

Python 금융분석 기초



1. 파이썬 개요
2. Numpy data structure
3. Pandas Data Analysis

Lecturer : Lee, Seunghee
storm@netsgo.com
github.com/seunghee-lee/python

1. 파이썬 개요

1. 프로그래밍 언어의 개념과 종류

사람이 이해하는 말 : 언어

컴퓨터가 이해하는 말 : 프로그래밍 언어

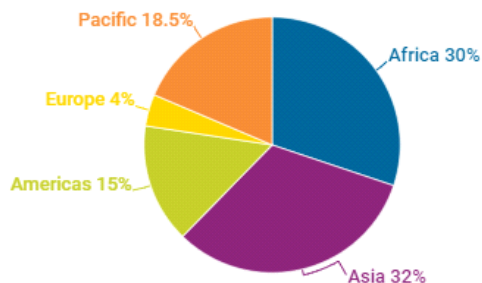
전 세계에서 인간이 사용하는 언어 : 7,111개(2019년 기준)

실제로 많이 사용하는 언어는 몇십 개에 불과

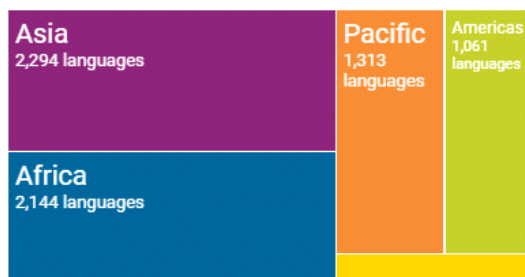
각 언어는 각기 다른 문법과 표현력, 사용용도 보유

기본적인 언어의 역할은 소통

Percentage of the world's languages, by region



Languages by region of origin



Population by region of origin



(출처: www.ethnologue.com)














컴퓨터 프로그래밍 언어 : 700개 이상

프로그래밍 언어도 각기 다른 문법과 표현력, 사용용도 보유

컴퓨팅 환경과 활용 범위에 따라 언어의 인기가 달라짐

새로운 언어의 등장과 기존 언어의 소멸도 다반사

(출처: www.tiobe.com/tiobe-index)

May 2024	May 2023	Change	Programming Language	Ratings	Change
1	1		 Python	16.33%	+2.88%
2	2		 C	9.98%	-3.37%
3	4	▲	 C++	9.53%	-2.43%
4	3	▼	 Java	8.69%	-3.53%
5	5		 C#	6.49%	-0.94%
6	7	▲	 JavaScript	3.01%	+0.57%
7	6	▼	 Visual Basic	2.01%	-1.83%
8	12	▲	 Go	1.60%	+0.61%
9	9		 SQL	1.44%	-0.03%
10	19	▲	 Fortran	1.24%	+0.46%
11	11		 Delphi/Object Pascal	1.24%	+0.23%
12	10	▼	 Assembly language	1.07%	-0.13%
13	18	▲	 Ruby	1.06%	+0.26%
14	15	▲	 MATLAB	1.06%	+0.18%
15	14	▼	 Swift	1.01%	+0.09%
16	8	▼	 PHP	0.97%	-0.62%
17	13	▼	 Scratch	0.93%	-0.02%

2. 파이썬 소개



(출처: www.python.org)

프로그래밍 언어인 파이썬(python)은 배우기 쉽고 결과도 바로 확인할 수 있다.

파이썬은 1991년 프로그래머인 귀도 반 로섬(Guido van Rossum)이 발표한 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamically typed) 대화형 언어이다. 파이썬이라는 이름은 귀도가 좋아하는 코미디 <Monty Python's Flying Circus>에서 따온 것이다.

파이썬은 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반 개발 모델을 가지고 있다. C언어로 구현된 C파이썬 구현이 사실상의 표준이다.



귀도 반 로섬(Guido van Rossum)
(출처: 위키피디아)

<파이썬 특징>

1. 강력한 기능을 무료로 사용
2. 읽기 쉽고 사용하기 쉽다.
3. 사물인터넷과 잘 연동된다.
4. 다양하고 강력한 외부 라이브러리들이 풍부하다.
라이브러리(모듈)가 풍부하여, 대학을 비롯한 여러 교육 기관, 연구 기관 및 산업계에서 이용이 증가
5. 강력한 웹 프레임워크를 사용할 수 있다. - 장고(Django), 플래스크(Flask) 등 - 대표적인 웹사이트로 인스타그램

파이썬의 단점

1. 느린 속도
2. 메뉴나 메시지가 영문

<파이썬의 핵심 철학>

"아름다운게 추한 것보다 낫다." (Beautiful is better than ugly)

"명시적인 것이 암시적인 것 보다 낫다." (Explicit is better than implicit)

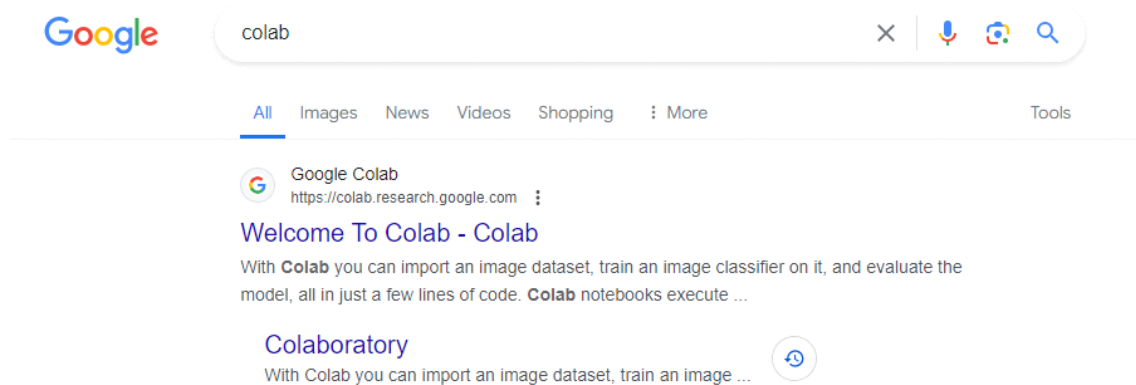
"단순함이 복잡함보다 낫다." (Simple is better than complex)

"복잡함이 난해한 것보다 낫다." (Complex is better than complicated)

"가독성은 중요하다." (Readability counts)

3. Google Colab

Chrome 실행 후 Colab 검색



1. Jupyter Notebook

1) 주피터 프로젝트

◆ 개요



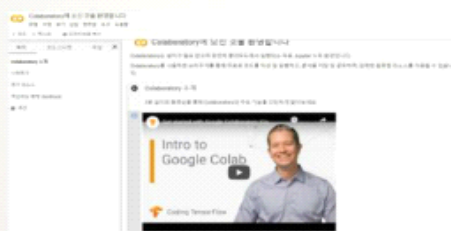
- 웹브라우저 상에서 파이썬 코드를 작성하고 실행해 볼 수 있는 개발 도구
- 모든 프로그래밍 언어에서 대화형 데이터 과학 및 과학 컴퓨팅을 지원

