데사 B0002

데이터마이닝이해와실습

김 태 완

kimtwan21@dongduk.ac.kr

파이썬 및 Numpy 자료구조

자료구조	특징	예제
리스트	순서 있고, 수정 가능, 여러 자료형 포함	[1, ‘a’, T]
튜플	순서 있고, 수정 불가능	(1, 2, “A”)
딕셔너리	key, value 쌍	{name : Alex, score : 99}
Ndarray	for 문 필요 없이 데이터 배열 처리 가능	a = np.array([1,2,3,4])

- 판다스(*Pandas*) : 파이썬의 데이터 분석 라이브러리
  - 데이터 테이블(data table)을 다루는 도구

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.9	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	48.9	4.9671	2	242.0	17.8	396.9	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.9	5.33	1

시리즈(Series)

데이터프레임 중 하나의 열에 해당하는 데이터의 모음 객체

데이터프레임(DataFrame)

데이터 테이블 전체를 포함하는 객체

- 기본적으로 *Numpy* 사용
  - *Numpy* : Python에서 배열을 다루는 최적의 라이브러리
  - *Pandas*는 *Numpy*를 효율적으로 사용하기 위해 인덱싱, 연산, 전처리 등 다양한 함수 제공

```
pip install pandas
```

Pandas

- 일반적으로 데이터 처리를 직면할 경우 아래와 같은 2차원 배열의 데이터를 가장 많이 접함
  - 예제 : 33명의 성인 여성에 대한 나이에 대한 암 발생 여부

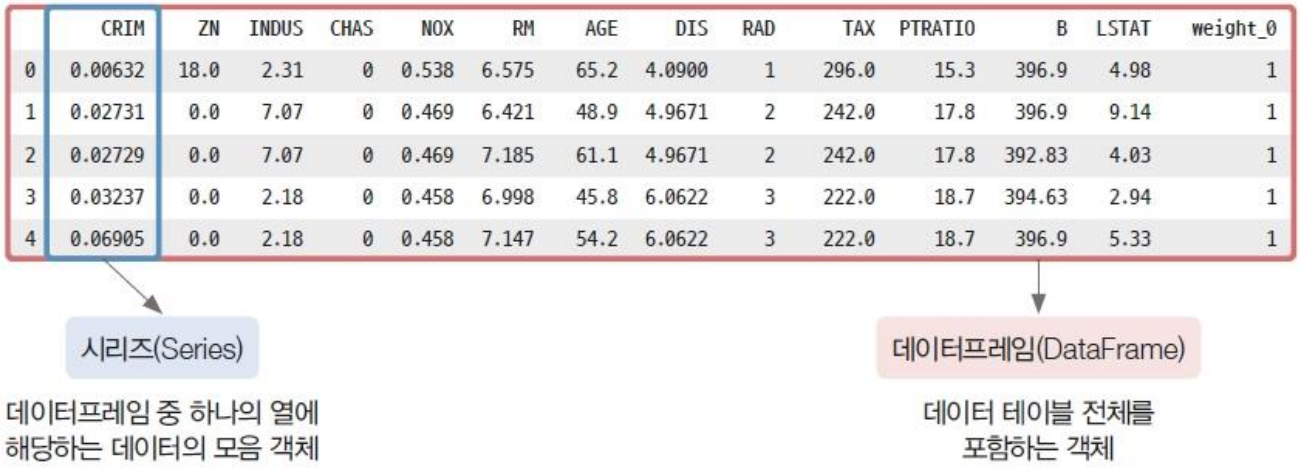
Age	CD	Age	CD	Age	CD
22	0	40	0	54	0
23	0	41	1	55	1
24	0	46	0	58	1
27	0	47	0	60	1
28	0	48	0	60	0
30	0	49	1	62	1
30	0	49	0	65	1
32	0	50	1	67	1
33	0	51	0	71	1
35	1	51	1	77	1
38	0	52	0	81	1

