Session 2

Nick Hathaway; nicholas.hathaway@umassmed.edu

Contents

Scripting

Like most scientific fields we are always concerned with reproducibility and to that in a programming language like R you accomplish reproducibility by putting all your code into what is called scripts. To create a new R script in RStudio you simply click the `+` sign in the upper left hand corner and click `R Script` or you can use the hotkey shortcut of `Cmd + shift + n`.

Generic Layout of a script

To make your life easier and the life of anyone looking at your scripts easier you normally want to keep them fairly organized. A normal layout is to have all packages or other files you will be using. Next comes any code that deals with reading and tidying up data tables. Then comes the code that does actually analysis followed by any code that writes tables or creates figures.

Commenting

To make your code more readable by other people it's good practice to do what is called commenting of your code. To do this you use the `#` symbol, whenever R sees the symbol '#' it completely ignores everything that comes after it until the next line

```
#this will be ignored
print("hello") # this will also be ignored
```

```
[1] "hello"
```

Path

When someone refers to a `path` they are normally talking about the location of a file or folder on a filesystem. Depending on the operating system (Windows vs Unix based(Mac, Ubuntu, etc.)) this will be represented slightly differently, specifically the use of `"/"`(Unix) vs `"\"` (Windows). Lucklily R takes care of this subtlety for you and you can also use `"/"`. Another piece of terminology that is important is folders are also refered to as directories. The path is represented by naming the parent directories to a file and the path to file/folder can be relative to your current working directory (more on this below). Also giving only `"/"` is considered the very top of your fileystem or the "root" position. An easy way to show this is to use the function `list.files()`

```
print(list.files("/"))
```

```
 [1] "bin"   "boot"  "dev"   "etc"   "home"  "lib"   "lib64" "media" "mnt"   "opt"   "proc"  "root"  "r
[15] "srv"   "sys"   "tmp"   "usr"   "var"
```

Two other important pieces of information is the special way to refer to the current directory (`"."`) and the directory above the current directory (`".."`). Again lets use the list.files.

```
print(list.files(".",full.names = T))
```

```
[1] "./BosEpi.tab.txt"      "./ExampleData.xlsx"    "./Session_2.Rmd"       "./Session_2.html"
[5] "./Session_2.pdf"       "./Temperatures.txt"    "./WorEpi.tab.txt"      "./ends with example.R"
[9] "./example.R"
```

```
print(list.files("..",full.names = T))
```

```
 [1] "../Additional_Resources" "../Cookbook"             "../Examples"             "../Session_1"
 [5] "../Session_2"            "../Session_3"            "../Session_4"            "../create_pages.sh"
 [9] "../datasets"             "../images"               "../index.html"           "../prep"
[13] "../resources"            "../stylesheet.css"
```

```
print(getwd())
```

```
[1] "/var/www/html/bootstrappers/bootstrappers-courses/pastCourses/rCourse_2016-04/Session_2"
```

Another way to explore this idea is to use the `file.choose()` method, which actually just returns the path to whatever file you choose.

```
print(file.choose())
```

```
[1] "/Users/nick/bootstrappers/bootstrappers-courses/pastCourses/rCourse_2016-04/datasets/BosEpi.tab.tx
```

Setting/Getting Working Directory

When working within R you have a working directory, which is where things will be output and this affects how you specify a location's path because it will be relative to this working directory. To get the working directory you use the function `getwd()`.

```
getwd()
```

```
[1] "/var/www/html/bootstrappers/bootstrappers-courses/pastCourses/rCourse_2016-04/Session_2"
```

You can also set your working directory by using the function `setwd()` and giving it the path to a new directory. It might be useful to save your old working directory

```
outWd = getwd()

setwd("/")
list.files(".")
```

```
 [1] "bin"   "boot"  "dev"   "etc"   "home"  "lib"   "lib64" "media" "mnt"   "opt"   "proc"  "root"  "r
[15] "srv"   "sys"   "tmp"   "usr"   "var"
```

```
print(getwd())
```

```
[1] "/"
```

```
setwd(outWd)
list.files(".")
```

```
[1] "BosEpi.tab.txt"      "ExampleData.xlsx"    "Session_2.Rmd"       "Session_2.html"      "Session_2.
[6] "Temperatures.txt"    "WorEpi.tab.txt"      "ends with example.R" "example.R"
```

```
print(getwd())
```

```
[1] "/var/www/html/bootstrappers/bootstrappers-courses/pastCourses/rCourse_2016-04/Session_2"
```

More R Basics

Logic operators and the %in% operator

In session 1 we touched a little upon using logic comparators. They are displayed below again for reference.

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

When using these comparers you should only compare data of the same type/class (number to numbers, characters to characters). Also some of the comparisons only make sense for certain types of data. For example `"A" > "a"` still returns something but it might not actually be what you want.

