Session 2

Nick Hathaway; nicholas.hathaway@umassmed.edu

Contents

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

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.

```
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
```

```
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
```

```
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 a very commonly used object in R. 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 a different type, 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 read in a data.frame by using functions that read them in, but below are some examples of how you can create a data.frame by hand.

```
dat = data.frame(names = c("Nick", "Jake", "Mercedeh", "Jack", "Michael"), duration = c(7, 3, 3, 2, 7),
print(dat)
```

```
     names duration program
1     Nick        7  MD/PhD
2     Jake        3     PhD
3 Mercedeh        3     PhD
4     Jack        2     PhD
5  Michael        7  MD/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", "Jake", "Mercedeh", "Jack", "Michael"), duration = c(7, 3, 3, 2, 7),
str(dat)
```

```
'data.frame':   5 obs. of  3 variables:
 $ names   : Factor w/ 5 levels "Jack","Jake",..: 5 2 3 1 4
 $ duration: num  7 3 3 2 7
 $ program : Factor w/ 2 levels "MD/PhD","PhD": 1 2 2 2 1
```

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", "Jake", "Mercedeh", "Jack", "Michael"), duration = c(7, 3, 3, 2, 7),
str(dat)
```

```
'data.frame':   5 obs. of  3 variables:
 $ names   : Factor w/ 5 levels "Jack","Jake",..: 5 2 3 1 4
 $ duration: num  7 3 3 2 7
 $ program : Factor w/ 3 levels "MD/PHD","MD/PhD",..: 2 3 3 3 1
```

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.

You can also use the function called `View()` to see the data in a spreadsheet like Viewer within RStudio.

```
dat = data.frame(names = c("Nick", "Jake", "Mercedeh", "Jack", "Michael"), duration = c(7, 3, 3, 2, 7),
View(dat)
```

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). Luckily R takes care of this subtlety for you and you can also use `"/"`. Another piece of terminology that is important is folders are also referred 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 filesystem 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"
 [8] "media" "mnt"   "opt"   "proc"  "root"  "run"   "sbin"
[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] "./Both.xlsx"
 [2] "./ExampleData.xlsx"
 [3] "./Mel.csv"
 [4] "./Session_2.Rmd"
 [5] "./Session_2.html"
 [6] "./Session_2.pdf"
 [7] "./Session_2.tex"
 [8] "./Temperatures.txt"
 [9] "./WorEpi.tab.txt"
[10] "./avg_temps_usa_wide.tab.txt"
[11] "./images_workzone"
[12] "./time.series.data.txt"
```

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

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

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

```
[1] "/var/www/html/bootstrappers/bootstrappers-courses/pastCourses/rCourse_2018-04/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/ownCloud/documents/exp_design_and_data_analysis_bootcamp_2016/introToR/time.series.data
```

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_2018-04/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"
 [8] "media" "mnt"   "opt"   "proc"  "root"  "run"   "sbin"
[15] "srv"   "sys"   "tmp"   "usr"   "var"
```

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

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

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

```
 [1] "Both.xlsx"
 [2] "ExampleData.xlsx"
 [3] "Mel.csv"
 [4] "Session_2.Rmd"
 [5] "Session_2.html"
 [6] "Session_2.pdf"
 [7] "Session_2.tex"
 [8] "Temperatures.txt"
 [9] "WorEpi.tab.txt"
[10] "avg_temps_usa_wide.tab.txt"
[11] "images_workzone"
[12] "time.series.data.txt"
```

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

```
[1] "/var/www/html/bootstrappers/bootstrappers-courses/pastCourses/rCourse_2018-04/Session_2"
```

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 are some of the libraries that I use the most in R. The tidyverse library is a collection of libraries that help to read in data, manipulate data, and plot data.

```
#For reading in data, organizing data, and plotting
install.packages(c("tidyverse"))
```

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

```
library(tidyverse)
```

Part 1 Excercises

Here are the links for the data files mentioned below.

ExampleData.xlsx
Both.xlsx
Temperatures.txt
Mel.csv
WorEpi.tab.txt

1. Create a folder on your computer to store some files we will be reading into R (do this without using R or if you want to challenge yourself, Google how to create a folder with R and try it out!).
2. Now download the following files, ExampleData.xlsx, Both.xlsx, Temperatures.txt, Mel.csv, WorEpi.tab.txt and put them in this folder (again do this without R or again challenge yourself and Google or use the `help` function to how to use the R function `download.file` to do this)
3. Set your working directory to this folder and list the files in the folder
4. Install the package `tidyverse` and load it in R.

