입문_p1

강창현 | aaakch0316@gmail.com

판다스를 배우는 이유

- 데이터 분석의 80~90%가 데이터 수집과 정리
- 데이터 정리란 분석이 가능한 형태로 만드는 것
- 판다스 라이브러리는 데이터를 수집하고 정리하는데 최적화된 도구이다.

판다스 자료구조

> 판다스의 만들어진 목적 : 여러가지 유형의 데이터를 공통의 포맷으로 정리하는 것

시리즈(Series) : 1차원의 데이터 구조 (// 데이터프레임의 열)
데이터프레임(DataFrame) : 2차원의 데이터 구조 (// 표)

파이썬을 판다스로 변환

파이썬 : 리스트, 딕셔너리(, 튜플)
판다스 : 시리즈, 데이터프레임

시리즈(Series)_p4

> index와 value값이 일대일 대응 (실습을 통해 확인)

- 딕셔너리 -> 시리즈 **pandas.Series**

```
import pandas as pd
dic = {'a':1, 'b':2, 'c':3}
sr = pd.Series(dic)

# output
a    1
b    2
c    3
dtype: int64
```

시리즈(Series)

> index와 value값이 일대일 대응 (실습을 통해 확인)

- 리스트 -> 시리즈 **pandas.Series**

```
import pandas as pd
list_data = ["영우", "글로벌", "러닝"]
sr = pd.Series(list_data)
sr

# output
0     영우
1    글로벌
2     러닝
dtype: object
```

시리즈(Series)

> index 옵션 추가하기

```
import pandas as pd
list_data = ["영우", "글로벌", "러닝"]
sr = pd.Series(list_data, index=["하나", "둘", "셋"])
sr

# output
하나     영우
둘     글로벌
셋      러닝
dtype: object
```

시리즈(Series) 원소선택

> Series객체.index
> Series객체.values

```
sr.index
sr.values
```

시리즈(Series) 데이터 추출

> 2가지 추출 방법 :위치인덱스, 인덱스 라벨

```
이름              파이썬
생년월일    2021-01-01
성별                 남
학생여부          True
dtype: object
```

print(sr[0])

print(sr['이름'])

print(sr[[0, 1]])

print(sr[["이름", "성별"]])

시리즈

> 실습

데이터프레임(DataFrame)_p11

> 엑셀(Microsoft Excel)에 작성하는 것과 같이, python의 판다스로 자료정리를 하고 있다고 보면된다. 파이썬의 리스트나 딕셔너리로 된 것을 엑셀과 같이 표 형식으로 만드는 것이 데이터프레임으로 변환하는 것이다.

- 행과 열로 이루어진 2차원 구조의 데이터프레임은 데이터 분석 실무에서 자주 사용
- 여러 개의 시리즈들이 모여서 구성되고 데이터의 열은 시리즈 객체이다.
- 행과 열은 다양하게 불리어 진다. **행** - row, 레코드(record), 관측값(observation), **열** - column, 공통의 속성을 갖는 일련의 데이터, 속성, 범주. **변수(variable)로 활용된다.**

데이터프레임(DataFrame)

- 리스트 => 데이터프레임

```
df = pd.DataFrame([[18, '남','김천고'], [19, '여', '울산고']],
                  index=['진현', '민지'],
                  columns=['나이', '성별', '학교'])
```

- 딕셔너리 => 데이터프레임

> index는 쓰지 않음

```
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
df = pd.DataFrame(dict_data)
```

데이터프레임 인덱스 배열 조회_p15

> 인덱스 배열 조회!!!!!!!!!!!!

