R 语言数据分析组队学习

张晋、杨佳达、牧小熊、杨杨卓然、姚昱君

目录

欢迎！

欢迎来到由 DataWhale 主办的 R 语言数据分析组队学习课程。

贡献者信息

姓名	介绍	
张晋	Datawhale 成员，算法竞赛爱好者	
杨佳达	数据挖掘师，Datawhale 成员，目前在国内某第三方数据服务公司做数据分析挖掘及数据产品	
牧小熊	华中农业大学研究生，Datawhale 成员，Datawhale 优秀原创作者	
杨杨卓然	混吃等死统计休学穷酸书生	
姚昱君	悉尼大学，Datawhale 成员	

课程简介

- 课程设计成员：**张晋**、杨佳达、牧小熊、杨杨卓然、姚昱君
- 学习周期：16 天，每天平均花费时间 1 小时-3 小时不等，根据个人学习接受能力强弱有所浮动。
- 学习形式：理论学习 + 练习
- 人群定位：对数据科学有基本了解，希望学习 R 语言的同学。
- 先修内容：无
- 难度系数：

课程大纲

Task00：熟悉规则与 R 语言入门（1 天）

- 安装
- 环境配置

Task01 数据结构与数据集（3 天）

- 编码基础
- 数据类型
- 特殊数据类型
- table like 数据类型
- 加载数据 (csv, rds, excel, Rdata)
- 实例

Task02 数据清洗与准备（3 天）

- 重复值处理
- 缺失值识别与处理
- 异常值识别与处理
- 特征处理
- 规范化与偏态数据

Task03 基本统计分析（3 天）

- 多种方法获取描述性统计量
- 分组计算描述性统计
- 频数表和列联表
- 相关
- 方差分析

Task04 数据可视化（3 天）

- ggplot2 包介绍
- 散点图
- 直方图
- 柱状图
- 饼状图
- 折线图
- ggplot2 扩展包主题

Task05 模型（3 天）

- 回归模型
- 分类模型

关于 Datawhale

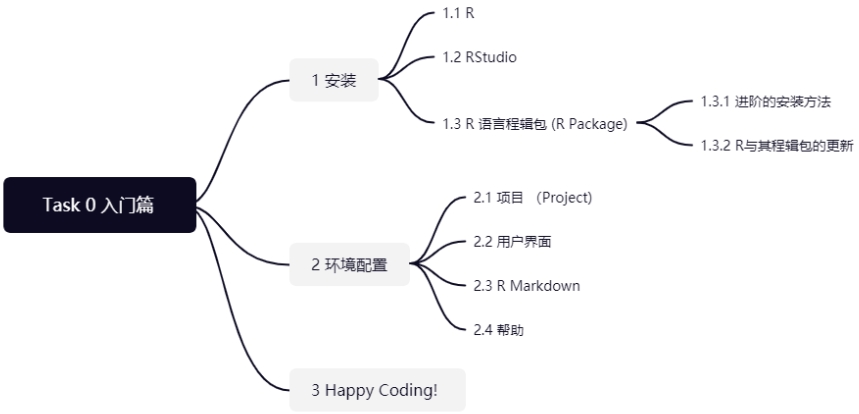
Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成

员。Datawhale 以 “For the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：

第一部分

准备工作

熟悉规则与 R 语言入门



0.0.1 安装

0.0.1.1 R

- R 语言是一门用于统计计算与绘图的编程语言和开源软件 (The R Foundation)。
- R 语言是当今应用最多的统计软件之一。
- 截止到这份文档编写时，在 CRAN (the Comprehensive R Archive Network) 上总共发行了 17955 个 R 语言程辑包。
- 除了 CRAN 以外，R 语言的程辑包作者也在其他线上代码托管与研发协作平台发布了数不尽的作品。这样的平台包括 GitHub、GitLab、Gitee 等。

你可以从 CRAN 的网站上下载 R：https://cloud.r-project.org/。你也可以在这里选择任意一个镜像网站来下载 R.

0.0.1.2 RStudio

RStudio 是针对 R 语言设计的集成开发环境。如果没有 RStudio 的话，R 本身只提供一个简单的文本编辑器。如果把 R 语言本身比喻成飞机的话，那

么 RStudio 便是飞机场：你不需要它就可以飞，但是有了它会极大增加效率。它包括一个控制台、语法突出显示的编辑器、直接执行代码的支持，以及用于绘图、历史记录、调试和工作区管理的工具。

你可以从其官网下载开源版本：https://rstudio.com/products/rstudio/

在本文档中，我们会介绍 RStudio 的用户界面和部分功能，帮助你尽快上手使用 RStudio 的数据分析。

0.0.1.3 R 语言程辑包（R Package）

R 语言程辑包是 R 语言必不可少的部分。R 语言能有他今天在统计学里的位置正是归功于其程辑包在统计计算方面的发展。一个程辑包为用户提供函数（function）、数据（data）或者插件（addins）。除了 R 本身自带的基础程辑包（base、utils、stats 等）以外，你还可以用以下代码来从 CRAN 上下载并安装额外的程辑包：

```
install.packages("tidyverse")
```

我们将会在这次组队学习中多次用到tidyverse。它其实是一系列程辑包的组合，主要提供数据清洗与处理的工具。

0.0.1.3.1 进阶的安装方法

当你应用 R 语言的能力到一定阶段之后，你会发现自己需要安装不在 CRAN 上发布的程辑包，或者你需要最新版本的程辑包（CRAN 上的包为了保证代码的可靠性，发布前需要经过一系列的检查与测试，这就导致 CRAN 上的版本往往不是最新的开发版本）。以安装 GitHub 上发布的程辑包为例，你可以使用以下代码：

```
# 安装 remotes 包
install.packages("remotes")
# 使用 remotes 从 GitHub 上安装 username 名下的 repo
    包
remotes::install_github("username/repo")
```

我们需要先安装 remotes 包，并使用其中的 install_github 函数来完成操作。注意这里是从源代码安装，在本地编译。Windows 用户需要使用 Rtools 作为背后的编译工具。关于 Rtools 的安装信息见 https://cran.r-project.org/bin/windows/Rtools/

0.0.1.3.2 R 与其程辑包的更新

在本文档编写之时，R 语言已更新到版本 R version 4.1.0 (2021-05-18)。当新的版本发布时，你可以使用 installr 包中的 installr 函数来完成 R 的更新（你当然也可以手动下载更新，如果不嫌麻烦的话）。代码如下：

```
# 安装 installr 包
install.packages(installr)
# 更新 R
installr::installr()
```

根据对话窗口中的提示完成整个安装过程即可。

你也可以使用以下代码来更新 R 的程辑包：

```
# 手动确认是否将各个更新到最新版本，或者
update.packages()
# 更新所有包到最新版本
update.packages(ask = FALSE)
```

0.0.2 环境配置

0.0.2.1 项目（Project）

在 RStudio 中一个项目（Project）本质上是一个把项目相关的文件储存在一个地方的文件夹。如果使用项目相关的功能的话，你不需要担心使用的文件是否在当前的工作目录（Working Directory）。项目功能提供了一个将不同目的的文件分隔开的方式，同时自动保存上次相应的工作进度。

0.0.2.1.1 * 练习

为这次组队学习建立一个新的项目。

每次进行组队学习的时候，不要忘记去打开这个项目。在结束工作退出 R，或者切换到另一个项目的时候，为了下次打开 RStudio 的时候有一个干净的工作环境，建议不去保存“工作空间镜像”（Workspace image，即在当前进程中加载的数据、函数）。

创建新的项目，可以在下拉菜单 **File** （或者 RStudio 界面的右上角）找到 **New Project** 选项。在弹出的对话框中，如果你想创建一个新的文件夹作为项目文件夹，选择 **New Directory** ；如果你想用一个已经存在的文件夹作为项目文件夹，选择 **Existing Directory** 。

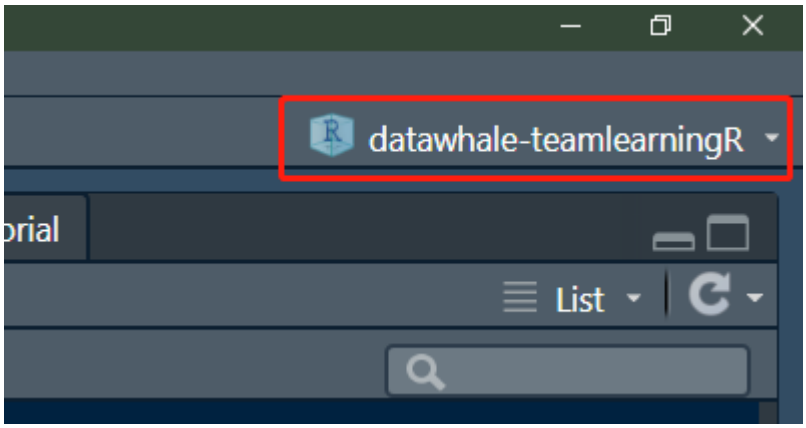
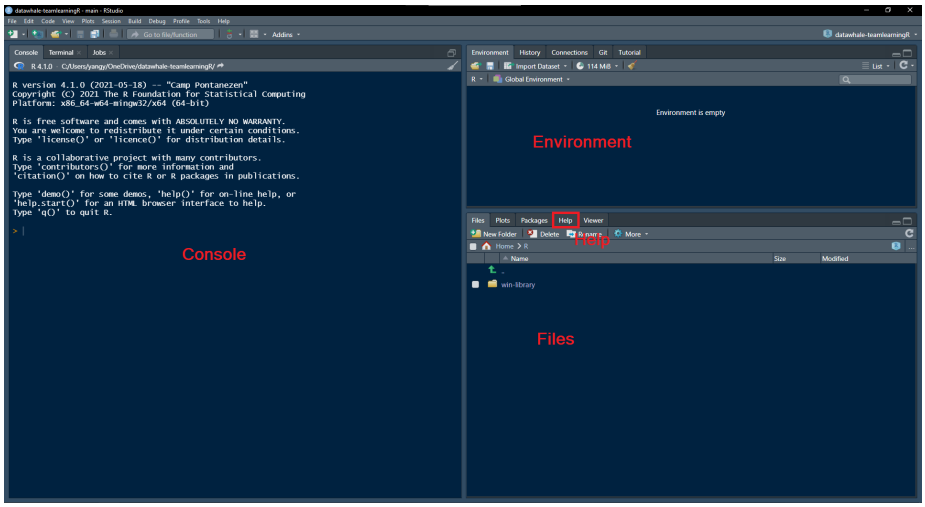


图 1: RStudio 界面右上角的项目设置[1]

0.0.2.2 用户界面

接下来让我们关注一下 RStudio 用户界面里的各种面板和标签。在这一部分有四个基础面板值得注意。

[1] 小提示：如果想要改变 RStudio 的主题颜色，可以通过 **Tools** » **Global Options..** » **Appearance**



- **Console** 控制台位于界面左侧。你可以在这里运行命令、浏览输出结果。
- **Environment** 位于界面右上方。环境面板会总结出当前加载的数据、定义的函数等。现在在你看来可能是空的，因为我们还没有做任何事情。
- **Files** 位于界面右下方。文件面板展示当前文件夹的信息。
- **Help** 帮助面板也位于界面右下方。在这里你可以找到相应数据或者函数的帮助信息。

0.0.2.3 R Markdown

你可以控制台（Concole）直接运行命令，但是这样跑出来的指令不会被保存下来。一般来说，我们更建议将命令写在叫做 R script 的脚本中，或者写在一个叫做 R Markdown 的文件中。

0.0.2.3.1 * 练习

创建一个脚本 R script.

- **File » New File » R Script**

0.0.2.3.2 * 练习

创建一个 R Markdown 文件。

- **File » New File » R Markdown..**

在脚本或者 R Markdown 的界面中，界面上方可以找到一个运行所有代码的按键。

一个 R Markdown 文件是一个可以将代码与 *markdown* 标准文本（一种纯文本的格式语法）结合在一起的文本文档。使用 R Markdown 文件可以很容易地生成 pdf 文件或者 html 文件，其中不止包含了你的文本，还有代码以及运行代码所生成的结果。点击界面上方的 **Knit** 按键[2]即可。再也不需要复制粘贴、屏幕截图输出结果到 Word 了。R Markdown 文档的一个主要优势是可复现。只要有了同样的代码和数据，你可以获得与其他人一模一样的结果，只要生成文档就可以了。

在 R Markdown 文件里写代码，需要使用特定的代码块（code chunks）来告诉 R Markdown 这部分是需要运行的代码而不只是文本。[3]

```
```{r}

# 在这里写你的代码
# 使用三个反引号和 {r} 起始，三个反引号结束来构建代码
    块
# 在代码块里使用井号 # 写评论（纯文本）

```
```

0.0.2.4 帮助

R 能够发展到其今天的地位，很大的一个因素是他提供了相对详细的帮助文档，对初学者相对友好。一个相对完整的 R 包最低标准便是有函数的帮助文档。需要查看一个具体函数或者数据的帮助时可以用 ?fun（等同于

[2]如果你想要生成 pdf 文件，你需要安装 LaTeX。可以看看很好地兼容了 R 的TinyTex。

[3]R Markdown 的更多语法可以看看 R Markdown cheatsheet。

help(fun))，该函数 fun 的帮助文档便会出现在之前提到过的帮助面板里。这个是已经知道需要什么函数了之后查找具体函数的用法的方式，如果你不记得具体的函数名字，可以使用两个问号加关键字来搜索：??keyword。

其次 R 包会有一个或多个 vignette。vignette 文档的目的主要是当使用者不知道用什么函数，对这个包不了解的时候提供一份入门简介一样的东西，一般会对常用的函数做出说明和演示，以及一些理论的阐述。这个包如果是哪一篇论文的副产品，vignette 甚至有可能是这篇论文。根据包的大小不同，vignette 的数量也不一样。如果是针对于一个问题写出的精炼的小包的话会只有一个 vignette 。如果包的用途比较广泛或者作者想说的话比较多，会针对每个问题有一个单独的 vignette。浏览所有已安装的 vignette 用 browseVignettes() ，查看具体包的用browseVignettes("packagename")。以上两个是通过 CRAN 发行的包的标配。

如果这个包没有在 CRAN 上发行，只在 GitHub 上，或者 GitHub 上有开发版本的话，一般会有一个 README.md 的文档。这个文档相对于 vignette 来说更加简短，一般都只写明如何安装，以及最主要的命令的演示，没有太多的说明。文档最后有可能会说明这个包用的是什么许可证。如果有这么一个文件的话，就可以很快速的知道这个包最主要的命令是什么。这个文档就需要到相对应的 R 包的资源库搜索了。

```
## 总结
# 查看具体函数的帮助文档
?fun
help(fun)
# 在帮助文档中搜索关键词
??keyword
# 浏览所有已安装的vignette
browseVignettes()
# 查看具体包的vignette
browseVignettes("packagename")
```

0.0.3 Happy Coding!

这次的 R 语言数据分析组队学习的入门篇便到这里了。接下里请移步组队学习的正篇第一部分：数据结构与数据集。

玩得开心！

本章作者

Fin

https://yangzhuoranyang.com

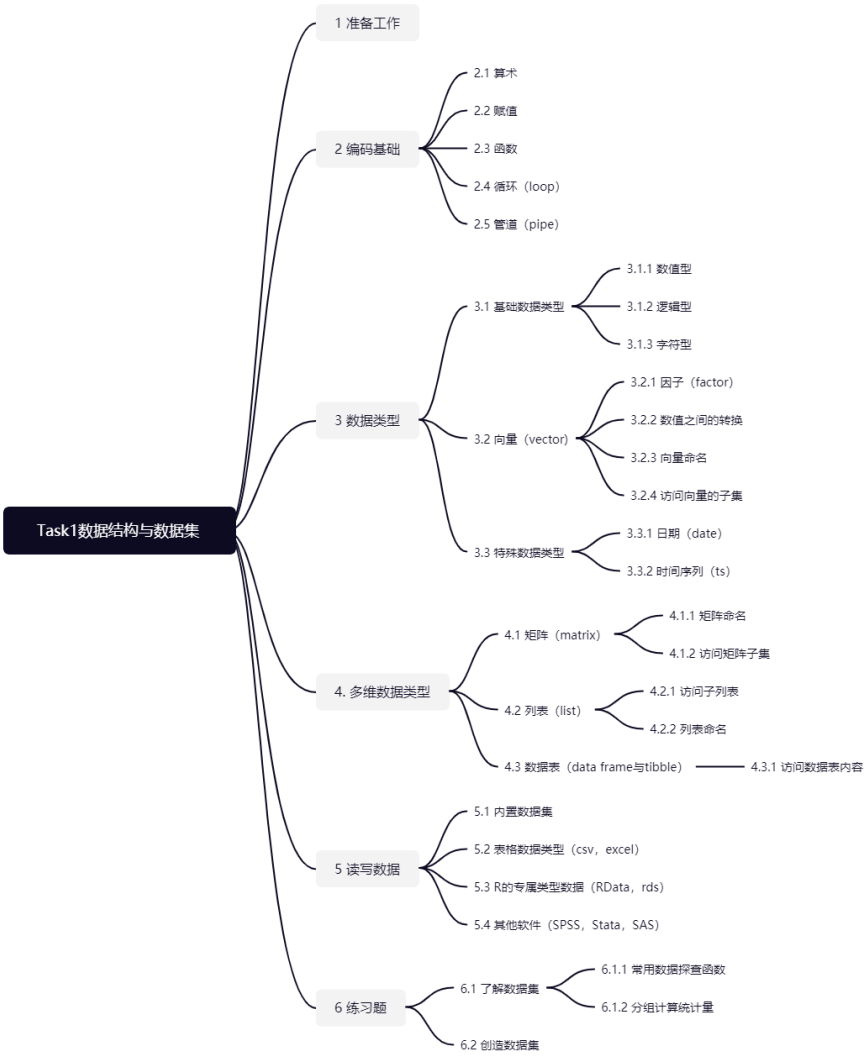
关于 Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以 “for the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：

第二部分

开始干活

0.1 数据结构与数据集



0.1.1 准备工作

这节组队学习的目的主要是帮助你上手 R 的基本编程逻辑，了解一些 R 编程的基本概念，包括各个数据类型和数据集的读取与保存。

在开始我们的学习之前，不要忘记在 R Studio 中切换到组队学习专属的项目，打开一个 R script 文件或者 R Markdown 文件（详见入门篇）。

0.1.2 编码基础

首先我们来了解一些基本的编码操作。在 R Studio 中想要运行代码可以在控制台 Console 中键入代码后点击回车。这样运行的代码会被保存在当前项目的 .Rhistory 文件中，也可以在 R Studio 界面右上角的 History 面板中找到，但是不会被明确地保存下来作为一个脚本文件。一般只有在我们想要运行一些简单的指令或者计算的时候才会采取这种方式。更常见的是将代码写在脚本文件中，选中相应的代码后点击界面上方的 Run 或者快捷键（Ctrl + Enter）来运行。

0.1.2.1 算术

你可以直接运行计算命令。计算符号包括加+、减–、乘*、除/、求幂^以及求余数%%等。值得一提的是开平方根有他自己单独的函数sqrt。

```
1 + 1
```

```
## [1] 2
```

```
1 - 1
```

```
## [1] 0
```

```
1 * 2
```

```
## [1] 2
```

```
1 / 2
```

```
## [1] 0.5
```

```
3 %% 2
```

```
## [1] 1
```

```
2^(1 / 2)
```

```
## [1] 1.414214
```

```
sqrt(2)
```

```
## [1] 1.414214
```

0.1.2.2 赋值

在 R 里，我们可以为一个“东西”取一个名字，这个“东西”可以是一个值、一个向量、或者一个函数等，这样我们就可以之后再获取存储在这个名字下面的信息。

```
# 将数字42赋予名叫x的变量
x <- 42
# 在R中运行一个物体的名字
# R将会打印出（print）该物体的值
x
```

```
## [1] 42
```

在 R 中基础赋值的符号有三种：

1. 一个向左的箭头<–表示将箭头右方的值取名叫做箭头左侧的名字，或者将箭头右侧的值存储在箭头左侧的名字里；
2. 一个向右的箭头–>表示将箭头左侧的值储存在箭头右侧的名字里；
3. 一个等号=表示将箭头右侧的值存储在箭头左侧的名字里（同 1）。

在早期的键盘中有一个单独的按键就是向左的箭头，虽然后来的键盘不再设立这个按键，但是使用箭头的编程习惯流传了下来。赋值符号的选择取决于个人习惯，但是我们大多数情况下都推荐使用箭头（尤其是向左的箭头）作为赋值的符号。这是 R 语言于其他语言不同的地方，有以下原因：

1. 箭头明确了赋值方向，这是等号做不到的；

2. 等号用在顶层环境中的时候是赋值，用在函数中则是设参（或者叫做在函数层面赋值）。这种二义性不小心区分则可能会引发错误。而等号即使用在函数中也是赋值；
3. 箭头可以做到多次赋值（a <− b <− 42）甚至是不同方向上多次赋值（a <− 42 −> b）（尽量避免!）；
4. 虽然这次组队学习中不会学到，但是更高级的赋值工具包括 <<− 和 −>> 对应向左或向右的箭头；
5. 同时使用=与==（判断是否相等）会降低可读性（a <− 1 == 2 vs a = 1 == 2）。

总结，**凡是赋值都用<−**，凡是设参（之后会说到）都用=。

在 R Studio 中可以使用快捷键Alt + −来输入<−，这样就不用每次都点两次键啦。这样还有一个好处，就是 R Studio 会自动识别当前的输入语言，从而选择最佳的赋值符号。大多数情况下这也就是说他会输入 R 的<−，如果你在 R Studio 里用 Python 的话就会自动变成=啦。

原来作用于单纯数字上的算术运算现在即可用变量名称代替具体的数值。

```
y <− 21
x + y
```

```
## [1] 63
```

```
x <− x + y
```

0.1.2.3 函数

R 是一个非纯函数式编程（Functional Programming）的语言，与你平时可能所熟悉的面向对象程序设计（Object-Oriented Programming）的编程语言（比如 Python）不一样。这意味着在 R 中，相对于以类（Class）与对象（Object）的思路思考问题，我们要更多地定义函数（Function）以及考虑函数的输入与输出来进行运算。（如果你不知道我在这里说什么，请忽略这段话。）

在 R 中，所有的运算都是通过函数来达成的。我们可以用和之前一样的赋值方法（<−）来将一个函数存储在一个名字下。请看以下示例：

```
addone <- function(x = 0) {
  x + 1
}
```

这里我创建了一个名为addone的函数，这个函数的作用就是将输入值在函数内部存储在名为x的参数里，在名为x的值上加一，再返回结果。如果没有输入值的话，x的默认值是x = 0。

```
# 作用在数字上
addone(42)
```

```
## [1] 43
```

```
# 作用在变量上
y <- 42
addone(y)
```

```
## [1] 43
```

如你可见，调用函数的方法就是在函数的名字后边加小括号，再在小括号中输入参数（arguments）。当函数有多个可选参数的时候，建议输入参数的时候使用等号=明确函数名称。这便是之前提到过的等号的设参用法。

```
addone(x = 42)
```

```
## [1] 43
```

如果你没有使用小括号而是直接在控制台中运行函数的名字的话，像以前一样，R 会直接打印出这个函数的内容，即源代码：

```
addone
```

```
## function(x = 0) {
##   x + 1
## }
## <environment: 0x0000000015e84508>
```

当你完成了一个复杂的计算，不要忘记把结果储存在一个名字下，否则结果不会保存下来，只会在控制台中一闪而过。

```
y <- 42
y_plusone <- addone(y)
```

0.1.2.4 循环（loop）

使用代码很重要的一个原因是可以重复进行多次相同或有规律的操作，也就是循环了。

R 中的循环函数包括for，while，和repeat。在这里我们简单用一个例子来介绍一下最灵活的for循环：

```
x <- 0
for(i in 1:3){
  x <- x + i
  print(x)
}
```

```
## [1] 1
## [1] 3
## [1] 6
```

在最开始的时候，我们让x等于 0。在接下来进行的循环操作中，紧跟在for之后的小括号里给出了每个回合当中会变化的参数，叫做i。i后边的in之后给出的是参数i在回合中的可能取值，也就是从 1 到 3 的正整数。最后大括号中给出每个回合的操作，在x上加上i的值，重新取名为x，再打印出来。

整个流程下来：

第一个回合x一开始是 0，在第一个回合中i是 1，经过计算赋值x变成了 1，打印后进入第二个回合；
第二个回合x一开始是 1，在第二个回合中i是 2，经过计算赋值x变成了 3，打印后进入第二个回合；
第三个回合x一开始是 3，在第三个回合中i是 3，经过计算赋值x变成了 6，打印后结束循环。

0.1.2.5 管道（pipe）

如果我们想要对一个对象进行多个函数的操作，比如说想要使用我们刚刚定义的addone函数，还有新定义的addtwo，addthree，我们可以按照普通调用函数的方法一个套一个：

```
addone <- function(x) x+1
addtwo <- function(x) x+2
addthree <- function(x) x+3

x <- 0
addthree(addtwo(addone(x)))
```

```
## [1] 6
```

在这种常规的方法下，函数运行的顺序，和我们读函数的顺序，都是从内到外的。比如在上边的操作中，我们先用了addone给 0 加 1，又用了addtwo，最后用了addthree。这样的坏处也是显而易见的，即可读性很差。想象一下你要对一个数据列表连续使用十几个函数，每个函数里都有其自己不同的参数，这么一系列操作如果用这个常规方法的话必然会使代码变成一个很难读的庞然大物。

magrittr包提供了另一种使用函数的办法，即使用%>%这个符号函数进行方法链（method chain）的操作。你可以把这个符号叫做管道。如果我们用管道来重写之前的一连串操作，代码会变成：

```
# tidyverse也包含了管道符号
library(tidyverse)

x %>%
  addone() %>%
  addtwo() %>%
  addthree()
```

```
## [1] 6
```

这个符号的具体含义简单来说是 “将上一步运行的结果放在下一步运行的函数的第一个参数的位置上”。在这个例子中，x被当作addone的第一个参数

被加一。addone的运行结果被当成下一步addtwo的第一个参数被加二，其运行结果最后被当成addthree的第一个参数被加三，最终得到结果。

经过了管道的改写之后，函数的可读性得到了大幅上升。从常规的 “从内到外” 读法，变成了 “从上到下，从左到右”。虽然需要运行额外 R 包，但是由于符合阅读习惯和数据清洗流程的特点，管道在数据分析的领域被普遍使用。

关于管道符号的具体使用规则详见?‘%>%‘。

0.1.3 数据类型

0.1.3.1 基础数据类型

在 R 中有五种基础数据类型，包括三个数值型、一个逻辑型和一个字符型。

0.1.3.1.1 数值型

数值型数据包括三种，分别是默认的实数数值型数据（double）、整数类型（integer）和复数类型（complex）：

```
# numeric
a <- 132.2345
# Inf
# integer
b <- 132L
# complex
c <- 2 + 3i
```

实数数值型数据每个值占用 8 个字节（bytes），是最常见的数值型数据。如果没有做特别处理，我们平时见到的数字都是这个类型的——单纯的数字罢了。

整数类型，正如它的名字一样，只包含整数而没有小数部分。我们可以在整数末尾加上一个大写的 L 来表示这个数字是一个整数类型的数据。如果没有加大写的 L 的话，虽然只输入了一个整数，但是这个整数是实数数值类型的整数，而不是整数类型。他们的区别在于实数数值类型的整数和和非整

数一样都占用 8 个字节，而整数类型只占用 4 个字节。平时用起来区别不大，但是如果数据量比较大且都是整数的话推荐使用整数类型来节约空间。

复数类型便是包含复数部分的数值类型了。只要在实数部分后边加上虚数部分并且用小写字母 i 来代表虚数单位，这个数值便是复数类型。鉴于数据分析领域基本不会涉及复数，我们在这次组队学习不去讨论复数类型。

