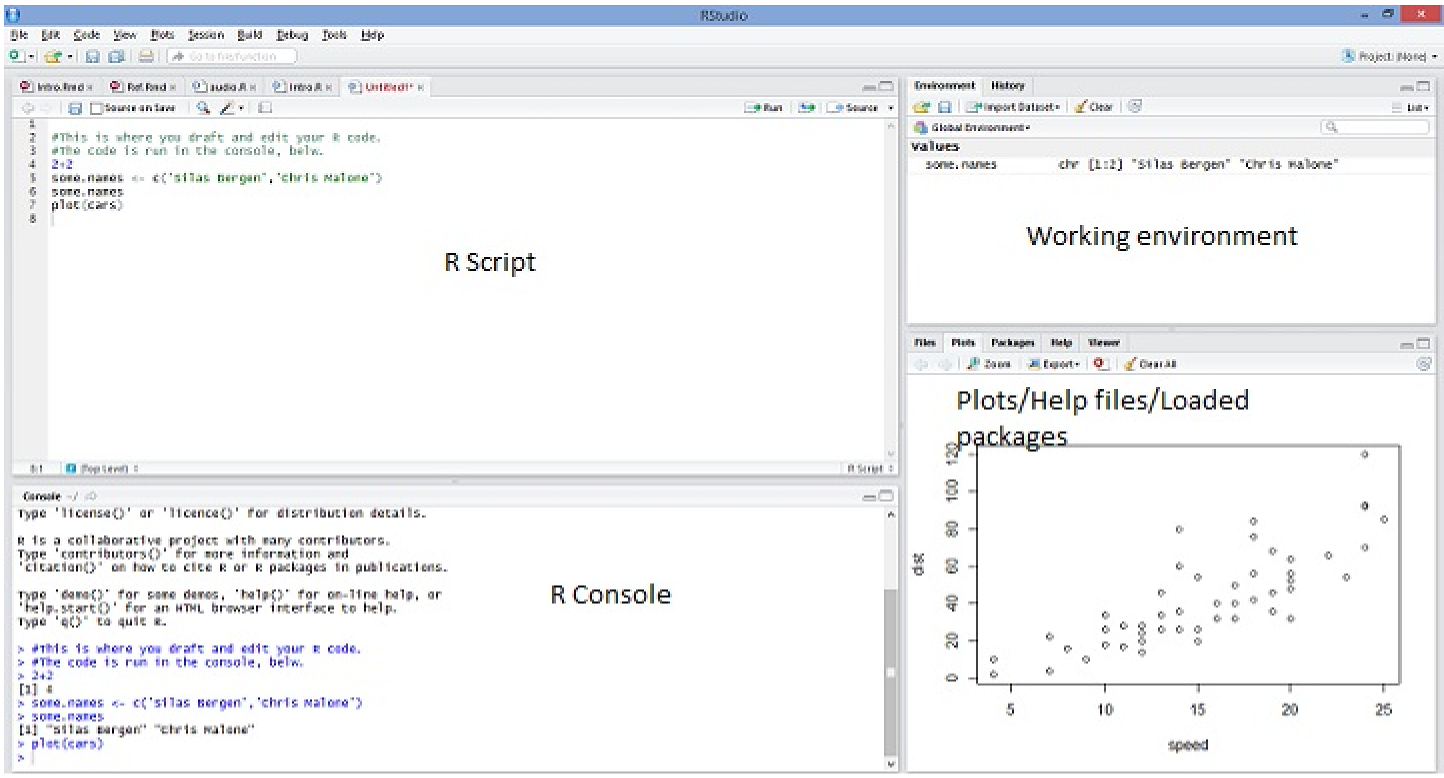
Introduction to R

This handout introduces you to R. R is a powerful open-source software that has many different statistical capabilities. We will assume no prior knowledge of R as we get started. We encourage you to install R Studio to work through this handout. R Studio is an elegant, powerful user interface for working with R. Although there is no difference in the coding syntax used between base R and R Studio, R Studio has the advantage in that it organizes your source code, console, graphs, and data much better than base R. You must first install R from R cran, and then install R Studio.

