R 语言编程：基于 tidyverse

第 01 讲　前言

张敬信

2022 年 2 月 10 日

哈尔滨商业大学

一. R 简介

Python 和 R 是数据科学领域最受欢迎的编程语言：

- Python 更全能，适合将来做程序员或在工业企业工作
- R 更侧重数据统计分析，适合将来做科研学术

R 语言是专业的统计编程语言，具有顶尖水准的绘图功能：

- R 语言是统计学家开发，为统计计算、数据分析和可视化而设计
- R 语言适合做数据处理和数据建模（数据预处理、数据探索性分析、识别数据隐含的模式、数据可视化）。

- 2016-2019 年, KDnuggets 数据科学领域最受欢迎编程语言调研：Python 和 R 位于前 3 名
- TIOBE 指数逐月排名：近年一般在 10 名左右徘徊
- IEEE Spectrum 2021 年度编程语言排名：第 7 名

R 语言的优势：

- 免费开源，软件体积小根据需要安装扩展包，兼容各种常见操作系统，有强大活跃的社区
- 专门为统计和数据分析开发的语言，有丰富的扩展包
- 拥有顶尖水准的制图功能
- 面向对象和函数，比 Python 简单易学

在热门的机器学习领域：

- 有足以媲美 Python 的 sklearn 机器学习库的 R 机器学习包：`mlr3verse` 或 `tidymodels`.

ggplot2 曾经是 R 语言的一张名片，受到广泛的赞誉；从与时俱进的角度来说，tidyverse 应该成为如今 R 语言的一张名片！

近年来，R 语言在国外蓬勃发展，ggplot2 这个 “点” 在 2016 年以来，已被 Hadley 大神 “连成线、张成面、形成体（系）”，这就是 tidyverse 包，集

数据导入—数据清洗— 数据操作— 数据可视化—数据建模—可重现与交互报告

整个数据科学流程于一身，而且是以 “现代的”、“优雅的” 方式，以管道式、泛函式编程技术实现。不夸张地说，tidyverse 操作数据比 pandas 更加好用、易用！再加上可视化本来就是 R 所擅长，可以说 R 在数据科学领域已强于 Python。

- Tidyverse 包是专为数据科学而开发的一系列包的合集，基于整洁数据，提供了一致的底层设计哲学、一致的语法、一致的数据结构。
- Tidyverse 用 “现代的”、“优雅的” 方式，以管道式、泛函式编程技术实现了数据科学的整个流程：数据导入、数据清洗、数据操作、数据可视化、数据建模、可重现与交互报告。
- Tidyverse 操作数据的优雅，就体现在：
    - 每一步要 “做什么”，就写 “做什么”，用管道依次做下去，得到最终结果
    - 代码读起来，就像是在读文字叙述一样，顺畅自然，毫无滞涩

