Empirical Reasoning Center
R Workshop (Summer 2017)
Session 1

This guide reviews the examples we will cover in today's workshop. It should be a helpful introduction to R, but for more details, the ERC will offer a more extensive R user guide.

1 Writing and executing code in R

Although you can write and execute R code at the command line on the R console, a better option is to write your code in a script that you can save and modify as necessary. To start a new script, access the *File* menu, chose *New File*, and then choose *R Script*.

To execute the code in a script, you can select and run specific lines of code by highlighting the code and either clicking the *Run* icon in RStudio (above the script in the source pane) or using the keyboard shortcut Ctrl+Enter (Windows) or Command+Enter (Mac). You can also run all or part of a script line by line without selecting code: if you click the *Run* icon or use the appropriate keyboard shortcut, R will execute only the line of code where your cursor is located and move the cursor to the next line.

You can use multiple lines to write one command or function. In general, no special characters are required to indicate that a function continues onto the next line. However, you will need to run all of the lines of a function to execute it. You can highlight the entire multi-line function and run the block of code, or you can run one block at a time.

2 Performing Basic Tasks

2.1 Using file paths

File paths are a key to using R for data analysis. A file path specifies the location of a file on your computer. For example, you might notice that when you download a file from the internet, your computer probably adds the file to your "Downloads" folder. Generally, whether you use a Mac or a PC, you have your files stored in folders (and folders within folders). You may have file icons or shortcuts on your desktop, but even those files are stored in a folder—perhaps a "Desktop" folder.

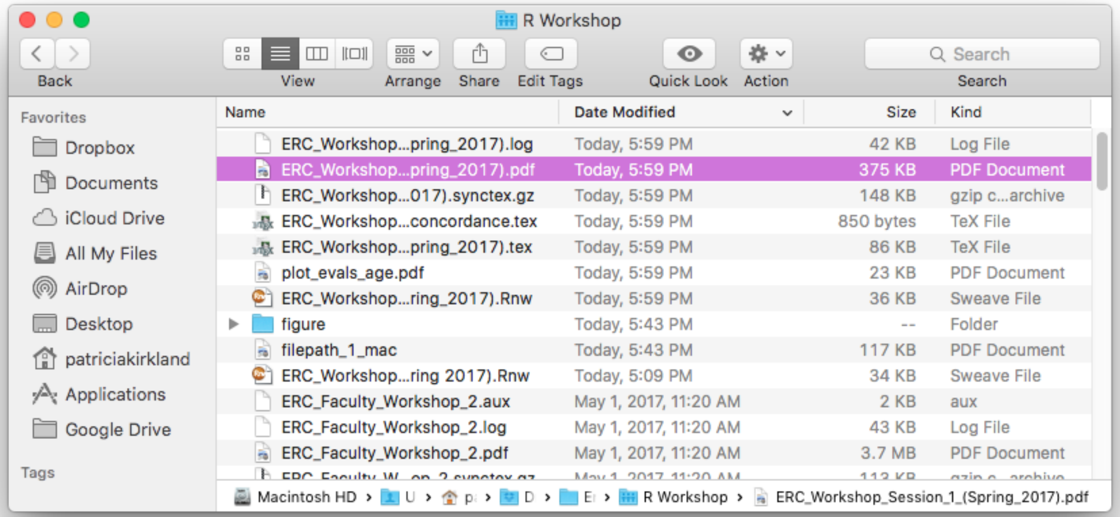
Why are file paths so important when you use R (and many other applications)? You will want to import or load data, and you will need to use a file path to specify where R should access the file. You may also want to save or export data, tables, and figures, so you will need to specify where R should locate these files.

How do you determine the file path? There are multiple ways to figure out a file path. For example, you may know where a file is located—perhaps you saved it to your "Downloads" folder. Your file path might look something like this: /Users/YourName/Downloads (Mac) or C:\Users\Downloads (PC).

You can see a Mac example below (a Windows example follows). Figure 1 shows a Finder window on a Mac. To access Finder, click the Finder icon (Finder is always open and its icon is always in your

Dock). There are a few things to notice here. First, a file for the ERC Workshop is highlighted. You can see from the top of the window that the file is inside a folder called "R Workshop," and at the bottom of the window, you can see that this folder is nested in a series of folders.

Figure 1: Example—Mac Finder Window

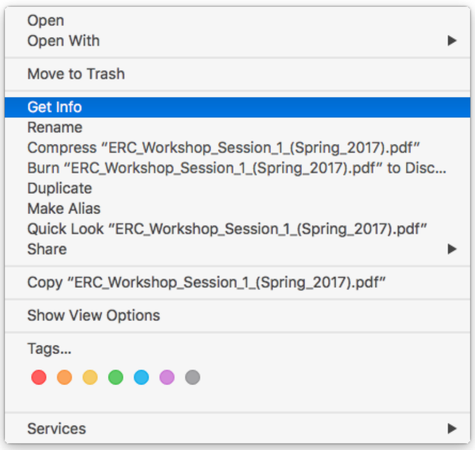


Name	Date Modified	Size	Kind
ERC_Workshop...pring_2017).log	Today, 5:59 PM	42 KB	Log File
ERC_Workshop...pring_2017).pdf	Today, 5:59 PM	375 KB	PDF Document
ERC_Workshop...017).synctex.gz	Today, 5:59 PM	148 KB	gzip c...archive
ERC_Workshop...concordance.tex	Today, 5:59 PM	850 bytes	TeX File
ERC_Workshop...pring_2017).tex	Today, 5:59 PM	86 KB	TeX File
plot_evals_age.pdf	Today, 5:59 PM	23 KB	PDF Document
ERC_Workshop...ring_2017).Rnw	Today, 5:59 PM	36 KB	Sweave File
figure	Today, 5:43 PM	--	Folder
filepath_1_mac	Today, 5:43 PM	117 KB	PDF Document
ERC_Workshop...ring 2017).Rnw	Today, 5:09 PM	34 KB	Sweave File
ERC_Faculty_Workshop_2.aux	May 1, 2017, 11:20 AM	2 KB	aux
ERC_Faculty_Workshop_2.log	May 1, 2017, 11:20 AM	43 KB	Log File
ERC_Faculty_Workshop_2.pdf	May 1, 2017, 11:20 AM	3.7 MB	PDF Document
ERC_Faculty_W...op_2.synctex.gz	May 1, 2017, 11:20 AM	113 KB	gzip c...archive

Macintosh HD › U › p › D › E › R Workshop › ERC_Workshop_Session_1_(Spring_2017).pdf

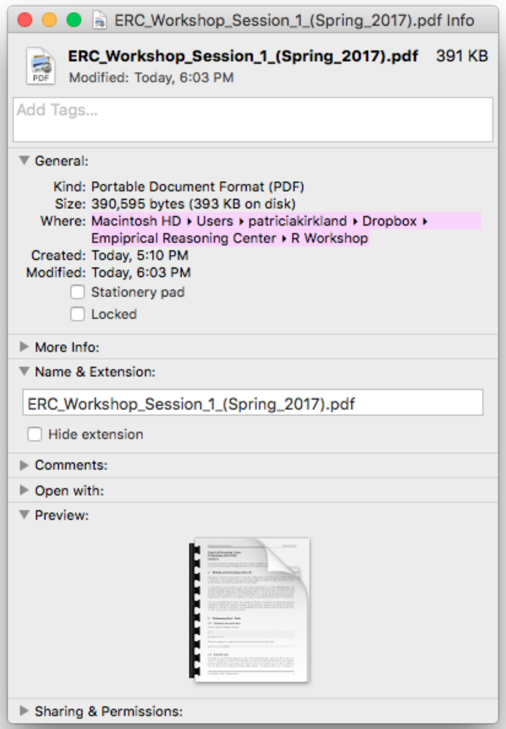
One easy way to find the file path for this document is to right click (or secondary click—you often do this on a Mac by clicking with two fingers on your trackpad) on the file. You will see a menu like the one in Figure 2. Select "Get Info" for details on the file.

Figure 2: Example—File Menu


Open
Open With
Move to Trash
Get Info
Rename
Compress "ERC_Workshop_Session_1_(Spring_2017).pdf"
Burn "ERC_Workshop_Session_1_(Spring_2017).pdf" to Disc...
Duplicate
Make Alias
Quick Look "ERC_Workshop_Session_1_(Spring_2017).pdf"
Share
Copy "ERC_Workshop_Session_1_(Spring_2017).pdf"
Show View Options
Tags...
Services

Note that in the example in Figure 3, all of the information in the "Where:" field is highlighted—this is the file path. You can highlight this information, copy it, and paste it into an R script. Your computer will copy only the portion of the file path that you need, and it should automatically paste the file path in a readable format, i.e., the arrows in the "Get Info" window will be replaced by slashes (If you have an older Mac OS, you might have to change the arrows to slashes manually).

Figure 3: Example—Mac Get Info



ERC_Workshop_Session_1_(Spring_2017).pdf Info

ERC_Workshop_Session_1_(Spring_2017).pdf 391 KB
Modified: Today, 6:03 PM

Add Tags...

