PH 718 Data Management and Visualization in R

Part 7: Introduction to `tidyverse`

Zhiyang Zhou (zhou67@uwm.edu, zhiyanggeezhou.github.io)

2026/03/31 00:57:53

What is the `tidyverse` (https://www.tidyverse.org/)

A collection of `R` packages designed for data science. These packages share a common philosophy, grammar, and data structure, making data manipulation, visualization, and modeling easier and more efficient.

Table 1: Summary of Key `tidyverse` Packages

Package	Purpose
`ggplot2`	Data visualization
`dplyr`	Data manipulation
`tidyr`	Rreshaping data
`readr`	Reading CSV and text files
`tibble`	Modern replacement for data frames
`purrr`	Functional programming (working with lists)
`stringr`	String manipulation
`forcats`	Handling categorical data (factors)

Why use `tidyverse`

- Consistency: Functions follow similar syntax and principles.
- Readability: Code is easier to write and understand.
- Efficiency: Optimized for modern data workflows.
- Integration: `tidyverse` functions work well together.

Installing and loading `tidyverse`

```r
install.packages("tidyverse")
library(tidyverse) # load ALL core tidyverse packages
```

If you need a specific package, you can load it individually, e.g.,

```r
library(dplyr)
```

Toy examples comparing `tidyverse` and base `R`

Data manipulation with `dplyr`

```r
mtcars_filtered1 =
  mtcars %>%
  filter(cyl == 6) %>%        # Keep only cars with 6 cylinders
  select(mpg, hp, wt) %>%     # Select relevant columns
  arrange(desc(mpg))          # Arrange in descending order of mpg
```

or, equivalently, without using functions in `tidyverse`,

```r
mtcars_filtered3 <- mtcars[mtcars$cyl == 6, ]
mtcars_filtered3 <- mtcars_filtered3[, c("mpg", "hp", "wt")]
mtcars_filtered3 <- mtcars_filtered3[order(-mtcars_filtered3$mpg), ]
```

Creating elegant, layered plots with `ggplot2`

```r
ggplot(mtcars, aes(x = wt, y = mpg, color = factor(cyl))) +
  geom_point() +
  theme_minimal()
```

or, similarly, without using `ggplot2`

```r
colors <- c("red", "blue", "green")
cyl_factor <- as.factor(mtcars$cyl)
color_map <- colors[as.numeric(cyl_factor)]
plot(mtcars$wt, mtcars$mpg, col = color_map, pch = 16,
     xlab = "Weight (wt)", ylab = "Miles per Gallon (mpg)",
     main = "Scatter plot of wt vs mpg by cyl")
legend("topright", legend = levels(cyl_factor), col = colors, pch = 16, title = "Cylinders")
```

Reshaping data with `tidyr`

```r
# Converting wide data to the long format
wide_data <- data.frame(
  name = c("Alice", "Bob", "Cindy"),
  math = c(90, 85, 69),
  science = c(88, 92, 99)
)
long_data1 <- wide_data %>%
  pivot_longer(cols = math:science, names_to = "subject", values_to = "score")
# gather() is similar but is superseded by pivot_longer()
```

or, equivalently, without using functions in `tidyverse`,

```r
# Converting wide data to the long format
wide_data <- data.frame(
  name = c("Alice", "Bob", "Cindy"),
  math = c(90, 85, 69),
  science = c(88, 92, 99)
)
long_data2 <- reshape(
  wide_data,
  varying = c("math", "science"),
  v.names = "score",
  timevar = "subject",
  times = c("math", "science"),
  direction = "long"
)
rownames(long_data2) <- NULL
long_data2 = subset(long_data2, select = -id)
```

`tibble`

What is a tibble?

- A modern version of data frame optimized for better readability and usability in `R`
- Working seamlessly with packages belonging to `tidyverse`
- Rumor: "Tibble" is similar to how New Zealanders pronounce "table", and Dr. Hadley Wickham, the creator of `tibble`, is from New Zealand.

Creating tibbles

```r
# Creating a tibble manually with tibble::tibble()
students1 <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  age = c(23, 25, 22),
  score = c(90, 85, 88)
)
students1
```

```r
# For small datasets, tibble::tribble() allows for a more intuitive way
students2 <- tribble(
  ~name,    ~age, ~score,
  "Alice",    23,    90,
  "Bob",      25,    85,
  "Charlie",  22,    88
)
students2
```

```r
# Converting between tibbles and data frames
iris
iris_t <- as_tibble(iris)
iris_t
iris_df = as.data.frame(iris_t)
iris_df
```

Accessing subsets of a tibble

```r
iris_t$Species
iris_t[,c("Petal.Length","Petal.Width","Species")]
iris_t[iris_t$Species=='setosa', ]
subset(iris_t, select = c(Petal.Length, Petal.Width, Species))
subset(iris_t, select = -c(Petal.Length, Petal.Width, Species))
```

Additional advantages of tibbles

```r
# Tibbles print out only the first 10 rows and all the columns that fit on screen
starwars
as.data.frame(starwars)
```

Ex. 7.1. Mini exercise on tibbles

- Convert `moderndive::evals` to a tibble and store it in an object called `evals_t`.
- Create a tibble containing only the following columns from `evals_t`: `score`, `age`, and `gender`.
- Create a tibble that contains only the rows where the instructor's gender is female.
- Create a tibble that contains all columns except: `pic_outfit`, `pic_color`.
- Create a tibble that contains only instructors older than 50 and display only columns `score`, `age` and `rank`.

```r
library(tidyverse)
library(moderndive)
evals_t = as_tibble(moderndive::evals)
evals_t_sub1 = subset(
  evals_t,
  select = c(score, age, gender)
)
evals_t_sub2 = subset(
  evals_t,
  gender == 'female'
)
evals_t_sub3 = subset(
  evals_t,
  select = -c(pic_outfit, pic_color)
)
evals_t_sub4 = subset(
  evals_t,
  age > 50 & gender == 'male',
  select = c(score, age, rank)
)
```

