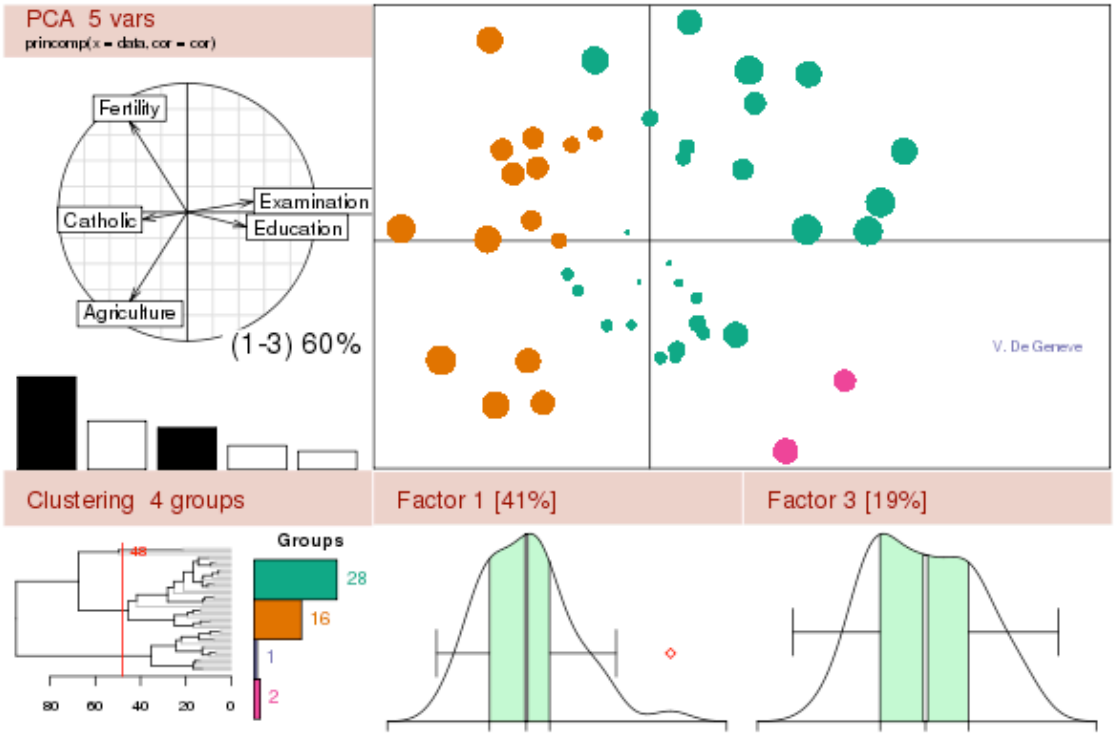
벡터에 대해서

Base Data Objects

유 충현
블로그 모음 9탄(http://blog.naver.com/bdboys) • (주)오픈베이스 • 2012년 10월 3일



벡터에 대해서

R은 자료의 변환 및 가공이 무척이나 수월하다. 통계 계산을 목적으로 설계된 언어이기 때문에 새삼 놀랄 일은 아니다. 그러나 R이외의 다른 언어를 사용 경험이 있는 사람들에게는 획기적인 기능들우 많이 있다. 이번에는 벡터를 통해서R의 자료조작의 방법에 대해서 감을 잡아보자.

통계학에서의 열벡터, 행벡터는 Vector라는 자료 형으로 처리하고, 행렬은 Matrix라는 자료형을 이용해서 처리한다. 물론 사전적인 의미도 같다.

벡터가 행렬의 한 종류라는 것을 알고 있을 것이다. m by n 차원을 갖는 것이 행렬이라면 열벡터는 m by 1의 차원을, 행벡터는 1 by n의 차원을 갖는 행렬이라고 할 수 있다.

다음 예를 보자.

```
> matrix(1:6,ncol=3)   # 2 by 3 행렬
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> dim(matrix(1:6,ncol=3)) # 행렬의 차원
[1] 2 3

> matrix(1:6,ncol=1) # 6 by 1 행렬,  열벡터
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6

> dim(matrix(1:6,ncol=1)) # 열벡터의 차원
[1] 6 1

> matrix(1:6,nrow=1) # 1 by 6 행렬, 행벡터
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
```

> dim(matrix(1:6,nrow=1)) # 행벡터의 차원
[1] 1 6

이것이 수학 세계에서의 행렬과 벡터의 관계일 것이다. 그러나 R에서는 조금 이야기가 다르다. 개념상의 포함관계는 동일하지만 자료를 저장하는 자료형의 입장에서는 조금 차이가 있다. matrix함수를 이용해서 만든 자료는 is.vector의 함수에서 FALSE의 결과로 나타난다. Vector가 아니란다.

