Introduction to Data Mining Lecture

Week 5: Pandas (1)

Joon Young Kim

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

Week 5 Outline

- Week 4 Review
- Pandas Overview
- Pandas Basics
  - Data Structure
  - Series & DataFrame
  - Row & Column
  - Index Object
- Pandas Main Feature
  - Binary Operation
  - Statistics
  - Function
- Conclusion

Week 5 Outline

- Week 4 Review
- Pandas Overview
- Pandas Basics
  - Data Structure
  - Series & DataFrame
  - Row & Column
  - Index Object
- Pandas Main Feature
  - Binary Operation
  - Statistics
  - Function
- Conclusion

Week 4 Review

■ 넘피내 유니버셜 기능 및 어레이 매니지먼트 학습

→ 뷰/카피 및 브로드캐스팅 포함한 다양한 기능 학습

클래스	numpy.broadcast
매개변수	in1, in2, ... : 유사배열, 입력 매개변수
반환값	b : 브로드캐스트 객체

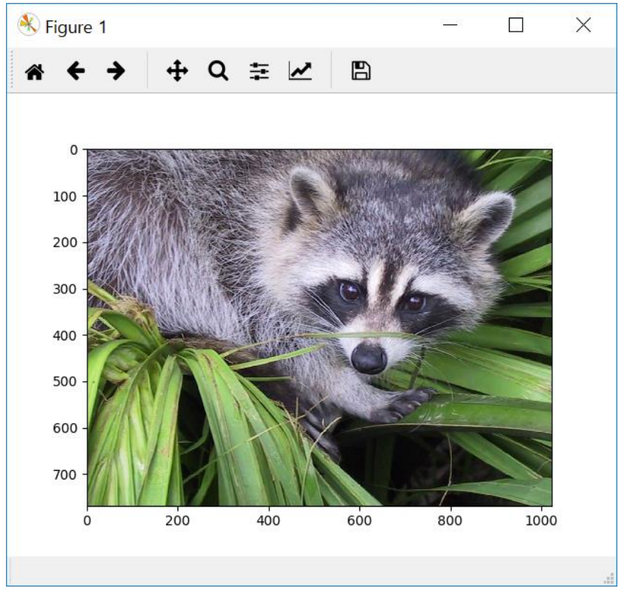
구 분	종 류	기 능
속성	index	브로드캐스트 된 결과에서 현재 색인
	iters	self의 요소에 따르는 반복자(iterators)의 튜플
	nd	브로드캐스트 된 결과의 차원 수
	ndim	브로드캐스트 된 결과의 차원 수(넘파이 1.12.0 이후)
	numiter	브로드캐스트 된 결과에 의해 소유된 반복자의 수
	shape	브로드캐스트 된 결과의 shape
	size	브로드캐스트 된 결과의 총 크기
메서드	reset()	브로드캐스트 된 결과의 iterators를 리셋

Week 4 Review

- 이미지 처리 통한 실제 numpy 연계 실습

  → 너구리 이미지 통한 모듈 불러오기 및 색감 실습

```
import scipy.misc
import matplotlib.pyplot as plt
face = scipy.misc.face()
face.shape
(768, 1024, 3)
face.max(), face.min(), face.mean()
(255, 0, 110.16274388631184)
face.dtype
dtype('uint8')
plt.gray()
plt.imshow(face)
<matplotlib.image.AxesImage at 0xbaa470>
plt.show()
```



Week 5 Outline

- Week 4 Review
- Pandas Overview
- Pandas Basics
  - Data Structure
  - Series & DataFrame
  - Row & Column
  - Index Object
- Pandas Main Feature
  - Binary Operation
  - Statistics
  - Function
- Conclusion

Pandas

■ 판다스 개요

→ pandas는 panel data syste의 약어

→ 2008년 초 Wes McKinney에 의해 개발 착수

→ 2015년 비영리단체 NumFOCUS에 의해 오픈소스로 관리됨

■ 판다스 주요 기능

→ 데이터 정렬과 손실 데이터의 통합처리

→ 데이터 셋의 reshaping과 pivoting, slicing, indexing, subsetting

→ 데이터 구조 열을 삽입하고 지우기

→ 데이터 셋에 split-apply-combine 연산 및 merging/joining

→ 다양한 시계열 기능

Pandas

■ 판다스의 구성 요소

→ 라벨처리된 배열 데이터 구조의 셋인 Series와 DataFrame

→ 단순 축 인덱싱 및 멀티레벨/계층적 축 인덱싱의 인덱스 객체

→ 데이터 셋을 종합하고 변형하기 위한 엔진에 의한 통합그룹

→ 날짜 구간 발생 및 커스텀 날짜 옵셋

→ 입력/출력 도구

→ 효율적인 메모리의 표준데이터 구조

→ 이동 윈도우 통계(이동평균, 이동표준편차)

