Ten Awesome Recent Developments in R

If you haven't been keeping on top of things recently, here are ten things you should know about

Keith McNulty 6 days ago · 8 min read ★

From Carl Heyerdahl via unsplash.com

Development in the R ecosystem has been moving at pace over the past year or so. New functionality in the base R language, in key R packages and in the RStudio IDE have made it easier for native R programmers like me to do their day-to-day work. Most of these developments have also made coding in R more pleasurable and enjoyable and in many cases have removed previous headaches or introduced functionality that aligns better with other programming languages.

If you've not been keeping up to date, here are ten developments that have excited me over the past year or so — in no particular order. Hopefully this will encourage you to experiment or re-engage with the language if it's been a while.

1. Native pipe

I'm starting with the most recent new development. Anyone who has had more than a fleeting introduction to R will be aware of the pipe operator `%>%` which is unique to R and which was introduced to make code more logical and readable. The pipe had become so natural to most R users that many were trying to use it immediately without realizing that it was not part of base R. Many, many times I would try to use it before installing `dplyr` or `magrittr`, and the resulting errors were becoming frustrating and annoying.

For a while now it has been acknowledged that the pipe needs

to be available in base R, and as of R version 4.1.0, that operator is available and is notated as `|>`. Native pipe will take the expression preceding it and place it into the function that follows it as the first argument. For example:

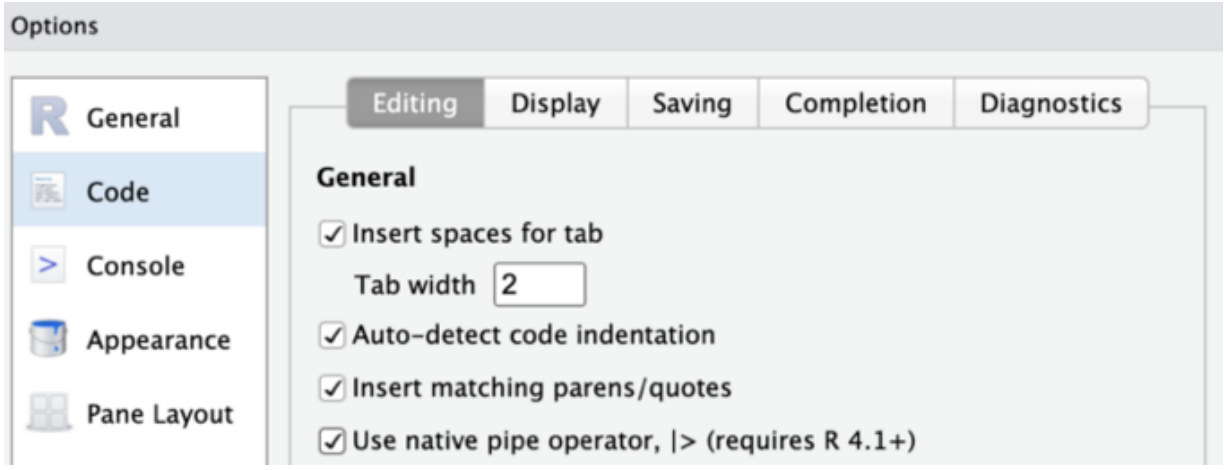
```
mtcars |>
  subset(subset = cyl == 4)
```

will subset `mtcars` down to just observations with `cyl == 4`. Note that, unlike with `%>%`, you cannot use `.` to pipe directly into other arguments of a function — so for example:

```
row.names(mtcars) |>
  grepl("Mazda", .)
```

will not work. To do this we need to use a function call (see number 2).

It is envisioned that the native pipe will replace the 'old' pipe in the long run. However this will take some further developments in the key packages that use the old pipe, so you'll need to be prepared to use both for a while. If you do want to start using the native pipe immediately, you can swap over the previous Cmd/Ctrl-Shift-M shortcut to code the new pipe by clicking the box under Global Options > Code in RStudio:



Swapping Cmd/Ctrl-Shift-M to the new native pipe in RStudio (Author generated image)

2. New anonymous function (Lambda-like) syntax in R

Another concurrent development in R 4.1.0 is a new syntax for writing anonymous functions that are intended for one-off usage, similar to Lambdas in Python. In general, R has previously been set up to encourage all functions to be named in memory. For example:

```
check_mazda <- function(x) {
  grepl("Mazda", x)
}
```

and then we would pipe into that function as follows:

```
mtcars |>
  row.names() |>
  check_mazda()
```