General:

Kind: Portable Document Format (PDF)
Size: 390,595 bytes (393 KB on disk)
Where: Macintosh HD ▸ Users ▸ patriciakirkland ▸ Dropbox ▸ Empirical Reasoning Center ▸ R Workshop
Created: Today, 5:10 PM
Modified: Today, 6:03 PM
☐ Stationery pad
☐ Locked

More Info:

Name & Extension:

ERC_Workshop_Session_1_(Spring_2017).pdf
☐ Hide extension

Comments:

Open with:

Preview:

Sharing & Permissions:

Finding the file path on a Windows system is similar. Figure 4 shows a File Explorer window. You can access File Explorer from the taskbar or via the Start menu. The name of the current folder is displayed at the very top of the window. Just under the window's pull-down menus, you will see the location of the current folder on your computer (it's highlighted in pink in Figure 4). This information is the file path for the folder displayed in the window. If you click on the file path, you will see it change to a format with backslashes instead of arrows, and you can copy the file path from here.

Note that you will need to modify the Windows file path to use it in R. Windows file paths include backslashes (\), which you will need to change. This modification is necessary because the backslash (\) is a character that has a designated meaning in R. [1] There are two ways to resolve this minor issue:

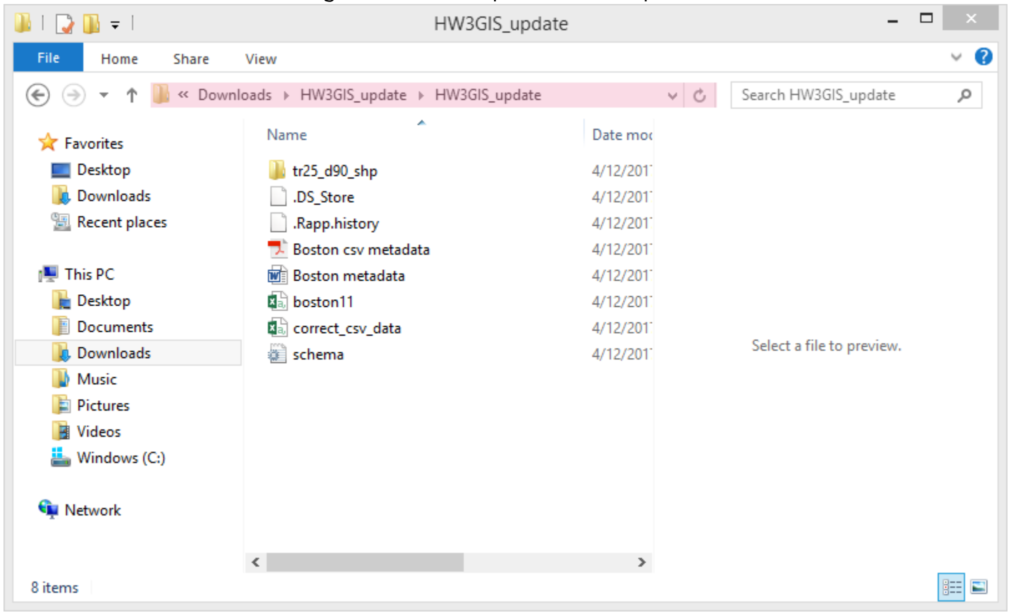
1. You can simply delete the backslashes and replace them with forward slashes

   C:\Users\Downloads ⇒ C:/Users/Downloads

2. You can add a second backslash directly before or after each existing backslash

   C:\Users\Downloads ⇒ C:\\Users\\Downloads

Figure 4: Example—File Explorer

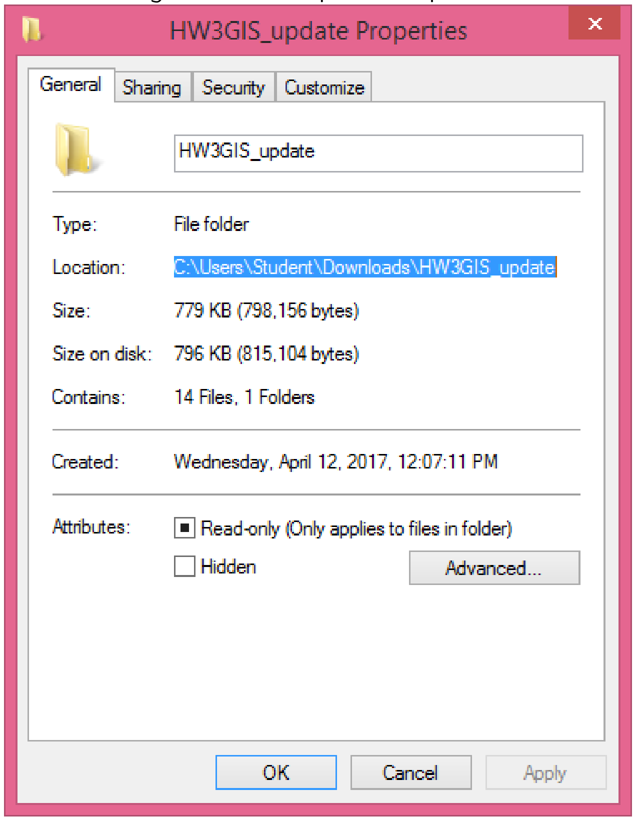


Name	Date moc
tr25_d90_shp	4/12/201
.DS_Store	4/12/201
.Rapp.history	4/12/201
Boston csv metadata	4/12/201
Boston metadata	4/12/201
boston11	4/12/201
correct_csv_data	4/12/201
schema	4/12/201

Another way to find the file path in Windows is to right-click on a file and select "Properties." Figure 5 shows an example of a "Properties" window. Note the "Location" field, which is the file path for the file or folder. You can copy and paste the file path from here as well. You will still need to either replace the backslashes (\) with forward slashes (/) or add an additional backslash next to each backslash in the file path.

[1] In R, the backslash (\) functions as an escape character—you can check the internet for more information.

Figure 5: Example—Properties


HW3GIS_update Properties

General | Sharing | Security | Customize

HW3GIS_update

Type: File folder

Location: C:\Users\Student\Downloads\HW3GIS_update

Size: 779 KB (798,156 bytes)

Size on disk: 796 KB (815,104 bytes)

Contains: 14 Files, 1 Folders

Created: Wednesday, April 12, 2017, 12:07:11 PM

Attributes: ■ Read-only (Only applies to files in folder)

☐ Hidden

2.2 Workflow & Organization

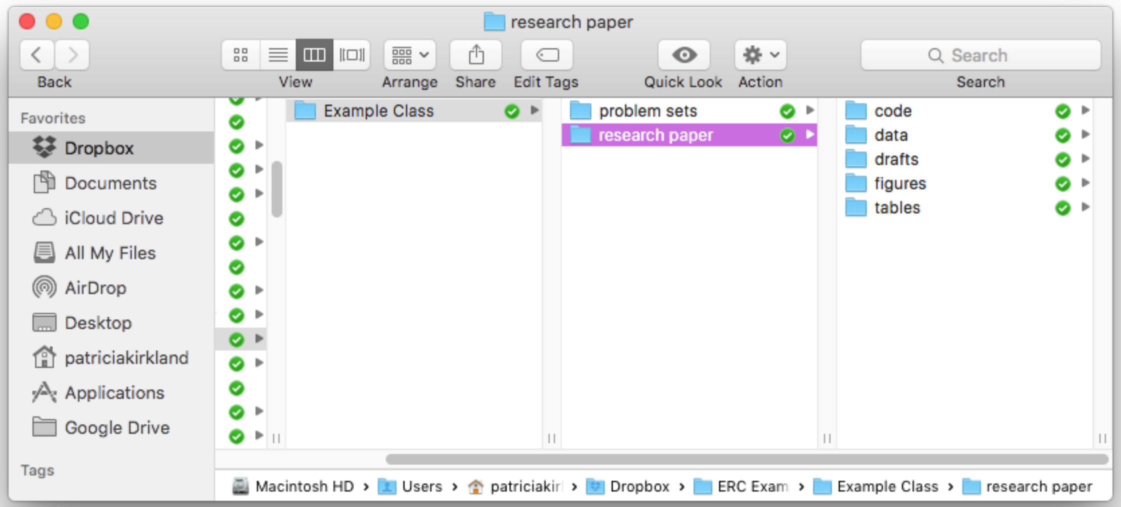
R is a powerful tool for manipulating, analyzing, and visualizing data. One convenient feature of R is its ability to export tables and figures that you can use in assignments and projects. File paths are essential to working in R because you must use them to load or import data and to save or export data and other types of files, including tables and figures. It is helpful to think about how you might organize the files on your computer to maximize the efficiency and benefits of using R. If you organize all of your files in the same way, you will likely find it much easier to specify file paths. Developing a system for organizing your files will probably make it easier to find things on your computer.

