기본연산

Seokjin Woo

R과 빅데이터에 대한 기본 소개

개요

- 데이터 사이언스는 실제 자료를 이해할 수 있고, 인사이트가 있는, 그리고 지식도 전달하는 수단임
- 이번 코스는 데이터 사이언스 분야에서 Python과 더불어 가장 많이 사용되고 있는 R을 통해서 데이터 사이언스를 배우는 것임
- 특히, 경제분야 모델을 세우고 추정하는 방법에 대해서 집중적으로 배우게 될 것임

빅데이터: the new ‘The Future’

- 데이터가 돈이 된다. 새로운 유전이다.
- 디지털 뉴딜
- 3V’s: **V**olume, **V**ariety, **V**elocity

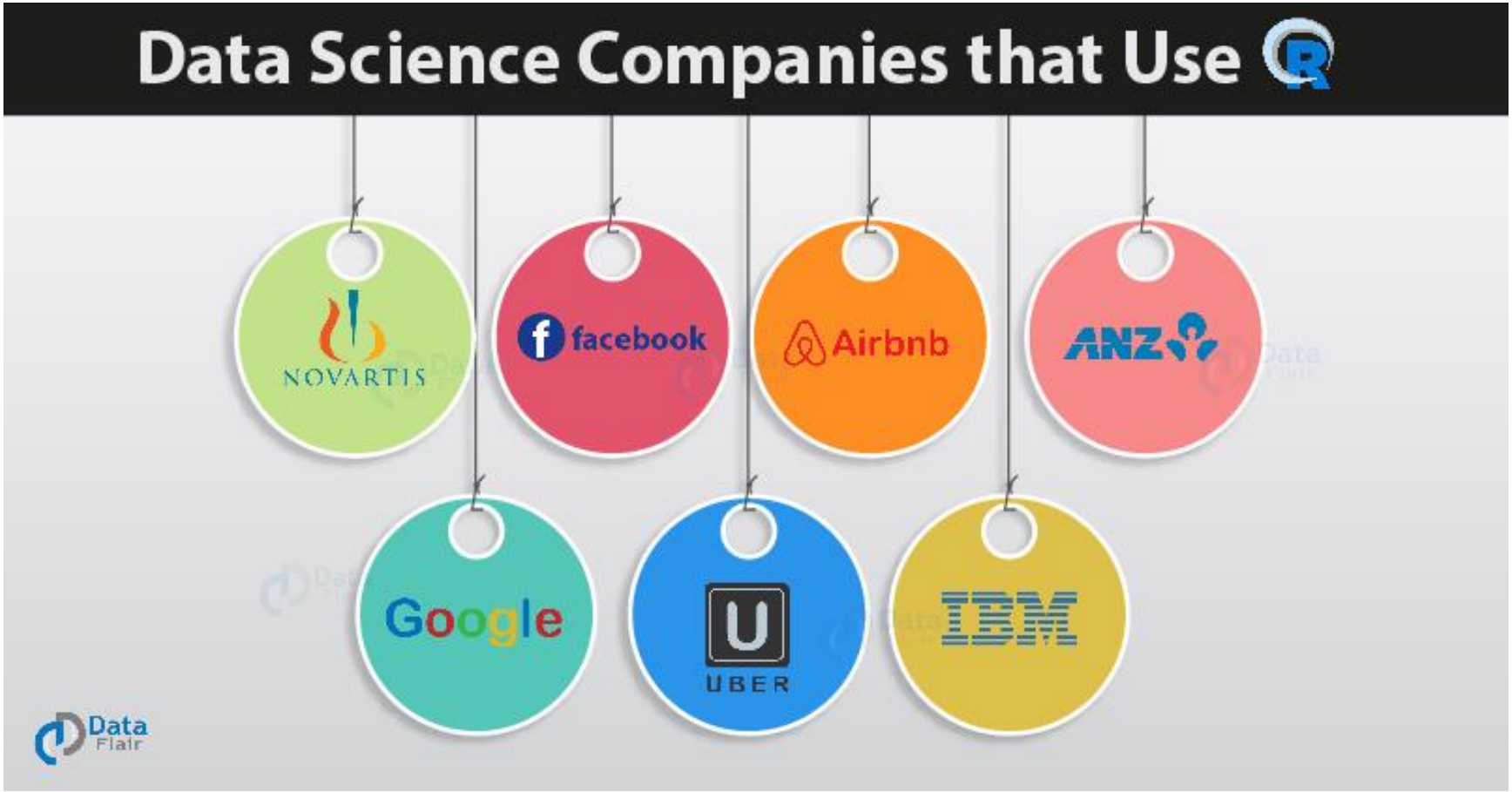
Why R

- developed by **R**oss Ihaka and **R**obert Gentleman
- 통계 및 시각화 부문에서 널리 사용되고 있음
- 10,000 개 이상의 패키지(CRAN repository)
- 많은 통계, 계량경제학자가 사용
- 시각화 우수

R의 장점

- data wrangling (dplyr, purrr, readxl,…)
- 다양한 통계모형
- 시각화 우수(ggplot2, scatterplot3D)
- 머신러닝, 딥러닝을 할 수 있는 다양한 패키지

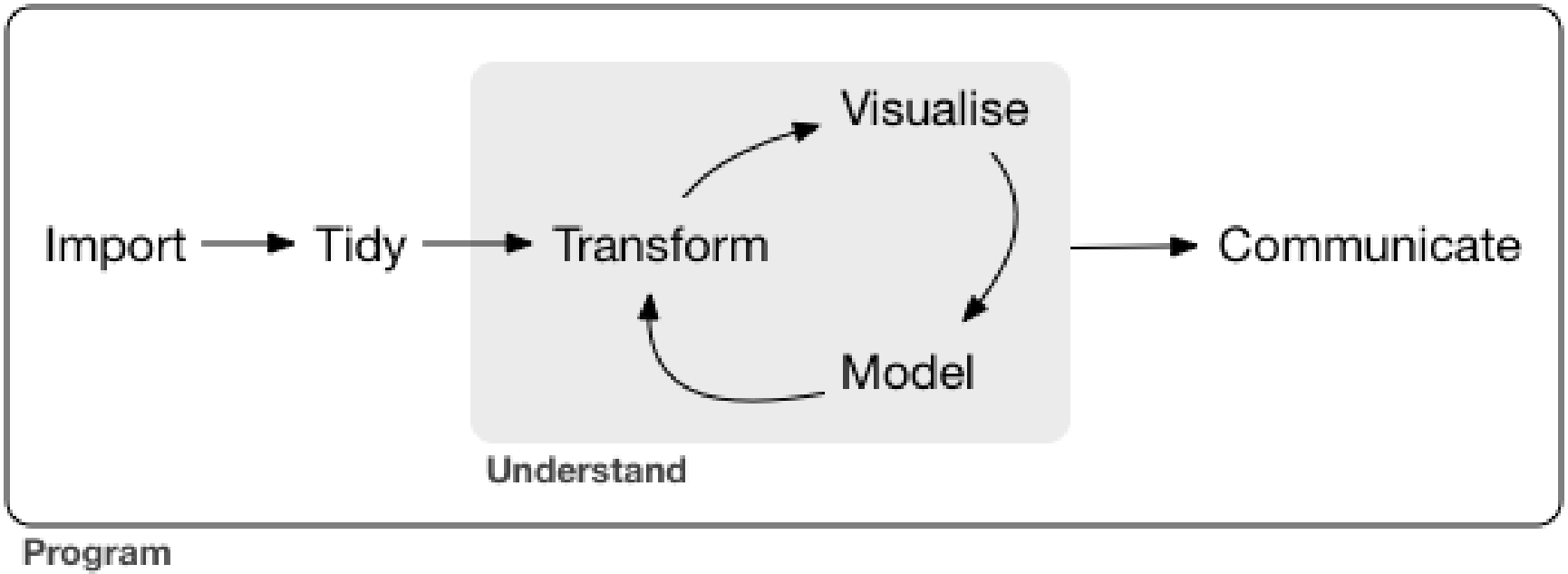
R을 사용하는 회사들


Data Science Companies that Use
NOVARTIS
facebook
Airbnb
ANZ
Google
UBER
IBM
Data Flair

R에 대한 소개

- R은 컴퓨터 랭귀지, 자료의 조작, 통계, 시각화에 사용
- 기본 R에는 다양한 통계 및 그래픽 분석을 위한 툴들이 포함되어 있고, 사용자들이 개발한 패키지들을 추가시킬 수 있음
- C, C++, FORTRAN 같은 언어도 효율적으로 포함시켜 사용할 수 있음

기본적인 흐름


Visualise
Import
Tidy
Transform
Communicate
Model
Understand
Program

R을 이용한 기본 연산

기본 연산 1

- R은 기본적으로 계산기 기능을 제공
- 4칙 연산을 기본으로 할 수 있고, 제곱, 세제곱 같은 연산도 할 수 있음
- 예컨대 덧셈은 아래와 같이 할 수 있다. 콘솔 창에 1+1을 치고 enter 키를 누르면 다음과 같은 연산 결과를 얻을 수 있음

```
> 1+1
[1] 2
> 2+2
[1] 4
```

기본 연산 2

- 뺄셈도 덧셈과 마찬가지 방법으로 할 수 있다.

```
> 5-3
[1] 2
```

- 곱셈은 연산자 *을 이용하여 할 수 있다.

```
> 2*3
[1] 6
```

- 나눗셈은 /을 이용하여 연산할 수 있다.

```
> 4/2
[1] 2
```

기본 연산 3

- 거듭제곱은 숫자 6에 붙어 있는 ^을 이용하여 구할 수 있다. 예를 들어 2의 3제곱근은 아래와 같이 구할 수 있다.

```
> 2^3
[1] 8
```

- 간단한 함수 연산도 할 수 있음
- 밑이 10인 $\log_{10}10$ 의 값은 아래와 같이 구할 수 있다.

```
> log(10, base = 10)
[1] 1
```

