R 语言编程：基于 tidyverse

第 08 讲　数据读写

张敬信

2022 年 2 月 11 日

哈尔滨商业大学

- 前文的 R 基本语法，涵盖了**向量化编程思维**（同时操作一堆数据）、**函数式编程思维**（自定义函数解决问题 + 泛函式循环迭代）。
- R 语言更多的是与数据打交道，下面正式进入 `tidyverse` 系列，将全面讲解“管道流、整洁流”操作数据的基本语法，包括：数据读写、数据连接、数据重塑，以及各种数据操作。
- 本章最核心的目的是训练数据思维，那么什么是数据思维?

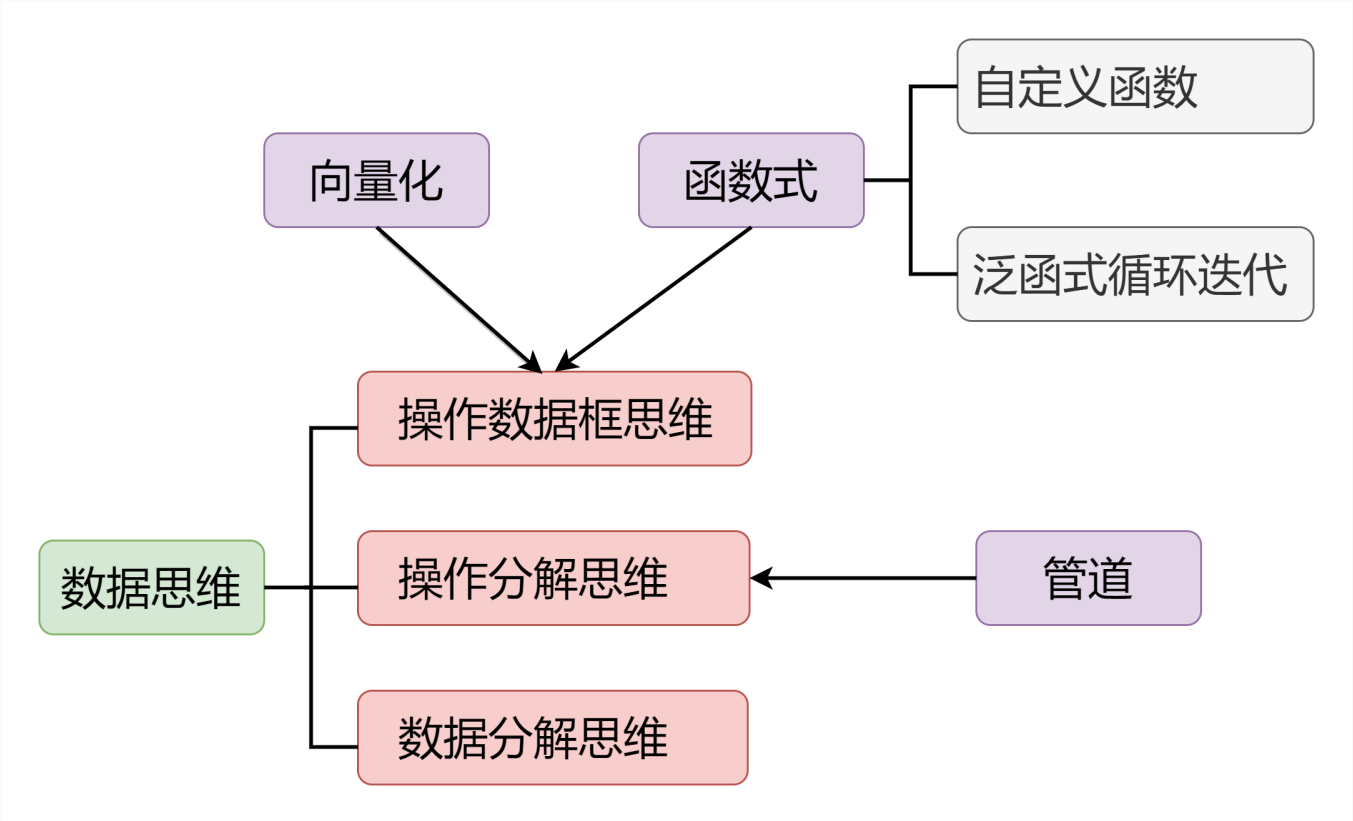

图 1: tidyverse 优雅编程思维

数据思维最关键的有三点：

⑴ **更进一步，将向量化编程思维和函数式编程思维，纳入到数据框或更高级的数据结构中来**

- 比如，向量化编程同时操作一个向量的数据，变成在数据框中操作一列的数据，或者同时操作数据框的多列，甚至分别操作数据框每个分组的多列；函数式编程变成为想做的操作自定义函数（或现成函数），再依次应用到数据框的多个列上，以修改列或做汇总。

