Product Plots

By Hadley Wickham and Heike Hofmann

Jess Rees, Data Science Lab

August 12, 2018

Contents

Framework

Explained in detail in http://vita.had.co.nz/papers/prodplots.pdf

A new framework for visualising tables of counts, proportions and probabilities. Foundations:

- computation of area as a product of height and width
- statistical concept of generating a joint distribution from the product of conditional and marginal distributions

Focusses on area charts, where the area of a graphical element is proportional to the underlying count, proportion, or probability. Key development of the products plot framework is the inverse operation: the factorisation of high-dimensional data to products of low-dimensional plots.

Constraints:

- **area must be proportional to count** (or proprtion, or probability)
- **partitions must be disjoint**; to see the complete area, each rectangle must be non-overlapping. This does not imply that the tiling must be space-filling
- **partitions must be rectangular**, because then many perceptual tasks only require comparing lengths, or positions along a common scale; these tasks are generally easier than comparing areas. Rectangles are also computationally simple, recursive (we can always tile a rectangle with smaller rectangles), and used in many existing visualisations.

Primitives:

- **bars (1d)**: *height is proportional to value*, while width equally divides space. Can be horizontally (hbar) or vertically (vbar) arranged.

- **spines (1d)**: *width is proportional to value*, and height fills the range. Space-filling. Can be horizontally (hspine) or vertically (vspine) arranged, or automatically by splitting the largest dimension (spine).
- **tiles (1d)**: *area is proportional to value*, with no restrictions on height or width other than trying to keep the aspect ratio of each rectangle close to 1.
- **fluct (2d)**: has *height and width proportional to the square root of the value.* Each rectangle is arranged on a regular grid formed by the levels of the two variables, allowing comparisons both vertically and horizontally.

Note: Bars and spines are indistinguishable when the underlying data is evenly distributed across the categories. Comparison is the easiest with bars (comparing positions on a common scale). But spines and tiles work better recursively, since they occupy the complete space.

Plots that fit into this framework:

- bar chart (1 hbar)
- column chart (1 vbar)
- spine plot (1 spine)
- fluctuation (1 fluct)
- stacked bar chart (1 hbar and 1 vspine)
- nested bar chart (2 hbars)
- equal bin size plot (1 fluct and 1 vspine)
- mosaic plot (alternating hspines and vspines)
- double decker plot (n-1 hspines and 1 vspine)
- treemap (n spines)
- squarified treemap (n tiles)
- generalised treemap (any plot ending with a tile)

Plot display

For labelling, use a combination of colour and axis labels. **Axis labelling is not very well supported in the package**. Tool tips may be very helpful.

Extensions

Continuous data

Continuous data can be binned to make them discrete, either into bins of equal width, or bins with an equal number of points. Leads to histograms (analogue to bar chart) and spinograms (analogue to spine chart)

Non-rectangular partitions

Radial plots can be seen as **polar transformations** of product plots. For example, a pie chart is an hspine drawn in polar coordinates with the x-coordinate mapped to angle and the y-coordinate to radius. Generally, the y axis (mapped to radius) must be square-root transformed to ensure that that counts stay proportional to areas.

- concentric pie chart (1 hspine)
- doughnut plot (1 hspine and 1 vspine)
- racetrack plot (1 vbar)
- infoslices (n vbars, using half of the polar plane)

Note that research suggests that **visualisations in polar coordinates are harder to read accurately than visualisations in Cartesian coordinates**.

Using the package

Available from http://github.com/hadley/productplots. Two main functions:

- **prodcalc**: computes the coordinates of each rectangle
- **prodplot**: displays the rectangles with ggplot2

Each graphical primitive is represented by a function: **hspine(), vspine(), spine(), hbar(), vbar(), tile(), and fluct()**.

```
knitr::opts_chunk$set(echo = TRUE)
#devtools::install_github("hadley/productplots")
library(productplots)
library(plyr)
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------------------ ti
```

```
## v ggplot2 3.0.0     v purrr   0.2.4
## v tibble  1.4.2     v dplyr   0.7.4
## v tidyr   0.8.0     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------------------------------------ tidyvers
## x dplyr::arrange()   masks plyr::arrange()
## x purrr::compact()   masks plyr::compact()
## x dplyr::count()     masks plyr::count()
## x dplyr::failwith()  masks plyr::failwith()
## x dplyr::filter()    masks stats::filter()
## x dplyr::id()        masks plyr::id()
## x dplyr::lag()       masks stats::lag()
## x dplyr::mutate()    masks plyr::mutate()
## x dplyr::rename()    masks plyr::rename()
## x dplyr::summarise() masks plyr::summarise()
## x dplyr::summarize() masks plyr::summarize()
```

```
library(ggplot2)
```

Examples from the paper

Using the happiness dataset:

```
getwd()
```

```
## [1] "C:/Users/root/Desktop"
```

```
d <- load("happy.rda") # downloaded from github; reference directory as needed
happy <- happy %>% filter(!is.na(happy))
head(happy)
```

```
##   id         happy year age    sex       marital         degree
## 1  1 not too happy 1972  23 female never married       bachelor
## 2  2 not too happy 1972  70   male       married lt high school
```

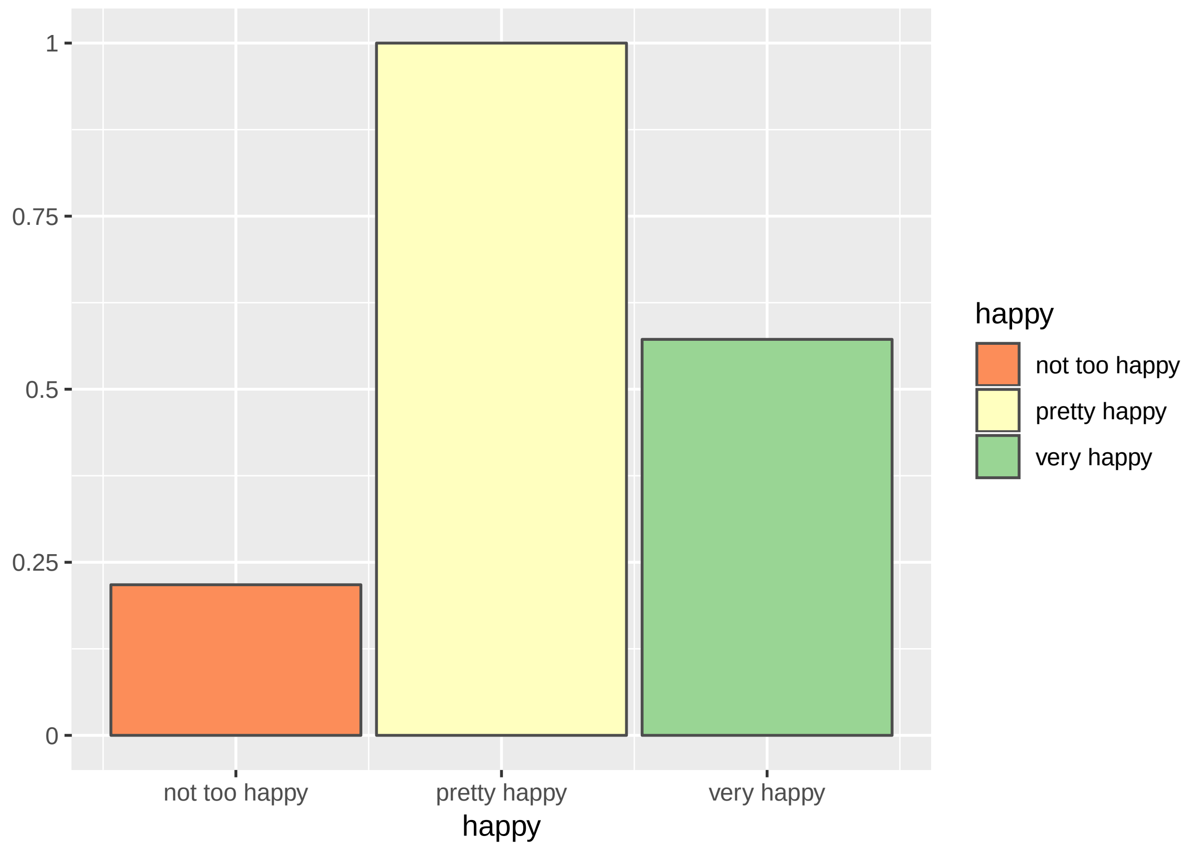
```
## 3  3  pretty happy 1972  48 female       married    high school
## 4  4 not too happy 1972  27 female       married       bachelor
## 5  5  pretty happy 1972  61 female       married    high school
## 6  6  pretty happy 1972  26   male never married    high school
##         finrela    health wtssall
## 1       average      good  0.4446
## 2 above average      fair  0.8893
## 3       average excellent  0.8893
## 4       average      good  0.8893
## 5 above average      good  0.8893
## 6 above average      good  0.4446
```

