Introduction to Data Mining Lecture

Week 9: Pandas Advanced (2)

Joon Young Kim

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

Week 9 Outline

- Week 7 Review
- 데이터의 그룹 연산
  - 데이터 객체의 그룹 연산
  - GroupBy 객체의 그룹별 연산 및 변환
  - GroupBy 객체를 이용한 분리, 적용 및 통합
  - 기타 그룹연산
- 계산 도구의 사용
  - 통계함수
  - 윈도우 함수
  - 종합 연산(Aggregation)
  - 기타 윈도우 적용
- Conclusion

Week 9 Outline

- Week 7 Review
- 데이터의 그룹 연산
  - 데이터 객체의 그룹 연산
  - GroupBy 객체의 그룹별 연산 및 변환
  - GroupBy 객체를 이용한 분리, 적용 및 통합
  - 기타 그룹연산
- 계산 도구의 사용
  - 통계함수
  - 윈도우 함수
  - 종합 연산(Aggregation)
  - 기타 윈도우 적용
- Conclusion

Week 7 Review

■ 판다스 데이터 처리 관련 실습

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

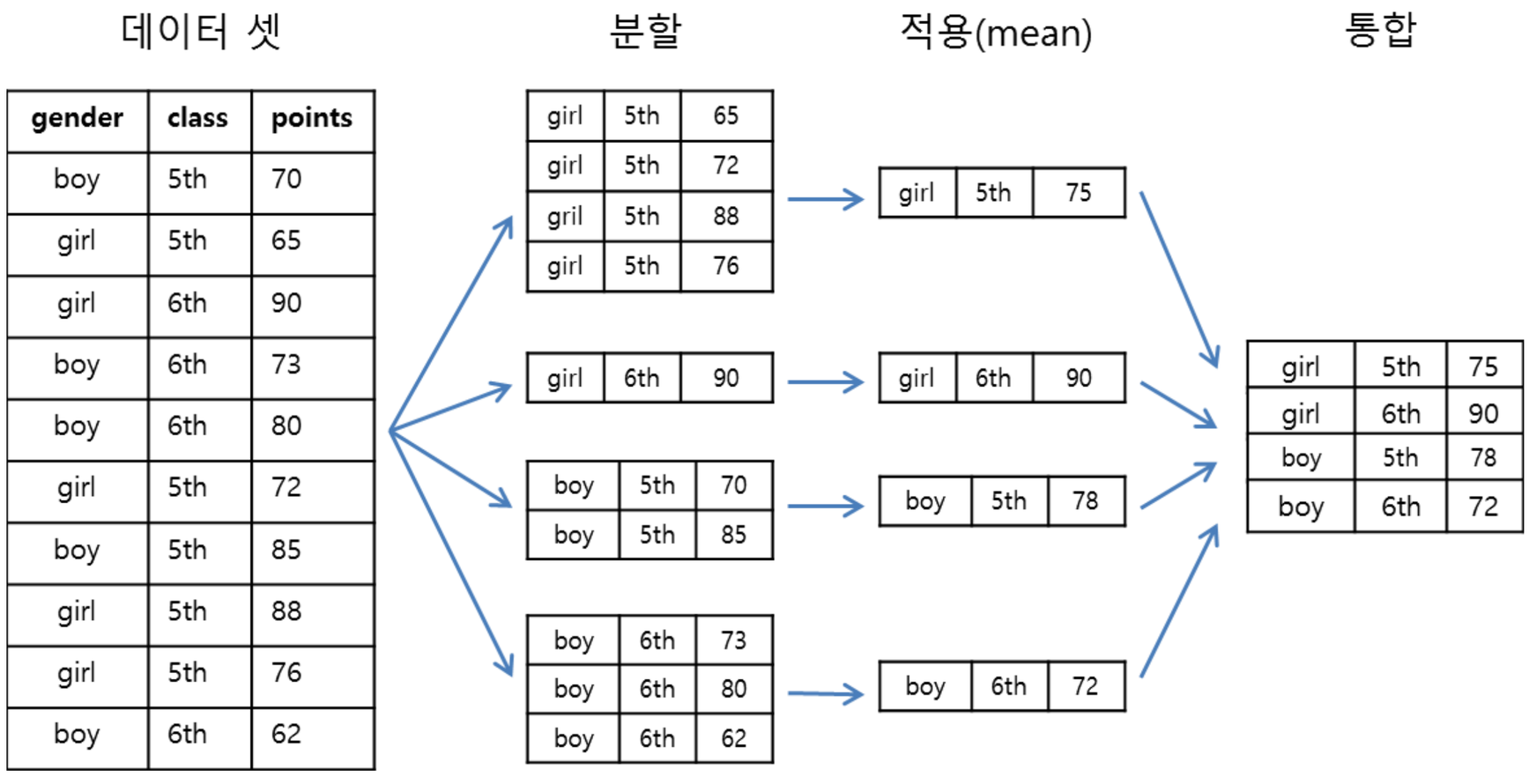
메서드	기능 설명
lower()	배열의 strings를 소문자로 변경
upper()	배열의 strings를 대문자로 변경
len()	배열의 각 문자열의 길이를 계산
strip()	배열의 각 문자열로부터 공백(newlines 포함)을 제거
lstrip()	배열의 각 문자열의 왼쪽부터 공백을 제거
rstrip()	배열의 각 문자열의 오른쪽부터 공백을 제거
replace()	Series/Index에서 pattern/regex의 적용된 것으로 대치
split()	주어진 정규표현식의 패턴이나 구분자로 각 문자열을 분리
cat()	주어진 구분자로 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

