6. More Loops, Control Structures, and Bootstrapping

Ken Rice
Timothy Thornotn

University of Washington

Seattle, July 2013

In this session

We will introduce additional looping procedures as well as control structures that are useful in R. We also provide applications to bootstrapping.

- Repeat and While loops,
- If-Then and If-Then-Else structures
- Introduction to the bootstrap, with examples

Repeat loops

The repeat loop is an infinite loop that is often used in conjunction with a **break** statement that terminates the loop when a specified condition is satisfied. The basic structure of the repeat loop is:

```
repeat {
expression
expression
expression
if(condition)
break
}
```

Repeat loops

Below is a repeat loop for printing the square of integers from 1 to 10.

```
i <- 1
repeat {
print(i^2)
i <- i+1
if(i > 10)
   break
}
```

While loops

The while loop is used when you want to keep iterating as long as a specific condition is satisfied. The basic structure of the while loop is:

```
while (condition) {
    commands
}
```

While loops

Below is a while loop for printing out the first few Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13,..., where each number is the sum of the previous two numbers in the sequence.

```
a = 0
b = 1
print(a)
while (b < 50) {
    print(b)
    temp = a + b
    a = b
    b = temp
}
```

While loops

Below is a while loop that creates a vector containing the first 20 numbers in the Fibonacci sequence

```
x = c(0,1)
n=20
while (length(x) < n) {
    position = length(x)
    new = x[position] + x[position-1]
    x = c(x,new)
}
```

If-Then and If-Then-Else structures

Sometimes a block of code in a program should only be executed if a certain condition is satisfied. For these situations, *if-then* and *if-then-else* structures can be used:

The *if-then* structure has the following general form:

```
if (condition)

{expression
 expression}
```

The *if-then-else* structure has the following general form:

```
if (condition)
{expression
 expression} else
 {expression
  expression}
```

If-Then and If-Then-Else structures

Below is an *if-then-else* statement that takes the square root of the product of two numbers x and y if the product is positive:

```
x=3
y=7
  if( (x<0 & y<0)| (x>0 & y>0))
 {myval=sqrt(x*y)} else
 {myval=NA}
```

And the value of myval when x=3 and y=7 is:

```
> myval
[1] 4.582576
```

What is myval if x=2 and y=-10?

```
> myval
[1] NA
```

Introduction to bootstrapping

Bootstrapping is a very useful tool when the distribution of a statistic is unknown or very complex.

Bootstrapping is a non-parametric resampling method that allows for the computation of standard errors and confidence intervals, as well as hypothesis testing.

The method is often used when sample sizes are small and asymptotic distribution assumptions, such as normality, may not be appropriate.

"The bootstrap is a computer-based method for assigning measures of accuracy to sample estimates." [B. Efron and R. J. Tibshirani, An Introduction to the Bootstrap, Boca Raton, FL: CRC Press, 1994.]

Introduction to bootstrapping

Bootstrapping generally has the following three steps:

- Resample a given data set **with replacement** a specified number of time, where each "bootstrap sample" is the same size as the original sample

- Calculate a statistic of interest for each of the bootstrap samples.

- The distribution of the statistic from the bootstrap samples can then be used to obtain estimated standard errors, create confidence intervals, and to perform hypothesis testing with the statistic.

Example: bootstrapping the median

Bootstrapping can be easily implemented in R using loops.

The `sample(x, size, replace, prob)` function is very useful for resampling a given data set in `R`:

- The first argument of `sample` is a vector containing the data set to be resampled or the **indices of the data** to be resampled.

- The *size* option specifies the sample size, with the default being the same size as the data set being resampled.

- The *replace* option determines if the sample will be drawn with or without replacement where the default value is FALSE, i.e., sampling is performed without replacement.

