R：tidyverse 分析流程

2023 年 12 月 28 日

tidyverse 是什么 tidyverse 出自于 R 大神 Hadley Wickham 之手，他是 Rstudio 首席科学家，也是 ggplot2 的作者。tidyverse 就是他将自己所写的包整理成了一整套数据处理的方法，包括 ggplot2，dplyr，tidyr，readr，purrr，tibble，stringr, forcats。同时也出了一本《R for Data Science》.

在 R 语言中，tidyverse 是一个庞大的数据分析生态系统，它由一系列数据可视化和数据处理软件包组成，能够极大地提高数据分析的效率和准确性。

在使用 Tidyverse 的过程中，我们会经常用到以下几个工具：

- ggplot2：用于数据可视化，可以绘制各种类型的图表，如散点图、柱状图、折线图等。
- dplyr：数据整理和转换工具，使用 pipe（%>%）操作符来实现数据的转换和筛选。
- tidyr：用于数据整理，可以将数据从宽型转换成长型，或将多个变量合并为一个变量。
- readr：用于读取常见的数据格式，如 CSV、TXT 等。
- stringr：用于字符串处理，可以进行字符串匹配、提取、替换等操作。
- tibble：用于创建数据框。

[1]:
```
library(tidyverse)
library(rio)
library(showtext) # 支持中文
showtext_auto()
library(formatR) # auto pretty code: `ctrl + L`
# library(plotly) # 将 ggplot 图表转换为交互式图表 关闭悬停时的报错显示:
 ↪`ggplotly(plot, tooltip = NULL)`
# library(bruceR)
# library(ggplot2)
# library(data.table)
```

```
── Attaching core tidyverse packages ──
tidyverse 2.0.0
 dplyr     1.1.3      readr     2.1.4
```

```
 forcats   1.0.0      stringr   1.5.0
 ggplot2   3.4.4      tibble    3.2.1
 lubridate 1.9.3      tidyr     1.3.0
 purrr     1.0.2
  Conflicts
tidyverse_conflicts()
 dplyr::filter()  masks stats::filter()
 purrr::flatten() masks
jsonlite::flatten()
 dplyr::lag()     masks stats::lag()
 Use the conflicted package
(<http://conflicted.r-lib.org/>) to force all conflicts to
become errors
载入需要的程辑包：sysfonts

载入需要的程辑包：showtextdb
```

1 readr

readr：readcsv(); readtsv(); readdelim(); readfwf(); readtable(); readlog(); readxl：readxls(); readxlsx(); haven：打开 SAS 、SPSS、Stata 等外部数据。

1.1 直接创建 tibble 格式数据

[2]:
```
x <- c(1, 4, 5)
y <- c(211, 23, 45)
z <- c(20, 32)
```

[3]:
```
my_tibble_2 <- tibble(v1 = x, v2 = y)
my_tibble_2
```

A tibble: 3 × 2

v1	v2
<dbl>	<dbl>
1	211
4	23
5	45

1.2 转换数据：as_tibble()

[4]:
```
iris %>%
    head(2)
```

A data.frame: 2 × 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	3.0	1.4	0.2	setosa

[5]:
```
iris %>%
    head(2) %>%
    as_tibble()
```

A tibble: 2 × 5

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
<dbl>	<dbl>	<dbl>	<dbl>	<fct>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa

2 dplyr

dplyr 是一个强大的 R 库，用于数据的整理和转换，它具有简单易用的语法和高效的设计，通常使用%>% 运算符来组合多种操作。

dplyr 提供一些基本操作，包括:

- 选择数据列（select）
- 重命名数据列（rename）
- 过滤观测值（filter）
- 排序（arrange）
- 添加新变量（mutate）
- 分组汇总（summarize）

- 连接数据集（join）等。

2.1 filter

[6]:
```
iris %>%
    filter(Species == "virginica", Sepal.Length > 7.5)
```

A data.frame: 6 × 5

Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
7.6	3.0	6.6	2.1	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica
7.7	2.8	6.7	2.0	virginica
7.9	3.8	6.4	2.0	virginica
7.7	3.0	6.1	2.3	virginica

2.2 arrange

[7]:
```
iris %>%
    head(5) %>%
    arrange(Species)
```

A data.frame: 5 × 5

Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

2.3 rename

[8]:
```
iris1 <- iris %>% head(3)
iris1
```

A data.frame: 3 × 5

	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa

[9]:
```
iris1 %>% names()
```

1. 'Sepal.Length' 2. 'Sepal.Width' 3. 'Petal.Length' 4. 'Petal.Width' 5. 'Species'

[10]:
```
iris1 %>%
    rename(SPECIES = Species)  # 原名称在后
```

A data.frame: 3 × 5

	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	SPECIES <fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa

2.4 select

[11]:
```
iris %>%
    select(Species, Sepal.Length) %>%
    head(5)
```

A data.frame: 5 × 2

	Species <fct>	Sepal.Length <dbl>
1	setosa	5.1
2	setosa	4.9
3	setosa	4.7
4	setosa	4.6
5	setosa	5.0

2.5 mulate

[12]:
```
iris %>%
    mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%
    select(-Sepal.Length, -Sepal.Width) %>%
    head(5)
```

A data.frame: 5 × 4

	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>	Sepal.Area <dbl>
1	1.4	0.2	setosa	17.85
2	1.4	0.2	setosa	14.70
3	1.3	0.2	setosa	15.04
4	1.5	0.2	setosa	14.26
5	1.4	0.2	setosa	18.00

2.6 summarise

[13]:
```
iris %>%
    mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%
    filter(Sepal.Area < 15) %>%
    summarise(count = n(), mean = mean(Sepal.Area))
```

A data.frame: 1 × 2

count <int>	mean <dbl>
29	13.36724

2.7 groupby

[14]:
```
iris %>%
    mutate(Sepal.Area = Sepal.Width * Sepal.Length) %>%
    filter(Sepal.Area < 15) %>%
    group_by(Species) %>%
    summarise(count = n(), mean = mean(Sepal.Area))
```

A tibble: 3 × 3

Species	count	mean
<fct>	<int>	<dbl>
setosa	11	13.69545
versicolor	15	13.15333
virginica	3	13.23333

2.8 Join

[15]:
```
df1 <- data.frame(id = c(1,2,3), var1 = c("a","b","c"))
df1
```

A data.frame: 3 × 2

id	var1
<dbl>	<chr>
1	a
2	b
3	c

[16]:
```
df2 <- data.frame(id = c(1,2,4), var2 = c("A","B","D"))
df2
```

A data.frame: 3 × 2

id	var2
<dbl>	<chr>
1	A
2	B
4	D

[17]:
```
inner_join(df1, df2, by="id")
```

A data.frame: 2 × 3

id	var1	var2
<dbl>	<chr>	<chr>
1	a	A
2	b	B

3 tidyr

tidyr 是 Hadley（Tidy Data 的作者 Hadley Wickham）写的非常有用、并且经常会使用到的包，常与 dplyr 包结合使用（这个包也是他写的）...elt

tidyr 是一个 R 库，用于数据整理和转换，它强调数据的长型与宽型转换，经常与 dplyr 结合使用，提供了许多有用的函数，如 gather、spread、separate 和 unite 等。

- gather：将数据从宽型转换成长型。
- spread：将数据从长型转换成宽型。
- separate：将一个变量拆分成多个变量。
- unite：将多个变量合并为一个变量。

[18]:
```
library(tidyr)
```

[19]:
```
stu <- data.frame(grade = c("A", "B", "C", "D", "E"), female = c(5, 4, 1, 2, 3),
    male = c(1, 2, 3, 4, 5))
stu
```

A data.frame: 5 × 3

grade <chr>	female <dbl>	male <dbl>
A	5	1
B	4	2
C	1	3
D	2	4
E	3	5

3.1 gather：宽变长

gather 函数类似于 Excel（2016 起）中的数据透视的功能，能把一个变量名含有变量的二维表转换成一个规范的二维表（类似数据库中关系的那种表，具体看例子）

[20]:
```
stu_1 <-
gather(stu, gender, count, -grade)
stu_1
```

A data.frame: 10 × 3

grade <chr>	gender <chr>	count <dbl>
A	female	5
B	female	4
C	female	1
D	female	2
E	female	3
A	male	1
B	male	2
C	male	3
D	male	4
E	male	5

3.2 spread：长变宽

spread 是 gather 逆向操作。

用来扩展表，把某一列的值（键值对）分开拆成多列。

spread(data, key, value, fill = NA, convert = FALSE, drop =TRUE, sep = NULL)

key 是原来要拆的那一列的名字（变量名），value 是拆出来的那些列的值应该填什么（填原表的哪一列）

[21]:
```
spread(stu_1,gender,count)
```

A data.frame: 5 × 3

grade <chr>	female <dbl>	male <dbl>
A	5	1
B	4	2
C	1	3
D	2	4
E	3	5

3.3 separate：数据分列

separate 负责分割数据，把一个变量中就包含两个变量的数据分来（上例 gather 中是属性名也是一个变量，一个属性名一个变量）

separate 用法如下：

- separate(data, col, into, sep (= 正则表达式), remove =TRUE,convert = FALSE, extra = “warn”, fill = “warn”, ...)
- 第一个参数放要分离的数据框；
- 第二个参数放要分离的列；
- 第三个参数是分割成的变量的列（肯定是多个），用向量表示；
- 第四个参数是分隔符，用正则表达式表示，或者写数字，表示从第几位分开

[22]:
```
stu2 <- data.frame(grade = c("A", "B", "C", "D", "E"), female_1 = c(5, 4, 1, 2,␣
↪3),
   male_1 = c(1, 2, 3, 4, 5), female_2 = c(4, 5, 1, 2, 3), male_2 = c(0, 2, 3,␣
↪4,
       6))
stu2
```

A data.frame: 5 × 5

grade <chr>	female_1 <dbl>	male_1 <dbl>	female_2 <dbl>	male_2 <dbl>
A	5	1	4	0
B	4	2	5	2
C	1	3	1	3
D	2	4	2	4
E	3	5	3	6

[23]:
```
stu2_new <- gather(stu2, gender_class, count, -grade)
stu2_new
```

A data.frame: 20 × 3

grade <chr>	gender_class <chr>	count <dbl>
A	female_1	5
B	female_1	4
C	female_1	1
D	female_1	2
E	female_1	3
A	male_1	1
B	male_1	2
C	male_1	3
D	male_1	4
E	male_1	5
A	female_2	4
B	female_2	5
C	female_2	1
D	female_2	2
E	female_2	3
A	male_2	0
B	male_2	2
C	male_2	3
D	male_2	4
E	male_2	6

[24]:
```
separate(stu2_new,gender_class,c("gender","class"))
```

A data.frame: 20 × 4

grade <chr>	gender <chr>	class <chr>	count <dbl>
A	female	1	5
B	female	1	4
C	female	1	1
D	female	1	2
E	female	1	3
A	male	1	1
B	male	1	2
C	male	1	3
D	male	1	4
E	male	1	5
A	female	2	4
B	female	2	5
C	female	2	1
D	female	2	2
E	female	2	3
A	male	2	0
B	male	2	2
C	male	2	3
D	male	2	4
E	male	2	6

3.4 unite：多列数据合并

`unite(data, col, ..., sep = "_", remove = TRUE)` * 参数说明：data 数据；* col 构成新列的名称；* …选择你需要组合的列；* sep 值之间的分隔符，默认情况为”_”；* remove 是否删除被组合的列。

[25]:
```
stu_2 <- separate(stu2_new,gender_class,c("gender","class")) %>% head(10)
stu_2
```

A data.frame: 10 × 4

	grade <chr>	gender <chr>	class <chr>	count <dbl>
1	A	female	1	5
2	B	female	1	4
3	C	female	1	1
4	D	female	1	2
5	E	female	1	3
6	A	male	1	1
7	B	male	1	2
8	C	male	1	3
9	D	male	1	4
10	E	male	1	5

[26]:
```
unite(stu_2,"gender_class",c("gender","class"),sep="_")
```

A data.frame: 10 × 3

	grade <chr>	gender_class <chr>	count <dbl>
1	A	female_1	5
2	B	female_1	4
3	C	female_1	1
4	D	female_1	2
5	E	female_1	3
6	A	male_1	1
7	B	male_1	2
8	C	male_1	3
9	D	male_1	4
10	E	male_1	5

4 stringr

R 语言支持字符处理，内置了系列函数 (grep、gsub 等)，但系列函数定义混乱，对使用者极不方便。**stringr 包是专门用于字符处理的 R 包，函数定义简洁、使用方式统一，是使用率较高的 R 包。stringr 包中的大部分函数具有统一风格的命名方式，以 str_ 开头，正则表达式也完全适用该包。**

4.1 str_c：字符串拼接

字符串拼接函数 str_c，与 R 语言自带的 paste 和 paste0 函数具有相同的作用。

[27]:
```
# 默认无向量分割符拼接
str_c("a","b")
```

'ab'

[28]:
```
# 指定向量分隔符
str_c("a","b",sep = "_")
```

'a_b'

[29]:
```
# 指定向量折叠符
str_c(c("a","b","c"),collapse = "_")
```

'a_b_c'

[30]:
```
# 混合应用
str_c(c("a","b"),c("c","d"),sep = "/",collapse = "_")
```

'a/c_b/d'

4.2 str_count：字符计数

字符计数函数 str_count，计算字符串中指定字符的个数。

[31]:
```
# 单个目标字符计数
str_count(string = c("sql","json","java"),pattern = "s")
```

1. 1 2. 1 3. 0

[32]:
```
# 多个目标字符计数
str_count(string = c("sql","json","java"),pattern = c("s","j","a"))
```

1. 1 2. 1 3. 2

[33]:
```
# 元字符查找计数（fixed 包裹元字符）
str_count(string = "a..b",pattern = fixed("."))
```

2

4.3 str_detect：字符检查

字符检查函数 str_detect，检查字符串中是否包含指定字符，返回逻辑向量。

[34]:
```
str_detect(string = c("sql","json","java"),pattern = "s")
```

1. TRUE 2. TRUE 3. FALSE

4.4 str_dup：字符复制

字符复制函数 str_dup，将字符向量重复若干次，返回重复后的字符向量。

[35]:
```
str_dup(string = c("sql","json","java"),times = 2)
```

1. 'sqlsql' 2. 'jsonjson' 3. 'javajava'

4.5 str_extract：字符提取

字符提取函数 str_extract 和 str_extract_all，对字符串进行提取，str_extract_all 函数返回所有的匹配结果。

[36]:
```
# 提取第一个匹配到的字符
str_extract(string = "banana",pattern = "a")
```

'a'

[37]:
```
# 提取所有匹配到的字符（返回列表）
str_extract_all(string = "banana",pattern = "a")
```

1. (a) 'a' (b) 'a' (c) 'a'

[38]:
```
# 提取所有匹配到的字符（返回矩阵）
str_extract_all(string = "banana",pattern = "a",simplify = T)
```

A matrix: 1 × 3 of type chr a a a

4.6 Lstr_glue：字符串格式化

字符串格式化函数 str_glue，用花括号 {} 表示占位符，括号内的变量被替换成全局变量值。

[39]:
```
# 定义全局变量
name <- "jack"
age <- 12
# 字符串格式化
str_glue("My name is {name},","\n my age is {age}.")
```

'My name is jack,\nmy age is 12.'

4.7 str_length：字符串长度

字符串长度函数 str_length，计算字符串长度。

[40]:
```
str_length(string = "banana")
```

6

4.8 str_locate：字符位置提取

字符位置提取函数 str_locate 和 str_locate_all, 返回匹配到的字符的位置。

[41]:
```
# 返回第一个匹配到的字符的位置
str_locate(string = "banana",pattern = "a")
```

A matrix: 1 × 2 of type int

start	end
2	2

[42]:
```
# 返回所有匹配到的字符的位置
str_locate_all(string = "banana",pattern = "a")
```

1. A matrix: 3 × 2 of type int

start	end
2	2
4	4
6	6

4.9 str_match：字符匹配

字符匹配函数 str_match 和 str_match_all 与字符提取函数 str_extract 类似，返回匹配到的字符，不同之处在于返回格式。

[43]:
```
# 返回第一个匹配到的字符（矩阵）
str_match(string = "banana",pattern = "a")
```

A matrix: 1 × 1 of type chr a

[44]:
```
# 返回所有匹配到的字符（列表）
str_match_all(string = "banana",pattern = "a")
```

1. A matrix: 3 × 1 of type chr
   a
   a
   a

4.10 str_pad：字符补齐

字符补齐函数 str_pad，用于在字符串中添加单个字符，可选择添加的位置，在参数 side 中进行设置。

[45]:
```
# 默认字符串左边补齐
str_pad(string = "jack",width = 6,pad = "S")
```

'SSjack'

[46]:
```
# 字符串右边补齐
str_pad(string = "jack",width = 6,side = "right",pad = "S")
```

'jackSS'

[47]:
```
# 字符串两边补齐
str_pad(string = "jack",width = 6,side = "both",pad = "S")
```

'SjackS'

4.11 str_remove：字符删除

字符删除函数 str_remove 和 str_remove_all，用于删除字符串中的部分字符。

[48]:
```
# 删除第一个匹配到的字符
str_remove(string = "banana",pattern = "a")
```

'bnana'

[49]:
```
# 删除所有匹配到的字符
str_remove_all(string = "banana",pattern = "a")
```

'bnn'

4.12 str_replace：字符替换

字符替换函数 str_replace、str_replace_all 和 str_replace_na，用于替换字符串中的部分字符。

[50]:
```
# 替换第一个匹配到的字符
str_replace(string = "banana",pattern = "a",replacement = "A")
```

'bAnana'

[51]:
```
# 替换所有匹配到的字符
str_replace_all(string = "banana",pattern = "a",replacement = "A")
```

'bAnAnA'

[52]:
```
# NA 替换成 NA 字符
str_replace_na(string = c("banana",NA))
```

1. 'banana' 2. 'NA'

4.13 str_sort：字符排序

字符排序函数 str_sort 和 str_order，对字符向量进行排序。

[53]:
```
# 字符向量升序排序，返回字符向量
str_sort(c("sql","json","python"))
```

1. 'json' 2. 'python' 3. 'sql'

[54]:
```
# 字符向量降序排序，返回字符向量
str_sort(c("sql","json","python"),decreasing = TRUE)
```

1. 'sql' 2. 'python' 3. 'json'

[55]:
```
# 字符向量升序排序，返回索引向量
str_order(c("sql","json","pythn"))
```

1. 2 2. 3 3. 1

4.14 str_split：字符分割

字符分割函数 str_split 和 str_split_fixed，对字符串进行分割。

[56]:
```
# 字符分割，返回列表
str_split(string = "banana",pattern = "")
```

1. (a) 'b' (b) 'a' (c) 'n' (d) 'a' (e) 'n' (f) 'a'

[57]:
```
# 字符分割，返回矩阵
str_split(string = "banana",pattern = "",simplify = T)
```

A matrix: 1 × 6 of type chr b a n a n a

[58]:
```
# 字符分割，指定分割块数
str_split_fixed(string = "banana",pattern = "",n = 3)
```

A matrix: 1 × 3 of type chr b a nana

4.15 str_sub：字符过滤

字符过滤函数 str_sub 和 str_subset，str_sub 函数通过指定开始和结束位置，过滤出字符串的部分字符串。str_subset 函数通过匹配模式，过滤出满足模式的字符串。

[59]:
```
# 字符过滤（正向索引）
str_sub(string = "banana",start = 1,end = 3)
```

'ban'

[60]:
```
# 字符过滤（反向索引）
str_sub(string = "banana",start = -2,end = -1)
```

'na'

[61]:
```
# 字符过滤，并赋值
x <- "banana"
str_sub(string = x,start = 1,end = 1) <- "A"
print(x)
```

```
[1] "Aanana"
```

[62]:
```
# 字符串过滤（返回字符串）
str_subset(string = c("java","sql","python"),pattern = "^s")
```

'sql'

[63]:
```
# 字符串过滤（返回位置）
str_which(string = c("java","sql","python"),pattern = "^s")
```

2

4.16 其他

stringr 包中其他的有用函数，用于常见的字符处理。

[64]:
```
# 删除字符串两边的空格
str_trim(string = "  you are beautiful!  ")
```

'you are beautiful!'

[65]:
```
# 删除字符串中多余的空格
str_squish(string = "  you are  beautiful!  ")
```

'you are beautiful!'

[66]:
```
# 字符转为小写
dog <- "The quick brown dog"
str_to_lower(dog)
```

'the quick brown dog'

[67]:
```
# 字符转为大写
str_to_upper(dog)
```

'THE QUICK BROWN DOG'

[68]:
```
# 字符转为标题
str_to_title(dog)
```

'The Quick Brown Dog'

[69]:
```
# 字符转为语句
str_to_sentence(dog)
```

'The quick brown dog'

5 ggplot2

[70]:
```
dat1 <- data.frame(sex = factor(c("Female", "Female", "Male", "Male")), time =␣
↪factor(c("Lunch",
    "Dinner", "Lunch", "Dinner"), levels = c("Lunch", "Dinner")), total_bill =␣
↪c(13.53,
    16.81, 16.24, 17.42))
dat1
```

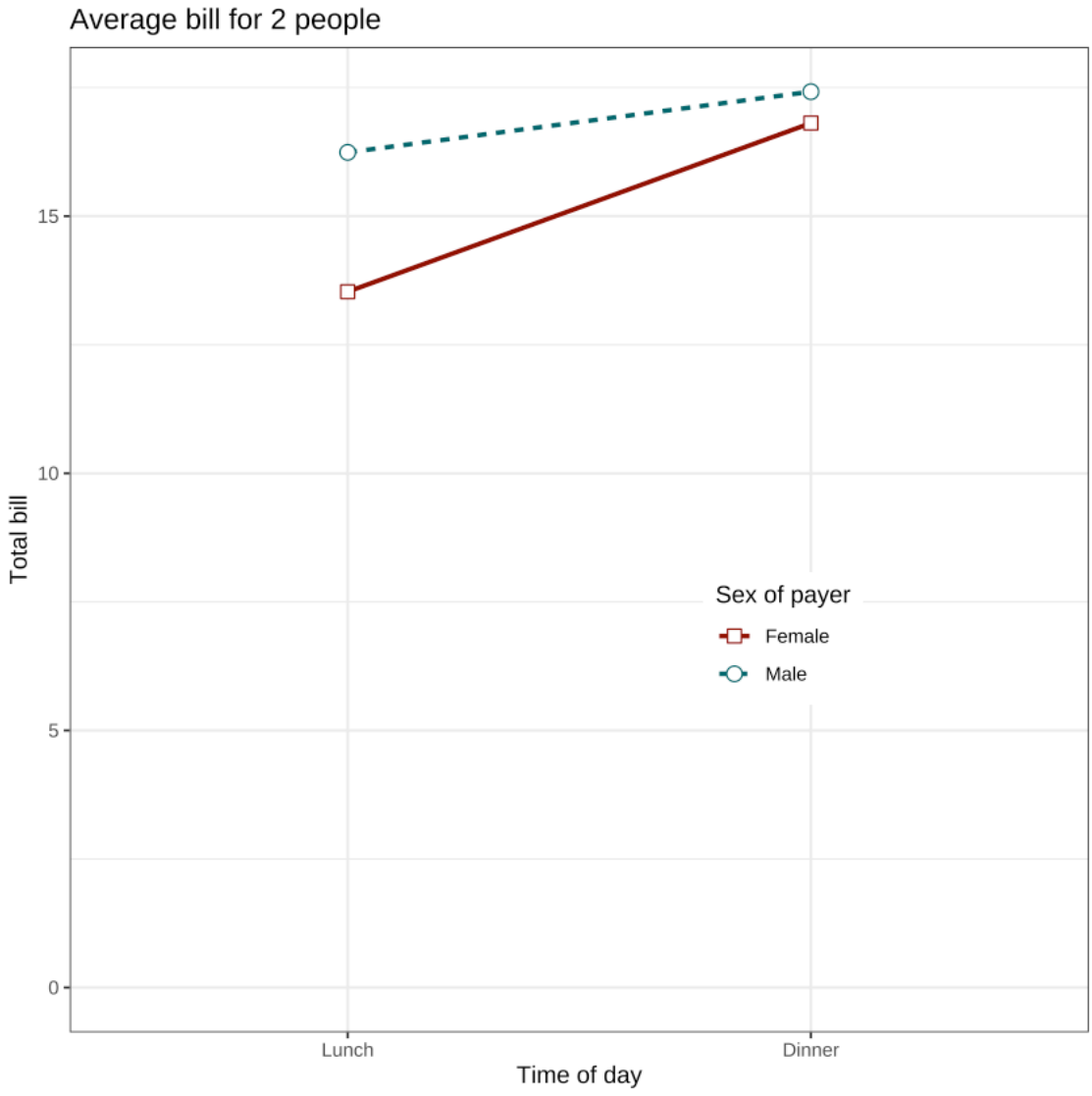
A data.frame: 4 × 3

sex	time	total_bill
<fct>	<fct>	<dbl>
Female	Lunch	13.53
Female	Dinner	16.81
Male	Lunch	16.24
Male	Dinner	17.42

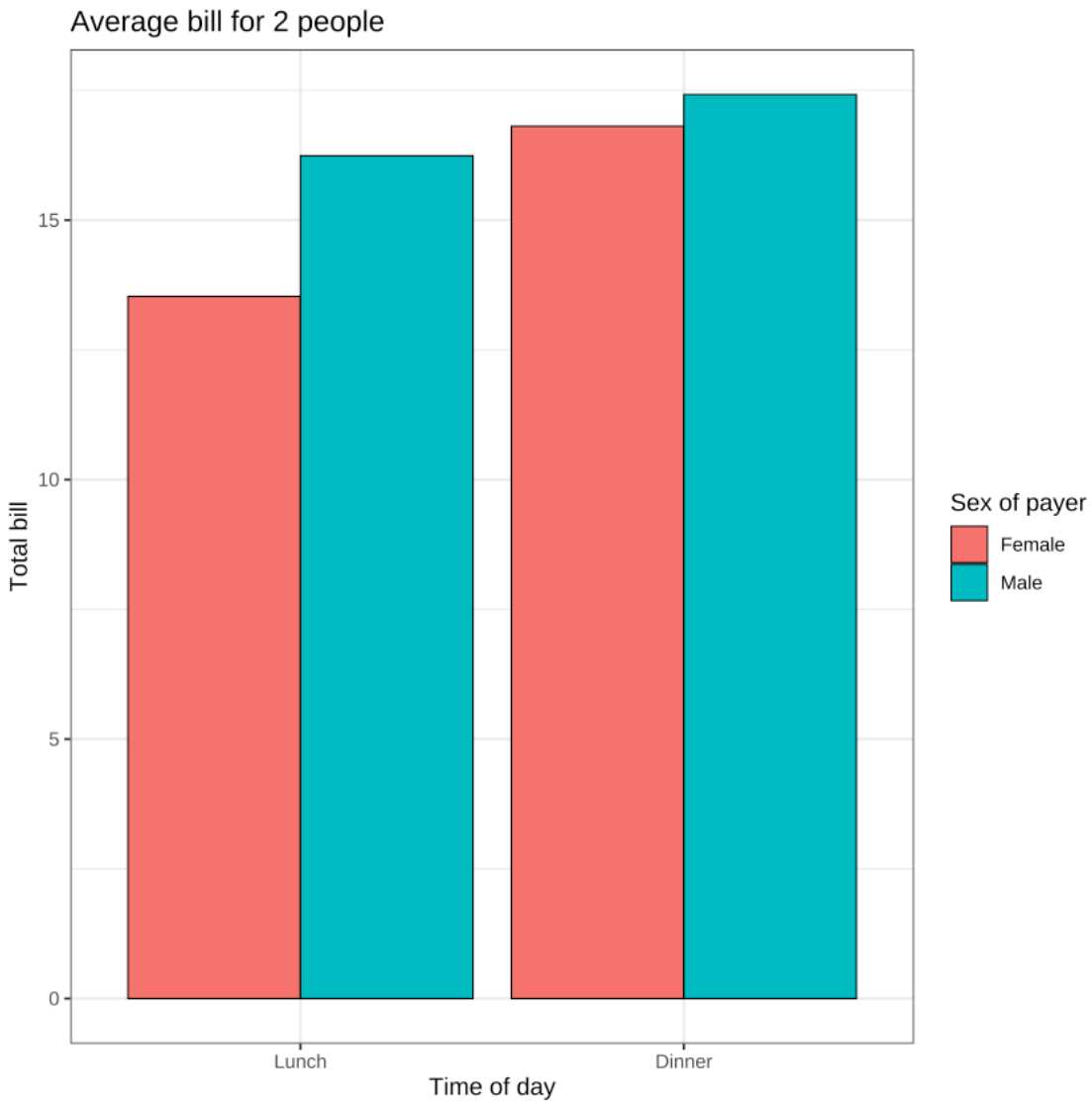
[71]:
```
# 一个线图
ggplot(data=dat1, aes(x=time, y=total_bill, group=sex, shape=sex, colour=sex))␣
↪+
    geom_line(aes(linetype=sex), size=1) +      # 按性别设定线型
    geom_point(size=3, fill="white") +          # 使用更大的点，并用颜色填充
    expand_limits(y=0) +                        # 将 0 包含仅 y 轴
    scale_colour_hue(name="Sex of payer",       # 设定图例标题
                     l=30)  +                   # 使用更深的颜色 (lightness=30)
    scale_shape_manual(name="Sex of payer",
```

```
                      values=c(22,21)) +      #
  scale_linetype_discrete(name="Sex of payer") +
  xlab("Time of day") + ylab("Total bill") + # 设定轴标签
  ggtitle("Average bill for 2 people") +     # 设定标题
  theme_bw() +
  theme(legend.position=c(.7, .4))           # 图例的位置
```

```
Warning message:
 "Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
  Please use `linewidth` instead."
```

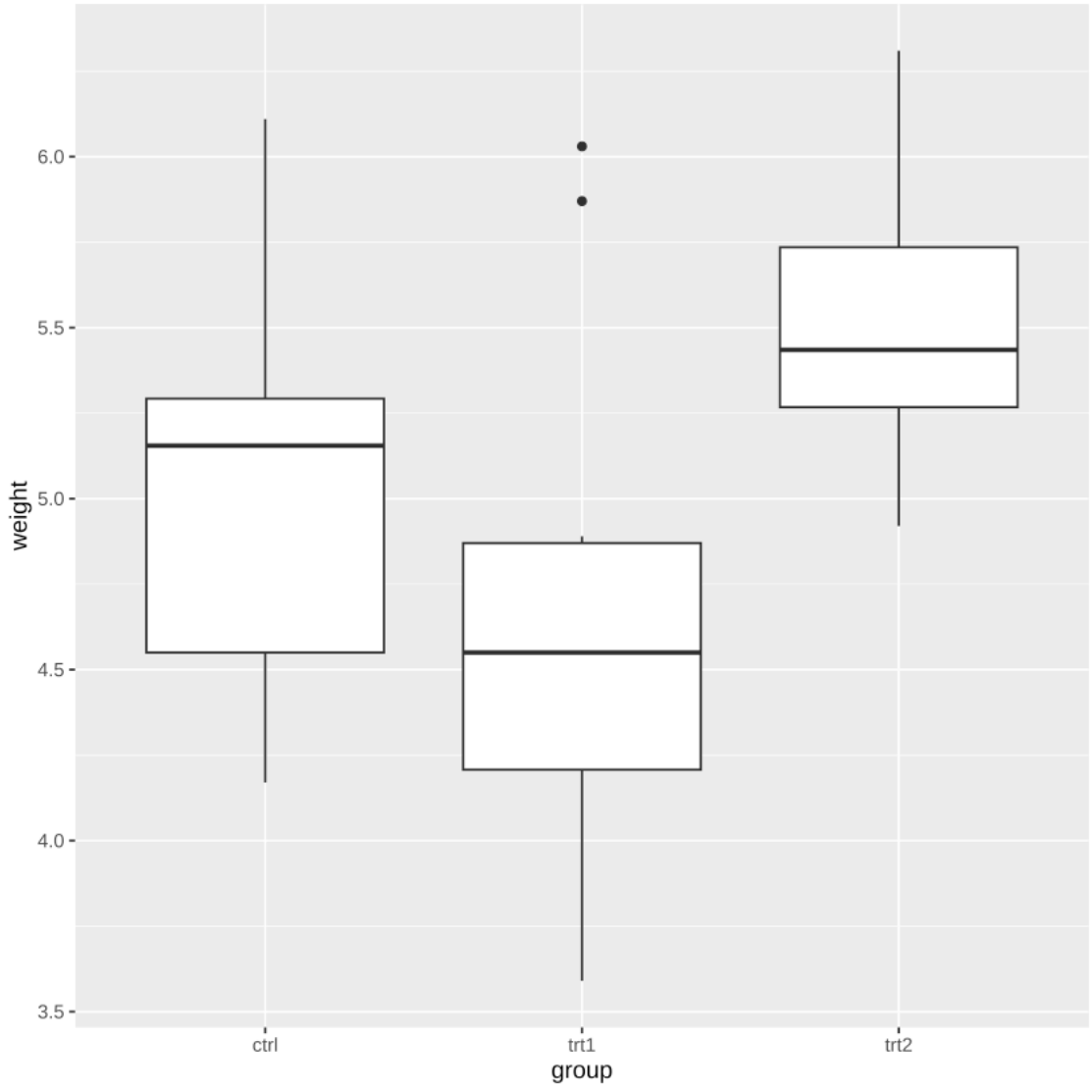


[72]:
```
# 一个条形图
ggplot(data=dat1, aes(x=time, y=total_bill, fill=sex)) +
    geom_bar(colour="black", stat="identity",
             position=position_dodge(),
             linewidth=.3) +                          # 更粗的线
    scale_fill_hue(name="Sex of payer") +      # 设定图例标题
    xlab("Time of day") + ylab("Total bill") + # 设定轴标签
    ggtitle("Average bill for 2 people") +     # 设定题目
    theme_bw()
```



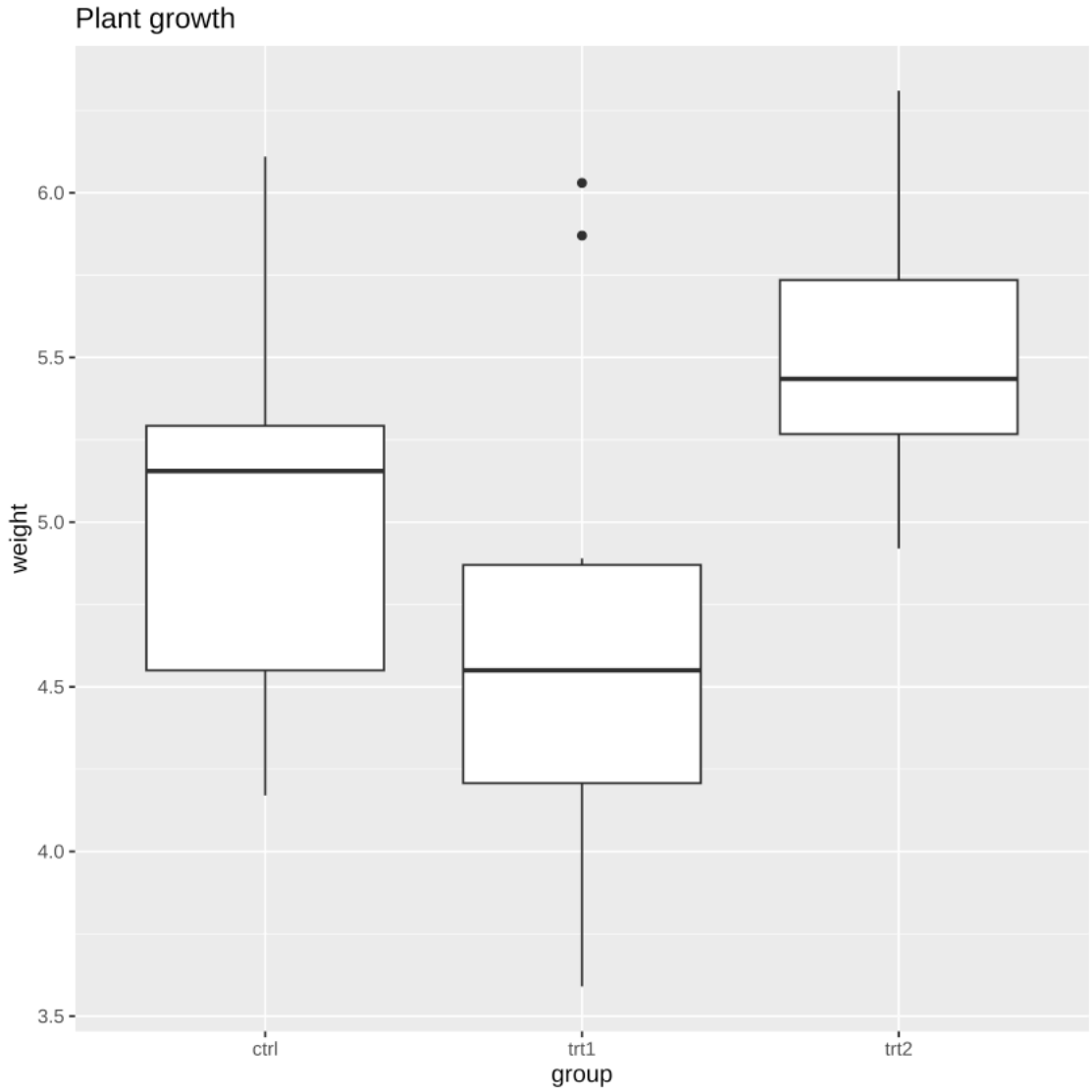
5.1 基本语法

[73]:
```
bp <- ggplot(PlantGrowth, aes(x = group, y = weight)) +
  geom_boxplot()
bp
```

