

데이터분석 동아리

Python 기초

경남대학교
데이터분석동아리
김동우

데이터분석 동아리

INDEX

- 1 파이썬이란 무엇인가
- 2 파이썬의 기초 자료형
- 3 데이터프레임 조작하기
- 4 기초적인 통계보기

1.숫자형

-숫자형이란 숫자 형태로 이루어진 자료형을 말한다

정수형 – 말 그대로 정수를 뜻하는 자료형

```
>>> a = 123  
>>> a = -178  
>>> a = 0
```

실수형 – 파이썬에서 실수형은 소수점이 포함된 숫자를 말한다

```
>>> a = 1.2  
>>> a = -3.45
```

```
>>> a = 4.24E10  
>>> a = 4.24e-10
```

항목	사용 예
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수	0o34, 0o25
16진수	0x2A, 0xFF

2.숫자형을 활용하기 위한 연산자

-사칙연산(+,-,*,/)

-제곱연산자(**)

-나눗셈 후 나머지를 반환하는 연산자(%)

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a * b
12
>>> a / b
0.75
```

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

```
>>> 7 % 3
1
>>> 3 % 7
3
```

1.문자열 자료형

-문자열이란 문자,단어 등으로 구성된 문자들의 집합을 의미한다.

```
"Life is too short, You need Python"  
"a"  
"123"
```

문자열 만들기

-큰따옴표로 양쪽 둘러싸기

```
"Hello World"
```

-큰따옴표 3개 연속쓰기

```
"""Life is too short, You need python"""
```

-작은따옴표로 양쪽 둘러싸기

```
'Python is fun'
```

-작은따옴표 3개 연속쓰기

```
'''Life is too short, You need python'''
```

2.여러 줄인 문자열을 변수에 대입하고 싶을때

줄을 바꾸기 위한 이스케이프 코드 \n 삽입하기

```
>>> multiline = "Life is too short\nYou need python"
```



```
Life is too short  
You need python
```

2.여러 줄인 문자열을 변수에 대입하고 싶을때

줄을 바꾸기 위한 이스케이프 코드 \n 삽입하기

```
>>> multiline = "Life is too short\nYou need python"
```



```
Life is too short
You need python
```

코드	설명
<code>\n</code>	문자열 안에서 줄을 바꿀 때 사용
<code>\t</code>	문자열 사이에 탭 간격을 줄 때 사용
<code>\\</code>	문자 <code>\</code> 를 그대로 표현할 때 사용
<code>\'</code>	작은따옴표(<code>'</code>)를 그대로 표현할 때 사용
<code>\"</code>	큰따옴표(<code>"</code>)를 그대로 표현할 때 사용
<code>\r</code>	캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동)
<code>\f</code>	폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동)
<code>\a</code>	벨 소리(출력할 때 PC 스피커에서 '뽕' 소리가 난다)
<code>\b</code>	백 스페이스
<code>\000</code>	널 문자

문자열 더하기

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

문자열 곱하기 응용

```
# multistring.py

print("=" * 50)
print("My Program")
print("=" * 50)
```



```
=====
My Program
=====
```

문자열 길이 구하기

len() 이라는 함수를 사용하기 문자열의 길이를 구할수 있다.

```
>>> a = "Life is too short"
>>> len(a)
17
```


02-2 문자열 자료형

■ 문자열 인덱싱

■ 인덱싱(Indexing)

- '가리킨다'는 의미
- 파이썬은 0부터 숫자를 셈
- a[번호]
 - 문자열 안의 특정 값 뽑아냄
 - 마이너스(-)
 - 문자열 뒤부터 셈

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
0										1									2										3				
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3

```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

```
a[0]: 'L', a[1]: 'i', a[2]: 'f', a[3]: 'e', a[4]: ' ', ...
```

02-2 문자열 자료형

■ 문자열 슬라이싱

■ 슬라이싱(Slicing)

