Introduction to R

Learning objectives

- Introduction to calculations, vectors, data frames, if/else, functions, etc
- Tutorial partially based on University of Michigan Genome Analysis and software carpentry courses
- www.r-project.org for download and help

R overview

- Free and has a large community of users
- R implements many common statistical procedures
- R provides excellent graphics functionality
- Useful for many data analysis and statistics projects
- R defaults to an interactive mode
- A prompt ">" is presented to users

Commenting

- Use # signs to comment.
- Anything to the right of a # is ignored by R.
- Helpful to comment code.

Assignment operator

- <- or = is the assignment operator
- Assigns values on the right to objects on the left
- Like an arrow that points from the value to the object
- Some people prefer <- and others prefer =
- For input in a function, you need = (i.e. rnorm(n=100))

```
# Create the variable A, which equals 4
A<-4

A # Display the value of A
```

```
## [1] 4
```

```
# Create the variable B, which equals 3
B=3

B # Display the value of B
```

```
## [1] 3
```

R as a Calculator

```
# Simple arithmetic
4+3
```

```
## [1] 7
```

```
# Multiplication, division, subtraction
4*((4-3)/3)
```

```
## [1] 1.333333
```

```
# Exponentiation
3 ^ 2
```

```
## [1] 9
```

```
# Square root
sqrt(9)
```

```
## [1] 3
```

```
# Basic mathematical functions are available
exp(1)
```

```
## [1] 2.718282
```

```
# The constant pi is predefined
pi
```

```
## [1] 3.141593
```

```
# Recall A=4, B=3
A+B
```

```
## [1] 7
```

```
# Check if A equals B
A==B
```

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

```
# Check if A does not equal B
A!=B
```

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

```
# Check if A is greater than B
A>B
```

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

```
# Check is A is less than B
A<B
```

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

```
# R is case sensitive
a
```

```
## Error in eval(expr, envir, enclos): object 'a' not found
```

R Vectors

- A series of numbers created with
  - c() to concatenate elements or sub-vectors
  - rep() to repeat elements or patterns
  - seq() or m:n to generate sequences
- Most mathematical functions and operators can be applied to vectors

```
# Repeats the number 1,10 times
rep(1,10)
```

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

```
# Sequence of integers between 2 and 6
seq(from=2,to=6,by=1)
```

```
## [1] 2 3 4 5 6
```

```
# Equivalent to 2:6
c(2:6)
```

```
## [1] 2 3 4 5 6
```

```
# Every 4th integer between 4 and 20
seq(from=4,to=20,by=4)
```

```
## [1]  4  8 12 16 20
```

```
# Create 2 vectors x and y
x <- c(2,0,0,4)
y <- c(1,1,2,3)

# Sum the elements of two vectors
x + y
```

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

```
# Multiply each element of x by 4
x * 4
```

```
## [1]  8  0  0 16
```

```
# Multiply x and y
x*y
```

```
## [1]  2  0  0 12
```

```
# Sum of x times y
sum(x*y)
```

```
## [1] 14
```

```
# Square root of each element of x
sqrt(x)
```

```
## [1] 1.414214 0.000000 0.000000 2.000000
```

```
# Check the length of x
length(x)
```

```
## [1] 4
```

Accessing Vector Elements

- Use the [] operator to select elements
- To select specific elements:
  - Use index or vector of indexes to identify them
- To exclude specific elements:
  - Negate index or vector of indexes

```
x
```

```
## [1] 2 0 0 4
```

```
# Select the first element, equivalent to x[c(1)]
x[1]
```

```
## [1] 2
```

```
# Exclude the first element
x[-1]
```

```
## [1] 0 0 4
```

```
# Set the first element to 3 and display
x[1] <- 3 ; x
```

```
## [1] 3 0 0 4
```

```
# Set every element except the first to 5 and display
x[-1] <- 5 ; x
```

```
## [1] 3 5 5 5
```

```
#Check which elements of x are less than 4
x<4
```

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

```
# Set values of x less than 4 to 1
 x[x<4] = 1
 x
```

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

Matrices

```
# Create a matrix from x and y
matXY<-cbind(x,y)
matXY
```

```
##      x y
## [1,] 1 1
## [2,] 5 1
## [3,] 5 2
## [4,] 5 3
```

```
# Dimension of the matrix
dim(matXY)
```

```
## [1] 4 2
```

```r
# Name the columns of matXY
colnames(matXY)<-c("X","Y") #CAREFUL WITH QUOTES
matXY
```

```
##      X Y
## [1,] 1 1
## [2,] 5 1
## [3,] 5 2
## [4,] 5 3
```

```r
# Create a matrix using matrix
matT<-matrix(c(1:16),nrow=4,ncol=4,byrow=T)
matT
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

