

9 파이썬 라이브러리 삼총사

다음은 파이썬에서 가장 많이 사용하는 NumPy, Matplotlib, Pandas 에 대한 소개와 간단한 사용법을 알아보려고 한다.

9.1 수학과 과학연산 NumPy 와 SciPy

파이썬은 행렬이나 배열에 대한 수학 연산을 제공하지 않는다. 그래서 과학적 프로그래밍을 위한 라이브러리인 SciPy 와 NumPy 가 필요하다.

NumPy 는 수학 및 과학 연산을 위해 행렬이나 대규모 다차원 배열을 쉽게 처리할 수 있는 파이썬의 라이브러리이다. NumPy 는 데이터 구조 외에도 수치 계산을 위해 효율적으로 구현된 기능을 제공한다.

SciPy 는 NumPy 를 기반으로 만들어져 적분과 미분방정식 그리고 포트폴리오 최적화를 위해 꼭 필요한 optimize 모듈등 다양한 고급 수학 함수들을 제공한다.

NumPy와 SciPy 는 1995 년과 2001 년경에 공개되었는 데, 모두 트레이비스 올리펀트(Travis Oliphant)가 개발을 주도하였다. 올리펀트라는 인물은 아직 생소하겠지만 그는 미국의 데이터사이언티스트이자 파이썬계에서 그 유명한 아나콘다(Anaconda)의 창립자이다.

NumPy 를 사용하려면 다른 라이브러리와 마찬가지로 임포트를 먼저 해야 한다.

```
import numpy as np
```

9.1.1 배열과 행렬 만들기

배열과 행렬을 만드는 여러 가지 방법이 있지만 대표적인 몇 가지 방법을 간단히 살펴보면 다음과 같다.

리스트를 먼저 만들고 array 함수를 사용하여 배열을 만들 수 있다.

```
vec = [1, 2, 3, 4] # 4 개의 원소를 가진 리스트 vec 를 만든다
```

```
vecA = np.array(vec) # 리스트를 배열로 바꾼다
```

```
print(vecA)
```

결과:

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

array 함수로 행렬을 만들 수 있는 데, `[[],[]]` 형태로 복잡해 보이지만 안쪽의 `[]`는 행을 의미한다. `[1,2,3]`과 `[4,5,6]`은 행이다. 그리고 열은 쉼표로 구분된다.

```
vecB = np.array( [ [1, 2, 3], [4, 5, 6] ] ) # array()함수에 직접 리스트를 입력한다
```

결과:

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

zeros 함수는 행렬의 구성요소가 모두 0 인 행렬을 만들어주는 데, 행렬의 차원만 지정하면 된다

```
zeros = np.zeros( ( 2, 2 ) ) # 원소가 모두 0 인 2x2 행렬을 만든다
print(zeros)
```

결과:

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

ones 함수는 zeros 와 마찬가지로 행렬구성요소가 모두 1 인 단위행렬을 만든다. zeros 와 마찬가지로 행렬의 차원만 지정하면 된다.

```
ones = np.ones( ( 2, 3 ) ) # 원소가 모두 1 인 2x3 행렬을 만든다
print(ones)
```

결과:

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

full 함수는 지정한 행렬의 차원으로 지정한 값을 채운다.

```
fives = np.full( (2, 3), 5 ) # 원소가 모두 5 인 2x3 행렬을 만든다
print(fives)
```

결과:

```
[[5 5 5]
 [5 5 5]]
```

eye 함수는 지정한 크기의 항등행렬(대각행렬)을 만든다.

```
eye = np.eye(3) # 대각선상의 원소가 모두 1 인 3x3 행렬을 만든다
```

```
print(eye)
```

결과:

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

앞서 array 함수로 배열을 만들었다. 이 배열의 차원을 reshape 함수로 바꾸면 행렬을 만들 수 있다.

```
reshape = np.array( range(20) ).reshape( (4, 5) ) #20 개의 리스트를 만들어 4x5 의  
2 차원 배열을 만든다  
print(reshape)
```

결과:

```
[[ 0  1  2  3  4]  
 [ 5  6  7  8  9]  
 [10 11 12 13 14]  
 [15 16 17 18 19]]
```

9.1.2 배열과 행렬의 속성

배열과 행렬의 속성관련함수로는 dtype, shape, ndim 등이 있는 데, dtype 은 데이터타입, shape 는 행과 열의 갯수, ndim 은 차원을 돌려준다.

```
print(vecA.dtype) # 데이터 타입, int64 는 저장된 데이터의 타입이다  
print(vecA.shape) # 행과 열의 갯수, (4,)는 4 개의 원소를 가진 배열임을 의미한다  
print(vecA.ndim) # 차원, 1 을 돌려준다. 즉 1 차원 배열임을 알 수 있다
```

결과:

```
int64  
(4,)  
1
```

```
print(vecB.dtype) # 데이터 타입, int64 는 저장된 데이터의 타입이다  
print(vecB.shape) # 행과 열의 갯수, (2, 3)은 2 행 3 열이라는 의미이다  
print(vecB.ndim) # 차원을 돌려준다, 2 는 2 차원 배열임을 의미한다
```

결과:

```
int64  
(2, 3)
```

9.1.3 연산

NumPy 는 배열이나 행렬의 연산이 무척 빠르다. 순수한 파이썬 코드로 작성한다면 이들 연산을 위해 여러 개의 반복문을 사용하는 데, 반복문은 코드의 크기도 늘어나고 속도도 느리면 직관적이지 못하다. 그러나 NumPy 는 연산을 위한 반복문이 필요없으며 직관적으로 수학적 연산표기처럼 표현할 수 있다.

```
print(vecA*2+1) # 2 를 곱한 후 1 을 더하는 연산
print(vecA+vecA) # 두 array 개체 더하기
print(vecA-vecA) # 두 array 개체 빼기
print(1/vecA)
print(vecA**2)
print(vecA%2)
print(np.dot(vecA.T, vecA)) # 벡터와 행렬의 내적 계산(product)은 dot 함수를 사용
```

9.1.4 인덱싱/ 슬라이싱

파이썬 리스트의 인덱스가 0 부터 시작하는 것과 마찬가지로 NumPy 역시 그런 파이썬 방식을 따른다. 그리고 인덱스는 []안에 표시한다.

```
# 배열이나 행렬의 원소에 접근하려면 []를 사용한다
print(vecA[0], vecA[1], vecA[2], vecA[3]) # 배열의 각 인덱스의 값을 출력한다
print(vecB[0,0],vecB[0,1],vecB[0,2]) # 2 차원배열이라면 행과 열 인덱스로 출력한다
print(vecB[1,0],vecB[1,1],vecB[1,2])
```

결과:

```
1 2 3 4
1 2 3
4 5 6
```

파이썬의 리스트와 마찬가지로 NumPy 배열에서 인덱스에 -를 사용한다는 것은 역방향을 의미한다. 가령 19, 29, 39, 49 네 개의 원소를 가진 array 를 만들었는데, (순방향)인덱스는 데이터의 순서대로 0, 1, 2, 3 이다. 그리고 역방향 인덱스 역시 데이터의 순서대로 -4, -3, -2, -1 이다. 즉 0 과 -4, 1 과 -3, 2 와 -2, 3 과 -1 은 모두 같은 데이터를 가리키는 인덱스이다.

