BUS464: VISUALIZATION IN R

Roger Bivand

23 April 2018

Introduction

THIS SEMINAR/COURSE

- Visualization is one of the key strengths of R
- However, visualization involves many choices, and R offers such a wide range of choices that guidance is desirable
- There are two major underlying techologies, base graphics and grid graphics, and several toolboxes built on these (trellis graphics and grammar of graphics on grid graphics), in addition to JavaScript widgets for interaction
- In addition, much work has been done on the effective use of shapes and colours.

SCHEDULE

- Today, we'll borrow two talks from the Ihaka lecture series at Auckland University last month, review simple visualizations, and follow up some of the data used in the second talk by Dianne Cook (plus On-Ramp)
- Tuesday: binning, colours (appear.in with Achim Zeileis) and conditioning; Wednesday: base graphics and graphics devices, grid graphics with **lattice** and **ggplot2**, interactive graphics
- The underlying aim: to survey contemporary approaches to visualization using R
- Your (group) projects are a key part of the seminar

PROJECTS

- The group projects provide learning opportunities for exploring visualization, through comparison of implementation and maybe perception mini-experiments using **nullabor**
- Similar topics may be gleaned from R-bloggers and its Twitter feed; some of the claims deserve to be checked
- The aim is not to find winners, but to explore alternatives
- Thursday group work-day, Friday presentation/feedback day, hand in via WiseFlow by Friday 18 May 14:00.

LEARNING RESOURCES

- Needs for learning resources, and ways of making use of them, vary greatly between participants
- There are lots of books, but many now present one-size-fits-all solutions that may not be a best fit
- Other materials are described on the R site and on CRAN
- RStudio also provides an online learning page, with a number of options, like Datacamp and **swirl**

FINDING STUFF FOR INSTALLATION

- R is distributed from mirrors of the comprehensive R archive network (CRAN)
- The cloud mirror is the easiest, but a local server may be faster
- RStudio can be downloaded and installed after R has been installed
- R comes with many contributed packages — the ones we need are on CRAN, which lists them providing information; we'll get back to contributed packages later

Ihaka Lecture 3, 2018

Visual trumpery: How charts lie - and how they make us smarter

Simple visualizations

GRAPHICS AND VISUALIZATION IN R

- We can distinguish between presentation graphics and analytical graphics
- Presentation graphics are intended for others and are completed (even if interactive) - see work by Florence Nightingale
- Analytical graphics may evolve into presentation graphics, but their main purpose is to visualize the data being analysed (see Antony Unwin's book and website)
- Many of the researchers who have developed approaches to visualization have been involved with Bell Labs, where S came from

GRAPHICS AND VISUALIZATION IN R

- As installed, R provides two graphics approaches, one known as base graphics, the other trellis or lattice graphics
- Most types of visualization are available for both, but lattice graphics were conceived to handle conditioning, for example to generate matching plots for different categories
- Many of these were based on the data-ink ratio, favouring graphics with little or no extraneous detail (Edward Tufte) - see Lukasz Piwek's blog
- There are other approaches, such as Leland Wilkinson's Grammar of Graphics, implemented in Hadley Wickham's **ggplot2** package, which we will also be using here as its use is growing fast

GRAPHICS AND VISUALIZATION IN R

- So there are presentation and analytical graphics, and there can be a number of approaches to how best to communicate information in either of those modes
- R can create excellent presentation graphics, or provide input for graphics designers to improve for print or other media
- What we need now are the most relevant simple analytical graphics for the kinds of data we use

JACOBY: SYNTHETIC DATA AND METHODS

There are four numeric variables of the same length, read into a `data.frame`. The `summary` method shows their characteristics:

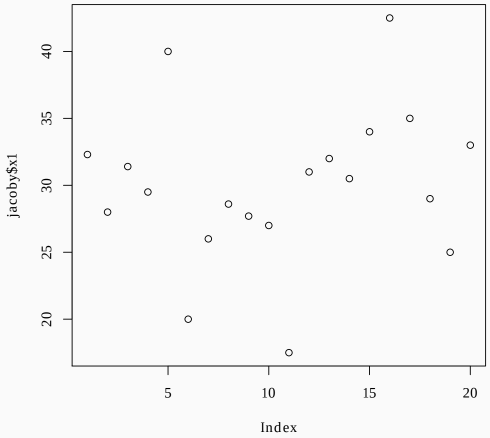
```
> jacoby <- read.table("data/jacoby.txt")
> summary(jacoby)
```

```
##        x1              x2              x3              x4       
##  Min.   :17.50   Min.   :21.60   Min.   :19.10   Min.   :22.10  
##  1st Qu.:27.52   1st Qu.:25.32   1st Qu.:25.98   1st Qu.:27.38  
##  Median :30.00   Median :30.00   Median :28.35   Median :29.60  
##  Mean   :30.00   Mean   :30.00   Mean   :30.00   Mean   :30.00  
##  3rd Qu.:32.48   3rd Qu.:34.67   3rd Qu.:34.12   3rd Qu.:30.23  
##  Max.   :42.50   Max.   :38.40   Max.   :40.90   Max.   :51.00  
```

SIMPLE PLOT METHODS

Using the `plot` method on a single numeric vector puts that variable on the vertical (y) axis, and an observation index on the horizontal (x) axis

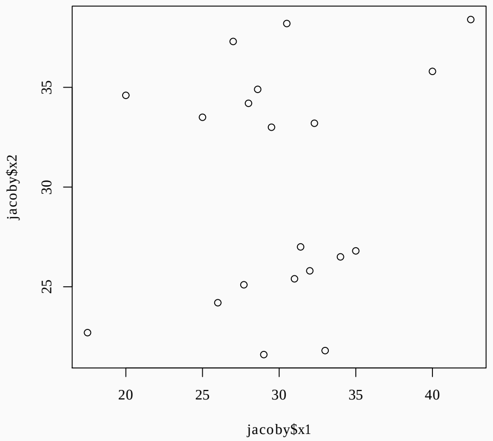
```
> plot(jacoby$x1)
```



SIMPLE PLOT METHODS

Using the `plot` method on two numeric vectors puts the first vector on the horizontal (x) axis, and the second on the vertical (y) axis

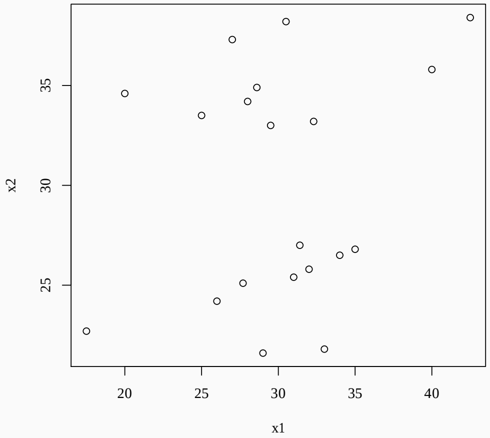
```
> plot(jacoby$x1, jacoby$x2)
```



SIMPLE PLOT METHODS

By convention, the x-axis is seen as causing the y-axis, so we can use a formula interface, and a `data` argument to tell the method where to find the vectors by name (column names in the data.frame)

```
> plot(x2 ~ x1, data=jacoby)
```



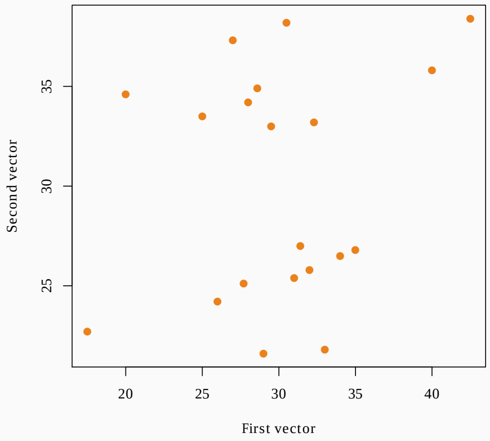
SIMPLE PLOT METHODS

- In these cases, the default plot shows points, which is reasonable here
- The default glyph (symbol) is a small unfilled circle, the `pch` argument lets us change this (and its colour by `col`)
- The display is annotated with value scales along the axes, and the axes are placed to add a little space outside the bounding box of the observations
- The display includes axis labels, but does not include a title

