




Tidy Intro

Рідлист

- [R Cookbook, 2e \(James \(JD\) Long and Paul Teetor\)](#)  — коли немає часу розбиратися з документацією, окрім прикладів базового R інкорпорує у себе також приклади роботи з **tidyverse**
- [R for Data Science, 2e \(Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund\)](#)  — всі основи (та більше) маніпуляції / візуалізації даних з **tidyverse**

Також можете спробувати [Statistical Inference via Data Science: A ModernDive into R and the Tidyverse! \(Chester Ismay and Albert Y. Kim\)](#) , що окрім декількох коротких розділів про **tidyverse** пропонує дуже м'яке та візуально підкріплене ознайомлення з низкою статистичних концептів

Tidyverse as is

Екосистема [tidyverse](#) представляє собою набір бібліотек, що покликані уніфікувати процес роботи з даними за допомогою R, усі пакети **tidyverse** мають спільну філософію дизайну та граматику. Остання, на момент створення цієї презентації, версія **tidyverse 2.0.0** включає у себе 31 пакет. Ядро **tidyverse** для щоденного використання включає у себе 8 пакетів, які одночасно можливо завантажити викликом `library(tidyverse)`:

- **tibble** — інтродукує новий клас формату даних аналогічний до `data.frame` — `tbl_df`
- **ggplot2** — декларативна система створення графіки, основана на філософії представлень у The Grammar of Graphics (Leland Wilkinson, 2005)
- **dplyr** — універсифікована та послідовна граMATика маніпуляції з даними
- **tidyr** — набір функцій для “очищення” даних та приведення їх до загальної форми
- **readr** — покращений імпорт даних прямокутного формату
- **stringr** — пакет для маніпуляції з текстовими даними
- **forcats** — пакет для маніпуляції з факторами
- **purrr** — тулкіт для функціонального програмування з R

Також існує безліч бібліотек, які хоча і не є офіційно частиною **tidyverse**, слідуєть тим же принципам дизайну і є сумісними з ними. Фактично близько 40% бібліотек [дані звідси](#), що наразі хостить CRAN, мають у своїх залежностях одну та більше бібліотек **tidyverse**. Окрім лінку наданого вище, CheatSheets до пакетів **tidyverse** та суміжних бібліотек можуть бути знайдені [тут](#) та [тут](#)

Читання та запис даних

Пакет `readr` надає аналоги базових команд для завантаження текстових даних прямокутного формату. Команди для читання даних з диску виглядають як `read_*()`, зворотній варіант для запису даних на диск виглядає як `write_*()`

```
1 my_data <- read_csv("path/to/file.csv")
2 write_csv(my_data, "path/to/my_data.csv")
```



Окрім `read_csv("file.csv")` наявні:

- `read_csv2("file.csv")` — дані розділені через `;`
- `read_tsv("file.tsv")` — дані розділені через `Tab`
- `read_fwf("file.tsv", fwf_widths())` — дані фіксованої ширини
- `read_delim("file.txt", delim = " ")` — узагальнений варіант формату з розділенням
- `read_lines("file.txt")` — текстові дані, по рядках
- `read_log("file.log")` — файли формату `.log`

Файли з розширенням `.gz`, `.bz2`, `.xz` або `.zip` будуть автоматично розпаковані, файли, що починаються з `http://`, `https://`, `ftp://` або `ftps://`, будуть автоматично завантажені

Читання та запис даних

Подавши вектор з текстових значень у якості першого аргументу, можливо прочитати декілька файлів у один спільний кадр

```
1 my_data <- read_csv(c("data1.csv", "data2.csv", "data3.csv"), id = "orig_file")
```



Тут аргумент `id` створить додаткову колонку з відповідною назвою (у випадку вище з назвою `orig_file`), що буде містити шлях до одного з відповідних файлів, з яких були прочитані дані

Низка інших опціональних аргументів функції сімейства `read_*`() включає у себе:

- `col_names` — якщо логічне значення то вказує чи має перший рядок розглядатися як імена колонок, якщо вектор текстових значень то використовується як user-supplied імена колонок (перший рядок тоді считується як частина кадру даних)
- `col_types` — тип даних у колонках, може бути наданий у вигляді листа значень через `list()` або `cols()`, де кожній колонці відповідає певна специфікація, або у вигляді компактної буквеної репрезентації, де кожна літера відповідає типу колонки. Якщо значення `NULL`, функція буде намагатися визначити тип даних у колонках шляхом оцінки перших 1000 рядків
- `na` — текстовий вектор, що позначає які значення мають бути інтерпретовані як `NA`
- `skip` — вказує кількість рядків, які будуть пропущені перед тим як дані будуть прочитані
- `comment` — текстовий знак, що має ідентифікувати коментарі у даних (текст, що ідентифіковано як коментар буде проігноровано при читанні даних)

Читання та запис даних

Пакет `readxl` (що поставляється як частина `tidyverse`, але не входить до ядра) дозволяє зчитувати дані у форматі `.xls` та `.xlsx`

```
1 data_xlsx <- readxl::read_xlsx("file.xlsx")
2 data_xls  <- readxl::read_xls("file.xls")
3 data_exel <- readxl::read_exel("file.*") # автоматично вибирає між .xls та .xlsx форматом
```

Функції для читання файлів формату `.xls/xlsx` окрім іншого мають опціональні аргументи `sheet` для вказання з якого конкретно листа електронної таблиці імпортувати дані та `range` для вказання з якої конкретно області листа (комірки від і до, наприклад "A1:D25") імпортувати дані

Бібліотеки для імпорту/експорту інших форматів:

- `googlesheets4` [↗](#) — для гугл-таблиць (поставляється у комплекті `tidyverse`)
- `haven` [↗](#) — для імпорту даних з SAS, SPSS та Stata (поставляється у комплекті `tidyverse`)
- `DBI` [↗](#) — для конекту R до Систем Управління Базами Даних
- `jsonlite` [↗](#) — для формату `json`
- `xml2` [↗](#) — для `XML`
- `httr2` [↗](#) — для роботи з веб-APIs
- `rvest` [↗](#) — для `HTML` та веб-скрапінгу (поставляється у комплекті `tidyverse`)

Tibble

Як `data.frame`, як написано у документації — “Tibbles are data.frames that are lazy and surly: they do less and complain more”. На відміну від стандартних кадрів даних tibbles:

- При створенні не перетворюють текстовий вектор на фактор та не змінюють не-синтактичні імена колонок
- Не схвалюють присутність назв рядків
- Створення є послідовним, по колонкам, тому є можливість звертатися до колонок безпосередньо при створенні об'єкту
- При заповненні колонки ресайклінгу підлягають лише вектори довжиною **1**
- Сабсетинг через `[` по замовчуванню повертає об'єкт типу `tbl_df`
- Сабсетинг через `$` не дозволяє часткового метчингу імені

Tibble не є заміною класу кадру даних, вони є *підкласом* даного класу. У цьому можна впевнитися перевіривши атрибути `tbl_df` об'єкту

```
1 library(palmerpenguins)
2 class(penguins)
3 #> [1] "tbl_df"      "tbl"        "data.frame"
```



Tibble

Створюються так само як звичайні кадри даних

```
1 tibt <- tibble(  
2   a = runif(3),  
3   b = rnorm(3),  
4   prod = a * b,      # референсинг до попередніх колонок у процесі створення  
5   let = letters[1:3])  
6 tibt  
7 #> # A tibble: 3 × 4  
8 #>       a       b     prod let  
9 #>   <dbl> <dbl>   <dbl> <chr>  
10 #> 1  0.307 -0.242 -0.0743 a  
11 #> 2  0.556  0.522  0.290  b  
12 #> 3  0.762  0.219  0.167  c
```

Також можливим є варіант створення “по рядках”, що інколи є зручним для маленьких наборів даних

```
1 tribb <- tribble(  
2   ~spec, ~val, ~sex,  
3   "a",    0.12,  "M",  
4   "a",    14,    "F",  
5   "b",    0.123, "I")  
6 tribb  
7 #> # A tibble: 3 × 3  
8 #>   spec      val sex  
9 #>   <chr>   <dbl> <chr>  
10 #> 1 a      0.12  M  
11 #> 2 a      14    F  
12 #> 3 b      0.123 I
```


Tidy data

Філософія екосистеми tidyverse передбачає роботу з даними у “чистому” або “довгому” форматі. Дані у довгому форматі на противагу широкому формату передбачають, що:

- кожна колонка відповідає одній змінній
- кожен рядок відповідає одному спостереженню
- кожна комірка відповідає одному значенню

