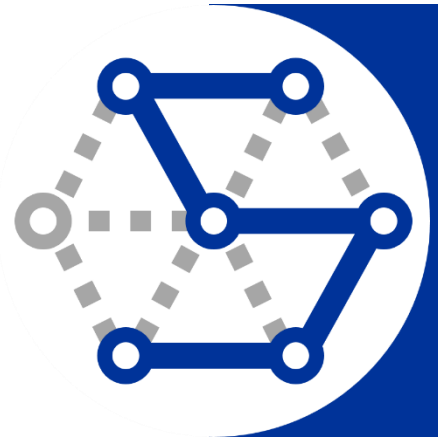
데이터 처리 프로그래밍

Data Processing Programming

13
데이터 연산 및 집계

목차



1. 데이터 연산

Pandas에서 데이터 연산

Pandas에서 데이터 연산

[79]:
```
S1 = pd.Series([1, 3, 5], index=[0, 1, 2])
```

[80]:
```
S2 = pd.Series([2, 4, 6], index=[1, 2, 3])
```

[81]:
```
S1 + S2
```

[81]:
```
0    NaN
1    5.0
2    9.0
3    NaN
dtype: float64
```

[82]:
```
S1.add(S2, fill_value=0)
```

[82]:
```
0    1.0
1    5.0
2    9.0
3    6.0
dtype: float64
```

[83]:
```
D1 = pd.DataFrame(np.random.randint(0, 10, (3, 4)),
                  columns=list('ABCD'))
```

[84]:
```
D1
```

[84]:

	A	B	C	D
0	9	5	6	8
1	0	7	6	1
2	1	1	3	1

[85]:
```
D2 = pd.DataFrame(np.random.randint(0, 10, (3, 3)),
                  columns=list('BAC'))
```

[86]:
```
D2
```

[86]:

	B	A	C
0	0	2	8
1	7	1	9
2	8	1	5

Pandas에서 데이터 연산

```
[87]: D1 + D2
```

[87]:

	A	B	C	D
0	11	5	14	NaN
1	1	14	15	NaN
2	2	9	8	NaN

```
[88]: D1_mean = D1.stack().mean()
```

```
[89]: D1_mean
```

[89]: 4.0

```
[90]: D1.add(D2, fill_value=D1_mean)
```

[90]:

	A	B	C	D
0	11	5	14	12.0
1	1	14	15	5.0
2	2	9	8	5.0

```
[91]: D1 - D1.iloc[0]
```

[91]:

	A	B	C	D
0	0	0	0	0
1	1	-8	-5	0
2	1	-5	-7	0

```
[92]: D1.subtract(D1['B'], axis=0)
```

[92]:

	A	B	C	D
0	-7	0	0	-1
1	2	0	3	7
2	-1	0	-2	4

Pandas에서 데이터 연산

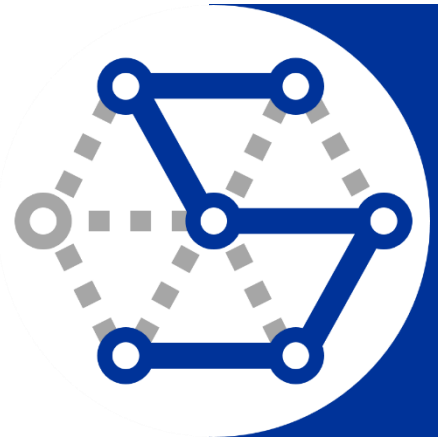
```
[93]: D1.iloc[0, ::2]
```

```
[93]: A    2
      C    9
      Name: 0, dtype: int32
```

```
[94]: D1 - D1.iloc[0, ::2]
```

[94]:

	A	B	C	D
0	0.0	NaN	0.0	NaN
1	1.0	NaN	-5.0	NaN
2	1.0	NaN	-7.0	NaN



2. 누락 값 처리

누락값 처리

누락값 처리

```
[95]: series = pd.Series([1, None, 3, np.nan])

[96]: series

[96]: 0    1.0
      1    NaN
      2    3.0
      3    NaN
      dtype: float64

[97]: series.isnull()

[97]: 0    False
      1     True
      2    False
      3     True
      dtype: bool

[98]: series.notnull()

[98]: 0     True
      1    False
      2     True
      3    False
      dtype: bool
```

```
[99]: series.dropna()

[99]: 0    1.0
      2    3.0
      dtype: float64

[100]: series.fillna(0)

[100]: 0    1.0
       1    0.0
       2    3.0
       3    0.0
       dtype: float64

[101]: series.fillna(method='ffill')

[101]: 0    1.0
       1    1.0
       2    3.0
       3    3.0
       dtype: float64

[102]: series.fillna(method='bfill')

[102]: 0    1.0
       1    3.0
       2    3.0
       3    NaN
       dtype: float64
```

