**Introducing adavis**

adamap, a Shiny app that maps Americans for Democratic Action voting scores (the so-called *Liberal Quotient*) between 1947-2015. It was built with highcharter, and hence it was nicely interactive but quite slow. I wanted to switch to another package since, and when I eventually ran into statebins, I knew what had to be done.

I was certain that statebins would definitely add some oomph to the design, but because it’s so easy to implement, I had some spare time to do other things. As it is often the case, one thing led to the other, and I came to the conclusion that the revamped app should feature one plot from every major graphics package. Of course, a strict implementation of that statement would be quite difficult, so I downgraded the challenge to just four plots using a different package each time. I ended with statebins for state-level mapping, plotly for plotting changes from the previous year, ggExtra for head-to-head state comparisons, and \*drum roll\* base R for graphing a single politican’s voting record over time. It turned out to be fun.

As I have already explained the data setup in my previous post linked above, I will skip to the visualisations. I won’t be including the actual code for the Shiny app, which includes reactive elements throughout.. Also, none of the codes are evaluated here (as I translate them from the app), so they will not work if you just plug them in. Hence, there are primarily for motivation rather than replication.

**Slicker US with Statebins**

There are many good things about statebins. First, you get identical-sized states so you don’t get biased by the variation in their size. It’s cleaner by definition, not featuring idiosyncratic shapes that are found in nature. Also, it plays really nice with viridis, which is important (maybe). In addition, you can define light and dark labels for the state abbreviations, ensuring they will not blend into the fill colour. statebins can be called as a function (as I did), or applied later to a ggplot object. The only thing that did not work for me was the ggplot2\_scale\_function argument; R kept saying no such function is defined (I’m using the dev version from GitHub) so I ended up passing the fill colours separately. It gives a warning about overriding the existing fill, but works otherwise. If you download the dataset and want to visualise how House Democrats in 1962 voted, something along the lines of:

#Not evaluated

library(ggplot2)

library(viridis)

library(statebins)

library(hrbrthemes)

theme\_set(theme\_ipsum\_rc())

#assumes data = data, year as "Year", state names in "State", voting scores in "ADA" etc.

us <- statebins(data[data$Year == 1962 & data$Chamber == 1 & data$Party == "Democrat", ],

state\_col = "State", value\_col = "ADA",

round = TRUE, font\_size = 7, state\_border\_col = "#232d33",

dark\_label = "white",

light\_label = "black",

state\_border\_size = .8, radius = grid::unit(10, "pt")) +

labs(title = "") +

theme(axis.text.x = element\_blank(), axis.text.y = element\_blank(), axis.ticks = element\_blank(),

panel.grid.minor = element\_blank(), panel.grid.major = element\_blank(),

panel.background = element\_rect(fill = "white", linetype = "blank"),

legend.position = c(.075, .85), legend.direction = "horizontal",

legend.text = element\_text(colour = "#232d33", size = 14),

legend.title = element\_text(colour = "#232d33", size = 18),

legend.key.height = grid::unit(.01, "snpc"),

legend.key.width = grid::unit(.05, "snpc"),

plot.margin = margin(-1, 0, 0, 0, "cm"))

us + scale\_fill\_viridis(direction = -1,

breaks = c(seq(-25, 100, 25)),

labels = c("No Rep", paste(seq(0, 100, 25), "% ")),

guide = guide\_legend(title = "", title.position = "top",

keywidth = 2, keyheight = 2, ncol = 1))

should do the trick. Now, I am totally cheating because the image is from the live app and the above code is not evaluated. However, it should give you an idea, mainly most of the clutter is about the layout rather than the content. we get magically created bins (and other shapes, for the ambitious) not just for US states but for everything? Cheers.

One trick regarding the data; the original data only have scores for selected representatives (naturally). Meaning, in any given year, there will be several states (approx. 8-10 per party) with no Democrat or Republican reps. As these are row-wise missing instead of NA, if you plot them as they are, those states will not show in the plot. If only there was a tidyverse function that would solve common data problems like this…

library(tidyverse)

#Add rows for missing states in party-year

#Use built-in states data

states <- data.frame(state.name, stringsAsFactors = FALSE)

states$state.no <- 1:50

dataset <- merge(dataset, states, by.x = "State", by.y = "state.name")

#I couldn't get this to work with strings so matched them by state.no

dataset <- dataset %>% tidyr::complete(state.no = full\_seq(state.no, period = 1), Year, Chamber, Party,

fill = list(ADA = -25, aADA = -25)) #Arbitrary low score instead of NA

dataset$State <- ifelse(is.na(dataset$State), states[dataset$state.no, 1], dataset$State)

**Interactive Charts with Plotly**

Moving on to challenge number #2, I wanted to keep to the same filter (Year > Chamber > Party), but with the amount of change from last year plotted instead. I haven’t used plotly much before so I learned on-the-go.

The main reason for going with plotly was its build-in interactivity. I wanted the users to just hover over points and see a block of text describing the shift from the previous year. This turned out to be easy, just with a somewhat ugly paste. One curious thing was the alpha functionality, which is governed with toRGB("colour", "alpha"), but called opacity in plotly. In the app, the default is showing Senate Republicans in 1990 (i.e. difference from 1989):

library(plotly)

#Store common args

ax <- list(

showline = FALSE,

showticklabels = TRUE,

showgrid = FALSE)

#Mandatory data should be subsetted before comment

plot\_ly(data, x = ~Score, key = data[, 1],

y = ~Change, mode = "markers", type = "scatter", hoverinfo = "text",

hoverlabel = list(font = list(family = "Roboto Condensed", size = 14)),

#Add custom hover text

text = ~paste(data$Chamber, data$Party, "from",

State, "voted\n", paste0(abs(round(Change, 2)), "% more"), Label, "in", data$Year),

color = ~Change, colors = viridis(direction = -1, n = 12),

marker = list(size = 30, opacity = .7)) %>%

layout(dragmode = "select", showlegend = FALSE,

xaxis = c(ax, list(title = "Selected Year Voting Score", zeroline = FALSE)),

yaxis = c(ax, list(title = "Change from Last Year", zeroline = TRUE, zerolinecolor = toRGB("black", .05))),

font = list(family = "Roboto Condensed")) %>%

config(displayModeBar = FALSE)

**Last but not least…Base R**

When I saw this brilliant Tufte, the plot I wanted to replicate the most was the very first one. The one that was done in base R. Some might even argue I added a representative look-up tab to the app just for trying this out. Hmm. Like plotly, I was out of practice with base R graphics, so I mimicked the original code as much as I could. One thing I wanted to convey with this graph is the consistency of a single politician over their tenure. I didn’t want to show minima and maxima, but just their mean score with some sort of confidence measure. I also learned that you can pass Greek letters with expression(), which is handy. Say, you want to plot the complete voting history of Nancy Pelosi:

#Store descriptives

v1 <- mean(data$Score)

v2 <- sd(data$Score)

v3 <- min(data$Year)

v4 <- max(data$Year)

v5 <- summary(data$Year)

#Base plot, data should be a representative subset

plot(data$Score ~ data$Year, xlab = "", ylab = "", axes = FALSE, family = "Roboto Condensed",

pch = 16, type = "b", lwd = 2)

#Upper sd

abline(h = v1 + v2, lty = 2, col = alpha("black", .2))

#Mean

abline(h = v1, lty = 2)

#Lower sd

abline(h = v1 - v2, lty = 2, col = alpha("black", .2))

#Right axis

axis(1, at = c(v3, v4, (v3 + v4) / 2),

labels = c(v3, v4, round((v3 + v4) / 2, 0)),

tick = FALSE, family = "Roboto Condensed")

#Bottom axis

axis(2, at = c(v1, v1 + v2, v1 - v2), labels = round(c(v1, v1 + v2, v1 - v2), 0),

las = 2, family = "Roboto Condensed", tick = FALSE, lty = 0)

#Left axis

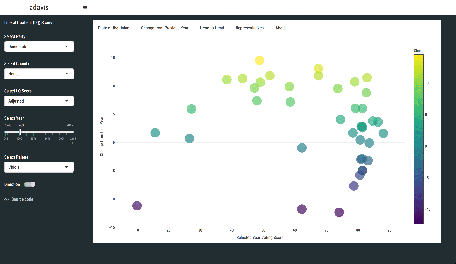
axis(4, at = c(v1, v1 + v2, v1 - v2), lwd = 0, las = 2,

labels = c(expression(mu), expression(sigma), expression(sigma)),

col = alpha("black", .2), family = "Roboto Condensed", ps = 20)

**Viridis Options**

I might be relying on viridis a lot, although I also utilise the RColorBrewer package as well. To be honest, I more or less only like the default palette, the namesake or option = "D", but others might fancy some good old diversity. To this end, I added a drop-down menu for switching viridis palettes, and a button for changing the direction (i.e. whether the palette should start from the lightest or the darkest colour). Both of these options are global, so you can switch any time at any tab. Except for the base R plot; that looks much better in black, *Tufte* style.



Implementation of Tufte in R

===========================

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 65

We start by plotting the most basic graph from page 65 of *The Visual Display of Quantitative Information* - a minimal line plot. This one is important because it illustrates the most elemental principle - that of minimalism with reduced ‘data-ink’. As Tufte explains, the ‘data-ink’ (total ink used to print the graphic) ratio should equal to ‘1 - proportion of graphic that can be erased without loss of data-information’. The primary challenge is therefore to modify the default graphs produced with *R* so that we remove as much of ‘non-data ink’ as possible. As you will soon see, this is done by subtracting and deconstructing existing *R* graphs to get rid of as much ‘non-data ink’ as possible.

*Minimal line plot in base graphics*

Parameter axis = F prevents from drawing all axes elements so they can be easily refined with axis() function. I use minimal and maximal values from the data to draw text() - it usually requires a bit of tweaking to get it right. Font is changed to serif with family.

x <- 1967:1977

y <- c(0.5,1.8,4.6,5.3,5.3,5.7,5.4,5,5.5,6,5)

pdf(width=10, height=6)

plot(y ~ x, axes=F, xlab="", ylab="", pch=16, type="b")

axis(1, at=x, label=x, tick=F, family="serif")

axis(2, at=seq(1,6,1), label=sprintf("$%s", seq(300,400,20)), tick=F, las=2, family="serif")

abline(h=6,lty=2)

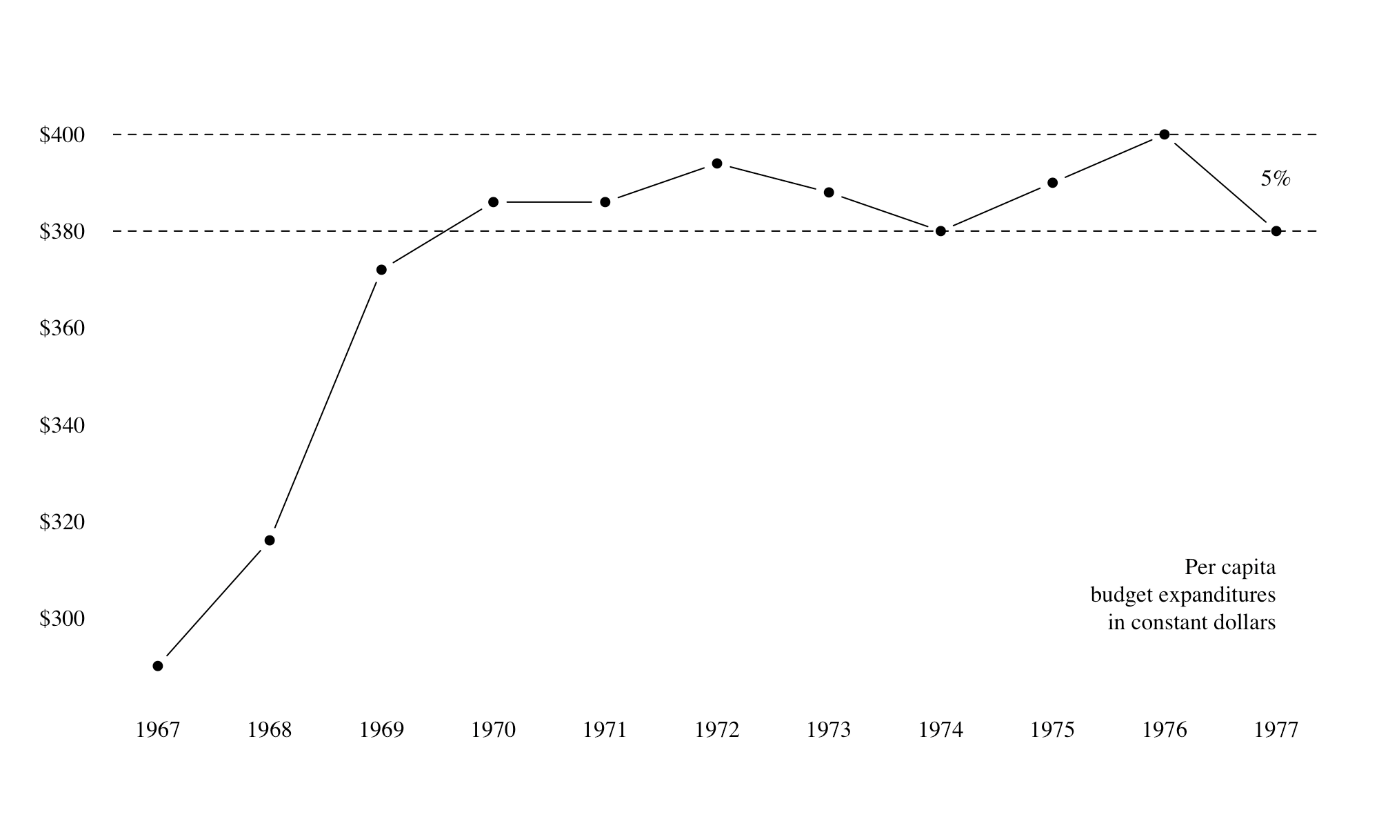
abline(h=5,lty=2)

text(max(x), min(y)\*2.5,"Per capita\nbudget expanditures\nin constant dollars", adj=1,

family="serif")

text(max(x), max(y)/1.08, labels="5%", family="serif")

dev.off()



*Minimal line plot in lattice*

Arguments scales and par.settings have to be used heavily to customise scales and get rid of box. I used [benbarnes axis hack](http://stackoverflow.com/questions/12901396/controlling-axis-ticks-and-axis-lines-separately-on-r-lattice-xyplot) from *Stackoverflow* to draw only axes ticks.

**library**(lattice)

x <- 1967:1977

y <- c(0.5,1.8,4.6,5.3,5.3,5.7,5.4,5,5.5,6,5)

xyplot(y~x, xlab="", ylab="", pch=16, col=1, border = "transparent", type="o",

abline=list(h = c(max(y),max(y)-1), lty = 2),

scales=list(x=list(at=x,labels=x, fontfamily="serif", cex=1),

y=list(at=seq(1,6,1), fontfamily="serif", cex=1,

label=sprintf("$%s",seq(300,400,20)))),

par.settings = list(axis.line = list(col = "transparent"), dot.line=list(lwd=0)),

axis = **function**(side, line.col = "black", **...**) {

**if**(side %in% c("left","bottom")) {axis.default(side = side, line.col = "black", **...**)}})

ltext(current.panel.limits()$xlim[2]/1.1, adj=1, fontfamily="serif",

current.panel.limits()$ylim[1]/1.3, cex=1,

"Per capita\nbudget expandures\nin constant dollars")

ltext(current.panel.limits()$xlim[2]/1.1, adj=1, fontfamily="serif",

current.panel.limits()$ylim[1]/5.5, cex=1, "5%")



*Minimal line plot in ggplot2*

I use excellent package [ggthemes](https://cran.r-project.org/package=ggthemes) by Jeffrey B. Arnold which provides a lot of useful functions for Tufte-like plots - including a dedicated theme\_tufte() function.

