Introduction to Data Mining Lecture

Week 7: Pandas Advanced (1)

Joon Young Kim

Assistant Professor, School of AI Convergence
Sungshin Women's University

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

Week 6 Review

■ 판다스 데이터 처리 학습

→ 데이터 선택/세팅 및 미싱 데이터 처리등 실습

포맷 타입	데이터 유형	읽기 함수	쓰기 함수
텍스트	CSV	read_csv	to_csv
	JSON	read_json	to_json
	HTML	read_html	to_html
	local clipboard	read_clipboard	to_clipboard
이진 데이터	MS Excel	read_excel	to_excel
	HDF5 Format	read_hdf	to_hdf
	Feather Format	read_feather	to_feather
	Parquet Format	read_parquet	to_parquet
	Msgpack	read_msgpack	to_msgpack
	Stata	read_stata	to_stata
	SAS	read_sas	-
	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
	Google Big Query	read_gbq	to_gbq

Week 6 Review

■ 판다스 데이터 I/O 및 종류

→ 텍스트 파일 부터 SQL DB까지 종합적인 학습 진행

5 test_no별 측정데이터 구하기

```
In [559]: df_chunk = pd.read_csv('air_test.csv', chunksize=10,000)

In [560]: ser = pd.Series([])
          for chunk in df_chunk:
             ser = ser.add(chunk['test_no'].value_counts, fill_value=0)
          ser = ser.sort_values()

In [561]: ser
Out[561]: 1    48600.0
          3    48600.0
          5    48600.0
          6    48600.0
          8    48600.0
          9    48600.0
          dtype: float64

In [562]: ser[:5]
Out[562]: 1    48600.0
          3    48600.0
          5    48600.0
```

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

데이터 가공하기

1.1 데이터 연접하기

■ pd.concat() 함수로 연접하기 – df1, df2, df3 객체 생성

```
In [1]: df1 = pd.DataFrame({‘A’: [‘A0’, ‘A1’, ‘A2’],
                            ‘B’: [‘B0’, ‘B1’, ‘B2’],
                            ‘C’: [‘C0’, ‘C1’, ‘C2’]}, index=[0, 1, 2])

In [2]: df2 = pd.DataFrame({‘A’: [‘A3’, ‘A4’, ‘A5’],
                            ‘B’: [‘B3’, ‘B4’, ‘B5’],
                            ‘C’: [‘C3’, ‘C4’, ‘C5’]}, index=[3, 4, 5])

In [3]: df3 = pd.DataFrame({‘A’: [‘A6’, ‘A7’, ‘A8’],
                            ‘B’: [‘B6’, ‘B7’, ‘B8’],
                            ‘C’: [‘C6’, ‘C7’, ‘C8’]}, index=[6, 7, 8])
```

데이터 연접하기

- pd.concat() 함수로 연접하기 – 매개변수로 list 또는 같은 타입의 dict
- .loc로 데이터 선택

```
In [4]: frames = [df1, df2, df3]

In [5]: result = pd.concat(frames)

In [6]: result1 = pd.concat(frames, keys=[‘x’, ‘y’, ‘z’])

In [7]: result                    In [8]: result1
In [9]: result1.loc[‘z’]
```

데이터 연접하기

- 축의 로직 설정과 append를 사용하는 연접
- 연접 시 다른 축들의 처리
  - join='outer'로 하고 모두 union을 취한다.
  - join='inner'로 하고 intersection을 취한다.
  - join_axes 인수로 전달 시 특정 index를 사용

df1과 df4의 연접 : join='outer'

```
In [10]: df4 = pd.DataFrame({'B': ['B2', 'B6', 'B7'],
                             'C': ['C2', 'C6', 'C7'],
                             'E': ['E2', 'E6', 'E7']}, index=[2, 6, 7]}

In [11]: result = pd.concat([df1, df4], axis=1, sort=False)
```

df1과 df4의 연접 : join='inner'

```
In [13] ]: result = pd.concat([df1, df4], axis=1, join='inner')

In [14]: result
```

데이터 연접하기

- pd.concat – 인수 join_axes
- pd.concat – 인수 ignore_index

```
In [15]: result = pd.concat([df1, df4], axis=1,
                                    join_axes=[df1.index])

In [16]: result
In [17]: result = pd.concat([df1, df4], ignore_index=True)

In [18]: result
```

데이터 연접하기

- df.append()로 concat()와 같은 결과 내기
- df.append() : sort=False

```
In [19]: result = df1.append(df2)

In [20]: result
In [21]: result = df1.append(df4, sort=False)

In [22]: result
```

데이터 연접하기

- df.append() – Series를 전달하여 1행을 추가
- df.append() – dict를 전달하여 2행을 추가

```
In [23]: s1 = pd.Series(['Q0', 'Q1', 'Q2', 'Q3'],
                        index=['A', 'B', 'C', 'D'])

In [24]: result = df1.append(s1, ignore_index=True)

In [25]: result
In [26]: dicts = [{'A': 1, 'B': 2, 'X': 3}, {'A': 4, 'B': 5, 'Y': 6}]

In [27]: result = df1.append(dicts, ignore_index=True,
                             sort=False)

In [28]: result
```

데이터 연접하기

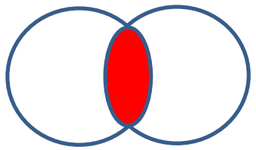
- 차원이 다른 Series와 DataFrame의 연접
  - Series와 DataFrame의 연접
  - 이름없는 Series와 DataFrame의 연접

```
In [29]: s2 = pd.Series(['Z0', 'Z1', 'Z2', 'Z3'], name='Z')

In [30]: result = pd.concat([df1, s2], axis=1)

In [31]: result
In [32]: s3 = pd.Series(['*0', '*1', '*2'])

In [33]: result = pd.concat([df1, s3, s3, s3], axis=1)

In [34]: result
```

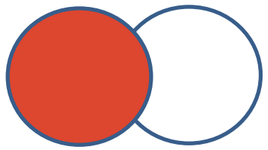
DB타입의 DataFrame 또는 Series를 합치고 붙이기

■ merge() 함수

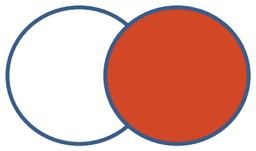
● 연산 방법



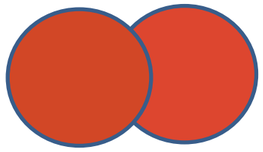
INNER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN



FULL JOIN

● 2개의 연산 대상 객체 df1, df2 생성

```
In [46]: df1 = pd.DataFrame({‘key’: [‘A’, ‘B’, ‘C’, ‘D’],
                              ‘value’: np.random.randn(4)})

In [47]: df2 = pd.DataFrame({‘key’: [‘B’, ‘D’, ‘D’, ‘E’],
                              ‘value’: np.random.randn(4)})
In [48]: pd.merge(df1, df2, on=’key’)
```

● SQL 신택스 - df1과 df2의 INNER JOIN

```
SELECT *
FROM df1
INNER JOIN df2
  ON df1.key = df2.key;
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- LEFT OUTER JOIN 연산

```
SELECT *
FROM df1
LEFT OUTER JOIN df2
  ON df1.key = df2.key;
```

```
In [51]: pd.merge(df1, df2, on='key', how='left')
```

- RIGHT JOIN 연산

```
SELECT *
FROM df1
RIGHT OUTER JOIN df2
  ON df1.key = df2.key;
```

```
In [52]: pd.merge(df1, df2, on='key', how='right')
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- FULL JOIN 연산

```
SELECT *
FROM df1
FULL OUTER JOIN df2
  ON df1.key = df2.key;
```

```
In [53]: pd.merge(df1, df2, on='key', how='outer')
```

- merge()의 주요 사항
  - key1, key2 기준의 교집합

```
In [54]: left = pd.DataFrame({'key1': ['Z0', 'Z0', 'Z1', 'Z2'],
                              'key2': ['ZO', 'Z1', 'Z0', 'Z1'],
                              'A': ['A0', 'A1', 'A2', 'A3'],
                              'B': ['B0', 'B1', 'B2', 'B3']})

In [55]: right = pd.DataFrame({'key1': ['Z0', 'Z1', 'Z1', 'Z2'],
                               'key2': ['ZO', 'Z0', 'Z0', 'Z0'],
                               'C': ['C0', 'C1', 'C2', 'C3'],
                               'D': ['D0', 'D1', 'D2', 'D3']})

In [56]: result = pd.merge(left, right, on=['key1', 'key2'])

In [57]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- merge()에서 인수 how

합치는 방법	SQL join Name	내용 설명
left	LEFT OUTER JOIN	왼쪽 프레임의 keys 사용
right	RIGHT OUTER JOIN	오른쪽 프레임의 keys 사용
outer	FULL OUTER JOIN	양쪽 프레임의 keys의 합집합 사용
inner	INNER JOIN	양쪽 프레임의 keys의 교집합 사용

- how='left'인 LEFT OUTER JOIN

```
In [58]: result = pd.merge(left, right, how='left', on=['key1', 'key2'])

In [59]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- how='right'인 RIGHT OUTER JOIN

```
In [60]: result = pd.merge(left, right, how='right',
                           on=['key1', 'key2'])

In [61]: result
```

- how='outer'인 FULL OUTER JOIN

```
In [62]: result = pd.merge(left, right, how='outer',
                           on=['key1', 'key2'])

In [63]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

■ ㅁㄴㅇㅁㅇ

- 2개의 다른 인덱스를 갖는 df를 결합하기

```
In [64]: left = pd.DataFrame({‘A’: [‘A0’, ‘A1’, ‘A2’],
                              ‘B’: [‘B0’, ‘B1’, ‘B2’]},
                             index=[‘Z0’, ‘Z1’, ‘Z2’])

In [65]: right = pd.DataFrame({‘C’: [‘C0’, ‘C2’, ‘C3’],
                               ‘D’: [‘D0’, ‘D2’, ‘D3’]},
                              index=[‘Z0’, ‘Z2’, ‘Z3’])

In [66]: result = left.join(right)

In [67]: result
```

- 합집합 범위인 how=‘outer’

```
In [68]: result = left.join(right, how=’outer’)

In [69]: result
```

- 교집합 범위인 how=‘inner’

```
In [70]: result = left.join(right, how=’inner’)

In [71]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- join() 메소드 – 인수 on

```
In [72]: left = pd.DataFrame({‘A’: [‘A0’, ‘A1’, ‘A2’, ‘A3’],
                              ‘B’: [‘B0’, ‘B1’, ‘B2’, ‘B3’],
                              ‘key’: [‘Z0’, ‘Z1’, ‘Z0’, ‘Z1’]})

In [73]: right = pd.DataFrame({‘C’: [‘C0’, ‘C1’],
                               ‘D’: [‘D0’, ‘D1’]},
                               index=[‘Z0’, ‘Z1’])

In [74]: result = left.join(right, on=’key’)

In [75]: result
```

- MultiIndex의 객체 합치기

```
In [77]: left = pd.DataFrame({‘A’: [‘A0’, ‘A1’, ‘A2’, ‘A3’],
                              ‘B’: [‘B0’, ‘B1’, ‘B2’, ‘B3’],
                              ‘key1’: [‘Z0’, ‘Z0’, ‘Z1’, ‘Z2’],
                              ‘key2’: [‘Z0’, ‘Z1’, ‘Z0’, ‘Z1’]})

In [78]: ind = pd.MultiIndex.from_tuples([(‘Z0’, ‘Z0’), (‘Z1’, ‘Z0’),
                                          (‘Z2’, ‘Z0’), (‘Z2’, ‘Z1’)])

In [79]: right = pd.DataFrame({‘C’: [‘C0’, ‘C1’, ‘C2’, ‘C3’],
                               ‘D’: [‘D0’, ‘D1’, ‘D2’, ‘D3’]}, index=ind)

In [80]: result = left.join(right, on=[’key1’, ‘key2’])

In [81]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- MultiIndex를 가지는 df1과 하나의 인덱스를 가지는 df2를 합치기

```
In [82]: left = pd.DataFrame({‘A’: [‘A0’, ‘A1’, ‘A2’], ‘B’: [‘B0’, ‘B1’, ‘B2’]}, index=pd.Index([‘ZO’, ’Z1’, ’Z2’], name=’key’))

In [83]: ind = pd.MultiIndex.from_tuples([(‘Z0’, ‘Y0’), (‘Z1’, ‘Y1’), (‘Z2’, ‘Y2’), (‘Z2’,‘Y3’)], names=[‘key’, ‘Y’])

In [84]: right = pd.DataFrame({‘C’: [‘C0’, ‘C1’, ‘C2’, ‘C3’],
                               ‘D’: [‘D0’, ‘D1’, ‘D2’, ‘D3’]}, index=ind)

In [85]: result = left.join(right, how=’inner’)