SIMPLE PLOT METHODS

Here, we'll manipulate the axis labels, and symbol type, colour and size; the symbol type, colour and size are also vectors, so can vary by observation or for groups of observations

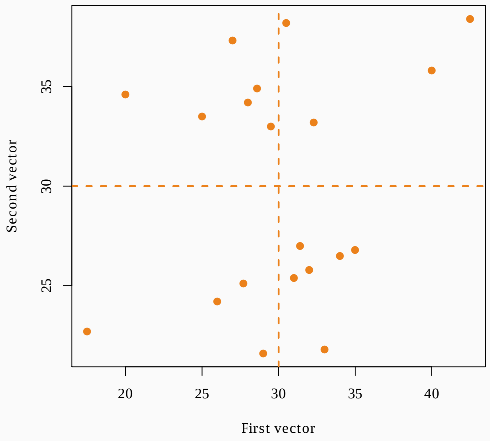
```
> plot(x2 ~ x1, data=jacoby, xlab="First vector",
+   ylab="Second vector", pch=16, col="#EB811B",
+   cex=1.2)
```



SIMPLE PLOT METHODS

Base graphics methods like plot can be added to, drawing more on the open device. We'll add lines with **abline** showing the means of the two vectors:

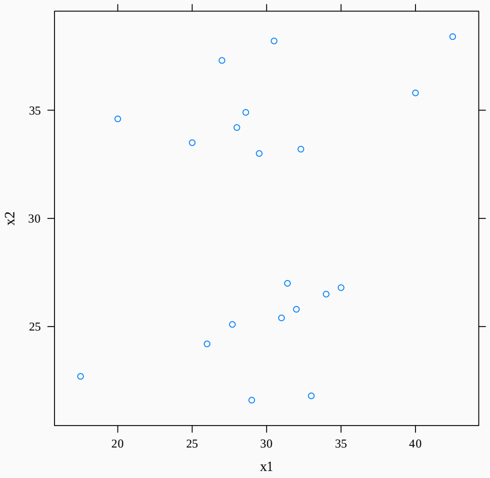
```
> plot(x2 ~ x1, data=jacoby, xlab="First vector",
+   ylab="Second vector", pch=16, col="#EB811B",
+   cex=1.2)
> abline(v=mean(jacoby$x1), h=mean(jacoby$x2),
+   lty=2, lwd=2, col="#EB811B")
```



SIMPLE PLOT METHODS

In **lattice** or trellis graphics, `xyplot()` is used for scatterplots; the formula interface is standard:

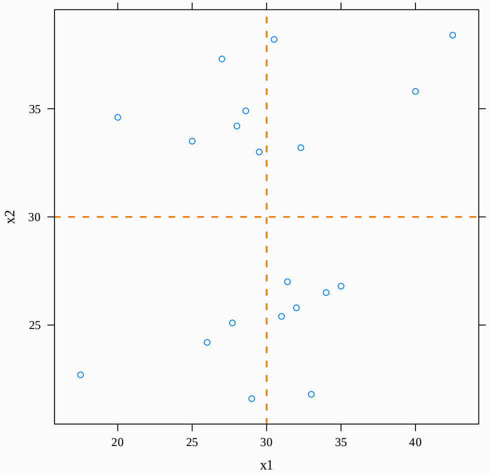
```
> library(lattice)
> xyplot(x2 ~ x1, jacoby)
```



SIMPLE PLOT METHODS

Extra elements may be manipulated by modifying the panel functions, and will then propagate to all the panels if grouping is used:

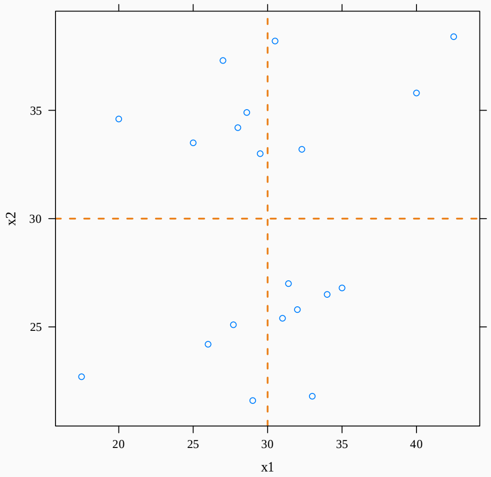
```
> xyplot(x2 ~ x1, jacoby, panel = function(x, y, ...) {
+   panel.xyplot(x, y, ...)
+   panel.abline(h = mean(y), v=mean(x), lty = 2,
+     lwd=2, col="#EB811B")
+ })
```



SIMPLE PLOT METHODS

Syntax similar to that of **ggplot2** was introduced in **latticeExtra**, adding layers to the :

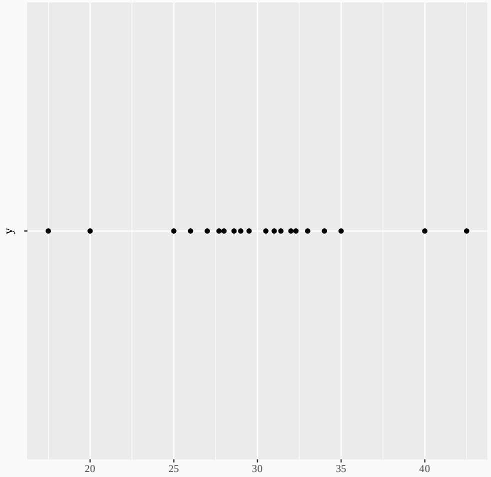
```
> xyplot(x2 ~ x1, jacoby) +
+   latticeExtra::layer(panel.abline(h=mean(y), v=mean(x),
+   lty = 2, lwd=2, col="#EB811B"))
```



SIMPLE PLOT METHODS

The **ggplot2** package builds graphic output layer by layer, like base graphics, using `aes` — aesthetics — to say what is being shown. Starting with the `jacoby` `"data.frame"` object, we plot points with `geom_point` choosing variable `"x1"` by a placeholder `""` empty string on the other axis

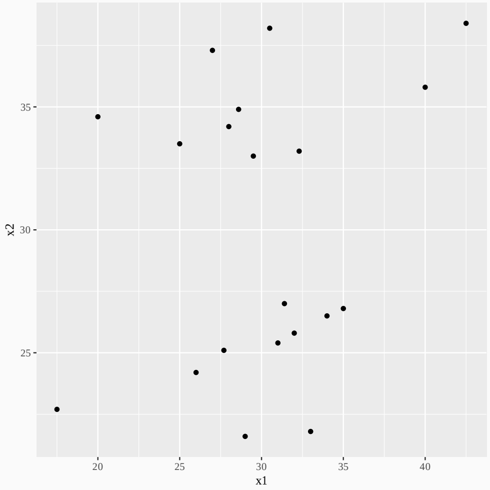
```
> library(ggplot2)
> ggplot(jacoby) + geom_point(aes(x=x1, y="")) + xlab("")
```



SIMPLE PLOT METHODS

Using `aes` on two numeric vectors puts the first vector on the horizontal (x) axis, and the second on the vertical (y) axis, to make a scatterplot

```
> ggplot(jacoby) + geom_point(aes(x1, x2))
```



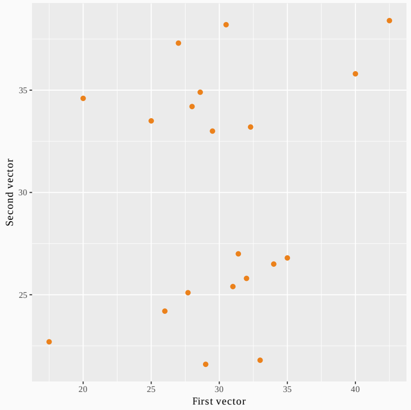
SIMPLE PLOT METHODS

- In these cases, the default plot shows points, which is reasonable here
- The default glyph (symbol) is a small filled circle, the `size` argument lets us change its size, and its colour by `colour`
- The display is annotated with value scales along the axes, and the axes are placed to add a little space outside the bounding box of the observations
- The display includes axis labels, but does not include a title
- In ggplot2, the functions output (summations of) grobs — grid objects, with a `print` method

