# R For Data Science *Cheat Sheet*

# Data Processing

EasyCharts团队-张杰出品

## 数据的基本类型

①数值型(numeric): a<-1, is.numeric(a)
②字符型(character): b<- "peter"; nchar(b) #字符串的长度
③日期型(Date): 最常用的是Date（仅储存日期）和POSIXct (储存日期与时间)

```
c<-as.Date("2012-06-12"); class(c)
d<-as.POSIXct("2012-06-12 17:32"); class(d)
```

④逻辑型(logical): e<-TRUE, f<-FALSE

## 常见的数据结构

常见的数据结构包括：向量(vector),数据框(data.frame)，矩阵(matrix)，列表（list）和数值（array）

## 向量(vector)

1. 向量(vector)的创建（# c代表合并（combine））

| 输入 | 输出 | 描述 |
|---|---|---|
| c(2,4,6) | 2 4 6 | 将元素连接成向量 |
| 2:6 | 2 3 4 5 6 | 等差整数数列 |
| seq(2, 3, by=0.5) | 2.0 2.5 3.0 | 步长为0.5的等差数列 |
| rep(1:2, times=3) | 1 2 1 2 1 2 | 将一个向量重复3次 |
| rep(1:2, each=3) | 1 1 1 2 2 2 | 将一个向量中的每个元素重复3次 |
| rnorm(3, mean = 0, sd = 3) | -2.09 -3.52 -4.25 | 均值为0，标准差为3的正态分布 |
| runif(3, min = 0, max = 1) | 0.63 0.05 0.61 | 最大值为1，最小值为0 的均匀分布 |
| sample(c("A","B","C"), 4, replace=TRUE) | "A" "A" "A" "B" | 从一个向量中随机抽取 |

2. 向量(vector)的处理

- 向量的排序

```
Vec<-c(1,4,3,5,2)
order<-sort(Vec, index.return=TRUE,decreasing = TRUE)
```

- 向量的逆序

```
Vec<-c(1,4,3,5,2)
Rev<-rev(Vec)
```

- 向量的唯一值

```
Vec<-c(4,4,5,5,2)
Uni<-unique(Vec)
```

- 连续向量的离散化

```
Num_Vector<- ceiling(runif(10,1,10))
Cut_Vector<-cut(Num_Vector,breaks=c(0,3,6,9,11),
                labels=c("0~3", "3~6", "6~9", ">9"), right = TRUE)
```

(3) 因子向量(factor vector)

- 普通向量默认的Levels根据字母顺序排序：(Fair,Good，Ideal, Premium, Very Good)

```
Cut<-c("Fair","Good","Very Good","Premium","Ideal")
Cut_Facor1<-as.factor(Cut)
```

- 更改因子向量的levels为（"Good","Fair","Very Good","Ideal","Premium"）

```
Cut_Facor2<-factor(x=c("Fair","Good","Very Good","Premium","Ideal"),
                   levels= c("Good","Fair","Very Good","Ideal","Premium"),
                   ordered=TRUE)
```

- 数值型因子向量的类型变换

```
Num _Facor<-factor(x=c(1,3,5,2), levels= c(5,3,2,1), ordered=TRUE)
Num_Vector1<-as.numeric(as.character(Num_Facor))  # 输出：1 3 5 2
Num_Vector2<-as.numeric(Num_Facor)  # 输出：4 2 1 3
```

(4)向量的选择

```
x<-c(1,4,3,5,2)
```

- 选择某个或多个元素：x[2]; x[-2]; x[2:4]; x[c(1,4)]
- 逻辑运算选择元素：x[x>2]; x[x==1]; x[x<=5]

## 数据框(data.frame)

(1) 数据框的基本信息

df<-data.frame(x= c("a","b","c"), y=1:3,z=c(2,5,3))

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | 3 |

- 数据框数据的选取

选取某一列：df[,2]，df$y，df[[2]]

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | 3 |

选取多列： df[c("x","y")]，df[,1:2]

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | 3 |

选取某一行：df[2,]

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | 3 |

- 获取数据框的行数、列数和维数：nrow()， ncol()， dim()

选取多行：df[1:2,]

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | 3 |

- 获取数据框的列名或行名：names()， rownames()， colnames()

重新定义列名names(df)<-c("X", "Y", "Z")

选取某个元素：df[2,2]

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | 3 |

- 观察数据框的内容：view(df)，head(df, n=3)，tail(df)

(2) 空数据框的创建：

- 创建一个名为Df_Empty，包括两个变量（var_a为numeric类型；var_b为character类型）的data.frame。但是注意：要加上stringsAsFactors=FALSE,否则在后面逐行输入数据的时候，会因为var_b的取值未经定义的factor level而报错。

```
Df_Empty1<- data.frame(var_a = numeric(),
                       var_b = character(),
                       stringsAsFactors=FALSE)
```

