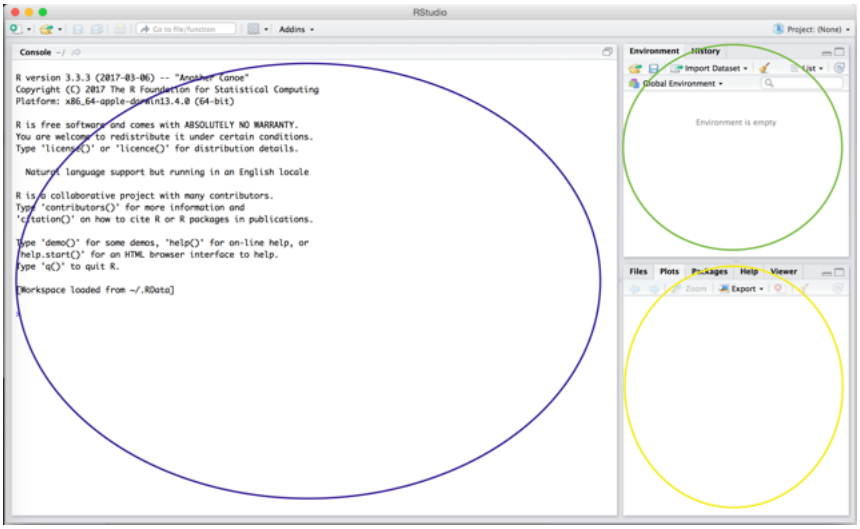
Discussion 1

Rummerfield & Berman

10/6/2017

R Basics

When you open R for the first time you get a confusing looking program. Below is a screenshot of what you get when you open R for the first time.

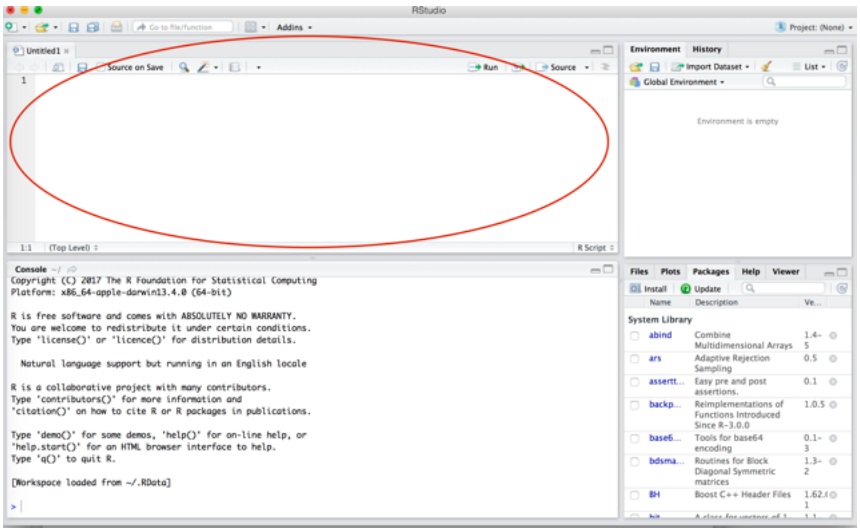


A brief tour of R: The blue circle is the console window; the yellow circle contains several different tabs, files, plots, packages, and the help panel are located here; the green circle contains the environmental and history tabs, it also contains a preview of your workspace and a shortcut on how to import data into R.

The console window is the place where R does actual work. While you can do your work in the console window, you are encouraged ***NOT*** to work within the console window.

Instead you should work in a script file! There are several reasons you should work in a script file, but the most important is that you can easily replicate and repeat your work. To open a script file, on the top of the console window click the "new" document tab in the upper left corner.

After opening a new script file, your R Studio will add a script window. When you type a new line in the script window, it will not immediately be executed.



Try some simple math in an R script file

```
3+2
3*2
3/2
```

To execute those lines, highlight the lines you want executed and click *Run* in the upper right corner of the script window. There is a keyboard shortcut for the *Run* button, **Ctrl+Enter**. If you've executed the lines you'll notice the output is within the console window.