Week 9 Outline

- Week 7 Review
- 데이터의 그룹 연산
  - 데이터 객체의 그룹 연산
  - GroupBy 객체의 그룹별 연산 및 변환
  - GroupBy 객체를 이용한 분리, 적용 및 통합
  - 기타 그룹연산
- 계산 도구의 사용
  - 통계함수
  - 윈도우 함수
  - 종합 연산(Aggregation)
  - 기타 윈도우 적용
- Conclusion

데이터의 그룹 연산

■ 데이터의 그룹연산 절차



데이터 객체의 그룹 연산

■ GroupBy object 속성

→ df 생성 – groups 속성

```
In [249]: df = pd.DataFrame({'A': ['ha', 'hi', 'ho', 'ha', 'ho'],
                             'B': ['one', 'two', 'one', 'one', 'two'],
                             'Data1': np.random.randn(5),
                             'Data2': np.random.randn(5)})

In [250]: df
Out[250]:
```

	A	B	Data1	Data2
0	ha	one	0.925169	0.827582
1	hi	two	-0.964132	0.317676
2	ho	one	0.345608	1.090303
3	ha	one	-0.642167	0.523911
4	ho	two	1.244183	-0.986792

```
In [251]: grouped1 = df.groupby('A')

In [252]: grouped1
Out[252]: <pandas.core.groupby.groupby.DataFrameGroupBy
          object at 0x000001F7F64D7630>

In [253]: gr_dict = dict(list(grouped1))

In [254]: gr_dict
Out[254]: {'ha':  A     B      Data1      Data2
             0  ha  one   0.925169   0.827582
             3  ha  one  -0.642167   0.523911,
           'hi':  A     B      Data1      Data2
             1  hi  two  -0.964132   0.317676,
           'ho':  A     B      Data1      Data2
             2  ho  one   0.345608   1.090303
             4  ho  two   1.244183  -0.986792}

In [255]: grouped1.groups
Out[255]: {'ha': Int64Index([0, 3], dtype='int64'),
           'hi': Int64Index([1], dtype='int64'),
           'ho': Int64Index([2, 4], dtype='int64')}
```

데이터 객체의 그룹 연산

■ 인덱스 라벨을 선택하여 해당 데이터 구하기

```
In [256]: gr_dict['ho']

In [257]: grouped1.get_group('ho')

In [258]: df.groupby(['A', 'B']).get_group(('ha', 'one'))
```

■ Date1, Date2 평균값 구하기

```
In [260]: grouped1.mean()
Out[260]:
```

A	Data1	Data2
ha	0.141501	0.675746
hi	-0.964132	0.317676
ho	0.794895	0.051755

데이터 객체의 그룹 연산

■ Data2의 평균값 구하기

```
In [261]: grouped2 = df['Data2'].groupby(df['A'])

In [262]: grouped2

In [263]: grouped2.mean()
```

■ df의 'A', 'B'열을 기준으로 Data1에 대한 평균값 구하기

```
In [264]: grouped3 = df['Data1'].groupby([df['A'], df['B']])

In [265]: grouped3.groups

In [266]: grouped3

In [267]: grouped3.mean()
```

데이터 객체의 그룹 연산

■ df와 같은 길이의 Series와 리스트를 groupby로 전달

```
In [268]: material = np.array([‘water’, ‘oil’, ‘oil’ ,’water’, ‘oil’])

In [269]: time = [‘1hr’, ‘1hr’, ‘2hr’, ‘2hr’, ‘1hr’]

In [270]: df[‘Data1].groupby([material, time]).mean()
```

■ GroupBy 순서 정렬

→ 인덱스 key 값은 디폴트로 순서 정렬됨

```
In [271]: df2 = pd.DataFrame({‘A’: [‘ho’, ‘hi’, ‘ha’],
                               ‘B’: [‘two’, ‘one’, ‘two’],
                               ‘Data1’: np.random.randn(3)})

In [272]: df2.groupby([‘A’]).sum()
```

```
In [273]: df2.groupby(‘A’, sort=False).sum()
```

