중복 데이터

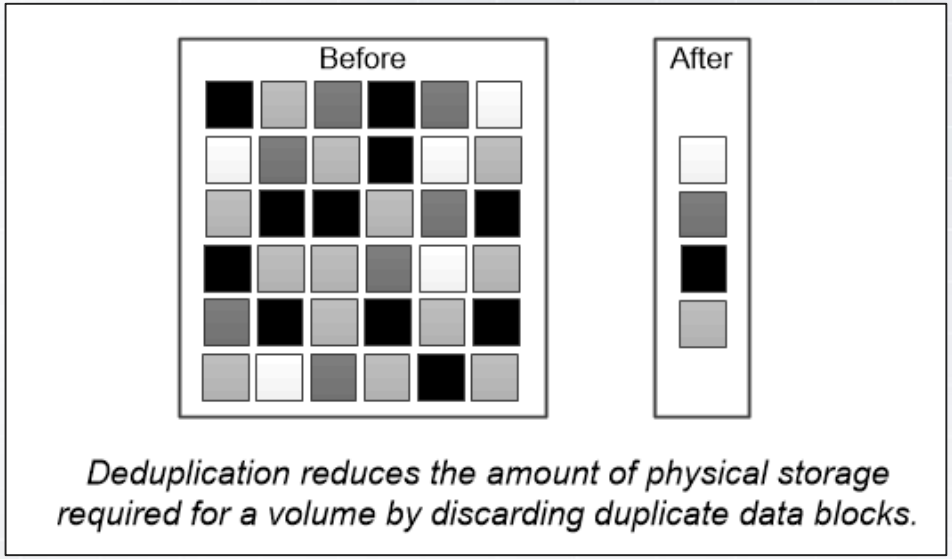
중복 데이터란?

- 데이터셋 내에서 동일한 정보나 기록이 두 번 이상 반복되는 데이터를 의미한다.
- 다양한 원인으로 발생할 수 있다. 예를 들어 데이터 입력 오류, 데이터 병합 과정에서의 문제, 시스템적 오류 등에 의해 생긴다.
- 실생활에서의 중복데이터를 아래 표와 같은 관점으로 볼 수 있으며, 중복데이터를 제거함으로써 이러한 문제를 예방하거나 해결할 수 있다.

분야	실생활 문제
병원	진료 예약 중복 신청으로 병원 스케줄 혼선
온라인 쇼핑	시스템 오류로 중복 주문되어 금액 이중 청구
투표	재투표로 인한 투표 결과 왜곡
회원가입	중복 이메일 계정 생성
식당 예약	동일한 시간대 같은 테이블 예약으로 문제 발생

중복 데이터란?

- 시스템적 관점으로 보면, 중복된 데이터로 인해 필요한 저장 공간을 차지할 수 있다.
- 실제 필요한 데이터만을 가지고 있으면 공간 절약과 효율적인 처리/관리가 가능하다.



Deduplication reduces the amount of physical storage required for a volume by discarding duplicate data blocks.

https://docs.netapp.com/

중복 데이터의 처리

- 인공지능 시스템의 정확도는 분석 데이터의 품질에 크게 의존한다.
- 중복 데이터는 품질과 신뢰성에 영향을 미치며, 결과의 정확성을 왜곡시킬 수 있다.
- 데이터 전처리 과정에서 이를 탐지하고 제거하는 것이 중요하다.

중복 데이터의 확인

- 동일한 관측값이 중복되었는지 여부를 확인하기 위해 예시로 아래 DataFrame을 생성한다.
- duplicated() 를 이용하면 전에 나온 행들과 비교를 하여 bool형 데이터를 반환한다.
- 중복 행의 경우 Ture를 반환하며, 처음 나오는 행에 대해서는 False를 반환한다.

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

중복 데이터의 확인

- df = pd.DataFrame({'c1':['a','a','b','a','b'], 'c2':[1,1,1,2,2], 'c3':[1,1,2,2,2]})
- 딕셔너리 형태로 DataFrame을 생성했으며, 키는 DataFrame의 열 이름이 되고, 값은 해당 열의 데이터가 된다.
- 이 예제에서의 DataFrame 은 'c1' 열은 문자열 데이터, 'c2'와 'c3' 열은 정수 데이터를 포함한 서로 다른 데이터 타입을 가지고 있다.

```
In [1]: import pandas as pd

In [2]: df = pd.DataFrame({'c1':['a','a','b','a','b'],
                           'c2':[1,1,1,2,2],
                           'c3':[1,1,2,2,2]})
```

중복 데이터의 확인

- df2는 duplicated()을 사용하여 df 데이터의 중복을 확인한다.
- 각 행의 모든 열 값이 이전에 나온 행과 동일하다면, 그 행은 중복으로 간주된다.

```
In [5]: df2 = df.duplicated()

In [6]: df2

Out[6]: 0    False
        1     True
        2    False
        3    False
        4    False
        dtype: bool
```

