# The mosaic package: helping students to think with data using R

*Randall Pruim, Daniel T Kaplan, and Nicholas J Horton*

*2015-12-29*

## Motivation

Many have argued that in order to make sense of the increasingly rich data that is available to them, students need additional facility to express statistical computations (e.g., (Nolan and Lang 2010; Ridgway 2015; Horton, Baumer, and Wickham 2015)). To be able to "think with data" (as coined by Diane Lambert of Google), students need tools for data management, exploratory analysis, visualization, and modeling. Yet many students enter statistics courses with little or no computational experience. Our students have demonstrated that it is feasible to integrate computing into our curricula early and often, in a way that provides students with success, confidence, and room to grow.

## A guiding principle: Less volume, more creativity

The `mosaic` (Pruim, Kaplan, and Horton 2015a) package originated in early attempts by each of the authors to ease new users into using R, primarily in the context of undergraduate statistics courses, and, in one case, also in calculus. One of the guiding principles behind the development of the `mosaic` package has been "Less volume, more creativity". Beginners are easily overwhelmed by the scope of R and its many packages. Often there are multiple ways to accomplish the same task, and authors of the many packages are not required to follow any particular style guidelines.

Early on in the development of `mosaic`, we decided to reduce the number of code templates that users would need to know to as few as possible, while still providing them with substantial power to be creative within the templates provided. A one-page list of commands that are more than sufficient for a first course, originally presented as part of a roundtable discussion at the Joint Statistics Meetings (Pruim 2011) now appears as a vignette in the package, along with some additional material on the less volume, more creativity approach.

## The formula template

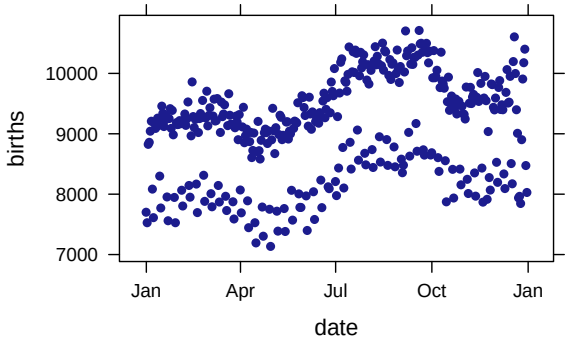
To successfully implement a "less volume, more creativity" approach, one must decide which tasks are most important to accomplish. We knew from the outset that this would include graphical and numerical summaries of data and various models and inference procedures. Because of this goal, our most important template makes use of a "formula interface" modeled after `lm()` and the plotting functions in `lattice` (Sarkar 2008).

We typically introduce the formula template in the context of exploring two variables as

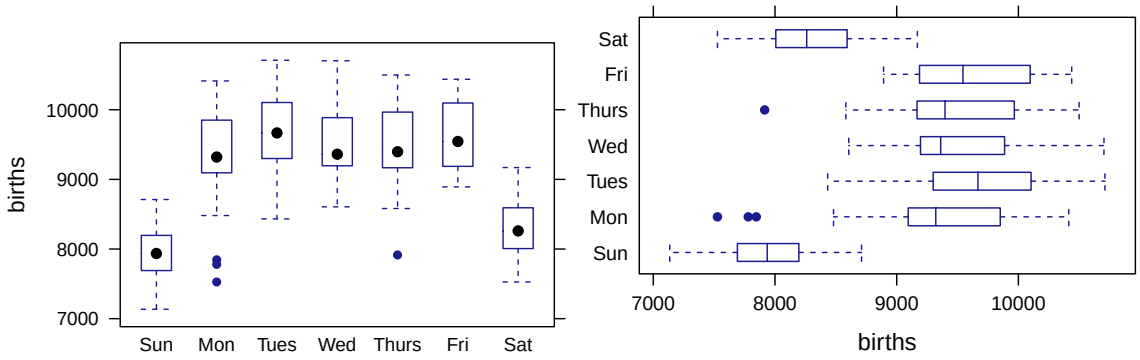
```
goal( y ~ x, data = mydata )    # pseudo-code for the formula template
```

For a plot, `goal` names the type of plot, `y` and `x` name the variables to be mapped to the vertical and horizontal axes, and `mydata` is the data frame in which these variables are found. This template allows us to create, for example, scatterplots and side-by-side box plots using `lattice` functions. Here we illustrate using the `Births78` data set from the `mosaicData` package, which, as the name suggests, contains data sets to accompany the `mosaic` package.

```
require(mosaic)     # instead of library() because students seem to remember it better
xyplot(births ~ date, data = Births78)
```




```
bwplot(births ~ wday, data = Births78)
bwplot(wday ~ births, data = Births78, pch = "|")
```




The same template can be used to create numerical summaries.

```
mean(births ~ wday, data = Births78)
```

```
##   Sun   Mon  Tues   Wed Thurs   Fri   Sat
##  7951  9371  9709  9498  9484  9626  8309
```

```
sd(births ~ wday, data = Births78)
```

```
##   Sun   Mon  Tues   Wed Thurs   Fri   Sat
##   410   608   527   461   551   488   390
```

```
favstats(births ~ wday, data = Births78)
```

```
##    wday  min   Q1 median    Q3   max mean  sd  n missing
## 1   Sun 7135 7691   7936  8196  8711 7951 410 53       0
## 2   Mon 7527 9097   9321  9838 10414 9371 608 52       0
## 3  Tues 8433 9304   9668 10084 10711 9709 527 52       0
## 4   Wed 8606 9196   9362  9880 10703 9498 461 52       0
## 5 Thurs 7915 9171   9397  9958 10499 9484 551 52       0
## 6   Fri 8892 9198   9544 10088 10438 9626 488 52       0
## 7   Sat 7527 8007   8260  8586  9170 8309 390 52       0
```

We introduced the `tally()` function for counting categorical variables. We illustrate its use with another data set from `mosaicData` (Pruim, Kaplan, and Horton 2015b). `HELPrct` contains data from the Health Evaluation and Linkage to Primary care study, a clinical trial for adult inpatients recruited from a detoxification unit. Notice that in the final example, conditional proportions are calculated.

```
tally(sex ~ substance, data = HELPrct)
```

```
##         substance
## sex      alcohol cocaine heroin
##   female      36      41     30
##   male       141     111     94
```

```
tally(sex ~ substance, data = HELPrct, margins = TRUE)
```

```
##         substance
## sex      alcohol cocaine heroin
##   female      36      41     30
##   male       141     111     94
##   Total      177     152    124
```

```
tally(sex ~ substance, data = HELPrct, margins = TRUE, format = "proportion")
```

```
##         substance
## sex      alcohol cocaine heroin
##   female   0.203   0.270  0.242
##   male     0.797   0.730  0.758
##   Total    1.000   1.000  1.000
```

Formula interfaces are provided for `mean()`, `median()`, `sd()`, `var()`, `cor()`, `cov()`, `quantile()`, `max()`, `min()`, `range()`, `IQR()`, `iqr()`, `fivenum()`, `prod()`, and `sum()`. In each case we have been careful not to break behavior of the underlying functions from `base` and `stats`.

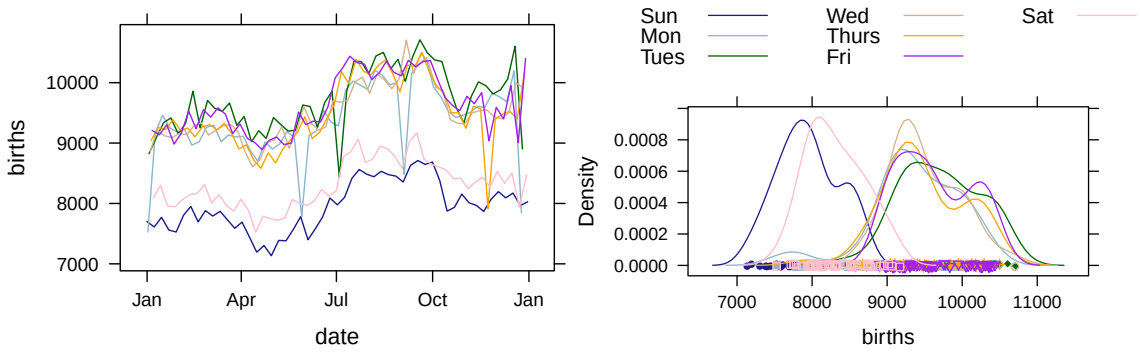
The formula template can be extended to handle one variable or more than two variables, but we recommend introducing it in the context of two-variable plots and summaries. This is for several reasons: (1) two-variable plots and numerical summaries are more "impressive" and less likely to be something students can as readily do with tools they already know, (2) working with more than one variable from the start (correctly) suggests that the most interesting parts of statistics involve more than one variable (Wild et al. 2011), and (3) the formula syntax for a single variable makes more sense in the context of two-sided formulas that it does in isolation.