2. Google Colab

1) 개요

◆ 코랩(Colaboratory)의 기능

- Jupyter Notebook과 Google Drive를 합성한 형태
- 클라우드 기반 Jupyter Notebook UI 및 기능 제공
- 구글 계정 전용의 가상 머신 지원
- 머신 러닝을 위한 GPU 및 TPU 무료 제공



2. Google Colab

1) 개요

◆ 코랩(Colaboratory)의 장점

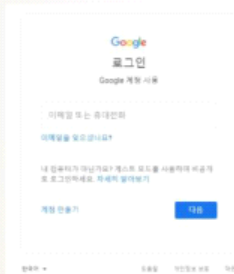
- 별도의 파이썬 설치없이 웹 브라우저 상에서 주피터 노트북과 같은 작업 수행이 가능함
- 다른 사용자와 공유가 쉬워서 연구·교육용으로 많이 사용함
- numpy, pandas, matplotlib, scikit-learn, tensorflow 등 데이터 분석 및 인공지능에 많이 사용되는 패키지들이 미리 설치되어 있음
- 12시간 동안 무료 GPU 사용이 가능함
- 구글 독스나 구글 스프레드시트 등과 같은 방식으로 공유와 편집이 가능함

2. Google Colab

2) 접속

◆ 코랩(Colaboratory)의 사용법

- 구글 계정 로그인



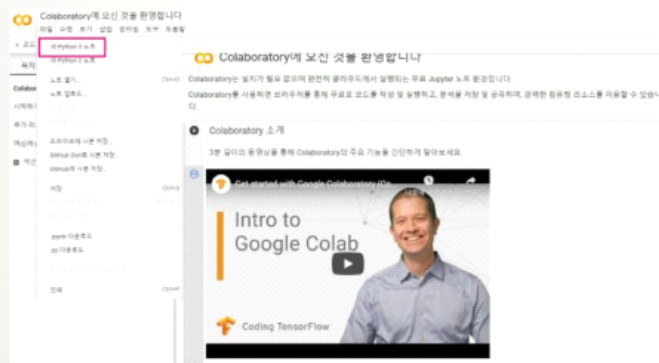
Jupyter Notebook과 사용 방법이 비슷함

2. Google Colab

2) 접속

◆ .ipynb 파일

- 새 Python3 노트 실행




```
from google.colab import drive
drive.mount('/content/gdrive/')
```

노트북에서 **Google Drive** 파일에 액세스하도록 허용하시겠습니까?

이 노트북에서 Google Drive 파일에 대한 액세스를 요청합니다. Google Drive에 대한 액세스 권한을 부여하면 노트북에서 실행되는 코드가 Google Drive의 파일을 수정할 수 있게 됩니다. 이 액세스를 허용하기 전에 노트북 코드를 검토하시기 바랍니다.

아니요 [Google Drive에 연결](#)

 Google 계정으로 로그인



계정 선택

[Google Drive for desktop\(으\)로 이동](#)



Seunghee Lee
storm386@gmail.com

② 다른 계정 사용


계속 진행하기 위해 Google에서 내 이름, 이메일 주소, 언어 환경설정, 프로필 사진을 Google Drive for desktop과(와) 공유합니다. 앱을 사용하기 전에 Google Drive for desktop의 [개인정보처리방침](#) 및 [서비스 약관](#)을 검토하세요.

한국어

도움말

개인정보처리방침

약관

 Google 계정으로 로그인



Google Drive for desktop 서비스로 로그인



storm386@gmail.com

계속하면 Google에서 내 이름, 이메일 주소, 언어 환경설정, 프로필 사진을 Google Drive for desktop 서비스와 공유합니다. Google Drive for desktop의 [개인정보처리방침](#) 및 [서비스 약관](#)을 참고하세요.

Google 계정에서 Google 계정으로 로그인을 관리할 수 있습니다.

취소

계속

한국어

도움말

개인정보처리방침

약관



Untitled8.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말



+ 코드 + 텍스트



[1] `from google.colab import drive`
`drive.mount('/content/gdrive/')`



Mounted at /content/gdrive/



코딩을 시작하거나 AI로 코드를 생성하세요.



2. Numerical Computing with NumPy

Computers are useless. They can only give answers.

—Pablo Picasso

2-1 Arrays with Python Lists

array : 집합체, 배열하다는 뜻

list 자료구조는 숫자들의 배열을 잘 반영

1	<code>v = [0.5, 0.75, 1.0, 1.5, 2.0]</code>
---	---

중첩된 리스트 구조를 이용하면 2차원이나 다차원 배열도 쉽게 생성

2	<code>m = [v, v, v]</code> <code>m</code>
---	--

인덱싱으로 특정한 행을 선택하거나 이중 인덱싱으로 특정한 원소를 선택 가능
그러나 열을 선택하는 것은 쉽지 않음

3	<code>m[1]</code>
---	-------------------

4	<code>m[1][0]</code>
---	----------------------

중첩을 이용하여 더 복잡한 구조 생성 가능

5	<code>v1 = [0.5, 1.5]</code> <code>v2 = [1, 2]</code> <code>m = [v1, v2]</code> <code>c = [m, m]</code> <code>c</code>
---	--

`[[[0.5, 1.5], [1, 2]], [[0.5, 1.5], [1, 2]]]`

6	<code>c[1][1][0]</code>
---	-------------------------

1

위의 객체 조합 방식은 참조 포인터를 사용한 방식

7	<code>v = [0.5, 0.75, 1.0, 1.5, 2.0]</code> <code>m = [v, v, v]</code> <code>m</code>
---	---

`[[0.5, 0.75, 1.0, 1.5, 2.0],`
`[0.5, 0.75, 1.0, 1.5, 2.0],`
`[0.5, 0.75, 1.0, 1.5, 2.0]]`

8	<code>v[0] = 'Python'</code> <code>m</code>
---	--

`[['Python', 0.75, 1.0, 1.5, 2.0],`
`['Python', 0.75, 1.0, 1.5, 2.0],`


```
['Python', 0.75, 1.0, 1.5, 2.0]]
```

v[0] 값을 바꾸었더니 m의 값도 따라서 바뀌는 문제 도출

이러한 문제를 해결하기 위해 deepcopy 함수를 사용

9	<pre>from copy import deepcopy v = [0.5, 0.75, 1.0, 1.5, 2.0] m = 3 * [deepcopy(v),] m</pre>
---	---

```
[[0.5, 0.75, 1.0, 1.5, 2.0],
 [0.5, 0.75, 1.0, 1.5, 2.0],
 [0.5, 0.75, 1.0, 1.5, 2.0]]
```