Pipe operators `%>%` and `|>`

What is the pipe operator

- A fundamental feature in `tidyverse`
- Designed to improve code readability and workflow efficiency
  - Chaining multiple operations together in a logical sequence
  - Making data manipulation in `R` clearer and more intuitive

History of piping

- 1970s: Pipe `|` was introduced in Unix, allowing command-line programs to pass output to the next command;
- Languages like F#, Elixir, and Haskell have similar piping mechanisms;
- 2014: `%>%` was introduced in `R` package `magrittr`;
- 2016: `%>%` was integrated into `tidyverse`;
- 2021: `|>` was introduced in base R 4.1 and an alternative for users who prefer not to load `tidyverse`.

Why use the pipe operator

```r
# Sample dataset
student_data <- tibble(
  name = c("Alice", "Bob", "Charlie", "David", "Eve", "Frank"),
  score = c(85, 92, 78, NA, 95, 88)
)
```

Given the above dataset of student exam scores, we want to: 1. Remove any students with missing scores (NA). 2. Standardize the scores. 3. Screen out students who scored below average after standardization. 4. Sort the remaining students by their standardized scores in a descending order.

If we use multiple intermediate variables, then

```r
clean_data <- na.omit(student_data)
clean_data$z_score <- scale(clean_data$score, center = TRUE, scale = TRUE)
filtered_data <- clean_data[clean_data$z_score >= 0, ]
sorted_data <- filtered_data[order(-filtered_data$z_score), ]
sorted_data

## Problem: requires multiple intermediate variables, making the workflow less readable.
```

If we avoid intermediate variables by using a nested function, then

```r
subset(
  transform(
    na.omit(student_data),
    z_score = (score - mean(score)) / sd(score)
  ),
  z_score > 0
)[order(-subset(
  transform(
    na.omit(student_data),
    z_score = (score - mean(score)) / sd(score)
  ),
  z_score > 0
)$z_score), ]

## Problem: reversed order of operations (we must read inside-out), making the workflow less readable
```

If we use `%>%` or `|>`, then

```r
student_data %>%
  na.omit() %>%  # Step 1: Remove missing values
  transform(z_score = (score - mean(score)) / sd(score)) %>%  # Step 2: standardize scores
  subset(z_score >= 0) %>%  # Step 3: Keep students with above-average scores
  .[order(-.$z_score), ] # Step 4: Sort in descending order

## More readable, following a left-to-right logic, no need for temporary variables, and easier to debug
```

How the pipe operators (`%>%` and `|>`) work

- `%>%` is part of `tidyverse`; `|>` is part of `base`.

- Feature 1: Passing the output of one function as the first argument to the next function
    - This feature is shared by `%>%` and `|>`

```r
mean(mtcars$mpg)
# equiv.
mtcars$mpg %>% mean()
```

```r
head(iris)
# equiv.
iris %>% head()
# equiv.
iris |> head()
```

```r
set.seed(718)
x <- runif(10)

round(exp(diff(log(x))), 1)
# equiv.
x %>% log() %>% diff() %>% exp() %>% round(1)
```

- Feature 2: Passing sth. to positions where placeholders (dots) lie
    - This is for `%>%` only

```r
student_data <- data.frame(
  name = c("Alice", "Bob", "Charlie", "Eve", "Frank"),
  score = c(85, 92, 78, 95, 88)
)
student_data %>% .[order(.$score), ]
# equiv.
student_data[order(student_data$score), ]
```

Ex. 7.2. Mini exercise on the pipe operator

- In `dplyr::starwars`, compute the average height of characters, excluding missing values.
- Create a new variable called `bmi_like`, defined as: `bmi_like=mass/height^2`. (Hint: `transform` function may help.)
- Use `%>%` together with the dot placeholder `.` to sort the dataset by height.

```r
library(tidyverse)
starwars$height |> mean(, na.rm = T)
starwars_bmi = starwars |> transform(bmi_like=mass/height^2)
starwars_ordered = starwars %>% .[order(.$height), ]
```

`tidyr`

Key Functions in `tidyr`

The `tidyr` package provides functions for reshaping and tidying data efficiently. Below is a summary of some of the most commonly used functions:

Table 2: Summary of Key `tidyr` Functions

Function	Purpose
`pivot_longer()`	Convert wide data to long format
`pivot_wider()`	Convert long data to wide format
`separate()`	Split a column into multiple columns
`unite()`	Merge multiple columns into one
`fill()`	Fill in missing values
`drop_na()`	Remove missing values

Reshaping data

One of the most common tasks in `tidyr` is reshaping wide data into long format

- Long format: each row corresponds to a unique combination of a row id and a column name

```r
wide_data <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  math = c(90, 85, 78),
  science = c(88, 92, 80),
  statistics = c(56, 25, 35)
)
# Wide -> long
long_data <- wide_data %>% pivot_longer(
  cols = c(math, science, statistics),
  names_to = "subject",
  values_to = "score"
)
long_data

# Long -> wide
wide_again <- long_data %>% pivot_wider(
  names_from = subject,
  values_from = score
)
wide_again
```

Splitting columns

```r
library(tidyverse)
data <- data.frame(
  full_name = c("Alice.Johnson", "Bob.Smith")
)
# Split full_name into first_name and last_name
separated_data <- data %>% separate(
  full_name,
  into = c("first_name", "last_name"),
  sep = "\\."
)
separated_data
```

