6. 그래프

6.1 산점도

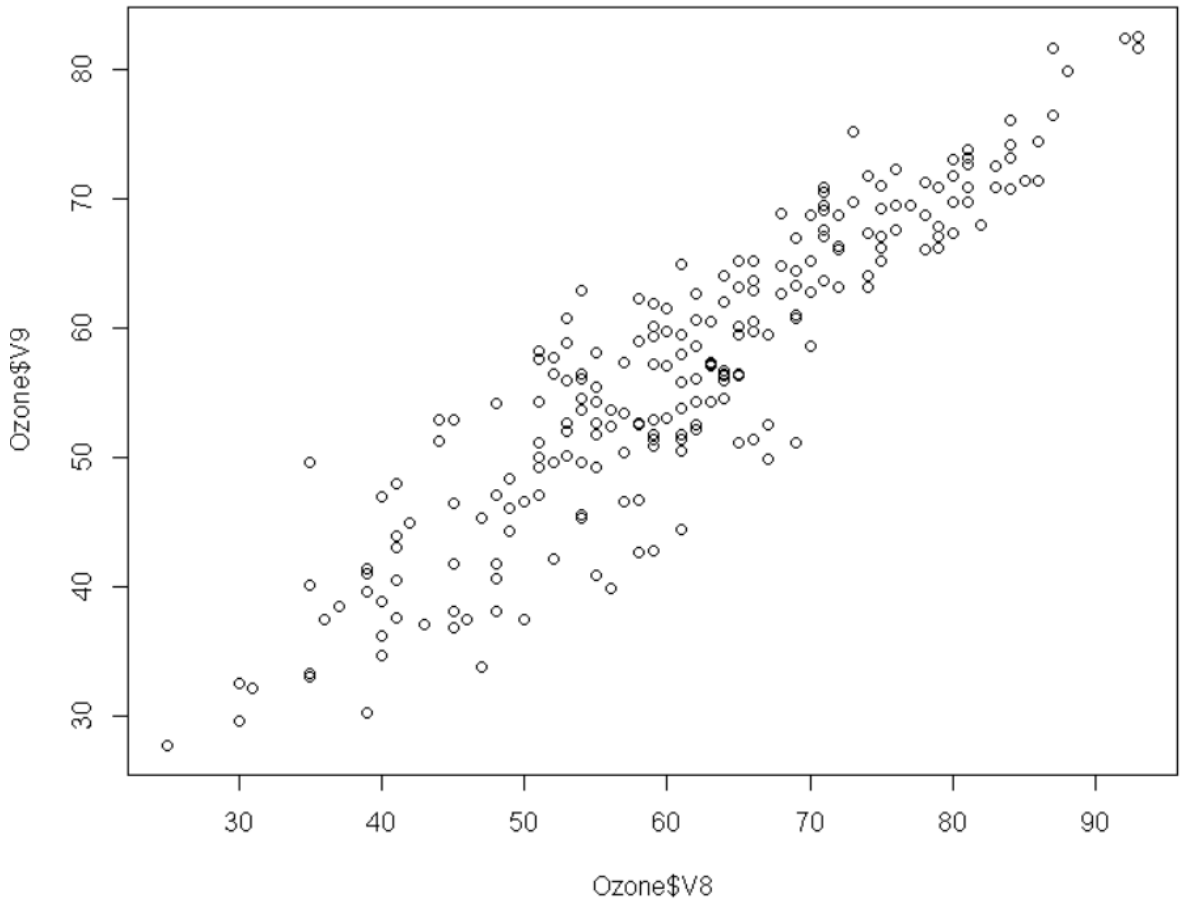
산점도(Scatter Plot)는 주어진 데이터를 점으로 표시해 흩뿌리듯이 시각화한 그림이다. 데이터의 실제 값들이 표시되므로 데이터의 분포를 한눈에 살펴보는데 유용하다. R에서 산점도는 plot() 함수로 그리는데, plot()은 산점도 뿐만 아니라 일반적으로 객체를 시각화하는 데 모두 사용할 수 있는 일반 함수다. 여기서 일반 함수란 주어진 데이터 타입에 따라 다른 종류의 plot() 함수 변형이 호출됨을 뜻한다. plot()이 어떤 객체들을 그려줄 수 있는지는 다음 명령으로 볼 수 있다.

```
> methods("plot")
 [1] plot.acf*            plot.data.frame*
 [3] plot.decomposed.ts*  plot.default
 [5] plot.dendrogram*     plot.density*
 [7] plot.ecdf            plot.factor*
 [9] plot.formula*        plot.function
[11] plot.hclust*         plot.histogram*
[13] plot.HoltWinters*    plot.isoreg*
[15] plot.lm*             plot.medpolish*
[17] plot.mlm*            plot.ppr*
[19] plot.prcomp*         plot.princomp*
[21] plot.profile.nls*    plot.R6*
[23] plot.raster*         plot.spec*
[25] plot.stepfun         plot.stl*
[27] plot.table*          plot.ts
[29] plot.tskernel*       plot.TukeyHSD*
see '?methods' for accessing help and source code
```

예를 들어, plot.lm은 lm이라는 클래스에 정의된 plot 메서드로, 'plot(lm객체)' 방식으로 호출하면 자동으로 lm클래스의 plot이 호출된다. 즉, plot()은 인자로 주어진 객체에 따라 달리 처리된다.

plot() 함수를 사용해 산점도를 그리는 방법을 다음 표에 정리했다.

mlbench 패키지에 있는 Ozone 데이터를 사용해 산점도를 그려보자.




```
plot(Ozone$V8,Ozone$V9)
```

6.2 그래프 옵션

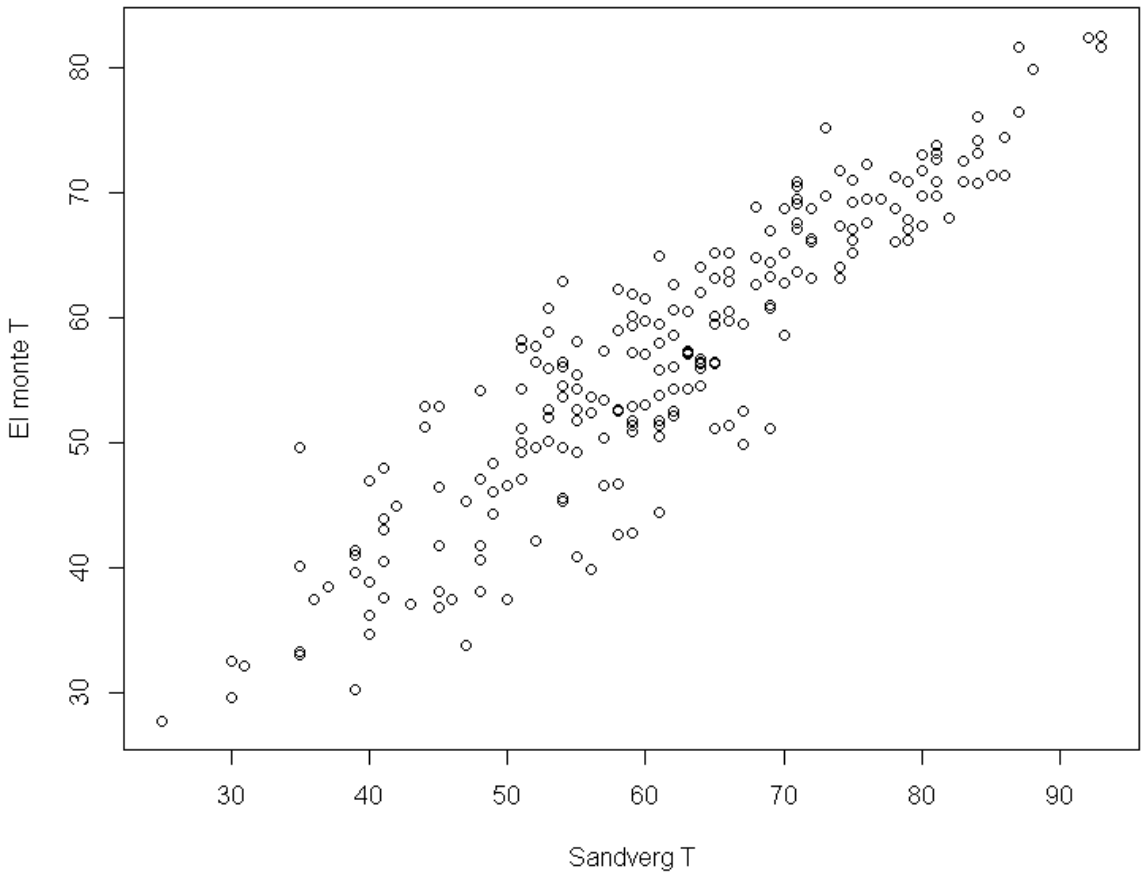
산점도에서 plot()으로 그려진 그림을 보면 축 이름, 점의 크기, 그래프 제목 등 여러 가지 개선할 점이 눈에 띈다.

그래프 옵션	의미
xlab,ylab	X,Y 축 이름
main	그래프 제목
pch	점의 종류
cex	점의 크기
col	색상
xlim,ylim	X,Y 축의 값 범위
type	그래프 유형, 점(p), 선(l), 점과 선 모두(b) 등

6.2.1 축 이름(xlab,ylab)

x축, y축의 레이블이 컬럼명이므로 그 의미를 잘 설명해주지 못한다는 것이다. 이를 해결하려면 plot함수에 xlab, ylab 옵션을 사용한다.

축 이름 지정	의미
plot(X,Y,xlab,ylab)	데이터를 그리되 x축의 축 이름을 xlab으로, y축의 축 이름을 ylab으로 설정한다.

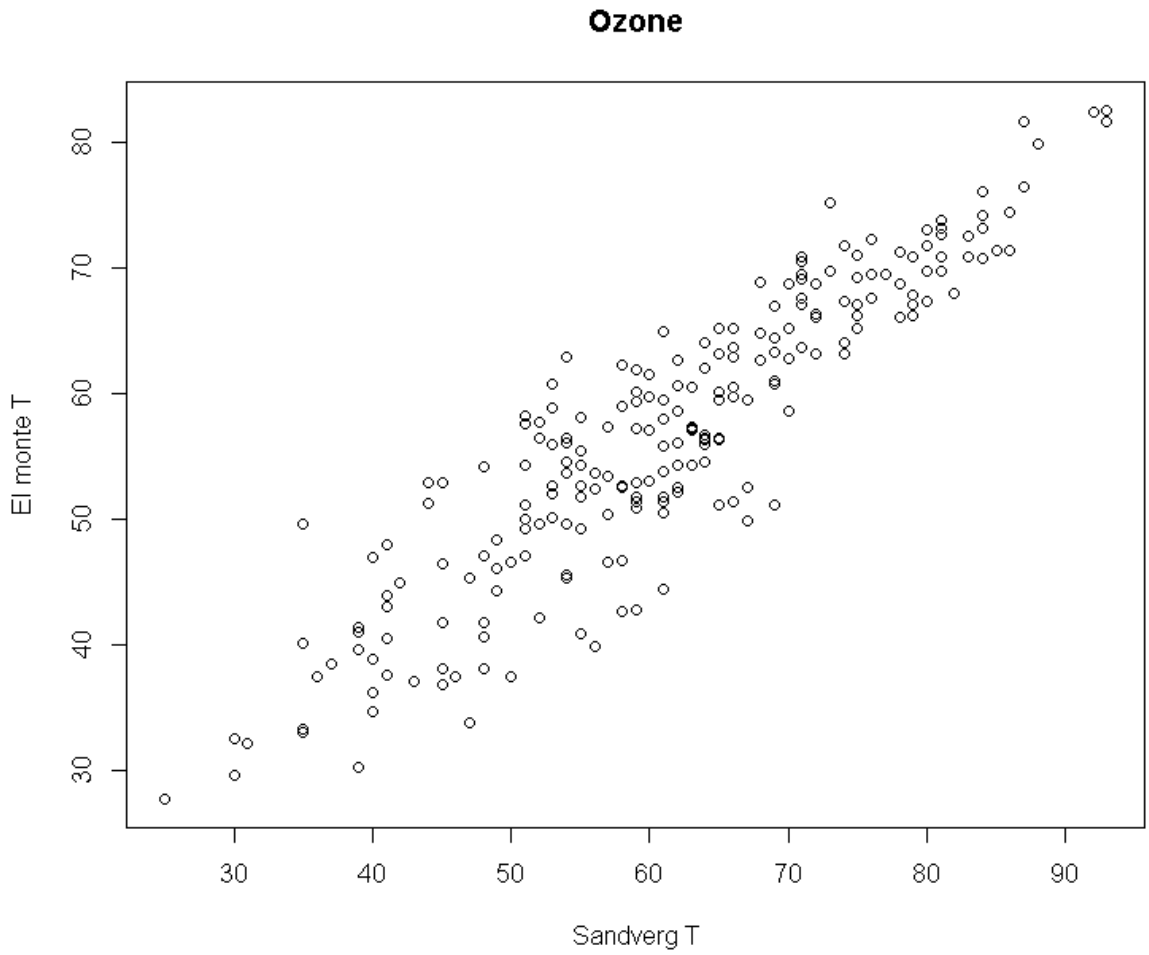



```
plot(Ozone$V8,Ozone$V9,
     xlab = 'Sandverg T',
     ylab = 'El monte T')
```

6.2.2 그래프 제목

그래프의 제목은 plot() 함수에 main 파라미터로 지정한다.

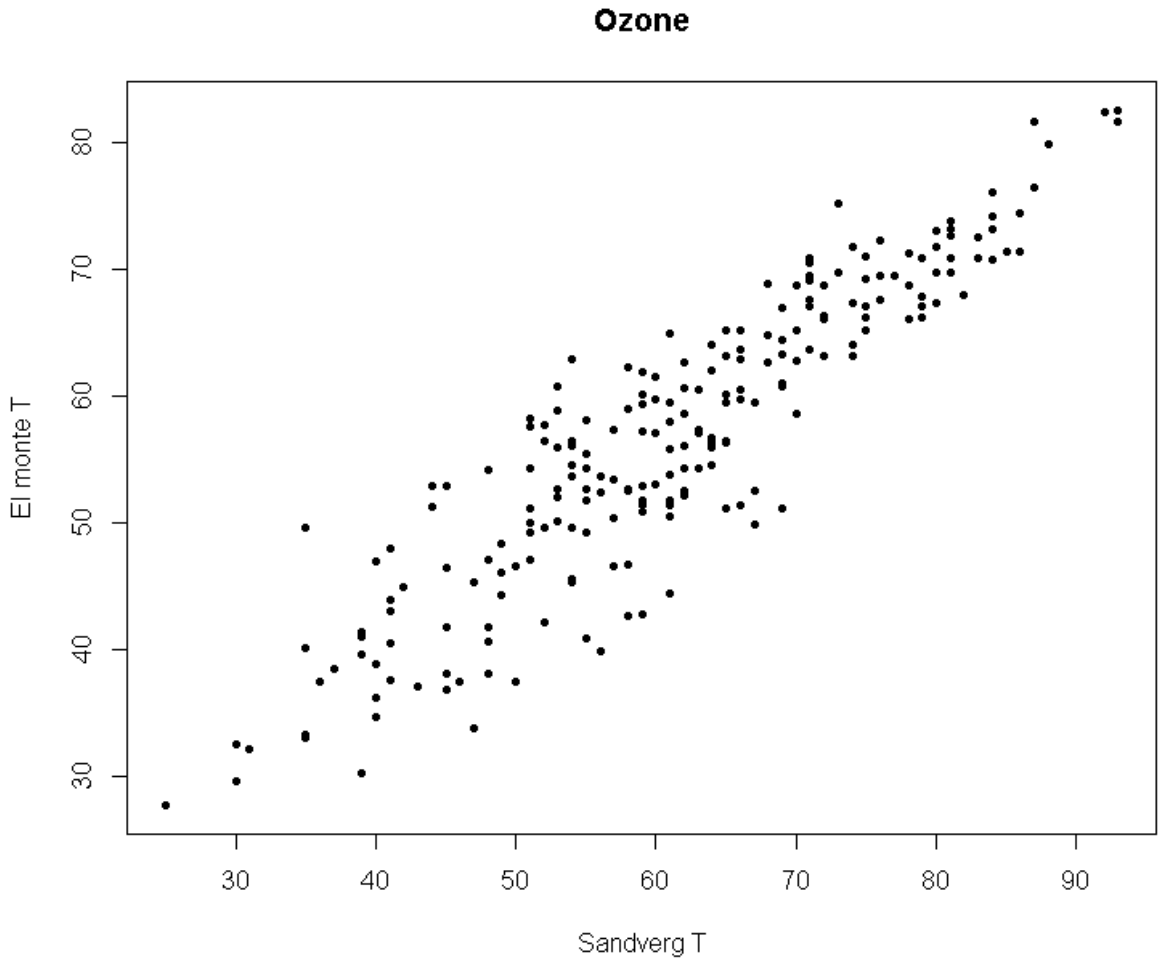
그래프 제목 지정	의미
plot(X,Y,main)	데이터를 그리되, 그래프의 제목을 main으로 지정한다.



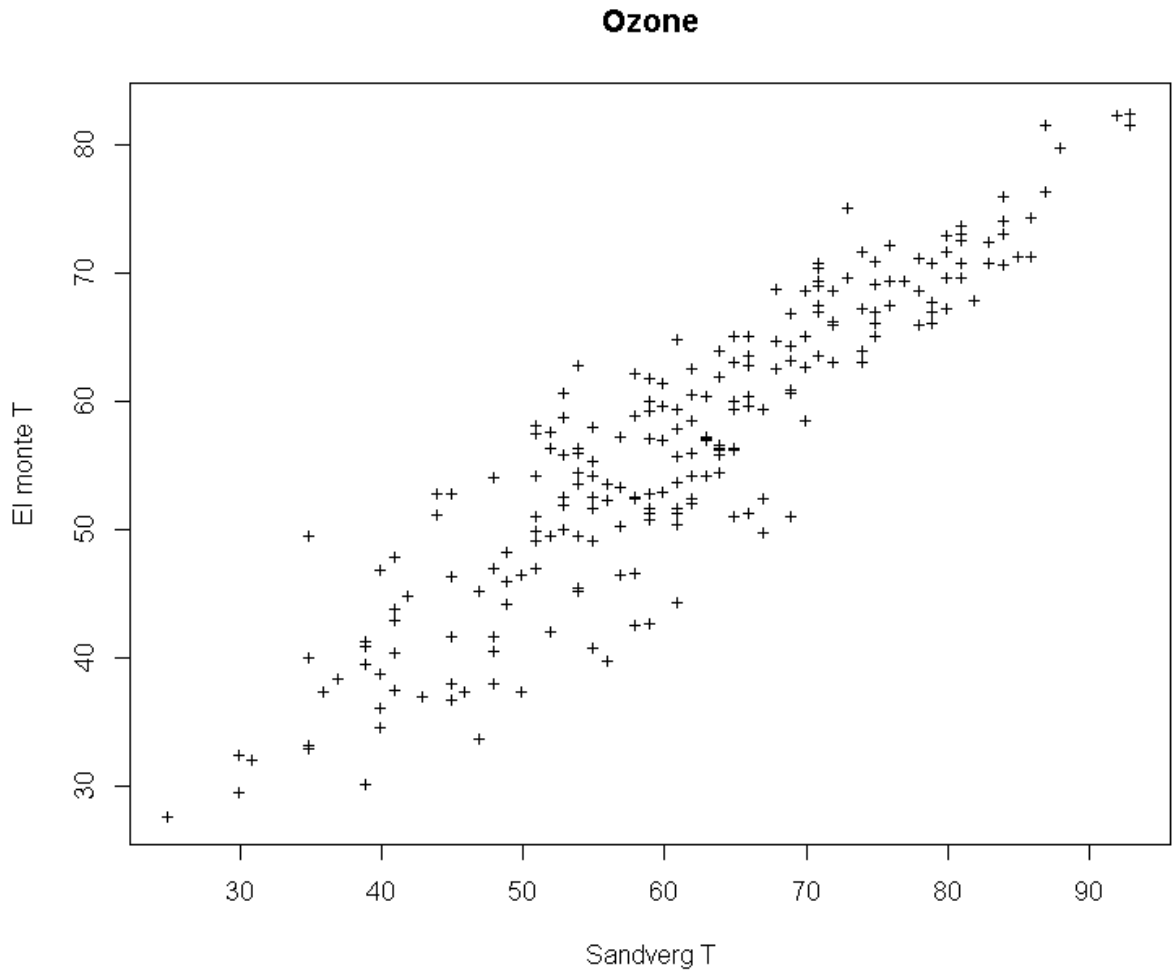

```
plot(Ozone$V8,Ozone$V9,
    xlab = 'Sandverg T',
    ylab = 'El monte T',
    main = 'Ozone')
```

6.2.3 점의 종류(pch)

그래프에 보이는 점의 모양은 pch로 지정하는데, pch에 숫자를 지정하면 미리 지정된 심볼이 사용되고, 문자(예를 들면, "+")를 지정하면 해당 문자를 사용해 점을 표시한다. 다음 표에 pch에 대해 정리했다.




```
plot(Ozone$V8,Ozone$V9,
    xlab = 'Sandverg T',
    ylab = 'El monte T',
    main = 'Ozone',
    pch = 20)
```




```
plot(Ozone$V8,Ozone$V9,
    xlab = 'Sandverg T',
    ylab = 'El monte T',
    main = 'Ozone',
    pch = "+")
```

6.2.4 점의 크기(cex)

산점도에 보인 점의 크기는 cex로 지정한다.

