
Writing custom scripts & running R batch mode analysis

1

The R scripting language

Scripting

- A script is a series of instructions that when executed sequentially automates a task
 - A script is a good solution to a repetitive problem
 - The art of good script writing is
 - understanding exactly what you want to do
 - expressing the steps as concisely as possible
 - making use of error checking
 - including descriptive comments
- R is a powerful scripting language, and embodies aspects found in most standard programming environments
 - procedural statements
 - loops
 - functions
 - conditional branching
- Scripts may be written in any standard text editor, e.g. notepad, gedit, kate
 - RGui (Mac and Windows) has a built-in text editor

An example script

Scripting

- Colony forming assays provide a measure of cellular proliferation. They are used as read outs for various biological systems
 - A well controlled study may involve multiple samples, treatments and controls (probably replicated).
 - This produces a lot of 'count' data, ideally suited to routine script processing
- Encapsulating the analysis into an R script requires a clear understanding of the problem and data structure

CFA experimental design

Scripting

- Expression of gene X may prevent cells from proliferating in high concentrations of compound Z. The theory is tested by knocking down gene X and growing cells in varying concentrations of compound Z.
 - Three repeat runs (same cell line)
 - Gene X knockdown --> KdX
 - Scramble gene X knockdown control --> ScX
 - Control (transfect empty vector) --> CnX
 - 4 concentrations of compound Z
 - High (HZ), Medium (MZ), Low (LZ), None (OZ)
 - The experiment is replicated over 3 successive weeks
 - Run1 (R1), Run2 (R2) and Run3 (R3)
 - 108 counts in total

Preparing the calculation(s)

Scripting

- We need to make barplots of counts for the KDX-CNX and SCX-CNX for each concentration of Z.
- We will group the repeat runs & replicates, and take an average.
- A Wilcoxon Rank Sum test will tell us whether there is a significant level of protection for KDX in concentrations of Z
- We'll add in some data quality checks
 - Boxplots of repeat runs
 - Variance within replicates

We can copy & paste lines of code into a blank text document, try them out and keep the ones that work!

Importing data

Scripting

	R1									R2									R3								
Plate	KDX			SCX			CNX			KDX			SCX			CNX			KDX			SCX			CNX		
Replicate	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
OZ																											
LZ																											
MZ																											
HZ																											

Fine for humans, bad for R

Run	Plate	Treatment	Replicate	Count
R1	KDX	OZ	1	100
R1	KDX	OZ	2	104
R1	KDX	OZ	3	91
R1	KDX	LZ	1	85
R1	KDX	LZ	2	71
R1	KDX	LZ	3	69
R1	KDX	MZ	1	32
R1	KDX	MZ	2	12
R1	KDX	MZ	3	45
R1	KDX	HZ	1	5
R1	KDX	HZ	2	11
R1	KDX	HZ	3	3
R1	SCX	OZ	1	136
R1	SCX	OZ	2	99
R1	SCX	OZ	3	154
R1	SCX	LZ	1	79
R1	SCX	LZ	2	45
R1	SCX	LZ	3	35
R1	SCX	MZ	1	3
R1	SCX	MZ	2	7
R1	SCX	MZ	3	9
R1	SCX	HZ	1	1
R1	SCX	HZ	2	0
R1	SCX	HZ	3	3
R1	CNX	OZ	1	110
R1	CNX	OZ	2	136
R1	CNX	OZ	3	79
R1	CNX	LZ	1	12
R1	CNX	LZ	2	31
R1	CNX	LZ	3	22
R1	CNX	MZ	1	0
R1	CNX	MZ	2	1
R1	CNX	MZ	3	0
R1	CNX	HZ	1	0
R1	CNX	HZ	2	0
R1	CNX	HZ	3	1

We will create a data frame of factors comprising Run, Plate, Treatment, Replicate
The response variable is counts.

