Rmd code chunk options visualized

This document is meant to help you visualize code chunk options in R Markdown. The behavior of the output changes depending on whether you "Knit" the document (to html, pdf, word, etc.) or use as an R Notebook (`output: html_notebook` in the header) or using the "Preview" button, so make sure to "Knit" the document.

The source code for this document may be accessed at https://github.com/jfrench/DataWrangleViz/blob/master/rmd-chunks-viz.Rmd. Knit the file (preferably to a pdf) to see visual examples of the options described below.

By default, the code to be evaluated (the source) is shown in a "source block" with a gray background in typewriter font and looks

```
"like this."
```

The evaluated code results (the output) are shown with `##` in front of it so that the output can be copied and pasted into the Console without causing problems. The evaluated code will be shown in an "output block" that looks

```
## [1] "like this."
```

Source evaluation (`eval` option)

Evaluating all lines of a code chunk

All code lines are evaluated (and shown by default) when `eval = TRUE`.

````
```{r, eval = TRUE}
1 + 1
```
````

```
1 + 1
```

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

Not evaluating a code chunk

No code lines are evaluated when `eval = FALSE`. The code simply passes through the Console without being evaluated.

````
```{r, eval = FALSE}
1 + 1
```
````

```
1 + 1
```

Evaluating selected lines of a code chunk

Selected code lines can be evaluated.

The results with `eval = 1:2`. The first and second lines are evaluated. The third line simply passes through the Console.

````
```{r, eval = 1:2}
"first line"
"second line"
"third line"
```
````

```
"first line"
```

```
## [1] "first line"
"second line"

## [1] "second line"
## "third line"
```

The results with `eval = -c(1, 3)`. All lines but the first and third lines are evaluated.

````
```{r, eval = -c(1, 3)}
"first line"
"second line"
"third line"
```
````

```
## "first line"
"second line"

## [1] "second line"
## "third line"
```

Printing the code to be evaluated (`echo` option)

Print all evaluated code

The code to be evaluated is printed when the option `echo = TRUE`.

````
```{r, echo = TRUE}
1:3
```
````

```
1:3

## [1] 1 2 3
```

Print no evaluated code

The code to be evaluated is NOT printed when the option `echo = FALSE`, but the results are still shown (assuming `eval = TRUE`).

````
```{r, echo = FALSE}
1:3
```
````

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

Print some evaluated code

Print the first two lines of evaluated code using `echo = 1:2`. The results from evaluating the third line are shown, but the source block for that line is not printed.

````
```{r, echo = 1:2}
1:3
4:6
7:9
```
````

```
1:3

## [1] 1 2 3
```

```
4:6
```

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

```
## [1] 7 8 9
```

Print all but the first line of evaluated code using `echo = -1`. The source blocks for all but the first line of the code chunk are printed.

````
```{r, echo = -1}
1:3
4:6
7:9
```
````

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

```
4:6
```

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

```
7:9
```

```
## [1] 7 8 9
```

Formatting the output block (`results` option)

Markup

If you specify `results = 'markup'`, then the output block looks like regular output from the Console.

````
```{r, results = 'markup'}
1:3
```
````

```
1:3
```

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

Regular text

If you specify `results = 'asis'`, the output block is printed as plain text.

````
```{r, results = 'asis'}
1:3
```
````

```
1:3
```

[1] 1 2 3

All results together (not interweaved with evaluated code)

If you specify `results = 'hold'`, then the source block is shown in its entirety, followed by the output block in its entirety, instead of being interweaved as the code is evaluated.

````
```{r, results = 'hold'}
1:3
4:6
```
````

```
1:3
4:6
```

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

Hide results

If you specify `results = 'hide'`, then the output block is not shown.

````
```{r, results = 'hide'}
1:3
```
````

```r
1:3
```

Collapse (combining the source and output block)

Specifying `collapse = TRUE` combines the source and output blocks into a single block.

````
```{r, collapse = TRUE}
1:3
```
````

```r
1:3
## [1] 1 2 3
```

