The art of flipbooking (building code movies)

With flipbookr and xaringan

Gina Reynolds, December 2019

Welcome

- Yes, there is now a package called `flipbookr`

Welcome

- Yes, there is now a package called `flipbookr`
- Yes, it's still under construction (the title of this book used to be the "fragile perilous art")

Welcome

- Yes, there is now a package called `flipbookr`
- Yes, it's still under construction (the title of this book used to be the "fragile perilous art")
- Yes, you can help make it less fragile and perilous by letting us know if/how it breaks and frustrates you and/or contributing at https://github.com/EvaMaeRey/flipbookr!

Spread the word and giving feedback

Please help us spread the word about flipbooks. Let your audience know *how* you created your flipbook with a quick acknowledgement, for example, *The flipbooked portion of this presentation was created with the new {flipbookr} package. Get it at remotes::install_github("EvaMaeRey/flipbookr")*

Also consider sharing your work on social media, and let me know what you've built on Twitter with a mention to @EvaMaeRey

Feedback? Contributions? Leave an issue at: https://github.com/EvaMaeRey/flipbookr

"Flipbooks"?

“Flipbooks” are tools that present side-by-side, aligned, incremental code-output evolution via automated code parsing and reconstruction. More about Flipbooks here. There now exists a package for making Flipbooks for R: `flipbookr`. This is under development, but you are welcome to try it out by installing from github:

```
devtools::install_github("EvaMaeRey/flipbookr")
```

You can see the template that was used to build this flipbook that you are looking at right now here.

Or, once you install the package (and restart RStudio?) a template for making the flipbook that you are looking at will also be available from within RStudio, File -> New File -> R Markdown -> From Template -> "A Minimal Flipbook".

How Flipbooking with Xaringan works

The flipbook you will be building here uses a member of the rmarkdown family called Xaringan (presentation ninja), which creates a slideshow in html. Dynamic documents like `rmarkdown` documents allow you to comingle code and prose in a single document.

It may be obvious by now, if you are following along with the source template, that slide breaks are indicated with `---` (be careful trailing white space is not allowed)

Flipbooks are built by spawning new *partial* code chunks from a single, user-input code chunk. The partial code chunks build up and are display consecutively in a slide show alongside its output which yields a movie-like experience; this should make each step easier to understand.

As you begin with flipbooks, I'd recommend using the code chunk option `include = F` for your "source" code chunks, and with no caching throughout. As you begin to get more comfortable with flipbooking, you might change these choices.

Set-up

We use the flipbookr package, of course! This does the work of disassembling a single code chunk and creating the "build" of multiple partial-code chunks. This is at the top of this file in the "setup" code chunk.

Also, at the top of this template in that "setup" code chunk, I set *code chunk* options for the code chunks that follow. These will apply to the spawned code chunks.

```
Error in file(filename, "r", encoding = encoding) :
  cannot open the connection
```

```
Error in file(filename, "r", encoding = encoding) :
  cannot open the connection
```

Using `flipbookr::chunk_reveal()`

You will use the `chunk_reveal()` function inline to generate the derivitive code chunks, rather than inside of a code chunk, so that the text that is generated is interpreted correctly when rendered. The inline code will look something like this:

```
`r chunk_reveal(chunk_name = "cars", break_type = "user")`
```

There are several modalities that you might be interested in using for "flipbookifying" your code and the next section is dedicated to demoing some of them below.

- **break type** -- *which lines of code should be revealed when*, `break_type` defaults to "auto"
- **display type** -- *display code and output, or just output, or just code?*, `display_type` defaults to "both"
- **assignment type** -- *does code chunk use left assignment?*, `left_assign` defaults to FALSE

At first we'll apply our flipbooking to the below input code - the code chunk is named "cars". For now I set echo = TRUE for this code chunk, so you can see the code content but sometimes you might like to set echo to FALSE. This code uses tidyverse tools, so we loaded that too in the "setup" code chunk at the beginning of the template.

`break_type`

Notice the regular comments and the special #BREAK comments, these will be used for a couple of the different "break type" modalities.

```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) + #BREAK
  geom_point(
    alpha = .3, #BREAK2
    color = "blue" #BREAK3
    ) + #BREAK
  aes(size = speed) #BREAK
```

break_type = "auto"

One parameter of flipbooking is the break_type. The default is "auto", in which appropriate breakpoints are determined automatically --- by finding where parentheses are balanced.

```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) + #BREAK
  geom_point(
    alpha = .3, #BREAK2
    color = "blue" #BREAK3
    ) + #BREAK
  aes(size = speed) #BREAK
```

```
cars
```

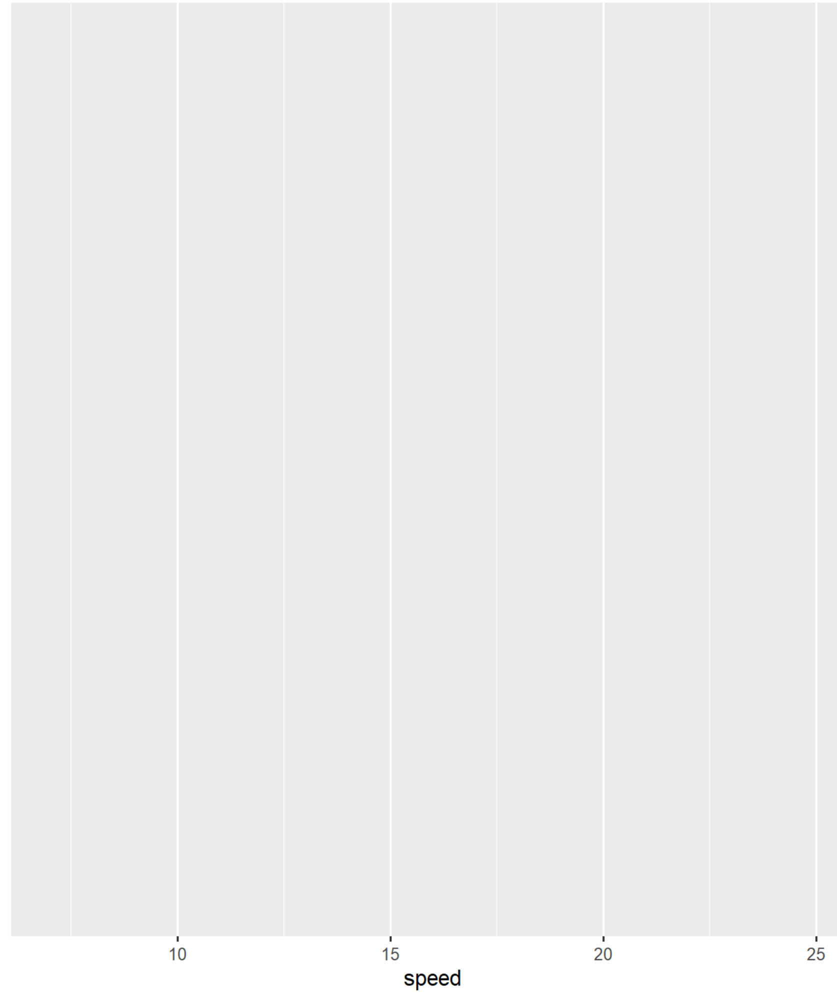
```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```r
cars %>%
  filter(speed > 4)
```

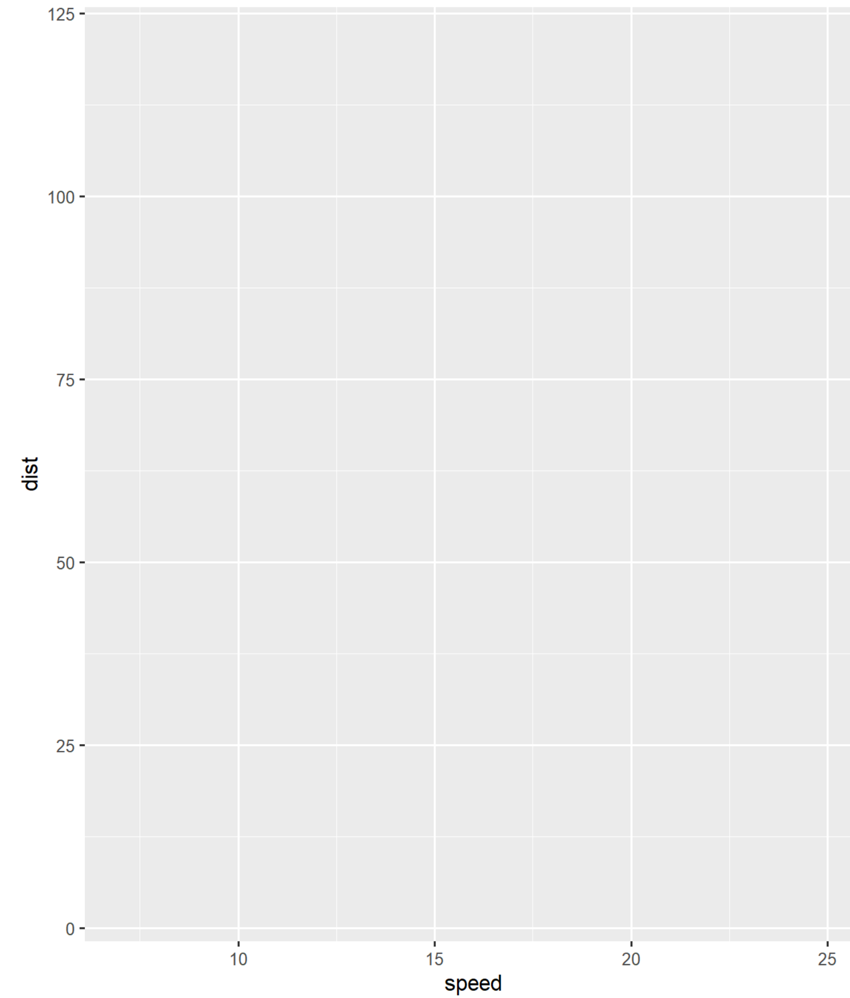
```
   speed dist
1      7    4
2      7   22
3      8   16
4      9   10
5     10   18
6     10   26
7     10   34
8     11   17
9     11   28
10    12   14
11    12   20
12    12   24
13    12   28
14    13   26
15    13   34
16    13   34
17    13   46
18    14   26
19    14   36
20    14   60
21    14   80
22    15   20
23    15   26
24    15   54
25    16   32
26    16   40
27    17   32
28    17   40
29    17   50
30    18   42
31    18   56
32    18   76
```

```r
cars %>%
  filter(speed > 4) %>%
  ggplot()
```

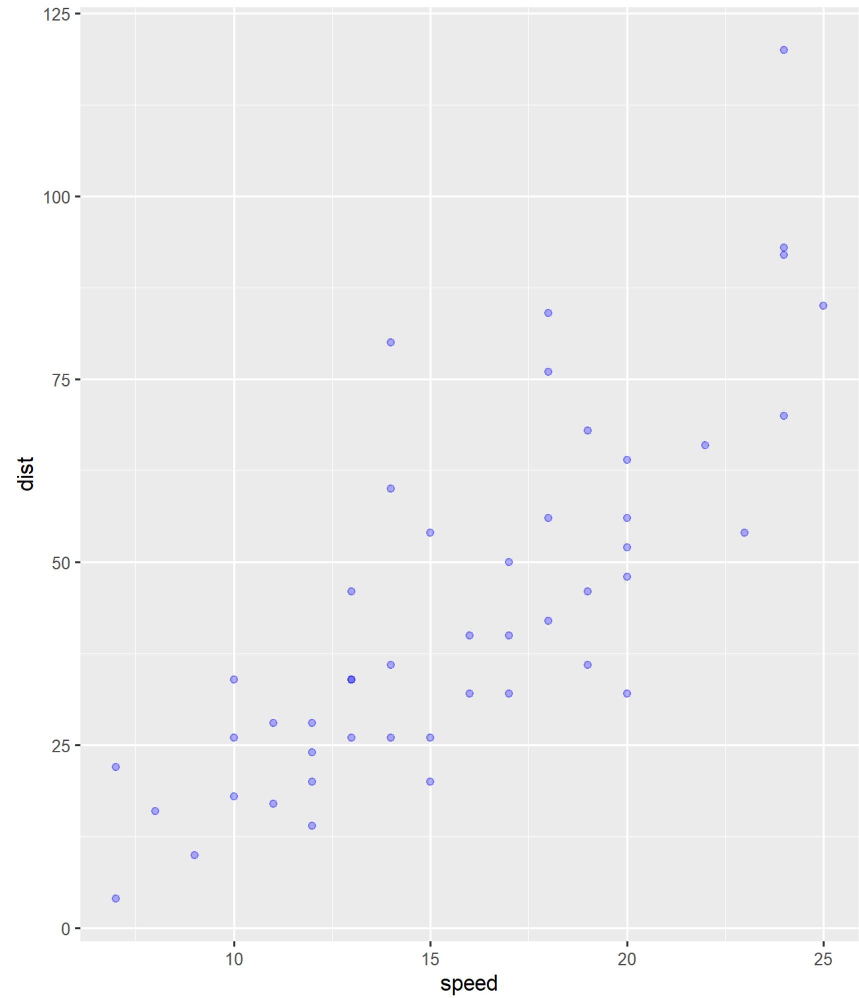
```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed)
```



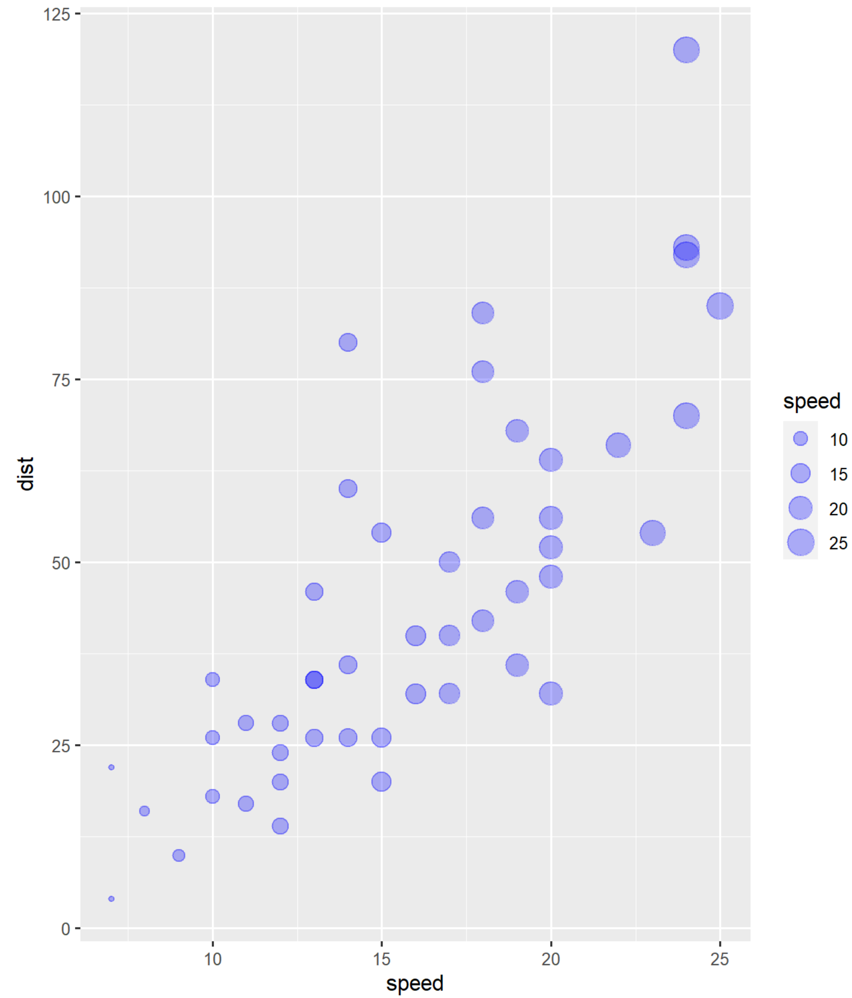
```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist)
```



```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    )
```



```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    ) +
  aes(size = speed)
```