Assuming you have R Studio, take note of your environment. A screenshot is shown below. We highly encourage that you first open a new R source file, by clicking File -> New Script. This is where you can draft and edit your R code, and save it for future reference. Code is not run in the script, but is run instead in the console. You can run your code from the script by clicking "Run," or by pressing Ctrl + R. You can run multiple lines by highlighting them and then clicking Ctrl + R or clicking "Run."



```

#This is where you draft and edit your R code.
#The code is run in the console, below.
2+2
some.names <- c('Silas Bergen','Chris Malone')
some.names
plot(cars)

```

```
Values
  some.names        chr [1:2] "Silas Bergen" "Chris Malone"
```

```
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> #This is where you draft and edit your R code.
> #The code is run in the console, below.
> 2+2
[1] 4
> some.names <- c('Silas Bergen','Chris Malone')
> some.names
[1] "Silas Bergen" "Chris Malone"
> plot(cars)
> 
```

The power and flexibility of R is also part of its learning curve: there is no pointing-and-clicking; anything you want to do must be done by typing in appropriate code. To get the feel for this, type the following commands into the console, or copy-and-paste into your R script and run the lines from there:

```
1+1
10/5
sqrt(4)
log(10)
log10(10)
1+4*3
c(1,4,16)
```

R uses "object-oriented" language, which means you can assign names to different objects (like data sets, or variables) and R will store them in memory so that you can refer to those objects by name. The gist of object oriented programming is as follows: `object.name <- whatever.you.want.to.assign`. For example, run the following lines of code. Note that as soon as you assign an object, it shows up in your "environment", and you can now refer back to it and perform operations on it.

```
new <- c(1,4,9)
new
new + 1
sqrt(new)
new^2
newer <- c(-1,-4,-9)
new + newer
abs(newer)
######################################################
#I am adding hashes to break up this code.
#Note that hashes are how you can comment your code:
#R ignores any line that starts with a hash
#I will now compute the area of a circle of radius 2.
######################################################
radius <- 2
area <- pi*radius^2
area
##Boolean operations
new > 4
new >= 4
new == 4 #Note: need double == to specify equivalence.  Try just one =, what happens?
```

Another fundamental aspect of R is its pervasive use of functions. R has a host of functions that are automatically installed when you download R, and there are many, many more that you can access via installation of packages. We will stick to the functions included with the base version of R; there are plenty there for us to work with. The basic form of functions is `function.name(argument1, argument2, ...)`. Note that we have already used a few functions in the above code, including `sqrt()`, `abs()`, and `c()`. Anytime you want to know about a function, you can refer to its help file by typing `?function.name`. These help files vary in their clarity; often it is helpful to skip down to the bottom for examples: sometimes doing is better than reading! Sometimes you want to do something but don't know what the function name is. In these cases, Google what you want to do and add "in R" at the end. You will most likely find a thread on Stack Overflow or some other source that answers your question. Even as a proficient R user, hardly a session goes by without me Googling "how to do x in R".

Reading in data

Now it's time to read in data! R can easily read in data stored in .csv format. We will read in some birth weight data, containing information on 2500 mothers and infants in King

County, Washington. First you will need to make yourself aware of your working directory. This is where R will look for any data. Type `getwd()` at the prompt, and you'll see your working directory. You can set your working directory using the `setwd()` function. For example, `setwd('C:/Users/SBergen/Dropbox/USCOTS2015/')`. You can also use the "../" to back up one folder as you navigate to your new working directory. One you have set your working directory, type `getwd()` again to verify that your new path is correct.

Once you have set your working directory to where the data is stored, we will read in the birth weight data using the following code.

```
births <- read.csv('Birthweight.csv')
```

Note that the data set now appears in your global environment. Click on it to view it. The following commands also help get a feel for the data set:

```
#Viewing data
head(births)    #Shows the first 6 observations
tail(births)    #Shows the last 6 observations
names(births)   #Gives the names of the variables
dim(births)     #Shows the number of rows and columns
nrow(births)    #Shows the number of rows
ncol(births)    #Shows the number of columns
head(births[,2]) #births[,2] gives all observations in 2nd column
head(births[,c('age','race','parity')])
head(births[,2:4])  #Equivalent to above.
births[1,]
births[1,3]
births$race[1:5]
```

Note a few important things from the above code. First, the use of square brackets `[]` to subset elements of the data. Data sets in R are called "data.frames." Data frames have two dimensions: the number of rows, and the number of columns. We subset data by referring to the appropriate *indices*. If we leave an index unspecified, R returns all elements of that index. For example, `births[1,]` returns the first row, all columns. The second thing to note from the above code is that whenever we want to refer to a specific variable, we use `$` to call it.

