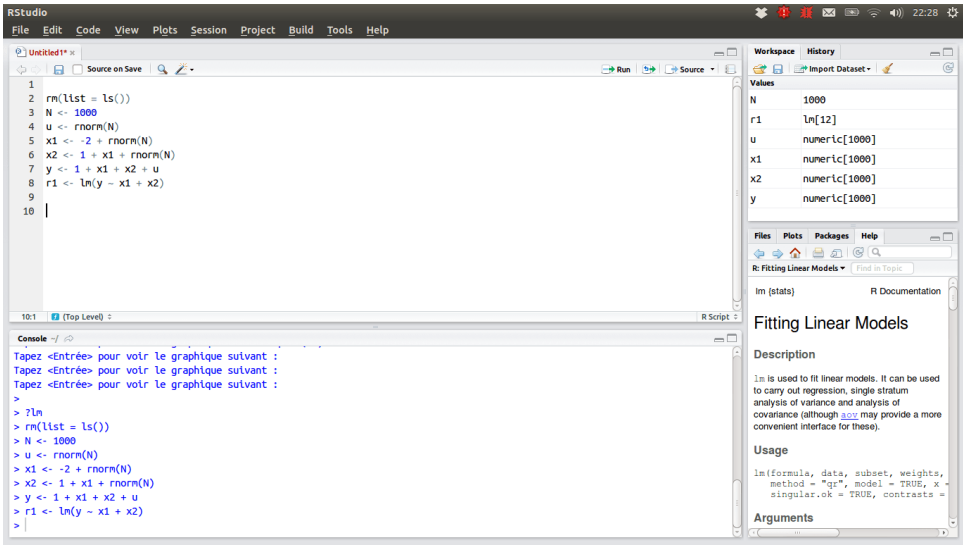
An introduction to R and programming

Jake Cooper and Andy Magee

So what is Rstudio anyways?



Rstudio makes using R easier! There are 4 windows, you can re-arrange them to your liking. The least important one is the upper right, we will ignore it.

The upper left is useful because it allows you to work on/look at a script. This is where you'll write your code. In the image, the user is working in an unsaved script file. This is a bad idea! Always save your files!

The lower left is called the console. It's where the code is actually run. You can type code directly in there, or you can run a line from the script you're working on (command + Enter runs the line your cursor is on, or the lines you highlight).

The lower right is important, because it is both where plots show up and where R help is displayed. If you ever have a question about how something works in R, you can type ? followed by the thing you want to get help about. Try the following line in your console.

```
?help
```

How does R work?

To the uninitiated, code can look a bit intimidating, and rather illegible. The purpose of this introduction and assignment is to provide some basic familiarity for a key part of the course (we will be using R for 3 lab assignments).

Coding is really just a bit of applied logic and a bit of applied math. Sound simple? It certainly can be! Like math and logic, you can build a lot of simple pieces into a complicated, powerful tool.

One thing we will need to be able to do is use variables. What is a variable? It stores a value (a number, some text, a DNA sequence, you name it). We assign variables with "<-". We can do arithmetic on these variables. If a and b are the sides of a right triangle, $c = \sqrt{a^2 + b^2}$, and we can make R do this for us. One thing you can't do in R is write that arithmetic like you did in calculus class. ab doesn't mean multiply a and b, R thinks it should go look for a variable named "ab".

Note that when running commands/lines in R, anything after a # is ignored. That means # is our comment character, we can write comments after a command (or, usually, on the lines above the commands) that explain what is going on.

```r
a <- 3 # our variable a now is 3, let's print to check
a
```

```
## [1] 3
```

```r
b <- 4 # our variable a now is 10

hello <- "hello world" # our variable b is now a character argument, let's print to check
hello
```

```
## [1] "hello world"
```

```r
c <- sqrt(a^2 + b^2) # ^ means "raise the first thing to the power of the second thing"
c
```