누락값 처리

[103]:
```
df = pd.DataFrame([[1, np.nan, 3],
                   [2, 3, 4],
                   [np.nan, 4, 5]])
df
```

[103]:

	0	1	2
0	1.0	NaN	3
1	2.0	3.0	4
2	NaN	4.0	5

[104]:
```
df.dropna()
```

[104]:

	0	1	2
1	2.0	3.0	4

[105]:
```
df.dropna(axis='columns')
```

[105]:

	2
0	3
1	4
2	5

[106]:
```
df[3] = np.nan
df
```

[106]:

	0	1	2	3
0	1.0	NaN	3	NaN
1	2.0	3.0	4	NaN
2	NaN	4.0	5	NaN

[107]:
```
df.dropna(axis='rows', thresh=3)
```

[107]:

	0	1	2	3
1	2.0	3.0	4	NaN

누락값 처리

[108]:
```
df.fillna(method='bfill', axis=0)
```

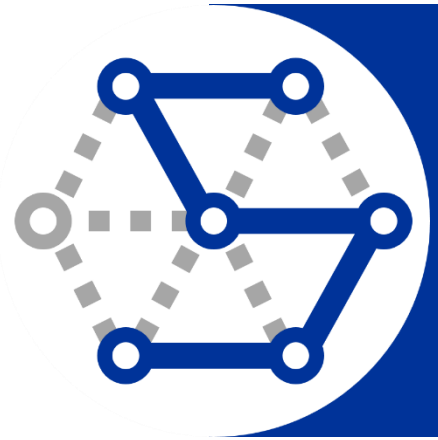
[108]:

	0	1	2	3
0	1.0	3.0	3	NaN
1	2.0	3.0	4	NaN
2	NaN	4.0	5	NaN

[109]:
```
df.fillna(method='ffill', axis=1)
```

[109]:

	0	1	2	3
0	1.0	1.0	3.0	3.0
1	2.0	3.0	4.0	4.0
2	NaN	4.0	5.0	5.0



3. 집계와 분류

집계와 분류

집계와 분류

[110]:
```
import seaborn as sns
titanic = sns.load_dataset('titanic')
titanic.shape
```

[110]:
```
(891, 15)
```

[111]:
```
titanic.head()
```

[111]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	w
0	0	3	male	22.0	1	0	7.2500	S	Third	m
1	1	1	female	38.0	1	0	71.2833	C	First	wom
2	1	3	female	26.0	0	0	7.9250	S	Third	wom
3	1	1	female	35.0	1	0	53.1000	S	First	wom
4	0	3	male	35.0	0	0	8.0500	S	Third	m

[112]:
```
titanic.dropna().describe()
```

[112]:

	survived	pclass	age	sibsp	parch	fare
count	182.000000	182.000000	182.000000	182.000000	182.000000	182.000000
mean	0.675824	1.192308	35.623187	0.467033	0.478022	78.919735
std	0.469357	0.516411	15.671615	0.645007	0.755869	76.490774
min	0.000000	1.000000	0.920000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	24.000000	0.000000	0.000000	29.700000
50%	1.000000	1.000000	36.000000	0.000000	0.000000	57.000000
75%	1.000000	1.000000	47.750000	1.000000	1.000000	90.000000
max	1.000000	3.000000	80.000000	3.000000	4.000000	512.329200

