R-Graphics tool

Jong-June Jeon

August, 2017

R Basic (Review)

유용한 필터링 함수

- subset 함수는 NA를 포함한 벡터에서 특정 원소를 필터링하는데 유용한 함수다.

```
x <- c(6, 1:3, NA, 12)
x
```

```
## [1]  6  1  2  3 NA 12
```

```
x[x>5]
```

```
## [1]  6 NA 12
```

```
subset(x, x>5)
```

```
## [1]  6 12
```

유용한 필터링 함수

- NA 값을 판별하는 함수

```
is.na(x)
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE FALSE
```

subset을 이용하지 않으면 is.na 함수를 사용해 NA 값을 지우고 작업을 진행할 수도 있을 것이다.

유용한 필터링 함수

- which 함수: 벡터내의 어떤 값이 조건에 맞는지 확인하고 그 위치를 알고 싶을 때 사용하는 함수 입력값으로 논리값 벡터를 받고, TRUE 값을 가진 원소의 위치를 반환한다.

```
x
```

```
## [1]  6  1  2  3 NA 12
```

```
which(x>5)
```

```
## [1] 1 6
```

벡터의 원소값이 5보다 큰 위치를 반환해준다.

유용한 필터링 함수

```
x <- c(3,1,4,1)
which(x>5)
```

```
## integer(0)
```

```
y <- which(x>5)
length(y)
```

```
## [1] 0
```

위 처럼 조건에 맞는 벡터내의 원소가 없는 경우 integer(0)를 반환한다. 이를 저장한 경우 그 벡터의 길이는 0이다. 이를 이용해서 조건을 만족시키는 원소가 하나라도 있는지 확인할 수 있다. 이는 다음 any 함수를 이용해서 확인 할 수도 있다.

```
x <- c(3,1,4,1)
any(x>5)
```

유용한 필터링 함수

- %in%함수

```
x <- c(3,1,4,1)
x%in% c(2,1,4)
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

x 의 각 원소가 어떤 벡터에 포함이 되어 있나?

유용한 필터링 함수

- match 함수

```
x <- c(3,1,4,1)
match(x ,c(2,1,4))
```

```
## [1] NA  2  3  2
```

x 의 각 원소가 벡터에 포함된 위치가 어디인가?

데이터 프레임

데이터 프레임은 데이터 테이블을 저장하는 R의 대표적인 데이터 형식이다.
data.frame 함수를 이용하여 생성한다.

```
kids <- c("Jack", "Jill")
ages <- c(12, 10)
d <- data.frame(kids, ages, stringsAsFactors = FALSE)
d
str(d)
```

- stringsAsFactors 는 data.frame 함수의 option 으로 문자형 변수를 R에서 정의한 팩터라는 변수형식으로 변환 여부를 결정한다.
- F라 하면 문자형 변수를 팩터 형식으로 변환하지 않는다.

데이터 프레임

- 데이터 프레임이 가지고 있는 값에 접근하기
- 데이터 프레임은 특별한 형태의 List 자료형이지만, 행렬 혹은 2차원 배열로 이해해도 된다.

```
names(d)
d$ages
class(d$ages)
```

- `names(d)` 는 데이터 프레임 `d` 가 가지고 있는 열의 이름을 보여준다.
- `d$age` 는 `age`이름을 가진 열을 호출하며 `d[,2]` 혹은 `d['age']`로 사용할 수 있다.
- `str(d)` 를 사용해 데이터 프레임 `d`의 구조를 확인해보자.

파일 읽기

- 절대경로, 상대 경로

```
getwd()
```

```
## [1] "C:/Users/uos_stat/Dropbox/class/2017 Bigdata/course-1"
```

- 상대경로의 지정: `./` (현재 폴더에 있는), `../` (한 단계 상위 폴더에 있는)
- ex) 다음을 해석하시오 `./Rplot.jpg`, `.../block/source/1.png`

```
setwd('D:/rProg')
getwd()
```

- 작업공간의 위치를 특정한 폴더로 옮길 수 있음.
- 각자 작업공간의 위치를 지정하고 실습 파일을 작업공간으로 옮기시오.

파일 읽기

```
A <-read.table("CO2.dat", header = TRUE, sep = " ", stringsAsFactors= F)
head(A)
class(A$Plant)
```

- CO2.dat 파일은 공백기호(space)로 데이터가 구분되어 있음.

```
install.packages('xlsx')
library(xlsx)
A <-read.xlsx("CO2.xlsx", header = TRUE, sheetIndex = 1, stringsAsFactors= F)
head(A)
class(A$Plant)
```

- `install.packages` 명령은 local 로 package 파일을 가져오는 명령으로 한번만 실행하면 됨.
- sheetIndex 는 엑셀 workbook에서 몇 번째 sheet를 가져올지 정해주는 부분임. sheetName 옵션을 사용해도 됨.

rbind와 cbind 함수 사용하기

- 데이터 프레임 역시 행렬과 마찬가지로 rbind와 cbind를 사용할 수 있음.

```
A = data.frame(x1 = rep(0,10), x2 = rep('b',10))
B = data.frame(x3 = rep(1,10), x2 = rep('d',10))
AB = cbind(A,B)
head(AB)
```

- rbind(A,B) 작동하지 않는 것을 확인하시오. 왜 작동하지 않을까?

Plotting points

plot

- 데이터의 준비:
    - mtcars 데이터에는 차종, 연비(mpg: mile per gallon), 배기량(disp: displacement) 등의 정보가 포함되어 있다.

```
attach(mtcars)
head(mtcars, n = 2)
```

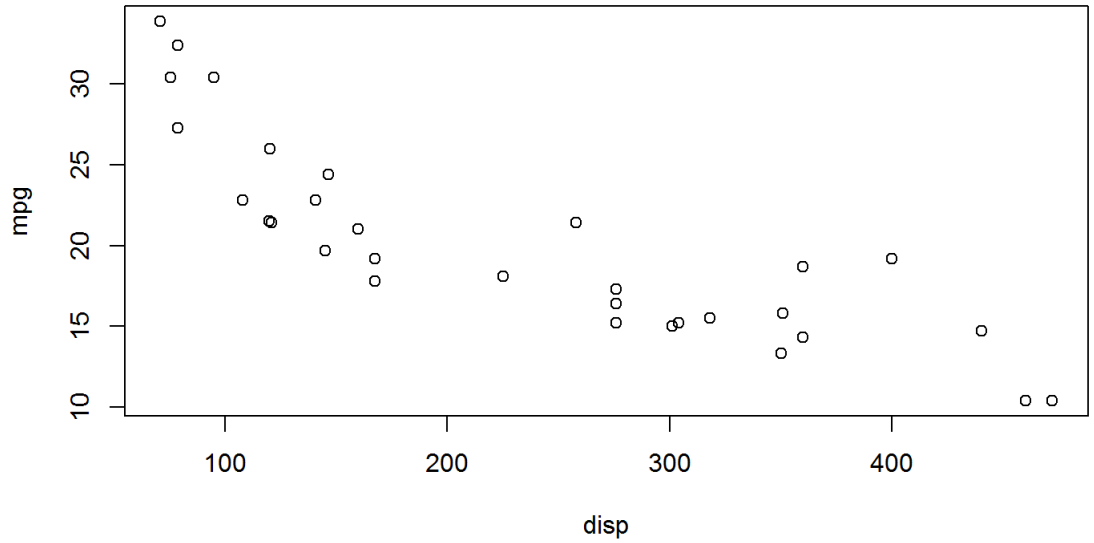
```
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

- 변수에 대한 설명은 `?mtcars` 를 R에 입력하여 확인할 수 있다.