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

```
## [1] 6
```

```
## [1] 1.5
```

You can program in R as well. You can assign values to variables, in the sense of computer science, by using the `<-` operator. For example, `x <- 5` assigns the value of 5 to `x`. While you can use `=` for assignment it is not recommended. Next, you can use the variable for mathematical purposes.

```
x = 5
x <- 5
x + 3
```

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

Often it is easier to work in vectors, lists, and data frames. One way to create a vector is via a sequence. By typing `x <- 1:12` you are creating a sequence from 1 to 12, incrementing by 1, so the result is the vector

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

Another way to create a vector is element by element. For example, if we wanted to make a vector that contains the numbers `1 2 3 5 7 11 13` then we can do that via the assignment operation and the combine function `c()`

```
x <- c(1, 2, 3, 5, 7, 11, 13)
x
```

```
## [1]  1  2  3  5  7 11 13
```

Note that when working with vectors, be careful! In general, R makes assumptions! This means that it assumes the user ***knows*** what the output should be. For example, R can carry out mathematical operations on a vectors element-by-element in some situations.

```
y <- 7:1
x + y
```

```
## [1]  8  8  8  9 10 13 14
```

```
x * y
```

```
## [1]  7 12 15 20 21 22 13
```

```
x + 2
```

```
## [1]  3  4  5  7  9 13 15
```

Other times it will not carry out operations as you expect and in such situations R may return a warning or an error.

```
x + c(1,2,3)
```

```
## Warning in x + c(1, 2, 3): longer object length is not a multiple of
## shorter object length
```

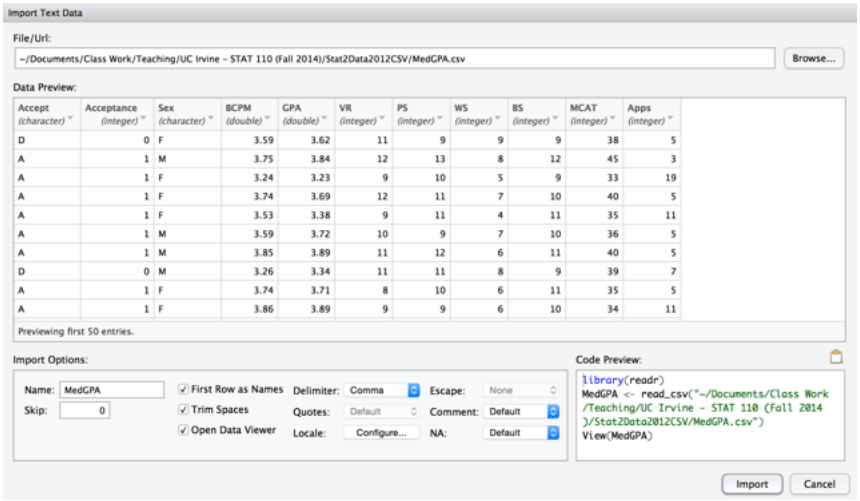
```
## [1]  2  4  6  6  9 14 14
```

When you do encounter an error or a warning, you might need some help understanding what went wrong. All functions in R have a description by typing `?` in front of the function or `help()` with the name of the function you need help with and R will return the help file for the function. If you have trouble understanding the help file or an error/warning message, try google or stackexchange. Remember: ***The purpose of this class is not R! You're expected to use R as a fancy calculator and when you get stuck, ask questions!***

Loading data in to R

Importing data via a .csv (or similar) file:

Click the "Import Dataset" tab on the environment (green circle) pane of the Rstudio window. Select the .csv option from the pulldown menu and either specify the url or the location of the .csv file you want to import in to R. Once that is done, the program will automatically generate the appropriate code and allow you to view the data.



