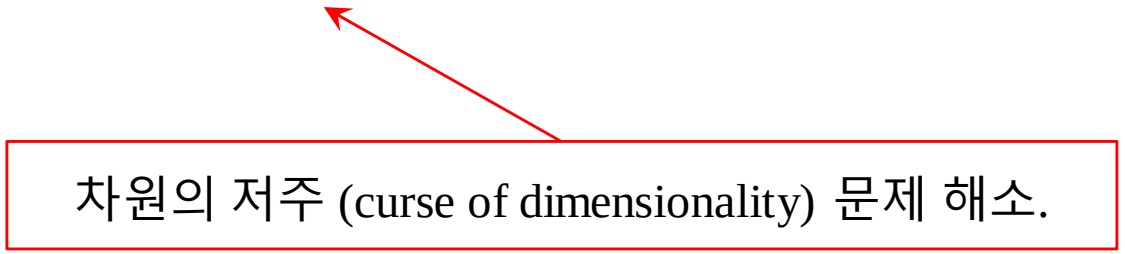
데이터 전처리

주성분 분석 : 목표

주성분 분석 (PCA)의 목표:

- 서로 상관관계가 있는 변수를 상관관계에서 자유로운 변수 (주성분)로 변환.
- 분산의 크기에 따라서 변수(주성분)를 정렬할 수 있습니다.
- 새롭게 만들어진 변수는 서로 직교(orthogonal) 관계입니다.
- 데이터 전처리, 모델링, 차원축소 등의 목적으로 사용됩니다.

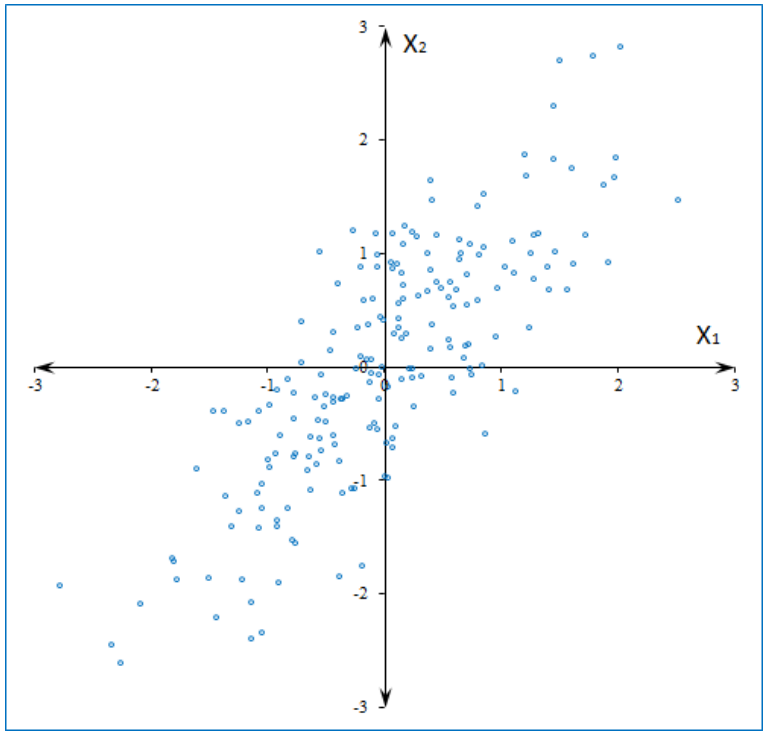


주성분 분석 : 결과

주성분 분석 (PCA)의 결과는 다음과 같습니다:

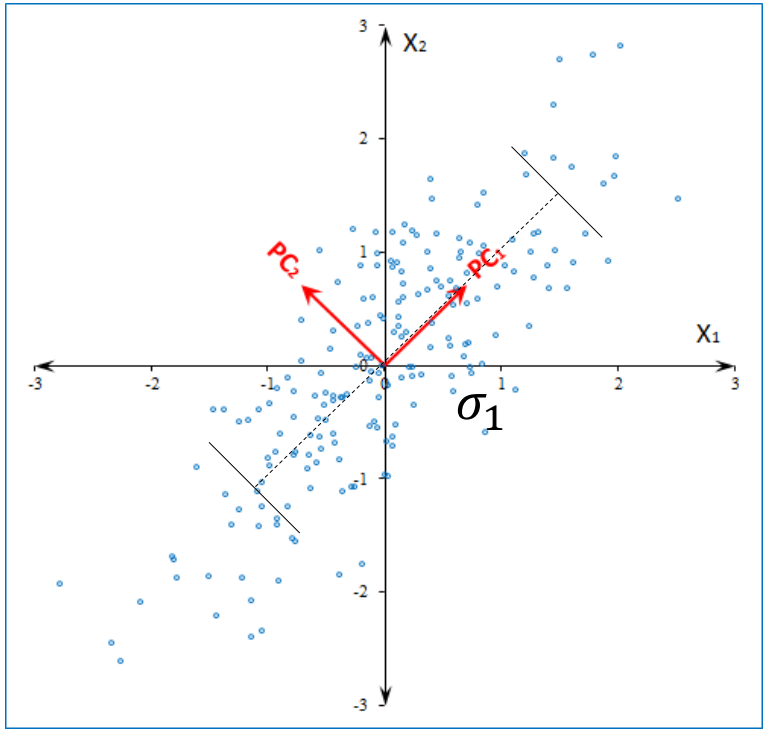
- Loading: 정규화된 주성분.
- Variance: 개개 주성분에 해당하는 분산 (~변동).
- Transformed scores: 주성분을 새로운 좌표축으로 사용하여 좌표를 투영방법으로 변환한 것.