In [86]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- MultiIndex를 가지는 df1과 df2를 합치기

```
In [88]: l_ind = pd.MultiIndex.from_product(
                                        [list('abc'), list('xy'), [1, 2]],
                                        names=['abc', 'xy', 'num'])

In [89]: left = pd.DataFrame({'z1': range(12)}, index=l_ind)

In [90]: r_ind = pd.MultiIndex.from_product([list('abc'), list('xy')],
                                            names=['abc', 'xy'])

In [91]: right = pd.DataFrame({'z2': [100 * i for i in range(1, 7)]},
                              index=r_ind)
In [92]: left                 In [93]: right
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- MultiIndex를 가지는 df1과 df2를 합치기 : how='inner'
- 열과 인덱스 레벨의 조합으로 합치기

```
In [94]: left.join(right, on=['abc', 'xy'], how='inner')
In [101]: l_ind = pd.Index(['Z0', 'Z0', 'Z1', 'Z2'], name='key1')

In [102]: left = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                               'B': ['B0', 'B1', 'B2', 'B3],
                          'key2': ['Z0', 'Z1', 'Z0', 'Z1']}, index=l_ind)

In [103]: r_ind = pd.Index(['Z0', 'Z1', 'Z2', 'Z2'], name='key1')

In [104]: right = pd.DataFrame({'C': ['C0', 'C1', 'C2', 'C3'],
                                'D': ['D0', 'D1', 'D2', 'D3'],
                                'key2': ['Z0', 'Z0', 'Z0', 'Z1']}, index=r_ind)

In [105]: result = left.merge(right, on=['key1', 'key2'])

In [106]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

■ 열의 중복 처리

- 열의 중복을 방지하기 위한 접미사

```
In [107]: left = pd.DataFrame({‘z’: [‘Z0’, ‘Z1’, ‘Z2’], ‘v’: [1, 2, 3]})

In [108]: right = pd.DataFraem({‘z’: [‘Z0’, ‘Z0’, ‘Z3’], ‘v’: [4, 5, 6]})

In [109]: result = pd.merge(left, right, on=’z’)

In [110]: result1 = pd.merge(left, right, on=’z’,
                             suffixes=[‘_l’, ‘_r’])

In [111]: result                    In [112]: result1
```

- df.join() - lsuffix와 rsuffix의 적용

```
In [113]: left = left.set_index(‘v’)

In [114]: right = right.set_index(‘v’)

In [115]: result = left.join(right, lsuffix=’_l’, rsuffix=’_r’)

In [116]: result
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

- Series 또는 DataFrame 열 내에서 값들을 합치기
  - combine_first() 메소드

```
In [117]: df1 = pd.DataFrame([[np.nan, 3., 5.], [-4.6, np.nan, np.nan], [np.nan, 7., np.nan]])

In [118]: df2 = pd.DataFrame([[-2.6, np.nan, -8.2], [-5., 1.6, 4]], index=[1, 2])

In [119]: result = df1. combine_first(df2)          In [120]: result1 = df2.combine_first(df1)

In [121]: result                                    In [122]: result1
```

DB타입의 DataFrame 또는 Series를 합치고 붙이기

■ update() 메소드

```
In [123]: df1.update(df2)

In [124]: df1
```

데이터의 재형성(reshaping)

■ 학교 체육 종목의 DataFrame 객체들을 피벗팅(pivoting)

- 체육 종목 참여 내역의 df객체 생성
- pivot() 메소드

```
In [125]: data = {‘name’: [‘haena’, ‘naeun’, ‘una’, ‘bum’, ‘suho’],
                  ‘type’: [‘tennis’, ‘tennis’, ‘swim’, ‘swim’, ‘tennis’],
                  ‘records’: [‘A’, ‘B’, ‘C’, ‘A’, ‘B’],
                  ‘sex’: [‘F’, ‘F’, ‘F’, ‘M’, ‘M’],
                  ‘period’: [3, 3, 1, 5, 2]}

In [126]: df = pd.DataFrame(data)

In [127]: df
```

```
In [128]: dfp = df.pivot(index=’name’, columns=’type’,
                         values=[’records’, ‘sex’])

In [129]: dfp
```

데이터의 재형성(reshaping)

- 피벗 테이블
  - pivot_table() 메소드 – 인수 aggfunc

```
In [130]: dfp = df.pivot_table(index='type', columns='records',
                               values='period', aggfunc=np.max)

In [131]: dfp
```

데이터의 재형성(reshaping)

- pivot_table() 실행을 위한 df 생성
- pivot_table() 실행

```
In [132]: import datetime

In [133]: df = pd.DataFrame({‘A’: [‘one’,’one’,’two’,’three’] * 6,
                             ‘B’: [‘x’, ‘y’, ‘w’] * 8,
                             ‘C’: [‘ha’, ‘ha’, ‘ha’, ‘hi’, ‘hi’, ‘hi’] * 4,
                             ‘D’: np.arange(24),
                             ‘E’: [datetime.datetime(2020,i,1) for i in range(1,13)]
                               + [datetime.datetime(2020,i,15) for i in range(1,13)]})

In [134]: df
In [135]: pd.pivot_table(df, values=’D’,
                         index=[‘A’, ’B’], columns=‘C’)
```

데이터의 재형성(reshaping)

- pivot_table() – 인수 aggfunc
- NaN을 빈공간으로 처리하기

```
In [136]: pd.pivot_table(df, values='D', index=['B'],
                         columns=['A', 'C'], aggfunc=np.sum)

In [137]: df_pt = pd.pivot_table(df, values='D',
                       index=['B'], columns=['A', 'C'],
                       aggfunc=np.sum)

In [138]: str_df = df_pt.to_string(na_rep='')

In [139]: print(str_df)
          A   one          three          two
          C    ha     hi     ha     hi     ha     hi
          B
          w  28.0  22.0           34.0  16.0
          x  12.0  30.0           18.0  24.0
          y  14.0  20.0   26.0                  32.0
```

데이터의 재형성(reshaping)

- 교차 분석(cross tabulations)
  - 교차분석 테이블 생성하기
  - 2개 Series로 도수 테이블 생성

```
In [140]: ha, hi, top, down, one, two = ‘ha’, ‘hi’, ‘top’, ‘down’, ‘one’, ‘two’

In [141]: a = np.array([ha, ha, hi, hi, ha, ha], dtype=object)

In [142]: b = np.array([one, one, two, one, two, one], dtype=object)

In [143]: c = np.array([top,top,down,top,top,down], dtype=object)

In [144]: pd.crosstab(a,[b,c], rownames=[‘a’], colnames=[‘b’,‘c’])
In [145]: df = pd.DataFrame({‘A’: [1,2,2,2,2], ‘B’: [3,3,7,7,7],
                             ‘C’: [1, 1, np.nan, 1, 1]})

