導言

資料科學（data science）是一門令人興奮的學科，它能讓你將原始資料轉化為理解、洞察和知識。*R 資料科學*的目標是幫助你學習 R 中最重要的工具，讓你能夠以有效率、可重現的方式進行資料科學研究，並在過程中獲得一些樂趣！讀完本書後，你將掌握各種工具，利用 R 的最佳部分應對各種資料科學挑戰。

第二版序言

歡迎來到《*R 資料科學*》（*R for Data Science*，*R4DS*）的第二版！這是對第一版的重大更新，刪除了我們認為不再有用的素材，添加了我們希望當初有包含在第一版中的內容，並對文字和程式碼進行了全面更新，以反映最佳實務做法的變化。我們也非常高興歡迎一位新的合著者：Mine Çetinkaya-Rundel，她是著名的資料科學教育家，也是我們在 Posit（前身為 RStudio 的公司）的同事之一。

下面簡要介紹一下最大的變化：

- 本書的第一篇更名為「遊戲全貌（Whole Game）」。這篇的目的是在我們深入探討資料科學的細節之前，向你簡介資料科學這「整個遊戲」的粗略樣貌。
- 本書的第二篇是「視覺化（Visualize）」。與第一版相比，本篇更全面地介紹資料視覺化工具和最佳實務做法。獲取所有詳細資訊的最佳途徑仍然是 ggplot2（*https://oreil.ly/HNIie*）一書，但現在 R4DS 涵蓋了更多最重要的技巧。
- 本書的第三篇現在稱為「變換（Transform）」，新增了關於數字、邏輯向量（logical vectors）和缺失值（missing values）的章節。這些內容之前是資料變換章節的一部分，但需要更多的篇幅來涵蓋所有細節。

- 本書的第四篇名為「匯入（Import）」。這是一組新的章節，除了讀取平面文字檔案外，還包括處理試算表（spreadsheets）、從資料庫（databases）獲取資料、運用大資料（big data）、矩形化階層式資料（rectangling hierarchical data）以及從網站上搜刮資料（scraping data）。
- 「程式（Program）」這篇仍然保留，但已從上到下重新編寫過，把焦點放在函式編寫和迭代最重要的部分。函式編寫現在涵蓋如何包裹 tidyverse 的函式（以因應 tidy evaluation 之挑戰）的細節，因為這在過去幾年中變得更加容易且重要。我們新增了一章，介紹你可能會在「野生」R 程式碼中看到的重要基礎 R 函式。
- 「建模（Modeling）」篇已被刪除。我們從來都沒有足夠的空間來充分介紹建模，而且現在已有更好的資源可用。我們通常會推薦使用 tidymodels 套件（*https://oreil.ly/0giAa*）並閱讀 Max Kuhn 和 Julia Silge 合著的《*Tidy Modeling with R*》（*https://oreil.ly/9Op9s*，O'Reilly 出版）。
- 「溝通（Communicate）」篇依然保留，但已全面更新，以 Quarto（*https://oreil.ly/_6LNH*）取代 R Markdown。本書的這一版是用 Quarto 編寫的，它顯然是未來的工具。

你會學到什麼

資料科學是一個廣闊的領域，你不可能只閱讀一本書就掌握所有知識。本書旨在為你打下堅實的基礎，讓你知曉最重要的工具並擁有足夠的知識，以便在必要時能找到讓你進一步學習的資源。我們典型的資料科學專案步驟之模型如圖 I-1 所示。

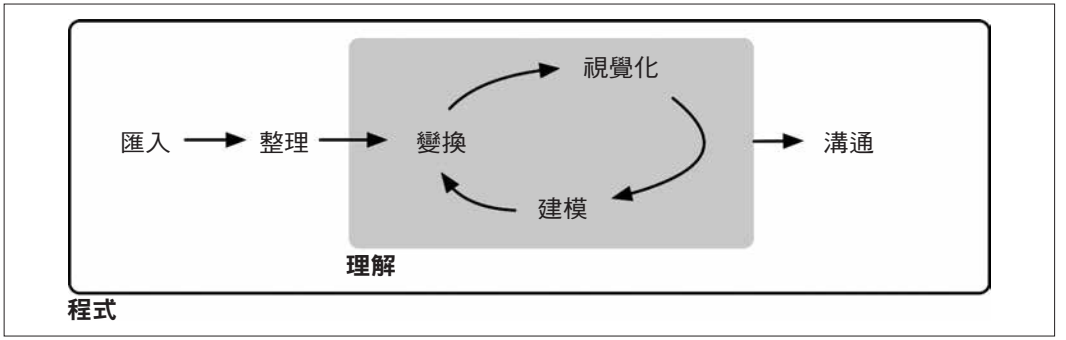


圖 I-1　在我們的資料科學流程模型中，首先是資料匯入和整理。然後，透過變換、視覺化和建模的迭代循環來理解你的資料。最後，將結果傳達給其他人來完成整個流程

首先，你必須將資料**匯入**（*import*）到 R 之中。這通常意味著，將儲存在檔案、資料庫或 Web API（application programming interface）中的資料載入（load）到 R 的某個資料框（data frame）中。如果你沒辦法將資料匯入 R，就不能進行資料科學研究！

匯入資料後，最好對其進行**整理**（*tidy*）。整理資料意味著以一致的形式儲存資料，使資料集（dataset）的語意（semantics）與儲存方式相匹配。簡而言之，你的資料經過整理後，每一欄（column）都會是一個變數（variable），每一列（row）則都是一個觀測值（observation）。資料的整理非常重要，因為一致的結構可以讓你集中精力回答資料相關的問題，而非為了不同的函式將資料變換成正確的形式而奮鬥。

有了整齊的資料後，下一步通常是對資料進行**變換**（*transform*）。變換包括縮小感興趣的觀測值之範圍（如一個城市的所有人口或去年的所有資料），建立作為現有變數之函數的新變數（如根據距離和時間計算出速度），以及計算一組彙總統計量（如計數值或平均值）。整理和變換合在一起被稱為「**整頓**（*wrangling*）」，因為將資料轉變為一種方便處理的自然形式，往往讓人感覺像在戰鬥（*wrangling* 原本有「吵架、爭論」的意思）！

一旦獲得了包含所需變數的整齊資料，知識的產生就會有兩個主要的引擎：視覺化（visualization）和建模（modeling）。它們的優缺點互補，因此任何真正的資料分析工作都會在它們之間反覆迭代多次。

視覺化（*visualization*）本質上是一種人類活動。好的視覺化能向你展示你意想不到的東西，或提出有關資料的新問題。良好的視覺化還可能暗示你問錯了問題，或者你需要蒐集不同的資料。視覺化可以為你帶來驚喜，但它們的規模擴充性並不是特別好，因為需要人類來解讀。

模型（*models*）是視覺化的補充工具。一旦你讓問題變得足夠精確，就可以使用模型來回答。從根本上說，模型是數學或計算工具，因此它們通常具有良好的規模擴充性。即使碰到並非如此的情況，購買更多的電腦通常也比購買更多的大腦便宜！但是，每個模型都有假設（assumptions），而就其本質而言，模型無法質疑自己的假設。也就是說，模型基本上不可能讓你感到驚訝。

資料科學的最後一步是**溝通**（*communication*），這是任何資料分析專案絕對關鍵的部分。如果你無法將結果傳達給他人，那麼你的模型和視覺化有多能幫助你理解資料，就一點也不重要了。

圍繞在所有這些工具之外的，是**程式設計**（*programming*）。程式設計是一種貫穿各領域的工具，幾乎在資料科學專案的每個部分都會用到。要想成為一名成功的資料科學家

（data scientist），你並不需要成為一名專業的程式設計師，但學習更多的程式設計知識是會有回報的，因為成為一名更好的程式設計師可以讓你更輕鬆地自動化常見任務並解決新問題。

在每個資料科學專案中，你都會用到這些工具，但對於大多數專案來說，這些工具並不足夠。這裡有一個粗略的 80/20 規則：使用本書所學的工具，你可以解決每個專案中大約 80% 的問題，但你還需要其他工具來解決剩下的 20%。在本書中，我們將為你提供資源，讓你知道哪裡可以學到更多。