점의 크기

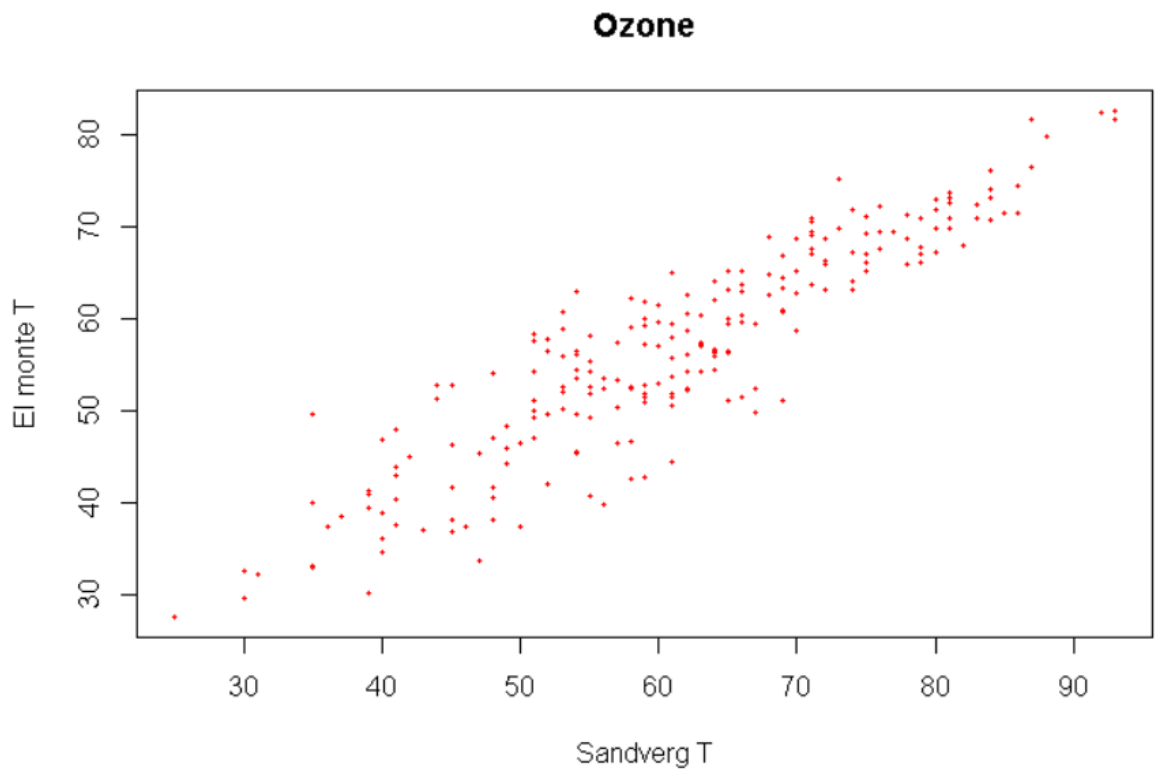
점의 크기 지정	의미
plot(X,Y,cex)	X,Y를 그리되 점의 크기를 cex로 한다. cex의 기본값은 1이며, cex가 작을수록 점의 크기가 비례하여 작아지고, 클수록 점의 크기가 cex에 비례해 커진다.


```
plot(Ozone$V8,Ozone$V9,
     xlab = 'Sandverg T',
     ylab = 'El monte T',
     main = 'Ozone',
     pch = 20,
     cex = 0.5)
```

6.2.5 색상(col)

색상은 col 파라미터로 지정한다. 앞서 pch, cex 등의 옵션은 점을 그릴 때만 해당하지만 색상은 점, 선 등 모두에 해당하는 차이가 있다.

다음 코드는 Ozone 산점도를 그리면서 col = "#FF0000"을 지정해 빨간색 점을 그렸다. col = "red" 처럼 색상 이름을 사용해도 같은 결과를 얻을 수 있다.




```
plot(Ozone$V8,Ozone$V9,
    xlab = 'Sandverg T',
    ylab = 'El monte T',
    main = 'Ozone',
    pch = 20,
    cex = 0.6,
    col = "#FF0000")
```

6.2.6 좌표축 값의 범위(xlim, ylim)

plot()이 기본으로 지정하는 X,Y 축 값의 범위가 들지 않는다면, 그래프에 그려질 x값의 범위, y값의 범위를 바꿔볼 수 있다.

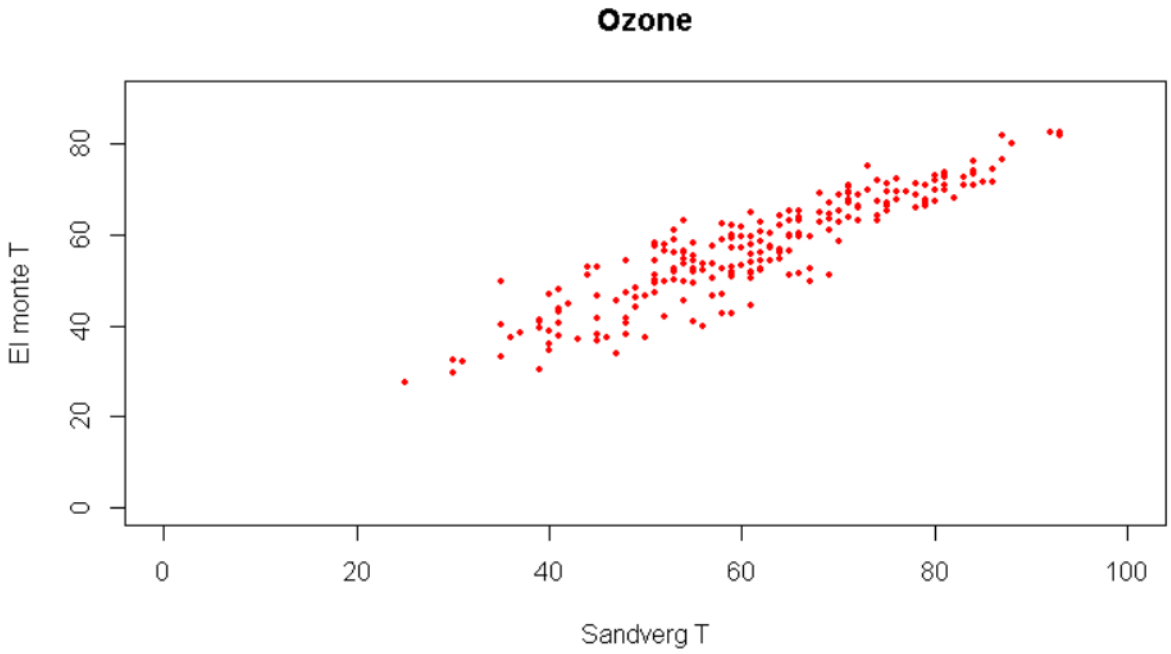
좌표축의 값의 범위를 지정하려면 먼저 데이터 값의 최소, 최대치를 알아야 한다.

Ozone$v8, Ozone$v9에는 NA 값이 포함되어 있으므로 min(), max() 함수에 na.rm = TRUE를 사용해 최솟값, 최댓값을 다음과 같이 구한다.

```
> min(Ozone$V8, na.rm = T)
[1] 25
> min(Ozone$V9, na.rm = T)
[1] 27.68
> max(Ozone$V8, na.rm = T)
[1] 93
> max(Ozone$V9, na.rm = T)
[1] 82.58
```

V8,V9의 최솟값, 최댓값을 참고해 다음 예에서는 V8의 범위를 c(0,100), V9의 범위를 c(0,90)으로 정했다.

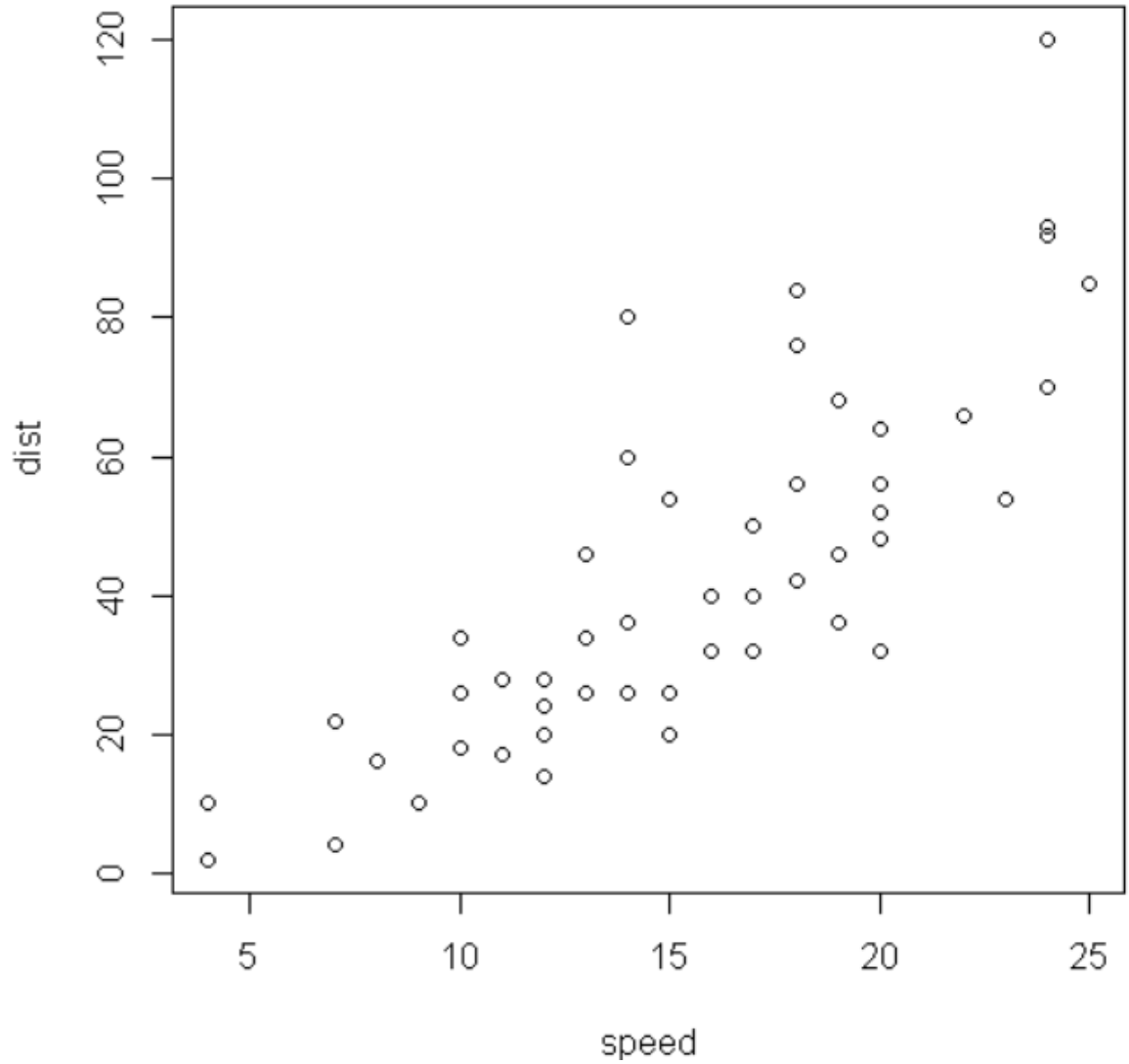
```
plot(Ozone$V8,Ozone$V9,
     xlab = 'Sandverg T',
     ylab = 'El monte T',
     main = 'Ozone',
     pch = 20,
     cex = 0.6,
     col = "#FF0000",
     xlim = c(0,100),
     ylim = c(0,90))
```



6.2.7 그래프 유형(type)

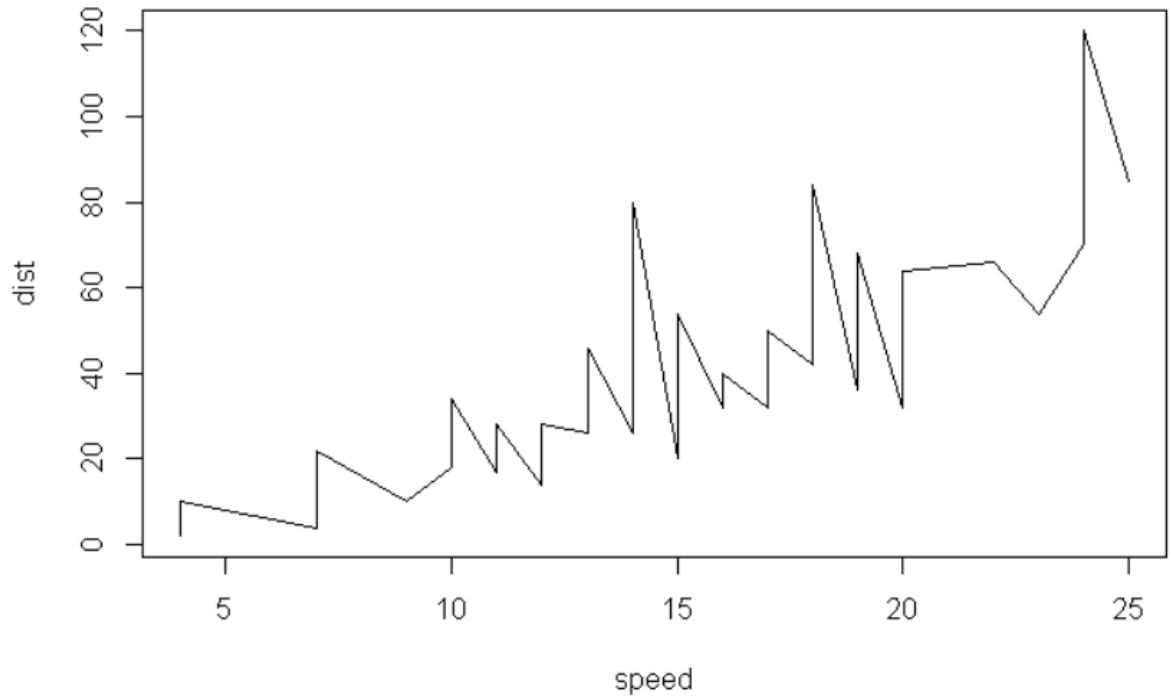
type은 plot()으로 데이터를 그래프로 그릴 때, 그래프 유형을 지정하는 옵션이다.

```
> data(cars)
> str(cars)
'data.frame':   50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> plot(cars)
```



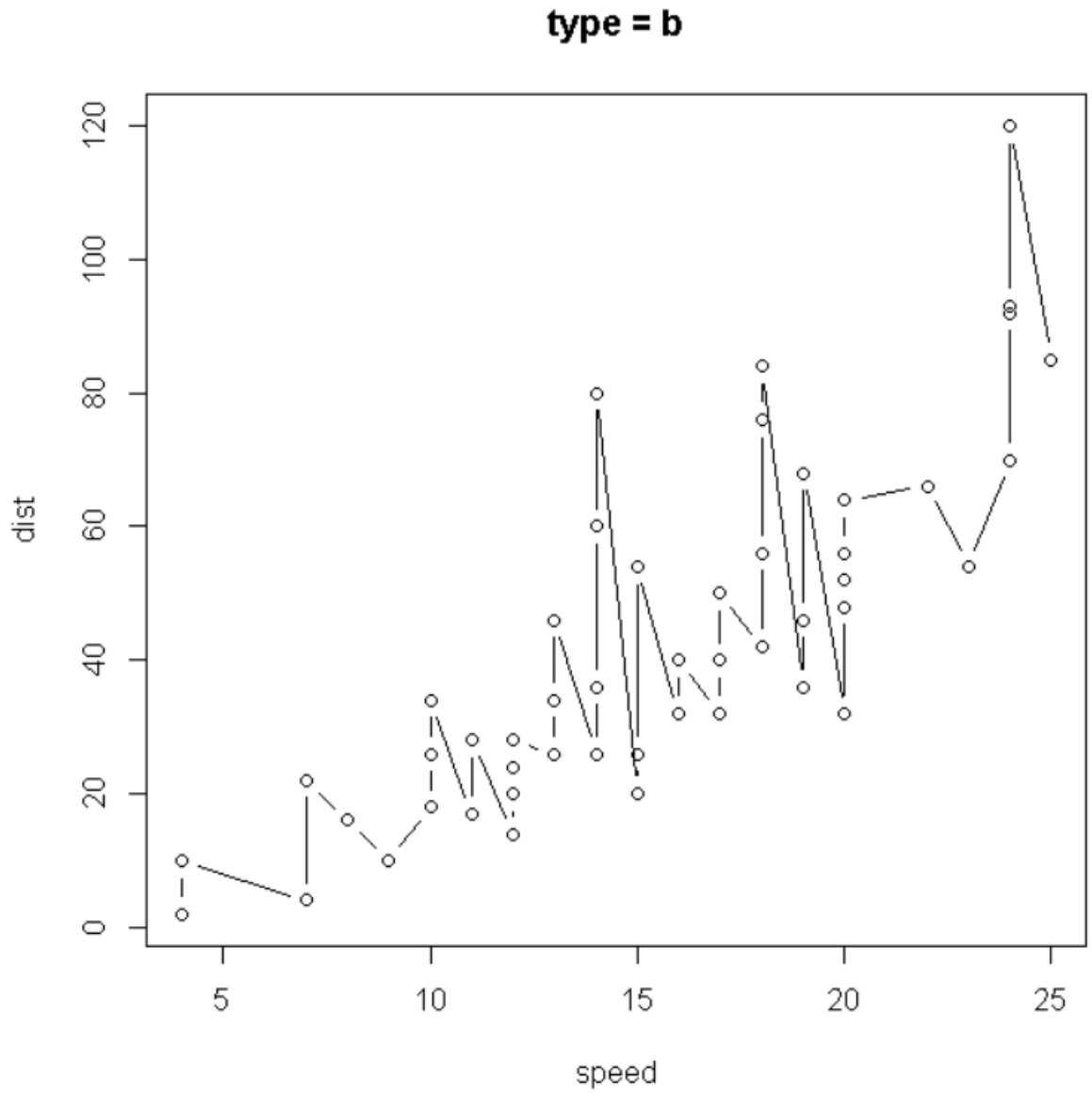
속도와 제동거리는 선형 상관관계가 있으리라는 것을 쉽게 짐작할 수 있다. 그렇다면 점으로 데이터를 표시하는 것보다는 꺾은 선으로 표시하는 것이 낫지 않을까? type = T을 지정하여 꺾은 선 그래프를 그릴 수 있다.

```
plot(cars, type = 'l')
```

