# Introduction to Coding in R

EKB

2023-02-06

## Introduction to R and RStudio

#### Student Learning Outcomes

- Students will be able to explain the benefit of each of the four main panels in RStudio
- Students will be able to do the following in the R languages:
  - perform mathematics
  - assign objects
  - use functions
  - explore and describe vectors (1D) and data frames (2D data)

### RStudio Cloud Tour

Let's first explore what each of the panels in RStudio do.

1. *Source* (upper left): This is where documents which have data or code in them are opened. You can save all the code you type here for future (re)use, which is a big reason coding in R is reproducible.
2. *Console* (bottom left): This is where code from the source is "run" and you see the outputs. You can also execute lines of code which you type into the console, but they will not be saved. You can think of this section as where RStudio really interfaces with R–it is where R actually evaluates code within RStudio.
3. *Environment* (upper right): This panel becomes more helpful as you get familiar with R and RStudio. It keeps track of data objects and other items you have created and gives a bit of information about them.
4. *Files/Help/etc.* (bottom right): This panel is (clearly) very multifaceted. The Files tab lets you see all the files in your current workspace. For us, the Help tab is probably what we will use the most. This is where we can search the R documentation for information about functions we use.

## Basics of Coding in R

How do you "run" code? Running or executing code means that you are sending a line of code to the console for R to interpret it.

In the source, there are a few different ways to run code.

- Sometimes you want to run only one line of code at a time. You can do this by putting your cursor on the line you want to run and either hitting the "Run" button in the upper right-hand corner of the source panel or holding down `Ctrl` + `Enter` (`Cmd` + `Enter` on Macs). If you want to run a couple lines of code, you can highlight them and do the same thing

- If you want to run an entire code chunk (see below), you can click on the green arrow on the right side of the code chunk.

In the console, you hit `Enter`.

## Using R as a calculator

For example:

```
3 + 5
```

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

```
15 / 5
```

```
## [1] 3
```

```
4^2
```

```
## [1] 16
```

**Let's practice in the console, too.**

You can do basic math in the console. The console only understands R code, so we don't need to use the `{r}` notation; we can type numbers and mathematical symbols. Try multiplying 5 and 3 (hint: * means multiply) in the console.

## Assigning Objects

Assignments are really key to almost everything we do in R. This is how we create permanence in R. Anything can be saved to an object, and we do this with the assignment operator, `<-`.

The short-cut for `<-` is `Alt + -` (or `Option + -` on a Mac). We will be typing this *a lot*.

```
mass <- 47.5
age <- 122
```

We can also perform mathematical functions on numeric objects.

```
# we can add comments to our code by using the # sign
# R doesn't read anything past a #, so you can either put one at the beginning of a whole line or after

mass <- mass * 2         # multiply
age <- age - 20          # subtract
mass_index <- mass/age   # divide
```

## Functions

Functions are pre-written bits of codes that perform specific tasks for us.

Functions are always followed by parentheses. Anything you type into the parentheses are called arguments. Arguments allow us to give the function additional information about how we want it to perform its task.

```
weight_kg <- sqrt(10) # square root

round(weight_kg) # rounding
```

```
## [1] 3
```

```
round(weight_kg, digits = 2) # round to 2 digits past 0
```

```
## [1] 3.16
```

To get more information about a function, use the help function.

```
# these two lines of code do the same thing
help(mean)
?mean
```

## Vectors

Vectors are the most common and basic data type in R. They make up most of the other data types we will work with in R. They are composed of series of values, which can either be numbers or characters.

We use the `c()` function (stands for concatenate) to create a vector.

```
# Let's create a vector of animal weights (numeric)
weight_g <- c(50, 60, 65, 82)
weight_g
```

```
## [1] 50 60 65 82
```

```
# A vector can also contain character strings (character)
animals <- c("mouse", "rat", "dog")
animals
```

```
## [1] "mouse" "rat"   "dog"
```

There are many functions we can use to look at vectors and learn more about them.

```
# how many elements
length(weight_g)
```

```
## [1] 4
```

```
length(animals)
```

```
## [1] 3
```

```
# type of data we are working with
class(weight_g)
```

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

```
class(animals)
```

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

```
# structure of an object
str(weight_g)
```

```
##  num [1:4] 50 60 65 82
```

```
str(animals)
```

```
##  chr [1:3] "mouse" "rat" "dog"
```

Vectors can only be one data type. Let's experiment with that.

```
test_vec <- c(weight_g, animals)
test_vec
```

```
## [1] "50"    "60"    "65"    "82"    "mouse" "rat"   "dog"
```

```
class(test_vec) # coerced everything into character (don't know how to make words numeric)
```

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

### Sub-setting by Index

Sometimes we want to pull out and work with specific values from a vector. This is called sub-setting (taking a smaller set of the original).