本書的組織方式

前面對資料科學工具的描述，大致是按照你在分析中使用這些工具的順序來編排的（當然，你會多次重複迭代使用這些工具）。不過，根據我們的經驗，先學習資料匯入和整理不是最理想的，因為在 80% 的時間裡，這都會是例行公事且枯燥乏味，而在另外 20% 的時間裡，這會是古怪且令人沮喪的事情。那不是學習新主題的良好起點！取而代之，我們將從已經匯入和整理過的資料之視覺化和變換開始。如此一來，攝入並整理自己的資料時，你還是能夠維持高動機，因為你知道那些痛苦是值得忍受的。

在每一章中，我們都會盡量遵循一種統一的模式：從一些激勵性的例子開始，讓你看到全局，然後深入細節。本書的每個章節都配有習題，幫助你實際演練所學到的知識。雖然跳過習題很有誘惑力，但沒有比在實際問題中練習運用知識更好的學習方法了。

你不會學到什麼

本書並未涉及幾個重要的主題。我們認為，重要的是要堅持不懈地專注於基本內容，這樣才能儘快上手並開始實作。這意味著本書不可能涵蓋到所有重要的主題。

建模

建模（modeling）對於資料科學來說超級重要，但這是一個很大的主題，遺憾的是，我們沒有足夠的空間在此對其進行完整的介紹。要瞭解建模的更多資訊，強烈推薦我們的同事 Max Kuhn 和 Julia Silge 合著的《*Tidy Modeling with R*》（*https://oreil.ly/9Op9s*，O'Reilly 出版）。這本書將向你介紹 tidymodels 系列套件，正如你可能從名稱中猜到的，它們與我們在本書中使用的 tidyverse 套件共享許多慣例。

大資料（Big Data）

我們自負地將本書的重點放在記憶體內的小型資料集。這是本書的正確起點，因為如果沒有處理小資料的經驗，就無法處理大資料。你在本書大部分內容中學到的工具可以輕鬆處理數百 MB 的資料，只要稍加注意，你通常可以使用它們來處理幾 GB 的資料。我們還將向你展示如何從資料庫和 parquet 檔案中獲取資料，這兩者經常被用來儲存大資料。你不一定能處理整個資料集，但那不是問題，因為你只需要一個子集（subset）或子樣本（subsample）來回答你感興趣的問題。

如果你經常需要處理較大型的資料（比如 10 ～ 100 GB），我們建議你多學習 data.table（*https://oreil.ly/GG4Et*）。在此我們不會教授它，因為它使用的介面和 tidyverse 不一樣，需要你學習一些不同的慣例。不過，它的速度快得令人難以置信，如果你正在處理大型資料，它的效能回報值得你投入一些時間去學習。

Python、Julia 和其他朋友

在本書中，你不會學到關於 Python、Julia 或其他對資料科學有用的程式語言的任何知識。這並不是因為我們認為這些工具不好，它們並不差！在實務上，大多數資料科學團隊會混合使用多種語言，通常至少會使用 R 和 Python。但我們堅信，一次最好只試圖精通一種工具，而 R 就是一個很好的開始。

預備知識

為了讓你從本書中獲得最大的收穫，我們對你已經掌握的知識做了一些假設。一般來說，你應該具備一定的數字識讀能力（numerically literate），如果你已經有了一些基本的程式設計經驗，會對你有所幫助。如果你以前從未學過程式設計，Garrett Grolemund 所著的《*Hands-On Programming with R*》（*https://oreil.ly/8uiH5*，O'Reilly 出版）可以是本書的重要輔助讀物。

要執行本書中的程式碼，你需要四樣東西：R、RStudio、被稱為 *tidyverse* 的 R 套件集合，以及其他一些套件。套件（packages）是可重現（reproducible）的 R 程式碼的基本單元。它們包括可重複使用的函式、描述如何使用它們的說明文件以及範例資料。

R

要下載 R，請前往 CRAN（*https://oreil.ly/p3_RG*），即 *c*omprehensive *R* *a*rchive *n*etwork。R 的主要版本（major version）每年釋出一次，次要版本（minor releases）每年釋出二

到三次。定期更新是個好主意。升級可能有點麻煩，尤其是需要重新安裝你所有套件的主要版本，但拖延只會讓情況更糟。本書推薦使用 R 4.2.0 或更高版本。

RStudio

RStudio 是用於 R 程式設計的 IDE（integrated development environment），你可以從 RStudio 下載頁面（*https://oreil.ly/pxF-k*）取得。RStudio 每年會更新幾次，有新版本釋出時，它會自動通知你，因此無須回頭自行檢查。定期升級是個好主意，這樣才能運用最新、最強大的功能。就本書而言，請確保你至少有 RStudio 2022.02.0。

啟動 RStudio 時，如圖 I-2 所示，你會看到介面中有兩個關鍵區域：主控台窗格（console pane）和輸出窗格（output pane）。現在，你只需在主控台窗格中鍵入 R 程式碼，然後按 Enter 執行即可。隨著我們往前進，你會學習到更多[1]！

Tidyverse

你還需要安裝一些 R 套件。R 套件是函式、資料和說明文件的集合，可擴充基礎 R 的能力。使用套件是成功運用 R 的關鍵。你將在本書中學到的大多數套件都屬於所謂的 tidyverse。tidyverse 中的所有套件在資料和 R 程式設計方面都有著共同的理念，並被設計為可以一起作業。

只需一行程式碼，你就可以安裝完整的 tidyverse：

```
install.packages("tidyverse")
```

在電腦上，於主控台打入這行程式碼，然後按 Enter 執行。R 將從 CRAN 下載套件並安裝到你的電腦上。

在載入套件之前，你都無法使用套件中的函式、物件或說明檔案。安裝套件後，可以使用 `library()` 函式載入它：

```
library(tidyverse)
#> ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
#> ✔ dplyr     1.1.0.9000     ✔ readr     2.1.4
#> ✔ forcats   1.0.0          ✔ stringr   1.5.0
#> ✔ ggplot2   3.4.1          ✔ tibble    3.1.8
#> ✔ lubridate 1.9.2          ✔ tidyr     1.3.0
#> ✔ purrr     1.0.1
#> ── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
#> ✖ dplyr::filter() masks stats::filter()
```

1 如果你想對 RStudio 的所有功能有個全面的總覽，請參閱 RStudio User Guide（*https://oreil.ly/pRhEK*）。

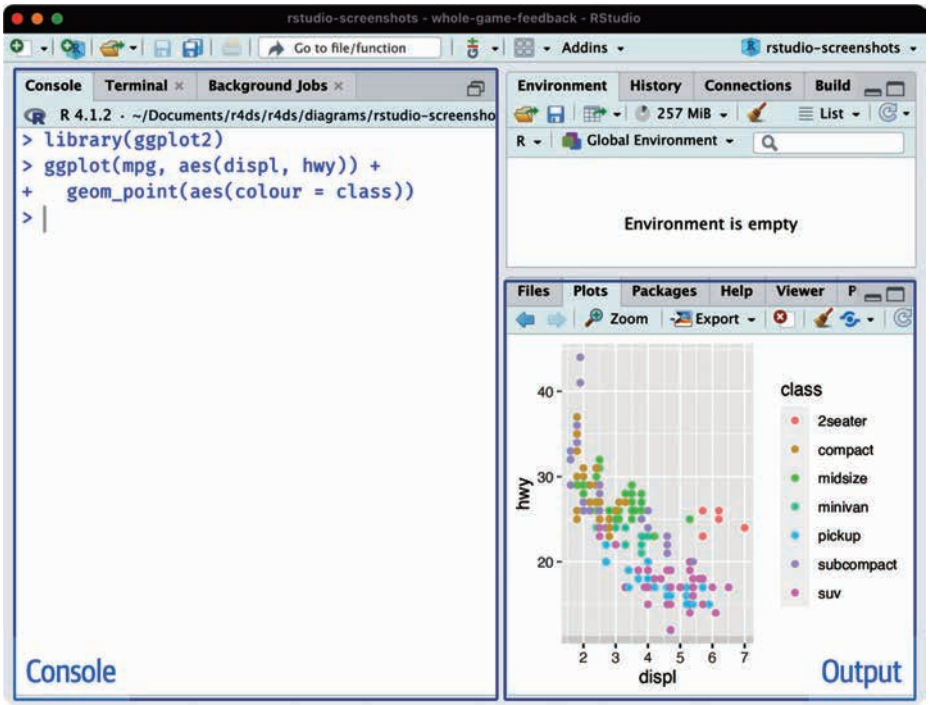


圖 I-2　RStudio IDE 有兩個關鍵區域：在左邊的主控台窗格中輸入 R 程式碼，並在右邊的輸出窗格中找尋繪製好的圖表

```
#> ✖ dplyr::lag()    masks stats::lag()
#> ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
#>   conflicts to become errors
```

