Chapter 1

벡터/행렬/배열

R은 분석가가 어떠한 분석활동을 진행하는데 도움을 주는 도구이므로, 이를 잘 다룬다면 분석활동을 빠르게 진행할 수도 있고 효율적으로 관리할수도 있다는 의미를 내포하고 있기도 합니다. 사용자 측면에서 본다면 이러한 도구를 잘 활용하는 것은 주어진 사용자 환경에 익숙해져 있다는 의미와도 같습니다. 따라서, 본 챕터에서는 R을 처음 접하는 사용자들이 R이라는 도구에 익숙해지는 것을 주 목적으로 제일먼저 계산기능으로의 활용을 먼저 다루도록 하겠습니다.

독자가 R이라는 도구에 다소 익숙해져있다는 전제하에 우리는 벡터, 행렬, 그리고 배열이라는 데이터형에 대해서 자세히 알아볼 것입니다. 그 이유는 추후에 다루어질 모든 내용들이 이것들의 활용에 매우 깊은 관계가 있기 때문입니다.

따라서, 독자는 이 부분의 내용이 어렵고 지루하더라도 꼭 다 익히고 다른 챕터들을 살펴보시길 부탁드립니다. 이 문서의 지은이 역시 독자가 최대한 쉽게 이해할 수 있도록 그 내용을 자세히 기술하도록 하겠습니다.

1.1 R의 시작과 종료

먼저 R을 시작하면 아래와 같은 화면을 볼 수 있으며, 커서가 우측화살표 (>) 바로 뒤쪽에서 깜빡깜빡거리는 것을 볼 수 있습니다. 이 우측 화살표를 **프롬프트** (prompt)라고 부르며, 우리는 이 프롬프트라는 기호 뒤에 명령어를 작성함으로서 R에게 "내가 이러한 명령을 줄테니 수행해 줄래?" 라고 하는 것입니다. 이렇게 내가 명령을 주고, R이 그 결과를 돌려받는 형식으로 사용하는 것을 "**대화식 사용**" (interactive mode) 한다고 합니다. 물론 명령어 입력이 끝났으면 엔터를 치면 됩니다. 또한, 이렇게 R과 사용자간의 대화가 이루어지는 공간을 **콘솔**이라고 합니다.

```
gnustats@CHL072:~$ R

R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i686-pc-linux-gnu (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
```

```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

R이 입력된 명령어를 처리하여 사용자에게 명령어의 수행 결과를 보여주는 것을 아래와 같음을 확인하기 위해서 숫자 1 을 넣어보고 2 도 넣어봅니다. 아래와 같이 R은 [1] 이라는 기호 뒤에 사용자가 입력한 숫자를 보여줍니다. ([1]이라는 기호는 실제로 R을 사용하는데 있어 특정한 의미를 가지고 있습니다. 이것에 대한 설명은 잠시 뒤에 나올 벡터를 다룰때 설명할 것입니다). 이러한 결과는 R이 "사용자님께서 1을 입력하셨으니까 내가 1이라는 것을 입력받았다는 것을 알려드립니다" 라고 생각하시면 됩니다.

```
> 1
[1] 1
> 1+1
[1] 2
```

이렇게 사용자가 입력하는 것과 R이 돌려주는 것들을 일반적으로 객체라는 개념으로 지칭합니다. (객체라는 개념은 R을 사용함에 있어 객체지향 프로그래밍과 관계된 매우 중요한 개념입니다. 그러나, 현재 여러분은 객체와 객체지향 프로그래밍의 의미를 모르셔도 괜찮습니다. 그 이유는 우리는 이러한 개념들에 대해서 추후에 다룰 것이기 때문입니다. 현재로서는 콘솔내에서 다루어지는 모든 것들을 객체라고 생각하시면 됩니다. 즉, 사용자가 입력하는 숫자 1을 R은 객체로 인식합니다. 만약 사용자가 객체가 무엇인지 지금 당장 알아야겠다면 **??**를 참고하시길 바랍니다).

이제 R을 이용하여 필요한 작업을 다 했다고 가정한 뒤에 R을 종료해야 한다고 가정합니다. 이를 위해서는 아래와 같이 `q()`라는 함수를 이용하면 됩니다. 여기에서 q는 quit의 약자입니다.