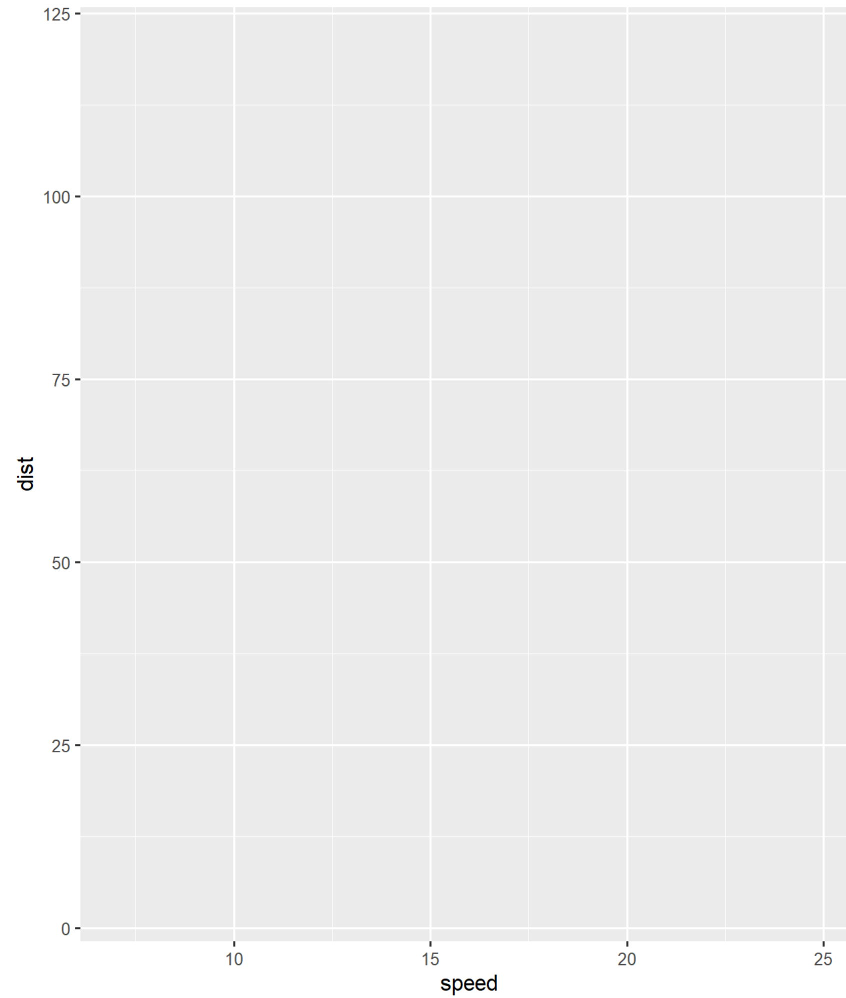


break_type = "user", with #BREAK

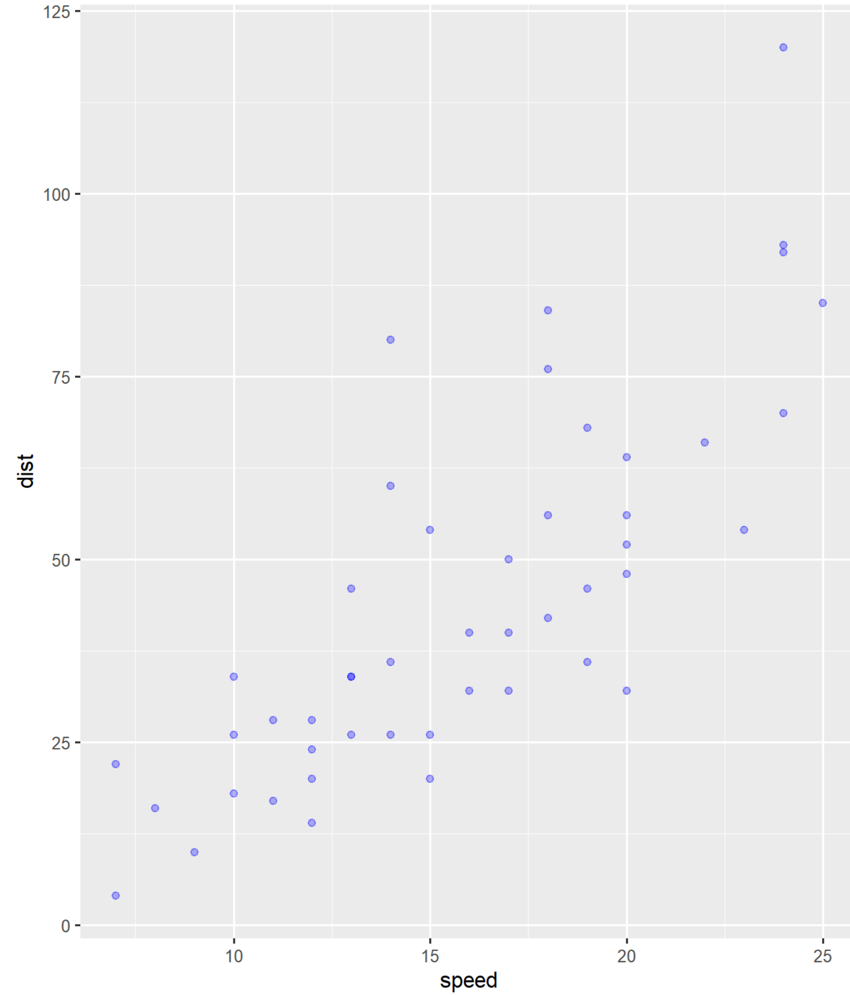
If the break_type is set to "user", the breakpoints are those indicated by the user with the special comment #BREAK

```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) + #BREAK
  geom_point(
    alpha = .3, #BREAK2
    color = "blue" #BREAK3
    ) + #BREAK
  aes(size = speed) #BREAK
```

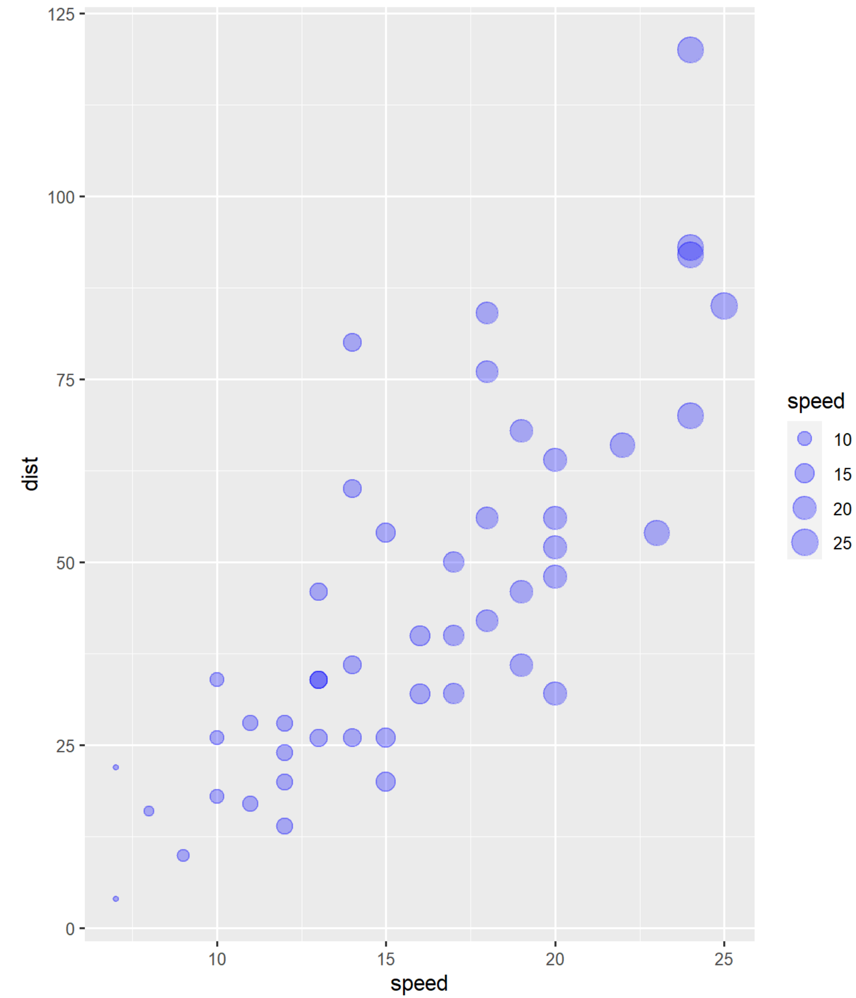
```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist)
```



```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    )
```



```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    ) +
  aes(size = speed)
```

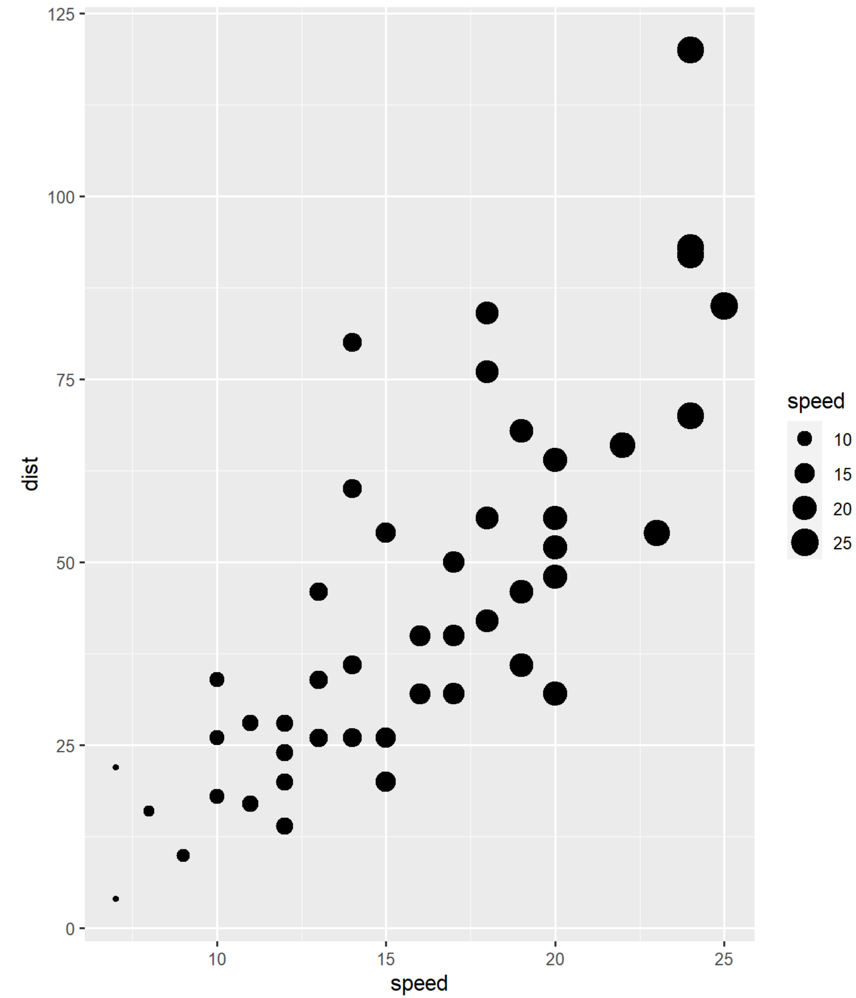


break_type = "non_seq", with #BREAK2, #BREAK3

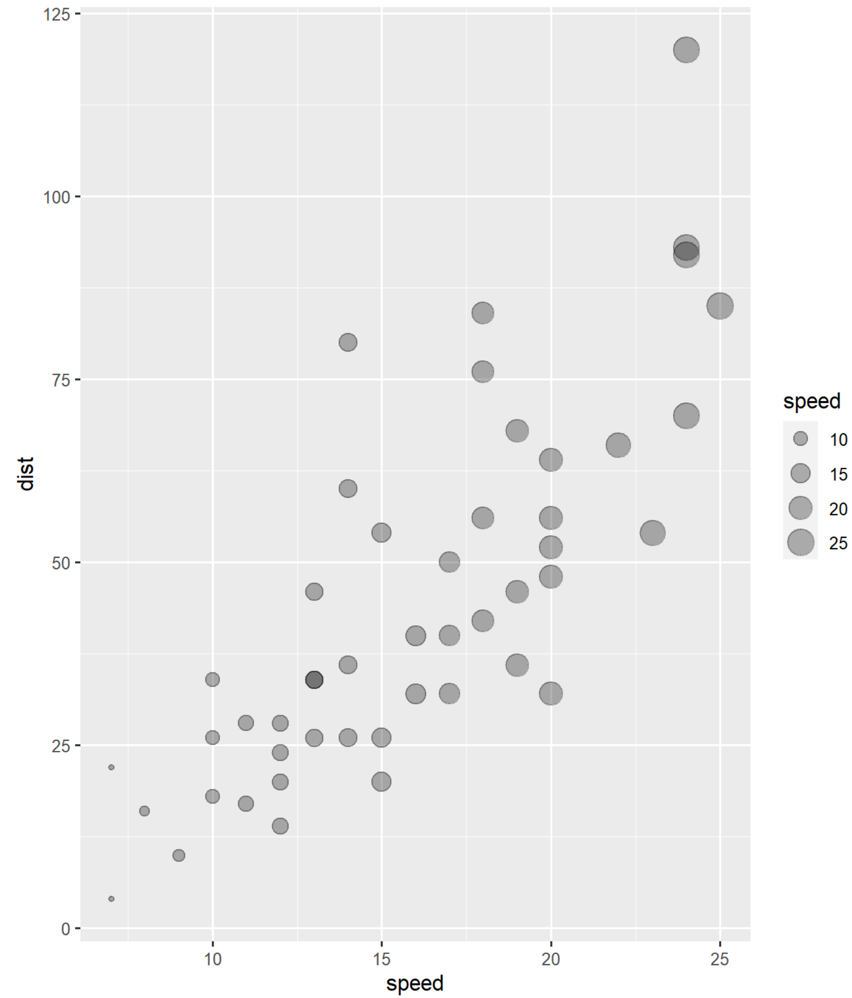
If the break_type is set to "non_seq", the breakpoints are those indicated by the user with the special numeric comment #BREAK2, #BREAK3 etc to indicate at which point in time the code should appear.

```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) + #BREAK
  geom_point(
    alpha = .3, #BREAK2
    color = "blue" #BREAK3
    ) + #BREAK
  aes(size = speed) #BREAK
```

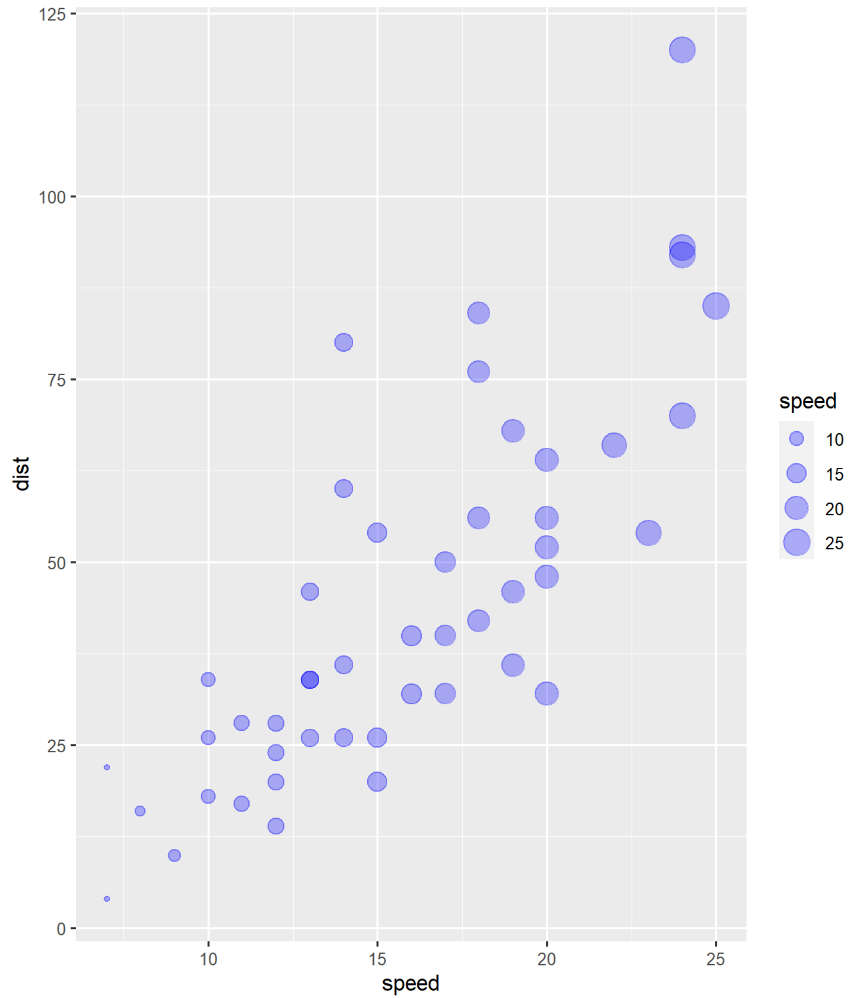
```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    ) +
  aes(size = speed)
```



```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    ) +
  aes(size = speed)
```



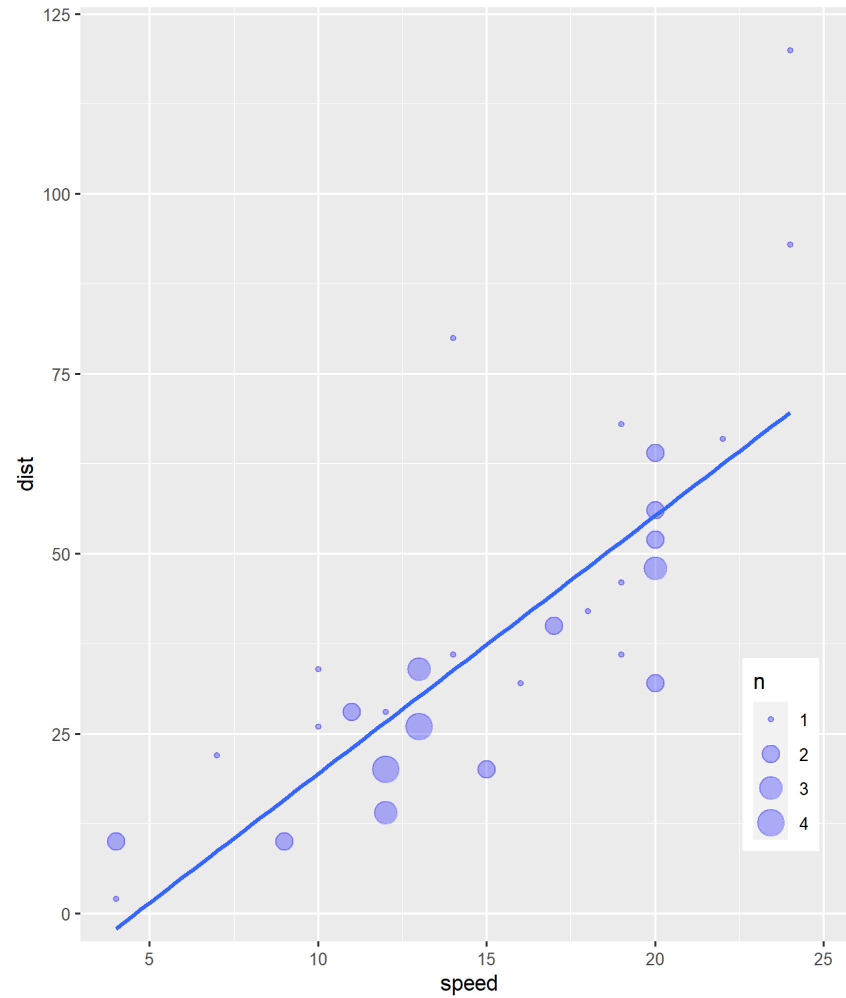
```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    ) +
  aes(size = speed)
```