중복 데이터의 확인

- 이 예제의 경우에는 1번 행만 중복된다.
  - 0번 행은 이전 행이 없으므로 중복될 수 없다. 따라서 False이다.
  - 1번 행의 모든 열 값 (a, 1, 1)은 0번 행의 값과 동일하므로 중복으로 간주된다. 따라서 True이다.
  - 2,3,4번 행은 이전 행들과 모두 다른 값을 가지고 있어 False이다.

```
In [5]: df2 = df.duplicated()

In [6]: df2

Out[6]: 0    False
        1     True
        2    False
        3    False
        4    False
        dtype: bool
```

중복

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

중복 데이터의 확인

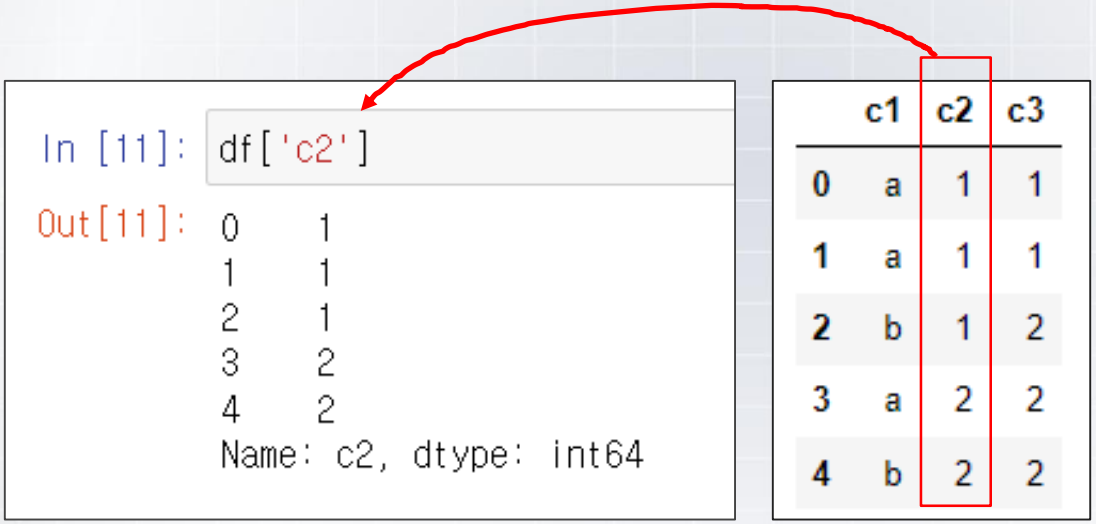
- keep : 중복 데이터를 식별할 때 어떤 데이터를 유지할 것인지를 지정하는 데 사용된다.
- keep = 'first' : 기본값으로, 처음의 값을 기준으로 한다.
- keep = 'last' : 마지막 값을 기준으로 한다.

```
In [5]: df2 = df.duplicated()

In [6]: df2

Out[6]: 0    False
        1     True
        2    False
        3    False
        4    False
        dtype: bool
```

```
In [31]: df2 = df.duplicated(keep='last')

In [32]: df2

Out[32]: 0     True
         1    False
         2    False
         3    False
         4    False
         dtype: bool
```

중복 데이터의 확인

- 특정 열에 대해서만 중복 데이터를 찾기 위해서는 DataFrame에서 'c2' 열을 선택한다.
- df['c2']를 통해 'c2' 열에 해당하는 데이터를 가져올 수 있다.



중복 데이터의 확인

① df에서 'c2'열 만을 추출해 새로운 변수를 만들어 저장한 후 사용하는 방법

② duplicated()를 바로 사용하는 방법

※ 원본 변수에서 바로 변경하려면 unplace=True 옵션을 추가해 사용한다.

①

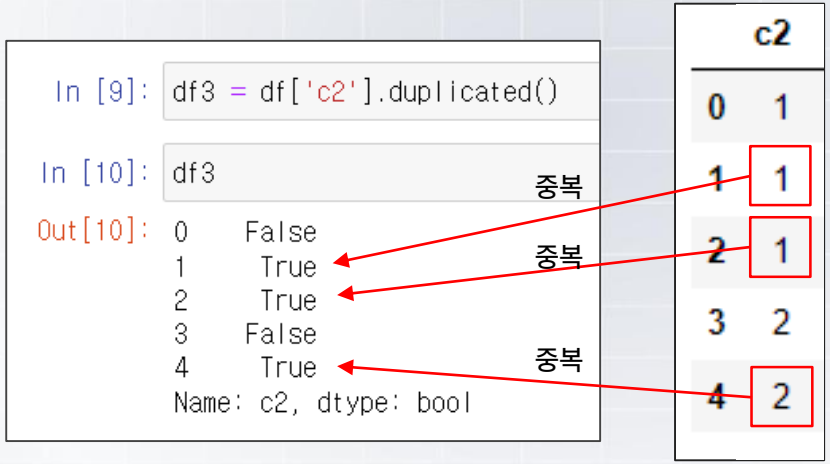
```
In [9]:  df3 = df['c2'].duplicated()

In [10]: df3

Out[10]: 0    False
         1     True
         2     True
         3    False
         4     True
         Name: c2, dtype: bool
```

