Practical 12: Functions and Loops in R

An Introduction to Spatial Data Analysis and Visualisation in R - Guy Lansley & James Cheshire (2016)

This extra practical provides a very basic introduction to some programming techniques which can make tasks in R less time consuming if they involve repetition or iterating operations for multiple data. Data for the practical can be downloaded from the **Introduction to Spatial Data Analysis and Visualisation in R** (https://data.cdrc.ac.uk/tutorial/an-introduction-to-spatial-data-analysis-and-visualisation-in-r) homepage.

In this tutorial we will:

- Learn how to create user defined functions in R
- Run a simple for loop
- Learn how to implement if…else statements

First, we must set the working directory and load the practical data.

```
# Set the working directory
setwd("C:/Users/Guy/Documents/Teaching/CDRC/Practicals")

# Load the data. You may need to alter the file directory
Census.Data <-read.csv("practical_data.csv")
```

We will also need to load the spatial data files from the previous practicals.

```
# load the spatial libraries
library("sp")
library("rgdal")
library("rgeos")
library("tmap")

# Load the output area shapefiles
Output.Areas <- readOGR(".", "Camden_oa11")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "Camden_oa11"
## with 749 features
## It has 1 fields
```

```
# join our census data to the shapefile
OA.Census <- merge(Output.Areas, Census.Data, by.x="OA11CD", by.y="OA")
```

One of the core advantages of R and other command line packages over traditional software is the ability to automate several commands. You can tell R to repeat your methodological steps with additional variables without the need to retype the entire code again.

There are two ways of automating codes. One means is by implementing user-written functions, another is by running for loops.

User-written functions

User-written functions (https://www.datacamp.com/community/tutorials/functions-in-r-a-tutorial#gs.PpZ=cIk) allow you to create a function which runs a series of commands, it can include and input and output variable. You must also name the function yourself. Once the function has been run, it is saved in your work environment by R and you can run data through it by calling it as you would do with any other function in R (I.e. myfunction(data)).

The general structure is provided below:

```
# to create a new function
myfunction <- function(object1, object2, ... ){
statements
return(newobject)
}

# to run the function
Output <- myfunction(data)
```

When you create a function you do not need the name of the variables you use. You simply create a temporary object name (say 'x' for example), then when you run a data object through your function, it will take the place of x.

So for purposes of example, we will work on rounding our data first.

To round a variable we can simply use the `round()` function. It only requires us to input a data object and to specify how many decimal places we want our data rounded to.

