01 Getting Started

Prof S

2025-01-14

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com (http://rmarkdown.rstudio.com).

All RMarkdown files have to have the header (the stuff between the three dashes above) and the first code chunk above called “r setup”.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

This is a code chunk (Driver 1)

We designate a code chunk with this special punctuation of three backwards apostrophes followed by a curly bracket `{` and then `r` to indicate that it should execute r code and finally some text to indicate the `chunk name` and then a `}` . Your code won’t run if you don’t have the “r” in the curly brackets, but a name to the code chunk isn’t required. Naming your code chunk helps debug things so you can see what section failed by chunk. Note, every code chunk has to have a *unique* name, otherwise you’ll get an error when you try to knit. I’ve named the code chunk below “example chunk”. Then we have some lines of code, and then we end the chunk with three more backward apostrophes.

```
a <- 3
b <- 5

a+b
```

```
## [1] 8
```

You can also point and click to start a new chunk by going to the “Code” menu above, and choose the very first function “Insert Chunk”. You can also use the hotkey combination of Ctrl-Alt-I on a PC or Command-Option-I on a Mac to add a new chunk (but you’ll need to remember to add the name of the chunk). Try to create your own

new code chunk below this text.

Practice: Your new code chunk here

Running code

You can run the code by hitting the green play button in the top right corner of this code chunk. You can also run all code preceding this chunk with the down play button. These choices will run the code in the Console in the bottom left part of your RStudio Screen and the output will be shown beneath the code chunk. You can also run this code by highlighting the code and hitting Command-Enter on a Mac or Control-Enter on a PC.

What are the basics of R grammar?

R is an object oriented language which means that we assign stuff to objects. We do this using the backward arrow sign `<-` You can see that we have created the object `a` as `a <- 3` , and the object `b` as `b <- 5` . Once we have created these objects we can use and manipulate them in code as we do here to add the two together as `a + b` . Note that we did NOT save the sum as an output, but instead it outputs the answer `8` in the console (or below the code chunk if you have pressed the green play button to run the code).

Practice: Create a new object

Create your own new object called `c` in a new code chunk and assign it as a numeric value. Then try adding your object c to `a+b` .

Using functions (Driver 2)

R has a lot of built in functions and built in datasets that we can use to see how they work. The first one we'll use is `chickwts` which has data on an experiment that compares chick growth on various feed supplements.

The first function we'll use is `str()` which gives us information about our dataset.

```
str(chickwts)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
##  $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

We can see that this object is a dataframe with 71 observations (rows) of two variables (columns). The first variable `weight` is numeric, and the second variable `feed` is a categorical factor with 6 levels (six different types of feed).

Next we'll use `head()` which will allow us to see the first six rows of a dataset. The syntax is `head(dataset)`

```
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

Getting help with functions

We can use `?name_of_function` to see the help file in the bottom R quadrant of our RStudio console

```
?help
```

We can also use the function `unique()` to see what different feeds are being compared. To do this we refer to a specific column in the dataset using the `$`. The syntax is `dataset$column` to refer to a specific column. Then we wrap that in `unique()` to get the unique entries in that column as `unique(dataset$column)`

```
unique(chickwts$feed)
```

```
## [1] horsebean linseed   soybean   sunflower meatmeal  casein
## Levels: casein horsebean linseed meatmeal soybean sunflower
```

You can see the six types of feed as “horsebean”, “linseed”, “soybean”, “sunflower”, “meatmeal”, and “casein”

Loading libraries

As an open source software R, anyone can identify a need and write a package that streamlines a specific workflow without you having to recreate the wheel. We install packages using the function `install.packages()` with the syntax `install.packages("name_of_package")`. It’s best to only load this line once, (or rerunning it will repeatedly install the package). I have the `install.packages()` “commented” out using the `#`. Get rid of the `#` to run this once, and then add the `#` back to re-comment the code. I am going to have you install the janitor package which helps us clean up column names when we are wrangling data. We’ll use this later on when we are working with the penguins data.