10	<pre>v[0] = 'Python' m</pre>
----	------------------------------

```
[[0.5, 0.75, 1.0, 1.5, 2.0],
 [0.5, 0.75, 1.0, 1.5, 2.0],
 [0.5, 0.75, 1.0, 1.5, 2.0]]
```

2-2 정규 NumPy 배열

11	<pre>import numpy as np</pre>
----	-------------------------------

12	<pre>a = np.array([0, 0.5, 1.0, 1.5, 2.0]) a</pre>
----	--

13	<pre>type(a)</pre>
----	--------------------

numpy.ndarray

list 객체를 이용해 배열구조를 만들 수 있으나 여러 가지 단점 존재

배열 타입을 잘 다루는 목적으로 만들어 진 것이 numpy.ndarray

n차원 배열을 효율적이고 고성능으로 다루기 위한 목적으로 개발됨

14	<pre>a = np.array(['a', 'b', 'c']) a</pre>
----	--

array(['a', 'b', 'c'], dtype='<U1')

15	<pre>a = np.arange(2, 20, 2) a</pre>
----	--------------------------------------

array([2, 4, 6, 8, 10, 12, 14, 16, 18])

arange함수는 (시작점, 끝점+1, 증가단위)로 지정할 수 있다.

16	<pre>a = np.arange(8, dtype=np.float64) a</pre>
----	---

array([0., 1., 2., 3., 4., 5., 6., 7.])

arange함수에서 파라미터 숫자를 한 개만 지정하면 0부터 1씩 증가해서 지정된 숫자 전까지 할당한다.

17	a[5:]
----	-------

```
array([5., 6., 7.])
```

18	a[:2]
----	-------

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

인덱싱은 [처음 값:끝 값+1]로 지정하며 값이 없으면 끝까지 또는 처음부터를 의미한다.

numpy.ndarray 클래스의 가장 큰 특징은 다양한 내장 메서드가 있다는 점이다.

19	a.sum()
----	---------

```
28.0
```

20	a.std()
----	---------

```
2.29128784747792
```

21	a.cumsum()
----	------------

```
array([ 0.,  1.,  3.,  6., 10., 15., 21., 28.])
```

22	1 = [0., 0.5, 1.5, 3., 5.] 2 * 1
----	-------------------------------------

```
[0.0, 0.5, 1.5, 3.0, 5.0, 0.0, 0.5, 1.5, 3.0, 5.0]
```

list자료에서 곱셈은 반복을 의미

ndarray 객체에 대해서는 벡터화된 형식의 수학 연산이 가능

23	a
----	---

```
array([0., 1., 2., 3., 4., 5., 6., 7.])
```

24	2 * a
----	-------

```
array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14.])
```

25	a ** 2
----	--------

```
array([ 0.,  1.,  4.,  9., 16., 25., 36., 49.])
```

26	2 ** a
----	--------

```
array([ 1.,  2.,  4.,  8., 16., 32., 64., 128.])
```

27	a ** a
----	--------

```
array([1.00000e+00, 1.00000e+00, 4.00000e+00, 2.70000e+01, 2.56000e+02,  
      3.12500e+03, 4.66560e+04, 8.23543e+05])
```

28	np.exp(a)
----	-----------

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,  
      5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03])
```

29	np.sqrt(a)
----	------------

```
array([0.          , 1.          , 1.41421356, 1.73205081, 2.          ,  
      2.23606798, 2.44948974, 2.64575131])
```

30	np.sqrt(2.5)
----	--------------

1.5811388300841898

31	import math math.sqrt(2.5)
----	-------------------------------

1.5811388300841898

32	%timeit np.sqrt(2.5)
----	----------------------

969 ns ± 316 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

33	%timeit math.sqrt(2.5)
----	------------------------

88.3 ns ± 0.886 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
NumPy보다 math 모듈의 계산속도가 빠름

다차원(Multiple Dimensions)

34	b = np.array([a, a * 2]) b
----	-------------------------------

array([[0., 1., 2., 3., 4., 5., 6., 7.],
[0., 2., 4., 6., 8., 10., 12., 14.]])

35	b[0] # 첫 번째 행
----	---------------

36	b[0, 2] # 첫 번째 행의 2번 원소
----	-------------------------

37	b[:, 1] # 1번 열
----	----------------

array([1., 2.])

38	b.sum() # 총 합계
----	----------------

84.0

39	b.sum(axis=0) # 0번 축을 따라 합계, 열(column) 합계
----	---

array([0., 3., 6., 9., 12., 15., 18., 21.])

40	b.sum(axis=1) # 1번 축을 따라 합계, 행(row) 합계
----	--

array([28., 56.])

처음에는 특정한 원소의 값이 없는 numpy.ndarray 객체를 만들고 나중에 코드를 실행하면서 각 원소의 값을 지정하는 방법을 원할 때 zeros나 ones함수를 이용한다.

41	c = np.zeros((2, 3), dtype='i', order='C') c
----	---

array([[0, 0, 0],
[0, 0, 0]], dtype=int32)

- shape: Either an int, a sequence of int objects, or a reference to another ndarray

- dtype (optional): A dtype—these are NumPy-specific data types for ndarray objects

- order (optional): The order in which to store elements in memory: C for C-like

(i.e., row-wise) or F for Fortran-like (i.e., column-wise)

42	<code>c = np.ones((2, 3, 4), dtype='i', order='C')</code> <code>c</code>
----	---

```
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int32)
```

43	<code>d = np.zeros_like(c, dtype='f16', order='C')</code> <code>d</code>
----	---

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

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]], dtype=float128)
```

속도 비교

44	<code>import random</code> <code>I = 5000</code>
----	---

45	<code>%time mat = [[random.gauss(0, 1) for j in range(I)] \</code> <code> for i in range(I)]</code>
----	---

CPU times: user 16.4 s, sys: 686 ms, total: 17.1 s

Wall time: 17.2 s

46	<code>%time sum([sum(l) for l in mat])</code>
----	---

CPU times: user 178 ms, sys: 374 µs, total: 178 ms

Wall time: 180 ms

-394.78119784368084

47	<code>%time mat = np.random.standard_normal((I, I))</code>
----	--

CPU times: user 730 ms, sys: 65.5 ms, total: 796 ms

Wall time: 796 ms

48	<code>%time mat.sum()</code>
----	------------------------------

CPU times: user 23.4 ms, sys: 0 ns, total: 23.4 ms

Wall time: 24.1 ms

-10072.918457592843

- NumPy 배열 연산과 알고리즘을 사용하면 순수 파이썬 코드보다 간결하고 읽기 쉬운 코드

를 만들 수 있고 성능도 획기적으로 개선할 수 있다.