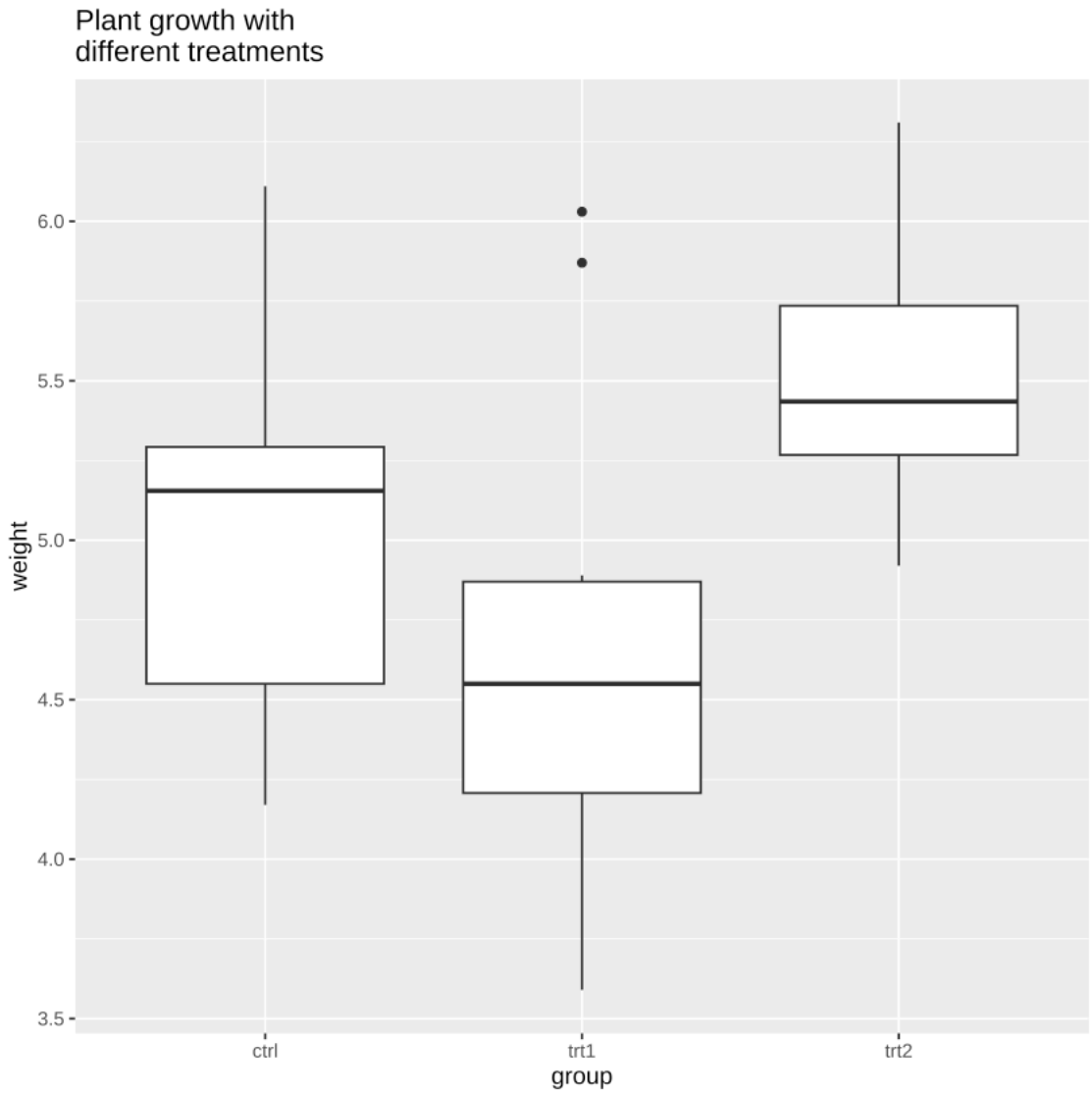


5.1.1 标题

[74]:
```
bp + ggtitle("Plant growth")
```

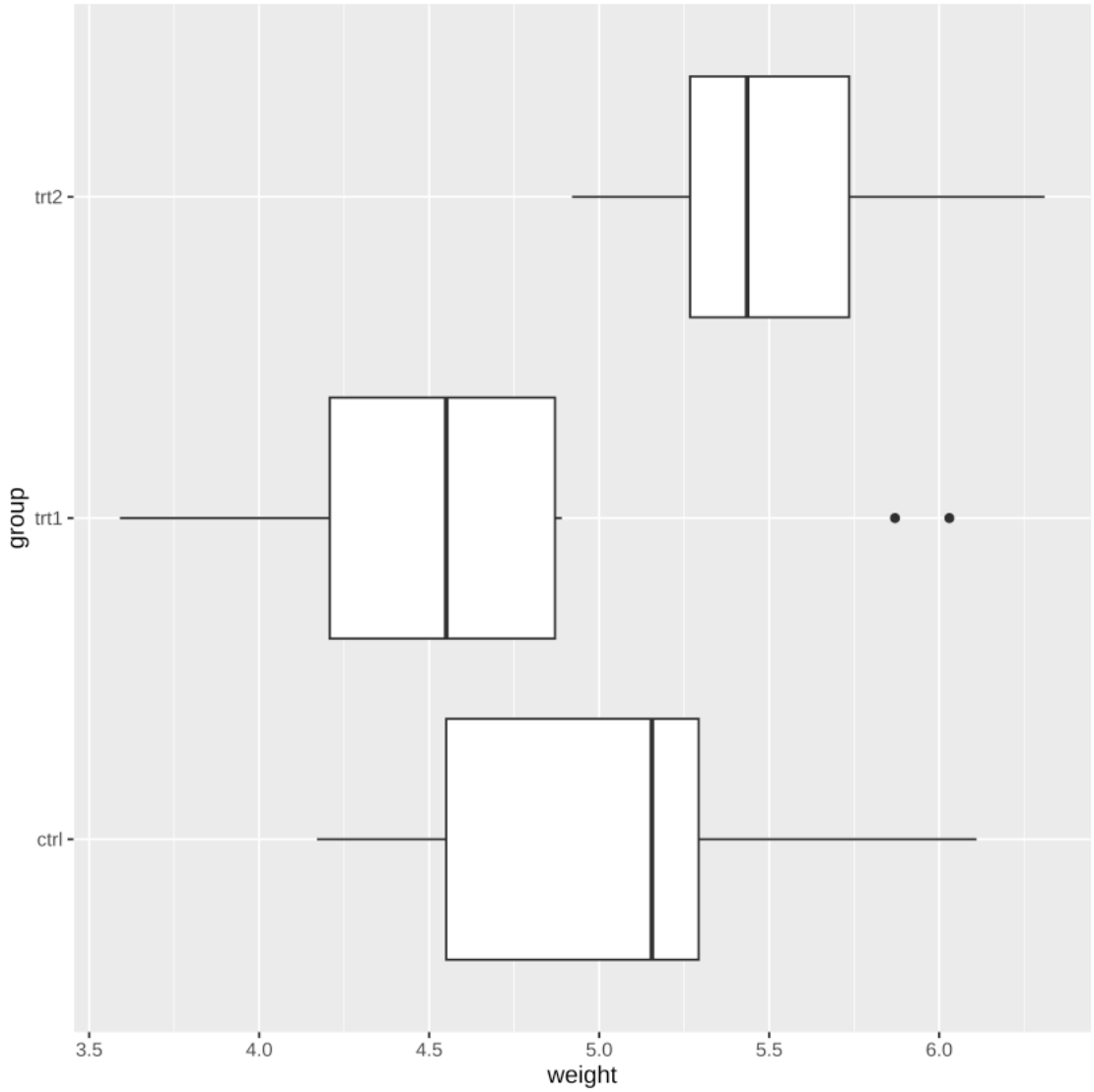


[75]:
```
## 等同于 bp + labs(title='Plant growth')
## 如果标题比较长，可以用 \n 将它分成多行来显示
bp + ggtitle("Plant growth with\ndifferent treatments")
```



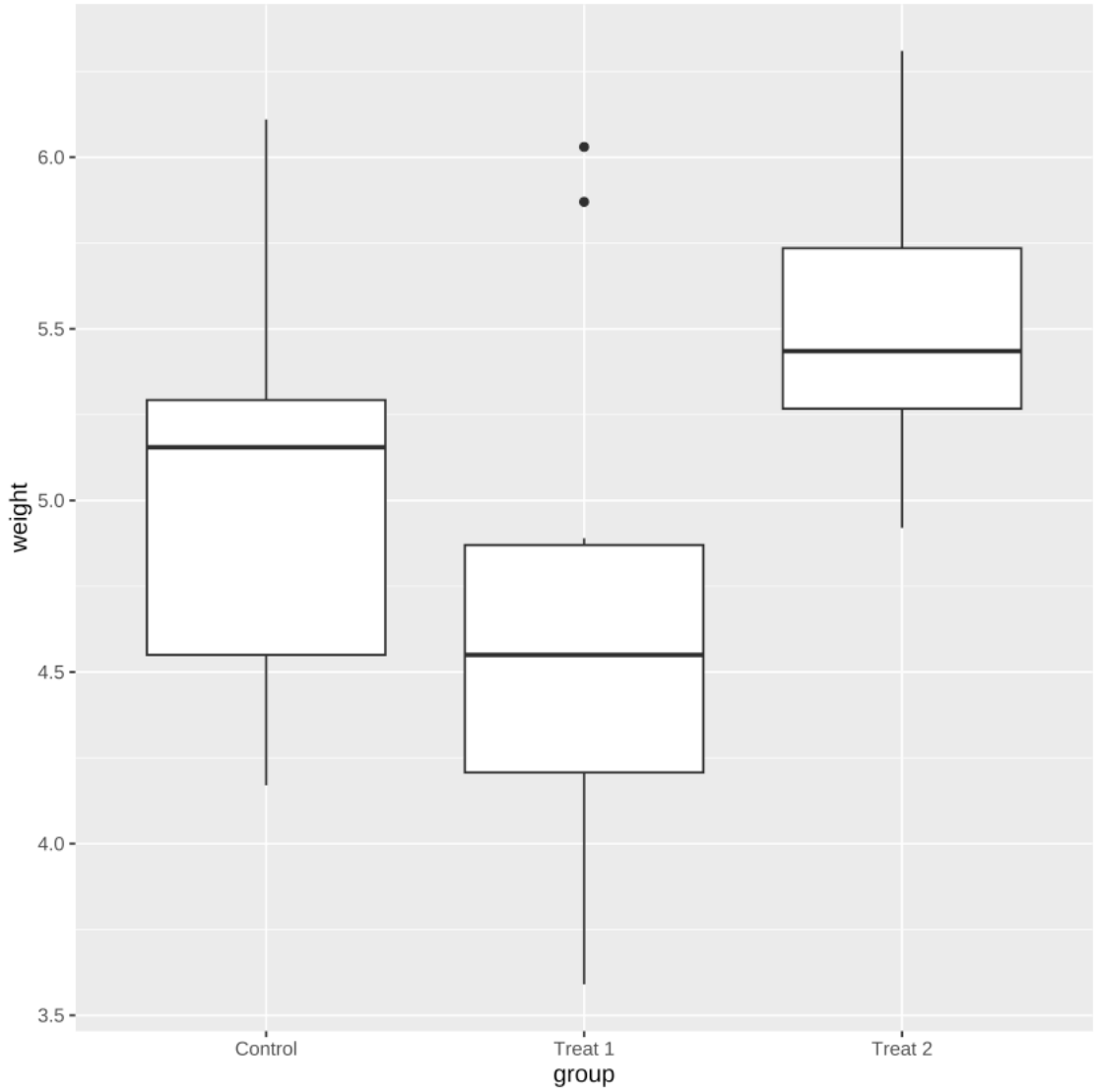
5.1.2 交换 x 和 y 轴

[76]:
```
# 交换 x 和 y 轴（让 x 垂直、y 水平）。
bp + coord_flip()
```



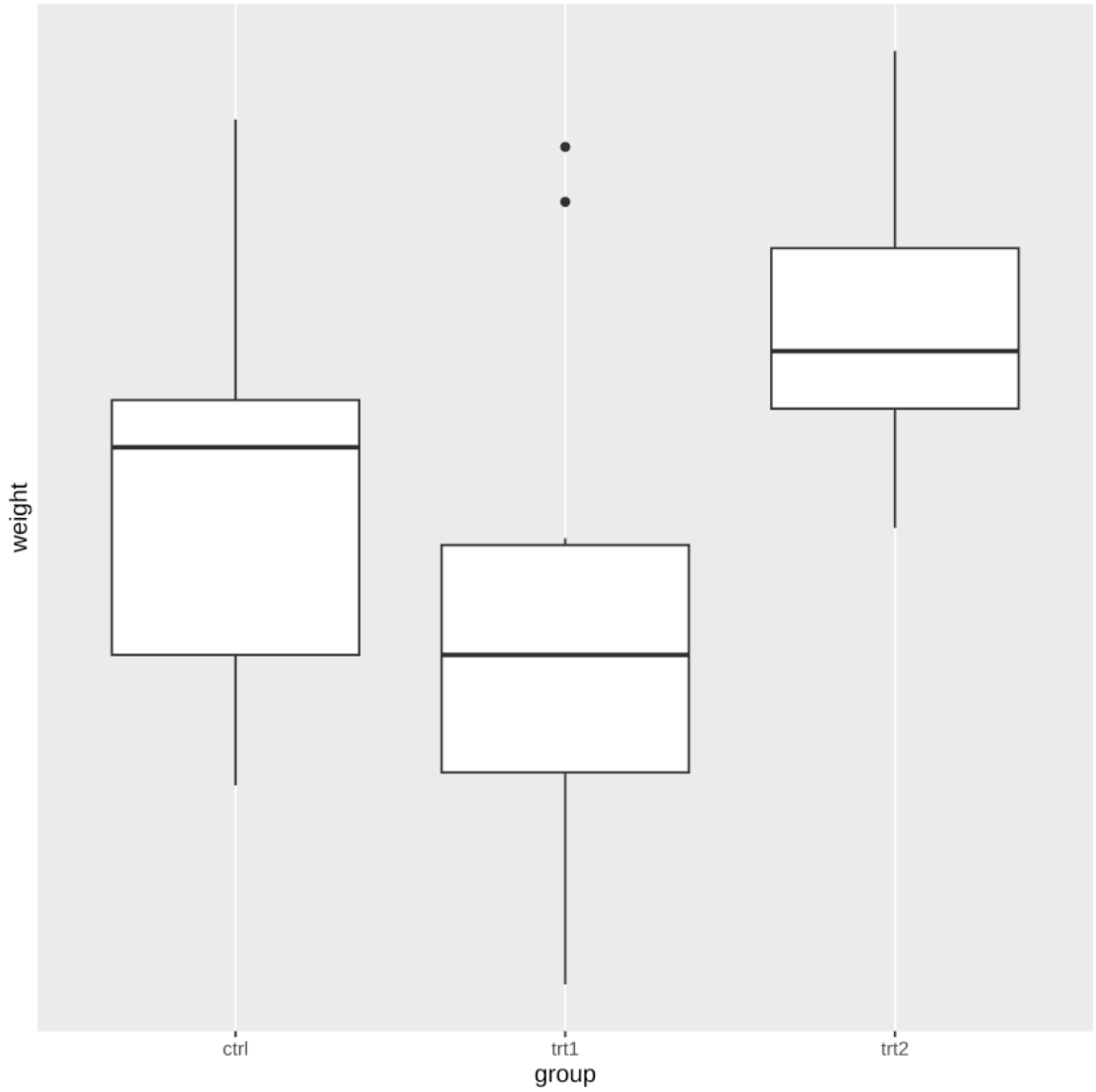
5.1.3 Label

[77]:
```
bp + scale_x_discrete(breaks = c("ctrl", "trt1", "trt2"), labels = c("Control",␣
↪"Treat 1",
   "Treat 2"))
```

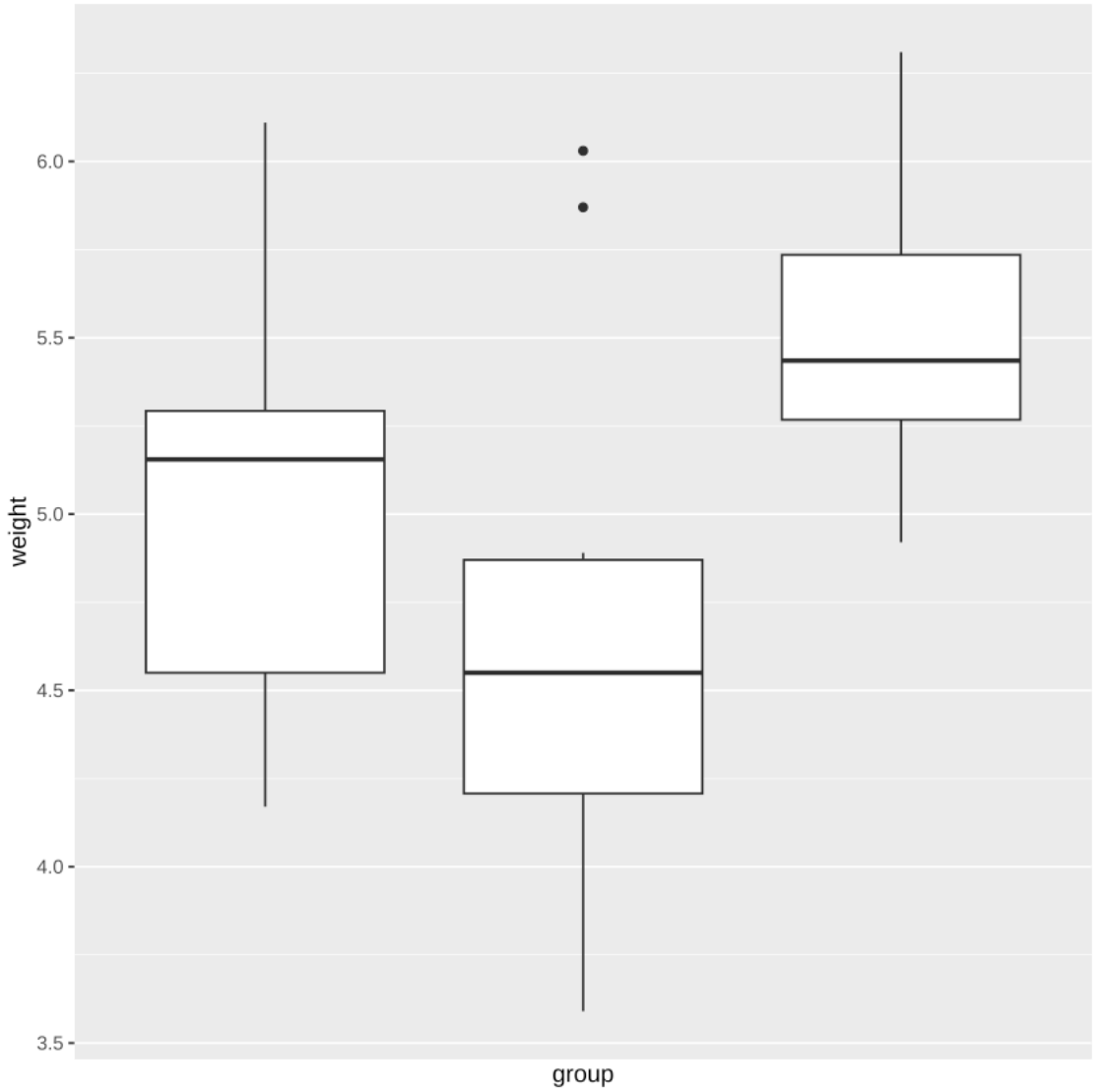


5.1.4 隐藏元素

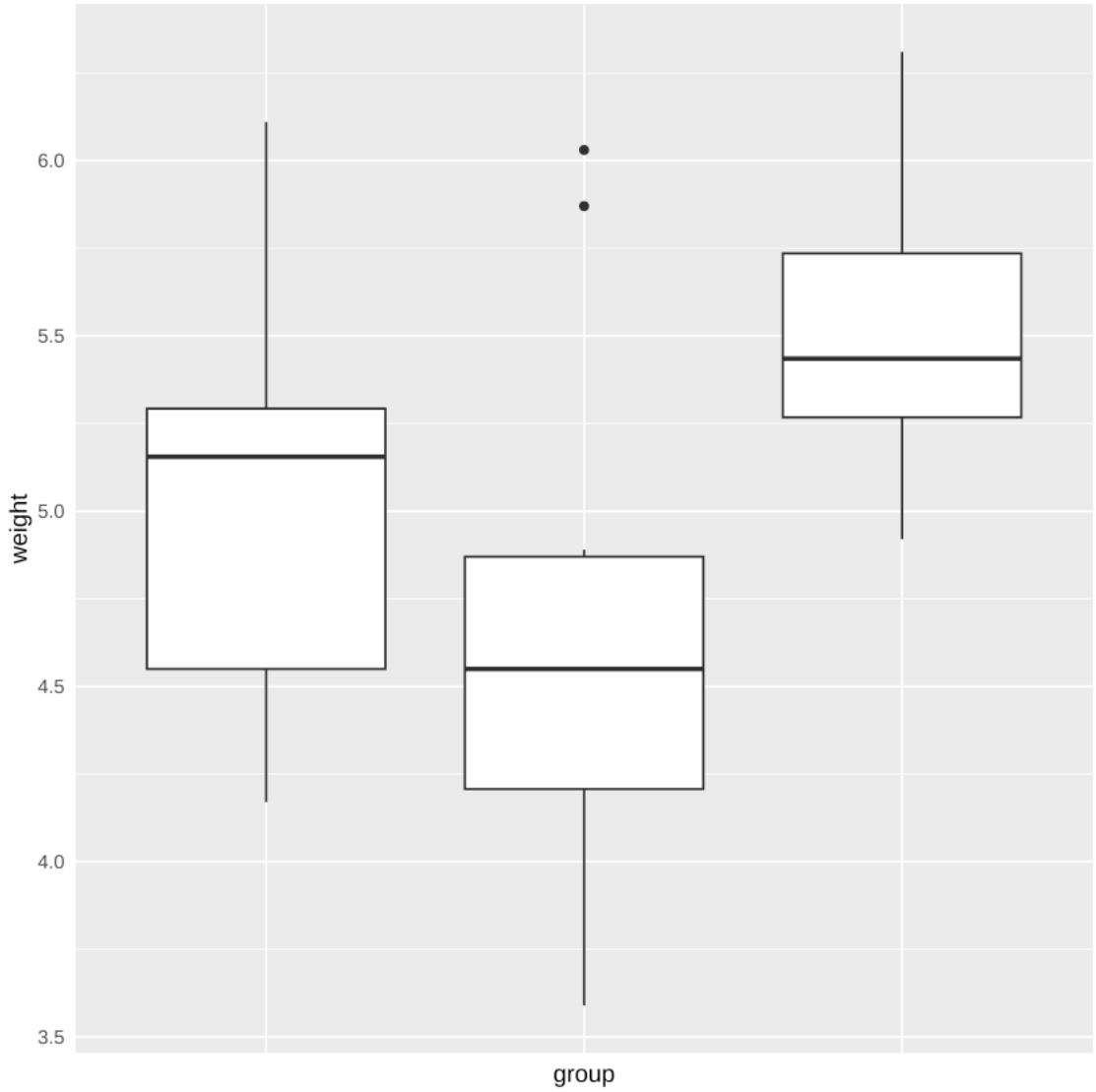
[78]:
```
# 抑制标签和网格线
bp + scale_y_continuous(breaks = NULL)
```



[79]:
```
# 隐藏 x 刻度、标签和网格线
bp + scale_x_discrete(breaks = NULL)
```



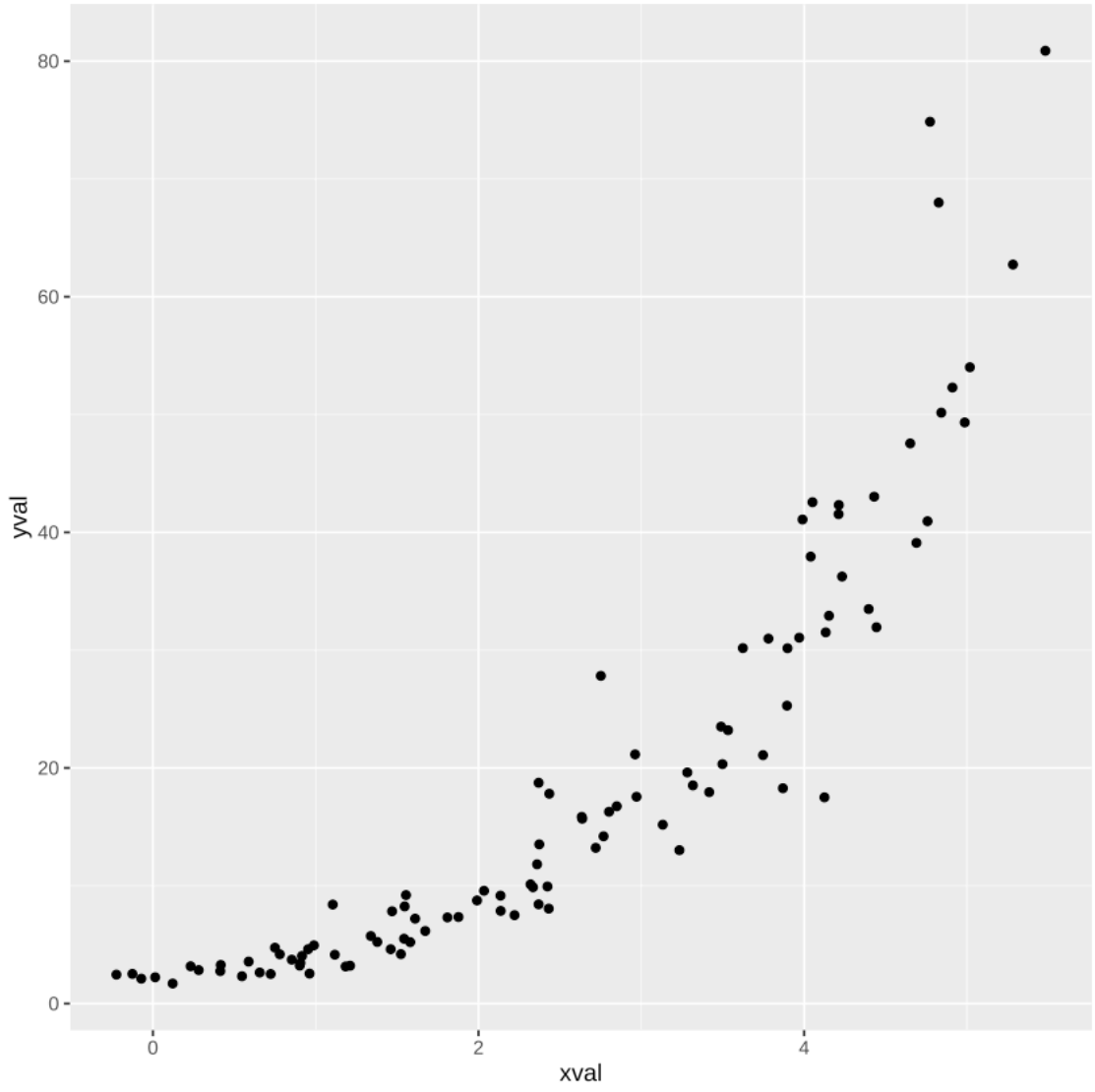
[80]:
```
# 隐藏所有的刻度和标签 (X 轴), 保留网格线
bp + theme(axis.ticks = element_blank(), axis.text.x = element_blank())
```



5.1.5 对数坐标轴

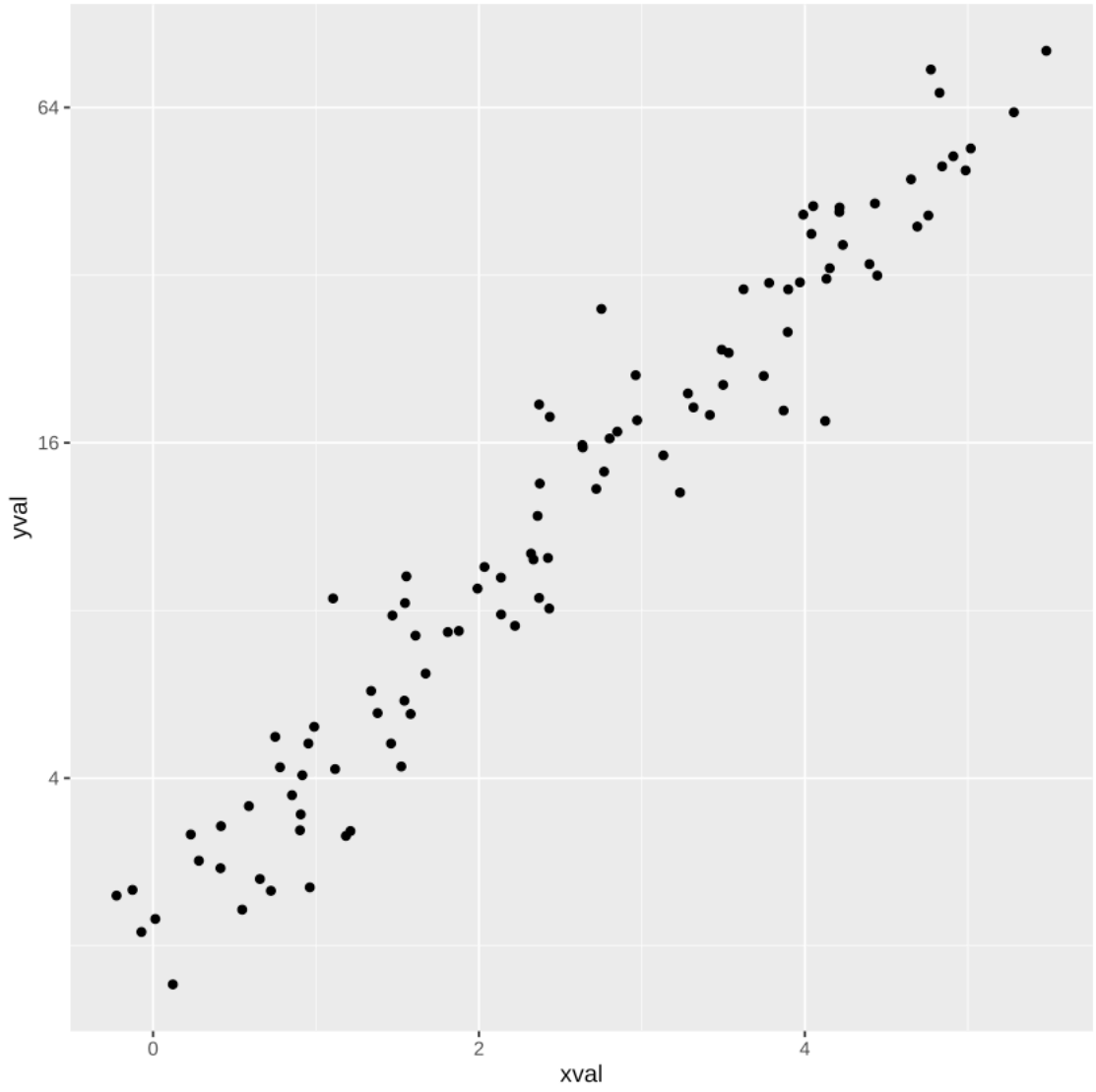
[81]:
```
# 创建指数分布数据
set.seed(201)
n <- 100
dat <- data.frame(xval = (1:n + rnorm(n, sd = 5))/20, yval = 2 * 2^((1:n +␣
 ↪rnorm(n,
    sd = 5))/20))
```

```
# 创建常规的散点图
sp <- ggplot(dat, aes(xval, yval)) + geom_point()
sp
```

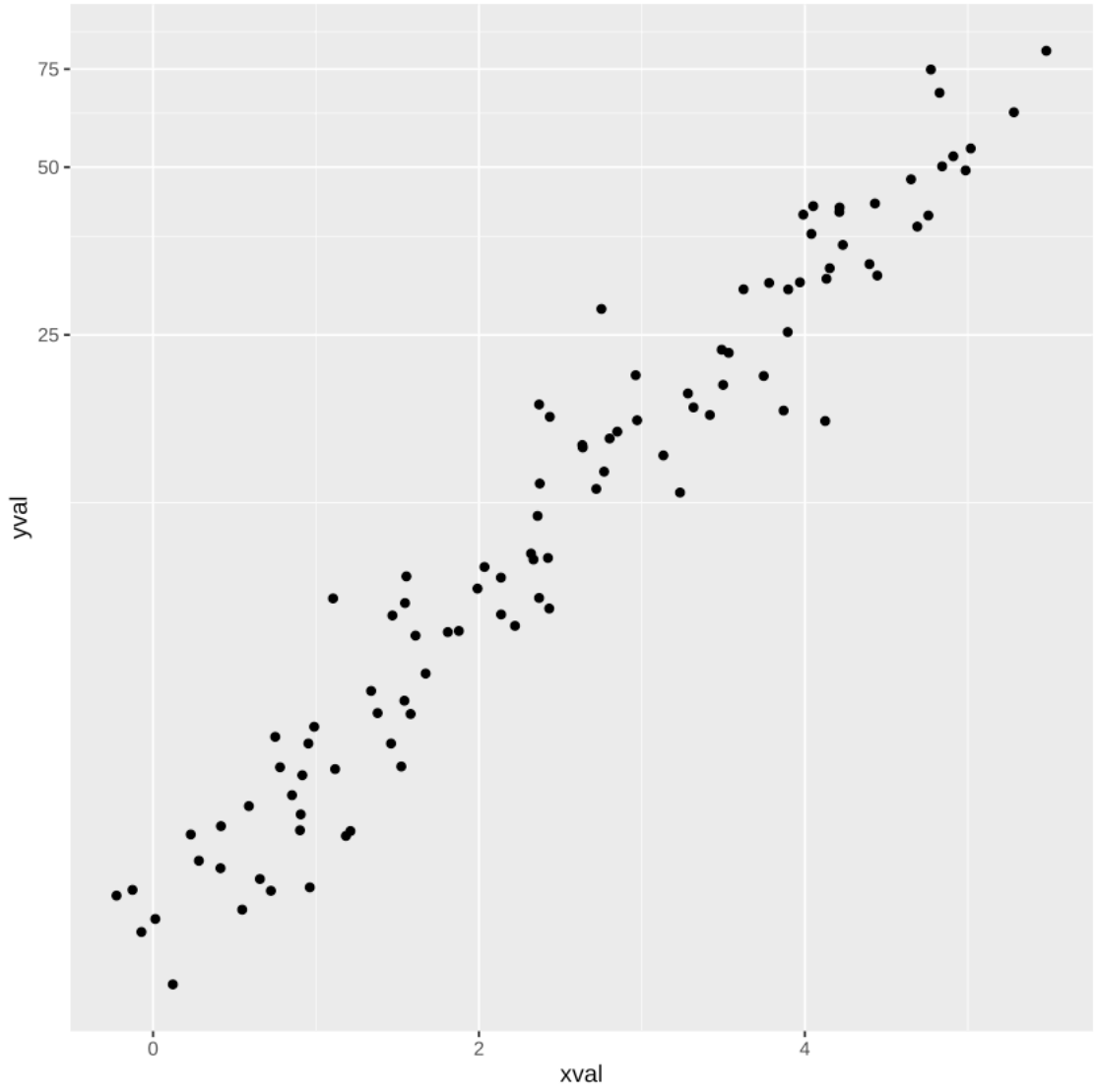


[82]:
```
# log2 比例化 (间隔相等)
library(scales)  # 需要 scales 包
sp + scale_y_continuous(trans = log2_trans())
```

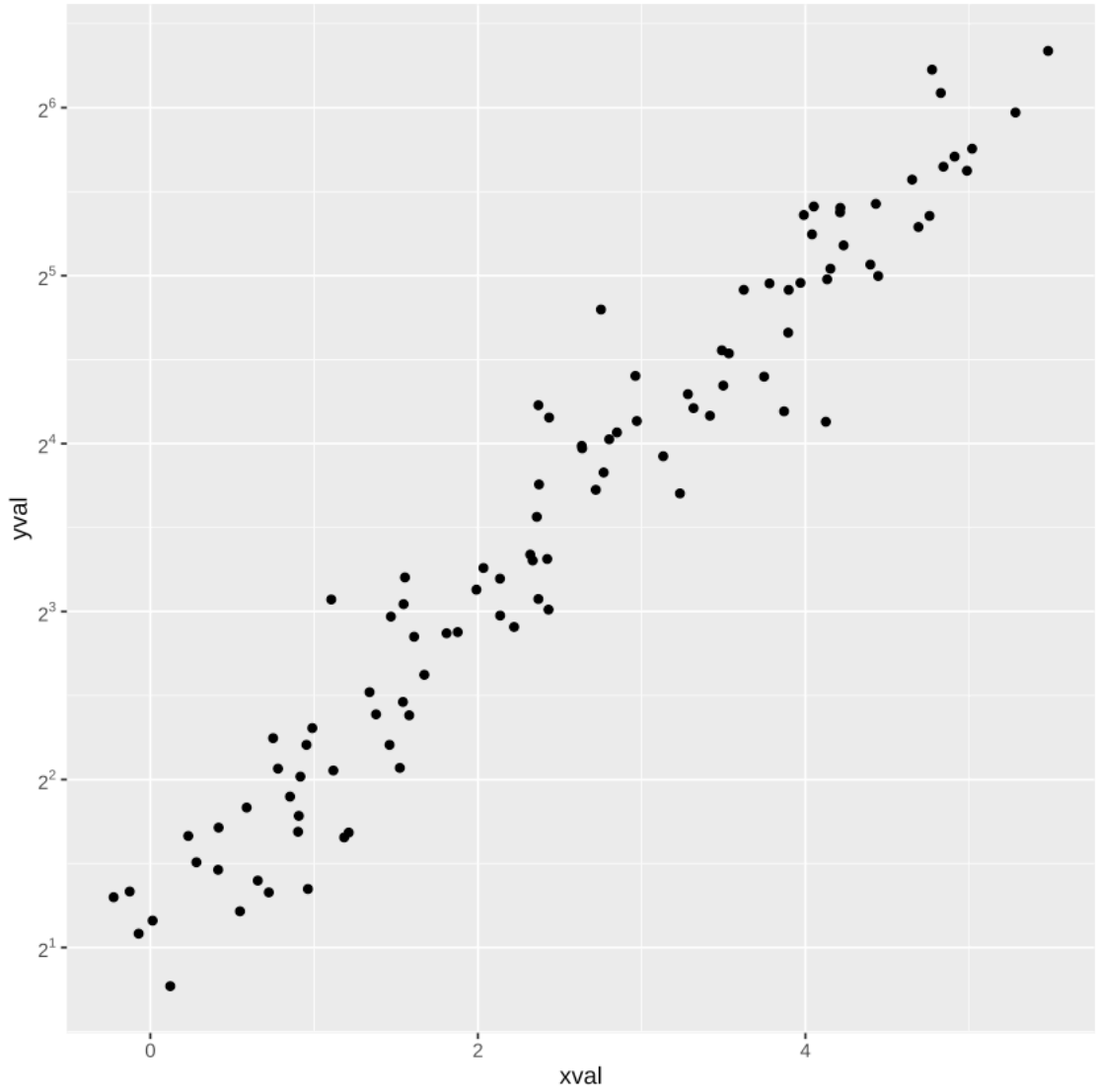
```
载入程辑包: 'scales'


The following object is masked from 'package:purrr':

    discard


The following object is masked from 'package:readr':

    col_factor
```