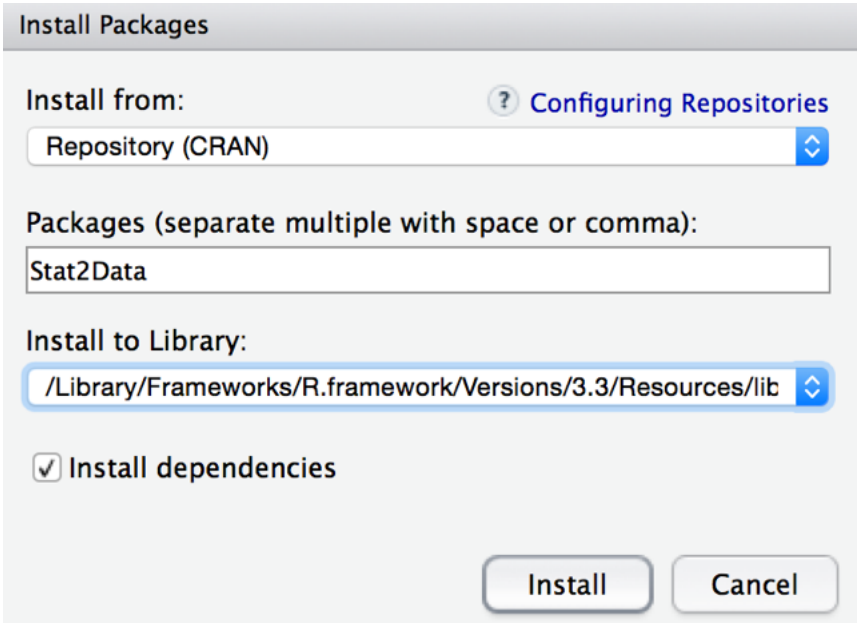
Importing data via read.table() function:

Another way to import data into R is to use a function. There are several functions all are of the form `read.` but the one we will focus on is `read.table()`. This function can be used when you have data stored as a .txt file. To use this function simply place either the url or path to the file within quotation marks.

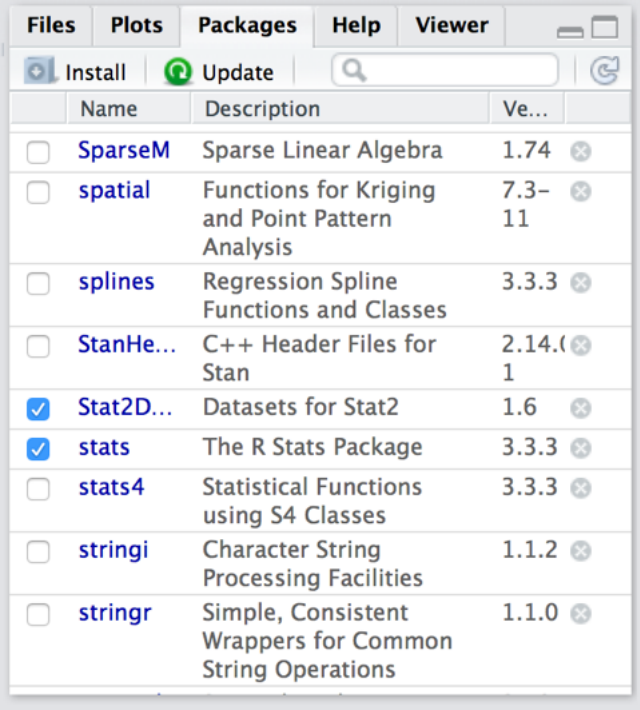
```
highway <- read.table("http://www.ics.uci.edu/~jutts/110/HighwaySign.txt")
```

Packages in R

There is another way to import datasets into R, but doing so requires a little understanding about packages in R. In the package tab in the lower right pane of the RStudio window you will see a list of packages. In R, each package is a collection of utilities, functions, and/or datasets. To add an additional package to your library, click the "Install" button in the package tab. A pop up window will appear and ask where are you getting the package from, select "Repository (CRAN)" (CRAN is the online comprehensive r archive network where developers publish their tools), in the name field, type the name of the package you want in your R library. Very important: be sure to check the box, "Install Dependencies" (this is due to the hierarchical nature of packages needing other functions). As an example, practice by installing the `Stat2Data` package.