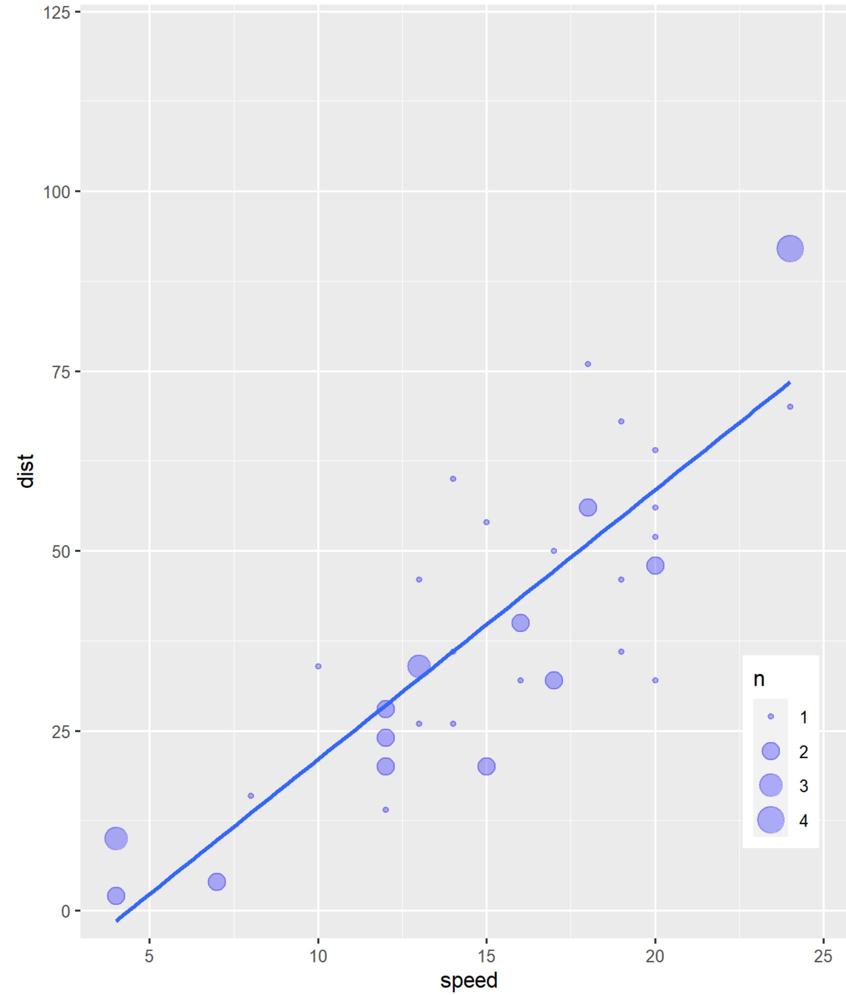
`break_type = 5` (or "entering the multiverse")

Another modality is to set break_type equal to a positive integer, indicating that you want the same code chunk to be displayed multiple times. This makes the most sense in a setting where there is some randomization or random sampling and you want to see different realizations. Let's see this used on the user input code chunk "cars_multi", whose first step is to randomly sample rows from the data set cars with replacement.

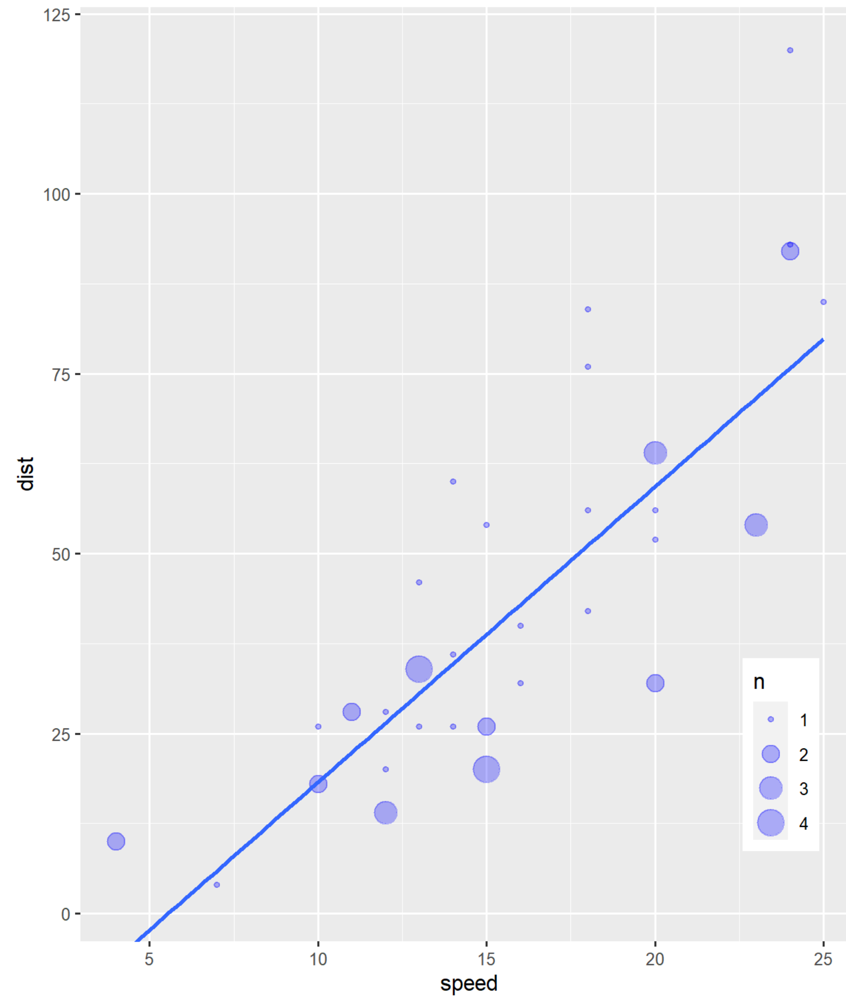
```r
cars %>%
  sample_frac(size = 1, replace = TRUE) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_count(
    alpha = .3,
    color = "blue"
    ) +
  geom_smooth(method = lm, se = FALSE) +
  coord_cartesian(xlim = range(cars$speed),
                  ylim = range(cars$dist)) +
  theme(legend.position = c(.9, .2))
```



```
cars %>%
  sample_frac(size = 1, replace = TRUE) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_count(
    alpha = .3,
    color = "blue"
    ) +
  geom_smooth(method = lm, se = FALSE) +
  coord_cartesian(xlim = range(cars$speed),
                  ylim = range(cars$dist)) +
  theme(legend.position = c(.9, .2))
```



```r
cars %>%
  sample_frac(size = 1, replace = TRUE) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_count(
    alpha = .3,
    color = "blue"
    ) +
  geom_smooth(method = lm, se = FALSE) +
  coord_cartesian(xlim = range(cars$speed),
                  ylim = range(cars$dist)) +
  theme(legend.position = c(.9, .2))
```



display_type

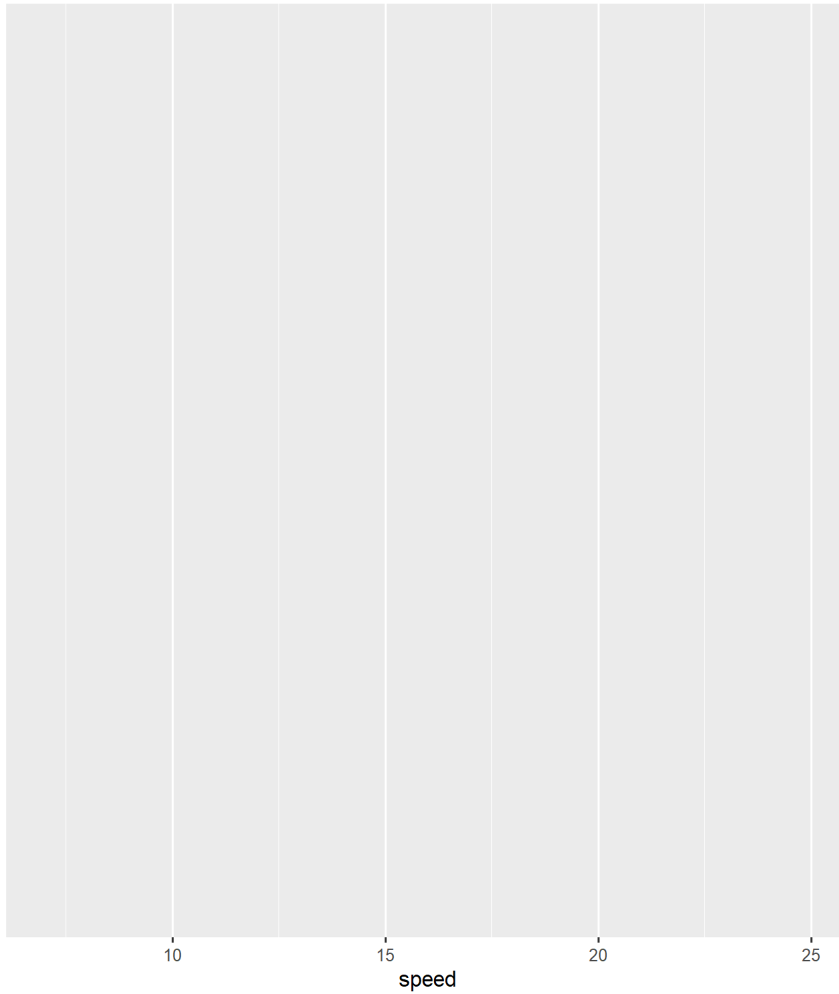
There are also different display modalities. Namely you can indicate if you want "both" the code and the output displayed in your flipbookification, or just the "output" (perhaps to be used in a traditional presentation), or just the "code" (which might be used to kind of test student expectations about some code). You have already seen the default where the parameter display_type is set to "both", but let's have a look at "output" and "code" only.

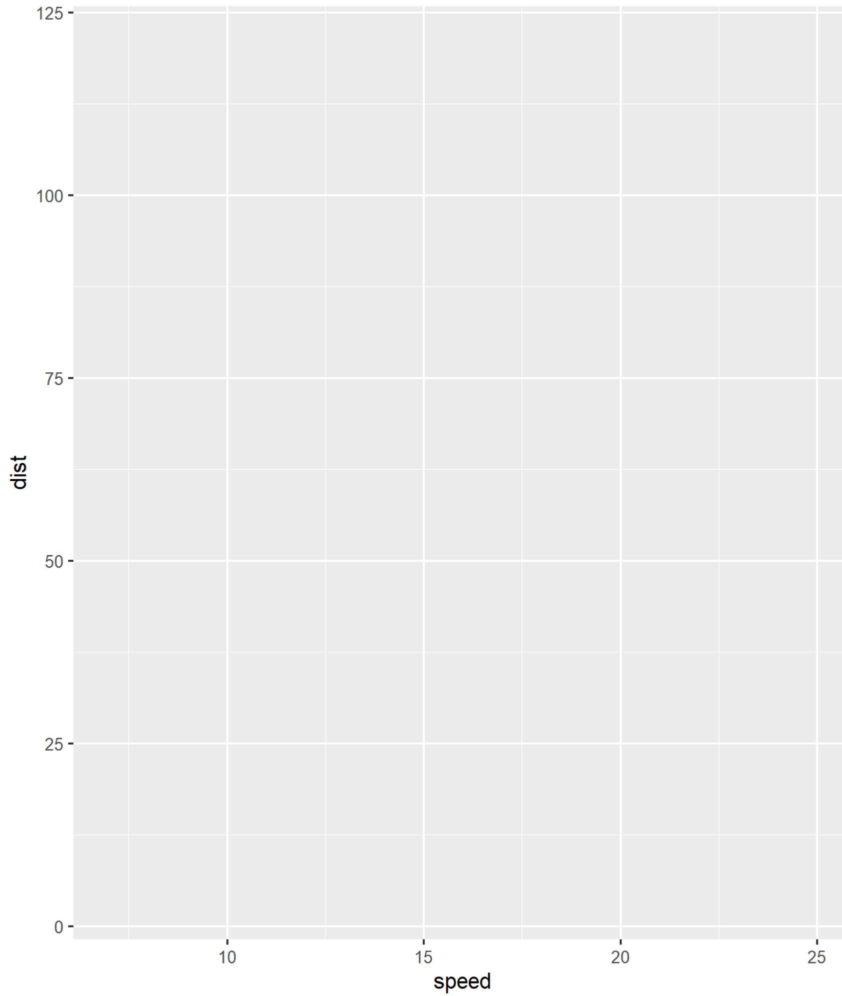
display_type = "output"

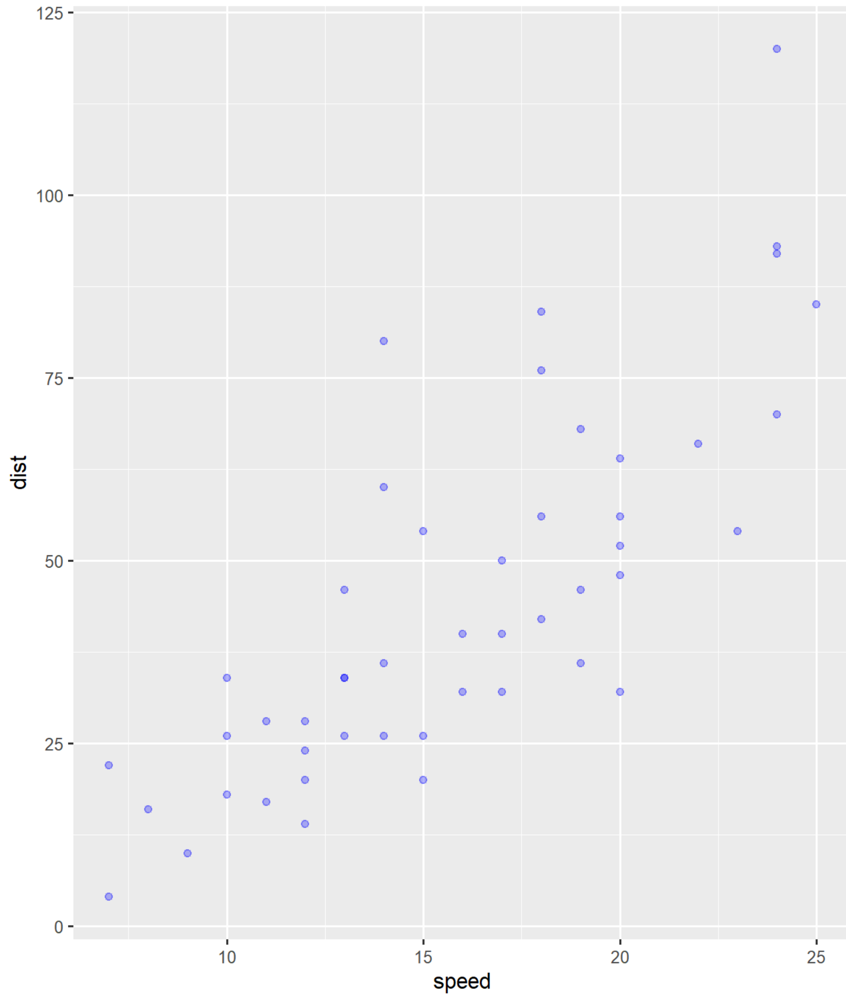
Let's look at where only the *output* is displayed for the "cars" code chunk.

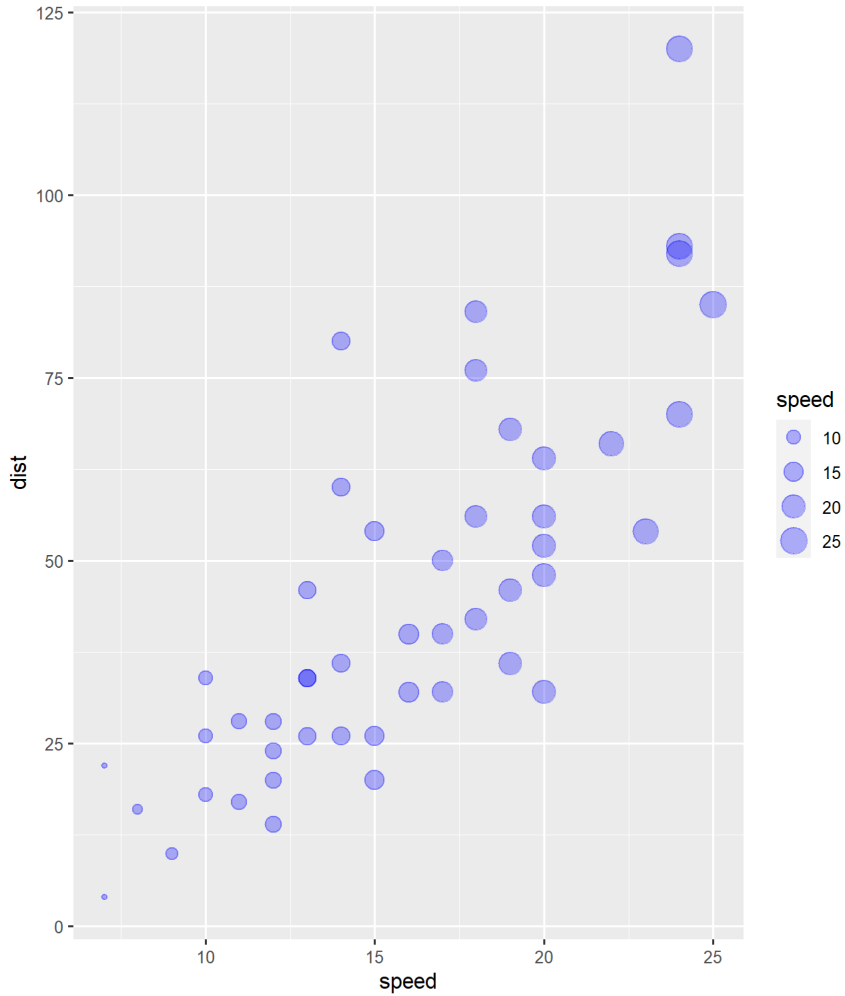
	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10
7	10	18
8	10	26
9	10	34
10	11	17
11	11	28
12	12	14
13	12	20
14	12	24
15	12	28
16	13	26
17	13	34
18	13	34
19	13	46
20	14	26
21	14	36
22	14	60
23	14	80
24	15	20
25	15	26

```
   speed dist
1      7    4
2      7   22
3      8   16
4      9   10
5     10   18
6     10   26
7     10   34
8     11   17
9     11   28
10    12   14
11    12   20
12    12   24
13    12   28
14    13   26
15    13   34
16    13   34
17    13   46
18    14   26
19    14   36
20    14   60
21    14   80
22    15   20
23    15   26
24    15   54
25    16   32
```


10
15
20
25
speed


125
100
75
50
25
0
dist
10
15
20
25
speed


125
100
75
50
25
0
dist
10
15
20
25
speed



125
100
75
50
25
0
dist
10
15
20
25
speed
speed
10
15
20
25

`display_type = "code"`

And now where only the *code* is displayed for the "cars" code chunk.

```
cars
```

```
cars %>%
  filter(speed > 4)
```

```r
cars %>%
  filter(speed > 4) %>%
  ggplot()
```

```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed)
```

```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist)
```

```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    )
```

```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    ) +
  aes(size = speed)
```

`display_type = c("output", "code")`

Not sure why you'd want to do this, but you can flip output and code. It's also not totally stable - jumps moving from tabular output to figure. Have to figure that one out. It is something to do with fixed height.