기본 연산 4

- 밑이 특별히 지정되지 않은 경우에는 자연로그가 계산된다.

```
> log(10)
[1] 2.302585
```

- 지수함수는 다음과 같이 계산할 수 있다.

```
> exp(1)
[1] 2.718282
> exp(log(10))
[1] 10
```

벡터

- R에서 가장 자주 사용되는 기본 단위는 벡터(vector)이다.
  - 나중에 소개할 함수들도 대부분 벡터 기반으로 만들어진 경우가 많다.
- 벡터는 c()함수를 통해서 만들어진다.
  - 여기에서 c는 영어 “combine” 의 약자로 알려져 있다.
- 다음과 같이 벡터를 만들 수 있다.

```
> c(1, 2, 3, 4, 5)
[1] 1 2 3 4 5
```

벡터 2

- 이렇게 만들어진 벡터를 특정 변수에 저장할 수 있다.
- 저장하는 방법은 " <-" 혹은 “=” 을 사용하면 된다.
  - <- 은 오른쪽 값을 왼쪽 변수에 넣으라고 알려주는 기호이다.
- 앞에서 만든 벡터를 변수 a에 아래와 같이 담을 수 있다.

```
> a <- c(1, 2, 3, 4, 5)
```

- 이렇게 만들어진 벡터는 `print()` 를 이용해서 살펴볼 수 있다.

```
> print(a)
[1] 1 2 3 4 5
```

벡터 3

- 벡터의 길이는 length()를 이용해서 살펴볼 수 있다.

```
> length(a)
[1] 5
```

- 이렇게 생성된 벡터 a를 이용하여 스칼라 변수처럼 다양한 연산을 할 수 있다.
- 아래 연산은 아래의 벡터값에 2를 곱하고 3을 더하는 연산으로서 결과는 다음과 같다.

```
> 2*a + 3
[1]  5  7  9 11 13
```

벡터 4

- 벡터에 함수를 적용할 수도 있다.
- 예컨대, $log(a)$ 를 하면 벡터값마다 자연로그를 취한 값들이 산출된다.

```
> a
[1] 1 2 3 4 5
> log(a)
[1] 0.0000000 0.6931472 1.0986123 1.3862944
1.6094379
```

부분벡터

- 전체 벡터가 아니라 벡터의 일부분만 필요한 경우가 있다.
- 이런 경우 벡터의 일부분인 부분벡터를 추출할 필요가 있다.
- 벡터 a의 1, 3 번째 값을 다음과 같이 추출할 수 있음

```
> a
[1] 1 2 3 4 5
> a[c(1, 3)]
[1] 1 3
```

부분벡터 2

- 해당 index에 마이너스 부호를 붙이면, 해당 값만 제외하고 추출할 수 있다.
- 예를 들어, 인덱스 a[-c(1,3)]는 a 벡터의 값 중 1, 3 번째 값만 제외하고 추출하라는 뜻이다.

```
> a[-c(1,3)]
[1] 2 4 5
```

패턴이 일정한 벡터

- 계량분석을 하다 보면 특정한 패턴을 가진 벡터가 필요한 경우가 있다.
- 이런 경우 rep(), seq() 등을 이용하면 쉽게 반복되는 패턴을 가진 벡터를 생성할 수 있다.
- 다음과 같이 1로만 구성된 일벡터를 만들 수 있다.

```
> ones <- rep(1, 10)
> print(ones)
 [1] 1 1 1 1 1 1 1 1 1 1
```

패턴이 일정한 벡터 2

- seq() 함수를 이용하면 수열로 구성된 벡터를 생성할 수 있다.
- 1부터 100까지 5씩 증가하는 수열인, 1, 6, 11,…, 91, 96 으로 구성된 벡터가 만들어진다.

```
> b <- seq(from = 1,
+          to   = 100,
+          by   = 5)
> print(b)
 [1]  1  6 11 16 21 26 31 36 41 46 51 56 61 66 71
76 81 86 91 96
```

패턴이 일정한 벡터 3

- 알파벳의 열

```
> letters[1:3]
[1] "a" "b" "c"
```

- 시계열의 경우에는 연도 변수를 만들 필요가 있다.

```
> year <- 2000:2018
> print(year)
 [1] 2000 2001 2002 2003 2004 2005 2006 2007 2008
2009 2010 2011 2012 2013 2014
[16] 2015 2016 2017 2018
```

- 벡터를 이어 붙일 수 있다.

```
> c(ones, b)
 [1]  1  1  1  1  1  1  1  1  1  1  1  6 11 16 21
26 31 36 41 46 51 56 61 66 71
```

행렬(MATRIX)

행렬

- 행렬은 벡터의 모음이라고 보면 된다.
- 1부터 12까지의 수열 벡터를 생성한 후 nrow = 4를 지정하면 4X3 행렬을 아래와 같이 만들 수 있다.

```
> A <- matrix(1:12, nrow = 4)
> print(A)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

행렬 2

- 동일한 수열에 nrow = 3 으로 바꾸면 3X4 행렬을 생성할 수 있다.

```
> B <- matrix(1:12, nrow = 3)
> print(B)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> Bp <- matrix(1:12, nrow = 3, byrow = TRUE)
> print(Bp)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

행렬 3

- 행렬 A의 차원은 다음과 같다

```
> dim(A)
[1] 4 3
```

- 행의 갯수를 알고 싶으면 다음과 같이 확인할 수 있다.

```
> nrow(A)
[1] 4
```

- 열의 갯수를 알고 싶으면 다음과 같이 확인할 수 있다.

```
> ncol(A)
[1] 3
```

행렬 4

- 행렬 A의 인자 a_{ij}는 A[i,j]로 추출할 수 있다.
- 전체 행은 A[i,], 전체 열은 A[, j]로 표시할 수 있다.

하부행렬

- 행렬 A는 아래와 같다.

```
> print(A)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

하부행렬 2

- 위 행렬 A에서 1,2 행과, 1,3 열을 추출하여 하부 행렬 A1을 아래와 같이 생성하자.

```
> A.1 <- A[c(1,2),c(1,3)]
> print(A.1)
     [,1] [,2]
[1,]    1    9
[2,]    2   10
```

- A1은 첫 번째와 세 번째 행 및 열을 추출해서 만든 정방행렬(square matrix)이다.

하부행렬 3

- 행렬식(determinant)을 계산해보자.
- 함수 det()를 이용하면 행렬식을 계산할 수 있다.

```
> det(A.1)
[1] -8
```

역행렬

- 역행렬은 함수 solve()를 이용하면 구할 수 있다.
- A.1의 역행렬 A.i은 다음과 같다.

```
> A.i <- solve(A.1)
> print(A.i)
          [,1]   [,2]
[1,] -1.25  1.125
[2,]  0.25 -0.125
```

역행렬 2

- 이렇게 구해진 역행렬이 제대로 구해졌는지 확인하려면 원래 행렬 A.1과 곱하면 단위행렬(identity matrix)이 되는지 확인해보면 된다.
  - 행렬의 곱은 "%*%" 이다.

```
> A.1 %*% A.i
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

- 위 행렬 곱은 순서를 바꿔도 결과는 동일하다.

```
> A.i %*% A.1
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

패턴이 있는 행렬

- 경제 분석을 위한 계량분석에서는 일정한 패턴을 가지고 있는 행렬이 자주 사용된다.
- R에서는 이러한 패턴을 가진 행렬을 비교적 쉽게 생성할 수 있다.
- 대각선에 위치한 값이 1이고, 대각선에 위치하지 않는 값은 0인 대각 행렬은 함수 diag()를 이용하면 생성할 수 있다.

패턴이 있는 행렬

```
> D <- diag(1, 5, 5)
> print(D)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

패턴이 있는 행렬 2

- 좀 더 일반적으로는 다음과 같이 대각행렬을 생성할 수 있다.
- 먼저, 다음과 같이 벡터를 만들어보자. (1,2)값을 갖는 벡터를 3번 연속해서 생성된다.

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

- 이번에는 반복되는 값을 아래와 같이 벡터로 지정해보자.

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

패턴이 있는 행렬 2

- 위의 값을 대각선 값으로 갖는 대각행렬은 다음과 같이 생성할 수 있다.

```
> diag(rep(c(1, 2), c(3, 3)))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    0    1    0    0    0    0
[3,]    0    0    1    0    0    0
[4,]    0    0    0    2    0    0
[5,]    0    0    0    0    2    0
[6,]    0    0    0    0    0    2
```

패턴이 있는 행렬 3

- 위의 값을 대각선 값으로 갖는 대각행렬은 다음과 같이 생성할 수 있다.

```
> diag(rep(c(1, 2), c(3, 3)))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    0    1    0    0    0    0
[3,]    0    0    1    0    0    0
[4,]    0    0    0    2    0    0
[5,]    0    0    0    0    2    0
[6,]    0    0    0    0    0    2
```

패턴이 있는 행렬 3

- diag()의 경우, 행렬이 들어가면 해당 행렬의 대각행렬의 값을 추출할 수 있다.
- 예컨대 행렬 A.1은 아래와 같은 행렬이다.

```
> A.1
     [,1] [,2]
[1,]    1    9
[2,]    2   10
```

- A.1의 대각선에 위취한 값들은 1과 10임을 확인할 수 있다.

```
> diag(A.1)
[1]  1 10
```

하삼각행렬 상삼각행렬

- 하삼각행렬(lower triangular matrix), 상삼각행렬(upper triangular matrix)도 구할 수 있다.