Basic examples

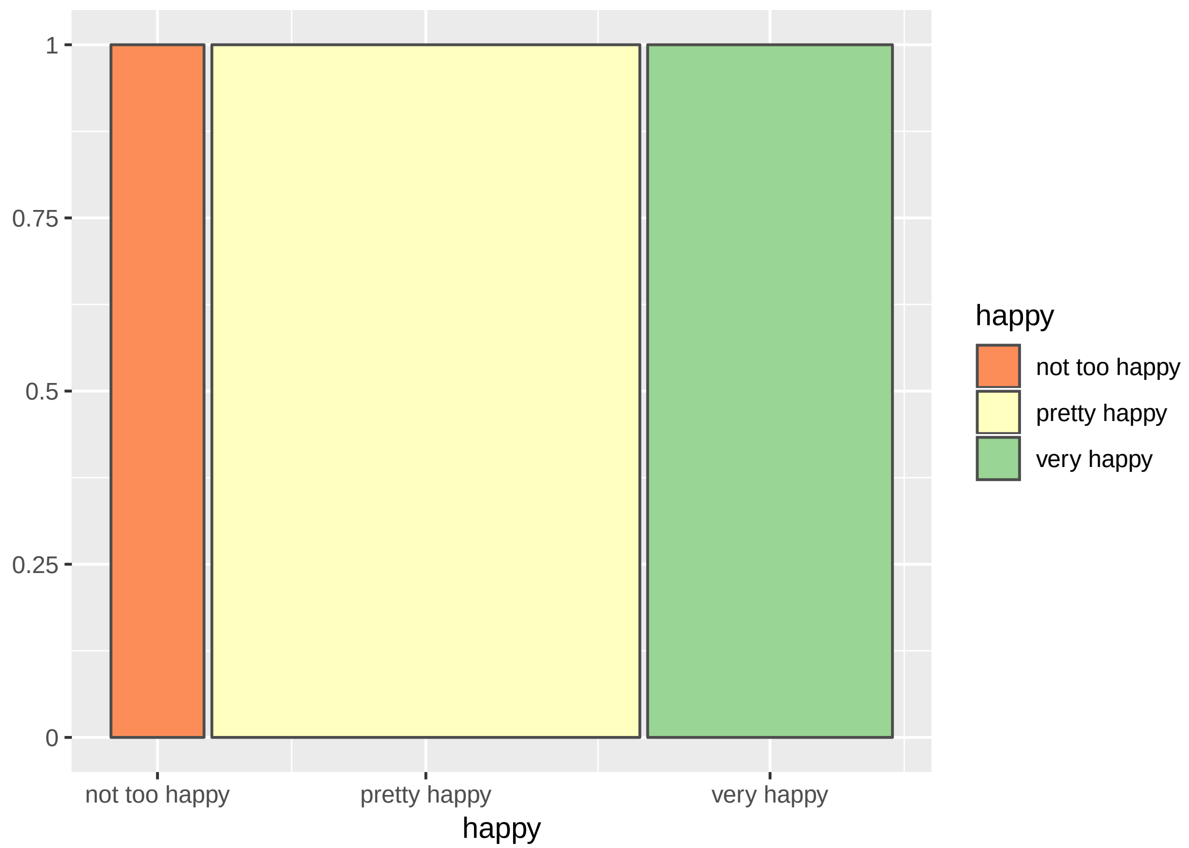
```r
prodcalc(happy, ~ happy, "hbar")
```

```
##           happy   .wt    l    r b         t level
## 1 not too happy  5629 0.00 0.32 0 0.2175543     1
## 2  pretty happy 25874 0.34 0.66 0 1.0000000     1
## 3    very happy 14800 0.68 1.00 0 0.5720028     1
```

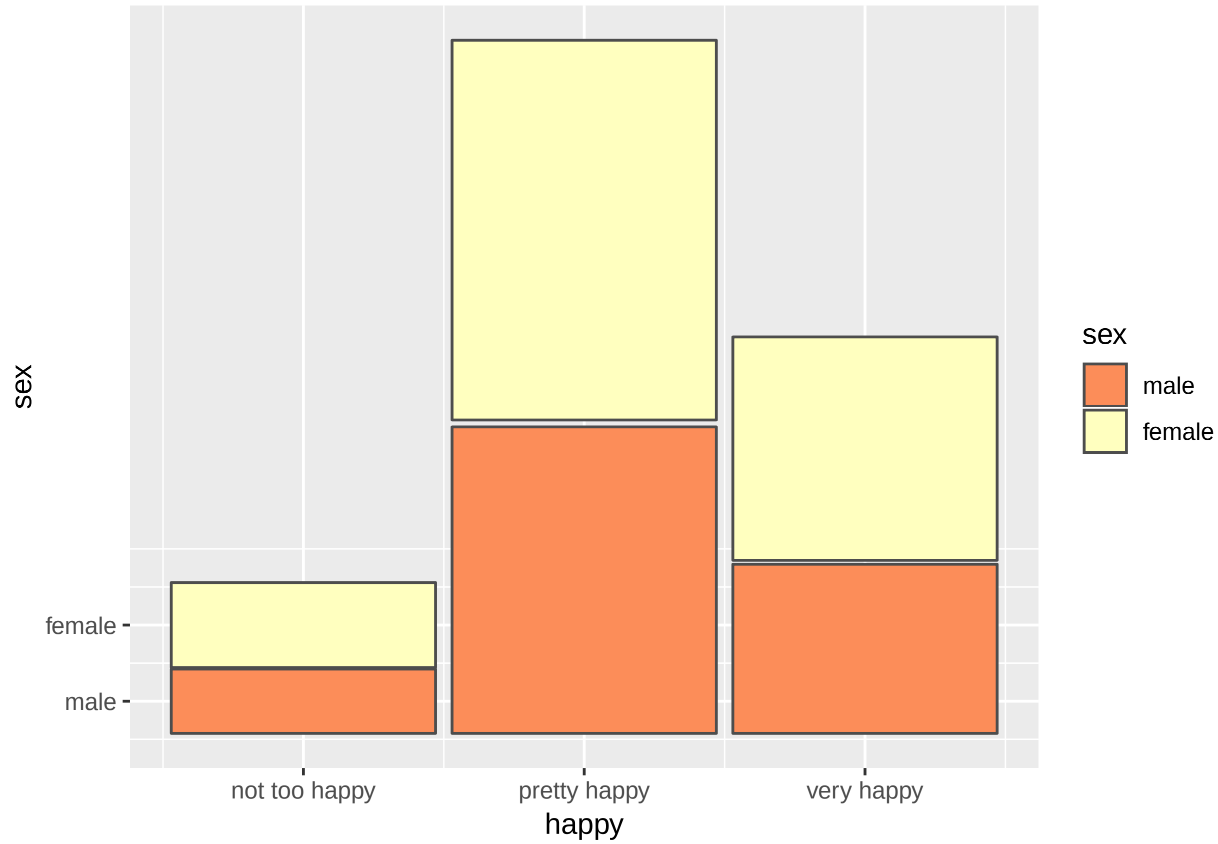
```r
prodplot(happy, ~ happy, "hbar") + aes(fill=happy)+
  scale_fill_brewer(palette="Spectral")
```



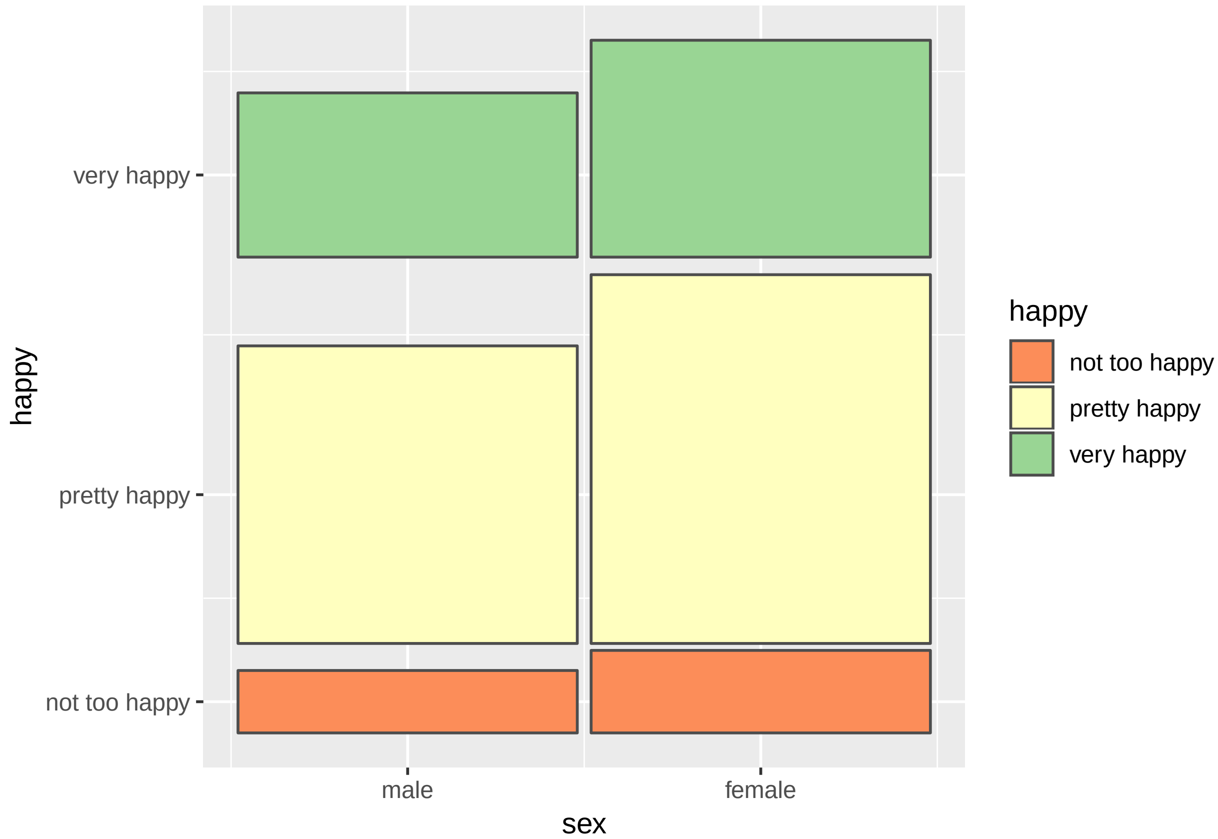

```r
prodplot(happy, ~ happy, "hspine") + aes(fill=happy)+
  scale_fill_brewer(palette="Spectral")
```



