Review of the course “R for Data Science” Part 01(Talk 01~04)

By Haoran Nie @ HUST Life ST

Partically translated by Rui Zhu @ HUST Life ST

双语版

This work is licensed under CC BY-NC-SA 4.0

Multi-omics data analysis and visualisation, #1

Talk 01

This section has ~~nothing~~ something to explain :)

Install R

Go go 华科镜像 (https://mirrors.hust.edu.cn/CRAN), R 支持主流的操作系统包括 Linux, Windows 和 MacOS，请根据操作系统下载对应的安装文件。

新版本的 Mac OS 还需要安装 XQuartz(http://xquartz.macosforge.org/landing/)。某些还需要用到 Xcode，可以从 AppStore 免费安装或者使用 `xcode --select install` 命令从控制台安装 “XCode 命令行工具”（体积更小更实用）。

目前大多 Linux 发行版都带有 R，因此可直接使用。从 CRAN 下载文件进行安装稍嫌复杂，要求用户对 Linux 系统有一定的了解，而且需要有管理员权限。建议初级用户在 Linux 高手指导下安装。点击”Download R forLinux” 后，发行版为 Redhat（红帽）或 Suse 的用户要先阅读网站上提供的 readme 或 readme.html 文件，然后其中的指示进行安装。这里就不再累述了。

Rstudio

RStudio 可以从 https://posit.co/downloads/ 下载，支持等主流的操作系统。

R language basics, part 1

Talk 02

基础数据类型

最基本的数据类型包括**数字**、**逻辑符号**和**字符串**，是其他数据类型的基本构建块。

数字

```
## 整数
287

## 小数
99.99

## 科学计数法
1e-3
1e2
```

逻辑符号

```
TRUE
T

FALSE
F

## 其本质是数字
1 + TRUE
2 * FALSE
```

字符串

```
'a sentence' ## 单括号
" 一个字符串" ## 双括号
'1.123'      ## 像是数字的字符串
'*%%*()!@###&@(9'   ## 乱码
```

简单数据类型

这包括向量和矩阵，它们都可以包含某种基本数据类型的多个值，如**矩阵 (matrix)** 由多个数字组成，**vector** 由多个字符串组成，等等。但是，**它们只能包含一种数据类型**。

```
c(100, 20, 30) ## Interger vector
c("String", "Array", "It's me".) ## String vector
c(TRUE, FALSE, TRUE, T, F) ## A logic vector
```

如上所示，数组通常用函数 `c ()`（又叫做 **concatenation function**）来定义。此外，包含连续整数**向量**可以使用：运算符来定义。

```
2:8
```

数据类型之间的转换

1. 自动转换

一个 `vector` 只能包含一种基本数据类型。因此，在定义数组时，如果输入值是混合的，某些基本数据类型会自动转换为其他类型，以确保数字类型的一致性；这在英语中被称为 **coerce**，具有强制转换的含义。此转换的优先级为：

- Logical types -> numeric types
- Logical Type -> String
- numeric type -> string
- 逻辑类型 -> 数字类型
- 逻辑类型 -> 字符串
- 数字类型 -> 字符串

vector 的数据类型转换规则

```
class( c(45, TRUE, 20, FALSE, -100) ); ## 逻辑和数字类型
str( c("string a", FALSE, "string b", TRUE) ); ## 逻辑和字符
str( c("a string", 1.2, "another string", 1e-3) ); ## 数字和字符
```

1. 手动切换

   除了自动转换之外，还可以手动转换向量中元素的类型:

   我们可以用 `class()` 或 `str()` 函数来判断 vector 包含的数据类型

   ```
   class(matrix( c(20, 30.1, 2, 45.8, 23, 14),
                 nrow = 2, byrow = T ));
   ```

```
[1] "matrix" "array"
```

```
str(matrix( c(20, 30.1, 2, 45.8, 23, 14),
            nrow = 2, byrow = T ));
```

```
 num [1:2, 1:3] 20 45.8 30.1 23 2 14
```

- Checking the type of a variable `class()`
- Checking of classes `is.type()`
- Conversion of classes `as.type()`

一些特殊值

- `NA` (Not Available) missing values
- `NaN` (Not a Number)is meaningless
- `-Inf` Negative Infinity
- `Inf` Positive Infinity
- `NULL` Null

Some functions to determine these special values:

- `is.na()`
- `is.finite()`
- `is.infinite()`

Vectors and Matrix

都是数组。A `vector` is a one-dimensional array and a matrix is a two-dimensional array. This means.

- There can be more dimensional arrays
- 高维数组，like `vector` and matrices, **只能包含一种基本数据类型**。
- Higher dimensional arrays can be defined by the `array()` function.

矩阵由函数 matrix() 定义，比如：

```
matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2, byrow = T );
```

矩阵的指定长度，即 nrow × ncol，可以不同于输入数据的长度。矩阵长度较小时，输入数据会被截短；而矩阵长度较大时，输入数据则会被重复使用。

```
## 生成一个 2x5 长度为 10 的矩阵，但输入数据的长度为 20
matrix( 1:20, nrow = 2, ncol = 5, byrow = T);

## 生成一个 2x3 长度为 6 的矩阵，但输入数据长度只有 3
matrix( 1:3, nrow = 2, ncol = 3, byrow = T );
```

下面两种情况，系统会报警告信息。
第一种情况，矩阵长度大于输入数据长度，且前者不是后者的整数倍。

```
matrix( 1:3, nrow = 2, ncol = 4, byrow = T );
```

第二种情况，矩阵长度小于输入数据的长度，且后者不是前者的整数倍。

```
matrix( letters[1:20], nrow = 3, ncol = 5, byrow = T );
```

加减乘除逻辑运算老一套

```
1 + 2 - 3 * 4 / 5; ## 加减乘除
1 + (2 - 3) * 4 / 5;  ##  改变优先级
2 ^ 6; ## 阶乘
5 %% 2; ## 取余
T | F; ## or
T & F; ## and
5 | 0; ## == 0 FALSE,! = 0 TRUE, 任何非零值视为逻辑真
```

通过 Console window 管理变量

```
ls();  ## 显示当前环境下所有变量
rm( x ); ## 删除一个变量
ls();

##rm(list=ls()); ## 删除当前环境下所有变量!!!
```

vector 算术 vectorisation：R 最重要的一个概念

```
x <- c(10,100,1000, 10000);
( y <- sqrt( x  ) * 4 + 10  ); ## 赋值的之后打印变量内容
```

核心在于数据**自动循环使用**

```
x / c(10,100);
[1]   1   1 100 100
x / c(10,100,1000); ## 会报错但仍会循环计算
Warning: 长的对象长度不是短的对象长度的整倍数 [1]    1    1    1 1000
```

matrix 算术

```
m <- matrix( c(20, 30.1, 2, 45.8, 23, 14), nrow = 2,
       dimnames = list( c("row_A", "row_B"), c("A", "B", "C") ) );
A         B         C
row_A      20.0       2.0        23
row_B      30.1      45.8        14

m / 10;
 A    B   C
row_A 2.00 0.20 2.3
row_B 3.01 4.58 1.4

m /  c(1,10,100);
A     B    C
row_A 20.00  0.02 2.30
row_B  3.01 45.80 0.14
```

更多 matrix 相关函数

```
dim(m);
nrow(m);
ncol(m);
range(m); ## Available when the content is numeric
summary(m); ## Can also be used in vector
```

Extra:

- Incorporation

- ```
  a <- 1:3;
  b <- LETTERS[1:3];

  ( ab <- c(a,b) );
  mode( ab ); ## 一个新的函数 ~ ...
  ```

- Take part `ab[1]`
- Replacement of individual values `ab[1] = c`
- Replacing multiple values `ab[c(2, 3)] = c("Weihua", "Chen")`
- Naming elements and replace values `names(ab) = as.character(ab)`
- Reverse `rev(1:10)`
- Sort&order

  ```
  lts = sample(LETTERS[1:20])
  sort(lts)
  ```

- 提取一行或多行

  ```
  # (There's already some data in workspace)

  $$ m
  > (List the content of matrix "m")

  $$ m[1, ]
  > (List the first row of matrix 'm')

  $$ m[1:2, ]
  > (List the first two rows of matrix 'm')
  ```

  You can also let the console to fetch multiple lines as the order you give.

  `m[c("row_B", "row_A")]`

  控制台将以 “row_B” 和 “row_A” 的顺序输出矩阵 “m” 的内容。

- 获取一列或多列

  As can be seen from the same principle, I only list codes here

  ```
  m[ , 1]
  m[ , c(1:2)]
  m[ , c("col_B", "col_A")]
  ```

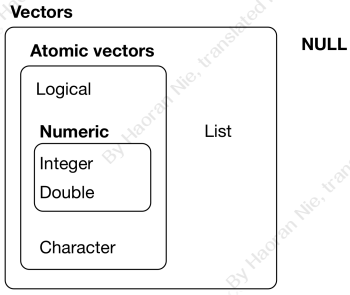
- Fetch parts `m[1:2, 2:3]`
  ```

- Replacement

```
m[1, ] = c(10)
m[, "C"] = c(230, 140)
m[1:2,] = matrix( 1:6, nrow=2)
m[1, c("C", "B")] = matrix(110:111, nrow = 1)
```

- 转置 t(m)

The hierarchy of R's vector types



You can use function `typeof()` to know the type of a vector.

Here are some examples of other `is.xxx()` function:

```
is.null( NULL )
is.numeric( NA )
is.numeric( Inf );
## 用于替代 typeof 的函数
is.list();
is.logical();
is.character();
is.vector();
# more ...
```

R language basics, part 2

Talk 03

`data.frame`

What is a `data.frame`?

- 二维表格
- 由不同列组成；每列是一个 **vector**，不同列的数据类型可以不同，但一列只包括一种数据类型（int, num, chr ...）
- 各列的长度相同

```
library(tidyverse);
library(kableExtra)
kbl(head(mpg),
    booktabs = T)
```

Here's the result:

manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact

Usage of `head()` and `tail()`

```
nrow(mpg); ## total number of rows
kbl( head(mpg,  n=3), booktabs = T); ## 显示前几行数据
kbl( tail(mpg,  n=3), booktabs = T); ## 显示最后 3 行数据
```

- `head()` is a function to display the first rows of some data (vectors etc.)
- `tail()` is a function to display the last rows of some data (vectors etc.)

Structure of `data.frame` & `tibble`

`str(mpg)`

This command shows the structure of the tibble `mpg`:

```
## tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
##  $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
##  $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
##  $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr [1:234] "f" "f" "f" "f" ...
##  $ cty         : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr [1:234] "p" "p" "p" "p" ...
##  $ class       : chr [1:234] "compact" "compact" "compact" "compact" ...
```

Make a new data.frame

You can use the function `data.frame()` to make a new `data.frame`

```
data2 =
        data.frame(
          data = sample(1:100, 10),
    group = sample(LETTERS[1:3], 10, replace = TRUE)
    data2 = 0.1
  )
```

How to add row(s)/col(s) to an existing data.frame

Create the "table header" first, then populate the `data.frame`

先创建" 表头"，再填充

```
df2 =
        data.frame(
    x = character(),
    y = integer(),
    z = double() ,
    stringsAsFactors = FALSE
  )

df2 =
        rbind(
    df2,
    data.frame(
      x = "a",
      y = 1L,
      z = 2.2
    )
  )

df2 =
        rbind(
    df2,
    data.frame(
      x = "b",
      y = 2,
      z = 4.4
    )
```

```
  )

df2
  x y   z
1 a 1 2.2
2 b 2 4.4

m <- cbind(1, 1:7) ; ## 产生两列数据 7 行数据 ..
( m <- cbind(m, 8:14) ); ## 增加一列 也有 7 行数据 ...
     [,1] [,2] [,3]
[1,]    1    1    8
[2,]    1    2    9
[3,]    1    3   10
[4,]    1    4   11
[5,]    1    5   12
[6,]    1    6   13
[7,]    1    7   14
```

ATTENTION

- Use `rbind()` function to add rows, use `cbind()` function to add columns.
- Define the new line using `data.frame()` function, the "header" needs to be the same as the merged table.

You can also use these functions to bind several data.frames.

`tibble`

`tibble` is kind of similar to `data.frame`.

注：Tibble class 是 data.frame 的升级版本；

Make new `tibble`

`tibble` 相关功能由 `tibble` 或 `tidiverse` 包提供

Almost all of the functions that you'll use in this book produce tibbles, as tibbles are one of the unifying features of the tidyverse. Most other R packages use regular data frames, so you might want to coerce a data frame to a tibble. You can do that with `as_tibble()`:

```
as_tibble(iris)
#> # A tibble: 150 × 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
```

```
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>
#> 1          5.1         3.5          1.4         0.2 setosa
#> 2          4.9         3            1.4         0.2 setosa
#> 3          4.7         3.2          1.3         0.2 setosa
#> 4          4.6         3.1          1.5         0.2 setosa
#> 5          5           3.6          1.4         0.2 setosa
#> 6          5.4         3.9          1.7         0.4 setosa
#> #   144 more rows
```

Another way to create a tibble is with `tribble()`, short for **tr**ansposed tibble. `tribble()` is customised for data entry in code: 列标题由公式定义（即以 "~" 开头），条目由逗号分隔。这使得以易于阅读的形式布置少量数据成为可能。

也可以用 tibble 函数创建

- 注意每列的数据类型
- 长度不足时，比如 **data2** 列，会循环使用
- `sample()` 函数的用法

```
tribble(
  ~x, ~y, ~z,
  #--|--|----
  "a", 2, 3.6,
  "b", 1, 8.5
)
#> # A tibble: 2 × 3
#>   x         y     z
#>   <chr> <dbl> <dbl>
#> 1 a         2   3.6
#> 2 b         1   8.5

## 用 tibble 函数创建，用法和 data.frame() 相似
dat <-
  tibble( data = sample( 1:100, 10 ),
        group = sample( LETTERS[1:3], 10, replace = TRUE),
        data2 = 0.1 )
# A tibble: 10 × 3
    data group data2
   <int> <chr> <dbl>
 1    86 A       0.1
 2    41 B       0.1
 3    72 B       0.1
```

```
 4    68 C       0.1
 5    10 A       0.1
 6    99 B       0.1
 7    66 B       0.1
 8    33 B       0.1
 9    81 A       0.1
10    42 C       0.1
```

- add_row()
- add_column()

```
## 新 tibble, with defined columns ... 创建表头
tb <- tibble( x = character(), y = integer(), z = double() );
dim(tb);

## 增加行 ...
tb <- add_row( tb, x = "a", y = 2, z = 3.6  );
tb <- add_row( tb, x = "b", y = 1, z = 8.5  );

## 显示
tb;

## 生成一个 tibble
df <- tibble(x = 1:3, y = 3:1);

# 在第二行之前插入
df <- add_row(df, x = 4, y = 0, .before = 2);

## 插入多行
df <- add_row(df, x = 4:5, y = 0:-1);

## 插入另一个 tibble （与另一个 tibble 合并）
df2 <- tibble( x = as.double(200:202), y = as.double(1000:1002) );
df3 <- add_row( df, df2 ); ## 可以运行 ...

tb3 <- tribble(
  ~x, ~y,  ~z,
  "a", 2,  3.6,
  "b", 1,  8.5
);
```

```
tb3 <- add_column( tb3, a = 98 ); ## recycle ...
tb3 <- add_column( tb3, b = LETTERS[1:2], c = c("CHEN", "WANG") );
```

tibble 元素替换

```
## 取得行
tb3[c(1,2), ];

## 取得列，按顺序取列
tb3[, c("z", "y")];

## 替换列
tb3[["z"]] <- c(4.6, 5.5);

## 替换行
tb3[ 1, ] <- tibble( x = "d", y = 20, z = 46, a = 10, b = "C", c = "LILI" );
```

Manipulate the tibble

> See "Manipulate the `data.frame`"

`tibble` to `data.frame`

- `as.data.frame()`
- `as_tibble()`

e.g.

```
library(tibble)
as.data.frame(head(as_tibble(iris)))
```

Differences between tibble and data.frame

Tibble 按顺序计算列

```
rm(x,y) # Delete possible x, y
tibble(x = 1:5, y = x^2); # You can do this with tibble
data.frame(x = 1:5, y = x ^ 2); # But data.frame doesn't work.
```

---

data.frame causes trouble when fetching subset operations

取子集成 vector

```
df1 =
        data.frame(x = 1:3, y = 3:1)
class(df1[, 1:2])

#> [1] "data.frame"

# Subset operation :takes a column and expects a data.frame ()
class(df1[, 1]) # The result is a vector ...

#> [1] "integer"

## Tibble doesn't.
df2 =
        tibble(x = 1:3, y = 3:1)
class(df2[, 1]) ## 永远都是 tibble

#> [1] "tbl_df" "tbl" "data.frame"
```

tibble allows controlled data type conversion

```
class(df2[[1]]); ## 取一列，转换为 vector
class(df2$x); ## 用 [[]] 或 $ 都可以哦

[1] "numeric"
[1] "numeric"
```

Recycling

```
data.frame(a = 1:6, b = LETTERS[1:2]); ##  data.frame 可以!!!
```

OUTPUT

```
#   a b
# 1 1 A
# 2 2 B
# 3 3 A
# 4 4 B
# 5 5 A
# 6 6 B
```

```r
tibble(a = 1:6, b = LETTERS[1:2]); ## 但 tibble 不行!!!
```

OUTPUT

```r
# Error:
# ! Tibble columns must have compatible sizes. ## * Size 6: Existing data.
# * Size 2: Column `b`.
#   Only values of size one are recycled.
```

ATTENTION!

`tibble` 的 recycling 仅限于长度为 1 或等长；而 `data.frame` 则为整除即可。

`data.frame` will do partial matching, while `tibble` will NEVER do it.