Once the two-variable summaries are understood, we can add a third or fourth variable with

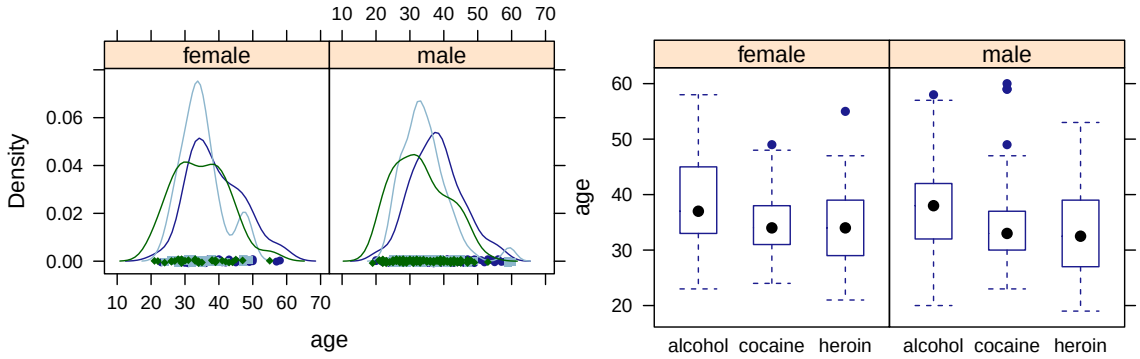
```
goal( y ~ x | z, groups = mygroups, data=mydata )    # pseudo-code
```

When plotting, `z` is used to create plots with subpanels (or facets) and `groups` indicates overlaid layers.

```
xyplot(births ~ date, groups = wday, data = Births78, type = "l")
densityplot( ~ births, groups = wday, data = Births78, auto.key=list(columns=3))
```




```
densityplot( ~ age | sex, groups = substance, data = HELPrct)
bwplot( age ~ substance | sex, data = HELPrct)
```




For numerical summaries the condition `z` and `groups` play essentially the same role, so each of the following produces an equivalent result.

```
mean( age ~ sex, data = HELPrct)
```

```
mean( ~ age | sex, data = HELPrct)
```

```
mean( ~ age, groups = sex, data = HELPrct)
```

```
## female   male
##   36.3   35.5
```

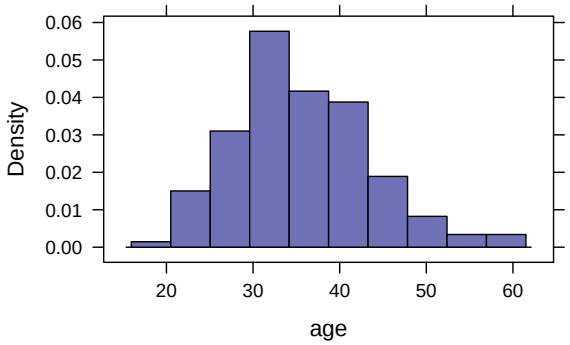
This allows us to compute numerical summaries by replacing the name of the plot with the name of the desired summary.

The one-variable template can be obtained by removing the left-hand side from the formula in a two-variable template.

```
goal( ~ x, data=mydata )                # psuedo-code
```

In the context of plotting, this makes sense since we are providing the data for the $x$-axis and allowing R to compute values for the $y$-axis:

```
histogram( ~ age, data = HELPrct)
```




Numerical summaries fit this pattern by analogy (and because R formulas are required to have a right hand side).

```
mean( ~ age, data = HELPrct)
```

```
## [1] 35.7
```

As students become familiar with the formula interface, all three forms can be brought together into a single template:

```
goal( formula, data=mydata, ... )      # psuedo-code
```

The formula template allows students to think about relationships between and among two or more variables and to test conjectures using graphical and numerical summaries. Having learned the formula interface to graphical and numerical summaries early on, new users are well prepared for modeling with `lm()`, `glm()`, and various "test" functions such as `t.test()` when the time comes. More importantly, they begin early to train their minds to ask questions of the form "How does this depend on that (and some other things)?".

By emphasizing the formula template, each of the following commands can be viewed as instances of a common template, rather than as separate things to learn.

```
bwplot(age ~ sex, data=HELPrct)
  mean(age ~ sex, data=HELPrct)
    sd(age ~ sex, data=HELPrct)
    lm(age ~ sex, data=HELPrct)
t.test(age ~ sex, data=HELPrct)
```

Similarly, by adding additional formula interfaces to `t.test()`, `binom.test()`, and `prop.test()`, and adding some additional plot types, for one-variable situations we have

```
       mean( ~ age, data=HELPrct)
         sd( ~ age, data=HELPrct)
   favstats( ~ age, data=HELPrct)
  histogram( ~ age, data=HELPrct)
    dotPlot( ~ age, data=HELPrct)   # dot plots
freqpolygon( ~ age, data=HELPrct)   # frequency polygon
    ashplot( ~ age, data=HELPrct)   # average shifted histogram
     t.test( ~ age, data=HELPrct)   # formula interface added in mosaic
 binom.test( ~ sex, data=HELPrct)   # formula interface added in mosaic
  prop.test( ~ sex, data=HELPrct)   # formula interface added in mosaic
```

Adding covariates to one- or two- variable graphical or numerical summaries fits readily into the template as well.

```
     mean( ~ age | sex, data=HELPrct)
       sd( ~ age | sex, data=HELPrct)
histogram( ~ age | sex, data=HELPrct)
   t.test( ~ age | sex, data=HELPrct)
```

While creating the correct formula can produce some challenges for new users, clearly explaining the roles of each component for plotting, for numerical summaries, and for model fitting helps demystify the situation. Instructors have had students create and interpret bivariate and trivariate graphical displays on the first day of class (Wang and Rush 2015). We have also found that explicit, early, low-stakes assessment of student mastery of the formula interface greatly improves student performance. A first quiz consisting of a single item (What is the formula template?) followed by one or two simple pencil-and-paper quizzes asking students to write the commands to recreate a handful of numerical and graphical summaries suffices.

## Handling missing data

When there are missing values, the numerical summary functions in `base` and `stats` return results that may surprise and mystify new users.

```
mean( ~ dayslink, data = HELPmiss)
```

```
## [1] NA
```

While there are workarounds using options to functions to drop values that are missing before performing the computation, these may be intimidating to new users.

```
mean( ~ dayslink, data = HELPmiss, na.rm = TRUE)
```

```
## [1] 257
```

We offer two other solutions to this situation. Our favorite is the `favstats()` function which computes a set of useful numerical summaries on the non-missing values and also reports the number of missing values.

```
favstats( ~ dayslink, data = HELPmiss)
```

```
##  min Q1 median  Q3 max mean  sd   n missing
##    2 75    363 365 456  257 151 447      23
```

The second solution is to change the default behavior of `na.rm` using `options()`. This will, of course, only affect the `mosaic` versions of these functions.

```
options(na.rm = TRUE)
mean(~ dayslink, data = HELPmiss)
```

```
## [1] 257
```

```
with(HELPmiss, base::mean(dayslink))
```

```
## [1] NA
```

Users also have the option of changing the default for `na.rm` back if they like.

```
options(na.rm = NULL)
mean(~ dayslink, data = HELPmiss)
```

```
## [1] NA
```

## Inspecting a data frame

Summaries of all variables in a data frame can be obtained using `inspect()`. For quantitative variables, the results of `favstats()` are displayed. Other summaries are provided for categorical and time variables.

```
inspect(Births78)
```

```
##
## categorical variables:
##   name   class levels   n missing                                    distribution
## 1 wday ordered      7 365       0 Sun (14.5%), Mon (14.2%), Tues (14.2%) ...
##
## quantitative variables:
##        name   class  min   Q1 median   Q3   max mean  sd   n missing
## 1    births integer 7135 8554   9218 9705 10711 9132 818 365       0
## 2 dayofyear integer    1   92    183  274   365  183 106 365       0
##
## time variables:
##   name   class      first       last min_diff max_diff   n missing
## 1 date POSIXct 1978-01-01 1978-12-31        1        1 365       0
```

## Creating and using functions

Especially in calculus, but also when modeling in statistics, it is useful to create functions defined by algebraic formulas. With `makeFun()` we can construct such functions using a formula interface and use `plotFun()` to plot them.

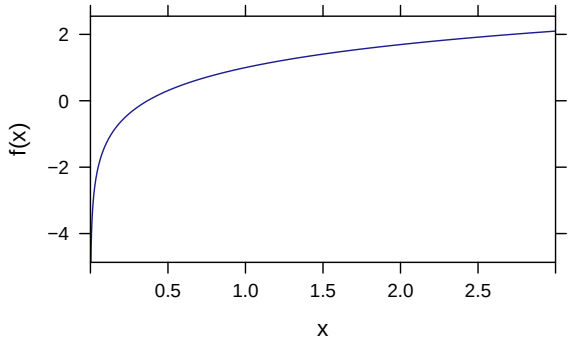
```
f <- makeFun(A + B * log(x) ~ x, A = 1, B = 1)
f
```

```
## function (x, A = 1, B = 1)
## A + B * log(x)
```

```
f(2)
```

```
## [1] 1.69
```

```
plotFun(f(x) ~ x, xlim = c(0,3))
```




More interestingly for statistics, we can use `makeFun()` to create functions from model objects created by `lm()` and `glm()`.

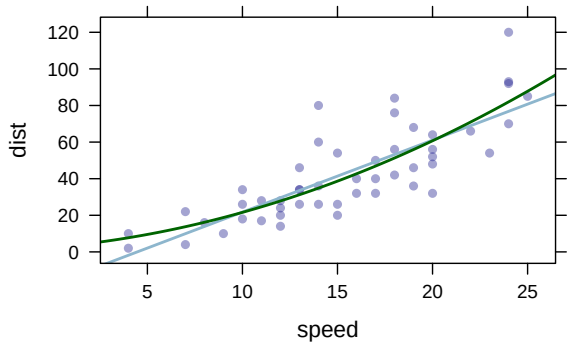
```
cars.mod1 <- lm(dist ~ speed, data = cars)
cars.mod2 <- lm(dist ~ poly(speed,2), data = cars)
dist1 <- makeFun(cars.mod1)
dist2 <- makeFun(cars.mod2)
dist2(speed = 15)
```

```
##    1
## 38.7
```

```
dist2(speed = 15, interval = "confidence")
```

```
##    fit lwr  upr
## 1 38.7  33 44.3
```

```
xyplot(dist ~ speed, data = cars, alpha = 0.4)
plotFun(dist1(s) ~ s, add = TRUE, col = 2, lwd = 2)
plotFun(dist2(s) ~ s, add = TRUE, col = 3, lwd = 2)
```

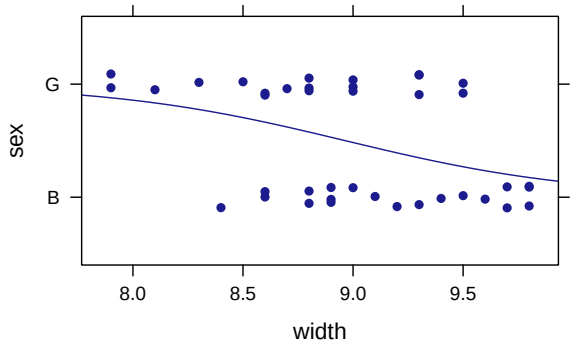



For logistic regression, when the response is coded as a factor, we need to adjust things slightly when plotting because the model function returns values between 0 and 1, but 2-level factors are coded as 1 and 2.

```
Feet.mod <- glm(sex ~ width, data = KidsFeet, family = binomial)
s <- makeFun(Feet.mod)
s(width = 8.5)
```