```r
# Dimension of matrix T
dim(matT)
```

```
## [1] 4 4
```

```r
# Element in row 1, column 2
matT[1,2]
```

```
## [1] 2
```

```r
# Row 1
matT[1,]
```

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

```r
# Column 2
matT[,2]
```

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

```r
matT %*% matXY # matrix multiplication (4x4)*(4x2)=(4x2)
```

```
##        X   Y
## [1,]  46  21
## [2,] 110  49
## [3,] 174  77
## [4,] 238 105
```

```r
dim(matT %*% matXY) #check dimensions
```

```
## [1] 4 2
```

```r
matXY %*% matT # Error in matXY %*% matT : non-conformable arguments
```

```
## Error in matXY %*% matT: non-conformable arguments
```

Data Frames

- Group a collection of related vectors
- Most of the time, when data is loaded, it will be organized in a data frame

Loading Data Sets

- Load from a text file using read.table()
- Example: bp <-read.table("bp.txt",header=T)
- For help: ?read.table
- Parameters header, sep, and na.strings control useful options
- read.csv() and read.delim() have useful defaults for comma or tab delimited files
- Create from scratch using data.frame()
  - Example:

```
bp<-data.frame(height=c(150,160),weight=c(65,72))
# Careful w/ quotes in R & word formatting
colnames(bp)<-c("HEIGHT","WEIGHT")
bp
```

```
##   HEIGHT WEIGHT
## 1    150     65
## 2    160     72
```

Accessing Data Frames

- Multiple ways to retrieve columns
- The following all retrieve weight data:

```
bp["WEIGHT"]
```

```
##   WEIGHT
## 1     65
## 2     72
```

```
bp[,2]
```

```
## [1] 65 72
```

```
bp$WEIGHT
```

```
## [1] 65 72
```

- The following excludes weight data:

```
bp[,-2]
```

```
## [1] 150 160
```

Lists

- Collections of related variables
- Created with list function

```
point <- list(x = 1, y = 1)
point
```

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

- Access to components follows similar rules as for data frames, the following all retrieve x:

```
point$x; point["x"]; point[1]; point[-2]
```

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

```
## $x
## [1] 1
```

```
## $x
## [1] 1
```

```
## $x
## [1] 1
```

Programming Constructs

- if ... else ...
- for loops
- repeat loops
- while loops

Example: if ... else ...

```
vec<-rep(c(0,1),times=7)
vec
```

```
##  [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

```
#Set i to 1
i<-1
if (vec[i] == 1){ print(paste("Yeah, the number", vec[i]))
}else{print(paste("Yippy, the number", vec[i]))
}
```

```
## [1] "Yippy, the number 0"
```

for

- Loop through variable in for statement

Example: for

```
# Cycle through the whole vec vector; paste in the whole section below
for(i in 1:length(vec)){
if (vec[i] == 1){
print(paste("Yeah, the number", vec[i]))
}else{
print(paste("Yippy, the number", vec[i]))
}# close the if else statement
} #close the for loop
```

```
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
## [1] "Yippy, the number 0"
## [1] "Yeah, the number 1"
```

repeat

- Continually evaluate expression
- Loop must be terminated with break statement

Example: repeat

```
# Sample with replacement from a set of N=10 objects until the number 7 is sampled twice
N<-10
M <- matches <- 0
repeat{
# Keep track of total connections sampled, increase by 1 each time
M <- M + 1
# Sample a new number from the set of 1 to 10
p = sample(N, 1)
# Increment matches whenever we sample 7
if (p == 7){matches <- matches + 1}
# Stop after 2 matches
if (matches == 2){break}
}
print(paste("Loop ran",M,"times in order to find",matches,"matches for the number 7"))
```

```
## [1] "Loop ran 26 times in order to find 2 matches for the number 7"
```

while

- While expression 1 is false, repeatedly evaluate expression 2

Example: while

```
# Sample with replacement from a set of N=10 objects until 7 and 8 are sampled consecutively
N<-10
match <- FALSE
while (match == FALSE)
{
```

```
# Sample a new element
p = sample(N, 1)
# If not 7, then go to next iteration
if (p != 7)
   next;
# Sample another element
q = sample(N, 1)
# If not 8, then go to next iteration
if (q != 8)
next;
   match = TRUE;
}
c(p,q)
```

```
## [1] 7 8
```