```r
df = data.frame(abc = 1)
df$ab; # Unwanted result ...

df2 = tibble(abc = 1)
df2$a; # Produce a warning and return NULL
```

OUTPUT

```r
#[1] 1
#Warning: Unknown or uninitialised column: `a`. NULL
```

Advanced tips for using `data.frame` and `tibble`

- `attach()`
- `detach()`
- `with()`
- `within()`

  Following is the introduction (Produced by ChatGPT)

这些函数——`attach ()`、`detach ()`、`with ()` 和 `within ()` ——在处理 R 中的数据帧或 tibble 时非常有用，有助于更流畅的工作流和代码可读性。下面是它们的功能分解：

`attach()` and `detach()`

- **Purpose**: 这些函数允许您将 data.frame 临时附加到搜索路径上，使其列可以通过名称直接访问。
- **Usage**:
  - `attach(df)` attaches the specified data frame `df`.
  - `detach(df)` detaches the specified data frame `df`.
- **Example**:

```r
data(mtcars) # Loading a sample dataset
attach(mtcars) # Attaching mtcars

# Now, columns can be accessed directly
summary(mpg)
mean(mpg)

detach(mtcars) # Detaching mtcars
```

- **Note**: 虽然使用 `attach ()` 很方便，但有时会导致混乱或意外的后果，例如屏蔽环境中的变量。由于潜在的副作用，通常建议避免使用 `attach ()`。

`with()`

- **Purpose**: `with()` 允许您在不使用 `$` 的情况下引用数据框架的列的环境中执行表达式。
- **Usage**:
  - `with(data, expr)` evaluates `expr` in the context of the specified data frame `data`.
- **Example**:

```r
data(mtcars) # Loading a sample dataset

with(mtcars, {
  mean(mpg)
  summary(cyl)
  })
```

- **Advantage**: 它有助于避免在处理其列时重复使用数据框架名称。

`within()`

- **Purpose**: Similar to `with()`, `within()` allows modification of a data frame by 计算其中的表达式。
- **Usage**:
  - `within(data, expr)` modifies `data` according to `expr` and returns the modified data frame.
- **Example**:

```r
data(mtcars) # Loading a sample dataset

modified_mtcars <- within(mtcars, {
  mpg_square <- mpg^2
  hp_doubled <- hp * 2
  })
head(modified_mtcars) # Checking the modified data frame
```

- **Advantage**: `within()` 非常有用，当你想创建或修改数据框架内的列时不必重复引用数据框架名称。

Remember, while these functions can streamline your code, it's crucial to use them judiciously to avoid unexpected behavior or cluttering your global environment.

Here's the console print output for the examples provided earlier:

```r
# Using attach() and detach()
data(mtcars) # Loading a sample dataset
attach(mtcars) # Attaching mtcars

# Now, columns can be accessed directly
summary(mpg)
# Output:
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#  10.40   15.43   19.20   20.09   22.80   33.90

mean(mpg)
# Output:
# [1] 20.09062

detach(mtcars) # Detaching mtcars

# Using with()
```

```r
data(mtcars) # Loading a sample dataset

with(mtcars, {
  mean(mpg)
  # Output:
  # [1] 20.09062

  summary(cyl)
  # Output:
  #    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  #   4.00    4.00    6.00   6.188   8.00    8.00
})

# Using within()
data(mtcars) # Loading a sample dataset

modified_mtcars <- within(mtcars, {
  mpg_square <- mpg^2
  hp_doubled <- hp * 2
})
head(modified_mtcars) # Checking the modified data frame
# Output:
#                   mpg cyl disp  hp drat    wt  qsec vs am gear carb mpg_square hp_doubled
# Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4     441.00       220
# Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4     441.00       220
# Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1     519.84       186
# Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1     457.96       220
# Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2     349.69       350
# Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1     327.61       210
```

File IO

Read from files

Using functions from the `readr` package

```r
# readr is part of tidyverse
library(tidyverse) # or alternatively
library(readr)
```

Some available functions:

- `read_csv()`: comma separated (CSV) files

- `read_tsv()`: tab separated files
- `read_delim()`: general delimited files
- `read_fwf()`: fixed width files
- `read_table()`: tabular files where columns are separated by white-space. read_log(): web log files

  Full documentation of the package is available through this link.

Usage

- Read with predifined column types

```
myiris2 =
        read_csv("../data/talk03/iris.csv",
          col_types =
                  cols(
             Sepal.Length = col_double(),
             Sepal.Width = col_double(),
             Petal.Length = col_double(),
             Petal.Width = col_double(),
             Species = col_character()
           )
         )
```

- To read from other formats, you can try the following packages:

  Similar to Python

  - `haven` - SPSS, Stata, and SAS files
  - `readxl` - excel files (.xls and .xlsx) DBI - databases
  - `jsonlite` - json
  - `xml2` - XML
  - `httr` - Web APIs
  - `rvest` - HTML (Web Scraping)

Write to files

Use the following functions to write object(s) to external files:

Default parameters are listed.

More related documents can be found in this link.

- Comma delimited file: 逗号分隔文件

```
write_csv(
  x,
  path,
  na = "NA",
  append = FALSE,
  col_names = !append
)
```

- File with arbitrary delimiter: 带有任意分隔符的文件

```
write_delim(
  x,
  path,
  delim = " ",
  na =" NA",
  append = FALSE,
  col_names = !append
)
```

- CSV for excel:

```
write_excel_csv(
  x,
  path,
  na = "NA",
  append = FALSE,
  col_names = !append
)
```

- String to file:

```
write_file(
  x,
  path,
  append = FALSE
)
```

- String vector to file, one element per line:

```
write_lines(
  x,
  path,
  na = "NA",
  append = FALSE
)
```

- Object to RDS file:

```
write_rds(
  x,
  path,
  compress =
          c(
      "none",
      "gz",
      "bz2",
      "xz"
    ),
  ...
)
```

- Tab delimited files:

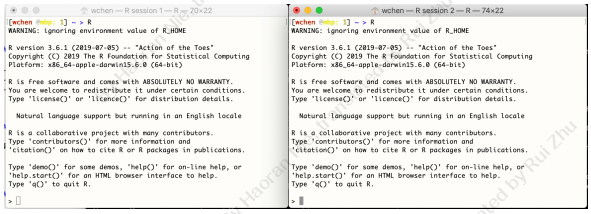
```
write_tsv(
  x,
  path,
  na = "NA",
  append = FALSE,
  col_names = !append
)
```

R language basics, part 3: factor

Talk 04

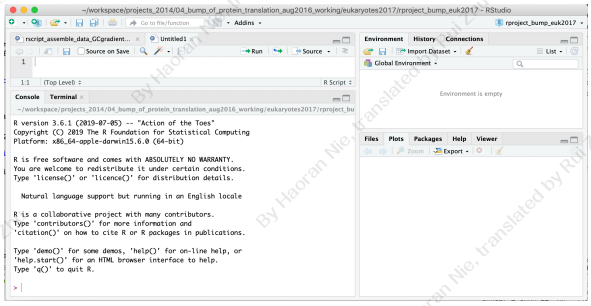
IO and working enviroment management

Each R session is a separate **work space** containing its own data, variables, and operation history.

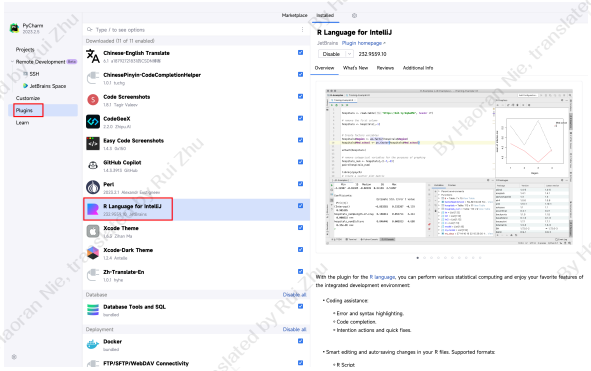


Each RStudio session is automatically associated with a R session

> Not only RStudio, PyCharm or VSCode also support R session.
>
> However, I'm keen on coding with PyCharm but not RStudio, for its wonderful Plug-in Environment, which can let me use plug-ins such as Code GeeX by Zhipu AI (a company founded by some student in KEG team inTsinghua University) or GitHub Copilot by GitHub to let the coding process more quickly, for the instruction from GPTs.
>
> tips: Rstudio 最新版也支持 Github Copliot, 可以在 options 内设置



If you want to coding with R using PyCharm or other JetBrains IDE (i.e. IntelliJ, CLion, etc.), remember to install the *R Language Plug-in*



For instruction how to get FREE Student Lisence of GitHub Pro, GitHub Copilot and JetBrains Products and their benefits, see their official website:

- GitHub Global Campus

  Make sure you don't use VPNs and use your phone to log in and apply (HUST Campus Network is recommended), give "Precise Location" permission to your browser. You may use your "[Student Number]@hust.edu.cn" mail to verify your identity as a student studying in HUST.

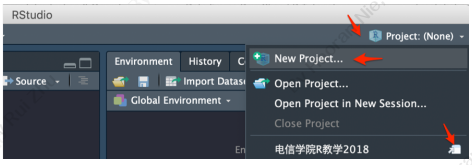
- JetBrains Products

  Because our email addresses ending with "@hust.edu.cn" are banned due to misuse, you should apply for an online verification report on CHSI (press the link to visit the website), instructions here.

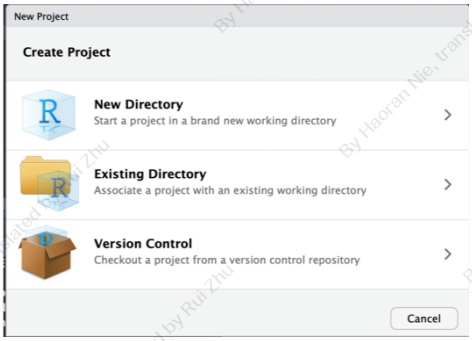
Start a new RStudio session by creating a new project

To start a new session in PyCharm, simply press the bottom corner and select a new session.

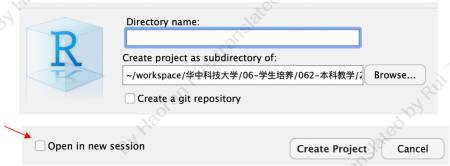
- Click the Project button in the upper right corner and select New Project in the pop-up menu …



- Select: New directory -> New Project in the popup window



- Enter a new directory name, choose its mother directory ...



Working Space

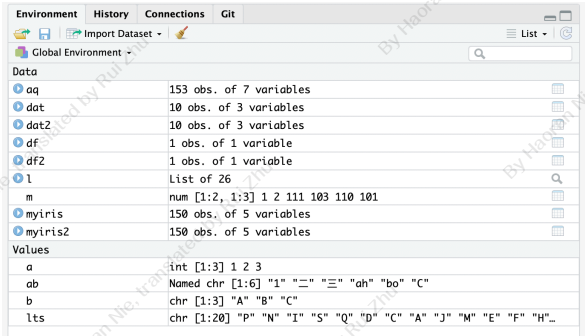
当前工作空间，包括所有已装入的数据、包和自制函数

可通过以下代码管理变量

```
ls();  ## 显示当前环境下所有变量
rm( x ); ## 删除一个变量
ls();

##rm(list=ls()); ## 删除当前环境下所有变量!!!
```

Variables in working space in RStudio

The ”Environment” window in the upper right corner of RStudio shows all the variables of the current workspace.



Save and restore work space

```
# Save all loaded variables into an external .RData file
save.image(file = "prj_r_for_bioinformatics_aug3_2019.RData")
# Restore (load) saved work space
load(file = "prj_r_for_bioinformatics_aug3_2019.RData")
```

Notes:

- Existing variables will be kept, however, those will the same names will be replaced by loaded variables
- Please consider using `rm(list=ls())` to remove all existing variables to have a clean start
- You may need to reload all the packages

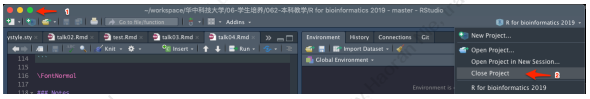
Save selected variables

Sometimes you need to transfer processed data to a collaborator ...

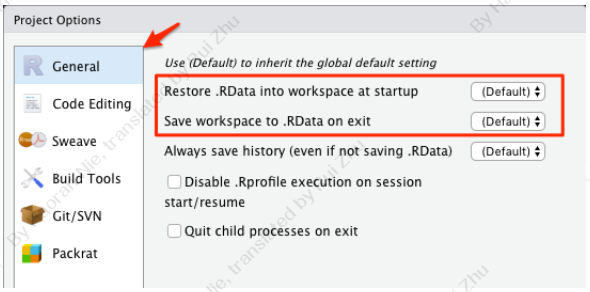
```
# Save selected variables to external
save(
  city,
  country,
  file="1.RData"
)
# You can specify directory name
load("1.RData")
```

Close and (re)open a project

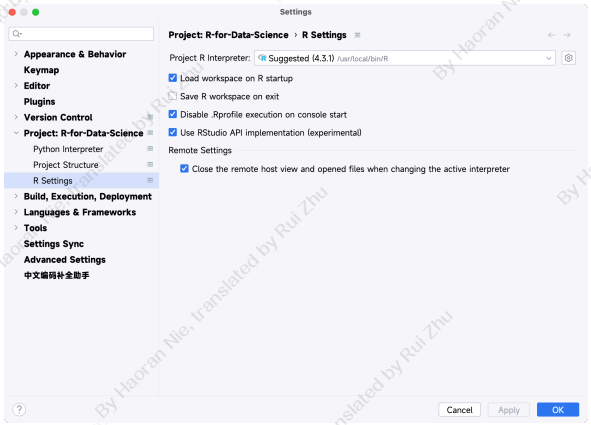
- To close a project



- In RStudio and similar IDEs, there are some preferences to choose



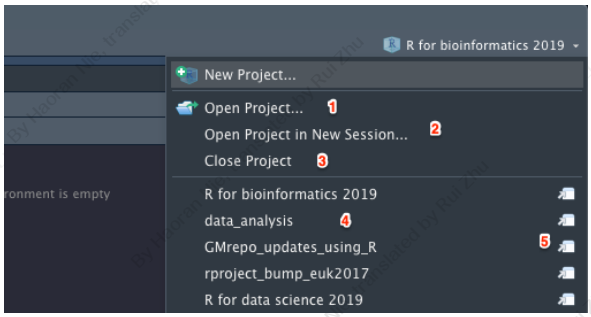
The UI in PyCharm



Notes:

- Save on exit
- Load on opening
- When the data is large, the loading time may be too long ...

Open a project



When in PyCharm, simply drag the working directory to its main window, remember to trust the project.

Factors

Factor 是一种用于字段的数据结构，它只接受预定义的、有限数量的值（分类数据）。它将限制输入数据的选取。

```
## create factor from scratch ...
x <- factor( c( "single", "married", "married", "single" ) );

## create factor as it is ...
```

```r
x <- c("single", "married", "married", "single");
x <- as.factor(x);

## please note the change in the displayed values ...
str(x);
#Factor w/ 2 levels "married","single": 2 1 1 2

# 限制输入数据的选择范围
x[ length(x) + 1 ] <- "widowed";
#Warning: 因子层次有错，产生了 NA
```

Play around with `levels()`

```r
## 利用 levels 解决
levels(x) <- c(levels(x), "widowed");
x[ length(x) + 1 ] <- "widowed";
str(x);

## other ways of assigning factors ...
y <-  as.factor( c( "single", "married", "married", "single" ) );
levels( y );
levels(y) <- c("single", "married", "widowed");
str(y);
## 这个代码现在就没有问题了
y[ length(y) + 1 ] <- "widowed";
```

注意用 `as.factor` 创建 factor 时，得到的 levels 按字母表排列；

但是，用 `levels( y )` 方式指定 levels 时，则按照指定的顺序；

`levels` 的顺序决定了排序的顺序

```r
##
y <-  as.factor( c( "single", "married", "married", "single" ) );
levels(y);
sort(y);
#[1] "married" "single"
#[1] married married single  single Levels: married single

##
y2 <- y;
levels(y2) <- c("single", "married", "widowed");
```

```r
sort(y2);
#[1] single  single  married married Levels: single married widowed
```

sort data in a meaningful way ...

```r
## Month
x1 <- c("Dec", "Apr", "Jan", "Mar");
sort(x1);

month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)

y1 <- factor(x1, levels = month_levels)
sort(y1);
#[1] "Apr" "Dec" "Jan" "Mar"
#[1] Jan Mar Apr Dec
#12 Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct ... Dec

## Sometimes you' d prefer that the order of the levels match the order of the first appearanc
f1 <- factor(x1, levels = unique(x1));
f1;
#[1] Dec Apr Jan Mar Levels: Dec Apr Jan Mar

library(forcats); ## just to make sure the codes will run smoothly ...
## you can also use fct_inorder in the forcats package ...
f2 <- x1 %>% factor() %>% fct_inorder()
f2
#[1] Dec Apr Jan Mar Levels: Dec Apr Jan Mar
```

Here are instructions of modifying factor levels

> Based on the textbook

The levels 既简洁又不一致。让我们调整使之更长，并使用一个平行的结构。Like most rename and recoding functions in the tidyverse, 新的值在左边，旧的值在右边：

`fct_recode`Change factor levels by hand

```r
load(gss_cat)

mutate(
```

```
  partyid = fct_recode(partyid,
    "Republican, strong"    = "Strong republican",
    "Republican, weak"      = "Not str republican",
    "Independent, near rep" = "Ind,near rep",
    "Independent, near dem" = "Ind,near dem",
    "Democrat, weak"        = "Not str democrat",
    "Democrat, strong"      = "Strong democrat"
  )
)

count(partyid)

#> # A tibble: 10 × 2
#>   partyid                  n
#>   <fct>                <int>
#> 1 No answer              154
#> 2 Don't know               1
#> 3 Other party            393
#> 4 Republican, strong    2314
#> 5 Republican, weak      3032
#> 6 Independent, near rep 1791
#> #   4 more rows
```

使用这种技术时要小心：如果你把完全不同的类别组合在一起，你最终会得到误导性的结果。

The order of the `levels` determines the sorting order.

Use factor to clean data

Usage of `fct_xxx()` functions.

Suppose I have a set of gender data that is written in a very irregular way:

```
## 假设我有一组性别数据，其写法非常不规整；
gender <- c("f", "m ", "male ","male", "female", "FEMALE", "Male", "f", "m");

gender_fct =
  as.factor(gender)

fct_count(gender_fct)
```

The output looks like this:

```
> fct_count(gender_fct)
# A tibble: 8 × 2
  f             n
  <fct>     <int>
1 "f"           2
2 "female"      1
3 "FEMALE"      1
4 "m"           1
5 "m "          1
6 "male"        1
7 "Male"        1
8 "male "       1
```

Now I request to replace with Female, Male.

```
## 要求：都改为 Female, Male
gender_fct =
  fct_collapse(
    gender,
    Female = c("f", "female", "FEMALE"),
    Male = c("m ", "m", "male ", "male", "Male")
  )

fct_count(gender_fct)
```

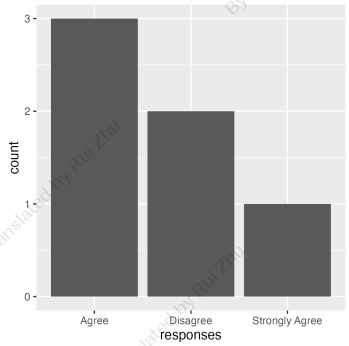
```
R-for-Data-Science
# A tibble: 2 × 2
  f             n
  <fct>     <int>
1 Female        4
2 Male          5
```

You can also use `fct_relabel()` to do the same thing

```
fct_relabel(
  gender,
  ~ ifelse(
    tolower(
      substring(., 1, 1)) == "f",
    "Female",
    "Male"
  )
)
```

factor 在做图中的应用（真正精髓）

```
## 一项 mock 调查结果数据
library(ggplot2)

responses =
  factor(
    c("Agree", "Agree", "Strongly Agree", "Disagree", "Disagree", "Agree")
  )

response_barplot =
  ggplot(
    data = data.frame(responses),
    aes(x = responses)
  ) +
  geom_bar()
```

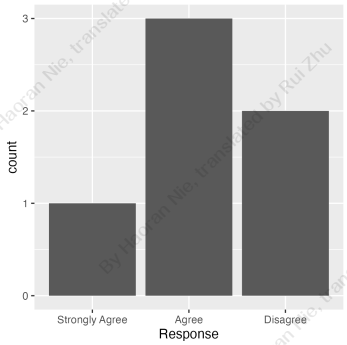


默认情况下，factor 按字母表排序：Agree -> Disagree -> Strong Agree 。ggplot2 也会按 factor 的排序作图

ggplot2 also plots `factor` in that order, so you can adjust the `factor` to adjust the drawing order.

```
res =
  data.frame(responses)
# Sort by level of agreement from strong -> weak
res$res =
  factor(
    res$res,
    levels =
       c("Strongly Agree", "Agree", "Disagree"),
    ordered = T
  )
```

```r
response_barplot2 =
  ggplot(
    data = res,
    aes(x = res)
  ) +
  geom_bar() +
  xlab("Response")+ ylab("Count")
```



You can also use the parameter `ordered` to let others know that your `factor` is ordered properly.

```r
responses =
  factor(
    c("Agree", "Agree", "Strongly Agree", "Disagree", "Disagree", "Agree"),
    ordered = TRUE
  )
is.ordered( responses )
```

```
> is.ordered(responses)
[1] TRUE
```

Using `factor` to vhange values

You can use `recode()` in `dplyr` package to change `value`

`dplyr` 是一种数据操作的语法，提供了一组一致的动词 that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

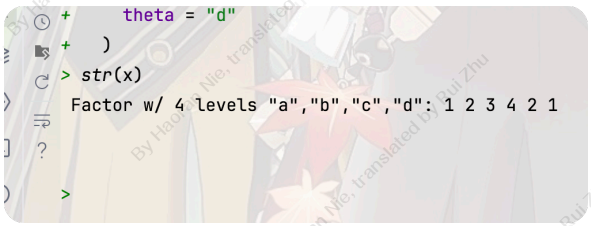
These all combine naturally with `group_by()` which allows you to perform any operation "by group". You can learn more about them in `vignette("dplyr")`. As well as these single-table verbs, dplyr also provides a variety of two-table verbs, which you can learn about in `vignette("two-table")`.

> Based on the introduction on the official website of `dplyr`.

Here's an example:

使用 `dplyr` 包的 `recode()` 函数改变 value

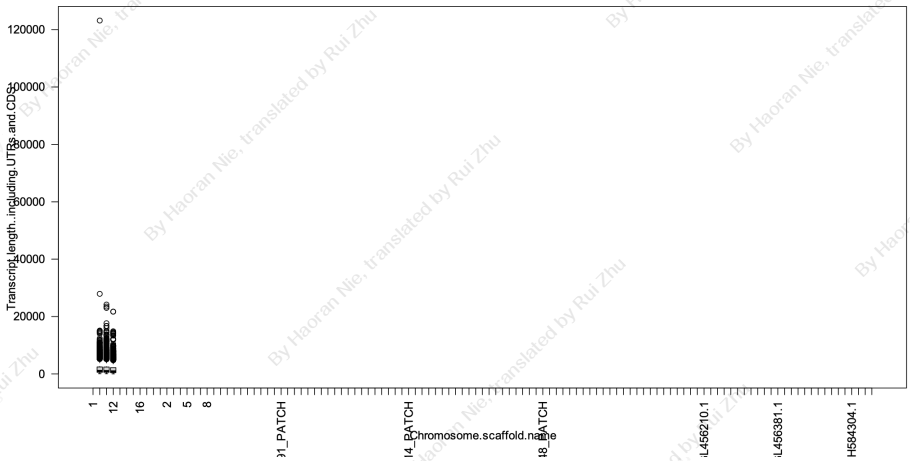
```
x =
  factor(
    c("alpha", "beta", "gamma", "theta", "beta", "alpha")
  )

x =
  recode(
    x,
    alpha = "a",
    beta = "b",
    gamma = "c",
    theta = "d"
  )
```



Delete useless `levels`

```
mouse.genes =
  read.delim(
    file = "data/talk04/mouse_genes_biomart_sep2018.txt",
    sep = "\t",
    header = T,
    stringsAsFactors = T
  )
```

If you draw a plot without deleting the useless `levels`, you will get this result:



`subset()` 无法去除不用的 factors ...

