DATA
SCIENCE LAB

Numpy
& Pandas

23.01.05 / 8기 손승진

Numpy
CONTENTS

Pandas
CONTENTS

Numpy & Pandas를 처음 접하시는 분은
세션에서 설명한 개념을 듣고 첨부한 코드를 실행해보면서 복습을,

Numpy & Pandas를 잘 아시는 분은
이번 기회에 데이터 전처리에 많이 쓰이는 함수들을 메모장 같은 곳에 정리하는 것을 추천 드립니다!

* 하다가 모르는 것이 있으면 슬랙 ' 코딩_수학 어려워요' 채널에 언제든지 질문 남겨주세요!
  혹은 구글에 검색해서 찾아보시는 걸 추천 드립니다!

DATA
SCIENCE LAB

Numpy

23.01.05 / 8기 손승진

0. INTRO

Numpy?

빠르고 효율적인 다차원 배열 객체 ndarray로 데이터 관리

벡터 및 행렬 연산에 있어서 매우 편리한 기능 제공

선형대수 계산, 난수 발생기

Pandas와 matplotlib의 기반이 됨

Numpy는 파이썬에 빠른 배열 처리 기능을 제공하며, 데이터 분석에서는 알고리즘에 사용할 데이터 컨테이너의 역할 수행

1. Array 생성

Array?

빠르고 유연하게 **행렬 연산** 가능

리스트와 비슷하지만, 3가지가 다름

1. 모든 원소가 같은 자료형이어야 함
2. 내부 배열 내 원소 개수가 모두 같아야 함
3. 리스트는 덧셈 시 항목을 이어 붙이는 concatenate를 수행하지만, array는 항목(원소) 간 연산을 수행

1. Array 생성

Array?

```
l1 = [1, 3, 5, 'a', 'b'] # 1, 3, 5는 숫자형, 'a', 'b'는 문자열
a1 = np.array([1, 3, 5, 'a', 'b']) # '1', '3', '5', 'a', 'b'의 문자열로 전환
print(a1) # 하나의 type으로 통일하기 위해 다 문자열로 바뀐 것을 알 수 있음
```

```
['1' '3' '5' 'a' 'b']
```

1. 모든 원소가 같은 자료형이어야 한다

```
l1 = [[1], [3, 5], [2, 4, 6]] # 문제 X
a1 = np.array([[1], [3, 5], [2, 4, 6]]) # 개수가 달라서 array 선언 불가능
#np.array([list([1]), list([3, 5]), list([2, 4, 6])], dtype=object)
print(l1)
print(a1)
```

```
[[1], [3, 5], [2, 4, 6]]
<ipython-input-11-885569368f72>:2: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
  a1 = np.array([[1], [3, 5], [2, 4, 6]]) # 개수가 달라서 array 선언 불가능
```

2. 내부 배열 내 모든 원소 개수가 같아야 한다

1. Array 생성

Array?

```
# 리스트 연산
l1 = [1, 3, 5]
l2 = [2, 4, 6]

print(l1 + l2)
print(l1 * 3)
```

```
[1, 3, 5, 2, 4, 6]
[1, 3, 5, 1, 3, 5, 1, 3, 5]
```

```
# 넘파이 어레이 연산
a1 = np.array([1, 3, 5])
a2 = np.array([2, 4, 6])

print(a1 + a2 )
print(a1 * a2 )
print(a1 + 2)
print(a1 * 3 )
```

```
[ 3  7 11]
[ 2 12 30]
[3 5 7]
[ 3  9 15]
```

3. 리스트는 덧셈 시 항목을 이어 붙이는 concatenate를 수행하지만,
array는 항목(원소) 간 연산을 수행

1. Array 생성

배열 생성하기

(1) 1차원 배열

```
a1 = np.array([1,2,3])
print(a1)
print(a1.shape) # 3개의 원소가 존재하는 1차원 배열 (3,)
print(a1[0], a1[2], a1[1]) # 리스트처럼 인덱싱으로 접근 가능
a1[0] = 10
print(a1)
```

```
[1 2 3]
(3,)
1 3 2
[10  2  3]
```

(2) 2,3차원 배열

```
a2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a2)
print(a2.shape) # 3 * 3의 array (3,3)
print(a2[0], a2[2], a2[2]) # 리스트처럼 인덱싱으로 접근 가능
a2[0] = 20
print(a2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
(3, 3)
[1 2 3] [7 8 9] [7 8 9]
[[20 20 20]
 [ 4  5  6]
 [ 7  8  9]]
```

1. Array 생성

배열 생성하기

ndim(), shape()

- ndim: 배열의 차원 수
- shape: 배열의 차원 표시

```
a2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a2)
print(a2.ndim)
print(a2.shape)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
2
(3, 3)
```

1. Array 생성

배열 생성 함수

(1) 초기화 함수 : zeros(), ones(), full()

- zeros() : 모든 요소를 0으로 초기화
- ones() : 모든 요소를 1로 초기화
- full() : 모든 요소를 지정한 값으로 초기화

```
np.zeros(10)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

np.zeros 예시

```
np.full((3,3),1.5) #3*3의 ndarray를 만들고 안은 1.5로 채워주세요
```

```
array([[1.5, 1.5, 1.5],
       [1.5, 1.5, 1.5],
       [1.5, 1.5, 1.5]])
```

np.full 예시

1. Array 생성

배열 생성 함수

(1) 초기화 함수 zeros() 로 one-hot encoding 해보기 참고

- zeros() : 모든 요소를 0으로 초기화
- ones() : 모든 요소를 1로 초기화
- full() : 모든 요소를 지정한 값으로 초기화