So for one variable it would be:

```
# rounds data to one decimal place
round(Census.Data$Qualification, 1)
```

```
##   [1] 73.6 69.9 67.6 60.8 66.0 74.2 62.4 60.4 70.1 66.7 66.7 64.5 73.5 65.4
##  [15] 72.9 74.8 73.7 69.1 58.2 23.0 72.0 64.9 70.5 70.5 76.1 65.9 70.3 72.4
##  [29] 66.2 65.3 80.0 79.3 70.3 65.6 68.9 65.7 72.8 70.8 36.3 25.5 32.3 72.4
##  [43] 66.2 57.9 29.4 26.1 26.4 65.2 76.2 55.7 58.5 35.2 26.6 30.9 34.5 32.5
##  [57] 56.9 31.7 49.3 32.1 33.5 44.5 51.6 57.1 47.2 64.2 50.9 67.1 62.6 68.2
##  [71] 62.7 46.7 60.7 57.2 72.9 27.6 65.6 68.8 62.6 31.7 25.9 26.1 38.5 20.9
##  [85] 44.9 58.0 37.0 40.3 66.2 70.7 62.0 42.9 55.6 34.3 76.4 58.2 40.0 71.8
##  [99] 46.3 69.7 66.8 76.9 67.9 47.2 72.6 57.3 60.4 78.7 50.7 48.7 43.4 61.9
## [113] 46.7 44.8 70.6 74.7 67.1 26.6 43.1 31.1 61.5 49.3 66.3 62.9 66.5 56.5
## [127] 35.9 29.4 47.0 42.1 57.1 59.5 27.1 27.3 37.1 24.5 30.6 58.7 48.0 39.9
## [141] 39.8 43.3 39.6 59.4 63.3 66.2 60.4 57.3 47.0 56.1 63.4 41.9 59.5 64.5
## [155] 56.3 51.5 48.8 65.0 50.7 67.9 50.2 65.0 50.2 30.9 25.0 51.1 61.3 59.9
## [169] 37.0 58.4 58.6 65.7 43.3 46.2 46.6 69.4 63.1 64.8 65.5 64.1 70.4 70.6
## [183] 67.5 67.6 62.2 70.5 56.0 69.6 55.1 65.9 65.7 66.1 70.6 57.9 64.8 28.5
## [197] 66.7 61.4 55.7 71.5 60.7 59.1 76.6 66.4 62.9 72.3 63.7 70.5 70.1 58.9
## [211] 79.4 61.1 59.6 70.4 70.0 68.8 68.3 55.1 37.0 58.9 60.5 68.0 65.0 59.4
## [225] 72.7 53.6 61.2 55.2 71.2 71.2 48.9 67.6 68.3 46.2 27.1 21.2 32.4 72.3
## [239] 24.7 35.6 71.9 27.0 23.4 19.8 40.4 23.6 28.2 69.3 37.5 18.1 55.5 65.8
## [253] 29.3 60.6 67.4 64.5 57.9 66.9 65.2 67.2 72.9 75.8 58.4 70.0 59.8 42.8
## [267] 47.8 61.5 34.4 27.7 43.7 23.2 28.3 73.8 76.9 50.8 70.6 66.2 71.6 55.6
## [281] 68.0 51.9 63.1 76.8 74.9 69.2 68.2 73.0 73.1 71.1 72.7 63.6 76.6 70.5
## [295] 66.0 69.8 69.5 75.5 70.1 72.9 75.5 70.5 37.0 71.2 73.2 73.9 46.0 68.2
```

```
## [309] 70.1 67.3 80.1 73.1 68.4 29.6 29.3 47.3 38.7 42.9 37.0 46.1 46.1 52.3
## [323] 28.5 29.7 30.3 23.3 70.7 45.4 27.8 29.7 27.5 60.2 18.6 16.4 38.9 73.6
## [337] 28.5 23.6 70.7 45.7 77.7 40.9 66.7 58.7 30.9 19.4 63.6 55.5 54.8 54.0
## [351] 33.1 66.3 72.5 64.6 50.4 60.2 57.7 20.9 55.0 41.0 32.3 63.7 53.2 70.0
## [365] 65.2 59.7 34.0 40.7 66.6 64.4 61.2 35.4 45.8 66.4 41.5 41.9 66.8 57.7
## [379] 51.7 65.6 66.9 52.9 71.3 67.4 59.9 68.9 39.4 39.8 69.0 37.4 67.2 56.5
## [393] 42.7 68.9 42.4 23.1 35.8 53.6 50.4 50.9 44.4 67.7 26.6 61.5 74.3 53.1
## [407] 31.7 43.7 31.6 37.6 42.2 63.5 26.8 30.2 43.8 45.1 59.5 26.8 40.1 68.3
## [421] 32.4 37.8 56.0 31.1 43.0 35.1 51.6 54.9 37.5 37.1 52.1 36.1 58.4 27.9
## [435] 44.2 25.3 49.4 52.5 65.2 62.3 60.6 48.1 59.8 67.4 45.5 42.3 58.5 60.9
## [449] 60.6 63.6 56.5 50.3 56.0 59.5 64.7 52.5 66.3 59.1 68.1 49.2 61.3 59.3
## [463] 56.9 45.0 68.2 80.1 57.4 69.8 28.7 38.3 70.6 30.8 32.7 32.3 46.5 22.5
## [477] 32.6 42.7 41.3 21.4 37.2 35.6 40.8 64.9 62.3 67.4 57.9 48.8 47.8 42.2
## [491] 71.4 51.4 66.9 52.4 41.3 33.6 23.8 63.3 61.8 44.1 26.7 22.6 67.6 28.0
## [505] 29.3 25.5 57.4 41.8 42.9 30.3 20.3 51.6 27.5 48.3 32.7 41.5 56.5 54.8
## [519] 54.2 47.1 65.8 57.7 67.1 69.3 27.9 35.5 40.0 29.8 37.5 18.8 47.2 20.8
## [533] 34.2 66.2 31.5 21.3 28.2 32.8 57.3 28.7 27.9 28.7 22.3 36.4 33.8 30.7
## [547] 67.1 65.6 21.6 23.2 30.6 25.0 36.5 59.3 28.8 23.4 37.1 11.6 17.4 34.5
## [561] 45.3 41.8 22.7 38.5 23.1 18.8 40.2 28.8 47.1 45.3 65.1 49.2 54.2 57.0
## [575] 52.2 43.9 52.2 53.1 24.5 59.5 37.2 28.4 34.8 22.8 24.5 33.2 21.6 16.5
## [589] 29.4 33.3 13.5 27.1 34.7 22.5 17.9 14.7 46.2 21.0 53.1 16.3 17.8 44.9
## [603] 60.9 59.5 24.2 34.9 40.8 25.7 20.5 18.8 27.6 29.9 28.0 19.9 38.8 30.2
## [617] 34.2 26.5 34.0 21.9 25.2 17.3 16.2 25.5 12.8 20.0 24.6 33.2 26.8 25.9
## [631] 72.0 71.8 74.7 70.5 63.8 50.4 71.7 63.5 68.7 70.8 63.1 34.5 50.9 57.3
## [645] 51.5 48.6 49.5 47.9 46.7 36.0 57.3 61.3 72.3 41.2 69.6 54.4 66.8 79.8
## [659] 47.4 65.9 69.3 35.7 74.0 73.5 71.0 72.2 61.2 80.4 30.2 78.6 52.8 65.8
## [673] 60.0 77.9 75.9 48.8 75.1 79.5 54.4 64.3 65.9 64.8 65.6 62.3 69.3 62.5
## [687] 41.8 62.6 76.6 74.8 49.1 40.5 67.0 57.8 32.1 49.5 53.7 48.3 62.9 42.3
## [701] 61.4 33.0 48.3 64.4 43.2 61.6 64.4 67.1 66.7 73.4 70.6 72.5 70.3 79.7
## [715] 79.4 64.7 64.2 29.4 26.8 13.7 55.4 30.2 17.8 63.0 55.6 57.9 61.6 61.9
## [729] 35.7 31.0 75.6 70.9 80.0 54.2 61.0 46.7 33.5 42.2 68.6 62.4 84.6 88.1
## [743] 60.4 52.8 37.1 50.7 53.2 45.3 24.7
```

