18.05 R Tutorial 1A: Basics

Time

We estimate this tutorial (1A and 1B) will take 20-30 minutes. That includes time for a bit of playing around with the commands.

Conventions

```
# This is an edited transcript of a R session.
# We've inserted comment lines, which begin with a '#',
```

Command lines begin with a '>'. You should enter everything after the '>' and hit return. R's response will be in the line or lines below the command.

Trick: you can use the up arrow to find previous commands.

Using a Tutorial

In order to (begin to) learn the commands you should have RStudio open and go through the tutorial line-by-line. When the tutorial shows you a command try it in RStudio. Then make up a few of your own variants and try them. If you wonder about the effect of some command, try it. The worst that will happen is R will print out an error message.

Introduction

R is a full featured statistics package as well as a full programming language. This is not a programming class so we will only ask you to issue simple commands. We will not ask you to do serious programming. Even so, you will be able to run statistical simulations and make beautiful plots of your data.

Starting R

If you've installed R and RStudio then simply start the RStudio application. The command window is the one with the >. It's probably the window in the lower left.

R as a Calculator

# The basic operations are *,+,-,/,^.

```
> 2+3
[1] 5
> 2*3
[1] 6
> 2/3
[1] 0.6666667
> 2^3
[1] 8
> 2*(3+1)^2
[1] 32
```

Don't worry for now about the `[1]`. It will play more of a role when R is printing numbers in a list.

Using Variables

# You can store results in variables and use them in calculations.

```
> x = 2+3
# When you assign a value to a variable it does not echo the answer to
the screen. You can see the value of x by just using x as a command.
>x
[1] 5
> y = 1+2
> x*y
[1] 15
> z = x^y
> z
[1] 125

# R has another notation for assignment: the arrow: <- . Many R
programmers use this. It may seem odd to programmers coming from other
languages.
> x <- 3
> x
[1] 3
> x <- 5.412
> x
[1] 5.412
```

Vectors

# A vector is a type of list. Often it is a list of numbers, but it can be a list of other types such as characters.
# You create vectors by using the `c()` function.

```
# A vector with 4 entries
> c(1, 2, 3, 4)
[1] 1 2 3 4
# You can store vectors in variables.
> x = c(1.1, 0.0, 3.14, 2.718)
> x
[1] 1.100 0.000 3.140 2.718

# Of course using the arrow instead of equal sign works here.
> x <- c(2,4,6)
> x
[1] 2 4 6

# Sequences of integers are so common that there is a shortcut for making
them.
> 1:4
[1] 1 2 3 4
> 3:10
[1]  3  4  5  6  7  8  9 10
> 9:2
[1] 9 8 7 6 5 4 3 2

# A long vector will be displayed over several lines. At
the start of each line in brackets is the index of the first entry on
that line.
> x = 1:40
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14
 [15] 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 [29] 29 30 31 32 33 34 35 36 37 38 39 40
```

Vector Arithmetic

# You can add a number to a vector. This means adding the number to every element of the vector.

```
> x = c(1,3,5)
> x + 7.1
[1]  8.1 10.1 12.1

# Subtraction, multiplication, division and powers work the same way.
> x = c(1,3,5)
> 7*x
[1]  7 21 35
> x/7
[1] 0.1428571 0.4285714 0.7142857
> 7/x
[1] 7.000000 2.333333 1.400000
> x^6
[1]     1   729 15625
> x^7
[1]     1  2187 78125
> 7^x
[1]     7   343 16807
```

# You can add, subtract, multiply and divide vectors of *the same size.*
# Check that the following give the right answers.

```
> x = c(1,2,3)
>> y = c(4,5,6)
> x+y
[1] 5 7 9
> x-y
[1] -3 -3 -3
> z = x*y
> z
[1]  4 10 18
> z = x/y
> z
[1] 0.25 0.40 0.50

# You can even raise a vector to another vector of the same length
> x^y
[1]   1  32 729
```