Comparing Numbers

```
10 > 12
```

```
[1] FALSE
```

```
12 < 14
```

```
[1] TRUE
```

```
10 > 8
```

```
[1] TRUE
```

```
0.43 > 0.1
```

```
[1] TRUE
```

```
0.5 > 0.9
```

```
[1] FALSE
```

```
10 <= 10
```

```
[1] TRUE
```

```
12 >= 10
```

```
[1] TRUE
```

```
10 != 9
```

```
[1] TRUE
```

```
10 != 10
```

```
[1] FALSE
```

```
10 == 10
```

```
[1] TRUE
```

Comparing Letters/characters

Most of the time for character comparisons you will be using `==` and `!=`

```
"a" == "a"
```

```
[1] TRUE
```

```
"This" != "That"
```

```
[1] TRUE
```

```
"This" == "That"
```

```
[1] FALSE
```

Combining multiple statements

You can also combine logic comparisons by using the `&` and `|` symbols, `&` means both statements need to be true to evaluate to true and `|` means either statement needs to be true to have the full statement true

```
10 > 10 & 10 > 9
```

```
[1] FALSE
```

```
10 >= 10 & 10 > 9
```

```
[1] TRUE
```

```
10 > 10 | 10 > 9
```

```
[1] TRUE
```

Comparing vectors of numbers

You can apply logic to multiple numbers at once if they are in a vector

```
#10 random numbers between 0 and 1
rNums = runif(10)
print(rNums)
```

```
 [1] 0.7300011 0.6628528 0.8278244 0.4579930 0.8156761 0.6769959 0.9211756 0.5027044 0.6235274 0.736056
```

```
print(rNums > 0.5)
```

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

If/else statements

Sometimes you will encounter a situation where you have piece of data and you want to run certain code if it falls into one category or different code if it falls into a another category. You can accomplish if you use an `if` statement combined with a logic comparison.

```
x = 10
print("Comparing x to 4")
```

```
[1] "Comparing x to 4"
```

```
if(x > 4){
  print("x is greater than 4")
}
```

```
[1] "x is greater than 4"
```

```
print("Comparing x to 12")
```

```
[1] "Comparing x to 12"
```

```
if(x > 12){
  print("x is greater than 12")
}
```

If the statement in the `if` statement evaluates to true the following code in the brackets that follow the statement will be run and will not be run if the if statements evaluates to false. To run code for when the if statement evaluates to false use the keyword `else`

```
x = 10
print("Comparing x to 12")
```

```
[1] "Comparing x to 12"
```

```
if(x > 12){
  print("x is greater than 12")
}else{
  print ("x is not greater than 12")
}
```

```
[1] "x is not greater than 12"
```

Again you can combine multiple statements in one `if` statement

```
x = 10
if(x > 8 & x < 12){
  print("x is greater than 8 and is less than 12")
}else{
  print ("x is either less than 8 or greater than 12")
}
```

```
[1] "x is greater than 8 and is less than 12"
```

You can also do multiple if statements but using 'else if'

```
name = "Arjan"
if(name == "Nick"){
  print("Name is Nick")
}else if (name == "Mike"){
  print("Name is Mike")
}else{
  print("Name is not Nick or Mike")
}
```

```
[1] "Name is not Nick or Mike"
```

%in% operator

The `%in%` operator looks a little funny but can be very useful for finding subsets of data that only contain certain values

```
names = c("Nick", "Henry", "Jack", "Mike", "Arjan", "Jill", "Jack", "Susan")
programs = c("MD/PhD", "MD/PhD", "MD", "MD/PhD", "PhD", "PhD", "Nursing", "MD/PhD")

print(programs %in% c("MD/PhD"))
```

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

```
print(programs %in% c("MD", "MD/PhD"))
```

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

```
print(programs %in% c("PhD", "MD/PhD"))
```

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

Looping

Sometimes you want to run the same block of code for a list of objects. This is done by using `for` loops. The general syntax for a `for` loop is `for(var in objects){}`. For example instead of writing the following.

```
print(1 ^ 2)
```

```
[1] 1
```

```
print(2 ^ 2)
```

```
[1] 4
```

```
print(3 ^ 2)
```

```
[1] 9
```

```
print(4 ^ 2)
```

```
[1] 16
```

You could write

```
for (num in seq(1,4)){
  print(paste("num is now",num))
  print(num ^ 2)
}
```

```
[1] "num is now 1"
[1] 1
[1] "num is now 2"
[1] 4
[1] "num is now 3"
[1] 9
[1] "num is now 4"
[1] 16
```

Num will become each object from the output of `seq(1,4)` (which is 1,2,3,4) and then the code in the brackets after the `for` statement will be run, num will be set to the next object and the code will be written again.

You could also use for loops to get all the files that end with a certain extension in the current directory

```
allFiles = list.files(".")
print(allFiles)
```

```
[1] "BosEpi.tab.txt"      "ExampleData.xlsx"    "Session_2.Rmd"       "Session_2.html"      "Session_2.
[6] "Temperatures.txt"    "WorEpi.tab.txt"      "ends with example.R" "example.R"
```

```
#this next syntax is how you create your own function
endsWith <- function(var, match) {
  substr(var, pmax(1, nchar(var) - nchar(match) + 1), nchar(var)) == match
}

#get all files that end with tab.txt
tabTxtFiles = c()
for(file in allFiles){
  if(endsWith(file, "tab.txt")){
    tabTxtFiles = c(tabTxtFiles, file)
  }
}
print(tabTxtFiles)
```

```
[1] "BosEpi.tab.txt" "WorEpi.tab.txt"
```

Part 1 Exercises

1. Create a directory for today's session and download the following datasets into it, WorEpi, BosEpi, Temperatures, and ExampleData.

