dplyr 介绍

在处理数据时，您必须：

- 弄清楚你想做什么。
- 以计算机程序的形式描述这些任务。
- 执行程序。

dplyr 包使这些步骤变得快速而简单：

- 通过限制您的选择，它可以帮助您考虑数据操作挑战。
- 它提供了简单的“动词”，即与最常见的数据操作任务相对应的函数，以帮助您将您的想法转化为代码。
- 它使用高效的后端，因此您等待计算机的时间更少。

本文档向您介绍了 dplyr 的基本工具集，并向您展示了如何将它们应用于数据框。dplyr 还通过 dbplyr 包支持数据库，一旦安装，请阅读`vignette("dbplyr")`以了解更多信息。

资料：星战

为了探索 dplyr 的基本数据操作动词，我们将使用 dataset `starwars`。该数据集包含 87 个字符，来自Star Wars API，并记录在`?starwars`

```
dim(starwars)
#> [1] 87 14
starwars
#> # A tibble: 87 x 14
#>   name      height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>      <int> <dbl> <chr>      <chr>       <chr>           <dbl> <chr> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue             19   male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow          112   none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red              33   none  mascu…
#> 4 Darth V…    202   136 none       white       yellow           41.9 male  mascu…
#> # … with 83 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

请注意，这`starwars`是对数据框的现代重新构想。它对大型数据集特别有用，因为它只打印前几行。您可以在https://tibble.tidyverse.org 上了解有关 tibbles 的更多信息；特别是您可以将数据帧转换为 tibbles `as_tibble()`。

单表动词

dplyr 旨在为数据操作的每个基本动词提供一个函数。这些动词可以根据它们使用的数据集的组件分为三类：

- 行：
  - `filter()` 根据列值选择行。
  - `slice()` 根据位置选择行。
  - `arrange()` 改变行的顺序。
- 列：
  - `select()` 更改是否包含列。
  - `rename()` 更改列的名称。
  - `mutate()` 更改列的值并创建新列。

    - `relocate()` 更改列的顺序。
- 行组：
    - `summarise()` 将一个组折叠成一行。

管道

所有 dplyr 函数都将数据框（或 tibble）作为第一个参数。dplyr 不是强制用户保存中间对象或嵌套函数，而是`%>%`从 magrittr 中提供操作符。`x %>% f(y)`变成`f(x, y)`这样，然后将一个步骤的结果“输送”到下一步。您可以使用管道重写多个操作，您可以从左到右、从上到下读取这些操作（将管道运算符读作“then”）。

过滤行 `filter()`

`filter()`允许您在数据框中选择行的子集。像所有单个动词一样，第一个参数是 tibble（或数据框）。第二个和后续参数引用该数据框中的变量，选择表达式为 的行`TRUE`。

例如，我们可以选择所有浅肤色和棕色眼睛的角色：

```
starwars %>% filter(skin_color == "light", eye_color == "brown")
#> # A tibble: 7 x 14
#>   name     height  mass hair_color skin_color eye_color birth_year sex    gender
#>   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>  <chr>
#> 1 Leia Or…    150    49 brown      light      brown             19 female femin…
#> 2 Biggs D…    183    84 black      light      brown             24 male   mascu…
#> 3 Cordé       157    NA brown      light      brown             NA female femin…
#> 4 Dormé       165    NA brown      light      brown             NA female femin…
#> # … with 3 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

这大致相当于这个基本的 R 代码：

```
starwars[starwars$skin_color == "light" & starwars$eye_color == "brown", ]
```

排列行 `arrange()`

`arrange()`与`filter()`除了过滤或选择行之外，它的工作方式与其他类似，只是对它们重新排序。它需要一个数据框和一组列名（或更复杂的表达式）来排序。如果您提供多个列名称，则每个附加列都将用于打破前列值之间的联系：

```
starwars %>% arrange(height, mass)
#> # A tibble: 87 x 14
#>   name      height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Yoda          66    17 white      green      brown            896 male  mascu…
#> 2 Ratts Ty…     79    15 none       grey, blue unknown           NA male  mascu…
#> 3 Wicket S…     88    20 brown      brown      brown              8 male  mascu…
#> 4 Dud Bolt      94    45 none       blue, grey yellow            NA male  mascu…
#> # … with 83 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

用于`desc()`按降序对列进行排序：