Continue getting a feel for the data by running the following code.

```
summary(Births)
summary(births$age)
summary(births$age)[4]
mean(births$age)
min(births$age)
max(births$age)
sd(births$age)
var(births$age)
quantile(births$age,0.5)
quantile(births$age,c(0,0.25,0.5,0.75,1))
summary(births$race)
#Create a new variable
births$wtpost <- births$wpre + births$wgain
head(births) #Check to see that 'wtpost' is added


#Contingency tables:
table(births$race,births$married)
prop.table(table(births$race,births$married))
prop.table(table(births$race,births$married),1)
prop.table(table(births$race,births$married),2)
```

```
round(prop.table(table(births$race,births$married),1),2)
#What sort of table does each command create, and how are they different?
```

Practice: After running the above code, create a table that easily lets you ascertain whether the percent of mothers on welfare differs comparing married to unmarried mothers.

Exercise 1

By now you know more than enough to write some of your own code!

Consider the following, a sample of observations on incoming solar radiation at a greenhouse:

```
11.1 10.6 6.3 8.8 10.7 11.2 8.9 12.2
```

1. Assign the data to an object called solar.radiation.
2. Find the mean, median and variance of the radiation observations.
3. Add 10 to each observation of solar.radiation, and assign the result to sr10. Find the mean, median, and variance of sr10. Do the statistics change as expected?
4. Multiply each observation by 2, and assign the result to srm2. Find the mean, median, and variance of srm2. How do the statistics change now?
5. There are two formulas commonly used for the variance of a set of numbers: $\frac{\sum(x_i-\bar{x})^2}{n}$ and $\frac{\sum(x_i-\bar{x})^2}{n-1}$. Write code to calculate both forms of variances "from scratch" on the solar.radiation data, and compare to the output of the `var()` function. Which formula does the `var()` function in R use?

Key to Exercise 1

```
#1
solar.radiation <- c(11.1, 10.6, 6.3, 8.8, 10.7, 11.2, 8.9, 12.2)

#2
mean(solar.radiation)

## [1] 9.975

median(solar.radiation)

## [1] 10.65

var(solar.radiation)

## [1] 3.525

#3
sr10 <- solar.radiation + 10
sr10
```

```
## [1] 21.1 20.6 16.3 18.8 20.7 21.2 18.9 22.2

mean(sr10)

## [1] 19.975

median(sr10)

## [1] 20.65

var(sr10)

## [1] 3.525

#4
srm2 <- 2*solar.radiation
mean(srm2)

## [1] 19.95

median(srm2)

## [1] 21.3

var(srm2)

## [1] 14.1

#5
v1 <- sum( (solar.radiation-mean(solar.radiation))^2)/length(solar.radiation)
v2 <- sum( (solar.radiation-mean(solar.radiation))^2)/(length(solar.radiation)-1)
v1

## [1] 3.084375

v2

## [1] 3.525

var(solar.radiation)  #var() uses n-1 in the denominator

## [1] 3.525
```

Classes

Note that the `summary()` command gives different types of summaries for different variables, depending on the data type. Understanding how R thinks about different types of data is important; often the same R command will give different output. Run the following commands to see some of the most common R classes: `integer`, `numeric`, and `factor`. We can also change the class of a variable, for example if a binary variable is coded 0/1:

```
class(births$race)
levels(births$race)
class(births$age)
class(births$bwt)
class(births$firstep)
births$FstStep <- as.factor(births$firstep)
head(births)
class(births$FstStep)
head(births$FstStep)
```

Plotting data

R is a powerful tool for creating graphical data summaries. It can take some time to understand all the ins and outs of graphing in R. As with R in general, the level of flexibility and control a user has over creating R graphics also makes mastering graphics a lifelong learning experience!

