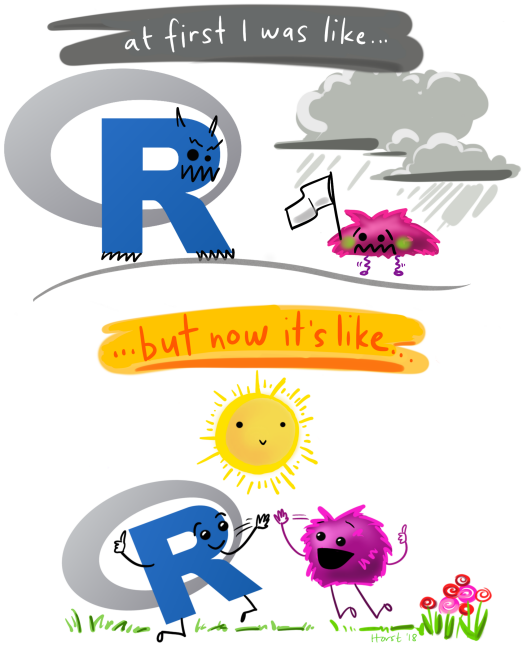
Getting Started in R

1. What is R?

R is a high level language designed for statisticians and data scientists. It allows for flexible and customizable data analysis as well as easy replicability. R is open-source and the R community is known for its active contributions in developing new packages/functions. The wide-range of functions available in R can implement a vast range of statistical learning techniques.

R is an interpreted language. Very loosely speaking this means that when you submit code in R, it is executed in real time. It processes and runs the code you submit line-by-line. This is in contrast to a compiled language (such as C, C++, or Java) whose code is first converted to machine-code by a compiler and then executed.



2. Installing R

1. Go to https://mirror.las.iastate.edu/CRAN/
2. Click the download link for your operating system (Windows, Mac, or Linux).
3. **For Windows**: Click on the base link, and download the R installer. Follow the instructions as prompted.
4. **For Mac**: Download the newest version of R (or an older version if applicable). Follow the on-screen instructions to complete install. Make sure your Mac software is current and recent updates/patches

have been installed. You may need to install XQuartz (https://www.xquartz.org/) if you don't have it already.

3. Basic Commands

R uses functions to perform operations. One of the most common functions you will use is the *concatenate* function: `c()`. Any numbers inside the parentheses are jointed together.

The following command instructs `R` to join together the numbers 1,3,2, and 5 and to save them as a *vector* named `x`.

```
x = c(1,3,2,5)
x
```

```
[1] 1 3 2 5
```

Hitting the *up* arrow multiple times will display the previous commands , which can then be edited. This is useful since one may often wish to repeat a similar command.

We can use `R` to carry out basic arithmetic. For example:

```
# define variables x and y
x = c(4,6,8)
y = c(2,2,1)
```

In order to perform arithmetic operations, we need to make sure our vectors are the same length. We can do that using the `length()` function:

```
length(x)
```

```
[1] 3
```

```
length(y)
```

```
[1] 3
```

```
#perform arithmetic on vectors, notice that the operations are element-wise
x+y
```

```
[1] 6 8 9
```

```
x*y
```

```
[1]  8 12  8
```

```
x/y
```

```
[1] 2 3 8
```

There are many built in `R` functions.

```
log(x)
```

```
[1] 1.386294 1.791759 2.079442
```

```
sqrt(x)
```

```
[1] 2.000000 2.449490 2.828427
```

```
max(x)
```

```
[1] 8
```

You can find out more about what a function does by typing in your R console `?function_name`. For example `?max` will give you information on the max function.

The `rnorm()` function generates a vector of random normal variables, with the first argument `n` being the sample size. Type in `?rnorm` to see more information. We will use this function frequently to generate random normal variables when we run simulations. Each time we call this function, we will get different values (since it is randomly generated).

Here we create two correlated sets of numbers, `x` and `y`, and use the `cor()` function to compute the correlation between them.

```
x = rnorm(50)
y = x+ rnorm(50,mean=50,sd=0.1)
cor(x,y)
```

```
[1] 0.9927645
```

By default, `rnorm()` creates standard normal random variables with a mean of 0 and a standard deviation of 1. However, these can be altered using the `mean` and `sd` arguments, as illustrated above.

Sometimes, we want our code to reproduce the exact same set of random variables, especially if we are sharing code that we would like someone else to be able to replicate exactly. When there are random quantities involved, one way to ensure that we all get the same results is to use the `set.seed()` function. The `set.seed()` function takes an (arbitrary) integer argument.

```
set.seed(100)
rnorm(5)
```

```
[1] -0.50219235  0.13153117 -0.07891709  0.88678481  0.11697127
```

If you ran the above code on your own laptop, you would get the exact same numbers as me.