```
starwars %>% arrange(desc(height))
#> # A tibble: 87 x 14
```

```
#>   name     height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Yarael …    264    NA none       white      yellow            NA male  mascul…
#> 2 Tarfful     234   136 brown      brown      blue              NA male  mascul…
#> 3 Lama Su     229    88 none       grey       black             NA male  mascul…
#> 4 Chewbac…    228   112 brown      unknown    blue             200 male  mascul…
#> # … with 83 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

使用它们的位置选择行 `slice()`

`slice()`允许您按（整数）位置索引行。它允许您选择、删除和复制行。

我们可以从行号 5 到 10 中获取字符。

```
starwars %>% slice(5:10)
#> # A tibble: 6 x 14
#>   name     height  mass hair_color  skin_color eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>       <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Leia Or…    150    49 brown       light      brown             19 fema… femin…
#> 2 Owen La…    178   120 brown, grey light      blue              52 male  mascu…
#> 3 Beru Wh…    165    75 brown       light      blue              47 fema… femin…
#> 4 R5-D4        97    32 <NA>        white, red red               NA none  mascu…
#> # … with 2 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

它伴随着一些常见用例的助手：

- `slice_head()`并`slice_tail()`选择第一行或最后一行。

```
starwars %>% slice_head(n = 3)
#> # A tibble: 3 x 14
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue              19 male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow           112 none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red               33 none  mascu…
#> # … with 5 more variables: homeworld <chr>, species <chr>, films <list>,
#> #   vehicles <list>, starships <list>
```

- `slice_sample()`随机选择行。使用选项道具选择一定比例的案例。

```
starwars %>% slice_sample(n = 5)
#> # A tibble: 5 x 14
#>   name   height  mass hair_color skin_color   eye_color birth_year sex    gender
#>   <chr>   <int> <dbl> <chr>      <chr>        <chr>          <dbl> <chr>  <chr>
#> 1 Dud B…     94    45 none       blue, grey   yellow            NA male   mascu…
#> 2 Bossk     190   113 none       green        red               53 male   mascu…
#> 3 Shaak…    178    57 none       red, blue, … black             NA female femin…
#> 4 Dormé     165    NA brown      light        brown             NA female femin…
#> # … with 1 more row, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
starwars %>% slice_sample(prop = 0.1)
#> # A tibble: 8 x 14
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
```

```
#> 1 Qui-Gon…    193    89 brown      fair        blue             92 male  mascu…
#> 2 Dexter …    198   102 none       brown       yellow           NA male  mascu…
#> 3 R4-P17       96    NA none       silver, red red, blue        NA none  femin…
#> 4 Lama Su     229    88 none       grey        black            NA male  mascu…
#> # … with 4 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

使用`replace = TRUE`进行引导样品。如果需要，您可以使用`weight`参数对样本进行加权。

- `slice_min()`并`slice_max()`选择具有最高或最低变量值的行。请注意，我们首先必须只选择不是 NA 的值。

```
starwars %>%
  filter(!is.na(height)) %>%
  slice_max(height, n = 3)
#> # A tibble: 3 x 14
#>   name     height  mass hair_color skin_color eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
#> 1 Yarael …    264    NA none       white      yellow            NA male  mascul…
#> 2 Tarfful     234   136 brown      brown      blue              NA male  mascul…
#> 3 Lama Su     229    88 none       grey       black             NA male  mascul…
#> # … with 5 more variables: homeworld <chr>, species <chr>, films <list>,
#> #   vehicles <list>, starships <list>
```

