MORE GGPLOT2 AND LATTICE CHARTS

[Adapted from Lanzetta's *R Data Visualization Recipes*]

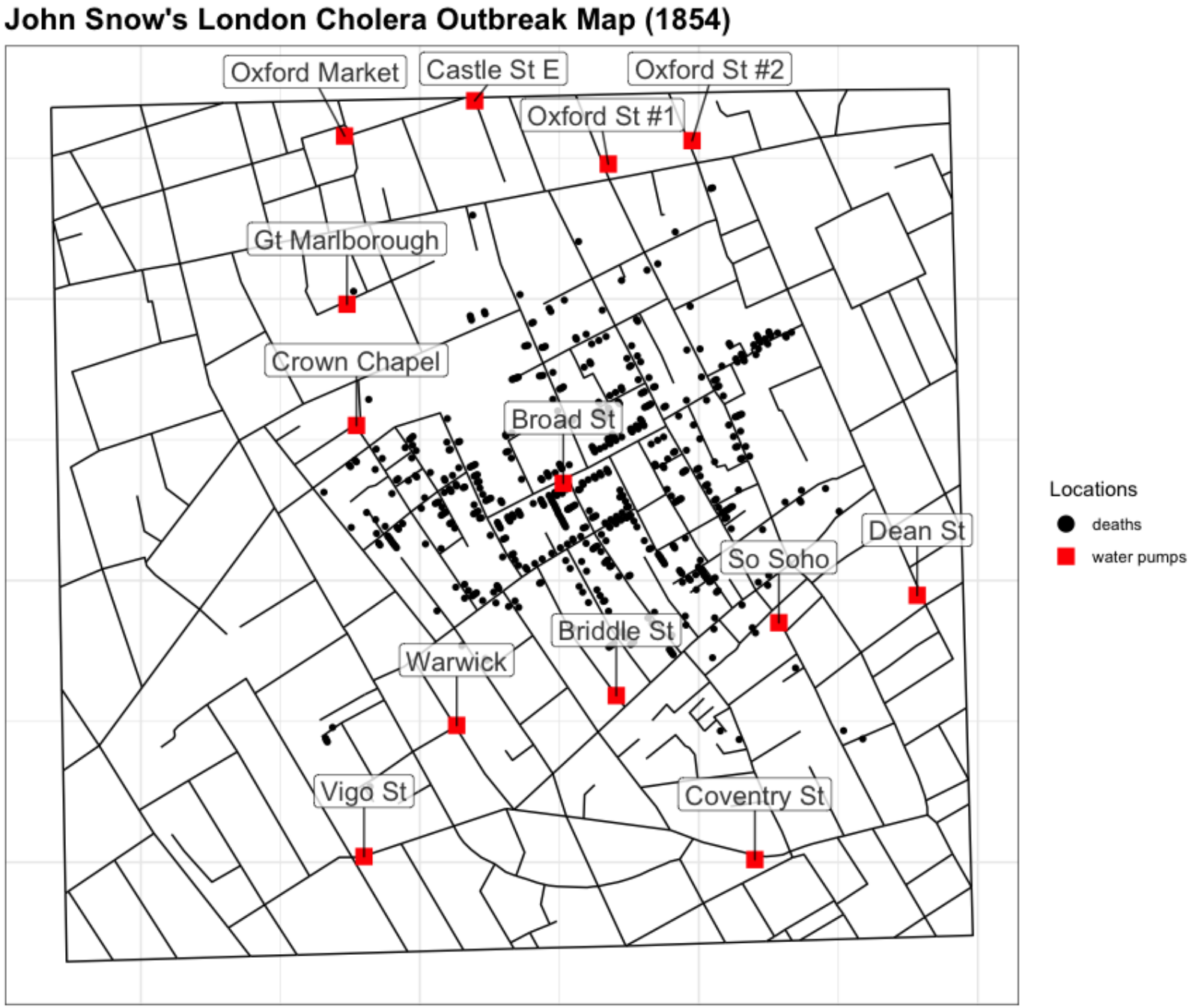
EXAMPLE: JOHN SNOW'S CHOLERA OUTBREAK MAP

We start with some code to re-create John Snow's **Cholera Outbreak Map:**

```
# getting the data and required  packages
> data(Snow.streets, Snow.deaths, Snow.pumps, package="HistData")
> library(ggplot2); library(ggrepel)


# street map, death locations, water pump locations
> ggplot(data=Snow.streets) + geom_path(aes(x=x, y=y, group=street)) +
     geom_point(data=Snow.deaths, aes(x=x, y=y, colour="black", shape="15")) +
     geom_point(data=Snow.pumps, aes(x=x, y=y, colour="red", shape="16"), size=4) +
   scale_colour_manual("Locations", values=c("black","red"), labels=c("deaths","water pumps")) +
     scale_shape_manual("Locations", values=c(16, 15), labels=c("deaths","water pumps")) +
     geom_label_repel(data=Snow.pumps, aes(x=x, y=y, label=label), colour="black", size=5,
         vjust=-1.5, alpha=0.8) +
     ggtitle("John Snow's London Cholera Outbreak Map (1854)")  + theme_bw() +
     theme(plot.title = element_text(size=16, face="bold"),
     axis.title.x=element_blank(), axis.text.x=element_blank(), axis.ticks.x=element_blank(),
     axis.title.y=element_blank(), axis.text.y=element_blank(), axis.ticks.y=element_blank())
```

[Adapted from Lanzetta's *R Data Visualization Recipes*]



[Follows M. Friendly's excellent tutorial: http://euclid.psych.yorku.ca/www/psy6135/tutorials/Minard.html]

EXAMPLE: MINARD'S MARCH TO MOSCOW

Now, some code to re-create Minard's **March to Moscow** chart.

```
# loading the packages and getting the data - take some time to explore it
> data(Minard.troops, Minard.cities, Minard.temp, package="HistData")
> library(ggplot2); library(scales); library(grid); library(gridExtra); library(dplyr),
library(ggrepel)


# troops/cities upper chart
> plot.troops.cities <- Minard.troops %>% ggplot(aes(long, lat)) +
      geom_path(aes(size = survivors, colour = direction, group = group),
                lineend="round") +
      scale_size("Survivors", range = c(0,20), breaks= c(1, 2, 3) * 10^5,
                labels=scales::comma(c(1, 2, 3) * 10^5 )) +
      scale_color_manual("Direction", values = c("#E80000", "#1F1A1B"),
                labels=c("Advance", "Retreat")) +
      ylim(53.8,56.0) + coord_cartesian(xlim = c(24, 38)) +
      labs(x = NULL, y = NULL) + theme_bw() + guides(color = FALSE, size = FALSE) +
      geom_point(data = Minard.cities, size=10, pch=22, color="black", fill="gray") +
      geom_label_repel(data = Minard.cities, aes(label=city), size=5, vjust=-1.5)
```

[Follows M. Friendly's excellent tutorial: http://euclid.psych.yorku.ca/www/psy6135/tutorials/Minard.html]