**library**(ggplot2)

**library**(ggthemes)

x <- 1967:1977

y <- c(0.5,1.8,4.6,5.3,5.3,5.7,5.4,5,5.5,6,5)

d <- data.frame(x, y)

ggplot(d, aes(x,y)) + geom\_line() + geom\_point(size=3) + theme\_tufte(base\_size = 15) +

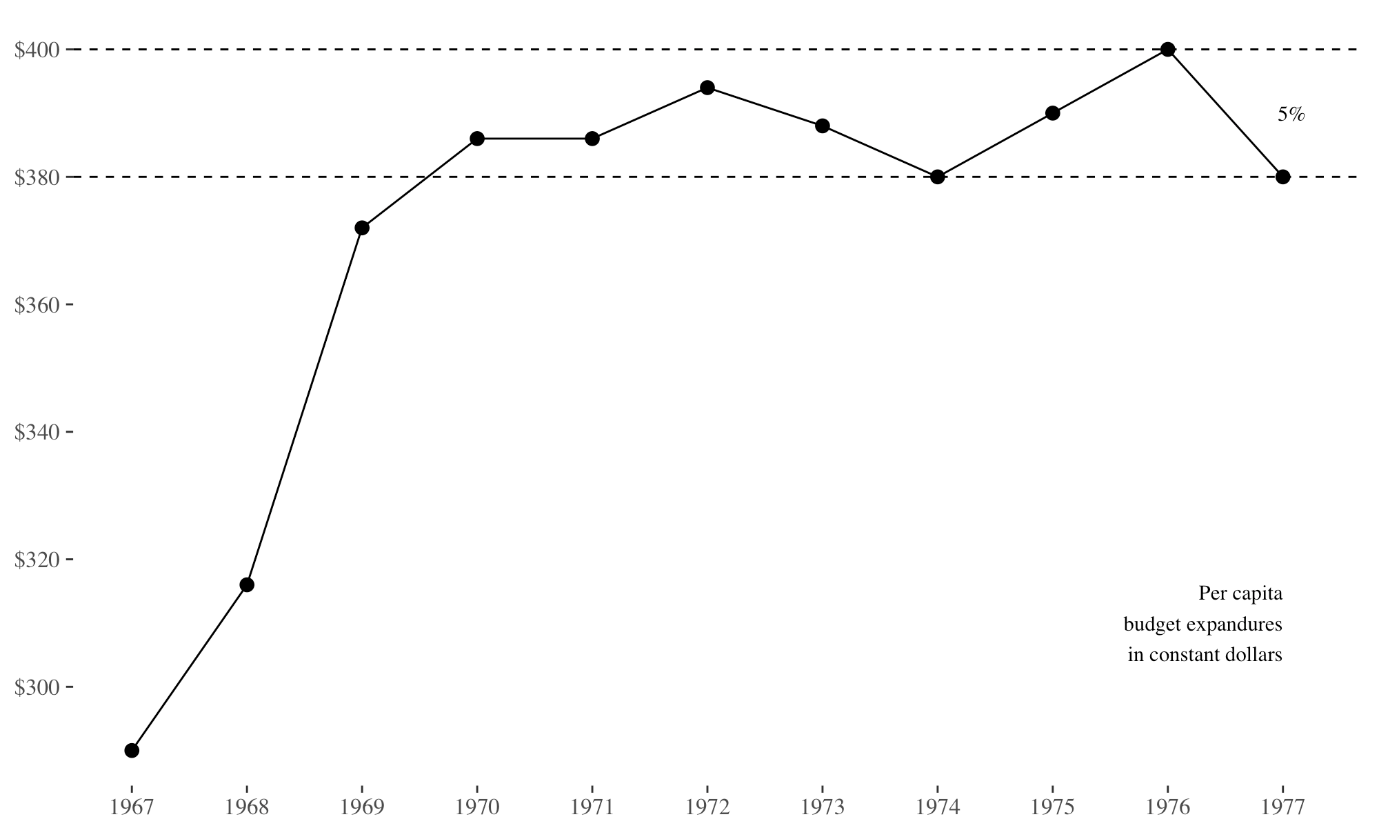
theme(axis.title=element\_blank()) + geom\_hline(yintercept = c(5,6), lty=2) +

scale\_y\_continuous(breaks=seq(1, 6, 1), label=sprintf("$%s",seq(300,400,20))) +

scale\_x\_continuous(breaks=x,label=x) +

annotate("text", x = c(1977,1977.2), y = c(1.5,5.5), adj=1, family="serif",

label = c("Per capita\nbudget expandures\nin constant dollars", "5%"))



*Minimal line plot - interactive plot with highcharter*

A new approach to create a dynamic plots with package [highcharter](http://jkunst.com/highcharter/) - seems its based on Java wrappers and includes dedicated hc\_theme\_tufte().

**library**(highcharter)

x <- 1967:1977

y <- c(290,318,372,385,385,372,386,380,390,400,380)

d <- data.frame(x, y)

highchart() %>%

hc\_chart(type = "scatter") %>%

hc\_subtitle(text = "Per capita budget expanditures in constant dollars") %>%

hc\_yAxis(labels = list(format = "${value}")) %>%

hc\_add\_series(data = d) %>%

hc\_add\_theme(hc\_theme\_tufte())

Per capita budget expanditures in constant dollars19671968196919701971197219731974197519761977$280$300$320$340$360$380$400$420

*Range-frame (or quartile-frame) scatterplot*

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 130-133. This doesn’t really replicate Tufte range frame because its a bit tricky to draw custom axis lines in basic graphics. As a rough starting point I use summary() to display values for minimum, maximum, median, mean and both quartiles on the axes.

*Range frame plot in base graphics*

x <- mtcars$wt

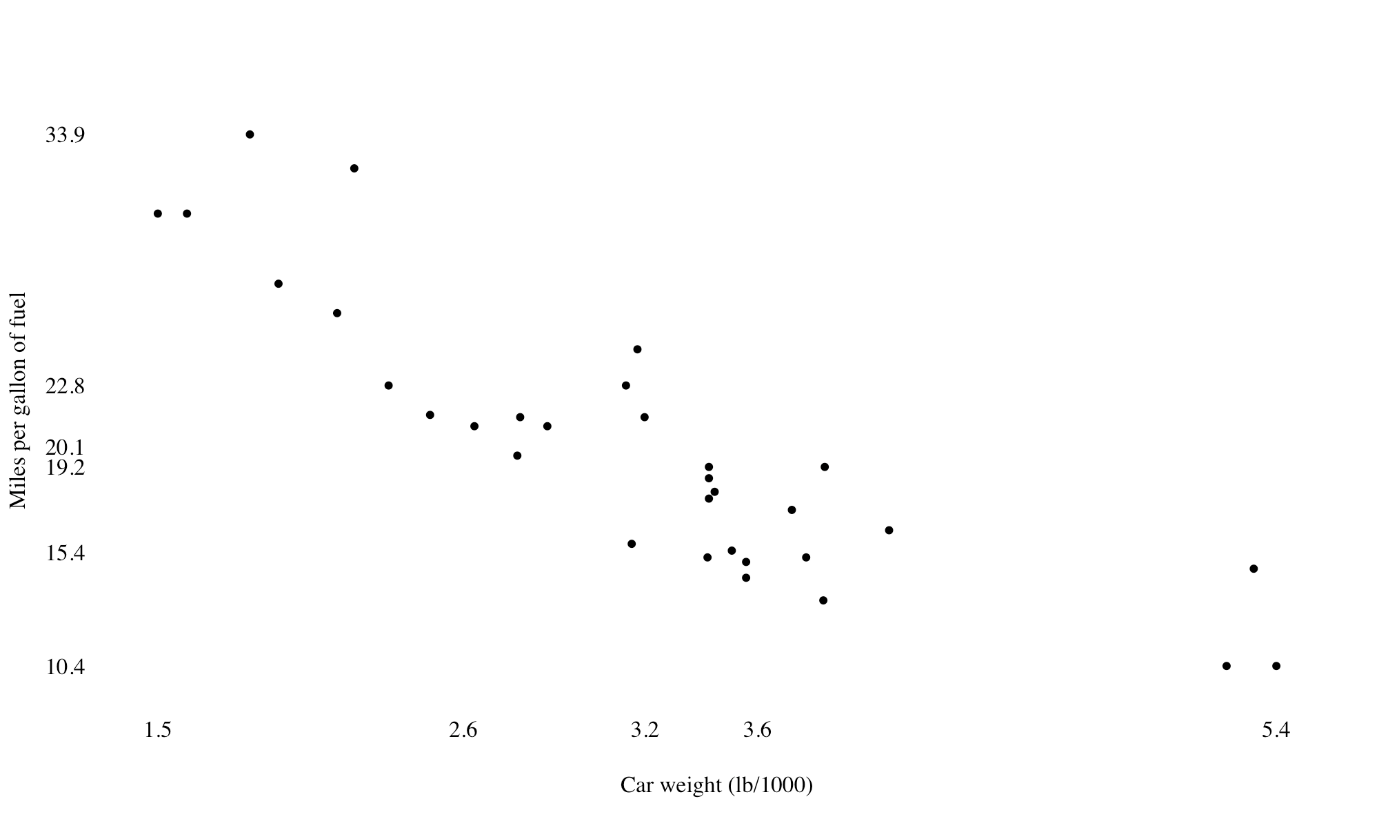
y <- mtcars$mpg

plot(x, y, main="", axes=FALSE, pch=16, cex=0.8, family="serif",

xlab="Car weight (lb/1000)", ylab="Miles per gallon of fuel")

axis(1,at=summary(x),labels=round(summary(x),1), tick=F, family="serif")

axis(2,at=summary(y),labels=round(summary(y),1), tick=F, las=2, family="serif")



*Range frame plot in lattice*

Again, I used [benbarnes axis hack](http://stackoverflow.com/questions/12901396/controlling-axis-ticks-and-axis-lines-separately-on-r-lattice-xyplot) from *Stackoverflow* to draw only axes ticks. Heavy use of par.settings to change the fontfamily to serif.

**library**(lattice)

x <- mtcars$wt

y <- mtcars$mpg

xyplot(y ~ x, mtcars, col=1, pch=16, fontfamily="serif",

xlab="Car weight (lb/1000)", ylab="Miles per gallon of fuel",

par.settings = list(axis.line = list(col="transparent"),

par.xlab.text=list(fontfamily="serif"),

par.ylab.text=list(fontfamily="serif")),

scales = list(x=list(at=summary(mtcars$wt),labels=round(summary(mtcars$wt),1),

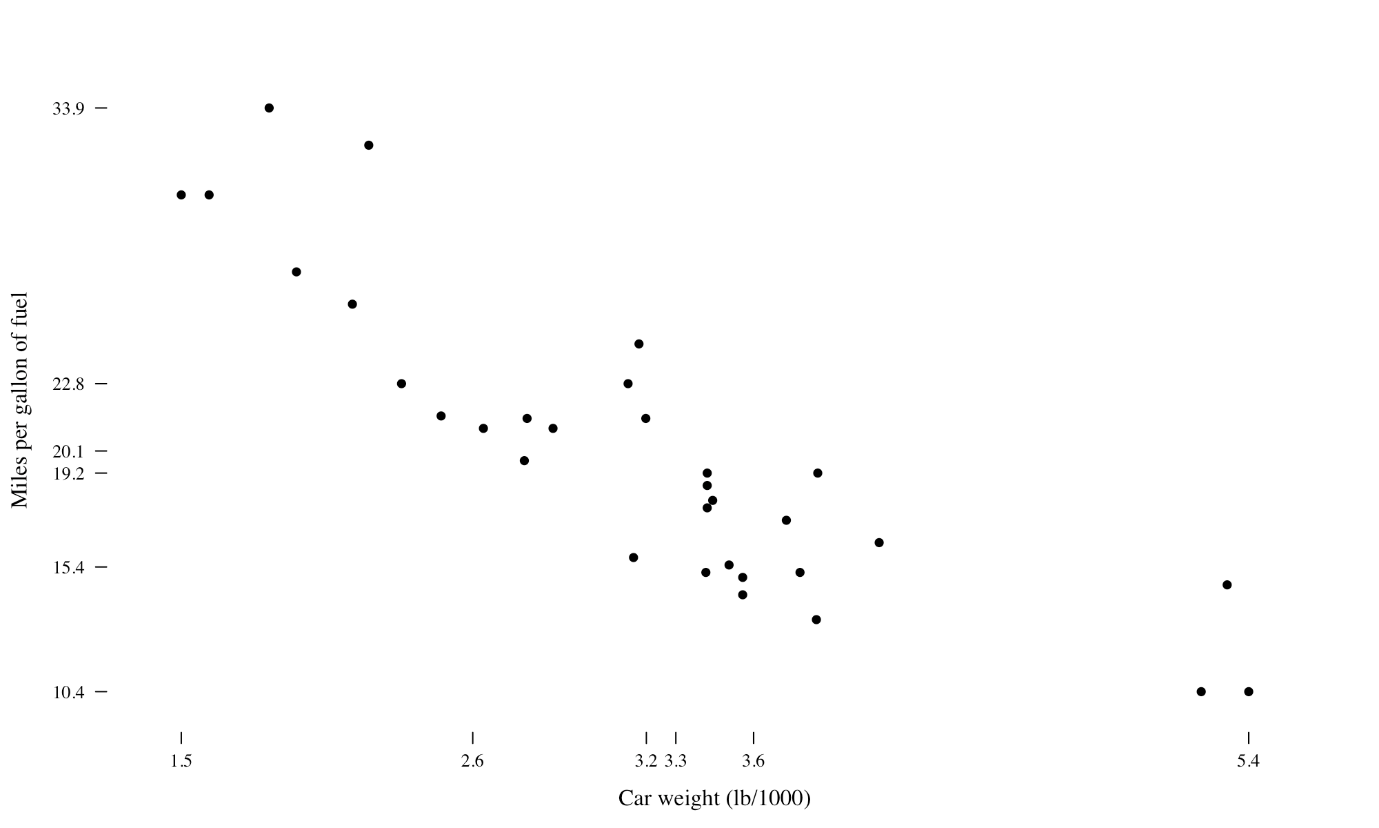
fontfamily="serif"),

y=list(at=summary(mtcars$mpg),labels=round(summary(mtcars$mpg),1),

fontfamily="serif")),

axis = **function**(side, line.col = "black", **...**) {

**if**(side %in% c("left","bottom")) {axis.default(side = side, line.col = "black", **...**)}})



*Range-frame plot in ggplot2*

Another use of package [ggthemes](https://cran.r-project.org/package=ggthemes) by Jeffrey B. Arnold - this time for geom\_rangeframe().