Merging columns

```r
data <- data.frame(
  first_name = c("Alice", "Bob"),
  last_name = c("Johnson", "Smith")
)
# Merge first_name and last_name into full_name
united_data <- data %>%
  unite(full_name, first_name, last_name, sep = " ")
united_data
```

Removing rows with missing values

```r
data <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  score = c(90, NA, 85)
)
clean_data <- data %>% drop_na()
clean_data
```

Filling missing values with neigbours

```r
pets <- tibble::tribble(
  ~rank, ~pet_type, ~breed,
  1L,        NA,    "Boston Terrier",
  2L,        NA,    "Retrievers (Labrador)",
  3L,        NA,    "Retrievers (Golden)",
  4L,        NA,    NA,
  5L,        NA,    "Bulldogs",
  6L,     "Dog",    "Beagles",
  1L,        NA,    "Persian",
  2L,        NA,    "Maine Coon",
  3L,        NA,    "Ragdoll",
  4L,     "Cat",    "Exotic",
  5L,        NA,    "Siamese",
  6L,        NA,    "American Short"
)
pets_filled1 <- pets %>% fill(breed, .direction = "down")
pets_filled2 <- pets %>% fill(
  pet_type, .direction = "updown"
)
```

Filling missing values with customized values

```r
pets$pet_type %>% replace_na('UnknownType')
pets_filled3 = pets %>% replace_na(
  list(pet_type = 'UnknownType', breed = 'UnknownBreed')
)
```

Ex. 7.3. Mini exercise on tidyr

- Use `drop_na()` to remove rows with missing `height` in `dplyr::starwars`.
- Create a new column combining `name` and `gender`: `mutate(name_gender = paste(name, gender, sep = "_"))`. Then use `separate()` to split this column into `character_name` and `character_gender`.
- Reshape data (wide -> long). Select the following variables: `name`, `height`, and `mass`. Convert the dataset into long format using `pivot_longer()` so that:
  - the measurement names (`height`, `mass`) appear in a column called `variable` and
  - the values appear in a column called `value`.

```
data = dplyr::starwars
data_1 = data |> drop_na(height)
data_2 = data |>
  mutate(name_gender = paste(name, gender, sep = "_")) |>
  separate(
    name_gender,
    into = c("character_name", "character_gender"),
    sep = "_"
  )
data_3 = data |>
  subset(select = c(name, height, mass)) |>
  pivot_longer(
    cols = c(height, mass),
    names_to = "variable",
    values_to = "value"
  )
```

`dplyr`

Why use `dplyr`

- Simplifies data manipulation (compared to base `R`)
- Readable and intuitive syntax (using verb-based functions)
- Faster execution (optimized with `C++`)
- Works seamlessly with `tibbles` and other `tidyverse` members
- Uses the pipe operator for chaining multiple operations

Key functions in `dplyr`

Function	Purpose
`filter()`	Select rows based on conditions
`slice()`	Select specific rows by position
`select()`	Choose specific columns
`mutate()`	Create or modify columns
`summarize()`	Compute summary statistics
`group_by()`	Group data for aggregation
`arrange()`	Sort rows by column values
`distinct()`	Remove duplicate rows

Function	Purpose
`join()` functions	Merge datasets (`left_join()`, `inner_join()`, etc.)

Removing duplicate rows

```r
students <- tibble(
  name = c("Alice", "Bob", "Charlie", "Alice", "Eve", "Charlie", "Eve"),
  sex = c("F", "M", "M", "F", "F", "M", "F"),
  subject = c("Math", "Math", "Science", "Math", "Science", "Writing", "Science"),
  score = c(88, 92, 85, 88, 78, 86, 78)
)
students

# Removing exact duplicate rows
students %>% distinct()

# Removing duplicates based on one column
students %>% distinct(name, .keep_all = TRUE)

# Removing duplicates based on multiple columns
students %>% distinct(name, subject, .keep_all = TRUE)
```

Selecting rows based on conditions

```r
mtcars %>% filter(cyl == 6)
# base R alternative: mtcars[mtcars$cyl == 6, ]
# equiv. mtcars |> subset(cyl == 6)
```

Selecting rows by position

```r
starwars %>% slice(2,4)  # Select the 2nd and 4th rows
# base R alternative: starwars[c(2,4), ]

starwars %>% slice_head(n = 3) # Select the first 3 rows
# base R alternative: head(starwars, 3)

starwars %>% slice_tail(n = 2)  # Select the last 2 rows
# base R alternative: tail(starwars, 2)

set.seed(718)  # Ensure reproducibility
starwars %>% slice_sample(n = 3)  # Randomly select 3 rows
# base R alternative: starwars[sample(nrow(starwars), 3),]

starwars %>% slice_max(order_by = height, n = 2)  # Select top 2 highest characters
# base R alternative: starwars[order(-starwars$height), ][1:2, ]
```

```
starwars %>% slice_min(order_by = height, n = 2)  # Select top 2 lowest characters
# base R alternative: starwars[order(starwars$height), ][1:2, ]
```

Selecting columns

```
mtcars %>% select(mpg, hp, wt)
# base R alternative: subset(mtcars, select = c(mpg, hp, wt))
# equiv. mtcars |> subset(section = c(mpg, hp, wt))

mtcars %>% select(-mpg, -hp, -wt)
mtcars %>% select(-c(mpg, hp, wt))
mtcars %>% select(!c(mpg, hp, wt))
# base R alternative: subset(mtcars, select = -c(mpg, hp, wt))
```