- 使用矩阵matrix创建空的数据框

```
Df_Empty2 <- data.frame(matrix(ncol = 2, nrow = 0))
colnames(Df_Empty2)<- c("var_a","var_b")
```

## 数据的导入导出

(1) csv格式数据的导入与导出

- 使用read.csv()函数，可以读入数据，并为data.frame格式存储

```
mydata<-read.csv("Data.csv",sep=",",na.strings="NA",
                 stringsAsFactors=FALSE)
```

- 使用write.csv()函数, 可以将data.frame的数据存储为csv文件

```
write.csv(mydata,file = "File.csv")
```

(2) txt格式数据的导入与导出

- 使用read.table()函数可以读入数据，并为data.frame格式存储

```
mydata<-read.table("Data.txt",header = TRUE)
```

- 使用write.table()函数可以将data.frame的数据存储为csv文件

```
write.table(mydata, file = "File.txt")
```

(3) excel格式数据的导入

- 使用readxl包的read_excel()函数可以读入数据

```
mydata<- readxl::read_excel("Data.xlsx", sheet = "Sheet Name")
```

- 缺失数据使用NA表示，is.na()函数判定
- 空白数据使用NULL表示，is.null()函数判定

## 缺失值的处理

数据的构建：df<-data.frame(x1=letters[1:4],x2=c(1,NA,NA,3))

| x | y |
|---|---|
| a | 1 |
| b | NA |
| c | NA |
| d | 3 |

→

| x | y |
|---|---|
| a | 1 |
| d | 3 |

直接删除带NA的行：
tidyr::drop_na(df,y)

| x | y |
|---|---|
| a | 1 |
| b | NA |
| c | NA |
| d | 3 |

→

| x | y |
|---|---|
| a | 1 |
| b | 1 |
| c | 3 |
| d | 3 |

使用最邻近的元素填充NA：
tidyr::fill(df,y)

| x | y |
|---|---|
| a | 1 |
| b | NA |
| c | NA |
| d | 3 |

→

| x | y |
|---|---|
| a | 1 |
| b | 2 |
| c | 2 |
| d | 3 |

使用指定的数值替代NA:
tidyr::fill(df,list(y=2))

## 控制流语句与函数

### (1) for循环语句

```
for (variable in sequence){
    Do something
}
```

```
for (i in 1:4){
    j <- i + 10
    print(j)
}
```

### (2) while循环语句

```
while (condition){
    Do something
}
```

```
while (i < 5){
    print(i)
    i <- i + 1
}
```

### (3) If条件语句

```
if (condition){
  Do something
} else {
  Do something different  }
```

```
if (i > 3){
    print('Yes')
} else {
    print('No')}
```

### (4)自定义函数function

```
function_name <- function(var){
    Do something
    return(new_variable)
}
```

```
square <- function(x){
    squared <- x*x
    return(squared)
}
```

## 表格的连接与融合

df1<-data.frame(x= c("a","b","c"), y=1:3)
df2<- data.frac(z= c("B","D","H"), g =c(2,5,3))
df3<- data.frame(x= c("g","d"), y =c(2,5))

df1

| x | y |
|---|---|
| a | 1 |
| b | 2 |
| c | 3 |

df2

| z | g |
|---|---|
| B | 2 |
| D | 5 |
| H | 3 |

df3

| x | y |
|---|---|
| g | 2 |
| d | 5 |

(1) 数据框添加列或者横向添加表格：
dat_cbind<-cbind(df1,df2)

| x | y | z | g |
|---|---|---|---|
| a | 1 | B | 2 |
| b | 2 | D | 5 |
| c | 3 | H | 3 |

(2) 数据框添加行或者纵向添加表格：
dat_rbind<-rbind(df1,df3)

| x | y |
|---|---|
| a | 1 |
| b | 2 |
| c | 3 |
| g | 2 |
| d | 5 |

(3) 两个表格的融合
df1<-data.frame(x= c("a","b","c"), y=1:3)
df2<- data.frame(x= c("a","b","d"), z =c(2,5,3))
df3<- data.frame(g= c("a","b","d"), z =c(2,5,3))
df4<- data.frame(x= c("a","b","d"), y=c(1,4,2),z =c(2,5,3))

df1

| x | y |
|---|---|
| a | 1 |
| b | 2 |
| c | 3 |

df2

| x | z |
|---|---|
| a | 2 |
| b | 5 |
| d | 3 |

df3

| g | z |
|---|---|
| a | 2 |
| b | 5 |
| d | 3 |

df4

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 4 | 5 |
| d | 2 | 3 |

- merge()函数：优势在于对于每个数据框可以指定不同的匹配列名；缺点在于运行速度比较慢
- dat_merge1 <-merge(df1,df2,by="x", all = TRUE)
- dat_merge2 <-merge(df1,df3,by.x="x",by.y="g")
- dat_merge3 <-merge(df1,df4,by=c("x","y"), all = TRUE)

dat_merge1

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | NA |
| d | NA | 3 |

dat_merge2

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |

dat_merge3

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | NA |
| b | 4 | 5 |
| c | 3 | NA |
| d | 2 | 3 |