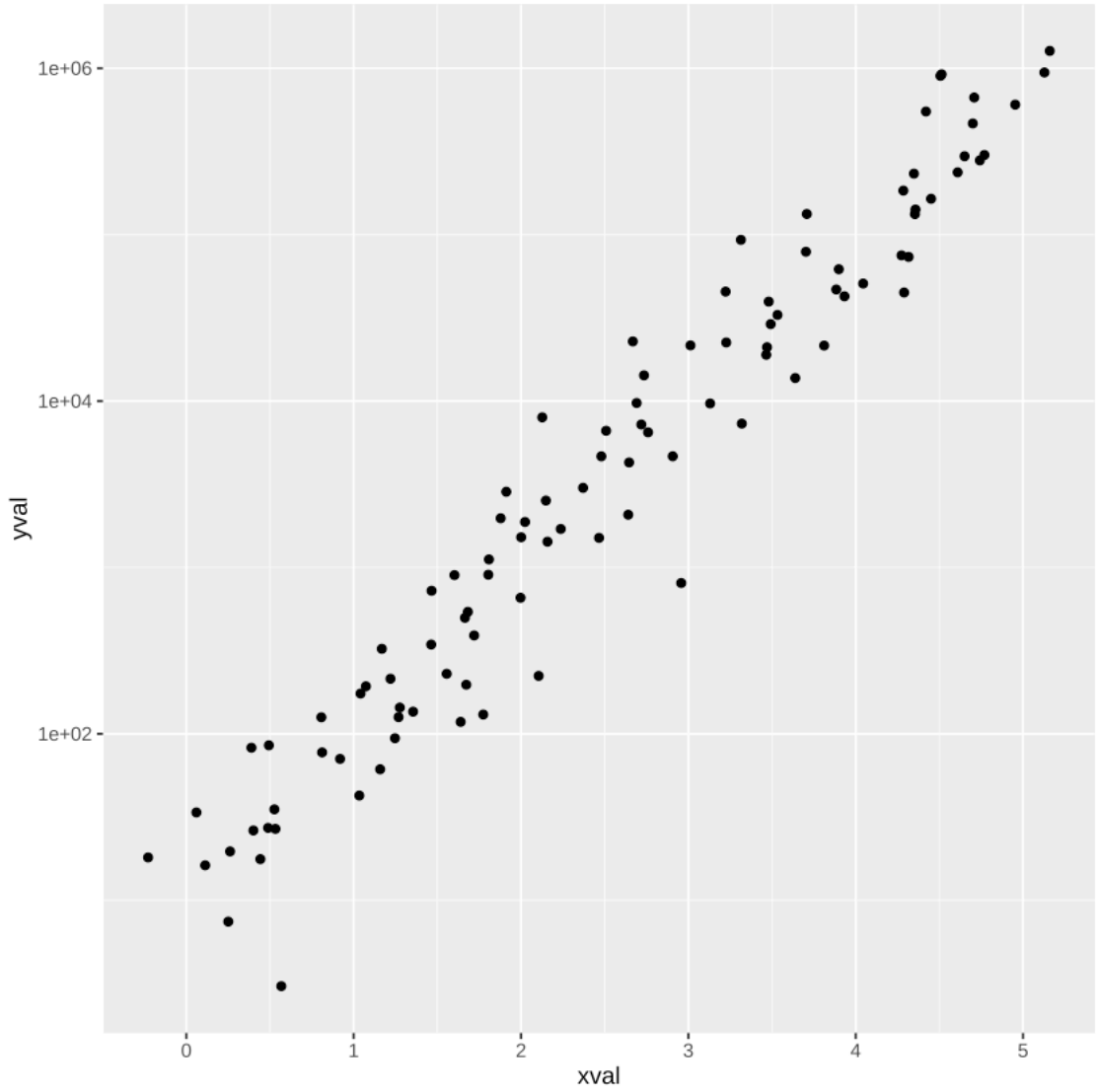
[83]:
```
# log2 坐标转换，空间间隔不同
sp + coord_trans(y = "log2")
```



[84]:
```
sp + scale_y_continuous(trans = log2_trans(), breaks = trans_breaks("log2",
  function(x) 2^x), labels = trans_format("log2", math_format(2^.x)))
```



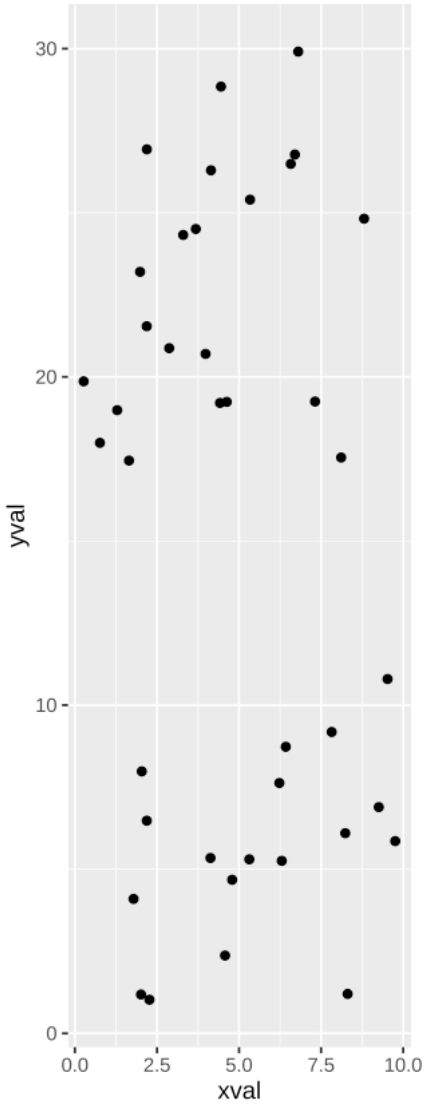
[85]:
```
set.seed(205)
n <- 100
dat10 <- data.frame(xval = (1:n + rnorm(n, sd = 5))/20,
  yval = 10 * 10^((1:n + rnorm(n, sd = 5))/20))

sp10 <- ggplot(dat10, aes(xval, yval)) + geom_point()

# log10
sp10 + scale_y_log10()
```



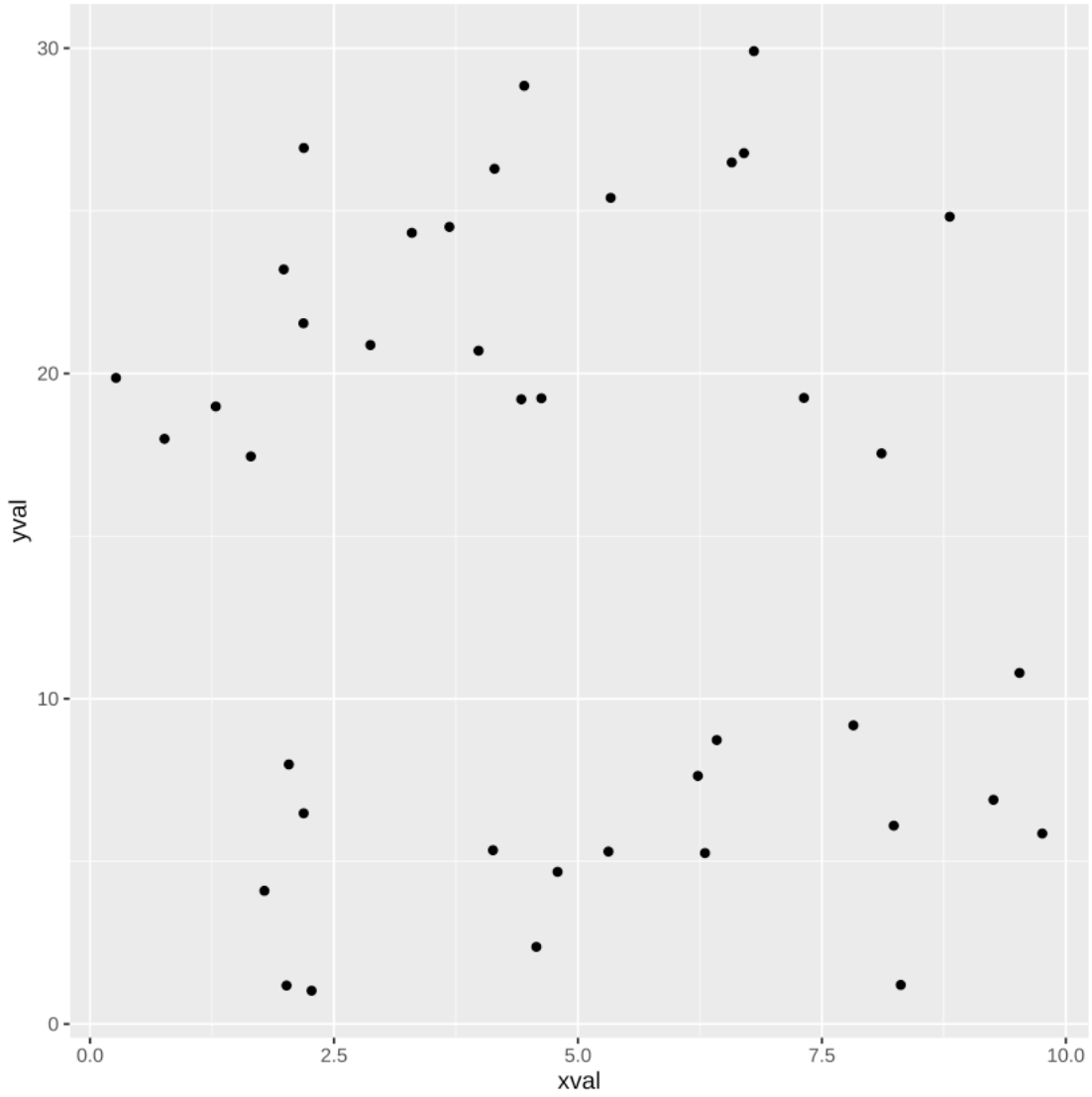
5.1.6 标度变化

[86]:
```
# x 范围 0-10, y 范围 0-30
set.seed(202)
dat <- data.frame(xval = runif(40, 0, 10), yval = runif(40, 0, 30))
sp <- ggplot(dat, aes(xval, yval)) + geom_point()

# 强制比例相等
```

```
sp + coord_fixed()
```



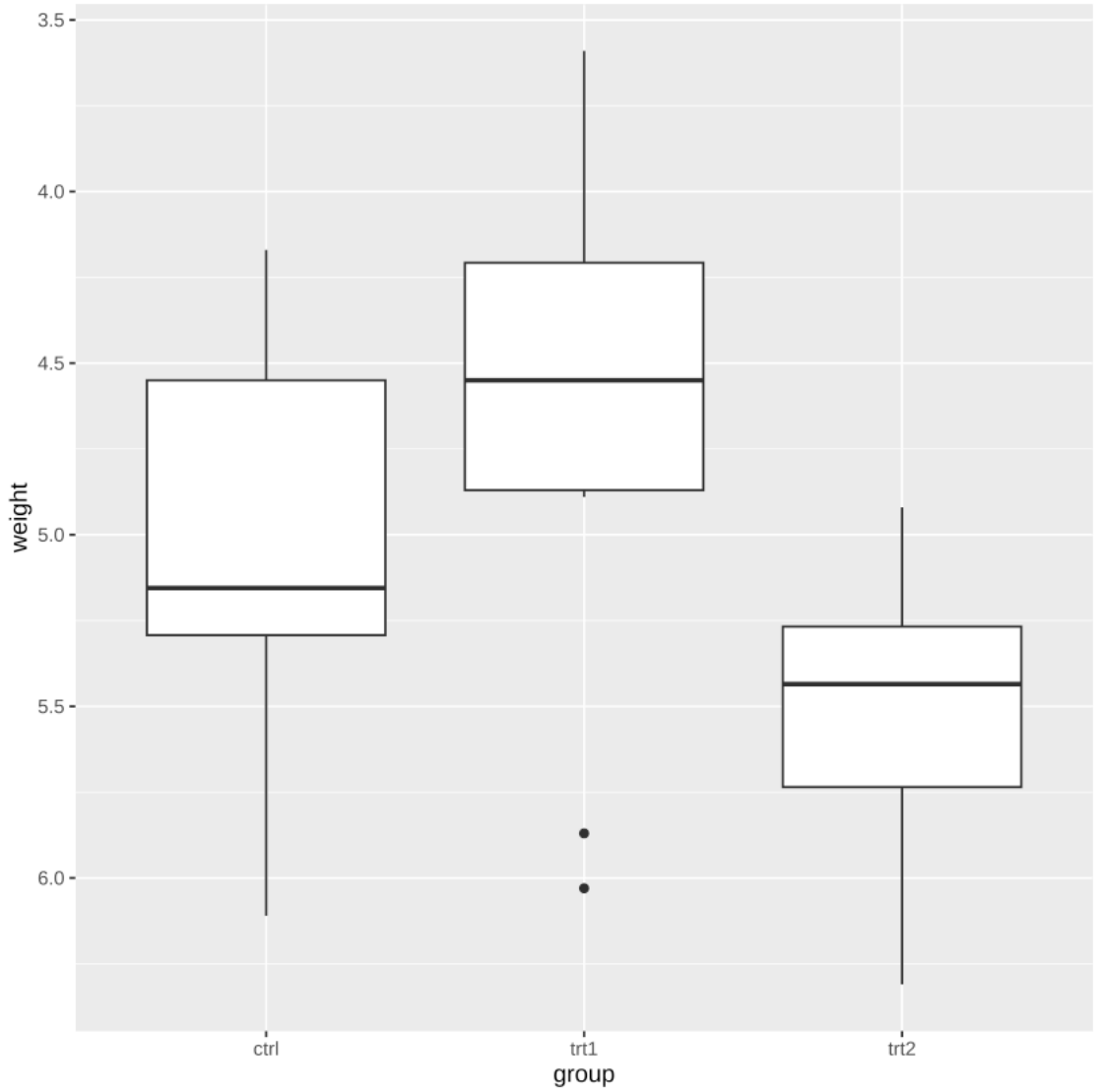
[87]:
```
# 相等的标度变化，让 x 的 1 个单位等同 y 的 3 个单位
sp + coord_fixed(ratio = 1/3)
```



[ ]:

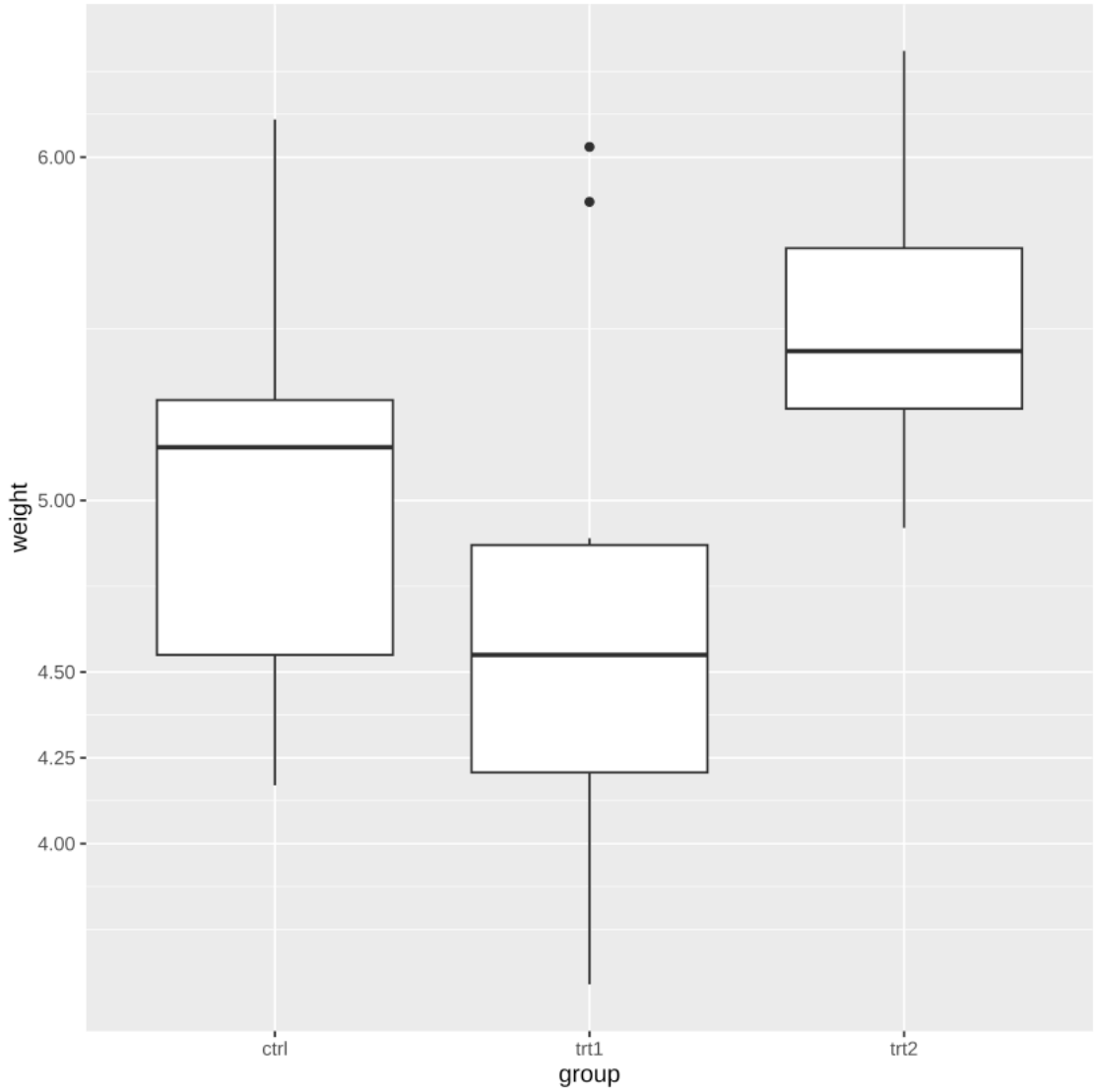
5.1.7 反转轴向

[88]:
```
# 反转一个连续值轴的方向
bp + scale_y_reverse()
```



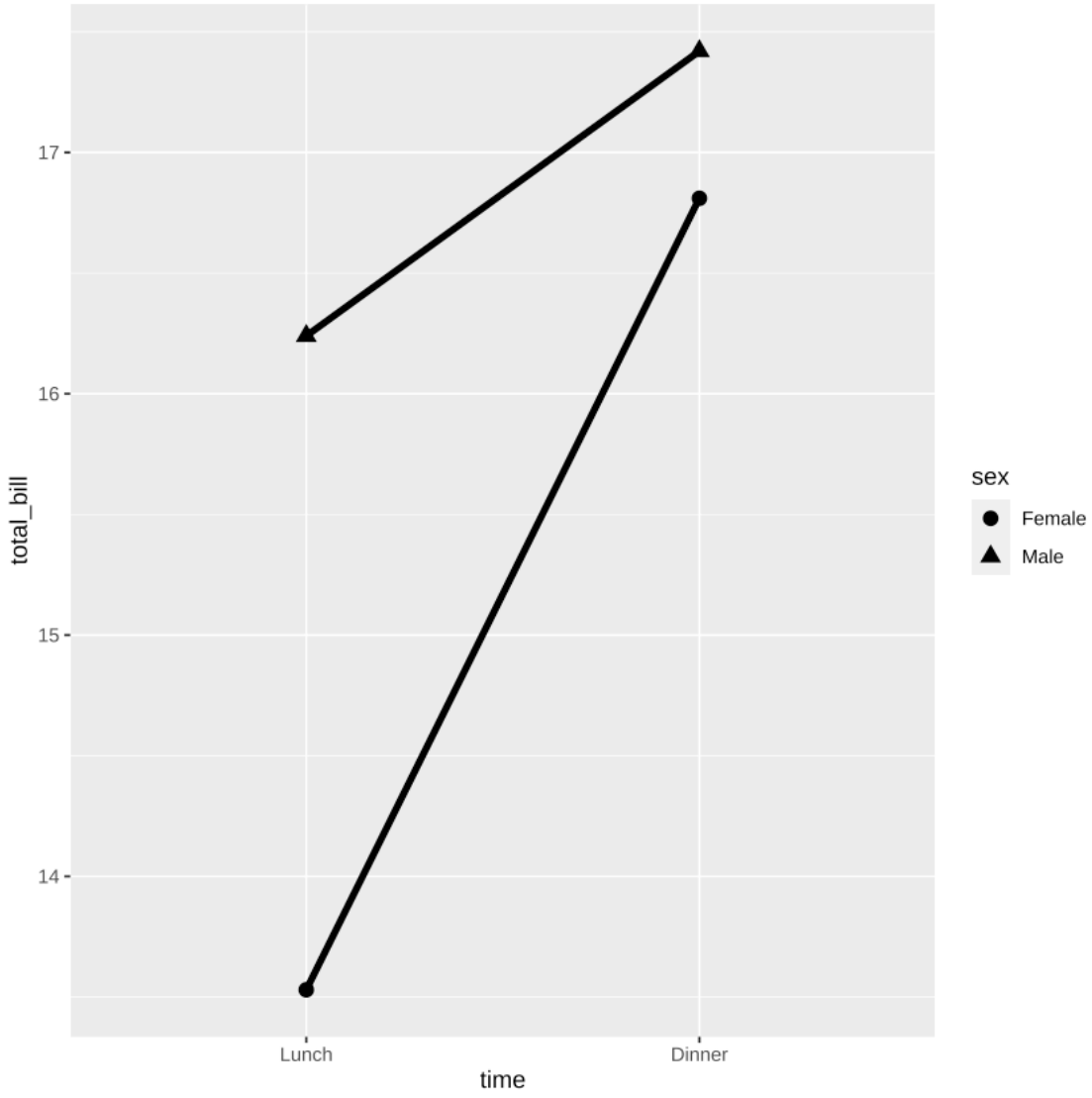
5.1.8 刻度

[89]:
```
# 刻度不平等变化
bp + scale_y_continuous(breaks = c(4, 4.25, 4.5, 5, 6, 8))
```



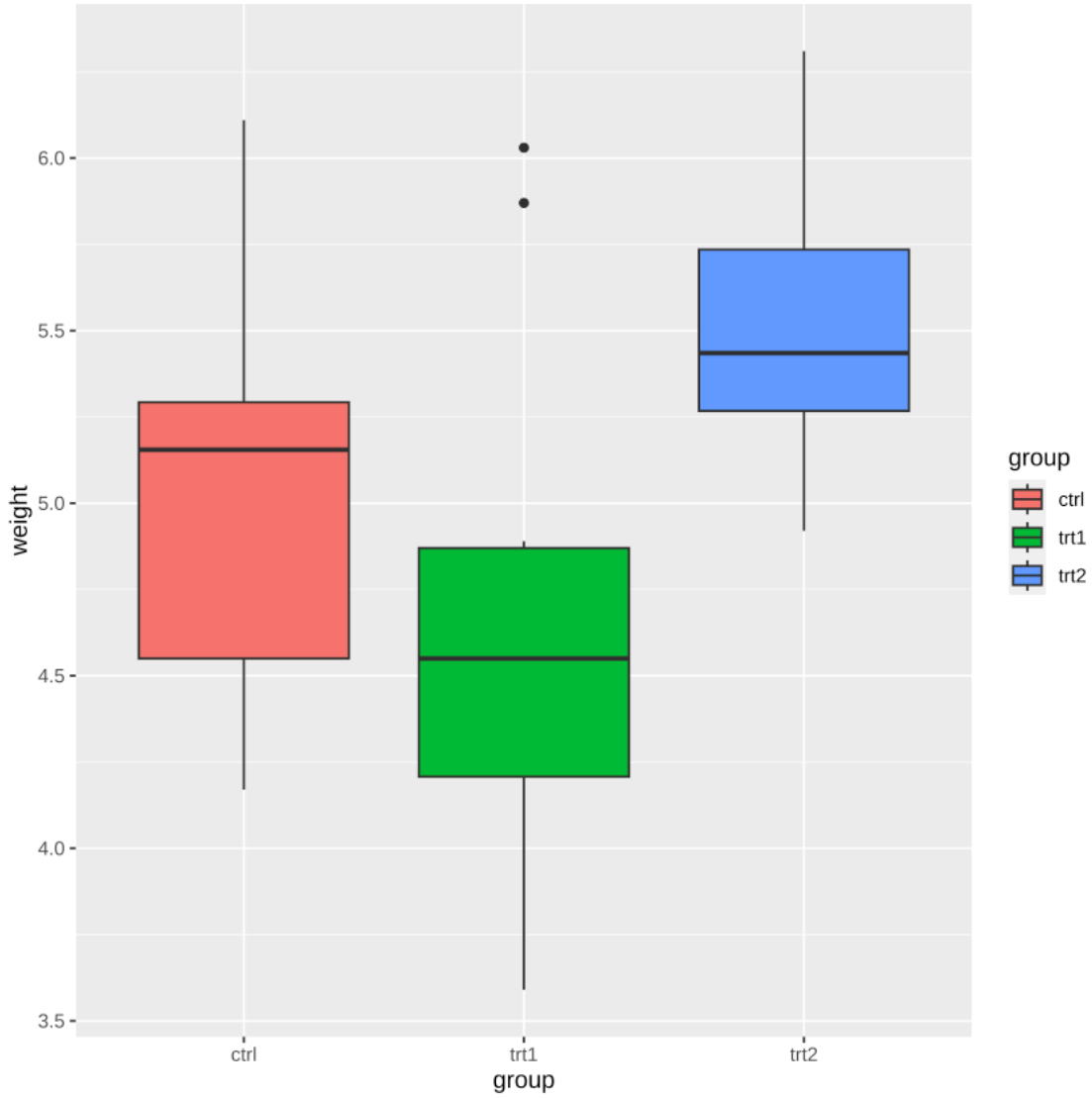
5.1.9 映射到颜色

[90]:
```
# 将性别映射到颜色
ggplot(data = dat1, aes(x = time, y = total_bill, group = sex,
 shape = sex)) +            # 映射到形状
 geom_line(linewidth=1.5) + # 设定线宽
 geom_point(size=3)         # 设定点大小
```



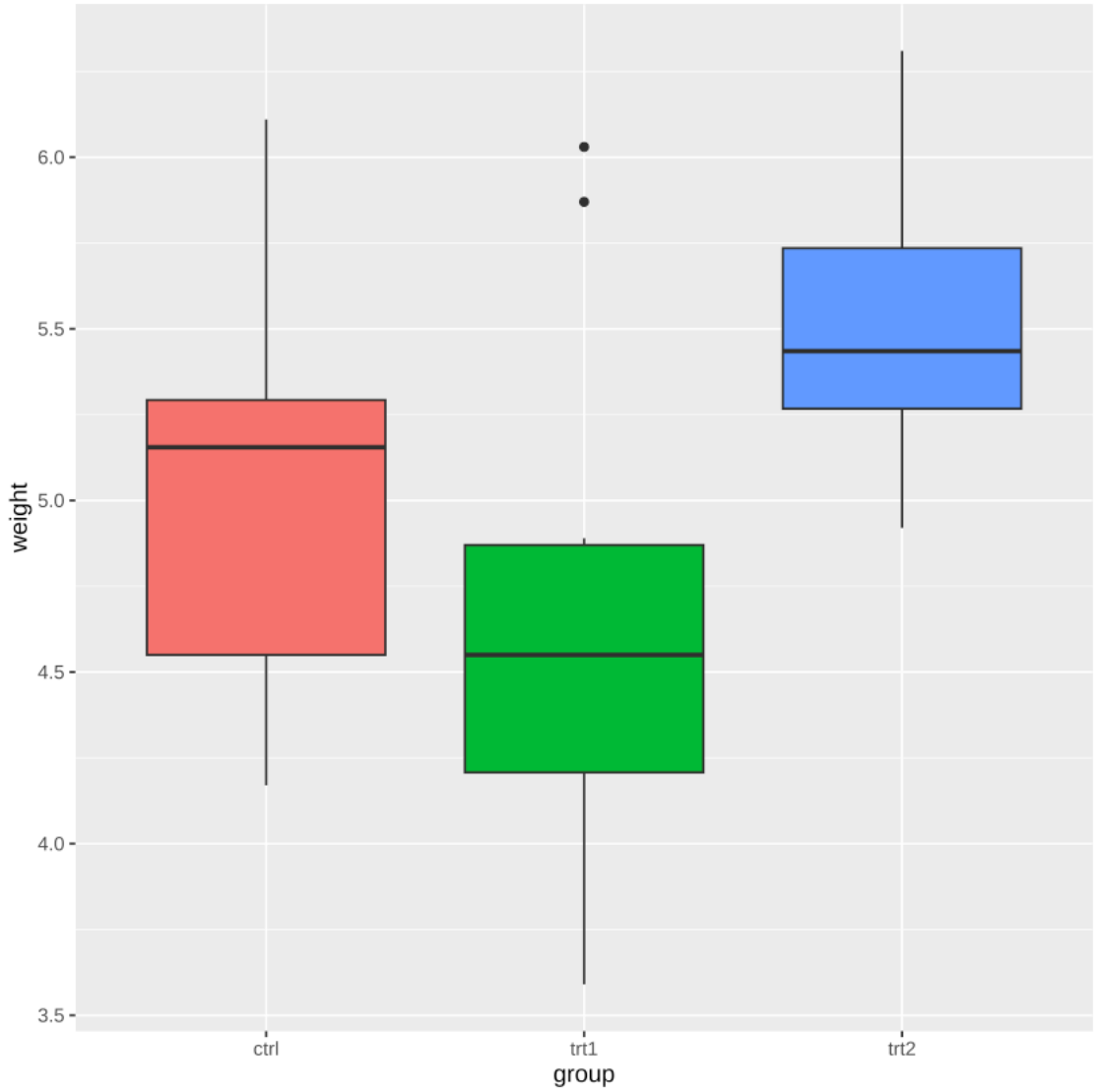
5.1.10 图例

[91]:
```
bp <- ggplot(data = PlantGrowth, aes(x = group, y = weight,
  fill = group)) + geom_boxplot()
bp
```



[92]:
```
# 删除特定美学的图例（填充）
bp + guides(fill = FALSE)
```

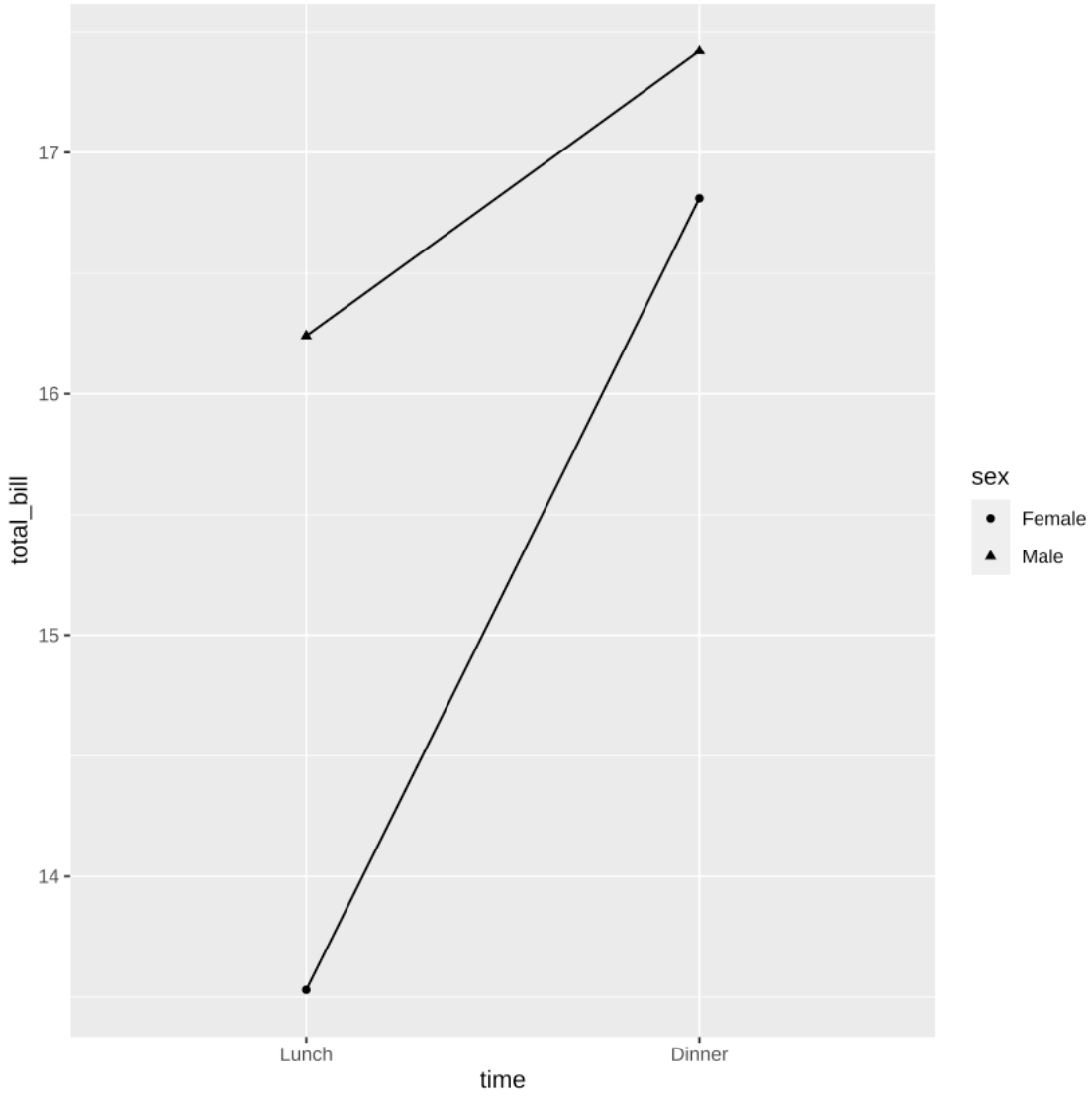
```
Warning message:
 “The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none"
instead as of ggplot2
3.3.4.”
```

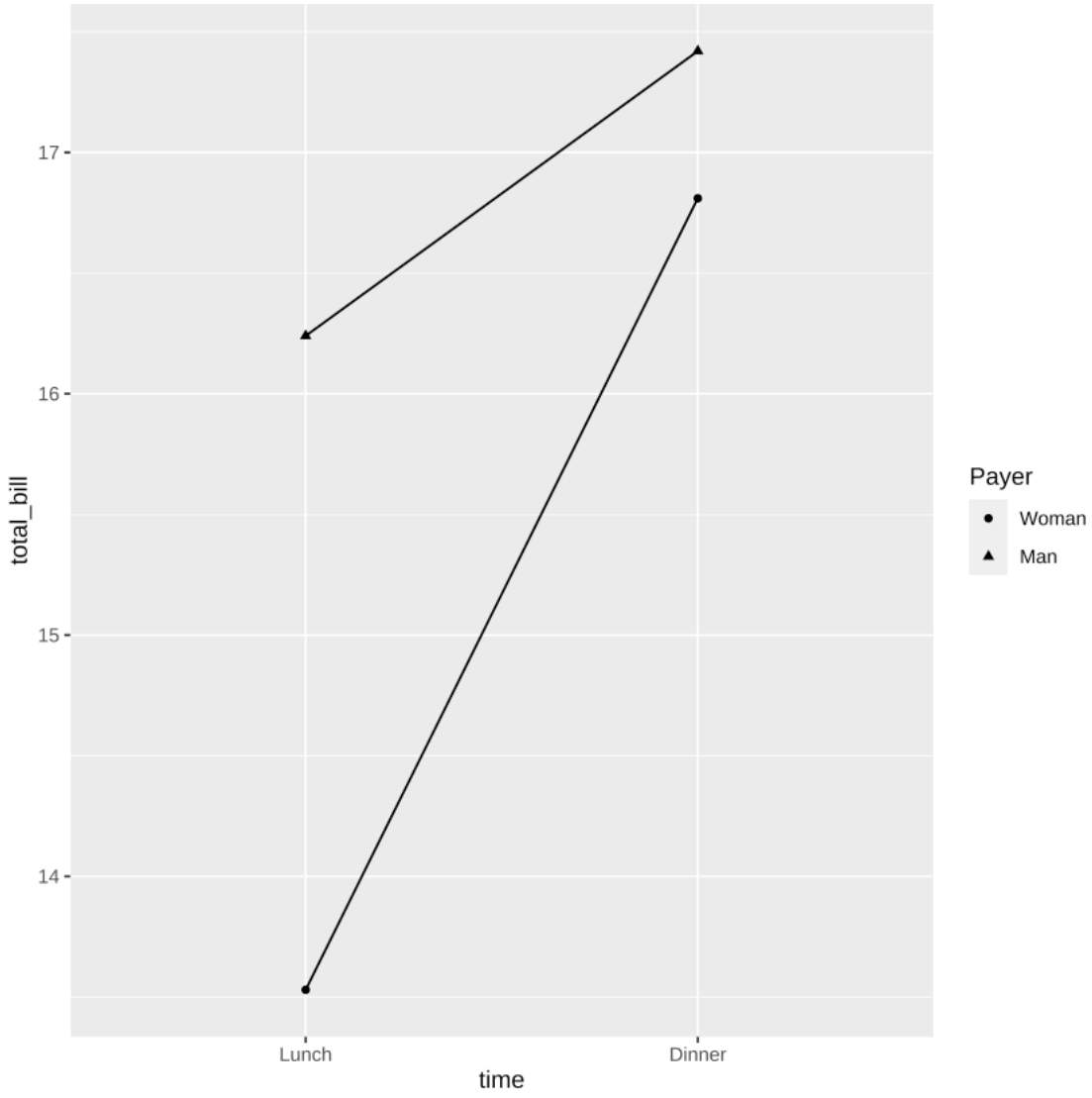


5.1.11 更改图例

[93]:
```
# 一个不同的数据集
df1 <- data.frame(sex = factor(c("Female", "Female", "Male",
  "Male")), time = factor(c("Lunch", "Dinner", "Lunch",
  "Dinner"), levels = c("Lunch", "Dinner")), total_bill = c(13.53,
  16.81, 16.24, 17.42))
```

```
# 基本的图表
lp <- ggplot(data = df1, aes(x = time, y = total_bill, group = sex,
  shape = sex)) + geom_line() + geom_point()
lp

# 更改图例
lp + scale_shape_discrete(name = "Payer", breaks = c("Female",
  "Male"), labels = c("Woman", "Man"))
```