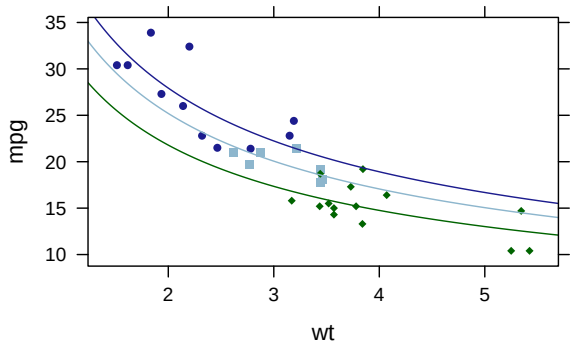
```
##     1
## 0.704
```

```
xyplot(sex ~ width, data = KidsFeet, jitter.y = TRUE)
plotFun(1 + s(w) ~ w, add = TRUE)
```



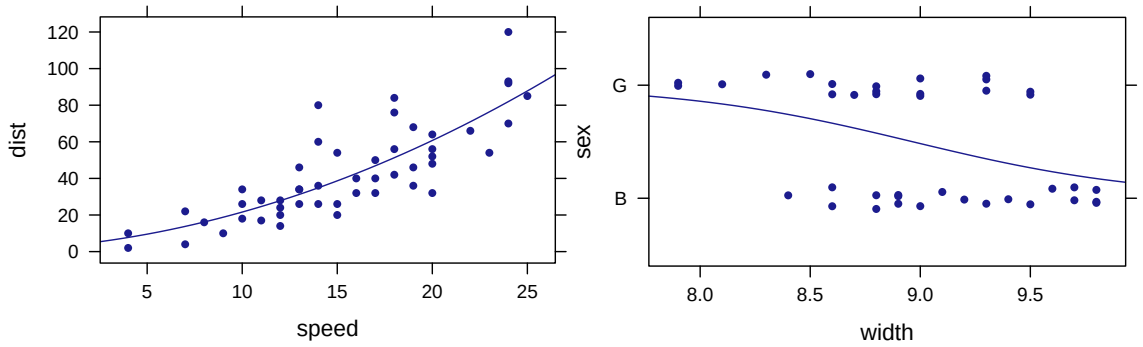

This wrapper around `predict()` is easier for beginners to use because (a) it returns a function to which inputs can be supplied without creating a data frame, (b) the resulting function returns values on the response scale by default, and (c) it back transforms a few common transformations of the response variable, including `log()` and `sqrt()` (and allows the user to provide a custom value to the `transform` argument to handle other cases).

```
mtcars.mod <- lm(log(mpg) ~ log(wt) + factor(cyl), data = mtcars)
mileage <- makeFun(mtcars.mod)
xyplot(mpg ~ wt, data = mtcars, groups = cyl)
plotFun( mileage(w, cyl=4) ~ w, add = TRUE, col = 1)
plotFun( mileage(w, cyl=6) ~ w, add = TRUE, col = 2)
plotFun( mileage(w, cyl=8) ~ w, add = TRUE, col = 3)
```



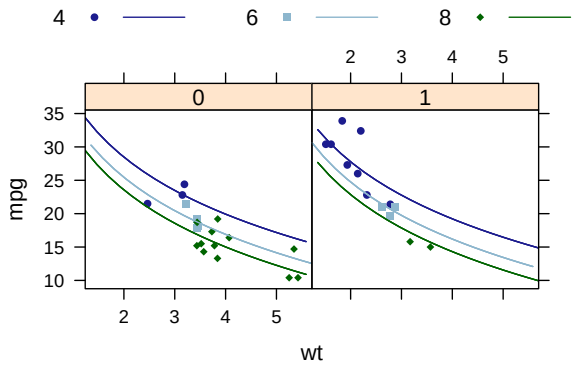

For many simple models, creating a plot can be even simpler using `plotModel()`, which also eliminates the need to manually adjust logistic regression plots when the response is a factor.

```
plotModel(cars.mod2)
plotModel(Feet.mod, jitter.y = TRUE)
```




The `plotModel()` function can also simplify visualization of more complex models.

```
mtcars.mod2 <- lm(mpg ~ log(wt) + factor(cyl) + factor(am), data = mtcars)
plotModel(mtcars.mod2, mpg ~ wt | factor(am))
```




# Randomization and Resampling

Resampling approaches have become increasingly important in statistical education [Tintle et al. (2015); Hesterberg:2015]. The `mosaic` package provides simplified functionality to support teaching inference based on randomization tests and bootstrap methods. Our goal was to focus attention on the important parts of these techniques (e.g., where randomness enters in and how to use the resulting distribution) while hiding some of the technical details involved in creating loops and accumulating values.

## A first example

As a first example, we often introduce the story of the lady tasting tea. (See (Salsburg 2002) for the details of this famous story.) But here we will test a coin to see whether it is a "fair coin". Suppose we flip the coin 20 times and observe only 6 heads, how suspicious should we be that the coin is not fair? The statistical punchline for either the lady tasting tea or testing a coin is that we want to compute the p-value for a binomial test via simulations rather than using formulas for the binomial distribution or normal approximations. But we want to do this on the first day of class, and without using any of the jargon of the preceding sentence.

Because students do not know about sampling distributions or random variables yet, but do understand the idea of a coin toss, we have provided `rflip()` to simulate tossing a coin one or several times:

```
rflip()
```

```
##
## Flipping 1 coin [ Prob(Heads) = 0.5 ] ...
##
## H
##
## Number of Heads: 1 [Proportion Heads: 1]
```

```
rflip(20)
```

```
##
## Flipping 20 coins [ Prob(Heads) = 0.5 ] ...
##
## T H T H H H H H H H T T T T H H T H H T
##
## Number of Heads: 12 [Proportion Heads: 0.6]
```

To test a null hypothesis of a fair coin, we need to simulate flipping 20 coins many times, recording for each simulation the number of heads that were observed. The `do()` function allows us to do just that using the following template

```
do(n) * {stuff to do}              # pseudo-code
```

where `{stuff to do}` is typically a single R command, but may be something more complicated. For example, we can flip 20 coins three times as follows.

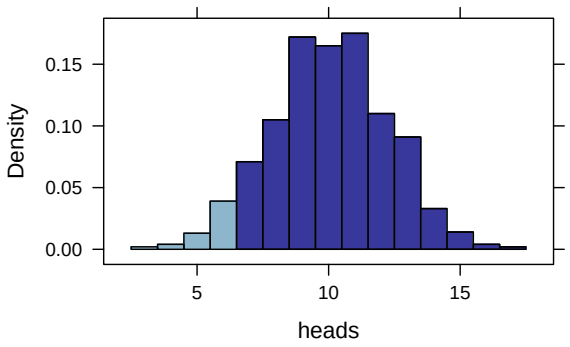
```
do(3) * rflip(20)
```

```
##    n heads tails prop
## 1 20    11     9 0.55
## 2 20     9    11 0.45
## 3 20     9    11 0.45
```

Notice that `do()` (technically `cull_for_do()`) has been clever about what information is stored for each group of 20 coin tosses. It is now a simple matter to do this many more times and use numerical or graphical summaries to investigate how unusual it is to get so few heads if the coin is indeed a fair coin.

```
Sims <- do (1000) * rflip(20)
histogram( ~ heads, data = Sims, width = 1, groups = heads <= 6)
tally ( ~(heads <= 6), data = Sims)
```

```
##
##  TRUE FALSE
##    58   942
```




(If you are familiar with `lattice`, you will notice that the `mosaic` package also adds some additional arguments to the `histogram()` function.)

## sample(), resample(), and shuffle()

To facilitate randomization and bootstrapping, `mosaic` extends `sample()` to operate on data frames. The `shuffle()` function is an alternative name for `sample()`, and `resample()` is `sample()` with `replace = TRUE`. With these in hand, all of the tests and confidence intervals seen in a traditional first course in statistics can be performed using a common outline:

1. Do it to your data
2. Do it to a randomized version of your data
3. Do it to lots of randomized versions of your data.

For example, we can use randomization in place of the two-sample t test to obtain an empirical p-value.

```
D <- diffmean(age ~ sex, data = HELPrct); D
```

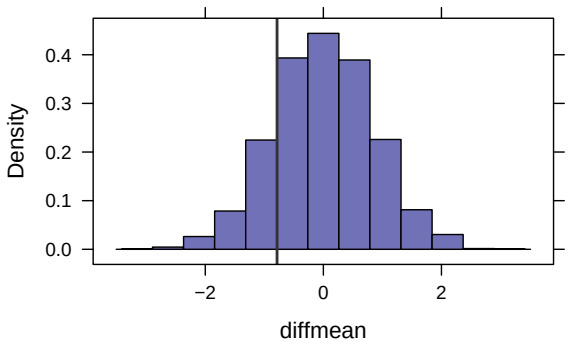
```
## diffmean
##   -0.784
```

```
do(1) * diffmean(age ~ shuffle(sex), data = HELPrct)
```

```
##   diffmean
## 1   -0.601
```

```
Null.dist <- do(5000) * diffmean(age ~ shuffle(sex), data = HELPrct)
histogram( ~ diffmean, data = Null.dist, v = D)
prop( ~ (diffmean < D), data = Null.dist, format = "prop")
```

```
##  TRUE
## 0.176
```




```
pval(t.test(age ~ sex, data = HELPrct, alternative = "greater"))
```

```
## p.value
##   0.177
```

The example above introduces three additional `mosaic` functions. The `prop()` function computes the proportion of logical vector that is (by default) `TRUE` or of a factor that is (by default) the first label; `diffmean()` is similar to `diff(mean())`, but labels the result differently (`diffprop()` works similarly for differences in proportions); and `pval()` extracts the p-value from an object of class `"htest"`.

