Lab Two – Basic Tasks in R

- Use `R` (R Core Team 2025) to complete the tasks below. Make sure to start your solutions in on a new line that starts with "**Solution**:".
- Make sure to use the Quarto Cheatsheet. This will make completing and writing up the lab *much* easier.

1 Libraries and Packages

1.1 Part a

Install the `esquisse` package for `R`. Report the code below, ensuring to set the `eval` option to `false`, using "#| eval: false", so that we don't install the package every time you compile your Quarto document.

Solution:

```
install.packages("esquisse")
```

1.2 Part b

Load the `esquisse` package for `R`. Report the code below. It does not matter whether you set "#| eval: false" because loading the package does not add a lot of time. Note you will not evaluate any code that uses the `esquisse` package, so there is no need to load it as you compile your Quarto document.

Solution:

```
library(esquisse)
```

1.3 Part c

How can you ask for help about the `esquisse` package for `R`? Report the code below, ensuring to set the `eval` option to `false`, using "#| eval: false", so that the help document isn't loaded every time you compile your Quarto document.

Solution:

```
help("esquisse")
```

1.4 Part d

Is a demo or vignette available for the `esquisse` package for `R`? Report the code below, ensuring to set the `eval` option to `false`, using "#| eval: false", so that the you don't get errors when you compile your Quarto document.

Solution:

```
demo("esquisse")
vignette("esquisse")
```

neither the demo or the vignette were available

1.5 Part e

Add the BibTeX citation for `esquisse` package for `R` to your `.bib` file and cite it in a sentence describing what the package does below. Note you can reference a citation named `esquisse` using "[@esquisse]".

Solution: the package @esquisse explores and visualizes data interactively

1.6 Part f

Run `esquisser()` in your console – not in a chunk of `R` code in your Quarto document.

i. Select `palmerpenguins` from the "Select an environment in which to search:" dropdown. This should automatically select `penguins` in the "Select a data.frame:" dropdown. Click "Import Data".
ii. Drag `body_mass_g` to the X box and `species` to the fill box.
iii. Describe what you see in words. What can you conclude about Adelie, Chinstrap, and Gentoo penguins based on the resulting plot?

Solution: I can see a bar chart with the x axis representing body mass in grams, ranging from 2500 to 6000. The y axis is the count, or number, of pengins at each mass. The data is colored in red, green and blue with the Adelie, Chinstrap, and Gentoo penguins each representing their respective colors. From the plot, the Adelie penguins are the lightest penguins and have the highest number of observations. The Adelie penguin seems to typically. be between 3000 and 4000 grams. the Chinstrap penguins are more spread out, but seem to be the second lightest penguin species. The Gentoo penguins also have a wide spread, but seem to be the heaviest penguins mostly ranging from 4500 to 5500 grams.

2 Objects and Vectors

Create the following vectors in `R`. In some cases, you may be able to use `seq()` or `rep()` while in others you cannot. Use these functions when possible, otherwise manually create the vector and explain why that was necessary.

Some Snowday Notes

There are three ways we will create a vector. Most simply, we can create one from scratch. For example, I create a vector of odds, and evens less than 10 below.

```
(odds <- c(1, 3, 5, 7, 9))
```

```
[1] 1 3 5 7 9
```

```
(evens <- c(2, 4, 6, 8))
```

```
[1] 2 4 6 8
```

There are also built-in functions like `seq(...)` for doing this:

```
(odds <- seq(from=1, to=9, by=2))
```

```
[1] 1 3 5 7 9
```

```
(evens <- seq(from=2, to=8, by=2))
```

```
[1] 2 4 6 8
```

Either approach is easy enough with a small number of elements, but what if I wanted odds less than 100? 1000? 1 million? The `rep(...)` and `seq()` approaches would be far more efficient.

We can also create repeating sequences by hand

```
(repeating.seq1 <- c(1, 2, 3, 1, 2, 3, 1, 2, 3))
```

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

```
(repeating.seq2 <- c(1, 1, 1, 2, 2, 2, 3, 3, 3))
```

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

or using a built-in function `rep(...)`

```
(repeating.seq1 <- rep(c(1,2,3), times=3))
```

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

```
(repeating.seq2 <- rep(c(1,2,3), each=3))
```

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

There are some vectors for which we can't use a `seq()` or `rep()`. For example, consider the prime numbers less than 10. The primes are not sequential (e.g., jump by a fixed amount), nor are they repeating.

```
primes <- c(2, 3, 5, 7)
```

Note: There is a `primes` package for `R` (Keyes and Egeler 2025) that contains a `generate_primes(min, max)` function for generating a vector of primes from `min` to `max`.

2.1 Part a

The Fibonacci Sequence is a recursive formula:

$$F_n = F_{n-1} + F_{n-2}$$

where $F_0 = 0$ and $F_1 = 1$.

Create a vector of the first 10 Fibonacci numbers.

Solution: (fibonacci <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)) # created fib vector for part 2.a

2.2 Part b

Triangular Numbers are the cumulative sums of natural numbers:

$$F_n = \frac{n(n+1)}{2}.$$

Create a vector of the first 10 Triangular Numbers.

Solution: (n <- seq(from = 1, to = 10, by = 1)) # created the vector for the n to be used in the final vector

(tri.vector <- n*(n+1)/2) # created the triangular numbers vector

2.3 Part c

Suppose I were designing a repeated meausres experiment with three treatment conditions. Each of $n = 10$ participants (with `ID` 1 to 10) will receive *all* experimental conditions, call them "Control", "Treatment A", and "Treatment B".

Consider setting up data entry for this experiment.

i. Create a vector containing each `ID` repeated three times, once for each treatment.
ii. Create a vector containing each `Treatment` repeated for each participant ID.

Solution:

```
(id <- seq(1:10)) # create ID object
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
(id.vector <- rep(id, each = 3)) # create ID vector
```

```
 [1]  1  1  1  2  2  2  3  3  3  4  4  4  5  5  5  6  6  6  7  7  7  8  8  8  9
[26]  9  9 10 10 10
```

```
(treatments <- c("Control", "Treatment A", "Treatment B")) # manually create treatements object
```

```
[1] "Control"     "Treatment A" "Treatment B"
```

```
(treatments.vector <- rep(treatments, times = 10)) # create treatments vector
```

```
 [1] "Control"     "Treatment A" "Treatment B" "Control"     "Treatment A"
 [6] "Treatment B" "Control"     "Treatment A" "Treatment B" "Control"
[11] "Treatment A" "Treatment B" "Control"     "Treatment A" "Treatment B"
[16] "Control"     "Treatment A" "Treatment B" "Control"     "Treatment A"
[21] "Treatment B" "Control"     "Treatment A" "Treatment B" "Control"
[26] "Treatment A" "Treatment B" "Control"     "Treatment A" "Treatment B"
```

2.4 Part d

Create a vector containing the `character` "MATH" and the `numeric` 240. What is the resulting class? Explain why in a sentence.

Solution: (vector <- c("MATH", 240)) # created vector class(vector) # checked class of the vector the vector had a class "character" because although the vector contains both a character and a numeric object, all the data takes the class of the item all the way to the right, which is a character in this case

References

Keyes, Os, and Paul Egeler. 2025. *Primes: Fast Functions for Prime Numbers*. https://doi.org/10.32614/CRAN.package.primes.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.