Working with 

A University of Queensland Advanced Workshop

Session 3: Graphics

Bill Venables, CSIRO/Data61, Dutton Park

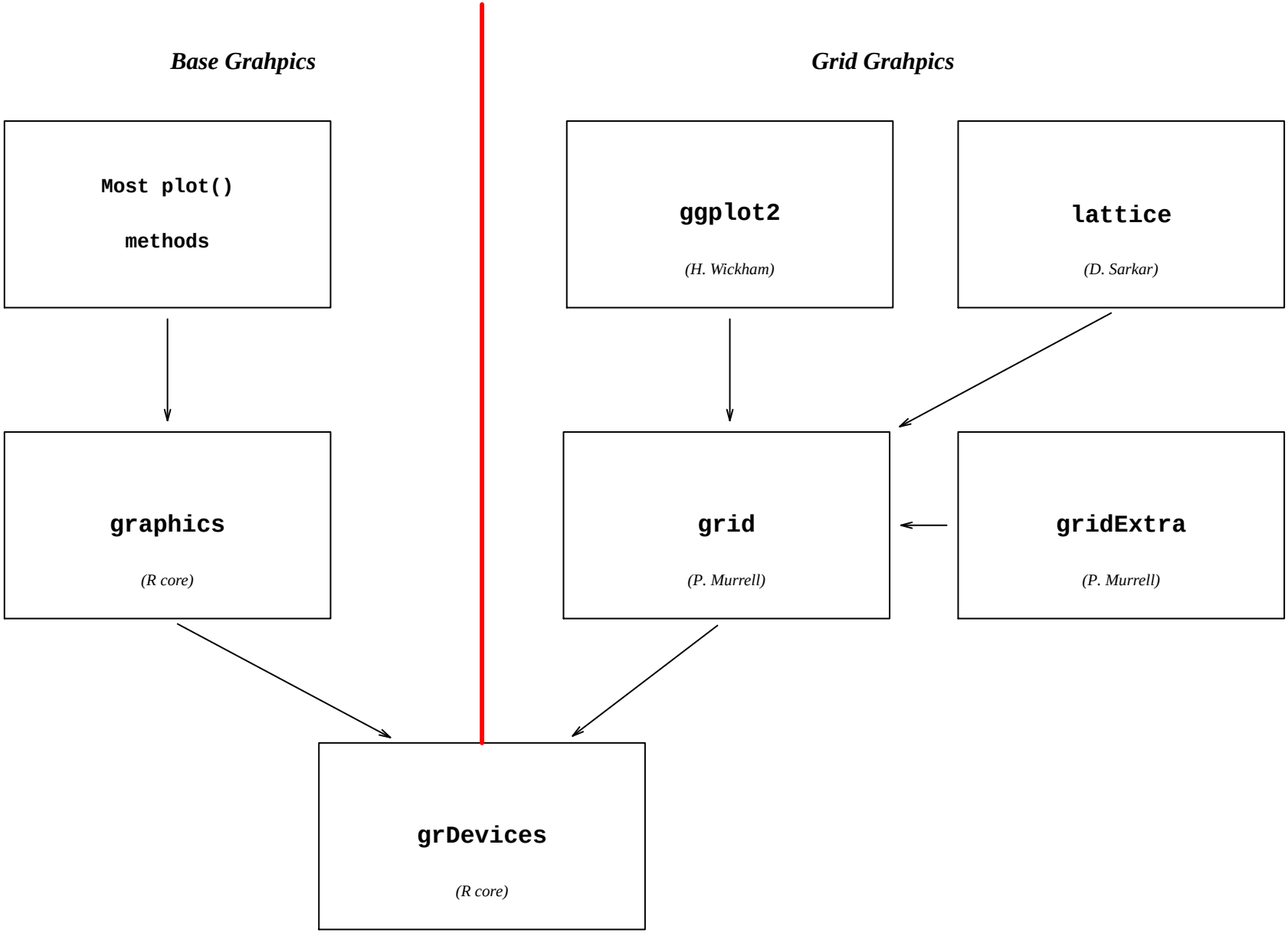
Rhetta Chappell, Griffith University

2–5 February, 2021

Contents

1 Graphics in R

- The *grDevices* package handles all communication with external graphics devices, as well as providing some useful tools and resources
- Traditional, or *base* graphics uses the *graphics* package, which communicates directly with *grDevices* and outputs to the open graphics channel immediately.
- The *grid* package provides a series of tools to construct *graphical objects*, for which the standard *print* method leads to graphical, rather than printed, output.
- Modern graphical packages, such as *lattice* and *ggplot2* work through *grid*: they produce objects for which only printing produces graphical output.


Base Grahpics
Grid Grahpics
Most plot()
methods
ggplot2
(H. Wickham)
lattice
(D. Sarkar)
graphics
(R core)
grid
(P. Murrell)
gridExtra
(P. Murrell)
grDevices
(R core)

1.1 Some differences

- *Traditional graphics* use "The Painter's model": output is produced immediately as a side-effect of the function call. The only way to change the graphic is to re-run the script from scratch. The value returned by the function is not printed and often of no interest.

  Hence functions like *plot*, *lines*, *points*, &c work equally well when called within other functions as they do when called from the command line.

- *grid-based graphics* usually have *no* side effects; calls to functions build up a *graphical object* to guide the graphic when eventually the object is *print*ed.

  In *ggplot2*, (and to a lesser extent in *lattice*), objects are built up *incrementally* using an 'overloaded' plus (+) operator.

  Hence *ggplot* and *lattice* plots, if called within functions either have to return the graphics object for later printing, or call the *print* function explicitly within the function itself (NOT recommended!)

2 Grammar of graphics

2.1 Some Terminology

Aesthetics Properties of the display which can be used to represent variables to be displayed. $x-$ and $y-$axes are primary, but *colours, sizes, shapes, grouping*, and many others are available

Geoms Devices used to establish the association between variables to be displayed.

Stats Nothing to do with statistics. Transformations of the data to allow the modify, or specify, the precise form of the association.

Most *geom*s imply a default *stat*; conversely most *stat*s imply a default *geom*.

Scales A control mechanism for aesthetics such as *colours, 'fills', sizes, shapes*, to establish a desired effect.

Guides Devices for meta-information, such as legends.

Faceting The partition of a display into a gridded set of panels, each displaying one section of the data, usually with a related context.

Themes Background properties of the display; the 'look and feel' properties independent of the information being displayed.

2.2 The structure of a *ggplot* object

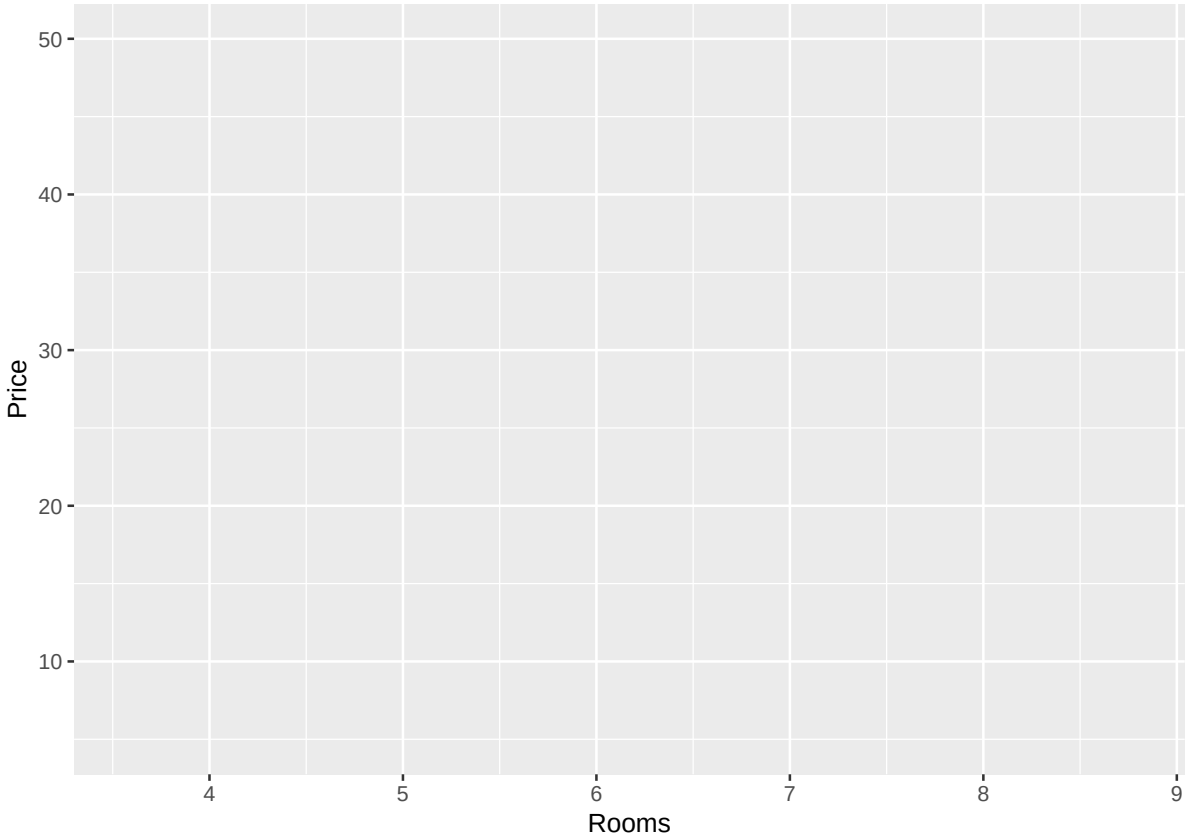
A *ggplot2* object starts with a call to the function *ggplot()* (*not* *ggplot2()*) which usually contains the name of the data frame from which most variables used in the display are to come. (It is not necessary to use variables in a data frame, but it is highly recommended you do so.)