It should be noted that although this is typically not done in simulation-based introductory statistics texts, one might prefer to calculate p-values by including the observed data in the randomization distribution. This avoids an empirical p-value of 0 and guarantees that the actual type I error rate will not exceed the nominal type I error rate.

```
count( ~ (diffmean < D), data = Null.dist)
```

```
##  TRUE
## 0.176
```

```
(1 + count( ~ (diffmean < D), data = Null.dist)) / (1 + nrow(Null.dist))  # p-value
```

```
##     TRUE
## 0.000235
```

If we are interested in a confidence interval for the difference in group means, we can use `resample()` and `do()` to generate a bootstrap distribution in one of two ways.

```
Boot.dist1 <- do(1000) * diffmean(age ~ sex, data = resample(HELPrct))
Boot.dist2 <- do(1000) * diffmean(age ~ sex, data = resample(HELPrct, groups = sex))
```

In the second example, the resampling happens within the sex groups so that the marginal counts for each sex remain fixed. This can be especially important if one of the groups is small, because otherwise some resamples might not include any observations of that group.

```
favstats(age ~ sex, data = HELPrct)
```

```
##      sex min Q1 median   Q3 max mean   sd   n missing
## 1 female  21 31     35 40.5  58 36.3 7.58 107       0
## 2   male  19 30     35 40.0  60 35.5 7.75 346       0
```

```
favstats(age ~ sex, data = resample(HELPrct))
```

```
##      sex min Q1 median Q3 max mean   sd   n missing
## 1 female  21 30     34 39  58 35.3 7.80  98       0
## 2   male  19 30     35 39  58 35.3 7.59 355       0
```

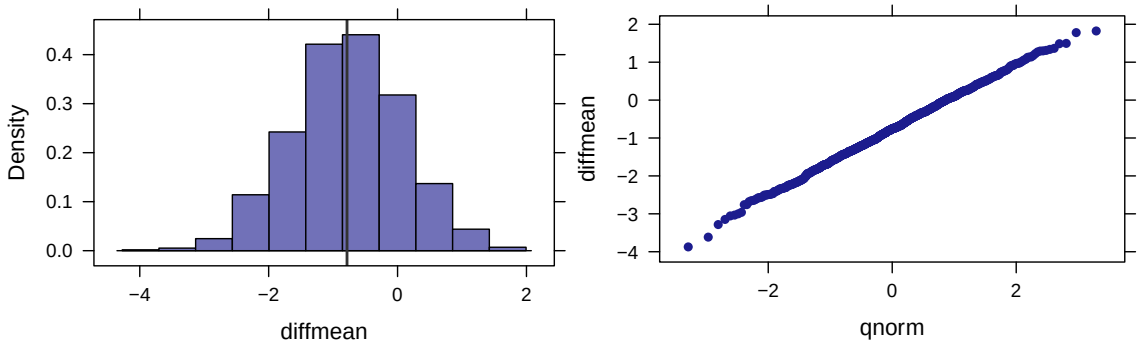
```
favstats(age ~ sex, data = resample(HELPrct, groups = sex))
```

```
##      sex min Q1 median Q3 max mean   sd   n missing
## 1 female  21 31     37 40  58 37.2 7.93 107       0
## 2   male  19 30     35 41  60 35.7 7.95 346       0
```

Using either bootstrap distribution, two simple confidence intervals can be computed. We typically introduce percentile confidence intervals first. A percentile confidence interval is calculated by determining the range of a central portion of the bootstrap distribution, which can be automated using `cdata()`. Visually inspecting the bootstrap distribution for skew and bias is an important step to make sure the percentile interval is not being applied in a situation where it may perform poorly.

```
histogram( ~ diffmean, data = Boot.dist2, v = D)
qqmath( ~ diffmean, data = Boot.dist2)
cdata( ~ diffmean, p = 0.95, data = Boot.dist2)
```

```
##       low        hi central.p
##    -2.473     0.913     0.950
```




Alternatively, we can compute a confidence interval based on a bootstrap estimate of the standard error.

```
SE <- sd( ~ diffmean, data = Boot.dist2); SE
```

```
## [1] 0.867
```

```
D + c(-1,1) * 2 * SE
```

```
## [1] -2.519  0.951
```

The primary pedagogical value of the bootstrap standard error approach is its close connection to the standard formula-based confidence interval methods. How to replace the constant 2 with an appropriate value to create more accurate intervals or to allow for different confidence levels is a matter of some subtlety (T. C. Hesterberg 2015). The simplest method is to use quantiles of a normal distribution, but this will undercover. Replacing the normal distribution with an appropriate t-distribution will widen intervals and can improve coverage, but the t-distribution is only correct in a few cases – such as when estimating the mean of a normal population – and can perform badly when the population is skewed. See the Discussion Section for more on this.

Calculating simple confidence intervals can be further automated using an extension to `confint()`.

```
confint(Boot.dist2, method = c("percentile", "stderr"))
```

```
##       name lower upper level     method estimate margin.of.error  df
## 1 diffmean -2.47 0.913  0.95 percentile   -0.784              NA  NA
## 2 diffmean -2.48 0.933  0.95     stderr   -0.784             1.7 452
```

# Extracting information

Modeled on functions like `resid()`, a number of additional functions have been added to `mosaic` to facilitate extracting information from more complicated objects. Some examples include

```
confint(t.test( ~ age, data = HELPrct))    # works for any "htest" object
```

```
##   mean of x lower upper level
## 1      35.7  34.9  36.4  0.95
```

```
pval(t.test(age ~ sex, data = HELPrct))    # works for any "htest" object
```

```
## p.value
##   0.354
```

```
stat(t.test(age ~ sex, data = HELPrct))    # works for any "htest" object
```

```
##    t
## 0.93
```

```
rsquared(lm(age ~ sex, data=HELPrct))
```

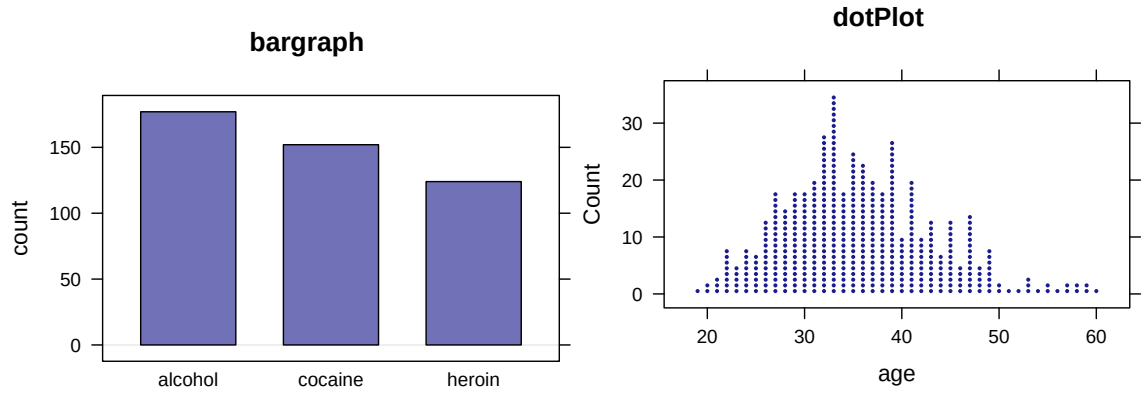
```
## [1] 0.00187
```

# Some additional visualization tools

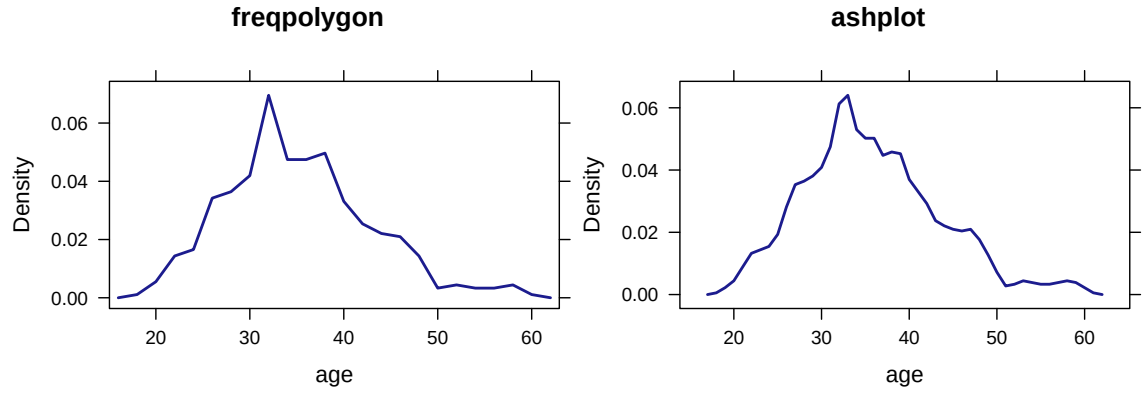
## Additional high-level lattice plots

The `mosaic` package provides several new high-level `lattice` plots, including `bargraph()`, `dotPlot()`, `freqpolygon()`, `ashplot()`, `xqqmath()`, and `plotPoints()`.

