Lab 00: Week 1 (Solutions)

Contents

The **specific aims** of this lab are:

- meet some of your fellow students (there will be group work later in semester),
- refresh your knowledge of R and RStudio (or introduce you to R and RStudio if this is your first time),
- understand projects and packages in R (especially the tidyverse suite of packages),
- generate a R Markdown document, and
- familiarise you with some additional resources and how to go about getting help.

The unit **learning outcomes** addressed are:

- LO2 Extract and combine data from multiple data resources.
- LO3 Construct, interpret and compare numerical and graphical summaries of different data types including large and/or complex data sets.
- LO8 Create a reproducible report to communicate outcomes using a programming language.

1 Getting to know each other

Your tutor will lead you through a getting to know each other exercise.

2 R and RStudio

The program/language we will be using to analyse data this semester is called R. We will mostly access R through the IDE [1] RStudio. Both are free to use.

You will need to install (or upgrade to the latest version if you already have them installed):

- latest version of R (v4.0.5 or later), and
- latest version of RStudio (v 1.4 or later).

2.1 Packages

When working in R, there are some functions and data sets that are always available, but the real strength of R comes from its community of developers who continually improve the set of available features and add additional functionality through an ecosystem of "packages."

A collection of packages, mostly backed by RStudio, called the **tidyverse** has attracted a lot of attention in the statistics and data science sphere (Wickham et al. 2019). You can install the entire suite of **tidyverse** packages using the command

```
install.packages("tidyverse")
```

This will install **ggplot2** (graphics), **dplyr** (data manipulation), **readr** (importing data) and a whole slew of other useful packages (Wickham 2016; Wickham et al. 2018; Wickham, Hester, and Francois 2017). You only need to use `install.packages()` the first time. When you actually want to *use* the packages, you need to load them into the *environment*,

```
library("tidyverse")
```

When you load the **tidyverse** master package using `library("tidyverse)`, R goes and loads a bunch of other packages. It also printed out a few things, telling us which packages were loaded (and which version), and it tells us that some functions that were previously available have now been masked by the newly loaded packages. For example if you wanted to use the `filter()` function from the **stats** package, you now need to use `stats::filter()`.

You could also load each package individually, e.g.

```
library("ggplot2")
library("dplyr")
library("readr")
```

For an overview of the functionality in the **tidyverse** see R for Data Science (Wickham and Garrett 2017).

2.2 Palmer Archipelago penguins

We're going to dive in the deep end. We're going to install a package called **palmerpenguins** that contains a neat data set for us to experiment with (Horst, Hill, and Gorman 2020).

Let's install it:

```
install.packages("palmerpenguins")
```

If all went well it's now installed on your computer (which you only need to do once), but it's not currently loaded (meaning the functionality is not yet available). We load the package using the `library()` function. To help think about this, **installing** the package adds it to our library collection but when we do that, the package is stored on the shelf in the library and not really accessible. To actually **use** the package we need to take it off the shelf and check it out of the library which we do using the `library()` function.

```
library("palmerpenguins")
```

What does this package do? We can see the help page using `?` or `help()`

```
?palmerpenguins
# help(palmerpenguins)
```

Most packages bundle up a set of functions and make them available to the user when it is loaded. The **palmerpenguins** package is a little unusual in that it it doesn't provide any functions, just two data sets, `penguins` and `penguins_raw`. We will start with the raw data in `penguins_raw`. We can find out a bit more about it using the help:

```
?penguins_raw
```