2. Set your working directory to this new directory and list the file in this directory

3. Using `runif()`, generate 20 random numbers and then iterate over them with a for loop and print the numbers that are greater than 0.4 and less than 0.6

4. Using the above example, gather all the files that end with ".csv"
5. Modify the above question to gather all files that end with ".txt"
6. Modify the above to gather all files that end with ".csv" **or** ".txt"
7. If you're feeling adventurous, try creating your very own function that takes the name of a directory and returns all the files that have a certain extension, the function should take two arguments, the name of the directory to search and the extension to look for

Reading in Data recap

readr

There are several useful functions in the library `readr`, chief among them are `read_tsv`, `read_csv`, `read_table` and `read_delim`. All of these functions are used for reading in data in tables from delimited files that are just plain text.

- **`read_tsv`** - Read in a table that the columns are delimited by tab, "`\t`"
- **`read_csv`** - Read in a table that the columns are delimited by commas, "`,`"
- **`read_table`** - Read in a table that the columns are delimited by variable amount of whitespace, which basically tabs and spaces, this can happen if you have output from program that uses a variable number of spaces to make your columns align
- **`read_delim`** - The above three commands assume a delimiter but this function allows you to supply one if you have a case that isn't any of the above delimitations.

readxl

readxl has two functions, one to list the names of the sheets in an excel document (`excel_sheets()`) and the other to read specific sheets `read_excel()`.

List sheets

```
#List excel sheets
require(readxl)

print(excel_sheets("ExampleData.xlsx"))
```

```
[1] "Experiment_1" "Experiment_2"
```

Read in sheets, when just give the filename, it assumes you mean the first sheet

```
#List excel sheets
require(readxl)
require(dplyr)

example = read_excel("ExampleData.xlsx")
print(example)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194
```

```
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786
```

You can name by sheet number

```
#List excel sheets
require(readxl)
require(dplyr)

#by number, 1
sheet1 = read_excel("ExampleData.xlsx", 1)
print(sheet1)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786
```

```
#by number, 2
sheet2 = read_excel("ExampleData.xlsx", 2)
print(sheet2)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.83832510    0.07841589   0.351696500     0.8633275
2    UID2    0.05855855    0.89335270   0.733971300     0.0798722
3    UID3    0.08451808    0.70534490   0.942385000     0.2886966
4    UID4    0.81227860    0.58841960   0.697871600     0.7730438
5    UID5    0.32957630    0.82490760   0.433716000     0.3501705
6    UID6    0.93690810    0.01461093   0.089643530     0.5121381
7    UID7    0.19946200    0.09978814   0.004611515     0.2895618
8    UID8    0.73442320    0.55567470   0.192202100     0.2289813
9    UID9    0.45824800    0.63147390   0.148589200     0.4936379
```

or by name

```
#List excel sheets
require(readxl)
require(dplyr)

print(excel_sheets("ExampleData.xlsx"))
```

```
[1] "Experiment_1" "Experiment_2"
```

```
#by name, "Experiment_1"
sheet1 = read_excel("ExampleData.xlsx", "Experiment_1")
print(sheet1)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786
```

```
#by name, "Experiment_2"
sheet2 = read_excel("ExampleData.xlsx", "Experiment_2")
print(sheet2)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.83832510    0.07841589   0.351696500     0.8633275
2    UID2    0.05855855    0.89335270   0.733971300     0.0798722
3    UID3    0.08451808    0.70534490   0.942385000     0.2886966
4    UID4    0.81227860    0.58841960   0.697871600     0.7730438
5    UID5    0.32957630    0.82490760   0.433716000     0.3501705
6    UID6    0.93690810    0.01461093   0.089643530     0.5121381
7    UID7    0.19946200    0.09978814   0.004611515     0.2895618
8    UID8    0.73442320    0.55567470   0.192202100     0.2289813
9    UID9    0.45824800    0.63147390   0.148589200     0.4936379
```

Combining looping and a new container called list, read in and store all excel sheets

A `list()` in R is able to store a different datatypes and save them under a key, for those of you who are familiar with other languages the list is similar to dictionaries or maps.

```
#List excel sheets
require(readxl)
require(dplyr)

#get names
sheetNames = excel_sheets("ExampleData.xlsx")
print(sheetNames)
```

```
[1] "Experiment_1" "Experiment_2"
```

```
sheets = list()
for(sheetName in sheetNames){
  #here the bracket [] operator takes a name key (sheetName) accepts another object in a list
  sheets[sheetName] = list(read_excel("ExampleData.xlsx", sheetName))
}

print(sheets)
```

```
$Experiment_1
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786

$Experiment_2
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.83832510    0.07841589   0.351696500     0.8633275
2    UID2    0.05855855    0.89335270   0.733971300     0.0798722
3    UID3    0.08451808    0.70534490   0.942385000     0.2886966
4    UID4    0.81227860    0.58841960   0.697871600     0.7730438
5    UID5    0.32957630    0.82490760   0.433716000     0.3501705
6    UID6    0.93690810    0.01461093   0.089643530     0.5121381
7    UID7    0.19946200    0.09978814   0.004611515     0.2895618
8    UID8    0.73442320    0.55567470   0.192202100     0.2289813
9    UID9    0.45824800    0.63147390   0.148589200     0.4936379
```