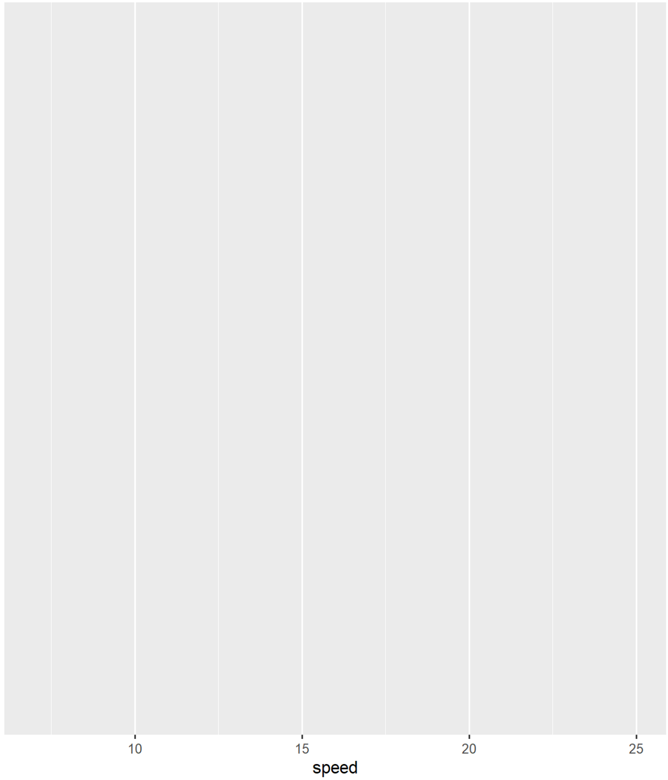
```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```
cars
```

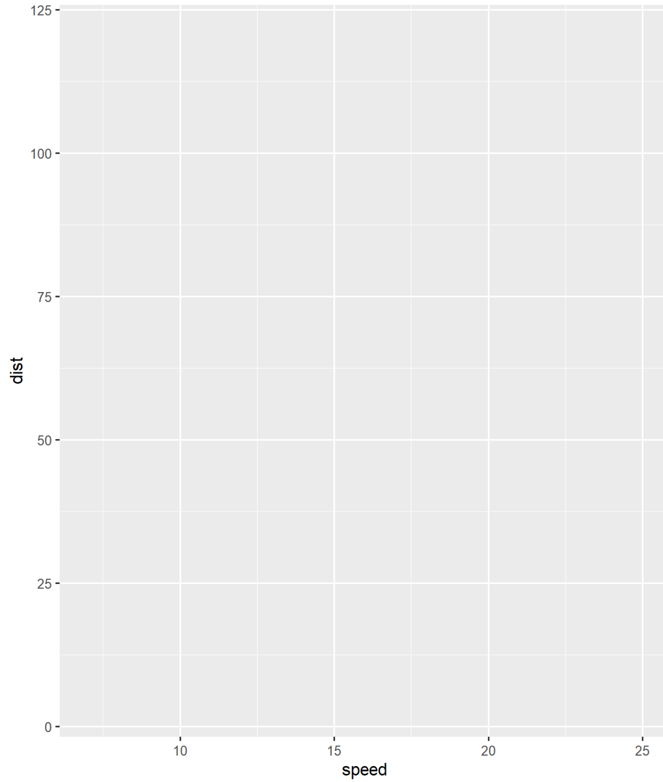
```
   speed dist
1      7    4
2      7   22
3      8   16
4      9   10
5     10   18
6     10   26
7     10   34
8     11   17
9     11   28
10    12   14
11    12   20
12    12   24
13    12   28
14    13   26
15    13   34
16    13   34
17    13   46
18    14   26
19    14   36
20    14   60
21    14   80
22    15   20
23    15   26
24    15   54
25    16   32
26    16   40
27    17   32
28    17   40
29    17   50
30    18   42
31    18   56
32    18   76
```

```
cars %>%
  filter(speed > 4)
```

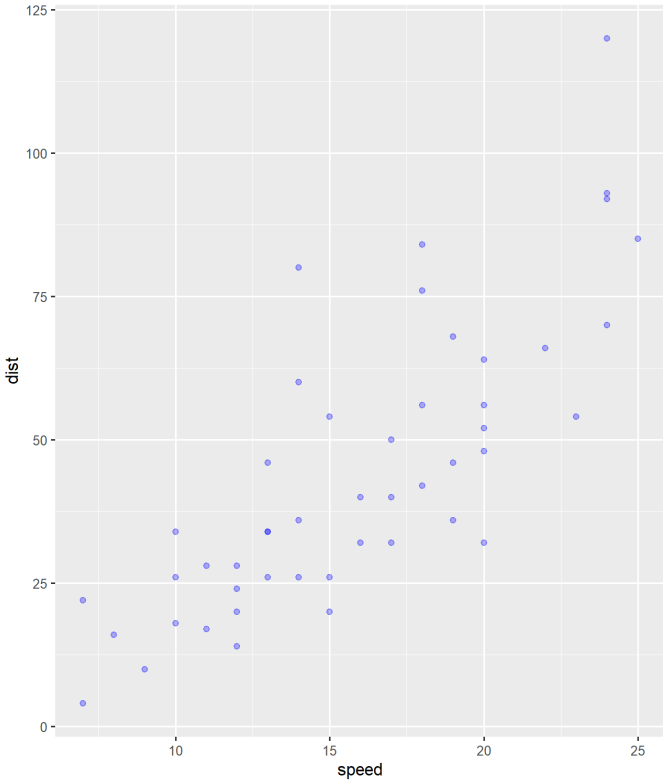
```r
cars %>%
  filter(speed > 4) %>%
  ggplot()
```



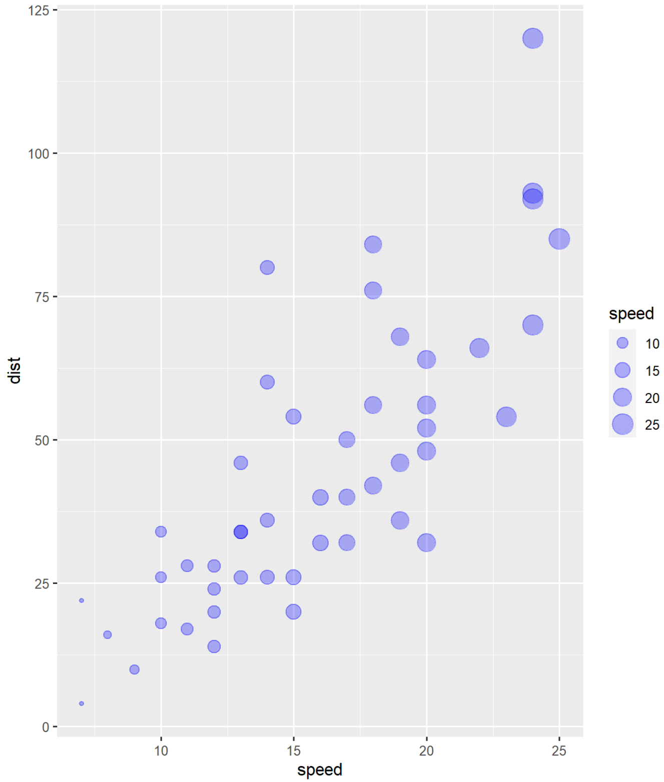

```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed)
```




```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist)
```




```r
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    )
```




```
cars %>%
  filter(speed > 4) %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point(
    alpha = .3,
    color = "blue"
    ) +
  aes(size = speed)
```

Assignment

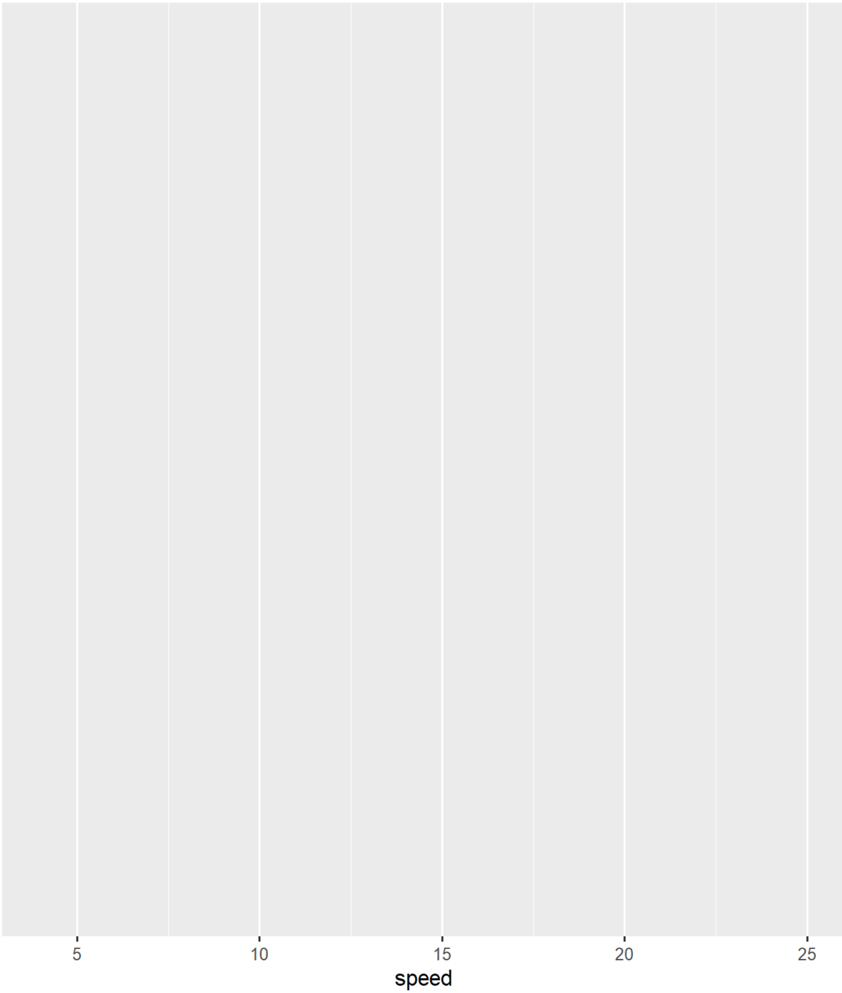
If you want to create an object in your flipbooks, it is most "natural" to use right assignment. Working sequentially with a pipeline of code, you get feedback all along the way until you get to the point of assigning all of what you have done to a new object with right assignment. Creating objects in one "source" code chunk, means that you can break up a pipeline of tasks into multiple code chunks. Let's see this in action.

```r
cars
```

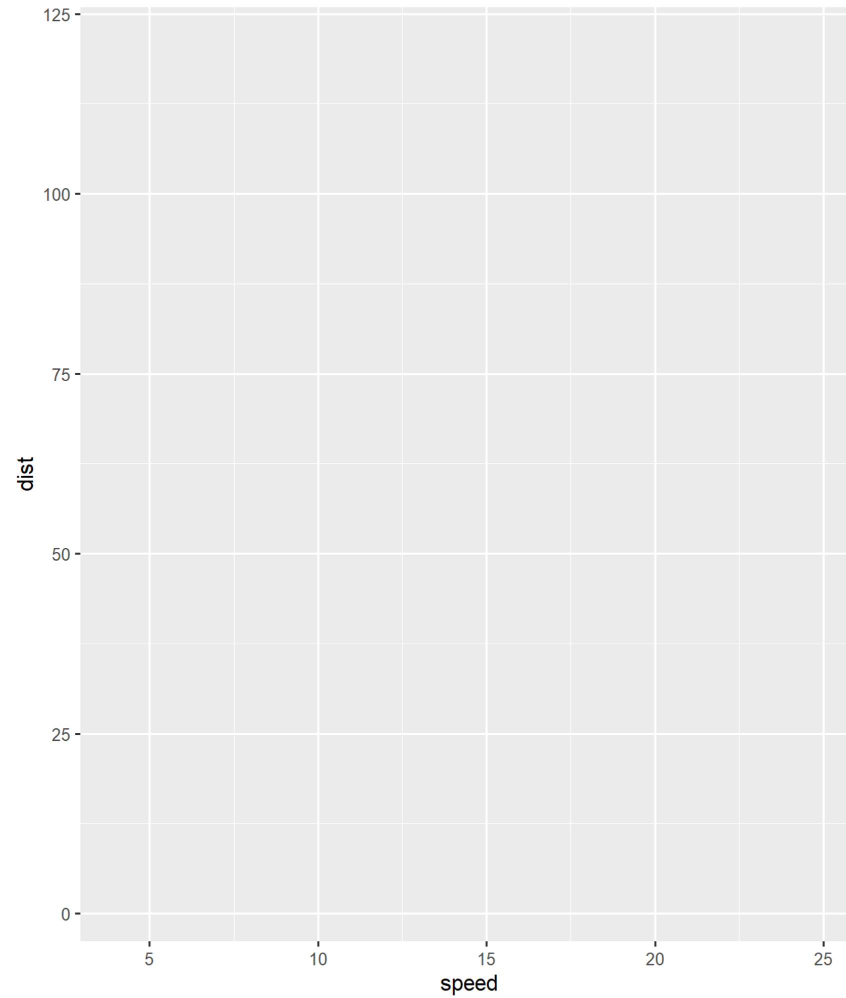
```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```
cars %>%
  ggplot()
```

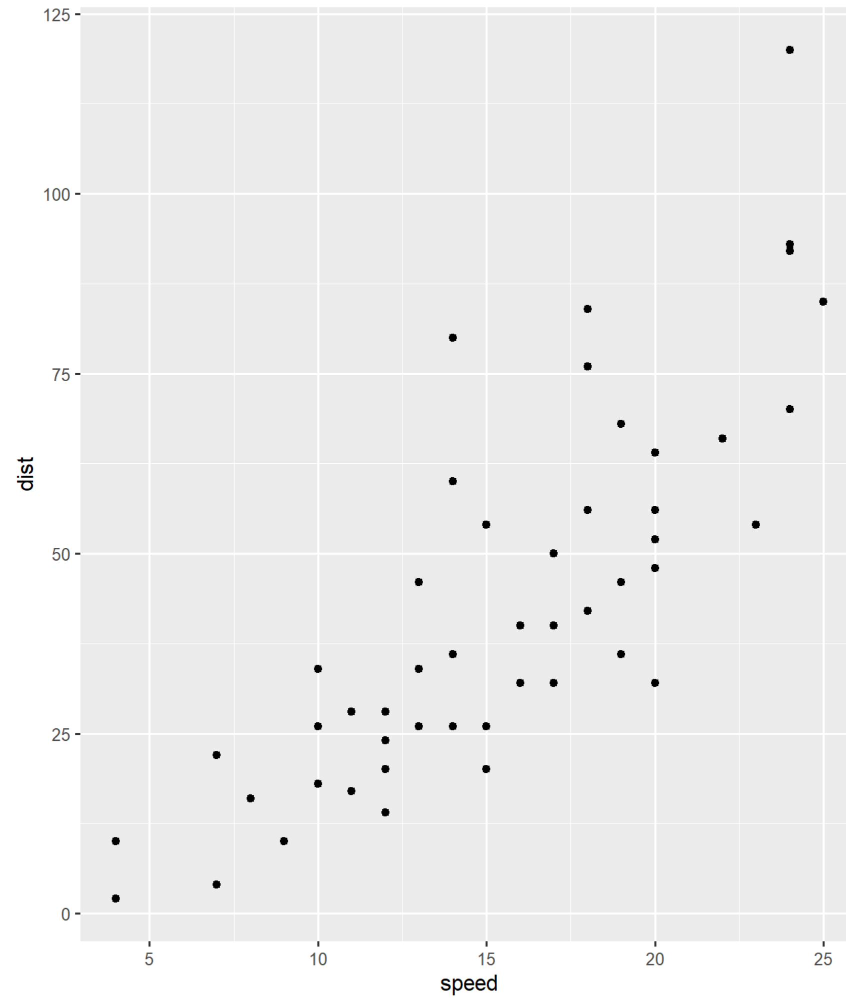
```r
cars %>%
  ggplot() +
  aes(x = speed)
```



```
cars %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist)
```