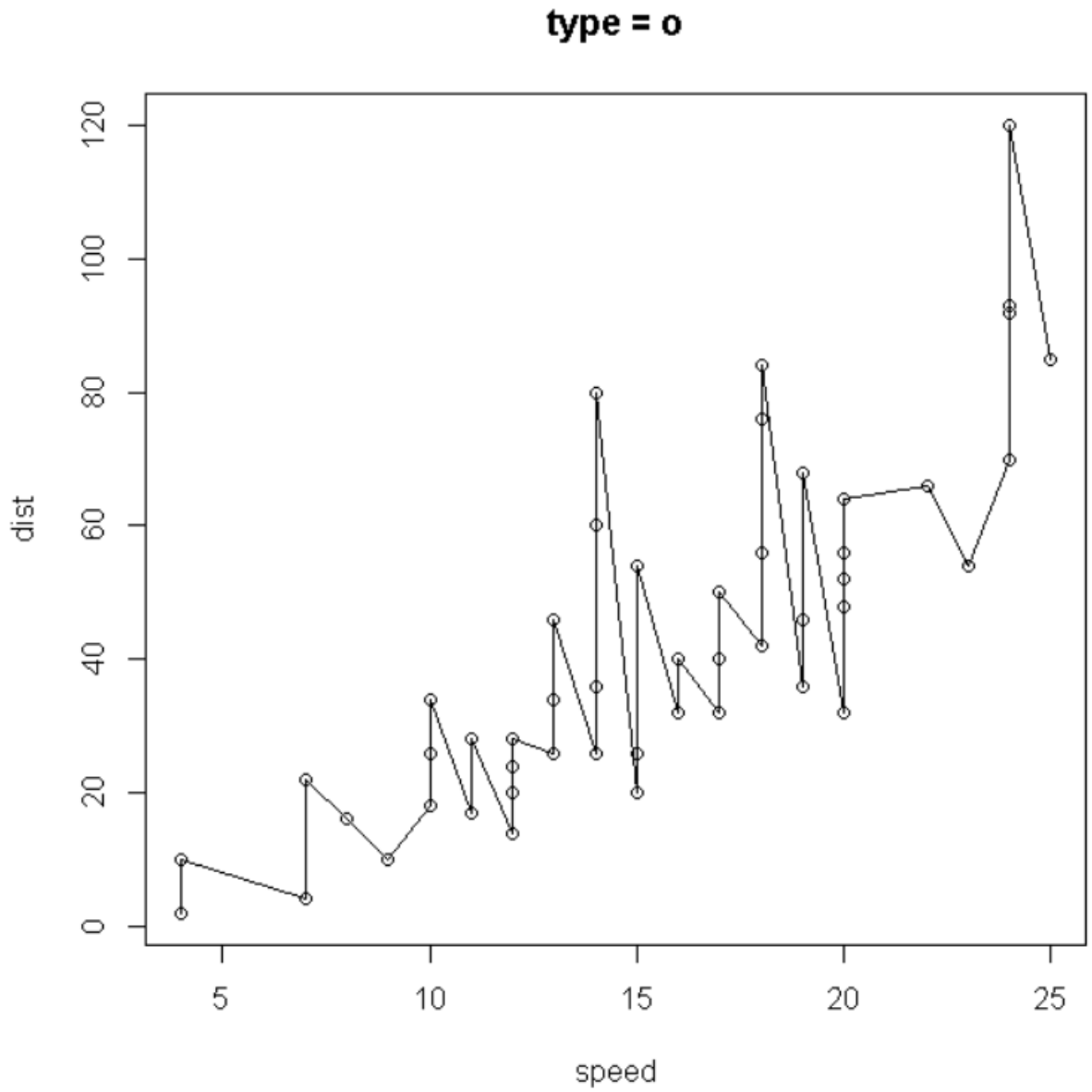


관찰된 점들과 선을 모두 그리려면 type = 'b'를, 중첩하여 그리려면 type = 'o' 를 지정한다. 이 둘은 점과 선을 모두 사용한다는 점은 비슷하지만, b는 점과 선이 중첩되지 않으며 o는 점 그래프를 그린 뒤 그 위에 꺾은 선 그래프를 중첩하여 그리는 차이가 있다. 다음은 이 두가지 경우의 예를 보여준다.

```
plot(cars, type = 'b',main="type = b")
```



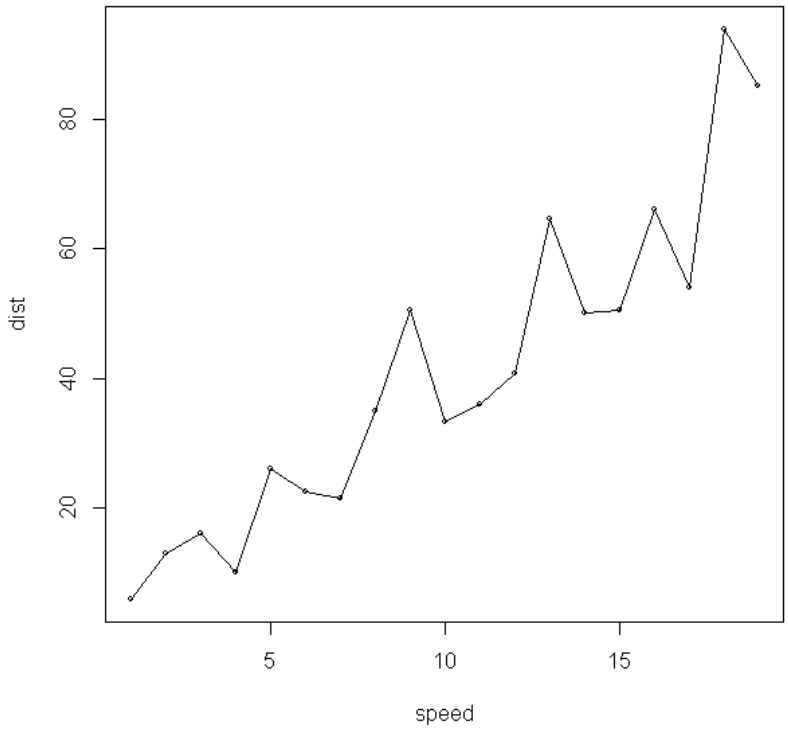

```
plot(cars, type = 'o',main="type = o")
```



위 그림을 보면 같은 주행 속도에 대해 두 개 이상의 제동 거리가 있는 경우가 많아 속도에 따른 제동 거리의 추세가 잘 드러나지 않는다. 이 문제를 해결하기 위해 tapply()를 사용해보자. 각 speed 마다 평균 dist를 tapply를 사용해 계산한 다음 이를 plot()하면 된다.

```
> tapply(cars$dist, cars $speed, mean)
       4        7        8        9       10       11
 6.00000 13.00000 16.00000 10.00000 26.00000 22.50000
      12       13       14       15       16       17
21.50000 35.00000 50.50000 33.33333 36.00000 40.66667
      18       19       20       22       23       24
64.50000 50.00000 50.40000 66.00000 54.00000 93.75000
      25
85.00000
```

```
plot(tapply(cars$dist, cars $speed, mean),
    type = 'o', cex = .5, xlab='speed', ylab = 'dist')
```

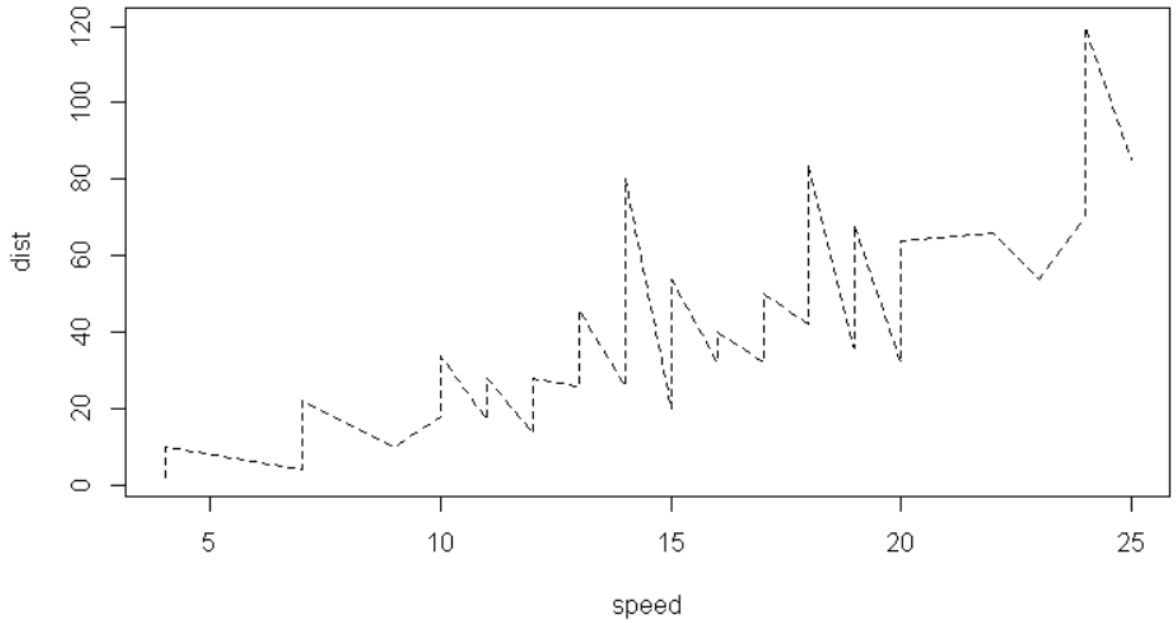


6.2.8 선 유형(lty)

선 유형 지정	의미
plot(X,Y,lty)	lty는 숫자 또는 문자열로 지정할 수 있다. 숫자 일 경우 0 = 그리지 않음, 1 = 실선(기본값), 2 = 대시(-) 표기, 3= 점, 4 = 점과 대시, 5 = 긴 대시, 6 = 두 개의 대시로 지정 가능함 문자열로 지정할 경우 위의 선 모양들은 순서대로, 'blank' , 'solid', 'dashed', 'dotted', 'dotdash', 'longdash', 'twodash'에 해당한다.

다음은 cars 데이터를 그릴 때 type = 'l'을 지정해 선으로 그리되 lty = 'dashed' 를 주면 다음과 같다.

```
plot(cars, type = 'l',lty = 'dashed')
```



6.2.9 그래프의 배열

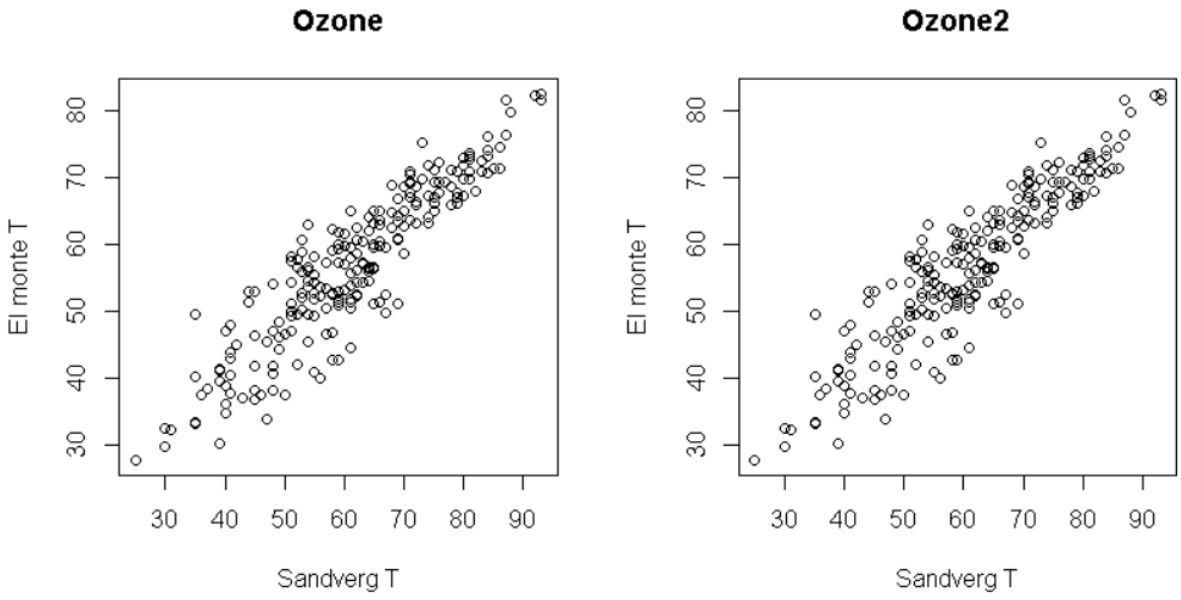
plot() 명령으로 그리는 그래프는 매번 새로운 창에 그려진다. 그러나 여러 그래프가 서로 연관되어 있다면 이들을 한 창에 그리고 싶을 때가 있다. par()는 그래픽 파라미터를 지정하는 함수다.

그래프 배열 방식은 그래픽 파라미터 중 mfrow로 지정한다.

그래프 배열	의미
par(mfrow = c(nr,nc))	그래프를 nr개의 행, nc개의 컬럼으로 배열한다.

par()를 호출하면 이전에 저장된 par 설정이 반환된다. 따라서 par(mfrow = c(nr,nc)) 호출시 반환 값을 opar 변수에 저장했다가 코드의 마지막에서 par(oper)를 호출하면, mfrow 지정 이전의 par 설정으로 환경을 되돌릴 수 있다.

```
oper <- par(mfrow = c(1,2))
plot(Ozone$V8,Ozone$V9,
     xlab = 'Sandverg T',
     ylab = 'El monte T',
     main = 'Ozone')
plot(Ozone$V8,Ozone$V9,
     xlab = 'Sandverg T',
     ylab = 'El monte T',
     main = 'Ozone2')
par(oper) #초기화
```



6.2.10 지터

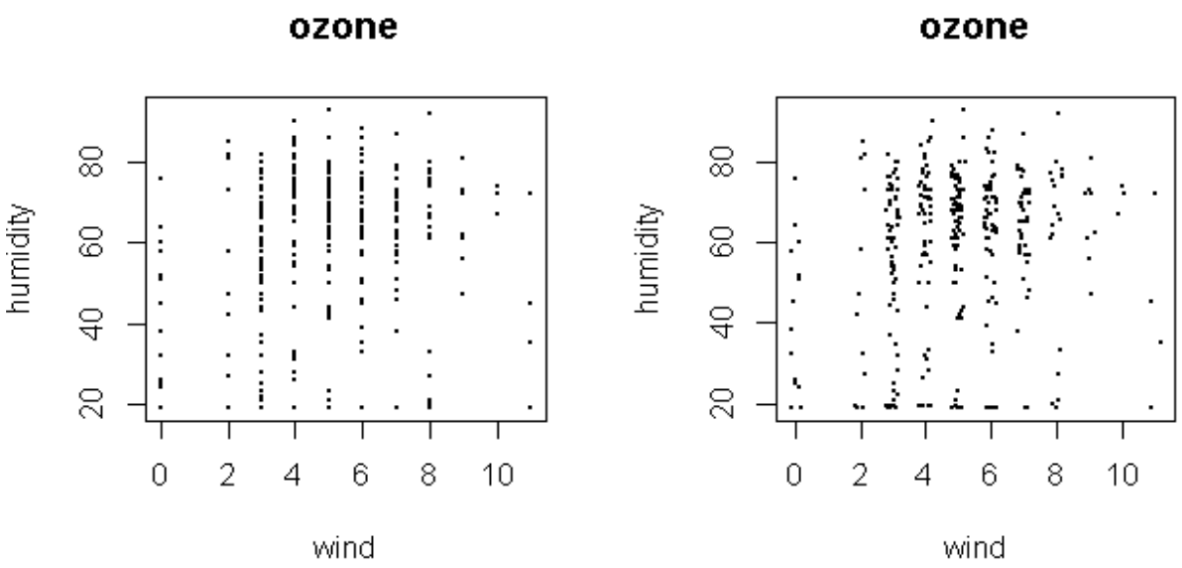
지터는 데이터 값에 약간의 노이즈를 추가하는 방법을 말한다. 노이즈를 추가하면 데이터 값이 조금씩 움직여서 같은 값을 가지는 데이터가 그래프에 여러 번 겹쳐서 표시되는 현상을 막아준다. 다음 표에 jitter() 함수에 대해 정리했다.

같은 값이 여러 번 나타나는 데이터의 예로 Ozone 데이터를 보자. Ozone의 V6과 V7은 각각 LA 공항에서의 풍속과 습도를 담고 있다.

```
> head(Ozone[, c('V6','V7')])
  V6 V7
1  8 20
2  6 NA
3  4 28
4  3 37
5  3 51
6  4 69
```

이처럼 값이 같은 데이터가 많은 경우에 (V6,V7) 순서쌍을 좌표 평면에 그리면 여러 점이 한 위치에 겹쳐서 표시되어 서로 구분이 되지 않는다. 이 경우 지터를 적용하면 데이터 값을 조금씩 움직여 값이 겹치지 않게 된다.

다음 코드는 원본 데이터와 지터를 사용한 경우를 각각 그래프로 그리는 코드다. 다음 그림에서 볼 수 있듯이 지터를 사용하면 데이터가 몰리는 점이 더 쉽게 파악된다.



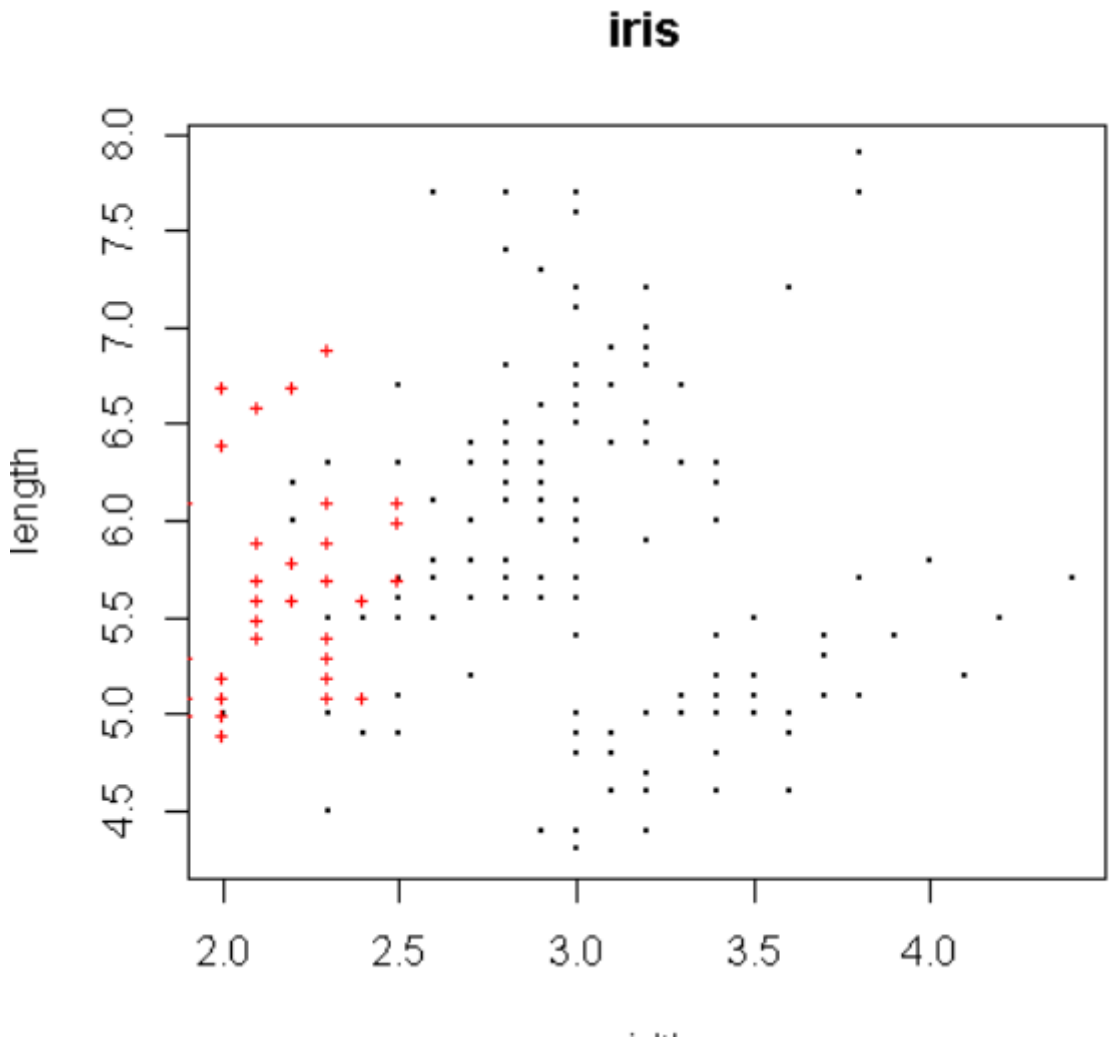
6.3 기본 그래프

이 절에서는 점, 선, 곡선, 다각형과 같은 기본적인 요소들로 그래프를 그리는 방법에 대해 살펴본다. 그 다음 그래프에 그려진 데이터를 식별하는 방법, 범례, 행렬에 저장된 데이터를 그리는 방법을 설명한다.

6.3.1 점(points)

points()는 점을 그리는 함수다. plot()을 연달아 호출하는 경우 매번 새로운 그래프가 그려지는 것과달리 points()는 이미 생성된 plot에 점을 추가로 그려준다. 다음은 아이리스 데이터의 Sepal.Width, Sepal.Length를 plot()으로 그린 다음 Petal.Width, Peral.Length를 같은 그래프 위에 points()로 덧그리는 예다.