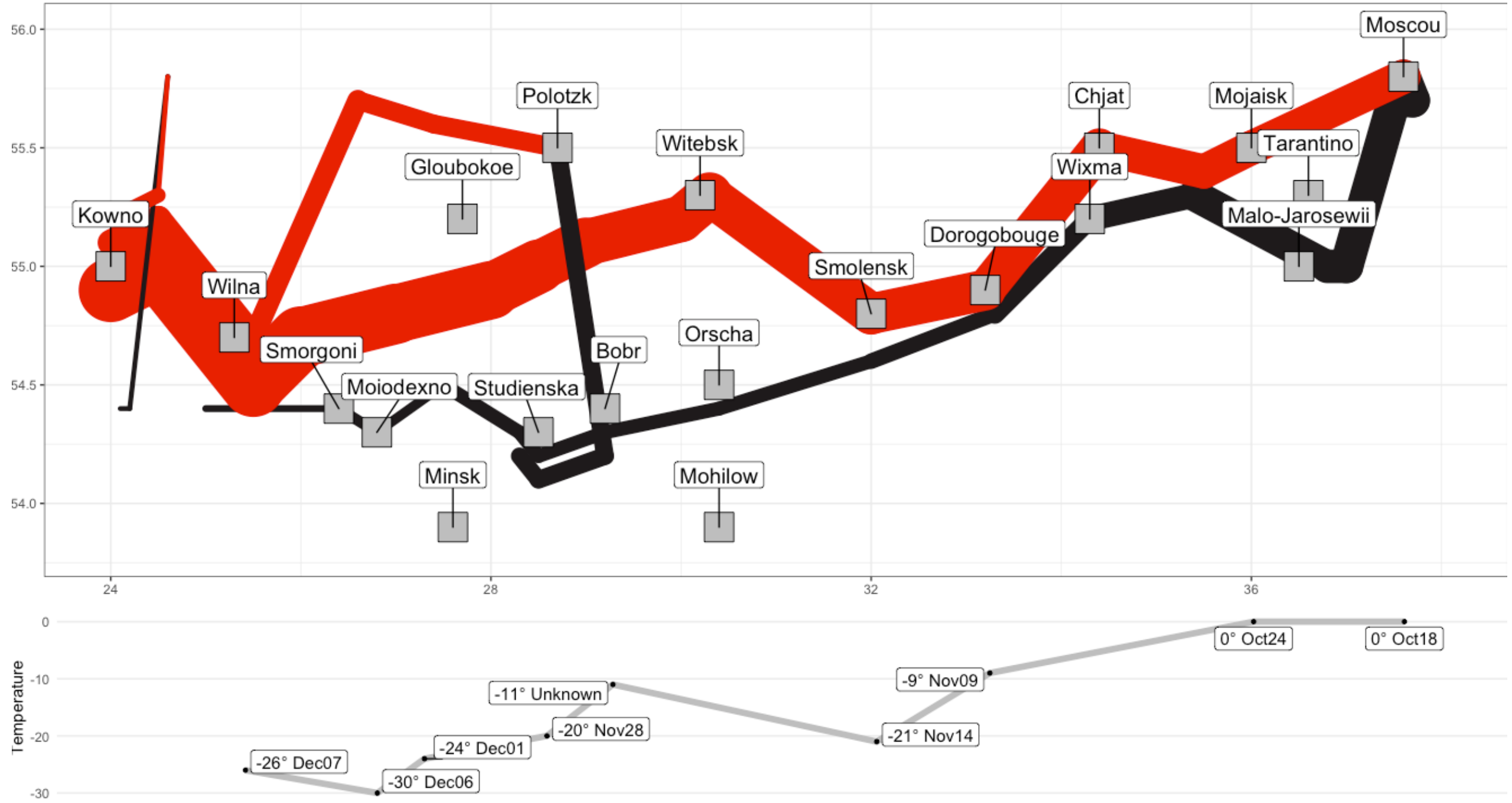
EXAMPLE: MINARD'S MARCH TO MOSCOW

```
# replacing a missing value in the temperature data
> Minard.temp$date = factor(Minard.temp$date, levels=c(levels(Minard.temp$date), "Unknown"))
> Minard.temp$date[is.na(Minard.temp$date)] <- "Unknown"


# the temperature lower chart
> plot.temp <- Minard.temp %>% mutate(label = paste0(temp, "° ", date)) %>%
    ggplot(aes(long, temp)) + geom_path(color="grey", size=2, group=1) +
    geom_point(size=1) + geom_label_repel(aes(label=label), size=4) +
    coord_cartesian(xlim = c(24, 38)) + labs(x=NULL, y="Temperature") +
    theme_bw() + theme(panel.grid.major.x = element_blank(),
                       panel.grid.minor.x = element_blank(),
                       panel.grid.minor.y = element_blank(),
                       axis.text.x = element_blank(), axis.ticks = element_blank(),
                       panel.border = element_blank())


# combining both charts
> grid.arrange(plot.troops.cities, plot.temp, nrow=2, heights=c(3.5, 1.2))
> grid.rect(width = .99, height = .99, gp = gpar(lwd = 2, col = "gray",
                       fill = NA))library(tidyverse) # for data manipulation
```

[Follows M. Friendly's excellent tutorial: http://euclid.psych.yorku.ca/www/psy6135/tutorials/Minard.html]



EXAMPLE: GAPMINDER'S HEALTH AND WEALTH OF NATIONS

Now, some code to produce **Gapminder**-style charts.

```
> library(tidyverse)    # for data manipulation
> library(ggplot2)      # for ggplot2 charts
> library(dslabs)       # where the gapminder dataset resides, run summary()
> library(wesanderson)  # for fancy colour palettes

> yr = 2009             # year of interest
> chart_title <- paste("Health & Wealth of Nations \nGampinder (",yr,")",sep="")
                        # chart title


# sort countries by inverse population, so small countries aren't hidden
> gapminder <- gapminder[with(gapminder, order(year, -1 * population)), ]


# select which countries will have their names labelled (otherwise, too busy)
> num.countries = 20
```

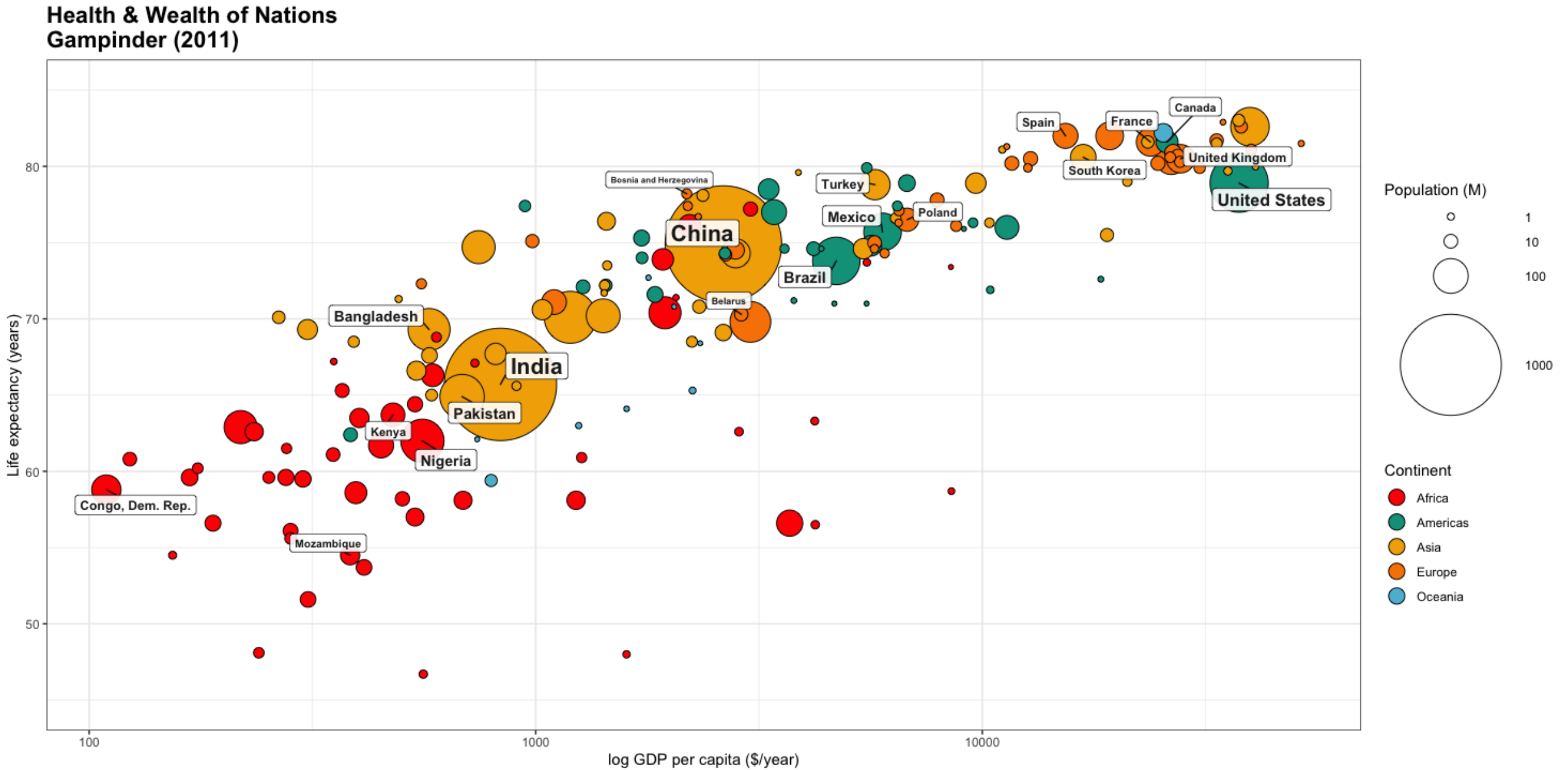
EXAMPLE: GAPMINDER'S HEALTH AND WEALTH OF NATIONS

```
# start by only keeping observations for the year of interest
> filtered.gapminder = gapminder %>%
    filter(year==yr) %>%
    mutate(pop_m = population/1e6, gdppc = gdp/population)

# randomly select which countries get labeled on the chart, proportional to pop_m
> weights = filtered.gapminder$pop_m/sum(filtered.gapminder$pop_m)
> p = sample(nrow(filtered.gapminder), num.countries, prob = weights)
> filtered.gapminder$country.display <- ifelse(
                ifelse(1:185 %in% p, TRUE,FALSE),
                   as.character(filtered.gapminder$country),"")
```