Creating new columns that are functions of existing variables

```
mtcars2 =  mtcars  %>% mutate(weight_kg = wt * 1000)
# base R alternative: mtcars$weight_kg <- mtcars$wt*1000
```

Summarizing data

```
mtcars %>% summarize(avg_mpg = mean(mpg), max_hp = max(hp))
mtcars %>% summarise(avg_mpg = mean(mpg), max_hp = max(hp))
# base R alternative: data.frame(avg_mpg = mean(mtcars$mpg), max_hp = max(mtcars$hp))
```

Grouping data by one or more variables before further move

```
mtcars %>%
  group_by(cyl) %>%
  summarize(avg_mpg = mean(mpg), max_hp = max(hp)) %>%
  ungroup()  # ensures further operations are done on the entire data frame
# (imperfect) base R alternative:
# aggregate(cbind(mpg, hp) ~ cyl, data = mtcars, FUN = function(x) c(mean = mean(x), max = max(x)))

mtcars %>%
  group_by(cyl, gear) %>%
  summarize(avg_mpg = mean(mpg), max_hp = max(hp)) %>%
  ungroup()
# (imperfect) base R alternative:
# aggregate(cbind(mpg, hp) ~ cyl+gear, data = mtcars, FUN = function(x) c(mean = mean(x), max = max(x))

mtcars %>%
  group_by(cyl, gear) %>%
  count() %>%
  ungroup()
# (imperfect) base R alternative:
```

```
# aggregate(mpg ~ cyl+gear, data = mtcars, FUN = length)

mtcars2 = mtcars %>%
  group_by(cyl, gear) %>%
  mutate(avg_mpg_cyl_gear = mean(mpg)) %>%
  ungroup()
# (imperfect) base R alternative:
# group_means <- aggregate(mpg ~ cyl + gear, data = mtcars, FUN = mean)
# colnames(group_means)[3] <- "avg_mpg_cy_gear"
# merge(mtcars, group_means, by = c("cyl", "gear"))
```

Arranging rows

```
mtcars %>% arrange(mpg) # Sort by mpg in ascending order
# base R alternative: mtcars[order(mtcars$mpg), ]

mtcars %>% arrange(desc(mpg)) # Sort by mpg in descending order
# base R alternative: mtcars[order(-mtcars$mpg), ]

mtcars %>% arrange(cyl, desc(mpg)) # Sort by cyl (ascending), then by mpg (descending)
# base R alternative: mtcars[order(mtcars$cyl, -mtcars$mpg), ]

mtcars %>% arrange(rownames(mtcars)) # Sort by cars name in alphabetical order
# base R alternative: mtcars[order(rownames(mtcars)), ]
```

Concatenating multiple functions

```
mtcars %>%
  filter(cyl == 6) %>%
  select(mpg, hp, wt) %>%
  arrange(desc(mpg))
# base R alternative:
# mtcars_filtered <- mtcars[mtcars$cyl == 6, ]
# mtcars_selected <- mtcars_filtered[, c("mpg", "hp", "wt")]
# mtcars_sorted <- mtcars_selected[order(-mtcars_selected$mpg), ]
```

Merging datasets

```
# two data frames on student names and scores, respectively
p_names <- data.frame(
  patient_id = c(1, 2, 3, 4),
  name = c("Alice", "Bob", "Charlie", "David")
)
results <- data.frame(
  patient_id = c(2, 3, 5),
  exam_res = c(85, 90, 78)
)
p_names
```

```r
results

# Keeps all rows from p_names, adding available results
# Use left_join() when your primary dataset is on the left
left_join(p_names, results, by = "patient_id")

# Keeps all rows from results, adding available p_names
# Use right_join() when your primary dataset is on the right
right_join(p_names, results, by = "patient_id")

# Keep only matched rows
# Use inner_join() when you only need matched records
inner_join(p_names, results, by = "patient_id")

# Keep all rows from both tables
# Use full_join() when you want to retain everything
full_join(p_names, results, by = "patient_id")
```

Ex.7.4. Mini exercise on `dplyr`

- Create a toy dataset on online store orders:

```r
library(tidyverse)
orders <- tibble(
  order_id = c(1,2,3,4,5,6,7,8,9,10),
  customer = c("Alice","Bob","Alice","David","Eve","Bob","Alice","Eve","David","Alice"),
  product = c("Book","Pen","Notebook","Book","Pen","Notebook","Book","Book","Pen","Notebook"),
  category = c("A","B","A","A","B","A","A","A","B","A"),
  price = c(12,3,8,12,3,8,12,12,3,8),
  quantity = c(1,5,2,1,10,1,3,2,4,1)
)
```

- Keep only orders where `category == "A"` and select only `customer`, `product`, and `price`.
- Create a new column `total_price = price * quantity`.
- Sort the dataset by `total_price` in descending order.
- For each customer, compute the total amount spent (`sum(total_price)`) and the average order value (`mean(total_price)`).
- Get a list of unique customers and products they bought (no duplicate combinations).
- Get the top 3 most expensive orders (based on `total_price`).
- For category "A", implement the following tasks in ONE pipeline:
  - create total_price
  - group by customer
  - compute total spending
  - sort from highest to lowest spender
- Create another dataset as below and join it with dataset `orders` to add `city` info (keep all orders).

```r
customers_info <- tibble(
  customer = c("Alice","Bob","David","Eve","Frank"),
  city = c("NY","LA","Chicago","NY","Boston")
)
```

`stringr`

Providing functions for manipulating strings in `R`.