Example: bootstrapping the median

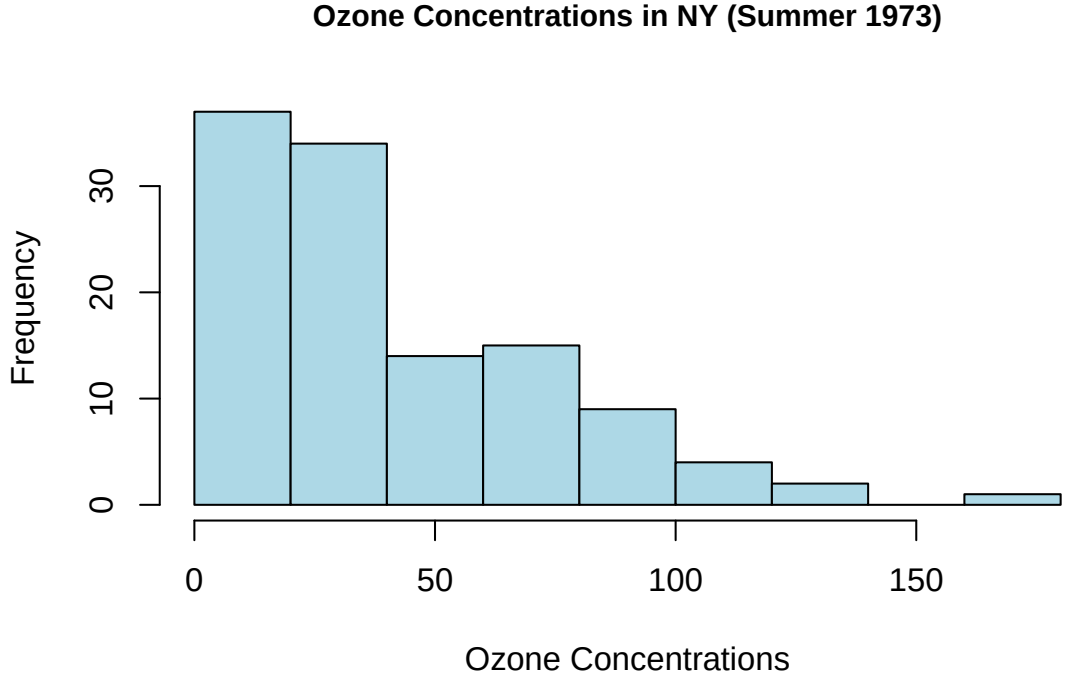
- The *prob* option takes a vector of length equal to the data set given in the first argument containing the probability of selection for each element of x. The default setting has each element with equal probability of being sampled.

In a typical bootstrapping situation the bootstrapping samples will be the same size as the data set being sampled and sampling will be done with replacement.

Example: bootstrapping the median

Let's consider the *airquality* dataset again. Below is a histogram of the daily ozone concentrations in New York, summer 1973.

```
hist(airquality$Ozone,col="lightblue",xlab="Ozone Concentrations",
main="Ozone Concentrations in NY (Summer 1973)")
```



What is the median ozone concentration level?

Example: bootstrapping the median

```
> median(airquality$Ozone)
[1] NA   # There are missing  daily ozone concentration values

> median(airquality$Ozone,na.rm=TRUE)
[1] 31.5
```

So the median is estimated to be 31.5.

Can we obtain a 95% confidence interval for the median? What is the distribution of the median ozone concentration?

Bootstrapping can be implemented for this!

Example: bootstrapping the median

We first obtain a vector of the ozone concentrations with missing values excluded:

```
ozone=airquality$Ozone[!is.na(airquality$Ozone)]
```

Using a `for()` loop, we can create 10,000 bootstrap samples and calculate the median for each sample:

```
nboot <-10000    #number of bootstrap samples
bootstrap.medians <-rep(NA, nboot)
set.seed(10)
for(i in 1:nboot){
bootstrap.medians[i]<-median(sample(ozone,replace=TRUE))
}
```

