Session 1

Nick Hathaway; nicholas.hathaway@umassmed.edu

Contents

R Basics

R is like most programming languages and operates by storing by data in `variabes/objects` and then operating on these objects with `functions`.

Variables/objects

Objects are able to store a very wide variety of data, some of the common types are explained below. To store data in R you can use two separate syntaxes, `<-` or `=`. The normal R convention is to use `<-` and there are some real subtle differences between using `<-` and `=` but for the majority of time you can get by by using either one.

```
x = 10
print(x)
```

```
[1] 10
```

```
#same as
y <- 10
print(y)
```

```
[1] 10
```

Functions

Functions are used on objects that have stored data to either output new data or to simply print information about the data stored in that object (e.g. above the function `print()` will print to the screen the data stored in the object that is given to print). You can store the output of a function into a new variable.

```
x = 9
sqrt(x)
```

```
[1] 3
```

```
y = sqrt(x)
print(y)
```

```
[1] 3
```

When people talk about functions, they will refer to the objects being given to the functions as the function's arguments. Depending on the function, it can take several different arguments. To find out more about a function and the functions it takes, you can use the `help()` function and give it the name of function or you can use `?` and then name of the function. If you don't know the name of the function exactly or want to do a keyword search you can `??` and then a keyword. You can also just go to the bottom right window in RStudio and go to the help tab.

```
help(print)
```

or

```
?print
```

or to search for any topic with the word `print`

```
??print
```

There are a large number of different functions in R that you will become accustomed to as you use R more. Also each library you load will have more functions as well. Also the arguments to functions have names and rather than giving arguments simply in the order in which they are listed in the function's definition you can give an argument by using `arg=value` syntax. For example take the `seq()` function, which is a function for creating different ranges. First call `help(seq)` to see the definition of `seq()`, whose definition looks like below.

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)
```

So using the seq function you can create a range from `2` to `4` by typing

```
print(seq(2,4))
```

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

or you can give the arguments by naming them

```
print(seq(from = 2, to = 4))
```

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

When naming the function arguments, order no longer matters

```r
print(seq(to = 4, from = 2))
```

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

When naming the arguments you don't have to name all of them

```r
print(seq(2,4, length.out = 6))
```

```
[1] 2.0 2.4 2.8 3.2 3.6 4.0
```

Data Types

Every object in R will have a different data type. To determine the type of any object just use the `class()` function.

```r
x <- 10
print(class(x))
```

```
[1] "numeric"
```

```r
y <- "Example"
print(class(y))
```

```
[1] "character"
```

Depending on the type of data, functions will have different behaviors

character

character type data is anything that can be represented by a string of letters/characters like a name or categorical data.

```r
name = "Nick"

loc = "Amp II"

condition = "Control"
```

numeric

numeric type data is anything that be represented by, well, numbers like measurements or data values

```r
speed = 10

time = 60

fraction = 0.5
```

factor

factor type data is basically character data that has been encoded into numeric values underneath but is represented by characters, this mostly used on categorical data that needs to be converted into numbers in order for certain modeling/statistics functions to work. More on factors latter.

logic/Boolean

Boolean refers to a type of data is simply either TRUE or FALSE and is normally used in conjunction with logic.

```
x = 10 > 11
print(x)
```

```
[1] FALSE
```

```
y = 12 > 11
print(y)
```

```
[1] TRUE
```

Logic Tests

operator	meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to

More Complex Data containers

The majority of the time you need to store more than just one value and therefore you will need containers that can hold multiple values at once, R comes equipped with several containers already

vectors

Vectors are just several of the same data type hold together in one container, the most common way to create a vector is to use the concatenate function, which in R is just called `c()` for short.

