R Note Lec 2-3

Luo Beier

目录

1 R 语言

1.1 Data Structure

Use **mode()** or **typeof()** function to find the type of the object

1.1.1 Vectors

1.1.1.1 1. Vectors Creation

- Directly Creation:

```
(c1 <- 1:10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- Sequential vector

```
  (vec_seq <- 1:10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
  (vec_seq <- seq(1,10,by=1)) # 从 1 到 10, 每个数之间相差为 1
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- rep()

```
  (vec_zeros <- rep(0,times=100)) #0 重复 100 遍
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
  (vec_pattern <- rep(1:3,4)) #1 到 3 重复 4 遍
```

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

```
  (vec_pattern2 <- rep(1:3,4,2)) #1 到 3 重复 4 遍其中每个数重复 2 遍
```

```
## [1] 1 2
```

- Created using the vector function c():

```
(c1 <- c(1,2,3))
```

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

- You can convert from other data types using the as.vector() function:

```
  # 创建一个数据框
df <- data.frame(num=seq(1:5))
df
```

```
##   num
## 1   1
## 2   2
## 3   3
## 4   4
## 5   5
```

```
  # 数据框转换为向量
c1 <- as.vector(df$num) # 默认为数字向量
c1
```

```
## [1] 1 2 3 4 5
```

```
c1 <- as.vector(df$num,mode='character') # 指定类型，只要不报错就可以生成指定类型的向量
c1
```

```
## [1] "1" "2" "3" "4" "5"
```

```
c1 <- seq(1:5)
c1[2] # 访问第二个元素
```

1.1.1.2 2. Access to Vector Elements

```
## [1] 2
```

```
names(c1) <- LETTERS[1:5] # 对向量进行命名
# 如果元素有名字，也可以通过元素名访问
c1
```

```
## A B C D E
## 1 2 3 4 5
```

```
c1['C']  # 元素名要用引号包围
```

```
## C
## 3
```

```
c1[3]
```

```
## C
## 3
```

```


# 添加元素
c1 <- c(1,2,3,4,5) # 创建一个向量
(c1 <- c(c1,5)) # 追加一个元素
```

1.1.1.3 3. Insert and Delete Vector Elements

```
## [1] 1 2 3 4 5 5
```

```
c1 <- c(c1,c(5,6)) # 追加一个向量
c1
```

```
## [1] 1 2 3 4 5 5 5 6
```

```
c1 <- c(c1[1:2],c(5,6),c1[3:5]) # 指定位置来添加的元素 # 在第 2 个元素和第 3 个元素间插入了
c1
```

```
## [1] 1 2 5 6 3 4 5
```

```
c1 <- append(c1,8) # 在向量最后追加一个元素 8
c1
```

```
## [1] 1 2 5 6 3 4 5 8
```

```
c1 <- append(c1,c(11,22)) # 在向量后追加向量
c1
```

```
##  [1]  1  2  5  6  3  4  5  8 11 22
```

```
c1 <- append(c1,35,3) # 在第 3 个元素后插入新元素，也可以插入向量
c1
```

```
##  [1]  1  2  5 35  6  3  4  5  8 11 22
```

```
# 删除元素
c1 <- c1[-1] # 从向量中指定位置为 1 的元素
c1
```

```
##  [1]  2  5 35  6  3  4  5  8 11 22
```

```
c1 <- c1[-c(2:3)] # 可以给定一个位置向量来删除多个元素
c1
```

```
## [1]  2  6  3  4  5  8 11 22
```

```
c1 <- c1[c(3:5)] # 与上面的方式相反，保留想要的元素
c1
```

```
## [1] 3 4 5
```

```
c1 <- c(1,2,3,4,5) # 创建一个向量
c1[1] <- 11 # 修改第一个元素的值为 11
c1
```

1.1.1.4 4. Change The Value of the Element in The Vector

```
## [1] 11  2  3  4  5
```

```
c1[2:5] <- 11 # 一次性修改多个元素的值修改为同一个值
c1
```

```
## [1] 11 11 11 11 11
```

```
c1[2:5] <- c(21,22,23) # 修改多个元素的值时，为每个元素指定一个值
```

```
## Warning in c1[2:5] <- c(21, 22, 23): 被替换的项目不是替换值长度的倍数
```

```
c1
```

```
## [1] 11 21 22 23 21
```

```
c1[c1 > 3] <- 11 # 按条件修改元素的值，符合条件的元素都被重新赋值
c1
```

```
## [1] 11 11 11 11 11
```

1.1.2 Boolean Vectors

TRUE or **FALSE**

```
x <- c(TRUE, TRUE, FALSE, TRUE)
x
```

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

```
x <- (c(1, 1, -2, 2) > 0)
x
```

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

```
x <- c(T, T, F, T) # Equivalent, but not recommended
x
```

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

```
'x <- c(True, False)' # error # 要全部大写
```

```
## [1] "x <- c(True, False)"
```

```
x <- c("TRUE", "FALSE") # character
x
```

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

1.1.3 Factors

【R 语言】R 中的因子（factor）- 知乎 (zhihu.com)

1.1.4 Matrix

- A matrix is a rectangular table of the same type.

```
mat <- matrix(data = c(2, 4, 6, 7, 8, 10), nrow = 2, ncol = 3, byrow = FALSE) # 创建一个
mat
```

```
##      [,1] [,2] [,3]
## [1,]    2    6    8
## [2,]    4    7   10
```

```
(mat[2,]) # 矩阵的第二行
```

```
## [1]  4  7 10
```

```
(mat[,3]) # 矩阵的第三列
```

```
## [1]  8 10
```

```
(mat[2,3]) # 矩阵上 (2,3) 位置的元素
```

```
## [1] 10
```

1.1.5 Array

Arrays are similar to matrices, but can have more than two dimensions.

```
dim1 <- c("A1", "A2", "A3", "A4")
dim2 <- c("B1", "B2", "B3")
dim3 <- c("C1", "C2")
arr <- array(1:24, c(4, 3, 2), dimnames = list(dim1, dim2, dim3)) #c(4,3,2) 代表 Array
```

