Chunk Option Showcase

Zach Bogart

9/21/2018

Code Chunks

Let's start with the chunk seen above (or rather, not seen, if you have knit this file). Every chunk is a way to tell R Markdown this is R code and we want you to execute it. It is delimited by triple back-ticks and starts with a curly bracket pair. Within the pair, the `r` says "this is R code". Incidentally, you can create a code chunk with different languages (for example, `{python}` creates a python code chunk. See more languages in the `Insert` drop-down.)

There's also a word `setup` separated from the `r` by a space. This is the *name* of the code chunk. Naming your code chunks will help you if you have a long file and want to identify potential problems or navigate your file easily. At the bottom of the text editor in R Studio, you will see a line showing the character and line numbers followed by a clickable drop-down. Chunks by default have no name and will appear as "Chunk n", where n is the nth chunk. This drop-down will show you the section of the document you are currently editing, including headers. For example, as I write this, my drop-down says `Code Chunks` with a orange icon since that is the header I am under. If I click into the code chunk above, it switches to "Chunk 1: setup" and a green icon. Something to keep in mind when your files get large. For a chat about this with pictures, checkout Markdown etiquette from the iris walkthrough.

Following the name is a comma and `include=FALSE`. This is a **chunk option**. After the `r` denoting the code chunk language, everything is comma separated. These chunk options allow you to specify things about how the code chunk should behave. Below I will go into more detail about chunk options.

Global Options

The chunk at the top of this page is special. It has a call of `knitr::opts_chunk$set()`, which sets *global settings* for all code chunks. This means every chunk will have the settings specified unless overwritten. Within this call, there is `echo = TRUE` meaning every code chunk will be included in the output file. Since code chunks are numbered, you can globally specify which chunks are shown using an interval (`echo=2:5`; 2 through 5 inclusive are included in output) or even by exclusion (`echo=-7`; include all chunks except the seventh).

Note: Wondering why the global setting doesn't apply to the code chunk it's in? It does, but they add the chunk option `include-FALSE` to that chunk, meaning the chunk will be executed, but it will not be included in the output document.

Examples

There are a bunch of chunk options, so I will not cover them all here. But I will show you common ones. There are super cool things you can do and I strongly recommend you checkout the knitr options page to see all the possible options.

A very important note about using chunk options: **tab-completion is your best friend**. For example, you can start typing `fig.align`, tab-complete, and see the possible values it can take ('default', 'left', 'right', 'center').

Now for some examples. We will start with a simple unnamed code chunk:

```r
library(ggplot2)
library(dplyr)
```

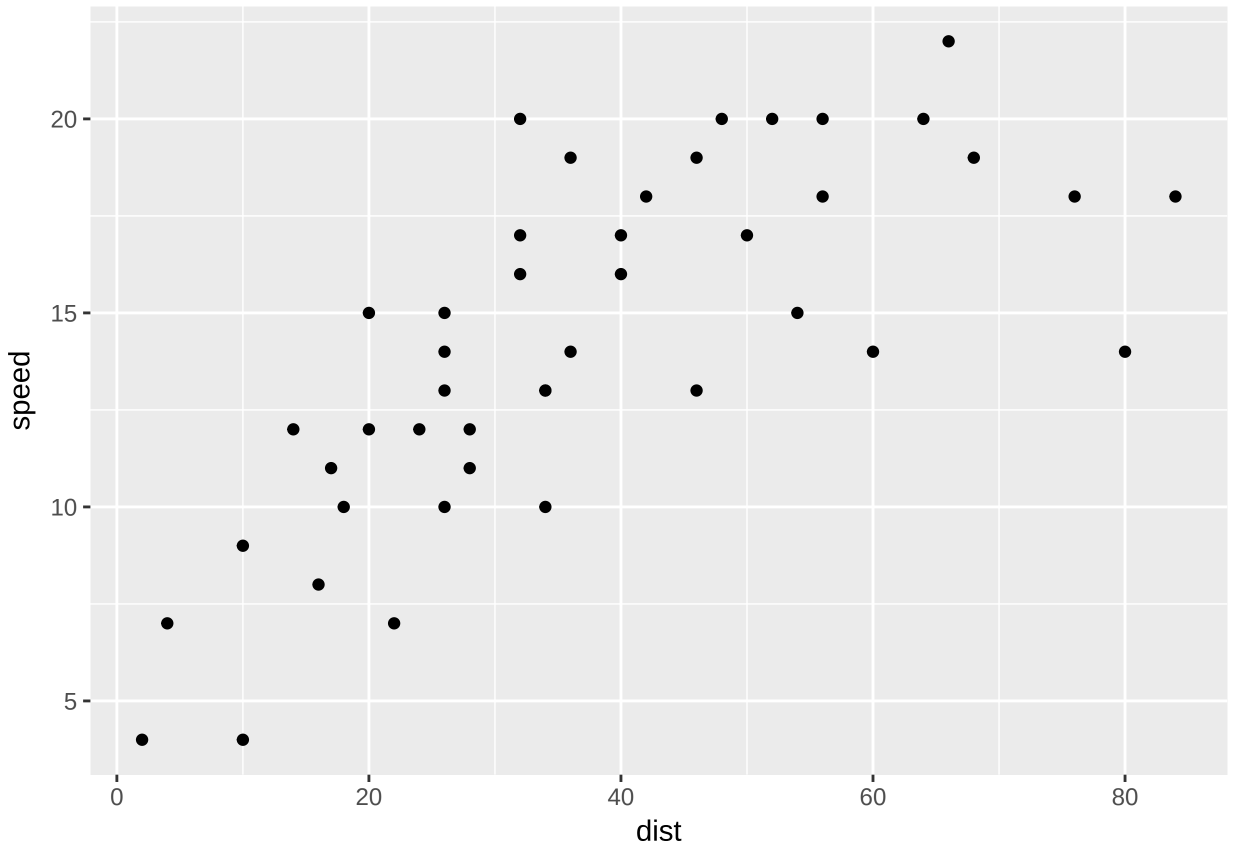
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