- '잘라낸다'는 의미
- a[시작 번호:끝 번호]
 - 시작 번호부터 끝 번호까지의 문자를 뽑아냄
 - 끝 번호에 해당하는 것은 포함하지 않음

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
0										1									2									3					
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3

```
>>> a = "Life is too short, You need Python"
>>> a[0:4]
'Life'
```

```
>>> a = "20010331Rainy"
>>> date = a[:8]
>>> weather = a[8:]
>>> date
'20010331'
>>> weather
'Rainy'
```

■ 숫자형 사용법

■ 정수형(Integer)

- 정수를 뜻하는 자료형

```
>>> a = 123
>>> a = -178
>>> a = 0
```

■ 실수형(Floating-point)

- 소수점이 포함된 숫자

```
>>> a = 1.2
>>> a = -3.45
```

```
>>> a = 4.24E10
>>> a = 4.24e-10
```

※ 컴퓨터식 지수 표현 방식

■ 8진수(Octal)

- 숫자 0 + 알파벳 소문자 o 또는 대문자 O

```
>>> a = 0o177
```

■ 16진수(Hexadecimal)

- 숫자 0 + 알파벳 소문자 x

```
>>> a = 0x8ff
>>> b = 0xABC
```

02-2 문자열 자료형

■ 문자열 포매팅

■ 포매팅(Formatting)

1. 숫자 바로 대입

- 문자열 포맷 코드 `%d`

```
>>> "I eat %d apples." % 3
'I eat 3 apples.'
```

```
>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'
```

2. 문자열 바로 대입

- 문자열 포맷 코드 `%s`

```
>>> "I eat %s apples." % "five"
'I eat five apples.'
```

■ 문자열 사용법

■ 문자열 만드는 방법

1. 큰따옴표("")

```
"Hello World"
```

2. 작은따옴표('')

```
'Python is fun'
```

3. 큰따옴표 3개(""")

```
"""Life is too short, You need python"""
```

4. 작은따옴표 3개(''')

```
'''Life is too short, You need python'''
```

02-2 문자열 자료형

■ 문자열 연산하기

1. 문자열 더해서 연결하기(Concatenation)

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

2. 문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

3. 문자열 길이 구하기

■ 파이썬 기본 내장 함수 len()

```
>>> a = "Life is too short"
>>> len(a)
17
```

02-2 문자열 자료형

■ 문자열 포매팅

■ 문자열 포맷 코드

코드	설명
%s	문자열(String)
%c	문자 1개(Character)
%d	정수(Integer)
%f	부동 소수(Floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 '%' 자체)

```
>>> number = 10
>>> day = "three"
>>> "I ate %d apples. so I was sick for %s days." % (number, day)
'I ate 10 apples. so I was sick for three days.'
```

```
>>> "I have %s apples" % 3
'I have 3 apples'
>>> "rate is %s" % 3.234
'rate is 3.234'
```

02-2 문자열 자료형

■ 문자열 관련 함수

- 문자열 자료형이 가진 내장 함수

- **count()**

- 문자 개수 세는 함수

```
>>> a = "hobby"
>>> a.count('b')
2
```

- **find()**

- 찾는 문자열이 처음 나온 위치 반환
 - 없으면 -1 반환

```
>>> a = "Python is the best choice"
>>> a.find('b')
14 ← 문자열에서 b가 처음 나온 위치
>>> a.find('k')
-1
```


02-2 문자열 자료형

■ 문자열 관련 함수

■ index()

- find와 마찬가지로,
찾는 문자열이 처음 나온 위치 반환
- 단, 찾는 문자열이 없으면 오류 발생

```
>>> a = "Life is too short"
>>> a.index('t')
8
>>> a.index('k')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

— k가 없기 때문에 오류 발생

■ join()

- 문자열 삽입

```
>>> ",".join('abcd')
'a,b,c,d'
```

■ upper()

- 소문자를 대문자로 변환

```
>>> a = "hi"
>>> a.upper()
'HI'
```

02-2 문자열 자료형

■ 문자열 관련 함수

■ replace()

- replace(바뀌게 될 문자열, 바꿀 문자열)
- 문자열 안의 특정 값을 다른 값으로 치환

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

■ split()

- 공백 또는 특정 문자열을 구분자로 해서 문자열 분리
분리된 문자열은 리스트로 반환됨

```
>>> a = "Life is too short"
>>> a.split() ← 공백을 기준으로 문자열 나눔
['Life', 'is', 'too', 'short']
>>> b = "a:b:c:d"
>>> b.split(':') ← : 기호를 기준으로 문자열 나눔
['a', 'b', 'c', 'd']
```

3.리스트 자료형

-숫자형이나 문자열의 모음(집합)을 표현한 방식

A = [] <- 리스트는 대괄호[] 로 묶어서 표현한다

리스트명 = [요소1, 요소2, 요소3, ...]