⑵ **将复杂数据操作分解为若干基本数据操作的能力**

- 复杂数据操作都可以分解为若干简单的基本数据操作：数据连接、数据重塑（长宽变换/拆分合并列）、筛选行、排序行、选择列、修改列、分组汇总。一旦完成问题的梳理和分解，又熟悉每个基本数据操作，用“管道”流依次对数据做操作即可。

(3) 接受数据分解的操作思维

- 比如，想对数据框进行分组，分别对每组数据做操作，整体来想这是不容易想透的复杂事情，实际上只需做 `group_by()` 分组，然后把你要对一组数据做的操作实现；再比如，`across()` 同时操作多列，实际上只需把对一列要做的操作实现。这就是数据分解的操作思维，这些函数会帮你**分解** + **分别操作** + **合并结果**，你只需要关心**分别操作**的部分，它就是一件简单的事情。

- 很多从 C 语言等过来的编程新手，有着根深蒂固地逐个元素 `for` 循环操作、每个计算都得 “眼见为实” 的习惯，这都是训练数据思维的大忌，是最应该首先摒弃的恶习。

一. tidyverse 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。**
- tidyverse‘用” **现代**的”、“**优雅**的” 方式，以**管道式**、**泛函式**编程技术实现了数据科学的整个流程：**数据导入、数据清洗、数据操作、数据可视化、数据建模、可重现与交互报告。**
- `tidyverse` 操作数据的优雅，就体现在：
- 每一步要 “做什么”，就写 “做什么”，用管道依次做下去，得到最终结果
- 代码读起来，就像是在读文字叙述一样，顺畅自然，毫无滞涩