While there were always ways round this, the most recent

release formalizes a shorthand for anonymous functions as follows:

```
mtcars |>
  row.names() |>
  {\(x) grepl("Mazda", x)}()
```

This sets things nicely for mapping anonymous functions across data structures. For example:

```
mtcars |>
  lapply(\(x) mean(x) > 20)
```

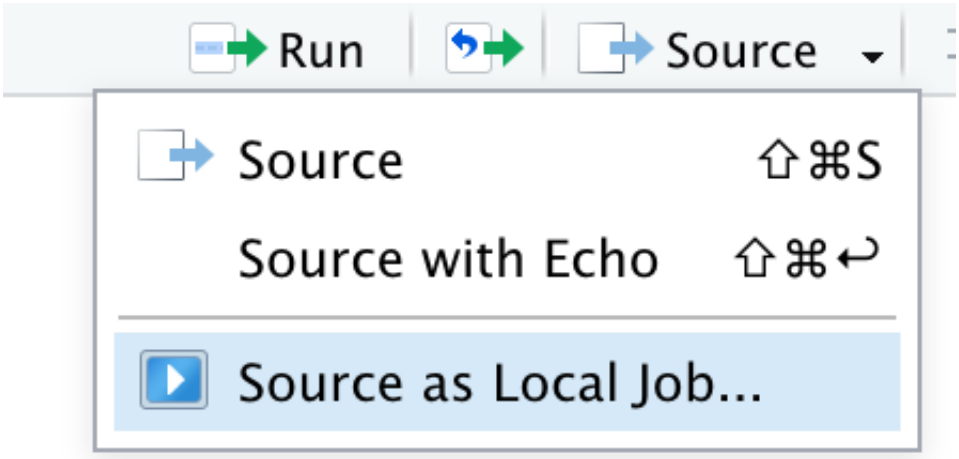
will test across all columns of `mtcars` whether the column has a mean greater than 20.

3. RStudio Jobs

Jobs allow you to multitask more effectively in RStudio. Previously, if you wanted to run lengthy R code and work on code at the same time, you'd have to launch two different instances of RStudio, which was a bit annoying. Now you can run your code as a Job and it will run in a separate R session within RStudio, allowing you to monitor it while working on something else.

If you are working in RStudio 1.2+, you can run a script as Job,

by using the pulldown Source menu.



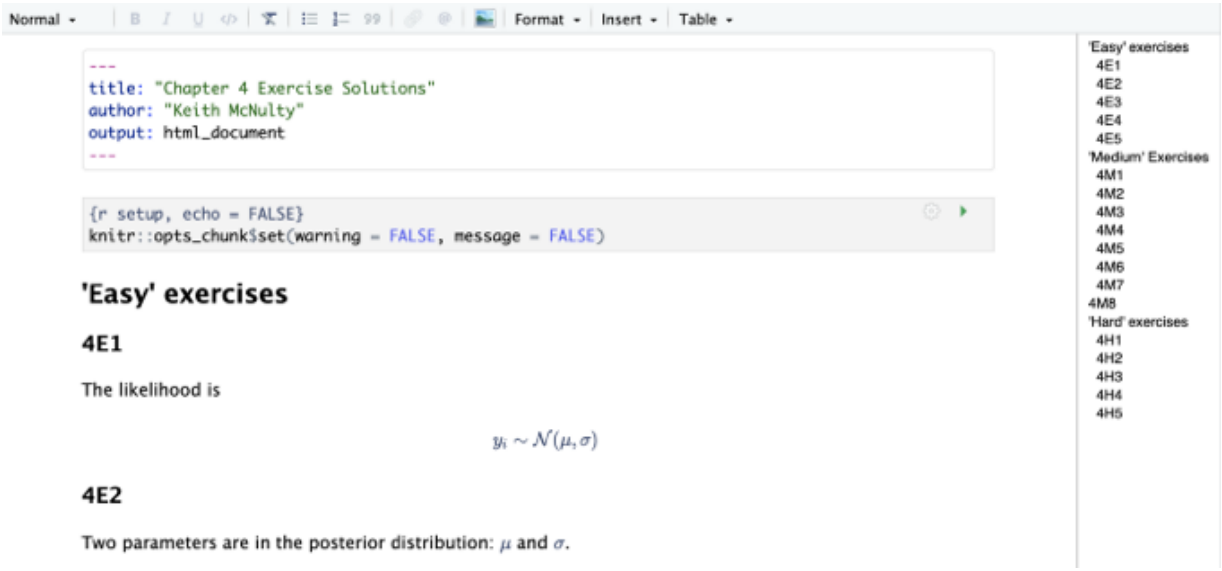
You'll need to make sure that your working directory is set up right and you'll need to choose where you want the results to be copied to. If your script doesn't write the results somewhere, and you choose not to copy them somewhere, then the process will execute without results, which is probably not what you wanted. You can also track the progress of the job using section comments in your script — you can learn more about that here.