SIMPLE PLOT METHODS

Here, we'll change the axis labels, and symbol colour and size; these are also vectors, so can vary by observation or for groups of observations; we turn off the legend which is not relevant here

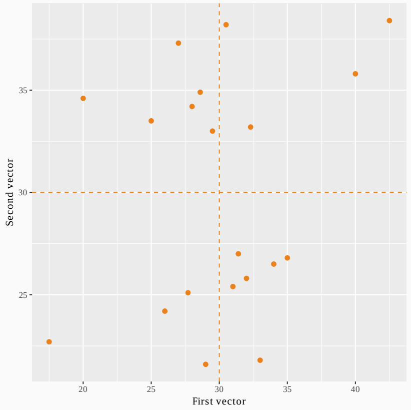
```
> p <- ggplot(jacoby) + geom_point(aes(x1, x2),
+   colour="#EB811B", size=2) + xlab("First vector") +
+   ylab("Second vector") +
+   theme(legend.position = "none")
> p
```



SIMPLE PLOT METHODS

We'll add lines with `geom_hline` and `geom_vline` showing the means of the two vectors:

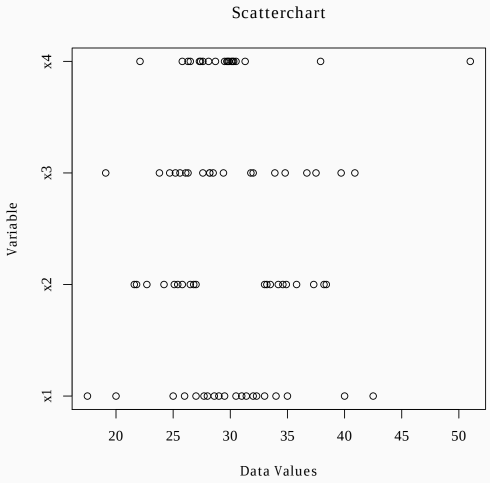
```
> p + geom_hline(yintercept=mean(jacoby$x2),
+   colour="#EB811B", linetype=2) +
+   geom_vline(xintercept=mean(jacoby$x1),
+   colour="#EB811B", linetype=2)
```



JACOBY: SYNTHETIC DATA STRIPCHARTS

We will use stripcharts (p. 6, pp. 30–32) to display all four vectors together on shared axes; now we see why the default symbol is a hollow circle

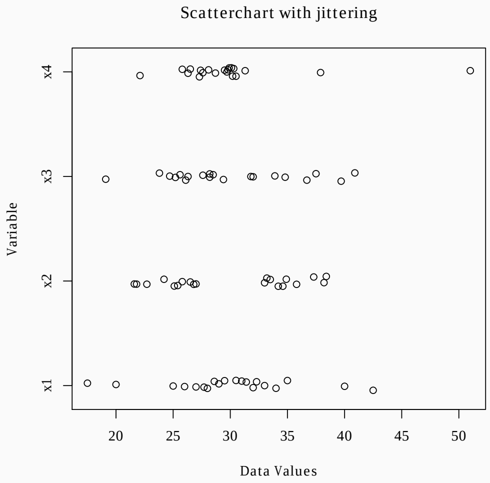
```
> stripchart(jacoby, pch=1, xlab="Data Values",
+   ylab="Variable", main="Scatterchart")
```



JACOBY: SYNTHETIC DATA STRIPCHARTS

We can choose to jitter the symbols in the vertical dimension (adding small random amounts to 0) make the data easier to see

```
> stripchart(jacoby, method=”jitter”,
+   jitter=0.05, pch=1,
+   xlab=”Data Values”,
+   ylab=”Variable”,
+   main=”Scatterchart with jittering”)
```



JACOBY: SYNTHETIC DATA STRIPPLOTS

Lattice stripplots do the same as stripcharts, using a formula interface rather than just an object to choose the correct method. To get started, we need to `stack` the data in “long” form. It is also possible to use `reshape2::melt`.

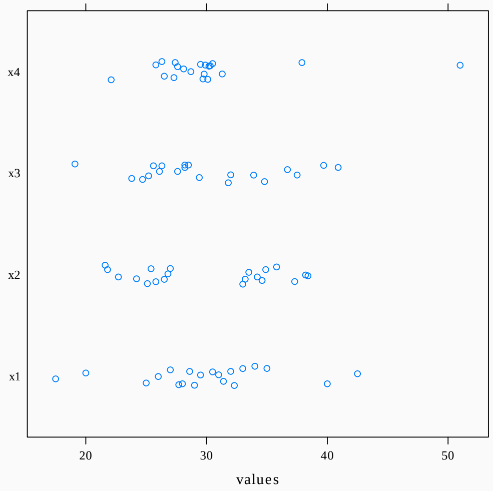
```
> jacobyS <- stack(jacoby)
> str(jacobyS, width=45, strict.width="cut")
```

```
## 'data.frame':    80 obs. of  2 variables:
##  $ values: num  32.3 28 31.4 29.5 40 20 26 ..
##  $ ind   : Factor w/ 4 levels "x1","x2","x"..
```

JACOBY: SYNTHETIC DATA STRIPPLOTS

The formula says that `ind` (y-axis) depends on `values` (x-axis)

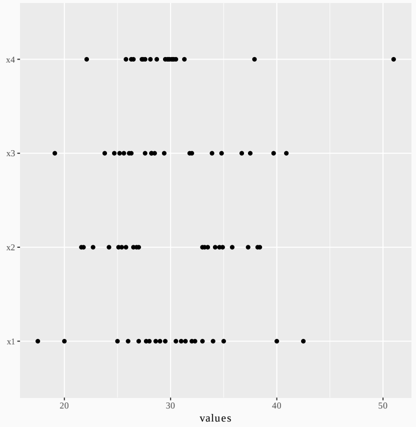
```
> stripplot(ind ~ values, data=jacobyS, jitter.data=TRUE)
```



JACOBY: SYNTHETIC DATA STRIPCHARTS

We will use stripcharts (p. 6, pp. 30–32) to display all four vectors together on shared axes

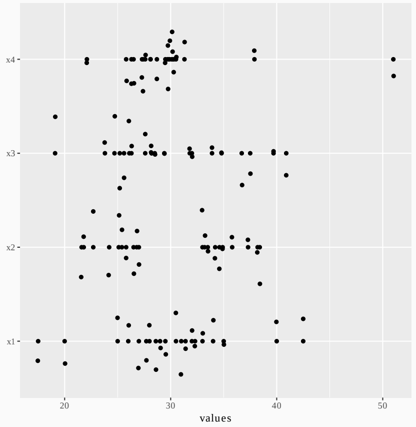
```
> #library(reshape2)
> #jacobyS <- melt(jacoby)
> p <- ggplot(jacobyS, aes(values, ind))
> p + geom_point() + ylab("")
```



JACOBY: SYNTHETIC DATA STRIPCHARTS

We can choose to jitter the symbols in the vertical dimension (adding small random amounts to 0, setting the RNG seed for reproducibility) to make the data easier to see

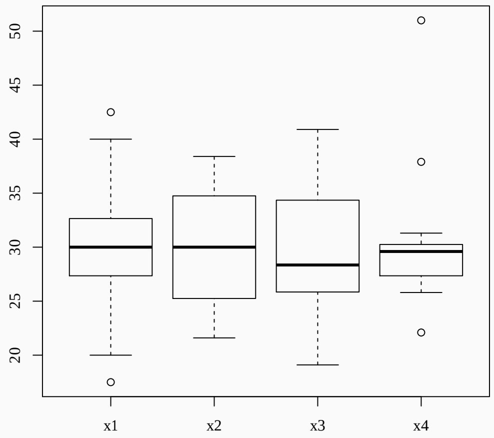
```
> set.seed(1)
> p + geom_point() + ylab("") +
+   geom_jitter(position=position_jitter(0.1))
```



