BITAmin
1주차 Session

7기 교육부 이효석

BITAMIN

분석프로그래밍1

D DataFrame

S Series

SD Summarizing Data

AD Assign DataFrame

IS Indexing and Slicing

BI Boolean Indexing

HM Handling Missing Data

BITAmin

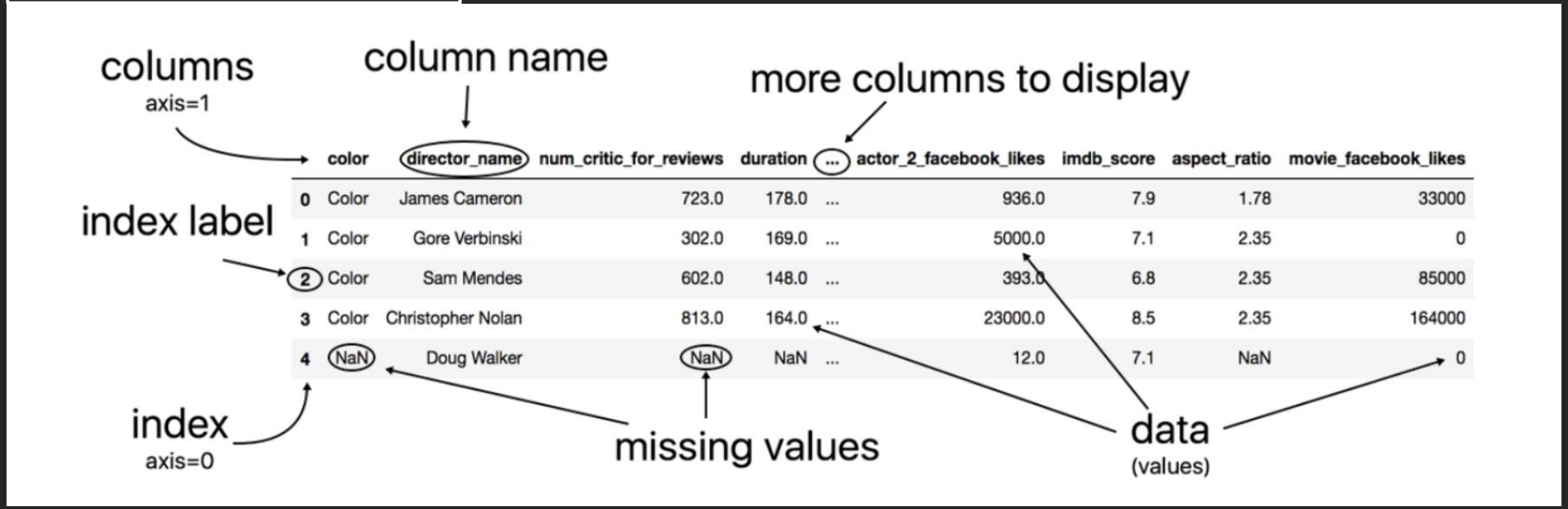
BITAMIN

DataFrame

B DataFrame

↳ A DataFrame represents a tabular, spreadsheet-like data structure containing an ordered collection of columns, each of which can be a different value type (numeric,string, boolean, etc.)

데이터프레임의 구조



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

B DataFrame

파이썬 문법으로 보는 데이터프레임 구조

```
In [1]:
data = {'state': ['서울', '서울', '서울', '부산', '부산', '부산'],
        'year': [2014, 2016, 2018, 2014, 2016, 2018],
        'pop': [997.5, 984.3, 970.5, 345.2, 344.7, 340.0]}
data
```

```
Out[1]:
{'state': ['서울', '서울', '서울', '부산', '부산', '부산'],
 'year': [2014, 2016, 2018, 2014, 2016, 2018],
 'pop': [997.5, 984.3, 970.5, 345.2, 344.7, 340.0]}
```

```
In [4]:
df = DataFrame(data)
df
```

Out[4]:

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

1.Dictionary 자료형을 통해 데이터프레임을 만들 수 있다.

2.Column이 list라는 점을 기억하자.

3.Key와 Value가 어디에 대응되는지 확인하자.

CSV로 데이터프레임 읽고 쓰기

```
In [9]:  df.to_csv('예제.csv', encoding = 'cp949', index=False)

In [10]: pd.read_csv('예제.csv', encoding = 'cp949')
```

Out[10]:

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

1.CSV란?

