第 二 章　数据操作

2.1 tidyverse 简介与管道

2.1.1 tidyverse 包简介

tidyverse 包是 Hadley Wickham 及团队的集大成之作，是专为数据科学而开发的一系列包的合集，**基于整洁数据，提供了一致的底层设计哲学、语法、数据结构**。

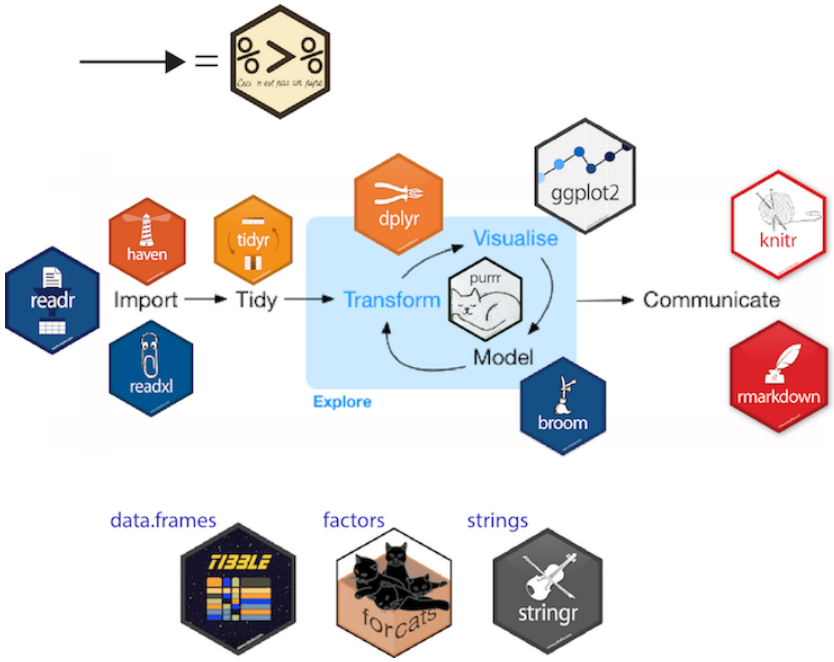


图 2.1: tidyverse 核心包

tidyverse 用 “现代的”、“优雅的” 方式，以**管道式、泛函式**编程技术实现了数据科学的整个流程：**数据导入、数据清洗、数据操作、数据可视化、数据建模、可重现与交互报告**。

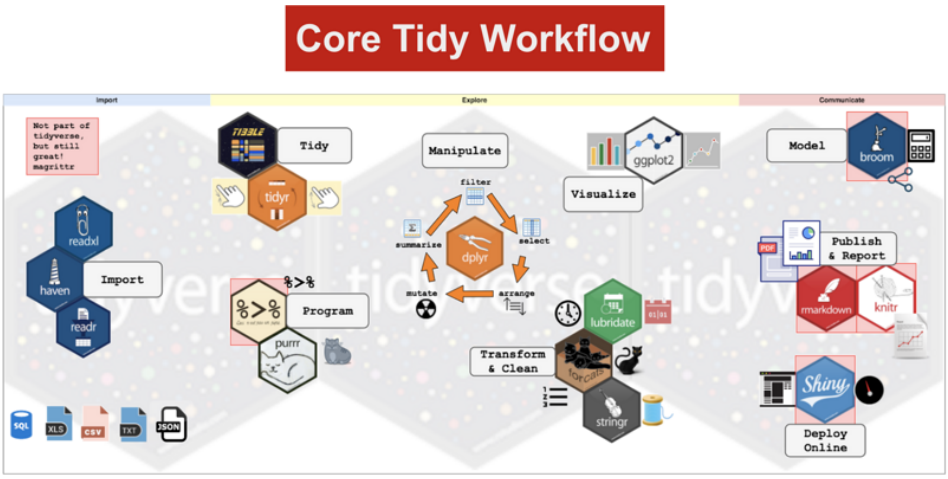


图 2.2: tidyverse 整洁工作流

在 tidyverse 包的引领下，近年来涌现出一系列具体研究领域的 tidy* 版本的包，比较综合的有 tidymodels（机器学习）、rstatix（统计）、tidybayes（贝叶斯模型）、tidyquant（金融）、fpp3（时间序列）、tidytext（文本挖掘）、sf（空间数据分析）、tidybulk（生信）等。

tidyverse 与 data.table

tidyverse 操作数据语法优雅、容易上手，但效率与主打高效的 data.table 包不可同日而语，处理几 G 甚至几十 G 的数据，需要用 data.table.