```
> q()
Save workspace image? [y/n/c]: n
```

이 명령어가 입력되면 작업공간 (workspace)의 이미지를 저장하겠는가는 메시지를 보게 됩니다. 이 메시지의 의미는 콘솔내 현재세션에서 작업하면서 사용한 명령어들과 데이터들, 그리고 작업을 하면서 발생하고 다루었던 모든 객체들에 대한 저장유무입니다. 경우에 따라서 다르겠으나, 데이터가 크고 계산시간이 오래 걸리는 작업을 수행했을 경우 작업공간의 이미지를 저장하는 것을 고려해 볼 수 있습니다.

1.2 1차원 벡터와 벡터라이제이션

R의 가장 큰 특징은 아마도 객체들을 다룸에 있어 벡터를 기반으로 작동하는 것이 것입니다. (아마도 독자는 이 말이 무슨 뜻인지 현재는 모를 것입니다. 그러나, 본 섹션을 읽어가면서 벡터방식의 객체조작과 관리라는 것을 알게 될 것이므로 너무 염려하지 않으셔도 됩니다).

벡터란 어떤 객체가 1차원 공간에서 동일한 형식을 가진 구성요소들이 순차적으로 나열되어 있는 것을 의미합니다. ~~벡터라는 개념은 컴퓨터가 어떠한 임의의 것을 1차원에서 다루는 것을 의미합니다.~~ 예를들어, 수학에서 1이라고 표시하는 것을 컴퓨터가 알아듣게 입력을 한다면 단순히 1 이라는 값만을 입력하면 됩니다. 그러나, 실제 작업에서 우리는 동일한 연산을 반복해야 하는 경우가 있습니다. 따라서, 숫자 1이 아닌 1, 3, 5, 7, 9 와 같은 여러개의 숫자들을 나열한 수열의 형태를 만듭니다. 이제, 1,3,5,7,9는 수열의 구성요소가 되는 것입니다. 이러한 수열의 표현방식을 R이 알아듣도록 하는 것을 우리는 벡터라는 개념이라고 이야기합니다.

벡터방식이란 개념을 이해하기 위해서 계산기로서 R은 어떤 기능을 가지고 있는지 먼저 이해해야 합니다.

1.2.1 산술연산

먼저 R은 아래와 같이 마치 간단한 전자계산기와 같이 사용하는 것도 가능합니다.

```
# 사칙연산 (더하기, 빼기, 곱하기, 나누기)
```

$$1 + 2 - 3 + 4 * 5 - 6/3 \tag{1.1}$$

이를 R로 수행하기 위해서는 다음과 같이 입력합니다.

```
> 1 + 2 - 3 + 4*5 - 6/3
[1] 18
```

연산자라는 것은 어떤 특정 역할을 수행하는 기호이며, 산술연산에 대한 연산자 우선순위는 수학적 연산순서와 동일합니다.

다음과 같은 다양한 수학적인 연산이 가능합니다.

```
# 제곱
> 3^2
[1] 9

# 자승
> 3^4
[1] 81

# 지수
> exp(3)
[1] 20.08554
```

```
# 로그
> log(3)
[1] 1.098612

# 파이상수 값
> pi
[1] 3.141593

# 삼각함수 사인, 코사인, 탄젠트
> sin(0)
[1] 0

> cos(0)
[1] 1

> tan(45)
[1] 1.619775

# 나누기
> 15/4
[1] 3.75

# 몫
> 15 %/% 4
[1] 3

# 나머지
> 15 %% 4
[1] 3
> 15 %% 2
[1] 1
```

이제 R의 벡터단위의 연산이라 것을 알아보도록 합니다.

먼저 아래와 같이 1, 2, 3, 4 라는 각각의 숫자들을 하나로 묶어 하나의 벡터 (즉, 수열)을 생성해보도록 합니다. 이때, 이렇게 숫자들을 하나로 묶어 주는 것은 c()라는 함수를 통하여 이루어지게 됩니다. 여기에서 c는 "합치다"라는 의미의 combine의 약자입니다.

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

어떤 작업을 하기 위해서 위와 같이 일련의 숫자들을 일일이 손가락으로 매번 입력을 해야한다면 매우 번거로울 것입니다. 따라서, R은 이렇게 입력된 내용을 어떤 이름을 주어 저장해주도록 하는 변수라는 기능

을 제공합니다. 이러한 변수를 사용하기 위해서 특별한 선언을 필요로 하지는 않으며, 아래와 같은 방법으로 사용합니다.

```
> x <- c(1,2,3,4)
> x
[1] 1 2 3 4
```

이것은 1,2,3,4라는 일련의 숫자들을 한데 합친 것을 x 라고 이름을 붙인 변수에 저장을 한다는 의미입니다. 그리고, 이것을 c(1,2,3,4)라는 벡터 객체를 변수 x에 대입 또는 할당했다고 하며, 이제부터는 x라는 변수명을 사용하기만 하면 위에서 입력한 벡터 객체를 다시 불러와 활용할 수 있습니다.

위에서 할당을 $<-$ 기호를 사용하였으나, 이러한 할당은 아래와 같은 방법으로도 가능합니다.

```
> x <- c(1,2,3,4)
> x
[1] 1 2 3 4
> c(1,2,3,4) -> x
> x
[1] 1 2 3 4
> x = c(1,2,3,4)
> x
[1] 1 2 3 4
```

이는 단순히 사용자의 편의를 위한 것뿐이니 마음에 들어하는 방식을 골라서 사용하시면 됩니다.

이제 R이 정말로 x라는 것을 벡터로 인식하는지 is.vector()라는 함수를 사용하여 알아보도록 합니다.