행 인덱스 배열 조회 : df.index

열 인덱스 배열 조회 : df.columns _ 정말 많이 쓴다!!!!!!!!!

```
df = pd.DataFrame([[18, '남','김천고'], [19, '여', '울산고']],
                  index=['진현', '민지'],
                  columns=['나이', '성별', '학교'])

print(df.index)
print(df.columns)
print(df.columns[0])

# output
Index(['진현', '민지'], dtype='object')
Index(['나이', '성별', '학교'], dtype='object')
나이
```

데이터프레임 행 조회 (loc, iloc)_ p21

- loc : 인덱스 이름을 기준으로 행을 선택한다.
- iloc : 정수형 위치 인덱스를 기준으로 행을 선택한다.
- 2개 이상의 행 인덱스를 추출하려면 리스트로 넣어서 뽑으면 된다.
- 슬라이스 기법도 사용가능하다.(python과의 차이점을 이해하자)
- 슬라이스 사용 시 loc와 iloc 차이점 중요
- loc = iloc

행 조회(loc, iloc)

> loc : 인덱스 이름을 사용

```
import pandas as pd
exam_data = {'수학':[100, 40, 70], '영어': [50, 70, 90], '생물': [50, 90, 70]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철'])
print(df)
print(df.loc['진현'])

# output
     수학  영어  생물
진현  100  50  50
민지   40  70  90
성철   70  90  70

수학    100
영어     50
생물     50
Name: 진현, dtype: int64
```

행 조회(loc, iloc)

> iloc : 정수형 위치 인덱스

```
import pandas as pd
exam_data = {'수학':[100, 40, 70], '영어': [50, 70, 90], '생물': [50, 90, 70]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철'])
print(df)
print(df.iloc[1])

# output
     수학  영어  생물
진현  100  50  50
민지   40  70  90
성철   70  90  70

수학    40
영어    70
생물    90
Name: 민지, dtype: int64
```

행 범위 슬라이싱 조회(loc, iloc)_p21

- loc : 끝 포함
- iloc : 끝 제외

```
# dataframe
     수학  영어  생물
진현  100  50  50
민지   40  70  90
성철   70  90  70

print(df.loc[:'민지'])
print(df.iloc[:1])

# output
     수학  영어  생물
진현  100  50  50
민지   40  70  90

     수학  영어  생물
진현  100  50  50
```

데이터프레임 열 조회 (loc, iloc)_ p23

> 아래 3가지 코드의 차이점을 이해한다.

```
df.수학
df['수학']  *
df[['수학']]
```

`df.수학` 과 `df['수학']` 은 **같은 결과값**으로 시리즈 객체를 도출한다.
`df[['수학']]` 로 2중 대괄호를 사용하면 시리즈가 아닌 **데이터프레임**을 반환한다.

```
# Tip  생각!
df.loc['수학']   // error
df.loc[:, '수학']  // True_ Series
df.loc[:, ['수학']]  // True_ DataFrame
```

열/행 추가 p31~34

- 열 추가는 간단히 딕셔너리와 비슷하게 추가를 해주면 된다.
- 행 추가는 loc를 이용해서 추가해 준다. 이 때 값 하나만 넣어도 전체 부분이 다 채워진다.

열 추가

> 딕셔너리 추가와 비슷

```
import pandas as pd
exam_data = {'수학':[100, 40, 70]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철'])
print(df)
df['국어'] = 80
print(df)
df['국사'] = [10, 20, 30]
print(df)

# output
     수학
진현  100
민지   40
성철   70

     수학  국어
진현  100  80
민지   40  80
성철   70  80

     수학  국어  국사
진현  100  80  10
민지   40  80  20
성철   70  80  30
```

열 추가_Tip

```
import pandas as pd
exam_data = {'수학':[100, 40, 70]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철'])
print(df)
df.loc[:, '국어'] = 80
print(df)
df.loc[:, '국사'] = [10, 20, 30]
print(df)

# output
     수학
진현  100
민지   40
성철   70

     수학  국어
진현  100  80
민지   40  80
성철   70  80

     수학  국어  국사
진현  100  80  10
민지   40  80  20
성철   70  80  30
```

