 머신러닝 알고리즘이 많음

```
#(정규화하고자 하는 값 - 데이터값 중 최솟값) /
  (데이터값 중 최댓값 - 데이터값 중 최솟값)
data['mpg2'] = (data['mpg'] -
data['mpg'].min())/(data['mpg'].max()-data['mpg'].min())
print(data)

#단위 환산 : 한국 단위로 환산
mpg_unit = 1.60934 / 3.78541

data['kpl'] = (data['mpg'] * mpg_unit).round(2)
print( data.head() )
```

# 데이터 변환

## 2 타입 전환

**숫자 형태의 데이터**

- 실제 데이터가 엑셀에서 문자·숫자 등 어떤 형태로 저장되든 숫자로 잘 읽어와 크게 문제가 되지 않음
- 엑셀의 경우, 숫자를 문자열 형태 또는 숫자 형태로 저장

수치 자료가 와야 할 열에 수치 자료가 아닌 다른 문자가 있을 때 문제가 발생하므로 처리 필요

**문제 해결 방법**

- 수치 자료가 와야 할 열에 들어간 문자 제거하기
- 문자가 들어 있는 행의 데이터 버리기
- 유형 전환하기

# 데이터 변환

## 2 타입 전환

**문자 형태의 데이터**

- 실제 데이터가 엑셀에서 문자·숫자 등 어떤 형태로 저장되든 숫자로 잘 읽어와 크게 문제가 되지 않음
- 엑셀의 경우, 숫자를 문자열 형태 또는 숫자 형태로 저장

문자열 → 범주형 자료 타입

.astype 함수를 이용해 타입 전환 필요

# 데이터 변환

## 2 타입 전환

```
#파일명 : exam11_5.py
#데이터 표준화

import pandas as pd
import numpy as np

data = pd.read_csv('./data/auto-mpg.csv')
print(data.info())
print(data.head())
```

#타입이 맞지 않을 경우 전환하여 사용

- 현재 사용하는 파이썬 버전은 문자열 데이터라도 숫자 형태면 자동으로 수치 자료로 처리
- 파이썬 버전에 따라 다르게 동작할 수 있음

```
data.columns=['mpg', 'cyl', 'disp', 'power', 'weight', 'acce', 'model']
print(data.dtypes)
print(data.head())

print( data['disp'].unique())
```

(계속)

# 데이터 변환

## 2 타입 전환

```
#잘못된 데이터를 NaN으로 먼저 바꿈
data['disp'].replace('?', np.nan, inplace=True)
print(data.head())

data.dropna(subset=['disp'], axis=0, inplace=True)
print(data.head())
print(data.dtypes)
data['disp'] = data['disp'].astype('float')
print(data.dtypes)

#범주형으로 바꿈
data['model'] = data['model'].astype('category')
print(data.dtypes)
```

- 데이터를 범주형으로 바꿀 때 astype('category') 사용
- 문자열의 형태로 입력된 데이터는 분석 시 반드시 범주형으로 바꾸어 처리

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 1 구간 나누기

- 경우에 따라 **연속 데이터를 불연속 데이터로 전환**해야 할 때가 있음

**연비**

- 국가에서 특정 조건을 만족할 경우 구간을 나누어 등급 부여
- 숫자 형태의 자료보다 등급 형태가 더 빠른 인식 가능
- 등급별 모델의 평가·분석을 해야 할 경우

- 구간 나누기를 통해 연속 데이터를 불연속 데이터로 전환 가능

- 불연속 데이터로 전환 후 실제 분석 시 특별한 방법을 사용하여 데이터 저장

- 연비를 등급으로 A, B, C, D 4개로 나누었다면, 머신러닝이나 딥러닝 분석에서 이를 그대로 사용하지 못함
- 머신러닝이나 딥러닝은 수학이므로 **원핫인코딩** 형태로 전환해야 사용 가능

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

- 범주형 데이터를 행렬의 형태로 전환
- 각각의 범주가 갖는 값을 다음과 같이 표현되도록 저장

| A | B | C | D |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

4개의 그룹이 있다고 가정
그룹의 형태를 A, B, C, D등급을
필드의 컬럼으로 만듦

해당하지 않는 요소의 위치

해당하는 요소의 위치

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

```
#파일명 : exam11_4.py
#구간 분할

import pandas as pd
import numpy as np

data = pd.read_csv('./data/auto-mpg.csv')
print(data.info())
print(data.head())
```

#타입이 맞지 않을 경우 전환하여 사용

- 현재 사용하는 파이썬 버전은 문자열 데이터라도 숫자 형태면 자동으로 수치 자료로 처리
- 파이썬 버전에 따라 다르게 동작할 수 있음

```
data.columns=['mpg', 'cyl', 'disp', 'power', 'weight', 'acce', 'model']
print(data.dtypes)
```

(계속)

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

```
#잘못된 데이터를 NaN으로 먼저 바꿈
data['disp'].replace('?', np.nan, inplace=True)
data.dropna(subset=['disp'], axis=0, inplace=True)
data['disp'] = data['disp'].astype('float')
```