하삼각행렬

- 하삼각행렬은 반대로 upper.tri() 함수를 이용하면 된다.
- upper.tri()는 상삼각에 속하는 경우를 지칭해주는 논리값(TRUE 혹은 FALSE)을 준다.
- 따라서 상삼각에 속하는 값을 0으로 바꾸면 하삼각행렬을 만들 수 있다.

```
> upper.tri(A.1)
      [,1]  [,2]
[1,] FALSE  TRUE
[2,] FALSE FALSE
```

하삼각행렬 2

- 위 값에 속하는 경우는 0으로 바꾸어주면, 하상각 행렬이 만들어진다.

```
> A.L <- A.1
> A.L[upper.tri(A.L)] <- 0
> A.L
     [,1] [,2]
[1,]    1    0
[2,]    2   10
```

상삼각행렬

- 유사한 방법으로 상감각 행렬도 만들 수 있다.

```
> A.U <- A.1
> A.U[lower.tri(A.U)] <- 0
> A.U
     [,1] [,2]
[1,]    1    9
[2,]    0   10
```

rbind()와 cbind()

- 한편 여러 개의 행렬 또는 벡터를 하나의 행렬로 묶을 수 있다.
- 보통 rbind(), cbind() 함수를 사용하게 된다.
- rbind()는 행으로 묶고, cbind()는 열로 묶게 된다.

```
> cbind(1, A.1)
     [,1] [,2] [,3]
[1,]    1    1    9
[2,]    1    2   10
```

rbind()와 cbind() 2

- 행으로 묶을 때는 rbind()를 이용하면 된다.

```
> rbind(1, A.1)
     [,1] [,2]
[1,]    1    1
[2,]    1    9
[3,]    2   10
```

R에서 프로그래밍 하기

벡터의 모드(mode)

- R에서 가장 단순한 자료 형태는 앞에서 소개한 벡터이다.
- 한 가지 주의해야 할 점은 벡터안에 포함된 값들은 모두 동일한 모드(mode)을 가지고 있어야 한다.
- 모드는 크게 보면, 숫자(numeric), 문자(character), 논리(logical)로 구분해볼 수 있다.
- 예컨대 벡터 x가 다음의 값을 가지는 벡터라고 해보자.

```
> x <- c(1:5)
> print(x)
[1] 1 2 3 4 5
```

numeric

- 벡터 x는 기본적으로 $1,2,\ldots,5$의 숫자를 갖고 있는 숫자벡터이다.
- x가 어떤 모드인지는 mode() 함수를 통해서 아래와 같이 확인할 수 있다.

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

- 위에 따르면 x는 numeric이다.

character

- character 벡터에는 문자가 들어간다.
- a,b,c,d,e 의 값을 갖고 있는 벡터이다.
- 문자 벡터의 값은 쌍따옴표 "" 에 들어가 있다.

```
> y <- c("a", "b", "c", "d", "e")
> print(y)
[1] "a" "b" "c" "d" "e"
```

- 모드는 mode() 함수를 통해 알 수 있다.
- 벡터 y는 character 임을 확인할 수 있다.

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

character 2

- 이런 문자 벡터는 다양한 용도로 사용이 가능하다. 자료 그 자체로 사용할 수도 있고
- 벡터나 행렬의 레이블로서도 사용이 가능하다.
- 예컨대, 다음과 같이 names()함수를 이용하여 위에서 생성한 벡터 x의 이름으로도 사용가능하다.

```
> names(x) <- y
> print(x)
a b c d e
1 2 3 4 5
```

logical

- 다음으로는 논리(logical) 벡터가 있다.
  - 논리 벡터의 경우 벡터의 특정값이 특정 조건을 만족시키는 지를 나타내주는 벡터이다.
  - 벡터의 값들이 조건을 만족시키면 TRUE, 만족시키지 못하면 FALSE 의 값을 갖는다.
- 앞에서 생성했던 벡터 x 의 경우 $x > 3$의 조건을 만족시키는지 여부를 이용하여 논리 벡터를 만들어보자.

```
> z <- x>3
> print(z)
    a     b     c     d     e
FALSE FALSE FALSE  TRUE  TRUE
```

logical 2

- z의 모드 값은 mode() 함수를 이용해 확인해보면 그 값은 logical임을 알 수 있다.

```
> mode(z)
[1] "logical"
```

LIST

리스트(list)

- list는 데이터를 보관할 수 있도록 기존의 벡터를 확장한 또 하나의 벡터라고 이해할 수 있다.
- 기존 벡터는 단순히 값만 가지고 있다면, list에는 그 외의 정보들도 포함되어 있다.
- list 는 다양한 모드의 벡터, 행렬, 데이터 프레임(data frame), 함수, 혹은 list 자체를 값으로 가질 수 있다.
- R에서 다루어지는 많은 자료들은 list의 형태로 저장되고 있다.

list 2

- 다음과 같은 예를 통해 list를 살펴보자.
  - 표준정규분포를 따르고 있는 표본을 생각해보자.
  - 벡터를 이용해 표본을 생성해보자.
  - 표준정규분포를 따르는 난수를 생성하기 위해서는 rnorm() 함수를 사용하면 된다.
  - set.seed()는 난수 생성 전에 시드를 정해주는 함수이다.
- rnorm(10)은 표준정규분포를 따르는 10개의 난수를 추출해준다.
- 이렇게 추출된 난수를 list.normal 이라는 벡터에 저장을 한다.
- 그리고 print() 를 이용하여 어떤 난수가 생성되었는지 확인을 한다.

list 3

```
> set.seed(123456)
> list.normal <- rnorm(10)
> print(list.normal)
 [1]  0.83373317 -0.27604777 -0.35500184
0.08748742  2.25225573  0.83446013
 [7]  1.31241551  2.50264541  1.16823174 -
0.42616558
```

list 4

- 벡터로 생성된 list.normal에는 위에서 생성된 값만 담기게 된다.
- 값 외에도 좀 더 많은 정보가 필요할 경우가 있다.
  - 예컨대, 난수를 어떤 분포에서 이 표본을 생성했는지, 분포의 모수는 어떤 것들을 사용했는지 등에 대한 정보가 필요할 수 있다.
  - 이런 경우 다음과 같이 list()를 이용하면 된다.

```
> list.normal <- list(sample = rnorm(10),
+                     dist   = "normal",
+                     param  = list(mean = 0,
+                                   sd   = 1)
+                     )
```

list 5

- 위 list.normal은 크게 보면 sample, dist, param 으로 구성되어 있다.
  - sample은 10개의 난수를 담고 있는 numeric이다.
  - dist는 어떤 분포에서 난수가 생성되었는지를 기록하고 있는 character 이다.
  - 구체적으로는 “normal”이라고 기록을 해 놓아 나중에 난수가 정규분포에서 추출되었음을 알 수 있다.
  - param은 다시 list 이다. 평균은 0, 표준편차는 1이라고 기록해 두었다.
- list.normal은 다음과 같다.

list 6

```
> print(list.normal)
$sample
 [1] -0.99612975 -1.11394990 -0.05573154
1.17443240  1.05321861  0.05760597
 [7] -0.73504289  0.93052842  1.66821097
0.55968789

$dist
[1] "normal"

$param
$param$mean
[1] 0

$param$sd
[1] 1
```

list 7

- 위 list의 각 요소를 추출하기 위해서는 “$”를 이용하면 된다.
- 예컨대, sample은 다음과 같이 추출할 수 있다.

```
> list.normal$sample
 [1] -0.99612975 -1.11394990 -0.05573154
1.17443240  1.05321861  0.05760597
 [7] -0.73504289  0.93052842  1.66821097
0.55968789
```

- dist 역시 동일한 방식으로 추출할 수 있다.

```
> list.normal$dist
[1] "normal"
```

list 7

- 한 걸음 더 나아가서 param 리스트 내에 mean 만 따로 추출할 수 있다.
  - “$”를 추가적으로 더해주면 된다.

```
> list.normal$param$mean
[1] 0
```

- sd도 유사하게 추출할 수 있다.

```
> list.normal$param$sd
[1] 1
```

논리 비교

- 논리적 비교는 자료 분석에서 매우 중요하다.
  - 예컨대, 성별, 학력, 경력 별로 임금을 계산할 때 일정 조건을 만족시키는 자료만을 추출할 필요가 있다.
  - 기본적으로 <, <=, >, >=, ==, != 을 사용할 수 있다.
  - 두 개의 조건이 동시에 만족해야 할 때는 “&”를, 하나의 조건만 만족해야 할 때에는 “|”를 사용하면 된다.
  - 조건의 반대는 “!”를 사용할 수 있다.

논리 비교 2

- 다음과 같은 벡터를 생성해보자. x는 표준정규분포를 따르는 10개의 난수로 구성된 벡터이다.

```
> set.seed(123456)
> x <- c(rnorm(10))
> print(x)
 [1]  0.83373317 -0.27604777 -0.35500184
0.08748742  2.25225573  0.83446013
 [7]  1.31241551  2.50264541  1.16823174 -
0.42616558
```

논리 비교 3

- 다음과 같은 $0 < x < 1.0$조건을 만족시키는지 여부를 확인할 수 있다.
- 그러면 4번째와 9 번째 값이 조건에 해당하여 TRUE 값을 갖고, 나머지 값은 조건을 만족하지 않아 FALSE 값을 갖게 된다.

```
> x > 0  & x <1.0
 [1]  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE
FALSE FALSE FALSE
```

논리 비교 4

- 이런 작업을 좀 더 쉽게 할 수 있게 해주는 함수는 which() 함수이다

  .

- 앞의 분석과 동일하게 1, 4, 6 번째 벡터값이 $0 < x < 1.0$조건을 만족시키고 있음을 알 수 있다.

```
> which(x > 0  & x < 1.0)
[1] 1 4 6
```

타입의 변환

- R에서는 변수 혹은 벡터의 타입을 강제로 전환해줄 수 있다.
- 그 전에 벡터가 어떤 타입인지 확인해보자. is.foo() 의 형태의 함수를 통해서 확인할 수 있다.
- 예를 들어, is.numeric(x) 를 통해서 numeric 인지 여부를 확인할 수 있다.

타입의 변환 2

- 아래 x는 numeric 인지 물어보았을 때 답은 TRUE 이다.