Measuring string length

Find the number of characters in each character's name:

```r
starwars %>%
  mutate(name_length = str_length(name)) %>%
  select(name, name_length) %>%
  head()
```

Converting case

Convert the character names to uppercase/lowercase:

```r
starwars %>%
  mutate(
    name_upper = str_to_upper(name),
    name_lower = str_to_lower(name)
  ) %>%
  select(name, name_upper, name_lower) %>%
  head()
```

Concatenating strings

Create a new column that concatenates the character's name with their species:

```r
starwars %>%
  mutate(name_species = str_c(name, " (", species, ")")) %>%
  select(name_species) %>%
  head()
```

Dectecting patterns

Filter out characters with "Sky" in names:

```r
starwars %>%
  filter(str_detect(name, "Yo"))
```

Replacing text

Replace the first occurrence/all occurrences of "a" with "X" in each name:

```r
starwars %>%
  mutate(name_replaced = str_replace(name, "a", "X")) %>%
  select(name, name_replaced) %>%
  head()

starwars %>%
  mutate(name_replaced = str_replace_all(name, "a", "X")) %>%
  select(name, name_replaced) %>%
  head()
```

Ex.7.5. Mini exercise on `stringr`

```r
library(tidyverse)
movies <- tibble(
  title = c(
    "The Dark Knight",
    "Inception",
    "Interstellar",
    "The Matrix",
    "Avatar",
    "Titanic",
    "The Avengers",
    "Jurassic Park"
  ),
  genre = c(
    "Action", "Sci-Fi", "Sci-Fi", "Sci-Fi",
    "Fantasy", "Romance", "Action", "Adventure"
  )
)
```

- Create a column `title_length` that counts the number of characters in each movie title.
- Create two new columns: `title_upper` (all uppercase) and `title_lower` (all lowercase).
- Create a new column: `title_genre` formatted like `The Dark Knight [Action]`.
- Filter out movies with "The" in the title.
- Replace the first occurrence of `"a"` with `"@"` in each title and then replace all occurrences of `"a"` with `"@"`.
- Do everything in ONE pipeline. For movies in `"Sci-Fi"`:
  - Convert `title` to uppercase.
  - Create `title_length`that counts the number of characters in each movie title.
  - Keep only titles with `length > 10`.
  - Select only `title` and `title_length`.

`ggplot2`

Why use `ggplot2`

- One of the most powerful and flexible visualization tools in `R`
- Building plots layer by layer
- Providing highly customizable and publication-quality visualizations

Basic syntax of `ggplot2`

`ggplot2` follows a layered grammar of graphics, where each component adds a layer to the plot.

```
ggplot(
  data = <DATA>, # Initializes the plot with a dataset
  aes(
    x = <X-VAR>, y = <Y-VAR>, color = <COLOR-VAR>, # maps data variables to visual properties
    ...
  )
) +
  <GEOM_FUNCTION>() +  # geometric shapes (i.e., type of plot; e.g., points, bars and lines)
  <SCALE_FUNCTION>() + #  sizes and scales
  <FACET_FUNCTION>() + # creates subplots
  <THEME_FUNCTION>() + # customizes the plot appearance
  <LABEL_FUNCTION>()   # labels and titles
```

Scatter plot

A scatter plot visualizes the relationship between two numerical variables (e.g., `mass` vs `height`).

```
starwars %>%
  ggplot(
    aes(x = mass, y = height)
  ) +
  geom_point() +  # scatter plot
  theme_bw() + # background theme
  labs( # titles and axis labels
    title = "Height vs. Mass of Star Wars Characters",
    x = "Mass (kg)", y = "Height (cm)"
  )
```

Filter out NAs and extreme values before plotting.

```
starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  ggplot(
    aes(x = mass, y = height)
  ) +
  geom_point() +
  labs(
    title = "Height vs. Mass",
    x = "Mass (kg)", y = "Height (cm)"
  ) +
  theme_bw()
```

Fit a trend line with prediction intervals using lm, loess, etc

loess = local polynomial regression

```
starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  ggplot(
    aes(x = mass, y = height)
  ) +
  geom_point() +
  labs(
    title = "Height vs. Mass",
    x = "Mass (kg)", y = "Height (cm)"
```

```r
  ) +
  theme_bw()+
  geom_smooth(method = loess, se = F)
```

Group points and color them by a categorical variable (e.g., `sex`).

```r
starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  replace_na(list(sex = 'Unknown')) %>%
  ggplot(
    aes(x = mass, y = height, color = sex)
  ) +
  geom_point() +
  labs(
    title = "Height vs. Mass",
    x = "Mass (kg)", y = "Height (cm)", color = "sex_no_na"
  ) +
  theme_light()+
  scale_color_manual( # Customize colors
    values = c("female" = "blue", "male" = "red", 'Unknown'='yellow', 'none'='green')
  )
```

Differentiate points using `shape` or `size`.

```r
starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  replace_na(list(sex = 'Unknown')) %>%
  ggplot(
    aes(
      x = mass, y = height,
      shape = sex,  # different symbols for different sexs
      size = mass # larger mass values create bigger points (called bubble plot sometimes)
    )
  ) +
  geom_point() +
  labs(
    title = "Height vs. Mass",
    x = "Mass (kg)", y = "Height (cm)", shape = "sex", size = 'Mass'
  ) +
  theme_grey()
```

Create subplots by a categorical variable.

```r
starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  replace_na(list(sex = 'Unknown')) %>%
  ggplot(aes(x = mass, y = height, color = sex)) +
  geom_point() +
  labs(
    title = "Height vs. Mass by sex",
    x = "Mass (kg)", y = "Height (cm)", size = 'Mass'
  ) +
  theme_minimal() +
  facet_wrap(~gender) +
  geom_smooth(method = lm, se= T)
```