```
import numpy as np
a = np.array([1, 0, 3])
b = np.zeros((a.size, a.max()+1))
b[np.arange(a.size),a] = 1
print(b)
```

```
[[0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]]
```

1. Array 생성

배열 생성 함수

(2) 단위 행렬 생성 eye()

- eye() : 주 대각 원소 1, 나머지 원소 모두 0인 정사각행렬 생성

```
np.eye(3) #identity matrix라서 size만 지정해주면 됨 -> 3*3 identity matrix
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

(3) 같은 shape 행렬 생성 _like()

- _like() : 지정된 배열과 shape이 같은 행렬 생성

```
print(a1)
np.zeros_like(a1) #a1과 같은 shape의 배열 생성
```

```
[1 3 5]
array([0, 0, 0])
```

1. Array 생성

배열 생성 함수

(2) 단위 행렬 생성 eye()로 one-hot encoding 하기 참고

- eye() : 주 대각 원소 1, 나머지 원소 모두 0인 정사각행렬 생성

```
import numpy as np
values = [1, 0, 3]
n_values = np.max(values) + 1
print(np.eye(n_values)[values])
```

```
[[0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]]
```

1. Array 생성

배열 생성 함수

(3) 범위 지정 행렬 생성 arange(), linspace(), logspace() 참고

- arange() :정수 범위로 배열 생성. Numpy 버전의 range 함수
- linspace() : 범위 내에서 균등 간격의 배열 생성
- logspace() : 범위 내에서 균등 간격으로 로그 스케일로 배열 생성

```
np.arange(0,30,2) # 0부터 30까지(30은 포함하지 말고) 2 간격으로 array 생성해줘
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

```
np.linspace(0,1,5) #0부터 1까지 균등하게 5개로 나눠줘
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

```
#np.logspace(start,end,num=개수,endpoint=True,base=10.0,dtype=자료형)
np.logspace(0,4,4,endpoint=False) #0부터 3까지를 등간격으로 나누어 4개의 데이터 생성
```

```
array([   1.,   10.,  100., 1000.])
```

1. Array 생성

배열 생성 함수

(4) 랜덤값 이용하여 행렬 생성 난수 생성 참고

- random.random() : 랜덤한 수의 배열 생성
- random.randint() : 일정 구간의 랜덤 정수의 배열 생성
- random.normal() : 정규 분포에서 랜덤한 수의 배열 생성

1. Array 생성

배열 생성 함수

(4) 랜덤값 이용하여 행렬 생성 난수 생성 참고

```
np.random.random((3,3)) # 랜덤한 실수 뽑아 3*3 array 형태로 반환
```

```
array([[0.15120601, 0.56474386, 0.39539684],
       [0.54524965, 0.48096174, 0.76891843],
       [0.93540104, 0.71791345, 0.15830274]])
```

```
np.random.normal(0,1, size=(3,3)) #정규분포를 고려해서 랜덤하게 뽑아 3*3 array 형태로 반환
```

```
array([[-0.90190939, -0.20943322,  0.32686689],
       [-2.27266352, -1.10775845, -0.72320351],
       [ 1.62021291,  1.07331713, -0.86505673]])
```

```
np.random.randint(0,10,size=(3,3)) # 0부터 10까지의 정수 중 랜덤하게 뽑아 3*3 array 형태로 반환
```

```
array([[6, 1, 8],
       [1, 4, 9],
       [7, 1, 1]])
```

2. 연산

브로드캐스팅

모양이 다른 배열들 간의 연산이 가능해지도록 배열 자동 변환 예시 참고

1	2	3	4
2	5	6	7
8	9	10	11
12	13	14	15

+

1

=

2	3	4	5
3	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
2	5	6	7
8	9	10	11
12	13	14	15

+

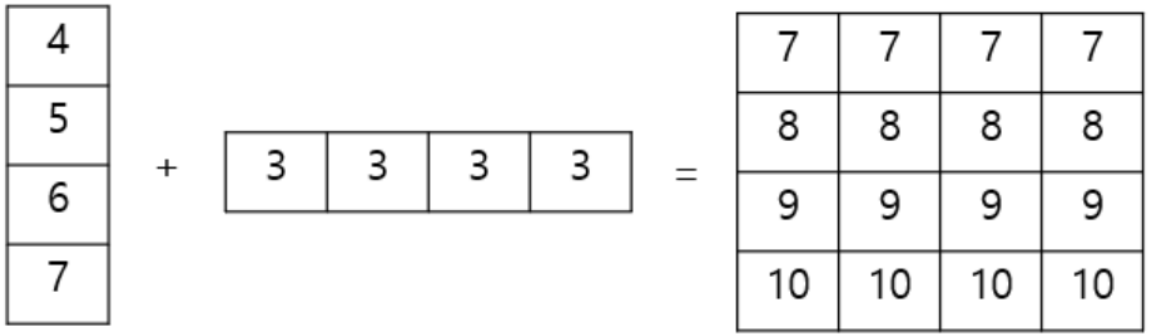
3	3	3	3

=

4	5	6	7
5	8	9	10
11	12	13	14
15	16	17	18

2. 연산

브로드캐스팅



2. 연산

비교 연산

참고

종 류	의 미
==	좌항과 우항의 값이 같다
!=	좌항과 우항의 값이 다르다
>, <	좌항 또는 우항이 크거나 작다
>=, <=	좌항 또는 우항이 크거나 같고 작거나 같다

3. 인덱싱 & 슬라이싱

다차원 배열

- 2차원 이상 배열의 경우에는 ‘,’를 이용하여 원소 하나하나에 접근할 수 있다.
- arr[,] : ‘,’를 기준으로 왼쪽은 행, 오른쪽은 열

```
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr # 설명을 위해 3*3 matrix 생성
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

3. 인덱싱 & 슬라이싱

다차원 배열

- 인덱싱