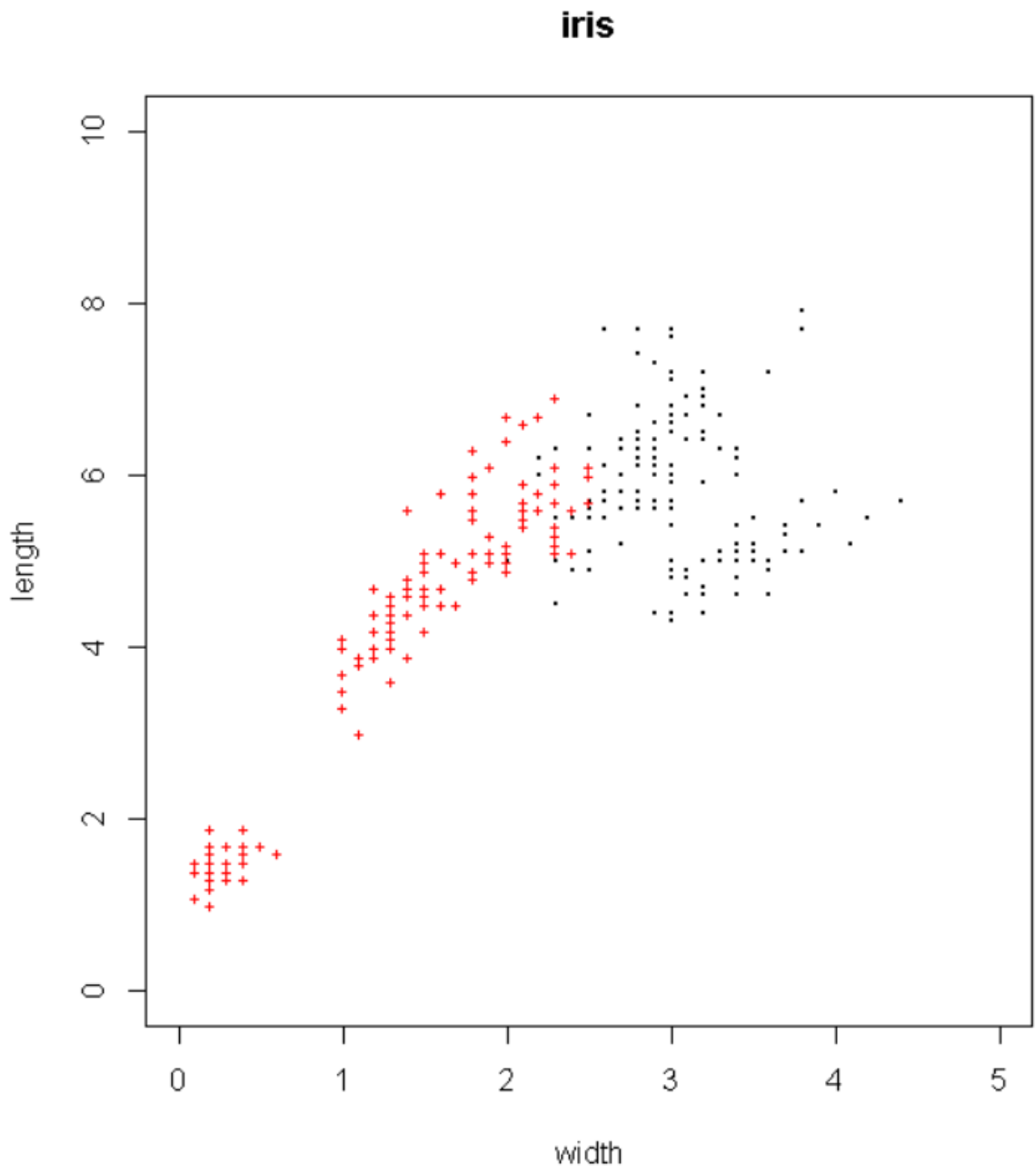
```
plot(iris$Sepal.Width,iris$Sepal.Length, xlab = "width"
        , ylab = 'length', main ='iris', pch = 20, cex = .5)
points(iris$Petal.Width, iris$Petal.Length,
       cex = .5,pch = "+", col = "#FF0000")
```



points()는 이처럼 이미 그려진 그래프에 추가로 점을 표시할 수 있다. 그런데 코드를 작성하다 보면 때에 따라 plot() 문을 수행할 때는 그패르에 표시할 데이터가 없다가, 이후에야 화면에 표시할 데이터가 준비되는 경우가 있다. 이럴 때는 type = 'n' 을 사용하여 plot()을 먼저 수행한다. 그러면 화면에 그려지는 데이터는 없으나 새로운 그래프가 시작된다. 그리고 데이터가 준비되면 points()로 그림을 그리면 된다.

다음은 (Sepal.Width, Sepal.Length), (Petal.Width, Petal.Length)를 두 개의 points() 호출로 그리는 예다.

```
with(iris, {
  plot(NULL, xlim = c(0,5),ylim = c(0,10), xlab = 'width',
  ylab = 'length', main = 'iris', type = 'n')
  points(Sepal.Width,Sepal.Length, cex = .5,pch = 20)
  points(Petal.Width,Petal.Length, cex = .5,pch = "+", col = '#FF0000')
})
```

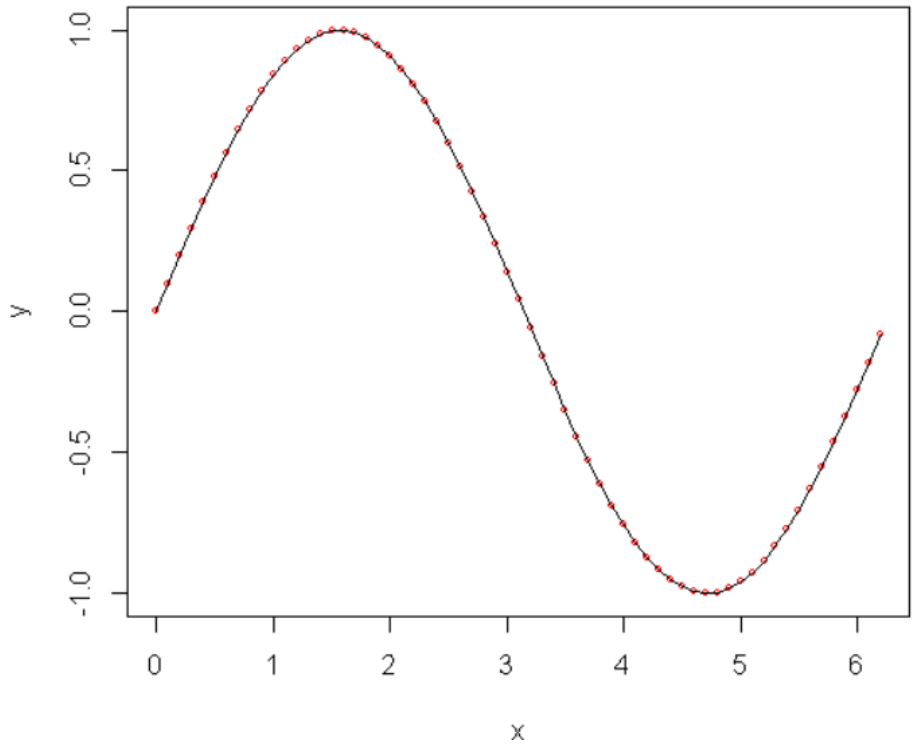


이처럼 type = 'n'을 사용하면 점진적인 방법으로 그래프를 그려나갈 수 있다. 그러나 최초 plot() 호출 시 xlim와 ylim을 적절하게 설정해줘야 하는 번거로움은 있다.

6.3.2 꺾은선(lines)

lines()는 points()와 마찬가지로 plot()으로 새로운 그래프가 시작된 뒤 그 위에 꺾은선을 추가하여 그리는 목적으로 사용한다. 꺾은선은 시계열 데이터에서 추세를 표현하거나, 여러 범주의 데이터를 서로 다른 색상 또는 선 유형으로 표현하는 데 사용한다.

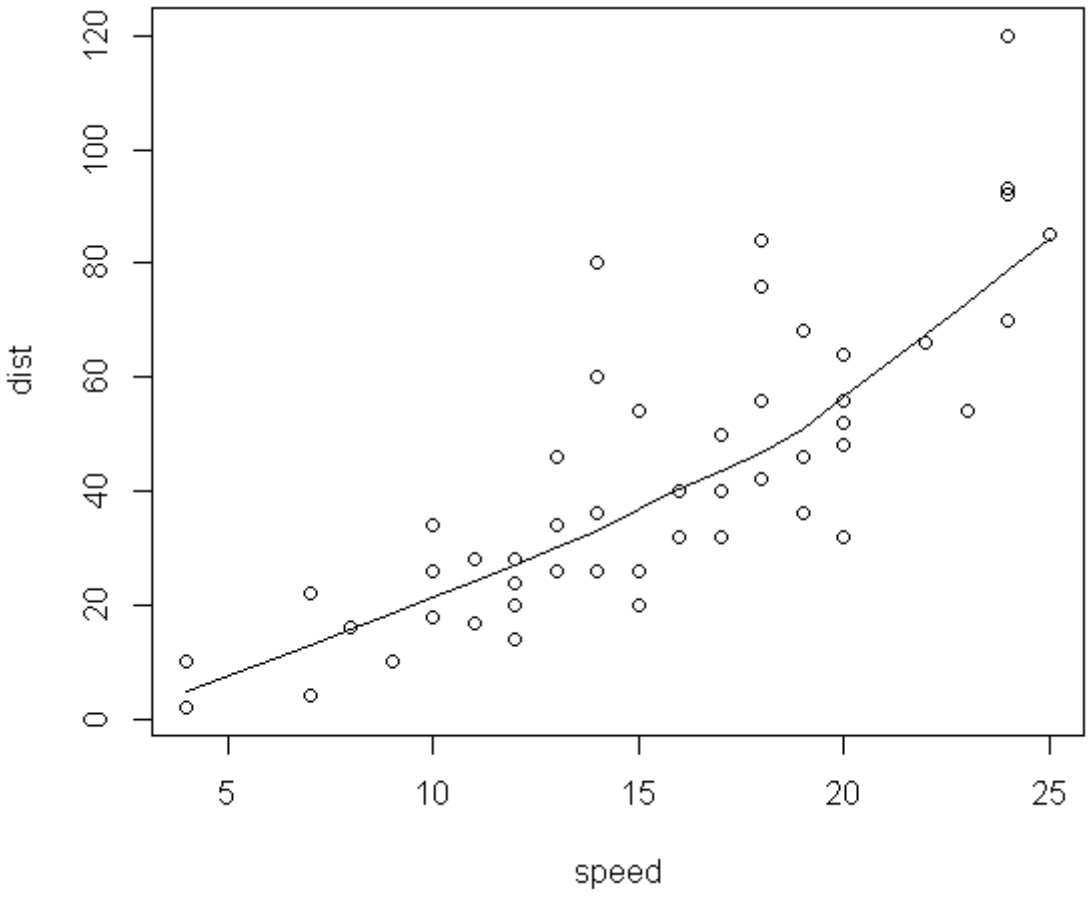
```
x <- seq(0,2*pi,0.1)
y<- sin(x)
plot(x,y,cex = .5,col = 'red')
lines(x,y)
```



또 다른 예로 example(lines)를 입력하면 cars 데이터에 LOWESS를 적용한 예를 볼 수 있다. LOWESS는 데이터를 설명하는 일종의 추세선을 찾는 방법이다. LOWESS는 데이터의 각 점을 y, 각 점의 주변에 위치한 점들을 x라고 할 때, y를 x로부터 추정하는 다항식을 찾는다. LOWESS가 찾는 다항식은 y = ax + b 또는 $y = ax^2 + bx + c$ 와 같은 저차 다항식이다. 다항식을 찾을 때는 추정하고자 하는 y에 가까운 x일수록 더 큰 가중치를 준다. 이런 이유로 LOWESS는 지역 가중 다항식 회귀라고 부르며, 그 결과는 각 점을 그 주변 점들로 설명하는 다항식들이 연결된 모양이 된다. 이렇게 각 점에서 찾아진 다항식들을 부드럽게 연결하면 데이터의 추세를 보여주는 선이 된다.

cars 데이터에 대해 LOWESS를 수행해보자.

```
> data(cars)
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```



Note_ 또다른 비모수적 회귀 분석 함수들

R은 이외에도 loess(), ksmooth(), smooth.spline(), earth() 등의 비모수적 회귀 방법을 제공한다.

6.3.3 직선(abline)

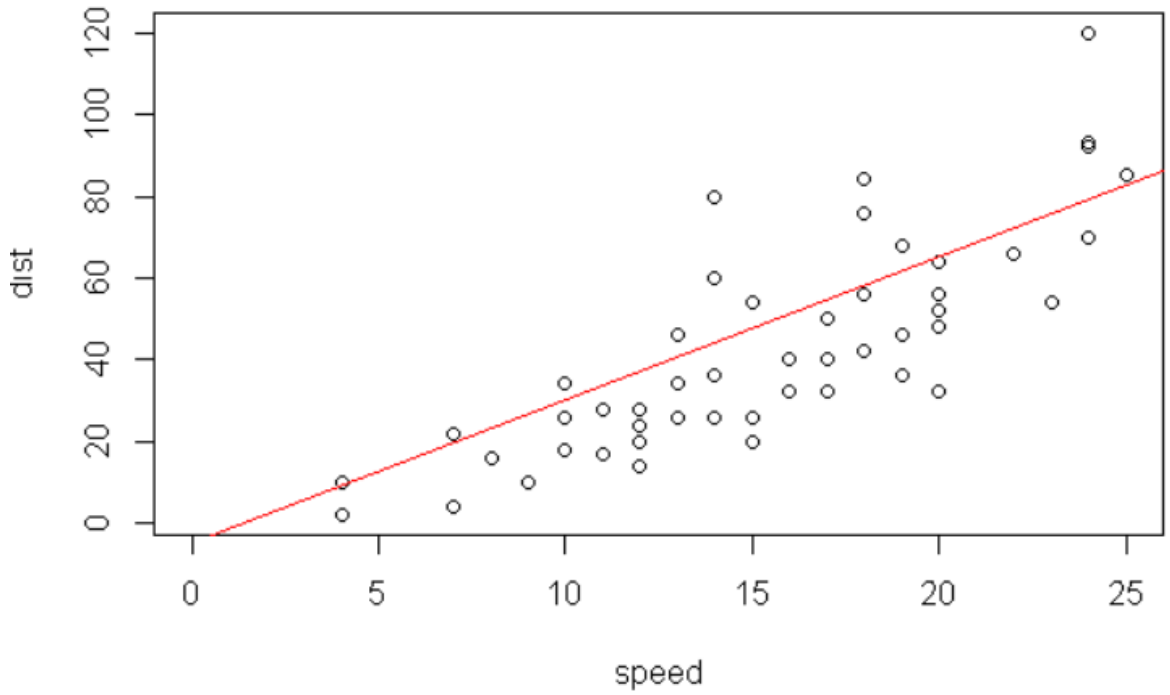
abline()은 y=ax +b 형태의 직선이나 y = h 형태의 가로로 그은 직선 또는 x = v 형태의 세로로 그은 직선을 그래프에 그린다. abline()이 그리는 것은 꺾이지 않고 그어진 일직선이다. 따라서 lines()가 주어진 (x,y) 좌표들을 연결하는 꺾은선을 긋는 것과는 차이가 있다.

abline : 그래프에 직선을 추가하여 그린다.

```
abline(
    #a,b 는 y = a + bx 형태의 직선을 그릴 때 절편과 기울기
    )
```

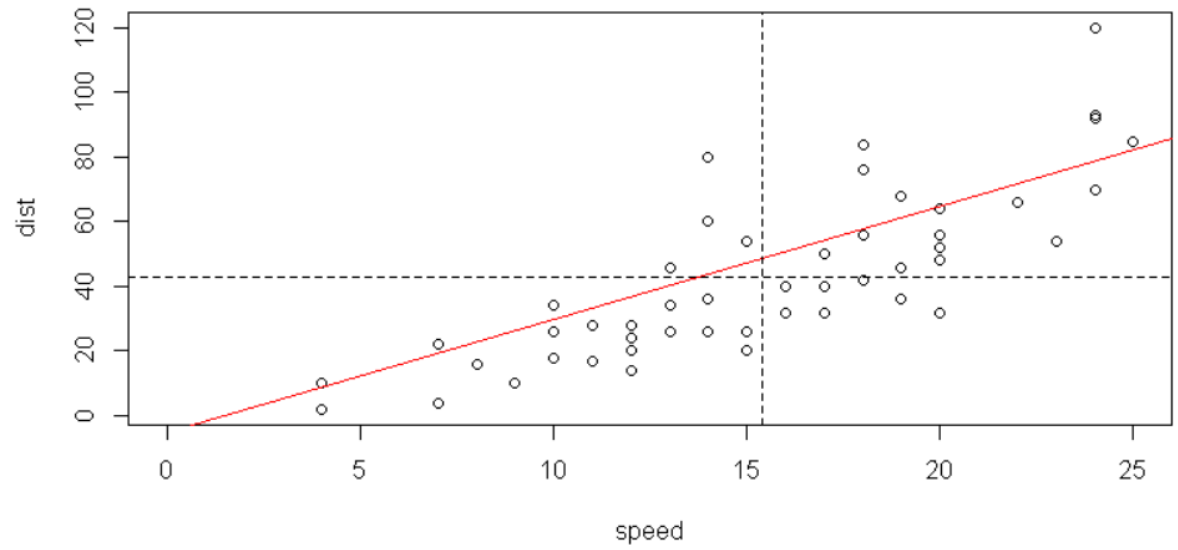
cars 데이터에서 제동거리와 속도가 dist = -5 + 3.5 x speed로 근사될 수 있다고 가정해보자. 그러면 다음과 같이 abline()을 사용하여 근사가 얼마나 잘 이뤄지는지를 시각화 할 수 있다.

```
plot(cars, xlim = c(0,25))
abline(a=-5,b = 3.5, col = 'red')
```



이 그래프에 speed와 dist의 평균까지 표시해보자. 다음 코드에서 lty는 선의 유형을 의미한다.

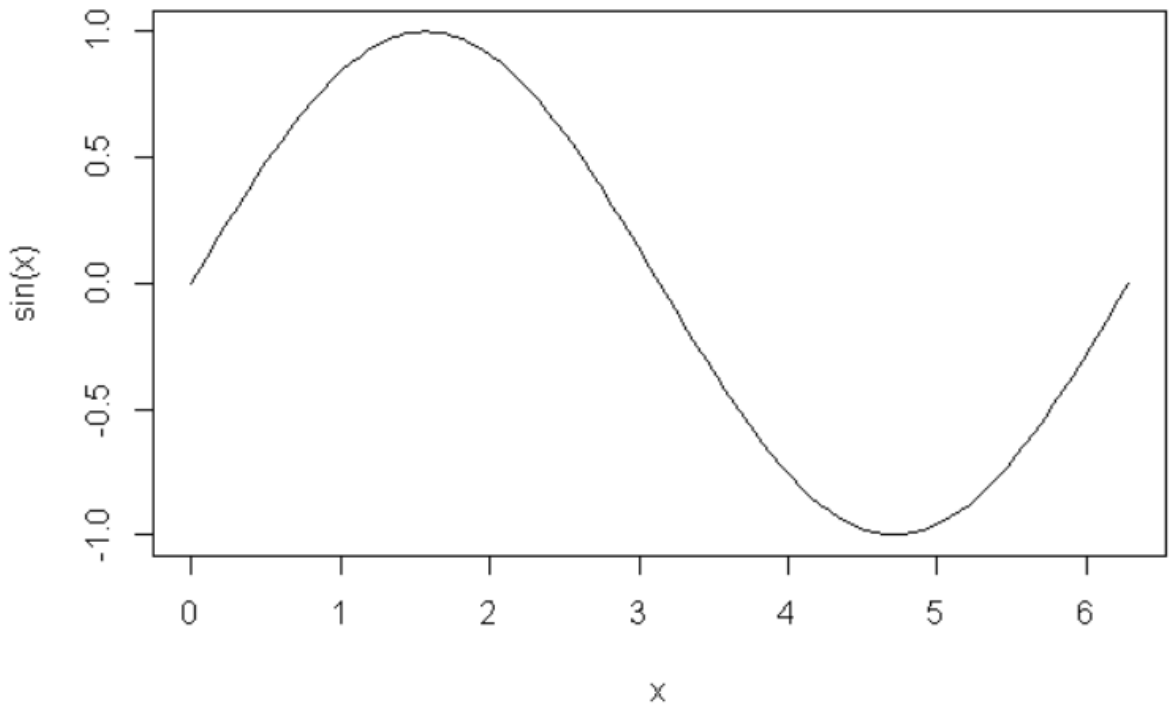
```
plot(cars, xlim = c(0,25))
abline(a=-5,b = 3.5, col = 'red')
abline(v= mean(cars$speed),lty = 2)
abline(h= mean(cars$dist),lty = 2)
```



6.3.4 곡선(curve)

curve()는 주어진 표현식에 대한 곡선을 그리는 함수다. lines 절에서 사인 곡선의 좌표르 미리 생성한 다음, 이를 lines()에 지정해 사인 그래프를 그리는 예를 살펴봤지만, curve()를 사용하면 이를 좀 더 직접적으로 표현할 수도 있다. 특히 curve()는 인자로 표현식, 시작점, 끝점을 받으므로 표현이 좀 더 편리하다.

