Advanced r programming course

I'm not robot
reCAPTCHA

Continue

This is a draft course for the Coursera Advanced R Programming class offered by Johns Hopkins University. Repo includes the following files: Part 1: Factor function factorial_code. R: R script is a file that contains four different versions of the feature factor and that calculates a summary of their performance factorial_output.txt: a text file prepared by factorial_code. R, which contains the results of a comparison of four different implementations Part 2: Longitudinal Data Class and oop_code Methods. R: R script is a file containing code implementation classes, methods, and general information for a longitudinal dataset oop_output. R: R script, which sources oop_code. R and produces the console output saved in oop_output.txt. oop_output.txt: A text file containing the console output produced oop_output.R. The console's output was manually added via copy-paste to the file. Page 2 is a draft course for the Coursera Advanced R Programming class offered by Johns Hopkins University. Repo includes the following files: Part 1: Factor function factorial_code. R: R script is a file that contains four different versions of the feature factor and that calculates a summary of their performance factorial_output.txt: a text file prepared by factorial_code. R, which contains the results of a comparison of four different implementations Part 2: Longitudinal Data Class and oop_code Methods. R: R script is a file containing code implementation classes, methods, and general information for a longitudinal dataset oop_output. R: R script, which sources oop_code. R and produces the console output saved in oop_output.txt. oop_output.txt: A text file containing the console output produced oop_output.R. The console's output was manually added via copy-paste to the file. Have at least basic knowledge of R, download the latest versions of R (3.4) and RStudio (1.1), Install a related advr38pkg package with devtools::install_github (privefl/advr38pkg). This will establish other dependencies that are needed in this book. You can't be an expert in R in just one course. However, this course aims to give a broad understanding of many aspects of RV. Some external resources will be mentioned in this book for you to be able to deepen what you would learn in this course. This course originally consists of 9 sessions (3 hours each): R programming and best practices (1.5 sessions) Data Analysis with tidyverse (3.5 sessions) R performance code (2 sessions) R packs (1 session) Bonus: Brilliant (1 session) Course on R debugging and reliable programming by Laurent Gatto and Robert Stnicoj, Data Challenge Lab Hadley Wickham, Advanced R Hadley Wickham, and some R Solutions for Data Science by Garrett Grolemund and Hadley Wickham, and some solutions, R packs Hadley Wickham, Effective R programming Colin Робин Лавлейс (Robin Lovelace), R Programming for Data Science Роджера Д. Пэна (Roger D. Peng), Роджер Д. Пэн (Roger D. Peng), Шон Кросс (Sean Kross) и Брук Андерсон (Brooke Anderson), другие ресурсы, связанные с этими материалами. Страница 2 Хороший стиль кодирования, как правильное пунктуация: вы можете обойтись без него, butitsuremakessingseasiertoread. - Hadley Wickham Пожалуйста, сделайте ваш код читаемым, следуя, например, этому стилю кодирования (примеры ниже приведены в данном руководстве). Вы можете использовать пакет styler (который обеспечивает RStudio addins), чтобы исправить свой стиль: Будьте умны с вашим именования. Я не могу сказать, сколько раз я видел df 0) - журнал (x) - еще сообщение (x отрицательное или нулевое) - еще й й й й й й й й й й <- as.matrix(mtcars)= on= stack= overflow.= put= a= space= before= and= after=when naming= arguments= in= function= calls.= most= infix= operators= (==, +,= -,=></-> <-, etc.) = are= also= surrounded= by= spaces,= except= those= with= relatively= high= precedence:= ^,= :,= ::,= and= :::.= always= put= a= space= after= a= comma,= and= never= before= (just= like= in= regular= english).= #= good= average=></-,> <- mean((feet= 12)= += inches,= na.rm=TRUE) x=></-> <- 1:10= base::sum= #= bad=></-> <-mean(feet 2+inches,na.rm=TRUE) x=></-mean(feet> <- 1= := 10= base= ::= sum= curly= braces,= {},= define= the= most= important= hierarchy= of= r= code.= to= make= this= hierarchy= easy= to= see,= always= indent= the= code= inside= {}= by= two= spaces.= this= should= be= automatic= in= rstudio.= #= good= if= (y=></-> < 0= &= debug)= {= message(y= is= negative)= }= if= (y== 0)= {= if= (x=> Плохо , если < 0= &= debug)= message(y= is= negative)= if= (y== 0)= {= if= (x=> (y 0) - журнал (x) - еще сообщение (x отрицательное или нулевое) . Это удобно помещается на печатной странице с шрифтом разумного размера. Если вы окажетесь работает из комнаты, это хороший признак того, что вы должны инкапсулировать некоторые работы в отдельной функции. Если вызов функции слишком длинный, чтобы поместиться на одной строке, используйте одну строку для имени функции, для каждого аргумента и для закрытия). Это упрощает чтение и изменение кода позже. Хороший do_something_very_complicated (что-то - что, требует много, аргументы - некоторые из которых могут быть длинными) - Bad do_something_very_complicated (что, требует, многие, аргументы, некоторые из которых могут быть длинными Использование <-, not=, for= assignment.