```
#numeric values
speeds = c(10.5, 11, 13, 14, 10)
print(speeds)
```

```
[1] 10.5 11.0 13.0 14.0 10.0
```

```r
print(class(speeds))
```

```
[1] "numeric"
```

```r
times = c(20,30,20,30,40,50)
print(times)
```

```
[1] 20 30 20 30 40 50
```

```r
print(class(times))
```

```
[1] "numeric"
```

```r
#charater values

names = c("Nick", "Michael", "Arjan", "Henry")
print(names)
```

```
[1] "Nick"    "Michael" "Arjan"   "Henry"
```

```r
print(class(names))
```

```
[1] "character"
```

```r
#Boolean/logical values

logics = c(10 < 11, 12 < 11, 10 > 9)
print(logics)
```

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

```r
print(class(logics))
```

```
[1] "logical"
```

Note: R will force everything in a container to be the same type if it can so be careful to not to actually mix types if you don't mean to.

```r
#Accidental converion to character rather than numberic vector
numbers = c(1,2,3,4,5,"6")
print(numbers)
```

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

```r
print(class(numbers))
```

```
[1] "character"
```

```r
#actual numeric vector
actualNumbers = c(1,2,3,4,5,6)
print(actualNumbers)
```

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

```r
print(class(actualNumbers))
```

```
[1] "numeric"
```

matrix

R's matrix is very similar to the vector where all things have to be the same type but contains values in rows and columns.

```r
mat = matrix(c(1,2,3,4,5,6,7,8,9,10,11,12))

print(mat)
```

```
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
[11,]   11
[12,]   12
```

```r
mat2 = matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol = 2)

print(mat2)
```

```
     [,1] [,2]
[1,]    1    7
[2,]    2    8
[3,]    3    9
[4,]    4   10
[5,]    5   11
[6,]    6   12
```

```r
mat3 = matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol = 2, byrow = TRUE)

print(mat3)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
[6,]   11   12
```

```
print(class(mat3))
```

```
[1] "matrix"
```

See `help(matrix)` for more info on how to use matrix.

data.frame

The data.frame is the main object in R and is the data container you most likely be dealing with the most. It is similar to an spreadsheet/table data structure you in something like Excel with rows and columns, both of which can have names. The data.frame is different from the matrix because each column can have different types, though all the elements in a column have to be the same type. You will rarely have to create a data.frame by hand and the majority of the time you

```
dat = data.frame(names = c("Nick", "Mike", "Arjan", "Henry", "Jill"), duration = c(5, 5, 4, 1, 4), prog
print(dat)
```

```
  names duration program
1  Nick        5  MD/PhD
2  Mike        5  MD/PhD
3 Arjan        4     PhD
4 Henry        1  MD/PhD
5  Jill        4     PhD
```

A useful function for looking at a data.frame is the `str()` function. It will tell you information about each column.

```
dat = data.frame(names = c("Nick", "Mike", "Arjan", "Henry", "Jill"), duration = c(5, 5, 4, 1, 4), prog
str(dat)
```

```
'data.frame':   5 obs. of  3 variables:
 $ names   : Factor w/ 5 levels "Arjan","Henry",..: 5 4 1 2 3
 $ duration: num  5 5 4 1 4
 $ program : Factor w/ 2 levels "MD/PhD","PhD": 1 1 2 1 2
```

With the `str()` function you can see that the we have three columns with the 1st and 3rd column being factors and the 2nd column being a numeric column. You can also see that for the variables that are factors you can see that they also have their coded numerical values next to them. This is important to note when you are dealing with typos, for instance if we had typed this instead.

```
dat = data.frame(names = c("Nick", "Mike", "Arjan", "Henry", "Jill"), duration = c(5, 5, 4, 1, 4), prog
str(dat)
```

```
'data.frame':   5 obs. of  3 variables:
 $ names   : Factor w/ 5 levels "Arjan","Henry",..: 5 4 1 2 3
 $ duration: num  5 5 4 1 4
 $ program : Factor w/ 3 levels "MD/PhD","Md/PhD",..: 2 1 3 1 3
```

You can see that we now have three levels for program rather than the two since we typed in one of the MD/PhD levels incorrectly.

Installing/Loading Libraries

To install new libraries you can use the `install.packages()` function and give it the name of a library that The Comprehensive R Archive Network(CRAN) has stored in their repositories. Below is the command from the prep work to install all the libraries we will be going in the course.