Once your package is installed it still must be loaded in to the R library. To do this either check the box next to the name of the package or excute the following command, `library(Stat2Data)`



Data Frames in R

Data frames are a unique structure in R. Technically, they are lists of equal length vectors, but what this means for us is that they are like a matrix where each row is one observation and each column is a statistical variable. To load a data frame from the `Stat2Data` package when need to use the `data()` function.

```
data("MedGPA")
head(MedGPA)
```

```
##   Accept Acceptance Sex BCPM  GPA VR PS WS BS MCAT Apps
## 1      D          0   F 3.59 3.62 11  9  9  9   38    5
## 2      A          1   M 3.75 3.84 12 13  8 12   45    3
## 3      A          1   F 3.24 3.23  9 10  5  9   33   19
## 4      A          1   F 3.74 3.69 12 11  7 10   40    5
## 5      A          1   F 3.53 3.38  9 11  4 11   35   11
## 6      A          1   M 3.59 3.72 10  9  7 10   36    5
```

We could have typed `MedGPA` as opposed to `head(MedGPA)` but this would have returned all 55 observations within the data frame. Using `head()` returns the first six observations of the data frame.

The first observation is a woman (Sex = F) who was denied admission to medical school (Acceptance = 0), with a 3.59 GPA in biology, chemistry, physics and math (BCPM = 3.59) and an overall GPA of 3.62, with an MCAT score of 38, who applied to five medical schools (Apps = 5). The column labeled GPA is a vector that contains the overall GPA of the 55 observations.

To access specific elements of the data frame we need to learn about the `$` operator and subsets. The `$` operator allows us to access a specific column of the data frame. The following will return a vector containing the overall GPA of the 55 observations.

```
MedGPA$GPA
```

We can use functions on the vector to find some descriptive statistics about the GPA of the 55 observations.

```
median( MedGPA$GPA )
```

```
## [1] 3.58
```

```
mean( MedGPA$GPA )
```

```
## [1] 3.553273
```

```
var( MedGPA$GPA )
```

```
## [1] 0.08205576
```

If you want the quantiles of the GPA of the 55 observations use the `quantile()` function

```
quantile( MedGPA$GPA )
```

```
##    0%   25%   50%   75%  100%
## 2.720 3.375 3.580 3.770 3.970
```

Another way to access the overall GPA of the 55 applicants is to the use the `[]` subsetting feature. When you type

```
MedGPA[ 3, 5 ]
```

```
## [1] 3.23
```

Returns the value of the fifth (overall GPA) of the third observation.

```
MedGPA[ 3, ]
```

```
##   Accept Acceptance Sex BCPM  GPA VR PS WS BS MCAT Apps
## 3      A          1   F 3.24 3.23  9 10  5  9   33   19
```

Returns the third row of the MedGPA data frame.

```
MedGPA[ , 5 ]
```

```
##  [1] 3.62 3.84 3.23 3.69 3.38 3.72 3.89 3.34 3.71 3.89 3.97 3.49 3.77 3.61
## [15] 3.30 3.54 3.65 3.54 3.25 3.89 3.71 3.77 3.91 3.88 3.68 3.56 3.44 3.58
## [29] 3.40 3.82 3.62 3.09 3.89 3.70 3.24 3.86 3.54 3.40 3.87 3.14 3.37 3.38
## [43] 3.62 3.94 3.37 3.36 3.97 3.04 3.29 3.67 2.72 3.56 3.48 2.80 3.44
```

Returns the fifth column of the MedGPA data frame.

Thus `summary( MedGPA$GPA )` returns the same output as `summary( MedGPA[ , 5] )`

It is worth noting that `[]` can also be used to access to specific elements within a vector. This time you do not add the second index.

```
MedGPA$GPA[ 3 ]
```

```
## [1] 3.23
```

This returns the third element of the overall GPA column.