plot

- 산점도를 통해 두 변수간의 관계를 알 수 있다

```
plot(mpg ~ disp, data = mtcars)
```



plot

```
plot(mpg ~ disp, data = mtcars)
```

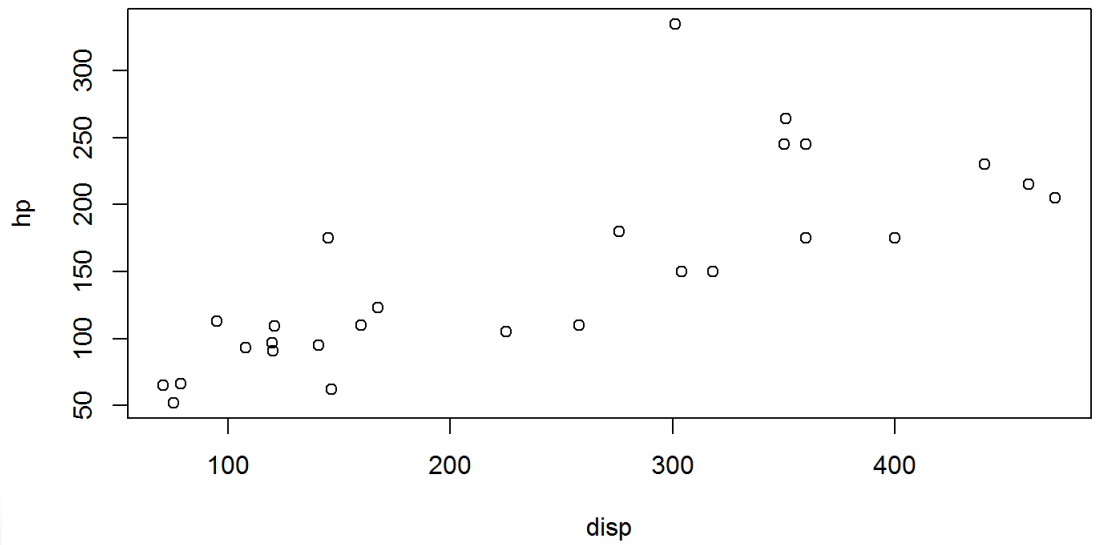
- 데이터는 데이터 프레임 형식이다.
- `mpg~disp` 에서 `mpg`와 `disp`는 mtcars 데이터 프레임의 열 이름이다.
- `mpg~disp` 는 y 축과 x축을 `mpg`와 `disp`변수 값으로 하는 산점도를 그리겠다는 뜻이다.

plot

- y 축을 `hp` (horse power) x 축을 `disp`로 하는 산점도를 그려보자.

plot

```
plot(hp ~ disp, data = mtcars)
```



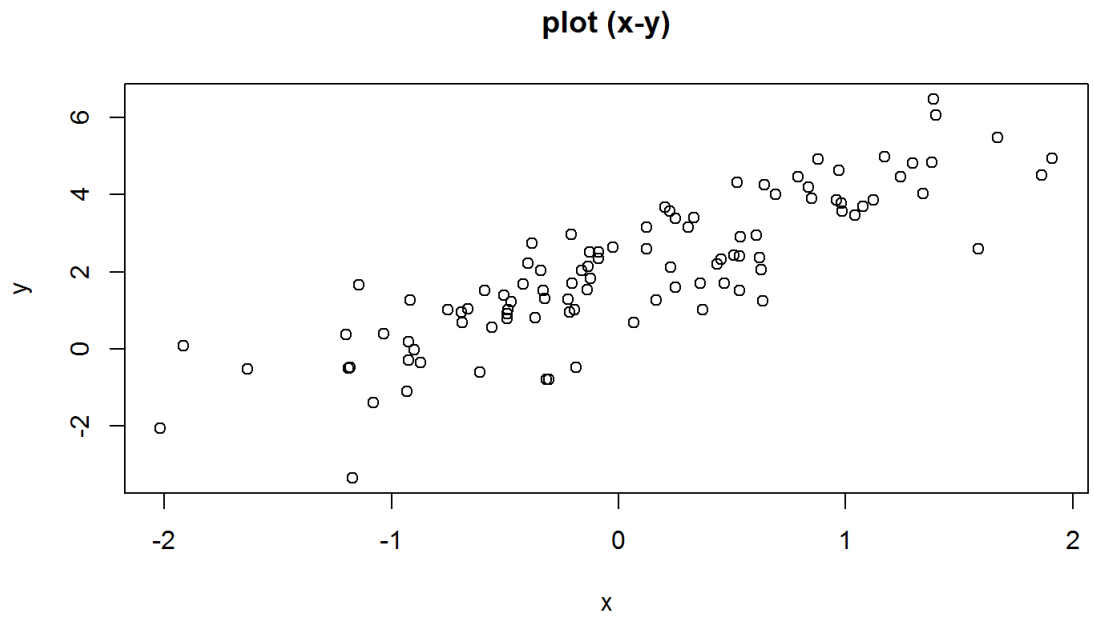
plot

- plot은 데이터의 값을 이용하여 좌표평면에 직접 산점도를 그릴 수 있다.

```
x = rnorm(100)
y = 2+2*x + rnorm(100)
plot(x,y, main = "plot (x-y)")
```

- `rnorm(100)`은 표준정규분포를 따른 랜덤샘플을 100개 생성한다.
- y는 (단,)) 으로 생성된다.
- `main = "plot (x-y)"` 는 산점도의 제목을 넣어준다.

plot



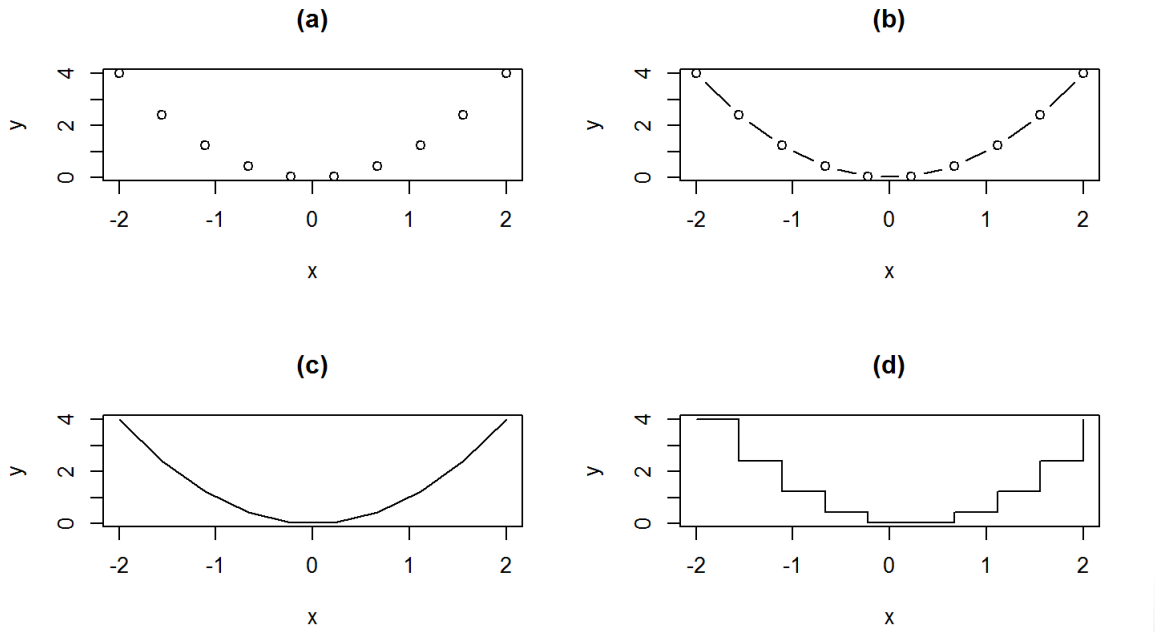
plot

- plot 함수는 연속되는 점과 점사이에 특정한 그래픽 작업을 할 수 있다.
- type option:
  - `type = 'p'`: point
  - `type = 'l'`: line
  - `type = 'b'` : both point and line
  - `type = 's'` : step
- type에 대한 정보는 `?plot`에서 얻을 수 있다.

```
x = seq(-2,2, length = 10)
y = x^2
plot(x,y, type = 'p', main = "y = x^2")
```

plot

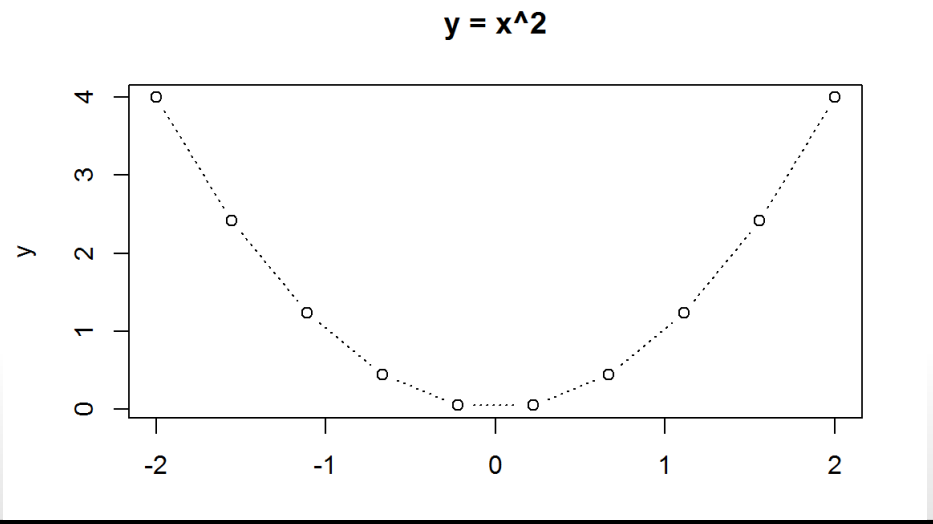
* 아래 그래프중 `type = 'b'`인것은 무엇인가?



plot

- plot 함수는 좌표평면상에 그려지는 선의 모양을 바꿀 수 있다.
  - lty (line type)를 사용한다
  - lty 에 대한 정보는 ?par에서 lty 항목에서 얻을 수 있다.

