Part I: Getting Data into R

1. Download the esoph.csv dataset from learnr.web.unc.edu
2. Open the dataset in excel and a text editor like notepad to see what the actual file looks like. Make sure not to save it (excel, trying to be helpful, will ask to convert it to a .xls).
3. Put the file somewhere on your computer and note the filepath (example: C:/Users/Me/Desktop)
4. Use the setwd() function to set your working directory to that location. Make sure to use forward slashes (on the ? key); backslashes are actually escape characters for entering non-standard text.
5. Check your working directory with getwd()
6. Import the csv with the esoph <- read.csv(“esoph.csv”)

Part II: Using a package to reshape data.

1. Load the built-in esoph dataset using the data() command
2. “attach” to the esoph dataset using the attach() command

Once attached, you can reference variables in the dataset without needing $. For example, agegp will point to esoph$agegp.

1. Using some combination of names(), head(), ?, class(), levels(), sum() answer the following questions:
	1. What is the study design?
	2. What are the exposure variables? How many levels does each have?
	3. How many cases and how many controls are there?

Epidemiologists often like to work with datasets in which one row corresponds to one person. Let’s use a package that does this for us.

1. Install the reshape package using the menu.
2. Load the reshape package using the library() function.
3. Look up the help page for the untable() function.
4. Use the untable() function to create an object called cases that contains one row per case. Use dim() on this object to confirm it contains the number of observations you expect. Make a new variable in this dataset called outcome and assign its value to 1.
5. Use the untable() function to create an object called controls that contains one row per control. Use dim() on this object to confirm it contains the number of observations you expect. Make a new variable in this dataset called outcome and assign its value to 0.
6. Row bind your cases and controls objects into a new object (e.g. esoph2) with the rbind() command. Confirm that this object has the correct number of observations, cases, and controls. If everything looks okay, drop the ncases and ncontrols variables from the esoph2 dataset by overwriting them with the NULL value.

Part II: Working with Factors

1. attach() to your new esoph2 dataset to make things easier.
2. Make a plot() of the three factors in your new esoph2 dataset. Use ? to find the arguments to plot do the following (hint: click through to plot.default and look at the arguments there):
	1. Add a main title
	2. Add a y-axis label
	3. Make the bar color blue
3. Create a table object called tobtab containing the number of cases and controls in each tobacco group (hint: use table() with two arguments).
	1. Evaluate tobtab to print the table
	2. Evaluate prop.table(tobtab) to get percents (relative to the whole table).
	3. Specity margin=1 or margin=2 in prop.table so that the percents for each tobacco group add up to 100.
	4. Save the result of your prop.table to a new object called tobtabp
	5. Extract the row (or column, depending on how you built the table) corresponding to the proportion of each tobacco group that are cases. Use this as an argument to barplot() to create a graph like this:



This makes a nice graph, but it isn’t exactly fair because cases and controls are likely being sampled at different frequencies. Let’s try using the famous 2x2 table, which will compare the exposure profiles of cases and controls.

1. Install the epiR package and load it with the library() function
2. Prepare a 2x2 table for the epi.2by2() function using the table command and a binary classification scheme. For instance:

tab <- table(tobgp %in% c(“20-29”,”30+”),outcome)

1. Overwrite rownames(tab) to something more informative, e.g. c(“low”,”high”)
2. Use epi.2by2() with method=”case.control” on your data table. This function expects exposure in rows and outcome in columns, with outcome-positive individuals with the exposure in the top left. If you need to reverse columns, rows, or both, use something like the following:
	1. tab[c(2,1), ]
	2. tab[ ,c(2,1]
	3. tab[c(2,1), c(2,1)]

This is just one example of a package for computing odds ratios. There are plenty of other packages if you don’t like the epiR syntax: abd, bstats, etc. Eventually you’ll use logistic regression with the built-in glm() function (next time!).

Part III: Using a list

Often we want to inspect the values of some function aggregated across categories. For instance, suppose we want to know how many cases and controls are in each tobacco/alcohol group in our esoph2 dataset. Or we want to know the average alcohol consumption in each age group. This is easy to do in R with the aggregate() function.

1. Create a new numeric variable called alc in your esoph2 dataset with all missing values (assign the value NA).

2. Replace the value of alc with the midpoint of each range. For example, to set the value of alc to 20 for all observations in the 0-39g group:

esoph2$alc[esoph2$alcgp==”0-39g/day”] <- 20

For the 120+ group, just assign 120.

3. The following example will compute the number of cases in each alcohol/tobacco group:

aggregate(esoph2$outcome,

 by=list(esoph2$tobgp,esoph2$alcgp),

 FUN=sum)

Adapt this code to calculate the mean and standard deviation of your new alc variable in each age group.

Part IV: Writing Functions

Writing functions can help you repeat common tasks.

1. Enter and run the function below. R won't say anything, but if there are no errors, you're good.

 norm.plot <- function(mean,sd) {

 x <- rnorm(mean=mean,sd=sd,n=1000)

 plot(density(x))

 }

This function lacks return(), so it won't give you anything back (much like the plot() it calls).

You can now run norm.plot() to make a normal plot.

2. Adapt this function to a new function called add.plot that calls lines() instead of plot()

lines() will draw on top of an existing plot instead of opening a new plot. You'll get an error if you try to run lines() without a plot open!

3. Use your norm.plot() and add.plot() functions to plot the mean and standard deviations for alcohol consumption from the previous questions on the same plot.

Functions are also useful for “black-boxing” steps in your code and returning a value. Adapt the code below (which does the first step, making a table of data) to black-box the code you wrote to calculate the odds ratio using epi.2by2().

oddsratio <- function(exposure,outcome) {

 tab <- table(exposure,outcome)

 return(tab)

}