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

- 반면, character인지 확인해보자. 답은 FALSE 이다.

```
> is.character(x)
[1] FALSE
```

타입의 변환 2

- 벡터 x의 경우 numeric 이지만 character 로 강제로 전환할 수 있다.
  - as.foo() 형태의 함수를 이용하면 된다.
- 예를 들면, as.character()를 사용하면 속성을 character로 전환할 수 있다.

```
> x.c <- as.character(x)
> print(x.c)
 [1] "0.833733170755138"  "-0.276047773214867" "-0.355001838033406"
 [4] "0.087487423804203" "2.2522573054322"  "0.834460129133672"
 [7] "1.3124155085886"   "2.5026454078906"  "1.168231743384 41"
[10] "-0.426165577414356"
```

타입의 변환 3

- is.character()를 이용해서 확인해보자.
- 새로 만든 x.c 는 character 타입임을 알 수 있다.

```
> is.character(x.c)
[1] TRUE
```

타입의 변환 4

- 벡터에 서로 다른 두 타입의 값이 포함된 경우에는 하나의 타입으로 강제 배정된다.
- 예컨대, 벡터 값이 1과 “a”의 경우 모두 character로 전환된다.

```
> c(1,"a")
[1] "1" "a"
```

타입의 변환 4

- 논리값과 숫자가 묶였을 때, numeric으로 전환된다.
  - 이 때 TRUE의 경우는 1의 값을 갖게 된다.

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

- 한편, 논리 변수와 문가 엮였을 때에는 character로 강제로 배정된다.

```
> c(1<0, "a")
[1] "FALSE" "a"
```

난수의 생성

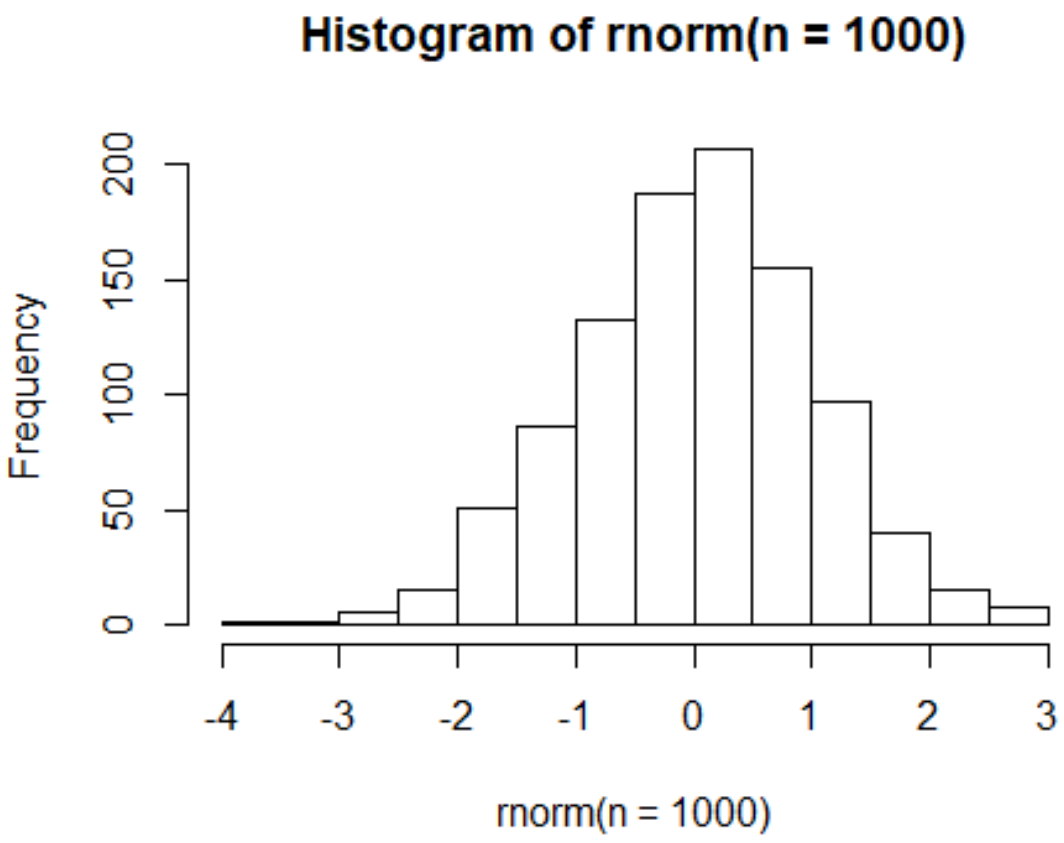
- R에서의 난수 생성은 난수생성기(random number generator)를 이용하여 생성한다.
- 다양한 분포로부터 난수를 뽑을 수 있도록 다양한 함수를 제공하고 있다.
- 난수 생성할 때 seed를 정하면 동일한 난수가 생성된다. - 만약 seed를 정하지 않으면 난수 생성할 때마다 서로 다른 난수가 생성된다.
- 난수 생성 이전에 set.seed() 함수를 이용하여 seed를 고정하자.

```
> set.seed(123456)
```

정규분포에서 추출

- 10개의 난수를 표준정규분포로부터 추출하기 위해서 rnorm() 함수를 이용해보자.
- 표준정규분포는 평균 0, 표준편차 1을 갖는 분포이다.

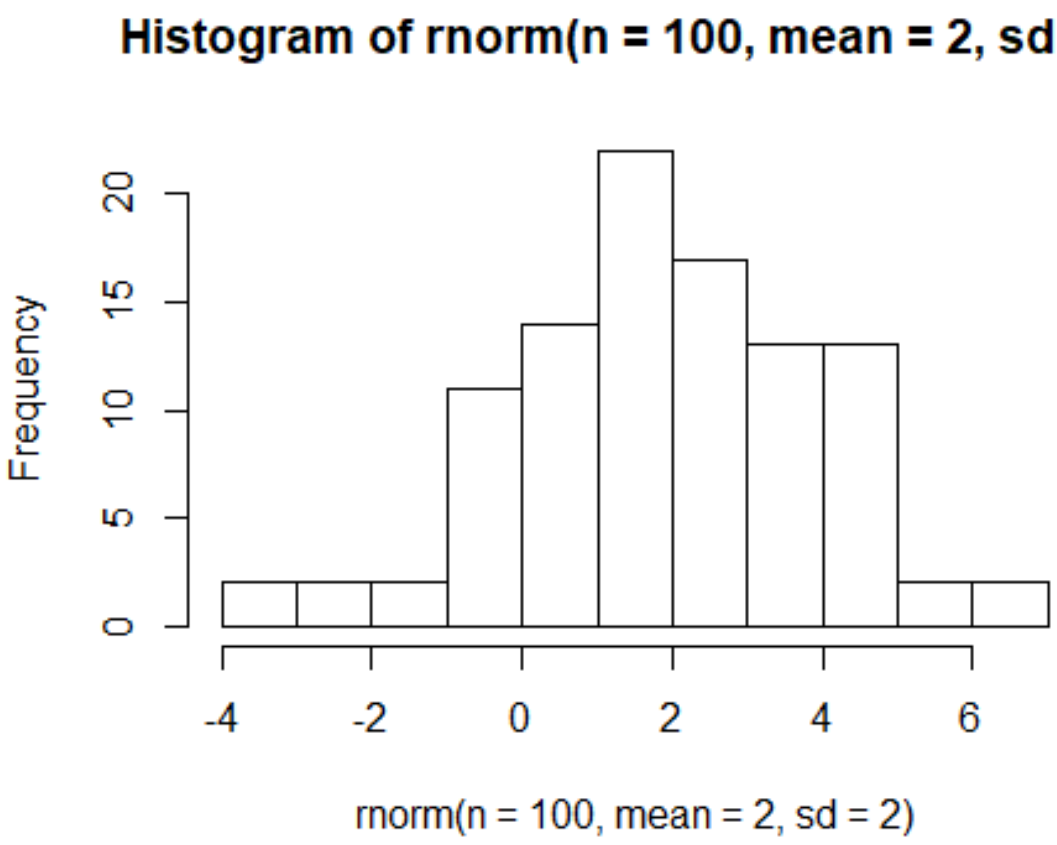
```
> hist(rnorm(n = 1000))
```


Histogram of rnorm(n = 1000)
Frequency
0
50
100
150
200
-4
-3
-2
-1
0
1
2
3
rnorm(n = 1000)

정규분포에서 추출 2

- 평균이 2, 표준편차 2인 정규분포를 따르는 난수 10개를 생성해보자.
- rnorm() 함수에 mean = 2, sd = 2 라는 모수를 정해주면 된다.