We have 3 spreadsheets of data
Run1counts.csv
Run2counts.csv
Run3counts.csv

We will need to write a procedure that reads in the data
Note that our script will require a consistent data format

.....
Factors

Response

Prepare for raw data

Script walkthrough 1

- Open a blank text document, and prepare to write this script
 - The data is contained in three files:
 - 11_CFA_Run1Counts.csv
 - 11_CFA_Run2Counts.csv
 - 11_CFA_Run3Counts.csv
 - Load in the data and concatenate it into a single data frame

```
# load in the data from the three runs into three separate data frames t1,  
t2, t3  
t1 = read.csv("11_CFA_Run1Counts.csv")  
t2 = read.csv("11_CFA_Run2Counts.csv")  
t3 = read.csv("11_CFA_Run3Counts.csv")  
  
# concatenate the three data frames into a single data frame  
colony = rbind(t1, t2, t3)
```

Example code:
11_CFAcountData.R

Import raw data

Script walkthrough 2

- Data is by default read in as factors, i.e. all input strings are enumerated and stored as numbers
- The three separate data frame have no indication of which number they came from. We will add a column indicating this:

```
# add the missing Run column - factors are stored as numbers !
runNum <- factor( rep( 1:3, each=36 ), labels=c("Run1","Run2","Run3") )
colony <- cbind( "Run" = runNum, colony )

# reorder factor levels in their natural order (instead of alphabetical)
colony$Treatment <- factor(colony$Treatment, c("OZ", "LZ", "MZ", "HZ"))
colony$Plate <- factor(colony$Plate, c("KDX","SCX","CNX"))

# show the full table
colony
```

The tapply function

a brief digression

- Assume we have the following data for heights of 5 males and females:

```
data <- data.frame(gender=c("Male", "Male", "Female",  
                           "Female", "Female"), height=c(6, 6.1, 5.8, 6, 5.95))
```

```
gender height  
1   Male    6.00  
2   Male    6.10  
3 Female    5.80  
4 Female    6.00  
5 Female    5.95
```

- By calling `mean()` on the height column we can get the average of all 5 people, but how do we get average separately for males and females?
- tapply()** lets us do exactly this
 - It applies a function to grouped data:
- tapply(data\$height, data\$gender, mean)**
 data groups function

Undertake data analysis

Script walkthrough 3

- We need the means of the triplicate counts for each Run
 - Broken down by plate type (KDX,SCX,CNX) and Z treatment concentration (OZ,LZ,MZ,HZ)

Part 2. Investigating data

```
tapply(colony$Count, list(colony$Run, colony$Plate,  
colony$Treatment), mean)
```

, , OZ

	KDX	SCX	CNX
Run1	98.33333	129.6667	108.3333
Run2	180.33333	206.0000	188.6667
Run3	282.33333	288.6667	265.6667

We can plot a graph of this. It gives us the variation in counts per run

```
par(oma=c(4,2,2,2))  
boxplot(Count~Run*Plate*Treatment, las=2, cex=0.2,  
data=colony)
```

, , LZ

	KDX	SCX	CNX
Run1	75.0000	53.0000	21.66667
Run2	136.3333	103.6667	32.00000
Run3	157.0000	180.6667	46.66667

Better still, lets plot a **grouped** bar chart of mean counts per plate type per Z treatment

```
barplot(tapply(colony$Count, list(colony$Plate,  
colony$Treatment), mean), beside=T)
```

, , MZ

	KDX	SCX	CNX
Run1	29.66667	6.333333	0.3333333
Run2	47.00000	11.66667	2.0000000
Run3	73.00000	17.33333	3.666667

, , HZ

	KDX	SCX	CNX
Run1	6.333333	1.333333	0.3333333
Run2	12.00000	0.333333	0.0000000
Run3	18.66667	0.333333	0.0000000

Summarize & save the analysis

Script walkthrough 4

- we need a reshaped, background corrected, table of results on which to perform our tests
- for clarity where possible use dollar (\$) notation (work only with data frames)

```
### Part 3. Summarizing data ###
```

```
result <- tapply(colony$Count, list(colony$Treatment, colony$Plate), mean)
result <- data.frame(result) # result of tapply is matrix, convert to dataframe
result
```

```
# calculate kdx and scx values after background correction
```

```
kdx = result$KDX - result$CNX
scx = result$SCX - result$CNX
```

