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 + Part d

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.

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.

Solutions

There are multiple ways in which we can learn more about the `esquisse` package. The first command we can run to help us with this is the `help("package = ...")` command. This retrieves the package's documentation. If we'd like to see the package in use, we can check if the creator provided a demo by using the `demo("package = ...")` command. Lastly, we can check for a more detailed/comprehensive guide to the package by using the `vignette("package = ...")` command.

```
help(package = "esquisse")
demo(package = "esquisse")
vignette(package = "esquisse")
```

After running these commands, we find that the `esquisse` package for `R` includes a vignette, but not a demo.

1.4 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

We can easily find the BibTeX citation for `esquisse` by using the command `citation(package = "...")`.

```
citation(package = "esquisse")
```

After adding the citation to our `.bib` file, we can also obtain the lovely in-text citation, (Meyer and Perrier 2025).

1.5 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

Following the instructions outlined in Part F, we generate a histogram with the body mass of penguins on the x axis and the total count on the y axis. The histogram is color-coded by species of penguins. The histogram is right skewed with a peak around the 3500 gram mark. We see that the Adelie penguins are more populous than the other two species. Furthermore, Adelie penguinshave an average mass of in the 3500-4000 gram range. The second most populous species is the Gentoo Penguin. Gentoo penguins, on average, weigh more than the other two species with an average mass of around the 5000 gram mark. The least populous species of penguins is the Chinstrap penguin. Their mass distribution loosely mimics that of the Adelie, with a similar center and spread, though it is slightly more symmetric than that of the Adelie penguin. The mass distribution for the Gentoo penguin is roughly symmetric.

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
```

```r
(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.

```r
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

We can create a vector of the first 10 Fibonacci numbers using manual vector creation:

```r
(fibonacci.ten = c(0,1,1,2,3,5,8,13,21,34))
```

```
 [1]  0  1  1  2  3  5  8 13 21 34
```

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

We can create a vector of the first 10 Triangular Numbers (beginning with F_1) by creating a vector for our n values, 1 through 10, and then applying the formula provided:

```r
(n = 1:10)
```

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

```r
(triangular.ten = (n * (1+n)) / 2)
```

```
 [1]  1  3  6 10 15 21 28 36 45 55
```

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

i. We construct the ID vector by telling `R` to take the vector of numbers one through ten, and to then repeat *each component* three times before moving to the next component.

```r
(ID = rep(x=1:10, each=3))
```

```
 [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
```

ii. To create the Treatment vector, we tell R to make the three-dimensional vector containing each treatment status, and then repeat the vector *itself* three times (as opposed to going component by component like we did for the ID vector).

```
(Treatment = rep(x = c("Control","Treatment A", "Treatment B"), times = 10))
```

```
 [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.

2.5 Part 2.d

```
class("MATH")
```

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

```
class(240)
```

```
[1] "numeric"
```

```
(mixed.class = c("MATH",240))
```

```
[1] "MATH" "240"
```

```
class(mixed.class)
```

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

The resulting class of the vector above is character. This is because when we construct a vector with multiple types of data in R, R will always reference the following hierarchy:

Logical → Integer → Numeric → Complex → Character

Using this hierarchy, R identifes the entry in the vector that has the "right-most" class, and encode all data as that class. In this example, we see that our vector has two classes: character ("MATH") and numeric (240). Because character is to the right of numeric in the hierarchy, thus making it the "dominant" class, 240 is converted to a character and our final vector has the class of character.

References

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

Meyer, Fanny, and Victor Perrier. 2025. *Esquisse: Explore and Visualize Your Data Interactively*. https://doi.org/10.32614/CRAN.package.esquisse.

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