IT창업개론

4주차 - apply(), 그룹 연산

apply() 활용

간단한 함수 만들기

- 제곱 함수와 n 제곱 함수 만들기

```
def my_sq(x):
    return x ** 2
```

```
def my_exp(x, n):
    return x ** n
```

```
my_sq(4)
```

```
16
```

```
my_exp(2, 4)
```

```
16
```

apply() 사용하기

```
import pandas as pd

df = pd.DataFrame({'a': [10, 20, 30], 'b': [20, 30, 40]})
df
```

	a	b
0	10	20
1	20	30
2	30	40

apply() 사용하기

a열을 제곱하기

```
df['a'] ** 2
```

```
0    100
1    400
2    900
```

apply()로 브로드캐스팅

- 시리즈의 모든 데이터에 제곱 함수 적용

```
def my_sq(x):
    return x ** 2
```

```
df['a'].apply(my_sq)
```

```
0    100
1    400
2    900
```

apply() 사용하기

- 2개 이상의 인자를 전달해야 한다면?

```
def my_exp(x, n):
    return x ** n
```

```
df['a'].apply(my_exp, n=2)
```

```
0     100
1     400
2     900
Name: a, dtype: int64
```

```
df['a'].apply(my_exp, n=3)
```

```
0      1000
1      8000
2     27000
Name: a, dtype: int64
```

데이터프레임과 apply()

```
df = pd.DataFrame({'a': [10, 20, 30], 'b': [20, 30, 40]})
df
```

	a	b
0	10	20
1	20	30
2	30	40

```
df['a']
```

```
0    10
1    20
2    30
Name: a, dtype: int64
```

```
df['b']
```

```
0    20
1    30
2    40
Name: b, dtype: int64
```

```
In [14]: def print_me(x):
             print(x)

In [20]: df.apply(print_me, axis=0)
0    10
1    20
2    30
Name: a, dtype: int64
0    20
1    30
2    40
Name: b, dtype: int64
Out[20]: a    None
         b    None
         dtype: object
```

데이터프레임과 apply()

- 3개의 인자를 입력받아 평균을 계산하는 함수를 구현한다면?

```
def avg_3(x, y, z):
    return (x + y + z) / 3
```

```
df.apply(avg_3)
```

```
def avg_3_apply(col):
    x = col[0]
    y = col[1]
    z = col[2]
    return (x + y + z) / 3


df.apply(avg_3_apply)
```

```
a    20.0
b    30.0
dtype: float64
```

```
TypeError: ("avg_3() missing 2 required positional arguments: 'y' and 'z'", 'occurred at index a')
```

데이터프레임과 apply()

	a	b
0	10	20
1	20	30
2	30	40

```
def avg_3_apply(col):
    sum = 0
    for item in col:
        sum += item
    return sum / df.shape[0]

df.apply(avg_3_apply)
```

```
a    20.0
b    30.0
dtype: float64
```

```
def avg_2_apply(row):
    sum = 0
    for item in row:
        sum += item
    return sum / df.shape[1]
```

```
print(df.apply(avg_2_apply, axis = 1))
```

```
0    15.0
1    25.0
2    35.0
dtype: float64
```

apply() 사용하기 - 고급

⊙ 데이터프레임의 누락값 처리하기 - 열 방향

titanic 데이터 집합에는 타이타닉 침몰 시 생존자에 대한 데이터가 들어 있음

```
import seaborn as sns

titanic = sns.load_dataset("titanic")
print(titanic.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
survived       891 non-null int64
pclass         891 non-null int64
sex            891 non-null object
age            714 non-null float64
sibsp          891 non-null int64
parch          891 non-null int64
fare           891 non-null float64
embarked       889 non-null object
class          891 non-null category
who            891 non-null object
adult_male     891 non-null bool
deck           203 non-null category
embark_town    889 non-null object
alive          891 non-null object
alone          891 non-null bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.6+ KB
None
```

apply() 사용하기 - 고급

- 누락값의 개수를 반환하는 함수

```
import numpy as np

def count_missing(vec):
    null_vec = pd.isnull(vec)
    null_count = np.sum(null_vec)
    return null_count
```

```
cmis_col = titanic.apply(count_missing)
cmis_col
```

```
survived       0
pclass         0
sex            0
age          177
sibsp          0
parch          0
fare           0
embarked       2
class          0
who            0
adult_male     0
deck         688
embark_town    2
alive          0
alone          0
dtype: int64
```