그림 177

순방향 인덱스	0	1	2	3
	19	29	39	49
역방향 인덱스	-4	-3	-2	-1

```
import numpy as np

vec = np.array([19, 29, 39, 49])
vec[-4]
19

vec[-3]
29

vec[-2]
39

vec[-1]
49
```

그러나 인덱스를 벗어난 값을 지정하면 에러가 난다.

```
vec[-5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index -5 is out of bounds for axis 0 with size 4
```

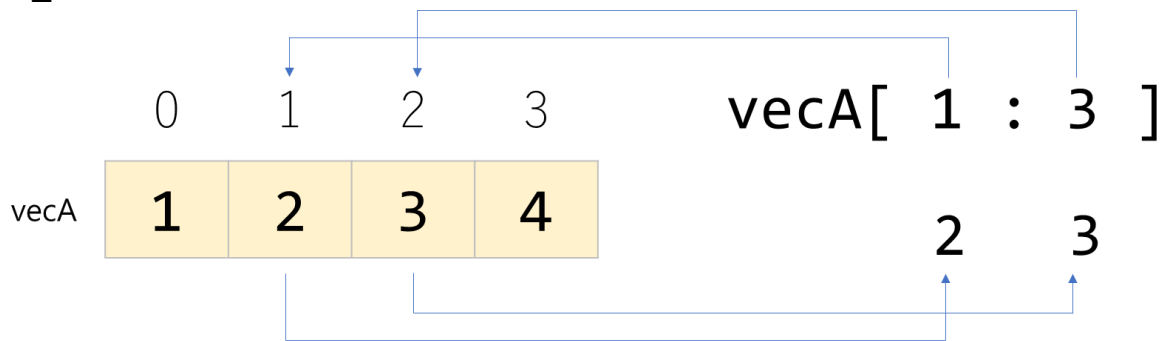
다음은 동일한 데이터를 가리키는 순방향과 역방향 인덱스를 사용한 예이다.

```
print(vecA[0], vecA[-4])
print(vecA[1], vecA[-3])
print(vecA[2], vecA[-2])
print(vecA[3], vecA[-1])
```

하나의 값이 아니라 여러 개의 값을 가져오려면 :(콜론)를 사용하여 범위를 [첫 인덱스 : 마지막 인덱스] 형식으로 지정한다. 이때 주의할 점은 마지막

인덱스까지가 아니라 마지막 인덱스의 직전 인덱스가 가리키는 값까지만 가져온다.

그림 178



`print(vecA[1:3])` # 인덱스 1 과 인덱스 3 이 아니라 2 가 가리키는 값을 가져온다.

9.1.5 난수만들기

난수와 관련하여 파이썬의 기본 라이브러리중 `random` 이 있지만 NumPy 도 `random` 모듈을 제공하여 여러 난수관련함수를 제공한다.

random.seed

난수에 필요한 시드값을 설정

```
np.random.seed(0)
print(np.random.rand(2, 3))
```

random.rand

난수배열 생성

```
print(np.random.rand(5))
print(np.random.rand(2, 3))
```

random.choice

1 차원배열에 임의의 샘플을 생성

```
#np.arange(5)에서 샘플 3 개를 추출한 1 차원 배열
print(np.random.choice(5, 3))
```

```
#np.arange(10)에서 샘플을 추출하여 (2, 3)의 2 차원 배열 생성
print(np.random.choice(10, (2, 3)))
```

random.randint

시작값≤임의의 난수<끝값, 시작값을 포함하여 끝값을 포함하지 않는 정수
난수생성

```
print(np.random.randint(2, size=5)) # 0≤난수<2
print(np.random.randint(2, 4, size=5)) # 2≤난수<4
print(np.random.randint(1, 5, size=(2, 3))) # 1≤난수<5
```

random.randn

표준정규분포(standard normal distribution)를 따르는 난수 생성

randn 함수는 임의의 표준정규분포 데이터를 만든다

```
rnd_num = np.random.randn(4, 4)
```

```
print(rnd_num)
```

```
print(rnd_num>0) #생성한 난수중 0 이상인 값만 출력
```

```
print((rnd_num>0).sum()) # 0 이상의 난수값의 합계를 출력
```

```
print(rnd_num.mean(), rnd_num.std(), rnd_num.var()) #난수의 평균, 표준편차,  
분산출력
```

9.2 미술 담당, Matplotlib

데이터 시각화(Data Visualization)란 차트나 도표를 그리는 일을 말한다. 그리고 Matplotlib 는 데이터 시각화도구이다. 2002 년 미국의 신경생물학자인 존 헌터(John D. Hunter)에 의해 개발된 Matplotlib 는 파이썬의 배열을 가지고 2D 도표를 그리는 일을 한다.

Matplotlib 이전에는 Matlab 을 이용하여 각종 도표를 만들었는데, 존 헌터 역시 시카고 대학에서 신경생물학(Neurobiology)박사후 과정에서 Matlab 에 만족하지 못해 Matplotlib 을 만들기 시작했다고 한다. 그러나 안타깝게도 2012 년 대장암으로 아내와 세 딸을 남겨두고 44 세에 요절하였다.

그림 179



HOLDING PAGE

AUGUST 30, 2012

REST IN PEACE: JOHN HUNTER, MATPLOTLIB AUTHOR, FATHER HAS PASSED AWAY.

by jesse in Programming, Python

I was extremely sad in hearing about this this morning - John Hunter, author of matplotlib and tireless open source/Python community contributor has passed away after an intense and short battle with cancer. He is survived by wife and three daughters.



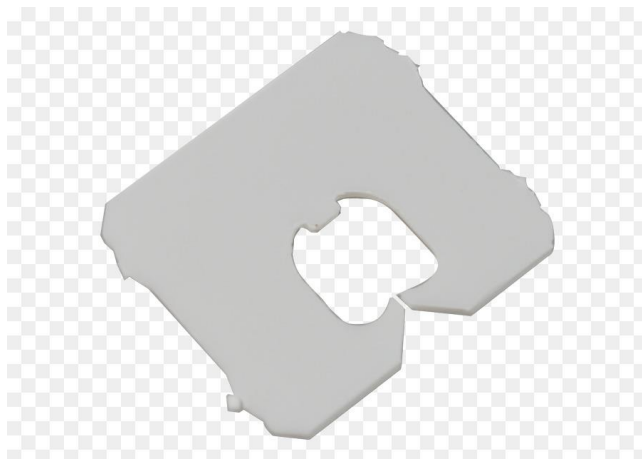
Many of us - both professionally and personally have benefited from John's work in the open source and Python community. Just a few weeks ago he delivered a keynote at the SciPy 2012

몇 가지 간단한 명령만으로 간단한 도표를 그릴 수 있으며 제한적이지만 3D 그래픽도 가능하다. 여기에서는 Matplotlib 로 라인차트나 막대차트 등 우리가 자주 사용하는 차트를 그리는 데 필요한 내용들만 다룰 것이다.

