Updates to R Programming in Foundations and Applications of Statistics

Randall Pruim*
Calvin College
Grand Rapids, MI

August, 2013

Since the publication of *Foundations and Applications of Statistics*, I have been working with colleagues from the NSF-funded Project MOSAIC to create and improve the mosaic package. Many functions originally in the fastR package have been moved to the mosaic package; some of these have subsequently been improved. Additional functionality has been added to the mosaic package over time that I would have used in *Foundations and Applications of Statistics*, had they existed at the time the book was written. This vignette points out some of these features for students and intructors who might prefer these alternative approaches.

1 Chapter 1: Summarizing Data

1.1 Taking Advantage of Formulas

One of the big changes in mosaic is the wider support for formula interfaces. Several instances of this approach could be used in Chapter 1. The use of a formula interface has several advantages, the chief among them being a systematization of numerical summaries, graphical summaries, and linear models into a common syntactic template:

```
goal(formula, data = mydata, ...)
```

Common formula shapes include the following

```
goal(~x, data = mydata)  # for single variable summaries
goal(y ~ x, data = mydata)  # for two-variable summaries and linear models
goal(y ~ x | z, data = mydata)  # for multi-variable summaries and faceting in plots
```

The function name typically names the goal for the computation (e.g., histogram(), mean(), tally(), etc.). The formula is described using variables in data frame mydata (and removing the need for the $ operator or with() constructions).

1.1.1 tally()

The tally() function provides a formula interface for constructing tables.

*rpruim@calvin.edu

```
require(fastR)
trellis.par.set(theme = col.mosaic())  # change default colors, etc.
table(iris$Species)

##
##     setosa versicolor  virginica
##         50         50         50

tally(~Species, data = iris)

##
##     setosa versicolor  virginica      Total
##         50         50         50        150
```

By default, `tally()` adds marginal totals, but these can be turned off, if desired:

```
tally(~Species, data = iris, margins = FALSE)

##
##     setosa versicolor  virginica
##         50         50         50
```

Tallies can be presented as counts, proportions, or percents:

```
tally(~Species, data = iris, format = "count")

##
##     setosa versicolor  virginica      Total
##         50         50         50        150

tally(~Species, data = iris, format = "percent")

##
##     setosa versicolor  virginica      Total
##      33.33      33.33      33.33     100.00

tally(~Species, data = iris, format = "proportion")

##
##     setosa versicolor  virginica      Total
##     0.3333     0.3333     0.3333     1.0000
```

The default format is chosen based on the shape of the formula.

1.1.2 Numerical Summaries

The `mosaic` package provides a formula interface for a number of numerical summary functions.

```
mean(~Sepal.Length, data = iris)

## [1] 5.843
```

```r
median(~Sepal.Length, data = iris)

## [1] 5.8

sd(~Sepal.Length, data = iris)

## [1] 0.8281

iqr(~Sepal.Length, data = iris)

## [1] 1.3

favstats(~Sepal.Length, data = iris)

##  min  Q1 median  Q3 max  mean     sd   n missing
##  4.3 5.1    5.8 6.4 7.9 5.843 0.8281 150       0
```

Furthermore, the use of a formula with left and right sides allows us to summarize within groups without using the `summary()` function:

```r
mean(Sepal.Length ~ Species, data = iris)

##     setosa versicolor  virginica
##      5.006      5.936      6.588

favstats(Sepal.Length ~ Species, data = iris)

##            min    Q1 median  Q3 max  mean     sd  n missing
## setosa     4.3 4.800    5.0 5.2 5.8 5.006 0.3525 50       0
## versicolor 4.9 5.600    5.9 6.3 7.0 5.936 0.5162 50       0
## virginica  4.9 6.225    6.5 6.9 7.9 6.588 0.6359 50       0
```

Use

```r
?mean
```

to get a list of additional functions that take advantage of the formula interface.