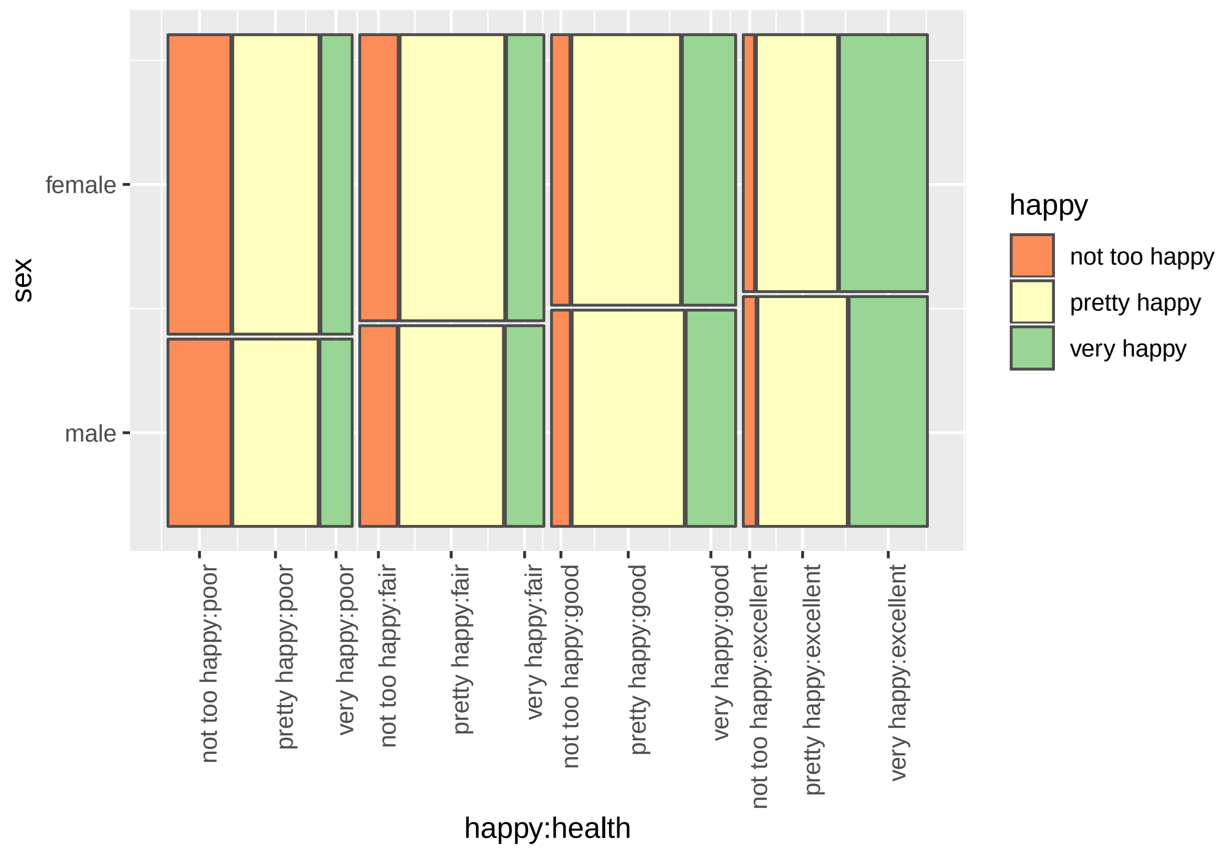

```
prodplot(happy, ~ sex + happy, c("vspine", "hbar")) + aes(fill=sex)+
  scale_fill_brewer(palette="Spectral")
```




```r
prodplot(happy, ~ sex + happy, stacked()) + aes(fill=happy)+
  scale_fill_brewer(palette="Spectral")
```




```
prodplot(happy %>% filter(!is.na(health)), ~ happy + sex | health, mosaic("h")) +
  aes(fill=happy) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_fill_brewer(palette="Spectral")
```



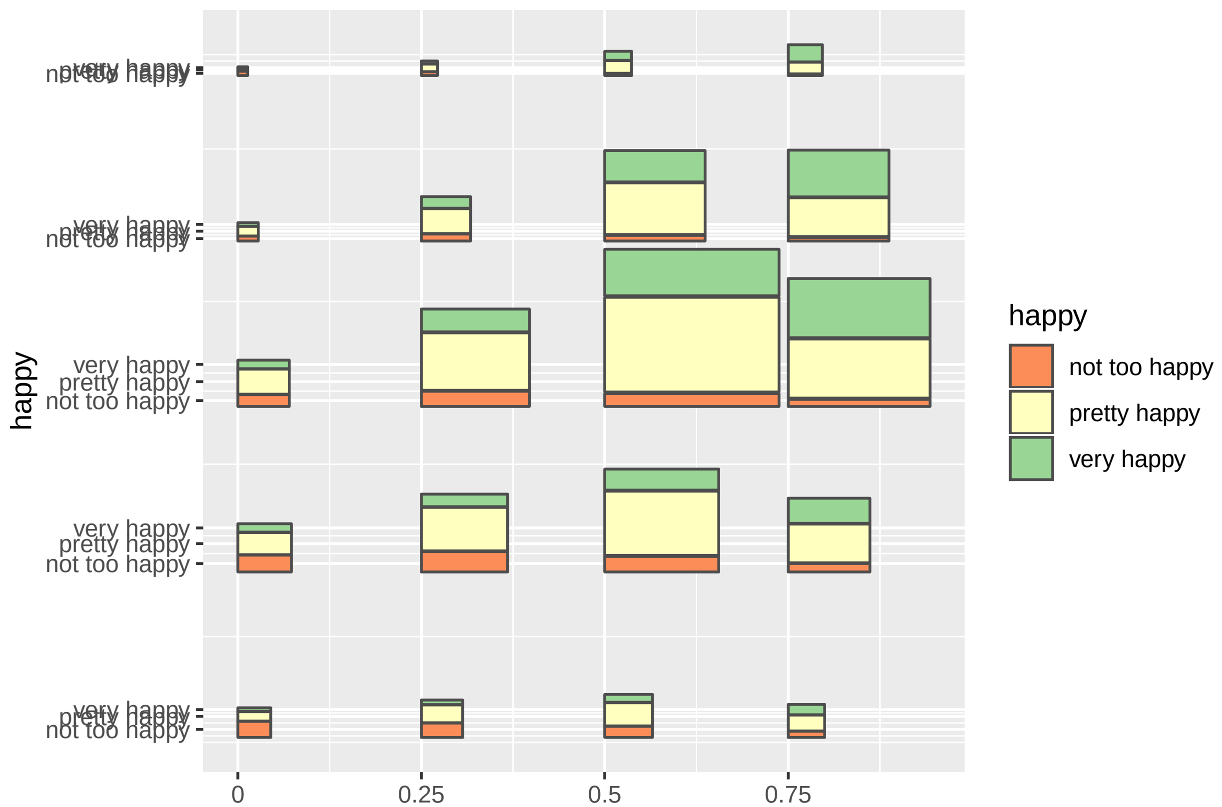
More complex examples

We'll use **fluct and spines to see the relationship between happiness, health, and financial status**.

f(happy,health, finrela), partitioned with a vspine and fluct

f(happy,health, finrela) = f(happy|health, finrela) x f(health, finrela)

```
newhappy <- happy %>%
  mutate(finrela = as.factor(finrela)) %>%
  filter(!is.na(happy)) %>%
  filter(!is.na(finrela)) %>%
  filter(!is.na(health))

prodplot(newhappy, ~ happy + finrela + health, c("vspine", "fluct")) +
  aes(fill=happy) +
  scale_fill_brewer(palette="Spectral")
```



This plot displays raw proportions, showing that most people are in good health and average financial standing. However, it is difficult to see how happiness varies within these conditions because we must compare areas, not positions.