= keep=for parameters.= #= good= x=></- ,> <- 5= system.time(= x=></-> <- rnorm(1e6)=)= #= bad= x=5 system.time(= x=rnorm(1e6))= don't= put= ;= at= the= end= of= a= line,= and= avoid= multiple= commands= on= the= same= line.= only= use= return()= for= early= returns.= otherwise= rely= on= r= return= the= result= of= the= last= evaluated= expression.= #= good= add_two=></-> <- function(x,= y)= {= x= += y= }= #= bad= add_two=></-> <- function(x,= y)= {= return(x= += y)= }= use= ,= not= ',= for= quoting= text.= the= only= exception= is= when= the= text= already= contains= double= quotes= and= no= single= quotes.= #= good= text= 'text= with= quotes'=></-> ссылки ' » Плохой 'Текст' 'Текст с двойной и и 'Download the latest version of RStudio' and use it! Learn more about the new RStudio v1.1 features. RStudio Features: Everything You Can Expect From Good IDE Keyboard Shortcuts that I use Ctrl and Space (automatic completion, better than Tab) Ctrl (Command History and Search) Ctrl - Click (Functional Source) Ctrl (Enter (Run Code) Ctrl - Shift (reformatted code) Ctrl - Shift Panels including Git and Terminal) Interactive data import from files and connections (see this webinar) R Projects: A meaningful structure in one folder, the Work Directory automatically switches to a project folder, the Tab File displays related files and folders in the history of Project R teams and open files Any settings associated with the project, such as git settings, are downloaded. R or. Rprofile is in the root catalog of the project so that every time people open a project, you can download project-specific settings. There are only two things that make @JennyBryan 🤴🤢💩. Instead, use the projects here:::: here () #rstats pic.twitter.com/GwxnHePL4n - Hadley Wickham (@hadleywickham) 11 Decembre 2017 More on and see the chapter Effective editing book Effective Programming R. Figure 2.1: Do you not use version control? Have you ever made changes to the code, realized it was a bug, and wanted to go back? Lost code or too old backup? Want to make changes to someone else's code? Want to share your code or let other people work on your code? In these cases, and undoubtedly others, the version management system should make your life easier (see . Version Management for Researcher: Don't do this, use git Version control for data analytics: a reproducible workflow, also, see . Using version management to work from anywhere in the world Working with GitHub can be a line on your resume (more): Many students told me later, even first year students, that using GitHub helped them a lot when they went on an internship or interview research position. They might say, Oh, I've already worked with GitHub. I'm familiar with that. I know how it works. So I think they're at least able to put that on their resume and go into a situation where there's research or data analyzing the group and say, yes, of course. I'm actually familiar with the same tools that you use. - Mine Cetinkaya-Rundel, Duke University, RStudio Main Git platforms (code share, collaborate): 3 main teams: pull: update local project The latest version of the main project commit: a snapshot of the code at a certain point in time push: combine local changes with the main project Simple (solo) use of git to prevent merger conflicts: after opening the project, always pull before closing the project, always commit/push How to link between the RStudio project and the GitHub repository? Can't remember useful features? Use cribs. You can search for specific R things on the . You should also read the documentation carefully. If you're using a package, look for vignettes and a GitHub repository. You can also use the overflow stack. The most common use of Stack overflow is when you have an error or issue, you Google it, and in most cases the first links to the overflow stack. You can ask questions on the overflow stack (using the r tag). You have to make a great R playable example if you want your question to be answered. Sometimes, when you make this reproducible example, you find the answer to your problem. If you're confident enough in your R skills, you can take the next step and answer questions about the overflow stack. This is a good way to improve your skills, or just put off when writing a scientific manuscript. You can also join communities, such as joining the French-speaking R community or joining the R-Ladies community. It tends to be a much friendlier and hospitable space compared to Stack Overflow. Page 3 This chapter