주성분 분석 : 원리



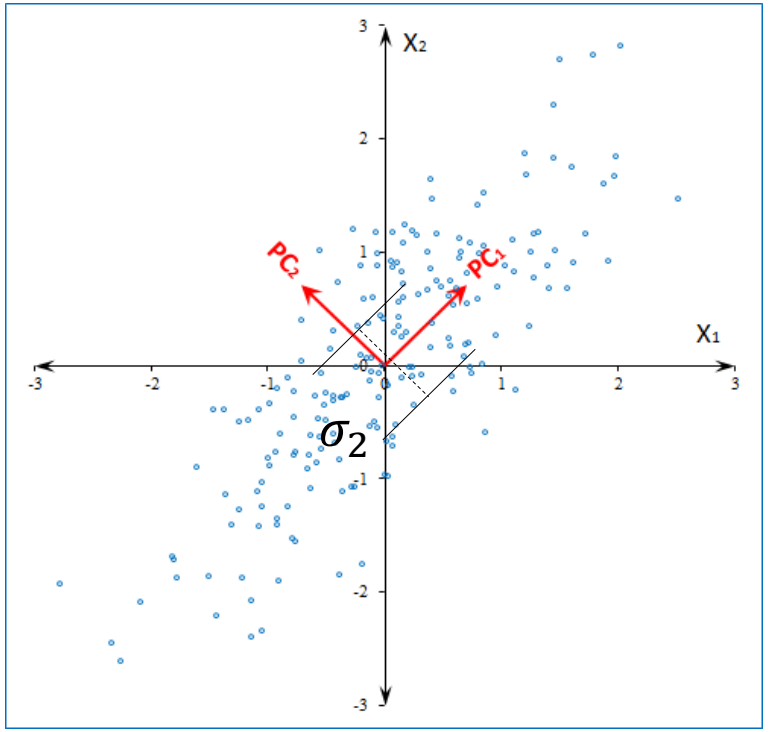
위와 같이 데이터 좌표가 분포되어 있다고 가정합니다.

주성분 분석 : 원리



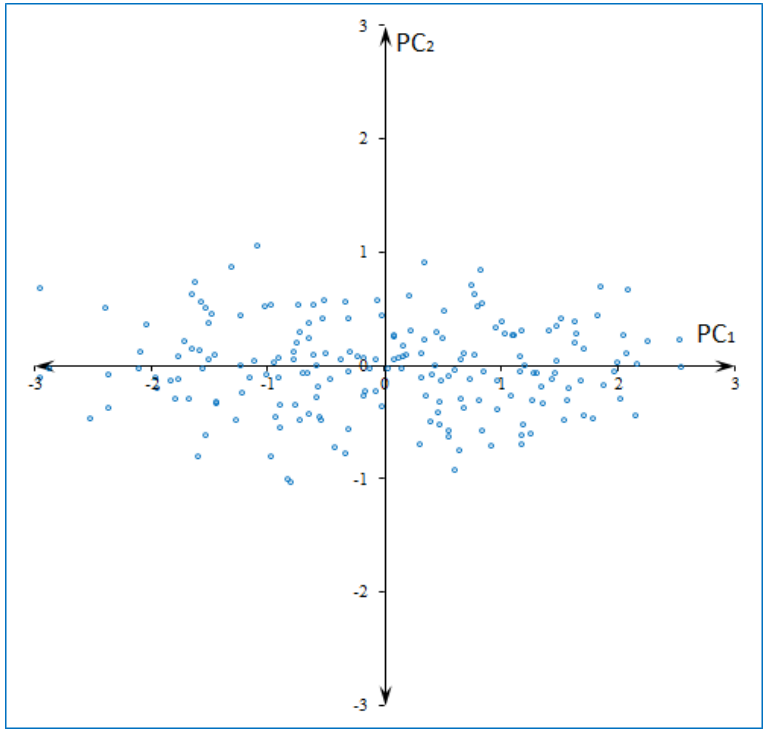
PC1은 가장 큰 변동의 방향을 향하고 있고, PC2는 PC1과 직교함.

주성분 분석 : 원리



PC1은 가장 큰 변동의 방향을 향하고 있고, PC2는 PC1과 직교함.

주성분 분석 : 원리



주성분(PC1 & PC2)을 새로운 좌표축으로 사용하였을 때.

주성분 분석 : 계산방법

주성분은 다음과 같은 방법으로 구할 수 있습니다:

- 데이터 행렬에 특이값 분해 (Singular Value Decomposition = SVD) 적용.
- 공분산 행렬 또는 상관계수 행렬의 고유값분해 (ED) 방법을 통해서 구할 수도 있습니다.
- 변수를 표준화하는 경우 이것은 상관계수행렬에 고유값분해를 적용하는 것과 같습니다.

주성분 분석 : 차원축소

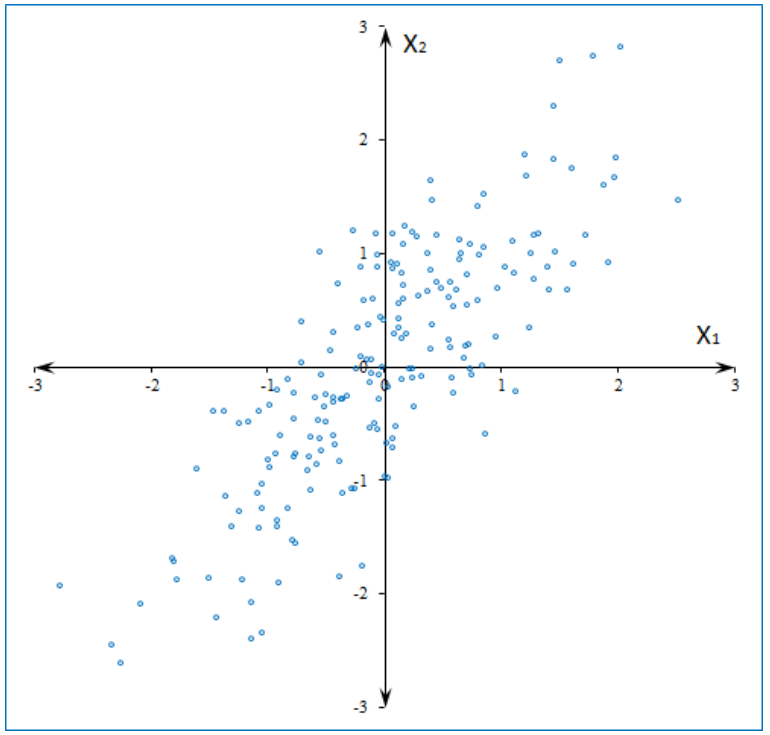
주성분 분석에서의 차원 축소 :

- 주성분을 새로운 좌표축으로 사용할 수 있습니다.
- 주성분의 개수는 원래 차원의 개수와 같습니다.
- 그런데 주성분은 분산의 크기 순서대로 정렬되어 있습니다.
- 그러므로, 분산이 작은 순서대로 차원을 축소해 나갈 수 있습니다.