```
curve(sin, 0, 2*pi)
```



6.3.5 다각형

ploygon()은 다각형을 그리는 데 사용하는 함수다.

선형 회귀 함수

lm : 데이터로부터 선형 모델을 만든다.

```
lm(
    formula, # 종속변수 ~ 설명변수
    data     # 포뮬러에 따라 선형 모델을 만들 데이터
    )
#반환 값은 선형 모델 객체다.
```

predict.lm : 예측을 수행한다.

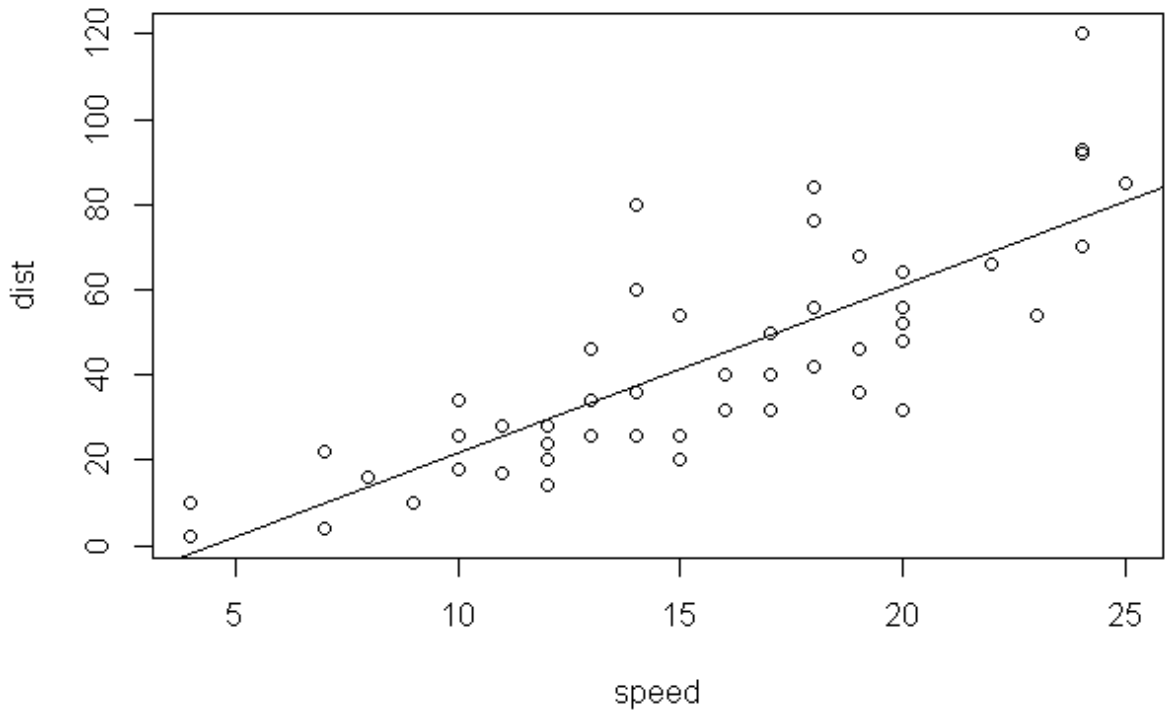
```
predict.lm(
    object,      # 선형 모델 객체
    newdata      # 예측을 수행할 새로운 데이터
                 # 이 값을 생략하면 선형 모델을 만들 때 사용한 데이터가 사용된다.
    )
#반환 값은 선형 모델 객체다.
```

cars에 대한 선형 회귀를 수행해보자. lm()에서 포뮬러는 dist로 주어졌는데, 이는 dist = a + b x speed + ϵ의 모델을 의미한다. 수식에서 a는 절편, b는 기울기 ϵ은 dist와 a + b x speed 사이의 차, 즉 오차를 뜻한다.

```
> (m<-lm(dist~speed, data = cars))

Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed
    -17.579        3.932
```

회귀 모델(모형)을 적합한 결과 dist = -17.579 + 3.932 x speed + ϵ 의 식이 얻어졌다. 앞에서도 설명했듯이 lm()으로 만든 모델은 단순히 abline() 함수에 넘겨주는 것만으로 그래프에 표시할 수 있다.

```
> plot(cars)
> abline(m) # abline()의 reg 인자에 선형 회귀 모델 m이 지정됨
```



선형 회귀 모델이 주어졌을 때 새로운 데이터에 대한 예측은 predict()로 수행한다. 함수 호출 시 인자로 interval = 'condifence' 를 지정해 신뢰 구간까지 구해보자.

```
> p <- predict(m , interval = 'confidence')
> head(p)
        fit        lwr       upr
1 -1.849460 -12.329543  8.630624
2 -1.849460 -12.329543  8.630624
3  9.947766   1.678977 18.216556
4  9.947766   1.678977 18.216556
5 13.880175   6.307527 21.452823
6 17.812584  10.905120 24.720047
```

위의 코드에서 예측 결과를 저장한 p의 데이터 타입은 행렬이며, fit은 회귀 모델로 적합된 y값, lwr은 신뢰 구간의 하한, upr은 신뢰 구간의 상한을 뜻한다. p의 각 행은 cars의 각 행에 대응한다. 예를 들어, p의 첫 행에 있는 fit 값 -1.849460은 cars의 첫 행에 있는 speed 값 4에 대한 dist 예측값이다.

polygon()으로 신뢰 구간을 그리려면 그래프에 그릴 다각형의 x좌표, y좌표를 구해야 한다. 이는 cars의 speed를 x좌표, 앞서 코드에서 구한 p의 lwr 과 upr을 각각 y좌표로 한 점들을 나열해 구할 수 있다. 단, 닫혀 있는 다각형을 그려야 하므로 시작점과 끝점이 만나야 한다는 점을 유의한다. 이를 표현한 코드는 다음과 같다.

```
x <- c(cars$speed,
       tail(cars$speed,1),
       rev(cars$speed),
       cars$speed[1])

y <- c(p[,"lwr"],
       tail(p[,"upr"],1),
       rev(p[,"upr"]),
       p[,"lwr"][1])
```

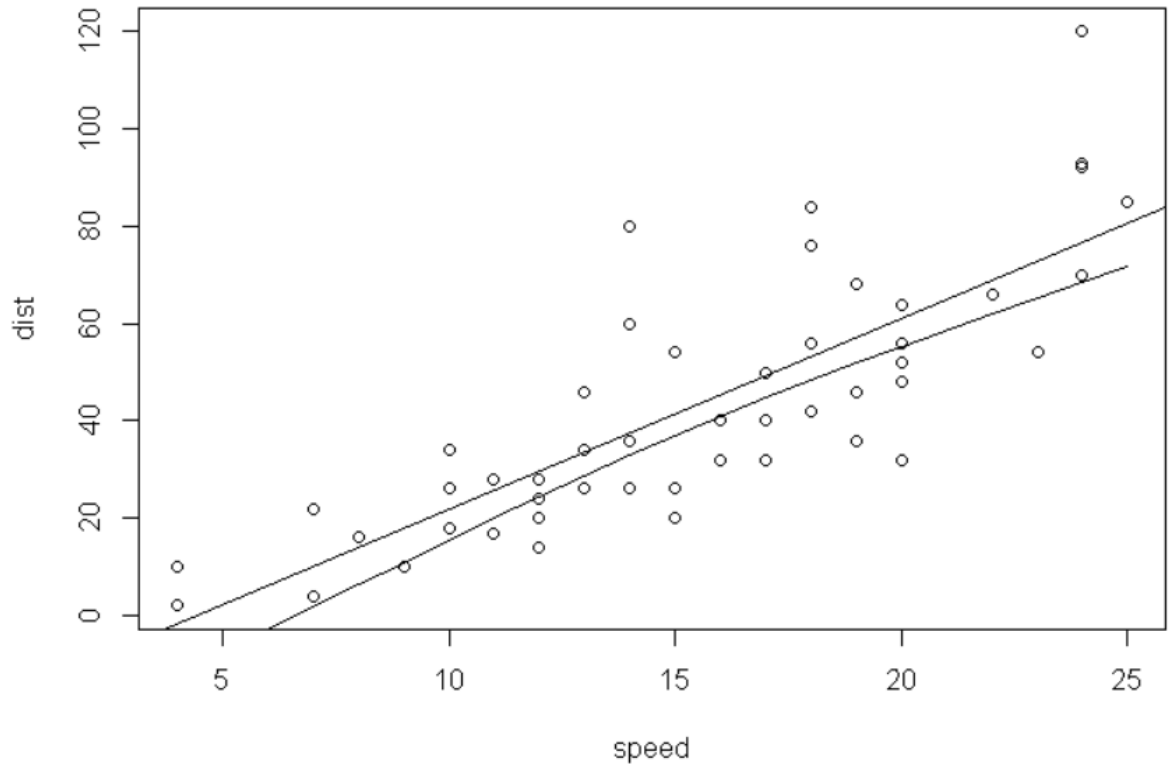
이 코드는 다음과 같이 이해할 수 있다. 먼저 (cars$speed,p[,"lwr"])의 좌표들을 나열한다. 그러면 하한에 대한 선이 완성된다. 다음으로 (cars$speed,1)의 가장 마지막 값, p[,'upr']의 가장 마지막 값으로 선을 그린다. 이때 사용한 tail()은 head()의 반대 기능을 하는 함수로, 데이터의 가장 마지막 값을 얻어오는데 사용했다. 이제 (cars$speed, p[,'upr'])의 점들을 따라 선을 그리되 그래프의 우측에서 좌측으로 그려나간다. 이는 rev() 함수를 사용해 수행한다. 마지막으로 시작점인 (cars$speed의 첫 번째 값, p[,"lwr"]의 첫번째 값)으로 선을 그려 도형을 닫는다.

신뢰 구간을 그래프에 그리는 것이므로, 이 영역을 회색으로 색칠하는 것이 보기에 좋을 것이다.

그러나 col = 'gray' 와 같이 색을 지정하면, ploygon() 호출 이전에 그래프에 그려진 내용이 모두 가려져 버린다. 따라서 rgb() 함수에 알파 값을 지정해 투명한 색을 사용한다.

전체 코드를 살펴보자.

```
m <- lm(dist~speed, data = cars)
p <- predict(m, interval = 'confidence')
x <- c(cars$speed,
       tail(cars$speed,1),
       rev(cars$speed),
       cars$speed[1])

y <- c(p[,"lwr"],
       tail(p[,"lwr"],1),
       rev(p[,"lwr"]),
       p[,"lwr"][1])

plot(cars)
abline(m)
polygon(x,y,col='#ff0000')
```

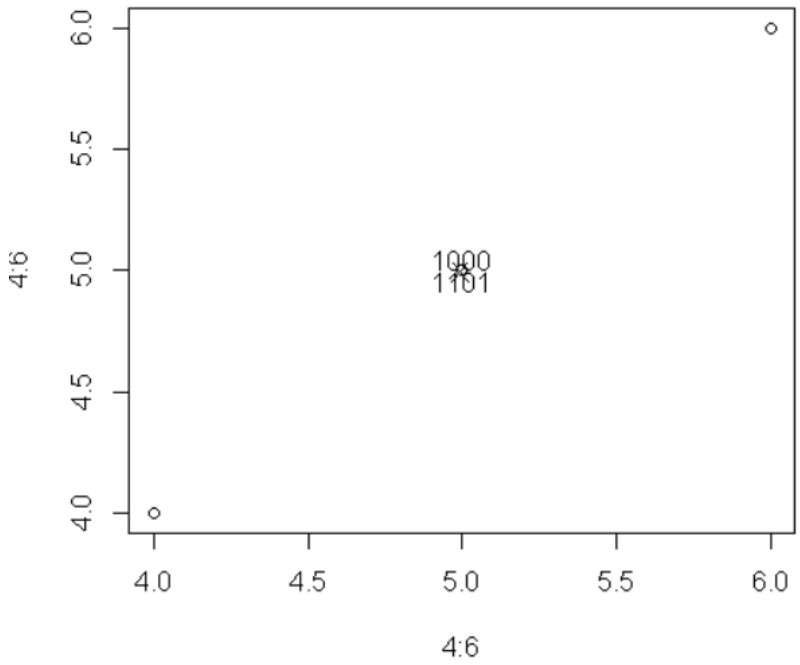


6.4 문자열(text)

text()는 그래프에 문자열을 그리는 데 사용한다.

text : 그래프에 문자열을 추가한다.

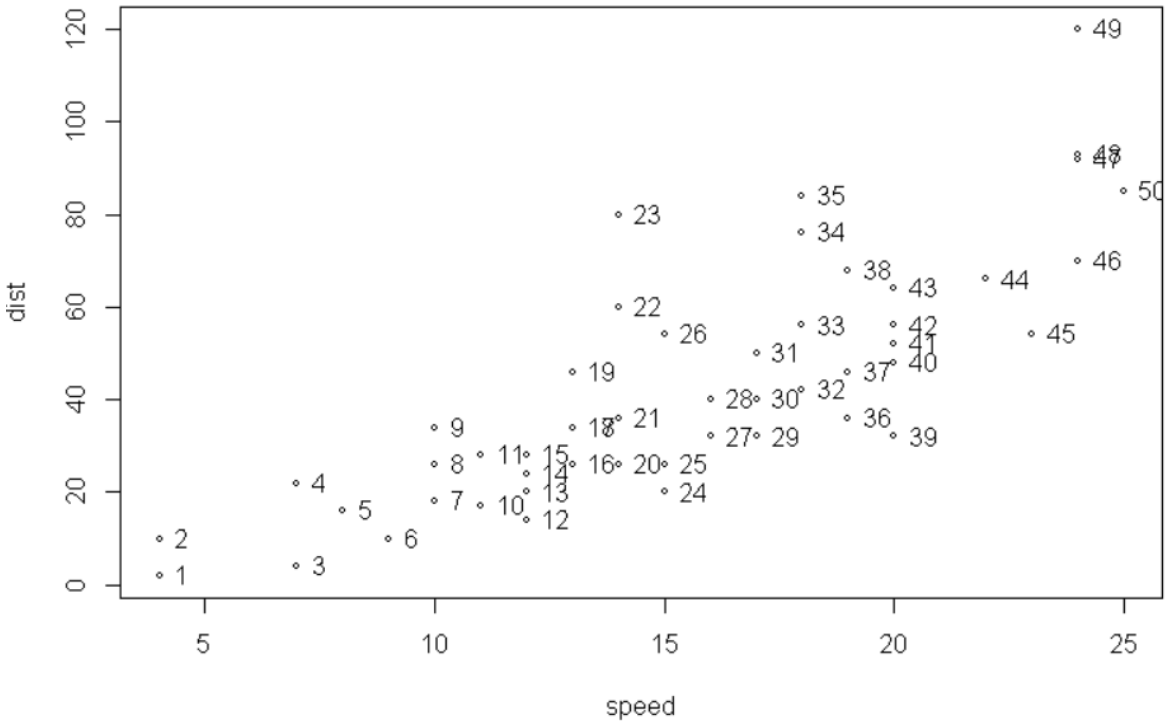
```
plot(4:6, 4:6)
text(5,5,'X')
text(5,5,'00', adj = c(0,0))
text(5,5,'01', adj = c(0,1))
text(5,5,'10', adj = c(1,0))
text(5,5,'11', adj = c(1,1))
```



실행 결과 (5,5) 위치에 문자열을 출력하더라고 adj에 따라 문자열 위치가 조정되는 것을 볼 수 있다.

text()에서 labels 값이 주어지지 않을 경우 기본값은 seq_along(x)이다. seq_along(x)은 1,2,3, ..., NROW(x) 까지의 정수를 반환하는 함수다. 따라서 label를 지정하지 않으면 (x,y) 좌표들에 데이터 순서에 따라 번호를 붙인다. 다음은 cars 데이터의 각 점에 데이터 번호를 표시하는 예다.

```
plot(cars, cex = .5)
text(cars$speed, cars$dist, pos = 4)
```



이처럼 text()를 사용하면 점들에 번호를 붙여 각 점이 어느 데이터에 해당하는지 쉽게 알 수 있다.

그러나 데이터가 몰려 있는 점에서 숫자가 겹쳐서 표시되므로, 각 점이 어느 데이터에 해당 하는지 구분이 어렵다. 또, 항상 모든 점에 레이블이 필요한 것은 아니다. 이런 경우에는 다음 절에서 설명할 indentify()를 사용한다.

6.5 그래프에 그려진 데이터의 식별

그래프를 그린 뒤 identidify()를 호출하면 마우스 커서가 십자 모양으로 변한다.

이 십자 커서로 그래프에서 특정 점을 클릭하면 클릭된 점과 가장 가까운 데이터의 레이블을 표시한다. 마우스 클릭을 중단하려면 그래프가 나타난 창을 닫으면 된다.

다음은 cars 데이터의 좌표를 식별하는 예다.

```
plot(cars, cex = .5)
identify(cars$speed, cars$dist)
```

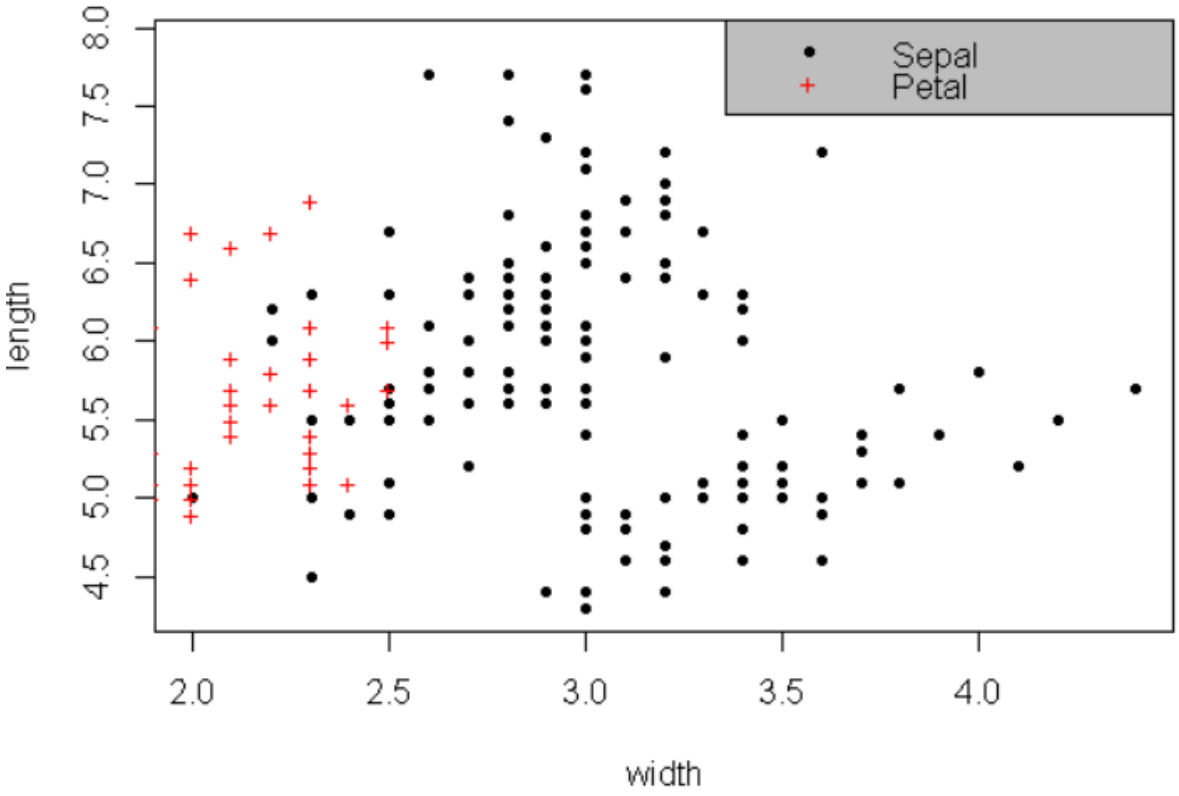
6.6 범례(legend)

legend()는 범례를 표시하는 데 사용한다.

다음 코드에서는 legend()의 위치를 topright로 했고, 범례는 Sepal.Petal을 보였다. Sepal은 pch 20 (원 형기호), Petal은 pch 43(+ 기호)으로 지정했으며, Sepal은 색상 black, Petal은 색상 red 로 지정했다. bg는 회색 배경을 의미함.

한 가지 주의할 점은 Petal을 points()로 그릴 때 pch = '+'를 사용했음에도 불구하고 legend에서 pch를 지정할 때는 pch = (20, '+')가 아닌 pch = (20,43)을 사용해야 한다는 것임. 이는 벡터에는 한 가지 타입의 입자만 저장할 수 있기 때문이다.