1.1.6 List

List

- A list is an ordered collection of data of arbitrary types.

```
mylist <- list(first = c(1, 3, 5),
               second = c("one", "three", "five"),
               third = "end")
names(mylist)
```

```
## [1] "first"  "second" "third"
```

```
mylist
```

```
## $first
## [1] 1 3 5
##
## $second
## [1] "one"   "three" "five"
##
## $third
## [1] "end"
```

```
typeof(mylist)
```

```
## [1] "list"
```

```
typeof(mylist[1])
```

```
## [1] "list"
```

```
typeof(mylist$first)
```

```
## [1] "double"
```

```
a <- list(a = 1:3,
          b = "a stgring",
          c = pi,
          d = list(-1,-5) )
a[1:2]
```

```
## $a
## [1] 1 2 3
##
## $b
```

```
## [1] "a stgring"
```

```
a[4]
```

```
## $d
## $d[[1]]
## [1] -1
##
## $d[[2]]
## [1] -5
```

```
a[[4]]
```

```
## [[1]]
## [1] -1
##
## [[2]]
## [1] -5
```

```
a[[4]][1]
```

```
## [[1]]
## [1] -1
```

```
a[[4]][[1]]
```

```
## [1] -1
```

1.1.7 Data Frame

- Data frame is a rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.

```
df <- data.frame(name      = c("Alice", "Bob", "Carl", "Dave"),
                 age       = c(23, 34, 23, 25),
                 marriage  = c(TRUE, FALSE, TRUE, FALSE),
                 color     = c("red", "blue", "orange", "purple"))
df
```

```
##    name age marriage  color
## 1 Alice  23     TRUE    red
## 2   Bob  34    FALSE   blue
## 3  Carl  23     TRUE orange
## 4  Dave  25    FALSE purple
```

1.1.7.1 1. Creation

```
time <- 1:3
value1 <- c(1, 2, 2)
value2 <- c(2, 0, 2)

data <- data.frame(time, value1, value2) # 一定要用长度相同的向量
data
```

```
##   time value1 value2
## 1    1      1      2
## 2    2      2      0
## 3    3      2      2
```

```
# 从索引访问第 2 行
data[2,]
```

1.1.7.2 2. Data Access

```
##   time value1 value2
## 2    2      2      0
```

```
# 从列名访问列
data$value1
```

```
## [1] 1 2 2
```

```
# 从行名访问行
data["3", ]
```

```
##   time value1 value2
## 3    3      2      2
```

```
# 访问指定的其他行列
# 只能通过索引操作，不能从行名、列名操作，行名、列名都是字符类型，不支持-操作符，如 data[-"2
data[-2,]
```

```
##   time value1 value2
## 1    1      1      2
## 3    3      2      2
```

```


rownames(data)
```

1.1.7.3 3. Gets the row name and column name

```
## [1] "1" "2" "3"
```

```
colnames(data)
```

```
## [1] "time"   "value1" "value2"
```

```
# 通过赋值可以对行名、列名进行更改
```

1.2 Function

1.2.1 Basic R function

- Summary functions include the following

```
sum(1:8)
prod(3:5)
min(c(1, 5, -10, 300))
max(c(9, 2, -2))
range(c(1, 5, -10, 300))
mean(1:10)
var(1:10)
sd(1:10)
median(1:10)
quantile(1:10)
```

- Functions for Boolean objects

```
which(c(1, 6, -3, -7, 0) <= 0)
any(c(1, 6, -3, -7, 0) <= 0)
all(c(1, 6, -3, -7, 0) <= 0)
```

1.2.2 Exercise

Exercise

- Consider the ages of British monarchs:

```
name <- c("Anna", "George I", "George II", "George III", "George IV", "William IV",
    "Victoria", "Edward VII", "George V", "Edward VIII", "George VI", "Elizabeth II")
age <- c(49, 67, 76, 81, 67, 71, 81, 68, 70, 77, 56, 96)
```

- What is the oldest age? What is the average age?
- Define a Boolean vector: which ages are greater than 70?
- Who is the longest-lived monarch?

```
name <- c("Anna", "George I", "George II", "George III", "George IV", "William IV",
"Victoria", "Edward VII", "George V", "Edward VIII", "George VI", "Elizabeth II")
age <- c(49, 67, 76, 81, 67, 71, 81, 68, 70, 77, 56, 96)
```

```
# 第一题
max(age)
```

```
## [1] 96
```

```
mean(age)
```

```
## [1] 71.58333
```

```
# 第二题
age > 70 #Be careful about the boolean vector!
```

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

```
# 第三题
name[which(age==96)]
```

```
## [1] "Elizabeth II"
```

Math functions

- Trigonometric functions

```
sin(pi/2)
tan(pi/4)
cos(pi)
```

- Special functions

```
factorial(4)
exp(0:3)
log(1:10)
round(3.833)
round(3.833, digits = 1)
ceiling(3.83)
floor(3.83)
choose(4, 2)
```

1.2.3 Sort function

```
x <- c(13,14,12,11,15)
y <- sort(x) # 从小到大排序
y
```

```
## [1] 11 12 13 14 15
```

```
r <- rank(x) # 每个位置的数的排序位置（从小到大）
r
```

```
## [1] 3 4 2 1 5
```

```
o <- order(x) # 从小到大排序后每个数对应原向量的位置
o
```

```
## [1] 4 3 1 2 5
```

```
x[o]
```

```
## [1] 11 12 13 14 15
```

1.2.4 Combine matrices or arrys

- rbind() 按行来合并

```
x <- c(1:5)
y <- x*2
rbind(x,y)
```

```
##   [,1] [,2] [,3] [,4] [,5]
## x    1    2    3    4    5
## y    2    4    6    8   10
```

- cbind() 按列来合并

```
cbind(x,y)
```

```
##      x  y
## [1,] 1  2
## [2,] 2  4
## [3,] 3  6
## [4,] 4  8
## [5,] 5 10
```

1.2.5 Exercise

Exercise

- Use R to calculate the following matrices:

$$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$$

$$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}^2$$

$$2\times\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}-\begin{pmatrix}7 & 8\\6 & 5\end{pmatrix}^T$$

```
# Exercise 1
(a <- matrix(c(0,1,1,0),nrow=2))
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
```

```
(b <- matrix(c(1,3,2,4),nrow=2))
```

