Quantitative Genomics and Genetics 2019

Computer Lab 2

Kate Harline

2/7/2019

– R Markdown
– Functions – For loops
– If/else statements
– Vector and Matrix calculation

---

0. Logistics

- Thursday Section: Our usual location is booked so we'll meet at Mann B30A
- Lab1 assignment from last week was optional, this week you'll turn in a small Rmd file by next lab section

1. R Markdown

Standard text formating

- We can *italicize it using LaTeX Syntax* or *italicize it using Markdown Syntax*.
- We can also **bold things using LaTex** or **bold it using Markdown**.

Text Size

We can make text bigger

We can make text smaller

We can set it back to normal

New line characters and vertical spacing

Look at how new lines in Markdown translate to new lines in the knitted document.

Did you catch the new line in the above chunk?

How about this one?

Text alignment

This text starts at the left margin.

This text starts one inch in.

This line will be centered

Lists

- This list
- is made using
- LaTeX syntax

- This list
- is made using
- Markdown syntax

1. This is a
2. numbered list
3. using LaTeX

Tables

Let's create a table listing the Top 10 Dog Movies as ranked by US Weekly.

Movie	Year Released
Lady and the Tramp	1955
Turner and Hooch	1989
Beethoven	1992
Homeward Bound: The Incredible Journey	1993
Lassie	1994
101 Dalmations	1996
Air Bud	1997
Best in Show	2000
My Dog Skip	2000
Marley and Me	2008

Creating line breaks of different widths

(1pt thickness)

(3pt thickness)

LaTeX equations

- conditional probability: $Pr(X_1|X_2) = \frac{Pr(X_1 \cap X_2)}{Pr(X_2)}$

- matrix:
$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$
- other matrices and delimiter syntax:

$$\begin{matrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{matrix}$$

$$\text{signum}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{otherwise} \end{cases}$$

- numbered equations:

$$f'(x=0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \tag{1}$$

- centered and unnumbered equations:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
$$\sum_{n=1}^{\infty} \frac{1}{n}$$
$$\int x^3 \, dx$$

- other symbols and equations: $\sum_i^m$ or $\cup$, $\in$, π

Code chunks

```r
example.vector1 <- 1:10
mean(example.vector1)
```

```
## [1] 5.5
```

Code options

- `include = FALSE` prevents code and results from appearing in the finished file. R Markdown still runs the code in the chunk, and the results can be used by other chunks.

- `echo = FALSE` prevents code, but not the results from appearing in the finished file. This is a useful way to embed figures.

For more information on using R Markdown: http://rmarkdown.rstudio.com/lesson-1.html

---

2. Functions

- We learned that a function is something that takes in an input and gives you an output.
- Functions require "( )", whereas data objects will have "[ ]"
- R has many built in functions for commonly used methods in statistics.

```
# Examples of built in functions
example.vector1 <- c(5,2,3,7,1,1,2,9,9)
# a function that calculates the mean
mean(example.vector1)
```

```
[1] 4.333333
```

```
# a function to index specific values
which(example.vector1==3)
```

```
[1] 3
```

```
which(!example.vector1==3)
```

```
[1] 1 2 4 5 6 7 8 9
```

```
which(example.vector1 %in% c(1,3))
```

```
[1] 3 5 6
```

- We can also build custom functions.

```
# the syntax for declaring functions, note the {} after function()
log10_add <- function(input1,input2){   # all the inputs are specified within the ( )
  cat("This is a custom function \n")
  cat("The inputs are = ",input1,input2,"\n")  # showing you the inputs
  output <- log10(input1) + log10(input2)       # creating an output within the function
  cat("The output is = ",output,"\n")          # print the output
  return(output)                               # return specifies the output
}
# Now we can call our custom functions like this
log10_add(100,1000)
```

```
This is a custom function
The inputs are =  100 1000
The output is =  5
```

```
[1] 5
```

```
# Note that the variable output is not created in our workspace
ls()
```

```
[1] "example.vector1" "log10_add"
```

```
# in order to save the result of a function to a variable we have to assign it to a variable
test.output <- log10_add(100,1000)
```

```
This is a custom function
The inputs are =  100 1000
The output is =  5
```

```
test.output
```

```
[1] 5
```

