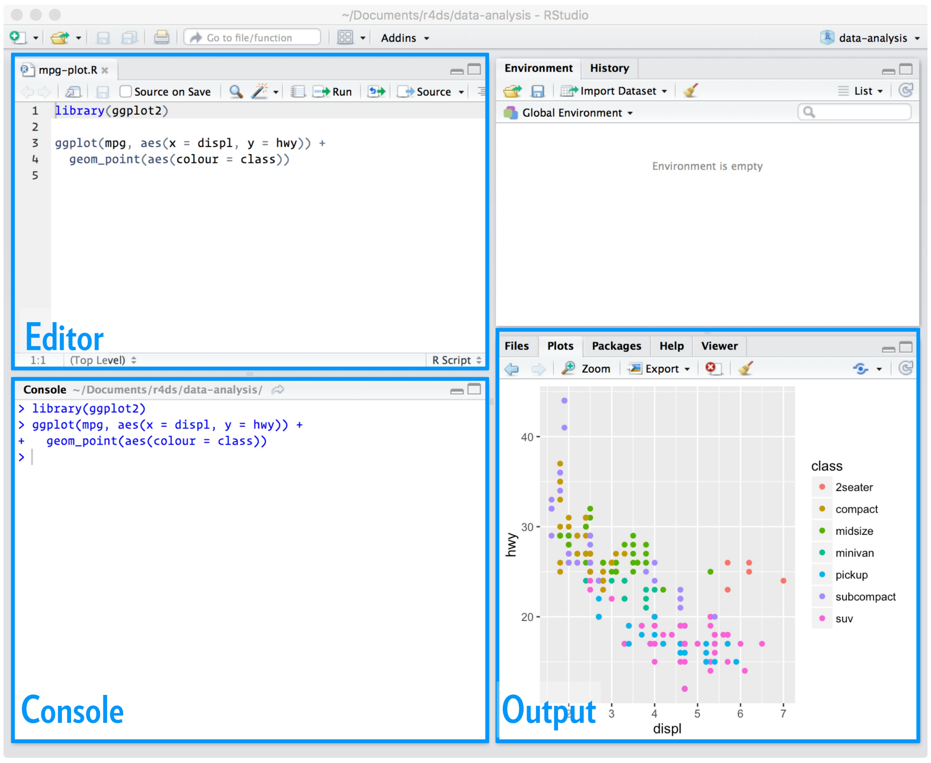
6 Workflow: scripts

So far you've been using the console to run code. That's a great place to start, but you'll find it gets cramped pretty quickly as you create more complex ggplot2 graphics and dplyr pipes. To give yourself more room to work, it's a great idea to use the script editor. Open it up either by clicking the File menu, and selecting New File, then R script, or using the keyboard shortcut Cmd/Ctrl + Shift + N. Now you'll see four panes:



```
library(ggplot2)

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = class))
```

```
> library(ggplot2)
> ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(aes(colour = class))
>
```

The script editor is a great place to put code you care about. Keep experimenting in the console, but once you have written code that works and does what you want, put it in the script editor. RStudio will automatically save the contents of the editor when you quit RStudio, and will automatically load it when you re-open. Nevertheless, it's a good idea to save your scripts regularly and to back them up.

6.1 Running code

The script editor is also a great place to build up complex ggplot2 plots or long sequences of dplyr manipulations. The key to using the script editor effectively is to memorise one of the most important keyboard shortcuts: Cmd/Ctrl + Enter. This executes the current R expression in the console. For example, take the code below. If your cursor is at █, pressing Cmd/Ctrl + Enter will run the complete command that generates `not_cancelled`. It will also move the cursor to the next statement (beginning with `not_cancelled %>%`). That makes it easy to run your complete script by repeatedly pressing Cmd/Ctrl + Enter.

```r
library(dplyr)
library(nycflights13)

not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(mean = mean(dep_delay))
```

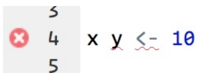
Instead of running expression-by-expression, you can also execute the complete script in one step: Cmd/Ctrl + Shift + S. Doing this regularly is a great way to check that you've captured all the important parts of your code in the script.

I recommend that you always start your script with the packages that you need. That way, if you share your code with others, they can easily see what packages they need to install. Note, however, that you should never include `install.packages()` or `setwd()` in a script that you share. It's very antisocial to change settings on someone else's computer!

When working through future chapters, I highly recommend starting in the editor and practicing your keyboard shortcuts. Over time, sending code to the console in this way will become so natural that you won't even think about it.

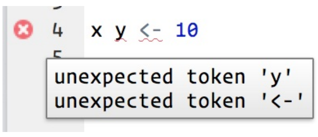
6.2 RStudio diagnostics

The script editor will also highlight syntax errors with a red squiggly line and a cross in the sidebar:



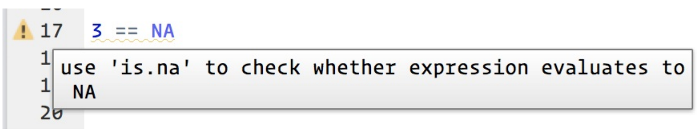
```
x y <- 10
```

Hover over the cross to see what the problem is:



```
x y <- 10
unexpected token 'y'
unexpected token '<-'
```

RStudio will also let you know about potential problems:



```
3 == NA
use 'is.na' to check whether expression evaluates to
 NA
```

6.3 Practice

1. Go to the RStudio Tips twitter account, https://twitter.com/rstudiotips and find one tip that looks interesting. Practice using it!

2. What other common mistakes will RStudio diagnostics report? Read https://support.rstudio.com/hc/en-us/articles/205753617-Code-Diagnostics to find out.