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

In R, a lot of math is done by functions. There are many functions already written in R, and there are many more that you can gain access to by installing code (called a package) written by other people. Let's look at a simple function, ${}_nC_k$ (often written $\binom{n}{k}$). In poker, when you're dealt 5 cards, there are $\binom{52}{5} = 2598960$ hands you could get from the dealer. The formula for counting these combinations is: $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ Let's write it out, and then we can test it on the card example.

```r
nCk <- function(n,k) { # Here we tell R that we're making a function called nCk
  # This function takes 2 arguments, one called n, one called k
  nck <- factorial(n)/(factorial(n-k)*factorial(k)) # Calculate the value, store it in nck
  return(nck) # Output the value we calculated
}

nCk(52,5)
```

```
## [1] 2598960
```

Note, also, that function arguments are not interchangeable. If you just give the function a long set of inputs, it assumes they're in the correct order. This problem can be solved by either 1) always putting the arguments in the right order or 2) always explicitly telling R what you're giving it (the better option)

```r
nCk(5,52) # This is not the same as nCk(52,5),
```

```
## Warning in gamma(x + 1): NaNs produced
```

```
## [1] NaN
```

What we just asked R to compute it in fact it is impossible, so it produced an error message and told us that the answer is Not a Number (NaN).

Now, let's use option 2 and see how we can avoid these issues.

```r
nCk(k=5,n=52) # This is the same as nCk(52,5)
```

```
## [1] 2598960
```

```r
nCk(n=52,k=5) # This is also the same as nCk(52,5)
```

```
## [1] 2598960
```

What if we wanted to know about more than one combination? What if we had 4 antibiotics on hand (like in your BIOL 180 lab), and we wanted to know how many 1-, 2-, 3-, and 4-antibiotic treatments we could make? We could use a loop to avoid writing out similar statements over and over and over (this matters more when the thing you have to write out is long). First we need to tell R to expect a bunch of output. We can use a vector to do that.

```r
number_of_treatments <- numeric(4) # We're making a new vector that has 4 slots for values

for (i in 1:4) { # We are going to loop over the numbers from 1 to 4 (1,2,3,4)
  number_of_treatments[i] <- nCk(4,i) # In the ith slot, we put the appropriate value
}
```

```
# Now we can print the vector and see how many treatments we could make.
number_of_treatments
```

```
## [1] 4 6 4 1
```

How is R useful to a scientists?

Alright, that's a whirlwind tour through basic programming, but that's only part of the story. R is great for data analysis, so we have to be able to get that data in, right? There are many ways to do this, but we're going to be working with a collection of R packages known as the "tidyverse" that are designed to make R programming easier. First you need to install the packages (if you haven't already), which is easy! There is one other package we suggest installing, a new part of the tidyverse not currently installed with all the others. We'll get to why you want it shortly.

```
install.packages("tidyverse") # This may take a few seconds
install.packages("readxl") # This is just one package, and will be faster
```

I already have this installed, but at this point a lot of text (red, unless you've changed the color scheme) will scroll across your console.

Now you can use all the packages you just installed! You first have to tell R to load them, then you can use any functions you want from them. Let's do some of the most common tasks in R. We'll read in a spreadsheet, plot a bit of data, and then test if the pattern we see is real. In the olden days before 2017, this could be a bit annoying. Most people have their data in a spreadsheet that they use in Excel. R couldn't read those, so you'd save it as a comma separated values (.csv) file and read that into R. Then you'd catch a mistake, fix it in Excel, re-export the data, and the cycle never ended. Now? Now there is `readxl`, which allows you to read Excel-style spreadsheets, so you don't have to keep two copies of anything!

```
library(tidyverse)
library(readxl)
```

The old way:

```
iris <- read_csv("iris.csv")
```

The new way:

```
iris <- read_excel("iris.xlsx")
```

We can preview the table by running a variety of commands, this one keeps us from printing too much:

```
head(iris,3)
```

```
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl>   <chr>
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

So, we can see that there are 4 measurements and data on the species. I bet petal length and petal width are related, no? First, let's see if there really is data on more than one species.

