Introduction to R: practical 1

Dr Colin S. Gillespie

April 28, 2014

This practical aims at introducing you to the R interface. By the end of this practical you should be able to load in data, calculate some summary statistics and construct some basic plots.

1 *Getting started*

1. Open `Rstudio`
2. In `Rstudio`, type a basic R command, say

```
x = 5
```

3. Press `Ctrl + Enter` anywhere on the line with `x = 5`. This should send the command `x = 5` to the R console in the bottom left hand window.
4. In `Rstudio`, save the file you are currently working on. Rstudio will (correctly) add the file extension `.R`

Other `Rstudio` commands are:

In Rstudio, click on `Help` and `Keyboard shortcuts` to see other shortcuts.

- Pressing `Ctrl + Enter` will send the current line to the R console.
- If you highlight a few lines of R code, pressing `Ctrl + Enter` will send that code to the R console.
- Pressing `Ctrl + Shift + R` sends the entire file to R console.

It's probably worth creating a directory to store any R files that you create.

2 *Course R package*

This practical uses the course R package. Installing this package is straightforward:[1]

[1] r-forge, CRAN and Bioconductor are package repositories, i.e. web-sites that host R packages. It is straightforward to set-up and maintain our own package repository.

```
install.packages("nclRcourses",
                 repos="http://R-Forge.R-project.org")
```

To load the package, use

```
library(nclRcourses)
```

3 *The data*

We are going to investigate the yeast data set described in chapter 1 of the notes.

3.1 *Retrieving the data*

To the data is available in the `nclRcourses` package. To load the data into R, use:[2]

```
data(yeast_long)
```

We can inspect the column names using:

```
## yeast_long is a data frame
colnames(yeast_long)

## [1] "ID"    "value" "type"  "rep"   "tps"
```

We easily change the column names, for example,

```
colnames(yeast_long)[1]

## [1] "ID"

colnames(yeast_long)[1] = "id"
```

We can select individual columns, using either their column name:

```
yeast_long$value
```

or their column number:

```
yeast_long[, 2]
```

When vectors or data frames are too large to manage, we use the function `head` to take a peek at the data:

```
head(yeast_long$value, 5)

## [1] 8.861 8.723 8.836 8.944 9.068
```

- In the above function call, what does `5` specify? What happens if you omit it?
- Another useful function is `tail`. What does this function do?

Using the `dim` function, how many columns and rows does `d` have?

4 *Scatter plots*

Let's start with some simple scatter plots. However, before we begin we'll set the variable `d` to be our data frame[3]

[2] You access the help page on this data set using the command `?yeast_long`

[3] We do this because we are too lazy to type `yeast_long`.

```
## Select all measurements on the first probe
d = yeast_long[1:15, ]
plot(d$tps, d$value)
```

Now,

- When you call `plot`, R guesses at suitable axis labels. Use the `xlab` and `ylab` arguments to specify better axis labels.

```
plot(d$tps, d$value, xlab = "A nice label")
```

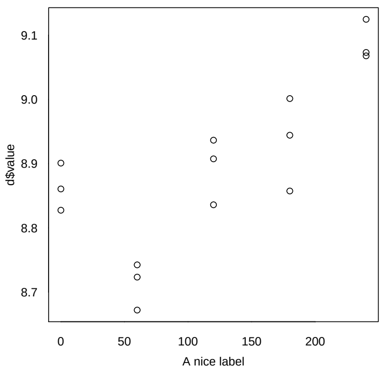

Figure 1: Adding an x-axis label using `xlab`.

- Use the `ylim` argument to specify a y-axis range from 1 to 10.
- Use the `col` argument to change the colour of the points.

```
plot(d$tps, d$value, col = d$rep)
```

- Use the `pch` argument to change the shape of the points.
- Use the `main` argument to give the plot a title.
- I tend to alter the default plot command using:

```
par(mar=c(3,3,2,1), mgp=c(2,.7,0), tck=-.01,
    las=1)
```

What do you think? Can you determine what `mar`, `mgp`, `tck` and `las` mean?

5 Summary Statistics

Use the commands `mean`, `median`, `sd` and range to calculate the summary statistics for the yeast expression levels.

Statistics	Value
Mean	
Median	
Std. dev	
Range	

Table 1: Summary statistics for the yeast data set.

6 Histograms

We will now investigate the distribution of yeast expression levels using histograms. Use the `hist` function to plot a histogram of the yeast expression

```
hist(yeast_long$value)
```

The default method for determining the number of bins used in your histogram isn't that great. So you can use different rules

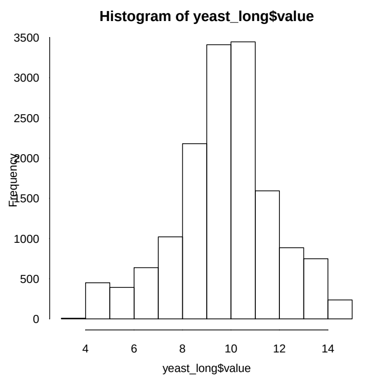

Figure 2: Histogram of all expression values.

```
hist(yeast_long$value, breaks = "FD")
```

The arguments that we encountered when looking at scatter plots can be used with histograms.