Code styling

Highlighting syntax

Consider the following source blocks when `highlight = TRUE` versus `highlight = FALSE`. The first source block is colored in various ways, while the second is black text.

````
```{r}
add <- function(a, b) { a + b }
```
````

```r
add <- function(a, b) { a + b }
```

````
```{r, highlight = FALSE}
add <- function(a, b) { a + b }
```
````

```
add <- function(a, b) { a + b }
```

Characters before output block

The `comment` option changes the text immediately before an output block. In the first block below, we use the default style (`##`), while in the second block we change it to something more similar to the Console output (`#>`). Note: The `#` is important to make the output easier for users to copy and paste into the Console.

````
```{r}
1 + 1
```
````

```r
1 + 1
```

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

````
```{r, comment = '#>'}
1 + 1
```
````

```r
1 + 1
```

```
#> [1] 2
```

Automatically reformatting text

There are too many options to list here, but here is an example using `blank` and `width.cutoff` to reformat the R out.

Compare the results of the following two source blocks when printed. Notice that in the first, the source block code is allowed to run off the page and blank lines are preserved. In the second, the code is automatically wrapped after 30 characters and the blank line is removed.

````
```{r}
# This is an outrageously long comment. For no real reason. Except the fact that I need to illustrate a

mean(1:3) # Blank line above
```
````

```r
# This is an outrageously long comment. For no real reason. Except the fact that I need to illustrate a

mean(1:3) # Blank line above
```

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

````
```{r, tidy = TRUE, tidy.opts = list(blank = FALSE, width.cutoff = 30)}
# This is an outrageously long comment. For no real reason. Except the fact that I need to illustrate a

mean(1:3) # Blank line above
```
````

```r
# This is an outrageously long
# comment. For no real reason.
# Except the fact that I need
# to illustrate a point.
mean(1:3)  # Blank line above
```

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

Plots

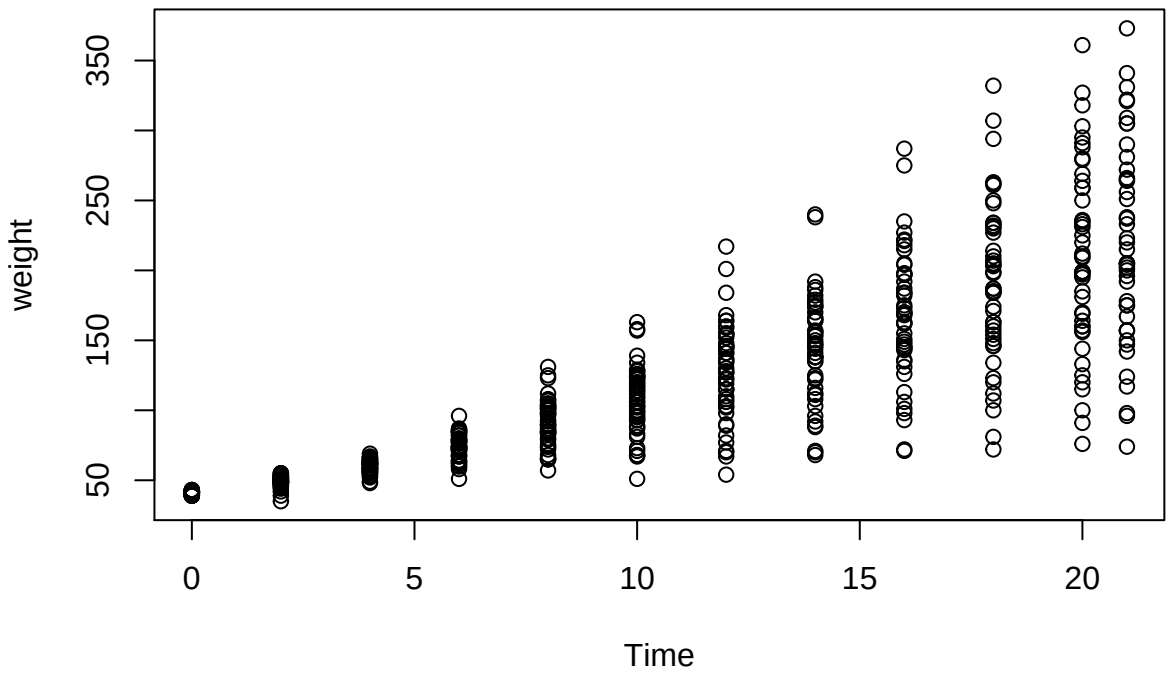
We'll demonstrate some of the plot-related options using the `ChickWeights` data set in the `datasets` package (installed by default in R).