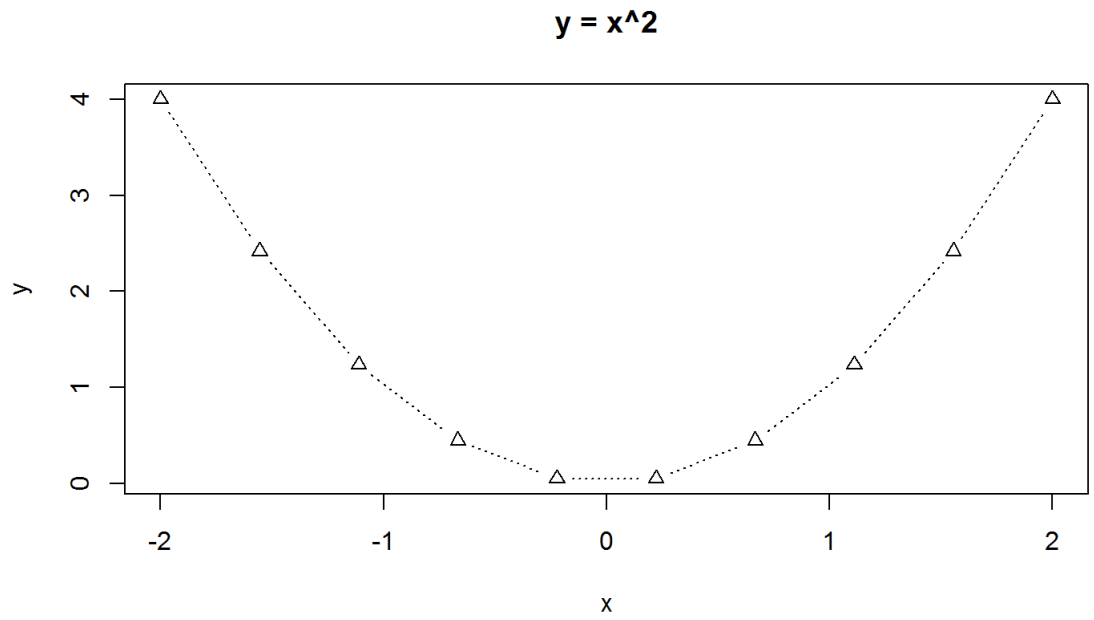
```
plot(x,y, type = "b", lty = 3, main = "y = x^2")
```



plot

- pch (plotting symbol) 를 이용하여 점의 모양을 바꿀수 있다.

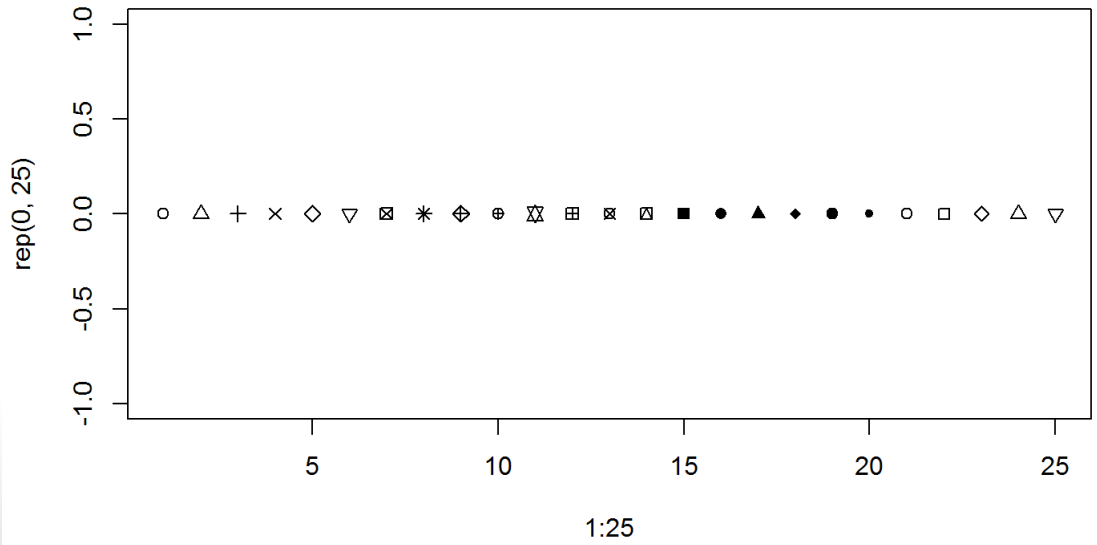
```
plot(x,y, type = "b", lty = 3, pch = 2, main = "y = x^2")
```



plot

- 다음 코드의 실행결과를 설명하시오

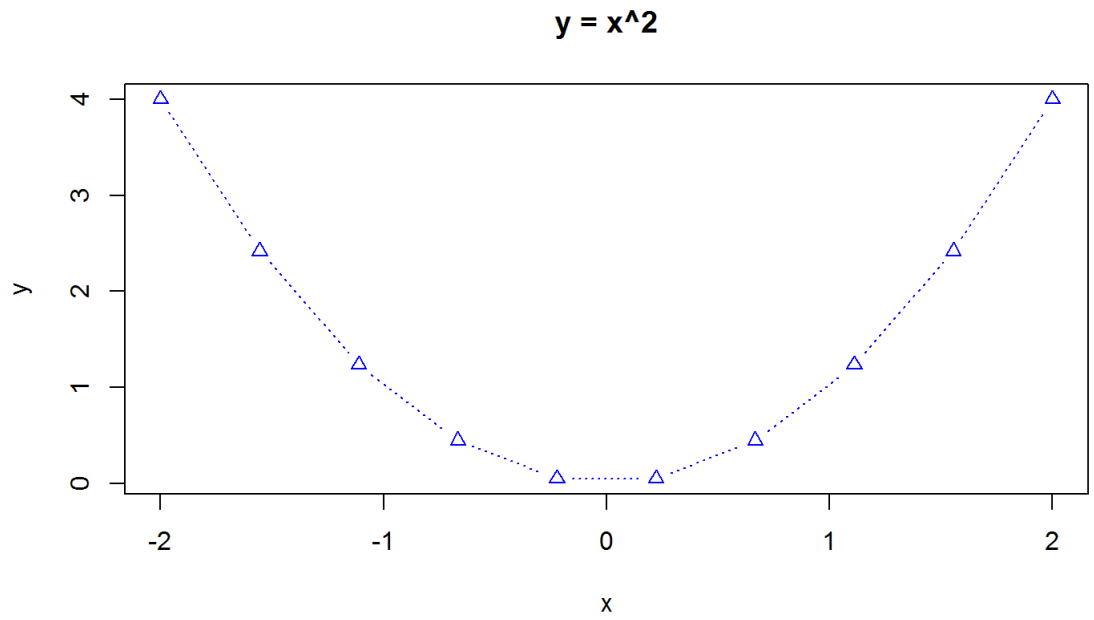
```
plot(x = 1:25,y = rep(0, 25), pch = 1:25)
```



plot

- col (color)를 이용하여 그래프의 색상을 변경할 수 있다.

```
plot(x,y, type = "b", lty = 3, pch = 2, col = "blue", main = "y = x^2")
```



plot

- color에 대한 정보는 `colors()` 를 이용해서 확인할 수 있다.
  - color 는 `?palette`에서 사용가능한 색상을 설정할 수 있다.