选择列 `select()`

通常，您使用包含许多列的大型数据集，但实际上只有少数列是您感兴趣的。`select()`允许您使用通常仅适用于数字变量位置的操作快速放大有用的子集：

```
# Select columns by name
starwars %>% select(hair_color, skin_color, eye_color)
#> # A tibble: 87 x 3
#>   hair_color skin_color  eye_color
#>   <chr>      <chr>       <chr>
#> 1 blond      fair        blue
#> 2 <NA>       gold        yellow
#> 3 <NA>       white, blue red
#> 4 none       white       yellow
#> # … with 83 more rows
# Select all columns between hair_color and eye_color (inclusive)
starwars %>% select(hair_color:eye_color)
#> # A tibble: 87 x 3
#>   hair_color skin_color  eye_color
#>   <chr>      <chr>       <chr>
#> 1 blond      fair        blue
#> 2 <NA>       gold        yellow
#> 3 <NA>       white, blue red
#> 4 none       white       yellow
#> # … with 83 more rows
# Select all columns except those from hair_color to eye_color (inclusive)
starwars %>% select(!(hair_color:eye_color))
#> # A tibble: 87 x 11
#>   name    height  mass birth_year sex   gender  homeworld species films vehicles
#>   <chr>    <int> <dbl>      <dbl> <chr> <chr>   <chr>     <chr>   <lis> <list>
#> 1 Luke S…    172    77       19   male  mascul… Tatooine  Human   <chr… <chr [2…
#> 2 C-3PO      167    75      112   none  mascul… Tatooine  Droid   <chr… <chr [0…
```

```
#> 3 R2-D2       96    32       33   none  mascul… Naboo     Droid   <chr… <chr [0…
#> 4 Darth …    202   136       41.9 male  mascul… Tatooine  Human   <chr… <chr [0…
#> # … with 83 more rows, and 1 more variable: starships <list>
# Select all columns ending with color
starwars %>% select(ends_with("color"))
#> # A tibble: 87 x 3
#>   hair_color skin_color  eye_color
#>   <chr>      <chr>       <chr>
#> 1 blond      fair        blue
#> 2 <NA>       gold        yellow
#> 3 <NA>       white, blue red
#> 4 none       white       yellow
#> # … with 83 more rows
```

这里有许多可以中使用的辅助功能`select()`，比如`starts_with()`，`ends_with()`，`matches()`和`contains()`。这些使您可以快速匹配满足某些标准的更大的变量块。见`?select`更多的细节。

您可以`select()`使用命名参数重命名变量：

```
starwars %>% select(home_world = homeworld)
#> # A tibble: 87 x 1
#>   home_world
#>   <chr>
#> 1 Tatooine
#> 2 Tatooine
#> 3 Naboo
#> 4 Tatooine
#> # … with 83 more rows
```

但是因为`select()`删除了所有未明确提及的变量，所以它没有那么有用。相反，使用`rename()`：

```
starwars %>% rename(home_world = homeworld)
#> # A tibble: 87 x 14
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue            19   male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow         112   none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red             33   none  mascu…
#> 4 Darth V…    202   136 none       white       yellow          41.9 male  mascu…
#> # … with 83 more rows, and 5 more variables: home_world <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

添加新列 `mutate()`

除了选择现有列的集合之外，添加作为现有列的函数的新列通常很有用。这是的工作`mutate()`：

```
starwars %>% mutate(height_m = height / 100)
#> # A tibble: 87 x 15
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue            19   male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow         112   none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red             33   none  mascu…
#> 4 Darth V…    202   136 none       white       yellow          41.9 male  mascu…
```

```
#> # … with 83 more rows, and 6 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>, height_m <dbl>
```

我们看不到刚刚计算的以米为单位的高度，但我们可以使用 select 命令修复它。

```
starwars %>%
  mutate(height_m = height / 100) %>%
  select(height_m, height, everything())
#> # A tibble: 87 x 15
#>   height_m height name     mass hair_color skin_color eye_color birth_year sex
#>      <dbl>  <int> <chr>   <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
#> 1     1.72    172 Luke S…    77 blond      fair       blue            19   male
#> 2     1.67    167 C-3PO      75 <NA>       gold       yellow         112   none
#> 3     0.96     96 R2-D2      32 <NA>       white, bl… red             33   none
#> 4     2.02    202 Darth …   136 none       white      yellow          41.9 male
#> # … with 83 more rows, and 6 more variables: gender <chr>, homeworld <chr>,
#> #   species <chr>, films <list>, vehicles <list>, starships <list>
```

`dplyr::mutate()`类似于 base `transform()`，但允许您引用您刚刚创建的列：