apply() 사용하기 - 고급

- 누락값의 비율을 계산하는 함수

```
def prop_missing(vec):
    num = count_missing(vec)
    dem = vec.size
    return num / dem
```

```
def prop_complete(vec):
    return 1 - prop_missing(vec)
```

```
pmis_col = titanic.apply(prop_missing)
pmis_col
```

```
survived       0.000000
pclass         0.000000
sex            0.000000
age            0.198653
sibsp          0.000000
parch          0.000000
fare           0.000000
embarked       0.002245
class          0.000000
who            0.000000
adult_male     0.000000
deck           0.772166
embark_town    0.002245
alive          0.000000
alone          0.000000
dtype: float64
```

apply() 사용하기 - 고급

- 데이터프레임의 누락값 처리하기 - 행 방향

```
cmis_row = titanic.apply(count_missing, axis=1)
pmis_row = titanic.apply(prop_missing, axis=1)
pcom_row = titanic.apply(prop_complete, axis=1)
```

```
cmis_row.head()
```

```
0    1
1    0
2    1
3    0
4    1
dtype: int64
```

```
pmis_row.head()
```

```
0    0.066667
1    0.000000
2    0.066667
3    0.000000
4    0.066667
dtype: float64
```

```
pcom_row.head()
```

```
0    0.933333
1    1.000000
2    0.933333
3    1.000000
4    0.933333
dtype: float64
```

apply() 사용하기 - 고급

- 누락값이 있는 데이터만 따로 모아서 보기

```
titanic['num_missing'] = titanic.apply(count_missing, axis=1)
titanic.iloc[:, [0, 1, 2, titanic.columns.size - 1]].head()
```

	survived	pclass	sex	num_missing
0	0	3	male	1
1	1	1	female	0
2	1	3	female	1
3	1	1	female	0
4	0	3	male	1

```
titanic.loc[titanic.num_missing > 1, :].iloc[:, [0, 1, 2, titanic.columns.size - 1]].sample(10)
```

	survived	pclass	sex	num_missing
732	0	2	male	2
773	0	3	male	2
240	0	3	female	2
95	0	3	male	2
82	1	3	female	2
109	1	3	female	2
358	1	3	female	2
29	0	3	male	2
415	0	3	female	2
420	0	3	male	2

그룹 연산

데이터 집계

- 수집한 데이터를 바탕으로 평균이나 합 등을 구하여 의미 있는 값을 도출해 내는 것을 '집계' 라고 함

- groupby()로 평균값 구하기

```
import pandas as pd
df = pd.read_csv('../data/gapminder.tsv', sep='₩t')
```

```
df.groupby('year').lifeExp.mean()
```

```
year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

lifeExp 열의 연도별 평균값 계산 과정 살펴보기

1. year 열의 데이터를 중복 없이 추출

```
years = df.year.unique()
years
```

```
array([1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 1997, 2002,
       2007], dtype=int64)
```

2. 각 연도별로 데이터 추출

```
y1952 = df.loc[df.year == 1952, :]
y1952.head()
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
12	Albania	Europe	1952	55.230	1282697	1601.056136
24	Algeria	Africa	1952	43.077	9279525	2449.008185
36	Angola	Africa	1952	30.015	4232095	3520.610273
48	Argentina	Americas	1952	62.485	17876956	5911.315053

lifeExp 열의 연도별 평균값 계산 과정 살펴보기

3. 과정 2에서 추출한 데이터에서 lifeExp 열의 평균값 계산

```
y1952_mean = y1952.lifeExp.mean()
y1952_mean
```

```
49.05761971830987
```

4. 결합

```
pd.DataFrame({"year":[1952, 1957, 1962, 2007],
              "":[y1952_mean, y1957_mean,y1962_mean,y2007_mean]})
```

	year	
0	1952	49.057620
1	1957	51.507401
2	1962	53.609249
3	2007	67.007423

groupby()와 함께 사용하는 집계 함수

count	누락값을 제외한 데이터 수를 반환
size	누락값을 포함한 데이터 수를 반환
mean	평균 값 반환
std	표준편차 반환
min	최솟값 반환
max	최댓값 반환
sum	전체 합 반환
var	분산 반환