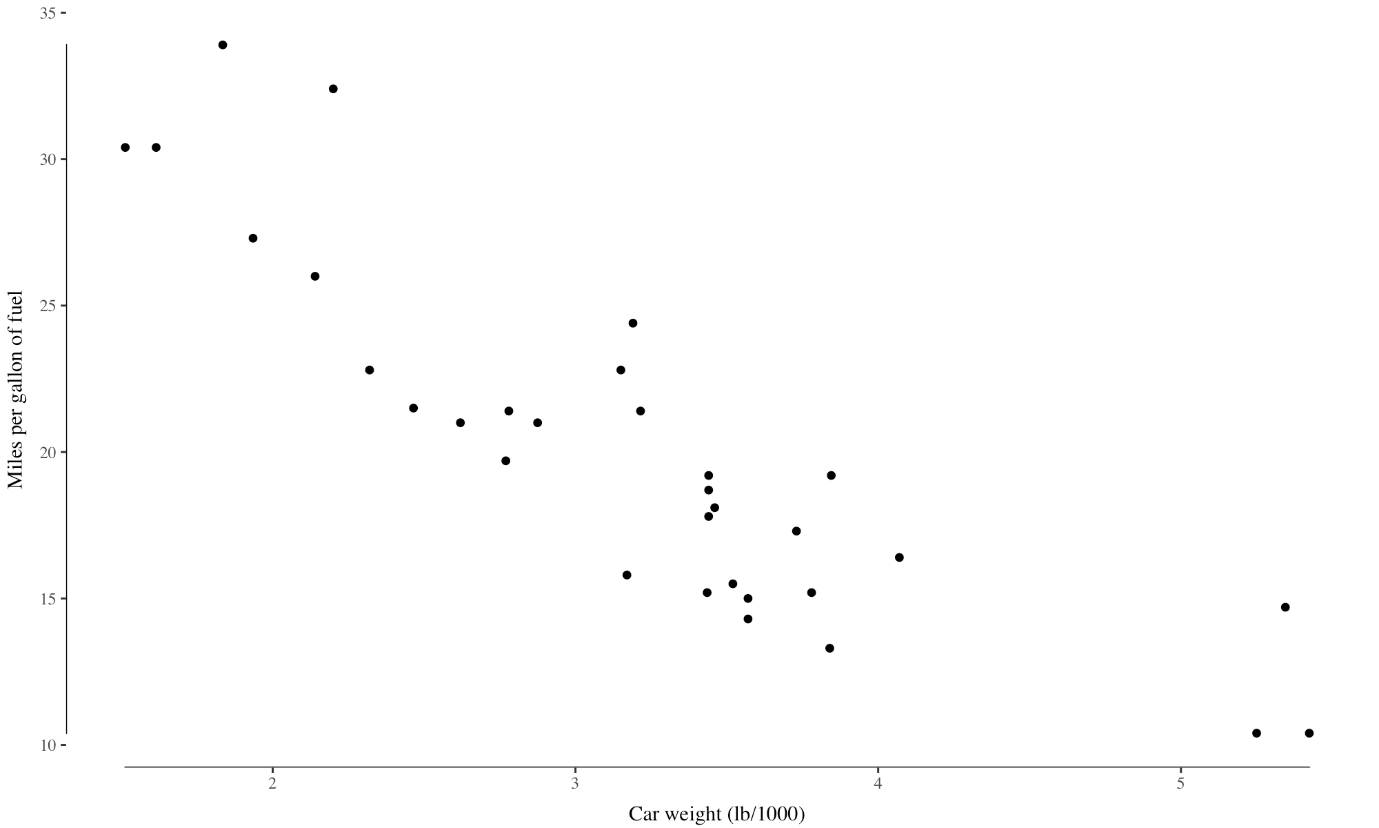
**library**(ggplot2)

**library**(ggthemes)

ggplot(mtcars, aes(wt, mpg)) + geom\_point() + geom\_rangeframe() + theme\_tufte() +

xlab("Car weight (lb/1000)") + ylab("Miles per gallon of fuel") +

theme(axis.title.x = element\_text(vjust=-0.5), axis.title.y = element\_text(vjust=1.5))



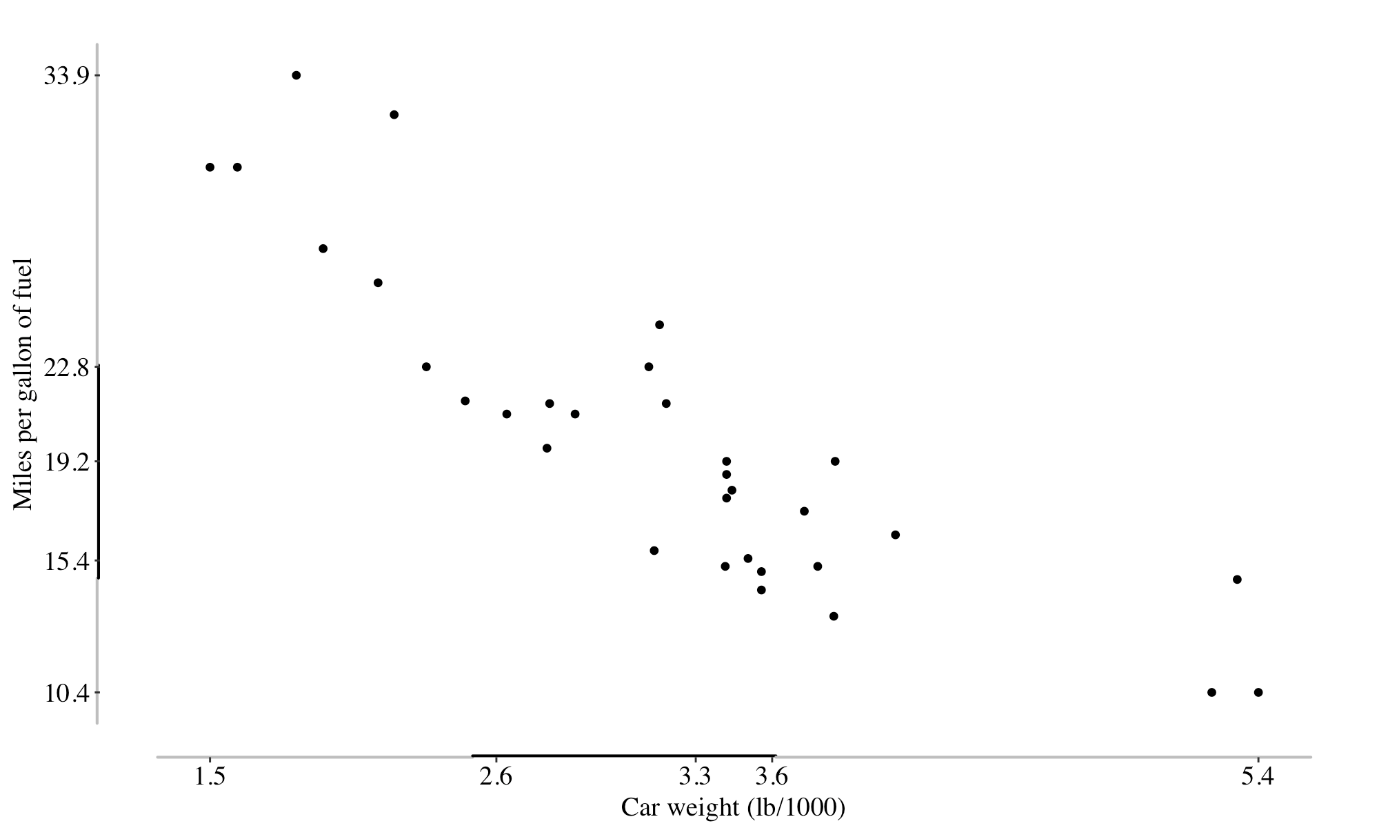
*Range-frame plot in ggplot2 with qfplot*

Function [qfplot](https://github.com/bearloga/Quartile-frame-Scatterplot) by Mikhail Y. Popov - limited customisation, but simple to deploy.

**library**(devtools)

source\_url('https://raw.githubusercontent.com/bearloga/Quartile-frame-Scatterplot/master/qfplot.R')

qfplot(x=mtcars$wt, y=mtcars$mpg, xlab="Car weight (lb/1000)", ylab="Miles per gallon of fuel")



*Dot-dash (or rug) scatterplot*

*Dot-dash plot in lattice*

**library**(lattice)

x <- mtcars$wt

y <- mtcars$mpg

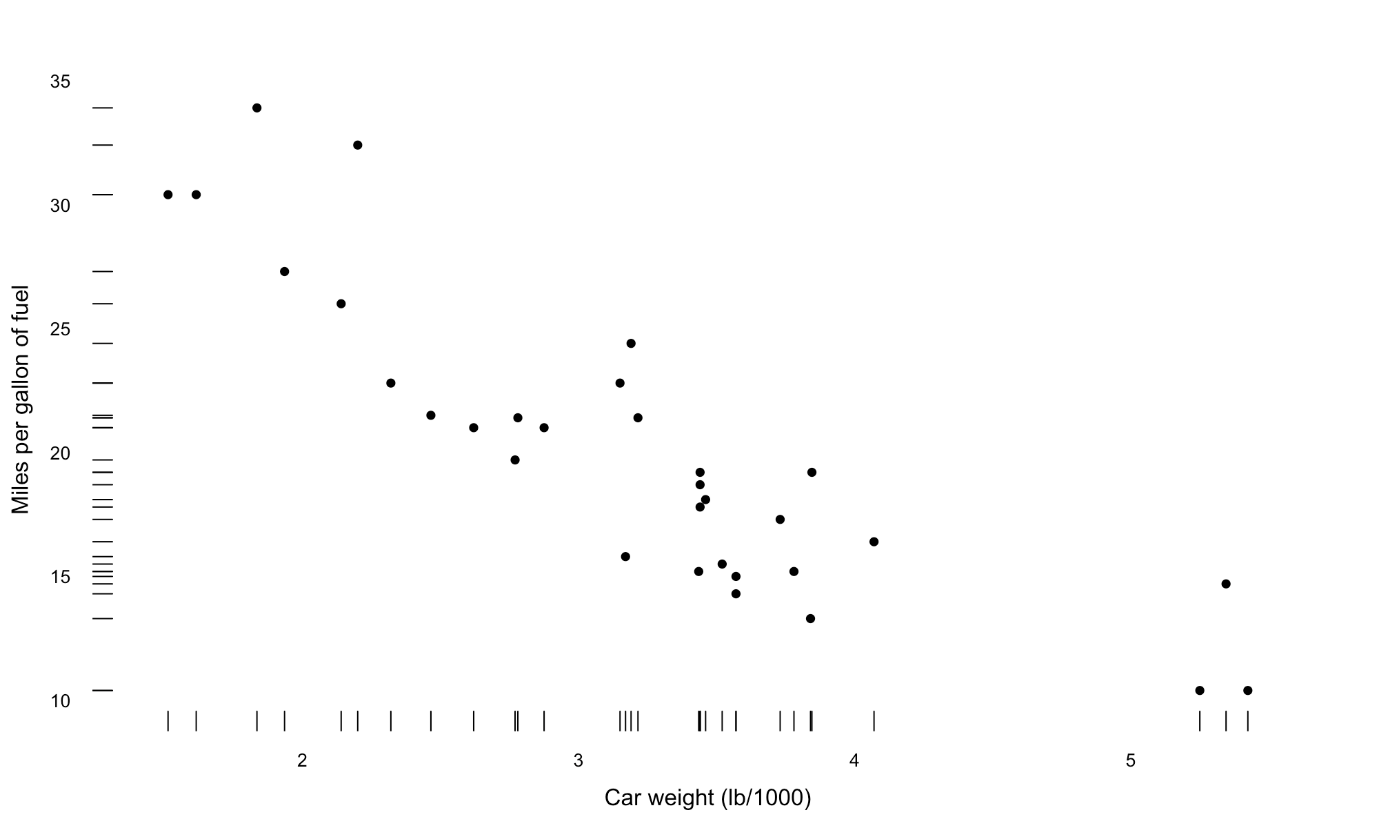
xyplot(y ~ x, xlab="Car weight (lb/1000)", ylab="Miles per gallon of fuel",

par.settings = list(axis.line = list(col="transparent")),

panel = **function**(x, y,**...**) {

panel.xyplot(x, y, col=1, pch=16)

panel.rug(x, y, col=1, x.units = rep("snpc", 2), y.units = rep("snpc", 2), **...**)})



*Dot-dash plot in ggplot2*

Here I use a geom\_rug() function from ggplot2.

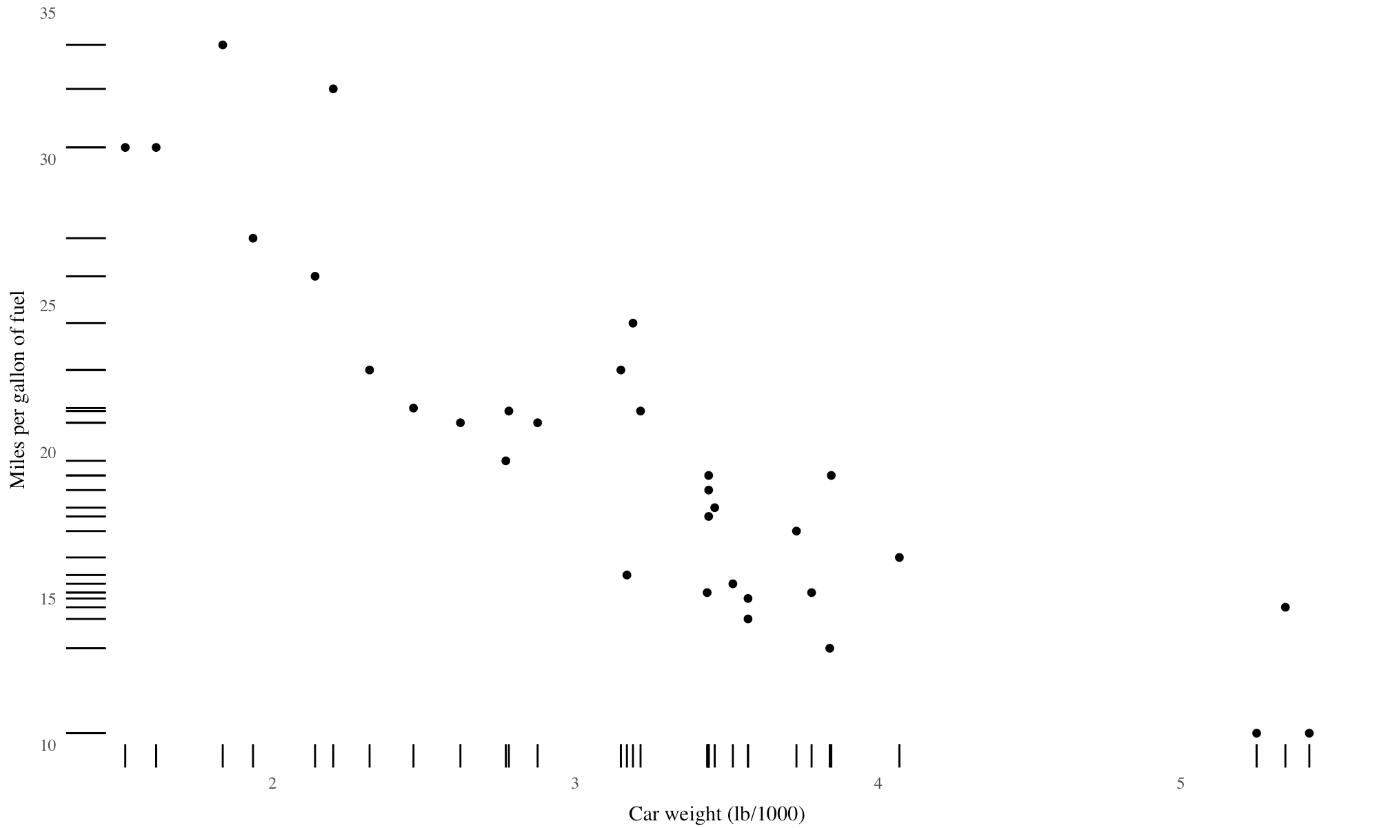
**library**(ggplot2)

**library**(ggthemes)

ggplot(mtcars, aes(wt, mpg)) + geom\_point() + geom\_rug() + theme\_tufte(ticks=F) +

xlab("Car weight (lb/1000)") + ylab("Miles per gallon of fuel") +

theme(axis.title.x = element\_text(vjust=-0.5), axis.title.y = element\_text(vjust=1))



*Marginal histogram scatterplot*

*Marginal histogram scatterplot in base graphics with fancyaxis*

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 133.