```
starwars %>%
  mutate(
    height_m = height / 100,
    BMI = mass / (height_m^2)
  ) %>%
  select(BMI, everything())
#> # A tibble: 87 x 16
#>     BMI name       height  mass hair_color skin_color eye_color birth_year sex
#>   <dbl> <chr>       <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
#> 1  26.0 Luke Skyw…    172    77 blond      fair       blue            19   male
#> 2  26.9 C-3PO         167    75 <NA>       gold       yellow         112   none
#> 3  34.7 R2-D2          96    32 <NA>       white, bl… red             33   none
#> 4  33.3 Darth Vad…    202   136 none       white      yellow          41.9 male
#> # … with 83 more rows, and 7 more variables: gender <chr>, homeworld <chr>,
#> #   species <chr>, films <list>, vehicles <list>, starships <list>,
#> #   height_m <dbl>
```

如果您只想保留新变量，请使用`transmute()`：

```
starwars %>%
  transmute(
    height_m = height / 100,
    BMI = mass / (height_m^2)
  )
#> # A tibble: 87 x 2
#>   height_m   BMI
#>      <dbl> <dbl>
#> 1     1.72  26.0
#> 2     1.67  26.9
#> 3     0.96  34.7
#> 4     2.02  33.3
#> # … with 83 more rows
```

更改列顺序 `relocate()`

使用与`select()`一次移动列块类似的语法

```
starwars %>% relocate(sex:homeworld, .before = height)
#> # A tibble: 87 x 14
#>   name      sex   gender  homeworld height  mass hair_color skin_color eye_color
#>   <chr>     <chr> <chr>   <chr>      <int> <dbl> <chr>      <chr>      <chr>
#> 1 Luke Sky… male  mascul… Tatooine     172    77 blond      fair       blue
#> 2 C-3PO     none  mascul… Tatooine     167    75 <NA>       gold       yellow
#> 3 R2-D2     none  mascul… Naboo         96    32 <NA>       white, bl… red
#> 4 Darth Va… male  mascul… Tatooine     202   136 none       white      yellow
#> # … with 83 more rows, and 5 more variables: birth_year <dbl>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

总结价值 `summarise()`

最后一个动词是`summarise()`。它将数据框折叠为单行。

```
starwars %>% summarise(height = mean(height, na.rm = TRUE))
#> # A tibble: 1 x 1
#>   height
#>    <dbl>
#> 1   174.
```

在我们学习`group_by()`下面的动词之前，它并没有那么有用。

共性

您可能已经注意到，所有这些动词的语法和功能都非常相似：

- 第一个参数是一个数据框。
- 随后的参数描述了如何处理数据框。您可以直接引用数据框中的列，而无需使用`$`.
- 结果是一个新的数据框

将这些属性结合在一起，可以轻松地将多个简单步骤链接在一起以实现复杂的结果。

这五个函数提供了数据操作语言的基础。在最基本的层面上，你只能通过五种有用的方式来改变一个整洁的数据框：你可以重新排列行 (`arrange()`)，选择感兴趣的观察和变量 (`filter()`and `select()`)，添加作为现有变量的函数的新变量 (`mutate()`)，或者将许多值折叠为一个摘要 (`summarise()`)。

将功能与 `%>%`

dplyr API 是功能性的，因为函数调用没有副作用。您必须始终保存他们的结果。这不会产生特别优雅的代码，尤其是当您想一次执行许多操作时。你要么必须一步一步地做：

```
a1 <- group_by(starwars, species, sex)
a2 <- select(a1, height, mass)
a3 <- summarise(a2,
  height = mean(height, na.rm = TRUE),
  mass = mean(mass, na.rm = TRUE)
)
```

或者，如果您不想命名中间结果，则需要将函数调用相互包装起来：

```
  summarise(
    select(
      group_by(starwars, species, sex),
      height, mass
    ),
    height = mean(height, na.rm = TRUE),
    mass = mean(mass, na.rm = TRUE)
  )
#> Adding missing grouping variables: `species`, `sex`
#> `summarise()` has grouped output by 'species'. You can override using the `.groups` argument.
#> # A tibble: 41 x 4
#> # Groups:   species [38]
#>   species  sex   height  mass
#>   <chr>    <chr>  <dbl> <dbl>
#> 1 Aleena   male      79    15
#> 2 Besalisk male     198   102
#> 3 Cerean   male     198    82
#> 4 Chagrian male     196   NaN
#> # … with 37 more rows
```