Functions in R

- As tasks become complex, it is a good idea to organize code into functions that perform defined tasks
- In R, it is good practice to give default values to function arguments

Function definitions

Of the form: name <- function(argument1, argument2, ...){ expression } Arguments can be assigned default values Return value is the last evaluated expression or can be set explicitly with return()

Defining Functions

```
square <- function(x = 10){
 x * x
}

# Default value is 10
square()
```

```
## [1] 100
```

```
# Run with 2
square(2)
```

```
## [1] 4
```

Basic Utility Functions

- length() returns the number of elements
- mean() returns the sample mean
- median() returns the sample mean
- range() returns the largest and smallest values
- unique() removes duplicate elements
- summary() calculates descriptive statistics
- diff() takes difference between consecutive elements
- rev() reverses elements

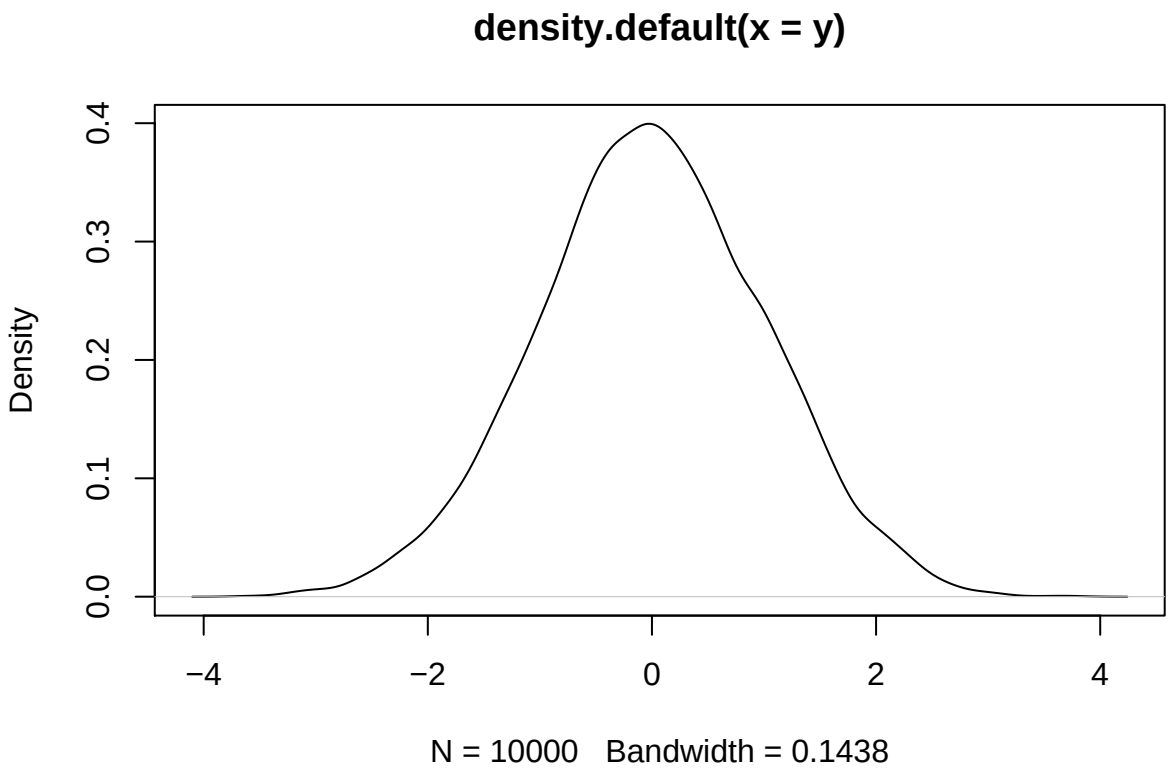
Random Generation in R

- set.seed(seed) can be used to select a specific sequence of random numbers
- sample(x, size, replace = FALSE) generates a sample of size elements from x
  - If x is a single number, sample is from 1:x

