

Types, vectors, and functions in R

2025-08-21

Vectors and Types

Vectors

`c(1, 3, 5)`

`c(TRUE, FALSE, TRUE, TRUE)`

`c("red", "blue")`

Vectors

Vectors have 1 dimension

Vectors have a length

```
1 length(c("blue", "red"))
```

```
[1] 2
```

Some vectors have names.

```
1 names(c("x" = 1, "y" = 1))
```

```
[1] "x" "y"
```

Vectors have types and classes

```
1 class(c("blue", "red"))
```

```
[1] "character"
```

Types

Numeric/double (c(1.0, 2.0, 3.0))

Integer (c(1L, 2L, 3L))

Character (c("a", "b", "c"))

Factor (factor(c("a", "b", "c")))

Logical (TRUE)

Dates and times

Packages to work with types and classes

Strings/character: stringr

Factors: forcats

Dates: lubridate

Making vectors

```
1 1:3
```

```
[1] 1 2 3
```

```
1 c(1, 2, 3)
```

```
[1] 1 2 3
```

```
1 rep(1, 3)
```

```
[1] 1 1 1
```

```
1 seq(from = 1, to = 3, by = .5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0
```

Your Turn 1

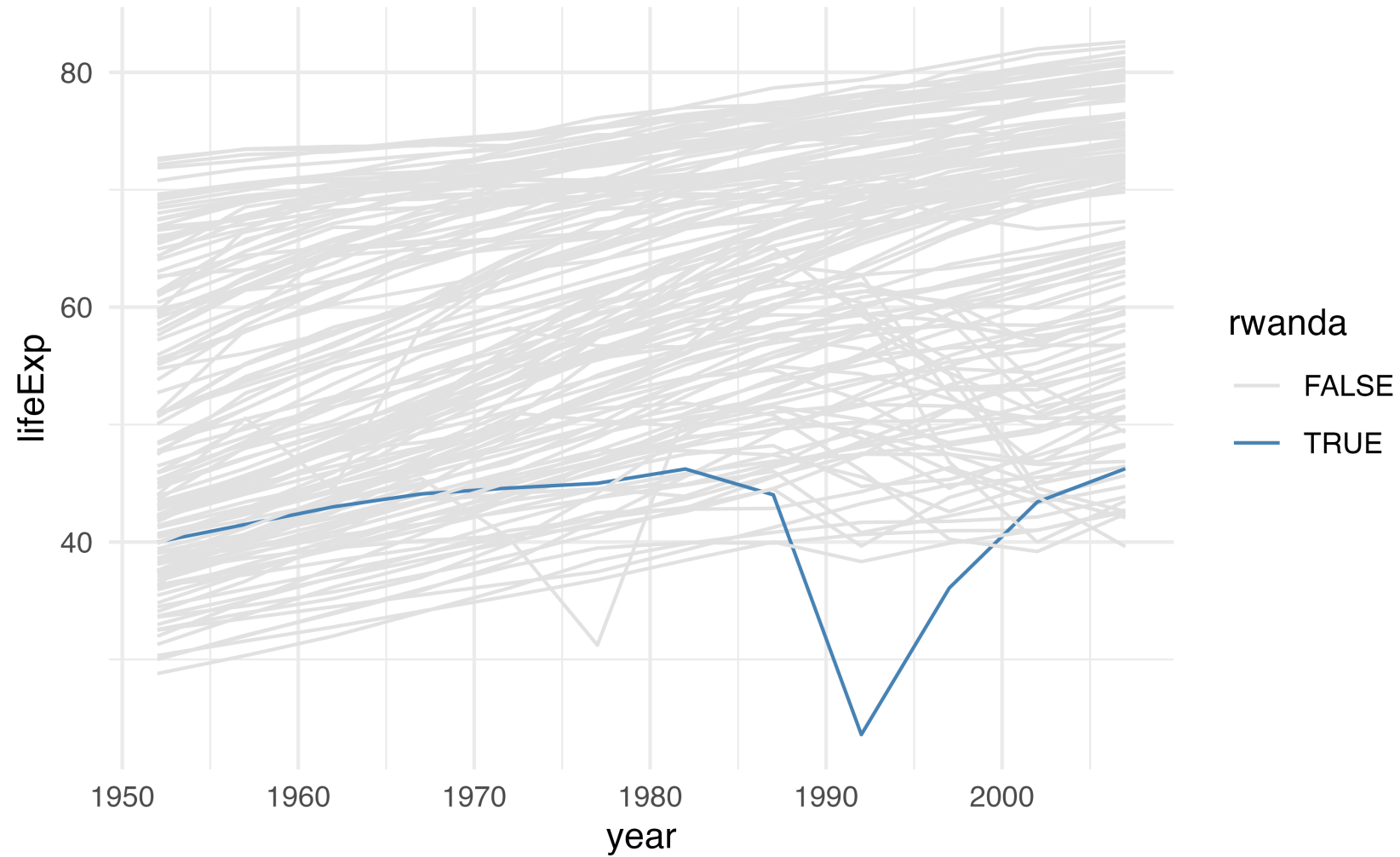
Create a character vector of colors using `c()`. Use the colors `"grey90"` and `"steelblue"`. Assign the vector to a name.

Use the vector you just created to change the colors in the plot below using `scale_color_manual()`. Pass it using the `values` argument.

