Rclub Solutions for exercise 1

Matt Boone

October 8, 2015

First off I want to remind everyone that there is more than one way to get at a problem, this is just one way to answer the question. I look forward to y'all posting your own answers!

Easy

Say you have a data set where you measured percent canopy cover at certain points. However, at a few points you could not get a value because they were actually in the middle of a pond.

```
data1<-data.frame(id=1:10, perCover=c(0.1,0.5,NA, 0.4,0.3,NA,1,0.9,0.7,0.7))
data1
```

```
##    id perCover
## 1   1      0.1
## 2   2      0.5
## 3   3       NA
## 4   4      0.4
## 5   5      0.3
## 6   6       NA
## 7   7      1.0
## 8   8      0.9
## 9   9      0.7
## 10 10      0.7
```

How would you calculate the average canopy cover?

Solution

This is easiest done by just using the mean function, with one caveat, if we ran this on the current data set it would fail:

```
mean(data1$perCover)
```

```
## [1] NA
```

That's why we need to add the agrument na.rm=T, so it will remove NAs before calculating.

```
mean(data1$perCover, na.rm=T)
```

```
## [1] 0.575
```

You can see other arguements for functions by looking at the help page in Rstudio and searching the function. Or typing:

?mean

Medium

Say you have acquired a long term data set from a banding site. Unsurprisingly, your data is riddled with NA's as there was no standard protocol across years.

```
data2<-matrix(1:50,10,5)
data2[sample.int(50,7)]<-NA
colnames(data2) <- c('id','tarsus','culmen','wingchord','tail')
data2
```

```
##       id tarsus culmen wingchord tail
##  [1,]  1     NA     NA        31   NA
##  [2,]  2     12     22        32   42
##  [3,]  3     13     23        NA   43
##  [4,]  4     14     24        34   44
##  [5,]  5     15     25        35   45
##  [6,]  6     16     26        36   NA
##  [7,]  7     17     27        37   47
##  [8,]  8     18     28        38   48
##  [9,]  9     19     29        39   NA
## [10,] 10     NA     30        40   50
```

You want to take the mean of each column. How do you do this? How would you do this without taking the mean of the ID column?

Solution:

To answer this question we're going to use the apply function since we want to take the mean of multiple columns. An apply allows us to apply a function either across the rows or down each column seperately. We have to tell it whether we want to run it down the runs or columns by entering either 1 or 2 in the second position. 1 means we want it to run **across** the rows, and 2 means we want it to run **down** the columns. So for this problem we'd type:

apply(data2, 2, mean, na.rm=T)

Where 2 tells it we want to run the function 'mean' down the columns of our data set data2. We then give it any extra arguments the function may need **after** the first 3 arguments we supplied.

```
apply(data2, 2, mean, na.rm=T)
```

```
##        id    tarsus    culmen wingchord      tail
##   5.50000  15.50000  26.00000  35.77778  45.57143
```

To only take the mean of the columns **not** called 'id' we have to tell it which columns aren't called 'id'

```
colnames(data2)!='id'
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE
```

And then feed this back to our dataframes column. Notice we are adding this statement **after** the comma, because to the left of the comma selects rows, and to the right selects columns. We now can do the same apply statement as before

```
apply(data2[ ,colnames(data2)!='id'], 2 , mean, na.rm=T)
```

```
##    tarsus    culmen wingchord      tail
##  15.50000  26.00000  35.77778  45.57143
```

Not we could just type data2[,-1] to take out the first column if we **knew** that the first column is always the id column. In this case the '-' sign followed by a number will show us all columns **but** that column

---

Hard

Say you have a data set where you measured the temperature at different minutes. You coded time as a decimal hour (in 24 hour format). Heres your data

```
data3<-data.frame(temp=c(32,32.1,32.4,32.4,32.5,32.8,33,32.9),
                  dhour=c(22.5,22.75,22.86,22.95,22.98,23.05,23.10,23.18))

data3
```

```
##   temp dhour
## 1 32.0 22.50
## 2 32.1 22.75
## 3 32.4 22.86
## 4 32.4 22.95
## 5 32.5 22.98
## 6 32.8 23.05
## 7 33.0 23.10
## 8 32.9 23.18
```

You want to know approximately what the temperature was at 11pm (23.0 decimal hours) by picking the temperature at the closest time to 23.0.