Для конверсії одного формату до іншого існують функції `pivot_wider()` та `pivot_longer()` відповідно. Широкий формат виглядає так:

```
1 wide_example
2 #> # A tibble: 3 × 5
3 #>       ID Name   Drug_A Drug_B Placebo
4 #>   <dbl> <chr>   <dbl>  <dbl>   <dbl>
5 #> 1     1 1 Subj A     85     78     95
6 #> 2     2 2 Subj B     72     80     88
7 #> 3     3 3 Subj X     90     88     84
```

Варіант коли треба перевести з широкого у довгий зустрічається частіше

```
1 long_example <- wide_example |>
2   pivot_longer(
3     cols = 3:5,
4     names_to = "Treatment",
5     values_to = "Score"
6   )
```

Tidy data

Довгий формат виглядає так:

```
1 long_example
2 #> # A tibble: 9 × 4
3 #>       ID Name   Treatment Score
4 #>   <dbl> <chr>   <chr>     <dbl>
5 #> 1     1 Subj A Drug_A      85
6 #> 2     1 Subj A Drug_B      78
7 #> 3     1 Subj A Placebo     95
8 #> 4     2 Subj B Drug_A      72
9 #> 5     2 Subj B Drug_B      80
10 #> 6     2 Subj B Placebo     88
11 #> 7     3 Subj X Drug_A      90
12 #> 8     3 Subj X Drug_B      88
13 #> 9     3 Subj X Placebo     84
```

Зворотно у широкий формат

```
1 wide_example <- long_example |>
2   pivot_wider(
3     names_from = Treatment,
4     values_from = Score
5   )
```

Функціональні еквіваленти

Пакет dplyr містить набір функцій низка з яких є аналогічними або ідентичними базовим функціям R, але на відміну від них часто мають (на мою думку) більш зрозумілий та послідовний синтаксис, більш передбачувану поведінку, з самого початку розроблялися для комбінування з ріре-оператором та групованими даними. Більш повна таблиця порівняння з прикладами [тут](#)

dplyr verb	base R verb
<code>filter(df, x)</code>	<code>subset()</code> or <code>df[which(x), , drop = F]</code>
<code>arrange(df, x)</code>	<code>order()</code> , <code>df[!duplicated(x), , drop = F]</code>
<code>distinct(df, x)</code>	<code>unique()</code>
<code>rename(df, y = x), rename_with()</code>	<code>names()</code> or <code>stats::setNames()</code>
<code>mutate(x = y + z)</code>	<code>df\$x <- df\$y + df\$z, transform()</code>
<code>select(df, x, y)</code>	<code>subset()</code> or <code>df[c("x", "y")]</code>
<code>summarise(df, fun(x))</code>	<code>tapply()</code> , <code>aggregate()</code> , <code>by()</code>
<code>pull(df, 1), pull(df, x)</code>	<code>df[[1]], df\$x</code>
<code>slice(df, c(1, 2, 5))</code>	<code>df[c(1, 2, 5), , drop = FALSE]</code> , also partially <code>sample()</code>
<code>*_join()</code>	<code>merge()</code>

Окрім цього `purrr::map()` та `purrr::map2()`, а також у певному сенсі `dplyr::across()`, є функціональними аналогами `lapply()`

Функції для рядків

Функція `filter()`, що повертає рядки кадру даних, що задовольняють певну логічну умову

```
1 iris |> filter(Sepal.Length > 6.5 & Petal.Length < 4.5)
2 #>   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
3 #> 1         6.7         3.1         4.4         1.4 versicolor
4 #> 2         6.6         3.0         4.4         1.4 versicolor
```

Функція `arrange()`, що упорядковує дані від меншого до більшого по обраних змінних

```
1 iris |> arrange(Sepal.Length, Petal.Length) |> head(4)
2 #>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
3 #> 1         4.3         3.0         1.1         0.1  setosa
4 #> 2         4.4         3.0         1.3         0.2  setosa
5 #> 3         4.4         3.2         1.3         0.2  setosa
6 #> 4         4.4         2.9         1.4         0.2  setosa
```

Функція `distinct()` повертає усі унікальні значення

```
1 iris |> distinct(Species)
2 #>   Species
3 #> 1  setosa
4 #> 2 versicolor
5 #> 3 virginica
```