Pandas

■ 적합한 처리의 데이터 종류

→ SQL이나 Excel과 같은 테이블형 데이터

→ 시계열 데이터

→ 행과 열 라벨이 있는 임의의 매트릭스 데이터

→ 어떤 형태의 관측/통계 데이터 셋

■ 판다스 API 라이브러리 구조

→ Series, DataFrame, Index, Scalars 등

Week 5 Outline

- Week 4 Review
- Pandas Overview
- Pandas Basics
  - Data Structure
  - Series & DataFrame
  - Row & Column
  - Index Object
- Pandas Main Feature
  - Binary Operation
  - Statistics
  - Function
- Conclusion

Data Structure

■ 판다스는 Series와 DataFrame의 데이터 구조를 지원

차원	이름	설명
1	Series	1차원의 라벨 표기된 homogenious 형태의 배열
2	DataFrame	heterogeneous 형태의 열을 가지며 2차원의 라벨이 표기되고 변동가능한 크기의 테이블형 구조

```
import numpy as np

import pandas as pd
```

Series

■ 시리즈 생성

→ data는 파이썬 사전형, ndarray 또는 스칼라, index는 축 라벨의 리스트

```
ser = pd.Series(data, index=index)
```

Series

- ndarray로부터 시리즈 객체 생성

```
In [3]: ser=pd.Series(np.random.randn(4), index=['a','b','c','d'])

In [4]: ser
```

- pandas.series 속성 values와 index

```
In [5]: ser.values

In [6]: ser.index
```

Series

■ 사전형(dict)으로부터 생성

```
In [6]: da = {'seoul' : 2000, 'busan' : 2500, 'daejeon' : 3000}

In [7]: pd.Series(da)
```

```
In [8]: da = {'a' : 0., 'b' : 1., 'c' : 2.}

In [9]: pd.Series(da)


In [10]: pd.Series(da, index=['b', 'c', 'd', 'a'])
```

Series

■ 스칼라 값으로부터 생성

```
In [11]: pd.Series(7., index=['a', 'b', 'c']
```

■ ndarray와 같은 시리즈

```
In [12]: ser[0]

In [13]: ser[:3]                          In [14]: np.exp(ser)
```

Series

■ dict같은 시리즈

```
In [15]: ser['a']
```

```
In [16]: ser['d'] = 7.
```

```
In [17]: ser
```

■ 시리즈로 벡터화 연산 및 라벨 정렬a

→ 시리즈객체도 넘파이와 같이 산술적 연산이 가능

```
In [18]: ser + ser
```

```
In [19]: ser * 2
```

Series

■ 라벨에 기반한 데이터의 자동 정렬

```
In [20]: ser[1:] + ser[:-1]
```

■ name 속성

→ Series의 name 속성과 rename() 메소드

```
In [21]: ser = pd.Series(np.random.randn(5),  name='seoul')

In [22]: ser

In [23]: ser1 = ser.rename('busan')

In [24]: ser1.name
```

DataFrame

- DataFrame에 적용되는 입력
  - → 1차원 ndarrays, lists, dicts 또는 Series의 Dict
  - → 2차원의 ndarray
  - → Series 또는 또다른 DataFrame

- dict로부터 데이터프레임 객체 생성
  - → 데이터프레임의 인덱스는 여러 Series의 인덱스들의 합집합

```
In [25]: d = {'one': pd.Series([1., 2., 3.],
                               index=['a', 'b', 'c']),
              'two': pd.Series([1., 2., 3., 4.],
                               index=['a', 'b', 'c', 'd'])}

In [26]: df = pd.DataFrame(d)

In [27]: df
```

DataFrame

- 인덱스와 열라벨의 표기에 의한 객체 생성

```
In [30]: pd.DataFrame(d, index=['d', 'b', 'a'])

In [31]: pd.DataFrame(d, index=['d', 'b', 'a'], columns=['two', 'three']]
```