**library**(devtools)

source\_url("https://raw.githubusercontent.com/sjmurdoch/fancyaxis/master/fancyaxis.R")

x <- faithful$waiting

y <- faithful$eruptions

plot(x, y, main="", axes=FALSE, pch=16, cex=0.8,

xlab="Time till next eruption (min)", ylab="Duration (sec)",

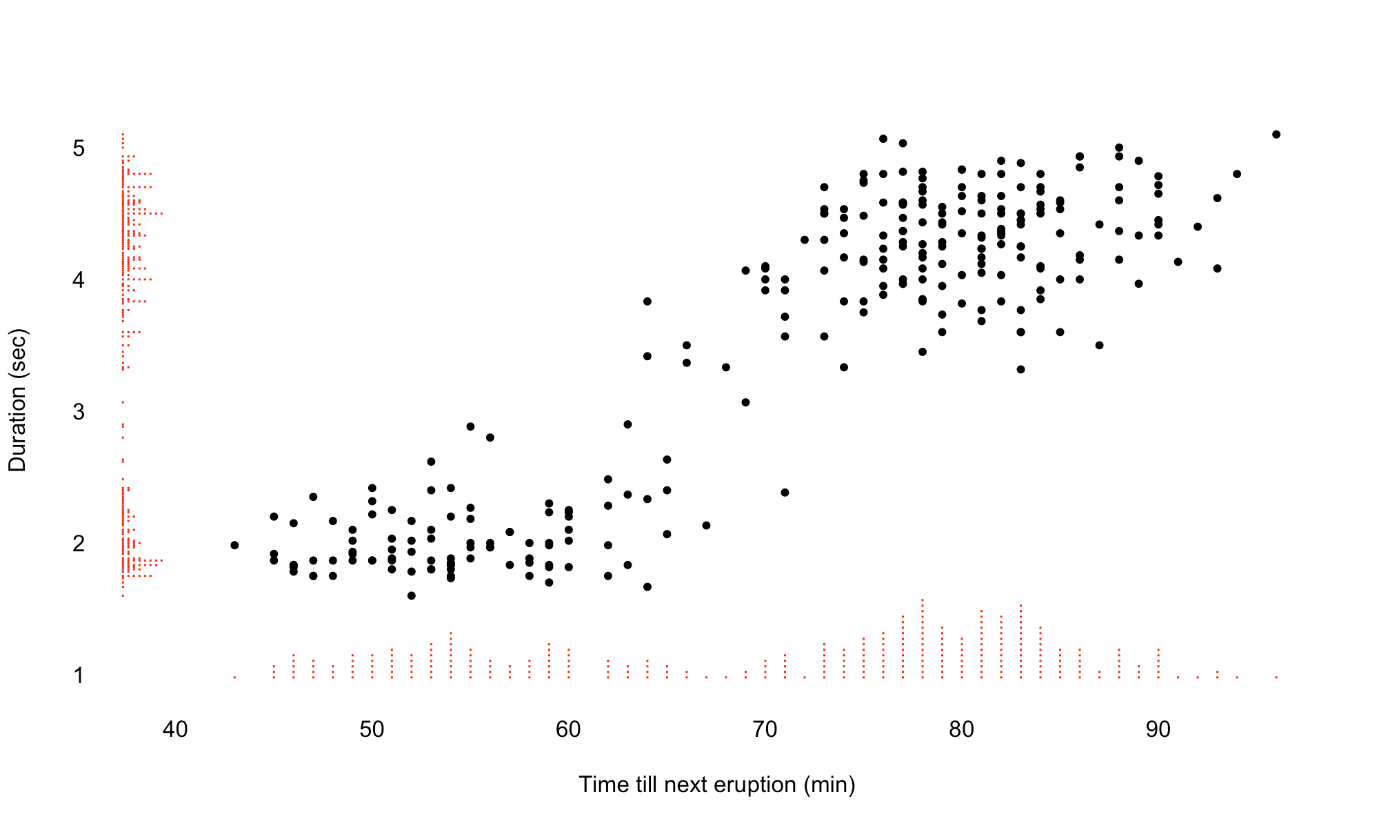
xlim=c(min(x)/1.1, max(x)), ylim=c(min(y)/1.5, max(y)))

axis(1, tick=F)

axis(2, tick=F, las=2)

axisstripchart(faithful$waiting, 1)

axisstripchart(faithful$eruptions, 2)



*Marginal histogram scatterplot in ggplot2 with ggMarginal*

Dean Attali’s ggExtra package includes function ggMarginal to create margin histograms. There is no option to create dot-histograms, just the standard bar-based histograms.

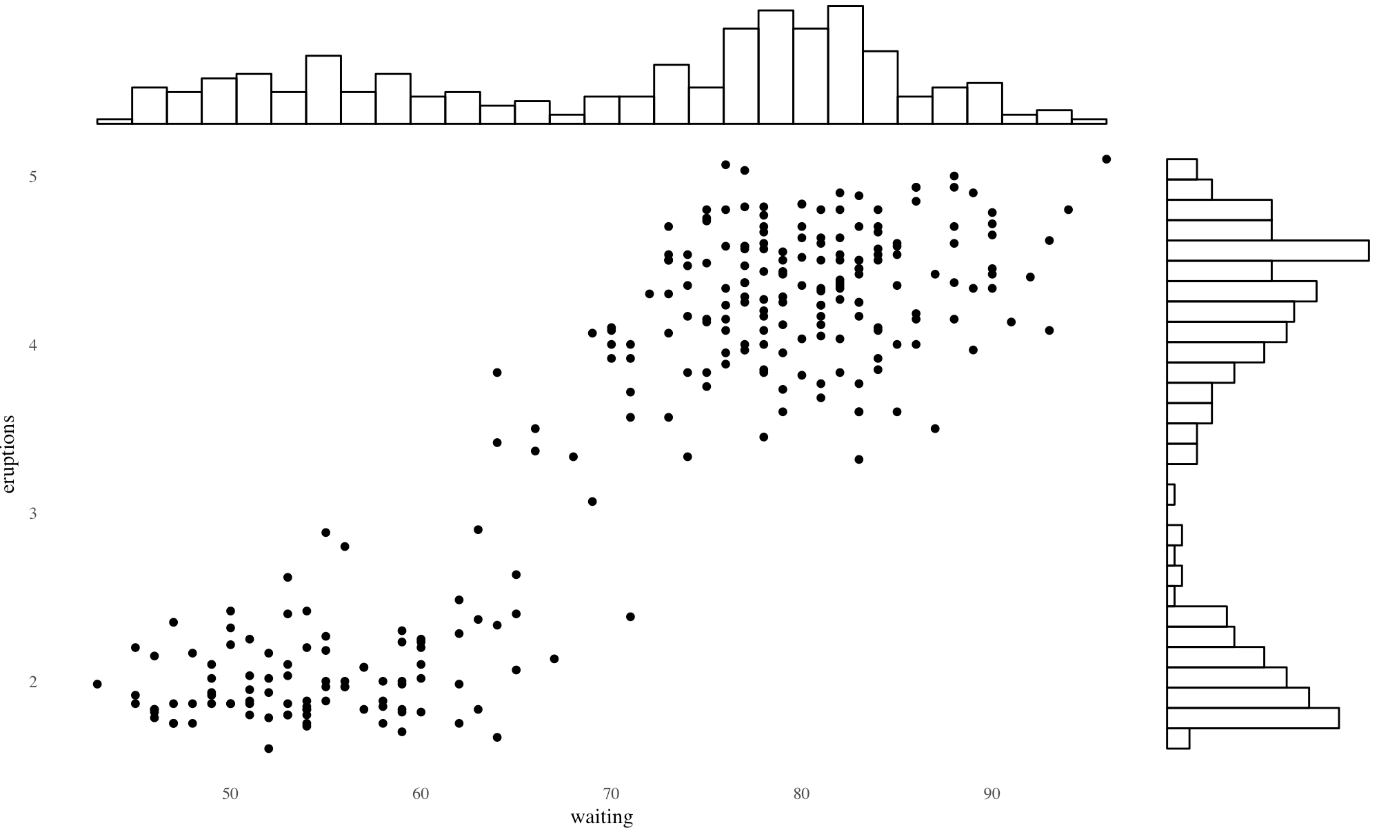
**library**(ggplot2)

**library**(ggExtra)

**library**(ggthemes)

p <- ggplot(faithful, aes(waiting, eruptions)) + geom\_point() + theme\_tufte(ticks=F)

ggMarginal(p, type = "histogram", fill="transparent")



However, [ggMarginal](http://deanattali.com/2015/03/29/ggExtra-r-package/) can be also used to quickly create margin densityplots using the same function:

**library**(ggplot2)

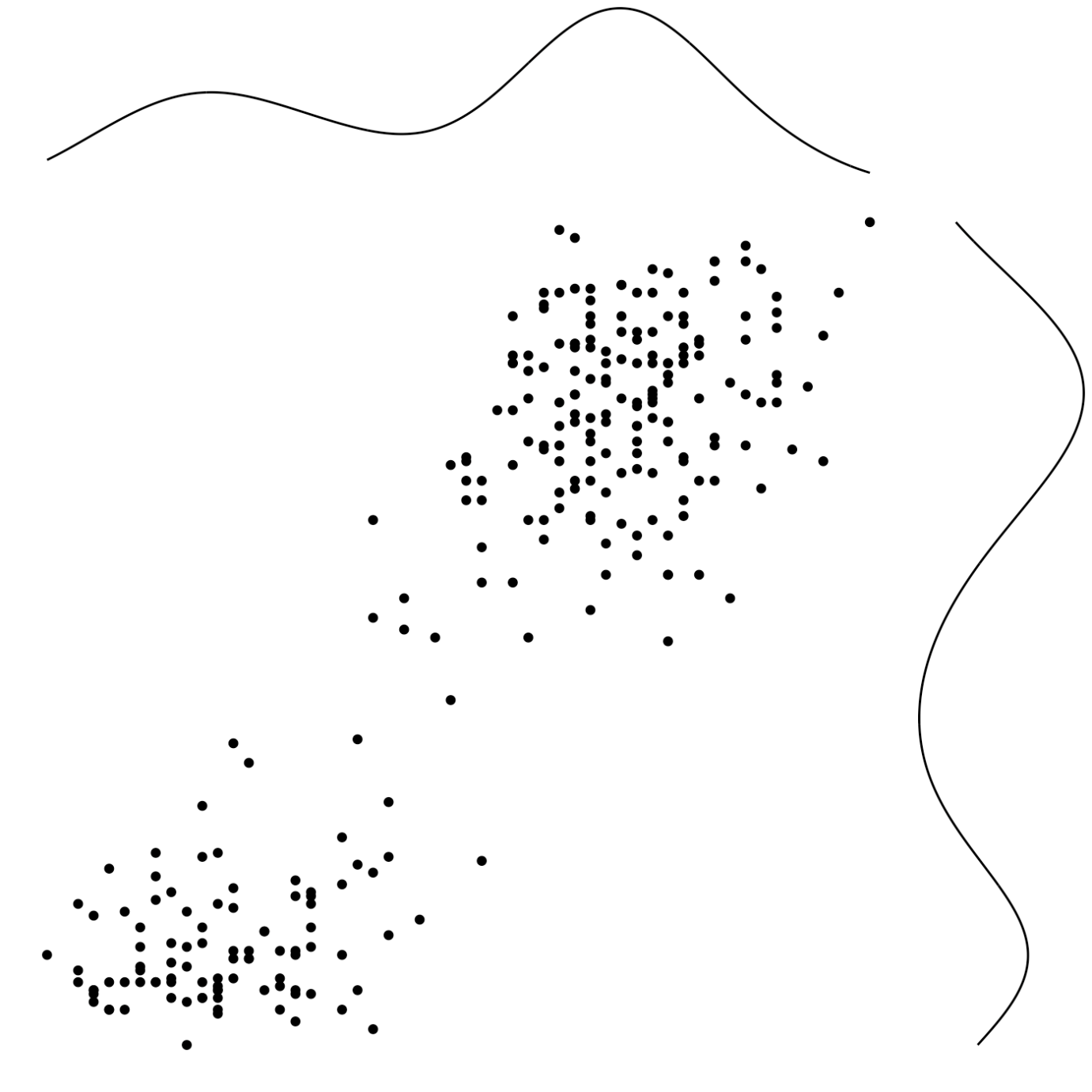
**library**(ggExtra)

**library**(ggthemes)

p <- ggplot(faithful, aes(waiting, eruptions)) + geom\_point() + theme\_tufte(ticks=F) +

theme(axis.title=element\_blank(), axis.text=element\_blank())

ggMarginal(p, type = "density")



Margin densityplot

…and it can also be used to create margin boxplots:

**library**(ggplot2)

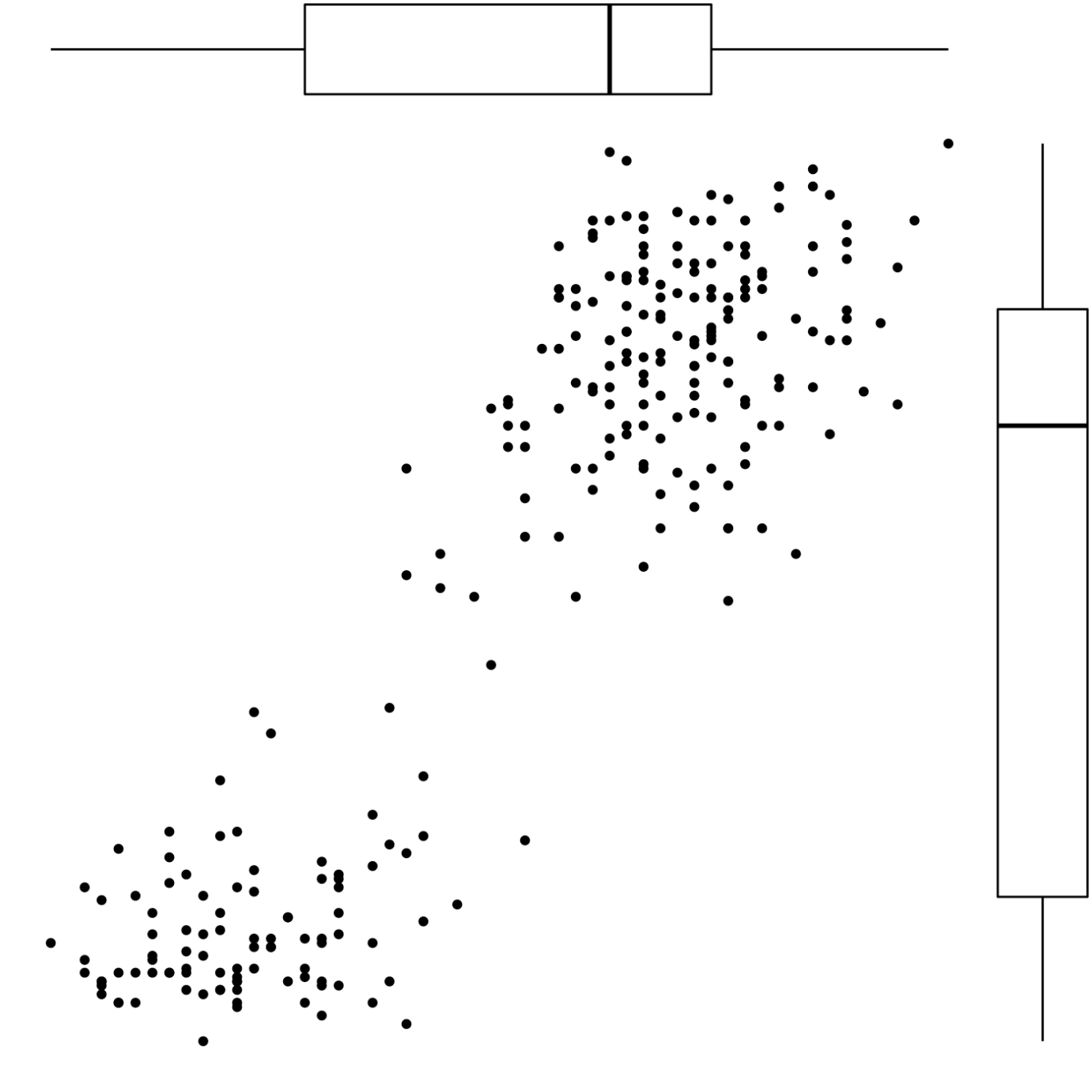
**library**(ggExtra)

**library**(ggthemes)

p <- ggplot(faithful, aes(waiting, eruptions)) + geom\_point() + theme\_tufte(ticks=F) +

theme(axis.title=element\_blank(), axis.text=element\_blank())

ggMarginal(p, type = "boxplot", size=10, fill="transparent")



Margin boxplot

*Minimal boxplot*

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 125 & 129. Argument pars is used to deconstruct the default base graphics boxplot.