```
plot(iris$Sepal.Width, iris$Sepal.Length, pch = 20, xlab = 'width', ylab =
'length')
points(iris$Petal.Width,iris$Petal.Length, pch = "+", col = "#FF0000")
legend("topright", legend=c("Sepal", "Petal "), pch = c(20,43), col = c("black",
"red"), bg = "gray")
```



6.7 행렬에 저장된 데이터 그리기 (matplot, matlines, matpoints)

matplot(), matlines(), matpoints()는 각각 plot(), lines(), points() 함수와 유사하지만 행렬 형태로 주어진 데이터를 그래프에 그린다는 점에서 차이가 있다. 함수 호출 방식 역시 큰 차이가 없고, 다만 입력이 행렬로 주어지기만 하면 되낟.

[-2π, 2π] 구간에서의 cos(x), sin(x) 그래프를 matplot()을 사용해 그려보자. x값은 다음과 같이

[-2π, 2π] 사이의 촘촘한 점들로 정한다.

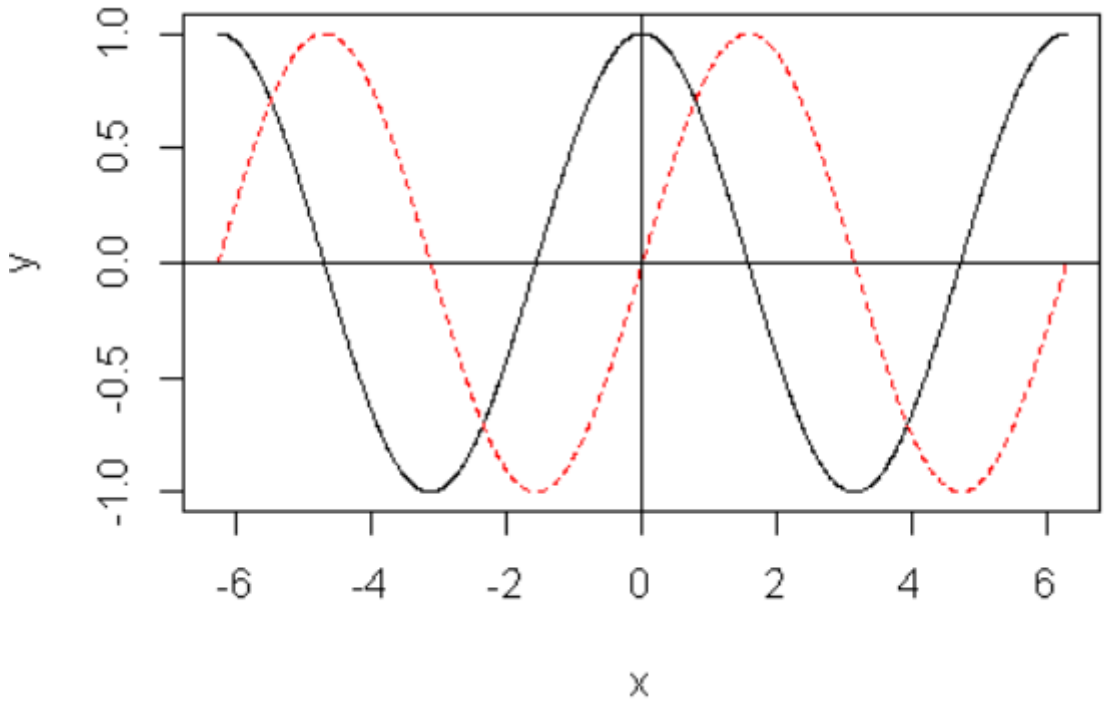
```
(x<- seq(-2*pi, 2*pi, 0.01))
head(x)
[1] -6.283185 -6.273185 -6.263185 -6.253185
[5] -6.243185 -6.233185
```

y축은 cos(x), sin(x)를 저장한 행렬로 만든다.

```
> (y<- matrix(c(cos(x), sin(x)), ncol = 2))
                [,1]          [,2]
   [1,]  1.0000000000  2.449213e-16
   [2,]  0.9999500004  9.999833e-03
   [3,]  0.9998000067  1.999867e-02
   [4,]  0.9995500337  2.999550e-02
   [5,]  0.9992001067  3.998933e-02
   [6,]  0.9987502604  4.997917e-02
   [7,]  0.9982005399  5.996401e-02
   [8,]  0.9975510003  6.994285e-02
   [9,]  0.9968017063  7.991469e-02
```

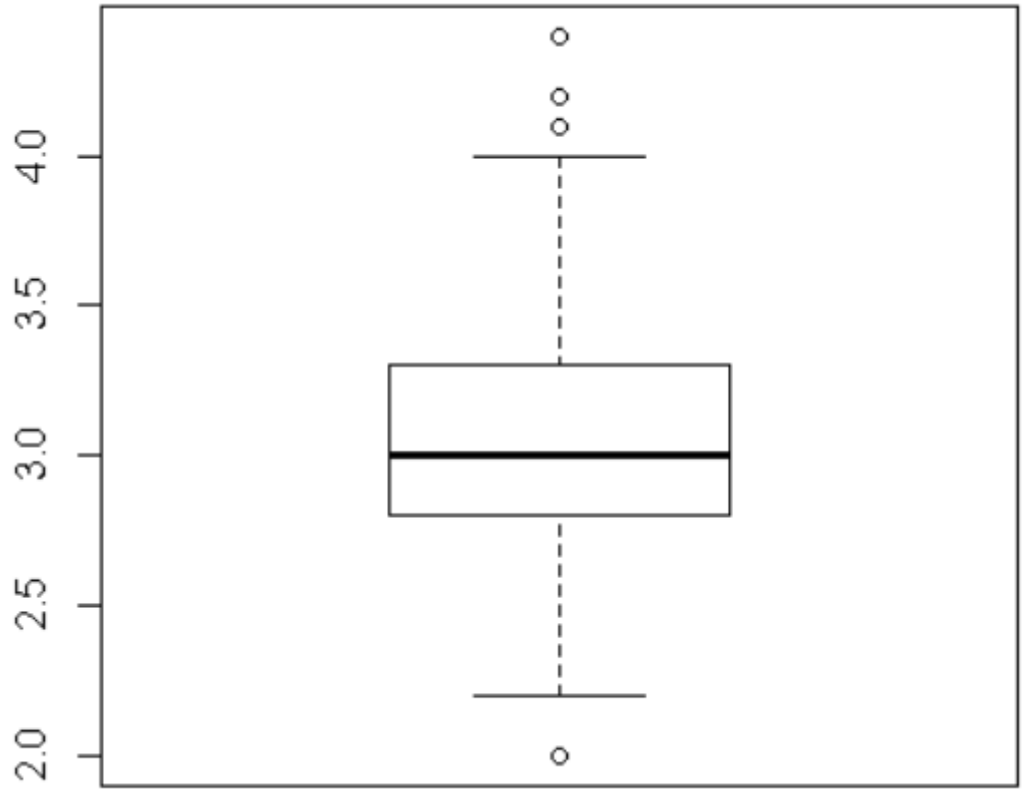
마지막으로 matplot()에 x,y를 지정해 그래프를 그리고 abline()으로 x=0, y=0의 두 축을 그려넣는다.

```
matplot(x,y,lty = c("solid", "dashed"), cex = .2, type = "l")
abline(h=0, v=0)
```



6.8 응용그래프

6.8.1 상자그림(boxplot)



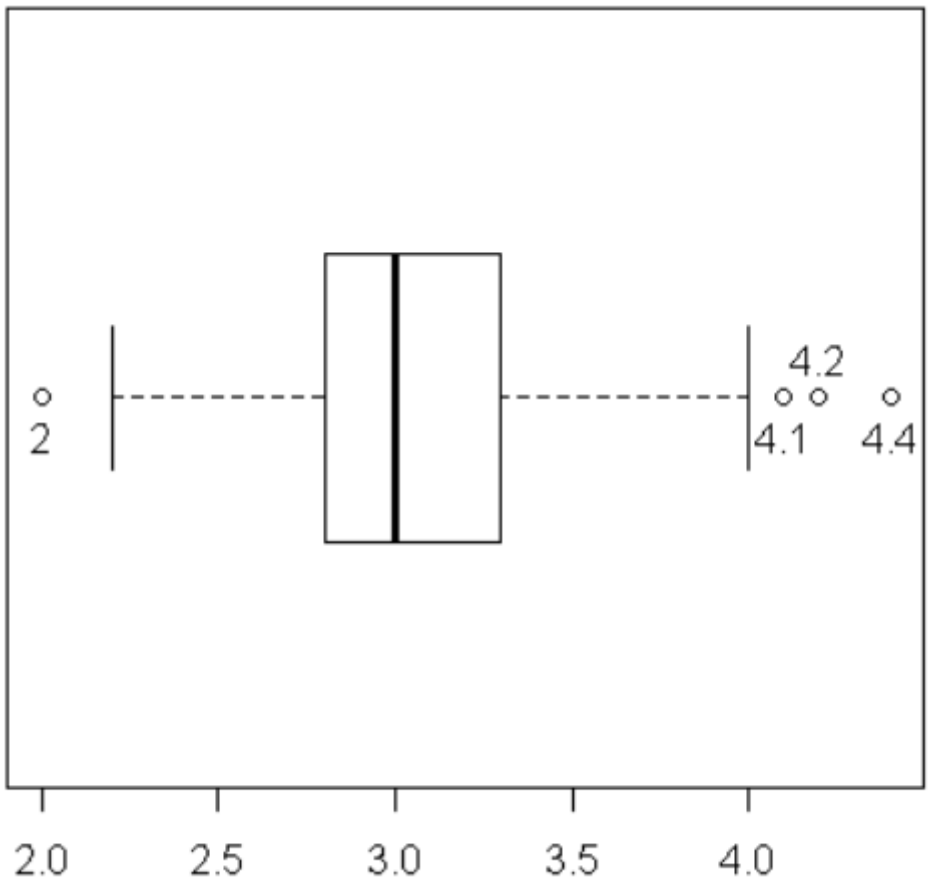

```
boxplot(iris$Sepal.Width)
```

상자그림에 표시된 이러한 값들을 정확히 확인하려면 boxplot()의 반환 값을 보면 된다. boxplot()의 반환 값은 리스트로, stats는 (lower whisker, lower hinge, 중앙값, upper hinge, upper whisker)를 포함하고 있고, out은 outlier를 저장하고 있다.

```
> (boxstats<-boxplot(iris$Sepal.Width))
$stats
     [,1]
[1,]  2.2
[2,]  2.8
[3,]  3.0
[4,]  3.3
[5,]  4.0

$n
[1] 150

$conf
         [,1]
[1,] 2.935497
[2,] 3.064503

$out
[1] 4.4 4.1 4.2 2.0

$group
[1] 1 1 1 1

$names
[1] "1"
```

이 정보를 사용해 앞서 보인 iris의 outlier 옆에 데이터 값을 표시해보자.

```
(boxstats<-boxplot(iris$Sepal.Width))

boxstats<- boxplot(iris$Sepal.Width, horizontal = T)
text(boxstats$out, rep(1, NROW(boxstats$out)), labels= boxstats$out, pos =
c(1,1,3,1))
```



boxplot() 호출 시 지정한 horizontal = T는 상자 그림을 가로로 그리게 한다. text()를 호출 할 때 y좌표는 rep(1, NROW(boxstats$out))을 사용했는데, 이 값은 boxstats$out의 길이가 4이므로 1이 4회 반복된 벡터인 c(1,1,1,1)이 된다. 즉, 문자열의 위치를 (outlier 값, 1) 로 잡은 것이다. pos = c(1,1,3,1)은 문자열을 점의 하단, 하단, 상단, 하단에 표시하라는 의미다.

bloxplot()에는 또 한가지 흥미로운 notch 라는 인자를 지정할 수 있다. 이 값을 지정하면 중앙값에 대한 신뢰 구간이 오목하게 그려진다. 따라서 2개의 상자 그림을 나란히 그렸을 때 만약 두 상자 그림의 notch가 겹친다면 두 상자 그림의 중앙값이 다르지 않다고 볼 수 있다. 만약 겹치지 않는다면 두 상자 그림의 중앙값이 유의하게 다르다고 본다.

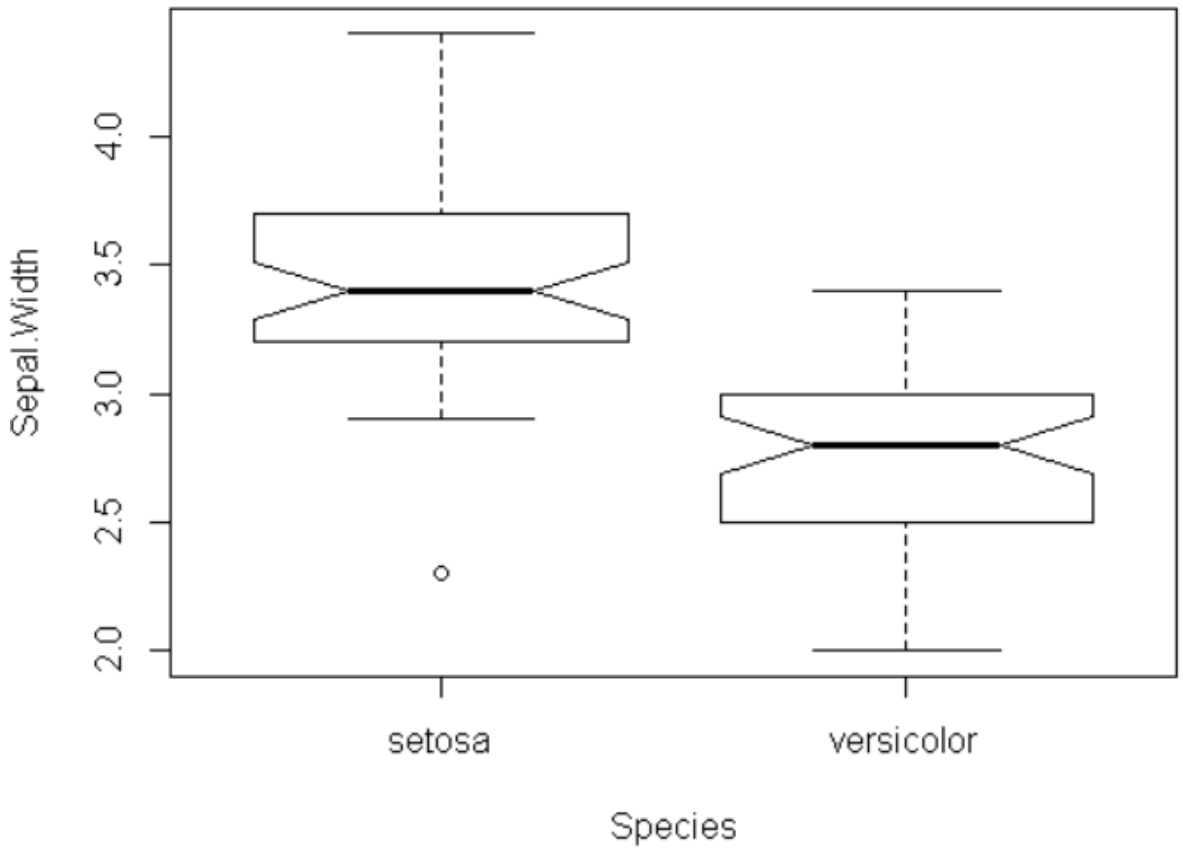
iris의 setosa 종과 versicolor 종의 Sepal.Width 에 대한 상자 그림을 그린다음, 이 두 종의 중앙값이 다른지 비교해보자.

```
sv<-subset(iris, Species=="setosa"|Species=="versicolor")
sv$Species <- factor(sv$Species)
boxplot(Sepal.Width~Species,data=sv, notch = T)
```

위 코드는 먼저 iris에서 Species가 setosa 또는 versicolor인 행을 선택했다. 이때 사용하는 OR연산자는 벡터연산이므로, ||가 아닌 |임에 유의하자.

코드의 두번째 줄에서는 sv$Species를 다시 한 번 factor로 변환했다. 이미 팩터인데 어째서 다시 팩터로 변환할까? 그 이유는 subset이 비록 두 개의 Species 만 선택하는 역활은 하지만, Species 변수 자체를 바꾸지는 않기 때문이다. sv$Species는 subset() 후에도 여전히 레벨이 setosa, versicolor, virginica이다. 따라서 실제 존재하는 레벨만 남기고 지우기 위해 다시 한 번 팩털고 변환해줬다. 그 결과 다음 그림의 가로축에는 setosa와 versicolor만 나열되어 있다. 만약 남아 있는 virginica를 없애주지 않으면 텅 빈 virginica 컬럼이 추가로 보이게 된다.

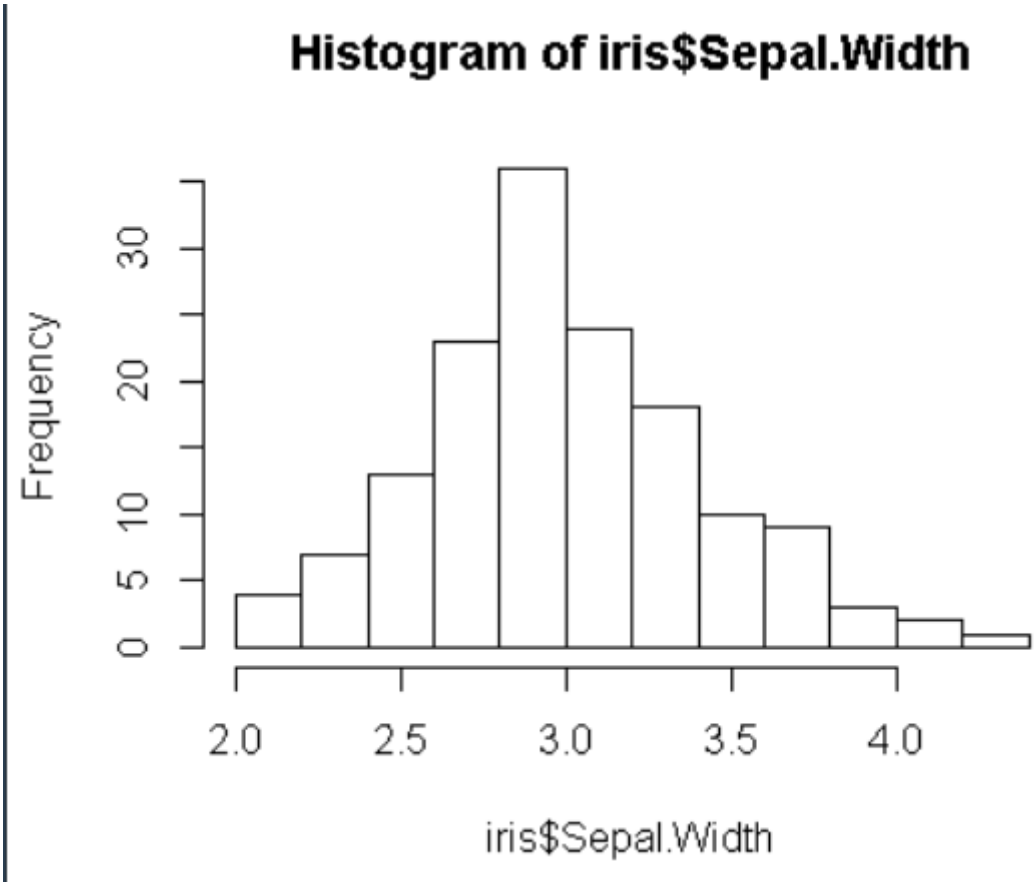
마지막으로 세 번째 줄에서는 Sepal.Width의 상자 그림을 Species 마다 그리기 위해 Sepal.Width ~ Species를 지정했다.



그림을 보면 setosa 종의 Sepal.Width가 전반적으로 versicolor 종의 Sepal.Width 보다 큼을 알 수 있다.

6.8.2 히스토그램

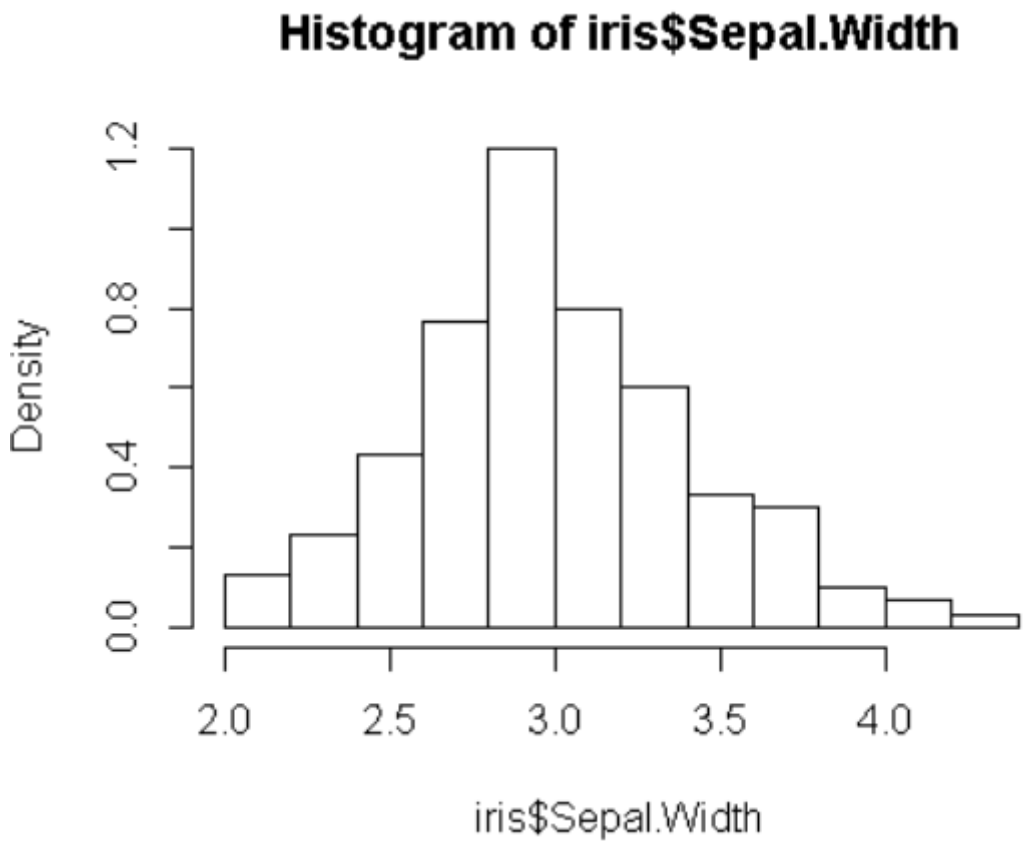
다음은 iris$Sepal.Width에 대한 히스토그램을 그리는 예다.




