L1: R Basic Introduction

Nan He, School of Basic Medical Sciences, Southern Medical University

2026-01-18

目录

R Basic Introduction

1. 引言

这一小节主要介绍如何更高效使用 R studio 这个编辑器来进行 R 的分析。
主要包括三种常见的 R 包下载读取方式，以及文件的写入写出，还有 R 不同源文件的使用方法。

2. R 包下载、读取和卸载

R 包下载

2.1 CRAN 下载 CRAN 是大多数 R 包寄存的网站。通过 CRAN 安装是最基础的安装方式，适用于绝大多数通用的统计绘图包（如 `ggplot2`, `dplyr`）。安装方式如下：

```
## 安装 ggplot2 包
install.packages("ggplot2")

## 进阶：指定镜像（国内推荐清华或中科大源，速度快）
install.packages("ggplot2", repos = "[https://mirrors.tuna.tsinghua.edu.cn/CRAN/](https://mirrors.
```

2.2 Bioconductor 下载 Bioconductor 生态是 R 语言关于生物信息学分析的最大的生态，绝大多数生信相关的 R 包都会放在上面（https://www.bioconductor.org/）。

安装方式与 CRAN 稍有不同，需要先装一个 `BiocManager` 的包，再用这个包来安装 Bioconductor 上的包。相当于 `BiocManager` 是一个中介：

```
## 先安装 BiocManager
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

## 再用 BiocManager 安装你需要的包
BiocManager::install("limma")

## 同时也支持安装特定版本的包
BiocManager::install("limma", version = "3.18")
```

2.3 Github 安装 很多新工具或开发中的包托管在 GitHub，尚未发布到 CRAN。你需要先安装 `devtools` 或 `remotes` 包。`remotes` 更轻量级，`devtools` 功能更全。

```
# 这里的 "username/repo" 对应 GitHub 网址末尾的部分
# 例如: [https://github.com/tyRa/TCMDATA](https://github.com/tyRa/TCMDATA)

# 方法 A: 使用 devtools
library(devtools)
install_github("tyRa/TCMDATA")

# 方法 B: 使用 remotes
# install.packages("remotes")
library(remotes)
install_github("tyRa/TCMDATA")
```

2.4 本地源码安装 有时候因为网络问题，比如有一些包放在 GitHub 上，没办法直接顺利安装，可以去 GitHub 仓库或者 CRAN 里下载它的源代码文件，通常是`.tar.gz` / `.zip` 格式的文件。然后再到本地安装.

比如这里以一个中介分析的 R 包 `regmeint` 为例，下载到本地是`.zip` 格式的文件。这样安装也需要用到 `devtools` 这个包：

```
library(devtools)
install_local("regmedint-master.zip")
```

R 包读取

R 包加载 一般读取就用 `library()`。但有一个函数叫做 `require()`，如果有一个包你没有下载，然后你 require 了，他不会像 library 那样报错，而是报一个 warning，然后返回 FALSE。`warning` 在 R 里是不

会终止运行的，一般只有 error 才需要管。

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.5.2
```

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## require 一个没有下载的包
require(abcd)
```

```
## Loading required package: abcd
```

```
## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'abcd'
```

R 包移除加载 当你加载了太多包，发现函数名冲突了（比如 dplyr 和 MASS 都有 select 函数），或者想让 R 的环境干净一点，但又不想卸载这个包，只是想让它暂时“退场”。

```
# 注意格式：必须写成 "package: 包名"
# unload=TRUE 表示彻底从内存中移除
detach("package:ggplot2", unload = TRUE)
```

detach 之后，如果再调用 ggplot2 相关的函数，则会报错，说明 detach 成功。

R 包卸载

如果包彻底坏了需要重装，或者再也不用了，此时就需要卸载。

```
# 这会从你的硬盘上彻底删除该包的文件
remove.packages("ggplot2")

# 提示：如果卸载失败，通常是因为该包正在被使用。
# 请先重启 R (Session -> Restart R)，然后再运行卸载命令。
```

扩展：双冒号 :: 与三冒号 :::

在阅读别人的代码时，你经常会看到 `dplyr::filter` 这种写法。这是一种**不加载包**就能直接用函数的方法。

双冒号 pkg::function 调用该包对外**公开**的函数。使用这个有两个理由，首先是为了避免同名函数的命名冲突，比如 `stats` 包和 `dplyr` 包都有 `filter` 函数。如果你直接用 `filter()`，R 不知道你用的是哪个包的 `filter()`。此时可以显式指定：

```
# 明确指定使用 dplyr 包的 filter 函数
# 即使没有 library(dplyr)，这行代码也能跑
dplyr::filter(mtcars, cyl == 6)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

第二个是，省内存或者写起来简便。如果只需用一次某包的函数，没必要 `library()` 整个包。

三冒号 pkg:::function 强行调用该包的内部/隐藏的函数。包的开发者在写包时，会把一些常用的函数公开给用户。但还有很多辅助函数是写给开发者自己用的，不想让用户看到（觉得用户用不上，或者容易报错）。如果你非要用这些“私房”函数，就得用:::。

```
stats:::acf
```

```
## function (x, lag.max = NULL, type = c("correlation", "covariance",
##     "partial"), plot = TRUE, na.action = na.fail, demean = TRUE,
##     ...)
## {
##     type <- match.arg(type)
##     if (type == "partial") {
##         m <- match.call()
##         m[[1L]] <- quote(stats::pacf)
##         m$type <- NULL
##         return(eval(m, parent.frame()))
##     }
##     series <- deparse1(substitute(x))
##     x <- na.action(as.ts(x))
##     x.freq <- frequency(x)
```