- 현재 업무용 소프트웨어로 가장 인기있는 엑셀 (Excel)을 많이 사용하고 있음
- 하지만 Pandas는 Numpy를 기반으로 하기 때문에 처리속도가 엑셀보다 빠르며, 행과 열로 잘 구조화된 데이터프레임을 제공, 이 데이터프레임을 조작할 수 있는 다양한 함수를 지원함
- Numpy의 2차원 행렬 데이터는 모두 같은 자료값을 가지는 단순한 수치 정보 중심으로만 구성되어 한계 존재
  - 해결 : 시리즈와 데이터프레임이라는 두 가지의 기본적이고도 다재 다능한 데이터 구조를 제공 (Pandas)

Pandas 자료구조

- 각 행과 열은 이름이 부여되며, 행의 이름을 인덱스 (index), 열의 이름을 컬럼 (column)

자료구조	특징
시리즈	Numpy 기반으로 만들어진 1차원 데이터를 위한 자료구조
데이터프레임	Numpy 기반으로 만들어진 2차원 데이터를 위한 자료구조



Pandas

- Import 두 가지 방식

```
from pandas import Series, DataFrame

a = Series()
b = DataFrame()
```

```
import pandas as pd

a = pd.Series()
b = pd.DataFrame()
```

Series 객체 생성

- 1차원 데이터를 위한 자료구조
  - 기본 데이터를 파이썬 자료구조로 표현
  - Series 클래스의 객체를 생성

```
from pandas import Series

a = [1,2,3]
s = Series(a)

print(s)
print(type(s))
```

Series 내부 구조

- 행 번호와 인덱스를 사용하여 데이터 관리
  - 자동으로 부여되는 행 번호 (수정 불가능)
  - 인덱스는 문자 혹은 숫자로 설정 가능
    - 만약 설정하지 않으면 0부터 시작하는 정수로 자동 할당

```
from pandas import Series

a = [1,2,3]
s = Series(a)

print(s)
```

행	인덱스	데이터
0	0	1
1	1	2
2	2	3

보이지 x

Series 내부 구조

- Series 생성 시 index 지정 가능
  - 파이썬 자료구조 딕셔너리의 키 (key) 와 유사한 모습

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(data = a, index=index)
# s = Series(a, index)

print(s)
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 속성

- Series 속성
  - index : Series 객체의 인덱스를 접근 (s.index)
  - values : Series 객체의 값에 접근 (s.values)
  - 딕셔너리의 key(), values() 메소드와는 달리 속성으로 갖고 있음

```
from pandas import Series

a = [1,2,3]
b = ['A', 'B', 'C']
s = Series(a, b)

print(s.index, s.values)
```

Series 예제

- 다음 데이터를 pandas Series 객체로 생성해 보자.

index	values
국어	100
수학	90
영어	85

Series 인덱싱

- iloc 속성을 이용하여 인덱싱
  - Series 객체의 행번호를 사용하여 인덱싱

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

print(s.iloc[0])
print(s.iloc[-1])
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 인덱싱

- loc 속성을 이용하여 인덱싱
  - Series 객체의 인덱스를 사용하여 인덱싱

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

print(s.loc['A'])
print(s.loc['C'])
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 인덱싱

- 대괄호 [ ] 이용하여 인덱싱

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

print(s['A'])
print(s[0])
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 인덱싱

- 연속적이지 않은 여러 개 값을 한 번에 인덱싱
  - 행 번호 또는 인덱스를 파이썬 리스트 자료구조로 구성함
  - iloc 또는 loc 속성에서 해당 리스트 사용
  - 기존 리스트에서는 동작하지 않는 기능

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

print(s.iloc[[0,2]])
print(s.loc[["A","C"]])
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 슬라이싱