JACOBY: SYNTHETIC DATA BOXPLOTS

The boxplot (pp. 38–43) display places all the vectors on the same axis, here the horizontal axis, places thicker lines at the medians, and boxes representing the interquartile ranges

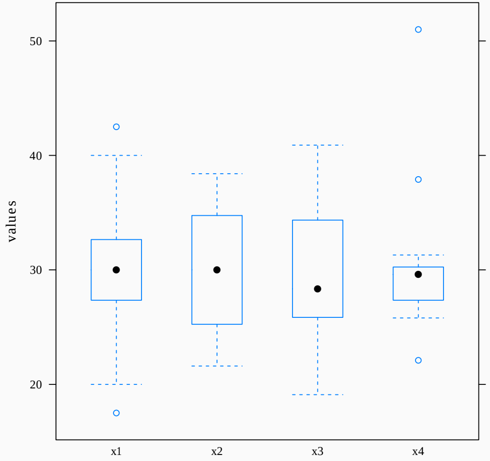
```
> boxplot(jacoby)
```



JACOBY: SYNTHETIC DATA BWPLOTS

The lattice version of boxplot is **`bwplot`** — box and whiskers plot; again we stay with the counter-intuitive horizontal display

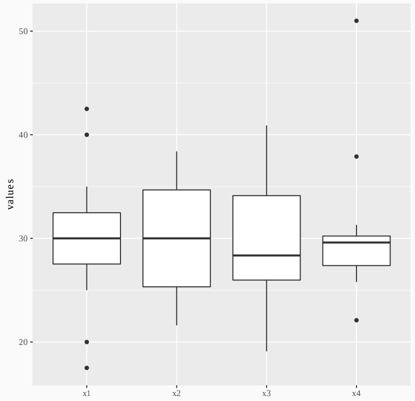
```
> bwplot(values ~ ind, data=jacobyS)
```



JACOBY: SYNTHETIC DATA BOXPLOTS

The boxplot (pp. 38–43) display places all the vectors on the same axis, here the horizontal axis, places thicker lines at the medians, and boxes representing the interquartile ranges

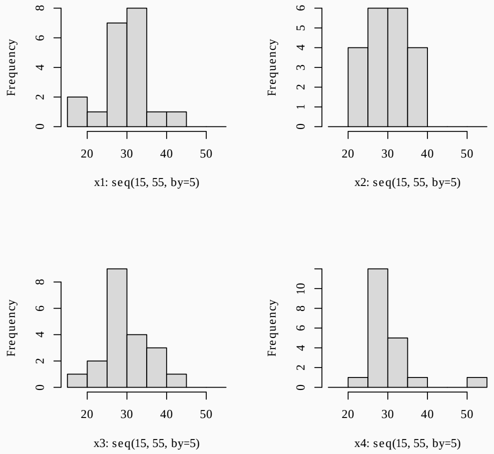
```
> p <- ggplot(jacobyS, aes(ind, values))
> p + geom_boxplot() + xlab("")
```



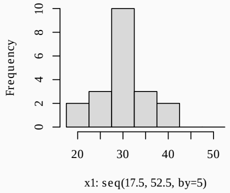
JACOBY: SYNTHETIC DATA HISTOGRAMS

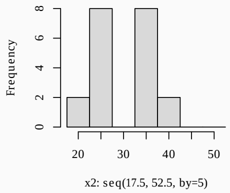
Histograms (pp. 13–17) are very frequently used to examine data, and are directly comparable to thematic maps — both need chosen break points, which have a start point, an end point, and intermediate points which may be evenly spaced, defining bins or intervals

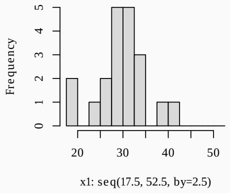
```
> oldpar <- par(no.readonly=TRUE)
> par(mfrow=c(2,2))
> brks <- seq(15,55,by=5)
> for (i in 1:4) {
+  hist(jacoby[,i], breaks=brks, col="grey85",
+  xlab=paste("x", i, ": seq(15, 55, by=5)", sep=""),
+  freq=TRUE, main="")
+ }
> par(oldpar)
```

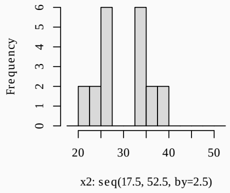


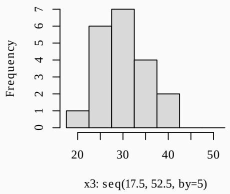
JACOBY: SYNTHETIC DATA HISTOGRAMS

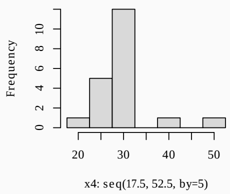


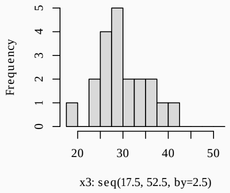


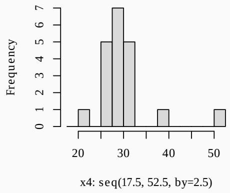








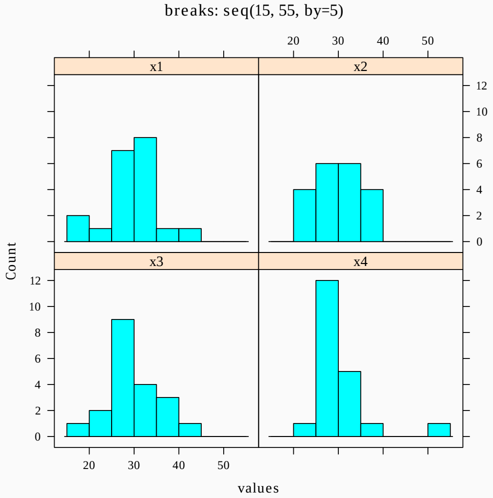




JACOBY: SYNTHETIC DATA LATTICE HISTOGRAMS

The lattice `histogram` syntax is perhaps simpler, with default starting points and bin widths driven by the data

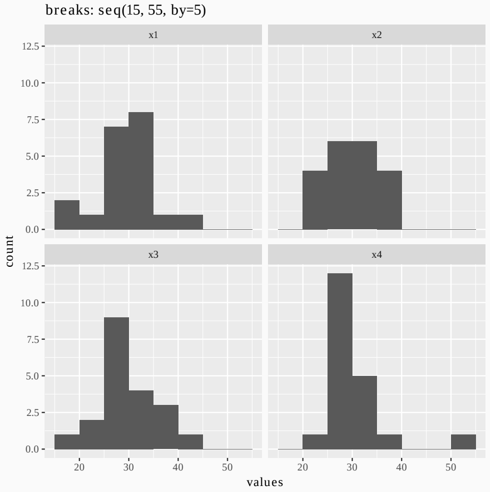
```
> histogram(~ values | ind, data=jacobyS,
+   breaks=seq(15,55,by=5), type="count",
+   index.cond=list(c(3,4,1,2)))
```



JACOBY: SYNTHETIC DATA GGPLOT HISTOGRAMS

Histograms (pp. 13–17) are very frequently used to examine data, and are directly comparable to thematic maps — both need chosen break points, which have a start point, an end point, and intermediate points which may be evenly spaced, defining bins or intervals; we use `facet_wrap()` to place the plots