②

```
In [12]: df3 = df['c2']

In [13]: df3_1 = df3.duplicated()

In [14]: df3_1

Out[14]: 0    False
         1     True
         2     True
         3    False
         4     True
         Name: c2, dtype: bool
```

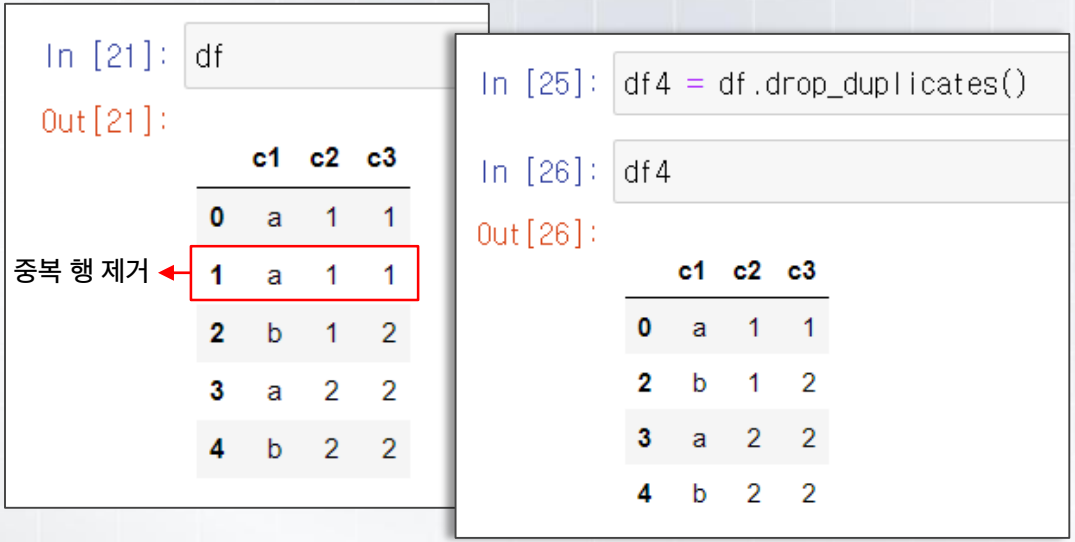
중복 데이터의 확인

- 이전에 등장했던 값 1과 중복되어 1행과 2행의 결과 : True
- 이전에 등장했던 값 2와 중복되어 4행의 결과 : True



중복 데이터의 제거

- 중복데이터 확인 후 제거하는 명령으로 drop_duplicates()를 사용할 수 있다.
- 실행 결과, 중복되는 행을 제거하고 고유한 값을 가진 행들만 남긴다.
- 1행의 데이터는 앞에 이웃하고 있는 0행의 데이터와 중복되므로 제거된다.



중복 데이터의 제거

- drop_duplicates()에는 subset이라는 옵션을 사용해 열 이름의 리스트를 전달할 수 있다.
- 중복 데이터를 탐색할 열들을 따로 지정하는 데 사용된다

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

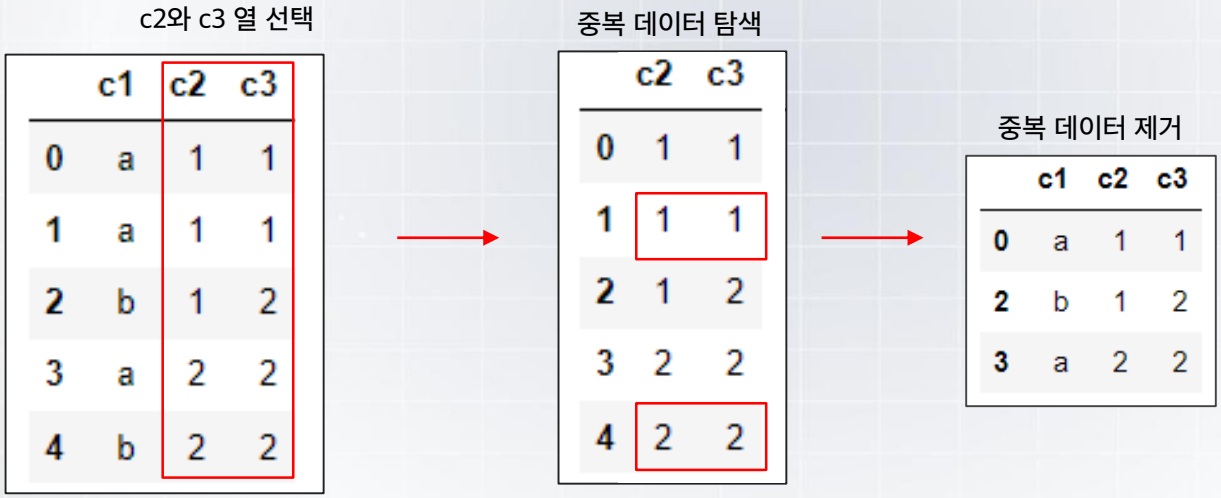
```
In [28]: df5 = df.drop_duplicates(subset=['c2','c3'])

In [29]: df5
```

Out[29]:

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2

중복 데이터의 제거

- ‘c2’, ‘c3’ 열을 기준으로 중복 데이터를 탐색한다.
- 처음 나타난 데이터(0행, 2행, 3행)를 제외한 중복 되는 데이터(1행, 4행)를 제거한다.