```
mouse.chr_10_12 <- subset( mouse.genes,  Chromosome.scaffold.name %in% c( "10", "11", "12" ) )
## plot length distribution --

boxplot( Transcript.length..including.UTRs.and.CDS. ~ Chromosome.scaffold.name,
         data = mouse.chr_10_12, las = 2 );
```

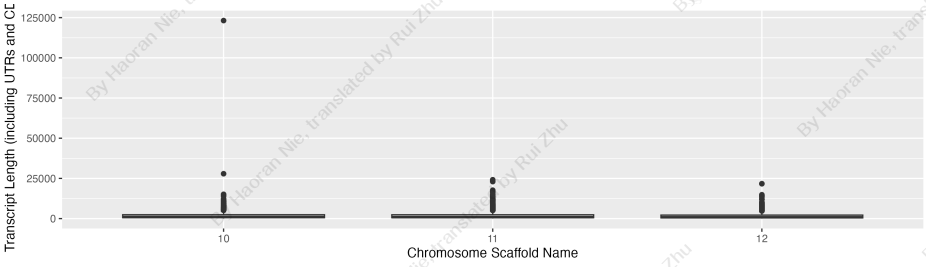
But when you delete the useless `level` using these commands:

```
mouse.chr_10_12$$Chromosome.scaffold.name =
  droplevels(mouse.chr_10_12$$Chromosome.scaffold.name)
levels( mouse.chr_10_12$$Chromosome.scaffold.name )
```

You will see that:

```
> droplevels( mouse.chr_10_12$Chromosome.scaffold.name )
> levels(mouse.chr_10_12$Chromosome.scaffold.name)
[1] "10" "11" "12"
```

Then, you'll get the plot like this:



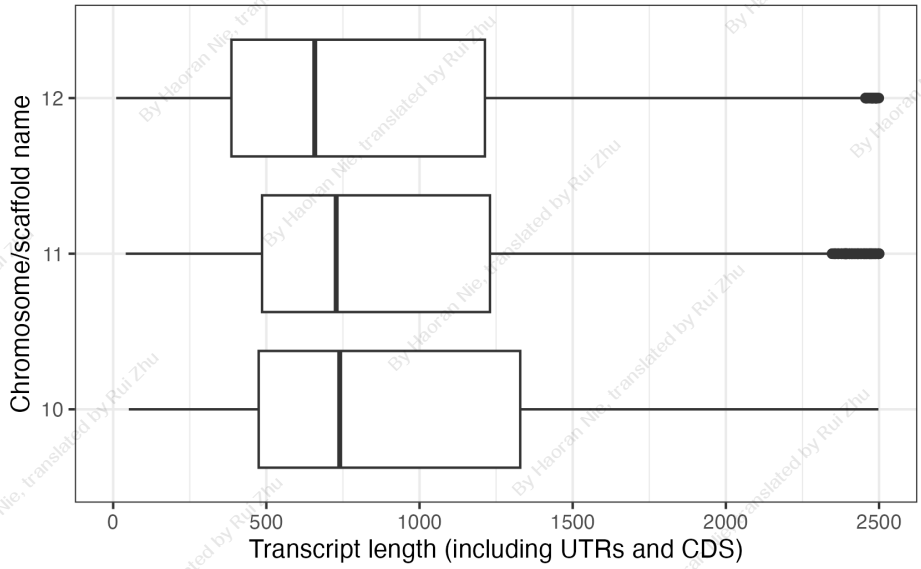
Source code:

```
mouse_gene_plot02 =
  ggplot(
    mouse.chr_10_12,
    aes(
      x = Chromosome.scaffold.name,
      y = Transcript.length..including.UTRs.and.CDS.
    )
  ) +
  geom_boxplot() +
  labs(
    x = "Chromosome Scaffold Name",
    y = "Transcript Length (including UTRs and CDS)"
  )
```

You can also use `tibble` to solve these problems: 完全不用担心 factor 的问题...

```
mouse.tibble =
  read_delim(
    file = "data/talk04/mouse_genes_biomart_sep2018.txt",
    delim = "\t",
    quote = "",
    show_col_types = FALSE
  )

mouse.tibble.chr10_12 =
  mouse.tibble %>% filter(
    `Chromosome/scaffold name` %in% c("10", "11", "12"))


mouse_gene_plot03 =
    ggplot(
        mouse.tibble.chr10_12,
        aes(
        x = Chromosome.scaffold.name,
        y = Transcript.length..including.UTRs.and.CDS.
        )
    ) +
    geom_boxplot() +
    labs(
        x = "Chromosome",
```

```
    y = "Transcript length (bp)"
  ) +
  coord_flip() +
  ylim(0, 2500) +
  theme_bw()
```



Advance usage

- Use `reorder()` function to reorder the level.
- `reorder()` 函数在 R 语言中通常用于改变因子（factor）水平的顺序。这对于数据可视化非常有用，特别是当你使用 ggplot2 包绘图时。在 ggplot2 中，`reorder()` 可以帮助你重新安排条形图、箱形图等的顺序。

  函数的基本用法是 `reorder(x, ...)`，其中 `x` 是你想要重新排序的因子，而 `...` 是额外的参数和方法，用于确定新的排序。最常见的用法是根据另一个变量的某种统计度量（如平均值、中位数等）来排序。

```
x = reorder(
  `Chromosome/scaffold name`,
  `Transcript length (including UTRs and CDS)`,
  median
)
```

- Use `forcats::fct_reorder()` to reorder factors

```
x = fct_reorder(
  `Chromosome/scaffold name`,
  `Transcript length (including UTRs and CDS)`,
  median
)
```

一些勘误

vector 和 factor 有什么区别?

1. **向量 (Vector)**:
   - **概念**: 向量是 R 中最基本的数据类型之一，用于存储相同类型的数据元素（如数值、字符或逻辑值）的序列。
   - **类型**: 向量可以是数值型（numeric），字符型（character），逻辑型（logical）等。
   - **用途**: 向量用于存储和操作数据集中的实际值。例如，一个存储温度读数或城市名称的序列。
2. **因子 (Factor)**:
   - **概念**: 因子是用于表示分类数据的数据类型。它类似于枚举类型，用于表示有限的、通常是预定义的值集合。
   - **水平**: 因子的值被称为水平（levels），这些水平代表了分类变量的可能值。
   - **用途**: 因子主要用于统计建模和图形表示中，用于处理分类数据。例如，在绘制条形图时，条形的类别就是由因子类型的变量确定的。

区别:

- **数据类型**: 向量直接存储值，而因子存储的是分类水平。
- **用途**: 向量用于表示一系列的值，而因子用于表示分类或分组。
- **统计分析**: 在进行统计分析时，因子用于指定数据的分类属性。例如，在回归分析中，因子可以用于指定自变量的分类。

data.frame 与 tibble 的区别

1. **data.frame**:
   - **历史**: `data.frame` 是 R 语言中最传统的数据集结构，自 R 诞生之初就存在。
   - **特性**: 它是一个表格型数据结构，每列可以包含不同类型的数据（数值型、字符型等），但每列数据类型必须一致。
   - **行为**: 在某些操作中，`data.frame` 可能会自动将字符串转换为因子（这取决于 `stringsAsFactors` 的默认设置）。
   - **使用**: `data.frame` 在所有标准的 R 函数和包中广泛支持。
2. **tibble**:
   - **历史**: `tibble` 是相对较新的数据集结构，由 `tidyverse` 生态系统引入，旨在解决 `data.frame` 的一些局限性。

- **特性**：tibble 保持了 data.frame 的所有基本特性，但添加了一些用户友好的改进，如更好的默认打印方法（显示更多信息，但更易于阅读）。
- **行为**：tibble 不会自动将字符串转换为因子，保留了数据的原始类型。
- **兼容性**：虽然 tibble 设计用于 tidyverse 生态系统，但它也兼容大多数接受 data.frame 的 R 函数。

关键区别：

- **因子转换**：tibble 默认不会将字符串自动转换为因子，而 data.frame 可能会这样做。
- **打印和查看数据**：tibble 提供了更友好的数据打印和查看方式，更适合于大型数据集。
- **设计理念**：tibble 是为了更好地适应现代数据分析需求而设计的，尤其是在 tidyverse 生态系统中。

Review of the course “R for Data Science” Part 02(Talk 05~08)

By Haoran Nie @ HUST Life ST

Partically translated by Rui Zhu @ HUST Life ST

双语版

This work is licensed under CC BY-NC-SA 4.0

R for bioinformatics, data wrangler, part 1

> Talk 05

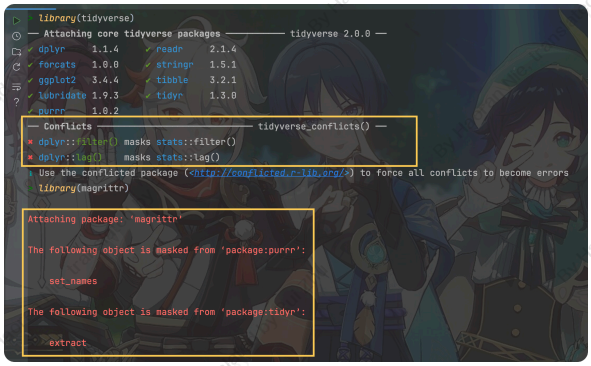
Pipe in R

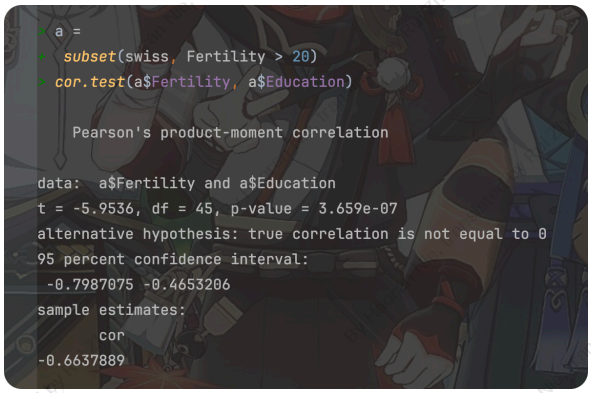
What is pipe in R?

- pipe 就是 `%>%`.
- It comes from the `magrittr` package by **Stefan Milton Bache**.
- Packages in the `tidyverse` load `%\>%` for you automatically, so you don't usually load `magrittr` explicitly.
- 实质是中间值的传递

Example

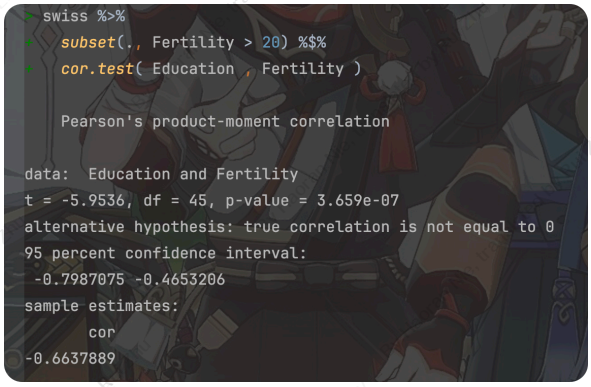
```
library(tidyverse)
library(magrittr)
a =
 subset(swiss, Fertility > 20)
cor.test(a$Fertility, a$Education)
```





The code above can be replaced by:

```
swiss %>%
  subset(., Fertility > 20) %$%
  cor.test(Education, Fertility)
```



所有函数都支持 pipe, 通常需要用 . 指代传递来的数据，并以参数的形式赋予下游函数

- **%>%**：最常见的管道操作符，用于将左侧表达式的结果作为右侧表达式的第一个参数。在这种情况下，右侧表达式的结果会成为整个管道表达式的结果。
- **%T>%** ：将左侧表达式的结果传递给右侧表达式。然而，与 **%>%** 不同的是，整个管道表达式的结果是左侧表达式的结果，而不是右侧表达式的结果。

- **%$%**：允许你在管道的右侧直接访问左侧对象的内部元素，而无需重复指定左侧对象的名称。特别适合用于那些需要从同一个数据对象中提取多个元素进行操作的情况，这在处理复杂的表达式时特别有用，可以使代码更加简洁和清晰。
- **%<>%**：结合了 %>% 和赋值操作的功能，允许你在对一个对象进行操作的同时更新这个对象本身。这意味着你可以在管道中对一个对象进行一系列的操作，并且这些操作的结果会直接反映到原始对象上，而不需要进行额外的赋值步骤。特别适用于数据处理和清理的场景，其中你需要对一个数据对象进行一系列的操作，并希望操作的结果直接更新到这个对象上。这样可以使得代码更加整洁，并减少潜在的错误，因为你不需要记住为每个中间步骤创建一个新的变量。

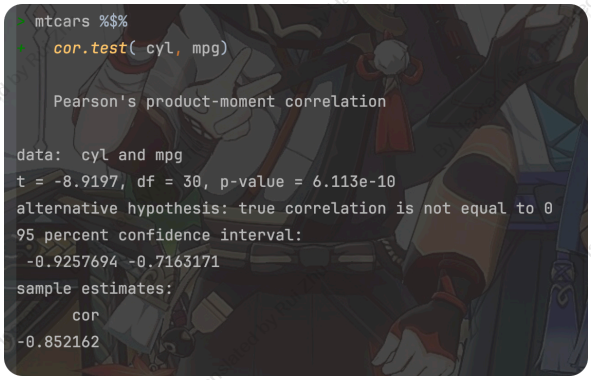
egs:

- %T>%: 返回上游值 (left-side values), 操作符在那些需要执行某些操作但不改变原始数据的场景中非常有用。例如，你可能想要打印或绘制原始数据的某些特性，同时保持数据本身不变以便后续操作。

```
res1 <-
  rnorm(100) %>%
    matrix(ncol = 2) %>%
    plot()# 此步骤将整个流程的结果赋值给 res1。但 plot() 函数不会返回数据，所以 res1 将不包含任何数

res2 <-
  rnorm(100) %>%
    matrix(ncol = 2) %T>%
    plot();# 由于使用 %T>%，res2 将包含步骤 2 中生成的矩阵，而不是 plot() 的输出
```

- %$>%: Attach ...

```
attach( mtcars ); ## note the warning message ...
cor.test( cyl, mpg ); ## 汽缸数与燃油效率

detach( mtcars );
with( mtcars, cor.test( cyl, mpg ) );

mtcars %$%
cor.test( cyl, mpg );
```



- %<>%

```
## 双向 pipe
mtcars %<>% transform(cyl = cyl * 2);
```

ATTENTION

1. pipe 的使用可以使思路更清晰

- 因此，尽量使用 %>% （方向明确），而不使用其它方向不明确的 pipe

Data Wrangler - dplyr

What is dplyr?

- The next iteration of plyr,
- Focusing on only data frames (also tibble),
- Row-based manipulation,
- dplyr is faster and has a more consistent API.

dplyr provides a consistent set of verbs that help you **solve the most common data manipulation challenges**:

1. select()

- **功能**：select() 用于从数据框中选择一列或多列。
- **常见用法**：select(data, column1, column2, ...)
- 参数

  - `data`：数据框对象。
  - `column1, column2, ...`：要选择的列的名称。

1. `filter()`

- **功能**：`filter()` 用于根据条件筛选数据框中的行。
- **常见用法**：`filter(data, condition)`
- 参数
  - `data`：数据框对象。
  - `condition`：筛选条件，可以是逻辑表达式。

1. `mutate()`

- **功能**：`mutate()` 用于在数据框中添加新列或修改现有列。
- **常见用法**：`mutate(data, new_column = expression)`
- 参数
  - `data`：数据框对象。
  - `new_column = expression`：创建或修改列的表达式。

1. `summarise()`

- **功能**：`summarise()` 用于对数据框中的数据进行汇总或聚合操作。
- **常见用法**：`summarise(data, summary = function(column))`
- 参数
  - `data`：数据框对象。
  - `summary = function(column)`：汇总或聚合操作，如求和、平均等。

1. `arrange()`

- **功能**：`arrange()` 用于根据一列或多列对数据框中的行进行排序。
- **常见用法**：`arrange(data, column)`
- 参数
  - `data`：数据框对象。
  - `column`：用于排序的列。可以添加多列进行多级排序。

e.g.

查看 mouse.tibble 的内容

```
# Read the file
library(tidyverse)
mouse.tibble =
  read_delim(
    file = "data/mouse_genes_biomart_sep2018.txt",
    delim = "\t",
    quote = "",
    show_col_types = FALSE
  )

# View mouse.tibble content
ttype.stats =
  mouse.tibble %>%
    count(`Transcript type`) %>%
    arrange(-n)

# View mouse.tibble content, cont.
chr.stats =
  mouse.tibble %>%
    count(`Chromosome/scaffold name`) %>%
    arrange(-n)
```

分析任务

1. 将染色体限制在常染色体和 XY 上（去掉未组装的小片段）；处理行
2. 将基因类型限制在 protein_coding, miRNA 和 lincRNA 这三种；处理行
3. 统计每条染色体上不同类型基因（protein_coding, miRNA, lincRNA）的数量
4. 按染色体（正）、基因数量（倒）进行排序

用 dplyr 实现

```
dat <- mouse.tibble %>%
  ## 1.

  filter( `Chromosome/scaffold name` %in% c( 1:19, "X", "Y" )   ) %>%

  ## 2.
```

```
  filter( `Transcript type` %in% c( "protein_coding", "miRNA", "lincRNA" ) ) %>%

  ## change column name ...
  select( CHR = `Chromosome/scaffold name`, TYPE = `Transcript type`,
          GENE_ID = `Gene stable ID`,
          GENE_LEN =  `Transcript length (including UTRs and CDS)`  ) %>%

  ## 3.
  group_by( CHR, TYPE ) %>%
  summarise( count = n_distinct( GENE_ID ), mean_len = mean( GENE_LEN ) ) %>%

  ## 4.
  arrange(  CHR  , desc( count ) );
```

检查运行结果

```
knitr::kable( head( dat, n = 15 ) );
```

CHR	TYPE	count	mean_len
1	protein_coding	1200	2699.59009
1	lincRNA	347	1206.76149
1	miRNA	128	97.97656
10	protein_coding	1020	2408.16454
10	lincRNA	398	1220.35543
10	miRNA	91	89.87912
11	protein_coding	1640	2431.87666
11	lincRNA	189	1134.49174
11	miRNA	137	87.48905
12	protein_coding	644	2523.94822
12	lincRNA	327	1277.14979
12	miRNA	146	86.24658
13	protein_coding	831	2380.41499
13	lincRNA	428	1251.04552
13	miRNA	97	105.52577

R for bioinformatics, data wrangler, part 2

Talk 06

`tidyr`

- `pivot_longer()` to take the place of `gather`
- `pivot_wider()` to take the place of `spread`

Data Wrangler - `tidyr`

You can get `tidyr` in the package set `tidyverse`, or simply install it the first time you want to use it via `install.packages("tidyr")`.

宽数据的特点

优点:

- 自然，易理解;

缺点:

- 不易处理;
- 稀疏时问题较大;

The usage of `tidyr`

- 宽和长数据的相互转换

```
# Eg 1
library(tidyverse)
grades2 =
        read_tsv(file = "data/grades2.txt")

grades3 =
        grades2 %>%
        pivot_longer(
    - name,# 所有除 name 列之外的列将被转换为长格式。
    names_to = "course",# 宽格式中的列名（如课程名称）将被转换并存储在名为 course 的新列中。
    values_to = "grade"# 原始数据框中的值（如成绩）将被转换并存储在名为 grade 的新列中。
  )
```

```r
# Eg 2
grades3_wide = grades3_long %>%
  pivot_wider(
    names_from = "course",# course 列的值将成为宽格式数据框的新列名。
    values_from = "grade"#grade 列的值将填充到相应的新列中。
  )
```

If you meet NA in the 1st example, you can do like this:

```r
grades3_1 =
        grades3[!is.na(grades3$grade), ]
grades3_2 =
        grades3[complete.cases(grades3), ]

# A better solution
grades3_long = grades2 %>%
  pivot_longer( - name,
                names_to = "course",
                values_to = "grade",
                 values_drop_na = TRUE# 删除任何包含 NA 的行。结果中只会包含完整的、没有缺失值的记
              )

# Pay attention to the variant named "values_drop_na"
```

More functions in tidyr: (See @ https://r4ds.hadley.nz/data-tidy.html)

tidyr::separate()

将包含多个信息的单一列分割成多个列，以便于进行更深入的数据分析和可视化。

- data: 数据框。
- col: 需要被分割的列。
- into: 一个字符串向量，包含新列的名称。
- sep: 分割符号。默认是非字母数字字符。
- remove: 是否移除原始列，默认为 TRUE。
- convert: 如果设置为 TRUE，尝试自动将分割后的字符串转换为适当的数据类型。

Usage:

```r
separate(
  data,
  col,
  into,
  sep = "[^[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
  extra = "warn",
  fill = "warn",
  ...
)

# Default parameters are listed.
```

tidyr::unite()

将多个列合并成一个单独的列。

- data: 数据框。
- new_col: 合并后的新列的名称。
- col1, col2, ...: 需要合并的列。
- sep: 合并时使用的分隔符，默认为下划线（_）。
- remove: 是否移除原始列，默认为 TRUE。

Usage:

```r
unite(
  data,
  new_col,
  col1, col2, ...,
  sep = "_",
  remove = TRUE,
  na.rm = FALSE
)

# Default parameters are listed.
```

R for bioinformatics, Strings and regular expression

Talk 07

`stringr`

1. basics

- length
- uppercase, lowercase
- unite, separate
- string comparisons, sub string

1. regular expression

Before you start...