```
hist(iris$Sepal.Width)
```

hist()의 주요 파라미터 중 하나는 freq이다. freq의 기본값은 NULL이며, 값을 지정하지 않으면 히스토그램 막대가 각 구간별 데이터의 개수(븍, 빈도)로 그려진다. 만약 이 값이 FALSE 면 다음 코드에서 보다시피 각 구간의 확률 밀도가 그려진다. 확률 밀도이므로 막대 너비의 합이 1이 된다.

```
hist(iris$Sepal.Width, freq =  F)
```



히스토그램을 그리면서 계산된 다양한 통계값은 hist()의 반환 값에 리스트로 저장되어 있다. 예를 들어, 다음 코드처럼 hist()의 반환 값을 사용해 densiry값의 합이 1임을 확인해볼 수 있다.

```
> (x<- hist(iris$Sepal.Width, freq = F))
$breaks
 [1] 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0
[12] 4.2 4.4

$counts
 [1]  4  7 13 23 36 24 18 10  9  3  2  1

$density
 [1] 0.13333333 0.23333333 0.43333333 0.76666667
 [5] 1.20000000 0.80000000 0.60000000 0.33333333
 [9] 0.30000000 0.10000000 0.06666667 0.03333333

$mids
 [1] 2.1 2.3 2.5 2.7 2.9 3.1 3.3 3.5 3.7 3.9 4.1
[12] 4.3

$xname
[1] "iris$Sepal.Width"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```

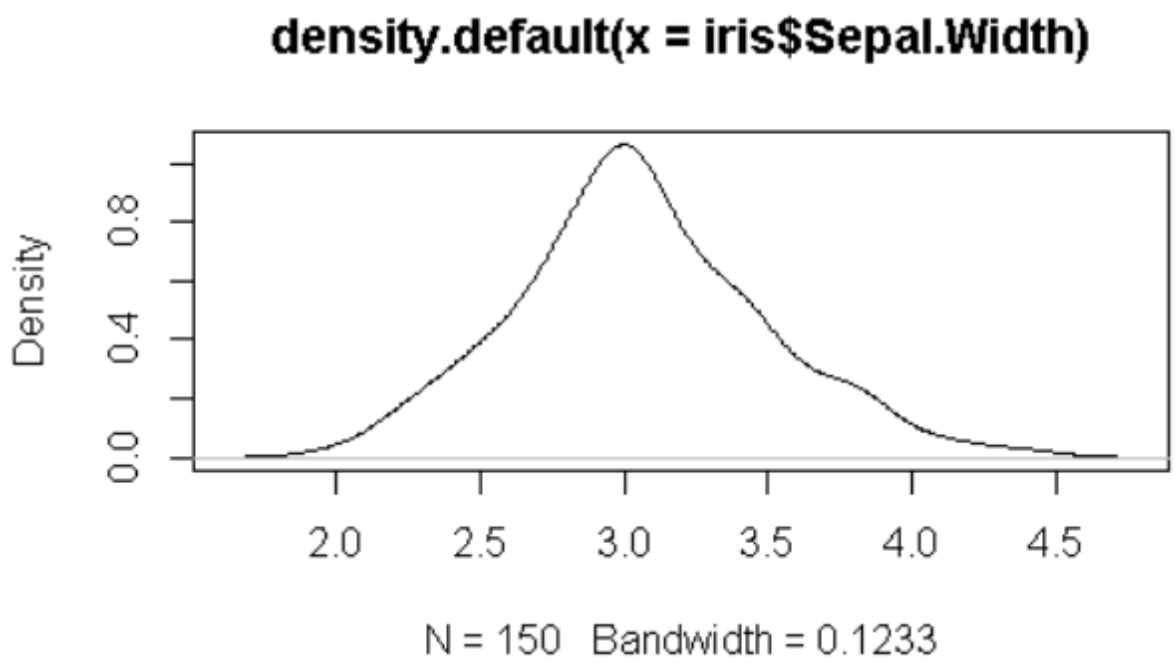
6.8.3 밀도 그림(density)

데이터의 분포를 살펴보는 그래프로는 히스토그램이 가장 잘 알려져 있지만, 히스토그램은 막대를 그리는 구간의 너비를 어떻게 잡는지에 따라 전혀 다른 모양이 될 수 있다는 단점이 있다. densiry()로 그리는 밀도 그림은 막대의 너비를 가정하지 않고 모든 점에서 데이터의 밀도를 추정하는 커널 밀도 추정(Kernel Density Estination) 방식을 사용해서 이러한 문제를 피한다.

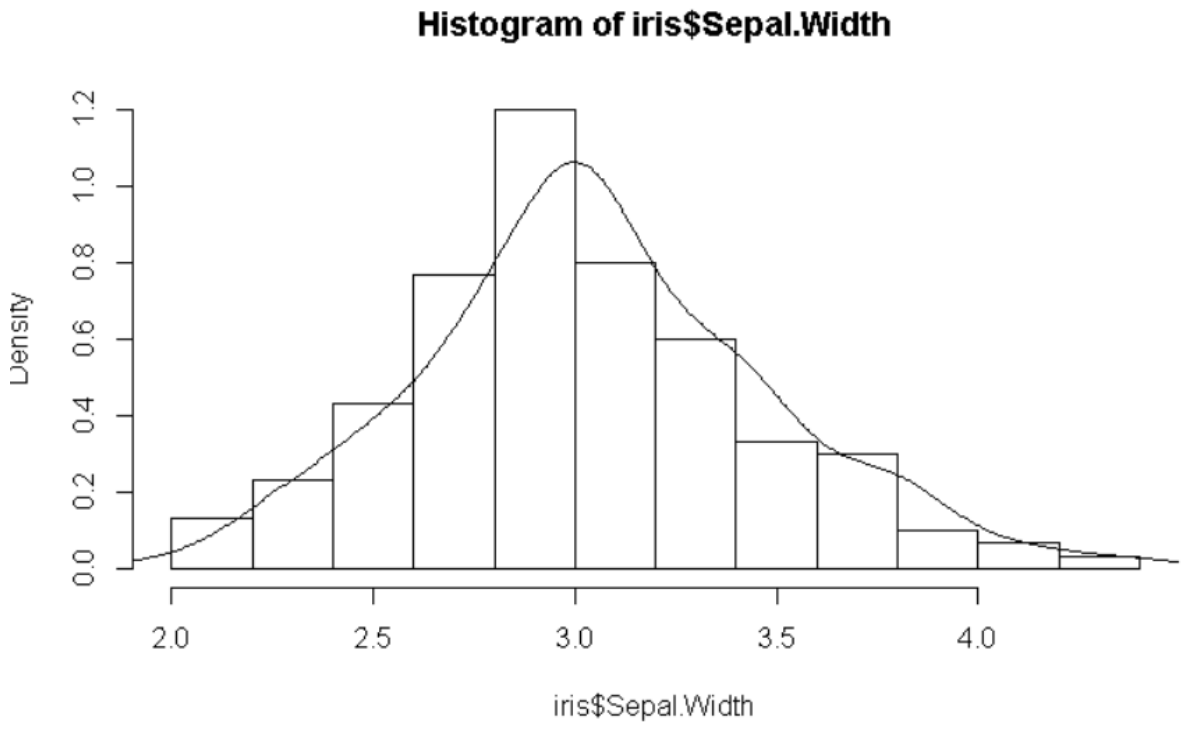
밀도 그림

다음은 iris$Sepal.Width 데이터에 대한 커널 밀도 추정 그림을 그린 예다.

```
plot(density(iris$Sepal.Width))
```

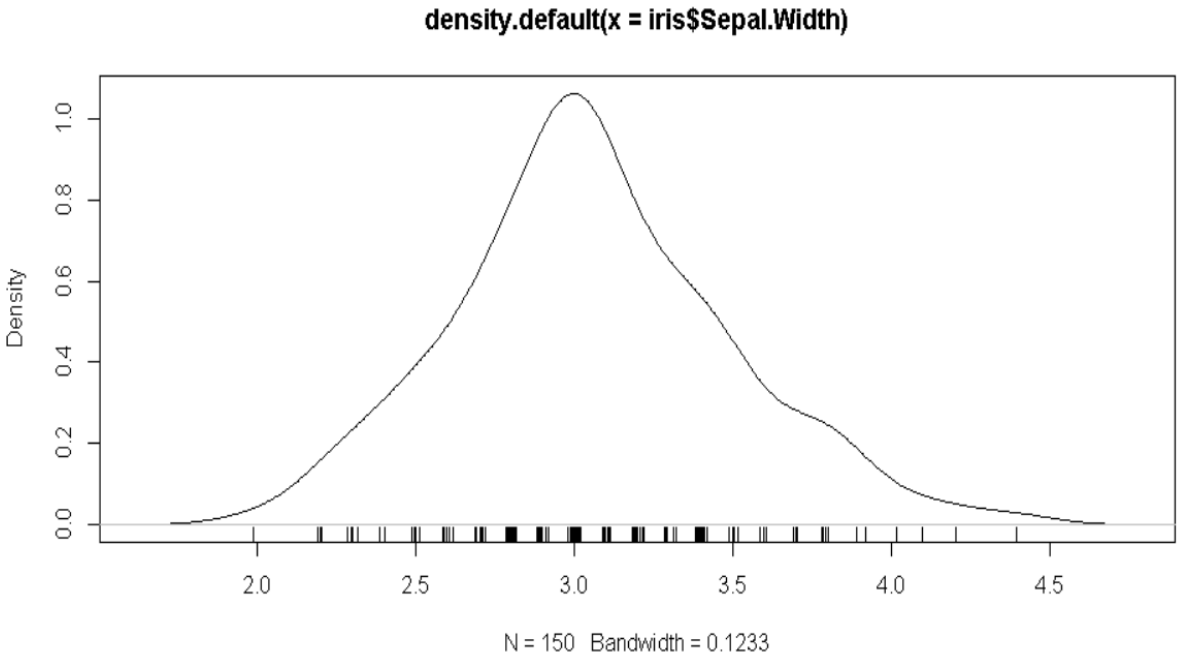


밀도그림은 히스토그램과 같이 그릴 수 있다. plot() 대신 lines()를 사용해 히스토그램 위에 선으로 밀도 그림을 그리면 된다.



```
hist(iris$Sepal.Width, freq =  F)
lines(density(iris$Sepal.Width))
```

밀도 그림에 rug() 함수를 사용하면 실제 데이터의 위치를 X축 위에 표시할 수 있다. 이때, 데이터가 중첩되는 경우가 많다면 jitter()를 같이 사용해 값이 구분되도록 한다.



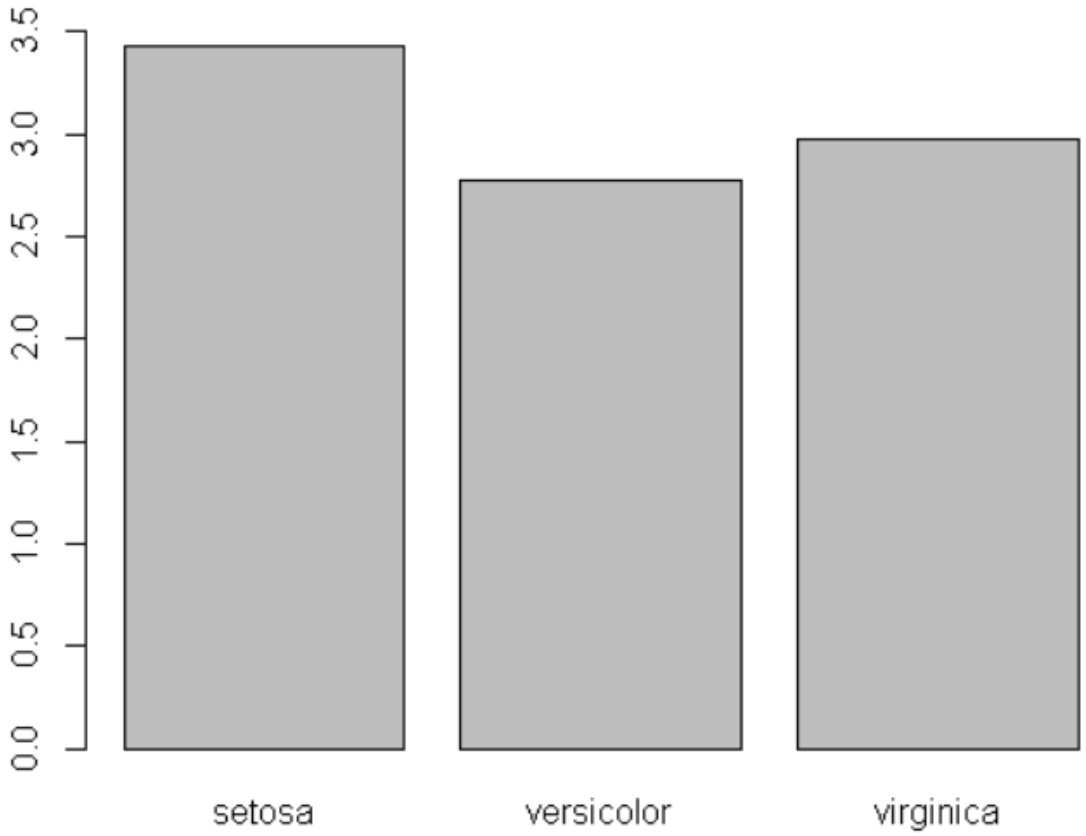

```
plot(density(iris$Sepal.Width))
rug(jitter(iris$Sepal.Width))
```

6.8.4 막대 그래프(barplot)

막대 그래프는 값을 막대로 표시한 그림으로, barplot() 함수를 사용해 그린다.

다음은 Sepal.Width의 평균값을 종별로 구하고 그 값을 막대 그래프로 나타낸 예다.

```
barplot(tapply(iris$Sepal.Width, iris$Species, mean), ylim = c(0,3.5))
```



6.8.5 파이 그래프 (pie)

파이 그래프는 데이터의 비율을 표현하는 목적으로 흔히 사용된다. 다음은 파이 그래프를 그리기 위해 이 절에서 사용한 함수들을 설명한 표다.

pie : 파이 그래프를 그린다

cut : 숫자들을 구간에 따라 분류해 팩터로 변환한다.

table : 분할표(교차표)를 구한다.

이 절에서는 Sepal.Length의 구간별 비율을 파이 그래프로 그려보기로 한다. 구간으로 데이터를 나누기 위해 cut() 함수를 사용한다. cut()은 구간으로 나눌 숫제 벡터(x)와 구간 (breaks)을 인자로 받는다. cut()에서 구간들은 (start, end] 형태로 정의된다.

1:10을 (0,5], (5,10]으로 나누는 예를 생각해보자. 이 경우 breaks에 c(1,5,10) 을 지정하면 안된다. 그러면 1이 어떤 구간에도 속하지 않기 때문이다.

```
> cut(1:10, breaks = c(0,5,10))
 [1] (0,5]  (0,5]  (0,5]  (0,5]  (0,5]  (5,10]
 [7] (5,10] (5,10] (5,10] (5,10]
Levels: (0,5] (5,10]
```

코드 실행 결과 데이터가 2개 라벨 (0,5] (5,10] 중 어느 곳에 속하는지를 나타내는 팩터가 반환되었다.

다음은 breaks에 원하는 구간의 개수를 지정해 1부터 10까지의 수를 3개 구간으로 나누는 예다.

```
> cut(1:10, breaks = 3)
 [1] (0.991,4] (0.991,4] (0.991,4] (0.991,4]
 [5] (4,7]     (4,7]     (4,7]     (7,10]
 [9] (7,10]    (7,10]
Levels: (0.991,4] (4,7] (7,10]
```

이러한 배경 지식을 통해 Sepal.Width 를 5개 구간으로 나눠보자.

```
> head(cut(iris$Sepal.Width, breaks = 5))
[1] (3.44,3.92] (2.96,3.44] (2.96,3.44]
[4] (2.96,3.44] (3.44,3.92] (3.44,3.92]
5 Levels: (2,2.48] (2.48,2.96] ... (3.92,4.4]
```

파이 그래프를 그리려면 이 팩터 데이터를 그대로 사용할 수는 없으며, 나뉜 각 구간에 몇 개의 데이터가 있는지 세야 한다. table() 함수는 이러한 목적으로 사용되며, 팩터 값을 받아 분할표(Contingency TableCross Tabulation/Cross Tab)을 만든다. 이 함수의 이용법을 이해하기 위해 여러 문자가 저장된 벡터가 있을 때 각 문자열의 개수를 세는 다음 예를 보자.