One way to do this is by an "index," meaning the position of the value or object in the vector. To do this, we use square brackets and a number to indicate the position.

```
weight_g[2]
```

```
## [1] 60
```

```r
weight_g[c(2,4)]
```

```
## [1] 60 82
```

```r
weight_g[c(1:4)]
```

```
## [1] 50 60 65 82
```

### Conditional subsetting

Another way in which we can subset data is through conditions. The vector will only return data which meets the conditions we set.

```r
weight_g[weight_g > 55]
```

```
## [1] 60 65 82
```

```r
animals[animals == "rat"]
```

```
## [1] "rat"
```

### Group Challenge

Let's practice! Write a few lines of code that do the following:

- create a vector with numbers from 6 to 1, in reverse order
- assign the vector to an object named `vec`
- using the index method, subset `vec` to include the last 3 numbers (should include 3, 2, 1)
- find the sum of the numbers (hint: use the `sum()` function)

```r
# the answer you should get out is 6
vec <- c(6, 5, 4, 3, 2, 1)
vec
```

```
## [1] 6 5 4 3 2 1
```

```r
vec <- vec[4:6]
vec
```

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

```r
sum(vec)
```

```
## [1] 6
```

## Working with Data Frames

Most of the data we work with is two-dimensional, i.e., it has columns and rows. Its structure resembles a spreadsheet.

- **rows** go side-to-side
- **columns** go up-and-down

R is really good with these types of data.

Data frames are made up of multiple vectors. Each vector becomes a column.

To explore data frames, we are going to use a package called `palmerpenguins`.

A *package* is a bunch of pre-written code, often in the form of functions, which we can bring into R and use. In this case, we are using a data package, which loads data into R that we can use. It is real data from penguins in Antarctica! You can learn more about the `palmerpenguins` package and data here.

```
# code for installing a package from the internet for future reference
# install.packages("palmerpenguins")
# to run the line of code above, remove the # symbol
```

I've already set it up where RStudio Cloud has `palmerpenguins` installed, AKA downloaded from the internet. We now need to tell RStudio that we want to use it. We will need to do that every time we open RStudio and want to use it.

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

Let's take a look at our data. We sometime call data frame 2-dimensional data because they have both rows and columns.

```
penguins
```

```
## # A tibble: 344 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
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

### Functions

As with vectors, there are many functions that are useful for taking a look at data frames. Many of the ones that work with vectors also work with data frames. Here are a few of the ones I find very helpful.

```r
head(penguins) # first 6 lines
```

```
## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g sex
##   <fct>   <fct>           <dbl>         <dbl>            <int>       <int> <fct>
## 1 Adelie  Torge~           39.1          18.7              181        3750 male
## 2 Adelie  Torge~           39.5          17.4              186        3800 fema~
## 3 Adelie  Torge~           40.3          18                195        3250 fema~
## 4 Adelie  Torge~           NA            NA                 NA          NA <NA>
## 5 Adelie  Torge~           36.7          19.3              193        3450 fema~
## 6 Adelie  Torge~           39.3          20.6              190        3650 male
## # ... with 1 more variable: year <int>
```

```r
head(penguins, 10) # can specify how many lines
```

```
## # A tibble: 10 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
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # ... with 2 more variables: sex <fct>, year <int>
```

```r
tail(penguins) # last 6 lines
```

```
## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g sex
##   <fct>   <fct>           <dbl>         <dbl>            <int>       <int> <fct>
## 1 Chinst~ Dream            45.7          17                195        3650 fema~
## 2 Chinst~ Dream            55.8          19.8              207        4000 male
## 3 Chinst~ Dream            43.5          18.1              202        3400 fema~
## 4 Chinst~ Dream            49.6          18.2              193        3775 male
## 5 Chinst~ Dream            50.8          19                210        4100 male
## 6 Chinst~ Dream            50.2          18.7              198        3775 fema~
## # ... with 1 more variable: year <int>
```

```r
str(penguins) # structure
```

```
## tibble [344 x 8] (S3: tbl_df/tbl/data.frame)
##  $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
##  $ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
##  $ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
##  $ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

```
nrow(penguins) # number of rows
```

```
## [1] 344
```

```
ncol(penguins) # number of columns
```

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

```
names(penguins) # same as colnames(penguins) in a df
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

### Sub-setting using Indexing

When subsetting data frames, we need to now specify 2 locations, the row and the column. In R, it is always row *then* column. Note that this is typically the opposite of spreadsheets.