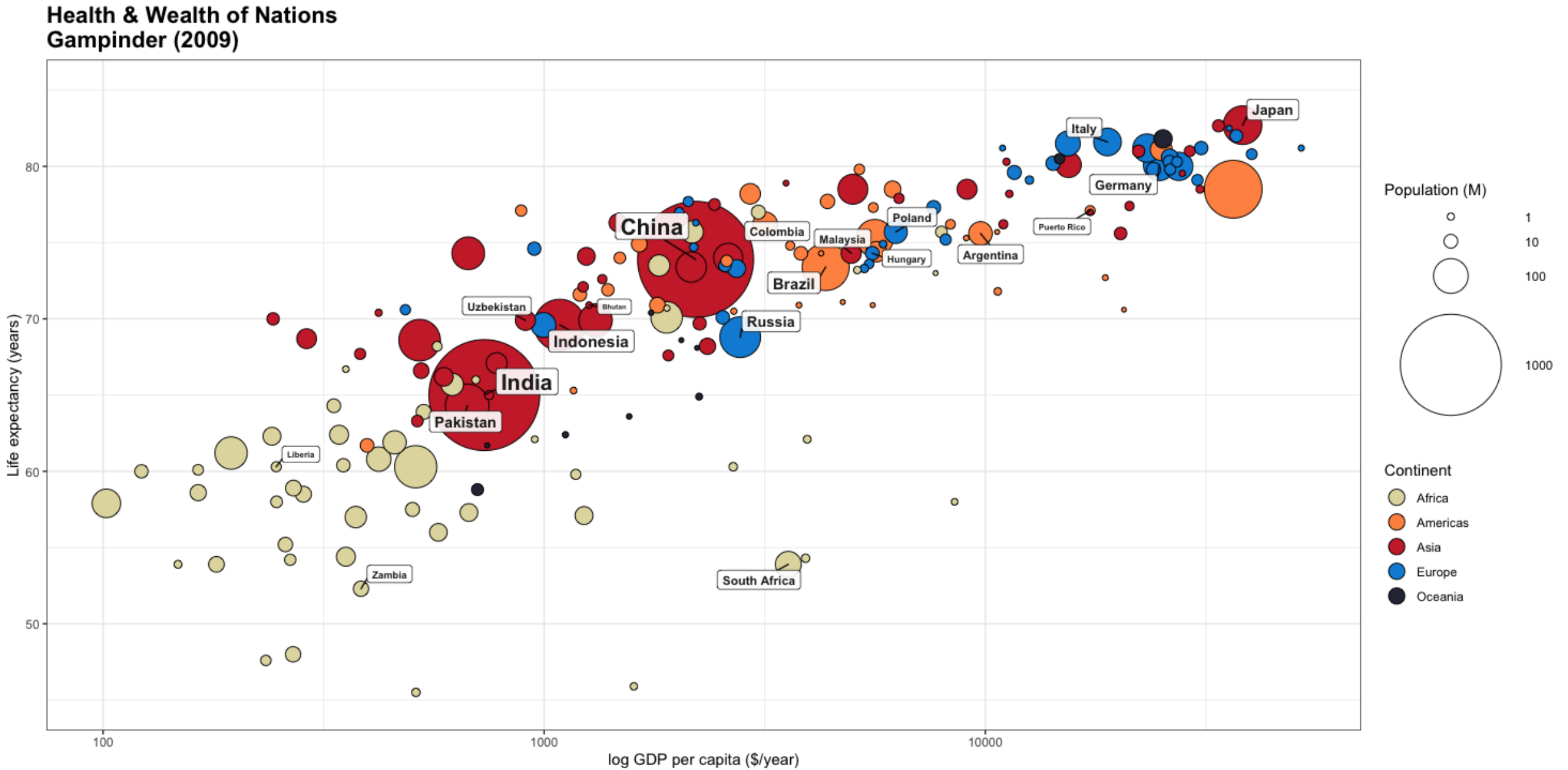
EXAMPLE: GAPMINDER'S HEALTH AND WEALTH OF NATIONS

```
> filtered.gapminder %>%
    ggplot(aes(x=gdppc, y=life_expectancy, size=pop_m)) +
    geom_pont(aes(fill=continent), pch=21) +
    scale_fill_manual(values=wes_palette(n=5, name="Darjeeling1")) +
    scale_x_log10() +
    geom_label_repel(aes(label=country.display, size=sqrt(pop_m/pi)),
              alpha=0.9, fontface="bold", min.segment.length = unit(0, 'lines'),
              show.legend=FALSE) +
    ggtitle(chart_title) +
    theme(plot.title = element_text(size=14, face="bold")) +
    xlab('log GDP per capita ($/year)') + ylab('Life expectancy (years)') +
    ylim(45,85) + scale_size_continuous(range=c(1,40), breaks = c(1,10,100,1000),
                        limits = c(0, 1500), labels = c("1","10","100","1000")) +
    guides(fill = guide_legend(override.aes = list(size = 5))) +
    labs(fill="Continent", size="Population (M)") + theme_bw() +
    theme(plot.title = element_text(size=16, face="bold"))
```



EXAMPLE: GAPMINDER'S HEALTH AND WEALTH OF NATIONS

```
> filtered.gapminder %>%
   ggplot(aes(x=gdppc, y=life_expectancy, size=pop_m)) +
   geom_pont(aes(fill=continent), pch=21) +
   scale_fill_manual(values=c("#E1DAAE","#FF934F","#CC2D35","#058ED9","#2D3142")) +
   scale_x_log10() +
   geom_label_repel(aes(label=country.display, size=sqrt(pop_m/pi)),
                    alpha=0.9, fontface="bold", min.segment.length = unit(0, 'lines'),
                    show.legend=FALSE) +
   ggtitle(chart_title) +
   theme(plot.title = element_text(size=14, face="bold")) +
   xlab('log GDP per capita ($/year)') + ylab('Life expectancy (years)') +
   ylim(45,85) + scale_size_continuous(range=c(1,40), breaks = c(1,10,100,1000),
                        limits = c(0, 1500), labels = c("1","10","100","1000")) +
   guides(fill = guide_legend(override.aes = list(size = 5))) +
   labs(fill="Continent", size="Population (M)") + theme_bw() +
   theme(plot.title = element_text(size=16, face="bold"))
```

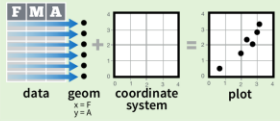


Data Visualization with ggplot2 Cheat Sheet

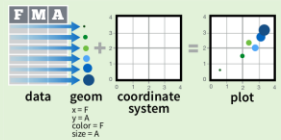
RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point"**)**

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, **aes(**x = cty, y = hwy**))**

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(
xend = long + delta_long,
yend = lat + delta_lat)**)**
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl"**)**
alpha, color, linetype, size

f + geom_smooth(model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map**) + expand_limits(**x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

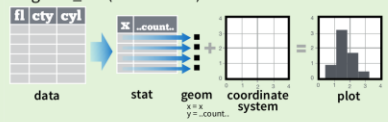
m + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

m + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. **stat_bin(geom="bar")** does the same as **geom_bar(stat="bin")**

stat function | layer specific mappings | variable created by transformation

i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100**)**

geom for layer | parameters for stat

1D distributions

a + stat_bin(binwidth = 1, origin = 10**)**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x"**)**
x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian"**)**
x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + stat_bin2d(bins = 30, drop = TRUE**)**
x, y, fill | ..count.., ..density..
f + stat_binhex(bins = 30**)**
x, y, fill | ..count.., ..density..
f + stat_density2d(contour = TRUE, n = 100**)**
x, y, color, size | ..level..