Accesing elements in a vector

To access only certain elements in a vector you use the `[]` operator. You either give index/position of the elements you want or a logical vector of the same length where all the `TRUE` will be extracted. For the positions R used 1-based positions vs the more command 0-based positions in various programming languages.

```
rNums = runif(20)
print(rNums)
```

```
 [1] 0.99052138 0.99239590 0.10589473 0.95861157 0.76307400 0.05249851 0.39216877 0.10257736 0.62800993
[11] 0.97007565 0.12461411 0.44674208 0.05151349 0.42551797 0.56944408 0.95131924 0.11117317 0.67394634
```

```
#get the first element
print(rNums[1])
```

```
[1] 0.9905214
```

```
#get the first five elements
print(rNums[1:5])
```

```
[1] 0.9905214 0.9923959 0.1058947 0.9586116 0.7630740
```

You can get various different positions by giving a vector of positions

```
#get the first 1st, 3rd, and 7th elements
print(rNums[c(1,3,7)])
```

```
[1] 0.9905214 0.1058947 0.3921688
```

You can also get multiple of the same position

```
#get the first 1st element three times
print(rNums[c(1,1,1)])
```

```
[1] 0.9905214 0.9905214 0.9905214
```

You can get the elements using logic TRUE and FALSE

```
#get the first 1st element three times
print(rNums> 0.5)
```

```
 [1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TR
[20]  TRUE
```

```
print(rNums[rNums > 0.5])
```

```
 [1] 0.9905214 0.9923959 0.9586116 0.7630740 0.6280099 0.9700757 0.5694441 0.9513192 0.6739463 0.792500
```

Accessing elements in a matrix/data.frame

For matrixes and data.frames there are mutliple ways to access certain subsets of the data, specifically rows and columns. To select rows and columns you use the `[]` operator again. You give rows and column number separated by a comma, leaving one blank means all of them

```
sheet1 = read_excel("ExampleData.xlsx", "Experiment_1")
#get the first row, all columns
sheet1[1,]
```

```
# A tibble: 1 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1     0.9594954    0.02578295     0.9898009     0.7006065
```

```r
#get the 1st and 3rd rows, all columns
sheet1[c(1,3),]
```

```
# A tibble: 2 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1     0.9594954    0.02578295     0.9898009     0.7006065
2    UID3     0.4385073    0.40017610     0.0650838     0.9467645
```

```r
#get the first column, all rows
sheet1[,1]
```

```
# A tibble: 9 x 1
  Patient
    <chr>
1    UID1
2    UID2
3    UID3
4    UID4
5    UID5
6    UID6
7    UID7
8    UID8
9    UID9
```

```r
#get the 1-3 columns, all rows
sheet1[,1:3]
```

```
# A tibble: 9 x 3
  Patient Group1-Group1 Group1-Group2
    <chr>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295
2    UID2    0.82969820    0.83762060
3    UID3    0.43850730    0.40017610
4    UID4    0.83518590    0.09643279
5    UID5    0.68717220    0.28007700
6    UID6    0.91114840    0.25461640
7    UID7    0.50003130    0.36041870
8    UID8    0.04682585    0.83349350
9    UID9    0.56947660    0.19843540
```

```r
#get the 1-3 columns, 1-3 rows
sheet1[1:3,1:3]
```

```
# A tibble: 3 x 3
  Patient Group1-Group1 Group1-Group2
    <chr>         <dbl>         <dbl>
1    UID1     0.9594954    0.02578295
2    UID2     0.8296982    0.83762060
3    UID3     0.4385073    0.40017610
```

Also the default is to assume you mean columns, so if you leave out the comma you will get those columns.

```
#get the 1-3 columns, all rows
sheet1[,1:3]
```

```
# A tibble: 9 x 3
  Patient Group1-Group1 Group1-Group2
    <chr>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295
2    UID2    0.82969820    0.83762060
3    UID3    0.43850730    0.40017610
4    UID4    0.83518590    0.09643279
5    UID5    0.68717220    0.28007700
6    UID6    0.91114840    0.25461640
7    UID7    0.50003130    0.36041870
8    UID8    0.04682585    0.83349350
9    UID9    0.56947660    0.19843540
```

```
#same as above
sheet1[1:3]
```

```
# A tibble: 9 x 3
  Patient Group1-Group1 Group1-Group2
    <chr>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295
2    UID2    0.82969820    0.83762060
3    UID3    0.43850730    0.40017610
4    UID4    0.83518590    0.09643279
5    UID5    0.68717220    0.28007700
6    UID6    0.91114840    0.25461640
7    UID7    0.50003130    0.36041870
8    UID8    0.04682585    0.83349350
9    UID9    0.56947660    0.19843540
```

Accesing elementins specific to data.frame

The above examples work for both matrix class and data.frame object but the next couple of examples only work for data.frames

Accesing by column names using []

