A Crash Course in R

1 Introduction

What is R?

- R is free statistical software.
- R is a programming language.
  - It is an open source implementation of the S programming language.
  - It is highly extendable. Users can write functions and easily add software libraries to R.
  - It is interactive. You type what you want and get out the corresponding results.

2 Help

There are many ways to get help for R:

- On the Internet at www.r-project.org
- If you know the command you want help for, from the command line type:
  - `help(command)`
    - E.g., `help(lm)`
  - `?command`
    - E.g., `?lm`
- If you only know the topic you want help for, from the command line type:
  - `??topic`
    - E.g., `??logarithm`

3 Data Structures

R operates on **data structures**. A data structure is simply some sort of "container" that holds certain kinds of information.

Common R data structures:

- Vector (a sequence of numerical, character, factor, or logical data).
- Matrices (multi-dimensional collection of vectors of the same type)
- Data Frame (multi-dimensional collection of possibly different data types)

A **vector** is a sequence of values of the same data type.

The `c` function (concatenate) can be used to join data from end to end to create vectors.

- `c(1, 2, 5.3, 6, -2, 4)`
- `c("one", "two", "three")`
- `c(TRUE, TRUE, FALSE, TRUE)`

The `seq` function (sequence) can be used to create an equidistant series of values.

- A sequence of numbers from 1 to 10 in increments of 1.
  - `seq(1, 10)`
  - `1:10`
- A sequence of numbers from 1 to 20 in increments of 2.
  - `seq(1, 20, by = 2)`
- A sequence of numbers from 10 to 20 of length 100
  - `seq(10, 20, len = 100)`

The `rep` function (replicate) can be used to create a vector by replicate values.

- Repeat the sequence 1, 2, 3 three times in a row.
  - `rep(1:3, times = 3)`
- Repeat "trt1" once, "trt2" twice, and "trt3" three times.
  - `rep(c("trt1", "trt2", "trt3", times = 1:3)`

- To store a data structure in the computer's memory we must assign it to an object.
- Data structures can be stored using the assignment operator "<-"
  - E.g., store the sequence 1 through 5 in an object named `v1`.
  - `v1 <- 1:5`
- To access the data stored in an object, we simply type the variable name into R and hit enter.
  - `v1`
- Vectors can be combined and stored in an object using the c function and the assignment operator.
  - `v2 <- c(1, 10, 11)`
  - `new <- c(v1, v2)`
  - `new`

- **Categorical** data should be stored as a factor in R.
- The factor function takes vectors of any data type and converts them to factors.
- Examples:

```
of1 <- factor(rep(1:6, times = 3))
of1
of2 <- factor(c("a", 7, "blue", "blue"))
of2
```


4 Helpful Functions

General Functions

- `length(x) #length of x`
- `sum(x) #sum elements in x`
- `mean(x) #mean of elements in x`
- `var(x) #sample variance of elements in x`
- `sd(x) #standard deviation of elements in x`
- `range(x) #range of elements in x`
- `log(x) #ln of elements in x`
- `summary(x) #5-number summary of x`

Functions related to statistical distributions

Suppose that a random variable X has the "dist" distribution

- `p[dist](q, ...)` – returns the cdf of X evaluated at q, i.e., $p = \Pr(X \leq q)$.
- `q[dist](p, ...)` – returns the inverse cdf (or quantile function) of X evaluated at `p`, i.e., $q = \inf\{x: \Pr(X \leq x) \geq p\}$.
- `d[dist](x, ...)` – returns the mass or density of X evaluated at `x` (depending on whether it's discrete or continuous).
- `r[dist](n, ...)` – returns an i.i.d. random sample of size `n` having the same distribution as X.
- … indicates that additional arguments describing the shape of the distribution.