We build an example in small increments to see the process.

```r
boston <- Boston %>% ## ?Boston gives the full documentation
  within({
    Riverside <- recode(chas, `0` = "No", `1` = "Yes")
    Poverty <- sqrt(lstat)
  }) %>%
  select(Price = medv, Rooms = rm, Poverty, Riverside) ## raname
gg_object <- ggplot(boston)  ## embryonic ggplot object;
gg_object + theme_grey()     ## blank canvas at this stage
```

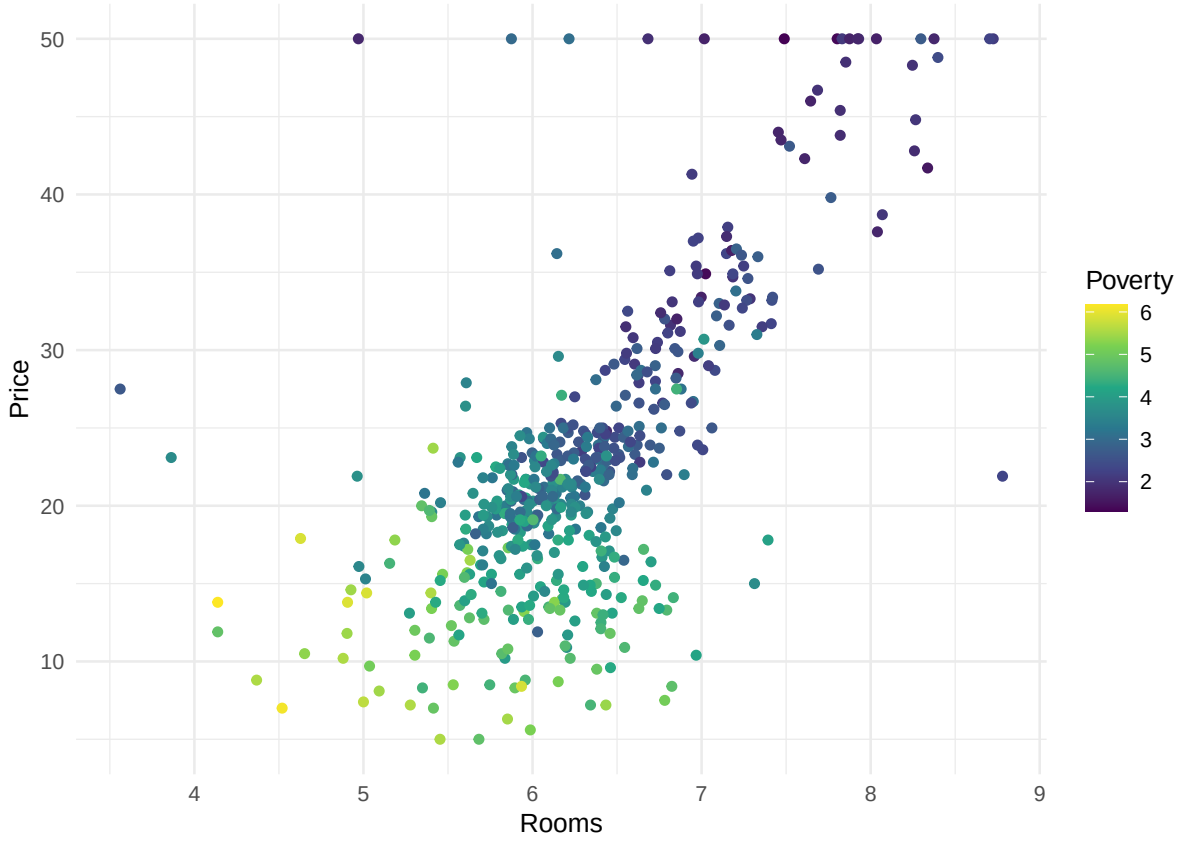
What happens when we nominate something for the axes?

```
gg_object <- gg_object + aes(x = Rooms, y = Price)
gg_object + theme_grey()  ## we get an axis system, but ...
```



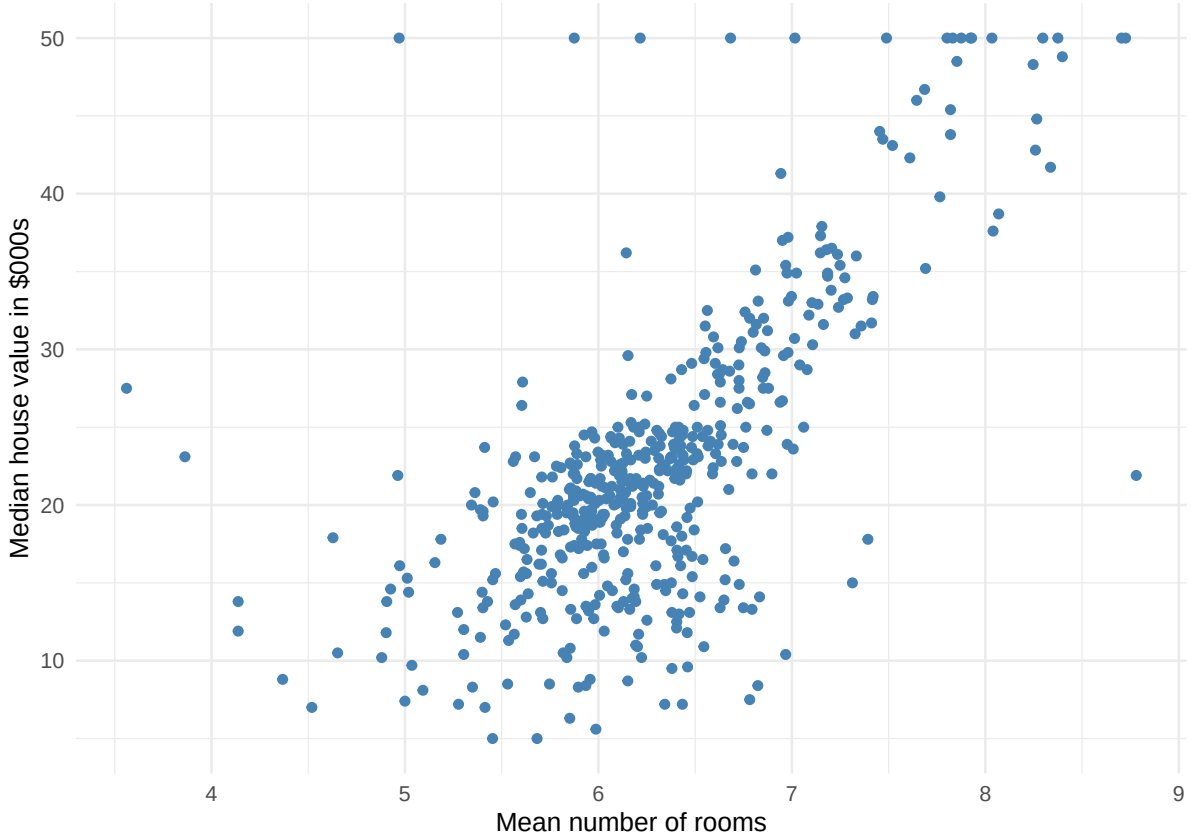
Now stipulate a few more details:

```
gg_object <- gg_object + aes(colour = Poverty)        ## use colour, but how?
gg_object + geom_point() + scale_colour_viridis_c()   ## 'colour-blind friendly'
```



Step back and do a simple plot that says something

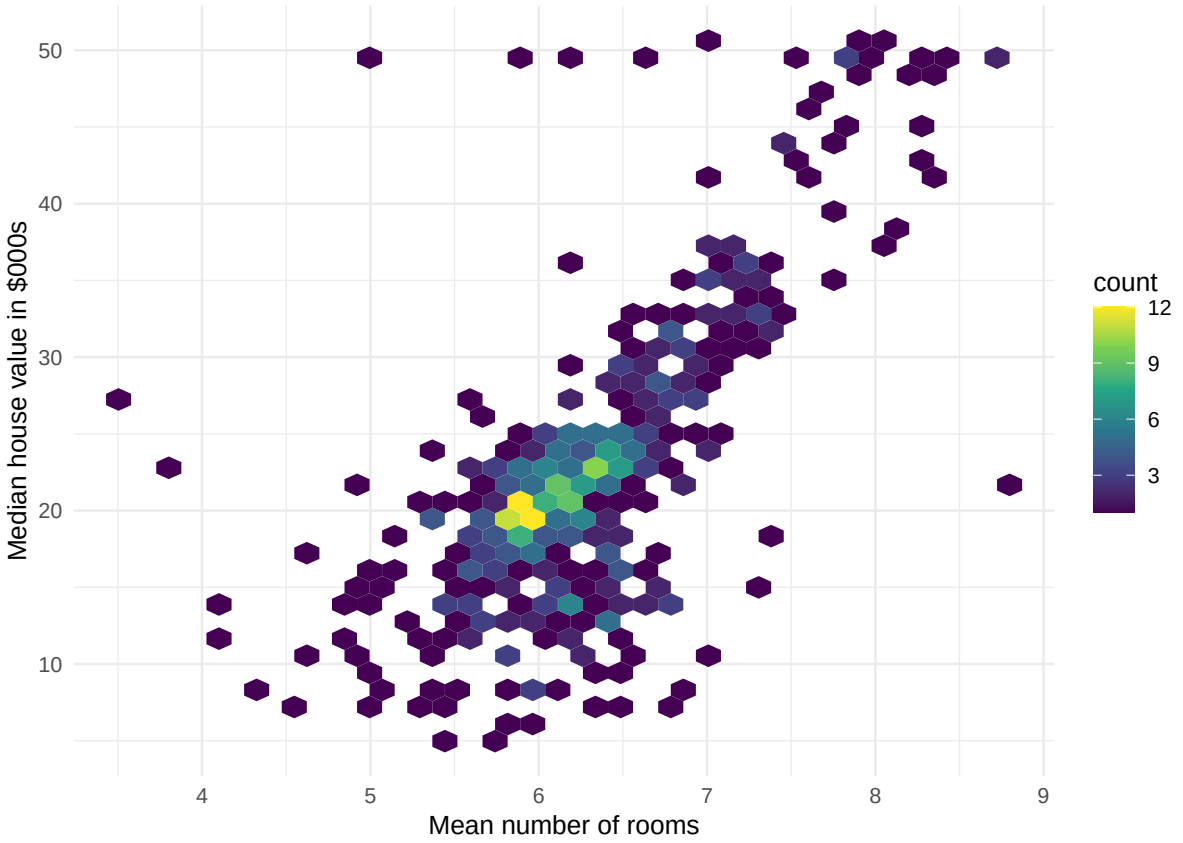
```
ggplot(boston) + aes(x = Rooms, y = Price) + geom_point(colour = "steelblue") +
  labs(x = "Mean number of rooms", y = "Median house value in $000s")
```