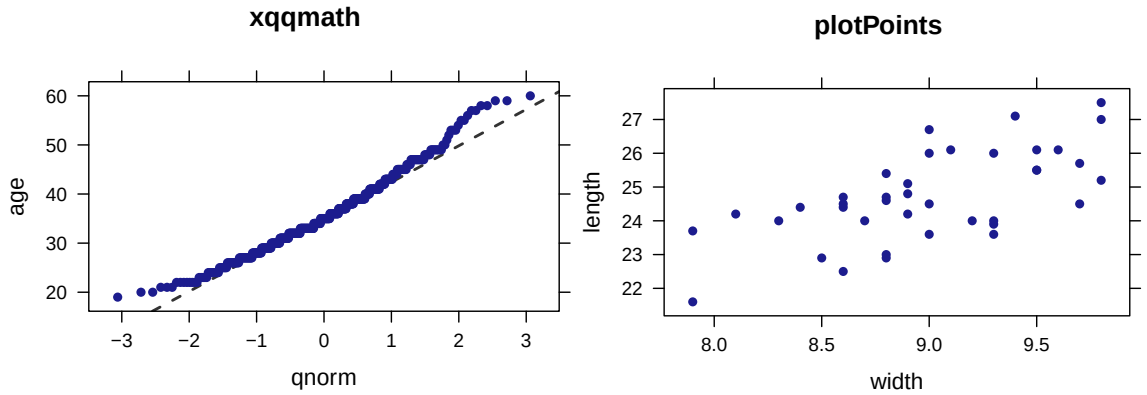
```
bargraph( ~ substance, data = HELPrct, main = "bargraph")
dotPlot( ~ age, data = HELPrct, width = 1, main = "dotPlot")
```




```
freqpolygon( ~ age, data = HELPrct, width = 2, main = "freqpolygon")
ashplot( ~ age, data = HELPrct, width = 2, main = "ashplot")
```



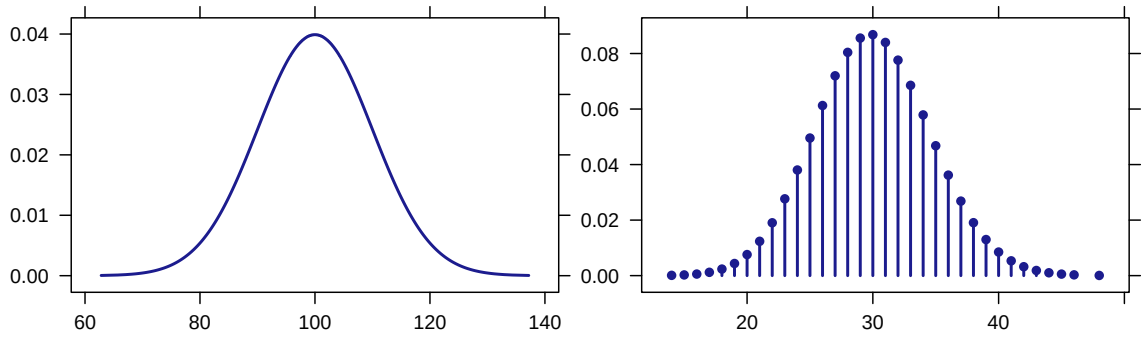

```
xqqmath( ~ age, data = HELPrct, main = "xqqmath")
plotPoints(length ~ width, data = KidsFeet, main = "plotPoints")
```




## Visualizing distributions of random variables

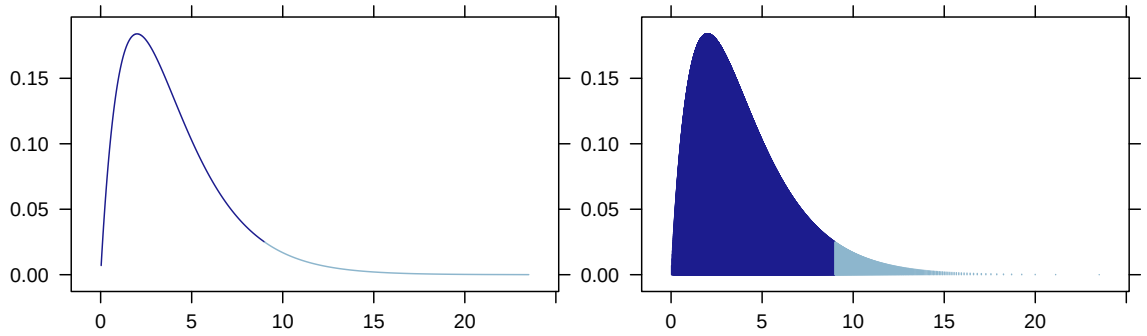
A number of functions make it simple to visualize random variables. `plotDist()` creates displays for any distribution for which standard d-, p-, and q- functions exist.

```
plotDist("norm", mean = 100, sd = 10)
plotDist("binom", size = 100, prob = 0.3)
```



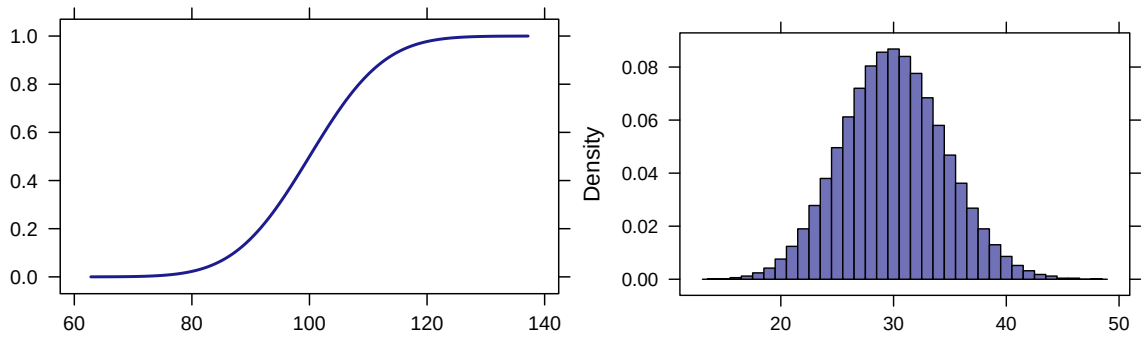

Tail probabilities can be highlighted using the `groups` argument.

```
plotDist("chisq", df = 4, groups = x > 9)
plotDist("chisq", df = 4, groups = x > 9, type = "h")
```




Using the `kind` argument, we can obtain other types of plots, including cdfs and probability histograms.

```
plotDist("norm", mean = 100, sd = 10, kind = "cdf")
plotDist("binom", size = 100, prob = 0.3, kind = "histogram")
```




For several distributions, we provide augmented versions of the distribution and quantile functions that assist students in understanding what values are returned by functions like `pnorm()` and `qnorm()`.

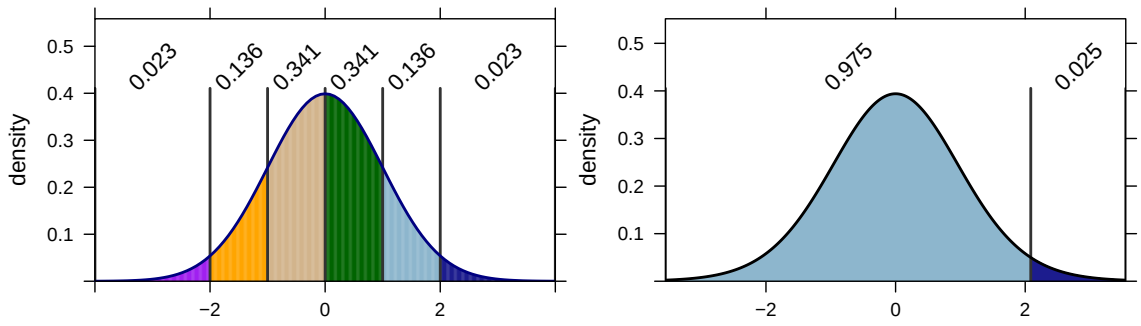
```
xpnorm(-2:2)
```

```
##
## If X ~ N(0,1), then
##
##  P(X <= -2) = P(Z <= -2) = 0.0228
##      P(X <= -1) = P(Z <= -1) = 0.1587
##      P(X <= 0) = P(Z <= 0) = 0.5
##      P(X <= 1) = P(Z <= 1) = 0.8413
##      P(X <= 2) = P(Z <= 2) = 0.9772
##  P(X >  -2) = P(Z >  -2) = 0.9772
##      P(X >  -1) = P(Z >  -1) = 0.8413
##      P(X >  0) = P(Z >  0) = 0.5
##      P(X >  1) = P(Z >  1) = 0.1587
##      P(X >  2) = P(Z >  2) = 0.0228
```

```
## [1] 0.0228 0.1587 0.5000 0.8413 0.9772
```

```
xqt(0.975, df = 20)
```