Reading in Data

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

File Formats

Plain Text Files

For reading in plain text files there is a great package called `readr` which is part of `tidyverse`

```
library(readr)
```

For reading in tab delimited files we have `read_tsv` (tsv = tab seperated values) and for comma delimited files we have `read_csv` (csv = comma separated values).
Also a helpful function might be to use `file.choose()`
Once a library is installed (vis **install.packages()**) you can just call its functions by using the package name followed by two : and the function name e.g. readr::read_tsv

```
#
worEpi = readr::read_tsv("WorEpi.tab.txt")
```

```
library(readr) #this loads all the readr package functions so that they don't have to be called with th
worEpi = read_tsv("WorEpi.tab.txt") # just read_tsv and don't need readr::read_tsv once library(readr)
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

```
worEpi = readr::read_tsv(file.choose())
str(worEpi)
worEpi
```

Here is a link to a comma separated file

```
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     0.650     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    12.1       1
6   204      1     1    28  1971     4.84      1
```

or

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

Reading from URL

R can also read files directly from url links

```
melanoma = read_csv("http://bootstrappers.umassmed.edu/bootstrappers-courses/courses/rCourse/Session_1/
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     0.650     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    12.1       1
6   204      1     1    28  1971     4.84      1
```

Sometimes your data will have row names as well, which the functions of readr will read in and put as the first column without a name

```
temps = read_tsv("Temperatures.txt")
print(temps)
```

```
# A tibble: 12 x 19
   X1        Boston `New York` Miami `Los Angeles` Aspen
   <chr>      <int>      <int> <int>         <int> <int>
 1 January       36         36    74            68    36
 2 February      39         40    75            69    39
 3 March         45         48    76            70    46
 4 April         56         58    79            73    53
 5 May           66         68    83            74    63
 6 June          76         77    87            78    73
 7 July          82         83    88            83    79
 8 August        80         81    89            84    76
 9 September     72         74    87            83    69
10 October       61         63    84            79    58
11 November      52         52    79            73    44
12 December      41         42    76            68    35
# ... with 13 more variables: Anchorage <int>, `New
#   Orleans` <int>, `San Diego` <int>, `San
#   Francisco` <int>, Austin <int>, Phoenix <int>,
#   Chicago <int>, Honolulu <int>, Atlanta <int>,
#   `Washington D.C.` <int>, Seattle <int>, `St.
#   Louis` <int>, Minneapolis <int>
```

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")
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.     0.650    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.    12.1      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)
head(worEpi)
```

```
# A tibble: 6 x 8
  disease       event  number loc   state  Year Month   Day
  <chr>         <chr>   <dbl> <chr> <chr> <dbl> <dbl> <dbl>
1 TYPHOID FEVE~ DEATHS     1. WORC~ MA    1894.    6.    1.
2 DIPHTHERIA    DEATHS     2. WORC~ MA    1894.    6.    1.
3 TUBERCULOSIS~ DEATHS     5. WORC~ MA    1894.    6.    8.
4 TUBERCULOSIS~ DEATHS     2. WORC~ MA    1894.    6.   15.
5 DIPHTHERIA    DEATHS     1. WORC~ MA    1894.    6.   15.
6 TUBERCULOSIS~ DEATHS     4. WORC~ MA    1894.    6.   29.
```

You can also read in by giving the sheet's name, to get what the sheets names are you can use

```
library(readxl)
print(excel_sheets("Both.xlsx"))
```

```
[1] "Mel"    "Sheet2"
```

```
worEpi = read_excel("Both.xlsx", sheet = "Sheet2")
head(worEpi)
```

```
# A tibble: 6 x 8
  disease       event  number loc   state  Year Month   Day
  <chr>         <chr>   <dbl> <chr> <chr> <dbl> <dbl> <dbl>
1 TYPHOID FEVE~ DEATHS     1. WORC~ MA    1894.    6.    1.
2 DIPHTHERIA    DEATHS     2. WORC~ MA    1894.    6.    1.
3 TUBERCULOSIS~ DEATHS     5. WORC~ MA    1894.    6.    8.
4 TUBERCULOSIS~ DEATHS     2. WORC~ MA    1894.    6.   15.
5 DIPHTHERIA    DEATHS     1. WORC~ MA    1894.    6.   15.
6 TUBERCULOSIS~ DEATHS     4. WORC~ MA    1894.    6.   29.
```

```
melanoma = read_excel("Both.xlsx", sheet = "Mel")
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.     0.650    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.    12.1      1.
6  204.     1.    1.   28. 1971.     4.84     1.
```

SAS and SPSS

For reading SAS and SPSS files 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()
```

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] 0.54051398 0.48319366 0.65729230 0.05174603 0.03607009
 [6] 0.67090626 0.48483815 0.65992517 0.92813687 0.02722826
[11] 0.04104496 0.45996039 0.36909958 0.05957051 0.40416004
[16] 0.93916787 0.97189602 0.92623347 0.09430825 0.81746693
```

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

```
[1] 0.540514
```

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

```
[1] 0.54051398 0.48319366 0.65729230 0.05174603 0.03607009
```

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] 0.5405140 0.6572923 0.4848382
```

You can also get multiple of the same position

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

```
[1] 0.540514 0.540514 0.540514
```

You can get the elements using logic TRUE and FALSE

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

```
 [1]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE
[10] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
[19] FALSE  TRUE
```

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

```
[1] 0.5405140 0.6572923 0.6709063 0.6599252 0.9281369
[6] 0.9391679 0.9718960 0.9262335 0.8174669
```

Accessing elements in a matrix/data.frame

For matrixes and data.frames there are multiple 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`
  <chr>             <dbl>           <dbl>           <dbl>
1 UID1              0.959          0.0258           0.990
# ... with 1 more variable: `Group2-Group2` <dbl>
```

