北京邮电大学
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Data Analytics with R

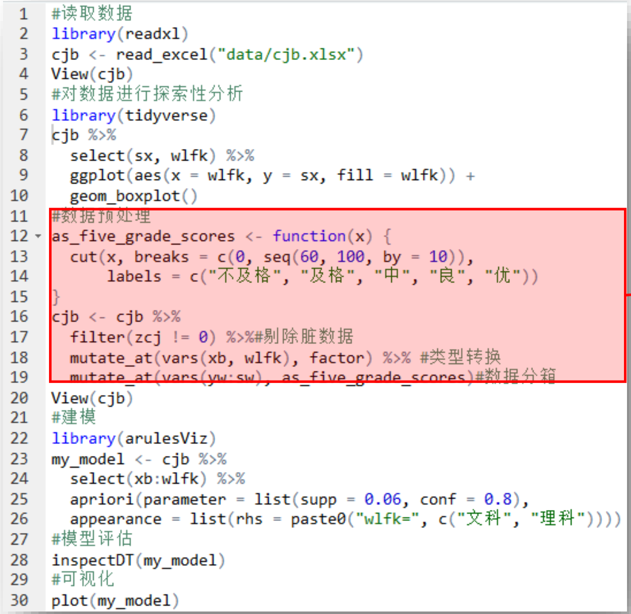
R语言数据分析

基础编程

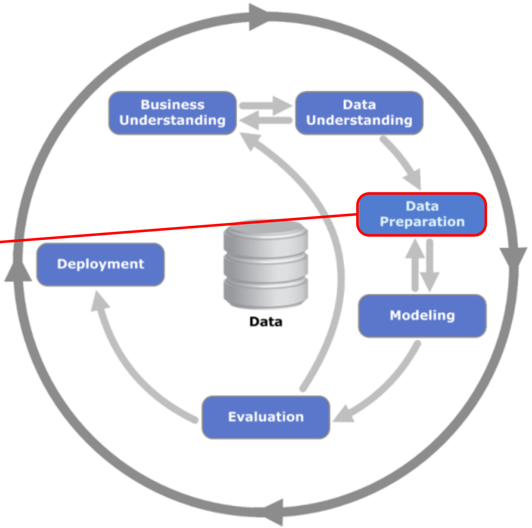
艾新波 / 2018 • 北京

课程体系

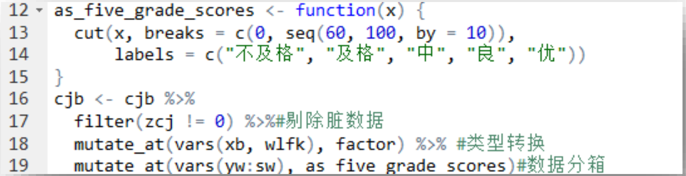
R语言数据分析Mini案例



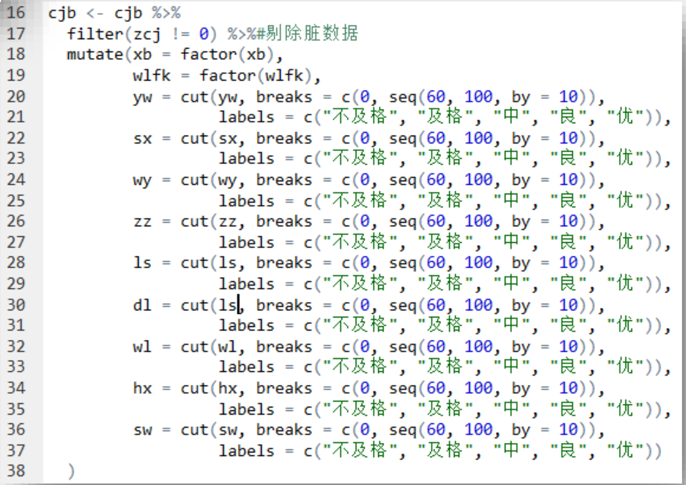
```r
#读取数据
library(readxl)
cjb <- read_excel("data/cjb.xlsx")
View(cjb)
#对数据进行探索性分析
library(tidyverse)
cjb %>%
  select(sx, wlfk) %>%
  ggplot(aes(x = wlfk, y = sx, fill = wlfk)) +
  geom_boxplot()
#数据预处理
as_five_grade_scores <- function(x) {
  cut(x, breaks = c(0, seq(60, 100, by = 10)),
      labels = c("不及格", "及格", "中", "良", "优"))
}
cjb <- cjb %>%
  filter(zcj != 0) %>%#剔除脏数据
  mutate_at(vars(xb, wlfk), factor) %>% #类型转换
  mutate_at(vars(yw:sw), as_five_grade_scores)#数据分箱
View(cjb)
#建模
library(arulesViz)
my_model <- cjb %>%
  select(xb:wlfk) %>%
  apriori(parameter = list(supp = 0.06, conf = 0.8),
  appearance = list(rhs = paste0("wlfk=", c("文科", "理科"))))
#模型评估
inspectDT(my_model)
#可视化
plot(my_model)
```