但 data.table 的语法高度抽象、不容易上手。本书不对 data.table 做过多展开，其简单使用可参阅附录 2。另一种不错的方案是使用专门的转化包：有不少包尝试底层用 data.table，上层用 tidyverse 语法包装（转化），如 dtplyr, tidyfst 等。

2.1.2 管道操作

1. 什么是管道操作？

magrittr 包引入了管道操作，能够通过管道将数据从一个函数传给另一个函数，从而用若干函数构成的**管道**依次变换你的数据。

例如，对数据集 `mtcars`，先按分类变量 `cyl` 分组，再对连续变量 `mpg` 做分组汇总计算均值：

```
library(tidyverse)
mtcars %>%
  group_by(cyl) %>%
  summarise(mpg_avg = mean(mpg))
```

```
## # A tibble: 3 x 2
##     cyl mpg_avg
##   <dbl>   <dbl>
## 1     4    26.7
## 2     6    19.7
## 3     8    15.1
```

管道运算符 `%>%` 的意思是：将左边的运算结果，以输入的方式传给右边函数。若干个函数通过管道链接起来，叫做**管道**（**pipeline**）。

```
x %>% f() %>% g()                    # 等同于 g(f(x))
```

对该管道示例应该这样理解：**依次对数据进行若干操作：先对 x 进行 f 操作，接着对结果进行 g 操作。**

管道，也支持 R base 函数：

```
month.abb %>%                    # 内置月份名缩写字符向量
  sample(6) %>%
  tolower() %>%
  str_c(collapse = "|")
```

```
## [1] "aug|sep|jul|feb|dec|oct"
```

使用管道的好处是：

- 避免使用过多的中间变量；
- 程序可读性大大增强：
- 管道操作的过程，读起来就是对原数据集依次进行一系列操作的过程。而非管道操作，读起来与操作的过程是相反的，比如同样实现上例：

```
str_c(tolower(sample(month.abb, 6)), collapse="|")
```

2. 常用管道操作

管道默认将数据传给下一个函数的第 1 个参数，且它可以省略

```
c(1, 3, 4, 5, NA) %>% mean(., na.rm = TRUE)      # "." 可以省略
c(1, 3, 4, 5, NA) %>% mean(na.rm = TRUE)         # 建议写法
```

这种机制使得管道代码看起来就是：从数据开始，依次用函数对数据施加一系列的操作（变换数据），各个函数都直接从非数据参数开始写即可，而不用再额外操心数据的事情，数据会自己沿管道向前“流动”。

正是这种管道操作，使得 tidyverse 能够优雅地操作数据。

所以，tidyverse 中的函数都设计为数据作为第 1 个参数，自定义的函数也建议这样做。

数据可以在下一个函数中使用多次

数据经过管道默认传递给函数的第 1 个参数（通常直接省略）；若在非第 1 个参数处使用该数据，必须用“.”代替（绝对不能省略），这使得管道作用更加强大和灵活。下面看一些具体实例：

```
# 数据传递给 plot 第一个参数作为绘图数据 (. 省略),
# 同时用于拼接成字符串给 main 参数用于图形标题
c(1, 3, 4, 5) %>% plot(main = str_c(., collapse=","))

# 数据传递给第二个参数 data
mtcars %>% plot(mpg ~ disp, data = .)
```

```
# 选择列
iris %>% .$Species                    # 选择 Species 列内容
iris %>% .[1:3]                       # 选择 1-3 列子集
```

再来看一个更复杂的例子：

```
mtcars %>%
  split(.$cyl) %>%                    # . 相当于 mtcars
  map(~ lm(mpg ~ wt, data = .x))
```

split() 是将数据框 mtcars 根据其 cyl 列（包含 3 个水平的分类变量）分组，得到包含 3 个成分的列表；列表接着传递给 map(.x, .f) 的第一个参数（直接省略），~ lm(mpg ~ wt, data = .x) 是第二参数 .f，为 purrr 风格公式写法。

整体来看，实现的是分组建模：将数据框根据分类变量分组，再用 map 循环机制依次对每组数据建立线性回归模型。

> 建议进行区分：. 用于管道操作中代替数据；.x 用于 purrr 风格公式（匿名函数）。

本节部分内容参阅 (Hadley Wickham 2017), (Desi Quintans 2019).