Your Turn 1

```
1 cols <- c("grey90", "steelblue")
2
3 gapminder |>
4   mutate(rwanda = country == "Rwanda") |>
5   ggplot(aes(year, lifeExp, color = rwanda, group = country)) +
6   geom_line() +
7   scale_color_manual(values = cols) +
8   theme_minimal()
```

Your Turn 1



Working with vectors

Subset vectors with `[]` or `[[]]`

```
1 x <- c(1, 5, 7)
2 x[2]
```

```
[1] 5
```

```
1 x[[2]]
```

```
[1] 5
```

```
1 x[c(FALSE, TRUE, FALSE)]
```

```
[1] 5
```

```
1 x[[c(FALSE, TRUE, FALSE)]]
```

Error in `x[[c(FALSE, TRUE, FALSE)]]`: attempt to select more than one element in vectorIndex

Working with vectors

Modify elements

```
1 x
```

```
[1] 1 5 7
```

Working with vectors

Modify elements

```
1 x[[2]] <- 100  
2 x
```

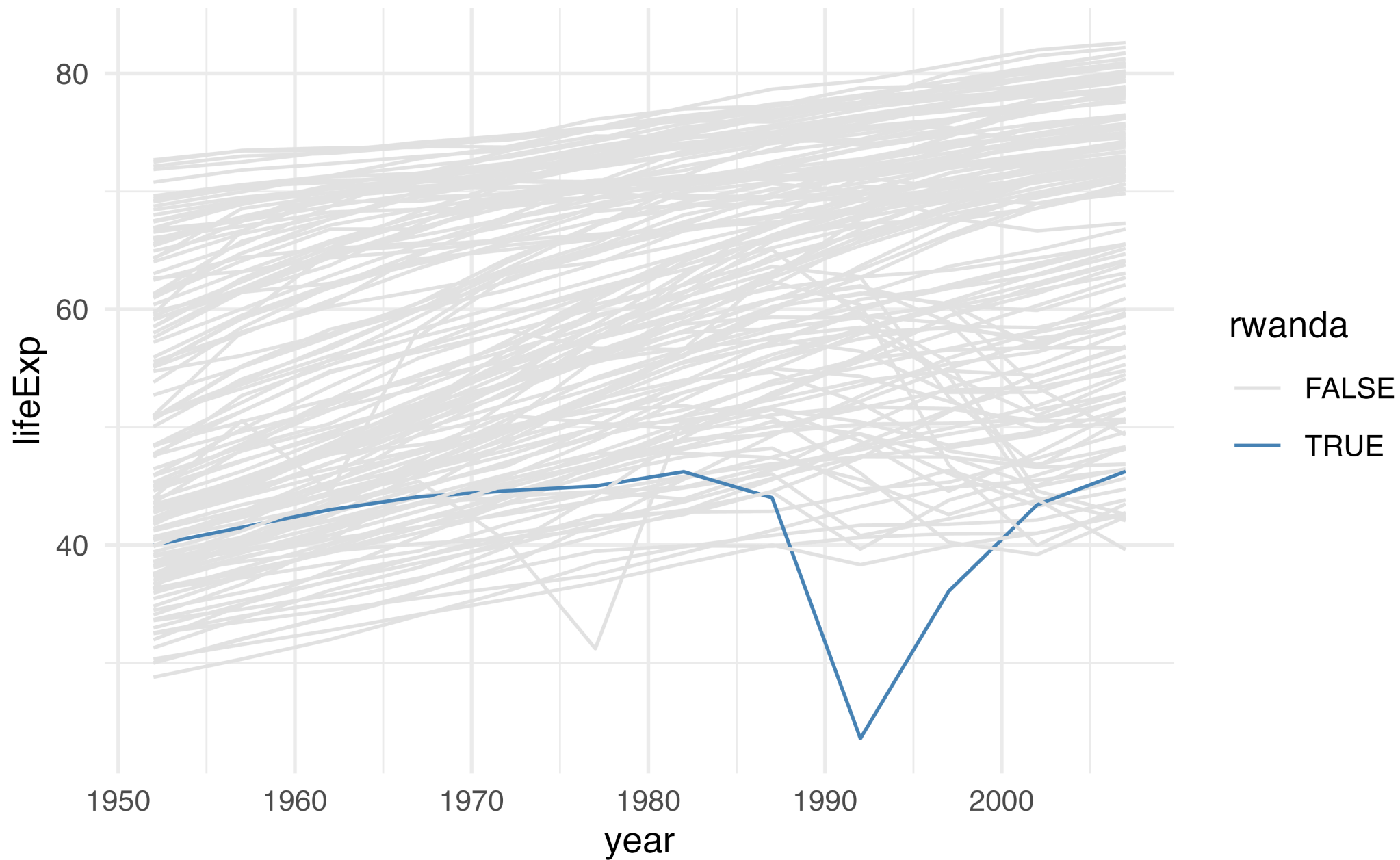
```
[1] 1 100 7
```

Working with vectors

Modify elements

```
1 x[x > 10] <- NA  
2 x
```

```
[1] 1 NA 7
```

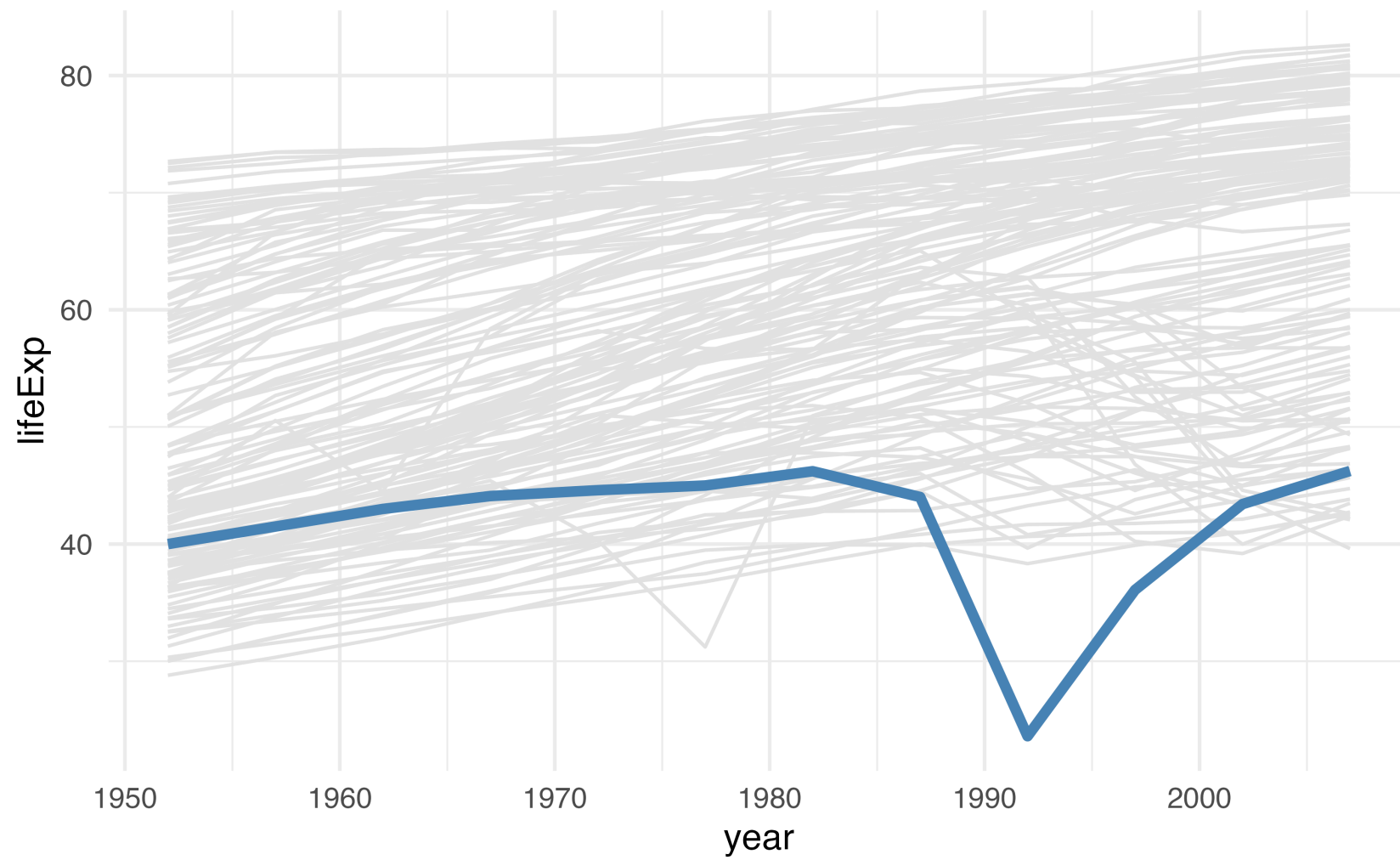


```
1 cols <- c("grey90", "steelblue")
2
3 gapminder |>
4   mutate(rwanda = ifelse(country == "Rwanda", TRUE, FALSE)) |>
5   ggplot(aes(year, lifeExp, color = rwanda, group = country)) +
6   geom_line() +
7   scale_color_manual(values = cols) +
8   theme_minimal()
```



```
1 cols <- c("grey90", "steelblue")
2
3 gapminder |>
4   mutate(rwanda = ifelse(country == "Rwanda", TRUE, FALSE)) |>
5   ggplot(aes(year, lifeExp, group = country)) +