2.데이터프레임을 CSV로 내보내기

3.CSV 파일을 읽어오기

4. encoding = ‘cp949’ 의 역할?

5. 파일을 내보낼 때 index = False 를 하는 이유

B DataFrame

데이터프레임의 주요 메소드들

Attributes or Methods	Results
df.index	Array-like row labels
df.columns	Array-like column labels
df.values	Numpy array, data
df.shape	(n_rows, m_cols)
df.dtypes	Type of each column
len(df)	Number of rows
df.head(), df.tail()	First/last rows
df.describe()	Summary stats
df.info()	Summary of a DF

1.index, columns, values – 주로 응용되어 사용

2. shape – 행렬계산시 형태의 중요성

3. dtypes – 데이터 살펴보기 → astype() 활용 가능 - datetime의 경우

4. len – 행의 길이 ⟷ size – 행의 개수 * 열의 개수 = 값의 개수

5.head(), tail() – 데이터 확인

6.describe() - 기술통계량

7.info() – 전체적인 데이터 살펴보기 **

B DataFrame

1.index, columns, values – 주로 응용되어 사용

실습

```
list(df.index)
```

```
[0, 1, 2, 3, 4, 5]
```

```
list(df.columns)
```

```
['state', 'year', 'pop']
```

```
df.values
```

```
array([['서울', 2014, 997.5],
       ['서울', 2016, 984.3],
       ['서울', 2018, 970.5],
       ['부산', 2014, 345.2],
       ['부산', 2016, 344.7],
       [nan, nan, nan]], dtype=object)
```

이렇게 열의 종류가 다양할 때
열의 이름들을 한번에 보고싶을 때

```
titanic= pd.read_csv('titanic.csv')
```

```
list(titanic.columns)
```

```
['PassengerId',
 'Survived',
 'Pclass',
 'Name',
 'Sex',
 'Age',
 'SibSp',
 'Parch',
 'Ticket',
 'Fare',
 'Cabin',
 'Embarked']
```

2. shape – 행렬계산시 형태의 중요성

실습

```
df.shape
(6, 3)

df.shape[0]
6

df.shape[1]
3
```

1. 데이터 병합시 shape을 맞춰서 진행

2. 출력값이 튜플이므로 indexing을 통해 열의 개수, 행의 개수 확인 가능

3. 선형대수에서 행렬계산시 확인 필요.

3. dtypes – 데이터 살펴보기 → astype() 활용 가능 - datetime의 경우

실습

```
df.dtypes
```

```
state     object
year       int64
pop      float64
dtype: object
```

```
df.year = df.year.astype('object')
```

```
df.dtypes
```

```
state     object
year      object
pop      float64
dtype: object
```

4. len – 행의 길이 ⟷ size – 행의 개수 * 열의 개수 = 값의 개수

실습 df 설정

```
df.iloc[5] = [np.nan, np.nan, np.nan]
```

```
df
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	NaN	NaN	NaN

len VS count VS size

```
len(df)
```

```
6
```

```
df.count()
```

```
state    5
year     5
pop      5
dtype: int64
```

```
df.size
```

```
18
```

5.head(), tail() – 데이터 확인

```
df.head(3)
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5

```
df.tail(3)
```

	state	year	pop
3	부산	2014	345.2
4	부산	2016	344.7
5	NaN	NaN	NaN

1.상위, 하위 데이터 일부 확인

2.데이터가 너무 많을 경우 전부 불러오는데, 메모리와 시간이 많이 들 수 있다.

3. Default는 50이다.

6.describe() - 기술통계량

실습

```
df.describe()
```

	pop
count	5.000000
mean	728.440000
std	350.207064
min	344.700000
25%	345.200000
50%	970.500000
75%	984.300000
max	997.500000

1. 오로지 numerical data type의 열에 대해서만 요약

2. 다양한 통계량에 대한 통계치를 구해준다.

3. 결측치(NAN)은 제외되고 요약이 실행된다.

B DataFrame

7.info() – 전체적인 데이터 살펴보기(출력값이 다르셔도 됩니다!)

