Miscellanea

Packages needed for this section

```
library(tidyverse)
library(ggrepel)
```

How do you read a file like this?

```
rat1T 8
rat2T11
rat3C 7
rat4C 4
rat5T12
```

- Columns:
  - 1st 4 columns are ID of rat
  - next 1 column is T (treatment) or C (control)
  - next 2 columns are value of response variable `y`.
- No delimiters!
- Each data value *same width*.
- `read_fwf`.

Reading in delimiterless data

- `read_fwf`, vector of widths and names for columns:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/c32/rats7.txt"
rat7 <- read_fwf(my_url, fwf_widths(
  c(4, 1, 2),
  c("id", "group", "y")
))
```

```
##
## -- Column specification --------------------------
## cols(
##   id = col_character(),
##   group = col_character(),
##   y = col_double()
## )
```

The "rat7" data

```
rat7
```

id	group	y
rat1	T	8
rat2	T	11
rat3	C	7
rat4	C	4
rat5	T	12

- Note that `read_fwf` determined that `y` was a number and the other things were text.
- You need to have a separate document telling you how many characters each column is.

Plotting series: The oranges data

```
my_url <- "http://www.utsc.utoronto.ca/~butler/c32/oranges.txt
oranges <- read_delim(my_url, " ")

##
## -- Column specification --------------------------------
## cols(
##   age = col_double(),
##   A = col_double(),
##   B = col_double(),
##   C = col_double(),
##   D = col_double(),
##   E = col_double()
## )
```

The data

```
oranges
```

age	A	B	C	D	E
118	30	30	30	33	32
484	51	58	49	69	62
664	75	87	81	111	112
1004	108	115	125	156	167
1231	115	120	142	172	179
1372	139	142	174	203	209
1582	140	145	177	203	214

- These are circumferences of five different trees at seven different ages (days).

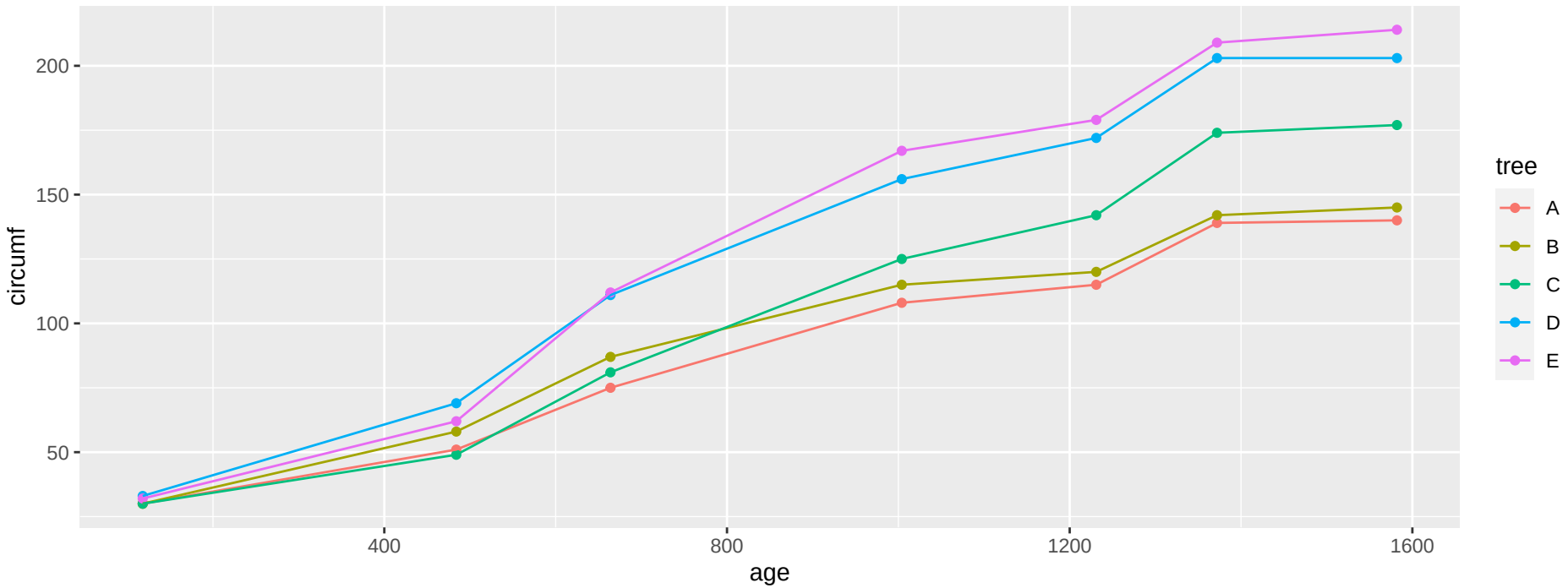
Plotting series

- Want to plot orange tree circumferences against age for each orange tree.
- Recall ggplot wants one column of x values and one column of y values, which we do not have.
- Gather up columns A through E, which are different trees but all circumferences.
- Then construct a plot (shown over):

