

# 13. R包开发 II

罗翔宇

中国人民大学统计与大数据研究院

此课件内容基于Hadley Wickham编著的《R包开发》  
(O'Reilly, 杨学辉译)

# 上周复习

- ▶ 创建R包
- ▶ R代码的书写要求 (R/)
- ▶ DESCRIPTION 文件 (DESCRIPTION)
- ▶ 函数帮助文档 (man/)
  - ▶ 利用roxygen2包在R代码上书写函数信息，自动生成对应的.Rd文件
- ▶ 长篇文档 (vignettes/)
  - ▶ 利用R Markdown进行书写，并用Knit进行自动编辑

# 测试

- ▶ 测试是R包开发的重要部分，确保代码做了你想做的事
- ▶ 主要讨论如何进行自动化测试（又叫做单元测试，unit test）
- ▶ 优点为
  - ▶ 更少的错误：你在两个地方描述了你的代码行为—代码中和测试中，它们可以实现交叉验证
  - ▶ 更好的代码结构：编写测试迫使你把复杂的代码分解成单独的功能，各个功能可以独立工作，因此功能将更容易测试、理解和使用
  - ▶ 容易重新启动：测试会更容易从上次结束的地方重新开始
  - ▶ 可靠的代码：如果你知道包中的所有主要功能都有一个相关的测试，就可以做出巨大的改变，不必担心意外破坏了什么

- # 测试
- ▶ 测试是R包开发的重要部分，确保代码做了你想做的事
  - ▶ 主要讨论如何进行自动化测试（又叫做单元测试，unit test）
  - ▶ 优点为
    - ▶ 更少的错误：你在两个地方描述了你的代码行为—代码中和测试中，它们可以实现交叉验证
    - ▶ 更好的代码结构：编写测试迫使你把复杂的代码分解成单独的功能，各个功能可以独立工作，因此功能将更容易测试、理解和使用
    - ▶ 容易重新启动：测试会更容易从上次结束的地方重新开始
    - ▶ 可靠的代码：如果你知道包中的所有主要功能都有一个相关的测试，就可以做出巨大的改变，不必担心意外破坏了什么

# 测试工作流程

- ▶ 打开你的Rproject文件“SASA”
- ▶ 在RStudio中输入下面代码

```
> devtools::test()  
No testing infrastructure found. Create it?  
  
1: Yes  
2: No  
  
selection: Yes
```

- ▶ 这会创建一个tests/testthat目录，并创建一个tests/testthat.R的R文件

Rpackage_development > SASA > tests				
名称	修改日期	类型	大小	
testthat	2020/5/11 14:26	文件夹		
testthat.R	2020/5/11 14:26	R 文件	1 KB	

# 测试工作流程

- ▶ 并且在DESCRIPTION的Suggests域中增加testthat

```
DESCRIPTION
1 Package: SASA
2 Type: Package
3 Title: What the Package Does (Title Case)
4 Version: 0.1.0
5 Author: Who wrote it
6 Maintainer: The package maintainer <yourself@somewhere.net>
7 Description: More about what it does (maybe more than one line)
8     Use four spaces when indenting paragraphs within the Description.
9 License: What license is it under?
10 Encoding: UTF-8
11 LazyData: true
12 RoxygenNote: 7.1.0
13 Suggests:
14     testthat
```

# 测试工作流程

- ▶ 测试文件放在tests/testthat/,测试文件的名字必须以test开始, 下面是一个例子

```
test_stringr.R x
Source on Save
1 library(stringr)
2 context("String length")
3
4 test_that("str_length is number of characters", {
5   expect_equal(str_length("a"), 1)
6   expect_equal(str_length("ab"), 2)
7   expect_equal(str_length("abc"), 3)
8   expect_equal(str_length("study"), 4)
9 })
```

- ▶ 测试是分层组织的: **期望**组成**测试**, **测试**组成**文件**
  - ▶ 期望是测试的原子单位。他描述了一个计算的预期结果: 他是否有正确的值和正确的类? 期望是以expect\_开始的一些函数
  - ▶ 测试是由多个期望组成, 用于测试简单函数的输出、复杂函数的某个参数的变化。也被称为单元
  - ▶ 文件是由多个相关的测试组成, 可利用context()给文件赋予可读的名称

# 测试工作流程

- ▶ 定义好test\_stringr.R后，我们在命令行再次输入devtools::test()则会有如下输出

```
> devtools::test()
Loading SASA
Testing SASA
√ | OK F W S | Context
x | 3 1      | String length
-----
test_stringr.R:8: failure: str_length is number of characters
str_length("study") not equal to 4.
1/1 mismatches
[1] 5 - 4 == 1
-----

== Results ==
OK:      3
Failed:  1
Warnings: 0
Skipped: 0
```

- ▶ 期望中，有3个ok，1个failed。这个failed出现在第8行，原因是“str\_length(“study”) not equal to 4”
- ▶ Context为 “String length”



# 期望

- ▶ 期望是测试的最小粒度，它对函数调用是否实现了期望的结果做一个断言
- ▶ 所有的期望都有类似的结构
  - ▶ 以expect\_开始
  - ▶ 有两个参数：一个是实际结果，另一个是期望值
  - ▶ 实际结果与预期不一致，testthat将抛出一个错误
- ▶ 测试相等的基本方法有两种：expect\_equal()和expect\_identical()。  
expect\_equal()是最常用的，它使用all.equal()来检查一个数值在误差内是否相等

```
> expect_equal(10, 10)
> expect_equal(10, 10 + 0.0001, tolerance = 0.001, scal = 1)
> expect_equal(10, 10 + 0.001, tolerance = 0.001, scal = 1)
> expect_equal(10, 10 + 0.01, tolerance = 0.001, scal = 1)
Error: 10 not equal to 10 + 0.01.
1/1 mismatches
[1] 10 - 10 == -0.01
```