The `mean()` and `var()` functions can be used to compute the mean and variance of a vector of numbers.

```
set.seed(3)
y = rnorm(100)
head(y) # this function will let you peak at the first few values of y
```

```
[1] -0.96193342 -0.29252572  0.25878822 -1.15213189  0.19578283  0.03012394
```

```
mean(y)
```

```
[1] 0.01103557
```

```
var(y)
```

```
[1] 0.7328675
```

```
sqrt(var(y))
```

```
[1] 0.8560768
```

```
sd(y)
```

```
[1] 0.8560768
```

4. Indexing Data

We often wish to examine part of a set of data. Suppose that our data is stored in the matrix named `A`.

```
A = matrix(1:16, 4,4)
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Then typing

```
A[2,3]
```

```
[1] 10
```

will select the element corresponding to the second row and the third column. The first number of `[, ]` always refers to the *row* and the second number always refers to the *column*. We can also select multiple rows and columns at a time, by providing vectors as the indices.

```
A[c(1,3),c(2,4)] #before you run this, ask yourself what should it ouput?
```

```
     [,1] [,2]
[1,]    5   13
[2,]    7   15
```

```
A[1:3,2:4] # how is this different than the above code?
```

```
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
```

```
A[1:2,]
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
```

```
A[,1:2]
```

```
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

The last two examples include either no index for the columns or no index for the rows. These indicated that `R` should include all columns or all rows, respectively. `R` treats a single row or column of a matrix as a vector:

```
A[1,]
```

```
[1]  1  5  9 13
```

The use of a negative sign `-` in the index tells `R` to keep all rows or columns except those indicated in the index:

```
A[-c(1,3),]
```

```
     [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    4    8   12   16
```

```
A[-1,]
```

```
     [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    3    7   11   15
[3,]    4    8   12   16
```

The `dim()` function outputs the number of rows followed by the number of columns of a given matrix.

```
dim(A)
```

```
[1] 4 4
```

5. Loading Data

For most analyses, the first step involves importing a data set into `R`.

Before attempting to load a data set, we must make sure that `R` knows to search for the data in the proper directory. You can check your working directory in `R` by typing in `getwd()`. You can then set your directory using `setwd()`. So for example, if I want to set my working directly to my DS301 folder, I would type:

```
setwd("/Users/lchu/GoogleDrive/DS301")
```

I would then save my data set to this folder. Now `R` knows to go into this folder to load the data set. Let's load in the `Auto.data` data set. You can find a copy of this on Canvas under Module 01.

Note that `Auto.data` is simply a text file, which you could also just open on your computer using a standard text editor. It's often a good idea to view a data set using a text editor or other software (such as Excel) before loading it into `R`. To import our data set into `R`, one of the primary ways to do so is using the `read.table()` function.

```
Auto = read.table("Auto.data", header=TRUE, na.strings="?")
```

In order to read in the data correctly, we need to include some additional options with the `read.table()` function. The option `header=TRUE` tells `R` that the first line of the file contains the variable names, and is not part of the data. The option `na.strings` tells `R` that any time it sees a particular character or set of characters (such as a question mark), it should be treated as a missing element of the data matrix.

This object `Auto` is known as a *data frame*. We will come back to this throughout the semester.

We can remove all the missing values (do this with caution and only after thorough exploratory analysis). In this case, there are only 5 rows with missing observations.

```
dim(Auto)
```

```
[1] 397   9
```

```
Auto[1:4,]
```

```
  mpg cylinders displacement horsepower weight acceleration year origin
1  18         8          307        130   3504         12.0   70      1
2  15         8          350        165   3693         11.5   70      1
3  18         8          318        150   3436         11.0   70      1
4  16         8          304        150   3433         12.0   70      1
                       name
1 chevrolet chevelle malibu
2         buick skylark 320
3        plymouth satellite
4             amc rebel sst
```

```
Auto = na.omit(Auto) # remove rows with missing values
dim(Auto)
```

```
[1] 392   9
```

Once the data are loaded correctly, we can use `names()` to check the variables names.

```
names(Auto)
```

```
[1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
[6] "acceleration" "year"         "origin"       "name"
```

To refer to a variable in a data set, you must tell `R` which data set that variable belongs to. You can do this using the `$`. For example, suppose we want to look at the first 5 observations of the variable `cylinders`. If we type in

```
cylinders[1:5]
```

```
Error in eval(expr, envir, enclos): object 'cylinders' not found
```

we will get an error message.

Instead, we need to type in:

```
Auto$cylinders[1:5]
```

```
[1] 8 8 8 8 8
```

```
summary(Auto$cylinders) #obtain summary statistics on the variable cylinders
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   4.000   5.472   8.000   8.000
```

```
summary(Auto$mpg) #obtain summary statistics on the variable mpg
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.00   17.00   22.75   23.45   29.00   46.60
```

If we want to get a quick overview of the `Auto` object, the `str()` function is very handy.

```
str(Auto)
```

```
'data.frame':   392 obs. of  9 variables:
 $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
 $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
 $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
 $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
 $ weight      : num  3504 3693 3436 3433 3449 ...
 $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
 $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
 $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ name        : chr  "chevrolet chevelle malibu" "buick skylark 320" "plymouth satellite" "amc rebel s
 - attr(*, "na.action")= 'omit' Named int [1:5] 33 127 331 337 355
  ..- attr(*, "names")= chr [1:5] "33" "127" "331" "337" ...
```