Random Generation

- rnorm(n, mean = 0, sd = 1)
  - Samples from Normal distribution
- Binomial: rbinom(n, size, prob); Uniform; runif(n, min = 1, max = 1); t: rt(n, df), and more!
- Example:

```
# Set the seed for reproducibility
set.seed(1)
# Generate 10000 samples from a normal distribution with mean 0 and sd 1
y<-rnorm(10000,mean=0,sd=1)
# Check the density plot (bell curve)
plot(density(y))
```



R Help System

- R has a built-in help system with useful information and examples
- help() provides general help
- help(plot) will explain the plot function or ?plot
- help.search("histogram") will search for topics that include the word histogram
- example(plot) will provide examples for the plot function

Useful statistics functions

- First, generate data.

```
# Set the seed for reproducibility
set.seed(1234)
# Sample size 100
n<-100

# Generate exposure x1, 0/1
x1<-rbinom(n,size=1,prob=0.5)

# Generate exposure x2, 0/1/2
x2<-rbinom(n,size=2,prob=0.5)

# Generate outcome y as a function of x1 and x2
y<-rnorm(n,mean=(0.9*x1+0.8*x2),sd=1)

# Create a matrix
dataS<-cbind(x1,x2,y)

# Create a data frame
dataS<-data.frame(dataS)

# Display the first 5 rows of dataS
dataS[1:15,]
```

```
##    x1 x2           y
## 1   0  0  0.41452353
## 2   1  1  1.22528153
## 3   1  1  1.76599349
## 4   1  0  0.39752222
## 5   1  0  0.07400141
## 6   1  1  1.86698928
## 7   0  0 -0.89626463
## 8   0  0  0.16818539
## 9   1  1  2.05496826
## 10  1  0  0.84789488
## 11  1  1  1.50406538
## 12  1  0  0.25093025
## 13  0  2  0.49023277
## 14  1  0  1.74927420
## 15  0  0  0.02236253
```

- Summary statistics.

```
# Mean and sd of y
round(c(mean(dataS$y),sd(dataS$y)),2)
```

```
## [1] 1.29 1.20
```

```
# Mean and sd of y for x1==0
round(c(mean(dataS$y[dataS$x1==0]),sd(dataS$y[dataS$x1==0])),2)
```

```
## [1] 1.05 1.23
```

```r
# Mean and sd of y for x1==1
round(c(mean(dataS$y[dataS$x1==1]),sd(dataS$y[dataS$x1==1])),2)
```

```
## [1] 1.59 1.11
```

```r
# Correlation of x1 and y
cor(x1,y)
```

```
## [1] 0.2242722
```

- Test if y is associated with x1 using a t test or test of correlation.

```r
# Test correlation (pearson, could use method = "spearman")
cor.test(dataS$y,dataS$x1,alternative = "two.sided",method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  dataS$y and dataS$x1
## t = 2.2782, df = 98, p-value = 0.02489
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.0291376 0.4029403
## sample estimates:
##       cor
## 0.2242722
```

```r
# t test
t.test(dataS$y[dataS$x1==1],dataS$y[dataS$x1==0],alternative = "two.sided")
```

```
##
##  Welch Two Sample t-test
##
## data:  dataS$y[dataS$x1 == 1] and dataS$y[dataS$x1 == 0]
## t = 2.3005, df = 96.903, p-value = 0.02357
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.07393957 1.00355442
## sample estimates:
## mean of x mean of y
##  1.590554  1.051807
```

```r
#Wilcoxon rank sum exact test
wilcox.test(dataS$y[dataS$x1==1],dataS$y[dataS$x1==0], alternative = "two.sided")
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  dataS$y[dataS$x1 == 1] and dataS$y[dataS$x1 == 0]
## W = 1586, p-value = 0.0159
## alternative hypothesis: true location shift is not equal to 0
```

- Linear regression.

```r
# Simple linear regression, x1 associated with y
lm1<-lm(y~x1,data=dataS)
summary(lm1)
```