> is.vector(matrix(1:6,nrow=1))
[1] FALSE
> is.vector(matrix(1:6,ncol=1))
[1] FALSE
> as.vector(matrix(1:6,nrow=1))
[1] 1 2 3 4 5 6
> as.vector(matrix(1:6,ncol=1))
[1] 1 2 3 4 5 6

그리고 자료형에는 열벡터, 행벡터가 없고 그저 벡터뿐이다. 즉, 자료형의 관점에서 벡터는 하나 이상의 동일한 데이터 타입을 갖는 객체를 의미한다.

> is.vector(1)
[1] TRUE
> is.vector("a")
[1] TRUE

다음은 자주 사용하는 벡터의 가공에 대해서 살펴본다.

10개의 원소를 갖는 x라는 정수형 벡터를 만들어 보자.

> x=1:10
> x
 [1] 1 2 3 4 5 6 7 8 9 10

x 벡터의 세번째 원소의 값은 무엇인가?
> x[3]
[1] 3

x 벡터의 첫번째부터 열번째 까지 원소의 값은 무엇인가?
> x[1:10]
 [1] 1 2 3 4 5 6 7 8 9 10

x 벡터의 세번째부터 일곱번째 까지 원소의 값은 무엇인가?

```
> x[3:7]
[1] 3 4 5 6 7
```

x 벡터에서 다섯번째 원소를 제거해보자.

```
> x[-5]
[1]  1  2  3  4  6  7  8  9 10
```

x 벡터에서 세번째부터 일곱번째 까지 원소를 제거해보자.

```
> x[-(3:7)]
[1]  1  2  8  9 10
```

x 벡터에서 원소의 값이 5보다 큰 건을 골라 보자.

```
> x[x>5]
[1]  6  7  8  9 10
```

x 벡터에서 원소의 값에 대해서 "5보다 크다"라는 조건을 만족하는 지 검증해보자.

```
> x>5
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
TRUE
```

x 벡터에서 원소의 값에 대해서 "5보다 크다"라는 조건을 만족하는 지 검증결과를 y 라는 이름의 벡터롤 만들어 보자. 이 벡터는 논리값을 갖는 벡터이다.

```
> y=x>5
> y
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
TRUE
```

x 벡터에서 원소의 값에 대해서 "5보다 크다"라는 조건을 만족하는 원소를 골라 보자.

```
> x[y]
[1]  6  7  8  9 10
```

x 벡터에서 원소의 값에 대해서 짝수번째 원소만 골라보자.

```
> x[c(FALSE,TRUE)]
[1]  2  4  6  8 10
```

사실 [] 연산자 안에는 벡터 x와 크기가 같은 논리형 벡터를 사용해야 한다.
하지만 모자랄경우 사용된 벡터를 반복해서 같은 수의 벡터를 만들게 된다.

2*5가 10이므로 rep(c(FALSE,TRUE),5) 정도로 생각할 수 있겠다. 그러나 4개일 경우는 어떨까?
4*2+2가 10이므로 두번 반복하고 앞의 두 원소를 사용할 것다.

c(rep(c(FALSE,TRUE,TRUE,FALSE),2),c(FALSE,TRUE,TRUE,FALSE)[1:2])정도로 생각하자.

```
〉 x[c(FALSE,TRUE,TRUE,FALSE)]
[1]  2  3  6  7 10
```

벡터에서 특정 조건을 만족하는 부분집합을 추출하는 것을 보였다. 그것은 연산자를 사용한 방법이었고 이번에는 subset이라는 함수를 이용하는 방법을 살펴보자.

x벡터의 원소 중 3으로 나눈 몫이 2인 원소들을 골라보자.

```
〉 subset(x, x%/%3==2)
[1] 6 7 8
```

x벡터의 원소 중 3으로 나눈 값이 2인 원소들을 골라보자.

```
〉 subset(x, x/3==2)
[1] 6
```

연산자를 사용하여 x벡터의 원소 중 3으로 나눈 몫이 2인 원소들을 골라보자.

```
〉 x[x%/%3==2]
[1] 6 7 8
```

연산자를 사용하여 x벡터의 원소 중 3으로 나눈 값이 2인 원소들을 골라보자.

```
〉 x[x/3==2]
[1] 6
```

결과는 동일하다.