When the package is loaded, the data is invisibly available (i.e. it doesn't show up in the Global Environment) until we use it for the first time. We can get an overview of the structure of the stored data using the `glimpse()` function from the **dplyr** package:

```
glimpse(penguins_raw)
```

```
Rows: 344
Columns: 17
$ studyName             <chr> "PAL0708", "PAL0708", "PAL0708", "PAL0…
$ `Sample Number`       <chr> "1", "2", "3", "4", "5", "6", "7", "8"…
$ Species               <chr> "Adelie Penguin (Pygoscelis adeliae)",…
$ Region                <chr> "Anvers", "Anvers", "Anvers", "Anvers"…
$ Island                <chr> "Torgersen", "Torgersen", "Torgersen",…
$ Stage                 <chr> "Adult, 1 Egg Stage", "Adult, 1 Egg St…
$ `Individual ID`       <chr> "N1A1", "N1A2", "N2A1", "N2A2", "N3A1"…
$ `Clutch Completion`   <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Ye…
$ `Date Egg`            <date> 2007-11-11, 2007-11-11, 2007-11-16, 2…
$ `Culmen Length (mm)`  <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9…
$ `Culmen Depth (mm)`   <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8…
$ `Flipper Length (mm)` <dbl> 181, 186, 195, NA, 193, 190, 181, 195,…
$ `Body Mass (g)`       <dbl> 3750, 3800, 3250, NA, 3450, 3650, 3625…
$ Sex                   <chr> "MALE", "FEMALE", "FEMALE", NA, "FEMAL…
$ `Delta 15 N (o/oo)`   <dbl> NA, 8.94956, 8.36821, NA, 8.76651, 8.6
```

```
$ `Delta 13 C (o/oo)`   <dbl> NA, -24.69454, -25.33302, NA, -25.3242…
$ Comments              <chr> "Not enough blood for isotopes.", NA, …
```

When you use `glimpse()`, it shows one line for each column in the data frame, with the variable name, as well as what *type* of variable R thinks the column is. Can you work out what each of them mean? Do they all make sense?

`chr` refers to character (i.e. text) variables, `dbl` are numeric variables (short for double) and `date` are date variables.

Other common data types that you'll encounter include `int` refers to integer variables, `dttm` are date time variables,`lgl` are logical variables (i.e. TRUE or FALSE)

Also, the variable `Sample number` is a sample identifier, so it's not really a numeric variable, it's actually a categorical variable. We can change it to a character vector as follows:

```
penguins_raw = penguins_raw %>%
  mutate(`Sample Number` = as.character(`Sample Number`))
# alternatively using
# penguins_raw$`Sample Number` = as.character(penguins_raw$`Sample Number`)
```

Note that the `%>%` is a pipe operator and pipes the data frame `penguins_raw` into the first argument of the function `mutate()` which then mutates (or creates if it didn't already exist) the variable `Sample Number`. Use `glimpse` again to confirm that the change has been applied.

Before we go any further, we need to notice that the variable names of `penguins_raw` do not lend themselves to easy use for coding. Specifically, spaces are tricky to deal with and special characters like parentheses or slashes aren't great to have in a variable name. We can fix this using one of my favourite packages, the **janitor** package. If it's your first time using the **janitor** package, you need to start with installing it:

```
install.packages("janitor")
```

The **janitor** package has an incredibly useful function called `clean_names()` that sensibly sanitises column names to make it easier for subsequent analysis.

```
old_names = colnames(penguins_raw)
penguins = penguins_raw %>%
    janitor::clean_names()
```

We stored the old names in `old_names`. Create a new variable called `new_names` with the clean column names and compare the old names and the "cleaned" names side by side using the `bind_cols` function from the **dplyr** package. Discuss the changes that have been made to the column names.

```
new_names = colnames(penguins)
```

```
  dplyr::bind_cols(original = old_names, clean = new_names)
```

```
# A tibble: 17 × 2
   original            clean
   <chr>               <chr>
 1 studyName           study_name
 2 Sample Number       sample_number
 3 Species             species
 4 Region              region
 5 Island              island
 6 Stage               stage
 7 Individual ID       individual_id
 8 Clutch Completion   clutch_completion
 9 Date Egg            date_egg
10 Culmen Length (mm)  culmen_length_mm
11 Culmen Depth (mm)   culmen_depth_mm
12 Flipper Length (mm) flipper_length_mm
13 Body Mass (g)       body_mass_g
14 Sex                 sex
15 Delta 15 N (o/oo)   delta_15_n_o_oo
16 Delta 13 C (o/oo)   delta_13_c_o_oo
17 Comments            comments
```

The default settings of the `janitor::clean_names()` function converts camelCase to snake_case, spaces get replaced with underscores, all text becomes lowercase and any special characters are removed (e.g. the parentheses and slashes). If present, it would also convert "%" to "percent." See the vignette for further details.