데이터 객체의 그룹 연산

■ MultiIndex가 있는 객체의 GroupBy

→ 2개 레벨의 MultiIndex가 있는 Series 생성

```
In [274]: arr = [['ha', 'ha', 'hi', 'hi', 'ho', 'ho'], ['one', 'two', 'one', 'one', 'two', 'two']]

In [275]: ind = pd.MultiIndex.from_arrays(arr, names = ['1st', '2nd'])

In [276]: ser = pd.Series(np.random.randn(6), index=ind)

In [277]: ser
```

데이터 객체의 그룹 연산

■ Level=0에 대한 sum() 연산

```
In [278]: ser.index

In [279]: grouped = ser.groupby(level=0)

In [280]: grouped.sum()
```

■ Level=1에 대한 sum() 연산

```
In [281]: ser.groupby(level='2nd').sum()
```

데이터 객체의 그룹 연산

■ 그룹객체의 반복 처리

→ GroupBy 객체를 통한 반복 처리

```
In [285]: grouped1 = df.groupby('A')
```

```
In [286]: for name, group in grouped1:
              print(name)
```

```
In [287]: for name, group in grouped1:
              print(name)
              print(group)
```

데이터 객체의 그룹 연산

■ GroupBy 객체를 통한 반복 처리 – 변수 (n1, n2) 및 group

```
In [288]: for (n1, n2), group in df.groupby(['A', 'B']):
                print((n1, n2))
                print(group)
```

GroupBy 객체의 그룹별 연산 및 변환

■ 데이터 종합 연산(Aggregation)

→ 데이터를 종합하고 요약 연산 : df객체 생성

```
In [289]: df = pd.DataFrame({‘A’: [‘ha’, ‘hi’, ‘ho’, ‘ha’, ‘ho’],
                             ‘B’: [‘one’, ‘two’, ‘one’, ‘one’, ‘two’],
                             ‘Data1’: np.random.randn(5),
                             ‘Data2’: np.random.randn(5)})

In [290]: df
```

GroupBy 객체의 그룹별 연산 및 변환

■ 데이터 요약 : agg() 메소드

```
In [291]: grouped1 = df.groupby('A')          In [292]: grouped1.agg(np.sum)

In [293]: grouped2 = df.groupby(['A', 'B'])   In [294]: grouped2.agg('sum')
```

GroupBy 객체의 그룹별 연산 및 변환

■ as_index 옵션

```
In [295]: grouped3 = df.groupby(['A', 'B'], as_index=False)

In [296]: grouped3.aggregate(np.sum)

In [297]: df.groupby('A', as_index=False).sum()
```

■ size() 메소드

```
In [298]: grouped2.size()
```

GroupBy 객체의 그룹별 연산 및 변환

■ 한번에 여러함수 적용하기

→ SeriesGroupBy 객체 : agg에 함수 list 또는 dict 전달

```
In [299]: grouped = df.groupby('A')

In [300]: grouped['Data1'].agg([np.sum, np.mean, np.std])
```

→ DataFrameGroupBy 객체 : agg에 함수 list 전달

```
In [301]: grouped.agg([np.sum, np.mean])
```

→ 열 이름의 변경

```
In [302]: grouped['Data1'].agg([np.sum, np.mean])
                                .rename(columns={'sum': '합계', 'mean': '평균'})
```

GroupBy 객체의 그룹별 연산 및 변환

■ DataFrameGroupBy - 열 이름의 변경

```
In [303]: grouped.agg([np.sum, np.mean]).rename(columns={‘sum’: ‘합계’,‘mean’: ‘평균’})
```

■ DataFrame 열들에 각기 다른 함수 적용하기

→ agg : 함수들의 dict를 전달

```
In [304]: grouped.agg({‘Data2’: np.sum, ‘Data1’: lambda x: np.sum(x)})
```

GroupBy 객체의 그룹별 연산 및 변환

■ aggregate에 dict전달 : 열의 순서 정렬을 위한 OrderedDict 사용

```
In [305]: from collections import OrderedDict

In [306]: grouped.agg({‘Data2’: ‘sum’, ‘Data1’: np.mean})


In [307]: df_grd = grouped.agg(OrderedDict([(‘Data2’, ‘sum’),
                                             (‘Data1’, ‘mean’)]))

In [308]: df_grd
```

GroupBy 객체의 그룹별 연산 및 변환

■ index라벨 변경 및 sum() 연산

```
In [309]: ind = [‘gold’, ‘silver’, ‘gold’]

In [310]: df_grd.groupby(ind).sum()
```

■ 방법1 : agg() 적용

```
In [313]: df.groupby(‘Branch’)[‘Ext Price’].agg(‘sum’)
```

■ 국내 자동차 지점 영업현황 데이터 연산 변환

→ 영업 현황 데이터 읽기

```
In [311]: df = pd.read_excel(‘car_sales.xlsx’)

In [312]: df
```

GroupBy 객체의 그룹별 연산 및 변환

■ name 변경 : rename() 메소드

```
In [314]: df.groupby('Branch')['Ext Price'].agg('sum').rename('Br_Total')
```