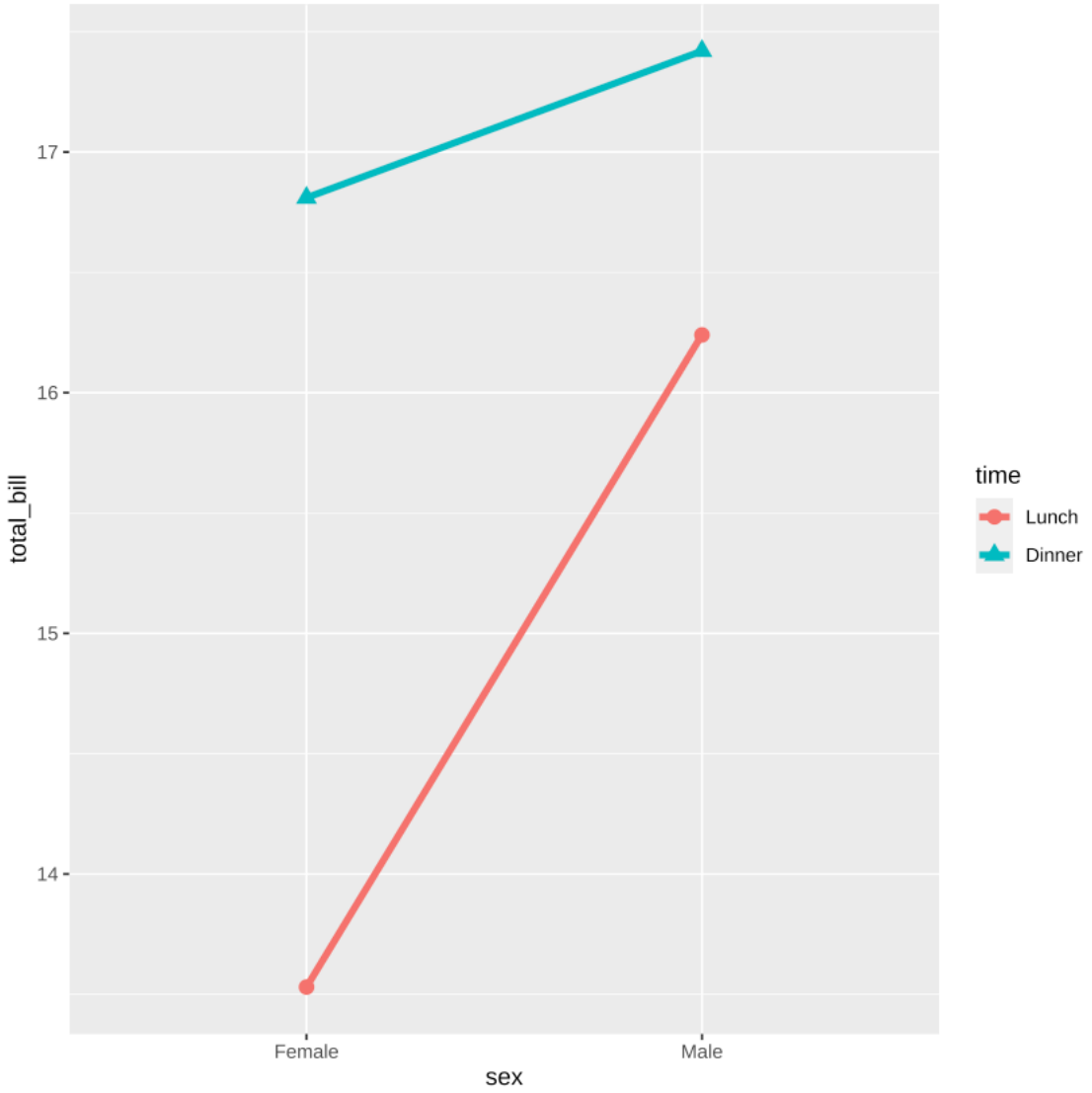




5.2 Maping

5.2.1 映射到形状

[94]:
```
ggplot(data = dat1, aes(x = sex, y = total_bill, group = time,
  shape = time, color = time)) +          # 映射到形状
 geom_line(linewidth=1.5) + # 设定线宽
 geom_point(size=3)         # 设定点大小
```

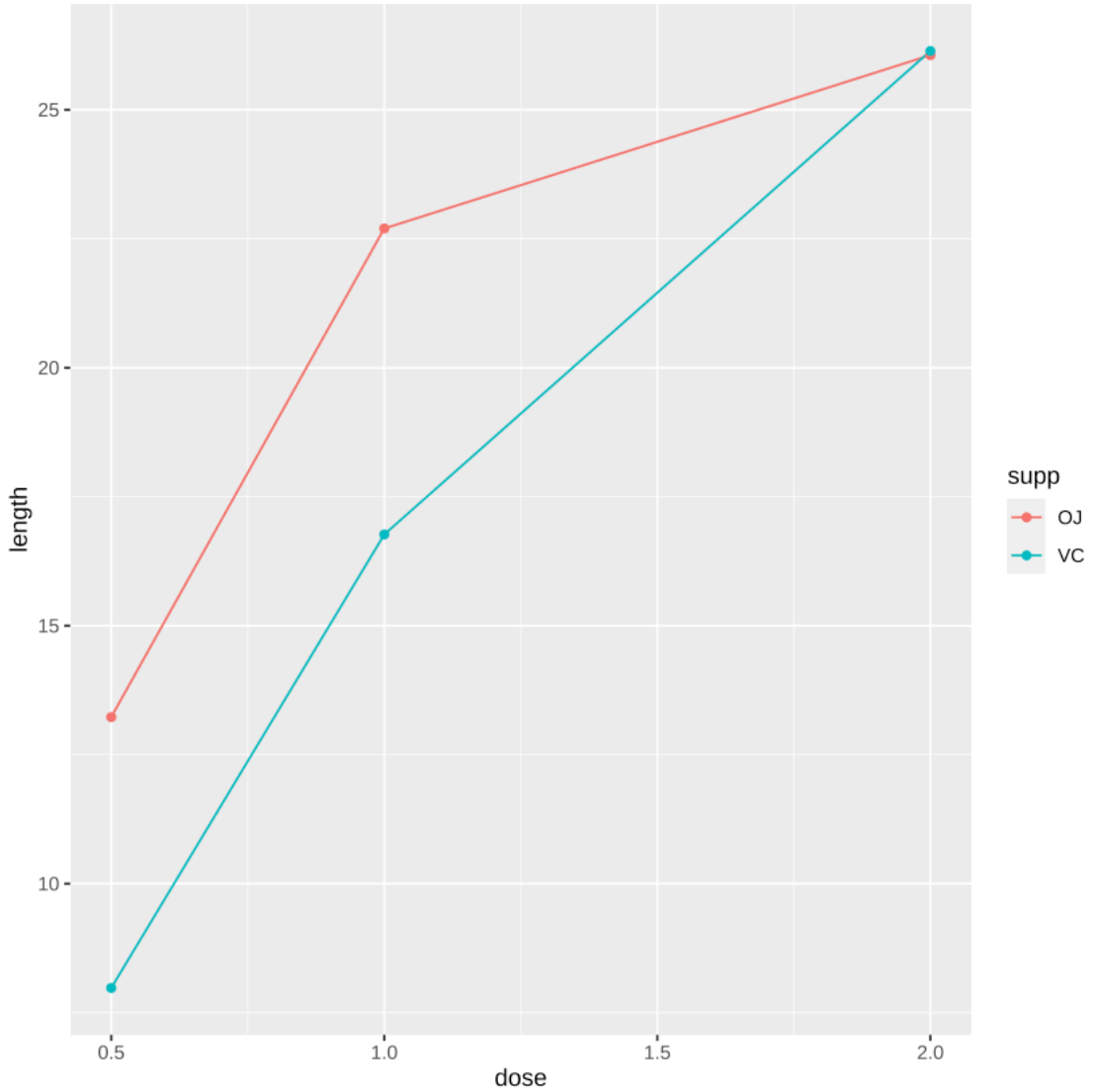


5.2.2 因子

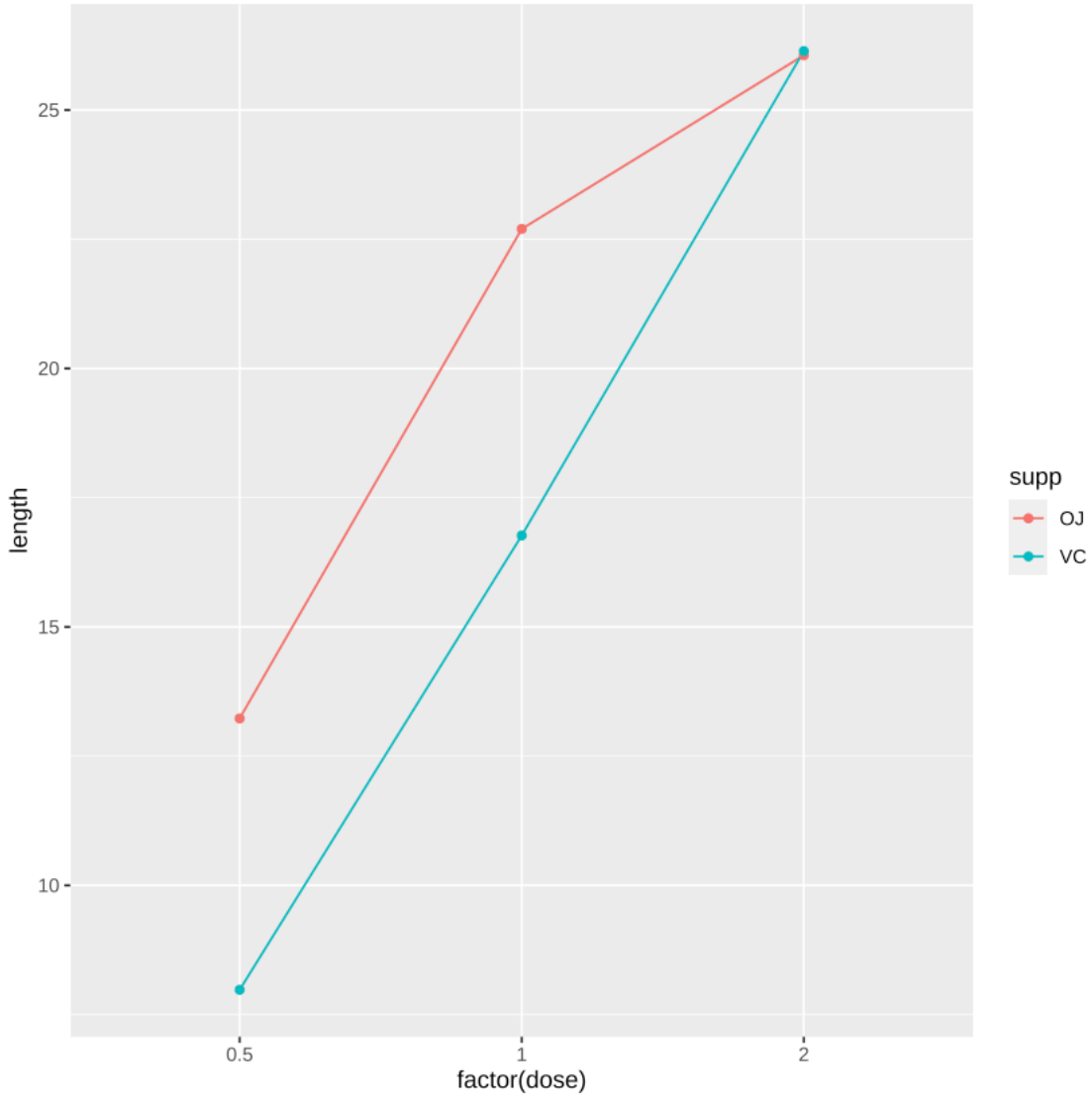
[95]:
```
datn <- read.table(header = TRUE, text = "
supp dose length
  OJ  0.5  13.23
  OJ  1.0  22.70
  OJ  2.0  26.06
  VC  0.5   7.98
```

```
  VC  1.0  16.77
  VC  2.0  26.14
")
```

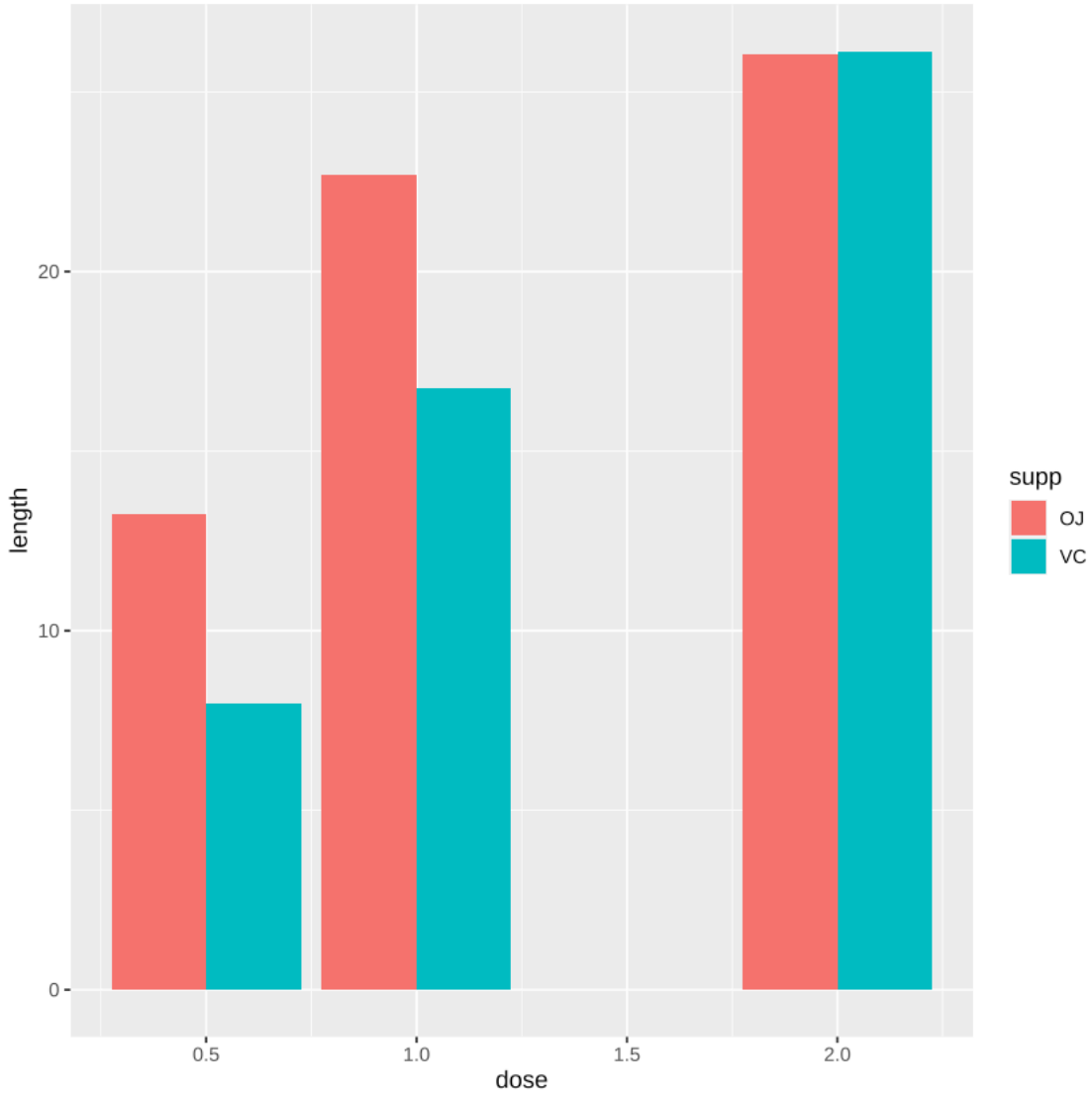
[96]:
```
# x-axis 作为连续变量
ggplot(data = datn, aes(x = dose, y = length, group = supp, colour = supp)) +␣
 ↪geom_line() +
    geom_point()
```



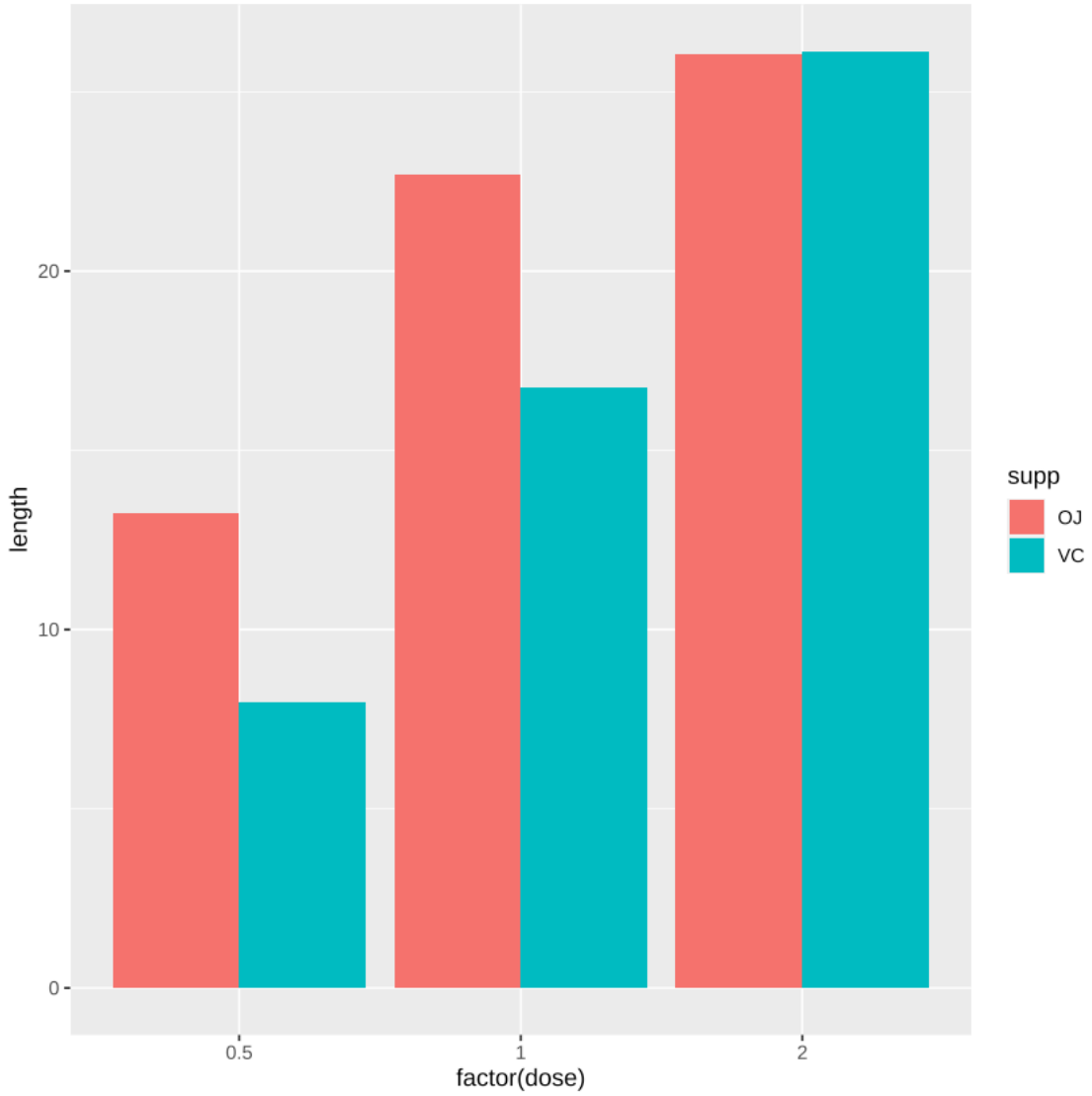
[97]:
```
# 使用原始的数据框，但使用 factor 函数在绘图时转换
ggplot(data = datn, aes(x = factor(dose), y = length, group = supp,
  colour = supp)) + geom_line() + geom_point()
```



[98]:
```
# 连续值作为分类变量使用时，也可以绘制条形图
ggplot(data = datn, aes(x = dose, y = length, fill = supp)) + geom_bar(stat =␣
 ↪"identity",
    position = position_dodge())
```



[99]:
```
ggplot(data = datn, aes(x = factor(dose), y = length, fill = supp)) +
  geom_bar(stat = "identity", position = position_dodge())
```



5.3 分面

[100]:
```
library(reshape2)
# 查看头几行数据
head(tips)
```

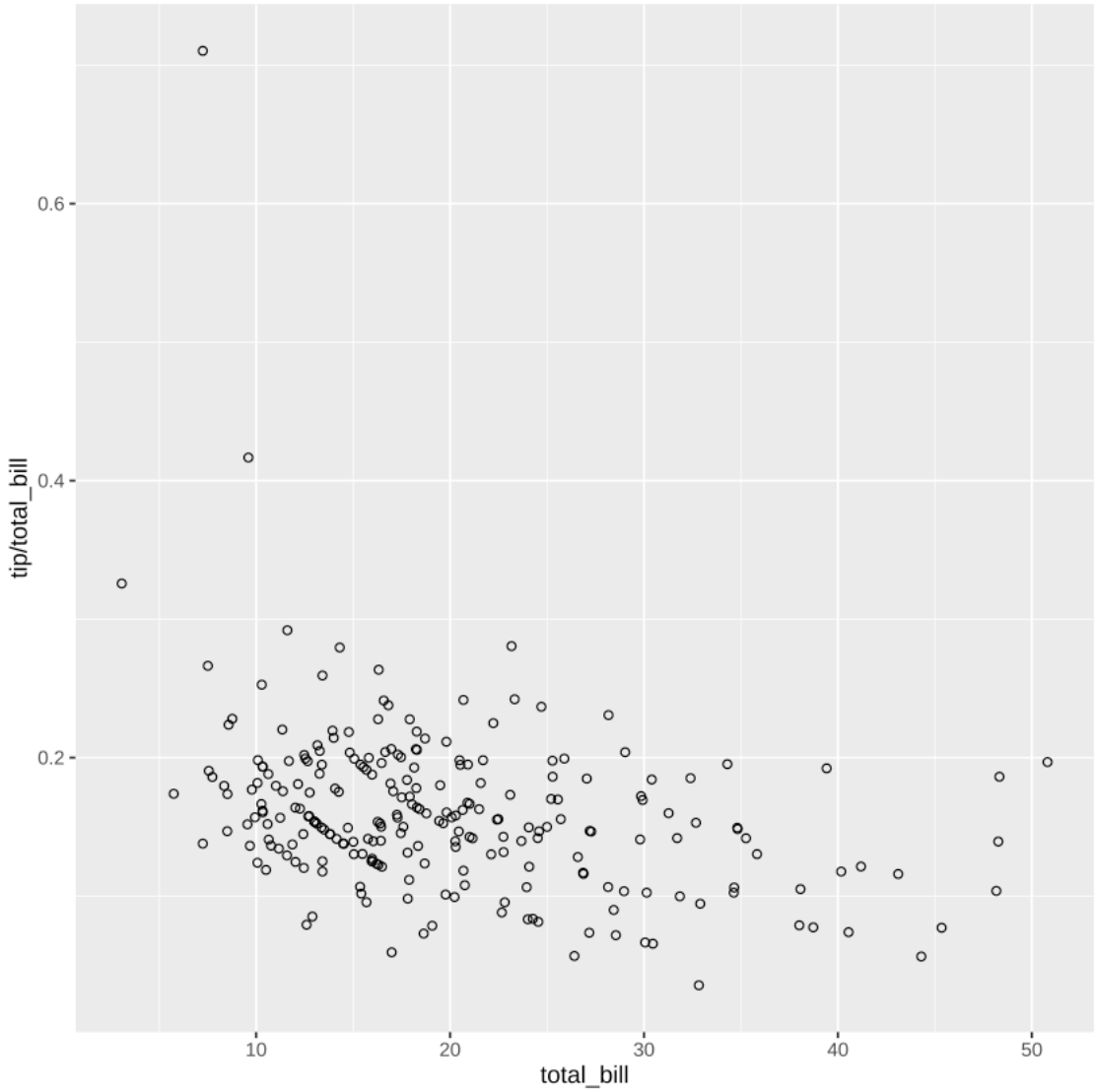
载入程辑包：'reshape2'

```
The following object is masked from 'package:tidyr' :

    smiths
```

A data.frame: 6 × 7

	total_bill <dbl>	tip <dbl>	sex <fct>	smoker <fct>	day <fct>	time <fct>	size <int>
1	16.99	1.01	Female	No	Sun	Dinner	2
2	10.34	1.66	Male	No	Sun	Dinner	3
3	21.01	3.50	Male	No	Sun	Dinner	3
4	23.68	3.31	Male	No	Sun	Dinner	2
5	24.59	3.61	Female	No	Sun	Dinner	4
6	25.29	4.71	Male	No	Sun	Dinner	4

[101]:
```
sp <- ggplot(tips, aes(x = total_bill, y = tip/total_bill)) + geom_point(shape
  ↪= 1)
sp
```



5.3.1 facet_grid

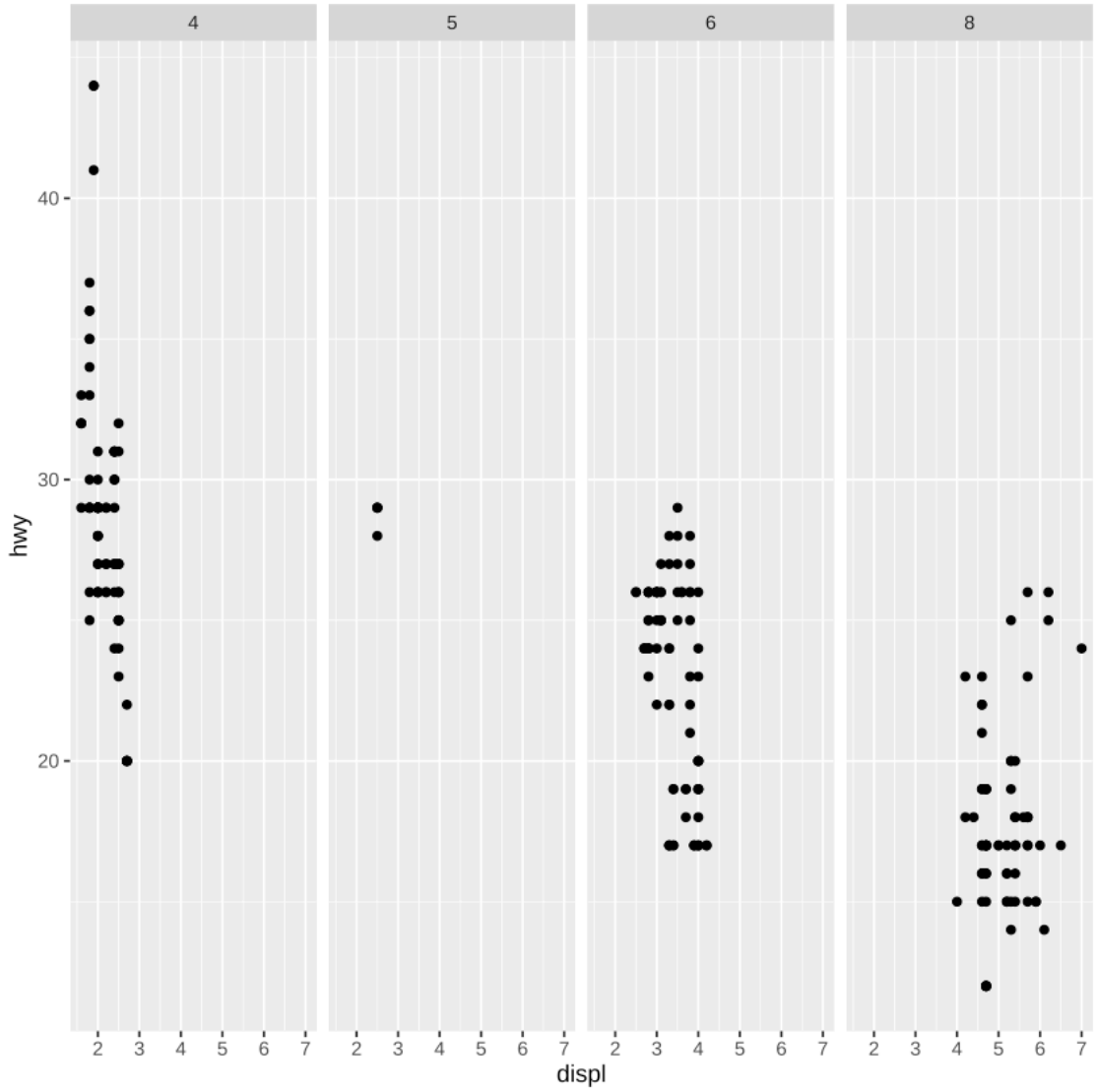
facet_grid() 函数可以将绘图面板按行和列的方式排列成一个矩阵。这对于比较 y 轴位置非常有用，因为垂直刻度是对齐的。你可以使用 facet_grid(rows ~ cols) 来指定行和列

[102]:
```
head(mpg)
```

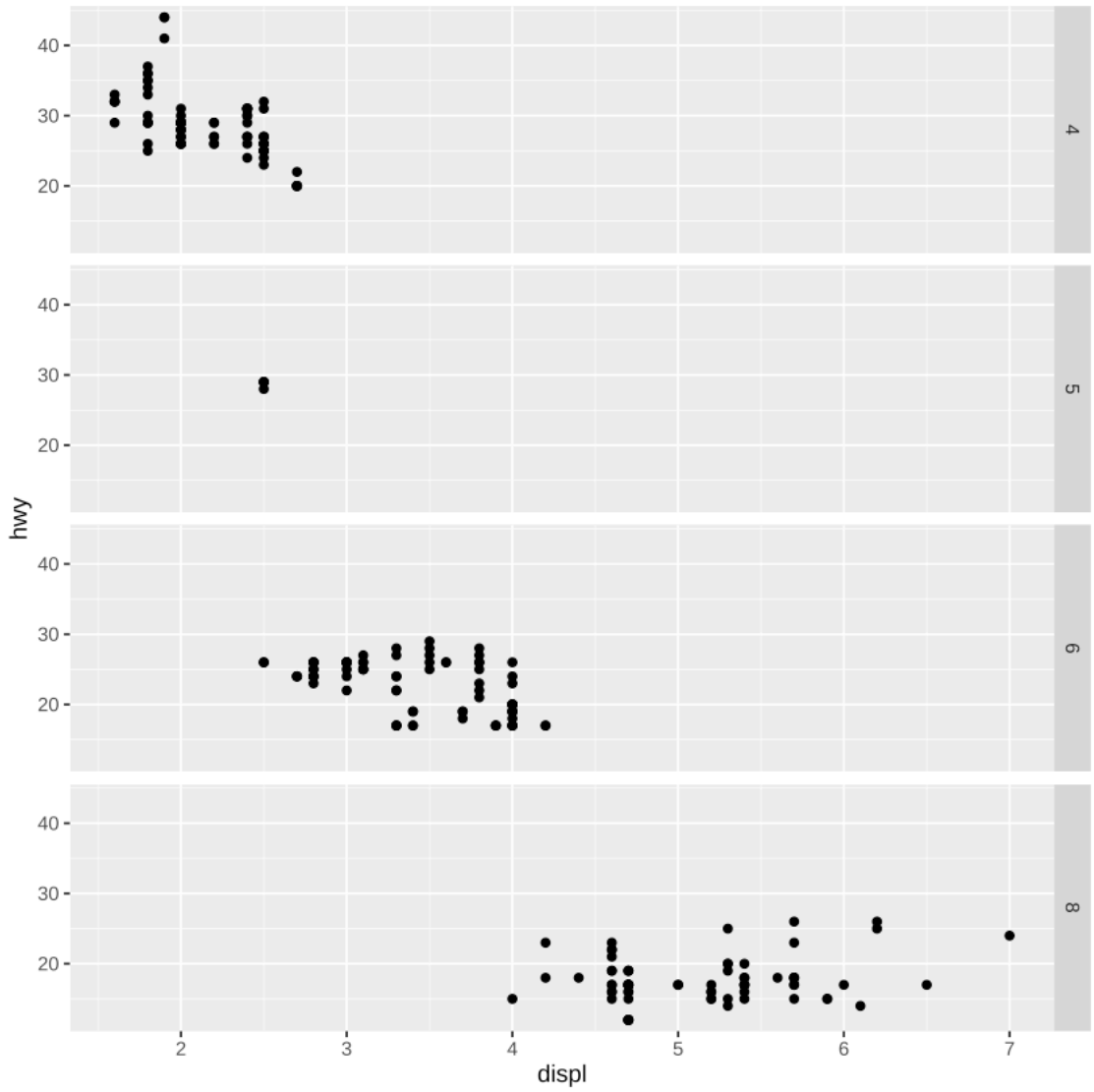
A tibble: 6 × 11

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

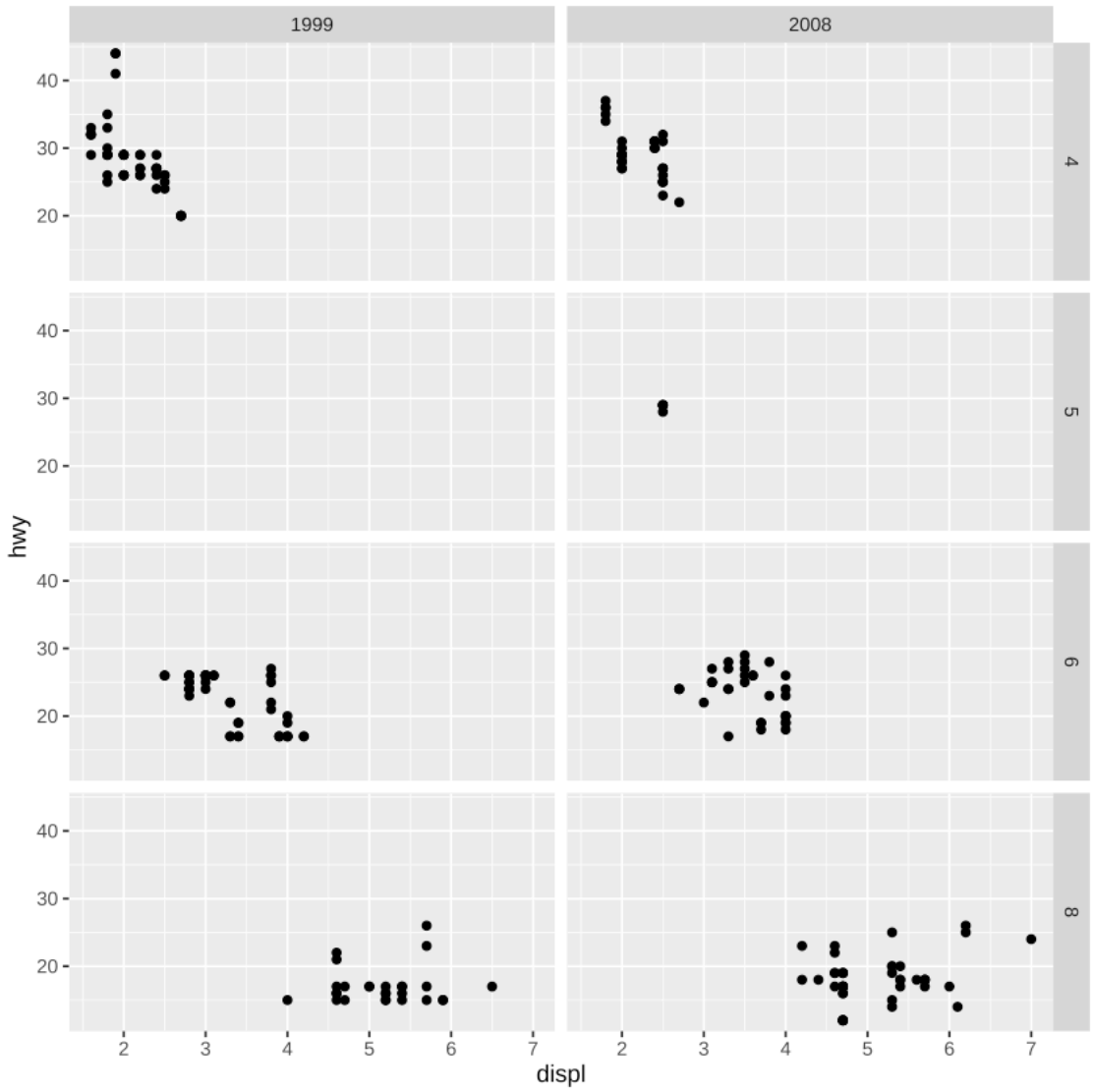
[103]:
```
p <- ggplot(mpg, aes(displ, hwy)) + geom_point()

# 根据 'sex' 按水平方向分割
p + facet_grid(. ~ cyl)
```



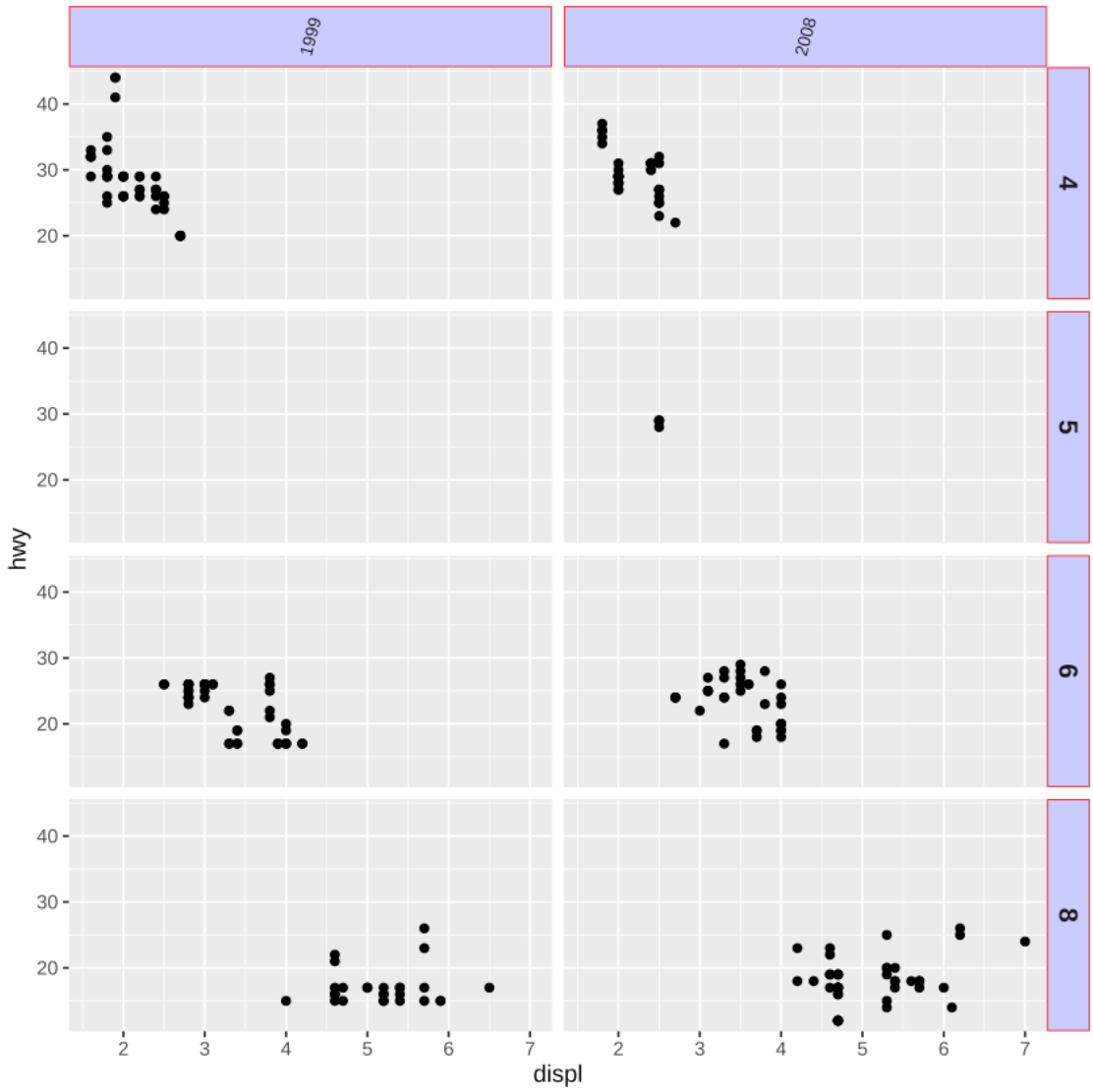
[104]:
```
# 根据 'sex' 按垂直方向分割
p + facet_grid(cyl~ .)
```