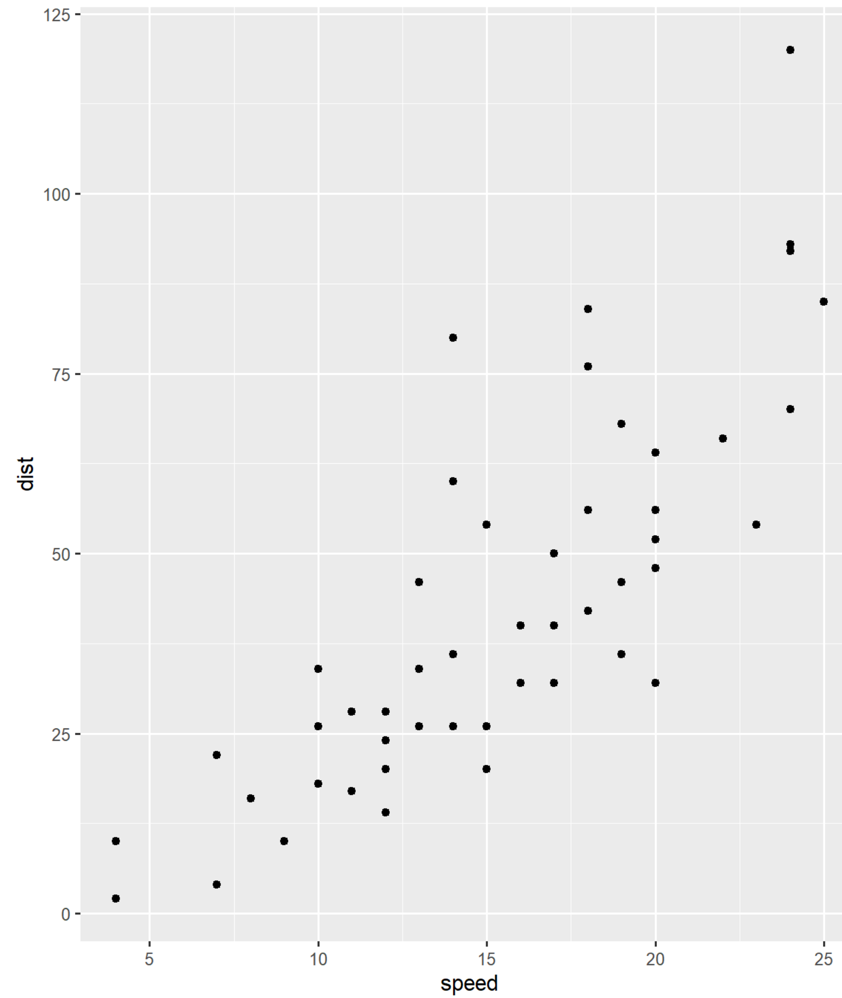


```
cars %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point()
```

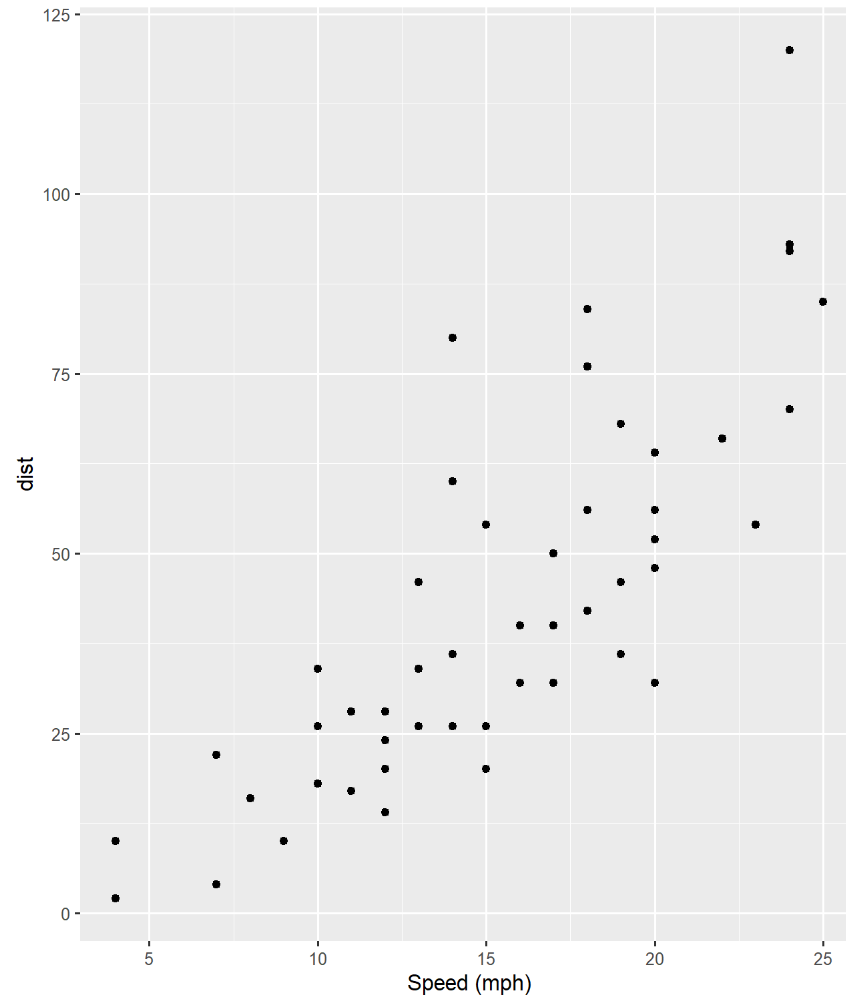


```
cars %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point() ->
cars_plot
```

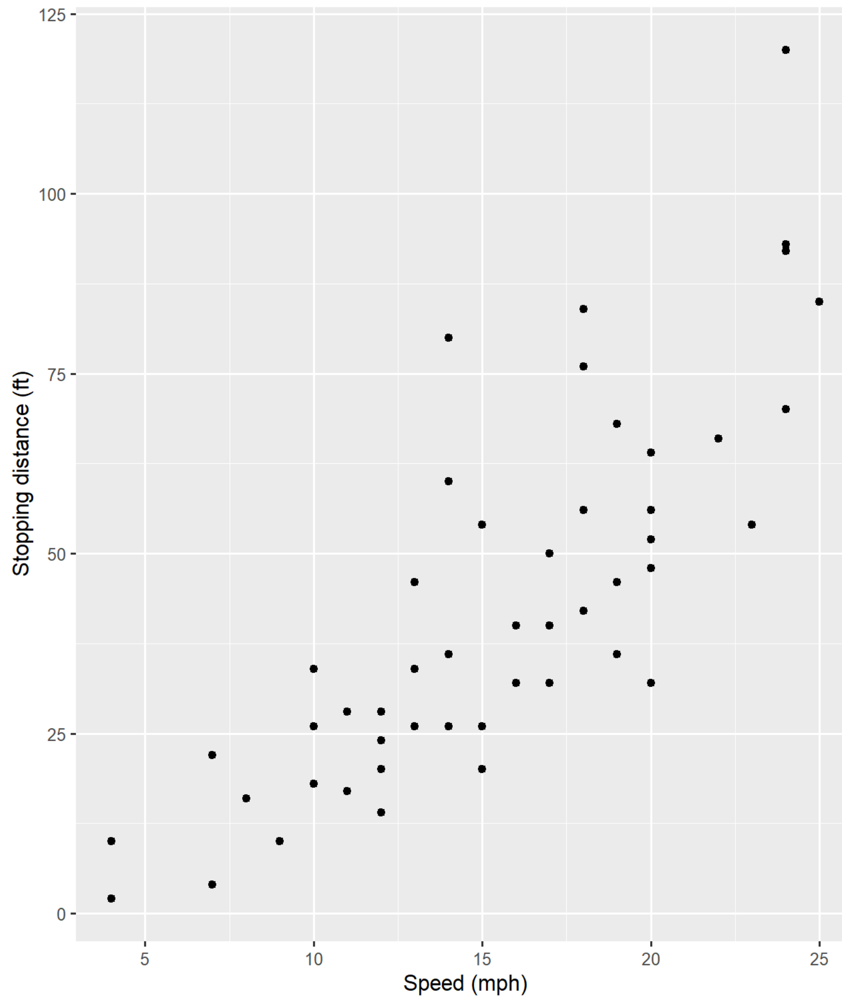
```
cars_plot
```



```
cars_plot +
  labs(x = "Speed (mph)")
```



```
cars_plot +
  labs(x = "Speed (mph)") +
  labs(y = "Stopping distance (ft)")
```



left_assign = TRUE

With left assignment in R, you don't get any feedback, so flipbooking prefers this step at the end of a pipeline, so we can enjoy all the nice feedback. So the parameter left_assign is by default set to FALSE.

But, setting the left_assign paramter to T and using left assignment, you can still create a meaningful flipbook that gives you feedback. When left_assign = TRUE, the first object that is created prints at the end of the derivative code chunks.

```r
my_plot <- cars    # the data

my_plot
```

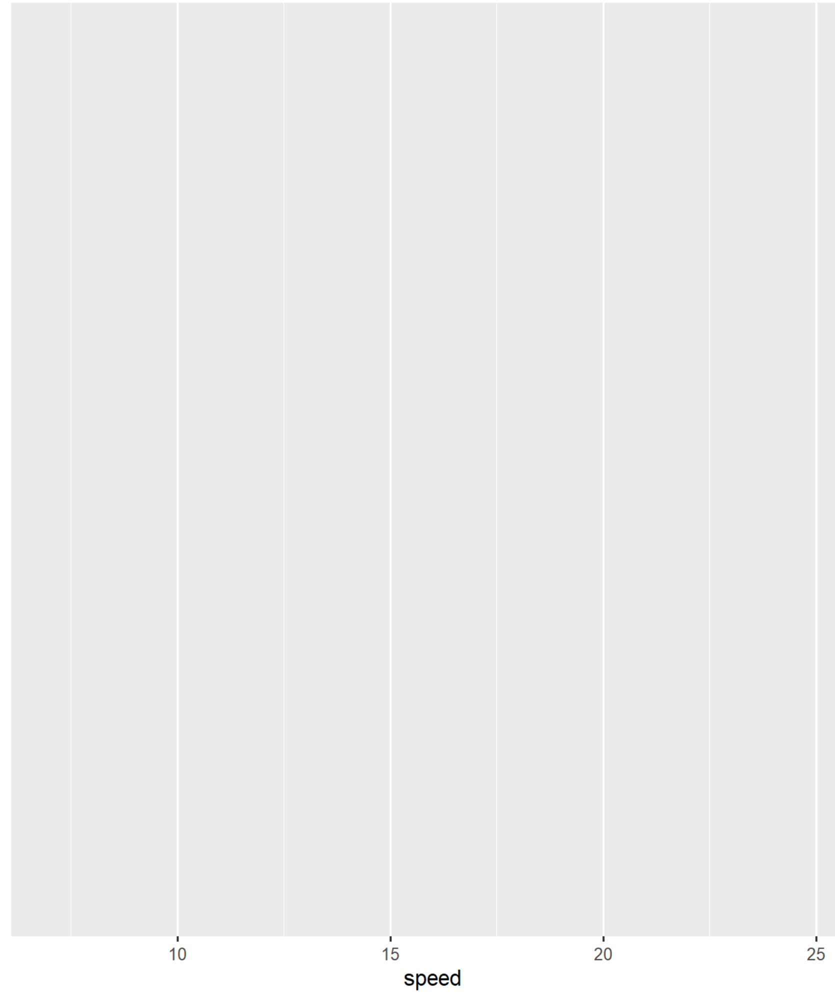
```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```r
my_plot <- cars %>%  # the data
  filter(speed > 4)    # subset

my_plot
```

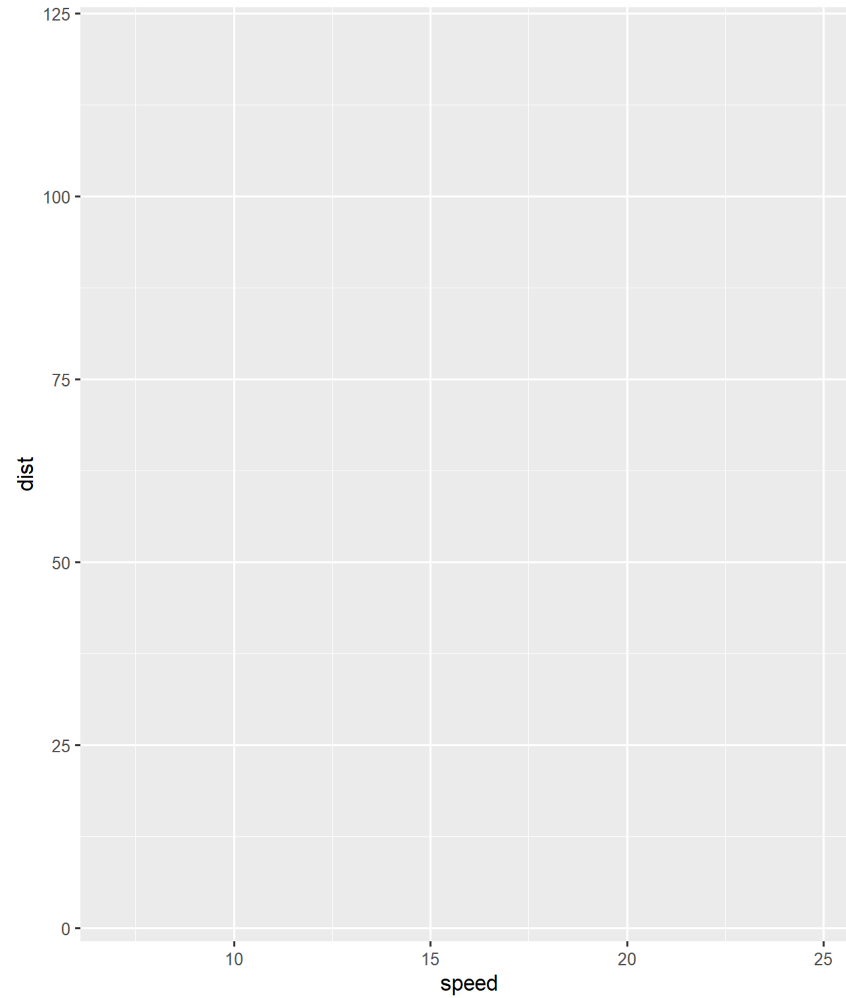
```
   speed dist
1      7    4
2      7   22
3      8   16
4      9   10
5     10   18
6     10   26
7     10   34
8     11   17
9     11   28
10    12   14
11    12   20
12    12   24
13    12   28
14    13   26
15    13   34
16    13   34
17    13   46
18    14   26
19    14   36
20    14   60
21    14   80
22    15   20
23    15   26
24    15   54
25    16   32
26    16   40
27    17   32
28    17   40
29    17   50
30    18   42
31    18   56
32    18   76
```

```
my_plot <- cars %>%  # the data
  filter(speed > 4) %>%  # subset
  ggplot()   # pipe to ggplot

my_plot
```

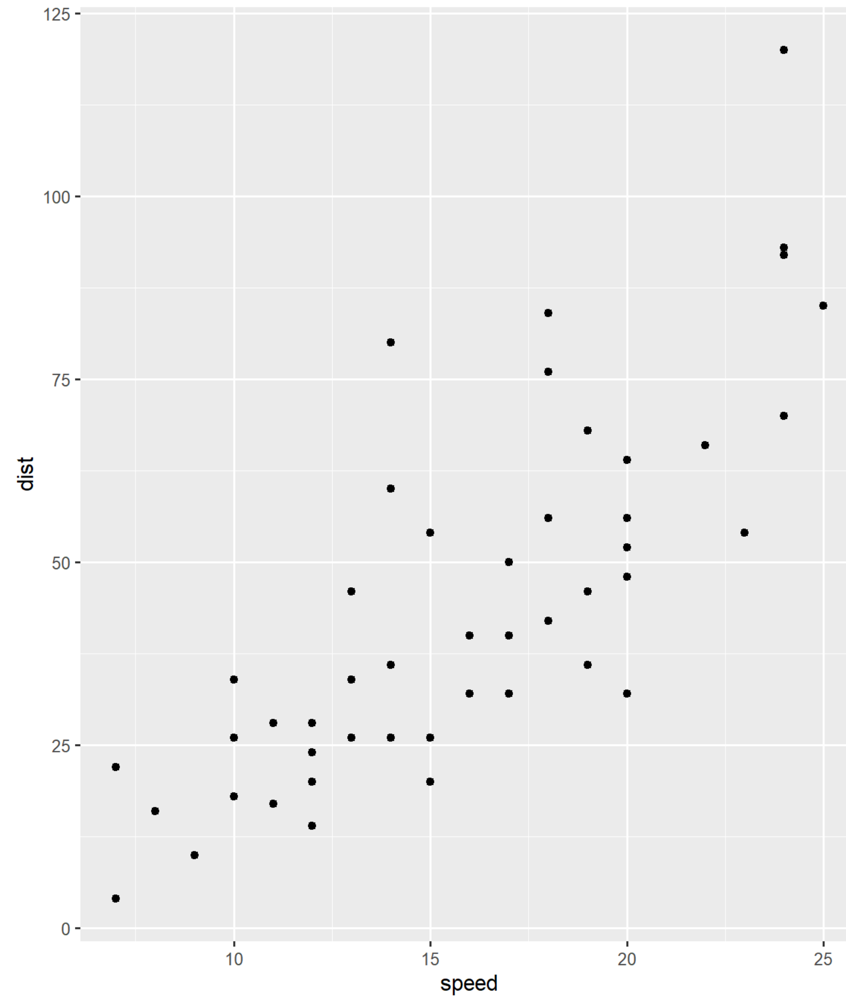
```r
my_plot <- cars %>%  # the data
  filter(speed > 4) %>%  # subset
  ggplot() +  # pipe to ggplot
  aes(x = speed)

my_plot
```



```r
my_plot <- cars %>%  # the data
  filter(speed > 4) %>%  # subset
  ggplot() +  # pipe to ggplot
  aes(x = speed) +
  aes(y = dist)

my_plot
```



```
my_plot <- cars %>%  # the data
  filter(speed > 4) %>%  # subset
  ggplot() +  # pipe to ggplot
  aes(x = speed) +
  aes(y = dist) +
  geom_point()

my_plot
```



Managing source code chunks

So, it is pretty cool that we can create a bunch of derivative code chunks from one input code chunk (a foundational blog post by Emi Tanaka on this here). But there are some considerations then for this source chunk. What should its chunk options be? The easy way is to set all "source" code chunks to include = F, as I do throughout the book. However, you might consider a combination of `eval` and `echo` instead; you can come back to this idea as you become a more seasoned flipbooker.

Beyond the tidyverse

It is no surprise that Flipbooks are born in the context of the popularity of the tidyverse tools --- tools that are designed be be used in sequential pipelines and that give a satisfying amount of feedback along the way!

But base R techniques and other popular tools can certainly also be employed.

