




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)`:

- **ggplot2** — декларативна система створення графіки, основана на філософії представлених 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
- **purrr** — тулкіт для функціонального програмування з R
- **tibble** — інтродукує новий клас формату даних аналогічний до `data.frame` — `tbl_df`
- **stringr** — пакет для маніпуляції з текстовими даними
- **forcats** — пакет для маніпуляції з факторами

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, 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](#)

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  0.760 -0.701 -0.533 a  
11 #> 2  0.560  1.04   0.582 b  
12 #> 3  0.805 -1.62  -1.31  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
```

Функціональні еквіваленти

Пакет dplyr містить набір функцій низка з яких є аналогічними або ідентичними базовим функціям R, але на відміну від них мають 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
arrange(df, x)	order(x) , df[!duplicated(x), , drop = F]
distinct(df, x)	unique()
filter(df, x)	subset() or df[which(x), , drop = F]
pull(df, 1) , pull(df, x)	df[[1]], df\$x
rename(df, y = x) , rename_with()	names() or stats::setNames()
mutate(x = y + z)	df\$x <- df\$y + df\$z, transform()
select(df, x, y)	subset() or df[c("x", "y")]
summarise(df, fun(x))	tapply() , aggregate(), by()
slice(df, c(1, 2, 5))	df[c(1, 2, 5), , drop = FALSE], also partialy sample()
*_join()	merge()

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

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_exaple
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_exaple |>
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   )
```


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 переіменувувати 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 -1.01
5 #> $ : num [1:2] 0.621 1.122
6 #> $ : num [1:3] 0.332 -0.462 -0.21
7 str(map(1:3, function(x) rnorm(x)))
8 #> List of 3
9 #> $ : num -0.637
10 #> $ : num [1:2] -1.23 0.22
11 #> $ : num [1:3] 1.57 0.337 1.339
```

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, зручно для використання при ітеративному збереженні даних або графіків

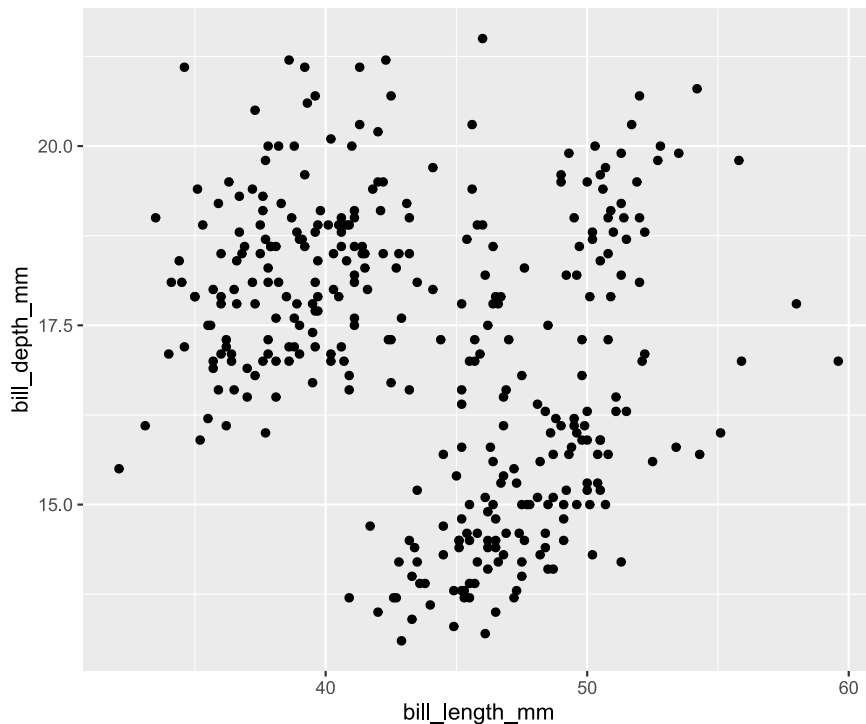
Найпростіший графік с ggplot2

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()
```

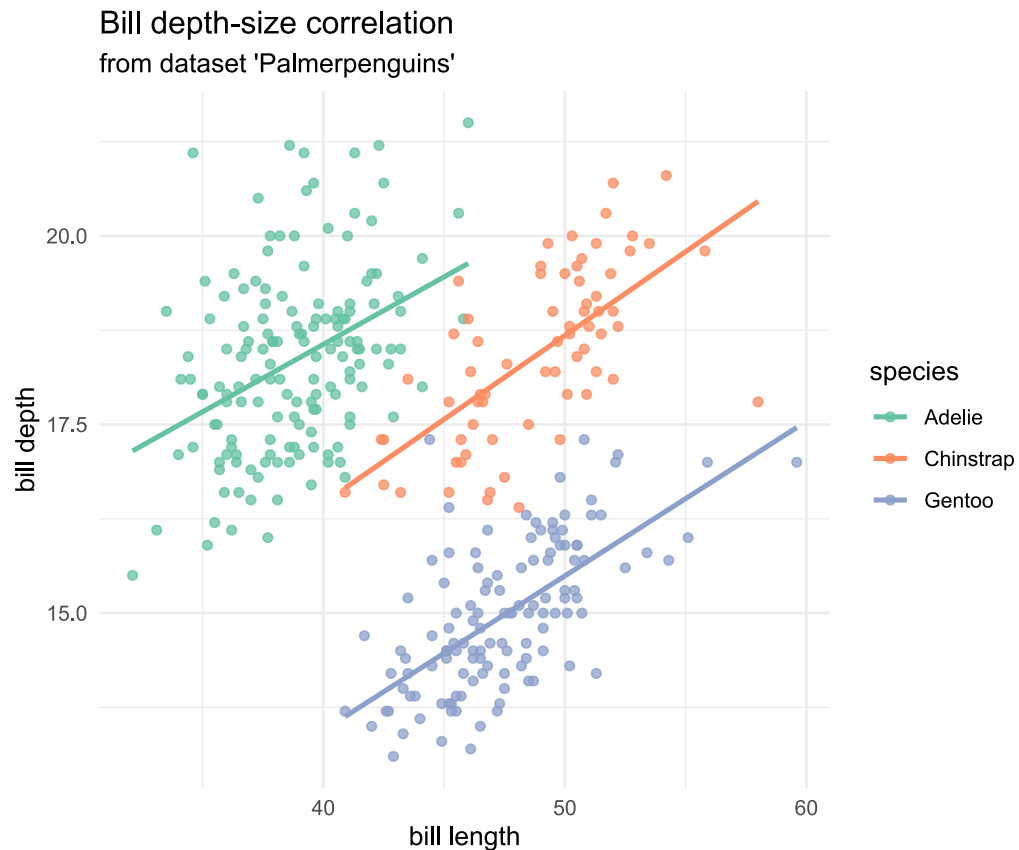


Низка інших геометрій

- `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](#)
- [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](#)