주성분 분석 : 차원축소

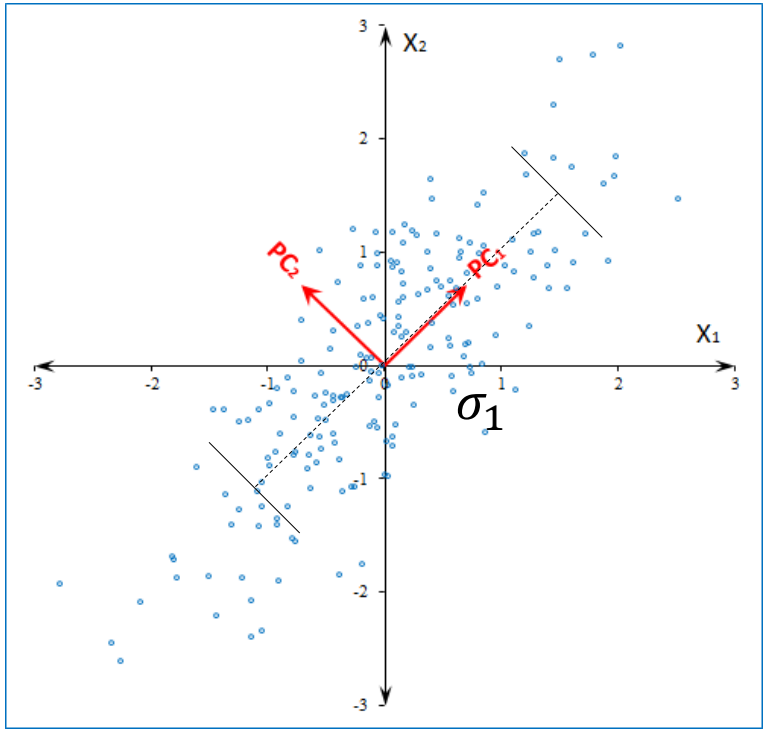
장점	단점
✓ 가장 뚜렷한 특징만 뽑아냄. ✓ 데이터 표현의 간소화. ✓ 연산, 메모리 부하 감소.	✓ 해석이 어려움. ✓ 작은 디테일 손상.

주성분 분석 : 차원축소의 원리



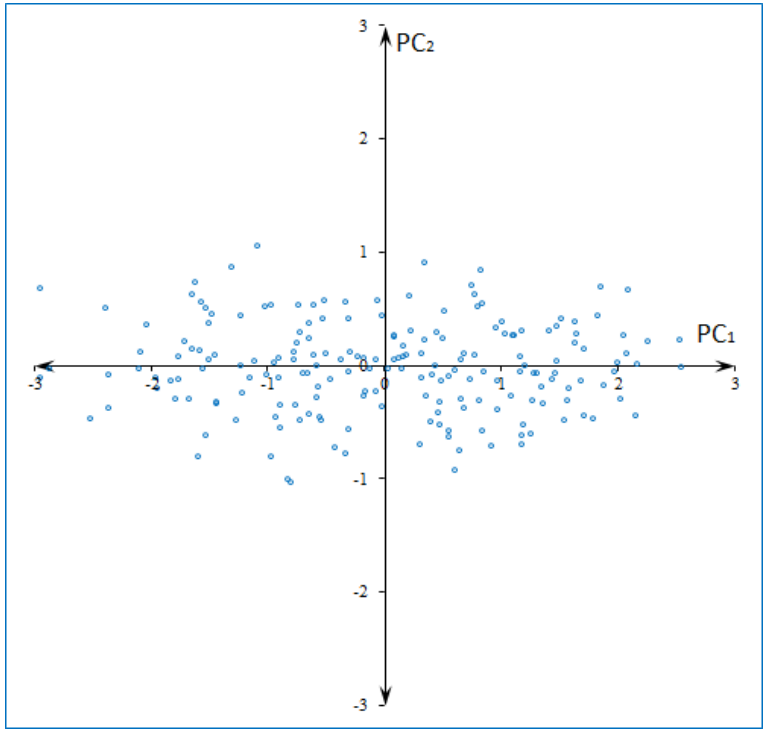
위와 같이 데이터 좌표가 분포되어 있다고 가정합니다.

주성분 분석 : 원리



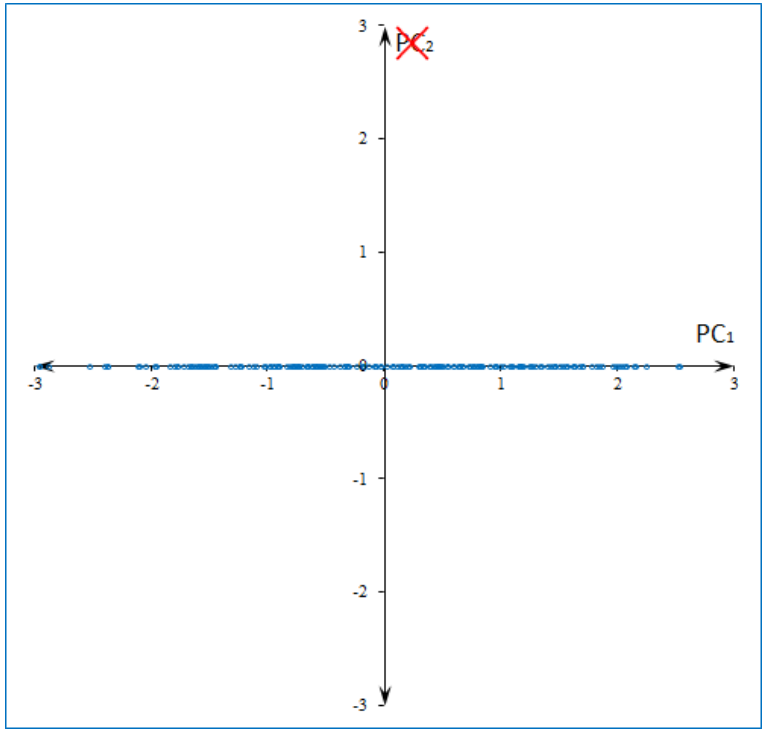
PC1은 가장 큰 변동의 방향을 향하고 있고, PC2는 PC1과 직교함.