這告訴你 tidyverse 載入了九個套件：dplyr、forcats、ggplot2、lubridate、purrr、readr、stringr、tibble 和 tidyr。這些套件被視為 tidyverse 的**核心**，因為幾乎所有分析都會用到它們。

tidyverse 中的套件變化相當頻繁。你可以透過執行 `tidyverse_update()` 檢查是否有更新。

其他套件

還有許多其他優秀的套件不屬於 tidyverse，因為它們解決的是不同領域的問題，或者是根據不同的基本原則而設計的。這並不意味著它們更好或更差，只是使它們不同而已。換句話說，與「tidyverse（整齊宇宙）」互補的，並非「messyverse（混亂宇宙）」，而是相互關聯的套件所組成的許多其他宇宙。使用 R 處理更多資料科學專案時，你就會學到新的套件和思考資料的新方式。

在本書中，我們將使用來自 tidyverse 之外的許多套件。舉例來說，我們將使用以下套件，因為它們提供有趣的資料集，供我們在學習 R 的過程中使用：

```
install.packages(c("arrow", "babynames", "curl", "duckdb", "gapminder", "ggrepel",
"ggridges", "ggthemes", "hexbin", "janitor", "Lahman", "leaflet", "maps",
"nycflights13", "openxlsx", "palmerpenguins", "repurrrsive", "tidymodels", "writexl"))
```

我們還將在一次性範例中使用一些其他套件。你現在不需要安裝它們，只要記住每當你看到類似這樣的錯誤時：

```
library(ggrepel)
#> Error in library(ggrepel) : there is no package called 'ggrepel'
```

就意味著你需要執行 `install.packages("ggrepel")` 來安裝該套件。

執行 R 程式碼

上一節向你展示了執行 R 程式碼的幾個範例。書中的程式碼看起來會像這樣：

```
1 + 2
#> [1] 3
```

若在本地端主控台執行相同的程式碼，結果看起來會像這樣：

```
> 1 + 2
[1] 3
```

主要有兩個差異。在主控台中，你鍵入的內容位於 `>` 之後，稱為**提示符號**（*prompt*）；而我們在本書中不顯示提示符號，輸出會用 `#>` 註解；而在主控台中，那會直接出現在程式碼之後。這兩點不同意味著，如果你正在使用本書的電子版，你可以很輕易地從書中複製程式碼並貼到主控台中。

在整本書中，我們使用一套前後一致的慣例來指涉程式碼：

- 函式以程式碼字型（code font）顯示，後面接著括弧（parentheses），如 `sum()` 或 `mean()`。
- 其他的 R 物件（如資料或函式引數）使用程式碼字型，但不帶括弧，如 `flights` 或 `x`。
- 有時，為了清楚地說明某個物件來自哪裡，我們會在套件名稱後使用兩個冒號（colons），如 `dplyr::mutate()` 或 `nycflights13::flights`。這也是有效的 R 程式碼。

第一篇

遊戲全貌

這篇的目標是要讓你快速瞭解資料科學的主要工具：匯入、整理、變換和視覺化資料，如圖 I-1 所示。我們希望向你展示資料科學的「遊戲全貌」，讓你對所有主要部分有足夠的瞭解，這樣你就能處理真實的簡單資料集。本書後續的部分將更深入地討論每個主題，從而增加你可以應對的資料科學挑戰之範圍。

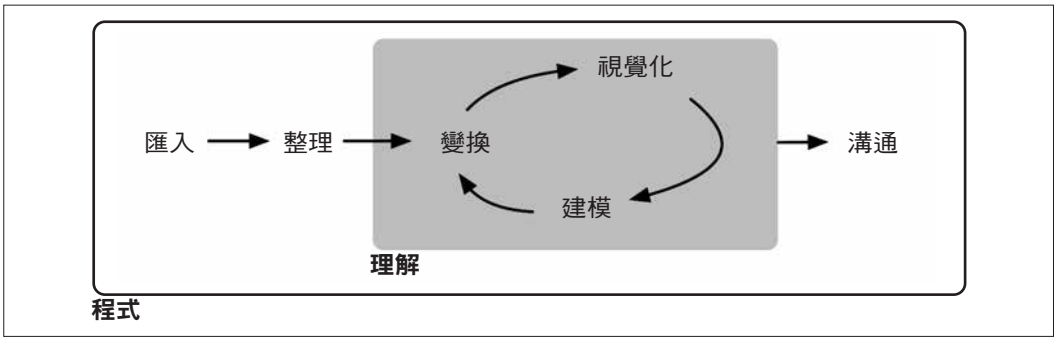


圖 I-1　你將在此篇學習如何匯入、整理、變換和視覺化資料

其中四章重點介紹資料科學的工具：

- 視覺化是開始學習 R 程式設計的絕佳起點，因為它的回報是顯而易見的：你可以繪製優雅而詳實的圖表，幫助你理解資料。在第 1 章中，你將深入學習視覺化，瞭解 ggplot2 圖表的基本結構，以及將資料轉化為圖表的強大技巧。
- 僅有視覺化通常是不夠的，因此在第 3 章中，你將學習一些關鍵動詞，讓你能夠選擇重要變數、篩選出關鍵觀測值、創建新變數並計算摘要。

- 在第 5 章中，你將學到整齊的資料（tidy data），這是一種前後一致的資料儲存方式，能讓變換、視覺化和建模變得更容易。你將瞭解其基本原理以及如何將資料轉化為整齊的形式。
- 在變換和視覺化資料之前，你需要先把資料弄到 R 裡面。在第 7 章中，你將學習將 `.csv` 檔案匯入 R 的基礎知識。

在這些章節中，還有其他四章關於 R 的工作流程（R workflow）。在第 2 章、第 4 章和第 6 章中，你將學習到編寫和組織 R 程式碼的良好工作流程實務做法。長遠來看，這些將為你的成功奠定基礎，因為它們會為你提供在處理實際專案時保持條理清晰的工具。最後，第 8 章將教你如何獲得幫助並持續學習。

第一章

資料視覺化

簡介

「簡單的圖表為資料分析師帶來的資訊，比任何其他工具都還要多。」

—John Tukey

R 有數個製圖系統，而 ggplot2 是其中最優雅、功能最多樣的一個。ggplot2 實作了圖形文法（*grammar of graphics*），這是一個用於描述和建置圖形的連貫系統。透過 ggplot2，你可以只學習一個系統並將其應用於多個地方，藉此更快地完成更多工作。

本章將教你如何使用 ggplot2 將資料視覺化。我們首先建立一個簡單的散佈圖（scatterplot），並用它來介紹美學映射（aesthetic mappings）和幾何物件（geometric objects），也就是 ggplot2 的基本構件。然後，我們將引導你對單一變數的分佈（distributions）進行視覺化，以及視覺化兩個或更多個變數之間的關係（relationships）。最後，我們將介紹如何儲存圖表和疑難排解的訣竅。

先決條件

本章的焦點放在 tidyverse 的核心套件之一 ggplot2。要存取本章使用的資料集、說明頁面和函式，請執行以下命令載入 tidyverse：

```
library(tidyverse)
#> ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
#> ✔ dplyr     1.1.0.9000     ✔ readr     2.1.4
#> ✔ forcats   1.0.0          ✔ stringr   1.5.0
#> ✔ ggplot2   3.4.1          ✔ tibble    3.1.8
#> ✔ lubridate 1.9.2          ✔ tidyr     1.3.0
```

```
#> ✔ purrr     1.0.1
#> ── Conflicts ──────────────────────────────────────── tidyverse_conflicts() ──
#> ✖ dplyr::filter() masks stats::filter()
#> ✖ dplyr::lag()    masks stats::lag()
#> ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
#>   conflicts to become errors
```

