paneldata\_balance

Luis

2015年2月17日

require(dplyr,quietly = T)  
require(lazyeval,quietly = T)  
  
#dplyr包里的主要函数都有加\_的版本，可以接受字符串参数  
#比如下面用的filter\_, group\_by\_  
#这样就可以接受lazyeval变量，在执行时拼出来参数  
#主要可以处理不同面板data.frame里面个体变量id和时间变量t命名不同的问题  
#直接将字符串传递到函数里  
  
p\_balance<-function(df,id="corp",t="year",from=2011,to=2013) {  
 #计算时间变量长度  
 ncount<-to-from+1  
 #如果时间变量是字符的，改成整数  
 df[[t]]<-as.integer(df[[t]])  
 #传递给dplyr::filter\_的参数，方便执行期动态转换为df实际的变量名  
 xt<-list(pt=as.symbol(t),pfrom=from,pto=to)  
 #真正用dplyr执行筛选平衡面板的部分  
 #先排序，然后根据id分组，选出程序参数里定好的时间范围  
 #选出来平衡的数据，然后去掉分组，返回处理好的df，时间变为integer了  
 df<-arrange\_(df,.dots=list(id,t)) %>%  
 group\_by\_(.dots=list(id)) %>%  
 filter\_(interp("pt>=pfrom & pt<=pto",.values = xt)) %>%  
 filter(n()==ncount) %>%  
 ungroup()  
 return(df)  
}

#描述面板数据的类型  
p\_summary<-function(df,id="corp",t="year"){  
 df[[t]]<-as.integer(df[[t]])  
 #输出是否有id和t重复的观测  
 n\_dup<-duplicated(df[,c(id,t)])  
 if(sum(n\_dup,na.rm=T)>0) {  
 print("There is duplicate record,check id and t variable!")  
 print(df[n\_dup,])  
 }  
   
 #用lazyeval来让运行时决定id和t的实际名  
 p\_max<-interp(quote(max(var)), var = as.name(t))  
 p\_min<-interp(quote(min(var)), var = as.name(t))  
 p\_n<-interp(quote(n()))  
 #参数列表  
 dots<-list(p\_min,p\_max,p\_n)  
 #先排序，按照id分组，分别统计最小、最大时间值和每个id观测数量  
 #然后再生成一个组合的字符变量pattern 小-大-长度  
 #再次根据pattern分组，然后统计计算总数  
 p<-arrange\_(df,.dots=list(id,t)) %>%  
 group\_by\_(.dots=list(id)) %>%  
 summarise\_(.dots = setNames(dots, c("min\_t","max\_t","obs\_t"))) %>%  
 mutate(pattern=paste(as.character(min\_t),as.character(max\_t),as.character(obs\_t),sep="-")) %>%  
 group\_by(pattern) %>%  
 summarise(style=n())  
   
  
 #输出时间分布类型  
 print(p)  
 barplot(p$style,names.arg=p$pattern)  
 #return(p)  
}

#Examplar data from stackoverflow  
unbal <- data.frame(PERSON=c(rep('Frank',5),rep('Tony',5),rep('Edward',5)), YEAR=c(2001,2002,2003,2004,2006,2001,2002,2003,2004,2001,2002,2003,2004,2005,2007), Y=c(21,22,23,24,25,5,6,NA,7,8,31,32,33,34,35), X=c(1:15))  
unbal

## PERSON YEAR Y X  
## 1 Frank 2001 21 1  
## 2 Frank 2002 22 2  
## 3 Frank 2003 23 3  
## 4 Frank 2004 24 4  
## 5 Frank 2006 25 5  
## 6 Tony 2001 5 6  
## 7 Tony 2002 6 7  
## 8 Tony 2003 NA 8  
## 9 Tony 2004 7 9  
## 10 Tony 2001 8 10  
## 11 Edward 2002 31 11  
## 12 Edward 2003 32 12  
## 13 Edward 2004 33 13  
## 14 Edward 2005 34 14  
## 15 Edward 2007 35 15

p\_balance(unbal,id="PERSON",t="YEAR",from=2001,to=2005)

## Source: local data frame [5 x 4]  
##   
## PERSON YEAR Y X  
## 1 Tony 2001 5 6  
## 2 Tony 2001 8 10  
## 3 Tony 2002 6 7  
## 4 Tony 2003 NA 8  
## 5 Tony 2004 7 9

p\_summary(unbal,id="PERSON",t="YEAR")