1.2 Treating data like distributions

In analogy to functions like `pnorm()` and `qnorm()`, the `mosaic` package provides `pdata()` and `qdata()`.

```r
qdata(0.5, Sepal.Length, data = iris)

## 50%
## 5.8

median(~Sepal.Length, data = iris)

## [1] 5.8
```

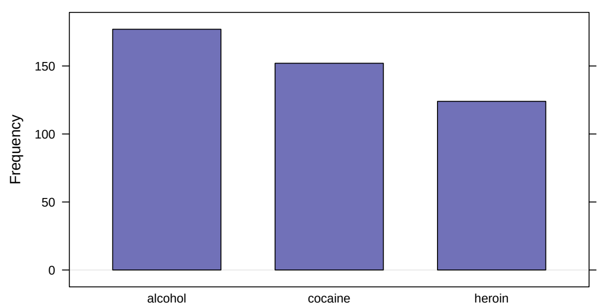
```
pdata(5, Sepal.Length, data = iris)

## [1] 0.2133

tally(~(Sepal.Length <= 5), data = iris, format = "proportion")

##
##   TRUE  FALSE  Total
## 0.2133 0.7867 1.0000
```

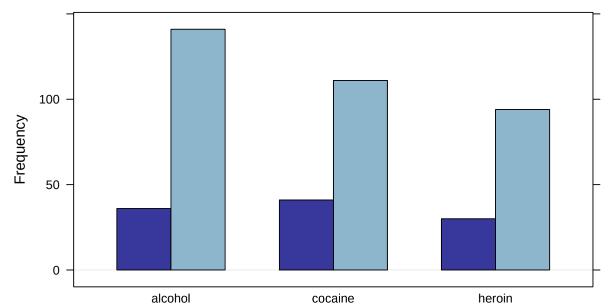
1.3 More plots

1.3.1 bargraph()

The mosaic function barchart() requires the user to first tally the data to be plotted. The bargraph() function makes it easy to create bar graphs in the same way other lattice plots are created.

```
bargraph(~substance, data = HELPrct)
bargraph(~substance, data = HELPrct, groups = sex)
```



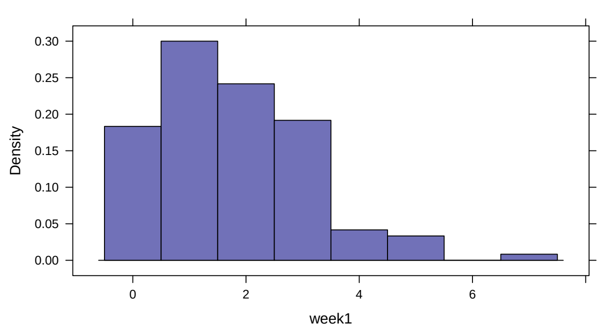


1.3.2 Augmented histogram()

The mosaic package adds several features to the histogram() function (taking advantage of some new features in the lattice package to change the default panel and prepanel functions used). With these changes, xhistogram() has been deprecated and histogram() has all the functionality of xhistogram().

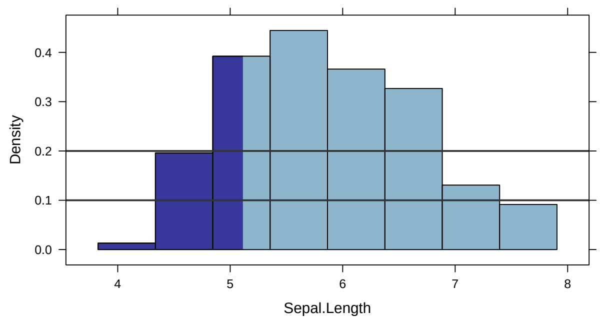
For example, one can choose the bins used for a histogram by setting values for center (defaults to 0) and width. Setting width to 1 is often useful for histograms of integer data with relatively few possible values.