구조화 배열

- NumPy는 여러 가지 각 열마다 다른 자료형을 사용할 수 있는 구조화 배열(structured array)을 지원

49	<code>dt = np.dtype([('Name', 'S10'), ('Age', 'i4'), ('Height', 'f'), ('Children/Pets', 'i4', 2)])</code>
----	---

50	<code>s = np.array([('Smith', 45, 1.83, (0, 1)), ('Jones', 53, 1.72, (2, 2))], dtype=dt)</code> <code>s</code>
----	---

```
array([(b'Smith', 45, 1.83, [0, 1]), (b'Jones', 53, 1.72, [2, 2])],  
      dtype=[('Name', 'S10'), ('Age', '<i4'), ('Height', '<f4'), ('Children/Pets', '<i4',  
(2,))])
```

51	<code>s['Name']</code> <code>array([b'Smith', b'Jones'], dtype=' S10')</code>
----	--

52	<code>s['Height'].mean()</code> 1.7750001
----	--

53	<code>s[1]['Age']</code> 53
----	--------------------------------

3-3 코드 벡터화

코드 벡터화는 코드를 더 간결하게 하고 실행 속도를 높이기 위한 전략

54	<code>np.random.seed(100)</code> <code>r = np.arange(12).reshape((4, 3))</code> <code>s = np.arange(12).reshape((4, 3)) * 0.5</code>
----	--

55	<code>r</code> <code>array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])</code>
----	---

56	<code>s</code> <code>array([[0. , 0.5, 1.], [1.5, 2. , 2.5], [3. , 3.5, 4.], [4.5, 5. , 5.5]])</code>
----	---

57	$r + s$
----	---------

```
array([[ 0. ,  1.5,  3. ],
       [ 4.5,  6. ,  7.5],
       [ 9. , 10.5, 12. ],
       [13.5, 15. , 16.5]])
```

58	$r + 3$
----	---------

```
array([[ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

59	$2 * r$
----	---------

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

60	$2 * r + 3$
----	-------------

```
array([[ 3,  5,  7],
       [ 9, 11, 13],
       [15, 17, 19],
       [21, 23, 25]])
```

61	r
----	-----

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

62	$r.shape$
----	-----------

```
(4, 3)
```

63	$s = np.arange(0, 12, 4)$ s
----	----------------------------------

```
array([0, 4, 8])
```

64	$r + s$
----	---------

```
array([[ 0,  5, 10],
       [ 3,  8, 13],
       [ 6, 11, 16],
       [ 9, 14, 19]])
```

65	$s = np.arange(0, 12, 3)$ s
----	----------------------------------

```
array([0, 3, 6, 9])
```

66	r + s
----	-------

ValueError Traceback (most recent call last)
 <ipython-input-130-ee3ad298df3e> in <cell line: 2>()
 1 # causes intentional error
 ----> 2 r + s

ValueError: operands could not be broadcast together with shapes (4,3) (4,)

67	r.transpose() + s
----	-------------------

array([[0, 6, 12, 18],
 [1, 7, 13, 19],
 [2, 8, 14, 20]])

68	sr = s.reshape(-1, 1) sr
----	-----------------------------

array([[0],
 [3],
 [6],
 [9]])

69	sr.shape
----	----------

(4, 1)

70	r + s.reshape(-1, 1)
----	----------------------

array([[0, 1, 2],
 [6, 7, 8],
 [12, 13, 14],
 [18, 19, 20]])

71	def f(x): return 3 * x + 5
----	-------------------------------

72	f(0.5)
----	--------

6.5

73	f(r)
----	------

array([[5, 8, 11],
 [14, 17, 20],
 [23, 26, 29],
 [32, 35, 38]])

대부분의 금융 관련 문제와 알고리즘은 배열 형태로 주어진다. NumPy는 `numpy.ndarray`라는 특수한 클래스를 제공하는데 이를 이용하면 코드를 편하고 깔끔하게 만드는 것과 동시에 성능도 향상시킬 수 있다.

3. Data Analysis with pandas

Data! Data! Data! I can't make bricks without clay!

—Sherlock Holmes

3-1 pandas 기초

1	<code>import pandas as pd</code>
---	----------------------------------

DataFrame 클래스는 인덱스와 라벨이 붙어있는 자료를 다루기 위해 설계
excel의 워크시트와 큰 차이 없음

2	<code>df = pd.DataFrame([10, 20, 30, 40], columns=['numbers'], index=['a', 'b', 'c', 'd'])</code> <code>df</code> numbers a 10 b 20 c 30 d 40
---	---

DataFrame 자료는 열(column)로 구성되고, 각 열은 이름을 가질 수 있음
인덱스는 문자열, 숫자, 시간 정보 등으로 구성할 수 있음

3	<code>df.index</code> <code>Index(['a', 'b', 'c', 'd'], dtype='object')</code>
---	---

4	<code>df.columns</code> <code>Index(['numbers'], dtype='object')</code>
---	--

5	<code>df.loc['c']</code> numbers 30 Name: c, dtype: int64
---	--

6	<code>df.loc[['a', 'd']] # 여러 개의 인덱스를 사용한 선택</code> numbers a 10 d 40
---	--

7	<code>df.iloc[1:3] # index 객체를 사용한 선택</code> numbers b 20 c 30
---	---

8	df.sum() # 열의 합
---	-----------------

numbers 100

dtype: int64

9	df.apply(lambda x: x ** 2) # lambda 함수를 사용하여 각 원소의 제곱
---	---

numbers

a 100

b 400

c 900

d 1600

NumPy ndarray 객체에 대해서 벡터화 연산을 구현하듯이 DataFrame 객체에도 벡터 연산을 구현할 수 있음

10	df ** 2
----	---------

numbers

a 100

b 400

c 900

d 1600

11	df['floats'] = (1.5, 2.5, 3.5, 4.5) # 새 열을 생성 df
----	---

numbers floats

a 10 1.5

b 20 2.5

c 30 3.5

d 40 4.5

12	df['floats'] # 열 선택
----	---------------------

a 1.5

b 2.5

c 3.5

d 4.5

Name: floats, dtype: float64

DataFrame 객체에 새로운 열을 정의할 수 있는데, 이 경우 인덱스에 맞추어 자료가 정렬

13	df['names'] = pd.DataFrame(['Yves', 'Sandra', 'Lilli', 'Henry'], index=['d', 'a', 'b', 'c']) df
----	---

numbers floats names

a 10 1.5 Sandra

b 20 2.5 Lilli

c 30 3.5 Henry

d 40 4.5 Yves

자료를 추가하는 경우 인덱스가 단순 정수인덱스로 변경되는 부작용 발생 가능

14	df.append({'numbers': 100, 'floats': 5.75, 'names': 'Jil'}, ignore_index=True) # 임시 객체 생성, df 자체는 변경되지 않음
----	---

<ipython-input-14-3cf68fa81a87>:1: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