Histograms

Run the following commands to graph the distribution of mothers' ages. Which histogram do you prefer? Note the use of the additional argments `xlab` and `main`, and `breaks` in the second line. Also note the use of the double-quotation marks for the main title. Why do we need them, here?

```
hist(births$age,xlab='Age',main="Histogram of mother's ages")
hist(births$age,xlab='Age',main="Histogram of mother's ages",breaks=30)
```

Scatterplots

There are an abundance of options for creating attractive scatterplots. The following code illustrates the use of the `col`, `pch`, and `cex` arguments. Try re-creating this plot, with some different point types, sizes, and colors. Make sure you change the legend to match! (Hint: see `?points`).

```
plot(bwt~gestation,data=births)
plot(bwt~gestation,data=births,
     xlab='Gestational Age (weeks)',
     ylab='Birth weight (grams)',
     main='Plot of birth weight by gestational age')


plot(bwt~gestation, data=births,
     xlab='Gestational Age (weeks)',
     ylab='Birth weight (grams)',
     main='Plot of birth weight by gestational age',
     col=c('black','red')[gender],
     pch=c(3,19)[gender],cex=1.5)


legend(x='topleft',legend=c('Male','Female'),pch=c(3,19),col=c('black','red'))
```

Often, you will want to add points or lines to an existing scatterplot. This can be done via the `points()` and `abline()` commands. Consider the following commands:

```
plot(bwt~gestation,data=births[births$gender=='M',],col='blue')
points(bwt~gestation,data=births[births$gender=='F',],col='red')
gest.F <- mean(births$gestation[births$gender=='F'])
gest.M <- mean(births$gestation[births$gender=='M'])
gest.F
gest.M
abline(v=c(gest.F,gest.M),col=c('red','blue'),lty=2,lwd=2)
```

Note the use of square brackets and boolean operations to subset data in the above code. Run some of the components of the above code separately, e.g. `births$gender=='F'`, to ensure you fully understand what each line is doing.

Practice: Add separate horizontal lines at the mean birth weight for males and females. Then, add an appropriate legend to the graph that describes what each line represents.

Boxplots

Run the following R code to create boxplots of birth weight by race. Note the use of the `boxwex` argument: which boxplot do you prefer?

```
#Quantitative by categorical:
boxplot(bwt~race,data=births)
boxplot(bwt~race,data=births,boxwex=.3) #What does boxwex do?
```

Bar graphs

Bar graphs take slightly more work to create than the previous graphs we've considered. Look at the help page by typing `?barplot`. Note that the `height` argument requires the categorical variable to be summarized before plotting. Thus, the best results will come from using a table as the primary argument to `barplot()`. Consider the following, which yields bar graphs for the `married` and `race` variables:

```
barplot(births$married) #Doesn't work!
table(births$married)
barplot(table(births$married))
barplot(table(births$race))
```

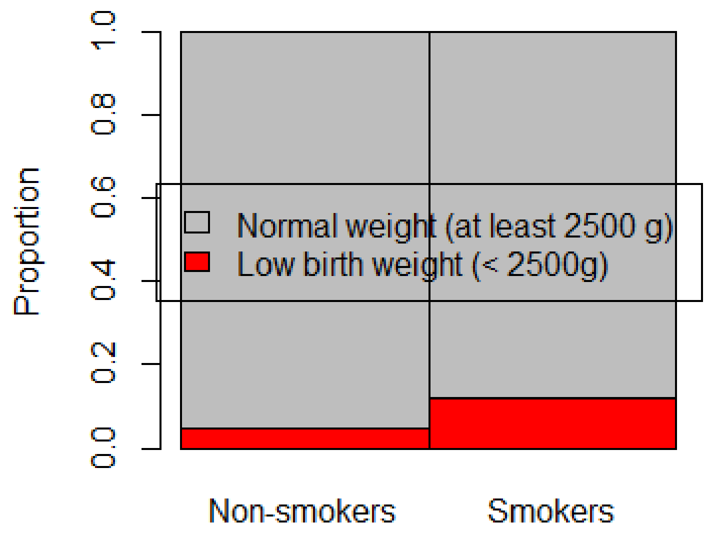
The kind of table passed to `barplot()` determines the y-axis of the bar graph. We considered the use of the `prop.table()` function earlier; we can employ it again here. Consider the following four bar graphs. If the intent of the graph is to examine association between race and marital status, which is best?

```
par(mfrow=c(2,2),mar=c(4,4,1,0))
#Barplot 1
barplot(table(births$married,births$race),
          beside=TRUE,legend=TRUE,
          args.legend=list(x='top'),cex.names=.8)
#Barplot 2
barplot(prop.table(table(births$married,births$race),1),
        beside=TRUE,legend=TRUE,
        args.legend=list(x='top'),ylab='Proportion',cex.names=.8)
#Barplot 3
barplot(prop.table(table(births$married,births$race),2),
        beside=TRUE,legend=TRUE,
        args.legend=list(x='top'),ylab='Proportion',cex.names=.8)
#Barplot 4
barplot(prop.table(table(births$married,births$race),2),
        beside=FALSE,legend=TRUE,
        args.legend=list(x='top'),ylab='Proportion',cex.names=.8)
```