Health is on the x-axis, financial status on the y-axis. **But the labels are not right...**

```
# instead of calling prodplot (https://github.com/hadley/productplots/blob/master/R/plot.r), we go thro
# this is all just what the function would do
levels = -1L
cascade=0
scale_max = T
na.rm = F
data = newhappy

formula = as.formula("~ happy + finrela + health") # same formula as above
vars <- parse_product_formula(formula)
p <- length(c(vars$cond, vars$marg))

divider <- c("vspine", "fluct") # same as above
if (is.function(divider)) divider <- divider(p)
div_names <- divider
if (is.character(divider)) divider <- llply(divider, match.fun)

# https://github.com/hadley/productplots/blob/master/R/calculate.r
res <- prodcalc(data, formula, divider, cascade, scale_max, na.rm = na.rm) # available from product plo

if (!(length(levels) == 1 && is.na(levels))) {
  levels[levels < 0] <-  max(res$level) + 1 + levels[levels < 0]
```

```r
  res <- res[res$level %in% levels, ]
}

# here is where we change it; prodplot calls "draw", but we need a new draw function
df = list(data=res, formula=formula, divider=div_names)
alpha = 1
colour = "grey30"
subset = NULL
data <- df$data

# look at data
data
```

```
##                finrela    health  .wt    l           b          r
## 21 far below average      poor  165 0.00 0.000000000 0.04476846
## 22 far below average      poor   97 0.00 0.019726870 0.04476846
## 23 far below average      poor   37 0.00 0.031471478 0.04476846
## 24     below average      poor  282 0.00 0.200000000 0.07286167
## 25     below average      poor  369 0.00 0.220922340 0.07286167
## 26     below average      poor  141 0.00 0.248119616 0.07286167
## 27           average      poor  190 0.00 0.400000000 0.06999955
## 28           average      poor  404 0.00 0.414824200 0.06999955
## 29           average      poor  137 0.00 0.445714398 0.06999955
## 30     above average      poor   31 0.00 0.600000000 0.02776422
## 31     above average      poor   61 0.00 0.606089780 0.02776422
## 32     above average      poor   23 0.00 0.617857946 0.02776422
## 33 far above average      poor   13 0.00 0.800000000 0.01320150
## 34 far above average      poor    8 0.00 0.805280599 0.01320150
## 35 far above average      poor    5 0.00 0.808570819 0.01320150
## 36 far below average      fair  186 0.25 0.000000000 0.30642653
## 37 far below average      fair  231 0.25 0.017774239 0.30642653
## 38 far below average      fair   58 0.25 0.039739484 0.30642653
## 39     below average      fair  549 0.25 0.200000000 0.36753726
## 40     below average      fair 1174 0.25 0.225486594 0.36753726
## 41     below average      fair  338 0.25 0.278917514 0.36753726
## 42           average      fair  515 0.25 0.400000000 0.39721071
## 43           average      fair 1946 0.25 0.419562405 0.39721071
## 44           average      fair  772 0.25 0.490209339 0.39721071
## 45     above average      fair  108 0.25 0.600000000 0.31701530
## 46     above average      fair  383 0.25 0.609005256 0.31701530
## 47     above average      fair  179 0.25 0.639575434 0.31701530
## 48 far above average      fair   19 0.25 0.800000000 0.27196862
## 49 far above average      fair   38 0.25 0.804720813 0.27196862
## 50 far above average      fair   15 0.25 0.813986689 0.27196862
## 51 far below average      good  165 0.50 0.000000000 0.56498406
## 52 far below average      good  347 0.50 0.013863266 0.56498406
## 53 far below average      good  118 0.50 0.042444699 0.56498406
## 54     below average      good  553 0.50 0.200000000 0.65538476
## 55     below average      good 2297 0.50 0.219945846 0.65538476
## 56     below average      good  752 0.50 0.298874745 0.65538476
## 57           average      good  704 0.50 0.400000000 0.73750000
## 58           average      good 5174 0.50 0.417477516 0.73750000
## 59           average      good 2537 0.50 0.533863411 0.73750000
## 60     above average      good  175 0.50 0.600000000 0.63670454
```

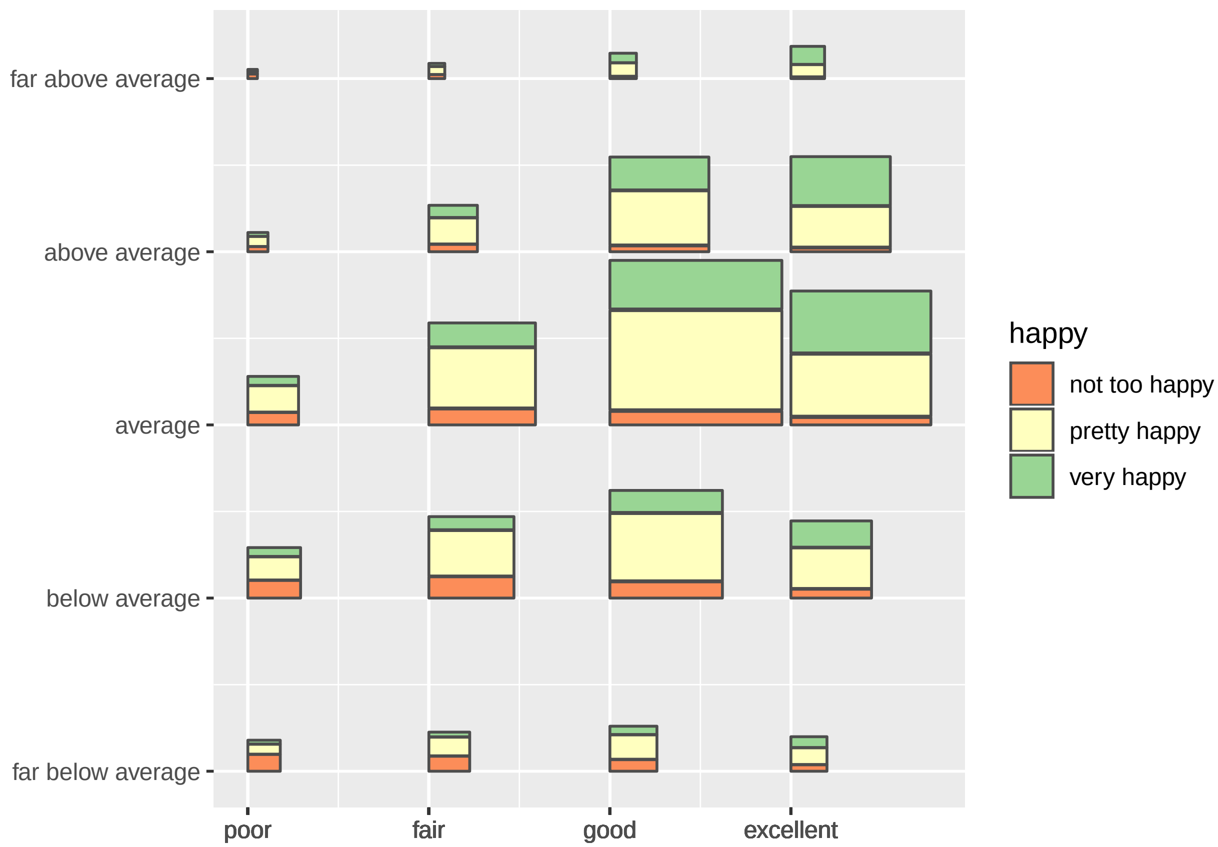
```
## 61     above average      good 1626 0.50 0.607820990 0.63670454
## 62     above average      good  987 0.50 0.671421356 0.63670454
## 63 far above average      good   17 0.50 0.800000000 0.53661437
## 64 far above average      good  107 0.50 0.802732896 0.53661437
## 65 far above average      good   76 0.50 0.818383342 0.53661437
## 66 far below average excellent   70 0.75 0.000000000 0.79980093
## 67 far below average excellent  182 0.75 0.007785096 0.79980093
## 68 far below average excellent  118 0.75 0.027388894 0.79980093
## 69     below average excellent  216 0.75 0.200000000 0.86138835
## 70     below average excellent  992 0.75 0.211081787 0.86138835
## 71     below average excellent  643 0.75 0.258774534 0.86138835
## 72           average excellent  314 0.75 0.400000000 0.94331208
## 73           average excellent 2631 0.75 0.410082603 0.94331208
## 74           average excellent 2630 0.75 0.483152974 0.94331208
## 75     above average excellent  112 0.75 0.600000000 0.88731609
## 76     above average excellent 1225 0.75 0.605384860 0.88731609
## 77     above average excellent 1476 0.75 0.653365142 0.88731609
## 78 far above average excellent   15 0.75 0.800000000 0.79653051
## 79 far above average excellent  124 0.75 0.802066358 0.79653051
## 80 far above average excellent  184 0.75 0.816443277 0.79653051
##              t level         happy
## 21 0.019368722     2 not too happy
## 22 0.031113331     2  pretty happy
## 23 0.035814767     2    very happy
## 24 0.220339447     2 not too happy
## 25 0.247536722     2  pretty happy
## 26 0.258289339     2    very happy
## 27 0.414264204     2 not too happy
## 28 0.445154401     2  pretty happy
## 29 0.455999639     2    very happy
## 30 0.605867666     2 not too happy
## 31 0.617635832     2  pretty happy
## 32 0.622211376     2    very happy
## 33 0.805174987     2 not too happy
## 34 0.808465207     2  pretty happy
## 35 0.810561199     2    very happy
## 36 0.017322827     2 not too happy
## 37 0.039288072     2  pretty happy
## 38 0.045141226     2    very happy
## 39 0.224546296     2 not too happy
## 40 0.277977216     2  pretty happy
## 41 0.294029805     2    very happy
## 42 0.418384719     2 not too happy
## 43 0.489031653     2  pretty happy
## 44 0.517768568     2    very happy
## 45 0.608469133     2 not too happy
## 46 0.639039312     2  pretty happy
## 47 0.653612239     2    very happy
## 48 0.804545064     2 not too happy
## 49 0.813810940     2  pretty happy
## 50 0.817574897     2    very happy
## 51 0.013343393     2 not too happy
## 52 0.041924826     2  pretty happy
## 53 0.051987246     2    very happy
```

```
## 54 0.218702768     2 not too happy
## 55 0.297631667     2  pretty happy
## 56 0.324307804     2    very happy
## 57 0.415577516     2 not too happy
## 58 0.531963411     2  pretty happy
## 59 0.590000000     2    very happy
## 60 0.606727354     2 not too happy
## 61 0.670327719     2  pretty happy
## 62 0.709363632     2    very happy
## 63 0.802439982     2 not too happy
## 64 0.818090427     2  pretty happy
## 65 0.829291495     2    very happy
## 66 0.007386689     2 not too happy
## 67 0.026990487     2  pretty happy
## 68 0.039840741     2    very happy
## 69 0.210190680     2 not too happy
## 70 0.257883427     2  pretty happy
## 71 0.289110678     2    very happy
## 72 0.408536107     2 not too happy
## 73 0.481606477     2  pretty happy
## 74 0.554649663     2    very happy
## 75 0.604286332     2 not too happy
## 76 0.652266613     2  pretty happy
## 77 0.709852870     2    very happy
## 78 0.801694114     2 not too happy
## 79 0.816071033     2  pretty happy
## 80 0.837224408     2    very happy
```

```
finrelabels = data %>%
  group_by(finrela) %>%
  filter(b == min(b)) %>% # want the label at the bottom of the block
  select(finrela, b) %>%
  distinct()

plot <- ggplot(data,
  ggplot2::aes_string(xmin = "l", xmax = "r", ymin = "b", ymax = "t")) +
  scale_y_product(df) + # from prodplots package
  scale_x_continuous(breaks = data$l, labels = data$health) + # put health labels at left of block
  scale_y_continuous(breaks = finrelabels$b, labels = finrelabels$finrela) # put finrela labels at bott
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