```
# in vectors, only 1 dimension, so we only need to specify one location
# data frames are 2-dimensional, so he have to specify 2 different locations

penguins[1:10, c(2,3)]
```

```
## # A tibble: 10 x 2
##    island    bill_length_mm
##    <fct>              <dbl>
##  1 Torgersen           39.1
##  2 Torgersen           39.5
##  3 Torgersen           40.3
##  4 Torgersen           NA
##  5 Torgersen           36.7
##  6 Torgersen           39.3
##  7 Torgersen           38.9
##  8 Torgersen           39.2
##  9 Torgersen           34.1
## 10 Torgersen           42
```

```
penguins[1:10, ]
```

```
## # A tibble: 10 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
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # ... with 2 more variables: sex <fct>, year <int>
```

```
penguins[ , c(1:4)]
```

```
## # A tibble: 344 x 4
##    species island    bill_length_mm bill_depth_mm
##    <fct>   <fct>              <dbl>         <dbl>
##  1 Adelie  Torgersen           39.1          18.7
##  2 Adelie  Torgersen           39.5          17.4
##  3 Adelie  Torgersen           40.3          18
##  4 Adelie  Torgersen           NA            NA
##  5 Adelie  Torgersen           36.7          19.3
##  6 Adelie  Torgersen           39.3          20.6
##  7 Adelie  Torgersen           38.9          17.8
##  8 Adelie  Torgersen           39.2          19.6
##  9 Adelie  Torgersen           34.1          18.1
## 10 Adelie  Torgersen           42            20.2
## # ... with 334 more rows
```

#### Select individual columns

Often, we want to select a specific column to perform calculations on or to plot. To do this, we use the $ operator.

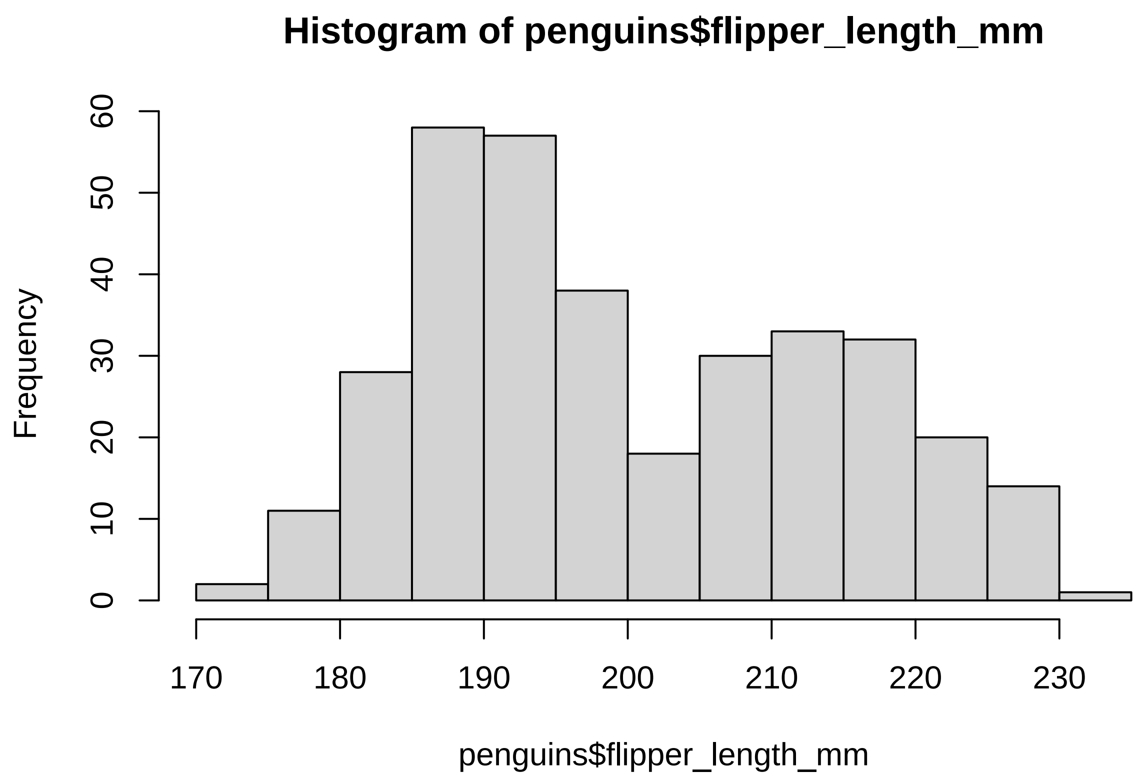
```
penguins$species
```