- 속성 index, columns

```
In [32]: df.index

In [33]: df.columns
```

DataFrame

- ndarrays/lists의 dict로부터 데이터프레임 객체 생성

```
In [34]: d = {'one': [1., 2., 3.],
              'two': [3., 2., 1.]}

In [35]: pd.DataFrame(d)

In [36]: pd.DataFrame(d, index=['a', 'b', 'c'])
```

DataFrame

■ 구조화된 배열 또는 레코드 배열로부터 객체 생성

```
In [37]: arr = np.zeros((2,), dtype=[('A', 'i4'), ('B', 'f4'), ('C', 'a10')])

In [38]: arr[:] = [(1, 2., 'Hello'), (2, 3., 'World')]

In [39]: arr

In [40]: pd.DataFrame(arr)

In [41]: pd.DataFrame(arr, index=['first', 'second'])

In [42]: pd.DataFrame(arr, columns=['C', 'A', 'B'])
```

DataFrame

- dicts의 list로부터 데이터프레임 객체 생성

```
In [43]: data = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]

In [44]: pd.DataFrame(data)

In [45]: pd.DataFrame(data, columns=['a', 'b'])
```

DataFrame

- 튜플의 dict로부터 데이터프레임 객체 생성

```
In [46]: pd.DataFrame({('a','b'): {('A','B'): 1, ('A','C'): 2},
                        ('a','a'): {('A','C'): 3, ('A','B'): 4},
                        ('a','c'): {('A','B'): 5, ('A','C'): 6},
                        ('b','a'): {('A','C'): 7, ('A','B'): 8},
                        ('b','b'): {('A','D'): 9, ('A','B'): 10}})
```

DataFrame

■ 데이터프레임 생성자

→ DataFrame.from_dict 생성자로부터 DataFrame 객체 생성

```
In [47]: dict([('A', [1, 2, 3]), ('B', [4, 5, 6])])

In [48]: pd.DataFrame.from_dict(dict([('A', [1, 2, 3]), ('B', [4, 5, 6])]))
```

→ 생성자에 orient='index'를 전달하면 keys가 행라벨이 된다.

```
In [49]: pd.DataFrame.from_dict(dict([('A', [1, 2, 3]), ('B', [4, 5, 6])]),
                                 orient='index', columns=['one', 'two', 'three'])
```

DataFrame

- DataFrame.from_records의 생성자로부터 DataFrame 객체 생성

```
In [50]: arr

In [51]: pd.DataFrame.from_records(data, index='C')
```

Row & Column

■ 행 또는 열의 선택, 추가 및 삭제 – 27번째 코드에서 df발췌

```
In [52]: df['one']

In [53]: df['two']

In [54]: df['three'] = df['one'] * df['two']

In [55]: df['flag'] = df['one'] > 2

In [56]: df
```

Row & Column

■ 열의 삭제 및 발췌

```
In [57]: del df['two']

In [58]: df.pop('three')

In [59]: df
```

Row & Column

■ 스칼라 값의 동적 할당

```
In [60]: df['ha'] = 'hiho'

In [61]: df
```

■ 시리즈 객체의 동적 할당

```
In [62]: df['trunced_one'] = df['one'][:2]

In [63]: df
```

Row & Column

- DataFrame.insert 함수 적용

```
In [64]: df.insert(1, 'hi', df['one'])
```

```
In [65]: df
```

Row & Column

- Series.drop 함수 적용

```
In [66]: ser = pd.Series(data=np.arange(3), index=['A', 'B', 'C'])

In [67]: ser

In [68]: ser.drop(labels=['B', 'C])
```

Row & Column

- DataFrame.drop를 이용한 특정 행 및 열의 제거

```
In [69]: df1= pd.DataFrame(np.arange(12).reshape(3, 4),
                           columns=['A', 'B', 'C', 'D'])

In [70]: df1
```

```
In [71]: df1.drop(['B', 'C'], axis=1)

In [72]: df1.drop([0, 1])
```

Row & Column

■ 인덱싱과 선택

→ 기본적인 인덱싱 방법

적용 방법	신택스	결과 타입
열 선택	df[col]	Series
라벨로 행 선택	df.loc[label]	Series
정수 위치에 의한 행 선택	df.iloc[loc]	Series
행 자르기(slice)	df[5:10]	DataFrame
불리언 벡터에 의한 행 선택	df[bool_vec]	DataFrame