```
# now we have to actually add the rectangles
levels <- split(data, data$level) # in this case, we only have 1
for (level in levels) {
  plot <- plot + geom_rect(data = level, colour = colour, alpha = alpha) #
}

plot +
  aes(fill=happy) + # as before
  scale_fill_brewer(palette="Spectral")
```



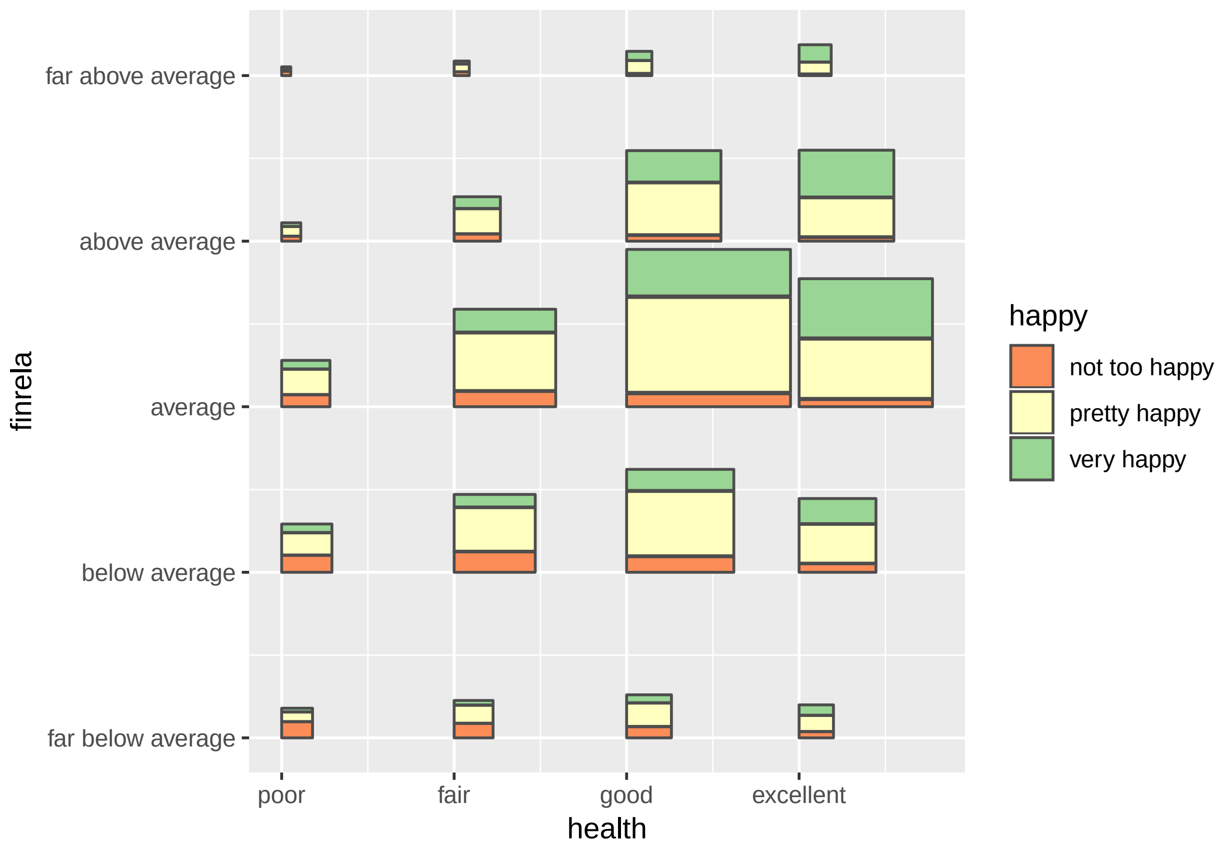
Now let's (kind of) generalise the code...

```
myprodplot <- function(data, str_formula, divider, forx, fory, forfill) {
  levels = -1L
  cascade=0
  scale_max = T
  na.rm = F
  alpha = 1
  colour = "grey30"
  subset = NULL

  formula = as.formula(str_formula)
  vars <- parse_product_formula(formula)
  p <- length(c(vars$cond, vars$marg))

  if (is.function(divider)) divider <- divider(p)
  div_names <- divider
  if (is.character(divider)) divider <- llply(divider, match.fun)

  res <- prodcalc(data, formula, divider, cascade, scale_max, na.rm = na.rm)

  if (!(length(levels) == 1 && is.na(levels))) {
    levels[levels < 0] <-  max(res$level) + 1 + levels[levels < 0]
    res <- res[res$level %in% levels, ]
  }

  df = list(data=res, formula=formula, divider=div_names)
```

```r
  data <- df$data

  colnum = which(colnames(data)==fory)
  colnames(data)[colnum] <- "y"
  ylabels = data %>%
    group_by(y) %>%
    filter(b == min(b)) %>%
    select(y, b) %>%
    distinct()

  colnum = which(colnames(data)==forx)
  colnames(data)[colnum] <- "x"
  xlabels = data %>%
    group_by(x) %>%
    filter(l == min(l)) %>%
    select(x, l) %>%
    distinct()

  plot <- ggplot(data,
    ggplot2::aes_string(xmin = "l", xmax = "r", ymin = "b", ymax = "t")) +
    scale_y_product(df) +
    scale_x_continuous(breaks = xlabels$l, labels = xlabels$x) +
    scale_y_continuous(breaks = ylabels$b, labels = ylabels$y)