In [146]: df                    In [147]: pd.crosstab(df.A, DF.B)
```

데이터의 재형성(reshaping)

- crosstab() 함수 – 인수 normalize

```
In [151]: pd.crosstab(df.A, df.B, normalize=True)
```

- 행 또는 열 내에서 값들의 정규화

```
In [152]: pd.crosstab(df.A, df.B, normalize='columns')
```

- crosstab() 함수 – 인수 aggfunc

```
In [153]: pd.crosstab(df.A, df.B, values=df.C, aggfunc
=np.sum)
```

- crosstab() 함수 – 인수 margins

```
In [154]: pd.crosstab(df.A, df.B, values=df.C, aggfunc=np.sum
, normalize=True, margins=True)
```

데이터의 재형성(reshaping)

■ 지표(indicator)/더미(dummy) 변수의 계산

● get_dummies() 함수

```
In [155]: df = pd.DataFrame({‘key’: list(‘bbacab’),
‘data1’: range(6)})

In [156]: pd.get_dummies(df[‘key’])
```

● dummies 객체 생성 – 접두사 ‘key’

```
In [157]: dummies = pd.get_dummies(df[‘key’], prefix=’key’)

In [158]: dummies
```

데이터의 재형성(reshaping)

- join() 메소드 – dummies객체를 덧붙이기

```
In [159]: df[[‘data1’]].join(dummies)
```

데이터의 재형성(reshaping)

- 이산함수 cut()와 함께 사용되는 get_dummies() 함수

```
In [160]: val = np.random.randn(7)

In [161]: val

In [162]: bins = [0, 0.2, 0.4, 0.6, 0.8, 1]

In [163]: pd.get_dummies(pd.cut(val, bins))
```

데이터의 재형성(reshaping)

■ stack과 unstack 메소드에 의한 재형성

●stack

df

		A	B
first	second		
ha	one	1	2
	two	3	4
hi	one	5	6
	two	7	8

MultiIndex

stacked = df.stack( )

first	second		
ha	one	A	1
		B	2
	two	A	3
		B	4
hi	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

●unstack

stacked

first	second		
ha	one	A	1
		B	2
	two	A	3
		B	4
hi	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

stacked.unstack( )

		A	B
first	second		
ha	one	1	2
	two	3	4
hi	one	5	6
	two	7	8

MultiIndex

데이터의 재형성(reshaping)

●멀티 인덱스를 갖는 객체 df1 생성

```
In [164]: tup = list(zip(*[['ha', 'ha', 'hi', 'hi', 'ho', 'ho', 'hu', 'hu'],
                           ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']])

In [165]: ind = pd.MultiIndex.from_tuples(tup,names=['1st','2nd'])

In [166]: df = pd.DataFrame(np.random.randn(8, 2), index=ind, columns=['A', 'B'])

In [167]: df1 = df[:4]

In [168]: df1
```

데이터의 재형성(reshaping)

- 열의 가장 낮은 레벨이 스택 처리
- 객체 stacke의 unstack

```
In [169]: stacked = df1.stack()

In [170]: stacked
In [171]: type(stacked)
In [172]: stacked.unstack()
```

데이터의 재형성(reshaping)

●stacked.unstack(1) 또는 stacked.unstack('2nd')

```
In [173]: stacked.unstack(1)
```

stacked

1st	2nd		
ha	one	A	1
		B	2
	two	A	3
		B	4
hi	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

stacked.unstack(1) 또는
stacked.unstack('2nd')

	2nd	one	two
1st			
ha	A	1	3
	B	2	4
hi	A	5	7
	B	6	8

MultiIndex

●stacked.unstack(0) 또는 stacked.unstack('1st')

```
In [174]: stacked.unstack('1st')
```

stacked

1st	2nd		
ha	one	A	1
		B	2
	two	A	3
		B	4
hi	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

stacked.unstack(0) 또는
stacked.unstack('1st')

	1st	ha	hi
2nd			
one	A	1	5
	B	2	6
two	A	3	7
	B	4	8

MultiIndex

데이터의 재형성(reshaping)

- stack/unstack의 index level을 순서정렬

```
In [175]: ind = pd.MultiIndex.from_product([[2, 1], ['a', 'b']])

In [176]: df = pd.DataFrame(np.random.randn(4), index=ind, columns=['top'])

In [177]: df                                          In [178]: df.unstack.stack()
In [179]: all(df.unstack().stack() == df.sort_index())

In [180]: all(df.unstack().stack() == df)
```

데이터의 재형성(reshaping)

- Melt에 의한 재형성
  - 한 개 이상의 열들이 식별자 변수로 포맷

df

0	first	last	age	score
1	Haena	Kang	30	100
2	Suho	Chae	18	85

df.melt(id_vars=['first', 'last'])

	first	last	variable	value
0	Haena	Kang	age	30
1	Suho	Chae	age	18
2	Haena	Kang	score	100
3	Suho	Chae	score	85

```
In [181]: df = pd.DataFrame({'first': ['Haena', 'Suho'], 'last': ['Kang', 'Chae'], 'age': [30, 18], 'score': [100, 85]})