- 행번호를 사용하여 슬라이싱
  - iloc[시작 행번호 : 끝 행번호 + 1]
  - 결과도 Series

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

print(s.iloc[0:2])
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 슬라이싱

- 인덱스를 사용하여 슬라이싱
  - loc[시작 인덱스 : 끝 인덱스]
  - iloc와 다르게 끝 인덱스 값 포함
  - 결과도 Series

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

print(s.loc['A':'B'])
```

행	인덱스	데이터
0	“A”	1
1	“B”	2
2	“C”	3

Series 인덱싱 / 슬라이싱 예시

```
list_name = ['이지연','하진영','박주희','김소연']
list_data = [95, 82, 79, 91]
series_data = Series(data = list_data, index=list_name)

print(series_data['하진영'])
print(series_data[1])

print(series_data['하진영':'김소연’])
print(series_data[1:4])

print(series_data[['하진영']])
print(series_data[['하진영','김소연']])
```

Series 값 추가

- 딕셔너리와 유사한 방식으로 값 추가
  - s.loc[인덱스] = 값

```
from pandas import Series

a = [1,2,3]
b = ['A', 'B', 'C']
s = Series(a, b)

s.loc['D'] = 4
print(s)
```

Series 삭제

- drop( ) 메소드
  - 원본은 유지하고 값이 삭제된 Series 객체 반환
  - s.drop('인덱스')
  - s.drop(['인덱스1', '인덱스2'])

```
from pandas import Series

a = [1,2,3]
b = ['A', 'B', 'C']
s = Series(a, b)

s1 = s.drop('B')
print(s1)
s2 = s.drop(['B', 'C'])
print(s2)
print(s)
```

Series 삭제

- drop( ) 메소드
  - 만약 원본에 바로 삭제하고 싶으면, inplace = True 삽입

```
from pandas import Series

a = [1,2,3]
b = ['A', 'B', 'C']
s = Series(a, b)

s.drop('B', inplace=True)
print(s)
```

Series 수정

- Series의 행 번호나 인덱스를 사용하여 값을 수정 가능
  - iloc / loc

```
from pandas import Series

a = [1,2,3]
index = ['A', 'B', 'C']
s = Series(a, index)

s.iloc[0] = 0
s.loc['C'] = 5
print(s)
```

행	인덱스	데이터
0	“A”	0
1	“B”	2
2	“C”	5

Series 연산

- 브로드캐스팅 (Broadcasting)
  - 연산이 Series 객체의 전체 값에 적용됨 : numpy 처럼 반복문 사용 x

```
from pandas import Series

s = Series([10, 20, 30])

print(s+10)
```

Series 연산

- 사칙 연산 : add, sub, mul, div 등의 함수 사용 가능

```
print(series_data + 5)
print(series_data.add(5))

print(series_data - 5)
print(series_data.sub(5))

print(series_data * 5)
print(series_data.mul(5))

print(series_data / 5)
print(series_data.div(5))
```

Series 연산

- 사칙 연산 : 값은 인덱스를 기준으로 사칙 연산 적용 → 반복문 사용 x

```
from pandas import Series

a = Series([10, 20, 30])
b = Series([0, 40, 100])
c = a - b

print(c)
```

Series 연산

- 사칙 연산 : 인덱스가 같은 데이터 간의 사칙 연산 적용
  - 순서와 상관없이 인덱스를 기준으로 계산

index	values
“a”	10
“b”	20
“c”	30

-

index	values
“a”	0
“b”	40
“d”	100

=

index	values
“a”	10
“b”	-20
“c”	NaN
“d”	NaN

```
from pandas import Series

a = Series(data = [10, 20, 30], index=['a','b','c'])
b = Series(data = [0, 40, 100], index=['a','b','d'])
c = a - b

print(c)
```

Series 연산

- 산술 연산 시 데이터가 한쪽에 데이터가 존재하지 않으면 그 결과는 NaN

```
김소연     NaN
김승주     NaN
박주희    42.0
이지상    84.0
정소민     NaN
하진영     NaN
dtype: float64
```