9.2.1 차트도해

빵집에서 식빵을 사면 비닐봉투를 묶는 플라스틱 '물건'이 있다. 많은 사람이 이 물건을 사용하지만, 정확한 명칭을 아는 사람은 별로 없다.

그림 180



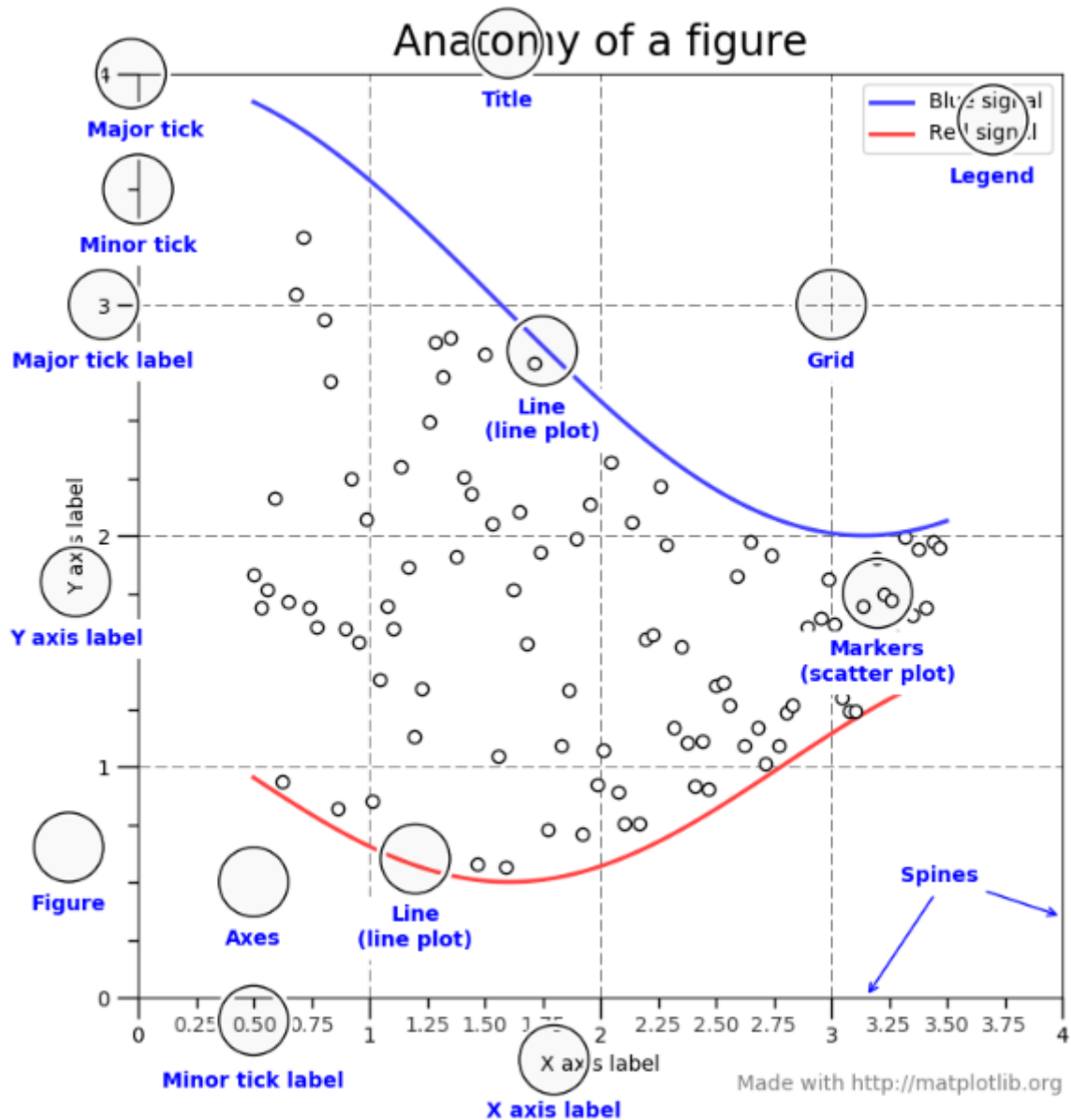
비록 사소한 물건일 수도 있지만, 사람이 접하는 대부분 사물에는 이름이 있다. 무심코 넘기는 차트 한 장에는 여러 가지 요소들이 있는 데, Matplotlib 를 다루기 위해서는 그런 요소들의 이름을 알아 둘 필요가 있다.

다음은 Matplotlib 공식문서중

<https://matplotlib.org/3.1.1/gallery/showcase/anatomy.html>

에서 제공하는 차트도해에 해당하는 그림인데, Matplotlib 를 이용하여 만들어 놓은 것이다. 다음의 그림에서 주요한 요소들만 챙겨본다면 차트의 전체영역은 우리가 차트라고 부르는 요소들이 모두 모아둔 그림판에 해당한다. 그래서 이를 figure 라고 한다. 우리는 전체 그림을 가지고 차트라고 하지만 선형 차트 또는 라인차트라고 부르는 것은 line plot 이라고 한다. 분산형 차트는 scatter plot 이라고 한다. plot 은 그림을 그린다는 의미인데, figure 가 전체임에 반해 plot 은 figure 의 일부분이다. 가로축과 세로축을 axis 라고 하며 축의 이름은 label, 축의 눈금은 tick, 차트의 제목은 title, 범례는 legend 이다.

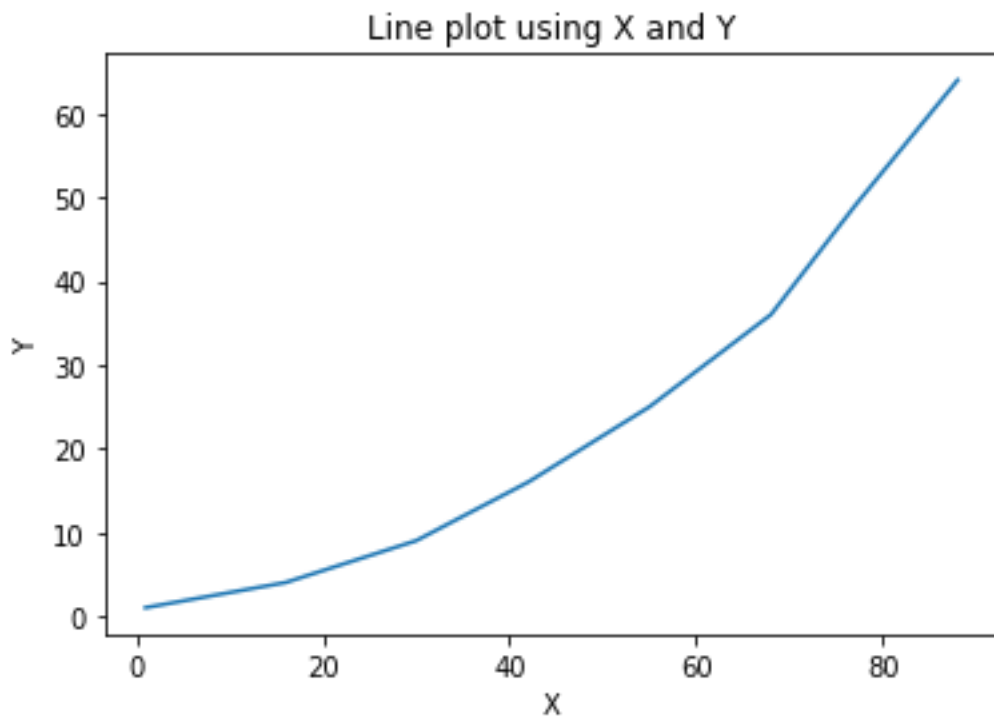
그림 181



9.2.2 라인차트(Line plots)