3 Variables

m + stat_contour(aes(z = z)**)**
x, y, z, order | ..level..
m+ stat_spoke(aes(radius= z, angle = z)**)**
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..
m + stat_summary2d(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

Comparisons

g + stat_boxplot(coef = 1.5**)**
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area"**)**
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

f + stat_ecdf(n = 40**)**
x, y | ..x.., ..y..
f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq"**)**
x, y | ..quantile.., ..x.., ..y..
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95**)**
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5)**)**
x | ..y..
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5)**)**
sample, x, y | ..x.., ..y..
f + stat_sum()
x, y, size | ..size..
f + stat_summary(fun.data = "mean_cl_boot"**)**
f + stat_unique()

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.


n <- b + geom_bar(aes(fill = fl)**)**
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R")**)**

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()**)** - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")**)** - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete

n <- b + geom_bar(aes(fill = fl)**)**

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
library(RColorBrewer)
display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red"**)**

Continuous

o <- a + geom_dotplot(aes(fill = ..x..)**)**

o + scale_fill_gradient(low = "red", high = "yellow"**)**

o + scale_fill_gradient2(low = "red", hight = "blue", mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colours = terrain.colors(6)**)**
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

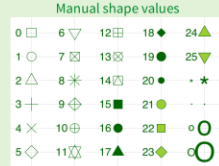
Shape scales

p <- f + geom_point(aes(shape = fl)**)**

p + scale_shape(solid = FALSE**)**

p + scale_shape_manual(values = c(3:7)**)**
Shape values shown in chart on right

Manual shape values



Size scales

q <- f + geom_point(aes(size = cyl)**)**

q + scale_size_area(max = 6**)**
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + geom_bar()

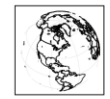
r + coord_cartesian(xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2**)**
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1 **)**
theta, start, direction
Polar coordinates

r + coord_trans(xtrans = "sqrt"**)**
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

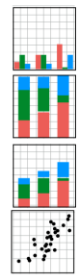


z + coord_map(projection = "ortho", orientation=c(41, -74, 0)**)**
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))



s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

s + geom_bar(position = "stack")
Stack elements on top of one another

f + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_classic()
White background no gridlines

r + theme_grey()
Grey background (default theme)

r + theme_minimal()
Minimal theme

ggthemes - Package with additional ggplot2 themes

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(y ~ x, scales = "free")
x and y axis limits adjust to individual facets
- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)**))**

α^c	α^d	α^e	α^p	α^r

t + facet_grid(. ~ fl, labeller = label_parsed)

c	d	e	p	r

Labels

t + ggtitle("New Plot Title"**)**
Add a main title above the plot
t + xlab("New X label"**)**
Change the label on the X axis
t + ylab("New Y label"**)**
Change the label on the Y axis
t + labs(title =" New title", x = "New x", y = "New y"**)**
All of the above

Use scale functions to update legend labels

Legends

t + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right"
t + guides(color = "none"**)**
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C")**)**
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)**

With clipping (removes unseen data points)

t + xlim(0, 100**) + ylim(**10, 20**)**
t + scale_x_continuous(limits = c(0, 100)**) + scale_y_continuous(**limits = c(0, 100)**)**

EXAMPLE: TITANIC DATASET

We will explore a dataset related to the Titanic. It can be found in a `txt` format. We use `read.table()` to download the dataset to a data frame.

```
> titanic.url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
> titanic <- read.table(titanic.url,  sep=",", header=TRUE)

> str(titanic)
## 'data.frame':    1313 obs. of  11 variables:
##  $ row.names: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ pclass   : Factor w/ 3 levels "1st","2nd","3rd": 1 1 1 1 1 1 1 1 1 1 ...
##  $ survived : int  1 0 0 0 1 1 1 0 1 0 ...
##  $ name     : Factor w/ 1310 levels "Abbing, Mr Anthony",..: 22 25 26 ...
##  $ age      : num  29 2 30 25 0.917 ...
##  $ embarked : Factor w/ 4 levels "","Cherbourg",..: 4 4 4 4 4 4 4 4 4 2 ...
##  $ home.dest: Factor w/ 372 levels "","?Havana, Cuba",..: 312 233 233 ...
##  $ room     : Factor w/ 54 levels "","2131","A-11",..: 13 42 42 42 41 ...
##  $ ticket   : Factor w/ 42 levels "","  ","111361 L57 19s 7d",..: 31 1 ...
##  $ boat     : Factor w/ 100 levels "","(101)","(103)",..: 88 1 13 1 ...
##  $ sex      : Factor w/ 2 levels "female","male": 1 1 2 1 2 2 1 2 1 2 ...
```

EXAMPLE: TITANIC DATASET

It contains 131 observations and 11 variables (the first of which is a row identifier). We see that most variables are **factors** (categorical variables).

Based on the name of some of the variables, we might expect some of them to be character variables instead.

```
> # head(titanic)   # uncomment to see first rows
```

We can remedy the situation by specifying that the first column corresponds to the row number, and by making sure that strings do not get imported as factors.

```
# stringsAsFactors=FALSE to keep strings as strings.
# row.names=1 tells read.table that column 1 contains the row names.
> titanic <- read.table( file=titanic.url, sep=",", header=TRUE,
        stringsAsFactors=FALSE, row.names=1)
```

EXAMPLE: TITANIC DATASET

We do not need to call `str()` to see the type for each of the variables; we can use `sapply()` instead.

```
> sapply(titanic, class)
##      pclass    survived        name         age    embarked   home.dest
## "character"   "integer" "character"   "numeric" "character" "character"
##        room      ticket        boat         sex
## "character" "character" "character" "character"
```

Now the problem is that some of the variables that should have been (ordered) factors or logicals are character strings.

EXAMPLE: TITANIC DATASET

We will fix that issue by using the `transform()` function.

```
> titanic <- transform(titanic,
                       pclass = ordered(pclass, levels=c("3rd", "2nd", "1st")),
                       survived = as.logical(survived),
                       embarked = as.factor(embarked),
                       sex = as.factor(sex))
```

The dataset can be now be summarized using the `summary()` function.

```
> summary(titanic)
```

EXAMPLE: TITANIC DATASET

```
##  pclass     survived          name                 age
##  3rd:711   Mode :logical   Length:1313        Min.    : 0.1667
##  2nd:280   FALSE:864       Class :character   1st Qu.:21.0000
##  1st:322   TRUE :449       Mode  :character   Median :30.0000
##                                               Mean   :31.1942
##                                               3rd Qu.:41.0000
##                                               Max.   :71.0000
##                                               NA's   :680

##          embarked    home.dest            room
##              :492   Length:1313        Length:1313
##  Cherbourg   :203   Class :character   Class :character
##  Queenstown : 45    Mode  :character   Mode  :character
##  Southampton:573

##     ticket             boat              sex
##  Length:1313        Length:1313        female:463
##  Class :character   Class :character   male  :850
##  Mode  :character   Mode  :character
```

EXAMPLE: TITANIC DATASET

We notice, among other things, that a number of `age` observations are missing, and that a fair number of passengers do not have a port of embarcation.

What can we say about this dataset as a whole?

First, let's build a **contingency table** of survival by passenger class.