```python
from pandas import Series

list_name1 = ['이지연','하진영','박주희','김소연']
list_data1 = [15, 32, 29, 41]
series_data1 = Series(data = list_data1, index=list_name1)

list_name2 = ['이지연','정소민','박주희','김승주']
list_data2 = [69, 22, 13, 61]
series_data2 = Series(data = list_data2, index=list_name2)

new_series = series_data1 + series_data2
print(new_series)
```

Series 연산

- 비교 연산 : True/False 값이 저장된 Series 객체가 리턴
- 비교 연산을 이용하여 True 값만 리턴 가능

```
from pandas import Series

s = Series([1,2,3,4,5])
cond = s > 3

print(cond)
print(type(cond))
print(s[cond])
```

Series 연산 예제

- a와 b의 차이가 4 초과인 경우의 b의 값을 출력해보자.

```
from pandas import Series

a = Series([30, 14, 24, 50, 12])
b = Series([0, 9, 22, 46, 13])

    # coding here #
```

Series 결측 값

- 우리가 앞으로 데이터를 다루게 될 경우, 정제되지 않은 데이터나 결측 데이터를 많이 접함
  - 결손 값 (missing value)
  - Pandas는 이를 탐지하고 수정하는 함수 제공

```
import numpy as np
import pandas as pd

series = pd.Series([1, 3, np.nan, 4])

print(series.isna())
print(sum(series.isna()))
```

Series 함수

- Series 함수 1 : unique
  - unique() 함수는 중복되는 값을 제거하고, 유일 값만 보여주는 시리즈 객체를 반환

```
from pandas import Series

data = ['라일락', '코스모스', '코스모스', '장미', '코스모스', '장미’]
series_data = Series(data)

print(series_data.unique())
```

Series 함수

- Series 함수 2 : value_counts
  - value_counts() 함수는 항목별 빈도 수를 구한 뒤 내림차순(역순)으로 출력

```
from pandas import Series

data = ['라일락', '코스모스', '코스모스', '장미', '코스모스', '장미']
series_data = Series(data)

print(series_data.value_counts())
```

Series 함수

- Series 함수 3 : isin
  - isin() 함수는 DataBase의 in 키워드와 유사한 개념

```
from pandas import Series

data = ['라일락', '코스모스', '코스모스', '장미', '코스모스', '장미']
series_data = Series(data)

print(series_data.isin(['코스모스']))
```

DataFrame 생성 (1/3)

- Dataframe : Series가 여러 개 합쳐진 자료형
  - 컬럼명을 딕셔너리의 key
  - 데이터는 딕셔너리의 values

	나이	성별	학교
김지연	20	여	A
이태형	21	남	B
전지희	24	여	C

```
from pandas import DataFrame

data = {
    '나이' : [20,21,24],
    '성별' : ['여','남','여'],
    '학교':['A','B','C']
}
index = ['김지연','이태형','전지희']
df = DataFrame(data=data, index=index)
print(df)
```

DataFrame 생성 (2/3)

- 2차원 배열에서 row 단위로 데이터를 리스트로 표현
  - data, index, columns를 각각 리스트로 표현

	나이	성별	학교
김지연	20	여	A
이태형	21	남	B
전지희	24	여	C

```
from pandas import DataFrame

data = [
    [20, '여', 'A'],
    [21, '남', 'B'],
    [24, '여', 'C']
]
index = ['김지연','이태형','전지희']
columns = ['나이', '성별', '학교']
df = DataFrame(data=data, index=index, columns=columns)
print(df)
```

DataFrame 생성 (3/3)

- 2차원 배열에서 row 단위로 데이터를 딕셔너리로 표현
  - 딕셔너리의 키 값이 columns로 자동으로 입력

	나이	성별	학교
김지연	20	여	A
이태형	21	남	B
전지희	24	여	C