With data.frame objects you can give the column name in `[]` to get those columns, you can give one or several

```
sheet1 = read_excel("ExampleData.xlsx", "Experiment_1")

sheet1["Patient"]
```

```
# A tibble: 9 x 1
  Patient
```

```
    <chr>
1    UID1
2    UID2
3    UID3
4    UID4
5    UID5
6    UID6
7    UID7
8    UID8
9    UID9
```

```
sheet1["Group1-Group1"]
```

```
# A tibble: 9 x 1
  Group1-Group1
          <dbl>
1    0.95949540
2    0.82969820
3    0.43850730
4    0.83518590
5    0.68717220
6    0.91114840
7    0.50003130
8    0.04682585
9    0.56947660
```

```
sheet1[c("Patient", "Group1-Group1")]
```

```
# A tibble: 9 x 2
  Patient Group1-Group1
    <chr>         <dbl>
1    UID1    0.95949540
2    UID2    0.82969820
3    UID3    0.43850730
4    UID4    0.83518590
5    UID5    0.68717220
6    UID6    0.91114840
7    UID7    0.50003130
8    UID8    0.04682585
9    UID9    0.56947660
```

Accesing by column names using $

You can also access just one column by using the $ symbol.

```
sheet1 = read_excel("ExampleData.xlsx", "Experiment_1")

sheet1$Patient
```

```
[1] "UID1" "UID2" "UID3" "UID4" "UID5" "UID6" "UID7" "UID8" "UID9"
```

```
sheet1$'Group1-Group1'
```

```
[1] 0.95949540 0.82969820 0.43850730 0.83518590 0.68717220 0.91114840 0.50003130 0.04682585 0.56947660
```

The difference here is that the `$` is going to give just a vector where as `[]` will actually give you back a data.frame

```
sheet1 = read_excel("ExampleData.xlsx", "Experiment_1")

patientMoney = sheet1$Patient
print(class(patientMoney))
```

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

```
patientBracket = sheet1["Patient"]
print(class(patientBracket))
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

Adding columns to data.frame

You can also add a column with either the `[]` or the `$`. You can either give a single value that will be repeated for all the values of the column or you can give a vector of the same size.

```
sheet1 = read_excel("ExampleData.xlsx", "Experiment_1")

sheet1$Experiment = "Experiment_1"
print(sheet1)
```

```
# A tibble: 9 x 6
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2   Experiment
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>        <chr>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065 Experiment_1
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115 Experiment_1
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645 Experiment_1
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185 Experiment_1
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827 Experiment_1
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194 Experiment_1
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689 Experiment_1
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864 Experiment_1
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786 Experiment_1
```

```
sheet2 = read_excel("ExampleData.xlsx", "Experiment_2")
print(rep("Experiment_2", nrow(sheet2)))
```

```
[1] "Experiment_2" "Experiment_2" "Experiment_2" "Experiment_2" "Experiment_2" "Experiment_2" "Experimer
[8] "Experiment_2" "Experiment_2"
```

```
sheet2["Experiment"] = rep("Experiment_2", nrow(sheet2))
print(sheet2)
```

```
# A tibble: 9 x 6
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2   Experiment
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>        <chr>
1    UID1    0.83832510    0.07841589   0.351696500     0.8633275 Experiment_2
2    UID2    0.05855855    0.89335270   0.733971300     0.0798722 Experiment_2
3    UID3    0.08451808    0.70534490   0.942385000     0.2886966 Experiment_2
4    UID4    0.81227860    0.58841960   0.697871600     0.7730438 Experiment_2
5    UID5    0.32957630    0.82490760   0.433716000     0.3501705 Experiment_2
6    UID6    0.93690810    0.01461093   0.089643530     0.5121381 Experiment_2
7    UID7    0.19946200    0.09978814   0.004611515     0.2895618 Experiment_2
8    UID8    0.73442320    0.55567470   0.192202100     0.2289813 Experiment_2
9    UID9    0.45824800    0.63147390   0.148589200     0.4936379 Experiment_2
```

Tidyr

The `tidyr` package is about making your data.frames "tidy". Now what is meant by "tidy"? There are considered two ways to organize data tables. One is refered as "wide" format where each cell is a different observation and you have row and column names to explain what those observations are. The other foramt is called "long" format and this format is that every column is a different variable and each row is a different observation and this "long" format is the format that R is the best at for organizing. `tidyr` is all about switching between the two formats.

gather

`gather()` will take a table in "wide"" format and change it into "long" format. It takes four imporant arguments, 1) the data.frame to work on, 2) the name of a new column that contain the old column names, 3) the name of new column to contain the observation that were spread out in the column table, 4) the column indexs to "gather" together.

```
require(tidyr)
require(dplyr)

sheet1Wide = read_excel("ExampleData.xlsx", "Experiment_1")
print(sheet1Wide)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
    <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786
```

```
sheet1Long = gather(sheet1Wide, "Conditions", "values", 2:ncol(sheet1Wide))
print(sheet1Long)
```

```
# A tibble: 36 x 3
   Patient    Conditions     values
     <chr>         <chr>      <dbl>
1     UID1 Group1-Group1 0.95949540
2     UID2 Group1-Group1 0.82969820
3     UID3 Group1-Group1 0.43850730
4     UID4 Group1-Group1 0.83518590
5     UID5 Group1-Group1 0.68717220
6     UID6 Group1-Group1 0.91114840
7     UID7 Group1-Group1 0.50003130
8     UID8 Group1-Group1 0.04682585
9     UID9 Group1-Group1 0.56947660
10    UID1 Group1-Group2 0.02578295
# ... with 26 more rows
```

spread

`spread()` will take a table in "long" format and change it into "wide" format, it basically just does the opposite of what `gather()`. It takes three arguments, 1) the data.frame to operate on, 2) the column to now use for column names, 3) the column that contains all the observation values.