4. R Markdown visual editor

R Markdown is absolutely awesome and my favourite way to publish data documents. And now as of RStudio 1.4 there is a visual editor for R Markdown, which is great for those who are not completely au fait with R Markdown syntax.

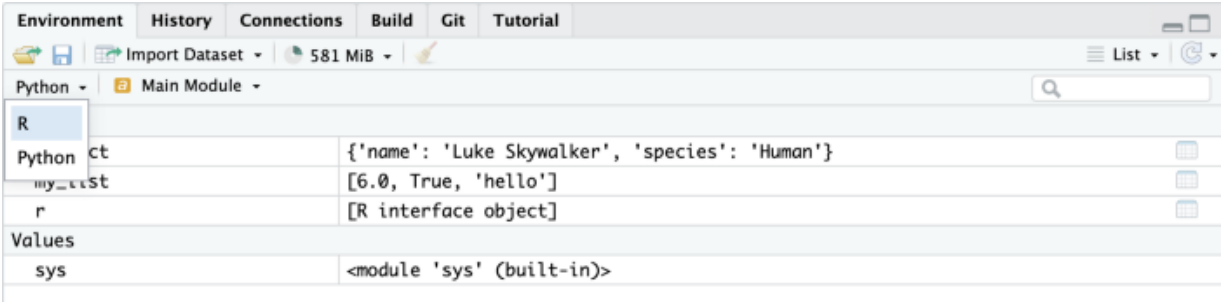
To edit your `Rmd` document using the visual editor, look for the

symbol in the top right of the document window. This will swap between the source editor and visual editor. Here's an example of what the visual editor looks like:



5. View and interact with your Python objects in RStudio

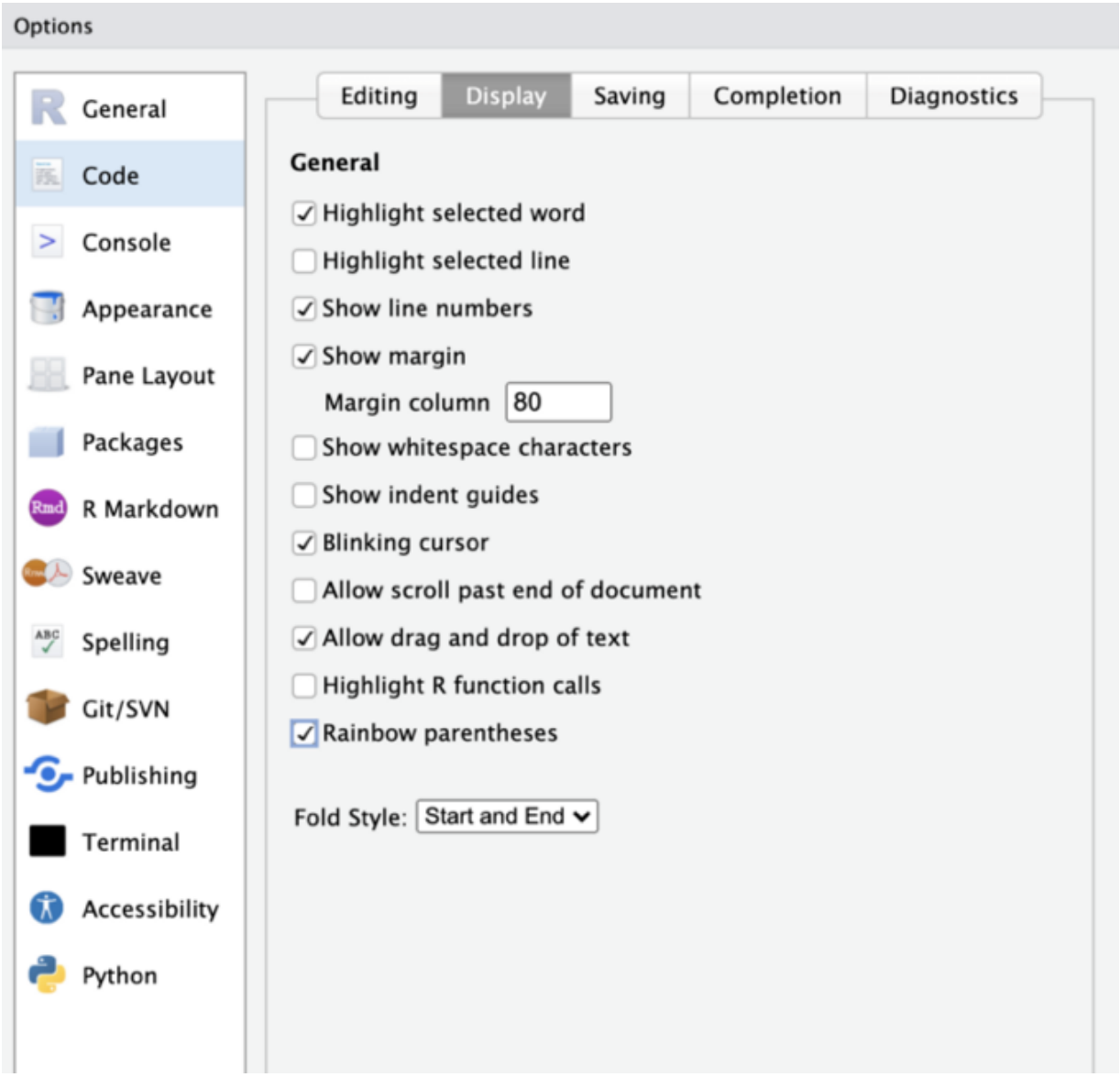
As of RStudio 1.4, whenever you work in both R and Python, you can easily swap between the two environments and see the objects in each. Assuming you have the `reticulate` package installed and loaded, you can view both your Python and R environments by clicking between them in the Environment pane:

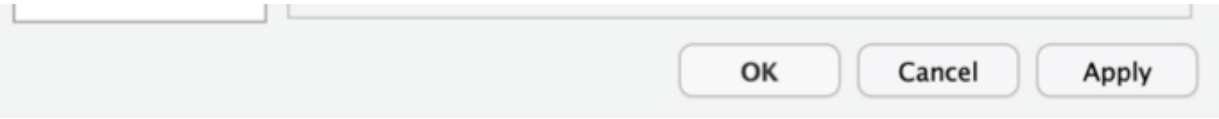


If you are interested in learning more about working with R and Python together in R Studio then see my other piece on this here.

6. Rainbow parentheses

This one is more cosmetic, but coloring your nested parentheses can help you troubleshoot your code so much more easily, especially where you have lots of nested plain, square or curly parentheses. This feature can now be enabled under Global Options > Code > Display:





7. List columns in dataframes

While not brand new, list columns have come into the limelight more since last year's major update of tidyverse packages `dplyr` and `tidyr`. Now columns of dataframes can be lists and not just vectors, which allows us the flexibility to put anything in dataframes, including models, graphics, other dataframes, whatever. This creates huge flexibility and opens up greater power for packages like `dplyr`. For example, you can now nest subset dataframes inside dataframes:

```
library(dplyr)
library(ggplot2)

mtcars |>
  nest_by(cyl)

# A tibble: 3 x 2
# Rowwise:  cyl
    cyl                data
  <dbl> <list<tibble[,10]>>
1     4           [11 × 10]
2     6            [7 × 10]
3     8           [14 × 10]
```