```
from pandas import DataFrame

data =[
    {'나이':20, '성별':'여','학교':'A'},
    {'나이':21, '성별':'남','학교':'B'},
    {'나이':24, '성별':'여','학교':'C'}
]
index = ['김지연','이태형','전지희']
df = DataFrame(data=data, index=index)

print(df)
```

DataFrame 인덱싱

열번호	0	1	2
행번호		나이	성별	학교
0	김지연	20	여	A
1	이태형	21	남	B
2	전지희	24	여	C

- Column 선택
  - 대괄호['컬럼이름'] 을 통해서 단일 Column 선택
  - 결과는 Column을 표현하는 Series 객체
  - df['나이']
    - index는 학생 이름, value는 나이

```
print(df['나이'])
```

index	value
김지연	20
이태형	21
전지희	24

DataFrame 인덱싱

열번호		0	1	2
행번호		나이	성별	학교
0	김지연	20	여	A
1	이태형	21	남	B
2	전지희	24	여	C

- 여러 개 Column 선택 : 결과는 Dataframe객체
  - df[[‘나이’, ‘학교’]]

```
print(df[['나이', '학교']])
```

	나이	학교
김지연	20	A
이태형	21	B
전지희	24	C

DataFrame 인덱싱

- 데이터프레임에서 row를 선택할 때는 iloc 또는 loc 속성을 사용
  - loc : 이름 (인덱스)으로 인덱싱
  - iloc : 숫자 (행번호)로 인덱싱

```
label1 = df.loc['김지연']
print(label1)

label2 = df.iloc[0]
print(label2)
```

행번호	index	나이	성별	학교
0	김지연	20	여	A
1	이태형	21	남	B
2	전지희	24	여	C

DataFrame 인덱싱

- 여러 개 row를 선택할 때도 iloc 또는 loc 속성을 사용
  - df.iloc[[0,1]]
  - df.loc[[‘김지연‘, ’이태형’]]

	나이	성별	학교
김지연	20	여	A
이태형	21	남	B
전지희	24	여	C

```
from pandas import DataFrame

data ={'나이':[20,21,24], '성별':['여','남','여'], '학교':['A','B','C']}
index = ['김지연','이태형','전지희']
df = DataFrame(data=data, index=index)

print(df.iloc[[0,1]])
print(df.loc[['김지연','이태형']])
```

DataFrame 슬라이싱

행번호 \ 열번호		0	1	2
		나이	성별	학교
0	김지연	20	여	A
1	이태형	21	남	B
2	전지희	24	여	C

- iloc,loc 속성을 사용하여 row 슬라이싱 : loc 속성은 끝 인덱스 값 포함

```
from pandas import DataFrame

data ={'나이':[20,21,24], '성별':['여','남','여'], '학교':['A','B','C']}
index = ['김지연','이태형','전지희']
df = DataFrame(data=data, index=index)

print(df.iloc[0:2])
print(df.loc['김지연':'이태형'])
```

DataFrame 값 가져오기

- df.iloc[행번호, 열번호]
- df.loc[인덱스, 컬럼명]

행번호 \ 열번호		0	1	2
		나이	성별	학교
0	김지연	20	여	A
1	이태형	21	남	B
2	전지희	24	여	C

```
from pandas import DataFrame

data ={'나이':[20,21,24], '성별':['여','남','여'], '학교':['A','B','C']}
index = ['김지연','이태형','전지희']
df = DataFrame(data=data, index=index)

print(df.iloc[1,1])
print(df.loc['이태형','나이'])
```

DataFrame 영역 가져오기

- df.iloc[행번호 리스트, 열번호 리스트]
- df.loc[인덱스 리스트, 컬럼명 리스트]

행번호 \ 열번호	0	1	2
	나이	성별	학교
0	김지연	20	여	A
1	이태형	21	남	B
2	전지희	24	여	C