groupby()와 함께 사용하는 집계 함수

quantile(q=0.25)	백분위 수 25%
quantile(q=0.50)	백분위 수 50%
quantile(q=0.75)	백분위 수 75%
sem	평균의 표준편차 반환
describe	데이터 수, 평균, 표준편차, 최솟값, 최댓값, 백분위 수 반환
first	첫 번째 행 반환
last	마지막 행 반환
nth	n번째 행 반환

agg()로 사용자 함수와 groupby() 조합하기

- 라이브러리에서 제공하는 집계 함수로 원하는 값을 계산할 수 없다면?

```
def my_mean(values):
    n = len(values)

    sum = 0
    for value in values:
        sum += value

    return sum / n
```

```
df.groupby('year').lifeExp.agg(my_mean)
```

```
year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

agg()로 사용자 함수와 groupby() 조합하기

- 2개의 인자값을 받아 처리하는 사용자 정의 함수

첫 번째 인자로 받은 열의 평균값을 구하여 두 번째 인자로 받은 값과의 차이를 계산한 다음 반환하는 함수

```
global_mean = df.lifeExp.mean()
global_mean
```

```
59.47443936619713
```

```
df.groupby('year').lifeExp.agg(my_mean_diff, diff_value=global_mean)
```

```
year
1952   -10.416820
1957    -7.967038
1962    -5.865190
1967    -3.796150
1972    -1.827053
1977     0.095718
1982     2.058758
1987     3.738173
1992     4.685899
1997     5.540237
2002     6.220483
2007     7.532983
Name: lifeExp, dtype: float64
```

```
def my_mean_diff(values, diff_value):
    n = len(values)
    sum = 0
    for value in values:
        sum += value
    mean = sum / n
    return mean - diff_value
```

여러 개의 집계 함수 한 번에 사용하기

```
import numpy as np
df.groupby('year').lifeExp.agg([np.count_nonzero, np.mean, np.std])
```

year	count_nonzero	mean	std
1952	142.0	49.057620	12.225956
1957	142.0	51.507401	12.231286
1962	142.0	53.609249	12.097245
1967	142.0	55.678290	11.718858
1972	142.0	57.647386	11.381953
1977	142.0	59.570157	11.227229
1982	142.0	61.533197	10.770618
1987	142.0	63.212613	10.556285
1992	142.0	64.160338	11.227380
1997	142.0	65.014676	11.559439
2002	142.0	65.694923	12.279823
2007	142.0	67.007423	12.073021

여러 개의 집계 함수 한 번에 사용하기

```
df.groupby('year').agg({'lifeExp': 'mean', 'pop': 'median', 'gdpPercap': 'median'})
```

year	lifeExp	pop	gdpPercap
1952	49.057620	3943953.0	1968.528344
1957	51.507401	4282942.0	2173.220291
1962	53.609249	4686039.5	2335.439533
1967	55.678290	5170175.5	2678.334741
1972	57.647386	5877996.5	3339.129407
1977	59.570157	6404036.5	3798.609244
1982	61.533197	7007320.0	4216.228428
1987	63.212613	7774861.5	4280.300366
1992	64.160338	8688686.5	4386.085502
1997	65.014676	9735063.5	4781.825478
2002	65.694923	10372918.5	5319.804524
2007	67.007423	10517531.0	6124.371109

데이터 필터링

- 30번 이상의 주문이 있는 테이블만 그룹화하기

```
tips = sns.load_dataset('tips')
tips.shape
```

```
(244, 7)
```

```
tips['size'].value_counts()
```

```
2    156
3     38
4     37
5      5
6      4
1      4
```

```
tips_filtered = tips.groupby('size').filter(lambda x: x['size'].count() >= 30)
```

```
tips_filtered.shape
```

```
(231, 7)
```

```
tips_filtered['size'].value_counts()
```

```
2    156
3     38
4     37
Name: size, dtype: int64
```

그룹 오브젝트

- groupby()의 리턴 타입은 그룹 오브젝트

```
tips_10 = sns.load_dataset('tips').sample(10, random_state=42)
tips_10
```

	total_bill	tip	sex	smoker	day	time	size
24	19.82	3.18	Male	No	Sat	Dinner	2
6	8.77	2.00	Male	No	Sun	Dinner	2
153	24.55	2.00	Male	No	Sun	Dinner	4
211	25.89	5.16	Male	Yes	Sat	Dinner	4
198	13.00	2.00	Female	Yes	Thur	Lunch	2
176	17.89	2.00	Male	Yes	Sun	Dinner	2
192	28.44	2.56	Male	Yes	Thur	Lunch	2
124	12.48	2.52	Female	No	Thur	Lunch	2
9	14.78	3.23	Male	No	Sun	Dinner	2
101	15.38	3.00	Female	Yes	Fri	Dinner	2