Jitter plot

A jitter plot reduces overplotting in scatter plots when points overlap. - Overplotting occurs when too many data points overlap, making it difficult to distinguish individual values in a plot. This happens especially in scatter plots when multiple points share the same or very similar coordinates. For example, age is measured in years, and height is given in centimeters. If you construct a scatter plot of weight versus age for a sufficiently large sample of people, there might be many people recorded as, say, 29 years and 70 kg, and therefore many markers plotted at the point (29, 70).

```r
starwars %>%
  filter(!is.na(height) & !is.na(mass) & mass < 1000) %>%
  ggplot(
    aes(x = mass, y = height, color = sex)
  ) +
  geom_point() +
  geom_jitter(
    width= .2, # each point is randomly moved up to 0.2 units left or right.
    height= .2, # each point is randomly moved up to 0.2 units up or down.
    alpha = .2, # the transparency
    color = "purple"  # color for jittered points
  ) +
  labs(
    title = "Height vs. Mass",
    x = "Mass (kg)", y = "Height (cm)"
  ) +
  theme_bw()
```

Histogram

Count (number of observations per bin) as the y-axis.

```r
starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
  geom_histogram(bins = 10, fill = "steelblue", color = "black") +
  labs(title = "Distribution of Character Heights",
       x = "Height (cm)", y = "Count") +
  theme_minimal()
```

Relative frequency (proportion of total observations per bin) as the y-axis. But the relative frequency is not a variable in the original dataset; we need to first calculate it. We use the `after_stat()` function to do so.

```r
starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
    geom_histogram(
      aes(y = after_stat(count / sum(count))),
      bins = 10, fill = "steelblue", color = "black"
    ) +
    labs(title = "Distribution of Character Heights",
         x = "Height (cm)", y = "Frequency") +
    theme_minimal()
```

Add a scaled fitted curve.

```r
binwidth_value = 20
starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
    geom_histogram(
      aes(y = after_stat(count / sum(count))),
      binwidth = binwidth_value, fill = "steelblue", color = "black"
    ) +
    labs(title = "Distribution of Character Heights",
         x = "Height (cm)", y = "Frequency") +
    theme_minimal()+
    geom_density(aes(y = after_stat(density)*binwidth_value), color = "red", linewidth = 1.2)
```

Density (=relative frequency divided by binwidth) as the y-axis with the fitted density curve.

```r
starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
    geom_histogram(
      aes(y = after_stat(density)),
      bins = 20, fill = "steelblue", color = "black"
    ) +
    labs(title = "Distribution of Character Heights",
         x = "Height (cm)", y = "Density") +
    theme_minimal()+
    geom_density(aes(y = after_stat(density)), color = "red", linewidth = 1.2)
```

Histograms of height by sex with different colors.

```r
starwars %>%
  filter(!is.na(height) & !is.na(sex)) %>%
  ggplot(aes(x = height, fill = sex)) +
  geom_histogram(
    bins=10,
    position = "dodge",  # identity: overlayed; dodge: side-by-side
    alpha = 0.2
  ) +
  # scale_fill_manual(values = c("blue", "red", 'yellow', 'green')) +
  labs(title = "Overlayed Histograms of Heights by Color",
       x = "Height (cm)", y = "Count", fill = "Sex") +
  theme_minimal()
```

Histograms of height by sex with different textures.

```r
library(ggpattern)
starwars %>%
  filter(!is.na(height) & !is.na(sex)) %>%
  ggplot(aes(x = height, pattern = sex)) + # Use 'pattern' instead of 'fill'
  geom_histogram_pattern( # Use 'geom_histogram_pattern' instead of 'geom_histogram'
    bins=10,
    position = "identity", alpha = 0.2,
    pattern_fill = "black",  # Color of the pattern lines
    pattern_angle = 30,      # Optional: Adjust pattern angle
    pattern_density = 0.1    # Optional: Adjust pattern thickness/spacing
  ) +
  labs(title = "Overlayed Histograms of Heights by Color",
      x = "Height (cm)", y = "Count", fill = "Sex") +
  theme_minimal()
```

Frequency polygon

Use lines to connect frequency points.

```r
starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
  geom_histogram(bins = 10, fill = "steelblue", color = "black") +
  geom_freqpoly(bins = 10, color = "orange", size = 1) +
  labs(title = "Distribution of Character Heights",
       x = "Height (cm)", y = "Count") +
  theme_minimal()
```

Change the scale of y-axis.

```r
starwars %>%
  filter(!is.na(height)) %>%
  ggplot(aes(x = height)) +
  geom_freqpoly(
    # aes(y = after_stat(density)), # density as the y-axis
    aes(y = after_stat(count / sum(count))), # relative frequency as the y-axis
    bins = 10, color = "orange", size = 1) +
  labs(title = "Distribution of Character Heights",
       x = "Height (cm)", y = "Relative Frequency") +
  theme_minimal()
```

Particularly useful for comparing multiple groups by overlaying multiple frequency polygons.

```r
starwars %>%
  filter(!is.na(height) & !is.na(sex)) %>%
  ggplot(aes(x = height, color = sex)) +
  geom_freqpoly(bins = 10, size = 1) +
  labs(title = "Distribution of Character Heights",
       x = "Height (cm)", y = "Count") +
  theme_minimal()
```

Bar plot

Visualize categorical variables by showing counts (by default)

```r
starwars %>%
  ggplot(aes(x = species)) +
  geom_bar(fill = "steelblue", color = "black") +
  labs(title = "Character Count by Species",
       x = "Species", y = "Count") +
  theme_minimal() +
  # coord_flip()  # Flip for better readability
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) # alternatively change the angle of text
```