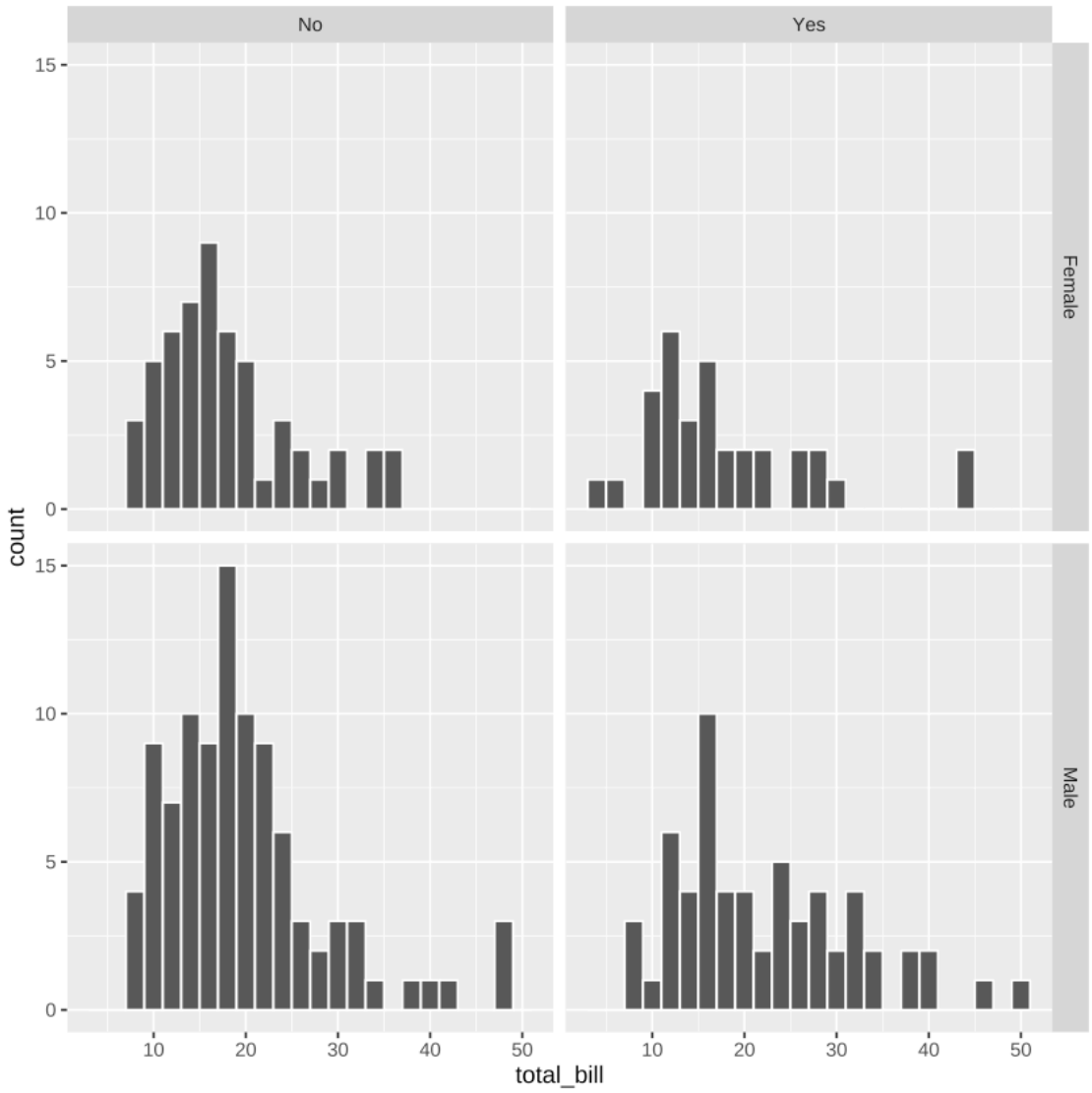
[105]:
```
# 垂直方向以 'cyl' 分割，水平方向以 'year' 分割。
p + facet_grid(cyl ~ year)
```



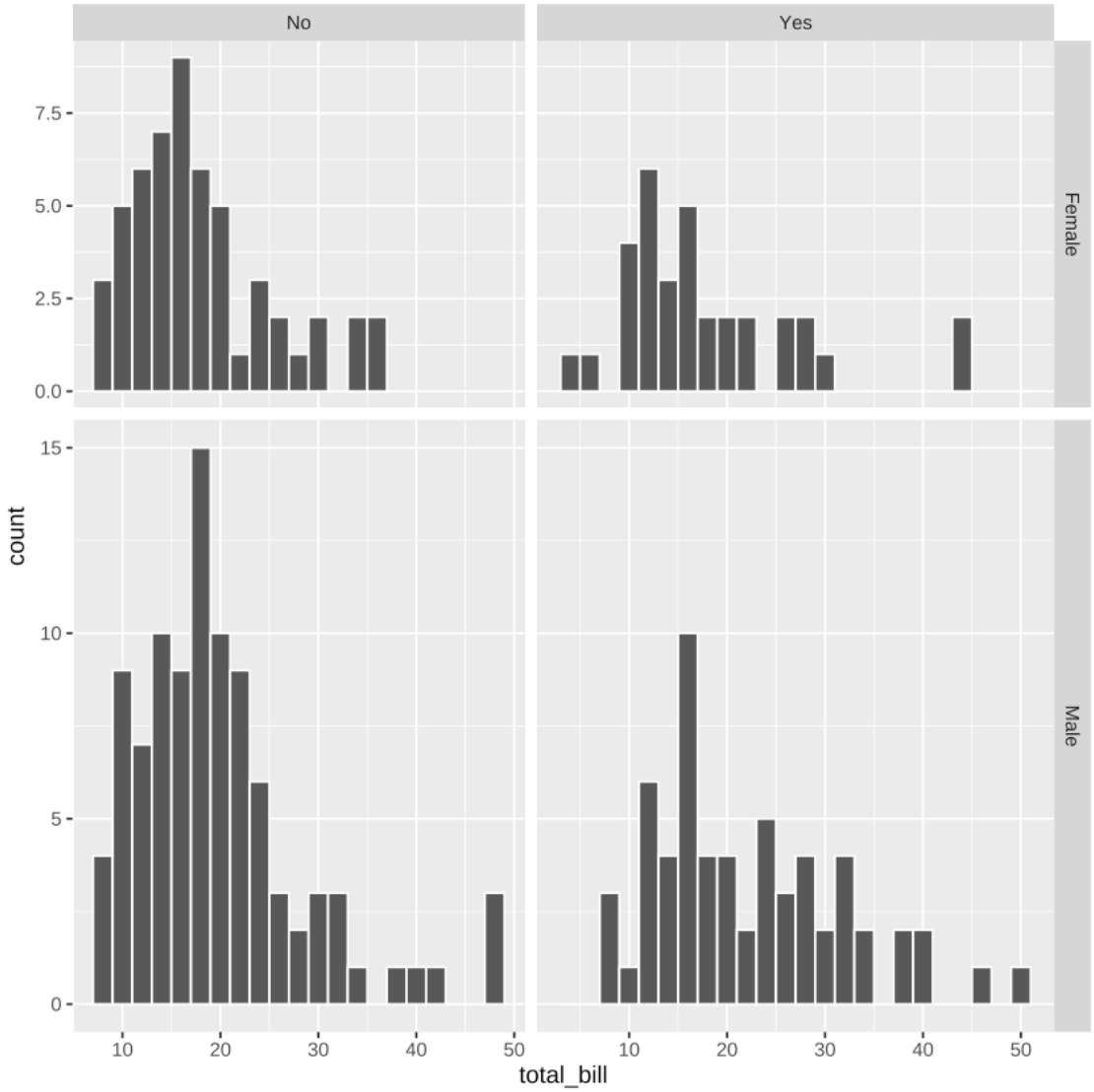
[106]:
```
p + facet_grid(cyl ~ year) + theme(strip.text.x = element_text(size = 8,
  angle = 75), strip.text.y = element_text(size = 12,
  face = "bold"), strip.background = element_rect(colour = "red",
  fill = "#CCCCFF"))
```



[107]:
```
# 描绘一个 total_bill 的柱状图
hp <- ggplot(tips, aes(x = total_bill)) + geom_histogram(binwidth = 2, colour =␣
 ↪"white")
# 根据性别和是否吸烟进行分面
hp + facet_grid(sex ~ smoker)
```



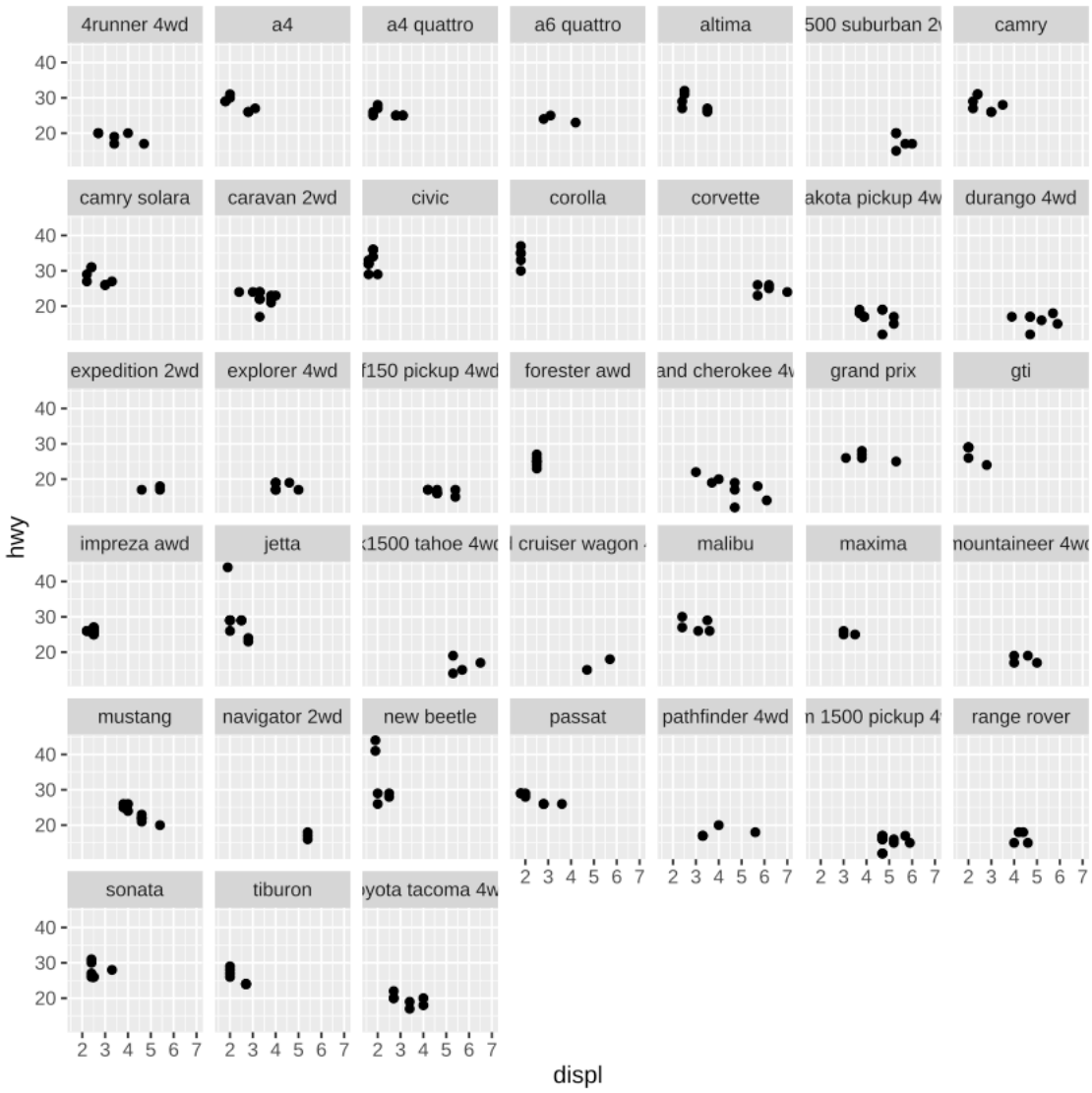
[108]:
```
# 画布的缩放比例不变，但各分面的范围有所改变，因此每个分面的物理大小都不一致
hp + facet_grid(sex ~ smoker, scales = "free", space = "free")
```



5.3.2 facet_wrap

facet_wrap() 函数可以将绘图面板按照一个变量的多个水平分组，并将其包装成 2D 的形式。这对于具有多个水平的单个变量的绘图排列非常有用。你可以使用 facet_wrap(~ variable) 来指定变量。

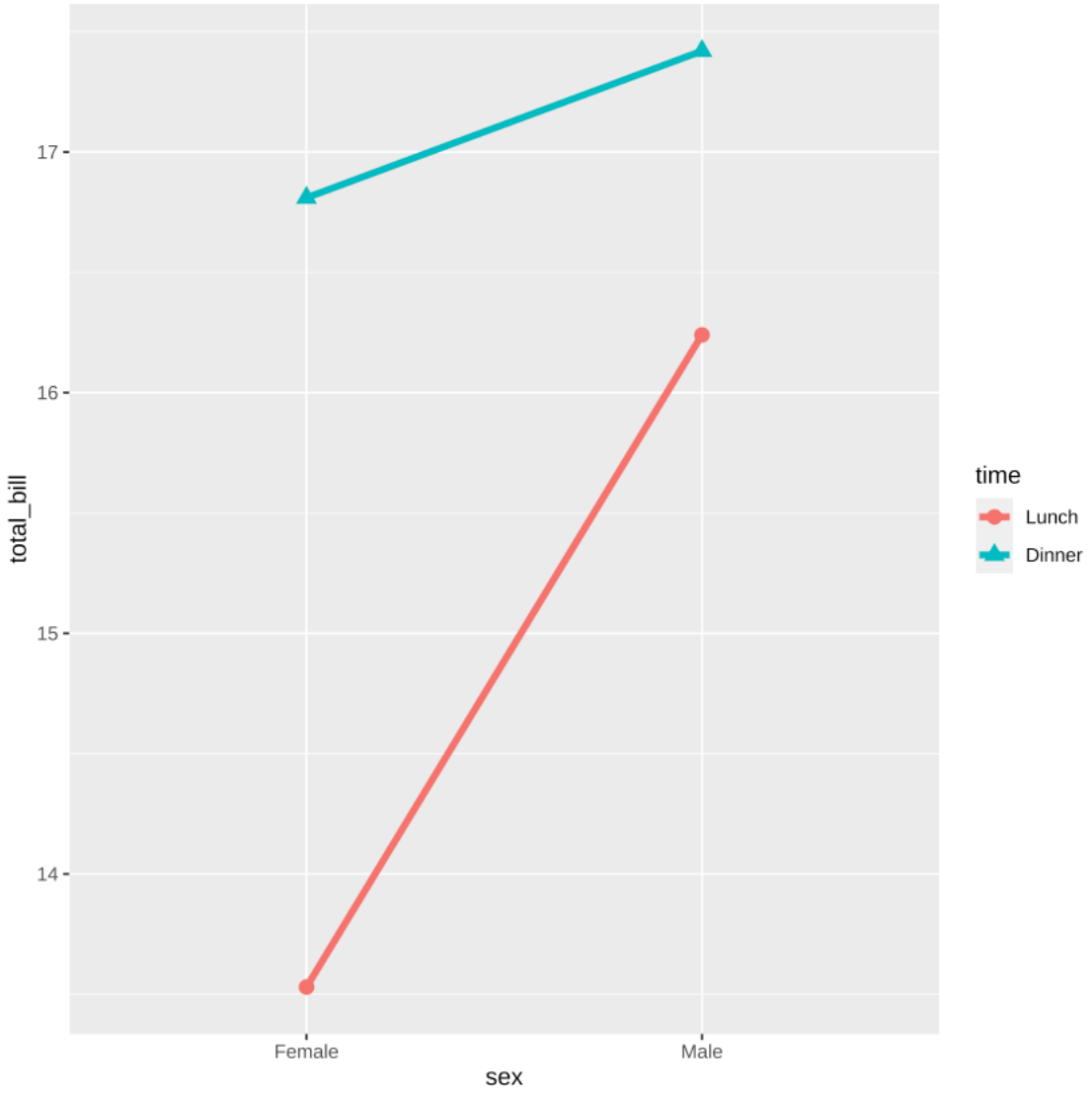
[109]:
```
p + facet_wrap(~ model)
```



5.4 Maping

5.4.1 映射到形状

[110]:
```
ggplot(data = dat1, aes(x = sex, y = total_bill, group = time,
  shape = time, color = time)) +          # 映射到形状
 geom_line(linewidth=1.5) + # 设定线宽
 geom_point(size=3)         # 设定点大小
```

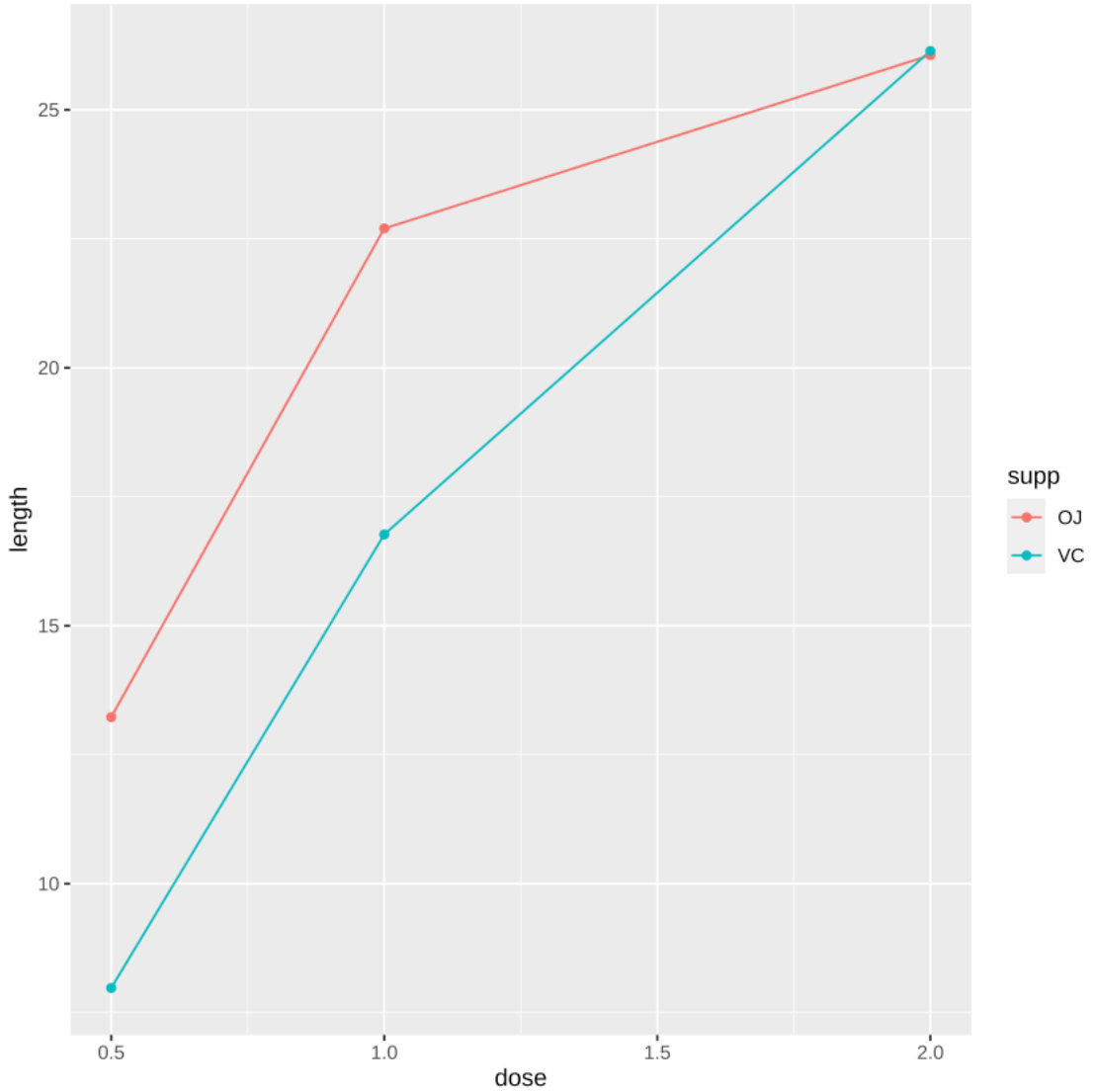


5.4.2 因子

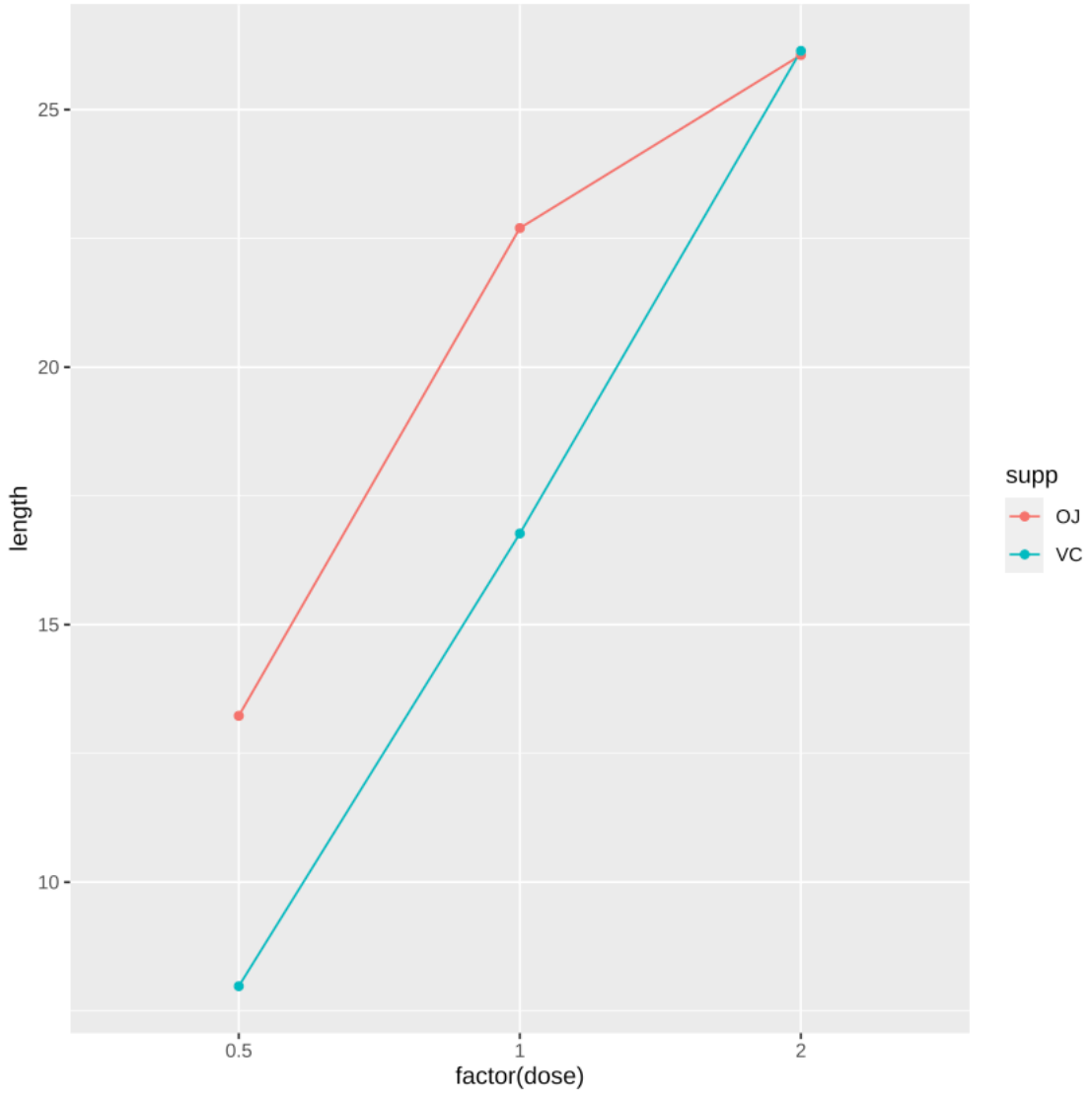
[111]:
```
datn <- read.table(header = TRUE, text = "
supp dose length
  OJ  0.5  13.23
  OJ  1.0  22.70
  OJ  2.0  26.06
  VC  0.5   7.98
```

```
  VC  1.0  16.77
  VC  2.0  26.14
")
```

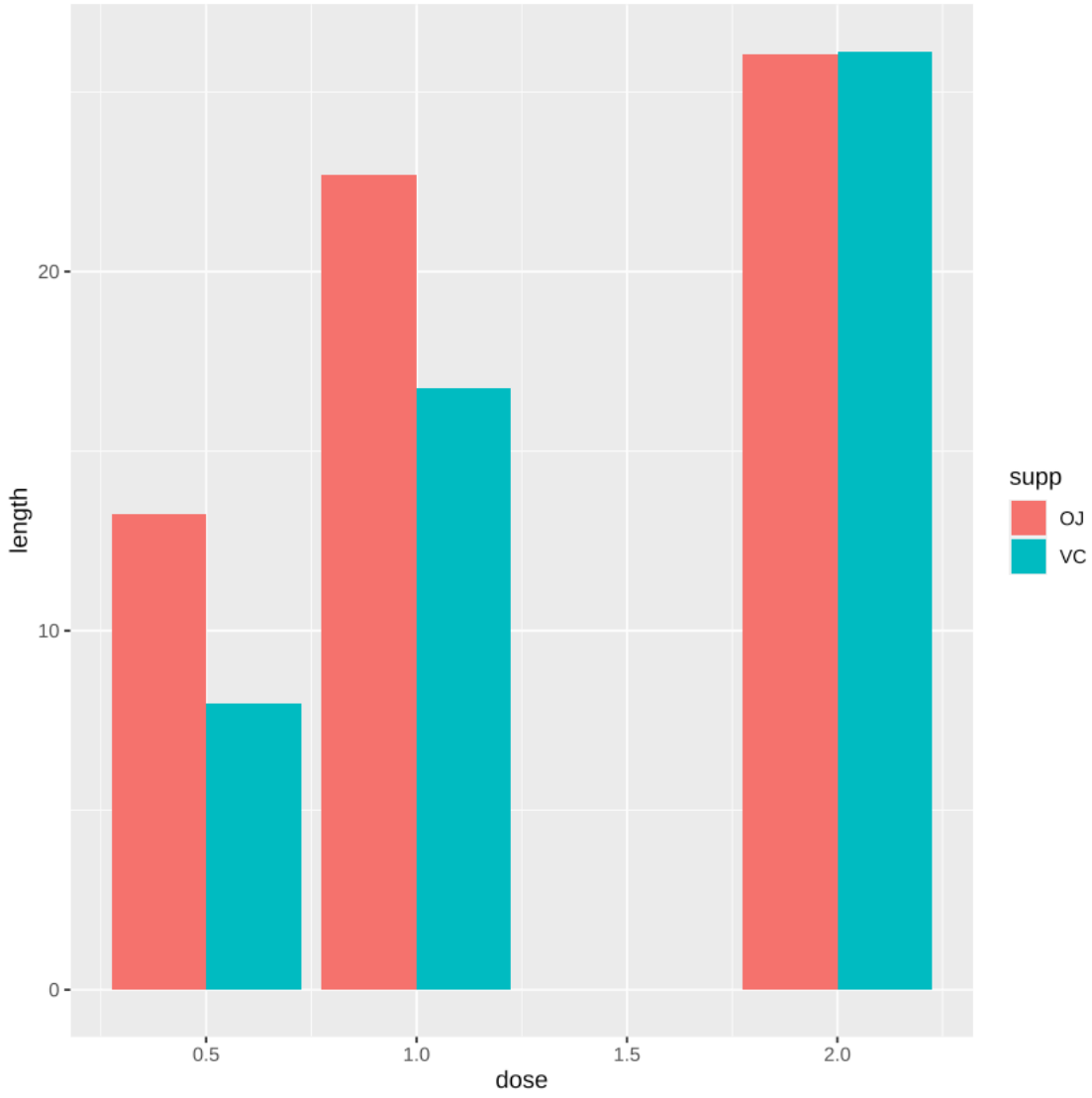
[112]:
```
# x-axis 作为连续变量
ggplot(data = datn, aes(x = dose, y = length, group = supp, colour = supp)) +␣
 ↪geom_line() +
    geom_point()
```



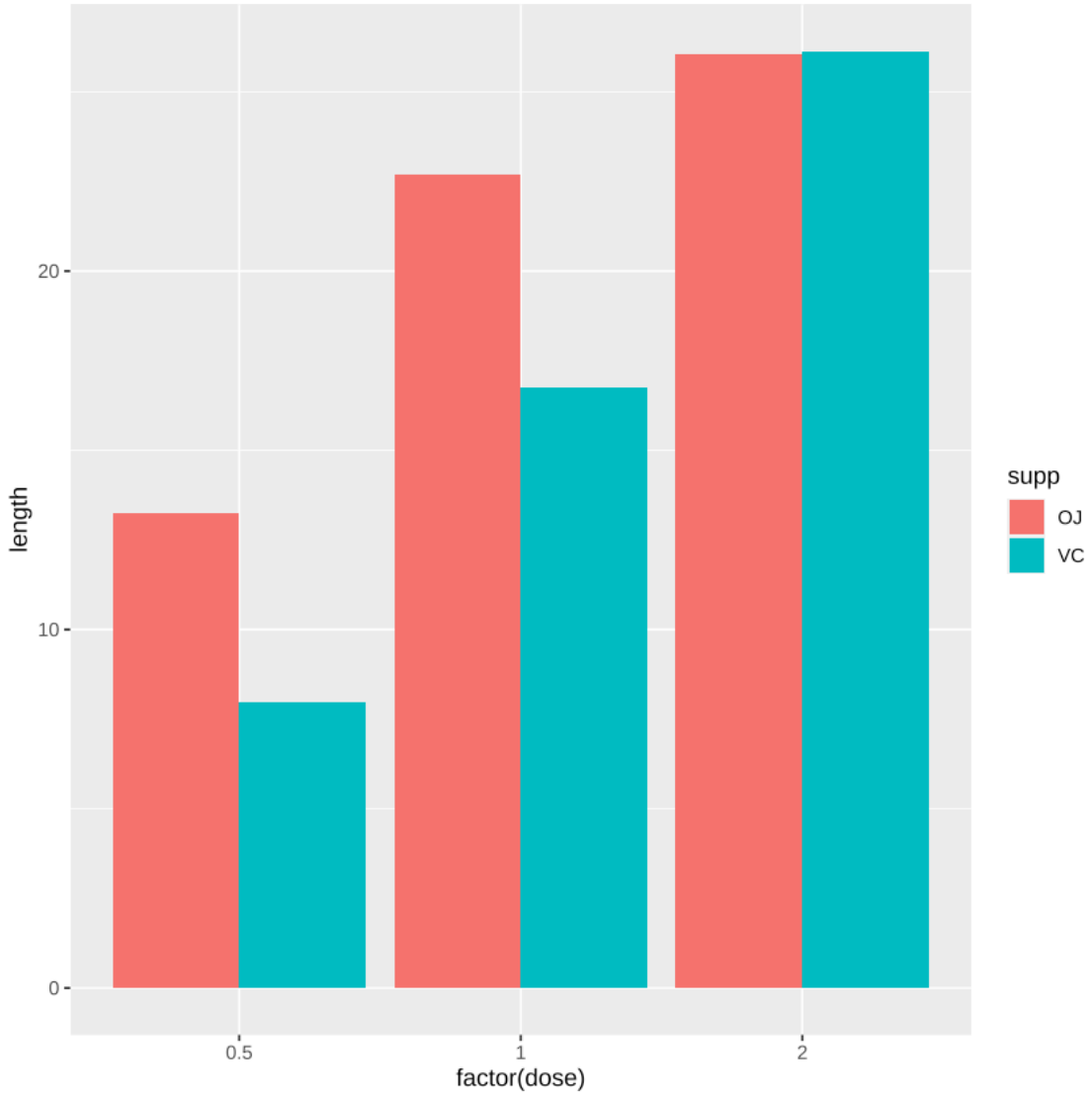
[113]:
```
# 使用原始的数据框，但使用 factor 函数在绘图时转换
ggplot(data = datn, aes(x = factor(dose), y = length, group = supp,
  colour = supp)) + geom_line() + geom_point()
```



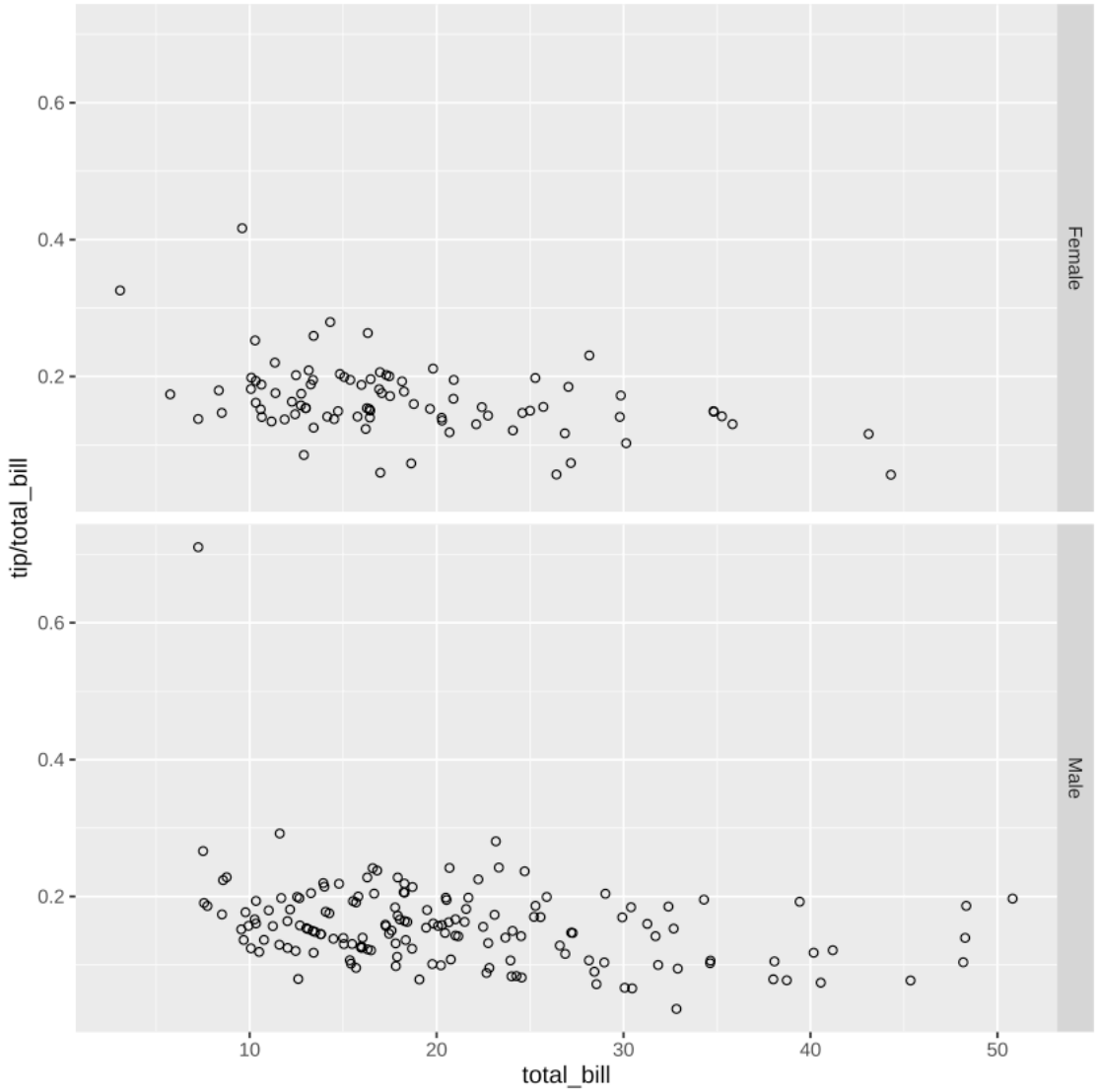
[114]:
```
# 连续值作为分类变量使用时，也可以绘制条形图
ggplot(data = datn, aes(x = dose, y = length, fill = supp)) + geom_bar(stat =␣
 ↪"identity",
    position = position_dodge())
```



