apply() 主要用于数组的边际。

MARGIN：指定应用函数的维度（1表示行，2表示列）

lapply() 总是返回一个列表。

对列表、向量或数据框的每一个元素应用函数，并返回一个列表。

sapply() 尝试简化结果，但可能仍然返回一个列表（如果简化不可能）。

vapply() 允许用户指定返回值的类型和结构，这有助于避免类型不一致的问题。

mapply() 允许对多个列表或向量的对应元素应用函数。

summary() vs. str()

1.

summary()函数主要用于描述数据集的统计摘要。当你对一个数据框（data frame）、向量（vector）、矩阵（matrix）或其他一些对象（如模型结果）使用summary()函数时，它会返回关于该对象的统计信息。

对于数值型数据（如向量或数据框中的列），summary()通常返回最小值、第一四分位数、中位数、均值、第三四分位数和最大值。

对于因子（factor）类型的数据，它会返回每个级别的频数。

对于模型结果（如lm()的输出），summary()会提供模型的详细统计信息，如R平方、F统计量、p值等。

2.

str()函数主要用于查看对象的结构。当你对一个R对象使用str()函数时，它会返回一个简短的描述，说明该对象是什么类型，以及它包含哪些组成部分。

对于数据框（data frame），str()会列出每一列的名称、类型以及可能的附加信息（如因子级别的数量）。

对于列表（list），str()会递归地列出列表中每个元素的类型。

对于向量（vector），str()会简单地指出它是哪个类型的向量（如字符型、数值型、逻辑型等）。

gather() vs. spread()

1.

gather() 函数用于将数据从宽格式转换为长格式。宽格式的数据通常具有多个列表示不同的变量或观察值，而长格式的数据则只有一列包含这些变量或观察值，并有一列来标识这些变量或观察值的来源。

参数：

data: 需要转换的数据框（data frame）。

key: 新列的名称，用于存储变量或观察值的标识符。

value: 新列的名称，用于存储原始数据框中各个列的值。

...: 需要转换的列名。如果不指定，则默认转换除 key 和 value 以外的所有列。

其他参数（如 na.rm、convert 等）用于控制转换过程。

假设我们有一个宽格式的数据框，其中包含了不同年份的销售数据：

|  |  |
| --- | --- |
|  | library(tidyr) |
|  |  |
|  | wide\_df <- data.frame( |
|  | ID = c(1, 2), |
|  | Sales\_2019 = c(100, 200), |
|  | Sales\_2020 = c(150, 250) |
|  | ) |
|  |  |
|  | long\_df <- gather(wide\_df, Year, Sales, -ID) |
|  | print(long\_df) |

输出：

复制代码

|  |  |
| --- | --- |
|  | ID Year Sales |
|  | 1 1 2019 100 |
|  | 2 2 2019 200 |
|  | 3 1 2020 150 |
|  | 4 2 2020 250 |

2.

spread() 函数与 gather() 函数相反，它用于将数据从长格式转换为宽格式。长格式的数据通常只有一列包含变量或观察值，并有一列来标识这些变量或观察值的来源，而宽格式的数据则具有多个列表示不同的变量或观察值。

参数：

data: 需要转换的数据框（data frame）。

key: 包含变量或观察值标识符的列名。

value: 包含原始数据框中各个列值的列名。

into: 新列的名称（或列名向量），用于存储从 key 列中获得的变量或观察值的数据。

其他参数（如 sep、convert 等）用于控制转换过程。

继续使用上面的长格式数据框：

|  |  |
| --- | --- |
|  | wide\_df\_again <- spread(long\_df, Year, Sales) |
|  | print(wide\_df\_again) |

输出：

|  |  |
| --- | --- |
|  | ID Sales\_2019 Sales\_2020 |
|  | 1 1 100 150 |
|  | 2 2 200 250 |

gsub()