실습

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   state   6 non-null      object
 1   year    6 non-null      int64
 2   pop     6 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 272.0+ bytes
```

1. 결측치의 개수도 확인 가능

2. 데이터프레임의 전체적인 구조 확인 가능

3. 열별 결측치의 개수, 데이터타입, 메모리사용량 등 전반적으로 확인
→ 데이터프레임의 구조를 낱낱이 보여주는 메소드

BITAMIN

Series

D
DataFrame

S
Series

SD
Summarizing Data

AD
Assign DataFrame

IS
Indexing and Slicing

BI
Boolean Indexing

HM
Handling Missing Data

B Series

↳ A single column of data from a DataFrame

Series vs DataFrame

```
df.state
0    서울
1    서울
2    서울
3    부산
4    부산
5    부산
Name: state, dtype: object
```

```
df['state']
0    서울
1    서울
2    서울
3    부산
4    부산
5    부산
Name: state, dtype: object
```

```
df[['state']]
```

	state
0	서울
1	서울
2	서울
3	부산
4	부산
5	부산

Series	Series	DataFrame

What is the difference?

1. df.pop ?
2. value_counts()
3. unique(), nunique()
4. Methods Chaining

1. df.pop ?

[ ] 사용 O

```
df['pop']

0    997.5
1    984.3
2    970.5
3    345.2
4    344.7
5    340.0
Name: pop, dtype: float64
```

[ ] 사용 X

```
df.pop

<bound method DataFrame.pop of   state  year    pop
0    서울  2014  997.5
1    서울  2016  984.3
2    서울  2018  970.5
3    부산  2014  345.2
4    부산  2016  344.7
5    부산  2018  340.0>
```

메소드 pop 과 column명 pop이 충돌하기 때문에 발생하는 문제
Column을 뽑는 목적이라면 [ ]를 사용하는 방법을 선호하도록 하자.

2. value_counts()

```
df['state'].value_counts()
```

```
서울    3
부산    2
Name: state, dtype: int64
```

1. 주로 Series에서 사용하는 메소드로 각 범주별로 몇번 나타났는지 count해준다.

2. 범주별로 빈도를 count 한 후 자동적으로 sorting을 해준다.

3. reversed=True를 이용해 역순으로 정렬도 가능하다.

3.unique(), nunique()

unique()

```
df['pop'].unique()
```

```
array([997.5, 984.3, 970.5, 345.2, 344.7,   nan])
```

해당 칼럼에서 unique한 값만 보여주는 메소드

nunique()

```
df['pop'].nunique() #NAN은 count하지 않음
```

```
5
```

해당 칼럼에서 unique한 값의 종류의 개수를 알려주는 메소드

3.unique()를 통해 확인하는 DataFrame과 Series의 차이

Series

```
df['pop'].unique()
```

```
array([997.5, 984.3, 970.5, 345.2, 344.7,   nan])
```

DataFrame

```
df[['pop']].unique()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-136-9764ea6a394f> in <module>
----> 1 df[['pop']].unique()

~₩anaconda3₩lib₩site-packages₩pandas₩core₩generic.py in __getattr__(self, name)
   5137             if self._info_axis._can_hold_identifiers_and_holds_name(name):
   5138                 return self[name]
-> 5139             return object.__getattribute__(self, name)
   5140
   5141     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'unique'
```

똑같은 결과가 출력되더라도, DataFrame이나 Series 중 어떤 형태로 뽑느냐에 따라 사용가능한 메소드가 다르다!

4. Methods Chaining

↳ The sequential invocation of methods using the dot notation is referred to as method chaining.

? titanic호에 가장 많이 탄 나이는 몇살일까?

Sol 1

```
titanic.Age.value_counts()
```

```
24.00    30
22.00    27
18.00    26
19.00    25
30.00    25
         ..
55.50     1
70.50     1
66.00     1
23.50     1
0.42      1
Name: Age, Length: 88, dtype: int64
```