■ 인덱스 리셋 : reset_index() 메소드

```
In [315]: br_total = df.groupby('Branch')['Ext Price'].agg('sum').rename('Br_Total').reset_index()

In [316]: br_total
```

GroupBy 객체의 그룹별 연산 및 변환

- df에 br_total 덧붙임 : merge()

```
In [317]: df_m =  df.merge(br_total)

In [318]: df_m
```

- 차종별 판매금액 비율 구하기

```
In [319]: df_m[‘Br_Pct’] = df_m[‘Ext Price’] / df_m[‘Br_Total’]

In [320]: df_m
```

- 방법2 : transform() 적용

```
In [321]: df.groupby(‘Branch’)[‘Ext Price’].transform(‘sum’)
```

GroupBy 객체의 그룹별 연산 및 변환

■ 지점 내 차량별 매출 비율 구하기

```
In [322]: df['Br_Total'] = df.groupby('Branch')['Ext Price'].transform('sum')

In [323]: df['Br_Pct'] = df['Ext Price'] / df['Br_Total']

In [324]: df
```

GroupBy 객체를 이용한 분리, 적용 및 통합

■ 학교별 수학 성적의 split, apply 및 combine의 적용

→ df 생성 – 학교명, 학생이름, 수학성적

```
In [325]: df = pd.DataFrame({'School': ['Yeonhi', 'Yeonhi', 'Sungsan', 'Sungsan', 'Sungsan'],
                             'Name': ['Haena', 'Gisu', 'Una', 'Naeun', 'Ziho'],
                             'Math_S': [92, 70, 88, 92, 70]})

In [326]: df
```

→ 학교별 평균치 구하기

```
In [327]: mean_s = df.groupby('School')['Math_S'].agg('mean')

In [328]: mean_s
```

GroupBy 객체를 이용한 분리, 적용 및 통합

■ 이름 변경하기 – rename() 메소드

```
In [329]: mean_s.rename('Avg_S')
In [330]: avg_score = mean_s.rename('Avg_S').reset_index()
In [331]: avg_score
```

■ 인덱스 리셋

```
In [332]: df1 = df.merge(avg_score)

In [333]: df1
```

■ 소숫점으로 변환

```
In [334]: df['Rating_S'] = df['Math_S'].apply(lambda x: x/100)

In [335]: df
```

GroupBy 객체를 이용한 분리, 적용 및 통합

■ 수학 성적의 변수 할당

```
In [336]: math_score = df['Math_S']
```

■ 학점 평가를 위한 연산

```
In [337]: grade = []
          for x in math_score:
              if x > 90:
                  grade = grade + ['A']
              elif x > 80:
                  grade = grade + ['B']
              elif x > 70:
                  grade = grade + ['C']
              else:
                  grade = grade + ['F']
```

■ 학점 리스트를 df에 동적 할당

```
In [338]: grade

In [340]: df['Grade'] = grade
```

GroupBy 객체를 이용한 분리, 적용 및 통합

- 수학평균치 Avg_S를 구하고 df에 열 추가 : transform() 메소드

```
In [342]: df['Avg_S'] = df.groupby('School')['Math_S'].transform('mean')

In [343]: df
```

GroupBy 객체를 이용한 분리, 적용 및 통합

■ 학교별 평균점수보다 크면 True

```
In [344]: df['Above_Avg'] = df['Avg_S'] < df['Math_S']

In [345]: df
```

GroupBy 객체를 이용한 분리, 적용 및 통합

■ apply 메소드의 적용

→ Math_S의 올림차순 순서 정렬

```
In [346]: def sort_math(dfs, n=3, column='Math_S'):
              return dfs.sort_values(by=column)[:n]

In [347]: sort_math(df, n=5)
```

GroupBy 객체를 이용한 분리, 적용 및 통합

■ 학교별로 수학점수를 오름차순으로 정렬하기

```
In [348]: df.groupby('School').apply(sort_math)
```

기타 그룹연산

■ 불필요한 부분의 자동배제

→ 열 B의 자동 제거

```
In [349]: df = pd.DataFrame({‘A’: [‘ha’, ‘hi’, ‘ho’, ‘ha’, ‘ho’],
                             ‘B’: [‘one’, ‘two’, ‘one’, ‘one’, ‘two’],
                             ‘Data1’: np.random.randn(5),
                             ‘Data2’: np.random.randn(5)})

In [350]: df
In [351]: df.groupby(‘A’).std()
```

기타 그룹연산

■ 순서정렬된 요소의 그룹화

```
In [352]: data = pd.Series(np.random.randn(16))

In [353]: data

In [354]: factor = pd.qcut(data, [0, .25, .5, .75, 1.])

In [355]: factor

In [356]: data.groupby(factor).mean()
```