```
arr[0] # 0번째 행을 가져와라
```

```
array([1, 2, 3])
```

```
arr[0,1] #행을 기준으로 0번째. 열을 기준으로 1번째 가져와라
```

```
2
```

```
arr[0][1] # arr[0] = [1,2,3] 그 중에 1번 index : 2
```

```
2
```

3. 인덱싱 & 슬라이싱

다차원 배열

- 슬라이싱

```
arr[0:2, 0:2] # 행을 기준으로 0,1번째 index
              # 열을 기준으로 0,1번째 index
```

```
array([[1, 2],
       [4, 5]])
```

```
arr[0,:] # 행 0번째 index
         # 열 전부 다
```

```
array([1, 2, 3])
```

3. 인덱싱 & 슬라이싱

다차원 배열

- 팬시 인덱싱(여러 개의 행, 열 추출, 이름은 중요 x)

만약 슬라이싱이 아니라 특정한 2개 이상의 열 혹은 행 불러오고 싶다면 리스트 형태로 넣어주면 된다

Ex) arr[[0,2],[1,1]] : 역시 ',' 왼쪽이 행, 오른쪽이 열

```
arr[[0,2]] # 행을 기준으로 0,2번째 index 추출
```

```
array([[1, 2, 3],
       [7, 8, 9]])
```

```
arr[[0,2],[1,1]] # 행 0.2번째 index
                 # 열 1번째 index
```

```
array([2, 8])
```

3. 인덱싱 & 슬라이싱

다차원 배열

- 불리언 인덱싱

Bool 자료형을 이용해 인덱싱

내가 원하는 조건에 맞게 추출 가능

```
# 불리언 인덱싱
# 짝수만 추출해보자
arr[arr%2==0]
```

```
array([2, 4, 6, 8])
```

- arr에서 짝수만 추출

4. 함수

배열 재구조화

.reshape() : 배열의 형상을 변경(원본 객체 변경 없음)

.resize() : 배열 형태 변경 (원본 객체 변경)

```
n1 = np.arange(1,10)
print(n1)
print(n1.reshape(3,3))
```

```
[1 2 3 4 5 6 7 8 9]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
n2 = np.random.randint(0,10, (2,5))
print(n2)
n2.resize((5,2))
print(n2)
```

```
[[0 6 7 8 0]
 [1 0 0 2 2]]
[[0 6]
 [7 8]
 [0 1]
 [0 0]
 [2 2]]
```

4. 함수

배열 추가

.append() : 배열의 끝에 값 추가

```
a2 = np.arange(1,10).reshape(3,3)
print(a2)
b2 = np.arange(10,19).reshape(3,3)
print(b2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

```
c2 = np.append(a2, b2)
print(c2)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

```
# axis = 0으로 지정
c2 = np.append(a2, b2, axis = 0)
print(c2)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]
 [16 17 18]]
```

```
# axis = 1로 지정
c2 = np.append(a2, b2, axis = 1)
print(c2)
```

```
[[ 1  2  3 10 11 12]
 [ 4  5  6 13 14 15]
 [ 7  8  9 16 17 18]]
```

DATA
SCIENCE LAB

Pandas

23.01.05 / 8기 손승진

0. INTRO

Pandas?

고수준의 자료 구조와 파이썬을 통한 빠르고 쉬운 데이터 분석 도구

Numpy기반 개발

DataFrame을 유용하게 처리할 수 있음! (구조화된 데이터를 빠르고 쉽게 가공할 수 있음)

0. INTRO

설명에 사용할 데이터셋

타이타닉 데이터셋

11개의 컬럼

(승객 id, class, 이름, 성별, 나이, 동승 형제자매 수, 동승 부모/자녀 수, 티켓 번호, 요금, 방 호수, 탑승지)

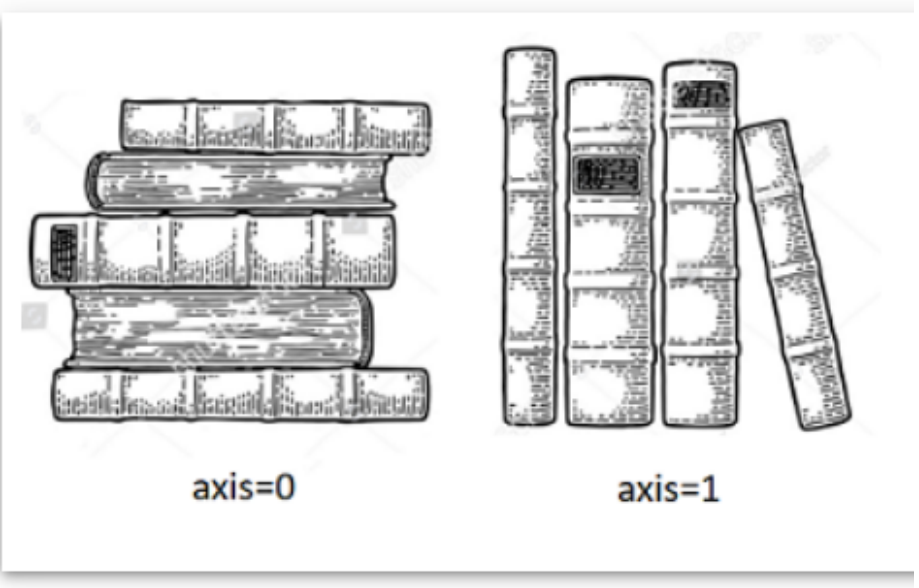
각 행은 탑승자 개인의 정보 의미 *결측치 존재

타이타닉 데이터셋을 예시로 pandas 데이터 기본 전처리 설명

	A	B	C	D	E	F	G	H	I	J	K
1	Passenger	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	892	3	Kelly, Mr.	male	34.5	0	0	330911	7.8292		Q
3	893	3	Wilkes, Mr	female	47	1	0	363272	7		S
4	894	2	Myles, Mr.	male	62	0	0	240276	9.6875		Q
5	895	3	Wirz, Mr. A	male	27	0	0	315154	8.6625		S
6	896	3	Hirvonen,	female	22	1	1	3101298	12.2875		S
7	897	3	Svensson,	male	14	0	0	7538	9.225		S
8	898	3	Connolly, I	female	30	0	0	330972	7.6292		Q
9	899	2	Caldwell, M	male	26	1	1	248738	29		S
10	900	3	Abrahim, M	female	18	0	0	2657	7.2292		C

0. INTRO

axis=0과 axis=1에 대해



- 행 row
- 열 column
- 행방향 axis=0 (2차원 기준). 행이 쌓이는 방향
- 열방향 axis=1 (2차원 기준). 열이 쌓이는 방향

1. DataFrame

Series와 DataFrame

Series
- 일련의 객체를 담을 수 있는 1차원 배열(array)
- 서로 다른 자료형(type)을 동시에 가질 수 있음 ↔ ndarray: 모든 원소가 같은 자료형
- Series가 모여 DataFrame이 된다

DataFrame
- 데이터를 담는 틀(frame)
- 행렬(matrix)과 유사한 구조 : 2차원, 행(row)과 열(column)로 구성

1. DataFrame

pd.read_csv

- 데이터프레임을 처음 받았을 때 이를 파이썬으로 읽어오는 작업
- 옵션 'index_col = 0' : '첫 번째 컬럼은 인덱스로 인식하겠다'