```
>>> a = []  
>>> b = [1, 2, 3]  
>>> c = ['Life', 'is', 'too', 'short']  
>>> d = [1, 2, 'Life', 'is']  
>>> e = [1, 2, ['Life', 'is']]
```

*비어있는 리스트는 A = list() 또는 A = []로 생성이 가능하다

2.리스트 인덱싱

-리스트의 인덱싱은 문자열 처럼 적용이 가능하다

```
In [6]: a = [1, 2, 3, 4, 5]
        print(a[0])
        print(a[1])
        print(a[2])
        print(a[4])
```

```
1
2
3
5
```

-뒤에서 부터 보기

```
In [8]: a[-1]
```

```
Out [8]: 5
```

-리스트의 인덱스 번호를 입력을 해서 연산이 가능하다

```
In [7]: a[0] + a[2]
```

```
Out [7]: 4
```

-리스트 안에 리스트 넣고 뽑아 보기

```
In [16]: b = [1, 2, ['a', 'b', [3, 4, 5]]]

        print(b[2])
        print(b[2][1])
        print(b[2][2][0])
```

```
['a', 'b', [3, 4, 5]]
b
3
```

3.리스트 슬라이싱

-문자열이랑 똑같다

```
In [17]: a = [1,2,3,4,5]
```

```
In [18]: a[1:3]
```

```
Out[18]: [2, 3]
```

-리스트 길이 구하기

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

리스트 더하기(+)

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

리스트 반복하기(*)

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

3.리스트 슬라이싱

-문자열이랑 똑같다

```
In [17]: a = [1,2,3,4,5]
```

```
In [18]: a[1:3]
```

```
Out[18]: [2, 3]
```

-리스트 길이 구하기

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

리스트 더하기(+)

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

리스트 반복하기(*)

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

4.리스트 수정과 삭제

리스트에서 값 수정하기 del 함수 사용해 리스트 요소 삭제하기

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

5.리스트 관련 함수들

추가하기 -> append()

정렬하기 -> sort()

뒤집기 -> reverse()

위치 반환 -> index()

요소 삽입 -> insert()

요소 제거 -> remove()

라스트 요소 끄집어내서 삭제하기 -> pop()

개수 세기 -> count()

리스트 확장 -> extend()

```
a = [1,2,3,1]

a.append(4)# 리스트에 추가 하기
print(a)
a.sort()# 리스트를 오름차순으로 정렬시키기
print(a)
a.sort(reverse=True)# 리스트를 내림차순으로 정렬
print(a)
a.reverse()#리스트를 뒤집기
print(a)
a.index(1)# 리스트의 인덱스 번호에 해당하는 값 받기
print(a)
a.insert(0,4)#리스트에 요소 삽입
print(a)
a.remove(4)#리스트에 요소 삭제
print(a)
a.pop()#라스트 요소 끄집어내서 삭제
print(a)
a.count(1)#개수 세기
print(a)
```

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

4.튜플 자료형

- 튜플 자료형은 몇가지만 제외하면 리스트와 거의 같다
- 리스트는 []를 쓴다면 튜플은 ()을 쓴다

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

튜플의 가장 큰 특징은 요솟값을 한 번 지정하고 나면 지우거나 변경이 불가능 하다

1. 튜플 요솟값을 삭제하려 할 때

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

2. 튜플 요솟값을 변경하려 할 때

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```


5.딕셔너리 자료형

- 딕셔너리는 대응관계를 나타낼수 있는 자료형이다
- 딕셔너리는 {}로 감싼다

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