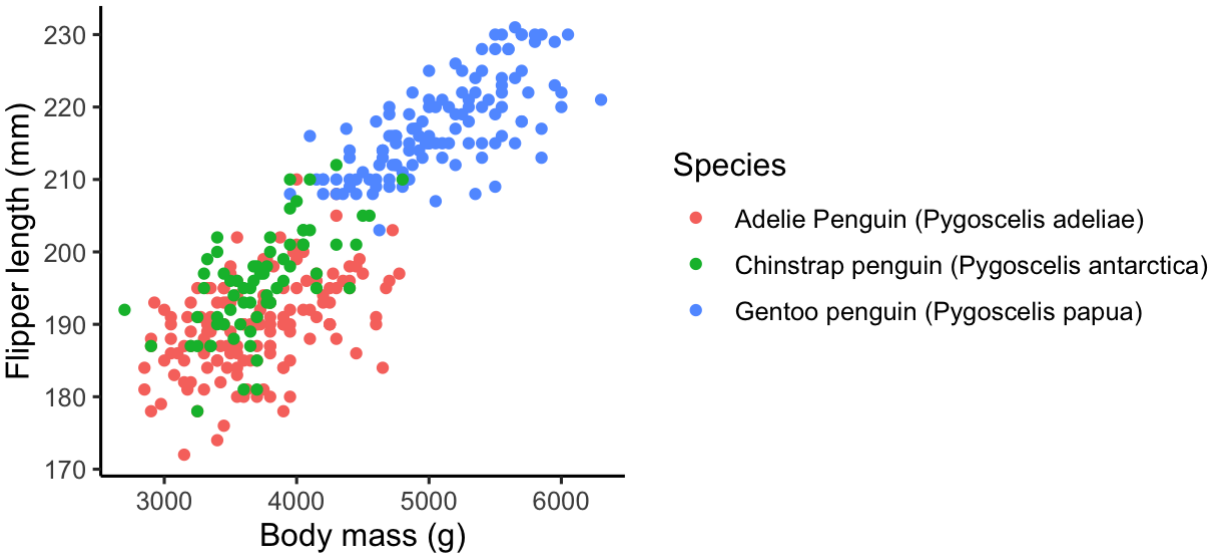
Whenever I read a new data set into R, the first thing I do is run `janitor::clean_names()` on the data frame so that I know all the variable names are in a consistent format. It makes downstream analysis and column referencing much easier.

Let's visualise some of the data using the **ggplot2** package. To make use of the **ggplot2** package, you need to install it (you probably already have, it comes when you install the **tidyverse**) and then load it (you may also have already done this if you loaded the **tidyverse** above).

Your tutor will work you through the details. The code below generates a scatter plot of flipper length against body mass and colours the points by species.

```
library("ggplot2")
penguins %>%
  ggplot() +
  # add the aesthetics
  aes(x = body_mass_g,
      y = flipper_length_mm,
      colour = species) +
  # add a geometry
  geom_point() +
  # tidy up the labels
```

```
  labs(x = "Body mass (g)",
       y = "Flipper length (mm)",
       colour = "Species")
```