Whether and how you organize your files is up to you, but one option to consider is creating a folder for each class, each assignment, or each project you will be working on. Figure 6 shows one example you might find helpful. Here, you can see a folder for "Example Class," which contains folders for problem sets and a research paper. Within the research paper folder, there are separate folders for code, data, drafts, figures, and tables. The code folder should contain only the R scripts for the paper, while the data file will hold all of the data files. The drafts folder should include drafts of the paper, and the figures and tables folders will hold files to be included in the paper.

2.3 Setting up your work space

See the objects currently in memory

Figure 6: Example—Organizing Files



```
ls()

## character(0)
```

Clear your workspace– many R users include this as the first line in any script

```
rm(list=ls(all=TRUE))
```

2.4 Working Directory

You can use the file path each time you want to use or create a file with R, but you can also set a working directory. Setting the working directory involves designating a file path for the location on your computer where R will access and save files. You can see your working directory, and you can set your working directory. You can set a working directory at the start of each R script to use files for specific projects or assignments.

```
getwd()

setwd("/Users/Your Name/Folder/Sub-folder")   ## add your own file path

getwd()          # check again
```

2.5 Installing and loading packages

Because R is an open-source application, many users develop packages to handle specific tasks. For example, there are packages for data visualization, data manipulation, and many types of statistical analyses. You will need to install packages to handle certain tasks. You only need to install packages once, but you will need to load them any time you want to use them.

To install packages:

```
## use dependencies = TRUE to install any other required packages
install.packages("dplyr", dependencies=TRUE)
install.packages("ggplot2", dependencies=TRUE)
install.packages("foreign", dependencies=TRUE)
install.packages("xtable", dependencies = TRUE)
install.packages("stargazer", dependencies = TRUE)
install.packages("arm", dependencies = TRUE)
install.packages("modeest", dependencies = TRUE)
install.packages("lmtest", dependencies = TRUE)
install.packages("sandwich", dependencies = TRUE)
install.packages("descr", dependencies = TRUE)
```

To load packages:

```
library(foreign)
library(xtable)
library(arm)
library(ggplot2)
library(dplyr)
library(stargazer)
library(modeest)
library(lmtest)
library(sandwich)
library(descr)
```

Some useful packages:

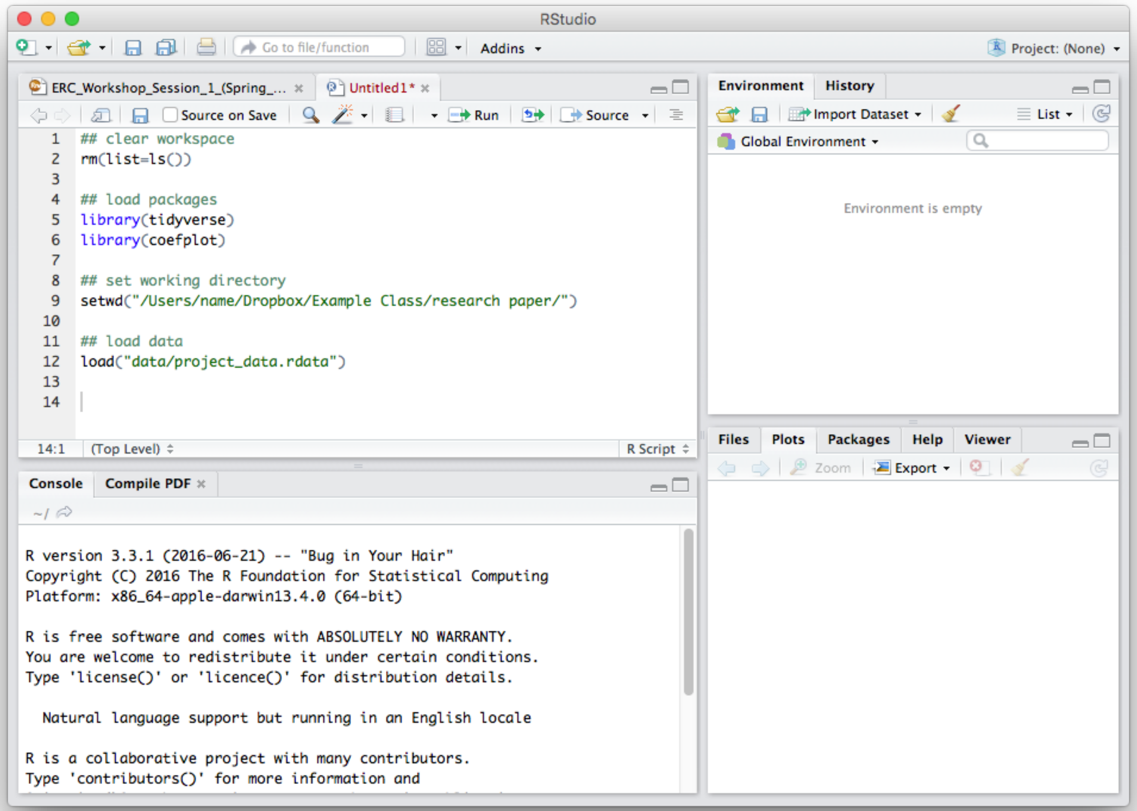
- foreign – load data formatted for other software
- xtable – export code to produce tables in LaTeX
- arm – applied regression and multi-level modeling
- ggplot2 – make plots and figures
- dplyr – user-friendly data cleaning & manipulation
- descr – produce frequency tables & crosstabs
- more packages: http://cran.r-project.org/web/packages/

2.6 Getting started

Each time you begin an analysis or project in R, you will want to create a script. To start a new script, access the *File* menu, chose *New File*, and then choose *R Script*. Most R users start each new script the same way... Figure 7 shows an example.

1. clear all objects in memory
2. load packages
3. set working directory
4. load or import data

Figure 7: Example—Creating a new script



```
## clear workspace
rm(list=ls())

## load packages
library(tidyverse)
library(coefplot)

## set working directory
setwd("/Users/name/Dropbox/Example Class/research paper/")

## load data
load("data/project_data.rdata")

```

```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
```

2.7 A few programming basics

Basic calculations

You can use R for simple (and not so simple) calculations.

```
4

## [1] 4

"yes"

## [1] "yes"
```

```
2+3

## [1] 5

1039/49

## [1] 21.20408

46^700

## [1] Inf

(3.5+2.7)/(900*2)

## [1] 0.003444444
```

Assignment operator
There are two operators that assign values to objects. Most R users opt to use the $<-$ operator, but you can also use $=$. One reason the $<-$ operator tends to be preferred is that unlike the equal sign, the $<-$ operator serves only one purpose. For the sake of clarity, we will use the $<-$ assignment operator exclusively.

```
x <- 3
x

## [1] 3

y <- "this is a string"
y

## [1] "this is a string"

z <- 2
z

## [1] 2

x+z

## [1] 5
```

Comments
Within an R script, you can use the hash sign (#) to designate text as a comment. Any text that follows # will be ignored when R executes a script. This feature enables you to annotate your code, and it can also be helpful for debugging code. You can "comment out" (or un-comment) a line of code using Ctrl+Shift+C (Windows) or Command+Shift+C (Mac) with your cursor anywhere on the line. You can highlight multiple lines and use the same shortcut to comment out a block of code.

```
n <- 100
n

## [1] 100

# n

# n <- 1
n

## [1] 100
```

HELP
If you need help with a specific function and know the name of the function, enter `?` followed by the function name at the command line—e.g., `?table` for information about the `table()` function. You can also search more generally for a topic by entering `??` followed by a search term. Using `??` can help you find a function to perform a specific task. For example, `??variance` will bring up search results for a variety of functions from different packages, allowing you to choose the option you want.

```
?table

??variance
```

Another option for finding help with R is Google. If you have a question about a function, a task, or an error in R, search for guidance on the internet using Google (or your preferred search engine). There are many sites that offer instructions as well as sites where users post questions and answers. Do not hesitate to ask the internet your R questions—you can be sure that advanced R users do. In fact, if you come to the ERC for assistance, do not be surprised if we suggest or access Google during your visit. As you can see from the example in Figure 8, a Google search for the `table()` function returns many helpful options.