- ▶ tolerance = 0.001, scal = 1, 表示绝对误差在0.001内就视为相等

# 期望

- ▶ 如果测试精确的相等，则使用expect\_identical()

```
> expect_equal(10, 10 + 10^(-7))  
> expect_identical(10, 10 + 10^(-7))  
Error: 10 not identical to 10 + 10^(-7).  
Objects equal but not identical
```

- ▶ expect\_match()检查一个字符向量是否和一个正则表达式匹配

```
> string <- "Testing is fun!"  
> expect_match(string, "Testing")  
> expect_match(string, "testing")  
Error: `string` does not match "testing".  
Actual value: "Testing is fun!"  
> expect_match(string, "testing", ignore.case = TRUE)
```

- ▶ expect\_output()检查打印输出； expect\_message()检查消息；  
expect\_warning()检查警告； expect\_error()检查错误

# 期望

```
> a <- list(1:10, letters)
> a
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[21] "u" "v" "w" "x" "y" "z"

> str(a)
List of 2
 $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
 $ : chr [1:26] "a" "b" "c" "d" ...
> expect_output(str(a), "List of 2")
> expect_output(str(a), "int [1:10]")
Error: `str\`a\` does not match "int [1:10]".
Actual value: "List of 2\\n \\\$ : int \\[1:10\\] 1 2 3 4 5 6 7 8 9 10\\n \\\$ : chr \\[1:26\\]
"a" "b" "c" "d" \\.\\.\\.\"
> expect_output(str(a), "int [1:10]", fixed = TRUE)

> expect_message(library(mgcv), "This is testthat")
Error: `library(mgcv)` produced unexpected messages.
Expected match: This is testthat
Actual values:
* Loading required package: nlme

* This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

# 编写测试

- ▶ 每个测试都有一个有意义的名称，并覆盖一个单一的功能单元
- ▶ 当一个测试失败时，你会知道出了什么问题，知道在代码的什么地方去找问题
- ▶ 编写测试的一些建议
  - ▶ 把重点放在测试功能的外部接口
  - ▶ 努力用仅有的一个测试来测试每个行为
  - ▶ 避免测试你对其有信心的代码。要把时间集中在你不确定的代码、脆弱的代码或具有复杂相互依存关系的代码上
  - ▶ 当你发现一个错误时，总是写一个测试。即总是从编写测试开始，然后写代码通过这些测试

# 编写测试

- ▶ 跳过测试：你可能没有互联网链接或者缺少一个重要的文件，或者写代码时使用的机器越多，越不可能运行所有的测试，则可以使用`skip()`函数

- ▶ 比如

```
check_api <-function(){  
  if(not_working()){  
    skip("API not available")  
  }  
}  
  
test_that("foo api returns bar when given baz",{  
  check_api()  
  ...  
})
```

- ▶ 测试的最高层结构是文件，每个文件应该包含一个单独的`context()`调用来对它的内容进行简要说明。有两种极端情况是不好的：在一个文件中包含所有的测试，每一个测试一个文件。

# 命名空间

- ▶ 包的命名空间：NAMESPACE文件中记录的内容
- ▶ 一个高质量的命名空间有助于封装包，并使其自成一体。这确保其他的包不会干扰你的代码，你的代码也不会干扰其他包，而且不管运行环境怎样，你的包总能够工作
- ▶ 其实我们之前已经使用了命名空间，比如::运算符

```
> devtools::test()
```

- ▶ plyr和Hmisc都提供了summarize()函数
- ▶ 若先加载plyr,再加载Hmisc,那么summarize()是Hmisc的版本
- ▶ 相反，则summarize()是使用了plyr版本
- ▶ 通过Hmisc::summarize()和plyr::summarize()则可显式地引用特定函数，包加载地顺序就无关紧要

# 命名空间

- ▶ 命名空间以两种方式相对独立, Import 和 Export
- ▶ Import定义一个包中的函数如何找到另一个包中的函数

```
> nrow
function (x)
  dim(x)[1L]
<bytecode: 0x000000000db4aa28>
<environment: namespace:base>
> dim <- function(x) c(1, 1)
> dim(matrix(0, 4, 5))
[1] 1 1
> nrow(matrix(0, 4, 5))
[1] 4
```

- ▶ nrow()并没有被破坏。因为当nrow()寻找dim()函数时, 它使用了包的命名空间, 它找到的是base包中的dim(), 而非我们在全局环境中创建的dim()
- ▶ Export指定哪些函数可在包外使用
  - ▶ 一般地, 要导出一组最小的函数集合; 导出的越少, 冲突的几率就越小

# 搜索路径

- ▶ 要调用函数，R首先在全局环境下找，如果找不到它，R就在搜索路径中你已经**附加**的所有包中寻找

```
> search()
```

```
[1] ".GlobalEnv"      "package:quadprog" "package:mgcv"      "package:nlme"  
[5] "devtools_shims"  "package:SASA"      "package:stringr"   "package:testthat"  
[9] "tools:rstudio"   "package:stats"      "package:graphics"  "package:grDevices"  
[13] "package:utils"   "package:datasets"  "package:methods"   "Autoloads"  
[17] "package:base"
```

- ▶ **加载**包和**附加**包的区别：

- ▶ **加载**一个已安装的包将加载代码、数据等，运行.onLoad()函数
- ▶ 加载后，包在内存中，但不在搜索路径中
  - ▶ 两种方式加载包，一个是使用::，需要的时候自动加载包；也可以使用requireNamespace()来显式加载
- ▶ **附加**一个已加载的包会将包放在搜索路径中，使用library()或者require()



# 搜索路径

- ▶ 通过::来利用testthat包中的expect\_equal函数，但是并未附加testthat
- ▶ 可以通过library(testthat)将其附加进来