Question 1

- Can you guess what is going to happen? 13 is returned (input = 2 + x = 11)

```
x <- 11
test_function <- function(y){
  output <- x + y
  return(output)
}
test_function(2)
```

- There are many ways to set your function arguments

```
test_function <- function(y=myInput){
  output <- x + y
  return(output)
}
myInput <- 3
test_function() #the default function argument is myInput
test_function(y=5) #we now overrtide the default to be 5
```

- Short intro to scope: what happens in the function, stays in the function
- Which means any object you want to use in the function should be set as an argument

```
output <- 5
x <- 1
test_function <- function(y=myInput){
  output <- x + y
  cat("The output is ",output,"\n")
  return(output)
}

myOuput<-test_function(4)
print(myOuput)
print(output)
```

Question 2

- Can you guess what is going to happen? the function returns 15, 10 + input (5)

```
test_function <- function(myInput=5){
  output <- 5 + myInput
  return(output)
}
test_function(m=10)
```

Installing and Loading packages

- We can also use functions by installing published packages if somebody else did the hard work for us.
- We can install packages that are published on CRAN by using install.packages().

```
# install.packages("ggplot2")
```

- Other packages can be install following the prompts from GitHub or Bioconductor: https://github.com/brooke-watson/BRRR
- Once the installation is complete, we to load the package into your current R session in order to use it.

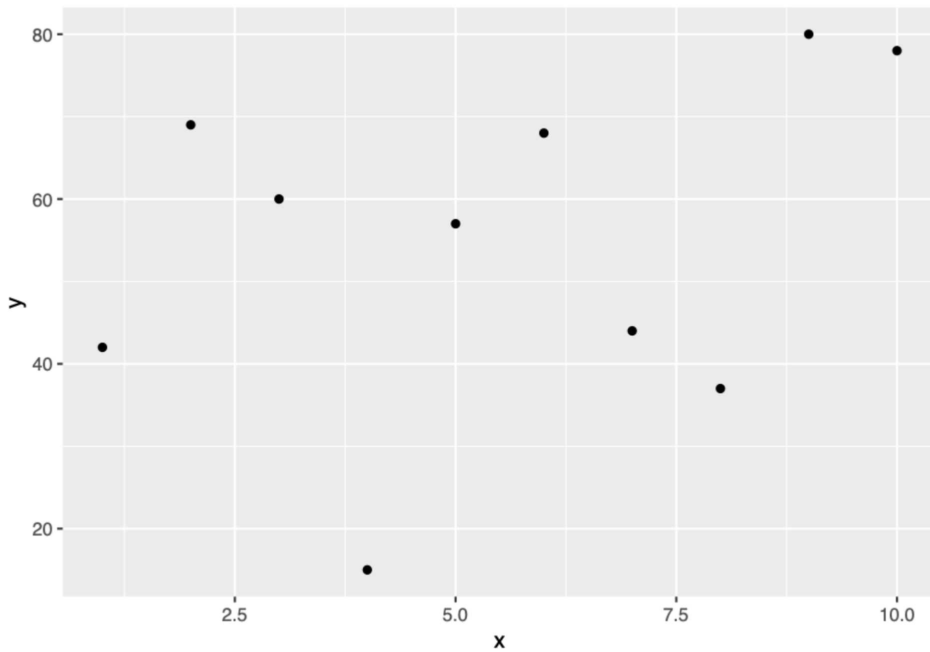
```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

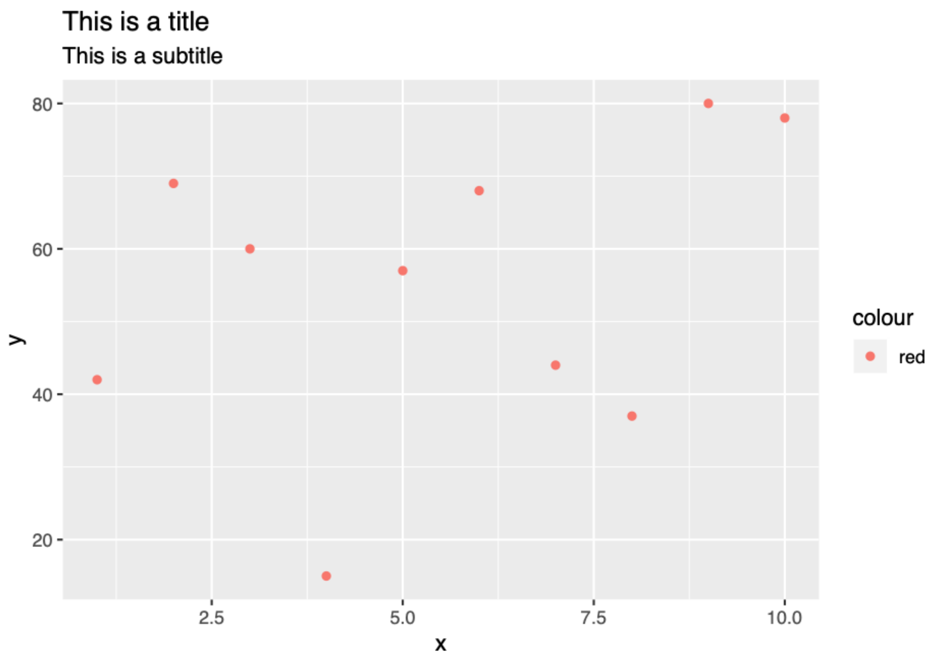
```r
require(ggplot2)
```

- Now we can use the functions from ggplot2
- ggplot is a really great function that I promise we'll get back to soon
- For now just note there are also base R functions that form plots, but ggplot looks much better