```
g <- oranges %>%
  gather(tree, circumf, A:E) %>%
  ggplot(aes(x = age, y = circumf, colour = tree)) +
  geom_point() + geom_line()
```

The plot

```
g
```



Labelling points on a plot

- My car data

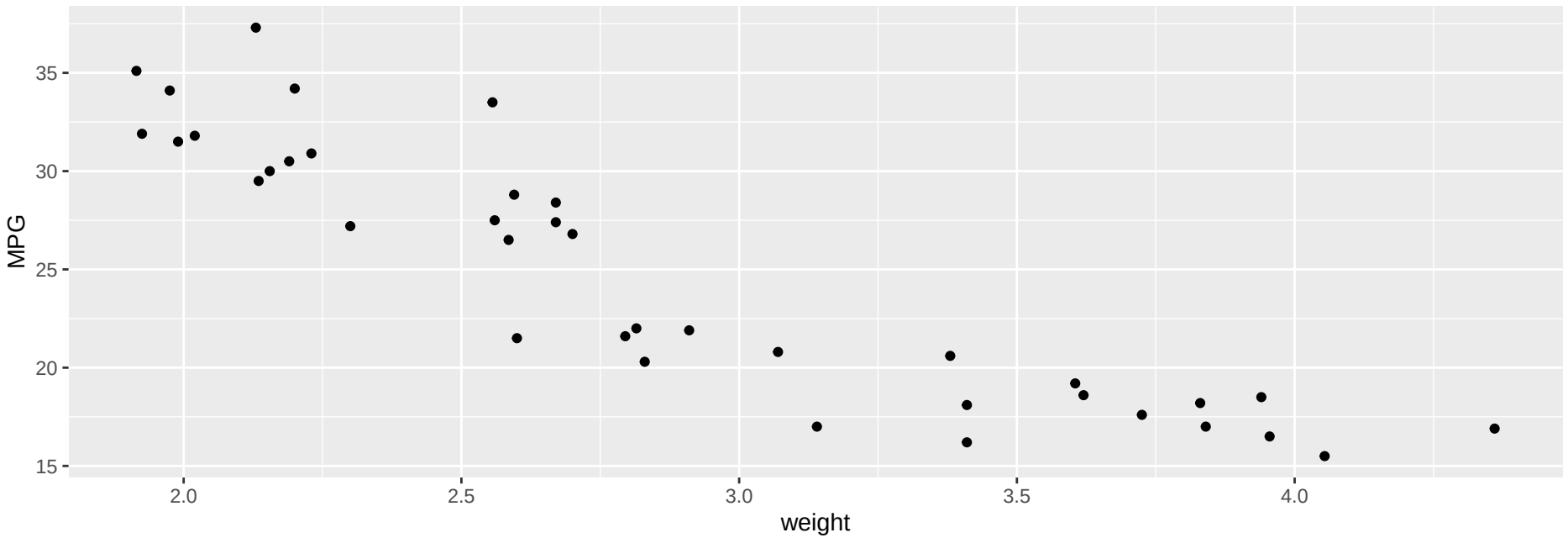
```
my_url <- "http://www.utsc.utoronto.ca/~butler/c32/cars.csv"
cars <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------
## cols(
##   car = col_character(),
##   MPG = col_double(),
##   weight = col_double(),
##   cylinders = col_double(),
##   hp = col_double(),
##   country = col_character()
## )
```

- Names of and information about 38 models of car (from a US car magazine)

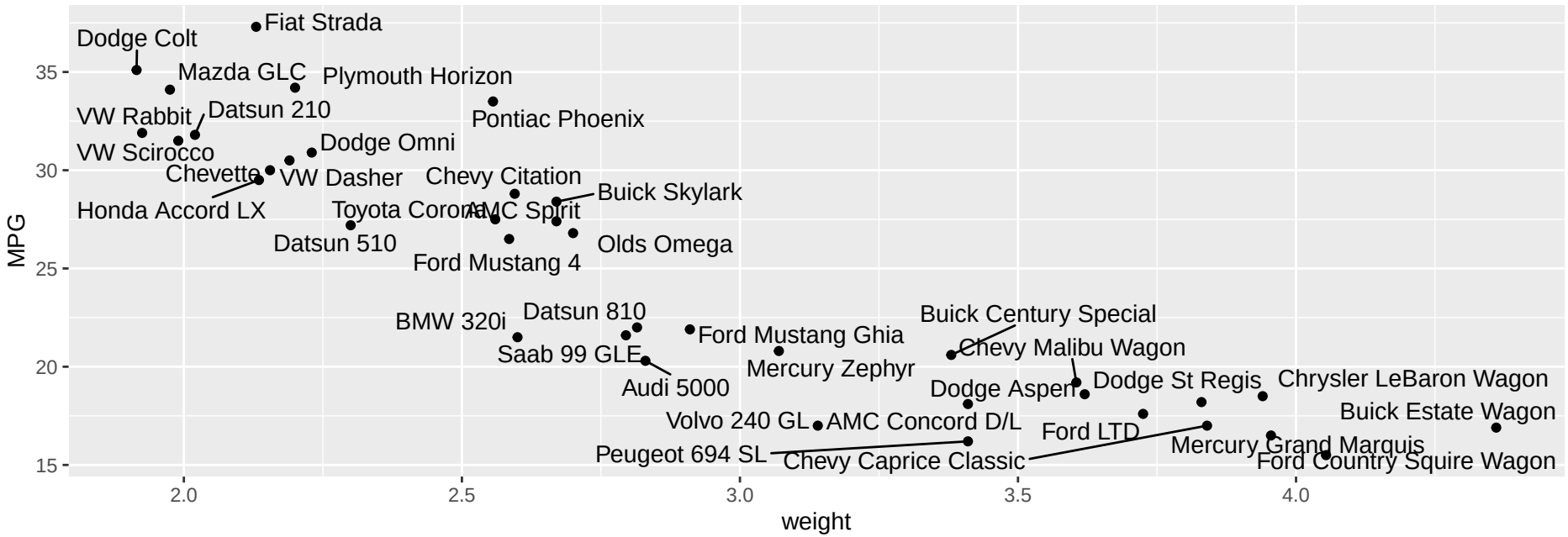
Plot gas mileage against weight