```
print(sheet1Long)
```

```
# A tibble: 36 x 3
   Patient    Conditions     values
     <chr>         <chr>      <dbl>
1     UID1 Group1-Group1 0.95949540
2     UID2 Group1-Group1 0.82969820
3     UID3 Group1-Group1 0.43850730
4     UID4 Group1-Group1 0.83518590
5     UID5 Group1-Group1 0.68717220
6     UID6 Group1-Group1 0.91114840
7     UID7 Group1-Group1 0.50003130
8     UID8 Group1-Group1 0.04682585
9     UID9 Group1-Group1 0.56947660
10    UID1 Group1-Group2 0.02578295
# ... with 26 more rows
```

```
sheet1WideAgain = spread(sheet1Long, "Conditions", "values")
print(sheet1WideAgain)
```

```
# A tibble: 9 x 5
  Patient Group1-Group1 Group1-Group2 Group2-Group1 Group2-Group2
*   <chr>         <dbl>         <dbl>         <dbl>         <dbl>
1    UID1    0.95949540    0.02578295     0.9898009     0.7006065
2    UID2    0.82969820    0.83762060     0.1888582     0.4694115
3    UID3    0.43850730    0.40017610     0.0650838     0.9467645
4    UID4    0.83518590    0.09643279     0.4300042     0.4458185
```

```
5    UID5    0.68717220    0.28007700     0.3018975     0.7624827
6    UID6    0.91114840    0.25461640     0.4620466     0.8967194
7    UID7    0.50003130    0.36041870     0.8670701     0.3029689
8    UID8    0.04682585    0.83349350     0.3874995     0.9897864
9    UID9    0.56947660    0.19843540     0.2006249     0.4643786
```

So you can see we now have the original data.frame ##separate Sometimes you have multiple variables in one columns, especially if you used `gather()` and you need to split this column into two separate columns, for this you can use `separate`. In order for this to work you need to have separated out your variables by somesort of separator.

```
print(sheet1Long)
```

```
# A tibble: 36 x 3
   Patient    Conditions     values
     <chr>         <chr>      <dbl>
1     UID1 Group1-Group1 0.95949540
2     UID2 Group1-Group1 0.82969820
3     UID3 Group1-Group1 0.43850730
4     UID4 Group1-Group1 0.83518590
5     UID5 Group1-Group1 0.68717220
6     UID6 Group1-Group1 0.91114840
7     UID7 Group1-Group1 0.50003130
8     UID8 Group1-Group1 0.04682585
9     UID9 Group1-Group1 0.56947660
10    UID1 Group1-Group2 0.02578295
# ... with 26 more rows
```

```
#give 1) the data.frame, 2) the column to
sheet1Long = separate(sheet1Long, Conditions, c("Condition1", "Condition2"), sep = "-")
print(sheet1Long)
```

```
# A tibble: 36 x 4
   Patient Condition1 Condition2     values
*    <chr>      <chr>      <chr>      <dbl>
1     UID1     Group1     Group1 0.95949540
2     UID2     Group1     Group1 0.82969820
3     UID3     Group1     Group1 0.43850730
4     UID4     Group1     Group1 0.83518590
5     UID5     Group1     Group1 0.68717220
6     UID6     Group1     Group1 0.91114840
7     UID7     Group1     Group1 0.50003130
8     UID8     Group1     Group1 0.04682585
9     UID9     Group1     Group1 0.56947660
10    UID1     Group1     Group2 0.02578295
# ... with 26 more rows
```

unite

`unite()` is the opposite of `separate()` function.

```r
sheet1Long = gather(sheet1Wide, "Conditions", "values", 2:ncol(sheet1Wide))
sheet1Long = separate(sheet1Long, Conditions, c("Condition1", "Condition2"), sep = "-")
print(sheet1Long)
```

```
# A tibble: 36 x 4
   Patient Condition1 Condition2     values
*    <chr>      <chr>      <chr>      <dbl>
1     UID1     Group1     Group1 0.95949540
2     UID2     Group1     Group1 0.82969820
3     UID3     Group1     Group1 0.43850730
4     UID4     Group1     Group1 0.83518590
5     UID5     Group1     Group1 0.68717220
6     UID6     Group1     Group1 0.91114840
7     UID7     Group1     Group1 0.50003130
8     UID8     Group1     Group1 0.04682585
9     UID9     Group1     Group1 0.56947660
10    UID1     Group1     Group2 0.02578295
# ... with 26 more rows
```

```r
sheet1Long = unite(sheet1Long, "Conditions", Condition1, Condition2, sep = "-")
print(sheet1Long)
```

```
# A tibble: 36 x 3
   Patient    Conditions     values
*    <chr>         <chr>      <dbl>
1     UID1 Group1-Group1 0.95949540
2     UID2 Group1-Group1 0.82969820
3     UID3 Group1-Group1 0.43850730
4     UID4 Group1-Group1 0.83518590
5     UID5 Group1-Group1 0.68717220
6     UID6 Group1-Group1 0.91114840
7     UID7 Group1-Group1 0.50003130
8     UID8 Group1-Group1 0.04682585
9     UID9 Group1-Group1 0.56947660
10    UID1 Group1-Group2 0.02578295
# ... with 26 more rows
```

Part2 Excercises

1. Read in the BosEpi(tab delimited) dataset and unite the Year, Month, Day columns into one column called Date, that has the format of %Y-%M-%D

2. Read in the WorEpi(tab delimited) dataset and unite the Year, Month, Day columns into one column called Date, that has the format of %Y-%M-%D

3. Read in both the Experiment_1 and Experiment_2 sheets of the ExampleData.xlsx file and convert to long format. For Experiment_1 and Experiment_2 long format tables add a column to both named "Experiment" that contains the proper experiment name for which experiment they came.

4. Take what you did for Experiment_1 above and then separate the column "Conditions" column into two separate columns

Dplyr

dplyr is all about manipulating datasets and getting stats on the them.

select

This is similar to above when we were selecting columns but dplyr can add some functionality to what we were doing above.