这很难阅读，因为操作的顺序是从内到外。因此，参数离函数很远。为了解决这个问题，dplyr 提供了`%>%`来自 magrittr的操作符。`x %>% f(y)`变成了`f(x, y)`这样你就可以用它来重写多个操作，你可以从左到右、从上到下读取这些操作（将管道运算符读作“then”）：

```
starwars %>%
  group_by(species, sex) %>%
  select(height, mass) %>%
  summarise(
    height = mean(height, na.rm = TRUE),
    mass = mean(mass, na.rm = TRUE)
  )
```

操作模式

dplyr 动词可以根据它们完成的操作类型进行分类（我们有时会谈论它们的**语义**，即它们的含义）。很好地掌握 select 和 mutate 操作之间的区别会很有帮助。

选择操作

dplyr 的一个吸引人的特性是您可以引用 tibble 中的列，就好像它们是常规变量一样。然而，引用裸列名称的句法一致性隐藏了动词之间的语义差异。提供给 的列符号与提供给`select()`的相同符号的含义不同`mutate()`。

选择操作需要列名和位置。因此，当您`select()`使用裸变量名调用时，它们实际上代表了它们在 tibble 中的位置。从 dplyr 的角度来看，以下调用完全等效：

```
# `name` represents the integer 1
select(starwars, name)
#> # A tibble: 87 x 1
#>   name
#>   <chr>
#> 1 Luke Skywalker
#> 2 C-3PO
#> 3 R2-D2
```

```
#> 4 Darth Vader
#> # … with 83 more rows
select(starwars, 1)
#> # A tibble: 87 x 1
#>   name
#>   <chr>
#> 1 Luke Skywalker
#> 2 C-3PO
#> 3 R2-D2
#> 4 Darth Vader
#> # … with 83 more rows
```

出于同样的原因，这意味着您不能从周围的上下文中引用变量，如果它们与列之一具有相同的名称。在下面的例子中，`height`仍然代表 2，而不是 5：

```
height <- 5
select(starwars, height)
#> # A tibble: 87 x 1
#>   height
#>    <int>
#> 1    172
#> 2    167
#> 3     96
#> 4    202
#> # … with 83 more rows
```

一个有用的微妙之处是，这仅适用于裸名称和选择像`c(height, mass)`或 之类的调用`height:mass`。在所有其他情况下，数据框的列不在范围内。这允许您在选择助手中引用上下文变量：

```
name <- "color"
select(starwars, ends_with(name))
#> # A tibble: 87 x 3
#>   hair_color skin_color  eye_color
#>   <chr>      <chr>       <chr>
#> 1 blond      fair        blue
#> 2 <NA>       gold        yellow
#> 3 <NA>       white, blue red
#> 4 none       white       yellow
#> # … with 83 more rows
```

这些语义通常是直观的。但请注意细微的区别：

```
name <- 5
select(starwars, name, identity(name))
#> # A tibble: 87 x 2
#>   name           skin_color
#>   <chr>          <chr>
#> 1 Luke Skywalker fair
#> 2 C-3PO          gold
#> 3 R2-D2          white, blue
#> 4 Darth Vader    white
#> # … with 83 more rows
```

在第一个参数中，`name`代表它自己的位置`1`。在第二个参数中，`name`在周围的上下文中计算并表示第五列。

很长一段时间，`select()`习惯只了解列位置。从 dplyr 0.6 开始，它现在也能理解列名。这使得编程更容易一些`select()`：

```
vars <- c("name", "height")
select(starwars, all_of(vars), "mass")
#> # A tibble: 87 x 3
#>   name           height  mass
#>   <chr>           <int> <dbl>
#> 1 Luke Skywalker    172    77
#> 2 C-3PO             167    75
#> 3 R2-D2              96    32
#> 4 Darth Vader       202   136
#> # … with 83 more rows
```