	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

	c2	c3
0	1	1
1	1	1
2	1	2
3	2	2
4	2	2

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2

데이터 표준화

데이터의 표준화

- 표준화(standardization)는 데이터의 값이 서로 다른 범위에 있을 때, 이를 일정한 범위로 조정하는 과정을 의미한다.
- 이는 다양한 데이터를 비교 분석하기 쉽게 만들어주며, 특히 인공지능 시스템에서 중요한 전처리 단계 중 하나이다.
- 실생활에서의 표준화 예로 환율을 들 수 있다. 국가간 물가 수준과 경제 지표를 고려한 비율 적용한 것.

데이터의 표준화

- 실무의 데이터셋은 다양한 출처와 사람들로부터 온다.
- 이로 인해 데이터에는 단위 차이, 대소문자 구분, 약칭 사용 등 다양한 형태의 표현이 섞여 있다.
- 처음에는 깔끔해 보여도 자세히 살펴보면 같은 정보가 다른 형식으로 표현되거나 다른 단위가 섞여 있을 수 있다.
- 이런 일관성 없는 데이터 표현 역시 분석의 정확도를 떨어뜨리기에 데이터 포맷을 표준화하는 작업 역시 중요한 과정이다.
- 그 중 데이터의 표준화를 위해 단위환산과 자료형 변환을 진행할 것

데이터의 표준화 ① 단위 환산

- 데이터셋 내에서 다양한 측정 단위를 사용하면 데이터의 일관성이 떨어지며, 분석의 정확도에 문제가 발생한다.
- 외국 데이터셋을 활용할 때 단위 문제가 발생한다. 영미권의 단위와 국내에서 사용하는 단위와의 차이가 존재하기 때문.
- 따라서 영미권 단위인 마일, 야드, 온스 등을 국내에서 일반적으로 사용하는 미터, 평, 그램 등으로 통일하는 작업이 필요하다.

데이터의 표준화 ① 단위 환산

• 데이터셋에서 'mpg'열은 영미권에서 사용하는 갤런당 마일(mile per gallon) 단위로 연비를 표시하고 있다.

• 한국에서는 익숙한 표기법인 리터당 킬로미터(km/L) 단위로 변환해보자

1 Mile = 1.609344 km

1 Gallon = 3.78541 L

1 mpg = km ÷ L

▶ 1 mpg의 연비를 km/L로 변환하면, 1.609344 km / 3.78541 L ≈ 0.425 km/L

데이터의 표준화 ① 단위 환산

- mpg (Miles Per Gallon) 값을 KmL(Kilometers Per Liter, Km/L) 값으로 변환하는 코드
- mpg 값을 기반으로 KmL 값을 계산하고 출력한다.
  - mpg : 마일 당 갤런 값을 설정한다.
  - Km : 1 마일이 몇 킬로미터인지 설정한다.
  - L : 1 갤런이 몇 리터인지를 설정한다.

MPG → Km/L 변환

In [22]:
```
mpg = 1
km = 1.60934
L =  3.78541

kml = mpg * (km / L)

print(kml)
```

```
0.42514285110463595
```

데이터의 표준화 ① 단위 환산

•UCI 자동차 연비 데이터셋을 사용하기 위해 파일을 다운받는다. auto-mpg(2).csv(zip) - 열 이름 추가 버전

- https://l22j.github.io/AI-system/download/download.html

	A	B	C	D	E	F	G	H	I	J	K
1	mpg	cylinders	displacem	horsepow	weight	acceleratic	model yea	origin	name		
2	18	8	307	130	3504	12	70	1	chevrolet chevelle malibu		
3	15	8	350	165	3693	11.5	70	1	buick skylark 320		
4	18	8	318	150	3436	11	70	1	plymouth satellite		
5	16	8	304	150	3433	12	70	1	amc rebel sst		
6	17	8	302	140	3449	10.5	70	1	ford torino		
7	15	8	429	198	4341	10	70	1	ford galaxie 500		
8	14	8	454	220	4354	9	70	1	chevrolet impala		
9	14	8	440	215	4312	8.5	70	1	plymouth fury iii		
10	14	8	455	225	4425	10	70	1	pontiac catalina		
11	15	8	390	190	3850	8.5	70	1	amc ambassador dpl		
12	15	8	383	170	3563	10	70	1	dodge challenger se		
13	14	8	340	160	3609	8	70	1	plymouth 'cuda 340		
14	15	8	400	150	3761	9.5	70	1	chevrolet monte carlo		
15	14	8	455	225	3086	10	70	1	buick estate wagon (sw)		
16	24	4	113	95	2372	15	70	3	toyota corona mark ii		
17	22	6	198	95	2833	15.5	70	1	plymouth duster		

데이터의 표준화 ① 단위 환산

• pd.read_csv를 사용하여 파일을 불러온 후 확인

```
In [1]: import pandas as pd
        df = pd.read_csv(r'C:₩Users₩ADMIN₩Desktop/auto-mpg(2).csv')

In [2]: df
```

Out[2]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

398 rows × 9 columns

데이터의 표준화 ① 단위 환산

•0행에 들어있는 차량의 연비는 18mpg이다.