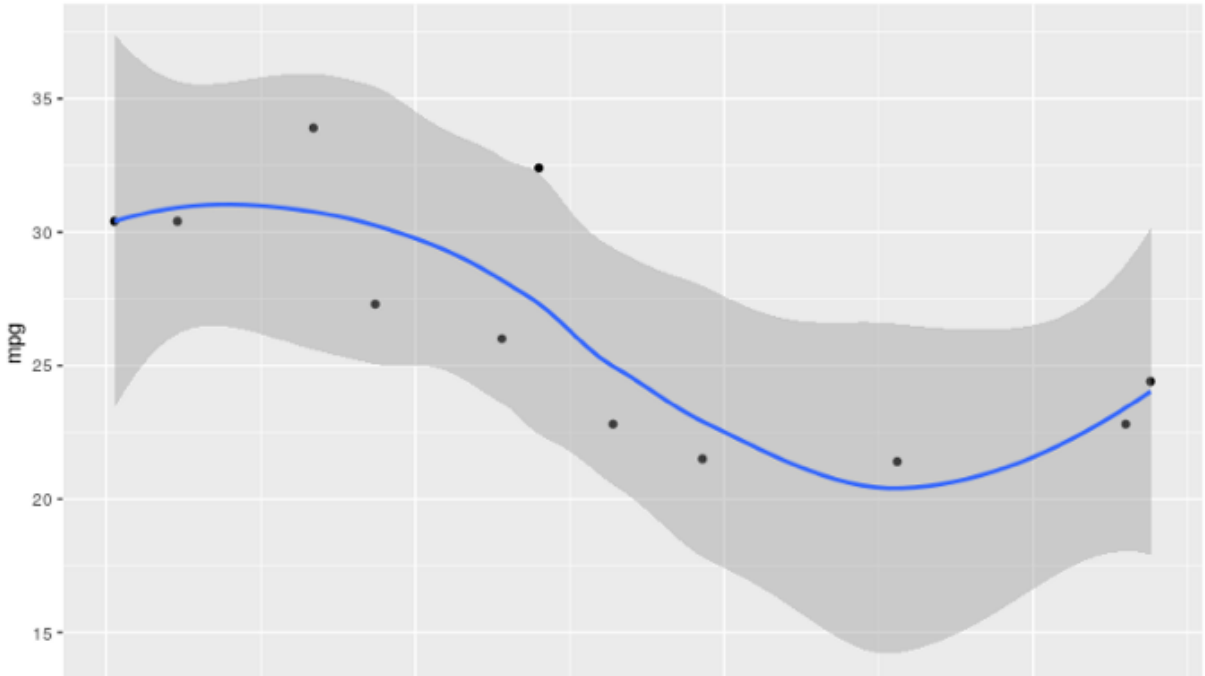
And then you can further mutate a column containing plots of each subset:

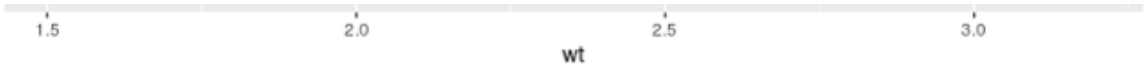
```
plotframe <- mtcars |>
  nest_by(cyl) |>
  mutate(plot = list(ggplot(data = data, aes(x = wt, y =
mpg)) +
           geom_point() +
           geom_smooth()))

# A tibble: 3 x 3
# Rowwise:  cyl
    cyl                 data plot
  <dbl> <list<tibble[,10]>> <list>
1     4           [11 × 10] <gg>
2     6            [7 × 10] <gg>
3     8           [14 × 10] <gg>
```

Now we have a list column containing dataframes and a list column containing plots. To see a specific plot you can just call an element of the list column:

```
plotframe$plot[1]
```





This can be extended to use dataframes to support large batch tasks — see an example of this in the production of batches of parameterized powerpoint documents in my piece here.

8. Pipelining your data science processes

The newly developed `targets` package provides a toolkit to automate and productionize your data science processes in R. This is particularly aimed at overcoming situations where these processes are built on interdependent R scripts that execute separately in sequence. This model is not ideal, and `targets` encourages programmers to set up their process in a pipeline of abstracted R functions instead of using scripts. The targets package will execute a pipeline of such functions intelligently and store intermediate and final outputs for examination.

When parameters change, `targets` knows which parts of your pipeline this affects and only executes the affected parts of the pipeline, using previously generated outputs for the parts that are not affected. This can save considerable time, and helps a great deal with bug-fixing. `targets` also contains useful functions like `tar_glimpse()` which can instantly create a visualization of your pipeline. More on `targets` here.