```
{Key1:Value1, Key2:Value2, Key3:Value3, ...}
```

```
In [56]: a = {1 : '김동우', '직업' : '데이터', 3 : 971125}
```

```
print(a)
print(a[1])
print(a['직업'])
print(a[3])
```

```
{1: '김동우', '직업': '데이터', 3: 971125}
김동우
데이터
971125
```

2.딕셔너리 조작하기

딕셔너리 쌍 추가하기

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
```

```
>>> a['name'] = 'pey'
>>> a
{1: 'a', 2: 'b', 'name': 'pey'}
```

딕셔너리 요소 삭제하기

```
>>> del a[1]
>>> a
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

3.딕셔너리 관련 함수들

Key 리스트 만들기(keys)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}  
>>> a.keys()  
dict_keys(['name', 'phone', 'birth'])
```

```
>>> list(a.keys())  
['name', 'phone', 'birth']
```

3.딕셔너리 관련 함수들

Value 리스트 만들기(values)

```
>>> a.values()  
dict_values(['pey', '0119993323', '1118'])
```

Key, Value 쌍 얻기(items)

```
>>> a.items()  
dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])
```

3.딕셔너리 관련 함수들

Key로 Value얻기(get)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

그냥 뽑는거랑 get()함수를 이용한 뽑기랑 가장 큰차이점은 그냥 뽑으면 ERROR를 띄어 실행이 안되지만

Get()함수를 쓰면 없는 값을 입력을 해도 None(거짓) 이라고 출력이 되면서 실행은 된다

해당 Key가 딕셔너리 안에 있는지 조사하기(in)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

6. 불(bool) 자료형

-불 자료형은 참(True)과 거짓(False)만을 나타내는 자료형이다

```
>>> a = True
>>> b = False
```

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

```
>>> bool([1,2,3])
True
>>> bool([])
False
>>> bool(0)
False
>>> bool(3)
True
```

데이터프레임 조작하기

```
In [1]: import pandas as pd
```

```
In [5]: df = pd.read_csv('C:/Users/Administrator/Desktop/공부 ppt/gapminder.csv')
```

-데이터 프레임 불러오기

```
In [10]: type(df)
```

클래스 확인

```
Out[10]: pandas.core.frame.DataFrame
```

```
In [12]: df.shape
```

행과 열 알아보기

```
Out[12]: (1704, 6)
```

```
In [13]: df.columns
```

변수가 무엇으로 이루어져 있는지 알아보기

```
Out[13]: Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

데이터프레임 조작하기

In [7]:

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   country     1704 non-null   object
1   continent   1704 non-null   object
2   year        1704 non-null   int64
3   lifeExp     1704 non-null   float64
4   pop         1704 non-null   int64
5   gdpPercap   1704 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
```

memory usage: 80.0+ KB

In [9]:

df.dtypes

Out[9]:

```
country      object
continent    object
year          int64
lifeExp      float64
pop           int64
gdpPercap    float64
dtype: object
```

-데이터 프레임 정보보기

데이터프레임 조작하기

```
In [16]: df['country']
Out[16]:
0      Afghanistan
1      Afghanistan
2      Afghanistan
3      Afghanistan
4      Afghanistan
...
1699    Zimbabwe
1700    Zimbabwe
1701    Zimbabwe
1702    Zimbabwe
1703    Zimbabwe
Name: country, Length: 1704, dtype: object
```

```
In [18]: df['country'].head()
Out[18]:
0      Afghanistan
1      Afghanistan
2      Afghanistan
3      Afghanistan
4      Afghanistan
Name: country, dtype: object
```

```
In [19]: df['country'].tail()
Out[19]:
1699    Zimbabwe
1700    Zimbabwe
1701    Zimbabwe
1702    Zimbabwe
1703    Zimbabwe
Name: country, dtype: object
```