x벡터에 음수를 취해보자.

```
〉 -x
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
```

x벡터를 2로 나눠보자.

```
〉 x/2
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 # 실수형 벡터
〉 x%/%2
 [1] 0 1 1 2 2 3 3 4 4 5 # 정수형 벡터
```

그러면 x벡터에 log를 취해보자. 여기서는 transform함수를 사용해본다. 사실 subset과 transform함수는 벡터에는 잘 어울리지 않다. 하지만 학습 측면에서 예

시해 본다.

```
> transform(x,x=log(x))
          x
1  0.0000000
2  0.6931472
3  1.0986123
4  1.3862944
5  1.6094379
6  1.7917595
7  1.9459101
8  2.0794415
9  2.1972246
10 2.3025851
```

그런데 연산 결과가 벡터가 아니다.

```
> is.vector(transform(x,x=log(x)))
[1] FALSE
```

transform함수가 데이터프레임으로 데이터 타입을 변경시켰다.

```
> is.data.frame(transform(x,x=log(x)))
[1] TRUE
```

그러면 데이터프레임을 벡터로 변경시키자.

```
> transform(x,x=log(x))[,1]
 [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595
1.9459101
 [8] 2.0794415 2.1972246 2.3025851
```

제대로 변경되었음을 알 수 있다.

```
> is.vector(transform(x,x=log(x))[,1])
[1] TRUE
```

문자형 벡터를 만들어 보자.

```
> human=c("이순신","강감찬","홍길동","임꺽정","이성계","정약용")
> human
[1] "이순신" "강감찬" "홍길동" "임꺽정" "이성계" "정약용"
```

human 벡터의 원소중에서 "정"이라는 문자가 들어 있는 이름을 골라보자.

```
> human[grep("정",human)]
[1] "임꺽정" "정약용"
```

human 벡터의 원소중에서 성이 "정"인 이름을 골라보자.

```
> human[grep("^정",human)]
[1] "정약용"
```

human 벡터의 원소중에서 이름이 "정약용", "홍길동"인 것을 제외해 보자.

```
> subset(human, !human %in% c("정약용","홍길동"))
[1] "이순신" "강감찬" "임꺽정" "이성계"
```

정수형 벡터 x와 논리형 벡터 y를 묶어보자.

```
> c(x,y)
 [1]  1  2  3  4  5  6  7  8  9 10  0  1  0  1  0  1  0  1  0  1
```

벡터란 하나 이상의 동일한 데이터 타입을 갖는 객체하고 하였다. 그런데 x와 y는 데이터 타입이 다르다. 이때는 자동적으로 형 변환이 발생한다. 논리형의 벡터가 정수형의 벡터로 바뀌었다. TRUE는 1로 FALSE는 0으로 바뀌었음을 알 수 있다.

다섯번째 원소의 값을 10으로 바꿔보자.

```
> x[5]=10
> x
 [1]  1  2  3  4 10  6  7  8  9 10]
```

마지막으로 다섯번째 원소의 값을 "a"로 바꿔보자.

```
> x[5]="a"
> x
 [1] "1"  "2"  "3"  "4"  "a"  "6"  "7"  "8"  "9"  "10"
```

정수형 벡터와 문자형 벡터의 연산에서는 문자형 벡터로 형변환이 이루어졌음을 알 수 있다.

자료분석에서 분석 모델 못지않게 중요한 것이 자료의 변환이다. 자료를 원하는대로 가공,변환하는 방법에는 여러 가지가 있으며 여기서 보인 예제는 그 일부에 지나지 않는다. R에서는 벡터가 자료의 기본이다. 벡터의 가공,변환하는 방법을 익히면 나머지 자료형의 가공 및 변환도 쉽게 익힐 수 있다.

벡터에 대해서

지난 번에는 벡터의 변환 및 가공의 관점에서 벡터를 개괄적으로 살펴 보았다. 이번에는 이들 벡터를 만드는 몇가지 방법에 대해서 알아 보자.

벡터를 이용하는 방법에는 함수를 이용하는 방법과 연산자를 이용하는 방법으로 나눌 수 있다. 함수를 이용하는 방법은 함수(Function)의 인수(parameter)가 벡터를 구체적으로 정의하고, 연산자를 이용하는 방법은 연산자(Operator)의 피연산자(Operand)가 벡터를 구체화 한다.

우선 연산자를 이용하는 방법에 대해서 알아 보자.