```
> search()
[1] ".GlobalEnv"      "tools:rstudio"    "package:stats"    "package:graphics"
[5] "package:grDevices" "package:utils"    "package:datasets" "package:methods"
[9] "Autoloads"       "package:base"
> testthat::expect_equal(2, 1+1)
> search()
[1] ".GlobalEnv"      "tools:rstudio"    "package:stats"    "package:graphics"
[5] "package:grDevices" "package:utils"    "package:datasets" "package:methods"
[9] "Autoloads"       "package:base"
> library(testthat)
Warning message:
编辑包‘testthat’是用R版本3.6.3 来建造的
> search()
[1] ".GlobalEnv"      "package:testthat" "tools:rstudio"    "package:stats"
[5] "package:graphics" "package:grDevices" "package:utils"    "package:datasets"
[9] "package:methods" "Autoloads"        "package:base"
> |
```

# 搜索路径

- ▶ 在DESCRIPTION文件中，我们学习到了Depends或Imports中列出一个包会确保它在需要时被安装
- ▶ 主要区别是Imports是加载包，Depends是附加包
- ▶ 除非有很好的理由，否则应该总是使用Imports而不是Depends。这是因为一个好的包是独立的，并最大限度地减少对全局环境（包括搜索路径）的改变。
- ▶ 唯一的例外是如果包被设计为与另一个包一起使用
  - ▶ 例如，analogue包建立在vegan包之上。没有vegan包，analogue包是没有用的，因此analogue把vegan放在Depends里面而不是Imports里

# 命名空间

- ▶ 以下是testthat的NAMESPACE的部分节选

```
# Generated by roxygen2: do not edit by hand

S3method(as.data.frame,testthat_results)
S3method(as.expectation,default)
export(CheckReporter)
export(DebugReporter)
import(rlang)
importFrom(R6,R6Class)
importFrom(magrittr,"%>%")
useDynLib(testthat, .registration = TRUE)
```

- ▶ 每行包含一个指令，指令描述了一个R对象，说明它是从这个包中导出的还是从另一个包中导入的
- ▶ export() 导出函数； exportPattern() 导出所有匹配某个模式的函数；  
exportClasses() exportMethods() 导出S4类和方法； S3method 导出S3方法
- ▶ import() 从包中导入所有函数； importFrom() 导入选定的函数；  
importClassesFrom()和importMethodsFrom() 导入S4类和方法； useDynLib()  
从C中导入一个函数

# 命名空间

- ▶ 不建议手写这些指令，而是利用roxygen2包生成NAMESPACE文件
- ▶ 主要有以下优势
- ▶ 1. 命名空间定义在关联函数旁边，当读到代码时，容易看到什么导入或导出
- ▶ 2. roxygen2抽象了NAMESPACE的一些细节，只需要学习一个标签@export，它将会为函数、S3方法等自动生成正确的指令
- ▶ 3. roxygen2使NAMESPACE更整洁。在代码中无论使用多少次@importFrom foo bar, 在NAMESPACE中只能得到一个importFrom(foo, bar)

# 导出

- ▶ 要使一个函数可在包外使用，必须 **导出** 它
- ▶ 当创建一个新包时，产生的NAMESPACE将导出包中所有不以.开头的东西
- ▶ 最好只导出需要的函数

```
#' @export  
foo <- function(x, y, z){  
  ...  
}
```

- ▶ 这将根据对象的类型产生export(), exportMethods(), exportClass()或S3method()

# 导入

- ▶ NAMESPACE通过导入控制哪些外部函数可以被包调用而无需使用::
- ▶ NAMESPACE中提到的每个包必须在DESCRIPTION文件中的Imports域或Depends域中出现
- ▶ 如果使用外面包的函数次数不多:在DESCRIPTION文件的Imports域列出包, 然后显式使用 pkg::fun()
- ▶ 如果反复使用函数, 在NAMESPACE中, 用@importFrom pkg fun 来避免::
  - ▶ 若使用另一个包的许多函数, 则使用@import pkg 导入包中的所有函数

# 外部数据

## ► 导出的数据

- 包中数据最常用的位置是data/, 此目录中的每个文件应该是利用save()创建的.RData文件
- 如果DESCRIPTION中包含LazyData:true, 将延迟加载数据集。在你使用它们之前, 它们不会占用任何内存
- data/ 中的对象总是被有效导出, 必须有相应的文档

```
#' Prices of 50,000 round cut diamonds
#'
#' A dataset containing the prices and other attributes of almost 54,000 diamonds
#'
#' @format A dataframe with 53940 rows and variables:
#' \describe{
#'   \item{price}{price, in US dollars}
#'   \item{carat}{weight of the diamond, in carats}
#' }
#' @source \url{http://www.diamondse.info/}
"diamonds"
```

## ► 内部数据

- R/sysdata.rda
- 其中对象不会被导出

## ► 原始数据

- inst/extdata

# 其他知识

## ► 编译过的代码

- R是一门高级的、富有表现力的语言，但这是以速度为代价的
- 需要结合低级的编译语言(比如C或者C++)，它们的速度可以比R快上几个数量级
- 这些文件放入src/目录下
- Rcpp使得在R中连接C++很容易 (<http://adv-r.had.co.nz/Rcpp.html>)
- 安装文件：当一个包被安装后，inst/目录下的所有内容都会被复制到包的顶层目录
  - inst/AUTHOR , inst/COPYRIGHT
  - inst/CITATION
  - inst/java, inst/python



# 其他知识

## ► 引用其他包

```
> citation()
```

To cite R in publications use:

```
R Core Team (2019). R: A language and environment for statistical  
computing. R Foundation for Statistical Computing, Vienna, Austria. URL  
https://www.R-project.org/.
```

A BibTeX entry for LaTeX users is

```
@Manual{,  
  title = {R: A Language and Environment for Statistical Computing},  
  author = {{R Core Team}},  
  organization = {R Foundation for Statistical Computing},  
  address = {Vienna, Austria},  
  year = {2019},  
  url = {https://www.R-project.org/},  
}
```

We have invested a lot of time and effort in creating R, please cite it when using it for data analysis. See also 'citation("pkgname")' for citing R packages.