```
library(stringr)
```

Also notice other famous packages used to manipulating string:

`stringi`(Following are based on the official R Documentation)

Description `stringi` is THE R package for fast, correct, consistent, and convenient string/text manipulation. It gives predictable results on every platform, in each locale, and under any native character encoding.

Usage of `writeLines()` (from official R Documentation)

Description Write text lines to a connection.

Usage

```
writeLines(text, con = stdout(), sep = "\n", useBytes = FALSE)
```

Arguments

`text`	A character vector
`con`	A connection object or a character string.
`sep`	character string. A string to be written to the connection after each line of text.
`useBytes`	logical. See 'Details'.

Details 如果 `con` 是一个字符串，函数调用 `file` 来获得一个文件连接，该文件连接在函数调用期间被打开。(tilde expansion of the file path is done by `file`.)

如果连接是打开的，则从其当前位置写入。如果未打开，则在 `wt` 模式下在调用期间打开，然后再次关闭。

正常情况下，`writeLines` 用于文本模式连接，默认分隔符转换为该平台的正常分隔符（Unix/Linux 上为 LF,Windows 上为 CRLF）。为了获得更多的控制，打开一个二进制连接，并在 `sep` 中指定要写入文件的精确值。为了更好的控制，在二进制连接上使用 `writeChar`。

`useBytes` is for expert use. Normally (when false) character strings with marked encodings are converted to the current encoding before being passed to the connection (which might do further re-encoding). `useBytes = TRUE` suppresses the re-encoding of marked strings so they are passed byte-by-byte to the connection: this can be useful when strings have already been re-encoded by e.g. `iconv`. (It is invoked automatically for strings with marked encoding `"bytes"`.)

Difference between double quote(“”) and single quote(’)

In R and its string manipulation package `stringr`, there is no difference between strings defined with double quotes (") and single quotes ('). Both are used to define strings and you can use either depending on your preference or the situation.

例如，如果字符串包含单引号，您应该用双引号将字符串括起来，反之亦然。Here's an example:

```r
# Using double quotes when the string contains a single quote
string1 = "It's a beautiful day"

# Using single quotes when the string contains a double quote
string2 = 'He said, "Hello, world!"'
```

In both cases, R will interpret the contents between the quotes as a string.

Some of the functions in the `stringi` package are similar in function to those that come with the system.

Here are some functions in the `stringi` package that share similar functionalities with base R's string functions, along with examples showcasing their differences:

1. **`stri_length()` vs. `nchar()`**:

- `stri_length()` in `stringi` calculates the number of code points in a string, accounting for Unicode characters.
- `nchar()` in base R counts the number of characters in a string, but it might not handle Unicode characters as accurately as `stri_length()`.

```
library(stringi)

# Using stri_length from stringi
string = "café"
stri_length(string)
# Output: 4

# Using nchar from base R
nchar(string)
# Output: 4
```

In this example, both `stri_length()` and `nchar()` return the same count for ASCII characters. However, when dealing with Unicode characters, `stri_length()` can accurately count them as individual code points, whereas `nchar()` might not handle them correctly.

2. **`stri_split_*()` vs. `strsplit()`**:
   - `stri_split_*()` functions in `stringi` split a string based on various criteria like fixed patterns, regular expressions, or character classes.
   - `strsplit()` in base R performs a similar operation but might differ in handling certain edge cases and Unicode characters.

```
# Using stri_split_* from stringi
string = "apple, orange, café"
stri_split_fixed(string, pattern = ", ")
# Output: list("apple", "orange", "café")

# Using strsplit from base R
strsplit(string, split = ", ")
# Output: list("apple", "orange", "caf", "é")
```

Here, `stri_split_fixed()` correctly splits the string, including the accented character "é," while `strsplit()` treats the accented "é" as two separate characters due to how it handles Unicode.

3. **`stri_detect()` vs. `grepl()`**:
   - `stri_detect()` in `stringi` checks if a pattern exists in a string and returns a logical value.

- `grepl()` in base R performs a similar task but might differ in its handling of Unicode characters and certain pattern matching options.

```
# Using stri_detect from stringi
string = "This is a café"
stri_detect(string, regex = "café")
# Output: TRUE

# Using grepl from base R
grepl("café", string)
# Output: FALSE
```

In this example, `stri_detect()` correctly detects the presence of the word "café," while `grepl()` returns a different result due to potential differences in Unicode handling or pattern matching options.

The examples highlight how `stringi` functions like `stri_length()`, `stri_split_*()`, and `stri_detect()` differ from their base R counterparts (`nchar()`, `strsplit()`, and `grepl()`) by providing more accurate handling of Unicode characters and often more versatile string manipulation options.

Some of the functions in the `stringr` package are similar in function to those that come with the system.

Here are examples comparing some functions from the `stringr` package with their counterparts from base R:

string length

1. **`str_length()` vs. `nchar()`:**

```
library(stringr)

# Using str_length from stringr
string = c("apple", NA, "banana", "")
str_length(string)
# Output: 5  NA  6  0

# Using nchar from base R
nchar(string)
# Output: 5  NA  6  0
```

`str_length()` 和 `nchar()` 都计算每个字符串元素中的字符数。然而，`str_length()` 通过返回 `NA` 来更一致地处理缺失值，而 `nchar()` 在某些情况下可能会以不同的方式处理 `NA`。

1. **str_sub() vs. substr():**

```r
# Using str_sub from stringr
string = c("hello", "world", "example")
str_sub(string, start = 2, end = 4)
# Output: "ell" "orl" "xam"

# Using substr from base R
substr(string, start = 2, stop = 4)
# Output: "ell" "orl" "xam"
```

str_sub() 和 substr() 都根据指定的开始和结束位置提取子字符串。然而，str_sub() 允许负索引从字符串的末尾开始计数，并且它更一致地处理缺失值。

1. **str_replace() vs. sub() or gsub():**

```r
# Using str_replace from stringr
string = c("apple pie", "banana bread", "cherry cake")
str_replace(string, pattern = "a", replacement = "X")
# Output: "Xpple pie"    "bXnana bread" "cherry cXke"

# Using sub from base R
sub(pattern = "a", replacement = "X", x = string)
# Output: "Xpple pie" "bXnana bread" "cherry cXke"
```

str_replace() 和 sub() 都用于替换字符串的部分内容。然而，str_replace() 有一个更直观的界面，与 sub() 相比，它能更优雅地处理缺失值。

1. **paste()vs.str_c()**

string combine

```r
## 系统自带
paste( "a", "b", "c", sep = "" );
#[1] "abc"

## stringr
str_c( "a", "b", "c" );
#[1] "abc"
```

string comparison

strcmp 函数:

- **参数**:
  - `str1`: 第一个字符串。
  - `str2`: 第二个字符串。
- **返回值**:
  - logical, i.e. `TRUE` if `s1` and `s2` have the same length as character vectors and all elements are equal as character strings, else `FALSE`.

`strcmpi` 函数:

- **参数**:
  - `str1`: 第一个字符串。
  - `str2`: 第二个字符串。
- **返回值**:
  - 类似于 `strcmp`，但是在不区分大小写的情况下进行比较。

```
## direct comparison ; 可用于排序 ...
"A" > "abc";
#[1] FALSE

##
library(pracma);
strcmp( "chen", "chenweihua" );
strcmpi( "chen", "CHEN" );

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

These examples demonstrate how `stringr` functions can be more consistent and user-friendly in handling various string operations compared to their base R counterparts.

(In the slide) Difference between `toupper()`, `tolower()` and `stri_reverse()`

The functions `toupper()` and `tolower()` in base R and `stri_reverse()` in the `stringi` package perform similar tasks, but there are some differences in their functionality and usage:

1. **`toupper()` and `tolower()` in Base R:**
   - `toupper()` 将字符串中的字符转换为大写。

- `tolower()` 将字符串中的字符转换为小写。

```r
# Using toupper and tolower from base R
string = "Hello World!"

toupper(string)
# Output: "HELLO WORLD!"

tolower(string)
# Output: "hello world!"
```

These functions are straightforward and work well for ASCII characters, converting them to uppercase or lowercase, respectively. However, they might not handle Unicode characters or locale-specific transformations.

2. **`stri_reverse()` in `stringi`:**

- `stri_reverse()` 颠倒字符串中字符的顺序，包括处理多字节字符和 Unicode 序列。

```r
library(stringi)

# Using stri_reverse from stringi
string = "café"

stri_reverse(string)
# Output: "éfac"
```

`stri_reverse()` reverses the characters in the string accurately, even when dealing with Unicode characters or multibyte sequences. It ensures correct reversal of characters irrespective of their encoding.

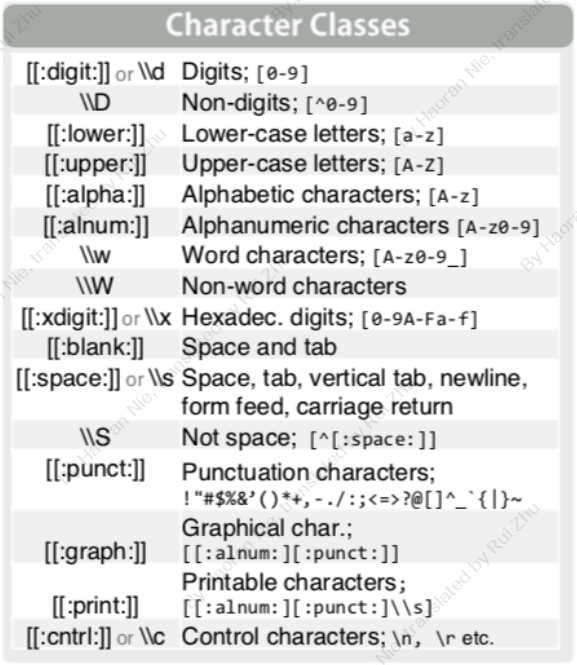
The key distinction lies in the handling of character cases and character sequence reversal. While `toupper()` and `tolower()` focus on case transformations for ASCII characters, `stri_reverse()` in `stringi` concentrates on accurately reversing character sequences, making it more suitable for handling multibyte characters and Unicode strings.

Tricks

- `stringi` The functions in the package all start with `stri_`.
- `strinr` starts with `str_`.

Regex - Regular Expression

1. Character classes:What characters are (not) matched?

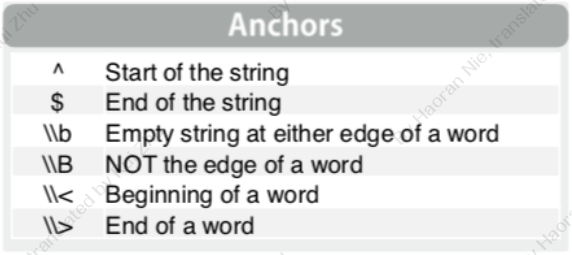


Character Classes	
[[:digit:]] or \\d	Digits; `[0-9]`
\\D	Non-digits; `[^0-9]`
[[:lower:]]	Lower-case letters; `[a-z]`
[[:upper:]]	Upper-case letters; `[A-Z]`
[[:alpha:]]	Alphabetic characters; `[A-z]`
[[:alnum:]]	Alphanumeric characters `[A-z0-9]`
\\w	Word characters; `[A-z0-9_]`
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; `[0-9A-Fa-f]`
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; `[^[:space:]]`
[[:punct:]]	Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Graphical char.; `[[:alnum:][:punct:]]`
[[:print:]]	Printable characters; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] or \\c	Control characters; `\n`, `\r` etc.

```
## 比如: [ab] 表示寻找 a 或 b
c( "abc", "chen", "liu", "blah" ) %>% str_subset( "[ab]" );

#[1] "abc"  "blah"

## 匹配并取出字符中间的数字
c( "a1334bc", "ch13e_45n", "liu", "bl00ah" ) %>% str_extract( "\\d+" );

#[1] "1334" "13"   NA     "00"

# Example 01
"abc_123_??$$^" %>% str_extract("\\s+") # Does this string include spaces?
"abc_123_??$$^" %>% str_extract("\\d+") # Numbers?
"abc_123_??$$^" %>% str_extract("\\w+") # [A-z0-9_]

[1] NA
[1] "123"
[1] "abc_123_"
```

`str_extract` : Take out the first match.

1. Matching position

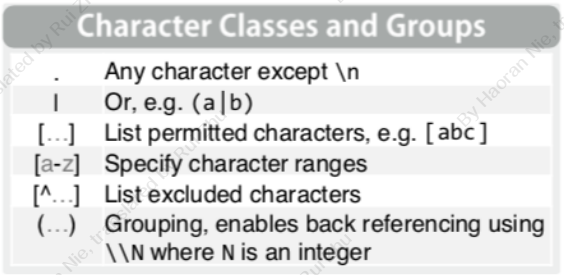



```
# Example 02
# STRING ending in 'wei'
c("chen wei hua", "chen wei", "chen") %>% str_subset("wei$")

#[1] "chen wei"

# CHARACTER ending in 'wei'
c("chen wei hua", "chen wei", "chen") %>% str_subset("wei\\b")

#[1] "chen wei hua" "chen wei"
```

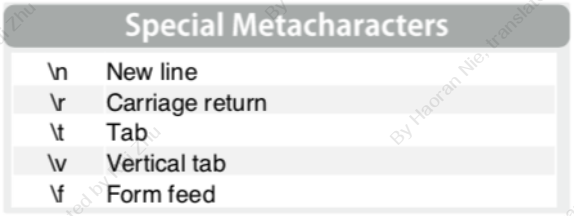
1. Number of matches

```
# Example 03
"1234abc" %>% str_extract("\\d+")
"1234abc" %>% str_extract("\\d{3}")
"1234abc" %>% str_extract("\\d{5,6}")
"1234abc" %>% str_extract("\\d{2,6}")

[1] "1234"
[1] "123"
[1] NA
[1] "1234"
```

1. Classes and groups



2. Special characters



tasks of regular expression

detect patterns： 检查目标 string 里有无 pattern

```
grep( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
grepl( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
c( "123", "abc", "wei555hua" ) %>% str_detect( "\\d+" );
#
[1] 1 3
[1]  TRUE FALSE  TRUE
[1]  TRUE FALSE  TRUE
```

count patterns：统计匹配的数量

```
x <- c("why", "video", "cross", "extra", "deal", "authority");
str_detect(x, "[aeiou]");

str_count(x, "[aeiou]");
#
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[1] 0 3 1 2 2 4
```

locate patterns (定位)

```
regexpr( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
#
[1]  1 -1  4
attr(,"match.length")
[1]  3 -1  3
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

extract patterns (抽取匹配的字串)

```
c( "123", "abc", "wei555hua" ) %>% str_extract ( "\\d+" );
c( "123", "abc", "wei555hua" ) %>% str_match ( "\\d+" );
```

useful tools

https://regexr.com/
https://regex101.com/

str_extract vs. str_match

str_extract:

- **功能**：用于从字符串中提取匹配正则表达式的部分。
- **返回值**：返回匹配到的第一个子字符串（或整个字符串）。

str_match:

- **功能**：用于从字符串中提取匹配正则表达式的全部信息，包括所有匹配的子字符串和捕获组。
- **返回值**：返回一个矩阵，每一行表示一个匹配，每一列表示一个捕获组。

```
x;
#
[1] "why"       "video"     "cross"     "extra"     "deal"
[6] "authority"

str_extract(x, "[aeiou]");

#
[1] NA  "i" "o" "e" "e" "a"

str_match(x, "(.)[aeiou](.)"); ## extract the characters on either side of the vowel ????

#
     [,1]  [,2] [,3]
[1,] NA    NA   NA
[2,] "vid" "v"  "d"
[3,] "ros" "r"  "s"
[4,] NA    NA   NA
[5,] "dea" "d"  "a"
[6,] "aut" "a"  "t"
```

str_extract_all 和 str_match_all

str_extract_all:

- **功能**：用于从字符串中提取所有匹配正则表达式的部分。

- **返回值**：返回一个列表，其中每个元素是一个字符向量，包含与正则表达式匹配的所有子字符串。

str_match_all:

- **功能**：用于从字符串中提取所有匹配正则表达式的全部信息，包括所有匹配的子字符串和捕获组。
- **返回值**：返回一个列表，其中每个元素是一个矩阵，表示一个匹配，矩阵的每一列表示一个捕获组。

```
x;
str_extract_all( x, "[aeiou]+" );
str_match_all( x, "[aeiou]+" );
```

replace patterns (匹配并替换)

```
str_replace( c( "123", "abc", "wei555hua" ) , "\\d+", "###");
str_replace_all( "123_abc_456_789" , "\\d+", "###");

#[1] "###"       "abc"       "wei###hua"
#[1] "###_abc_###_###"
```

split by patterns

```
str_split(x, "");
```

R for bioinformatics, data iteration & parallel computing

Talk 08

TOC

- for loop
- `apply` functions
- The essence of `dplyr` is traversal.
- `map` functions in `purrr` package
- Iteration and Parallel Computing

Iteration Basics

`for loop` , getting data ready

Look at this example:

```
df =
        tibble(
    a = rnorm(100),
    b = rnorm(100),
    c = rnorm(100),
    d = rnorm(100)
  )

# Calculate row means
res1 =
        vector("double", nrow(df))
for(row_idx in 1:nrow(df)){
  res1[row_idx] =
          mean( as.numeric(df[row_idx, ]))
}

res2 = c()
for(row_idx in 1:nrow(df)){
  res2[length(res2) + 1] =
          mean(as.numeric(df[row_idx, ]))
}
```

```r
# Similar to Python

# Calculate column means
res2 =
        vector("double", ncol(df))
for(col_idx in 1:ncol(df)){
  res2[col_idx] =
          mean(df[[col_idx]])
}
```

You can replace it with `for loop`:

```r
rowMeans(df)
colMeans(df)
```

Here are some other functions:

```r
rowSums(df)
colSums(df)
```

apply functions

```r
apply(X, MARGIN, FUN, ...);
```

MARGIN : 1 = 行，2 = 列；c(1,2) = 行 & 列

FUN : 函数，可以是系统自带，也可以自己写

You can use `apply` with customizable function.