Row & Column

- DataFrame.loc와 DataFrame.iloc – df는 65번째 코드

```
In [73]: df.loc['b']

In [74]: df.iloc[2]
```

Row & Column

■ 데이터 정렬 및 산술 연산

→ 인덱스와 열 사이를 자동정렬

```
In [75]: df = pd.DataFrame(np.random.randn(5, 4), columns=['A', 'B', '
C', 'D'])

In [76]: df2 = pd.DataFrame(np.random.randn(3, 3), columns=['A', 'B',
'C'])

In [77]: df + df2
```

Row & Column

■ 데이터 프레임과 시리즈간 연산

```
In [79]: df * 10 + 2
```

Row & Column

■ 불리언 연산자의 적용

```
In [80]: df1 = pd.DataFrame({'a': [1, 0, 1], 'b': [0, 1, 1]}
, dtype=bool)

In [81]: df2 = pd.DataFrame({'a': [0, 1, 1], 'b': [1, 1, 0]}
, dtype=bool)
```

```
In [82]: df1
```

```
In [83]: df2
```

```
In [84]: df1 & df2
```

```
In [85]: df1 | df2
```

```
In [86]: df1 ^ df2
```

```
In [87]: -df1
```

Row & Column

■ 전치(Transpose)

→ T 속성의 적용

```
In [88]: df[:2]
In [89]: df[:2].T
```

Row & Column

■ 넘파이 함수들과 데이터프레임 상호연동

→ Exp 등 넘파이 ufuncs와 넘파이 함수들의 연동 사용

```
In [90]: df3 = pd.DataFrame(np.arange(12).reshape(3, 4), columns=['A', 'B', '
C', 'D'])

In [91]: np.exp(df3)

In [92]: np.asarray(df3)
```

Index Object

■ 넘파이 함수들과 데이터프레임 상호연동

→ Index 관련 객체

```
ind = pd.Index([1, 3, 5, 7, 9, 11])
ind
ind[::2]
ind.ndim, ind.shape
ind[1] = 6 (error)
```

■ 클래스 pandas.Index

```
pd.Index([1, 2, 3])
pd.Index(list('abc'))
```

Index Object

■ 클래스 pandas.RangeIndex

```
 df = pd.DataFrame(np.arange(12).reshape(2, 6), columns=list('ABCD
EF'))
 df
 df.index
```

■ 클래스 pandas.CategoricalIndex

→ 순서를 가지나 산술연산은 불가능 (사용 메모리 절약)

```
ser = pd.Series(['ha', 'hi'] * 1000)
ser.nbytes
ser.astype('category').nbytes
```

```
ser
    ser.astype('category')
```

Index Object

■ 클래스 pandas.Categorical

→ 사용예

```
s1 = pd.Categorical([1, 2, 3, 1, 2, 3])
s1
type(s1)
s1.dtype
s2 = pd.Categorical(['a', 'b', 'c', 'a', 'b', 'c'])
s2
```

→ Ordered=True인 경우

```
s3 = pd.Categorical(['a','b','c','a','b','c'], ordered=True, categories=['c', 'b', 'a'])
s3
s3.min(), s3.max()
```

Index Object

■ 시리즈 구성 시 dtype='category'

```
In [93]: ser = pd.Sereis(['a', 'b', 'c', 'a'],
                                    dtype='category')

In [94]: ser
```

■ category dtype로의 변환

```
In [95]: df = pd.DataFrame({'A': ['a', 'b', 'c', 'a']})

In [96]: df['B'] = df['A'].astype('category')

In [97]: df                                  In [98]: df.dtypes
```

■ DataFrame 생성자에 dtype을명시하여 생성 중 전환

```
In [99]: df = pd.DataFrame({'A': list('abca'),
                                  'B': list('bccd')}, dtype='category')
 In [100]: df.dtypes
```

Index Object

■ 열 단위로 변환

```
In [101]: df['A']
```

```
In [102]: df['B']
```

■ DataFrame.astype()을 사용하여 한꺼번에 변환

```
In [103]: df = pd.DataFrame({'A': list('abca'), 'B': list('bccd')})

In [104]: df_cat = df.astype('category')
```

```
In [105]: df_cat.dtype
```