가장 기본적인 차트인 라인차트를 만드는 간단한 예를 만들어 보기로 하자. 차트는 matplotlib 를 임포트하는 것으로 시작한다. 그리고 차트에 그릴 데이터를 만들고 plot()함수에 데이터를 전달하고 show()함수로 차트를 그리는 것이다.

그림 182



<코드>라인차트

```
01: import matplotlib.pyplot as plt
02: y = [1, 4, 9, 16, 25, 36, 49, 64]
03: x = [1, 16, 30, 42, 55, 68, 77, 88]
04: plt.plot(x, y)
05: plt.xlabel('X')
06: plt.ylabel('Y')
07: plt.title('Line plot using X and Y')
08: plt.show()
```

</코드>

#Matplotlib 의 matplotlib.pyplot 를 plt 로 임포트한다

```
01: import matplotlib.pyplot as plt
```

#X 와 Y 축에 그릴 데이터를 만든다

```
02: y = [1, 4, 9, 16, 25, 36, 49, 64]
```

```
03: x = [1, 16, 30, 42, 55, 68, 77, 88]
```

plot()함수에 변수를 주어 라인차트를 만든다.

```
04: plt.plot(x, y)
```

X 축과 Y 축의 이름(xlabel 과 ylabel), 차트의 제목(title)을 지정한다

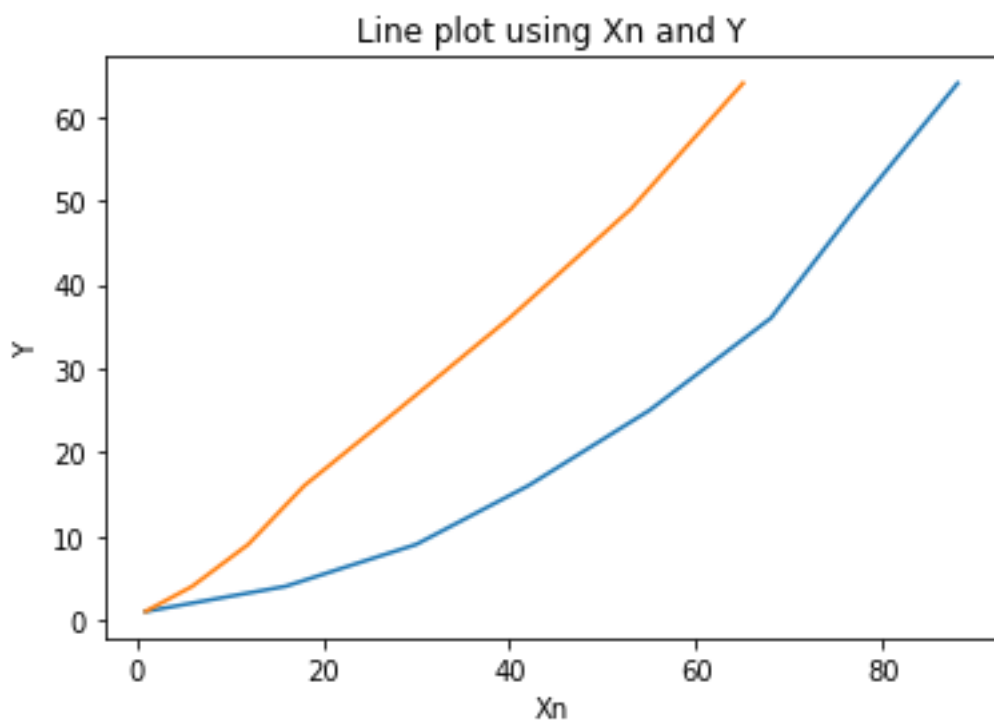
```

05: plt.xlabel('X')
06: plt.ylabel('Y')
07: plt.title('Line plot using X and Y')
# 차트를 화면에 그린다
08: plt.show()

```

그런데 앞서 간단한 차트에 라인(시리즈)을 하나 더 추가하려면 plot()함수를 한번 더 실행한다.

그림 183



<코드> 두 개의 시리즈를 가진 라인차트

```

01: import matplotlib.pyplot as plt
02: y = [1, 4, 9, 16, 25, 36, 49, 64]
03: x1 = [1, 16, 30, 42, 55, 68, 77, 88]
04: x2 = [1, 6, 12, 18, 28, 40, 53, 65]
05: plt.plot(x1, y)
06: plt.plot(x2, y)
07: plt.xlabel('Xn')

```

```
08: plt.ylabel('Y')
09: plt.title('Line plot using Xn and Y')
10: plt.show()
</코드>
```

```
01: import matplotlib.pyplot as plt
02: y = [1, 4, 9, 16, 25, 36, 49, 64]
# x1 과 x2 두 개의 리스트를 차트에 추가하기 위해 데이터를 입력한다
03: x1 = [1, 16, 30, 42, 55, 68, 77, 88]
04: x2 = [1, 6, 12, 18, 28, 40, 53, 65]
# plot()함수를 사용하여 차트에 두 개의 시리즈를 추가한다
05: plt.plot(x1, y)
06: plt.plot(x2, y)
# x 와 y 축, 각 축의 제목을 입력한다
07: plt.xlabel('Xn')
08: plt.ylabel('Y')
09: plt.title('Line plot using Xn and Y')
10: plt.show()
```

앞서 두 개의 차트는 오직 선으로만 구성된 것인데, 선의 색상/스타일(실선이나 점선), 마커(marker) 또는 연결점(선의 중간중간을 이어주는 기호) 등을 정할 수 있다. 다음은 자주 사용하는 선의 색상/스타일, 연결점을 정리한 것이다.

color	색상설명	linestyle	선종류	marker	연결점
R	Red	-	Solid line	+	+
G	Green	--	Dashed line	.	Dot
B	Blue	...	Dotted line	O	Circle
C	Cyan	-.-.-	Dash	*	*
M	Magenta	None	No line	P	Pentagon
Y	Yellow			S	Square
K	Black			X	X
W	White			D	Diamond
				H	Hexagon

				^	Triangle
--	--	--	--	---	----------

plot()함수에는 축의 값외에 marker, linestyle, color, label 매개변수를 사용하여 연결점(marker), 라인스타일, 라인색상, 레이블을 지정할 수 있다.

```
plt.plot(x, y, marker='o', linestyle='--', color='r', label='x')
```