This is very straight forward. We will now put this in a new function called “myfunction” which will comprise this code.

```
# function is called "myfunction", accepts one object (to be called x)
myfunction <- function(x){

# x is rounded to one decimal place, creates object z
z <- round(x,1)

# object z is returned
return(z)

# close function
}
```

Now run the function, it should appear in your environment panel. To run it we simply run data through it like so:

```
# lets create a new data frame so we don't overwrite our original data object
newdata <- Census.Data

# runs the function, returns the output to new Qualification_2 variable in newdata
newdata$Qualificaton_2 <- myfunction(Census.Data$Qualification)
```

This example is very simple. However, we can add further functions into our function to make it undertake more tasks. In the demonstration below, we also run logarithmic transformation prior to rounding the data.

```
# creates new function which logs and then rounds data
myfunction <- function(x){

z <- log(x)
y <- round(z,4)

return(y)

}

# resets the newdata object so we can run it again
newdata <- Census.Data

# runs the function
newdata$Qualification_2 <- myfunction(Census.Data$Qualification)
newdata$Unemployed_2 <- myfunction(Census.Data$Unemployed)
```

So here we create the z variable first, then run it through a second function to produce a y variable. However, as the code is run in order, we could give both z and y the same name to overwrite them as we run the function. I.e…

```
myfunction <- function(x){

z <- log(x)
z <- round(z,4)

return(z)

}
```

Now we will try something more advanced. We will enter our mapping code using tmap into a function so we are able to create maps more easily in the future. Here we have called the function ‘map’. The code has three arguments this time (of course we could include more if we wanted to add more customisations) they are described in the table below.

