BUS464: VISUALIZATION IN R

Roger Bivand

24 April 2018

- Today: binning intervals; colour use and palettes; and conditioning (trellis, lattice, facets)
- Tomorrow: base graphics; grid graphics; interactive graphics
- Colour use and palettes thanks to Achim Zeileis (Innsbruck via appear.in; Somewhere over the Rainbow: How to Make Effective Use of Colors in Statistical Graphics)
- After the lunch break, we'll go on to look at conditioning, also known as using facets

Binning intervals

BINNING (CLASS) INTERVALS

- We saw yesterday that histograms and many other kinds of chart require user choices about the number of bins to be used and bin/class intervals
- This may also be termed quantization: the division of part of the real line on which we have measured a variable into intervals
- This can also apply to combined categories if they are recoded to reduce the number of alternatives to be displayed
- Class intervals are much used in thematic cartography, and I'm the author of the **classInt** package

RECREATING DATA OBJECTS FROM MONDAY

```
> QQQ1 <- readRDS("../mon/dicook/pisa_subset.rds")
> countries <- readRDS("../mon/dicook/countries.rds")
> countries$Alpha_2[countries$Name == "Kosovo"] <- "XK"
> a0 <- split(QQQ1, list(QQQ1$CNT, QQQ1$ST004D01T))
```

```
> math_mean <- sapply(a0, function(x) weighted.mean(x$math_mean, w=x$SENWT))
> n2 <- length(math_mean)/2
> country <- sapply(strsplit(names(math_mean), "\\."), "[", 1)[1:n2]
> co <- match(country, countries$CNT)
> gender <- factor(sapply(strsplit(names(math_mean), "\\."), "[", 2))
```

RECREATING DATA OBJECTS FROM MONDAY

```
> library(matrixStats)
> sqn <- sqrt(sapply(a0, function(x) sum(x$SENWT)))
> math_se <- sapply(a0, function(x) weightedSd(x$math_mean, w=x$SENWT))/sqn
```

```
> read_mean <- sapply(a0, function(x) weighted.mean(x$read_mean, w=x$SENWT))
```

```
> sci_mean <- sapply(a0, function(x) weighted.mean(x$sci_mean, w=x$SENWT))
```

RECREATING DATA OBJECTS FROM MONDAY

```
> math_mean_female <- math_mean[gender == "Female"]
> math_mean_male <- math_mean[gender == "Male"]
> math_se_female <- math_se[gender == "Female"]
> math_se_male <- math_se[gender == "Male"]
> math_gap <- math_mean_female - math_mean_male
> read_mean_female <- read_mean[gender == "Female"]
> read_mean_male <- read_mean[gender == "Male"]
> read_gap <- read_mean_female - read_mean_male
> sci_mean_female <- sci_mean[gender == "Female"]
> sci_mean_male <- sci_mean[gender == "Male"]
> sci_gap <- sci_mean_female - sci_mean_male
> df <- data.frame(math_mean_female, math_mean_male, math_se_female, math_se_male,
+   read_mean_female, read_mean_male, sci_mean_female, sci_mean_male,
+   math_gap, read_gap, sci_gap, iso_a3=country, iso_a2=countries$Alpha_2[co])
```

We use a 1:50m set of country boundaries (in the 1:110m set, Singapore disappears).

```
> library(rnaturalearth)
> library(rnaturalearthdata)
> data(countries50)
> library(sf)
```

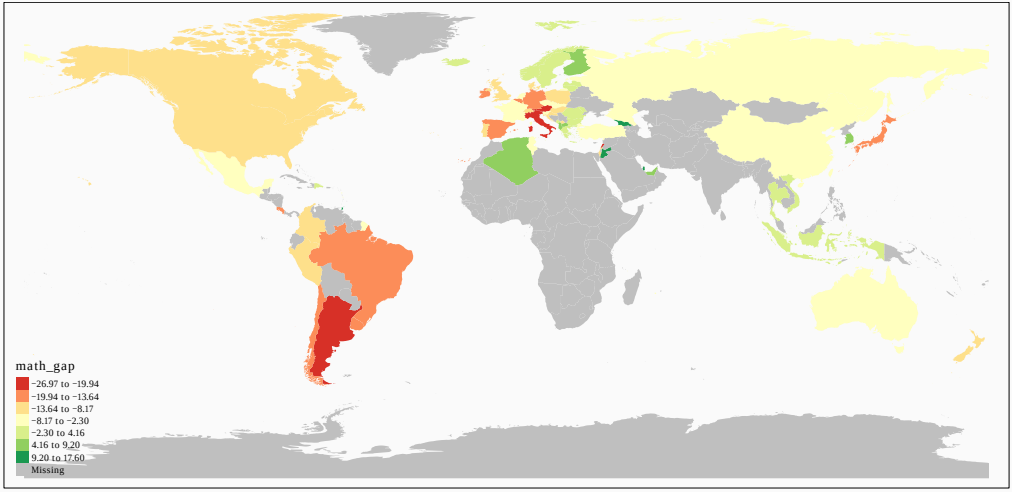
```
## Linking to GEOS 3.6.2, GDAL 2.2.4, proj.4 5.0.0
```

```
> countries50 <- st_as_sf(countries50)
> countries50$iso_a2[countries50$name == "Kosovo"] <- "XK"
```

We can `merge()` the gaps data with the country boundaries, and the **tmap** package to create the map. In doing so yesterday, we did not look at the arguments. `palette=` sets the colour palette (yesterday `"div"` for diverging at zero), `n=` the intended number of bins/classes, `style=` the method for constructing the bins, and `auto.palette.mapping=` (yesterday `TRUE`) to split the bins on zero.

```
> world_gaps <- merge(countries50[!is.na(countries50$iso_a2),], df, by="iso_a2", all.x=TRUE)
> library(tmap)
> tm_shape(world_gaps) + tm_fill("math_gap", palette="RdYlGn", n=7, style="jenks",
+   auto.palette.mapping=FALSE)
```

MATHS GAPS


math_gap
−26.97 to −19.94
−19.94 to −13.64
−13.64 to −8.17
−8.17 to −2.30
−2.30 to 4.16
4.16 to 9.20
9.20 to 17.60
Missing

- Class intervals can be chosen in many ways, and some have been collected for convenience in the **classInt** package
- The first problem is to assign class boundaries to values in a single dimension, for which many classification techniques may be used, including pretty, quantile, natural breaks among others, or even simple fixed values
- From there, the intervals can be used to generate colours from a colour palette, using the very nice `colorRampPalette()` function
- Because there are potentially many alternative class memberships even for a given number of classes, choosing a communicative set matters

We may choose the number of intervals ourselves arbitrarily or after examination of the data, or use provided functions, such as `nclass.Sturges()`, `nclass.scott()` or `nclass.FD()`. In `hist()`, `nclass.Sturges()` is used by default. We can also split on `sign()`, but handling diverging intervals often involves more work.

```
> x <- na.exclude(world_gaps$math_gap)
> nclass.Sturges(x)

## [1] 8

> nclass.scott(x)

## [1] 6

> (n <- nclass.FD(x))

## [1] 7

> nclass.Sturges(x[sign(x) == 1L])

## [1] 6

> nclass.Sturges(x[sign(x) == -1L])

## [1] 7
```

The default intervals for bins in `hist()` are `pretty(range(x), n = breaks, min.n = 1)`, where `breaks <- nclass.Sturges(x)`. The function computes a sequence of about n+1 equally spaced 'round' values which cover the range of the values in x. The values are chosen like values of coins or banknotes (1, 2, 5, etc.)