```
require(tidyr)
require(dplyr)
require(readr)

bosEpi = read_tsv("BosEpi.tab.txt")

# select column names
select(bosEpi, disease, event, number)
```

```
# A tibble: 30,062 x 3
                         disease  event number
                           <chr>  <chr>  <int>
1  TYPHOID FEVER [ENTERIC FEVER] DEATHS      4
2                     DIPHTHERIA DEATHS      7
3     WHOOPING COUGH [PERTUSSIS] DEATHS      2
4  TYPHOID FEVER [ENTERIC FEVER] DEATHS      2
5                     DIPHTHERIA DEATHS      5
6     WHOOPING COUGH [PERTUSSIS] DEATHS      2
7  TYPHOID FEVER [ENTERIC FEVER] DEATHS      5
8                     DIPHTHERIA DEATHS      8
9  TYPHOID FEVER [ENTERIC FEVER] DEATHS      2
10                 SCARLET FEVER DEATHS      1
# ... with 30,052 more rows
```

But rather than using specific function, dplyr supplies several useful functions that you take advantage of, like `ends_with()`

```
require(tidyr)
require(dplyr)
require(readr)

bosEpi = read_tsv("BosEpi.tab.txt")

# select column names
select(bosEpi, ends_with("e"))
```

```
# A tibble: 30,062 x 2
                         disease state
                           <chr> <chr>
1  TYPHOID FEVER [ENTERIC FEVER]    MA
2                     DIPHTHERIA    MA
3     WHOOPING COUGH [PERTUSSIS]    MA
```

```
4  TYPHOID FEVER [ENTERIC FEVER]    MA
5                      DIPHTHERIA    MA
6     WHOOPING COUGH [PERTUSSIS]    MA
7  TYPHOID FEVER [ENTERIC FEVER]    MA
8                      DIPHTHERIA    MA
9  TYPHOID FEVER [ENTERIC FEVER]    MA
10                  SCARLET FEVER    MA
# ... with 30,052 more rows
```

function	meaning
contains()	Select columns whose name contains a character string
ends_with()	Select columns whose name ends with a string
everything()	Select every column
matches()	Select columns whose name matches a regular expression
num_range()	Select columns named x1, x2, x3, x4, x5
one_of()	Select columns whose names are in a group of names
starts_with()	Select columns whose name starts with a character string

filter

You can also use dplyr `filter()` to select only certain rows of under specific conditions

```
require(tidyr)
require(dplyr)
require(readr)

bosEpi = read_tsv("BosEpi.tab.txt")

#get data from only specific years, like the 1950-1999

filter(bosEpi,  Year >= 1950 & Year < 2000)
```

```
# A tibble: 1,961 x 8
                     disease event number    loc state  Year Month   Day
                       <chr> <chr>  <int>  <chr> <chr> <int> <int> <int>
1                    MEASLES CASES    220 BOSTON    MA  1951    12    30
2                 TRICHINIASIS CASES      1 BOSTON    MA  1951    12    30
3  WHOOPING COUGH [PERTUSSIS] CASES     37 BOSTON    MA  1951    12    30
4                    MEASLES CASES    325 BOSTON    MA  1952     1     6
5  WHOOPING COUGH [PERTUSSIS] CASES     17 BOSTON    MA  1952     1     6
6                    MEASLES CASES    226 BOSTON    MA  1952     1    13
7  WHOOPING COUGH [PERTUSSIS] CASES     81 BOSTON    MA  1952     1    13
8                  DIPHTHERIA CASES      1 BOSTON    MA  1952     1    20
9                    MEASLES CASES    232 BOSTON    MA  1952     1    20
10 WHOOPING COUGH [PERTUSSIS] CASES     31 BOSTON    MA  1952     1    20
# ... with 1,951 more rows
```

```
#get only certain diseases
filter(bosEpi,  disease == "DIPHTHERIA")
```

```
# A tibble: 3,866 x 8
      disease  event number    loc state  Year Month   Day
        <chr>  <chr>  <int>  <chr> <chr> <int> <int> <int>
1  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     7    22
2  DIPHTHERIA DEATHS      5 BOSTON    MA  1888     7    29
3  DIPHTHERIA DEATHS      8 BOSTON    MA  1888     8     5
4  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     8    12
5  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     8    19
6  DIPHTHERIA DEATHS      4 BOSTON    MA  1888     8    26
7  DIPHTHERIA DEATHS      5 BOSTON    MA  1888     9     2
8  DIPHTHERIA DEATHS      5 BOSTON    MA  1888     9     9
9  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     9    16
10 DIPHTHERIA DEATHS     11 BOSTON    MA  1888     9    23
# ... with 3,856 more rows
```