연산자를 이용하는 방법

1. Sequence연산자 (:)

Sequence는 이항연산자로 수치 벡터를 생성한다.(정수, 실수) 특징은 Sequence의 사전적 의미처럼 연속적인 벡터값을 생성한다. 등차수열을 생성한다고 생각하면 된다. 연산자는 from:to 처럼 사용하며 from이 초기 값이고, to가 종기 값이다. 공차는 1이다. 즉, from에서 시작해서 to까지의 1씩 증가하는 값을 생성한다.

'from, from+1, ..., to'

from이 정수가 아닐 경우에는 from+(n*1)이 to보다 작거나 같은 값이 벡터의 마지막 값이 된다. 만약 from〉to 일 경우에는 공차가 -1로 감소수열의 벡터를 생성한다. 또 다른 특징은 from의 값이 실수이면 실수, 정수이면 정수의 벡터를 생성한다는 점이다.

다음 예를 보자.

1부터 4의 값을 갖는 벡터를 만들어 보자.

```
〉 1:4
[1] 1 2 3 4
```

pi부터 6까지의 값을 갖는 벡터를 만들어 보자. 실수형 벡터가 생성된다.

```
〉 pi:6
[1] 3.141593 4.141593 5.141593
```

6부터 pi까지의 값을 갖는 벡터를 만들어 보자. 정수형 벡터가 생성된다.

```
〉 6:pi
[1] 6 5 4
```

-5부터 5까지의 값을 갖는 정수형 벡터를 만들어 보자.

```
> -5:5
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

복소수 3+5i에서 5+6i까지의 복소수형 벡터를 만들어 보자.

```
> 3+5i:5+6i
[1] 3+6i 4+6i 5+6i 6+6i 7+6i 8+6i
Warning messages:
1: imaginary parts discarded in coercion
2: out-of-range values treated as 0 in coercion to raw
```

아마 3+5i 4+5i 5+5i의 값을 생각했을 지 모르겠다. 그리고 Warning messages 도 출력됐다. 그런데 결과는 3+6i 4+6i 5+6i 6+6i 7+6i 8+6i로 나타났다.
이유를 살펴보자.

우선 연산자 우선 순위로 인해서 5i:5가 먼저 연산되었다.

```
> 5i:5
[1] 0 1 2 3 4 5
Warning messages:
1: imaginary parts discarded in coercion
2: out-of-range values treated as 0 in coercion to raw
```

여기서 5i는 0으로 바뀌어 실행이 되었다. 그러므로 0:5가 되서 0 1 2 3 4 5로 연산되었다. 두번 째로는 5i:5의 계산결과에 3이 더해졌다.

```
> 3+5i:5
[1] 3 4 5 6 7 8
Warning messages:
1: imaginary parts discarded in coercion
2: out-of-range values treated as 0 in coercion to raw
```

마지막으로 뒤의 6i가 더해져서 최종 연산의 결과가 출력 된 것이다.

연산의 순서를 정리하면 다음과 같다.

$$3 + 5i : 5 + 6i$$

$$\frac{\frac{1}{2}}{\frac{2}{3}}$$

"a"부터 "z"까지의 문자 벡터를 만들어 보자. 역시 오류가 발생하였다. 피연산자가 수치형만 지원하기 때문이다.

```
> "a":"z"
Error: unimplemented type 'character' in 'asReal'
```

그러면 "0"부터 "9"까지의 문자 벡터를 만들어 보자. 혹시나 형변환이 일어날 지 모르기 때문이다.

```
> "0":"9"
Error: unimplemented type 'character' in 'asReal'
```

역시 오류가 발생하였다. 피연산자가 수치형만 지원하기 때문이다. 형변환은 일차 연산 후 2차 연산으로 넘어갈 때나, 연산 후 특정 객체에 값을 할당할 때 발생하기 때문이다.

다음처럼 먼저 형변환 후 연산을 하면 문제가 발생하지 않는다.

```
> as.integer("0"):as.integer("9")
 [1] 0 1 2 3 4 5 6 7 8 9
```

함수를 이용하는 방법

1. Combine함수 (c)

c 함수는 값들을 묶어 벡터를 만드는 함수로 함수 원형은 다음과 같다.

c(..., recursive=FALSE)

...의 의미는 임의의 갯수를 갖는 인수를 의미한다. 문법이 허락하는 범위의 어떤 값이 다 올 수 있다.