"chaining" by overwriting objects

```r
cars_mod <- cars

cars_mod
```

```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```r
cars_mod <- cars
cars_mod$half_dist <- cars$dist / 2

cars_mod
```

```
   speed dist half_dist
1      4    2       1.0
2      4   10       5.0
3      7    4       2.0
4      7   22      11.0
5      8   16       8.0
6      9   10       5.0
7     10   18       9.0
8     10   26      13.0
9     10   34      17.0
10    11   17       8.5
11    11   28      14.0
12    12   14       7.0
13    12   20      10.0
14    12   24      12.0
15    12   28      14.0
16    13   26      13.0
17    13   34      17.0
18    13   34      17.0
19    13   46      23.0
20    14   26      13.0
21    14   36      18.0
22    14   60      30.0
23    14   80      40.0
24    15   20      10.0
25    15   26      13.0
26    15   54      27.0
27    16   32      16.0
28    16   40      20.0
29    17   32      16.0
30    17   40      20.0
31    17   50      25.0
32    18   42      21.0
```

```r
cars_mod <- cars
cars_mod$half_dist <- cars$dist / 2
names(cars_mod)[2] <- "distance"

cars_mod
```

```
   speed distance half_dist
1      4        2       1.0
2      4       10       5.0
3      7        4       2.0
4      7       22      11.0
5      8       16       8.0
6      9       10       5.0
7     10       18       9.0
8     10       26      13.0
9     10       34      17.0
10    11       17       8.5
11    11       28      14.0
12    12       14       7.0
13    12       20      10.0
14    12       24      12.0
15    12       28      14.0
16    13       26      13.0
17    13       34      17.0
18    13       34      17.0
19    13       46      23.0
20    14       26      13.0
21    14       36      18.0
22    14       60      30.0
23    14       80      40.0
24    15       20      10.0
25    15       26      13.0
26    15       54      27.0
27    16       32      16.0
28    16       40      20.0
29    17       32      16.0
30    17       40      20.0
31    17       50      25.0
32    18       42      21.0
```

```r
cars_mod <- cars
cars_mod$half_dist <- cars$dist / 2
names(cars_mod)[2] <- "distance"
cars_mod <- cars_mod[cars_mod$distance > 10,]

cars_mod
```

```
   speed distance half_dist
4      7       22      11.0
5      8       16       8.0
7     10       18       9.0
8     10       26      13.0
9     10       34      17.0
10    11       17       8.5
11    11       28      14.0
12    12       14       7.0
13    12       20      10.0
14    12       24      12.0
15    12       28      14.0
16    13       26      13.0
17    13       34      17.0
18    13       34      17.0
19    13       46      23.0
20    14       26      13.0
21    14       36      18.0
22    14       60      30.0
23    14       80      40.0
24    15       20      10.0
25    15       26      13.0
26    15       54      27.0
27    16       32      16.0
28    16       40      20.0
29    17       32      16.0
30    17       40      20.0
31    17       50      25.0
32    18       42      21.0
33    18       56      28.0
34    18       76      38.0
35    18       84      42.0
36    19       36      18.0
```

```r
cars_mod <- cars
cars_mod$half_dist <- cars$dist / 2
names(cars_mod)[2] <- "distance"
cars_mod <- cars_mod[cars_mod$distance > 10,]
cars_mod <- cars_mod["distance"]

cars_mod
```

```
   distance
4        22
5        16
7        18
8        26
9        34
10       17
11       28
12       14
13       20
14       24
15       28
16       26
17       34
18       34
19       46
20       26
21       36
22       60
23       80
24       20
25       26
26       54
27       32
28       40
29       32
30       40
31       50
32       42
33       56
34       76
35       84
36       36
```

using the .[] and .[[]] syntax with the migrittr pipe - %>%

Flipbooking can also be applied to logical indexing workflows if the steps are broken up using the %>% followed by .[] and .[[]]. Thus flipbooking can also be used with base R logical indexing and with the popular `data.table` package.

```r
cars
```

```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```r
cars %>%
  .[cars$speed > median(cars$speed),]
```

```
   speed dist
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
33    18   56
34    18   76
35    18   84
36    19   36
37    19   46
38    19   68
39    20   32
40    20   48
41    20   52
42    20   56
43    20   64
44    22   66
45    23   54
46    24   70
47    24   92
48    24   93
49    24  120
50    25   85
```

```r
cars %>%
  .[cars$speed > median(cars$speed),] %>%
  .["speed"]
```

```
   speed
27    16
28    16
29    17
30    17
31    17
32    18
33    18
34    18
35    18
36    19
37    19
38    19
39    20
40    20
41    20
42    20
43    20
44    22
45    23
46    24
47    24
48    24
49    24
50    25
```

```r
cars %>%
  .[cars$speed > median(cars$speed),] %>%
  .["speed"] %>%
  .[,1]
```

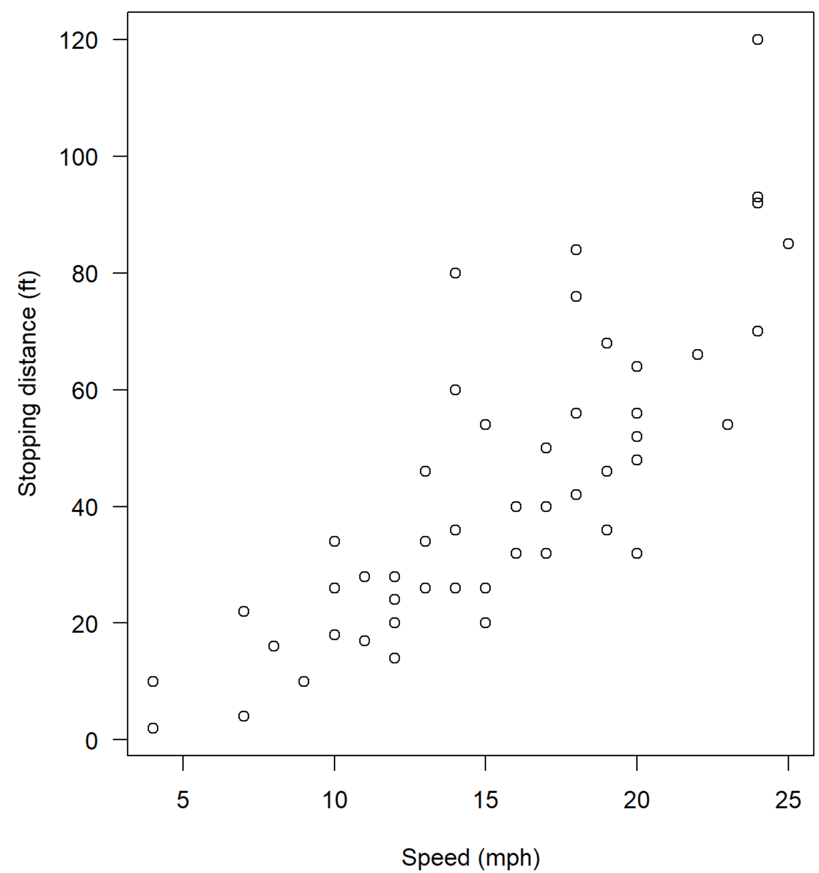
```
[1] 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

```r
cars %>%
  .[cars$speed > median(cars$speed),] %>%
  .["speed"] %>%
  .[,1] ->
top_speeds
```

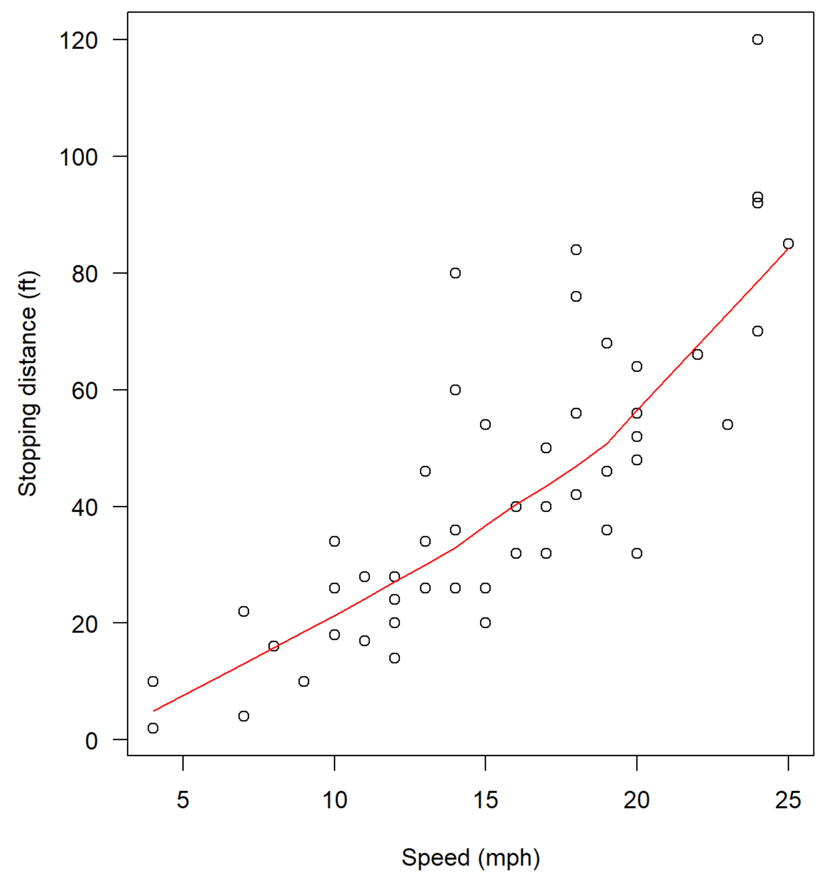
Base R plotting

It has been a while since I've done much plotting with base R, but I think it is important to have an example or two.

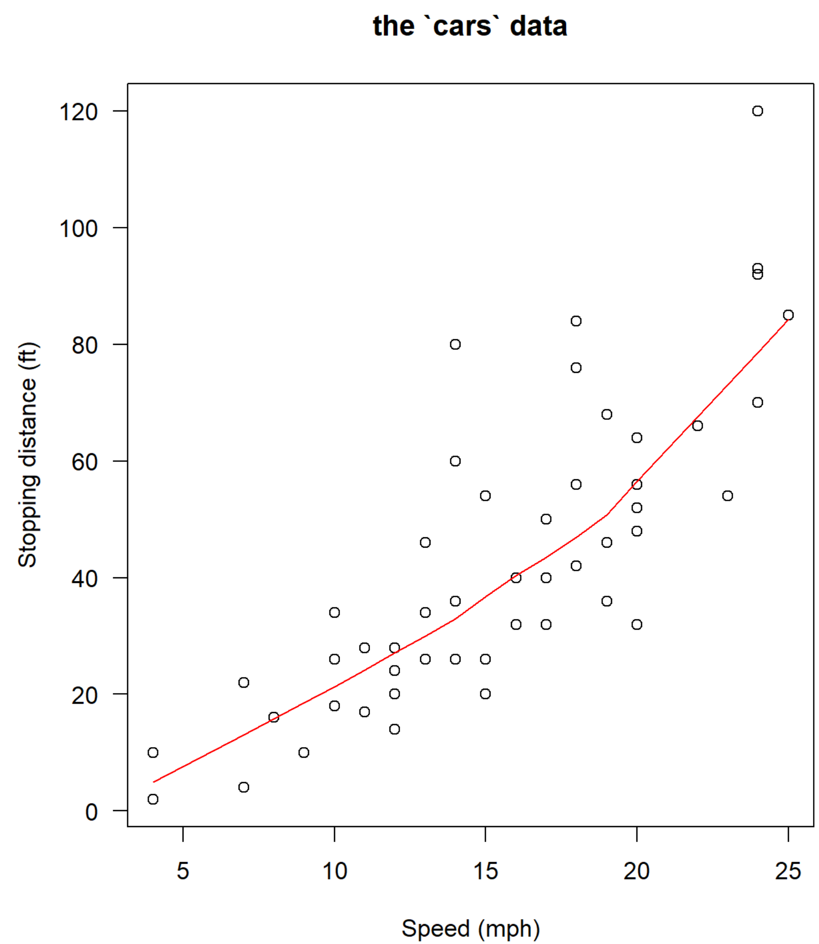
```
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
```



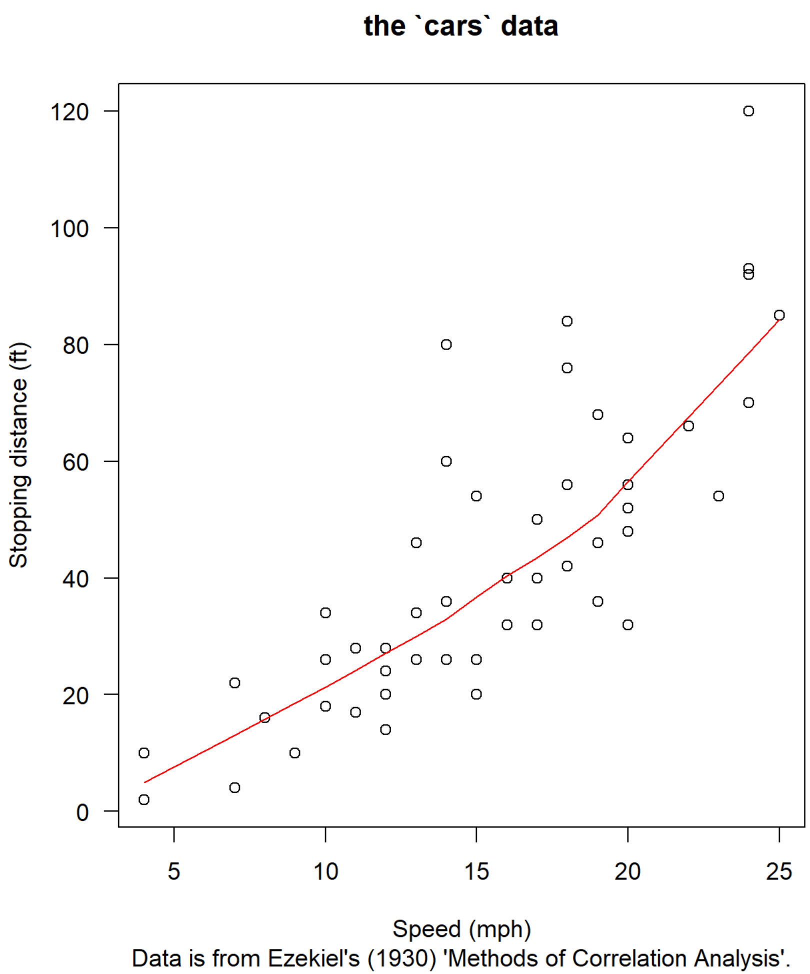
```r
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
lines(lowess(cars$speed, cars$dist,
             f = 2/3, iter = 3),
      col = "red")
```



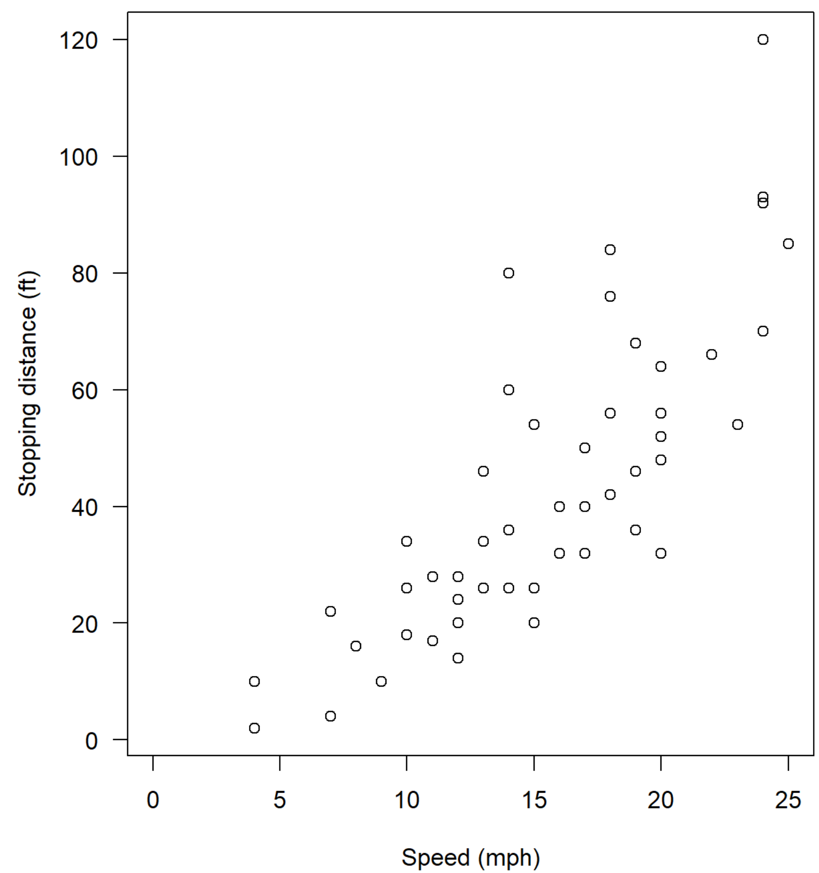
```r
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
lines(lowess(cars$speed, cars$dist,
             f = 2/3, iter = 3),
      col = "red")
title(main = "the `cars` data")
```