  levels <- split(data, data$level)
  for (level in levels) {
    plot <- plot + geom_rect(data = level, colour = colour, alpha = alpha)
  }

  plot <- plot +
    ggplot2::aes_string(fill=forfill) +
    xlab(forx) +
    ylab(fory) +
    scale_fill_brewer(palette="Spectral")

  return(plot)
}

p = myprodplot(newhappy, "~ happy + finrela + health", c("vspine", "fluct"), "health", "finrela", "happy
```

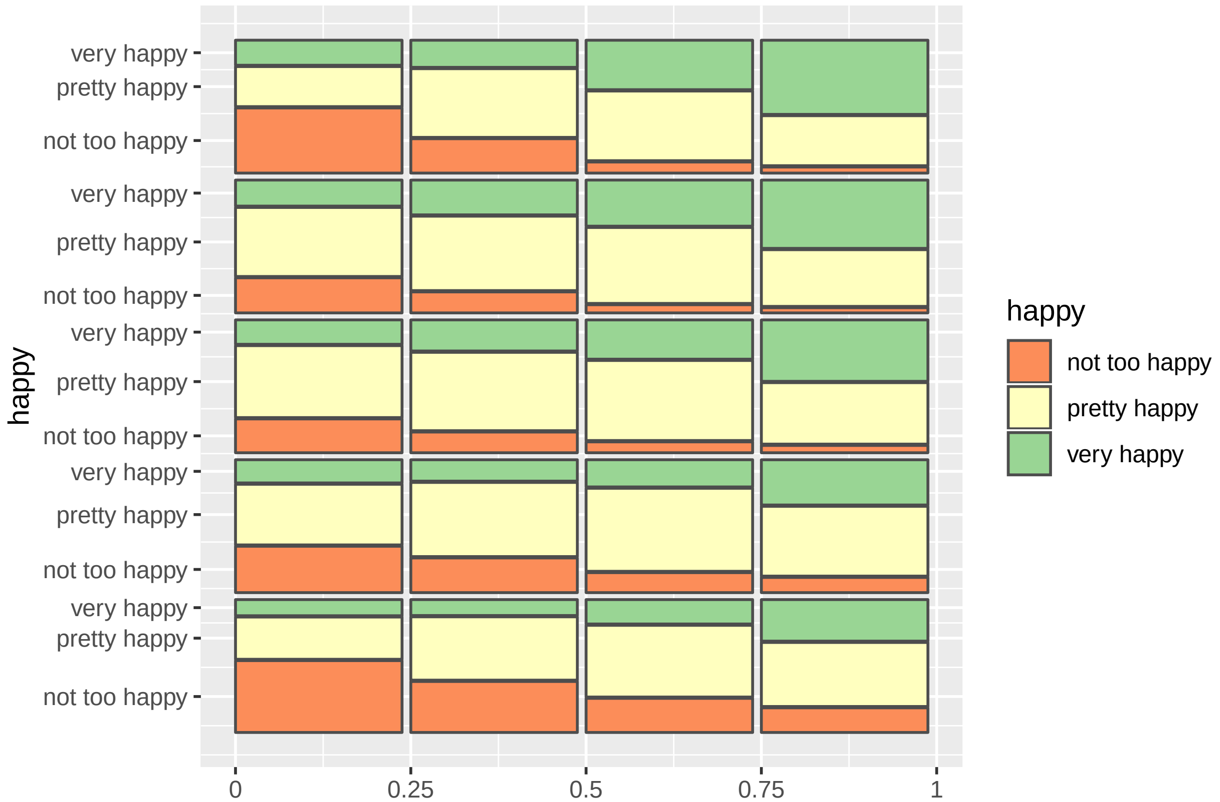
```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

```r
p
```



f(happy | health, finrela), partitioned with a vspine and fluct

```
prodplot(newhappy, ~ happy | finrela + health, c("vspine", "fluct")) +
  aes(fill=happy) +
  scale_fill_brewer(palette="Spectral")
```

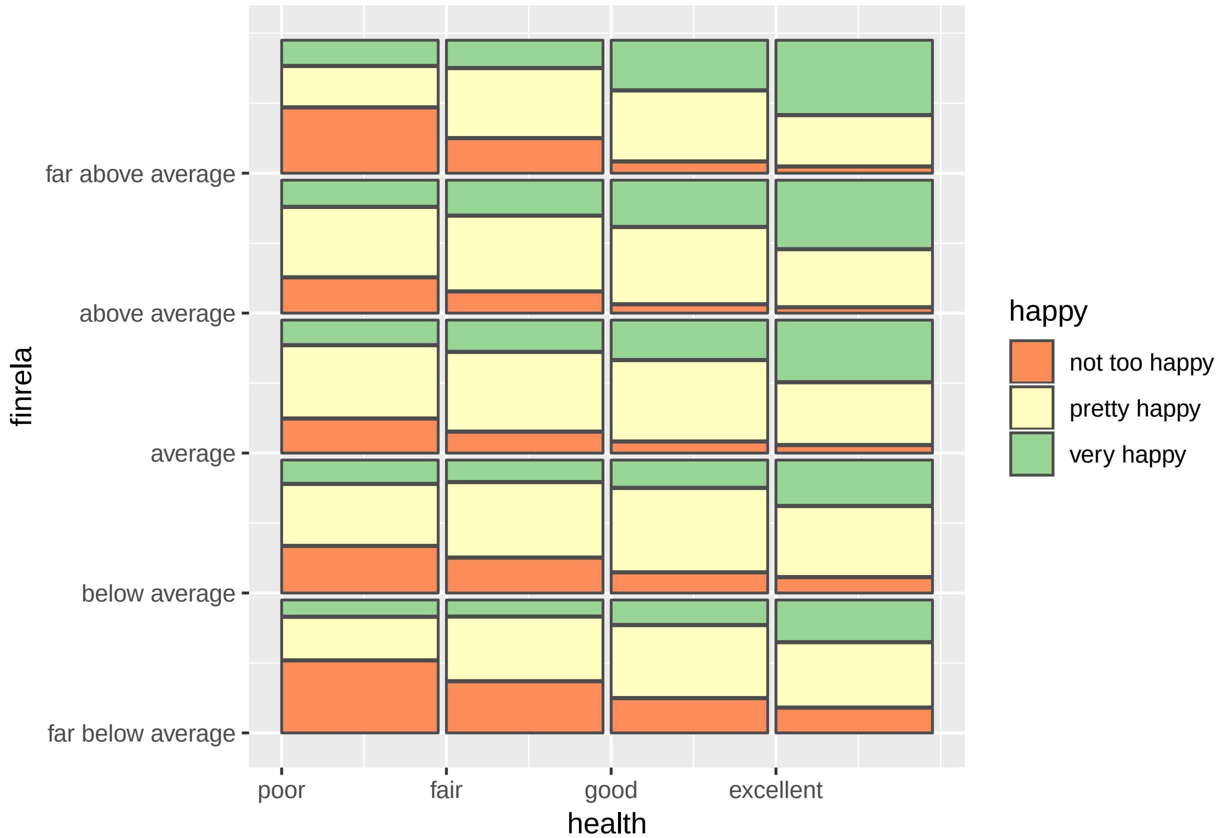


Again, the labels are not what we need...

```
p = myprodplot(newhappy, "~ happy | finrela + health", c("vspine", "fluct"), "health", "finrela", "happy
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