变异操作

变异语义与选择语义完全不同。而`select()`期望列名或位置，`mutate()`期望列向量。我们将设置一个较小的 tibble 用于我们的示例。

```
df <- starwars %>% select(name, height, mass)
```

当我们使用 时`select()`，裸列名称代表它们在 tibble 中的位置。对于`mutate()`在另一方面，列符号表示存储在tibble实际的列向量。考虑如果我们给一个字符串或一个数字会发生什么`mutate()`：

```
mutate(df, "height", 2)
#> # A tibble: 87 x 5
#>   name           height  mass `"height"`   `2`
#>   <chr>           <int> <dbl> <chr>      <dbl>
#> 1 Luke Skywalker    172    77 height         2
#> 2 C-3PO             167    75 height         2
#> 3 R2-D2              96    32 height         2
#> 4 Darth Vader       202   136 height         2
#> # … with 83 more rows
```

`mutate()`获取长度为 1 的向量，将其解释为数据框中的新列。这些向量被回收，因此它们匹配行数。这就是为什么提供像`"height" + 10`to这样的表达式没有意义`mutate()`。这相当于将 10 添加到字符串中！正确的表达是：

```
mutate(df, height + 10)
#> # A tibble: 87 x 4
#>   name           height  mass `height + 10`
#>   <chr>           <int> <dbl>         <dbl>
#> 1 Luke Skywalker    172    77           182
#> 2 C-3PO             167    75           177
#> 3 R2-D2              96    32           106
#> 4 Darth Vader       202   136           212
#> # … with 83 more rows
```

同样，如果这些值代表有效列，您可以从上下文中取消引用这些值。它们的长度必须为 1（然后被回收）或与行数具有相同的长度。在以下示例中，我们创建了一个新向量，并将其添加到数据框中：

```
var <- seq(1, nrow(df))
mutate(df, new = var)
#> # A tibble: 87 x 4
```

```
#>   name           height  mass   new
#>   <chr>           <int> <dbl> <int>
#> 1 Luke Skywalker    172    77     1
#> 2 C-3PO             167    75     2
#> 3 R2-D2              96    32     3
#> 4 Darth Vader       202   136     4
#> # … with 83 more rows
```

一个恰当的例子是group_by()。虽然您可能认为它具有选择语义，但它实际上具有变异语义。这非常方便，因为它允许按修改后的列进行分组：

```
group_by(starwars, sex)
#> # A tibble: 87 x 14
#> # Groups:   sex [5]
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue            19   male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow         112   none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red             33   none  mascu…
#> 4 Darth V…    202   136 none       white       yellow          41.9 male  mascu…
#> # … with 83 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
group_by(starwars, sex = as.factor(sex))
#> # A tibble: 87 x 14
#> # Groups:   sex [5]
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <fct> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue            19   male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow         112   none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red             33   none  mascu…
#> 4 Darth V…    202   136 none       white       yellow          41.9 male  mascu…
#> # … with 83 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
group_by(starwars, height_binned = cut(height, 3))
#> # A tibble: 87 x 15
#> # Groups:   height_binned [4]
#>   name     height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>     <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Sk…    172    77 blond      fair        blue            19   male  mascu…
#> 2 C-3PO       167    75 <NA>       gold        yellow         112   none  mascu…
#> 3 R2-D2        96    32 <NA>       white, blue red             33   none  mascu…
#> 4 Darth V…    202   136 none       white       yellow          41.9 male  mascu…
#> # … with 83 more rows, and 6 more variables: homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>, height_binned <fct>
```

这就是为什么您不能为group_by(). 这相当于创建一个包含循环到行数的字符串的新列：

```
group_by(df, "month")
#> # A tibble: 87 x 4
#> # Groups:   "month" [1]
#>   name           height  mass `"month"`
#>   <chr>           <int> <dbl> <chr>
#> 1 Luke Skywalker    172    77 month
#> 2 C-3PO             167    75 month
#> 3 R2-D2              96    32 month
```

```
#> 4 Darth Vader       202   136 month
#> # … with 83 more rows
```