Reproducibility & Literate Programming

Cecina Babich Morrow

2023-09-29

Introduction to reproducibility

Numerous scientific fields are undergoing a replication crisis in which it is often impossible to confirm study results independently. While it might seem that computational results should be easier to replicate than those in medicine for example, computational disciplines have not been immune. One major barrier to reproducibility in computational fields in particular is lack of access to code. The 2020 State of AI report found that only 15% of artificial intelligence papers published their code. Papers with Code, a project aiming to document code from machine learning papers, shows around 30% of papers, an improvement but still underwhelming. Pushes to establish norms of reproducibility are striving to democratize access to the code and data needed to validate claims in the field, increasing the potential for identifying bugs.

Literate programming

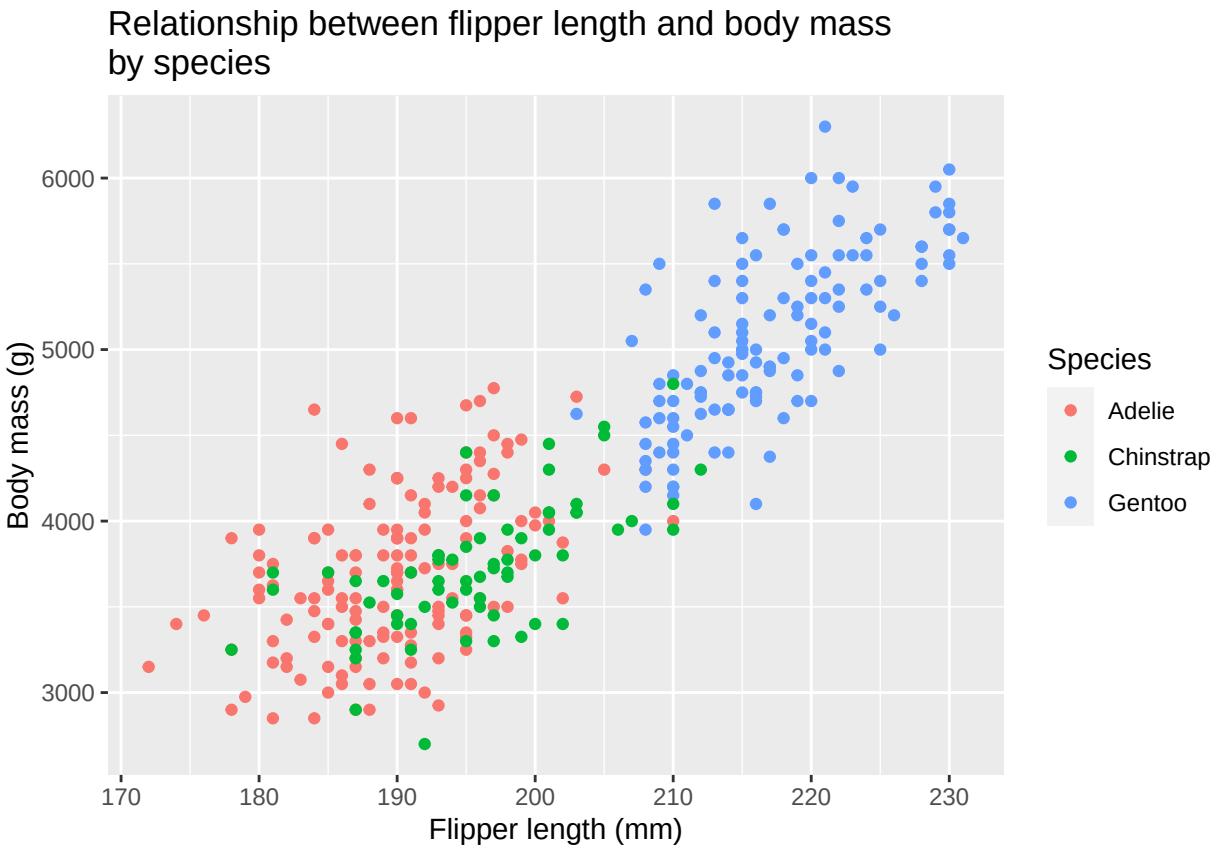
Literate programming is a process by which code is accompanied by human-readable descriptions of what that code aims to accomplish. RMarkdown, as well as the Python analogue Jupyter Notebook, is a tool for literate programming. Literate programming tools are useful to communicate the process of an analysis at a high level, while allowing for plots and other results to be continuously updated when new underlying data are obtained. RMarkdown can intersperse text and formatting (as above) with code chunks and plots:

```
head(penguins)
```

```
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
## 1 Adelie  Torgersen           39.1          18.7               181        3750
## 2 Adelie  Torgersen           39.5          17.4               186        3800
## 3 Adelie  Torgersen           40.3          18                 195        3250
## 4 Adelie  Torgersen           NA            NA                  NA          NA
## 5 Adelie  Torgersen           36.7          19.3               193        3450
## 6 Adelie  Torgersen           39.3          20.6               190        3650
## # i 2 more variables: sex <fct>, year <int>
```