행 추가

> loc를 이용해서 추가해 준다.

```
import pandas as pd
exam_data = {'수학':[100, 40, 70], '영어': [50, 70, 90], '생물': [50, 90, 70]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철'])
print(df)
df.loc['진현'] = [11, 22, 333]
print(df)
df.loc['진현2'] = [0, 0, 0]
print(df)

#output
     수학  영어  생물
진현  100  50  50
민지   40  70  90
성철   70  90  70

    수학  영어   생물
진현  11  22  333
민지  40  70   90
성철  70  90   70

     수학  영어   생물
진현   11  22  333
민지   40  70   90
성철   70  90   70
진현2   0   0    0
```

행 추가_Tip

```
import pandas as pd
exam_data = {'수학':[100, 40, 70], '영어': [50, 70, 90], '생물': [50, 90, 70]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철'])
print(df)
df.loc['진현', :] = [11, 22, 333]
print(df)
```

실습

> CREATE, READ 실습

INDEX NAME, COLUMNS NAME 변경하기_p15

> df.index, df.columns

```
df = pd.DataFrame([[18, '남','김천고'], [19, '여', '울산고']],
                  index=['진현', '민지'],
                  columns=['나이', '성별', '학교'])
print(df)
df.index=['학생1', '학생2']
print(df)
df.columns=['연령', '남녀', '소속']
print(df)

# output
    나이 성별   학교
진현  18  남  김천고
민지  19  여  울산고

     나이 성별   학교
학생1  18  남  김천고
학생2  19  여  울산고

     연령 남녀   소속
학생1  18  남  김천고
학생2  19  여  울산고
```

INDEX NAME, COLUMNS NAME 변경하기_p16

> inplace=True 를 넣지 않으면, 원본 객체를 직접 수정하는 것이 아니라 새로운 데이터프레임 객체를 반환한다. 따라서 inplace=True를 잊지말고 넣어주자.

`rename 메소드`

```
# df
    나이 성별   학교
진현  18  남  김천고
민지  19  여  울산고

df.rename(columns={'나이':'연령', '성별':'남녀', '학교':'소속'}, inplace=True)
df.rename(index={'진현':'학생1', '민지':'학생2' }, inplace=True)
print(df)

     연령 남녀   소속
학생1  18  남  김천고
학생2  19  여  울산고
```

INDEX NAME, COLUMNS NAME 변경하기_Tip

> `rename`은 일부를 선택하여 변경할 수 있다는 장점이 있지만, 가독성이 좋지 않아 잘 쓰지는 않는다.

- df.rename은 잘 쓰지 않는다.
- df.columns로 주로 변경한다.
- df.index는 사실 변경할 일이 없다. (ex_len(df.index))

```
print(df.columns)

# output
Index(['연령', '남녀', '소속'], dtype='object')
```

열을 인덱스로 지정하기_p39

> 딕셔너리로 들어온 열중에 그 열을 인덱스로 지정하고 싶을 때가 있다. 그때는 `set_index()` 메소드를 적용하여 새로운 인덱스로 지정할 수 있다. 그러면 지정된 인덱스에 덮어쓰기가 된다.

```
exam_data = {'수학':[100, 40, 70, 30], '영어': [50, 70, 90, 80], '생물': [50, 90, 70, 18], '도덕': [88, 68, 58, 77]}
df = pd.DataFrame(exam_data, index=['진현', '민지', '성철', '지산'])
print(df)
df.set_index('수학', inplace=True)
print(df)

# output
     수학 영어 생물 도덕
진현  100  50  50  88
민지   40  70  90  68
성철   70  90  70  58
지산   30  80  18  77


    영어 생물 도덕
수학
100  50  50  88
40   70  90  68
70   90  70  58
30   80  18  77
```

열을 인덱스로 지정하기_Tip

- 추가적으로 `set_index('수학')`과 `set_index(['수학'])`은 같다라는 것을 참고하자.
- inplace=True는 원본값을 직접 고쳐준다. index로 리스트로 담아 set_indx(['수학', '생물'])처럼 적용해도 된다.
- 기초 단계에서는 2개의 인덱스를 적용할 일이 많이 없다는 점을 참고하자.