那一行程式碼會載入核心 tidyverse，也就是你在幾乎所有資料分析中都會用到的套件。它還會告訴你，tidyverse 中的哪些函式與基礎 R 中的函式（或你可能載入的其他套件中的函式）相衝突[1]。

如果你執行這段程式碼，得到的錯誤訊息是 `there is no package called 'tidyverse'`（沒有名為「tidyverse」的套件），你就得先安裝它，然後再次執行 `library()`：

```
install.packages("tidyverse")
library(tidyverse)
```

你只需安裝一次套件，但每次啟動新的工作階段（session）時都需要載入它。

除 tidyverse 外，我們還將使用 palmerpenguins 套件，其中包括 `penguins` 資料集（包含 Palmer Archipelago 三個島嶼上企鵝的身體測量資料）和 ggthemes 套件（提供對色盲友善的調色盤）。

```
library(palmerpenguins)
library(ggthemes)
```

最初的步驟

鰭肢（flippers）長的企鵝比鰭肢短的企鵝重還是輕呢？你可能已經有了答案，但請試著讓你的答案更加精確。鰭肢長和體重之間的關係是怎樣的？是正相關？負相關？線性關係？非線性？企鵝的種類不同，兩者之間的關係也會不同嗎？企鵝生活的島嶼也會有影響嗎？讓我們建立視覺化來回答這些問題。

penguins 資料框

你可以使用 palmerpenguins（即 `palmerpenguins::penguins`）中的 `penguins` 資料框（data frame）查驗這些問題的答案。資料框是變數（欄中的）和觀測值（列中的）的矩形集合。

1 使用 conflicted 套件可以消除該訊息，並在需要時強制進行衝突解析，隨著你載入更多的套件，這會變得越來越重要。有關 conflicted 的更多資訊，請前往該套件的網站（*https://oreil.ly/01bKz*）。

`penguins` 包含 Kristen Gorman 博士和 Antarctica LTER 的 Palmer Station 蒐集到並提供出來的 344 個觀測值[2]。

為了讓討論更容易進行，我們來定義一些術語：

變數（*Variable*）

可以測量（measure）的數量（quantity）、特質（quality）或特性（property）。

值（*Value*）

測量一個變數時，它的狀態（state）。不同次的測量，變數的值可能會發生變化。

觀測值（*Observation*）

在相似條件下進行的一組測量（你通常會在同一時間對同一物體做出一次觀測中的所有測量動作）。一個觀測值將包含多個值，每個值與一個不同的變數相關聯。我們有時會把一個觀測值稱為一個資料點（*data point*）。

表格式資料（*Tabular data*）

一組值，每個值與一個變數和一個觀測值相關聯。如果每個值都放在自己的「單元格（cell）」中，每個變數都放在自己的欄（column）中，而且每個觀測值都放在自己的列（row）中，那麼表格式資料就是整齊（*tidy*）的。

在這種情境下，一個變數指涉的是所有企鵝的某個屬性（attribute），而觀測值指的是單隻企鵝的所有屬性。

在主控台中鍵入資料框的名稱，R 將印出其內容的預覽。注意到，在這個預覽的頂端寫著 `tibble`。在 tidyverse 中，我們使用名為 *tibbles* 的特殊資料框，你很快就會學到。

```
penguins
#> # A tibble: 344 × 8
#>   species island    bill_length_mm bill_depth_mm flipper_length_mm
#>   <fct>   <fct>              <dbl>         <dbl>             <int>
#> 1 Adelie  Torgersen           39.1          18.7               181
#> 2 Adelie  Torgersen           39.5          17.4               186
#> 3 Adelie  Torgersen           40.3          18                 195
#> 4 Adelie  Torgersen           NA            NA                  NA
#> 5 Adelie  Torgersen           36.7          19.3               193
#> 6 Adelie  Torgersen           39.3          20.6               190
#> # … with 338 more rows, and 3 more variables: body_mass_g <int>, sex <fct>,
#> #   year <int>
```

2 Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. R package version 0.1.0. *https://oreil.ly/ncwc5*. doi: 10.5281/zenodo.3960218.

這個資料框包含八欄。要檢視所有變數和每個變數的前幾個觀測值，請使用 `glimpse()`。或者，若是在 RStudio 中，請執行 `View(penguins)` 來開啟互動式資料檢視器（data viewer）。

```
glimpse(penguins)
#> Rows: 344
#> Columns: 8
#> $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, A…
#> $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torge…
#> $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.…
#> $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.…
#> $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, …
#> $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347…
#> $ sex               <fct> male, female, female, NA, female, male, female, m…
#> $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2…
```

`penguins` 的變數包括：

`species`

企鵝的種類（Adelie、Chinstrap 或 Gentoo）

`flipper_length_mm`

企鵝鰭肢的長度，以公釐（millimeters）為單位

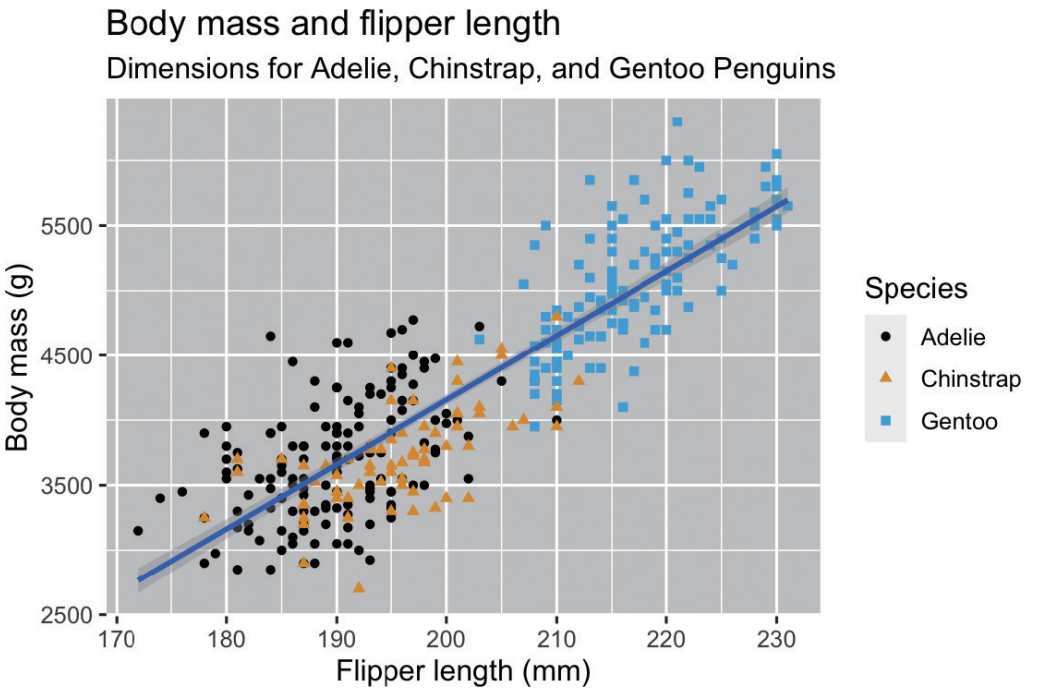
`body_mass_g`

企鵝的體重，以公克（grams）為單位。

要瞭解有關 `penguins` 的更多資訊，請執行 `?penguins` 開啟說明頁面。

最終目標

本章的最終目標是在考慮到企鵝種類（species）的情況下，重建以下視覺化圖表，顯示這些企鵝的鰭長（flipper lengths）和體重（body masses）之間的關係。