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g, color = species)) +
  geom_point() +
  labs(x = "Flipper length (mm)",
       y = "Body mass (g)",
       color = "Species",
       title = "Relationship between flipper length and body mass\nby species")
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```



R & RMarkdown practice

Largest Palindrome Product

(Product Euler Problem 4)

Find the largest palindrome made from the product of two 3-digit numbers.

First, create a function that checks if a given number is a palindrome:

```
is_palindrome <- function(number) {
  reversed <- as.integer(stri_reverse(number))
  return(number == reversed)
}
```

We can start with a brute force approach checking all products of three-digit numbers to find the largest palindrome:

```
# Create a vector of all 3 digit numbers
three_digit_numbers <- 100:999
largest_palindrome <- 0 # initialize the largest to 0

tictoc::tic() # start time counter
for (i in 1:length(three_digit_numbers)) {
  for (j in 1:length(three_digit_numbers)) {
```

```
    product_ij <- three_digit_numbers[i]*three_digit_numbers[j]
    # If the product is a palindrome
    # and if it is larger than the previous largest palindrome,
    # set it as the largest
    if(is_palindrome(product_ij) & product_ij > largest_palindrome) {
      largest_palindrome <- product_ij
    }
  }
}

largest_palindrome
```

```
## [1] 906609
```

```
tictoc::toc() # report time elapsed
```

```
## 1.773 sec elapsed
```

We can improve the algorithm in the following ways: first, we can search starting with the highest three-digit numbers rather than the lowest, and second, we can use the fact that 11 must be a factor of one of the two numbers:

```
# Create a vector of all 3 digit numbers
three_digit_numbers <- 100:999
largest_palindrome <- 0 # initialize the largest to 0

tictoc::tic() # start time counter
for (i in 1:length(three_digit_numbers)) {
  number_i <- three_digit_numbers[length(three_digit_numbers)+1-i]
  if(number_i %% 11 == 0) {
    for (j in 1:length(three_digit_numbers)) {
      number_j <- three_digit_numbers[length(three_digit_numbers)+1-j]
      product_ij <- number_i*number_j
      # If the product is a palindrome
      # and if it is larger than the previous largest palindrome,
      # set it as the largest
    if(is_palindrome(product_ij) & product_ij > largest_palindrome) {
      largest_palindrome <- product_ij
    }
    }
  }
}

largest_palindrome
```

```
## [1] 906609
```

```
tictoc::toc()
```

```
## 0.17 sec elapsed
```

Sum Square Difference

(Project Euler Problem 6)

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

```
# Calculate the sum of squares
sum_of_squares <- 0 # Initialize to 0
for (i in 1:100) {
  sum_of_squares <- sum_of_squares + i^2
}

# Calculate the square of sums
sum <- 0 # Initialize to 0
for (j in 1:100) {
  sum <- sum + j
}
square_of_sum <- sum^2
```

The difference between the sum of the squares of the first one hundred natural numbers (338350) and the square of the sum (25502500) is 25164150.