Index Object

■ 클래스 pandas.MultiIndex

→ MultiIndex의 생성

- MultiIndex.from_arrays()
- MultIndex.from_product()
- MultIndex.from_tuples()

```
In [114]: arr = [[1, 1, 2, 2], ['red', 'blue', 'red', 'blue']]

In [115]: pd.MultiIndex.from_arrays(arr, names=('number', 'color'))
```

Index Object

■ tuple 리스트의 전달

```
In [116]: arr=[['ha', 'ha', 'hi', 'hi', 'ho', 'ho',],
                    ['one', 'two', 'one', 'two', 'one', 'two']]

In [117]: tuples = list(zip(*arr))


In [118]: tuples

In [119]: ind = pd.MultiIndex.from_tuples(tuples,
                                          names=['first', 'second'])
```

```
In [120]: ind

In [121]: ser = pd.Series(np.random.randn(6), index=ind)

In [122]: ser
```

Index Object

■ 배열 리스트를 Series나 DataFrame으로 전달하여 MultiIndex 생성

```
In [123]: arr = [np.array(['ha', 'ha', 'hi', 'hi', 'ho', 'ho']),
                 np.array(['one','two','one','two','one','two'])]

In [124]: ser = pd.Series(np.random.randn(6), index=arr)

In [125]: ser
```

```
In [126]: df = pd.DataFrame(np.random.randn(6, 4), index=arr)

In [127]: df
```

Index Object

■ 축에 튜플을 라벨로 사용

```
In [132]: pd.Series(np.random.randn(6), index=tuples)
```

Week 5 Outline

- Week 4 Review
- Pandas Overview
- Pandas Basics
  - Data Structure
  - Series & DataFrame
  - Row & Column
  - Index Object
- Pandas Main Feature
  - Binary Operation
  - Statistics
  - Function
- Conclusion

Pandas Main Feature

■ head()와 tail() 메소드의 적용

```
In [133]: ser = pd.Series(np.random.randn(1000))

In [134]: ser.head()

In [135]: ser.tail(3)
```

■ index와 columns 속성

```
In [136]: ind = pd.date_range('1/1/2019', periods=5)

In [137]: ser = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])

In [138]: df = pd.DataFrame(np.random.randn(5, 3), index=ind, columns=['A', 'B', 'C'])

In [139]: df[:2]
```

Binary Operation

■ 판다스 객체의 바이너리 연산

→ df 생성

```
In [140]: df = pd.DataFrame({'one':pd.Series(np.random.randn(2),index=['a','b']),
                             'two':pd.Series(np.random.randn(3),index=['a','b','c']),
                             'three':pd.Series(np.random.randn(2),index=['b','c'])})

In [141]: df

In [142]: df.iloc[1]                    In [143]: df['two']

In [144]: row = df.iloc[1]              In [145]: col = df['two']
```

Binary Operation

■ 축의 값에 따른 sub() 메소드 적용

```
In [146]: df.sub(row, axis='columns')
In [147]: df.sub(col, axis=0)
```

Binary Operation

■ df의 생성

```
In [148]: d = {'one': [1., 2., np.nan],
               'two': [3., 2., 1.],
               'three': [np.nan, 1., 1.]}

In [149]: df = pd.DataFrame(d, index=list('abc')

In [150]: df
```

```
In [151]: d1 = {'one': pd.Series([1.,2.], index=['a','b']),
                'two': pd.Series([1.,1.,1.], index=['a','b','c']),
                'three': pd.Series([2.,2.,2.], index=['a','b','c'])}

In [152]: df1 = pd.DataFrame(d1)

In [153]: df1
```

Binary Operation

■ 손실값의 대치

```
In [154]: df + df1
```

```
In [155]: df.add(df1, fill_value=0)
```

■ df 생성 및 스칼라 연산

```
df = pd.DataFrame({'angles': [0, 3, 4], 'degrees': [360, 180, 360]},
                                  index=['circle', 'triangle', 'rectangle'])
df
```

```
df + 1
```

Binary Operation

■ 축에 의해 리스트를 빼는 연산

```
df - [1, 2]                                    df.sub([1, 2], axis='columns')
```

■ df에서 시리즈를 빼는 연산

```
df1 = df.sub(pd.Series([1, 2, 3], index=['circle, 'triangle', 'rectangle']), axis='index')
df1
```