```
# titanic 데이터셋 불러오기
# 마운트 완료되었으면 파일 찾아서 경로 복사해서 pd.read_csv안에 넣으면 됩니다
titanic_df = pd.read_csv('/content/drive/MyDrive/Titanic.csv', index_col = 0)
```

1. DataFrame

pd.read_csv

- 데이터프레임을 처음 받았을 때 이를 파이썬으로 읽어오는 작업
- 옵션 'index_col = 0' : '첫 번째 컬럼은 인덱스로 인식하겠다'

Index_col 따로 안 한 상태

	PassengerId	Pclass	Name
0	892	3	Kelly, Mr. James
1	893	3	Wilkes, Mrs. James (Ellen Needs)
2	894	2	Myles, Mr. Thomas Francis
3	895	3	Wirz, Mr. Albert
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)
...	...	...	...
413	1305	3	Spector, Mr. Woolf
414	1306	1	Oliva y Ocana, Dona. Fermina
415	1307	3	Saether, Mr. Simon Sivertsen
416	1308	3	Ware, Mr. Frederick
417	1309	3	Peter, Master. Michael J

index_col = 0 옵션 부여

PassengerId	Pclass	Name
892	3	Kelly, Mr. James
893	3	Wilkes, Mrs. James (Ellen Needs)
894	2	Myles, Mr. Thomas Francis
895	3	Wirz, Mr. Albert
896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)
...	...	...
1305	3	Spector, Mr. Woolf
1306	1	Oliva y Ocana, Dona. Fermina
1307	3	Saether, Mr. Simon Sivertsen
1308	3	Ware, Mr. Frederick
1309	3	Peter, Master. Michael J

2. 데이터 살펴보기

데이터 살펴보기

- shape : 데이터프레임 크기
- dtypes :각 열 데이터 타입 확인 (데이터 타입을 바꾸고 싶다면 astype)
- isnull().sum() :열별로 결측치가 몇 개 있는지 파악 가능

```
titanic_df.shape
```

```
(418, 10)
```

```
titanic_df.dtypes
```

```
Pclass        int64
Name         object
Sex          object
Age         float64
SibSp         int64
Parch         int64
Ticket       object
Fare        float64
Cabin        object
Embarked     object
dtype: object
```

```
titanic_df.isnull().sum()
```

```
Pclass        0
Name          0
Sex           0
Age          86
SibSp         0
Parch         0
Ticket        0
Fare          1
Cabin       327
Embarked      0
dtype: int64
```

2. 데이터 살펴보기

데이터 살펴보기 (참고만 하세요!)

- Pandas로 불러온 결과도 인덱싱, 슬라이싱 가능!

```
titanic_df.isnull().sum()
```

```
Pclass        0
Name          0
Sex           0
Age          86
SibSp         0
Parch         0
Ticket        0
Fare          1
Cabin       327
Embarked      0
dtype: int64
```

```
titanic_df.isnull().sum()['Cabin']
```

```
327
```

2. 데이터 살펴보기

데이터 살펴보기

- rename : 열 이름 재설정
- set_index : 열 인덱스로 설정
- reset_index : 인덱스 리셋

```
titanic_df.rename(columns={"Name":"Passenger_Name", "Ticket":"Ticket_Num"}, inplace=False)
```

PassengerId	Pclass	Passenger_Name	Sex	Age	SibSp	Parch	Ticket_Num
892	3	Kelly, Mr. James	male	34.5	0	0	330911
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272

- Rename을 이용해서
  'Name' 열 이름을 'Passenger_Name'으로, 'Ticket' 열 이름을 'Ticket_Num'으로 바꾼 코드

2. 데이터 살펴보기

데이터 살펴보기

- dropna : 결측치 제거 (axis = 0 하면 결측치가 있는 행 제거)
- fillna : 결측치 채우기 (axis = 1 하면 열을 기준으로 채워짐)