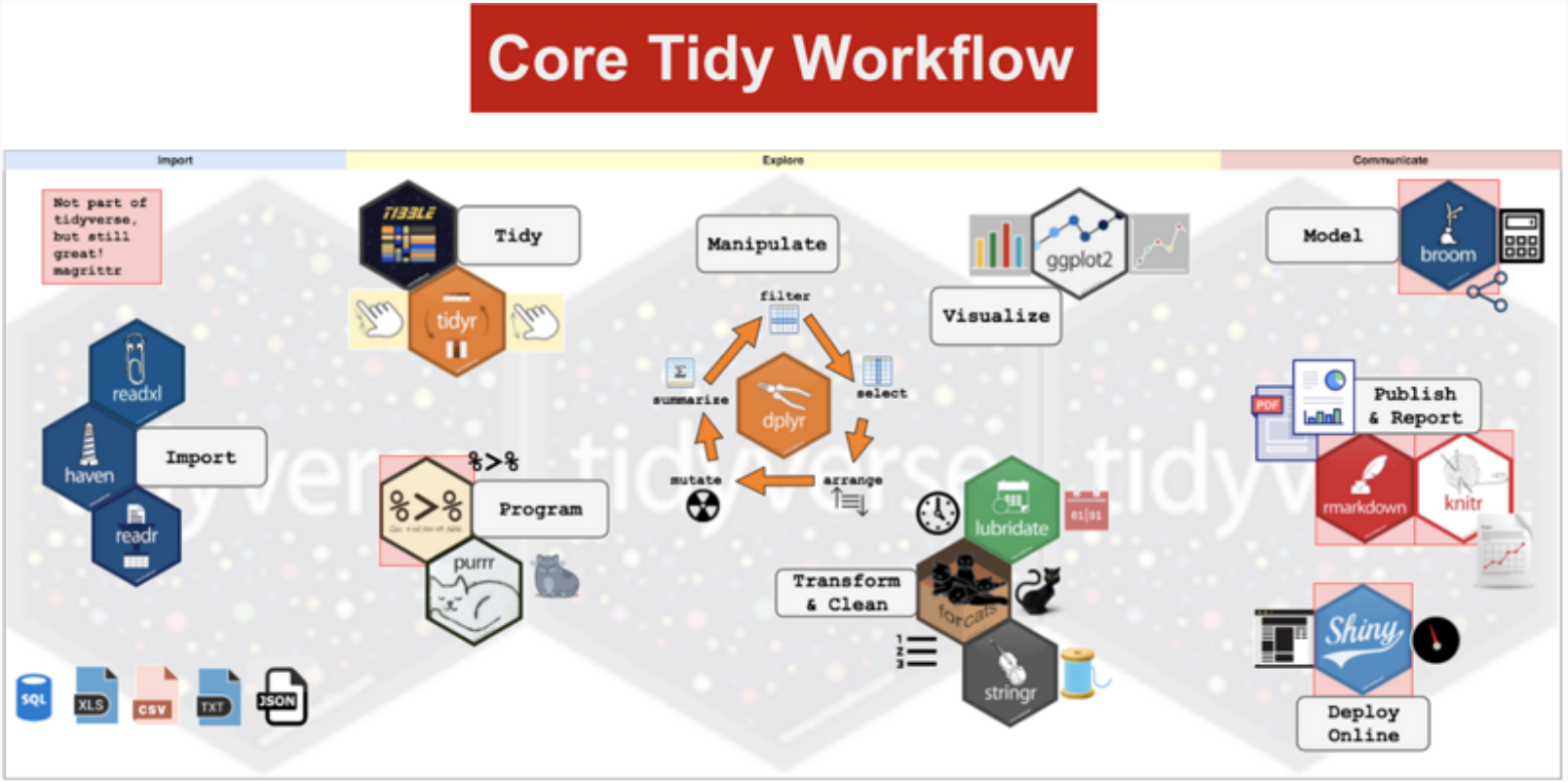


图 1: Tidyverse 核心工作流

这种整洁、优雅的 tidy-流，又带动了 R 语言在很多研究领域涌现出了一系列 tidy-风格的包：**tidymodels**（统计与机器学习）、**mlr3verse**（机器学习）、**rstatix**（应用统计）、**tidybayes**（贝叶斯模型）、**tidyquant**（金融）、**fpp3**（时间序列）、**quanteda**（文本挖掘）、**tidygraph**（网络图）、**sf**（空间数据分析）、**tidybulk**（生信）、**sparklyr**（大数据）等。

其中机器学习/数据挖掘领域，曾经的 R 靠单打独斗的包，如今也正在从整合技术上迎头赶上 Python，出现了 tidy-风格的 **tidymodels** 包，以及真正最新理念、最新技术、最新一代的机器学习 **mlr3verse** 包，它比 sklearn 还先进，基于 R6 类面向对象，**data.table** 神速数据底层，开创性的 Graph-流模式（图/网络流，区别于通常的线性流）。

附：Hadley 大神

图 2: R 语言超级大神：Hadley

二. 怎么学习编程语言?

- 编程语言是人与计算机沟通的一种形式语言，根据设计好的编程元素和语法规则，来严格规范地表达我们想要做的事情的每一步（程序代码），使得计算机能够明白并正确执行，得到期望的结果。
- 错误的编程："**学编程，就是照着别人的代码敲代码**"—事倍功半，关键是学不会真正的编程!
- 正确的编程：先学习并掌握**编程元素**和**语法规则**，比如数据结构（容器）、分支/循环结构、自定义函数等，然后遇到具体问题，分解问题、借助实例梳理，根据掌握的编程元素和语法规则翻译成代码并调试通过，从而自己写出代码解决问题。

学习任何一门编程语言，根据我的经验，有这么几点建议（步骤）：

（1）理解该编程语言的核心思想，比如 R 语言是面向函数也面向对象，另外，高级编程语言还都倡导向量化编程。在核心思想的引领下去学它去思考去写代码。

（2）学习该编程语言的基础知识（包括数据类型及数据结构（容器）、分支/循环结构、自定义函数、文件读写、可视化等），这些基础知识本质上是相通的同样的东西，只是在不同编程语言下披上了其特有的外衣（编程语法）。

（3）前两步完成之后，就算基本入门[1]了，可以根据需要，根据遇到的问题，借助网络搜索、借助帮助，遇到问题解决问题，逐步提升，用的越多会的越多，也越熟练。

[1]至少要经历过一种编程语言的入门，再学习其他编程语言就会很快。

以上是学习编程语言的正确、快速、有效的方法，切忌不学基础语法，用到哪就突击哪，找别人的代码一顿不知其所以然的瞎改，这样的结果是：**自以为节省时间，实际上是浪费了几十倍的时间**，关键是始终无法入门，更谈不上将来提高。

如何跨越“能看懂别人的代码”到“自己写代码”的鸿沟?