```
#get only certain diseases
filter(bosEpi,  disease %in% c("DIPHTHERIA", "WHOOPING COUGH [PERTUSSIS]"))
```

```
# A tibble: 6,038 x 8
                     disease  event number    loc state  Year Month   Day
                       <chr>  <chr>  <int>  <chr> <chr> <int> <int> <int>
1                  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     7    22
2  WHOOPING COUGH [PERTUSSIS] DEATHS      2 BOSTON    MA  1888     7    22
3                  DIPHTHERIA DEATHS      5 BOSTON    MA  1888     7    29
4  WHOOPING COUGH [PERTUSSIS] DEATHS      2 BOSTON    MA  1888     7    29
5                  DIPHTHERIA DEATHS      8 BOSTON    MA  1888     8     5
6                  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     8    12
7  WHOOPING COUGH [PERTUSSIS] DEATHS      3 BOSTON    MA  1888     8    12
8                  DIPHTHERIA DEATHS      7 BOSTON    MA  1888     8    19
9  WHOOPING COUGH [PERTUSSIS] DEATHS      1 BOSTON    MA  1888     8    19
10                 DIPHTHERIA DEATHS      4 BOSTON    MA  1888     8    26
# ... with 6,028 more rows
```

summarize and group_by

```
require(tidyr)
require(dplyr)
require(readr)

bosEpi = read_tsv("BosEpi.tab.txt")
bosEpi = select(bosEpi, disease, event, number, Year)
bosEpi = group_by(bosEpi,event, disease, Year )

print(summarise(bosEpi,number = sum(number)))
```

```
Source: local data frame [746 x 4]
Groups: event, disease [?]

   event                       disease  Year number
   <chr>                         <chr> <int>  <int>
1  CASES BRUCELLOSIS [UNDULANT FEVER]  1952      2
```

```
2  CASES       CHICKENPOX [VARICELLA]  1923     97
3  CASES       CHICKENPOX [VARICELLA]  1924   1810
4  CASES       CHICKENPOX [VARICELLA]  1925   1129
5  CASES       CHICKENPOX [VARICELLA]  1926   2145
6  CASES       CHICKENPOX [VARICELLA]  1927   2777
7  CASES       CHICKENPOX [VARICELLA]  1928   2112
8  CASES       CHICKENPOX [VARICELLA]  1929   2265
9  CASES       CHICKENPOX [VARICELLA]  1930   2009
10 CASES       CHICKENPOX [VARICELLA]  1931   2242
# ... with 736 more rows
```

```
print(summarise(bosEpi,number = median(number)))
```

```
Source: local data frame [746 x 4]
Groups: event, disease [?]

   event                      disease  Year number
   <chr>                        <chr> <int>  <dbl>
1  CASES BRUCELLOSIS [UNDULANT FEVER]  1952    1.0
2  CASES       CHICKENPOX [VARICELLA]  1923   97.0
3  CASES       CHICKENPOX [VARICELLA]  1924   30.0
4  CASES       CHICKENPOX [VARICELLA]  1925   27.5
5  CASES       CHICKENPOX [VARICELLA]  1926   27.5
6  CASES       CHICKENPOX [VARICELLA]  1927   56.0
7  CASES       CHICKENPOX [VARICELLA]  1928   34.5
8  CASES       CHICKENPOX [VARICELLA]  1929   47.0
9  CASES       CHICKENPOX [VARICELLA]  1930   46.0
10 CASES       CHICKENPOX [VARICELLA]  1931   49.0
# ... with 736 more rows
```

```
bosEpi = group_by(bosEpi, event, disease)
print(summarise(bosEpi,number = median(number)))
```

```
Source: local data frame [39 x 3]
Groups: event [?]

   event                      disease number
   <chr>                        <chr>  <dbl>
1  CASES BRUCELLOSIS [UNDULANT FEVER]    1.0
2  CASES       CHICKENPOX [VARICELLA]   40.0
3  CASES                     DENGUE    0.0
4  CASES                  DIPHTHERIA   20.0
5  CASES                   INFLUENZA    3.0
6  CASES                     LEPROSY    0.0
7  CASES                     MALARIA    8.5
8  CASES                     MEASLES   45.0
9  CASES                  MENINGITIS    1.0
10 CASES                       MUMPS   12.0
# ... with 29 more rows
```

Useful function for the summarize function

function	meaning
min(), max()	Minimum and maximum values
mean()	Mean value
median()	Median value
sum()	Sum of values
var, sd()	Variance and standard deviation of a vector

bind_rows

Combine two data.frames into one data.frame

```
require(tidyr)
require(dplyr)
require(readr)

bosEpi = read_tsv("BosEpi.tab.txt")
worEpi = read_tsv("WorEpi.tab.txt")

allEpi = bind_rows(bosEpi, worEpi)
```

Part3 Excercises

1. Read in both the Experiment_1 and Experiment_2 sheets of the ExampleData.xlsx file and convert to long format. For Experiment_1 and Experiment_2 long format tables add a column to both named "Experiment" that contains the proper experiment name for which experiment they came.

2. Take what you did for Experiment_1 above and then separate the column "Conditions" column into two separate columns

3. Combine the two data.frames into one data.frame

4. group by condition 1 and by Experiment and get the mean, median, and sd