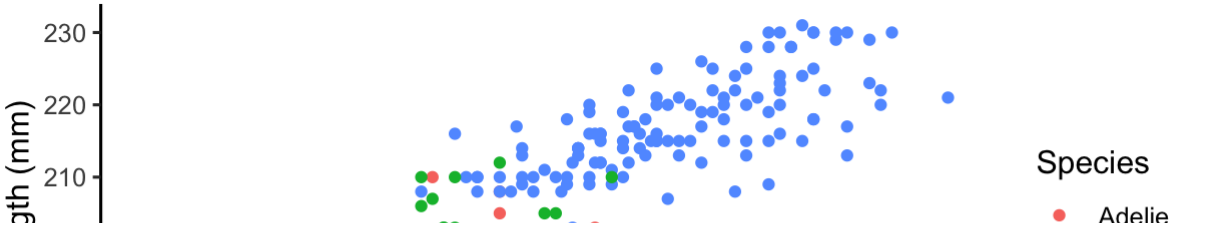


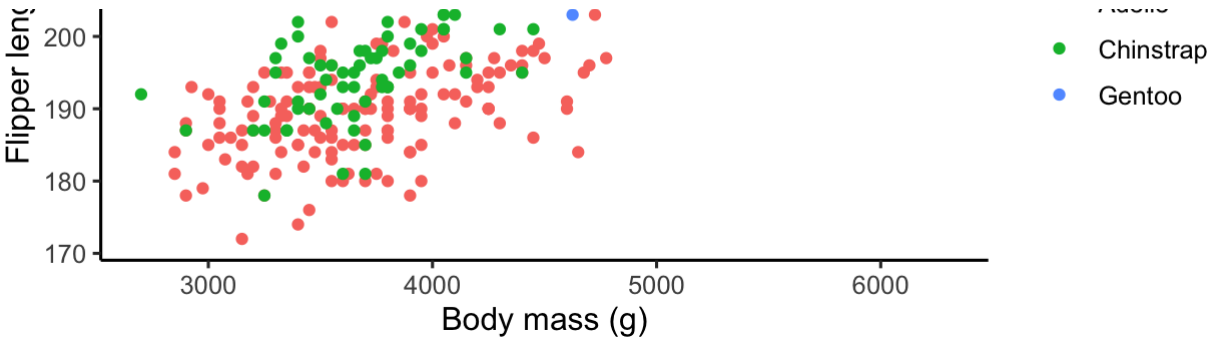
Note that the species variable is a bit long, we really only need to keep the first word, so let's do that using the `word()` function from the **stringr** package (also part of the tidyverse). In the code below, not that we're overwriting the `species` column in the `penguins` data frame using the `mutate()` function from the **dplyr** package.

```
penguins = penguins %>%
    mutate(species = stringr::word(species, start = 1, end = 1))
```

Now regenerate the plot.

```
library("ggplot2")
penguins %>%
  ggplot() +
  # add the aesthetics
  aes(x = body_mass_g,
      y = flipper_length_mm,
      colour = species) +
  # add a geometry
  geom_point() +
  # tidy up the labels
  labs(x = "Body mass (g)",
       y = "Flipper length (mm)",
       colour = "Species")
```





Let's save that plot as a `png` file so you can print it out and stick it on the fridge!