주성분 분석 : 차원축소의 원리



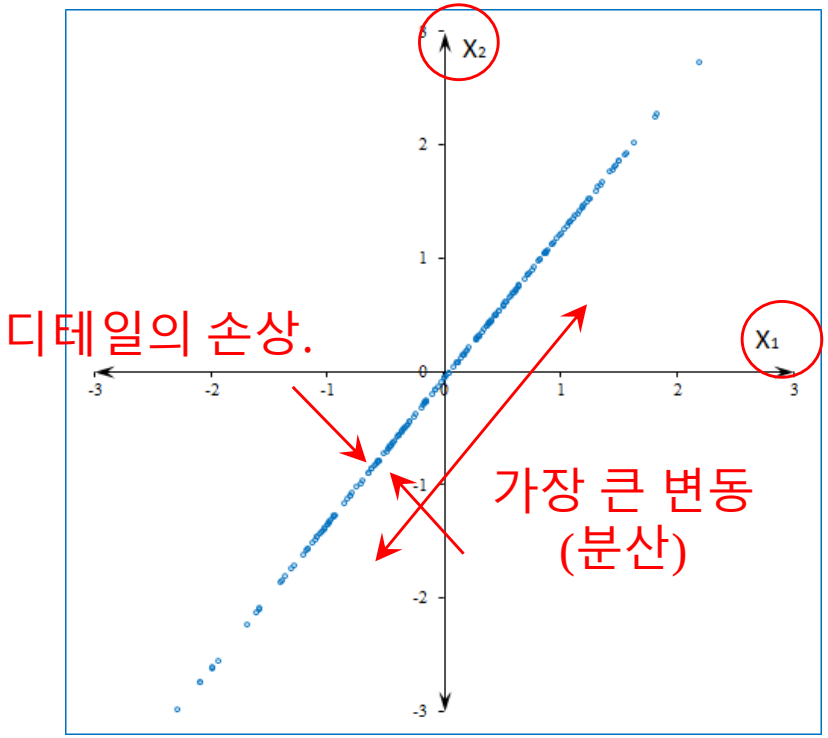
주성분(PC1 & PC2)을 새로운 좌표축으로 사용하였을 때.

주성분 분석 : 차원축소의 원리



상대적으로 작은 분산에 해당하는 PC2 방향으로 차원을 축소했습니다.

주성분 분석 : 차원축소의 원리



원 좌표축으로 돌아와 봅니다.

주성분 분석 : 차원축소 계산

차원 축소를 위해서는 다음과 같은 수치를 계산합니다:

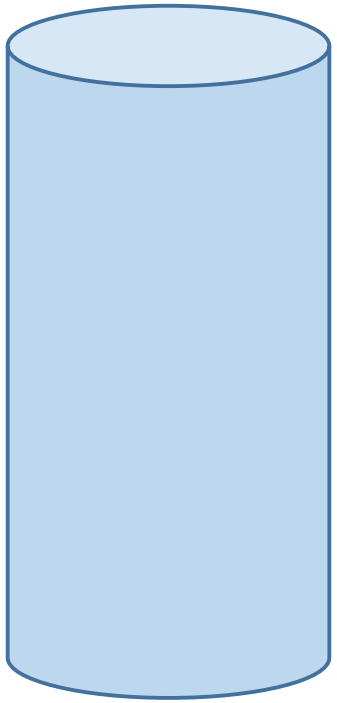
- Communality: 차원축소 후 개개 변수가 어느정도의 변동을 나타내는지 알려줌.
- Loadings: 정규화된 주성분.
- Reduced dimension input: 차원축소 후의 데이터 좌표.

주성분 분석 : 클러스터 시각화 응용

클러스터 시각화 응용 :

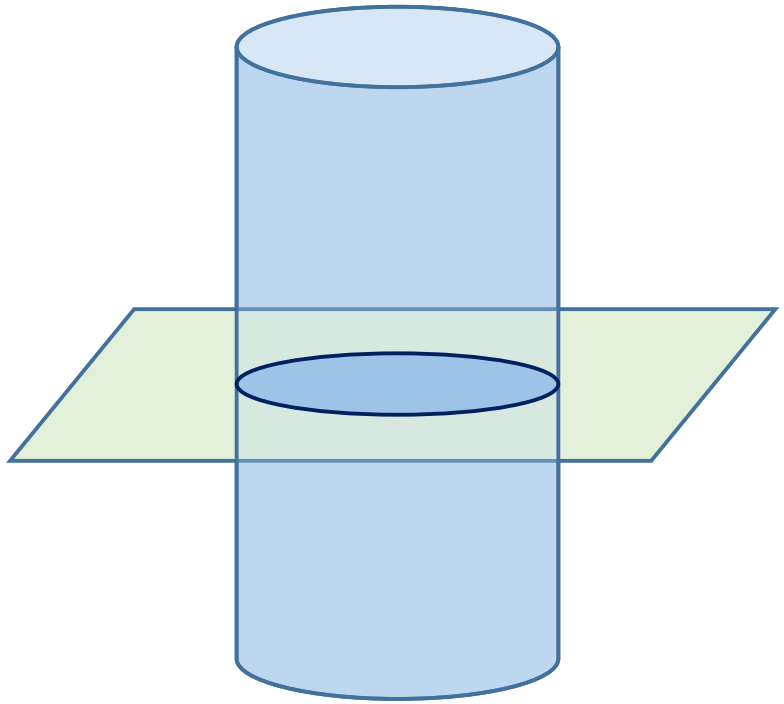
- 고차원 클러스터를 평면 (2D)에 투영하여 시각화하려고 합니다.
- 분산이 큰 순서대로 두개의 주성분 (PC1 & PC2)을 사용합니다.
- 이 두 주성분은 새로운 평면을 정의합니다.
- 고차원 데이터 좌표를 PC1과 PC1 평면상의 좌표로 투영합니다.

주성분 분석 : 클러스터 시각화 원리



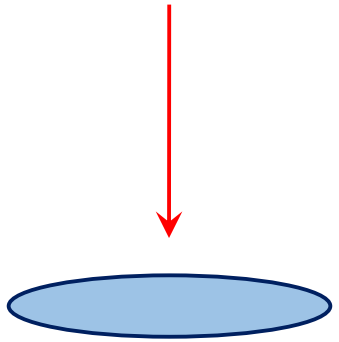
실린더가 있다고 가정해 봅니다.

주성분 분석 : 클러스터 시각화 원리



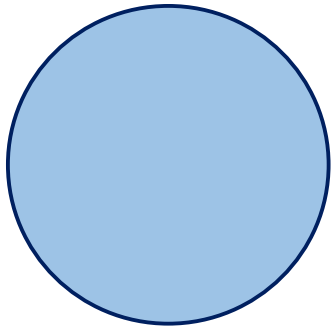
2D 단면을 위와 같이 자릅니다.