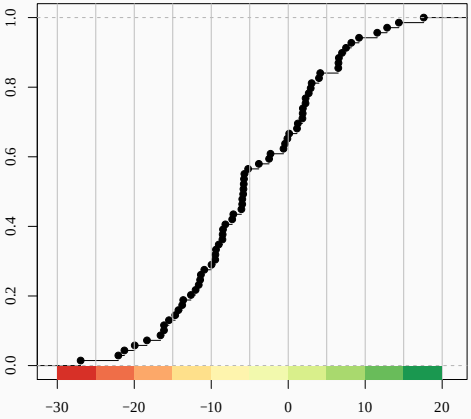
```
> n

## [1] 7

> range(x)

## [1] -26.96514  17.60435

> (p <-pretty(x, n=n))

##  [1] -30 -25 -20 -15 -10  -5   0   5  10  15  20

> length(p)

## [1] 11

> (p <-pretty(x, n=n, high.u.bias=3))

## [1] -30 -20 -10   0  10  20
```

If we use the `classIntervals()` function from **classInt**, we can pass through arguments to the function called through `style=`, and note that `n` will not necessarily be the number of output classes. By default, `intervalClosure=` is `"left"`, so `[-30, -20)` means numbers greater than and equal to (>=) -30 and less than (<) -20; `[10, 20]` is numbers >= 10 and <= 20.

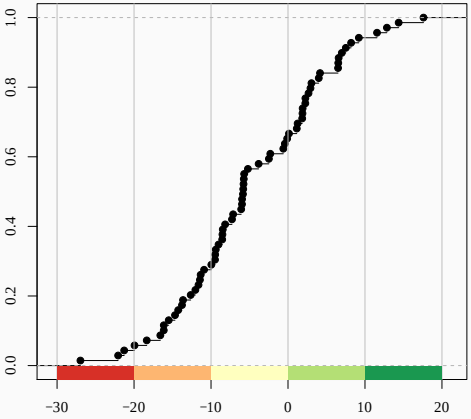
```
> suppressPackageStartupMessages(library(classInt))
> (ppd <- classIntervals(x, n=n, style="pretty",
+   cutlabels=TRUE))


## style: pretty
##   one of 49,280,065,120 possible partitions of this varia
## [-30,-25) [-25,-20) [-20,-15) [-15,-10)  [-10,-5)
##         1         2         6        10        20
##    [-5,0)     [0,5)    [5,10)   [10,15)   [15,20]
##         6        13         7         3         1


> (pp3 <- classIntervals(x, n=n, style="pretty",
+   high.u.bias=3, cutlabels=TRUE))


## style: pretty
##   one of 814,385 possible partitions of this variable int
## [-30,-20) [-20,-10)   [-10,0)    [0,10)   [10,20]
##         3        16        26        20         4
```

CLASS INTERVAL PLOT METHOD

Pretty (default)



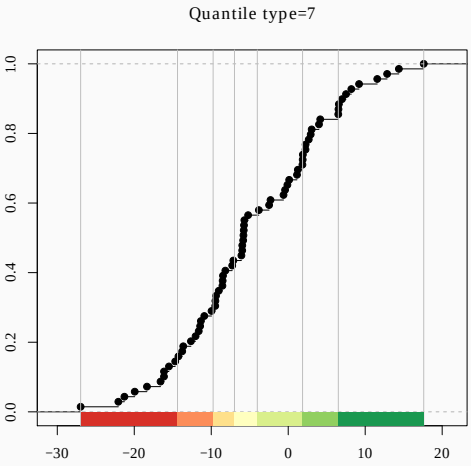
Pretty (compressed)

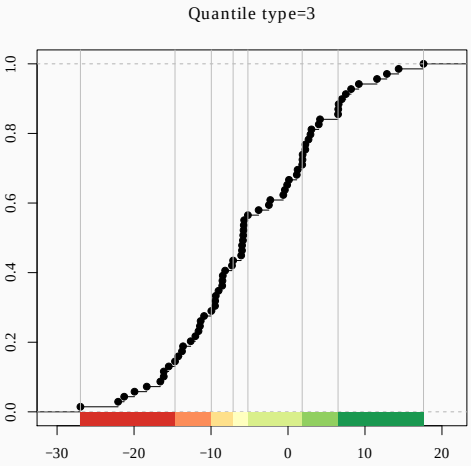


Quantiles may seem simple, but there are many ways of implementing the breaks, see `quantile()`. We can also set the `dataPrecision=` to make the class breaks easier to read:

```
> (pq7 <- classIntervals(x, n=n, style="quantile",
+   type=7L, dataPrecision=2))

## style: quantile
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-14.38)  [-14.38,-9.75)   [-9.75,-6.97)
##              10              10              10
##   [-6.97,-4.01)    [-4.01,1.87)     [1.87,6.51)
##               9              10              10
##    [6.51,17.61]
##              10

> (pq3 <- classIntervals(x, n=n, style="quantile",
+   type=3L, dataPrecision=2))

## style: quantile
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-14.67)  [-14.67,-9.96)   [-9.96,-7.12)
##               9              10              10
##    [-7.12,-5.2)     [-5.2,1.85)     [1.85,6.51)
##               9              10              10
##    [6.51,17.61]
##              11
```

CLASS INTERVAL PLOT METHOD



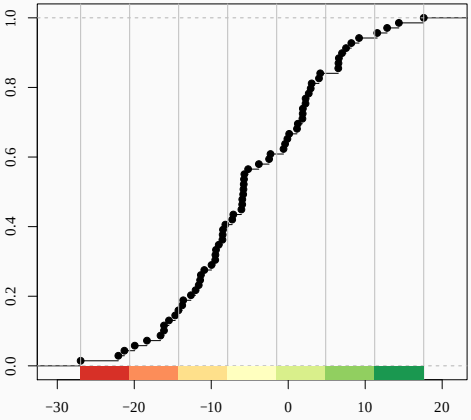


The `"equal"` style divides the range into `n` equal parts, while `"sd"` centres and scales the variable before using `pretty` on the result, converting back to get the breaks:

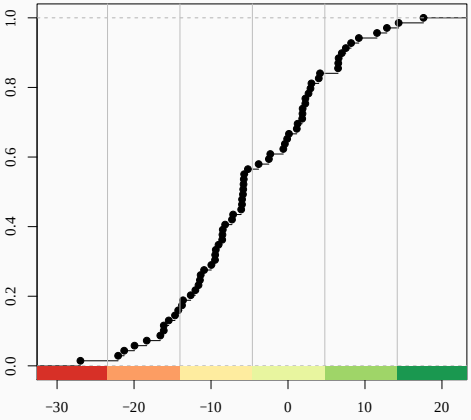
```
> (peq <- classIntervals(x, n=n, style="equal",
+   dataPrecision=2))

## style: equal
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-20.59) [-20.59,-14.23)  [-14.23,-7.86)
##               3               8              17
##   [-7.86,-1.49)    [-1.49,4.88)    [4.88,11.24)
##              14              16               7
##   [11.24,17.61]
##               4

> #diff(peq$brks)
> (psd <- classIntervals(x, n=n, style="sd",
+   high.u.bias=3, dataPrecision=2))

## style: sd
##   one of 10,424,128 possible partitions of this variable
## [-32.85,-23.44) [-23.44,-14.03)  [-14.03,-4.62)
##               1              10              28
##    [-4.62,4.79)     [4.79,14.2)    [14.2,23.61]
##              19               9               2
```

CLASS INTERVAL PLOT METHOD

Equal intervals



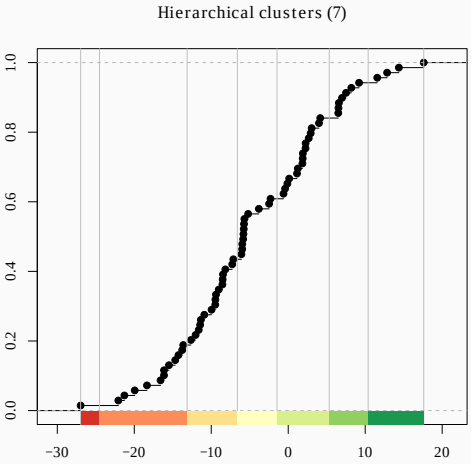
Standard deviations

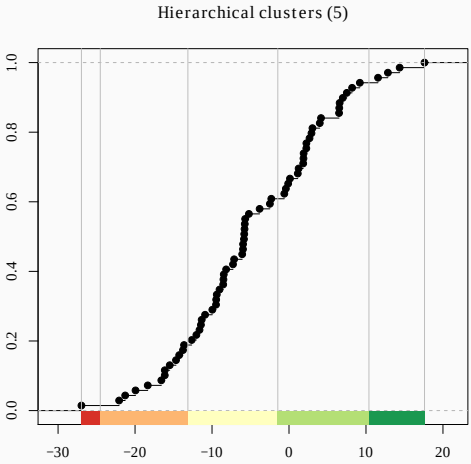


Hierarchical clustering is a well-known approach to trying to find similarities between multivariate observations based on dissimilarities or distances. If we use distances on the real line, we can build cluster trees for our univariate observations and chosen `method=`, and cut the trees for chosen numbers of classes:

```
> (phc7 <- classIntervals(x, n=n, style="hclust",
+   method="complete", dataPrecision=2))

## style: hclust
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-24.51) [-24.51,-13.13)   [-13.13,-6.6)
##               1              12              17
##    [-6.6,-1.45)    [-1.45,5.34)    [5.34,10.39)
##              12              16               7
##   [10.39,17.61]
##               4

> (phc5 <- getHclustClassIntervals(phc7, 5))

## style: hclust
##   one of 814,385 possible partitions of this variable int
## [-26.96514,-24.51651)  [-24.51651,-13.1365)
##                     1                   12
##  [-13.1365,-1.453571)  [-1.453571,10.38568)
##                    29                   23
##   [10.38568,17.60435]
##                     4
```

CLASS INTERVAL PLOT METHOD



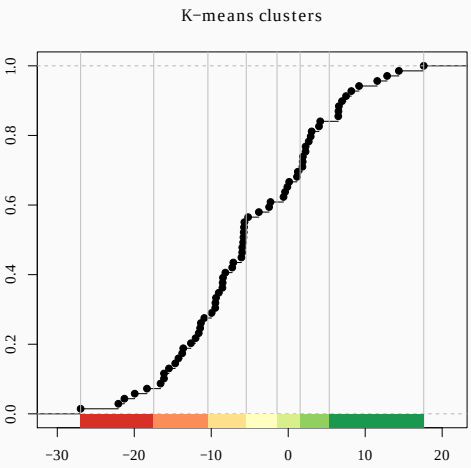


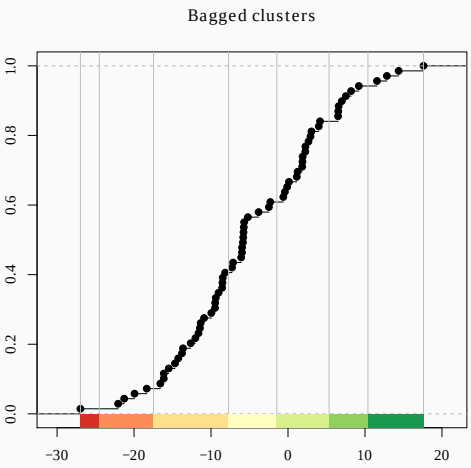
`kmeans` provides a non-hierarchical clustering approach, and can be combined with hierarchical clustering using bagged clustering provided by `e1071::bclust()`; here we are using `set.seed(1)` to be able to reproduce the output:

```
> (pk7 <- classIntervals(x, n=n, style="kmeans",
+   dataPrecision=2))

## style: kmeans
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-17.46) [-17.46,-10.43)  [-10.43,-5.45)
##               5              14              19
##   [-5.45,-1.45)    [-1.45,1.55)     [1.55,5.34)
##               4               6              10
##    [5.34,17.61]
##              11
```

```
> (pbc7 <- classIntervals(x, n=n, style="bclust",
+   verbose=FALSE, dataPrecision=2))

## style: bclust
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-24.51) [-24.51,-17.46)  [-17.46,-7.72)
##               1               4              23
##   [-7.72,-1.45)    [-1.45,5.34)    [5.34,10.39)
##              14              16               7
##   [10.39,17.61]
##               4
```

CLASS INTERVAL PLOT METHOD



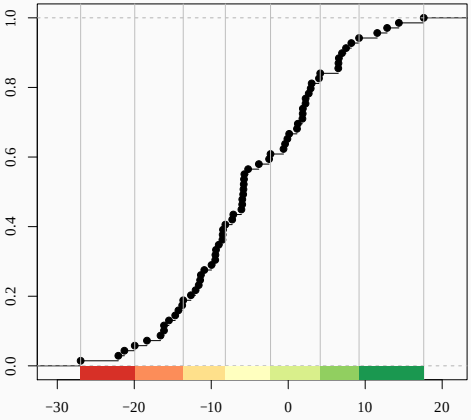


There are two implementations of the cartographic natural breaks approach, contributed by Hisaji Ono. As we have already seen, most methods for defining intervals differ in details. As with the cluster approaches, the intention is to place the breaks in obvious gaps in the distribution of the variable.