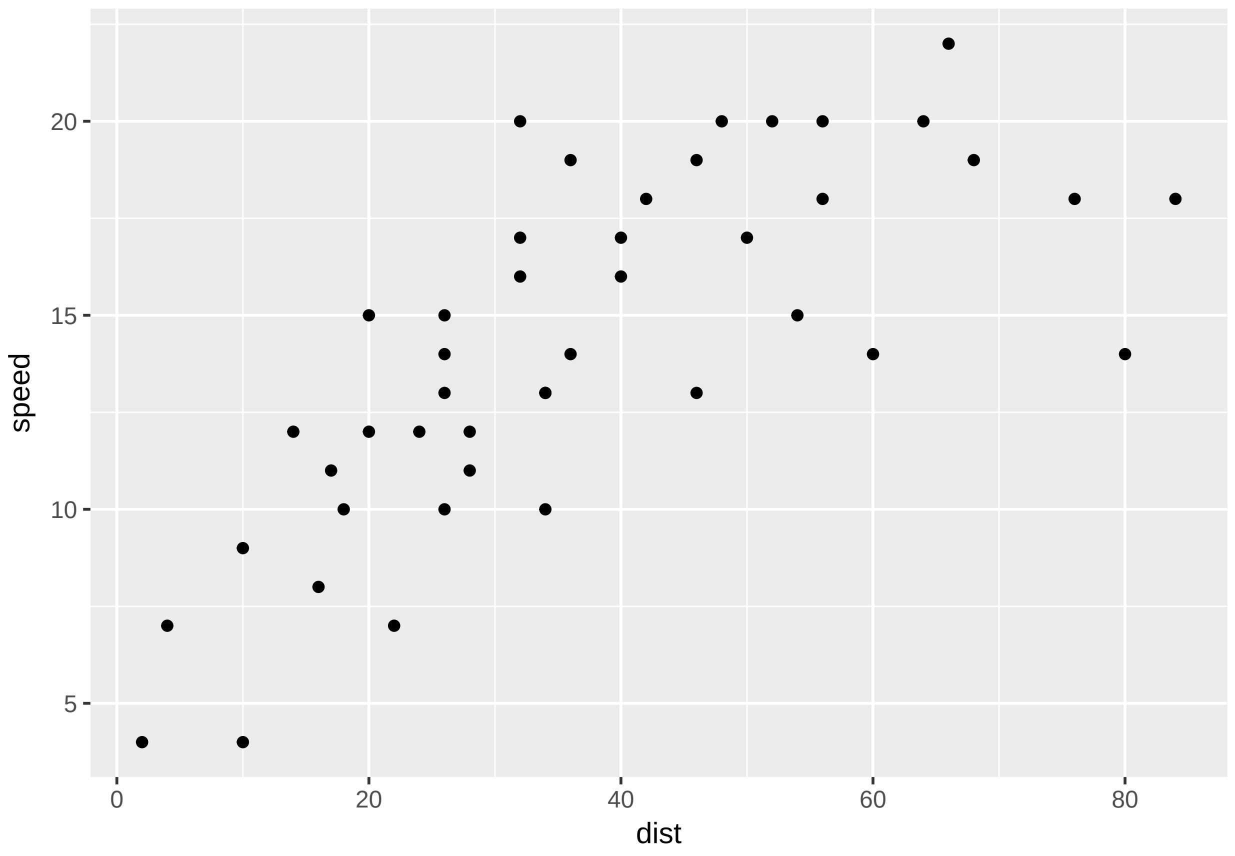
```r
cars_mod <- cars %>%
  filter(speed < 23)

ggplot(cars_mod, aes(dist, speed)) +
  geom_point()
```



echo=FALSE

Notice that this code chunk has no chunk options, but if you knit the document, you will see the code chunk printed. This is because of that global `echo=TRUE`. We can overwrite this to print just the graph without the code chunk that creates it:



So now only the graph appears.

message=FALSE

We still have this annoying message from `dplyr` informing us about some function masking. To suppress this, use `message`:

```
library(ggplot2)
library(dplyr)

cars_mod <- cars %>%
  filter(speed < 23)

ggplot(cars_mod, aes(dist, speed)) +
  geom_point()
```

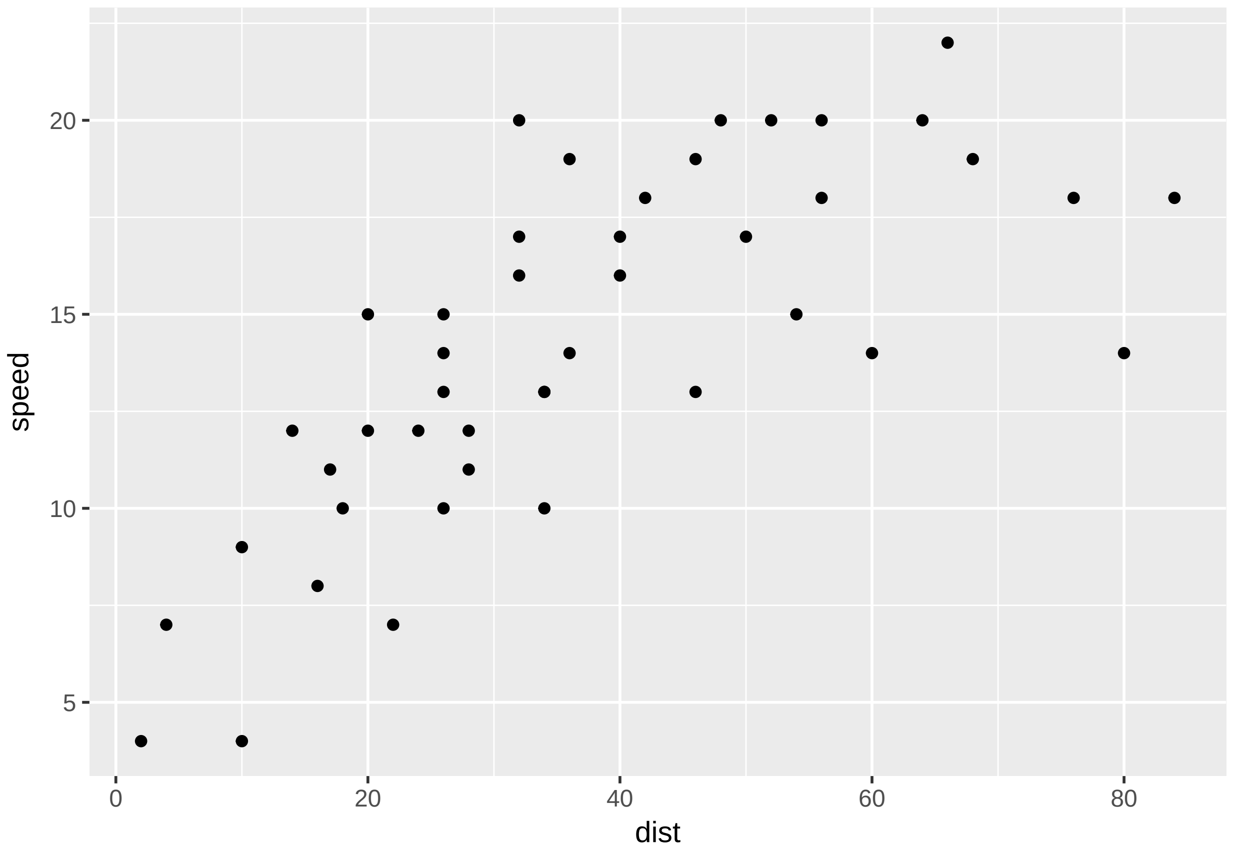


fig.height and fig,width

This plot is pretty big. Let’s make it smaller with `fig.height` and `fig.width`:

```r
library(ggplot2)
library(dplyr)

cars_mod <- cars %>%
  filter(speed < 23)

ggplot(cars_mod, aes(dist, speed)) +
  geom_point()
```