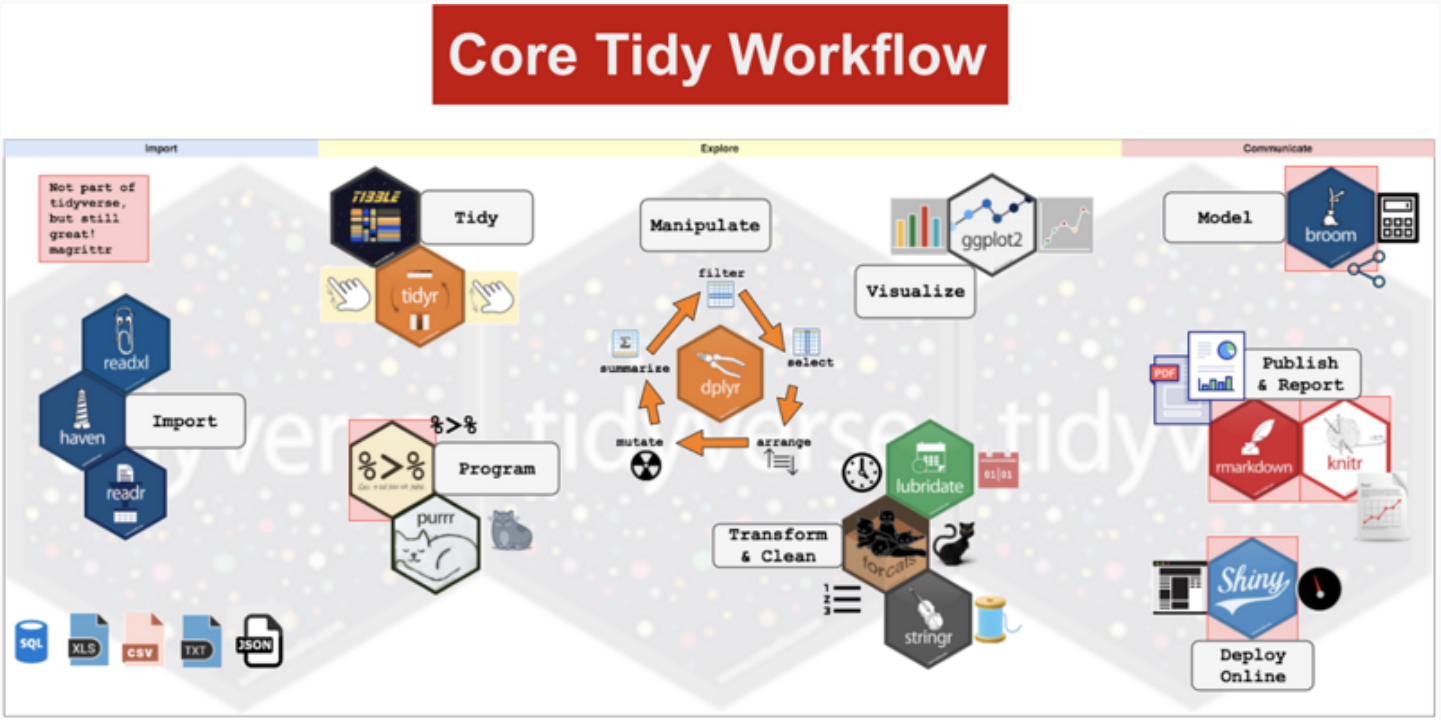

图 2: tidyverse 整洁工作流

```
library(tidyverse)
```

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 甚至十几 G 的数据，需要用 data.table.
- 但 data.table 的语法高度抽象、不容易上手。一种不错的方案是使用专门的转化包：底层用 data.table，上层用 tidyverse 语法包装（转化），如 dtplyr，tidyfst 等。

二. 管道操作

- magrittr 包引入了管道操作%>%[1]（现在 R 4.1 也开始支持管道 |>），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：

```
x %>% f() %>% g()
```

- 表示依次对数据进行若干操作：先对数据 x 进行 f 操作，接着对结果数据进行 g 操作
- 使用管道的好处：
  - 避免使用过多的中间变量
  - 程序可读性大大增强：对数据集依次进行一系列操作

[1] Windows 快捷键：Shift+Ctrl+M; Mac 快捷键：Cmd+Shift+M.

- 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：

```
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，也支持 base R 函数：

```
month.abb %>%                  # 内置月份名缩写字符向量
  sample(6) %>%
  tolower() %>%
  str_c(collapse = "|")
#> [1] "nov|oct|may|jun|dec|feb"
```

- 对比非管道操作，读起来与操作的过程是相反的：

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

常用的管道操作

(1) 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”[2]。

[2]正是这种管道操作，使得 tidyverse 能够**优雅**地操作数据. 因此，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.

(2) 数据可以在下一个函数中使用多次

- 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

```
# 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
```

- 分组批量建模：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

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

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

1. 数据读写的包与函数汇总

(1) readr 包[3]

- 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
  - 读入数据到数据框：read_csv() 和 read_tsv()
  - 读入欧式格式数据[4]：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

[3] readr 2.0 版本发布，read_csv() 采用 vroom 引擎读取性能大大提升，同时支持批量读取文件.

[4] UTF-8 本身不带 BOM，由于 Windows 历史原因，也有带 BOM 的 UTF-8.

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

(4) jsonlite 包

- 读写 JSON 数据，与 R 数据结构相互转换：
  - 读：read_json()，fromJSON()
  - 写：write_json()，toJSON()

(5) 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，常用于文本挖掘或数据收集；
- readtext 包还支持读取 csv, tab, json, xml, html, pdf, doc, docx, rtf, xls, xlsx 等。

```
library(readtext)
document = readtext("datas/十年一觉.txt") # 返回数据框
document  # `doc_id`列为文档标识
#> readtext object consisting of 1 document and 0 docvars.
#> # Description: df [1 x 2]
#>   doc_id       text
#>   <chr>        <chr>
#> 1 十年一觉.txt "\"    “这位公子爷\"..."
          # `text`列为读取的全部文本内容（1 个字符串）
```

2. 数据读写实例

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

- col_select: 支持 dplyr 选择列语法选择要读取的列;
- col_types: 设置列类型，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），可为每列单独设置，例如设置 3 列的列类型（缩写）：coltypes="cnd"，默认 NULL（全部猜测）
- locale: 设置区域语言环境（时区, 编码方式, 小数标记、日期格式），如从默认"UTF-8" 编码改为"GBK" 编码：locale = locale(encoding = "GBK")

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

- `col_types`: 设置列类型，可选列类型：“skip”（跳过该列），“guess”（猜测该列），“logical”, “numeric”, “date”, “text”, “list”，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）

(1) 读取 csv 文件