```
> citation("devtools")
```

在出版物中使用程序包时引用‘devtools’:

```
Hadley Wickham, Jim Hester and Winston Chang (2020). devtools: Tools to  
Make Developing R Packages Easier. R package version 2.3.0.  
https://CRAN.R-project.org/package=devtools
```

A BibTeX entry for LaTeX users is

```
@Manual{,  
  title = {devtools: Tools to Make Developing R Packages Easier},  
  author = {Hadley Wickham and Jim Hester and Winston Chang},  
  year = {2020},  
  note = {R package version 2.3.0},  
  url = {https://CRAN.R-project.org/package=devtools},  
}
```

# 其他知识

## ► inst/CITATION 文件如何书写

```
citHeader("To cite the testthat package in publications, use:")

citEntry(
  entry = "Article",
  author = personList(as.person("Hadley Wickham")),
  title = "testthat: Get Started with Testing",
  journal = "The R Journal",
  year = 2011,
  volume = 3,
  pages = "5--10",
  url = "https://journal.r-project.org/archive/2011-1/RJournal\_2011-1\_Wickham.pdf",
  textVersion = paste(
    "Hadley Wickham. testthat: Get Started with Testing.",
    "The R Journal, vol. 3, no. 1, pp. 5--10, 2011"
  )
)
```

# 其他知识

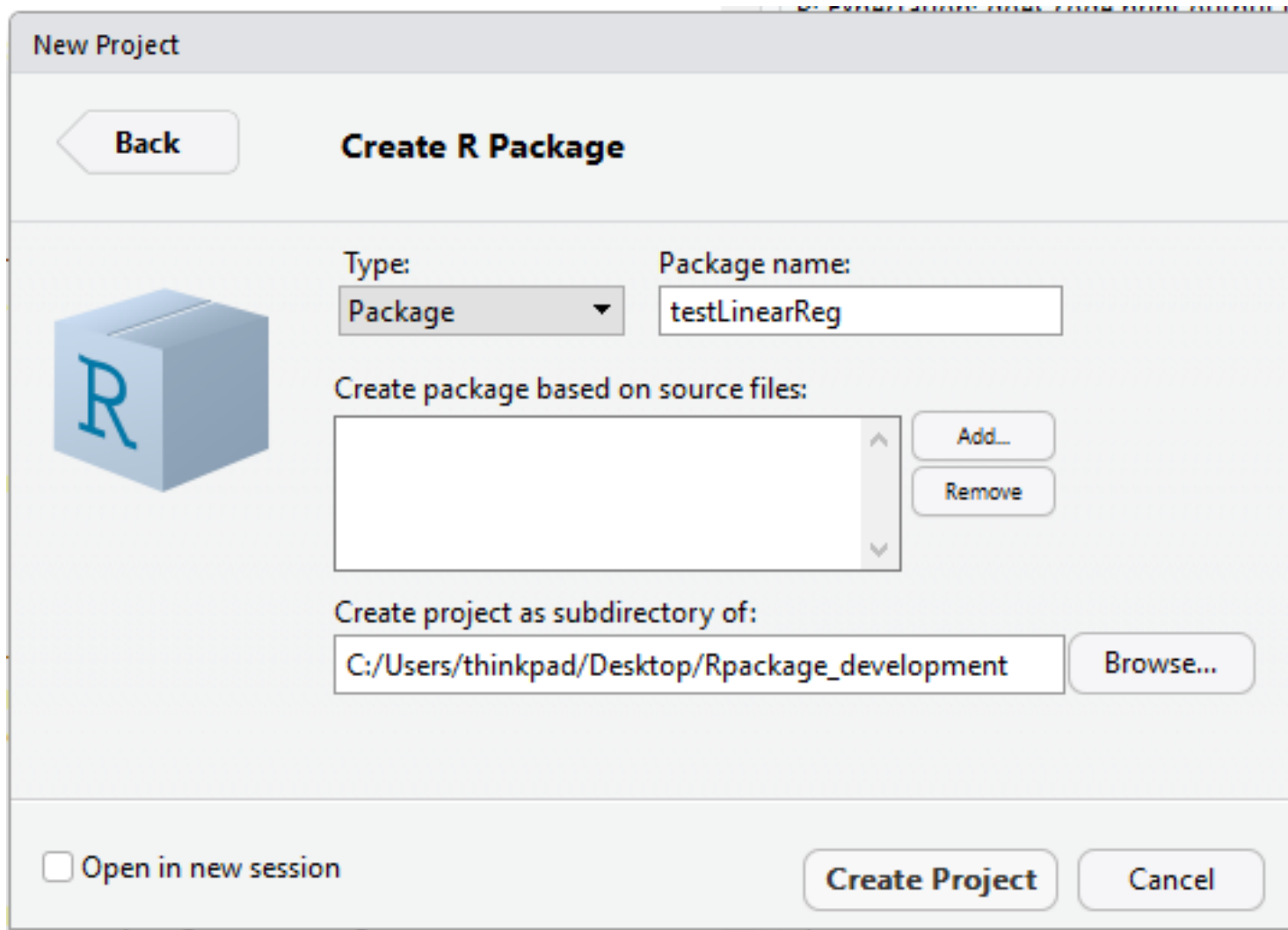
- ▶ 还有四个目录是有效的顶层目录，但是较少使用
- ▶ demo/ 包的演示目录
- ▶ exec/ 可执行脚本
- ▶ po/ 翻译信息
- ▶ tools/ 在配置过程中需要的辅助文件，或者是用来生成脚本的源代码

# 一个例子

- ▶ 制作一个R包名字为testLinearReg，满足下列要求
  - ▶ 主要函数为LinReg，输入数据响应向量 $y$ ，协变量矩阵 $X$ ，以及新观察的 $x\_new$ ，返回线性回归方程  $y = X\beta + \epsilon$  中预测值 $x\_new\beta$
  - ▶ 利用roxygen2标注出这些输入、输出信息
  - ▶ 添加data/
  - ▶ 完善DESCRIPTION，NAMESPACE中的信息
  - ▶ 建立、完善test/
  - ▶ 安装并加载你的R包，利用data/中的数据，运行你的R包中的函数LinReg

# 一个例子

- 1. 建立一个Rproject文件，名字为testLinearReg



# 一个例子

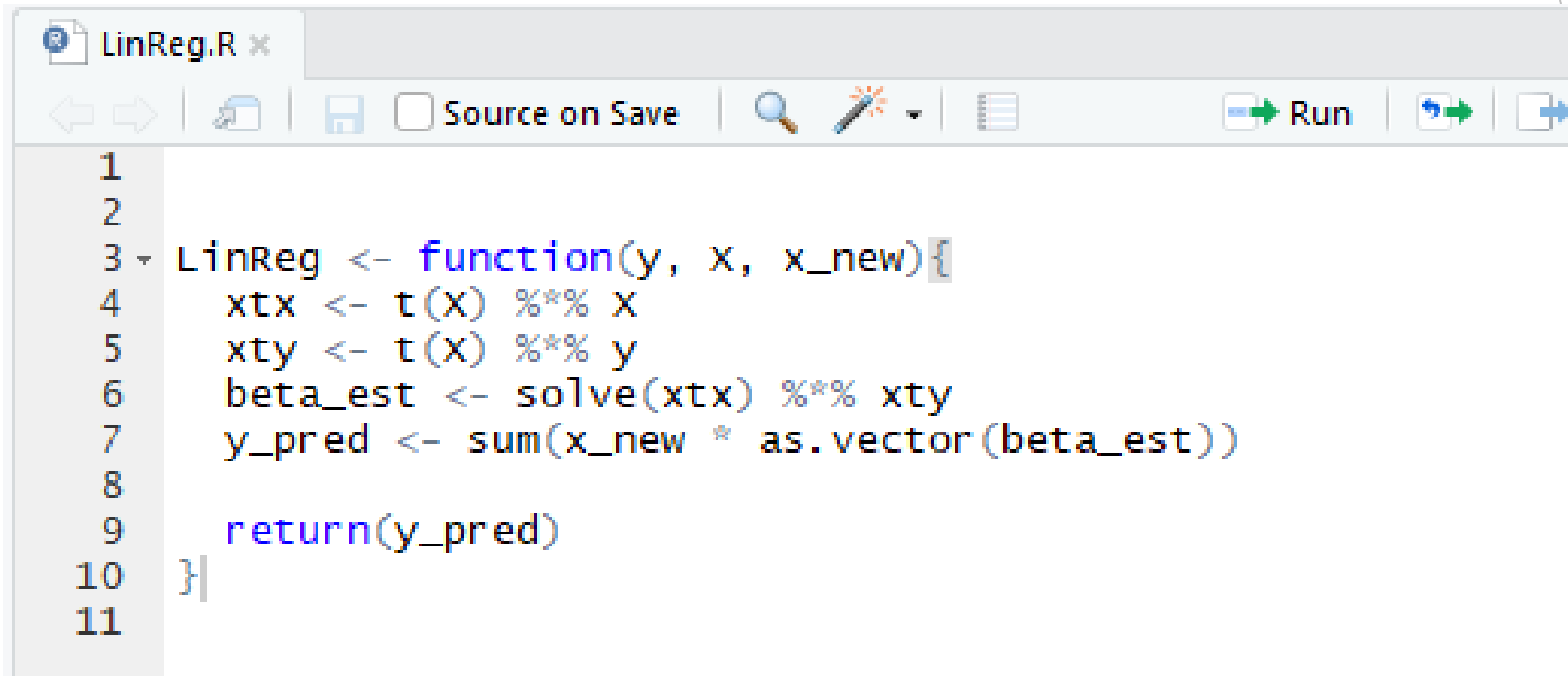
- ▶ 2. 默认的弹出的R文件为hello.R,将其另存为LinReg.R,并删除hello.R

此电脑 > 桌面 > Rpackage\_development > testLinearReg > R

名称	修改日期	类型	大小
hello.R	2020/5/12 15:28	R 文件	1 KB
LinReg.R	2020/5/12 15:29	R 文件	1 KB

# 一个例子

## ► 3. 在LinReg.R文件中写出对应的函数

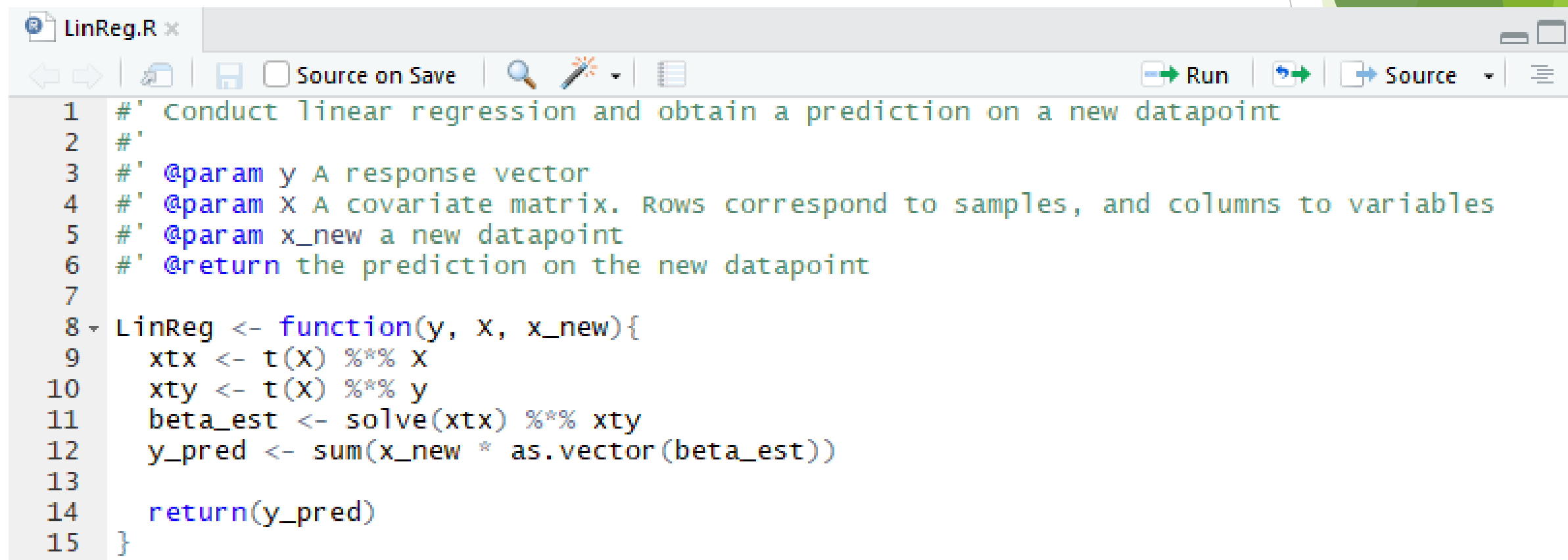


The image shows a screenshot of an R script editor window titled "LinReg.R". The window has a toolbar with icons for navigation, saving, and running. The script content is as follows:

```
1  
2  
3 LinReg <- function(y, X, x_new){  
4   xtx <- t(X) %*% X  
5   xty <- t(X) %*% y  
6   beta_est <- solve(xtx) %*% xty  
7   y_pred <- sum(x_new * as.vector(beta_est))  
8  
9   return(y_pred)  
10 }  
11
```

# 一个例子

## ► 4. 在LinReg.R文件中添加roxygen注释



```
LinReg.R x
Source on Save
Run
Source

1 #' Conduct linear regression and obtain a prediction on a new datapoint
2 #'
3 #' @param y A response vector
4 #' @param X A covariate matrix. Rows correspond to samples, and columns to variables
5 #' @param x_new a new datapoint
6 #' @return the prediction on the new datapoint
7
8 LinReg <- function(y, X, x_new){
9   xtx <- t(X) %*% X
10  xty <- t(X) %*% y
11  beta_est <- solve(xtx) %*% xty
12  y_pred <- sum(x_new * as.vector(beta_est))
13
14  return(y_pred)
15 }
```



# 一个例子

## ► 5. 运行devtools::document()生成对应的帮助文档

```
> devtools::document()  
Updating testLinearReg documentation  
First time using roxygen2. Upgrading automatically...  
Loading testLinearReg  
Warning: The existing 'NAMESPACE' file was not generated by r  
tten.  
Writing LinReg.Rd
```

## ► 在man/目录下删去不要的hello.Rd,就剩下LinReg.Rd

电脑 > 桌面 > Rpackage\_development > testLinearReg > man


名称	修改日期	类型	大小
LinReg.Rd	2020/5/12 16:11	RD 文件	1 KB

# 一个例子

## ► 6. 添加example data

```
x <- matrix(rnorm(20), 10 ,2)
y <- x %*% c(1, 2)
save(list = c("x", "y"), file = "LinRegData.RData")
```

电脑 > 桌面 > Rpackage\_development > testLinearReg > data

名称	修改日期	类型	大小
 LinRegData	2020/5/12 16:04	R Workspace	1 KB

# 一个例子

## ► 7. 完善DESCRIPTION文件

```
DESCRIPTION
1 Package: testLinearReg
2 Type: Package
3 Title: Conduct a simple linear regression and obtain a prediction
4 Version: 0.5.13
5 Author: Mike
6 Maintainer: Mike <mmmmikeeee@ruc.edu.cn>
7 Description: First estimate the regression coefficients and then obtain the value of the estimated
8             linear function at the new datapoint.
9 License: GPL-3
10 Encoding: UTF-8
11 LazyData: true
12 RoxygenNote: 7.1.0
13 Depends: R (>= 3.5.0)
14 Suggests:
15     testthat
16
```

# 一个例子

- 8. 建立test文件夹，对函数进行简单测试，是否符合预期

```
> devtools::test()  
No testing infrastructure found. Create it?
```

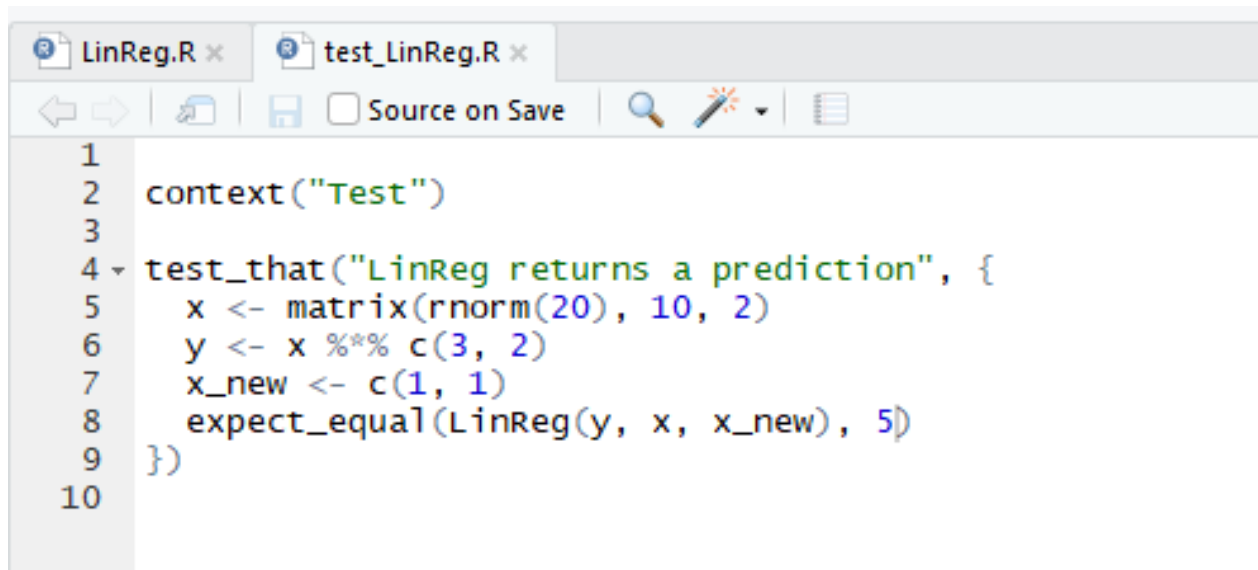
```
1: Yes
```