2.2 数据读写

2.2.1 数据读写的包与函数

先来罗列一下读写常见数据文件的包和函数，具体使用可查阅其帮助。

1. readr 包

读写带分隔符的文本文件，如 csv 和 tsv; 也能读写序列化的 R 对象 rds，若想保存数据集后续再加载回来，rds 将保存元数据和该对象的状态，如分组和数据类型。

- 读入数据到数据框：read_csv() 和 read_tsv()
- 读入欧式格式数据[1]：read_csv2() 和 read_tsv2()
- 读写 rds 数据：read_rds() 和 write_rds()
- 写出数据到文件：write_csv(), write_tsv(), write_csv2(), write_tsv2()
- 转化数据类型：parse_number(), parse_logical(), parse_factor() 等

2. readxl 包

专门读取 Excel 文件，包括同一个工作簿中的不同工作表。

[1]欧式格式数据以 ";" 为分隔符，"," 为小数位.

- `read_excel()`: 自动检测 xls 或 xlsx 文件
- `read_xls()`: 读取 xls 文件
- `read_xlsx()`: 读取 xlsx 文件

读写 Excel 文件好用的包，还有 openxlsx.

3. haven 包

读写 SPSS, Stata, SAS 数据文件。

- 读：`read_spss()`, `read_stata()`, `read_sas()`
- 写：`write_spss()`, `write_stata()`, `write_sas()`

4. readtext 包

读取全部文本文件的内容到数据框，每个文件变成一行，常用于文本挖掘[2]或数据收集；readtext 还支持读取 csv, tab, json, xml, html, pdf, doc, docx, rtf, xls, xlsx 等。

- readtext(): 返回数据框，`doc_id` 列为文档标识，`text` 列为读取的全部文本内容（1 个字符串）。

```
library(readtext)
document = readtext("datas/十年一觉.txt")
document
```

```
## readtext object consisting of 1 document and 0 docvars.
## # Description: df[,2] [1 x 2]
##   doc_id       text
##   <chr>        <chr>
## 1 十年一觉.txt "\"     “这位公子爷\"..."
```

2.2.2 数据读写实例

以读取 csv 和 Excel 文件为例演示，读取其它类型的数据文件，换成其它读取函数即可。

[2]做文本挖掘 R 包有 `tidytext`，中文文本挖掘相比英文多了 `jiebaR` 包分词的前期步骤.

```
read_csv(file, col_names, col_types, locale, skip, na, n_max, ...)
```

- file: 数据文件所在相对或绝对路径
- col_names: 第一行是否作为列名
- skip: 开头跳过的行数
- na: 设置什么值解读为缺失值
- n_max: 读取的最大行数
- col_types: 设置列类型[a]，默认 NULL（全部猜测），可总体设置一种类型（循环使用）或为每列单独设置, 例如设置 3 列的列类型：coltypes="cnd"
- locale: 设置区域语言环境（时区, 编码方式, 小数标记、日期格式），主要是用来设置所读取数据文件的编码方式，如从默认"UTF-8"编码改为"GBK"编码：locale = locale(encoding = "GBK")

还有参数 comment（忽略的注释标记），skip_empty_rows 等。

[a] read_csv() 可选列类型："c"（字符型），"i"（整数型），"n"（数值型），"d"（浮点型），"l"（逻辑型），"f"（因子型），"D"（日期型），"T"（日期时间型），"t"（时间型），"?"（猜测该列类型），"_ 或-"（跳过该列）.

```
read_xlsx(path, sheet, range, col_names, col_types, skip, na, n_max, ...)
```

- path: 数据文件所在相对或绝对路径
- sheet: 要读取的工作表
- range: 要读取的单元格范围
- col_names: 第一行是否作为列名
- col_types: 设置列类型[a]，可总体设置一种类型（循环使用）或为每列单独设置，默认 NULL（全部猜测）

也有参数：skip, na, n_max.