```
mean_age = titanic_df['Age'].mean()
print(mean_age)
titanic_df.fillna(mean_age,axis=1, inplace=False)
```

```
30.272590361445783
```

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch
892	3	Kelly, Mr. James	male	34.50000	0	0
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.00000	1	0
894	2	Myles, Mr. Thomas Francis	male	62.00000	0	0
895	3	Wirz, Mr. Albert	male	27.00000	0	0
896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.00000	1	1

*Pandas로 다양한 수학, 통계 매서드를 이용할 수 있습니다.
Ex) mean,count,min,max,quantile,sum,var,cov…

왼쪽의 예시 코드는 결측치를 'Age'열의 평균으로 채워달라는 코드입니다

2. 데이터 살펴보기

데이터 살펴보기

- describe : 기초 통계량

```
titanic_df.describe()
```

	Pclass	Age	SibSp	Parch	Fare
count	418.000000	332.000000	418.000000	418.000000	417.000000
mean	2.265550	30.272590	0.447368	0.392344	35.627188
std	0.841838	14.181209	0.896760	0.981429	55.907576
min	1.000000	0.170000	0.000000	0.000000	0.000000
25%	1.000000	21.000000	0.000000	0.000000	7.895800
50%	3.000000	27.000000	0.000000	0.000000	14.454200
75%	3.000000	39.000000	1.000000	0.000000	31.500000
max	3.000000	76.000000	8.000000	9.000000	512.329200

2. 데이터 살펴보기

데이터 살펴보기

- value_counts : 개수 세기

```
titanic_df['Pclass'].value_counts()
```

```
3    218
1    107
2     93
Name: Pclass, dtype: int64
```

1등석 107명
2등석 93명
3등석 218명

```
titanic_df[['Pclass','Sex']].value_counts()
```

```
Pclass  Sex
3       male      146
        female     72
2       male       63
1       male       57
        female     50
2       female     30
dtype: int64
```

1등석 & 남자 57명
1등석 & 여자 50명
2등석 & 남자 63명
2등석 & 여자 30명
3등석 & 남자 146명
3등석 & 여자 72명

2. 데이터 살펴보기

데이터 살펴보기

- sort_values : 정렬

.sort_values(by = '' , ascending=False) 내림차순

```
titanic_df.sort_values(by='Age', ascending=False, inplace=False)
```

PassengerId	Pclass	Name	Sex	Age
988	1	Cavendish, Mrs. Tyrell William (Julia Florence...	female	76.0
973	1	Straus, Mr. Isidor	male	67.0
1128	1	Warren, Mr. Frank Manley	male	64.0
1197	1	Crosby, Mrs. Edward Gifford (Catherine Elizabe...	female	64.0
1071	1	Compton, Mrs. Alexander Taylor (Mary Eliza Ing...	female	64.0

타이타닉 승객 중 제일 나이가
많은 사람은 76세였군요…

```
titanic_df.sort_values(by=['Pclass','Age'], ascending=(False,False), inplace=False)
```

PassengerId	Pclass	Name	Sex	Age
1044	3	Storey, Mr. Thomas	male	60.5
917	3	Robins, Mr. Alexander A	male	50.0
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0
911	3	Assaf Khalil, Mrs. Mariana (Miriam")"	female	45.0
1201	3	Hansen, Mrs. Claus Peter (Jennie L Howard)	female	45.0

여러 개를 기준으로도 정렬 가능!

Pclass 기준으로 먼저 내림차순 정렬하고, Pclass가
동점인 경우에 Age를 기준으로 내림차순 정렬

2. 데이터 살펴보기

데이터 살펴보기

- nunique: 고유한 값 개수

```
titanic_df.nunique()
```

```
Pclass        3
Name        418
Sex           2
Age          79
SibSp         7
Parch         8
Ticket      363
Fare        169
Cabin        76
Embarked      3
dtype: int64
```

3. 인덱싱 & 슬라이싱

인덱싱 & 슬라이싱

DataFrame 인덱싱하기
*색인(index)로 인덱싱할 때는 끝점을 포함한다는 것에 주의!

1. Column 이름을 이용한 인덱싱
Ex) titanic_df['Pclass'] : Pclass에 해당하는 열 출력. 행은 다

2. 행을 이용한 인덱싱
* 행 단위로 인덱싱할 때는 항상 슬라이싱 해야 함!
Ex) titanic_df[:2]: 0,1번째 행 출력. 열은 다

3. 인덱싱 & 슬라이싱

인덱싱 & 슬라이싱

```
titanic_df['Pclass'] #Pclass 열 다 보여주는 코드
```

```
PassengerId
892     3
893     3
894     2
895     3
896     3
       ..
1305    3
1306    1
1307    3
1308    3
1309    3
Name: Pclass, Length: 418, dtype: int64
```

Pclass 열 추출

```
titanic_df[:2] #0,1번째 행을 보여주는 코드
```

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S

0,1번째 행 추출

3. 인덱싱 & 슬라이싱

불리언 인덱싱

DataFrame에서 원하는 조건을 만족하는 행과 열을 추출하고 싶을 때 사용
df[원하는 조건]
Ex) titanic_df[titanic_df['Pclass'] == 1]
: 1등석 승객의 정보 추출. 열은 다

3. 인덱싱 & 슬라이싱

불리언 인덱싱 코드 예시