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite

•1mpg = Km ÷ L

•한국식 연비 표현(Km/L)로 변환하면 18mpg는 7.65 Km/L 이다.

```
In [6]: mpg = 18
        km = 1.60934
        L =  3.78541

        KmL = mpg * (km / L)

        print(KmL)

        7.652571319883447
```

데이터의 표준화 ① 단위 환산

- mpg_to_kmL 은 mpg 값을 Km/L 값으로 변환하기 위한 값인 0.425가 들어가 있다.
- df에 'kmL' 라는 새로운 열을 만들어 'mpg ' 값에 0.425를 곱해준 변환 값을 추가한다.

```
In [3]: km = 1.60934
        L =  3.78541

        mpg_to_kmL = km / L

In [4]: df['kmL'] = df['mpg'] * mpg_to_kmL
```

데이터의 표준화 ① 단위 환산

•df를 출력하면 변환된 값인 'KmL'이 추가되어 있음을 확인할 수 있다.

```
In [7]: df
```

Out [7]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name	kmL
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	7.652571
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	6.377143
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	7.652571
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.802286
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	7.227428
...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl	11.478857
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup	18.706285
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage	13.604571
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger	11.904000
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10	13.179428

398 rows × 10 columns

데이터의 표준화 ① 단위 환산

•표현의 간편함을 위해 round() 함수를 사용하여 수치를 반올림해준다.

```
In [8]: df['kmL'] = df['kmL'].round(2)
```

• df의 'kmL' 열에 있는 각 값들을 소수점 둘째 자리까지 반올림하고, 반올림된 결과를 원래의 'kmL' 열에 다시 저장한다.

※ 예를들어 3.141592 같은 값에 round(2)를 실행한다면, 해당 값은 3.14로 바뀐다.

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name	kmL
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	7.65
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	6.38
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	7.65
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	6.80
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	7.23
...	...	...	...	...	...	...	...	...	...	

데이터의 표준화 ① 단위 환산

- 반올림 함수인 round()에 대해 조금 더 알아보자
- pi 라는 변수를 만들어서 원주율 3.141592.. 을 넣어준 후, round(3)를 실행한다면, 해당 값은 지정한 자릿수까지 반올림된다.
- 자리수가 긴 값들, 혹은 자릿수가 모두 다른 값들을 round()함수를 사용하여 일관성 있게 표현을 간소화 시켜줄 수 있다.

```
In [63]: pi = 3.141592

In [64]: pi_r = round(pi,3)

In [65]: print(pi_r)
3.142
```

데이터의 표준화 ② 자료형 변환

- 데이터셋에는 때때로 숫자 값이 문자열로 저장되어 있는 경우가 있다.
- 이럴 때는 해당 데이터의 자료형을 숫자형으로 변환해주는 과정이 필요하다.
- 데이터프레임 내 각 열의 자료형을 확인하기 위해 dtypes 속성을 활용할 수 있으며, info() 함수를 통해서도 같은 정보를 얻을 수 있다.
- UCI 자동차 연비 데이터셋에서는 엔진 출력을 나타내는 'horsepower'열의 경우 숫자형이 적절하고, 출시연도를 나타내는 'model year' 열과 출시국가를 뜻하는 'origin' 열 같은 경우에는 카데고리를 나타내기 때문에 범주형으로 표현하는 것이 적절하다.
- 따라서 데이터셋의 데이터 형식을 확인하고 각각 적절한 자료형으로 변환해준다.

데이터의 표준화 ② 자료형 변환

- 데이터의 정보를 확인하기 위해 두 가지 방법을 사용할 수 있다.
- 간략한 자료형 정보만 필요하다면 dtypes를, 더 상세한 전반적인 정보를 원한다면 info()를 사용하는 것이 좋다.

```
In [13]: df.dtypes

Out[13]: mpg             float64
         cylinders         int64
         displacement    float64
         horsepower       object
         weight            int64
         acceleration    float64
         model year        int64
         origin            int64
         name             object
         kmL             float64
         dtype: object
```

```
In [14]: df.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 398 entries, 0 to 397
         Data columns (total 10 columns):
          #   Column        Non-Null Count  Dtype
         ---  ------        --------------  -----
          0   mpg           398 non-null    float64
          1   cylinders     398 non-null    int64
          2   displacement  398 non-null    float64
          3   horsepower    398 non-null    object
          4   weight        398 non-null    int64
          5   acceleration  398 non-null    float64
          6   model year    398 non-null    int64
          7   origin        398 non-null    int64
          8   name          398 non-null    object
          9   kmL           398 non-null    float64
         dtypes: float64(4), int64(4), object(2)
         memory usage: 31.2+ KB
```

