Really getting to know R: practical 2

Kevin Keenan

4 May 2015

Introduction

In this practical we will go over some of the topics shown in the lecture, as well as picking up some new tricks along the way. The purpose of this practical is to get you comfortable working in the R environment, rather than providing you with an encyclopaedic knowledge of the language. It is expected that you will search online, ask questions and use the built-in help system when figuring out the answer to each question.

Before you begin

We will record all of our answers in an R script. This will allow us to check back on previous answers that might help us answer later questions etc. Using RStudio, open a new R script and save it as "practical_2_script.R"

Some basic skills

In this section we will get used to the concepts of **data structures**, **indexing**, and **functions**.

Interacting with R

1. **Numeric Vectors**

   a. Create a vector of length 10 named `x` with the following values:

   ```
   1 2 3 4 5 6 7 8 9 10
   ```

   Hint: The functions '`seq`', '`c`' and '`:`' can help you here.

   b. Can you extract the 5^{th} element from `x`?

   c. Add the 5^{th} and the 7^{th} elements of `x` together.

   d. Create another vector, `y`, with the value x^2

   e. Can you plot y x?

2. **Character vectors**

   a. Create a character vector, `my_name`, where the first element is your title (e.g. Dr, Miss, Mr etc.), the second element is your first name, and the third element is you second name.

   b. Using the function `nchar` count how many characters are in each element of `my_name`.

   c. Using the function `paste` can you figure out how you could combine all three elements of `my_name` into a single string?

3. **Matrices**
   Remember that a matrix is simply a vector with two dimensions (rows and columns).

   a. Using the function `matrix`, can you create a matrix named `mat` from the vectors `x` and `y`? Your matrix should have two columns.

b. Using logical testing (i.e. `==`), can you figure out how you could test if one column is actually the square of the other?

```
Hint: mat[,1] == mat[,2] gives:
TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

c. Can you calculate the variance (`?var`) of both columns in your matrix

d. The function `colMeans` can be used to return the mean of each column in a matrix. Can you think of an alternative way to calculate the same value, but using the `apply` and the `mean` functions instead?

4. **Lists**
Indexing elements of a list can be carried out a number of ways. If your list elements are named then the `$` operator can be used to access them:

```
list_name$element_name
```

Alternatively, you can use double brackets combine with standard numerical indexing:

```
list_name[[1]]
list_name[[2]]
```

a. Create a list named `all_data`, where the first element is `x`, the second is `y` and the third is `mat`.

```
all_data <- list(x = x, ..., etc.)
```

b. What are the three way that you can return `mat` from `all_data`? (Hint: see the dataframes section of the lecture).

c. Can you write **logical test** to prove that the second column in `mat` is equal to `y`, using only `all_data`?

5. **Dataframes** Dataframes are simply lists where all of the elements are the same length and are stored in two-dimensions (rows and columns).
The code below will create a small dataframe:

```
my_df <- data.frame(x = x, y = y)
```

x	y
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

a. Using the same indexing skills you have learned for lists, write a **logical test** to show that column `y` in `my_df` is equal to the square of column `y`.

b. Plot `y` against `x` in a simple scatterplot. (See `?plot` for details).

A more realistic experience

So that is the basics out of the way. We should all be comfortable with typing commands in R and getting results (and sometimes errors!). The next challenge is slightly more realistic in that we will begin to analyse 'real' data that we first have to read into the R environment (the workspace).

When analysing data in R, it is common that it is stored in files as data tables (e.g. excel style, csv or tab delimited).

Below is a typical data set:

id	site	haplotype	sex	length
1	site C	HapA	F	117.88
2	site D	HapA	F	150.63
3	site C	HapA	F	120.07
4	site D	HapA	F	146.96
5	site C	HapA	F	120.15
6	site D	HapA	M	99.98
...	...	...	...	...

These data were recorded for brown trout (*Salmo trutta*) sampled from four distinct locations (sites A, B, C and D), putatively representing four separate populations. A mitochondrial assay revealed a total of six unique haplotypes distributed at different frequencies among each of the separate population samples. Fish length (mm) and gender was also recorded. Throughout this practical we will use these data to learn some new skills in R. We will also explore the flexibility of R by demonstrating the simplicity with which genetic parameters can be calculated.

6. **Reading data into R**
   There are many functions for reading data into R. In general, if you have saved your data in tabular format (as above) then it will either be in a tab delimited or csv file. Tab delimited files can typically be read using the functions `read.table` or `read.delim`, while csv files can be read using `read.csv`.
   a. Using the function `read.delim`, can you read the file *trout_genetics.txt* into R and assign it to the value `trout_data`?
   b. Use the function `str` to inspect the structure of the data. Do you notice any obvious problems? (Are all variables in the format expected?)
   c. If you have noticed a problem, what do you think is the cause?
   d. There is an argument `na.strings` that might help you fix this problem. Try re-reading the file using an appropriate argument for `na.strings`.
   e. Running the code below, make sure that your results match:

```
sapply(trout_data, class)
```

```
      id      site haplotype       sex    length
"integer"  "factor"  "factor"  "factor" "numeric"
```

   These results show that the first column (id) is an `integer vector`, site, haplotype and sex are all `factors`, and length is a `numeric vector`.
7. **Exploring your data with code**
   Now that we have managed to get our data into R, we can begin to try and understand it a little better.
   a. How many trout have been sampled for this study?
   b. Using the function `levels`, can you check how many unique sites have been sampled from?

c. By combining the functions `tapply` and `length`, count how many individual were collected from each site. Save the results in the variable `pop_sizes`.

d. A more sensible way we could do this is using the `table` function. Check that both method give the same results.

e. Let's calculate the mean trout length per site. The function `tapply` will be very helpful here. Also, remember the missing data point? That will cause a problem. Can you figure out how to fix it?

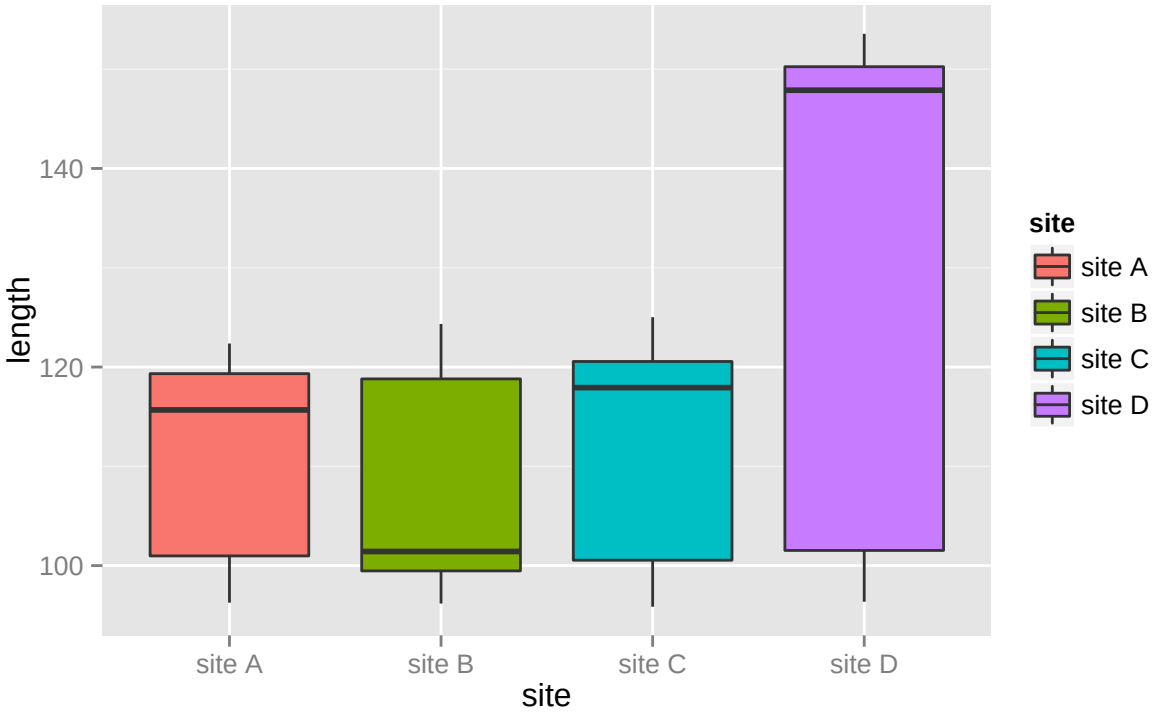
f. Which site has the largest fish?

8. **Exploring you data with plots**

Calculating descriptive statistics from your data is very useful. However, visualising your data can be even more useful. In this section we will look at how visually exploring data can often reveal details not obvious otherwise.

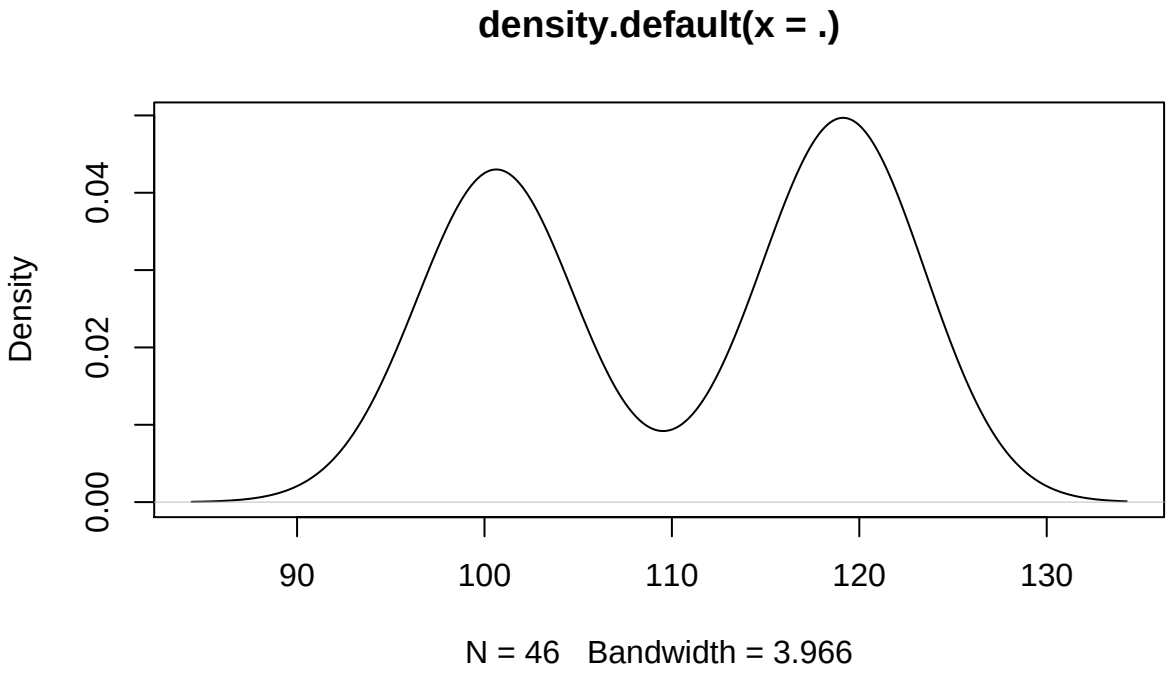
a. Below is an example of how we could plot fish lengths grouped by sampling site.