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

```
(c <- matrix(c(0,1,1,0),nrow=2))
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
```

```
a %*% b %*% c
```

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

```
#Exercise 2
a %*% a
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
#Exercise 3
d <- matrix(c(7,6,8,5),nrow=2)
t(d)
```

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

```
2 * b - t(d)
```

```
##      [,1] [,2]
## [1,]   -5   -2
## [2,]   -2    3
```

1.2.6 Matrix calculation: Squires matrics

Exercise

- Use R to calculate the following matrices:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^2$$

$$2 \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} - \begin{pmatrix} 7 & 8 \\ 6 & 5 \end{pmatrix}^T$$

Matrix calculation: Square matrics

- A square matrix is a matrix with equal number of rows and columns.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

```
A = matrix(1:9, nrow = 3)
A
```

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

- Diagonal vector of A: $\text{diag}(A) = (a_{11}, a_{22}, \dots, a_{nn})$

```
diag(A)
```

```
## [1] 1 5 9
```

Matrix calculation: Square matrics

- Trace of a square matrix:

$$\text{trace}(A) = \sum_{i=1}^{n} a_{ii}.$$

```
sum(diag(A))
```

```
## [1] 15
```

- Diagonal matrices:

$$A = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

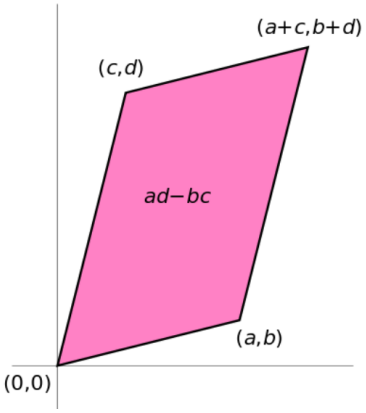
- The diagonal matrix generated from a vector $\mathbf{a} = (a_1, \dots, a_n)$:

```
a <- c(3, 1, 2)
diag(a)
```

```
##      [,1] [,2] [,3]
## [1,]    3    0    0
## [2,]    0    1    0
## [3,]    0    0    2
```

Determinant

- Determinant of a matrix: `det()`




```
a <- 5
b <- 1
c <- 2
d <- 6
A <- matrix(c(a, b, c, d),
            nrow = 2,
            byrow = TRUE)
A
```

```
##      [,1] [,2]
## [1,]    5    1
## [2,]    2    6
```

```
det(A)
```

```
## [1] 28
```

Inverse of a matrix

- In linear algebra, an $n \times n$ square matrix is called *invertible* if there exists an $n \times n$ square matrix B such that $AB = BA = I_n$, where I_n is the $n \times n$ identity matrix.
- If $\det(A) \neq 0$, then A is invertible.
- The matrix B is uniquely determined by A, and is denoted by A^{-1}.
- If A is invertible, then for any n-dim vector $\mathbf{b}$, the linear equation $Ax = \mathbf{b}$ has a unique solution, which is given by $x = A^{-1}\mathbf{b}$.

```
(B <- solve(A))
```

```
##             [,1]        [,2]
## [1,]  0.21428571 -0.03571429
## [2,] -0.07142857  0.17857143
```

```
round(A %*% B, 10)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
b = c(4, 7)
(x = solve(A, b))
```

```
## [1] 0.6071429 0.9642857
```

Eigenvalues and Eigenvectors

- Mathematically, x is an eigenvector of A if there exists a real number λ such that
$$Ax = \lambda x.$$
- Use `eigen()` function

```
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 7 4
##
## $vectors
##            [,1]       [,2]
## [1,] -0.4472136 -0.7071068
## [2,] -0.8944272  0.7071068
```

```
lam <- eigen(A)$values
x <- eigen(A)$vectors
A %*% x[, 1] - lam[1] * x[, 1]
```

```
##      [,1]
## [1,]    0
## [2,]    0
```

1.2.7 Exercise

- Find $\det(A^{-1})$.
- What's the relation between $\det(A^{-1})$ and $\det(A)$?
- Find the eigenvalues and eigenvectors of A^{-1}.
- Check whether $\det(A) = \prod_{i=1}^{n} \lambda_i$? Here, $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A.

```
#Exercise 1
A <- matrix(c(5,2,1,6),nrow=2)
(B <- solve(A))
```

```
##             [,1]        [,2]
## [1,]  0.21428571 -0.03571429
## [2,] -0.07142857  0.17857143
```

```
det(B)
```

```
## [1] 0.03571429
```

```
#Exercise 2
det(A) * det(B)
```

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

```
#Exercise 3
eigen(B)
```

```
## eigen() decomposition
## $values
## [1] 0.2500000 0.1428571
##
```

```
## $vectors
##             [,1]      [,2]
## [1,]  0.7071068 0.4472136
## [2,] -0.7071068 0.8944272
```

```
det(A) == prod(eigen(A)$values)
```

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

```
A <- matrix(data = c(3, 1, 2,
                     1, 4, 0,
                     2, 0, 9),
            nrow = 3)
```

- Check whether A is positive definite or not.

```
A <- matrix(c(3,1,2,
              1,4,0,
              2,0,9),
            nrow=3)
M <- any(eigen(A)$values>0)
if(M){
  print("A is positive definite!")
} else{
  print("A is not positive definite!")
}
```

```
## [1] "A is positive definite!"
```

1.2.8 Function of characters

Functions for characters

- Character functions extract information from textual data.

```
x <- c("ab", "cde", "fghij")
nchar(x[3])
```

```
## [1] 5
```

```
y <- "abcdefg"
substr(y, 2, 4)
```

```
## [1] "bcd"
```

```
student <- c("Zhang, San", "Li, Si", "Wang, Wu", "Wang, Liu")
grep("Wang, ", student)
```

```
## [1] 3 4
```

grep() 能对向量中特定条件的元素进行查询

grep()

代码如下:

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
     fixed = FALSE, useBytes = FALSE, invert = FALSE)
```

grep()函数参数:

参数	功能
pattern	包含正则表达式的字符串
x	寻找匹配的字符向量，或者可以通过字符向量强制转换的对象。支持长向量
ignore.case	如果为FALSE，则模式匹配区分大小写；如果为TRUE，则在匹配期间忽略大小写
perl	如果为TRUE，使用perl匹配的正则表达式
value	如果为FALSE，则返回包含由grep确定的匹配的索引的向量，如果为TRUE，则返回包含匹配元素本身的向量
fixed	如果为TRUE，则pattern是要按原样匹配的字符串
useBytes	如果为TRUE，则匹配是逐字节而不是逐字符完成的
invert	如果为TRUE，则返回不匹配的元素的索引或值

1.2.9 apply() Family

首先我们创建一个简单的数值矩阵

```
data <- matrix(1:20,5,4)
data
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

首先让我们从 apply() 开始

1.2.9.1 apply() apply 函数的结构是 apply(X,MARGINE,FUN,...)