## [1] "There is duplicate record,check id and t variable!"  
## PERSON YEAR Y X  
## 10 Tony 2001 8 10  
## Source: local data frame [3 x 2]  
##   
## pattern style  
## 1 2001-2004-5 1  
## 2 2001-2006-5 1  
## 3 2002-2007-5 1

主要参考内容是nse的vignette，可以用vignette(nse)查看 #Non-standard evaluation

2015-01-13

Dplyr uses non-standard evaluation (NSE) in all of the most important single table verbs: filter(), mutate(), summarise(), arrange(), select() and group\_by(). NSE is important not only to save you typing, but for database backends, is what makes it possible to translate your R code to SQL. However, while NSE is great for interactive use it’s hard to program with. This vignette describes how you can opt out of NSE in dplyr, and instead rely only on SE (along with a little quoting).

Behind the scenes, NSE is powered by the lazyeval package. The goal is to provide an approach to NSE that you can learn once and then apply in many places (dplyr is the first of my packages to use this approach, but over time I will adopt it everywhere). You may want to read the lazyeval vignettes, if you like to learn more about the underlying details, or if you’d like to use this approach in your own packages.

Standard evaluation basics

Every function in dplyr that uses NSE also has a version that uses SE. There’s a consistent naming scheme: the SE is the NSE name with \_ on the end. For example, the SE version of summarise() is summarise\_(), the SE version of arrange() is arrange\_(). These functions work very similarly to their NSE cousins, but the inputs must be “quoted”:

# NSE version:

summarise(mtcars, mean(mpg)) #> mean(mpg) #> 1 20.09062

# SE versions:

summarise\_(mtcars, ~mean(mpg)) #> mean(mpg) #> 1 20.09062 summarise\_(mtcars, quote(mean(mpg))) #> mean(mpg) #> 1 20.09062 summarise\_(mtcars, "mean(mpg)") #> mean(mpg) #> 1 20.09062 There are three ways to quote inputs that dplyr understands:

With a formula, ~ mean(mpg). With quote(), quote(mean(mpg)). As a string: "mean(mpg)". It’s best to use a formula, because a formula captures both the expression to evaluate, and the environment in which it should be a evaluated. This is important if the expression is a mixture of variables in the data frame and objects in the local environment:

constant1 <- function(n) ~n summarise\_(mtcars, constant1(4)) #> n #> 1 4 # Using anything other than a formula will fail because it doesn't # know which environment to look in constant2 <- function(n) quote(n) summarise\_(mtcars, constant2(4)) #> Error in eval(expr, envir, enclos): binding not found: 'n' Setting variable names

If you also want to output variables to vary, you need to pass a list of quoted objects to the .dots argument:

n <- 10 dots <- list(~mean(mpg), ~n) summarise\_(mtcars, .dots = dots) #> mean(mpg) n #> 1 20.09062 10

summarise\_(mtcars, .dots = setNames(dots, c("mean", "count"))) #> mean count #> 1 20.09062 10 Mixing constants and variables

What if you need to mingle constants and variables? Use the handy lazyeval::interp():

library(lazyeval) # Interp works with formulas, quoted calls and strings (but formulas are best) interp(~ x + y, x = 10) #> ~10 + y interp(quote(x + y), x = 10) #> 10 + y interp("x + y", x = 10) #> [1] "10 + y"

# Use as.name if you have a character string that gives a variable name

interp(~ mean(var), var = as.name("mpg")) #> ~mean(mpg) # or supply the quoted name directly interp(~ mean(var), var = quote(mpg)) #> ~mean(mpg) Because every action in R is a function call you can use this same idea to modify functions:

interp(~ f(a, b), f = quote(mean)) #> ~mean(a, b) interp(~ f(a, b), f = as.name("+")) #> ~a + b interp(~ f(a, b), f = quote(if)) #> ~if (a) b If you already have a list of values, use .values:

interp(~ x + y, .values = list(x = 10)) #> ~10 + y

# You can also interpolate variables defined in the current

# environment, but this is a little risky becuase it's easy

# for this to change without you realising

y <- 10 interp(~ x + y, .values = environment()) #> ~x + 10