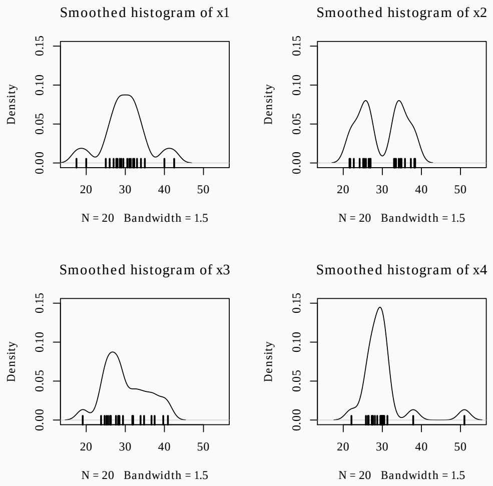
```
> ggplot(jacobyS, aes(x=values)) +
+   geom_histogram(breaks=seq(15, 55, by=5)) +
+   facet_wrap(~ ind, ncol=2)
```



JACOBY: SYNTHETIC DATA DENSITY PLOTS

When we realise that histograms can be constructed by choice of break points to create an impression that may (or may not) mislead, we can look at smoothed histograms (density plots (pp. 18–30)) as an alternative. Here we use a fixed bandwidth

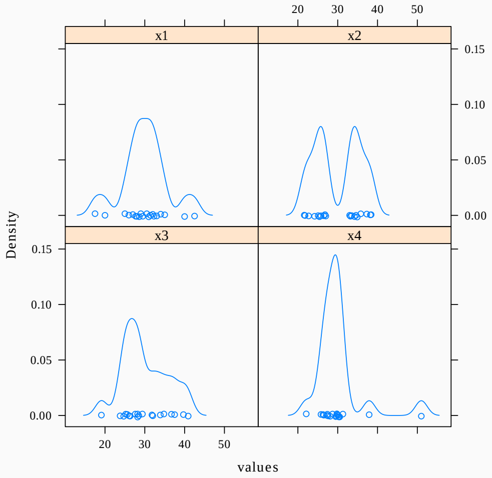
```
> oldpar <- par(no.readonly=TRUE)
> par(mfrow=c(2,2))
> for (i in 1:4) {
+   plot(density(jacoby[,i], bw=1.5), main="",
+     xlim=c(15,55), ylim=c(0, 0.15))
+   rug(jacoby[,i], ticksize=0.07, lwd=2)
+   title(main=paste("Smoothed histogram of x",
+     i, sep=""))
+ }
> par(oldpar)
```



JACOBY: SYNTHETIC DATA DENSITY PLOTS WITH LATTICE

The **lattice** approach again uses the formula interface, and a rug by default:

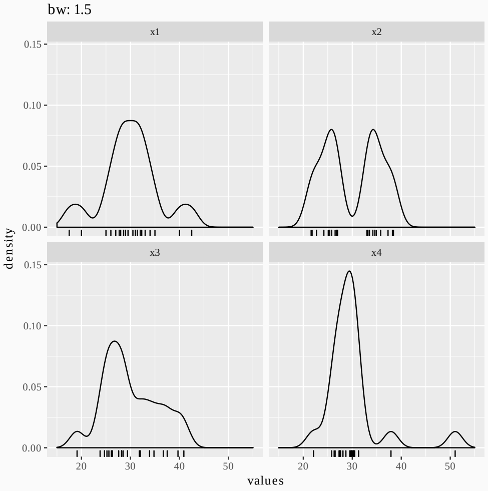
```
> densityplot(~ values | ind, data=jacobyS, bw=1.5,
+   index.cond=list(c(3,4,1,2)))
```



JACOBY: SYNTHETIC DATA DENSITY PLOTS

By default, ggplot facets fix the scales, but it isn't clear whether the bandwidth is fixed, and it does not seem to be reported

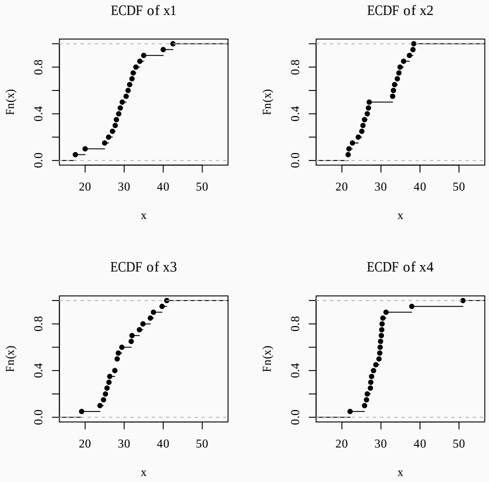
```
> ggplot(jacobyS, aes(x=values)) +
+   geom_density(bw=1.5) + geom_rug() +
+   facet_wrap(~ ind, ncol=2) +
+   xlim(c(15, 55))
```



JACOBY: EMPIRICAL CUMULATIVE DISTRIBUTION FUNCTION

Empirical cumulative distribution functions need no tuning arguments:

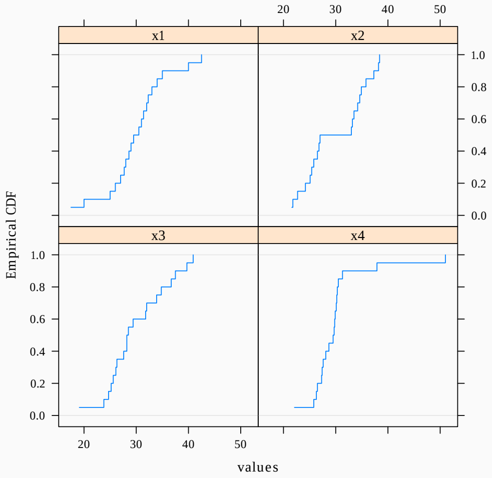
```
> oldpar <- par(no.readonly=TRUE)
> par(mfrow=c(2,2))
> for (i in 1:4) {
+   plot(ecdf(jacoby[,i]), main="",
+     xlim=c(15,55))
+   title(main=paste("ECDF of x",
+     i, sep=""))
+ }
> par(oldpar)
```



JACOBY: EMPIRICAL CUMULATIVE DISTRIBUTION FUNCTION

We need **latticeExtra** to present the ECDF:

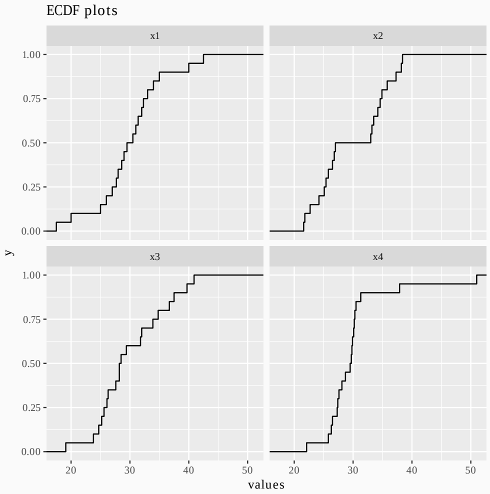
```
> library(latticeExtra)
> ecdfplot(~ values | ind, data=jacobyS,
+   index.cond=list(c(3,4,1,2)))
> detach(package:latticeExtra)
```



JACOBY: EMPIRICAL CUMULATIVE DISTRIBUTION FUNCTION

In ggplot, the `stat_ecdf()` operator is used

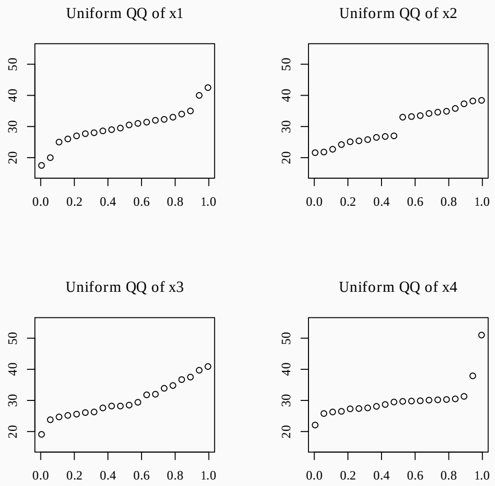
```
> ggplot(jacobyS, aes(x=values)) +
+   stat_ecdf() +
+   facet_wrap(~ ind, ncol=2)
```