Statistics

■ 통계처리 계산에서의 손실데이터 처리

→ 데이터를 더할 때 NA값들은 0로 취급

→ 데이터가 모두 NA이면 그 결과는 0

→ cumsum()과 cumprod()는 디폴트로 NA값들을 무시하지만 결과 배열에서는 유지

```
In [156]: df

In [157]: df.mean(0)                                    In [158]: df.mean(1)
```

Statistics

■ skipna옵션 – 디폴트는 True

```
In [159]: df.sum(0, skipna=False)
```

```
In [160]: df.sum(1, skipna=True)
```

■ std() 함수 적용

```
In [161]: df.std()

In [162]: df.std(axis=1)  # 디폴트로 ddof=1
```

```
In [163]: np.std(df, axis=1)    # 디폴트로 ddof=0
In [164]: np.std(df, ddof=1, axis=1)
```

```
In [165]: df[['one', 'two', 'three']].std()
```

Statistics

■ cumsum() 메소드 적용 & mean() 함수 적용

```
In [166]: df.cumsum()

In [167]: np.mean(df['one'])
```

■ Series.nunique() 메소드 적용 – non-NA 종류 반환

■ describe() 메소드 적용

```
In [168]: ser = pd.Series(np.random.randn(500))

In [169]: ser[20:500] = np.nan

In [170]: ser[10:20] = 5

In [171]: ser.nunique()
```

```
In [172]: ser = pd.Series(np.random.randn(1000))

In [173]: ser[::2] = np.nan

In [174]: ser.describe()
```

Statistics

■ describe() 메소드 – 백분위수 선택

```
In [178]: ser.describe(percentiles=[0.05, 0.25, .75, .95])
```

■ 수치가 아닌 객체의 describe()

```
In [179]: ser = pd.Series(['a','a','b','c','c',np.nan, 'c', 'd'])

In [180]: ser.dercribe()
```

Statistics

■ 수치와 수치가 아닌 객체 혼합에 대한 describe()

```
In [181]: df = pd.DataFrame({'a': ['Yes', 'Yes', 'No', 'No'], 'b': range(4)})

In [182]: df.describe()
```

Statistics

■ describe() 인수 include/exclude

```
In [183]: df.describe(include=['object'])
```

```
In [184]: df.describe(include=['numbers'])
```

```
In [185]: df.describe(include='all')
```

Statistics

■ 최대값, 최소값 요소의 해당 인덱스 계산 – idxmax, idxmin

```
In [186]: ser = pd.Series(np.random.randn(5))
```

```
In [187]: ser
```

```
In [188]: ser.idxmin(), ser.idxmax()
```

Statistics

- idxmax, idxmin메소드의 인수로 축을 전달

```
In [189]: df = pd.DataFrame(np.random.randn(4, 3), columns=['A', 'B', 'C'])

In [190]: df

In [191]: df.idxmin(axis=0)
```

```
In [192]: df.idxmin()
```

```
In [193]: df.idxmax(axis=1)
```

Statistics

■ idxmin, idxmax 메소드 – 첫 번째 매칭 인덱스 반환

```
In [194]: df1 = pd.DataFrame([2, 1, 1, 3, np.nan], columns=['A'], index=list('edcba'))
```

```
In [195]: df1
```

```
In [196] : df1['A'].idxmin()
```

Statistics

■ value_count() 메소드

```
In [197]: data = np.random.randint(0, 7, size=30)

In [198]: a

In [199]: ser1 = pd.Series(data)

In [200]: ser1.value_counts()                    In [201] : pd.value_counts(data)
```

Statistics

■ cut() 함수

```
In [202]: pd.cut(np.array([1, 7, 5, 4, 6, 3]), 3)
```

■ 인수 retbins=True인 경우는 bins를 반환

```
In [203]: pd.cut(np.array([1, 7, 5, 4, 6, 3]), 3, retbins=True)
```

■ bins를 특정 라벨로 할당

```
In [204]: pd.cut(np.array([1, 7, 5, 4, 6, 3]), 3, labels=['bad', 'medium', 'good'])
```

Statistics

- 인수가 labels=False인 cut() 함수

```
In [205]: pd.cut([0, 1, 1, 2], bins=4, labe
ls=False)
```

