R Handout: Learning to Speak R

Basic Terminology

R is a statistical programming language. In this class, we'll use R with RStudio, which is an IDE (integrated development environment). If you're describing your programming experience to someone, you might say "I've used R" - RStudio is *how* you've worked with R.

Base R and Packages

When you open R + RStudio for the fist time, there's lots of functions that are included by default (e.g., `mean()`, `lm()`, etc.). These functions are known as "Base R" - its what's included with R "out of the box." In this class, we'll make use of several *packages* like `tidyverse`. Packages are collections of functions that aren't part of base R.

To use packages in R, you first need to install them using the `install()` function. You only need to run this command once on your computer. Each time you open R, however, you'll need to *load* the packages you want to use, using the `library()` function. This is a way of telling R, "I want to use the functions in this package".

Scripts and Running Code in R

In this class, we'll general work with R scripts - in your file explorer, these files will have a `.R` file extension. Scripts are collections of R code. To run code in an R script, you can click on the line of code (or select it with your mouse) and press `Ctrl + Enter` (or `Apple logo / Cmd key + Enter` on Mac).

If you want to run several lines of code at one time, you can highlight all of the lines you want to run, then use the hot keys above. You can also use the `Run` button at the top of your screen.

Scripts and the Command Line

When you open `Rstudio`, one of the four panels you'll see is the `Console`. You can run code in the command line by clicking to the right of the `>` symbol. This can be a convenient way to check a line of code you're writing, or to access help files for a function by typing `?` and then the function name (e.g., you can type `?mean` to read documentation for the `mean` function).

AN IMPORTANT NOTE: Whenever you're doing analysis that you want to be able to locate or use later, make sure to write your code in a script file that you can save! Commands entered in the console won't be saved across R sessions (meaning, if you close R you'll lose what you've entered there).

Comments in your Code

An important part of well-written code is *comments*. You can create a comment by starting a line with a hash tag - `#`. This tells R, "ignore this next line." Leaving comments is part of *documenting* your code - explaining what you're doing so that other people can understand your analysis and read your code.

Take a look at the sample R output below:

```
# This is a comment - R will ignore this line

2 + 2
```

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

Basic Data Structures in R

Through most of this class, we'll be working with data sets. Before we jump into working with data in R, however, let's take a second to unpack how R works with data in general. Let's start by creating an *object* named a that stores the number 5 in the code below:

```
# Create object named a storing the value 5...

a <- 5

# We can now access the value stored in a in equations, functions or other code

a
```

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

```
a + 2
```

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

You can find this new object a in the Environment pane in RStudio. This object is also available now by calling a in the command line.

The Assignment Operator

Notice how we used <- above to *assign* the value of 5 to the object a? In R, <- is the *assignment operator*. You can read it as synonmous with "equals". You can use the hot key combination Alt + - to save having to type both symbols separately.

Object Types in R

In R, we can create several different kinds of objects or data types using the assignment operator. Some of the most important are listed below:

- **Numeric**: stores a single value (like a stores 5 above)
- **Logical**: stores binary values TRUE or FALSE
- **Character**: also known as string objects, these stores strings of text (like "hello world")

We can use is functions in R to check whether an object is the type that we're expecting. These functions will return either TRUE or FALSE. Take a look at the sample code below:

```
# Let's start by checking  if our a object from above is numeric

is.numeric(a)
```

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

```
# Now we can create a character object and check it using is.character()

a.string <- "123 Main St"

is.character(a.string)
```

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

```r
# When we create a logical object, note that we don't put TRUE in quotes

a.logical <- TRUE

is.logical(a.logical)
```

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

Vectors in R

Vectors are the basic "building blocks" of data in R - they're collections of items stored together. In general, these items should generally all share one of the basic R object types. To create vectors, we'll use the `c()` function to define the list of items we want to combine, like so:

```r
# Create a vector with 3 elements - note the commas between each item!

b.vector <- c(1, 2, 3)

b.vector
```

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

```r
# Now we can call functions like mean() using this new vector

mean(b.vector)
```

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

Just like our `a` object, `b.vector` is now stored in the `Environment` pane.

Dataframes in Base R

In Base R, data sets are stored as *data frames* (this will differ slightly when we start using `tidyverse` but the differences aren't all that important right now). Data frames are collections of vectors of the same length. Let's create a sample data set below:

```r
# To start, let's create vectors storing various object types

person.ID <- c(12, 24, 54, 65)
address   <- c("123 Main St", "274 Long St", "789 Right St", "467 Left St")
employed  <- c(TRUE, TRUE, FALSE, TRUE)
wage.inc  <- c(12500, 15750, 0, 14100)

# Combine each of the individual vectors into a data frame

data <- data.frame(person.ID, address, employed, wage.inc)

head(data)
```

```
##   person.ID      address employed wage.inc
## 1        12  123 Main St     TRUE    12500
## 2        24  274 Long St     TRUE    15750
## 3        54 789 Right St    FALSE        0
## 4        65  467 Left St     TRUE    14100
```

List of Useful Functions in Base R

Here's a list of commonly used base R functions for your reference. Examples are included in the code section below.

Given a vector of numeric values, you can calculate the following:

- `mean()`: average
- `median()`: median
- `var()`: variance
- `sd()`: standard deviation
- `quantile()`: percentiles (requires second argument, see below)

You can generate normally-distributed random variables using `rnorm()`, and uniformly-distributed random variables using `runif()`.

```
# Start by creating a vector with 100 draws from a normally-distributed random
# variables using the rnorm function - set mean = 1 and sd 2

normal.vector <- rnorm(100, mean = 1, sd = 2)

# Now we can use summary stats function above to inspect this vector

mean(normal.vector)
```

```
## [1] 1.071179
```

```
median(normal.vector)
```

```
## [1] 0.996973
```

```
var(normal.vector)
```

```
## [1] 4.304047
```

```
sd(normal.vector)
```

```
## [1] 2.07462
```

```
# We can calculate percentiles using the quantile function and inputting
# values between 0 and 1 for the percentile we want

quantile(normal.vector, 0.1)  # 10th percentile
```

```
##       10%
## -1.630444
```

```
quantile(normal.vector, 0.9)  # 90th percentile
```

```
##      90%
## 3.877823
```