```
ta = titanic.Age.value_counts()
ta.index[0]
```

```
24.0
```

Sol 2

```
titanic.Age.value_counts().index[0]
```

```
24.0
```

Method Chaining을 사용한다면 훨씬 간략하게 표현가능!

B Check!

잠시! 앞서 사용했던 코드를 다시 이용해 아래와같은 df를 사용하여 뒤의 실습을 진행합니다!

In [1]:
```
data = {'state': ['서울', '서울', '서울', '부산', '부산', '부산'],
        'year': [2014, 2016, 2018, 2014, 2016, 2018],
        'pop': [997.5, 984.3, 970.5, 345.2, 344.7, 340.0]}
data
```

Out[1]:
```
{'state': ['서울', '서울', '서울', '부산', '부산', '부산'],
 'year': [2014, 2016, 2018, 2014, 2016, 2018],
 'pop': [997.5, 984.3, 970.5, 345.2, 344.7, 340.0]}
```

In [4]:
```
df = DataFrame(data)
df
```

Out[4]:

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

BITAMIN

Summarizing Data

B Summarizing Data

데이터 summarizing을 위한 메소드

count:	Number of non-null observations
sum:	Sum of values
mean:	Mean of values
mad:	Mean absolute deviation
median:	Arithmetic median of values
min:	Minimum
max:	Maximum
mode:	Mode
prod:	Product of values
std:	Bessel-corrected sample standard deviation
var:	Unbiased variance
sem:	Standard error of the mean
skew:	Sample skewness (3rd moment)
kurt:	Sample kurtosis (4th moment)

1.DataFrame → summarizing → Series

```
df.min()
```

```
state      부산
year     2014
pop       340
dtype: object
```

2. Count는 non-null observations를 센다는 점 주의!

3. quantile (0~1)

```
df.quantile(0.5) #median & default
```

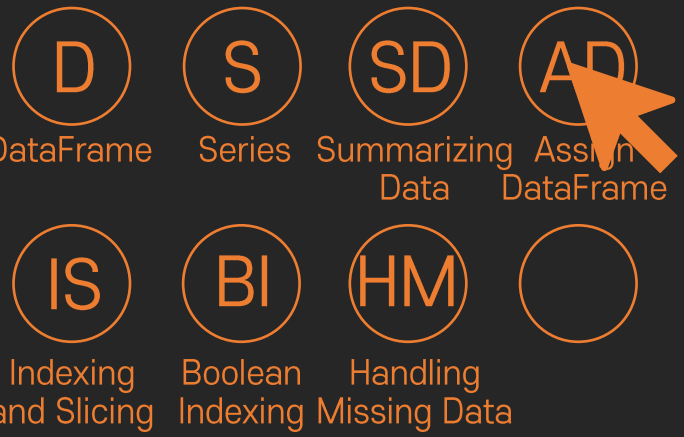
```
year    2016.00
pop      657.85
Name: 0.5, dtype: float64
```

```
df.quantile(0.25)
```

```
year    2014.500
pop      344.825
Name: 0.25, dtype: float64
```

BITAMIN

Assign DataFrame



B Assign DataFrame

데이터프레임 assign

```
dx = df.copy()
display(dx)
dy = dx
dx.iloc[1,1] = pd.NA
display(dx)
display(dy)
display(df)
```

dx

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

dx, dy, df

	state	year	pop
0	서울	2014	997.5
1	서울	<NA>	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

	state	year	pop
0	서울	2014	997.5
1	서울	<NA>	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

1. DataFrame 변수를 assign할 때 (즉, dy = dx), 주소(address)가 복사되고 내용은 복사되지 않음.

2. 원본 데이터를 그대로 보존하면서 DataFrame변수의 내용을 다른 변수에 복사할 경우 copy( ) 메소드 사용

3. pd.NA or np.nan을 사용해 Missing value를 나타낼 수 있다. (not NAN)

B Assign DataFrame

매직커맨드(예시입니다 출력값은 다르게 나올거에요!)