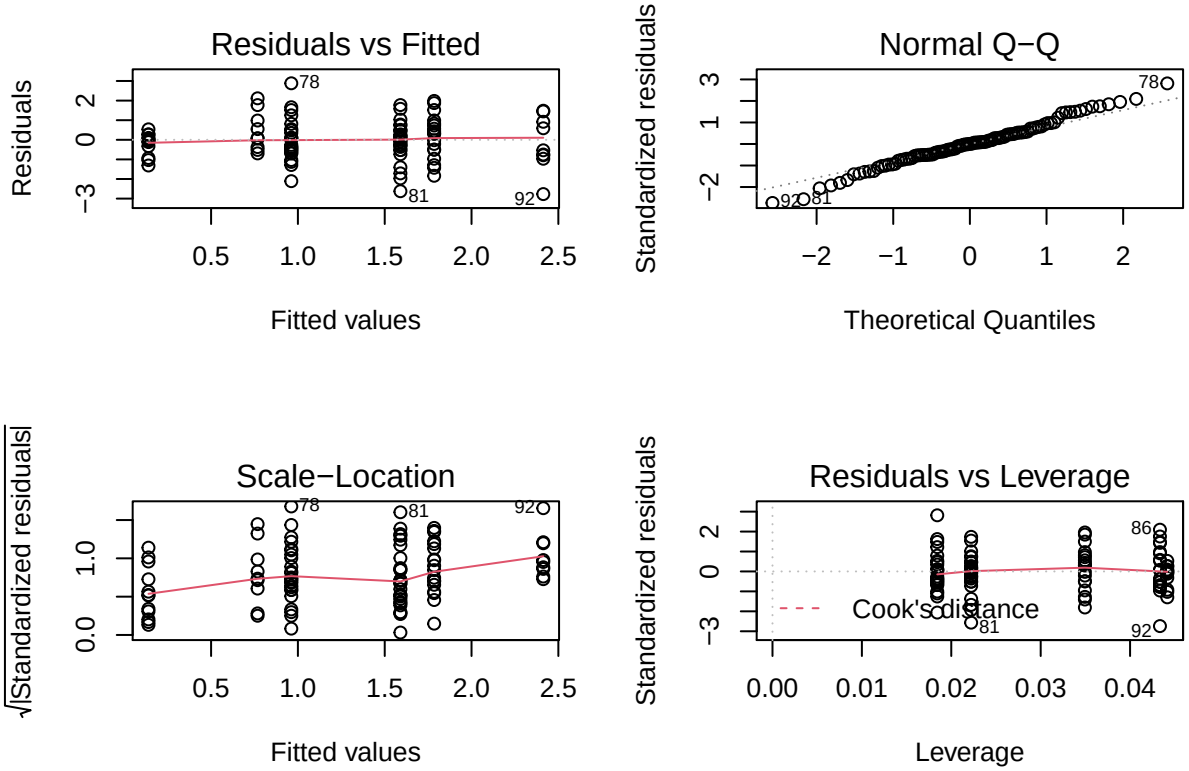
```
##
```

```
## Call:
## lm(formula = y ~ x1, data = dataS)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.62277 -0.74681 -0.08463  0.72669  2.79196
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.0518     0.1586   6.630 1.84e-09 ***
## x1            0.5387     0.2365   2.278   0.0249 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.176 on 98 degrees of freedom
## Multiple R-squared:  0.0503, Adjusted R-squared:  0.04061
## F-statistic:  5.19 on 1 and 98 DF,  p-value: 0.02489
```

```
# Multiple linear regression, x1 associated with y adjusting for x2
lm12<-lm(y~x1+x2,data=dataS)
summary(lm12)$coef
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 0.1393645  0.2171744 0.6417169 5.225705e-01
## x1          0.6284954  0.2083720 3.0162185 3.267918e-03
## x2          0.8226941  0.1501900 5.4776883 3.393599e-07
```

```
# Check residual plots
par(mfrow=c(2,2))
plot(lm12)
```



- Basic power functions. You need to have the package pwr installed first. Enter install.packages("pwr") in the command prompt and pick a mirror. Don't need to install a package everytime. Use library to laod the package.

```
#install the package
#install.packages("pwr")

#load the library
library(pwr)

# power for test of correlation
pwr.r.test(n = 100, r = cor(x1,y), alternative = c("two.sided"))
```

```
##
##      approximate correlation power calculation (arctangh transformation)
##
##               n = 100
##               r = 0.2242722
##       sig.level = 0.05
##           power = 0.6167201
##     alternative = two.sided
```

Managing Workspaces

- As you generate functions and variables, these are added to your current workspace
- Use ls() to list workspace contents and rm() to delete variables or functions
- When you quit, with the q() function, you can save the current workspace for later use

R resources

- Harvard Catalyst or other free Harvard courses https://online-learning.harvard.edu/subject/r
- There are online courses through coursera https://www.coursera.org/learn/r-programming
- Software carpentry offers really fun 2 day workshops. You can check when there is one in Boston and make sure to sign up right away because they fill up quickly https://software-carpentry.org/workshops/