Функції для колонок

Функція `mutate()`, що дозволяє створити нову колонку з розрахунками, що виконані на основі даних з інших колонок

```
1 iris |> mutate(Petal.lw.ratio = Petal.Length / Petal.Width) |> head(2)
2 #>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Petal.lw.ratio
3 #> 1         5.1         3.5         1.4         0.2   setosa              7
4 #> 2         4.9         3.0         1.4         0.2   setosa              7
```

Функція `relocate()`, що дозволяє перемістити колонку

```
1 iris |> relocate(Species, .before = everything()) |> head(2)
2 #>   Species Sepal.Length Sepal.Width Petal.Length Petal.Width
3 #> 1   setosa         5.1         3.5         1.4         0.2
4 #> 2   setosa         4.9         3.0         1.4         0.2
```

Функція `rename()`, що дозволяє перейменувувати колонки

```
1 iris |> rename(Plant.Species = Species) |> head(2)
2 #>   Sepal.Length Sepal.Width Petal.Length Petal.Width Plant.Species
3 #> 1         5.1         3.5         1.4         0.2         setosa
4 #> 2         4.9         3.0         1.4         0.2         setosa
```

Та функція `select()`, що дозволяє вибрати конкретну колонку (або колонки) з кадру

```
1 iris |> select(Sepal.Length, Petal.Length) |> head(2)
2 #>   Sepal.Length Petal.Length
3 #> 1         5.1         1.4
4 #> 2         4.9         1.4
```

Групування даних

Функція `group_by()`, як очевидно з назви, створює групи даних, що відображується у метаданих кадру

```
1 iris |> group_by(Species) |> head(3)
2 #> # A tibble: 3 × 5
3 #> # Groups:   Species [1]
4 #>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
5 #>         <dbl>         <dbl>         <dbl>         <dbl> <fct>
6 #> 1         5.1         3.5         1.4         0.2 setosa
7 #> 2         4.9         3         1.4         0.2 setosa
8 #> 3         4.7         3.2         1.3         0.2 setosa
```

Функції `summarise()`, `reframe()` та `count()` дозволяють застосувати дескриптивні статистичні функції до груп даних

```
1 iris |>
2   group_by(Species) |>
3   summarise(mean_sepal_len = mean(Sepal.Length), n = n())
4 #> # A tibble: 3 × 3
5 #>   Species      mean_sepal_len      n
6 #>   <fct>          <dbl> <int>
7 #> 1 setosa         5.01    50
8 #> 2 versicolor     5.94    50
9 #> 3 virginica       6.59    50
```

Ітерації


Ітеративне застосування певної функції до декількох колонок усередині `mutate()` або `summarise()` може бути досягнуто за використання `across()`

```
1 iris |>
2   group_by(Species) |>
3   summarise(across(where(is.numeric), mean), n = n())
4 #> # A tibble: 3 × 6
5 #>   Species      Sepal.Length Sepal.Width Petal.Length Petal.Width      n
6 #>   <fct>          <dbl>         <dbl>         <dbl>         <dbl> <int>
7 #> 1 setosa         5.01           3.43           1.46           0.246    50
8 #> 2 versicolor     5.94           2.77           4.26           1.33     50
9 #> 3 virginica       6.59           2.97           5.55           2.03     50
```

Разом з `filter()` можуть бути застосовані `if_all()` та `if_any`

```
1 penguins |> filter(if_all(3:6, is.na))
2 #> # A tibble: 2 × 8
3 #>   species island      bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
4 #>   <fct>   <fct>          <dbl>          <dbl>          <int>         <int>
5 #> 1 Adelie Torgersen           NA              NA              NA             NA
6 #> 2 Gentoo  Biscoe           NA              NA              NA             NA
7 #> # i 2 more variables: sex <fct>, year <int>
```

Ітерації

Пакет **purrr** має аналог базового **lapply** під назвою **map()** (точніше ціле сімейство **map**, див. [документацію](#) ) , що картує функцію до кожного елементу листа або атомарного вектор.

```
1 # базовий варіант map() є функціонально ідентичним lapply()
2 str(lapply(1:3, function(x) rnorm(x)))
3 #> List of 3
4 #> $ : num 0.106
5 #> $ : num [1:2] 0.649 0.869
6 #> $ : num [1:3] 0.323 -0.775 1.372
7 str(map(1:3, function(x) rnorm(x)))
8 #> List of 3
9 #> $ : num -0.279
10 #> $ : num [1:2] 0.546 0.404
11 #> $ : num [1:3] 0.464 0.834 -1.978
```