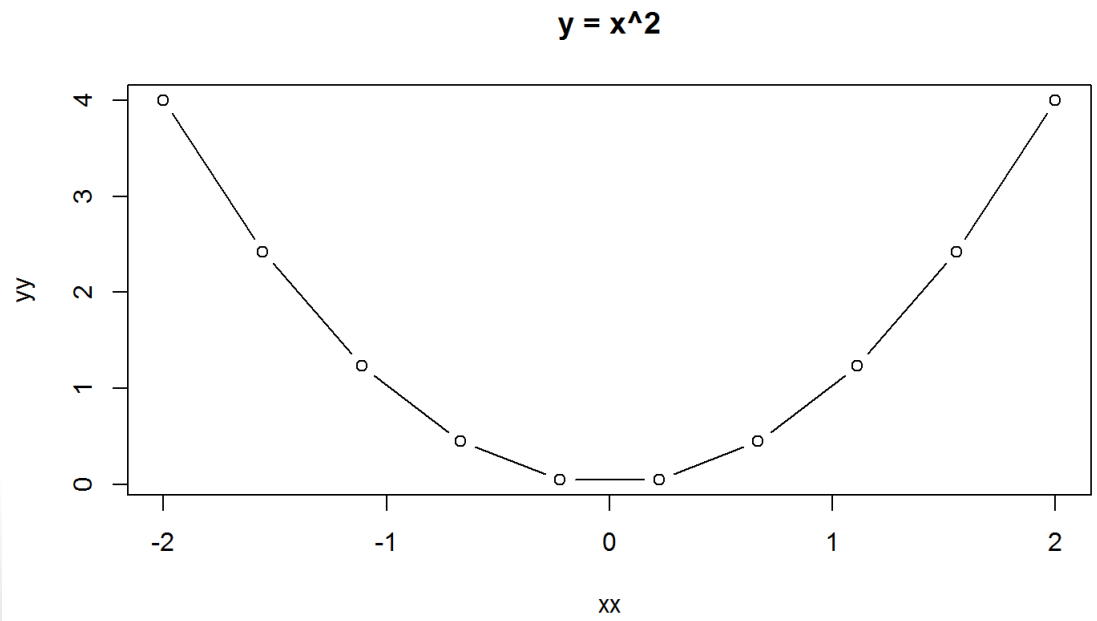
```
colors()[1:10]
```

```
##  [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
##  [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
##  [9] "aquamarine1"   "aquamarine2"
```

plot

- 좌표축에 label 붙이기: xlab, ylab (label) 를 이용한다.

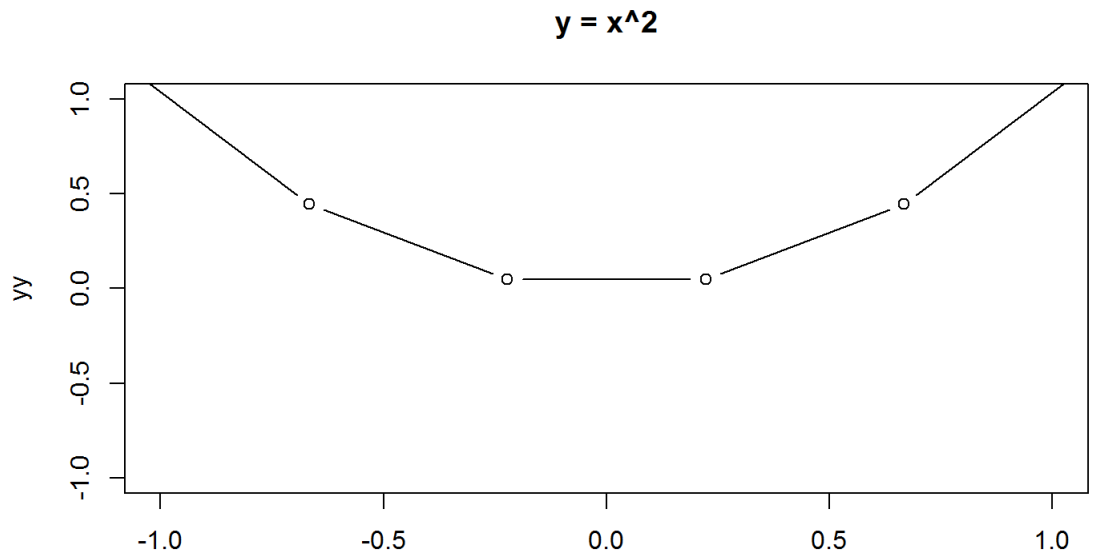
```
plot(x,y, type = "b", xlab = "xx", ylab = "yy", main = "y = x^2")
```



plot

- 좌표축의 범위 설정하기: xlim, ylim 을 이용한다.

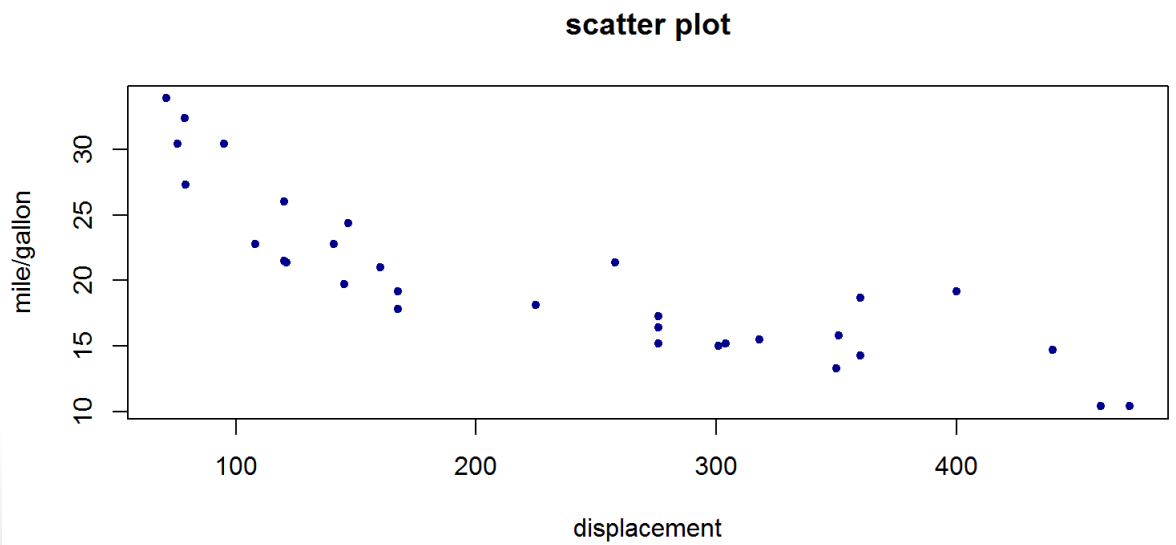
```
plot(x,y, type = "b", xlab = "xx", ylab = "yy", main = "y = x^2",
     xlim = c(-1,1), ylim = c(-1,1))
```



plot

- plot 연습

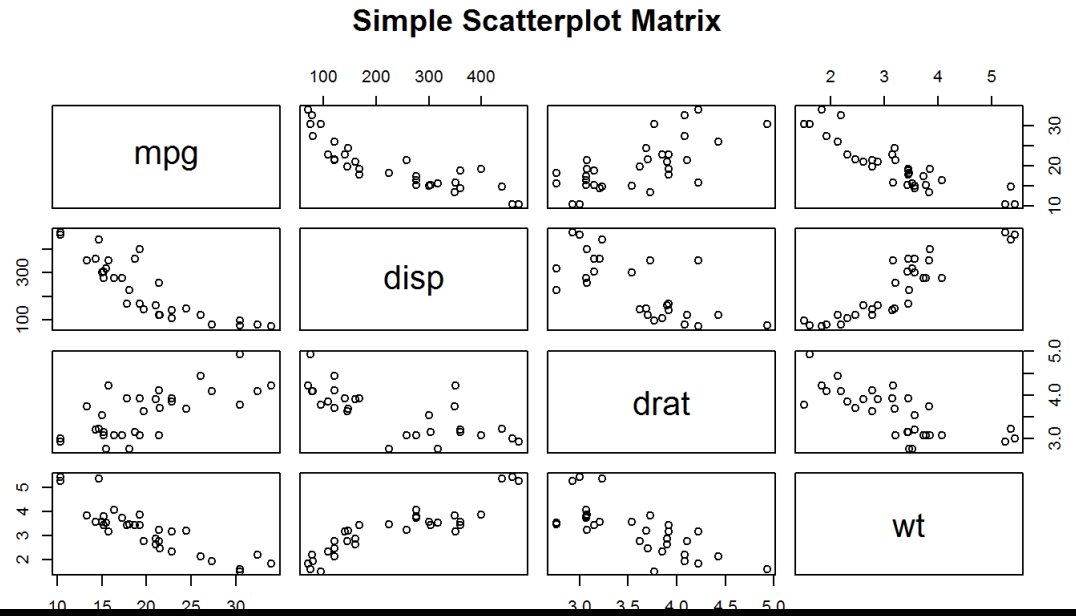
```
plot(mpg~disp, data = cars, xlab = "displacement", ylab = "mile/gallon",
     main = "scatter plot", pch = 20, col = 'darkblue')
```



plot

- 여러개의 산점도를 표현하기 위해서는 pairs()를 사용한다.

```
pairs(~mpg+disp+drat+wt,data=mtcars,
   main="Simple Scatterplot Matrix")
```