```
> is.vector(x)
[1] TRUE
```

한개의 숫자와 하나의 벡터를 구분짓는 특징은 몇개의 숫자들이 한데 묶였는가 이므로 이를 확인하기 위해서는 length()라는 함수를 사용합니다.

```
> length(x)
[1] 4
```

벡터단위의 연산 이제 벡터단위의 연산이 무엇인가 알아보도록 합니다. 위에서 생성한 x 라는 변수에 3을 더해봅니다. 이때, 우리는 상식적으로 x라는 변수에 있는 벡터를 구성하는 모든 구성요소에 3이 더해진 결과를 볼 수 있기를 기대합니다.

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

이제 x라는 벡터에 제곱을 해봅니다. 여러분들은 벡터의 각 구성요소의 값들이 제곱이 되었음을 알 수 있습니다.

```
> x^2
[1]  1  4  9 16
```

이번에는 10으로 나누어 보겠습니다.

```
> x/10
[1] 0.1 0.2 0.3 0.4
>
```

이렇게 어떤 벡터에 특정 주어진 연산을 수행하고자 할때 벡터의 각 구성요소에 주어진 연산이 한 번에 모두 수행되는 것을 바로 벡터단위의 연산이라고 합니다. 이를 좀 더 멋있게 부르는 말이 어떤 연산에 대한 벡터라이징이라고 하는 것입니다.

1.2.2 벡터를 생성하는 다양한 방법들

위에서는 단순히 c()라는 함수를 이용하여 벡터를 생성하였으나, R은 다양한 보다 다양한 방법을 제공하고 있습니다.

seq()함수의 사용 먼저 아래와 같이 사용되는 seq()라는 함수가 있습니다.

```
> seq(0, 1, 0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

위에서 사용한 명령어는 0과 1 사이에 0.1 을 단위로 하는 시퀀스(즉, 수열)을 생성하라는 명령어입니다.

여기에서 R을 사용하는 한가지 팁을 알려주고자 합니다. seq() 함수의 예제와 같이 사용하고자 하는 함수에는 여러가지 입력받는 인자들이 있습니다. 입력되는 인자들의 순서가 틀리면 R은 잘 못된 결과를 보여줍니다. 따라서, 이러한 실수를 줄이고자 함수를 사용할때 가급적이면 인자명과 함께 사용하는 것이 좋습니다. 인자명을 영문으로는 argument name 이라고도 할 수 있으나, R Documentation에서 이를 설명하거나 시스템 메시지에서 사용되는 경우는 주로 named argument(지시된 인자) 이라고 합니다. 따라서, 영문 메시지를 읽을때 named argument 란 단어가 보인다면 이는 함수에 사용된 인자명을 말하는 것입니다.

```
> seq(from=0, to=1, by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

by 라는 인자명을 사용하지 않고, length.out 이라는 인자명을 이용한다면 아래와 같은 결과를 얻을 수 있습니다. 이는 0 부터 1 사이에서 균등한 길이를 가지는 5개의 숫자를 생성한다는 의미입니다.

```
> seq(from=0, to=1, length.out=5)
[1] 0.00 0.25 0.50 0.75 1.00
```

만약 by 라는 인자를 넣지 않는다면 seq()함수의 기본작동원리는 by=1이라고 가정하며, 이를 우리는 디폴트 (기본값)이라고 합니다.

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

콜론 연산자의 사용 seq()라는 함수외에도 이러한 수열은 콜론 연산자를 이용하여 생성할 수도 있습니다. 먼저 seq(0,5,1)과 같은 결과를 콜론을 이용하여 생성해보도록 합니다.

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

이 수열을 거꾸로도 생성할 수 있습니다.

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

콜론의 경우 수의 증감은 1로 고정됩니다.

rep() 함수의 사용 수열을 생성하는 또 다른 방법은 rep() 함수를 활용하는 것인데, 이는 replicates (반복)이라는 의미입니다. 따라서, 1이라는 숫자를 5 번 반복하고자 한다면 아래와 같이 합니다.

```
> rep(1,5)
[1] 1 1 1 1 1
```

이제 살짝 응용해봅니다. 먼저 1,2,3 이라는 수열을 만든뒤 이 수열자체를 세번 반복하고자 한다면 아래와 같이 할 수 있습니다.

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

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

조금 더 응용해 봅니다. 0을 1번 반복하고, 1을 2번 반복하고, 2를 3번 반복한 수열을 생성해 봅니다.

```
> c(rep(0,1), rep(1,2), rep(2,3))
[1] 0 1 1 2 2 2
```

이제 c(), seq(), 그리고 rep()를 모두 사용하여 벡터를 생성해 보겠습니다.