这里，

- X 是指我们要对其进行操作的数据集
- MARGIEN 是指我们需指定是按行还是按列来进行操作
- MARGIEN = 1 代表按行操作；MARGIEN = 2 代表按列操作；MARGIEN = 1:2 代表对全体操作
- FUN 是指我们将要对 X 进行的操作函数

一个简单的示例：

```
mean_rows <- apply(data,1,mean)
mean_rows
```

```
## [1]  8.5  9.5 10.5 11.5 12.5
```

```
sum_cols <- apply(data,2,sum)
sum_cols
```

```
## [1] 15 40 65 90
```

```
all_sqrt <- apply(data,1:2,sqrt)
all_sqrt
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 1.000000 2.449490 3.316625 4.000000
## [2,] 1.414214 2.645751 3.464102 4.123106
## [3,] 1.732051 2.828427 3.605551 4.242641
## [4,] 2.000000 3.000000 3.741657 4.358899
## [5,] 2.236068 3.162278 3.872983 4.472136
```

apply 家族也可以用于多参数的函数：

```
fn = function(x1, x2, x3)
{
    return(x1^2 + x2 * x1 + x3)
}
b = 2
c = 1

# apply along each row:
row_fn <- apply(data, 1, fn, x2 = b, x3 = c)
row_fn
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    4    9   16   25   36
## [2,]   49   64   81  100  121
## [3,]  144  169  196  225  256
## [4,]  289  324  361  400  441
```

```
# apply along each column:
col_fn <- apply(data, 2, fn, x2 = b, x3 = c)
col_fn
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    4   49  144  289
## [2,]    9   64  169  324
## [3,]   16   81  196  361
## [4,]   25  100  225  400
## [5,]   36  121  256  441
```

1.3 Import data

1.3.1 Keyboard import

- 我们可以创建一个表格，并对表格进行编辑:

```
mydata <- data.frame(
age = numeric(0),
gender = character(0),
weight = numeric(0)
)
## mydata <- edit(mydata)
```

- 或者直接用代码输入

```
mydatatxt <- "
age gender weight
25 m 166
30 f 115
18 f 120
"
mydata <- read.table(header = TRUE, text = mydatatxt)
```

1.3.2 From a delimited text file: read.table()

- The format is

```
my_data <- read.table(file,options)
```

1.3.3 Exercise

Student grades: csv file (此处直接以文本显示，实际操作中需将文件放入 R 所读取的文件包中)

StudentID,First,Last,Math,Science,Social Studies

011,Bob,Smith,90,80,67

012,Jane,Weary,75,,80

010,Dan,"Thornton, III",65,75,70

040,Mary,"O'Leary",90,95,92

```
grades <- read.table("studentgrades.csv",
header=TRUE,
row.names="StudentID",
sep=",")  ##sep 代表两个数据之间的间隔符号，从而将数据区分开
```

- Build-in data sets

R 里有很多数据包可以调用数据

```
data(iris)
data(mtcars, package = "datasets")
```

attach and **detach** functions:

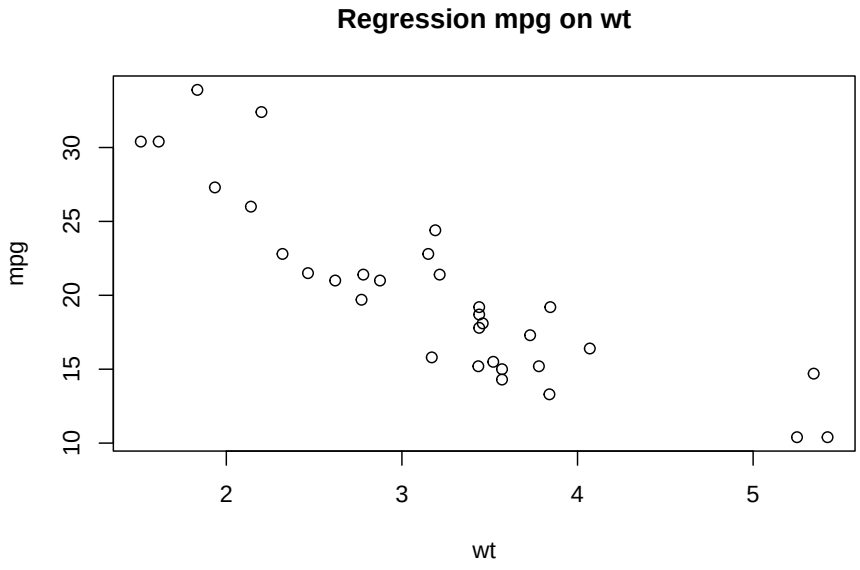
```
## If we directly input Sepal_Length    Error!
attach(iris)
Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

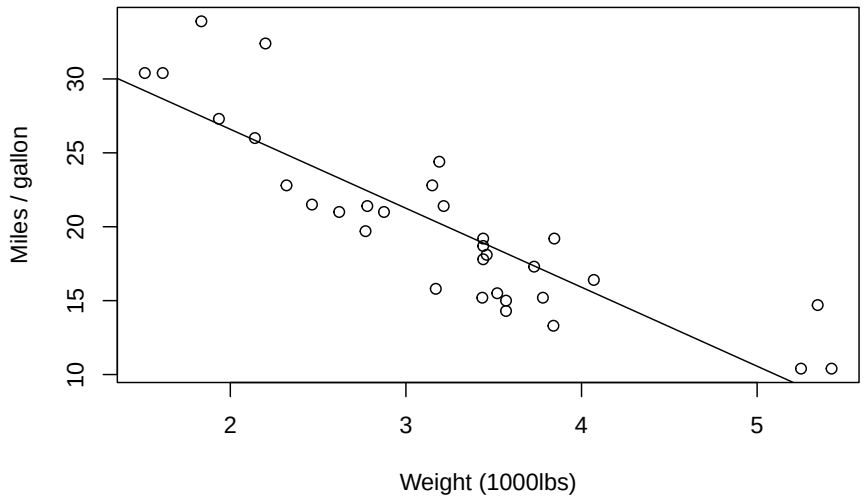
```
detach(iris)
```

1.4 Basic Graphs

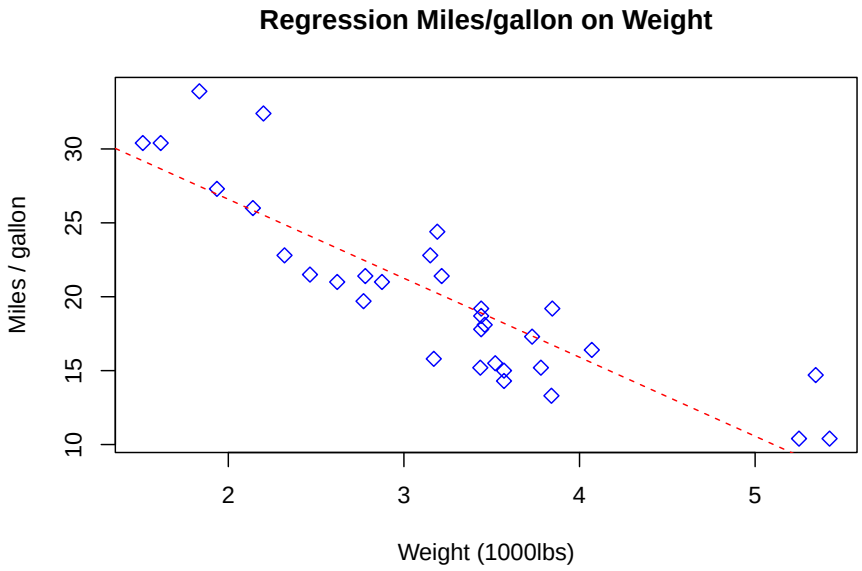
```
attach(mtcars)
plot(wt,mpg)
title("Regression mpg on wt")
```




```
plot(wt, mpg, xlab = "Weight (1000lbs)", ylab = "Miles / gallon") ## 改变横纵坐标名称
abline(lm(mpg~wt)) ## 画出线性回归线
```




```
plot(wt, mpg, col = "blue", pch = 5,
xlab = "Weight (1000lbs)", ylab = "Miles / gallon")
title("Regression Miles/gallon on Weight")
abline(lm(mpg~wt), col = "red", lty = 2)    ## 改变颜色, 形状
```




```
detach(mtcars)
```

1.5 ggplot2

```
library(tidyverse)
```

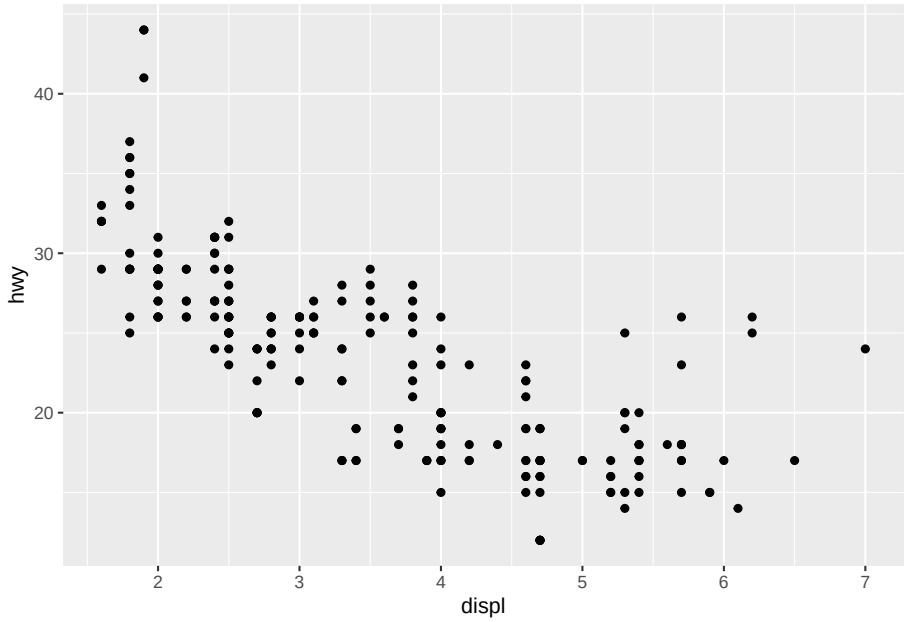
```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
mpg
```

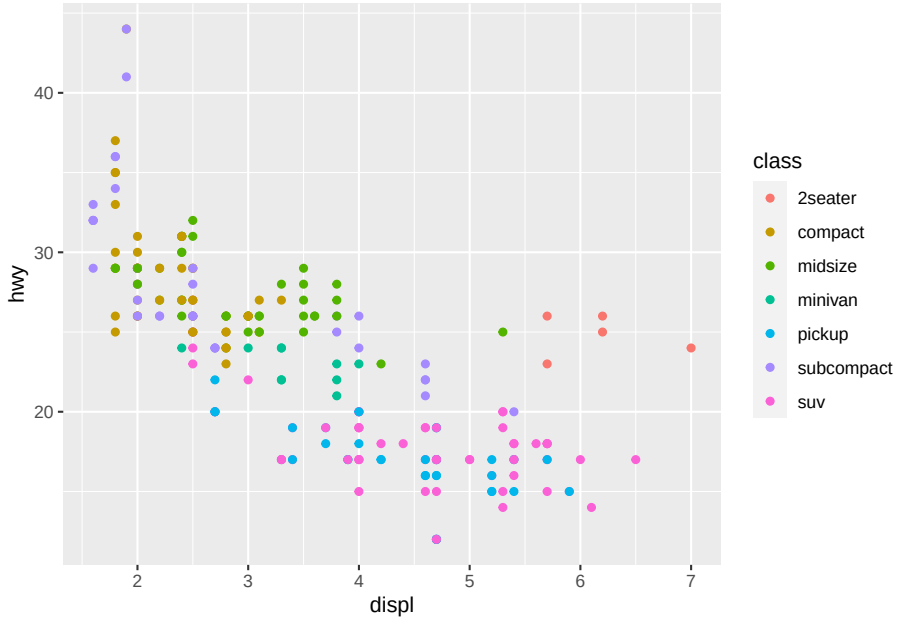
```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 224 more rows
```