```
#For reading data in
install.packages(c("readr","readxl", "haven"))

#For organizing Data
install.packages(c("dplyr", "tidyr", "reshape2"))

#For plotting
install.packages(c("ggplot2", "RColorBrewer"))

#For Downloading more packages not managed by R in install.packages()
install.packages(c("devtools"))

#Addition commands to ggplot2
devtools::install_github("kassambara/easyGgplot2")

#Interactive Plots
install.packages(c("networkD3", "metricsgraphics", "shiny"))
devtools::install_github("rstudio/d3heatmap")
```

Once you have installed a library, you can load it's functions by using the `library()` function

```
library(ggplot2)
help(ggplot)
```

Reading in Data

Data files can come in multiple formats but some of the most common types are plaint text files that have each column delimited by either tabs `"\t"` or commas `","`. R can also read in files that saved in excel format.

File Formats

Plain Text Files

For reading in plain text files we will being using the package `readr`

```
library(readr)
```

For reading in tab delimited files we have `read_tsv()` and for comma delimited files we have `read_csv()` Here is a link to a tab delimited file Also a helpful function might be to use `file.choose()`

```
library(readr)
worEpi = read_tsv("WorEpi.tab.txt")
str(worEpi)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   22276 obs. of  8 variables:
 $ disease: chr  "TYPHOID FEVER [ENTERIC FEVER]" "DIPHTHERIA" "TUBERCULOSIS [PHTHISIS PULMONALIS]" "TUBE
 $ event  : chr  "DEATHS" "DEATHS" "DEATHS" "DEATHS" ...
 $ number : int  1 2 5 2 1 4 3 2 1 7 ...
 $ loc    : chr  "WORCESTER" "WORCESTER" "WORCESTER" "WORCESTER" ...
 $ state  : chr  "MA" "MA" "MA" "MA" ...
 $ Year   : int  1894 1894 1894 1894 1894 1894 1894 1894 1894 1894 ...
 $ Month  : int  6 6 6 6 6 6 6 6 7 7 ...
 $ Day    : int  1 1 8 15 15 29 6 6 13 20 ...
 - attr(*, "spec")=List of 2
  ..$ cols   :List of 8
  .. ..$ disease: list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ event  : list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ number : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ loc    : list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ state  : list()
  .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
  .. ..$ Year   : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ Month  : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ Day    : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  ..$ default: list()
  .. ..- attr(*, "class")= chr  "collector_guess" "collector"
  ..- attr(*, "class")= chr "col_spec"
```

or

```
library(readr)
worEpi = read_tsv(file.choose())
str(worEpi)
head(worEpi)
```

Here is a link to a comma separated file

```
library(readr)
melanoma = read_csv("Mel.csv")
str(melanoma)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   205 obs. of  7 variables:
 $ time     : int  10 30 35 99 185 204 210 232 232 279 ...
 $ status   : int  3 3 2 3 1 1 1 3 1 1 ...
 $ sex      : int  1 1 1 0 1 1 1 0 1 0 ...
 $ age      : int  76 56 41 71 52 28 77 60 49 68 ...
 $ year     : int  1972 1968 1977 1968 1965 1971 1972 1974 1968 1971 ...
 $ thickness: num  6.76 0.65 1.34 2.9 12.08 ...
 $ ulcer    : int  1 0 0 0 1 1 1 1 1 1 ...
 - attr(*, "spec")=List of 2
  ..$ cols   :List of 7
  .. ..$ time     : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ status   : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ sex      : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ age      : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ year     : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  .. ..$ thickness: list()
  .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
  .. ..$ ulcer    : list()
  .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
  ..$ default: list()
  .. ..- attr(*, "class")= chr  "collector_guess" "collector"
  ..- attr(*, "class")= chr "col_spec"
```