Order of plots (`fig.show` option)

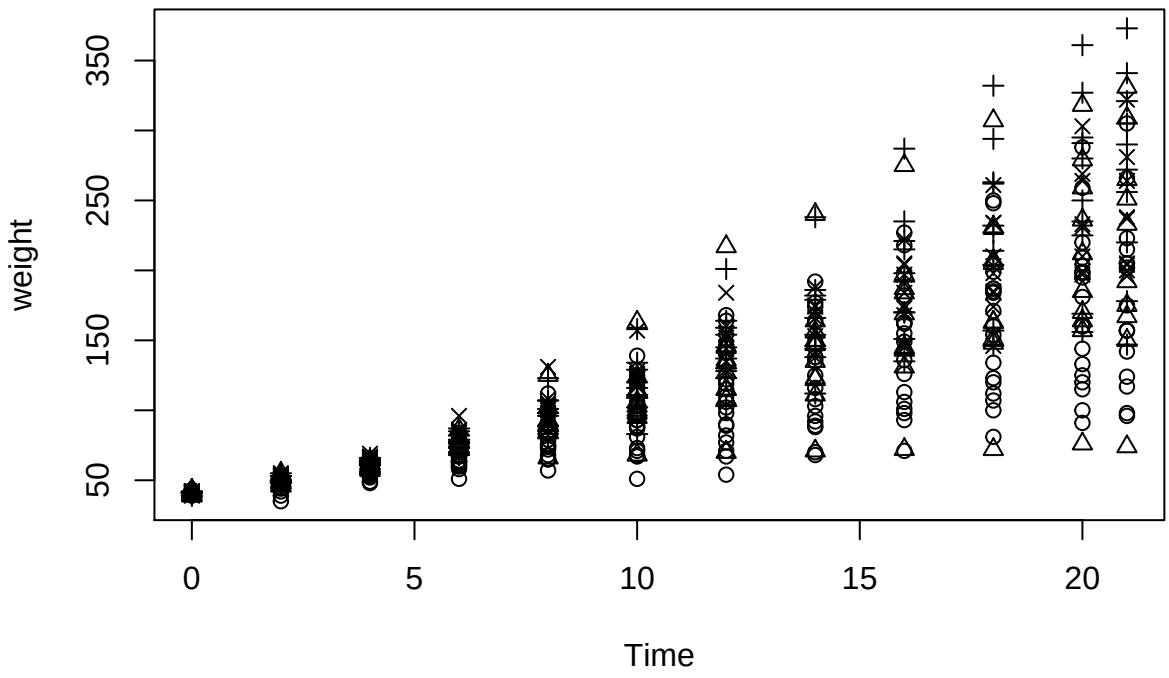
Not specifying the `fig.show` option directly will produce the plots in the same way they'd be produced in the Console.

````
```{r}
plot(weight ~ Time, data = ChickWeight)
plot(weight ~ Time, data = ChickWeight, pch = as.numeric(ChickWeight$Diet))
```
````

```r
plot(weight ~ Time, data = ChickWeight)
```




```
plot(weight ~ Time, data = ChickWeight, pch = as.numeric(ChickWeight$Diet))
```



If we specify `fig.show = 'hide'`, then the plots will be hidden from the output.