적용 실습

```
    수학 영어 생물 도덕
이름
진현  100  50  50  88
민지   40  70  90  68
성철   70  90  70  58
지산   30  80  18  77

print(df2.iloc[1, 1:3])
print(df2.loc['민지', '영어':'도덕'])
print(df2.loc['민지', ['영어', '생물']])

# output
영어    70
생물    90
Name: 민지, dtype: int64

영어    70
생물    90
도덕    68
Name: 민지, dtype: int64

영어    70
생물    90
Name: 민지, dtype: int64
```

특정 데이터 값 변환

> df.iloc[1, 1] == df.iloc[1][1]

```
df.iloc[1, 1] = 100
df.iloc[1][1] = 20
```

Q. df.loc[?][?]

전치_p37

> `.transpose()` == `.T` !!!!!!

```
exam_data = {'이름': ['진현', '민지', '성철'],
             '수학':[100, 40, 70], '영어': [50, 70, 90], '생물': [50, 90, 70]}
df = pd.DataFrame(exam_data)
print(df)
df = df.transpose()
print(df)

# output
  이름  수학 영어 생물
0  진현  100  50  50
1  민지   40  70  90
2  성철   70  90  70

      0    1    2
이름  진현 민지 성철
수학  100  40  70
영어   50  70  90
생물   50  90  70
```

행인덱스 수정_p41

> `reindex()` 매우 안 중요!!!!!!!

- reindex() : 행인덱스 재배열
- fill_value 옵션 : NaN값이 있는 경우 지정값으로 변경한다.
- reset_index() : 인덱스 초기화 ()
    - df.reset_index() : 처음상태로 복귀
    - df.reset_index(drop=True) : 인덱스 삭제 (미복귀)

행인덱스 수정_실습예제_ `reindex()`

```
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)

new_index = ['r0', 'r1', 'r2', 'r3', 'r4']
ndf = df.reindex(new_index)
print(ndf)

# output
    c0  c1  c2  c3  c4
r0   1   4   7  10  13
r1   2   5   8  11  14
r2   3   6   9  12  15

     c0   c1   c2    c3    c4
r0  1.0  4.0  7.0  10.0  13.0
r1  2.0  5.0  8.0  11.0  14.0
r2  3.0  6.0  9.0  12.0  15.0
r3  NaN  NaN  NaN   NaN   NaN
r4  NaN  NaN  NaN   NaN   NaN

+ ndf2 = df.reindex(new_index, fill_value=0)
```

인덱스, 컬럼 수정_Tip

> raw 데이터 보전을 위해 아래와 같이 많이 사용한다. (추가_다중공산성)

```
ndf2.index = ["a", "b", "c", "d", "e"]
print(ndf2)

# output
   c0  c1  c2  c3  c4
a   1   4   7  10  13
b   2   5   8  11  14
c   3   6   9  12  15
d   0   0   0   0   0
e   0   0   0   0   0
```

- df.index = 리스트
- df.columns = 리스트

행인덱스 수정_실습예제_`reset_index()`_p42

> 인덱스 초기화

```
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
ndf = df.reset_index()
print(ndf)

#output
    c0  c1  c2  c3  c4
r0   1   4   7  10  13
r1   2   5   8  11  14
r2   3   6   9  12  15

  index  c0  c1  c2  c3  c4
0    r0   1   4   7  10  13
1    r1   2   5   8  11  14
2    r2   3   6   9  12  15
```

행인덱스 수정_실습예제2_ `reset_index()` _p42

> 인덱스 초기화

```
dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df)
ndf = df.reset_index(drop=True)
print(ndf)

#output
    c0  c1  c2  c3  c4
r0   1   4   7  10  13
r1   2   5   8  11  14
r2   3   6   9  12  15

   c0  c1  c2  c3  c4
0   1   4   7  10  13
1   2   5   8  11  14
2   3   6   9  12  15
```

정렬_ `sort_index()` `sort_values` _p43

> 행 인덱스 기준으로 정렬을 한다.