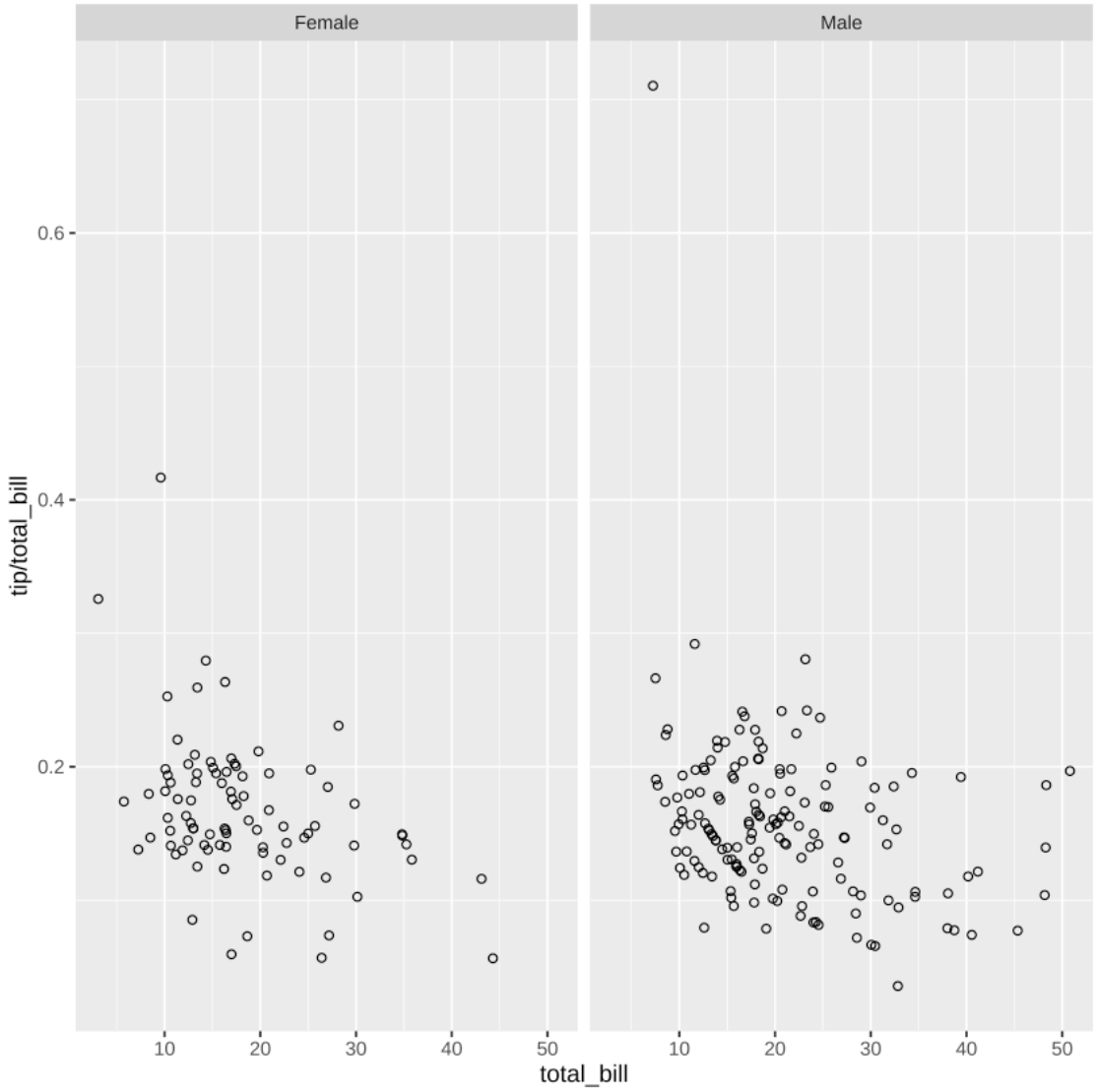
[115]:
```
ggplot(data = datn, aes(x = factor(dose), y = length, fill = supp)) +
  geom_bar(stat = "identity", position = position_dodge())
```



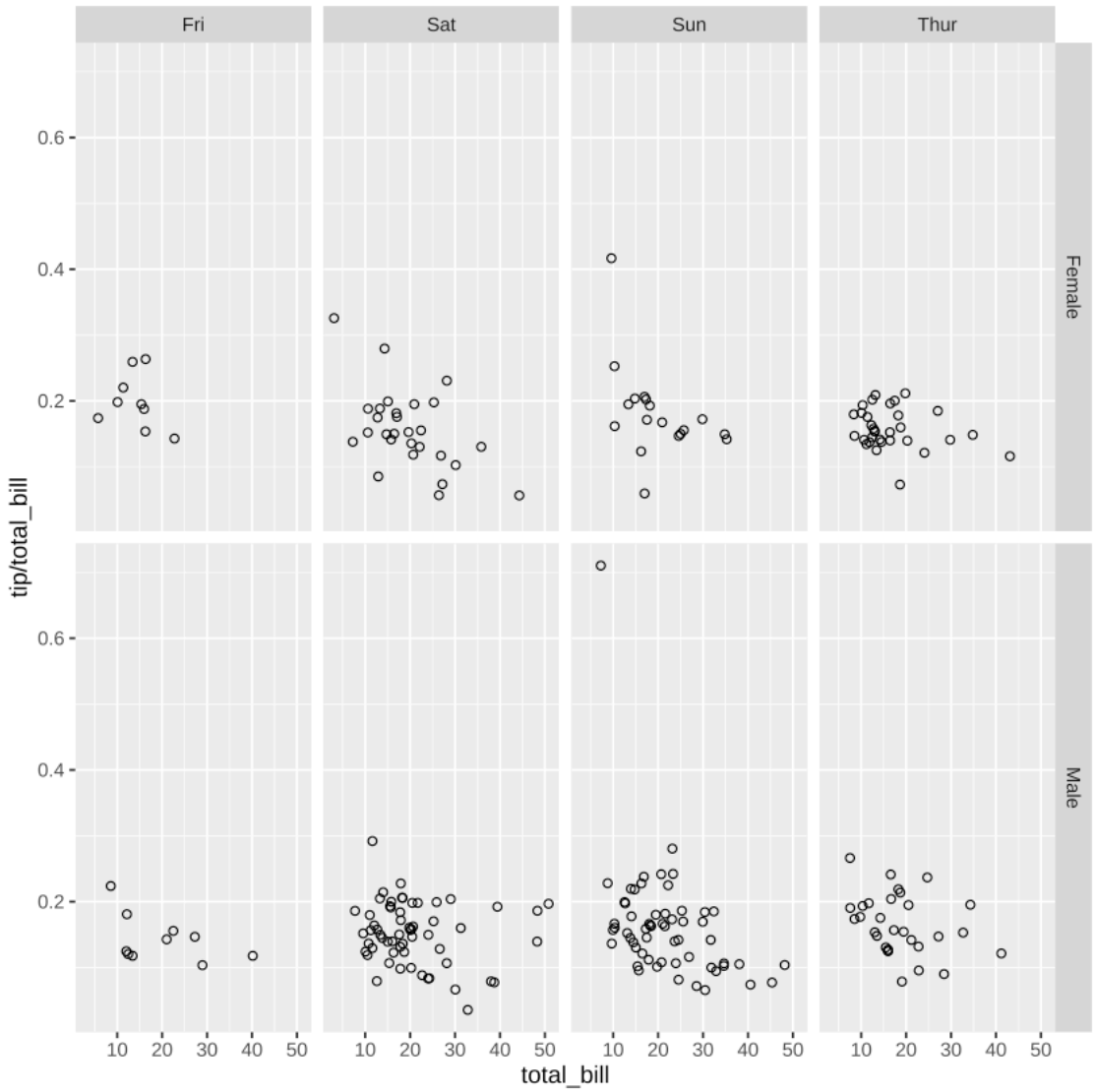
[116]:
```
# 根据 'sex' 按垂直方向分割
sp + facet_grid(sex ~ .)
```



[117]:
```
# 根据 'sex' 按水平方向分割。
sp + facet_grid(. ~ sex)
```

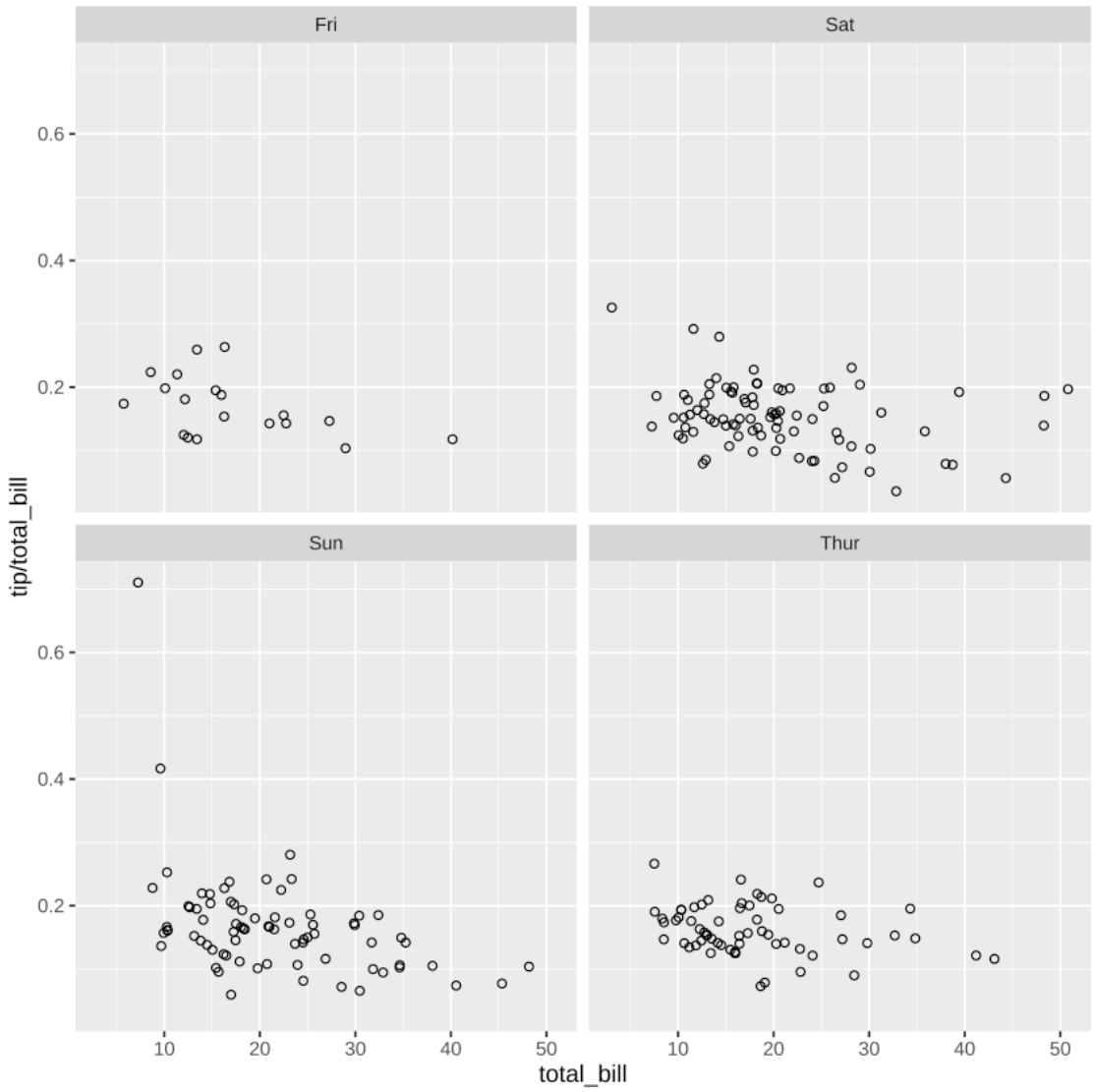


[118]:
```
# 垂直方向以 'sex' 分割，水平方向以 'day' 分割。
sp + facet_grid(sex ~ day)
```

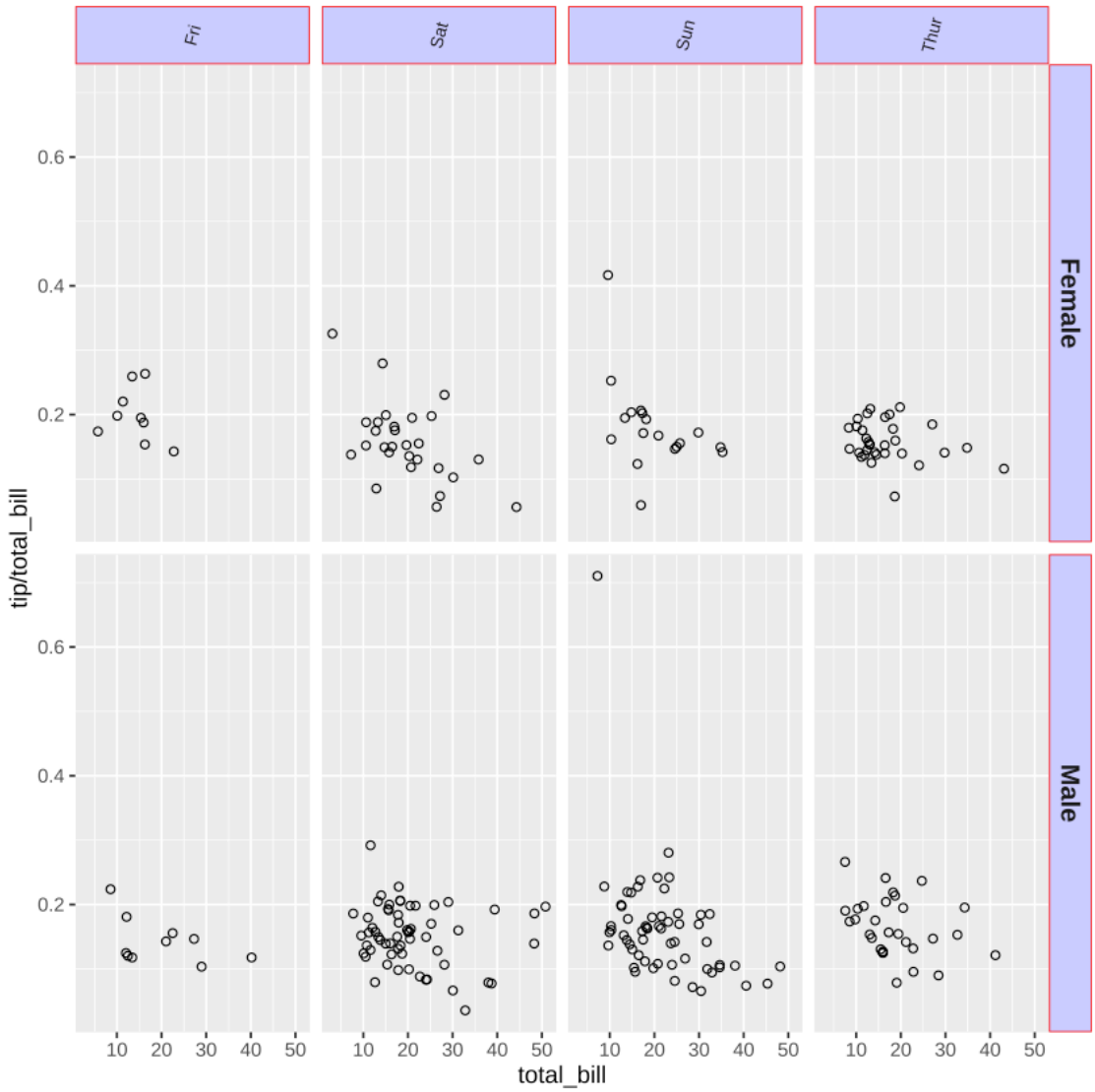


除了能够根据单个变量在水平或垂直方向上对图进行分面，facet_wrap() 函数可以通过设置特定的行数或列数，让子图排列到一起。此时每个图像的上方都会有标签。

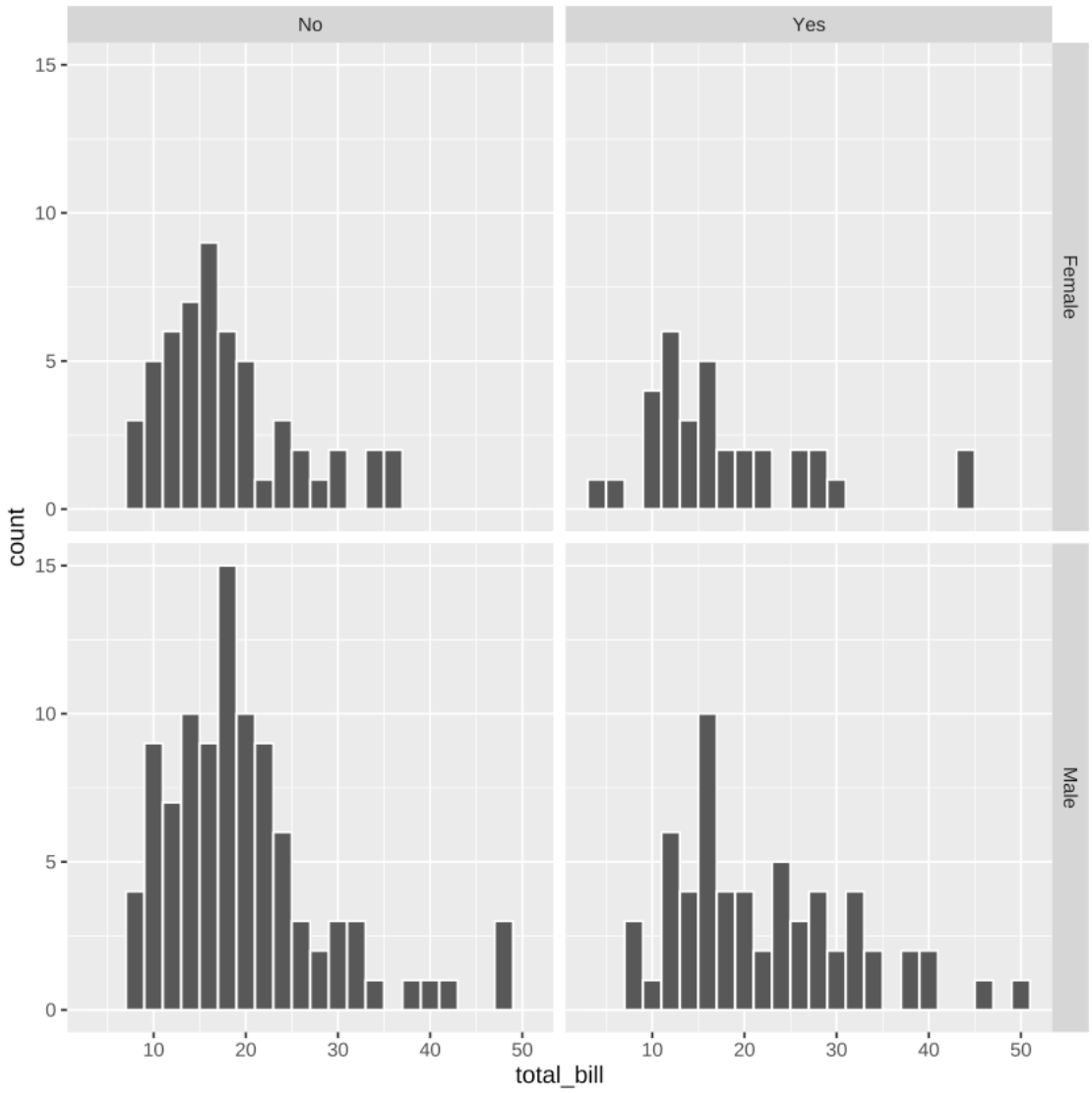
[119]:
```
# 以变量 `day` 进行水平分面，分面的行数为 2。
sp + facet_wrap(~day, ncol = 2)
```



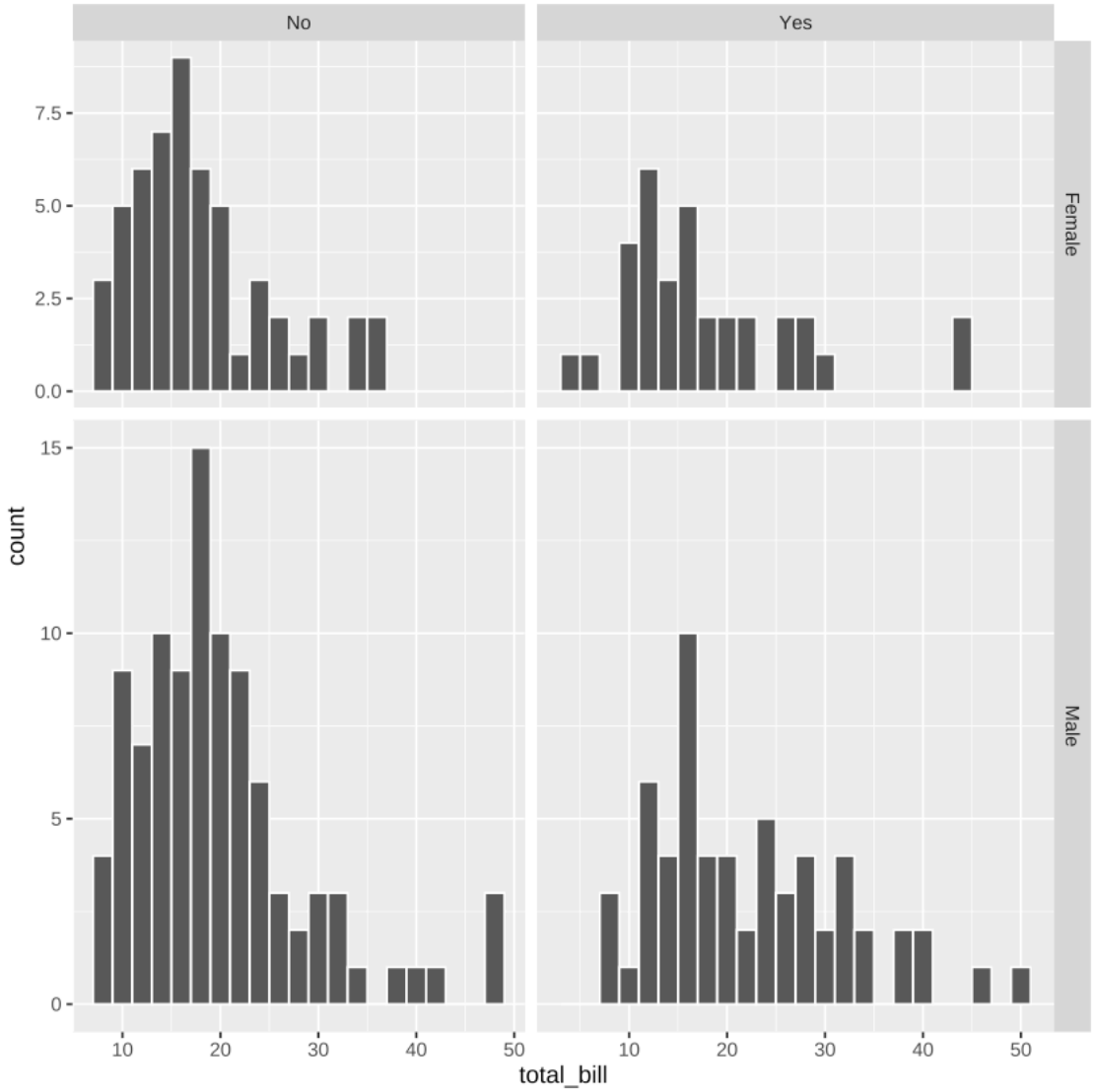
[120]:
```
sp + facet_grid(sex ~ day) + theme(strip.text.x = element_text(size = 8, angle␣
↪= 75),
   strip.text.y = element_text(size = 12, face = "bold"), strip.background =␣
↪element_rect(colour = "red",
       fill = "#CCCCFF"))
```



[121]:
```
# 描绘一个 total_bill 的柱状图
hp <- ggplot(tips, aes(x = total_bill)) + geom_histogram(binwidth = 2, colour =␣
↪"white")
# 根据性别和是否吸烟进行分面
hp + facet_grid(sex ~ smoker)
```



[122]:
```
# 画布的缩放比例不变，但各分面的范围有所改变，因此每个分面的物理大小都不一致
hp + facet_grid(sex ~ smoker, scales = "free", space = "free")
```



5.5 基本图形

5.5.1 直方图与核密度曲线

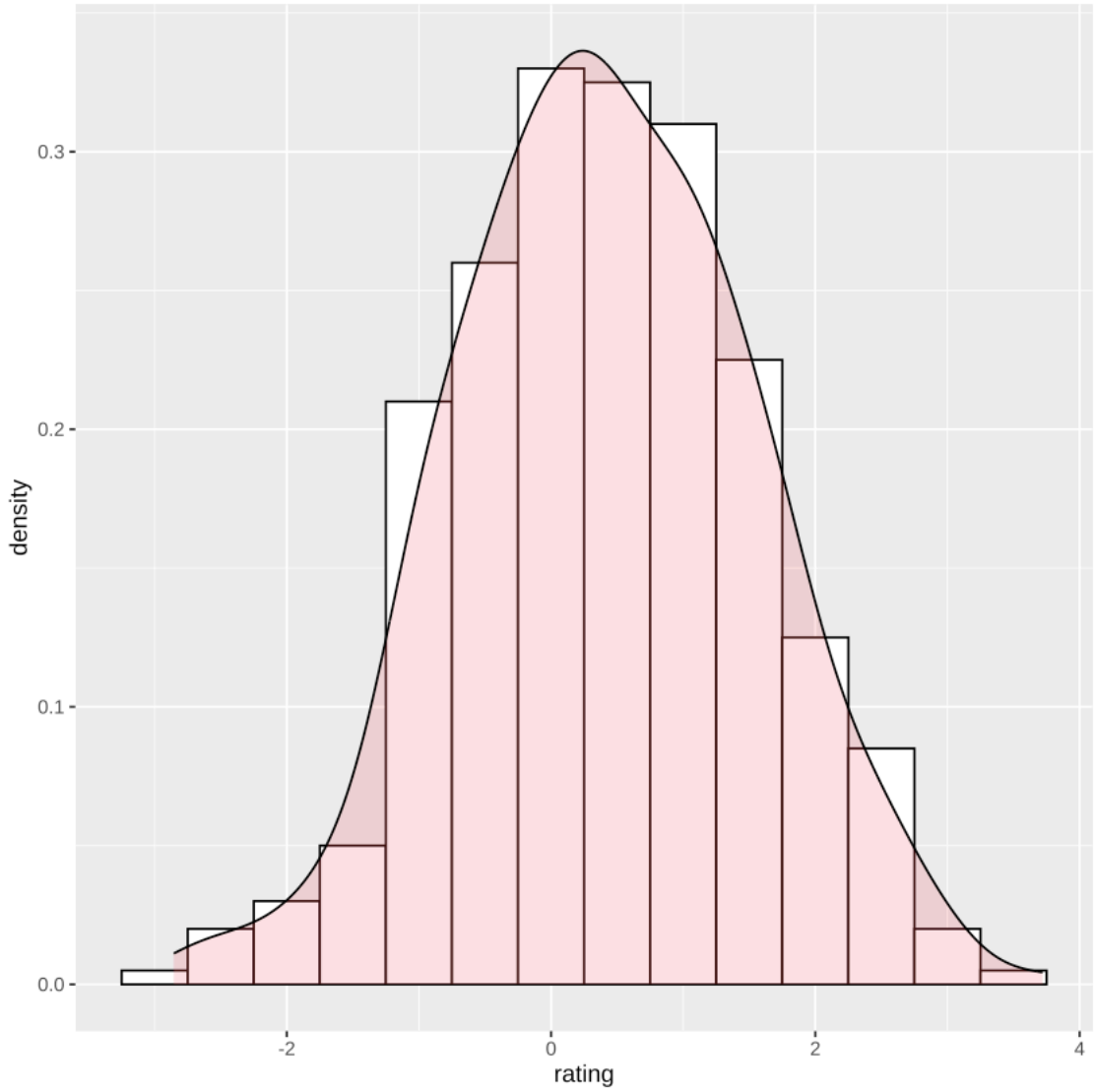
[123]:
```
set.seed(1234)
dat <- data.frame(cond = factor(rep(c("A", "B"), each = 200)), rating =␣
↪c(rnorm(200),
   rnorm(200, mean = 0.8)))
```

```
# 查看数据
head(dat)
```

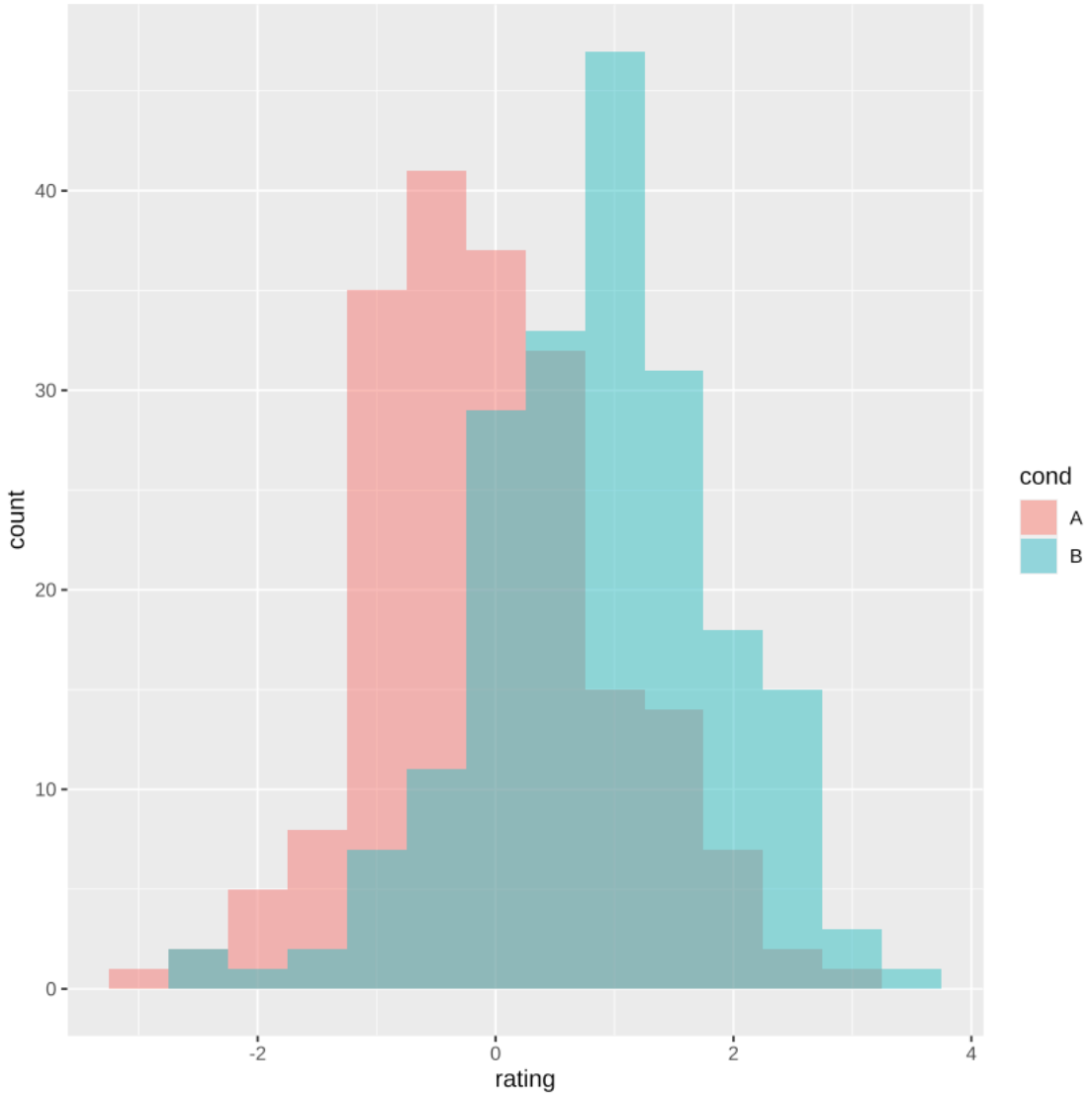
A data.frame: 6 × 2

	cond <fct>	rating <dbl>
1	A	-1.2070657
2	A	0.2774292
3	A	1.0844412
4	A	-2.3456977
5	A	0.4291247
6	A	0.5060559

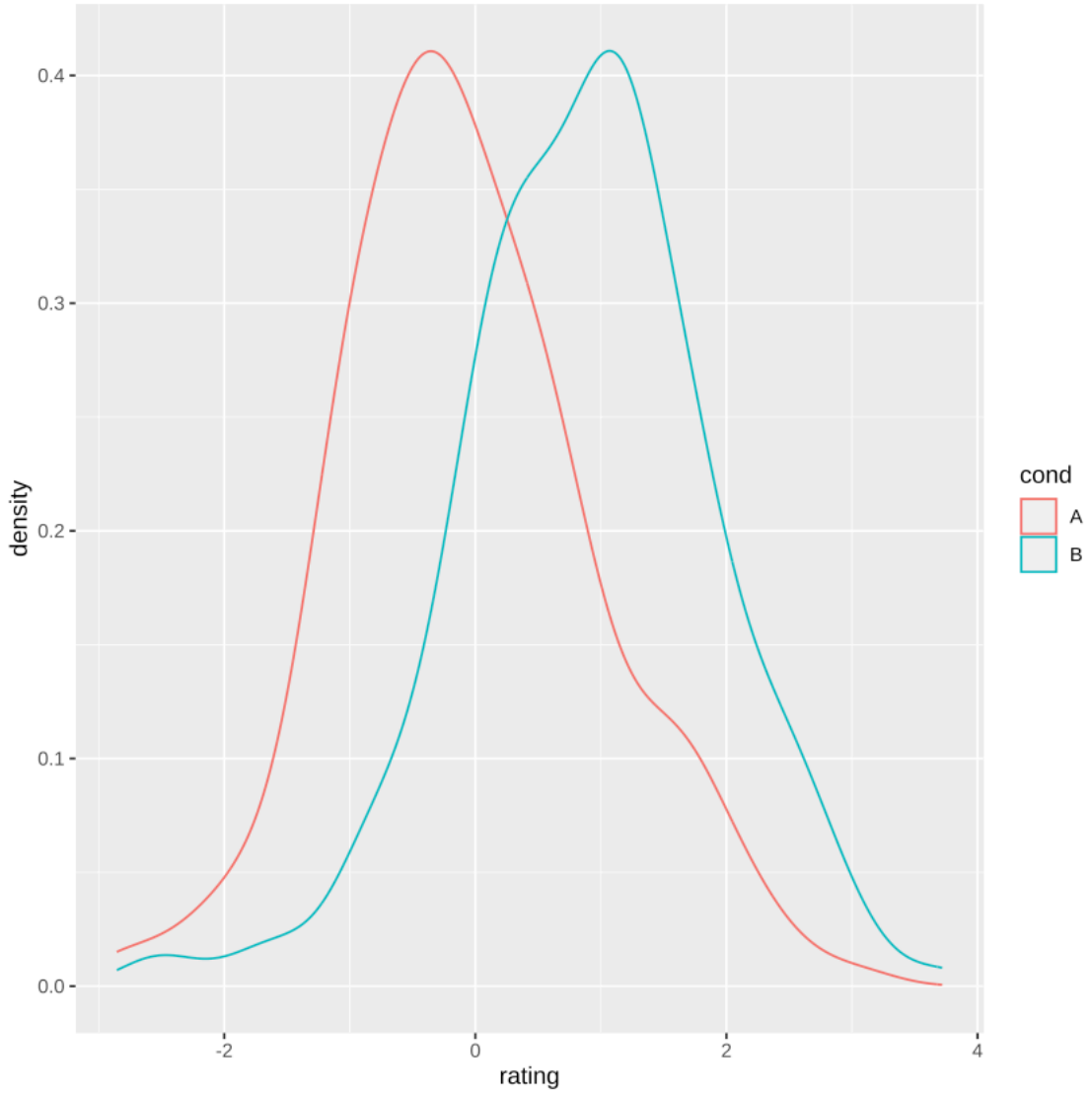
[124]:
```
# 直方图与核密度曲线重叠
ggplot(dat, aes(x=rating)) +
    geom_histogram(aes(y=after_stat(density)),      # 这里直方图以 density (密度) 为 y 轴
                   binwidth=.5,
                   colour="black", fill="white") +
    geom_density(alpha=.2, fill="#FF6666")  # 重合部分透明填充
```

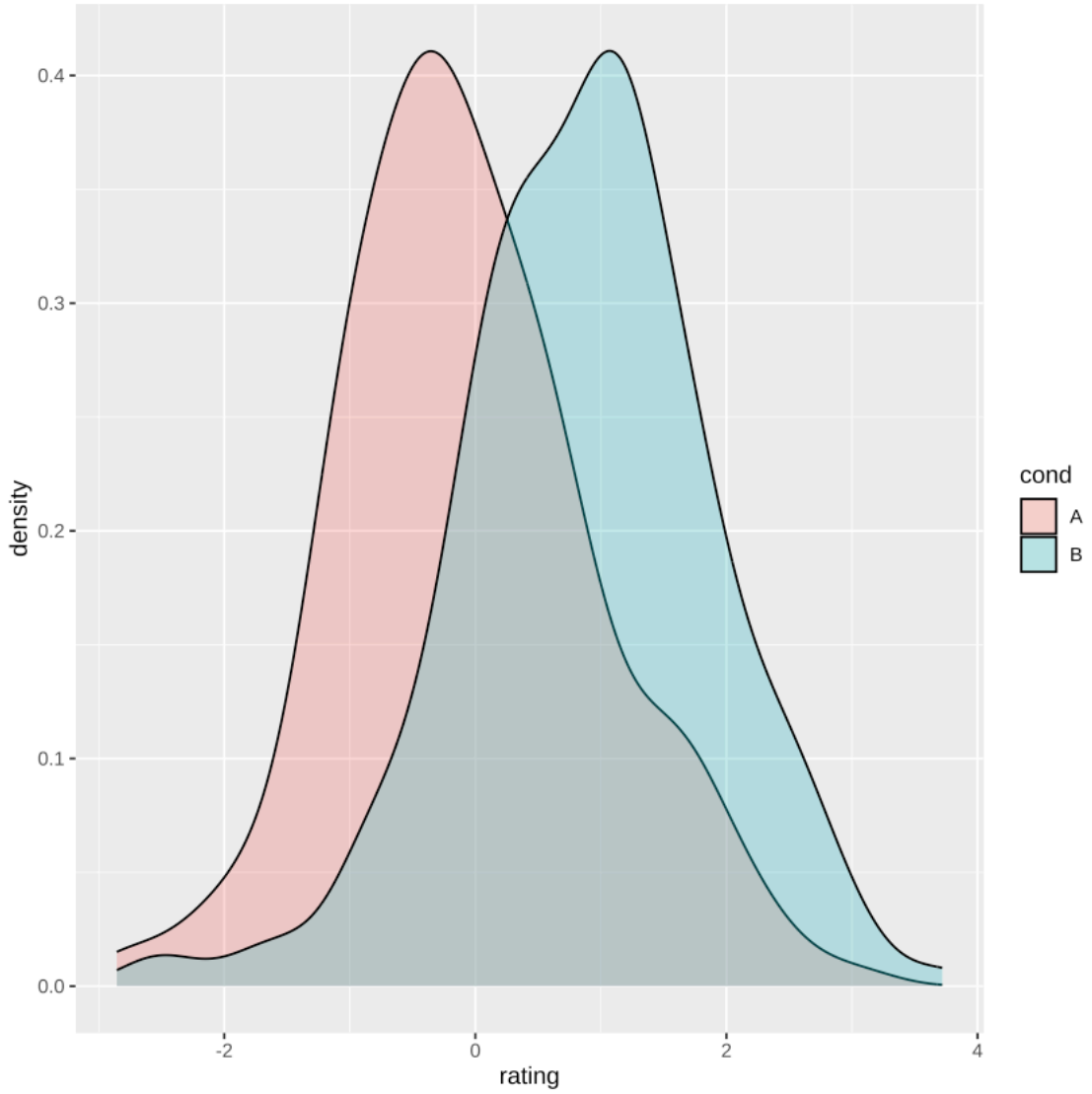


[125]:
```
# 重叠直方图
ggplot(dat, aes(x = rating, fill = cond)) + geom_histogram(binwidth = 0.5,␣
↪alpha = 0.5,
    position = "identity")  # identity 表示将每个对象直接显示在图中，条形会彼此重
叠。
```



