

Practice exercise2

Matt Boone

October 13, 2015

We're back again, time for your weekly practice, practice, practice. Again, there's more than one way to do anything in R (as we've seen in our dplyr lesson taught by Des). Work your way as far down as you can go before you start struggling. **DONT GIVE UP**. If things get too hard, feel free to google or check the help files for various functions. R is about googling as much as anything else!

Today we focus on **subsetting** otherwise known as taking a data set and filtering it to our choosing. There's **many** different ways to do this

Easy

I want to buy a car. Say we have a data set of cars and their various performance statistics. Hey wait we do have that, it comes free with R! Its called the 'mtcars' data set. It's automatically loaded when you type in 'mtcars'

Now say I want to use this data set to help me decide what car to buy. I want to subset this list so I can see the 6 cylinder cars (cyl), but I'm only interested in seeing each cars horsepower (hp) and miles per gallon (mpg). Subset the 'mtcars' data set so I only have 6 cylinder cars and only the two columns 'hp' and 'mpg'

```
head(mtcars)
```

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Solution

I'm going to use the subset function, which is fairly straight forward

```
subset(mtcars, cyl==6, select=c(hp, mpg))
```

##	hp	mpg
## Mazda RX4	110	21.0
## Mazda RX4 Wag	110	21.0
## Hornet 4 Drive	110	21.4
## Valiant	105	18.1
## Merc 280	123	19.2
## Merc 280C	123	17.8
## Ferrari Dino	175	19.7

but we can do it without functions:

```
mtcars[mtcars$cyl==6 , c('hp','mpg') ]
```

```
##           hp  mpg
## Mazda RX4    110 21.0
## Mazda RX4 Wag 110 21.0
## Hornet 4 Drive 110 21.4
## Valiant      105 18.1
## Merc 280      123 19.2
## Merc 280C     123 17.8
## Ferrari Dino  175 19.7
```

We also can do this using dplyr!

```
###we have to add the row names as its own column first
mtcars$name <- row.names(mtcars)
mtcars %>% filter(cyl==6) %>% select(name, hp, mpg)
```

```
##           name  hp  mpg
## 1   Mazda RX4  110 21.0
## 2  Mazda RX4 Wag 110 21.0
## 3 Hornet 4 Drive 110 21.4
## 4     Valiant   105 18.1
## 5     Merc 280   123 19.2
## 6     Merc 280C  123 17.8
## 7   Ferrari Dino  175 19.7
```

Medium

Now I'm a picky person and I have a lot of needs and wants. What I really want is a car that satisfies these 4 criteria.

1. I do not want a heavy car ('wt' is less than 4kg)
2. I want a 4 or a 6 cylinder engine ('cyl' means cylinder)
3. I want it to get at least 20 miles to the gallon ('mpg')
4. Because I am a scientist, I want the car that has the highest value of this metric I call the 'awesome' metric, which is obviously:

$$\text{mpg}^{\text{hp}} / ((10\text{wt})^{(20\text{cyl})})$$

At the end I want it to **only** show me the car that fits all of these criteria. Subset the data set so it shows me what car I'm going to buy.

Solution

Just like before, we'll use subset first, but be mindful of your parenthesis, R will read this whole statement as is

```
final <- subset(mtcars, wt<4 & (cyl==4|cyl==6) & mpg >=20)
```

To calculate our awesome metric I'm going to go a step up and use the 'with' function, it works similar to 'mutate' in dplyr and lets us to do something to a data set without actually messing with the original data set. You first give it the data.frame, then within the {} you tell it what you want to do inside that dataset. Here we're going to calculate our awesome metric. The great thing about doing it this way is we don't have to say

```
final$awesome <- final$mpg^ final$hp ...etc
```

```
awesome <- with(final , {
  mpg^(hp) /(((10*wt)^(20*cyl))
})
```

Finally we're going to use the which.max function, which tells us the position of which number is the highest of our awesome vector (as opposed to max which gives us the actual value)

```
final[which.max(awesome),]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
##                name
## Lotus Europa Lotus Europa
```

Bonus (Medium-Hard difficulty): I want a Mercedes (the 'Merc' means Mercedes) and instead of my awesome metric I want the highest horsepower possible ('hp'). In this situation I want my final dataframe to only contain **ONE** row, my wining entry. So you must choose the name 'Merc' as well, **WITHOUT** manually typing in the number of the row for the Mercedes!