is about basic R things that I find important and which are often overlooked or unknown to most R users. Find out more using the Book Advanced R. If you use R and you think you are in hell, this is the map for you. – Patrick Burns #> [1] FALSE print(c(0.1, 0.2, 0.3), digits = 20) #> [1] 0.10000000000000000555 0.20000000000000001110 0.29999999999999998890 all.equal(0.1 + 0.2, 0.3) ## equality with some tolerance #> [1] TRUE isTRUE(all.equal(0.1 + 0.2, 0.4)) ## if you want a boolean, use isTRUE() #> [1] FALSE #> [1] -1 #> [1] 118 #> function (..., na.rm = FALSE) #> NULL #> [1] -1 #> function (x, ...) #> NULL #> [1] -1 #> function (x, na.rm = FALSE, ...) #> NULL Always use one vector if you're not sure: #> [1] -1 #> [1] 118 #> [1] 40.66667 #> [1] 5 #> [1] 9 10 8 4 3 2 6 7 1 5 #> [1] 6 2 9 5 8 7 4 1 3 10 #> [1] 2 4 9 3 1 5 10 8 7 6 #> [1] 1 0 seq_len(0) ## prefer using seq_len (e.g. in for-loops) #> integer(0) n <- 10 1 (n'1 q is) (1:n) q 1, so 2:(n q 1) q1 3 4 5 6 7 8 9 10 11 1 2 3 4 5 6 7 8 9 10 11 qgt; q1 2 3 4 5 6 7 8 9 10 11 There are many atomic data types: logical, more integral, double and symbols (in this order see below). There are also raw and complex, but they are rarely used. You can't types in the atomic vector (you can list). Enforcement will occur automatically if you mix the types: logical zgt; zgt; zgt; zgt; zgt; 1 2 3 4 5 6 7 8 9 10 integer c (a, b) b) 0 > 0 1 2 3 4 5 6 7 8 9 10 > 10,5 > «двойной» (d <- c(b, в)) принудительный к числовому > 1,0 2,0 3,0 4,0 5,0 6,0 7,0 8,0 9,0 10,0 10,5 c(д, а) принудили к персонажу No > 1 2 3 4 5 6 7 8 9 10 10,5 a > No1 «> » 1 »> »> »>» (>) «a» (> c 0s и 1s) «Истинно использовать автоматическое принуждение типа для преобразования этой булеаной матрицы в числовую » (c 0s и 1s). «Что нужно изменить, чтобы получить матрицу с integer вместо этого?» (mat <- matrix(sample(c(TRUE, FALSE), 12, replace = TRUE), 3)) #> [,1] [,2] [,3] [,4] #> [1,] TRUE FALSE TRUE TRUE #> [2,] TRUE FALSE TRUE TRUE #> [3,] FALSE FALSE TRUE TRUE atomic vector: vector of one base type (see above). scalar: this doesn't exist, this is a vector of length 1. matrices / arrays: a vector with some dimensions (attribute). #> [1] 1 2 3 4 5 6 7 8 9 10 11 12 #> [,1] [,2] [,3] [,4] #> [1,] 1 4 7 10 #> [2,] 2 5 8 11 #> [3,] 3 6 9 12 #> [1] matrix dim(vec) <- c(3, 2, 2) vec #> , , 1 #> #> [,1] [,2] #> [1,] 1 4 #> [2,] 2 5 #> [3,] 3 6 #> #> , , 2 #> #> [,1] [,2] #> [1,] 7 10 #> [2,] 8 11 #> [3,] 9 12 #> [1] array list: vector of elements with possibly different types in it. data.frame : список, элементы которого имеют одинаковую длину и отформатированы несколько как матрица. The [accessor is used to access a subset of the data with the same class. #> [1] 0.3742428 0.1967849 0.9906526 0.3354000 0.7317043 #> [1] 0.1967849 0.9906526 #> [1] 0.1967849 0.9906526 0.3354000 0.7317043 NA NA NA (y <- matrix(runif(12), 3)) #> [,1] [,2] [,3] [,4] #> [1,] 0.7640510 0.9464476 0.02809067 0.8516801 #> [2,] 0.5257313

0.4339700 0.10169373 0.5641369 #> [3,] 0.9711716 0.7862585 0.99179505 0.5386078 y[4:9] ## a matrix is a vector #> [1] 0.94644762 0.43397003 0.78625854 0.02809067 0.10169373 0.99179505 (l <- list(a = 2:3, b = toto, c = runif(10))) #> $a #> [1] 2 3 #> #> $b #> [1] toto #> #> $c #> [1] 0.7729944 0.2658974 0.9983454 0.9055118 0.1364265 0.5631251 0.1666813 0.4986730 #> [9] 0.3231522 0.6208681 #> $b #> [1] toto #> #> $c #> [1] 0.7729944 0.2658974 0.9983454 0.9055118 0.1364265 0.5631251 0.1666813 0.4986730 #> [9] 0.3231522 0.6208681 #> Sepal.Length Sepal.Width Petal.Length Petal.Width Species #> 1 5.1 3.5 1.4 0.2 setosa #> 2 4.9 3.0 1.4 0.2 setosa #> 3 4.7 3.2 1.3 0.2 setosa #> 4 4.6 3.1 1.5 0.2 setosa #> 5 5.0 3.6 1.4 0.2 setosa #> 6 5.4 3.9 1.7 0.4 setosa #> Petal.Length Petal.Width #> 1 1.4 0.2 #> 2 1.4 0.2 #> 3 1.3 0.2 #> 4 1.5 0.2 #> 5 1.4 0.2 > 6 1,7 0,4 > «data.frame» Вы также можете использовать логические и символьные векторы для индексации этих объектов. (> 0,4454293 0,4200315 0,4739141 0,5814565 x'c(FALSE, TRUE, FALSE, FALSE, TRUE)» (> 0,4200315 0,5814565 x'c(FALSE, логические векторы перерабатываются «> »1» 0.4200315 0.5814565 головка (iris'c«Petal.length», «Виды») > Petal . Length Виды No > 1 1,4 сетоза No > 2 1,4 сетоза No > 3 1,3 сетоза No > 4 1,5 сетоза No > 5 1,4 сетоза No > 6 1,7 1,7 The [[accessor is used to access a single element. #> [1] 1 2 3 4 5 6 7 8 9 10 #> [1] 3 #> [1] toto #> [1] setosa setosa setosa setosa setosa setosa setosa setosa #> [8] setosa setosa setosa setosa setosa setosa setosa #> [15] setosa setosa setosa setosa setosa setosa setosa #> [22] setosa setosa setosa setosa setosa setosa setosa #> [29] setosa setosa setosa setosa setosa setosa setosa #> [36] setosa setosa setosa setosa setosa setosa setosa #> [43] setosa setosa setosa setosa setosa setosa setosa #> [50] setosa versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [57] versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [64] versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [71] versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [78] versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [85] versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [92] versicolor versicolor versicolor versicolor versicolor versicolor versicolor #> [99] versicolor virginica virginica virginica virginica virginica virginica virginica #> [106] virginica virginica virginica virginica virginica virginica virginica #> [113] virginica virginica virginica virginica virginica virginica virginica #> [120] virginica virginica virginica virginica virginica virginica virginica #> [127] virginica virginica virginica virginica virginica virginica virginica #> [134] virginica virginica virginica virginica virginica virginica virginica #> [141] virginica virginica virginica virginica virginica virginica virginica #> [148] virginica virginica virginica virginica #> Levels: setosa versicolor virginica Figure 3.1: Indexing lists in R. [Source: x <- list(aardvark = 1:5) x$a #> [1] 1 2 3 4 5 #> NULL #> [1] 1 2 3 4 5 Special use of the [accessor for array-like data. #> [,1] [,2] [,3] [,4] #> [1,] 1 4 7 10 #> [2,] 2 5 8 11 #> [3 ,] 3 6 9 12 #> [1] 1 4 7 10 #> [,1] [,2] #> [1,] 1 4 #> [2,] 2 5 #> [3,] 3 6 #> [1] 1 4 mat[1, 1:2, drop = FALSE] #> [,1] [,2] #> [1,] 1 4 #> [1] 1 4 #> [«1», > 1 4 7 10 (>) 2 5 8 11 > 3,» 3 6 9 12 > (> >) 2 5 8 11 (>) 3 6 9 12 (>),> 1 2 2 2 2 2 «> 2», 2 2 2 2 «>» 2 2 2 2 2 просто добавьте запятую для каждого нового измерения. Используйте атрибут измерения, чтобы сделать функцию, которая вычисляет суммы каждый n элементы вектора. В каком порядке хранятся матричные элементы? «Какие особые случаи вы должны рассмотреть?» advr38pkg::sum_every (1:10, 2) > 3 7 11 15 19 Вычислить средства каждого числового столбца набора данных радужной оболочки глаза. Ожидаемый результат: No > Sepal.Length Sepal.Width Petal.Length Petal.Width No > 5.843333 3.057333 1.199333 Преобразование этой матрицы в вектор путем замены c(0,0) -> 0; c(0,1) -> 1; c(1,1) -> 2; c(1,0) -> -> mat <- matrix(0, 10, 2); mat[c(5, 8, 9, 12, 15, 16, 17, 19)] <- 1; mat #> [,1] [,2] #> [1,] 0 0 #> [2,] 0 1 #> [3,] 0 0 #> [4,] 0 0 #> [5,] 1 1 #> [6,] 0 1 #> [7,] 0 1 #> [8,] 1 0 #> [9,] 1 1 #> [10,] 0 0 by using this matrix: (decode <- matrix(c(0, NA, 1, 2), 2)) #> [,1] [,2] #> [1,] 0 1 #> [2,] NA 2 Begin by using apply() and then replace it by a special accessor; what is the advantage? Ожидаемый результат: «> »1» 0 0 0 2 1 1 NA 2 0 В этом разделе, я упоминаю некоторые полезные функции базы R (также см. этот полный список на французском языке и этот на английском языке): » Чтобы получить некоторую помощь ?topic No > Нет документации для «темы» в определенных пакетах и библиотеках: «> вы можете попробовать '?? Тема' - Запустите код из примера раздела пример (сумма) > > сумма> «Передавайте вектор к сумме, и он добавит элементы вместе. Сумма > сумма> (1:5) > 15 > > сумма> « Пройти несколько номеров, чтобы суммировать, и он также добавляет элементы. сумма >> сумма (1, 2, 3, 4, 5) > 15 > > сумма> на самом деле, вы можете передать векторы в несколько аргументов, и все добавляется. > сумма > (1:2, 3:5) > 15 > > сумма> Если отсутствуют значения, сумма неизвестна, т.е.> > > также отсутствует >>. .>... unless we exclude missing values explicitly: #> sum> sum(1:5, NA, na.rm = TRUE) #> [1] 15 # Structure overview str(iris) #> 'data.frame': 150 obs. of 5 variables: #> $ Sepal.Length: num 5.1 4.9 4.7 4.6 5.5.4 4.6 5.4.4 4.9 ... #> $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ... #> $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ... #> $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ... #> $ Species : Factor w/ 3 levels setosa,versicolor,..: 1 1 1 1 1 1 1 1 1 1 ... # List objects of environment a <- 1:4; b <- list() ls() #> [1] a b c d #> [5] decode l mat n #> [9] vec WORDS_TO_IGNORE x y # Remove objects from the environment rm(list = ls()) ## remove all objects in the environment # list of available S3 methods methods(summary) #> [1] summary.aov summary.aovlist* #> [3] summary.aspell* summary.check_packages_in_dir* #> [5] summary.connection summary.data.frame #> [7] summary. Дата summary.default (>) summary.ec > >df. «> резюме.manova summary.matrix (> >) summary.mlm. PDF_Dictionary «>» резюме. PDF_Stream резюме. POSIXct > резюме. POSIXlt резюме.ppr » > (25) summary.prcomp summary.princomp > summary.proc_time summary.srcfile No > (29) summary.srcref summary.stepfun No > > . .tukeysmooth »> см. '?