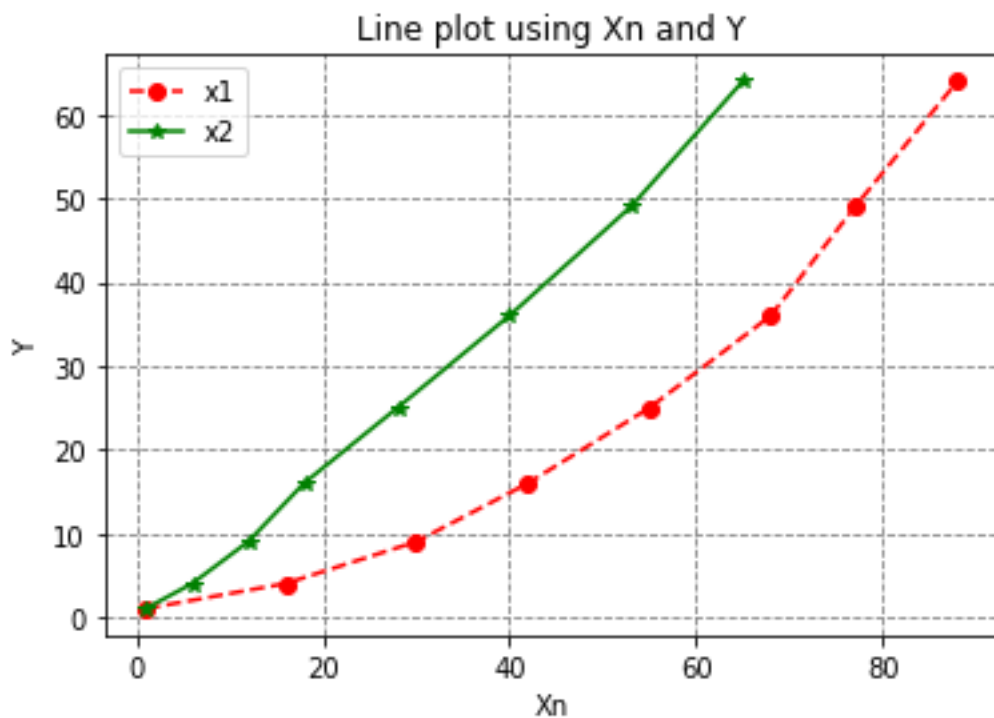
그리고 차트에 범례를 표시할 수 있는 데, plot()함수에 label 매개변수를 지정하고 legend()함수를 사용하여 범례의 위치(위/아래, 왼쪽/오른쪽)를 지정할 수 있다.

```
plt.legend( loc='upper left' )
```

그리고 차트에 가로세로 구분선을 추가할 수 있다.

```
plt.grid( True, color='gray', linestyle='--' )
```

그림 184



다음은 앞의 그림과 같이 선의 스타일/색상, 연결점, 범례, 구분선을 지정하여 차트를 그리는 코드 것이다.

<코드>

```

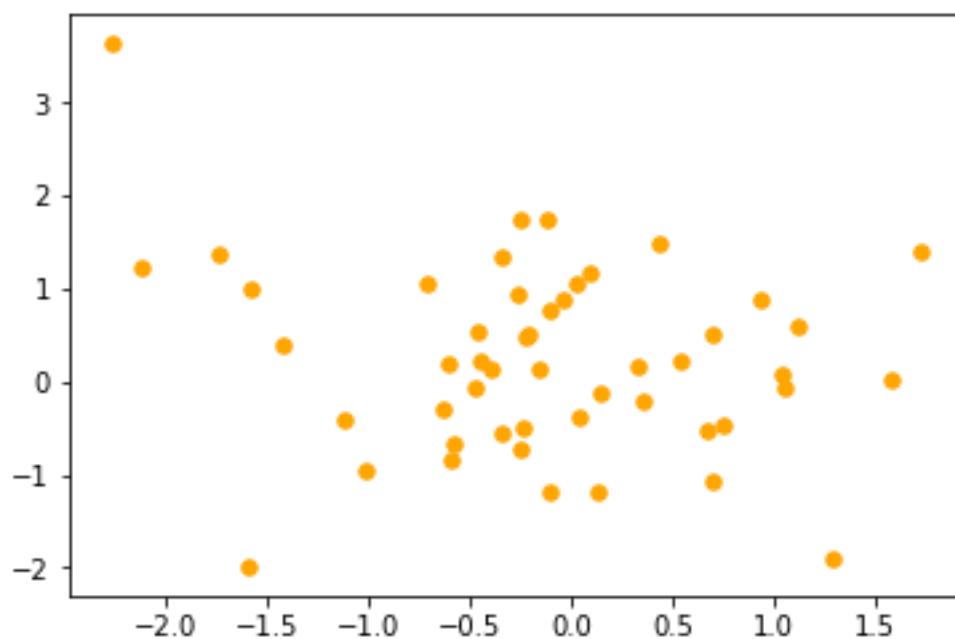
01: import matplotlib.pyplot as plt
02: y = [1, 4, 9, 16, 25, 36, 49, 64]
03: x1 = [1, 16, 30, 42, 55, 68, 77, 88]
04: x2 = [1, 6, 12, 18, 28, 40, 53, 65]
05: plt.plot(x1, y, marker='o', linestyle='--', color='r', label='x1')
06: plt.plot(x2, y, marker='*', linestyle='-', color='g', label='x2')
07: plt.xlabel('Xn')
08: plt.ylabel('Y')
09: plt.title('Line plot using Xn and Y')
10: plt.legend(loc='upper left')
11: plt.grid(True, color='gray', linestyle='--')
12: plt.show()
</코드>

```

9.2.3 분산형 차트

분산형 차트는 데이터의 집합이나 쌍을 비교하여 데이터간의 관계를 표시하는 차트이다. 다음은 50 개의 난수를 만들어 분산형 차트를 그린 것이다.

그림 185



분산형 차트는 scatter()함수를 그리며 점의 색상(color)이나 크기(s)등을 지정할 수 있다.

```
plt.scatter(x, y, color='orange', s=30)
```

<코드>

```
01: import matplotlib.pyplot as plt
```

```
02: import numpy as np
```

```
03: x = np.random.randn(1, 50)
```

```
04: y = np.random.randn(1, 50)
```