Object label	**Argument**
x	A polygon shapefile
y	A variable of the shapefile
z	A Colour palette

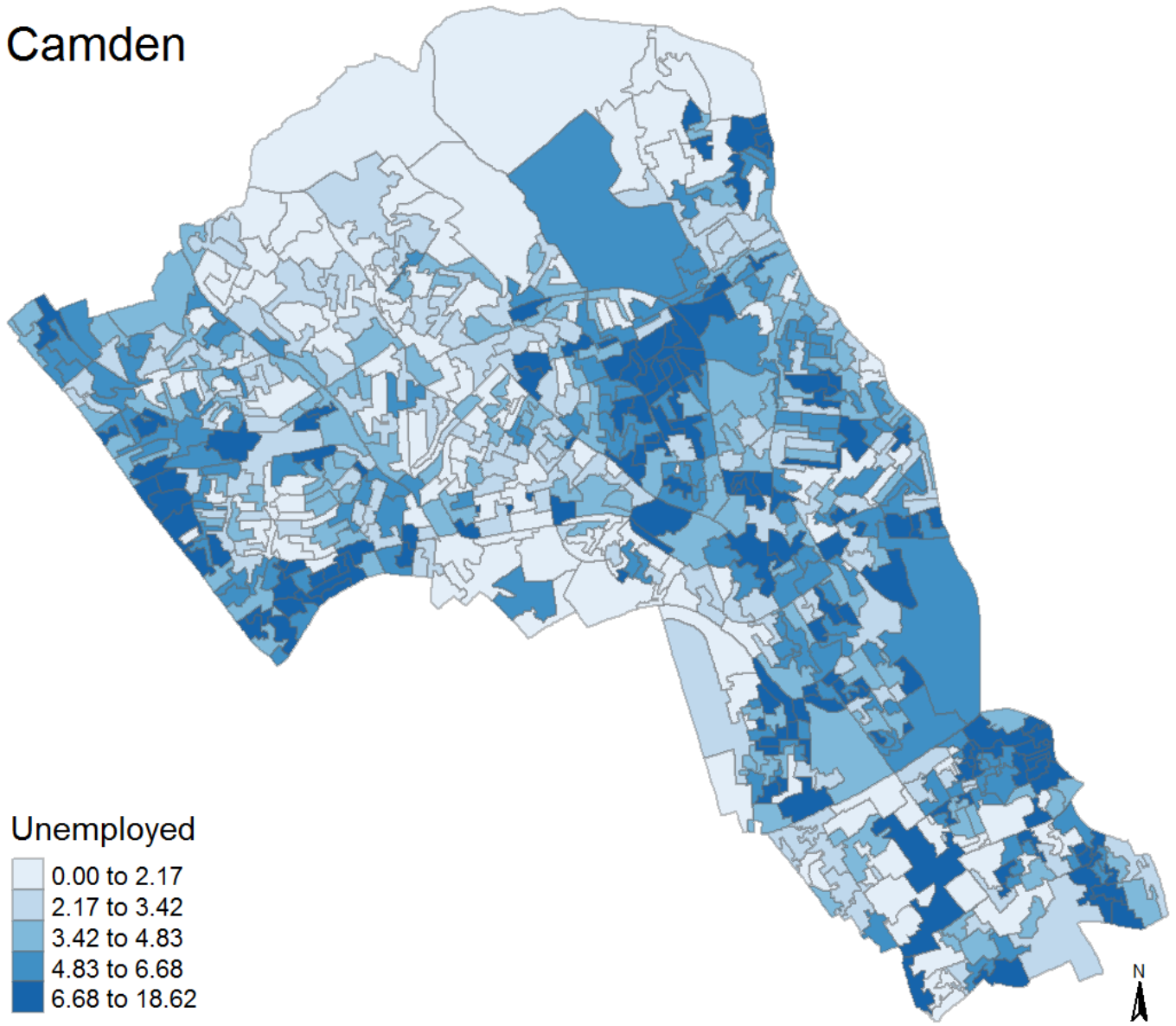
So the code is as follows…

```
# map function with 3 arguments
map <- function(x,y,z){

tm_shape(x) + tm_fill(y, palette = z, style = "quantile") + tm_borders(alpha=.4) +
tm_compass(size = 1.8, fontsize = 0.5) +
tm_layout(title = "Camden", legend.title.size = 1.1, frame = FALSE)

}
```