```
ggplot(cars, aes(x = weight, y = MPG)) +
  geom_point()
```



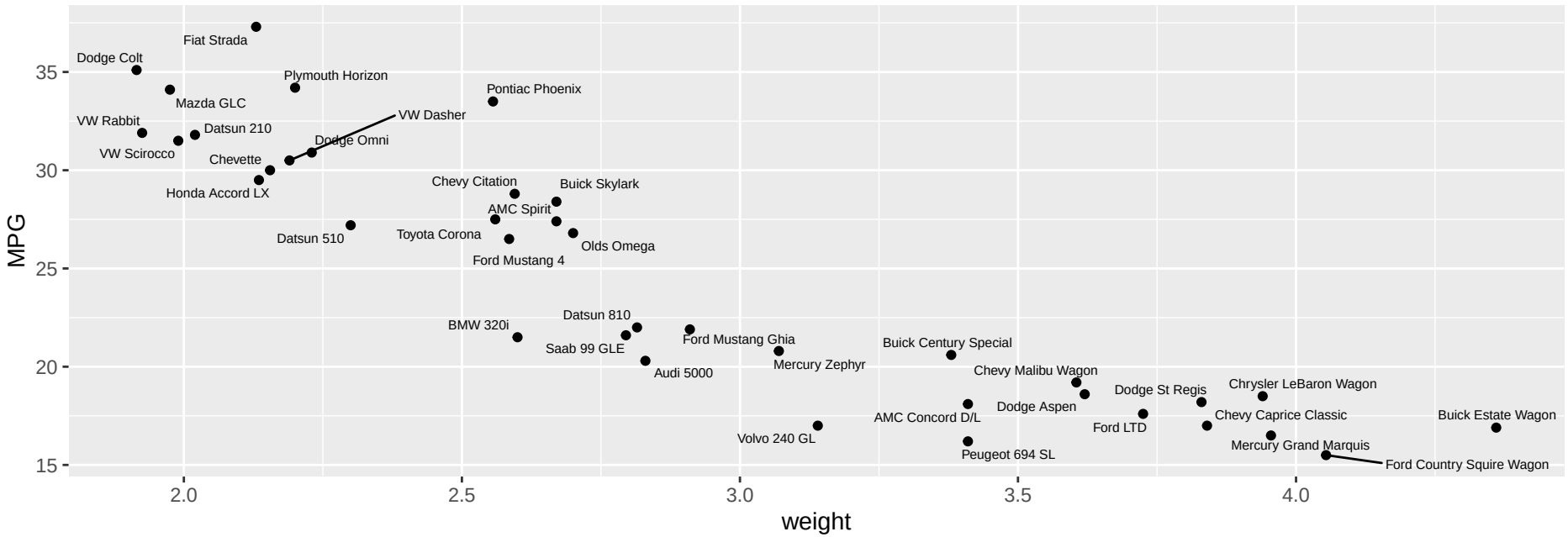
Label points with name of car they belong to

```
ggplot(cars, aes(x = weight, y = MPG, label = car)) +
  geom_point() + geom_text_repel()
```



Make labels smaller

```
ggplot(cars, aes(x = weight, y = MPG, label = car)) +
  geom_point() + geom_text_repel(size = 2)
```



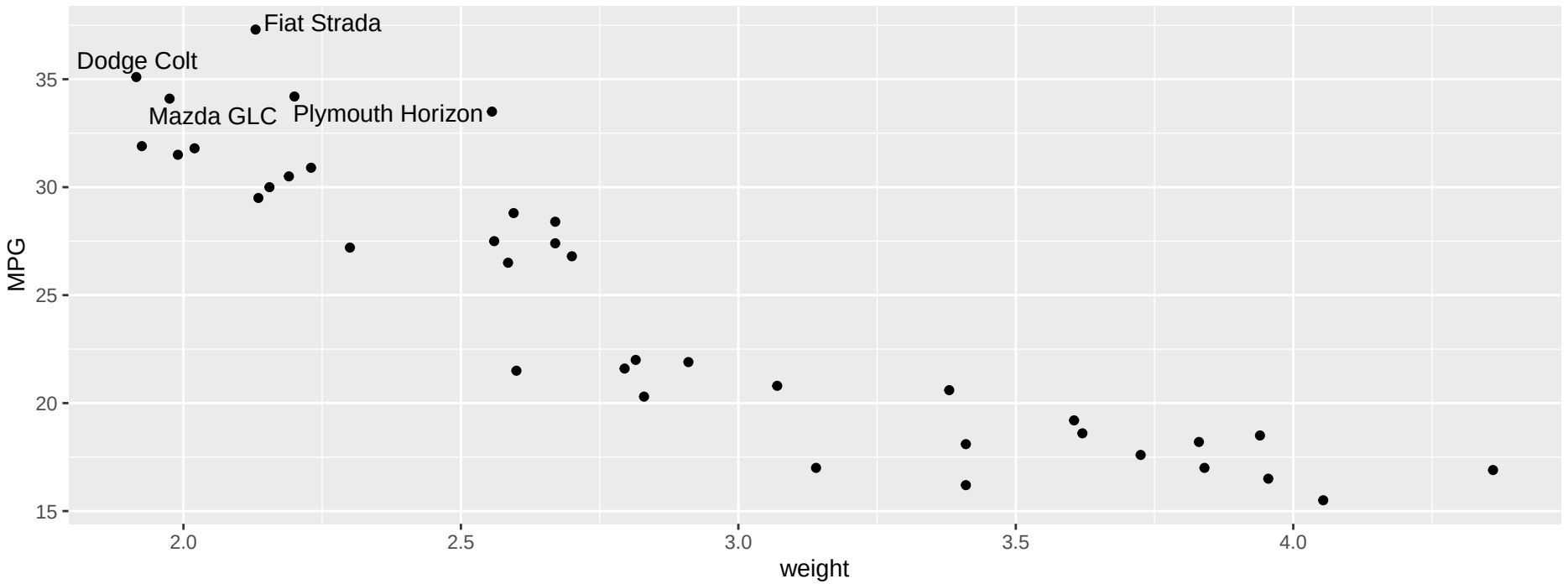
Labelling some of the cars

- Maybe you want to draw attention only to some of the individuals
  - for example labelling only certain cars or ones that satisfy a condition
- Mechanism: define a new label variable that contains:
  - the label, for the individual you want to label
  - blank text for those you don't
- Handy function ifelse, like Excel `=IF`.
- Label cars with MPG over 34:

```
cars %>%
  mutate(newlabel = ifelse(MPG > 34, car, "")) %>%
  ggplot(aes(x = weight, y = MPG, label = newlabel)) +
  geom_point() +
  geom_text_repel() -> g
```

The plot

```
g
```


Fiat Strada
Dodge Colt
Mazda GLC
Plymouth Horizon
MPG
35
30
25
20
15
2.0
2.5
3.0
3.5
4.0
weight

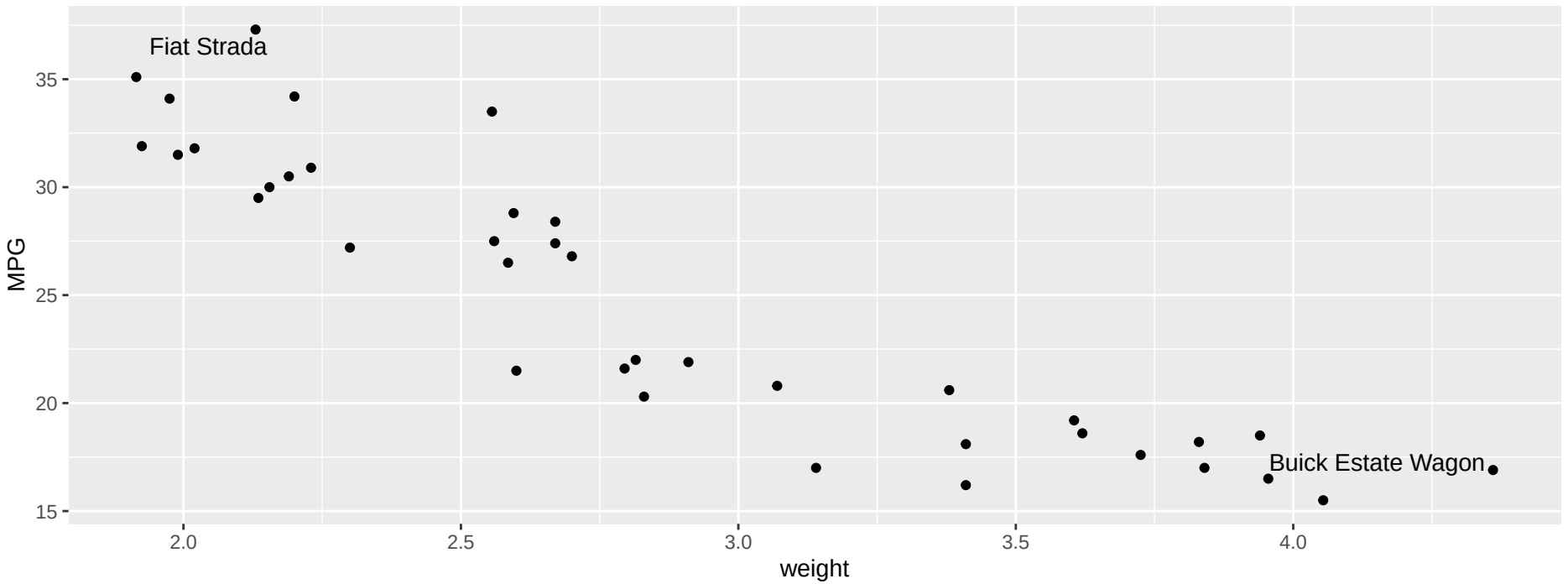
Labelling cars by row number

- Suppose we knew that the cars we wanted to label were in rows 4 and 9 of data frame.
- How to use `ifelse` with row numbers? Define new column of row numbers, and then use it in `ifelse`, thus:

```
g <- cars %>%
  mutate(row = row_number()) %>%
  mutate(newlabel = ifelse(row == 4 | row == 9, car, "")) %>%
  ggplot(aes(x = weight, y = MPG, label = newlabel)) +
  geom_point() +
  geom_text_repel()
```

The plot

```
g
```


Fiat Strada
Buick Estate Wagon
MPG
35
30
25
20
15
2.0
2.5
3.0
3.5
4.0
weight

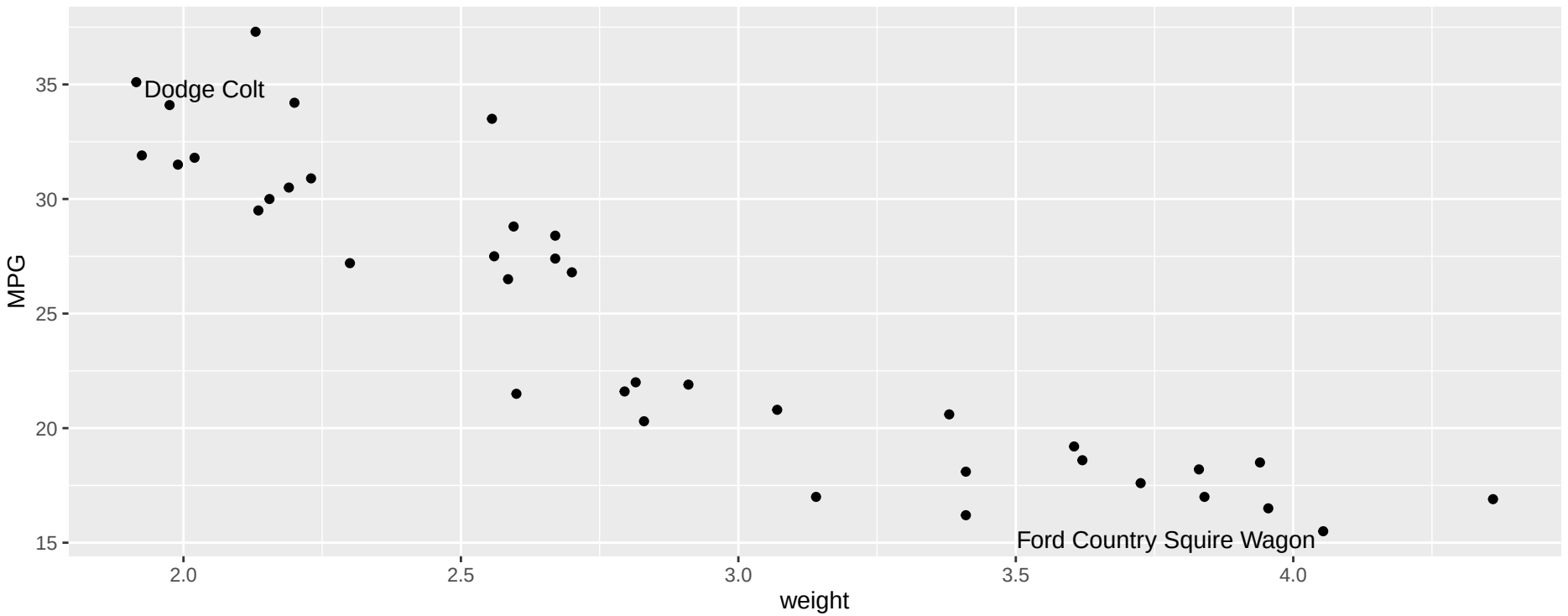
Lightest weight and worst gas-mileage cars

- Suppose you didn't know which cars were the ones you wanted. Then you have to find them first.
- Now try for lightest weight and worst gas-mileage cars:

```
cars %>%
  mutate(tolabel = (weight == min(weight) |
    MPG == min(MPG))) %>%
  mutate(newlabel = ifelse(tolabel, car, "")) %>%
  ggplot(aes(x = weight, y = MPG, label = newlabel)) +
  geom_point() +
  geom_text_repel() -> g
```

The plot

```
gg
```

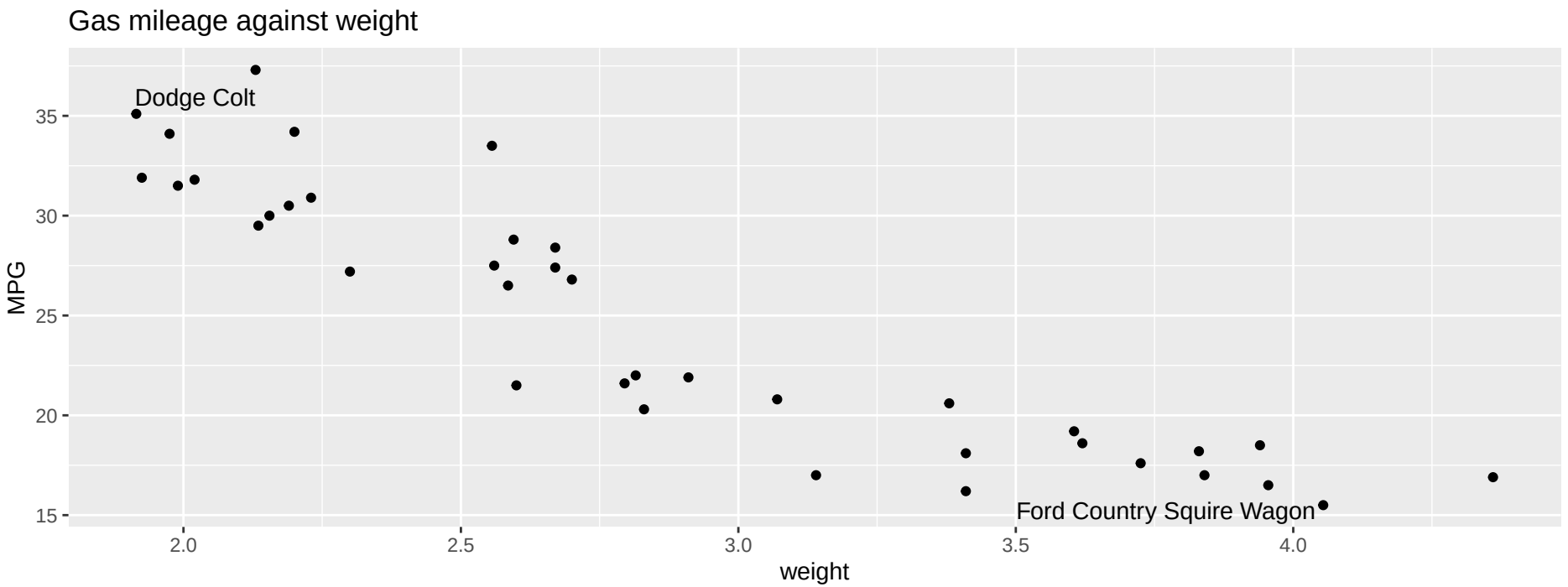

Dodge Colt
Ford Country Squire Wagon
MPG
35
30
25
20
15
2.0
2.5
3.0
3.5
4.0
weight

Miscellaneous graph things

- Title for graph
- Axis labels
- We use previous graph as base (to save drawing again).

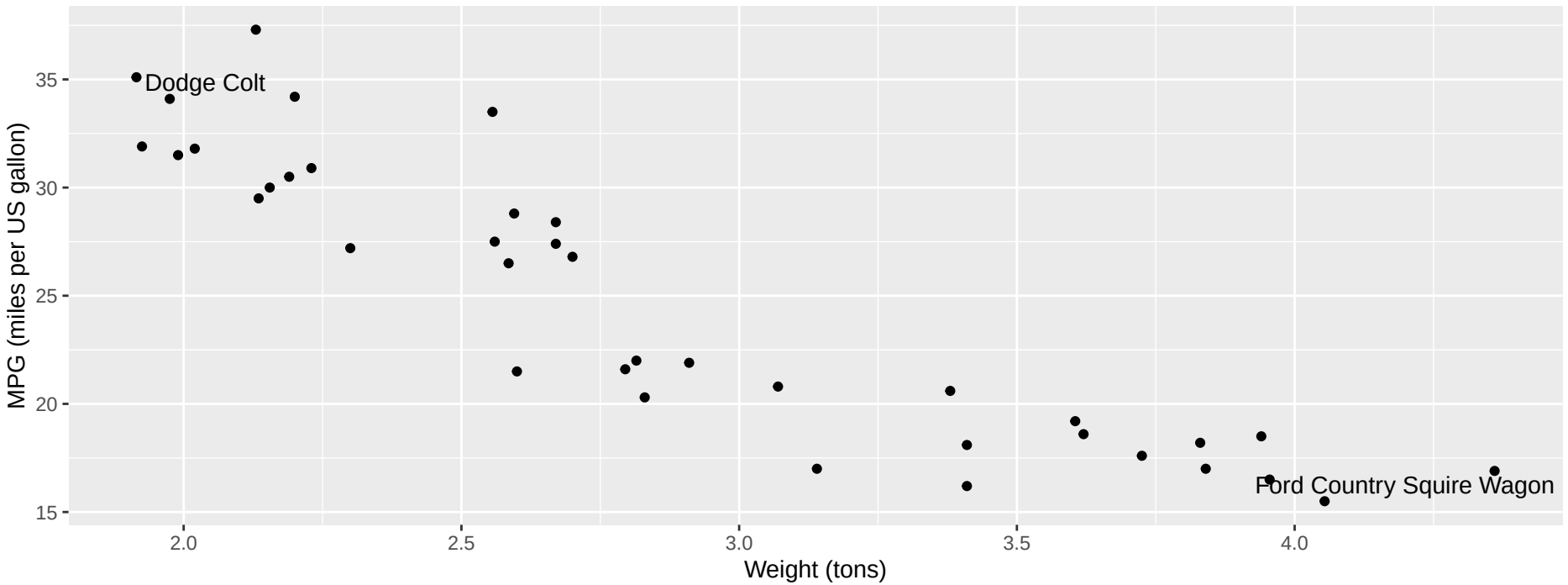
With title