In [182]: df

In [183]: df.melt(id_vars=['first', 'last'])
In [184]: df.melt(id_vars=['first', 'last'], var_name='personal')
```

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

파이썬 정규표현식(Regular Expression)

- 정규표현식이란 ?

- REs, regexes, regex pattern
- 대상이 되는 문자열에 정규표현식의 룰을 매칭하여 원하는 결과인 문자열의 집합을 도출
- re모듈을 임포팅
- 찾기 패턴을 정의하는 문자들의 시퀀스
- 패턴은 문자열을 찾는 알고리즘

■ 정규표현식 신택스(Syntax)

- 정규 표현식 용법

- 특수문자나 특별한 형식을 위해 '₩' 사용
- r"₩n"는 단지 '₩'와 'n'인 2개의 문자
- '₩n'는 새로운 줄을 의미
- '^'는 반대되는 의미의 complement
- white space : 여백, 탭, 새로운 줄, cr, ff, vertical tab

파이썬 정규표현식(Regular Expression)

● 정규표현식 문자매칭 특수문자

특수문자	기능 설명
₩d	어떤 숫자에 매칭한다. 클래스 [0-9]와 같다.
₩D	숫자가 아닌 문자에 매칭한다. 클래스 [^0-9]와 같다.
₩s	화이트스페이스(whitespace) 문자에 매칭한다. 클래스 [₩t₩n₩r₩f₩v]와 같다.
₩S	화이트스페이스 문자가 아닌 문자에 매칭한다. 클래스 [^₩t₩n₩r₩f₩v]와 같다.
₩w	영숫자(alphanumeric) 문자에 매칭한다. 클래스 [a-zA-Z0-9_]와 같다.
₩W	영숫자 문자가 아닌 문자에 매칭한다. [^a-zA-Z0-9_]와 같다.
₩number	같은 수의 그룹들의 내용을 매칭한다. 예를 들면 (.+) ₩1는 ’ 77 77’ 또는 ‘love love’로 매칭되나 ‘lovelove’로 매칭될 수는 없다.
₩A	문자열 시작에서만 매칭된다.
₩b	word boundary로서 빈 문자열을 매칭하나 단어의 처음과 끝에서만 매칭한다. 예를 들면 r’₩bha₩b’는 ‘ha’, ‘ha.’, ‘hi ha ho’를 매칭하나 ‘hahi’와 ‘ha7’를 매칭 안한다. 특수문자 ‘.’는 word bounday로 이해된다.
₩B	word boundary가 아닌 빈 문자열에 매칭하나 단어의 처음이나 끝에 있지 않을 때만 적용된다. r’py₩B’는 ‘python’, ‘py3’, ‘py2’에 매칭하나 ‘py’, ‘py.’, 또는 ‘py!’에는 매칭하지 않는다. ₩b와는 반대이다.
₩Z	문자열의 끝에서만 매칭한다.

파이썬 정규표현식(Regular Expression)

●반복 표기 또는 문자열의 메타 문자

특수문자	기능 설명
+	왼쪽에 대해 1회 이상의 패턴을 발생한다. ‘i+’는 1개 이상의 ‘i’이다.
*	왼쪽에 대해 0회 이상의 패턴을 발생한다.
?	왼쪽에 대해 0 또는 1회의 패턴 발생을 매칭한다.
₩	escape문자를 나타내는 메타 문자
{m}	이전 RE에 m번 복사하여 적용한다. a{6}은 6개의 ‘a’ 문자에 매칭한다.
{m, n}	연산결과의 RE가 이전 RE에 m부터 n까지 반복을 매칭한다. a{3, 5}는 3개 부터 5개 까지의 ‘a’문자에 매칭한다.
{m, n}?	연산결과의 RE가 이전 RE에 m부터 n까지 가능한 적게 반복 매칭한다. 예를 들면 6문자 ‘aaaaaa’에 대해 a{3, 5}는 5개의 ‘a’에 매칭하고 a{3, 5}?는 3개의 문자만에 매칭한다.
.	(Dot.) 디폴트 모드에서 새로운 줄을 제외하고 어떤 문자에도 매칭되며 DOTALL flag가 명시되면 새로운 줄을 포함 어떤 문자에도 매칭된다.
^	(Caret) 문자열 시작에 매칭하고 MULTILINE 모드에서 각각이 새로운 줄 뒤에 직접 매칭한다.
$	문자열의 끝이나 문자열 끝에서 새로운 줄 전에 매칭한다. ha는 ‘ha’ 및 ‘hahi’에 매칭하나 ha$는 ‘ha’만을 매칭한다.
|	A나 B는 임의의 REs일 수 있는 A|B는 A나 B에 매칭하는 정규표현식을 생성한다. 즉 either or가 된다.

파이썬 정규표현식(Regular Expression)

- []는 문자들의 집합을 나타내는 문자클래스라고하며 set에서 다음과 같은 특성으로 적용
  - 문자들은 개별적으로 나열될 수 있다. 예를 들면 [you]는 'y', 'o' 또는 'u'를 매칭
  - 문자의 범위는 2개의 문자를 '-'로 분리시켜 표시된다. [a-z]는 소문자 ASCII 문자를 매칭하고
  - [0-5][0-9]는 00부터 59까지의 모든 2개의 숫자를 매칭.
  - 집합 내에서 특수문자들의 의미를 잃음. [(+*)]는 문자 '(', '+', '*', ')'를 매칭.
  - ₩w 또는 ₩S와 같은 문자 클래스는 집합 내부에서 또한 허용.
  - 범위 내에 없는 문자들은 집합을 여집합으로 연산함으로써(complementing) 매칭. 집합의 첫 번째
  - 문자가 '^'이면 집합에 없는 모든 문자가 매칭. [^7]는 '7'을 제외한 모든 문자에 매칭.
  - 집합에서 +, *, ., |, (), $, {}는 특별한 의미를 가지지 않는다.
  - 집합 내에 문자 그대로의 ']'를 매칭하기 위해 백슬래시 뒤에 두거나 집합의 처음에 위치시킨다. 예를 들면 [()[₩]{}]와 []()[{}]는 둘 다 괄호에 매칭한다.