기타 그룹연산

■ 각 그룹의 행 취하기

→ groupby의 head, tail 호출

```
In [357]: df = pd.DataFrame([[1, 2], [1, 4], [5, 6], [5, 8]], columns=['A', 'B'])

In [358]: df                          In [359]: gr = df.groupby('A')

In [360]: gr.head(1)                  In [361]: gr.tail(1)
```

기타 그룹연산

■ nth()의 사용

```
In [362]: gr.nth(0)

In [363]: gr.nth(1)                    In [364]: gr.nth(-1)
```

기타 그룹연산

- nth()의 사용 – 인수 dropna
- groupby – 인수 as_index= False

```
In [365]: df = pd.DataFrame([[1, np.nan], [1, 4], [5, 6], [5, 8]], columns=['A', 'B'])

In [366]: df

In [367]: gr = df.groupby('A')

In [368]: gr.nth(0)                    In [369]: gr.nth(0, dropna='any')
In [370]: gr.first()                   In [371]: gr.last()
In [372]: gr1 = df.groupby('A', as_index=False)

In [373]: gr1.nth(0)
```

Week 9 Outline

- Week 7 Review
- 데이터의 그룹 연산
  - 데이터 객체의 그룹 연산
  - GroupBy 객체의 그룹별 연산 및 변환
  - GroupBy 객체를 이용한 분리, 적용 및 통합
  - 기타 그룹연산
- 계산 도구의 사용
  - 통계함수
  - 윈도우 함수
  - 종합 연산(Aggregation)
  - 기타 윈도우 적용
- Conclusion

계산 도구의 사용

■ 퍼센트 변화율(Percent Change)

→ 주어진 수의 기간에 대한 변화율 : pct_change()

→ pct_change() – 인수 periods=3

```
In [374]: ser = pd.Series([1, 2, 3, 4, 5, 6])

In [375]: ser                          In [376]: ser.pct_change()
In [377]: ser.pct_change(periods=3)
```

통계함수

■ pct_change() – 인수 axis

In [378]: df = pd.DataFrame({‘2018’: [0.12, 0.24], ‘2019’: [0.14, 0.26], ‘2020’: [0.10, 0.22]}, index=[‘CO2’, ‘H2O’])

In [379]: df

In [380]: df.pct_change()

In [381]: df.pct_change(axis=’columns’)

통계함수

■ 공분산(Covariance)

→ Series.cov()

```
In [382]: ser1 = pd.Series(np.random.randn(100))
In [383]: ser2 = pd.Series(np.random.randn(100))
In [384]: ser1.cov(ser2)
```

→ DataFrame.cov() – Series 사이의 쌍 단위

```
In [385]: df = pd.DataFrame(np.random.randn(1000, 3),
                            columns=['a', 'b', 'c'])

In [386]: df.cov()
```

→ DataFrame.cov() – 인수 min_periods

```
In [387]: df = pd.DataFrame(np.random.randn(10, 3),  columns=['a', 'b', 'c'])
In [388]: df.loc[df.index[:3], 'a'] = np.nan
In [389]: df.loc[df.index[3:6], 'b'] = np.nan
In [390]: df.cov()
In [391]: df.cov(min_periods=5)
```

통계함수

■ 자기상관(Correlation)

→ corr() – 매개변수 method

method 이름	기능 설명
pearson (디폴트)	표준 자기상관 계수
kendall	Kendall Tau 자기상관 계수
spearman	Spearman rank 자기상관 계수

→ 열 a와 b 그리고 df 열들의 쌍단위의 자기상관

```
In [395]: df['a'].corr(df['b'])
In [396]: df['a'].corr(df['b'], method='spearman')
In [397]: df.corr()
```

→ df 생성

```
In [392]: df = pd.DataFrame(np.random.randn(500, 3),
                            columns=['a', 'b', 'c'])
In [393]: df.iloc[::2] = np.nan
In [394]: df.head(6)
```

통계함수

■ min_periods 키워드 적용

```
In [398]: df1 = pd.DataFrame(np.random.randn(20, 3), columns=['a', 'b', 'c'])

In [399]: df1.loc[df1.index[:5], 'a'] = np.nan

In [400]: df1.loc[df1.index[5:10], 'b'] = np.nan

In [401]: df1.corr()
In [402]: df1.corr(min_periods=12)
```

통계함수

■ 두 프레임 객체들의 shape이 다르면 NaN

```
In [403]: ind = ['a', 'b', 'c', 'd']                    In [404]: col = ['one', 'two', 'three']

In [405]: df1 = pd.DataFrame(np.random.randn(4, 3), index=ind, columns=col)

In [406]: df2 = pd.DataFrame(np.random.randn(3, 3), index=ind[:3], columns=col)

In [407]: df1.corrwith(df2)          In [408]: df2.corrwith(df1, axis=1)
```

통계함수

■ 데이터 순위(ranking)

→ rank() 메소드

```
In [409]: ser = pd.Series(np.random.randn(5), index=list('abcde'))

In [410]: ser                          In [411]: ser['d'] = ser['b']

In [412]: ser.rank()
```