The object has not been saved, but has been printed and discarded.

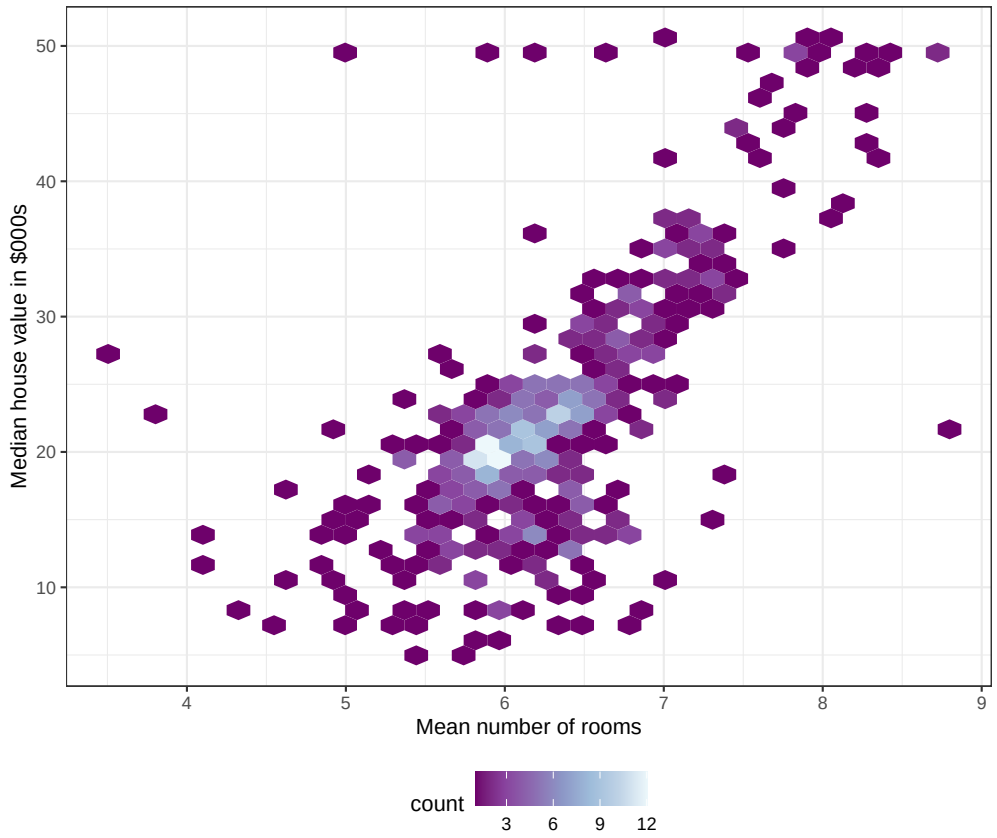
See how *stat*s and *geom*s can be used as alternatives.

```
gg_object <- gg_object + labs(x = "Mean number of rooms",
                              y = "Median house value in $000s")
gg_object + geom_hex(bins = 35) + scale_fill_viridis_c()
```



Now a similar plot using *stat* and implying *geom*:

```
gg_object + stat_bin_hex(bins = 35) + scale_fill_distiller(palette = 3) +
  theme_bw() + theme(legend.position = "bottom")
```



2.3 A cars example

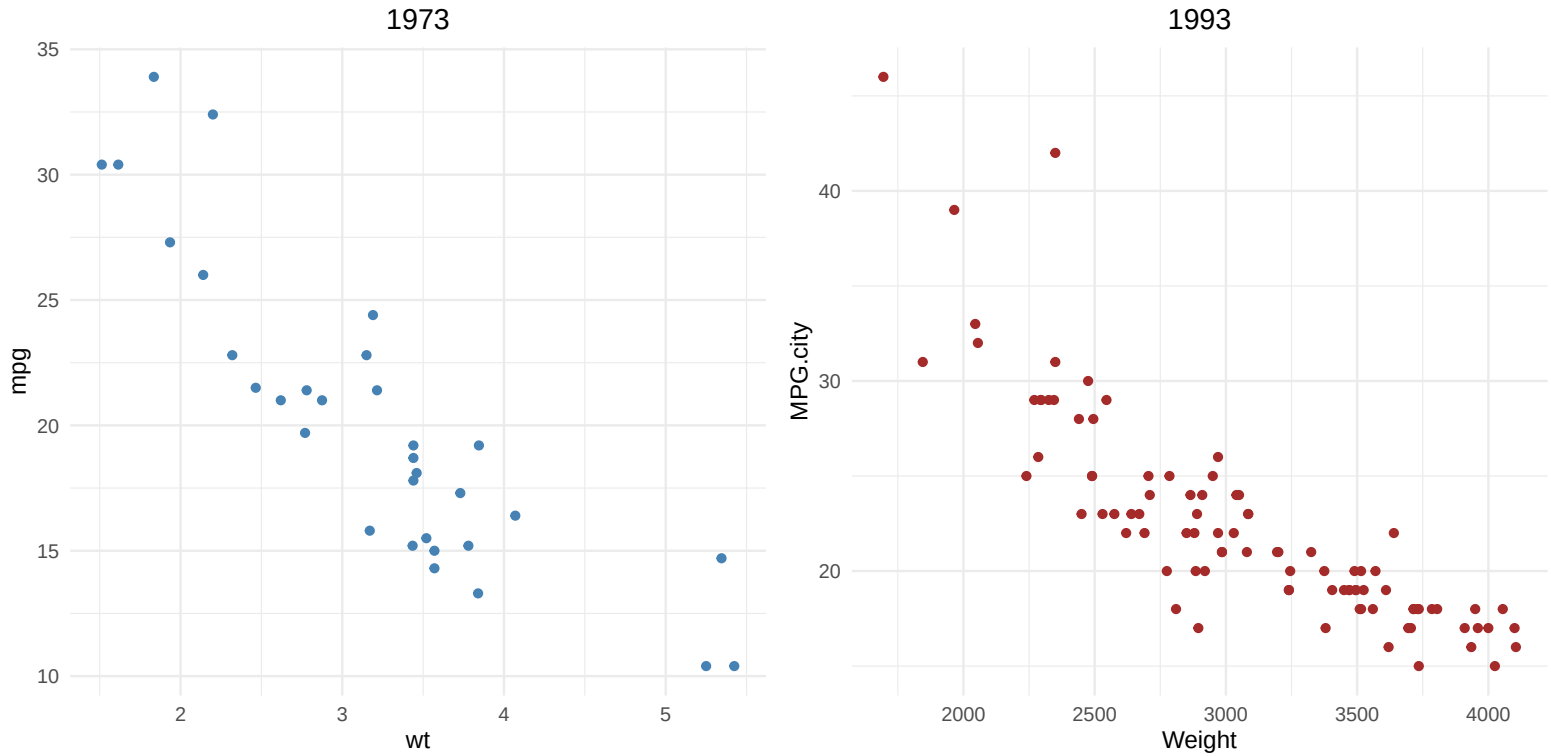
The oil crisis of 1973 led to a sudden and extreme rise in the price of petrol in the USA, particularly, and in turn led to greater attention to the fuel economy of new cars. The *datasets* package in **R** also has a data set, *mtcars*, of data from models released in 1973. The *MASS* package has a data set *Cars93* containing data from models released in 1993. Our job will be compare the fuel performance of vehicles from these two data sets.

```
sapply(c("mtcars", "Cars93"), find) %>% noquote
```

```
          mtcars           Cars93
package:datasets  package:WWRData
```