- 시리즈를 입력으로 전달하는 cut() 함수 – categorical dtype인 시리즈 반환

```
In [206]: ser = pd.Series(np.array([2, 4, 6, 8, 10]), index=['a', 'b', 'c', 'd', 'e']

In [207]: pd.cut(ser, 3)
```

Statistics

■ qcut() 함수

```
In [212]: pd.qcut(range(5), 4)

In [213]: pd.qcut(range(5), 3, labels=['good', medium', 'bad'])

In [214]: pd.qcut(range(5), 4, labels=False)
```

Statistics

■ qcut()과 qcut() 함수의 차이

```
In [215]: pd.cut(np.random.randn(25), 5).value_counts()
In [216]: pd.qcut(np.random.randn(25), 5).value_counts()
```

Function

■ 데이터프레임이나 시리즈의 행, 열 및 요소단위의 함수 적용

→ 테이블 형태의 함수를 적용 : pipe()

→ 행 또는 열 단위의 함수를 적용 : apply()

→ Aggregation API : agg()와 transform()

→ 요소단위로 함수를 적용 : applymap()

■ 테이블 형태의 함수 적용

→ Series, DataFrame, GroupBy에 함수들을 체이닝

→ f(g(h(df), arg1=a), arg2=b, arg3=c) 대신 상용

```
(df.pipe(h)
. . .    .pipe(g, arg1=a)
. . .    .pipe(f, arg2=b, arg3=c)
. . .  )
```

Function

■ 사칙연산의 pipe메소드 사용 – 데이터프레임 생성

→ 생성된 data객체에 2를 더하고, 그 결과에 3으로 나누고 그 결과에 5를 곱하고 그 결과에 1을 빼는 연산

```
In [217]: data = pd.DataFrame([[1, 1, 1], [2, 2, 2], [3, 3, 3]], index=['A', 'B', 'C'], columns=['one', 'two', 'three'])

In [218]: data
```

```
In [219]: def add(data, arg1):
              data1 = data + arg1
              return data1

In [220]: def div(data1, arg2):
              data2 = data1/arg2
              return data2

In [221]: def mul(data2, arg3):
              data3 = data2 * arg3
              return data3

In [222]: def sub(data3, arg4):
              data4 = data3 - arg4
              return data4
```

```
In [223]: (data.pipe(add, arg1=2)
               .pipe(div, arg2=3)
               .pipe(mul, arg3=5)
               .pipe(sub, arg4=1))
```

Function

- 가족 구성원의 성별로 평균나이 구하기 – df 생성
- 성별로 그룹화하고 각 성별로 평균연산 및 모든 열 라벨을 대문자로 한 후 데이터프레임 열에 동적 할당.

```
In [224]: df = pd.DataFrame()


In [225]: df['name'] = ['jsun', 'jin', 'ujung', 'naeun', 'suho']
              df['sex']=['female', 'male', 'female', 'female', 'male']
              df['age'] = [84, 58, 53, 27, 18]

In [226]: df
```

```
In [227]: def mean_age_group(dataframe, col):
                return dataframe.groupby(col).mean()

In [228]: def column(dataframe):
                dataframe.columns = dataframe.columns.str.upper()
                return dataframe
```

Function

- 성별로 그룹화하여 평균값을 구하고 열 라벨을 대문자로 변환
- 행 또는 열 단위의 함수를 적용

  → df 생성

```
In [229]: (df.pipe(mean_age_group, col='sex')
                .pipe(cap_column)
           )
```

```
In [230]: data = [{'one': 1.0, 'two': 1.2}, {'one': 0.5, 'two': 1.1, 'three': 0.7},
                  {'one': 0.7, 'two': 0.9, 'three': -1.6}, {'two': 1.4, 'three': -1.2}]

In [231]: df = pd.DataFrame(data, index=list('abcd'))

In [232]: df
```

Function

- apply메소드를 사용하여 np.mean 적용 – 행 단위 및 axis=1인 열 단위 연산

```
In [233]: df.apply(np.mean)
```

```
In [234]: df.apply(np.mean, axis=1)
```

- df에 apply 메소드를 사용하여 함수들을 적용

```
In [235]: df.apply(lambda x: x.max() - x.min())
```

```
In [236]: df.apply(np.cumsum)
```

Function

■ 문자열 메소드 이름으로 처리되는 apply() 메소드

```
In [237]: df.apply(np.exp)
```

```
In [238]: df.apply('mean')
```

```
In [239]: df.apply('mean', a
xis=1)
```