Example: bootstrapping the median

From the bootstrap medians we can obtain the .025 and .975 quantiles:

```
alpha=.05
sort(bootstrap.medians)[nboot*alpha/2]
sort(bootstrap.medians)[nboot*(1-alpha/2)]
```

```
> sort(bootstrap.medians)[nboot*alpha/2]
[1] 23.5
> sort(bootstrap.medians)[nboot*(1-alpha/2)]
[1] 39
```

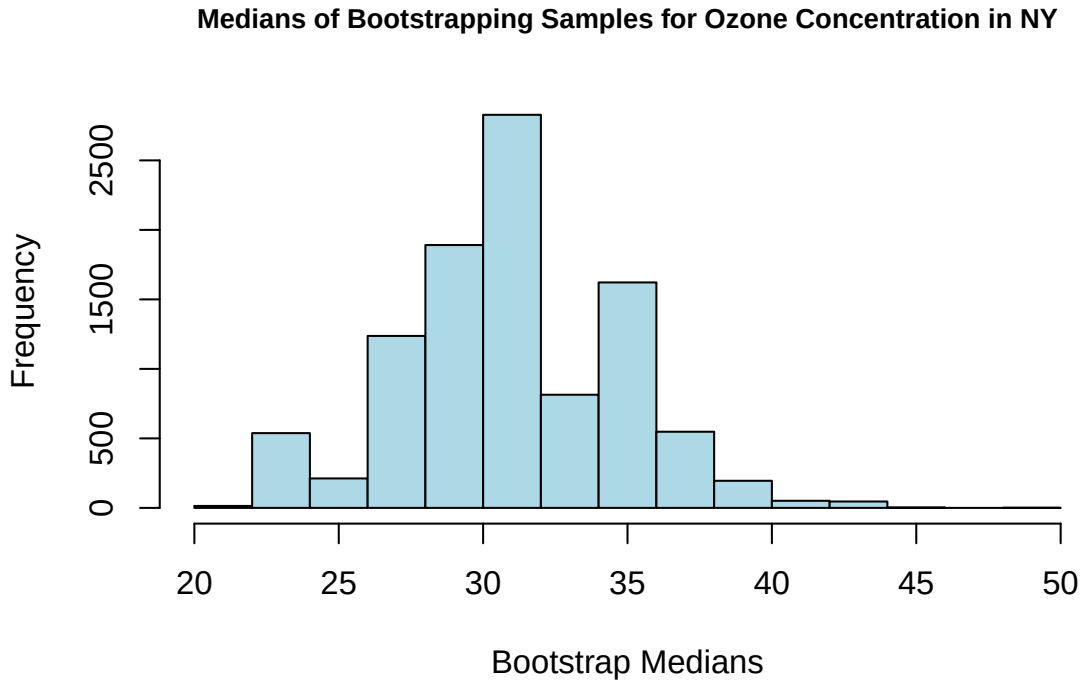
So our bootstrap 95% confidence interval for the median ozone concentration levels is **(23.5,39.0)**.

NB could also use `quantile(bootstrap.medians, c(0.025, 0.975))`

Example: bootstrapping the median

Below is a histogram of the medians from the 10,000 bootstrap samples;