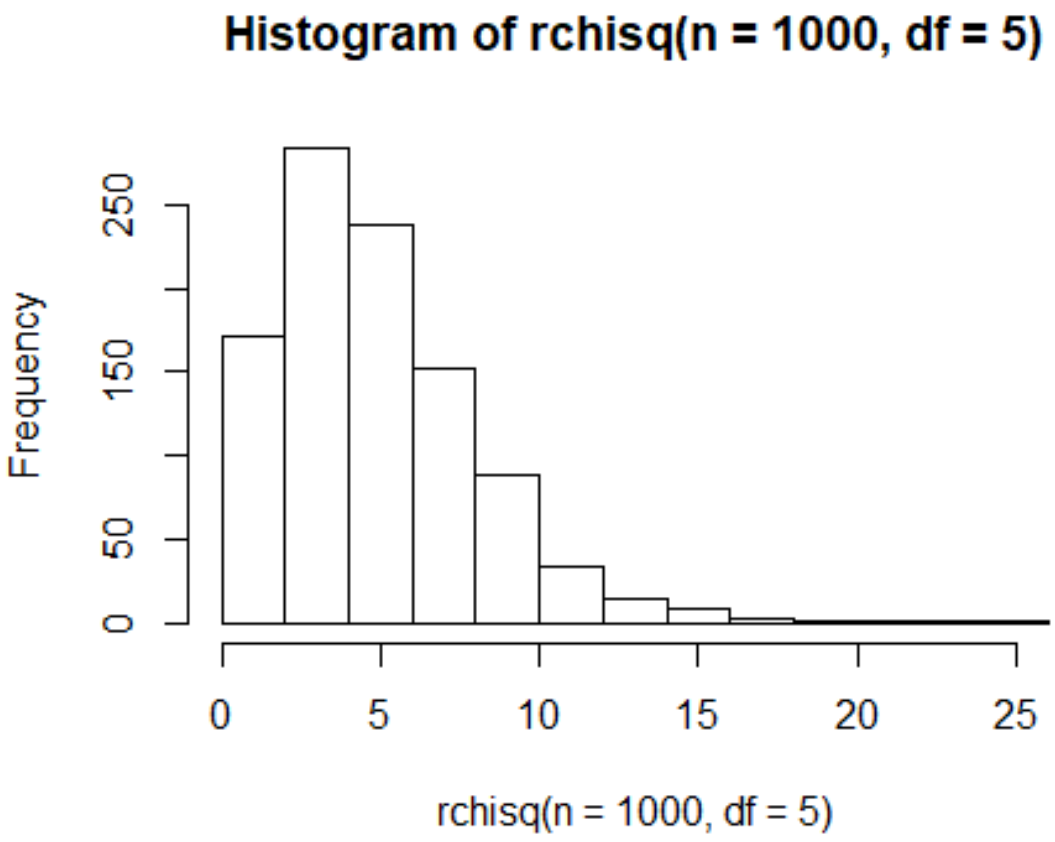
```
> hist(rnorm(n = 100, mean = 2, sd = 2))
```


Histogram of rnorm(n = 100, mean = 2, sd = 2)
Frequency
0
5
10
15
20
-4
-2
0
2
4
6
rnorm(n = 100, mean = 2, sd = 2)

카이제곱

- $\chi^2(5)$의 분포에서도 난수를 추출할 수 있다. rchisq() 함수를 사용하면 된다.
  - 단 자유도 df 를 정해주어야 한다.
  - 여기에서는 df = 5 라고 정하였다.

```
> hist(rchisq(n = 1000, df = 5))
```


Histogram of rchisq(n = 1000, df = 5)
Frequency
0
50
150
250
0
5
10
15
20
25
rchisq(n = 1000, df = 5)

sampling

- 주어진 자료에서 표집을 할 수도 있다.
- 예컨대 다음과 같은 표본이 있다고 해보자. 표본 s는 1부터 10까지의 정수이다.

```
> s <- c(1:10)
> print(s)
 [1]  1  2  3  4  5  6  7  8  9 10
```

sampling 2

- sample() 함수를 이용하면 표집을 할 수 있다.
- 모집단을 s 로 하고 5개를 표집하자. size = 5를 추가하면 된다.

```
> set.seed(123456)
> sample(s, size = 5)
[1] 8 7 4 3 9
```

sampling 3

- 여기에 이미 추출된 숫자를 복원하여 추출할 것인지, 아니면 비복원 추출을 할 지는 replace를 이용하여 정할 수 있다.
  - replace = TRUE 인 경우 복원 추출을 하게 된다. 아래 추출된 표본 중에 8이 2번 반복하여 추출되었음을 확인할 수 있다.

```
> set.seed(1234567)
> sample(s,
+        size    = 5,
+        replace = TRUE)
[1]  6  8 10  1  8
> set.seed(1234567)
> sample(s,
+        size    = 5,
+        replace = FALSE)
[1] 6 7 8 1 5
```

sampling 4

- 각 숫자들이 추출될 수 있는 확률도 지정해줄 수 있다. - 빨간색 red 와 파란색 blue 중에서 5번을 복원 추출로 추출하되,
- red가 나올 가능성은 0.7, blue가 나올 가능성은 0.3으로 지정하였다

.

```
> set.seed(1234567)
> sample(c("red", "blue"),
+        size    = 5,
+        replace = TRUE,
+        prob    = c(0.7, 0.3))
[1] "red"  "blue" "blue" "red"  "blue"
```

IF 와 FOR

if 구문과 for 루프

- R도 다른 프로그래밍 언어와 마찬가지로 if/else 구문과 for 루프를 사용할 수 있다.
- 조건문을 걸거나 반복적인 작업을 수행할 때 if/for 구문은 매우 유용하다.

if

- if 구문은 기본적으로 다음과 같은 형식으로 주어진다.

```
> if (cond) {
+   expr1
+ } else {
+   expr2
+ }
```

- 위 구문의 경우 cond에 해당하는 조건이 만족되는 경우 expr1를 실행한다.
- 만약 cond가 만족되지 않으면 expr2를 실행하게 된다.

if 2

- 다음과 같은 예를 통해서 if 구문을 살펴보자.
- 먼저, 표준정규분포를 따르는 확률변수 20개를 담고 있는 확률벡터 x.r 을 다음과 같이 생성하자.

```
> set.seed(1234567)
> x.r <- rnorm(n = 20)
> head(x.r, n = 3)
[1] 0.1567038 1.3738112 0.7306702
```

if 3

- 여기에 50%의 확률로 x.r의 합을 계산하거나 다른 50%의 확률로 평균을 계산해보자.

```
> set.seed(1)
>
> if (rnorm(1) > 0) {
+   print("The sum is ")
+   sum(x.r)
+ } else {
+   print("The mean is ")
+   mean(x.r)
+ }
[1] "The mean is "
[1] -0.01962018
```

for

- for 루프는 반목적인 계산을 하는데 유용하게 사용할 수 있다.
- 아래의 예를 통해서 for 루프를 사용해보자.
  - 여기에서 year는 순차적으로 2010, 2011 등의 값을 가지게 된다.
  - 그리고 해당 값을 가지고 루프 안으로 들어가서 해당 값을 가지고 명령을 실행하게 되는 것이다.

for 2

```
> for (year in c(2010:2015)){
+   print(paste("The year is", year))
+ }
[1] "The year is 2010"
[1] "The year is 2011"
[1] "The year is 2012"
[1] "The year is 2013"
[1] "The year is 2014"
[1] "The year is 2015"
```

for 3

- 다음 예는 for 루프와 if 구문을 섞은 것이다.
  - i는 1에서 10까지의 숫자를 가지고 순차적으로 루프 안으로 들어간다.
  - 이렇게 들어간 i를 2로 나누어서 나머지가 있으면 다음 루프로 가고
  - 나머지가 없으면 그 값을 print() 하게 된다.
  - 즉, 1에서 10 사이의 짝수를 프린트 하게 된다.

```
> for (i in 1:10) {
+   if (i %% 2){
+     next
+   }
+     print(i)
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

for 4

- 한편, 홀수는 위의 for 루프 안에서 if 구문에 반대를 나타내는 “!”를 추가해주면 된다.

```
> for (i in 1:10) {
+   if (!i %% 2){
+     next
+   }
+     print(i)
+ }
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
```

apply

- 다만, R에서 루프는 될 수 있으면 사용하지 않는 것이 바람직하다.
- R에서는 lapply 혹은 sapply와 같은 벡터화된 함수들이 있다.
- 될 수 있으면 for 루프 보다는 apply류의 함수를 사용하는 것이 속도 면에서 좋다.

apply 2

- iris 자료를 이용하여 numeric 성질을 가진 열만 추출하는 for-loop 을 짜보자.
- 먼저, iris 자료를 읽어들이자.

```
> data("iris")
```

apply 2

- iris의 열의 갯수 만큼 for 루프를 돌린다.
  - iris 의 각 열의 타입이 numeric인 경우 빈 벡터인 iris_num에
  - 해당 열을 cbind()를 이용하여 옆으로 붙여 나간다.
  - 이렇게 해서 완성된 행렬은 data.fram() 함수를 이용해서 데이터 프레임으로 선언해주자.
  - 그러면 iris_num은 numeric 컬럼들로만 구성된 데이터 프레임이 된다.

```
> iris_num <- NULL
> for( i in 1:ncol(iris)){
+   if(is.numeric(iris[, i])) {
+     iris_num <- cbind(iris_num, iris[, i])
+   }
+ }
>
> iris_num <- data.frame(iris_num)
```

```
> head(iris_num, n= 3)
   X1  X2  X3  X4
1 5.1 3.5 1.4 0.2
2 4.9 3.0 1.4 0.2
3 4.7 3.2 1.3 0.2
```

apply 3

- 이와 같이 긴 for 루프는 다음과 같은 한 줄의 sapply 함수를 이용하여 다시 쓸 수 있다.
- sapply(iris, is.numeric)은 iris 의 각 열마다, 각 열이 numeric 인지 확인하는 작업을 하고, TRUE 인 경우인 열만 추출하여 iris_num 에 저장하는 것이다.

```
> iris_num <- iris[, sapply(iris, is.numeric)]
```

```
> head(iris_num)
  Sepal.Length Sepal.Width Petal.Length
Petal.Width
1          5.1         3.5          1.4
0.2
2          4.9         3.0          1.4
0.2
3          4.7         3.2          1.3
0.2
4          4.6         3.1          1.5
0.2
5          5.0         3.6          1.4
0.2
6          5.4         3.9          1.7
0.4
```

function 만들기

- 함수 function을 필요에 따라 만들어두면 프로그래밍 할 때 매우 편리하다.
- 반복적인 작업은 모두 함수로 만들어서 사용하면 좋다.

function 만들기 2

- 기본적으로 R의 함수는 다음과 같은 형태를 가지고 있다.
- 크게 보면 3개의 파트로 나누어진다.
  - 함수의 이름 function.name
  - 함수의 인자 arguments
  - 그리고 함수의 내용인 body로 구성된다.
  - body에는 함수의 산출물로서 무엇을 내보낼지에 관한 것도 포함된다.