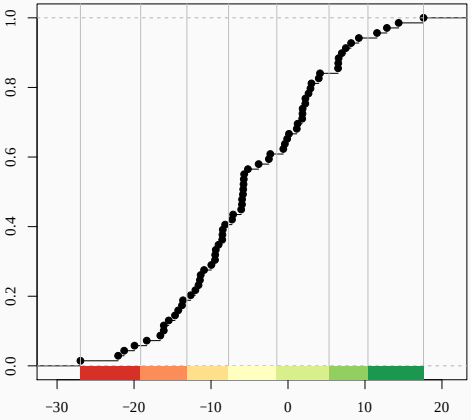
```
> (pj7 <- classIntervals(x, n=n, style="jenks",
+   dataPrecision=2))

## style: jenks
##   one of 109,453,344 possible partitions of this variable
## [-26.97,-19.94] (-19.94,-13.64]  (-13.64,-8.17]
##               4                9               15
##    (-8.17,-2.3]     (-2.3,4.15]      (4.15,9.2]
##              14               16               7
##      (9.2,17.6]
##               4

> (pf7 <- classIntervals(x, n=n, style="fisher",
+   dataPrecision=2))

## style: fisher
##   one of 109,453,344 possible partitions of this variable
## [-26.96,-19.14) [-19.14,-13.13)  [-13.13,-7.72)
##               4                9               15
##   [-7.72,-1.45)    [-1.45,5.34)    [5.34,10.39)
##              14               16               7
##   [10.39,17.61]
##               4
```

CLASS INTERVAL PLOT METHOD

Jenks natural breaks



Fisher natural breaks



Once found, the breaks need to be applied to the data vector (or vectors - data to be contrasted may need shared intervals). In **classInt** the `findCols()` function can be used, wrapping `base::findInterval()`; `base::cut()` could also be used:

```
> str(findCols(pj7))

##  num [1:69] 6 6 1 1 4 2 5 2 3 3 ...

> str(findInterval(x, pj7$brks, all.inside=TRUE,
+   left.open=TRUE))

##  int [1:69] 6 6 1 1 4 2 5 2 3 3 ...

> str(as.integer(cut(x, breaks=pj7$brks,
+   include.lowest=TRUE)))

##  int [1:69] 6 6 1 1 4 2 5 2 3 3 ...
```

HOW DOES GGPLOT DO THIS?

```
> suppressPackageStartupMessages(library(ggplot2))
> stat_bin


## function (mapping = NULL, data = NULL, geom = "bar", position = "stack",
##     ..., binwidth = NULL, bins = NULL, center = NULL, boundary = NULL,
##     breaks = NULL, closed = c("right", "left"), pad = FALSE,
##     na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
## {
##     layer(data = data, mapping = mapping, stat = StatBin, geom = geom,
##         position = position, show.legend = show.legend, inherit.aes = inherit.aes,
##         params = list(binwidth = binwidth, bins = bins, center = center,
##             boundary = boundary, breaks = breaks, closed = closed,
##             pad = pad, na.rm = na.rm, ...))
## }
## <environment: namespace:ggplot2>
```

HOW DOES GGPLOT DO THIS?

```
> StatBin


## <ggproto object: Class StatBin, Stat>
##     aesthetics: function
##     compute_group: function
##     compute_layer: function
##     compute_panel: function
##     default_aes: uneval
##     extra_params: na.rm
##     finish_layer: function
##     non_missing_aes:
##     parameters: function
##     required_aes: x
##     retransform: TRUE
##     setup_data: function
##     setup_params: function
##     super:  <ggproto object: Class Stat>
```

HOW DOES GGPLOT DO THIS?

```
> cat(capture.output(print(StatBin$compute_group))[-(1:5)], sep="\n")
```

```
##   <Inner function (f)>
##     function (data, scales, binwidth = NULL, bins = NULL, center = NULL,
##     boundary = NULL, closed = c("right", "left"), pad = FALSE,
##     breaks = NULL, origin = NULL, right = NULL, drop = NULL,
##     width = NULL)
## {
##     if (!is.null(breaks)) {
##         bins <- bin_breaks(breaks, closed)
##     }
##     else if (!is.null(binwidth)) {
##         bins <- bin_breaks_width(scales$x$dimension(), binwidth,
##             center = center, boundary = boundary, closed = closed)
##     }
##     else {
##         bins <- bin_breaks_bins(scales$x$dimension(), bins, center = center,
##             boundary = boundary, closed = closed)
##     }
##     bin_vector(data$x, bins, weight = data$weight, pad = pad)
## }
```

HOW DOES GGPLOT DO THIS?

```
> ggplot2:::bin_breaks_bins

## function (x_range, bins = 30, center = NULL, boundary = NULL,
##     closed = c("right", "left"))
## {
##     stopifnot(length(x_range) == 2)
##     bins <- as.integer(bins)
##     if (bins < 1) {
##         stop("Need at least one bin.", call. = FALSE)
##     }
##     else if (bins == 1) {
##         width <- diff(x_range)
##         boundary <- x_range[1]
##     }
##     else {
##         width <- (x_range[2] - x_range[1])/(bins - 1)
##     }
##     bin_breaks_width(x_range, width, boundary = boundary, center = center,
##         closed = closed)
## }
## <environment: namespace:ggplot2>
```

HOW DOES GGPLOT DO THIS?

```
> cat(capture.output(print(ggplot2:::bin_breaks_width))[-(4:8)], sep="\n")

## function (x_range, width = NULL, center = NULL, boundary = NULL,
##     closed = c("right", "left"))
## {
##     if (!is.null(boundary) && !is.null(center)) {
##         stop("Only one of 'boundary' and 'center' may be specified.")
##     }
##     else if (is.null(boundary)) {
##         if (is.null(center)) {
##             boundary <- width/2
##         }
##         else {
##             boundary <- center - width/2
##         }
##     }
##     x_range <- as.numeric(x_range)
##     width <- as.numeric(width)
##     boundary <- as.numeric(boundary)
##     shift <- floor((x_range[1] - boundary)/width)
##     origin <- boundary + shift * width
##     max_x <- x_range[2] + (1 - 1e-08) * width
##     breaks <- seq(origin, max_x, width)
##     bin_breaks(breaks, closed = closed)
## }
```

HOW DOES GGPLOT DO THIS?

```
> ggplot2:::bin_breaks


## function (breaks, closed = c("right", "left"))
## {
##     bins(breaks, closed)
## }
## <environment: namespace:ggplot2>
```

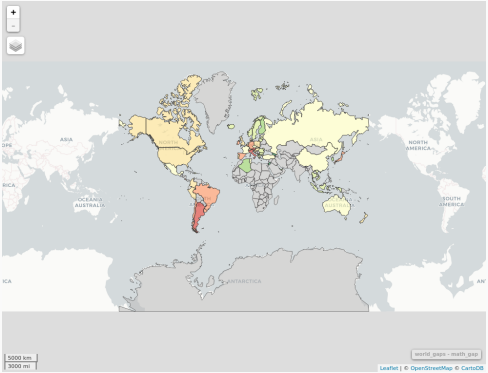
HOW DOES GGPLOT DO THIS?

```
> ggplot2:::bins
```

```
## function (breaks, closed = c("right", "left"), fuzz = 1e-08 *
##     stats::median(diff(breaks)))
## {
##     stopifnot(is.numeric(breaks))
##     closed <- match.arg(closed)
##     breaks <- sort(breaks)
##     if (closed == "right") {
##         fuzzes <- c(-fuzz, rep.int(fuzz, length(breaks) - 1))
##     }
##     else {
##         fuzzes <- c(rep.int(-fuzz, length(breaks) - 1), fuzz)
##     }
##     structure(list(breaks = breaks, fuzzy = breaks + fuzzes,
##         right_closed = closed == "right"), class = "ggplot2_bins")
## }
## <environment: namespace:ggplot2>
```