6   geom_line(
7     data = function(x) filter(x, !rwanda),
8     color = cols[[1]]
9   ) +
10  theme_minimal()
```

```
1 cols <- c("grey90", "steelblue")
2
3 gapminder |>
4   mutate(rwanda = ifelse(country == "Rwanda", TRUE, FALSE)) |>
5   ggplot(aes(year, lifeExp, color = rwanda, group = country)) +
6   geom_line(
7     data = function(x) filter(x, !rwanda),
8     color = cols[[1]]
9   ) +
10  geom_line(
11    data = function(x) filter(x, rwanda),
12    color = cols[[2]],
13    linewidth = 1.5
14  ) +
15  theme_minimal()
```



Your Turn 2

Create a numeric vector that has the following values: 3, 5, NA, 2, and NA.

Try using `sum()`. Then add `na.rm = TRUE`.

Check which values are missing with `is.na()`; save the results to a new object and take a look

Change all missing values of `x` to -9999.

Try `sum()` again without `na.rm = TRUE`.

Your Turn 2

```
1 x <- c(3, 5, NA, 2, NA)
2 sum(x)
```

```
[1] NA
```

Your Turn 2

```
1 sum(x, na.rm = TRUE)
```

```
[1] 10
```

Your Turn 2

```
1 x_missing <- is.na(x)
2 x_missing
```

```
[1] FALSE FALSE TRUE FALSE TRUE
```

```
1 x[x_missing] <- -9999
2 x
```

```
[1]      3      5 -9999      2 -9999
```

Writing Functions

Functions that return vectors

```
1 gdpPercap <- gapminder |>  
2   filter(year == 1952, continent == "Americas") |>  
3   pull(gdpPercap)  
4  
5 sum(gdpPercap)
```

```
[1] 101976.6
```