```
p
```

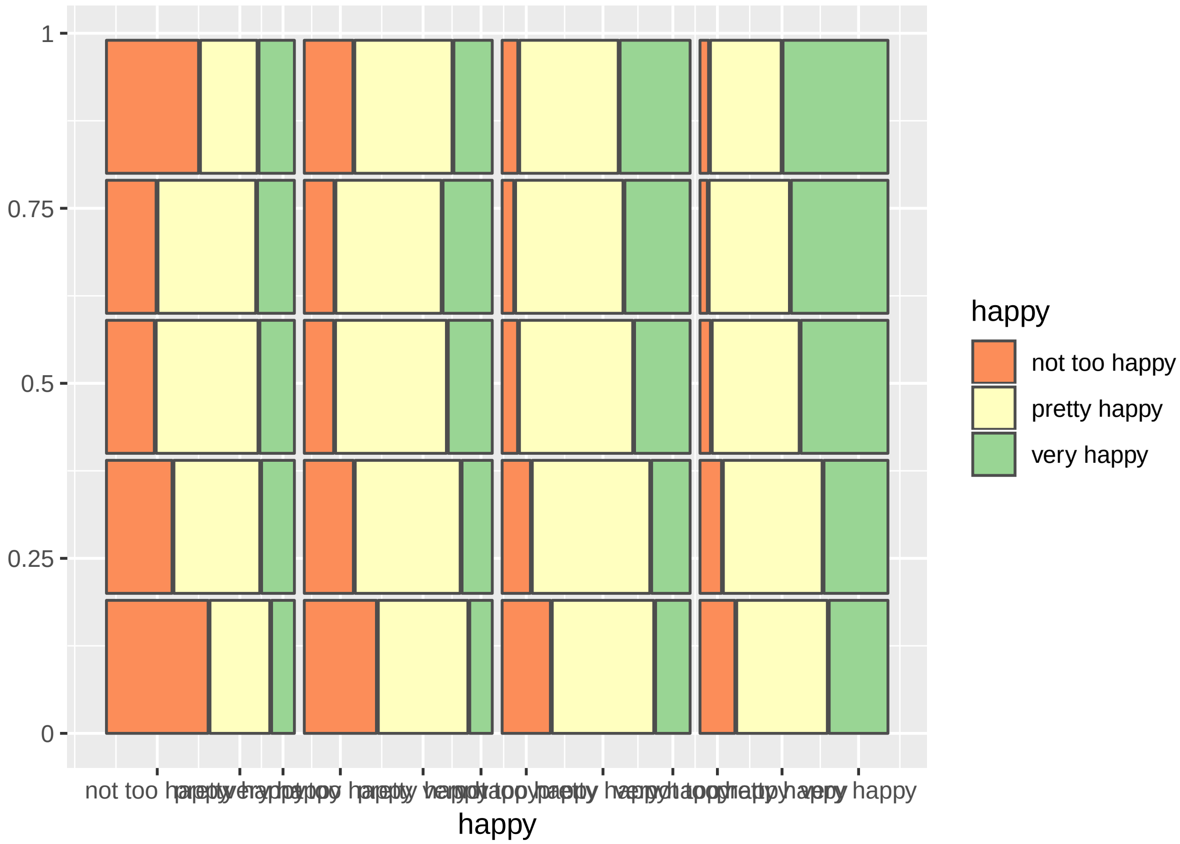


We can no longer see the joint distribution of health and financial status, but it is much easier to see the conditional distribution of happiness. Healthier and richer people are happier: maybe money does buy happiness?

Conditioning on financial status and health produces this plot (equal bin size plot) and makes it easier to see the conditional distribution of happiness given sex and health, because comparing positions along a common scale is an easier perceptual task. Depending on the comparison we are most interested in, we can make it easier to compare across wealth given health, or health given wealth, as in the next figure.

f(happy | health, finrela), partitioned with a hspine and fluct

```
prodplot(newhappy, ~ happy | finrela + health, c("hspine", "fluct"))+
  aes(fill=happy) +
  scale_fill_brewer(palette="Spectral")
```

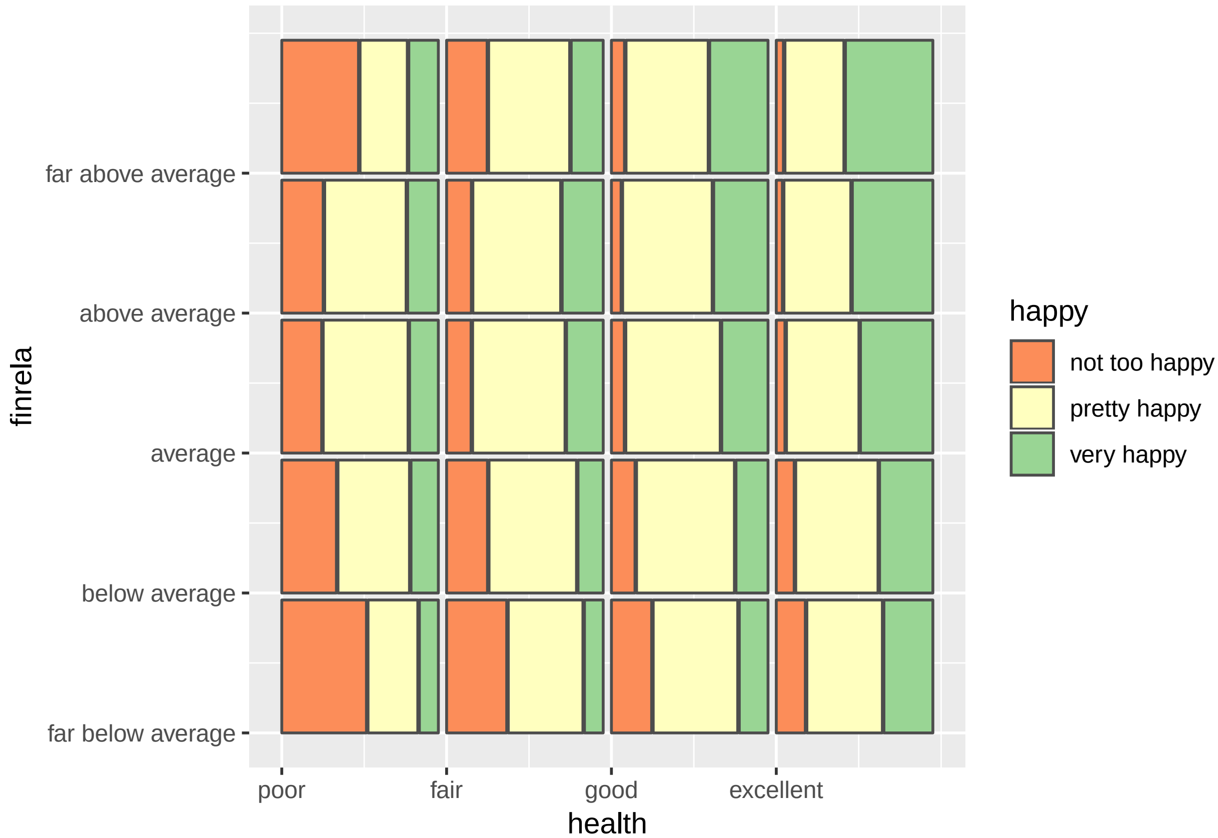


Label issues...

```
p = myprodplot(newhappy, "~ happy | finrela + health", c("hspine", "fluct"), "health", "finrela", "happy
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

```
p
```



f(happy | health, finrela) partitioned with a fluct and hspine, emphasizing the relationship of happiness with finances, whereas the previous plot emphasizes the relationship with health.

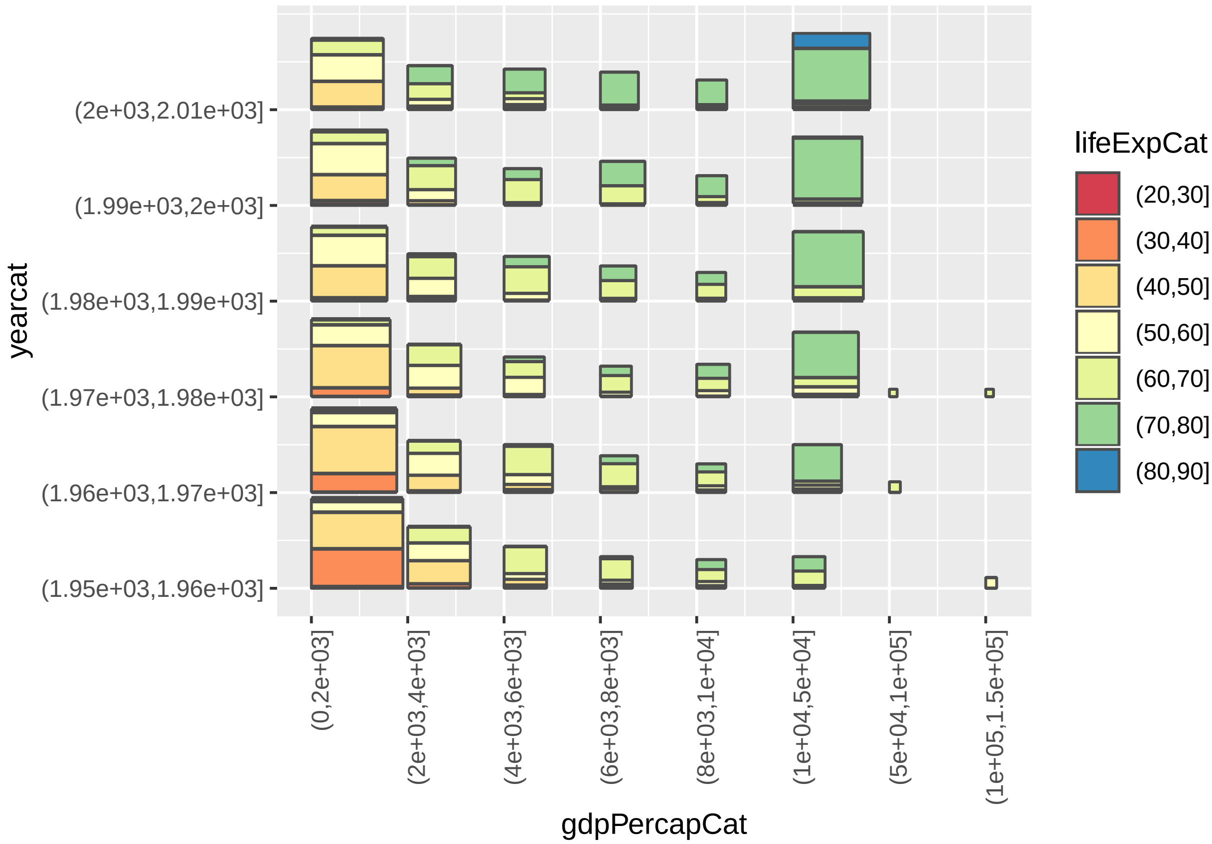
Here we see that for a fixed income level, better health is correlated to increased happiness. The same is not true for a fixed level of health: rich people with poor health seem to be less happy than poorer people in poor health.

