# Linear Regression

Jin Seo Jo

```
library(tidyverse)
```

```
data(stackloss)

fit <- lm(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc., data = stackloss)

coefficient <- coef(fit) %>%
  # Access ’enframe’ function from the ’tibble’ package
  tibble::enframe(name = "term", value = "estimate")

confidence <- confint(fit) %>%
  as_tibble(rownames = "term")

summary_tibble <- left_join(coefficient, confidence, by = "term")

coefficient
```

```
## # A tibble: 4 x 2
##   term        estimate
##   <chr>          <dbl>
## 1 (Intercept)  -39.9
## 2 Air.Flow       0.716
## 3 Water.Temp     1.30
## 4 Acid.Conc.    -0.152
```

```
confidence
```

```
## # A tibble: 4 x 3
##   term        ‘2.5 %‘ ‘97.5 %‘
##   <chr>         <dbl>    <dbl>
## 1 (Intercept) -65.0    -14.8
## 2 Air.Flow      0.431    1.00
## 3 Water.Temp    0.519    2.07
## 4 Acid.Conc.   -0.482    0.178
```

```
summary_tibble
```

```
## # A tibble: 4 x 4
##   term        estimate ‘2.5 %‘ ‘97.5 %‘
##   <chr>          <dbl>   <dbl>    <dbl>
## 1 (Intercept)  -39.9   -65.0    -14.8
## 2 Air.Flow       0.716   0.431    1.00
## 3 Water.Temp     1.30    0.519    2.07
## 4 Acid.Conc.    -0.152  -0.482    0.178
```

- `enframe()` converts named atomic vectors or lists to one- or two- column data frames.

- `as_tibble()` turns an existing object, such as a data frame or matrix, into a so-called tibble, a data frame with class `tbl_df`.

The script loads the stackloss data, fits a linear regression, extracts the coefficients and confidence intervals and then puts them together in a single object.

Scripts are excellent for "on the fly" programming, but for repeating task it is useful to wrap this up in a function. So the next task is to make a function called `tidy_lm` with the function signature (aka how it's called) `tidy_lm(formula, data)`.

Make a funciton called `tidy_lm` with the function signature `tidy_lm(formula, data)`:

```
tidy_lm <- function(formula, data) {

  fit <- lm(formula, data)
  coefficient <- coef(fit) %>% tibble::enframe(name = "term", value = "estimate")
  confidence <- confint(fit) %>% as_tibble(rownames = "term")
  summary_tibble <- left_join(coefficient, confidence, by = "term")
}

data(stackloss)
summary_tibble <- tidy_lm(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc., stackloss)
```

The function that I have written is possibly not *safe*.
To see if that's true, restart R (in the Session menu) and try to run that script again. Then I get the following error:
'Error in coef(fit) %>% tibble::enframe(name = "term", value = "estimate") :
could not find function "%>%" '

What went wrong? I didn't import `tidyverse` so the function can't find the pipe.
There are two options here. The first is to re-write the code so that any packages are explicitly called.

Re-write the funciton without using a pipe:

```
tidy_lm <- function(formula, data) {

  fit <- lm(formula, data)
  coefficient <- tibble::enframe(coef(fit), name = "term", value = "estimate")
  confidence <- as_tibble(confint(fit), rownames = "term")
  summary_tibble <- left_join(coefficient, confidence, by = "term")
}

data(stackloss)
summary_tibble <- tidy_lm(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc., stackloss)
```

We get the following error:
'Error in as_tibble(confint(fit), rownames = "term") :
could not find function "as_tibble" '

We need to add some package identifiers (like `tibble::as_tibble`):

```
tidy_lm <- function(formula, data) {

  fit <- lm(formula, data)
  coefficient <- tibble::enframe(coef(fit), name = "term", value = "estimate")
  confidence <- tibble::as_tibble(confint(fit), rownames = "term")
  summary_tibble <- dplyr::left_join(coefficient, confidence, by = "term")
}

data(stackloss)
summary_tibble <- tidy_lm(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc., stackloss)
summary_tibble
```

```
## # A tibble: 4 x 4
##   term        estimate ‘2.5 %‘ ‘97.5 %‘
##   <chr>          <dbl>   <dbl>    <dbl>
## 1 (Intercept)  -39.9   -65.0    -14.8
## 2 Air.Flow       0.716   0.431    1.00
## 3 Water.Temp     1.30    0.519    2.07
## 4 Acid.Conc.    -0.152  -0.482    0.178
```

The second is to check if a required package is loaded and if it is not either load it or print an error message.

To do this we need `.packages()`. The `.packages()` function returns a vector of strings that name each package that is attached.
For instance the `stat` package is alwyas attached in an R session (it's a base package with things like `rnorm` in it).

```
"stats" %in% .packages()
```

```
## [1] TRUE
```

What if a package isn't installed?
We can use `require(dplyr)` instead of `library(dplyr)`. The difference is that `require` returns a logical values (TRUE/FALSE) depending on if the package is available, whereas `library` will just throw an error.