判断一个数值是什么类型，可以用typeof():

```
typeof(a)
```

```
## [1] "double"
```

```
typeof(b)
```

```
## [1] "integer"
```

```
typeof(c)
```

```
## [1] "complex"
```

0.1.3.1.2 逻辑型

逻辑型（logical）数据只包括两个值，TRUE（T）和 FALSE（F）:

```
TRUE
```

```
## [1] TRUE
```

```
T
```

```
## [1] TRUE
```

```
FALSE
```

```
## [1] FALSE
```

```
F
```

```
## [1] FALSE
```

尽管一个字母的缩写和全拼效果是一样的，但是一个好的编程习惯是始终使用大写全拼的TRUE和FALSE。这样可以增加可读性，也会减小因为命名产生的使用错误。比如，有些时候涉及到时间序列时，一些用户喜欢将最大时序上限命名为T，这个时候就不能用T来代表TRUE了。

说到逻辑型数据，就不得不说到逻辑算符。这里我们只考虑三个，分别是“和”（and）&、“或”（or）|、“否”（not）!。

```
TRUE & FALSE
```

```
## [1] FALSE
```

```
TRUE | FALSE
```

```
## [1] TRUE
```

```
!TRUE
```

```
## [1] FALSE
```

0.1.3.1.3 字符型

字符型数据（character）可以总结为“任何带引号的值”。它可以是一个字母、一个单词、一句话、或者任何用引号框起来的数值或逻辑。

```
string_a <- "A"
string_b <- "letter"
string_c <- "This is a sentence."
string_d <- "42"
string_e <- "TRUE"
```

在输入的时候，即使是数字或者逻辑型的TRUE和FALSE，只要加上了引号，他们就变成了字符型的数据，而不再带有数值型或逻辑型的特性。要注意区分。

```
typeof(42)
```

```
## [1] "double"
```

```
typeof("42")
```

```
## [1] "character"
```

```
typeof(TRUE)
```

```
## [1] "logical"
```

```
typeof("TRUE")
```

```
## [1] "character"
```

字符型是最“自由”的数据类型，因为它的内容可以是任何字符，任何其他的数据类型也可以转化为字符型数据。比如你可以把一个数值型数据加上引号来当作字符型数据来解读。但是反过来却不可以：你没有办法把一个字符当作数字来解读。数据之间的转化我们会在第 3.2.2 节讲到。

在 R 中，单引号（'）和双引号（"）是等效的，但是我们推荐大多数情况下使用双引号，只有在引号内有双引号的时候使用单引号去引双引号（比如' This is an "example". '）。这主要是为了帮助其他语言（C, C++, Java 等）的用户区分单双引号的细微区别。在 C 语言里，单双引号不是等效的。R 语言中的（单）双引号大致是与 C 语言中的双引号等效的。

0.1.3.2 向量（vector）

这里说到的向量主要指基础向量类型（atomic vector）。向量是由一组相同类型的值组成的一维序列。根据值的类型不同，我们会有不同类型的向量。相对应之前的数值、逻辑和字符型的基础数据类型，这里我们也有数值、逻辑和字符型的向量类型。

```
vec_num <- c(1, 2, 3)
vec_log <- c(TRUE, FALSE, TRUE)
vec_cha <- c("A", "B", "middle school")
```

使用函数c()来构建向量。可以进行向量上的运算，而不用一个一个值地单独去计算。

```
vec_A <- c(1, 2, 3)
vec_B <- c(3, 5, 6)
vec_A + vec_B # 等同于 c(1 + 3, 2 + 5, 3 + 6)
```

```
## [1] 4 7 9
```

```
!vec_log
```

```
## [1] FALSE  TRUE FALSE
```

也有相应的作用于向量上的函数，可以计算相应的统计量。比如求和的sum、求方差的var、平均值的mean等：

```
sum(vec_A)
```

```
## [1] 6
```

```
var(vec_A)
```

```
## [1] 1
```

```
mean(vec_A)
```

```
## [1] 2
```

0.1.3.2.1 因子（factor）

除了之前提到的基础数据类型组成的向量外，还有一类重要的的向量类型便是因子，可以使用函数factor和c组合来创建。

```
vec_fac <- factor(c("male", "female", "male", "female"
    , "female"))
vec_fac
```

```
## [1] male   female male   female female
## Levels: female male
```

从表面上看，一个因子向量和字符向量很相似，都是一系列带引号的字符组成的。它与字符向量的主要区别在于因子向量的独特值（levels）是有限个数的。因子向量的所有元素都是由这些有限个数的独特值组成的。比如在以上的例子中，虽然vec_fac由五个元素组成，但是只包括了两个独特值 “male” 和 “female”。

```
# 查看因子向量的独特值
levels(vec_fac)
```

```
## [1] "female" "male"
```

你也可以用函数ordered或者factor里的ordered = TRUE参数（argument）创造一个有内在顺序的因子向量，内在顺序可以用 levels 参数来手动设定：

```
educ <- ordered(
  c("kindergarten", "primary␣school", "middle␣school",
    "primary␣school", "middle␣school", "kindergarten")
      ,
  levels = c("kindergarten", "primary␣school", "middle
    ␣school")
)
# 等同于
educ <- factor(
  c("kindergarten", "primary␣school", "middle␣school",
    "primary␣school", "middle␣school", "kindergarten")
      ,
  ordered = TRUE,
  levels = c("kindergarten", "primary␣school", "middle
    ␣school")
)

educ
```

```
## [1] kindergarten   primary school middle school
    primary school middle school
## [6] kindergarten
```

```
## Levels: kindergarten < primary school < middle
    school
```

实质上，R 把因子向量当作整数型数值向量来对待。这也就意味着用因子向量替代字符向量可以剩下很多字节。

0.1.3.2.2 数值之间的转换

不同的向量/数据类型之间是可以互相转换的。相互转换的可行性取决于数据类型的复杂程度（或者说自由程度）。按照自由程度将已经提到的几种向量以从高到低的排序可得

字符 > 数值 > 逻辑

在数值型内的排序从自由度高到低为

复数 > 实数 > 整数

越靠近字符的类型越“自由”，自由度低的类型可以（随意）转化为同层或自由度更高的类型。字符型向量是最自由的：它可以包含任何原始值，其他任何类型都可以转化为它。我们以一个最受限制的逻辑向量为例，在这里展示如何根据这个排序使用几个常见的类型转换函数：

```
vec_loc <- c(TRUE, FALSE, TRUE)
# 从逻辑型到数值型
vec_num <- as.numeric(vec_loc)
vec_num
```

```
## [1] 1 0 1
```

```
# 从数值型到字符型
vec_cha <- as.character(vec_num)
vec_cha
```

```
## [1] "1" "0" "1"
```

```
# 从逻辑型到字符型
vec_cha2 <- as.character(vec_loc)
vec_cha2
```

```
## [1] "TRUE"  "FALSE" "TRUE"
```

```
## 倒序
# 从字符型到数值型
as.numeric(vec_cha)
```

```
## [1] 1 0 1
```

```
# 从字符型到逻辑型
as.logical(vec_cha2)
```

```
## [1]  TRUE FALSE  TRUE
```

```
# 从数值型到逻辑型
as.logical(vec_num)
```

```
## [1]  TRUE FALSE  TRUE
```

这里我们可以看到逻辑型的TRUE和FALSE实际上对应数值型的 1 和 0。

从一个低自由的类型可以随便转化到高自由的类型，但是反过来，从一个高自由的类型要转化到一个低自由的类型必须要符合一些特定值。比如：

1. 从字符型转化到数值型的时候，字符的值一定要符合数字的格式；
2. 从数值型转化到逻辑型，0 会转化为FALSE，其他数值会转化为TRUE；
3. 从字符型转化到逻辑型，字符的值只能是TRUE和FALSE。

如果不符合这个规则的话，会得到NA。NA是 “Not Available” 的缩写，即所谓的缺失值。缺失值的处理在下一篇《数据清洗与准备》会讲到。

```
# 产生缺失值
as.logical(c("some", "random", "strings"))
```

```
## [1] NA NA NA
```

因子型是一个相对特殊的类型，它可以和数值型与字符型相互转换。

```
vec_fac <- factor(c("male", "female", "male", "female"
    , "female"))
# 从因子型到数值型
vec_num <- as.numeric(vec_fac)
vec_num
```

```
## [1] 2 1 2 1 1
```

```
# 从因子型到字符型
vec_cha <- as.character(vec_fac)
vec_cha
```

```
## [1] "male"   "female" "male"   "female" "female"
```

```
# 从字符型到因子型
as.factor(vec_cha)
```

```
## [1] male   female male   female female
## Levels: female male
```

```
# 从整数型到字符型
as.factor(vec_num)
```

```
## [1] 2 1 2 1 1
## Levels: 1 2
```

正如之前所说，R 内部将因子变量当作整数变量来处理，这也就是为什么一个看上去像是字符的东西可以被变成数值。需要注意的是，把因子型转化为其他类型的时候会丢失一定的信息：

1. 因子向量变成字符向量会丢失独特值的信息；
2. 因子向量变成数值型的时候会丢失字面信息，只会保留独特值的编码，即根据独特值排序的正整数。

0.1.3.2.3 向量命名

除了向量自己的名字，我们也可以给向量里的每个元素一个名字。

```
# 先命名向量
# 再命名向量的元素
vec <- c(1, 2, 3)
names(vec) <- c("A", "B", "C")
vec
```

```
## A B C
## 1 2 3
```

```
# 或者
# 创造向量的时候命名向量的元素
vec <- c(A = 1, B = 2, C = 3)
vec
```

```
## A B C
## 1 2 3
```

0.1.3.2.4 访问向量的子集

三种截取子集的符号：[、[[和 $（其中$不能用在基础向量上）。

六种截取向量子集的方法：

1. 正整数：根据元素的序号提取元素；
2. 负整数：根据元素的序号去除元素；
3. 和向量长度一样的逻辑向量：将逻辑向量的元素与向量元素一一对应，TRUE 选择该元素，FALSE去除该元素；
4. Nothing：选择原向量；
5. 零（0）：什么都不选择；
6. 字符向量：选择根据元素名字选择元素。

使用[作为选取符号的示例：

```
vec <- c(a = 1.2, b = 5.6, c = 8.4, d = 9.5)
# 1. 正整数
vec[c(1,3)]
```

```
##   a   c
## 1.2 8.4
```

```
# 2. 负整数
vec[c(-1,-3)]
```

```
##   b   d
## 5.6 9.5
```

```
# 3. 逻辑向量
vec[c(TRUE, FALSE, FALSE, TRUE)]
```

```
##   a   d
## 1.2 9.5
```

```
# 4. Nothing
vec[]
```

```
##   a   b   c   d
## 1.2 5.6 8.4 9.5
```

```
# 5. 零
vec[0]
```

```
## named numeric(0)
```

```
# 6. 字符向量
vec[c("a", "c")]
```

```
##   a   c
## 1.2 8.4
```

[[在向量的场景里只能选择一个元素，而不是像[一样选择一个子集：

```
# 可以
vec[[1]]
```

```
## [1] 1.2
```

```
vec[1]
```

```
##   a
## 1.2
```

```
vec[c(1, 3)]
```

```
##   a   c
## 1.2 8.4
```

```
# 不可以
vec[[c(1, 3)]]
```

```
## Error in vec[[c(1, 3)]]: attempt to select more
    than one element in vectorIndex
```

正是因为这个原因，我们提倡在只选择一个元素的时候多使用[[而不是[。这样在函数产生预期外的行为，选择多余一个元素的时候可以及时被错误信息提醒。

0.1.3.3 特殊数据类型

0.1.3.3.1 日期（date）

R 中有蕴含日期的特殊类型Date，有日期-时间类型的POSIXct和POSIXlt。但在这一节我主要想介绍一下专注于日期处理的包lubridate。

```
library(lubridate)
```

日期的本质实质上只是数字罢了，但是日期也有特殊的计算方式，特殊的进制。比如一个月有可能有 30 天或 31 一天，多少天进一个月也需要相应变化。lubridate包中的年月日ymd函数就是用来帮助解决这个问题的：

```
sevenseven <- ymd("2021-07-07")

sevenseven

## [1] "2021-07-07"

typeof(sevenseven)

## [1] "double"

class(sevenseven)

## [1] "Date"
```

注意这里打印出来的日期是符合阅读习惯的年月日，但是属于Date的 class，又是double的类别，也就意味着可以把这个日期当作一个单纯的数来计算。比如七月七日加上一天就是七月八日：

```
sevenseven + 1

## [1] "2021-07-08"
```

七月七日加上一个月就是八月七日：

```
sevenseven + months(1)

## [1] "2021-08-07"
```

年月日ymd函数所做的只是把输入的字符串自动识别并输出日期格式的数值，只要输入的字符串符合 “年月日” 的类似格式顺序。如果字符串不是年月日的格式，也没关系，lubridate也提供相应的月年日myd，日月年dmy，月日年mdy，日年月dym，甚至是年季yq 的函数。

lubridate的更多用法详见lubridate主页。

0.1.3.3.2 时间序列（ts）

时间序列作为一种有自相关性质的特殊数据类型，在 R 中也是可以分开处理的。制造时间序列的函数叫做ts，也就是 time series 的缩写：

```
xts <- ts(rnorm(12), start = c(2021, 1), frequency =
    4)
xts
```

```
##              Qtr1        Qtr2        Qtr3
    Qtr4
## 2021  1.39373167 -1.63884871 -0.30564944
    -0.69364278
## 2022  0.50782468 -0.05971921 -0.98942648
    -1.90179608
## 2023  1.21372863 -0.09665474  0.76143324
    -0.07260762
```

在这里创造的序列便拥有了时间序列的性质。ts函数的start参数设定了时间序列开始的时间，frequency参数设定了时间序列的周期性。在上面的例子中，我们创造了一个从 2021 年第一季度开始的，具有季节性的时间序列，跨度三年。我们也有相应的函数可以提取这些时间序列的信息：

```
# 起始日期
start(xts)
```

```
## [1] 2021    1
```

```
# 结束日期
end(xts)
```

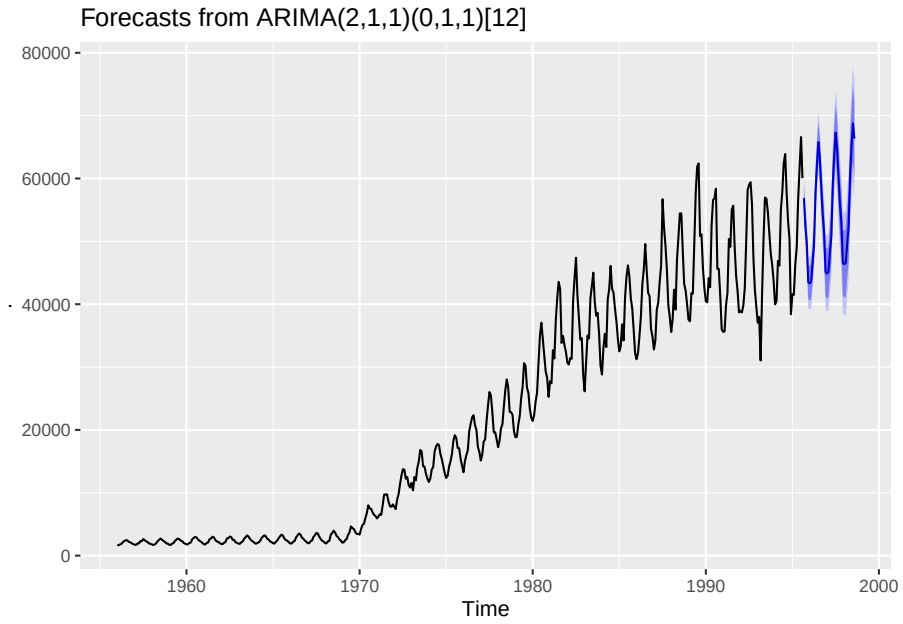
```
## [1] 2023    4
```

```
# 周期性
frequency(xts)
```

```
## [1] 4
```

使用时间序列的好处在于我们可以用一些很简单的命令来使用时间序列的模型，比如使用forecast包来用一个 ARIMA 模型对澳大利亚燃气月生产量进行预测：

```
library(forecast)
gas %>%
  auto.arima() %>%
  forecast(36) %>%
  autoplot()
```



关于时间序列的分析与预测的更多信息可见tidyverts系列包，forecast包等。

0.1.4 多维数据类型

之前我们讨论的数据类型都是一个序列（向量），都是一维的数据。在这章里我们会学习二维甚至多于二维的数据类型。

0.1.4.1 矩阵（matrix）

在 R 中的矩阵和数学概念上的矩阵很相似。在数学概念里，矩阵是一个按照长方阵列排列的数字集合，它有着固定的行数和列数。在 R 里，矩阵是一个按照长方阵列排列的、有着固定行数和列数的、包含同一类型数据的集合。你可以使用函数matrix来创建一个矩阵：

```
matrix(1:9, nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

在这里，第一个参数是矩阵中数据的具体内容。1:9 是 c(1, 2, 3, 4, 5, 6, 7, 8, 9) 的一个缩写，用于创建间隔为 1 的整数序列。

第二个参数告诉 R 这个矩阵应该有多少行。你也可以使用ncol来告诉 R 这个矩阵有多少列。默认状态下，R 会把数值按照从上到下、从左到右的顺序填充在这个固定行数列数的矩阵里。如果你想让 R 先从左到右填充（横向按照行填充），则需要将byrow参数设置为TRUE：

```
matrix(1:9, ncol = 3, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

R 中的矩阵不局限于数值型矩阵，它只要求包含的数据从属于同一类型：如果是数值型，那每一个格子里都是数值型；如果是字符型，所有值都是字符型数据。

```
mat_month <- matrix(month.name, nrow = 4, byrow = TRUE
    )
mat_month
```

```
##      [,1]      [,2]       [,3]
## [1,] "January" "February" "March"
## [2,] "April"   "May"      "June"
## [3,] "July"    "August"   "September"
## [4,] "October" "November" "December"
```

0.1.4.1.1 矩阵命名

对于一个矩阵来说，主要的命名集中于行名rownames和列名colnames：

```
# 你可以用这两个函数去更改行名和列名
rownames(mat_month) <- c("Quarter1", "Quarter2", "
    Quarter3", "Quarter4")
colnames(mat_month) <- c("Month1", "Month2", "Month3")

mat_month

##          Month1     Month2     Month3
## Quarter1 "January"  "February" "March"
## Quarter2 "April"    "May"      "June"
## Quarter3 "July"     "August"   "September"
## Quarter4 "October"  "November" "December"

# 也可以用这两个函数去获取行名和列名
rownames(mat_month)

## [1] "Quarter1" "Quarter2" "Quarter3" "Quarter4"

colnames(mat_month)

## [1] "Month1" "Month2" "Month3"

# 或者用一个函数获取所有维度的名称
dimnames(mat_month)

## [[1]]
## [1] "Quarter1" "Quarter2" "Quarter3" "Quarter4"
##
## [[2]]
## [1] "Month1" "Month2" "Month3"
```

0.1.4.1.2 访问矩阵子集

和在向量里一样，访问矩阵的子集也可以用[或者[[。区别在于矩阵中我们有两个维度，所以需要同时给定两个维度的坐标：

```
# 访问矩阵中第1行第2列格子的元素
mat_month[[1, 2]]
```

```
## [1] "February"
```

```
# 在逗号前不输入数字的时候
# 根据列号截取整列
mat_month[, 2]
```

```
##  Quarter1  Quarter2  Quarter3  Quarter4
## "February"     "May"  "August" "November"
```

```
# 在逗号后不输入数字的时候
# 根据行号截取整行
mat_month[1, ]
```

```
##    Month1    Month2    Month3
## "January" "February"   "March"
```

```
# 如果有行名和列名的话
# 也可以用字符串来截取特定范围
mat_month[["Quarter1", "Month3"]]
```

```
## [1] "March"
```

0.1.4.2 列表（list）

列表是 R 中比较基础的数据类型中最灵活的类型。它和向量或者矩阵不一样，在一个列表中可以储存各种不同的基本数据类型。你既可以存三个数字，也可以把数值型、字符型、逻辑型混合：

```
list(1, 2, 3)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```
list(1, "lol", TRUE)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "lol"
##
## [[3]]
## [1] TRUE
```

列表甚至可以储存列表本身，也就意味着你可以一层套一层地设置列表。夹杂其他各种类型，就可以创造一个庞然大物：

```
stuff <- list(
  list(
    1:12,
    "To be or not to be",
    c(TRUE, FALSE)),
  42,
  list(
    list(
      ymd("2021-07-07"),
      "remembrance"),
    2L+3i)
)
```

```
stuff

## [[1]]
## [[1]][[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## [[1]][[2]]
## [1] "To be or not to be"
##
## [[1]][[3]]
## [1]  TRUE FALSE
##
##
## [[2]]
## [1] 42
##
## [[3]]
## [[3]][[1]]
## [[3]][[1]][[1]]
## [1] "2021-07-07"
##
## [[3]][[1]][[2]]
## [1] "remembrance"
##
##
## [[3]][[2]]
## [1] 2+3i
```

0.1.4.2.1 访问子列表

列表同样可以用中括号来访问子列表。单个中括号[和两个中括号[[的区分在列表中特别重要。简单来说，单个中括号返回的列表元素类型还是列表，双中括号返回的列表元素是它本身的类型。想要返回多个子列表，就只能用单括号了，因为元素本身的类型不允许多个类型在一个序列中保存。

```
# 返回前两个子列表
stuff[1:2]
```

```
## [[1]]
## [[1]][[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## [[1]][[2]]
## [1] "To be or not to be"
##
## [[1]][[3]]
## [1]  TRUE FALSE
##
##
## [[2]]
## [1] 42
```

```
# 返回第一个子列表中的第二个子列表
stuff[[1]][[2]]
```

```
## [1] "To be or not to be"
```

0.1.4.2.2 列表命名

列表的维度，或者说层数可是比矩阵多多了，这也就意味着，列表中可以命名的地方多多了。

```
# 给列表的顶层三个列表命名
names(stuff) <- c("I", "II", "III")

# 给列表的第一个列表里的三个子列表命名
names(stuff[[1]]) <- c("I", "II", "III")
```

如果列表有名字，自然可以用包含名字的字符串获取子列表。

```
# 访问名为“I”的列表中名为“II”的子列表
stuff[["I"]][["II"]]
```

```
## [1] "To be or not to be"
```

之前我们一直没有使用过美元符号$来获取子集，列表提供了一个最佳的展示场景，以下这行代码可以起到于上一行代码一样的效果，而不用加各种括号和引号：

```
stuff$I$II
```

```
## [1] "To be or not to be"
```

0.1.4.3 数据表（data frame 与 tibble）

数据表将是进行数据分析的时候接触的最多的数据类型了。一个数据表（data frame）的本质是一个列表（list），但是采取了矩阵（matrix）的展示形式：

```
df <- data.frame(x = 1:12, y = month.abb, z = month.
    name)
df
```

```
##       x   y         z
## 1     1 Jan   January
## 2     2 Feb  February
## 3     3 Mar     March
## 4     4 Apr     April
## 5     5 May       May
## 6     6 Jun      June
## 7     7 Jul      July
## 8     8 Aug    August
## 9     9 Sep September
## 10   10 Oct   October
## 11   11 Nov  November
## 12   12 Dec  December
```

数据表的每一列是一个子列表。将几个长度相同的子列表并排放在一起，就组成了一个长方形的矩阵形式。这种特殊的处理使得数据表包含了两种数

据形式的优势。列与列之间可以使用不同的基础数据类型，也就是说一列的数据是数值型的数据，下一列数据可以是字符型的数据。长方形的形状保证了列与列之间的数值是一一对应的，每一行都是一个观察量。这很符合日常会遇到的数据的形式。

tibble是tidyverse系列包中的tibble包提供的一种数据形式。使用tibble比较明显的好处是，当你把tibble打印在控制台里的时候，它有一个更干净直观的打印方式。与 data frame 试图打印所有的行、一股脑把所有信息扔给你不同，tibble 默认只会打印前几行给你一个数据长什么样的感觉，还会告诉你每一列的数据是什么类型的：

```
tb <- tibble(a = 1:100, b = 101:200)
tb
```

```
## # A tibble: 100 x 2
##          a     b
##    <int> <int>
##  1     1   101
##  2     2   102
##  3     3   103
##  4     4   104
##  5     5   105
##  6     6   106
##  7     7   107
##  8     8   108
##  9     9   109
## 10    10   110
## # ... with 90 more rows
```

除了看起来好看以外，tibble在原始数据表的基础上保留了有用的功能，去除了多余的功能。它干得更少，比如它不会自发修改变量类型或变量名字，也不会做部分匹配；同时它抱怨得更多，比如当一个变量不存在的时候就会触发错误信息。这样用户就能及早发现错误，不会等到代码堆成小　山。

0.1.4.3.1　访问数据表内容

既然看上去像矩阵，听起来像列表，那就应该可以用适用于矩阵和列表的方法访问数据表元素。事实上也的确是这样：

```
# 访问数据表名为x的列
df[["x"]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
df$x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
# 访问第一行第二个数值
df[1, 2]
```

```
## [1] "Jan"
```

```
# 访问tibble第2列
tb[, 2]
```

```
## # A tibble: 100 x 1
##        b
##    <int>
##  1   101
##  2   102
##  3   103
##  4   104
##  5   105
##  6   106
##  7   107
##  8   108
##  9   109
## 10   110
## # ... with 90 more rows
```

```
# 访问tibble第1行第2列的数值
tb[1, 2]
```

```
## # A tibble: 1 x 1
##       b
##   <int>
## 1   101
```

```
tb$a
#太长了还是不显示了吧
```

tibble 的另一个特性是其访问的子列表也是tibble类型的数据表，即使是用单引号返回一个格子的元素。

关于tibble更多信息，详见tibble主页。

0.1.5 读写数据

这一章我们主要讨论根据不同数据保存方式区分的读写数据的方法。

0.1.5.1 内置数据集

R 本身和一些 R 包都会有内置的数据集。使用data命令来查看、使用可用数据集。

```
# 查看R本身自带的数据集
data()

# 查看某一R包自带的数据集
data(package = "dplyr")

# 载入AirPassengers数据集
data("AirPassengers")

glimpse(AirPassengers)

##  Time-Series [1:144] from 1949 to 1961: 112 118 132
    129 121 135 148 148 136 119 ...
```

0.1.5.2 表格类型数据（csv, excel)

h1n1 流感问卷数据储存在名为 “h1n1_flu.csv” 的文件中，我们会在下一篇《数据清洗与准备》中用到。假设 “h1n1_flu” 有不同的储存类型，我们列举一些读写数据表类型数据的方法。

```
# 读取csv文件
library(readr)
h1n1_flu <- read_csv("h1n1_flu.csv")
# 保存csv文件
write_csv(h1n1_flu, "h1n1_flu.csv")


# 读取excel文件
library(readxl)
# 自动识别文件后缀
h1n1_flu <- read_excel("h1n1_flu.xls")
# 读取xls文件
h1n1_flu <- read_xls("h1n1_flu.xls")
# 读取xlsx文件
h1n1_flu <- read_xlsx("h1n1_flu.xlsx")
```

不建议在 R 中直接编辑 excel 文件，csv 文件应该满足日常所需了。如果有编辑 excel 文件的需求，可以看看openxlsx包。

0.1.5.3 R 的专属类型数据（RData, rds）

有一些数据存储方式是 R 中独有的。我们在这里讨论两类。一类是 rds 文件，一类是 RData 文件。

1. rds 文件储存一个 R 中的对象。这个对象不一定是四四方方的数据表，而可以是任何形式，包括复杂的列表等。因为他储存的是一个对象，所以读取的时候也是读取出来一个对象，需要被保存在一个名字下。
2. RData 储存的是一个或多个、任意结构的、带有自己名字的对象。读取的时候会将储存的对象直接载入当前的环境中，使用的是对象自己