[a], read_xlsx() 可选列类型："skip"（跳过该列），"guess"（猜测该列），"logical", "numeric", "date", "text", "list".

readr 包读取数据的函数，默认会保守猜测各列的列类型。若在读取数据时部分列有丢失信息，则建议先将数据以文本（字符）型读取进来，再用 dplyr 修改列类型。

1. 读入单个 csv 文件

```
df = read_csv("datas/六 1 班学生成绩.csv")
df
```

```
## # A tibble: 4 x 6
##   班级  姓名   性别   语文  数学  英语
```

```
##   <chr> <chr>  <chr> <dbl> <dbl> <dbl>
## 1 六1班 何娜   女       87    92    79
## 2 六1班 黄才菊 女       95    77    75
## 3 六1班 陈芳妹 女       79    87    66
## 4 六1班 陈学勤 男       82    79    66
```

2. 批量读取 Excel 文件

批量读取的数据文件往往具有相同的列结构（列名、列类型），读入后紧接着需要按行合并为一个数据框。批量读取并合并，道理很简单，总共分三步：

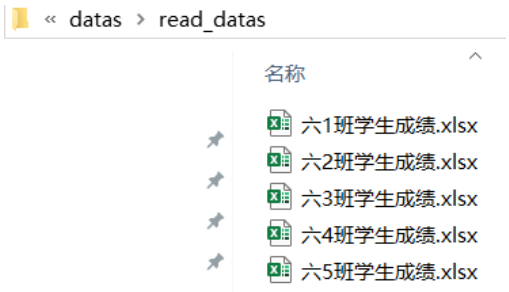
- 获取批量数据文件的路径
- 循环机制批量读取
- 合并成一个数据文件

强大的 purrr 包，使得后两步可以同时做，即借助

```
map_dfr(.x, .f, .id)
```

将函数.f 依次应用到序列.x 的每个元素返回数据框，再 bind_rows 按行合并为一个数据框，.id 可用来增加新列描述来源。

比如，在 read_datas 文件夹下有 5 个 xlsx 文件，每个文件的列名都是相同的：



首先要得到要导入的全部 Excel 文件的完整路径，可以任意嵌套，只需将参数 recursive 设为 TRUE：

```
files = list.files("datas/read_datas", pattern = "xlsx",
                   full.names = TRUE, recursive = TRUE)
files
```

```
## [1] "datas/read_datas/六1班学生成绩.xlsx"
## [2] "datas/read_datas/六2班学生成绩.xlsx"
## [3] "datas/read_datas/六3班学生成绩.xlsx"
## [4] "datas/read_datas/六4班学生成绩.xlsx"
## [5] "datas/read_datas/六5班学生成绩.xlsx"
```

接着，用 map_dfr() 在该路径向量上做迭代，应用 read_xlsx() 到每个文件路径，再按行合并。另外，再多做一步：用 set_names() 将文件路径字符向量创建为命名向量，再结合参数 .id 将路径值作为数据来源列。

```
library(readxl)
df = map_dfr(set_names(files), read_xlsx, .id = " 来源")
head(df)
```

```
## # A tibble: 6 x 7
##   来源                                班级  姓名  性别   语文  数学  英语
##   <chr>                               <chr> <chr> <chr> <dbl> <dbl> <dbl>
## 1 datas/read_datas/六1班学生成绩.xlsx~ 六1班 何娜  女       87    92    79
## 2 datas/read_datas/六1班学生成绩.xlsx~ 六1班 黄才菊~ 女      95    77    75
## 3 datas/read_datas/六1班学生成绩.xlsx~ 六1班 陈芳妹~ 女      79    87    66
## 4 datas/read_datas/六1班学生成绩.xlsx~ 六1班 陈学勤~ 男      82    79    66
## 5 datas/read_datas/六2班学生成绩.xlsx~ 六2班 黄祖娜~ 女      94    88    75
## 6 datas/read_datas/六2班学生成绩.xlsx~ 六2班 徐雅琦~ 女      92    86    72
```

函数 read_xlsx() 的其它控制读取的参数，可直接“作为” map_dfr 参数在后面添加，或改用 purrr 风格公式形式：

```
map_dfr(set_names(files), read_xlsx, sheet = 1, .id = " 来源")   # 或者
map_dfr(set_names(files), ~ read_xlsx(.x, sheet = 1), .id = " 来源")
```

若批量 Excel 数据是来自同一 xlsx 的多个 sheet

还是上述数据，只是在“学生成绩.xlsx” 的 5 个 sheet 中：

	A	B	C	D	E	F
1	班级	姓名	性别	语文	数学	英语
2	六1班	何娜	女	87	92	79
3	六1班	黄才菊	女	95	77	75
4	六1班	陈芳妹	女	79	87	66
5	六1班	陈学勤	男	82	79	66
6						