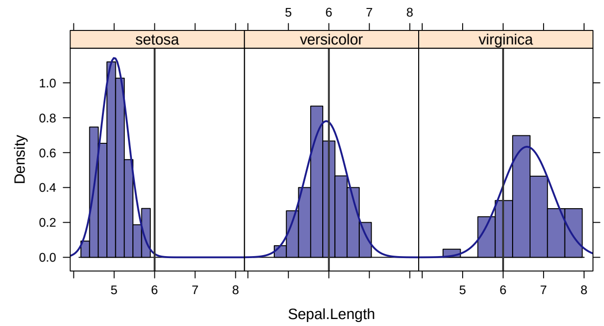
```
histogram(~week1, data = fumbles, width = 1)
```



Here are some additional features:

```
histogram(~Sepal.Length, data = iris, groups = Sepal.Length > 5, h = c(0.1,
    0.2))
histogram(~Sepal.Length | Species, data = iris, fit = "normal", v = 6)

## Loading required package:  MASS
```





1.3.3 mPlot()

For RStudio users, the `mosaic` package provides an interactive interface for creating a wide variety of `lattice` and `ggplot2` graphics using the `mPlot()` function. The code used to create these plots can subsequently be exported to the console and copied and pasted into other documents. `mPlot()` requires a data frame and a default plot to produce (scatter plot if none is specified) and allows the user to select variables and several other properties of the plots.

```
mPlot(iris)
mPlot(HELPrct, "density")
```

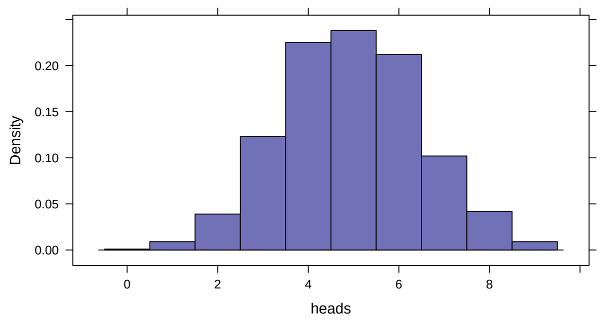
2 Chapter 2: Probability and Random Variables

2.1 The Lady Tasting Tea, rflip(), and do()

For those who want to introduce randomization methods early, the **`rflip()`** function provides a natural way to simulate coin tosses, and the `do()` function does things repeatedly and stores the results in a useful format. For example, the Lady Tasting Tea example can be handled using the following commands.

```
rflip(10)

##
## Flipping 10 coins [ Prob(Heads) = 0.5 ] ...
##
## T T H H H H H H T H
##
## Number of Heads: 7 [Proportion Heads: 0.7]

do(3) * rflip(10)
```

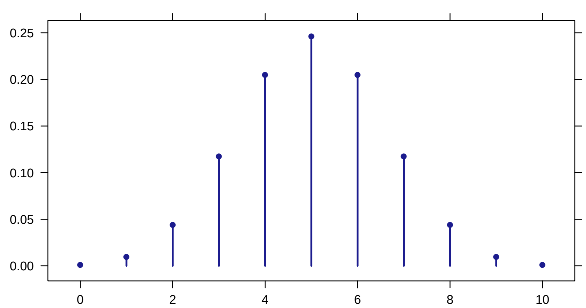
```
##    n heads tails prop
## 1 10     5     5  0.5
## 2 10     6     4  0.6
## 3 10     6     4  0.6

Flips <- do(1000) * rflip(10)
tally(~heads, data = Flips)

##
##     0     1     2     3     4     5     6     7     8     9 Total
##     1     9    39   123   225   238   212   102    42     9  1000

histogram(~heads, data = Flips, width = 1)
```

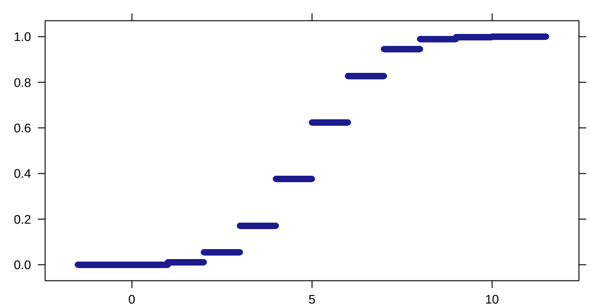


2.2 Plotting Distributions

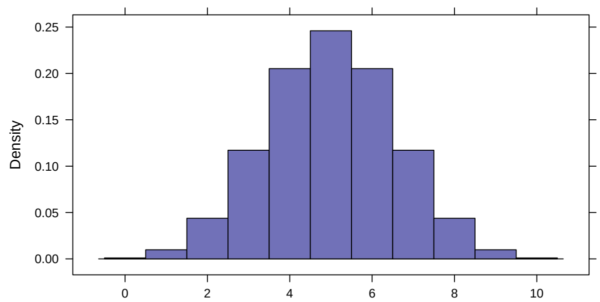
We can use `plotDist()` to plot discrete and continuous distributions in a number of ways.