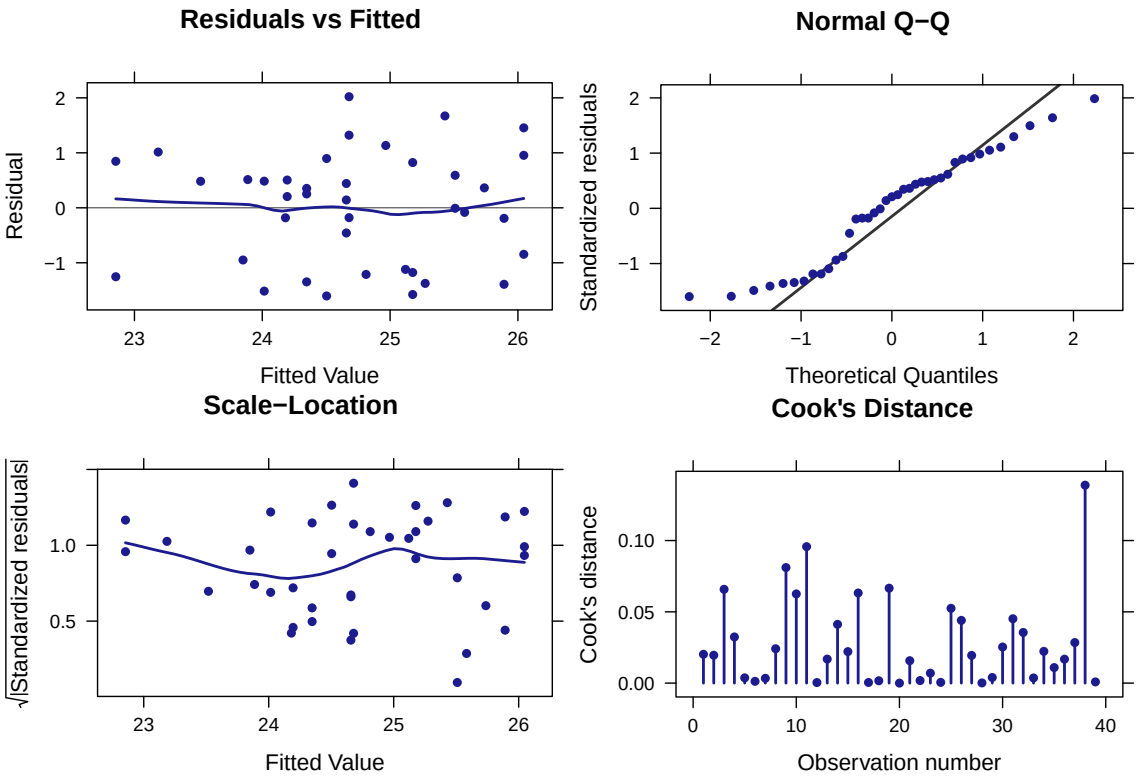
```
## [1] 2.09
```



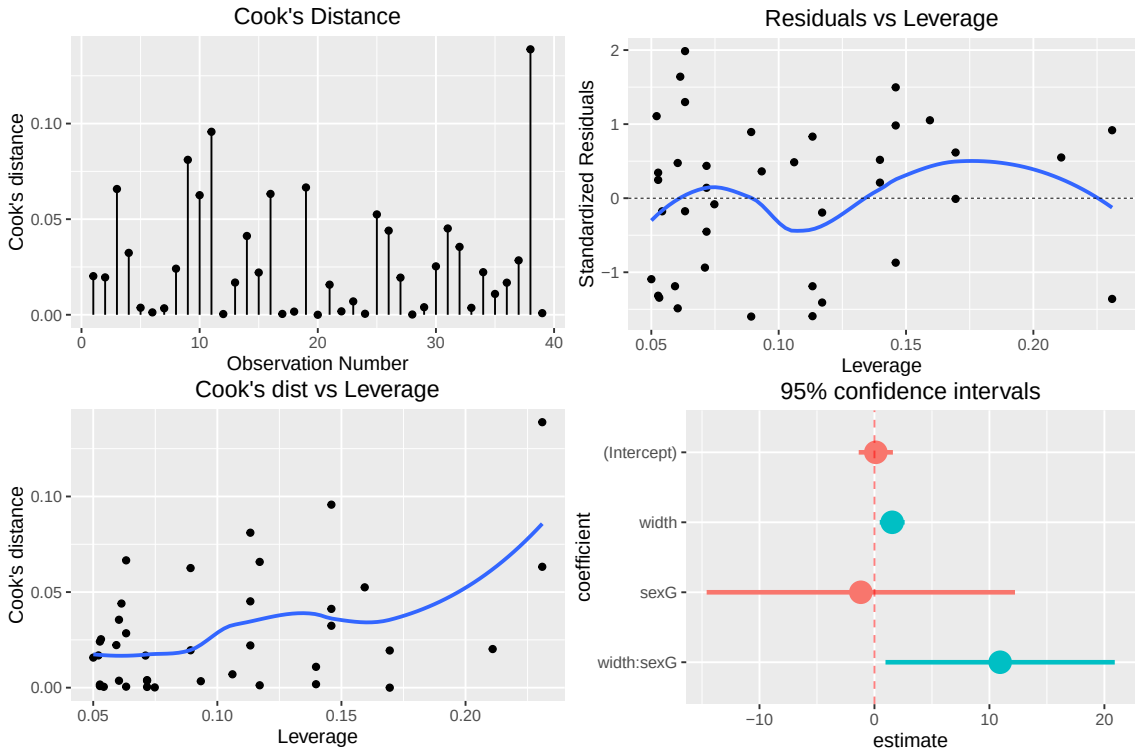

## mplot()

The `mplot()` function has two primary use cases: creating diagnostic plots for lm and glm objects, and interactively creating data visualizations using the variables in a data frame. Given a model object as its first argument, `mplot()` provides similar diagnostic plots to those produced via `plot()` but with two primary differences: the user may select to use either `lattice` or `ggplot2` (Wickham 2009) graphics instead of base graphics, and an additional plot type is provided to visualize the confidence intervals for the coefficients of the model.

```
mod <- lm(length ~ width * sex, data = KidsFeet)
mplot(mod, system = "lattice", which = 1:4)
```

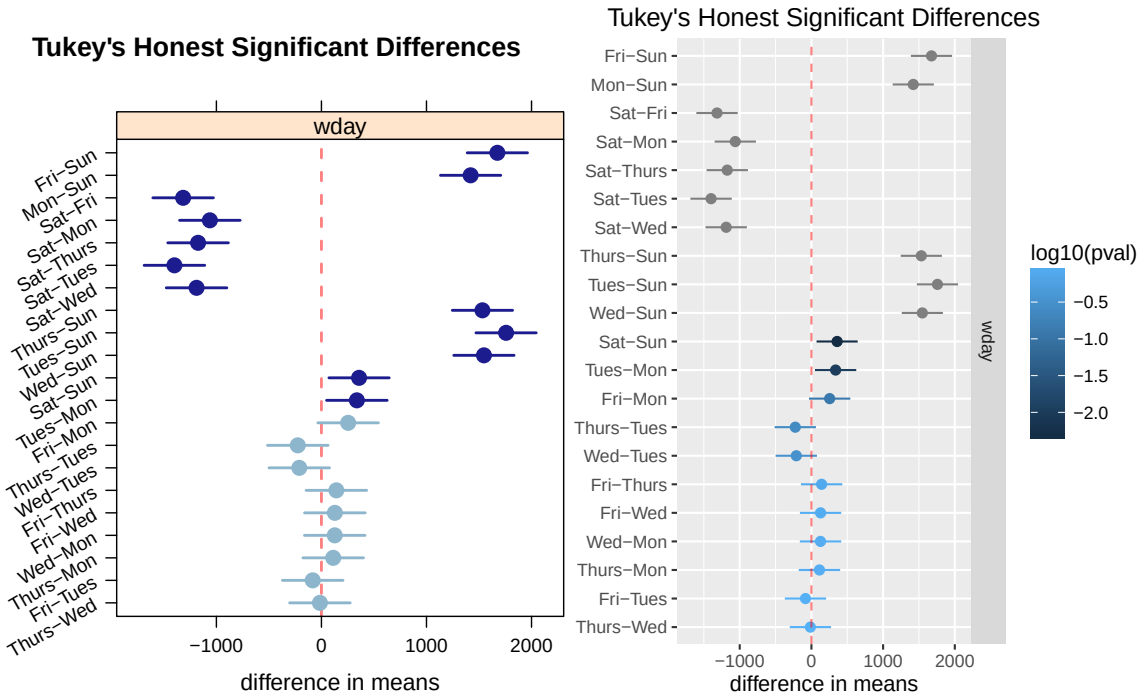



```
mplot(mod, system = "ggplot2", which = 4:7)
```




We can also use `mplot()` to visually represent the results of `TukeyHSD()`, which we can apply directly to objects produced by `lm()`.

```
mplot(TukeyHSD(lm(births ~ wday, data = Births78)), order = "pval")
mplot(TukeyHSD(lm(births ~ wday, data = Births78)), order = "pval", system = "ggplot2")
```




Again, there are options to create either `lattice` or `ggplot2` plots, and the resulting plots are formatted in a way that makes them usable in a wider range of scenarios than are those produced using `plot()`.

A second use for `mplot()` is to create `lattice` and `ggplot2` plots interactively within RStudio. Issuing the following command in RStudio will bring up a plot that can be modified by making choices interactively.

```
mplot(HELPrct)
```

The menu (see Figure~1) allows the user to choose either `lattice` or `ggplot2` graphics, to select the type of plot and the variables used, and to control a few of the most commonly used features that modify a plot (faceting, color, legends, log-scaling, fitting a linear model or LOESS smoother). The "Show Expression" button exports the command used to create the plot into the console. From there it can be edited or copied and pasted into an R Markdown document. This can be very useful for new users working to master the syntax for a particular graphical display.

# Additional features

The `mosaic` package depends on `lattice` and `ggplot2` so that plots can be made using either system whenever the `mosaic` package is attached. It also depends on `dplyr` (Wickham and Francois 2015), but for a different reason. The functions in `dplyr` implement a "less volume, more creativity" approach to data transformation and we encourage its use along side `mosaic`. Unfortunatley, there are several function names – most notably `do()` and `tally()` – that exist in both packages. After the release of `dplyr` we modified the functions in `mosaic` so that the two packages can coexist amicably as long as `mosaic` comes before `dplyr` in the search path.