創建一個 ggplot

讓我們逐步重建這個圖表。

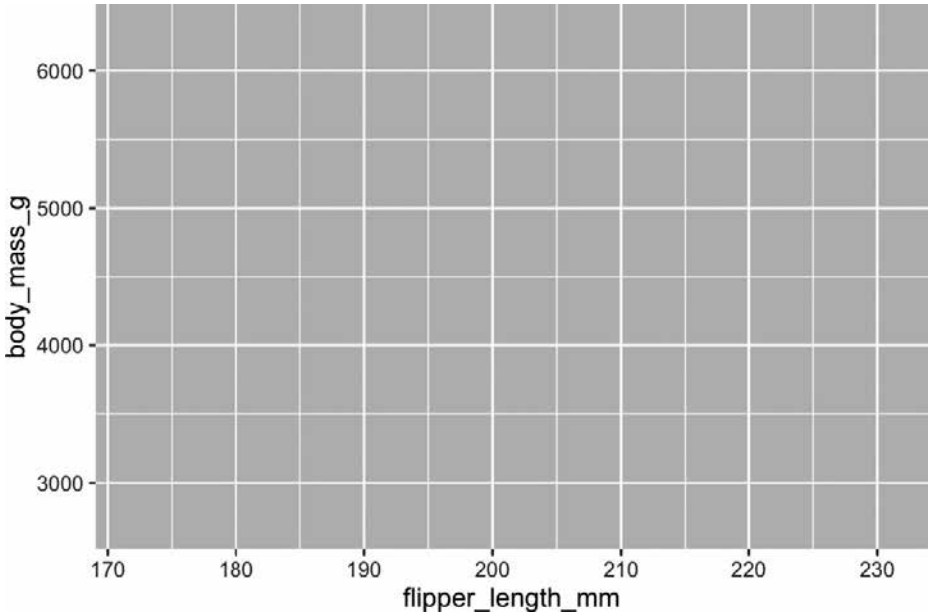
使用 ggplot2 時，首先要用函式 `ggplot()` 定義一個圖表物件（plot object），然後為它新增圖層（*layers*）。`ggplot()` 的第一個引數是要在圖中使用的資料集，因此 `ggplot(data = penguins)` 會建立一個經過預填的空圖，可用來顯示 `penguins` 的資料，但由於我們還沒有告訴它如何將資料視覺化，所以現在它還是空的。這並不是一個非常令人興奮的圖表，但你可以把它想像成一塊空白畫布，其中你可以繪製圖表的其餘圖層。

```
ggplot(data = penguins)
```

接下來，我們需要告訴 `ggplot()` 如何以視覺化的方式呈現我們資料中的資訊。`ggplot()` 函式的 `mapping` 引數定義如何將資料集中的變數映射（mapped to）到圖表的視覺特性（*aesthetics*，美學元素）上。`mapping` 引數在 `aes()` 函式中總是有定義的，而 `aes()` 的 `x` 和 `y` 引數指定了要映射到 x 軸和 y 軸的變數。ggplot2 會在 `data` 引數（本例中為 `penguins`）中尋找所映射的變數。

下列圖表顯示了新增這些映射後的結果。

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
)
```



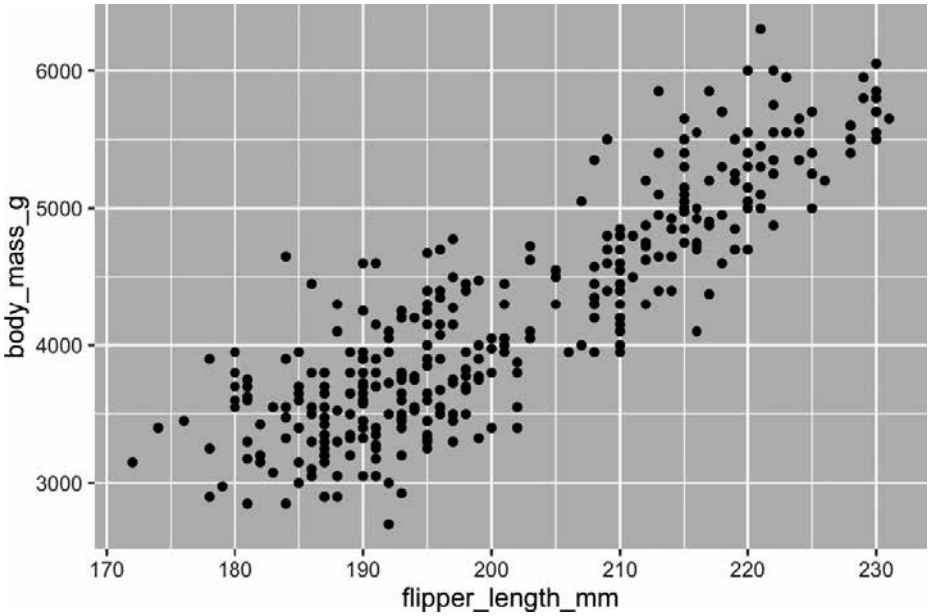
我們的空畫布現在有了更多的結構：很明顯地，鰭肢的長度將會顯示（在 x 軸上），而身體質量也將顯示（在 y 軸上）。但是那些企鵝本身還沒有出現在圖上。這是因為我們尚未在程式碼中闡明如何在圖表裡呈現資料框中的觀測值。

為此，我們需要定義一個 *geom*：圖表用來表示資料的幾何物件（geometrical object）。在 ggplot2 中，以 `geom_` 開頭的函式供應那些幾何物件。人們經常透過圖表所用的 geom 類型來描述圖表。舉例來說，長條圖（bar charts）使用 bar geoms（`geom_bar()`）；折線圖（line charts）使用 line geoms（`geom_line()`）；盒狀圖（boxplots）使用 boxplot

geoms（`geom_boxplot()`）；散佈圖（scatterplots）使用 point geoms（`geom_point()`），諸如此類。

函式 `geom_point()` 會在圖表上新增一層圖點，從而建立散佈圖。ggplot2 附有許多 geom 函式，每個函式都會在圖表中加上不同類型的圖層。在本書中，特別是第 9 章，你將學習到大量的 geom 函式。

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point()
#> Warning: Removed 2 rows containing missing values (`geom_point()`).
```



現在，我們有了一個看起來像「散佈圖（scatterplot）」的東西。它還不是符合我們「最終目標」的圖表，但利用這個圖就可以開始回答促使我們進行探索的問題了：「鰭長和體重之間的關係是怎樣的？」。這種關係似乎是正相關的（鰭肢的長度增加，體重也增加）、相當線性（點集中在一條直線而非一條曲線周圍）和中等強度（在這樣一條線周圍沒有太多分散的點）。鰭長的企鵝通常身體質量較高。

在為這個圖表新增更多圖層之前，我們先暫停一下，回顧我們得到的警告訊息：

> Removed 2 rows containing missing values (`geom_point()`).

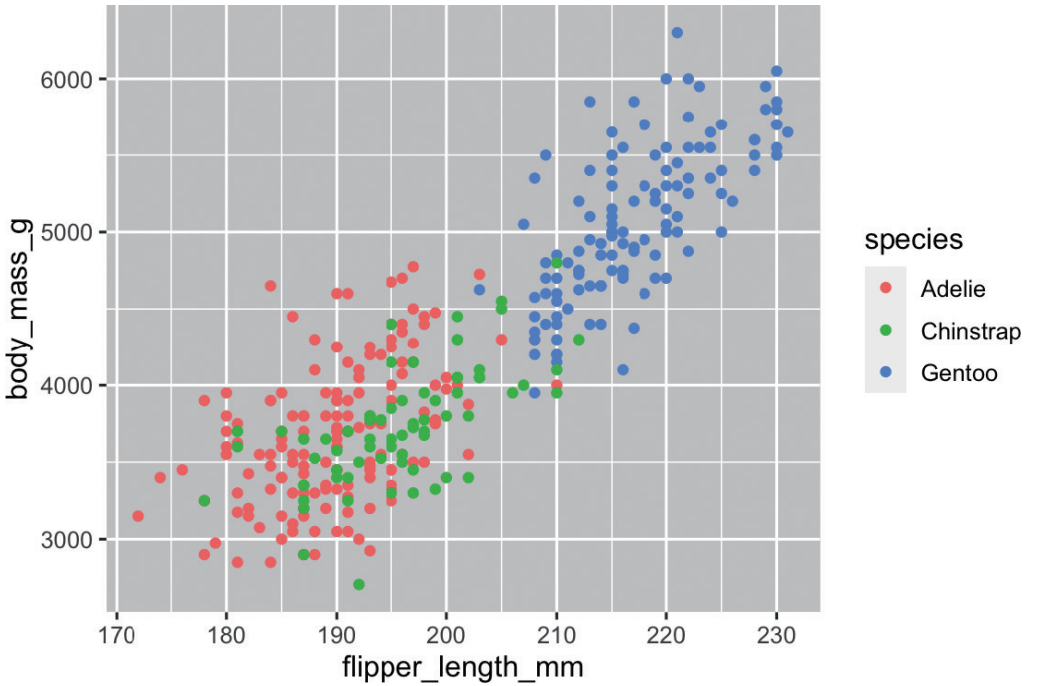
我們之所以會看到這條訊息，是因為我們的資料集中有兩隻企鵝缺少體重或鰭長值，而 ggplot2 無法在沒有這兩個值的情況下於圖上表示它們。和 R 一樣，ggplot2 也認為缺失值不應該無聲無息地被忽略。這種類型的警告可能是你在處理真實資料時，最常見的警告類型之一：缺少某些值是一種常見問題，你將在本書中瞭解到更多有關它們的處理方式，尤其是在第 18 章中。在本章的其他圖表中，我們將抑制這種警告，這樣就不會在繪製每幅圖表時都印出這種警告。