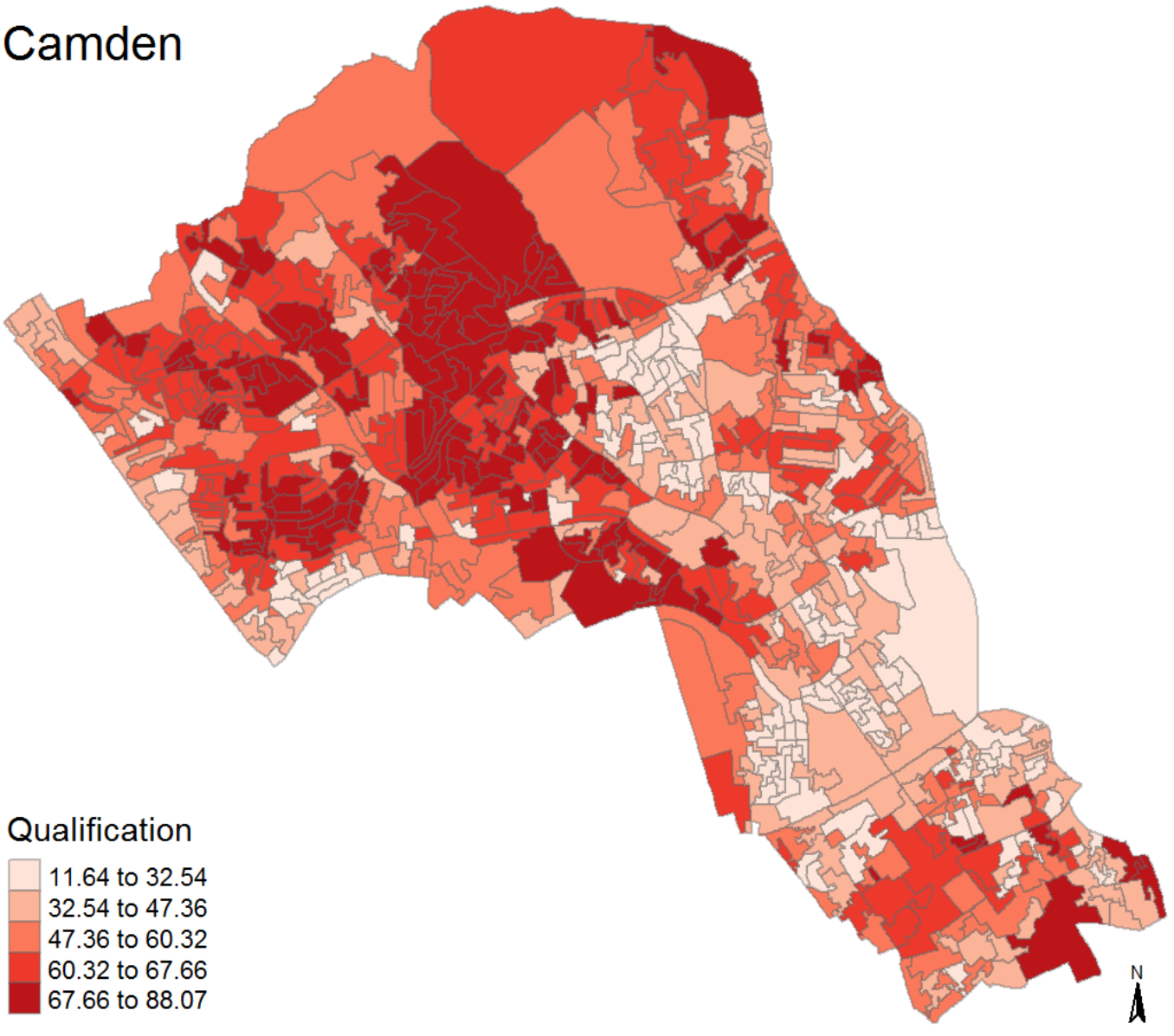
Now to run the *map* function, we can very simply call the function and enter the three input parameters (x,y,z). Running our function is an easier way to create consistent maps in the future as it requires only one small line of code to call it.