Functions that return data frames

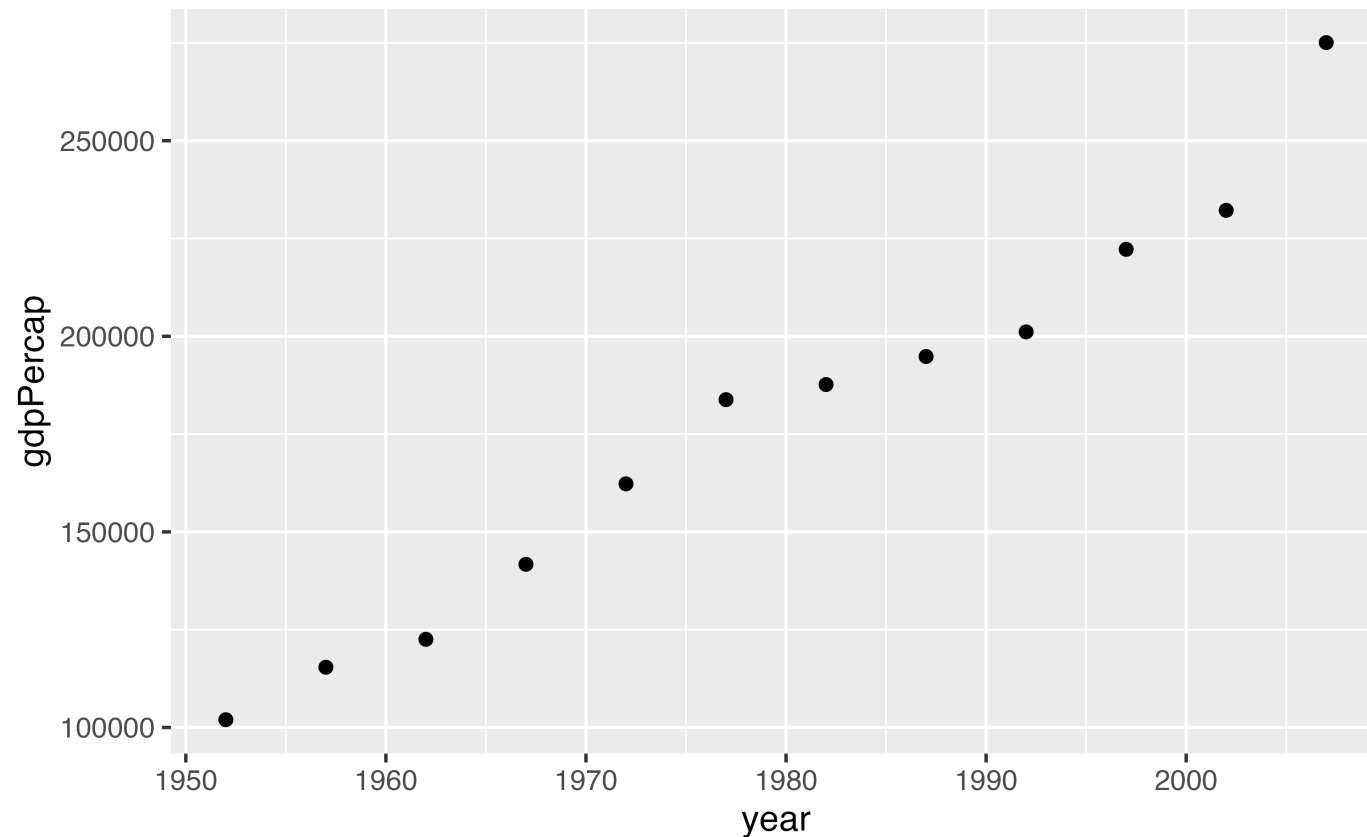
```
1 gapminder |>
2   group_by(year) |>
3   filter(continent == "Americas") |>
4   summarize(gdpPerCap = sum(gdpPerCap))
```

A tibble: 12 × 2

	year	gdpPerCap
	<int>	<dbl>
1	1952	101977.
2	1957	115401.
3	1962	122539.
4	1967	141706.
5	1972	162283.
6	1977	183800.
7	1982	187668.
8	1987	194835.
9	1992	201123.
10	1997	222233.
11	2002	222100.

Functions that make plots

```
1 gapminder |>  
2   group_by(year) |>  
3   filter(continent == "Americas") |>  
4   summarize(gdpPercap = sum(gdpPercap)) |>  
5   ggplot(aes(year, gdpPercap)) +  
6   geom_point()
```



Why write functions?

To make repetitive code reusable

To make complex code understandable

To make useful code shareable

Writing functions

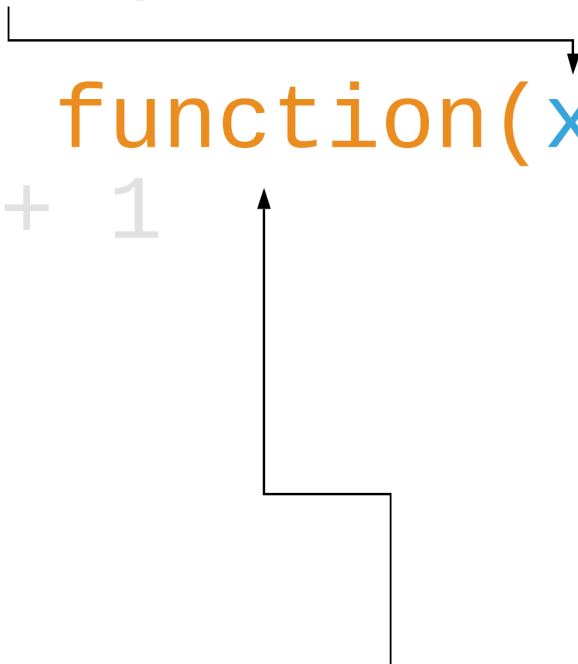
```
add_one <- function(x) {  
  x <- x + 1  
  x  
}
```

```
add_one(1)  
#> 2
```