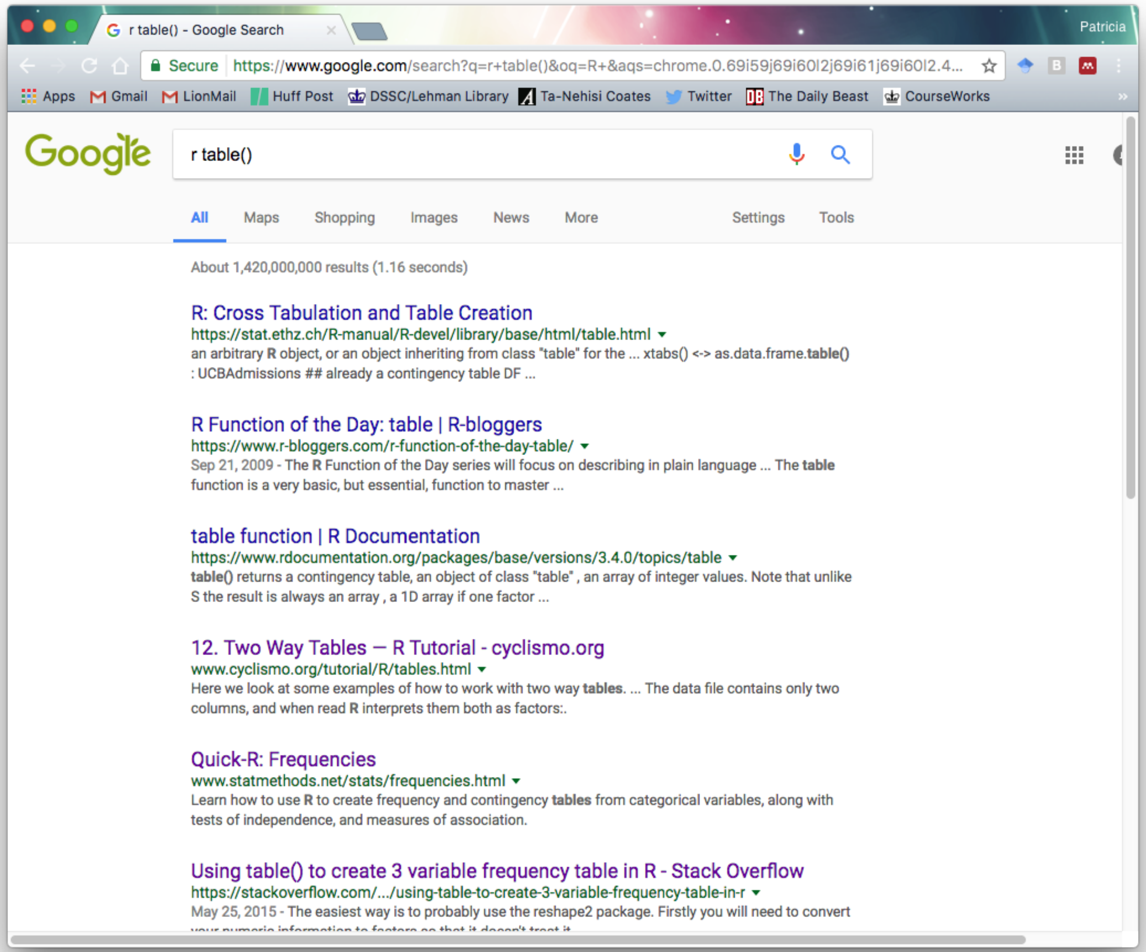
One resource you will see often is Stack Overflow, a site where users post questions and answers. This site cover a tremendous amount of material for multiple applications, and users provide comments and "vote up" the best answers. You also can post or answer questions. If, however, you do opt to post questions to question and answer sites such as Stack Overflow, be sure to familiarize yourself with the etiquette of the site. For example, you should thoroughly search existing posts to see if your question has been addressed before, and you should provide a minimal, complete, and verifiable example of the code that generates the problem or question. Sites like Stack Overflow are incredibly helpful partly because active users police compliance with site norms and informal rules.

2.8 Basic math operators and functions

Below are a few of the most common mathematical operators and functions. You can use these to perform calculations in a script or at the command line, as well as to create or manipulate data objects.

A few examples...

Figure 8: Example—Finding help via Google



About 1,420,000,000 results (1.16 seconds)

R: Cross Tabulation and Table Creation
https://stat.ethz.ch/R-manual/R-devel/library/base/html/table.html
an arbitrary **R** object, or an object inheriting from class "table" for the ... xtabs() <-> as.data.frame.**table()** : UCBAdmissions ## already a contingency table DF ...

R Function of the Day: table | R-bloggers
https://www.r-bloggers.com/r-function-of-the-day-table/
Sep 21, 2009 - The **R** Function of the Day series will focus on describing in plain language ... The **table** function is a very basic, but essential, function to master ...

table function | R Documentation
https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/table
table() returns a contingency table, an object of class "table" , an array of integer values. Note that unlike S the result is always an array , a 1D array if one factor ...

12. Two Way Tables — R Tutorial - cyclismo.org
www.cyclismo.org/tutorial/R/tables.html
Here we look at some examples of how to work with two way **tables**. ... The data file contains only two columns, and when read **R** interprets them both as factors:.

Quick-R: Frequencies
www.statmethods.net/stats/frequencies.html
Learn how to use **R** to create frequency and contingency **tables** from categorical variables, along with tests of independence, and measures of association.

Using table() to create 3 variable frequency table in R - Stack Overflow
https://stackoverflow.com/.../using-table-to-create-3-variable-frequency-table-in-r
May 25, 2015 - The easiest way is to probably use the reshape2 package. Firstly you will need to convert

addition	+
subtraction	−
multiplication	*
division	/
exponentiation	^ or **
log (natural)	`log()`
square root	`sqrt()`
absolute value	`abs()`

```
z <- 1
y <- 0
x <- 5
w <- 100
```

```
a <- z + y
a

## [1] 1

b <- x^2
b

## [1] 25

c <- x * w
c

## [1] 500

d <- sqrt(w) + x
d

## [1] 15

e <- log(w)
e

## [1] 4.60517
```

2.9 Logical Operators

Logical operators test conditions. For example, you might want a subset of data that includes observations for which a specific variable exceeds some value, or you may want to find observations with missing values. You can also use these operators to generate variables and data– often using the `if()` or `ifelse()` function.

less than	<
less than or equal to	<=
greater than	>
greater than or equal to	>=
exactly equal to	==
not equal to	! =
Not x	!x
x *or* y	x | y
x *and* y	x & y
test if x is true	`isTRUE(x)`
test for missing value	`is.na(x)`

```
x==5    # this is a logical operator

## [1] TRUE
```

```
x

## [1] 5

x <- TRUE       # assign logical values to variables

x+z             # explain this output ## numeric value of TRUE = 1, so 1 + 2

## [1] 2

x <- FALSE
x==0

## [1] TRUE
```

2.10 Data objects in R

Objects are the building blocks of R. When you use R to analyze data, you will typically direct R to perform a series of functions (often commands or calculations) on a data object—usually, a data frame. Data frames are the key unit of data analysis in R, but other R objects include vectors, matrices, and lists.

Vectors
A vector is a one-dimensional matrix or array, a group of elements in one row or column. Vectors are important to R for several reasons. A vector can be a data object, i.e., you can create a vector, store it, and perform functions on or with it. The function `c()` allows you to concatenate multiple elements into a vector. These elements can be numbers or strings (characters). As you will learn, you can often use `c()` to pass vectors to functions. You will also use `c()` to create vectors of arguments or instructions to functions.

```
x <- c(1,2,3,4)
x

## [1] 1 2 3 4

x[2]

## [1] 2

y <- c(5,6,7,8,9)
y

## [1] 5 6 7 8 9

y[5]

## [1] 9
```

You can append one vector to another

```
z <- c(x,y)
z

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

Another way to produce a vector containing a sequence of integers

```
q <- 1:5
q

## [1] 1 2 3 4 5
```

You can repeat vectors multiple times

```
ab <- rep(1:5, times=10)
ab

##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
## [36] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

ab <- rep(1:5, 10) ## you do not need the "times" with rep;
                   ## also, notice that R lets you overwrite

cd <- rep(c(1,3,7,9), times=2)
cd

## [1] 1 3 7 9 1 3 7 9

a <- seq(from=2, to=100, by=2)
a

##  [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34
## [18]  36  38  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68
## [35]  70  72  74  76  78  80  82  84  86  88  90  92  94  96  98 100
```

Matrices
You can build matrices using vectors. Use the `cbind()` function to attach vectors as columns, or use `rbind()` to attach vectors in rows.

```
matrix <- as.matrix(cbind(a, ab))

matrix <- as.matrix(rbind(y, q))
```

Data Frames
Data frames are fundamental objects for data analysis in R. When you use R to manipulate, clean, or analyze data, you will typically work with data frames. Generally, each column is a variable, and each row is an observation. You could think of the term "data frame" as R-speak for a dataset. However, data frames can be a bit more flexible. For example, you could save regression results or a table in a data frame, which can help you output results, data, or statistics in your preferred format (more on

this later).

A few important features of data frames:

- you assign elements to a data frame— often an existing dataset but also vectors
- you can have multiple data frames in memory at any given time— i.e., you can have multiple datasets open at the same time
- you can combine data frames to add rows (append) or columns (merge)
- you must name data frames subject to general parameters for naming objects in R
  - names cannot include spaces
  - names can include underscores (_) and/or periods (.)
  - it is best to avoid using functions (per se) as names
  - you can overwrite an existing object by assigning different elements to it (no errors, no warnings)
- to work with an element of a data frame (e.g., a variable in a dataset), you must reference the data frame

You can turn an object, such as a matrix or a list, into a data frame as long as its elements are vectors of equal length. You can also build data frames with vectors using `cbind()` or `rbind()`.

