Introduction to R

Introduction to R (your new best friend?)

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of platforms: UNIX, Windows, and MacOS. The homepage for the R Project is:

cran.r-project.org

Installing R (newest version: 3.4.1 "Single Candle")
To install the program on your personal computer, go to the above address and download the version appropriate for your platform. The program has already been downloaded from a CRAN (Comprehensive R Archive Network) site and installed on the computers in Kingsbury N134, our Thursday computer lab. But I encourage you to install it on your own computer(s) as well.

Updating R
For those of you who already use R, you very likely have an old version installed. To update your version of R *without losing all the packages you have installed over time*, I highly recommend following the simple strategy described here:

www.datascienceriot.com/r/upgrade-R-packages/

Orientation to the program
When you start R, a window will open called the **R Console**. The R Console is R's user-interface, your means of communicating with the engine of R. This is the window that accepts commands from you (input), sends those commands to R's brain for processing, and returns an answer (output).

Some examples:

```
# R as a calculator
sqrt(4)
[1] 2
2^3
[1] 8
```

```
# Assign values to a symbolic variable
x <- 2

# Assign a vector of values to a vector name
yield <- c(30,55,67,48)
yield
[1]   30    55     67    48

# List variables
ls()
“x” “yield”
# Operate with vectors of equal length
oil_content <- c(0.25,0.20,0.17,0.23)
oil_yield <- yield * oil_content
oil_yield
[1]  7.50 11.00 11.39 11.04

# Calculate correlations and test hypotheses
cor(yield, oil_content)
[1] -0.9665154

cor.test(yield, oil_content)
      Pearson's product-moment correlation

data:  yield and oil_content
t = -5.3266, df = 2, p-value = 0.03348
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.99932450 -0.07635285
sample estimates:
       cor
-0.9665154
```

You get the idea. The second type of window in R, and one which we use a lot, is called the **Script** window. Scripts are *permanent, repeatable, annotated, shareable, cross-platform archives of your analysis* (language from Beckerman and Petchey, "Getting Started with R: An Introduction for Biologists"). The **Script** window is a bit like a word processor, in that you can write things; but there is one powerful difference: There are combinations of keyboard strokes that send information from a Script to the R Console, automatically.

To start a new Script, you would simply go to the File menu and choose *New document*. In this script, you can type the same commands as you did above:

```
sqrt(4)
2^3

x <- 2

yield <- c(30,55,67,48)
yield

ls()

oil_content <- c(0.25,0.20,0.17,0.23)
oil_yield <- yield * oil_content
oil_yield

cor(yield, oil_content)
cor.test(yield, oil_content)
```

And you can now save these commands (this script) by clicking Save under the File menu. Notice when you typed these commands into the **Script** window that R did not process any of them. To send them to the **Console** for processing, simply highlight the lines you want R to crunch and press Command-Return (Mac) or Control-R (PC). If you only want to send one line at a time, it's sufficient to have your cursor somewhere on that line and press the platform-appropriate key combination. Try it.

Now that you know how to make a script, let's open one up that I've already made so we can walk through it. Please open the file *Lab1ex1.R*.

Example 1 *[Lab1ex1.R]*

```
#BIOL933
#Lab 1
#Script 1

#This script creates a new data set called SeedCount (a list) and calculates
 some basic statistics

SeedCount <- c(210,221,218,228,220,227,223,224,192)

length(SeedCount)
mean(SeedCount)
sd(SeedCount)
se(SeedCount)

se <- function(x) {sd(x)/sqrt(length(x))}
se(SeedCount)

CV <- function(x) {100 * sd(x)/mean(x)}
CV(SeedCount)
```

Things to Learn

1. Run (submit) an R Script (or parts of an R Script) directly from the Script window using the keyboard (Again, for Windows: control + R; for MacOS: command+return)
2. Look to the **R Console** for errors (red-type)
3. Save scripts to disk

Quick exercise

Using the script from Example 1 as a guide, create and save a new script that calculates the average disease severity (% area of infected tissue) of the following five tissue samples:

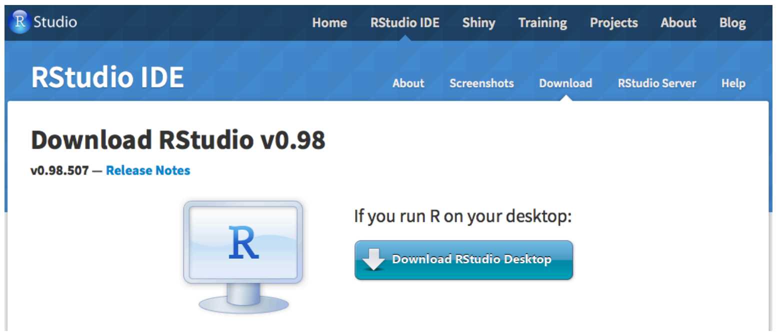
Leaf	Total area of tissue (cm^2)	Diseased area (cm^2)
1	14	5
2	16	3
3	12	4
4	15	6
5	17	8

For the interested: Why isn't the mean of the five severities equal to the severity of the means? Which is right?

<u>Introduction to RStudio (your new best-best friend?)</u>

Like command-line R, the more user-friendly RStudio interface is powerful and open-source (free)! RStudio is a windows-based and drop-down menu front-end IDE (integrated development environment) that makes working in R easier. To download the desktop version for your computer, go to:

http://www.rstudio.com/ide/download/



Please install the latest version, namely v1.0.153. (Note that RStudio requires R to also be installed on your computer). Again, on the lab computers, RStudio has already been installed for us.

The RStudio layout

Script (or Source) Pane: As in the command-line version of R, this pane is the place to write new scripts, open/modify/run old scripts, and send script commands to the console. On PCs running Windows, the RStudio Script Pane offers better code annotation/formatting than the command-line version.

Console Pane: This is the same Console Pane you know and love from R.

Script Pane: Same as in R.

Session Pane: RStudio includes some very useful features related to data input and memory management, via the two tabs within its Session Pane:

> *Workspace tab:* Here RStudio allows you to very easily import data into R (yes!), clear objects from R's memory (rm()), and view a live summary of all the objects and values in R's memory.
>
> *History tab:* This tab maintains a running archive of all commands sent to the Console, both from the current session and all previous sessions. Note that, like the Workspace tab, this History can be completely cleared.

Management Pane: Finally, RStudio has some useful features related to the management of files, plots, and packages, through its Management Pane:

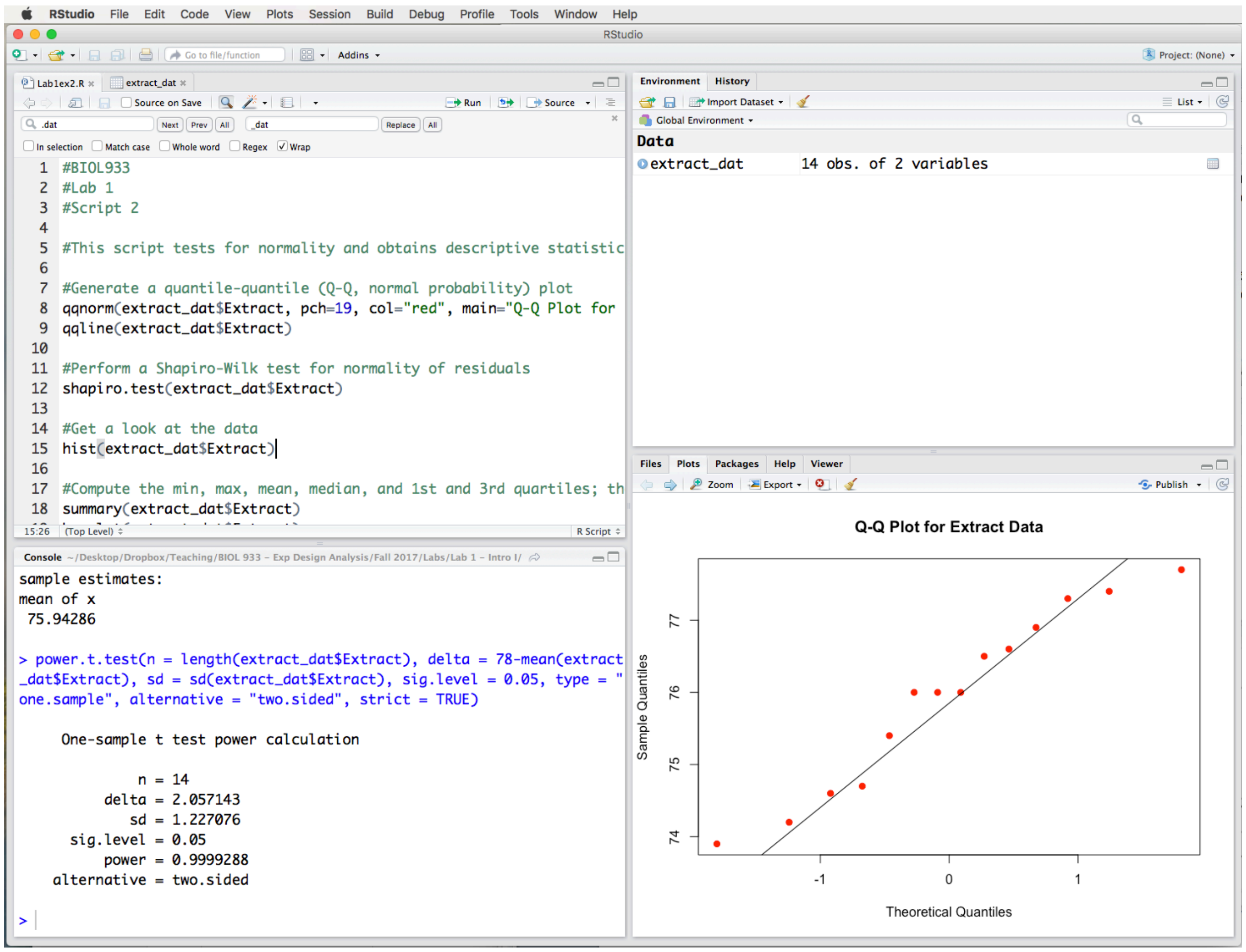
> *Files:* This tab provides a visual representation of your computer's file system and allows you to easily select your session's "Working Directory" (via the "More" menu).

Plots: All plots generated during your session can be found here. This tab includes an "Export" menu to facilitate exporting final plots in a variety of formats.

Packages: This tab gives you a list of all R packages that are installed on your system and which of those packages are activated in the current session (library()). This tab also facilitates the installation of new packages, as you need them, and finding package updates.

Help: An intuitive Help tab for R.

In the screenshot below, the Script Pane is upper-left. Going clock-wise, you see the Session Pane, the Management Pane, and the Console Pane. Note, however, that the layout and contents of the above Panes are completely customizable (look under RStudio Preferences).



```
#BIOL933
#Lab 1
#Script 2

#This script tests for normality and obtains descriptive statistic

#Generate a quantile-quantile (Q-Q, normal probability) plot
qqnorm(extract_dat$Extract, pch=19, col="red", main="Q-Q Plot for
qqline(extract_dat$Extract)

#Perform a Shapiro-Wilk test for normality of residuals
shapiro.test(extract_dat$Extract)

#Get a look at the data
hist(extract_dat$Extract)

#Compute the min, max, mean, median, and 1st and 3rd quartiles; th
summary(extract_dat$Extract)
```

```
Data
extract_dat        14 obs. of 2 variables
```

```
Console ~/Desktop/Dropbox/Teaching/BIOL 933 - Exp Design Analysis/Fall 2017/Labs/Lab 1 - Intro I/
sample estimates:
mean of x
 75.94286

> power.t.test(n = length(extract_dat$Extract), delta = 78-mean(extract
_dat$Extract), sd = sd(extract_dat$Extract), sig.level = 0.05, type = "
one.sample", alternative = "two.sided", strict = TRUE)

     One-sample t test power calculation

              n = 14
          delta = 2.057143
             sd = 1.227076
      sig.level = 0.05
          power = 0.9999288
    alternative = two.sided

>
```

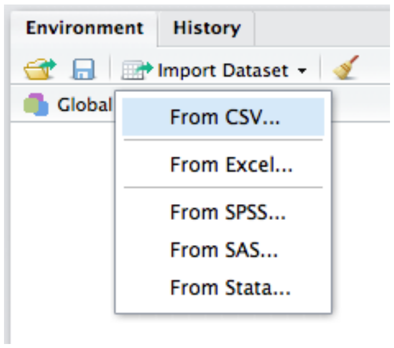
Example 2 *[Lab1ex2.R]*

In this next example, we will read in a dataset (barley extract values) from a comma-delimited (.csv) file, test normality, and perform a t-test.