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

```
# A tibble: 2 x 5
  Patient `Group1-Group1` `Group1-Group2` `Group2-Group1`
  <chr>             <dbl>           <dbl>           <dbl>
1 UID1              0.959          0.0258          0.990
2 UID3              0.439          0.400           0.0651
# ... with 1 more variable: `Group2-Group2` <dbl>
```

```
#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
```

```
#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              0.959          0.0258
2 UID2              0.830          0.838
3 UID3              0.439          0.400
4 UID4              0.835          0.0964
5 UID5              0.687          0.280
6 UID6              0.911          0.255
7 UID7              0.500          0.360
8 UID8              0.0468         0.833
9 UID9              0.569          0.198
```

```
#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              0.959          0.0258
2 UID2              0.830          0.838
3 UID3              0.439          0.400
```

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              0.959          0.0258
```

```
2 UID2             0.830          0.838
3 UID3             0.439          0.400
4 UID4             0.835          0.0964
5 UID5             0.687          0.280
6 UID6             0.911          0.255
7 UID7             0.500          0.360
8 UID8             0.0468         0.833
9 UID9             0.569          0.198
```

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

```
# A tibble: 9 x 3
  Patient `Group1-Group1` `Group1-Group2`
  <chr>             <dbl>           <dbl>
1 UID1              0.959           0.0258
2 UID2              0.830           0.838
3 UID3              0.439           0.400
4 UID4              0.835           0.0964
5 UID5              0.687           0.280
6 UID6              0.911           0.255
7 UID7              0.500           0.360
8 UID8              0.0468          0.833
9 UID9              0.569           0.198
```

Also helpful functions here are `ncol()` and `nrow()` which gives you the number of columns and rows for a data.frame/matrix. This can be used if you want certain columns starting at a position and then until the end of the dataframe.

```
#get the 2nd column to the end of the columns, all rows
sheet1[,2:ncol(sheet1)]
```

```
# A tibble: 9 x 4
  `Group1-Group1` `Group1-Group2` `Group2-Group1`
            <dbl>           <dbl>           <dbl>
1          0.959           0.0258          0.990
2          0.830           0.838           0.189
3          0.439           0.400           0.0651
4          0.835           0.0964          0.430
5          0.687           0.280           0.302
6          0.911           0.255           0.462
7          0.500           0.360           0.867
8          0.0468          0.833           0.387
9          0.569           0.198           0.201
# ... with 1 more variable: `Group2-Group2` <dbl>
```

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           0.959
2           0.830
3           0.439
4           0.835
5           0.687
6           0.911
7           0.500
8           0.0468
9           0.569
```

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

```
# A tibble: 9 x 2
  Patient `Group1-Group1`
  <chr>             <dbl>
1 UID1              0.959
2 UID2              0.830
3 UID3              0.439
4 UID4              0.835
5 UID5              0.687
6 UID6              0.911
7 UID7              0.500
8 UID8              0.0468
9 UID9              0.569
```

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"
[9] "UID9"
```

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

```
[1] 0.95949540 0.82969820 0.43850730 0.83518590 0.68717220
[6] 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`
  <chr>             <dbl>           <dbl>           <dbl>
1 UID1              0.959           0.0258          0.990
2 UID2              0.830           0.838           0.189
3 UID3              0.439           0.400           0.0651
4 UID4              0.835           0.0964          0.430
5 UID5              0.687           0.280           0.302
```

```
6 UID6             0.911           0.255           0.462
7 UID7             0.500           0.360           0.867
8 UID8             0.0468          0.833           0.387
9 UID9             0.569           0.198           0.201
# ... with 2 more variables: `Group2-Group2` <dbl>,
#   Experiment <chr>
```

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

```
[1] "Experiment_2" "Experiment_2" "Experiment_2"
[4] "Experiment_2" "Experiment_2" "Experiment_2"
[7] "Experiment_2" "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`
  <chr>             <dbl>           <dbl>           <dbl>
1 UID1              0.838           0.0784          0.352
2 UID2              0.0586          0.893           0.734
3 UID3              0.0845          0.705           0.942
4 UID4              0.812           0.588           0.698
5 UID5              0.330           0.825           0.434
6 UID6              0.937           0.0146          0.0896
7 UID7              0.199           0.0998          0.00461
8 UID8              0.734           0.556           0.192
9 UID9              0.458           0.631           0.149
# ... with 2 more variables: `Group2-Group2` <dbl>,
#   Experiment <chr>
```

Part 2 Excercises

time.series.data.txt

1. Read in the temperature dataset from above and find the max, min and mean temperature of your favorite month (columns) and favorite city (rows).
2. Create a folder and download the following file, time.series.data.txt, link above, which is a dataset of expression of genes in different cells in several exposure conditions. The rownames are the gene names, the file is tab (`\t`) separated file.
3. Read in time.series.data.txt
4. Find the `max`, `min`, and `max` of the control column and one of the LPS exposure columns (the columns that start with LPS).
5. Find the `max`, `min`, and `max` of the several of your favorite genes (or just pick one) (hint: be careful of selection)