```
txt = "You + I am smart + (special)"
lst = re.findall('[+,()]', txt)
lst
```

파이썬 정규표현식(Regular Expression)

- ()에서의 그룹화 패턴 적용

  - (...)는 괄호 안의 어떤 정규표현식이라도 매칭하며 그룹의 처음과 끝을 의미
  - (?...)는 확장 표기로서 '?' 뒤의 첫 번째 문자는 구문의 의미와 신택스가 무엇인지를 결정
  - (?aiLmsux)는 집합 'a', 'i', 'L', 'm', 's', 'u', 'x'로 부터의 1개 이상의 철자를 의미. 그룹은 빈 문자를 매칭하고 철자는 전체적인 정규표현식에 대해 re.A(ASCII-only matching), re.I(ignore case), re.L(locale dependent), re.M(multi-line), re.S(dot matches all), re.U(Unicode matching) 및 re.X(verbose)와 같은 해당하는 flags를 설정.
  - (?:...)는 그룹에 의해 매칭된 문자열은 패턴에서 매칭한 이후에 복구될 수 없거나 이후에 참조될 수 없다.
  - (?aiLmsux-imsx:...)는 'i', 'm', 's', 'x'로부터 1개 이상의 철자 앞에 있는 '-'가 있으며 선택적으로 '-'앞에 있는 집합 'a', 'i', 'L', 'm', 's', 'u', 'x'으로부터의 0개 이상의 철자를 의미한다. 철자들은 해당하는 표현식의 부분에 대해서 해당하는 flags를 설정하거나 제거한다. 철자 'a', 'L' 및 'u'는 인라인 flags로서 사용될 때 서로 배타적이고 '-'뒤에 있지 않으며 조합을 이룰 수 없다.
  - (?P<name>...)는 정규적인 괄호와 비슷하지만 그룹에 의해 매칭된 문자열은 그룹을 상징하는 이름인 name을 경유하여 접근할 수 있다. 그룹 이름은 유효한 파이썬 확인자이고 각각의 그룹 이름은 정규표현식 내에서 한번만 정의되어야 한다. 상징하는 그룹은 마치 그룹이 이름이 없는 것처럼 또한 숫자가 매겨진 그룹이다. 이름이 있는 그룹은 3가지와 관련하여 참조될 수 있다. 만일 패턴이 ("?P<quote>['"]).*?(?P=quote)이라면 즉 작은 따옴표나 큰 따옴표로 인용된 문자열을 매칭하는 경우 다음과 같다.

파이썬 정규표현식(Regular Expression)

“quote”를 그룹하기 위한 참조 관련	참조 방법
같은 패턴 자체에서	(?P=quote) ₩1
매치 객체 m을 처리할 때	m.group(‘quote’) m.end(‘quote’) 등
re.sub()의 repl 인수로 전달된 문자열에서	₩g<quote> ₩g<1> ₩1

- (?P=name)는 이름이 있는 그룹을 역참조하란 의미로서 name으로 이름 붙여진 초기 그룹에 의해 매칭되었던 어떤 텍스트에도 매칭한다. 역참조(backreference)란 어떤 캡쳐그룹에 의해 이전에 매칭된 것과 같은 텍스트를 매칭하는 것을 말한다. 예로서 ([a-c])d₩1e₩1는 adaea, bdbeb 및 cdcec와 매칭한다. 괄호안의 캡쳐 그룹인 [a-c]의 이름이 1이고 ₩1는 그룹1을 역참조하라는 의미이다.
- (?#...)는 주석이며 괄호의 내용은 단지 무시된다.

파이썬 정규표현식(Regular Expression)

- (?=...)는 ...가 다음에 매칭되면 매칭하지만 문자열의 어떤것도 소모하지 않는다. 예를 들면 yaho(?=suho)는 'suho'가 뒤에 있기만 한다면 'yaho'에 매칭한다.
- (?!...)는 ...가 다음에 매칭하지 않으면 매칭한다. yaho(?!suho)는 'suho'가 뒤에 없으면 'yaho'에 매칭한다.
- (?<=...)는 현재 위치에서 끝나는 ...에 대한 매칭이 문자열에서 현재의 위치에 선행한다면 매칭한다. (?<=abc)def는 'abcdef'에서 매칭을 구한다.

- ()에서의 그룹화 패턴 적용 예제

```
import re
m = re.search('(?<=abc)def', 'abcdef')
m.group()
'def'
```

```
m = re.search('(?<=-)₩w+', 'spam-egg')
m.group(0)
'egg'
```

파이썬 정규표현식(Regular Expression)

- (?<!...)는 문자열에서 현재의 위치가 ...에 대한 매칭보다 선행하지 않으면 매칭한다.
- (?(id/name)yes-pattern|no-pattern)는 주어진 id나 name이 존재하면 yes-pattern으로 매칭하고 그렇지 않으면 no-pattern으로 매칭한다.
- (₩d*)(?P<haena>[a-d]+)|(₩w*)(?P<haena>[0-7]+)는 다음과 같은 의미를 가진다.
  + (₩d*)는 group1이다.
  + (?P<haena>[a-d]+)는 'haena'라 불리는 group2이다.
  + (₩w*)는 분기이므로 group2이다.
  + (?P<haena>[0-7]+)는 또한 'haena'라 불리므로 또한 group2이다.

■ re모듈의 주요 내용

❶ re.compile(regex_pattern, flags=0)

- 정규표현식 패턴을 정규표현식 객체로 컴파일
- 컴파일된 객체는 Pattern으로 표현
- 연산결과를 인수에 flags값을 명시하여 변경

파이썬 정규표현식(Regular Expression)

- flag의 종류 및 기능

flag의 종류	기능 설명
re.I re.IGNORECASE	대소문자에 관계없이 매칭을 실행한다. [A-Z]와 같은 표현식도 소문자에 매칭한다.
re.L re.LOCALE	₩w ₩W, ₩b, ₩B와 대소문자에 관계없는 매칭이 현재의 local에 의존하도록 한다.
re.M re.MULTILINE	명시되면 패턴문자 '^' 는 문자열의 시작과 각각의 줄의 시작에 매칭한다. 패턴문자 '$' 는 문자열 끝과 각각의 줄의 끝에 매칭한다.
re.S re.DOTALL	특수문자 '.' 를 새로운 줄을 포함하여 어떤 문자에라도 매칭하도록 한다. 이 플래그가 없으면 '.' 는 새로운 줄을 제외하고 매칭한다.
re.X re.VERBOSE	시각적으로 패턴의 논리적 부분을 분리하고 코멘트를 추가하도록 함으로써 정규표현식을 더 가독성있게 만든다.

파이썬 정규표현식(Regular Expression)

- 함수 compile()의 연산 시퀀스

```
pattern = re.compile(regex_pattern)
result = pattern.match(string)
```

2. re.search(regex_pattern, string, flags=0)
   - 정규표현식 regex_pattern이 매칭하는 첫 번째 위치의 문자열을 탐색
   - 문자열을 탐색 후 이에 상응하는 match객체 반환

```
txt = “The azalea is beautiful”
mat = re.search(‘^The.*ful$’, txt)
mat
<re.Match object; span=(0, 23), match=’The azalea is beautiful’>
```

3. re.match(regex_pattern, string, flags=0)
   - string 시작에서 0개 이상의 문자가 regex_pattern에 매칭하면 이에 상응하는 match를 반환
   - 매칭하지 않으면 None을 반환

파이썬 정규표현식(Regular Expression)

4. re.split(regex_pattern, string, maxsplit=0, flags=0)
   - 문자열 strin을 패턴인 regex_pattern의 발생으로 분리시킴
   - 분리되는 리스트를 반환

```
txt = “The azalea in Korea”
lst = re.split(‘₩s’, txt)
lst
type(lst)
lst1 = re.split(‘₩s’, txt, 2)
lst1
```

```
re.split(’\W+’, ‘Ha, hi, ho’)
re.split(’\W+’, ‘Ha, hi, ho.’)
re.split(’(\W+)’, ‘Ha, hi, ho.’)
re.split(’\W+’, ‘Ha, hi, ho.’, 1)
re.split(‘[a-f]+’, ‘0a3B7’, flags=re.IGNORECASE)
```

파이썬 정규표현식(Regular Expression)

5 re.findall(regex_pattern, string, flags=0)

- 문자열에서 regex_pattern의 모든 겹치지 않는 매칭을 문자열이 리스트로 반환

```
txt = “The azalea in Korea”
lst = re.findall(‘a’, txt)
lst
```

6 re.sub(regex_pattern, repl, string, count=0, flags=0)

- string에서 가장 왼쪽의 겹치지 않는 regex_patter의 발생을 대치하는 repl로 위치시킴.
- repl로 위치시킴으로써 얻어진 문자열을 반환.