```r
df %>% apply(
  .,
  2,
  function(x) {
    return(
      c(
        n = length(x),
        mean = mean(x),
        median = median(x)
      )
    )
  }
)
```

Something about `tapply()`:

The `tapply()` function in R 用于在向量的子集上应用函数，通过因子或因子列表将其拆分。It stands for "table apply" and is particularly useful for summarizing data by groups or categories.

Here's a breakdown of its usage:

```
tapply(X, INDEX, FUN)
```

用 **index** 将 **x** 分组后，用 **fun** 进行计算

- `X`: 要对其应用函数的向量（或数组）。
- `INDEX`: 定义组的因子或因子列表。These factors determine how the vector `X` is split.
- `FUN`: The function to be applied to each subset of `X`.

For example:

```
## 注意 pipe 操作符的使用
mtcars %$% tapply( mpg, cyl, mean ); ## 汽缸数 与 每加仑汽油行驶里程 的关系
#
       4        6        8
26.66364 19.74286 15.10000
```

然而，使用 `dplyr` 思路会更清晰

```
mtcars %>% group_by( cyl ) %>% summarise( mean = mean( mpg ) );
```

`tapply` 和 `dplyr` 都是基于行的操作!!

`lapply` 和 `sapply`

基于**列**的操作

输入：

- vector ： 每次取一个 element
- data.frame, tibble, matrix ： 每次取一列
- list ： 每次取一个成员

输入是 tibble

```
df %>% lapply( mean );
df %>% sapply( mean );
```

输入是 list ， 使用自定义函数

```
list( a = 1:10, b = letters[1:5], c = LETTERS[1:8] ) %>%
  sapply( function(x) { length(x) } );
```

- `lapply` 是针对列的操作
- 输入是 tibble, matrix, data.frame 时，功能与 `apply( x, 2, FUN )` 类似...

Diffrences between `apply` in base R and the package `dplyr`:

1. **`apply` functions in base R:**
   - The `apply` family of functions (`apply()`, `lapply()`, `sapply()`, `vapply()`, etc.) in base R are used for applying a function over margins of arrays or data structures like matrices, arrays, and lists.
   - `apply()` is used primarily for applying functions to the rows or columns of matrices or arrays, while `lapply()` and `sapply()` are more focused on lists.
   - 这些函数对于跨行或跨列的重复操作非常有用，而无需显式使用循环。

   Example:

   ```
   # Creating a matrix
   mat = matrix(1:12, nrow = 3, ncol = 4)

   # Applying sum function to rows (1) or columns (2) of the matrix
   apply(mat, 1, sum)  # Sums of each row
   apply(mat, 2, sum)  # Sums of each column
   ```

2. **`dplyr` package:**
   - `dplyr` is a powerful package in R for data manipulation and transformation. It provides a set of functions (`filter()`, `mutate()`, `select()`, `group_by()`, `summarize()`, etc.) that enable easy and intuitive data manipulation.
   - It's designed to work well with data frames and offers a more streamlined and readable syntax for performing common data manipulation tasks.

   Example:

   ```
   library(dplyr)

   # Creating a sample data frame
   df = data.frame(
     Name = c("Alice", "Bob", "Charlie"),
     Age = c(25, 30, 28),
   ```

```
  Salary = c(40000, 50000, 45000)
)

# Filtering and selecting specific rows and columns
filtered_df = df %>%
  filter(Age > 25) %>%
  select(Name, Salary)

filtered_df
```

This `dplyr` example filters rows where `Age` is greater than 25 and selects only the `Name` and `Salary` columns. The `%>%` operator (pipe) chains together multiple operations, making the code more readable and concise.

In summary, the `apply` family in base R is ideal for applying functions to matrices, arrays, or lists across rows or columns, while `dplyr` focuses on intuitive data manipulation operations for data frames, providing a cleaner syntax and ease of use for common data transformation tasks.

More on iteration: `purrr` package

About `purrr` (from official website https://purrr.tidyverse.org)

`purrr` enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. If you've never heard of FP before, the best place to start is the family of `map()`functions which allow you to replace many for loops with code that is both more succinct and easier to read. The best place to learn about the `map()` functions is the iteration chapter in R for data science.

Usage

The following example uses `purrr`to solve a fairly realistic problem: split a data frame into pieces, fit a model to each piece, compute the summary, then extract the R^2.

```
library(purrr)

mtcars |>
  split(mtcars$cyl) |>  # from base R
  map(\(df) lm(mpg ~ wt, data = df)) |>
  map(summary) %>%
  map_dbl("r.squared")

#>         4         6         8
#> 0.5086326 0.4645102 0.4229655
```

This example illustrates some of the advantages of `purrr` functions over the equivalents in base R:

- The first argument is always the data, so purrr works naturally with the pipe.
- All `purrr` functions are type-stable. They always return the advertised output type (`map()` returns lists; `map_dbl()` returns double vectors), or they throw an error.
- All `map()` functions accept functions (named, anonymous, and lambda), character vector (used to extract components by name), or numeric vectors (used to extract by position).

Detailed Usage

`purrr` is a powerful package in R that focuses on enhancing and simplifying the process of working with functions and vectors. Developed as part of the tidyverse ecosystem, `purrr` provides a consistent and coherent set of tools for functional programming, iteration, and working with lists and vectors.

Here are some key aspects and functionalities of `purrr`:

1. **Functional Programming:**
   - `purrr` promotes functional programming paradigms in R, enabling users to work with functions as first-class objects.
   - It provides functions like `map()`, `map2()`, `pmap()`, `walk()`, and more, which allow applying functions over elements of lists or vectors.
2. **Consistency Across Data Structures:**
   - `purrr` functions exhibit consistent behavior across various data structures, such as lists, vectors, and data frames.
   - These functions can work seamlessly with different data structures, making code more readable and maintainable.
3. **Iteration and Mapping:**
   - `map()` is a key function in `purrr` that iterates over elements of a list or vector, applying a function to each element and returning the results.
   - `map2()` is similar to `map()` but allows iterating over two vectors simultaneously.
   - `pmap()` extends this functionality to iterate over multiple vectors or lists simultaneously.
4. **Simplified and Cleaner Syntax:**
   - `purrr` functions often provide a more consistent and cleaner syntax compared to base R functions for similar operations.

- The use of the pipe `%>%` from the tidyverse allows chaining `purrr` functions together, resulting in more readable code.

5. **Working with Lists and Data Frames:**
   - `purrr` provides functions to efficiently manipulate and iterate over elements in lists and data frames.
   - Functions like `map()` and `map_dbl()` can be used to apply functions to each column of a data frame and collect results in an output structure.

Example of `map()` in `purrr`:

```r
library(purrr)

# Applying sqrt function to each element of a list
numbers = list(a = 1:5, b = 6:10)
result = map(numbers, sqrt)

result
# Output: List of 2
# $ a: num [1:5] 1 1.41 1.73 2 2.24
# $ b: num [1:5] 2.45 2.65 2.83 3 3.16
```

This code applies the `sqrt()` function to each element of the list `numbers`, returning a list with the square roots of each element.

`purrr` simplifies and enhances functional programming in R, offering a consistent and expressive way to work with functions, lists, vectors, and data frames, making data manipulation and iteration more straightforward and concise.

Examples

Here are some specific functionalities and examples of `purrr`:

1. **Mapping Functions:**

   `map( FUN )`:

   1. 遍历每列（tibble）或 slot （list），
   2. 运行 FUN 函数，
   3. 将计算结果返回至 list

   对应：`lapply`

```r
library(purrr)

# Squaring each element in a vector using map()
numbers = 1:5
squared = map(numbers, ~ .x^2)

squared
# Output: List of 5
# $ : int 1
# $ : int 4
# $ : int 9
# $ : int 16
# $ : int 25
```

2. **Mapping Functions over Multiple Inputs:**

   `map2()` allows applying a function that takes two inputs to corresponding elements of two vectors.

```r
# Multiplying elements of two vectors element-wise
vector1 = 1:5
vector2 = 6:10
product = map2(vector1, vector2, ~ .x * .y)

product
# Output: List of 5
# $ : int 6
# $ : int 14
# $ : int 24
# $ : int 36
# $ : int 50
```

3. **Working with Data Frames:**

   `map_df()` and similar functions allow applying a function to each column of a data frame and combining results into a data frame.

```r
# Creating a data frame
df = data.frame(A = 1:5, B = 6:10)

# Doubling each column in the data frame
doubled = map_df(df, ~ .x * 2)

doubled
# Output: A tibble: 5 × 2
```

```
#       A     B
#   <dbl> <dbl>
# 1     2    12
# 2     4    14
# 3     6    16
# 4     8    18
# 5    10    20
```

4. **Iteration and Applying Functions:**

   `walk()` applies a function to each element without returning a result, useful for side effects or performing operations without output.

   ```
   # Printing each element of a list using walk()
   fruits = list("apple", "banana", "orange")
   walk(fruits, print)
   # Output:
   # [1] "apple"
   # [1] "banana"
   # [1] "orange"
   ```

对应 `sapply` 的 `map_` 函数

- `map_lgl()` makes a logical vector.
- `map_int()` makes an integer vector.
- `map_dbl()` makes a double vector.
- `map_chr()` makes a character vector.

```
df %>% map_dbl( mean ); ## 注：返回值只能是单个 double 值
```

`purrr` simplifies functional programming by providing intuitive functions (`map()`, `map2()`, `walk()`, etc.) that allow iteration over elements, applying functions, and collecting results, making code more concise and readable in scenarios involving lists, vectors, and data frames.

`map` 的高阶应用

为每一个汽缸分类计算：燃油效率与吨位的关系

```
plt1 <-
  mtcars %>%
  ggplot( aes( mpg, wt ) ) +
  geom_point(  ) + facet_wrap( ~ cyl );
```

```
mtcars %>%
split( .$cyl ) %>%
map( ~ cor.test( .$wt, .$mpg ) ) %>%
map_dbl( ~.$estimate );
```

`split( .$cyl )`：由 purrr 提供的函数，将 mtcars 按 cyl 列分为三个 tibble，返回值存入 list

注意：`.` 在 pipe 中代表从上游传递而来的数据；在某些函数中，比如 `cor.test()`，必须指定输入数据，可以用 `.` 代替。

```
## 正规写法：
map( function(df) { cor.test( df$wt, df$mpg ) } )
## 简写：
map( ~ cor.test( .$wt, .$mpg ) )
```

`~` 的用法: 用于取代 `function(df)`

```
## 完整版
map_dbl( function(eq) { eq$estimate} );
## 简写版
map_dbl( ~.$estimate )
```

split 与 group_by 的区别

split:

- **功能：** split 函数用于将数据框或向量按照指定的因子或列表进行分割，生成一个列表，其中每个元素都包含原始数据的一个子集。

group_by:

- **功能：** group_by 函数通常与 dplyr 包一起使用，用于按照某一列或多列的值对数据进行分组。

(in the slide) Function reduce() and accumulate()

Both `reduce()` and `accumulate()` are powerful functions from the `purrr` package that facilitate iterative calculations over a sequence, accumulating or reducing values based on a specified function.

Here's an explanation of each:

1. **`reduce()` Function:**

- `reduce()` 将列表的元素逐个进行二元操作，从左到右累积计算，最终返回一个单一的值。
- The function provided to `reduce()` should take two arguments and return a single value.
- It starts by applying the function to the first two elements, then uses the result along with the next element, and so on, until the sequence is exhausted.

Example of `reduce()`:

```
library(purrr)

# Summing all elements in a vector using reduce()
numbers = 1:5
total_sum = reduce(numbers, `+`)

total_sum
# Output: 15 (1 + 2 + 3 + 4 + 5 = 15)
```

Here, `reduce()` adds all the elements in the `numbers` vector by applying the addition function (`+`) iteratively.

2. **`accumulate()` Function:**

- `accumulate()` is similar to `reduce()` 但返回的是一个累积计算的向量，而不是一个单一的值。它保留了每一步的计算结果。
- It applies a function cumulatively to the sequence and returns a vector of values, representing the intermediate results at each step.

Example of `accumulate()`:

```
# Calculating cumulative product of elements in a vector using accumulate()
factors = c(2, 3, 4, 5)
cumulative_product = accumulate(factors, `*`)

cumulative_product
# Output: 2 6 24 120 (2, 2*3, 2*3*4, 2*3*4*5)
```

Here, `accumulate()` applies the multiplication function (`*`) to each element in `factors`, returning a vector of cumulative products at each step.

`reduce()` aggregates a sequence into a single value based on a function, while `accumulate()` returns a sequence of intermediate results. Both functions are helpful for iterative calculations and provide different ways to process sequences of values in R.

Parallel Computing

并行计算介绍

并行计算一般需要 3 个步骤：

1. 分解并发放任务
2. 分别计算
3. 回收结果并保存

Related Packages

- `parallel` 包：检测 CPU 数量；
- `doParallel` 包：将全部或部分分配给任务
- `foreach` 包：提供 `%do%` 和 `%dopar%` 操作符，以提交任务，进行顺序或并行计算
  - `%do%` loop - foreach notation, but not parallel
  - `%dopar%` adds parallelization
- 辅助包：

  `iterators` 包：将 data.frame, tibble, matrix 分割为行/列用于提交并行任务。

Step-by-step Guidance

1. **Prepare Data:**

   Assume you have a large data frame named `my_data` that you want to process in parallel.

2. **Setup Parallel Processing:**

   Load necessary packages and initialize parallel processing capabilities.

```
library(parallel)
library(doParallel)
library(foreach)
library(iterators)


# Set the number of cores/processors to be used
num_cores = detectCores()## 检测有多少个 CPU

# Initialize parallel backend
cl = makeCluster(num_cores)## 创建了一个并行计算的集群
registerDoParallel(cl)## 将这个集群注册为后端，以便后续的并行计算使用。
```

3. **Split Data Frame into Chunks:**

   Use the `iter()` function from the `iterators` package to create an 迭代器 for chunks of your data frame.

   ```
   # Define chunk size
   chunk_size = nrow(my_data) / num_cores

   # Create an iterator for the chunks
   my_iterator = iter(my_data, by = "row", chunksize = chunk_size)
   ```

4. **Perform Parallel Computation:**

   Use `foreach()` from the `foreach` package along with `%dopar%` to apply a function to each chunk in parallel.

   ```
   # Define a function to process each chunk
   process_chunk = function(chunk) {
     # Your processing logic for each chunk goes here
     # For example: summary(chunk)
     # Replace summary() with your specific data processing task
   }

   # Apply the function to each chunk in parallel
   results = foreach(chunk = my_iterator, .combine = rbind) %dopar% {
     process_chunk(chunk)
   }
   ```

5. **Combine Results:**

   Collect and combine the results obtained from parallel processing.

   ```
   # Combine or process the results obtained from parallel computation
   final_result = do.call(rbind, results)

   # Close the parallel cluster
   stopCluster(cl)
   ```

Replace the `process_chunk()` function with your specific data processing task. This approach parallelizes the processing of chunks of the data frame across multiple cores, allowing for faster computations, especially with large datasets.

Note:

- When the task is completed, the allocated CPU core is reclaimed.
- Ensure that your specific data processing task is compatible with parallelization and that the benefits of parallel computing outweigh the overhead of parallelization.

- Also, consider potential dependencies or shared resources among iterations when parallelizing computations.

(in the slide) Function `foreach()`

Simple usage

Description `%do%` and `%dopar%` are binary operators that operate on a `foreach` object and an `R` expression. The expression, `ex`, is evaluated multiple times in an environment that is created by the `foreach` object, and that environment is modified for each evaluation as specified by the `foreach` object. `%do%` evaluates the expression sequentially, while `%dopar%` evaluates it in parallel. The results of evaluating `ex` are 求值的结果以列表形式返回，但可以通过`.confine` 参数进行修改

`.combine = 'c'` 参数的可能值:

- 'c'：将返回值合并为 vector ； 当返回值是单个数字或字符串的时候使用
- 'cbind'：将返回值按列合并
- 'rbind'：将返回值按行合并
- 默认情况下返回 `list`

Usage

```
foreach(
  ...,
  .combine,
  .init,
  .final = NULL,
  .inorder = TRUE,
  .multicombine = FALSE,
  .maxcombine = if (.multicombine) 100 else 2,
  .errorhandling = c("stop", "remove", "pass"),
  .packages = NULL,
  .export = NULL,
  .noexport = NULL,
  .verbose = FALSE
)
e1 %:% e2
when(cond)
obj %do% ex
obj %dopar% ex
times(n)
```

嵌套 (nested) foreach

Expanded knowledge, not featured on slide, for understanding only.

Nested `foreach` loops in R allow for the iteration over multiple levels of nested structures or combinations of iterators. This approach is particularly useful when dealing with hierarchical data or when you need to perform computations on multiple levels of nested objects simultaneously.

有些情况下需要用到嵌套循环，使用以下语法：

```
foreach( ... ) %:% {
  foreach( ... ) %dopar% {

  }
}
```

1. **Nested Iteration:**

   `foreach` supports nesting, allowing you to iterate over multiple levels of nested structures, such as lists within lists or matrices within lists.

   ```
   library(foreach)

   # Example: Nested foreach loop iterating over a list of lists
   outer_list = list(list(a = 1, b = 2), list(c = 3, d = 4))

   foreach(inner_list = outer_list) %:% {
     foreach(element = inner_list) %do% {
       # Process each element within the nested structure
       print(element)
     }
   }
   ```

   This code iterates over each element of `outer_list`, which contains inner lists. Within each inner list, it iterates over the elements.

2. **Combining Iterators:**

   You can combine different iterators using `%:%` to create nested iterations.

   ```
   # Example: Nested foreach loop with combined iterators
   values = 1:3
   letters = letters[1:4]

   foreach(i = values) %:% foreach(letter = letters) %do% {
     # Perform operations using both iterators
     print(paste("Value:", i, "| Letter:", letter))
   }
   ```

This code creates nested iterations, iterating over `values` and `letters` simultaneously.

3. **Applying Nested Functions:**

   Nested `foreach` loops are valuable when applying functions or performing operations that require iterating over multiple levels of nested data structures or combinations.

   ```
   # Example: Applying a function with nested foreach loops
   matrix_list = list(matrix(1:4, nrow = 2), matrix(5:8, nrow = 2))

   foreach(mat = matrix_list) %:% foreach(element = as.vector(mat)) %do% {
     # Perform computations on each element of each matrix
     print(element * 2)
   }
   ```

   Here, it iterates over a list of matrices and then iterates over each element within the matrices to perform computations.

Nested `foreach` loops in R allow for flexible and efficient iterations over hierarchical or nested structures, enabling complex computations, data manipulations, or simulations involving multiple levels of nested objects or iterators.

Review of the course “R for Data Science” Part 03(Talk 09~12)

By Haoran Nie @ HUST Life ST

Partically translated by Rui Zhu @ HUST Life ST

双语版

This work is licensed under CC BY-NC-SA 4.0

R for bioinformatics, data visualisation

Talk 09

TOC

- basic plot functions
- basic `ggplot2`
- special letters
- equations
- advanced `ggplot2`

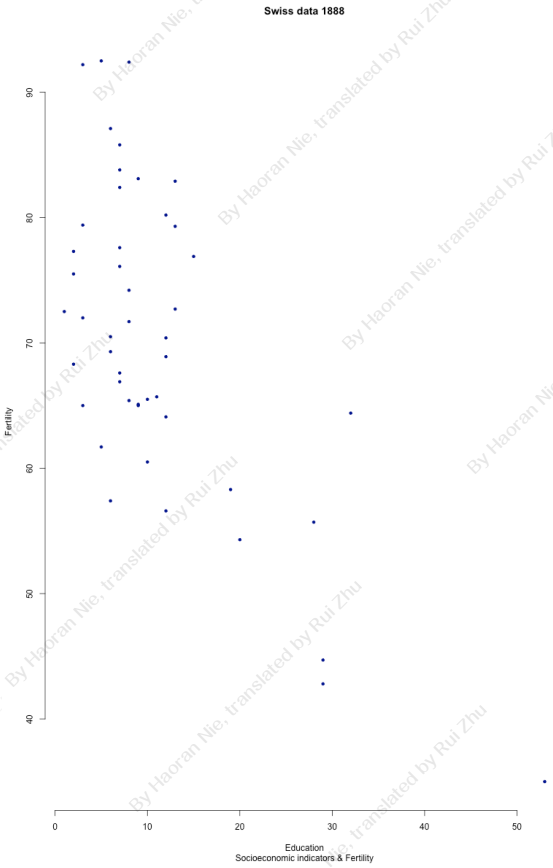
Basic plot functions using R

Dot plot 散点图

An example:

```
with(
  swiss,
  # 将数据框 swiss 中的列作为环境，可以直接使用列名而不需要再加上 swiss$ 前缀。
  plot(
    Education,
    Fertility,
    type = "p",
    main = "Swiss data 1888",
    sub = "Socioeconomic indicators & Fertility",
    xlab = "Education",
```

```
    ylab = "Fertility",
    col = "darkblue",
    xlim = range( Education ),
    ylim = range( Fertility ),
    pch = 20,
    frame.plot = F## 去除图的边框
  )
)
```



Function usage:

```
## Default S3 method:
plot(
  x,
  y,
  type = "p",
  xlim = NULL, ylim = NULL,
  log = "",
  main = NULL, sub = NULL,
  xlab = NULL, ylab = NULL,
  ann = par("ann"), ##par("ann")=T
```

```
  axes = TRUE, frame.plot = axes,
  panel.first = NULL, panel.last = NULL, asp = NA,
  xgap.axis = NA, ygap.axis = NA,
  ...
)
# Default Parameters are listed.
```

Arguments

PARAMETERS	DETAILS
`x, y`	the `x` and `y` arguments provide the x and y coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function `xy.coords` for details. If supplied separately, they must be of the same length.
`type`	1-character string giving the type of plot desired. The following values are possible, for details, see `plot`: `"p"` for points, `"l"` for lines, `"b"` for both points and lines, `"c"` for empty points joined by lines, `"o"` for 重叠绘制点和线, `"s"` and `"S"` for stair steps and `"h"` for像柱状图一样的垂直线。 Finally, `"n"` does not produce any points or lines.
`xlim`	the x limits (x1, x2) of the plot. Note that `x1 > x2` is allowed and leads to a '反向轴'.The default value, `NULL`, indicates that the range of the finite values to be plotted should be used.
`ylim`	the y limits of the plot.
`log`	a character string which contains `"x"` if theX轴变为对数的， `"y"` if the Y轴变为对数的 and `"xy"` or `"yx"` if both axes 变为对数的
`main`	a main title for the plot, see also `title`.
`sub`	a subtitle for the plot.
`xlab`	a label for the x axis, defaults to a description of `x`.
`ylab`	a label for the y axis, defaults to a description of `y`.
`ann`	一个逻辑值，指示默认注释（标题以及x轴和y轴标签）是否应显示在图上。
`axes`	逻辑值，指示是否应在图上绘制两个轴。 Use graphical parameter `"xaxt"` or `"yaxt"` to suppress just one of the axes.
`frame.plot`	a logical indicating whether a box should be drawn around the plot.
`panel.first`	要在设定绘图轴之后但在进行任何绘图之前求值的"表达式"。这对于绘制背景网格或散点图平滑非常有用。注意，这是通过惰性求值来实现的：从其他`plot`方法传递这个参数可能不起作用，因为它可能过早地被求值。
`panel.last`	an expression to be evaluated after plotting has taken place but before the axes, title and box are added. See the comments about `panel.first`.
`asp`	Y/X宽高比, see `plot.window`.
`xgap.axis`, `ygap.axis`	the `x/y` axis gap factors, passed as `gap.axis` to the two `axis()` calls (when `axes` is true, as per default).

You can also use `ggplot` to draw the plot above:

```
ggplot(
  swiss,
  aes(x = Education, y = Fertility)
) +
geom_point() +
scale_x_log10() +
scale_y_log10() +
xlab("Education") +
ylab("Fertility") +
ggtitle "Swiss data 1888")
```

High-level and low-level

- **high level**: 绘图函数在图形设备上创建新的绘图
- **low level**: 绘图函数向现有绘图添加更多信息

Low level plots

- `points`：点图
- `lines`：线图
- `abline`：直线
- `polygon`：多边形
- `legend`：图例
- `title`：标题
- `axis`：轴

High level plots

- `plot`：通用画图函数
- `pairs`
- `coplot`
- `qqnorm`
- `hist`
- `dotchart`
- `image`
- `contour`

注：可以用 `add = TRUE` 参数（如果可用）将 high level 函数强制转换为 low level

图形相关参数（系统函数）

`par()` 函数：显示或修改当前**图形设备**的参数。用以下命令查看支持的内容：

```
par( c( "mar", "bg" ) ); ## 显示指定参数的值
## 显示所有参数
par();
```

调整 par() 参数前请备份

par() 用于指定全局参数，因此在改变前尽量备份

```
oldpar <- par(); ## 备份

do some changes here ...

## 恢复
par( oldpar );
```

常用图形参数及调整: margin

图形边距（figure margins ）

分别指定下 -> 左 -> 上 -> 右的边距，即从下面开始，顺时针移动。

```
# 单位是:  text lines
par( mar = c( 5.1, 4.1, 4.1, 2.1 )); ## 设置新 martin
# 单位是:  inch
par( mai = c( 5.1, 4.1, 4.1, 2.1 )); ## 设置新 martin
```

常用图形参数及调整: 多 panel

画 2x3 共 6 个 panel，从左到右。(2 行 3 列)

```
par( mfrow=c(2,3) );
for( i in 1:6 )
  plot( sample( 1:10, 10 ), main = i );
```

画 2x3 共 6 个 panel，从上到下。(2 行 3 列)

```
par( mfcol=c(2,3) );
for( i in 1:6 )
  plot( sample( 1:10, 10 ), main = i );
```

重要概念: 图形设备

图形设备是指图形输出的设备，可以将**图形设备**理解为**保存格式**。

默认设备是:

- X11() : Unix
- windows() : windows

- quartz()：OS X

图形显示在**显示器**上。

图形设备：cont.

常用其它设备有：

- pdf()
- png()
- jpeg()

分别对应输出文件格式。

常用图形设备：pdf()

使用方法如下：

```
pdf( file = "/path/to/dir/<file_name>.pdf", height = 5, width = 5 ); ## 创建一个新设备/ pdf 文件
plot(1:10); ## 作图;
dev.off(); ## 关闭设备
```

说明

1. 默认文件名为 `Rplots.pdf` ，
2. `dev.off()` 必须关闭。关闭后，返回到最近使用的图形设备
3. `height` 和 `width` 参数的单位是 inch
4. 如果运行多个 high level 作图命令，则会产生**多页 pdf**

请尽量使用 pdf 作为文件输出格式

1. 生信图片大多是点线图，适合保存为矢量格式（如 pdf, ps 等）；
2. 矢量图可无限放大而不失真（变成像素）；
3. 可由 Adobe Illustrator 等矢量图软件进行编辑

ggplot2

You should know that

1. `xy -axes` will automatically adjust based on the data you give;
2. `ggplot2` plotting results can be saved in variables and more layers can be added;
3. Layers using their own data need to be specified with `data =`, while global data is not. You can just specify it via `ggplot(data = data.frame(...))`.

Some basic parameters of ggplot2

1. **aes** (aesthetics) 美学：控制全局参数，包括：x,y 轴使用的数据，颜色（colour, fill），形状（shape），大小（size），分组（group）等等；
2. **图层**：`geom_<layer_name>`；每张图可有多个图层（此处有两个）；图层可使用全局数据 (df) 和参数 (aes)，也可以使用自己的 aes 和数据；

   `geom_<name_of_the_layer>`:

- `geom_point` , `geom_line`: 点线图，用于揭示两组数据间的关系；
- `geom_smooth`：常与 `geom_point` 联合使用，揭示数据走势
- `geom_bar`： bar 图
- `geom_boxplot`：箱线图，用于比较 N 组数据，揭示区别
- `geom_path`：与 `geom_line` 相似，但也可以画其它复杂图形
- `geom_histogram`, “ geom_density “‘：数据的分布，也可用于多组间的比较
- ……

1. `scale` - Display Control 其它控制函数

   `scale_< 控制内容 >_< 控制手段 >`,

   e.g. `scale_color_manual()`：以**手选方式**控制 **颜色**

- `scale_color_...`
  - `..._gradient()`: 为不同数量的变量及其颜色使用渐变色。
    - `..._gradient2()`
    - `..._gradientn()`
  - `..._brewer`: 使用默认调色板。
- `scale_fill_...`

  - `scale_fill_manual()`：允许用户手动指定每个类别或组的填充颜色。
  - `scale_fill_discrete()`：用于离散变量，自动为每个类别分配不同的颜色。
  - `scale_fill_continuous()`：用于连续变量，根据连续数值分配颜色。
  - `scale_fill_gradient()`、
    `scale_fill_gradient2()`、
    `scale_fill_gradientn()`：用于创建渐变颜色填充，适用于表示连续数值范围。

- `scale_shape_...`
  - `scale_shape_manual()`：允许用户手动指定每个因子水平的形状。
  - `scale_shape_discrete()`：用于离散变量，自动为每个因子水平分配不同的形状。
  - `scale_shape_continuous()`：用于连续变量，根据连续值分配形状（不常用）。

- `scale_size_...`

`fill` 与 `colour` 有什么区别???

- `colour` 定义几何图形**轮廓**的颜色（形状的“笔划”）
- `fill` 定义用于**填充**几何图形的颜色
- point 一般只有一个**颜色 没有填充**
- However, point shapes **21–25** that include **both** a colour and a fill.
- `colour` 在 shape = 21 时，为描边（stroke）色；
- 可用 `stroke` 控制线条粗细；

调色板和其他包中的相应函数

Included in ggplot2 `scale_color_hue`, `scale_color_manual`, `scale_color_grey`, `scale_colour_viridis_d`, `scale_color_brewer` ...

From the `RColorBrewer` package `scale_color_brewer(palette = "<palette name>")`

From the `viridis` package `scale_color_viridis(discrete=TRUE, option="<palette name>")`

Other packages ...

- `paletteer` package: `scale_color_paletteer_xx` functions
- `ggsci` package
- `ggsci`: 论文发表用的色板!!!
  - contents
    `scale_color_<journal>` 和 `scale_fill_<journal>` functions and color palettes

```
- supported journals

      * NPG `scale_color_npg()`, `scale_fill_npg()`

      * AAAS, NEJM, Lancet, JAMA ...
```

1. `size`
2. `shape`

Question:

像 `size`、`colour` 等参数。可以在 `aes()` 里面或外面，有什么区别?

Answer:

在内部时，以指定列的值确定**大小**，或按 factor 的数量确定颜色、形状的**数量**。

放在 `aes()` 外部意味着这些属性与特定的固定值相关联，而不是与数据的某个变量相关联。

Coordinate System 坐标系

1. 线性坐标系

- `coord_cartesian()`,
  默认的坐标系统，可使用 xlim, ylim 等参数，实现**缩放局部**
- `coord_flip()`,
  翻转 x 轴和 y 轴的位置，使得原本在 x 轴上的变量显示在 y 轴上，反之亦然。
- `coord_fixed(ratio = y/x)`
  用特定的长宽比例 (aspect ratio) 作图

1. Nonlinear coordinate system

- `coord_trans(x = "identity", y = "identity", ...)`

x 和 y: 分别指定 x 轴和 y 轴的变换类型。默认为 "identity"，表示不进行变换。

limx,limy: 限制 xy 的显示范围

- coord_polar(theta = "x", start = 0, direction = 1, clip = "on")

创建极坐标图，默认为 coord_polar("x")

柱图变饼图

```
base <- ggplot(mtcars, aes(factor(1), fill = factor(cyl))) +
  geom_bar(width = 1) + theme(legend.position = "none") +
  scale_x_discrete(NULL, expand = c(0, 0)) +
  scale_y_continuous(NULL, expand = c(0, 0))

base + coord_polar(theta = "y") ## 变饼图
```

- coord_map()

  World Map

faceting

panel, strip, axis, tick, tick label, axis label...

- facet_grid(rows ~ cols)

  创建一个由行和列构成的网格布局，使得不同的子图（面板）可以基于数据的一个或多个分类变量进行排列。

  - rows: 行变量，用于在垂直方向上分割面板。
  - cols: 列变量，用于在水平方向上分割面板。

- facet_wrap()

  用于指定行数、列数和方向。

  ```
  facet_wrap(
    facets,
    nrow = NULL,
    ncol = NULL,
    scales = "fixed",
    shrink = TRUE,
    labeller = "label_value",
    as.table = TRUE,
    switch = NULL,
    drop = TRUE,
  ```

```
    dir = "h",
    strip.position = "top"
  )

  # Default parameters are listed.
```

layered grammer (图层语法) 的成分

- 图层 (geom_xxx)
- scale (scale_xxx)
- 坐标系统
- faceting (facet_xxx)

如何在一张图中画多个 panel?

key requirements for multi-panel plots

- order / position
- labeling
- layout

combine multiple plots Useful packages:

- `gridExtra`
- `cowplot`
- `grid`
- `lattice`

`cowplot::plot_grid` parameters

```
plot_grid(
  plot1, plot2,
  ...,
  plotlist = NULL,
  align = c("none", "h", "v", "hv"),
  axis = c("none", "l", "r", "t", "b", "lr", "tb", "tblr"),
  nrow = NULL,
  ncol = NULL,
```

```
  rel_widths = 1,
  rel_heights = 1,
  labels = NULL,
  label_size = 14,
  label_fontfamily = NULL,
  label_fontface = "bold",
  label_colour = NULL,
  label_x = 0,
  label_y = 1,
  hjust = -0.5,
  vjust = 1.5,
  scale = 1,
  greedy = TRUE,
  byrow = TRUE,
  cols = NULL,
  rows = NULL
)
```

- `plot1, plot2, ...`: 这些是你想组合在一起的图形对象。
- `nrow` 和 `ncol`: 指定网格的行数和列数。
- `labels`: 用于每个子图的标签。默认是"AUTO"，自动为每个子图创建标签。
- `label_size`: 设置标签的字体大小。
- `...`: 其他参数，例如 `align` 和 `rel_widths`，用于微调图形的布局。

用 draw_plot 调整 graph 的相对大小

```
plot <-
  ggdraw() +
  draw_plot(plot.iris, x=0, y=.5, width=1, height=0.5) +
  draw_plot(sp, 0, 0, 0.5, 0.5) +
  draw_plot(bp, 0.5, 0, 0.5, 0.5) +
  draw_plot_label(
      c("A", "B", "C"), c(0, 0, 0.5), c(1, 0.5, 0.5), size = 15
  );
```

`draw_plot(plot, x = 0, y = 0, width = 1, height = 1)` 详解:

- plot: the plot to place (ggplot2 or a gtable)
- x: The x location of the lower left corner of the plot.
- y: The y location of the lower left corner of the plot.
- width, height: the width and the height of the plot

use gridExtra::grid.arrange to arrange multiple graphs

```
grid.arrange(
    plot1, plot2,
    ...,
    nrow = 1,
    ncol = 1,
    top = NULL,
    bottom = NULL,
    ...)
```

- plot1, plot2, ... : 这些是你想组合在一起的 grid 图形对象。
- nrow 和 ncol: 指定网格的行数和列数。
- top 和 bottom: 可以用来添加顶部和底部标题。
- ...: 其他参数，可以用于微调图形的布局和样式。
- use layout_matrix parameter in grid.arrange
  - layout_matrix 参数接受一个矩阵，其中的每个元素代表页面上的一个单元格。
  - 你可以通过在这个矩阵中指定数字来控制哪些图形的位置和占据空间。

```
grid.arrange(bp, dp, vp, sc,
             ncol = 2,  ## two columns
             layout_matrix = cbind(c(1,1,1), c(2,3,4)) ## specify the layout
             );
```

Different layouts

Nothing to explain, for it's too intricate.

在图中加入公式和统计信息

Similar to LaTeX

You can just type the formula as exactly what you type in LaTeX, and using `annotiate()` function to add it in your plot.

Remember to attach the library `latex2exp`

```
fig1 =
  fig1 +
  annotate(
    "text",
```

```
    x = 25,
    y = 15,
    label = paste0("y = ", eq, "x + (", intercept, ")\n","Shaded areas are confidence interval
    family = "Aptos Serif"
  )

... +
labs(
    x = TeX("Position of $P_2/\\phi$"),
    y = TeX("Light Intensity/$10^{-7}$A"),
    title = TeX("Intensity of Light with Different $\\phi$")
  )
```

Something about `hjust` and `vjust`

```
td = expand.grid(
    hjust=c(0, 0.5, 1),
    vjust=c(0, 0.5, 1),
    angle=c(0, 45, 90),
    text="text"
)

ggplot(td, aes(x=hjust, y=vjust)) +
    geom_point() +
    geom_text(aes(label=text, angle=angle, hjust=hjust, vjust=vjust)) +
    facet_grid(~angle) +
    scale_x_continuous(breaks=c(0, 0.5, 1), expand=c(0, 0.2)) +
    scale_y_continuous(breaks=c(0, 0.5, 1), expand=c(0, 0.2))
```

In talk09

```
## 计算 ...
m = lm(Fertility ~ Education, swiss);
c = cor.test( swiss$Fertility, swiss$Education );

## 生成公式
eq <- substitute( atop( paste( italic(y), " = ",  a + b %.% italic(x), sep = ""),
                        paste( italic(r)^2, " = ", r2, ", ", italic(p)==pvalue, sep = "" ) ),
                  list(a = as.vector( format(coef(m)[1], digits = 2) ),
                       b =  as.vector(  format(coef(m)[2], digits = 2) ),
                       r2 =  as.vector( format(summary(m)$r.squared, digits = 2) ),
                       pvalue =  as.vector( format( c$p.value , digits = 2) ) ) )
```

```
    );

## 用 as.expression 对公式进行转化  !!!!
eq <- as.character(as.expression(eq));

## 作图，三个图层；特别是 geom_text 使用自己的 data 和 aes ...
ggplot(swiss, aes( x = Education,  y = Fertility ) ) +
        geom_point( shape = 20 ) +
        geom_smooth( se = T ) + ## smooth line ...
        geom_text( data = NULL,
                   aes( x = 30, y = 80, label= eq, hjust = 0, vjust = 1), ## hjust, vjust ???
                   size = 4, parse = TRUE, inherit.aes=FALSE); ## 注意:  parse = TRUE !!!
```

equation 的其它写法（更复杂难懂）

```
## 计算 ...
m = lm(Fertility ~ Education, swiss);
c = cor.test( swiss$Fertility, swiss$Education );

## 生成公式
eq <- substitute( atop( italic(y) == a + b %.% italic(x),
                        italic(r)^2~"="~r2*","~italic(p)==pvalue ),
                  list(a = as.vector( format(coef(m)[1], digits = 2) ),
                       b =  as.vector(  format(coef(m)[2], digits = 2) ),
                       r2 =  as.vector( format(summary(m)$r.squared, digits = 2) ),
                       pvalue =  as.vector( format( c$p.value , digits = 2) ) )
    );
## 用 as.expression 对公式进行转化  !!!!
eq <- as.character(as.expression(eq));

## 作图，三个图层；特别是 geom_text 使用自己的 data 和 aes ...
ggplot(swiss, aes( x = Education,  y = Fertility ) ) +
        geom_point( shape = 20 ) +
        geom_smooth( se = T ) + ## smooth line ...
        geom_text( data = NULL,
                   aes( x = 30, y = 80, label= eq, hjust = 0, vjust = 1), ## hjust, vjust ???
                   size = 4, parse = TRUE, inherit.aes=FALSE); ## 注意:  parse = TRUE !!!
```

分类	R的表达式	显示结果
代数符号	`expression(x + y)`	$x+y$
	`expression(x - y)`	$x-y$
	`expression(x * y)`	xy
	`expression(x / y)`	x/y
	`expression(x %+-% y)`	$x \pm y$
	`expression(x %/% y)`	$x \nabla \cdot y$
	`expression(x %*% y)`	$x \times y$
	`expression(x %.% y)`	$x \cdot y$
	`expression(x[i])`	x_i
	`expression(x^2)`	x^2
	`expression(sqrt(x))`	$\sqrt{x}$
	`expression(sqrt(x,y))`	$\sqrt[y]{x}$
	`expression(list(x,yz))`	x, y, z

希腊字符

```
library(ggplot2);
greeks <- c("Alpha", "Beta", "Gamma", "Delta", "Epsilon", "Zeta",
            "Eta", "Theta", "Iota", "Kappa", "Lambda", "Mu",
            "Nu", "Xi", "Omicron", "Pi", "Rho", "Sigma",
            "Tau", "Upsilon", "Phi", "Chi", "Psi", "Omega");

dat <- data.frame( x = rep( 1:6, 4 ), y = rep( 4:1, each = 6), greek = greeks );

plot2 <-
  ggplot( dat, aes(x=x,y=y) ) + geom_point(size = 0) +
    # 画希腊字符，注意下面两行代码的区别
    geom_text( aes( x, y + 0.1, label = tolower( greek ) ), size = 10, parse = T ) +
    geom_text( aes( x, y - 0.1, label = tolower( greek ) ), size = 5  );

##
```

R for bioinformatics, data summarisation and statistics

Talk 10

TOC

- Data summarisation functions (vector data)
  - median, mean, sd, quantile, summary
- Graphical data summarisation (two-D data/ tibble/ table)
  - dot plot
  - smooth
  - linear regression
  - correlation & variance explained
  - groupping & bar/ box/ plots
- statistics
  - parametric tests
    - t-test
    - one way ANNOVA
    - two way ANNOVA
    - linear regression
    - model / prediction / coefficients
  - non-parametric comparison

Vector Summarization

Describe Normal Distribution

You can use `mean` and `sd` to describe normal distributions.

- 是对称的。
- 均值和中位数是一样的。
- 最常见的值接近平均值; 不太常见的价值观与之相去甚远。
- 标准差表示平均值到拐点的距离

Functions to generate random normal distrubions

```
# 生成 10000 个随机数字，使其 mean = 0, sd = 1，且为 normal distribution ...
x <- rnorm(10000, mean = 0, sd = 1);
ggplot( data.frame( data = x ), aes( data ) ) + geom_density(  );
```

Other regular distributions

1. Uniform Distributions 均匀分布

```
# 为向量 q 中的值生成 CDF 概率
pnorm(q, mean = 0, sd = 1)

# 生成向量 p 中概率的分位数
qnorm(p, mean = 0, sd = 1)

# 生成向量 x 中值的概率密度函数
dnorm(x, mean = 0, sd = 1)
```

1. Non-parametric Distributions 非参数分布

```
bi =
        c(7, 3, 2, 1, 7,
    3, 4, 5, 7, 6,
    2, 2, 1, 3, 7,
    2, 6, 8, 2, 7,
    2, 2, 1, 3, 5,
    8, 2, 6, 7, 8,
    6, 2, 8, 7, 9,
    2, 7, 5, 1, 8,
    8, 2, 3, 7, 3
   )
ggplot(
  data.frame(dat = bi),
  aes(dat)) +
geom_density()
```

uniform distribution 的各种函数

注：以下函数中的 n 需要自行决定

```
# generate n random numbers between 0 and 25
runif(n, min = 0, max = 25)
```

```r
# generate n random numbers between 0 and 25 (with replacement)
sample(0:25, n, replace = TRUE)

# generate n random numbers between 0 and 25 (without replacement)
sample(0:25, n, replace = FALSE)
```

other distributions, cont.