为什么大家普遍自己写代码解决具体问题时感觉无从下手呢?

这是因为你总想一步就从**问题**到**代码**，没有中间的过程，即使编程高手也做不到。

正确的做法是：**分解问题** + **实例梳理** + **翻译及调试**

具体如下:

- 将难以入手大问题分解为可以逐步解决的小问题
- 用计算机的思维去思考解决每步小问题
- 借助类比的简单实例和代码片段，梳理出详细算法步骤
- 将详细算法步骤用逐片段地用编程语法翻译成代码并调试通过

可以说高级编程语言的程序代码就是逐片段调试出来的，借助简单实例按照算法步骤，从上一步的结果调试得到下一步的结果，依次向前推进直到到达最终的结果。

经验之谈：**写代码时，随时跟踪关注每一步执行，变量、数据的值是否到达你所期望的值，非常有必要！**

案例：计算并绘制 ROC 曲线

- ROC 曲线是在不同分类阈值上对比真正率（TPR）与假正率（FPR）的曲线。
- 分类阈值就是根据预测概率判定预测类别的阈值，要让该阈值从 0 到 1 以足够小的步长变化，对于每个阈值 c，比如 0.85，则预测概率 ≥ 0.85 判定为 “Pos”，< 0.85 判定为 “Neg”. 这样就得到了预测类别。
- 根据真实类别和预测类别，就能计算混淆矩阵：

		真实类别（y）	
		+	-
预测类别（$\hat{y}$）	+	TP（真正）	FP（假正）
	-	FN（假负）	TN（真负）

图 3: 混淆矩阵示意图

- 进一步就可以计算：

$$TPR = \frac{TP}{TP + FN}, \qquad FPR = \frac{FP}{FP + TN}$$

- 有一个阈值，就能计算一组 TPR 和 FPR，循环迭代都计算出来并保存。再以 FPR 为 x 轴，以 TPR 为 y 轴绘图，则得到 ROC 曲线。

于是，梳理一下经过分解后的问题：

(1) 让分类阈值以某步长在 $[1, 0]$ 上变化取值；

(2) 对某一个阈值，

- 计算预测类别
- 计算混淆矩阵
- 计算 TPR 和 FPR

(3) 循环迭代，计算所有阈值的 TPR 和 FPR

(4) 根据 TPR 和 FPR 数据绘图

- 拿一个小数据算例，借助代码片段来推演上述过程

```
library(tidyverse)
df = tibble(
  ID = 1:10,
  真实类别 = c("Pos","Pos","Pos","Neg","Pos",
               "Neg","Neg","Neg","Pos","Neg"),
  预测概率 = c(0.95,0.86,0.69,0.65,0.59,0.52,
               0.39,0.28,0.15,0.06))
```

```
knitr::kable(df)
```

ID	真实类别	预测概率
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Neg	0.39
8	Neg	0.28
9	Pos	0.15
10	Neg	0.06

先来解决对某一个阈值，计算 TPR 和 FPR。以 $c = 0.85$ 为例。

计算预测类别，实际上就是 If-else 语句根据条件赋值，当然是用整洁的 tidyverse 来做。顺便多做一件事情：把类别变量转化为因子型，以保证 “Pos” 和 “Neg” 的正确顺序，与混淆矩阵中一致。

```
c = 0.85
df1 = df %>%
  mutate(
    预测类别 = ifelse(预测概率 >= c, "Pos", "Neg"),
    预测类别 = factor(预测类别, levels = c("Pos", "Neg")),
    真实类别 = factor(真实类别, levels = c("Pos", "Neg")))
```

```
knitr::kable(df1)
```

ID	真实类别	预测概率	预测类别
1	Pos	0.95	Pos
2	Pos	0.86	Pos
3	Pos	0.69	Neg
4	Neg	0.65	Neg
5	Pos	0.59	Neg
6	Neg	0.52	Neg
7	Neg	0.39	Neg
8	Neg	0.28	Neg
9	Pos	0.15	Neg
10	Neg	0.06	Neg

计算混淆矩阵，实际上就是统计交叉频数，本来为 “Pos” 预测为 “Pos” 的有多少，等等：

```
cm = table(df1$预测类别, df1$真实类别)
cm
#>
#>          Pos Neg
#>    Pos     2   0
#>    Neg     3   5
```