```
> a <- c(1, 2, 3, 4)
> b <- seq(1,4)
> c <- 1:4
>
> a
[1] 1 2 3 4
> b
[1] 1 2 3 4
> c
[1] 1 2 3 4
>
> d <- rep(seq(from=1, to=4), 3)
> d
```

```
[1] 1 2 3 4 1 2 3 4 1 2 3 4
>
> e <- rep(a, each=3)
> e
[1] 1 1 1 2 2 2 3 3 3 4 4 4
>
```

rep()에서 each의 사용에 따른 결과 차이를 발견하시기 바랍니다.

1.3 행렬연산과 2차원

여기까지 우리는 벡터라는 1차원 수열에 대해서 이야기를 했습니다. 그러나, 지금부터는 행과 열이라는 개념을 이용하여 1차원에서 2차원의 숫자들의 배치에 대해서 이야기 할 것입니다. 이러한 숫자들의 나열을 행렬(matrix)이라고 하며, R에서는 아래와 같은 다양한 연산기능을 제공하고 있습니다.

먼저 행렬 연산에 대한 이해를 돕기 위해서 아래와 같은 간단한 행렬을 생각해 봅니다.

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \tag{1.2}$$

여기에서 행렬의 구성요소는 $1, 2, 3, 4, 5, 6$ 이며, 크기는 3행 2열입니다.

행렬의 생성: 위에서 수학적으로 표현된 A 라는 행렬을 R에 입력하기 위해서는 아래와 같이 합니다.

```
> M <- matrix(c(1,2,3,4, 5, 6), ncol=2)
> M
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
>
```

그러나, 사용자는 때때로 행렬의 요소들을 행방향으로 나열하고자 할 수도 있습니다.

```
> M1 <- matrix(c(1,2,3,4, 5, 6), ncol=2, byrow=TRUE)
> M1
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

ncol은 열의 수를 지정해 주는 역할을 하는 것이고 byrow가 수가 입력되는 순서를 결정합니다.

그런데, 여기에서 반드시 알고 넘어가야 할 부분이 바로 벡터와 행렬간의 관계입니다. 행렬을 생성하기 위해서는 첫번째 인자에 벡터를 사용합니다. 이는 바로 R이 행렬을 어떻게 처리해 주는지 알게 되는 부분입니다.

먼저, 컴퓨터는 벡터와 행렬이 무엇인지 구분을 하지 못합니다. 단순히 숫자의 나열이라고만 알고 있습니다. 따라서, 위에서 `c(1,2,3,4, 5, 6)`라는 벡터를 행렬임을 알게 해주기 위해서는 행과 열이라는 속성을 부여함으로서 R이 행렬임을 알게 하는 것입니다. 다음과 같이 `attributes()`라는 함수를 이용하여 M이라는 행렬이 어떤 속성을 가지고 있는지 확인해 봅니다.

```
> attributes(M)
$dim
[1] 3 2
```

행과 열이라는 속성은 행렬을 생성시 인자명을 통해서 생성된 것이므로, attributes()로 확인했을 때 이들의 정보가 dim 에 저장되어 있음을 알 수 있습니다. 그런데, 행과 열의 속성은 행렬의 크기를 나타내기도 하기 때문에 이들의 값을 따로 빼내어 사용하고 싶다면 dim() 이라는 함수를 이용하면 됩니다.

```
> dim(M)
[1] 3 2
>
```

첫번째 요소는 행의 개수이고, 두번째 요소는 열의 개수입니다.
또다른 예제는 아래와 같습니다.

```
> M2 <- matrix(1:15, ncol=5, nrow=3)
> M2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15
> attributes(M2)
$dim
[1] 3 5
```

이제 행렬을 생성하였으니, 간단한 행렬의 연산에 대해서 살펴보도록 하겠습니다.

행렬의 전치: 2행 2열인 정방행렬을 열방향으로 나열하는 것을 행방향으로 나열하게 된다면, 이는 행렬의 전치를 의미하게 됩니다. 따라서, 행렬의 전치를 수행했을대 동일한 결과를 가지게 될 것입니다.

```
> M <- matrix(1:4, ncol=2)
> M
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> t(M)
```

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

t()는 전치(transpose)의 약어입니다.

행렬을 다시 벡터로 전환하기: 어떤 경우는 다시 행렬을 벡터의 형식으로 불러와야 할 경우도 있습니다.

```
> c(M)
[1] 1 2 3 4
>
```

대각행렬 기능활용: 행렬 A의 대각행렬의 원소들은 1과 4인데, 이를 얻는 방법은 아래와 같이 diag() 함수를 사용하는 것입니다.

```
> diag(M)
[1] 1 4
```

만약, 대각행렬의 원소가 c(3,4)를 가지는 정방대각행렬을 생성하고자 한다면 아래와 같이 사용할 수 있습니다 .

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

또한, I 행렬을 생성하는데 사용할 수 있습니다.

```
> diag(2)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
>
```

본 챕터에서의 목적은 R의 기초적 사용에 익숙해지는 것이므로, 아래의 주제에 대해서는 "수학/확률/수치해석" 이라는 챕터를 살펴보아 주시길 바랍니다.