- dplyr包提供了left_join(), right_join(), inner_join()和full_ join()四个函数
- 只保留左表的所有数据：
  dat_join1 <-dplyr::left_join (x=df1,y=df2,by="x")
- 只保留右表的所有数据：
  dat_join2 <-dplyr::right_join (x=df1,y=df2,by= "x")
- 只保留两个表中公共部分的信息：
  dat_join3 <-dplyr::inner_join (x=df1,y=df2,by="x")
- 保留两个表的所有信息：
  dat_join4 <-dplyr:: full_join(x=df1,y=df2,by="x")

dat_join1

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | NA |

dat_join2

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| d | NA | 3 |

dat_join3

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |

dat_join4

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | NA |
| d | NA | 3 |

- by=c("x","y")表示多列匹配,
dat_join5 <-dplyr::left_join (x=df1,y=df4,by= c("x","y"))
- by=c("x"="g")可以根据两个表的不同列名合并
dat_join6 <-dplyr::left_join (x=df1,y=df3,by= c("x"="g"))
- 如果和表合并的过程中遇到有一列两个表都同名，但是值不同，合并的时候又都想保留下来，就可以用suffixes给每个表的重复列名增加后缀
dat_join7<-dplyr::left_join (x=df1,y=df4,by="x", suffix=c(".1",".2"))

dat_join5

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | NA |
| c | 3 | NA |

dat_join6

| x | y | z |
|---|---|---|
| a | 1 | 2 |
| b | 2 | 5 |
| c | 3 | NA |

dat_join7

| x | y.1 | y.2 | z |
|---|---|---|---|
| a | 1 | 1 | 2 |
| b | 2 | 4 | 5 |
| c | 3 | NA | NA |

## 表格的变换与排序

(1) 表格的变换
df<- data.frame(x=c('A','B','C'),'2010'=c(1,3,4),'2011'=c(3,5,2),check.names=FALSE)

- 宽数据转换为长数据，将多行聚集成列，二维表变成一维表：
- df_melt<- reshape2::melt(df, id.vars='x',variable.name="year", value.name = "value")
- df_gather<- tidyr:: gather(df,year,value,-x)

- 长数据转换为宽数据，将一列根据变量展开为多行：
- df_dcast<- reshape2:: dcast (df_melt,x~year,value.var="value")
- df_spread <- tidyr::spread(df_gather,year, value)

df

| x | 2010 | 2011 |
|---|---|---|
| A | 1 | 3 |
| B | 3 | 5 |
| C | 4 | 2 |

df_melt/ df_gather

| x | year | Value |
|---|---|---|
| A | 2010 | 1 |
| B | 2010 | 3 |
| C | 2010 | 4 |
| A | 2011 | 3 |
| B | 2011 | 5 |
| C | 2011 | 2 |

df_dcast/ df_spread

| x | 2010 | 2011 |
|---|---|---|
| A | 1 | 3 |
| B | 3 | 5 |
| C | 4 | 2 |

## 表格的变换与排序

(2) 变量的转换，根据原有数据框大的列，计算添加新的列：

- dat1<-transform(df_melt, value2=value*2)
- dat2<- transform(df_melt, value2=ifelse(year=="2011", value*2, value))
- dat2<- dplyr::mutate(df_melt, value2=ifelse(year=="2011", value*2, value))

dat1

| X | year | value | value2 |
|---|---|---|---|
| A | 2010 | 1 | 2 |
| B | 2010 | 3 | 6 |
| C | 2010 | 4 | 8 |
| A | 2011 | 3 | 6 |
| B | 2011 | 5 | 10 |
| C | 2011 | 2 | 4 |

dat2

| X | year | value | Value2 |
|---|---|---|---|
| A | 2010 | 1 | 1 |
| B | 2010 | 3 | 3 |
| C | 2010 | 4 | 4 |
| A | 2011 | 3 | 6 |
| B | 2011 | 5 | 10 |
| C | 2011 | 2 | 4 |

(3)表格的排序：根据数据框的某列数值对整个表排序

- dat_arrange1<- dplyr:: arrange (df_melt, value)
- dat_arrange2<- dplyr:: arrange (df_melt, desc(value))

dat_arrange1

| X | year | Value |
|---|---|---|
| A | 2010 | 1 |
| C | 2011 | 2 |
| B | 2010 | 3 |
| A | 2011 | 3 |
| C | 2010 | 4 |
| B | 2011 | 5 |

dat_arrange2

| X | year | Value |
|---|---|---|
| B | 2011 | 5 |
| C | 2010 | 4 |
| A | 2011 | 3 |
| B | 2010 | 3 |
| C | 2011 | 2 |
| A | 2010 | 1 |