주성분 분석 : 클러스터 시각화 원리



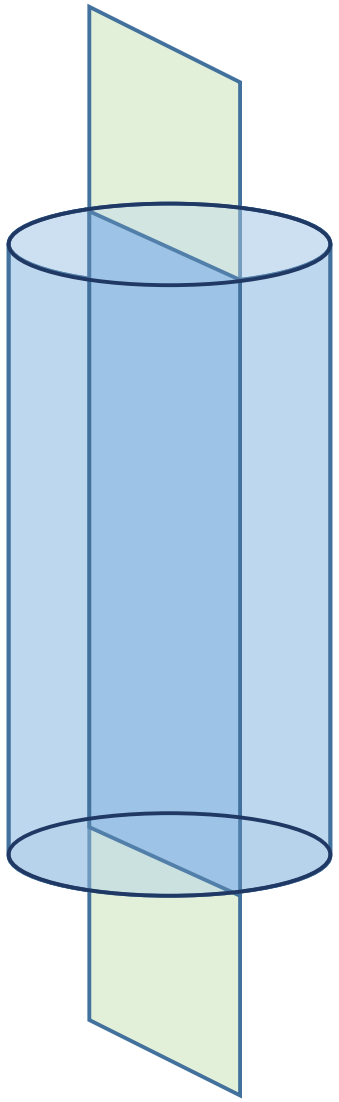
단면을 정면에서 봅니다.

주성분 분석 : 클러스터 시각화 원리



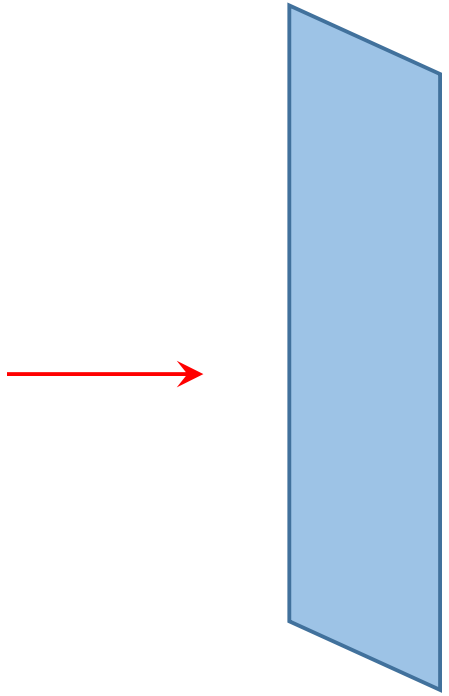
그러면 이렇게 보이겠죠.

주성분 분석 : 클러스터 시각화 원리



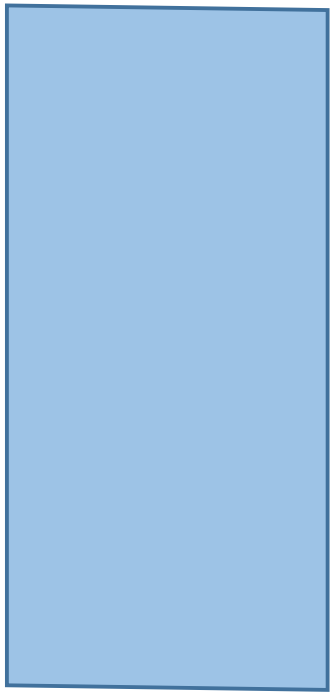
또다른 2D 단면을 그림과 같이 자릅니다.

주성분 분석 : 클러스터 시각화 원리



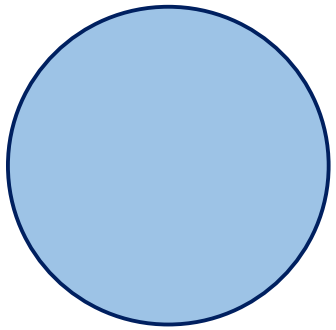
단면을 정면에서 봅니다.

주성분 분석 : 클러스터 시각화 원리

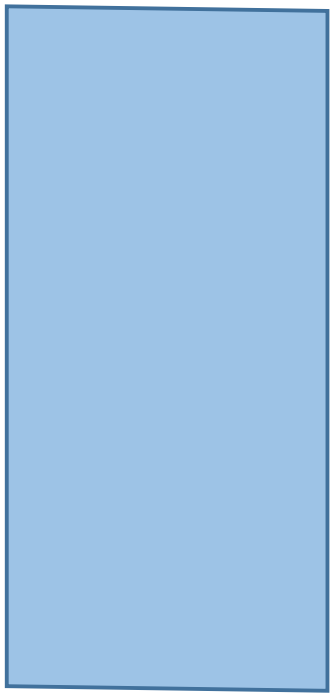


그러면 이렇게 보이겠죠.