```
function.name <- function(arguments){
        computations on the arguments
        body
        return(output)

}
```

function 만들기 3

- 다음과 같은 예를 이용해 간단한 함수를 만들어보자.

- 함수 이름은 normalize 라고 하자.
  - x라는 함수의 인자를 받는다.
  - x에서 x의 평균 mean(x)을 빼서 numerator 라는 변수에 저장한다.
  - x의 표준편차 sd(x)를 이용해서 계산하고 denominator 라는 변수에 저장한다.
  - 위에서 계산한 numerator를 denominator로 나누면 정규화가 되고, 이 값을 normalize에 저장한다. return() 을 이용하여 함수의 값으로 normalize를 보고한다.

function 만들기 4

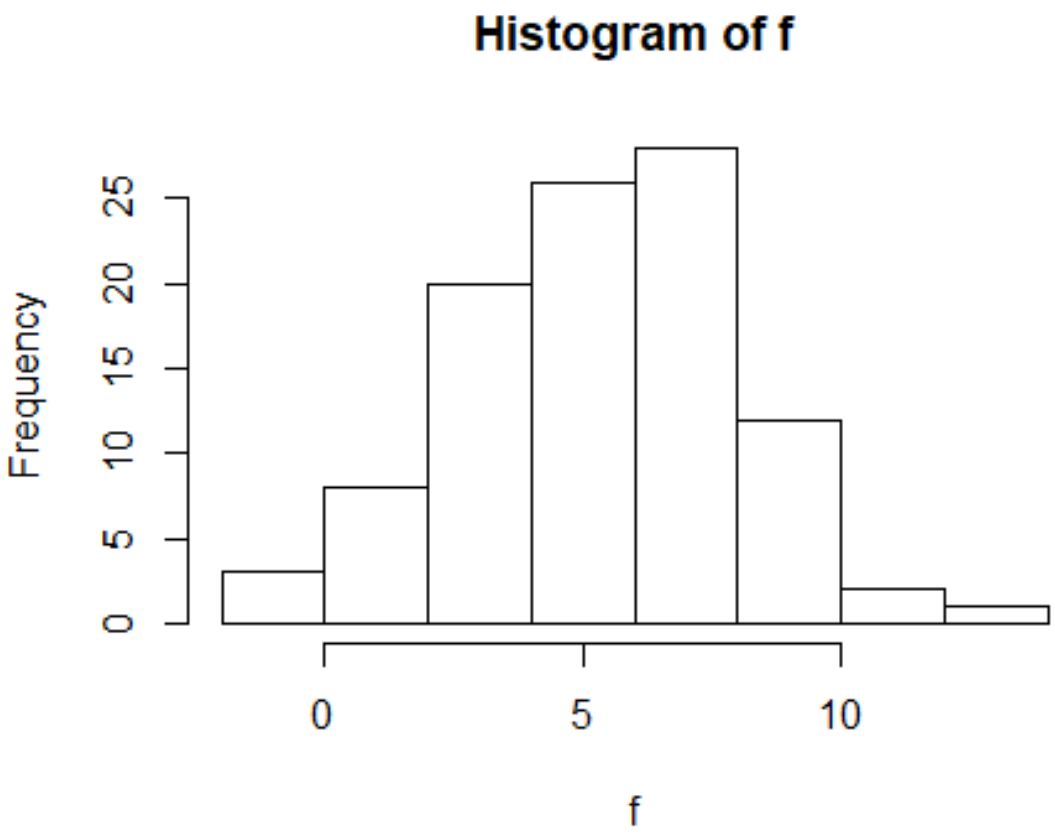
- 위에서 설명한대로 함수를 작성하면 아래와 같다.

```
> normalize <- function(x){
+   # step 1: create the numerator
+   numerator <- x-mean(x)
+
+   # step 2: create the denominator
+   denominator <- sd(x)
+
+   # step 3: divide nominator by denominator
+   normalize <- numerator/denominator
+
+   # return the value
+   return(normalize)
+ }
```

function 만들기 5

- 이렇게 정의된 함수가 어떻게 작동하는지 테스트해보기 위해서 아래와 같이 평균 5, 표준편차 3인 정규분포를 따르는 벡터 f를 생성하자.

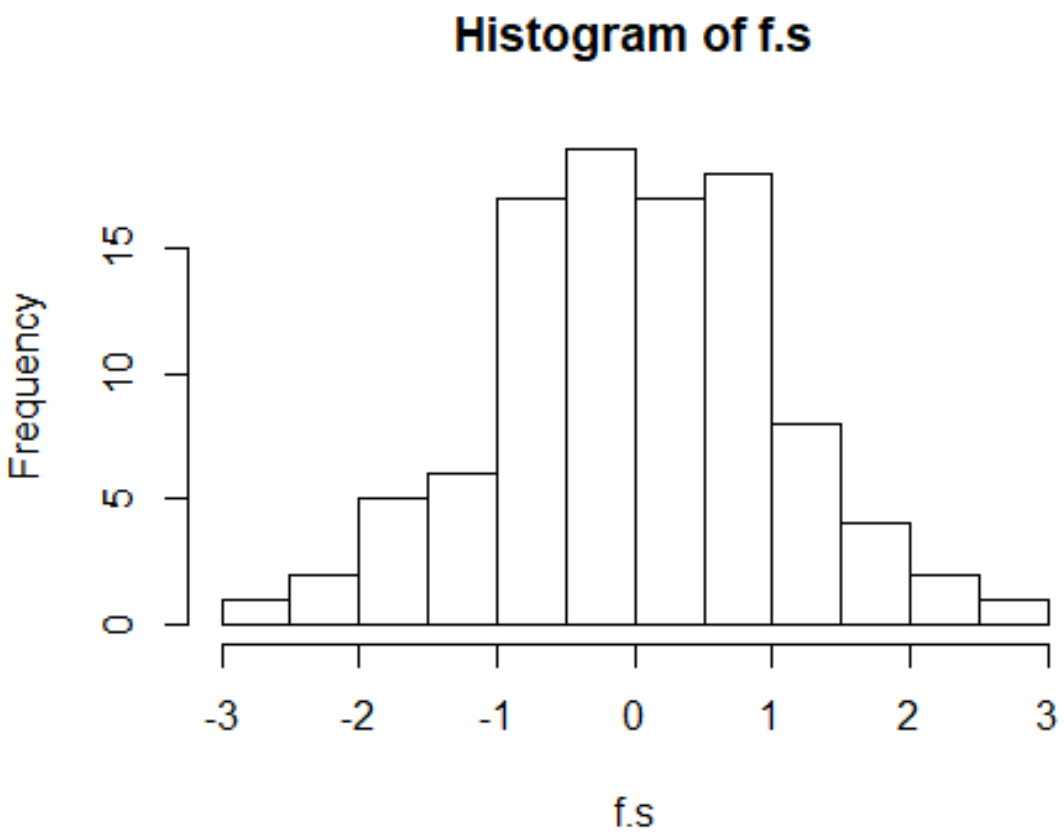
```
> f <- rnorm(n    = 100,
+            mean = 5,
+            sd   = 3)
> hist(f)
```


Histogram of f
Frequency
0
5
10
15
20
25
0
5
10
f

function 만들기 6

- 이러한 f를 위에서 정의한 normalize() 함수를 적용해보자.

```
> f.s <- normalize(f)
> hist(f.s)
```


Histogram of f.s
Frequency
0
5
10
15
-3
-2
-1
0
1
2
3
f.s

function 만들기 7

- normalize() 함수를 적용한 결과인 f.s 가 표준화되었는지를 확인하기 위해서 평균과 표준편차를 구해보자.

```
> mean(f.s)
[1] -8.334696e-17
```

- 벡터 f.s 의 표준편차는 1로 정확히 표준화 되었음을 알 수 있다.

```
> sd(f.s)
[1] 1
```

자료 관리하기(DATA MANAGEMENT)

개요

- R에서 통계 패키지에서 자료라고 부르는 것에 해당하는 것은 데이터 프레임(data frame)이다.
- 데이터 프레임은 같은 길이의 벡터의 모임 혹은 factor 의 모임이다.
  - 그래서 데이터 프레임은 사각형의 모양을 가지고 있다.
  - 데이터 프레임의 행은 관측치, 열은 변수를 의미한다.

개요 2

- 데이터 프레임의 특징은 다음과 같이 요약해볼 수 있다.

1. 데이터 프레임은 행과 열의 집합이다.
2. 각 열은 같은 길이를 가지면 자료의 타입도 동일하다.
3. 각 행은 같은 길이를 가지지만 자료의 타입은 다를 수도 있다.
4. 데이터 프레임은 matrix와 list의 특성을 동시에 가질 수 있다.
5. 데이터 프레임의 인덱싱은 [] 를 사용한다.

data frame

- 데이터 프레임은 다음과 같이 생성할 수 있다.
- data.frame() 함수를 이용하면 된다.
  - 기본적으로 벡터들을 모아서 데이터 프레임으로 지정하면 된다.

```
> rm(list = ls())
> x <- data.frame(
+   x1 = 1:10,
+   x2 = 11:20,
+   x3 = 21:30
+   )
```

data frame 2

- 그러면 class는 기본적으로 데이터 프레임을 갖게 된다.

```
> class(x)
[1] "data.frame"
```

- 모드는 list 이다.

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

data frame 2

- 다른 예를 들어보자.
- R에서 자주 사용되는 자료를 하나 불러들이자.
- mtcars 라는 자료로서 아래와 같이 불러들일 수 있다. mtcars는 자동차에 대한 자료로서
- 이미 data.frame의 특성을 가지고 있다.

```
> rm(list = ls())
> data("mtcars")
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4
22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95
123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21
3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
```

data frame 3

- 다음과 같이 자료 관리를 할 수 있다.
- 예컨대 1-3행까지를 아래와 같이 추출할 수 있다.

```
> # select rows 1-3
> mtcars[1:3,]
                mpg cyl disp  hp drat    wt  qsec
vs am gear carb
Mazda RX4      21.0   6  160 110 3.90 2.620 16.46
0  1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02
0  1    4    4
Datsun 710     22.8   4  108  93 3.85 2.320 18.61
1  1    4    1
```

data frame 4

- mtacrs의 1-3행과 1, 4열을 아래와 같이 추출할 수 있다.