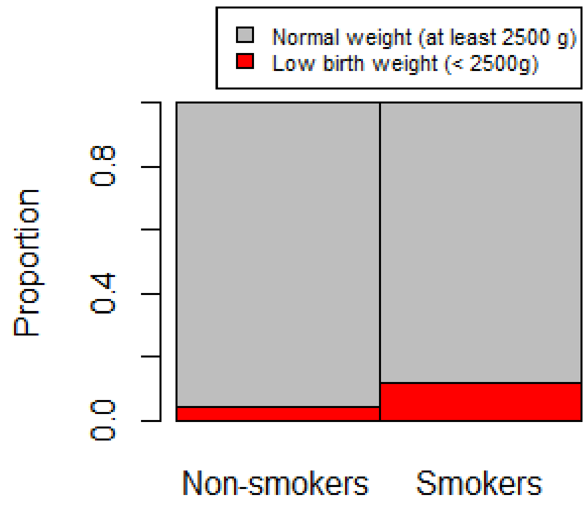
Note the use of the `par()` command in the above code. All options for graphs in base R are implemented via `par()`, and any personal graph designs almost always come by way of this command. Understanding all the options comes with time and lots of Stack Overflow searches, as well as trial and error. Try editing the `mar` and `mfrow` arguments, and note how the output changes. Also note the use of the `cex.names` argument. Try eliminating this argument and re-running the code. Why do we need it?

Exercise 2

Create the following graph. Hint: you may find the function `ifelse()` useful. Try to use the `?barplot` help page rather than looking at the key code.



Optional: edit the code to put the legend above the graph, as follows:



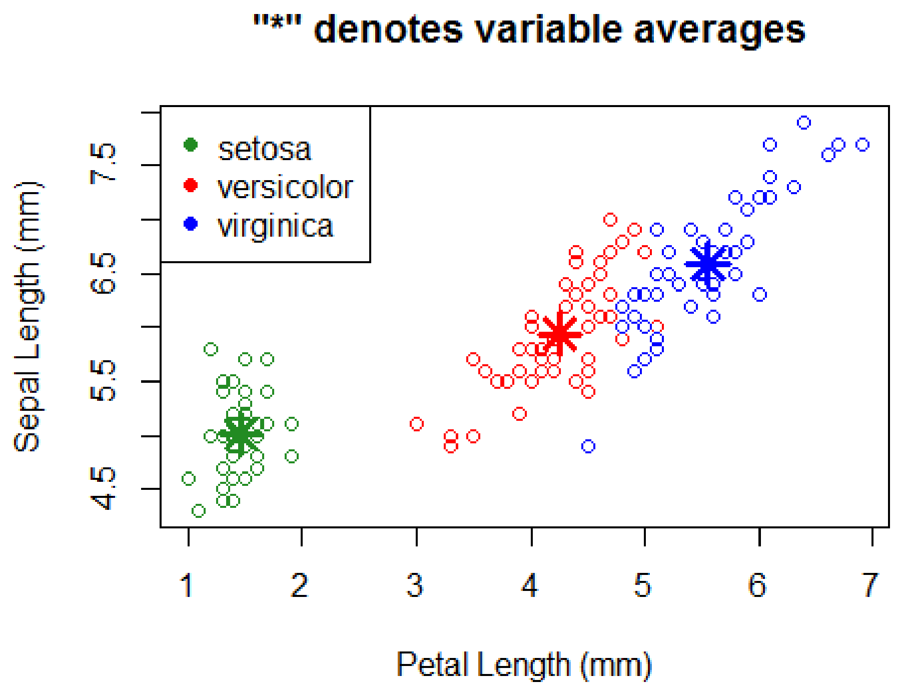
Key to Exercise 2

```r
births$lbw <- ifelse(births$bwt<2500,'LBW','Normal')
my.tab <- prop.table(table(births$lbw,births$smoker),2)
barplot(my.tab,names.arg=c('Non-smokers','Smokers'),
          legend.text=c('Low birth weight (< 2500g)','Normal weight (at least 2500 g)'),
          args.legend=c(x='center'),ylab='Proportion',
          col=c('red','grey'),space=0)

barplot(my.tab,names.arg=c('Non-smokers','Smokers'),
          legend.text=c('Low birth weight (< 2500g)','Normal weight (at least 2500 g)'),
          args.legend=c(x=2,y=1.3,cex=.7),ylab='Proportion',
          col=c('red','grey'),space=0,xpd=NA)  #Note: need to use xpd=NA; see ?par
```