```
head(melanoma)
```

```
# A tibble: 6 x 7
   time status   sex   age  year thickness ulcer
  <int>  <int> <int> <int> <int>     <dbl> <int>
1    10      3     1    76  1972      6.76     1
2    30      3     1    56  1968      0.65     0
3    35      2     1    41  1977      1.34     0
4    99      3     0    71  1968      2.90     0
5   185      1     1    52  1965     12.08     1
6   204      1     1    28  1971      4.84     1
```

or

```
library(readr)
melanoma = read_csv(file.choose())
str(melanoma)
```

Excel Sheets

For excel sheets we have the library `readxl`

```
library(readxl)
```

Here is a link to an excel sheet of both previous files as separate sheets

```
library(readxl)
melanoma = read_excel("Both.xlsx")
str(melanoma)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   205 obs. of  7 variables:
 $ time     : num  10 30 35 99 185 204 210 232 232 279 ...
 $ status   : num  3 3 2 3 1 1 1 3 1 1 ...
 $ sex      : num  1 1 1 0 1 1 1 0 1 0 ...
 $ age      : num  76 56 41 71 52 28 77 60 49 68 ...
 $ year     : num  1972 1968 1977 1968 1965 ...
 $ thickness: num  6.76 0.65 1.34 2.9 12.08 ...
 $ ulcer    : num  1 0 0 0 1 1 1 1 1 1 ...
```

```
head(melanoma)
```

```
# A tibble: 6 x 7
   time status   sex   age  year thickness ulcer
  <dbl>  <dbl> <dbl> <dbl> <dbl>     <dbl> <dbl>
1    10      3     1    76  1972      6.76     1
2    30      3     1    56  1968      0.65     0
3    35      2     1    41  1977      1.34     0
4    99      3     0    71  1968      2.90     0
5   185      1     1    52  1965     12.08     1
6   204      1     1    28  1971      4.84     1
```

By default it just reads the first sheet, you can tell it which sheet to read

```
library(readxl)
worEpi = read_excel("Both.xlsx", sheet = 2)
str(worEpi)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   22276 obs. of  8 variables:
 $ disease: chr  "TYPHOID FEVER [ENTERIC FEVER]" "DIPHTHERIA" "TUBERCULOSIS [PHTHISIS PULMONALIS]" "TUBE
 $ event  : chr  "DEATHS" "DEATHS" "DEATHS" "DEATHS" ...
 $ number : num  1 2 5 2 1 4 3 2 1 7 ...
 $ loc    : chr  "WORCESTER" "WORCESTER" "WORCESTER" "WORCESTER" ...
 $ state  : chr  "MA" "MA" "MA" "MA" ...
 $ Year   : num  1894 1894 1894 1894 1894 ...
 $ Month  : num  6 6 6 6 6 6 6 6 7 7 ...
 $ Day    : num  1 1 8 15 15 29 6 6 13 20 ...
```

```
head(worEpi)
```

```
# A tibble: 6 x 8
                             disease  event number
                               <chr>  <chr>  <dbl>
1      TYPHOID FEVER [ENTERIC FEVER] DEATHS      1
2                          DIPHTHERIA DEATHS      2
3 TUBERCULOSIS [PHTHISIS PULMONALIS] DEATHS      5
```

```
4 TUBERCULOSIS [PHTHISIS PULMONALIS] DEATHS      2
5                          DIPHTHERIA DEATHS      1
6 TUBERCULOSIS [PHTHISIS PULMONALIS] DEATHS      4
# ... with 5 more variables: loc <chr>, state <chr>,
#   Year <dbl>, Month <dbl>, Day <dbl>
```

SAS and SPSS

For reading we have the library `haven`. I don't have any example datasets but the functions are below

```
library(haven)
read_sas()
read_spss()
read_stata()
```