```
txt = “The azalea in Korea”
str = re.sub(‘₩s’, ‘_’, txt)
str
str1 = re.sub(‘₩s’, ‘_’, txt, 2)
```

파이썬 정규표현식(Regular Expression)

- 정규표현식(Regular Expression) 객체

  정규표현식의 적용
  - re모듈의 compile메소드를 통한 컴파일된 정규표현식 객체로 표기되는 Pattern을 적용
  - Pattern 객체를 통하는 방법 - Pattern.search(string[, pos[, endpos]])
    - 첫 번째 위치를 문자열에서 찾기 위해 스캔하고 이에 상응하는 match 객체를 반환
    - 문자열에서 어떤 위치도 패턴에 매칭하지 않으면 None
  - 매개변수 pos는 디폴트가 0이고 탐색이 시작되는 문자열에서의 index
  - 매개변수 endpos는 찾는 문자열이 얼마나 떨어져 있는지를 제한

```
pattern = re.compile('e')
pattern.search('haena')
pattern.search('haena', 2)
pattern.search('haena', 3)
```

파이썬 정규표현식(Regular Expression)

- Pattern.match(string[, pos[, endpos]])
  - 문자열의 시작에서 0개 이상의 문자가 정규표현식에 매칭하면 상응하는 match객체를 반환
  - 문자열이 패턴과 매칭하지 않으면 None을 반환
  - 매개변수 pos는 디폴트가 0이고 탐색이 시작되는 문자열에서의 index
    - 매개변수 endpos는 찾는 문자열이 얼마나 떨어져 있는지를 제한
- Pattern.split(), Pattern.findall, Pattern.sub()는 컴파일된 패턴을 사용할 때의 함수와 같음.

```
pattern = re.compile(‘u’)
pattern.match(‘naeun’)
pattern.match(‘naeun’, 3)
pattern.match(‘haena’, 2)
```

파이썬 정규표현식(Regular Expression)

■ Match 객체

- span(), sgring 및 group()

```
txt = “The azalea in Korea”
mat = re.search(r‘₩bK₩w+’, txt)
mat
mat.span()
mat.string
mat.group()
```

- 이멜 주소의 추출

```
txt = ‘smart jchae-justin@google.com expert’
mat = re.search(r‘₩w+@₩w+’, txt)
mat.group()
mat1 = re.search(r’[₩w.−]+@[₩w.−]+’, txt)
mat1.group()
```

파이썬 정규표현식(Regular Expression)

- group() 메소드의 적용

```
txt2 = ‘smart jchae-justin79@google.com expert’
mat2 = re.search(r’[a-zA-Z0-9.+-]+@[a-zA-Z]+ ₩.com’, txt2)
mat2.group()
mat3 = re.search(r’([₩w.−]+)@([₩w.−]+)’, txt)
mat3.group()
mat2.group(1)
mat2.group(2)
```

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

텍스트 데이터의 작업

● 문자열 메소드 : str()의 메소드

메서드	기능 설명
lower()	배열의 strings를 소문자로 변경
upper()	배열의 strings를 대문자로 변경
len()	배열의 각 문자열의 길이를 계산
strip()	배열의 각 문자열로부터 공백(newlines 포함)을 제거
lstrip()	배열의 각 문자열의 왼쪽부터 공백을 제거
rstrip()	배열의 각 문자열의 오른쪽부터 공백을 제거
replace()	Series/Index에서 pattern/regex의 적용된 것으로 대치
split()	주어진 정규표현식의 패턴이나 구분자로 각 문자열을 분리
cat()	주어진 구분자로 문자열 배열을 연접한다.
contains()	각 문자열이 정규표현식의 주어진 패턴을 포함하는지를 불리언 배열로 반환
count()	각 문자열에서 패턴이 일치하는 개수를 계산
findall()	패턴이나 정규표현식의 모든 발생의 경우를 구한다.
get()	배열의 요소에 있는 lists, tuples 또는 strings로부터 요소를 추출
extract()	전달된 정규표현식을 사용하여 각 문자열에서 그룹들을 찾는다.
extractall()	정규표현식의 패턴에 모두 매칭하는 그룹들을 찾는다.
endswith()	각 문자열이 전달된 패턴으로 끝나는지를 나타내는 불리언 배열
startswith()	각 문자열이 전달된 패턴으로 시작하는지를 나타내는 불리언 배열

텍스트 데이터의 작업

str 메소드

- 클래스 pandas.Series와 pandas.Index의 메소드
- pandas.core.strings.StringMethods의 별칭
- str 메소드는 문자열을 처리하는 메소드들을 가짐
- 호출할 수 없는 메소드로 소괄호를 생략

```
In [185]: ser = pd.Series([‘Suho’, ‘AA’, np.nan, ‘rabbit’])
In [186]: type(ser.str)
In [187]: ser.str.lower()                    In [188]: ser.str.upper()
In [189]: ser.str.len()
```

텍스트의 빈공간 제거

```
In [190]: ind = pd.Index([‘  ha’, ‘hi  ‘, ‘  ho  ‘, ‘hu’])
In [191]: ind.str.strip()
In [192]: ind.str.lstrip()
In [193]: ind.str.rstrip()
```

텍스트 데이터의 작업

■ Index클래스에 str 메소드 적용 – df 생성

```
In [194]: df = pd.DataFrame(np.random.randn(2, 2), columns=[‘ Column A ’, ‘ Column B ‘], index=range(2))

In [195]: df

In [196]: df.columns
```

■ df columns 객체에 str 메소드 적용

```
In [197]: df.columns.str.strip()

In [198]: df.columns.str.lower()
```

텍스트 데이터의 작업

■ str 메소드의 replace() 메소드 적용

```
In [199]: df.columns = df.columns.str.strip().str.lower().str.replace(‘ ’, ‘_’)

In [200]: df
```

■ 문자열을 분리하고 대치하기

■ str 메소드의 split() 메소드 적용

```
In [201]: ser1 = pd.Series([‘ha_a_b’, ‘hi_c_d’, np.nan, ‘ho_e_f’])

In [202]: ser1.str.split(‘_’)
```

텍스트 데이터의 작업

■ split lists 요소에 접근하기

```
In [203]: ser1.str.split('_').str.get(1)
```

```
In [204]: ser1.str.split('_').str[1]
```

■ DataFrame 반환 : expand=True

```
In [205]: ser1.str.split('_', expand=True)
```

텍스트 데이터의 작업

●문자열 대치 :