Visualize categorical data by showing proportions - Include `aes(y = after_stat(count / sum(count)))` in `geom_bar()`

```r
starwars %>%
  ggplot(aes(x = species)) +
  geom_bar(
    aes(y = after_stat(count / sum(count))),
    fill = "steelblue", color = "black") +
  labs(title = "Character Count by Species",
       x = "Species", y = "Proportion") +
  theme_minimal() +
  coord_flip()
```

Stacked bar chart

```r
starwars %>%
  ggplot(aes(
    x = fct_rev(species), # Reverses the order of species to correct flipping
    fill = sex
  )) +
  geom_bar(position = "stack") +  # position = "fill" or "stack"
  labs(title = "Stacked Bar Chart",
       x = "sex", y = "Count") +
  theme_minimal() +
  coord_flip()
```

Polar chart

A polar chart is simply a bar chart with a single category that is transformed using `coord_polar(theta = "y")`.

```r
ggplot(starwars, aes(x = '', fill = sex)) +
  geom_bar(width = 1) +
  coord_polar(theta = "y") +
  labs(title = "Numbers of Characters by sex", x='', y='') +
  theme_minimal()
```

Change the labels to proportions instead of counts.

```r
starwars %>%
  count(sex) %>%
  mutate(prop = n / sum(n)) %>%
  ggplot(aes(x = "", y = prop, fill = sex)) +
  geom_bar(stat = "identity", width = 1) + # stat = "count" (by default), "identity" (precomputed value
  coord_polar(theta = "y") +
  labs(title = "Proportion of Characters by sex", x='', y='') +
  theme_minimal()
```

Box plot

Showing summary statistics and illustrating the distribution of a variable.

Component	Description
Thick horizontal segment	Sample median.
Quartiles (Box edges)	25% quantile (Q1) and 75% quantile (Q3).
Whiskers (Thin vertical segments)	Extend up to 1.5x(Q3-Q1).
Points outside the whiskers	Outliers.

```r
starwars %>%
  filter(!is.na(height)) %>%
  replace_na(list(sex = 'Unknown')) %>%
  ggplot(aes(x = sex, y = height, fill = sex)) + # "fill=sex": paint boxes by sex
  geom_boxplot() +
  labs(title = "Height Distribution by sex",
       x = "sex", y = "Height (cm)") +
  theme_minimal()
```

Density plot

Shows distributions of continuous variables

```r
starwars %>%
  filter(!is.na(mass) & !is.na(sex) & mass<1000) %>%
  ggplot(aes(x = mass, fill = sex)) +
  geom_density(alpha = 0.6) + # alpha controls the transparency
  labs(title = "Density Plot of Mass by sex",
       x = "Mass (kg)",
       y = "Density") +
  theme_minimal()
```

Violin plot with a box plot inside

A combination of a box plot and a density plot.

Component	Description
Violin shape	Density curve.
Thick horizontal segment	Sample median.
Quartiles (Box edges)	25% quantile (Q1) and 75% quantile (Q3).
Whiskers (Thin vertical segments)	Extend up to 1.5x(Q3-Q1).
Points outside the whiskers	Outliers.

```r
starwars %>%
  ggplot(aes(x = sex, y = height, fill = sex)) +
  geom_violin(alpha = 0.5) +
  geom_boxplot(width = 0.1) +
  labs(title = "Violin Plot with Box Plot",
       x = "sex", y = "Height (cm)") +
  theme_minimal()
```

Contour plot

Visualize a 3D surface in 2D by drawing lines of equal values.

```r
x <- seq(-pi, pi, length = 50)
y <- x
z_matrix <- outer(x, y, function(x, y) x^2 + y^2)
df <- expand.grid(x = x, y = y) %>% mutate(z = as.vector(z_matrix))

ggplot(df, aes(x = x, y = y, z = z)) +
  geom_contour(
    bins = 10, # number of contour levels
    color = "blue"
  ) +
  metR::geom_text_contour(skip = 0, stroke = 0.2) + # Add contour labels
  labs(title = "Function Visualization: f(x, y) = x^2 + y^2",
       x = "X-axis", y = "Y-axis") +
  theme_minimal()
```

Could be more illustrative by filling the regions between adjacent lines.

```r
ggplot(df, aes(x = x, y = y, z = z)) +
  geom_contour_filled(bins = 10) +
  scale_fill_viridis_d(direction = -1, option = "H") +
  labs(title = "Function Visualization: f(x, y) = x^2 + y^2",
       x = "X-axis", y = "Y-axis", fill = "Z value") +
  theme_minimal()
```

Interactive 2D contour plot with `plotly`

```r
p_contour = ggplot(df, aes(x = x, y = y, z = z)) +
  geom_contour(
    bins = 10, # number of contour levels
    color = "blue"
  ) +
  labs(title = "Function Visualization: f(x, y) = x^2 + y^2",
       x = "X-axis", y = "Y-axis") +
  theme_minimal()
plotly::ggplotly(p_contour)
```

Interactive 3D plot with `plotly`

```r
library(plotly)
plot_ly(x = ~x, y = ~y, z = ~z_matrix , type = "surface", colorscale = "Viridis") %>%
  layout(
    title = "3D Surface Plot: f(x, y) = x^2 + y^2",
    scene = list(
      xaxis = list(title = "X-axis"),
      yaxis = list(title = "Y-axis"),
      zaxis = list(title = "Z-axis")
    )
  )
```

Heatmap

Represent the magnitude of values as colors.

```r
x <- seq(-pi, pi, length = 50)
y <- x
z_matrix <- outer(x, y, function(x, y) x^2 + y^2)
df <- expand.grid(x = x, y = y) %>% mutate(z = as.vector(z_matrix))
df %>%
  ggplot(aes(x = x, y = y, fill = z)) +
  geom_tile() +
  scale_fill_gradient(low="yellow", high="red") +
  labs(title = "Function Visualization: f(x, y) = x^2 + y^2",
       x = "X-axis", y = "Y-axis", fill = "f(x,y)") +
  theme_minimal()
```