Solution This problem might be harder for people than I anticipated. I'm hoping that through google y'all have found the function 'grep' and 'grepl'. These search using something we call regularExpressions which is a way for codes to 'search' through character strings for something of interest. This example is pretty straight forward as we're only looking for the phrase 'merc' in each row.name. 'grepl' shows us the positions of a vector that match, 'grep' shows us the value. So we want to use grepl.

The 1st arguement is what character string we're searching for ('Merc' in our case), and the 2nd argument is **where** we want R to search for that phrase (the row.names or name column in our case). This **is** cap sensitive.

```
grepl('Merc', row.names(mtcars))
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE
## [12] TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
mtcars[grepl('Merc', row.names(mtcars)),]
```

```
##           mpg cyl  disp  hp drat   wt  qsec vs am gear carb      name
## Merc 240D  24.4   4 146.7  62 3.69 3.19 20.0  1  0    4    2  Merc 240D
## Merc 230   22.8   4 140.8  95 3.92 3.15 22.9  1  0    4    2  Merc 230
```

```
## Merc 280      19.2    6 167.6 123 3.92 3.44 18.3 1 0 4 4 Merc 280
## Merc 280C     17.8    6 167.6 123 3.92 3.44 18.9 1 0 4 4 Merc 280C
## Merc 450SE    16.4    8 275.8 180 3.07 4.07 17.4 0 0 3 3 Merc 450SE
## Merc 450SL    17.3    8 275.8 180 3.07 3.73 17.6 0 0 3 3 Merc 450SL
## Merc 450SLC   15.2    8 275.8 180 3.07 3.78 18.0 0 0 3 3 Merc 450SLC
```

Now we can just add this to our previous subset

```
final <- subset(mtcars, wt<4 & (cyl==4|cyl==6) & mpg >=20 & grepl('Merc', row.names(mtcars)) )
final[which.max(final$hp),]
```

```
##           mpg cyl  disp hp drat   wt  qsec vs am gear carb    name
## Merc 230 22.8   4 140.8 95 3.92 3.15 22.9  1  0   4    2 Merc 230
```

And our answer is the Mercedes 230! Price is **clearly** not part of my criteria

Notice we have used all logic statements that give us T/ F, if we've statements that gave us **positions** we couldn't mix them with T/F logical statments



Hard

Part 1. Say we have a 10 x 10 gridded wetland. In each cell we've measured two habitat characteristics water height and percent cover of grasses. We want to characterize each cell as a pond, mudflat, wetland, wet field, or upland based on our two measurements and put this into **one** categorical column called 'HabitatType'.

- pond = water height > 90mm and percent cover of grass <= 0.5
- mudflat = water height <= 90mm and percent cover of grass <= 0.5
- wetland = water height > 50mm and percent cover of grass > 0.5
- wetfield = water height <= 50mm and percent cover of grass > 0.5
- upland = water height <= .1 and percent cover of grass > 0.50
- bare = water height <=.1 and percent cover of grass <0.5

Heres your data set:

```
data <- data.frame(
  lat=rep(seq(30,30.9,.1),each=10),
  long=rep(seq(100,100.9,.1), by=10),
  water = sample(seq(1,120,.1), 100, replace=T),
  PerCover = sample(seq(.3,1,.1), 100, replace=T))

head(data)
```

```
##   lat  long water PerCover
## 1  30 100.0  31.1      0.6
## 2  30 100.1   9.9      1.0
## 3  30 100.2  21.3      0.3
## 4  30 100.3  89.2      1.0
## 5  30 100.4  36.7      0.7
## 6  30 100.5  62.2      0.5
```