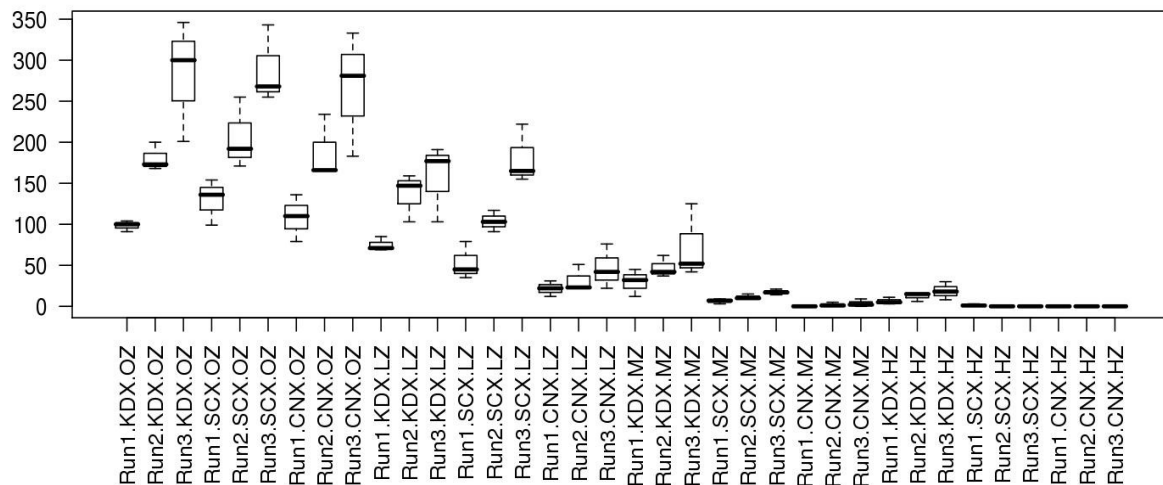
```
result <- cbind(kdx, scx)
# remove the 0Z entry
result <- result[-1,]
barplot(result, beside=T)
```

-ve subscripts
mean 'delete'

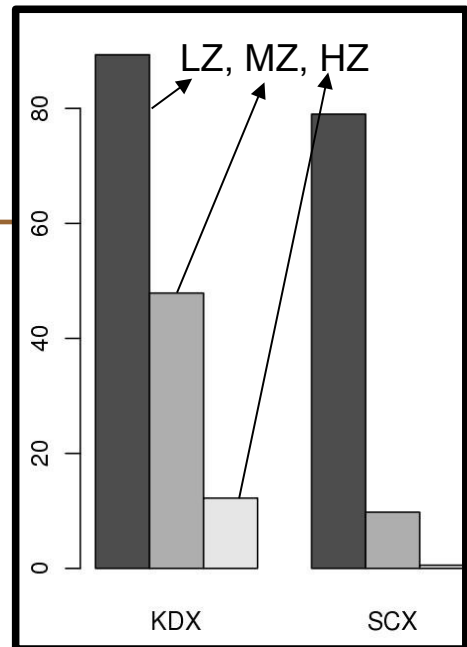
```
wilcox.test(result[,1], result[,2], paired=T)
cor.test(result[,1], result[,2], paired=T)
write.csv(result, "CFAresults.csv")
```

We can plot the results as a barchart, and undertake an appropriate two sample classical test

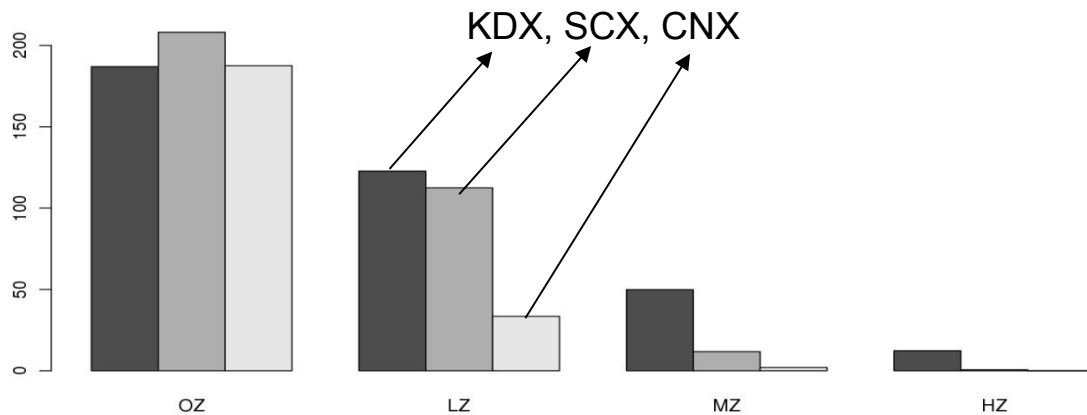
We find that the difference in means is not significant (would expect observation to occur 1:4 times), and that the scramble and knockdown counts have a 90% correlation



Boxplot shows variation in replicates. Run 3 had greatest count variation.