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

Then we load packages using the command `library(name_of_package)`

```
library(janitor)
```

```
## Warning: package 'janitor' was built under R version 4.3.3
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

Our workhorse package in the course is called `tidyverse`. We'll load that here. We also use the `here` package to help tell R where to look for our files when we are reading in data, and where to put our output. Typically we load all the packages we're going to need up in a code chunk near the top of our RMarkdown

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

```
library(here)
```

```
## Warning in readLines(f, n): line 1 appears to contain an embedded nul
```

```
## here() starts at /Volumes/Selden_SSD/Wellesley Courses/BISC198_S25/BISC198_S25_R
```

The first step in most of our coding exercises will be to read in data. To read in a csv file we use the function `read_csv()` in the tidyverse package and `here()` in the here package. The syntax is `read_csv(here::here(file_path_to_data))`. So in this example, the dataset we want is in the folder `AdditionalData` and is called `palmerpenguin_raw.csv`

```
penguin <- read_csv(here::here("AdditionalData/palmerpenguin_raw.csv"))
```

```
## Rows: 344 Columns: 17
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (9): studyName, Species, Region, Island, Stage, Individual ID, Clutch C...
## dbl  (7): Sample Number, Culmen Length (mm), Culmen Depth (mm), Flipper Leng...
## date (1): Date Egg
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Practice: Examine dataset

Use the functions `str()` and `head()` to examine the penguin dataset using the chunk below.

Data Wrangling (Driver 3)

Manually renaming column names

You can see that some of the column names have spaces on them which R doesn't like, and you'd have to refer to them using the backwards apostrophes. For example `Flipper Length (mm)`. We can rename that column manually using the function `rename()`.

But to do that we need to be able to save our work. The tidyverse package has some nice syntax that we'll use throughout the course where we take the old dataset and do something to it and save our work in either a new object or overwriting the old object. We'll use the overwrite the old object method here. The tidyverse syntax for this is: `dataset <- dataset %>% function()`. The `%>%` is called a "pipe". Essentially we can read the pipe as "then". So it's take my dataset "then" apply our function to it. They syntax for the rename function is `data <- data %>% rename(new_name=old_name)`

```
penguin <- penguin %>%
  rename(flipper_length_mm =`Flipper Length (mm)`)
```

We can see that this worked by using the `colnames(data)` function

```
colnames(penguin)
```

```
##  [1] "studyName"          "Sample Number"      "Species"
##  [4] "Region"             "Island"             "Stage"
##  [7] "Individual ID"      "Clutch Completion"  "Date Egg"
## [10] "Culmen Length (mm)" "Culmen Depth (mm)"  "flipper_length_mm"
## [13] "Body Mass (g)"      "Sex"                "Delta 15 N (o/oo)"
## [16] "Delta 13 C (o/oo)"  "Comments"
```

Practice: Rename `Sample Number`

Practice by renaming the column `Sample Number` to `sample_number`

Using `clean_names` to clean all the column names at once

This time, we are going to us the the `clean_names` function to clean all the column names at once, and save this clean dataset as a new object called `clean_penguin`. This particular function just takes the overall dataset as the input so we don't need anything inside the parentheses

```
clean_penguin <- penguin %>%
  clean_names()
```

Practice: View new column names

Use the `colnames` function to see the resulting new column names in `clean_penguin`

Saving output of data wrangling

We can save our newly cleaned data to a file using `write_csv` and we are going to save it in our "Output" folder. The syntax is `write_csv(dataset, here::here("path_to_output"))`

```
write_csv(clean_penguin, here::here("Output/clean_penguin.csv"))
```

Application: Reading in and Cleaning Up Sydney Beaches dataset (Driver 1)

1. Read in the Sydney Beaches dataset ("sydneybeaches.csv") also in the "AdditionalData" folder using `read_csv()`.
2. Clean up the column names using `clean_names()`.
3. Save the output as `clean_sydneybeaches.csv` in the "Output" folder.

Knitting the RMarkdown file

When we are finished with our code we can knit the file using the Knit button up top (ball of yarn with knitting needles). This also will save your code (though you should be saving frequently). Know that every time we knit it's a new R session, so everything you need has to be in *this file*. So your code might work when you hit the green play buttons if you have a data object loaded from a previous script, but it will fail if that data object isn't loaded in this particular file. And objects created when knitting are NOT saved in your Environment.

Cleaning up your workspace

It's good practice at the end of your session to clean up your workspace using the broom icon in the top right. That way you only have the objects loaded that you need in the specific script you are using. I also don't save my workspace, because it slows R down, so if prompted with something like "Save workspace image to /.RData?", say no.