Importing data with RStudio

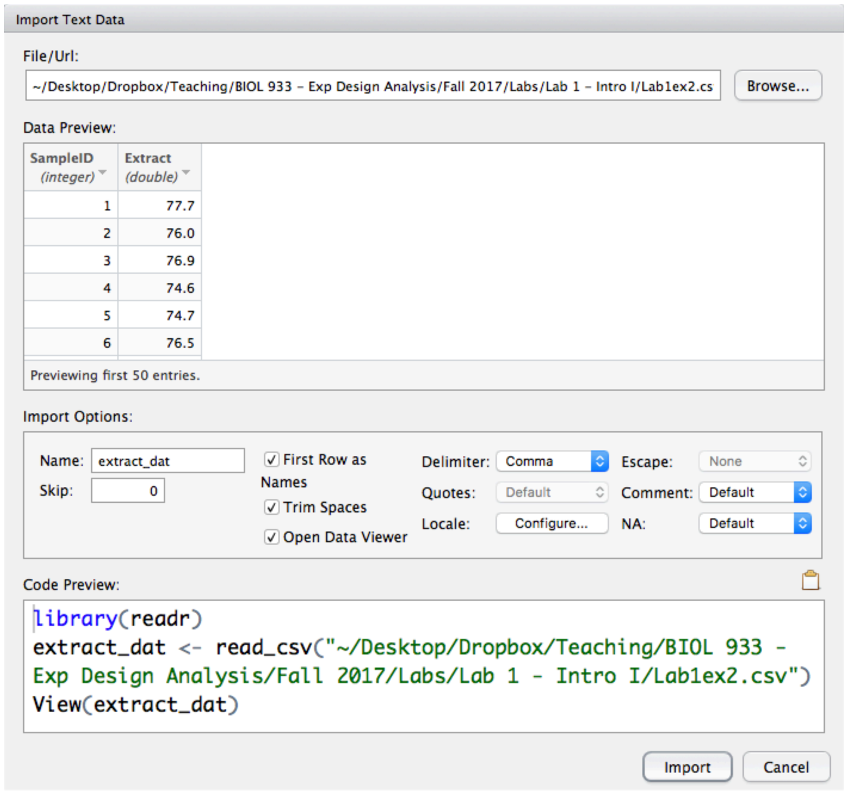
We're going to take advantage of RStudio's Import feature rather than importing data through the script itself (functions such as read.table(), etc.). To begin, clear R's memory using either the command rm(list=ls()) in the Console or by clicking on the Clear Objects button in the Workspace Pane (the button that looks like a broom:).

To import the Lab1ex2.csv data into R, click on the "Import Dataset" menu in the Workspace Pane and select "From CSV...":



Browse to the file Lab1ex2.csv (a **C**omma **S**eparated **V**alues text file, originally created in Excel) and select it.

When you select the file, a window will appear that gives you a preview of the dataset and allows you to designate some of its features:



Import Text Data

File/Url: ~/Desktop/Dropbox/Teaching/BIOL 933 – Exp Design Analysis/Fall 2017/Labs/Lab 1 – Intro I/Lab1ex2.cs

Data Preview:

SampleID (integer)	Extract (double)
1	77.7
2	76.0
3	76.9
4	74.6
5	74.7
6	76.5

Previewing first 50 entries.

Import Options:

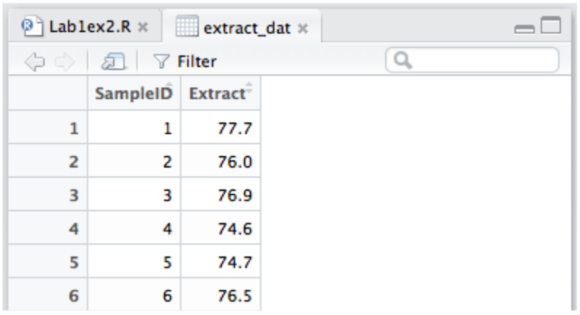
Name: extract_dat
Skip: 0
✓ First Row as Names
✓ Trim Spaces
✓ Open Data Viewer
Delimiter: Comma
Quotes: Default
Locale: Configure...
Escape: None
Comment: Default
NA: Default