新增美學元素和圖層

散佈圖對於顯示兩個數值變數之間的關係非常有用，但對於兩個變數之間的任何明顯關係，我們都應該抱持懷疑態度，並詢問是否有其他變數可以解釋或改變這種明顯關係的本質。舉例來說，鰭長和體重之間的關係是否因種類（species）而異？讓我們在圖表中加入種類，看看這是否能揭示這些變數之間明顯關係的任何額外洞見。我們將用不同顏色的點來表示種類。

為了達成這一目標，我們需要修改 aesthetic（美學元素）還是 geom（幾何物件）呢？如果你猜的是「在 `aes()` 內的美學映射（aesthetic mapping）中」，那麼你就已經掌握了使用 ggplot2 建立資料視覺化的訣竅！若非如此，也不用擔心。在本書中，你將製作更多的 ggplots，並有更多機會在製作過程中檢驗自己的直覺。

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point()
```

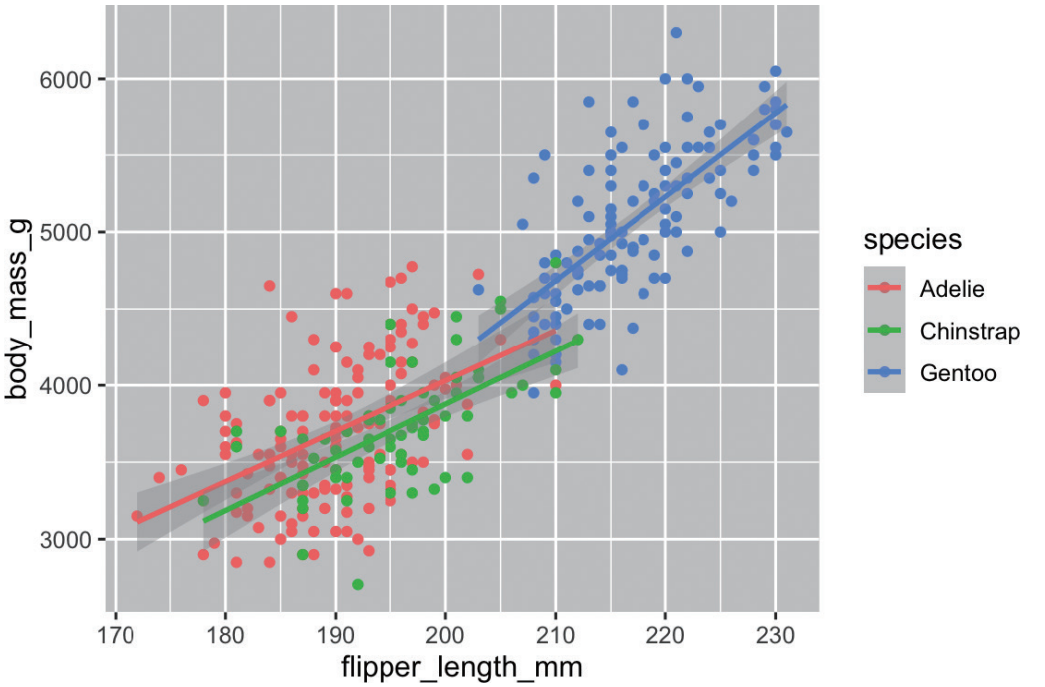


當一個類別變數（categorical variable）映射到一個 aesthetic 時，ggplot2 會自動為變數的每一個級別（level，三個種類中的每一個）配置一個唯一的美學元素值（aesthetic，在此為一種唯一的顏色），這個過程被稱為 *scaling*（標定）。ggplot2 還會新增一個圖例（legend），解釋哪個值對應哪個級別。

現在讓我們再新增一層：一條顯示體重與鰭長之間關係的平滑曲線（smooth curve）。繼續之前，請參考前面的程式碼，並思考如何將其添加到我們現有的圖表中。

由於這是一個代表我們資料的新幾何物件，我們將新增一個新的 geom 作為 point geom 上的一層：`geom_smooth()`。然後使用 `method = "lm"` 指出我們想要根據線性模型（linear model）繪製最佳擬合線。

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point() +
  geom_smooth(method = "lm")
```

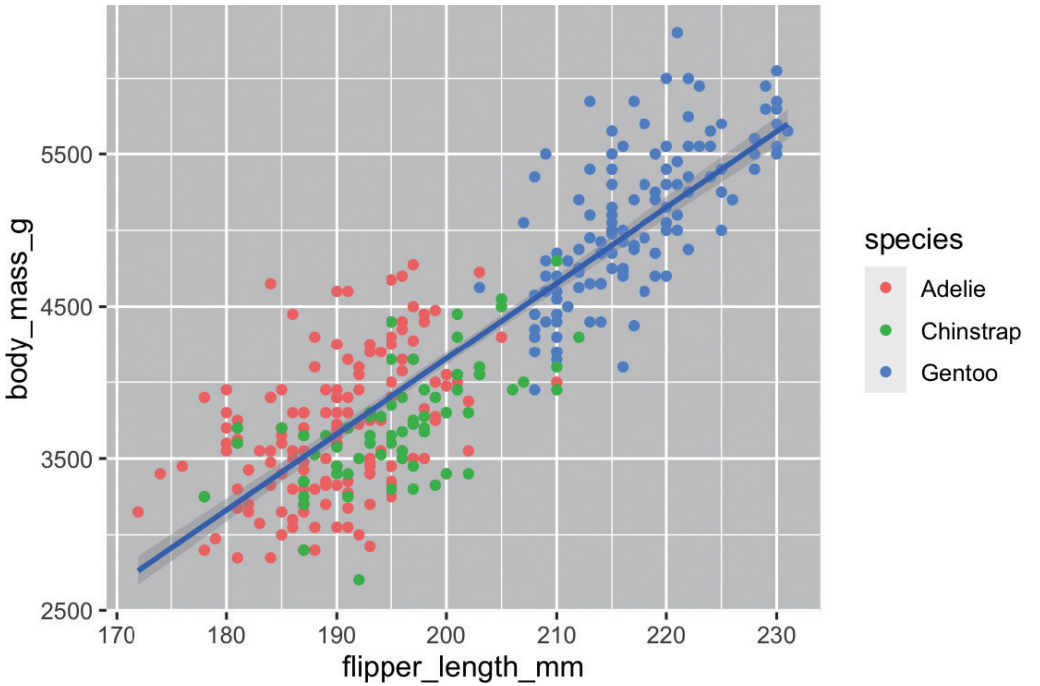


我們已經成功添加了線條，但這幅圖看起來並不像第 6 頁「最終目標」中的那幅圖，因為在那幅圖中，整個資料集只有一條線，而不是每個企鵝種類都有單獨的線條。

在 `ggplot()` 中定義了**全域**層級（*global* level）的美學映射（aesthetic mappings）後，這些映射會向下傳到圖表的每個後續的 geom 層。不過，ggplot2 中的每個 geom 函式也可以接受一個 `mapping` 引數，這樣就允許**區域**層級（*local* level）的美學映射被加到從全域層級繼承而來的那些美學映射之上。

由於我們希望根據種類（species）對點進行著色，但又不希望為它們分別繪製線條，因此我們應該只為 `geom_point()` 指定 `color = species`。