*Minimal boxplot in base graphics*

x <- quakes$mag

y <- quakes$stations

boxplot(y ~ x, main = "", axes = FALSE, xlab=" ", ylab=" ",

pars = list(boxcol = "transparent", medlty = "blank", medpch=16, whisklty = c(1, 1),

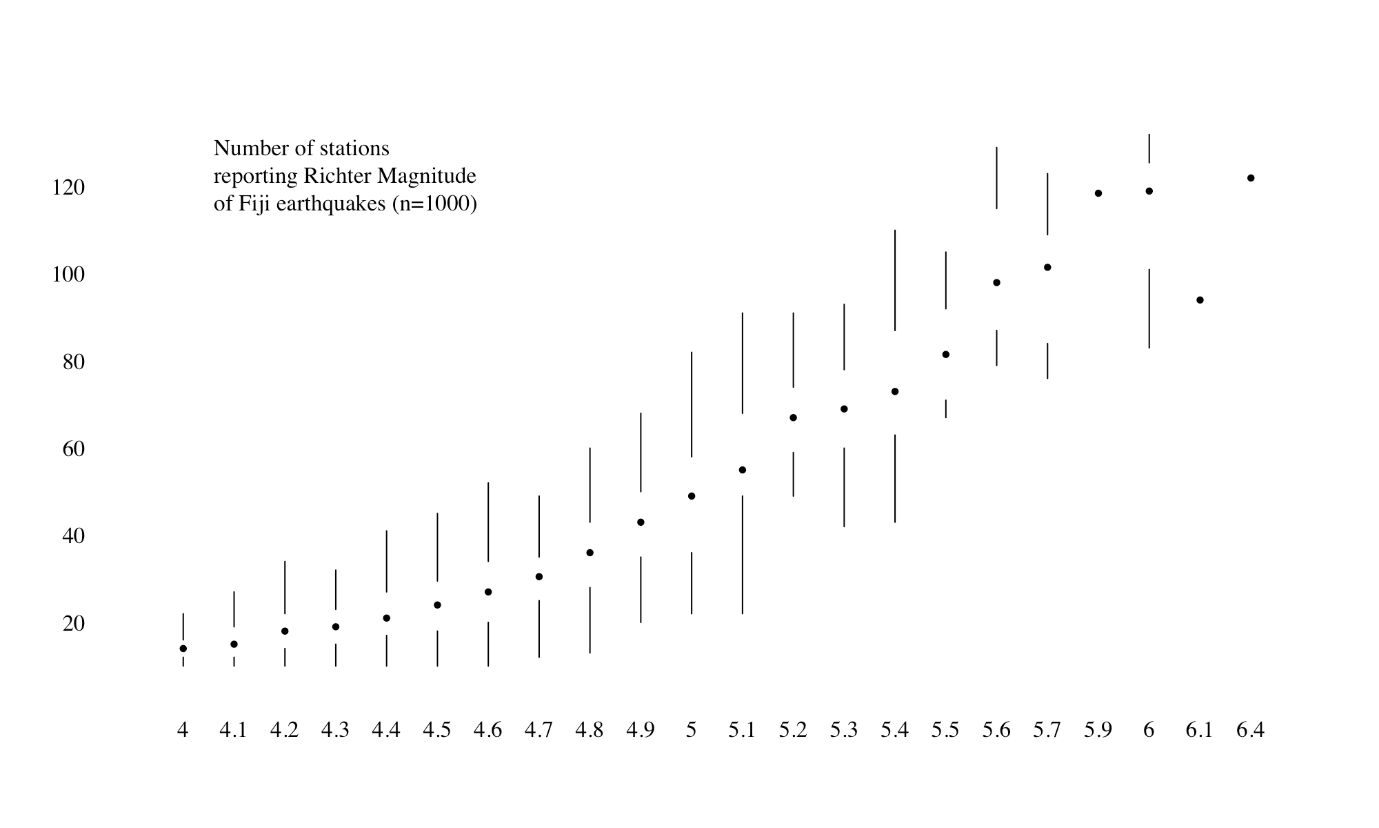
medcex = 0.7, outcex = 0, staplelty = "blank"))

axis(1, at=1:length(unique(x)), label=sort(unique(x)), tick=F, family="serif")

axis(2, las=2, tick=F, family="serif")

text(min(x)/3, max(y)/1.1, pos = 4, family="serif",

"Number of stations \nreporting Richter Magnitude\nof Fiji earthquakes (n=1000)")



*Minimal boxplot in base graphics with chart.Boxplot*

This uses chart.Boxplot function from PerformanceAnalytics package with dedicated as.Tufte=T argument. This solution requires data to be in wide table format and it has a limited customisation options.

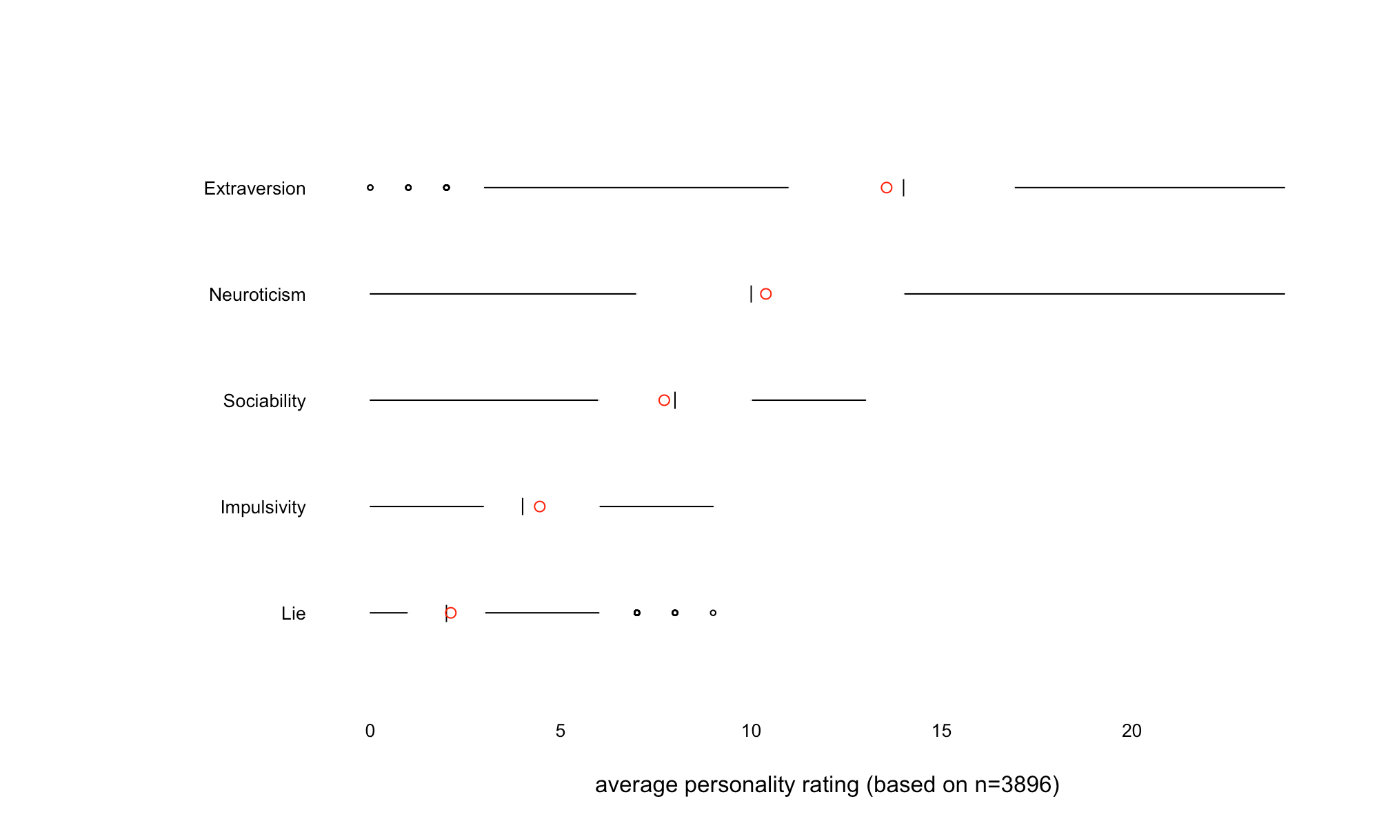
**library**(PerformanceAnalytics)

**library**(psych)

d <- msq[,80:84]

chart.Boxplot(d, main = "", xlab="average personality rating (based on n=3896)", ylab="",

element.color = "transparent", as.Tufte=TRUE)



*Minimal boxplot in lattice*

Argument par.settings is used to deconstruct the default lattice boxplot.

x <- quakes$mag

y <- quakes$stations

bwplot(y ~ x, horizontal=F, xlab="", ylab="", do.out = FALSE, box.ratio = 0,

scales=list(x=list(labels=sort(unique(x)), fontfamily="serif"),

y=list(fontfamily="serif")),

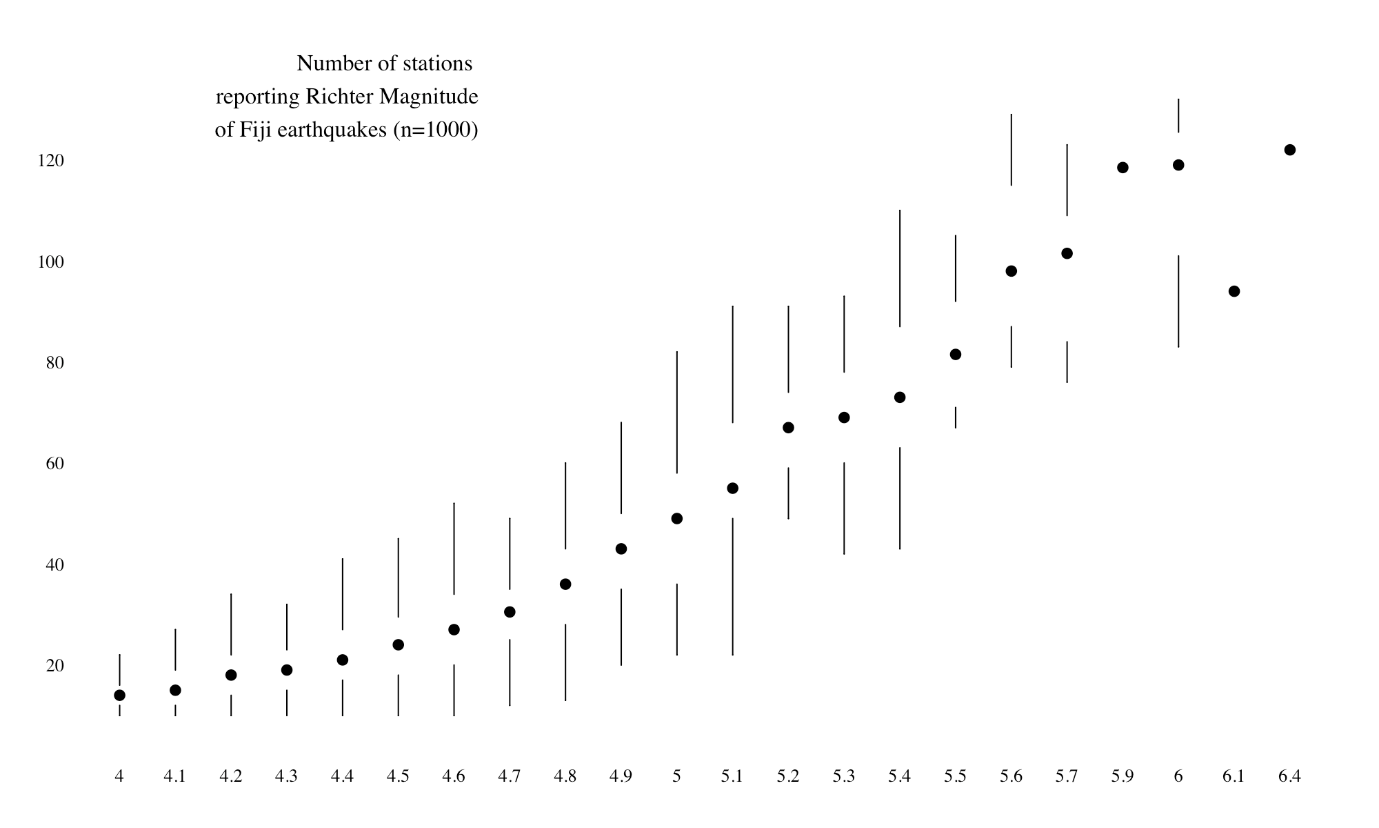
par.settings = list(axis.line = list(col = "transparent"), box.umbrella=list(lty=1, col= 1),

box.dot=list(col= 1), box.rectangle = list(col= c("transparent"))))

ltext(current.panel.limits()$xlim[1]+250, adj=1,

current.panel.limits()$ylim[2]+50, fontfamily="serif",

"Number of stations \nreporting Richter Magnitude\nof Fiji earthquakes (n=1000)")



*Minimal boxplot in ggplot2*

Function geom\_tufteboxplot from package ggthemes is used to draw boxplot in ggplot2.

**library**(ggplot2)

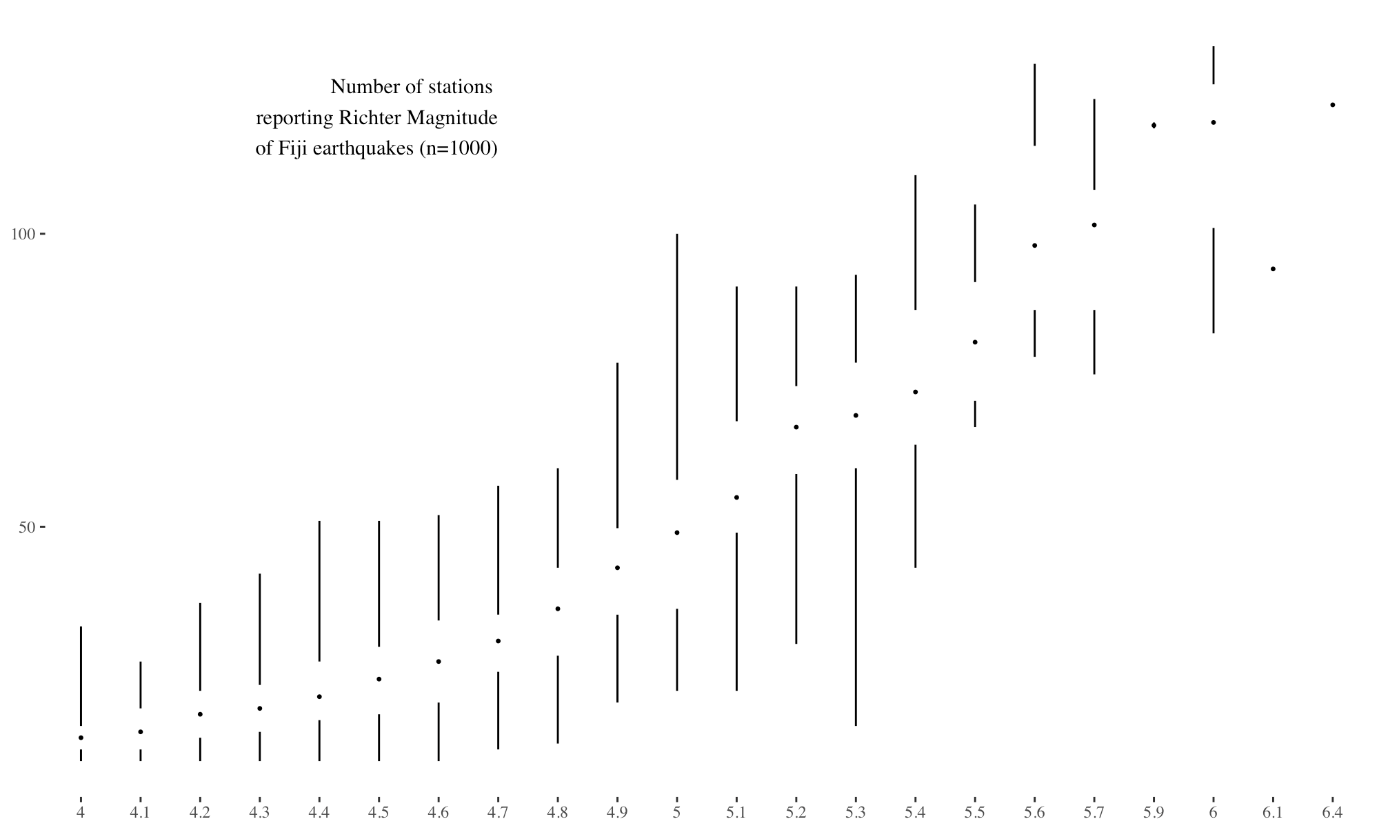
**library**(ggthemes)

ggplot(quakes, aes(factor(mag),stations)) + theme\_tufte() +

geom\_tufteboxplot(outlier.colour="transparent") + theme(axis.title=element\_blank()) +

annotate("text", x = 8, y = 120, adj=1, family="serif",

label = c("Number of stations \nreporting Richter Magnitude\nof Fiji earthquakes (n=1000)"))



*Minimal barchart*

*Minimal barchart in base graphics*

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 125 & 129. Basic graphics has an awkward way to change the width of bars in barplot - especially when you want to draw axis names separately. It requires some tweaking of arguments width and space, as well as location in axis() function. I use ablines to draw Tufte-like grid lines.