Code Preview:

```
library(readr)
extract_dat <- read_csv("~/Desktop/Dropbox/Teaching/BIOL 933 -
Exp Design Analysis/Fall 2017/Labs/Lab 1 - Intro I/Lab1ex2.csv")
View(extract_dat)
```

In the script, I refer to this dataset as "extract_dat", so first you should change its name from "Lab1ex2" to "extract_dat" during import (see "Name:" field in the box above). The dataset contains a header, so make sure the "First Row as Names" box is checked. It is also comma-separated (see "Delimiter" menu). Once you have changed the Name and inspected everything, click "Import."

When you do this, you will immediately see the new dataframe show up in your Workspace Pane. You will also see a copy of the dataset appear in the Excel-like dataviewer in a new tab within your Script Pane:



	SampleID	Extract
1	1	77.7
2	2	76.0
3	3	76.9
4	4	74.6
5	5	74.7
6	6	76.5

As you should always do, inspect your data and make sure everything looks right -- if it doesn't, fix it! Now that the data is in R's working memory, you can proceed with running the script. Easy. Thank you, RStudio.

Now go ahead and open the script Lab1ex2.R.

```
#BIOL933
#Lab 1
#Script 2

#This script tests for normality and obtains descriptive statistics

#First, convert the imported data into a "dataframe" object
extract_dat<-as.data.frame(extract_dat)

#Generate a quantile-quantile (Q-Q, normal probability) plot
qqnorm(extract_dat$Extract, pch=19, col="red", main="Q-Q Plot for Extract
   Data")
qqline(extract_dat$Extract)

#Perform a Shapiro-Wilk test for normality of residuals
shapiro.test(extract_dat$Extract)

#Get a look at the data
hist(extract_dat$Extract)

#Compute the min, max, mean, median, and 1st and 3rd quartiles; then
visualize them
summary(extract_dat$Extract)
boxplot(extract_dat$Extract)

#Hypothesis Testing...conduct a one-sample t-test (two-tailed) for a mean
t.test(extract_dat$Extract, mu=78) #null H, mean = 78
power.t.test(n = length(extract_dat$Extract), delta = 78-
   mean(extract_dat$Extract), sd = sd(extract_dat$Extract), sig.level = 0.05,
   type = "one.sample", alternative = "two.sided", strict = TRUE)
```

Comments on the script

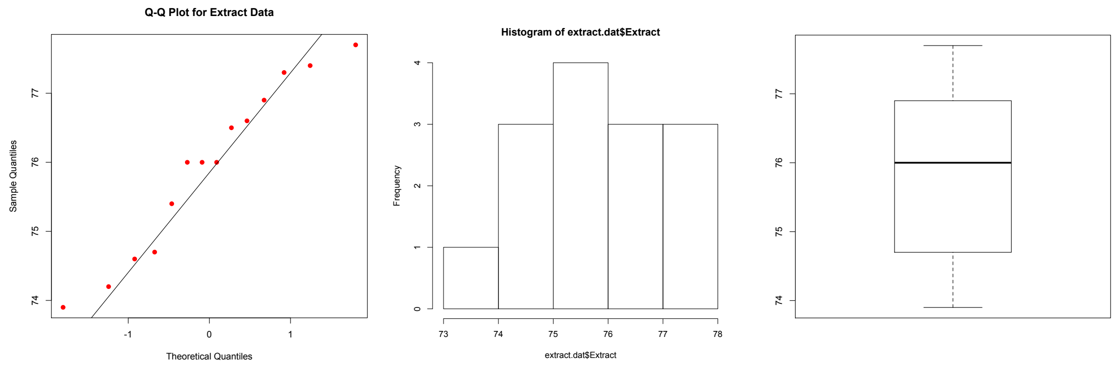
1. The hash character (#) at the beginning of a line tells R to ignore that line. It is a useful way of annotating (writing notes) in your script.
2. R works through the use of **functions**, which are commands followed by parentheses. For example, rm() is a function for removing (rm) something from R's memory. ls() is a function for listing (ls) everything in R's memory. By combining them rm(list=ls()), you can tell R to remove everything in its memory.
3. The string character ($) is used to designate a specific column in a dataframe. So, extract.dat$Extract is the column of extract values in our dataset.