计算 TPR 和 FPR。根据其计算公式，从混淆矩阵中取数计算即可。这里用更高级的向量化计算来实现。

向量化编程，关键是要用整体考量的思维来思考、来表示运算。比如这里计算 TPR 和 FPR，通过观察可以发现：混淆矩阵的第 1 行各元素，都除以其所在列和，正好是 TPR 和 FPR。

```
cm["Pos",] / colSums(cm)
#>  Pos  Neg
#> 0.4  0.0
```

这就完成了本问题的核心部分。接下来，要循环迭代对每个阈值，都计算一遍 TPR 和 FPR。用 `for` 循环当然可以，但咱们仍然更高级一点：泛函式编程。

先把上述计算封装为一个 **自定义函数**，该函数只要接受一个前文原始的数据框 `df` 和一个阈值 `c`，就能返回来你想要的 TPR 和 FPR。然后，再把该函数 **应用** 到数据框 `df` 和一系列的阈值上，循环迭代自然就完成了。这就是 **泛函式编程**。

```r
cal_ROC = function(df, c) {
  df = df %>%
    mutate(
      预测类别 = ifelse(预测概率 >= c, "Pos", "Neg"),
      预测类别 = factor(预测类别, levels = c("Pos", "Neg")),
      真实类别 = factor(真实类别, levels = c("Pos", "Neg")))
    cm = table(df$预测类别, df$真实类别)
    t = cm["Pos",] / colSums(cm)
    list(TPR = t[[1]], FPR = t[[2]])
}
```

测试一下这个自定义函数：

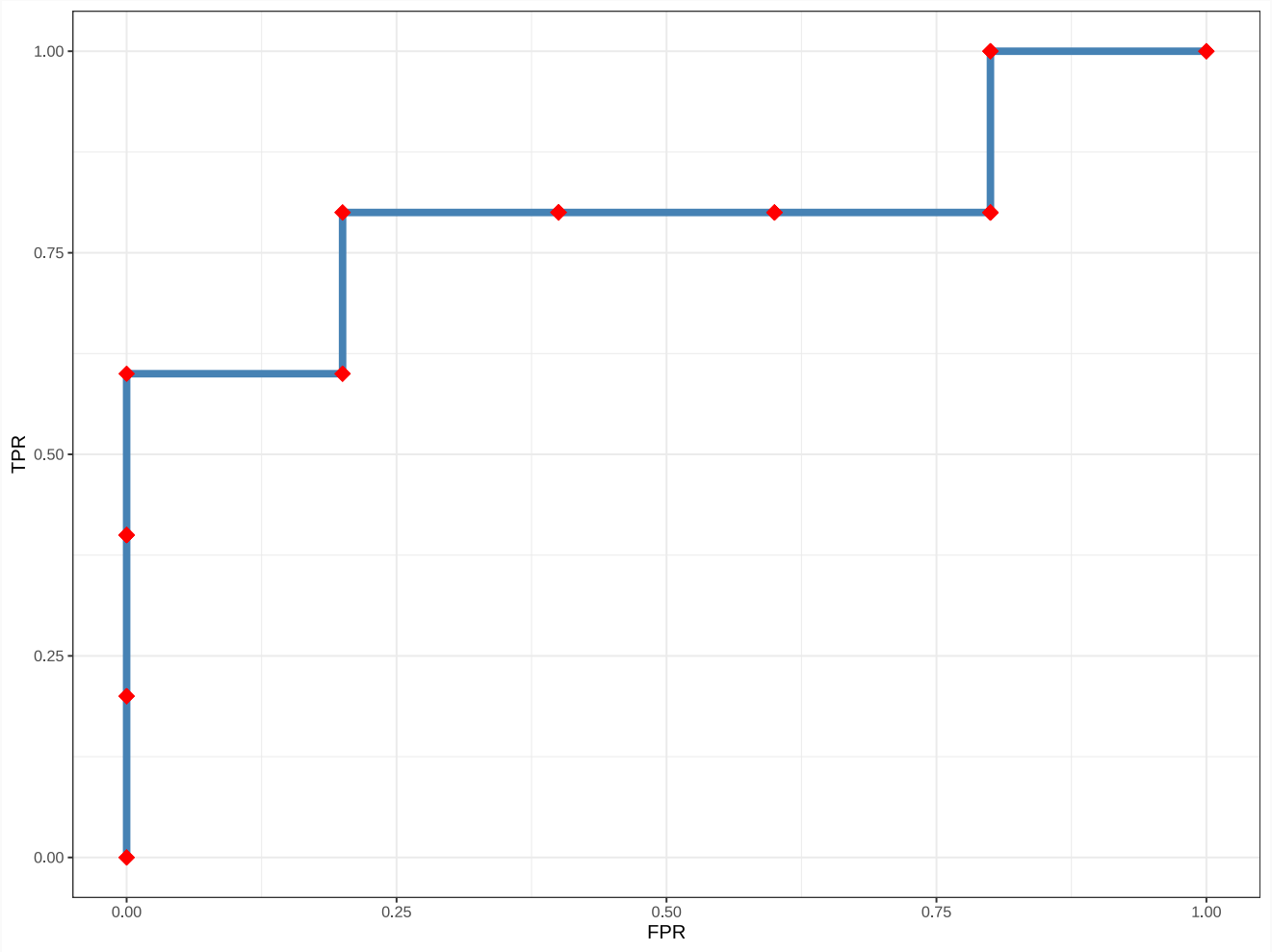
```
cal_ROC(df, 0.85)
#> $TPR
#> [1] 0.4
#>
#> $FPR
#> [1] 0
```