```
data <- data.frame(cbind(a, ab))

data <- data.frame(rbind(a, ab))
```

2.11 Reading in data

R can read data files in a variety of formats. Here, we will use a .csv file containing replication data from Hamermesh and Parker (2005).[2] See below for code to read other types of data files. Note that for most file formats, you must assign the dataset to an object. However, an .RData file will load the data frame as it was saved in .RData format, including the name.

Note: If the data file is stored in your working directory, you need only specify the file name. However, if the file is stored somewhere else on your computer, you will need to include the file path.

```
# csv file
data <- read.csv("teachingratingsexcel.csv", header=TRUE)
```

[2] Hamermesh, Daniel S., and Amy Parker. 2005. "Beauty in the Classroom: InstructorsâĂŹ Pulchritude and Putative Pedagogical Productivity." *Economics of Education Review* 24.4: 369-376. The ERC thanks Daniel Hamermesh for generously sharing data from his article with Amy Parker which investigates the relationship between assessments of university instructors' beauty and their course evaluations.

```
# .dta file (Stata)
dtafile <- read.dta("your_data.dta")
```

```
# .RData file
load("your_data.RData")
```

2.12 Data basics: Looking at data

The `names()` function will display the names of the columns (variables) in a data frame.

```
names(data)

## [1] "minority"    "age"         "female"      "onecredit"   "beauty"
## [6] "course_eval" "intro"       "nnenglish"
```

The `dim()` function returns the dimensions of the data frame. Note that rows are the first dimension and columns are the second dimension. This convention is consistent across R functions and packages.

```
dim(data)

## [1] 463   8

dim(data)[1]

## [1] 463

dim(data)[2]

## [1] 8
```

You can refer to specific rows or columns in a data frame by row or column number(s)— this allows you to see a subset of your data. You could even assign it to a new object and you would have effectively subset your data. Note the comma inside the square brackets. This comma differentiates rows from columns. If you refer to only a row or column, you still need the comma. Numbers or code to the left of the comma refer to rows, and numbers or code to the right of the comma refer to columns.

```
data[1,]        # row 1 only

##   minority age female onecredit    beauty course_eval intro nnenglish
## 1        1  36      1         0 0.2015666         4.3     0         0

data[1:3,]      # rows 1 to 3 only

##   minority age female onecredit     beauty course_eval intro nnenglish
## 1        1  36      1         0  0.2015666         4.3     0         0
## 2        0  59      0         0 -0.8260813         4.5     0         0
## 3        0  51      0         0 -0.6603327         3.7     0         0
```

```
data[,1]                # column 1 only
data[,2:4]      # columns 2 to 4 only
```

Print some or all of the data to the console by entering the name of the object

```
data
data[1:5,]     # view rows 1 thru 5 of all columns
data[,3]       # view all rows of column 3
```

The `head()` function will show you the first few rows of data for all of the columns or variables in the data frame.

```
head(data)

##   minority age female onecredit     beauty course_eval intro nnenglish
## 1        1  36      1         0  0.2015666         4.3     0         0
## 2        0  59      0         0 -0.8260813         4.5     0         0
## 3        0  51      0         0 -0.6603327         3.7     0         0
## 4        0  40      1         0 -0.7663125         4.3     0         0
## 5        0  31      1         0  1.4214451         4.4     0         0
## 6        0  62      0         0  0.5002196         4.2     0         0
```

You can also view specific variables by referencing the data frame and the variable name, linking the data frame and the variable together with a dollar sign ($).

```
data$course_eval
data$female
data$beauty

course_eval     # error! why?
```

Find out the classification or type of an object such as a data frame or a variable with the `class()` function

```
class(data)

## [1] "data.frame"

class(data$course_eval)

## [1] "numeric"

class(data$female)

## [1] "integer"
```

2.13 Writing data to disk

You can save data frames in a variety of formats, including .RData, .csv, and .dta. Save .RData files with the `save()` function. For other formats, you must write the file. In all cases, you reference the data frame first and name the file, and files will be saved in your working directory unless you specify an alternative file path. You can also save your workspace as an .RData file.

```r
write.csv(data, "evaluation_data.csv", row.names=FALSE)

write.dta(data, "evaluation_data.dta")

save(data, file="evaluation_data.RData")  # save just a data frame
save.image(file="course_evaluations.RData")  # save your current workspace
```

3 Basic Data Analysis

3.1 Tables

The basic `table()` function in R creates a very simple table, but the `table()` function is also incredibly flexible. Depending on your needs and preferences, you can take a quick look at frequencies for a variable or a crosstab, but you also can build a more elaborate custom table for display or publication (the next session will review how to export tables from R).

```r
# table() function
table(data$female, useNA="always")

##
##    0    1 <NA>
##  268  195    0
```

You can store a table as an object and continue to add features or make changes if you wish.

```r
crosstab <- table(data$female, data$minority, useNA="always",
                  dnn=c("Gender", "Race or Ethnicity"))     # add dimension names
crosstab

##       Race or Ethnicity
## Gender   0   1 <NA>
##   0    240  28    0
##   1    159  36    0
##   <NA>   0   0    0

crosstab <- crosstab[c(2, 1, 3), c(2, 1, 3)]   # change order of rows and columns
crosstab

##       Race or Ethnicity
## Gender   1   0 <NA>
##   1      36 159    0
##   0      28 240    0
##   <NA>    0   0    0

row.names(crosstab) <- c("Female", "Male", "NA")            # add row names
colnames(crosstab) <- c("Minority", "White", "NA")          # add column names
crosstab

##         Race or Ethnicity
## Gender   Minority White  NA
##   Female       36   159   0
##   Male         28   240   0
##   NA            0     0   0
```

You also can generate tables of proportions and marginal frequencies. Note that you do not have to save a table as an object to use the `margin.table()` and `prop.table()` functions.

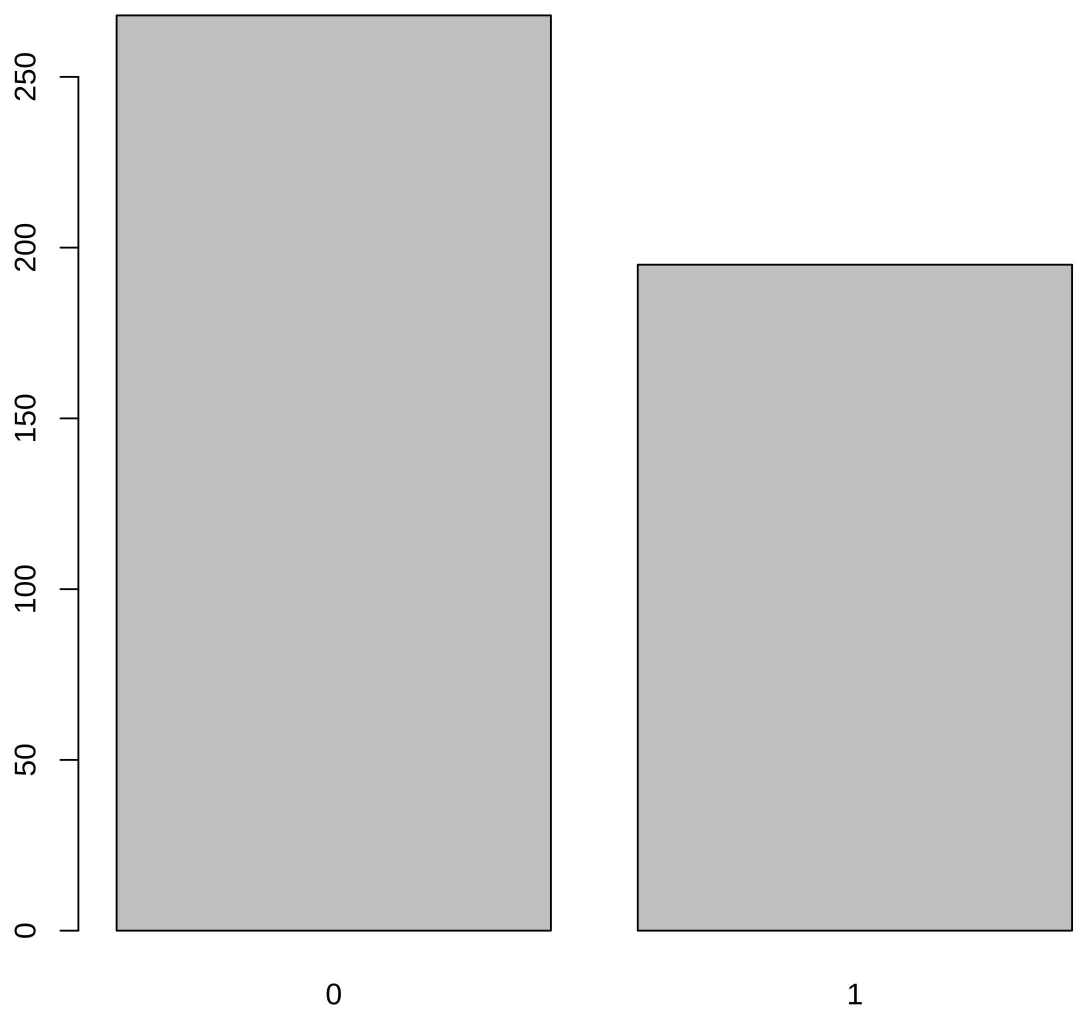
```
mytable <- table(data$female, data$minority, useNA="always",
                 dnn=c("Female", "Minority"))
margin.table(mytable, 1)    # marginal frequencies for 1st dimension