Commonly used to visualize correlation matrices, reflecting the covariation between two continous variables.

```r
# Convert the correlation matrix to long format
cor_df <- as.data.frame(cor(mtcars)) %>%
  rownames_to_column(var = "Var1") %>%
  pivot_longer(cols = -Var1, names_to = "Var2", values_to = "Correlation")
# Convert values in Var2 to a factor with reversed alphabetical order
cor_df <- cor_df %>%
  mutate(Var2 = factor(Var2, levels = rev(sort(unique(Var2)))))
# plotting
ggplot(cor_df, aes(Var1, Var2, fill = Correlation)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  labs(title = "Heatmap of mtcars Correlation Matrix",
       x = "Variable", y = "Variable") +
  theme_minimal()
```

Useful too when showing the relationship among three variables (e.g., flight delay by month and origin airport).

```r
nycflights13::flights %>%
  group_by(month, origin) %>%
  summarise(
    avg_dep_delay = mean(dep_delay, na.rm = T) # Summarize average departure delay by month and origin
  ) %>%
  ggplot(aes(x = factor(month), y = origin, fill = avg_dep_delay)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  labs(title = "Average Departure Delay by Month and Origin",
       x = "Month", y = "Origin Airport", fill = "Avg Delay (mins)") +
  theme_minimal()
```

Count plot

Visualize the number of observations for each combination of levels of two categorical variables, reflecting the relationship between categorical variables.

```r
diamonds |> # A dataset containing the prices and other attributes of almost 54,000 diamonds
  ggplot(aes(x = cut, y = color)) +
  geom_count(color = "white") +
  labs(x = "Cut",
       y = "Color",
       size = "Count") +
  theme_grey()
```

Line plot

Showing relationships between two continuous variables.

```r
economics %>%
  ggplot(aes(x = date, y = unemploy)) +
  geom_line(color = "blue", linewidth = 1) +
  labs(title = "U.S. Unemployment Over Time",
       x = "Time", y = "Number of unemployed in thousands") +
  theme_minimal()
```

Include more components: data points and auxiliary lines.

```
economics %>%
  ggplot(aes(x = date, y = unemploy)) +
  geom_line(color = "blue", linewidth = 1) +
  labs(title = "U.S. Unemployment Over Time",
       x = "Time", y = "Number of unemployed in thousands") +
  theme_minimal() +
  geom_point(color = "red") +
  geom_abline(aes(intercept = 0, slope=1), color='green', lty=2)+
  geom_vline(aes(xintercept = 1990), color='green', lty=2)+
  geom_hline(aes(yintercept = 8000), color='green', lty=2)
```

Two-line plot.

```
economics %>%
  ggplot(aes(x = date)) +
  geom_line(aes(y = unemploy, color = "Unemployment"), linewidth = 1) +
  geom_line(aes(y = pop, color = "total population"), linewidth = 1) +
  scale_color_manual(values = c("Unemployment" = "blue", "total population" = "red")) +
  labs(title = "Total Population and Unemployment Over Time",
       x = "Time", y = "Number in thousands", color = "Variable") +
  theme_minimal()
```

Two trajectories on different scales with a second y-axis added.

```
ggplot2::economics %>%
  ggplot(aes(x = date)) +
  geom_line(aes(y = unemploy, color = "Unemployment"), linewidth = 1) +
  geom_line(aes(y = psavert*1000, color = "Personal Savings Rate"), linewidth = 1) +
  scale_color_manual(values = c("Unemployment" = "blue", "Personal Savings Rate" = "red")) +
  # scale_y_continuous(sec.axis = sec_axis(~., name = "Test Axis"))
  scale_y_continuous(sec.axis = sec_axis(~./1000, name = "Savings Rate (%)")) +
  labs(title = "U.S. Unemployment and Savings Rate Over Time",
       x = "Year", y = "Unemployed People (in Thousands)", color = "Variable") +
  theme_minimal()
```

Saving figures

```
p = economics %>%
  ggplot(aes(x = date)) +
  geom_line(aes(y = unemploy, color = "Unemployment"), linewidth = 1) +
  geom_line(aes(y = psavert*1000, color = "Personal Savings Rate"), linewidth = 1) +
  scale_color_manual(values = c("Unemployment" = "blue", "Personal Savings Rate" = "red")) +
  scale_y_continuous(sec.axis = sec_axis(~./1000, name = "Savings Rate (%)")) +
  labs(title = "U.S. Unemployment and Savings Rate Over Time",
       x = "Year", y = "Unemployed People (in Thousands)", color = "Variable") +
  theme_grey()
p
ggsave(
  filename="c:/first_ggplot.tiff",
  plot = p, # Save the last plot if this `plot` option is unspecified
  width = 7.6, height = 1.2,
  unit = 'in', # 'in', 'cm', 'mm'
  dpi = 600, # DPI (dots per inch)
)
```

```r
# Specify pixel dimensions
# E.g., per the author guidelines of Water Research,
# a minimum of 531 x 1328 pixels (h x w) or proportionally more
# dpi = pixels/inch => inch = pixels/dpi
ggsave(
  filename="second_ggplot.tiff",
  plot = p, # Save the last plot if this `plot` option is unspecified
  width = 13280/600,
  height = 5310/600,
  unit = 'in', # 'in', 'cm', 'mm'
  dpi = 600, # DPI (dots per inch)
)
```