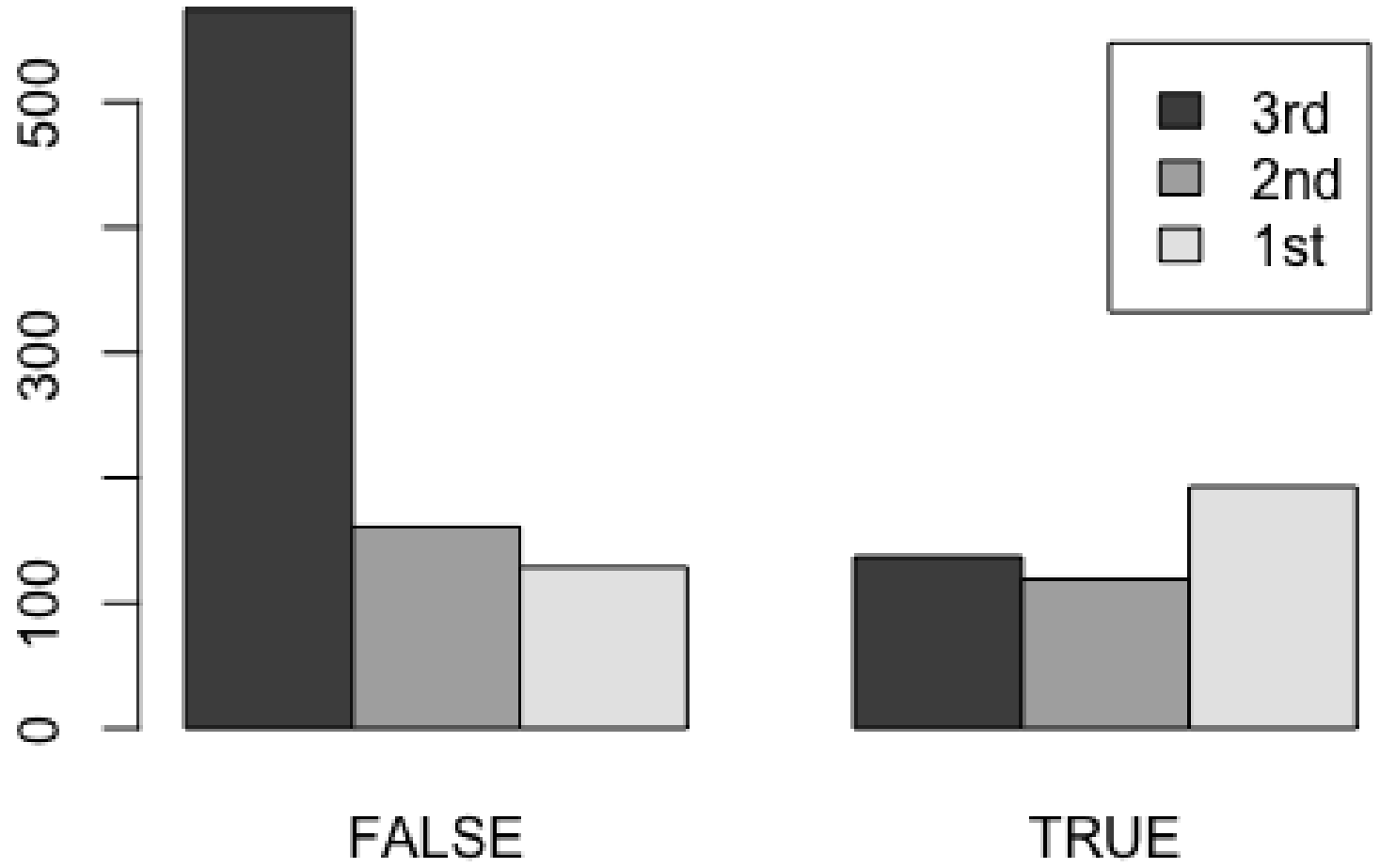
```
> (pclass.survived <- with(titanic, table(pclass, survived)))
##        survived
## pclass FALSE TRUE
##    3rd   574  137
##    2nd   161  119
##    1st   129  193
```

EXAMPLE: TITANIC DATASET

It certainly seems as though the passenger class was linked to survival (at least, at first glance).

A barplot can help illustrate the relationship.

```
> barplot(pclass.survived, beside=TRUE, legend.text=levels(titanic$pclass))
```

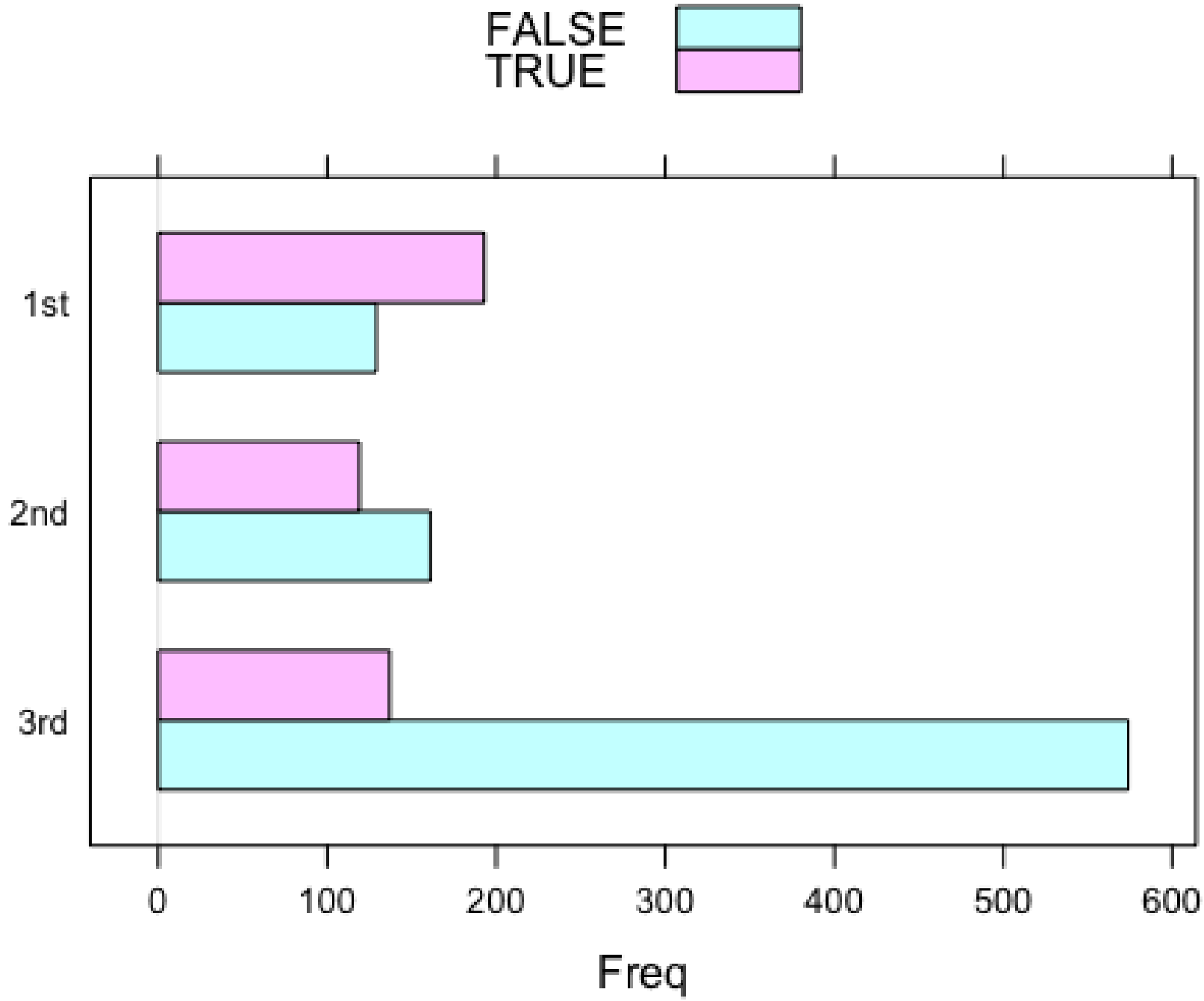

0
100
300
500
3rd
2nd
1st
FALSE
TRUE

EXAMPLE: TITANIC DATASET

Splitting along the survived (TRUE) / did not survive (FALSE) axis can muddle the situation to some extent: of course there were more 3rd class passenger who didn't survive – there were more 3rd class passengers, period.