- Series와 DataFrame의 정렬에 대해 알아보자.
- 기본적으로 정렬 할 기준이 **색인**인지, **객체**인지에 따라 사용하는 메서드가 다르다.

시리즈 색인 정렬_ `sort_index()`

- 로우나 컬럼의 색인인 경우는 sort_index 메서드를 사용한다.
- 디폴트 값은 오름차순이지만, 내림차순으로 정리하려면 **ascending=False** 옵션을 넣어준다. // inplace=True 넣어줘야 값이 변경된다.

```
data = pd.Series(range(4), index=['d', 'b', 'c', 'a'])
print(data)
print(data.sort_index())

d    0
b    1
c    2
a    3
dtype: int64

a    3
b    1
c    2
d    0
dtype: int64
```

데이터프레임 색인 정렬_ `sort_index()`

> axis를 활용하여 기준 축을 설정할 수 있다.

```
frame = pd.DataFrame(np.arange(8).reshape(2, 4),
                     index=['three', 'one'],
                     columns=['d', 'a', 'b', 'c'])
print(frame)
print(frame.sort_index())
print(frame.sort_index(axis=1))
```

```
       d  a  b  c
three  0  1  2  3
one    4  5  6  7

       d  a  b  c
one    4  5  6  7
three  0  1  2  3

       a  b  c  d
three  1  2  3  0
one    5  6  7  4
```

시리즈_데이터(객체 값) 정렬_ `sort_values()`

> 정렬 시 Series 객체에서 비어있는 값은 기본적으로 마지막에 위치한다.

```
# NaN은 가장 마지막에 위치한다
data = pd.Series([4, 9, np.nan, 3, -1, np.nan])
print(data)
print(data.sort_values())

# output
0    4.0
1    9.0
2    NaN
3    3.0
4   -1.0
5    NaN
dtype: float64

4   -1.0
3    3.0
0    4.0
1    9.0
2    NaN
5    NaN
dtype: float64
```

데이터프레임_데이터(객체 값) 정렬_ `sort_values()`

```
frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 1, 0]})
print(frame)
# b 컬럼 으로 정렬하기
print(frame.sort_values(by='b'))
# 여러 개의 컬럼 으로 정렬하기
# 리스트의 왼쪽인 'a'를 정렬한 후에 'b'를 정렬한다.
print(frame.sort_values(by=['a', 'b'], ascending=False))
```

```
   b  a
0  4  0
1  7  1
2 -3  1
3  2  0

   b  a
2 -3  1
3  2  0
0  4  0
1  7  1

   b  a
1  7  1
2 -3  1
0  4  0
3  2  0
```

행/열 삭제_ `drop()` _p17

> 데이터프레임의 행 또는 열을 삭제하기

- 행삭제 : axis=0(디폴트)
- 열삭제 ; axis=1
- 다수의 행과 열 삭제 시 list로 넣어준다.
- 원본 객체 변경 시 inplace = True를 추가해 준다.

행/열 삭제_ `drop()` _실습

```
df = pd.DataFrame([[18, '남','김천고'], [19, '여', '울산고']],
                  index=['진현', '민지'],
                  columns=['나이', '성별', '학교'])
print(df)
df.drop('진현', axis=0, inplace=True)
print(df)
df.drop(['나이', '학교'], axis=1, inplace=True)
print(df)

# output
    나이 성별   학교
진현  18  남  김천고
민지  19  여  울산고

    나이 성별   학교
민지  19  여  울산고


     성별
민지  여
```

연산_p46

> 5장 데이터 전처리 초반 부분을 배운 후에 배울 예정입니다.