Accessing entries in a vector.

```
# Entries in vectors are found with the notation x[j]
> x = c(2,4,6,8,10)
> x[1]
[1] 2
> x[2]
[1] 4
> x[3]
[1] 6
> x[4]
[1] 8
```

```
# R allows you to access more than one entry at a time
# x has 8 elements
> x = 2*c(1,2,3,4,5,6,7,8)
> x
[1]  2  4  6  8 10 12 14 16
# Notice that we have to put a vector of indices inside the brackets to
access the first and second entries in x
> x[c(1,2)]
[1] 2 4
> x[c(1,3,5)]
[1]  2  6 10
# We can access the same entry multiple times.
> x[c(2,2,2,1)]
[1] 4 4 4 2
# Using the colon method of creating vectors is useful here.
> x[2:5]
[1]  4  6  8 10
```

Functions on Vectors

R has all the functions you know and love. Most of them can be used on vectors.

```
# We'll start with functions numbers
> sin(1)
[1] 0.841471

> sin(1.4)
[1] 0.9854497

> sin(3)
[1] 0.14112

# R knows about pi
> pi
[1] 3.141593

> sin(pi/2)
[1] 1
> sin(pi/2)

# The exponential function is given by 'exp'.
> exp(0)
[1] 1

> exp(1)
[1] 2.718282

# Sin acts on vectors by taking the sin of each element.
```

```
> x = c(1,2,3,4)
> x
[1] 1 2 3 4
> sin(x)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025

# exp also acts on vectors.
> x = c(1,2,3,4)
> exp(x)
[1]  2.718282  7.389056 20.085537 54.598150

# In 18.05 we will use the sum and mean functions on vectors. They take
the sum and average respectively of the vectors entries
> x = 1:6
> x
[1] 1 2 3 4 5 6
> sum(x)
[1] 21
> mean(x)
[1] 3.5

# Example: find the sum of the integers from 1 to 1024.
> x = 1:1024
> sum(x)
[1] 524800

# This can be done in one command.
> sum(1:1024)
[1] 524800
```

A few more examples with powers

```
# Example: find the sum of the squares of the integers from 1 to 1024.
> x = 1:1024
> sum(x^2)
[1] 358438400

# This can be done in one command.
> sum((1:1024)^2)
[1] 358438400
```

Matrices

# R's method of creating matrices is a little bit clunky. It works by arranging the entries in a vector into rows and columns.

```
# A vector of length 10 can be arranged as
a 2x5 or a 5x2 matrix
# Again the syntax is clunky but very clear
> x = 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> y = matrix(x,nrow=2,ncol=5)
> y
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> z = matrix(x,nrow=5,ncol=2)
> z
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10

# Notice in the examples above that R builds the matrices one column at a
time. That is, our vector 1 to 10 runs in sequence down the columns. For
18.05 this is usually fine.

# However the matrix() function, like most R functions, has a lot of
optional parameters that allow you to control its behavior. Here's how to
make it buid a matrix by rows
> z = matrix(x,nrow=2,ncol=5, byrow = TRUE)
> z
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

Accessing Entries in Matrices

# The square bracket notation is used by giving both the row and column indices.

```
> x = 1:10
> y = matrix(x,nrow=2,ncol=5)
> y
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> y[1,1]
[1] 1
> y[2,3]
[1] 6
```

Sums and Means on Matrices

# Taking the sums or means of the columns (or rows) of a matrix will be very useful to us.

```
# Start by creating a matrix to practice on.
> x = 1:10
> y = matrix(x,nrow=2,ncol=5)
> y
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

# To sum each of the columns we use the colSums function.
> y
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

# y has five columns so colSums(y) produces 5 numbers.
> colSums(y)
[1]  3  7 11 15 19

# y has two rows so rowSums(y) produces 2 numbers.
> rowSums(y)
[1] 25 30


# Likewise rowMeans(y) and colMeans(y):
> rowMeans(y)
[1] 5 6
> colMeans(y)
[1] 1.5 3.5 5.5 7.5 9.5
```

Getting Help

```
# R and RStudio have complete documentation on all R functions. The lower
right window in RStudio has a help tab you can use. The help contains a
lot of information, so you will have to learn to filter out what you
don't need.

# It's often faster to ask for help from the command line using a
question mark.
> ?mean
# Try it, the result will appear in the help window.
```