```
> # select rows 1-3 and columns 1 and 4
> mtcars[1:3,c(3,5)]
                  disp drat
Mazda RX4          160 3.90
Mazda RX4 Wag      160 3.90
Datsun 710         108 3.85
```

data frame 5

- 논리 조건문을 걸어서 mpg가 30 이상인 관측치를 추출할 수 있다.
- []안에 mpg>30을 넣어주면 된다.

```
> # Find the rows where the MPG column is greater
than 30
> a <- mtcars[mtcars$mpg > 30, 1:3]
> head(a, n = 3)
                mpg cyl disp
Fiat 128       32.4   4 78.7
Honda Civic    30.4   4 75.7
Toyota Corolla 33.9   4 71.1
```

data frame 6

- mpg가 30보다 크고 cyl가 4기통인 자동차의 1-3열은 아래와 같이 추출할 수 있다.

```
> # select columns 1-3 for all rows
> # where MPG > 30 and cylinder is equal to 4
> a <- mtcars[mtcars$mpg > 30 & mtcars$cyl == 4,
1:3]
> head(a, n = 3)
                mpg cyl disp
Fiat 128       32.4   4 78.7
Honda Civic    30.4   4 75.7
Toyota Corolla 33.9   4 71.1
```

data frame 7

- 다음과 같이 tapply()를 적용하여 cyl별 mpg의 평균을 계산할 수도 있다.

```
> # Get the mean MPG by Transmission
> tapply(mtcars$mpg, mtcars$cyl, mean)
       4        6        8
26.66364 19.74286 15.10000
```

DATA WRANGLING

dplyr 패키지

- 데이터 프레임을 다룰 때 이전에는 R에 기본적으로 내장되어 있는 함수들을 주로 사용하였지만,
- 요즘에는 dplyr 패키지에 포함되어 있는 함수들을 사용하는 것이 보통이다.
- 따라서 예전 명령어들을 익히는 것보다는 요즘에 쓰는 dplyr 패키지에 포함되어 있는 명령어를 익히는 것이 훨씬 유익하다.

dplyr 패키지 2

- 이를 위해서 dplyr 패키지를 library() 함수를 이용하여 장착하자.

```
> library(dplyr)
```

dplyr 패키지 3

- 많이 쓰는 함수들은 다음과 같다.
  - select(): 변수 선택
  - filter(): 관측치 필터링
  - mutate(): 새로운 변수 생성
  - arrange(): 자료의 순서 정리
  - group_by(): 범주별로 정리
  - summarise(): 요약통계량 작성
  - join(): 서로 다른 데이터 프레임의 병합

dplyr 패키지 4

- dplyr 패키지를 이용한 자료 관리에서 가장 중요한 것은 이른바 파이프(pipe)라고 불리는 “%>%”를 적절히 사용하는 것이다.
- 파이프 %>% 는 기본적으로 앞 함수에서 구한 결과를 뒤의 함수로 넘겨주는 역할을 한다.
- 그래서 여러 개의 함수를 단계별로 이용해야 하는 경우 논리적 흐름에 따라 물 흐르듯이 자연스럽게 명령어의 조합을 구성할 수 있기 된다.

dplyr 패키지 5

- 다음과 같은 filter() 함수를 생각해보자.
- 아래 filter() 함수의 경우 data를 이용해서, 특정 변수 variable 이 특정된 numeric_value 를 갖는 관측치만을 골라내는 역할을 한다.

```
> filter(data, variable == numeric_value)
```

dplyr 패키지 6

- 파이프 %>% 를 이용하는 경우, 그 순서가 논리적으로 다음과 같이 진행된다.

1. data 라는 자료를 불러내고
2. 자료 중 variable이 특정 numeric_value를 갖는 관측치를 골라낸다.

```
> data %>%
+   filter(variable == numeric_value)
```

dplyr 패키지 7

- 다음과 같이 dplyr 패키지를 장착하자. 혹시 dplyr 패키지가 설치되어 있지 않은 경우에는 intall.packages("dplyr") 을 통해서 dplyr 패키지를 설치하자.

```
> if (!require(dplyr)) install.packages('dplyr')
> rm(list = ls())
> library(dplyr)
```

dplyr 패키지 8

- 그리고 다음과 같이 분석에 사용할 자료를 불러들이자.

```
> lfp <- read.csv("laborforce.csv",
+                 header = TRUE)
```

dplyr 패키지 9

- 어떤 자료인지는 str()함수를 이용하면 알 수 있다.
- 이 자료는 지난 18년 동안(2000-2017년) 지역별 경제활동인구 규모에 관한 자료이다(kosis.kr). 단위는 천명이다.
- 포함된 변수는 지역 region, 연도 year, 총경제활동인구 total, 남성 경제활동인구 male, 여성 경제활동인구 female이다.

dplyr 패키지 10

```
> str(lfp)
'data.frame':   306 obs. of  5 variables:
 $ region: Factor w/ 17 levels "강원도","경기도
",..: 9 8 6 12 5 7 11 2 1 17 ...
 $ year  : int  2000 2000 2000 2000 2000 2000
2000 2000 2000 2000 ...
 $ total : num  4918 1758 1148 1164 592 ...
 $ male  : num  2909 1028 667 716 345 ...
 $ female: num  2010 730 481 448 247 ...
```

select

- select() 함수는 데이터 프레임 중 필요한 변수만을 골라내는 역할을 한다.
- 예를 들어 lfp 변수 중 total을 제외한 지역 region, 연도 year, 남성 규모 male, 여성 규모 female 변수를 골라내자.

select 2

```
> lfp.sub <- lfp %>%
+   select(region, year, male, female)
> head(lfp.sub)
     region year male female
1 서울특별시 2000 2909   2010
2 부산광역시 2000 1028    730
3 대구광역시 2000  667    481
4 인천광역시 2000  716    448
5 광주광역시 2000  345    247
6 대전광역시 2000  360    249
```

select 3

- 포함될 변수명을 써주는 대신 제외할 변수명을 써주면 된다.

```
> lfp.sub <- lfp %>%
+   select(-total)
> head(lfp.sub)
      region year male female
1 서울특별시 2000 2909   2010
2 부산광역시 2000 1028    730
3 대구광역시 2000  667    481
4 인천광역시 2000  716    448
5 광주광역시 2000  345    247
6 대전광역시 2000  360    249
```

filter

- filter() 함수는 특정 변수가 특정 조건을 만족시키는 관측치를 골라 내는 함수이다.
- 예를 들면, region이 서울특별시인 관측치만 골라내보자.

```
> lfp.seoul <- lfp %>%
+   filter(region == "서울특별시")
```

filter 2

- filter() 함수 안에서는 아래와 같은 논리 연산자들을 사용할 수 있다

< Less than

!= Not equal to > Greater than

%in% Group membership == Equal to

is.na is NA <= Less than or equal to

!is.na is not NA >= Greater than or equal to

&,|,! Boolean operators

filter 3

- filter() 함수의 조건은 여러 개가 될 수 있다. 예컨대 서울특별시 자료중 2010년 이후만 따로 골라낼 수 있다.

```
> lfp.seoul.2010 <- lfp %>%
+   filter(region == "서울특별시" &  year > 2009)
```

```
> head(lfp.seoul.2010, n = 5)
      region year total male female
1 서울특별시 2010  5308 3029   2280
2 서울특별시 2011  5395 3070   2325
3 서울특별시 2012  5371 3035   2336
4 서울특별시 2013  5355 3016   2339
5 서울특별시 2014  5449 3057   2392
```

group_by

- group_by() 함수는 범주형 변수의 값별로 데이터 프레임을 정리해 주는 역할을 한다.
- 예컨대, 연도별로 그룹핑을 하는 경우를 생각해보자.

```
> lfp %>%
+   select(region, year, total) %>%
+   group_by(year) %>%
+   head()
# A tibble: 6 x 3
# Groups:   year [1]
  region       year total
  <fct>       <int> <dbl>
1 서울특별시  2000  4918
2 부산광역시  2000  1758
3 대구광역시  2000  1148
4 인천광역시  2000  1164
5 광주광역시  2000   592
6 대전광역시  2000   608
```

summarise

- summarise() 함수는 요약통계량을 구하는 함수이다. 연도별로 지역수 N, 지역평균 mean, 지역합 sum이 계산할 수 있다.

```
> lfp %>%
+   select(region, year, total) %>%
+   group_by(year) %>%
+   summarise(N = n(), mean = mean(total), sum =
sum(total)) %>%
+   head(n = 5)
# A tibble: 5 x 4
   year     N  mean   sum
  <int> <int> <dbl> <dbl>
1  2000    16 1384. 22152
2  2001    16 1407  22512
3  2002    16 1436. 22981
4  2003    16 1440. 23042
5  2004    16 1472. 23546
```

mutate

- 자료를 처리하다 보면 기존 변수들을 이용해서 새로운 변수를 생성할 필요가 있다.
- mutate() 함수는 새로운 변수를 만들 때 사용할 수 있다.

mutate 2

- 예컨대, 전체 경제참가자 중 여성 비율을 어떻게 되는지를 나타내는 변수를 생성하고 싶다고 해보자.
  - 먼저 새로운 변수 계산에 필요한 변수들을 select() 함수를 통해서 솔라낸다.
  - 그리고 mutate()함수를 통해서 경제활동참가자 중 여성의 비율을 계산하고
  - 이를 lfp.ratio.female 이라고 저장한다.
  - 이렇게 새로 생성된 변수인 lfp.ratio.female을 연도별로 그룹핑하여 관측치, 평균, 표준편차를 계산한다.
  - 이렇게 계산된 결과를 t.2 에 저장하고 프린트 한 것이 아래의 결과이다.