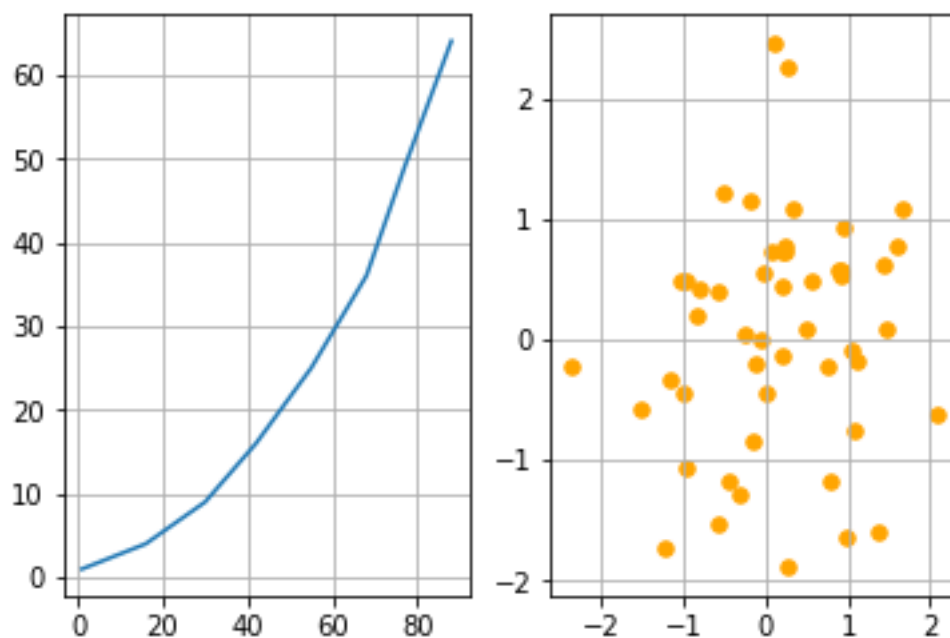
```
05: plt.scatter(x, y, color='orange', s=30)
```

```
06: plt.show()
```

</코드>

앞서에서는 모두 하나의 차트만 그려왔다. 이번에는 하나의 차트영역에 두 개의 차트를 그리는 방법을 알아본다.

그림 186



차트영역에 두 개이상의 차트를 그리려면 figure 라는 새로운 개체가 필요하다.

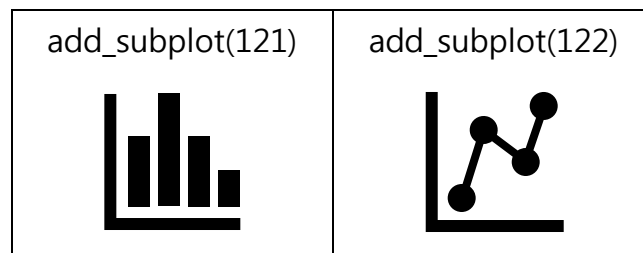
```
fig = plt.figure()
```


figure 개체는 `add_subplot()` 함수를 사용하여 그리려면 차트의 위치를 지정할 수 있다.

```
sp1 = fig.add_subplot(121)
sp2 = fig.add_subplot(122)
```

`add_subplot()` 함수에 들어간 숫자는 행과 열, 그림의 위치를 의미한다. 가령 121 은 차트영역은 1 행 2 열이며 1 번째 위치에 들어간다는 의미이다. 그리고 122 는 차트영역은 1 행 2 열이며 2 번째 위치에 들어간다는 의미이다.

그림 187



<코드>

```
01: import matplotlib.pyplot as plt
02: import numpy as np
# figure 개체를 생성한다
03: fig = plt.figure()
# 왼쪽 차트(라인 차트)을 그린다
04: sp1 = fig.add_subplot(121)
05: x = [1, 16, 30, 42, 55, 68, 77, 88]
06: y = [1, 4, 9, 16, 25, 36, 49, 64]
07: plt.plot(x, y)
08: sp1.grid(True)
# 오른쪽 차트(분산형 차트)를 그린다
09: sp2 = fig.add_subplot(122)
10: x = np.random.randn(1, 50)
11: y = np.random.randn(1, 50)
```

```
12: plt.scatter(x, y, color='orange', s=30)
```

```
13: sp2.grid(True)
```

```
14: plt.show()
```

</코드>

9.2.4 히스토그램

히스토그램은 빈도데이터를 보여주는 세로막대형 차트이다. 히스토그램은 hist()함수에 데이터는 넘겨주면 자동으로 그려준다. bins 매개변수에 계급의 수를 지정할 수 있다.

```
plt.hist(x, bins=20)
```

그림 188

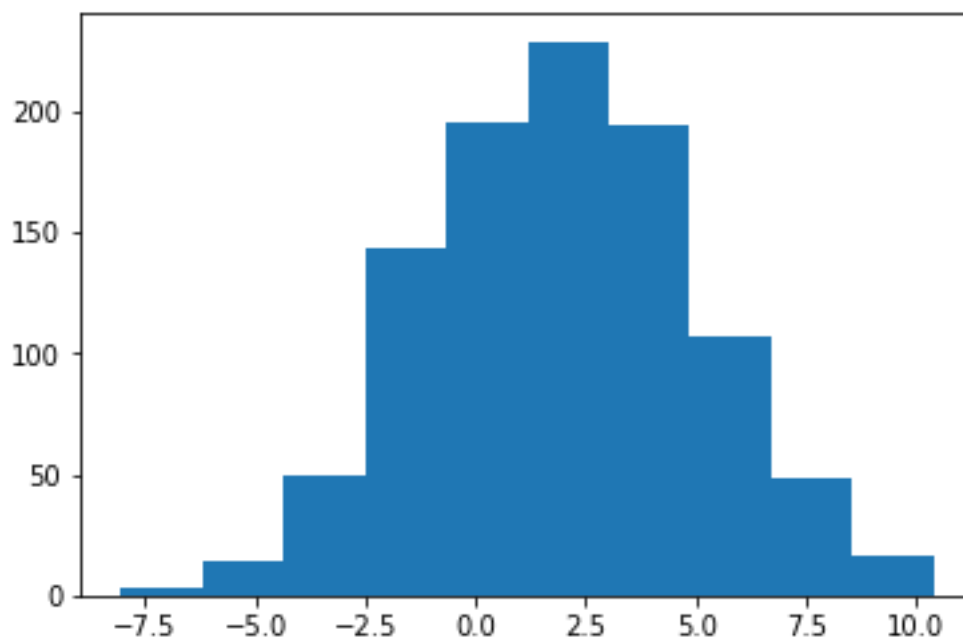
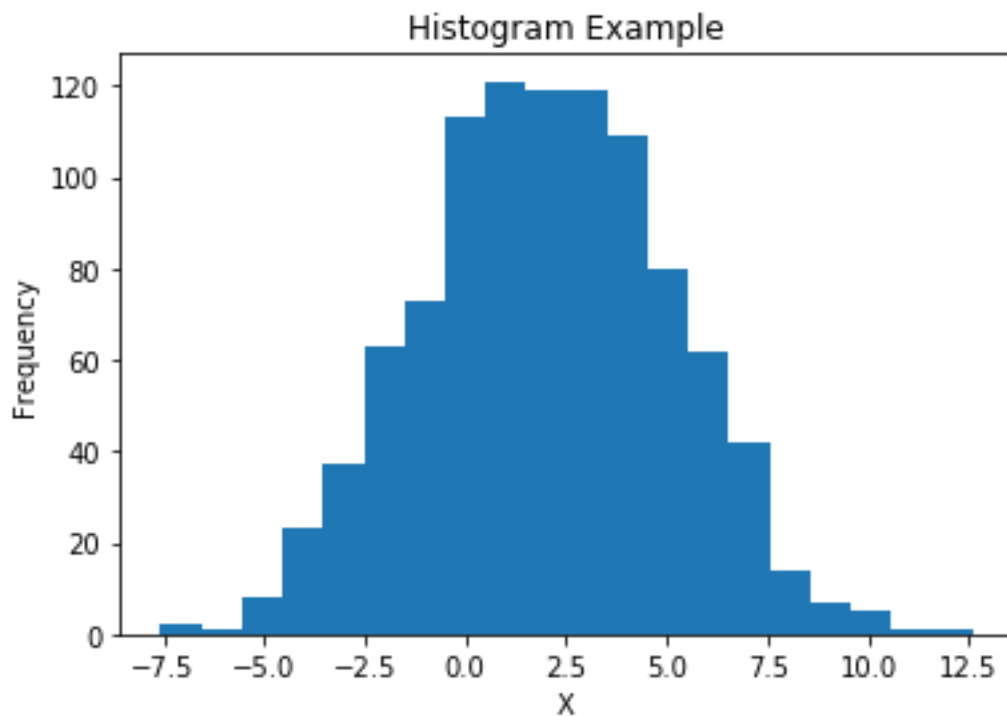