데이터의 표준화 ② 자료형 변환

- 'horsepower'의 자료형을 확인해보면 문자열을 뜻하는 object 자료형으로 표시된다.
- 엔진 출력의 크기를 나타내는 데이터 만큼 숫자형으로 변환해주는 것이 적절하다.

```
In [13]: df.dtypes

Out[13]: mpg             float64
         cylinders         int64
         displacement    float64
         horsepower       object
         weight            int64
         acceleration    float64
         model year        int64
         origin            int64
         name             object
         kmL             float64
         dtype: object
```

데이터의 표준화 ② 자료형 변환

- 데이터 처리나 분석 작업을 할 때, 특정 변수나 열의 자료형이 예상과 다르게 저장되어 있을 경우가 있다.
- 'horsepower'와 같은 데이터가 문자열로 저장되어 있다면, 그 이유를 파악하는 것이 중요하다.
- 실제 값들을 살펴보면 왜 문자열로 저장되어 있는지, 그리고 어떻게 처리해야 하는지에 대한 단서를 얻을 수 있다.
- 이유는 다양하며, 원래 문자열로 제공되었을 수 있고, 데이터 내에 숫자 외의 문자나 기호가 포함되어 있을 가능성이 있다.
- 'horsepower'의 데이터를 출력해보자.

데이터의 표준화 ② 자료형 변환

- unique()는 선택한 series에 포함된 고유한 값(유일한 값)을 출력한다.

※ 예를 들어 변수에 [1,1,1,1,2,3,4] 라는 값이 담겨있다면 unique를 사용하면 1,2,3,4 라는 출력을 통해 중복되지않는 유일한 고유값을 출력한다.

```
In [17]: print(df['horsepower'].unique())
```

```
['130' '165' '150' '140' '198' '220' '215' '225' '190' '170' '160' '95'
 '97' '85' '88' '46' '87' '90' '113' '200' '210' '193' '?' '100' '105'
 '175' '153' '180' '110' '72' '86' '70' '76' '65' '69' '60' '80' '54'
 '208' '155' '112' '92' '145' '137' '158' '167' '94' '107' '230' '49' '75'
 '91' '122' '67' '83' '78' '52' '61' '93' '148' '129' '96' '71' '98' '115'
 '53' '81' '79' '120' '152' '102' '108' '68' '58' '149' '89' '63' '48'
 '66' '139' '103' '125' '133' '138' '135' '142' '77' '62' '132' '84' '64'
 '74' '116' '82']
```

데이터의 표준화 ② 자료형 변환

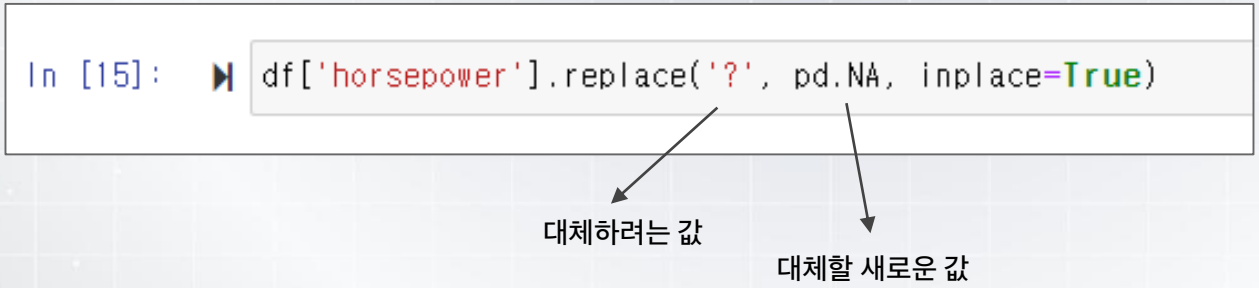
- 하나의 열(column)이 여러 가지 데이터 타입을 가질 수 없다.
- 만약 한 열에 대부분의 값들이 숫자(int 또는 float 등)여도 소수의 값들이 문자열이라면, 해당 열 전체의 데이터 타입은 object로 표시된다.
- 이 경우, 해당 열의 데이터 타입을 적절하게 변환하려면 문자열 값을 변환하거나 제거해야 한다.

```
In [17]: print(df['horsepower'].unique())
```

```
['130' '165' '150' '140' '198' '220' '215' '225' '190' '170' '160' '95'
 '97' '85' '88' '46' '87' '90' '113' '200' '210' '193' '?' '100' '105'
 '175' '153' '180' '110' '72' '86' '70' '76' '65' '69' '60' '80' '54'
 '208' '155' '112' '92' '145' '137' '158' '167' '94' '107' '230' '49' '75'
 '91' '122' '67' '83' '78' '52' '61' '93' '148' '129' '96' '71' '98' '115'
 '53' '81' '79' '120' '152' '102' '108' '68' '58' '149' '89' '63' '48'
 '66' '139' '103' '125' '133' '138' '135' '142' '77' '62' '132' '84' '64'
 '74' '116' '82']
```