Output

```
      Shapiro-Wilk normality test

data:  extract.dat$Extract
W = 0.9458, p-value = 0.4974

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  73.90   74.88   76.00   75.94   76.82   77.70
```



NOTE: The Shapiro-Wilk "W" statistic measures the linear correlation between the data and their normal scores. The closer W is to 1, the better correlated the distribution is to a normal distribution. Normality is rejected when W is sufficiently smaller than one, that is, when the value Pr < W is less than 0.05. In this example, p = 0.4974 > 0.05, so we conclude the data exhibit a normal distribution.

How to get help within R

Some of the most useful command in R are **?** or alternatively **help** and **??** (or alternatively **help.search**). Use **?** (or **help(yourcommand)**) if you know the exact command name and use **??yourcommand** (or **help.search("yourcommand")**) if you don't know the command name. R will return a list of commands relevant to your search. Some examples:

```
?setwd        # the same as  help(setwd)
??shapiro     # the same as  help.search("shapiro")
```

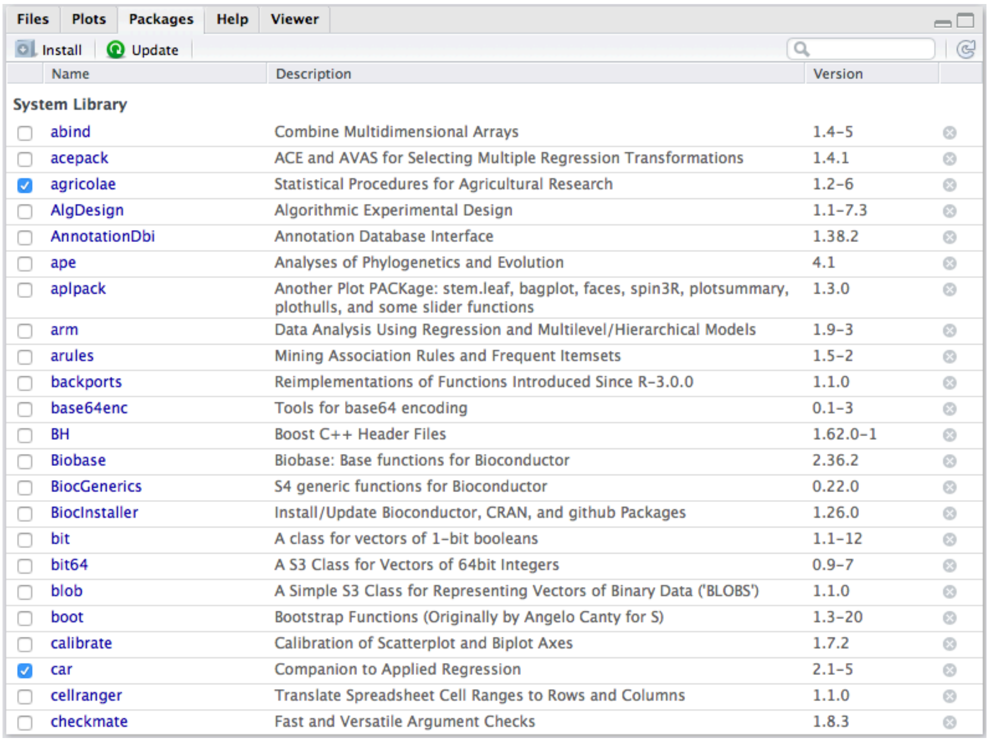
Any R function can be queried in this way. For example:

```
?t.test       # check the parameters that can be set
```

And so on...

Package installation and management

Another very nice feature of RStudio is that it makes the installation and management of Packages very easy. The "Packages" tab in the Management Pane shows you a list of all the packages that have been installed on your system. To activate an installed package in the current session, simply click on its checkbox.