Extra: How can you interpolate to this time point? You can use whatever interpolation method you want.

Solution:

I expect people have come up with different answers for this problem. My solution is

```
data3$temp[ which.min( abs( data3$dhour - 23) ) ]
```

```
## [1] 32.5
```

When I create a solution for a problem I code **inside -> out** as that's how R is going to evaluate the expression.

So first I think of the simplest form of this problem. We need to know what number is closest to 23 and choose that number. So simple subtraction should work.

```
data3$dhour-23
```

```
## [1] -0.50 -0.25 -0.14 -0.05 -0.02  0.05  0.10  0.18
```

However we notice that subtraction could give us a positive or negative value, we just want the closest value. So we need to take the absolute value of these numbers

```
abs(data3$dhour-23)
```

```
## [1] 0.50 0.25 0.14 0.05 0.02 0.05 0.10 0.18
```

Now we just need to choose which is the smallest value. This can be done in many ways. Some people might want to sort the numbers and choose the first value, though then we'd have to make sure the position of the vector is followed throughout the process. I choose to use min, specifically a which.min which is going to tell me which position in our vector has the smallest value:

```
which.min(abs(data3$dhour-23))
```

```
## [1] 5
```

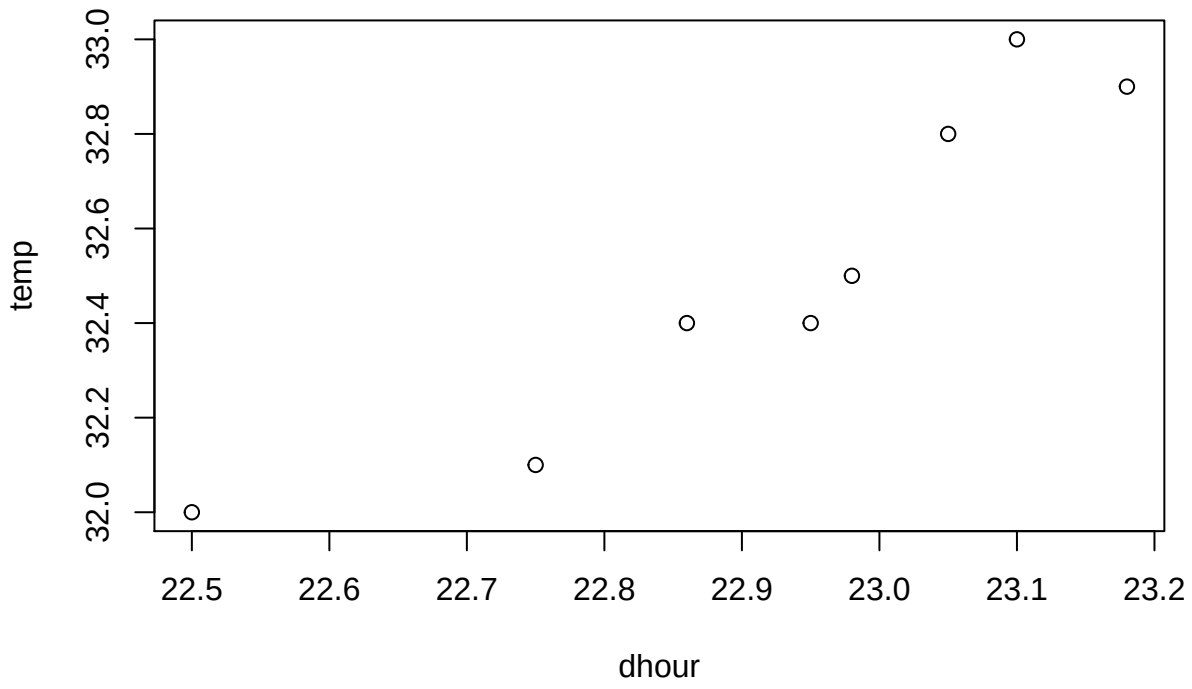
Now that we have the position, we can feed it back to our temperature column to get the answer:

```
data3$temp[which.min(abs(data3$dhour-23))]
```

```
## [1] 32.5
```