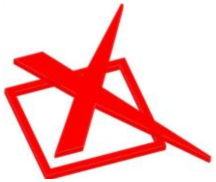
两种不同的写法



```r
as_five_grade_scores <- function(x) {
  cut(x, breaks = c(0, seq(60, 100, by = 10)),
      labels = c("不及格", "及格", "中", "良", "优"))
}
cjb <- cjb %>%
  filter(zcj != 0) %>%#剔除脏数据
  mutate_at(vars(xb, wlfk), factor) %>% #类型转换
  mutate_at(vars(yw:sw), as_five_grade_scores)#数据分箱
```



```r
cjb <- cjb %>%
  filter(zcj != 0) %>%#剔除脏数据
  mutate(xb = factor(xb),
         wlfk = factor(wlfk),
         yw = cut(yw, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         sx = cut(sx, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         wy = cut(wy, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         zz = cut(zz, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         ls = cut(ls, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         dl = cut(ls, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         wl = cut(wl, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         hx = cut(hx, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优")),
         sw = cut(sw, breaks = c(0, seq(60, 100, by = 10)),
                  labels = c("不及格", "及格", "中", "良", "优"))
  )
```



copy-and-paste

什么时候需要编写函数

You should consider writing a function whenever you’ve copied and pasted a block of code more than twice (i.e. you now have three copies of the same code).

—*Hadley Wickham, Chief Scientist at Rstudio*

图片来自：http://hadley.nz/

什么时候需要编写函数

事不过三

- **Rule of three:** the code can be copied once, but that when the same code is used three times, it should be extracted into a new procedure
- Do not Repeat Yourself (or DRY) principle
- Separation of Concerns

其它一些编码法则请参阅： https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

编写函数

```
fun_name <- function(arg1, arg2 = default1, …) {
    #注释
    表达式(循环/判别/……)
    return(返回值)
}
```

- R里边一切都是对象，对象通过赋值来产生，函数也不例外
- 函数声明关键字是function， function返回值就是函数
- 参数列表是以逗号分割，函数主体可以是任何合法的R表达式，通常是一组由大括弧 { 和 } 括起来的表达式
- 若无return语句，最后一个表达式的值作为返回值
- 以fun_name(arg1,arg2,…)的形式调用函数

函数的定义

```
#摄氏度（Celsius）到华氏度（Fahrenheit）的转换
ce2fa <- function(ce) { #参数ce为输入
  fa <- 1.8 * ce + 32 #对输入进行处理
  return(fa) #输出相应的值
}
ce2fa(0)#0℃相当于32℉
#> [1] 32
ce2fa(0:10)#将0~10℃转换为相应的℉
#> [1] 32.0 33.8 35.6 37.4 39.2 41.0 42.8 44.6 46.4 48.2 50.0
```

函数的定义

```
ce2fa

#> function(ce) { #参数ce为输入

#>   fa <- 1.8 * ce + 32 #对输入进行处理

#>   return(fa) #输出相应的值

#> }

#> <bytecode: 0x0000000144b5d28>
```

位置参数和名义参数

```
#位置参数和名义参数
frm <- function(name, frm = "BUPT") {
  cat(name," is frm ", frm)
}


frm()#出错
#> Error in cat(name, " is frm ", frm) :
#>   argument "name" is missing, with no default
frm("axb")#参数的缺省值
#> axb  is frm  BUPT
```