그림 189



<코드>

```
01: import matplotlib.pyplot as plt
02: import numpy as np
03: mean = 2.0
04: std = 3.0
05: nums = 1000
06: x = np.random.normal(mean, std, nums)
07: plt.hist(x)
08: plt.show()
```

</코드>

차트를 그리는 방법은 차트종류에 따라 사용하는 함수만 다르며, 기타 차트의 제목, 축의 레이블, 범례등은 공통으로 사용가능한 부분이다. 많은 독자들이 엑셀의 차트에 익숙하여 파이썬에서 차트를 그리는 것이 매우 불편하고 어렵게 느낄 수도 있을 것이다. 그러나 Matplotlib 는 파이썬이 현재와 같은 인기를 얻기 전부터 존재하였고 초창기에 차트를 구현하는 가장 좋은 방법이었다. 또한 차트를 그리는 방법 역시 엑셀에 비할 바는 아니지만 프로그래밍언어치곤 간단한

편이라 인기가 많기도 하다. 일단 숙달해지면 엑셀의 차트보다 빠르고 정밀한 차트를 만들 수 있을 것이다.

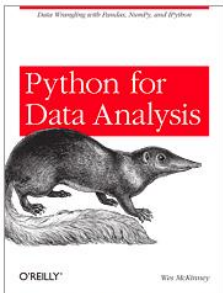
9.3 데이터 담당, Pandas

Pandas 는 미국의 프로그래머이자 사업가인 웨스 맥키니(Wes McKinney)가 개발하여 공개한 데이터분석을 위한 파이썬의 중요한 라이브러리이다. Pandas 는 마치 사무실의 엑셀과 같은 역할을 한다. 배열이나 행렬을 이용한 데이터 처리에 있어 Pandas 만만 편하고 강력한 라이브러리를 다른 언어에서도 찾아 보길 힘들다.

그림 190 웨스 맥키니(Wes McKinney) 홈페이지, <https://wesmckinney.com/>

Wes McKinney

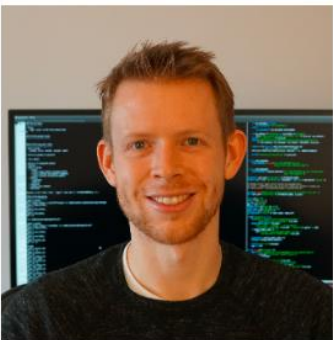
[?](#) About [🗣️](#) Talks [📖](#) Book [🐙](#) GitHub [🐦](#) Twitter [☰](#) Blog



I published the first edition in 2012, and the 2nd edition was published in 2017. [Click to read more](#)

Blog

I write about technical topics (mostly related to Python programming), with occasional diversions into other things that interest me.



About me

Learn more about my present past life and projects I've worked on.

Presentations

I give frequent presentations and tutorials about my work at Python and industry conferences and user meetups.

Code

Most of my open source software is hosted on GitHub.

웨스 맥키니(Wes McKinney)는 2007 년 MIT 수학과 학부 과정을 마친 뒤 코네티컷 주 그리니치의 AQR 캐피탈 매니지먼트에서 계량분석업무를 하였다. 그가 Pandas 를 개발하게 된 동기는 거추장스러운 데이터 분석 툴에 실망하였기 때문이라고 한다.

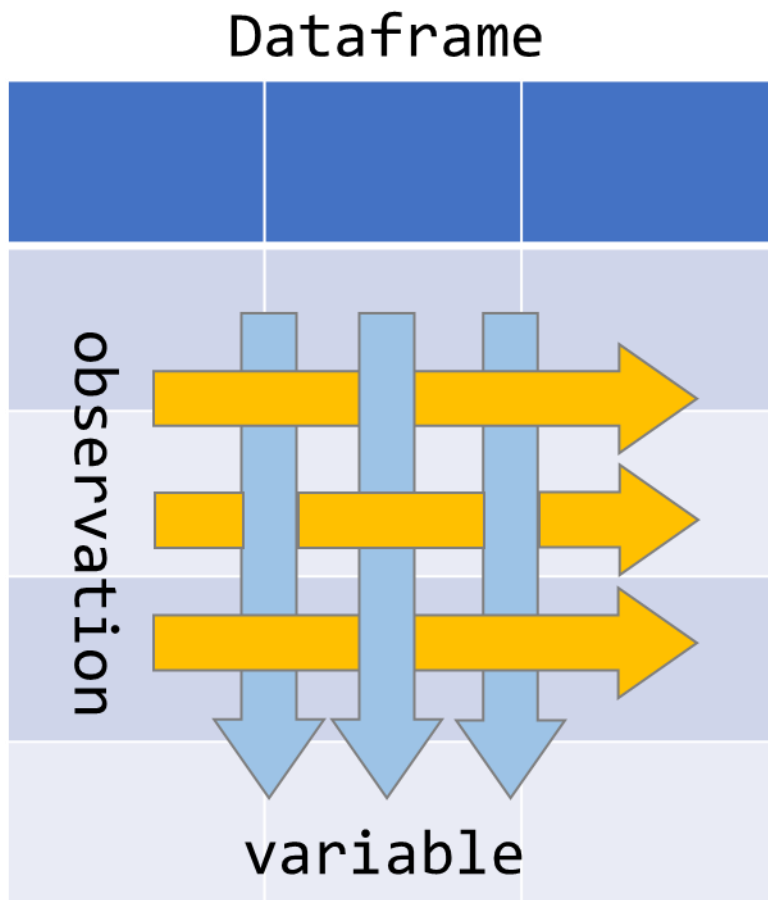
이후 웨스는 DataPad 를 창업하였고, DataPad 는 2014 년 Cloudera 에 인수되었다. 그 후 그는 아파치 소프트웨어 재단(Apache Software Foundation)의 아파치 애로우와 아파치 파켓 프로젝트를 위한 프로젝트 관리 위원회(Project Management Committees for the Apache Arrow and Apache Parquet projects)에 참여하면서 빅데이터 기술에 관여하게 되었다. 현재는 Python 과 R 을 위한 데이터사이언스툴 개발을 위한 비영리 단체인 Ursa Labs 의 이사다.