```r
x <- seq(1,10)
y <- sample(1:100,10,replace=T)
ggplot(data.frame(x,y), aes(x,y)) + geom_point()
```

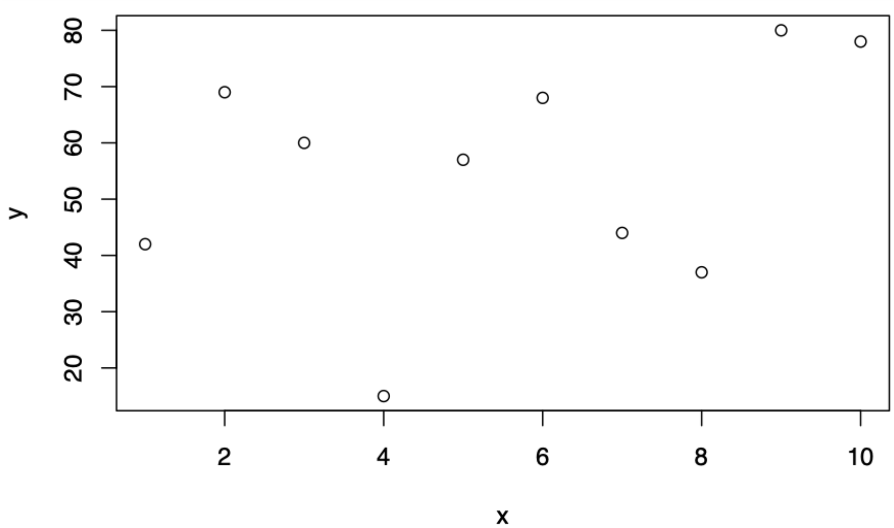



```r
ggplot(data.frame(x,y), aes(x,y,color="red")) + geom_point() +labs(title="This is a title",subtitle="Th
```



This is a title
This is a subtitle
80
60
40
20
y
2.5
5.0
7.5
10.0
x
colour
red


```r
plot(x,y)
```



80
70
60
50
40
30
20
y
2
4
6
8
10
x

3. for loops

- For loops are mainly used in cases where you want to do a task multiple times.
- Note the notation of R-command (parameters) { what to do} is the same as the function

```
N <- 3
for ( i in 1:N ){
    cat("Processing loop number = ",i,"\n")
}
```

```
## Processing loop number =  1
## Processing loop number =  2
## Processing loop number =  3
```

- You can also create a loop within a loop

```
for ( outer in 1:3 ){
    cat("Processing Outer Loop #", outer, "\n")
  for ( inner in 1:2 ){
      cat("Processing |_ Inner Loop #", inner, "\n")
    }
}
```

```
## Processing Outer Loop # 1
## Processing |_ Inner Loop # 1
## Processing |_ Inner Loop # 2
## Processing Outer Loop # 2
## Processing |_ Inner Loop # 1
## Processing |_ Inner Loop # 2
## Processing Outer Loop # 3
## Processing |_ Inner Loop # 1
## Processing |_ Inner Loop # 2
```

Question3

- What is the final value of N ? 6