Writing functions

Function arguments

```
add_one <- function(x) {  
  x <- x + 1  
  x  
}
```

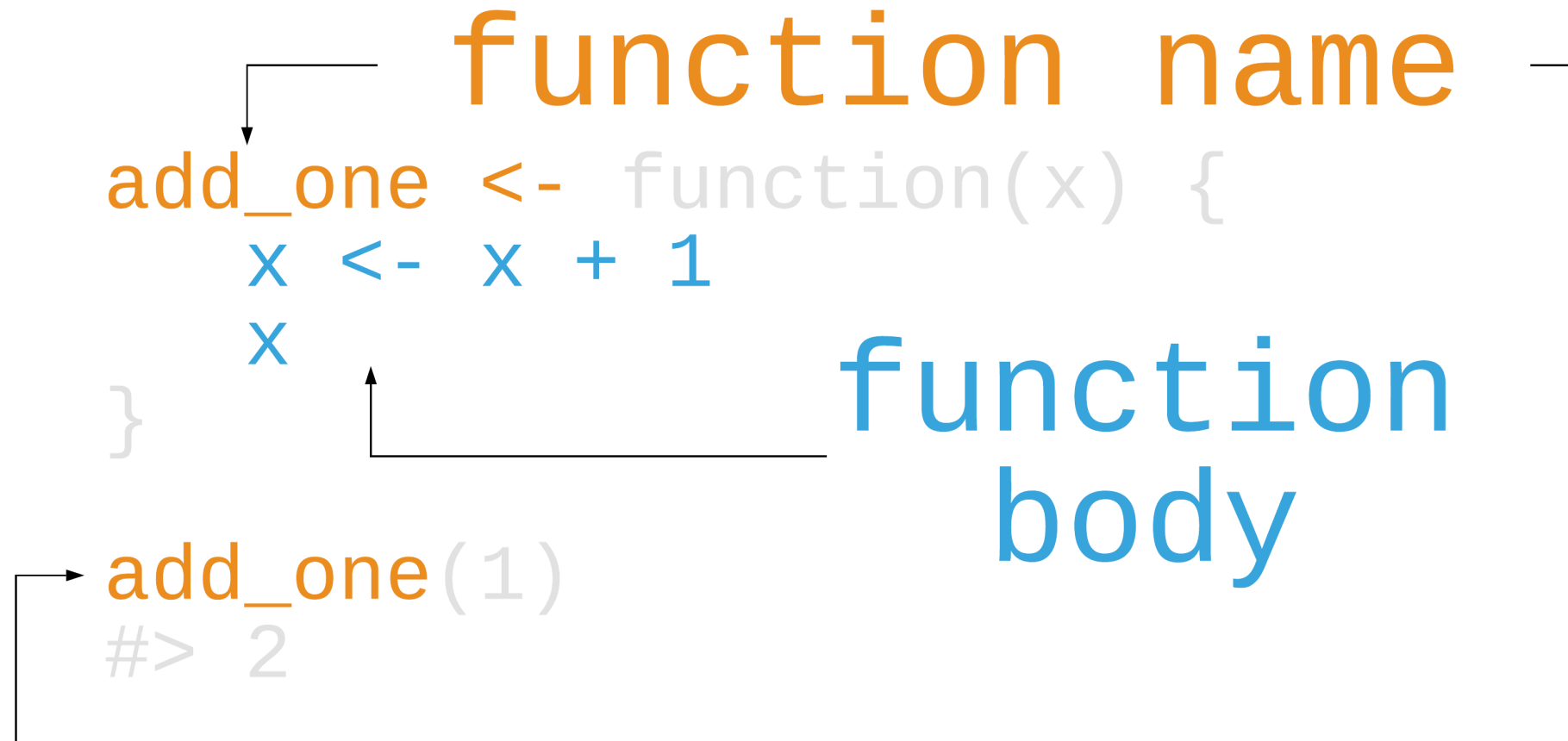


A diagram consisting of two arrows. The first arrow starts at the 'x' parameter in the function definition 'function(x)' and points to the '1' argument in the function call 'add_one(1)'. The second arrow starts at the '1' argument in the function call and points to the 'x' parameter in the function definition.

```
add_one(1)  
#> 2
```

Create function

Writing functions



Writing functions

```
add_one <- function(x) {  
  x <- x + 1  
  x  
}  
add_one(1)  
#> 2
```

output

input

```
graph LR
    input[1] --> x_in[x]
    x_in --> output[2]
```


Your Turn 3

Create a function called `sim_data` that doesn't take any arguments.

In the function body, we'll return a `tibble`.

For `x`, have `rnorm()` return 50 random numbers.

For `sex`, use `rep()` to create 50 values of "male" and "female". Hint: You'll have to give `rep()` a character vector for the first argument. The `times` argument is how many times `rep()` should repeat the first argument, so make sure you account for that.

For `age` use the `sample()` function to sample 50 numbers from 25 to 50 with replacement.

Call `sim_data()`

Your Turn 3

```
1 sim_data <- function() {  
2   tibble(  
3     x = rnorm(50),  
4     sex = rep(c("male", "female"), times = 25),  
5     age = sample(25:50, size = 50, replace = TRUE)  
6   )  
7 }  
8  
9 sim_data()
```

Your Turn 3

```
# A tibble: 50 × 3
      x sex    age
  <dbl> <chr> <int>
1  0.751 male    36
2 -0.256 female  38
3 -0.344 male    29
4 -0.0498 female  35
5  0.409 male    46
6  0.376 female  48
7 -1.17  male    33
8 -0.211 female  43
9  0.0396 male    34
10 1.67  female  36
" # A tibble: 40 × 3" >
```

E-Values

The strength of unmeasured confounding required to explain away a value

Rate ratio: 3.9 = E-value: 7.3

Your Turn 4

Write a function to calculate an E-Value given an RR.

Call the function `evaluate` and give it an argument called `estimate`. In the body of the function, calculate the E-Value using `estimate + sqrt(estimate * (estimate - 1))`

Call `evaluate()` for a risk ratio of 3.9

Your Turn 4

```
1 evaluate <- function(estimate) {  
2   estimate + sqrt(estimate * (estimate - 1))  
3 }
```

Invoking the function with ()

```
1 evaluate(3.9)
```

```
[1] 7.263034
```

Control Flow

```
1 if (PREDICATE) {  
2     do_something()  
3 }
```

Control Flow

```
1 if (PREDICATE) {  
2   do_something()  
3 } else {  
4   do_something_for_everything_else()  
5 }
```


Control Flow

```
1 if (PREDICATE) {  
2   do_something()  
3 } else if (ANOTHER_PREDICATE) {  
4   do_something_else()  
5 } else {  
6   do_something_for_everything_else()  
7 }
```

Make a prediction

What will **y** be?

```
1 x <- 99
2 if (x < 10) {
3   y <- "a"
4 } else if (is.na(x)) {
5   y <- NA
6 } else {
7   y <- "z"
8 }
9
10 y
```

```
[1] "z"
```

If statements are for a single **TRUE** or **FALSE**

```
1 x <- rnorm(100)
2 if (x > 0) {
3   x + 1
4 }
```

Error in if (x > 0) {: the condition has length > 1

Conditional values with vectors

```
1 ifelse(PREDICATE_VECTOR, true_result, false_result)
2 dplyr::case_when(
3   PREDICATE_VECTOR ~ true_result,
4   PREDICATE_VECTOR ~ next_true_result,
5   .default = default_result
6 )
7 # also see `switch()`
```

Conditional values with vectors

```
1 x <- -2:2  
2 ifelse(x >= 0, "positive", "negative")
```

```
[1] "negative" "negative" "positive" "positive" "positive"
```

If-else with vectors

```
1 case_when(  
2   x > 0 ~ "positive",  
3   x == 0 ~ "zero",  
4   .default = "negative"  
5 )
```

```
[1] "negative" "negative" "zero"      "positive" "positive"
```

Validation and stopping

```
if (is.numeric(x)) stop(), warn()
```

```
1 function(x) {  
2   if (is.numeric(x)) stop("x must be a character")  
3   # do something with a character  
4 }
```

Your Turn 5

Use `if ()` together with `is.numeric()` to make sure `estimate` is a number. Remember to use `!` for not.