[126]:
```
# 密度图
ggplot(dat, aes(x = rating, colour = cond)) + geom_density()
# 半透明填充的密度图
ggplot(dat, aes(x = rating, fill = cond)) + geom_density(alpha = 0.3)
```


0.4
0.3
0.2
0.1
0.0
density
-2
0
2
4
rating
cond
A
B



5.5.2 添加均值线

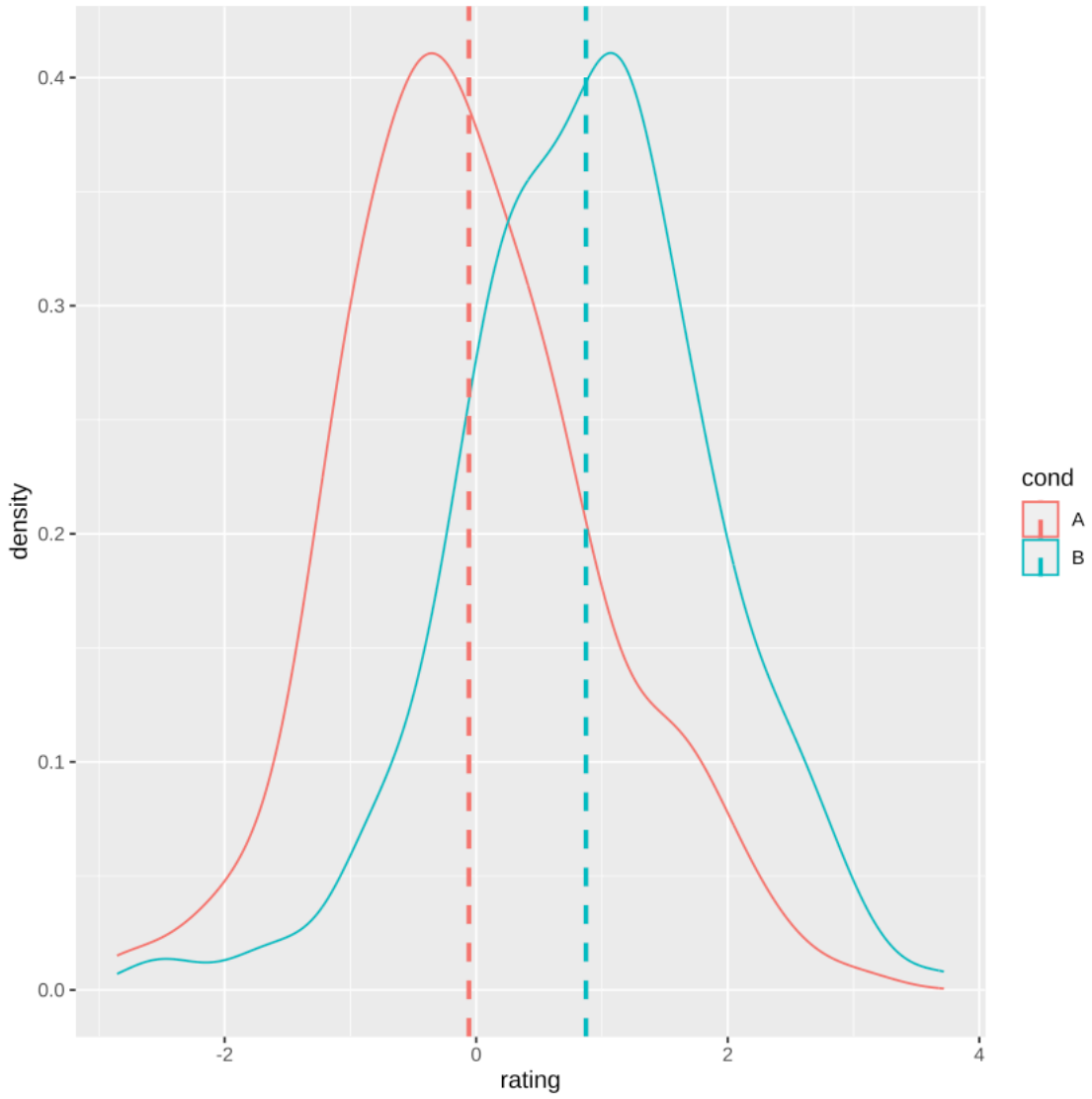
[127]:
```
# 求均值
library(plyr)
cdat <- ddply(dat, "cond", summarise, rating.mean = mean(rating))
cdat

# 给密度图添加均值线
```

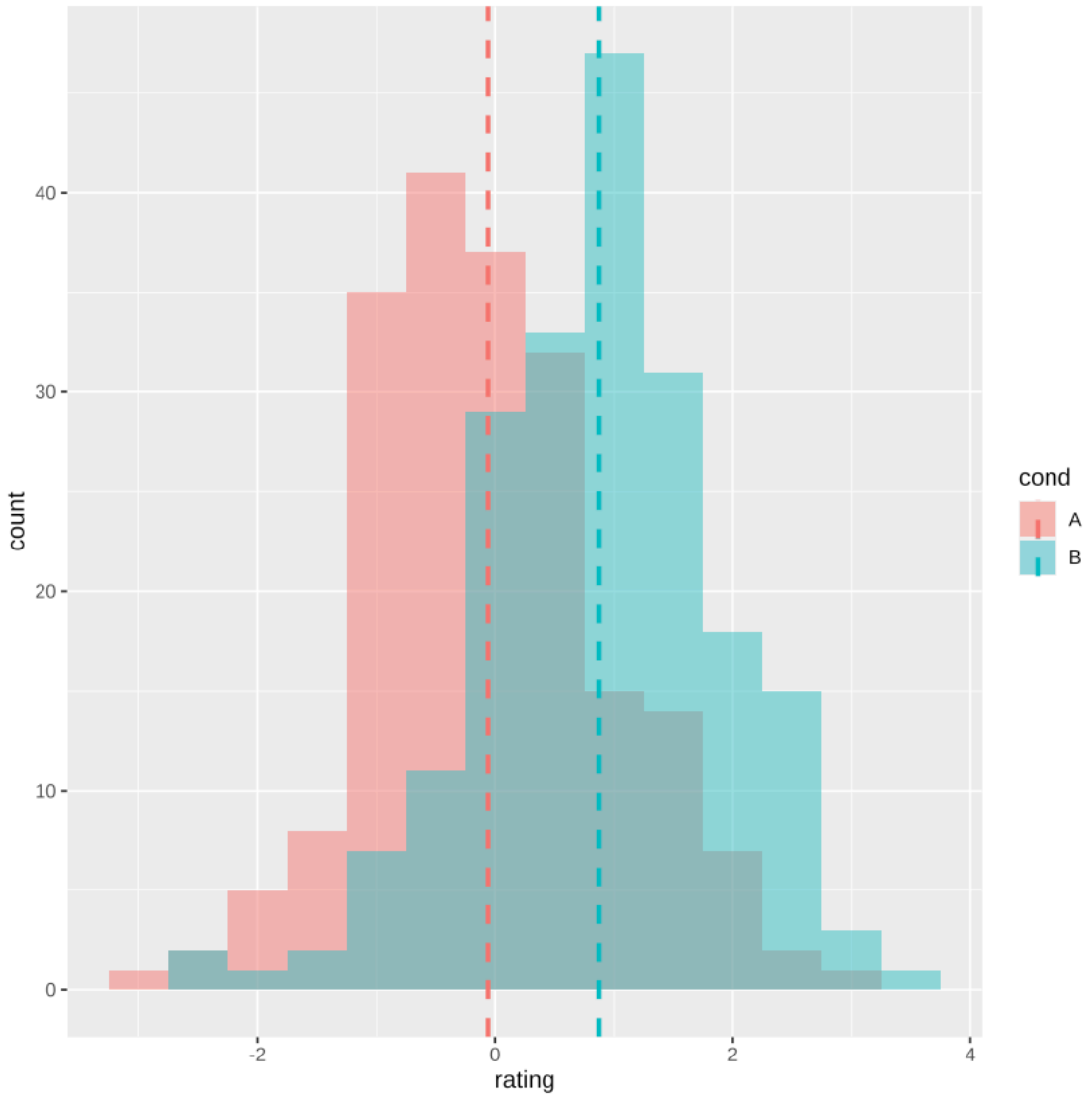
```
ggplot(dat, aes(x = rating, colour = cond)) + geom_density() + geom_vline(data␣
 ↪= cdat,
    aes(xintercept = rating.mean, colour = cond), linetype = "dashed", size = 1)

# 给重叠直方图添加均值线
ggplot(dat, aes(x = rating, fill = cond)) + geom_histogram(binwidth = 0.5,␣
 ↪alpha = 0.5,
    position = "identity") + geom_vline(data = cdat, aes(xintercept = rating.
 ↪mean,
    colour = cond), linetype = "dashed", size = 1)
```

```
------------------------------------------------------------------------------

You have loaded plyr after dplyr - this is likely to cause problems.
If you need functions from both plyr and dplyr, please load plyr first, then
dplyr:
library(plyr); library(dplyr)

------------------------------------------------------------------------------


载入程辑包：‘plyr’


The following objects are masked from ‘package:dplyr’ :

    arrange, count, desc, failwith, id, mutate, rename, summarise,
    summarize


The following object is masked from ‘package:purrr’ :

    compact
```

A data.frame: 2 × 2

cond	rating.mean
<fct>	<dbl>
A	-0.05775928
B	0.87324927





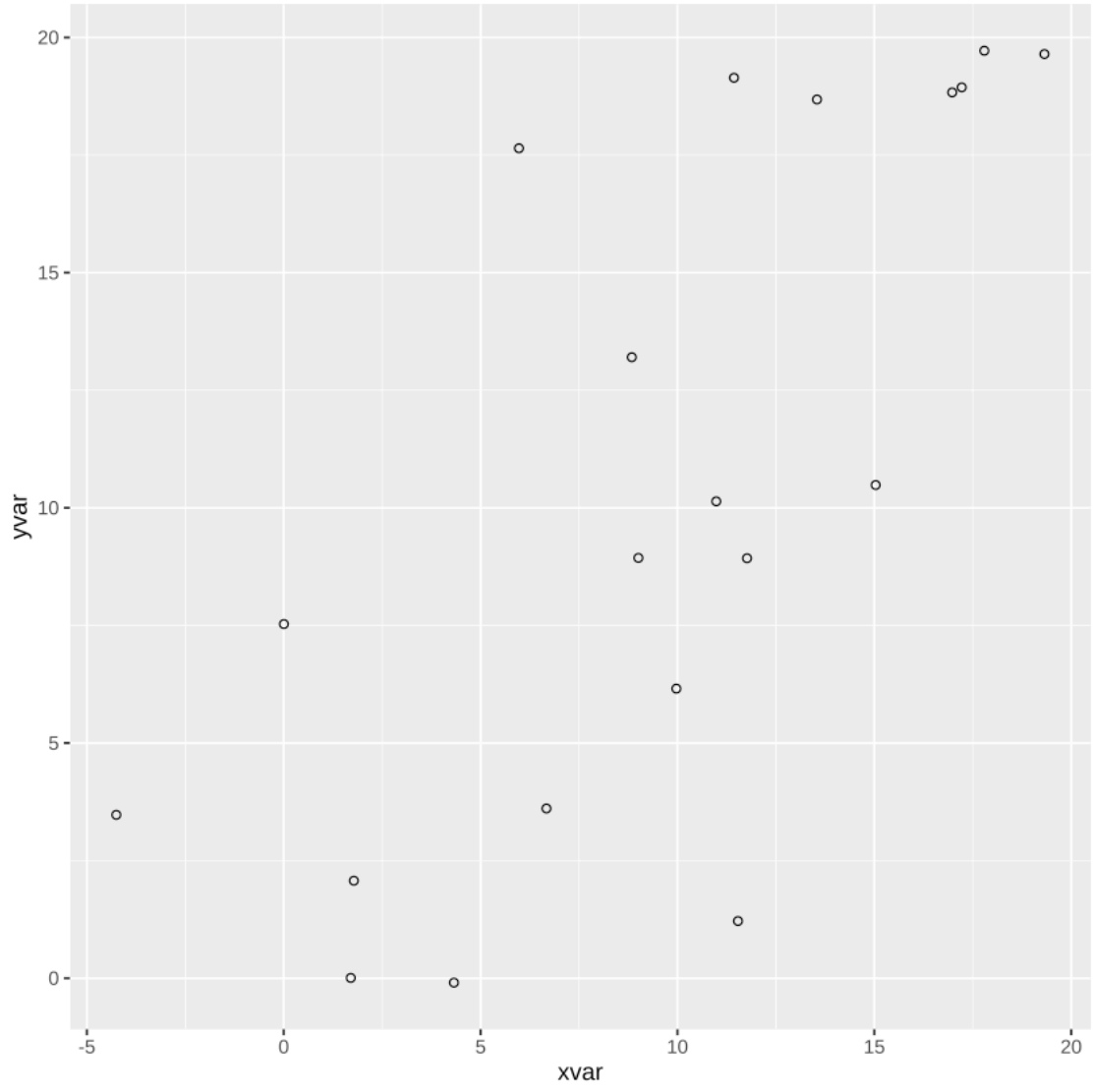
5.5.3 散点图

[128]:
```
set.seed(955)
# 创建一些噪声数据
dat <- data.frame(cond = rep(c("A", "B"), each = 10), xvar = 1:20 +
  rnorm(20, sd = 3), yvar = 1:20 + rnorm(20, sd = 3))
head(dat)
```

A data.frame: 6 × 3

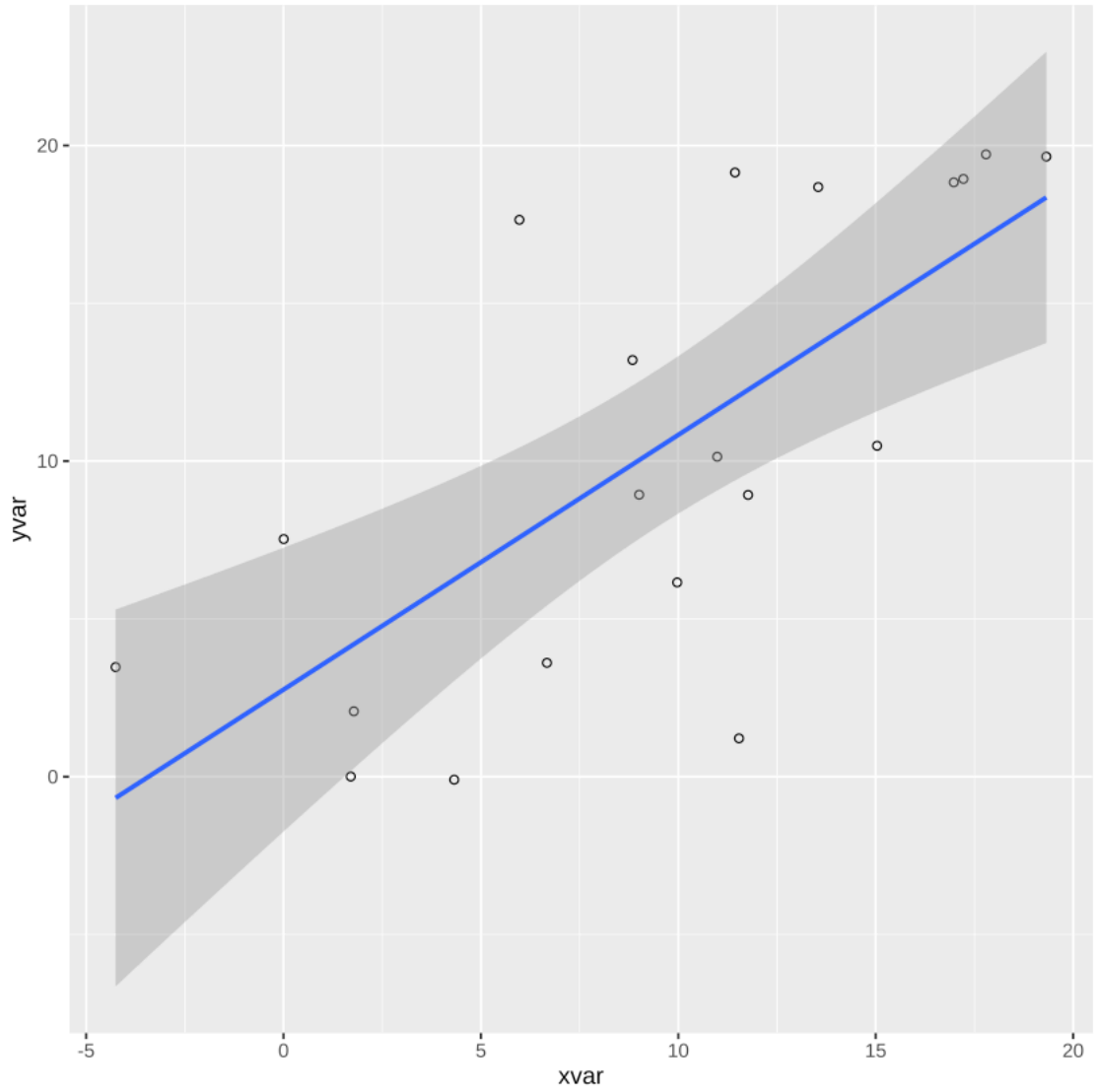
	cond	xvar	yvar
	<chr>	<dbl>	<dbl>
1	A	-4.252354	3.473157275
2	A	1.702318	0.005939612
3	A	4.323054	-0.094252427
4	A	1.780628	2.072808278
5	A	11.537348	1.215440358
6	A	6.672130	3.608111411

[129]:
```
ggplot(dat, aes(x = xvar, y = yvar)) + geom_point(shape = 1)  # 使用空心圆
```



[130]:
```
# 默认包含 95% 置信区间
ggplot(dat, aes(x=xvar, y=yvar)) +
    geom_point(shape=1) +    # 使用空心圆
    geom_smooth(method=lm)   # 添加回归线
```

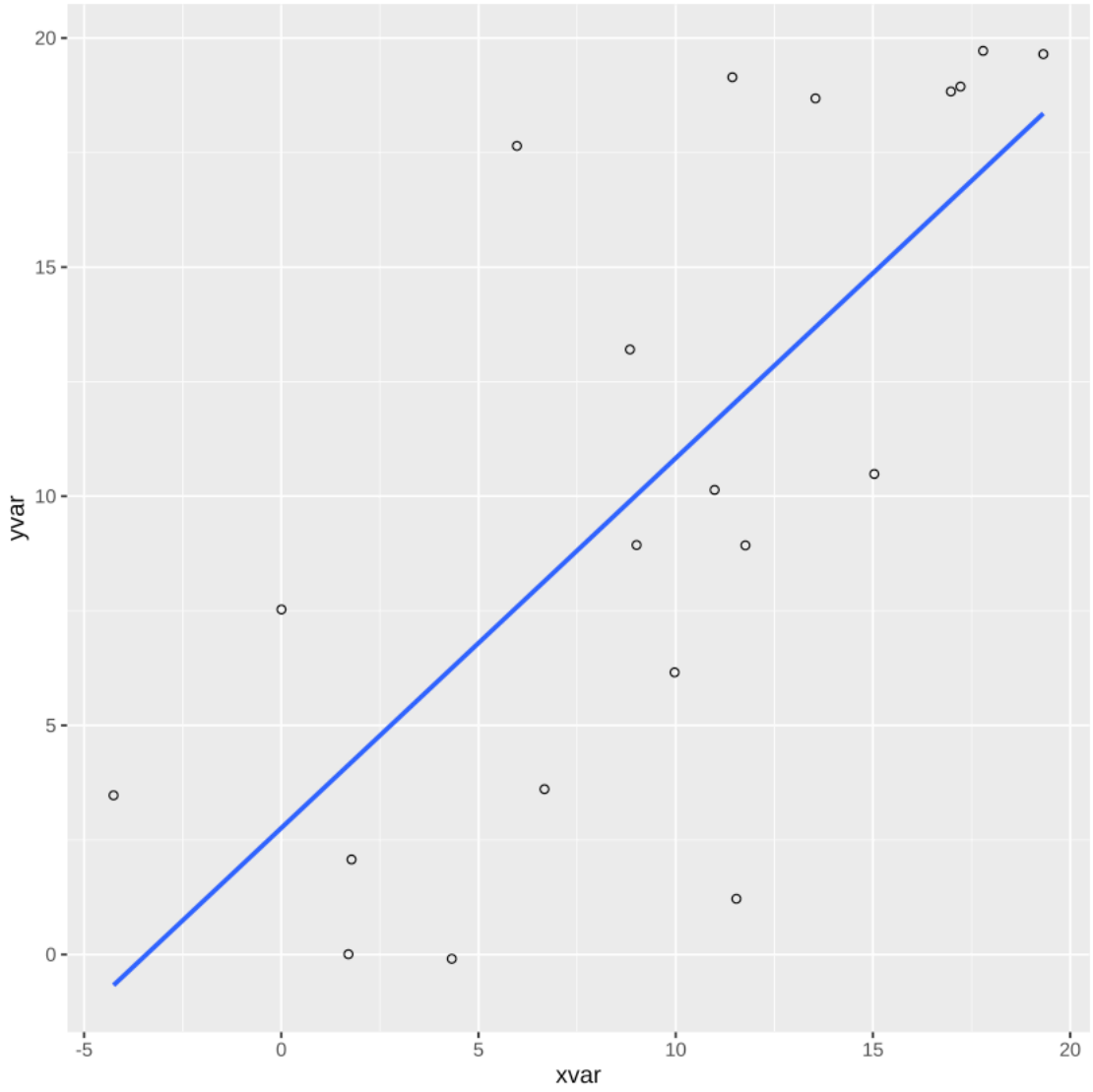
```
`geom_smooth()` using formula = 'y ~ x'
```



[131]:
```
# (不包含 95% 置信区间)
ggplot(dat, aes(x=xvar, y=yvar)) +
    geom_point(shape=1) +    # 使用空心圆
```

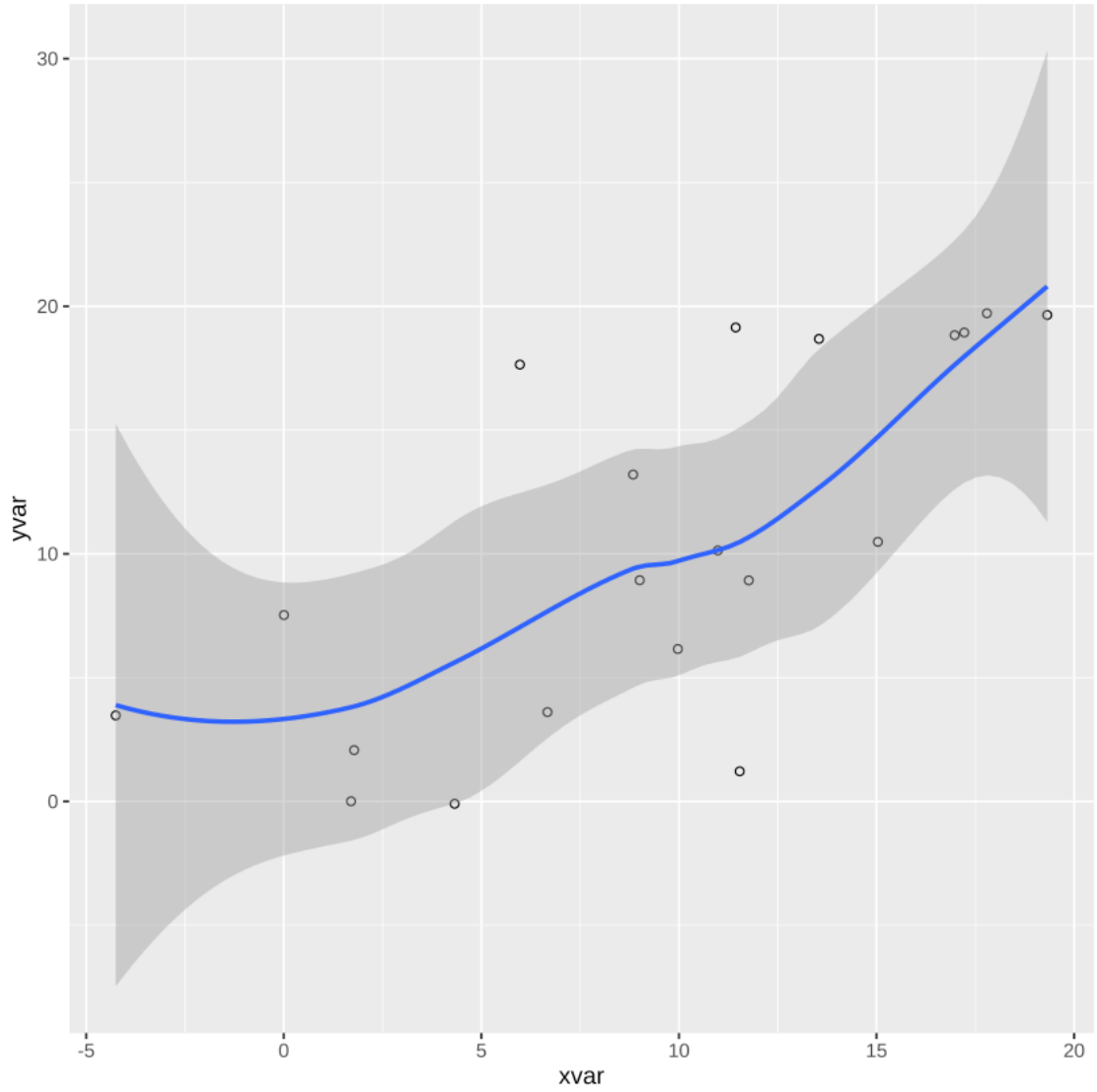
```
    geom_smooth(method=lm,   # 添加回归线
                se=FALSE)    # 不加置信区域
```

```
`geom_smooth()` using formula = 'y ~ x'
```



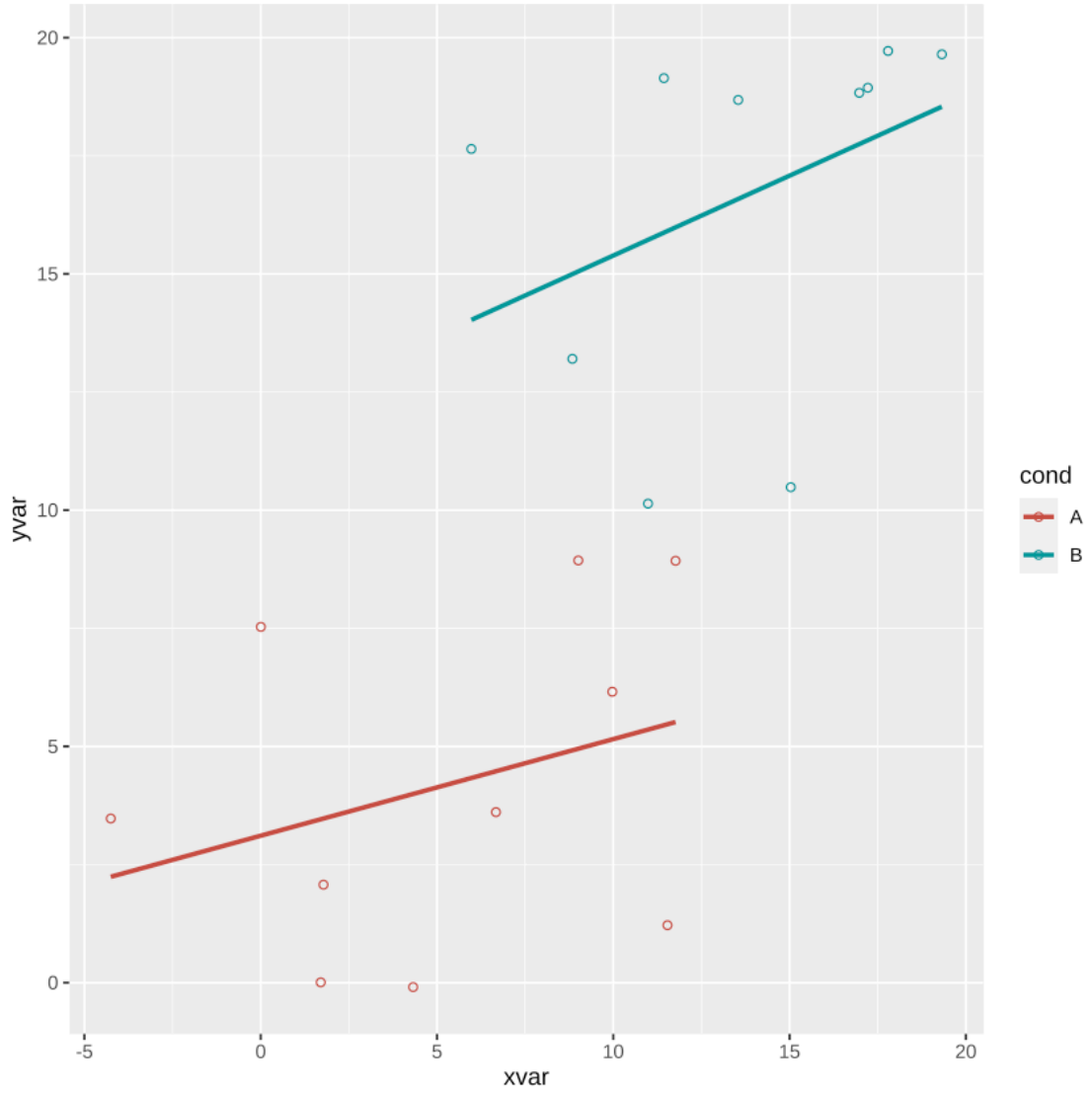
[132]:
```
ggplot(dat, aes(x=xvar, y=yvar)) +
    geom_point(shape=1) +    # 使用空心圆
    geom_smooth()            # 添加带置信区间的平滑拟合曲线
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



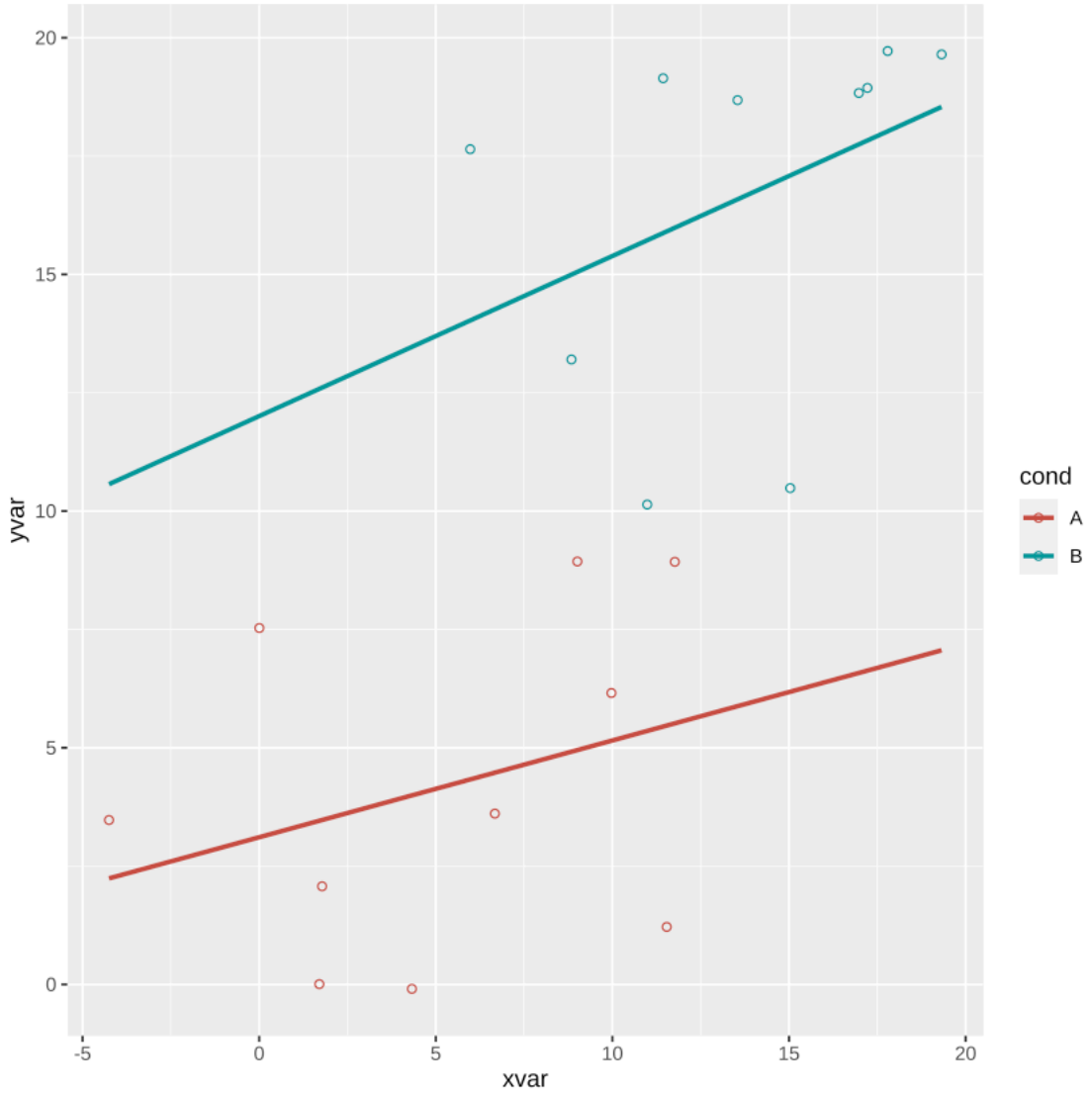
[133]:
```
# 同上，但这里带了回归线
ggplot(dat, aes(x=xvar, y=yvar, color=cond)) +
    geom_point(shape=1) +
    scale_colour_hue(l=50) + # 使用稍暗的调色板
    geom_smooth(method=lm,
                se=FALSE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



[134]:
```
# 拓展回归线到数据区域之外（带预测效果）
ggplot(dat, aes(x=xvar, y=yvar, color=cond)) + geom_point(shape=1) +
    scale_colour_hue(l=50) +
    geom_smooth(method=lm,
                se=FALSE,
                fullrange=TRUE)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



5.6 组合多图

5.6.1 multiplot()

最简单的方法就是使用 multiplot() 函数。

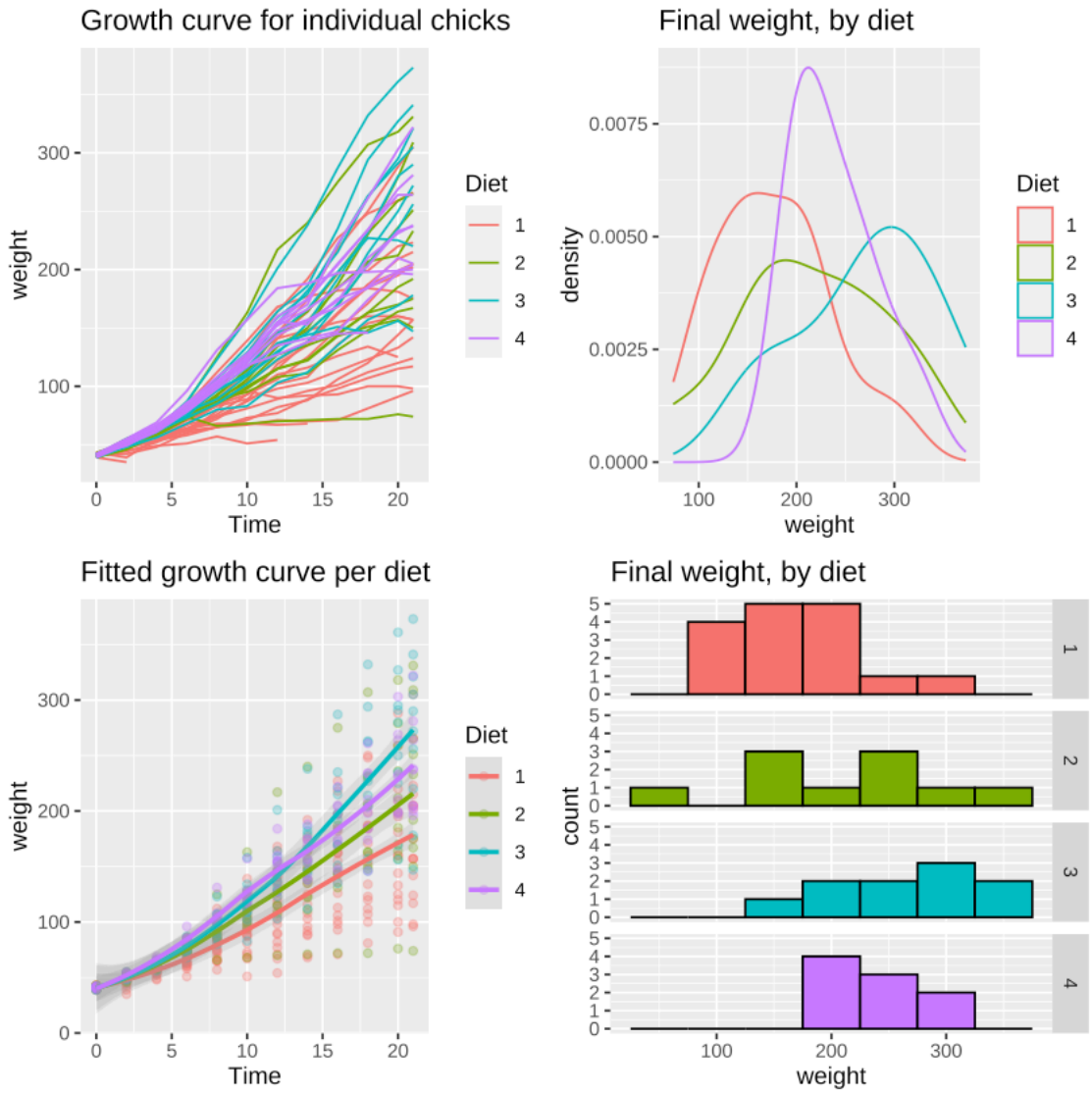
multiplot() 函数可以将任意数量的图像对象作为参数，或者可以构建一个图像对象列表传递到该函数的 plotlist 参数中。

[135]:
```r
# 以下例子使用的是 ggplot2 包中自带的 Chickweight