```
%whos
```

```
Variable    Type         Data/Info
-------------------------------
DataFrame   type         <class 'pandas.core.frame.DataFrame'>
Series      type         <class 'pandas.core.series.Series'>
data        dict         n=3
demo        DataFrame                ID  성별  연령  <...>₩n[3542 rows x 4 columns]
df          DataFrame      state  year    pop₩n0  <...>4.7₩n5    부산  2018  340.0
dt          DataFrame        a   b   c₩n0  1   6  1<...>  4   9  14₩n4  5  10  15
movie       DataFrame          color      director<...>n[4916 rows x 28 columns]
pd          module       <module 'pandas' from 'C:<...>es₩₩pandas₩₩__init__.py'>
state_k     Series       0    서울₩n1    서울₩n2    서울<...>ame: state, dtype: object
state_pop   Series       0    997.5₩n1    984.3₩n2<...>Name: pop, dtype: float64
tran        DataFrame                 ID 상품대분류명  <...>[101692 rows x 9 columns]
year_k      Series       0    2014₩n1    2016₩n2  <...>nName: year, dtype: int64
```

* 현재 내가 이 주피터에서 사용할 수 있는 변수의 종류 확인 가능

* 확인하는 이유?

BITAMIN

Indexing and Slicing

B Indexing and Slicing (with loc,iloc)


loc과 iloc 의 구분

df.loc[row_label, col_label]
df.iloc[row_index, col_index]

→ loc은 label이 기준
iloc은 index가 기준

참조 – 둘다 자유자재로 다뤄야함

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows x 12 columns

iloc이 더 유용한 case

여러 개의 행 또는 열을 연속적으로 뽑아야할 때

loc이 더 유용한 case

열의 개수와 종류가 많을 때 필요한 열만 선택하고자 할 때

B Indexing and Slicing (with loc,iloc)

Some Rows & All Columns

```
df.iloc[0]
df.loc[0]
df.iloc[0,:]
```

```
state        서울
year       2014
pop       997.5
Name: 0, dtype: object
```

모든 열을 선택할 경우 ":" 의 유무는 상관없다.

```
df.loc[:1]
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3

```
df.iloc[:1]
```

	state	year	pop
0	서울	2014	997.5

loc 과 iloc의 차이를 다시 상기

All Rows & Some Columns

```
df.loc[:,'state']
df.iloc[:,0]
df.state
df['state']
```

```
0    서울
1    서울
2    서울
3    부산
4    부산
5    부산
Name: state, dtype: object
```

모두 같은 결과

```
df.loc[:,'state':'pop']
#df[['state','pop']]
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

주석과의 차이점?

loc을 활용해 칼럼명도 slicing 가능하다.

Subset of Rows and Columns

```
df.loc[0:2, ['state', 'year']]
```

	state	year
0	서울	2014
1	서울	2016
2	서울	2018

BITAMIN

Boolean Indexing

D
DataFrame

S
Series

SD
Summarizing Data

AD
Assign DataFrame

IS
Indexing and Slicing

BI
Boolean Indexing

HM
Handling Missing Data

B Boolean Indexing

Boolean Indexing의 구조 이해하기

```
df['pop']>500
```

```
0     True
1     True
2     True
3    False
4    False
5    False
Name: pop, dtype: bool
```

bool 자료형으로 추출

비교 연산자(==, >, >=, <=, !=)
in 연산자(in, ==, not in, !=)
논리 연산자(and, or, not)
활용(bool 자료형 추출을 원하므로)

```
df[df['pop']>500]
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5

리스트[] 내에 위에서 추출하는 Boolean 조건을 데이터프레임에서 indexing 하면 True 값을 가지는 행들만 추출된다.

B Boolean Indexing


isin을 통한 boolean indexing

```
df['pop'].isin([997.5,984.3])
```

```
0     True
1     True
2    False
3    False
4    False
5    False
Name: pop, dtype: bool
```

bool 자료형 추출을 연산자가 아닌,
isin이라는 메소드를 사용

따라서 이후의 진행은 boolean indexing과 동일

```
df[df['pop'].isin([997.5,984.3])]
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3

B Boolean Indexing(with query)

query 함수를 통한 조건에 맞는 데이터 추출

↳ query 함수의 사용법 : DataFrame.query('조건문')

1.query 함수 사용시 조건문은 ' ' 또는 " "로 문자열처럼 묶여서 들어가야한다.

** 2. query 조건문에는 column명만 들어간다.

3. query에서 여러 조건문들을 동시에 사용하고 싶을 때 논리연산자를 함께 활용한다.