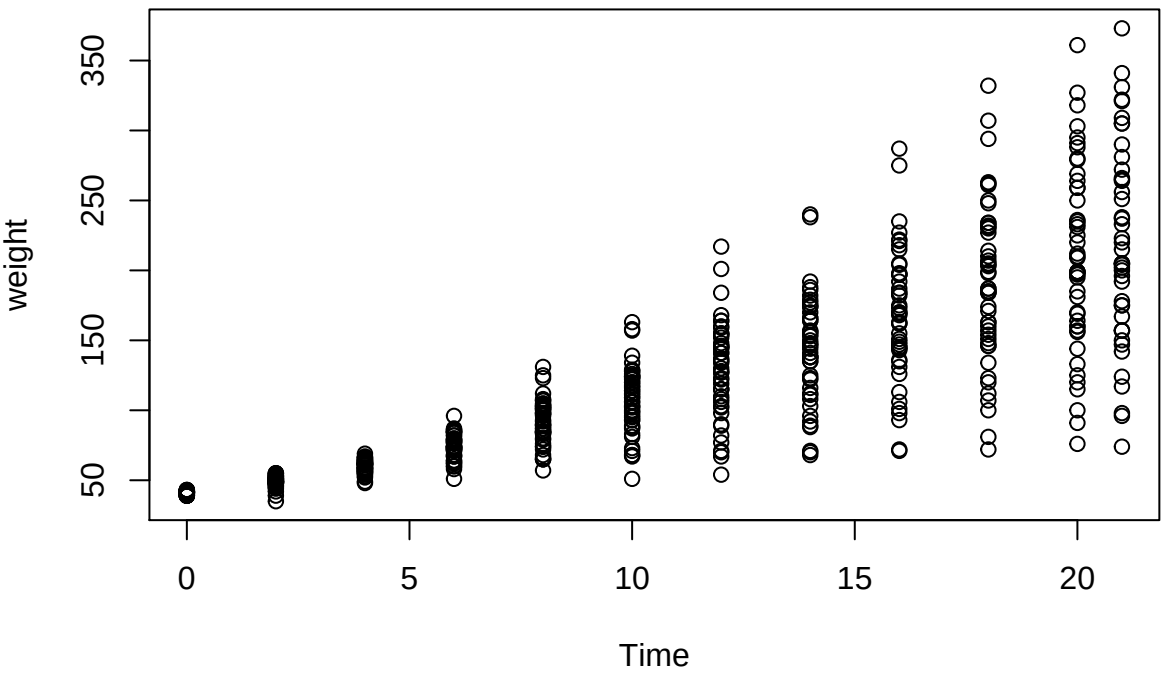
```
```{r, fig.show = 'hide'}
plot(weight ~ Time, data = ChickWeight)
```
```

```
plot(weight ~ Time, data = ChickWeight)
```

If we specify `fig.show = 'hold'`, then the plots are held until the end of the code chuck and produced consecutively.

```
```{r, fig.show = 'hold'}
plot(weight ~ Time, data = ChickWeight)
plot(weight ~ Time, data = ChickWeight, pch = as.numeric(ChickWeight$Diet))
```
```

```
plot(weight ~ Time, data = ChickWeight)
plot(weight ~ Time, data = ChickWeight, pch = as.numeric(ChickWeight$Diet))
```