```
##     x <- as.matrix(x)
##     if (!is.numeric(x))
##         stop("'x' must be numeric")
##     sampleT <- as.integer(nrow(x))
##     nser <- as.integer(ncol(x))
##     if (is.na(sampleT) || is.na(nser))
##         stop("'sampleT' and 'nser' must be integer")
##     if (is.null(lag.max))
##         lag.max <- floor(10 * (log10(sampleT) - log10(nser)))
##     lag.max <- as.integer(min(lag.max, sampleT - 1L))
##     if (is.na(lag.max) || lag.max < 0)
##         stop("'lag.max' must be at least 0")
##     if (demean)
##         x <- sweep(x, 2, colMeans(x, na.rm = TRUE), check.margin = FALSE)
##     lag <- matrix(1, nser, nser)
##     lag[lower.tri(lag)] <- -1
##     acf <- .Call(C_acf, x, lag.max, type == "correlation")
##     lag <- outer(0:lag.max, lag/x.freq)
##     acf.out <- structure(list(acf = acf, type = type, n.used = sampleT,
##         lag = lag, series = series, snames = colnames(x)), class = "acf")
##     if (plot) {
##         plot.acf(acf.out, ...)
##         invisible(acf.out)
##     }
##     else acf.out
## }
## <bytecode: 0x10a570040>
## <environment: namespace:stats>
```

或者可以用 getFromNamespace()：

```
f <- getFromNamespace("add1", "stats")
f
```

```
## function (object, scope, ...)
## UseMethod("add1")
## <bytecode: 0x12dd5ca68>
## <environment: namespace:stats>
```

3. 文件的写入写出

预先生成一个示例数据，方便后续演示

```r
test_data <- data.frame(
  GeneID = c("TP53", "EGFR", "KRAS", "BRCA1"),
  LogFC = c(2.5, -1.2, 0.5, 3.1),
  PValue = c(0.001, 0.05, 0.8, 0.0001),
  Group = c("Up", "Down", "Stable", "Up"))

test_data
```

```
##   GeneID LogFC PValue  Group
## 1   TP53   2.5  1e-03     Up
## 2   EGFR  -1.2  5e-02   Down
## 3   KRAS   0.5  8e-01 Stable
## 4  BRCA1   3.1  1e-04     Up
```

3.1 CSV 文件

CSV (逗号分隔值) 是生信分析中最常用的交换格式，体积小，任何软件都能打开。

```r
# 重点：row.names = FALSE
# 如果不加这个，R 会自动把行号（1,2,3...）写成第一列，导致数据错位
write.csv(test_data, file = "gene_results.csv", row.names = FALSE)
```

写入 CSV

```r
df_base <- read.csv("gene_results.csv")

# 或者用 readr 的 read_csv 函数，更快
library(readr)
df_tidy <- read_csv("gene_results.csv")
```

读取 CSV

3.2 Excel 文件

虽然分析时尽量不用 Excel（容易被 Excel 自动改格式），但经常需要读取协作者发来的数据。

读写 Excel　我一般使用 openxlsx 来进行 excel 文件的读写：

```r
# install.packages("openxlsx")
library(openxlsx)
```

```r
## 读 xlsx
read.xlsx("gene_results.xlsx")

## 写 xlsx
write.xlsx(test_data, path = "gene_results.xlsx")
```

或者还可以用 readxl 包：

```r
# install.packages("readxl")
library(readxl)

# 读取第一个 Sheet
df_excel <- read_xlsx("gene_results.xlsx")

# 读取指定 Sheet (通过名字或索引)
# df_excel_2 <- read_xlsx("gene_results.xlsx", sheet = "Sheet1")
```

3.3 TXT 文件

很多生信原始数据（如 GEO 下载的矩阵）是 .txt 或 .tsv 格式。

```r

# sep = "\t" 表示用制表符分隔 (即 TSV 格式)
# quote = FALSE 表示不给字符加双引号 (看起来更干净)
write.table(test_data, file = "gene_results.txt",
            sep = "\t", row.names = FALSE, quote = FALSE)
```

写入 TXT

```r

# header = TRUE 表示第一行是列名
# sep = "\t" 必须与写入时一致
df_txt <- read.table("gene_results.txt", header = TRUE, sep = "\t")
```

读取 TXT

3.4 RDS 文件

这是 R 的“私有格式”。它可以把任何 R 对象（不仅仅是表格，还可以是列表、模型结果、Seurat 对象等）完整地“冻结”保存下来。

优点：

1. 保留属性：CSV 会把 Factor 变成 Character，RDS 则原样保存。

2. **体积小**：自带压缩。

3. **速度快**：读写速度远超 CSV。

```
# 保存任何对象
saveRDS(test_data, file = "gene_data.rds")
```

保存 RDS

读取 RDS **注意**：readRDS 需要赋值给一个变量，它不会自动载入原来的变量名。

```
my_restored_data <- readRDS("gene_data.rds")
```

4. R 文件

R 中最常见的文件类型就是.R 和.rmd，前者是脚本，后者是一个交互式的笔记本。

对于我自己而言，通常会用.rmd 的文件来进行作业汇报、流程展示等。而.R 用来写函数，然后 source()，更多用于本地开发和代码调试。

但是，比如在服务器上，这个时候一般就用.R 来写脚本，然后用直接终端运行：

```
Rscript xxx.R
nohup Rscript xxx.R &
```