Demonstration of the Mass Production of QR Codes

Anthony Castellani

This is a demonstration of the use of R to build QR codes for various purposes. While there are many ways that QR codes can be generated, R has the advantage of enabling the user to produce these codes in large quantities with many variations, quickly and easily, and at no additional cost to the organization. Many free and paid apps are available for use on modern smartphones that can be used to scan these codes.

1. Load project packages

```r
library(png)
library(qrcode)
```

2. On the most basic level, the `qrcode` package allows the user to quickly and easily generate a QR code. The details of the code are dictated by the option choices inside the function call. For example, a QR code that takes the user to the homepage of the R project can be made as simply as:

```r
qrcode_gen("www.r-project.org")
```

3. Modification of the QR code is also simple in this case. One of the function's options is the error correction level (L, M, Q, or H), which are codes that represent the amount of

redundant data that will be encoded in the code. The lower the error correction level (towards L), the smaller the code can be displayed and still be machine-readable. The higher the error correction level (towards H), the more of the code can be rendered unreadable (either through the physical destruction of the code, or by the overlaying of a logo) before the code is no longer machine readable. There are other options in this function, including the modification of code color. Let us take advantage of the H level of code, and overlay a logo (in this case the logo of the R project).

- The logo can be downloaded here: www.r-project.org/logo/
- The logo licence can be found here: creativecommons.org/licenses/by-sa/4.0/

```r
qrcode_gen(dataString = "www.r-project.org", ErrorCorrectionLevel = "H")
rasterImage(readPNG("Rlogo.png"), xleft = 0.15, ybottom = 0.27, xright =
0.59, ytop = 0.95)
```

This demonstrates that a code scanned by a smartphone app can be used to take the user to a particular web address. QR codes are not limited, however, to the encoding of a web address. Another example of QR code usage might include contact information printed on a business card. Note the use of \n; each instance of \n will move the following block of text onto a new line when it is displayed on the user's smartphone.

```r
qrcode_gen("John Smith\n123 Main Street\nEverytown, USA 12345\n(555)555-
5555\nwww.mywebsite.com")
```

Another interesting use for QR codes involves linking to a web address, but in a much more targeted manner. Let us assume for a moment that an organization is unable or unwilling to harvest personal or geographic information directly from a user that lands on that organization's web site, but still desires to know some basic information regarding the circumstances that brought a user to that web site.

One way to do this is to distribute a number of QR codes, each one unique from the others, each one linked to exactly one web address that is only ever accessed by its respective QR code. If the organization then maintains a record of what codes are where and are connected to which QR code, then every time one of these controlled web addresses logs a hit, the organization knows that a particular QR code was used, and therefore what particular function was just executed, when and where.

One example might be a worker in the field who scans a unique code every time a task is performed (perhaps a worker inspecting wind turbines scans the code located on each turnbine with each successful inspection, with a unique web address logging the action and storing it in a database).

Another example might be a link to a landing page that immediately redirects to the organization's 24/7 online chat system. If the organization posts the code outside of its storefronts, then by monitoring the traffic to each of those unique landing pages the organization is positioned to not only provide continued customer service after hours, but to also know which store the customer was standing outside of when the code was scanned.

Or perhaps an organization has a link to a customer satisfaction survey posted prominently in the lobby of its branches. Identical surveys could be posted on many unique web addresses; tie a unique QR code to each unique web address, and the organization knows which branch produced which survey result.

Regardless of the reason, this use requires the production of many unique QR codes. Doing so one at a time can be tedious, especially for a large number of codes. With a little upfront work, however, producing any number of unique codes is trivial with R. What is needed is to write a custom function that can take a spreadsheet as an input and produce QR codes as the output.