```
2: No
```

```
selection: yes
```

```
Enter an item from the menu, or 0 to exit
```

```
selection: 1
```



```
LinReg.R x test_LinReg.R x  
Source on Save  
1  
2 context("Test")  
3  
4 test_that("LinReg returns a prediction", {  
5   x <- matrix(rnorm(20), 10, 2)  
6   y <- x %*% c(3, 2)  
7   x_new <- c(1, 1)  
8   expect_equal(LinReg(y, x, x_new), 5)  
9 })  
10
```

# 一个例子

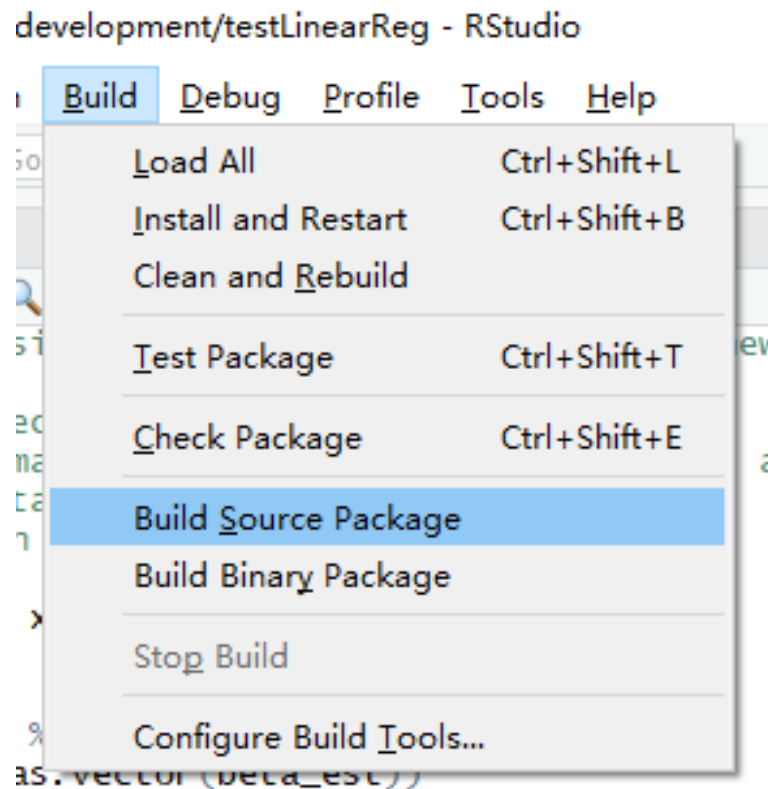
- ▶ 8. 建立test文件夹，对函数进行简单测试，是否符合预期

```
> devtools::test()
Loading testLinearReg
Testing testLinearReg
✓ | OK F W S | Context
✓ | 1      | Test
```

```
== Results =====
OK:      1
Failed:   0
Warnings: 0
Skipped:  0
Warning message:
package 'testthat' was built under R version 3.6.3
.
```

# 一个例子

## ► 9. 建立所对应的R包



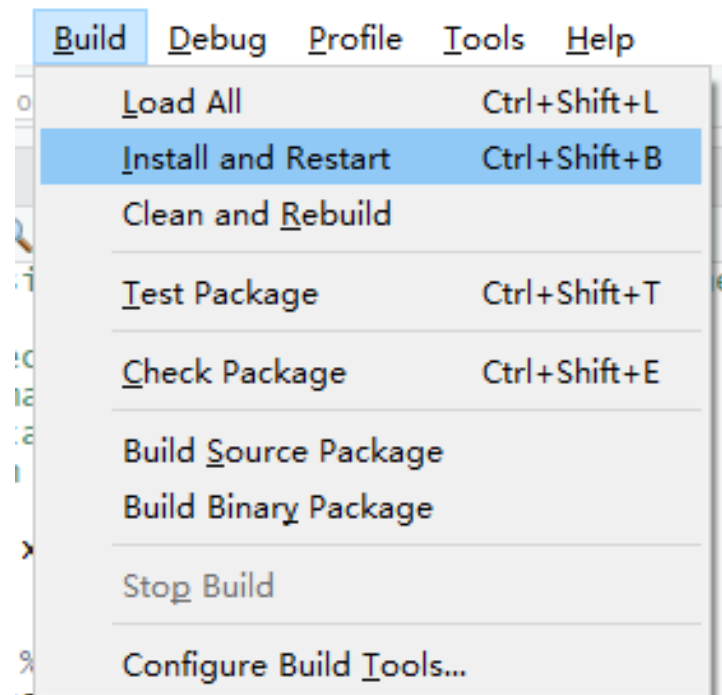
电脑 > 桌面 > Rpackage\_development

名称	修改日期	类型	大小
 SASA	2020/5/12 10:08	文件夹	
 testLinearReg	2020/5/12 16:23	文件夹	
 testLinearReg_0.5.13.tar	2020/5/12 16:33	好压 GZ 压缩文件	2 KB

# 一个例子

## ► 10. 加载R包，并利用包中的数据执行一个例子

development/testLinearReg - RStudio



```
> library(testLinearReg)
```

载入程辑包: 'testLinearReg'

The following object is masked \_by\_ '.GlobalEnv':

**LinReg**

```
> data(LinRegData)
```

```
> LinReg(y, x, c(1, 1))
```

```
[1] 3
```

```
> LinReg(y, x, c(1, 3))
```

```
[1] 7
```

# 一个例子

## ► 10. 还可以通过?LinReg查看帮助文档

LinReg {testLinearReg}

R Documentation

Conduct linear regression and obtain a prediction on a new datapoint

### Description

Conduct linear regression and obtain a prediction on a new datapoint

### Usage