Solution

This can be done in only a few steps, or a longer series of convoluted middle steps

```
data$HabitatType <- NA

data$HabitatType[data$water>90 & data$PerCover<=0.50] <- "pond"
data$HabitatType[data$water<=90 & data$PerCover<=0.50] <- "mudflat"
data$HabitatType[data$water>50 & data$PerCover>0.50] <- "wetland"
data$HabitatType[data$water<=50 & data$PerCover>0.50] <- "wetfield"
data$HabitatType[data$water<=10 & data$PerCover>0.50] <- "upland"
data$HabitatType[data$water<=10 & data$PerCover<=0.50] <- "bare"
head(data)
```

```
##   lat  long water PerCover HabitatType
## 1  30 100.0  31.1      0.6    wetfield
## 2  30 100.1   9.9      1.0      upland
## 3  30 100.2  21.3      0.3    mudflat
## 4  30 100.3  89.2      1.0    wetland
## 5  30 100.4  36.7      0.7    wetfield
## 6  30 100.5  62.2      0.5    mudflat
```

Here the important thing to note is if you create the column before hand and fill it with NAs then we can use subsetting to tell it where to put in the correct values. This doesn't work if this column has never been created.

Bonus 1. (Very Hard) During my field work I noticed that wetlands tend to occur next to other wetlands. I want to figure out how many of my wetland cells have atleast 1 neighbor that is also a wetland (This is a loose measure of contagion in landscape ecology).

Count neighbors as only their north/south/east/west neighbor (4 neighbor rule). Assume anything outside our study area is a 0.

Hint - In this instance you'll have to make a 10x10 matrix with latitude as rows, columns as longitude and wetland is coded as 1 or 0.

Solution This solution just represents what I came up with, I think you can do this many different ways and I'd love to see other ways people did it. Let first make our matrix. Now I've decided to go the easy route here, and assume that I know how R is going to fill the matrix in. This is because to R a matrix is just one long vector, but with defined dimensions. So if we tell it the matrix should be 10 x 10 then it will lay the first 10 entries down in the first column, 2nd 10 entries in the 2nd column and so on. So we sort first and then just give the column to our matrix.

You'll notice we can give it the dimension names using the 'dimnames' argument and then supply it a list where the first entry is the row names and second is the column. This is useful, but for the solution I chose

later we have to leave this out (you'll see why). So I show it to you now, but then overwrite it without dim names.

```
data$wetland<-0
data$wetland[data$HabitatType=='wetland']<-1
data2<-data[,c('lat','long','wetland')]
data2<-data2[order(-data2$lat, data2$long),]

##this is typically what we'd do
spatial<-matrix(data2$wetland, 10, 10, dimnames=list(unique(data2$lat),unique(data2$long)))
spatial
```

```
##      100 100.1 100.2 100.3 100.4 100.5 100.6 100.7 100.8 100.9
## 30.9   0    1    1    0    0    1    0    0    0    0
## 30.8   1    1    1    0    1    0    0    0    1    0
## 30.7   0    0    0    0    0    1    1    0    0    0
## 30.6   1    0    1    0    0    0    1    1    0    1
## 30.5   1    0    1    1    1    0    0    0    0    0
## 30.4   1    1    0    0    0    0    0    0    0    0
## 30.3   0    0    0    1    0    1    0    1    1    1
## 30.2   1    1    0    0    1    0    0    1    0    0
## 30.1   1    0    0    0    0    1    1    1    1    0
## 30     1    0    0    0    1    1    1    1    1    0
```

```
##but for the next analysis we're going to do this, just trust me the rows and columns are correct
spatial<-matrix(data2$wetland, 10, 10)
```

This isn't necessarily the only way to make this into a 10X10 matrix, but it sure as heck is the easiest.

Now we need to calculate for each point whether it's 4 neighbors are also wetlands. Many of you were probably thinking of using a for loop, and I do think you can in this situation. That loop won't scale particularly well, and you still will have to create very specific control structures for when its the 1st row, 1st column, last row, or last column. The solution I came up with is completely vectorized. Basically if you shift the matrix one column over, then our east neighbor is now where our original points were. Say we only wanted to figure out if the east neighbor was a wetland, we could artificially shift it one column over, and ask R if that value is 1 or 0.

