Do smtg repeatedly to group of rows
df %>% group_by(col1) %>% summarise(mean = mean(col2, na.rm = TRUE))
library (purrr) map_* functions ~ lapply()
Functional: func + * → c()
map_dbl → c(dbl), map_df → tibble()
map_dbl(df, fun, na.rm = TRUE)
df %>% mutate(col1 = case_when(col1 == val1 ~ new_val,
TRUE ~ col1))

```
func ← function(x, n = 2, z) {
    var ← ""
    for (i in seq_along(1:n)) {
        var ← paste0(var, x)
    }
    var
}
```

Lazy ← eval'd if accessed so above works with z
Even though z not used
R has Name Masking (tiers to vars), dynamic lookup (checks var at func call), Fresh start(each func call is indep.)

test_that(msg, {expect_*()})
expect_identical, expect_equal - tolerance, expect_equivalent (no attr)
expect_error, expect_warning, expect_output
expect_true, expect_false
1. Write your tests first (that call the function you haven't yet written), based on edge cases you expect or can calculate by hand
2. If necessary, create some "helper" data to test your function with (this might be done in conjunction with step 1)
3. Write your function to make the tests pass (in this process you might think of more tests that you want to add)
Rinse and Repeat
Use stop(msg) for defensive coding - Look Before You Leap-esque
try({x <- 1 + 2 x / 3})

Roxygen2 is useful when we setup packages and use ?func

```
#' Converts temperatures from Fahrenheit to Celsius.
#'
#' @param temp a vector of temperatures in Fahrenheit
#'
#' @return a vector of temperatures in Celsius
#'
#' @examples
#' fahr_to_celsius(-20)
```

source("code.R")
Annoying to keep using source(), hence pkg to use library()
- Each package attached by library() becomes one of the parents of the global environment
- The immediate parent of the global environment is the last package you attached, the parent of that package is the second to last package you attached, …
When you attach another package with library(), the parent environment of the global environment changes

Anon func: (function(x) x + 1)(1) → 2
(function(params) {what to do with input})(input)
map_*(data, function(arg) function_being_called(arg, other_arg)) same as map_*(data, ~ function_being_called(., other_arg))

Nested_df → A data frame which contains other data frames. This data structure is extremely useful when fitting many models and allows you to keep the data, the model meta data, and the model and its results all associated together as a single row in a data frame.
df$data[[1]] # first df in nested column, each row is group

1. Create a list column using a function such as `nest`

2. Create other intermediate list-columns by transforming existing list columns with `map`

3. Simplify the list-column back down to a data frame or atomic vector using `unnest` & `mutate` + `map_dbl`:

(data masking) - When functions like `filter` are called, there is a delay in evaluation and the data frame is temporarily promoted as first class objects, we say the data masks the workspace. This is to allow the promotion of the data frame, such that it masks the workspace (global environment) .When this happens, R can then find the relevant columns for the computation.

- code evaluation is delayed
- the `filter` function quotes columns `country` and `year`
- the `filter` function then creates a data mask (to mingle variables from the environment and the data frame)
- the columns `country` and `year` and unquoted and evaluated within the data mask

col ← enquo(col) quotes column name, !!col removes quotes for using in dplyr
Instead we can use {{col}} to replace both enquo() and !!.

Use select({{ col }} :=) (walrus) for assigning values

If you are only passing on variable to a tidyverse function, and that variable is not used in logical comparisons, or in variable assignment, you can get away with passing the dots:

```
sort_gap <- function(x, ...) { print (x+1) \n arrange(gapminder, ...)}
sort_gap(1, year, continent, country)
```

Can use if + stop on {{ col }} to check if col passes certain conditions