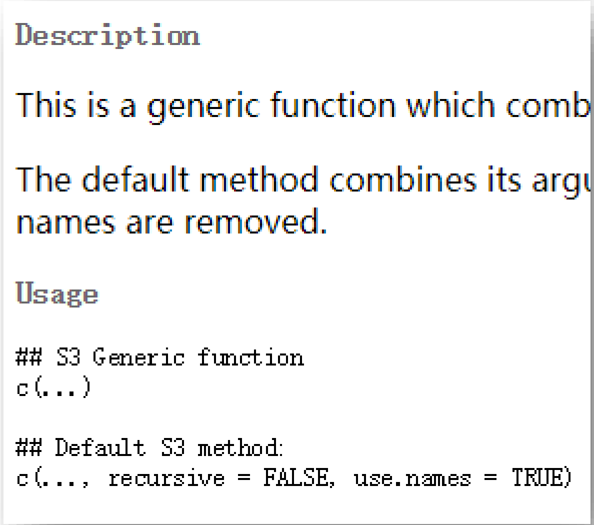
位置参数和名义参数

```
#位置参数和名义参数
frm <- function(name, frm = "BUPT") {
  cat(name," is frm ", frm)
}


frm(name = "AXB", frm = "BJTU")
frm(“AXB”, “BJTU”) #和上述语句等价
#> AXB  is frm  BJTU
frm(frm = "BJTU", name = "AXB")
#> AXB  is frm  BJTU
```

特殊的参数...

```
#看几行我们比较熟悉的代码
xm <- c("周黎", "汤海明", "舒江辉", "翁柯", "祁强", "湛容")
yw <- c(94, 87, 92, 91, 85, 92)
xb <- c(FALSE, TRUE, TRUE)
#再看看sum函数
sum(94, 87, 92, 91, 85, 92)
sum(1, 3, 5, 7)
?c
?sum
```

特殊的参数...



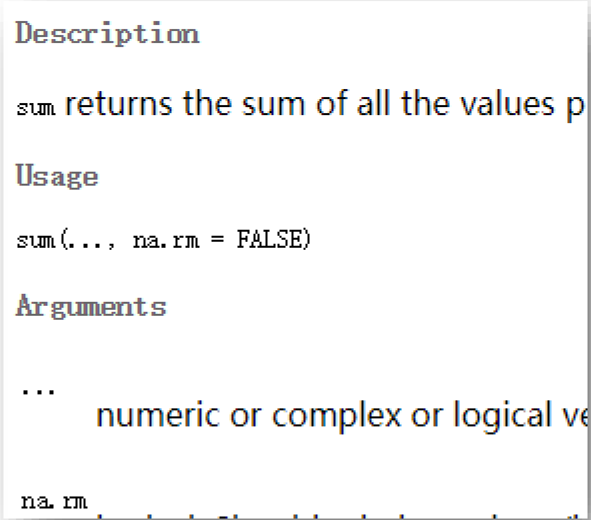
Description

This is a generic function which comb

The default method combines its argu
names are removed.

Usage

```
## S3 Generic function
c(...)

## Default S3 method:
c(..., recursive = FALSE, use.names = TRUE)
```



Description

`sum` returns the sum of all the values p

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

`...`
numeric or complex or logical ve

`na.rm`

... 可以包含任意多的参数。它通常在参数个数未知或者某些参数会传递给其它函数的情况下使用

特殊的参数...

```
my_func <- function(...) {
  cat(“The second arg is ”, ..2) #第二个参数
  dot_args <- list(...) #通过list来捕捉任意参数
  message("\nThe sum is ",sum(dot_args[[1]], dot_args[[5]]))
}
my_func(1, 'arg2', 3, 4, 5, 6, 7, 8)
#> The second arg is  arg2
#> The sum is 6
```

熟而不觉的函数：作为函数的二元操作符

```
#+、-、*、/ 其实都是函数
1 + 2
"+"(1, 2)
#> [1] 3
'/'(2, 3)
#> [1] 0.6666667
'^'(10, 2)
#> [1] 100
“>"(2, 1)
#[1] TRUE
```

```
'<-'(new_var, 3)
new_var
#> [1] 3
#:本质上也是一个函数
':'(1, 10)
#> [1]  1  2  3  4  5  6
7  8  9 10
#下标也是函数
'['(1:10, 2)
#> [1] 2
```