Table 1 lists some additional functions in the `mosaic` package not highlighted above. The package also contains three templates for creating R Markdown documents in RStudio. Each ensures that the `mosaic`

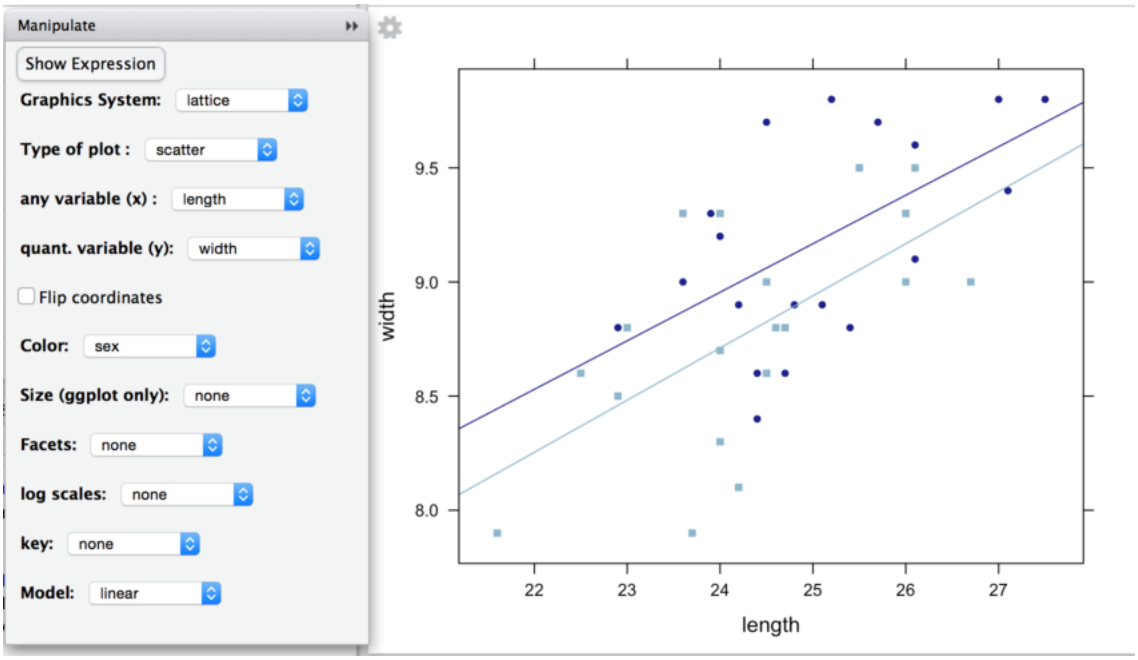



Figure 1: In RStudio, `mplot()` can be used to interactively generate plots using variables in a data frame.

package is attached, sets the default theme for `lattice` graphics to `theme.mosaic()`, chooses a somewhat smaller default size for graphics, and includes a comment reminding users to attach any packages they intend to use. The "fancy" template demonstrates several features of R Markdown, and the "plain" templates allow users to start with a clean slate. See (Baumer et al. 2014) for a discussion of how R Markdown can be used in statistics courses.

# Discussion

## Advantages of the mosaic approach

One of the keys to successfully empowering students to think with data is providing them both a conceptual framework that allows them to know what to look for and how to interpret what they find, and a computational toolbox that allows them to do the looking. The approach made possible with the `mosaic` package simplifies the transition from thinking to computing by reducing the number of computational templates students learn so that cognitative effort can be spent elsewhere, and by having those templates reflect, support, and deepen the underlying thinking (Grolemund and Wickham 2014). Because of the connection between conceptual understanding and these computational tools, the use of R can also reveal misunderstandings that might otherwise go unnoticed.

For students who take additional courses after the first course, R has the capability to support the increasing complexity of the data and analyses students encounter in subsequent courses and research projects. Eventually, students will need to learn more about the structure of R as a language, the types of objects it supports, and alternative ways of approaching the same task. But early on, it is more important that students can successfully and independently exercise computational and statistical creativity.

| function | uses |
|---|---|
| `CIsim()` | demonstrate coverage rates of confidence intervals. |
| `statTally()` | investigate test statistics and their empirical distributions. |
| `panel.lmbands()` | add confidence and prediction bands to scatter plots. |
| `ladd()` | simplified layering in `lattice` plots. |
| `xchisq.test()` | an extension to `chisq.test()` that prints a table including observed and expected counts, contribution to the chi-squared statistics and residuals. |
| `zscore()` | convert a numeric vector into z-scores. |
| `D()`, `antiD()` | derivative and antiderivative operators that take a function as input and return a function. For simple functions, the operations are done symbolically. |
| `col.mosaic()` | a `lattice` theme with colors that project better than the `lattice` defaults. |
| `dot()`, `project()`, `vlength()` | linear algebra on vectors. |
| `ediff()` | like `diff()`, but the returned vector is padded with `NA`s so that the length is the same as the input vector. |
| `SAD()`, `MAD()` | all pairs sum and mean of absolute differences |
| `rgeo()` | randomly sample latitidue, longitude pairs uniformly over the globe |
| `googleMap()` | show google maps in a browser. Together with `rgeo()`, this can be used to view maps of randomly selected points on the globe. |

Table 1: Some additional functions in the `mosaic` package.

## Challenges of using R in introductory courses

But using R is not without some challenges. The first challenge is to get all of the students up and running with R. The use of an RStudio server allows an institution or instructor to install and configure R and its packages and students to work within a web browser, essentially eliminating the start-up costs for the students. Otherwise, instructors must assist students as they navigate installation of R and whichever additional packages are required.

Once students have access to R, the `mosaic` package reduces, but does not eliminate, the amount of syntax students need to learn. It is important to emphasize the similarity among commands within a template, to remind students that R is case sensitive, to show them how to take advantage of short cuts like tab completion and code history navigation, and to explicitly teach students how to interpret some of the most common R error messages. This goes a long way toward smoothing the transition to a command line interface that is not as forgiving as Google search, which may be many students' only other experience with a command line interface.

In our experience, the most commonly occuring struggles for students using `mosaic` are

1. General ainxiety over typing commands.

   Although students are very familiar with using computers and computerized devices like smart phones, many of them have little experience typing commands that require following syntax rules. The "Less Volume, More Creativity" approach helps with this, by reducing the volume, but it remains important to highlight repeatedly the similarities among commands and to help students learn to understand the most common error messages R produces so that they can quickly, easily, and comfortably recover from innevitable typing errors. Even if a class does not typically meet in a computer laboratory or take advantage of studetn laptops, it can be useful to arrange some sessions early in the course where students are using RStudio while someone is there to quickly help them when they get stuck. Avoiding frustration in students' early experience with R goes a long way in overcoming anxiety.

   As a bonus instructional method, the authors make frequent typing mistakes in front of the class. While

we could not avoid this if we tried, it does serve to demonstrate both how to recover from errors and that nothing drastic has happened when an error message is displayed.

One big advantage of the command line interface is that it is much easier to help students by email or in a discussion forum. Encourage students to copy both their commands and the error messages or output that were produced. Even better, have them share their work in the form of an R Markdown file. We find students are much more capable of doing this than they are of correctly describing the chain of events they initiated in a menu-driven system. (It is also much easier to give detailed instructions and examples.)

2. Confusion over the tilde (~).

   The tilde is a small symbol, easily overlooked on the screen or on paper, so students will sometimes omit it, or put it where it doesn't belong. For several of our functions, we allow `x` in place of `~ x` to help ease the pain of mistyping things. But we recommend that instructors teach the use of `~` in all situations. A similar thing occurs with explicitly naming the `data` argument, which is not required for the `lattice` functions, but is for several others. Teaching the forms that work in all contexts is easier than teaching which contexts allow which forms.

   As a visual aid, we recommend surrounding the `~` with a space on either side, even in 1-sided formulas.

3. Difficulty in setting up the R environment

   This is all but eliminated when using and RStudio server, but in situations where instructors prefer a local R installation for each student, there are often a few issues involved in getting all students up and running. Installation of R and RStudio is straightforward, but one should make sure that students all have the latest version of each. To use the `mosaic` package, a number of additional packages must be installed. We recommend beginning with

```
update.packages()
```

or the equivalent operation from the RStudio Packages tab to make sure all packages currently on the system are up to date. In most cases,

```
install.packages("mosaic")
```

(again, this can also be done via the Packages tab in RStudio) will take care of the rest. But ocassionally some package will not install correctly on a particular student's computer. Installing that package directly rather than as part of the dependencies of `mosaic` often solves this problem or at least provides a useful diagnostic regarding what the problem might be.

## Better bootstrap confidence intervals

The percentile and "t with bootstrap standard error" confidence intervals have been improved upon in a number of ways. We generally do little more than mention this fact to students in a first course. One improvement is the bootstrap-t interval. Rather than attempting to determine the best degrees of freedom for a Student's t-distribution, the bootstrap-t approximates the actual distribution of

$$t = \frac{\hat{\theta} - \theta}{SE}$$

using the boostrap distribution of

$$t^* = \frac{\hat{\theta^*} - \hat{\theta}}{SE^*} \ ,$$

where $\hat{\theta^*}$ and $SE^*$ are the estimate and estimated standard error computed from each bootstrap distribution. Implementing the bootstrap-t interval requires either an extra level of conceptual framework or much more

calculation to determine the values of $SE^*$. If a standard error formula exists (e.g., $SE = s/\sqrt{n}$), this can be applied to each bootstrap sample along with the estimator. An alternative is to iterate the bootstrap procedure (resampling from each resample) to estimate $SE^*$. Since standard errors are easier to estimate than confidence intervals, fewer resamples are required (per resample) at the second level; nevertheless, the additional computational overhead is significant.

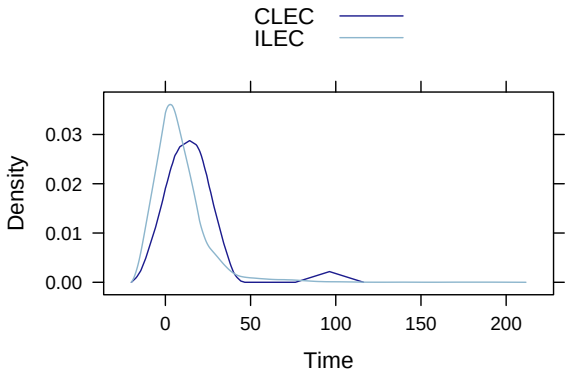
The `mosaic` package does not attempt to provide a general framework for the bootstrap-t or other "second-order accurate" boostrap methods. Packages such as `resample` (T. Hesterberg 2015) are more appropriate for situations where speed and accuracy are of utmost importance. But the bootstrap-t confidence interval can be computed using `confint()`, `do()` and `favstats()` in the case of estimating a single mean or the difference between two means.