Another way to subset a vector or matrix is to give R a vector of logicals (ie all entries are TRUE/FALSE). For example if you wanted to find the mean GPA for all the admitted students in the MedGPA dataset we could create a logical vector where each admitted student is a TRUE and each denied student is a FALSE we need to utilize a logical operation.

```
MedGPA$Acceptance == 1
```

```
##  [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
## [23]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
```

```
## [34] FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
## [45] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Then subsetting the GPA vector by the logical operation above yields,

```
MedGPA$GPA[ MedGPA$Acceptance == 1 ]
```

```
##  [1] 3.84 3.23 3.69 3.38 3.72 3.89 3.71 3.89 3.97 3.49 3.77 3.54 3.54 3.89
## [15] 3.71 3.91 3.88 3.56 3.58 3.40 3.82 3.62 3.89 3.86 3.87 3.14 3.62 3.94
## [29] 3.97 3.48
```

Notice that the result above does not contain 55 entries; it contains 30 entries. We can then use the same functions we used above to analyze it.

```
summary( MedGPA$GPA[ MedGPA$Acceptance == 1 ] )
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.140   3.545   3.715   3.693   3.888   3.970
```

The basics of simple linear regression

We will cover this topic more in depth in the future but for today we are focusing on how to run simple linear regression in R. Practice this by assuming MCAT score is a function of BCPM (Biology, Chemistry, Physics, Mathematematics GPA).

The 'lm()' function (lm = linear model) allows you to obtain the estimated coefficients for your linear model, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$, and much more. The format of the function is the following: lm(y ~ x, data), where y is your response variable and x is your explanatory variable and data is the name of your dataframe. After saving your model (called 'fit' in the example below) the summary() function gives you information, including but not limited to, standard errors for each coefficient ($s_{\hat{\beta}}$), the test statistic under the assumption that $H_0 : \beta = 0$, and a two-sided p-value ($H_A : \beta \neq 0$). Furthermore, by taking the square root of the Multiple R-squared you can obtain the correlation.

```
fit <- lm( MCAT ~ BCPM, data = MedGPA)
summary( fit )
```

```
##
## Call:
## lm(formula = MCAT ~ BCPM, data = MedGPA)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9569  -3.0064  -0.0703   2.9936   8.5397
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   11.447      5.699   2.008   0.0497 *
## BCPM           7.092      1.620   4.377 5.68e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.167 on 53 degrees of freedom
## Multiple R-squared:  0.2655, Adjusted R-squared:  0.2517
## F-statistic: 19.16 on 1 and 53 DF,  p-value: 5.677e-05
```

Now suppose we are interested in the relationship between BCPM and MCAT only for those who were accepted. You can subset the data by adding a 'subset' argument in lm() like that seen below.

```r
fit <- lm( MCAT ~ BCPM, data = MedGPA, subset = Acceptance == 1)
summary( fit )
```

```
##
## Call:
## lm(formula = MCAT ~ BCPM, data = MedGPA, subset = Acceptance ==
##     1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4833 -2.3550  0.4837  2.0650  7.8771
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   12.051      9.525   1.265   0.2162
## BCPM           7.107      2.595   2.738   0.0106 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.752 on 28 degrees of freedom
## Multiple R-squared:  0.2112, Adjusted R-squared:  0.1831
## F-statistic: 7.499 on 1 and 28 DF,  p-value: 0.01061
```

The TL;DR

- R is a fancy calculator that can easily cause frustration. When you get stuck try google, stackexchange, the help files and the TA and instructor.
- R is not the purpose of this class! The underlying statistics is the purpose of this class!
- There are always several different ways to accomplish the same task in R. Do not be surprised if two different sources have two different ways to accomplish the same thing.
- You can do everything from complicated mathematics to simple arithemitic in R.
- There are two principal ways we will import data into R:
  - Using the "Import Dataset" shortcut in the environment tab.
  - Using the read.table() function.
  - Loading the dataset through a package then using the `data()` function.
- We started learning about the data frame object.
- We experimented with some descriptive statistics like `mean()`, `median()`, `var()`, `quantile()` and `summary()`.
- We started learning about how to carry out simple linear regression in R.