的名字，所以不需要再额外起名字。

```
# 读取
h1n1_flu <- read_rds("h1n1_flu.rds")
# 存储
write_rds(h1n1_flu, "h1n1_flu.rds")

# 读取
load("h1n1_flu.RData")
# 存储
save(h1n1_flu, file = "h1n1_flu.RData")
```

0.1.5.4 其他软件（SPSS, Stata, SAS）

R 也可以直接读取其他软件的数据类型。这里列举使用haven包读写 SPSS 的 sav 和 zsav、Stata 的 dta、SAS 的 sas7bdat 和 sas7bcat。

```
library(haven)

# SPSS
read_spss()
write_spss()

# Stata
read_dta()
write_dta()

# SAS
read_sas()
write_sas()
```

0.1.6 练习题

0.1.6.1 了解数据集

请使用之前读取的h1n1_flu完成以下任务。

0.1.6.1.1 常用数据探查函数

请尝试使用以下常用的数据探查函数，挑出两个你最喜欢的描述他们的功能。别忘了可以用?fun查看帮助文档。

```
glimpse(h1n1_flu)
str(h1n1_flu)
head(h1n1_flu)
tail(h1n1_flu)
View(h1n1_flu)
summary(h1n1_flu)
nrow(h1n1_flu)
length(h1n1_flu$sex)
class(h1n1_flu$sex)
summary(h1n1_flu)
table(h1n1_flu$sex)
```

0.1.6.1.2 分组计算统计量

```
h1n1_flu %>%
  group_by(sex, employment_status) %>%
  summarise(n())
```

```
## # A tibble: 8 x 3
## # Groups:   sex [2]
##    sex    employment_status  `n()`
##   <chr>  <chr>              <int>
## 1 Female Employed            7416
## 2 Female Not in Labor Force  6918
```

```
## 3 Female Unemployed              735
## 4 Female <NA>                    789
## 5 Male   Employed                6144
## 6 Male   Not in Labor Force      3313
## 7 Male   Unemployed              718
## 8 Male   <NA>                    674
```

请问上边这几行代码在计算什么？你可不可以使用同样的方法计算一些其他的统计量？别忘了看看帮助文档?summarise。

0.1.6.2 创造数据集

我们说过数据表的本质是将列表排列在一起，所以数据表就会有列表的性质。而我们又知道列表可以包含任何类型的数据，无论是单个的数值或者是向量、矩阵等。

1. 请你创造一个数据表，其中的某一列变量的每一个格子包含的不再是常规的单个数值或者字符串，而是一个向量或者矩阵等多维的数据类型；
2. 请你描述一个可以在数据分析时运用此类特性的使用场景。

本章作者

Fin

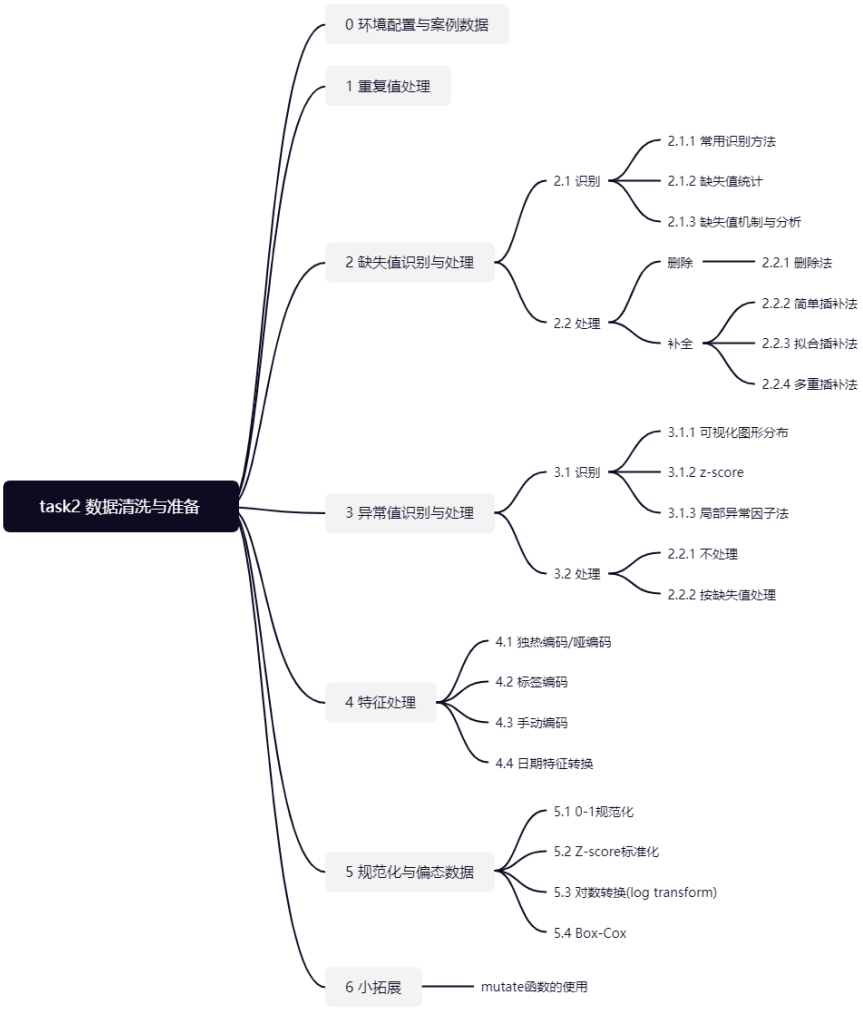
https://yangzhuoranyang.com

关于 Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以 “for the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力

人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：

0.2 数据清洗与准备



Task 02 共计 6 个知识点，预计需学习 5~8 小时，请安排好学习任务。

环境配置

library(mlbench) # 将会使用到包中的*BostonHousing*数据集
library(funModeling) # 探索性数据分析工具包，本节内容中将会使用到它的*status()*函数，打印整体数据质量

library(tidyverse) # 数据转化工具包，本节内容中将会使用它包含的$dplyr$中的管道函数 %>%

library(VIM) # 缺失值可视化工具包，本节内容中将会使用到它的$aggr()$函数

library(mice) # 缺失值处理工具包，本节内容会使用它来进行多重插补

library(Rlof) # 用于*LOF*异常值检测方法，本节内容将会使用到它的$lof()$函数

library(fastDummies) # 用于生成*dummy*的包，本节内容将会使用到它的$dummy_cols()$函数

library(sjmisc) # 用于生成*dummy*的包，本节内容将会使用到它的$to_dummy()$函数

library(MASS) # 基于此包进行$box-cox$转换

library(dlookr) # 本节内容将会使用到它的$transform()$函数

案例数据

本节内容将会使用到两个数据集。

数据集 1 h1n1 流感问卷数据集

0.2.0.0.1 * 数据说明

目前提供的数据集来自关于 h1n1 流感调查问卷的部分内容，可以从这个网站上看到具体字段的详细说明：https://www.drivendata.org/competitions/66/flu-shot-learning/page/211/

数据集包含 26,707 个受访者数据，共有 32 个特征 +1 个标签（是否接种 h1n1 疫苗）。

0.2.0.0.2 * 加载并查看部分数据

首先加载数据，了解数据集大小。

```
h1n1_data <- read.csv("./datasets/h1n1_flu.csv",
    header = TRUE)
dim(h1n1_data)
```

```
## [1] 26707    33
```

注：为了简化本章的示例，我们在这 32 个特征中，筛选出了 10 个特征，作为一个子集，来学习如何使用 R 做数据清洗与准备。如有兴趣，可以把下面这块筛选去掉，自己用完整数据集做一次探索。

```
h1n1_data <- h1n1_data[, c(1, 3, 11, 12, 15, 16, 19,
    20, 22, 23, 33)]
head(h1n1_data)
```

```
##   respondent_id h1n1_knowledge doctor_recc_h1n1
    chronic_med_condition
## 1             0               0                0
                        0
## 2             1               2                0
                        0
## 3             2               1               NA
                        1
## 4             3               1                0
                        1
## 5             4               1                0
                        0
## 6             5               1                0
                        0
##   health_insurance opinion_h1n1_vacc_effective
    age_group         education
## 1                1                          3 55 -
     64 Years        < 12 Years
## 2                1                          5 35 -
     44 Years          12 Years
```

```
## 3                 NA                                3 18 -
     34 Years  College Graduate
## 4                 NA                                3
    65+ Years           12 Years
## 5                 NA                                3 45 -
     54 Years       Some College
## 6                 NA                                5
    65+ Years           12 Years
##       sex              income_poverty h1n1_vaccine
## 1 Female                Below Poverty             0
## 2    Male                Below Poverty             0
## 3    Male <= $75,000, Above Poverty                 0
## 4 Female                Below Poverty             0
## 5 Female <= $75,000, Above Poverty                 0
## 6    Male <= $75,000, Above Poverty                 0
```

数据集 2 波士顿房价数据集

0.2.0.0.3 * 数据说明

数据集来自mlbench包，请提前装好。数据字段说明可从网址查看：https://blog.csdn.net/weixin_46027193/article/details/112238597

数据集包含 506 条房价信息，共有 13 个特征 +1 个预测字段（房屋价格）。

0.2.0.0.4 * 加载并查看部分数据

```
data(BostonHousing)
dim(BostonHousing)
```

```
## [1] 506  14
```

```
head(BostonHousing)
```

```
##      crim zn indus chas   nox    rm  age    dis rad
    tax ptratio      b lstat
```

```
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1
    296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2
    242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2
    242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3
    222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3
    222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3
    222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

0.2.1 重复值处理

在某些情况下，我们需要对数据进行去重处理。unique()函数可以对数据进行整体去重，distinct ()函数可以针对某些列去重。

```
# 整体去重
h1n1_data_de_dup1 <- unique(h1n1_data)

# 指定根据列respondent_id,h1n1_knowledge去重，并保留所有列
h1n1_data_de_dup2 <- distinct(h1n1_data, respondent_id, h1n1_knowledge, .keep_all = T)
```

0.2.2 缺失值识别与处理

现实环境中，由于数据来源及搜集过程，可能有各种不规范，导致数据往往会存在缺失。缺失值识别与处理，无论是在统计还是数据管理中，往往是数据清洗的第一步。

0.2.2.1 缺失值识别

0.2.2.1.1 常用识别方法

在 R 语言中，惯用会把缺失值表示为 NA，一般可使用is.na(a)，!complete.cases(a)来识别a是否为缺失值。

```
# 假设定义的一个变量中存在缺失值
y <- c(1, 2, 3, NA)

# 用is.na在识别是否为缺失值
is.na(y)

## [1] FALSE FALSE FALSE  TRUE

# 用!complete.cases()在识别是否为缺失值
!complete.cases(y)

## [1] FALSE FALSE FALSE  TRUE
```

0.2.2.1.2 缺失值统计

统计缺失值总数。

```
# 数据集中总缺失数据量
sum(is.na(h1n1_data))

## [1] 15912

# 数据集中某一列缺失数据量
sum(is.na(h1n1_data["h1n1_knowledge"]))
```

```
## [1] 116
```

如果想按行或按列统计，可以写函数。

```
pMiss <- function(x) {
  sum(is.na(x)) / length(x) * 100
}
apply(h1n1_data, 2, pMiss) # 按列统计缺失比率%
```

```
##                 respondent_id
    h1n1_knowledge
##                     0.0000000
    0.4343431
##             doctor_recc_h1n1
    chronic_med_condition
##                     8.0877673
    3.6357509
##             health_insurance
    opinion_h1n1_vacc_effective
##                    45.9579885
    1.4640356
##                     age_group
    education
##                     0.0000000
    0.0000000
##                           sex
    income_poverty
##                     0.0000000
    0.0000000
##                  h1n1_vaccine
##                     0.0000000
```

```
# apply(h1n1_data,1,pMiss) #按行统计缺失比率%
```

或调用一些现成的包。比如，我们可以使用funModeling包中的status()函数，

直接观测案例数据中包含的 0 值，缺失值（NA），在每个特征中的分布情况。以 h1n1 flu 数据集为例：

```
data_quality <- status(h1n1_data)
data_quality %>% mutate(across(where(is.numeric), ~
    round(., 3))) # 保留4位小数
```

```
##
    variable q_zeros p_zeros  q_na
## respondent_id
    respondent_id        1    0.000      0
## h1n1_knowledge
    h1n1_knowledge     2506    0.094    116
## doctor_recc_h1n1
    doctor_recc_h1n1    19139    0.717   2160
## chronic_med_condition
    chronic_med_condition    18446    0.691    971
## health_insurance
    health_insurance     1736    0.065  12274
## opinion_h1n1_vacc_effective
    opinion_h1n1_vacc_effective        0    0.000    391
## age_group
    age_group        0    0.000      0
## education
    education        0    0.000      0
## sex
                                                        sex
           0    0.000      0
## income_poverty
    income_poverty        0    0.000      0
## h1n1_vaccine
    h1n1_vaccine    21033    0.788      0
##                                  p_na q_inf p_inf
    type unique
## respondent_id                  0.000      0      0
```

```
    integer   26707
## h1n1_knowledge                0.004      0      0
    numeric       3
## doctor_recc_h1n1              0.081      0      0
    numeric       2
## chronic_med_condition         0.036      0      0
    numeric       2
## health_insurance              0.460      0      0
    numeric       2
## opinion_h1n1_vacc_effective   0.015      0      0
    numeric       5
## age_group                     0.000      0      0
    character       5
## education                     0.000      0      0
    character       5
## sex                           0.000      0      0
    character       2
## income_poverty                0.000      0      0
    character       4
## h1n1_vaccine                  0.000      0      0
    integer       2
```

结合案例数据 h1n1 flu 来看，存在缺失值的有 5 个特征字段。

```
missing_Value <- data_quality[which(data_quality$p_na
    > 0), ]
missing_Value$variable
```

```
## [1] "h1n1_knowledge"               "doctor_recc_h1n1
    "
## [3] "chronic_med_condition"        "health_insurance
    "
## [5] "opinion_h1n1_vacc_effective"
```

0.2.2.1.3 缺失值机制与分析

统计学家通常将缺失数据分为 3 类，为了更好的处理缺失值，我们可以基于缺失值机制来识别以下 3 种缺失模式：

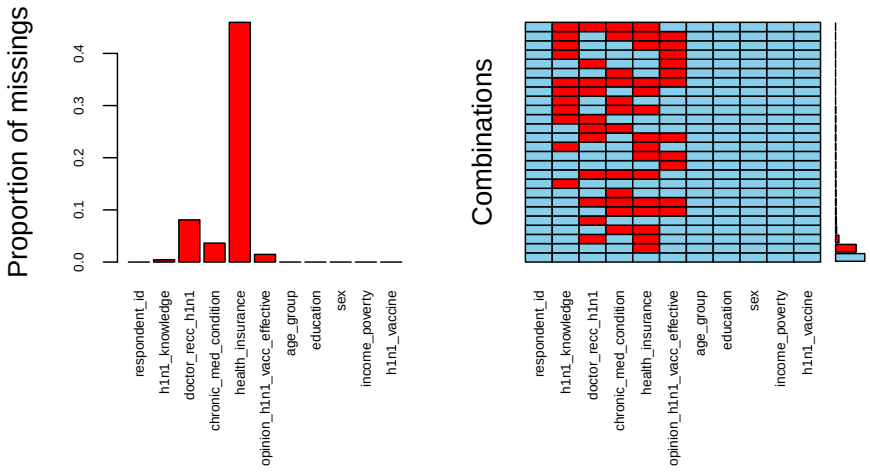
- MCAR（完全随机缺失）：如果数据的缺失与任何值（观察或缺失）之间没有关系，则为 MCAR。
- MAR（随机缺失）：考虑 MAR 与 MCAR 有何不同，如果缺失和观测值之间存在系统关系，则为 MAR。例如-男性比女性更容易告诉自己的体重，因此体重就是 MAR。“ Weight” 变量的缺失取决于变量 “Sex” 的观测值。
- MNAR（非随机缺失）：若缺失数据不属于 MCAR 和 MAR，数据的缺失依赖于不完全变量本身，则数据为非随机缺失。例如，抑郁程度高的人更不容易填写抑郁调查问卷。

MNAR 是最复杂的情况，处理 MNAR 的策略是找到更多有关缺失原因的数据，或者执行假设分析，查看结果在各种情况下的敏感程度。大部分处理缺失数据的方法都假定数据是 MCAR 或 MAR，此时，可以忽略缺失数据的生成机制，在替换或删除缺失数据后，直接对感兴趣的关系进行建模。

以下介绍几种可视化分析缺失数据关联的方法：

我们用VIM包里的aggr()函数，直观看一下具体的缺失情况。

```
aggr(h1n1_data, cex.axis = .6, oma = c(9, 5, 5, 1)) #
    cex.axis调整轴字体大小，oma调整外边框大小
```



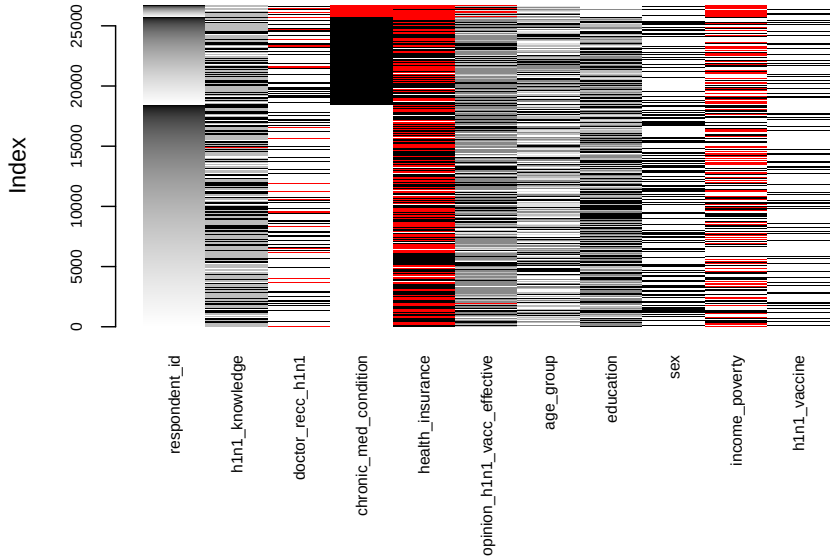
通过用VIM包里的矩阵图matrixplot()函数，可以检查某些变量的缺失值模式是否与其他变量的真实值有关联。矩阵图中，观测数据以黑白色阶显示（颜色越深，数值越高），缺失值会被标记为红色。我们对某一个存在缺失值的变量进行排序，来找寻含缺失值变量与其他变量的关系。

在此案例中，我们按照health_insurance进行分组排序。可以看到是否有慢性病chronic_med_condition的缺失，与opinion_h1n1_vacc_effective的缺失相对较集中。除此之外，也可以看到有慢性病的人年龄普遍较大。

```
# 先简单处理一下一些类别变量的顺序
h1n1_data_matplt <- h1n1_data
h1n1_data_matplt$age_group <- factor(h1n1_data_matplt$
    age_group)
h1n1_data_matplt$education <- factor(h1n1_data_matplt$
    education, levels = c("", "< 12 Years", "12 Years",
     "Some College", "College Graduate"))
h1n1_data_matplt$sex <- factor(h1n1_data_matplt$sex)
h1n1_data_matplt$income_poverty <- factor(h1n1_data_
    matplt$income_poverty, levels = c("18 - 34 Years",
    "<= $75,000, Above Poverty", "> $75,000"))
```

```
# levels(h1n1_data_matplt$age_group) # 查看顺序

# 矩阵图可视化
par(mar = c(9, 4.1, 2.1, 2.1)) # x轴标签太长，调用par()函数调整外边框的大小
matrixplot(h1n1_data_matplt, sortby = "chronic_med_condition", cex.axis = 0.7) # cex.axis为调整坐标轴字体大小
```



用相关性探索缺失值。首先生成一个影子矩阵，用指示变量替代数据集中的数据（1 表示缺失，0 表示存在）。

```
shadow_mat <- as.data.frame(abs(is.na(h1n1_data[, -1])))
head(shadow_mat)
```

```
##   h1n1_knowledge doctor_recc_h1n1
    chronic_med_condition health_insurance
## 1                  0                 0
                       0                 0
```

```
## 2                0                0
                         0                0
## 3                0                1
                         0                1
## 4                0                0
                         0                1
## 5                0                0
                         0                1
## 6                0                0
                         0                1
##   opinion_h1n1_vacc_effective age_group education
    sex income_poverty
## 1                           0         0         0
     0              0
## 2                           0         0         0
     0              0
## 3                           0         0         0
     0              0
## 4                           0         0         0
     0              0
## 5                           0         0         0
     0              0
## 6                           0         0         0
     0              0
##   h1n1_vaccine
## 1            0
## 2            0
## 3            0
## 4            0
## 5            0
## 6            0
```

```
# 可提取含缺失值的变量
shadow_mat <- shadow_mat[which(apply(shadow_mat, 2,
```

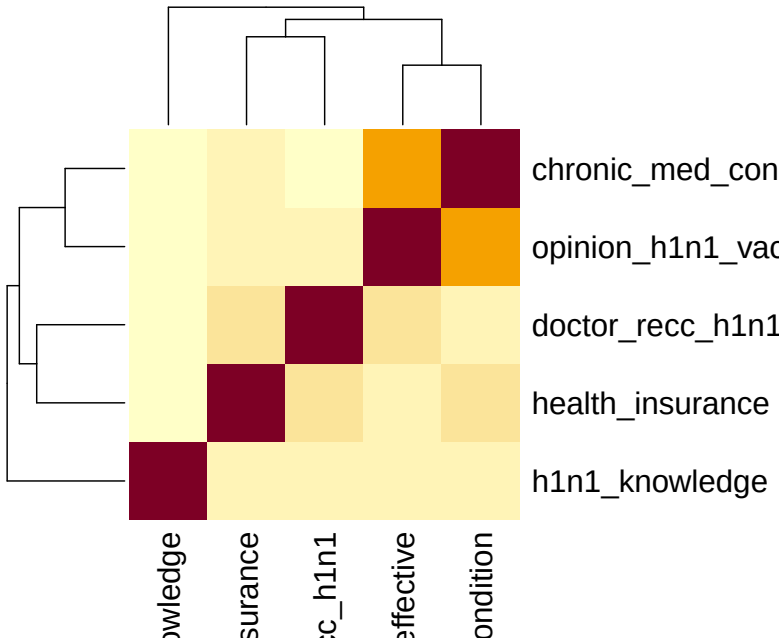
```
    sum) > 0)]

# 计算相关系数
cor(shadow_mat)
```

```
##                                 h1n1_knowledge
    doctor_recc_h1n1
## h1n1_knowledge                     1.00000000
    0.00546769
## doctor_recc_h1n1                   0.00546769
    1.00000000
## chronic_med_condition              0.02367388
    0.09572429
## health_insurance                  -0.01292316
    0.22136525
## opinion_h1n1_vacc_effective        0.01565202
    0.14793032
##                                 chronic_med_condition
    health_insurance
## h1n1_knowledge                                0.02367388
         -0.01292316
## doctor_recc_h1n1                              0.09572429
          0.22136525
## chronic_med_condition                         1.00000000
          0.15724626
## health_insurance                              0.15724626
          1.00000000
## opinion_h1n1_vacc_effective                   0.47431031
          0.10403005
##
    opinion_h1n1_vacc_effective
## h1n1_knowledge
    0.01565202
## doctor_recc_h1n1
```

```
    0.14793032
## chronic_med_condition
    0.47431031
## health_insurance
    0.10403005
## opinion_h1n1_vacc_effective
    1.00000000
```

```
# 相关系数热力图
heatmap(cor(shadow_mat))
```



根据缺失相关性矩阵,opinion_h1n1_vacc_effective 与 chronic_med_condition 缺失相关性较大。

综上，在案例中，变量之间的存在部分相关性，考虑为 MAR。

其他数据缺失关系分析，可参考附录数据的预处理基础。

0.2.2.2 缺失值处理

缺失值一般有三种方式：

- 将缺失值作为变量值使用。比如在民意调查中，当选民不投票时，可以将缺失值处理为 “无法确定”。
- 删除数据。主要有删除样本值和删除特征值。但可能会损失掉一些有用信息。
- 插补法。如均值/中位数/同类均值插补（数值变量），众数插补（类别变量），手动插补 (根据主观理解)，多重插补等。

以下我们主要介绍删除法和插补法：

0.2.2.2.1 删除法

行删除，可以直接用complete.cases()或na.omit()来过滤掉数据集中所有缺失行。

```
h1n1_data_row_del1 <- h1n1_data[!complete.cases(h1n1_
    data), ]
h1n1_data_row_del2 <- na.omit(h1n1_data)
```

列删除，一般对于缺失率极高又没有太大作用的特征值，我们直接删除，如可以用dataset[,−5]去掉第五列，或subset(dataset, select = −c(col1, col2))去掉列 col1 和列 col2。

比如，我们把health_insurance变量删除。

```
h1n1_data_col_del1 <- subset(h1n1_data, select = -c(
    health_insurance))
```

0.2.2.2.2 简单插补法

注意在空值插补的时候，要区分类别变量与数值变量，均值插补不适用于类别变量。我们这里随机选择了一个变量演示impute()函数用法，在实际插补的时候，请大家根据情况进行选择。

```
h1n1_data_sim_imp <- h1n1_data
h1n1_data_sim_imp$h1n1_knowledge <- impute(h1n1_data_
    sim_imp$h1n1_knowledge, 1) # 填充特定值
```

```
h1n1_data_sim_imp$h1n1_knowledge <- impute(h1n1_data_
    sim_imp$h1n1_knowledge, median) # 插补中位数
h1n1_data_sim_imp$h1n1_knowledge <- impute(h1n1_data_
    sim_imp$h1n1_knowledge, mean) # 插补均值
```

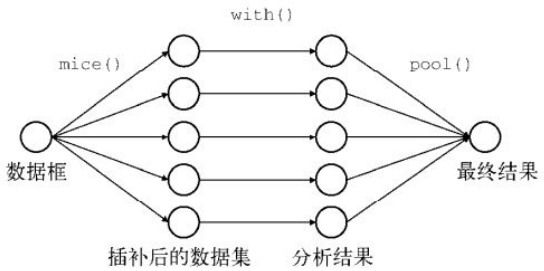
0.2.2.2.3 拟合插补法

利用有监督的机器学习方法，比如回归、最邻近、随机森林、支持向量机等模型，对缺失值作预测。

0.2.2.2.4 多重插补法

多重插补（MI）是一种基于重复模拟的处理缺失值的方法。其思想来源于贝叶斯估计，认为待插补的值是随机的，它的值来自于已观测到的值。具体实践上通常是估计出待插补的值，然后再加上不同的噪声，形成多组可选插补值（通常是 3 到 10 个）。根据某种选择依据，选取最合适的插补值。与单个插补（例如均值）相比，创建多个插补可解决缺失值的不确定性。R 中可利用Amelia、mice和mi包来执行这些操作。

本节中，我们将用案例介绍 mice 包（通过链式方程进行的多元插补）提供的方法。使用 mice 生成 m 个完整的插补数据集。然后利用with−pool的方法来评估选择哪一个数据集。首先使用with()函数依次对每个完整数据集应用统计模型如 lm，glm 等，用summary()输出数据集检验，看某数据集是否合格。接下来pool()函数把 5 个回归模型汇总，用summary()输出汇总数据集检验，查看整体插补方法是否合格。检验结果分析可参考附录mice检验结果解释




```
# 先处理下数据，把数据集中一些类别变量转换回来
```

```
# imp是一个包含m个插补数据集的列表对象，同时还含有完成
    插补过程的信息。
# 参数m的默认值为5，这里我们将m设为4，生成4个无缺失数
    据集
# 参数method，对于每个变量的拟合，可以指定所用的拟合方
    法,method传入的参数可以是一个具体方法，也可以为不同
    列指定具体方法，具体方法选择可参考附录mice使用文
    档。这里我们使用默认值。
imp <- mice(h1n1_data, m = 4, seed = 122, printFlag =
    FALSE)