```
df.append({'numbers': 100, 'floats': 5.75, 'names': 'Jil'},
numbers      floats  names
0          10     1.50  Sandra
1          20     2.50   Lilli
2          30     3.50   Henry
3          40     4.50    Yves
4          100     5.75     Jil
```

15	df = df.append(pd.DataFrame({'numbers': 100, 'floats': 5.75, 'names': 'Jil'}, index=['y',])) df
----	---

```
numbers      floats  names
a          10     1.50  Sandra
b          20     2.50   Lilli
c          30     3.50   Henry
d          40     4.50    Yves
y          100     5.75     Jil
```

16	df = df.append(pd.DataFrame({'names': 'Liz'}, index=['z',]), sort=False) df
----	--

```
numbers      floats  names
a          10.0     1.50  Sandra
b          20.0     2.50   Lilli
c          30.0     3.50   Henry
d          40.0     4.50    Yves
y          100.0     5.75     Jil
z           NaN     NaN     Liz
```

결측치는 NaN(Not a Number)로 표시됨

17	df.dtypes
----	-----------

```
numbers      float64
floats       float64
names        object
dtype: object
```

결측치가 있어도 대부분의 메서드 호출은 정상적으로 작동

18	df[['numbers', 'floats']].mean()
----	----------------------------------

```

numbers    40.00
floats      3.55
dtype: float64

```

19	df[['numbers', 'floats']].std()
----	---------------------------------

```

numbers    35.355339
floats      1.662077
dtype: float64

```

2-2 DataFrame Class 다루기 2단계

20	import numpy as np np.random.seed(100)
----	---

9개의 행과 4개의 열을 가진 NumPy ndarray 형태의 표준정규분포 난수를 생성

21	a = np.random.standard_normal((9, 4)) a
----	--

```

array([[ -1.74976547,  0.3426804 ,  1.1530358 , -0.25243604],
       [ 0.98132079,  0.51421884,  0.22117967, -1.07004333],
       [-0.18949583,  0.25500144, -0.45802699,  0.43516349],
       [-0.58359505,  0.81684707,  0.67272081, -0.10441114],
       [-0.53128038,  1.02973269, -0.43813562, -1.11831825],
       [ 1.61898166,  1.54160517, -0.25187914, -0.84243574],
       [ 0.18451869,  0.9370822 ,  0.73100034,  1.36155613],
       [-0.32623806,  0.05567601,  0.22239961, -1.443217  ],
       [-0.75635231,  0.81645401,  0.75044476, -0.45594693]])

```

DataFrame 객체를 바로 생성할 수 있지만 ndarray 객체를 만들고 이를 인수로 하여 DataFrame 객체를 생성하는 것이 일반적으로 좋은 선택. 이런 방식이 금융에서 일반적으로 쓰이는 방법

22	df = pd.DataFrame(a) df
----	----------------------------

```

      0         1         2         3
0  -1.749765  0.342680  1.153036 -0.252436
1   0.981321  0.514219  0.221180 -1.070043
2  -0.189496  0.255001 -0.458027  0.435163
3  -0.583595  0.816847  0.672721 -0.104411
4  -0.531280  1.029733 -0.438136 -1.118318
5   1.618982  1.541605 -0.251879 -0.842436
6   0.184519  0.937082  0.731000  1.361556
7  -0.326238  0.055676  0.222400 -1.443217
8  -0.756352  0.816454  0.750445 -0.455947

```

23	df.columns = ['No1', 'No2', 'No3', 'No4'] # 열 이름 지정 df			
	No1	No2	No3	No4
0	-1.749765	0.342680	1.153036	-0.252436
1	0.981321	0.514219	0.221180	-1.070043
2	-0.189496	0.255001	-0.458027	0.435163
3	-0.583595	0.816847	0.672721	-0.104411
4	-0.531280	1.029733	-0.438136	-1.118318
5	1.618982	1.541605	-0.251879	-0.842436
6	0.184519	0.937082	0.731000	1.361556
7	-0.326238	0.055676	0.222400	-1.443217
8	-0.756352	0.816454	0.750445	-0.455947

24	df['No2'][3] # No2 열의 3번 인덱스 위치 자료 0.816847071685779
----	---

25	df['No2'].mean() # No2 열의 평균 0.7010330941456459
----	--

26	dates = pd.date_range('2023-1-1', periods=9, freq='M') dates DatetimeIndex(['2023-01-31', '2023-02-28', '2023-03-31', '2023-04-30', '2023-05-31', '2023-06-30', '2023-07-31', '2023-08-31', '2023-09-30'], dtype='datetime64[ns]', freq='M')
----	---

pandas는 시간 index 처리를 잘함. date_range함수를 사용하여 2023-1-1부터 시작하는 월 말 시점 9개를 생성

27	df.index = dates # DatetimeIndex로 DataFrame 인덱스 명칭 변경 df			
	No1	No2	No3	No4
2023-01-31	-1.749765	0.342680	1.153036	-0.252436
2023-02-28	0.981321	0.514219	0.221180	-1.070043
2023-03-31	-0.189496	0.255001	-0.458027	0.435163
2023-04-30	-0.583595	0.816847	0.672721	-0.104411
2023-05-31	-0.531280	1.029733	-0.438136	-1.118318
2023-06-30	1.618982	1.541605	-0.251879	-0.842436
2023-07-31	0.184519	0.937082	0.731000	1.361556
2023-08-31	-0.326238	0.055676	0.222400	-1.443217
2023-09-30	-0.756352	0.816454	0.750445	-0.455947

28	df.values
----	-----------

```
array([[ -1.74976547,  0.3426804 ,  1.1530358 , -0.25243604],
       [ 0.98132079,  0.51421884,  0.22117967, -1.07004333],
       [-0.18949583,  0.25500144, -0.45802699,  0.43516349],
       [-0.58359505,  0.81684707,  0.67272081, -0.10441114],
       [-0.53128038,  1.02973269, -0.43813562, -1.11831825],
       [ 1.61898166,  1.54160517, -0.25187914, -0.84243574],
       [ 0.18451869,  0.9370822 ,  0.73100034,  1.36155613],
       [-0.32623806,  0.05567601,  0.22239961, -1.443217  ],
       [-0.75635231,  0.81645401,  0.75044476, -0.45594693]])
```

29	np.array(df).round(6)
----	-----------------------

```
array([[ -1.749765,  0.34268 ,  1.153036, -0.252436],
       [ 0.981321,  0.514219,  0.22118 , -1.070043],
       [-0.189496,  0.255001, -0.458027,  0.435163],
       [-0.583595,  0.816847,  0.672721, -0.104411],
       [-0.53128 ,  1.029733, -0.438136, -1.118318],
       [ 1.618982,  1.541605, -0.251879, -0.842436],
       [ 0.184519,  0.937082,  0.731 ,  1.361556],
       [-0.326238,  0.055676,  0.2224 , -1.443217],
       [-0.756352,  0.816454,  0.750445, -0.455947]])
```

pandas의 DataFrame명령을 사용하면 ndarray 객체를 DataFrame으로 생성할 수 있으며, 반대로 NumPy의 array명령으로 DataFrame객체를 ndarray 객체로 변환할 수 있다.

3-3 기초적인 분석

30	df.info()
----	-----------

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 9 entries, 2023-01-31 to 2023-09-30
Freq: M
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   No1     9 non-null       float64
 1   No2     9 non-null       float64
 2   No3     9 non-null       float64
 3   No4     9 non-null       float64
dtypes: float64(4)
memory usage: 360.0 bytes
```