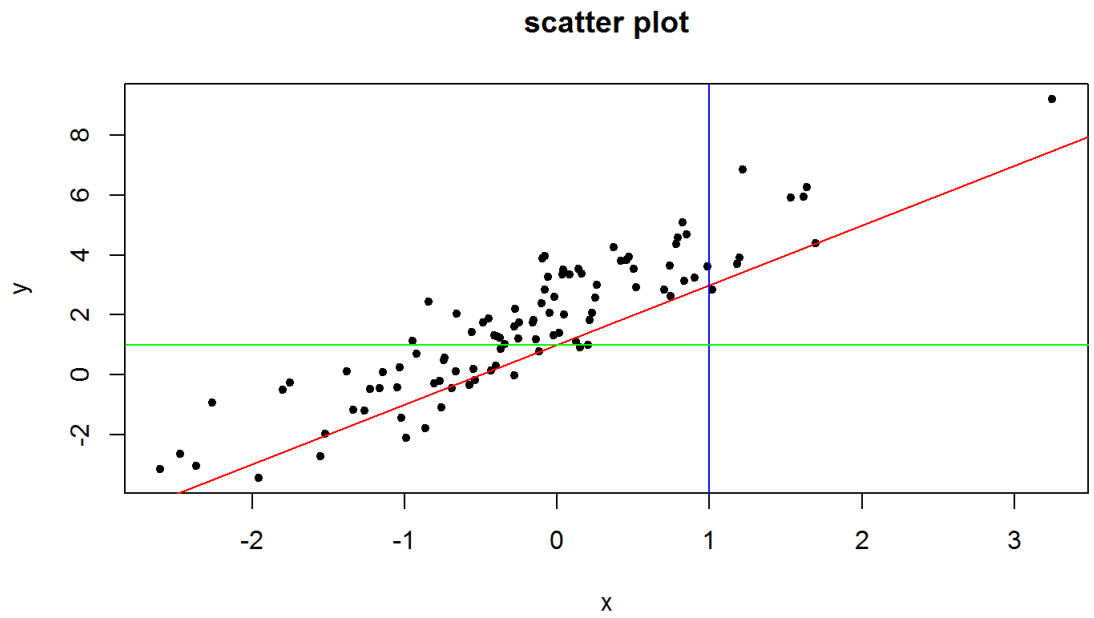
Add lines and points

abline

- abline 은 만들어진 플롯 위에 직선을 그려주는 함수다.
  - 절편(a)과 기울기(b)를 가진 직선
  - 수평선(h)
  - 수직선(v)

```
x = rnorm(100)
y = 2+2*x + rnorm(100)
plot(x,y, pch = 20, main = 'scatter plot')
abline(a = 1, b = 2, col = "red")
abline(v = 1, col = "blue")
abline(h = 1, col = "green")
```

abline

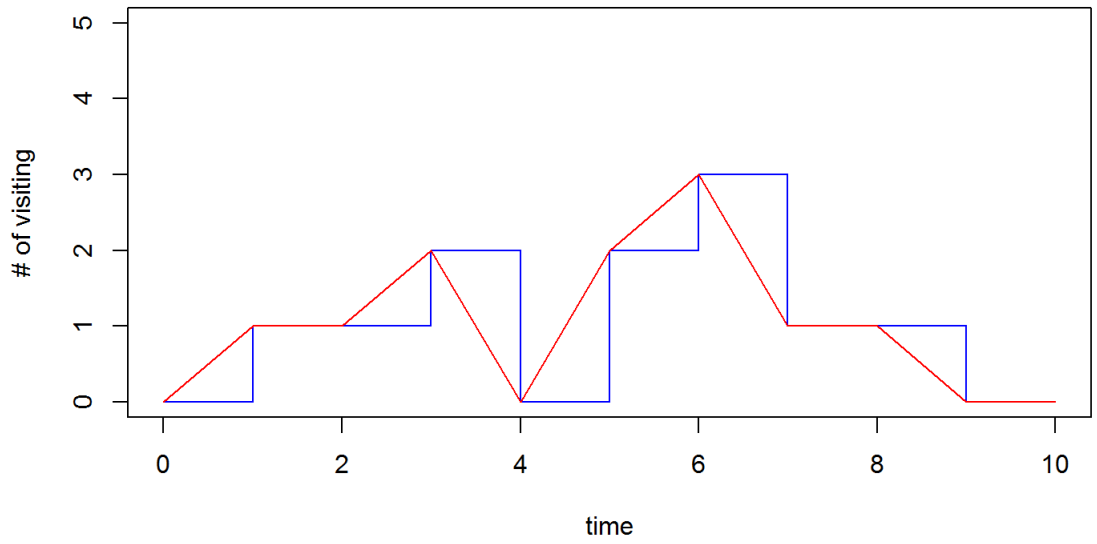


points

- points 은 만들어진 플롯 위에 점을 찍어주는 함수다.
  - type을 이용해서 점과 점사이를 잇는 방법을 정해줄 수 있다.
  - lty 를 이용해서 선의 형태를 정해줄 수 있다.

```
plot(x = 1,y = 1, type = 'n', xlim = c(0,10), ylim = c(0,5),
     xlab = 'time', ylab = '# of visiting')
x = 0:10
set.seed(1)
y = rpois(length(x), lambda = 1)
points(x, y, col = "blue", type = "s")
points(x, y, col = "red", type = "l")
```

points

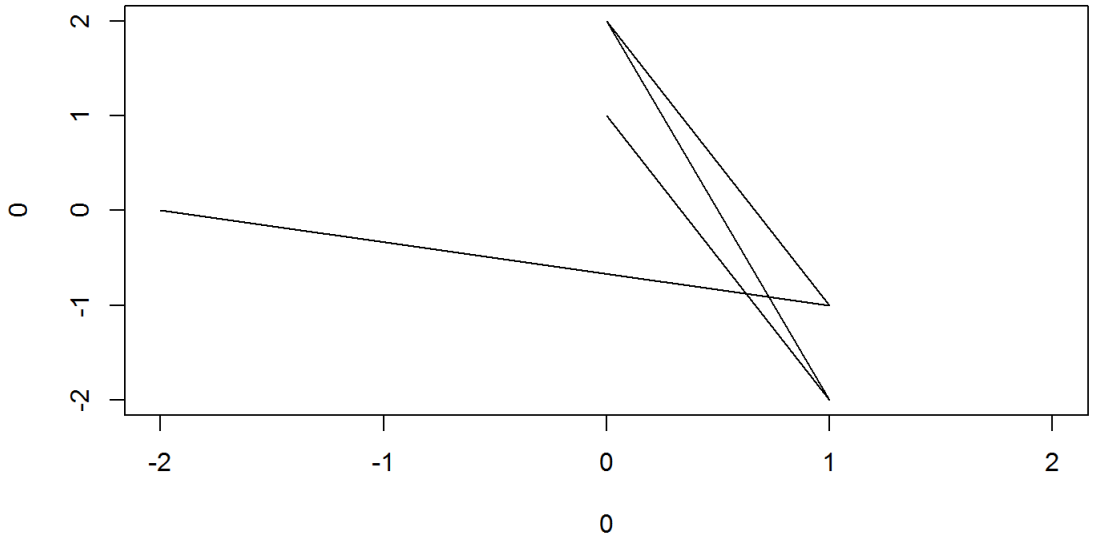


lines

- lines() 는 만들어진 그래프 위에 선을 그린다.
- 연속된 점들을 이어준다

```
plot(0,0, type = 'n', xlim = c(-2,2), ylim = c(-2,2))
x = c(-2, 1, 0, 1, 0)
y = c(0, -1, 2, -2, 1)
lines(x,y)
```

lines

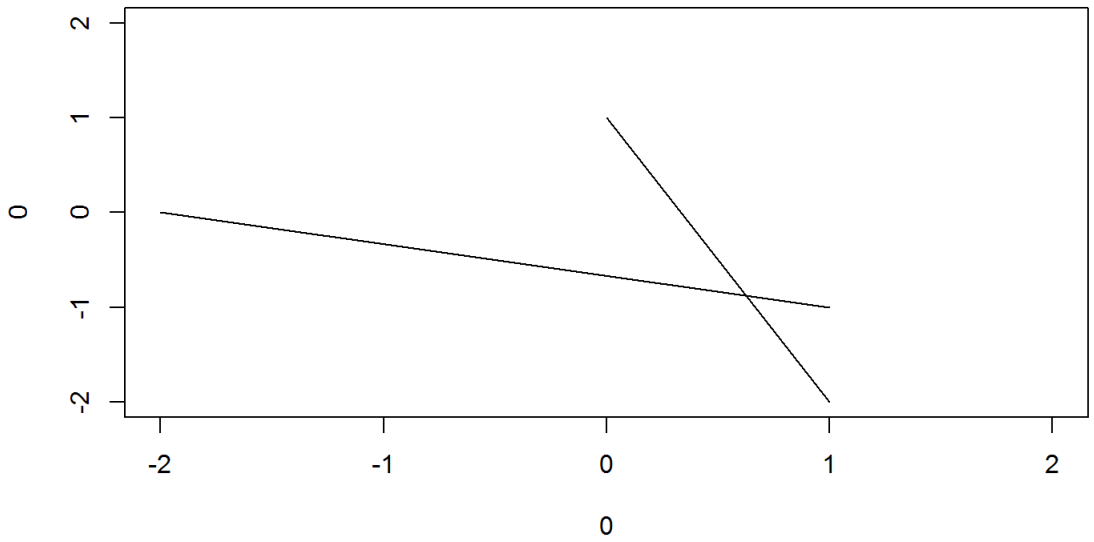


lines

- lines() 는 만들어진 그래프 위에 선을 그린다.
- NA 값을 이용하여 연결을 끊을 수 있다.

```
plot(0,0, type = 'n', xlim = c(-2,2), ylim = c(-2,2))
x = c(-2, 1, NA, 1, 0)
y = c(0, -1, NA, -2, 1)
lines(x,y)
```