# 查看变量h1n1_knowledge在几个插补数据集中的插补结果
# imp$imp$h1n1_knowledge

# 查看每个变量所用的插补方法
# imp$method

# 设定应用于m个插补数据集的统计分析方法。方法包括做线
    性回归模型的lm()函数、做广义线性模型的glm()函数、做
    广义可加模型的gam()，做负二项模型的nbrm()函数
fit <- with(imp, lm(h1n1_vaccine ~ h1n1_knowledge +
    doctor_recc_h1n1 + chronic_med_condition + health_
    insurance + opinion_h1n1_vacc_effective))

# 输出每个数据集检验
print.data.frame(summary(fit), digits = 4)

##                          term estimate std.error
    statistic    p.value  nobs
## 1                 (Intercept) -0.30492  0.010809
      -28.209 1.557e-172 26707
## 2               h1n1_knowledge  0.03645  0.003661
       9.956  2.596e-23 26707
## 3             doctor_recc_h1n1  0.34604  0.005568
```

```
       62.147   0.000e+00 26707
## 4         chronic_med_condition   0.03033   0.005015
       6.048   1.485e-09 26707
## 5              health_insurance   0.07826   0.006754
       11.587   5.706e-31 26707
## 6   opinion_h1n1_vacc_effective   0.08317   0.002245
       37.054 4.116e-293 26707
## 7                   (Intercept) -0.30718   0.010901
      -28.179 3.509e-172 26707
## 8                 h1n1_knowledge   0.03689   0.003683
       10.016   1.429e-23 26707
## 9              doctor_recc_h1n1   0.33876   0.005563
       60.893   0.000e+00 26707
## 10        chronic_med_condition   0.02972   0.005031
       5.907   3.521e-09 26707
## 11             health_insurance   0.07776   0.006957
       11.178   6.028e-29 26707
## 12  opinion_h1n1_vacc_effective   0.08385   0.002258
       37.128 2.986e-294 26707
## 13                  (Intercept) -0.30981   0.010830
      -28.607 2.603e-177 26707
## 14                h1n1_knowledge   0.03666   0.003679
       9.965   2.386e-23 26707
## 15             doctor_recc_h1n1   0.33489   0.005557
       60.262   0.000e+00 26707
## 16        chronic_med_condition   0.02948   0.005035
       5.855   4.814e-09 26707
## 17             health_insurance   0.08090   0.006742
       12.000   4.334e-33 26707
## 18  opinion_h1n1_vacc_effective   0.08415   0.002258
       37.272 1.851e-296 26707
## 19                  (Intercept) -0.30608   0.010910
      -28.055 1.047e-170 26707
```

```
## 20                h1n1_knowledge   0.03702   0.003685
       10.046   1.056e-23 26707
## 21             doctor_recc_h1n1   0.33370   0.005564
       59.970   0.000e+00 26707
## 22        chronic_med_condition   0.02969   0.005040
        5.891   3.877e-09 26707
## 23             health_insurance   0.07557   0.006896
       10.959   6.877e-28 26707
## 24 opinion_h1n1_vacc_effective   0.08423   0.002259
       37.278  1.490e-296 26707

# 包含m个统计分析平均结果的列表对象
pooled <- pool(fit)

# 这是一个总体评估结果
pooled

## Class: mipo    m = 4
##                          term m    estimate
           ubar             b
## 1                 (Intercept) 4 -0.30699871
   1.179991e-04 4.368721e-06
## 2               h1n1_knowledge 4  0.03675472
   1.352049e-05 6.410610e-08
## 3            doctor_recc_h1n1 4  0.33834805
   3.094965e-05 3.095473e-05
## 4       chronic_med_condition 4  0.02980518
   2.530162e-05 1.342220e-07
## 5            health_insurance 4  0.07812323
   4.675346e-05 4.779575e-06
## 6 opinion_h1n1_vacc_effective 4  0.08385005
   5.085296e-06 2.294296e-07
##               t dfcom            df          riv
   lambda          fmi
```

```
## 1 1.234600e-04 26701   1446.448456 0.046279160
    0.044232134 0.045550936
## 2 1.360062e-05 26701 20305.470346 0.005926755
    0.005891836 0.005989737
## 3 6.964306e-05 26701      9.710629 1.250205046
    0.555596055 0.625522404
## 4 2.546940e-05 26701 19168.975299 0.006631097
    0.006587415 0.006691047
## 5 5.272792e-05 26701    231.386715 0.127786668
    0.113307482 0.120873547
## 6 5.372083e-06 26701   1010.568516 0.056395341
    0.053384693 0.055252579

# 这里修改action的参数（范围1–m），选择一个数据集作为
    我们已填充完成的数据集
h1n1_data_complete <- complete(imp, action = 2)
```

0.2.3 异常值识别与处理

0.2.3.1 异常值识别

本节的异常值指离群点。为了让数据统计或数据建模更加准确，我们通常会识别并对处理一些离群点。有些模型会对异常值较敏感，参考附录什么样的模型对缺失值更敏感?。总的来说，有几种常用方法，包括可视化图形分布识别（箱线图）、z-score 识别、局部异常因子法（LOF 法）、聚类法等。

我们这里用波士顿房价数据集来演示一下异常值识别的处理过程。

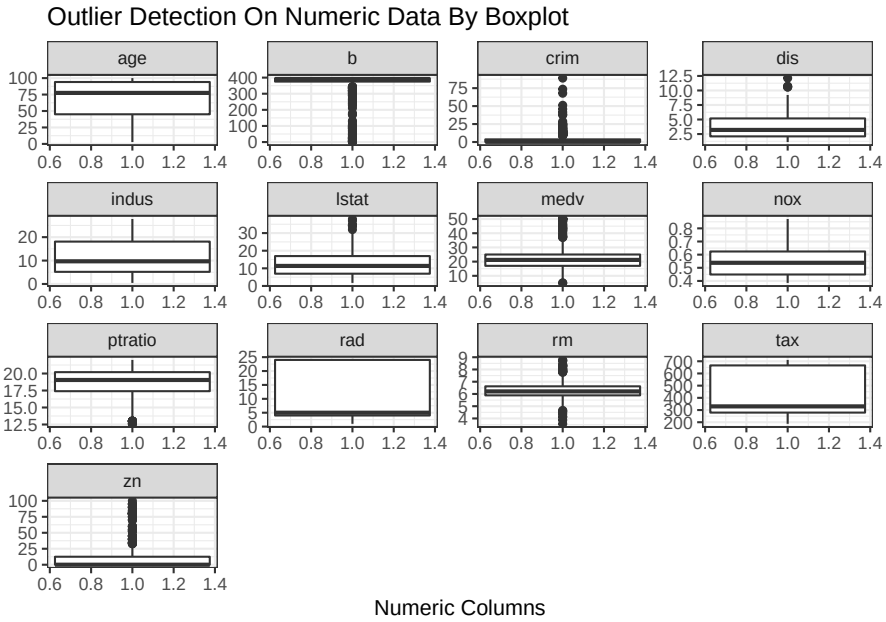
0.2.3.2 可视化图形分布

首先是可视化图形分布识别，将数值型变量筛选出来，用 boxlpot 看看分布。

```
# 提取数值字段
nums <- unlist(lapply(BostonHousing, is.numeric))
nums_data <- BostonHousing[, nums]
```

```
# 数据变形
nums_data.new <- nums_data %>%
  as.data.frame() %>%
  mutate(Cell = rownames(.)) %>%
  gather(., key = colname, value = "value", -Cell)

# 用ggplot画出箱线图
ggplot(data = nums_data.new, aes(x = colname, y =
    value)) +
  geom_boxplot(aes(1)) +
  facet_wrap(~colname, scales = "free") +
  theme_grey() +
  labs(title = "Outlier Detection On Numeric Data By
      Boxplot", x = "Numeric Columns", y = "") +
  theme(legend.position = "top") +
  theme_bw()
```



通过可视化分布，可以选择剔除一些不合理的离群值，比如在数据集中将

dis>10.0 的数据剔除。

0.2.3.3 z-score

z-score 是一种一维或低维特征空间中参数异常检测方法。它假定数据是高斯分布,异常值是分布尾部的数据点,因此远离数据的平均值。一般将 z-score 低于-3 或高于 3 的数据看成是异常值。

```
# 定义一个识别异常点的函数，x是输入数据（matrix或df），
    zs是异常临界值，z-score超过zs的被识别为异常点
outliers <- function(x, zs) {
  temp <- abs(apply(x, 1, scale))
  return(x[temp > zs])
}
# 打印出z-score<3的值
outliers(nums_data, 3)
```

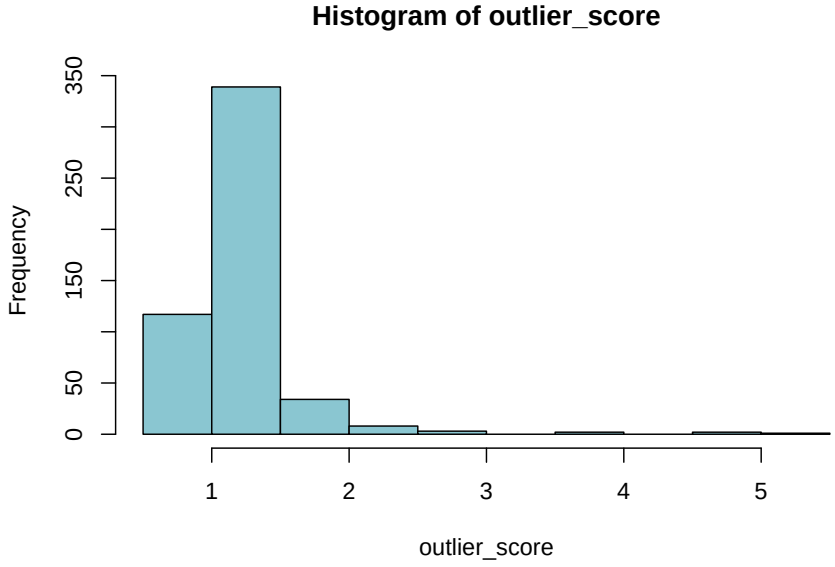
```
##  [1]   7.380   0.700   0.573   5.889  17.400
    20.200 392.400 396.900 396.900
## [10] 393.680 396.900 368.570 396.900 377.730
    375.330 396.900 391.980 100.630
## [19] 388.520 255.230 374.680 392.680 395.770
    12.430  11.280  27.710  10.210
## [28]   6.860   9.880   9.620   4.210  13.000
    25.410  16.900  29.550   6.360
## [37]   4.850   4.700   4.610  13.270   2.960
    24.560  19.370  14.100  14.330
## [46]  22.800  33.400
```

0.2.3.4 局部异常因子法

局部异常因子法 (LOF)，是一种无监督的离群检测方法，是基于密度的离群点检测方法中一个比较有代表性的算法。适用于在中等高维数据集上执行异常值检测。

```
# k是计算局部异常因子所需要判断异常点周围的点的个数
outlier_score <- lof(data = nums_data, k = 5)

# 绘制异常值得分的直方分布图
hist(outlier_score, col = "#8ac6d1")
```




```
# 排序，挑出得分排前五的数据（找到索引）作为异常值
names(outlier_score) <- 1:nrow(nums_data)
sort(outlier_score, decreasing = TRUE)[1:5]
```

```
##       489       493       381       492       406
## 5.133201  4.534088  4.529170  3.732775  3.559666
```

0.2.3.5 异常值处理

首先需要确定是否是真的异常值，有些值虽然离群，但其实并不是异常值，处理掉反而会影响后续任务的准确性。如果确定需要处理，可以参考缺失值的处理方式进行处理。

0.2.4 特征编码

为什么要进行特征编码？我们拿到的原始数据中，一般会有一些类别变量，但是在统计或机器学习中，我们通常需要把类别变量转化为数值型变量，才能应用于一些方法中。

0.2.4.1 独热编码/哑编码

One-hot encoding 和 dummy, 是将类别变量扩充为多个只显示 1，0 的变量，每个变量代表原类别变量中的一个类。注意他们之间的区别：https://www.cnblogs.com/lianyingteng/p/7792693.html

- 优点：解决了分类器不好处理分类数据的问题，在一定程度上也起到了扩充特征的作用。它的值只有 0 和 1，不同的类型存储在垂直的空间。

- 缺点：当类别的数量很多时，特征空间会变得非常大，容易造成维度灾难。（为避免维度灾难，后续可以考虑降维处理）

R 里面有很多现成的转化编码的包，我们这里使用了dummy_cols()函数做演示，可以看到原来的类别类型字段，已经扩充为多个 0，1 编码的字段。

```
h1n1_data_dummy <- dummy_cols(subset(h1n1_data_
    complete, select = c(age_group)), select_columns =
    c("age_group"))
head(h1n1_data_dummy)
```

```
##       age_group age_group_18 - 34 Years
    age_group_35 - 44 Years
## 1 55 - 64 Years                        0
                              0
## 2 35 - 44 Years                        0
                              1
## 3 18 - 34 Years                        1
                              0
```

```
## 4       65+ Years                          0
                              0
## 5 45 - 54 Years                          0
                              0
## 6       65+ Years                          0
                              0
##    age_group_45 - 54 Years age_group_55 - 64 Years
    age_group_65+ Years
## 1                       0                       1
                         0
## 2                       0                       0
                         0
## 3                       0                       0
                         0
## 4                       0                       0
                         1
## 5                       1                       0
                         0
## 6                       0                       0
                         1
```

0.2.4.2 标签编码

标签编码 (Label Encoder) 是将类别变量转换成连续的数值型变量，通常对有序的变量进行标签编码，既保留了顺序信息，也节约了空间（不会扩充变量）

R 里有一个特殊的结构 factor（factor 是有序的分类变量），我们这里可以利用 factor 来做标签编码。首先根据实际情况设置 factor 的类别顺序，然后直接用as.numeric()转化为数字。

```
h1n1_data_complete_lab_encoder <- h1n1_data_complete
h1n1_data_complete_lab_encoder$income_poverty_lab_
    encoder <- as.numeric(factor(h1n1_data_complete_lab
```

```
    _encoder$income_poverty, levels = c("Below␣Poverty"
    , "<=␣$75,000,␣Above␣Poverty", ">␣$75,000")))
head(subset(h1n1_data_complete_lab_encoder, select = c
    (income_poverty, income_poverty_lab_encoder)))
```

```
##                income_poverty
    income_poverty_lab_encoder
## 1                 Below Poverty
                                 1
## 2                 Below Poverty
                                 1
## 3 <= $75,000, Above Poverty
                                 2
## 4                 Below Poverty
                                 1
## 5 <= $75,000, Above Poverty
                                 2
## 6 <= $75,000, Above Poverty
                                 2
```

0.2.4.3 手动编码

比如，当某一个特征中有很多类别，我们认为某些类别可以合为一类，可以用case_when()函数手动处理。

```
h1n1_data_manual <- subset(h1n1_data_complete, select
    = c(age_group))
h1n1_data_manual$age_group_manual <- case_when(
  h1n1_data_manual$age_group %in% c("18␣-␣34␣Years") ~
      1,
  h1n1_data_manual$age_group %in% c("35␣-␣44␣Years", "
     45␣-␣54␣Years", "55␣-␣64␣Years") ~ 2,
  h1n1_data_manual$age_group %in% c("65+␣Years") ~ 3
)
```

```
head(h1n1_data_manual)
```

```
##         age_group age_group_manual
## 1 55 - 64 Years                 2
## 2 35 - 44 Years                 2
## 3 18 - 34 Years                 1
## 4       65+ Years               3
## 5 45 - 54 Years                 2
## 6       65+ Years               3
```

0.2.4.4 日期特征转换

参考附录R语言日期时间处理

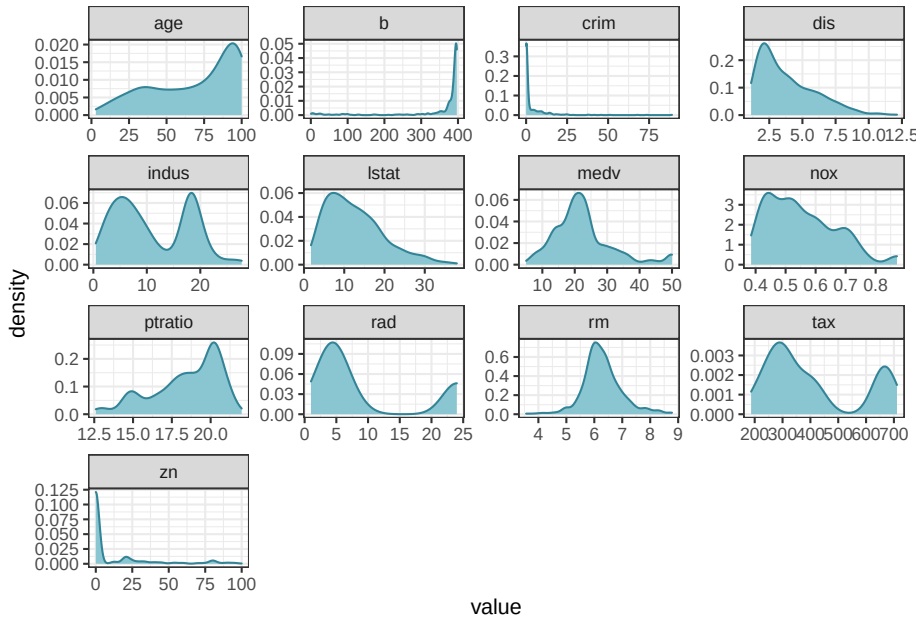
0.2.5 规范化与偏态数据

为什么要数据规范化？简单来说是为了去除数据量纲和数据大小的差异，确保数据是在同一量纲或者同一数量级下进行比较，一般用在机器学习算法之前。数据规范化又可以使用 0-1 规范化，Z-score 等方法。为什么要处理偏态数据?。很多模型会假设数据或参数服从正态分布。例如线性回归 (linear regression)，它假设误差服从正态分布。

提示：注意在测试数据与训练数据分布差别很大的情况下，对测试数据运用一些规范化方法时，可能因为数据分布不匹配而带来误差。

这里我们使用波士顿房价数据集来做演示。可以看到图中数据的偏态分布及量纲差别。

```
BostonHousing %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") +
  geom_density(color = "#348498", fill = "#8ac6d1") +
  theme_bw()
```



0.2.5.1 0-1 规范化

0-1 规范化是将原始数据缩放到 [0,1] 区间内，一般方法是最小最大规范的方法，公式如下：

$$x^* = \frac{x - min}{max - min}$$

这里用循环计算出每一列的最大最小值，再根据公式求出缩放后的数据。

```
nums_data_norm1 <- nums_data
for (col in names(nums_data_norm1))
{
  xmin <- min(nums_data_norm1[col])
  xmax <- max(nums_data_norm1[col])
  nums_data_norm1[col] <- (nums_data_norm1[col] - xmin
      ) / (xmax - xmin)
}

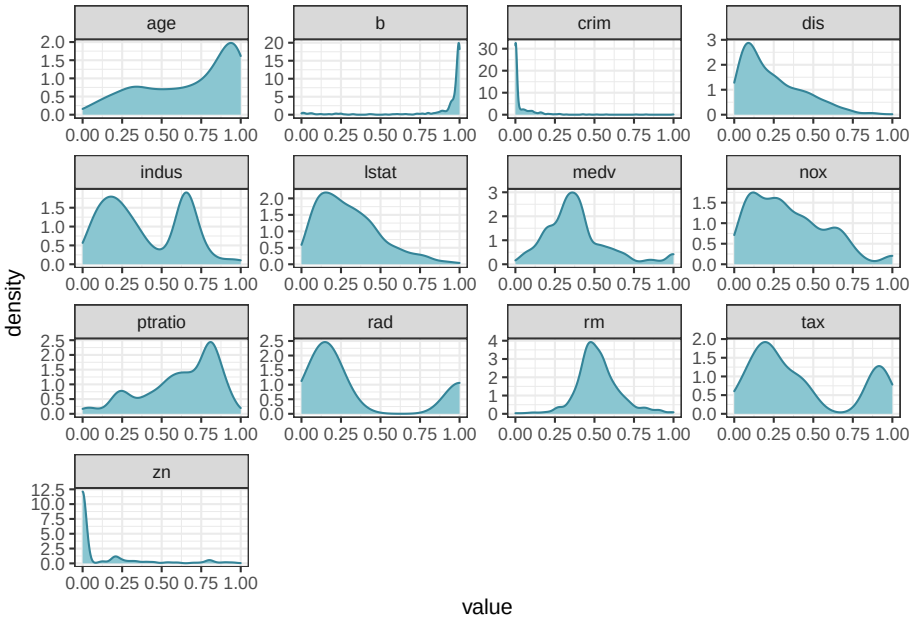
head(nums_data_norm1)
```

```
##           crim    zn       indus        nox         rm
         age        dis
## 1 0.0000000000 0.18 0.06781525 0.3148148 0.5775053
    0.6416066 0.2692031
## 2 0.0002359225 0.00 0.24230205 0.1728395 0.5479977
    0.7826982 0.3489620
## 3 0.0002356977 0.00 0.24230205 0.1728395 0.6943859
    0.5993821 0.3489620
## 4 0.0002927957 0.00 0.06304985 0.1502058 0.6585553
    0.4418126 0.4485446
## 5 0.0007050701 0.00 0.06304985 0.1502058 0.6871048
    0.5283213 0.4485446
## 6 0.0002644715 0.00 0.06304985 0.1502058 0.5497222
    0.5746653 0.4485446
##          rad        tax    ptratio          b
    lstat       medv
## 1 0.00000000 0.20801527 0.2872340 1.0000000
    0.08967991 0.4222222
## 2 0.04347826 0.10496183 0.5531915 1.0000000
    0.20447020 0.3688889
## 3 0.04347826 0.10496183 0.5531915 0.9897373
    0.06346578 0.6600000
## 4 0.08695652 0.06679389 0.6489362 0.9942761
    0.03338852 0.6311111
## 5 0.08695652 0.06679389 0.6489362 1.0000000
    0.09933775 0.6933333
## 6 0.08695652 0.06679389 0.6489362 0.9929901
    0.09602649 0.5266667
```

转换完再看一下分布，已经缩放到 0-1 之间了。

```
nums_data_norm1 %>%
  keep(is.numeric) %>%
  gather() %>%
```

```
ggplot(aes(value)) +
facet_wrap(~key, scales = "free") +
geom_density(color = "#348498", fill = "#8ac6d1") +
theme_bw()
```



此外可以用 dlookr 包里的transform()函数。

```
nums_data_norm2 <- nums_data
nums_data_norm2$crim <- dlookr::transform(nums_data$
    crim, method = "minmax")
```

0.2.5.2 Z-score 标准化

Z-score 标准化是原数据减去期望再除以标准差，将数据按比例缩放，使其落入到一个小的区间内，标准化后的数据可正可负，但是一般绝对值不会太大。

$$x^* = \frac{x - \mu}{\sigma}$$

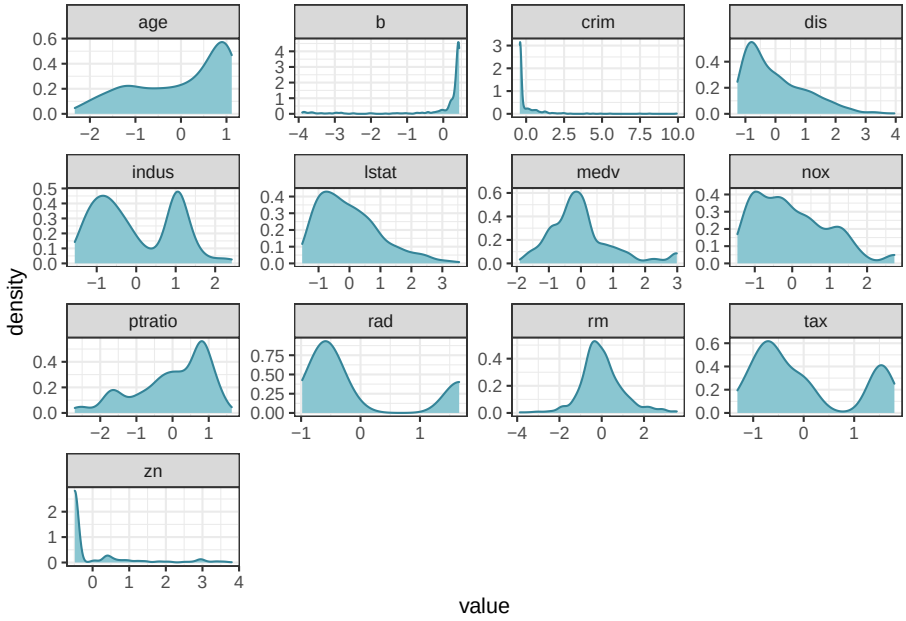
R 里面可以用scale()函数来计算 z-score。也可以 dlookr 包里的中transform()函数。

```
nums_data_zscore <- nums_data
nums_data_zscore <- scale(nums_data_zscore)
head(nums_data_zscore)
```

```
##          crim          zn       indus         nox
          rm         age       dis
## 1 -0.4193669  0.2845483 -1.2866362 -0.1440749
   0.4132629 -0.1198948 0.140075
## 2 -0.4169267 -0.4872402 -0.5927944 -0.7395304
   0.1940824  0.3668034 0.556609
## 3 -0.4169290 -0.4872402 -0.5927944 -0.7395304
   1.2814456 -0.2655490 0.556609
## 4 -0.4163384 -0.4872402 -1.3055857 -0.8344581
   1.0152978 -0.8090878 1.076671
## 5 -0.4120741 -0.4872402 -1.3055857 -0.8344581
   1.2273620 -0.5106743 1.076671
## 6 -0.4166314 -0.4872402 -1.3055857 -0.8344581
   0.2068916 -0.3508100 1.076671
##           rad        tax    ptratio          b
   lstat       medv
## 1 -0.9818712 -0.6659492 -1.4575580 0.4406159
   -1.0744990  0.1595278
## 2 -0.8670245 -0.9863534 -0.3027945 0.4406159
   -0.4919525 -0.1014239
## 3 -0.8670245 -0.9863534 -0.3027945 0.3960351
   -1.2075324  1.3229375
## 4 -0.7521778 -1.1050216  0.1129203 0.4157514
   -1.3601708  1.1815886
## 5 -0.7521778 -1.1050216  0.1129203 0.4406159
   -1.0254866  1.4860323
## 6 -0.7521778 -1.1050216  0.1129203 0.4101651
   -1.0422909  0.6705582
```

转换完再看一下分布，数据缩放后在 0 周围的一个小区间了。

```
data.frame(nums_data_zscore) %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") +
  geom_density(color = "#348498", fill = "#8ac6d1") +
  theme_bw()
```



0.2.5.3 对数转换 (log transform)

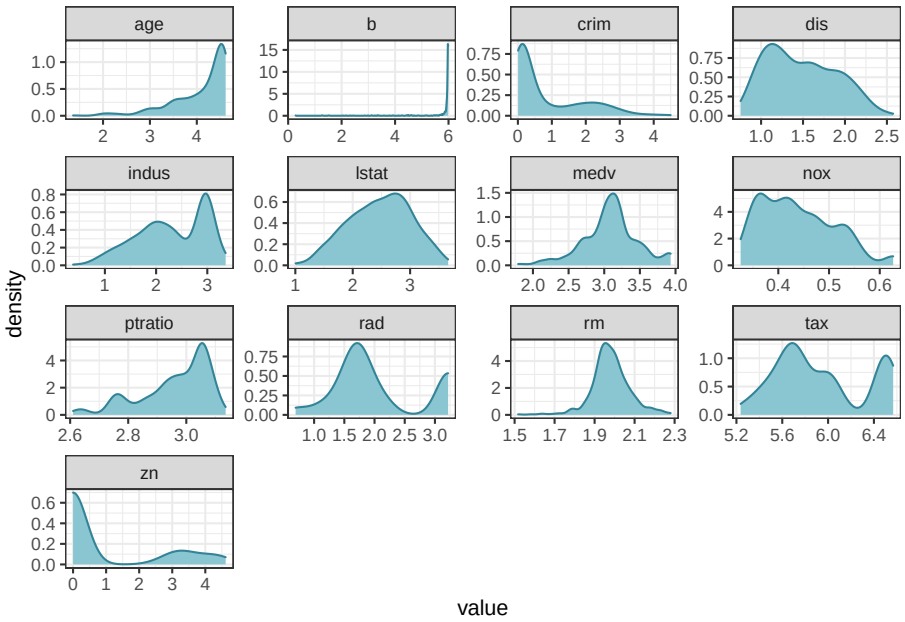
使用对数转换也是一种常见的处理偏斜特征的方法，但要注意原数据中不能含有负值。此外为了避免 0 值，我们通常使用 log1p，公式为lg(x+1)。可以直接用 dlookr 包里的transform()函数，一般结合 mutate 函数一起使用。