31	df.describe()			
	No1	No2	No3	No4
count	9.000000	9.000000	9.000000	9.000000
mean	-0.150212	0.701033	0.289193	-0.387788
std	0.988306	0.457685	0.579920	0.877532
min	-1.749765	0.055676	-0.458027	-1.443217
25%	-0.583595	0.342680	-0.251879	-1.070043
50%	-0.326238	0.816454	0.222400	-0.455947
75%	0.184519	0.937082	0.731000	-0.104411
max	1.618982	1.541605	1.153036	1.361556

수치자료에 대해 자주 사용되는 통계수치는 describe()로 한번에 구할 수 있다.

32	df.sum()			
No1	-1.351906			
No2	6.309298			
No3	2.602739			
No4	-3.490089			
dtype: float64				

33	df.mean()			
No1	-0.150212			
No2	0.701033			
No3	0.289193			
No4	-0.387788			
dtype: float64				

34	df.mean(axis=0) # 열 기준 평균			
No1	-0.150212			
No2	0.701033			
No3	0.289193			
No4	-0.387788			
dtype: float64				

35	df.mean(axis=1) # 행 기준 평균			
2023-01-31	-0.126621			
2023-02-28	0.161669			
2023-03-31	0.010661			
2023-04-30	0.200390			
2023-05-31	-0.264500			
2023-06-30	0.516568			
2023-07-31	0.803539			
2023-08-31	-0.372845			

2023-09-30 0.088650

Freq: M, dtype: float64

36	df.cumsum() # 누적 합계			
	No1	No2	No3	No4
2023-01-31	-1.749765		0.342680	1.153036
2023-02-28	-0.768445		0.856899	1.374215
2023-03-31	-0.957941		1.111901	0.916188
2023-04-30	-1.541536		1.928748	1.588909
2023-05-31	-2.072816		2.958480	1.150774
2023-06-30	-0.453834		4.500086	0.898895
2023-07-31	-0.269316		5.437168	1.629895
2023-08-31	-0.595554		5.492844	1.852294
2023-09-30	-1.351906		6.309298	2.602739

37 np.log(df) # log의 ()안은 양수가 들어가야 함, warning 발생
/usr/local/lib/python3.10/dist-packages/pandas/core/internals/blocks.py:351:
RuntimeWarning: invalid value encountered in log

	No1	No2	No3	No4
2023-01-31	NaN	-1.070957	0.142398	NaN
2023-02-28	-0.018856	-0.665106	-1.508780	NaN
2023-03-31	NaN	-1.366486	NaN	-0.832033
2023-04-30	NaN	-0.202303	-0.396425	NaN
2023-05-31	NaN	0.029299	NaN	NaN
2023-06-30	0.481797	0.432824	NaN	NaN
2023-07-31	-1.690005	-0.064984	-0.313341	0.308628
2023-08-31	NaN	-2.888206	-1.503279	NaN
2023-09-30	NaN	-0.202785	-0.287089	NaN

38	np.sqrt(abs(df)) # 절대값으로 변경 후 루트 값을 계산함			
	No1	No2	No3	No4
2023-01-31	1.322787	0.585389	1.073795	0.502430
2023-02-28	0.990616	0.717091	0.470297	1.034429
2023-03-31	0.435311	0.504977	0.676777	0.659669
2023-04-30	0.763934	0.903796	0.820196	0.323127
2023-05-31	0.728890	1.014757	0.661918	1.057506
2023-06-30	1.272392	1.241614	0.501876	0.917843
2023-07-31	0.429556	0.968030	0.854986	1.166857
2023-08-31	0.571173	0.235958	0.471593	1.201340
2023-09-30	0.869685	0.903578	0.866282	0.675238

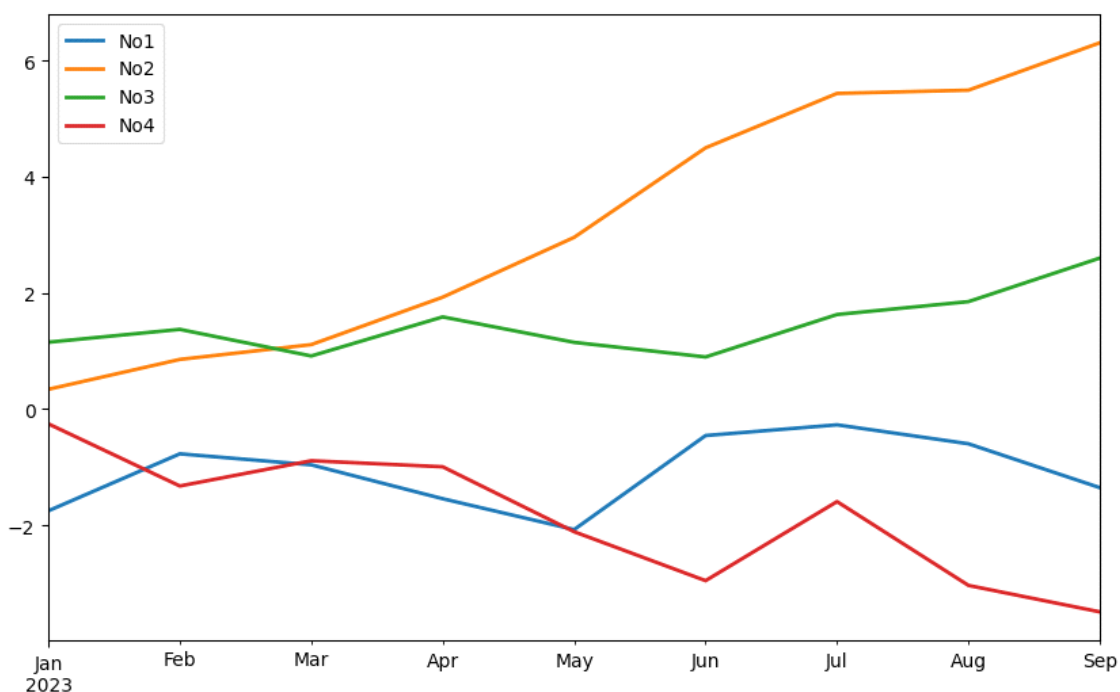
39	100 * df + 100			
	No1	No2	No3	No4
2023-01-31	-74.976547	134.268040	215.303580	74.756396
2023-02-28	198.132079	151.421884	122.117967	-7.004333
2023-03-31	81.050417	125.500144	54.197301	143.516349
2023-04-30	41.640495	181.684707	167.272081	89.558886
2023-05-31	46.871962	202.973269	56.186438	-11.831825
2023-06-30	261.898166	254.160517	74.812086	15.756426
2023-07-31	118.451869	193.708220	173.100034	236.155613
2023-08-31	67.376194	105.567601	122.239961	-44.321700
2023-09-30	24.364769	181.645401	175.044476	54.405307