Compare these two matrices

```
east<-cbind(spatial[,2:10], rep(0,10))
spatial
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    1    0    0    1    0    0    0    0
## [2,]    1    1    1    0    1    0    0    0    1    0
## [3,]    0    0    0    0    0    1    1    0    0    0
## [4,]    1    0    1    0    0    0    1    1    0    1
## [5,]    1    0    1    1    1    0    0    0    0    0
## [6,]    1    1    0    0    0    0    0    0    0    0
## [7,]    0    0    0    1    0    1    0    1    1    1
## [8,]    1    1    0    0    1    0    0    1    0    0
## [9,]    1    0    0    0    0    1    1    1    1    0
## [10,]   1    0    0    0    1    1    1    1    1    0
```

```
east
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    1    0    0    1    0    0    0    0    0
## [2,]    1    1    0    1    0    0    0    1    0    0
## [3,]    0    0    0    0    1    1    0    0    0    0
## [4,]    0    1    0    0    0    1    1    0    1    0
## [5,]    0    1    1    1    0    0    0    0    0    0
## [6,]    1    0    0    0    0    0    0    0    0    0
## [7,]    0    0    1    0    1    0    1    1    1    0
## [8,]    1    0    0    1    0    0    1    0    0    0
## [9,]    0    0    0    0    1    1    1    1    0    0
## [10,]   0    0    0    1    1    1    1    1    0    0
```

The east matrix is just the spatial dataframe shifted 1 column over (with 0's replacing the 10th column). If we ask R whether our dataframes neighbor to the right has a 1 in it. Does this not give us the answer?

```
east==1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] TRUE  TRUE FALSE FALSE TRUE  FALSE FALSE FALSE FALSE FALSE
## [2,] TRUE  TRUE FALSE TRUE  FALSE FALSE FALSE TRUE  FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE TRUE  TRUE  FALSE FALSE FALSE FALSE
## [4,] FALSE TRUE  FALSE FALSE FALSE TRUE  TRUE  FALSE TRUE  FALSE
## [5,] FALSE TRUE  TRUE  TRUE  FALSE FALSE FALSE FALSE FALSE FALSE
## [6,] TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [7,] FALSE FALSE TRUE  FALSE TRUE  FALSE TRUE  TRUE  TRUE  FALSE
## [8,] TRUE  FALSE FALSE TRUE  FALSE FALSE TRUE  FALSE FALSE FALSE
## [9,] FALSE FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE  FALSE FALSE
## [10,] FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE  TRUE  FALSE FALSE
```

```
spatial
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    1    0    0    1    0    0    0    0
## [2,]    1    1    1    0    1    0    0    0    1    0
## [3,]    0    0    0    0    0    1    1    0    0    0
## [4,]    1    0    1    0    0    0    1    1    0    1
## [5,]    1    0    1    1    1    0    0    0    0    0
## [6,]    1    1    0    0    0    0    0    0    0    0
## [7,]    0    0    0    1    0    1    0    1    1    1
## [8,]    1    1    0    0    1    0    0    1    0    0
## [9,]    1    0    0    0    0    1    1    1    1    0
## [10,]   1    0    0    0    1    1    1    1    1    0
```

If you accept that, then you can do that for all 4 neighbors, add them up, and then see how many neighbors does each point have that has a 1

```
east<-cbind(spatial[,2:10], rep(0,10))
west<-cbind(rep(0,10),spatial[,1:9])
north<-rbind(rep(0,10), spatial[1:9,])
south<-rbind(spatial[2:10,], rep(0,10))
add<-east+west+north+south
add
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    2    2    2    1    2    0    1    0    1    0
## [2,]    1    3    2    2    0    3    1    1    0    1
## [3,]    2    1    2    0    2    1    2    2    1    1
## [4,]    1    2    1    2    1    2    2    1    2    0
## [5,]    2    3    2    2    1    1    1    1    0    1
## [6,]    2    1    2    2    1    1    0    1    1    1
## [7,]    2    2    1    0    3    0    2    2    2    1
## [8,]    2    1    1    2    0    3    2    2    3    1
## [9,]    2    2    0    0    3    2    3    4    2    1
## [10,]   1    1    0    1    1    3    3    3    2    1
```

```
spatial
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    1    0    0    1    0    0    0    0
## [2,]    1    1    1    0    1    0    0    0    1    0
## [3,]    0    0    0    0    0    1    1    0    0    0
## [4,]    1    0    1    0    0    0    1    1    0    1
## [5,]    1    0    1    1    1    0    0    0    0    0
## [6,]    1    1    0    0    0    0    0    0    0    0
## [7,]    0    0    0    1    0    1    0    1    1    1
## [8,]    1    1    0    0    1    0    0    1    0    0
## [9,]    1    0    0    0    0    1    1    1    1    0
## [10,]   1    0    0    0    1    1    1    1    1    0
```