For the bonus section, I'm going to describe less of how I figured out the probem, as it was more of an exploratory problem for everyone to work on.

I decided to use predict, as I had never explored the prediction side of R. In this case, we can feed it a model (any model) and then have it use that model to predict a point. Since our data is sort of linear.



The syntax of predict is a bit strange though. We first feed it our model, which can be either an object we stored our model as, or just the model itself. But then we have to feed it new data under the argument 'newdata' this data needs to be a dataframe with the column name of our x variable. In this solution we only have 1 point so it looks funny

```
model1<-lm(temp ~ dhour, data=data3)
predict(model1, newdata=data.frame(dhour=23))
```

```
##        1
## 32.63502
```

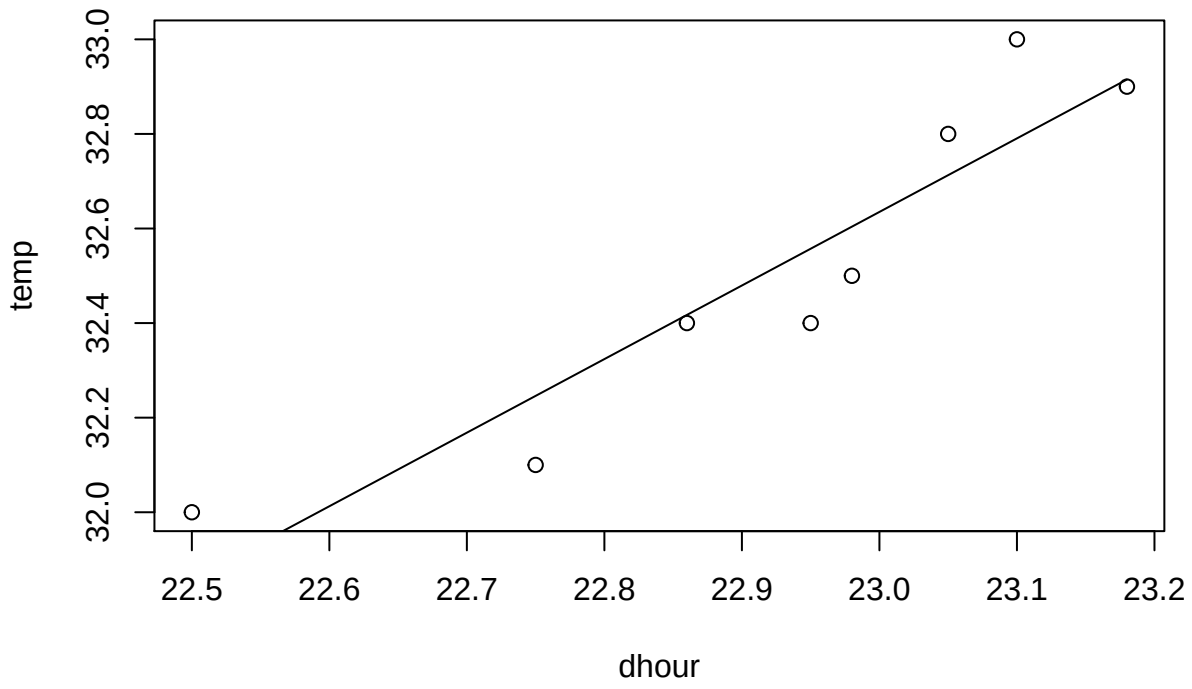
But we can do this with an entirely new data set

```
predict(model1, newdata=data.frame(dhour=21:24))
```

```
##        1        2        3        4
## 29.52334 31.07918 32.63502 34.19086
```