System Library

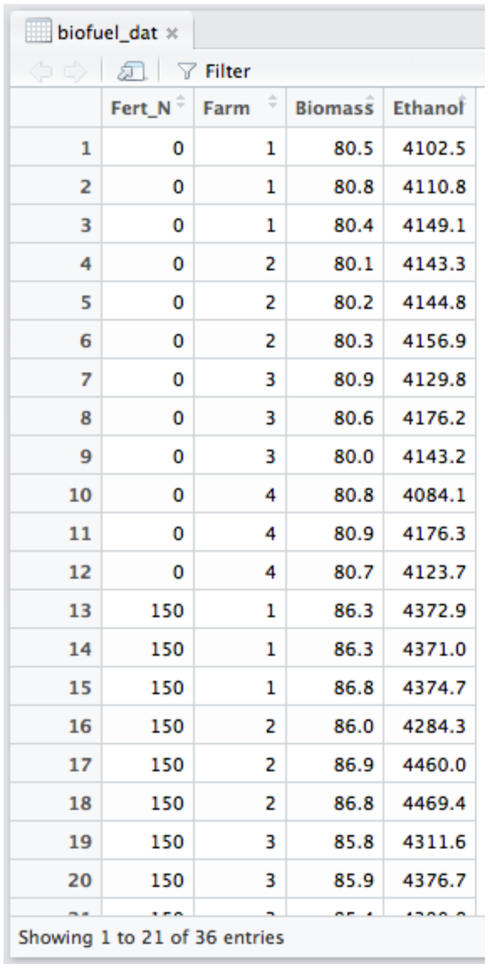
	Name	Description	Version
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
✓	agricolae	Statistical Procedures for Agricultural Research	1.2-6
☐	AlgDesign	Algorithmic Experimental Design	1.1-7.3
☐	AnnotationDbi	Annotation Database Interface	1.38.2
☐	ape	Analyses of Phylogenetics and Evolution	4.1
☐	aplpack	Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plotsummary, plothulls, and some slider functions	1.3.0
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.9-3
☐	arules	Mining Association Rules and Frequent Itemsets	1.5-2
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.0
☐	base64enc	Tools for base64 encoding	0.1-3
☐	BH	Boost C++ Header Files	1.62.0-1
☐	Biobase	Biobase: Base functions for Bioconductor	2.36.2
☐	BiocGenerics	S4 generic functions for Bioconductor	0.22.0
☐	BiocInstaller	Install/Update Bioconductor, CRAN, and github Packages	1.26.0
☐	bit	A class for vectors of 1-bit booleans	1.1-12
☐	bit64	A S3 Class for Vectors of 64bit Integers	0.9-7
☐	blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.1.0
☐	boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-20
☐	calibrate	Calibration of Scatterplot and Biplot Axes	1.7.2
✓	car	Companion to Applied Regression	2.1-5
☐	cellranger	Translate Spreadsheet Cell Ranges to Rows and Columns	1.1.0
☐	checkmate	Fast and Versatile Argument Checks	1.8.3

To install new packages, click the "Install Packages" button and follow the prompts.

Operating on data: Subsetting

Some fun and arguably useful challenges to get our minds into R

Take a moment to clear R's memory; then import the dataset Lab1ex3.csv, naming it whatever you like (I named mine "biofuel_dat"). This is hypothetical data of lignocellulosic biomass yields (tons/acre) and ethanol yields (gallons/acre) of a new biofuel feedstock grown on four farms under three different fertility regimes.



	Fert_N	Farm	Biomass	Ethanol
1	0	1	80.5	4102.5
2	0	1	80.8	4110.8
3	0	1	80.4	4149.1
4	0	2	80.1	4143.3
5	0	2	80.2	4144.8
6	0	2	80.3	4156.9
7	0	3	80.9	4129.8
8	0	3	80.6	4176.2
9	0	3	80.0	4143.2
10	0	4	80.8	4084.1
11	0	4	80.9	4176.3
12	0	4	80.7	4123.7
13	150	1	86.3	4372.9
14	150	1	86.3	4371.0
15	150	1	86.8	4374.7
16	150	2	86.0	4284.3
17	150	2	86.9	4460.0
18	150	2	86.8	4469.4
19	150	3	85.8	4311.6
20	150	3	85.9	4376.7
21	150	3	85.4	[illegible]

Showing 1 to 21 of 36 entries

Challenge 1: Suppose you are interested in all the biomass yields (3rd column) from Farm 2 (i.e. rows 4-6, 16-18, and 28-30). How could you select these multiple elements?