-데이터 프레임 변수지정해서 보기

-데이터 프레임 위에서 5개만 보기

-데이터 프레임 뒤에서 5개만 보기

```
In [20]: df['country'].head(10)
Out[20]:
0      Afghanistan
1      Afghanistan
2      Afghanistan
3      Afghanistan
4      Afghanistan
5      Afghanistan
6      Afghanistan
7      Afghanistan
8      Afghanistan
9      Afghanistan
Name: country, dtype: object
```

```
In [22]: df[['country', 'year']]
Out[22]:
```

	country	year
0	Afghanistan	1952
1	Afghanistan	1957
2	Afghanistan	1962
3	Afghanistan	1967
4	Afghanistan	1972
...
1699	Zimbabwe	1987
1700	Zimbabwe	1992
1701	Zimbabwe	1997
1702	Zimbabwe	2002
1703	Zimbabwe	2007

1704 rows × 2 columns

```
In [21]: df[['country', 'year']]
KeyError: ('country', 'year')
Traceback (most recent call last):
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, toler
    3079         try:
-> 3080             return self._engine.get_loc(casted_key)
    3081         except KeyError as err:
pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()
KeyError: ('country', 'year')
The above exception was the direct cause of the following exception:
KeyError
<ipython-input-21-2946e395f637> in <module>
    1 df[['country', 'year']]
```

-데이터 프레임 위에서 10개 보기

-데이터 프레임 변수 두개 선택

데이터프레임 조작하기

속성	설명
loc	인덱스를 기준으로 행 데이터 추출
iloc	행 번호를 기준으로 행데이터 추출

```
In [23]: df.loc[1]
```

```
Out[23]: country      Afghanistan
continent      Asia
year          1957
lifeExp       30.332
pop           9240934
gdpPercap     820.85303
Name: 1, dtype: object
```

```
In [24]: df.iloc[1]
```

```
Out[24]: country      Afghanistan
continent      Asia
year          1957
lifeExp       30.332
pop           9240934
gdpPercap     820.85303
Name: 1, dtype: object
```

데이터프레임 조작하기

속성	설명
loc	인덱스를 기준으로 행 데이터 추출
iloc	행 번호를 기준으로 행데이터 추출

```
In [27]: df.iloc[:,1]
```

```
Out[27]: 0      Asia
1      Asia
2      Asia
3      Asia
4      Asia
...
1699   Africa
1700   Africa
1701   Africa
1702   Africa
1703   Africa
Name: continent, Length: 1704, dtype: object
```

```
In [28]: df.loc[:,1]
```

```
-----
KeyError                                Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:3079: in try:
-> 3080         return self._engine.get_loc(casted_key)
      3081     except KeyError as err:

pandas\libs\index.pyx in pandas._libs.index.IndexEngine.get_loc

pandas\libs\index.pyx in pandas._libs.index.IndexEngine.get_loc

pandas\libs\hashtable_class_helper.pxi in pandas._libs.hash

pandas\libs\hashtable_class_helper.pxi in pandas._libs.hash

KeyError: 1

The above exception was the direct cause of the following except

KeyError                                Traceback (most recent call last)
<ipython-input-28-8c88efd7517d> in <module>
```

```
In [31]: df.loc[:, 'country']
```

```
Out[31]: 0      Afghanistan
1      Afghanistan
2      Afghanistan
3      Afghanistan
4      Afghanistan
...
1699   Zimbabwe
1700   Zimbabwe
1701   Zimbabwe
1702   Zimbabwe
1703   Zimbabwe
Name: country, Length: 1704, dtype: object
```

기초적인 통계 계산

```
In [32]: df.mean()
```

```
Out[32]: year          1.979500e+03  
lifeExp        5.947444e+01  
pop           2.960121e+07  
gdpPercap     7.215327e+03  
dtype: float64
```

```
In [34]: df.min()
```

```
Out[34]: country      Afghanistan  
continent      Africa  
year           1952  
lifeExp        23.599  
pop            60011  
gdpPercap     241.165876  
dtype: object
```

```
In [33]: df.max()
```

```
Out[33]: country      Zimbabwe  
continent      Oceania  
year           2007  
lifeExp        82.603  
pop           1318683096  
gdpPercap     113523.1329  
dtype: object
```