We can also pass palettes and class intervals through (maybe need to stretch the min-max interval bounds)

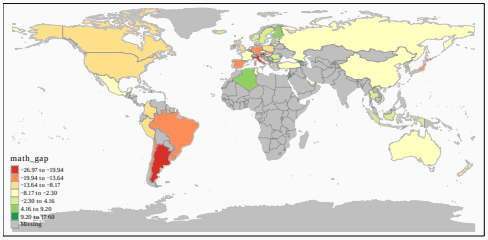
```
> library(mapview)
> m <- mapview(world_gaps, zcol="math_gap",
+   col.regions=pal, at=pj7$brks)
> mapshot(m, file=paste0(getwd(), "/map1.png"))
```



- The tmap package provides cartographically informed, grammar of graphics (gg) based functionality now, like **ggplot2** using **grid** graphics.
- John McIntosh tried with ggplot2, with quite nice results
- I suggested he look at **tmap**, and things got better, because **tmap** can switch between interactive and static viewing
- **tmap** also provides direct access to **classInt** class intervals

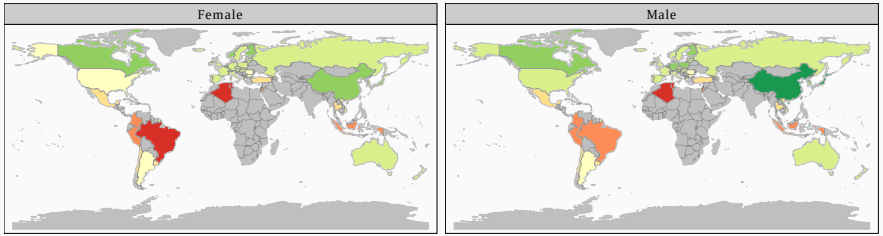
Like the `sf::plot()` method, **tmap** plotting can use **classInt** internally and accepts a palette (try looking at `tmaptools::palette_explorer()` for ColorBrewer palettes):

```
> library(tmap)
> tm_shape(world_gaps) + tm_fill("math_gap",
+   n=n, style="jenks", palette=pal) +
+   tm_borders(lwd=0.5, alpha=0.4)
```



If we give a short vector of column names, we will get facet/panel displays, but need to use `tm_facets(free.scales=FALSE)` to use the same class intervals

```
> tm_shape(world_gaps) + tm_fill(c("math_mean_female", "math_mean_male"), n=7, style="jenks", palette=pal) +
+   tm_facets(free.scales=FALSE) + tm_borders(lwd=0.5, alpha=0.4) + tm_layout(panel.labels=c("Female", "Male"))
```



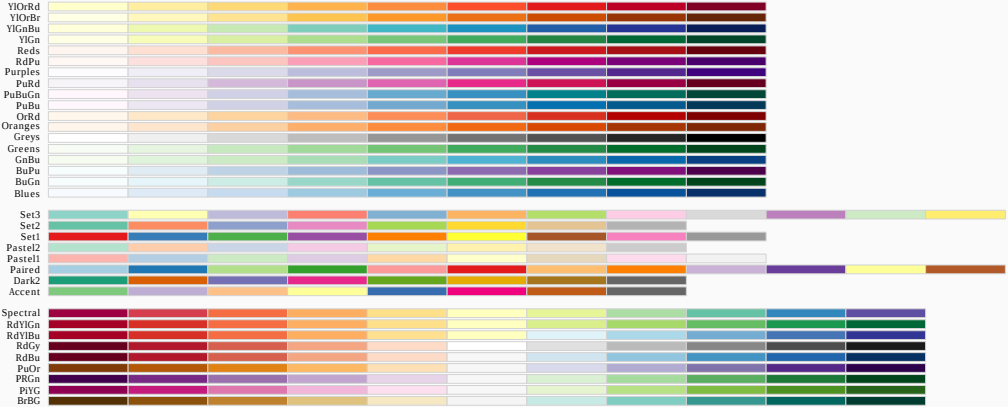
Somewhere over the Rainbow

- Zeileis and others have discussed the opportunities to be found for effective visual communication in statistical graphics using colours; we distinguish between sequential, diverging and qualitative palettes
- There are differences in opinions with regard to colour look-up (continuous or discrete) between implementations - RColorBrewer is discrete
- There are several palettes, such as `rainbow`, `cm.colors`, `heat.colors` and others in grDevices
- Other packages include viridis, colorspace, and further palettes provided in plotting methods (`sf.colors` uses `sp::bpy.colors`)

Fortunately, the **RColorBrewer** package gives by permission access to the ColorBrewer palettes accesible from the ColorBrewer website. Note that ColorBrewer limits the number of classes tightly, only 3–9 sequential classes

```
> library(RColorBrewer)
> display.brewer.all()
```

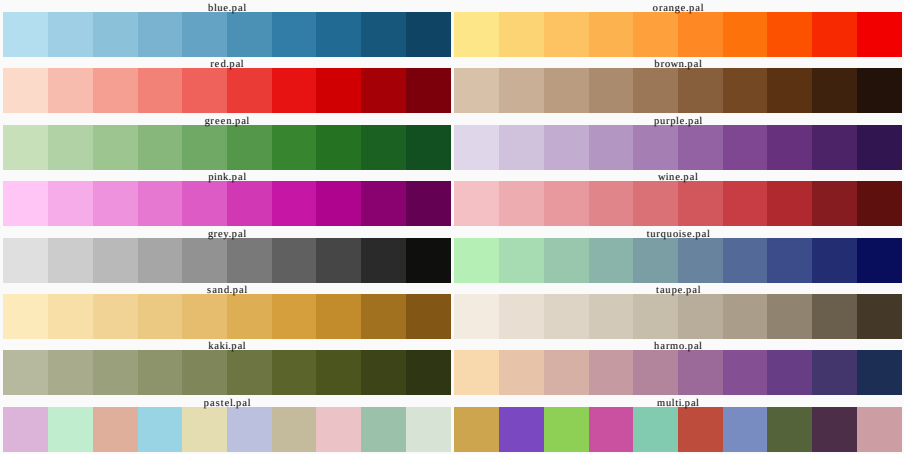
COLORBREWER



VIRIDIS


terrain.colors
heat
ggplot default
brewer blues
brewer yellow–green–blue
viridis
magma