We can now use the function to create a map for any spatial polygon data frame which contains numerical data. All we have to do is call the function and enter the three arguments. There are two examples below.

```
# runs map function, remember we need to include all 3 arguments of the function
map(OA.Census, "Unemployed", "Blues")
```




```
map(OA.Census, "Qualification", "Reds")
```



Loops

A loop (https://www.datacamp.com/community/tutorials/tutorial-on-loops-in-r#gs.Jg1AjKM) is a means to repeat code under defined conditions. This is a great way for reducing the amount of code you may need to write. For example, you could use the loop to iterate a series of commands through a number of different positions (i.e. rows) in a data frame.

For loop

The most basic means of looping commands in R is known as the for loop. To implement it, you run the `for()` function and within the parameters, you define the sequence through which the loop will input data. Usually, an argument is assigned (i) and given a range of numerical positions (i.e. `in 2:5` which runs from position 2 to position 5). You next enter the commands which the loop will run within curly brackets following the parameters. The defined argument (i) is used to determine the input data for each iteration. This has been demonstrated in the example below which loops the code for four columns of our data frame.

Note in this example we are running the loop from columns 2 onwards as column 1 does not contain our numeric data

```
#create a new data frame of the same properties as our data file
newdata <- Census.Data

# a for loop where i iterates from 2 to 5
for(i in 2:5){
 # i is used to identify the column number
 newdata[, i] <- round(Census.Data[,i], 1)

}

# open the new data
View(newdata)
```

We can also use the `ncol()` function if you do not know the total number of columns or accept that the number may change under certain circumstances.

```
# use ncol to determine the number of columns
for(i in 2: ncol (Census.Data)){

 newdata[, i] <- Census.Data[,i]/100

}
```

We will be using the ratio data from newdata in the upcoming section.

If. else statements

There are various commands we can run within for loops to edit their functionality. If-else statements allow us to change the outcome of the loop depending on a particular statement. For instance, if a value is below a certain threshold, you can tell the loop to round it up, otherwise (else) do nothing.

This time we are going to run the loop for every row in one variable to recode them to “low” and “high” labels. In this example, we are asking if the value in our data is below 0.5. If it is then we are giving it a “low” label. The else argument determines what R will do if the value does not meet the criterion, in this case it is given a “high” label.

```
# creates a new newdata object so we have it saved somewhere
newdata1 <- newdata

for(i in 1:nrow(newdata)){

if (newdata$White_British[i] < 0.5) {
   newdata$White_British[i] <- "Low";

} else {
   newdata$White_British[i] <- "High";

}
}
```

We can also input multiple else statements.

