In-Class Computing Task

Math 253: Statistical Computing & Machine Learning

Thursday Feb 11, 2016

Histogram function

Task 1

Write a function `myHistogram()` that packages up the commands you wrote for drawing a histogram into a function. It should draw the histogram (setting up a frame, as appropriate) and return the bin counts.

Give it an argument `fill` to set the color to fill in the bars. Give this a nice default value.

In addition, your program should draw a normal (a.k.a. gaussian) density in red over the histogram. The function to calculate density values is `dnorm()`. You'll need to give it the mean and standard deviation of the data being binned, which you can calculate directly from those data.

```
myHistogram <- function(v, fill = "gray") {
    dnorm(7)
}
```

Density estimation

Task 2

Work throught the following steps to generate a density plot.

You've got a set of numbers `v`. You want to plot the density of `v`.

```
v <- rnorm(10, mean = 100, sd = 1)
```

1. Pick a kernel wide enough to span the gaps between most points. Call the width the *bandwidth* (`bw`). A good starting value is the range divided by the square root of the number of values. Then create a function called `kernel()` that is a smooth density centered on any specified value `x`

```
bw <- diff(range(v))/sqrt(length(v))
kernel <- function(v, x) {
    dnorm(v, mean = x, sd = bw)
}
```

2. Create a set of values `x` at which to evaluate the density. This should extend several bandwidths to either side of the range of `v`.

```
x <- seq(min(v) - 5 * bw, max(v) + 5 * bw, length = 200)
```

3. Now, for each of the values in v, for each of the values in x, calculate the value of your kernel() function.

```
Dvals <- outer(v, x, FUN = kernel)
```

The result will be a matrix with length(v) rows and length(x) columns.

4. To convert this to a density, density, calculate the column sums of the matrix and divide by length(v). (See colSums().) You can check your result by confirming that the sum of dens divided by the interval between successive values of x gives 1.

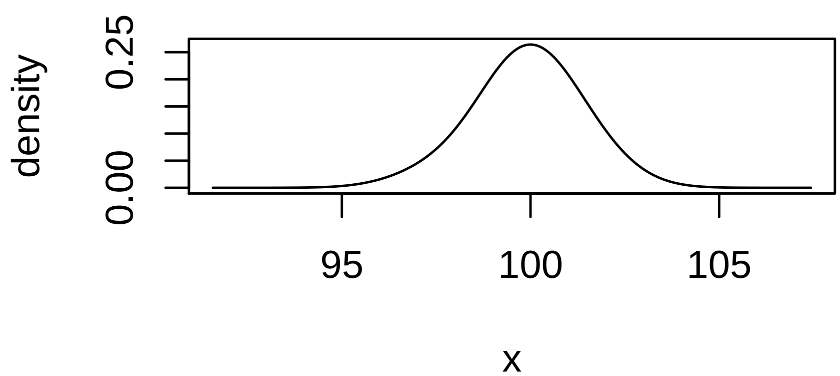
```
density <- colSums(Dvals)/nrow(Dvals)
```

5. Produce a data frame with components x and density.

```
Density <- data.frame(x = x, density = density)
```

6. You can plot that data frame to see the density estimate.

```
plot(density ~ x, data = Density, type = "l")
```



Task 3

Package your commands into a function, plotdensity() that takes v as an argument and produces a nice density plot. Allow the user to override xlim by including it as an argument with a default value of NULL.

The function should return the data frame from step (5) above with x and density. Since it's a graphics function, some people will want to avoid having to look at the return value; they will be interested just in the graphical *side effect*. You can use invisible() in place of return() to accomplish this.

```
plotdensity <- function(v, xlim = NULL) {
    bw <- diff(range(v))/sqrt(length(v))
    kernel <- function(v, x) {
        dnorm(v, mean = x, sd = bw)
    }
    x <- seq(min(v) - 5 * bw, max(v) + 5 * bw, length = 200)
    Dvals <- outer(v, x, FUN = kernel)
    density <- colSums(Dvals)/nrow(Dvals)
    Density <- data.frame(x = x, density = density)
    plot(density ~ x, data = Density, type = "l", xlim = xlim)
    invisible(Density)
}
```

Above and beyond

One of the nice features of R is the ability to use variables as arguments to functions without quoting them. For instance:

```
plot(y ~ x, data = MyData)
```

You can arrange your `plotdensity()` function to work this way. Here's a template without the details of drawing the density plot.

```
plotdensity <- function(v, data = parent.frame(), ...) {
    v <- substitute(v)  # a version of v that won't be evaluated until asked
    vname <- as.character(v)
    vvals <- eval(v, envir = data)
    # Your statements go here.  Use vvals as the data and vname as the label for the
    # x-axis
}
```