집계와 분류


```
[113]: titanic.count()

[113]: survived       891
       pclass         891
       sex            891
       age            714
       sibsp          891
       parch          891
       fare           891
       embarked       889
       class          891
       who            891
       adult_male     891
       deck           203
       embark_town    889
       alive          891
       alone          891
       dtype: int64

[114]: titanic.mean()

[114]: survived       0.383838
       pclass         2.308642
       age           29.699118
       sibsp          0.523008
       parch          0.381594
       fare          32.204208
       adult_male     0.602694
       alone          0.602694
       dtype: float64
```

```
[115]: titanic.std()

[115]: survived       0.486592
       pclass         0.836071
       age           14.526497
       sibsp          1.102743
       parch          0.806057
       fare          49.693429
       adult_male     0.489615
       alone          0.489615
       dtype: float64

[116]: titanic.mad()

[116]: survived       0.473013
       pclass         0.761968
       age           11.322944
       sibsp          0.713780
       parch          0.580742
       fare          28.163692
       adult_male     0.478908
       alone          0.478908
       dtype: float64
```

집계와 분류

```
[117]: titanic.groupby('class')['fare'].median()
```

```
[117]: class
       First     60.2875
       Second    14.2500
       Third      8.0500
       Name: fare, dtype: float64
```

```
[118]: titanic.groupby('class')['alive'].count()
```

```
[118]: class
       First     216
       Second    184
       Third     491
       Name: alive, dtype: int64
```

```
[119]: titanic.groupby('class')['age'].describe()
```

[119]:

class	count	mean	std	min	25%	50%	75%	max
First	186.0	38.233441	14.802856	0.92	27.0	37.0	49.0	80.0
Second	173.0	29.877630	14.001077	0.67	23.0	29.0	36.0	70.0
Third	355.0	25.140620	12.495398	0.42	18.0	24.0	32.0	74.0

```
[120]: titanic.groupby('class').aggregate([np.median, max])
```

[120]:

	survived		pclass		age		sibsp		pa
	median	max	median	max	median	max	median	max	median
class									
First	1	1	1	1	37.0	80.0	0	3	0
Second	0	1	2	2	29.0	70.0	0	3	0
Third	0	1	3	3	24.0	74.0	0	8	0

```
[121]: titanic.groupby('class').std()
```

[121]:

class	survived	pclass	age	sibsp	parch	fare	adult_male
First	0.484026	0.0	14.802856	0.611898	0.693997	78.380373	0.498555
Second	0.500623	0.0	14.001077	0.601633	0.690963	13.417399	0.499911
Third	0.428949	0.0	12.495398	1.374883	0.888861	11.778142	0.477552

집계와 분류


```
[122]: titanic.groupby('class').filter(lambda x: x['fare'].mean() > 50)
```

[122]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True
23	1	1	male	28.0	0	0	35.5000	S	First	man	True	A	Southampton	yes	True
27	0	1	male	19.0	3	2	263.0000	S	First	man	True	C	Southampton	no	False
30	0	1	male	40.0	0	0	27.7208	C	First	man	True	NaN	Cherbourg	no	True
31	1	1	female	NaN	1	0	146.5208	C	First	woman	False	B	Cherbourg	yes	False
34	0	1	male	28.0	1	0	82.1708	C	First	man	True	NaN	Cherbourg	no	False
35	0	1	male	42.0	1	0	52.0000	S	First	man	True	NaN	Southampton	no	False
52	1	1	female	49.0	1	0	76.7292	C	First	woman	False	D	Cherbourg	yes	False
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
55	1	1	male	NaN	0	0	35.5000	S	First	man	True	C	Southampton	yes	True
61	1	1	female	38.0	0	0	80.0000	NaN	First	woman	False	B	NaN	yes	True
62	0	1	male	45.0	1	0	83.4750	S	First	man	True	C	Southampton	no	False
64	0	1	male	NaN	0	0	27.7208	C	First	man	True	NaN	Cherbourg	no	True

집계와 분류

[123]:
```
titanic["class_fare_mean"] = titanic.groupby('class')['fare'].transform('mean')
```

[124]:
```
titanic.head()
```

[124]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	class_fare_mean
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False	13.675550
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	84.154687
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True	13.675550
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	84.154687
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True	13.675550

[125]:
```
titanic.groupby('sex')[['survived']].mean()
```

[125]:

sex	survived
female	0.742038
male	0.188908

집계와 분류

[126]:
```
titanic.groupby(['sex', 'class'])['survived'].aggregate('mean').unstack()
```