9. Expanded powers of abstraction in dplyr

`dplyr` is a package of massive significance in the R ecosystem,

and as of the release of version 1.0.0 last year, its power has substantially grown. In particular, many commonly loved functions have been abstracted to work across multiple columns at once, saving a lot of coding effort. In combination with `tidyselect` which allows you to select columns programatically, code can be massively shortened now using functions like `across` inside `summarise`. For example, where previously we may have written:

```
mtcars |>
  group_by(cyl) |>
  summarise(mean_disp = mean(disp),
            sd_disp = sd(disp),
            mean_drat = mean(drat),
            sd_drat = sd(drat))
```

We can now simply write:

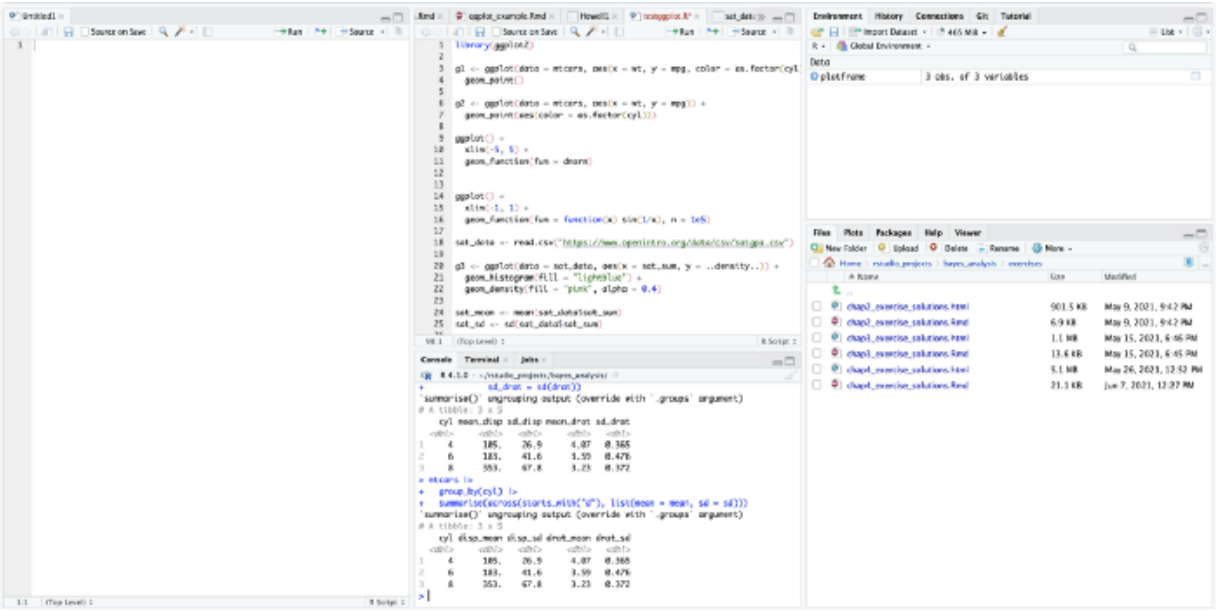
```
mtcars |>
  group_by(cyl) |>
  summarise(across(starts_with("d"), list(mean = mean, sd
= sd)))
```

I've written in detail on some of this amazing new abstracted functionality here.

10. Create more coding real estate in RStudio using Source Columns

If you like to have several source files open and viewable at the

same time, and you want more real estate for that without sacrificing some of your other panes, you can now do this as of RStudio 1.4 by simply adding source columns. You can do this through Global Options > Pane Layout and then choose the new 'Add Column' option. This will create a brand new Source pane to the left of your existing panes. You can even add more if you wish.



Here I've added an extra Source column on the left

These are just recent developments in R that I have made great use of. If you have others feel free to add them in the comments so that other readers can benefit.

Originally I was a Pure Mathematician, then I became a Psychometrician and a Data Scientist. I am passionate about applying the rigor of all those disciplines to complex people questions. I'm also a coding geek and a massive fan of Japanese RPGs. Find me on LinkedIn or on Twitter. Also check out my blog on drkeithmcnulty.com or my soon to be released textbook on People Analytics.