**library**(psych)

d <- colMeans(msq[,c(2,7,34,36,42,43,46,55,68)], na.rm = T)\*10

barplot(d, xaxt="n", yaxt="n", ylab="", border=F, width=c(.35), space=1.8)

axis(1, at=(1:length(d))-.26, labels=names(d), tick=F, family="serif")

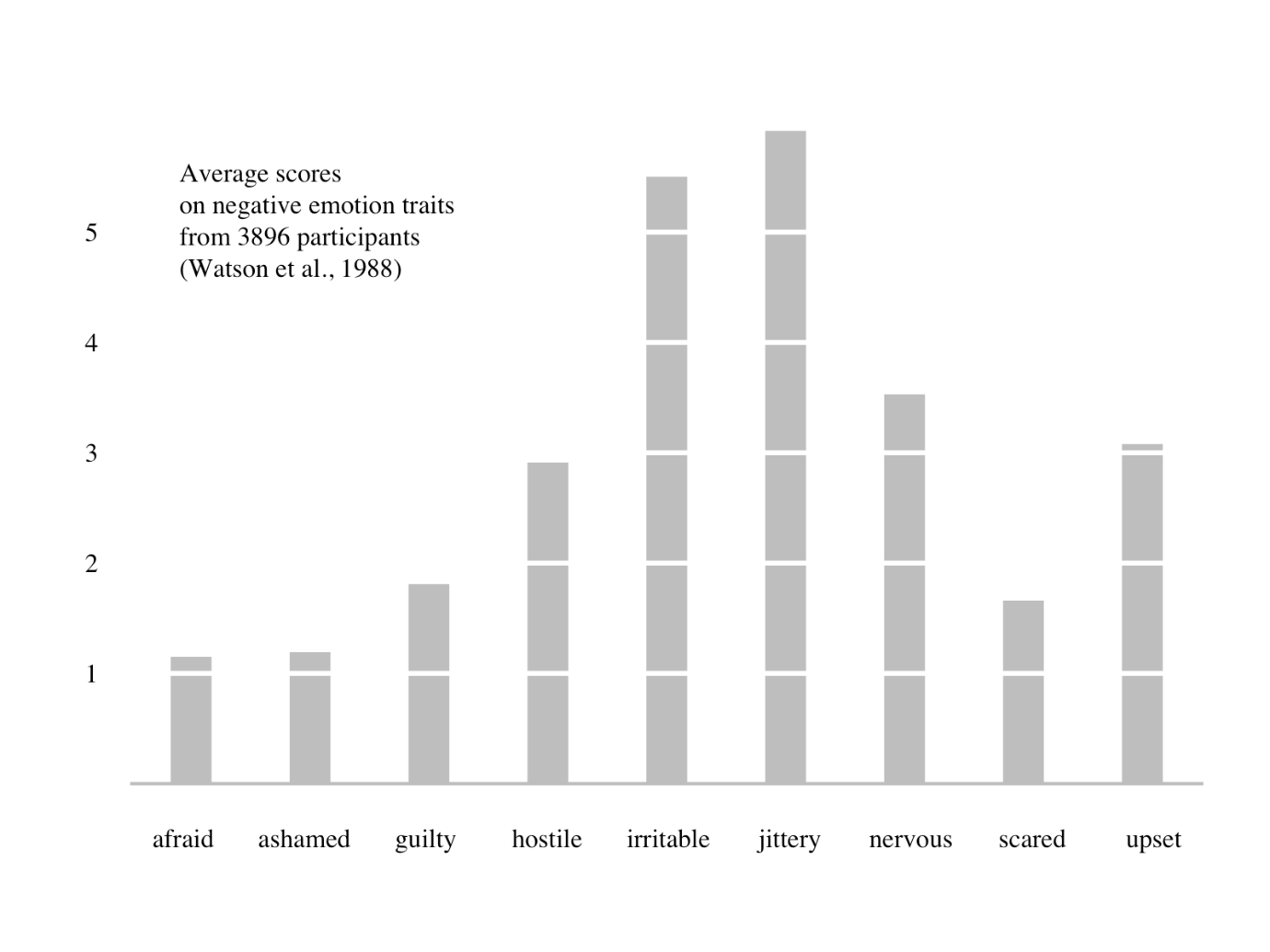
axis(2, at=seq(1, 5, 1), las=2, tick=F, family="serif")

abline(h=seq(1, 5, 1), col="white", lwd=3)

abline(h=0, col="gray", lwd=2)

text(min(d)/2, max(d)/1.2, pos = 4, family="serif",

"Average scores\non negative emotion traits\nfrom 3896 participants\n(Watson et al., 1988)")



*Minimal barchart in lattice*

Lattice barchart draws bars horizontally by default and it gets messy if you change it to vertical bars. Function panel.abline is used to draw grid lines.

**library**(lattice)

**library**(psych)

d <- colMeans(msq[,c(2,7,34,36,42,43,46,55,68)],na.rm = T)\*10

barchart(sort(d), xlab="", ylab="", col = "grey", origin=1,

border = "transparent", box.ratio=0.5,

panel = **function**(x,y,**...**) {

panel.barchart(x,y,**...**)

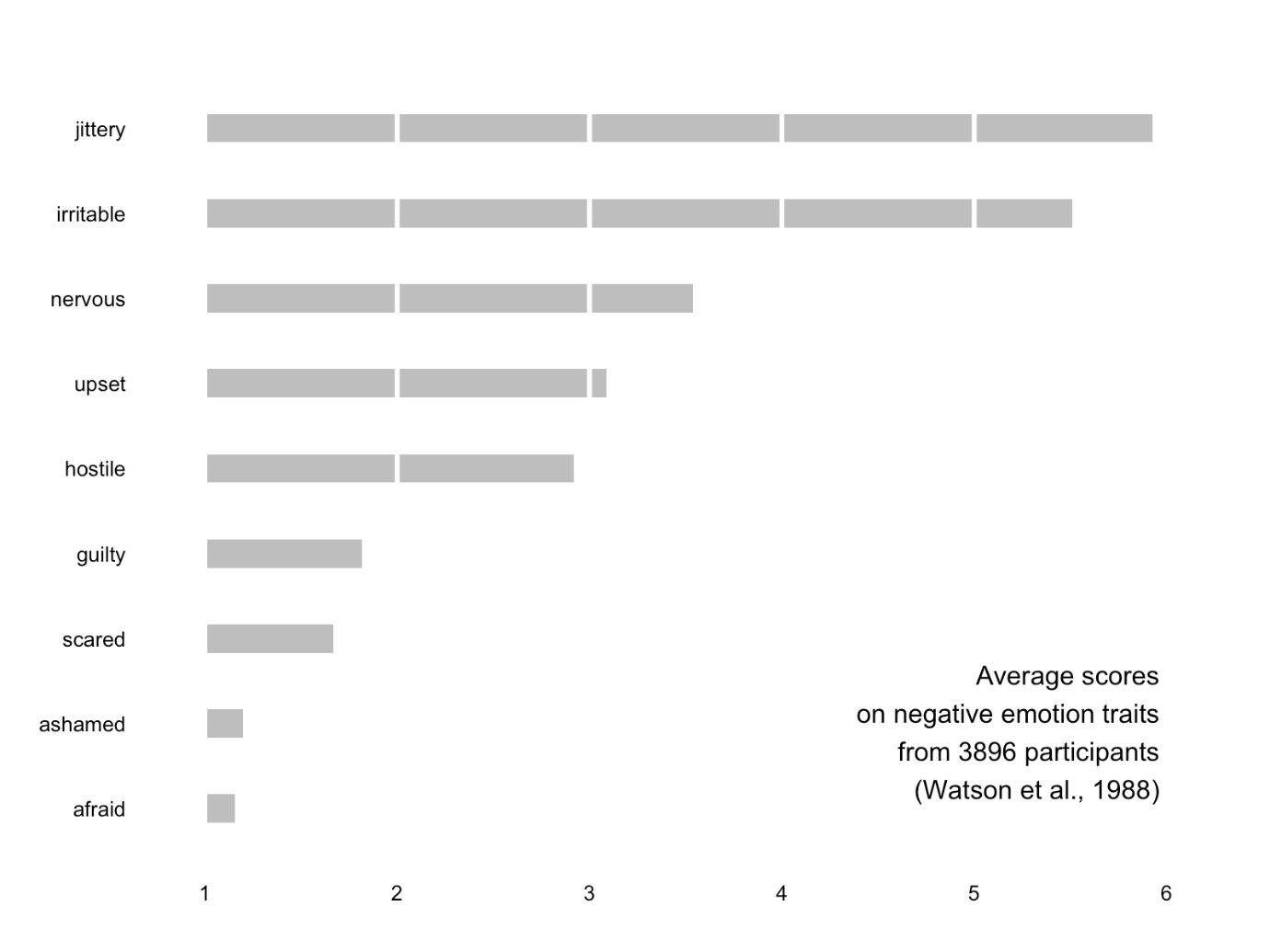
panel.abline(v=seq(1,6,1), col="white", lwd=3)},

par.settings = list(axis.line = list(col = "transparent")))

ltext(current.panel.limits()$xlim[2]-50, adj=1,

current.panel.limits()$ylim[1]-100,

"Average scores\non negative emotion traits\nfrom 3896 participants\n(Watson et al., 1988)")



*Minimal barchart in ggplot2*

**library**(ggplot2)

**library**(ggthemes)

**library**(psych)

**library**(reshape2)

d <- melt(colMeans(msq[,c(2,7,34,36,42,43,46,55,68)],na.rm = T)\*10)

d$trait <- rownames(d)

ggplot(d, aes(x=trait, y=value)) + theme\_tufte(base\_size=14, ticks=F) +

geom\_bar(width=0.25, fill="gray", stat = "identity") + theme(axis.title=element\_blank()) +

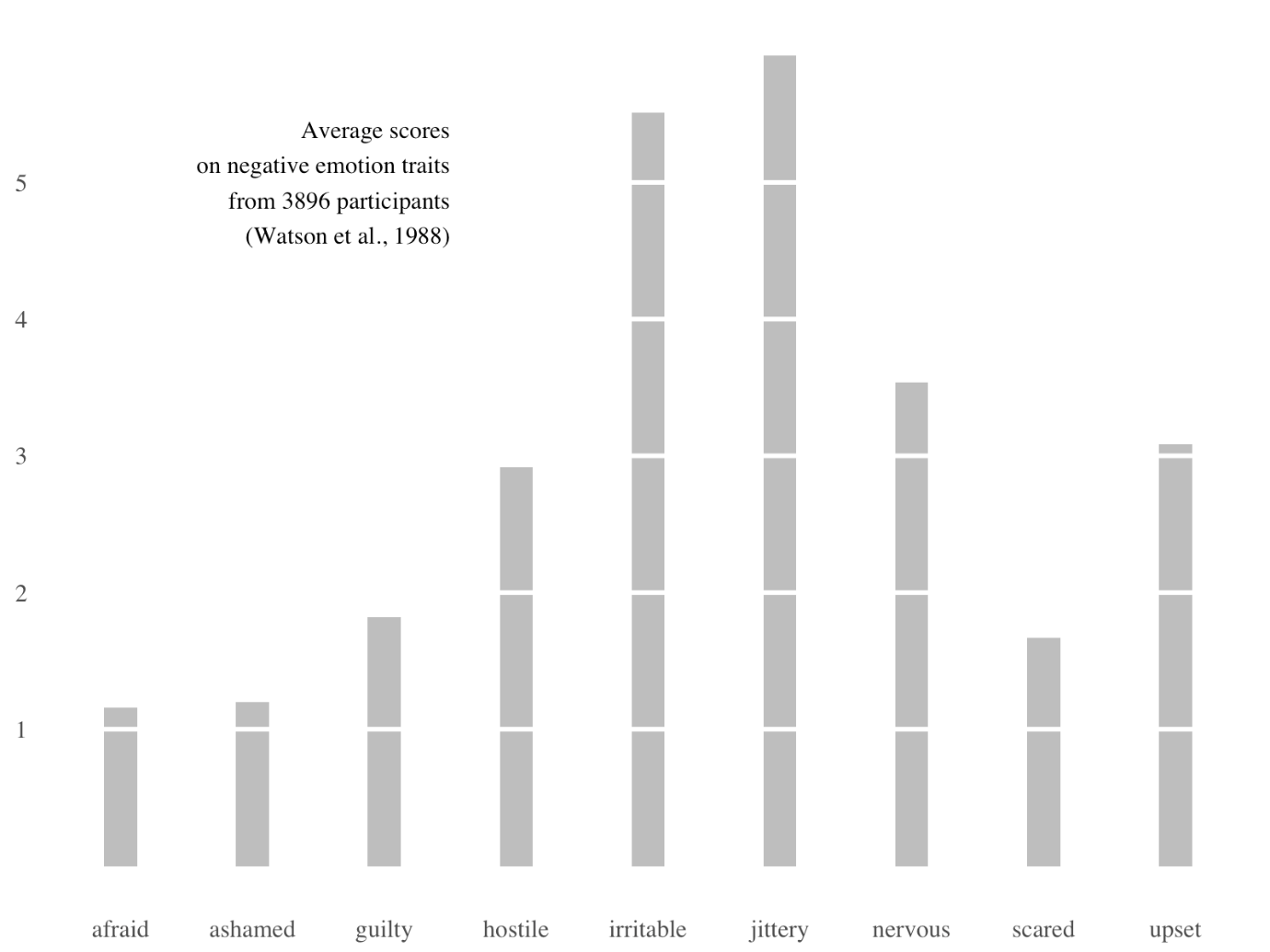
scale\_y\_continuous(breaks=seq(1, 5, 1)) +

geom\_hline(yintercept=seq(1, 5, 1), col="white", lwd=1) +

annotate("text", x = 3.5, y = 5, adj=1, family="serif",

label = c("Average scores\non negative emotion traits

from 3896 participants\n(Watson et al., 1988)"))



*Minimal barchart - interactive with highcharter*

**library**(psych)

**library**(reshape)

**library**(highcharter)

values <- 1 + abs(rnorm(12))

d <- melt(colMeans(msq[,c(2,7,34,36,42,43,46,55,68)], na.rm = T)\*10)

trait <- row.names(d)

value <- as.vector(d[,1])

highchart() %>%

hc\_chart(type = "column") %>%

hc\_add\_series(data = value) %>%

hc\_xAxis(categories = row.names(d)) %>%

hc\_add\_theme(hc\_theme\_tufte2())

afraidashamedguiltyhostileirritablejitterynervousscaredupset01234567

*Slopegraph*

*Slopegraph in base graphics*

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 158. Using slopegraph package.