методы' для доступа к справке и исходный код »> »1» добавить1 псевдоним anova case.names принудить »> »6» ограниченный cooks.distance отклонения dfbeta dfbetas »> »11» drop1 dummy.coef extractAIC family 16 formula formula influence initialize kappa #> [21] labels logLik model.frame model.matrix nobs #> [26] plot predict print proj qr #> [31] residuals rstandard rstudent show simulate #> [36] slotsFromS3 summary variable.names vcov #> see '?methods' for accessing help and source code # Call a function with arguments as a list (list_of_int <- lapply(1:5, function(i) i)) #> [[1]] #> [1] 1 #> #> [[2]] #> [1] 2 #> #> [[3]] #> [1] 3 #> #> [[4]] #> [1] 4 #> #> [[5]] #> [1] 5 do.call('c', list_of_int) #> [1] 1 2 3 4 5 #> [1] 1 2 3 4 5 6 7 8 9 10 seq(1, 10, by = 2) ## of type double #> [1] 1 3 5 7 9 seq(1 , 100, length.out = 10) #> [1] 1 12 23 34 45 56 67 78 89 100 #> [1] 1 2 3 4 5 #> [1] 1 2 3 4 #> [1] 1 2 3 4 1 2 3 4 #> [1] 1 1 2 2 3 3 4 4 #> [1] 1 1 1 1 2 2 2 3 3 4 #> [1] 1 2 3 1 2 3 1 2 репликации (5, rnorm(10)) Как использовать многолинейное выражение? #> [,1] [,2] [,3] [,4] [,5] #> [1,] -0.5088130 1.49534876 -1.57672742 1.209146592 -0.2330422 #> [2,] 1.8018798 0.20847523 -1.19125327 1.781712985 1.7505097 #> [3,] 1.7394319 -0.86809087 1.04378069 0.006355165 -1.0317443 #> [4,] -0.5260837 0.89470546 2.22787831 -0.781838677 0.2569092 #> [5,] -1.2306774 -1.36646753 0.85436122 0.357834120 -0.4859491 #> [6,] 0.8516872 1.16793210 -0.04125081 -0.724865082 -1.0674001 #> [7,] 1.8707400 2.18597997 -0.73642035 -0.373385460 -0.1700187 #> [8,] 1.8103187 1.41278136 1.61459563 0.621432210 0.2580942 #> [9,] 0.5264246 0.05163903 1.41141023 0.633884566 0.8802899 #> [10,] -0.6339748 1.97101677 0.67260521 -0.391832393 -0.9342480 #> [1] 1 2 2 6 8 order(c(1, 6, 8, 2, 2), c(0, 0, 0, 2, 1)) #> [1] 1 5 4 2 3 #> [1] 1.0 4.0 5.0 2.5 2.5 rank(c(1, 6, 8, 2, 2), ties.method = first) #> [1] 1 4 5 2 3 sort(c(a1 , a2, a10)) #> [1] a1 a10 a2 gtools ::mixedsort(c(a1, a2, a10)) ## not in base, obviously #> [1] a1 a2 a10 which.max(c(1, 5, 3, 6, 2, 0)) #> [1] 4 which.min(c(1, 5, 3, 6, 2, 0)) #> [1] 6 unique(c(1, NA, 2, 3, 2, NA, 3)) #> [1] 1 NA 2 3 #> #> 1 2 3 4 #> 4 3 2 1 table(A = c(1, 1, 1, 2, 2), B = c(1, 2, 1, 2, 1)) #> B #> A 1 2 #> 1 2 1 #> 2 1 1 #> [1] 10 2 1 9 3 6 5 8 7 4 #> [1] 9 6 10 7 4 sample(3:10, 50 , replace = TRUE) #> [1] 10 9 10 7 9 3 4 6 9 3 8 5 8 4 10 10 9 8 8 3 6 8 7 9 6 7 3 3 #> [29] 4 9 5 6 3 4 4 7 5 3 10 10 5 5 8 8 7 5 3 10 10 4 round(x <- runif(10, max = 100)) #> [1] 86 79 41 61 86 91 81 44 17 95 #> [1] 86.13 79.26 41.10 60.97 86.07 91.39 81.11 43.85 17.21 95.41 #> [1] 90 80 40 60 90 90 80 40 20 100 #> [1] 1 2 2 1 #> [1] 4 3 3 4 #> [,1] [,2] [,3] #> [1,] 2 3 4 #> [2 ,] 3 4 5 #> [3,] 4 5 6 #> [4,] 5 6 7 expand.grid(param1 = c(5, 50), param2 = c(1, 3, 10)) #> param1 param2 #> 1 5 1 #> 2 50 1 #> 3 5 3 #> 4 50 3 #> 5 5 10 #> 6 50 10 Also see this nice Q/A on functions and the *apply family and this book chapter about looping. > «Я – это я» (>) «test0» (PC1) > «PC1», «PC2» «PC3», «PC4», «PC5», «PC6», «PC7», «PC8», «PC9» «PC10» me <- «Florian» клей::«Клей»)» не в базе, но так полезно > что я Флориан <- list.files(pattern=\\. Rmd$,= full.names=TRUE)) #=>(x 1 ./good-practices. Rmd ./index. Rmd ./intro. Rmd > ./пакеты. Rmd ./performance. Rmd ./rprog. Rmd (>) ./shiny. Rmd ./tidyverse. Rmd суб (K. Rmd$, .pdf, x) #> [1] ./good-practices.pdf ./index.pdf ./intro.pdf #> [4] ./packages.pdf ./performance.pdf ./rprog.pdf #> [7] ./shiny.pdf ./tidyverse.pdf (y <- sample(letters[1:4],= 10,= replace=TRUE)) #=>[1] a a a a c b a a a a #> [1] 1 1 1 1 3 2 1 1 1 1 #> [1] TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE #> Run Length Encoding #> lengths: int [1:4] 4 3 2 1 #> values : int [1:4] 1 2 3 4 split(1:12, rep(letters[1:3], 4)) #> $a #> [1] 1 4 7 10 #> #> $b #> [1] 2 5 8 11 #> #> $c #> [1] 3 6 9 12 intersect(letters[1:4], letters[3:5]) #> [1] c d union(letters[1:4] , letters[3:5]) #> [1] a b c d e setdiff(letters[1:4], letters[3:5]) #> [1] a b tryCatch(TRUE | stop(will go there) , error = function(e) print(e)) #> <simpleError in= dotrycatch(return(expr),= name,= parentenv,= handler):= will= go= there=>TRUE || stop(won't go there) ## won't evaluate second condition if first one is TRUE #> [1] TRUE c(TRUE, FALSE, TRUE, TRUE) & c(FALSE, TRUE, TRUE, TRUE) && c(FALSE, TRUE, TRUE, TRUE) ## no warning!! #> [1] FALSE #> [1] 0.36627318 -0.94538659 -0.37411320 -0.49420286 -0.46650356 -0.12747210 0.04273725 #> [8] -0.07248727 -0.20516713 1.65278545 #> [1] 0.36627318 0.94538659 0.37411320 0.49420286 0.46650356 0.12747210 0.04273725 #> [8] 0.07248727 0.20516713 1.65278545 Beware with ifelse() (learn more there), for example #> [1] 1 #> [1] 1 2 3 4 5 Use sample(), rep_len() and split() to make a function that randomly splits some indices in a list of K groups of indices. «Какие особые случаи вы должны рассмотреть?» advr38pkg::split_ind(1:40, 3) #> $'1' #> [1] 1 6 9 18 19 20 24 25 28 31 32 35 36 38 #> #> $'2' #> [1] 7 10 13 14 16 17 21 30 33 34 37 39 40 #> #> $'3' #> [1] 2 3 4 5 8 11 12 15 22 23 26 27 29 Use replicate() and sample() to get a 95% confidence interval for the mean of the following vector: set.seed(1) (x <- rnorm(10))= #=>[1] -0.6264538 0.1836433 -0.8356286 1.5952808 0.3295078 -0.8204684 0.4874291 #> [8] 0.7383247 0.5757814 -0.3053884 #> [1] 0.1322028 #> 2.5% 97.5% #> -0.3150315 0.5994491 Use match() and some accessor to add a column my_val to this data my_mtcars by putting the corresponding value of the column specified in my_col. «Можно ли использовать ваше решение для любого количества имен столбца?» my_mtcars <- mtcars[c(mpg ,= hp)]= my_mtcars$my_col=></-> <- sample(c(mpg,= hp),= size=nrow(my_mtcars), replace=TRUE) head(my_mtcars)= #=>л.