First, build a function that creates QR codes. As was mentioned previously, the function `qrcode_gen` is quite malleable, meaning it has a lot of ways that it can be modified in order to make just the QR code that the user desires. As was previously demonstrated, the function `rasterImage` can be used to overlay a logo on the QR code. Also, another function that hasn't been discussed yet, the png function, can be used to export the QR code to a png file. The all-in-one QR code building function will need to have inputs for all options that we might want to modify in those three sub-functions.

```r
QRcodeBuilder <- function(fn, wi, he, un, re, ds, ecl, wc, bc, ln, le, bo,
ri, to) {
  png(filename = paste(fn, ".png", sep = ""), width = wi, height = he, units
= un, res = re)
  qrcode_gen(dataString = ds, ErrorCorrectionLevel = ecl,
                   wColor = wc, bColor = bc)
  rasterImage(readPNG(source = ln), xleft = le, ybottom = bo,
              xright = ri, ytop = to)
  dev.off()
}
```

In actual usage, `QRcodeBuilder` works just fine. For the purposes of an R Markdown demonstration, however, the QR codes need to remain inline. For that reason, I will use a modified version of `QRcodeBuilder` that is lacking the export functionality.

```r
QRcodeBuilder <- function(ds, ecl, wc, bc, ln, le, bo, ri, to) {
  qrcode_gen(dataString = ds, ErrorCorrectionLevel = ecl,
             wColor = wc, bColor = bc)
  rasterImage(readPNG(source = ln), xleft = le, ybottom = bo,
              xright = ri, ytop = to)
}
```

The next step is to load the data. For this we will use a csv file. Each column of the file will represent one option in the `QRcodeBuilder` function that can be modified. Each row will represent a unique QR code. In this way, it is easy to create a portfolio of unique codes by modifying a spreadsheet, a tool that many office workers have ready access to and can use. This makes this process more collaborative; perhaps one department plans how the codes should look and what should be stored in them, while another department is proficient in the use of R, and can run the code on receipt of the spreadsheet.

```
input.csv <- read.csv("QRcodeInput.csv", stringsAsFactors = FALSE)

head(input.csv)
```

```
     FileName width height units res                dataString
1 DemoQRcode1   7.5    7.5    in 192  First unique data string
2 DemoQRcode2   7.5    7.5    in 192 Second unique data string
3 DemoQRcode3   7.5    7.5    in 192  Third unique data string
  ErrorCorrectionLevel wColor bColor  LogoName xleft ybottom xright ytop
1                    H  white  black Rlogo.png  0.15    0.27   0.59 0.95
2                    H  white  black Rlogo.png  0.15    0.27   0.59 0.95
3                    H  white  black Rlogo.png  0.15    0.27   0.59 0.95
```

Once the data is loaded, the next step is to take advantage of the `Map` function's power to feed multiple inputs in parallel to a user-selected function. The first slot inside of the `Map` function will be filled with the previously-built QR code generating function, `QRcodeBuilder`. The remainder of the slots in the `Map` function will be filled with the column names of the csv file. When called, the `Map` function will feed one row at a time of all of the data object calls to `QRcodeBuilder`, which will take those inputs and build QR codes.

```
Map(QRcodeBuilder, input.csv$FileName, input.csv$width, input.csv$height,
    input.csv$units, input.csv$res, input.csv$dataString,
    input.csv$ErrorCorrectionLevel, input.csv$wColor, input.csv$bColor,
    input.csv$LogoName, input.csv$xleft, input.csv$ybottom, input.csv$xright,
    input.csv$ytop)
```

While the previous call to the `Map` function is the right move in an actual production scenario, our modified `QRcodeBuilder` needs a similarly shortened call to `Map`.

By producing three codes at one time, the final demonstration will illustrate the ability of the user to produce an arbitrarily large body of unique QR codes. Note the subtitles indicating the contents of these QR codes.

```
Map(QRcodeBuilder, input.csv$dataString, input.csv$ErrorCorrectionLevel,
    input.csv$wColor, input.csv$bColor, input.csv$LogoName, input.csv$xleft,
    input.csv$ybottom, input.csv$xright, input.csv$ytop)
```

```
$`First unique data string`
NULL

$`Second unique data string`
NULL

$`Third unique data string`
NULL
```