3-4 기초적인 시각화

40	<pre>from pylab import plt, mpl plt.style.use('seaborn-v0_8') mpl.rcParams['font.family'] = 'serif' %matplotlib inline</pre>
----	--

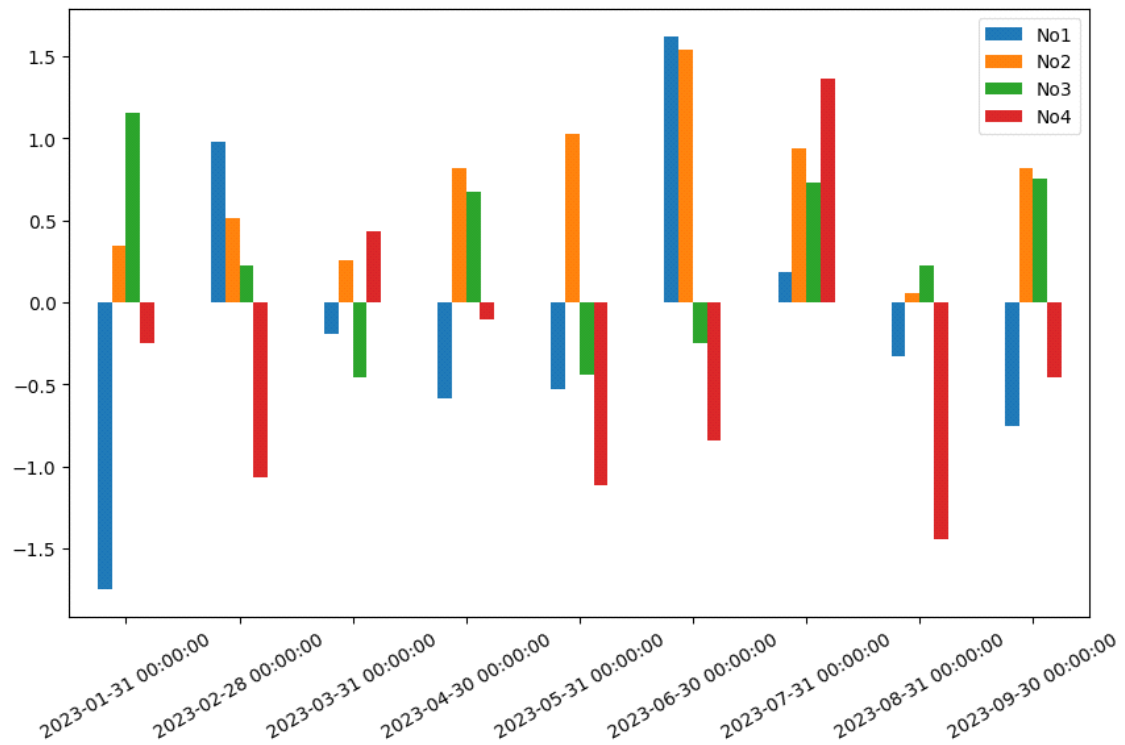
%matplotlib inline은 그래프를 notebook에서 표시하도록 만드는 magic function

41	<code>df.cumsum().plot(lw=2.0, figsize=(10, 6));</code>
----	---



plot 함수로 선 그래프 표시

42	<code>df.plot.bar(figsize=(10, 6), rot=30);</code>
----	--



bar 차트 구현됨

2-5 Series Class

43	type(df)
----	----------

pandas.core.frame.DataFrame

44	S = pd.Series(np.linspace(0, 15, 7), name='series') S
----	--

0 0.0

1 2.5

2 5.0

3 7.5

4 10.0

5 12.5

6 15.0

Name: series, dtype: float64

45	type(S)
----	---------

pandas.core.series.Series

pandas에는 단일 시계열 전용인 Series 클래스도 존재

46	s = df['No1'] s
----	--------------------

```

2023-01-31    -1.749765
2023-02-28     0.981321
2023-03-31    -0.189496
2023-04-30    -0.583595
2023-05-31    -0.531280
2023-06-30     1.618982
2023-07-31     0.184519
2023-08-31    -0.326238
2023-09-30    -0.756352
Freq: M, Name: No1, dtype: float64

```

47	type(s)
----	---------

```
pandas.core.series.Series
```

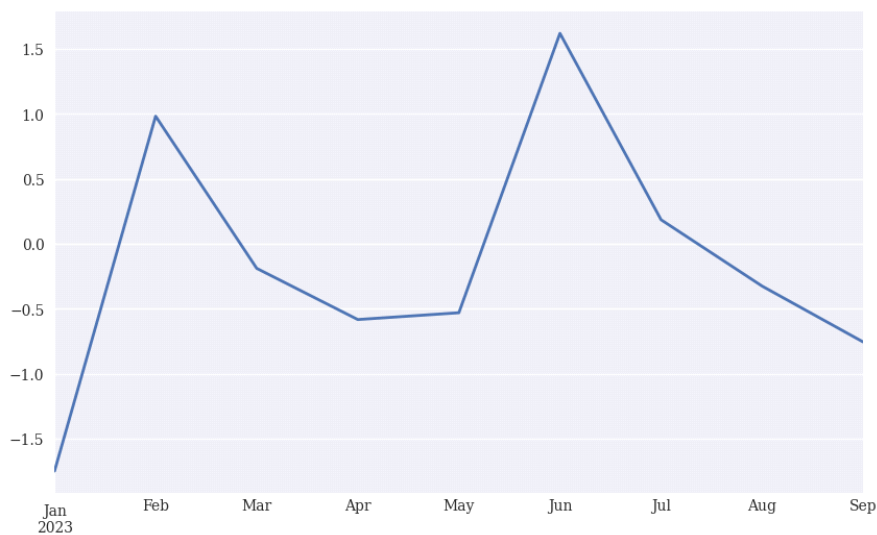
DataFrame 객체에서 하나의 열만 선택하면 Series 객체가 된다.