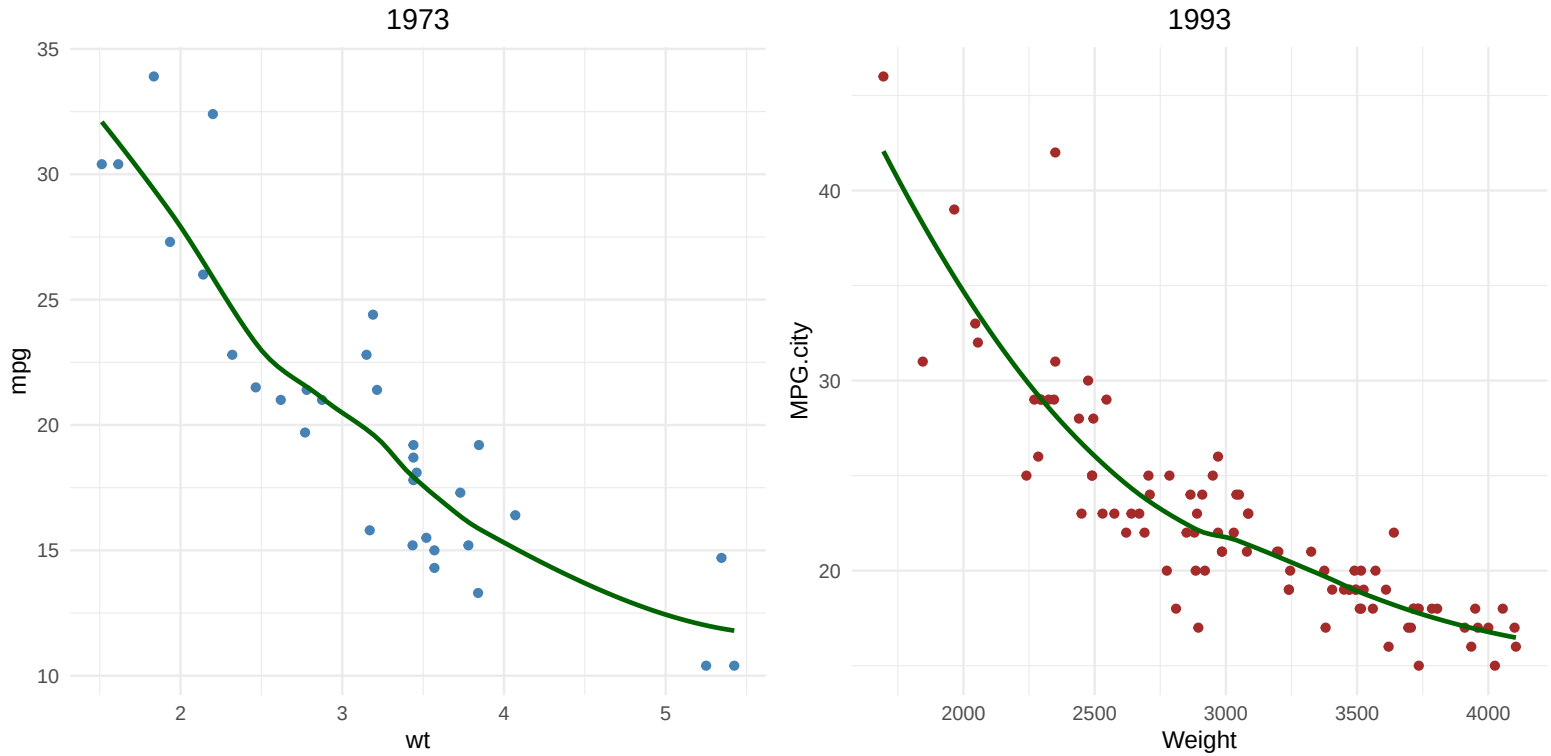
First look at the marginal relationship between miles/gallon and weight:

```
# library(gridExtra)  ## library(patchwork) is an alternative
plt73 <- ggplot(mtcars) + aes(x = wt, y = mpg) +
  geom_point(colour = "steelblue") + labs(title="1973")
plt93 <- ggplot(Cars93) + aes(x = Weight, y = MPG.city) +
  geom_point(colour = "brown") + labs(title="1993")
plt73 + plt93
```



Adding a smoother allows the pattern to be more clearly seen:

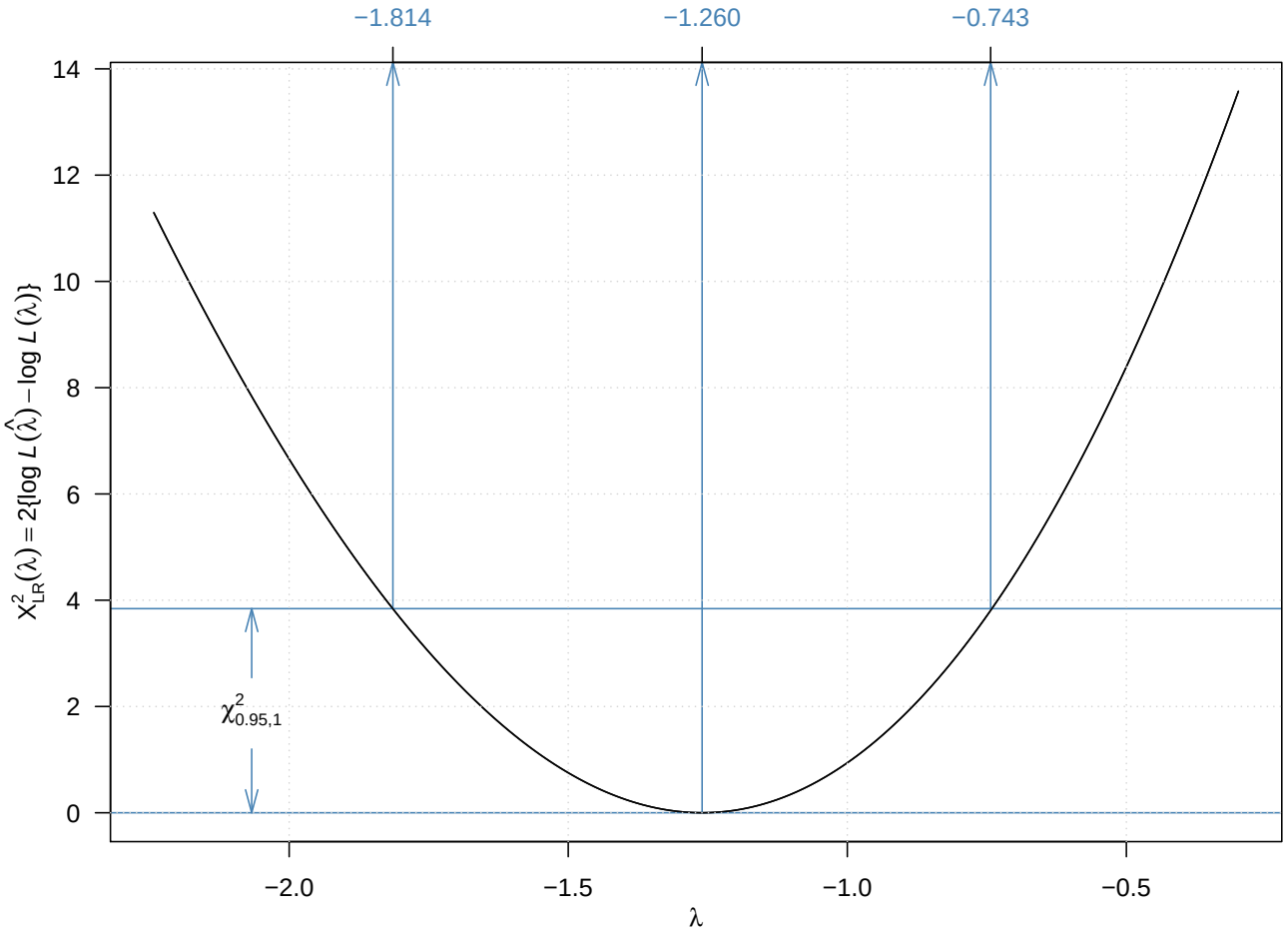
```
sm <- geom_smooth(method = "loess", se = FALSE,
                  colour = "darkgreen", formula = y ~ x)
(plt73 + sm) + (plt93 + sm)
```



Clearly MPG is not a good scale to use. (Why?)