(계속)

**잘못된 데이터가 있는 경우**

- NaN값으로 먼저 전환
- NaN을 포함하고 있는 행이나 열을 삭제하는 게 빠름

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

```
#범주형으로 바꿈
data['model'] = data['model'].astype('category')

#연속적인 값을 구간으로 나누어 비연속적인 값(범주형)으로 나눔
#print(data['power'])

print(data['power'].isnull().sum(axis=0))
data.dropna(subset=['power'], inplace=True)
count, bin_dividers = np.histogram(data['power'], bins=4)
print( bin_dividers)
```

(계속)

| | |
|---|---|
| np.histogram | ▪ 구간을 나누는 함수<br>▪ 데이터의 범위를 알아내서 지정된 구간을 나누고 구간의 경계에 대한 값을 가져옴<br>▪ 구간을 나누고자 하는 필드와 구간의 개수를 전달하면 구간의 경계, 구간의 데이터 개수를 반환 |

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

```
bin_names = ["D", "C", "B", "A"]
data["grade"] = pd.cut( x=data['power'], bins=bin_dividers,
    labels = bin_names, include_lowest=True)

print( data )
```

(계속)

- cut 함수, x인자에 나누고자 하는 구간의 필드를 정해줌
- bins 인자에 np.histogram이 생성한 구간 정보를 전달
- labels에 각 구간에 붙일 라벨명을 지정
- include_lowest=True는 경계값을 포함하라는 의미

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

- 실행 결과

| | mpg | cyl | disp | power | weight | acce | model | grade |
|---|---|---|---|---|---|---|---|---|
| 1 | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | B |
| 2 | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | B |
| 3 | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | B |
| 4 | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | B |
| 5 | 15.0 | 8 | 429.0 | 198.0 | 4341 | 10.0 | 70 | A |
| .. | ... | ... | ... | ... | ... | ... | ... | ... |
| 393 | 27.0 | 4 | 140.0 | 86.0 | 2790 | 15.6 | 82 | D |
| 394 | 44.0 | 4 | 97.0 | 52.0 | 2130 | 24.6 | 82 | D |
| 395 | 32.0 | 4 | 135.0 | 84.0 | 2295 | 11.6 | 82 | D |
| 396 | 28.0 | 4 | 120.0 | 79.0 | 2625 | 18.6 | 82 | D |
| 397 | 31.0 | 4 | 119.0 | 82.0 | 2720 | 19.4 | 82 | D |

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

```
Y_class = np.array(data['grade']).reshape(-1,1)

from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
enc.fit(Y_class) #분류에 사용되는 클래스 식별, 메타데이터에 기록

Y_class_onehot = enc.transform(Y_class).toarray()
Y_class_recovery = np.argmax(Y_class_onehot, axis=1).
reshape(-1,1)

print(Y_class_onehot)
```

# 데이터 변환

## 3 구간 나누기와 원핫인코딩

### 2 원핫인코딩(One-hot Encording)

- 데이터 필드가 1차원인 상태로는 원핫인코딩이 진행되지 않음

| reshape 함수 | ▪ 배열의 구조 변환 |
|---|---|

1차원 Series 타입 → 2차원 Ndarray 타입

[ ['A']['B'] ...]

reshape 함수를 이용한 차원 변경 → sklearn.preprocessing의 원핫인코더 객체 형성 → transform 함수 수행

| np.argmax 함수 | ▪ 원핫인코딩 복원 시 사용<br>▪ 전달받은 함수 중 가장 큰 값을 갖는 인자를 반환하는 함수 |
|---|---|

# 학습정리

## 1. 데이터 정제

- isnull 함수 등을 이용해 데이터의 누락치가 있는지 먼저 체크함
- isnull 함수는 True, False 값만 반환하므로 전체적인 누락 여부 확인이 어려움
- isnull() 결과로 sum() 함수를 호출하면 NaN 데이터가 존재하는지를 쉽게 알 수 있음
- 누락된 데이터의 처리
  - 누락 데이터 건수가 적어서 전체 분석에 영향을 미치지 않을 거라고 판단될 경우 삭제 진행
  - 누락 데이터 건수가 많아서 전체 영향을 미친 거라고 판단되면 평균값이나 등으로 대체해서 사용

# 학습정리

## 2. 데이터 변환

- 분석 시 사용하는 데이터
  - 연속형 : 평균값을 중요하게 여김
  - 불연속형(비연속형, 범주형) : 개수를 중요하게 여김
- 연속형이 아닌 데이터 타입들은 불연속형으로 다뤄야 하기 때문에 문자열형은 불연속형으로 전환해서 사용
- 연속형 데이터라도 불연속형으로 전환해서 분석해야 할 경우 구간 나누기를 사용
- 구간 나누기 후 실제 분석에서는 데이터를 원핫인코딩 방식으로 전환하여 처리