If the estimate is less than 1, set `estimate` to be equal to `1 / estimate`.

Call `evaluate()` for a risk ratio of 3.9. Then try 0.80. Then try a character value.

Your Turn 5

```
1 evaluate <- function(estimate) {  
2   if (!is.numeric(estimate)) stop("`estimate` must be numeric")  
3   if (estimate < 1) estimate <- 1 / estimate  
4   estimate + sqrt(estimate * (estimate - 1))  
5 }
```

Your Turn 5

```
1 evaluate(3.9)
```

```
[1] 7.263034
```

```
1 evaluate(.80)
```

```
[1] 1.809017
```

```
1 evaluate("3.9")
```

```
Error in evaluate("3.9"): `estimate` must be numeric
```

Your Turn 6

Add a new argument called `type`. Set the default value to `"rr"`

Check if `type` is equal to `"or"`. If it is, set the value of `estimate` to be `sqrt(estimate)`

Call `evaluate()` for a risk ratio of 3.9. Then try it again with `type = "or"`.

Your Turn 6

```
1 evaluate <- function(estimate, type = "rr") {  
2   if (!is.numeric(estimate)) stop("`estimate` must be numeric")  
3   if (type == "or") estimate <- sqrt(estimate)  
4   if (estimate < 1) estimate <- 1 / estimate  
5   estimate + sqrt(estimate * (estimate - 1))  
6 }
```

Your Turn 6

```
1 evaluate(3.9)
```

```
[1] 7.263034
```

```
1 evaluate(3.9, type = "or")
```

```
[1] 3.362342
```

Your Turn 7: Challenge!

Create a new function called `transform_to_rr` with arguments `estimate` and `type`.

Use the same code above to check if `type == "or"` and transform if so. Add another line that checks if `type == "hr"`. If it does, transform the estimate using this formula: $(1 - 0.5^{\sqrt{\text{estimate}}}) / (1 - 0.5^{\sqrt{1 / \text{estimate}}})$.

Move the code that checks if `estimate < 1` to `transform_to_rr` (below the OR and HR transformations)

Return `estimate`

In `evaluate()`, change the default argument of `type` to be a character vector containing “rr”, “or”, and “hr”.

Get and validate the value of `type` using `match.arg()`. Follow the pattern `argument_name <- match.arg(argument_name)`

Transform `estimate` using `transform_to_rr()`. Don't forget to pass it both `estimate` and `type`!

Your Turn 7: Challenge!

```
1 transform_to_rr <- function(estimate, type) {
2   if (type == "or") estimate <- sqrt(estimate)
3   if (type == "hr") {
4     estimate <-
5       (1 - 0.5^sqrt(estimate)) / (1 - 0.5^sqrt(1 / estimate))
6   }
7   if (estimate < 1) estimate <- 1 / estimate
8
9   estimate
10 }
11
12 evaluate <- function(estimate, type = c("rr", "or", "hr")) {
13   # validate arguments
14   if (!is.numeric(estimate)) stop("`estimate` must be numeric")
15   type <- match.arg(type)
16
17   # calculate evaluate
18   estimate <- transform_to_rr(estimate, type)
19   estimate + sqrt(estimate * (estimate - 1))
20 }
```

Your Turn 7: Challenge!

```
1 evaluate(3.9)
```

```
[1] 7.263034
```

```
1 evaluate(3.9, type = "or")
```

```
[1] 3.362342
```

```
1 evaluate(3.9, type = "hr")
```

```
[1] 4.474815
```

```
1 evaluate(3.9, type = "rd")
```

Error in match.arg(type): 'arg' should be one of "rr", "or", "hr"

Programming with the tidyverse

Pass the dots: . . .

```
1 select_gapminder <- function(...) {  
2   gapminder |>  
3     select(...)  
4 }  
5  
6 select_gapminder(pop, year)
```

Pass the dots: . . .

```
# A tibble: 1,704 × 2
```

```
      pop  year  
    <int> <int>  
1  8425333 1952  
2  9240934 1957  
3 10267083 1962  
4 11537966 1967  
5 13079460 1972  
6 14880372 1977  
7 12881816 1982  
8 13867957 1987  
9 16317921 1992  
10 22227415 1997
```

```
" " " " " " " " " " " " " " " "
```

Your Turn 8

Use `...` to pass the arguments of your function, `filter_summarize()`, to `filter()`.

In `summarize`, get the `n` and mean life expectancy for the data set

Check `filter_summarize()` with `year == 1952`.

Try `filter_summarize()` again for 2002, but also filter countries that have the word " and " in the country name (e.g., it should detect "Trinidad and Tobago" but not "Iceland"). Use `str_detect()` from the `stringr` package.