Setting `breaks=FD` constructs the histogram using the Freedman-Diaconis rule for calculating binwidths. The other rules are: `Sturge` (default) and `Scott`. See `?hist` for further details.

7 *Box plots*

Let's generate a boxplot for the expression levels:

```
boxplot(yeast_long$value)
```

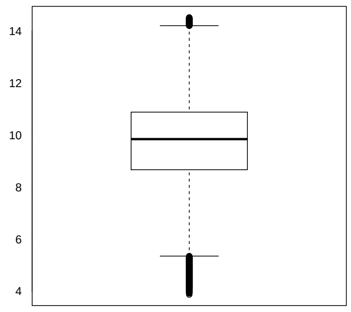

Figure 3: Boxplot of all expression values.

All the usual arguments, such as `xlab`, can be used here. Now lets, separate the yeast data by whether its type

```
boxplot(yeast_long$value ~ yeast_long$type)
```

Try generating a similar boxplot, but for other variables. What happens when you condition on more than one variable?

8 *Spicing up your graph (bonus material)*

We want have time to cover this section in the computer lab, but I want to show you that with a bit of thought, it is possible to generate some very nice graphs in R.[4] However, we need to move away from the defaults. Figure 1 shows the three replicates over time for probe `1769308_at`. It also shows the mean value of the three probes. First we specify the colours we want to use:

[4] Try and figure out what each individual R command does. Try varying the arguments. Use help.

```
cols = c(rgb(85, 130, 169, alpha=40, maxColorValue=255),
         rgb(200, 79, 178, alpha=200, maxColorValue=255))
```

Next we alter the size of the graphic window:

```
par(mar=c(3,3,2,1), mgp=c(2,0.4,0), tck=-.01,
    cex.axis=0.9, las=1, xaxs='i', yaxs='i')
```

Then we plot the three replicates

```
plot(d[1:5,]$tps, d[1:5, ]$value, type="l", frame=FALSE,
     axes=FALSE,
     panel.first=abline(h=seq(8.75, 9.5, 0.25),
                        lwd=3, col="lightgray", lty="dotted"),
     xlab="Time", ylab="Expression level",
     xlim=c(0, 250), ylim=c(8.5, 9.5),
     col=cols[1], lwd=2, cex.lab=0.9)
lines(d[6:10,]$tps, d[6:10, ]$value, col=cols[1], lwd=2)
lines(d[11:15,]$tps, d[11:15, ]$value, col=cols[1], lwd=2)
```

We explicitly add the axis labels and title

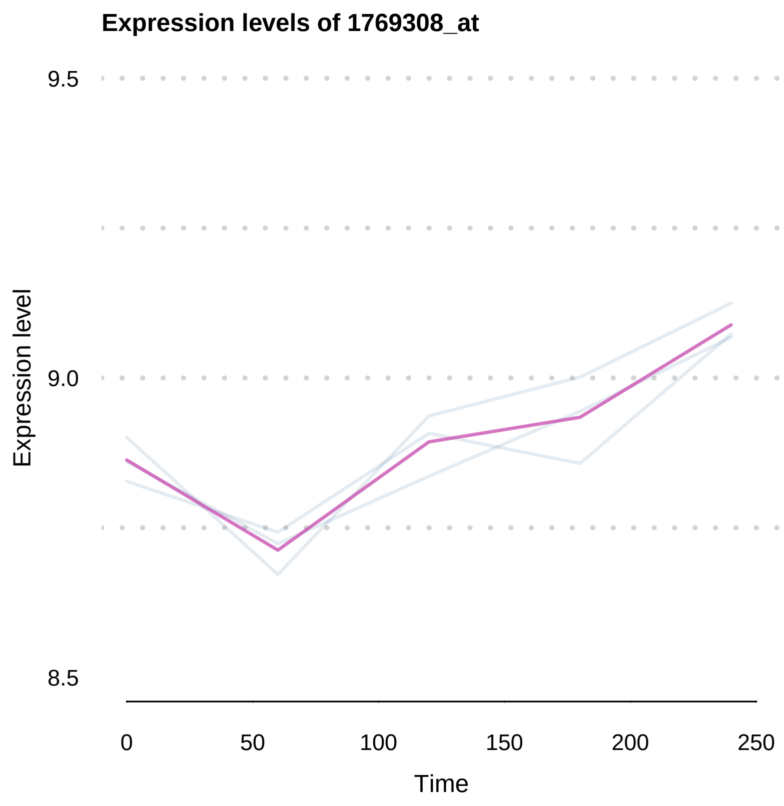

Figure 4: Expression levels of three replications. Mean level is also shown.

```
axis(2, c(8.5, 9, 9.5), c("8.5", "9.0", "9.5"), tick=FALSE,
     cex.axis=0.8)
axis(1, seq(0, 250, 50), seq(0, 250, 50), cex.axis=0.8)
title("Expression levels of 1769308_at", adj=0,
      cex.main=0.9, font.main=2, col.main="black")
```

Finally we add the mean level[5]

[5] We'll cover `tapply` on later.

```
lines(seq(0, 240, 60), tapply(d$value, d$tps, mean),
      col=cols[2], lwd=2)
```

to get figure 4.