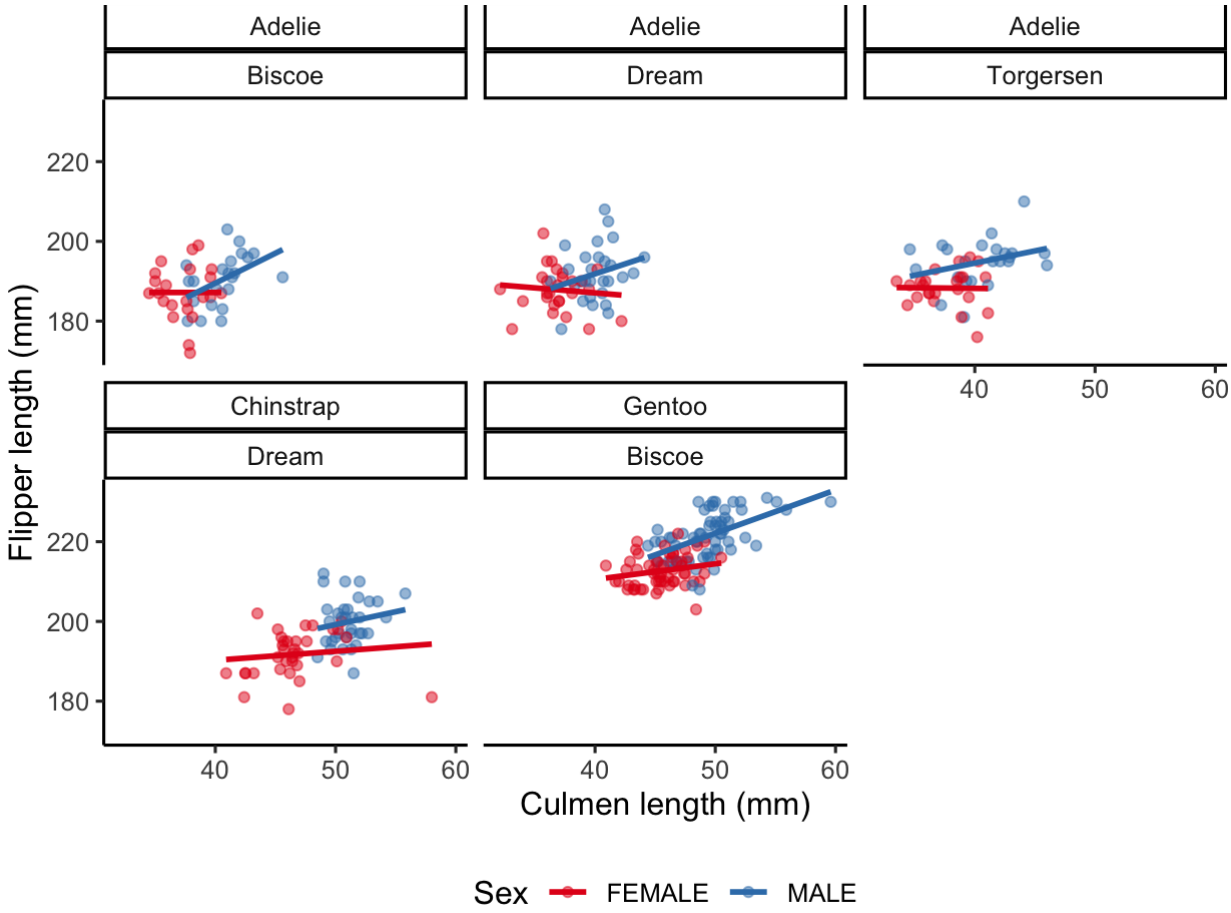
```
ggsave(filename = "myfirstggplot.png")
```

The **ggplot2** cheat sheet is a great, concise resource to find out some of what's possible. You can also access this from withing RStudio by clicking the menu item `Help > Cheatsheets`.

2.2.1 Exercises

1. Generate a scatter plot for another pair of (numeric) variables.
2. Colour by sex and use `facet_wrap()` to generate a plot for each species and island combination.
3. Try including a line of best fit by adding another geometry layer `geom_smooth(method = "lm")`.
4. Use a different geometry, `geom_histogram()` to create a histogram for flipper length, coloured by species.
5. Save an updated version of your plot using `ggsave()`.
6. Try outputting the data to a CSV file using the `write_csv()` function which can be found in the **readr** package.

```
penguins %>%
  drop_na(sex) %>%
  ggplot() +
  aes(x = culmen_length_mm,
      y = flipper_length_mm,
      colour = sex) +
  geom_point(alpha = 0.5) +
  facet_wrap(vars(species, island)) +
  geom_smooth(method = "lm", se = FALSE) +
  labs(x = "Culmen length (mm)",
       y = "Flipper length (mm)",
       colour = "Sex") +
  scale_color_brewer(palette = "Set1") +
  theme(legend.position = "bottom")
```