데이터의 표준화 ② 자료형 변환

- 이전에 배운 결측치 탐색 및 제거 방법을 사용해서 문자열을 제거할 수 있다.
- replace는 특정 값을 다른 값으로 대체하는 데 사용된다. 다양한 형태의 데이터에 적용될 수 있다.
- replace('?', pd.NA, inplace=True)는 선택된 'horsepower' 열에서 '?' 값을 pd.NA로 대체하는 역할을 합니다.
- pd.NA는 pandas에서 결측치를 만들어 내는 방법
- inplace=True는 대체된 결과를 원본 데이터프레임에 바로 적용하도록 지정



데이터의 표준화 ② 자료형 변환

- 출력하여 결측치(NA 값)으로 제대로 변경되어 있는걸 확인할 수 있다.

```
In [34]: print(df['horsepower'].unique())
```

```
['130' '165' '150' '140' '198' '220' '215' '225' '190' '170' '160' '95'
 '97' '85' '88' '46' '87' '90' '113' '200' '210' '193' <NA> '100' '105'
 '175' '153' '180' '110' '72' '86' '70' '76' '65' '69' '60' '80' '54'
 '208' '155' '112' '92' '145' '137' '158' '167' '94' '107' '230' '49' '75'
 '91' '122' '67' '83' '78' '52' '61' '93' '148' '129' '96' '71' '98' '115'
 '53' '81' '79' '120' '152' '102' '108' '68' '58' '149' '89' '63' '48'
 '66' '139' '103' '125' '133' '138' '135' '142' '77' '62' '132' '84' '64'
 '74' '116' '82']
```

데이터의 표준화 ② 자료형 변환

- df.dropna를 사용해서 결측치를 제거해주기
- subset=['horsepower']는 'horsepower' 열을 기준으로 결측치를 확인하고 제거할 행을 선택하도록 지정하는 것을 의미

```
In [16]: df.dropna(subset=['horsepower'], inplace=True)
```

데이터의 표준화 ② 자료형 변환

- 확인해보면 문자열 ?도 결측치(NA)도 없어진 걸 확인할 수 있다.

```
In [38]: print(df['horsepower'].unique())
```

```
['130' '165' '150' '140' '198' '220' '215' '225' '190' '170' '160' '95'
 '97' '85' '88' '46' '87' '90' '113' '200' '210' '193' '100' '105' '175'
 '153' '180' '110' '72' '86' '70' '76' '65' '69' '60' '80' '54' '208'
 '155' '112' '92' '145' '137' '158' '167' '94' '107' '230' '49' '75' '91'
 '122' '67' '83' '78' '52' '61' '93' '148' '129' '96' '71' '98' '115' '53'
 '81' '79' '120' '152' '102' '108' '68' '58' '149' '89' '63' '48' '66'
 '139' '103' '125' '133' '138' '135' '142' '77' '62' '132' '84' '64' '74'
 '116' '82']
```

데이터의 표준화 ② 자료형 변환

- 원래 데이터 타입이 object였다면, 결측치를 제거해도 데이터 타입은 그대로일 수 있다.
- 때문에 .astype을 사용하여 데이터타입을 명시적으로 실수형(float)로 변경해주기로 한다.

```
In [21]: df['horsepower'] = df['horsepower'].astype(float)
```

데이터의 표준화 ② 자료형 변환

- 값들이 실수이기때문에 소수점 아래의 값을 가질 수 있어서 데이터에 점이 추가되어 보이는 것
- 값 뒤에 점이 붙어있는 것은 출력 형식에 따른 표기 방식이기에, 값 자체에는 영향이 있는 것은 아니다.

```
In [18]: print(df['horsepower'].unique())
```

```
[130. 165. 150. 140. 198. 220. 215. 225. 190. 170. 160.  95.  97.  85.
  88.  46.  87.  90. 113. 200. 210. 193. 100. 105. 175. 153. 180. 110.
  72.  86.  70.  76.  65.  69.  60.  80.  54. 208. 155. 112.  92. 145.
 137. 158. 167.  94. 107. 230.  49.  75.  91. 122.  67.  83.  78.  52.
  61.  93. 148. 129.  96.  71.  98. 115.  53.  81.  79. 120. 152. 102.
 108.  68.  58. 149.  89.  63.  48.  66. 139. 103. 125. 133. 138. 135.
 142.  77.  62. 132.  84.  64.  74. 116.  82.]
```

데이터의 표준화 ② 자료형 변환

- 마지막으로 다시한번 데이터들의 정보를 확인하여 제대로 데이터 타입이 변환된 것을 확인한다.
- 중간중간 데이터의 변화를 체크하여 코드가 제대로 실행되었는지 확인하는 방식은 이후의 발생할 수 있는 오류들을 초기에 예방할 수 있는 방법이다.

```
In [19]: print(df['horsepower'])

0      130.0
1      165.0
2      150.0
3      150.0
4      140.0
       ...
393     86.0
394     52.0
395     84.0
396     79.0
397     82.0
Name: horsepower, Length: 392, dtype: float64
```

```
In [46]: df.dtypes

Out[46]: mpg             float64
         cylinders         int64
         displacement    float64
         horsepower      float64
         weight            int64
         acceleration    float64
         model year        int64
         origin            int64
         name             object
         kmL             float64
         dtype: object
```