주성분 분석 : 클러스터 시각화 원리

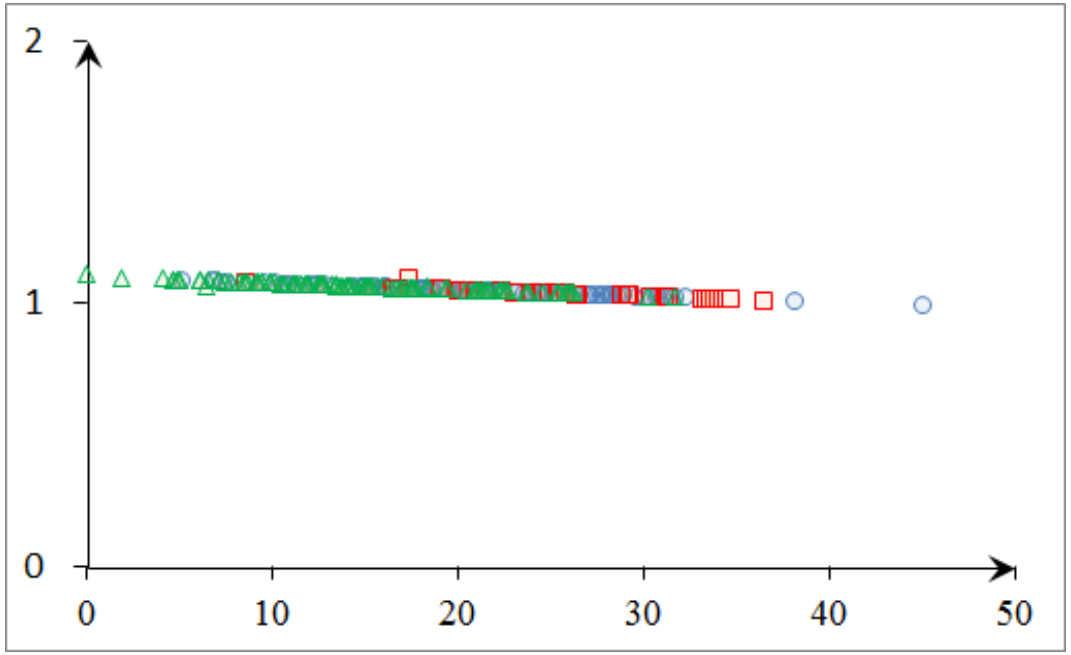


대



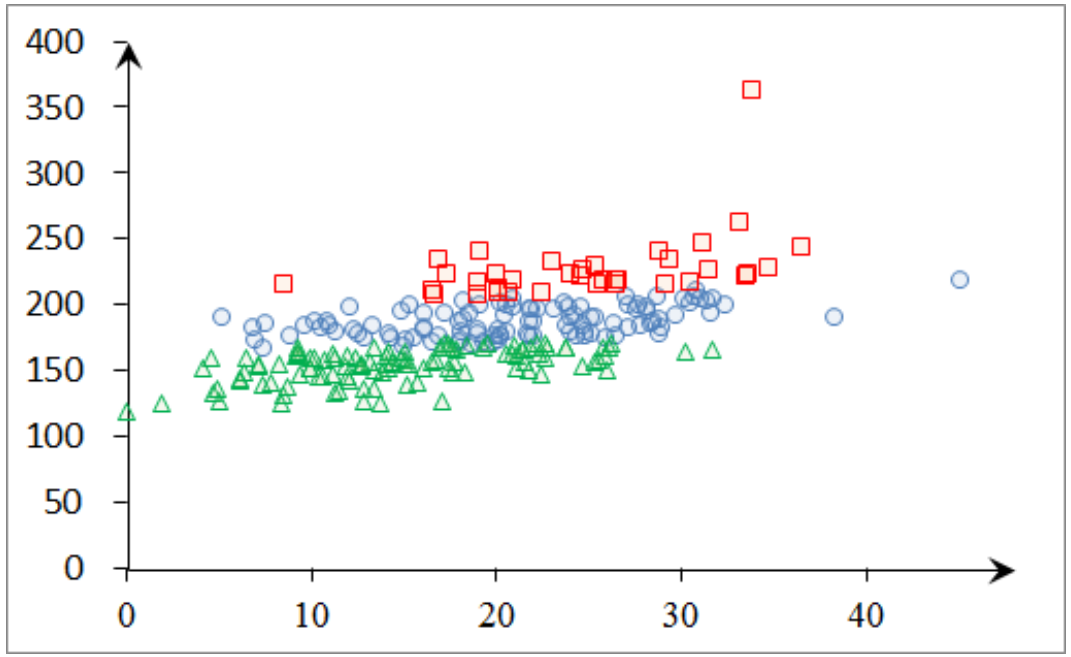
- ✓ 더 넓게 퍼져있는 평면.
- ✓ 투영하기에 좋음.

주성분 분석 : 클러스터 시각화 예



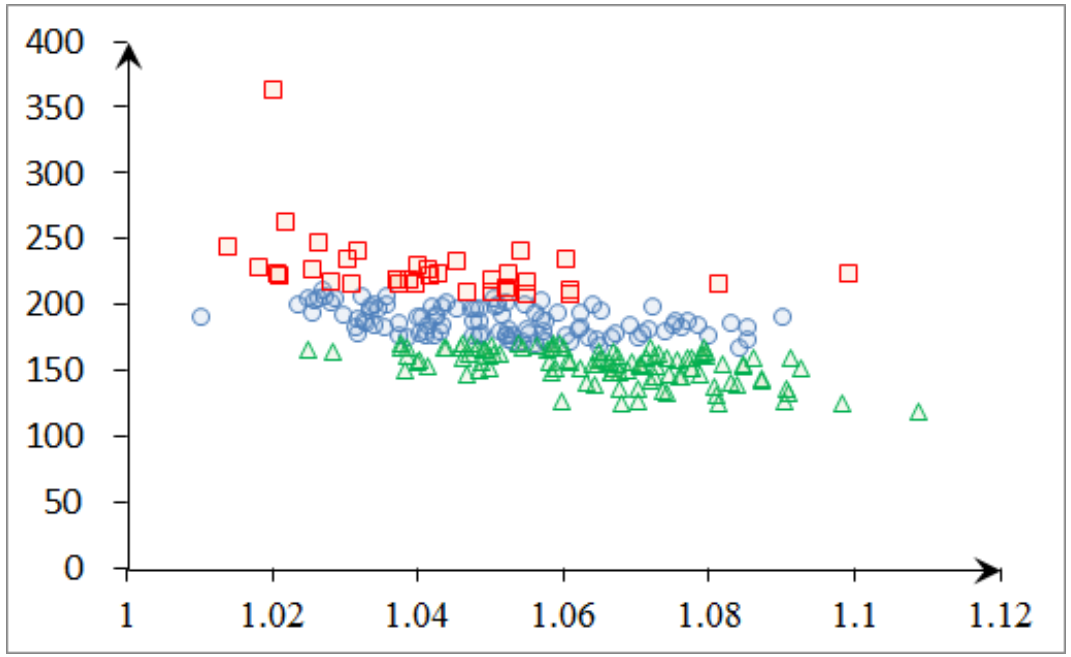
원 좌표 X1과 X2 그대로 사용.