JACOBY: QUANTILE-QUANTILE PLOTS

Quantile-quantile plots require the choice of a theoretical distribution, and as Deepayan Sarkar says, the ECDF plot uses a uniform theoretical distribution, so is a Uniform QQ plot in a different orientation:

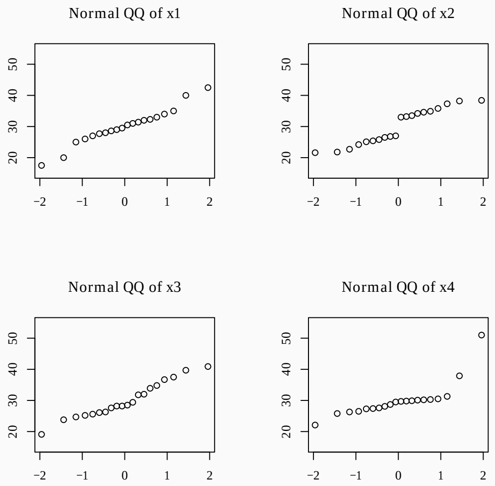
```
> oldpar <- par(no.readonly=TRUE)
> par(mfrow=c(2,2))
> x <- qunif(ppoints(100))
> for (i in 1:4) {
+   qqplot(x=x, y=jacoby[,i])
+   title(main=paste("Uniform QQ of x",
+     i, sep=""))
+ }
> par(oldpar)
```



JACOBY: QUANTILE-QUANTILE PLOTS

The **qqnorm( )** function gives a Normal QQ plot directly:

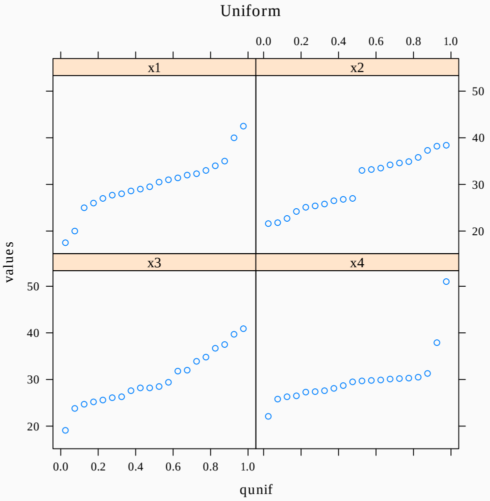
```
> oldpar <- par(no.readonly=TRUE)
> par(mfrow=c(2,2))
> for (i in 1:4) {
+   qqnorm(y=jacoby[,i], xlab="", ylab="",
+     ylim=c(15, 55), main="")
+   title(main=paste("Normal QQ of x",
+     i, sep=""))
+ }
> par(oldpar)
```



JACOBY: QUANTILE-QUANTILE PLOTS

In **lattice**, `qqmath()` takes a `distribution` argument:

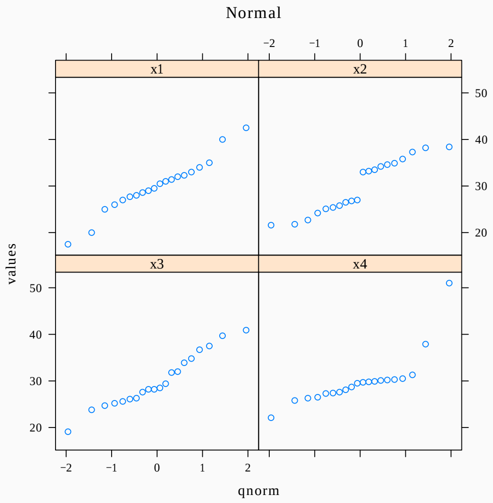
```
> qqmath(~ values | ind, data=jacobyS,
+   distribution=qunif,
+   index.cond=list(c(3,4,1,2)))
```



JACOBY: QUANTILE-QUANTILE PLOTS

so this reproduces `qqnorm`:

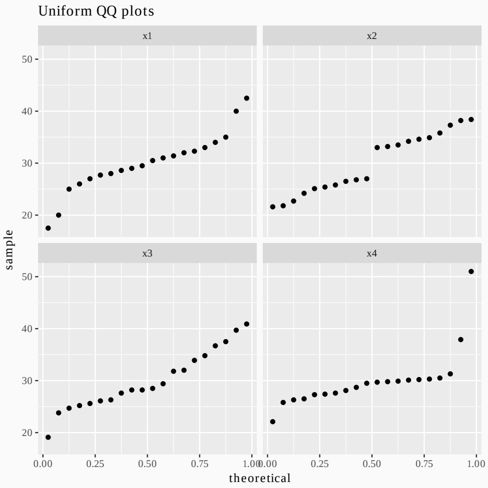
```
> qqmath(~ values | ind, data=jacobyS,
+   distribution=qnorm,
+   index.cond=list(c(3,4,1,2)))
```



JACOBY: QUANTILE-QUANTILE PLOTS

In ggplot, the `stat_qq()` operator is used with a specified distribution function, here Uniform:

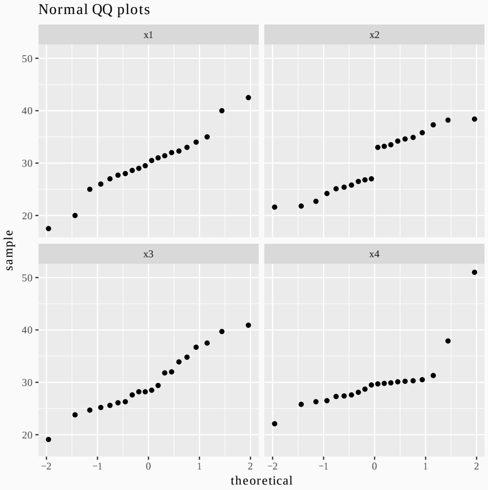
```
> ggplot(jacobyS, aes(sample=values)) +
+   stat_qq(distribution=qunif) +
+   facet_wrap(~ ind, ncol=2)
```



JACOBY: QUANTILE-QUANTILE PLOTS

and for Normal QQ plots

```
> ggplot(jacobyS, aes(sample=values)) +
+   stat_qq(distribution=qnorm) +
+   facet_wrap(~ ind, ncol=2)
```



Ihaka Lecture 1, 2018

Myth busting and apophenia in data visualisation: Is what you see really there?

PISA DATA SUBSET

- In the talk, the PISA 2015 data set is mentioned
- An OECD technical report describes in more detail how the plausible values from fitted item response models are generated, in 2015 10 PVs for each participant, 5 PVs before
- In the talk, Di Cook uses the first plausible value; we can check what she does in the code blocks in her slides

IMPORTING THE SPSS DATA

The data provided for the over 500,000 individual students include over 900 variables, but we'll look at the score plausible values (PV), taking means and standard deviations of the 10 PVs given for math, reading and science for each student. In the talk, only PV1 was used. We reread the data to get three-letter country codes.

```
> QQQ <- read.spss("CY6_MS_CMB_STU_QQQ.sav", to.data.frame=TRUE)
> QQQ1 <- QQQ[,c(1,2,4,29,30,35,64,810:839,920)]
> QQQ1$math_mean <- apply(as.matrix(QQQ1[,8:17]), 1, mean)
> QQQ1$math_sd <- apply(as.matrix(QQQ1[,8:17]), 1, sd)
> QQQ1$read_mean <- apply(as.matrix(QQQ1[,18:27]), 1, mean)
> QQQ1$read_sd <- apply(as.matrix(QQQ1[,18:27]), 1, sd)
> QQQ1$sci_mean <- apply(as.matrix(QQQ1[,28:37]), 1, mean)
> QQQ1$sci_sd <- apply(as.matrix(QQQ1[,28:37]), 1, sd)
> QQQ <- read.spss("CY6_MS_CMB_STU_QQQ.sav", to.data.frame=TRUE, use.value.labels=FALSE)
> QQQ1$CNT_vl <- QQQ1$CNT
> QQQ1$CNT <- QQQ$CNT
> saveRDS(QQQ1, file="dicook/pisa_raw_subset.rds")
```