weight
50
150
250
350
0
5
10
15
20
Time

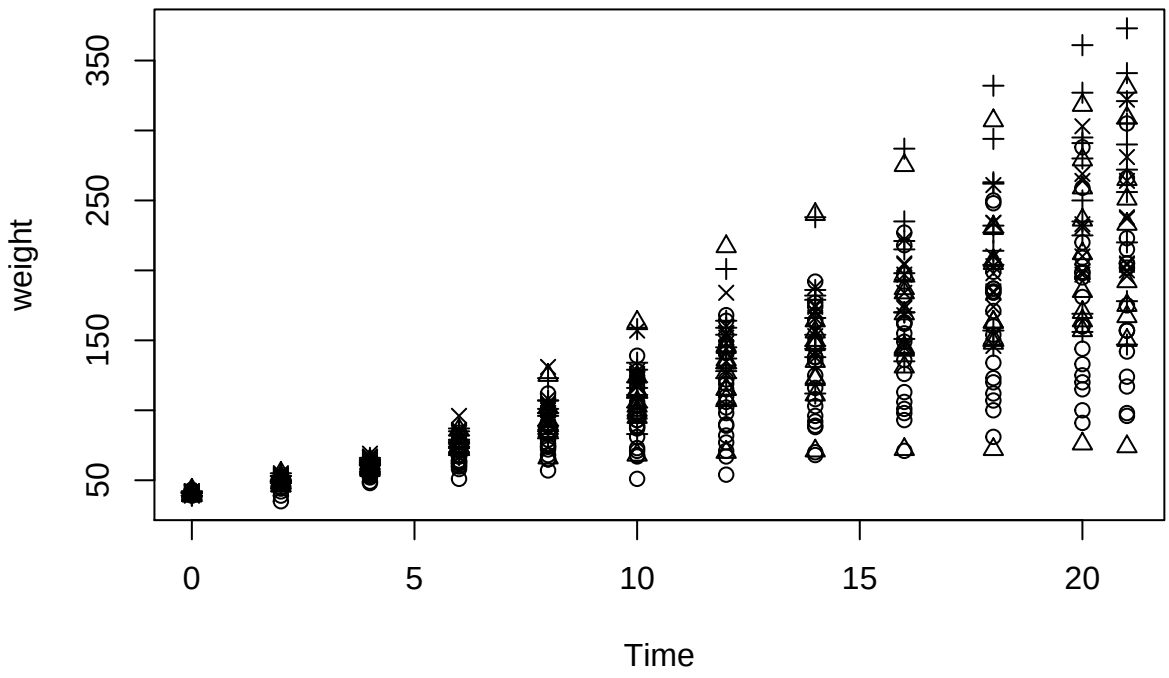


Figure alignment, captions, and size

We'll demonstrate how to align figures, change the caption, and change the size with the example below, with the relevant option name being obvious.

```
```{r, fig.align='center', fig.cap = 'Fig 1: Figure duex?', fig.height = 4, fig.width = 4}
plot(weight ~ Time, data = ChickWeight)
```
```

```r
plot(weight ~ Time, data = ChickWeight)
```

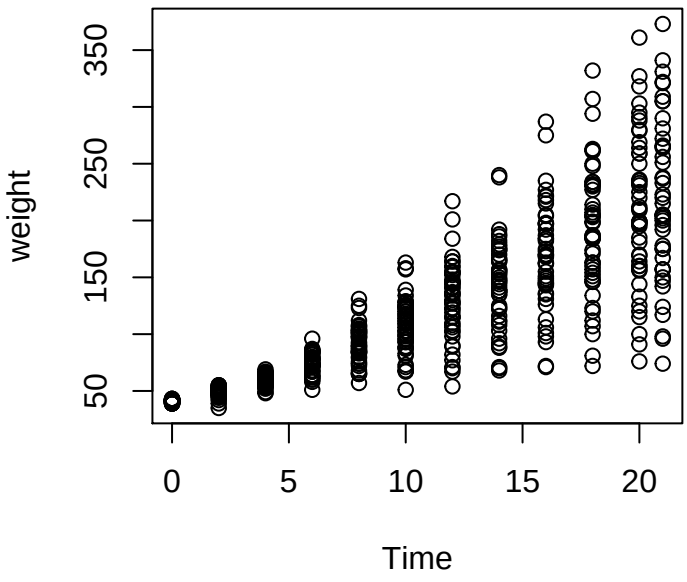


Figure 1: Fig 1: Figure duex?