데이터의 표준화 ② 자료형 변환 2

- origin 열의 고유 값들을 unique로 출력해보면 1과 2와 3으로 이루어져있다.
- origin 은 출시국가를 표시한 열인데, 실제로는 1은 미국(USA), 2는 영국(EU), 3은 일본(JPN)에 해당하는 정보이다.

```
In [47]: print(df['origin'].unique())
[1 3 2]
```

1 : 미국(USA)

2 : 영국(EU)

3 : 일본(JPN)

데이터의 표준화 ② 자료형 변환 2

- 더 좋은 식별을 위해 replace를 사용하여 숫자 코드에 대응하는 실제 국가 이름으로 변경해줄 것

```
In [47]: print(df['origin'].unique())
[1 3 2]
```

데이터의 표준화 ② 자료형 변환 2

- df['origin'].replace({1: 'USA', 2: 'EU', 3: 'JPN'}, inplace=True)
- 딕셔너리 형태를 사용하여 각 숫자 값을 해당 문자열로 대체하도록 지정한다.

```
In [48]: df['origin'].replace({1:'USA',
                               2 : 'EU',
                               3 : 'JPN'},  inplace=True)
```

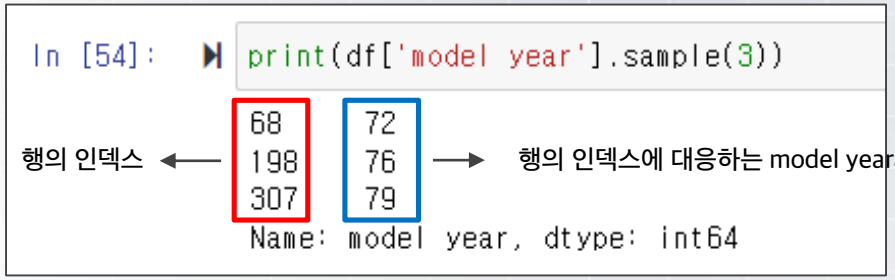
데이터의 표준화 ② 자료형 변환 2

- 'origin' 열의 고유값들을 출력해서 확인하고, 출력해봄으로써 replace 작업으로 인해 변경된 값들을 확인한다.
- 이런 데이터의 경우 반복되는 3개의 값 (범주) 을 가지고 있기 때문에 'category' 라는 데이터형이 더욱 적합하다.

```
In [49]: print(df['origin'].unique())

['USA' 'JPN' 'EU']

In [50]: df['origin']

Out[50]: 0      USA
         1      USA
         2      USA
         3      USA
         4      USA
               ...
         393    USA
         394     EU
         395    USA
         396    USA
         397    USA
         Name: origin, Length: 392, dtype: object
```

데이터의 표준화 ② 자료형 변환 3

- 마지막으로 'model year' 라는 열의 자료형에 대해 알아보자.
- 자동차의 출시년도를 의미하는 데이터이다.
- sample 을 사용하면 무작위로 n개의 데이터 포인트를 선택하여 출력할 수 있다.



데이터의 표준화 ② 자료형 변환 3

- 값들이 년도에 해당하는 숫자로 저장되어 있어, 해당 데이터 타입이 정수형이다.
- 정수형으로 남겨두어도 무방하지만, 연도는 시간적인 순서를 가지고 있기에 범주형으로 표현하는 것이 조금 더 적합하다.
- 또한 제한된 수의 고유값들로 구성되어 있기에 해당 열을 범주형으로 변환하는 것이 데이터 분석에 더욱 적합하다.
- 범주형 데이터 타입은 범주별로 그룹화, 정렬, 비교하기가 더 쉬워진다.

```
In [60]: df['model year'] = df['model year'].astype('category')
```

데이터의 표준화 ② 자료형 변환 3

•다시 한 번 'model year' 열에서 n개의 샘플을 출력하여 확인한다.

•이번에는 데이터 타입이 범주형으로 변경된 후의 샘플임을 확인할 수 있다.

```
In [60]: df['model year'] = df['model year'].astype('category')

In [71]: print(df['model year'].sample(5))

178    75
344    81
213    76
131    74
112    73
Name: model year, dtype: category
Categories (13, int64): [70, 71, 72, 73, ..., 79, 80, 81, 82]
```

•데이터 타입을 category로 변경하면 아래와 같이 categories가 어떻게 이루어져있는지 정보가 추가된다.

• 여기서는 70부터 82까지 총 13개의 서로 다른 범주(년도)가 정수형태로 있다는 것을 의미한다.

•단순한 정수가 아닌, 특정한 범주에 속하는 값이 된 것