```
##   [1] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##   [8] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [15] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [22] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [29] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [36] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [43] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [50] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [57] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [64] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [71] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [78] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [85] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [92] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
##  [99] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [106] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [113] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [120] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [127] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [134] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [141] Adelie    Adelie    Adelie    Adelie    Adelie    Adelie    Adelie
## [148] Adelie    Adelie    Adelie    Adelie    Adelie    Gentoo    Gentoo
## [155] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [162] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [169] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [176] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [183] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [190] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [197] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [204] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [211] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [218] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [225] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [232] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [239] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [246] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [253] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [260] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [267] Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo    Gentoo
## [274] Gentoo    Gentoo    Gentoo    Chinstrap Chinstrap Chinstrap Chinstrap
## [281] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [288] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [295] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [302] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [309] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [316] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [323] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [330] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [337] Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap Chinstrap
## [344] Chinstrap
## Levels: Adelie Chinstrap Gentoo
```

```r
# we can save single columns as vectors with the assignment operator
flipper_lenght_mm <- penguins$flipper_length_mm
```

Let's plot a histogram with the flipper length data.

```r
# Plot a histogram
hist(penguins$flipper_length_mm) # same as hist(flipper_length_mm)
```



We can also perform calculations on these vectors.

```
mean(penguins$flipper_length_mm)
```

```
## [1] NA
```

```
sd(penguins$flipper_lenght_mm)
```

```
## Warning: Unknown or uninitialised column: ‘flipper_lenght_mm‘.
```

```
## [1] NA
```

```
# min, max, median, mode are other functions we might want to use
```

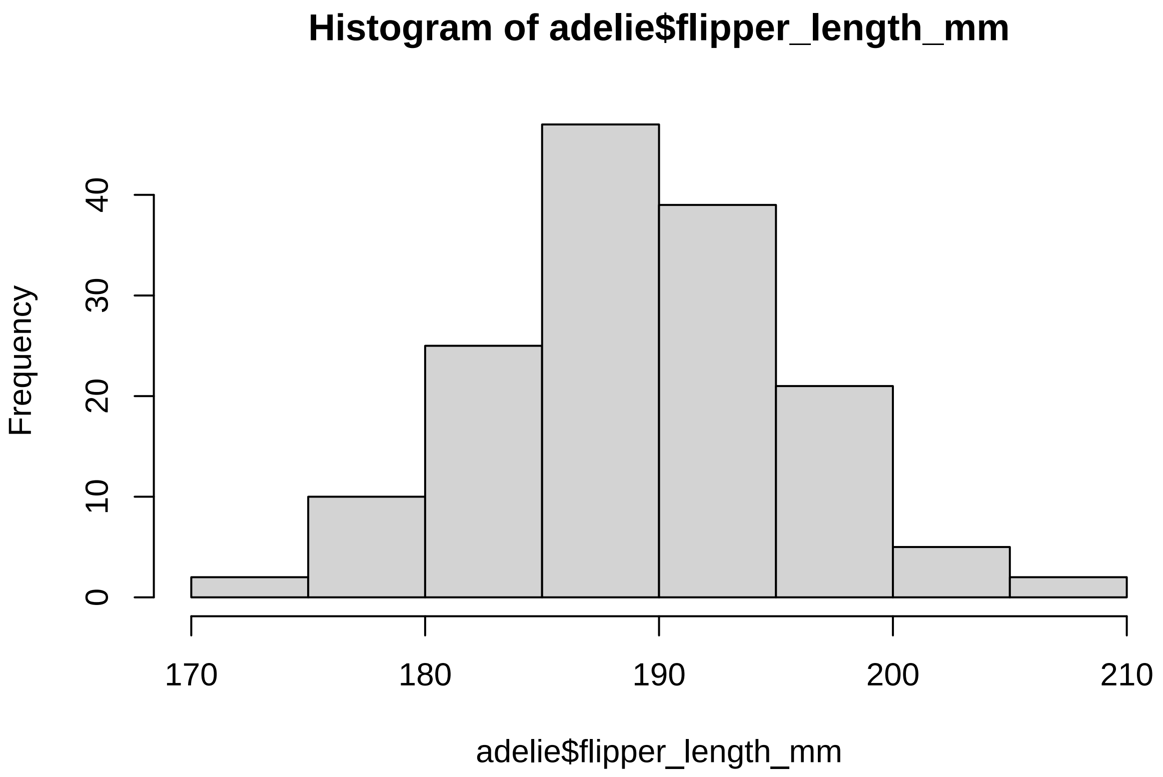
### Conditional Sub-setting

As with vectors, we can use conditional formatting to select specific observations (typically rows).