没问题，下面将该函数应用到一系列的阈值上（循环迭代），并一步到位将每次计算的两个结果按行合并到一起，这就彻底完成数据计算：

```r
c = seq(1, 0, -0.02)
rocs = map_dfr(c, cal_ROC, df = df)
head(rocs)      # 查看前 6 个结果
#> # A tibble: 6 x 2
#>     TPR   FPR
#>   <dbl> <dbl>
#> 1   0       0
#> 2   0       0
#> 3   0       0
#> 4   0.2     0
#> 5   0.2     0
#> 6   0.2     0
```

最后，用著名的 ggplot2 包绘制 ROC 曲线图形：

```
rocs %>%
  ggplot(aes(FPR, TPR)) +
  geom_line(size = 2, color = "steelblue") +
  geom_point(shape = "diamond", size = 4, color = "red") +
  theme_bw()
```


1.00
0.75
0.50
0.25
0.00
TPR
0.00
0.25
0.50
0.75
1.00
FPR

三. R 语言编程思想

1. 面向对象

R 是一种基于对象的编程语言，即在定义类的基础上，创建与操作对象；数值向量、函数、图形等都是对象。Python 的**一切皆为对象**也适用于 R。

```
a = 1L
class(a)
#> [1] "integer"

b = 1:10
class(b)
#> [1] "integer"

f = function(x) x + 1
class(f)
#> [1] "function"
```

早期和底层 R 语言中的面向对象编程是通过泛型函数来实现的，以 S3 类、S4 类为代表。新出现的 R6 类更适合用来实现通常所说的面向对象编程（OOP），包含类、属性、方法、继承、多态等概念。

2. 面向函数

笼统来说，R 语言就两个事情：数据，对数据应用操作。这个操作就是函数，包括 R 自带的函数，各种扩展包里的函数，自定义的函数。

所以，使用 R 大部分时间都是在与函数打交道，学会了使用函数，R 语言也就学会了一半[2]。很多人说 R 简单易学，也是因为此。

编程中的函数，是用来实现某个功能。很多时候，使用 R 自带的或来自其它包中的现成函数就够了。

那么，如何找到并使用现成函数解决自己想要解决的问题？比如想做线性回归，通过 Bing 搜索知道是用自带 `lm()` 函数实现。那么先打开该函数的帮助：

```
?lm
```