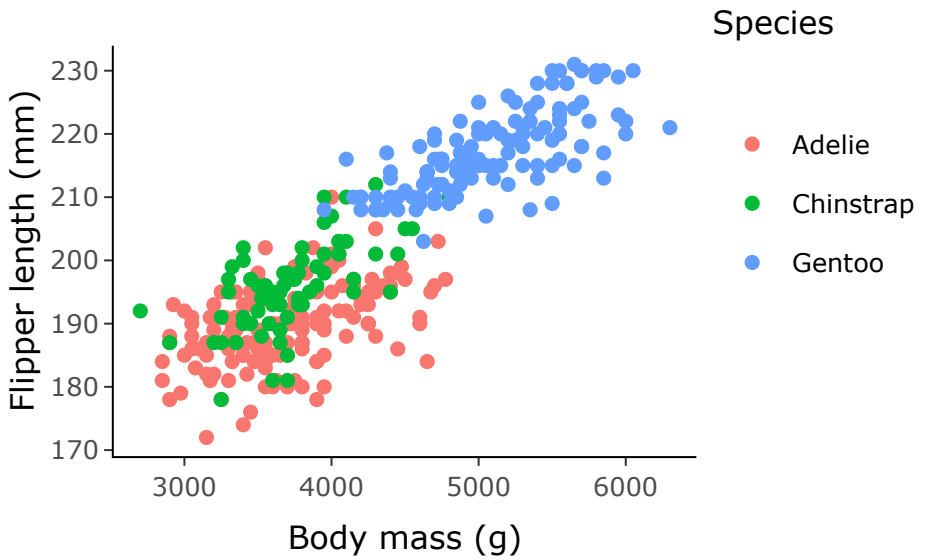



```
ggsave("my_awesome_plot.png")
readr::write_csv(penguins, file = "penguins_cleaned.csv")
```

2.2.2 Advanced: interactivity

Just for fun, let's make it interactive using the **plotly** package (Sievert et al. 2017).

```
# install.packages("plotly")
library("plotly")
myplot = penguins %>%
  ggplot() +
  # add the aesthetics
  aes(x = body_mass_g,
      y = flipper_length_mm,
      colour = species) +
  # add a geometry
  geom_point() +
  # tidy up the labels
  labs(x = "Body mass (g)",
       y = "Flipper length (mm)",
       colour = "Species")
plotly::ggplotly(myplot)
```



Here we have created an "object" in R called `myplot` which stores the ggplot. You can print the standard ggplot by typing `myplot` in the console and pressing Enter. We can also do other things with the object, such as add additional themes to it, e.g. `myplot + theme_linedraw()` or as is done above, we can feed it as an input into the function `ggplotly` from the **plotly** package.

Note that if you are implementing this in an R Markdown document (see below) the interactive graphics will only work if you output as a HTML document, it won't work for PDF or Word outputs.

2.3 R Markdown

Markdown is a lightweight markup language (in the same way the HTML is a markup language). One of the big advantages of markdown as a language is its simplicity - it forces you to focus on content rather than play with styling. R markdown is a great way to do reproducible research and generate reports. You can compile (or knit) R Markdown documents into a variety of formats, including HTML, Word, PDF, as well as presentations.

For more details on using R Markdown see http://rmarkdown.rstudio.com. A useful guide to help you get started can be found here and there's a cheat sheet here. A book on R Markdown which has everything you could possibly want to know about R Markdown and a whole lot more (Xie, Allaire, and Grolemund 2018). There's also R Markdown for Scientists which gives a more concise overview.

2.3.1 Super brief overview

1. Create a new `Rmd` file (Rmd is the R Markdown file extension). In RStudio `File -> New File -> R Markdown...`
2. When you have a Rmd file open in RStudio there's a `Knit` button up the top of the source window. You click that button to turn the markdown into HTML (or PDF or Word).