Bar chart shows KDX had greater viability in Z compared to SCX. Treatments are Low, Medium and High.



Bar chart shows run average counts of various plate types in different Z treatments. KDX is the only plate type that had growth in high Z.

```
> wilcox.test(result[,1],result[,2],paired=TRUE)
```

Wilcoxon signed rank test

data: result[, 1] and result[, 2]

V = 6, p-value = 0.25

alternative hypothesis: true location shift is not equal
to 0

```
> cor.test(result[,1],result[,2],paired=TRUE)
```

Pearson's product-moment correlation

data: result[, 1] and result[, 2]

t = 2.5584, df = 1, p-value = 0.2372

alternative hypothesis: true correlation is not equal
to 0

sample estimates:

cor

0.9313792

Would expect to see trend 1 in 4 times. There is a 93% correlation between the knockdown of gene X and scramble control and cell counts response when grown in compound Z.

Script steps review

Script walkthrough 5

- Excel formatted data needs to be exported as comma separated values text (or tab!)
- Get the data into R
 - `read.csv()` ... to assign the data to an object
- Produce exploratory plots
 - `boxplot()` ↑
 - `barplot()` ↑
- Undertake statistical tests
 - `cor.test()` ↑
 - Spearman's rank correlation test
 - `wilcox.test()` ↑
 - Wilcoxon test with two sets of paired data ... Mann-Whitney U test
- Write out the results
 - `write.csv()` ↑
 - exports data as comma separated list
 - `save.image()` ↑
 - could also save the R environment after analysis (we didn't do this)

Exercise

Colony forming assay script

- Enter the text of the count data script, and save the file.
 - To run the count data script in R, type
 - `source("filename")` \uparrow # the script is available as 11_countData.R
- Each step of the script is executed, and the results displayed.
- We need to export the graphical output to a file, and the R objects also need to be saved.
 - Modify the script as follows:

Section 3, line directly after ***tapply*** command insert:

```
jpeg(file="fig1.jpg",width=1600,height=800,res=150)
par(oma=c(4,2,2,2))
... <boxplot commands in middle> ...
dev.off()
jpeg(file="fig2.jpg",width=675,height=900,res=150)
... <barplot commands in middle> ...
dev.off()
```

Section 4, line directly after ***result*** command insert:

```
jpeg(file="fig3.jpg",width=1600,height=800,res=150)
... <barplot commands in middle> ...
dev.off()
```


Batch processing R scripts

Scripting

- Scripts can be run without ever launching R, using R CMD batch mode.

quit R and type the following in a linux terminal

```
R CMD BATCH -no-restore 11_CFAcountData.R
```

or if you write all of graphical output to files:

```
Rscript 11_CFAcountData.R
```

 (works only with recent R versions)

Advanced example: multiple file handling

Reading files using loops and regular expressions

- Earlier we read in each of the three colony files one-by-one
- Easy in this case, but what if there were many more than three files?
- Notice that each file contains the terms 'Counts.csv'

```
# look for patterns like 'Counts.csv'  
this.pattern <- "*Counts.csv"
```

```
# find all filenames in the current directory containing this pattern  
matching.files <- dir(pattern=glob2rx(this.pattern))
```

```
# see what has been found  
matching.files
```

```
[1] "11_CFA_Run1Counts.csv" "11_CFA_Run2Counts.csv" "11_CFA_Run3Counts.csv"
```

Advanced example: multiple file handling

Reading files using loops and regular expressions

- Using '*' in the pattern is a 'wild card' – means we search for anything that ends in 'Counts.csv'
- glob2rx() is a function which translates this pattern into something the file system can recognise
- Now loop through the matching filenames and open each in turn

```
# start with an empty data frame
colony <- data.frame()

# loop through vector of file names
for(filename in matching.filesnames){
  # open each file
  t <- read.csv(matching.filename)
  # append rows
  colony <- rbind(colony, t)
}
```