Функція **map**, як **lapply**, може приймати скорочений варіант звернення до анонімної функції, а також має свій власний варіант синтаксису

```
1 map(1:3, function(x) rnorm(x))
2 # теж саме що і
3 map(1:3, \(x) rnorm(x))
4 # теж саме що і purr-стиль звернення до функції
5 map(1:3, ~ rnorm(.x))
```

Також існує **map2**, що може виконувати ітерацію двох аргументів одночасно

Selector helpers

`tidyselect` надає низку “допоміжних” функцій, що можуть бути використані усередині таких команд як `select`, `rename`, `relocate` та `across` та деяких інших для вибору колонок, що задовольняють певні умови. Приклади використання деяких допоміжних функцій вже були на минулих слайдах.

Селекція за певним патерном:

- `starts_with("a")` — усі змінні, чиє ім'я починається з певного префіксу
- `ends_with("z")` — усі змінні, чиє ім'я закінчується певним суфіксом
- `contains("abc")` — усі змінні, що містять певну послідовність літер у назві
- `matches("a.c")` — усі змінні, що містять певний RegEx патерн у назві
- `num_range("wk", 1:3)` — усі змінні, що містять певний патерн з числами, e.g. wk1, wk2, wk3

Селекція за певною умовою:

- `where(is.numeric)` — усі змінні, що задовольняють логічну умову (`is.numeric`, `is.character` т.д.)
- `everything()` — просто усі змінні
- `last_col()` — остання змінна

*_join

Сімейство функцій аналогів базового `merge()`, що покликані виконувати операцію об'єднання двох кадрів даних у один. Два іграшкових набори даних для прикладу

```
1 kbl(rare_species)
```

species	protection
Galanthus nivalis	red book
Liparis loeselii	red book & BERN
Frittilaria meleagris	red book
Stipa borysthena	red book

```
1 kbl(obs_species)
```

species	n_obs
Liparis loeselii	1
Asclepias syriaca	28
Ballota nigra	15
Galanthus nivalis	2

`dplyr` має шість варіантів функції `*_join`: `left_join()`, `right_join()`, `inner_join()`, `semi_join()`, `anti_join()` та `full_join()`. Усі вони приймають два кадри даних (`x` та `y`) та повертають один кадр даних, об'єднаний на основі пар ключів. *Первинний ключ* це змінна (або змінні), що є унікальним ідентифікатором для кожного рядка спостереження. У прикладі вище `rare_species$species` є *первинним ключем*, що унікально ідентифікує певний вид рослини, що є захищеним у рамках того чи іншого документу. Для об'єднання первинний ключ `rare_species$species` буде порівнюватися з *зовнішнім ключем* `obs_species$species`. Ключі можуть складатися з однієї змінної, як у прикладі вище, або з декількох змінних (складені ключі)

left_join та right_join

Функція `left_join(x, y)` створює новий кадр, який містить у собі усі рядки з кадру `x`

species	protection
Galanthus nivalis	red book
Liparis loeselii	red book & BERN
Frittilaria meleagris	red book
Stipa borysthenica	red book

species	n_obs
Liparis loeselii	1
Asclepias syriaca	28
Ballota nigra	15
Galanthus nivalis	2

```
1 left_join(rare_species, obs_species)
```



species	protection	n_obs
Galanthus nivalis	red book	2
Liparis loeselii	red book & BERN	1
Frittilaria meleagris	red book	NA
Stipa borysthenica	red book	NA

left_join та right_join

Функція `right_join(x, y)` створює новий кадр, який містить у собі усі рядки з кадру `y`

species	protection
Galanthus nivalis	red book
Liparis loeselii	red book & BERN
Frittilaria meleagris	red book
Stipa borysthenica	red book

species	n_obs
Liparis loeselii	1
Asclepias syriaca	28
Ballota nigra	15
Galanthus nivalis	2

```
1 right_join(rare_species, obs_species)
```

species	protection	n_obs
Galanthus nivalis	red book	2
Liparis loeselii	red book & BERN	1
Asclepias syriaca	NA	28
Ballota nigra	NA	15

inner_join та full_join

Функція `inner_join(x, y)` повертає лише ті значення, що одночасно є у кадрі `x` та у кадрі `y`

```
1 inner_join(rare_species, obs_species)
```

species	protection	n_obs
Galanthus nivalis	red book	2
Liparis loeselii	red book & BERN	1