## 表格的分组操作

```
df<- data.frame(x=c('A','B','C', 'A', 'C'),'2010'=c(1,3,4,4,3),
                '2011'=c(3,5,2,8,9), check.names=FALSE)
df_melt<- reshape2::melt(df, id.vars='x',
                variable.name="year",value.name = "value")
```

| X | 2010 | 2011 |
|---|---|---|
| A | 1 | 3 |
| B | 3 | 5 |
| C | 4 | 2 |
| A | 4 | 8 |
| C | 3 | 9 |

| Sum |
|---|
| 4 |
| 8 |
| 6 |
| 12 |
| 12 |

| X | year | Value |
|---|---|---|
| A | 2010 | 1 |
| B | 2010 | 3 |
| C | 2010 | 4 |
| A | 2010 | 4 |
| C | 2010 | 3 |
| A | 2011 | 3 |
| B | 2011 | 5 |
| C | 2011 | 2 |
| A | 2011 | 8 |
| C | 2011 | 9 |

| | |
|---|---|
| 15 | 27 |

(1) 分行row或列column操作，
可以使用apply()函数按行或列求和：
df_rowsum<-apply(df[,2:3],1,sum)
df_colsum<-apply(df[,2:3],2,sum)

(2)分组group操作数据

- 只根据year分组操作
- df_group1<- aggregate(value~year,df_melt,mean)
- df_group1<-df_melt %>% dplyr ::group_by(year) %>%
  dplyr ::summarise(avg = mean(value))

先分组group_by(year)

| X | year | Value |
|---|---|---|
| A | 2010 | 1 |
| B | 2010 | 3 |
| C | 2010 | 4 |
| A | 2010 | 4 |
| C | 2010 | 3 |
| A | 2011 | 3 |
| B | 2011 | 5 |
| C | 2011 | 2 |
| A | 2011 | 8 |
| C | 2011 | 9 |

后计算df_group1

| year | avg |
|---|---|
| 2010 | 3.0 |
| 2010 | 5.4 |

- 同时根据year和x两个变量分组操作
- df_group2<- aggregate(value~year+x,df_melt,mean)
- df_group2<-df_melt %>% dplyr ::group_by(year,x) %>%
  dplyr::summarise(avg = mean(value))

先分组group_by(year,x)

| X | year | Value |
|---|---|---|
| A | 2010 | 1 |
| A | 2010 | 4 |
| B | 2010 | 3 |
| C | 2010 | 4 |
| C | 2010 | 3 |
| A | 2011 | 3 |
| A | 2011 | 8 |
| B | 2011 | 5 |
| C | 2011 | 2 |
| C | 2011 | 9 |

后计算df_group2

| year | X | avg |
|---|---|---|
| 2010 | A | 2.5 |
| 2010 | B | 3.0 |
| 2010 | C | 3.5 |
| 2011 | A | 5.5 |
| 2011 | B | 5.0 |
| 2011 | C | 5.5 |

**多步操作连接符%>%**

dplyr包里还新引进了一个操作符,%>%, 使用时把数据集名作为开头，然后依次对此数据进行多步操作。这种运算符的编写方式使得编程者可以按数据处理时的思路写代码，一步一步操作不断叠加，在程序上就可以非常清晰的体现数据处理的步骤与背后的逻辑。

## 数据的基本操作包package

- base包

base包是安装R语言时会自带的包，已经包含一些数据框基本操作的函数，比如rbind, cbind等函数。

- dplyr包

dplyr包是 Hadley Wickham (ggplot2包的作者，被称作“一个改变R的人”)的杰作, 并自称 a grammar of data manipulation, 他将原本plyr 包中的ddply()等函数进一步分离强化，专注接受dataframe对象，大幅提高了速度，并且提供了更稳健的与其它数据库对象间的接口。

- tidyr包

tidyr包往往与dplyr包结合使用，目前渐有取代reshape2包之势，是值得关注的一个R包。在tidyr包中，有四个常用的函数，分别是：gather()：宽数据转换为长数据；spread()：长数据转换为宽数据；unite()：多列合并为一列；separate()：将一列分离为多列。

- reshape2包

reshape2包是由Hadley Wickham开发的用于数据重构的包，其主要功能函数为melt、cast，实现了长数据和宽数据之间的转换，包中还包含其它函数和数据集。

install.packages('dplyr') #包package的安装
library(dplyr)　#包package的导入，包里面的函数可直接使用
dplyr::select　#使用包里面的特定某个函数
?? select　#可以实现对某个函数功能的检索


**张杰：微信公众号-EasyCharts**

欢迎关注