Examples:

- `pnorm(1.96, mean = 0, sd = 1)` returns the probability that a normal random variable with mean 0 and standard deviation 1 is less than or equal to 1.96.
- `qunif(0.6, min =0, max = 1)` returns the value x such that $P(X \leq x) = 0.6$ for a uniform random variable on the interval $[0, 1]$.
- `dbinom(2, size = 20, prob = .2)` returns the probability that $\Pr(X = 2)$ for $X \sim \text{Binom}(n = 20, \theta = 0.2)$.
- `dexp(1, rate = 2)` returns the density of an exponential random variable with mean = ½.
- `rchisq(100, df = 5)` returns a sample of 100 observations from a chi-squared random variable with 5 df.

5 Plotting

- The plotting capabilities of R are one of its most powerful and attractive features.
- It is relatively simple to construct histograms, (parallel) boxplots, scatterplots, etc.
- A histogram is created using the `hist` function.
- A boxplot is created using the `boxplot` function.
- A scatterplot is created using the `plot` function.

Histograms

Histogram with a custom x-axis label and title

```
x <- rnorm(100, mean = 100, sd = 10)
hist(x, xlab = "x-values",
   main = "Histogram of 100 observations from
N(100, 10^2)")
```


Boxplots

Single Boxplot

```
y <- rnorm(100, mean = 80, sd = 3)
boxplot(y)
```

Parallel Boxplot

```
grp <- factor(rep(c("Grp 1", "Grp 2"), each
= 100)) #make groups for x and y
dat <- c(x, y)
boxplot(dat ~ grp, xlab = "Group")
```


Scatterplots

Construct a scatterplot with x on the x-axis and y on the y-axis:

```
#generate vectors
x <- runif(20)
y <- 2 + 3 * x + rnorm(20)
plot(x, y)
```

Scatterplot with custom labels and title:

```
plot(x, y, xlab="1st variable", ylab="2nd
variable")
title("Title of plot")
```

Lineplot of density

```
x <- seq(-4, 4, len = 1000)
y <- dnorm(x, mean = 0, sd = 1)
plot(x, y, xlab="x", ylab="density", type =
"l")
title("Density of Standard Normal")
```


"Histogram" scatterplot of probability mass function

```
#plot of Binomial(n = 20, p = .3) pmf
x <- 0:20
y <- dbinom(x, size = 20, prob = .3)
plot(x, y, xlab="# Successes", ylab="Prob",
type = "h")
title("pmf of Binomial(n = 20, p = .3)")
```


6 Data Frames

- Date frames are created by passing vectors into the `data.frame` function.
  - The names of the columns in the data frame are the names of the vectors you give the `data.frame` function.
- Example:
  - `d <- c(1, 2, 3, 4)`
  - `e <- c("red", "white", "blue", NA)`
  - `f <- c(TRUE, TRUE, TRUE, FALSE)`
  - `mydataframe <- data.frame(d,e,f)`
  - `mydataframe`

- The columns of a data frame can be renamed using the `names` function on the data frame.
  - `names(mydataframe) <- c("ID", "Color", "Passed")`
  - `mydataframe`
- The columns of a data frame can be named when you are first creating the data frame by using "name =" for each vector of data.
  - `dataframe2 <- data.frame(ID=d, Color=e, Passed=f)`
  - `dataframe2`

- The vectors of a data frame may be accessed using "$" and specifying the name of the desired vector.
- Access the Color vector in `mydataframe`:
  - `mydataframe$Color`
- The vectors of a data frame may be accessed by specifying the desired row(s) or column(s) in square brackets.
- Access first row of `mydataframe`
  - `mydataframe[1,]`
- Access third column of `mydataframe`
  - `mydataframe[,3]`
- Access the ID column of `dataframe2` and assign it to `newID`
  - `newID <- dataframe2$ID`

7 Importing Data

- The `read.table` function imports data into R as a data frame.
- Useage: `read.table(file, header = TRUE, sep = ",")`
- `file` is the filepath and name of the file you want to import into R
- If you don't know the file path, set `file = file.choose()` will bring up a dialog box asking you to locate the file you want to import.