- 행렬값 (determinant) 구하기
- 역행렬 (inverse matri) 구하기
- 역행렬을 이용하여 선형 연립방정식의 해를 구하기
- 크로넥터 프로덕트 (Kronecker product) 계산하기
- 행렬의 분해 1 - singular value decomposition

- 행렬의 분해 2 - cholesky decomposition
- 행렬의 분해 3 - spectral decomposition
- 고유값 (eigen value)와 고유벡터 (eigen vector) 구하기

여기까지 1개의 행렬을 이용한 기초 행렬 연산을 살펴보았습니다.

행렬의 사칙연산 이제 2개의 이상의 행렬을 이용한 사칙연산을 살펴봅니다.

```
> A <- matrix(c(1,2,3,4), ncol=2)
> B <- matrix(c(5,6,7,8), ncol=2)

# 행렬의 덧셈
> A + B
     [,1] [,2]
[1,]    6   10
[2,]    8   12

# 행렬의 뺄셈
> A - B
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4

# 행렬의 곱셈
> A %*% B
     [,1] [,2]
[1,]   23   31
[2,]   34   46

# 행렬의 구성요소 단위의 곱셈
> A * B
     [,1] [,2]
[1,]    5   21
[2,]   12   32
>

# 나머지 구하기
> B %% A
     [,1] [,2]
[1,]    0    1
[2,]    0    0
```

```
# 나누기
> B / A
     [,1]     [,2]
[1,]    5 2.333333
[2,]    3 2.000000
```

위에서 살펴본 바와 같이 행렬은 모두 구성요소 단위의 연산을 수행하게 됩니다.

행렬의 결합 간혹 두 개 이상의 행렬들을 결합할 경우가 있습니다. 이런 경우에는 행방향 혹은 열방향의 결합이 아래와 같은 방법으로 가능합니다.

만약, 아래와 같이 A와 B 라는 행렬이 존재한다면,

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> B
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

이 두개의 행렬을 행방향으로 결합하고자 한다면 rbind()라는 함수를 이용합니다.

```
> R <- rbind(A, B)
> R
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    7
[4,]    6    8
```

열방향으로 결합을 하고자 한다면 cbind()함수를 이용합니다.

```
> C <- cbind(A, B)
> C
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
>
```

TODO:

- is.matrix() 에 대한 설명 필요
- 연습문제가 제공되어지면 좋을 것 같음.

1.4 배열과 다차원

여기까지 벡터와 행렬이라는 1차원과 2차원에 숫자들을 나열하는 방법을 살펴보았습니다. 이제 3차원에 숫자들을 나열하려면 어떻게 해야할까요? 보다 더 나아가서 n 차원에 숫자를 나열하는 것은 어떻게 할까요? 이러한 방법을 제공하는 것이 바로 배열입니다.

배열의 생성원리는 배열의 원리와 동일하게 벡터의 값을 입력한 뒤 차원이라는 속성을 이용하여 표현하는 것입니다. 아래의 예에서 8개의 원소로 구성된 a 라는 벡터를 각 차원이 2행 2열인 행렬을 가지는 2개의 차원으로 표시한 것입니다.

```
> a <- 1:8
> A <- array(a, dim=c(2,2,2))
> A
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
>
```

여기에서 dim=c(2,2,2)라고 입력된 차원정보는 (열의개수, 행의개수, 차원의 개수) 라는 형식을 가지고 있습니다. 그런데 다음의 예제와 같이 만약 1에서 20까지의 정수를 3차원 배열에 입력할 때 차원과 열의 순서에 따라 차례대로 숫자가 입력되고 나머지 공간에는 다시 1부터 시작하여 숫자가 채워지는 것에 주의해야 합니다.

```
> a <- c(1:20)
> A <- array(a, dim=c(3, 3, 3))
> A
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2
```

```
     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

     [,1] [,2] [,3]
[1,]   19    2    5
[2,]   20    3    6
[3,]    1    4    7
```

TODO:

- 연습문제가 제공되어지면 좋을 것 같음.

1.5 인덱싱과 라벨링

여기까지는 아주 작은 크기의 벡터, 행렬, 그리고 배열을 예를 들어 설명했습니다. 그러나, 실제적으로는 매우 다양한 크기의 벡터, 행렬, 그리고 배열을 다루게 될 것이며, 이들을 구성하는 요소들중 특수한 구성요소들만을 사용해야 할 경우도 있을 것입니다.

1.5.1 인덱싱 하기

위에서 언급한 바와 같이 벡터, 행렬, 배열은 모두 벡터를 기초로 하되, 그 속성의 지정만이 다른것입니다. 그리고, 속성에 따라 구성원소들이 배치가 될때 이들은 기본적으로 열방향으로 나열됩니다. 왜 기본적으로 열방향으로 나열하는가에 대한 답변은 일반적인 통계처리는 변수를 중심으로 이루어지고, 이 변수는 일반적인 데이터 나열시 열방향으로 존재하기 때문입니다. 즉, 개별 관측치는 여러개의 변수들을 1행에 나열하는 반면, 각 변수는 여러개들의 관측치를 1개의 열에 넣기 때문입니다.