```
library("ggplot2")
ggplot(trout_data, aes(x = site, y = length, fill = site)) +
  geom_boxplot()
```



Oh dear! There is something strange looking here. The median line for each box is skewed suggesting these data are not normal. We can plot the length data for one site and see what it looks like:

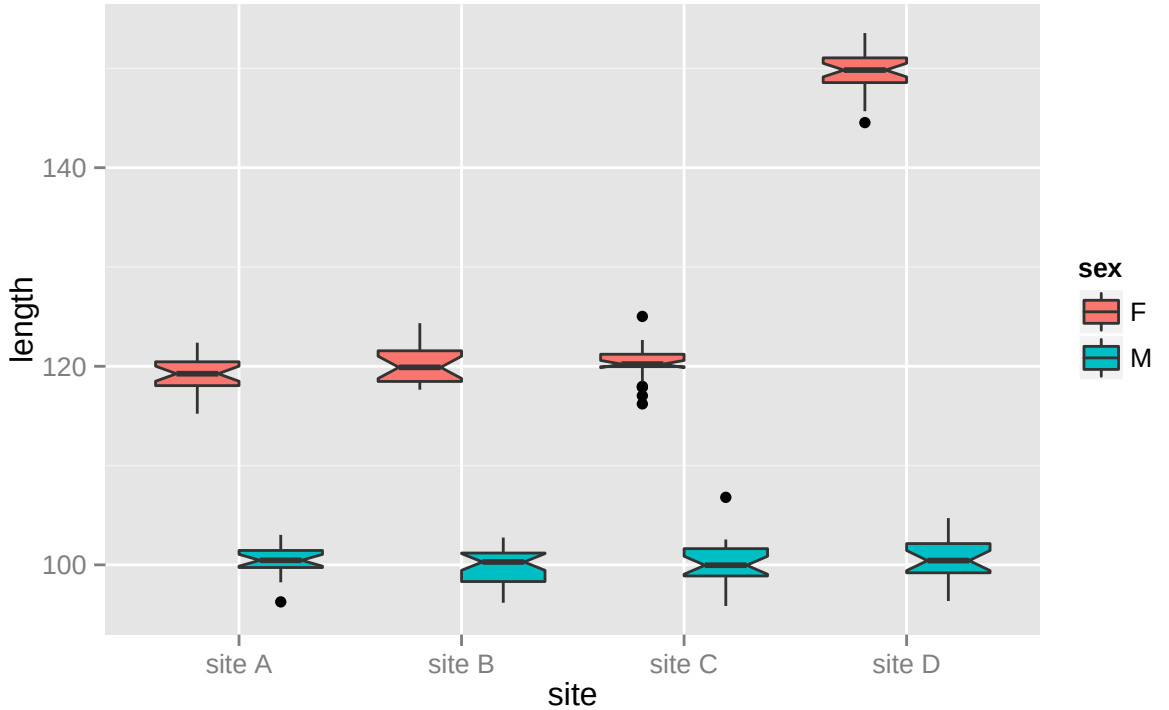
```
library("dplyr")
trout_data %>%
  filter(site == "site A") %>%
  select(length) %>%
  unlist() %>%
  density() %>%
  plot()
```



It is clear that we have bio-modal data here.

b. Looking at your data can you think of a variable that might explain this pattern?
c. Below show how we could visually test our hypothesis:

```
library("ggplot2")
ggplot(trout_data, aes(x = site, y = length, fill = sex)) +
  geom_boxplot(notch = TRUE)
```



It is obvious that there is a difference in length between the sexes. In particular, females in *site D* are very large. We could obviously used the information we have learned about our data to inform further steps in the analysis process (e.g. anova of length etc.)

9. **Can we calculate hyplotype frequencies?**
A fun challenge using these data would be to calculate the hyplotype frequencies and possibly visualise them some how.

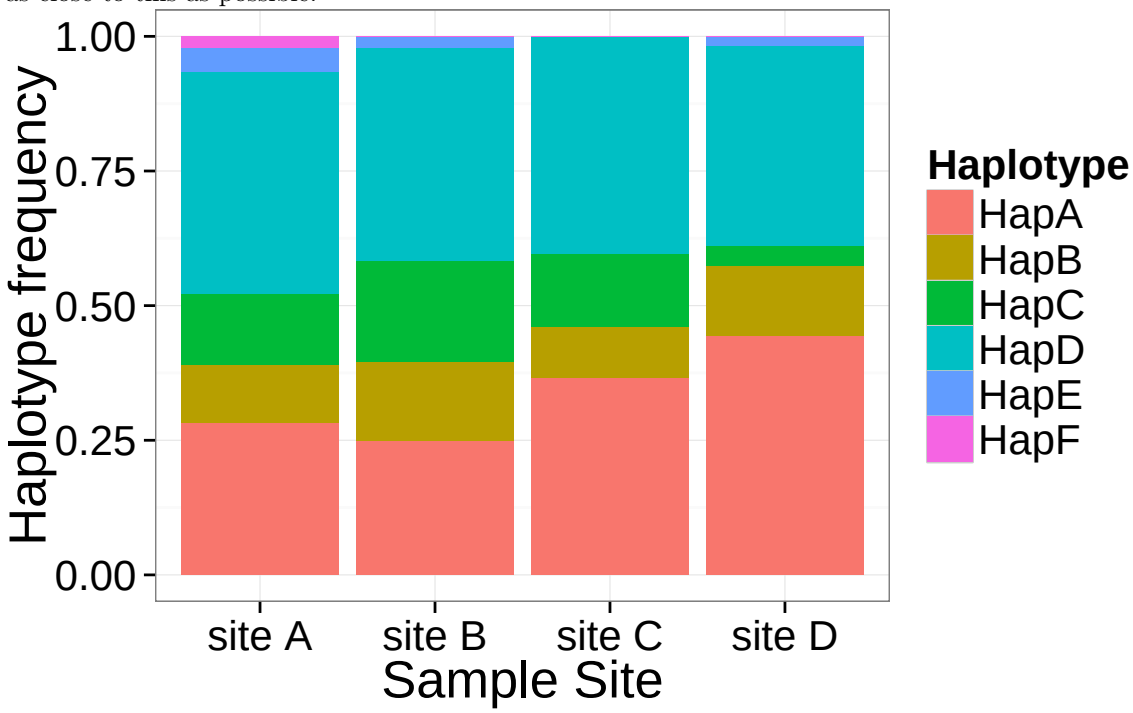
   a. Do you think the `table` function and `tapply` combined with your sample sizes from earlier could be used to calculate haplotype frequencies? Try it simply at first, maybe looking at one site, then trying to expand to all sites. Your results should look something like this:

	site A	site B	site C	site D
HapA	0.2826	0.25	0.3654	0.4444
HapB	0.1087	0.1458	0.09615	0.1296
HapC	0.1304	0.1875	0.1346	0.03704
HapD	0.413	0.3958	0.4038	0.3704
HapE	0.04348	0.02083	0	0.01852
HapF	0.02174	0	0	0

   b. How can you test that all site frequencies sum to 1.0?

10. **Visualising haplotype frequencies**
Now that we have successfully calculated our haplotype frequencies per site, we might want to visually inspect the different frequencies across sites. One particularly useful method for this is using the **stacked barplot**. This method is documented in the `diveRsity` packages vignette.

   a. Using pages 27-28 of the vignette, can you figure out how to plot hap_freq? Your plot should look as close to this as possible:



THE END