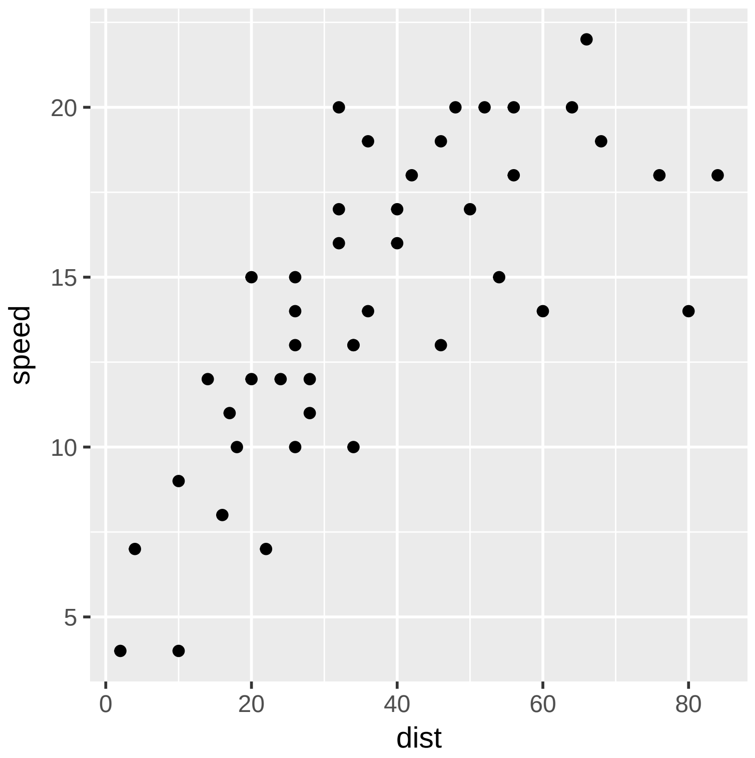


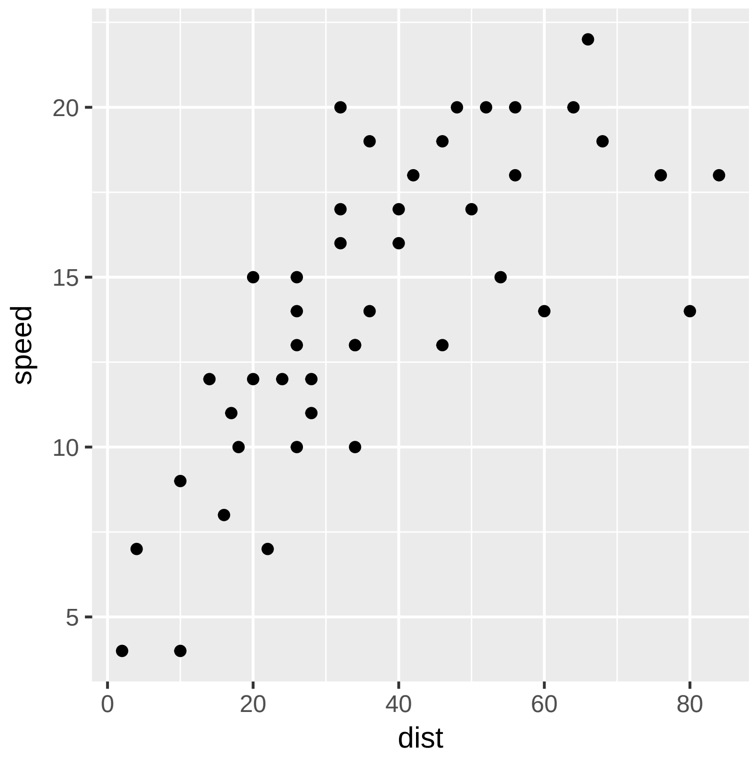
fig.align

Now it's noticeably left aligned, but I want it centered. Use `fig.align`:

```r
library(ggplot2)
library(dplyr)

cars_mod <- cars %>%
  filter(speed < 23)

ggplot(cars_mod, aes(dist, speed)) +
  geom_point()
```



Helful Items

cache

If you find your document takes a long time to generate and you keep tweaking small things and re-rendering, consider using `cache=TRUE` to skip re-rendering things that haven't changed between renders (stores them locally).

fig.dim

Annoyed by writing out `fig.height` and `fig.width`? You can use `fig.dim`, which specifies both. For example, `fig.dim = c(5, 6)` is the same as `fig.width = 5, fig.height = 7`.

aliases

Annoyed writing out chunk options in general? You can name aliases for chunk options as a global: `aliases = c(h = 'fig.height', w = 'fig.width')`. Now you can use `h` and `w` as you would `fig.height` and `fig.weight`, respectively.

Explore

There are a bunch of chunk options to explore. This showcase is just a small selection. Checkout the the knitr options page for documentation of every chunk option. I know I have referenced this several times now, but it deserves multiple mentions. It is the first thing I go to when I need to explore or try out a chunk option.