Pandas 의 백미는 데이터프레임(dataframe)인데, 데이터프레임을 중심으로 빠르게 알아보도록 한다.

9.3.1 데이터프레임

Pandas 의 기본은 데이터프레임(Dataframe)이다. 데이터프레임은 행과 열로 이루어진 행렬 또는 엑셀의 스프레드시트와 같은 것이다. 데이터프레임의 열을 variable(변수)라고 하고 , 행은 observation(관측치)라고 하며, 1+1 와 같은 스칼라 연산처럼 (벡터화)연산이 가능하다.

그림 191



Pandas 를 사용하려면 우선 Pandas 라이브러리를 다음과 같이 임포트한다

<코드>

```
import pandas as pd
```

</코드>

9.3.2 데이터프레임 만들기, DataFrame

데이터프레임을 직접 만들려면 DataFrame()함수를 이용하는 데, 다음과 같이 인덱스가 1,2,3 이고 컬럼(즉 variable)의 이름이 a, b, c 인 데이터프레임은 다음과 같이 만든다.

그림 192

	a	b	c
1	40	50	60
2	70	80	90
3	10	20	30

```
df1 = pd.DataFrame({  
    "a": [40, 50, 60],  
    "b": [70, 80, 90],  
    "c": [10, 20, 30]},  
    index=[1, 2, 3])  
print(df1)
```

또는 다음과 같이 컬럼명을 columns 매개변수에 전달하여 만든다

```
df2 = pd.DataFrame([  
    [41, 51, 61],  
    [71, 81, 91],  
    [11, 21, 31]],  
    index=[1, 2, 3],  
    columns=['a', 'b', 'c'])  
print(df2)
```

9.3.3 데이터프레임 합치기, concat 와 merge

앞서 만든 두 개의 데이터프레임 df1 과 df2 를 concat()함수를 사용하여 하나의 데이터프레임으로 합쳐 df3 를 만들 수 있다.

```
df3 = pd.concat([df1, df2])
```

그림 193 cat 함수

df1

	a	b	c
1	40	50	60
2	70	80	90
3	10	20	30

+

df2

	a	b	c
1	41	51	61
2	71	81	91
3	11	21	31

=

df3

	a	b	c
1	40	50	60
2	70	80	90
3	10	20	30
1	41	51	61
2	71	81	91
3	11	21	31

cat()함수는 두 개의 데이터프레임을 수직으로 세워 합치는 방식이지만 merge()함수는 데이터프레임을 수평으로 나란히 모아 합치는 함수이다.

```
df3 = pd.merge(df1, df2)
```

그림 194 merge 함수

df1				df2				df3								
	a	b	c		a	b	c		a	b	c	a	b	c		
1	40	50	60	+	1	41	51	61	=	1	40	50	60	41	51	61
2	70	80	90		2	71	81	91		2	70	80	90	71	81	91
3	10	20	30		3	11	21	31		3	10	20	30	11	21	31

9.3.4 인덱스 새로 만들기, reset_index

동일한 인덱스를 가진 df1 과 df2 를 합치다 보니 인덱스가 1, 2, 3, 1, 2, 3 반복되는 모습이다. 그래서 reset_index()함수를 사용하여 인덱스를 새로 매길 수 있다.

```
df3 = df3.reset_index()
print(df3)
```

인덱스를 새로 만들어 새 인덱스는 0,1,...,4,5 가 되었다. 그러나 기존의 인덱스는 사라지지 않고 index 라는 컬럼으로 바뀌었다.

	index	a	b	c
0	1	40	70	10
1	2	50	80	20
2	3	60	90	30
3	1	41	51	61
4	2	71	81	91
5	3	11	21	31

9.3.5 데이터프레임 컬럼삭제, drop

drop 함수를 사용하여 불필요한 컬럼을 삭제할 수 있다.

```
df3 = df3.drop(columns=['index'])
print(df3)
```

결과:

	a	b	c
0	40	70	10
1	50	80	20
2	60	90	30

```
3  41  51  61
4  71  81  91
5  11  21  31
```

9.3.6 컬럼을 행으로 모으기, melt

df3 데이터프레임은 3 개의 컬럼으로 구성되어 있는 데, 이번에는 melt()함수를 사용하여 (variable, value)의 형태로 분해하고 df4 를 만든다.

```
df4 = pd.melt(df3)
print(df4)
```

결과:

```
   variable  value
0         a     40
1         a     50
2         a     60
...      ...    ...
15        c     61
16        c     91
17        c     31
```

9.3.7 정렬하기, sort_values

앞서 만든 df4 의 컬럼중 value 컬럼을 (오름차순)정렬한다.

```
df4.sort_values('value')
```

반대로 내림차순으로 정렬하려면 ascending 매개변수를 False 로 지정한다

```
df4.sort_values('value', ascending=False)
```

9.3.8 쿼리하기, query

데이터프레임의 특정컬럼에 대한 쿼리를 수행할 수 있다. df4 의 value 컬럼에서 50 보다 큰 값을 가진 행만 가져오거나 출력하려면 query 함수에 조건을 지정한다.

```
df4.query('value>50') 또는 df4[df4.value>50]
```

결과:

	variable	value
2	a	60
4	a	71
6	b	70
7	b	80
8	b	90
9	b	51
10	b	81
15	c	61
16	c	91

쿼리한 결과를 다시 쿼리할 수 있는데, 함수의 결과에 바로 함수를 연결할 수 있다(이를 chaining 이라고 한다) 위의 결과에서 variable 인 b 인 결과를 얻고 싶다면 다음과 같이 할 수 있다.

```
df4.query('value>50').query('variable=="b"')
```

결과:

	variable	value
6	b	70
7	b	80
8	b	90
9	b	51
10	b	81

9.3.9 데이터프레임 컬럼명 바꾸기, rename

rename() 함수를 사용하여 컬럼의 이름을 바꿀 수 있다. 다음은 variable 을 var 로, value 를 val 로 바꾼다.

```
df4.rename(columns={'variable':'var', 'value':'val'})
```

결과:

	var	val
0	a	40
1	a	50
2	a	60
...

9.3.10 중복된 데이터 지우기, drop_duplicates

특정 컬럼속 데이터가 반복되어 있고, 중복되지 않는 데이터만 필요하다면 drop_duplicates 함수를 사용한다.

```
df4['variable'].drop_duplicates()
```

또는

```
df4.variable.drop_duplicates()
```

결과:

0	a
6	b
12	c

9.3.11 데이터프레임 앞부분, 뒷부분 살짝 보기, head, tail

head 와 tail 함수를 사용하여 데이터프레임의 앞부분이나 뒷부분을 필요한 만큼 볼 수 있다. 앞에서 5 개만 또는 뒤로부터 5 개 데이터만 보려면 다음과 같다.

```
df4.head(5)
```

결과:

	variable	value
0	a	40
1	a	50
2	a	60
3	a	41
4	a	71