lines



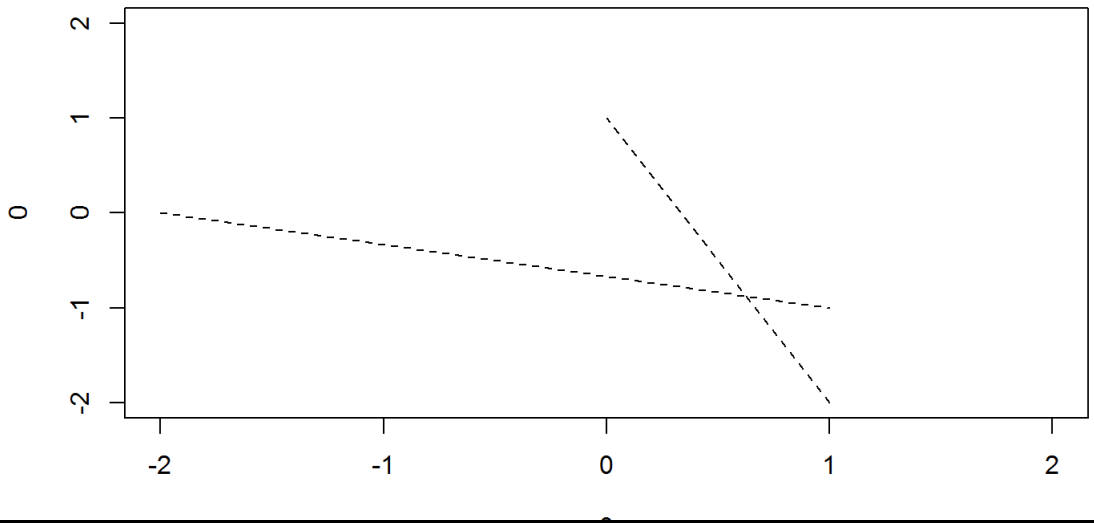
lines

- lines() 는 만들어진 그래프 위에 선을 그린다.
- `lty`를 이용하여 line의 형태를 바꾸어 줄 수 있다.

```
plot(0,0, type = 'n', xlim = c(-2,2), ylim = c(-2,2))
x = c(-2, 1, NA, 1, 0)
y = c(0, -1, NA, -2, 1)
lines(x,y, lty = 2)
```

lines

```
plot(0,0, type = 'n', xlim = c(-2,2), ylim = c(-2,2))
x = c(-2, 1, NA, 1, 0)
y = c(0, -1, NA, -2, 1)
lines(x,y, lty = 2)
```



lines

- 다음은 코드를 설명하시오

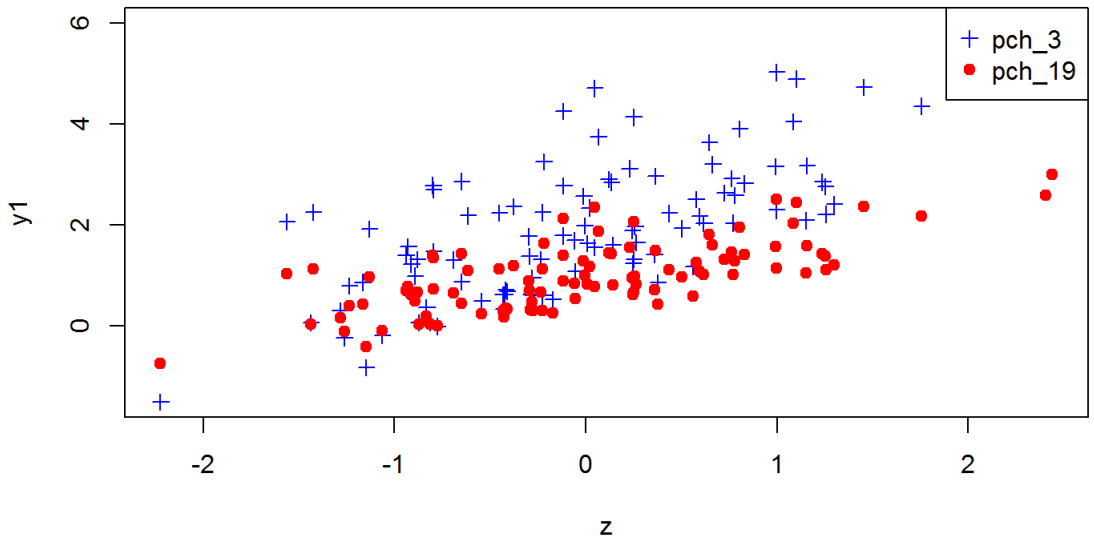
```
plot(0,0, type = 'n', xlim = c(1,5), ylim = c(0,2))
x = seq(1,5, by = 1)
abline(v = x, lty = 1:length(x))
```

legend

- legend()는 그림에 범례를 추가한다.
  - 위치, 범례, 색깔, 점모양, 선 등을 표시한다.
  - 세부사항은 ?legend에서 확인할 수 있다.

```
z = sort(rnorm(100))
y1 = 2+ x + rnorm(100)
plot(z, y1, col = "blue", pch = 3)
points(z, y1/2, col = "red", pch = 19)
legend("topright", c("pch_3", "pch_19"), col = c("blue", "red"),
       pch = c(3,19))
```

legend



legend

- 범례를 오른쪽 아래에 표시하시오

Graphic parameter: par()

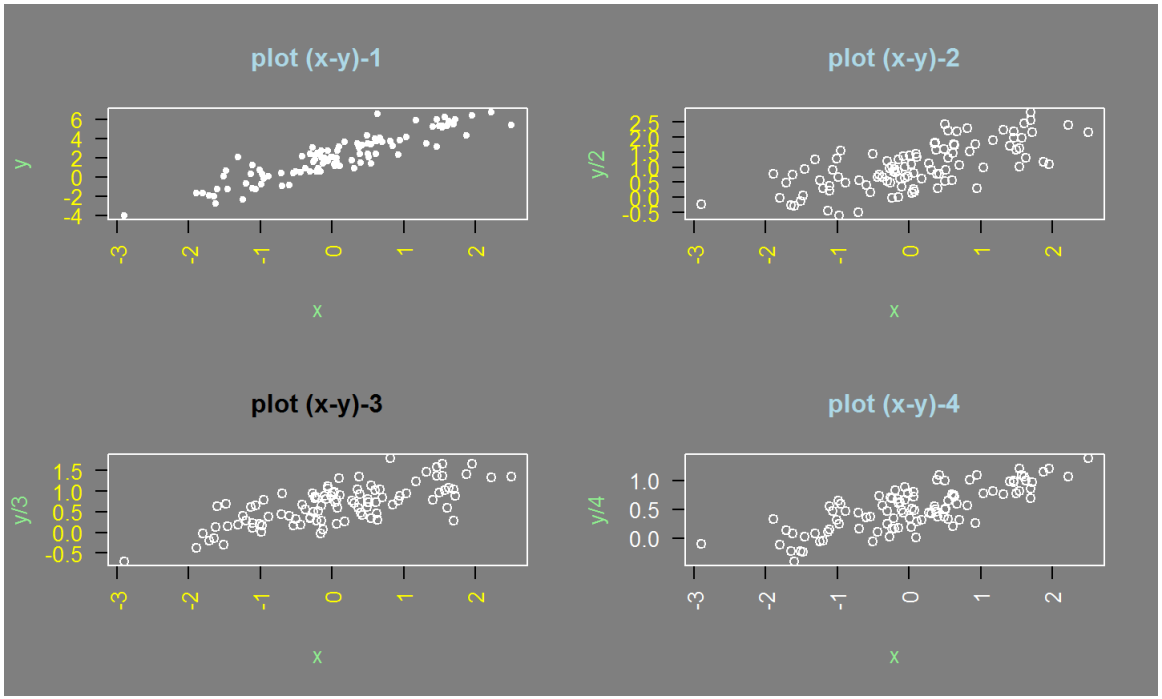
par()