六1班 | 六2班 | 六3班 | 六4班 | 六5班

```
path = "datas/学生成绩.xlsx"                  # Excel 文件路径
df = map_dfr(set_names(excel_sheets(path)),
             ~ read_xlsx(path, sheet = .), .id = "sheet")
head(df)
```

```
## # A tibble: 6 x 7
##   sheet 班级  姓名  性别   语文  数学  英语
##   <chr> <chr> <chr> <chr>  <dbl> <dbl> <dbl>
```

```
## 1 六1班 六1班 何娜   女      87    92    79
## 2 六1班 六1班 黄才菊 女      95    77    75
## 3 六1班 六1班 陈芳妹 女      79    87    66
## 4 六1班 六1班 陈学勤 男      82    79    66
## 5 六2班 六2班 黄祖娜 女      94    88    75
## 6 六2班 六2班 徐雅琦 女      92    86    72
```

> `excel_sheets()` 函数作用在该 Excel 文件上，提取各个 `sheet` 名字，得到字符向量；然后同样是实现批量读取，只是这次是在 `sheet` 名字的字符向量上循环而已。

3. 写出到一个 Excel 文件

用 `readr` 包中的 `write_csv()` 和 `write_rds()`，或 `writexl` 包中的 `write_xlsx()` 可以保存数据到文件。

以写出到 Excel 文件为例：

```
library(writexl)
write_xlsx(df, "datas/output_file.xlsx")
```

4. 批量写出到多个 Excel 文件

比如有多个数据框，存在一个列表中，依次将它们写入文件，需要准备好文件名；在该数据框列表和文件名上，依次应用写出函数 `write_xlsx()`，又不需要返回值，故适合用 `purrr` 包中的 `walk2()` 函数：

```
df = group_split(Species)        # 鸢尾花按组分割，得到数据框列表
files = paste0("datas/", levels(iris$Species), ".xlsx")  # 准备文件名
walk2(df, files, write_xlsx)
```

5. 保存与载入 rds 数据

除了 `save()` 和 `load()` 函数外，下面以导出数据到 `.rds` 文件为例，因为它能保存数据框及其元数据，如数据类型和分组等。

```
write_rds(iris, "my_iris.rds")
dat = read_rds("my_iris.rds")          # 导入.rds 数据
```