## Female
##    0    1 <NA>
##  268  195    0

margin.table(mytable, 2)    # marginal frequencies for 2nd dimension

## Minority
##    0    1 <NA>
##  399   64    0

prop.table(mytable)

##       Minority
## Female          0          1       <NA>
##   0    0.51835853 0.06047516 0.00000000
##   1    0.34341253 0.07775378 0.00000000
##   <NA> 0.00000000 0.00000000 0.00000000

prop.table(mytable, 1)      # proportions for 1st dimension

##       Minority
## Female         0         1      <NA>
##   0    0.8955224 0.1044776 0.0000000
##   1    0.8153846 0.1846154 0.0000000
##   <NA>

prop.table(mytable, 2)      # proportions for 2nd dimension
```

```
##       Minority
## Female         0         1 <NA>
##   0    0.6015038 0.4375000
##   1    0.3984962 0.5625000
##   <NA> 0.0000000 0.0000000
```

While the `table()` function offers considerable flexibility to generate custom tables, the `descr` package includes functions that return frequency tables and crosstabs. The `descr` package can also handle weighted data.

The `freq()` function returns a frequency table as well as a bar plot showing the distribution of the variable in question. To omit the plot, simply add the argument `plot = FALSE` to the function.

```
library(descr)

freq(data$female)  ## frequency table w/ plot
```



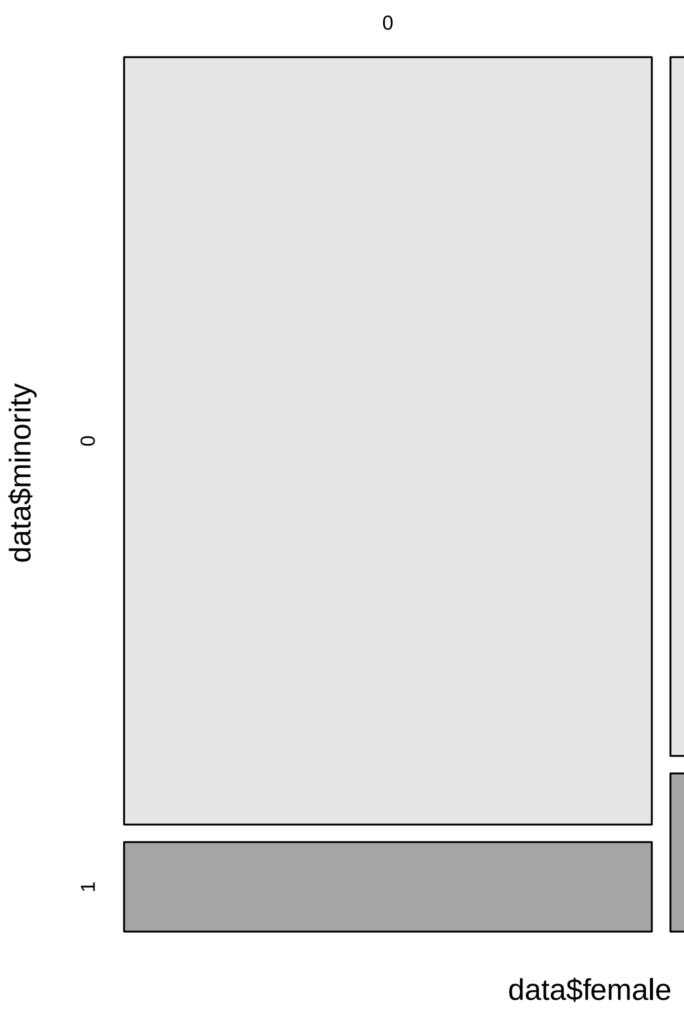
```
## data$female
##       Frequency Percent
## 0           268   57.88
## 1           195   42.12
## Total       463  100.00
```

```
freq(data$minority, plot = FALSE)  ## frequency table

## data$minority
##       Frequency Percent
## 0           399   86.18
## 1             64   13.82
## Total        463  100.00
```

The `crosstab()` function returns a crosstab as well as a mosaic plot. To omit the plot, simply add the argument `plot = FALSE`. The `crosstab()` function can take additional arguments that add a variety of different features, such as proportions by row or column and a χ^2 test. For more details on these options, consult R Help or the documentation for the `descr` package.

```
crosstab(data$minority, data$female, plot = FALSE)

##    Cell Contents
## |-------------------------|
## |                   Count |
## |-------------------------|
##
## ==================================
##                  data$female
## data$minority      0     1   Total
## ----------------------------------
## 0                240   159     399
## ----------------------------------
## 1                 28    36      64
## ----------------------------------
## Total            268   195     463
## ==================================
```