```r
n <- 10000;
uni <- tibble( dat = runif(n), type = "uni" );
norm <- tibble( dat = rnorm(n), type = "norm" );
binom <- tibble( dat = rbinom(n, size = 100, prob = 0.5), type = "binom" );
poisson <- tibble( dat = rpois(n, lambda = 4), type = "poisson" );
exp <- tibble( dat = rexp(n, rate = 1) , type = "exp");
gamma <- tibble( dat = rgamma(n, shape = 1) , type = "gamma");
```

量化描述数据

- **mean**: aka average, is the sum of all of the numbers in the data set divided by the size of the data set;
- **median**: The median is the value that is in the middle when the numbers in a data set are sorted in increasing order;
- **sd**: standard deviation;
- **var**: measures how far a set of numbers are spread out;
- **range**: range of values.

量化描述函数

```r
mean( norm$dat );
median( norm$dat );
  mode( norm$dat ); ## 确定给定对象的存储模式 "numeric", "character", "list"

sd(norm$dat);
var(norm$dat);
range(norm$dat);
```

quantile and summary

```r
quantile( norm$dat );
```

```r
## quantile 还接受其它参数
quantile( norm$dat, probs = seq(0, 1, length = 11));

## summary ...
summary( norm$dat );
## summary 也可应用于非数值
summary( combined$type );
## summary 可应用于整个表格; 相当于对每列进行 summary ...
summary( combined );
```

table 函数

返回 vector 当中 unique 值和它们的出现次数

count in dplyr

ntile 函数的参数

... *tile 函数都是 equal size

cut 函数

按指定的间隔 (breaks) 对数据进行分割。

不仅可用于 equal distance，还可以用于任意间距

Statistics

Parametric tests

parametric test 的要求

1. 随机取样
2. 值或 residuals 为正态分布；residules 是指观察值与预测值 (mean) 之差
3. 有相同的 variance 方差

how to detect outlier ??

一个很模糊的定义：Outliers are extreme values that fall a long way outside of the other observations. For example, in a normal distribution, outliers may be values on the tails of the distribution.

对于 normal distribution，通常 mean +- 2 or 3 * sd

IQR = s["3rd Qu."]-s["1st Qu."]

outlier: s["1st Qu."] - 1.5 * IQR, s["3rd Qu."] + 1.5 * IQR

`t-test` 检测分布是否与预期一致; e.g., whether the number of steps per day for boys is significantly different from 10,000.

检验评估两个样本的均值是否有显著差异，假设样本服从正态分布，方差近似相等。

There are different types of t-tests in R, depending on the nature of the comparison:

1. **One-Sample t-test:**

   用于确定单个样本的均值与已知或假设的总体均值是否存在显著差异。

   Example:

   ```
   RCopy code
   # One-sample t-test example
   sample_data = c(17, 21, 19, 23, 20, 18, 22)
   t.test(sample_data, mu = 20)
   ```

   This code performs a one-sample t-test on `sample_data` to test if its mean differs significantly from 20.

2. **Independent Samples t-test (or Two-Sample t-test):**

   比较两个独立组的均值，以确定它们是否存在显著差异.

   Example:

   ```
   RCopy code
   # Independent samples t-test example
   group1 = c(23, 25, 28, 22, 20)
   group2 = c(18, 21, 24, 19, 17)
   t.test(group1, group2)
   ```

   This code performs an independent samples t-test on `group1` and `group2` to test if their means are significantly different.

3. **Paired t-test:**

   比较两个相关组的均值（例如，测量前和测量后），以确定它们是否有显著差异。

   Example:

   ```
   RCopy code
   # Paired t-test example
   before = c(32, 28, 30, 29, 31)
   after = c(30, 25, 28, 27, 29)
   t.test(before, after, paired = TRUE)
   ```

This code performs a paired t-test on `before` and `after` to test if there's a significant difference.

The `t.test()` function in R is used to conduct these t-tests. 它返回检验统计量 (t 值)、自由度、p 值和置信区间，从而深入了解观察到的差异是否具有统计显著性。

在 R 中执行 t 检验时，确保满足正态性和等方差的假设对于获得可靠的结果是很重要的。如果违反了这些假设，那么替代检验或数据转换可能更合适。

One-way ANNOVA In R, the one-way analysis of variance (ANOVA) iS 用于检验三个或三个以上独立（不相关）组的均数之间的显著差异。它评估这些群体的平均水平是否彼此显著不同。

The one-way ANOVA assumes that the data meet certain assumptions, including:

- -正态性：每组应遵循正态分布。
- -方差同质性：每组内的方差应大致相等。
- -独立性：每组内的观测值应相互独立。

Here's an example of performing a one-way ANOVA in R:

```r
# Example of one-way ANOVA
group1 = c(15, 20, 25, 30, 35)
group2 = c(10, 18, 25, 32, 40)
group3 = c(12, 22, 28, 32, 38)

# Combining data into a data frame
my_data = data.frame(
  Values = c(group1, group2, group3),
  Group = factor(rep(1:3, each = 5))  # Creating a factor for groups
)

# Performing one-way ANOVA
result_anova = aov(Values ~ Group, data = my_data)
summary(result_anova)
```

Explanation of the code:

1. The data for three groups (`group1`, `group2`, `group3`) are created.
2. The data are combined into a data frame (`my_data`) where the `Values` column contains the measurements and the `Group` column represents the group labels as a factor.

3. The `aov()` function is used to perform the one-way ANOVA, specifying the formula `Values ~ Group`, indicating that `Values` is the dependent variable and `Group` is the independent variable.
4. `summary(result_anova)` provides the ANOVA table with the F-statistic, degrees of freedom, p-value, and other relevant statistics.

The output from `summary(result_anova)` will includeF-统计量、自由度、p-值和组内变异性，允许您确定组间均值是否有显著差异。

若 p 值小于选定的显著性水平（通常为 0.05），则表明组均值之间存在显著差异。Additionally, post-hoc tests like Tukey's HSD test or pairwise t-tests can be performed to identify which specific groups differ significantly from each other after obtaining a significant result in the ANOVA.

Two-way ANNOVA A two-way analysis of variance (ANOVA) in R 用于检验两个分类自变量（因子）之间对连续因变量的交互作用效应。

Here's an example:

```r
# Example of two-way ANOVA
# Assume we have a dataset with 'Treatment', 'Gender', and 'Response' variables

# Creating sample data
set.seed(123)
Treatment = rep(c("A", "B", "C"), each = 20)
Gender = rep(c("Male", "Female"), times = 30)
Response = rnorm(60, mean = c(50, 60, 70), sd = 10)

# Combining data into a data frame
my_data = data.frame(Treatment, Gender, Response)

# Performing two-way ANOVA
result_anova = aov(Response ~ Treatment + Gender + Treatment:Gender, data = my_data)
summary(result_anova)
```

Explanation of the code:

1. Sample data is created with three variables: `Treatment`, `Gender`, and `Response`.
2. The data are combined into a data frame (`my_data`), where `Treatment` and `Gender` are categorical factors, and `Response` is the continuous dependent variable.
3. The `aov()` function performs the two-way ANOVA. The formula `Response ~ Treatment + Gender + Treatment:Gender` specifies the main effects of `Treatment` and `Gender`, as well as their interaction effect.

4. `summary(result_anova)` provides the ANOVA table with F-statistics, degrees of freedom, p-values, and other statistics for each factor and their interaction.

The output from `summary(result_anova)` will include 治疗和性别的主要影响的信息，以及它们之间的交互作用效果。它支持您确定每个因子是否独立存在显著效应，以及它们的交互作用是否显著影响" 响应" 变量。

The interpretation of a two-way ANOVA 包括分析与每个因子及其交互作用相关联的 p 值。显著的 p 值表示相应的因子或交互作用对因变量有显著的影响。

Additionally, post-hoc tests or further analyses can be conducted to explore specific comparisons between groups or factors after obtaining significant results in the ANOVA.

Linear Regression 线性回归是一种统计方法，通过对观测数据拟合线性方程来模拟因变量和一个或多个自变量之间的关系。

In R, linear regression can be performed using the `lm()` function, which stands for "linear model."

Here's an example:

```r
# Example of simple linear regression
# Suppose we have a dataset with 'x' as the independent variable and 'y' as the dependent vari

# Creating sample data
set.seed(123)
x = 1:50
y = 2 * x + rnorm(50, mean = 0, sd = 5)  # Generating 'y' as a linear function of 'x' with so

# Creating a data frame
my_data = data.frame(x, y)

# Performing linear regression
model = lm(y ~ x, data = my_data)
summary(model)
```

Explanation of the code:

1. Sample data is generated with an independent variable `x` and a dependent variable `y`. In this example, `y` is generated as a linear function of `x` with some added noise using `rnorm()` to simulate real-world variability.
2. The data are combined into a data frame `my_data`.
3. The `lm()` function fits a linear regression model where `y` is the dependent variable and `x` is the independent variable (`y ~ x`). The argument `data = my_data` specifies the data frame containing the variables.

4. `summary(model)` provides a summary of the linear regression model, including coefficients, standard errors, t-values, p-values, R-squared, and other statistics.

Interpreting the output from `summary(model)`:

- The coefficients section shows the estimated coefficients for the intercept and the slope of the regression line (`Intercept` and `x`).
- The p-values associated with the coefficients indicate the significance of each variable in predicting the dependent variable. Lower p-values suggest stronger evidence against the null hypothesis of no effect.
- The R-squared value represents the proportion of variance in the dependent variable explained by the independent variable(s). Higher R-squared values indicate a better fit of the model to the data.

Linear regression in R can also 通过在模型中包含多个自变量，将其扩展到多元线性回归 (`y ~ x1 + x2 + ...`).

Additionally, diagnostic plots and further analyses can be performed to assess model assumptions and goodness of fit.

Model / Prediction / Coefficients In linear regression, the model equation is expressed as:

$$[y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + ... + \beta_n \cdot x_n + \epsilon]$$

Where:

- (y) is the dependent variable.
- ($x_1, x_2, ..., x_n$) are the independent variables.
- (β_0) is the intercept (constant term).
- ($\beta_1, \beta_2, ..., \beta_n$) are the coefficients (slope parameters) that represent the change in (y) associated with a one-unit change in the corresponding (x) variable, assuming all other variables remain constant.
- (ϵ) represents the error term.

In R, after fitting a linear regression model using `lm()`:

```r
# Assuming 'model' is the linear regression model obtained previously
summary(model)
```

The output from `summary(model)` provides information including:

- **Coefficients:** 本节显示每个自变量的估计系数（" 估计"），包括截距。这些系数代表因变量的估计变化，对应的自变量的一个单位的变化，保持其他变量不变。
- **Residuals:** 残差代表因变量的观测值和预测值之间的差异。这些用于评估模型的拟合优度。
- **R-squared:** 指示由自变量解释的因变量中的方差比例。较高的值指示模型对数据的拟合更好。

After obtaining the coefficients from the model, predictions can be made using new or existing data:

```
# Assuming 'new_data' contains the new data for prediction
predicted_values = predict(model, newdata = new_data)
```

Replace `new_data` with the data for which you want to make predictions. The `predict()` function uses the coefficients from the model to generate predicted values of the dependent variable based on the independent variables in the new data.

The coefficients obtained from the linear regression model (`model$coefficients`) represent the slopes of the regression line and the intercept, which are crucial for predicting new values and understanding the relationships between variables in the model.

Non-parametric Comparison

非参数方法是当参数方法所要求的正态性、方差齐性或线性假设被数据违反或不满足时所使用的统计技术。这些方法不依赖于特定的总体分布假设，对于分析可能不遵循正态分布的数据是有用的。

Here are some commonly used non-parametric methods for comparison in R:

1. **Mann-Whitney U Test (Wilcoxon Rank-Sum Test):**

   用于在 t 检验的假设不满足时比较两个独立的组。

   Example:

   ```
   # Assuming 'group1' and 'group2' are vectors of numeric data
   wilcox.test(group1, group2)
   ```

2. **Kruskal-Wallis Test:**

   Mann-Whitney U 检验的扩展，用于比较两个以上的独立组。

   Example:

   ```
   # Assuming 'group1', 'group2', 'group3' are vectors of numeric data
   kruskal.test(list(group1, group2, group3))
   ```

3. **Wilcoxon Signed-Rank Test:**

   用于比较配对样本或相关样本。

   Example:

   ```r
   # Assuming 'before' and 'after' are vectors of paired numeric data
   wilcox.test(before, after, paired = TRUE)
   ```

4. **Mood's Median Test:**

   测试两个或多个独立组中的中位数是否相等。

   Example:

   ```r
   # Assuming 'group1', 'group2', 'group3' are vectors of numeric data
   median_test = mood.test(group1, group2, group3)
   median_test
   ```

5. **Friedman Test:**

   Wilcoxon Signed-Rank 检验的扩展，用于比较两个以上配对或相关组。

   Example:

   ```r
   # Assuming 'group1', 'group2', 'group3' are matrices or data frames of paired numeric dat
   friedman.test(group1, group2, group3)
   ```

这些非参数检验提供了传统的参数检验的替代品，并对违反某些假设是强大的。它们在处理有序或倾斜数据或样本容量较小时特别有用，因为它们比参数检验依赖更少的分布假设。

Linear and nonlinear regression

Talk 11

This topic is particularly complex, so it is voluminous and obscure.

TOC

- linear regression
- nonlinear regression
- modeling and prediction
- **K-fold** & **X times** cross-validation
- external validation

Linear Regression

What is linear regression?

线性回归是一种利用数理统计中的回归分析来确定两个或多个变量之间相互依赖的数量关系的统计分析方法。

- `Y` can be explained by a variable `X`: One-way Linear Regression
- `Y` can be explained by multiple variables such as `X`, `Z`: Multiple Linear Regression

Linear regression in R is typically performed using the `lm()` function, which stands for "linear model." Here's how to fit a linear regression model in R and some useful functions related to linear regression:

Fitting a Linear Regression Model:

```r
# Example of fitting a linear regression model
# Assuming 'my_data' is a data frame with 'x' as the independent variable and 'y' as the deper
model = lm(y ~ x, data = my_data)
summary(model)  # Display summary statistics of the model
```

- **`lm()` Function:** Fits a linear regression model. The formula `y ~ x` specifies that `y` is the dependent variable and `x` is the independent variable. `data = my_data` indicates the data frame containing the variables.
- **`summary()` Function:** Displays a summary of the linear regression model, including coefficients, standard errors, t-values, p-values, R-squared, and other statistics.

glm vs. lm

```
lm(formula, data, ...)
glm(formula, family=gaussian, data, ...)
```

glm:

1. 当 `family=gaussian` 时，二者是一样的。

```
library(texreg);
m.lm <- lm(am ~ disp + hp, data=mtcars);
m.glm <- glm(am ~ disp + hp, data=mtcars);
screenreg(l = list(m.lm, m.glm))
```

glm 还可用于其它类型数据的分析

1. Logistic regression (family=binomial)

预测的结果（Y）是 binary 的分类，比如 `Yes`, `No`，且只能有两个值;

```
dat <- iris %>% filter( Species %in% c("setosa", "virginica") );
bm <- glm( Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
           data = dat, family = binomial );

data.frame( predicted = bm %>% predict( dat, type = "response" ),
            original = dat$Species ) %>% sample_n(6) %>% arrange( original );
```

注意:

`predict(., type = "response")` 指示 `predict()` 函数返回模型的响应值。

对于逻辑回归模型，这意味着返回的是概率而不是对数几率。

glm 的 Poisson regression (family=poisson)

泊松回归是一种特殊类型的回归，其中响应变量由**计数数据组成**.

Asumptions:

1. 响应变量由计数数据组成。
2. 观察是独立的。
3. 模型的均值和方差相等。
4. 计数的分布遵循泊松分布。

Other Useful Functions for Linear Regression Analysis:

1. **`coefficients()` Function:**

   检索线性回归模型的系数。

   Example:

   ```
   coef = coefficients(model)
   coef
   ```

2. **`predict()` Function:**

   使用拟合模型生成预测。

   Example:

   ```
   new_data = data.frame(x = c(10, 20, 30))  # New data for prediction
   predicted_values = predict(model, newdata = new_data)
   predicted_values
   ```

3. **`residuals()` Function:**

   检索残差（观测值和预测值之间的差异）。

   Example:

   ```
   residuals = residuals(model)
   residuals
   ```

4. **`fitted()` Function:**

   检索拟合（预测）值。

   Example:

   ```
   fitted_values = fitted(model)
   fitted_values
   ```

5. **`vcov()` Function:**

   计算系数的方差-协方差矩阵。

   Example:

   ```
   var_cov_matrix = vcov(model)
   var_cov_matrix
   ```

6. **`anova()` Function:**

   对拟合模型执行方差分析（ANOVA）。

   Example:

   ```
   anova_table = anova(model)
   anova_table
   ```

7. **`summary()` Function:**

   提供拟合模型的概述，包括参数估计值、标准误差、收敛信息和拟合优度统计量。

   Example:

   ```
   summary(model)
   ```

8. **`coef()` Function:**

   从拟合模型中提取估计系数。

   Example:

   ```
   coef(model)
   ```

9. **`confint()` Function (for prediction intervals):**

   计算拟合模型中预测值的预测区间。

   Example:

   ```
   prediction_intervals = predict(model, interval = "prediction")
   prediction_intervals
   ```

10. **`deviance()` Function:**

    计算拟合模型的偏差，这是缺乏拟合的度量。

    Example:

    ```
    deviance(model)
    ```

11. **`update()` Function:**

    允许使用不同的设置或数据更新或改装模型。

    Example:

    ```
    updated_model = update(model, start = list(a = 2, b = 2, c = 2))
    summary(updated_model)
    ```

12. **`AIC()` and `BIC()` Functions:**

    计算 Akaike 信息准则（阿灵顿资产投资）和贝叶斯信息准则（BIC）来评估模型质量和比较模型。

    Example:

    ```
    AIC(model)
    BIC(model)
    ```

13. **`plot()` Function (for diagnostic plots):**

    生成诊断图以评估模型的充分性（例如，残差与拟合值、Q-Q 图）。

    Example:

```
plot(model)
```

These functions help in obtaining and analyzing various aspects of the linear regression model, such as coefficients, predictions, residuals, variance-covariance matrix, and ANOVA tables, aiding in model interpretation, examining the fitted model's statistics, coefficients, goodness-of-fit measures, prediction intervals, model comparisons, and diagnostics, allowing for a comprehensive analysis of nonlinear regression models in R.

Nolinear Regression

当变量之间的关系不能用线性模型充分描述时，使用非线性回归。在 R 中，拟合非线性模型包括估计参数来描述变量之间的非线性关系。Here's an example using the `nls()` function, along with some relevant functions for nonlinear regression analysis:

Fitting a Nonlinear Regression Model:

1. `nls( equation, data = data, start = ... )`
2. `y~a*x/(b+x)`