```
df = read_csv("datas/read_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
#> # ... with 1 more row
```

(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，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

注： purrr::map_dfr() 使得后两步可以同时做。

读取 read_datas 文件夹下所有 xlsx 文件

- 首先，获取要读入的全部 Excel 文件的完整路径，可以任意嵌套，只需设置 recurse=TRUE:

```
files = fs::dir_ls("datas/read_datas",
                   recurse = TRUE, glob = "*.xlsx")
files
#> datas/read_datas/六 1 班学生成绩.xlsx        datas/read_dat
#> datas/read_datas/六 4 班学生成绩.xlsx        datas/read_dat
#> datas/read_datas/新建文件夹/六 2 班学生成绩.xlsx
```

- 接着，用 map_dfr() 在该路径向量上做迭代，依次应用 read_xlsx() 到每个文件路径，再按行合并

- 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。

```r
library(readxl)
# 增加一列表明数据来自哪个文件
map_dfr(set_names(files), read_xlsx, .id = " 来源")
#> # A tibble: 20 x 7
#>   来源                                    班级  姓名   性别  
#>   <chr>                                   <chr> <chr>  <chr> <
#> 1 datas/read_datas/六 1 班学生成绩.xlsx 六 1 班 何娜   女 
#> 2 datas/read_datas/六 1 班学生成绩.xlsx 六 1 班 黄才菊 女 
#> 3 datas/read_datas/六 1 班学生成绩.xlsx 六 1 班 陈芳妹 女 
#> # ... with 17 more rows
```

- 函数 read_xlsx() 的其他控制读取的参数，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

```
path = "datas/学生成绩.xlsx"          # Excel 文件路径
# excel_sheets()`提取所有`sheet`名字
map_dfr(set_names(excel_sheets(path)),
              ~ read_xlsx(path, sheet = .x), .id = "sheet")
#> # A tibble: 20 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
#> # ... with 17 more rows
```

- readr 2.0 提供了非常简单方法实现批量读取 + 合并 csv 文件（列名/列类型相同）：

```
files = fs::dir_ls("datas/read_datas",
                   recurse = TRUE, glob = "*.csv")
read_csv(files)
```

3. **写出到一个 Excel 文件**

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

4. 批量写出到 Excel 文件

- 先将多个数据框放在一个列表中，再依次将它们写入文件，需要准备好文件名
- 依次做事情又不需要返回值，用 walk2() 作用在该数据框列表和文件名向量上即可

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

- 若要多个数据框分别写入一个 Excel 文件的多个 sheet，先将多个数据框创建为命名列表（名字将作为 sheet 名），再用 write_xlsx() 写出：

```
dfs = dfs %>%
  set_names(levels(iris$Species))
write_xlsx(dfs, "datas/iris.xlsx")
```

5. **保存与载入 rds 数据**

- 自带的 save() 和 load() 保存和加载.rda；.rds 包含更多元信息，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
```

四. 关于中文编码

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。

- 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。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**。