```
crosstab(data$minority, data$female, prop.c = TRUE)
```



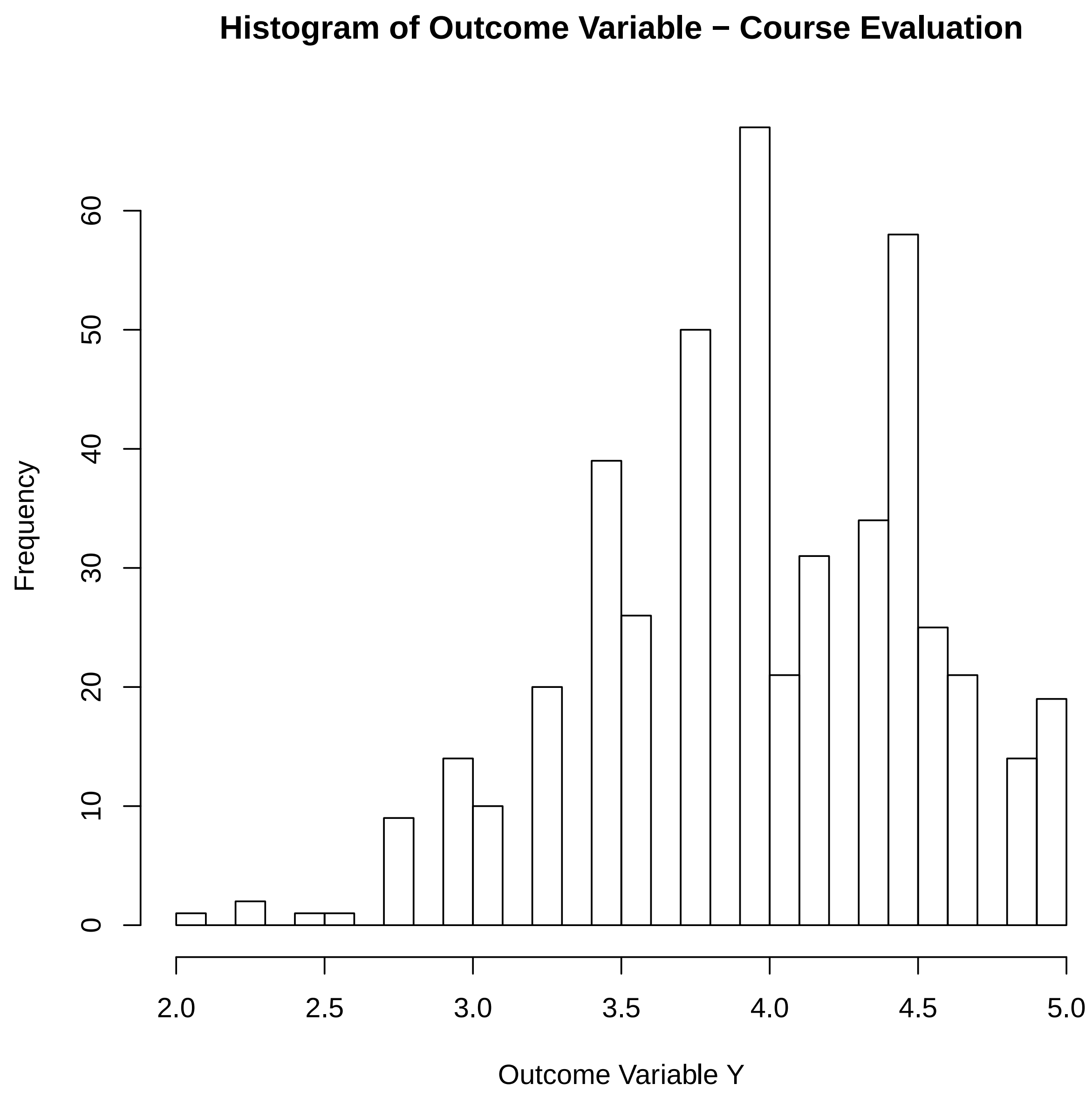
```
##    Cell Contents
## |-------------------------|
## |                   Count |
## |          Column Percent |
## |-------------------------|
##
## ======================================
##                  data$female
## data$minority        0       1   Total
## --------------------------------------
## 0                  240     159     399
##                  89.6%   81.5%
## --------------------------------------
## 1                   28      36      64
##                  10.4%   18.5%
## --------------------------------------
## Total              268     195     463
##                  57.9%   42.1%
## ======================================
```

3.2 Basic histograms and scatterplots

To quickly visualize the distribution of your data, you can generate basic histograms and scatter plots. You can produce very basic plots, or you can add features or options to enhance the appearance of plots. A later session will cover the use of the `ggplot2` package to produce more complex figures.

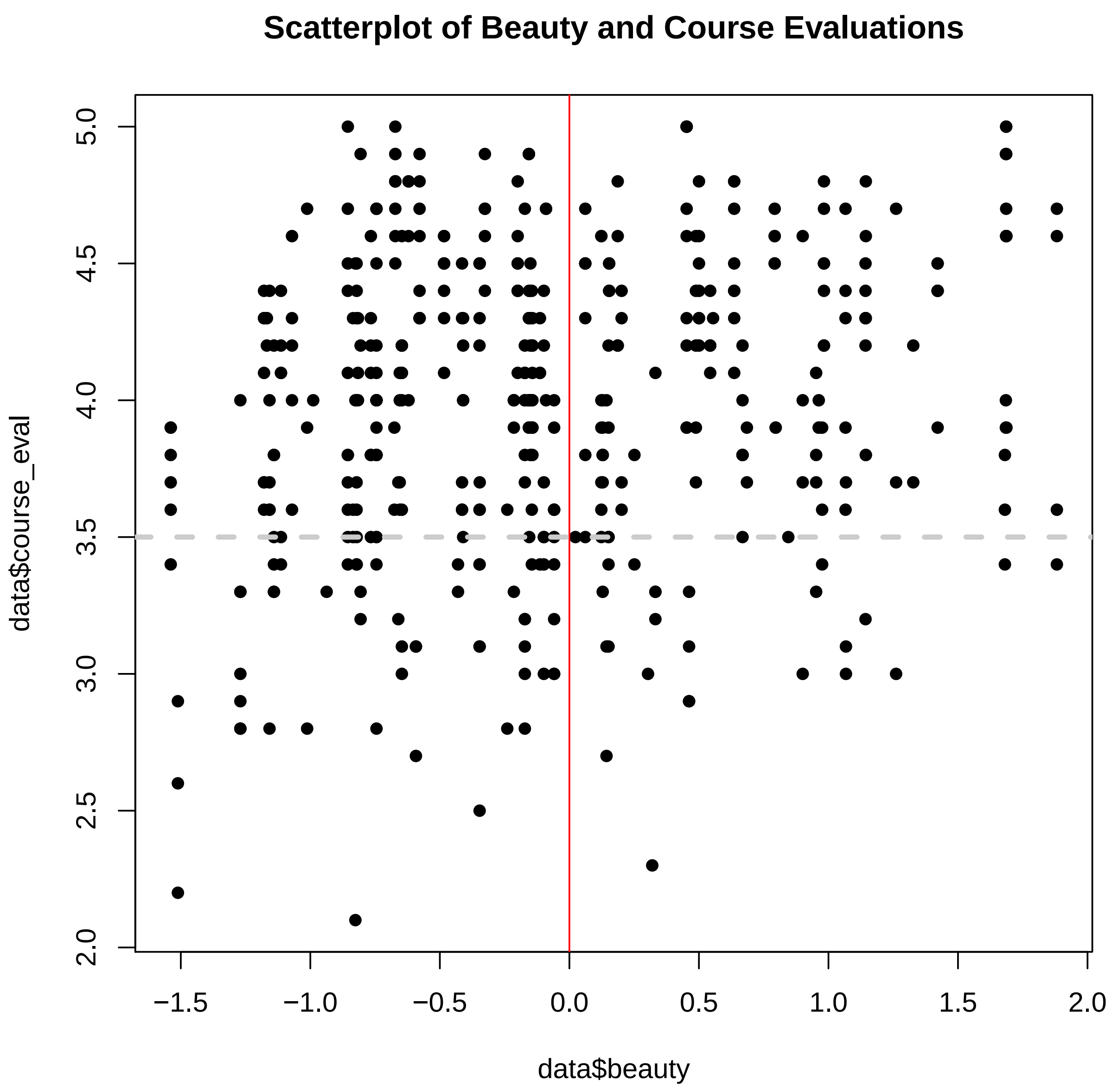
Histogram

```
# hist()
hist(data$course_eval, breaks=25,
     main="Histogram of Outcome Variable - Course Evaluation",
     xlab="Outcome Variable Y")
```



Scatterplot

```
# plot()
plot(data$beauty, data$course_eval,
     main="Scatterplot of Beauty and Course Evaluations",
     pch=16)             # pch changes the shape the points
abline(v=0, col="red")  # abline adds a line
abline(h=3.5, col="grey80", lty=2, lwd=3)
```



You can save a plot in PDF format. R will save the file to your working directory unless you specify a different file path.

```
# save to disk
pdf("basic_plot.pdf")
plot(data$beauty, data$course_eval,
     main="Scatterplot of Beauty and Course Evaluations", pch=16)
abline(v=0, col="red")
abline(h=3.5, col="grey80", lty=2, lwd=3)
dev.off()
```

3.3 Summary statistics

The `summary()` function returns descriptive statistics for your entire dataset or for a specific variable.

```
# summary() function
summary(data)

##     minority            age            female          onecredit
##  Min.   :0.0000   Min.   :29.00   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:42.00   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :0.0000   Median :48.00   Median :0.0000   Median :0.00000
##  Mean   :0.1382   Mean   :48.37   Mean   :0.4212   Mean   :0.05832
##  3rd Qu.:0.0000   3rd Qu.:57.00   3rd Qu.:1.0000   3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :73.00   Max.   :1.0000   Max.   :1.00000
##      beauty           course_eval        intro           nnenglish
##  Min.   :-1.53884   Min.   :2.100   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:-0.74462   1st Qu.:3.600   1st Qu.:0.0000   1st Qu.:0.00000
##  Median :-0.15636   Median :4.000   Median :0.0000   Median :0.00000
##  Mean   :-0.08835   Mean   :3.998   Mean   :0.3391   Mean   :0.06048
##  3rd Qu.: 0.45725   3rd Qu.:4.400   3rd Qu.:1.0000   3rd Qu.:0.00000
##  Max.   : 1.88167   Max.   :5.000   Max.   :1.0000   Max.   :1.00000

summary(data$beauty)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.53900 -0.74460 -0.15640 -0.08835  0.45730  1.88200
```

The `quantile()` function also describes the distribution of a specific variable.

```
quantile(data$course_eval)

##   0%  25%  50%  75% 100%
##  2.1  3.6  4.0  4.4  5.0
```

There are multiple functions to obtain specific summary statistics. Many of these are included in the `summary()` function, but several are not.

NOTE: Missing values will create problems with these functions, so be sure to exclude them. Generally, for functions that evaluate 1 variable (e.g., mean, variance), use the argument `na.rm = TRUE`. Functions that evaluate 2 variables (e.g., covariance), take the argument `use = complete.obs`.

Minimum and Maximum Values

```
min(data$course_eval)

## [1] 2.1

max(data$course_eval)

## [1] 5
```

Mean

```
mean(data$course_eval, na.rm=TRUE)

## [1] 3.998272

mean(data$course_eval[data$female == 0])  ## mean for male instructors

## [1] 4.06903

mean(data$course_eval[data$female == 1])  ## mean for female instructors

## [1] 3.901026
```

Median

```
median(data$course_eval, na.rm=TRUE)

## [1] 4

median(data$course_eval[data$female == 0])

## [1] 4.15

median(data$course_eval[data$female == 1])

## [1] 3.9
```

Mode

```
mfv(data$course_eval) # basic evaluation - most frequent value

## [1] 4

mlv(data$course_eval, method="mfv") # can choose from a variety of estimators

## Mode (most likely value): 4
## Bickel's modal skewness: 0.04535637
## Call: mlv.default(x = data$course_eval, method = "mfv")
```

Standard Deviation

```
sd(data$course_eval, na.rm=TRUE)

## [1] 0.5548656

sd(data$course_eval[data$female == 0])

## [1] 0.5566518

sd(data$course_eval[data$female == 1])

## [1] 0.5388026
```

The `by()` function offers an easy option to obtain univariate statistics for subsets of a data frame by a factor (categorical) variable. The first argument is the variable variable to describe, the second argument is the factor or group variable, and these are followed by the function to apply and an argument for handling missing values. Note that `by()` can be used with many functions including `summary()`. Several examples are included below.