2.2.3 关于中文编码

中文乱码是让很多编程者头痛的问题。

1. 什么是编码？

文字符号在计算机中是用 0 和 1 的字节序列表示的，编码就是将字节序列与所要表示的文字符号建立起映射。

要把各个国家不同的所有文字符号（字符集）正常显示和使用，需要做两件事情：

- 各个国家不同的所有文字符号一一对应地建立数字编码
- 数字编码按一定编码规则用 0-1 表示出来

第一件事情已有一种 Unicode 编码（万国码）来解决：它给全世界所有语言的所有文字符号规定了独一无二的数字编码，字符间分隔的方式是用固定长度字节数。

这样各个国家只需要做第二件事情：为自己国家的所有文字符号设计一种编码规则来表示对应的 Unicode 编码[3]。

> 从 Unicode 到各国具体编码，称为**编码过程**；从各国具体编码到 Unicode，称为**解码过程**。

再来说中国的第二件事情：汉字符号（中文）编码。历史原因产生了多种中文编码，从图来看更直观：

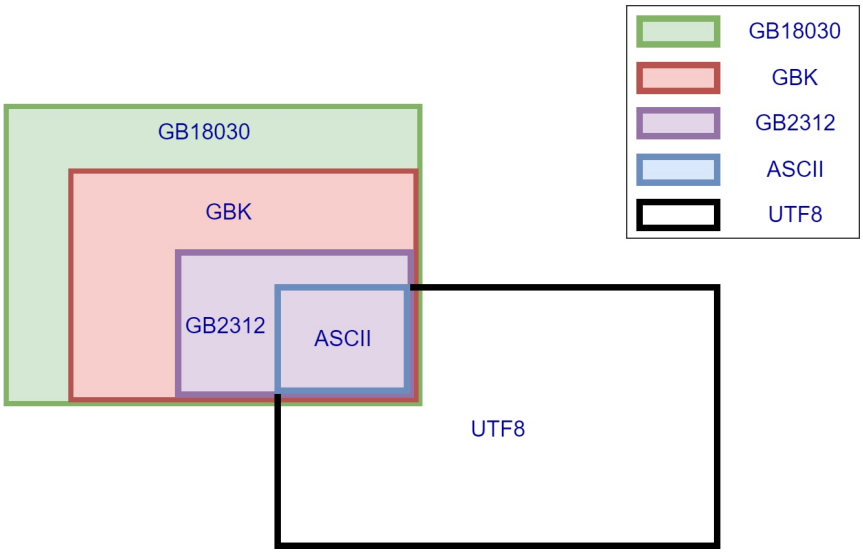


图 2.3: 几种中文编码及兼容性

所谓兼容性，可以理解为子集，同时存在也不冲突。由图 2.3 可见，ASCII（128 个字母和符号，英文够用）被所有编码兼容，而最常见的 UTF-8 与 GBK 之间除了 ASCII 部分之外没有交集。

文件采用什么编码方式，就用什么编码方式打开。只要是用不兼容的编码方式打开文件，就会出现乱码，日常最容易导致乱码场景就是：

[3] Unicode 为了表示“万国”语言，额外增大了存储开销，这第二件事也顺便节省存储开销。

> 用 UTF-8 （GBK）编码方式去读取 GBK（UTF-8）编码的文字，就会出现各种乱码

GBK（国标扩展）系列，根据包含汉字符号从少到多，依次是

- GB2312: 只包含 6763 个汉字
- GBK: 包含 20902 个汉字，基本足够用
- GB18030: 又分 GB1830-2000 和 GB1830-2005，包含七万多个汉字

GBK 编码的汉字基本是 2 字节，节省空间，但只适合国内中文环境。

UTF-8 编码（Unicode 转换格式），是 Unicode 的再表示，支持各个国家的文字符号，兼容性非常好。所以，目前 UTF-8 有**一统天下**的趋势。

UTF-8 是一种变长编码，解决字符间分隔的方式是通过对二进制中最高位连续 1 的个数来决定该字是几字节编码。所有常用汉字的 Unicode 值均可用 3 字节的 UTF-8 表示出来。

UTF-8 通常不带 BOM （字节序标记 EF BB BF，位于文件的前 3 个字节）也不需要带 BOM，但 Windows 历史遗留问题又会经常遇到有 BOM UTF-8 的数据文件。

其它常见的编码:

- ANSI: 不是真正的编码，而是 Windows 系统的默认编码的统称，对于简体中文系统就是 GB2312；对于繁体中文系统就是 Big5 等
- Latin1: 又称 ISO-8859-1，欧洲人发明的编码，也是 MySQL 的默认编码
- Unicode big endian: 用 UCS-2 格式存储 Unicode 时，根据两个字节谁在前谁在后，分为 Little endian（小端）和 Big endian（大端）
- UTF-16, UTF-32: Unicode 的另两种再表示，分别用 2 字节和 4 字节。

2. 中文乱码的解决办法

首先，查看并确认你的 windows 系统的默认编码方式:

```
Sys.getlocale("LC_CTYPE")       # 查看系统默认字符集类型
```

```
## [1] "Chinese (Simplified)_China.936"
```

代码 936 就表明是 ‘‘中国 - 简体中文（GB2312）’’。

> 注意：不建议修改系统的默认编码方式，因为可能会导致一些软件、文件乱码。

大多数中文乱码都是 GBK 与 UTF-8 不兼容导致的，常见的有两种情形。

R 文件中的中文乱码

在你的电脑不中文乱码的 R 脚本、Rmarkdown 等，拷贝到另一台电脑上时出现中文乱码。

解决办法： 前文在配置 Rstudio 时已讲到，设置 code – saving 的 Default text encoding 为兼容性更好的 UTF-8.

读写数据文件中文乱码

数据文件采用什么编码方式，就用什么编码方式打开或读取。采用了不兼容的另一种编码打开或读取，肯定出现中文乱码。

下面以最常见的中文编码 GBK、UTF-8、BOM UTF-8 来讲解。

R 自带函数读取 GBK 或 UTF-8

- 与所用操作系统默认编码相同的数据文件，即 GBK，R 自带的函数 read.csv()、read.table()、readLines() 都可以正常读取但不能直接读取 UTF-8
- 但在 read.csv() 和 read.table() 中设置参数 fileEncoding = "UTF-8"，可以读取 UTF-8，但无论如何不能读取 BOM UTF-8
- 在 readLines() 中设置参数 encoding = "UTF-8"，可以读取 UTF-8 和 BOM UTF-8

```
read.csv("datas/bp-gbk.csv")           # GBK, 直接读取
read.csv("datas/bp-utf8nobom.csv",     # UTF-8, 设置参数读取
         fileEncoding = "UTF-8")

readLines("datas/bp-gbk.csv")          # GBK, 直接读取
# UTF-8 和 BOM UTF-8, 设置参数读取
readLines("datas/bp-utf8nobom.csv", encoding = "UTF-8")
readLines("datas/bp-utf8bom.csv", encoding = "UTF-8")
```