```
g + ggtitle("Gas mileage against weight")
```



Axis labels

```
g + xlab("Weight (tons)") + ylab("MPG (miles per US gallon)")
```



Permanence

- When you close R Studio, you are offered the option to "save your workspace". If you choose "yes", all of the data frames and other things you have created are saved, so that when you open R Studio in the same project later, you will be able to access all of these things. ("Everything is permanent" in that sense.)
- If you choose not to save your workspace, you will have to recreate all your objects next time (eg. re-read data from files). But you have a script to do that, don't you?
- There is a school of thought that says you should not save your workspace, but keep scripts to re-create everything.
  - Pro: keeps your workspace "clean" of old objects that you created but don't need any more, and you know exactly why everything is there.
  - Con: some objects take time and effort to re-create, and you won't want to do that every time.

Saving and restoring objects

- It is possible to save and re-load large/complicated objects so that they don't have to be re-created. Uses `saveRDS` and `readRDS`:

```
xx <- sample(1:10, 5)
xx
```

```
## [1] 7 3 1 2 5
```

```
saveRDS(xx, "xx.rds")
rm(xx)
xx # gone
```

```
## Error in eval(expr, envir, enclos): object 'xx' not found
```

```
xx <- readRDS("xx.rds")
xx # back
```

```
## [1] 7 3 1 2 5
```