```
df.query('pop > 500')
```

	state	year	pop
0	서울	2014	997.5
1	서울	2016	984.3
2	서울	2018	970.5

```
df.query('year in [2014, 2016] and state == "부산"')
```

	state	year	pop
3	부산	2014	345.2
4	부산	2016	344.7

B Boolean Indexing(with query)

** 2. query 조건문에는 column명만 들어간다.

Ex1

```
df.query('state ==부산')
-----------------------------------
KeyError
```

Ex2

```
k = df['pop'].value_counts().index[0]
```

```
df.query('pop ==k')
--------------------
KeyError
```

Why error? How to solve?

B Boolean Indexing(with query)

** 2. query 조건문에는 column명만 들어간다.

Ex1 – string 조건

조건문에 사용되는 ‘ ’(“ ”)와 다른 “ ”(‘ ’)를 사용하여 문자열 구성

```
df.query('state =="부산"')
```

	state	year	pop
3	부산	2014	345.2
4	부산	2016	344.7
5	부산	2018	340.0

Ex2 – 변수를 조건문에 활용하고 싶음

조건문에 변수를 사용할 시 변수 앞에 @를 붙이기

```
k = df['pop'].value_counts().index[0]
```

```
df.query('pop==@k')
```

	state	year	pop
4	부산	2016	344.7

BITAMIN

Handling Missing Data

B Handling Missing Data

결측치란?

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

What is problem? How to solve?

B Handling Missing Data

결측치 확인

```
df3 = DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                 'data1': range(7)})
df4 = DataFrame({'rkey': ['a', 'b', 'd'], 'data2': range(3)})
df5 = pd.merge(df3, df4, left_on='lkey', right_on='rkey', how='outer')
df5
display(df5.isnull())
df5.isnull().sum()
```

	lkey	data1	rkey	data2
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	False	False	False	False
6	False	False	True	True
7	True	True	False	False

```
lkey     1
data1    1
rkey     1
data2    1
dtype: int64
```

다양한 방법을 통해 확인 가능
But 열별로 결측치를 확인하는 경우가 많으니
isnull()을 소개

1.isnull() <-> notnull()

2.열별 결측치 개수 세기

B Handling Missing Data

결측치 제거

dropna()를 통해 결측치가 있는 행을 제거할 수 있다.

Case1 : how ='any'

```
df5.dropna(how='any')
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0

현재 데이터프레임에서 하나라도 NAN 값이 있을 때 해당 열을 제거해라.

Case2 : how ='all'

```
df5.dropna(how='all')
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	c	3.0	NaN	NaN
7	NaN	NaN	d	2.0

현재 데이터프레임에서 한열이 모두 NAN 값일 경우 해당 열을 제거해라.

B Handling Missing Data

결측치 제거

결측치가 있는 행을 제거 시 index가 맞지 않는 경우 발생

```
df5.iloc[6] = [np.nan, np.nan, np.nan, np.nan]
df5
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	NaN	NaN	NaN	NaN
7	NaN	NaN	d	2.0

```
df5.dropna(how='all')
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
7	NaN	NaN	d	2.0

```
df5.dropna(how='all').reset_index(drop=True)
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	NaN	NaN	d	2.0

Why do?

B Handling Missing Data

결측치 치환

fillna()를 통해 결측치를 다른 값으로 대체하기.

```
df5.fillna(-1)
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	-1	-1.0	-1	-1.0
7	-1	-1.0	d	2.0

결측치는 주로 0, 최빈값, 평균값, 중앙값 등을 활용해서 치환을 많이 한다.

But 일반적으로 우리는 열별로 같은 값으로 결측치를 채울 일은 많지않다.

B Handling Missing Data

결측치 치환

열별로 다른 값으로 결측치 치환하기

Dictionary를 활용

```
df5.fillna({'data1': 1.5, 'data2': 0.5, 'lkey': 'Y', 'rkey': 'a'})
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	Y	1.5	a	0.5
7	Y	1.5	d	2.0

주변 값으로 결측치 치환하기. (이전값)

주변 값으로 결측치 치환하기. (이후값)

B Handling Missing Data

결측치 치환

원본

```
df5
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	NaN	NaN	NaN	NaN
7	NaN	NaN	d	2.0

주변 값으로 결측치 치환하기. (이전값)

```
df5.fillna(method='ffill')
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	a	5.0	a	0.0
7	a	5.0	d	2.0

주변 값으로 결측치 치환하기. (이후값)

```
df5.fillna(method='bfill')
```

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	NaN	NaN	d	2.0
7	NaN	NaN	d	2.0

Q & A

수고하셨습니다