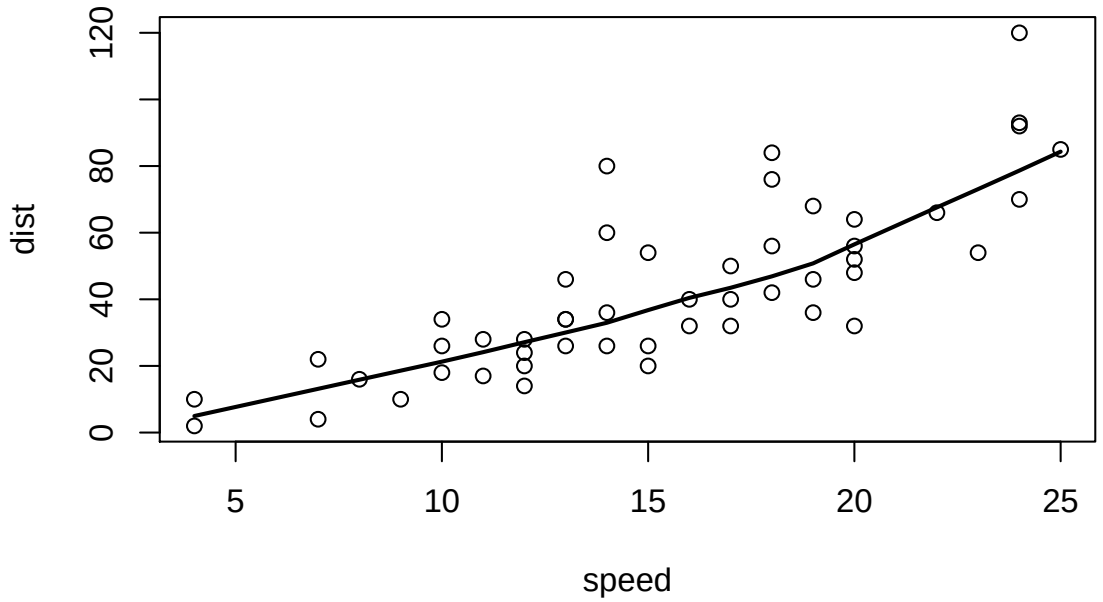
```
hist(bootstrap.medians,col="lightblue",xlab="Bootstrap Medians",
main="Bootstrap Medians for Ozone Concentrations in NY",cex.main=.8)
```



Example: bootstrap for lowess curve

Recall the `cars` data, and the line we put through it;

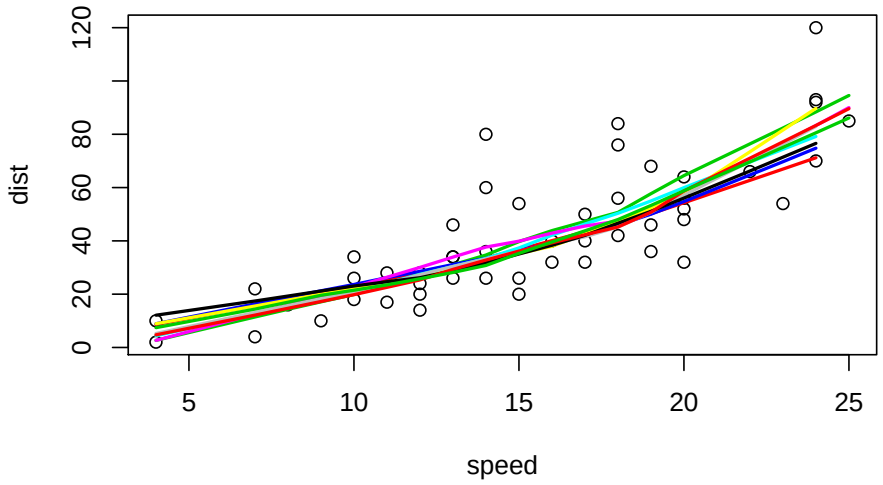
```
data(cars)
plot(dist~speed,data=cars)
with(cars, lines(lowess(speed, dist), col="tomato", lwd=2))
```



Example: bootstrap for lowess curve

To bootstrap the curve, a first step is to resample entire observations;

```
m=dim(cars)[1]     # obtain the sample size
nboot=20
for(i in 1:nboot){
   mysample <- sample(1:m,replace=T)
   with(cars, lines(lowess(speed[mysample], dist[mysample]),
                    col=(i+1), lwd=2)
        )}
```



Example: bootstrap for lowess curve

`lowess()` only produces output at the sampled points – so we extrapolate to the others using `approx()`;

```
nboot <- 1000
boot.speed <- matrix(NA, 1000,m)
set.seed(1314)
for(i in 1:nboot){
mysample <- sample(1:m,replace=T)
low1 <- with(cars, lowess(speed[mysample], dist[mysample]))
low.all <- approx(low1$x, low1$y, xout=cars$speed, rule=2)
boot.speed[i,] <- low.all$y
}
```