Your Turn 8

```
1 filter_summarize <- function(...) {  
2   gapminder |>  
3     filter(...) |>  
4     summarize(n = n(), mean_lifeExp = mean(lifeExp))  
5 }
```

```
1 filter_summarize(year == 1952)
```

```
# A tibble: 1 × 2  
      n mean_lifeExp  
  <int>      <dbl>  
1    142        49.1
```

```
1 filter_summarize(year == 2002, str_detect(country, " and "))
```

```
# A tibble: 1 × 2  
      n mean_lifeExp  
  <int>      <dbl>  
1      4        69.9
```

Writing functions with dplyr, ggplot2, and friends

```
1 plot_hist <- function(x) {  
2   ggplot(gapminder, aes(x = x)) + geom_histogram()  
3 }  
4  
5 plot_hist(lifeExp)
```

```
Error in `geom_histogram()`:  
! Problem while computing aesthetics.  
i Error occurred in the 1st layer.  
Caused by error:  
! object 'lifeExp' not found
```

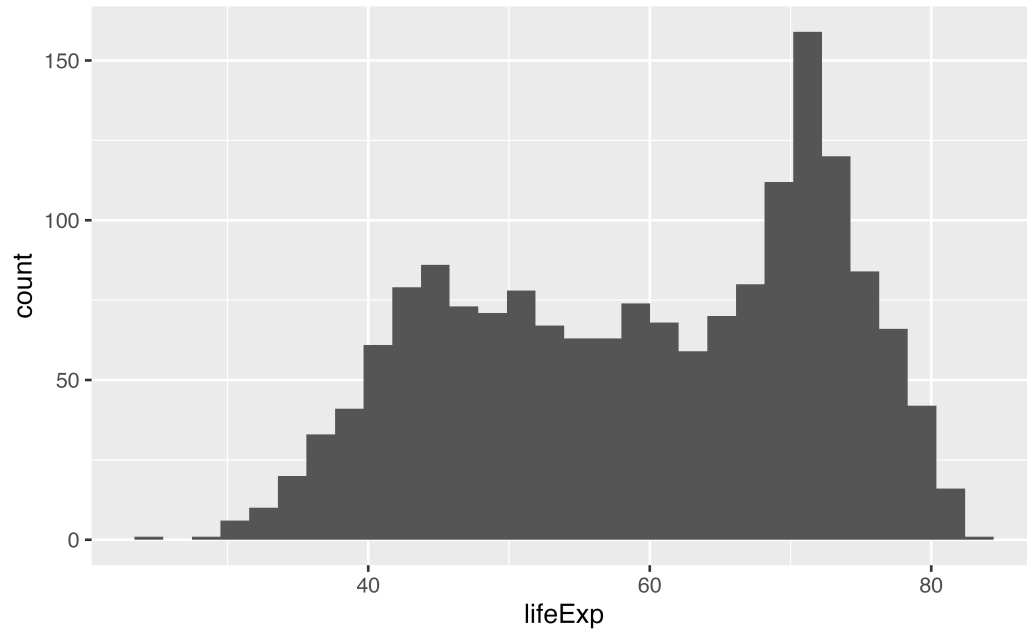
Programming with dplyr, ggplot2, and friends

```
1 plot_hist <- function(x) {  
2   ggplot(gapminder, aes(x = x)) + geom_histogram()  
3 }  
4  
5 plot_hist("lifeExp")
```

```
Error in `geom_histogram()`:  
! Problem while computing stat.  
i Error occurred in the 1st layer.  
Caused by error in `setup_params()`:  
! `stat_bin()` requires a continuous x aesthetic.  
✖ the x aesthetic is discrete.  
i Perhaps you want `stat="count"`?
```


Curly-curly: `{{ variable }}`

```
1 plot_hist <- function(x) {  
2   ggplot(gapminder, aes(x = {{ x }})) + geom_histogram()  
3 }  
4  
5 plot_hist(lifeExp)
```



Your turn 9

Filter `gapminder` by `year` using the value of `.year` (notice the period before hand!). You do NOT need curly-curly for this. (Why is that?)

Arrange it by the variable. This time, do wrap it in curly-curly!

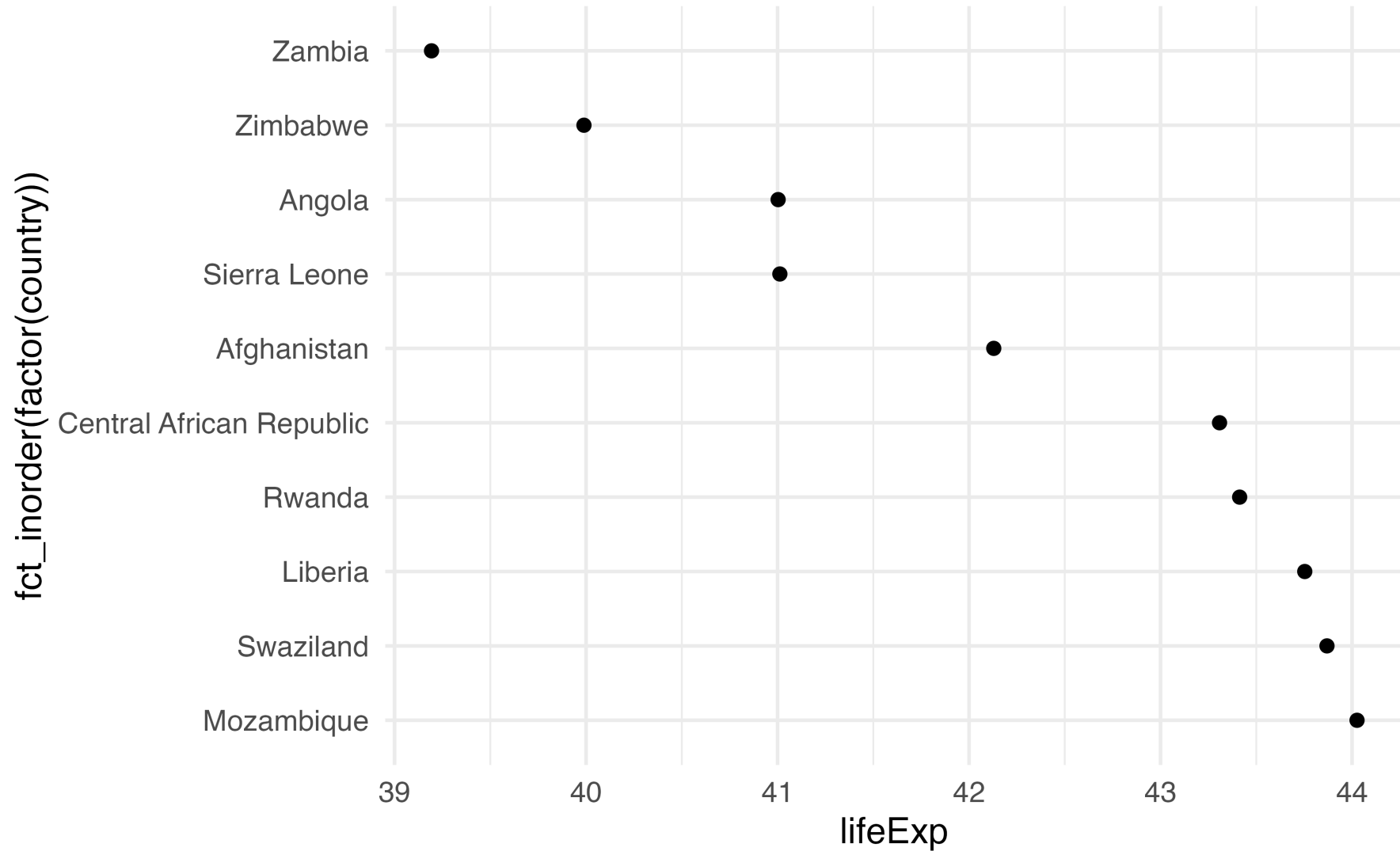
Make a scatter plot. Use `variable` for x. For y, we'll use `country`, but to keep it in the order we arranged it by, we'll turn it into a factor. Wrap the the `factor()` call with `fct_inorder()`. Check the help page if you want to know more about what this is doing.