Compare with the following chart.

```
> library(lattice)
> barchart(pclass.survived, stack=FALSE, auto.key=TRUE)
```

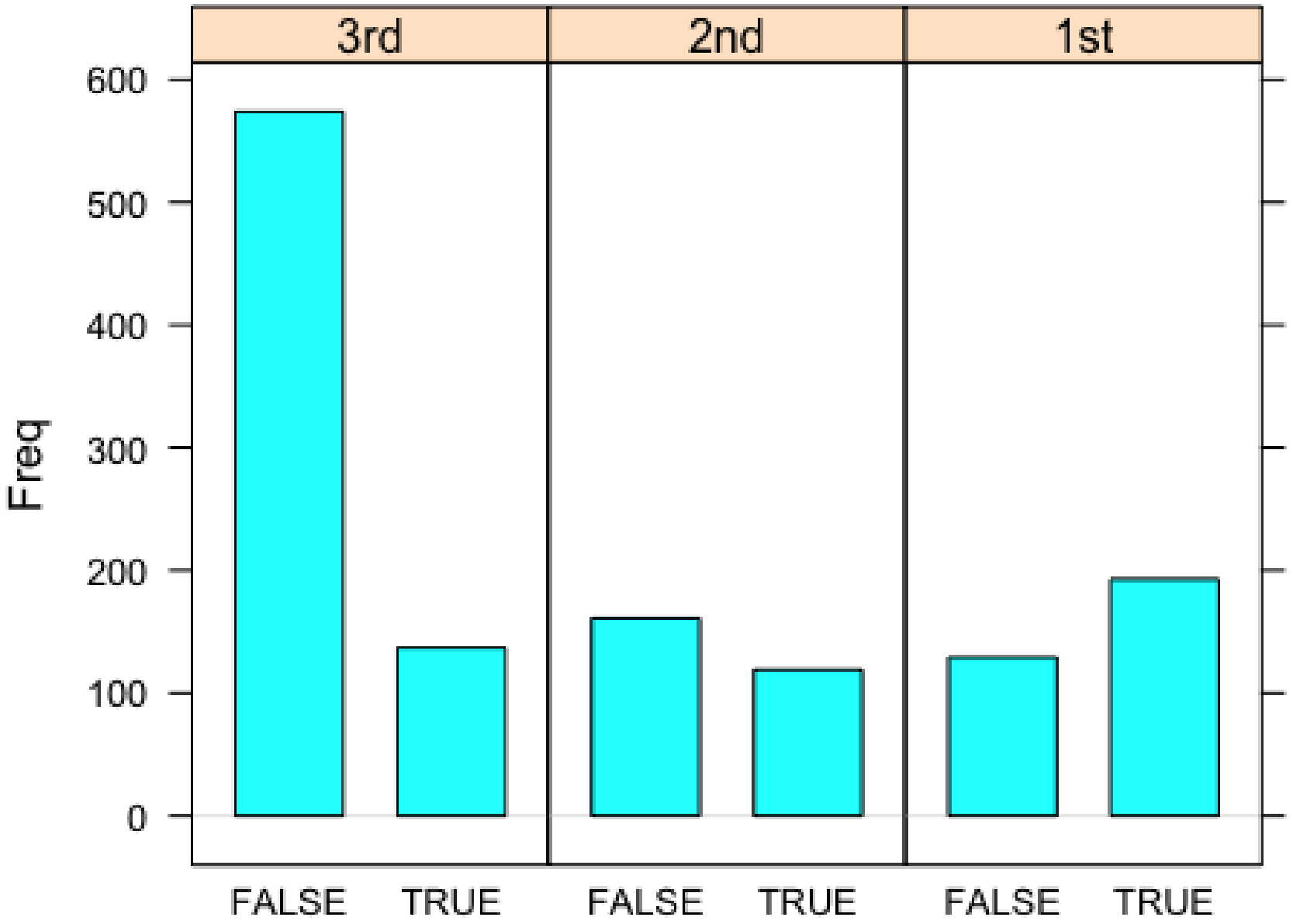

FALSE
TRUE
1st
2nd
3rd
0
100
200
300
400
500
600
Freq

EXAMPLE: TITANIC DATASET

The relative proportions of survivors per passenger class is striking.

Here is another way to display the same information.

```
> barchart(t(pclass.survived), groups=FALSE, layout=c(3,1), horizontal=FALSE)
```


3rd
2nd
1st
Freq
600
500
400
300
200
100
0
FALSE
TRUE
FALSE
TRUE
FALSE
TRUE

EXAMPLE: TITANIC DATASET

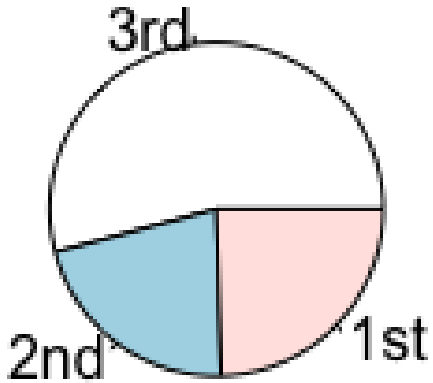
Visualization experts claim that the most efficient visualization dimensions for comparison of quantities are, in order:

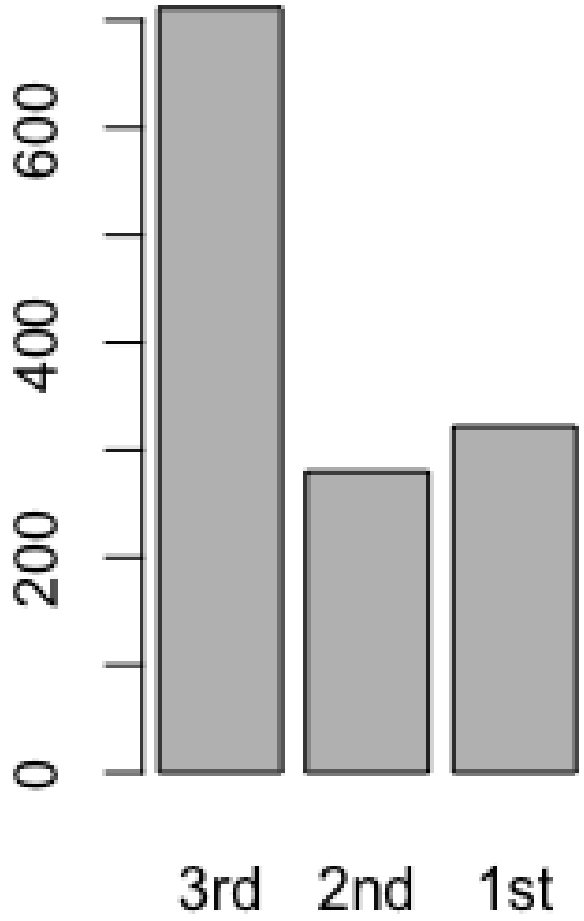
- position, length, slope, and angle

Let's see if that seem reasonable, by comparing how length and angles do the job.

```
> (pass.class <- table(titanic$pclass))
## 3rd 2nd 1st
## 711 280 322

> par(mfrow=c(1,2))    # two charts side by side
> pie(pass.class)      # angle
> barplot(pass.class) # length
```


3rd
2nd
1st


0
200
400
600
3rd
2nd
1st

EXAMPLE: TITANIC DATASET

What do you think? Is the much-maligned pie chart THAT bad?

Let's see what things look like when we combine 3 variables instead of 2.