벡터기반 인덱싱 벡터의 구성요소를 선택하는 것을 인덱싱이라고 합니다. 이 인덱싱은 구성요소가 주어진 벡터안에 몇 번째에 놓여있는가에 대한 위치에 대한 정보입니다. 그리고, 이렇게 파악된 위치 정보를 이용하여 구성요소를 선택한 값을 보여주는 것은 열린 중괄호 '['와 닫힌 중괄호 ']'를 이용하여 표시합니다. 아래의 예를 살펴봅니다.

먼저 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112 의 수열을 가진 x 라는 벡터를 생각합니다.

```
> x <- 101:112
> x
 [1] 101 102 103 104 105 106 107 108 109 110 111 112
```

이제 x 벡터의 세번째 값을 알고 싶습니다.

```
> x[3]
[1] 103
```

여기에서 x의 세번째 라는 것이 위치정보라는 인덱스이고, 세번째에 해당하는 x의 값인 103을 불러오는 것은 x 벡터를 인덱싱했다고 합니다. 인덱싱은 벡터형식의 인덱스를 활용할 수도 있습니다. 만약, 3번째부터 7번째까지의 x의 값들을 가져오고 싶다면 아래와 같이 합니다.

```
> x[3:7]
[1] 103 104 105 106 107
```

조금 더 자유롭게 인덱싱을 해보겠습니다. x의 홀수번째에 해당하는 값들만 뽑아봅니다.

```
> x[c(1,3,5,7,9,11)]
[1] 101 103 105 107 109 111
```

위에서 보는 것과 같이 R은 벡터식 연산이라는 특징을 이용하여 활용할 수 있습니다.

벡터의 라벨을 이용한 인덱싱 인덱스를 이용하여 벡터의 구성요소를 인덱싱할 수 있으나, 벡터 구성요소 자체에 이름을 붙여 이들의 값을 불러올 수 도 있습니다. 먼저 x 벡터의 구성요소들에 이름을 붙이기 위해서 names()라는 함수를 사용합니다.

```
> names(x) <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l")
> x
  a   b   c   d   e   f   g   h   i   j   k   l
101 102 103 104 105 106 107 108 109 110 111 112
```

이제 "a"와 "i" 라는 이름에 해당하는 x의 값을 불러옵니다.

```
> x[c("a", "i")]
  a   i
101 109
```

행렬의 인덱싱 위에서는 1차원 벡터의 경우를 살펴보았으므로, 이제 2차원 행렬의 인덱싱을 살펴 봅니다. 먼저, xm이라는 4행 3열인 행렬을 생각해 봅니다.

```
> x <- 101:112
> xm <- matrix(x, ncol=3)
> xm
     [,1] [,2] [,3]
[1,]  101  105  109
[2,]  102  106  110
[3,]  103  107  111
[4,]  104  108  112
>
```

위에서 설명한대로 행렬이 벡터라는 것을 이해하기 위해서는 아래를 살펴봅니다. 먼저 xm 이라는 행렬의 2행 3열에 위치한 값은 110 임을 알 수 있습니다. 만약 열방향 벡터대로 이를 처리한다면 110 이라는 값은 이 행렬이 가진 벡터의 위치가 10번째일 것입니다.

```
> xm[2,3]
[1] 110
> xm[10]
[1] 110
```

이제, 하나의 구성요소말고 본 행렬의 부분집합에 대해서 알아봅니다. 먼저, 행렬 xm의 3번째 열에 해당하는 값들만 뽑습니다.

```
> xm[,3]
[1] 109 110 111 112
```

이제는 2번째 행에 해당하는 값들만 뽑습니다.

```
> xm[2,]
[1] 102 106 110
```

좀 더 나아가서, 1번째와 2번째 행에 해당하되 3번째 열에 해당하는 구성요소만 뽑고 싶습니다.

```
> xm[1:2, 3]
[1] 109 110
```

좀 더 확장해봅니다. 1과 2행에 있으며 2와 3열에 있는 구성요소를 뽑고 싶습니다.

```
> xm[1:2, 2:3]
     [,1] [,2]
[1,]  105  109
[2,]  106  110
```

한 번 더 연습합니다.

```
> xm[c(1,4), c(1,3)]
     [,1] [,2]
[1,]  101  109
[2,]  104  112
>
```

이제 이러한 구성요소들을 행렬의 행과 열의 이름을 활용해 보도록 합니다. 이를 위해서 먼저, 행렬의 행과 열에 각각 이름을 붙여줘야 합니다. 함수 `rownames()`와 `colnames()`는 이를 가능하게 해줍니다.

```
> rownames(xm) <- c("R1", "R2", "R3", "R4")
> colnames(xm) <- c("C1", "C2", "C3")
> xm
    C1  C2  C3
R1 101 105 109
R2 102 106 110
R3 103 107 111
R4 104 108 112
```