SUBSETTING THE SCORES

Next we change some country codes to use ISO 3-letter alphabetical codes, and drop three entities as in the original script, and re-save this observation subset.

```
> QQQ1 <- readRDS("dicook/pisa_raw_subset.rds")
> recodes <- c("'QES'='ESP'", "'QCH'='CHN'", "'QAR'='ARG'", "'TAP'='TWN'")
> for (str in recodes) QQQ1$CNT <- car::recode(QQQ1$CNT, str)
> QQQ2 <- droplevels(QQQ1[!(as.character(QQQ1$CNT) %in% c("QUC", "QUD", "QUE")),])
> library(ISOcodes)
> data("ISO_3166_1")
> scores <- merge(QQQ2, ISO_3166_1[,c("Alpha_3", "Name")], by.x="CNT", by.y="Alpha_3", all.x=TRUE)
> scores$Name[scores$CNT == "KSV"] <- "Kosovo"
> saveRDS(scores, file="dicook/pisa_subset.rds")
```

ADDING COUNTRY CODES AND NAMES

Some of this repeats the steps above, and could be rationalsed. It is intended to yield a helper object listing countries and country codes with numbers of students, hence the use of `table()` and `aggregate()`.

```
> QQQ1 <- readRDS("dicook/pisa_raw_subset.rds")
> recodes <- c("'QES'='ESP'", "'QCH'='CHN'", "'QAR'='ARG'", "'TAP'='TWN'")
> CNT_count <- as.data.frame(table(QQQ1$CNT))
> names(CNT_count) <- c("CNT", "n")
> for(str in recodes) CNT_count$CNT <- car::recode(CNT_count$CNT, str)
> CNT_count1 <- aggregate(CNT_count$n, list(CNT_count$CNT), sum)
> CNT_count2 <- droplevels(CNT_count1[!(as.character(CNT_count1$Group.1) %in% c("QUC", "QUD", "QUE")),])
> names(CNT_count2) <- c("CNT", "n")
> library(ISOcodes)
> data("ISO_3166_1")
> countries <- merge(CNT_count2, ISO_3166_1, by.x="CNT", by.y="Alpha_3", all.x=TRUE)
> countries$Name[countries$CNT == "KSV"] <- "Kosovo"
> saveRDS(countries, file="dicook/countries.rds")
```

SPLITTING INTO GROUPS

Once these preparations are complete, we have about 500,000 observations on 10 PVs for each of three school subjects by binary gender and country, and use `split()` to make a list of two genders times 69 countries of student data frames.

```
> QQQ1 <- readRDS(”dicook/pisa_subset.rds”)
> countries <- readRDS(”dicook/countries.rds”)
> a0 <- split(QQQ1, list(QQQ1$CNT, QQQ1$ST004D01T))
```

AGGREGATING WEIGHTED MATH MEANS

Using `sapply()` on this list gives us one weighted mean of the means per student of maths PVs. We weight using normalized "senate" weights to permit comparisons between countries: "The senate weight makes the population of each country to be 5,000 to ensure an equal contribution by each of the countries in the analysis."

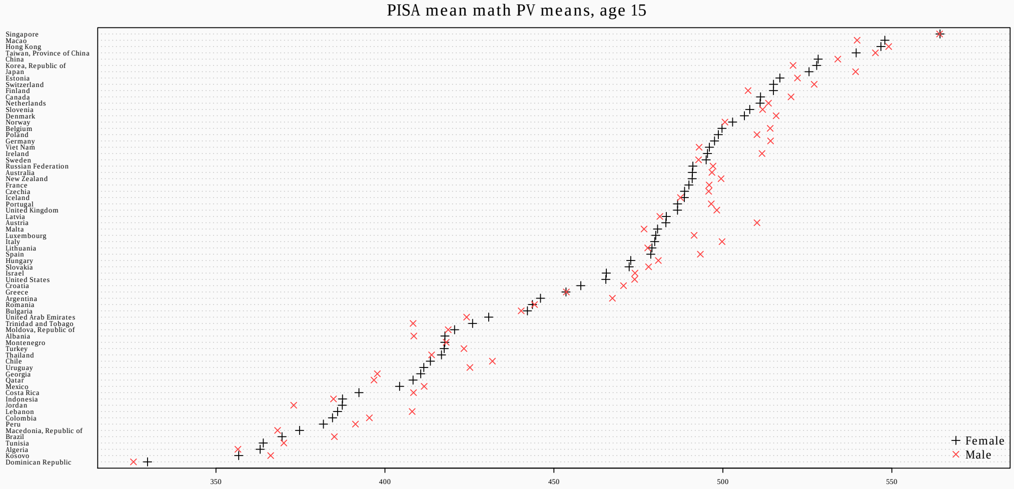
```
> math_mean <- sapply(a0, function(x) weighted.mean(x$math_mean, w=x$SENWT))
> n2 <- length(math_mean)/2
> country <- sapply(strsplit(names(math_mean), "\\."), "[", 1)[1:n2]
> co <- match(country, countries$CNT)
> nms <- countries$Name[co]
> gender <- factor(sapply(strsplit(names(math_mean), "\\."),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

MATH MEANS CLEVELAND DOTCHART

The workhorse is a Cleveland dotchart, which shows one point per math mean per gender and country. Standard dotcharts would show either one grouping category, or two separate dotcharts, one per gender, but here we wish to compare means of PV means:

```
> dotchart(math_mean[o], label=nms[o], cex=0.5, pt.cex=0.9, pch=3, xlim=c(325, 575))
> points(math_mean[n2+o], 1:n2, cex=0.9, pch=4, col="brown1")
> legend("bottomright", legend=levels(gender), col=c("black", "brown1"), pch=3:4, pt.cex=0.9, bty="n", cex=0.8)
> title("PISA mean math PV means, age 15")
```

MATH MEANS CLEVELAND DOTCHART



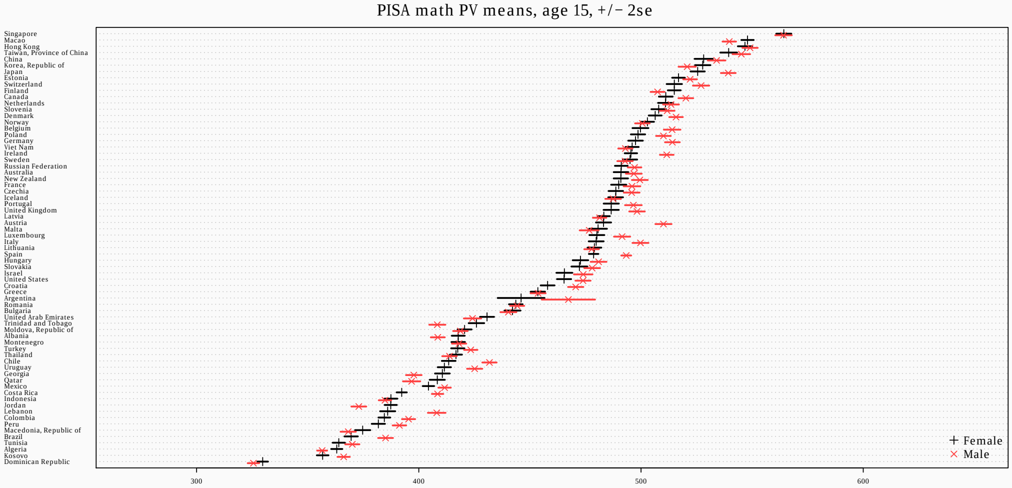
BUT WHAT ABOUT A +/- 2 STANDARD ERROR CATERPILLAR PLOT?

However, in much applied work, it is useful to see whether apparent differences are only apparent. The caterpillar plot is useful for this, it is like a dotplot, but adds lines usually showing a multiple of standard errors (here +/- 2). We use a weighted standard deviation and sums of weights to ensure that the means and standard errors are coherent. We can insert the lines using a `for()` loop:

```
> 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

> for (i in 1:n2) segments(x0=math_mean[o][i]-(2*math_s2[o][i]), y0=i,
+   x1=math_mean[o][i]+2*(math_se[o][i]), lty=2)
```

MATH MEANS CLEVELAND DOT/CATERPILLAR CHART



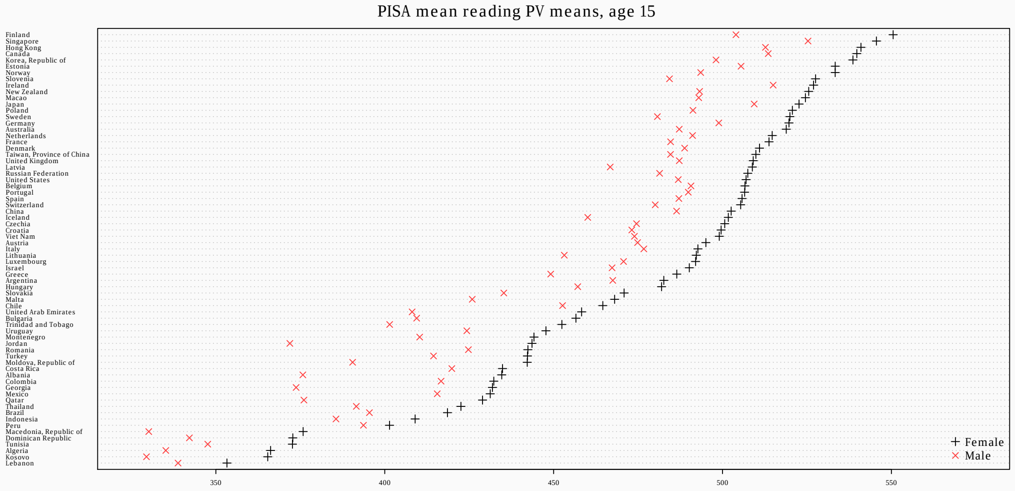
AGGREGATING WEIGHTED READING AND SCIENCE MEANS

We repeat the simple comparative dotplot for the remaining school subjects:

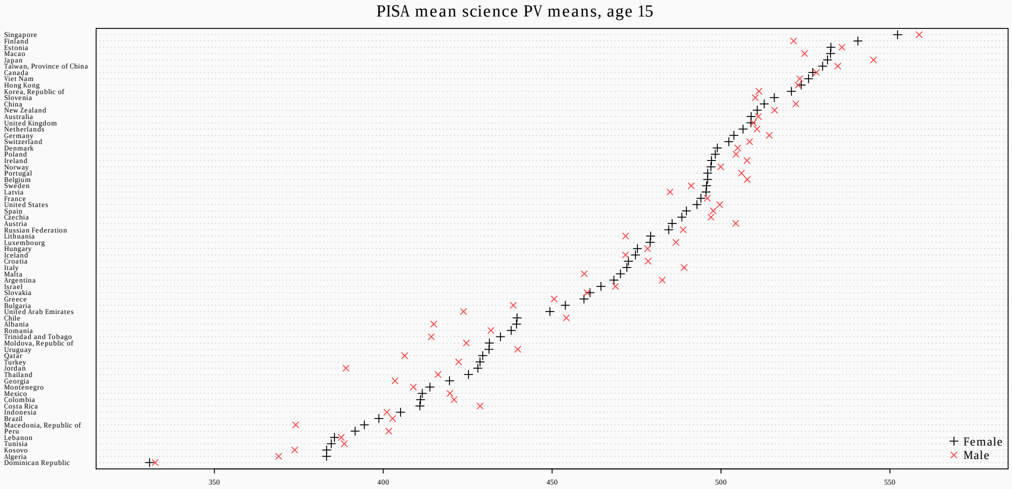
```
> read_mean <- sapply(a0, function(x) weighted.mean(x$read_mean, w=x$SENWT))
> ro <- order(read_mean[1:n2])

> sci_mean <- sapply(a0, function(x) weighted.mean(x$sci_mean, w=x$SENWT))
> so <- order(sci_mean[1:n2])
```

READING MEANS CLEVELAND DOTCHART



SCIENCE MEANS CLEVELAND DOTCHART



MAPPING GAPS FEMALE-MALE

As in the talk, we can make maps of the apparent gender gaps between the means of mean plausible values, first calculating the gaps and adding country identifiers:

```
> math_gap <- math_mean[1:n2] - math_mean[n2+(1:n2)]
> read_gap <- read_mean[1:n2] - read_mean[n2+(1:n2)]
> sci_gap <- sci_mean[1:n2] - sci_mean[n2+(1:n2)]
> gaps <- data.frame(math_gap, read_gap, sci_gap, iso_a3=country,
+   iso_a2=countries$Alpha_2[co])
```

COUNTRY BOUNDARIES

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
```

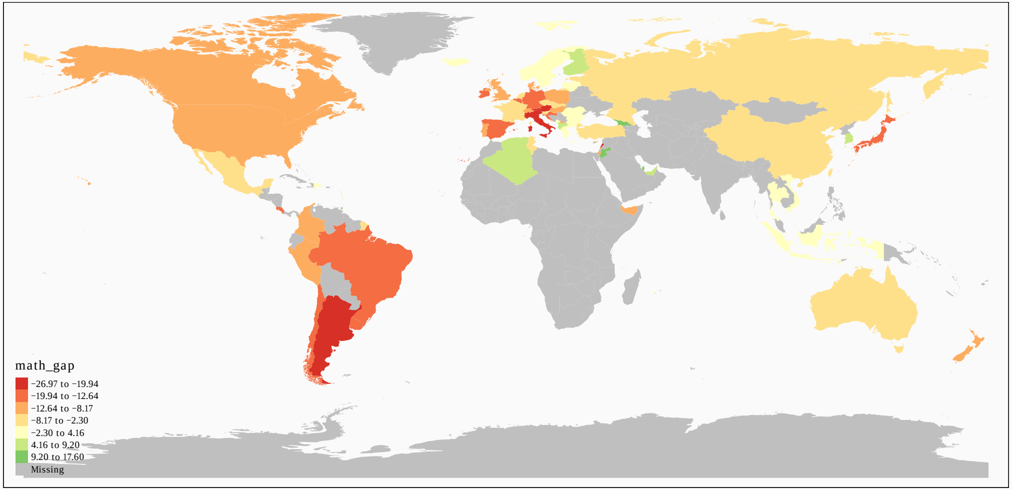
```
> countries50 <- st_as_sf(countries50)
```

USING TMAP FOR MAPS

We can `merge()` the gaps data with the country boundaries, and the **tmap** package to create the map. Tomorrow we'll look at binning and colours. Using `ttm()`, we can toggle to an interactive format, using `last_map()` to recall the last generated map.

```
> world_gaps <- merge(countries50, gaps, by="iso_a2", all.x=TRUE)
> library(tmap)
> tm_shape(world_gaps) + tm_fill("math_gap", palette="div", n=7, style="jenks")
> #ttm()
> #last_map()
```

MATHS GAPS


math_gap
−26.97 to −19.94
−19.94 to −12.64
−12.64 to −8.17
−8.17 to −2.30
−2.30 to 4.16
4.16 to 9.20
9.20 to 17.60
Missing

OTHER POINTS

- In the talk, boxplots are used to explore the variability between students by gender and country; here we tried a caterpillar plot. The PV means also flatten variability - would medians be more sensible?
- Could aspects of the PISA data set be used in thinking about projects?
- Other data sets are made available by Di Cook, and the **nullabor** package seems interesting, and maybe could also work (well) for projects, maybe also varying graphical representation?
- We'll see some other tools on Tuesday and Wednesday

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] tmap_1.11-1            sf_0.6-1               rnaturalearthdata_0.1.0
## [4] rnaturalearth_0.1.0    matrixStats_0.53.1     RColorBrewer_1.1-2
## [7] ggplot2_2.2.1          lattice_0.20-35        extrafont_0.17
##
## loaded via a namespace (and not attached):
```