Your turn 9

```
1 top_scatter_plot <- function(variable, .year) {  
2   gapminder |>  
3     filter(year == .year) |>  
4     arrange(desc({{ variable }})) |>  
5     # take the 10 lowest values  
6     tail(10) |>  
7     ggplot(aes(  
8       x = {{ variable }},  
9       y = fct_inorder(factor(country))  
10    )) +  
11    geom_point() +  
12    theme_minimal()  
13 }
```

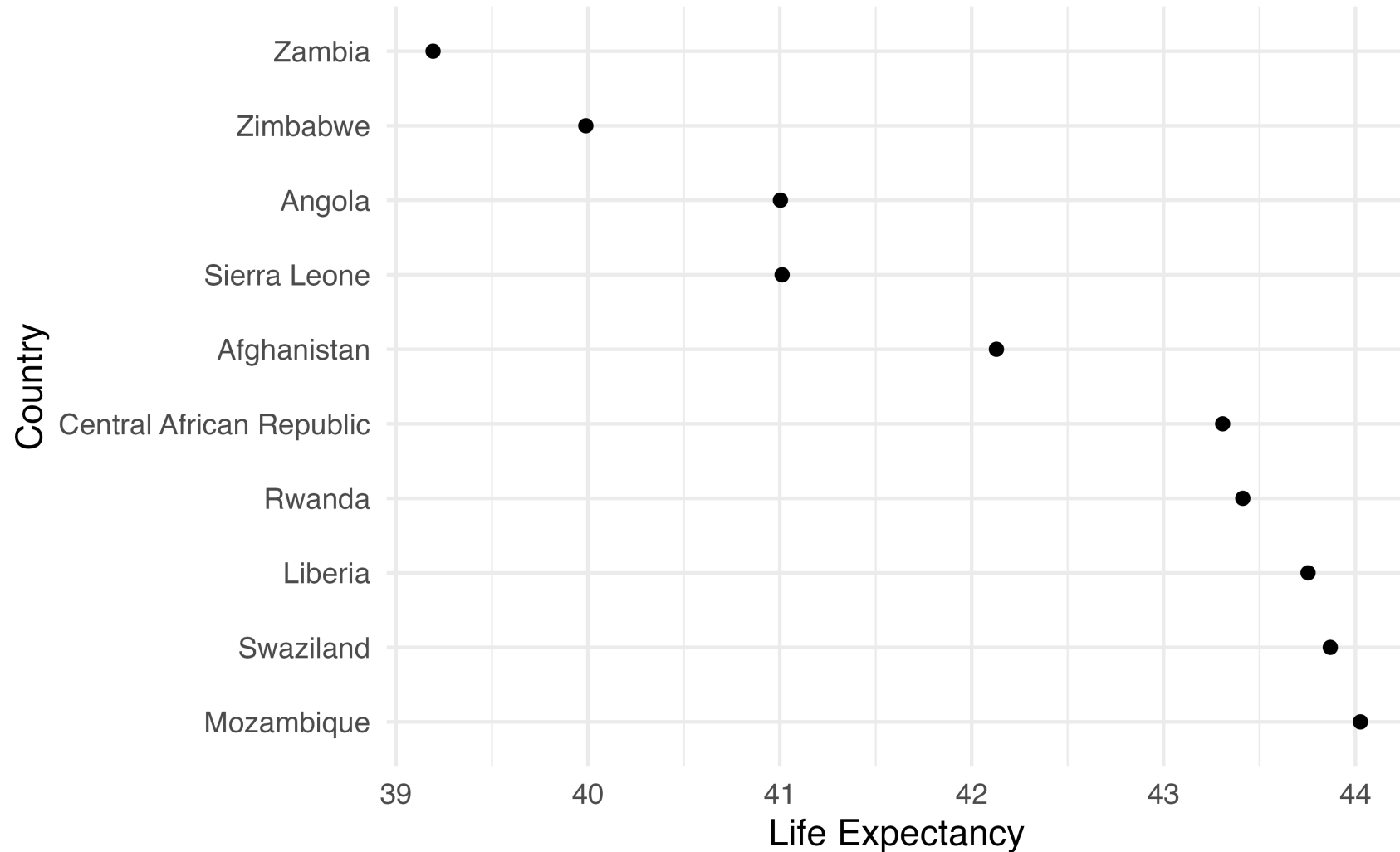
Your turn 9

```
1 top_scatter_plot(lifeExp, 2002)
```



Your turn 9

```
1 top_scatter_plot(lifeExp, 2002) +  
2   labs(x = "Life Expectancy", y = "Country")
```



Resources

R for Data Science, 2nd ed.: A comprehensive but friendly introduction to the Tidyverse. Free online.

Advanced R, 2nd ed.: Detailed guide to how R works and how to make your code better. Free online.