```
from pandas import DataFrame

data ={'나이':[20,21,24], '성별':['여','남','여'], '학교':['A','B','C']}
index = ['김지연','이태형','전지희']
df = DataFrame(data=data, index=index)

print(df.iloc[[0,1], [0,1]])
print(df.loc[['김지연','이태형'],['나이','성별']])
```

DataFrame row/column 이름 변경

- Dataframe : 행/열 이름 변경

	Age	Gender	School
학생1	20	여	A
학생2	21	남	B
학생3	24	여	C

```
print(df.index)
print(df.columns)
```

```
df.index = ['학생1', '학생2', '학생3']
df.columns = ['Age','Gender','School’]

print(df.index)
print(df.columns)
```

DataFrame row/column 이름 변경

- Dataframe : 행/열 이름 변경

```
print(df.index)
print(df.columns)
```

	Age	Gender	School
학생1	20	여	A
학생2	21	남	B
학생3	24	여	C

```
df.index = ['학생1', '학생2', '학생3']
df.columns = ['Age','Gender','School’]
```

```
df.rename(columns={'나이':'age','성별':'gender','학교':'school'}, inplace=True)
```

```
print(df)
print(df.index)
print(df.columns)
```

DataFrame row/column 삭제

- Dataframe : 행/열 삭제
  - 행 삭제

	나이	성별	학교
김지연	20	여	A
전지희	24	여	C

```
df2=df.copy()
df2.drop('이태형', inplace=True)
print(df2)
```

  - 열 삭제

	나이	학교
김지연	20	A
이태형	21	B
전지희	24	C

```
df2=df.copy()
df2.drop('성별', axis=1, inplace=True)
print(df2)
```

여러 DataFrame 합치기

- concat
  - 두 개의 데이터프레임을 하나로 합침
  - pd.concat(df_list, axis = 0 or 1, join = ‘outer’ or ‘inner’)
    - axis : 0 (행 늘려서 붙임), 1 (열 늘려서 붙임)
    - join : ‘outer’ (합집합), ‘inner’ (교집합)

```
from pandas import DataFrame
import pandas as pd

df1 = DataFrame([[20,'여','A'], [21,'남','B'], [24,'여','C']],index=['김지연','
이태형','전지희'], columns=['나이','성별','학교'])
df2 = DataFrame([[34,'남','D'], [33,'여','E'], [36,'여','F']],index=['정희재','
이지수','김미연'], columns=['나이','성별','학교'])

df3 = pd.concat([df1, df2], axis=0, join='outer')
print(df3)
```

여러 DataFrame 합치기

- merge()
  - concat()는 판다스의 함수로 결합을 위하여 하나 이상의 데이터프레임을 인자로 받음
  - merge()
    - 데이터프레임의 메소드
    - 두 데이터프레임을 각 데이터에 존재하는 고유값(key)을 기준으로 병합할 때 사용

```
DataFrame.merge(right, how='inner', on=None)
```

right : 현재의 데이터프레임과 결합할 데이터프레임
how : 결합의 방식 'left', 'right', 'inner', 'outer' 가능
on : 조인 연산을 수행하기 위해 사용할 레이블(두 데이터프레임 모두에 존재해야 함)

여러 DataFrame 합치기

- merge()

```
from pandas import DataFrame
import pandas as pd

df1 = DataFrame([[20,'여','A'], [21,'남','B'], [24,'여','C']],index=['김지연','
이태형','전지희'], columns=['나이','성별','학교'])
df2 = DataFrame([[34,'남','D'], [33,'여','E'], [36,'여','F']],index=['정희재','
이지수','김미연'], columns=['나이','성별','학교'])

print('left outer\n', df1.merge(df2, how='left'))
print('right outer\n', df1.merge(df2, how='right'))
print('full outer\n', df1.merge(df2, how='outer'))
print('inner\n', df1.merge(df2, how='inner'))
```

감사합니다

kimtwan21@dongduk.ac.kr

김 태 완