```r
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
     las = 1)
lines(lowess(cars$speed, cars$dist,
             f = 2/3, iter = 3),
      col = "red")
title(main = "the `cars` data")
title(sub = "Data is from Ezekiel's (1930) 'M
```

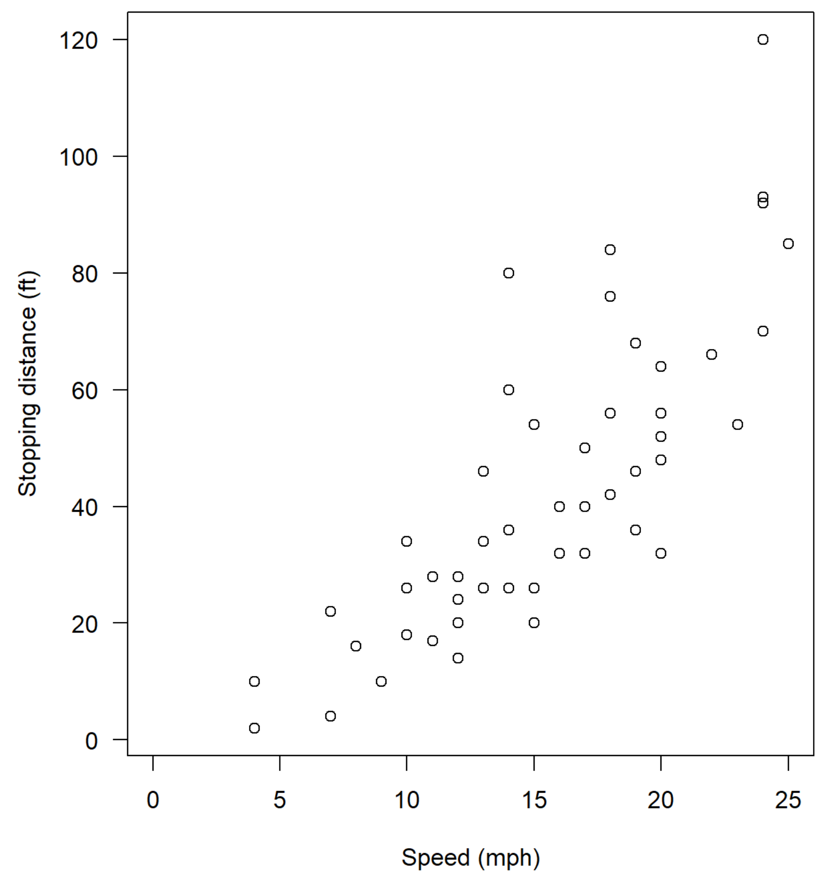


```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))
```



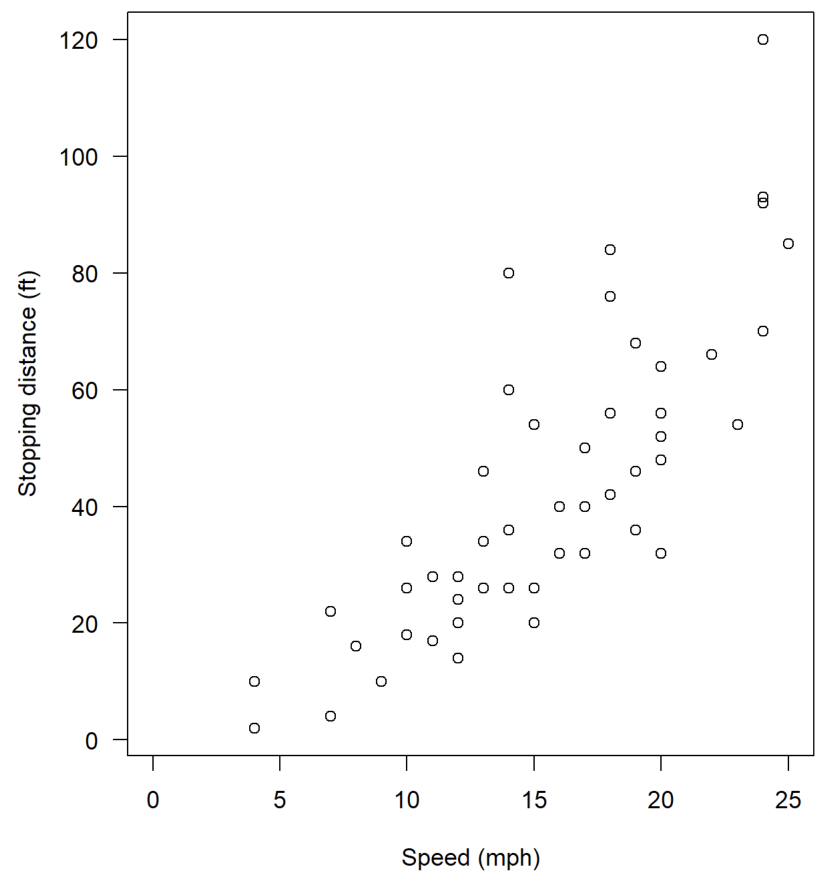
```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))
```

```r
lm(dist ~ poly(speed, 3),
   data = cars)
```

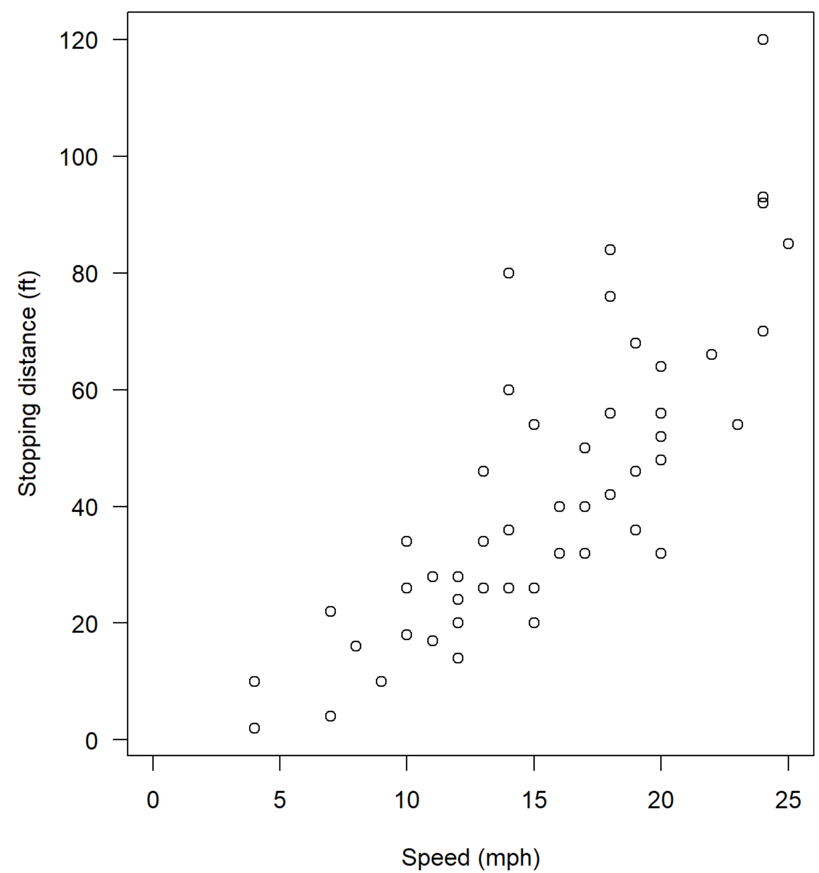



```
Call:
lm(formula = dist ~ poly(speed, 3), data = cars)

Coefficients:
    (Intercept)  poly(speed, 3)1  poly(speed, 3)2  poly(speed, 3)3
          42.98           145.55            23.00            13.80
```

```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))

lm(dist ~ poly(speed, 3),
   data = cars) ->
model
```

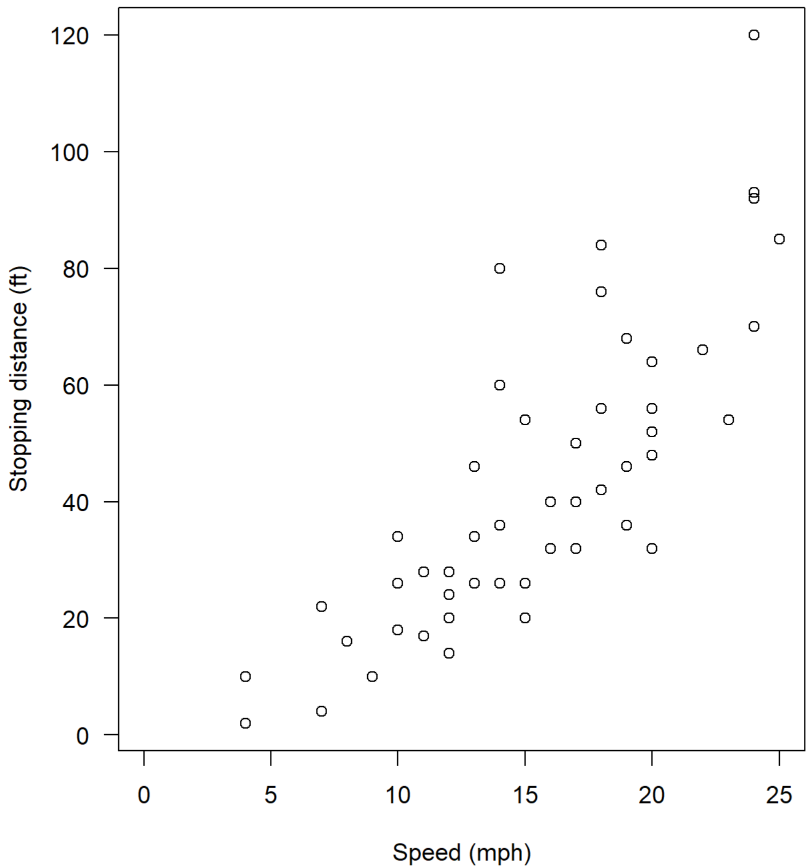


```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))

lm(dist ~ poly(speed, 3),
   data = cars) ->
model

seq(0, 25, length.out = 25)
```

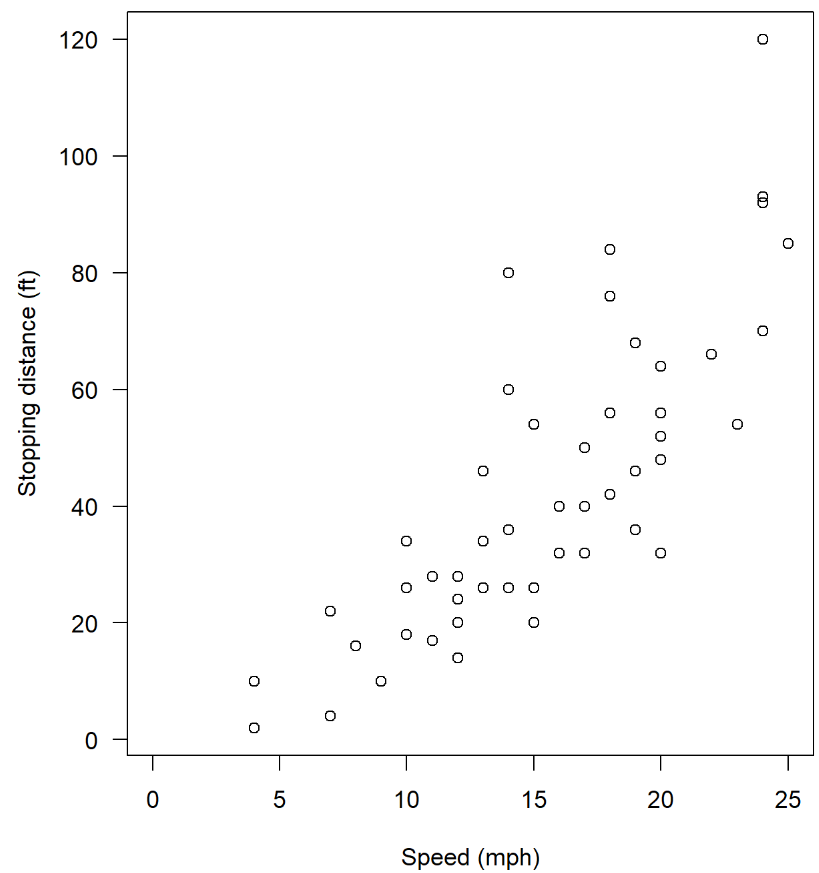



```
 [1]  0.000000  1.041667  2.083333  3.125000  4.166667  5.208333  6.250000  7.29166
 [9]  8.333333  9.375000 10.416667 11.458333 12.500000 13.541667 14.583333 15.62500
[17] 16.666667 17.708333 18.750000 19.791667 20.833333 21.875000 22.916667 23.95833
[25] 25.000000
```

```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))

lm(dist ~ poly(speed, 3),
   data = cars) ->
model

seq(0, 25, length.out = 25) ->
inputs_of_x
```

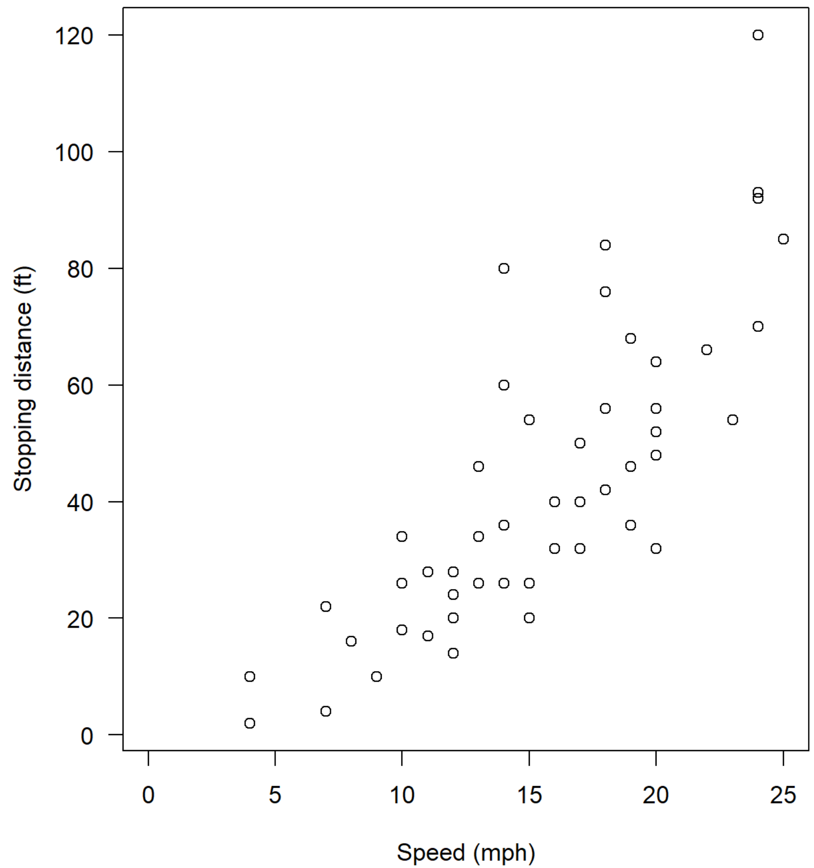


```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))

lm(dist ~ poly(speed, 3),
   data = cars) ->
model

seq(0, 25, length.out = 25) ->
inputs_of_x

predict(model,
        data.frame(speed = inputs_of_x))
```

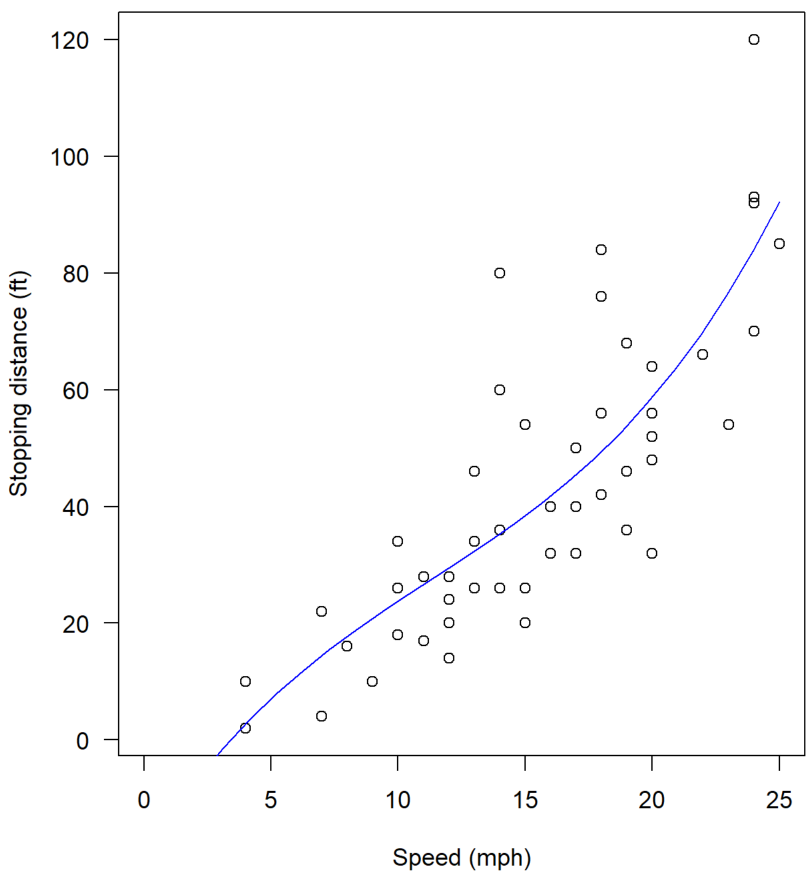



```
        1         2         3         4         5         6         7 
-19.505049 -12.788379 -6.760989 -1.353352  3.504057  7.880764 11.846296 
        8         9        10        11        12        13        14 
15.470179 18.821939 21.971103 24.987195 27.939743 30.898273 33.932311 
       15        16        17        18        19        20        21 
37.111382 40.505014 44.182731 48.214061 52.668530 57.615663 63.124987 
```

```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))

lm(dist ~ poly(speed, 3),
   data = cars) ->
model

seq(0, 25, length.out = 25) ->
inputs_of_x

predict(model,
        data.frame(speed = inputs_of_x)) ->
prediction_y
```



```r
## An example of polynomial regression
plot(cars, xlab = "Speed (mph)",
     ylab = "Stopping distance (ft)",
    las = 1, xlim = c(0, 25))

lm(dist ~ poly(speed, 3),
   data = cars) ->
model

seq(0, 25, length.out = 25) ->
inputs_of_x

predict(model,
        data.frame(speed = inputs_of_x)) ->
prediction_y

lines(inputs_of_x,
      prediction_y,
      col = "blue")
```



And arithmetic operation

```r
(4 + 5)
```

```
[1] 9
```

```r
(4 + 5) /
  6
```

```
[1] 1.5
```

```
(4 + 5) /
  6 *
  7
```

```
[1] 10.5
```

```r
(4 + 5) /
  6 *
  7 -
  3
```

```
[1] 7.5
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10
```

```
[1] 7.5

 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 33
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 4
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2