In the example below, we analyse a data set from the `resample` package. The `Verizon` data set contains repair times for customers in CLEC (competitive) and ILEC (incumbant) local exchange carrior.

```
# NB: the resample package has name collisions with mosaic, so we only load the data
data(Verizon, package = "resample")
ILEC <- Verizon %>% filter(Group == "ILEC")       # dplyr is a dependency of mosaic
favstats( ~ Time, groups = Group, data = ILEC)
```

```
##   Group min   Q1 median   Q3 max mean   sd    n missing
## 1  CLEC  NA   NA     NA   NA  NA  NaN   NA    0       0
## 2  ILEC   0 0.73   3.59 7.08 192 8.41 14.7 1664       0
```

```
 ashplot( ~ Time, groups = Group, data = Verizon, auto.key = TRUE, width = 20)
```




The skewed distributions of the repair times and unequal sample sizes highlight differences between the bootstrap-t and simpler methods.

```
BootT1 <- do(1000) * favstats(~ Time, data = resample(ILEC))
confint(BootT1, method = "boot")
```

```
##   name lower upper level      method estimate
## 1 mean  7.85  9.14  0.95 bootstrap-t     8.41
```

```
BootT2 <- do(1000) * favstats( ~ Time, groups = Group, data = resample(Verizon, groups = Group))
confint(BootT2, method = "boot")
```

```
##       name lower upper level      method estimate
## 1 diffmean -22.3 -2.44  0.95 bootstrap-t     -8.1
```

This can also be accomplished manually, although the computations are a bit involved for the 2-sample case. Here are the manual computations for the 1-sample case:

```
estimate <- mean( ~ Time, data = ILEC); estimate
```

```
## [1] 8.41
```

```
SE <- sd( ~ mean, data = BootT1); SE
```

```
## [1] 0.341
```

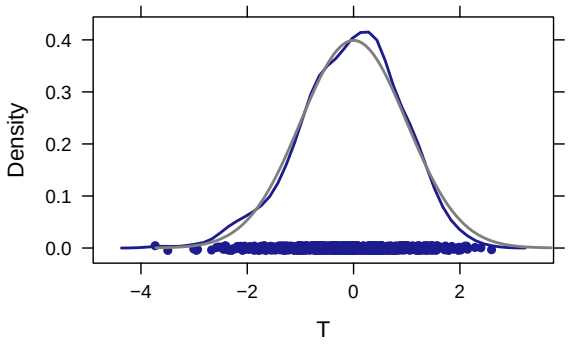
```
BootT1a <- BootT1 %>% mutate( T = (mean - mean(mean)) / (sd/sqrt(n)))
q <- quantile(~ T, p = c(0.975, 0.025), data = BootT1); q
```

```
## 97.5%  2.5%
##     1     1
```

```
estimate - q * SE
```

```
## 97.5%  2.5%
##  8.07  8.07
```

```
densityplot( ~ T, data = BootT1a)
plotDist("norm", add = TRUE, col="gray50")
```




For comparison, here are the intervals produced by `t.test()` and the percentile method.

```
confint(t.test( ~ Time, data = ILEC))
```

```
##   mean of x lower upper level
## 1      8.41  7.71  9.12  0.95
```

```
BootT1b <-
  do(1000) * mean( ~ Time, data = resample(ILEC))
confint(BootT1b, method = "perc")
```

```
##   name lower upper level     method estimate
## 1 mean  7.73  9.13  0.95 percentile     8.41
```

```
confint(t.test(Time ~ Group, data = Verizon))
```

```
##   mean in group CLEC mean in group ILEC  lower upper level
## 1               16.5               8.41 -0.362  16.6  0.95
```

```
BootT2b <-
  do(1000) * diffmean(Time ~ Group, data = resample(Verizon, groups = Group))
confint(BootT2b, method = "perc")
```

```
##       name lower upper level     method estimate
## 1 diffmean -16.6 -1.58  0.95 percentile     -8.1
```

The intervals produced by `t.test()` are narrower, do the least to compensate for skew, undercover, and miss more often in one direction than in the other (T. C. Hesterberg 2015).

## Efficiency Issues

For applications where speed is of utmost importance, it is better to avoid some of the `mosaic` wrappers. For example, for the numerical summary functions, the `mosaic` versions cannot be faster than their counterparts in `base` or `stats` (because eventually they call the underlying functions) and may be noticeable slower in contexts where they are called many times. In particular, using the formula interface requires parsing the formula and creating a new object to contain the data described by the formula.

```
microbenchmark::microbenchmark(
  base::mean(rnorm(1000)),
  mosaic::mean(rnorm(1000)),
  mosaic::mean(~ rnorm(1000)))
```

```
## Unit: microseconds
##                         expr   min    lq  mean median    uq  max neval cld
##     base::mean(rnorm(1000))  90.3  93.4  97.8   95.5  97.7  174   100 a
##   mosaic::mean(rnorm(1000)) 233.1 242.6 260.2  255.6 268.0  472   100  b
##  mosaic::mean(~rnorm(1000)) 706.3 740.5 797.9  755.0 804.9 1521   100   c
```

```
microbenchmark::microbenchmark(
  base::mean(rnorm(10000)),
  mosaic::mean(rnorm(10000)),
  mosaic::mean(~ rnorm(10000)))
```

```
## Unit: microseconds
##                          expr  min   lq mean median   uq  max neval cld
##     base::mean(rnorm(10000))  799  806  861    808  826 3071   100 a
##   mosaic::mean(rnorm(10000))  952  964 1011    975 1008 1711   100  b
##  mosaic::mean(~rnorm(10000)) 1420 1462 1585   1508 1597 4373   100   c
```

On the other hand, for aggregated numerical summaries, the loss in performance may represent a small price to pay for the simplified syntax.

```
microbenchmark::microbenchmark(
  aggregate = with(iris, aggregate(Sepal.Length, list(Species), base::mean)),
  dplyr = iris %>%
    sample_frac(size = 1.0, replace = TRUE) %>%
    group_by(Species) %>%
    summarise(mean = base::mean(Sepal.Length)),
  mosaic = mean(Sepal.Length ~ Species, data = resample(iris))
)
```

```
## Unit: microseconds
##       expr  min   lq mean median   uq  max neval cld
##  aggregate  779  844  919    897  935 1662   100 a
##      dplyr 1257 1364 1544   1433 1526 4935   100  b
##     mosaic 1542 1639 1739   1682 1787 2800   100   c
```

Similarly, using `do()` comes at a price, although here the price has more to do with the extra work involved in culling the objects and reformatting the results. The looping itself is as fast as using `replicate()` – indeed the underlying code is very similar.

```
microbenchmark::microbenchmark( times = 50,
  do = do(500) * diffmean( age ~ shuffle(sex), data = HELPrct),
  replicate = replicate(500, diffmean( age ~ shuffle(sex), data = HELPrct))
)
```

```
## Unit: milliseconds
##       expr min  lq mean median  uq max neval cld
##         do 705 715  724    721 727 865    50   b
##  replicate 686 701  708    705 711 860    50  a
```

Furthermore, `do()` can take advantages of multiple cores if the `parallel` package is attached. Even on a laptop with a single quad-core processor, the speed-up is noticible.

```
library(parallel)
options("mosaic:parallelMessage" = FALSE)
microbenchmark::microbenchmark( times = 50,
  do = do(500) * diffmean( age ~ shuffle(sex), data = HELPrct),
  replicate = replicate(500, diffmean( age ~ shuffle(sex), data = HELPrct))
)
```

```
## Unit: milliseconds
##       expr min  lq mean median  uq max neval cld
##         do 516 530  549    540 552 898    50  a
##  replicate 687 699  732    730 757 898    50   b
```

## Acknowledgements

Partial support for this work was provided by the National Science Foundation DUE 0920350 (Project MOSAIC). We thank Xiaofei (Susan) Wang for helpful comments and suggestions.

Baumer, Ben, Mine Cetinkaya-Rundel, Andrew Bray, Linda Loi, and Nicholas J Horton. 2014. "R Markdown: Integrating a Reproducible Analysis Tool into Introductory Statistics." *Technology Innovations in Statistics Education* 8 (1). http://escholarship.org/uc/item/90b2f5xh.

Grolemund, Garrett, and Hadley Wickham. 2014. "A Cognitive Interpretation of Data Analysis." *International Statistical Review* 82 (2): 184–204. http://EconPapers.repec.org/RePEc:bla:istatr:v:82:y:2014:i:2:p:184-204.

Hesterberg, Tim. 2015. *Resample: Resampling Functions*. https://CRAN.R-project.org/package=resample.

Hesterberg, Tim C. 2015. "What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum." *The American Statistician*.

Horton, Nicholas J., Benjamin S. Baumer, and Hadley Wickham. 2015. "Setting the Stage for Data Science: Integration of Data Management Skills in Introductory and Second Courses in Statistics." *CHANCE* 28 (2): 40–50.

Nolan, Deborah, and Duncan Temple Lang. 2010. "Computing in the Statistics Curricula." *The American Statistician* 64 (2): 97–107. http://dx.doi.org/10.1198/tast.2010.09132.

Pruim, Randall. 2011. "Teaching Statistics with R." In *Joint Statistics Meetings Roundtable*.

Pruim, Randall, Daniel Kaplan, and Nicholas Horton. 2015a. *Mosaic: Project MOSAIC Statistics and Mathematics Teaching Utilities*.

———. 2015b. *MosaicData: Project MOSAIC (Mosaic-Web.org) Data Sets*.

Ridgway, J. 2015. "Implications of the Data Revolution for Statistics Education." *International Statistical Review*.

Salsburg, David. 2002. *The Lady Tasting Tea: How Statistics Revolutionized Science in the Twentieth Century*. Paperback; Holt Paperbacks.

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with R*. New York: Springer. http://lmdvr.r-forge.r-project.org.

Tintle, N., B. Chance, G. Cobb, S. Roy, T. Swanson, and J. VanderStoep. 2015. "Combating anti-statistical thinking using simulation-based methods throughout the undergraduate curriculum." *The American Statistician* 69 (4).

Wang, Xiaofei (Susan), and Cynthia Rush. 2015. "Visualization as the Gateway Drug to Statistics in Week One." In *United States Conference on Teaching Statistics*.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://had.co.nz/ggplot2/book.

Wickham, Hadley, and Romain Francois. 2015. *Dplyr: A Grammar of Data Manipulation*. https://github.com/hadley/dplyr.

Wild, C J, Pfannkuch M, Regan M, and Horton N J. 2011. "Towards More Accessible Conceptions of Statistical Inference." *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 174 (part 2): 247–95.