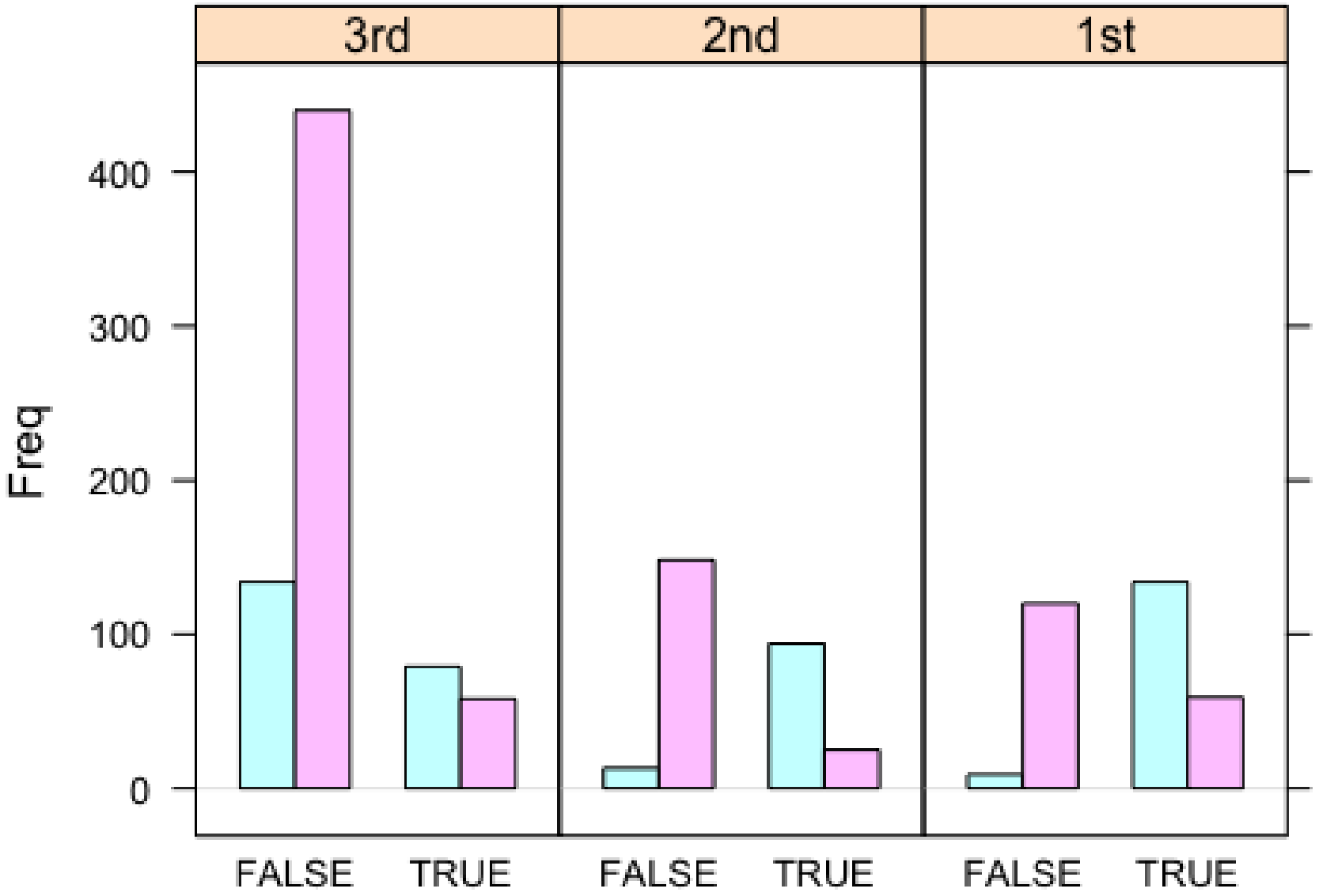
```
(sv.pc.sx <- with(titanic, table(survived, pclass, sex)))
## , , sex = female
##          pclass
## survived 3rd 2nd 1st
##    FALSE 134  13   9
##    TRUE   79  94 134

## , , sex = male
##          pclass
## survived 3rd 2nd 1st
##    FALSE 440 148 120
##    TRUE   58  25  59
```

The table of frequencies is difficult to read/interpret in a first pass.

But we can get a good sense of the underlying data distributions with a bar chart.

```
> barchart(sv.pc.sx, horizontal=FALSE, stack=FALSE,
    layout=c(3,1), auto.key=list(columns=2))
```

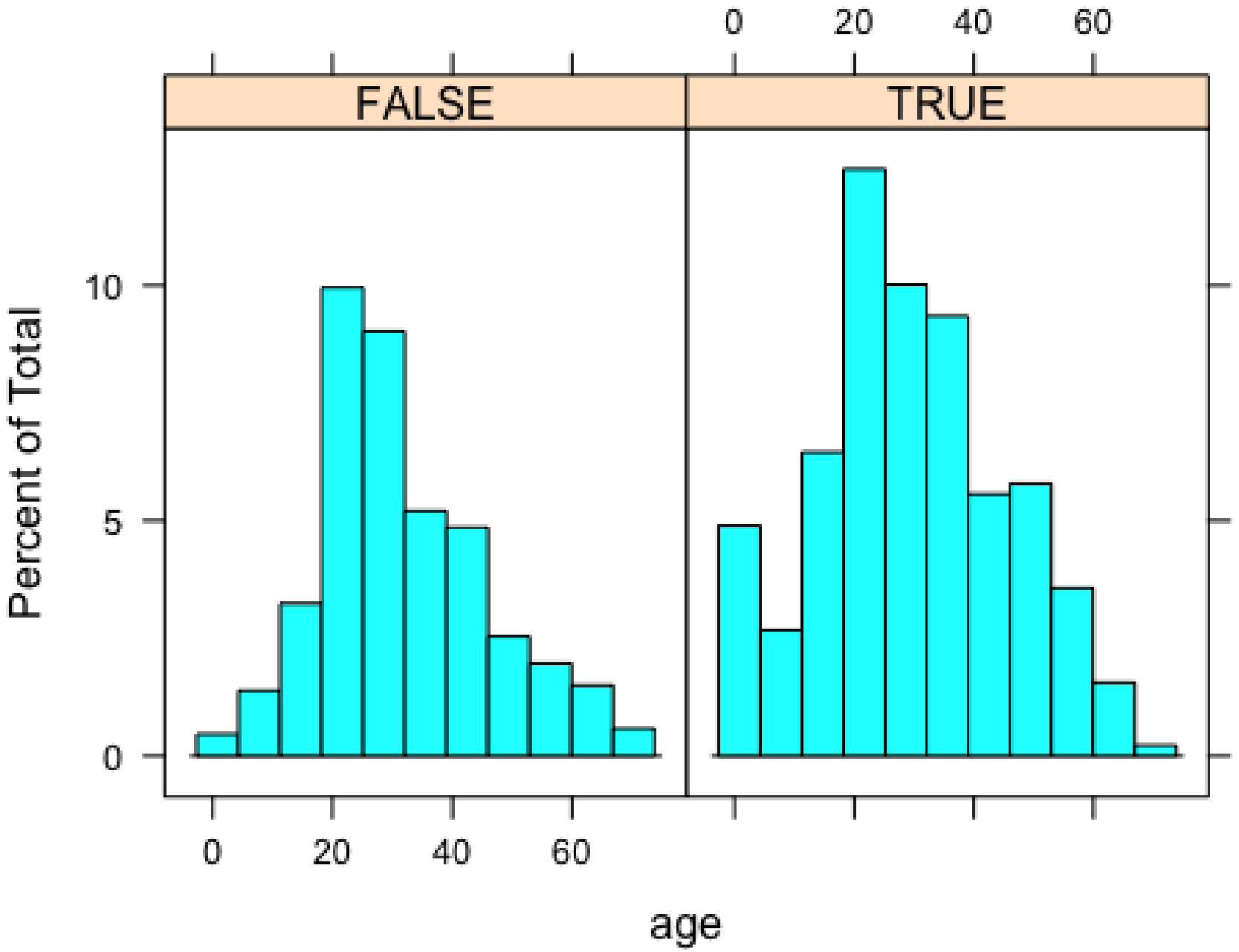

3rd
2nd
1st
Freq
400
300
200
100
0
FALSE
TRUE
FALSE
TRUE
FALSE
TRUE

EXAMPLE: TITANIC DATASET

Histograms, density plots, and dot plots can also provide an idea of the underlying distributions (implemented with `lattice`'s `histogram()`, `densityplot()`, and `dotplot()`)

Here are histograms of age by survival status:

```
> histogram(~age | survived, data=titanic)
```