통계함수

■ NaN값은 순위에서 제외

```
In [413]: df = pd.DataFrame(np.random.randn(5, 3))

In [414]: df

In [415]: df[0][:2]
```

```
In [416]: df[2] = df[0][:2]

In [417]: df

In [418]: df.rank(1)
```

윈도우 함수

- window나 이동 통계 자료를 계산
- count, sum, mean, correlation, variance, standard deviation 등
- rolling, expanding 메소드는 DataFrameGroupBy 객체로 부터 사용

```
In [419]: s = pd.Series(np.random.randn(1000),
                        index=pd.date_range('1/1/2020', periods=1000))

In [420]: s
```

```
In [421]: ser = s.cumsum()

In [422]: ser
```

윈도우 함수

■ window=60인 rolling() 메소드

```
In [423]: roll = s.rolling(window=60)

In [424]: roll

In [425]: type(roll)

In [426]: roll.<Tab>

In [427]: roll.mean()
```

윈도우 함수

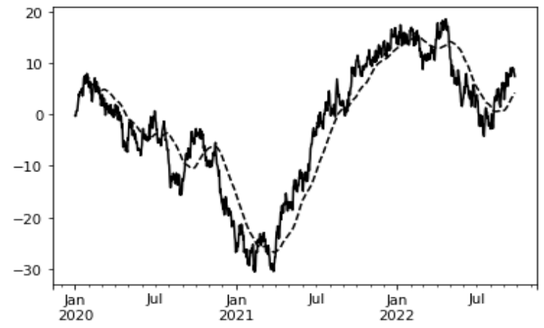
■ rolling 객체가 지원하는 메소드

메소드 종류	기능 설명
count()	null이 아닌 관찰치의 수
sum()	값들의 합
mean()	값들의 평균
median()	산술적인 메디안 값
min()	최소
max()	최대
std()	Bessel 보정의 표본 표준편차
var()	편향되지 않는(Unbiased) 분산
skew()	표본 비대칭도(sample kurtoosis) 또는 표본 왜도 (3rd moment)
kurt()	표본 첨도(sample kurtosis) (4th moment)
quantile()	표본 분위수(sample quantile)로 %로 표기되는 값
apply()	일반적인 적용
cov()	편향되지 않는 공분산(binary)
corr()	자기상관(binary)

윈도우 함수

■ ser.plot과 roll.mean()의 그래프 비교

```
In [428]: ser.plot(style='k')
          roll.mean().plot(style='k—')
```



→ rolling 메소드 적용 – df 생성

```
In [429]: df = pd.DataFrame(np.random.randn(1000, 3), index=pd.date_range('1/1/2020', periods=1000), columns=['A', 'B', 'C'])

In [430]: df
```

윈도우 함수

→ rolling 메소드 – 열방향의 cumsum 메소드 적용

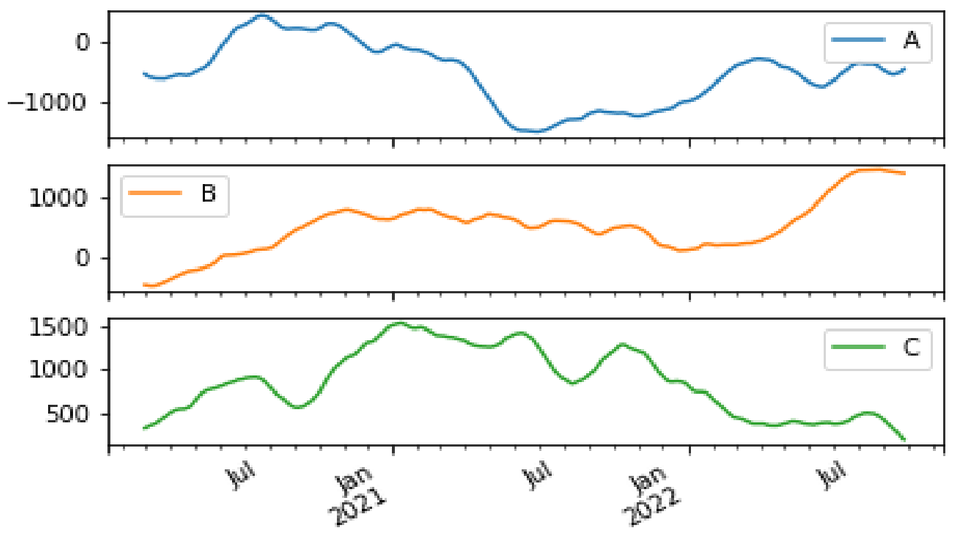
```
In [431]: dfc = df.cumsum()

In [432]: dfc
```

→ rolling 메소드 – window=60

```
In [433]: dfc.rolling(window=60).sum()
```

→ rolling 메소드 – 그래프 그리기

```
In [434]: dfc.rolling(window=60).sum().plot(subplots=True)
```