CARTOGRAPHY



Trellis/lattice/facets: conditioning

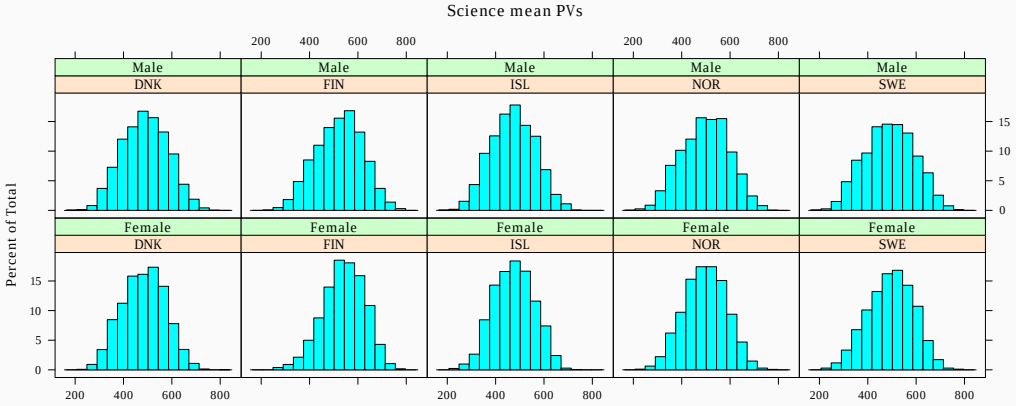
- The underlying logic of conditioned graphics is that multiple displays (windows, panes) use the same scales and representational elements for comparison
- Using the same scales and representational elements for comparison can be done manually, imposing the same scales, colours and shapes in each plot and laying the plots out in a grid
- Trellis graphics automated this in S, and **lattice** provides similar but enhanced facilities in R with a formula interface
- **ggplot2** and other packages also provide similar functionalities

To try this out, let's use about 25,000 PISA Science PV means by country, gender and count of books at home, choosing the appropriate country and gender `data.frame` objects from the big list:

```
> nordics <- c("DNK", "FIN", "ISL", "NOR", "SWE")
> a1 <- a0[which(sapply(strsplit(names(a0), "\\."),
+   "[", 1) %in% nordics)]
> a11 <- do.call("rbind", a1)
> a11$CNT <- droplevels(a11$CNT)
> levels(a11$ST013Q01TA) <- sub("More than",
+   ">", sub(" books", "", levels(a11$ST013Q01TA)))
> saveRDS(a11, "../mon/dicook/a11.rds")
> library(lattice)
> library(ggplot2)
```

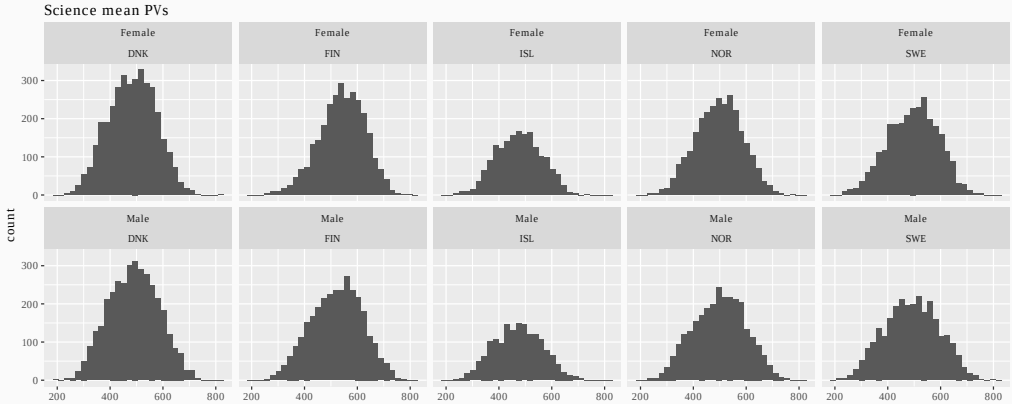
LATTICE - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> histogram(~ sci_mean | CNT*ST004D01T, data=a11, main="Science mean PVs", xlab="")
```



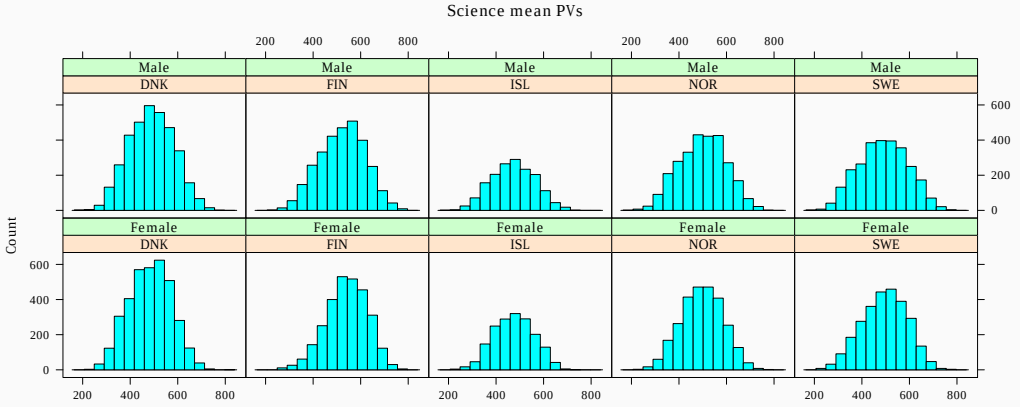
GGPLOT - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> ggplot(a11, aes(x=sci_mean)) + geom_histogram() + facet_wrap(~ ST004D01T + CNT, nrow=2) +
+   ggtitle("Science mean PVs") + xlab("")

## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



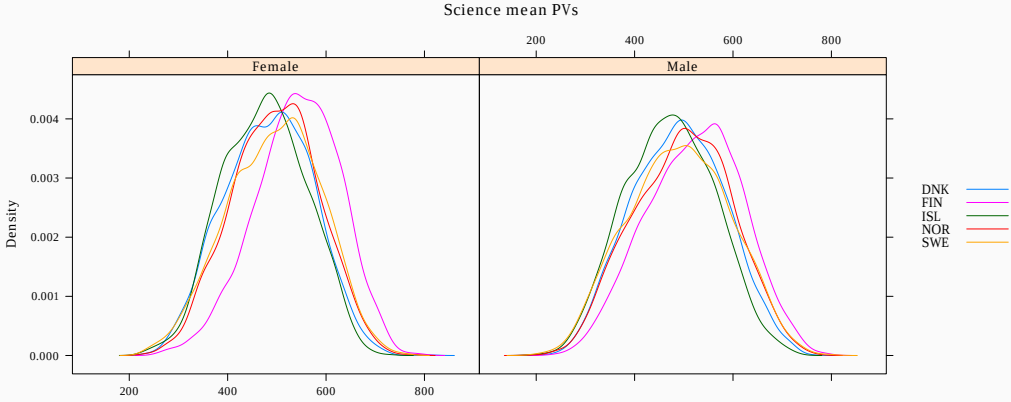
LATTICE - SCIENCE MEAN PVS BY GENDER AND COUNTRY (COUNTS)