readr 包读取 GBK 或 UTF-8

- readr 包中的 read_csv()、read_table2()、read_lines() 默认读取 UTF-8 和 BOM UTF-8;
- 但不能直接读取 GBK, 需要设置参数 locale = locale(encoding="GBK")

```
read_csv("datas/bp-utf8nobom.csv")          # UTF-8, 直接读取
read_csv("datas/bp-utf8bom.csv")            # BOM UTF-8, 直接读取
read_csv("datas/bp-gbk.csv",
         locale = locale(encoding="GBK"))   # GBK, 设置参数读取
```

写入 GBK 或 UTF-8 文件

- R 自带的 `write.csv()`, `writeLines()` 仍是跟随操作系统默认编码，即默认写出为 GBK 文件；设置参数 `fileEncoding = "UTF-8"` 可写为 UTF-8
- readr 包中的 `write_csv()`, `write_lines()` 默认写为 UTF-8, 但不能被 Excel 软件正确打开
- `readr::write_excel_csv()` 可以写为 BOM UTF-8, Excel 软件能正确打开

```
write.csv(df, "file-GBK.csv")              # 写出为 GBK 文件
write.csv(df, "file-UTF8.csv",
          fileEncoding = "UTF-8")          # 写出为 UTF-8 文件

write_csv(df, "file-UTF8.csv")             # 写出为 UTF-8 文件
write_excel_csv(df, "file-BOM-UTF8.csv")   # 写出为 BOM UTF-8 文件
```

不局限于上述编码，一个数据文件只要知道了其编码方式，就可以通过在读写时指定该编码而避免乱码。那么关键的问题就是：**怎么确定一个数据文件的编码？**

Notepad++是一款优秀开源的文本编辑器，用它打开数据文件，点**编码**，在下拉菜单黑点标记的编码方式即为该文件的编码，还可以对数据文件做编码转换：

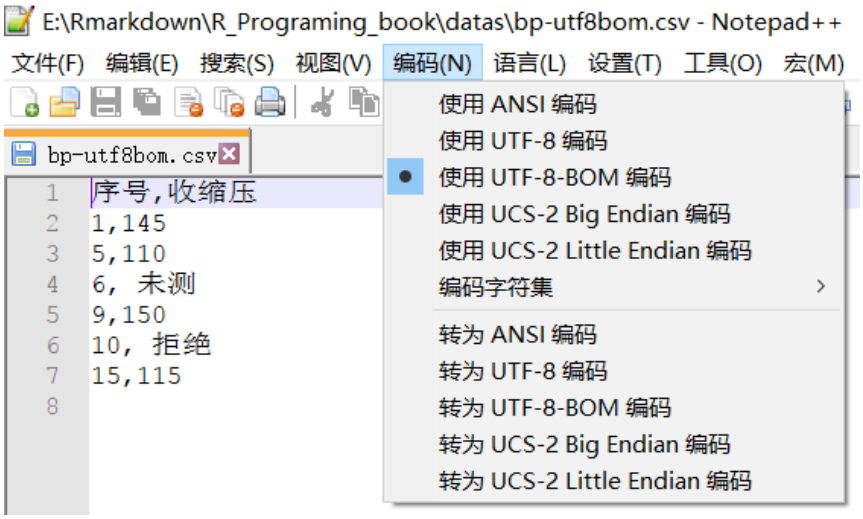


图 2.4: 用 Notepad++ 检测或转换文件编码

另外，readr 包和 rvest 包（爬虫）都提供了函数 `guess_encoding()`，可检测文本和网页的编码方式；python 有一个 chardet 库在检测文件编码方面更强大。

本节部分内容参阅 (李东风2020), (Desi Quintans 2019), 程序员必备：彻底弄懂常见的 7 种中文字符编码。

2.3 数据连接