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE)

pattern：需要查找和替换的正则表达式模式，也可以是一个简单的字符串。

replacement：用于替换匹配模式的字符串。它也可以是一个函数，在这种情况下，每个匹配的模式都会作为输入传递给这个函数，并使用返回的结果进行替换。

x：需要进行模式匹配和替换的字符串或字符向量。

ignore.case：一个逻辑值，指定是否忽略大小写进行匹配，默认为FALSE。

perl：一个逻辑值，指定是否使用Perl风格的正则表达式，默认为FALSE。

fixed：一个逻辑值，指定是否将pattern看作固定的字符序列（即不使用正则表达式），默认为FALSE

strings <- c("aple", "banana", "aple pie")

corrected\_strings <- gsub("aple", "apple", strings)

print(corrected\_strings)

# string manipulation

library (stringr)

1.

substr() or str\_sub()

str <- "Hello, World!"

# 使用substr()提取从第8个字符开始的3个字符

sub\_str <- substr(str, 8, 10)

2.

strsplit() or str\_split()

str <- "apple,banana,cherry"

# 使用strsplit()按逗号拆分字符串

split\_str <- strsplit(str, ",")

print(unlist(split\_str))

请注意，strsplit()返回的是一个列表的列表（在只有一个输入字符串的情况下），因此我使用unlist()来扁平化结果。如果你在使用stringr包，你可能会使用str\_split()函数，但通常它返回的是一个列表，不需要额外的unlist()步骤。

import files

read.table("Tests\_PGP3.txt", sep='\t', header = T)

read.csv(file="FHY is rank1")

knitr

1.

eval控制了代码块是否执行。若设定为eval=TRUE，则markdown会执行代码块里的代码；若eval=FALSE，则markdown不会执行代码。

2.

echo参数控制了markdown是否显示代码块。若echo=TRUE，则表示代码块显示在markdown文档显示代码块；反之，代码块不出现在输出结果中。

What is P value?

Probability of observing a value as or more extreme as the one you observed if the null hypothesis were true.

头文件

library(tidyverse) # 据说只要加这一条剩下来都包括了

library(ggplot2)

library(dplyr)

library(tidyr)

library(knitr)

library(tinytex)

library(rmarkdown)

library(cowplot)

library(ggpubr)

想把一个data.frame的按照其中某一个column的值对这个数据框的每一行重新排序，应该怎么写

在R语言中，如果你想根据数据框（data.frame）中的某一列的值对整个数据框的行进行重新排序，你可以使用order()函数配合data.frame的行索引来实现。以下是一个简单的例子：

假设你有一个名为df的数据框，并且你想根据名为column\_name的列的值进行排序：

|  |  |
| --- | --- |
|  | # 创建一个示例数据框 |
|  | df <- data.frame( |
|  | column\_name = c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3), |
|  | other\_column = letters[1:10] |
|  | ) |
|  |  |
|  | # 使用order()函数获取排序后的行索引 |
|  | sorted\_indices <- order(df$column\_name) |
|  |  |
|  | # 使用这些索引重新排序数据框 |
|  | df\_sorted <- df[sorted\_indices, ] |
|  |  |
|  | # 查看排序后的数据框 |
|  | print(df\_sorted) |

在这个例子中，order(df$column\_name)会返回一个向量，其中包含了根据column\_name列的值对数据框行进行排序后的索引。然后，你可以使用这些索引来重新排序原始数据框。

注意，order()函数默认是进行升序排序的。如果你想进行降序排序，可以传递一个递减的向量给order()函数，如order(-df$column\_name)。

# 假设我们想要factor的levels按照 "e", "d", "c", "b", "a" 的顺序

df$my\_factor <- factor(df$my\_factor, levels = c("e", "d", "c", "b", "a"))

# 直接根据factor的levels进行排序

sorted\_indices <- order(df$my\_factor)

# 使用这些索引重新排序数据框

df\_sorted <- df[sorted\_indices, ]

# 查看排序后的数据框

print(df\_sorted)