Function

■ Aggregation API : agg()와 transform()

→ 행에 대해 함수들을 종합 연산하기 위한 df객체 생성

```
 df = pd.DataFrame([[1, 2, 3],
. . .                              [4, 5, 6],
. . .                              [7, 8, 9],
. . .                              [np.nan, np.nan, np.nan]], columns=['A', 'B', 'C'])
 df
 df.agg(['sum', 'min'])
```

Function

■ 각 열에 대해 다른 종합연산 & 행에 대한 종합연산

```
df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
```

```
df.agg('mean', axis='columns')
```

■ DataFrame.aggregate() 또는 DataFrame.agg()- df 생성

```
In [240]: adf = pd.DataFrame(np.random.randn(6, 3),
                             columns=['A','B','C'],
                             index=pd.date_range('7/1/2019',  period
s=6)

In [241]: adf.iloc[2:4] = np.nan
```

```
In [242]: adf
```

Function

■ adf객체의 sum 연산

→ Series에 대한 단일 통합연산

→ 연산 결과는 행으로 나타남

→ 복수의 함수들은 복수의 행들을 남김

```
In [243]: adf.agg(np.sum)
 In [244]: adf.agg('sum')
In [245]: adf.sum()
In [246]: adf.A.agg('sum')
In [247]: adf.agg(['sum'])
In [248]: adf.agg(['sum', 'mean'])
```

Function

- lamda 함수의 전달
- 사용자 정의함수 전달
- 함수의 특정 열 적용
- 리스트 전달

```
In [249]: adf.A.agg(['sum', lambda x: x.mean()])
In [250]: def mymean(x):
              return x.mean()

 In [251]: adf.A.agg(['sum', mymean])
In [252]: adf.agg({'A': 'mean', 'B': 'sum'})
In [253]: adf.agg({'A': ['mean', 'min'], 'B': 'sum'})
```

Function

■ transform() - df데이터의 lambda함수 변환

```
df = pd.DataFrame({'A': range(3), 'B': range(1, 4)})
df                                                    df.transform(lambda x: x + 1)
```

■ transform() – np.abs

```
In [258]: adf.transform(np.abs)
```

Function

■ transform() – 복수의 함수 전달

```
In [260]: adf.transform([np.abs, lambda x: x + 1])
```

Function

- transform() – Series로 복수의 함수 전달하면 DataFrame이 된다.

```
In [261]: adf.A.transform([np.abs, lambda x: x + 1])
```

- transform() – dict 전달하여 열마다 적용

```
In [262]: adf.transform({'A': np.abs, 'B': lambda x: x + 1})
```

Function

■ transform() – list가 있는 dict 전달로 멀티인덱스 발생

```
In [263]: adf.transform({'A': np.abs, 'B': [lambda x: x + 1, 'sqrt']})
```

■ 요소단위로 함수들을 적용

→ DataFrame에 applymap() 적용

```
In [264]: df = pd.DataFrame([[1, 2.12], [3.356, 4.567]])

In [265]: df

In [266]: df.applymap(lambda x: len(str(x)))
```

Function

■ applymap()의 미적용 --- df**2의 연산이 더 빠르다

```
In [267]: df.applymap(lambda x: x**2)
```

```
In [268]: df ** 2
```

■ Series로의 map 적용

```
In [269]: ser = pd.Series(['cat', 'dog', np.nan, 'rabbit'])

In [270]: ser
```

```
In [271]: ser.map({'cat': 'kitten', 'dog': 'puppy'})
```

Function

■ map – 함수 수용

```
In [272]: ser.map('I am a {}'.format)
```

■ map – 인수 na_action

```
In [273]: ser.map('I am a {}'.format, na_action='ignore')
```

Week 5 Outline

- Week 4 Review
- Pandas Overview
- Pandas Basics
  - Data Structure
  - Series & DataFrame
  - Row & Column
  - Index Object
- Pandas Main Feature
  - Binary Operation
  - Statistics
  - Function
- Conclusion

Week 5 Key Takeaway

■ 판다스 기초 부분 학습

→ Series & DataFrame부터 행렬 및 인덱스 객체까지 실습

■ 판다스 주요 기능 학습

→ 바이너리 동작, 통계 및 함수 학습 및 실습

다음 장에서는

- Pandas (2)