с. my_col No > Mazda RX4 21.0 110 на галлон No > Mazda RX4 Wag 21.0 110 л.с. > Datsun 710 22,8 л.с. No > Hornet 4 Drive 21.4 110 миль</-> </-> </simpleError> </-> </-> на галлон > Hornet Hornet 18,7 175 л.с. No > Valiant 18,1 105 миль на галлон Ожидаемый результат (голова): > миль на галлон my_col my_val > Mazda RX4 21.0 110 миль на галлон 21.0 > Mazda RX4 Wag 21.0 110 л.с. 110 > Datsun 71 0 22,8 л 93 л.с. 93 No > Hornet 4 Drive 21.4 110 миль на галлон 21.4 No > Hornet Sportabout 18.7 175 л.с. 175 No > Valiant 18,1 105 миль на галлон 18.1 В следующем кадре данных (напомним, что это также список), для первых 3 3 миль на галлон 18.1 В следующем кадре данных (напомним, что это также список), для первых 3 3 заменить буквы на соответствующие номера на основе кода: df <- data.frame(= id1=c(a, f,= a),= id2=c(b , e,= e),= id3=c(c, d,= f),= inter=c(7.343, 2.454,= 3.234),= stringsasfactors=FALSE)= df= #=>id1 id2 id3 inter > No1 <7> 1 a b c 7.343 No > 2 f e d 2.454 No > 3 a e f 3.234 (код b c d e f > <- setnames(1:6,= letters[1:6]))= #=>1 2 3 4 5 6 Ожидаемый результат: No > id1 id2 id3 inter > No0 <2> 1 1 2 3 7,343 No > 2 6 5 4 2,454 No > 3 1 5 6 3.234 Лексический скоупинг определяет, где искать значения , а не когда их искать. R ищет значения, когда функция запущена, а не когда она создана. Это означает, что выход функции может отличаться в зависимости от объектов за <- function()= {= x=></-> <- 10= g=></-> <- function()= {= f=></-> <- function()= {= x= += 1= }= print(environment(f))= print(ls(environment(f)))= ##= list= the= objects= in= the= environment= print(parent.env(environment(f)))= ##= environment(g)= f()= }= print(environment(g))= print(ls(environment(g)))= print(parent.env(environment(g)))= g()= }= x=></-> <- 100= h()= #=> <environment: 0x62ef478=>пределами своей среды: h 'g> g g x <environment: r_globalenv=>> > > f > > 11 Variable x не определяется внутри f, поэтому R будет смотреть на <environment: 0x67fc630=>среду f <environment: 0x62ef478=>(где f был определен) , а затем на родительную среду, и так далее. Здесь, первый x, который находится имеет значение 10. Имейте в виду, что для функций среды пакетов проверяются последними, так что вы можете переопределить функции, не замечая. c <- function(...) = paste0(...) = c(1,= 2,= 3)= #=>база 123: c(1, 2, 3) вам нужно четко издать пакет > 1 2 3 rm(c) удалить новую функцию из окружающей среды c(1, 2, 3) > 1 2 3 Вы можете использовать <- operator= to= change= the= value= of= an= object= in= an= upper= environment:= count1=></-> <- 0= count2=></-> <- 0= f=></-> <- function(i)= {= count1=></-> <- count1= += 1= ##= will= assign= a= new= (temporary)= count1= count2=></-> <- count2= += 1= ##= will= assign= count2= on= top= i= += 1= }= sapply(1:10,= f)= #=>2 3 4 5 6 7 8 9 10 11 > > В основном, вы копируете и втираете то , что ставится в ...: f1 $a «> » function(...) = {= list(...) = }= f1(a=2, b=3) #=>2 »> »> $b »> » 3 »> $a »> » 2 »> »> $b »> »1» 3 Подробнее о функциях и правилах scoping R с R Программирование для науки о данных книги. Атрибуты — это метаданные, связанные с объектом. Вы можете получить/установить список атрибутов с атрибутами () или один конкретный атрибут с attr (). «> $names «>» «Sepal.Length» «Sepal.Width» «Petal.Length» «Petal.Width» (Виды) »> «> $row.names »> »1» 1</-> </-> </-> </environment:> </environment:> </environment:> </environment:> </-> </-> </-> </-> 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 #> [22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 #> [43] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 #> [64] 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 #> [85] 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 #> [106] 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 #> [127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 #> [148] 148 149 150 #> #> $class #> [1] data.frame #> [1] data.frame #> [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 #> [22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 #> [43] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 #> [64] 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 #> [85] 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 #> [106] 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 #> [127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 #> [148] 148 149 150 You can использовать структуру () для создания объекта и добавления некоторых произвольных атрибутов. структура (1:10, my_fancy_attribute - blabla) No1 > 1 2 3 4 5 6 7 8 9 10 > attr(my_fancy_attribute) > blabla есть также некоторые атрибуты с конкретными функциями доступа, чтобы