4
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2

[1] 4
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2

4 ^
  5
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2

[1] 1024
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2

4 ^
  5

matrix(1:4, ncol = 1)
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2

[1] 1024

     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
```

```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2

4 ^
  5

matrix(1:4, ncol = 1) %*%
  matrix(1:4, nrow = 1)
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2

[1] 1024

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16
```

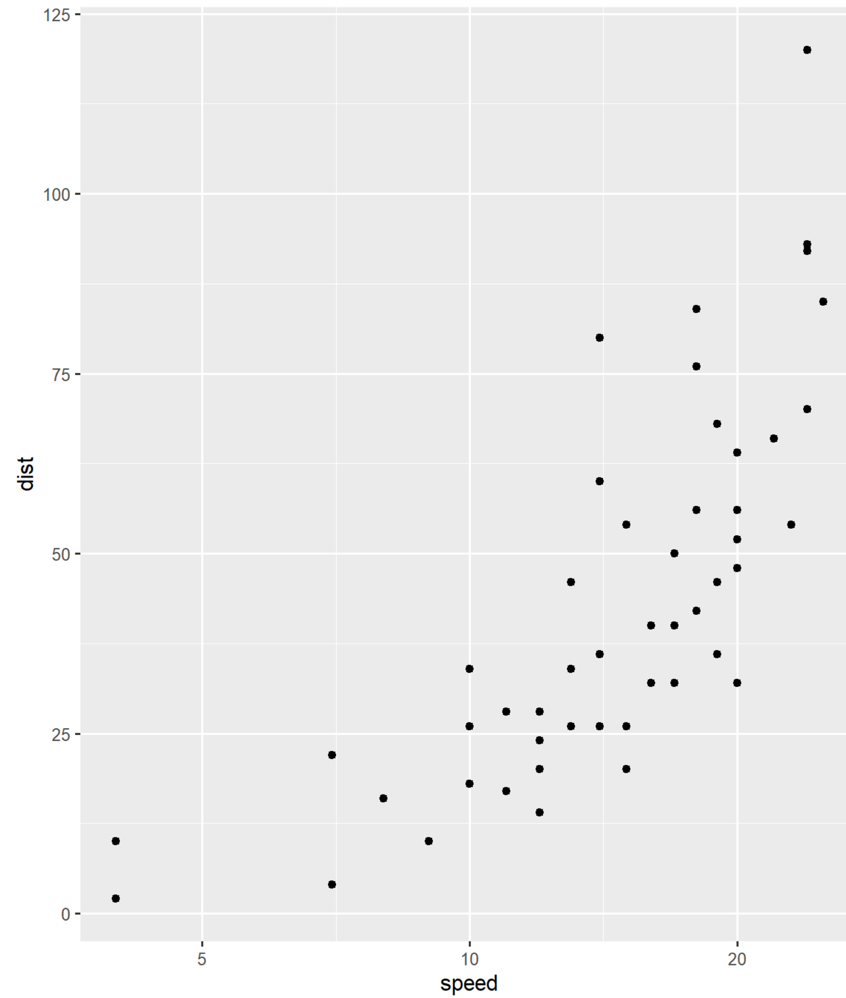
```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2

4 ^
  5

matrix(1:4, ncol = 1) %*%
  matrix(1:4, nrow = 1)

matrix(1:4, ncol = 4)
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2

[1] 1024

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
```

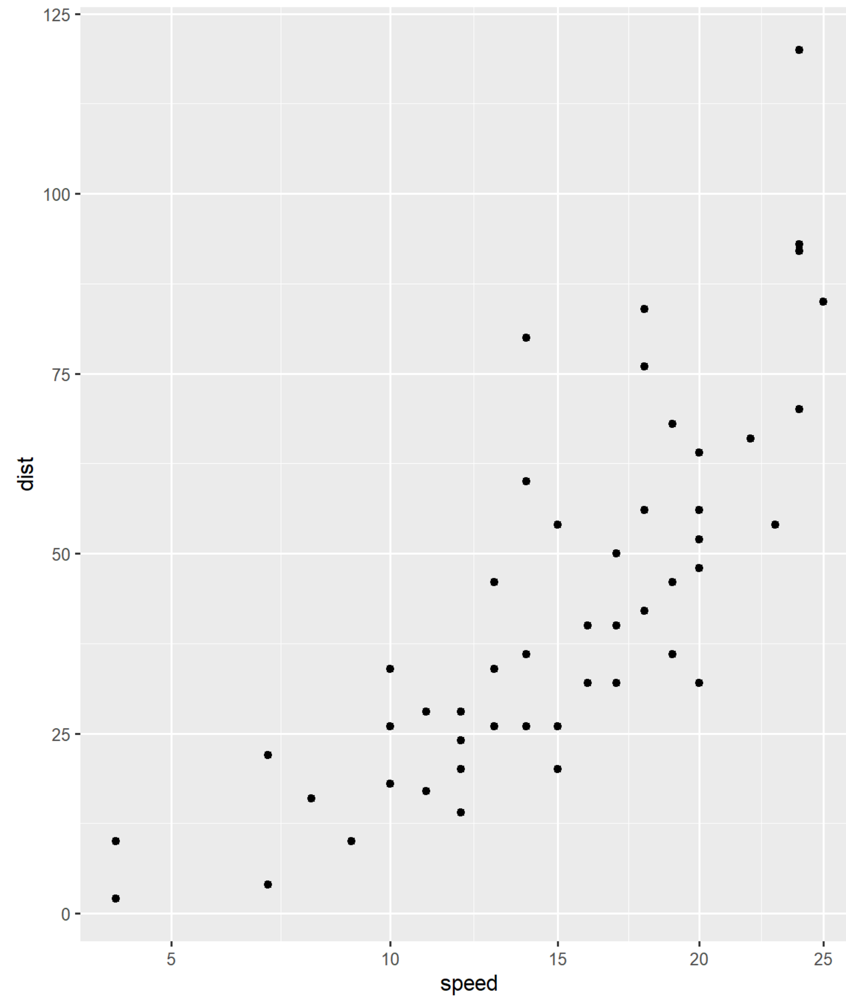
```r
(4 + 5) /
  6 *
  7 -
  3

1:10 %%
  3

1:10 %/%
  3

33 %%
  15

4 %/%
  2

4 ^
  5

matrix(1:4, ncol = 1) %*%
  matrix(1:4, nrow = 1)

matrix(1:4, ncol = 4) %*%
  matrix(1:4, nrow = 4)
```

```
[1] 7.5

 [1] 1 2 0 1 2 0 1 2 0 1

 [1] 0 0 1 1 1 2 2 2 3 3

[1] 3

[1] 2

[1] 1024

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16

     [,1]
[1,]   30
```

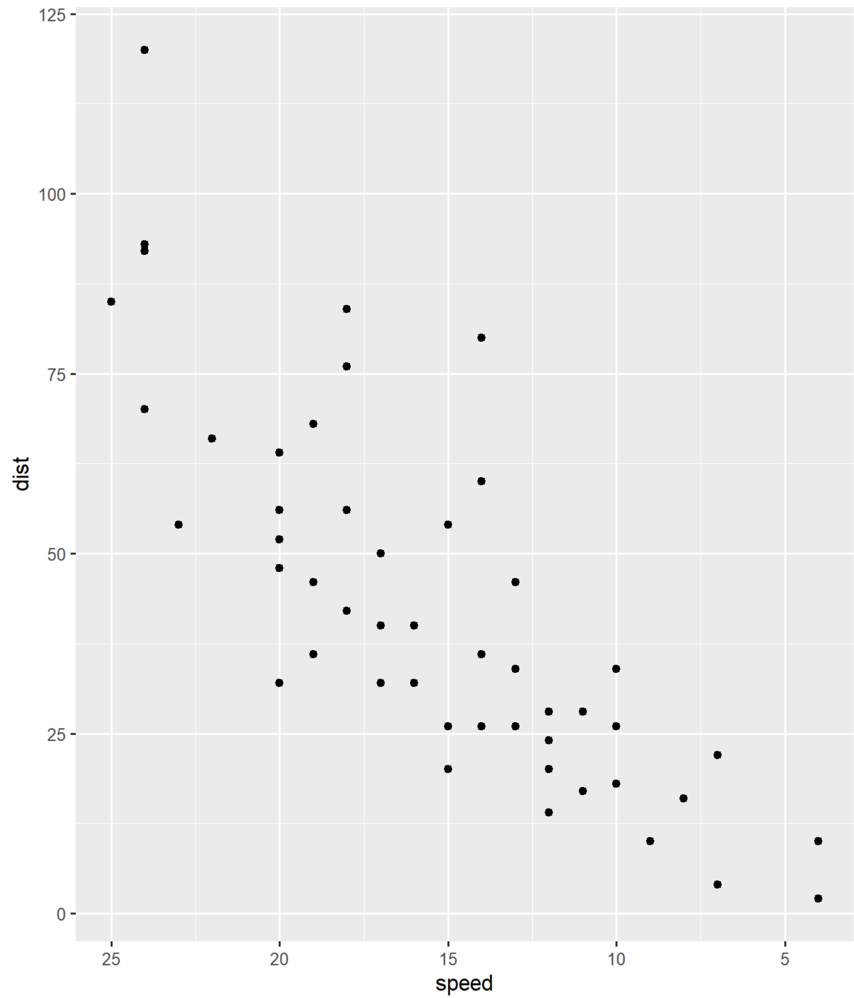
```
ggplot(data = cars) +
  aes(x = speed) +
  aes(y = dist) +
  geom_point() +
  scale_x_log10()
```



```
ggplot(data = cars) +
  aes(x = speed) +
  aes(y = dist) +
  geom_point() +
  scale_x_sqrt()
```



```
ggplot(data = cars) +
  aes(x = speed) +
  aes(y = dist) +
  geom_point() +
  scale_x_reverse()
```



A new addition is the %$% pipe from the magrittr library. And example follows.

```r
library(magrittr)
```

```r
library(magrittr)

cars
```

```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```r
library(magrittr)

cars %$%
  cor(x = speed,
      y = dist)
```

```
[1] 0.8068949
```

Custom Styling

Pipe to correlation coefficient

```
library(magrittr)
```

Pipe to correlation coefficient

```r
library(magrittr)

cars
```

```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
```

Pipe to correlation coefficient

```r
library(magrittr)

cars %$%
  cor(x = speed,
      y = dist)
```

```
[1] 0.8068949
```

hello

```r
library(magrittr)
```

goodbye

```
library(magrittr)

cars
```

$$\frac{\sum(1)}{2}$$

```r
library(magrittr)

cars %$%
  cor(x = speed,
      y = dist)
```

hello

goodbye

```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```
[1] 0.8068949
```

$$\frac{\sum(1)}{\mathbf{2}}$$

hello

goodbye

$$\frac{\sum(1)}{2}$$

```r
cars
```

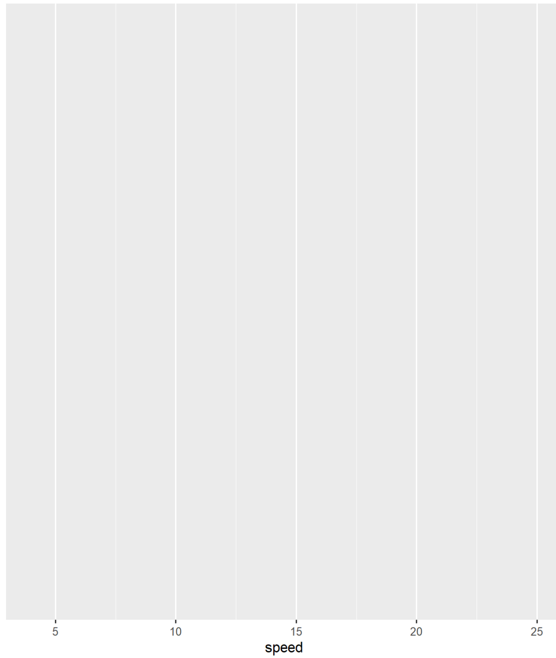
```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
```

```r
cars %>%
  ggplot()
```

```
function (data = NULL, mapping = aes(), ...
{
    UseMethod("ggplot")
}
<bytecode: 0x00000226c24fe700>
<environment: namespace:ggplot2>
```

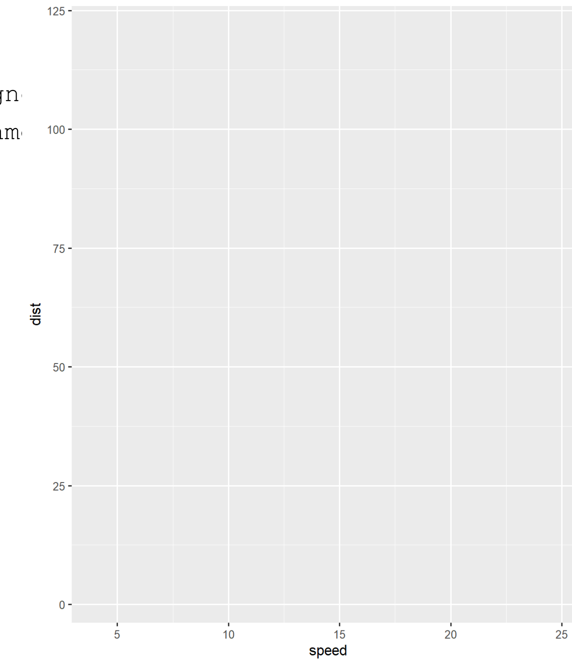
```r
cars %>%
  ggplot() +
  aes(x = speed)
```

```
function (x, y, ...)
{
    exprs <- enquos(x = x, y = y, ..., .ign
    aes <- new_aes(exprs, env = parent.fram
    rename_aes(aes)
}
<bytecode: 0x00000226c0cede20>
<environment: namespace:ggplot2>
```



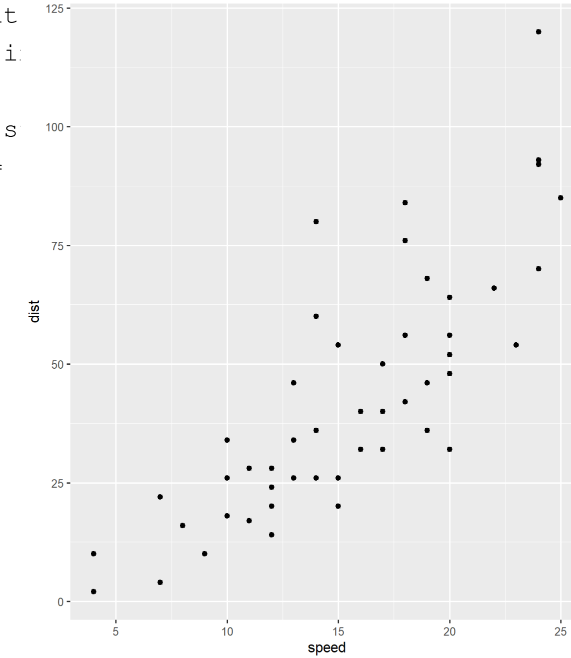
```
cars %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist)
```

```
function (x, y, ...)
{
    exprs <- enquos(x = x, y = y, ..., .ign
    aes <- new_aes(exprs, env = parent.fram
    rename_aes(aes)
}
<bytecode: 0x00000226c0cede20>
<environment: namespace:ggplot2>
```



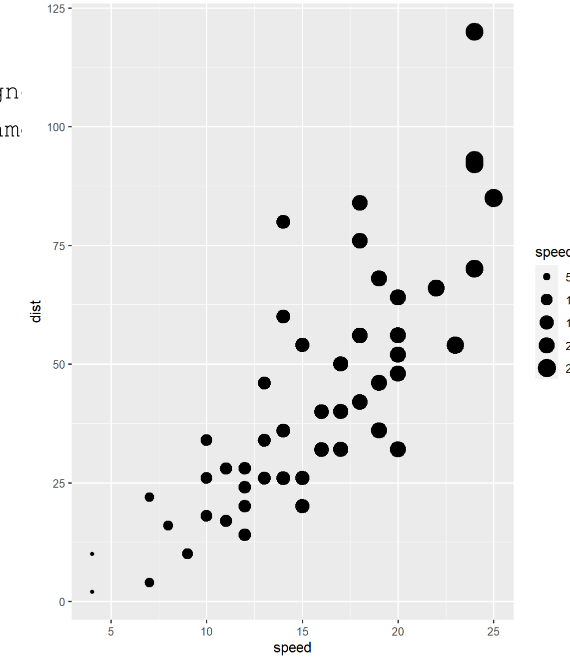
```
cars %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point()
```

```
function (mapping = NULL, data = NULL, stat
    ..., na.rm = FALSE, show.legend = NA, i
{
    layer(data = data, mapping = mapping, s
        position = position, show.legend =                    s,
        params = list(na.rm = na.rm, ...))
}
<bytecode: 0x00000226c2357670>
<environment: namespace:ggplot2>
```



```
cars %>%
  ggplot() +
  aes(x = speed) +
  aes(y = dist) +
  geom_point() +
  aes(size = speed)
```

```
function (x, y, ...)
{
    exprs <- enquos(x = x, y = y, ..., .ign
    aes <- new_aes(exprs, env = parent.fram
    rename_aes(aes)
}
<bytecode: 0x00000226c0cede20>
<environment: namespace:ggplot2>
```



Slow Message

This

is

my

text.

'One driver of equality we should invest in is upskilling everyone - not just the select few.'

digital inclusion.

Xaringan slide show look and feel

To quickly change the look and feel of your {xaringan} slide show, you might check out the available themes from the xaringan package and xaringanthemer package.

Another extremely useful resource for xaringan styling is Alison Hill's "Meet xaringan: Making slides in R Markdown".

Sharing your flipbooks

Flipbooks created with Xaringan are multi-file creations. The figures produced are stored separately from the main html document. This presents a little bit of a challenge for sharing your work. You can zip up all the associated files and share that way. Alternatively, you can share as a website. I've shared my work on github with github pages.

- flip, zip, and ship
- get it on github, with github pages. A good walk through is the one that I learned with (to get the ggplot2 flipbook online) by Brian Caffo https://www.youtube.com/watch?v=BBCesiebEuQ Larger flipbooks will take longer to load online - something to keep in mind as you are building.

The flipbooked portion of this presentation was created with the new {flipbookr} package. Get it with remotes::install_github("EvaMaeRey/flipbookr")