FALSE
TRUE
0
20
40
60
Percent of Total
0
5
10
age

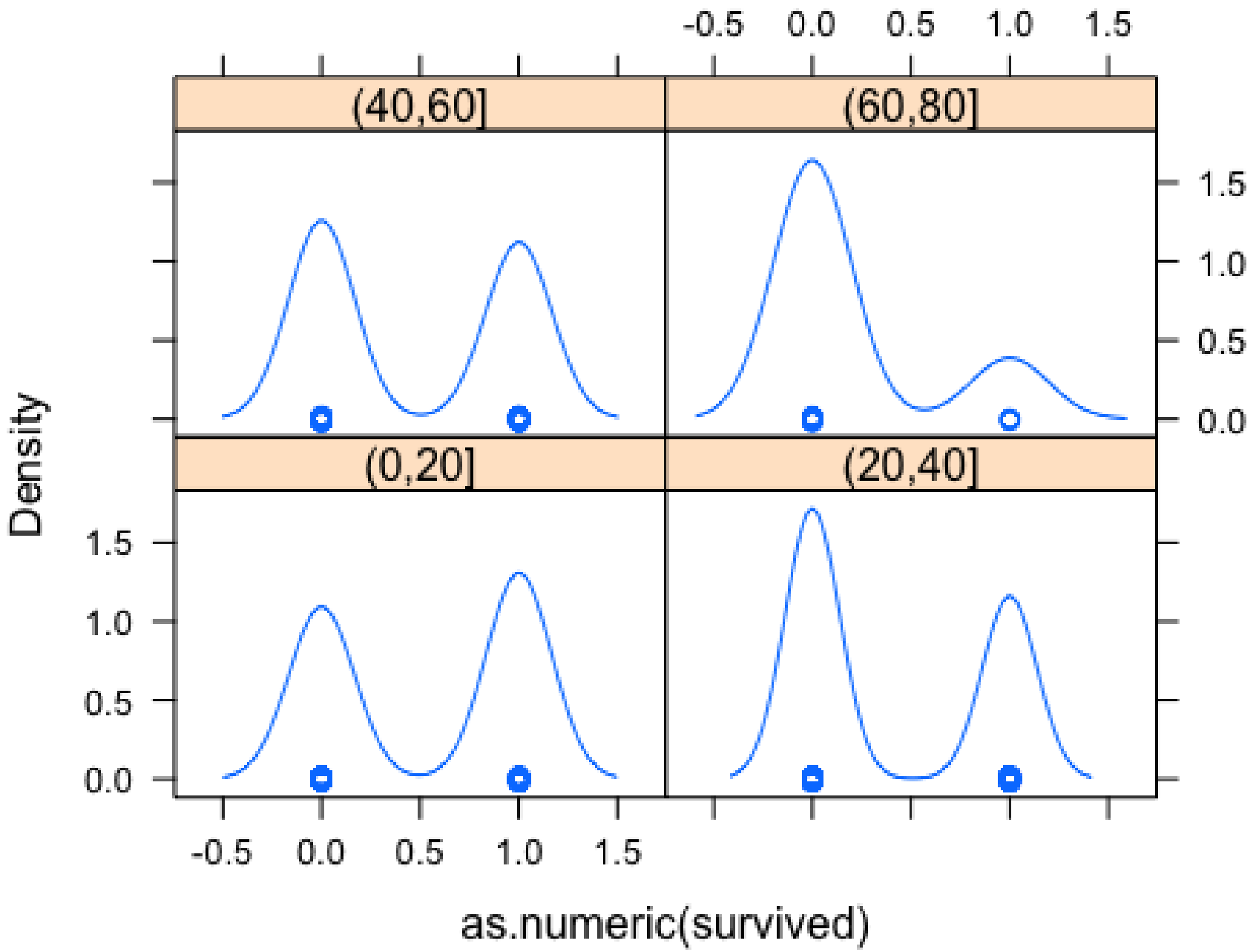
EXAMPLE: TITANIC DATASET

We might want to reverse this and get the distributions of survival statuses by age:

```
# start by creating age groups
> titanic$age.groups <- cut(titanic$age, breaks=seq(0,80,by=20))

> table(titanic$age.groups)
##
##   (0,20] (20,40] (40,60] (60,80]
##      145     327     140      21

> densityplot(~as.numeric(survived) | age.groups, data=titanic)
```

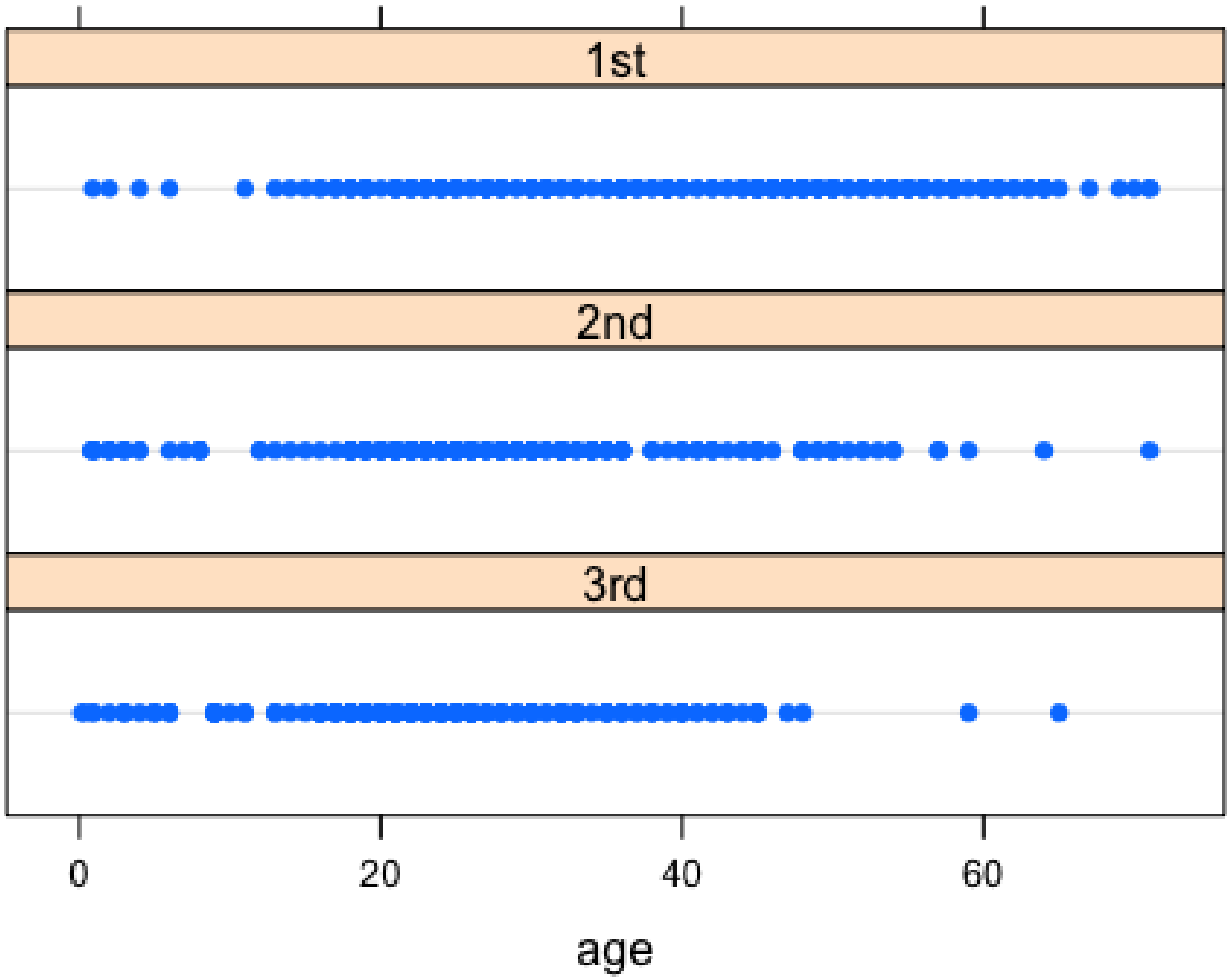

(40,60]
(60,80]
(0,20]
(20,40]
Density
as.numeric(survived)
-0.5
0.0
0.5
1.0
1.5
0.0
0.5
1.0
1.5

EXAMPLE: TITANIC DATASET

Is what's represented in this chart clear to you? What happens to the passengers for which no age was recorded?

We could also look at the distribution of age within each passenger class:

```
dotplot(~age | pclass, data=titanic, layout=c(1,3))
```


1st
2nd
3rd
0
20
40
60
age