```
LinReg(y, X, x_new)
```

### Arguments

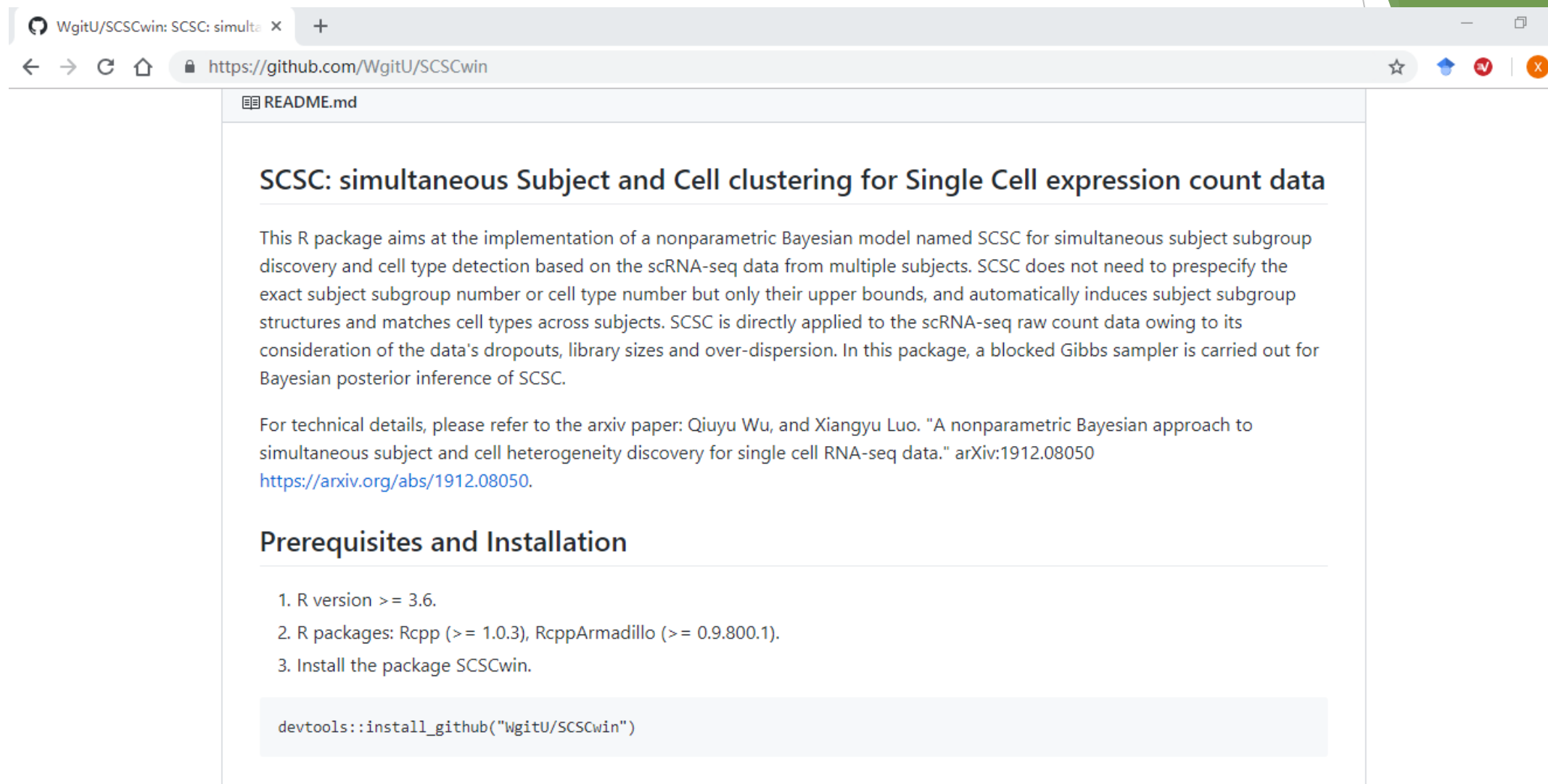
<code>y</code>	A response vector
<code>X</code>	A covariate matrix. Rows correspond to samples, and columns to variables
<code>x_new</code>	a new datapoint

### Value

the prediction on the new datapoint



# 如何从GitHub上下载包?



WgitU/SCSCwin: SCSC: simulta x +

https://github.com/WgitU/SCSCwin

README.md

## SCSC: simultaneous Subject and Cell clustering for Single Cell expression count data

This R package aims at the implementation of a nonparametric Bayesian model named SCSC for simultaneous subject subgroup discovery and cell type detection based on the scRNA-seq data from multiple subjects. SCSC does not need to prespecify the exact subject subgroup number or cell type number but only their upper bounds, and automatically induces subject subgroup structures and matches cell types across subjects. SCSC is directly applied to the scRNA-seq raw count data owing to its consideration of the data's dropouts, library sizes and over-dispersion. In this package, a blocked Gibbs sampler is carried out for Bayesian posterior inference of SCSC.

For technical details, please refer to the arxiv paper: Qiuyu Wu, and Xiangyu Luo. "A nonparametric Bayesian approach to simultaneous subject and cell heterogeneity discovery for single cell RNA-seq data." arXiv:1912.08050  
<https://arxiv.org/abs/1912.08050>.

## Prerequisites and Installation

1. R version  $\geq 3.6$ .
2. R packages: Rcpp ( $\geq 1.0.3$ ), RcppArmadillo ( $\geq 0.9.800.1$ ).
3. Install the package SCSCwin.

```
devtools::install_github("WgitU/SCSCwin")
```

# 如何从GitHub上下载包?

通过 `devtools::install_github( )` 进行安装

```
> devtools::install_github("wgitu/SCSCwin")
Downloading GitHub repo wgitu/SCSCwin@master
These packages have more recent versions available.
It is recommended to update all of them.
which would you like to update?

1: All
2: CRAN packages only
3: None
4: Rcpp          (1.0.3      -> 1.0.4.6    ) [CRAN]
5: RcppArmad... (0.9.800.3.0 -> 0.9.870.2.0) [CRAN]

Enter one or more numbers, or an empty line to skip updates:

✓ checking for file 'C:\Users\thinkpad\AppData\Local\Temp\RtmpyU5GGx\
U-SCSCwin-a1a3022\DESCRIPTION' ...
- preparing 'SCSCwin':
```