# 数据集 第一幅图像
p1 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet,
  group = Chick)) + geom_line() + ggtitle("Growth curve for individual chicks")

# 第二幅图像
p2 <- ggplot(ChickWeight, aes(x = Time, y = weight, colour = Diet)) +
  geom_point(alpha = 0.3) + geom_smooth(alpha = 0.2, size = 1) +
  ggtitle("Fitted growth curve per diet")

# 第三幅图像
p3 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight,
  colour = Diet)) + geom_density() + ggtitle("Final weight, by diet")

# 第四幅图像
p4 <- ggplot(subset(ChickWeight, Time == 21), aes(x = weight,
  fill = Diet)) + geom_histogram(colour = "black", binwidth = 50) +
  facet_grid(Diet ~ .) + ggtitle("Final weight, by diet") +
  theme(legend.position = "none")  # 为了避免冗余，这里不添加图例
```

[136]:
```r
library(Rmisc)
multiplot(p1, p2, p3, p4, cols = 2)
```

```
载入需要的程辑包：lattice
```

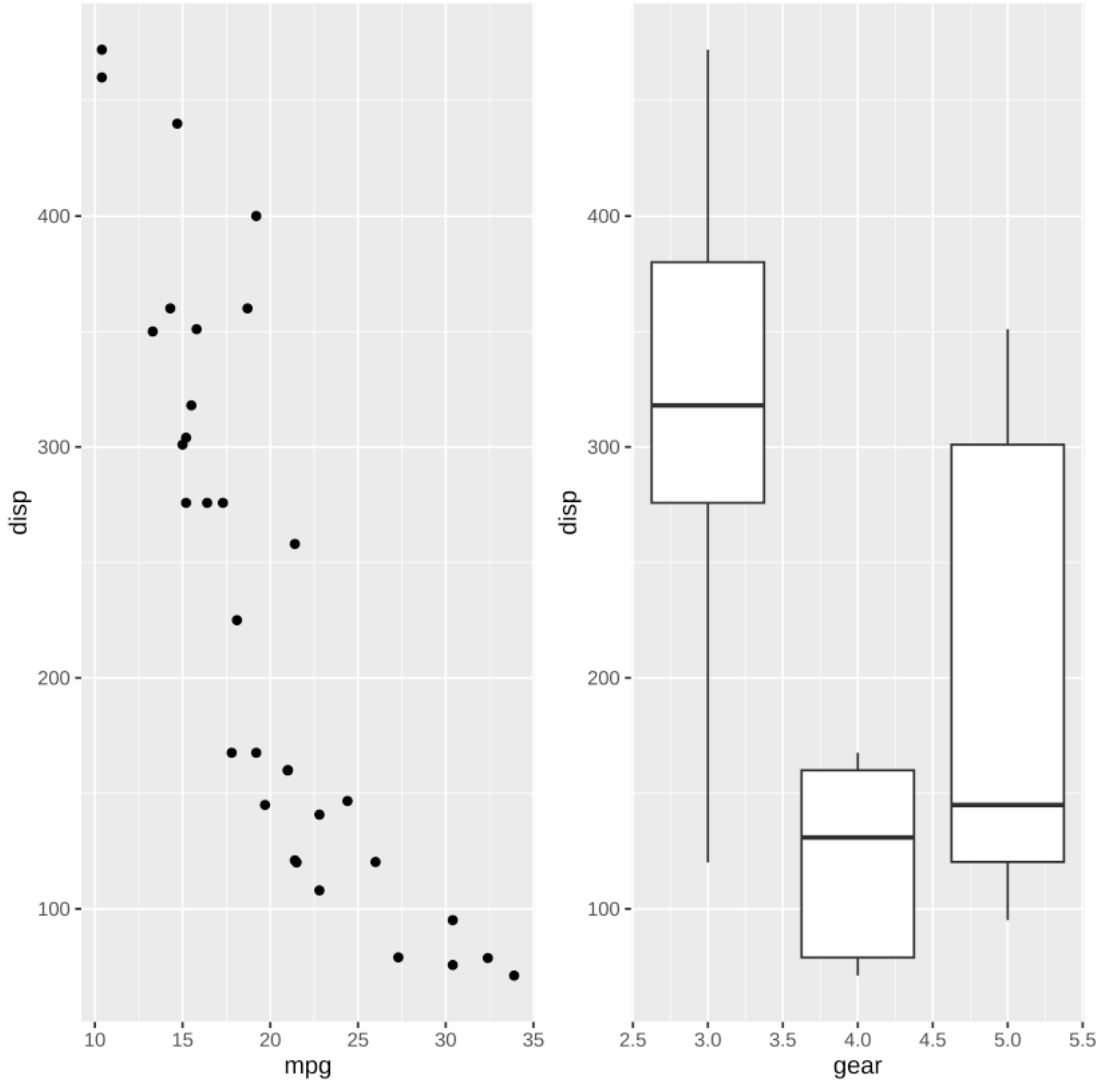
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



5.6.2 patchwork

[137]:
```
library(ggplot2)
library(patchwork)

p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
```

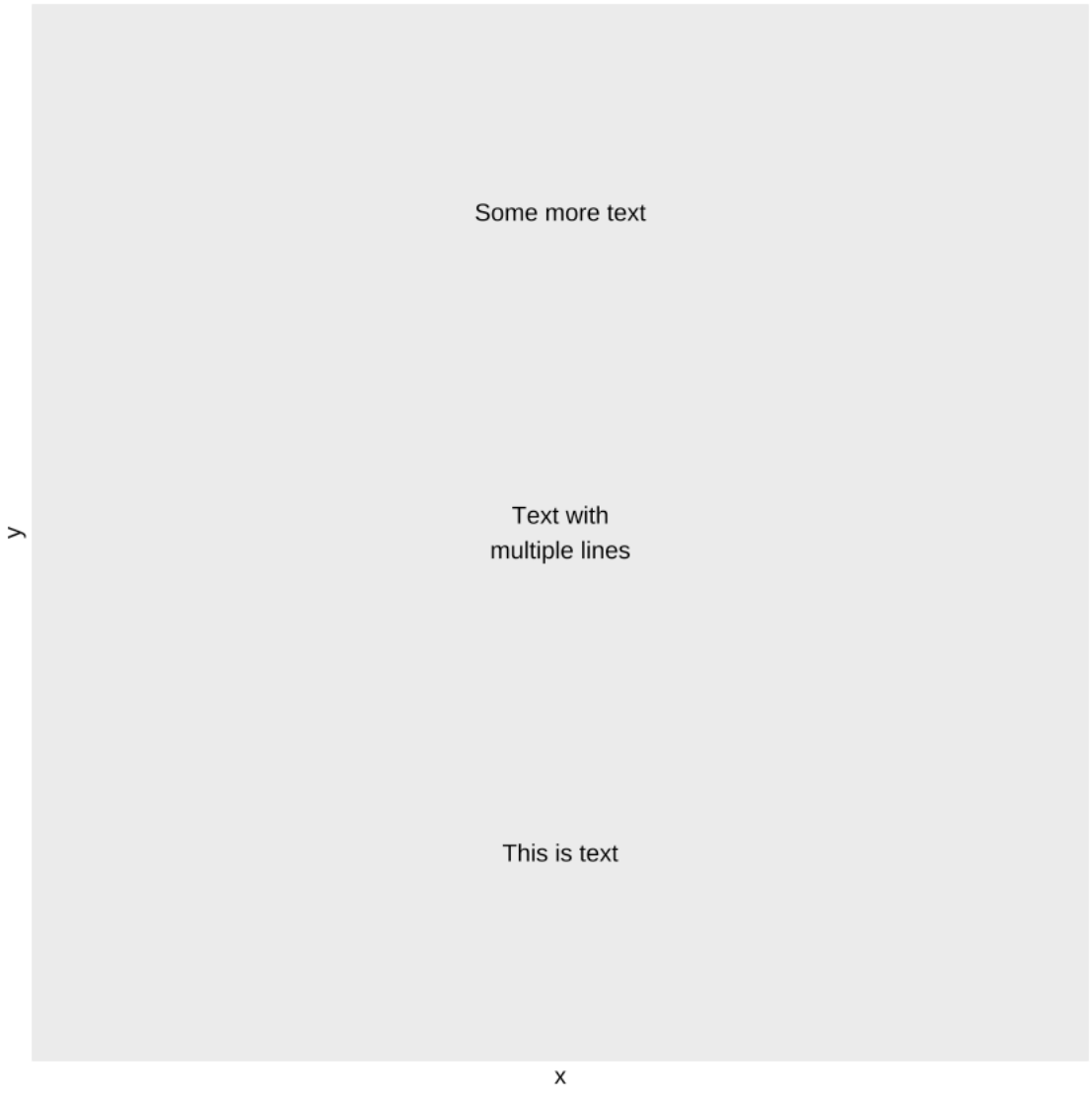
```
p1 + p2
```



5.7 字体

[138]:
```
dat <- data.frame(y = 1:3, text = c("This is text", "Text with\nmultiple lines",
  "Some more text"))

library(ggplot2)
```

```
p <- ggplot(dat, aes(x = 1, y = y)) + scale_y_continuous(limits = c(0.5,
  3.5), breaks = NULL) + scale_x_continuous(breaks = NULL)

p + geom_text(aes(label = text))
```

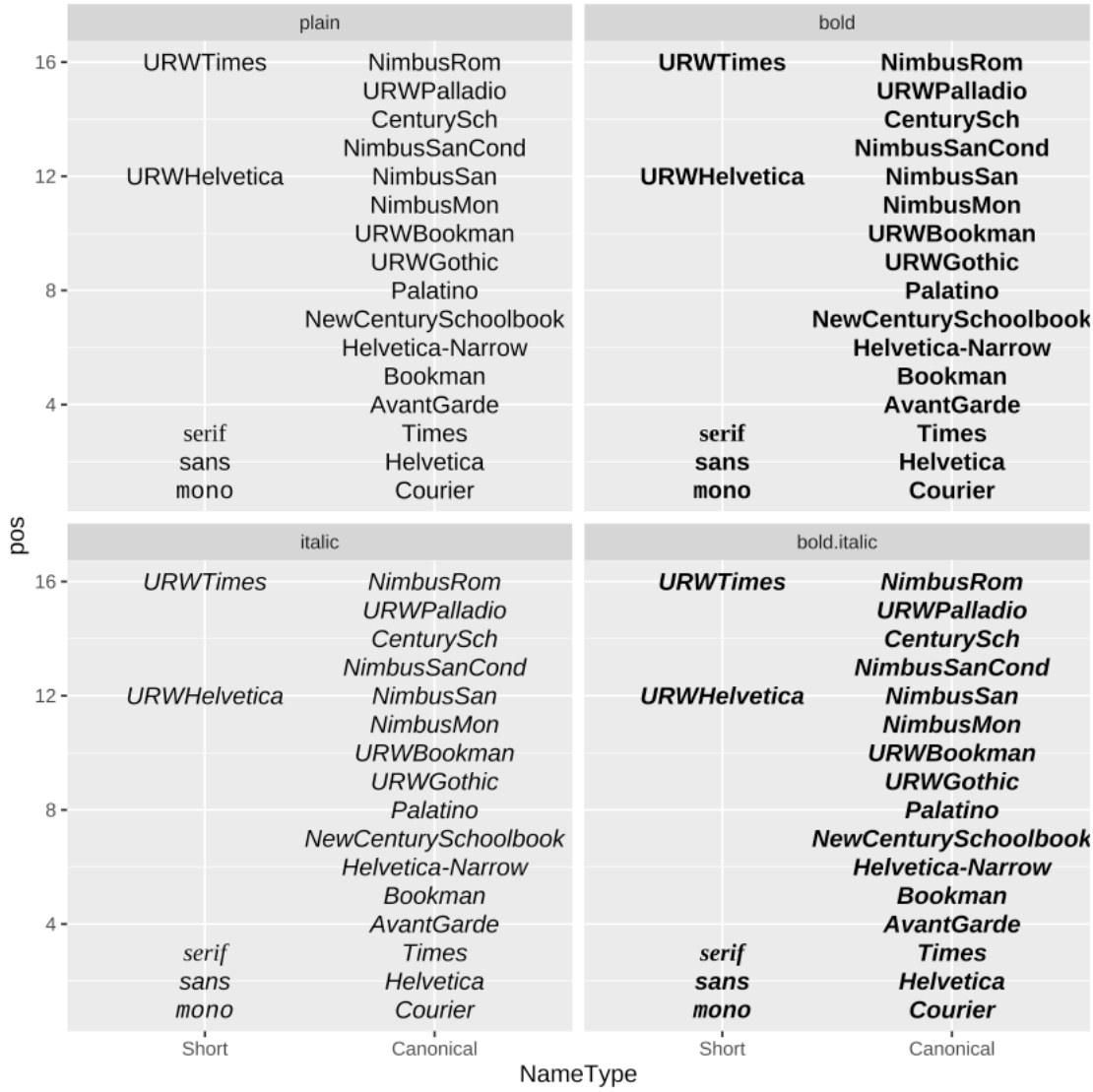


[139]:
```
options(warn=-1)
p + geom_text(aes(label = text), family = "Times", fontface = "italic",␣
  ↪lineheight = 0.8) +
```

```
  annotate(geom = "text", x = 1, y = 1.5, label = "Annotation text", colour =␣
↪"red",
       size = 7, family = "Courier", fontface = "bold", angle = 30)
```

[140]:
```
fonttable <- read.table(header = TRUE, sep = ",", stringsAsFactors = FALSE,
  text = "
Short,Canonical
mono,Courier
sans,Helvetica
```

```
serif,Times
,AvantGarde
,Bookman
,Helvetica-Narrow
,NewCenturySchoolbook
,Palatino
,URWGothic
,URWBookman
,NimbusMon
URWHelvetica,NimbusSan
,NimbusSanCond
,CenturySch
,URWPalladio
URWTimes,NimbusRom
")

fonttable$pos <- 1:nrow(fonttable)

library(reshape2)
fonttable <- melt(fonttable, id.vars = "pos", measure.vars = c("Short",
  "Canonical"), variable.name = "NameType", value.name = "Font")

# 创建一个分面形式的图表。确保因子的顺序是正确的
facetable <- data.frame(Face = factor(c("plain", "bold",
  "italic", "bold.italic"), levels = c("plain", "bold",
  "italic", "bold.italic")))

fullfonts <- merge(fonttable, facetable)

options(warn=-1)
pf <- ggplot(fullfonts, aes(x = NameType, y = pos)) + geom_text(aes(label =␣
 ↪Font,
  family = Font, fontface = Face)) + facet_wrap(~Face,
  ncol = 2)
pf
```



5.8 交互式图表

[141]:
```
library(plotly) # 将 ggplot 图表转换为交互式图表 关闭悬停时的报错显示:
↪`ggplotly(plot, tooltip = NULL)`
```

载入程辑包：‘plotly’

```
The following objects are masked from 'package:plyr':

    arrange, mutate, rename, summarise


The following object is masked from 'package:rio':

    export


The following object is masked from 'package:ggplot2':

    last_plot


The following object is masked from 'package:stats':

    filter


The following object is masked from 'package:graphics':

    layout
```

[142]:
```r
# 使用 gather 函数将数据从宽格式变为长格式
iris_long <- iris %>%
    gather(key = "measurement", value = "value", Sepal.Length:Petal.Width)
iris_long %>% head(10)
# 计算每朵鸢尾花的大小
iris <- iris %>%
    mutate(size = Petal.Length * Petal.Width)
```

A data.frame: 10 × 3

	Species <fct>	measurement <chr>	value <dbl>
1	setosa	Sepal.Length	5.1
2	setosa	Sepal.Length	4.9
3	setosa	Sepal.Length	4.7
4	setosa	Sepal.Length	4.6
5	setosa	Sepal.Length	5.0
6	setosa	Sepal.Length	5.4
7	setosa	Sepal.Length	4.6
8	setosa	Sepal.Length	5.0
9	setosa	Sepal.Length	4.4
10	setosa	Sepal.Length	4.9

[143]:
```
plot <- ggplot(iris, aes(x = Species, y = size, fill = Species)) +␣
↪geom_boxplot() + theme_bw() +
    labs(title = "不同品种鸢尾花的大小") + xlab("Species") + ylab("Size")
ggplotly(plot, tooltip = NULL)
```

```
HTML widgets cannot be represented in plain text (need html)
```

[144]:
```
plot2 <-
ggplot(iris_long, aes(x = measurement, y = value, fill = Species)) +␣
↪geom_boxplot() +
    facet_grid(. ~ Species) + theme_bw() + labs(title = "不同品种鸢尾花的花瓣长度
和宽度") +
    xlab("Measurement") + ylab("Value")
ggplotly(plot2, tooltip = NULL)
```

```
HTML widgets cannot be represented in plain text (need html)
```

6 Split - Apply - Combine

[145]:
```
players_scores <- data.frame(player = rep(c("Tom", "Dick", "Jim"), times = c(2,␣
↪5,
    3)), score = round(runif(10, 1, 100), -1))
players_scores
```

A data.frame: 10 × 2

player	score
<chr>	<dbl>
Tom	10
Tom	90
Dick	20
Dick	100
Dick	70
Dick	50
Dick	60
Jim	20
Jim	10
Jim	50

相同的 score 是同一个分组，填充到同一个列表项中：

[146]:
```
# 分组数据
(scores_by_player <- with(players_scores,split(score,player)))
```

$Dick 1. 20 2. 100 3. 70 4. 50 5. 60

$Jim 1. 20 2. 10 3. 50

$Tom 1. 10 2. 90

当数据分割之后，对每个分组计算平均分。使用 lapply() 函数，对于每个列表项，应用 mean() 函数，计算单个列表项的平均值，例如：

[147]:
```
list_mean_by_player <- lapply(scores_by_player,mean)
```

[148]:
```
# lapply() 函数返回的结果是一个列表对象，每一个列表项都是一个向量，
# 因此可以使用 unlist() 函数，把列表转换为向量，例如：
unlist(list_mean_by_player)
```

Dick 60 **Jim** 26.6666666666667 **Tom** 50

6.1 apply 家族

在 apply 家族函数中，每个函数都用于特定的数据类型：* apply 函数只能用于矩阵，* lapply 函数能够用于向量和列表（list），其工作原理是把一个函数应用于一个列表中的每个元素上，并且把结果作为列表返回；* sapply 处理列表，返回向量。* mapply 函数，把调用的函数应用到多个列表

的每一个元素中。* tapply 函数用于分组聚合运算，在研究数据时，有时需要对数据按照特定的字段进行分组，然后统计各个分组的数据，这就是 SQL 语法中的分组聚合。

[149]:
```
# 使用 tapply() 函数一次完成 “拆分 - 应用 - 合并” 三个步骤，一气呵成：
with(players_scores,tapply(score,player,mean))
```

Dick 60 **Jim** 26.6666666666667 **Tom** 50

tapply() 函数常用的参数共有三个，第一个参数是数据框对象或向量，第二个参数是因子列表，也就是分组字段，第三个参数是指对单个分组应用的函数：

by() 函数和 aggregate() 函数是 tapply() 函数的包装函数，功能相同，接口稍微不同。* by(data, INDICES, FUN, ..., simplify = TRUE) * aggregate(x, by, FUN, ..., simplify = TRUE, drop = TRUE)

[150]:
```
# 使用 daply() 函数计算每个 player 的平均得分
# 在示例中，daply() 函数返回的类型是 list，通过 unlist() 函数转换为向量。
unlist(daply(players_scores,.(player),summarize,varScore=mean(score)))
```

Dick 60 **Jim** 26.6666666666667 **Tom** 50

6.2 plyr

plyr 包是 apply 家族函数的升级版本，使用 plyr 包可以实现：在一个函数内同时完成 “Split - Apply - Combine”，并且，plyr 包实现 R 类型（vector, list, data.frame）之间的分组变换，基本上可以取代 Base 包中的 apply 家族函数。

plyr 包对核心函数的命名采用统一的格式：**ply，所有的函数名都由 5 个字符组成，且最后三个字符是 ply，函数名的第一个字符代表输入数据的类型，第二个字符代表输出数据的类型，R 类型的简写是：

- d：data.frame
- l：list
- a：array，vector，matrix
- r：代表 replicate，重复多次
- m：多输入
- _：舍弃输出结果

其他操作函数 * each()：each(min, max) 等价于 function(x) c(min = min(x), max = max(x))。* colwise()：colwise(median) 将计算列的中位数。* arrange()：超级顺手的函数，可以方便的给

dataframe 排序。* rename()：又是一个 handy 的函数，按变量名而不是变量位置重命名。* count()：返回 unique 值，等价于 length(unique(**))。* match_df()：方便的配合 count() 等，选出符合条件的行，有点像 merge(…,all=F) 的感觉。* join(): 对于习惯 SQL 的童鞋，可能比 merge() 用起来更顺手吧（当然也更快一点），不过灵活性还是比不上 merge()。

6.2.1 plyr::ddply

plyr 包中最常用的函数是 ddply() 函数，该函数对数据框进行操作，对每一列调用一个函数，并返回数据框类型：`ddply(.data, .variables, .fun = NULL, ...)`

[151]:
```
# 在 ddply 函数或 R 中较新的 plyr::ddply 函数的上下文中，.(color) 用于指定一个变量或列，您希望根据该变量或列对数据进行分组，以便进行后续汇总
plyr::ddply(diamonds, .(color), summarize, avg_price = mean(price), avg_carat =␣
↪mean(carat))
```

A data.frame: 7 × 3

color <ord>	avg_price <dbl>	avg_carat <dbl>
D	3169.954	0.6577948
E	3076.752	0.6578667
F	3724.886	0.7365385
G	3999.136	0.7711902
H	4486.669	0.9117991
I	5091.875	1.0269273
J	5323.818	1.1621368

6.2.2 plyr::each 函数

plyr 包的 each() 函数，能够把多个函数整合到一个函数中，每一个函数必须只能返回一个数值

[152]:
```
aggregate(cbind(price, carat) ~ cut + color, diamonds, plyr::each(mean, sum))␣
↪%>%
  head(10)
```

A data.frame: 10 × 4

	cut	color	price	carat
	<ord>	<ord>	<dbl[,2]>	<dbl[,2]>
1	Fair	D	4291.061, 699443	0.9201227, 149.98
2	Good	D	3405.382, 2254363	0.7445166, 492.87
3	Very Good	D	3470.467, 5250817	0.6964243, 1053.69
4	Premium	D	3631.293, 5820962	0.7215471, 1156.64
5	Ideal	D	2629.095, 7450854	0.5657657, 1603.38
6	Fair	E	3682.312, 824838	0.8566071, 191.88
7	Good	E	3423.644, 3194260	0.7451340, 695.21
8	Very Good	E	3214.652, 7715165	0.6763167, 1623.16
9	Premium	E	3538.914, 8270443	0.7177450, 1677.37
10	Ideal	E	2597.550, 10138238	0.5784012, 2257.50

6.2.3 plyr::rename

[153]:
```
plyr::rename(mtcars, c(disp = "displacement")) %>%
    head(10)
```

A data.frame: 10 × 11

	mpg	cyl	displacement	hp	drat	wt	qsec
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02
Valiant	18.1	6	225.0	105	2.76	3.460	20.22
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30

6.2.4 plyr::arrange

Order a data frame by its colums

[154]:
```
mtcars %>%
    plyr::arrange(cyl, disp) %>%
```

```
    head(10)
```

A data.frame: 10 × 11

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>	am <dbl>	g <dbl>
1	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4
2	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4
3	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4
4	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4
5	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5
6	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4
7	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3
8	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5
9	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4
10	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4

6.2.5 plyr::mutate

该函数和 transfrom 函数十分相似，不过，mutate() 函数是递进式的，这使得后期的转换可以使用早期创建的变量。

[155]:
```
# Things transform can't do
plyr::mutate(mtcars, mpg1 = mpg^2, mpg2 = mpg1/mpg) %>%
    head(10)
```

A data.frame: 10 × 13

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1

6.2.6 plyr::name_rows

在设计时，没有 plyr 函数会保留行名称 (row names)。如果想保留行名称，可以使用 name_rows() 把行名称转换为显式的列值，在执行为相应的 plyr 操作之后，再使用 name_rows 把列值转换为行名称。

[156]:
```
plyr::name_rows(mtcars) %>%
    head(10)
```

A data.frame: 10 × 12

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	g
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<
1	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4
2	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4
3	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4
4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3
5	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3
6	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3
7	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3
8	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4
9	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4
10	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4

参数 df： 数据框对象，拥有 rownames，或者显式的列名.rownames

7 实战：iris 数据分析

演示如何使用 R 语言中的 ggplot2、dplyr 和 tidyr 库进行数据分析。我们将使用一个真实的数据集，并进行数据导入、清洗、转换、分析和可视化等多个任务。通过本部分的演示，读者可以更好地理解 ggplot2、dplyr 和 tidyr 的相关知识，并在相似的数据分析任务中应用它们。

我们的任务是分析一份 R 语言自带的花的数据集。数据集包含了：

- Sepal.Length：萼片长度，以厘米为单位
- Sepal.Width：萼片宽度，以厘米为单位
- Petal.Length：花瓣长度，以厘米为单位
- Petal.Width：花瓣宽度，以厘米为单位
- Species：鸢尾花的品种，包括三个类别：setosa、versicolor 和 virginica

我们将尝试回答以下问题：

- 不同品种鸢尾花的花瓣长度和宽度是否存在差异?
- 鸢尾花的大小是否与种类有关?

[157]:
```
# 加载 iris 数据集
data(iris)
```

[158]:
```
# 查看前几行数据
head(iris)
```

A data.frame: 6 × 5

	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	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	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

7.1 花瓣长度和宽度是否存在差异？

首先，我们用 gather() 函数将数据从宽格式变为长格式，以便更好地进行可视化。

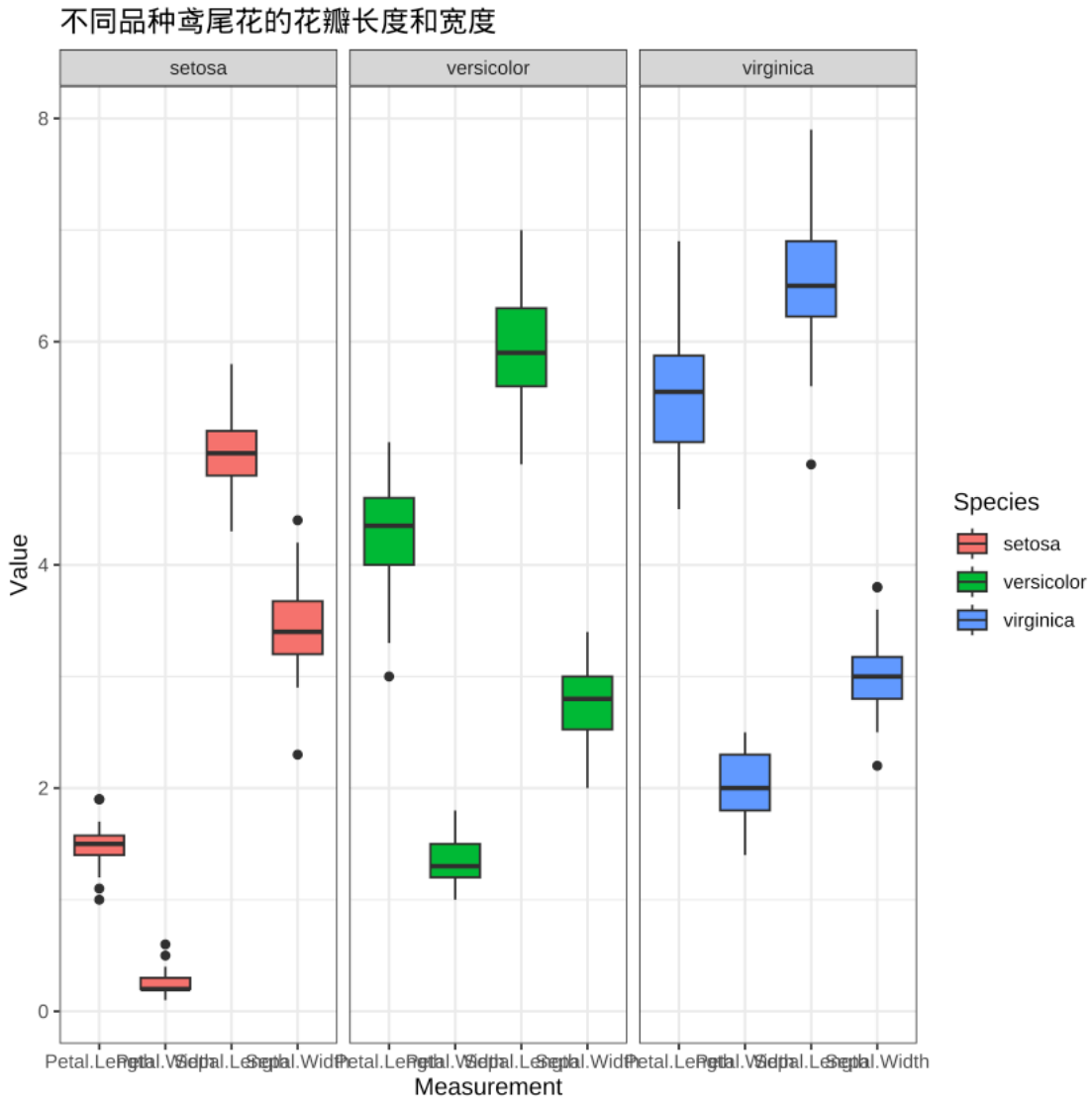
[159]:
```
# 使用 gather 函数将数据从宽格式变为长格式
iris_long <- iris %>%
    gather(key = "measurement", value = "value", Sepal.Length:Petal.Width)
iris_long %>% head(10)
```

A data.frame: 10 × 3

	Species <fct>	measurement <chr>	value <dbl>
1	setosa	Sepal.Length	5.1
2	setosa	Sepal.Length	4.9
3	setosa	Sepal.Length	4.7
4	setosa	Sepal.Length	4.6
5	setosa	Sepal.Length	5.0
6	setosa	Sepal.Length	5.4
7	setosa	Sepal.Length	4.6
8	setosa	Sepal.Length	5.0
9	setosa	Sepal.Length	4.4
10	setosa	Sepal.Length	4.9

然后，我们使用 facet_grid() 函数将不同种类的鸢尾花绘制在不同的子图中，更好地比较不同种类之间的差异。

[160]:
```
# 使用 facet_grid() 函数将不同品种鸢尾花绘制在不同的子图中
options(warn=-1) # 在 ggplot2 函数的开头添加 options(warn=-1) 来关闭所有警告
ggplot(iris_long, aes(x = measurement, y = value, fill = Species)) +␣
 ↪geom_boxplot() +
    facet_grid(. ~ Species) + theme_bw() + labs(title = "不同品种鸢尾花的花瓣长度和宽度") +
    xlab("Measurement") + ylab("Value")
```



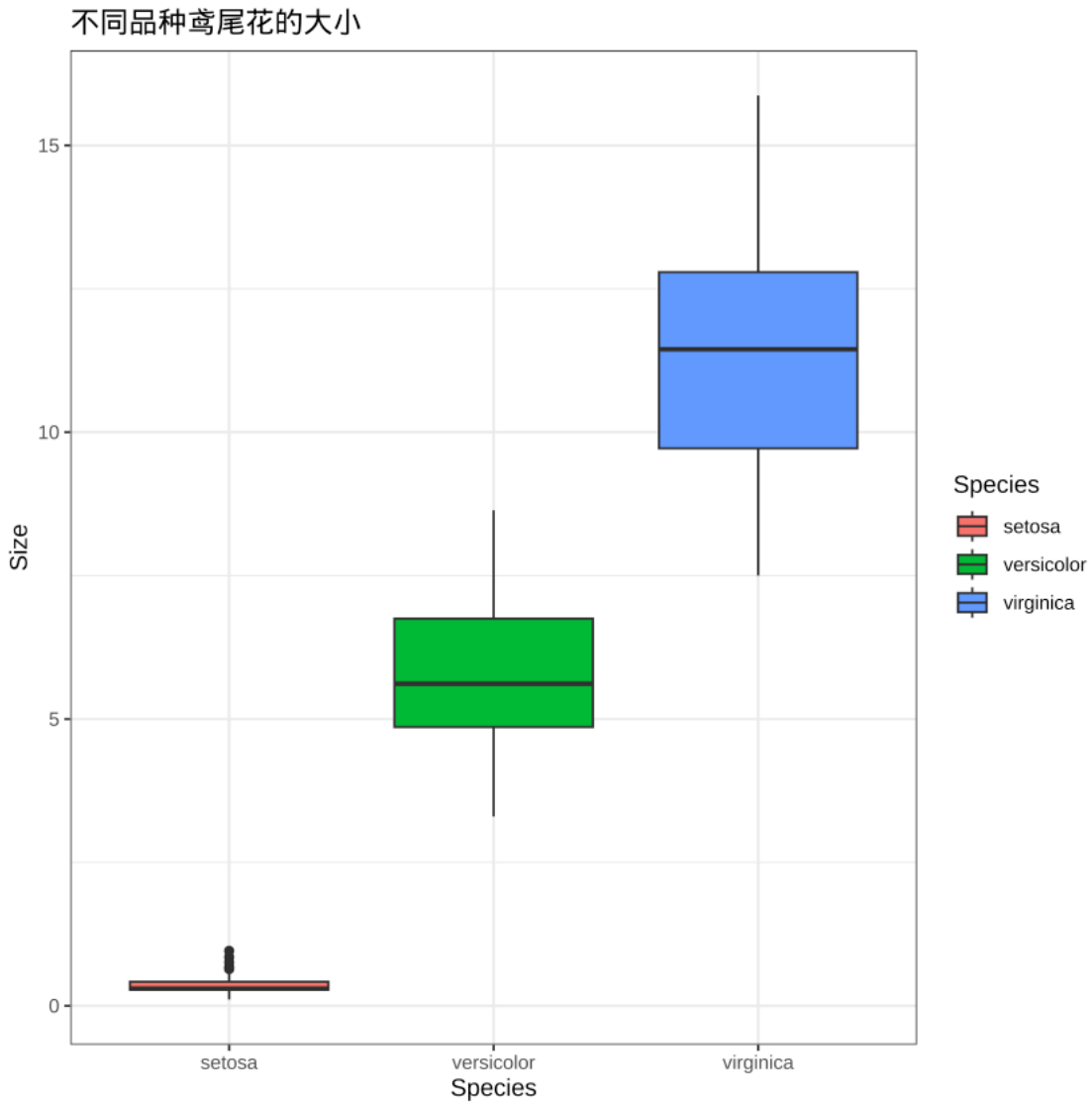
可以看到，不同品种鸢尾花的花瓣长度和宽度确实存在差异。鸢尾花 “setosa” 的花瓣相对较短而宽，而鸢尾花 “versicolor” 和 “virginica” 的花瓣相对较长但宽度较窄。

7.2 花的大小是否与种类有关?

接下来，我们可以使用 dplyr 库计算每朵鸢尾花的大小，并使用 ggplot2 可视化不同品种鸢尾花的大小分布情况。

[161]:
```r
# 计算每朵鸢尾花的大小
iris <- iris %>%
    mutate(size = Petal.Length * Petal.Width)
```

[162]:
```r
# 绘制不同品种鸢尾花的大小分布情况
ggplot(iris, aes(x = Species, y = size, fill = Species)) + geom_boxplot() +␣
↪theme_bw() +
    labs(title = "不同品种鸢尾花的大小") + xlab("Species") + ylab("Size")
```



可以看到，不同品种鸢尾花的大小分布情况也存在差异。鸢尾花 “virginica” 的大小分布最大，而鸢尾花 “setosa” 的大小分布最小。

8 结果导出

[163]:
```
library(broom)
```

[164]:
```
fit <- lm(Sepal.Length~Sepal.Width+
    Petal.Length+Petal.Width,
    data=iris)
summary(fit)
```

```
Call:
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width,
    data = iris)

Residuals:
     Min       1Q   Median       3Q      Max
-0.82816 -0.21989  0.01875  0.19709  0.84570

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.85600    0.25078   7.401 9.85e-12 ***
Sepal.Width   0.65084    0.06665   9.765  < 2e-16 ***
Petal.Length  0.70913    0.05672  12.502  < 2e-16 ***
Petal.Width  -0.55648    0.12755  -4.363 2.41e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3145 on 146 degrees of freedom
Multiple R-squared:  0.8586,        Adjusted R-squared:  0.8557
F-statistic: 295.5 on 3 and 146 DF,  p-value: < 2.2e-16
```

[165]:
```
# broom 对 fit 格式化
glance(fit)
```

A tibble: 1 × 12

r.squared <dbl>	adj.r.squared <dbl>	sigma <dbl>	statistic <dbl>	p.value <dbl>	df <dbl>	logLik <dbl>	AIC <dbl>
0.8586117	0.8557065	0.3145491	295.5391	8.588101e-62	3	-37.32136	84.64272

[166]:
```
# 分组回归
reg <- iris %>%
```

```
    group_by(Species) %>%
    do(fit = lm(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, data =␣
 ↪.))
tidy(fit)
```

A tibble: 4 × 5

term	estimate	std.error	statistic	p.value
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
(Intercept)	1.8559975	0.25077711	7.400984	9.853855e-12
Sepal.Width	0.6508372	0.06664739	9.765380	1.199846e-17
Petal.Length	0.7091320	0.05671929	12.502483	7.656980e-25
Petal.Width	-0.5564827	0.12754795	-4.362929	2.412876e-05

9 R 语言便捷快捷的代码技巧

- 在 R 中**读取和写入数据文件**是数据分析任务中非常常见的操作。可以使用 base R 中的 read.table() 或 write.table() 函数，也可以使用 readr 包中的 read_csv() 或 write_csv() 等函数。
- 为了简化代码和提高效率，可以编写自己的函数来执行这些重复的任务。在 R 中，函数的编写非常简单，可以使用 function() 和 return() 语句创建自己的函数并执行特定操作。

- 使用**管道符（%>%）**：该符号可以大大减少代码长度和提高代码可读性。它允许你将一系列函数链接在一起，并将中间结果传递给下一个函数。
- 使用**匿名函数**：在某些情况下，使用匿名函数可以减少代码量。通过使用 `(function(x) x^2)(5)` 这样的语法，可以直接对 5 进行平方运算，而不必定义一个具名函数。
- 使用**向量化操作**：R 中很多函数都是向量化的，这意味着它们可以同时处理整个向量。在处理数据分析任务中的大数据时，这是非常有用的。

- `options(warn=-1)` 在 ggplot2 函数的开头添加 options(warn=-1) 来关闭所有警告

- 如果你在脚本或函数中使用 ggplot2 进行绘图，必须使用 print() 命令确保图像得到渲染。