```
adelie <- penguins[penguins$species == 'Adelie', ]
adelie
```

```
## # A tibble: 152 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
##  7 Adelie  Torgersen           38.9          17.8                181        3625
##  8 Adelie  Torgersen           39.2          19.6                195        4675
##  9 Adelie  Torgersen           34.1          18.1                193        3475
## 10 Adelie  Torgersen           42            20.2                190        4250
## # ... with 142 more rows, and 2 more variables: sex <fct>, year <int>
```

```
hist(adelie$flipper_length_mm)
```



```
# dealing with numeric columns with NA values
mean(adelie$flipper_length_mm)
```

```
## [1] NA
```

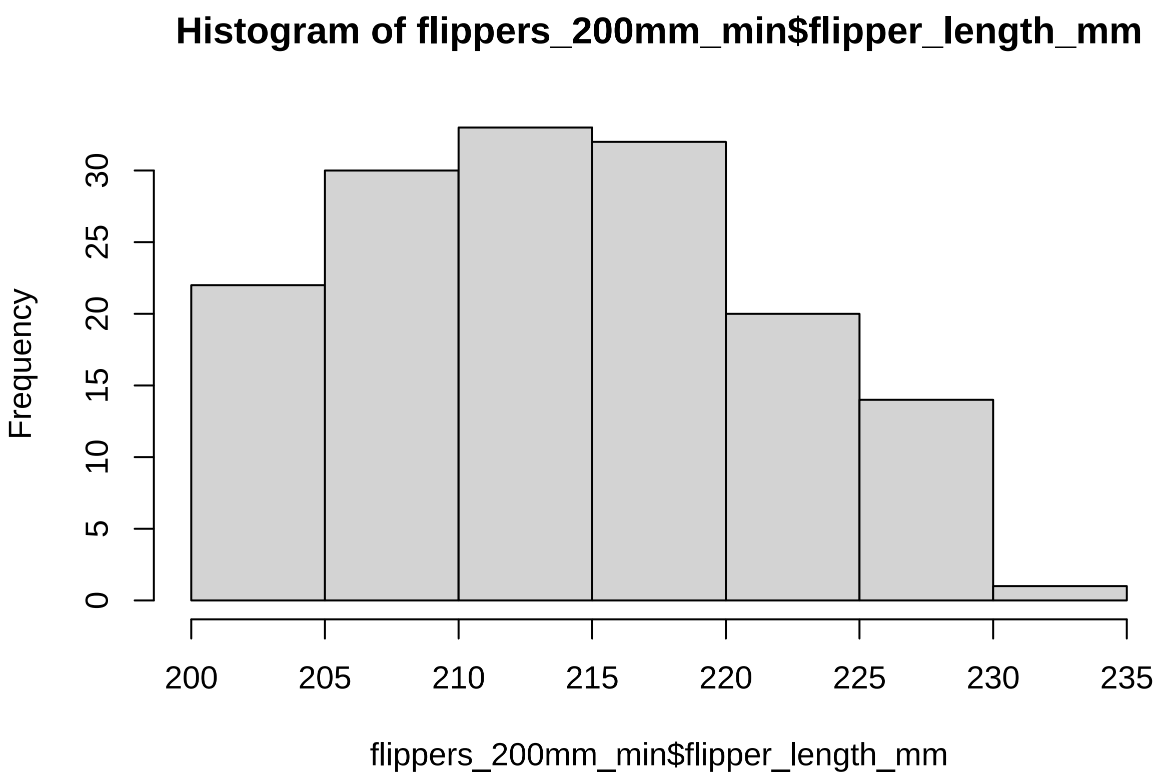
```
mean(adelie$flipper_length_mm, na.rm = TRUE)
```

```
## [1] 189.9536
```

We can also use conditional formatting to filter rows based on numeric conditions.