```
N <- 3
for( i in 1:N){
  cat("Processing loop = ", i, "\n")
  N <- N + 1
  # cat("\tN = ", N, "\n")
}
print(N)
```

---

4. If / else statements

- By using if and else statements you can insert condition specific executions in your script
- The code inside an if statement will only be executed when the condition is TRUE

```
if (condition) {
  do stuff
} else {
  do stuff
}
```

```r
# OR you can add more levels by using else if
if(condition){
  do stuff
} else if (condition 2){
  do stuf
} else {
  do stuff
}
```

- Here is a simple example

```r
example.vector <- seq(1,25,by= 2)
# Loop over individual elements in example.vector
for( i in example.vector){
    if( i < 10 ){
        cat(i, "is smaller than 10 \n")
    } else if ( 10 <= i & i <= 20){
        cat(i, "is in the interval [10,20] \n")
    } else {
        cat(i, "is larger than 20 \n")
    }
}
```

```
1 is smaller than 10
3 is smaller than 10
5 is smaller than 10
7 is smaller than 10
9 is smaller than 10
11 is in the interval [10,20]
13 is in the interval [10,20]
15 is in the interval [10,20]
17 is in the interval [10,20]
19 is in the interval [10,20]
21 is larger than 20
23 is larger than 20
25 is larger than 20
```

---

5. Vector and Matrix calculations

- If you want to modify each element of a vector by a scalar value you can use the math operations that we have learned last week.

```r
example.vector1
```

```
## [1] 5 2 3 7 1 1 2 9 9
```

```r
2 * example.vector1
```

```
## [1] 10  4  6 14  2  2  4 18 18
```

```r
1 + example.vector1
```

```
## [1]  6  3  4  8  2  2  3 10 10
```

```
example.vector1 ^2
```

```
## [1] 25  4  9 49  1  1  4 81 81
```

- If you are interested in the dot product of two vectors you have to use a special operator

```
example.vector1 %*% example.vector1
```

```
##      [,1]
## [1,]  255
```

- The same applies for matrices

```
example.matrix1 <- matrix(c(1,1,1,2,2,2), nrow = 2, ncol = 3, byrow= TRUE)
example.matrix1
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    2    2    2
```

```
2 * example.matrix1
```

```
##      [,1] [,2] [,3]
## [1,]    2    2    2
## [2,]    4    4    4
```

```
example.matrix1 ^ 2
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    4    4    4
```

```
example.matrix1 - 1
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    1    1    1
```

- Here is how you can do matrix calculations

```
# t() is transposing the matrix
example.matrix1 %*% t(example.matrix1)
```

```
     [,1] [,2]
[1,]    3    6
[2,]    6   12
```

```
# Note the dimensions 2 x 3 %*% 3 x 2  = 2 x 2
```

- Here are some useful functions that can be used in matrix calculations

```
# creating a diagonal matrix with the first input as values on the diagonal
diag(2,nrow = 3)
```

```
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    2
```

```
# calculating the inverse of a matrix
A <- matrix(c(2,-3,1,0.5),nrow = 2)
solve(A)
```

```
      [,1]  [,2]
[1,] 0.125 -0.25
[2,] 0.750  0.50
```

```
# we can check this by
A %*% solve(A) # which results in an identity matrix
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Problem

Write this in an Rmarkdown (.Rmd) file and upload it to CMS before the next Lab section. Please include your netID in the file.

Part 1: Write a function that will print every value from 2 through 10, skipping 5 and printing 0 for each multiple of 3. Demonstrate use of for and if-else statments in this function.

```
func1 <- function(first, last, skip, mult, p){
  output <- vector()
  for (i in first:last) {
    if (i != skip){
      if(i %% mult == 0){
        output <- c(output, p)
      }
      else {
        output <- c(output, i)
      }
    }

  }
  return(output)
}
test1 <- func1(2, 10, 5, 3, 0)
test1
```

```
[1]  2  0  4  0  7  8  0 10
```

Part 2: Try to create a vector of the values printed above without using loops or if-else statements.

```
v <- c(2:10)
v <- v[c(1:3,5:9)]
v <- replace(v, v%%3 == 0, 0)
v
```

```
[1]  2  0  4  0  7  8  0 10
```