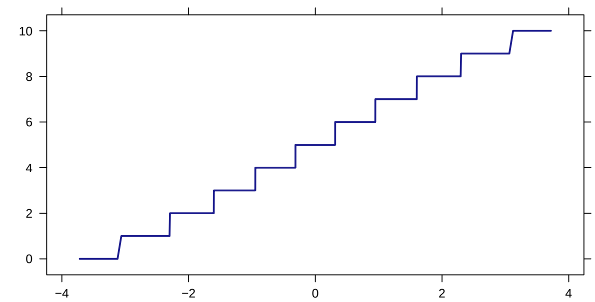
```
plotDist("binom", params = list(size = 10, prob = 0.5))
plotDist("binom", params = list(size = 10, prob = 0.5), kind = "cdf")
```



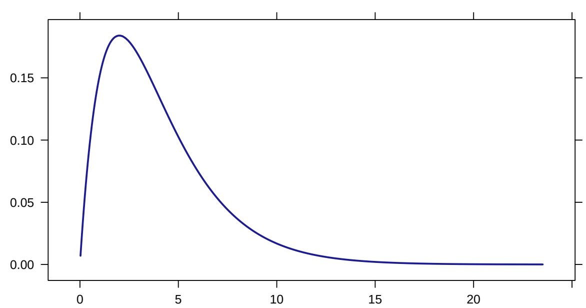


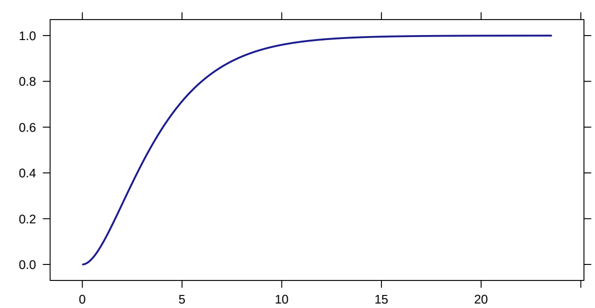

```
plotDist("binom", params = list(size = 10, prob = 0.5), kind = "hist")
plotDist("binom", params = list(size = 10, prob = 0.5), kind = "qq")
```





```
plotDist("chisq", params = list(df = 4))
plotDist("chisq", params = list(df = 4), kind = "cdf")
```





2.3 Formulas for `binom.test()`

```
binom.test(~sex, data = HELPrct)

##
##  Exact binomial test
##
## data:  HELPrct$sex
## number of successes = 107, number of trials = 453, p-value <
## 2.2e-16
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.1978 0.2781
## sample estimates:
## probability of success
##                 0.2362
```

Also, if you only want to extract the p-value or a confidence interval from a hypothesis test object, the `pval()` and `confint()` functions will do this for you.

```
pval(binom.test(~sex, data = HELPrct))

##   p.value
## 1.932e-30
```

```
confint(binom.test(~sex, data = HELPrct))

## probability of success                     lower                      upper
##                 0.2362                    0.1978                     0.2781
##                  level
##                 0.9500
```

3 Chapter 3: Continuous Distributions

3.1 makeFun()

For functions that are essentially algebraic in nature, the mosaic package provides a simplified method of defining functions via makeFun().

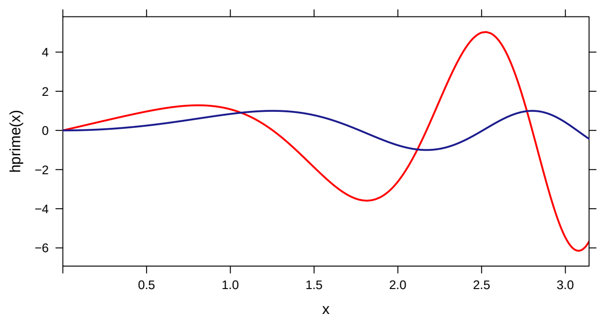
```
f <- makeFun(x^2 ~ x)
f(3)

## [1] 9

g <- makeFun(A * x^2 + B * x + C ~ x, A = 1, B = 2, C = 3)
g(2)

## [1] 11

g(2, A = 3, B = 2, C = 1)

## [1] 17
```

3.2 Calculus with D() and antiD()

The mosaic package provides functions for computing derivatives and antiderivates. Each of these functions returns a *function*, which can then be evaluated as needed. This is often easier than working with, for example, integrate() which returns an object from which the value of the integral must be extracted.