```
# penguins with flippers greater than or equal to 200 mm
flippers_200mm_min <- penguins[penguins$flipper_length_mm >= 200, ]

# create a histogram
# hist(flippers_200mm_min) # why doesn't this work? We haven't specified a column
hist(flippers_200mm_min$flipper_length_mm)
```



**Group Challenge**

Write some lines of code to do the following: calculate the minimum and maximum body mass values for Gentoo penguins. Remember the `na.rm` argument!

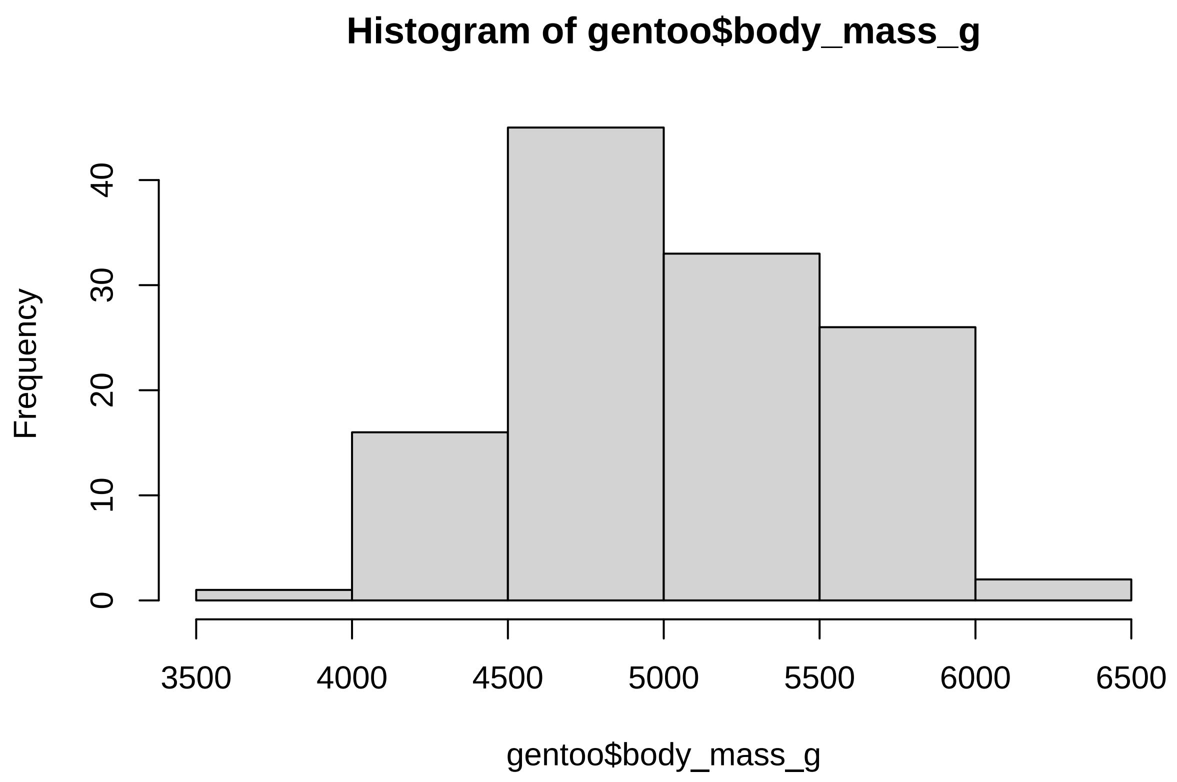
```
gentoo <- penguins[penguins$species == "Gentoo", ]
min(gentoo$body_mass_g, na.rm = TRUE)
```

```
## [1] 3950
```

```
max(gentoo$body_mass_g, na.rm = TRUE)
```

```
## [1] 6300
```

```
hist(gentoo$body_mass_g)
```


Histogram of gentoo$body_mass_g
Frequency
0
10
20
30
40
3500
4000
4500
5000
5500
6000
6500
gentoo$body_mass_g