получить и установить значения. Например, используйте имена (x), dim (x) и class (x) вместо attr (x, имена), attr (x, dim) и attr (x, класс). класс (mylm <- lm(Sepal.Length - ., данные и радужная оболочка)) No > lm Я только что установил линейную модель для того, чтобы предсказать переменную длины сепаральной оболочки глаза набора данных на основе других переменных. Использование lm() получает меня объектом класса lm. Какие методы я могу использовать для этого объекта? methods(class = class(mylm)) #> [1] add1 alias anova case.names coerce #> [6] confint cooks.distance deviance dfbeta dfbetas #> [11] drop1 dummy.coef effects extractAIC family #> [16] formula hatvalues influence initialize kappa #> [21] labels logLik model.frame model.matrix nobs #> [26] plot predict print proj qr #> [31] residuals rstandard rstudent show simulate #> [36] slotsFromS3 summary variable.names vcov #> see '?methods' for accessing help and source code #> #> Call: #> lm(formula = Sepal.Length ~ ., data = iris) #> #> Residuals: #> Min 1Q Median 3Q Max #> -0.79424 -0.21874 0.00899 0.20255 0.73103 #> #> Coefficients: #> Estimate Std. Error t value Pr(>|t|) #> (Intercept) 2.17127 0.27979 7.760 1.43e-12 *** #> Sepal.Width 0.49589 0.08607 5.761 4.87e-08 *** #> Petal.Length 0.82924 0.06853 12.101 < 2e-16 *** #> Petal.Width 0.15120 -2.084 0.03889 * #> Speciesversicolor -0.72356 0.24017 -3.013 0.00306 ** #> Speciesvirginica -1.02350 0.33373 -3.067 0.00258 ** #> --- #> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 No > > Остаточная стандартная погрешность: 0.3068 при 144 градусах свободы > несколько R-квадратов: 0.8673, Скорректированный R-квадрат: 0.8627 > F-статистика: 188.3 на 5 и 144 DF, p-значение: < 2.2e-16= r= has= the= easiest= way= to= create= a= class= and= to= use= methods= on= objects= of= this= class;= it= is= called= s3.= if= you= want= to= know= more= about= the= other= types= of= classes,= see= the= advanced= r= book.= agent007=> <- list(first=James ,= last=Bond)= agent007= #=>$first «> »1» «Джеймс» »> »> $last »> »1» класс «Бонд» (агент007) <- person= ##= agent007= is= now= an= object= of= class= person)= #= just= make= a= function= called=></-> <method_name>.<class_name> () печать. Человек Меня зовут Бонд, Джеймс <- function(x)= {= print(glue::glue(my= name= is= {x$last},= {x$first}= {x$last}.)) = invisible(x)= }= agent007= #=> <- function(first,= last)= {= structure(list(first=first, last=last), class=Person)= }= (me=></-> <- person(florian,= privé)= #=>Бонд. Объект может иметь много классов: <- function(first,= last,= job)= {= obj=></-> <- person(first,= last)= obj$job=></-> <- job= class(obj)=></-> <- c(worker,= class(obj))= obj= }= print.worker=></-> <- function(x)= {= print.person(x)= print(glue::glue(i= am= a= {x$job}.)) = invisible(x)= }= (worker_007=></-> <- worker(james,= bond,= secret= agent))= #=>Рабочий Меня > зовут Бонд, Джеймс Бонд. (worker_me Меня зовут <- worker(florian,= privé,= phd= student))= #=>> Приве, Флориан Приве. Страница 4 Страница 5 Некоторые ресурсы, используемые здесь или для дальнейшего чтения: Расширенный R Эффективный R программирования Люди, которые говорят, что R просто всегда медленно, как правило, не велики R программистов. Это правда, что написание неэффективного R-кода легко, но написание эффективного R-кода также возможно, когда вы знаете, что вы делаете. В этой главе вы узнаете, как написать код R (cpp), который быстро. Смотрите эту новую главу Расширенный R. Пример расчета совокупных сумм вектора: x пользовательская <- rnorm(1e4)= #= try= also= with= n=1e5 system.time({= current_sum=></-> <- 0= res=></-> <- c()= for= (x_i= in= x)= {= current_sum=></-> <- current_sum= += x_i= res=></-> <- c(res,= current_sum)= }= })= #=>система прошла > 0,156 0,000 0,156 Здесь, на каждой итерации, вы перераспределяете вектор (увеличения размера). Распределение памяти требует времени, а также вычислений. Это делает ваш код квадратным с размером x (если умножить размер x на 2, можно ожидать, что время выполнения будет умножено на 4, для больших размеров выборки), в то время как оно должно быть только линейным. Хорошим решением является всегда предварительное распределение результатов (если вы знаете размер): ('current_sum пользовательская система прошла <- 0= res2=></-> <- double(length(x))= for= (i= in= seq_along(x))= {= current_sum=></-> <- current_sum= += x[i]= res2[i]=></-> <- current_sum= }= }) = #=>> 0.004 0.000 0.003 .003 .