```
ggplot(data = mpg)  ##Create a blank picture
```

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy))  ##ase() 中的代表对点的分类依据
```

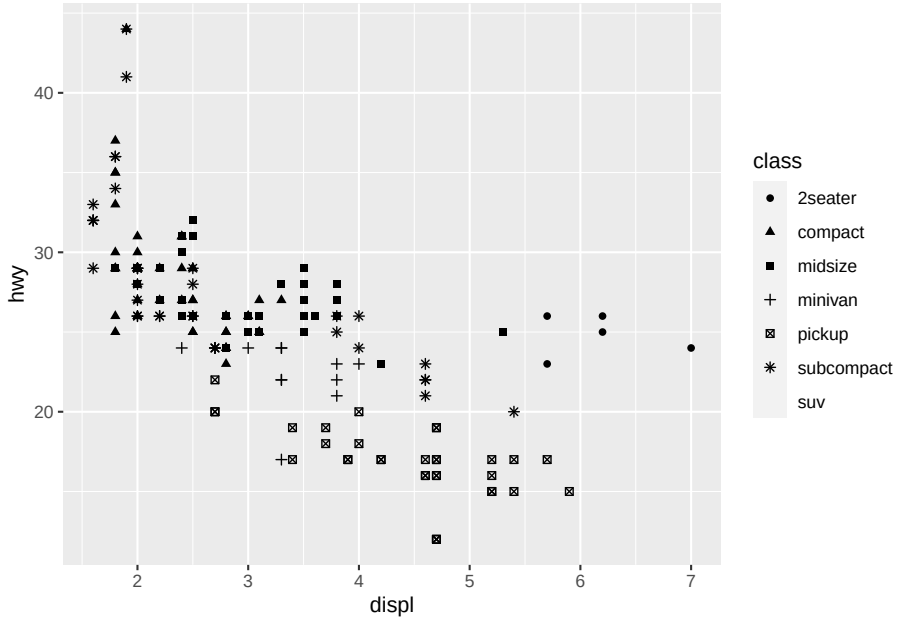



```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy, color = class)) ## 分类依据加上 color 按 cl
```

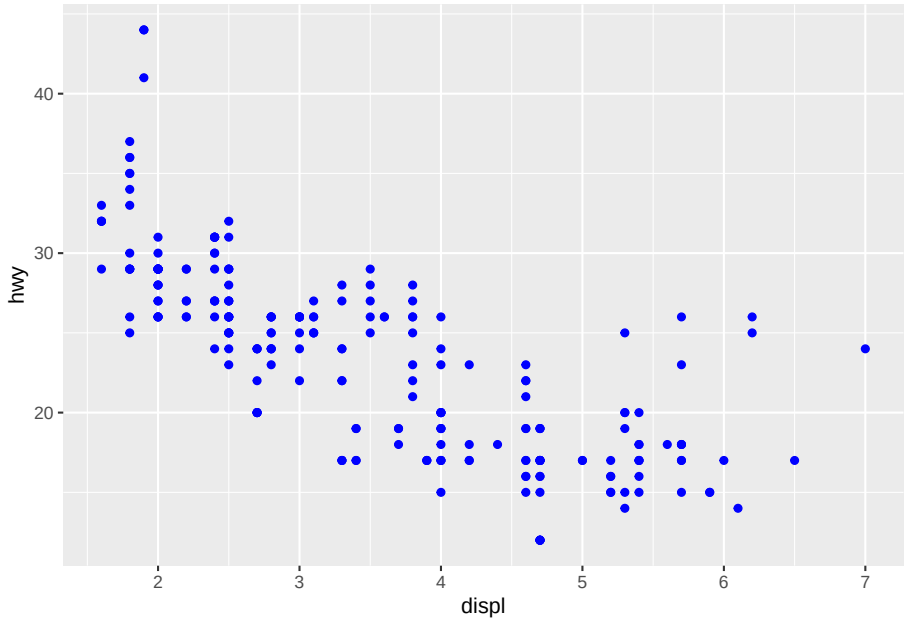



```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy, shape = class)) ## 分类依据加上 shape 按 cl
```