[2]前提是不把 R 当一门编程语言，只想简单套用现成的算法模型。

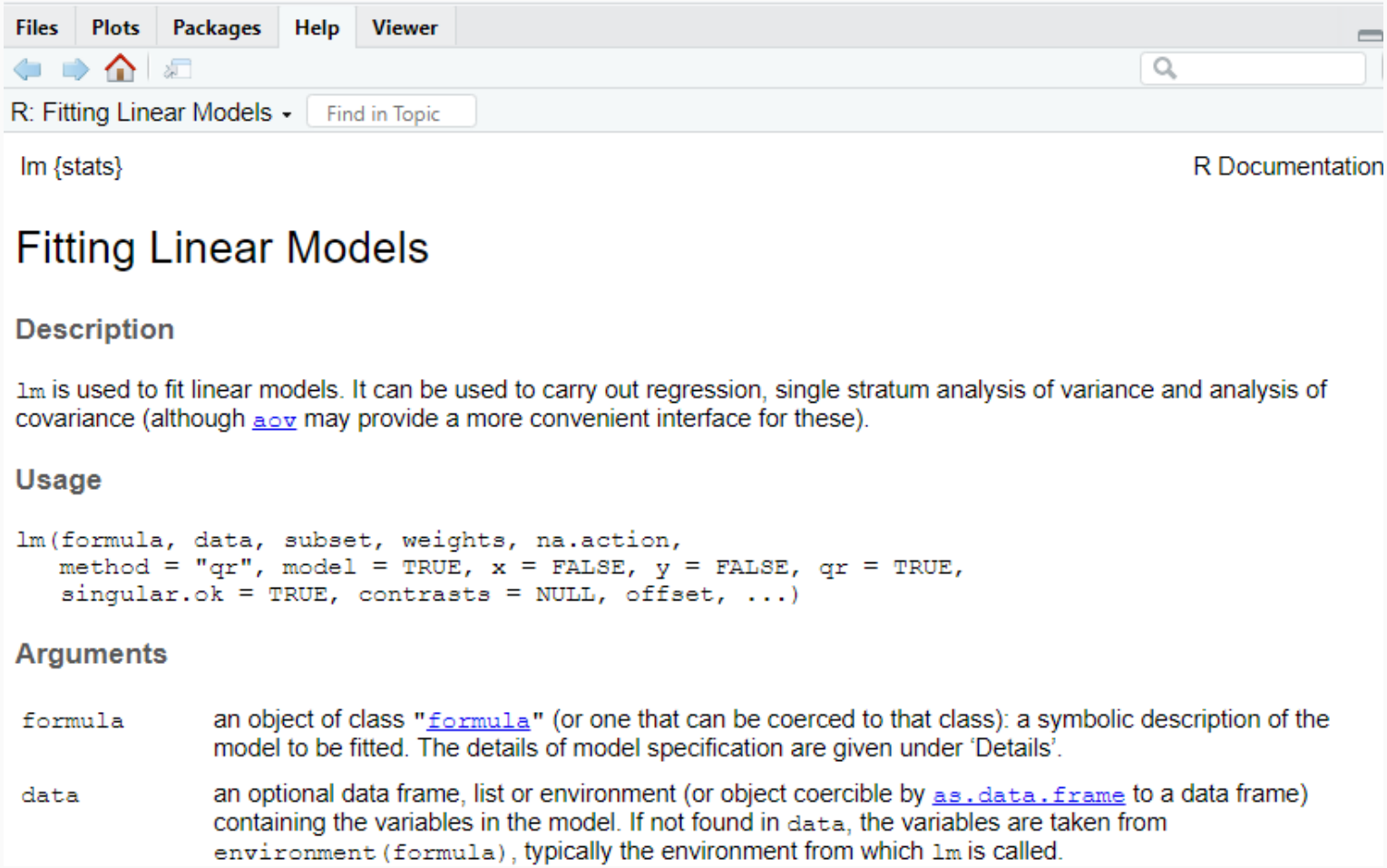

Files
Plots
Packages
Help
Viewer
R: Fitting Linear Models
Find in Topic
lm {stats}
R Documentation
Fitting Linear Models
Description
lm is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although aov may provide a more convenient interface for these).
Usage
lm(formula, data, subset, weights, na.action,
method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
singular.ok = TRUE, contrasts = NULL, offset, ...)
Arguments
formula
an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
data
an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which lm is called.

图 4: R 函数的帮助页面

执行？ 函数名，若函数来自扩展包需要事先加载包，则在 Rstudio 右下角窗口打开函数帮助界面，一般至少包括如下内容：

- 函数描述
- 函数语法格式
- 函数参数说明
- 函数返回值
- 函数示例

通过阅读函数描述、参数说明、返回值，再调试示例，就能快速掌握该函数的使用。

函数包含很多参数，常用参数往往只是前几个。比如 lm( ) 的常用参数是：

- formula: 设置线性回归公式形式：因变量 ~ 自变量 + 自变量
- data: 提供数据（框）

使用自带的 mtcars 数据集演示，按照函数参数要求的对象类型提供实参：

```
head(mtcars, 3)
#>                mpg cyl disp  hp drat   wt qsec vs am gear
#> Mazda RX4     21.0   6  160 110 3.90 2.62 16.5  0  1    4
#> Mazda RX4 Wag 21.0   6  160 110 3.90 2.88 17.0  0  1    4
#> Datsun 710    22.8   4  108  93 3.85 2.32 18.6  1  1    4
model = lm(mpg ~ disp, data = mtcars)
summary(model)      # 查看回归汇总结果
#>
#> Call:
#> lm(formula = mpg ~ disp, data = mtcars)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -4.892 -2.202 -0.963  1.627  7.231
#>
#> Coefficients:
```

编程中一种重要的思维就是**函数式思维**，包括自定义函数（把解决某问题的过程封装成函数）和泛函式编程（把函数依次应用到一系列的对象上）。

如果找不到现成的函数解决自己的问题，那就需要自己自定义函数，R 自定义中函数的基本语法为：

```
函数名 = function(输入 1, ..., 输入 n) {
  ...
  return(输出)          # 若有多个输出，需要打包成一个 list
}
```