```
> c(1,7:9)
[1] 1 7 8 9
> c(1:5, 10.5, "next")
[1] "1"    "2"    "3"    "4"    "5"    "10.5" "next"
> c(TRUE, FALSE)
[1]  TRUE FALSE
> c(TRUE, FALSE, 4)
[1] 1 0 4
> c(1,1:-2,c(pi,5))
[1]  1.000000  1.000000  0.000000 -1.000000 -2.000000  3.141593
5.000000
```

벡터의 원소에 이름을 부여할 수도 있다.

```
〉 x=c(a = 1, b = 2)
〉 x
a b
1 2
〉 is.vector(x) # 분명 x는 벡터이다.
[1] TRUE
〉 names(x)  # x벡터의 이름이 a와 b이다.
[1] "a" "b"
```

2. Sequence함수 (seq)

Sequence함수 seq도 연속적인 수치 벡터를 생성하며 Sequence 연산자 ":"와 유사하다. ":" 기능을 확장한 함수 정도로 이해하면 된다.

이 함수는 다음과 같은 다섯 가지의 사용 방법이 있다.

```
seq(from, to)                  --〉 type 1
seq(from, to, by= )            --〉 type 2
seq(from, to, length.out= )  --〉 type 3
seq(along.with= )              --〉 type 4
seq(from)                      --〉 type 5
```

a. type 1

seq(from, to) : from:to와 동일하다.

```
〉 seq(pi,6)
[1] 3.141593 4.141593 5.141593
〉 seq(6,pi)
[1] 6 5 4
```

b. type 2

seq(from, to, by) : from:to에서 공차(증감의 단위)가 by인 벡터를 생성한다. by가 1이거나 -1이면 from:to와 동일하다.

```
〉 seq(pi,6, by=1)
[1] 3.141593 4.141593 5.141593
```

> seq(0,9, by=1.5)
[1] 0.0 1.5 3.0 4.5 6.0 7.5 9.0
> seq(6, pi, by=-0.5)
[1] 6.0 5.5 5.0 4.5 4.0 3.5

c. type 3
seq(from, to, length) : from과 to를 length개의 등간격으로 나눈 벡터를 생성한다.

> seq(0,1, length=11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(6,pi, length=5)
[1] 6.000000 5.285398 4.570796 3.856194 3.141593

d. type 4
seq(along.with) : along.with의 길이 만큼의 벡터를 생성한다.
1:length(along.with) 정도로 생각하면 된다.

> seq(5:10)
[1] 1 2 3 4 5 6
> x=rnorm(10)
> seq(x)
[1] 1 2 3 4 5 6 7 8 9 10

e. type 5
seq(from) : 1:from과 같다. 그러므로 함수의 원형은 seq(from)보다 seq(to)이 더 타당하겠다.
즉 from이 1인 from:to라 할 수 있다.

> seq(5)
[1] 1 2 3 4 5
> seq(-5)
[1] 1 0 -1 -2 -3 -4 -5

3. Sequence함수 (sequence)
Sequence함수 sequence도 연속적인 수치 벡터를 생성하다. 함수 원형은 다음과 같다.
sequence(nvec)

인수 nvec는 벡터의 값으로 이 원소들의 값이 seq(from)의 인수로 사용되는 벡터들의 집합을 생성한다. 예를 들면, nvec라는 벡터가 3개의 원소로 이루어졌다고 가정하면,
sequence(nvec)는 c(seq(nvec[1]),seq(nvec[2]),seq(nvec[3]))이 된다.

```
> sequence(c(3,2))
[1] 1 2 3 1 2
> sequence(c(3,2,4))
[1] 1 2 3 1 2 1 2 3 4
> sequence(c(3,2,-4))
 [1]  1  2  3  1  2  1  0 -1 -2 -3 -4
```

4. Replicate함수 (rep)
rep함수는 벡터의 원소들을 반복하여 벡터를 확장하는 함수다. 함수의 원형은 다음과 같다.
rep(x, times, ...)
rep(x, times, length.out, each, ...) # S 버전 3에서의 함수 원형

여기서 x는 반복하고자 하는 벡터를 나타내고 times는 반복의 횟수로 벡터값이나 스칼라 값이 온다. length.out는 반환하는 벡터의 길이를 지정하는 인수이고, each는 x의 각각 반복수를 나타낸다.

1부터 4의 벡터를 두번 반복해 보자.

```
> rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4
```

1부터 4의 벡터를 원소에 대해 각각 두번씩 반복해 보자.