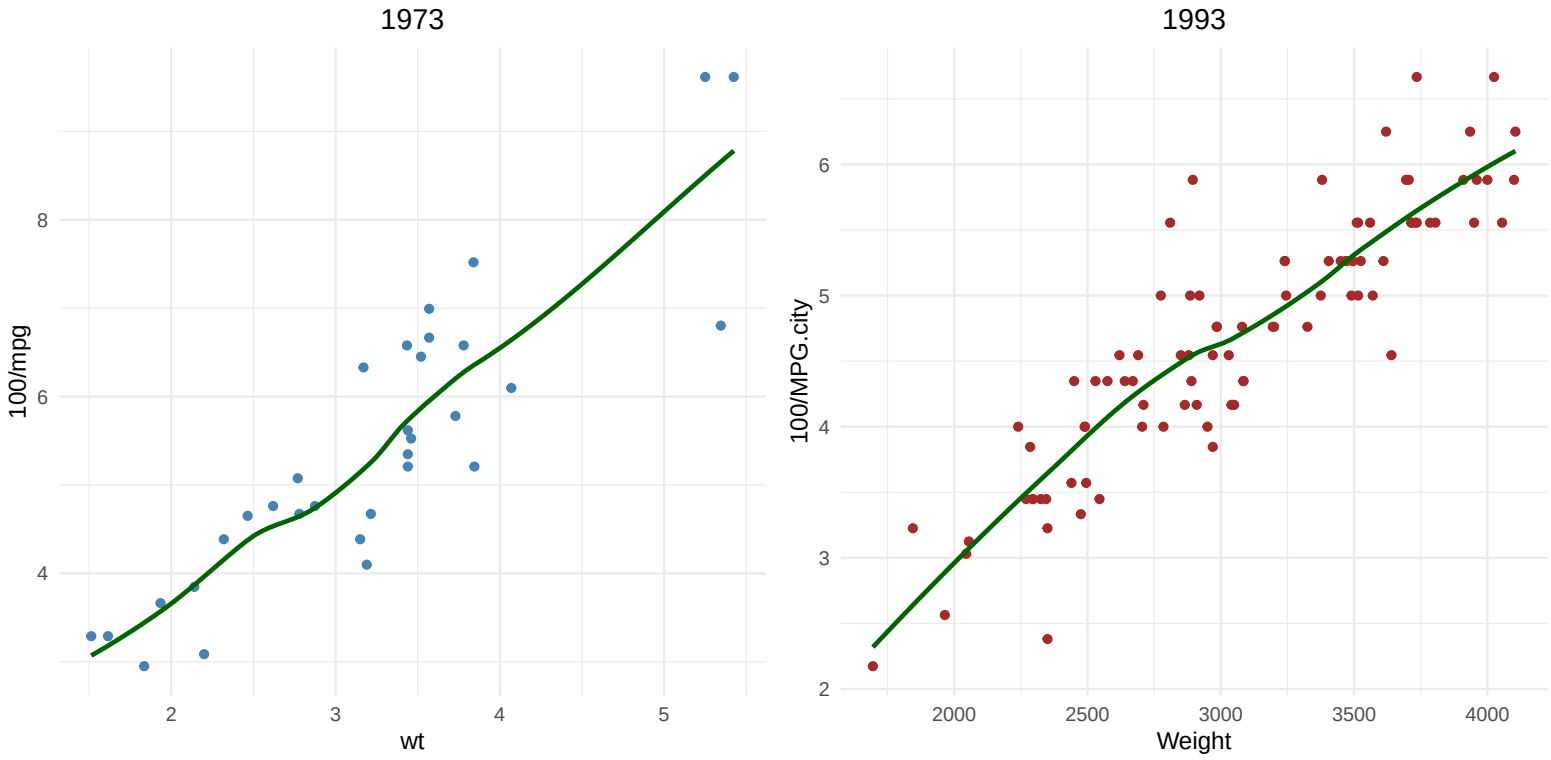
A Box-Cox transform suggest an inverse scale:

```
modl93 <- lm(MPG.city ~ Weight, Cars93)  ## assume we want a straight line
box_cox(modl93)                          ## traditional graphics, in WWRUtilities
```



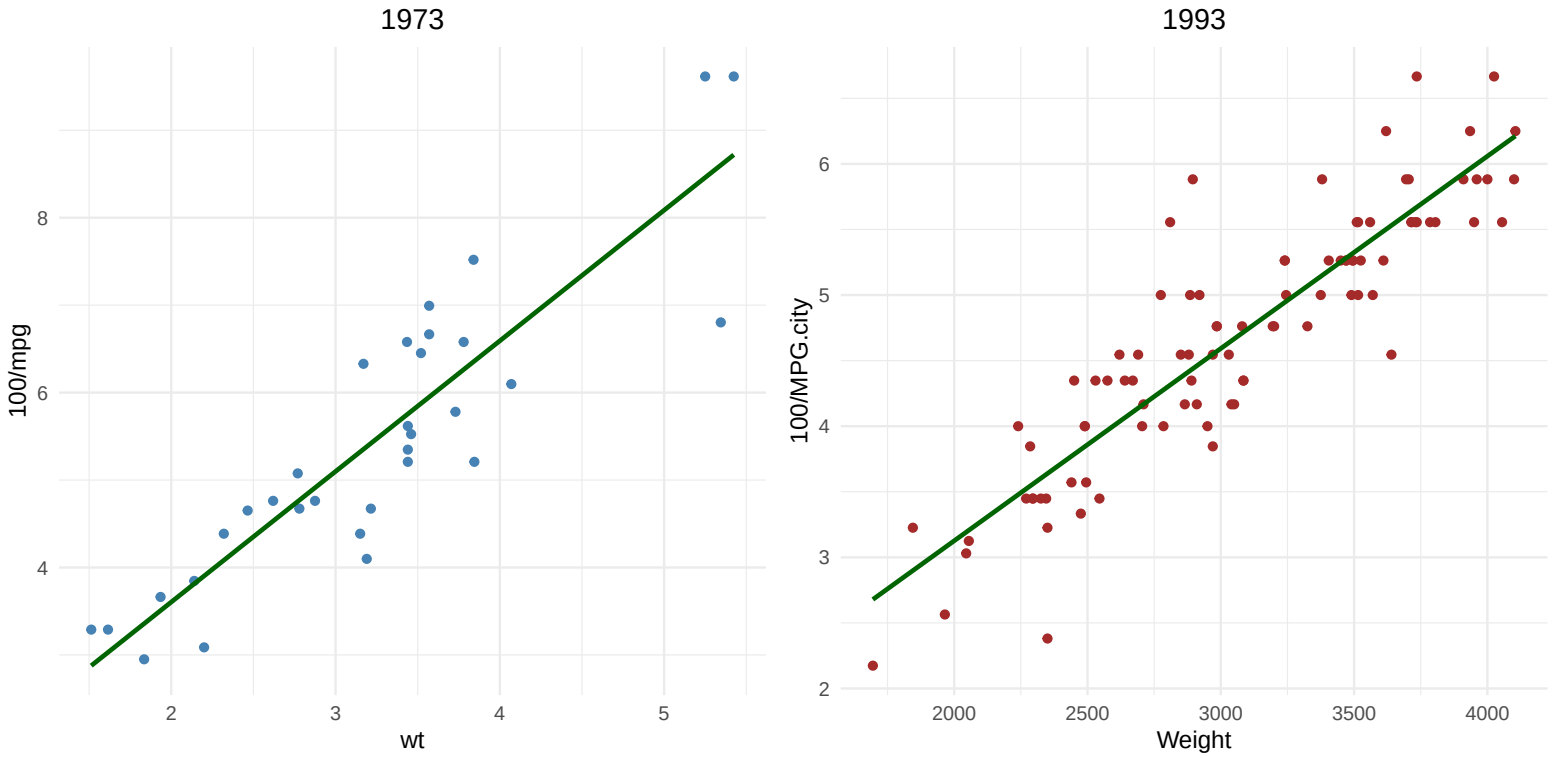
Consider a GPM (Gallons/100 miles) scale instead.

```
plt73 <- ggplot(mtcars) + aes(x = wt, y = 100/mpg) +
  geom_point(colour = "steelblue") + labs(title="1973")
plt93 <- ggplot(Cars93) + aes(x = Weight, y = 100/MPG.city) +
  geom_point(colour = "brown") + labs(title="1993")
(plt73 + sm) + (plt93 + sm)
```



How does a straight line look?

```
sml <- geom_smooth(method = "lm", se = FALSE,
                   colour = "darkgreen", formula = y ~ x)
(plt73 + sml) + (plt93 + sml)
```



Since the scales are different, it is a bit hard to see if the lines look parallel.

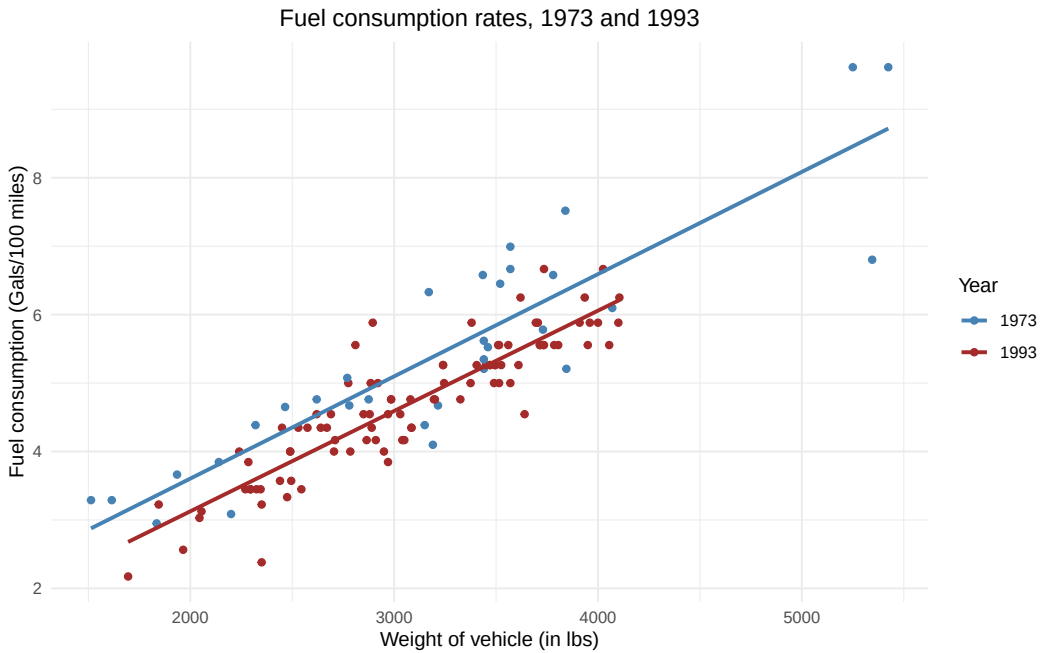
Putting the two data sets together:

```
options(stringsAsFactors = FALSE)  ## no auto conversion
cu.in2litres <- 0.0163871 ## conversion factor
data73 <- with(mtcars, {
  data.frame(Make = row.names(mtcars), Year = "1973", GPM = 100/mpg,
             Displacement = disp * cu.in2litres, Weight = wt * 1000)
})
data93 <- with(Cars93, {
  data.frame(Make = as.character(Make), Year = "1993", GPM = 100/MPG.city,
             Displacement = EngineSize, Weight = Weight)
})
(Cars <- bind_rows(data73, data93) %>% arrange(GPM))
```

```
                        Make Year      GPM Displacement Weight
1                  Geo Metro 1993 2.173913     1.000000   1695
2                 Honda Civic 1993 2.380952     1.500000   2350
3                Suzuki Swift 1993 2.564103     1.300000   1965
....
122                Duster 360 1973 6.993007     5.899356   3570
123                Camaro Z28 1973 7.518797     5.735485   3840
124       Cadillac Fleetwood 1973 9.615385     7.734711   5250
125      Lincoln Continental 1973 9.615385     7.538066   5424
```