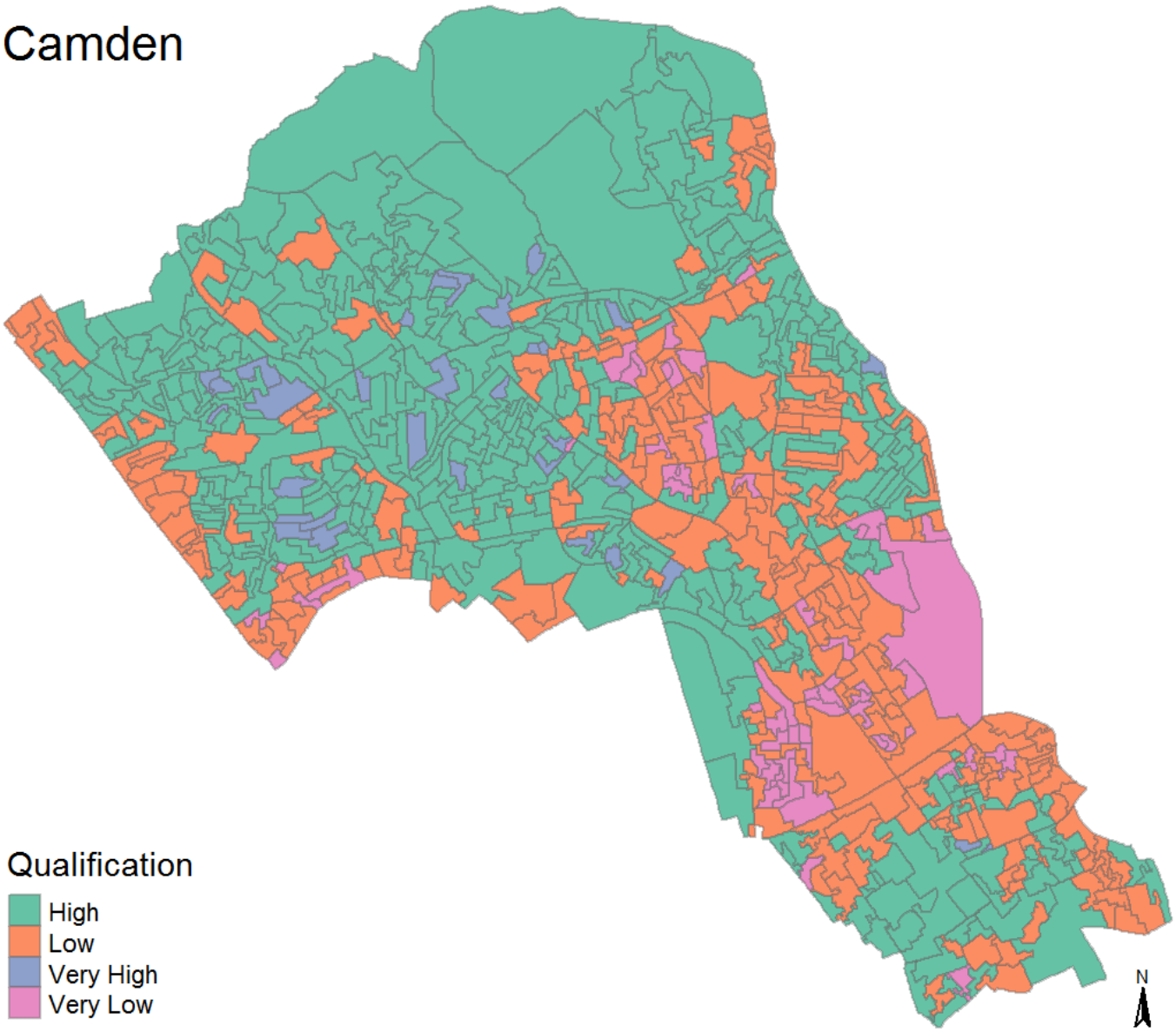
```
# copies the numbers back to newdata so we can start again
newdata <- newdata1

for(i in 1:nrow(newdata)){

if (newdata$White_British[i] < 0.25) {
   newdata$White_British[i] <- "Very Low";

} else if (newdata$White_British[i] < 0.50){
   newdata$White_British[i] <- "Low";

} else if (newdata$White_British[i] < 0.75){
   newdata$White_British[i] <- "High";

} else {
   newdata$White_British[i] <- "Very High";

}
}
```

We can also nest multiple loops if we wish to loop through two different arguments. In this example, we are going to loop both the columns and rows for our newdata data frame. This will reassign all of our variables in the data into four interval labels.

```
# copies the numbers back to newdata so we can start again
newdata <- newdata1


for(j in 2: ncol (newdata)){

for(i in 1:nrow(newdata)){

if (newdata[i,j] < 0.25) {
   newdata[i,j] <- "Very Low";

} else if (newdata[i,j] < 0.50){
   newdata[i,j] <- "Low";

} else if (newdata[i,j] < 0.75){
   newdata[i,j] <- "High";

} else {
   newdata[i,j] <- "Very High";

}
}
}
```

We can join the newdata data frame to our output area shapefile to map it using our earlier generated *map* function.

```
# merge our new formatted data with the output areas shapefile
shapefile <- merge(Output.Areas, newdata, by.x = "OA11CD", by.y = "OA")

# runs our predefined map function
map(shapefile, "Qualification", "Set2")
```



The rest of the online tutorials in this series can be found at: https://data.cdrc.ac.uk/tutorial/an-introduction-to-spatial-data-analysis-and-visualisation-in-r (https://data.cdrc.ac.uk/tutorial/an-introduction-to-spatial-data-analysis-and-visualisation-in-r)