- par() 는 R에서 표현되는 그림, 제목, 축의 문자들의 크기, 그림의 layout, 심볼들의 설정, 색상 팔레트의 설정등 그래픽 작업에 필요한 많은 파라메터 값을 조정해주는 함수다.
  - `par(mfrow = c(2,1))`: 그림은 행이 2개 열이 1개인 layout 이용
  - `par(cex = 1.2)`: 글자 크기(character expansion)를 1.2배로
  - `par(bg = 'gray90')`
  - `par(las = 2)`: 축 서식의 문자는 항상 축에 직교하게
  - `par( mai = c(1,2,3,4))`: 아래부터 시계방향을 그림 margin을 1,2,3,4로
- `?par` 로 자세한 사항을 확인할 수 있다.

par()

```
par (mfrow = c(2,2), bg = 'gray50', col = 'white',
     col.main = 'lightblue', col.axis = 'yellow',
     col.lab = 'lightgreen')
x = rnorm(100)
y = 2+2*x + rnorm(100)
plot(x,y, main = "plot (x-y)-1", pch = 20)
y = 2+x + rnorm(100)
plot(x,y/2, main = "plot (x-y)-2")
y = 2+x + rnorm(100)
plot(x,y/3, main = "plot (x-y)-3", col.main = 'black')
y = 2+x + rnorm(100)
plot(x,y/4, main = "plot (x-y)-4", col.axis = 'white')
```

par()



Visualization of K-nearest neighborhood method

Regression

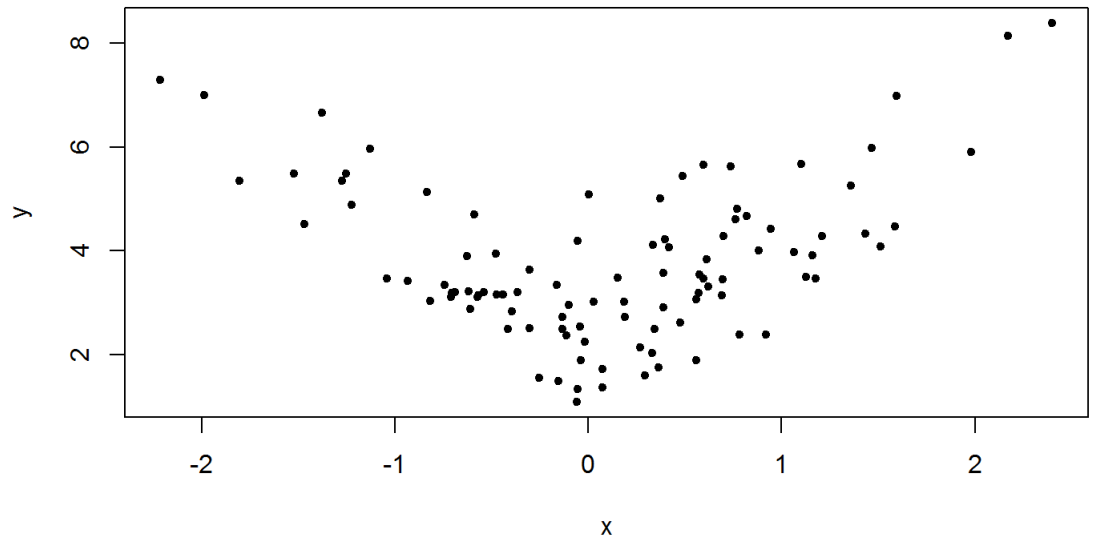
- 반응변수와 설명변수가 각각 1차원인 회귀분석 문제를 생각해보자.
- 편의상 회귀모형을 다음과 같이 설정하였다.

( )

- R에서는 다음과 같이 데이터를 100쌍
  생성하였다.

```
set.seed(1)
x <- sort(rnorm(100))
y<- 3+x^2 + rnorm(100)
plot(x, y, pch = 20)
```

Regression



Regression

- 이 데이터를 이용하여 선형회귀모형에 적합을 한 후 적합된 모형을 그려보자.
- 모형의 적합은 `lm()` 함수를 이용한다.

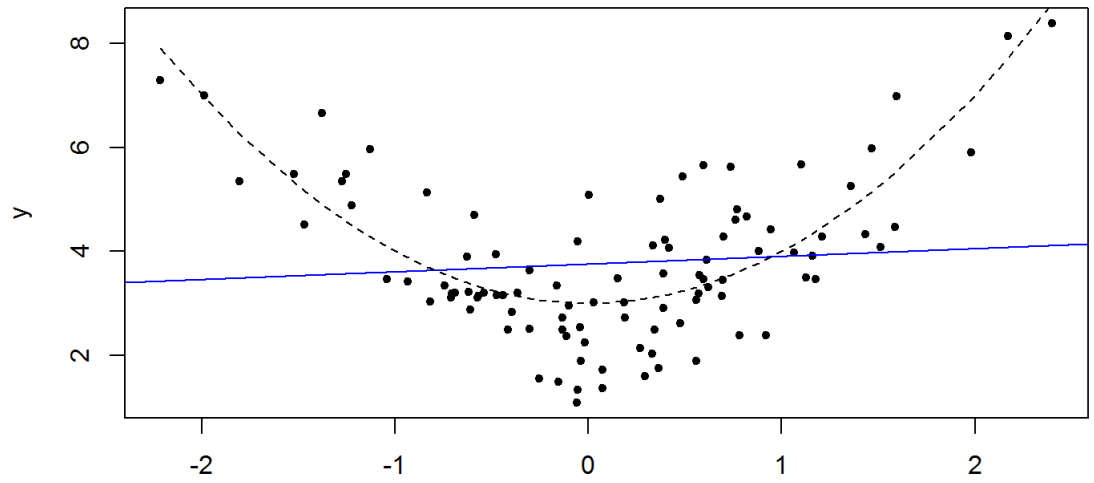
```
fit <- lm(y~x)
fit$coefficient
```

```
## (Intercept)           x
##   3.7565367   0.1488341
```

- `lm()` 함수의 결과물은 리스트 형태로 `fit` 변수로 저장된다.
- `fit`은 적합된 모형 계수를 `coefficient`의 이름으로 가지고 있다.
- `fit$coefficient`로 모형 계수 값을 호출할 수 있다.

Regression

```
plot(x, y, pch = 20)
abline(a = fit$coefficients[1], b = fit$coefficients[2], col = 'blue')
yTrueMean <- 3+ x^2
lines(x, yTrueMean , lty=2, col='black')
```



Regression

- k-근방 알고리즘

$$\hat{y}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

- $\hat{y}(x)$:설명변수 값이 x일 때, y의 예측값
- $N_k(x)$: x 값에서 k번째 가까운 설명변수의 index 모임
- $N_k(x)$를 구하기 위해서 FNN 라이브러리에 있는 `knnx.index` 함수를 사용

Regression

```
library(FNN)
knnx.index(x, 0, k = 10)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   47   46   48   45   44   43   42   41   49    50
```

```
idx <- c(knnx.index(x, 0, k = 10))
idx
```

```
##  [1] 47 46 48 45 44 43 42 41 49 50
```

Regression

- $x = 0$ 일때 yhat 구하기

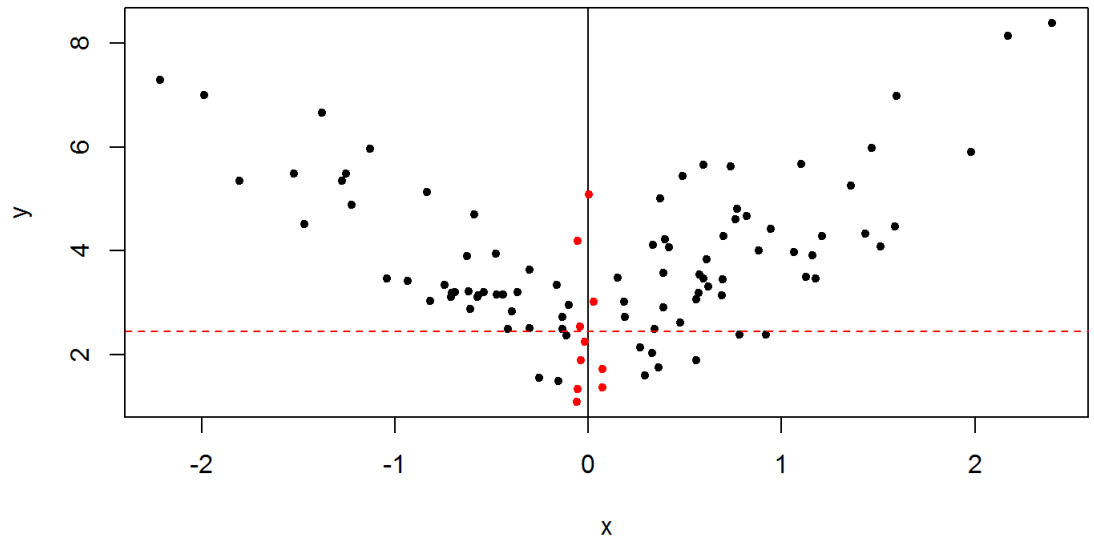
```
yhat <- mean( y[idx] )
yhat
```

```
## [1] 2.446989
```