比如，想要计算很多圆的面积，就有必要把如何计算一个圆的面积定义成函数，需要输入半径，才能计算想要的面积：

```
AreaCircle = function(r) {
  S = pi * r * r
  return(S)
}
```

这样再计算圆的面积，你只需要把输入给它，它就能在内部进行相应处理，把你想要的输出结果返回给你。如果想批量计算圆的面积，按泛函式编程思维，只需要将该函数依次应用到一系列的半径上即可。

比如计算半径为 5 的圆的面积和批量计算半径为 2,4,7 的圆的面积：

```
AreaCircle(5)
#> [1] 78.5
rs = c(2,4,7)
map_dbl(rs, AreaCircle)       # purrr 包
#> [1]  12.6  50.3 153.9
```

所以，定义函数就好比创造一个模具，调用函数就好比用模具批量生成产品。使用函数最大的好处，就是将实现某个功能，封装成模具，从而可以反复使用。这就避免了写大量重复的代码，程序的可读性也大大加强。

3. 向量化编程

高级编程语言都提倡**向量化编程**，说白了就是，对一列/矩阵/多维数组的数同时做同样的操作，既提升程序效率又大大简化代码。

向量化编程，关键是要用整体考量的思维来思考、来表示运算，这需要用到《线性代数》的知识，其实我觉得《线性代数》最有用的知识就是向量/矩阵化表示运算。

比如考虑 n 元一次线性方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \quad \cdots\cdots \qquad \cdots\cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

若从整体的角度来考量，引入矩阵和向量：

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

则前面的 n 元一次线性方程组，可以向量化表示为：

$$Ax = b$$

可见，向量化表示大大简化了表达式。这放在编程中，就相当于本来用两层 `for` 循环才能表示的代码，简化为了短短一行代码。

向量化编程其实并不难，关键是要转变思维惯式：很多人学完 C 语言后的后遗症，就是首先想到的总是逐元素的 for 循环。摆脱这种想法，调动头脑里的《线性代数》知识，尝试用向量/矩阵表示，长此以往，向量化编程思维就有了。

下面以计算决策树算法中的样本经验熵为例来演示向量化编程。

对于分类变量 D，$\frac{|D_k|}{|D|}$ 表示第 k 类所占的比例，则 D 的样本经验熵为

$$H(D) = -\sum_{k=1}^{K} \frac{|D_k|}{|D|} \ln \frac{|D_k|}{|D|}$$

其中，$|\cdot|$ 表示集合包含的元素个数。

实际中经常遇到要把数学式子变成代码，首先你要看懂式子，拿简单实例逐代码片段调试就能解决。

以西瓜分类数据中的因变量”好瓜”为例，表示是否为好瓜：

```
y = c(rep(" 是", 8), rep(" 否", 9))
y
#>  [1] " 是" " 是" " 是" " 是" " 是" " 是" " 是" " 是" " 否"
#> [16] " 否" " 否"
```

则 D 分为两类：D_1 为好瓜类，D_2 为坏瓜类。

从内到外先要计算 $|D_k|/|D|$, $k = 1, 2$, 用向量化的思维同时计算，就是统计各分类的样本数，再除以总样本数：

```
table(y)                          # 计算各分类的频数，得到向量
#> y
#> 否 是
#>  9  8
p = table(y) / length(y)    # 向量除以标量
p
#> y
#>    否    是
#> 0.529 0.471
```

继续代入公式计算，记住 R 自带函数天然就接受向量做输入参数：

```
log(p)                          # 向量取对数
#> y
#>      否      是
#> -0.636 -0.754
p * log(p)                      # 向量乘以向量，对应元素做乘法
#> y
#>      否      是
#> -0.337 -0.355
- sum(p * log(p))               # 向量求和
#> [1] 0.691
```

看着挺复杂的公式用向量化编程，核心代码只有两行：计算 p 和最后一行。这个实例虽然简单，但基本涉及所有常用的向量化操作：

- 向量与标量做运算；
- 向量与向量做四则运算；
- 函数作用到向量。

四. 本课程内容安排

第 1 章 R 语言基本语法

- 包括：搭建 R 环境、常用数据类型（数据结构）、控制结构（分支、循环）、自定义函数；
- 让读者打好 R 语言基本语法的基础，以及训练**函数式编程思维**：自定义函数解决问题 + 泛函式循环迭代；
- **基本语法**是编程的**编程元素**和**语法规则**，编写所有 R 程序都是用它们组合出来的；
- **函数式编程**，是下一章训练**数据思维**的基础。**函数式编程和数据思维**，是 R 语言编程的最核心编程思维，这些也是学习 R 语言的关键所在。