You should note this is also how we add the predicted line to a map (in base plot)

```
plot(temp ~ dhour, data=data3)
lines(x=data3$dhour, predict(model1), type='l')
```



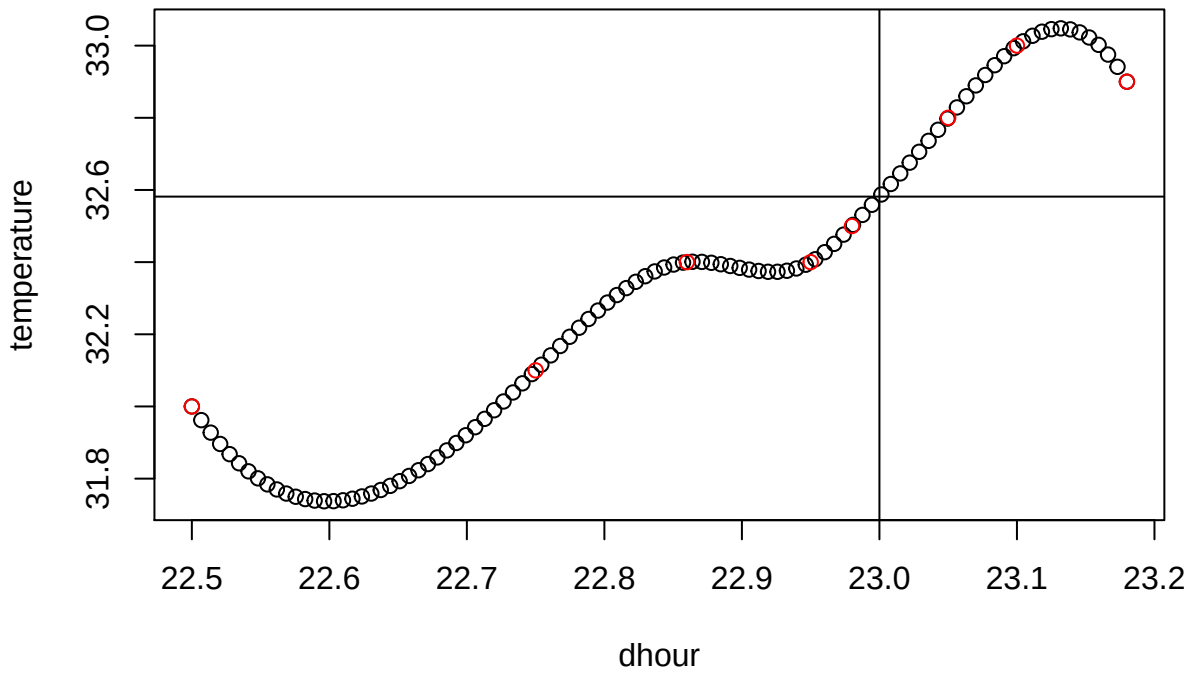
Though this might go to show you how a linear fit might not have been the best. Another idea I thought of is using a spline interpolation. In this case the argument 'xout' will tell it what point to interpolate to.

```
predic_sp<-spline(data3$temp ~ data3$dhour,xout=23)
predic_sp
```

```
## $x
## [1] 23
##
## $y
## [1] 32.5816
```

```
spline1<-spline(data3$temp ~ data3$dhour, n=100)

plot(spline1, xlab='dhour', ylab='temperature')
points(temp ~ dhour, data=data3, col='red')
abline(v=predic_sp$x, h=predic_sp$y)
```



In this graph, the red dots are our real values, black are interpolated values, and the cross is the value of our interpolation at 23.

I just like splines, they're fun.