```
# Example of fitting a nonlinear regression model (assuming a quadratic function)
# Assuming 'my_data' is a data frame with 'x' as the independent variable and 'y' as the depen

# Fitting a quadratic model: y = a * x^2 + b * x + c
model = nls(y ~ a * I(x^2) + b * x + c, data = my_data, start = list(a = 1, b = 1, c = 1))
summary(model)  # Display summary statistics of the model
```

- **`nls()` Function:** Fits a nonlinear regression model. The formula `y ~ a * I(x^2) + b * x + c` specifies a quadratic function. `data = my_data` indicates the data frame containing the variables, and `start = list(a = 1, b = 1, c = 1)` provides initial parameter values.
- **`summary()` Function:** Displays a summary of the nonlinear regression model, including parameter estimates, standard errors, t-values, convergence information, and other statistics.

Other Useful Functions for Nonlinear Regression Analysis:

1. **`predict()` Function:**

   Generates predictions using the fitted nonlinear model.

   Example:

```r
new_data = data.frame(x = c(10, 20, 30))  # New data for prediction
predicted_values = predict(model, newdata = new_data)
predicted_values
```

2. **`residuals()` Function:**

   Retrieves the residuals (differences between observed and predicted values).

   Example:

   ```r
   residuals = residuals(model)
   residuals
   ```

3. **`confint()` Function:**

   Computes confidence intervals for model parameters.

   Example:

   ```r
   conf_intervals = confint(model)
   conf_intervals
   ```

4. **`nls.control()` Function:**

   为 `nls()` 函数提供控制参数，允许对非线性拟合过程进行调整。

   Example:

   ```r
   control_params = nls.control(maxiter = 100, tol = 1e-6)
   model = nls(y ~ a * I(x^2) + b * x + c, data = my_data, start = list(a = 1, b = 1, c = 1)
   ```

5. **`anova()` Function:**

   Performs analysis of variance (ANOVA) for the fitted nonlinear model.

   Example:

   ```r
   anova_table = anova(model)
   anova_table
   ```

6. **`nlsLM()` Function (from the `minpack.lm` package):**

   - An alternative to `nls()` that provides enhanced convergence properties and extended functionality for nonlinear least squares.

   Example:

   ```r
   library(minpack.lm)
   model_nlsLM = nlsLM(y ~ a * I(x^2) + b * x + c, data = my_data, start = list(a = 1, b = 1
   summary(model_nlsLM)
   ```

7. **`augment()` Function (from the `broom` package):**

   - 使用其他列（如拟合/预测值、残差和其他模型信息）创建整洁的数据框架。

Example:

```
library(broom)
augmented_model = augment(model)
head(augmented_model)
```

8. **`glance()` Function (from the `broom` package):**
   - 提取模型级统计信息，以简洁的格式提供模型的摘要。

   Example:

   ```
   glance_summary = glance(model)
   glance_summary
   ```

9. **`tidy()` Function (from the `broom` package):**
   - 将模型系数和相关统计数据提取到一个整洁的数据框架中。

   Example:

   ```
   tidy_summary = tidy(model)
   tidy_summary
   ```

10. **`nlstools::nlstools()` Function:**
    - 为非线性回归模型提供诊断工具和可视化，帮助模型评估。

    Example:

    ```
    library(nlstools)
    model_tools = nlstools(model)
    plot(model_tools)
    ```

11. **`nls2()` Function:**
    - 提供具有多个起始值的'nls（）'的扩展版本，以改进收敛性。

    Example:

    ```
    model_nls2 = nls2(y ~ a * I(x^2) + b * x + c, data = my_data, start = list(a = 1, b = 1,
    summary(model_nls2)
    ```

These functions are commonly used for nonlinear regression analysis in R, helping in prediction, residual analysis, confidence interval computation, and controlling the fitting process.

Modeling and Prediction

When it comes to modeling and prediction using regression analysis, especially nonlinear regression, understanding the model, making predictions, and assessing the model's performance are crucial. Here's a step-by-step guide on how to approach modeling and prediction:

Modeling and Prediction Steps:

1. **Data Preparation:**

Prepare your dataset with variables for the dependent (response) and independent (predictor) variables.

1. **Fitting the Nonlinear Model:**

使用 `nls()` 或类似函数拟合非线性回归模型，指定适当的公式和初始参数值。

Example:

```
model =
nls(y ~ a * I(x^2) + b * x + c,
    data = my_data,
    start = list(a = 1, b = 1, c = 1))
```

1. **Model Summary and Assessment:**

Use `summary()` and other functions (`glance()`, `tidy()`, etc.) 以获得模型的概述，包括系数、拟合优度度量和诊断。

Example:

```
summary(model)
```

1. **Prediction with the Fitted Model:**

使用拟合模型为新数据或现有数据生成预测。

Example:

```
new_data = data.frame(x = c(10, 20, 30))  # New data for prediction
predicted_values = predict(model, newdata = new_data)
predicted_values
```

1. **Model Evaluation:**

Evaluate the model's performance using various metrics (e.g., residuals, R-squared, RMSE) and diagnostic plots (e.g., residuals vs. fitted values, Q-Q plots) to assess how well the model fits the data.

1. **Adjustment and Refinement:**

Depending on the evaluation results, consider refining the model by adjusting parameters, exploring different models, or including/excluding variables to improve performance.

1. **Prediction Intervals and Uncertainty:**

Compute prediction intervals using `predict()` with `interval = "prediction"` to quantify the uncertainty around predictions.

1. **Model Comparison (if applicable):**

Compare multiple models using metrics like AIC, BIC, or likelihood ratio tests to select the most appropriate model.

By following these steps, you'll be able to build, evaluate, and utilize nonlinear regression models for prediction in R effectively. Remember, interpreting the model's results, assessing its assumptions, and validating predictions are crucial aspects of regression analysis.

K-fold & X times cross-validation

K-fold 交叉验证和 X 次交叉验证都是用于评估机器学习模型（包括回归模型）的性能和泛化能力的技术，方法是将数据划分为子集进行训练和验证。

K-fold Cross-Validation:

k-fold 交叉验证涉及将数据集拆分成 K 个大小相等的折叠。对模型进行 K 次训练，每次使用 K-1 折叠进行训练，其余折叠进行验证。这个过程确保每个数据点都用于训练和验证。

Steps for K-fold Cross-Validation:

1. **Partition Data:** 将数据集分割成 K 个大小相等的折叠。
2. **Model Training:** 对模型进行 K 次训练，每次使用 K-1 折叠作为训练数据。
3. **Validation:** 验证模型在其余部分（训练中未使用）上的性能，并计算评估指标。.
4. **Average Metrics:** 对 K 个迭代中的评估指标求平均值，以获得对模型性能的总体评估。

X times Cross-Validation:

X 次交叉验证，也被称为重复的 K-fold 交叉验证，类似于 K-fold 的交叉验证，但这个过程是重复 X 次。它重复创建随机分区的数据到 K 折叠，训练模型，并评估其性能。该方法通过对多次试验的结果求平均值来提供更稳健的模型性能估计。

Steps for X times Cross-Validation:

1. **Partition Data Repeatedly:** 将数据集随机分成 K 个折叠，X 次。
2. **Model Training:** 为每个迭代训练模型，使用 K-1 折叠进行训练。
3. **Validation:** 验证模型在剩余折叠上的性能并计算评估指标。
4. **Average Metrics:** 对 X 次迭代中的评估指标求平均值，以获得更稳定、更可靠的模型性能估计值。

Implementation in R:

In R, you can perform K-fold and X times cross-validation using functions from various packages like `caret`, `rsample`, or `crossval`. For example, using `caret`:

K-fold Cross-Validation in R with `caret`:

```
library(caret)
# Define a train control using k-fold cross-validation
train_control =
        trainControl(method = "cv", number = K)
# Specify K for the number of folds

# Train the model using k-fold cross-validation
model =
        train(
    formula,
    data = my_data,
    method = "lm",
    trControl = train_control
  )
```

X times Cross-Validation in R with `caret`:

```
library(caret)
# Define a train control using repeated k-fold cross-validation
train_control =
trainControl(
  method = "repeatedcv",
  number = K, repeats = X)
# Specify K for folds and X for repeats

# Train the model using repeated k-fold cross-validation
```

```
model =
        train(
    formula,
    data = my_data,
    method = "lm",
    trControl = train_control
  )
```

Replace `formula` and `my_data` with the appropriate regression formula and your dataset. Adjust the parameters `K` and `X` according to your preferences for the number of folds and repetitions.

这些交叉验证技术有助于估计模型的性能，并帮助评估模型如何推广到看不见的数据，从而提供洞察其鲁棒性和可靠性。调整这些技术可以提高回归模型的准确性和稳定性的评价.

External Validation

外部验证是指使用在模型开发过程中未使用的独立数据集来评估预测模型的性能。它是评估一个模型如何从不同的来源或时间段推广到新的、看不见的数据的关键步骤，以验证它在实际应用中的可靠性和健壮性。

Steps for External Validation:

1. **Obtain an Independent Dataset:** 获取与用于模型训练和验证的数据集不同的单独数据集。理想情况下，这个数据集应该代表相同的问题或领域，但来自不同的源、时间框架或人群。
2. **Preprocess Data:** 对独立数据集进行类似于训练数据集的预处理（例如，处理缺失值、编码分类变量、缩放特征），以确保兼容性。
3. **Apply Trained Model:** 使用在原始数据集上训练的模型对独立数据集进行预测。
4. **Evaluate Model Performance:** 使用相关评估指标（例如准确度、RMSE、精度、召回率）评估模型在独立数据集上的性能，并将这些指标与在训练/验证数据集上实现的性能进行比较。
5. **Analyze Results:** 分析从独立数据集获得的性能指标，以确定模型是否保持其预测能力和泛化能力。性能良好的独立数据集上的模型表明，良好的泛化能力和可靠性。

Implementation in R:

In R, the process involves loading the trained model and applying it to the independent dataset for prediction. Use appropriate evaluation functions (`predict()`, evaluation metrics) to assess the model's performance on the external dataset.

Example in R:

```r
# Load the trained model (replace 'model' with your trained model)
load("trained_model.RData")

# Load and preprocess the independent dataset
# (replace 'independent_data.csv' with your dataset)
independent_data = read.csv("independent_data.csv")
# Perform similar preprocessing steps as used for the training data

# Apply the trained model to the independent dataset for prediction
predicted_values = predict(model, newdata = independent_data)

# Evaluate model performance on the independent dataset
# Use appropriate evaluation metrics
# (e.g., RMSE, accuracy)
# and compare with training/validation results
```

Replace the file paths, data loading, and evaluation steps with your specific dataset and evaluation procedures. Ensure the compatibility of the independent dataset with the preprocessing steps applied to the original dataset for accurate evaluation.

Machine learning basics

Talk 12

This topic is particularly complex, so it is voluminous and obscure.

TOC

机器学习可分为以下几类

\FontSmall

1）**回归算法**

2）基于实例的算法

3）**决策树学习**

4）贝叶斯方法

5）基于核的算法

6）**聚类算法**

7）**降低维度算法**

8）关联规则学习

9）集成算法

10）人工神经网络

1. 回归算法

回归算法是试图采用对误差的衡量来探索变量之间的关系的一类算法。常见的回归算法包括：

- 最小二乘法（Ordinary Least Square），
- 逻辑回归（Logistic Regression），
- 逐步式回归（Stepwise Regression），
- 多元自适应回归样条（Multivariate Adaptive Regression Splines）以及
- 本地散点平滑估计（Locally Estimated Scatterplot Smoothing）。

2. 基于实例的算法

基于实例的算法常常用来对决策问题建立模型，这样的模型常常先选取一批样本数据，然后根据某些近似性把新数据与样本数据进行比较。通过这种方式来寻找最佳的匹配。

因此，基于实例的算法常常也被称为“赢家通吃”学习或者“基于记忆的学习”。常见的算法包括：

- k-Nearest Neighbor(KNN)，
- 学习矢量量化（Learning Vector Quantization，LVQ），以及
- 自组织映射算法（Self-Organizing Map，SOM）。

深度学习的概念源于人工神经网络的研究。含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更表示属性类别或特征，以发现数据的分布式特征表示。

3. 决策树学习

决策树算法根据数据的属性采用树状结构建立决策模型，决策树模型常常用来解决分类和回归问题。常见的算法包括：

- 分类及回归树（Classification And Regression Tree，CART），
- ID3 (Iterative Dichotomiser 3)，
- C4.5，Chi-squared Automatic Interaction Detection(CHAID)，
- Decision Stump, 随机森林（Random Forest），
- 多元自适应回归样条（MARS）以及
- 梯度推进机（Gradient Boosting Machine，GBM）。

4. 贝叶斯方法

贝叶斯方法算法是基于贝叶斯定理的一类算法，主要用来解决分类和回归问题。常见算法包括：

- 朴素贝叶斯算法，
- 平均单依赖估计（Averaged One-Dependence Estimators，AODE），以及
- Bayesian Belief Network（BBN）。

5. 基于核的算法

基于核的算法中最著名的莫过于支持向量机（SVM）了。基于核的算法把输入数据映射到一个高阶的向量空间，在这些高阶向量空间里，有些分类或者回归问题能够更容易的解决。常见的基于核的算法包括：

- 支持向量机（Support Vector Machine，SVM），
- 径向基函数（Radial Basis Function，RBF），以及
- 线性判别分析（Linear Discriminate Analysis，LDA）等。

6. 聚类算法

聚类，就像回归一样，有时候人们描述的是一类问题，有时候描述的是一类算法。聚类算法通常按照中心点或者分层的方式对输入数据进行归并。所以的聚类算法都试图找到数据的内在结构，以便按照最大的共同点将数据进行归类。常见的聚类算法包括

- k-Means 算法以及
- 期望最大化算法（Expectation Maximization，EM）。

7. 降低维度算法

像聚类算法一样，降低维度算法试图分析数据的内在结构，不过降低维度算法是以非监督学习的方式试图利用较少的信息来归纳或者解释数据。这类算法可以用于高维数据的可视化或者用来简化数据以便监督式学习使用。常见的算法包括：

- 主成份分析（Principle Component Analysis，PCA），
- 偏最小二乘回归（Partial Least Square Regression，PLS），
- Sammon 映射，
- 多维尺度（Multi-Dimensional Scaling, MDS），
- 投影追踪（Projection Pursuit）等。

8. 关联规则学习

关联规则学习通过寻找最能够解释数据变量之间关系的规则，来找出大多元数据集中有用联规则。常见算法包括

- Apriori 算法和
- Ec t 算法等。

9. 集成算法

集成算法用一些相对较弱的学习模型独立地就同样的样本进行训练，然后把结果整合起来进行整体预测。集成算法的主要难点在于究竟集成哪些独立的较弱的学习模型以及如何把学习结果整合起来。这是一类非常强大的算法，同时也非常流行。常见的算法包括：

- Boosting，
- Bootstrapped Aggregation（Bagging），
- AdaBoost，
- 堆叠泛化（Stacked Generalization，Blending），
- 梯度推进机（Gradient Boosting Machine, GBM），
- 随机森林（Random Forest）

10. 人工神经网络

人工神经网络算法模拟生物神经网络，是一类模式匹配算法。通常用于解决分类和回归问题。人工神经网络是机器学习的一个庞大的分支，有几百种不同的算法。（其中**深度学习**就是其中的一类算法），重要的人工神经网络算法包括：

- 感知器神经网络（Perceptron Neural Network），
- 反向传递（Back Propagation），
- Hopfield 网络，
- 自组织映射（Self-Organizing Map, SOM）。
- 学习矢量量化（Learning Vector Quantization，LVQ）。

section 3: 随机森林

随机森林 – Random forest

本文大部分内容取自 “easyai.tech” 网站的《随机森林 – Random forest》一文，有修改。

随机森林是一种由**决策树**构成的集成算法，适用于**小样本**数据，在很多情况下都能有不错的表现。

随机森林属于 **集成学习**中的 Bagging（Bootstrap AGgregation 的简称）方法。

决策树 - decision tree

决策树是一种很简单的算法，他的解释性强，也符合人类的直观思维。这是一种基于 if-then-else 规则的有监督学习算法，

Steps to Implement Random Forest in R:

1. Load Required Library: Start by loading the necessary library (`randomForest`) if not already installed.

```
install.packages("randomForest")  # Install package if not installed
library(randomForest)
```

2. Prepare Data: 将数据集加载到 R 中并执行必要的预处理步骤，如处理缺失值、对分类变量进行编码以及将数据拆分为训练集和测试集。

```
# Example: Assuming 'my_data' is your dataset
# Split data into features (X) and target variable (Y)
X = my_data[, -target_column_index]  # Features
Y = my_data[, target_column_index]   # Target variable
```

3. Train the Random Forest Model: Use the `randomForest()` function to 通过指定公式和训练数据来训练模型。

```
# Example: Training a Random Forest model for regression
model = randomForest(Y ~ ., data = my_data)
# For classification: model = randomForest(factor(Y) ~ ., data = my_data)
```

4. Model Tuning (Optional): Adjust hyperparameters such as the number of trees (`ntree`), maximum depth (`max_depth`), and others to optimize model performance.

```
# Example: Setting number of trees and maximum depth
model = randomForest(Y ~ ., data = my_data, ntree = 100, max_depth = 10)
```

5. Make Predictions: Use the trained model to make predictions on new or test data.

```
# Example: Making predictions on test data
predicted_values = predict(model, newdata = test_data)
```

6. Model Evaluation: Evaluate the model's performance using appropriate metrics (e.g., RMSE for regression, accuracy, confusion matrix for classification) on test/validation data.

```
# Example: Evaluate model performance (for regression)
error = sqrt(mean((predicted_values - true_values)^2))  # Calculate RMSE
```

Example - Random Forest for Regression:

Here's an example using a built-in dataset (`mtcars`) in R for regression:

```
library(randomForest)

# Load dataset
data(mtcars)

# Split data into features and target variable
X = mtcars[, -1]  # Exclude the first column (target variable)
Y = mtcars[, 1]   # First column (target variable)

# Train the Random Forest model
model = randomForest(Y ~ ., data = mtcars)

# Make predictions on the same dataset (for demonstration)
predicted_values = predict(model, newdata = mtcars)

# Evaluate model performance (RMSE)
error = sqrt(mean((predicted_values - mtcars$mpg)^2))
```

Replace `my_data`, `test_data`, and the respective column names with your specific dataset and target variable. Adjust hyperparameters based on your problem and dataset characteristics for better model performance. Additionally, use appropriate evaluation metrics for assessing model accuracy and performance.

Feature Selection

特征选择是机器学习中的一个关键步骤，旨在选择最相关、最具信息量的特征，以提高模型性能、降低计算复杂度并减轻过拟合。在 R 中，各种方法可以帮助识别重要的特征。

Feature Selection Techniques in R:

1. Correlation Analysis: 计算特征与目标变量之间或特征本身之间的相关系数，以识别高度相关的特征。删除冗余或高度相关的功能。

```
# Example: Using cor() for correlation analysis
correlation_matrix = cor(my_data)
```

2. Filter Methods: 使用统计方法（例如，卡方检验，互信息），以排名和选择的基础上，他们的个人相关的目标变量的功能。

```r
# Example: Using chi-squared test for feature selection
library(caret)
feature_selection = nearZeroVar(my_data)
```

3. Wrapper Methods: 通过迭代训练模型并选择可实现最佳模型性能的子集来评估特征子集。

```r
# Example: Using recursive feature elimination (RFE)
library(caret)
control = rfeControl(functions = rfFuncs, method = "cv", number = 10)
feature_selection = rfe(X, Y, sizes = c(1:10), rfeControl = control)
```

4. Embedded Methods: 模型训练期间的特征选择，其中算法本身执行特征选择（例如，LASSO、随机森林、梯度提升）。

```r
# Example: Using Random Forest for feature selection
library(randomForest)
model = randomForest(Y ~ ., data = my_data)
importance = importance(model)
```

5. Dimensionality Reduction Techniques: 主成分分析（PCA）或 t-Distributed 随机近邻嵌入（t-SNE）等技术通过创建捕获大部分原始信息的新变量来缩小特征空间。

```r
# Example: PCA for feature reduction
pca_result = prcomp(my_data, scale. = TRUE)
```

根据数据集的特征、问题复杂性和计算资源选择特征选择方法。多种技术的组合通常会产生最好的结果，因为不同的方法可以捕获特征重要性的不同方面。进行试验和迭代，为您的机器学习模型找到最适合的功能。

This is the end of the article. Take a rest~