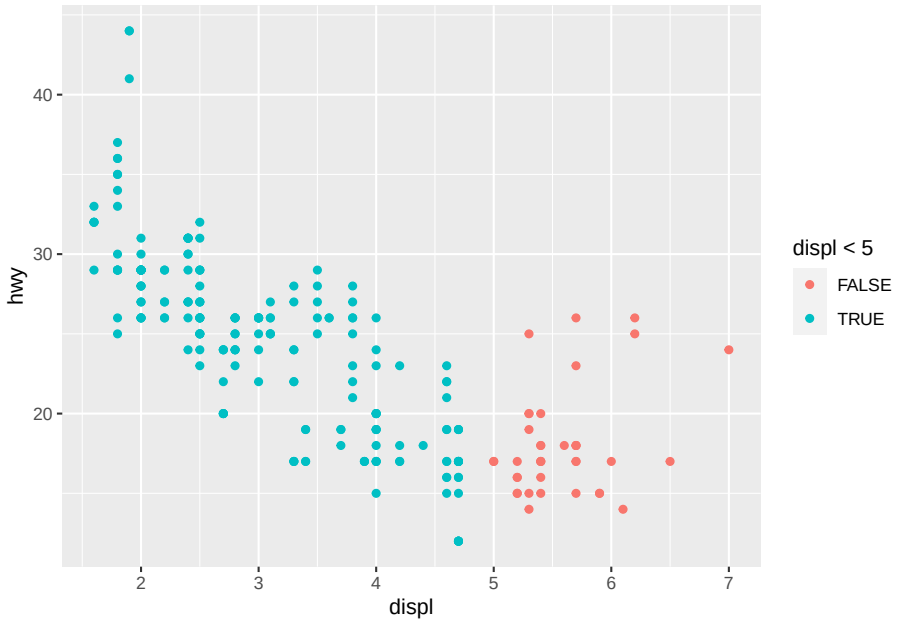
```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.

## Warning: Removed 62 rows containing missing values (geom_point).
```




```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```



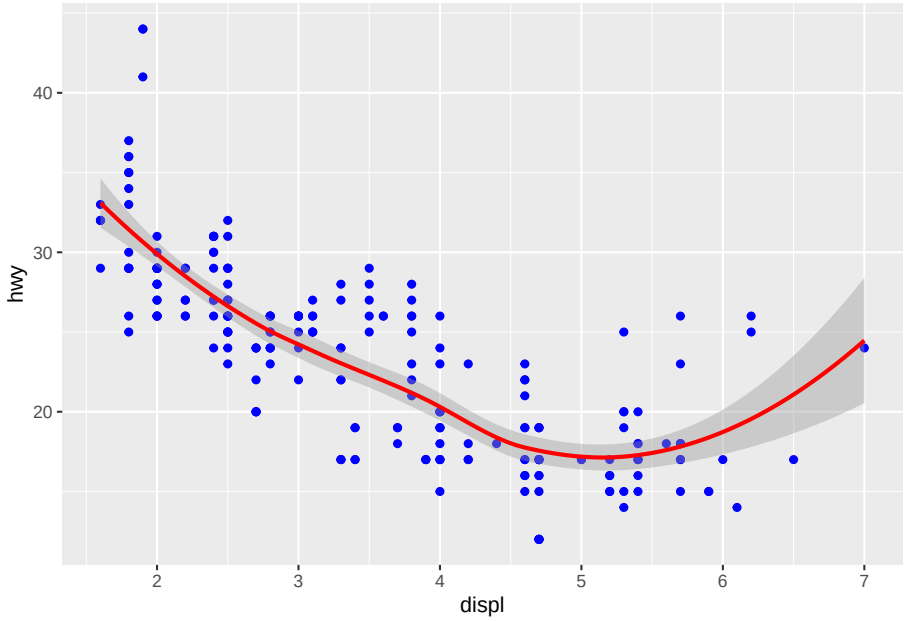
```
## 将 color 放在外面说明不将其作为分类依据而是对整体进行改色
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy, color = displ < 5))
```



Other geom functions

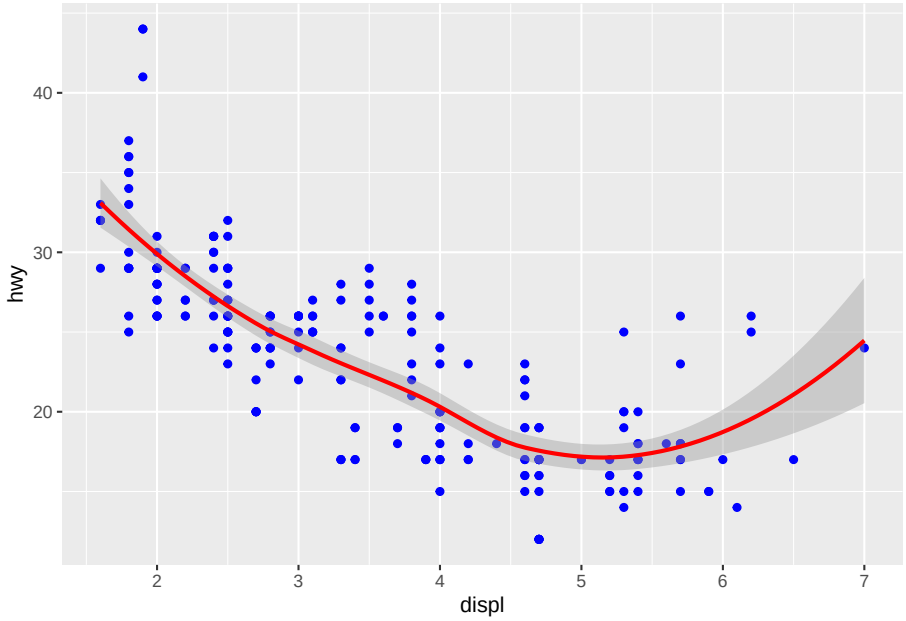
```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy), color = "blue") +
geom_smooth(mapping = aes(x = displ, y = hwy), color = "red")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



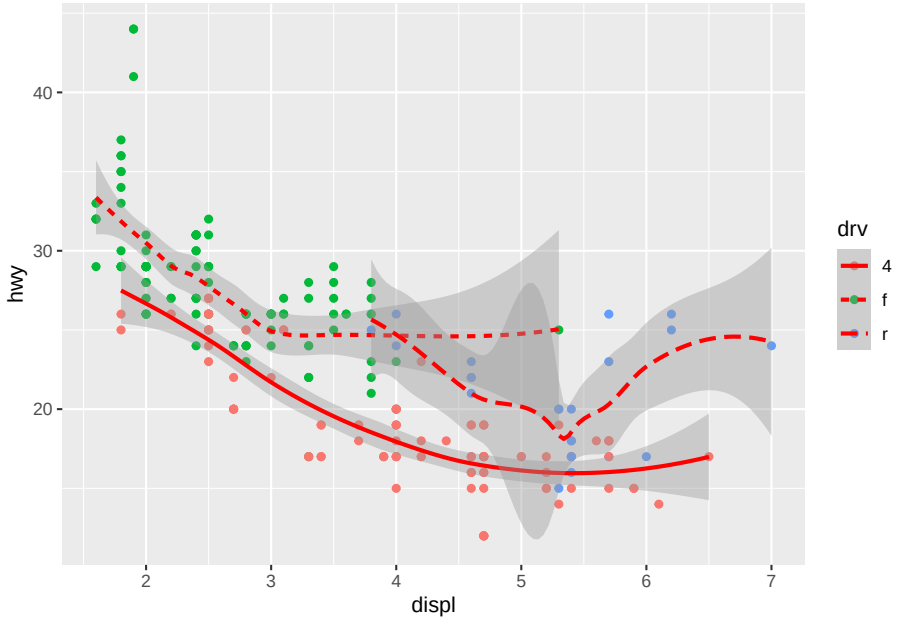

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
geom_point(color = "blue") +
geom_smooth(color = "red") ## 也可以进行全局的分类
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