Visualization of KNN

```
eval.point = 0
plot(x, y, pch = 20)
abline( v = 0, col = 'black')
idx<- c( knnx.index(x, eval.point, k = 10) )
points( x[idx], y[idx], col = 'red', pch = 20)
abline(h = mean(y[idx]), lty = 2, col = 'red')
mean(y[idx])
```

Visualization of KNN

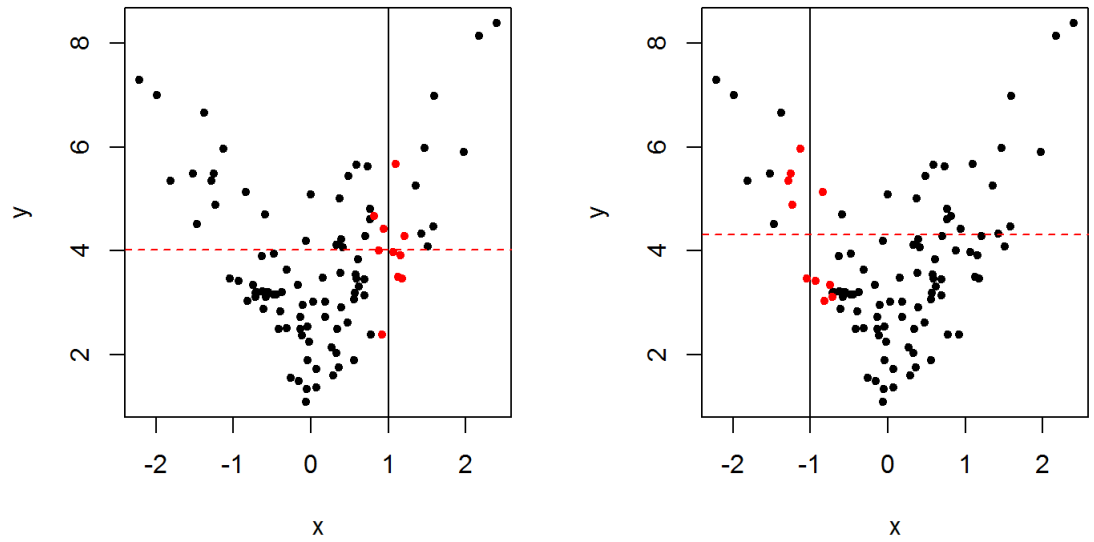



```
## [1] 2.446989
```

Visualization of KNN

- x 가 1과 -1 일 때 knn 결과를 표현하라.

Visualization of KNN

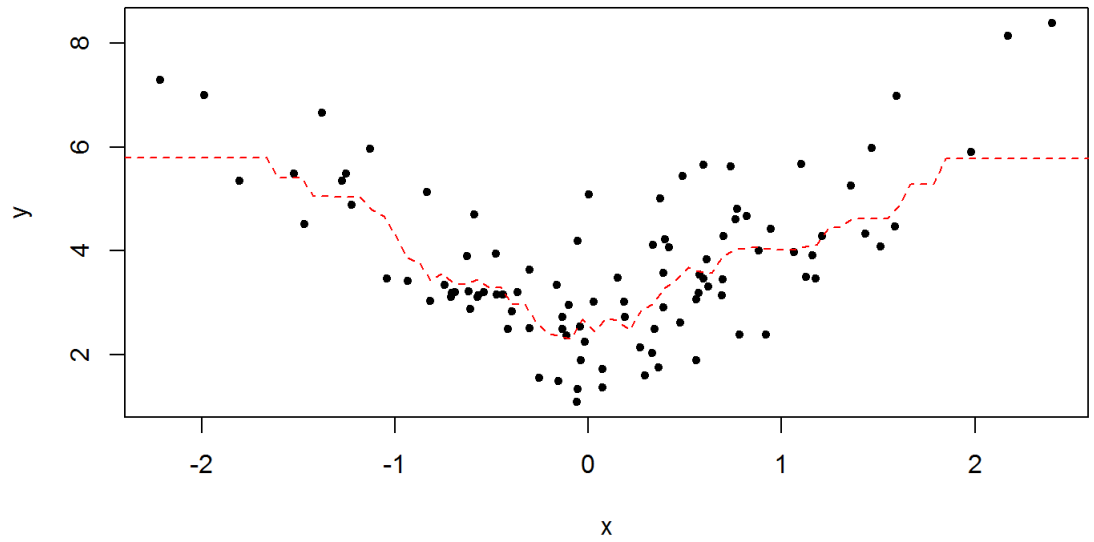


Visualization of KNN

- 반복문을 이용하여 -3에서 3사이의 여러개의 점에서 를 구 한후에 그 값을 표현하였다.

```
eval.n = 100
eval.point = seq(-3,3, length= eval.n)
plot(x, y, pch = 20)
idx.mat<- knnx.index(x, eval.point , k = 10)
yhat = rep(0,eval.n)
for (i in 1:eval.n)   yhat[i]<-mean(y[idx.mat[i,]])
lines(eval.point , yhat, type= 'l', lty = 2, col = 'red')
```

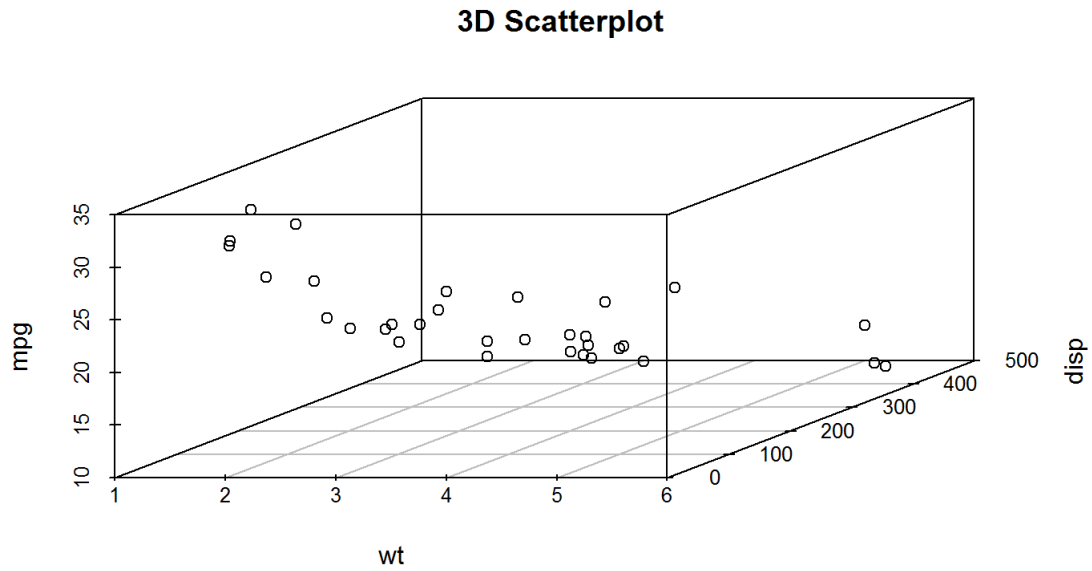
Visualization of KNN



3D plotting

3d scatterplot

```
library(scatterplot3d)
attach(mtcars)
scatterplot3d(wt, disp, mpg, main="3D Scatterplot")
```



Spinning 3d scatterplot

```
library(rgl)
attach(mtcars)
plot3d(wt, disp, mpg, col="red", size=3)
```

- color를 다음과 같이 할당해보자.

```
mypal = c('blue', 'red', 'green')
class(mtcars$cyl)
factor(mtcars$cyl)
mypal[factor(mtcars$cyl)]
plot3d(wt, disp, mpg, col= a[factor(mtcars$cyl)], size=10)
```

- `mycol`은 색상을 저장하고, `mycars$cyl`의 정보를 이용해서 색상을 데이터 순서대로 나열한다.
- 원래 `mtcars$cyl`이 문자열이나 `as.factor()` 를 통해 팩터형으로 바꾸어서 순서를 가지도록 만든다.

Draw 3d a surface

```
library(rgl)
z <- 2 * volcano        # Exaggerate the relief
x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
## Don't draw the grid lines :  border = NA
par(bg = "slategray")
persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE,
      ltheta = -120, shade = 0.75, border = NA, box = FALSE)
```

- persp 함수는 입력된 이차원 좌표상의 높이 값을 이용하여 표면을 그려준다.

Spinning 3d a surface

```
persp3d(x, y, z, col = "green3")
```