윈도우 함수

■ rolling 합계 구하기 – ser 생성

```
>>> ser = pd.Series([1, 2, 3, 4, 5])        >>> ser
```

■ rolling() – center=True

```
>>> ser.rolling(3).sum()                >>>  ser.rolling(3, center=True).sum()
```

윈도우 함수

■ 시간인식(time aware) rolling

→ df 생성

```
In [435]: dft = pd.DataFrame({'val': [0, 1, 2, np.nan, 4]},
                             index=pd.date_range('20200101 09:00:00',
                                                 periods=5, freq='s'))

In [436]: dft
```

→ rolling() – min_periods

```
In [437]: dft.rolling(2).sum()

In [438]: dft.rolling(2, min_periods=1).sum()
```

윈도우 함수

■ 시간인식(time aware) rolling

→ rolling() 메소드 – 옵셋 '2s'

```
In [439]: dft.rolling('2s').sum()
```

→ rolling() 메소드 – 인수 window

```
In [440]: dft1 = pd.DataFrame({'val': [0, 1, 2, np.nan, 4]},
                               index=pd.Index([pd.Timestamp('20200101 09:00:00'),
                                               pd.Timestamp('20200101 09:00:02'),
                                               pd.Timestamp('20200101 09:00:03'),
                                               pd.Timestamp('20200101 09:00:05'),
                                               pd.Timestamp('20200101 09:00:06')],
                                              name='ha'))

In [441]: dft1                                 In [442]: dft1.rolling(2).sum()
```

윈도우 함수

■ 시간인식(time aware) rolling

→ rolling() 메소드 – 옵셋 '2s'

```
In [443]: dft1.rolling('2s').sum()
```

→ rolling() 메소드 – 열 인덱스 명시를 위한 인수 on

```
In [444]: dft2 = dft1.reset_index()

In [445]: dft2
```

```
In [446]: dft2.rolling('2s', on='ha').sum()
```

윈도우 함수

■2진 윈도우 함수

→ df 생성

```
In [447]: df = pd.DataFrame(np.random.randn(500, 3),
                            index=pd.date_range('1/1/2020', periods=500),
                            columns=['A', 'B', 'C'])

In [448]: dfc = df.cumsum()
In [449]: dfc.head()
```

→ rolling 윈도우를 적용하고 자기상관 구하기

```
In [450]: df1 = dfc[:20]

In [451]: df1.head()
```

→ df1 객체 생성

```
In [452]: df1.rolling(window=5).corr(df1['B'])
```

윈도우 함수

- df1객체에 rolling윈도우 적용하고 df1과의 자기상관 구하기

```
In [452]: df1.rolling(window=5).corr(df1[‘B’])
```

윈도우 함수

- 쌍단위의 공분산과 자기상관을 롤링윈도우로 계산하기

  → 롤링 윈도우 적용하고 공분산 구하기

```
In [453]: cvar = (df[['B', 'C']].rolling(window=50).cov(df[['A', 'B']],
                                                        pairwise=True))

In [454]: cvar
```

→ loc메소드 적용

```
In [455]: cvar.loc['2021-05-12':]
```

윈도우 함수

■ df객체에 롤링 윈도우 적용후 자기상관 구하기

```
In [456]: dfr = df.rolling(window=50).corr()

In [457]: dfr.loc['2021-05-12':]
```

윈도우 함수

■ dfr객체에 unstack() 적용

```
In [458]: dfr.unstack().tail(4)
```

■ dfr.unstack에서 tail() 적용

```
In [459]: dfr.unstack()[('A', 'C')]

In [460]: dfr.unstack()[('A', 'C')].tail(3)
```

■ dfr.unstack에서 plor() 적용

```
In [461]: dfr.unstack()[('A', 'C')].plot()
```

종합 연산(Aggregation)

- Rolling 객체를 생성하여 agg() 메소드 적용

```
In [462]: df = pd.DataFrame(np.random.randn(1000, 3),
                            index=pd.date_range('1/1/2020', periods=1000),
                            columns=['A', 'B', 'C'])

In [463]: df

In [464]: rol = df.rolling(window=60, min_periods=1)

In [465]: rol
```

종합 연산(Aggregation)

- Rolling 객체를 생성하여 agg() 메소드 : numpy.sum 함수 적용

```
In [466]: rol.aggregate(np.sum)
In [467]: rol['A'].aggregate(np.sum)
```

```
In [468]: rol[['A', 'C']].agg(np.sum)
```

종합 연산(Aggregation)

■ DataFrame 열들에 여러 함수들 적용하기

→ 종합연산 적용 – 사전형 전달

→ 종합연산 적용 – 문자열의 전달함수

```
In [471]: rol.agg({‘A’: np.sum, ‘C’: lambda x:
                                np.std(x, ddof=1)})
```

```
In [472]: rol.agg({‘A’: ‘sum’, ‘C’: ‘std’})
```