Функція `full_join(x, y)` повертає об'єднання між `x` та `y`

```
1 full_join(rare_species, obs_species)
```

species	protection	n_obs
Galanthus nivalis	red book	2
Liparis loeselii	red book & BERN	1
Frittilaria meleagris	red book	NA
Stipa borysthenica	red book	NA
Asclepias syriaca	NA	28
Ballota nigra	NA	15

semi_join та anti_join

Функція `semi_join(x, y)` повертає усі рядки з `x` які мають пару у `y`

```
1 semi_join(rare_species, obs_species)
```

species	protection
Galanthus nivalis	red book
Liparis loeselii	red book & BERN

Функція `anti_join(x, y)` повертає усі рядки з `x` до яких немає пари у `y`

```
1 anti_join(rare_species, obs_species)
```

species	protection
Frittilaria meleagris	red book
Stipa borysthenica	red book

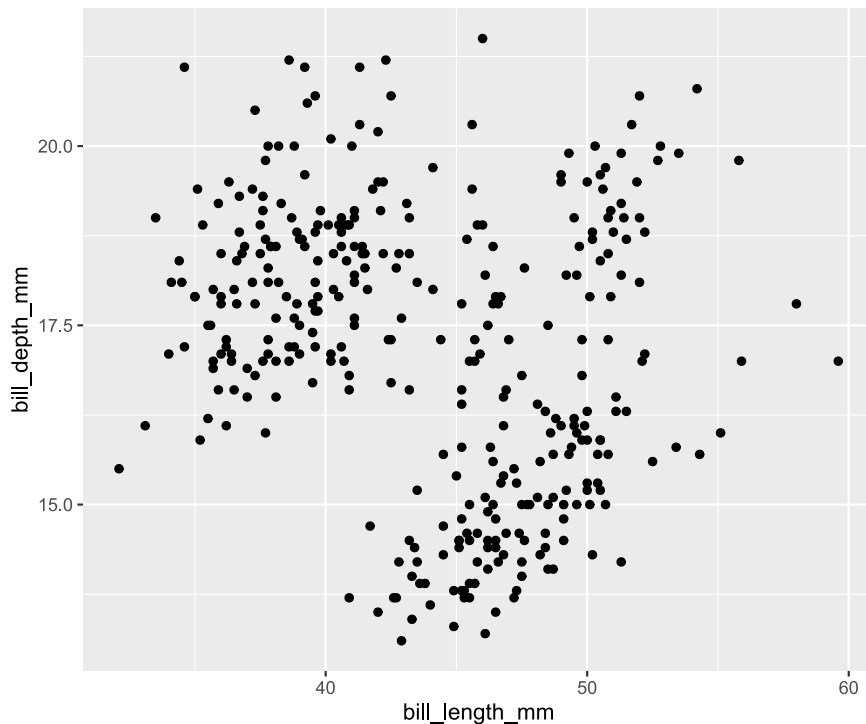
Найпростіший графік с ggplot2

Базовий синтаксис ggplot2 має вигляд

```
1 ggplot(data, aes(x, y)) + # дані та осі
2   geom()                  # одна з доступних геометрій
```

Де `ggplot()` створює об'єкт класу `gg` на який надалі за допомогою оператора `+` накладаються нові шари специфікації того, що у термінології `ggplot2` буквально називається естетикою

```
1 penguins |>
2   ggplot(aes(x = bill_length_mm, y = bill_depth_mm)) +
3   geom_point()
```