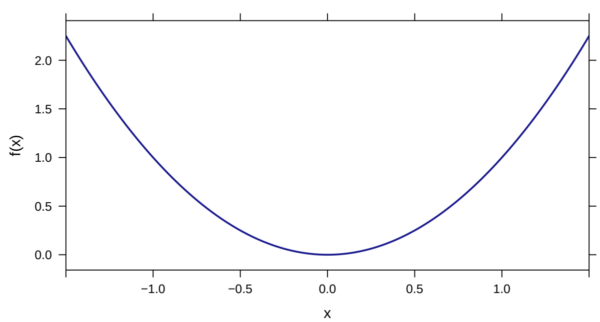
```
fprime <- D(f(x) ~ x)
fprime(2)

## [1] 4

fprime

## function (x)
## 2 * (x)

gprime <- D(g(x) ~ x)

## Warning:  Implicit variables without default values (dangerous!):  A, B, C

gprime(3)
```

```
## [1] 8

gprime(3, A = 3, B = 2, C = 1)

## [1] 20

h <- makeFun(sin(x^2) ~ x)
hprime <- D(h(x) ~ x)
plotFun(hprime(x) ~ x, col = "red", x.lim = c(0, pi))
plotFun(h(x) ~ x, x.lim = c(0, pi), add = TRUE)
```



Antiderivatives work similarly.

```
plotFun(f(x) ~ x, type = "h")
F <- antiD(f(x) ~ x)
F(1) - F(0)

## [1] 0.3333
```



4 Chapter 4: Parameter Estimation and Testing

4.1 t.test()

As was the case for `binom.test()`, we can now use formulas for the 1-sample t-test:

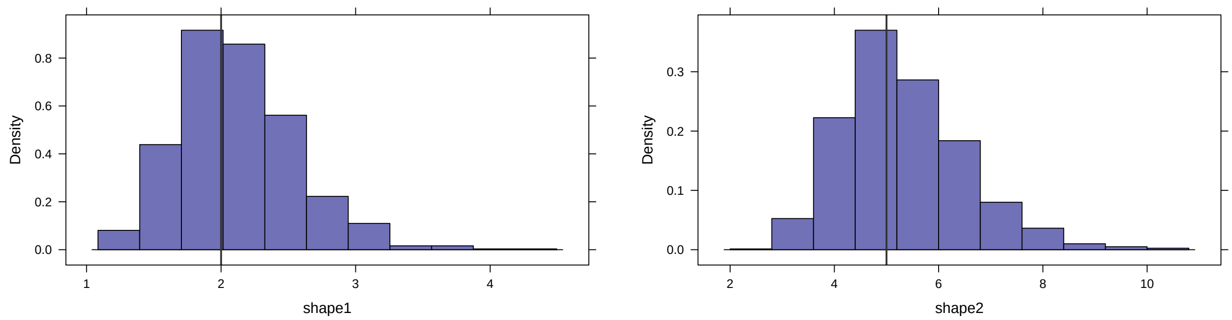
```
t.test(~age, data = HELPrct)

##
##  One Sample t-test
```

```
##
## data:  data$age
## t = 98.42, df = 452, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  34.94 36.37
## sample estimates:
## mean of x
##     35.65
```