```
In [206]: ser = pd.Series(['Suho', 'bAAa', np.nan, 'cute_dog'])

In [207]: ser
Out[207]: 0         Suho
          1         bAAa
          2          NaN
          3     cute-dog
          dtype: object

In [208]: ser.str.replace('^.a|dog', '***', case=False)
Out[208]: 0         Suho
          1        ***Aa
          2          NaN
          3    cute_***
          dtype: object
```

●str.replace()의 re.compile의 수용

```
In [209]: import re

In [210]: pattern = re.compile('^.a|dog',
                               flags=re.IGNORECASE)

In [211]: ser.str.replace(pattern, '***')
Out[211]: 0         Suho
          1        ***Aa
          2          NaN
          3    cute-***
          dtype: object
```

텍스트 데이터의 작업

- 연접(concatenation)
  - 단일 Series를 문자열로 연접

```
In [212]: ser = pd.Series(['ha', 'hi', 'ho'])     In [213]: ser.str.cat(sep=',')

In [214]: ser.str.cat()
```

- 연접시 손실값은 무시되고 별도 표기

```
In [215]: ser1 = pd.Series(['ha', np.nan, 'hi'])   In [216]: ser1.str.cat(sep=',')
In [217]: ser1.str.cat(sep=',', na_rep='*')
```

텍스트 데이터의 작업

- cat()에 전달되는 인수

```
In [218]: ser.str.cat([‘A’, ‘I’, ‘O’])
```

- cat() 인수 na_rep

```
In [219]: ser.str.cat(ser1)
In [220]: ser.str.cat(ser1, na_rep=’*’)
```

- concat() : axis=1, na_rep

```
In [221]: df = pd.concat([ser1, ser], axis=1)

In [222]: df
```

```
In [223]: ser.str.cat(df, na_rep=’*’)
```

텍스트 데이터의 작업

- cat() : 길이가 같지 않은 객체의 연접 – 인수 join

```
In [224]: ser2 = pd.Series([‘z’, ‘a’, ‘b’, ‘d’], index=[-1, 0, 1, 3])

In [225]: ser2                    In [226]: ser

In [227]: ser.str.cat(ser2, join=’left’, na_rep=’*’)
```

```
In [228]: ser.str.cat(ser2, join=’outer’, na_rep=’*’)
```

텍스트 데이터의 작업

■ str로 인덱스 처리하기

```
In [229]: ser = pd.Series(['Suho', 'AB', np.nan, 'rabbit', 'C'])

In [230]: ser.str[0]                                    In [231]: ser.str[1]
```

텍스트 데이터의 작업

■ 일기형식의 텍스트 데이터 처리

```
In [232]: day_plan = [“1st_seq: getting up at 05:45am”,
                      “2nd_seq: swimming from 06:00am to 07:00am”,
                      “3rd_seq: My morning food is American style”,
                      “4th_seq: Writing some proposal from 02:00pm to 06:00pm”,
                      “5th_seq: Arriving at JongGak at 07:00pm”,
                      “6th_seq: Fun with friends enjoying beer till 09:30pm”,
                      “7th_seq: My house at 10:30pm and sleeping by 12:00pm”]

In [233]: df = pd.DataFrame(day_plan, columns=[‘schedule’]

In [234]: df
```

텍스트 데이터의 작업

■ Split() 메소드 : 문자열을 리스트 형식으로 분리

```
In [235]: df[‘schedule’].str.split()
```

■ Split() 메소드 : 문자열의 수를 연산

```
In [236]: df[‘schedule’].str.split().str.len()
```

텍스트 데이터의 작업

■ 단어 'My'를 포함하는지를 확인

```
In [237]: df['schedule'].str.contains('My')
```

■ 각 문자열에서 숫자 개수 확인

```
In [238]: df['schedule'].str.count('₩d')
```

■ 모든 숫자 발생 내역

```
In [239]: df['schedule'].str.findall('₩d')
```

■ 정규표현식 패턴 매칭

```
In [240]: df['schedule'].str.findall((₩d₩d):(₩d₩d)')
```

텍스트 데이터의 작업

■ 패턴 적용 : ‘_seq’ 제거

```
In [241]: df[‘schedule’].str.replace(r‘(₩w+_seq₩b)’,
lambda x: x.groups()[0][0:3])
```

■ 시간 추출

```
In [242]: df[‘schedule’].str.extract(’(₩d₩d):(₩d₩d)’)
```

텍스트 데이터의 작업

■ 정규표현식 패턴 매칭 – 전체 시간 표기

```
In [243]: df['schedule'].str.extractall('((₩d?₩d):(₩d₩d) ?([ap]m))')
```

텍스트 데이터의 작업

■ 패턴 매칭 – 열을 시간 단위별로 표기

```
In [244]: dfx = df['schedule'].str.extractall('(?P<times>(?P<hr>₩d₩d):(?P<min>₩d₩d)?(?P<periods>[ap]m))')

In [245]: dfx
```

텍스트 데이터의 작업

■ 멀티 인덱스 객체 – 행과 열의 이름 변경

```
In [246]: dfx.index
In [247]: dfx.index = pd.MultiIndex(levels=[[‘one’, ‘two’, ‘three’, ‘four’, ‘five’, ‘six’], [‘1st’, ‘2nd’]], labels=
[[0,1,1,2,2,3,4,5,5], [0,0,1,0,1,0,0,0,1]], names=[‘step’, ‘match’])
```

텍스트 데이터의 작업

■ 최종 완성된 객체 dfx

```
In [248]: dfx
```

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

Mid-term Briefing

- 일정: 10월 26일 수요일 12~3시
- 시험 범위: 1주~7주차
- 시험 문제: 5문제 (코드형, LMS에 제출)
- 오픈북, 오픈노트, 오픈인터넷 (단, 카톡 포함한 SNS 금지)
- 시험 환경: 수정관 506호 H/W 실습실
- 미제출 시 0점처리

  → 시험 중간중간마다 제출 필요

Week 7 Outline

- Week 6 Review
- Data Concatenating, Reshaping
- Regular Expression
- Text Data Operation
- Mid-term Briefing
- Conclusion

Week 7 Key Takeaway

- 판다스 데이터 처리 관련 실습

  → 다양한 데이터 연정, Reshaping, 정규표현등 학습 및 텍스트 실습

다음 장에서는

- 판다스 고급 (2)

다음 주는 중간고사입니다!!! ☺