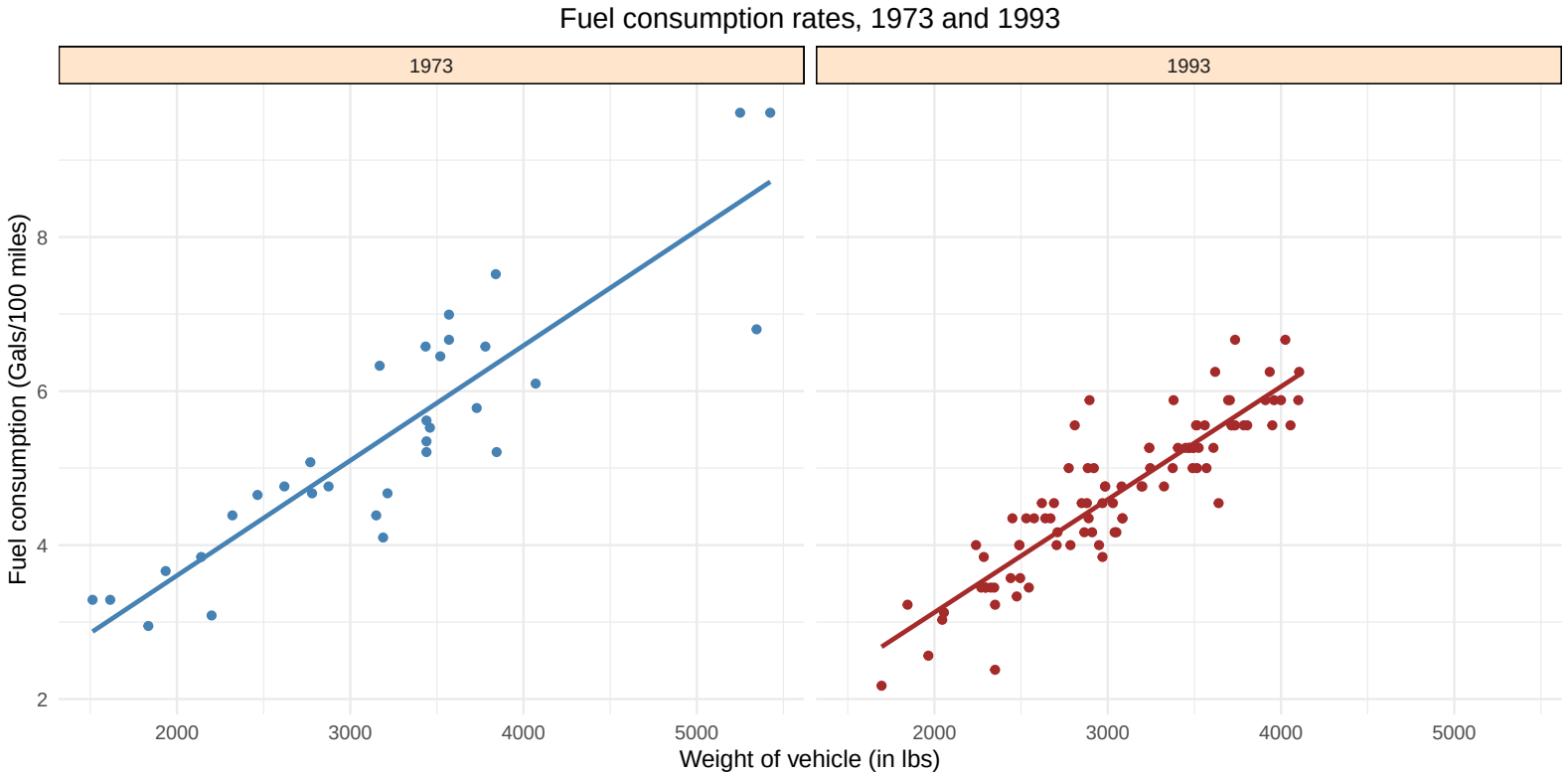
Now put the two together in a single display with the same coordinate system:

```
plt <- ggplot(Cars) + aes(x = Weight, y = GPM, colour = Year) +
  geom_point() + geom_smooth(method = "lm", se = FALSE, formula = y ~ x) +
  labs(title = "Fuel consumption rates, 1973 and 1993",
       x = "Weight of vehicle (in lbs)", y = "Fuel consumption (Gals/100 miles)") +
  theme(plot.title = element_text(hjust = 0.5)) +  ## centre the title
  scale_colour_manual(values = c(`1973` = "steelblue", `1993` = "brown"))
plt
```



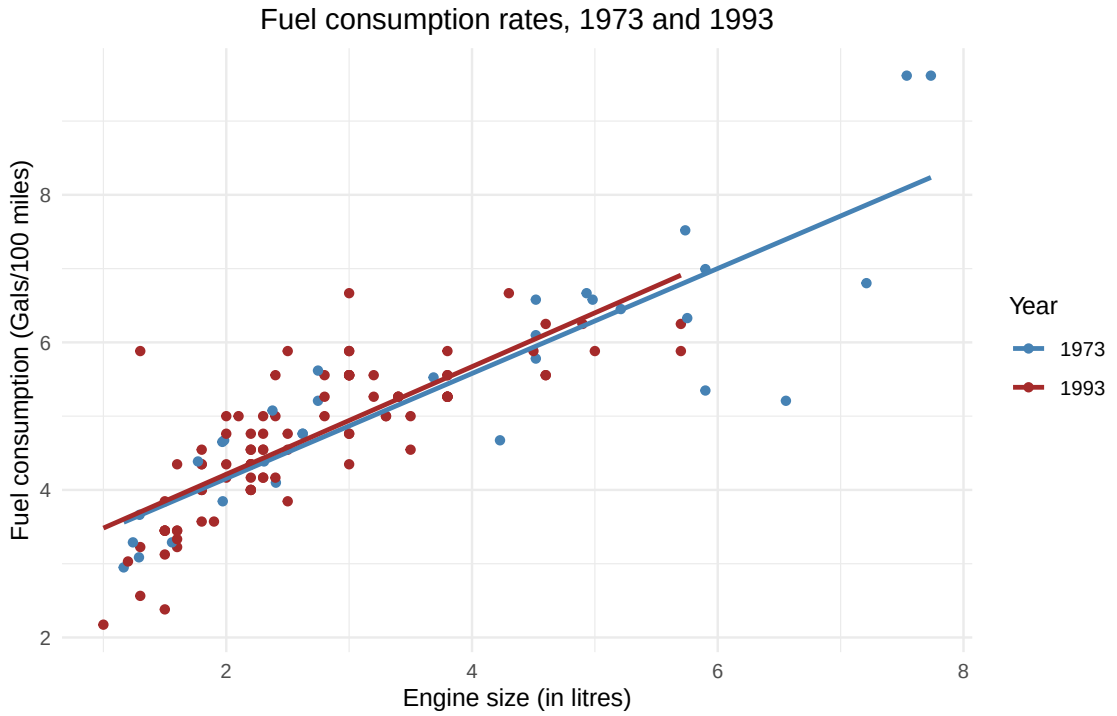
Parallel lines look very convincing. The gap represents the increase in fuel efficiency. Now put them side-by-side

```
plt + facet_wrap(~ Year, nrow = 1) +
  theme(legend.position = "none", plot.title = element_text(hjust = 0.5))
```



Using engine capacity ('displacement') rather than weight offers a different story:

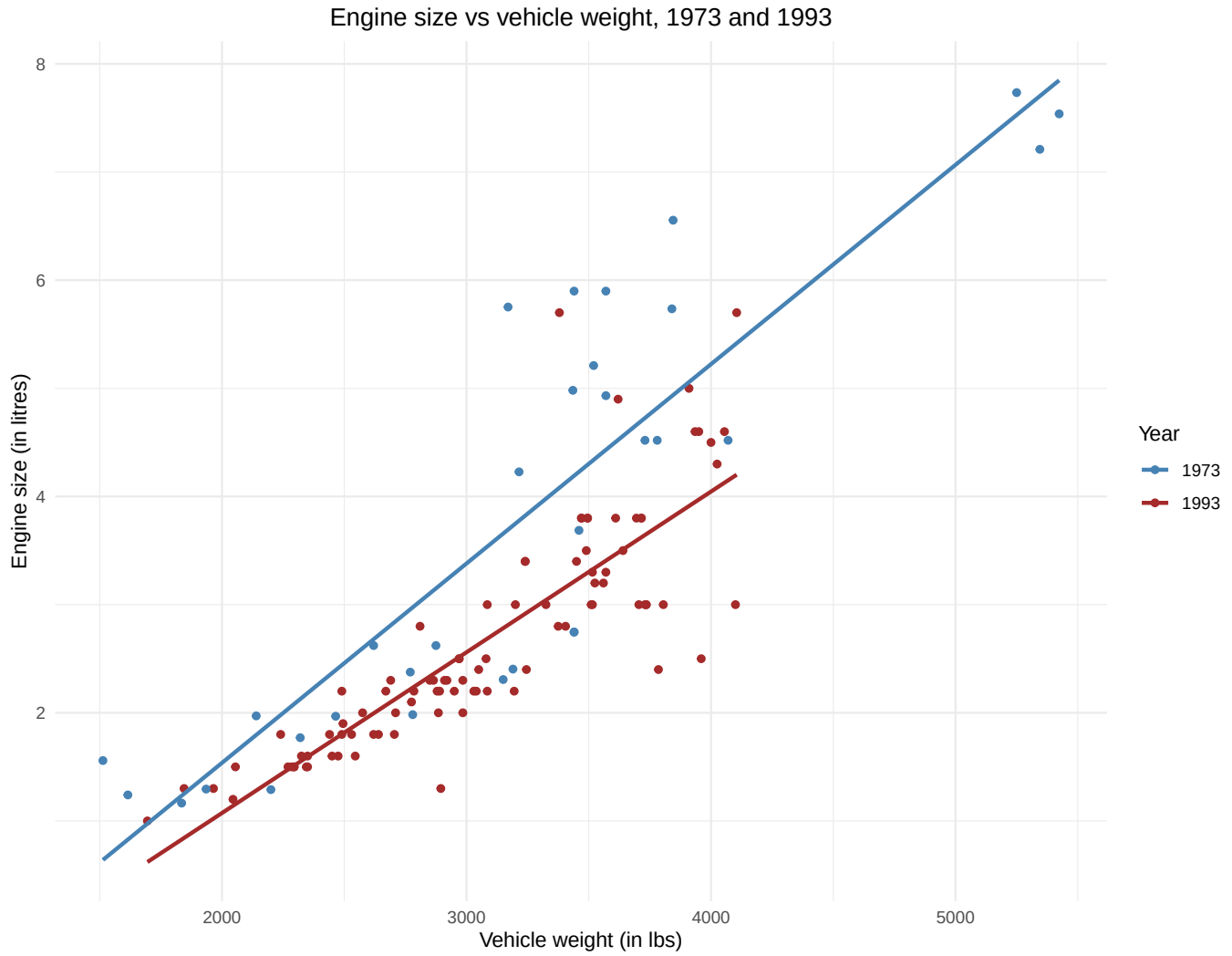
```
plt2 <- ggplot(Cars) + aes(x = Displacement, y = GPM, colour = Year) +
  geom_point() + geom_smooth(method = "lm", se = FALSE, formula = y ~ x) +
  labs(title = "Fuel consumption rates, 1973 and 1993",
       x = "Engine size (in litres)", y = "Fuel consumption (Gals/100 miles)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_colour_manual(values = c(`1973` = "steelblue", `1993` = "brown"))
plt2
```



Fuel consumption grows at almost identical rates with respect to engine size for both years. What is going on?

One clue might be to check the relationship between engine size and weight.

```
plt3 <- ggplot(Cars) + aes(x = Weight, y = Displacement, colour = Year) +
  geom_point() + geom_smooth(method = "lm", se = FALSE, formula = y ~ x) +
  labs(title = "Engine size vs vehicle weight, 1973 and 1993",
       x = "Vehicle weight (in lbs)", y = "Engine size (in litres)") +
  scale_colour_manual(values = c(`1973` = "steelblue", `1993` = "brown")) +
  theme_minimal() +   ## take care of background
  theme(plot.title = element_text(hjust = 0.5)) ## must come after theme_minimal()
plt3
```



Could it be that the slight gain in fuel efficiency over the period was achieved mainly by making the engines smaller and hence less powerful in vehicles of comparable weight?

Finally, what are the makes of the bigger cars?

```
library(ggrepel)  ## add-on package to ggplot2
bigCars <- Cars %>% filter(Displacement > 5.5)
plt3 + geom_text_repel(data = bigCars, aes(label = Make),
                       size = 3, colour = "black") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```

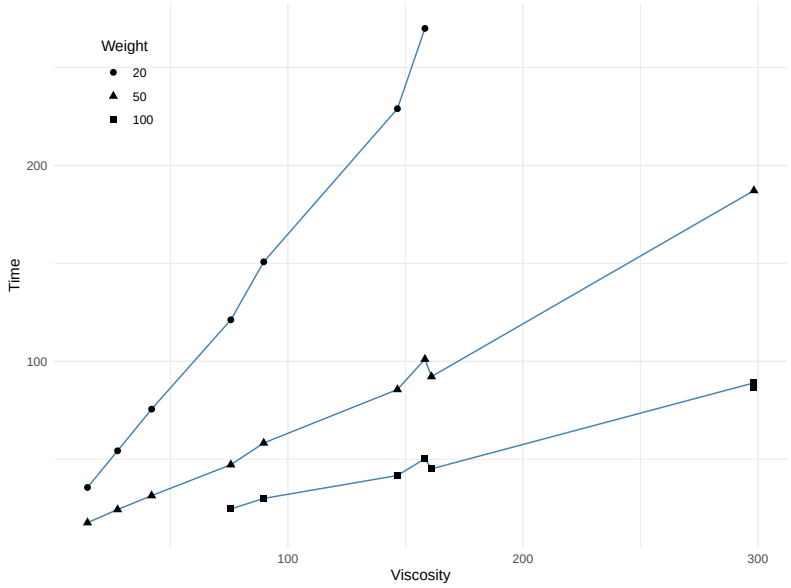
3 A contour plot

A non-linear regression model with two parameters: The Stormer Data

$$T = \frac{\beta V}{W - \theta} + \varepsilon, \qquad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

The data:

```
ggplot(mutate(Stormer, Weight=factor(Weight))) +
  aes(x=Viscosity, y=Time, group=Weight) + geom_line(colour="steelblue") +
  geom_point(aes(shape=Weight), size=2) + theme(legend.position=c(0.1, 0.85))
```



Fitting the model:

```
fm <- nls(Time ~ beta*Viscosity/(Weight - theta), Stormer,
          start = list(beta = 30, theta = 2))
summary(fm)$coef
```

```
       Estimate Std. Error   t value     Pr(>|t|)
beta  29.401257  0.9155336 32.113792 2.457603e-19
theta  2.218274  0.6655217  3.333136 3.155742e-03
```

```
bt <- coef(fm)
se <- sqrt(diag(vfm <- vcov(fm)))
cov2cor(vfm)
```

```
            beta      theta
beta   1.0000000 -0.9199708
theta -0.9199708  1.0000000
```

Look more closely at the least squares surface as a function of β and θ:

```
ssq <- function(beta, theta) {
  sum((Time - beta*Viscosity/(Weight - theta))^2)
}
environment(ssq) <- as.environment(Stormer) ## get Weight &c
parent.env(environment(ssq)) <- baseenv() ## get sum(), ^, &c
SSQ <- Vectorize(ssq)   ## vectorization on the fly!
```

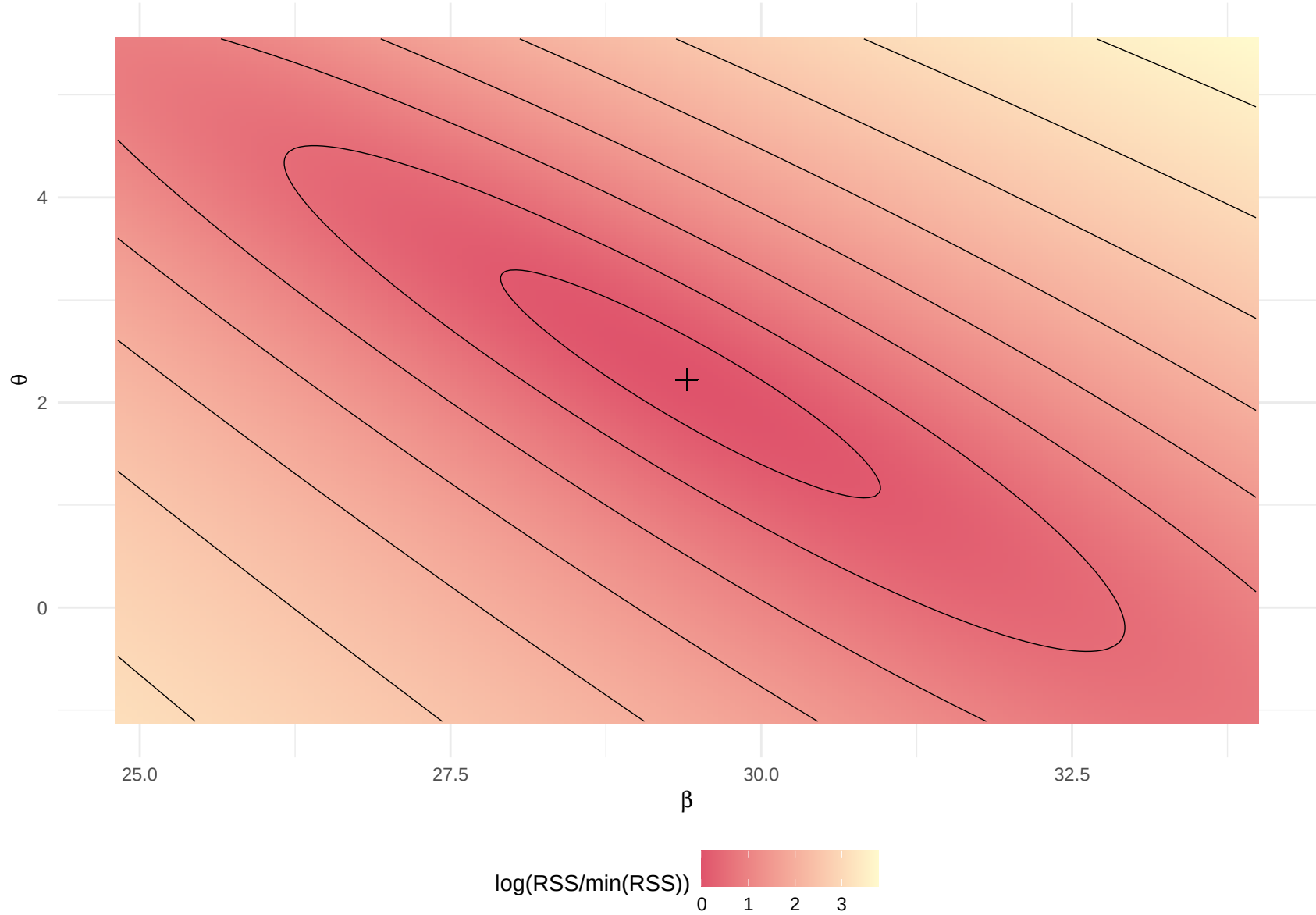
Evaluate this on a wide grid covering the MLE:

```
beta0  <- bt[["beta"]];  bspan <- 5*se[[ "beta"]]
theta0 <- bt[["theta"]]; tspan <- 5*se[["theta"]]

Beta  <- seq(beta0 - bspan, beta0 + bspan, length.out = 201)
Theta <- seq(theta0- tspan, theta0+ tspan, length.out = 201)

RSSq <- expand.grid(beta = Beta, theta = Theta) %>%
  within({
    Rssq <- SSQ(beta, theta)
    logF <- log(Rssq) - log(ssq(beta0, theta0))
  })
```

Now for the display.

```
ggplot(RSSq) + aes(x = beta, y = theta) +
  geom_raster(aes(fill = logF)) +
  geom_point(x = beta0, y = theta0, shape = 3, size = 3)+
  scale_fill_continuous(high = "lemon chiffon", low = "#DF536B",
                        name = "log(RSS/min(RSS))") +
  stat_contour(aes(z = logF),
               colour = "black",
               size = 0.25,
               breaks = c(1/8, (1:7)/2)) +
  labs(x = expression(beta), y = expression(theta),
       title = expression(Time == ' '*frac(beta%*%Viscosity,
                                           (Weight-' '*theta))+epsilon)) +
  theme(plot.title = element_text(hjust = 0.5),
        legend.position = "bottom")
```

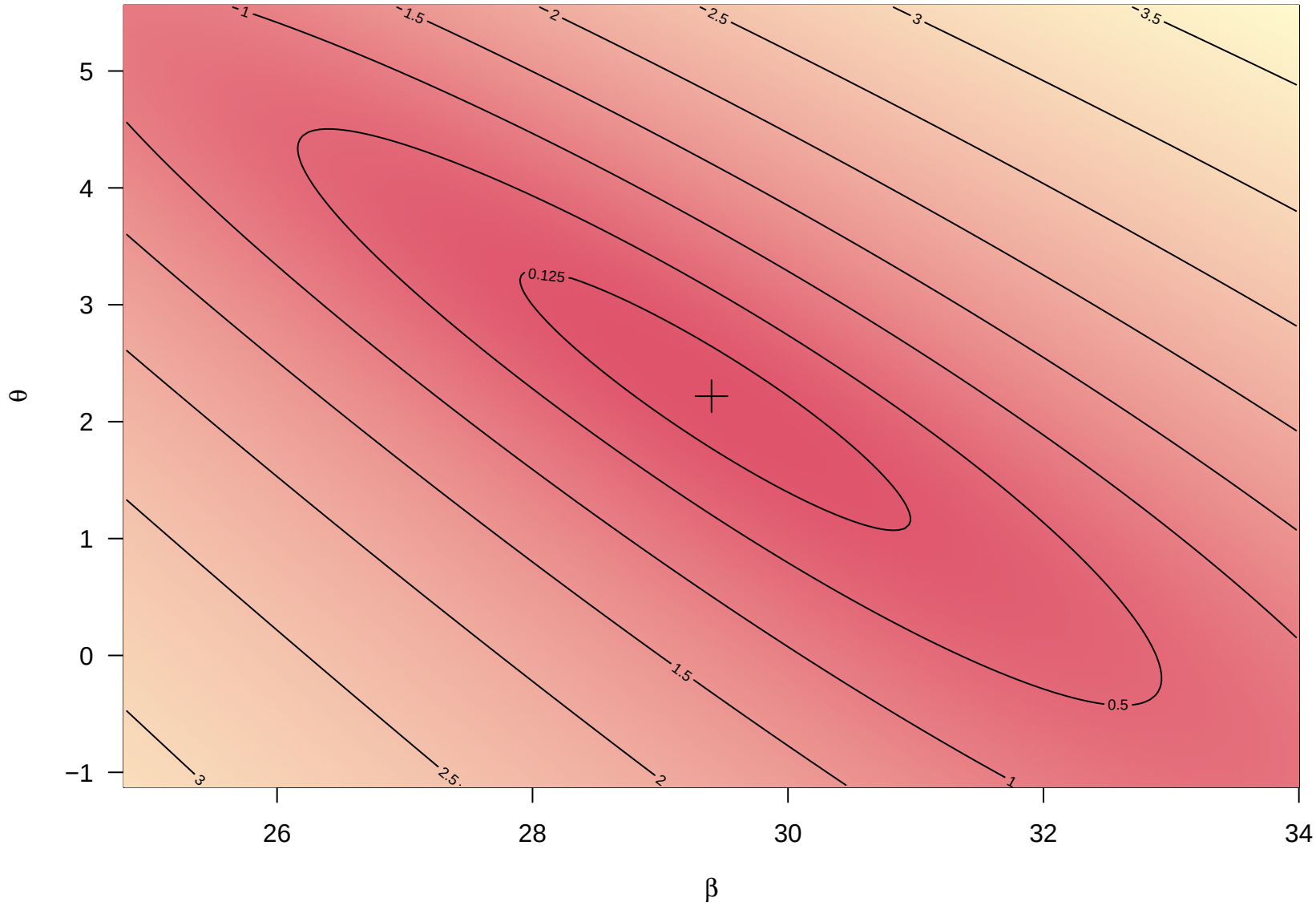
$$\text{Time} = \frac{\beta \times \text{Viscosity}}{(\text{Weight} - \theta)} + \varepsilon$$



A similar display with traditional graphics. The data has to be in a different form first:

```
pal <- colorRampPalette(c("#DF536B", "lemon chiffon"))
logF <- log(outer(Beta, Theta, SSQ))
logF <- logF - log(ssq(beta0, theta0))
image(x = Beta, y = Theta, z = logF, col = pal(250),
      xlab = expression(beta), ylab = expression(theta),
      main = expression(Time == ' '*frac(beta%*%Viscosity,
                                         (Weight-' '*theta))+epsilon))
contour(x = Beta, y = Theta, z = logF, levels = c(1/8, (1:7)/2), add = TRUE)
points(beta0, theta0, pch = 3, cex = 2)
```

$$\text{Time} = \frac{\beta \times \text{Viscosity}}{(\text{Weight} - \theta)} + \varepsilon$$



Session information

Date: 2021-01-29

- R version 4.0.3 (2020-10-10), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 20.04.1 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: dplyr 1.0.3, english 1.2-5, forcats 0.5.1, ggplot2 3.3.3, ggrepel 0.9.1, ggthemes 4.2.4, gridExtra 2.3, knitr 1.31, lattice 0.20-41, patchwork 1.1.1, purrr 0.3.4, readr 1.4.0, scales 1.1.1, stringr 1.4.0, tibble 3.0.5, tidyr 1.1.2, tidyverse 1.3.0, WWRCourse 0.2.3, WWRData 0.1.0, WWRGraphics 0.1.2, WWRUtilities 0.1.2, xtable 1.8-4
- Loaded via a namespace (and not attached): assertthat 0.2.1, backports 1.2.1, broom 0.7.3, cellranger 1.1.0, cli 2.2.0, colorspace 2.0-0, compiler 4.0.3, crayon 1.3.4, DBI 1.1.1, dbplyr 2.0.0, digest 0.6.27, ellipsis 0.3.1, evaluate 0.14, fansi 0.4.2, farver 2.0.3, fractional 0.1.3, fs 1.5.0, generics 0.1.0, glue 1.4.2, grid 4.0.3, gtable 0.3.0, haven 2.3.1, hexbin 1.28.2, highr 0.8, hms 1.0.0, httr 1.4.2, isoband 0.2.3, iterators 1.0.13, jsonlite 1.7.2, labeling 0.4.2, lazyData 1.1.0, lifecycle 0.2.0, lubridate 1.7.9.2, magrittr 2.0.1, MASS 7.3-53, Matrix 1.3-2, mgcv 1.8-33, modelr 0.1.8, munsell 0.5.0, nlme 3.1-151, parallel 4.0.3, PBSmapping 2.73.0, pillar 1.4.7, pkgconfig 2.0.3, R6 2.5.0, randomForest 4.6-14, RColorBrewer 1.1-2, Rcpp 1.0.6, readxl 1.3.1, reprex 1.0.0, rlang 0.4.10, rpart 4.1-15, rstudioapi 0.13, rvest 0.3.6, SOAR 0.99-11, splines 4.0.3, stringi 1.5.3, tidyselect 1.1.0, tools 4.0.3, vctrs 0.3.6, viridisLite 0.3.0, withr 2.4.1, xfun 0.20, xml2 1.3.2