> 1) Правда Если вы не знаете размер результатов, вы можете хранить их в списке и объединить их потом: system.time (' current_sum <- 0= res3=></-> <- list() list()=></-> list() >/-> </-> </-> </-> </-> </-> </class_name> </method_name> </-> </-> (i в seq_along (x)) current_sum <-current_sum - x'i' res3-i'i <- current_sum) > пользовательская система прошла > 0,008 0,000 0 .005 all.equal (unlist (res3), res) > (1) TRUE С последними версиями R (> 3.4), вы можете эффективно вырастить вектор с помощью system.time (является current_sum < - 0 res4 <- c() для (i в current_sum < seq_along current_sum < current_sum (x)>) система прошла > 0,004 0,000 0,005 > ,1) TRUE Назначение элемента вектора за текущую длину теперь переоделяется на небольшую долю. Новый вектор отмечен внутренне как вырастаемый, а истинная длина нового вектора хранится в истинном поле. Это делает создание векторного результата путем присвоения следующему элементу за пределами текущей длины более эффективным, хотя предварительное распределение по-прежнему предпочтительнее. Реализация может быть изменения и не предназначена для использования в пакетах в настоящее время. (НОВОСТИ) Я НЕ An even better solution would be to avoid the loop by using a vectorized function: system.time(res5 <- cumsum(x)) #> user system elapsed #> 0 0 0 #> [1] TRUE x <- rnorm(1e7) system.time(cumsum(x)) #> user system elapsed #> 0.036 0.012 0.050 As a second example, let us generate a matrix of uniform values (max changing for every column): n <- 1e3 max <- 1:1000 system.time({ mat <- NULL for (m in max) { mat <- cbind(mat, runif(n, max = m)) } }) #> user system elapsed #> 0.560 0.028 0.588 #> [1] 0.9999165 1.9979255 2.9954357 3.9937308 4.9959766 5.9940244 6.9865828 7.9811018 #> [9] 8.9987889 9.9838704 So, we can either pre-allocate a list or a matrix: system.time({ l <- vector(list , length(max)) for (i in seq_along(max)) { l[[i]] <- runif(n, max = max[i]) } mat2 <- do.call(cbind, l) }) #> user system elapsed #> 0.028 0.000 0.028 apply(mat2, 2, max)[1:10] #> [1] 0.997812 1.995219 2.996246 3.998122 4.998680 5.994925 6.984907 7.999305 8.992457 #> [10] 9.971088 system.time({ mat3 <- matrix(0 , n, длина (макс)) для (i in seq_along (max)) - mat3, i' <- runif (n, max - max'i > >) 2, макс) 1:10 » »> »1» 0.9997669 1.9946318 2.9963005 3.9843856 4.9987015 5.9957843 6 .9919149 7.9928761 (>) 8.9948675 9.9859396 Что приятно с помощью списка является то, что вам не нужно предварительно выделить. Действительно, в отличие от атомных векторов, каждый элемент списка находится в разных местах памяти, так что вам не придется перераспределять все данные при добавлении элемента в список. system.time (я <-список() для (i в seq_along (< макс)) макс и макси) - мат4 <- do.call (cbind , l) No > пользовательская система прошла > 0,028 0,000 0,029 применения (mat4, 2, макс) > 0,9997064 1,9988569 2,9924569 2,9924517 3,9925108 4,997340 5.9958494 6.9949641 7.9949342 (>) 8.9962179 9.9982783 Вместо предварительного распределения себя, Вы можете sapply (or lapply and calling do.call () after, as previously done): system.time (mat4 zlt;- sapply (max, (max, runif (n, max q m)) : the user system has passed the 0.032 0.000 0.030 application (mat4, 2, max) qqt; 0.99994166 1.9990283 2.9995770 3.99933 4.9938192 5,998879 7.990 Don't listen to people telling you that sapply () is a vectorized operation that is much faster than loops. When you do calculations on the matrix, remember that the matrix is only a vector with some dimensions. vec zlt; 1:20 dim (vec) c (4, Vec (1) 13 17, No2, 2 6 10 14 18 qqt; 3 7 11 15 19 qqt; 4 8 12 16 20 So, as you can see in this example, the R Matrix is focused on the column, which means that the elements of the same memory column are stored in the same memory column. Thus, access to the elements of the same column is quick. Often, to optimize the code, you can simply find the right function to do what you need to do. For example, using rowMeans (x) instead of applying (x, 1, mean) can save you a lot of time. For example, if you want more efficient features that apply to matrix lines and columns, you can check the matrixStats package. Another example is reading large text files; in such cases, they prefer to use data.table::fread() rather than read.table. Typically, packages that use C/Rcpp are effective. Programmers spend a huge amount of time thinking or worrying about the speed of non-critical parts of their programs, and these efficiency attempts actually have a strong negative impact on debugging and maintenance. Donald Knuth. If you try to optimize every part of your code, you end up losing a lot of time writing it, and it's probably less readable. R is great for prototyping quickly because you can write code in a short and easy way. Start with that. If performance matters, then profile the code to see which part of the code takes too long and optimize only that part! Learn more about how to profile your code in RStudio in this article. See this great blog by Noam Ross to understand vectorization. The integration of Monte Carlo (example from the book Effective R programming) Suppose we want to evaluate integral no (int_0 x 1 x 2 dx) using the Monte Carlo method. Essentially, we throw darts at the curve and calculate the number of darts that fall below the curve (as on the next digit). Monte Carlo Integration Pseudo-Code Initialize: Hits No. 0 for i in 1:N (i) Create two random numbers, (U_1) and (U_2), 0 to 1 (i) If (U_2 qlt; U_1 x2