```
titanic_df[titanic_df['Pclass'] == 1] # 불리언 인덱싱
```

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
903	1	Jones, Mr. Charles Cresson	male	46.0	0	0	694	26.0000	NaN	S
904	1	Snyder, Mrs. John Pillsbury (Nelle Stevenson)	female	23.0	1	0	21228	82.2667	B45	S
906	1	Chaffee, Mrs. Herbert Fuller (Carrie Constance...	female	47.0	1	0	W.E.P. 5734	61.1750	E31	S
912	1	Rothschild, Mr. Martin	male	55.0	1	0	PC 17603	59.4000	NaN	C
914	1	Flegenheim, Mrs. Alfred (Antoinette)	female	NaN	0	0	PC 17598	31.6833	NaN	S

3. 인덱싱 & 슬라이싱

인덱싱 & 슬라이싱

DataFrame 인덱싱하기
4. iat,loc,iloc 매서드
- 이 중 많이 쓰이는 loc 매서드에 대해 소개
- df.loc[추출하고 싶은 행 이름, 추출하고 싶은 열 이름]
- ':' 는 모든 행 혹은 모든 열을 추출하고 싶을 때 사용
- 슬라이싱도 가능

*색인(index)로 인덱싱할 때는 끝점을 포함한다는 것에 주의!

Ex) titanic_df.loc[:,'Name'] : 모든 행과 'Name' 열 추출
titanic_df.loc[:,['Name','Pclass']] : 모든 행과 'Name', 'Pclass' 열 추출
titanic_df.loc[:,'Name':'Age'] 모든 행과 'Name'부터 'Age' 열까지 추출('Name','Sex','Age')

3. 인덱싱 & 슬라이싱

loc 코드 예시

여러 개의 행 인덱싱(리스트 형태로 받으면 됨)

```
titanic_df.loc[:,['Name','Pclass']]
```

PassengerId	Name	Pclass
892	Kelly, Mr. James	3
893	Wilkes, Mrs. James (Ellen Needs)	3
894	Myles, Mr. Thomas Francis	2
895	Wirz, Mr. Albert	3
896	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	3
...	...	...
1305	Spector, Mr. Woolf	3
1306	Oliva y Ocana, Dona. Fermina	1
1307	Saether, Mr. Simon Sivertsen	3
1308	Ware, Mr. Frederick	3
1309	Peter, Master. Michael J	3

슬라이싱

```
titanic_df.loc[:,'Name':'Age']
```

PassengerId	Name	Sex	Age
892	Kelly, Mr. James	male	34.5
893	Wilkes, Mrs. James (Ellen Needs)	female	47.0
894	Myles, Mr. Thomas Francis	male	62.0
895	Wirz, Mr. Albert	male	27.0
896	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0
...	...	...	...
1305	Spector, Mr. Woolf	male	NaN
1306	Oliva y Ocana, Dona. Fermina	female	39.0
1307	Saether, Mr. Simon Sivertsen	male	38.5
1308	Ware, Mr. Frederick	male	NaN
1309	Peter, Master. Michael J	male	NaN

418 rows × 3 columns

3. 인덱싱 & 슬라이싱

loc 코드 예시

loc 이용해서 불리언 인덱싱 하면 조건에 맞는 행과 열 중에서도 원하는 열만 추출 가능

```
titanic_df.loc[titanic_df['Pclass'] == 1,'Name']
# loc를 이용하면 원하는 조건을 만족하는 행과 열까지 지정해서 추출 가능
```

```
PassengerId
903                              Jones, Mr. Charles Cresson
904          Snyder, Mrs. John Pillsbury (Nelle Stevenson)
906      Chaffee, Mrs. Herbert Fuller (Carrie Constance...
912                                Rothschild, Mr. Martin
914                    Flegenheim, Mrs. Alfred (Antoinette)
                                ...
1295                               Carrau, Mr. Jose Pedro
1296                           Frauenthal, Mr. Isaac Gerald
1299                             Widener, Mr. George Dunton
1303       Minahan, Mrs. William Edward (Lillian E Thorpe)
1306                           Oliva y Ocana, Dona. Fermina
Name: Name, Length: 107, dtype: object
```

Pclass = 1을 만족하는 DataFrame에서 행은 다, 열은 'Name'만 추출

3. 인덱싱 & 슬라이싱

인덱싱 & 슬라이싱

DataFrame 인덱싱 이용해서 새로운 컬럼 만들기
df['만들고 싶은 새로운 열 이름'] = 값
Ex) titanic_df['Survived'] = 1 : 'Survived'라는 이름의 새로운 열을 만들고 값은 다 1로 넣어주세요