The most promising slopegraph functions for *base graphics* and *ggplot2* comes from slopegraph package. Thomas’s solutions have evolved gradually and it’s now the most efficient method to create slopegraphs in *R*. However, a major limitation is inability to efficently offset left and right side labels to avoid don’t overlap (as seen below).

**library**(slopegraph)

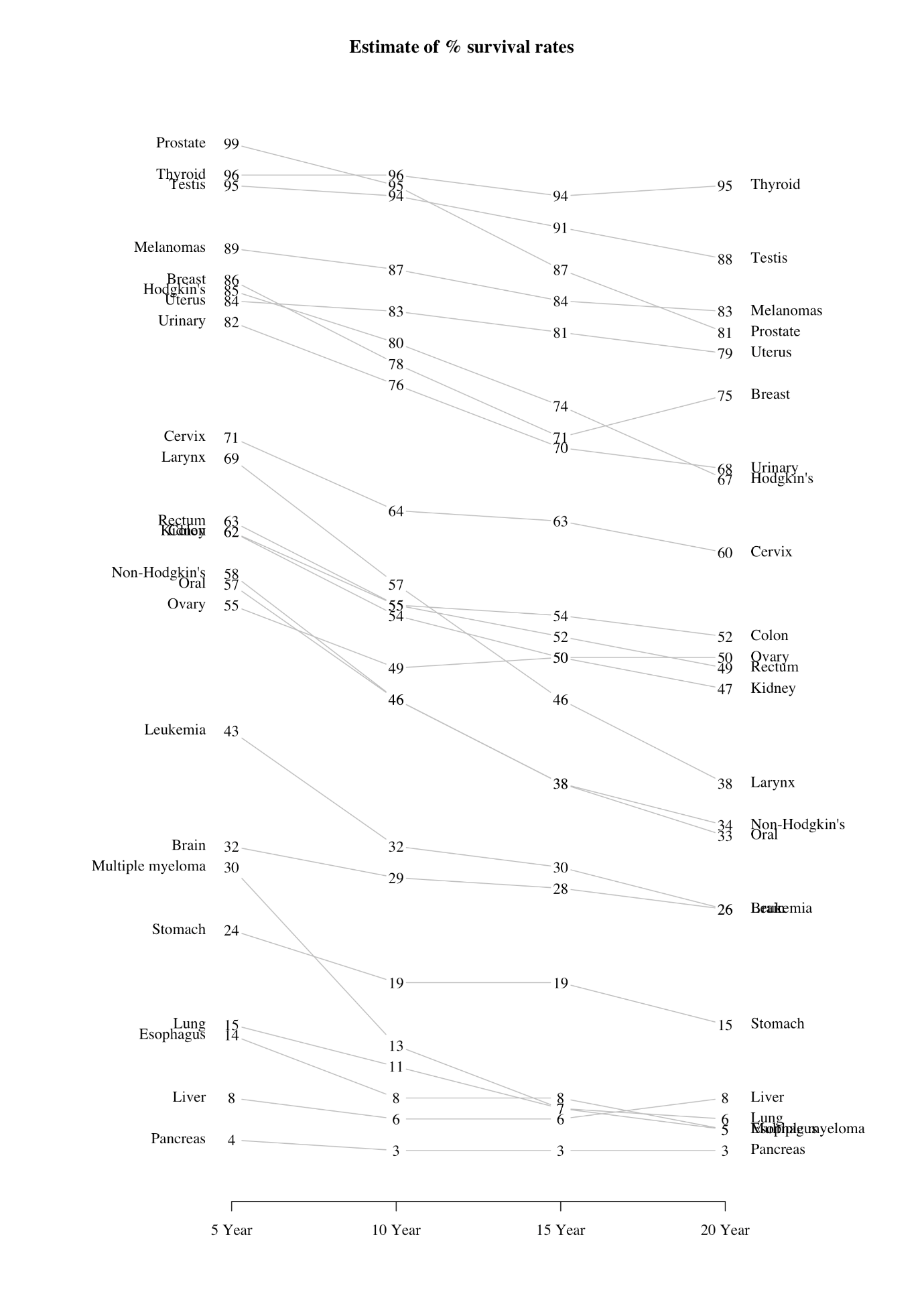
data(cancer)

slopegraph(cancer, col.lines = 'gray', col.lab = 1, col.num = 1,

xlim = c(-.2,5),

main = "Estimate of % survival rates",

xlabels = c('5 Year','10 Year','15 Year','20 Year'))



*Slopegraph in ggplot2 with plot\_slopegraph*

*# library(ggplot2)*

*# library(ggthemes)*

*# library(devtools)*

*# library(RCurl)*

*# library(plyr)*

*# source\_url("https://raw.githubusercontent.com/jkeirstead/r-slopegraph/master/slopegraph.r")*

*# d <- read.csv(text = getURL("https://raw.githubusercontent.com/jkeirstead/r-slopegraph/master/cancer\_survival\_rates.csv"))*

*# df <- build\_slopegraph(d, x="year", y="value", group="group", method="tufte", min.space=0.04)*

*# df <- transform(df, x=factor(x, levels=c(5,10,15,20),*

*# labels=c("5 years","10 years","15 years","20 years")), y=round(y))*

*# plot\_slopegraph(df) + labs(title="Estimates of % survival rates") +*

*# theme\_tufte(base\_size=16, ticks=F) + theme(axis.title=element\_blank())*

*Sparklines*

There is no ‘out-of-box’ solution in the existing packages that truly replicate Tufte-style sparklines. Main issues are scaling the size of the plot and labeling of the points - those factors are likely to change depending on the data set you’re plotting, so you will have to adjust specific parameters (which I highlight for every graphical system). To make the output more consistent, every sparkline plot will be automatically saved in the working directory in a vector format as a PDF (using pdf() and dev.off() functions).

A word of warning - in its current format, making sparklines requires a bit more advanced knowledge of *R*. Its far from perfect - proceed with caution.

*Sparklines in base graphics*

Sparklines in base graphics use some elements of functions from [YaleToolkit](https://cran.r-project.org/web/packages/YaleToolkit/index.html) developed by John Emerson and Walton Green. In particular, it’s a result of mine and [Ben’s](http://stackoverflow.com/users/1036500/ben) hacking of YaleToolkit functions on *[Stackoverflow](http://stackoverflow.com/questions/8337980/r-yaletoolkit-how-to-change-the-font-size-of-tick-labels-on-the-sparklines/8345519" \l "8345519)*. I’ve use a simple loop that takes a number of columns in a data set and creates as much sparklines as there are columns. In the same manner I use mfrow parameter in par() function to set the number of rows to a number of columns in data frame.

Data on US Rate of Crime (per 100,000 people) from 1960 to 2014 comes from *FBI UCS Annual Crime Reports* compiled by [disastercenter.com](http://www.disastercenter.com/crime/uscrime.htm).

**library**(RCurl)

dd <- read.csv(text = getURL("https://gist.githubusercontent.com/GeekOnAcid/da022affd36310c96cd4/raw/9c2ac2b033979fcf14a8d9b2e3e390a4bcc6f0e3/us\_nr\_of\_crimes\_1960\_2014.csv"))

d <- dd[,c(2:11)]

pdf("sparklines\_base.pdf", height=10, width=6)

par(mfrow=c(ncol(d),1), mar=c(1,0,0,8), oma=c(4,1,4,4))

**for** (i **in** 1:ncol(d)){

plot(d[,i], lwd=0.5, axes=F, ylab="", xlab="", main="", type="l", new=F)

axis(4, at=d[nrow(d),i], labels=round(d[nrow(d),i]), tick=F, las=1, line=-1.5,

family="serif", cex.axis=1.2)

axis(4, at=d[nrow(d),i], labels=names(d[i]), tick=F, line=1.5,

family="serif", cex.axis=1.4, las=1)

text(which.max(d[,i]), max(d[,i]), labels=round(max(d[,i]),0),

family="serif", cex=1.2, adj=c(0.5,3))

text(which.min(d[,i]), min(d[,i]), labels=round(min(d[,i]),0),

family="serif", cex=1.2, adj=c(0.5,-2.5))

ymin <- min(d[,i]); tmin <- which.min(d[,i]); ymax<-max(d[,i]); tmax<-which.max(d[,i]);

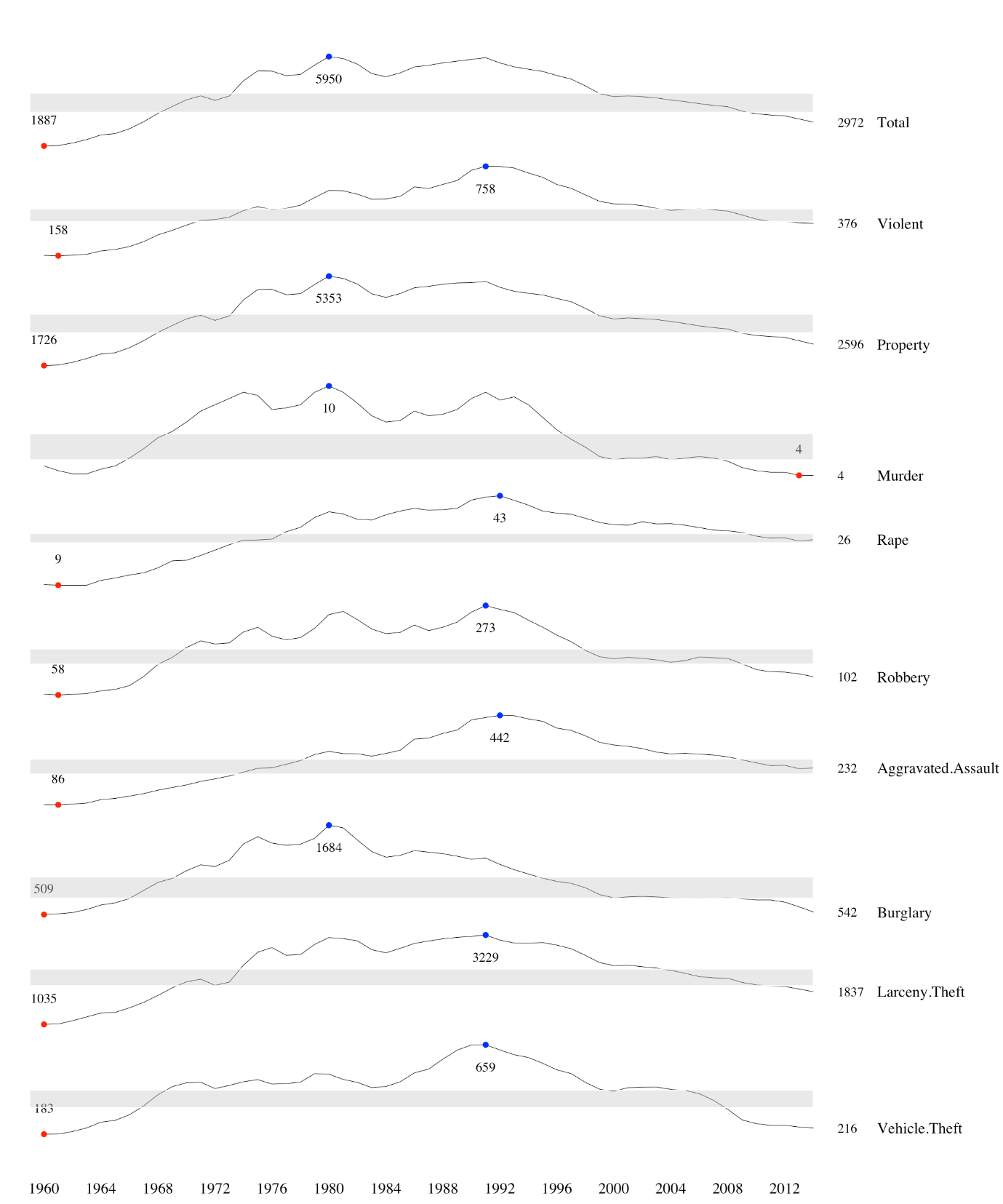
points(x=c(tmin,tmax), y=c(ymin,ymax), pch=19, col=c("red","blue"), cex=1)

rect(0, summary(d[,i])[2], nrow(d), summary(d[,i])[4], border=0,

col = rgb(190, 190, 190, alpha=90, maxColorValue=255))}

axis(1, at=1:nrow(dd), labels=dd$Year, pos=c(-5), tick=F, family="serif", cex.axis=1.4)

dev.off()



*Sparklines in base graphics with plotSparklineTable*

This uses plotSparklineTable function from epanetReader package by Bradley Eck. This compact solution requires data to be in long table format and it has a limited customisation options. Great for making a rapid summaries with sparklines.

**library**(epanetReader)

**library**(reshape)

**library**(RCurl)

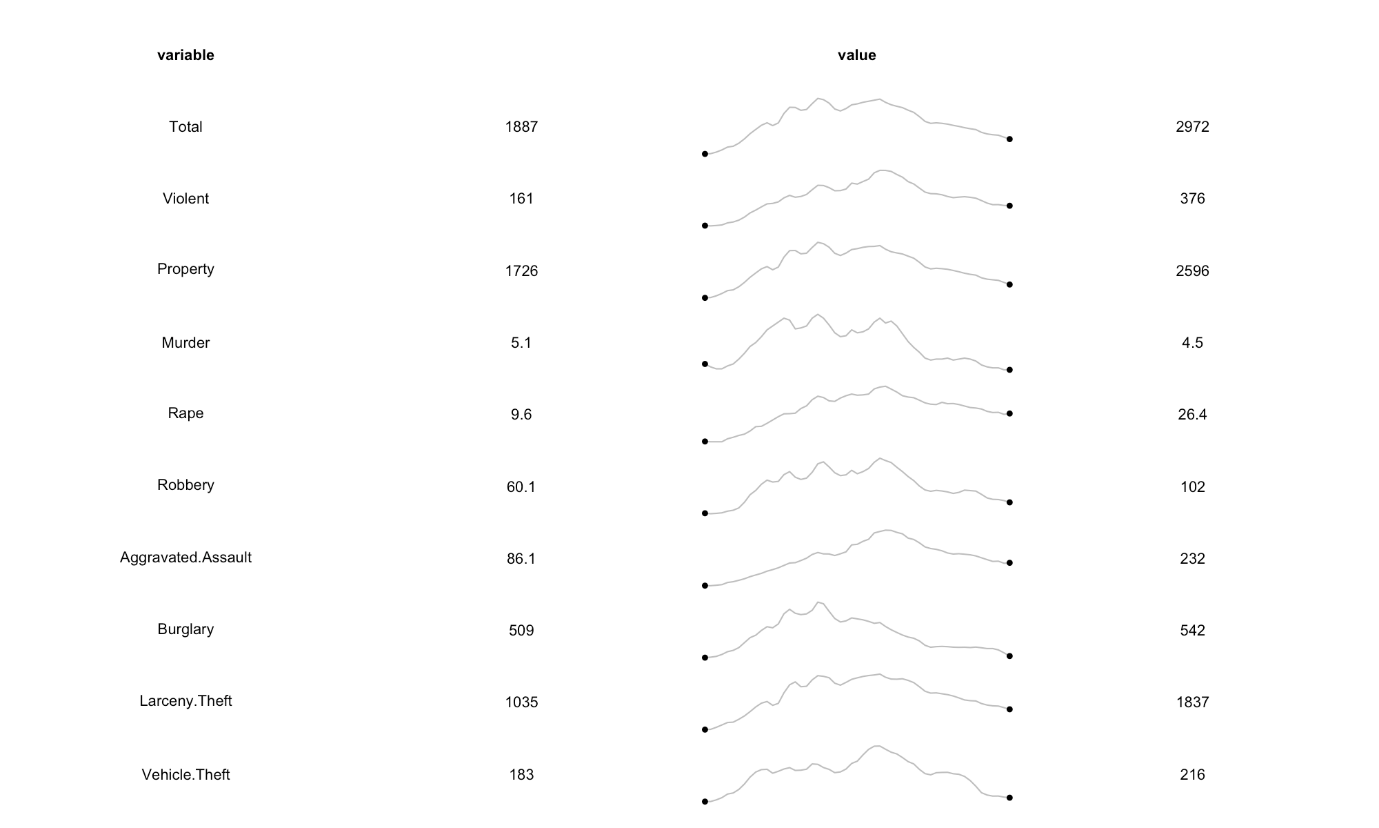
dd <- read.csv(text = getURL("https://gist.githubusercontent.com/GeekOnAcid/da022affd36310c96cd4/raw/9c2ac2b033979fcf14a8d9b2e3e390a4bcc6f0e3/us\_nr\_of\_crimes\_1960\_2014.csv"))

d <- melt(dd[,c(2:11)])

pdf("sparklines\_base\_epanetReader.pdf", height=6, width=10)

plotSparklineTable(d, row.var = 'variable', col.vars = 'value')

dev.off()



*Sparklines in lattice*

You have much better control over the location and size of sparklines when you use lattice. The only problem are right-side labels for which you have to use grid library in order to ‘hack’ the view parameters with functions pushViewport() and popViewport().