Check if `dplyr` and `tibble` are attached and, if they are not, use `stop()` to send a useful error message.

```
if(!all(c("dplyr", "tibble") %in% .packages())) {
    stop("You must have the dplyr and tibble packages attached!")
}
```

Check if `dplyr` and `tibble` are attached and, if they are not attach them and add it to the top of the function.

```
if(!all(c("dplyr", "tibble") %in% .packages())) {
  library(dplyr)
  library(tibble)
  # install.packages("dplyr")
}
```

Use `require` to attach a package if it is installed and throw a useful error message if it is not.

```
if(!require(dplyr)) {
  stop("The dplyr packages must be installed. Run install.packages(\"dplyr\") and then try again.")
}
if(!require(tibble)) {
  stop("The tibble packages must be installed. Run install.packages(\"dplyr\") and then try again.")
}
```

Finally, no funciton is complete without some documentation! Good function documentation should - Describe what it does - Describe what goes in - Describe what comes out - Give a quick example of how it works

A skeleton is here:

```
my_function <- function(a, b = "2009"){
## my_function computes something
## Example my_function computes the death and birth rates in Canadian provinces ## in a given year
##
## Input:
## - a: A [type] that [what should it mean]. [If there is something that needs
  ##      to be true, say it here]. Example: A character vector of two-letter
  ##      Province abbreviations.
  ## - b: (Optional) A [type] that [what should it mean]. Default = "2009".
  ##   Example: The year as a string. Any year between 2000 and 2015
  ##
  ## Output:
  ## - Returns a [type] that [describe how to interpret the return]. Example:
  ##   Returns a list of birth and death rates for the provinces in a in year b.
  ##   The first element is the birth rate in year b, the second element is the
  ##   death rate in 2009.
  ##
  ## - Example:
  ## rates <- my_function(c("ON", "NB"), "2010")
  # Function code goes here
}
```

If we are building an R package instead of just documenting a loose funciton, we should use `Roxygen2`. The major difference is the specific formatting, but it lets us automatically generate package documentaion!

Now we write documentation for the `tidy_lm` function.

```
tidy_lm <- function(formula, data) {
  ## tidy_lm performs the linear regression lm(formula, data) and then
  ## collects the estimates and the confidence interval in a single tibble.
  ##
  ## Input:
  ## - formula: A formula object for the linear regression
  ## - data: Data for the linear regression
  ##
  ## Output:
  ## - A tibble that has columns for the estimate. The 2.5% confidence boundary
  ## 97.5% confidence boundary. Each row is one term in the formula.
  ##
  ## Example:
  ## data(stackloss)
  ## summary <- tidy_lm(stack.loss ~ ., stackloss)

  if(!require(dplyr)) {
    stop("The dplyr packages must be installed. Run install.packages(\"dplyr\") and then try again.")
  }
  if(!require(tibble)) {
    stop("The tibble packages must be installed. Run install.packages(\"dplyr\") and then try again.")
  }

  fit <- lm(formula, data)
  coefficient <- tibble::enframe(coef(fit), name = "term", value = "estimate")
  confidence <- tibble::as_tibble(confint(fit), rownames = "term")
  summary_tibble <- dplyr::left_join(coefficient, confidence, by = "term")
}
```

Finally, this exercise was a partial reimplementation of the funtion `broom:::tidy.lm`. We should look at the code for `broom:::tidy.lm` and the output of `tidy(fit, conf.int = TRUE)` to compare out code with some professional R code.

Use the data set `mtcars` to compare the broom implementation and our implementation.

```
data(mtcars)
my_tidy <- tidy_lm(mpg ~ ., mtcars)
my_tidy
```

```
## # A tibble: 11 x 4
##    term        estimate `2.5 %` `97.5 %`
##    <chr>          <dbl>    <dbl>    <dbl>
##  1 (Intercept)  12.3    -26.6     51.2
##  2 cyl          -0.111   -2.28     2.06
##  3 disp          0.0133  -0.0238   0.0505
##  4 hp           -0.0215  -0.0668   0.0238
##  5 drat          0.787   -2.61     4.19
##  6 wt           -3.72    -7.65     0.224
##  7 qsec          0.821   -0.699    2.34
##  8 vs            0.318   -4.06     4.69
##  9 am            2.52    -1.76     6.80
## 10 gear          0.655   -2.45     3.76
## 11 carb         -0.199   -1.92     1.52
```

```
fit <- lm(mpg ~ ., mtcars)
their_tidy <- broom::tidy(fit, conf.int = TRUE)
their_tidy
```

```
## # A tibble: 11 x 7
##    term        estimate std.error statistic p.value conf.low conf.high
##    <chr>          <dbl>     <dbl>     <dbl>   <dbl>    <dbl>     <dbl>
##  1 (Intercept)  12.3      18.7        0.657  0.518  -26.6      51.2
##  2 cyl          -0.111     1.05      -0.107  0.916   -2.28      2.06
##  3 disp          0.0133    0.0179     0.747  0.463   -0.0238    0.0505
##  4 hp           -0.0215    0.0218    -0.987  0.335   -0.0668    0.0238
##  5 drat          0.787     1.64       0.481  0.635   -2.61      4.19
##  6 wt           -3.72      1.89      -1.96   0.0633  -7.65      0.224
##  7 qsec          0.821     0.731      1.12   0.274   -0.699     2.34
##  8 vs            0.318     2.10       0.151  0.881   -4.06      4.69
##  9 am            2.52      2.06       1.23   0.234   -1.76      6.80
## 10 gear          0.655     1.49       0.439  0.665   -2.45      3.76
## 11 carb         -0.199     0.829     -0.241  0.812   -1.92      1.52
```

*A notte on naming conventions in R*: Why is that function called `broom:::tidy.lm`? Firstly, the three colons says that this function isn't directly exported for uses use. The broom package exports the function `tidy` instead. It has very simple code

```
broom::tidy
```

```
## function (x, ...)
## {
##     UseMethod("tidy")
## }
## <bytecode: 0x7f9f0a081c78>
## <environment: namespace:generics>
```

The function `UseMethod("tidy")` basically tells R to do the following two steps: 1. Work out what class `x` by calling `class(x)`. (`fit` has class `lm`) 2. Find a fucntion called `tidy.[class(x)]` and execute that. In this case it finds `tidy.lm()` deep inside the `broom` package and calls that.

This is a trick that is used frequently in R programming to make sure that generic functions (like `tidy` or `summary`) can work across a bunch of different types of inputs.