Using gapminder data

```
library(gapminder)
gm = gapminder

# first need to discretise
gm$yearcat = cut(gm$year, c(1950, 1960, 1970, 1980, 1990, 2000, 2010))
gm$lifeExpCat = cut(gm$lifeExp, c(20, 30, 40, 50, 60, 70, 80, 90))
gm$popCat = cut(gm$pop, c(0, 1000000, 10000000, 100000000, 1000000000, 10000000000))
gm$gdpPercapCat = cut(gm$gdpPercap, c(0, 2000, 4000, 6000, 8000, 10000, 50000, 100000, 150000))

p = myprodplot(gm, "~ lifeExpCat + yearcat + gdpPercapCat", c("vspine", "fluct"), "gdpPercapCat", "yearc
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

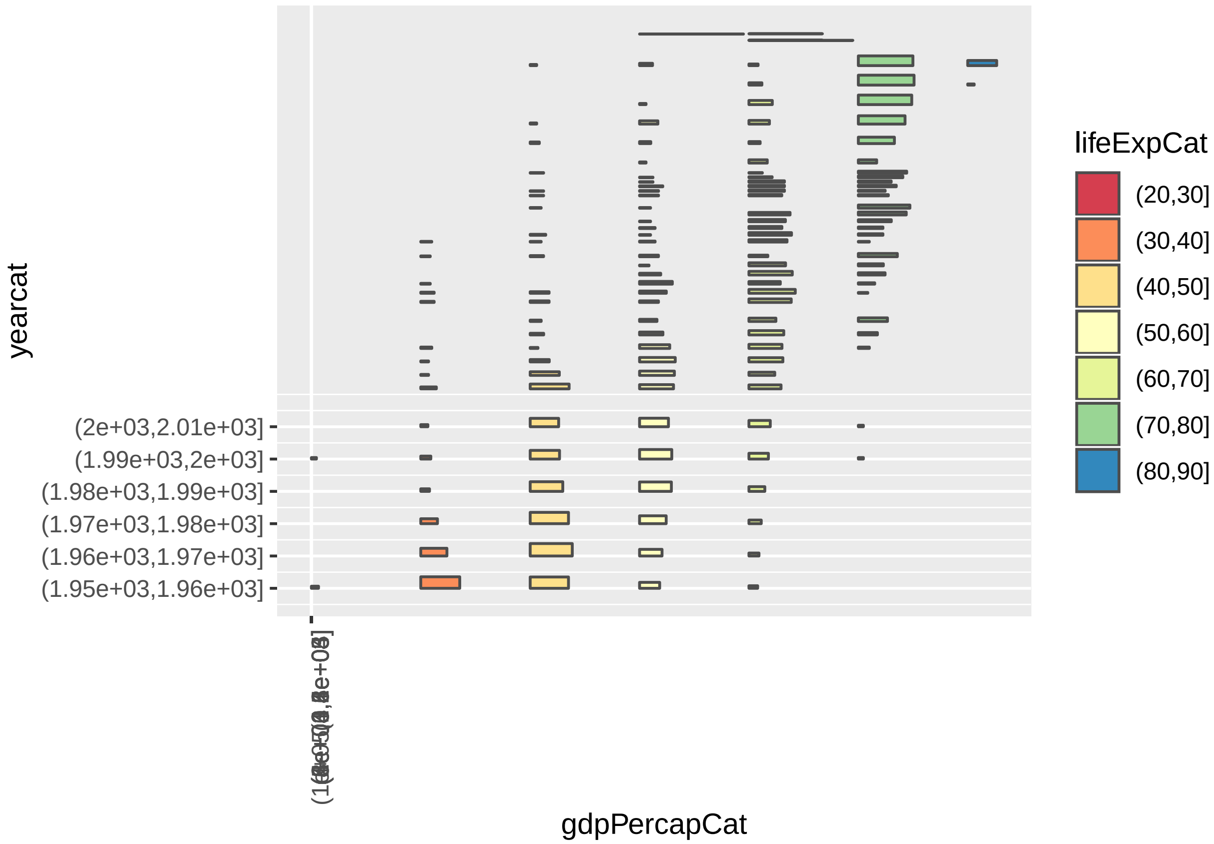
```
p
```




```
r = myprodplot(gm, "~ lifeExpCat + yearcat + gdpPercapCat", c("fluct", "vspine"), "gdpPercapCat", "yearc
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

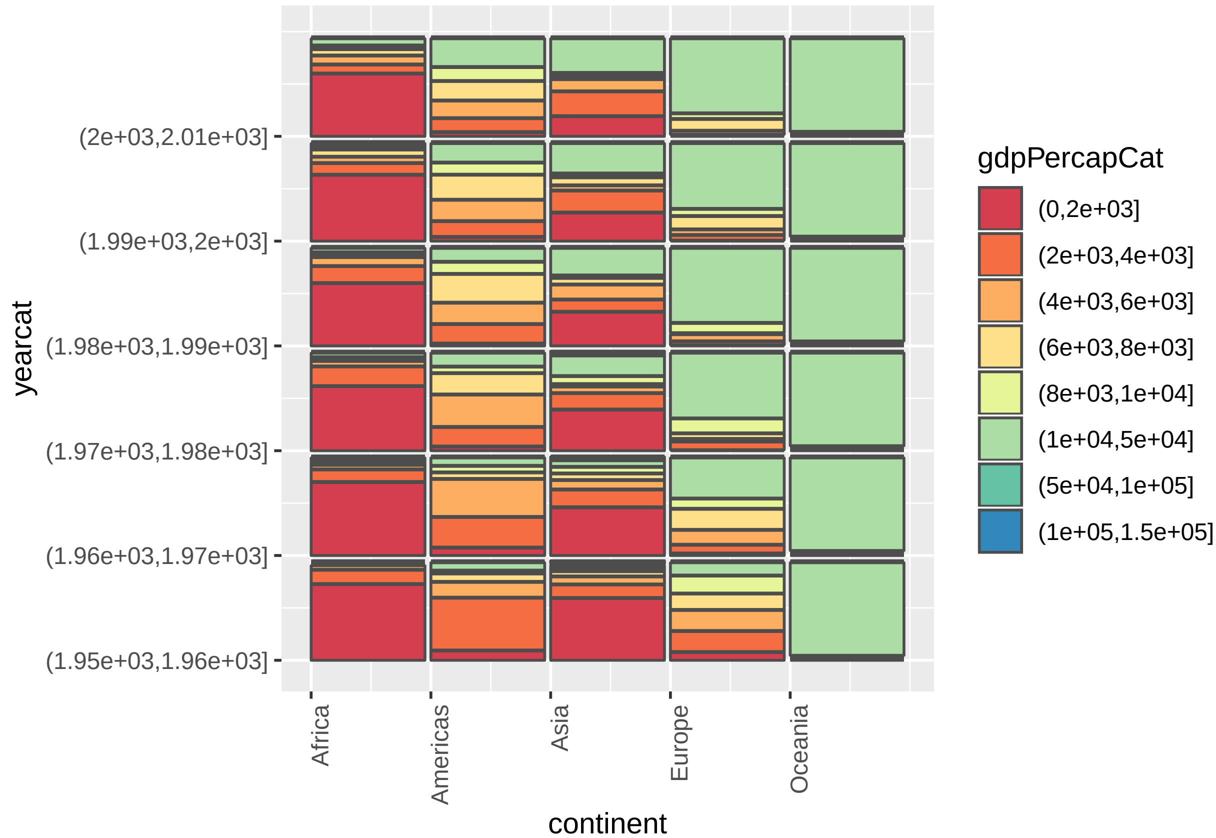
```
r
```




```
q = myprodplot(gm, "~ gdpPercapCat | yearcat + continent", c("vspine", "fluct"), "continent", "yearcat"
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

```
q
```



yearcat
(2e+03,2.01e+03]
(1.99e+03,2e+03]
(1.98e+03,1.99e+03]
(1.97e+03,1.98e+03]
(1.96e+03,1.97e+03]
(1.95e+03,1.96e+03]
Africa
Americas
Asia
Europe
Oceania
continent
gdpPercapCat
(0,2e+03]
(2e+03,4e+03]
(4e+03,6e+03]
(6e+03,8e+03]
(8e+03,1e+04]
(1e+04,5e+04]
(5e+04,1e+05]
(1e+05,1.5e+05]