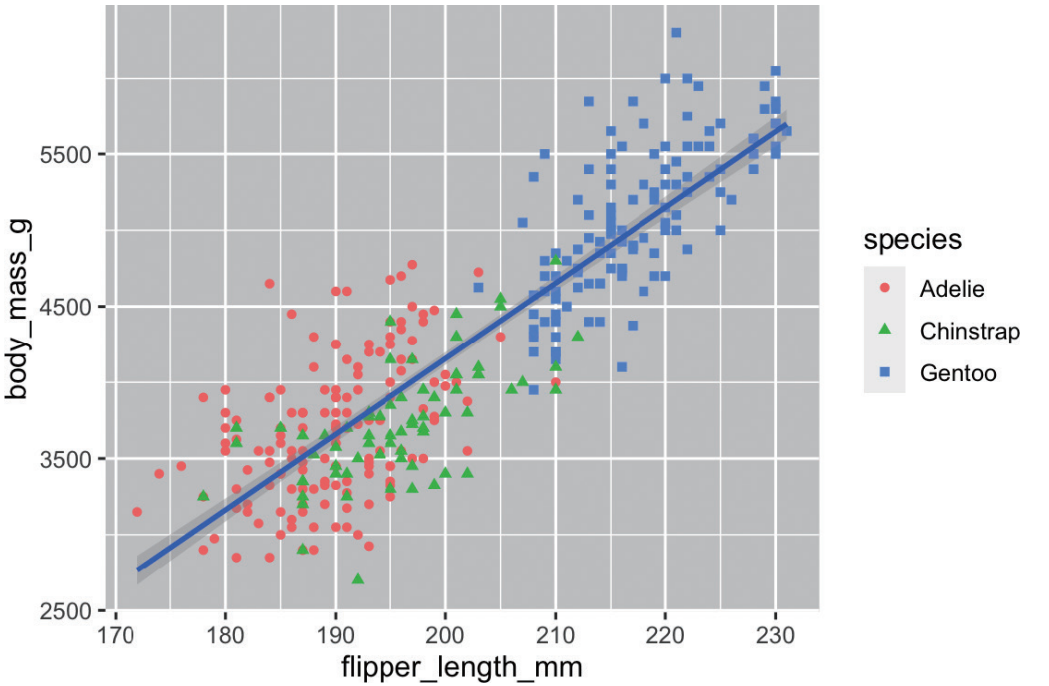
```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(mapping = aes(color = species)) +
  geom_smooth(method = "lm")
```



大功告成！雖然還不夠完美，但我們已經有了一個看起來非常像我們最終目標的東西。我們還需要為每種企鵝使用不同的形狀，並改善標籤（labels）。

一般來說，在圖表中只用顏色來表示資訊並不是一個好主意，因為人們會因為色盲或其他色覺差異而對顏色產生不同的感知。因此，除了色彩，我們還可以將 `species` 映射到 `shape` 美學元素。

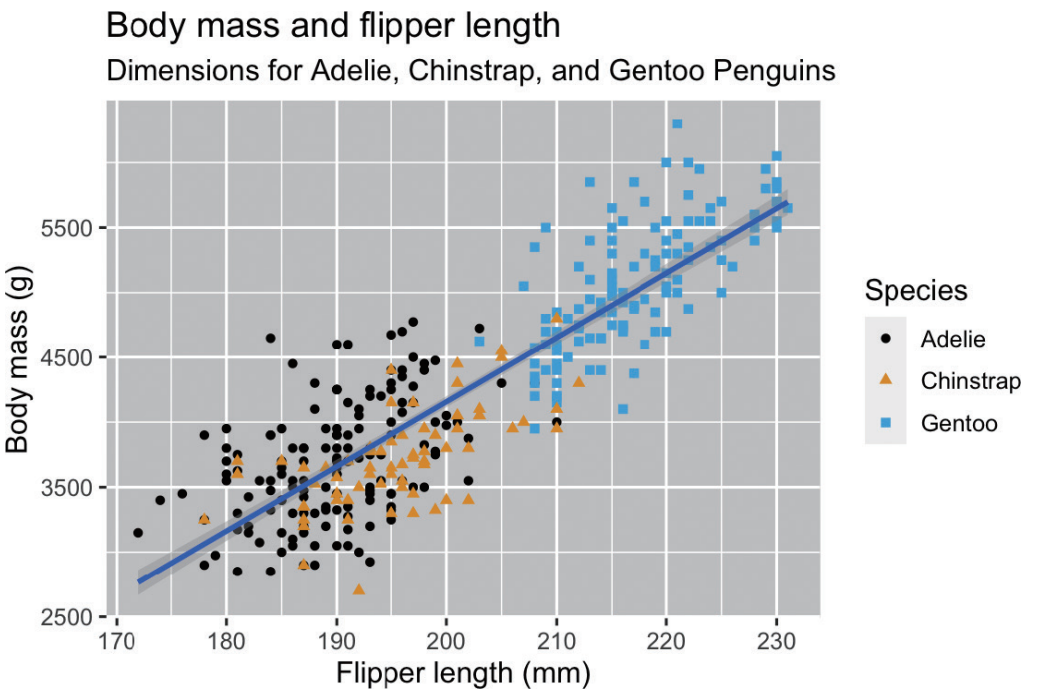
```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = "lm")
```



請注意，圖例也會自動更新，以反映點的不同形狀。

最後，我們可以在新圖層中使用 `labs()` 函式來改良我們圖表的標籤。`labs()` 的一些引數可能不言自明：`title` 新增一個標題（title），而 `subtitle` 新增一個副標題（subtitle）。其他引數則與美學映射相匹配：`x` 是 x 軸標籤，`y` 是 y 軸標籤，而 `color` 和 `shape` 則定義了圖例的標籤。此外，我們還可以使用 ggthemes 套件中的 `scale_color_colorblind()` 函式來改善調色盤，使其具有色盲友善性（color-blind safe）。

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(aes(color = species, shape = species)) +
  geom_smooth(method = "lm") +
  labs(
    title = "Body mass and flipper length",
    subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
    x = "Flipper length (mm)", y = "Body mass (g)",
    color = "Species", shape = "Species"
  ) +
  scale_color_colorblind()
```



我們終於有了一個完全符合我們「最終目標」的圖表！

習題

1. `penguins` 有多少列（rows）？有多少欄（columns）？
2. `penguins` 資料框中的 `bill_depth_mm` 變數描述什麼？請閱讀 `?penguins` 的說明以瞭解詳情。
3. 繪製 `bill_depth_mm` vs. `bill_length_mm` 的散佈圖（scatterplot）。也就是說，請製作一個散佈圖，其中 `bill_depth_mm` 在 y 軸，而 `bill_length_mm` 在 x 軸。描述這兩個變數之間的關係。
4. 若是繪製出 `species` vs. `bill_depth_mm` 的散佈圖，會發生什麼事？什麼會是更好的 geom 選擇？
5. 為什麼下列程式碼會出現錯誤，你要如何解決呢？