이제 2번째와 3번재 열에 해당하며 2번째와 4번째 행에 해당하는 구성요소를 행과 열의 이름을 이용하여 선택해 봅니다.

```
> xm[c("R2", "R4"), c("C2", "C3")]
    C2  C3
R2 106 110
R4 108 112
```

배열의 인덱싱 이제 배열의 경우를 살펴보도록 하겠습니다. 먼저 2행 2열이 3차원이 놓은 구성요소를 가진 배열 A 을 생성합니다.

```
> a <- 101:112
> A <- array(a, dim=c(2,2,3))
> A
, , 1

     [,1] [,2]
[1,]  101  103
[2,]  102  104

, , 2

     [,1] [,2]
[1,]  105  107
[2,]  106  108

, , 3

     [,1] [,2]
[1,]  109  111
[2,]  110  112
```

이 배열을 잘 살펴보면 각 차원 내의 행렬이 모두 열의 방향으로 구성요소들의 값이 나열됨을 알 수 있습니다. 따라서 7번째의 값은 2차원내의 1행 2열에 존재할 것입니다.

```
> A[7]
[1] 107
> A[1,2,2]
[1] 107
```

이제부터 2차원에 있는 값들만 뽑아봅니다.

```
> A[,,2]
     [,1] [,2]
```

```
[1,]  105  107
[2,]  106  108
```

그럼 차원에 관계없이 1행에 해당하는 값들만 뽑아봅니다.

```
> A[1,,]
     [,1] [,2] [,3]
[1,]  101  105  109
[2,]  103  107  111
```

마지막으로 차원에 관계없이 2행 1열에 해당하는 값들만 뽑아보겠습니다.

```
> A[2,1,]
[1] 102 106 110
>
```

TODO:

- `is.matrix()` 에 대한 설명 필요
- 연습문제가 제공되어지면 좋을 것 같음.

1.6 숫자말고 문자형, 그리고 모드

여기까지 우리는 기초사용에 익숙해지기 위해서 숫자값들만을 이용하여 벡터, 행렬, 그리고 배열의 기초적인 조작법을 살펴보았습니다. R은 통계분석에 필요한 프로그래밍을 위한 보다 다양한 형태의 데이터 형식을 가지고 있습니다. 이들은 요인(factor), 데이터 프레임(data frame), 리스트(list) 라는 것이 있는데 이들에 대해서는 "데이터형과 조작" 이라는 챕터를 살펴보시길 바랍니다.

여기에서는 "데이터형과 조작" 이라는 챕터에서 다루어질 데이터들의 종류에 대해서 한가지만 더 알고 넘어가도록 하겠습니다. 그것은 바로 문자형입니다. 문자형이란 1,2,3,...과 같은 숫자들이 아닌 a,b,c, ... 와 같은 기호를 의미합니다.

R은 숫자와 문자라는 것을 사용자가 직접 알려주기 전까지는 알 수 없습니다. 따라서, 문자형인 데이터를 사용할 때에는 꼭 큰따옴표를 이용하여 "a", "b", "c", ... 와 같은 형식으로 사용해야 합니다.

문자형 구성요소 "a" "b" "c" "A" "B" "C"로 이루어진 벡터를 한 번 만들어 봅니다.

```
> a <- c("a", "b", "c", "A", "B", "C")
> a
[1] "a" "b" "c" "A" "B" "C"
```

이렇게 벡터로 잘 입력이 되었다면 이들의 값을 얻는 방법은 숫자형으로 이루어진 벡터에서 인덱싱하는 방법과 동일합니다. 세번째 문자를 뽑으려면 아래와 같이 합니다.

```
> a[3]
[1] "c"
```

간혹 대문자 "A"부터 "Z"까지 생성해보고 싶다면 아래와 같이 LETTERS라는 이미 주어진 벡터를 활용하세요. 이와 같이 이미 주어진 것들에 대한 것을 내장되어 있다고 합니다.

```
> a <- LETTERS
> a
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
> a[21:25]
[1] "U" "V" "W" "X" "Y"
>
```

지금 본 섹션에서는 숫자형 데이터를 다루지 않고 문자형 데이터를 다룹니다. 그럼 만약 어떤 객체가 주어졌을때 이 객체가 숫자형인지 문자형인지 어떻게 알 수가 있는 것일까요? 아래와 같이 is 계열의 함수를 통하여 확인할 수 있습니다.

```
> is.numeric(a)
[1] FALSE

> is.numeric(x)
[1] TRUE

> is.character(a)
[1] TRUE
```

그럼, 실제로 이 데이터들은 어떻게 저장되는 것인가요? x 라는 것은 숫자형이므로 numeric 이고, a는 문자형이므로 character 라고 합니다. 즉, 객체의 유형을 mode()라는 함수를 통하여 알 수 있습니다.

```
> mode(x)
[1] "numeric"

> mode(a)
[1] "character"
```