- 中国的第二件事情：汉字符号（中文）编码，因为历史原因产生了多种中文编码：

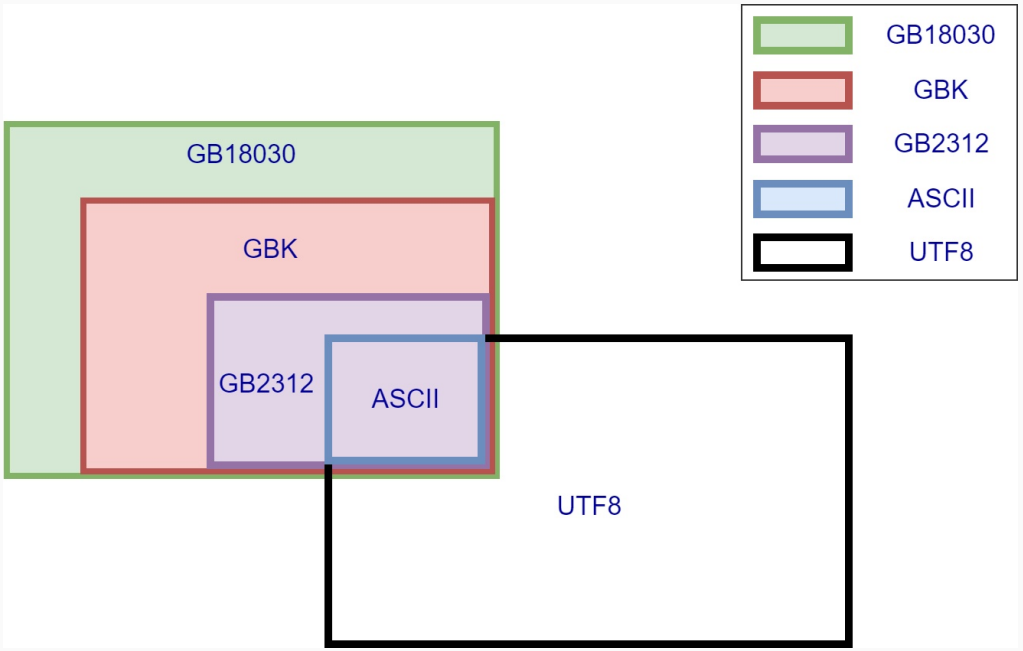

图 3: 几种中文编码及兼容性

常见编码

- 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[5]，是 Unicode 的再表示，支持各个国家的文字符号，兼容性非常好
- 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
- UTF-16, UTF-32: 是 Unicode 的另两种再表示，分别用 2 字节和 4 字节

[5] UTF-8 本身不带 BOM，由于 Windows 历史原因，也有带 BOM 的 UTF-8.

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。

(1) 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：配置 Rstudio 时，设置 code-saving 的 Default text encoding 为兼容性更好的 UTF-8.

(2) 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。

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 read.csv()、read.table()、readLines() 都可以正常读取但不能直接读取 UTF-8
- 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

```r
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",          # GBK, 设置参数读取
         locale = locale(encoding="GBK"))
```

(3) 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

```r
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
```

- 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?**

- AkelPad 是一款优秀开源小巧的文本编辑器，用它打开数据文件，自动在窗口下方显示文件的编码：

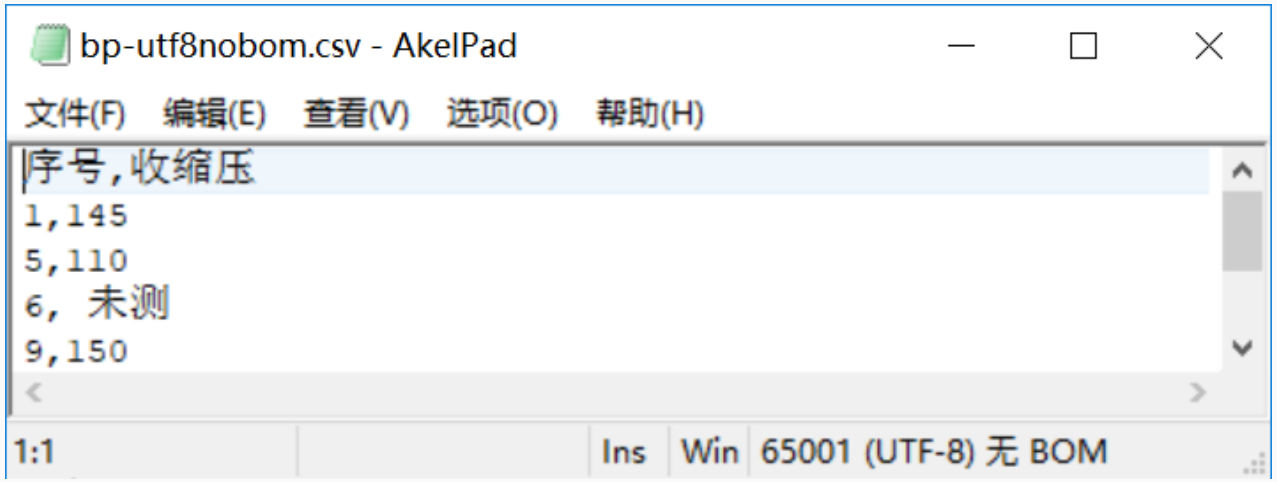

图 4: 用 AkelPad 检测文件编码

- 若要转换编码，只需要点文件另存为，在代码页下拉框选择想要的编码方式，保存即可。
- 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大。

本篇主要参阅 (张敬信, 2022), (Hadley Wickham, 2017), (李东风, 2020), (Desi Quintans, 2019), 以及 RStudio 博文等，模板感谢 (黄湘云, 2021), (谢益辉, 2021).

参考文献

Desi Quintans, J. P. (2019). *Working in the Tidyverse*. HIE Advanced R workshop.

Hadley Wickham, G. G. (2017). *R for Data Science*. O' Reilly, 1 edition. ISBN 978-1491910399.

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

李东风 (2020). *R* 语言教程.

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

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