熟而不觉的函数：作为函数的二元操作符

```
#%in%运算符：左侧的每个元素是否在右侧的集合之中
c(1, 3, 9) %in% 1:3
'%in%'(c(1, 3, 9), 1:3)
#> [1]  TRUE  TRUE FALSE

#自己定义二元操作符函数：a、b为直角边，c为斜边
"%ab2c%" <- function(a, b) {
  sqrt(sum(a^2, b^2))
}
3 %ab2c% 4
# [1] 5
```

熟而不觉的函数：作为函数的二元操作符

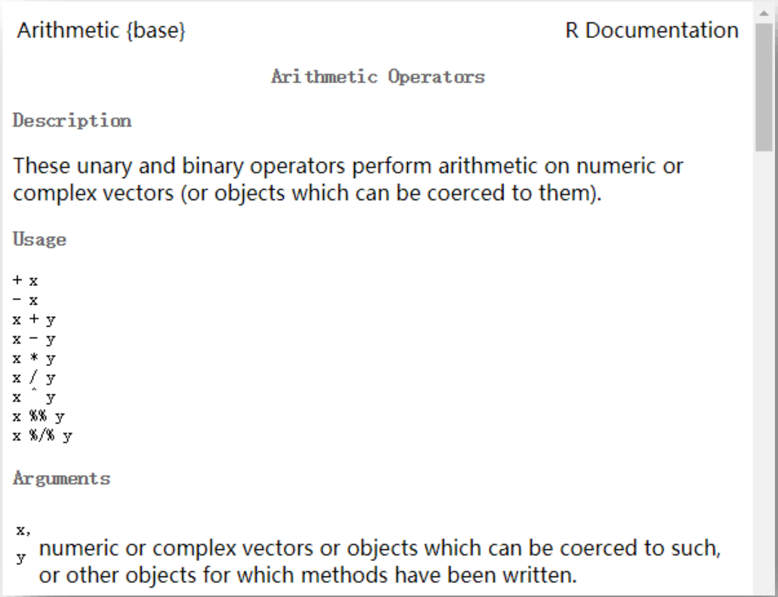
```r
#看完%ab2c%之后，对下边的符号，也就觉得不过如此了
library(purrr)
x <- c(17, 28, 17, 12, 15, 12, 49)
x %>%
  unique() %>%
  sort()
#等价于下边的代码，不过是更加简洁优雅
x <- c(17, 28, 17, 12, 15, 12, 49)
x2 <- unique(x)
x3 <- sort(x2)
x3
#> [1] 12 15 17 28 49
```

熟而不觉的函数：作为函数的二元操作符

```
5 + 2
#> [1] 7
#来点恶作剧
"+" <- function(x, y) {
  x * y
}

5 + 2
#[1] 10

rm(“+”) #消除恶作剧的+运算
5 + 2
#> [1] 7
```

熟而不觉的函数：帮助文档的查询

```
#特殊函数的帮助文档

?"+"  #双引号

?'+'  #单引号

?`+`  #反单引号

?'%in%'

?'round`

?round
```



Arithmetic {base}   R Documentation

Arithmetic Operators

Description

These unary and binary operators perform arithmetic on numeric or complex vectors (or objects which can be coerced to them).

Usage

```
+ x
- x
x + y
x - y
x * y
x / y
x ^ y
x %% y
x %/% y
```

Arguments

x, y numeric or complex vectors or objects which can be coerced to such, or other objects for which methods have been written.

谢谢聆听

Thank you

教师个人联系方式

艾新波

手机：13641159546

QQ：23127789

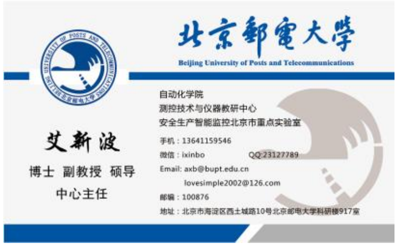
微信：13641159546

E-mail: 13641159546@126.com

axb@bupt.edu.cn

地址：北京邮电大学科研楼917室

课程网址：https://github.com/byaxb/RDataAnalytics



北京郵電大學
Beijing University of Posts and Telecommunications

艾新波
博士 副教授 硕导
中心主任

自动化学院
测控技术与仪器教研中心
安全生产智能监控北京市重点实验室

手机：13641159546
微信：ixinbo QQ:23127789
Email: axb@bupt.edu.cn
lovesimple2002@126.com
邮编：100876
地址：北京市海淀区西土城路10号北京邮电大学科研楼917室