문자형의 경우에는 그 유형은 character 라는 형식으로 저장할 수 있으나, 수치형일 경우에는 조금 더 세분화 됩니다. 위에서 우리는 x라는 벡터가 1부터 8까지 이루어진 정수형 벡터를 활용했습니다. 따라서, x는 numeric 이라는 수치형을 가지고 있으며 정수라는 특별한 형식을 사용합니다. 이를 확인할때 typeof() 라는 함수를 사용합니다.

```
> x <- 1:8
> mode(x)
[1] "numeric"
> typeof(x)
[1] "integer"
```

이제 0부터 1사이에 0.1을 단위로 하는 수열을 생성해 본 뒤 이를 확인합니다.

```
> x1 <- seq(from=0, to=1, by=0.1)
> x1
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> mode(x1)
[1] "numeric"
> typeof(x1)
[1] "double"
```

이렇게 R은 정수형과 부동소수형에 대한 형식을 수치형이라는 클래스 내에 가지고 있습니다. 이러한 구분은 수치연산과 관련된 부분이므로 설명하지 않습니다.

TODO:

- 연습문제가 제공되어지면 좋을 것 같음.

1.7 결측값과 변수초기화

다음 챕터로 넘어가기전에 숫자형과 문자형 데이터를 다룰 때 반드시 알아두어야 할 결측치에 대한 설명을 한 뒤에 본 챕터를 마무리 합니다.

결측치라는 것은 아래와 같은 경우에 발생합니다.

벡터를 생성하려고 하는데, 사용자가 무슨 값을 생성해야 할지 모릅니다. 예를들어 1,2, 그리고... 값을 몰라서 입력을 못해서 머릿속에 ???? 인데, 그 다음의 값은 4 라는 것을 알고 있습니다. 이런 경우에 R에 값을 입력하기 위해서는 아래와 같이 합니다.

```
> x <- c(1,2,NA,4)
> x
[1]  1  2 NA  4
```

이처럼, R에서는 존재하지 않는 어떤 값 (즉, 결측치)를 NA 이라는 예약어를 이용하여 표시합니다. 여기에서 NA란 Not Available 의 약자입니다.

그렇다면 어떤 주어진 객체 x 라는 것이 있을 때 그 구성요소가 결측이라는 것을 알기 위해서는 is.na()이라는 함수를 사용하면 될 것입니다.

```
> is.na(x)
[1] FALSE FALSE  TRUE FALSE
```

문자형의 경우에도 동일합니다.

```
> a <- c("a", "b", NA, "d")
> a
[1] "a" "b" NA  "d"
> is.na(a)
[1] FALSE FALSE  TRUE FALSE
>
```

결측값이 있더라도 입력한 값이 수치형이라면 is.numeric()은 TRUE를 반환합니다. 이것은 입력한 값이 문자형이라도 마찬가지 입니다.

```
> is.numeric(x)
[1] TRUE
```

```
> is.character(a)
[1] TRUE
```

어떤 경우 (특히 수치연산시)에는 어떠한 연산의 결과를 저장해야 할 빈공간을 만들어야 할 경우가 있습니다. 이런 경우에는 상식적으로 어떤 값이 어떻게 입력될지 모르니까 NA로 만들어 넣어야겠군 이라고 생각할 수 있습니다. 그런데, 이러한 경우 R에서는 수치형인지 문자형인지에 따라 해당 공간을 만들어 주는 기능이 있습니다.

```
> x <- numeric(10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
> a <- character(10)
> a
 [1] "" "" "" "" "" "" "" "" "" ""
```

TODO:

- 강제형변환(coere)이라는 개념이 빠져있음.
- as 와 is 계열의 함수들을 알아두는 것은 추후에 사용자 정의함수를 작성시에 매우 유용함
- 논리연산자를 다룰때 꼭 알려줘야 할 것. (언제 & 를 쓰고 && 를 쓰는가?)
- 논리연산 프로그래밍 팁 – 많은 경우 == 이라는 기호를 이용해서 같은가를 물어보지만, R에서는 identical()이라는 함수를 더 많이 사용함 – 왜냐하면 가독성때문에.
- 추후에 다룰 데이터프레임이 곧 행렬에 추가적인 속성을 붙인것 뿐이므로, 행렬을 잘 다루는것이 데이터 조작에 유리하다는 것을 알려줘야 함.
- TRUE와 FALSE의 약자 쓰지 말것.
- 1:5 와 같은 벡터 생성시 seq(1,5)라고 할 수도 있는데, 이것보다는 seq_len()이라는 것이 실제로 더 많이 쓰임 - 가독성 때문에.
- 수치연산 파트에서 integer, double 형 구분하는 법과 .Machine 값 찾는 내용 작성해주기.
- rownames(), colnames()에 대해서 알려줄 것 – 추후에 데이터프레임에서는 row.names()와 col.names() – 두가지 엄청 헷갈리니 조심해야 함.
- 이와 유사하게 dim()과 dimnames()에 대해서도 반드시 설명할 것.