```
> histogram(~ sci_mean | CNT*ST004D01T, data=a11, main="Science mean PVs", xlab="", type="count")
```



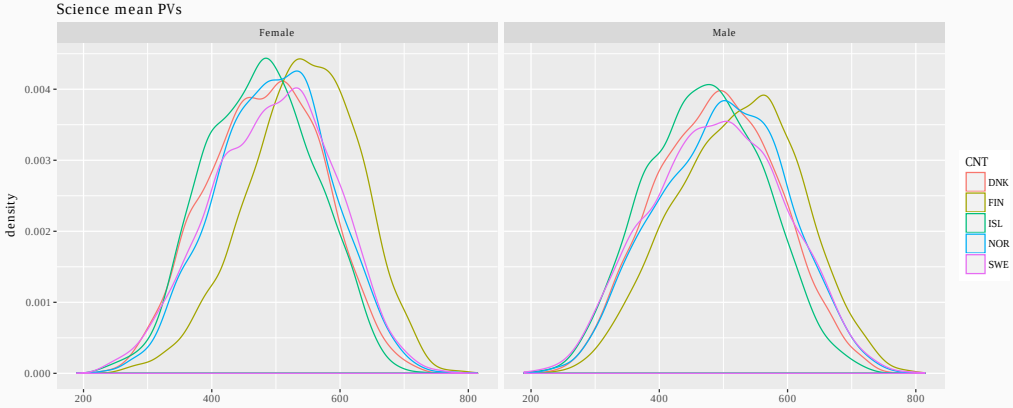
LATTICE - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> densityplot(~ sci_mean | ST004D01T, groups=CNT, data=a11, auto.key=list(space="right"),
+   plot.points=FALSE, main="Science mean PVs", xlab="")
```



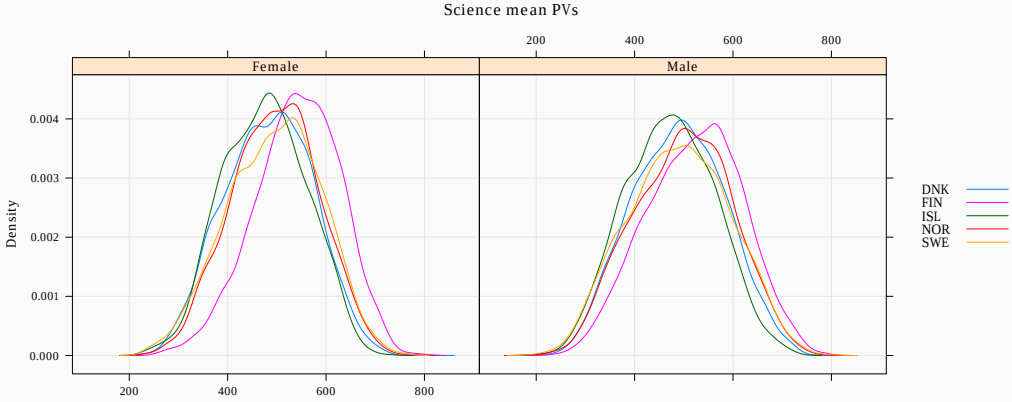
GGPLOT - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> ggplot(a11, aes(x=sci_mean)) + geom_density(aes(colour=CNT)) + facet_wrap(~ ST004D01T, ncol=2) +
+   ggtitle("Science mean PVs") + xlab("")
```



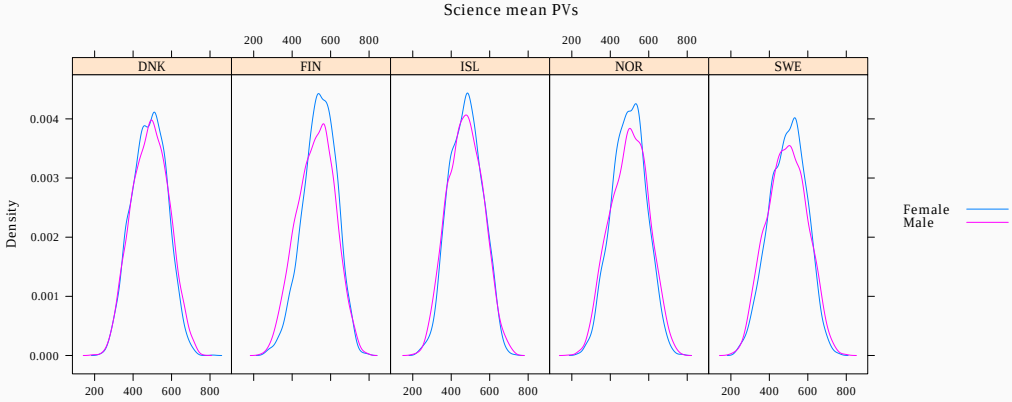
LATTICE - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> densityplot(~ sci_mean | ST004D01T, groups=CNT, data=a11, auto.key=list(space="right"),
+   plot.points=FALSE, main="Science mean PVs", xlab="", type=c("l", "g"))
```



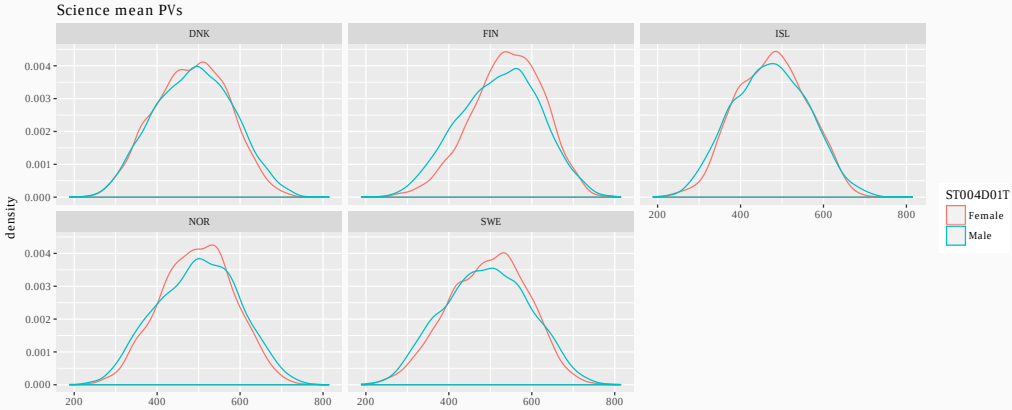
LATTICE - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> densityplot(~ sci_mean | CNT, groups=ST004D01T, a11, auto.key=list(space="right"),
+   plot.points=FALSE, main="Science mean PVs", xlab="")
```



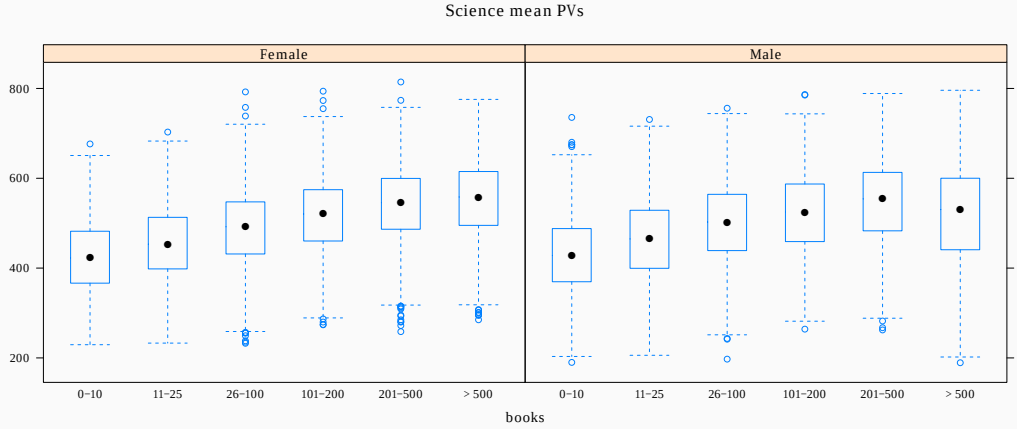
GGPLOT - SCIENCE MEAN PVS BY GENDER AND COUNTRY