```
unique(iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

There are three species in the dataset, let's just look at one. First we will run a regression and print the output, then we'll plot the data points and the regression line.

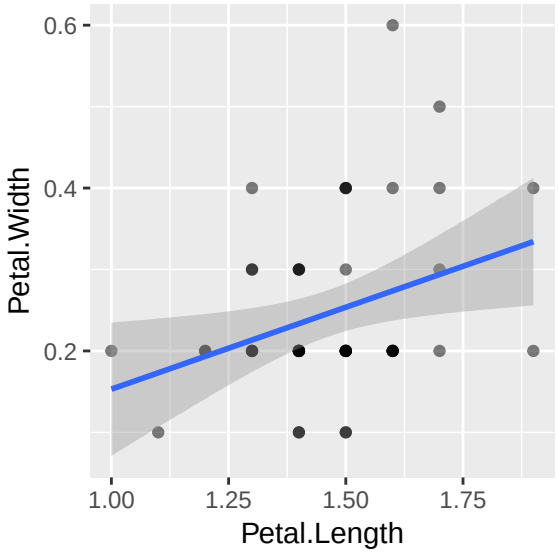
```
setosa <- filter(iris, Species == "setosa")

my_hypothesis_test <- lm(Petal.Width ~ Petal.Length, data=setosa)
```

```
summary(my_hypothesis_test)
```

```
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length, data = setosa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.15365 -0.05365 -0.03352  0.06632  0.32623
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.04822    0.12164  -0.396   0.6936
## Petal.Length  0.20125    0.08263   2.435   0.0186 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1005 on 48 degrees of freedom
## Multiple R-squared:   0.11,  Adjusted R-squared:  0.09144
## F-statistic: 5.931 on 1 and 48 DF,  p-value: 0.01864
```

Well, our slope is significant, so it looks like we were right that there is a relationship. And the slope is positive, which makes sense. To the plot!

```
ggplot(setosa, aes(x=Petal.Length,y=Petal.Width)) + geom_point(alpha=0.5) + geom_smooth(method="lm")
```



Fin

The goal of this worksheet is not to intimidate you, but to help you gain familiarity with a lot of important features in R we'll be seeing throughout the quarter. You can be successful coding in R! To make your lives easier, we'll be providing you templates or completed scripts. One of the most useful skills you can have as a computer-using biologist is the ability to take a piece of code that nearly does what you want, and rework it to do exactly what you want. It's a skill we use as biologists all the time, when adapting protocols or extrapolating ideas from microbes to animals.

Assignment

Please turn in a .R file with the code for parts 1 and 2a. Write everything else up in a word file.

1

Write a function that either computes the length of the hypotenuse of a triangle given a and b, or that computes ${}_nP_k$ (${}_nP_k = \frac{n!}{(n-k)!}$, useful if we asked how many 5 card poker hands there are, paying attention to which card was dealt first, second, third, etc.).

2

a)

Read in the iris data, and test a different hypothesis than the one above. You can test the same relationship in another species, or a different relationship in the same species, just not "petal width is related to petal length in *Iris setosa*.

b)

Plot the data and a regression line. (Take a screenshot of this and add it to your report.)

c)

Is the slope significant?

3

a)

When we plotted things, we used geom_point(alpha=0.5). What is alpha doing?

b)

The plot of the regression line includes the line and a grey area around that. What do you think that is?

c)

When we did the regression, we wrote out "formula = Petal.Width ~ Petal.Length" what was that about?

d)

How well did our regression do at explaining the relationship between petal length and petal width? In other words, how much of the variance in petal width is explained by the variance in petal length? How does this compare to the hypothesis you tested? What do you think that might say about the biology of the genus *Iris*?

Hint: If you're having trouble figuring out any of these, try asking R for help, and don't be afraid to click on a few links and try a few pages.