Challenge 2: Suppose you are interested in knowing which farms and which fertility treatments resulted in ethanol yields greater than 4400 gallons/acre. How could you select these multiple elements?

Some solutions to these challenges are on the following pages.

Operating on data: Subsetting (key)

Challenge 1: Suppose you are interested in all the biomass yields (3rd column) from Farm 2 (i.e. rows 4-6, 16-18, and 28-30). How could you select these multiple elements?

First, convert the imported data into a "dataframe" object:

```
biofuel_dat<-as.data.frame(biofuel_dat)
```

A brute force approach:

```
biofuel_dat[c(4:6,16:18,28:30),3]
[1] 80.1 80.2 80.3 86.0 86.9 86.8 89.0 88.1 88.7
```

Remember, c(4:6,16:18,28:30) is a vector of numbers [c() is for combine]. We could similarly select these same rows, with all the columns:

```
biofuel_dat[c(4:6,16:18,28:30),]
   Fert_N Farm Biomass Ethanol
4       0    2    80.1  4143.3
5       0    2    80.2  4144.8
6       0    2    80.3  4156.9
16    150    2    86.0  4284.3
17    150    2    86.9  4460.0
18    150    2    86.8  4469.4
28    300    2    89.0  4292.9
29    300    2    88.1  4236.6
30    300    2    88.7  4232.9
```

Using the subset() function:

```
sub_farm2 <- subset(biofuel_dat, Farm == 2)
sub_farm2
   Fert_N Farm Biomass Ethanol
4       0    2    80.1  4143.3
5       0    2    80.2  4144.8
6       0    2    80.3  4156.9
16    150    2    86.0  4284.3
17    150    2    86.9  4460.0
18    150    2    86.8  4469.4
28    300    2    89.0  4292.9
29    300    2    88.1  4236.6
30    300    2    88.7  4232.9
```

```
sub_farm2$Biomass
[1] 80.1 80.2 80.3 86.0 86.9 86.8 89.0 88.1 88.7
```

Challenge 2: Suppose you are interested in knowing which farms and which fertility treatments resulted in ethanol yields greater than 4400 gallons/acre. How could you select these multiple elements?

One way to do this, again, is with the subset() function:

```
sub_biofuel <- subset(biofuel_dat, Ethanol >= 4400)
sub_biofuel
   Fert_N Farm Biomass Ethanol
17    150    2    86.9  4460.0
18    150    2    86.8  4469.4
22    150    4    86.8  4411.7
23    150    4    86.9  4441.7
24    150    4    86.6  4423.2
25    300    1    92.9  4401.3
27    300    1    92.4  4411.7
32    300    3    89.8  4416.9
34    300    4    92.3  4489.5
35    300    4    92.2  4468.6
36    300    4    92.4  4538.2
```

The subset() function allows any number of conditionals. For example, you could extract those rows that have ethanol yields of at least 4400 g/ac AND (&) biomass yields of less than 90 tons/ac:

```
sub2_biofuel <- subset(biofuel_dat, Ethanol >= 4400 & Biomass <
     90)
sub2_biofuel
   Fert_N Farm Biomass Ethanol
17    150    2    86.9  4460.0
18    150    2    86.8  4469.4
22    150    4    86.8  4411.7
23    150    4    86.9  4441.7
24    150    4    86.6  4423.2
32    300    3    89.8  4416.9
```

Now here is another way to do all of this. As well as storing numbers and character strings, R can also store *logicals*: TRUE and FALSE. To make a new vector, with elements that are TRUE if ethanol yield is at least 4400 g/ac and FALSE if it is below 4400 g/ac:

```
min_4400 <- biofuel_dat$Ethanol >= 4400
```

Which farms and fertility treatments were these, and what were their ethanol yields?

```
biofuel_dat[min_4400,c(1,2,4)]
   Fert_N Farm Ethanol
17    150    2  4460.0
18    150    2  4469.4
22    150    4  4411.7
23    150    4  4441.7
24    150    4  4423.2
25    300    1  4401.3
27    300    1  4411.7
32    300    3  4416.9
34    300    4  4489.5
35    300    4  4468.6
36    300    4  4538.2
```