```
> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
```

1부터 4의 벡터를 원소에 대해 각각 두번씩 반복해 보자.

```
> rep(1:4, c(2,2,2,2))
[1] 1 1 2 2 3 3 4 4
```

1부터 4의 벡터를 원소에 대해 각각 두번/한번/두번/한번씩 반복해 보자.

```
> rep(1:4, c(2,1,2,1))
[1] 1 1 2 3 3 4
```

1부터 4의 벡터를 각각 1:4회 반복해 보자.

```
> rep(1:4, 1:4)
 [1] 1 2 2 3 3 3 4 4 4 4
```

1부터 4의 벡터를 원소에 대해 각각 두번씩 반복하여 4개만 출력해 보자.

```
> rep(1:4, each = 2, len = 4)
[1] 1 1 2 2
```

1부터 4의 벡터를 원소에 대해 각각 두번씩 반복하여 10개를 출력해 보자.
4*2=8이므로 두 개가 남는 데 이 경우는 앞의 1의 두번 반복 수를 더해서 출력 한다.

```
> rep(1:4, each = 2, len = 10)
 [1] 1 1 2 2 3 3 4 4 1 1
```

1부터 4의 벡터를 원소에 대해 각각 두번씩 반복하는 것을 세번 수행한다.

```
> rep(1:4, each = 2, times = 3)
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

1을 8번 반복한다. 그런데 7번 밖에 반복이 안됐다. round off error로 인한 결과다.

```
> rep(1, 40*(1-.8))
[1] 1 1 1 1 1 1 1
```

1을 8번 반복한다. round off error를 방지했다.

```
> rep(1, 40*(1-.8)+1e-7)
[1] 1 1 1 1 1 1 1 1
```

5. vector 함수

vector 함수는 벡터를 만드는 함수이다.

vector(mode = "logical", length = 0)의 함수 원형을 갖는다. length개의 mode를 갖는 벡터를 생성한다.

vector 함수의 기본형으로 논리형으로 원소가 0인 벡터를 생성한다.

```
> vector()
logical(0)
```

mode의 값으로 integer를 사용하여 5개의 원소를 갖는 수치 벡터를 생성한다.

```
> vector(mode="integer", 5)
[1] 0 0 0 0 0
```

mode의 값으로 duoule을 사용하여 5개의 원소를 갖는 수치 벡터를 생성한다.

```
> vector(mode="double", 5)
[1] 0 0 0 0 0
```

5개의 원소를 갖는 수치 벡터를 생성한다. mode의 값으로 float는 오류를 발생시킨다.

```
> vector(mode="float", 5)
Error in vector(mode = "float", 5) : vector: cannot make a vector of
mode "float".
```

mode의 값으로 numeric을 사용하여 5개의 원소를 갖는 수치 벡터를 생성한다.

```
> vector(mode="numeric", 5)
[1] 0 0 0 0 0
```

integer, double은 모두 numeric형으로 변환된다. 즉, 오류는 나지 않더라도 numeric을 사용해야 한다.

```
> vector(mode="integer")
numeric(0)
```

5개의 원소를 갖는 논리 벡터를 생성한다.

```
> vector(mode="logical", 5)
[1] FALSE FALSE FALSE FALSE FALSE
```

5개의 원소를 갖는 문자열 벡터를 생성한다.

```
> vector(mode="character", 5)
[1] "" "" "" "" ""
```

5개의 원소를 갖는 복소수 벡터를 생성한다.

```
> vector(mode="complex", 5)
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
```

vector함수는 벡터를 정의하는 것보다는 선언의 차원으로 많이 사용한다. 즉, 벡터의 초기화하는 함수라고 할 수 있다.

6. 기타함수

벡터의 원소는 logical, numeric, complex, character의 네가지 형의 원소를 갖는다. 그리고 이들d에 대해 벡터를 만드는 함수를 가지고 있다. 함수의 원형은 다음과 같다.

logical(length = 0)

numeric(length = 0)

complex(length = 0)

character(length = 0)

원소가 5인 논리형 벡터를 생성한다.

```
〉 logical(5)
[1] FALSE FALSE FALSE FALSE FALSE
```

원소가 5인 수치형 벡터를 생성한다.

```
〉 numeric(10)
 [1] 0 0 0 0 0 0 0 0 0 0
```

원소가 5인 복소수형 벡터를 생성한다.

```
〉 complex(5)
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
```

원소가 5인 문자형 벡터를 생성한다.

```
〉 character(5)
[1] "" "" "" "" ""
```