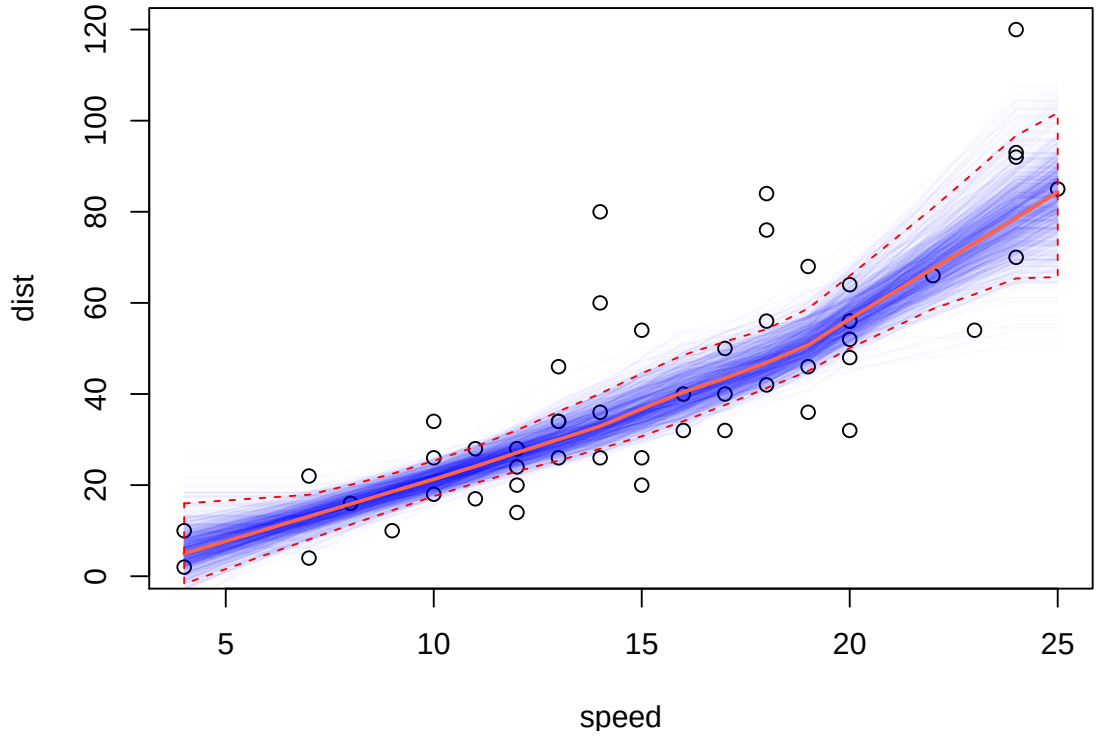
Now work out the lower and upper ranges of the lines, at each values of `speed`;

```
upper <- rep(NA, m)
lower <- rep(NA, m)
for(j in 1:m){
upper[j] <- quantile(boot.speed[,j], 0.975)
lower[j] <- quantile(boot.speed[,j], 0.025)}
```

Example: bootstrap for lowess curve

And make a nice picture;

```
plot(dist~speed,data=cars)
for(i in 1:nboot){
   lines(x=cars$speed, y=boot.speed[i,], col="#0000FF05")}
with(cars, lines(lowess(speed, dist), col="tomato", lwd=2))
polygon(x=c(cars$speed, rev(cars$speed)), y=c(upper, rev(lower)),
        density=0, col="red", lty=2)
```



Summary

- `while{}` and `repeat{}` are useful tools for looping until a condition is satisfied

- *if-then* and *if-then-else* structures allow blocks of code to be executed under different specified conditions

- bootstrapping is a powerful statistical technique when the distribution of a statistic is uknown

- bootstrapping can be very easily implemented in R using loops and the `sample()` function