- 正式进入 `tidyverse` 系列，将全面讲解“管道流、整洁流”操作数据的基本语法，包括：数据读写、数据连接、数据重塑，以及各种数据操作；
- 本章最核心的目的是训练读者的数据思维：

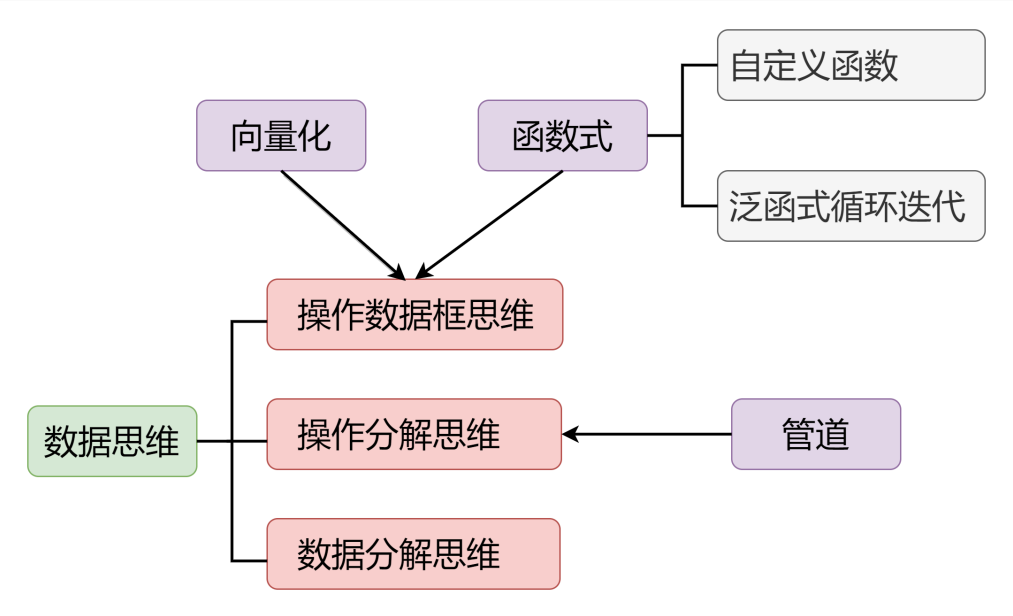


图 5: 我理解的数据思维

第 3 章可视化与建模技术

- 可视化历来是 R 语言的强项，本章将介绍最经典的 ggplot2 绘图，包括 ‘ggplot 的语法的 10 个部件：
  - **数据** (data)
  - **映射** (mapping)
  - **几何对象** (geom)
  - 标度 (scale)
  - 统计变换 (stats)
  - 坐标系 (coord)
  - 位置调整 (Position adjustments)
  - 分面 (facet)
  - 主题 (theme)
  - 输出 (output)
- 统计建模技术，将围绕整洁模型结果、建模辅助函数、批量建模展开。

第 4 章应用统计

R 语言就是因统计分析而生的编程语言，可以很方便地完成各种统计计算、统计模拟、统计建模等；将从四个方面展开：

- 描述性统计，介绍适合描述不同数据的统计量、统计图、列联表；
- 参数估计，主要介绍点估计与区间估计，包括 Bootstrap 法估计置信区间，以及常用的参数估计方法：最小二乘估计、最大似然估计；
- 假设检验，将介绍假设检验原理，基于理论的假设检验、基于重排的假设检验
- 回归分析：多元线性回归、逐步回归、回归诊断等。

- 数据清洗，包括缺失值探索与处理、异常值识别与处理；
- 特征工程，包括特征缩放（标准化/归一化/行规范化/数据平滑)、特征变换（非线性特征/正态性变换/连续变量离散化)、基于 PCA 的特征降维；
- 探索变量间的关系，包括分类变量之间、分类变量与连续变量、连续变量之间的关系。

- 如何进行可重复研究，用 R markdown 家族生成各种文档
  - R markdown 的基本使用
  - R 与 Latex 交互编写期刊论文/幻灯片/书籍
  - R 与 Git/Github 交互进行版本控制
- 用 R Shiny 轻松制作交互网络应用程序（Web App）
- 开发和发布 R 包的最新工作流程

本篇主要参阅 (张敬信, 2022), (王敏杰, 2021), 模板感谢 (黄湘云, 2021), (谢益辉, 2021).

参考文献

张敬信 (2022). *R* 语言编程：基于 *tidyverse*. 人民邮电出版社, 北京.

王敏杰 (2021). 数据科学中的 *R* 语言. bookdown.org.

谢益辉 (2021). *rmarkdown: Dynamic Documents for R*.

黄湘云 (2021). *Github: R-Markdown-Template*.