```
grouped = tips_10.groupby('sex')
grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001AFE46CD388>
```

```
grouped.groups
```

```
{'Male': Int64Index([24, 6, 153, 211, 176, 192, 9], dtype='int64'),
 'Female': Int64Index([198, 124, 101], dtype='int64')}
```

그룹 오브젝트

- 그룹 오브젝트의 평균 구하기

```
grouped.mean()
```

sex	total_bill	tip	size
Male	20.02	2.875714	2.571429
Female	13.62	2.506667	2.000000

```
tips_10.dtypes
```

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

그룹 오브젝트

그룹 오브젝트를 반복문에 사용하기

sex 열을 기준으로 그룹화한 tips 데이터 집합은 여성 그룹과 남성 그룹으로 나누어져 있음

```
for sex_group in grouped:
    print(sex_group)
```

```
('Male',      total_bill   tip   sex smoker   day    time  size
24         19.82  3.18  Male     No   Sat  Dinner     2
6           8.77  2.00  Male     No   Sun  Dinner     2
153        24.55  2.00  Male     No   Sun  Dinner     4
211        25.89  5.16  Male    Yes   Sat  Dinner     4
176        17.89  2.00  Male    Yes   Sun  Dinner     2
192        28.44  2.56  Male    Yes  Thur   Lunch     2
9          14.78  3.23  Male     No   Sun  Dinner     2)
('Female',      total_bill   tip     sex smoker   day    time  size
198        13.00  2.00  Female    Yes  Thur   Lunch     2
124        12.48  2.52  Female     No  Thur   Lunch     2
101        15.38  3.00  Female    Yes   Fri  Dinner     2)
```

```
for sex_group in grouped:
    print(type(sex_group))
```

```
<class 'tuple'>
<class 'tuple'>
```

그룹 오브젝트

- 그룹 오브젝트에서 특정 데이터만 추출하기

sex 열로 그룹화한 그룹 오브젝트에 성별이 여성인 데이터만 추출

```
female = grouped.get_group('Female')
print(female)
```

```
     total_bill   tip     sex smoker   day    time  size
198       13.00  2.00  Female    Yes  Thur   Lunch     2
124       12.48  2.52  Female     No  Thur   Lunch     2
101       15.38  3.00  Female    Yes   Fri  Dinner     2
```

그룹 오브젝트

- 여러 열을 사용해 그룹 오브젝트를 만들고 계산하기

```
bill_sex_time = tips_10.groupby(['sex', 'time'])
group_avg = bill_sex_time.mean()
group_avg
```

sex	time	total_bill	tip	size
Male	Lunch	28.440000	2.560000	2.000000
	Dinner	18.616667	2.928333	2.666667
Female	Lunch	12.740000	2.260000	2.000000
	Dinner	15.380000	3.000000	2.000000

```
type(group_avg)
```

```
pandas.core.frame.DataFrame
```

```
group_avg.columns
```

```
Index(['total_bill', 'tip', 'size'], dtype='object')
```

```
group_avg.index
```

```
MultiIndex([(  'Male',  'Lunch'),
            (  'Male', 'Dinner'),
            ('Female',  'Lunch'),
            ('Female', 'Dinner')],
           names=['sex', 'time'])
```

그룹 오브젝트

- 여러 열을 사용해 그룹 오브젝트를 만들고 계산하기
  - Multiindex의 경우 reset_index()로 데이터프레임의 인덱스를 새로 부여 가능

```
tips_10.groupby(['sex', 'time']).mean().reset_index()
```

	sex	time	total_bill	tip	size
0	Male	Lunch	28.440000	2.560000	2.000000
1	Male	Dinner	18.616667	2.928333	2.666667
2	Female	Lunch	12.740000	2.260000	2.000000
3	Female	Dinner	15.380000	3.000000	2.000000

```
tips_10.groupby(['sex', 'time'], as_index=False).mean( )
```

	sex	time	total_bill	tip	size
0	Male	Lunch	28.440000	2.560000	2.000000
1	Male	Dinner	18.616667	2.928333	2.666667
2	Female	Lunch	12.740000	2.260000	2.000000
3	Female	Dinner	15.380000	3.000000	2.000000

Thank You!