**library**(lattice)

**library**(latticeExtra)

**library**(grid)

**library**(reshape)

**library**(RCurl)

dd <- read.csv(text = getURL("https://gist.githubusercontent.com/GeekOnAcid/da022affd36310c96cd4/raw/9c2ac2b033979fcf14a8d9b2e3e390a4bcc6f0e3/us\_nr\_of\_crimes\_1960\_2014.csv"))

d <- melt(dd, id="Year")

names(d)[1] <- "time"

pdf("sparklines\_lattice.pdf", height=10, width=8)

xyplot(value~time | variable, d, xlab="", ylab="", strip=F, lwd=0.7, col=1, type="l",

layout=c(1,length(unique(d$variable))), between = list(y = 1),

scales=list(y=list(at=NULL, relation="free"), x=list(fontfamily="serif")),

par.settings = list(axis.line = list(col = "transparent"),

layout.widths=list(right.padding=20, left.padding=-5)),

panel = **function**(x, y, **...**) {

panel.xyplot(x, y, **...**)

pushViewport(viewport(xscale=current.viewport()$xscale-5,

yscale=current.viewport()$yscale, clip="off"))

panel.text(x=tail(x,n=1), y=tail(y,n=1), labels=levels(d$variable)[panel.number()],

fontfamily="serif", pos=4)

popViewport()

panel.text(x=x[which.max(y)], y=max(y), labels=round(max(y),0), cex=0.8,

fontfamily="serif",adj=c(0.5,2.5))

panel.text(x=x[which.min(y)], y=min(y), labels=round(min(y),0), cex=0.8,

fontfamily="serif",adj=c(0.5,-1.5))

panel.text(x=tail(x,n=1), y=tail(y,n=1), labels=round(tail(y,n=1),0), cex=0.8,

fontfamily="serif", pos=4)

panel.points(x[which.max(y)], max(y), pch=16, cex=1)

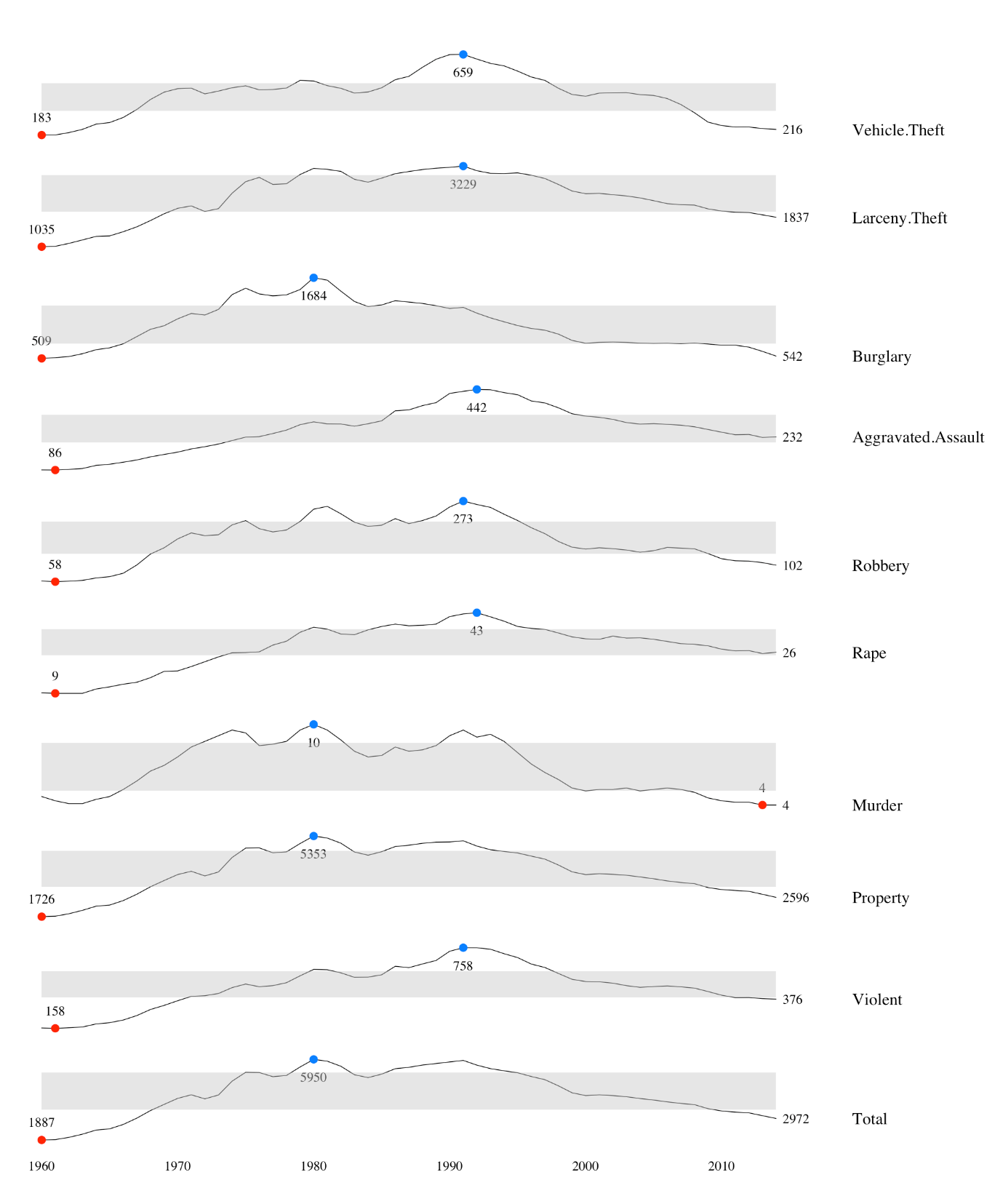
panel.points(x[which.min(y)], min(y), pch=16, cex=1, col="red")

panel.rect(min(x), quantile(y, 0.25), max(x), quantile(y, 0.75),

col = "grey", border = "transparent", alpha = 0.4)

})

dev.off()



*Sparklines in ggplot2*

**library**(ggplot2)

**library**(ggthemes)

**library**(dplyr)

**library**(reshape)

**library**(RCurl)

dd <- read.csv(text = getURL("https://gist.githubusercontent.com/GeekOnAcid/da022affd36310c96cd4/raw/9c2ac2b033979fcf14a8d9b2e3e390a4bcc6f0e3/us\_nr\_of\_crimes\_1960\_2014.csv"))

d <- melt(dd, id="Year")

names(d) <- c("Year","Crime.Type","Crime.Rate")

d$Crime.Rate <- round(d$Crime.Rate,0)

mins <- group\_by(d, Crime.Type) %>% slice(which.min(Crime.Rate))

maxs <- group\_by(d, Crime.Type) %>% slice(which.max(Crime.Rate))

ends <- group\_by(d, Crime.Type) %>% filter(Year == max(Year))

quarts <- d %>% group\_by(Crime.Type) %>%

summarize(quart1 = quantile(Crime.Rate, 0.25),

quart2 = quantile(Crime.Rate, 0.75)) %>%

right\_join(d)

pdf("sparklines\_ggplot.pdf", height=10, width=8)

ggplot(d, aes(x=Year, y=Crime.Rate)) +

facet\_grid(Crime.Type ~ ., scales = "free\_y") +

geom\_ribbon(data = quarts, aes(ymin = quart1, max = quart2), fill = 'grey90') +

geom\_line(size=0.3) +

geom\_point(data = mins, col = 'red') +

geom\_point(data = maxs, col = 'blue') +

geom\_text(data = mins, aes(label = Crime.Rate), vjust = -1) +

geom\_text(data = maxs, aes(label = Crime.Rate), vjust = 2.5) +

geom\_text(data = ends, aes(label = Crime.Rate), hjust = 0, nudge\_x = 1) +

geom\_text(data = ends, aes(label = Crime.Type), hjust = 0, nudge\_x = 5) +

expand\_limits(x = max(d$Year) + (0.25 \* (max(d$Year) - min(d$Year)))) +

scale\_x\_continuous(breaks = seq(1960, 2010, 10)) +

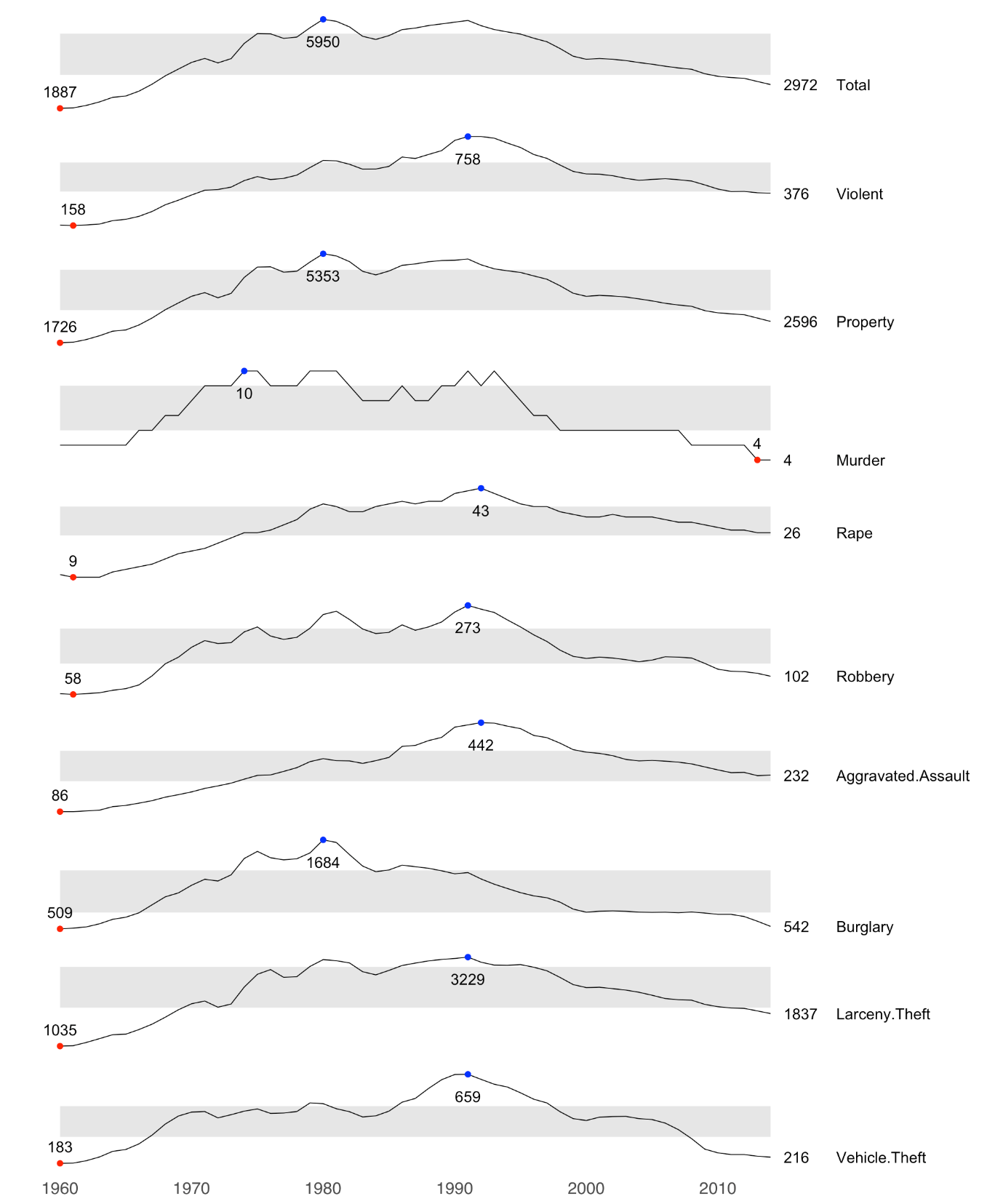
scale\_y\_continuous(expand = c(0.1, 0)) +

theme\_tufte(base\_size = 15, base\_family = "Helvetica") +

theme(axis.title=element\_blank(), axis.text.y = element\_blank(),

axis.ticks = element\_blank(), strip.text = element\_blank())

dev.off()



*Stem-and-leaf display*

Edward Tufte, *The Visual Display of Quantitative Information* (Cheshire, 1983), p. 140 Based on John Tukey, *Exploratory Data Analysis* (Addison-Wesley, 1970).

Stem-and-leaf display is not exactly a ‘Tuftesque’ solution as it invented in the beginning of 20 century but was only popularised in 1980s by John Tukey. A stem-and-leaf display is a display for presenting quantitative data in a graphical format, similar to a histogram, to assist in visualizing the shape of a distribution. Stem-and-leaf plot is the only visualisation in this collection thats printed in the console in *R* rather than being processed with any graphical system.

*Stem-and-leaf display in console with base graphics*

Data shows the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

stem(faithful$eruptions)

##

## The decimal point is 1 digit(s) to the left of the |

##

## 16 | 070355555588

## 18 | 000022233333335577777777888822335777888

## 20 | 00002223378800035778

## 22 | 0002335578023578

## 24 | 00228

## 26 | 23

## 28 | 080

## 30 | 7

## 32 | 2337

## 34 | 250077

## 36 | 0000823577

## 38 | 2333335582225577

## 40 | 0000003357788888002233555577778

## 42 | 03335555778800233333555577778

## 44 | 02222335557780000000023333357778888

## 46 | 0000233357700000023578

## 48 | 00000022335800333

## 50 | 0370

*Stem-and-leaf display in console with CarletonStats*

Data set taken from package *MASS*. It shows risk factors associated with low infant birth weight. The data were collected at Baystate Medical Center, Springfield, Mass during 1986.

The stemPlot function expands the basic stem plot by accepting a factor variable as a second argument to create stem plots for each of the levels.

**library**(CarletonStats)

**library**(MASS)

stemPlot(birthwt$bwt, birthwt$smoke, varname="infant birth weight (in grams)",

grpvarname="whether mother smoked during pregnancy (1) or not (0)")

##

## \*\*\*Stem and Leaf plot for infant birth weight (in grams) \*\*\*

## Grouped by levels of whether mother smoked during pregnancy (1) or not (0)

##

## 0

## :

## The decimal point is 2 digit(s) to the right of the |

##

## 10 | 2

## 12 | 3

## 14 | 799

## 16 | 03

## 18 | 9037

## 20 | 66809

## 22 | 4480358

## 24 | 14450025

## 26 | 24423558

## 28 | 144468822288

## 30 | 6668990088

## 32 | 003333377227

## 34 | 0266794479

## 36 | 011355037779

## 38 | 0366814478

## 40 | 00551577

## 42 |

## 44 | 9

## 46 |

## 48 | 9

##

##

## 1

## :

## The decimal point is 3 digit(s) to the right of the |

##

## 0 | 7

## 1 | 1

## 1 | 889999

## 2 | 11112223344444444

## 2 | 5555566677888899999

## 3 | 0000011111233333444

## 3 | 6666778999

## 4 | 2