```
# 直接公式转换
nums_data_log1p1 <- log(nums_data + 1)
```

```
# 用 transform() 函数
nums_data_log1p2 <- nums_data
nums_data_log1p2$b <- dlookr::transform(nums_data_
    log1p2$b, method = "log+1")
```

转换完再看一下分布，大多变量转换后接近正态分布了。但是这里要特别注意离散数据。

```
nums_data_log1p1 %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~key, scales = "free") +
  geom_density(color = "#348498", fill = "#8ac6d1") +
  theme_bw()
```



0.2.5.4 Box-Cox

Box-Cox 变换是 Box 和 Cox 在 1964 年提出的一种广义幂变换方法，在变换后可以一定程度上减小不可观测的误差和预测变量的相关性，在机器学习中经常用来处理偏态分布。其一个显著优点是通过求变换参数来确定变换形式，而这个过程完全基于数据本身而无须任何先验信息，这无疑比凭经验或通过尝试而选用对数、平方根等变换方式要客观和精确。计算公式如下：

$$y(\lambda)=\begin{cases}\dfrac{y_i^{\lambda}-1}{\lambda}, \text{如果 } \lambda \neq 0; \\ \ln(y_i), \text{如果 } \lambda = 0.\end{cases}$$

示例参考附录基于R语言进行Box−Cox变换

0.2.6 小拓展

R 语言中，mutate 类似于 SQL 中，根据表的现有变量，生成新变量。使用 mutate 集中处理变量转换，代码显示较整洁。

```
h1n1_data_de <- h1n1_data_complete %>%
  to_dummy(education, suffix = "label") %>%
  bind_cols(h1n1_data_complete) %>%
  mutate(
    # 标签编码(label encoder)
    sex = as.factor(as.numeric(factor(sex))),
    income_poverty = (as.numeric(factor(
      income_poverty,
      levels = c(
        "Below Poverty",
        "<= $75,000, Above Poverty",
        "> $75,000"
      )
    ))),
    # 手动编码
    age_group = as.factor(
      case_when(
```

```
      age_group %in% c("18 – 34 Years") ~ 1,
      age_group %in% c("35 – 44 Years", "45 – 54 
          Years", "55 – 64 Years") ~ 2,
      age_group %in% c("65+ Years") ~ 3
    )
  ),
  # 标准化
  across(
    c(
      "h1n1_knowledge",
      "doctor_recc_h1n1",
      "chronic_med_condition",
      "opinion_h1n1_vacc_effective",
      "age_group",
      "income_poverty"
    ),
    ~ scale(as.numeric(.x))
  )
) %>%
dplyr::select(-one_of("education", "education_"))

head(h1n1_data_de)
```

```
##   education_< 12 Years education_12 Years
    education_College Graduate
## 1                    1                   0
                            0
## 2                    0                   1
                            0
## 3                    0                   0
                            1
## 4                    0                   1
                            0
## 5                    0                   0
```

```
                                0
## 6                             0                    1
                                0
##   education_Some College respondent_id
    h1n1_knowledge doctor_recc_h1n1
## 1                      0             0
    -2.0416901        -0.5258839
## 2                      0             1
    1.1935904        -0.5258839
## 3                      0             2
    -0.4240499        -0.5258839
## 4                      0             3
    -0.4240499        -0.5258839
## 5                      1             4
    -0.4240499        -0.5258839
## 6                      0             5
    -0.4240499        -0.5258839
##   chronic_med_condition health_insurance
    opinion_h1n1_vacc_effective
## 1             -0.6284091                1
                      -0.8439071
## 2             -0.6284091                1
                       1.1407906
## 3              1.5912605                1
                      -0.8439071
## 4              1.5912605                1
                      -0.8439071
## 5             -0.6284091                0
                      -0.8439071
## 6             -0.6284091                1
                       1.1407906
##     age_group sex income_poverty h1n1_vaccine
## 1 -0.09109418   1      -1.8905904            0
```

```
## 2 −0.09109418    2       −1.8905904          0
## 3 −1.58547517    2       −0.2945789          0
## 4  1.40328681    1       −1.8905904          0
## 5 −0.09109418    1       −0.2945789          0
## 6  1.40328681    2       −0.2945789          0
```

注意在机器学习中，尽量在数据集划分后，分别在训练集与验证集、测试集上进行数据清洗，避免数据泄露。R 中的数据集划分方法参考附录R中数据集分割。

0.2.7 思考与练习

看完了本节数据清洗与准备，尝试着选取一个完整的数据集（从本节中选取或使用自己的数据集），来做一次清洗吧！

附录：参考资料

理论资料

数据的预处理基础：如何处理缺失值 https://cloud.tencent.com/developer/article/1626004

多重插补法：处理缺失值之多重插补（Multiple Imputation）https://zhuanlan.zhihu.com/p/36436260

异常值检测： R 语言–异常值检测 https://blog.csdn.net/kicilove/article/details/76260350

异常值检测之 LOF：异常检测算法之局部异常因子算法-Local Outlier Factor(LOF) https://blog.csdn.net/BigData_Mining/article/details/102914342

规范化：规范化、标准化、归一化、正则化 https://blog.csdn.net/u014381464/article/details/81101551

什么样的模型对缺失值更敏感?：https://blog.csdn.net/zhang15953709913/article/details/88717220

R 语言函数用法示例

funModeling用法示例：https://cran.r-project.org/web/packages/funModeling/vignettes/funModeling_quickstart.html

tidyverse官方文档：https://www.tidyverse.org/

VIM教学网页：https://www.datacamp.com/community/tutorials/visualize-data-vim-package

mice使用文档 (Multivariate Imputation by Chained Equations)：https://cran.r-project.org/web/packages/mice/mice.pdf

mice使用中文解释：https://blog.csdn.net/sinat_26917383/article/details/51265213

mice检验结果解释：http://blog.fens.me/r-na-mice/

caret包数据预处理：https://www.cnblogs.com/Hyacinth-Yuan/p/8284612.html

R 语言日期时间处理：https://zhuanlan.zhihu.com/p/83984803

基于 R 语言进行 Box-Cox 变换：https://ask.hellobi.com/blog/R_shequ/18371

R 中数据集分割：https://zhuanlan.zhihu.com/p/45163182

本章作者

June

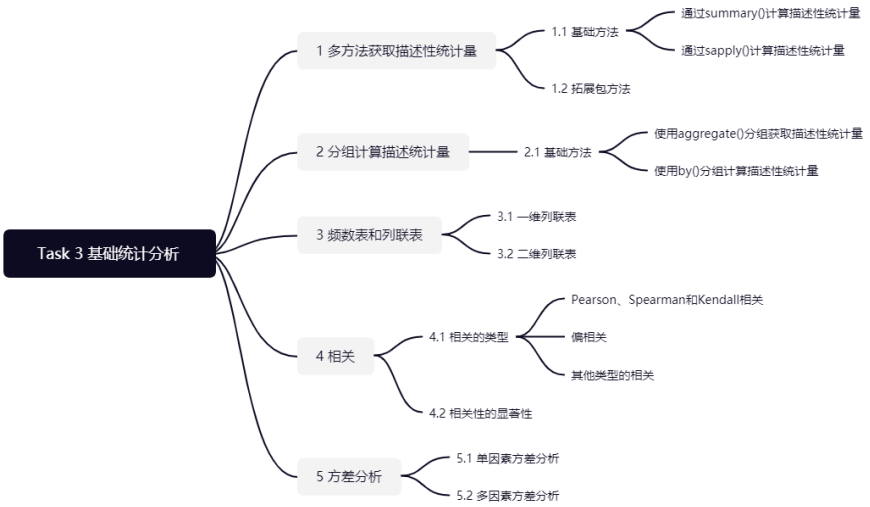
悉尼大学研究生，Datawhale 成员
https://blog.csdn.net/Yao_June

关于 Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成

员。Datawhale 以 “for the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：

0.3 基本统计分析



准备工作

如果没有相关的包，则使用 install .packages('package_name')进行安装以下包。

```
library(pastecs)
library(psych)
library(ggm)
```

读取数据，使用 H1N1 流感数据集和波士顿房价数据集。

```
flu <- read.table("./datasets/h1n1_flu.csv", header =
    TRUE, sep = ",")
housing <- read.csv("./datasets/BostonHousing.csv",
    header = TRUE)
```

0.3.1 多种方法获取描述性统计量

0.3.1.1 基础方法

通过 summary 计算数值型变量的最大值、最小值、分位数以及均值，类别变量计算频数统计。

```
summary(flu[c("household_children", "sex")])
```

```
##  household_children      sex
##  Min.   :0.0000     Length:26707
##  1st Qu.:0.0000     Class :character
##  Median :0.0000     Mode  :character
##  Mean   :0.5346
##  3rd Qu.:1.0000
##  Max.   :3.0000
##  NA's   :249
```

```
summary(flu[c("h1n1_concern", "h1n1_knowledge")])
```

```
##   h1n1_concern    h1n1_knowledge
##  Min.   :0.000   Min.   :0.000
##  1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000   Median :1.000
##  Mean   :1.618   Mean   :1.263
##  3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :3.000   Max.   :2.000
##  NA's   :92      NA's   :116
```

通过 sapply() 计算描述性统计量，先定义统计函数，在进行聚合计算。

```
mystats <- function(x, na.omit = FALSE) {
  if (na.omit) {
    x <- x[!is.na(x)]
  }
  m <- mean(x)
  n <- length(x)
```

```
  s <- sd(x)
  skew <- sum((x - m)^3 / s^3) / n
  kurt <- sum((x - m)^4 / s^4) / n - 3
  return(c(n = n, mean = m, stdev = s, skew = skew,
      kurtosis = kurt))
}

sapply(flu[c("h1n1_concern", "h1n1_knowledge")],
    mystats)
```

```
##           h1n1_concern h1n1_knowledge
## n                26707           26707
## mean                NA              NA
## stdev               NA              NA
## skew                NA              NA
## kurtosis            NA              NA
```

0.3.1.2 拓展包方法

通过 pastecs 包中的 stat.desc() 函数计算描述性统计量，可以得到中位数、平均数、平均数的标准误、平均数置信度为 95% 的置信区间、方差、标准差以及变异系数。

```
stat.desc(flu[c("household_children", "sex")])
```

```
##              household_children sex
## nbr.val            2.645800e+04  NA
## nbr.null           1.867200e+04  NA
## nbr.na             2.490000e+02  NA
## min                0.000000e+00  NA
## max                3.000000e+00  NA
## range              3.000000e+00  NA
## sum                1.414400e+04  NA
## median             0.000000e+00  NA
```

```
## mean                 5.345831e-01  NA
## SE.mean              5.706247e-03  NA
## CI.mean.0.95         1.118455e-02  NA
## var                  8.615057e-01  NA
## std.dev              9.281733e-01  NA
## coef.var             1.736256e+00  NA
```

通过 psych 包中的 describe() 计算描述性统计量。

```
describe(flu[c("household_children", "sex")])
```

```
##                    vars     n mean   sd median
    trimmed mad min max range skew
## household_children    1 26458 0.53 0.93      0
    0.34   0   0   3     3 1.54
## sex*                  2 26707 1.41 0.49      1
    1.38   0   1   2     1 0.38
##                    kurtosis   se
## household_children     1.04 0.01
## sex*                  -1.85 0.00
```

0.3.2 分组计算描述性统计

0.3.2.1 基础方法

0.3.2.1.1 * 使用 aggregate() 分组获取描述性统计

1. 分组计算不同性别收入贫困计数。
2. 是否属于查尔斯河的房价中位数平均值。

```
aggregate(flu[c("income_poverty")], by = list(sex =
    flu$sex), length)
```

```
##      sex income_poverty
## 1 Female           15858
## 2   Male           10849
```

```
aggregate(housing$medv, by = list(medv = housing$chas)
    , FUN = mean)
```

```
##   medv        x
## 1    0 22.09384
## 2    1 28.44000
```

0.3.2.1.2 * 使用 by() 分组计算描述性统计量

```
by(flu[c("income_poverty", "sex")], flu$sex, length)
```

```
## flu$sex: Female
## [1] 2
##
```

```
## flu$sex: Male
## [1] 2
```

0.3.3 频数表和列联表

```
table(flu$sex)
```

```
##
## Female   Male
##  15858  10849
```

0.3.4 相关

0.3.4.1 相关的类型

0.3.4.1.1 * Pearson、Spearman 和 Kendall 相关

R 可以计算多种相关系数，包括 Pearson 相关系数、Spearman 相关系数、Kendall 相关系数、偏相关系数、多分格（polychoric）相关系数和多系列

（polyserial）相关系数。1. 计算房价数据的相关系数，默认是 Pearson 相关系数。

```
cor(housing)
```

```
##                X         crim           zn
    indus         chas
## X          1.000000000  0.40740717 -0.10339336
    0.39943885 -0.003759115
## crim       0.407407172  1.00000000 -0.20046922
    0.40658341 -0.055891582
## zn        -0.103393357 -0.20046922  1.00000000
    -0.53382819 -0.042696719
## indus      0.399438850  0.40658341 -0.53382819
    1.00000000  0.062938027
## chas      -0.003759115 -0.05589158 -0.04269672
    0.06293803  1.000000000
## nox        0.398736174  0.42097171 -0.51660371
    0.76365145  0.091202807
## rm        -0.079971150 -0.21924670  0.31199059
    -0.39167585  0.091251225
## age        0.203783510  0.35273425 -0.56953734
    0.64477851  0.086517774
## dis       -0.302210959 -0.37967009  0.66440822
    -0.70802699 -0.099175780
## rad        0.686001976  0.62550515 -0.31194783
    0.59512927 -0.007368241
## tax        0.666625924  0.58276431 -0.31456332
    0.72076018 -0.035586518
## ptratio    0.291074227  0.28994558 -0.39167855
    0.38324756 -0.121515174
## b         -0.295041232 -0.38506394  0.17552032
    -0.35697654  0.048788485
## lstat      0.258464770  0.45562148 -0.41299457
    0.60379972 -0.053929298
```

```
## medv     -0.226603643 -0.38830461  0.36044534
    -0.48372516  0.175260177
##                   nox           rm           age
           dis           rad
## X          0.39873617 -0.07997115  0.20378351
    -0.30221096  0.686001976
## crim       0.42097171 -0.21924670  0.35273425
    -0.37967009  0.625505145
## zn        -0.51660371  0.31199059 -0.56953734
    0.66440822 -0.311947826
## indus      0.76365145 -0.39167585  0.64477851
    -0.70802699  0.595129275
## chas       0.09120281  0.09125123  0.08651777
    -0.09917578 -0.007368241
## nox        1.00000000 -0.30218819  0.73147010
    -0.76923011  0.611440563
## rm        -0.30218819  1.00000000 -0.24026493
    0.20524621 -0.209846668
## age        0.73147010 -0.24026493  1.00000000
    -0.74788054  0.456022452
## dis       -0.76923011  0.20524621 -0.74788054
    1.00000000 -0.494587930
## rad        0.61144056 -0.20984667  0.45602245
    -0.49458793  1.000000000
## tax        0.66802320 -0.29204783  0.50645559
    -0.53443158  0.910228189
## ptratio    0.18893268 -0.35550149  0.26151501
    -0.23247054  0.464741179
## b         -0.38005064  0.12806864 -0.27353398
    0.29151167 -0.444412816
## lstat      0.59087892 -0.61380827  0.60233853
    -0.49699583  0.488676335
## medv      -0.42732077  0.69535995 -0.37695457
```

```
    0.24992873 -0.381626231
##                   tax    ptratio          b
    lstat       medv
## X          0.66662592  0.2910742 -0.29504123
    0.2584648 -0.2266036
## crim       0.58276431  0.2899456 -0.38506394
    0.4556215 -0.3883046
## zn        -0.31456332 -0.3916785  0.17552032
   -0.4129946  0.3604453
## indus      0.72076018  0.3832476 -0.35697654
    0.6037997 -0.4837252
## chas      -0.03558652 -0.1215152  0.04878848
   -0.0539293  0.1752602
## nox        0.66802320  0.1889327 -0.38005064
    0.5908789 -0.4273208
## rm        -0.29204783 -0.3555015  0.12806864
   -0.6138083  0.6953599
## age        0.50645559  0.2615150 -0.27353398
    0.6023385 -0.3769546
## dis       -0.53443158 -0.2324705  0.29151167
   -0.4969958  0.2499287
## rad        0.91022819  0.4647412 -0.44441282
    0.4886763 -0.3816262
## tax        1.00000000  0.4608530 -0.44180801
    0.5439934 -0.4685359
## ptratio    0.46085304  1.0000000 -0.17738330
    0.3740443 -0.5077867
## b         -0.44180801 -0.1773833  1.00000000
   -0.3660869  0.3334608
## lstat      0.54399341  0.3740443 -0.36608690
    1.0000000 -0.7376627
## medv      -0.46853593 -0.5077867  0.33346082
   -0.7376627  1.0000000
```

2. 指定计算 Spearman 相关系数

```
cor(housing, method = "spearman")
```

```
##                X         crim          zn
    indus         chas
## X          1.000000000  0.46103705 -0.1605047
    0.32462127 -0.003759115
## crim       0.461037054  1.00000000 -0.5716602
    0.73552374  0.041536888
## zn        -0.160504702 -0.57166021  1.0000000
    -0.64281060 -0.041936998
## indus      0.324621271  0.73552374 -0.6428106
    1.00000000  0.089841379
## chas      -0.003759115  0.04153689 -0.0419370
    0.08984138  1.000000000
## nox        0.432491886  0.82146466 -0.6348284
    0.79118913  0.068426283
## rm        -0.035641354 -0.30911647  0.3610737
    -0.41530129  0.058812916
## age        0.208323439  0.70413998 -0.5444226
    0.67948671  0.067791779
## dis       -0.373498683 -0.74498614  0.6146265
    -0.75707970 -0.080248080
## rad        0.588480705  0.72780697 -0.2787672
    0.45550745  0.024578885
## tax        0.536928176  0.72904490 -0.3713945
    0.66436139 -0.044485772
## ptratio    0.297897432  0.46528319 -0.4484754
    0.43371046 -0.136064621
## b         -0.154474321 -0.36055532  0.1631351
    -0.28583984 -0.039810497
## lstat      0.257542491  0.63476026 -0.4900739
    0.63874741 -0.050574829
```

```
## medv     −0.273633481 −0.55889095  0.4381790
    −0.57825539  0.140612154
##                    nox           rm          age
           dis          rad          tax
## X          0.43249189 −0.03564135  0.20832344
    −0.37349868  0.58848071  0.53692818
## crim       0.82146466 −0.30911647  0.70413998
    −0.74498614  0.72780697  0.72904490
## zn        −0.63482840  0.36107373 −0.54442256
    0.61462654 −0.27876717 −0.37139450
## indus      0.79118913 −0.41530129  0.67948671
    −0.75707970  0.45550745  0.66436139
## chas       0.06842628  0.05881292  0.06779178
    −0.08024808  0.02457888 −0.04448577
## nox        1.00000000 −0.31034391  0.79515291
    −0.88001486  0.58642870  0.64952656
## rm        −0.31034391  1.00000000 −0.27808202
    0.26316822 −0.10749220 −0.27189846
## age        0.79515291 −0.27808202  1.00000000
    −0.80160979  0.41798261  0.52636644
## dis       −0.88001486  0.26316822 −0.80160979
    1.00000000 −0.49580647 −0.57433641
## rad        0.58642870 −0.10749220  0.41798261
    −0.49580647  1.00000000  0.70487572
## tax        0.64952656 −0.27189846  0.52636644
    −0.57433641  0.70487572  1.00000000
## ptratio    0.39130908 −0.31292257  0.35538428
    −0.32204056  0.31832966  0.45334546
## b         −0.29666158  0.05366004 −0.22802200
    0.24959532 −0.28253261 −0.32984308
## lstat      0.63682829 −0.64083156  0.65707079
    −0.56426219  0.39432245  0.53442319
## medv      −0.56260883  0.63357643 −0.54756169
```

```
    0.44585685 -0.34677626 -0.56241063
##              ptratio           b       lstat
    medv
## X          0.29789743 -0.15447432  0.25754249
    -0.2736335
## crim       0.46528319 -0.36055532  0.63476026
    -0.5588909
## zn        -0.44847543  0.16313510 -0.49007389
    0.4381790
## indus      0.43371046 -0.28583984  0.63874741
    -0.5782554
## chas      -0.13606462 -0.03981050 -0.05057483
    0.1406122
## nox        0.39130908 -0.29666158  0.63682829
    -0.5626088
## rm        -0.31292257  0.05366004 -0.64083156
    0.6335764
## age        0.35538428 -0.22802200  0.65707079
    -0.5475617
## dis       -0.32204056  0.24959532 -0.56426219
    0.4458569
## rad        0.31832966 -0.28253261  0.39432245
    -0.3467763
## tax        0.45334546 -0.32984308  0.53442319
    -0.5624106
## ptratio    1.00000000 -0.07202734  0.46725885
    -0.5559047
## b         -0.07202734  1.00000000 -0.21056185
    0.1856641
## lstat      0.46725885 -0.21056185  1.00000000
    -0.8529141
## medv      -0.55590468  0.18566412 -0.85291414
    1.0000000
```

3. 城镇人均犯罪率与房价的相关系数

```
x <- housing
y <- housing[c("medv")]
cor(x, y)
```

```
##                medv
## X       -0.2266036
## crim    -0.3883046
## zn       0.3604453
## indus   -0.4837252
## chas     0.1752602
## nox     -0.4273208
## rm       0.6953599
## age     -0.3769546
## dis      0.2499287
## rad     -0.3816262
## tax     -0.4685359
## ptratio -0.5077867
## b        0.3334608
## lstat   -0.7376627
## medv     1.0000000
```

0.3.4.1.2 * 偏相关

偏相关是指在控制一个或多个定量变量时，另外两个定量变量之间的相互关系。使用 ggm 包中的 pcor() 函数计算偏相关系数。

0.3.4.2 相关性的显著性检验

```
cor.test(housing[, c("crim")], housing[, c("medv")])
```

```
##
##  Pearson's product-moment correlation
```

```
##
## data:  housing[, c("crim")] and housing[, c("medv")
    ]
## t = -9.4597, df = 504, p-value < 2.2e-16
## alternative hypothesis: true correlation is not
    equal to 0
## 95 percent confidence interval:
##  -0.4599064 -0.3116859
## sample estimates:
##       cor
## -0.3883046
```

0.3.5 方差分析

方差分析（ANOVA）又称 “变异数分析” 或 “F 检验”，用于两个及两个以上样本均数差别的显著性检验。

0.3.5.1 单因素方差分析

从输出结果的 F 检验值来看，p<0.05 比较显著，说明是否在查尔斯河对房价有影响。

```
fit <- aov(housing$medv ~ housing$chas)
summary(fit)
```

```
##                Df Sum Sq Mean Sq F value   Pr(>F)
## housing$chas    1   1312  1312.1   15.97 7.39e-05
    ***
## Residuals     504  41404    82.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
    '.' 0.1 ' ' 1
```

0.3.5.2 多因素方差分析

构建多因素方差分析，查看因子对房价的影响是否显著。

```
fit <- aov(housing$medv ~ housing$crim * housing$b)
summary(fit)
```

```
##                              Df Sum Sq Mean Sq F value
      Pr(>F)
## housing$crim                  1   6441    6441   96.05
    < 2e-16 ***
## housing$b                     1   1697    1697   25.30
    6.83e-07 ***
## housing$crim:housing$b        1    917     917   13.68
    0.000241 ***
## Residuals                   502  33662      67
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
    '.' 0.1 ' ' 1
```

本章作者

杨佳达

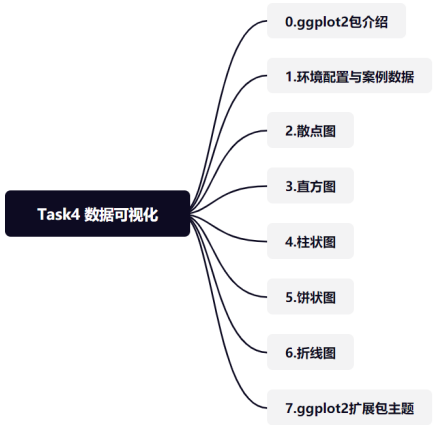
数据挖掘师，Datawhale 成员，目前在国内某第三方数据服务公司做数据分析挖掘及数据产品
https://github.com/yangjiada

关于 Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以 “for the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale

用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：

0.4 数据可视化



ggplot2 包介绍

ggplot2 包由 Hadley Wickham 编写，提供了一种基于 Wilkinson 所述图形语法的图形系统。ggplot2 包的目标是提供一个全面的、基于语法的、连贯一致的图形生成系统，允许用户创建新颖的、有创新性的数据可视化图形。

总的来说有以下几点：

- ggplot2 的核心理念是将绘图与数据分离，数据相关的绘图与数据无关的绘图分离
- ggplot2 保有命令式作图的调整函数，使其更具灵活性
- ggplot2 将常见的统计变换融入到了绘图中。
- ggplot2 是按图层作图

ggplot2 图像的三个基本构成：数据、图形属性映射、几何对象

按照 ggplot2 的绘图理念，Plot(图)= data(数据集)+ Aesthetics(美学映射)+ Geometry(几何对象)。

例如：

```
# ggplot(data,aes(x=x,y=y))+geom_point()
```

- 数据：用于绘制图形的数据
- 映射：aes() 函数是 ggplot2 中的映射函数, 所谓的映射即为数据集中的数据关联到相应的图形属性过程中一种对应关系, 图形的颜色，形状，分组等都可以通过通过数据集中的变量映射。
- 几何对象：我们在图中实际看到的图形元素，如点、线、多边形等。

ggplot2 绘图代码如同数据公式一般，只需要套相应的公式即可绘制出丰富的图形，后续的讲解也会按照此方法。

ggplot2 参考链接：

- https://ggplot2.tidyverse.org/reference/
- https://ggplot2-book.org/

ggplot2 的安装方法

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

0.4.1 环境配置

```
library(ggplot2) # 画图工具ggplot2
library(ggpubr) # 将多个图形拼接
library(plyr) # 数据处理包
```

在本讲中会用到 ggpubr 中的 ggrrange 这个多图拼接工具，详细使用方法参见:http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/81-ggplot2-easy-way-to-mix-multiple-graphs-on-the-same-page/

案例数据

本节内容将会使用到两个数据集

1.1h1n1 流感问卷数据集

h1n1 流感问卷数据集是关于 h1n1 流感问卷调查的一个数据，属于外部数据数据集包含 26,707 个受访者数据，共有 32 个特征 +1 个标签（是否接种 h1n1 疫苗）

读取相关的数据集

```
h1n1_data <- read.csv("./datasets/h1n1_flu.csv",
    header = TRUE)
```

1.2 波士顿房价数据集

波士顿房价数据集属于 R 语言自带数据集，也可以通过外部读取

读取相关的数据集