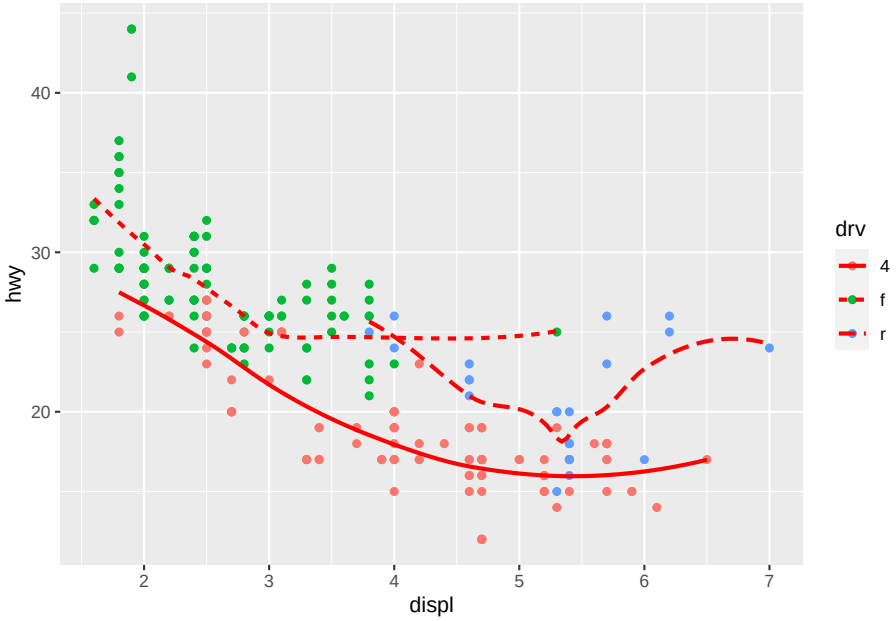

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, linetype = drv, color = drv)) +
geom_point() +
geom_smooth(color = "red")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
## 全局分类下进行局部分类
 ggplot(data = mpg, mapping = aes(x = displ, y = hwy, linetype = drv, color = drv))+
    geom_point()+
    geom_smooth(color = "red",se=FALSE) ## 除去阴影 (置信区间)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Common used geom_ type functions

function	graph	options
geom_bar	条形图	color, fill, alpha
geom_boxplot	箱线图	color, fill, alpha, notch, width
geom_density	密度图	color, fill, alpha, linetype
geom_hline	水平线	linetype, size, ...
geom_vline	垂直线	linetype, size, ...
geom_histogram	直方图	binwidth, bins, ...
geom_jitter	抖动点	shape, size, ...
geom_point	散点图	shape, size, ...
geom_rug	地毯图	side, ...
geom_smooth	拟合线	method, formula, ...

Common options

option	description
color	对点、线和填充区域的边界着色
fill	对填充区域着色
alpha	颜色的透明度，从 0（完全透明）到 1（不透明）
linetype	图案的线条，从 1 到 6。
size	点的尺寸和线的宽度
shape	点的形状，从 0, 1, 2, … 开始
position	条形图和点的位置
binwidth	直方图的宽度
notch	表示方块图是否缺口
sides	地毯图的安置，"b" = 底部等
width	箱线图的宽度

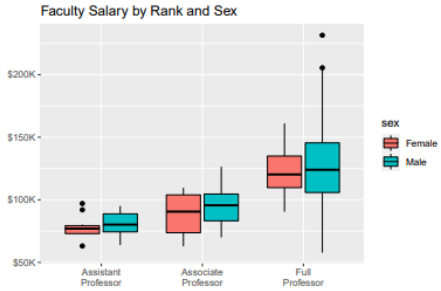
q

Options for geom_smooth()

option	description
method	使用的平滑函数。包括lm, glm, smooth, rlm, gam
formula	使用的公式。包括y ~ x, y ~ log(x)等
se	是否绘制置信区间
level	使用的置信区间水平
fullrange	是否涵盖全图，或仅仅是数据

Examples

```
data(Salaries, package = "carData")
ggplot(Salaries, aes(x = rank, y = salary, fill = sex)) +
  geom_boxplot(color = "black", notch = FALSE) +
  scale_x_discrete(breaks = c("AsstProf", "AssocProf", "Prof"),
                   labels = c("Assistant\nProfessor",  "Associate\nProfessor",  "Full\nProfessor"
  scale_y_continuous(breaks = 50000 * 1:4,  labels = c("$50K", "$100K", "$150K", "$200K")) +
  labs(title = "Faculty Salary by Rank and Sex", x = "", y = "")
```



Facet functions

- Sometimes relationships are clearer if groups appear in side-by-side graphs rather than overlapping in a single graph.
- You can create faceted graphs using the `facet_wrap()` and `facet_grid()` functions.

template	result
`facet_wrap(~var, ncol = n)`	将每个var水平排成n列的独立图
`facet_wrap(~var, nrow = n)`	将每个var水平排成n行的独立图
`facet_grid(rowvar ~ colvar)`	rowvar 和 colvar 组合的独立图
`facet_grid(rowvar ~ .)`	
`facet_grid(colvar ~ .)`	

Another example

```
library(ggplot2)
ggplot(Salaries, aes(x = yrs.since.phd, y = salary, color = rank, shape = rank)) +
  geom_point() +
  facet_grid(.~sex)
```