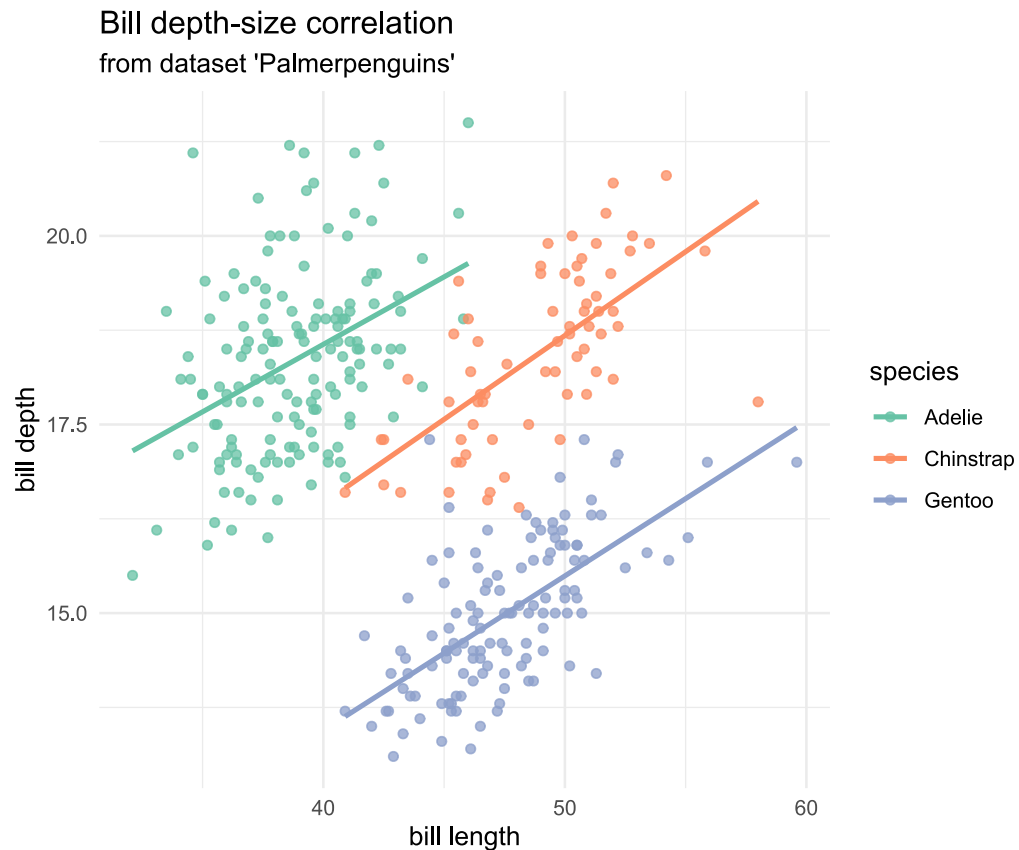


Низка інших геометрій

- `geom_jitter`
- `geom_bar`, `geom_col`
- `geom_histogram`, `geom_density`
- `geom_boxplot`, `geom_violin`
- `geom_line`, `geom_smooth`
- `geom_errorbar`
- `stat_summary`

Найпростіший графік с ggplot2

```
1 penguins |>
2   ggplot(aes(bill_length_mm, bill_depth_mm, color = species)) + # ще одна естетика
3   geom_point(alpha = .75) +
4   geom_smooth(method = lm, se = F) + # додаткова геометрія
5   labs(x = "bill length", y = "bill depth", title = "Bill depth-size correlation",
6        subtitle = "from dataset 'Palmerpenguins'") +
7   scale_color_brewer(palette = "Set2") +
8   theme_minimal()
```



P.S. реальні дані з реального життя

Очевидно неповний список лінків звідки можливо дістати реальні набори даних відносно великих розмірів аби попрактикуватися у візуалізації даних, а також статистичному моделюванні та машинному навчанні

- [R4DS Online Learning Community \(2023\). Tidy Tuesday: A weekly social data project.](#) — GitHub репозиторій проєкту TidyTuesday, новий датасет кожен понеділок
- [Awesome public datasets core](#) — GitHub репозиторій, що колекціонує лінки на публічні датасети, по категоріям
- [Архів проєкту Inter-university Consortium for Political and Social Research \(ICPSR\)](#)
- [Гарвардський Dataverse архів](#)
- [UC Irvine Machine Learning Repository](#)
- [European Data Portal](#) — портал-каталог даних з країн Європи
- [Papers With Code](#) (ці більше спрямовані конкретно на машинне навчання)
- Мультидисциплінарний open-access журнал [Data in Brief](#)
- [GitHub репозиторій з колекцією журналів, які спеціалізуються на публікації наборів даних](#)
- Пакети наборів даних, що є додатками до книг серії [OpenIntro](#) та [ISLR2](#)