```
boston_data <- read.csv("./datasets/BostonHousing.csv"
    , header = TRUE)
```

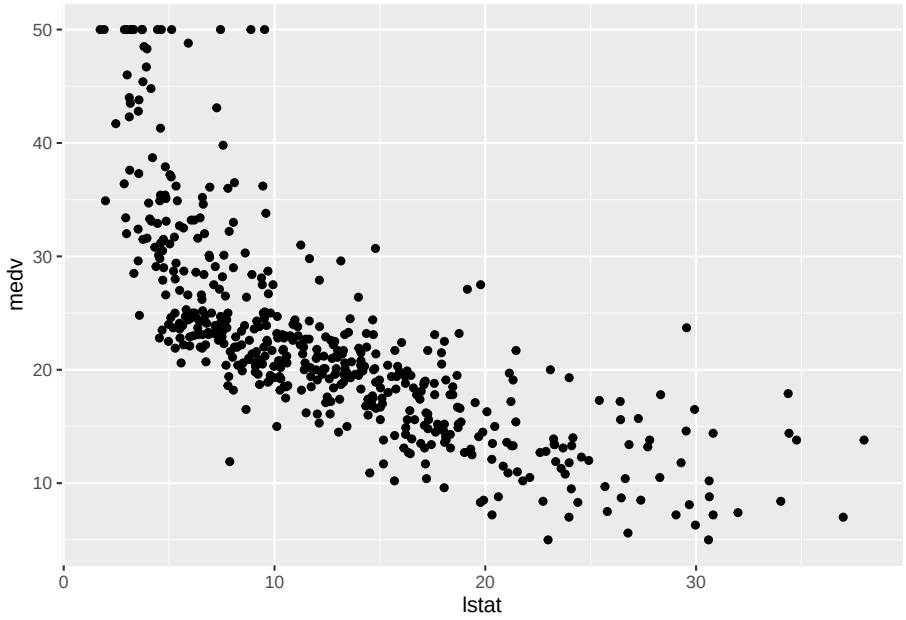
0.4.2 散点图

散点图是指在数理统计回归分析中，数据点在直角坐标系平面上的分布图，散点图表示因变量随自变量而变化的大致趋势，由此趋势可以选择合适的函数进行经验分布的拟合，进而找到变量之间的函数关系。

散点图的优势：

- 数据用图表来展示，显然比较直观，在工作汇报等场合能起到事半功倍的效果，让听者更容易接受，理解你所处理的数据。
- 散点图更偏向于研究型图表，能让我们发现变量之间隐藏的关系为我们决策作出重要的引导作用。
- 散点图核心的价值在于发现变量之间的关系，包括线性与非线性之间的关系。

```
# 读取数据
boston_data <- read.csv("./datasets/BostonHousing.csv"
    , header = TRUE)
# 绘制简单的散点图 x轴选择的是lstat ,y轴选择的是medv
ggplot(data = boston_data, aes(x = lstat, y = medv)) +
  geom_point()
```



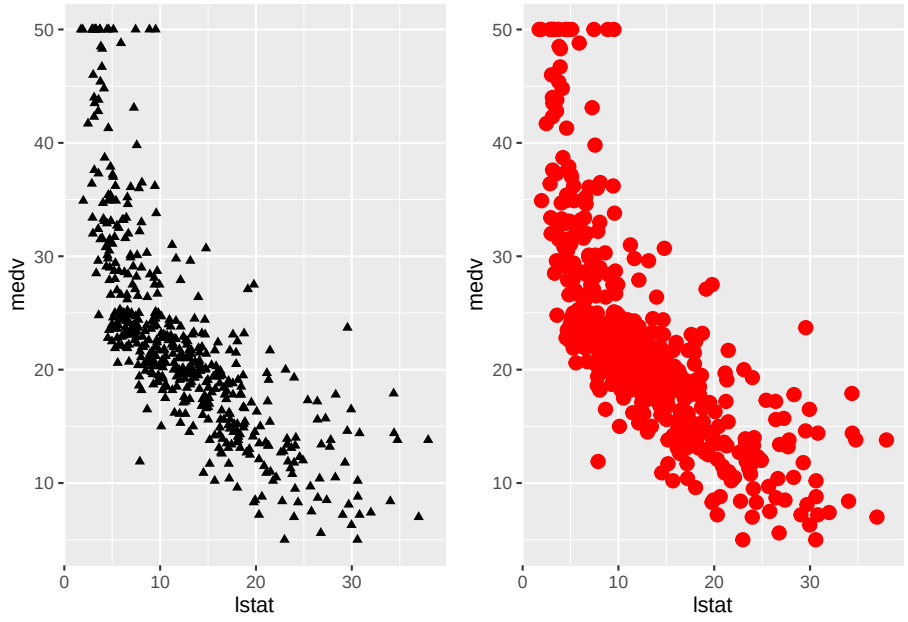
上图选择的是 lstat 为 x 轴，medv 为 y 轴绘制的散点图，x 轴表示弱势群体人口所占比例，y 轴表示房屋的平均价格，通过图上的数据可以看到，弱势人群的比例增加会影响房价，这 2 个变量呈现一定的负相关。

ggplot2 可以修改散点图的性状和大小,R 语言中存储了一些相关的形状

size 参数修改点的大小，color 参数修改点的颜色

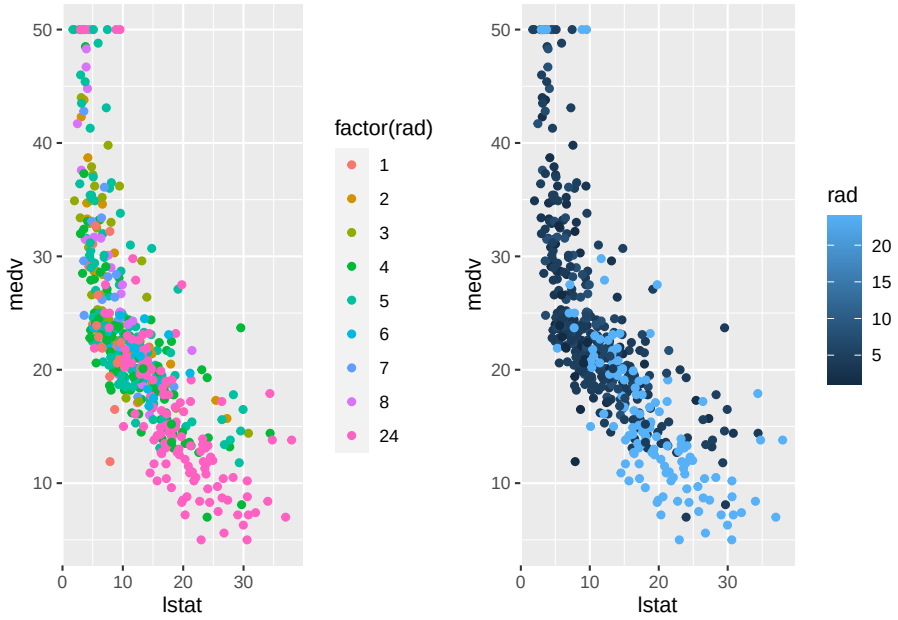
```
# 使用第17号形状
```

```
p1 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv)) +
  geom_point(shape = 17)
# size参数修改点的大小, color参数修改点的颜色
p2 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv)) +
  geom_point(size = 3, color = "red")
ggarrange(p1, p2, nrow = 1)
```



可将数据集的其它属性映射到散点图的颜色属性中

```
p3 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = factor(rad))) +
  geom_point()
p4 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = rad)) +
  geom_point()
ggarrange(p3, p4, nrow = 1)
```



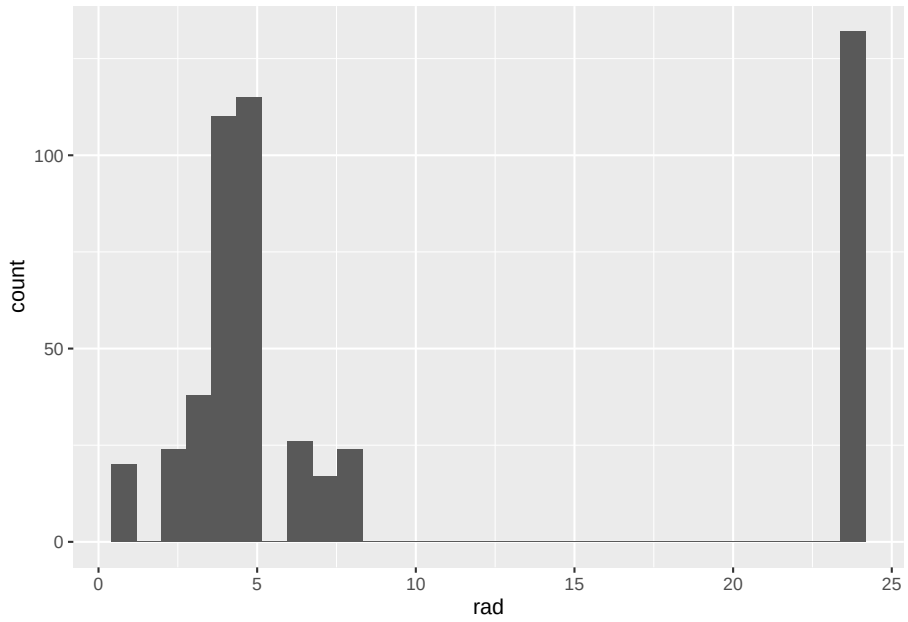
ggplot2 关于散点图的相关做法有很详细的介绍，相关参考链接：https://ggplot2.tidyverse.org/reference/geom_point.html

0.4.3 直方图

直方图是一种统计报告图，由一系列高度不等的纵向条纹或线段表示数据分布的情况。一般用横轴表示数据类型，纵轴表示分布情况。直方图可以很好的查看数据的分布情况，是常用的数据可视化展示图形。

我们对 rad 变量进行直方图分析

```
ggplot(data = boston_data, aes(x = rad)) +
  geom_histogram()
```

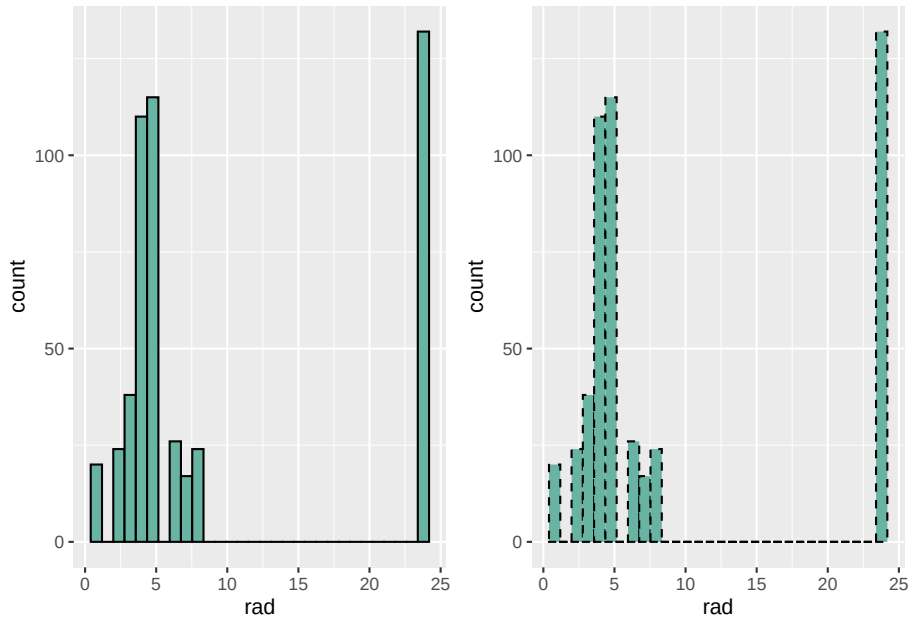


可以看到 ggplot2 可以自动对数据进行直方图的统计

我们给直方图填充颜色，同时改变直方图类型 color 表示直方图的边框，fill 表示直方图中的填充颜色，ggplot2 支持 RGB 颜色表的配色方案，linetype 表示直方图线的类型

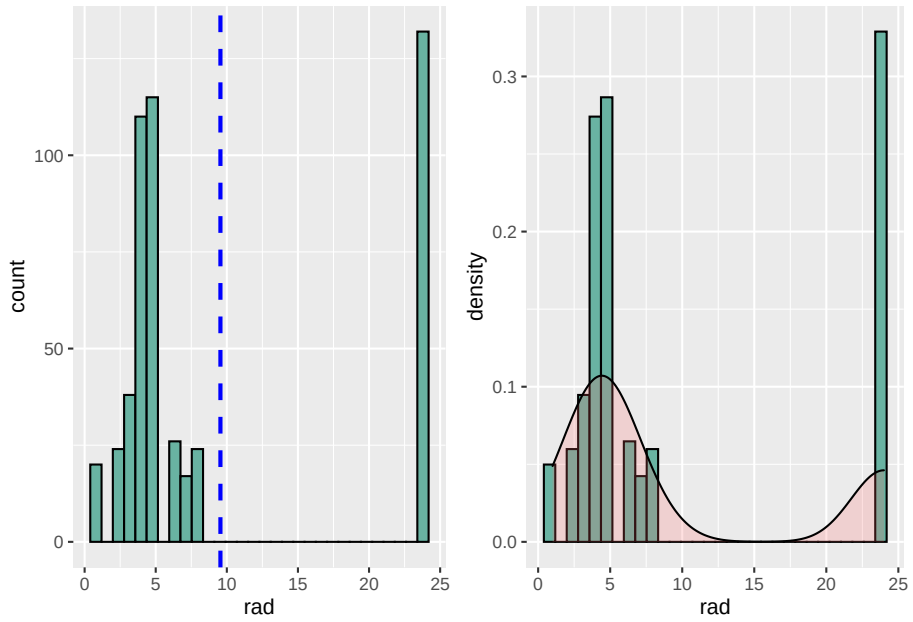
RGB 颜色表可以参考：http://www.mgzxzs.com/sytool/se.htm

```
p5 <- ggplot(data = boston_data, aes(x = rad)) +
  geom_histogram(color = "black", fill = "#69b3a2")
p6 <- ggplot(data = boston_data, aes(x = rad)) +
  geom_histogram(color = "black", fill = "#69b3a2",
      linetype = "dashed")
ggarrange(p5, p6, nrow = 1)
```



ggplot2 也支持在直方图上添加平均线和密度图

```
p7 <- p5 + geom_vline(aes(xintercept = mean(rad)),
    color = "blue", linetype = "dashed", size = 1)
p8 <- ggplot(data = boston_data, aes(x = rad)) +
  geom_histogram(color = "black", fill = "#69b3a2",
      aes(y = ..density..)) +
  geom_density(alpha = .2, fill = "#FF6666")
ggarrange(p7, p8, nrow = 1)
```



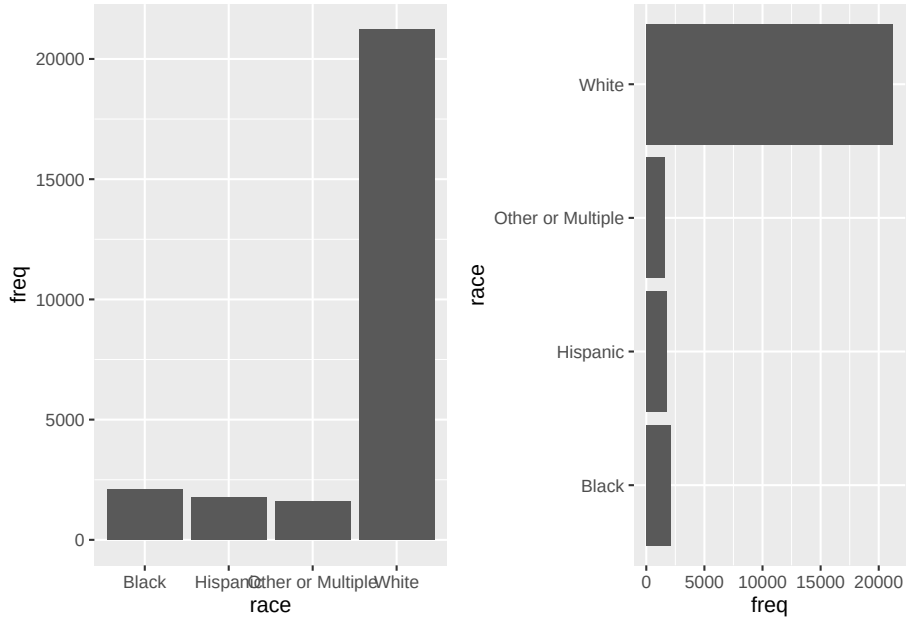
ggplot2 关于直方图的相关做法有很详细的介绍，相关参考链接：https://ggplot2.tidyverse.org/reference/geom_histogram.html

0.4.4 柱状图

柱状图是一种常用的数据可视化图形，根据翻译的不同，柱状图又叫长条图、柱状统计图、条状图、棒形图柱状图图用来比较两个或以上的价值（不同时间或者不同条件），只有一个变量，通常利用于较小的数据集分析。长条图亦可横向排列，或用多维方式表达。需要注意的是柱状图与直方图是不同的数据可视化方法，不要弄混淆了。

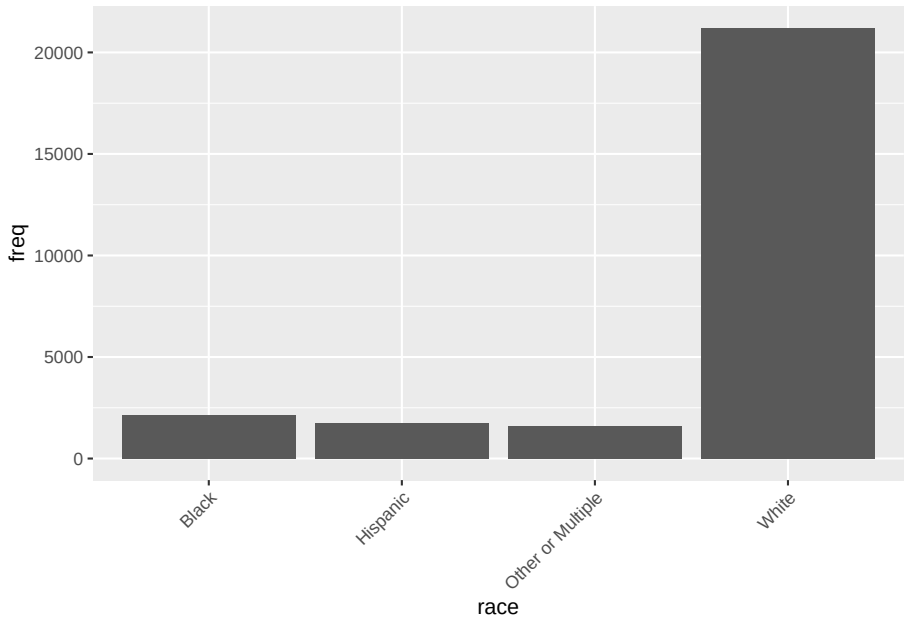
对 h1n1 数据集中填写人的受教育情况进行可视化展示, 使用 pylr 包中的 count 对 edcation 进行计数统计

```
data <- count(h1n1_data["race"])
p <- ggplot(data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity")
# 也可以进行水平放置
p1 <- p + coord_flip()
ggarrange(p, p1)
```



可以看到左边的柱状图文字有点挡住了，我们把文字旋转 45°

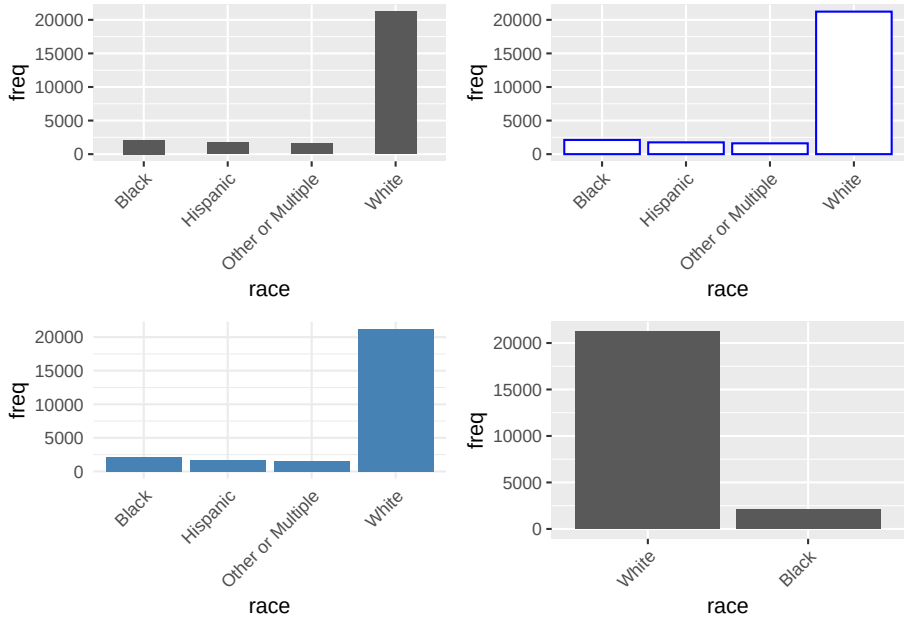
```
data <- count(h1n1_data["race"])
ggplot(data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust =
      1))
```



对柱状图的样式进行修改

```
# 更改条的宽度和颜色:
# 更改条的宽度
p2 <- ggplot(data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity", width = 0.5) +
  theme(axis.text.x = element_text(angle = 45, hjust =
      1))
# 改变颜色
p3 <- ggplot(data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity", color = "blue", fill = "
    white") +
  theme(axis.text.x = element_text(angle = 45, hjust =
      1))
# 最小主题+蓝色填充颜色
p4 <- ggplot(data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust =
```

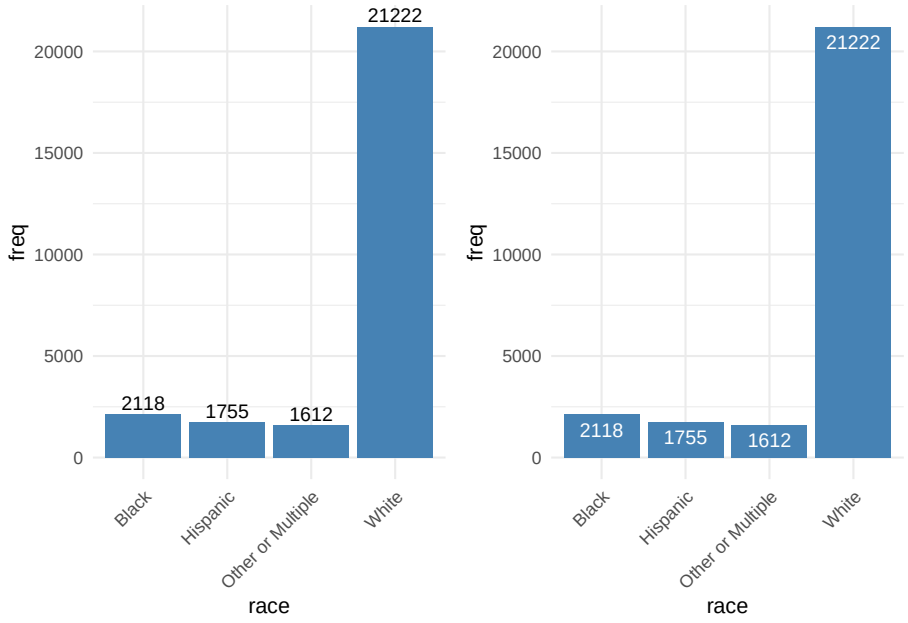
```
        1))
# 选择要显示的项目
p5 <- p + scale_x_discrete(limits = c("White", "Black"
    )) + theme(axis.text.x = element_text(angle = 45,
    hjust = 1))
ggarrange(p2, p3, p4, p5)
```



对柱状图进行标签显示

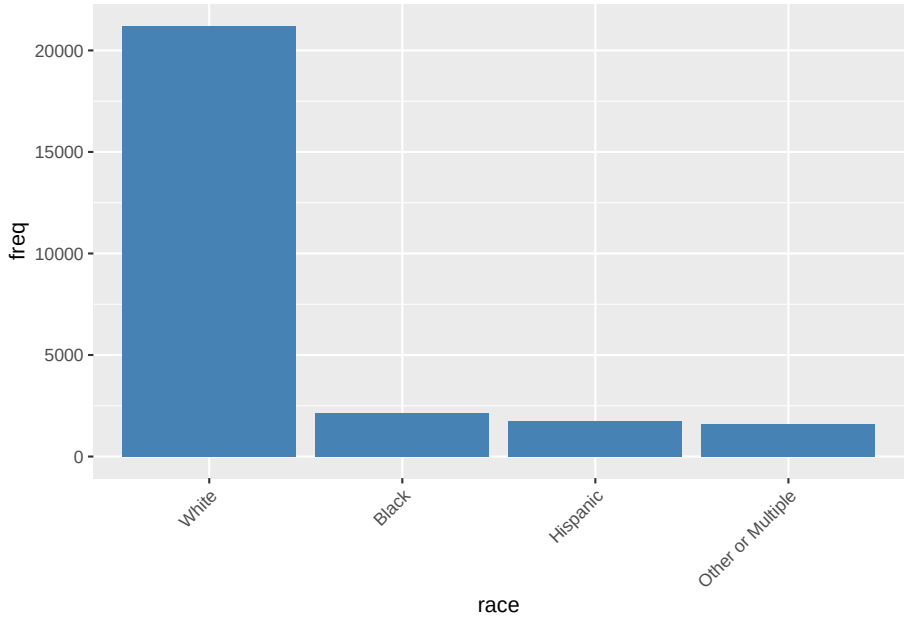
```
p6 <- ggplot(data = data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  geom_text(aes(label = freq), vjust = -0.3, size =
      3.5) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust =
      1))
# 条形内部标签
p7 <- ggplot(data = data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity", fill = "steelblue") +
```

```
  geom_text(aes(label = freq), vjust = 1.6, color = "
      white", size = 3.5) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust =
      1))
ggarrange(p6, p7, nrow = 1)
```



如果觉得柱状图的顺序不是你想要的，可以对柱状图的顺序进行修改