4.2 Simulations with `do()`

The simulations done using `replicate()` can be done with `do()` instead. `do()` is slower because it does more packaging up of the results, but the format of the data returned is often easier to work with. Here's some code that could replace the code in Example 4.3.3.

```
snippet("mom-beta01")  # to define beta.mom

##
##
##  snippet(mom-beta01)
##  ------- ~~~~~~~~~~
##
## > beta.mom <- function(x,lower=0.01,upper=100) {
## +     x.bar <- mean (x)
## +     n <- length(x)
## +     v <- var(x) * (n-1) / n
## +     R <- 1/x.bar - 1
## +
## +     f <- function(a){             # note: undefined when a=0
## +         R * a^2 / ( (a/x.bar)^2 * (a/x.bar + 1) ) - v
## +     }
## +
## +     u <- uniroot(f,c(lower,upper))
## +
## +     return( c(shape1=u$root, shape2=u$root * R) )
## + }
##
## > x <- rbeta(50,2,5); beta.mom(x)
## shape1 shape2
##  2.166  5.481

results <- do(1000) * beta.mom(rbeta(50, 2, 5))
head(results, 2)

##   shape1 shape2
## 1  1.920  5.122
## 2  1.817  4.231

histogram(~shape1, data = results, type = "density", v = 2)
histogram(~shape2, data = results, type = "density", v = 5)
```



The advantages of using `do()` are even more pronounced when working with `lm()`. See https://github.com/rpruim/mosaic/blob/master/inst/PDFs/Resampling-vignette.pdf?raw=true for more examples using `do()`.

5 Chapter 5: Likelihood-Based Statistics

5.1 Zermelo's Algorithm

Section 5.6 focuses on the main ideas of the Bradley-Terry model and uses software to do the fitting. But it is not difficult to simplify the (large) system of partial differential equations involved in the maximum likelihood estimation into a form that leads to both a natural characterization of the MLE and an iterative algorithm for approximating the MLE that go back to Zermelo.

6 Chapter 6: Introduction to Linear Models

6.1 Converting models to functions with `makeFun()`

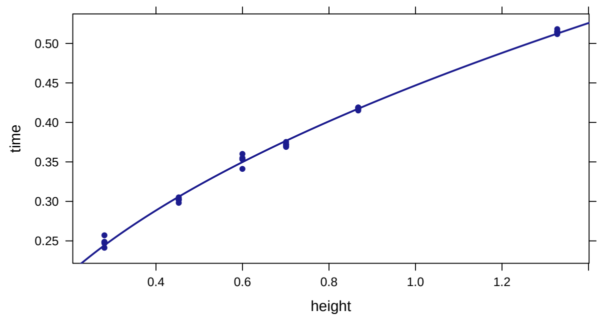
`makeFun()` can convert models made with `lm()` and `glm()` into functions. In both cases the functions produced is a wrapper around `predict()`. These functions take care of any transformations of the explanatory variables *but not transformations of the response variable*. In the case of `glm()` models, the default type is `"response"` rather than `"link"` since this is more natural for beginners.

```
ball.model <- lm(time ~ sqrt(height), data = balldrop)
time <- makeFun(ball.model)
time(height = 0.8)

##      1
## 0.4014

time(height = 0.8, interval = "confidence")

##      fit    lwr    upr
## 1 0.4014 0.3993 0.4035
```

6.2 And adding fitted functions to plots with `plotFun()`

We can add the model fit function to our scatter plot using `plotFun()`.

```r
xyplot(time ~ height, data = balldrop)
plotFun(time(height) ~ height, add = TRUE)
```



7 Chapter 7: More Linear Models

7.1 TukeyHSD() no longer requires use of aov()

```r
# TukeyHSD() can take a model created by lm()
model <- lm(pollution ~ location, data = airpollution)
TukeyHSD(model)

##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = x)
##
## $location
##                            diff    lwr   upr  p adj
## Plains Suburb-Hill Suburb   -6 -40.29 28.29 0.7643
## Urban City-Hill Suburb      15 -19.29 49.29 0.3019
## Urban City-Plains Suburb    21 -13.29 55.29 0.1601

# we can even let TukeyHSD build the model for us
TukeyHSD(pollution ~ location, data = airpollution)

##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = x)
##
## $location
##                            diff    lwr   upr  p adj
## Plains Suburb-Hill Suburb   -6 -40.29 28.29 0.7643
## Urban City-Hill Suburb      15 -19.29 49.29 0.3019
## Urban City-Plains Suburb    21 -13.29 55.29 0.1601
```