Exercise 3

The `iris` data are available in R with the basic installation. Type `data(iris)` and `head(iris)`, to take a quick look at the format. Using this data set, create the following graph:



Key to Exercise 3

```r
plot(iris$Petal.Length,iris$Sepal.Length,
      col=c('forestgreen','red','blue')[iris$Species],
      xlab='Petal Length (mm)',ylab='Sepal Length (mm)',
      main=' "*" denotes variable averages')
legend('topleft',legend=levels(iris$Species),col=c('forestgreen','red','blue'),pch=19)
petal.means <- by(iris$Petal.Length,iris$Species,mean)
#by() is a quick way to get group-specific summaries; we will discuss this in the next handout.
sepal.means <- by(iris$Sepal.Length,iris$Species,mean)
points(petal.means,sepal.means,pch=8,cex=2,col=c('forestgreen','red','blue'),lwd=3)
```

Statistical tests

In this section we discuss how to carry out statistical tests in R.

The t-test

Suppose we want to carry out the 2-sample t-test to test whether the mean birth weights of infants born to smokers is different from the mean birth weight of infants born to non-smokers. This is easily done using the `t.test()` function. Examine the following code. Note that we can assign the output from `t.test()` to an object (I call it `mytest` here) so we can easily access different parts of the output.

```
t.test(bwt~smoker,data=births)
names(t.test(bwt~smoker,data=births))
mytest <- t.test(bwt~smoker,data=births)
mytest$p.value
mytest$conf.int
mytest$statistic
#Get the p-value from scratch:
pt(mytest$statistic,df=198.331,lower.tail=FALSE)*2
```

The χ^2 test of association

Suppose we want to determine whether there is a statistically significant association between race and marital status. We explored this relationship graphically earlier in this handout, but now we will conduct a formal test. We start with the contingency table of this relationship, then use the `chisq.test()` function:

```
mytab <- table(births$race,births$married)
chisq.test(mytab)
```

Exercise 4

Write code to calculate the chi-square statistic and the p-value from scratch. Recall:

$$\chi^2 = \sum_{all\ cells} \frac{(observed - expected)^2}{expected}.$$

Hint: assign the output of `chisq.test()` to an object.

Key to Exercise 4

```
my.chisq <- chisq.test(mytab)
chi2 <- sum((my.chisq$observed-my.chisq$expected)^2/my.chisq$expected)
pchisq(chi2,df=4,lower.tail=FALSE)
```

Exercise 5

Nationally, 8% of infants are low birth weight. Carry out the binomial exact test to determine whether there is statistically significant evidence that the percent of low birth weight infants among smokers exceeds 8%. Compare your answer to the normal approximation. Use `binom.test()` to carry out the binomial exact test, then write code to carry out this test "from scratch" using `pbinom()`. Use `prop.test()` for the normal approximation, then write code to carry out this test "from scratch" using `pnorm()`.