```
data <- within(data, {
  race <- factor(race, levels = c("White", "Black", "
      Hispanic", "Other or Multiple"))
})
ggplot(data, aes(x = race, y = freq)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  theme(axis.text.x = element_text(angle = 45, hjust =
      1))
```



ggplot2 关于柱状图的相关做法有很详细的介绍，相关参考链接：https://ggplot2.tidyverse.org/reference/geom_bar.html

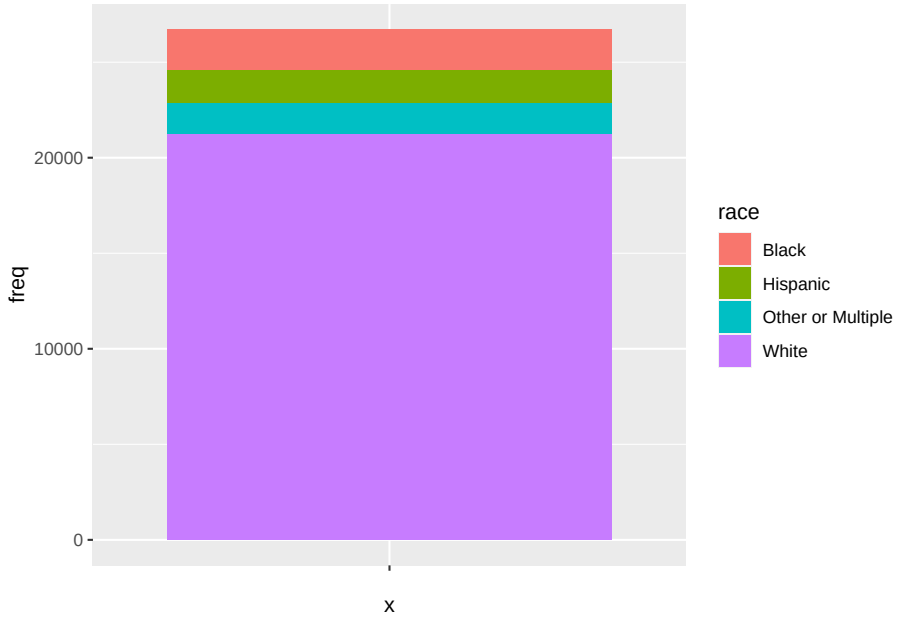
0.4.5 饼状图

饼状图作为常用的数据可视化图形之一，广泛的使用在各个领域，能够很清楚展示数据的所占的百分比。ggplot2 并没有类似于 geom_pie() 这样的函数实现饼图的绘制，但 ggplot2 有一个理念，就是通过极坐标变换绘制饼图

饼图在 ggplot2 中就是通过极坐标变换获得，在绘制饼图之前需要绘制堆叠的条形图，通过将条形图进行极坐标变换后，就能实现饼图绘制了。

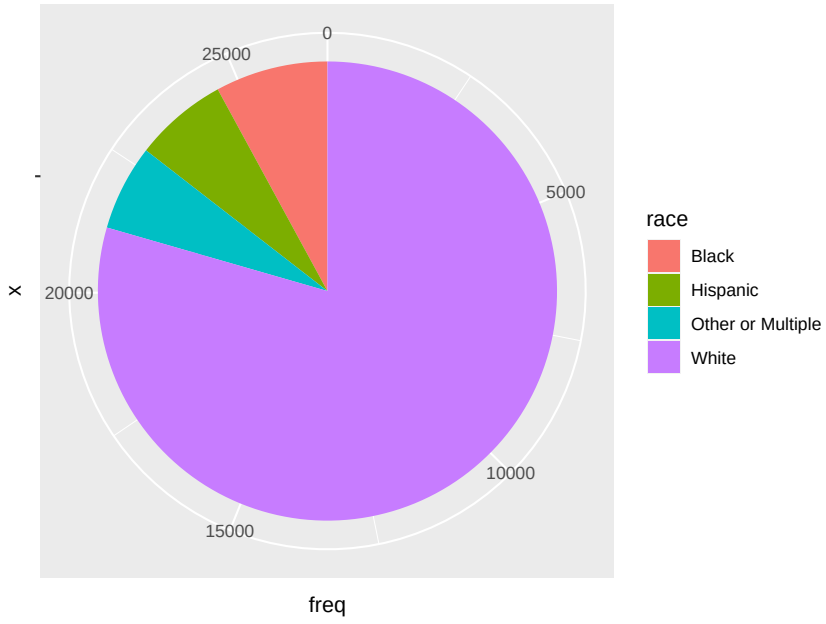
对 h1n1 问卷表中 race 数据进行数据展示

```
data <- count(h1n1_data["race"])
ggplot(data = data, aes(x = "", y = freq, fill = race)
    ) +
  geom_bar(stat = "identity")
```



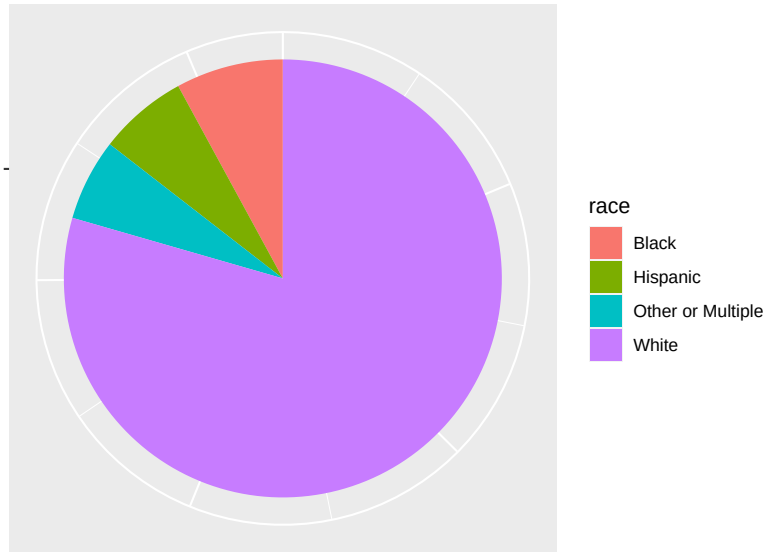
堆叠的条形图绘制完后，接下来就需要进行极坐标变换了，ggplot2 中 coord_polar() 函数可以非常方便的实现极坐标变换。

```
ggplot(data = data, aes(x = "", y = freq, fill = race)
    ) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y")
```



看起来像饼图了，但是饼图周围还有多余的数字，如何清除呢？这里的标签其实就是坐标轴的标签，可以通过 labs() 函数将其清除。

```
ggplot(data = data, aes(x = "", y = freq, fill = race)
    ) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  labs(x = "", y = "", title = "") +
  theme(axis.text = element_blank())
```



接下来就是显示各个所占的比例第一种方法，将百分比直接显示在图例中，这种方式适合分类较多的情况。

```
label_value <- paste("(", round(data$freq / sum(data$
    freq) * 100, 1), "%)", sep = "")
label_value
```

```
## [1] "(7.9%)"  "(6.6%)"  "(6%)"    "(79.5%)"
```

将计算的百分比和 race 匹配

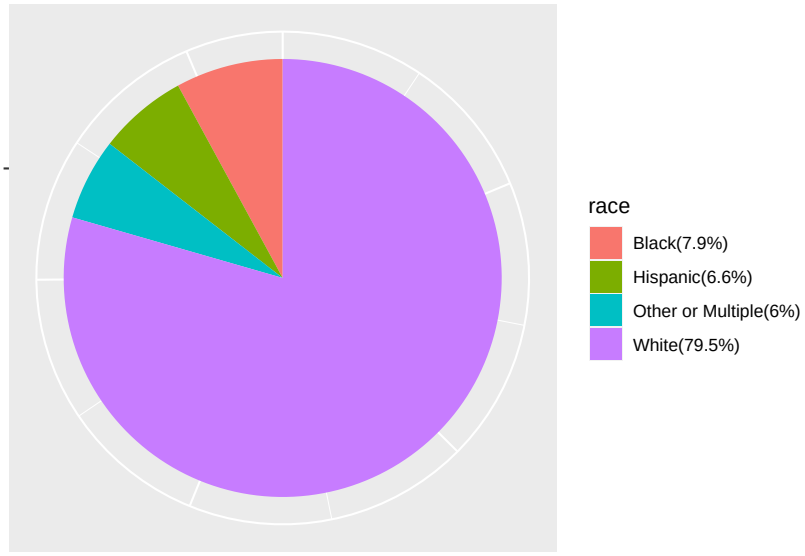
```
label <- paste(data$race, label_value, sep = "")
label
```

```
## [1] "Black(7.9%)"           "Hispanic(6.6%)"
          "Other or Multiple(6%)"
## [4] "White(79.5%)"
```

接下来就是将这些百分比标签放到图例中

```
ggplot(data = data, aes(x = "", y = freq, fill = race)
    ) +
```

```
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  labs(x = "", y = "", title = "") +
  theme(axis.text = element_blank()) +
  scale_fill_discrete(labels = label)
```

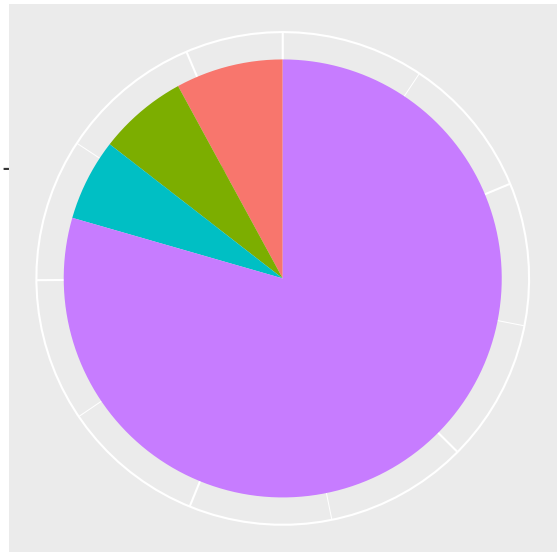


看起来就很不错～

第二种方法，直接将百分比放到各自的饼区中。

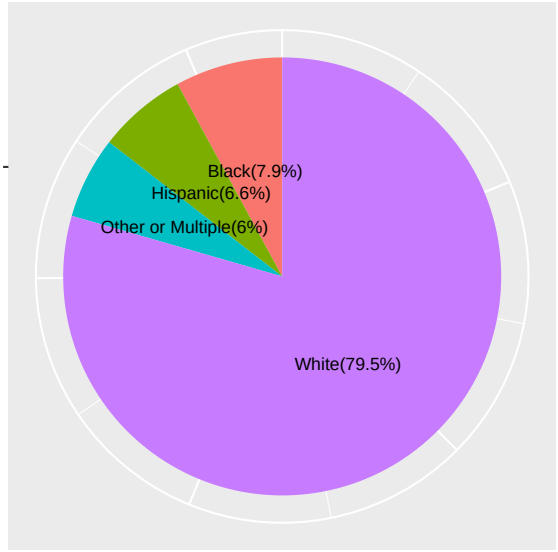
首先是去掉饼图中的图例

```
ggplot(data = data, aes(x = "", y = freq, fill = race)
    ) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  labs(x = "", y = "", title = "") +
  theme(axis.text = element_blank()) +
  theme(legend.position = "none")
```



将标签放置在饼图中

```
ggplot(data = data, aes(x = "", y = freq, fill = race)
    ) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  labs(x = "", y = "", title = "") +
  theme(axis.text = element_blank(), legend.position =
      "none") +
  geom_text(aes(label = label), size = 3, position =
      position_stack(vjust = 0.5))
```



0.4.6 折线图

折线图作为反映数据变化的趋势是常用的数据可视化图形之一，在 ggplot2 中通过 geom_line() 这个函数进行绘制。

对波士顿房价中 rad 进行可视化展示，使用 pylr 包中的 count 对 edcation 进行计数统计

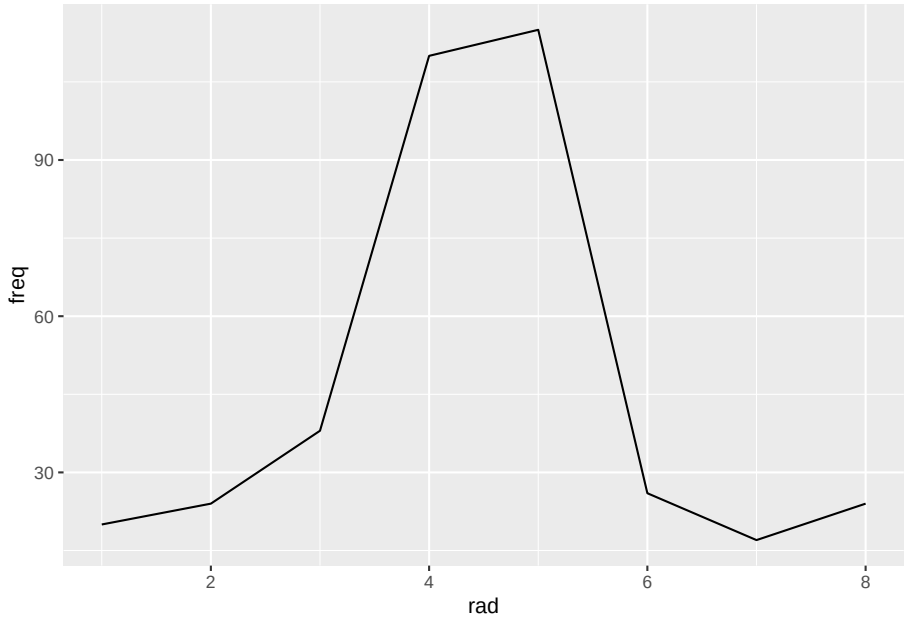
```
data <- count(boston_data["rad"])
data
```

```
##    rad  freq
## 1    1    20
## 2    2    24
## 3    3    38
## 4    4   110
## 5    5   115
## 6    6    26
## 7    7    17
```

```
## 8     8    24
## 9    24   132
```

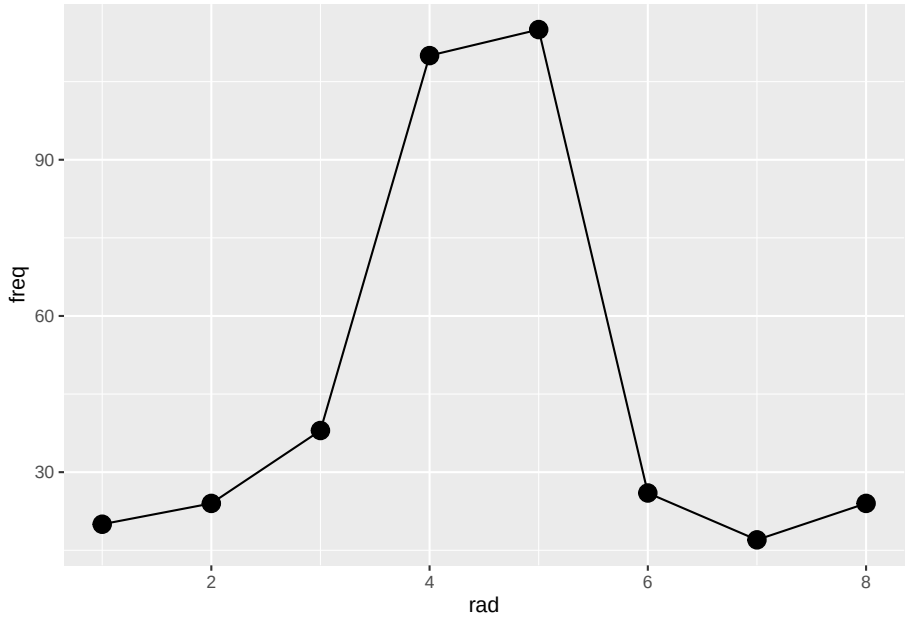
把 rad 为 24 的数据去除掉

```
data <- data[1:8, ]
ggplot(data, aes(x = rad, y = freq)) +
  geom_line()
```



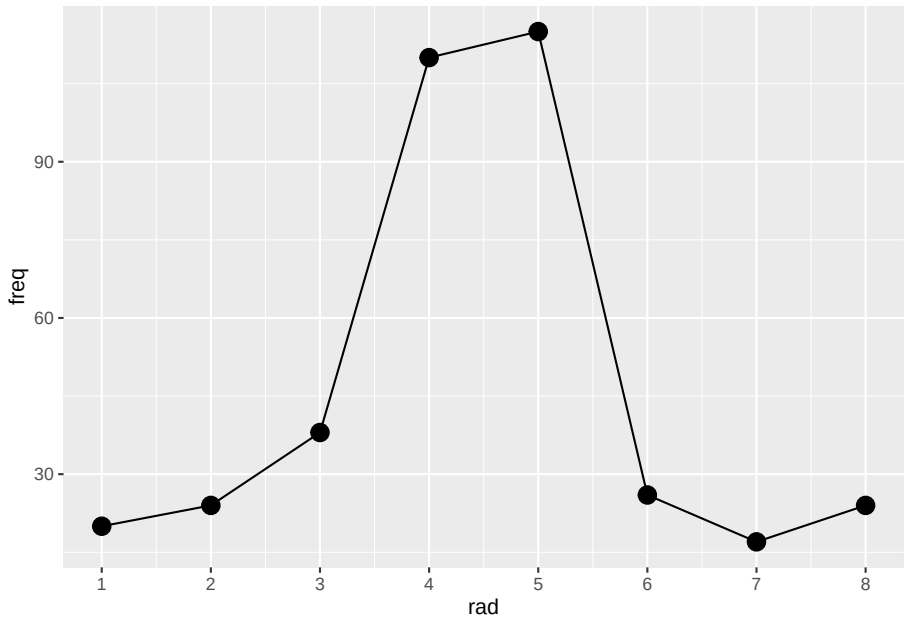
有时候我们需要在折线图上显示对应 x 轴的点数据，从而可以更加清晰的辨别原始数据, 这特别适合数据比较稀疏的情况

```
ggplot(data, aes(x = rad, y = freq)) +
  geom_line() +
  geom_point(size = 4)
```



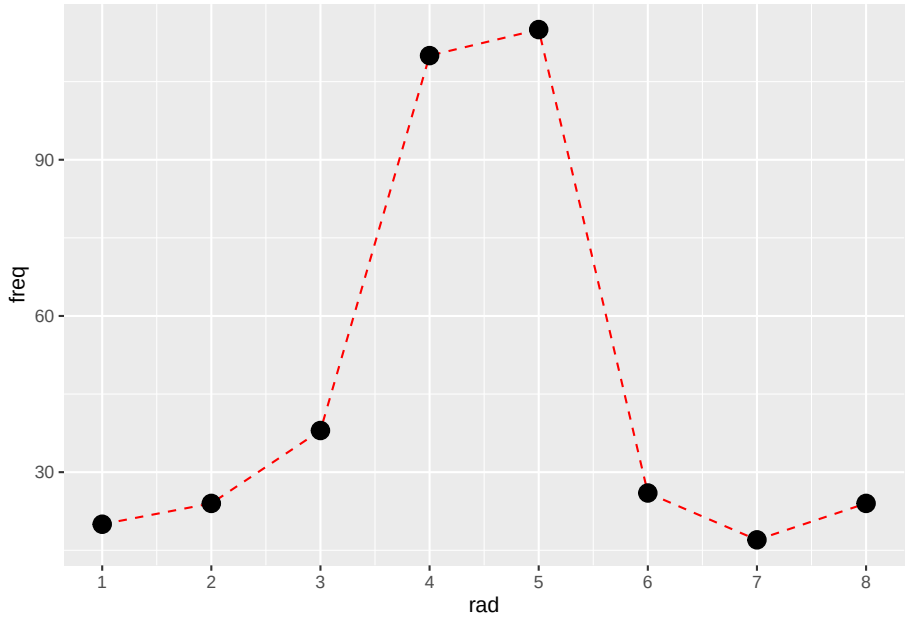
我们调整横坐标的显示刻度

```
ggplot(data, aes(x = rad, y = freq)) +
  geom_line() +
  geom_point(size = 4) +
  scale_x_continuous(breaks = c(1:8))
```



也可以修改线的类型和颜色

```
ggplot(data, aes(x = rad, y = freq)) +
  geom_line(linetype = "dashed", color = "red") +
  geom_point(size = 4) +
  scale_x_continuous(breaks = c(1:8))
```



ggplt2 关于折线图的相关做法的参考链接：https://ggplot2.tidyverse.org/reference/geom_abline.html

0.4.7 ggplot2 扩展包主题

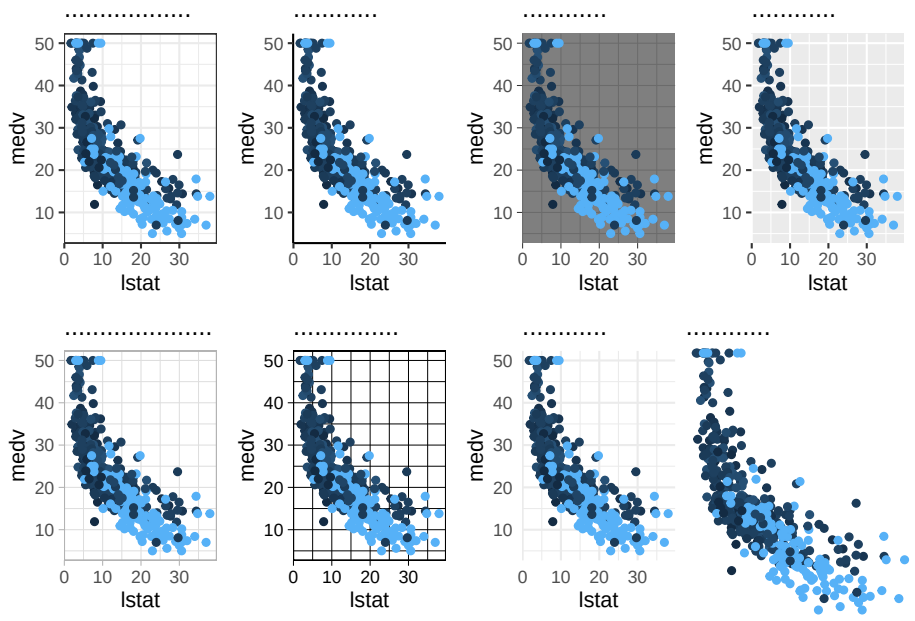
R 语言中的 ggplot2 包里面的风格固定，在需要特殊的图形时，需要更改甚至自定义设置主题。ggplot2 内置了 8 种风格的主题

主题函数	效果
theme_bw()	网格白色主题
theme_classic()	经典主题
theme_dark()	暗色主题，可用于对比
theme_gray()	默认主题
theme_light()	浅色坐标带网格
theme_linedraw()	黑色网格线
theme_minimal()	极简主题
theme_void()	空白主题

我们来试一试不同的主题

```
p <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = rad)) +
  geom_point()
p1 <- p + theme_bw() + labs(title = "网格白色主题") +
    theme(legend.position = "none")
p2 <- p + theme_classic() + labs(title = "经典主题") +
     theme(legend.position = "none")
p3 <- p + theme_dark() + labs(title = "暗色主题") +
    theme(legend.position = "none")
p4 <- p + theme_gray() + labs(title = "默认主题") +
    theme(legend.position = "none")
p5 <- p + theme_light() + labs(title = "浅色坐标带网格
    ") + theme(legend.position = "none")
p6 <- p + theme_linedraw() + labs(title = "黑色网格线"
    ) + theme(legend.position = "none")
p7 <- p + theme_minimal() + labs(title = "极简主题") +
     theme(legend.position = "none")
p8 <- p + theme_void() + labs(title = "空白主题") +
    theme(legend.position = "none")

ggarrange(p1, p2, p3, p4, p5, p6, p7, p8, ncol = 4,
    nrow = 2, heights = 1.2)
```



除了 ggplot2 自带的主题外，还有许多拓展主题包，比如：ggthemes、ggthemr ggthemes 在 cran 上发布，因此推荐使用这个 ggthemr 色彩很好看，因此推荐这个

ggthemes 相关链接：https://github.com/jrnold/ggthemes

ggthemr 相关链接：https://github.com/Mikata-Project/ggthemr

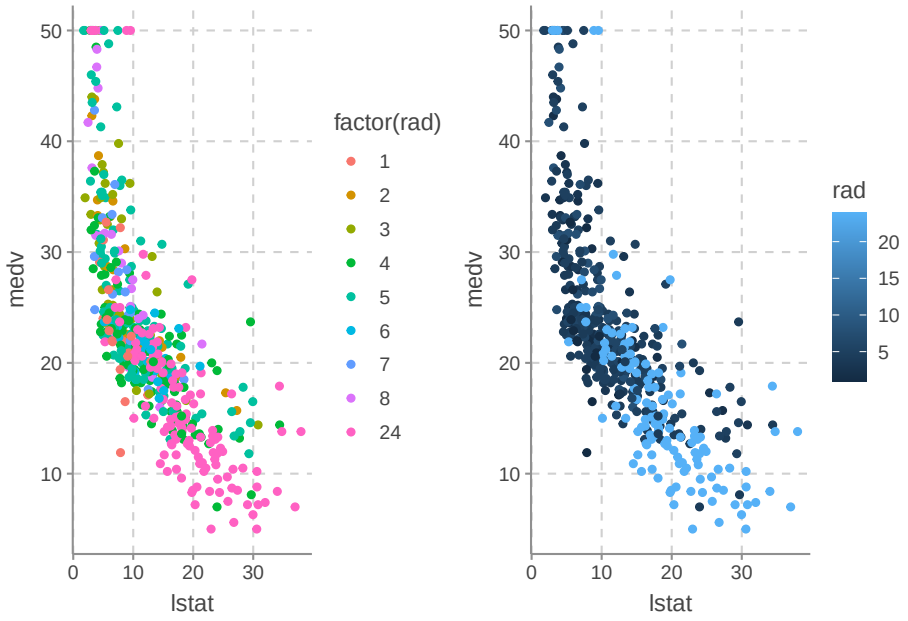
因为 ggthemr 没有上 cran，因此需要通过 github 安装

```
# devtools::install_github('Mikata-Project/ggthemr')
```

使用方法也是非常简单，这里用我比较喜欢的 greyscale 主题方案

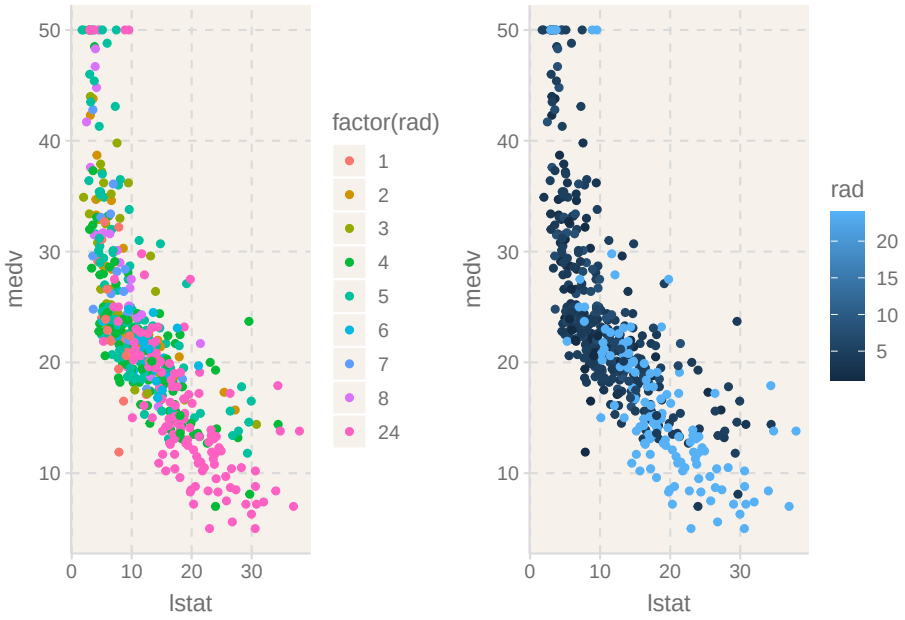
```
library(ggthemr)
ggthemr("greyscale")
p3 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = factor(rad))) +
  geom_point()
p4 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = rad)) +
  geom_point()
```

