

# 판다스 자료형

- 자료형 다루기
- 카테고리 자료형

## ■ 자료형 다루기

### ● 자료형 변환하기

- seaborn 라이브러리의 tips 데이터 집합 확인

```
import pandas as pd
import seaborn as sns

tips = sns.load_dataset("tips")

print(tips.shape)
print(tips.columns)
```

(244, 7)

Index(['total\_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'], dtype='object')

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

## ■ 자료형 다루기

### ● 여러가지 자료형을 문자열로 변환하기 (astype)

- 흡연여부(smoker) 데이터를 문자열로 변환하여 저장

```
tips['smoker_str'] = tips['smoker'].astype(str)
print(tips.dtypes)
```

```
total_bill    float64
tip           float64
sex           category
smoker        category
day           category
time         category
size          int64
smoker_str    object
dtype: object
```

판다스에서 문자열은 오브젝트로 취급

## ■ 자료형 다루기

### ● 여러가지 자료형을 문자열로 변환하기 (astype)

- 전체금액(total\_bill) 데이터를 문자열로 변환

```
tips['total_bill'] = tips['total_bill'].astype(str)
print(tips.dtypes)
```

```
total_bill    object
tip           float64
sex           category
smoker        category
day           category
time          category
size          int64
smoker_str    object
dtype: object
```

## ■ 자료형 다루기

### ● 여러가지 자료형을 문자열로 변환하기 (astype)

- 전체금액(total\_bill) 데이터를 실수로 변환

```
tips['total_bill'] = tips['total_bill'].astype(float)
print(tips.dtypes)
```

```
total_bill    float64
tip           float64
sex           category
smoker        category
day           category
time          category
size          int64
smoker_str    object
dtype: object
```

category로 변환할 때는 `astype('category')`

## ■ 자료형 다루기

### ● 잘못 입력한 데이터 처리하기

- 숫자형태의 데이터에 문자열을 입력하면 object 형태로 자동 변경됨
- astype() 으로 숫자 변환 시에는 문자 데이터 처리가 불가능함

```
ex_df.loc[1, 'age'] = 'empty'  
print(ex_df.dtypes)
```

	name	age	job
0	a	20	designer
1	b	21	programmer
2	c	22	dba



	name	age	job
0	a	20	designer
1	b	empty	programmer
2	c	22	dba

```
name    object  
age      int64  
job      object  
dtype: object
```

```
name    object  
age      object  
job      object  
dtype: object
```

```
ex_df['age'] = ex_df['age'].astype(int)
```

-----  
ValueError

Traceback

<ipython-input-24-c964617de2e1> in <module>

----> 1 ex\_df['age'] = ex\_df['age'].astype(int)

ValueError: invalid literal for int() with base 10: 'empty'

## ■ 자료형 다루기

### ● 잘못 입력한 데이터 처리하기 (to\_numeric)

- to\_numeric 메소드를 사용해도 비슷한 오류가 발생하지만 errors 옵션으로 'raise', 'coerce', 'ignore'를 지정하여 제어 가능

옵션	설명
raise	숫자로 변환할 수 없는 값이 있으면 오류 발생 (기본값)
coerce	숫자로 변환할 수 없는 값을 누락값으로 지정
ignore	아무 작업도 하지 않음

## ■ 자료형 다루기

### ● 잘못 입력한 데이터 처리하기 (to\_numeric)

- float 데이터에 문자열을 입력하여 object 데이터로 자동 변환

```
tips_sub_miss = tips.head(10)
tips_sub_miss.loc[[1, 3, 5, 7], 'total_bill'] = 'missing'

print(tips_sub_miss.dtypes)
print(tips_sub_miss)
```

	total_bill	tip	sex	smoker	day	time	size	smoker_str
0	16.99	1.01	Female	No	Sun	Dinner	2	No
1	missing	1.66	Male	No	Sun	Dinner	3	No
2	21.01	3.50	Male	No	Sun	Dinner	3	No
3	missing	3.31	Male	No	Sun	Dinner	2	No
4	24.59	3.61	Female	No	Sun	Dinner	4	No
5	missing	4.71	Male	No	Sun	Dinner	4	No
6	8.77	2.00	Male	No	Sun	Dinner	2	No
7	missing	3.12	Male	No	Sun	Dinner	4	No
8	15.04	1.96	Male	No	Sun	Dinner	2	No
9	14.78	3.23	Male	No	Sun	Dinner	2	No

```
total_bill    object
tip           float64
sex           category
smoker        category
day           category
time          category
size          int64
smoker_str    object
dtype: object
```



## ■ 자료형 다루기

### ● 잘못 입력한 데이터 처리하기 (to\_numeric)

- 잘못된 값을 누락값으로 변환

```
pd.to_numeric(tips_sub_miss['total_bill'], errors='coerce')
```

```
0    16.99
1     NaN
2    21.01
3     NaN
4    24.59
5     NaN
6     8.77
7     NaN
8    15.04
9    14.78
Name: total_bill, dtype: float64
```

- 잘못된 값 무시

```
pd.to_numeric(tips_sub_miss['total_bill'], errors='ignore')
```

```
0    16.99
1  missing
2    21.01
3  missing
4    24.59
5  missing
6     8.77
7  missing
8    15.04
9    14.78
Name: total_bill, dtype: object
```

## ■ 카테고리 자료형

### ● 카테고리 자료형의 장점과 특징

- 용량과 속도 면에서 매우 효율적
- 주로 동일한 문자열이 반복되어 데이터를 구성하는 경우에 사용

```
tips['smoker'] = tips['smoker'].astype('str')  
tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 244 entries, 0 to 243  
Data columns (total 8 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   total_bill  244 non-null   float64  
1   tip         244 non-null   float64  
2   sex        244 non-null   category  
3   smoker     244 non-null   object  
4   day        244 non-null   category  
5   time       244 non-null   category  
6   size       244 non-null   int64  
7   smoker_str  244 non-null   object  
dtypes: category(3), float64(2), int64(1), object(2)  
memory usage: 10.7+ KB
```

## ■ 카테고리 자료형

### ● 카테고리 자료형의 장점과 특징

- 용량과 속도 면에서 매우 효율적
- 주로 동일한 문자열이 반복되어 데이터를 구성하는 경우에 사용

```
tips['smoker'] = tips['smoker'].astype('category')  
tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 244 entries, 0 to 243  
Data columns (total 8 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   total_bill  244 non-null    float64  
1   tip         244 non-null    float64  
2   sex         244 non-null    category  
3   smoker      244 non-null    category  
4   day         244 non-null    category  
5   time        244 non-null    category  
6   size        244 non-null    int64  
7   smoker_str  244 non-null    object  
dtypes: category(4), float64(2), int64(1), object(1)  
memory usage: 9.2+ KB
```

메모리 사용량이 줄어듦