Key to Exercise 5

```
table(births$smoker,births$lbw)
x <- 21
n <- 175

##Binomial exact test:
#Using binom.test():
binom.test(x,n,0.08,alternative='greater')
#From "scratch":
pbinom(x-1,size=n,prob=0.08,lower.tail=FALSE) #NOTE: Why x-1?  See ?pbinom.

##Normal approximation:
#Using prop.test():
prop.test(21,175,p=0.08,alt='greater',correct=FALSE)
#From scratch:
phat <- x/n
se <- sqrt(phat*(1-phat)/n)
pnorm(phat,mean=0.08,sd=se,lower.tail=FALSE)  #Note: Different p-value from prop.test(), which
uses p0 to calculate SE.  Verify this.
```

Linear regression models

Consider the following models of birth weight on gestational age. Model 1 is a simple linear regression; Model 2 allows for different intercepts for gender; and Model 3 allows for separate intercepts and slopes by gender. I.e.,

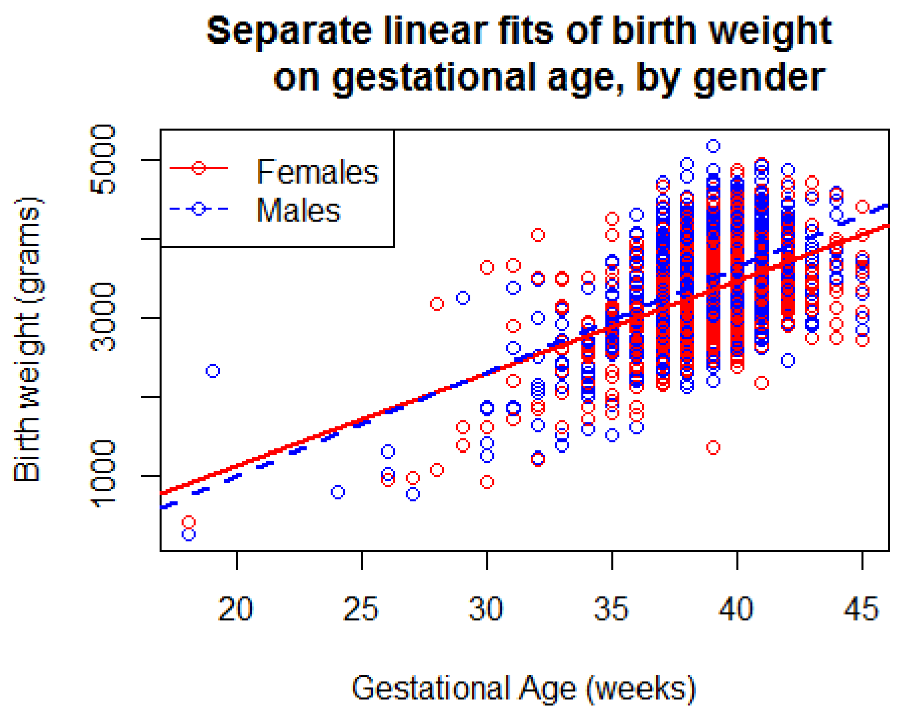
-Model 1: $Bwt = \beta_0 + \beta_1 * Gest$
-Model 2: $Bwt = \beta_0 + \beta_1 * Gest + \beta_2 * I(Gender = M)$
-Model 3: $Bwt = \beta_0 + \beta_1 * Gest + \beta_2 * I(Gender = M) + \beta_3 * Gest * I(Gender = M)$

Fit these models, and investigate the resulting output.

```
#Simple regression model:
mod1 <- lm(bwt~gestation,data=births)
summary(mod1)
names(mod1)
plot(bwt~gestation,data=births,
     xlab='Gestational Age (weeks)',ylab='Birth weight (grams)',
     main='Plot of birth weight by gestational age')
abline(a = mod1[1], b = mod1[2])
#Different intercepts by gender:
mod2 <- lm(bwt~gestation+gender,data=births)
summary(mod2)
#Different intercepts and slopes by gender:
mod3 <- lm(bwt~gestation+gender+gestation*gender,data=births)
summary(mod3)
```

Exercise 6

Create the following plot:



Key to Exercise 6

```
plot(bwt~gestation,data=births,
     xlab='Gestational Age (weeks)',
     ylab='Birth weight (grams)',
     main='Separate linear fits of birth weight
     on gestational age, by gender',col=c('red','blue')[gender])
abline(a=mod3$coef[1],mod3$coef[2],col='red',lwd=2)
abline(a=mod3$coef[1]+mod3$coef[3],mod3$coef[2]+mod3$coef[4],col='blue',lwd=2,lty=2)
legend('topleft',c('Females','Males'),col=c('red','blue'),pch=1,lty=c(1,2))
```