```
ggarrange(p3, p4, nrow = 1)
```



试一试 light 这个主题，配色非常的温柔

```
library(ggthemr)
ggthemr("light")
p3 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = factor(rad))) +
  geom_point()
p4 <- ggplot(data = boston_data, aes(x = lstat, y =
    medv, colour = rad)) +
  geom_point()
ggarrange(p3, p4, nrow = 1)
```



实战部分：对提供的数据集我们可以试一试 ggthemr 中的不同主题，同时对波士顿房价进行其它的数据可视化的探索。

ggplot2 是一个非常经典的数据可视化 R 包，内容非常丰富，由于篇幅的原因没办法将 ggplot2 中的各种方法全部讲述，因此选择了几个常见的图形进行相关的讲解，以期达到抛砖引玉的效果。如果对 ggplot2 感兴趣的同学，可以去官网进行更加详细的学习，也非常期待大家的数据可视化作品 ~

本章作者

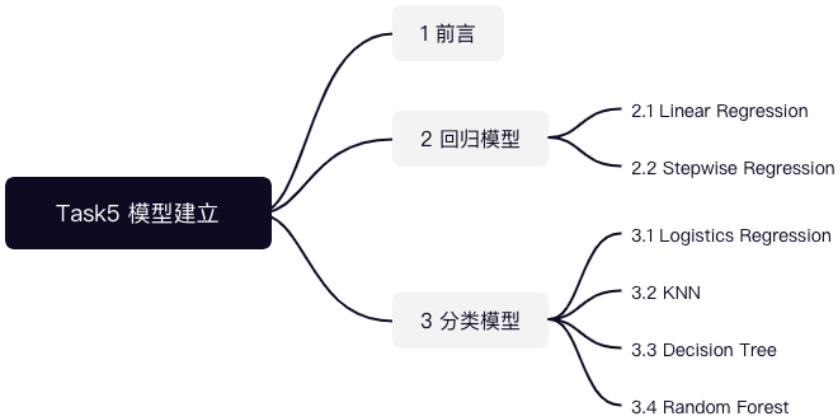
牧小熊

华中农业大学研究生，Datawhale 成员, Datawhale 优秀原创作者
知乎：https://www.zhihu.com/people/muxiaoxiong

关于 Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以 “for the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：

0.5 模型



Task05 共计 3 个知识点，预计需学习 2-3 小时，请安排好学习任务。

0.5.1 前言

为了帮助大家更好的使用 R 语言进行建模分析，本章节将借助波士顿房价数据集来展示常见的模型。本章节学习的目的是帮助大家了解模型的适用范围以及如何建模，不会对模型的底层原理进行深入的研究。并且迫于时间和精力有限，本章节仅介绍部分模型的实现。

- 回归模型：回归模型是一种有监督的、预测性的建模技术，它研究的是因变量和自变量之间的关系。
- 分类模型：分类模型也是一种有监督的机器学习模型。与回归模型不同的是，其标签 (因变量) 通常是有限个数的定类变量。最常见的是二分类模型。

我们主要使用波士顿房价数据集来实现各种模型。因此我们使用 2021 作为种子值生成 70% 的数据作为训练集，其余数据作为测试集。下面展示来各个数据集的大小。

```
# 导入BostonHousing数据
library(mlbench)
data(BostonHousing)

# 设置种子值，方便复现
set.seed(2021)

# 生成训练集的索引，用来划分训练集和测试集
train_index <- sample(dim(BostonHousing)[1], 0.7 * dim
    (BostonHousing)[1])
BostonHousingTrain <- BostonHousing[train_index, ]
BostonHousingTest <- BostonHousing[-train_index, ]

# 查看数据集的size
dim(BostonHousing)

## [1] 506  14

dim(BostonHousingTrain)

## [1] 354  14

dim(BostonHousingTest)

## [1] 152  14

# 查看数据集包含的变量名称
names(BostonHousing)

##  [1] "crim"    "zn"      "indus"   "chas"    "nox"
        "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "b"
         "lstat"   "medv"
```

回归模型回归模型有很多主要有 Linear Regression、Logistic Regression、Polynomial Regression、Stepwise Regression、Ridge Regression、Lasso Regression、ElasticNet 等。

本部分主要介绍有 Linear Regression、以及 Stepwise Regression 三种回归模型的实现。

0.5.1.1 Linear Regression

多元线性回归是一种最为基础的回归模型，其使用多个自变量和一个因变量利用 OLS 完成模型训练。下面我们将使用medv作为因变量，剩余变量作为自变量构建模型。

多元线性回归模型使用lm()命令，其中medv~.是回归公式，data=BostonHousingTrain是回归数据。对回归公式的构建进行一些补充，~左侧表示因变量，~右侧表示自变量，多个自变量使用+依次叠加。这里右侧使用了.，该符号的含义是除左侧变量外所有的变量。因此，medv~.等价于medv~crim + zn + indus + chas + nox + rm + age + dis + rad + tax + ptratio + b + medv。

```
# 构建模型，medv~.表示回归方程
lr_model <- lm(medv ~ ., data = BostonHousingTrain)

# summary输出模型汇总
summary(lr_model)
```

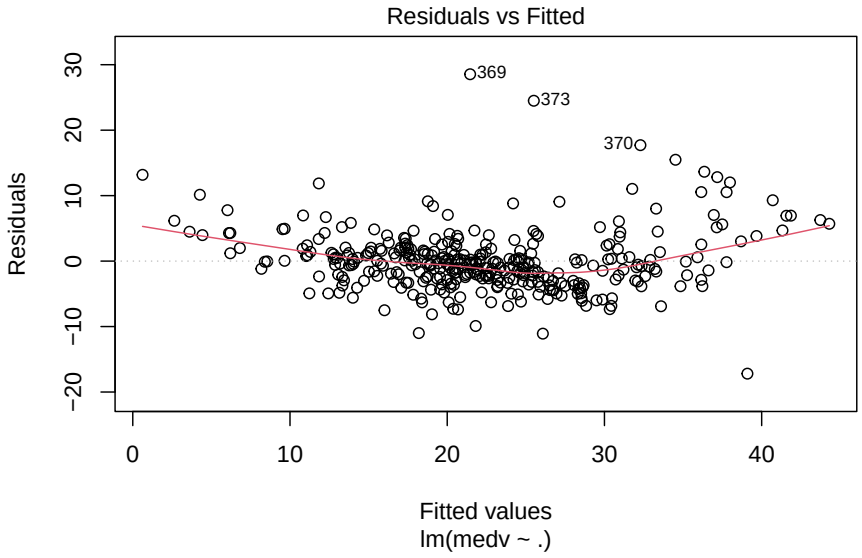
```
##
## Call:
## lm(formula = medv ~ ., data = BostonHousingTrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.1929  -2.6567  -0.3854   1.6261  28.5425
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  28.279554   6.464743   4.374 1.62e-05
    ***
## crim         -0.066574   0.051496  -1.293 0.196958
```

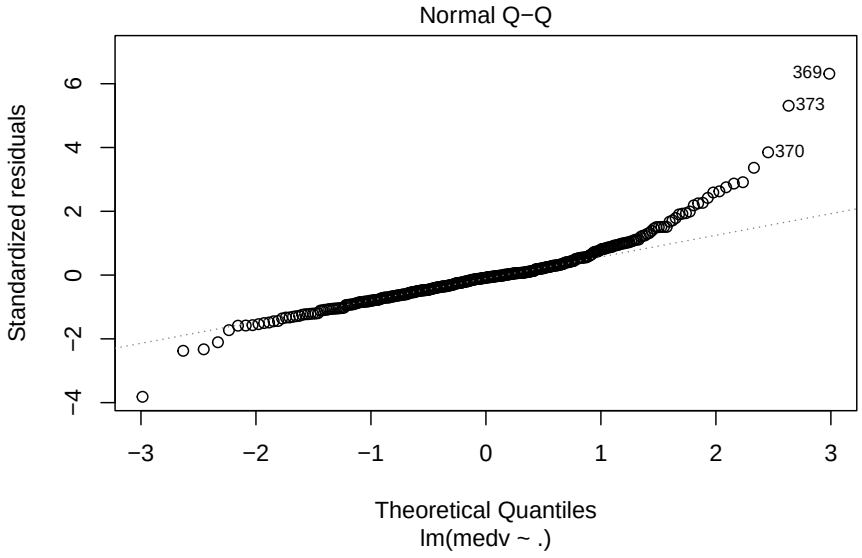
```
## zn              0.031466   0.016525    1.904 0.057733
    .
## indus           0.046583   0.069009    0.675 0.500115
## chas1           3.372501   1.065312    3.166 0.001687
    **
## nox           -14.103937   4.498414   -3.135 0.001866
    **
## rm              4.512687   0.547845    8.237 3.85e-15
    ***
## age            -0.010015   0.016016   -0.625 0.532197
## dis            -1.259008   0.245311   -5.132 4.82e-07
    ***
## rad             0.263841   0.077147    3.420 0.000702
    ***
## tax            -0.012026   0.004176   -2.880 0.004235
    **
## ptratio        -1.008997   0.160048   -6.304 8.99e-10
    ***
## b               0.014361   0.003406    4.217 3.18e-05
    ***
## lstat          -0.466948   0.062026   -7.528 4.66e-13
    ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
    '.' 0.1 ' ' 1
##
## Residual standard error: 4.776 on 340 degrees of
    freedom
## Multiple R-squared:  0.7299, Adjusted R-squared:
    0.7196
## F-statistic: 70.67 on 13 and 340 DF,  p-value: <
    2.2e-16
```

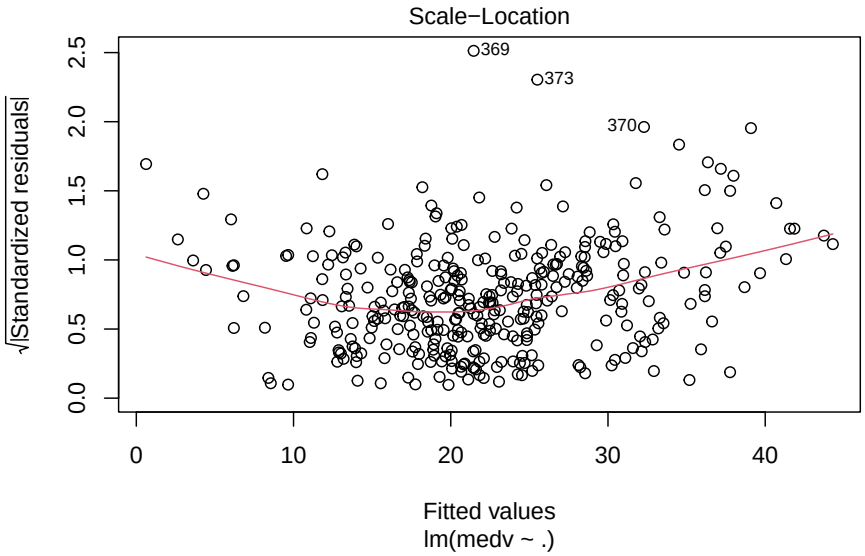
运用 plot 命令对模型进行诊断，各图含义参考 https://www.cnblogs.com/

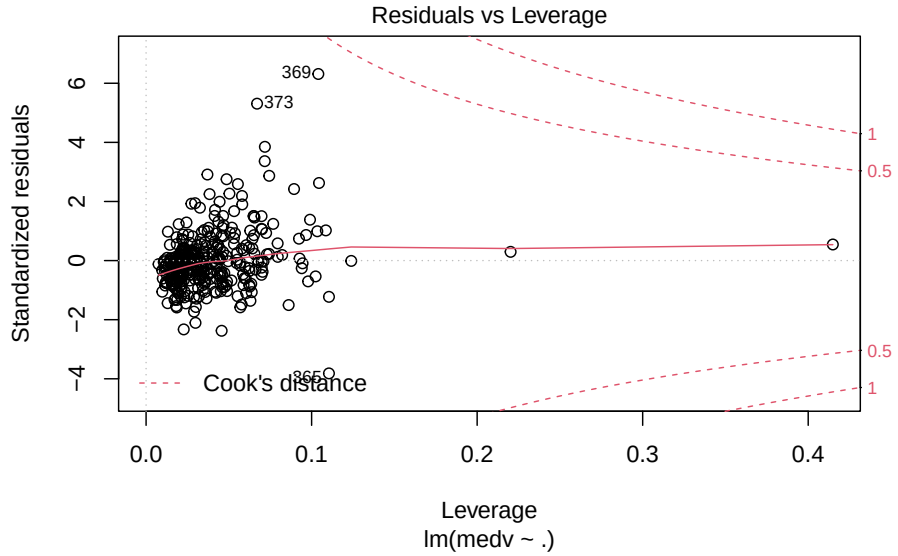
lafengdatascientist/p/5554167.html

plot(lr_**model**)









predict命令能够基于已经训练好的模型进行预测。

```
# 根据模型对新数据进行预测
BostonHousingTest$lr_pred <- predict(lr_model, newdata
```

```
    = BostonHousingTest)
```

0.5.1.2 Stepwise Regression

利用逐步回归分析可以对模型中的变量进行优化。R 语言中的step()命令，是以 AIC 信息统计量为准则，通过选择最小的 AIC 信息统计量来达到提出或添加变量的目的。

对于逐步回归，一般有前向、后向、双向等逐步方式。本部分将基于已经实现的lr_model进行双向逐步回归。前向和后向回归只需要更改step()命令行中的direstion参数即可。具体内容参照 https://blog.csdn.net/qq_38204302/article/details/86567356

```
# both逐步回归
step_model <- step(lr_model, direction = "both")
```

```
## Start:  AIC=1120.78
## medv ~ crim + zn + indus + chas + nox + rm + age +
    dis + rad +
##     tax + ptratio + b + lstat
##
##           Df Sum of Sq    RSS    AIC
## - age      1      8.92 7765.1 1119.2
## - indus    1     10.39 7766.6 1119.3
## - crim     1     38.13 7794.3 1120.5
## <none>                 7756.2 1120.8
## - zn       1     82.71 7838.9 1122.5
## - tax      1    189.16 7945.4 1127.3
## - nox      1    224.25 7980.5 1128.9
## - chas     1    228.62 7984.8 1129.1
## - rad      1    266.82 8023.0 1130.8
## - b        1    405.60 8161.8 1136.8
## - dis      1    600.89 8357.1 1145.2
## - ptratio  1    906.67 8662.9 1157.9
## - lstat    1   1292.88 9049.1 1173.4
```

```
## - rm       1    1547.84 9304.0 1183.2
##
## Step:  AIC=1119.19
## medv ~ crim + zn + indus + chas + nox + rm + dis +
    rad + tax +
##     ptratio + b + lstat
##
##           Df Sum of Sq    RSS    AIC
## - indus    1     10.22 7775.3 1117.7
## - crim     1     39.31 7804.4 1119.0
## <none>                 7765.1 1119.2
## + age      1      8.92 7756.2 1120.8
## - zn       1     92.34 7857.5 1121.4
## - tax      1    193.70 7958.8 1125.9
## - chas     1    225.98 7991.1 1127.3
## - nox      1    261.86 8027.0 1128.9
## - rad      1    278.77 8043.9 1129.7
## - b        1    398.83 8164.0 1134.9
## - dis      1    613.30 8378.4 1144.1
## - ptratio  1    916.06 8681.2 1156.7
## - lstat    1   1546.55 9311.7 1181.5
## - rm       1   1571.42 9336.5 1182.4
##
## Step:  AIC=1117.65
## medv ~ crim + zn + chas + nox + rm + dis + rad +
    tax + ptratio +
##     b + lstat
##
##           Df Sum of Sq    RSS    AIC
## - crim     1     41.19 7816.5 1117.5
## <none>                 7775.3 1117.7
## + indus    1     10.22 7765.1 1119.2
## + age      1      8.74 7766.6 1119.3
```

```
## - zn        1     88.58 7863.9 1119.7
## - tax       1    189.88 7965.2 1124.2
## - chas      1    231.63 8007.0 1126.0
## - nox       1    252.32 8027.7 1127.0
## - rad       1    269.59 8044.9 1127.7
## - b         1    395.78 8171.1 1133.2
## - dis       1    706.93 8482.3 1146.5
## - ptratio   1    906.25 8681.6 1154.7
## - lstat     1   1537.69 9313.0 1179.5
## - rm        1   1561.38 9336.7 1180.4
##
## Step:  AIC=1117.52
## medv ~ zn + chas + nox + rm + dis + rad + tax +
    ptratio + b +
##     lstat
##
##            Df Sum of Sq    RSS    AIC
## <none>                  7816.5 1117.5
## + crim      1     41.19 7775.3 1117.7
## + indus     1     12.10 7804.4 1119.0
## - zn        1     76.92 7893.5 1119.0
## + age       1      9.92 7806.6 1119.1
## - tax       1    182.40 7998.9 1123.7
## - rad       1    228.86 8045.4 1125.7
## - nox       1    236.90 8053.4 1126.1
## - chas      1    240.06 8056.6 1126.2
## - b         1    514.43 8331.0 1138.1
## - dis       1    673.74 8490.3 1144.8
## - ptratio   1    893.27 8709.8 1153.8
## - lstat     1   1589.98 9406.5 1181.1
## - rm        1   1636.60 9453.1 1182.8
```

summary(**step_model**)

```
##
## Call:
## lm(formula = medv ~ zn + chas + nox + rm + dis +
    rad + tax +
##     ptratio + b + lstat, data = BostonHousingTrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.8955  -2.6773  -0.4005   1.6707  28.5842
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  27.001770   6.354437   4.249 2.77e-05
    ***
## zn            0.029797   0.016219   1.837  0.06705
    .
## chas1         3.446516   1.061891   3.246  0.00129
    **
## nox         -13.578105   4.211269  -3.224  0.00138
    **
## rm            4.491255   0.529976   8.474 7.07e-16
    ***
## dis          -1.213451   0.223170  -5.437 1.03e-07
    ***
## rad           0.220392   0.069546   3.169  0.00167
    **
## tax          -0.010818   0.003824  -2.829  0.00494
    **
## ptratio      -0.991885   0.158427  -6.261 1.14e-09
    ***
## b             0.015446   0.003251   4.751 2.98e-06
    ***
## lstat        -0.482234   0.057733  -8.353 1.67e-15
```

```
    ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
    '.' 0.1 ' ' 1
##
## Residual standard error: 4.774 on 343 degrees of
    freedom
## Multiple R-squared:  0.7278, Adjusted R-squared:
    0.7199
## F-statistic: 91.71 on 10 and 343 DF,  p-value: <
    2.2e-16
```

对于分类模型还有较为常用的 Lasso Regression 和 Ridge Regression，我们将会在进阶教程中来更加具体的讲解模型知识。

0.5.2 分类模型

在进行分类模型前，我们需要构建分类标签。我们使用medv的中位数进行划分，其中 1 表示高房价，0 表示低房价。通过这样的转化将原本的数值型变量转化为二元标签。并使用相同的种子值划分测试集和训练集。

```
# 将连续变量转化成二分类变量
BostonHousing$medv <- as.factor(ifelse(BostonHousing$
    medv > median(BostonHousing$medv), 1, 0))
# 查看两种变量类别的数量
summary(BostonHousing$medv)
```

```
##   0   1
## 256 250
```

```
# 使用相同的种子值，复现训练集合测试集的划分
set.seed(2021)
train_index <- sample(dim(BostonHousing)[1], 0.7 * dim
    (BostonHousing)[1])
BostonHousingTrain <- BostonHousing[train_index, ]
```

```
BostonHousingTest <- BostonHousing[-train_index, ]
```

同时引入两个计算函数，用来计算 AUC 指标值。

```
# 引入auc计算函数
library("ROCR")
calcAUC <- function(predcol, outcol) {
  perf <- performance(prediction(predcol, outcol == 1)
      , "auc")
  as.numeric(perf@y.values)
}
```

0.5.2.1 Logistics Regression

逻辑回归是一种广义的线性回归分析模型，利用 sigmode 将线性回归结果转化成概率的形式。下面展示了利用glm()构建逻辑回归的过程。通过计算，训练集上的 auc 取值为 0.9554211，测试集上的 auc 取值为 0.9506969，说明模型效果整体不错。

```
# 逻辑回归模型构建
lr_model <- glm(medv ~ ., data = BostonHousingTrain,
    family = binomial(link = "logit"))
summary(lr_model)
```

```
##
## Call:
## glm(formula = medv ~ ., family = binomial(link = "
    logit"), data = BostonHousingTrain)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.00065  -0.34945  -0.01094   0.24116   3.00080
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)  4.641164   4.937497   0.940 0.347226
## crim        -0.053419   0.096982  -0.551 0.581760
## zn           0.005680   0.015218   0.373 0.708951
## indus        0.045677   0.048167   0.948 0.342973
## chas1        1.634949   0.798937   2.046 0.040717 *
## nox         -6.916586   3.286514  -2.105 0.035332 *
## rm           2.876778   0.651573   4.415 1.01e-05
    ***
## age         -0.034146   0.013493  -2.531 0.011383 *
## dis         -0.696695   0.209391  -3.327 0.000877
    ***
## rad          0.220168   0.074211   2.967 0.003009
    **
## tax         -0.009724   0.003446  -2.822 0.004769
    **
## ptratio     -0.611081   0.132894  -4.598 4.26e-06
    ***
## b            0.006135   0.003830   1.602 0.109159
## lstat       -0.267857   0.064765  -4.136 3.54e-05
    ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05
    '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to
    be 1)
##
##     Null deviance: 489.83  on 353  degrees of
    freedom
## Residual deviance: 187.85  on 340  degrees of
    freedom
## AIC: 215.85
##
```

```
## Number of Fisher Scoring iterations: 7

# 分别对训练集和测试集进行预测
lr_pred_train <- predict(lr_model, newdata =
    BostonHousingTrain, type = "response")
lr_pred_test <- predict(lr_model, newdata =
    BostonHousingTest, type = "response")

# 计算训练集和测试集的auc
calcAUC(lr_pred_train, BostonHousingTrain$medv)

## [1] 0.9554211

calcAUC(lr_pred_test, BostonHousingTest$medv)

## [1] 0.9506969
```

0.5.2.2 KNN

KNN 模型是一种简单易懂、可以用于分类和回归的模型。其中 K 表示在新样本点附近 (距离) 选取 K 个样本数据，通过在 K 个样本进行投票来判断新增样本的类型。

KNN 模型较难的一点是确定超参数 K，目前有一些指标和经验方法帮助确定最优 K 的取值。这部分内容会在后续进行讲解，这里使用 k=25 进行建模。KNN 模型在测试集上的 auc 值为 0.8686411，相比于逻辑回归效果较差。

```
# 导入knn模型的包
library(kknn)

# 构建knn模型
knn <- kknn(medv ~ ., BostonHousingTrain,
    BostonHousingTest, k = 25)

# 预测并计算测试集上的auc取值
```

```
knn_pred_test <- predict(knn, newdata =
    BostonHousingTest)
calcAUC(as.numeric(knn_pred_test), BostonHousingTest$
    medv)
```

```
## [1] 0.875784
```

0.5.2.3 Decision Tree

决策树是一种基于树模型进行划分的分类模型，通过一系列 if then 决策规则的集合，将特征空间划分成有限个不相交的子区域，对于落在相同子区域的样本，决策树模型给出相同的预测值。下面构建了决策树的分类模型

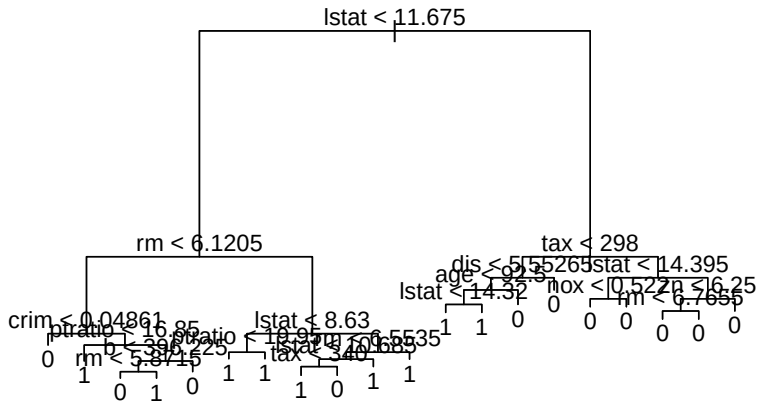
```
# 导入包
library(tree)

# 构建决策树模型函数，medv~.是决策树公式，用来表明变
    量。
# summary输出模型汇总信息
dt_model <- tree(medv ~ ., BostonHousingTrain)
summary(dt_model)
```

```
##
## Classification tree:
## tree(formula = medv ~ ., data = BostonHousingTrain)
## Variables actually used in tree construction:
##  [1] "lstat"   "rm"      "crim"    "ptratio" "b"
        "tax"     "dis"
##  [8] "age"     "nox"     "zn"
## Number of terminal nodes:  20
## Residual mean deviance:  0.2984 = 99.66 / 334
## Misclassification error rate: 0.07062 = 25 / 354
```

```
# plot可以对树模型进行绘制，但可能会出现书分支过多的情
    况。
```

```
plot(dt_model)
text(dt_model)
```



在构建决策树模型的基础上，分别对训练集和测试集进行预测并计算 auc 取值。该模型在训练集上的 auc 取值为 0.9281874，在测试集上的 auc 取值为 0.8789199。训练集和测试集间存在抖动，说明该模型可能出现过拟合。我们需要引入剪枝的操作来降低模型的过拟合，这部分供同学们自学。

```
# 预测
dt_pred_train <- predict(dt_model, newdata =
    BostonHousingTrain, type = "class")
dt_pred_test <- predict(dt_model, newdata =
    BostonHousingTest, type = "class")

# 计算auc取值
calcAUC(as.numeric(dt_pred_train), BostonHousingTrain$
    medv)
```

```
## [1] 0.9308756
```

```
calcAUC(as.numeric(dt_pred_test), BostonHousingTest$
    medv)
```

```
## [1] 0.8789199
```

0.5.2.4 Random Forest

随机森林是一个包含多个决策树的分类器，可以用于分类和回归问题。在解决分类问题是，其输出的类别是由个别树输出的类别的众数而定。相比于单树模型，随机森林具有更好地泛化能力。

使用randomForest()构建模型的过程中，可以通过ntree设定随机森林中包含的决策树数量。由于随机森林是对样本和变量的随机，因此可以通过important展示变量的重要性排序。通过模型预测，随机森林模型在训练集上的 auc 为 0.9615975，在测试集上的 auc 为 0.9247387。

```
# 导入随机森林包
library(randomForest)

# 随机森林模型
rf_model <- randomForest(medv ~ ., BostonHousingTrain,
    ntree = 100, nodesize = 10, importance = T)
# 展示模型变量的重要性
importance(rf_model)
```

```
##                0          1 MeanDecreaseAccuracy
    MeanDecreaseGini
## crim      3.0460631  1.5455430            3.9486776
          5.762997
## zn        3.1035729  1.5721594            3.6238915
          1.886801
## indus     3.8338867  1.4335357            4.6616469
          7.176498
## chas      1.6703290 -1.5235785            0.7998773
          1.100619
```

```
## nox       4.6899935  4.2616418           6.3944503
          16.005287
## rm       11.0161057 10.2260377          14.5799077
          24.681409
## age       5.6799908  3.3897131           6.9069090
           9.107270
## dis       4.2225512  3.8567841           6.1001670
           8.419924
## rad       0.9290789 -0.3819842           0.8369308
           1.449089
## tax       1.1409763  7.2597262           7.5416998
           8.688504
## ptratio   3.4528462  5.8912306           6.5636512
          11.890037
## b        -0.4174669  4.4680208           3.3717663
           3.990056
## lstat    14.5324793 12.5910741          18.7108835
          44.289292
```

```
# 预测
rf_pred_train <- predict(rf_model, newdata =
    BostonHousingTrain, type = "class")
rf_pred_test <- predict(rf_model, newdata =
    BostonHousingTest, type = "class")

# 计算auc取值
calcAUC(as.numeric(rf_pred_train), BostonHousingTrain$
    medv)

## [1] 0.9675499

calcAUC(as.numeric(rf_pred_test), BostonHousingTest$
    medv)

## [1] 0.9236934
```

思考与练习

本章节仅对模型进行简单介绍，更多详细、复杂的模型将在后面的进阶课程中展开。

学习完本章节，希望你能够尝试一些模型调优工作。如决策树剪枝，如尝试搜索 KNN 模型中最佳 K 取值等。

本章作者

张晋

Datawhale 成员，算法竞赛爱好者
https://blog.csdn.net/weixin_44585839/

关于 Datawhale

Datawhale 是一个专注于数据科学与 AI 领域的开源组织，汇集了众多领域院校和知名企业的优秀学习者，聚合了一群有开源精神和探索精神的团队成员。Datawhale 以 “for the learner，和学习者一起成长” 为愿景，鼓励真实地展现自我、开放包容、互信互助、敢于试错和勇于担当。同时 Datawhale 用开源的理念去探索开源内容、开源学习和开源方案，赋能人才培养，助力人才成长，建立起人与人，人与知识，人与企业和人与未来的联结。本次数据挖掘路径学习，专题知识将在天池分享，详情可关注 Datawhale：