And we see it does work. So now we just have to do the trivial matter of counting up how many of these cells are wetlands and have neighbors that are wetlands. Which is only trivial if you've gotten this far.

```
sum((add>0 & spatial>0))/(sum(spatial>0))
```

```
## [1] 0.8292683
```

If you remember we decided not to give our matrix dimension names, the reason is, when you add up all of the shifted dataframe R attempts to add the like named columns and rows, and so the result is an 11x11 matrix with an unnamed row and unnamed column. I currently don't know how to fix this, but I assume it's an easy fix.

Bonus 2. (INSANITY) THE COORDINATES ARE POLAR COORDINATES!

```
data1 <- data.frame(
  range = rep(seq(1,10,1), each=10),
  azimuth = rep(seq(0,350, 36), by=10),
  water = sample(seq(1,120,.1), 100, replace=T),
  PerCover = sample(seq(.3,1,.1), 100, replace=T))

head(data1)
```

```
##   range azimuth water PerCover
## 1     1       0  30.2      0.7
## 2     1      36  82.8      1.0
```



```
## 3      1      72 118.4      0.4
## 4      1     108  97.1      0.4
## 5      1     144  69.5      0.9
## 6      1     180  94.1      0.9
```

Solution The only reason this one is rated **insanity** but not the previous question, is because I expected polar coordinates to freak people out. If you've figured out the last problem, however, then this problem shouldn't be harder. The only thing you have to remember is that 1st column and 10th column are actually **next to each other** because it's a grided circle. I'm really just going to do the same analysis as before but instead of giving it a phantom set of 0's, I can fill it in with values from the opposite column. **notice, the rows don't change since this isn't some sort of crazy quad polar coordinate system or something.**

This first step isn't important, if you wanted you could just copy 'range' and 'azimuth' into the spot of lat and long for quickness.

```
data1$HabitatType <- NA
data1$HabitatType[data1$water>90 & data1$PerCover<=0.50] <- "pond"
data1$HabitatType[data1$water<=90 & data1$PerCover<=0.50] <- "mudflat"
data1$HabitatType[data1$water>50 & data1$PerCover>0.50] <- "wetland"
data1$HabitatType[data1$water<=50 & data1$PerCover>0.50] <- "wetfield"
data1$HabitatType[data1$water<=10 & data1$PerCover>0.50] <- "upland"
data1$HabitatType[data1$water<=10 & data1$PerCover<=0.50] <- "bare"

data1$wetland<-0
data1$wetland[data1$HabitatType=='wetland']<-1
data2<-data1[,c('range','azimuth','wetland')]
data2<-data2[order(data2$azimuth, data2$range),]

spatial<-matrix(data2$wetland, 10, 10)

##here is where we add in the opposite column as opposed to zeros.
east<-cbind(spatial[,2:10], spatial[,1])
west<-cbind(spatial[,10],spatial[,1:9])
north<-rbind(rep(0,10), spatial[1:9,])
south<-rbind(spatial[2:10,], rep(0,10))
add<-east+west+north+south

sum((add>0 & spatial>0))/(sum(spatial>0))
```

```
## [1] 0.8780488
```

I don't know why you'd actually set this experiment up in a polar grid, but you do find polar grids in remoting sensing and meteorology, so its good to understand.