종합 연산(Aggregation)

■ 종합연산 적용 – 중첩된 사전형 전달

```
In [473]: rol.agg({‘A’: [‘sum’, ‘std’], ‘C’: [‘mean’, ‘std’]})
```

기타 윈도우 적용

■ 확장 윈도우

→ expanding() 메소드

```
In [474]: df = pd.DataFrame(np.random.randn(100, 3),
                            index=pd.date_range('7/1/2020', periods=100),
                            columns=['A', 'B', 'C'])

In [475]: df.rolling(window=len(df), min_periods=1).mean()[:5]
```

```
In [476]: df.expanding(min_periods=1)

In [477]: df.expanding(min_periods=1).mean()[:5]
```

기타 윈도우 적용

■ 시리즈 객체 생성 : expanding() 메소드 적용

```
In [478]: ser = pd.Series([1, 2, np.nan, 3, np.nan, 4])

In [479]: ser.expanding().sum()          In [480]: ser.cumsum()
In [481]: ser.cumsum().fillna(method='ffill')
```

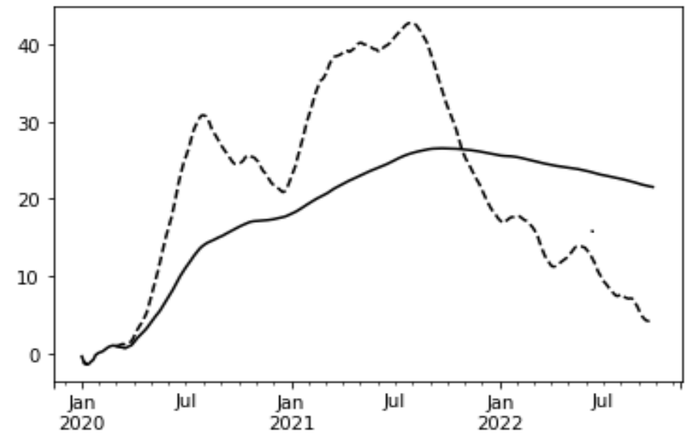
기타 윈도우 적용

■ expanding() –Serial 객체 생성

```
In [482]: s = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2020', periods=1000))

In [483]: ser = s.cumsum()

In [484]: rol = ser.rolling(window=60)
```

■ rolling()과 expanding()의 비교

```
In [485]: rol.mean().plot(stype='k—')
          ser.expanding().mean().plot(style='k')
```



기타 윈도우 적용

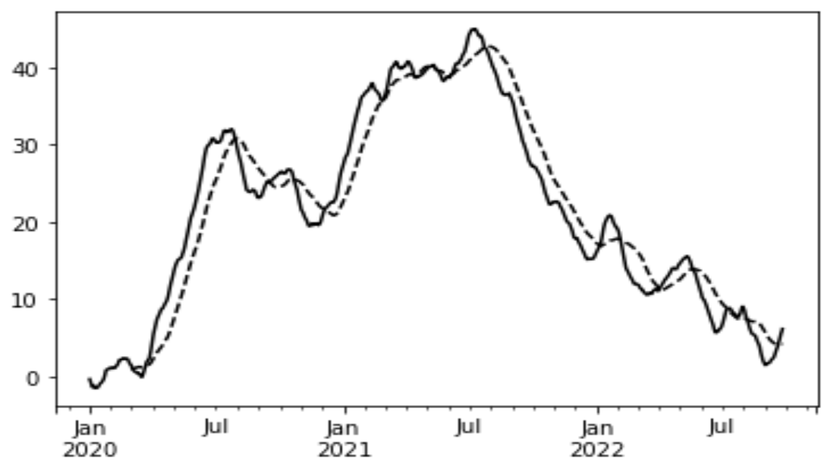
■ 지수 가중 윈도우(Exponentially Weighed Windows)

→ ewm() 메소드의 함수

함수	기능설명
mean()	EW 이동 평균
var()	EW 이동 분산
std()	EW 이동 표준편차
corr()	EW 이동 자기상관
cov()	EW 이동 공분산

→ ewm() 메소드의 적용

```
In [486]: ser.ewm(span=20).mean().plot(style='k')
          rol.mean().plot(style='k—')
```



Week 9 Outline

- Week 7 Review
- 데이터의 그룹 연산
  - 데이터 객체의 그룹 연산
  - GroupBy 객체의 그룹별 연산 및 변환
  - GroupBy 객체를 이용한 분리, 적용 및 통합
  - 기타 그룹연산
- 계산 도구의 사용
  - 통계함수
  - 윈도우 함수
  - 종합 연산(Aggregation)
  - 기타 윈도우 적용
- Conclusion

Week 9 Key Takeaway

- 판다스 고급 응용 학습

  → 데이터 그룹 연산 및 계산 도구 활용법 학습 및 실습

다음 주에서는

- matplotlib (1)