주성분 분석 : 클러스터 시각화 예



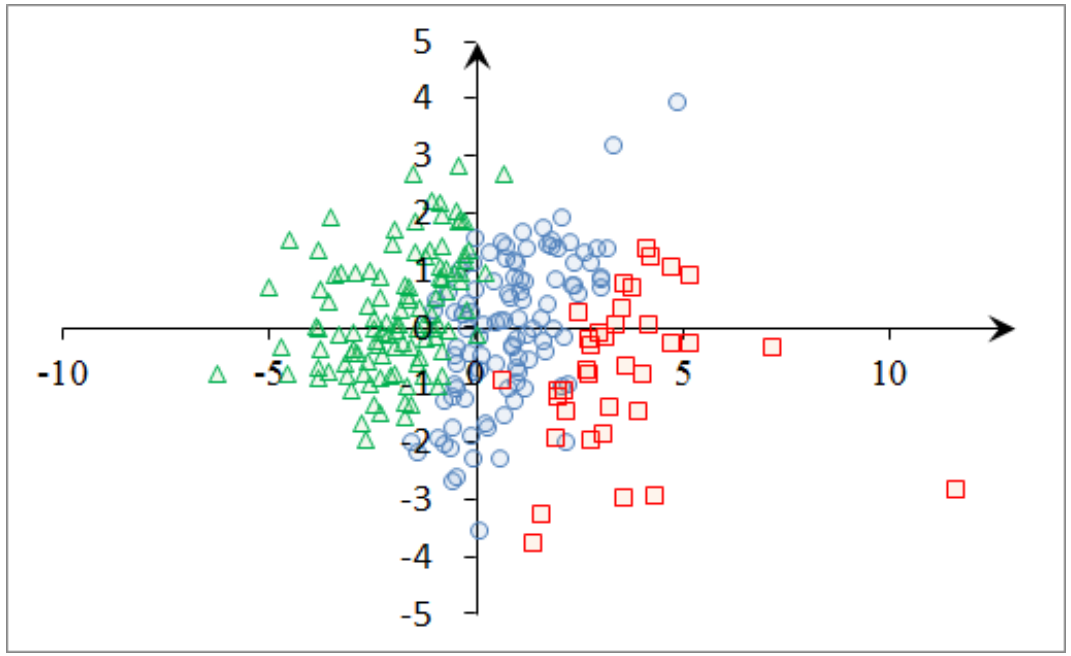
원 좌표 X1과 X3 그대로 사용.

주성분 분석 : 클러스터 시각화 예



원 좌표 X2와 X3 그대로 사용.

주성분 분석 : 클러스터 시각화 예



PC1과 PC2로 정의되는 단면에 투영.

유용한 함수

문자열 관련 함수:

함수	역할
x.lstrip()	왼쪽 공백 지우기.
x.rstrip()	오른쪽 공백 지우기.
x.strip()	양쪽 공백 지우기.
x.replace(str1, str2)	문자열 바꾸기 (str1 → str2).
x.count(str)	문자 (문자열) 개수 세기.
x.find(str)	위치 알려주기. (-1)
x.index(str)	위치 알려주기. (오류)
a.join(str_list)	a 삽입 문자열 연결.
x.split(a)	문자열을 a로 토막냄.
x.upper()	대문자로 변환.
x.lower()	소문자로 변환.
len(x)	문자열의 길이.

정규 표현식

정규 표현식 (regular expression)이란?

- 문자열 패턴을 만드는데 사용.
- 복잡한 문자열을 처리할 때 필요함.
- Python 뿐만이 아니라 많은 프로그래밍 언어에서 지원.
- 일반 문자열 함수의 조합 보다 함축적이며 강력한 기능.

텍스트 마이닝 : 정규 표현식

메타 문자 (meta character):

- 메타문자는 원래 그 문자가 가진 뜻이 아닌 특별한 용도로 사용되는 문자이다.

. ^ $ * + ? { } [ ] \ | ( )

- 정규 표현식에 위 메타 문자들이 사용되면 특별한 의미를 갖게 된다.
- 그럼 가장 간단한 정규 표현식 부터 시작해서 메타 문자의 의미와 사용법을 알아보자.

텍스트 마이닝 : 정규 표현식

메타 문자 (meta character):

- 특별한 의미로 사용되는 문자: . ^ $ * + ? { } [ ] \ | ( )
- ^ : 시작 일치. ⇒ ^i
- $: 끝 일치. ⇒ know$
- [ ] : 문자 클래스. ⇒ ^[Ii] , ^[0-9]
- [^] : Not 문자 클래스. ⇒ [^0-9]$
- . : 도트. 아무 문자. ⇒ a..b
- * : 0회 이상 반복. ⇒ ca*t

텍스트 마이닝 : 정규 표현식

메타 문자 (meta character):

- + : 1회 이상 반복. ⇒ ca+t
- {m} : m회 반복. ⇒ ca{2}t
- {m,n} : m~n회 반복. ⇒ ca{1,3}t
- ? : {0,1}의 의미. ⇒ ca?t
- | : or. ⇒ flood|fire

정규표현식

메타문자: 문자 클래스 [ ]

- “[와] 사이의 문자들과 매치”라는 의미를 갖는다.
- 문자 클래스를 만드는 메타 문자인 [와] 사이에는 어떤 문자도 들어갈 수 있다.
- [abc]라는 정규표현식의 의미는 “a, b, c 중 한 개의 문자와 매치”이다.

정규식	문자열	Match 여부	설명
[abc]	a	Yes	정규식과 일치하는 문자인 “a”가 있으므로 매치.
[abc]	before	Yes	정규식과 일치하는 문자인 “b”가 있으므로 매치.
[abc]	dude	No	“a”, “b”, “c” 중 어느 하나도 포함하고 있지 않음.

정규표현식

메타문자: 문자 클래스 [ ]

- [ ] 안의 두 문자 사이에 하이픈(-)을 사용하게 되면 두 문자 사이의 범위를 의미한다.

  예). [a-c]라는 정규 표현식은 [abc]와 동일하다.

  예). [0-5]는 [012345]와 동일하다.

  예). [a-zA-Z] : 알파벳 모두.

  예). [0-9] : 숫자.

- 또한 다음과 같은 **단축 표현**이 있다.

  ⇒ **w** = [a-zA-Z0-9_]

  ⇒ **W** = [^a-zA-Z0-9_]

  ⇒ **d** = [0-9]

  ⇒ **D** = [^0-9]

정규표현식

메타문자: NOT 문자 클래스 [^]

- “[^와] 사이의 문자들과 매치하지 않음”을 의미한다.
- [^abc]라는 정규표현식의 적용예:

정규식	문자열	Match 여부	설명
[^abc]	a	No	“a”가 “abc”에 포함됨.
[^abc]	before	Yes	“b”를 제외한 나머지는 “abc”에 포함되지 않음.
[^abc]	dude	Yes	“a”, “b”, “c” 중 어느 하나도 포함하고 있지 않음.