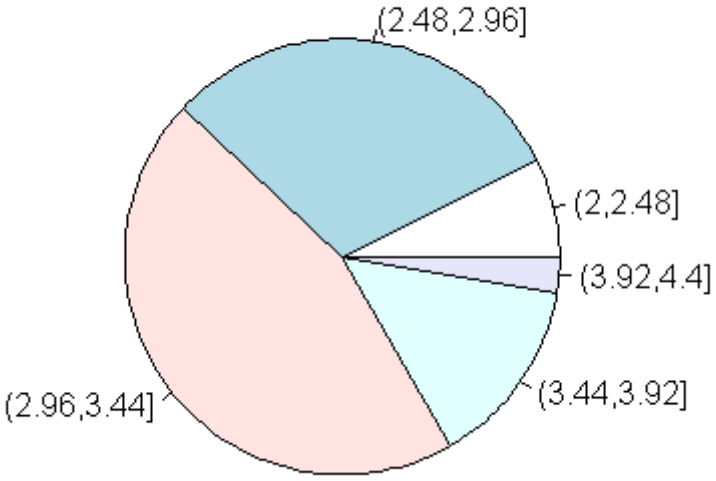
```
> rep(c("a","b","c"),1:3)
[1] "a" "b" "b" "c" "c" "c"

> table(rep(c("a","b","c"),1:3))
a b c
1 2 3
```

table()의 결과로 각 문자의 개수가 구해졌음을 볼 수 있다. 이를 응용해 Sepal.Width 를 10개 구간으로 나눈 뒤 각 구간에 몇 개의 데이터가 있는지 구해보자.

```
> table(cut(iris$Sepal.Width, breaks = 5))

   (2,2.48] (2.48,2.96] (2.96,3.44] (3.44,3.92]
         11          46          68          21
 (3.92,4.4]
          4
```

차후에 table() 함수에 대해 더 자세히 다루기로 하고, 여기서는 지금까지 구한 값을 사용해 파이 그래프를 그려보자.

```
pie(table(cut(iris$Sepal.Width, breaks = 5)))
```



6.8.6 모자이크 플롯(mosaicplot)

모자이크 플롯은 범주형(R에서는 팩터로 표현) 다변량 데이터 (하나 이상의 변수가 있는 데이터)를 표현하는 데 적절한 그래프다. 모자이크 플롯에는 사각형들이 그래프에 나열되며, 각 사각형의 넓이가 각 범주에 속한 데이터의 수에 해당한다.

이 절에서는 타이타닉호 생존자의 정보를 담고 있는 Titanic 데이터를 사용해 모자이크 플롯을 그려본다.

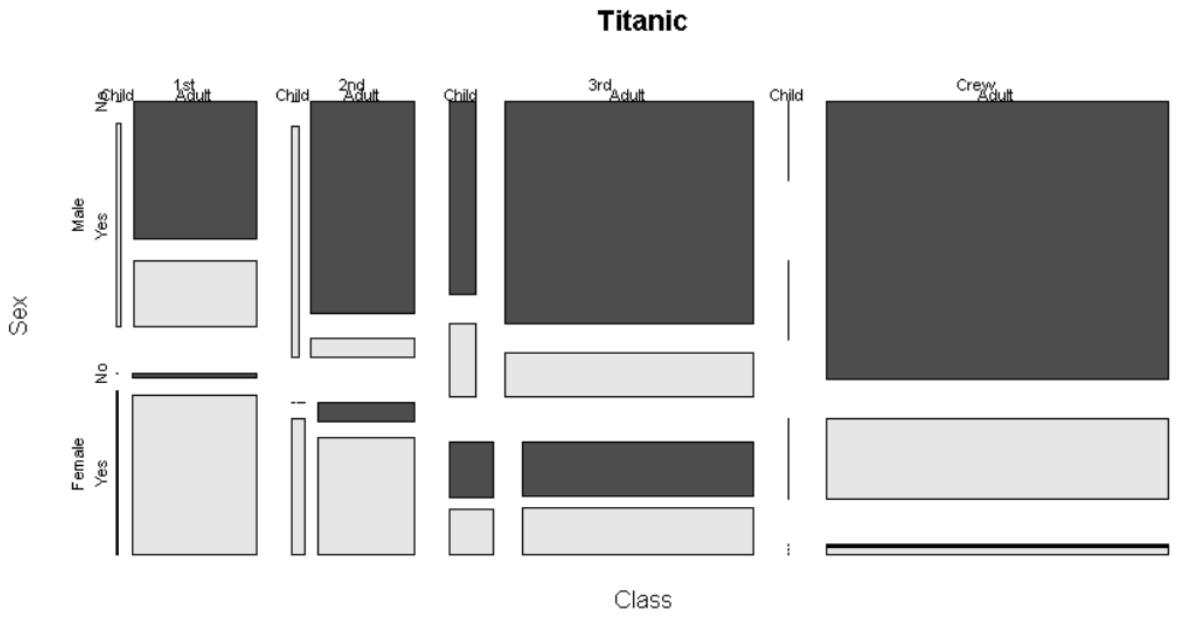
```
> data(Titanic)
> str(Titanic)
 'table' num [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
 - attr(*, "dimnames")=List of 4
  ..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
  ..$ Sex     : chr [1:2] "Male" "Female"
  ..$ Age     : chr [1:2] "Child" "Adult"
  ..$ Survived: chr [1:2] "No" "Yes"
```

Titanic은 table 클래스의 인스턴스이며, 속성은 객실 구분, 성별, 성인 여부, 생존 여부로 구성되어 있다.

실제 데이터를 살펴보자.

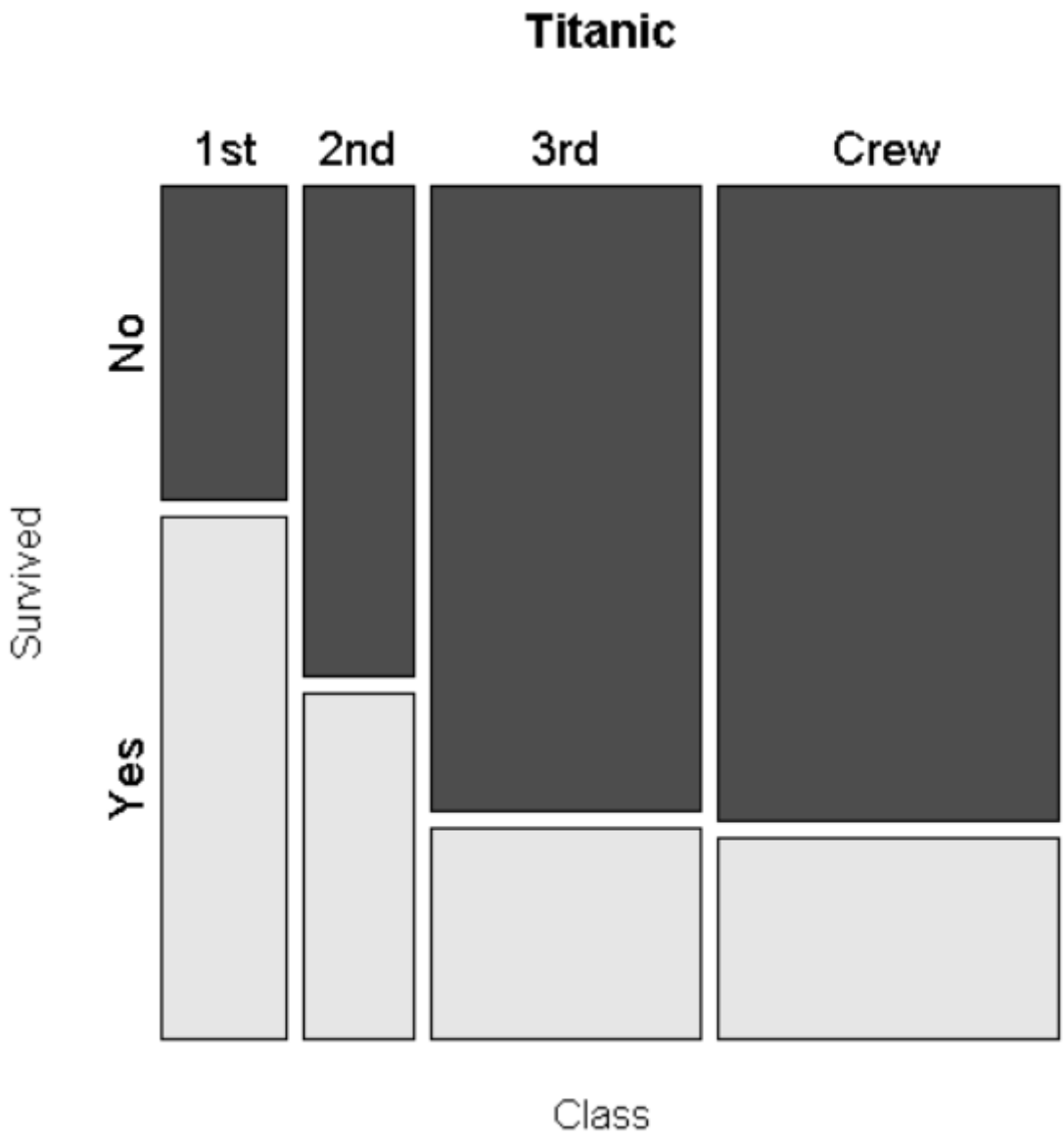
```
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No
```

Titanic으로부터 가장 간단한 모자이크 플롯을 그리는 방법은 이 표를 그대로 mosaicplot()에 넘기는 것이다.




```
mosaicplot(Titanic, color = T)
```

위 그림처럼 모든 조건을 나열해 그림을 그리면 그림이 복잡해 져서 개별 속성에 대한 분포를 살펴보기 힘들다. 이 경우 일부 속성에 대해서만 모자이크 플롯을 그리기 위해 mosicplot(formula, data) 형태를 사용할 수 있다. 예를 들어, 다음 코드는 객실과 생존 여부에 대한 모자이크 플롯을 그린다.



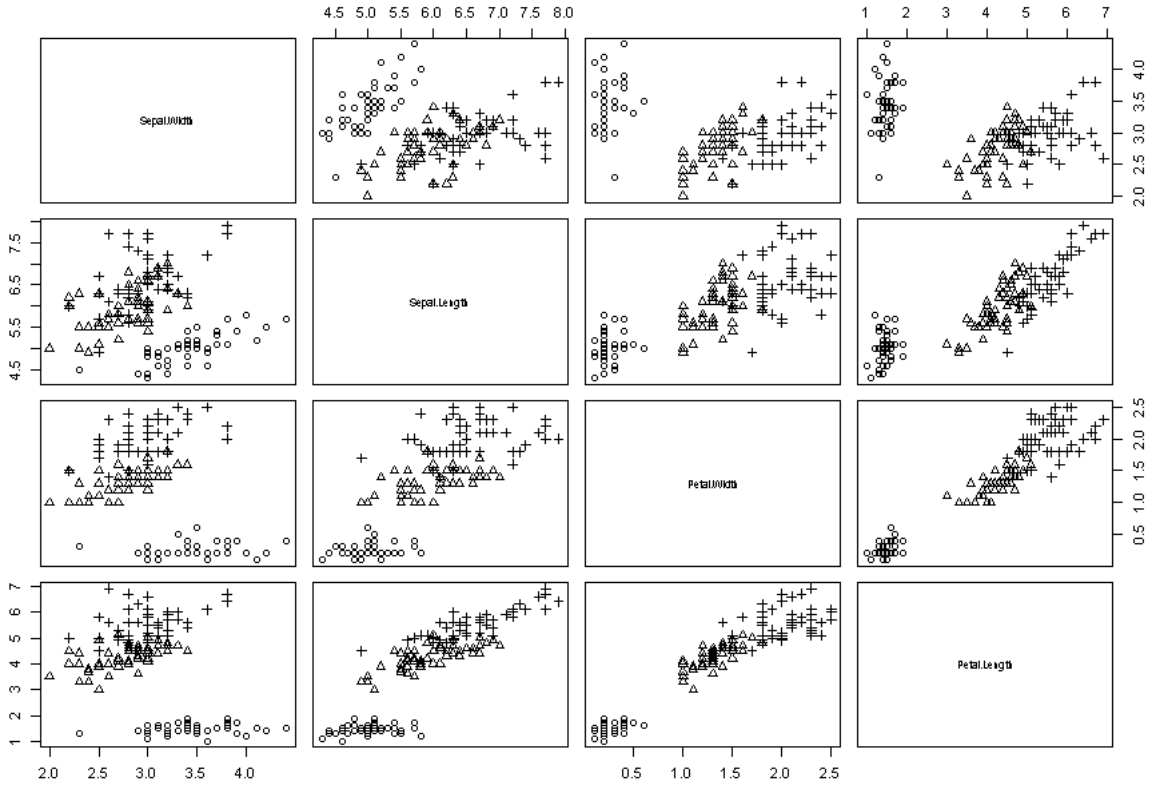

```
mosaicplot(~Class+Survived,data = Titanic, color = T, cex=1.2)
```

위의 그림을 보면 높은 등급 객실에서의 생존율이 낮은 등급 객실에서의 생존율보다 높았음을 쉽게 알 수 있다.

6.8.7 산점도 행렬(pairs)

산점도 행렬은 다변량 데이터에서 변수 쌍 간의 산점도들을 그린 그래프를 말한다. 산점도 행렬을 사용하면 여러 변수가 있을 때 모든 변수 간 산점도를 손쉽게 그릴 수 있고, 이를 들여다보면 변수들 간 상관관계 등의 특징을 쉽게 찾을 수 있다.

다음은 아이리스 데이터에서 각 종별에 대한 데이터의 산점도 행렬을 그리는 예다. pch는 각 조합별로 서로 다른 점 모양을 사용하게 하려고 지정했다.




```
pairs(~Sepal.Width+Sepal.Length+Petal.Width+Petal.Length,
      data = iris, pch=c(1,2,3)[iris$Species])
```

예제를 실행하면 setosa는 사각형, versicolor는 원형, virginica는 십자형 기호로 표시된다. 각 Species가 어떤 기호를 사용하게 되는지는 levels()로 범주의 목록을 살펴보면 알 수 있는데, 아래 코드에서 볼 수 있듯이 setosa가 1, versicolor가 2, virginica가 3에 해당해 각각 pch 1,2,3을 사용한다. pch 1,2,3이 각각 사각형, 원형, 십자형임은 example(points) 명령을 실행해보면 쉽게 알 수 있다.

```
> levels(iris$Species)
[1] "setosa"     "versicolor"
[3] "virginica"
> as.numeric(iris$Species)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [18] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
 [52] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [69] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [86] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3
[103] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[120] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[137] 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

6.8.8 투시도(persp), 등고선 그래프(contour)

투시도는 3차원 데이터를 마치 투시한 것처럼 그린 그래프며, 등고선 그래프는 높이가 같은 곳을 선으로 연결한 그래프다. 좌표로 말하자만 이들은 x,y,z 값을 그려주는 그래프로, $z = f(x, y)$ 처럼, z값이 x,y에 따라 정해지는 값일 때, (x,y)에 따라 z가 어떻게 변하는지를 한눈에 보여준다.

persp : 투시도를 그린다.

contour : 등고선 그래프를 그린다.

outer : 배열의 외적 (outer product)를 구한다.

persp(), contour()에서 유용한 함수가 outer()이다. 이 함수를 사용하면 $z = f(x, y)$의 함수 관계가 있을 때, z를 손쉽게 계산할 수 있다. 예를 들어, x= 1:5, y = 1:3 일 때, 모든 x,y 조합에 대해 x+y를 계산해보자.

```
> outer(X = 1:5, Y = 1:3 , "+")
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    3    4    5
[3,]    4    5    6
[4,]    5    6    7
[5,]    6    7    8
```

위 예에서 outer()의 결과는 행렬이며, 이 행렬의 (m,n)의 결과는 행렬이며, 이 행렬의 (m,n)에는 m+n 값이 저장되어 있음을 볼 수 있다.

fun에는 R 함수를 기술해도 된다.

```
> outer(X = 1:5, Y = 1:3 ,
+       function(x,y){x+y})
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    3    4    5
[3,]    4    5    6
[4,]    5    6    7
[5,]    6    7    8
```

outer()를 사용해 X축 그리드 seq(-3,3,1), Y축 그리드 seq(-3,3,1)에 대해 이변량 정규분포를 그려보자. persp()에 넘길 Z인자는 각 X,Y 그리드 점에 대해 확률 밀도인데, 다변량 정규분포의 확률 밀도는 mtvnorm 패키지의 dmvnorm()을 사용해 계산한다. 다음은 x=0, y=0에 대해 x,y의 평균이 각각 0이고, 공분산 행렬이 2x2 크기의 단위행렬일 때 밀도를 구하는 예다.

```
install.packages('mvtnorm')
library(mvtnorm)
> dmvnorm(c(0,0), rep(0,2), diag(2))
[1] 0.1591549
```

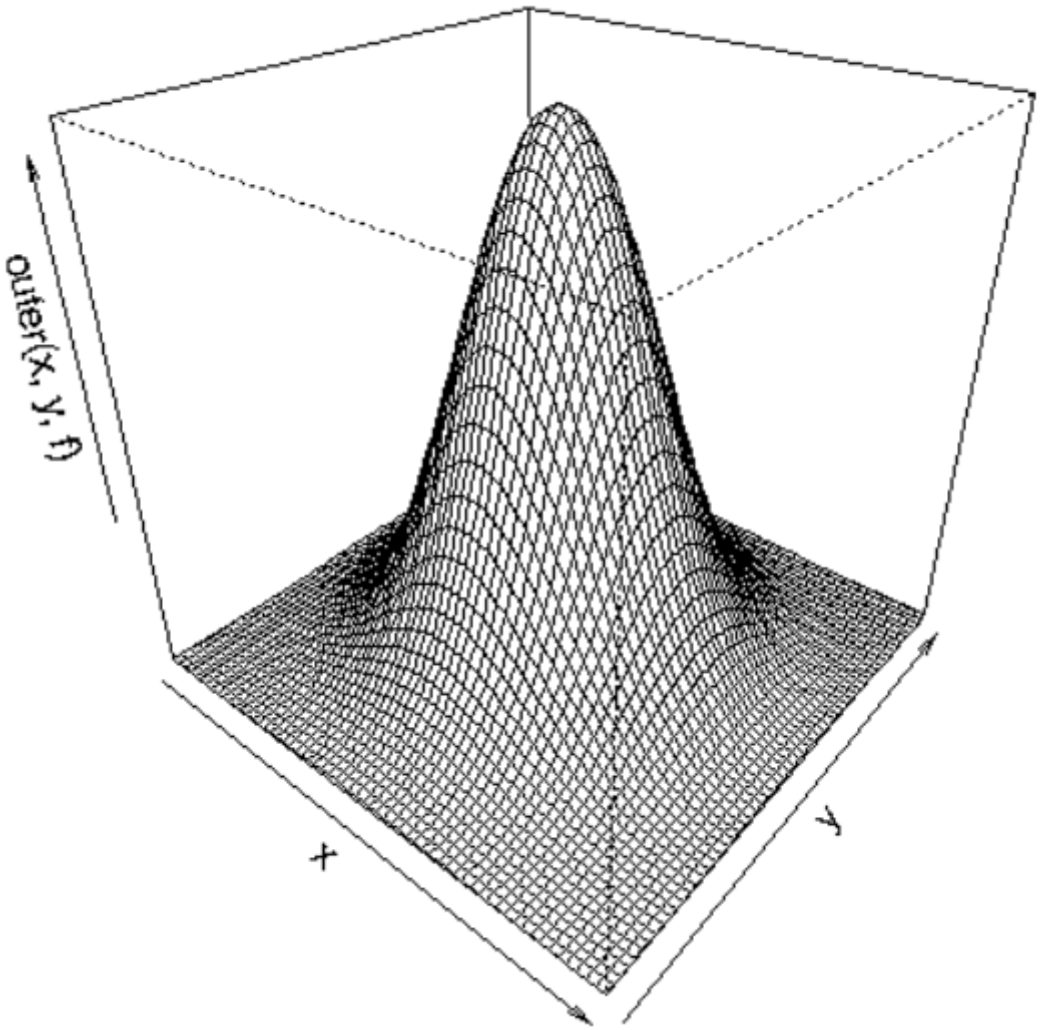
다음 코드는 seq(-3,3,1) 의 X,Y 축 그리드 조합에 대해 Z축 값을 구한다. dmvnorm()에서 평균의 기본값은 영행렬이고, 공분산 행렬의 기본값은 단위행렬이다.

```
x <-seq(-3,3,.1)
y<- x
outer(x,y,function(x,y) {dmvnorm(cbind(x,y))})
```

이제 모든 준비가 끝났다. 지금까지 살펴본 내용을 종합해 투시도를 그리는 코드는 다음과 같다.

```
x <-seq(-3,3,.1)
y<- x
f <-function(x,y) {dmvnorm(cbind(x,y))}
persp(x,y, outer(x,y,f), theta = 30,phi=30)
```

코드에서 theta와 phi는 그림의 기울어진 각도를 지정하는 인자다. 이 두 인자에 지정할 값은 여러 값을 넣어 그림을 그려보면서 적절하게 정하면 된다 결과는 다음과 같다.



등고선 그래프는 투시도와 유사하지만 투시한 3차원 그림 대신 값이 같은 곳들을 선으로 연결한 등고선을 이용해 데이터를 표시한다. 등고선 그래프는 contour()를 사용해 그린다. 다음은 이변량 정규 분포에 대한 등고선 그래프를 그리는 예다.