기초적인 통계 계산

```
In [37]: df.groupby('year').mean()
```

```
Out[37]:
```

	lifeExp	pop	gdpPercap
year			
1952	49.057620	1.695040e+07	3725.276046
1957	51.507401	1.876341e+07	4299.408345
1962	53.609249	2.042101e+07	4725.812342
1967	55.678290	2.265830e+07	5483.653047
1972	57.647386	2.518998e+07	6770.082815
1977	59.570157	2.767638e+07	7313.166421
1982	61.533197	3.020730e+07	7518.901673
1987	63.212613	3.303857e+07	7900.920218
1992	64.160338	3.599092e+07	8158.608521
1997	65.014676	3.883947e+07	9090.175363
2002	65.694923	4.145759e+07	9917.848365
2007	67.007423	4.402122e+07	11680.071820

```
In [38]: df.groupby('year')['pop'].mean()
```

```
Out[38]: year
1952    1.695040e+07
1957    1.876341e+07
1962    2.042101e+07
1967    2.265830e+07
1972    2.518998e+07
1977    2.767638e+07
1982    3.020730e+07
1987    3.303857e+07
1992    3.599092e+07
1997    3.883947e+07
2002    4.145759e+07
2007    4.402122e+07
Name: pop, dtype: float64
```

기초적인 통계 계산

```
name = 'pop', dtype = 'float64'
```

```
In [40]: df.groupby(['year', 'continent'])[['pop', 'gdpPercap']].mean()
```

```
Out[40]:
```

		pop	gdpPercap
year	continent		
1952	Africa	4.570010e+06	1252.572466
	Americas	1.380610e+07	4079.062552
	Asia	4.228356e+07	5195.484004
	Europe	1.393736e+07	5661.057435
	Oceania	5.343003e+06	10298.085650
1957	Africa	5.093033e+06	1385.236062
	Americas	1.547816e+07	4616.043733
	Asia	4.735699e+07	5787.732940
	Europe	1.459635e+07	6963.012816
	Oceania	5.970988e+06	11598.522455

기초적인 통계 계산

```
In [42]: pd.pivot_table(df, index = 'year', values = 'pop', aggfunc = ['mean', 'sum', 'std'])
```

```
Out[42]:
```

	mean	sum	std
	pop	pop	pop
year			
1952	1.695040e+07	2406957150	5.810086e+07
1957	1.876341e+07	2664404580	6.550429e+07
1962	2.042101e+07	2899782974	6.978865e+07
1967	2.265830e+07	3217478384	7.837548e+07
1972	2.518998e+07	3576977158	8.864682e+07
1977	2.767638e+07	3930045807	9.748109e+07
1982	3.020730e+07	4289436840	1.050986e+08
1987	3.303857e+07	4691477418	1.147562e+08
1992	3.599092e+07	5110710260	1.245026e+08

```
In [41]: pd.pivot_table(df, index = ['year', 'continent'], values = ['pop', 'gdpPercap'], aggfunc = ['mean', 'sum', 'std'])
```

```
Out[41]:
```

		mean		sum		std	
		gdpPercap	pop	gdpPercap	pop	gdpPercap	pop
year	continent						
1952	Africa	1252.572466	4.570010e+06	65133.768223	237640501	982.952116	6.317450e+06
	Americas	4079.062552	1.380610e+07	101976.563805	345152446	3001.727522	3.234163e+07
	Asia	5195.484004	4.228356e+07	171450.972133	1395357351	18634.890865	1.132267e+08
	Europe	5661.057435	1.393736e+07	169831.723043	418120846	3114.060493	1.724745e+07
	Oceania	10298.085650	5.343003e+06	20596.171300	10686006	365.560078	4.735083e+06
1957	Africa	1385.236062	5.093033e+06	72032.275237	264837738	1134.508918	7.076042e+06
	Americas	4616.043733	1.547816e+07	115401.093329	386953916	3312.381083	3.553706e+07
	Asia	5787.732940	4.735699e+07	190995.187018	1562780599	19506.515959	1.280961e+08
	Europe	6963.012816	1.459635e+07	208890.384478	437890351	3677.950146	1.783235e+07

```
In [ ]:
```