```
> ggplot(a11, aes(x=sci_mean)) + geom_density(aes(colour=ST004D01T)) + facet_wrap(~ CNT, ncol=3) +
+   ggtitle("Science mean PVs") + xlab("")
```



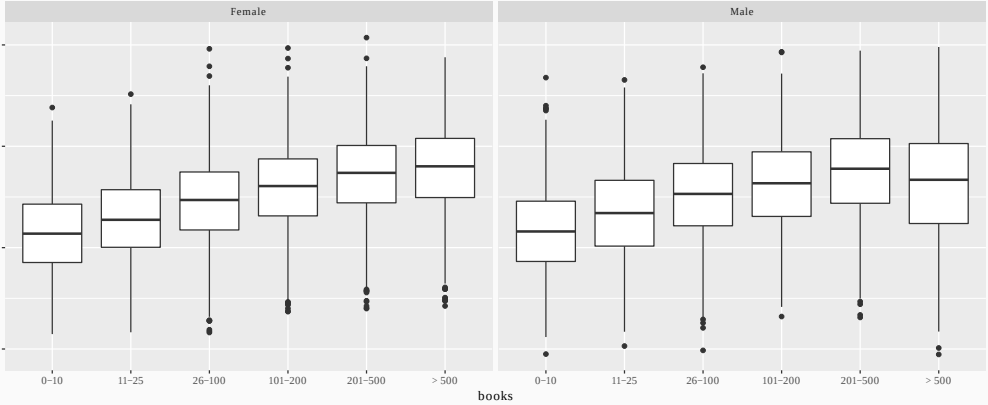
LATTICE - SCIENCE MEAN PVS BY GENDER AND BOOKS AT HOME

```
> bwplot(sci_mean ~ ST013Q01TA | ST004D01T, a11, main="Science mean PVs", xlab="books", ylab="")
```



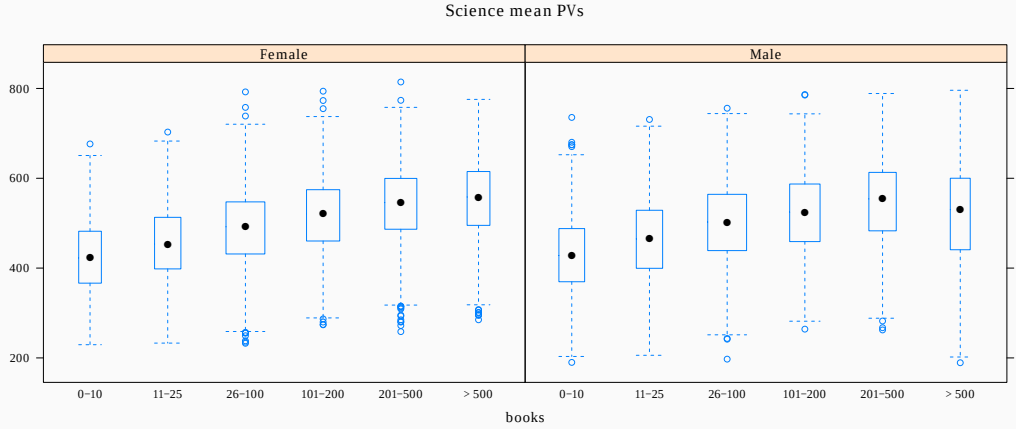
GGPLOT - SCIENCE MEAN PVS BY GENDER AND BOOKS AT HOME

```
> ggplot(na.omit(a11), aes(ST013Q01TA, sci_mean)) + geom_boxplot(aes(group=ST013Q01TA)) +
+   facet_wrap(~ ST004D01T) + ggtitle("Science mean PVs") + xlab("books") + ylab("")
```



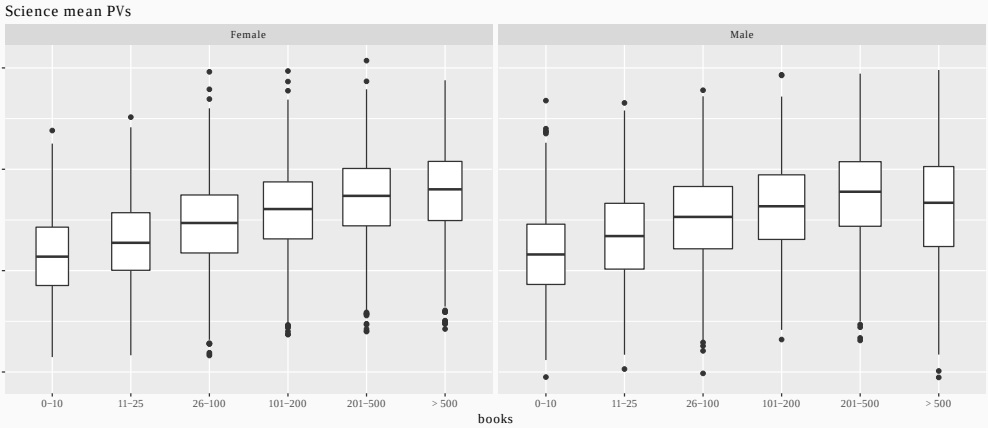
LATTICE - SCIENCE MEAN PVS BY GENDER AND BOOKS AT HOME

```
> bwplot(sci_mean ~ ST013Q01TA | ST004D01T, a11, main="Science mean PVs", xlab="books", ylab="", varwidth=TRUE)
```



GGPLOT - SCIENCE MEAN PVS BY GENDER AND BOOKS AT HOME

```
> ggplot(na.omit(a11), aes(ST013Q01TA, sci_mean)) + geom_boxplot(aes(group=ST013Q01TA), varwidth=TRUE) +
+   facet_wrap(~ ST004D01T) + ggtitle("Science mean PVs") + xlab("books") + ylab("")
```



R's sessioninfo()

```
> sessionInfo()

## R version 3.4.4 (2018-03-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Fedora 27 (Workstation Edition)
##
## Matrix products: default
## BLAS: /home/rsb/topics/R/R344-share/lib64/R/lib/libRblas.so
## LAPACK: /home/rsb/topics/R/R344-share/lib64/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C               LC_TIME=en_GB.UTF-8
##  [4] LC_COLLATE=en_GB.UTF-8     LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] lattice_0.20-35         cartography_2.0.2       sp_1.2-8                colorspace_1.3-2
##  [5] viridis_0.5.1           viridisLite_0.3.0       ggplot2_2.2.1           RColorBrewer_1.1-2
##  [9] classInt_0.2-3          spData_0.2.8.3          tmap_1.11-2             sf_0.6-1
## [13] rnaturalearthdata_0.1.0 rnaturalearth_0.1.0     matrixStats_0.53.1      extrafont_0.17
##
```