```
titanic_df['Survived'] = 1
titanic_df.head(2)
```

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	1
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1

*head(2)는 위에서 2개의 행만 보여달라는 코드입니다.

3. 인덱싱 & 슬라이싱

인덱싱 & 슬라이싱

DataFrame 행, 열 삭제
df.drop('삭제하고 싶은 행, 열 이름' axis=0(행 삭제) or 1(열 삭제))
Ex) titanic_df.drop('Survived', axis=1) : 'Survived'를 열 방향으로 없애주세요 (='Survived' 열 없애주세요)

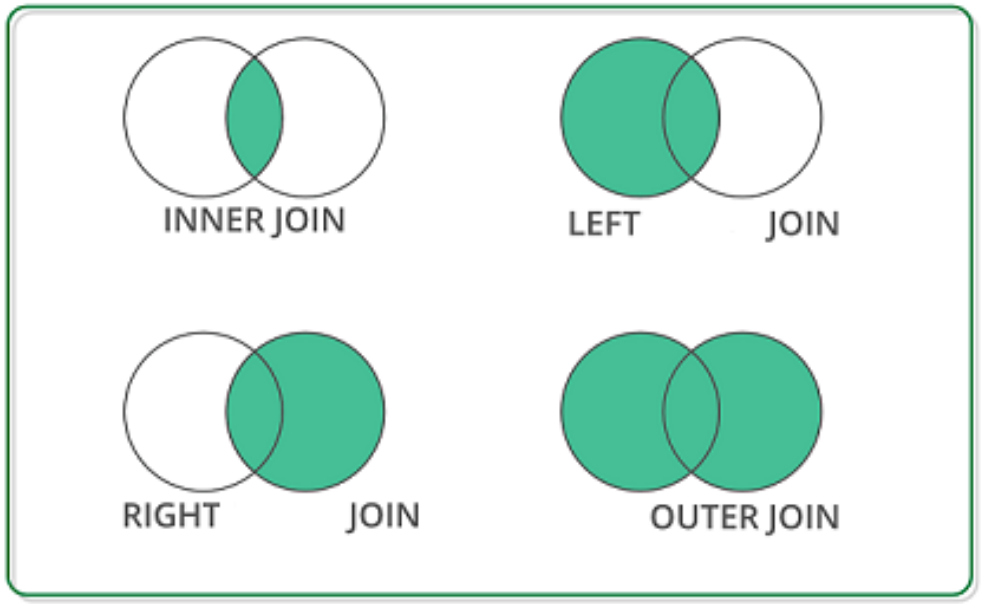
```
# Survived 열 삭제
titanic_df = titanic_df.drop('Survived', axis = 1)
titanic_df.head(2)
```

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId										
892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S

*head(2)는 위에서 2개의 행만 보여달라는 코드입니다.

4. DataFrame 결합

데이터 프레임 결합(merge,join)



4. DataFrame 결합

데이터 프레임 결합(merge,join)

```
# 데이터 병합 설명을 위해 데이터 프레임을 직접 만들어보겠습니다
# 직접 데이터 프레임 만드는 코드입니다
# 참고해주시면 감사하겠습니다
## list로 생성 & index(행이름) 및 columns(열이름) 지정
df_left = pd.DataFrame([['K0', 'A0', 'B0'], ['K1', 'A1', 'B1'],['K2','A2','B2'],['K3','A3','B3']],
                       index=[0,1,2,3],
                       columns=['KEY', 'A', 'B'])
df_left
```

	KEY	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
# 데이터 병합 설명을 위해 데이터 프레임을 직접 만들어보겠습니다
# 직접 데이터 프레임 만드는 코드입니다
# 참고해주시면 감사하겠습니다
## dictionary 이용해서 생성
df_right = pd.DataFrame({'KEY' : ['K2','K3','K4','K5'],
                         'C' : ['C2', 'C3', 'C4', 'C5'],
                         'D' : ['D2', 'D3', 'D4', 'D5']})
df_right
```

	KEY	C	D
0	K2	C2	D2
1	K3	C3	D3
2	K4	C4	D4
3	K5	C5	D5

설명을 위한 데이터 프레임 생성 (df_left, df_right)

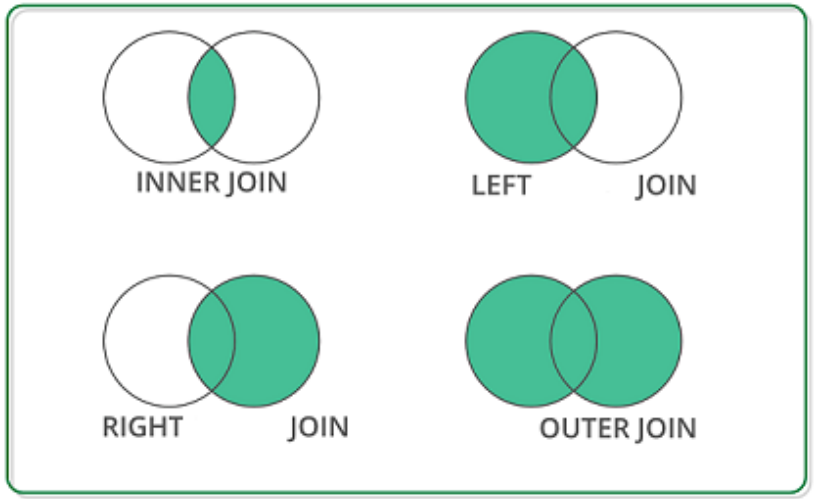
4. DataFrame 결합

데이터 프레임 결합(merge, join)

1. Merge

merge(): 각 DataFrame에서 기준이 되는 key를 지정한 후, 그 key를 기준으로 테이블을 병합

- inner join: 교집합. 해당 key의 데이터에서 두 테이블 모두에 그 데이터가 존재하는 행만 병합
- left(right) join: 한 테이블은 모두 보존하고, 그 테이블에 존재하는 key의 데이터들의 행만 다른 테이블에서 가져옴
- outer join: 두 테이블의 key 데이터들의 행을 전부 가져옴



4. DataFrame 결합

데이터 프레임 결합(merge, join)

Merge 기본 옵션

option	description
left	왼쪽 DataFrame을 가리킴
right	오른쪽 DataFrame을 가리킴
how	조인 방법 설정 - left: 왼쪽 DataFrame 기준 - right: 오른쪽 DataFrame 기준 - inner: 교집합 - outer: 합집합
on	결합의 기준이 되는 열 (key) 직접 설정
left_on	왼쪽 DataFrame의 key 직접 설정
right_on	오른쪽 DataFrame의 key 직접 설정
left_index	왼쪽 DataFrame의 index를 key로 사용
right_index	오른쪽 DataFrame의 index를 key로 사용

4. DataFrame 결합

데이터 프레임 결합(merge, join)