```
> t.2 <- lfp %>%
+   select(region, year, total, female) %>%
+   mutate(lfp.ratio.female = female/total*100)
%>%
+   group_by(year) %>%
+   summarise(N                = n(),
+             ratio.female =
mean(lfp.ratio.female),
+             sd               =
sd(lfp.ratio.female)) %>%
+   ungroup()
```

```
> head(t.2)
# A tibble: 6 x 4
   year     N ratio.female    sd
  <int> <int>        <dbl> <dbl>
1  2000    16         41.6  2.78
2  2001    16         41.9  2.75
3  2002    16         42.2  2.71
4  2003    16         41.4  2.69
5  2004    16         41.7  2.46
6  2005    16         41.9  2.55
```

GGPLOT2

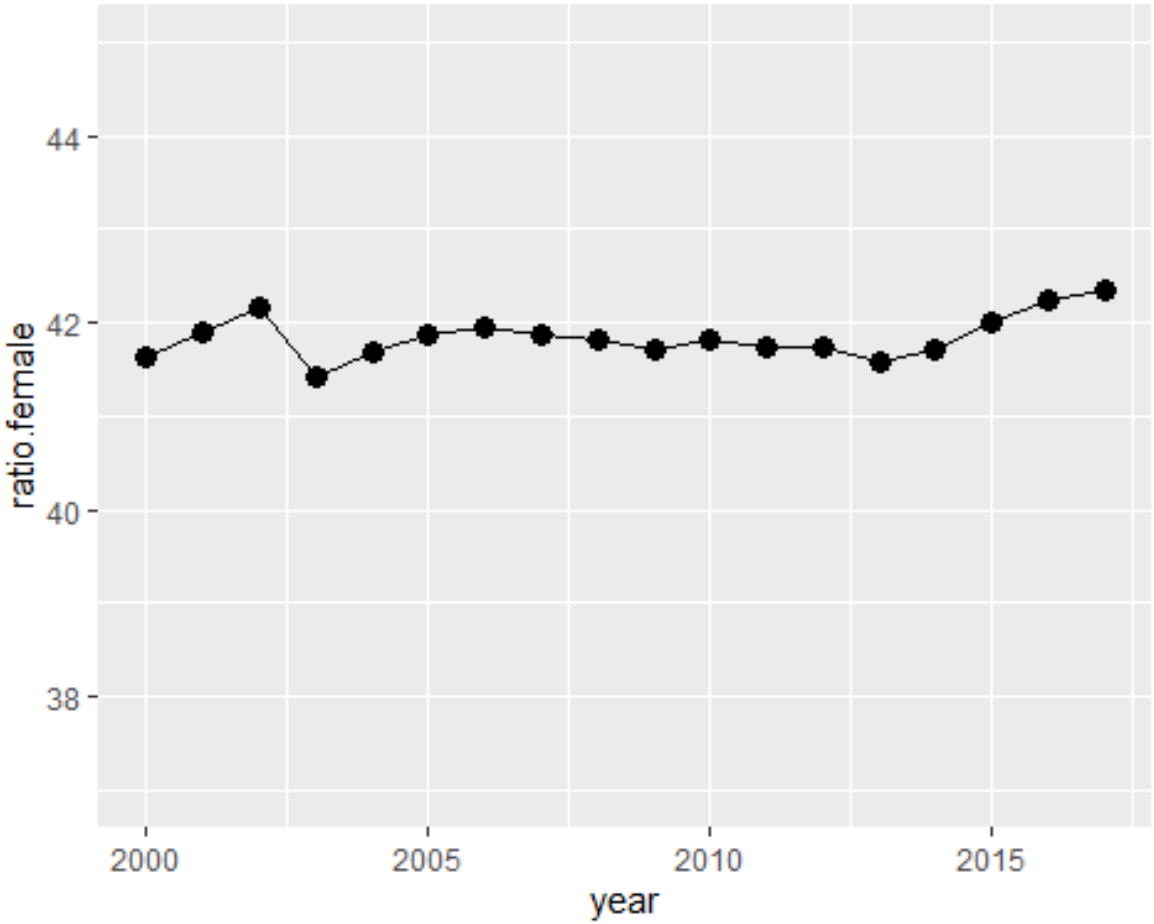
Summary

- 이렇게 dplyr 패키지를 이용하여 산출된 데이터 프레임을 이용하여 다음과 같이 간단한 그래프를 그려볼 수 있다.
- 그래프를 그리기 위해서는 R에 기본으로 장착된 그래프 함수들을 이용하는 것도 좋지만
- 요즘에는 대다수가 ggplot2 패키지를 이용해서 그래프를 그리고 있다.
- 처음부터 ggplot2 패키지에 포함된 함수들을 이용해서 그래프를 그리는 것을 배우는 것이 효율적이다.

Summary 2

- 단계
  - ggplot2를 library() 함수를 이용하여 장착한다.
  - dplyr의 자료 관리 함수들을 이용해서 바-그래프 그리기에 필요한 자료를 위의 예처럼 생성하였다.
  - 그렇게 주어진 자료를 ggplot() 함수로 패스하였다.
  - 뒤에 좀 더 자세히 설명하겠지만 x축은 연도, y축은 여성 비율이 되도록 aesthethics를 정해주었다.
  - 그리고, geom_line() 함수를 이용하여 아래와 같은 선-그래프를 그리고
  - 표준오차 밴드를 geom_ribbon()을 이용하여 그렸다.

```
> library(ggplot2)
> lfp %>%
+   select(region, year, total, female) %>%
+   mutate(lfp.ratio.female = female/total*100)
%>%
+   group_by(year) %>%
+   summarise(N = n(), ratio.female =
mean(lfp.ratio.female), sd =
sd(lfp.ratio.female)) %>%
+   ggplot(mapping = aes(x = year, y =
ratio.female)) +
+   geom_point(size = 3) +
+   geom_line() +
+   xlim(min = 2000, max = 2017) +
+   ylim(min = 37, max = 45)
```


44
42
40
38
ratio.female
2000
2005
2010
2015
year

자료 읽기와 쓰기

자료 읽기와 쓰기

- 자료 분석을 위해서는 R로 자료를 불러들여와야 한다.
- R에서 불러 들일 수 있는 자료는 크게 보면 다음과 같이 3가지 정도로 구분할 있다.

  1. txt, csv, html 과 같은 텍스트 파일들
  2. STATA, SAS, SPSS 같은 통계패키지 파일들
  3. 데이터 베이스 등과 같은 소스에서 읽어 들이기

자료 읽기와 쓰기 2

- 경제자료를 분석하기 위해서 필요한 첫 번째 작업은 자료를 읽는 것이다.
- 제일 간단한 형태는 txt 파일 혹은 csv 파일이다.
- txt 파일은 read.csv() 함수를 이용해서 읽을 수 있다.
  - 자료 읽기 전에 간단한 작업을 해줄 필요가 있다.
  - setwd() 함수를 이용해서 자료가 있는 작업 폴더로 이동하는 것이다.
  - 작업 폴더로 이동하지 않은 경우에는 자료를 읽을 때마다 경로를 지정해주어야 한다.

자료 읽기와 쓰기 3

```
> rm(list = ls())
> df.txt <- read.csv("swimming_pools.csv",
+                    header = TRUE,
+                    sep = ",")
```

자료 읽기와 쓰기 4

- 한편, readr 패키지도 많이 사용되고 있다.
  - 특히, 속도면에서 readr 패키지가 빠르기 때문에 빅데이터를 로딩할 때 자주 사용된다.

```
> if (!require(readr)) install.packages('readr')
> library(readr)
```

- 그러면 read.csv 대신에 read_csv 를 사용할 수 있다.

```
> df.csv2 <- read_csv("swimming_pools.csv")
> class(df.csv2)
[1] "spec_tbl_df" "tbl_df"      "tbl"
"data.frame"
```

엑셀자료 읽기

- 다음으로는 엑셀 형태의 자료를 읽어 보자.
- 엑셀 형태의 자료를 불러들이기 위해서는 xlsx 패키지가 필요하다.
  - 만약 xlsx 패키지가 설치되어 있지 않다면 install.packages("xlsx")를 통해서 해당 패키지를 설치하자.

```
> if (!require(readxl))
install.packages('readxl')
> library(readxl)
```

엑셀자료 읽기 2

- read.xlsx() 함수를 이용하여 urbanpop.xlsx 파일을 불러들이자.
  - 해당 파일은 8개 변수에 209개의 관측치가 있는 자료이다.

```
> df.xlsx <- read_excel("urbanpop.xlsx", sheet =
1)
```

- 이렇게 불러들인 자료 df.xlsx의 class 는 데이터 프레임이다.

```
> class(df.xlsx)
[1] "tbl_df"     "tbl"        "data.frame"
```

file.choose()

- 한편 file.choose()를 이용하는 것도 하나의 방법이다.
- file.choose() 함수를 포함시키는 경우 원도우 탐색창을 이용해서 읽어 들일 자료를 선택할 수도 있다.

```
> df.xlsx.2 <- read_excel(file.choose())
```

통계패키지 자료 읽기

- 통계패키지에서 생성된 자료를 R로 읽어 들이기 위해서는 적절한 패키지가 필요하다.
- STATA의 경우에는 foreign 패키지가, SAS와 SPSS의 경우에는 Hmisc 패키지를 사용하는 것이 적절하다.
- 예를 들어 STATA 파일인 확장자가 dta인 파일을 읽어들이자.
  - 먼저 foreign 패키지를 장착하자.
  - 최근 버젼의 STATA 파일을 읽어 들이기 위해서는 haven 패키지를 이용하는 것이 좋다.

```
> if (!require(foreign))
install.packages('foreign')
> if (!require(haven)) install.packages('haven')
>
> library(foreign)
> library(haven)
> df.stata <- haven::read_dta("auto.dta")
```

- haven 패키지 내의 read_sas, read_sav 등의 함수를 이용하면 SAS나 sav 파일도 읽어 들일 수가 있다.