3. Text and R code can be combined in the Rmd file. Code chunks begin with three back ticks followed by `r`, the (optional) chunk name and any arguments: ```` ```{r} ```` or ```` ```{r chunk_name, tidy = TRUE} ````. The chunk also ends with three back ticks ```` ``` ````. Examples can be seen in the template that opens along as a new file in RStudio (you can delete most of the template except the YAML code at the top).

2.3.2 Including Plots

You can embed static plots in a R Markdown document without doing anything special. Important chunk options are `fig.width` and `fig.height` to set the figure width and height for example ```` ```{r, fig.width = 4, fig.height = 6}. ````

2.3.3 Chunk options

Some useful chunk options:

- `tidy = TRUE` makes the R code more readable (proper spacing)
- `results = 'hide'` hide the results of the chunk output (i.e. don't show them)
- `results = 'hold'` hold the results of the chunk output until all commands in the chunk have been run
- `warning = FALSE` don't show any warning messages (e.g. when **ggplot2** drops observations)
- `message = FALSE` don't show any messages (e.g. when packages load)
- `{r chunkname}` you can name your chunks with text immediately after the `r`. This can be particularly useful when errors pop up as it makes it easier to identify which chunk the error occurs in.

2.3.4 Exercise

Take the work you did with the Palmer penguins date and write it up in a R Markdown document. Detail what you did, including the packages and functions you used, in the text for future you. Knit using HTML.

When you do this, you'll find that each time you knit your document, it re-runs all your code and loads all the libraries from scratch. This is a) a pain and b) fantastic for reproducibility. It's a pain because you've already done things in the "global" environment, loaded data and packages, generated figures, etc, and it takes time for things to be re-run. It's fantastic for reproducibility because it means that everything you do has to be in the `Rmd` file for the knit to be successful. I.e. you'll need to load all the packages you use in the `Rmd` file, you'll need to do all the data manipulation, and include all the plot code in the `Rmd` file.

3 Test submission

Later in semester you will need to submit a R Markdown report (the first assignment). To help familiarise you with this process, we strongly recommend you try submitting your R Markdown report to the **Lab 00 practice submission** assignment on Canvas. We want to make sure you're familiar with the process of uploading a HTML file. You should always double check to make sure it has actually been submitted.

There are no marks associated with Lab 00 practice submission.

4 Review questions

1. What does `?` followed by a function or package name do?
2. Why would you use a `#` in your R code?
3. What's the difference between `install.packages("palmerpenguins")` and `library("palmerpenguins")`?
4. How can you check if a package is installed on your computer?
5. What's the difference between a warning, an error and a message?

Packages used:

- **palmerpenguins**: has a built in data set
- **dplyr**: does lots of data manipulation things, so far we have used the `glimpse()` and `mutate()` functions.
- **ggplot2**: generates nice plots
- **plotly**: makes interactive plots
- **readr**: can write an R data frame to a csv file

Footnotes

1. Integrated developer environment [↩]

References

References

Horst, Allison Marie, Alison Presmanes Hill, and Kristen B Gorman. 2020. *Palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*. https://allisonhorst.github.io/palmerpenguins/.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2017. *Plotly: Create Interactive Web Graphics via 'Plotly.js'*. https://CRAN.R-project.org/package=plotly.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. New York, NY: Springer-Verlag. http://ggplot2.tidyverse.org/.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686.

https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2018. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, and Grolemund Garrett. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. Sebastopol, CA: O'Reilly Media. http://r4ds.had.co.nz/.

Wickham, Hadley, Jim Hester, and Romain Francois. 2017. *Readr: Read Rectangular Text Data*. https://CRAN.R-project.org/package=readr.

Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. 1st ed. Chapman & Hall/CRC the r Series. Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown/.