Merge

Ex) pd.merge(df_left, df_right, how='left', on='KEY')

```
pd.merge(df_left, df_right,
         how='left', #df_left 기준
         on = 'KEY') #KEY 변수 기준

# df_left의 KEY들이 유지되고(K0,K1,K2,K3) 거기에 해당되는 df_right를 붙임
# df_right의 KEY는 고려되지 x
# df_right는 K0와 K1을 인덱스로 가지고 있지 않아서 해당 부분은 NaN 값이 됨
```

	KEY	A	B	C	D
0	K0	A0	B0	NaN	NaN
1	K1	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

4. DataFrame 결합

데이터 프레임 결합(merge, join)

Merge 과정 (Left join)

Ex) pd.merge(df_left, df_right, how='left', on='KEY')

	KEY	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

df_left

Left Join

	KEY	C	D
0	K2	C2	D2
1	K3	C3	D3
2	K4	C4	D4
3	K5	C5	D5

df_right

	KEY	A	B	C	D
0	K0	A0	B0	NaN	NaN
1	K1	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

4. DataFrame 결합

데이터 프레임 결합(merge, join)

Join
결합의 기준이 색인(index)일 때는, join 함수를 사용하는 것이 편리
Ex) df_left.join(df_right2, how='inner')
*'outer'를 쓰면 df_left와 df_right의 모든 색인을 기준으로 join이 됨

	KEY	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

df_left

Join

	C	D
0	C2	D2
1	C3	D3
2	C4	D4
3	C5	D5

df_right2

	KEY	A	B	C	D
0	K0	A0	B0	C2	D2
1	K1	A1	B1	C3	D3
2	K2	A2	B2	C4	D4
3	K3	A3	B3	C5	D5

5. DataFrame 그룹화

데이터 프레임 그룹화(groupby)

groupby(): 공통된 값을 기준으로 그룹화. 요약정보 제공
*groupby를 하고 나면 꼭 그 그룹을 대표할 수 있는 요약통계량을 추출할 것! (mean, max…)
Ex) titanic_df.groupby('Pclass').mean()
: Pclass를 기준으로 그룹화를 진행하고, 각 그룹별로 평균값을 계산해주세요(계산 가능한 열만 계산)

```
group_df = titanic_df.groupby('Pclass')
group_df.mean()
```

Pclass	Age	SibSp	Parch	Fare
1	40.918367	0.476636	0.383178	94.280297
2	28.777500	0.376344	0.344086	22.202104
3	24.027945	0.463303	0.417431	12.459678

타이타닉 1등석 고객의 평균 나이는 약 40세,
평균 요금은 $94.28!

5. DataFrame 그룹화

데이터 프레임 그룹화(groupby)

groupby 기준 여러 개로도 가능
Ex) titanic_df.groupby(['Pclass', 'Sex']).mean()
: Pclass랑 성별을 기준으로 그룹을 만들고, 각 그룹의 열 별로 계산 가능한 평균 추출

```
group_df2 = titanic_df.groupby(['Pclass','Sex'])
group_df2.mean()
```

		Age	SibSp	Parch	Fare
Pclass	Sex				
1	female	41.333333	0.560000	0.500000	115.591168
	male	40.520000	0.403509	0.280702	75.586551
2	female	24.376552	0.533333	0.766667	26.438750
	male	30.940678	0.301587	0.142857	20.184654
3	female	23.073400	0.583333	0.597222	13.735129
	male	24.525104	0.404110	0.328767	11.826350

타이타닉 1등석 여성의 평균 나이는 약 41세,
요금은 평균 $115.59!

5. DataFrame 그룹화

데이터 프레임 그룹화(groupby)

*groupby 팁
여러 개를 기준으로 groupby를 했을 때는 reset_index를 하는 것이 편함

```
group_df2.mean().reset_index()
```

	Pclass	Sex	Age	SibSp	Parch	Fare
0	1	female	41.333333	0.560000	0.500000	115.591168
1	1	male	40.520000	0.403509	0.280702	75.586551
2	2	female	24.376552	0.533333	0.766667	26.438750
3	2	male	30.940678	0.301587	0.142857	20.184654
4	3	female	23.073400	0.583333	0.597222	13.735129
5	3	male	24.525104	0.404110	0.328767	11.826350

6. Write

데이터 프레임 저장(to_csv)

- to_csv : DataFrame을 csv, txt 파일로 저장
- df.to_csv("file path/file name", sep=',' , na_rep='', index=True, header=True)

```
titanic_df.to_csv('/content/drive/MyDrive/titanic_df', sep=',',na_rep='',index=True, header=True)
```

Summary

Numpy & Pandas를 처음 접하시는 분은

세션에서 설명한 개념을 듣고 첨부한 코드를 실행해보면서 복습을,

Numpy & Pandas를 잘 아시는 분은

이번 기회에 데이터 전처리에 많이 쓰이는 함수들을 메모장 같은 곳에 정리하는 것을 추천 드립니다!

Ex) is.null.sum(), sort_values(), value_counts(), 인덱싱, 슬라이싱, merge, groupby…

* 하다가 모르는 것이 있으면 슬랙 ' 코딩_수학 어려워요' 채널에 언제든지 질문 남겨주세요!

  혹은 구글에 검색해서 찾아보시는 걸 추천 드립니다!

Reference

- 7기 전혜령님 Numpy 세션 강의안
- 7기 박지은님 Pandas 세션 강의안
- 한솔지 교수님 'R과 파이썬 프로그래밍' 강의안
- 정승환 교수님 '공급사슬애널리틱스' 강의안
- 구글

감사합니다

DATA SCIENCE LAB

발표자 @@@ 010-1234-5678
E-mail: @@@@gmail.com