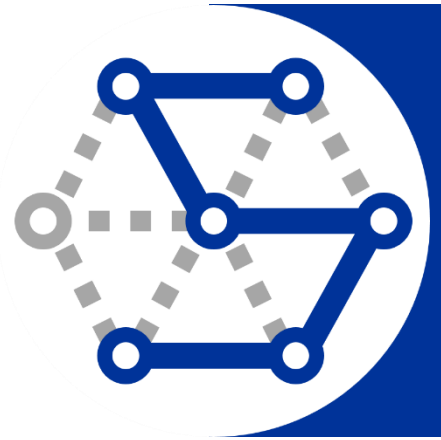
[126]:

sex	First	Second	Third
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

[127]:
```
titanic.pivot_table('survived', index='sex', columns='class')
```

[127]:

sex	First	Second	Third
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447



4. 고성능 연산 및 질의

고성능 연산 및 질의

고성능 연산 및 질의

[128]:
```
nrows, ncols = 100000, 100
rs = np.random.RandomState(42)
df1, df2, df3, df4 = (pd.DataFrame(rs.rand(nrows, ncols))
                      for i in range(4))
```

[129]:
```
%timeit df1 + df2 + df3 + df4
```
```
72.2 ms ± 1.32 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[130]:
```
%timeit pd.eval('df1 + df2 + df3 + df4')
```
```
33.5 ms ± 1.01 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[131]:
```
%timeit r1 = -df1 * df2 / (df3 + df4)
```
```
109 ms ± 2.02 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[132]:
```
%timeit r2 = pd.eval('-df1 * df2 / (df3 + df4)')
```
```
32.1 ms ± 1.38 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[133]:
```
%timeit r1 = (df1 < df2) & (df2 <= df3) & (df3 != df4)
```
```
49.6 ms ± 1.18 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[134]:
```
%timeit r2 = pd.eval('df1 < df2 <= df3 != df4')
```
```
31.2 ms ± 929 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[135]:
```
%timeit r1 = (df1 < 0.5) & (df2 < 0.5) | (df3 < df4)
```
```
170 ms ± 4.96 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[136]:
```
%timeit r2 = pd.eval('(df1 < 0.5) & (df2 < 0.5) | (df3 < df4)')
```
```
37.9 ms ± 2.47 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

고성능 연산 및 질의

[137]:
```
df = pd.DataFrame(rs.rand(10000, 3), columns=['A', 'B', 'C'])
df.head()
```

[137]:

	A	B	C
0	0.615875	0.525167	0.047354
1	0.330858	0.412879	0.441564
2	0.689047	0.559068	0.230350
3	0.290486	0.695479	0.852587
4	0.424280	0.534344	0.245216

[138]:
```
r1 = (df['A'] + df['B']) / (df['C'] - 1)
r2 = pd.eval('(df.A + df.B) / (df.C - 1)')
r3 = df.eval('(A + B) / (C - 1)')
```

[139]:
```
df.eval('D = (A + B) / C', inplace=True)
df.head()
```

[139]:

	A	B	C	D
0	0.615875	0.525167	0.047354	24.095868
1	0.330858	0.412879	0.441564	1.684325
2	0.689047	0.559068	0.230350	5.418335
3	0.290486	0.695479	0.852587	1.156439
4	0.424280	0.534344	0.245216	3.909296

[140]:
```
col_mean = df.mean(1)
r1 = df['A'] + col_mean
r2 = df.eval('A + @col_mean')
```

[141]:
```
r1 = df[(df.A < 0.5) & (df.B < 0.5)]
r2 = pd.eval('(df.A < 0.5) & (df.B < 0.5)')
r3 = df.eval('A < 0.5 and B < 0.5')
r4 = df.query('A < 0.5 and B < 0.5')
```

고성능 연산 및 질의

[140]:
```
col_mean = df.mean(1)
r1 = df['A'] + col_mean
r2 = df.eval('A + @col_mean')
```

[141]:
```
r1 = df[(df.A < 0.5) & (df.B < 0.5)]
r2 = pd.eval('(df.A < 0.5) & (df.B < 0.5)')
r3 = df.eval('A < 0.5 and B < 0.5')
r4 = df.query('A < 0.5 and B < 0.5')
```

[142]:
```
df.query('index < 10')
```

[142]:

	A	B	C	D
0	0.615875	0.525167	0.047354	24.095868
1	0.330858	0.412879	0.441564	1.684325
2	0.689047	0.559068	0.230350	5.418335
3	0.290486	0.695479	0.852587	1.156439
4	0.424280	0.534344	0.245216	3.909296
5	0.111085	0.566276	0.464323	1.458811
6	0.181763	0.511544	0.368457	1.881651
7	0.048330	0.779839	0.459669	1.801662
8	0.448611	0.415924	0.481001	1.797365
9	0.716939	0.031495	0.111552	6.709286

[143]:
```
titanic.query("survived == 1 and sex == 'female' and age < 5")
```

[143]:

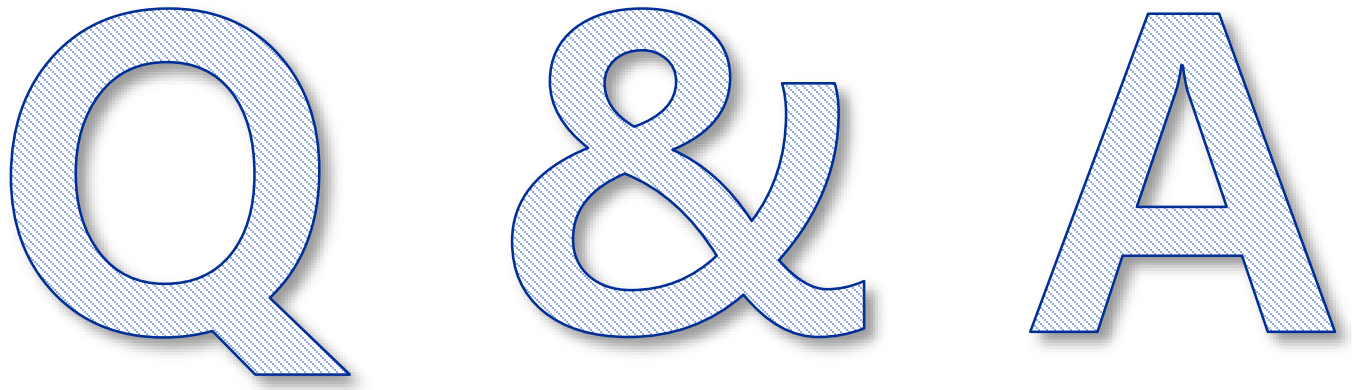
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	wh
10	1	3	female	4.00	1	1	16.7000	S	Third	chil
43	1	2	female	3.00	1	2	41.5792	C	Second	chil
172	1	3	female	1.00	1	1	11.1333	S	Third	chil
184	1	3	female	4.00	0	2	22.0250	S	Third	chil
381	1	3	female	1.00	0	2	15.7417	C	Third	chil
469	1	3	female	0.75	2	1	19.2583	C	Third	chil
479	1	3	female	2.00	0	1	12.2875	S	Third	chil
530	1	2	female	2.00	1	1	26.0000	S	Second	chil
618	1	2	female	4.00	2	1	39.0000	S	Second	chil
644	1	3	female	0.75	2	1	19.2583	C	Third	chil
691	1	3	female	4.00	0	1	13.4167	C	Third	chil
750	1	2	female	4.00	1	1	23.0000	S	Second	chil

고성능 연산 및 질의


```
[144]: titanic.query("survived == 1 and pclass == 3 and who == 'child'")
```

[144]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
10	1	3	female	4.00	1	1	16.7000	S	Third	child
22	1	3	female	15.00	0	0	8.0292	Q	Third	child
39	1	3	female	14.00	1	0	11.2417	C	Third	child
125	1	3	male	12.00	1	0	11.2417	C	Third	child
165	1	3	male	9.00	0	2	20.5250	S	Third	child
172	1	3	female	1.00	1	1	11.1333	S	Third	child
184	1	3	female	4.00	0	2	22.0250	S	Third	child
233	1	3	female	5.00	4	2	31.3875	S	Third	child
261	1	3	male	3.00	4	2	31.3875	S	Third	child
348	1	3	male	3.00	1	1	15.9000	S	Third	child
381	1	3	female	1.00	0	2	15.7417	C	Third	child
448	1	3	female	5.00	2	1	19.2583	C	Third	child
469	1	3	female	0.75	2	1	19.2583	C	Third	child
479	1	3	female	2.00	0	1	12.2875	S	Third	child
489	1	3	male	9.00	1	1	15.9000	S	Third	child
644	1	3	female	0.75	2	1	19.2583	C	Third	child


Q & A