정규표현식

메타문자: 점 “.”

- 정규 표현식의 점 “.” 메타문자는 모든 문자와 매치됨.
- 정규표현식 a.b의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
a.b	aab	Yes	가운데 문자 “a”가 모든 문자를 의미하는 .과 일치.
a.b	a0b	Yes	가운데 문자 “0”가 모든 문자를 의미하는 .과 일치.
a.b	abc	No	“a”문자와 “b”문자 사이에 어떤 문자라도 없음.

- 만약 앞에서 살펴본 문자 클래스 [ ] 안에 점 “.” 메타문자가 사용된다면 이것은 “모든 문자”라는 의미가 아닌 문자 “.” 그대로를 의미한다. 혼동하지 않도록 주의하자.

정규표현식

메타문자: 반복 “*”

- “*”의 의미는 바로 앞에 있는 문자가 0부터 무한대로 반복될 수 있다는 의미이다.
- 정규표현식 ca*t의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
ca*t	ct	Yes	“a”가 0번 반복되어 매치.
ca*t	cat	Yes	“a”가 0번 이상 반복되어 매치 (한번 반복).
ca*t	caaat	Yes	“a”가 0번 이상 반복되어 매치 (세번 반복).

정규표현식

메타문자: 반복 “+”

- “+”의 의미는 바로 앞에 있는 문자가 한번부터 무한대로 반복될 수 있다는 의미이다.
- 정규표현식 ca+t의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
ca+t	ct	No	“a”가 0번 반복되어 매치 아님.
ca+t	cat	Yes	“a”가 한번 이상 반복되어 매치 (한번 반복).
ca+t	caaat	Yes	“a”가 한번 이상 반복되어 매치 (세번 반복).

정규표현식

메타문자: 반복 “?”

- “?”의 의미는 바로 앞에 있는 문자가 0회 또는 1회 있다는 의미.
- 정규표현식 ca?t의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
ca?t	ct	Yes	“a”가 0번 반복됨. 매치 됨!
ca?t	cat	Yes	“a”가 한번 반복됨. 매치 됨!
ca?t	caat	No	“a”가 두번 반복됨. 매치 아님.

정규표현식

메타문자: 반복 {m}

- {m}의 의미는 바로 앞에 있는 문자가 정확하게 m번 반복된다는 의미이다.
- 정규표현식 ca{2}t의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
ca{2}t	ct	No	“a”가 0번 반복됨.
ca{2}t	cat	No	“a”가 한번 반복됨.
ca{2}t	caat	Yes	“a”가 정확하게 두번 반복됨. 매치!

정규표현식

메타문자: 반복 {m,n}

- {m,n}의 의미는 바로 앞에 있는 문자가 m~n회 반복된다는 의미.
- 정규표현식 ca{2,5}t의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
ca{2,5}t	cat	No	“a”가 1번 반복됨. 매치 아님.
ca{2,5}t	caaat	Yes	“a”가 3번 반복됨. 매치 됨!
ca{2,5}t	caaaaaat	No	“a”가 6번 반복됨. 매치 아님.

정규표현식

메타문자: 시작 일치 ^

- 메타문자 ^는 문자열의 맨 처음과 일치함을 의미한다.
- 정규표현식 ^Life의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
^Life	Life is short.	Yes	문장의 첫 단어가 Life 이다.
^Life	My Life is boring.	No	문장의 첫 단어가 Life 아니다.

정규표현식

메타문자: 끝 일치 $

- 메타문자 $는 문자열의 맨 마지막과 일치함을 의미한다.
- 정규표현식 Python$의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
Python$	Python is easy	No	문장의 마지막이 Python 아니다.
Python$	You need Python	Yes	문장의 마지막이 Python 이다.

정규표현식

메타문자: OR의 의미 |

- 메타문자 |는 OR의 의미이다.
- 정규표현식 love|hate의 적용예를 알아보자.

정규식	문자열	Match 여부	설명
love|hate	I love you	Yes	문자열에 love 포함됨.
love|hate	I hate him	Yes	문자열에 hate 포함됨.
love|hate	I like you	No	문자열에 love나 hate가 없음.

정규표현식

grep()함수와의 사용:

```
grep("^This",book)                # book은 문자열 list.
grep("end[.]$",book)
grep("^[Ii]",book)
grep("^[0-9]",book)
grep("b.d",book)                  # 아무런 문자.
grep("b...d",book)                # 아무런 문자 2회.
```

정규표현식

grep()함수와의 사용:

```
grep("a*d",book)                          # 'a' 0회 이상 반복.
grep("a+d",book)                          # 'a' 1회 이상 반복.
grep("r{2}d",book)                        # 'r' 2회 반복.
grep("r{1,2}d",book)                      # 'r' 1~2회 반복.
grep("r?ed",book)                         # 'r' 0~1회.
grep("love|hate",book)                    # 'love' or 'hate'.
len(grep("love|hate",book))/len(book)             # 'love' or 'hate'의 비율.
```

정규표현식

그룹 매칭:

```
my_regex  = re.compile( "([0-9]+)[^0-9]+([0-9]+)" )
m = reg_ex.search("I am 15 yers old. And, you are 12 years old.")
print(m.group(0))                    # 전체 매치.
print(m.group(1))                    # 첫 번째 그룹.
print(m.group(2))                     # 두 번째 그룹.
```

정규표현식

그룹 매칭: 단축 표현 사용.

```
reg_ex = re.compile("(\d+)\D+(\d+)")
m = reg_ex.search("I am 15 yers old. And, you are 12 years old.")
print(m.group(0))                    # 전체 매치.
print(m.group(1))                    # 첫 번째 그룹.
print(m.group(2))                     # 두 번째 그룹.
```

정규표현식

그룹 매칭: 응용 #1.

```
# 전화번호 별표처리.
reg_ex = re.compile("(\D+)(\d+)\D+(\d+)\D+(\d+)")
m = reg_ex.search("홍길동 010-1234-5678")
print((m.group(1)).strip() + " " + m.group(2) + "-****-****")
```

정규표현식

그룹 매칭: 응용 #2.

```
# 익명처리.
reg_ex = re.compile("(\D+)((\d+)\D+(\d+)\D+(\d+))")
m = reg_ex.search("홍길동 010-1234-5678")
print("전화번호 : " + m.group(2))                # 두번째 그룹이 전화번호 전체를 포함.
```