- `header` specifies whether the data file has a header (labels for each column of data in the first row of the data file).
  - If you don't specify this option in R or use `header=FALSE`, then R will assume the file doesn't have any headings.
  - `header=TRUE` tells R to read in the data as a data frame with column names taken from the first row of the data file.

- `sep` specifies the delimiter separating elements in the file.
  - If each column of data in the file is separated by a space, then use `sep = " "`
  - If each column of data in the file is separated by a comma, then use `sep = ","`
  - If each column of data in the file is separated by a tab, then use `sep = "\t"`.

8 Accessing Elements of a Data Structure

- Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets.
  - `a <- seq(2, 16, by = 2)`
  - `a`
- Access the 2nd, 4th, and 6th elements of `a`.
  - `a[c(2, 4, 6)]`
- Access all elements in `a` EXCEPT the 2nd, 4th, and 6th.
  - `a[-c(2, 4, 6)]`
- Access all elements in `a` except elements 3 through 6.
  - `a[-(3:6)]`

- Sometimes we need to know if the elements of an object satisfy certain conditions. This can be determined using the logical operators `<`, `<=`, `>`, `>=`, `==`, `!=`
  - "`==`" means "equal to" and "`!=`" means not equal to.
- values of `a` greater than 10
  - `a > 10`
- values of `a` less than or equal to 4
  - `a <= 4`
- values of `a` equal to 10
  - `a == 10`
- values of `a` not equal to 10
  - `a != 10`

- More complicated logical arguments can be made using & and |.
  - & means "and"
  - | means "or"
- Elements of `a` greater than 6 and less than or equal to 10
  - `(a > 6)&(a <= 10)`
- Elements of `a` less than or equal to 4 or greater than or equal to 12.
  - `(a <= 4)|(a >= 12)`

- Logical statements can be used to return parts of an object satisfying the appropriate criteria.
- Return elements of `a` less than 6.
  - `a[a < 6]`
- Return elements of `a` equal to 10.
  - `a[a == 10]`
- Return elements of `a` less than 6 or equal to 10.
  - `a[(a < 6)|(a == 10)]`

9 Functions

- A function is essentially a sequence of commands executed based on certain arguments supplied to the function.

In R, a function is defined using the general format:

```
myfunction <- function(arg1, arg2, arg3)
{
   code to execute
}
```

- The name of the function is "myfunction", and to use this function, I need to supply 3 arguments.

- A function may or may not return sometime back that you can store for later use.
- To use the function, you simply type

```
myfunction(arg1, arg2, arg3)
```


Example of a function that returns the standard deviation of a vector x

The sole argument is:

- x, the vector of values for which I want to determine the standard deviation

```
stdev <- function(x)
{
   s <- sqrt(sum((x - mean(x))^2)/(length(x) - 1))
   s
}
z <- rnorm(20)
stdev(z)
```


Example of a function that returns the density of a normal random variable with mean mu and standard deviation sigma for a vector x.

The arguments are:

- x, the vector of values at which I want to determine the density
- mean, the mean of the normal distribution
- sigma, the standard deviation of the normal distribution

```
normal.density <- function(x, mu = 0, sigma = 1)
{
   return(exp(-(x -
mu)^2/(2*sigma^2))/sqrt(2*pi*sigma^2))
}
```

Example: Create function that returns the mean and standard deviation of a vector x.

The sole argument is:

- x, the vector of values for which I want to determine the mean and standard deviation

```
ms <- function(x)
{
   m <- mean(x)
   s <- sd(x)

   return(list(m = m, s = s))
}
```


Resources about R that my be helpful:

- *Introductory Statistics with R*, by Peter Dalgaard (Good introduction to a R function for doing basic statistical analysis)
- *The Art of R Programming,* by Norman Matloff (Very good reviews)
- *R Cookbook*, by Paul Teetor