```
ggplot(data = penguins) +
  geom_point()
```

第三篇

變換

本書的第二篇深入探討了資料視覺化。在本書的這部分中，你將瞭解到在資料框（data frame）中會遇到的最重要的變數型別，並學習搭配這些變數使用的工具。

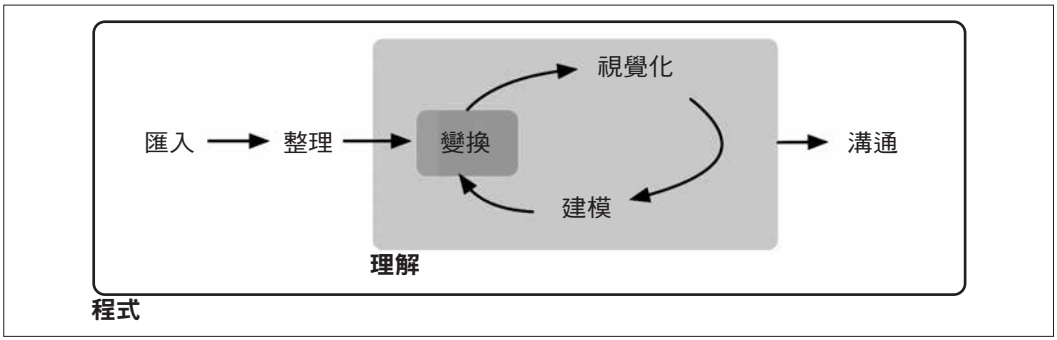


圖 III-1　資料變換的選擇在很大程度上取決於所涉及的資料型別，這也是本書此部分的主題

你可以根據自己的需要閱讀這些章節；它們在設計上大多是獨立的，因此可以不按順序閱讀。

- 第 12 章將介紹邏輯向量。邏輯向量是最簡單的向量型別，但功能非常強大。你將學習如何透過數值比較建立邏輯向量、如何以 Boolean 代數結合邏輯向量、如何在摘要中使用邏輯它們，以及如何將邏輯向量用於條件式變換。
- 第 13 章深入探討資料科學的核心，也就是數字向量（vectors of numbers）的工具。你將學到更多關於計數（counting）的知識，以及一系列重要的變換和摘要（summary）函式。

- 第 14 章為你提供處理字串（strings）的工具：將字串切片、切成小方塊，然後再將它們黏合在一起。本章主要關注 stringr 套件，但你也會學到一些專門用來從字串中提取資料的 tidyr 函式。
- 第 15 章將向你介紹正規表達式（regular expressions），這是一種強大的字串操作工具。本章將帶你從疑惑是否有隻貓從你的鍵盤上走過，到能夠讀寫複雜的字串模式。
- 第 16 章介紹因子（factors）：R 用來儲存類別資料（categorical data）的資料型別。當一個變數有一組固定的可能值，或者想要對字串進行非字母順序的排列時，就需要使用因子。
- 第 17 章為你提供處理日期和日期時間（date-times）的關鍵工具。遺憾的是，你對日期時間瞭解得越多，它們似乎就越複雜，但在 lubridate 套件的幫助下，你將學會如何克服最常見的挑戰。
- 第 18 章深入討論缺失值（missing values）。我們已經單獨提及過幾次缺失值，現在是全面討論缺失值的時候了，幫助你掌握隱含缺失值和明確缺失值之間的區別，以及如何和為什麼要在它們之間進行變換。
- 第 19 章將為你提供將兩個（或多個）資料框連接（join）在一起的工具，從而為本篇做結。學習連接會讓你不得不考慮索引鍵（keys）的概念，並思考如何識別資料集中的每一列（row）。

第十二章

邏輯向量

簡介

在本章中，你將學習處理邏輯向量（logical vectors）的工具。邏輯向量是最簡單的向量型別，因為每個元素只能是三種可能值之一：`TRUE`、`FALSE` 和 `NA`。在原始資料中發現邏輯向量的情況相對罕見，但在幾乎所有分析過程中都會建立和處理邏輯向量。

首先，我們將討論建立邏輯向量的最常見方法：數值比較（numeric comparisons）。然後，你將瞭解如何使用 Boolean 代數（或稱「布林代數」）來組合不同的邏輯向量，以及一些有用的摘要運算。最後，我們將學習 `if_else()` 和 `case_when()`，這兩個實用的函式可以利用邏輯向量進行條件式變化。

先決條件

本章將學習的大部分函式都由基礎 R 提供，因此我們不需要 tidyverse，但我們仍將載入它，以便使用 `mutate()`、`filter()` 和相關功能來處理資料框。我們還將繼續從 `nycflights13::flights` 資料集中擷取範例。

```
library(tidyverse)
library(nycflights13)
```

不過，隨著我們開始使用更多的工具，並非總能找到完美的真實範例。因此，我們將開始用 `c()` 製作一些虛擬資料：

```
x <- c(1, 2, 3, 5, 7, 11, 13)
x * 2
#> [1]  2  4  6 10 14 22 26
```

這樣就更容易解釋個別函式，但代價是更難看出那些函式如何套用於你的資料問題。請記住，我們對單獨存在的向量所做的任何操作，都可以透過 `mutate()` 和相關功能對資料框內的變數進行。

```
df <- tibble(x)
df |>
  mutate(y = x *  2)
#> # A tibble: 7 × 2
#>       x     y
#>   <dbl> <dbl>
#> 1     1     2
#> 2     2     4
#> 3     3     6
#> 4     5    10
#> 5     7    14
#> 6    11    22
#> # … with 1 more row
```

比較

建立邏輯向量的常見方法是使用 `<`、`<=`、`>`、`>=`、`!=` 和 `==` 進行數值比較。到目前為止，我們主要是在 `filter()` 中臨時建立邏輯變數：它們參與計算、被使用，然後被丟棄。舉例來說，下面的過濾器（filter）可以找到所有大致準時抵達的白天出發航班：

```
flights |>
  filter(dep_time > 600 & dep_time < 2000 & abs(arr_delay) < 20)
#> # A tibble: 172,286 × 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#> 1  2013     1     1      601            600         1      844            850
#> 2  2013     1     1      602            610        -8      812            820
#> 3  2013     1     1      602            605        -3      821            805
#> 4  2013     1     1      606            610        -4      858            910
#> 5  2013     1     1      606            610        -4      837            845
#> 6  2013     1     1      607            607         0      858            915
#> # … with 172,280 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>, …
```

這只是一種快捷方式，你可以使用 `mutate()` 明確建立底層的邏輯變數：

```
flights |>
  mutate(
    daytime = dep_time > 600 & dep_time < 2000,
    approx_ontime = abs(arr_delay) < 20,
    .keep = "used"
```

```
  )
#> # A tibble: 336,776 × 4
#>   dep_time arr_delay daytime approx_ontime
#>      <int>     <dbl> <lgl>   <lgl>
#> 1      517        11 FALSE   TRUE
#> 2      533        20 FALSE   FALSE
#> 3      542        33 FALSE   FALSE
#> 4      544       -18 FALSE   TRUE
#> 5      554       -25 FALSE   FALSE
#> 6      554        12 FALSE   TRUE
#> # … with 336,770 more rows
```

這對更複雜的邏輯尤其有用，因為命名中間步驟既便於程式碼的閱讀，也便於檢查每個步驟的計算是否正確。

總而言之，最初的過濾器與下列程式碼等效：

```
flights |>
  mutate(
    daytime = dep_time > 600 & dep_time < 2000,
    approx_ontime = abs(arr_delay) < 20,
  ) |>
  filter(daytime & approx_ontime)
```

浮點比較（Floating-Point Comparison）

要小心使用與數字搭配的 `==`。舉例來說，看起來這個向量包含數字 1 和 2：

```
x <- c(1 / 49 * 49, sqrt(2) ^ 2)
x
#> [1] 1 2
```

但如果測試它們是否相等，結果卻會是 `FALSE`：

```
x == c(1, 2)
#> [1] FALSE FALSE
```

這是怎麼回事？電腦儲存的數字有固定的小數位數（fixed number of decimal places），因此無法精確表示 1/49 或 `sqrt(2)`，後續的計算結果會略有偏差。我們可以透過呼叫帶有 `digits`[1] 引數的 `print()` 來檢視精確值：

```
print(x, digits = 16)
#> [1] 0.9999999999999999 2.0000000000000004
```

1 R 通常會為你呼叫 `print`（也就是說，`x` 是 `print(x)` 的一種捷徑），但如果你想提供其他引數，明確的呼叫 `print` 會很有用。

你可以理解為什麼 R 預設會將這些數字捨入（rounding）；它們確實非常接近你的預期。

既然你已經知道了 == 失敗的原因，那麼你能做些什麼呢？有種方法是使用 dplyr::near()，它會忽略微小的差異：

```
near(x, c(1, 2))
#> [1] TRUE TRUE
```

缺失值

缺失值代表未知數（unknown），因此具有「傳染性（contagious）」：幾乎所有涉及未知數的運算都會產生未知數：

```
NA > 5
#> [1] NA
10 == NA
#> [1] NA
```

最令人感到困惑的結果是這一個：

```
NA == NA
#> [1] NA
```

如果我們人為地提供更多的背景資訊，就更容易理解為什麼會這樣：

```
# 我們不知道 Mary 的年紀
age_mary <- NA

# 我們不知道 John 的年紀
age_john <- NA

# Mary 和 John 同年齡嗎？
age_mary == age_john
#> [1] NA
# 我們並不知道！
```

因此，如果要找出所有缺少 dep_time 的航班，下面的程式碼是行不通的，因為 dep_time == NA 會使每一列都產生 NA，而 filter() 會自動捨棄缺少的值：

```
flights |>
  filter(dep_time == NA)
#> # A tibble: 0 × 19
#> # … with 19 variables: year <int>, month <int>, day <int>, dep_time <int>,
#> #   sched_dep_time <int>, dep_delay <dbl>, arr_time <int>, …
```