48	s.mean()
----	----------

```
-0.15021177307319458
```

DataFrame의 주요 메서드는 Series 객체에도 사용 가능

49	s.plot(lw=2.0, figsize=(10, 6));
----	----------------------------------



2-6 GroupBy 연산

pandas는 groupby라는 강력하고 유연한 그룹 지정 기능을 보유

50	df['Quarter'] = ['Q1', 'Q1', 'Q1', 'Q2', 'Q2', 'Q2', 'Q3', 'Q3', 'Q3'] df
----	---

	No1	No2	No3	No4	Quarter
2023-01-31	-1.749765	0.342680	1.153036	-0.252436	Q1
2023-02-28	0.981321	0.514219	0.221180	-1.070043	Q1
2023-03-31	-0.189496	0.255001	-0.458027	0.435163	Q1
2023-04-30	-0.583595	0.816847	0.672721	-0.104411	Q2
2023-05-31	-0.531280	1.029733	-0.438136	-1.118318	Q2
2023-06-30	1.618982	1.541605	-0.251879	-0.842436	Q2
2023-07-31	0.184519	0.937082	0.731000	1.361556	Q3
2023-08-31	-0.326238	0.055676	0.222400	-1.443217	Q3
2023-09-30	-0.756352	0.816454	0.750445	-0.455947	Q3

51	groups = df.groupby('Quarter')
----	--------------------------------

그룹을 지정하면 그룹별 개수, 평균, 최댓값 등을 쉽게 구할 수 있다.

52	groups.size()
----	---------------

Quarter

Q1 3

Q2 3

Q3 3

dtype: int64

53	groups.mean()
----	---------------

	No1	No2	No3	No4
Quarter				
Q1	-0.319314	0.370634	0.305396	-0.295772
Q2	0.168035	1.129395	-0.005765	-0.688388
Q3	-0.299357	0.603071	0.567948	-0.179203

54	groups.max()
----	--------------

	No1	No2	No3	No4
Quarter				
Q1	0.981321	0.514219	1.153036	0.435163
Q2	1.618982	1.541605	0.672721	-0.104411
Q3	0.184519	0.937082	0.750445	1.361556

55	groups.aggregate([min, max]).round(2)
----	---------------------------------------

	No1		No2		No3		No4	
	min	max	min	max	min	max	min	max
Quarter								
Q1	-1.75	0.98	0.26	0.51	-0.46	1.15	-1.07	0.44
Q2	-0.58	1.62	0.82	1.54	-0.44	0.67	-1.12	-0.10
Q3	-0.76	0.18	0.06	0.94	0.22	0.75	-1.44	1.3

여러 개의 열을 기준으로 그룹 지정을 하는 것도 가능

56	<code>df['Odd_Even'] = ['Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']</code>
----	---

57	<code>groups = df.groupby(['Quarter', 'Odd_Even'])</code>
----	---

58	<code>groups.size()</code>
----	----------------------------

Quarter	Odd_Even	
Q1	Even	1
	Odd	2
Q2	Even	2
	Odd	1
Q3	Even	1
	Odd	2

dtype: int64

59	<code>groups[['No1', 'No4']].aggregate([sum, np.mean])</code>
----	---

		No1		No4	
		sum	mean	sum	mean
Quarter	Odd_Even				
Q1	Even	0.981321	0.981321	-1.070043	-1.070043
	Odd	-1.939261	-0.969631	0.182727	0.091364
Q2	Even	1.035387	0.517693	-0.946847	-0.473423
	Odd	-0.531280	-0.531280	-1.118318	-1.118318
Q3	Even	-0.326238	-0.326238	-1.443217	-1.443217
	Odd	-0.571834	-0.285917	0.905609	0.452805

2-7. 데이터 연결, 결합, 병합(Concatenation, Joining and Merging)

60	<code>df1 = pd.DataFrame(['100', '200', '300', '400'], index=['a', 'b', 'c', 'd'], columns=['A',])</code> <code>df1</code>
----	---

	A
a	100
b	200
c	300
d	400

61	df2 = pd.DataFrame(['200', '150', '50'], index=['f', 'b', 'd'], columns=['B',]) df2
----	--

	B
f	200
b	150
d	50

Concatenation

62	df1.append(df2, sort=False)
----	-----------------------------

	A	B
a	100	NaN
b	200	NaN
c	300	NaN
d	400	NaN
f	NaN	200
b	NaN	150
d	NaN	50

63	df1.append(df2, ignore_index=True, sort=False)
----	--

	A	B
0	100	NaN
1	200	NaN
2	300	NaN
3	400	NaN
4	NaN	200
5	NaN	150
6	NaN	50

64	pd.concat((df1, df2), sort=False)
----	-----------------------------------

	A	B
a	100	NaN
b	200	NaN
c	300	NaN
d	400	NaN

f	NaN	200
b	NaN	150
d	NaN	50

65	pd.concat((df1, df2), ignore_index=True, sort=False)	
	A	B
0	100	NaN
1	200	NaN
2	300	NaN
3	400	NaN
4	NaN	200
5	NaN	150
6	NaN	50

Joining

66	df1.join(df2)	
	A	B
a	100	NaN
b	200	150
c	300	NaN
d	400	50

67	df2.join(df1)	
	B	A
f	200	NaN
b	150	200
d	50	400

68	df1.join(df2, how='left')	
	A	B
a	100	NaN
b	200	150
c	300	NaN
d	400	50

69	df1.join(df2, how='right')	
	A	B
f	NaN	200
b	200	150
d	400	50

70	<code>df1.join(df2, how='outer')</code>
----	---

	A	B
a	100	NaN
b	200	150
c	300	NaN
d	400	50
f	NaN	200

71	<code>df = pd.DataFrame()</code>
----	----------------------------------

72	<code>df['A'] = df1['A']</code> <code>df</code>
----	--

	A
a	100
b	200
c	300
d	400

73	<code>df['B'] = df2</code> <code>df</code>
----	---

	A	B
a	100	NaN
b	200	150
c	300	NaN
d	400	50

74	<code>df = pd.DataFrame({'A': df1['A'], 'B': df2['B']})</code> <code>df</code>
----	---

	A	B
a	100	NaN
b	200	150
c	300	NaN
d	400	50
f	NaN	200

Merging

75	<code>c = pd.Series([250, 150, 50], index=['b', 'd', 'c'])</code> <code>df1['C'] = c</code> <code>df2['C'] = c</code>
----	---

76	df1	
----	-----	--

	A	C
a	100	NaN
b	200	250.0
c	300	50.0
d	400	150.0

77	df2	
----	-----	--

	B	C
f	200	NaN
b	150	250.0
d	50	150.0

78	pd.merge(df1, df2)		
----	--------------------	--	--

	A	C	B
0	100	NaN	200
1	200	250.0	150
2	400	150.0	50

79	pd.merge(df1, df2, on='C')		
----	----------------------------	--	--

	A	C	B
0	100	NaN	200
1	200	250.0	150
2	400	150.0	50

80	pd.merge(df1, df2, how='outer')		
----	---------------------------------	--	--

	A	C	B
0	100	NaN	200
1	200	250.0	150
2	300	50.0	NaN
3	400	150.0	50