```
by(data$course_eval, data$female, mean, na.rm=TRUE)

## data$female: 0
## [1] 4.06903
## --------------------------------------------------------
## data$female: 1
## [1] 3.901026

by(data$course_eval, data$female, median, na.rm=TRUE)

## data$female: 0
## [1] 4.15
## --------------------------------------------------------
## data$female: 1
## [1] 3.9

by(data$course_eval, data$female, sd, na.rm=TRUE)

## data$female: 0
## [1] 0.5566518
## --------------------------------------------------------
## data$female: 1
## [1] 0.5388026

by(data$course_eval, data$female, summary)

## data$female: 0
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.100   3.700   4.150   4.069   4.500   5.000
## --------------------------------------------------------
## data$female: 1
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.300   3.600   3.900   3.901   4.300   4.900
```

3.4 Basic Evaluations & Statistical Tests

Correlation

With the `cor()` function you can choose from several methods—Pearson correlation, Kendall rank correlation (Kendall's τ), or Spearman rank correlation (Spearman's ρ).

```
cor(data$course_eval, data$beauty, use="complete.obs")

## [1] 0.1890391

cor(data$course_eval, data$beauty, use="pairwise.complete.obs")

## [1] 0.1890391

## Pearson is the default
cor(data$course_eval, data$beauty, use="complete.obs", method="pearson")

## [1] 0.1890391

cor(data$course_eval, data$beauty, use="complete.obs", method="kendall")

## [1] 0.1113366

cor(data$course_eval, data$beauty, use="complete.obs", method="spearman")

## [1] 0.1640352

cor(data$course_eval[data$female == 0], data$beauty[data$female == 0])

## [1] 0.2724031

cor(data$course_eval[data$female == 1], data$beauty[data$female == 1])

## [1] 0.1329867
```

Correlation with significance test
Here, you have several options in addition to the method. You can specify a one- or two-tailed test and significance level for the confidence interval (for Pearson's product moment correlation only). There are additional options related to exact p-values and a continuity correction (for Kendall's τ and Spearman's ρ). For details on additional arguments to the `corr.test()` function, consult R help (enter `?corr.test` at the command line).

```
cor.test(data$course_eval, data$beauty)

##
##  Pearson's product-moment correlation
##
## data:  data$course_eval and data$beauty
## t = 4.1334, df = 461, p-value = 4.247e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.09962508 0.27542455
## sample estimates:
##       cor
## 0.1890391
## Pearson is the default
cor.test(data$course_eval, data$beauty, use="complete.obs", method="pearson")

##
##  Pearson's product-moment correlation
##
## data:  data$course_eval and data$beauty
## t = 4.1334, df = 461, p-value = 4.247e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.09962508 0.27542455
## sample estimates:
##       cor
## 0.1890391

cor.test(data$course_eval, data$beauty, use="complete.obs", method="kendall")

##
##  Kendall's rank correlation tau
##
## data:  data$course_eval and data$beauty
## z = 3.4749, p-value = 0.0005111
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.1113366

cor.test(data$course_eval, data$beauty, use="complete.obs", method="spearman")

## Warning in cor.test.default(data$course_eval, data$beauty, use = "complete.obs",
:  Cannot compute exact p-value with ties

##
##  Spearman's rank correlation rho
##
## data:  data$course_eval and data$beauty
## S = 13829000, p-value = 0.0003939
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.1640352
```

T-tests

The `t.test()` function can also take additional arguments to specify a one- or two-tailed test, μ, paired t-test, equal variance, and the significance level for confidence intervals. You can also indicate the data frame to use and a subset of the data to evaluate.

```
# independent 2-group - first variable is numeric, second is binary factor
t.test(data$course_eval ~ data$female)

##
##  Welch Two Sample t-test
##
## data:  data$course_eval by data$female
## t = 3.2667, df = 425.76, p-value = 0.001176
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.06691755 0.26909088
## sample estimates:
## mean in group 0 mean in group 1
##        4.069030        3.901026

# independent 2-group - both numeric variables
t.test(data$course_eval[data$female == 0], data$course_eval[data$female == 1])

##
##  Welch Two Sample t-test
##
## data:  data$course_eval[data$female == 0] and data$course_eval[data$female == 1]
## t = 3.2667, df = 425.76, p-value = 0.001176
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.06691755 0.26909088
## sample estimates:
## mean of x mean of y
##  4.069030  3.901026
```

Note that although the `t.test()` function returns a confidence interval for the difference in means, it does not actually display the difference. If you need to calculate the difference in means, you could use the `mean()` function or extract the means from the output to calculate the difference.

```
# math
mean(data$course_eval[data$female == 1]) - mean(data$course_eval[data$female == 0])

## [1] -0.1680042

## extract the means from the t.test() result

#  use names() to find which elements to extract
names(t.test(data$course_eval ~ data$female))

## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"

(t.test(data$course_eval ~ data$female)$estimate[2]
    - t.test(data$course_eval ~ data$female)$estimate[1])
```

```
## mean in group 1
##      -0.1680042
```

χ^2-test
Among the possible arguments to the `chisq.test` function is an option for simulating the *p*-value. Note, that if you have already saved tables as objects, you can simply use those existing objects/tables with the `chisq.test()` function. However, it is not necessary to assign a table to an object to perform a χ^2-test.

```
female_minority_crosstab <- table(data$female, data$minority)
chisq.test(female_minority_crosstab)

##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  female_minority_crosstab
## X-squared = 5.431, df = 1, p-value = 0.01978

## note that you do not need to save the table as an object
chisq.test(table(data$female, data$minority))

##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(data$female, data$minority)
## X-squared = 5.431, df = 1, p-value = 0.01978
```

3.5 Regression

The `lm()` function fits data to linear models, and the `glm()` function fits data to generalized linear models. These functions are similar in terms of syntax and output, but `glm()` requires an additional argument that specifies the variance and link functions.

This workshop will focus on linear regression models and provide some information on generalized linear models, but you can use R for a wide variety of other regression classes or models, including but not limited to two-stage least squares, MRP and other multilevel (hierarchical) models, and event history models (survival and/or hazard models).

If we begin by thinking of a basic OLS regression model,

$$y = \alpha + \beta x + \epsilon$$

.
To estimate such a model, simply specify the formula and the data R should use.

The most essential arguments include the following:

- `formula` is the model to estimate— syntax: `y ~ x1 + x2 + x3`

  - This formula includes an intercept
  - To omit the intercept, add -1 to the right hand side
  - You can omit reference to the data frame in the formula (i.e., you can simply use `x1` instead of `data$x1`) if you include the `data=` argument or if you attach your data

- `data` allows you to specify the data frame that contains the data you want to fit
- `family` (for `glm()`) provides the variance and link function necessary to fit a generalized linear model

Family	Variance	Link
gaussian	gaussian	identity
binomial	binomial	logit, probit, or clog log
poisson	poisson	log, identity, or sqrt
Gamma	Gamma	inverse, identity, or log
inverse.gaussian	inverse.gaussian	$1/\mu^2$
quasi	user-defined	user-defined

The following examples are OLS models. Note that the regression models are saved as objects. Although it is not necessary to save a fitted model to object, doing so provides a few benefits because you can easily access the results at any later point in your code without re-running the model. This feature will prove helpful for exporting and plotting results.

```
fit_1 <- lm(course_eval ~ female, data=data)
summary(fit_1)

##
## Call:
## lm(formula = course_eval ~ female, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.96903 -0.36903  0.03097  0.43097  0.99897
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.06903    0.03355  121.29  < 2e-16 ***
## female      -0.16800    0.05169   -3.25  0.00124 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5492 on 461 degrees of freedom
## Multiple R-squared:  0.0224,Adjusted R-squared:  0.02028
## F-statistic: 10.56 on 1 and 461 DF,  p-value: 0.001239
```

Include only a subset of the data by adding the `subset=` argument

```
fit_1_male <- lm(course_eval ~ beauty, data=data, subset=female==1)
summary(fit_1_male)

##
## Call:
## lm(formula = course_eval ~ beauty, data = data, subset = female ==
##     1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.62661 -0.37392  0.01511  0.41156  1.01511
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.89859    0.03836 101.624   <2e-16 ***
## beauty       0.08762    0.04700   1.864   0.0638 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5354 on 193 degrees of freedom
## Multiple R-squared:  0.01769,Adjusted R-squared:  0.0126
## F-statistic: 3.475 on 1 and 193 DF,  p-value: 0.06383

fit_1_female <- lm(course_eval ~ beauty, data=data, subset=female==0)
summary(fit_1_female)

##
## Call:
## lm(formula = course_eval ~ beauty, data = data, subset = female ==
##     0)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.83820 -0.37318  0.05899  0.39397  1.06764
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.10364    0.03362 122.042  < 2e-16 ***
## beauty       0.20027    0.04337   4.617 6.06e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5366 on 266 degrees of freedom
## Multiple R-squared:  0.0742,Adjusted R-squared:  0.07072
## F-statistic: 21.32 on 1 and 266 DF,  p-value: 6.056e-06
```

Include an interaction by joining variables with an asterisk (∗)— note that when you use ∗ to create an interaction term, the `lm()` function will automatically include the base terms in the model.

```
fit_2 <- lm(course_eval ~ female*beauty, data=data)
summary(fit_2)

##
## Call:
## lm(formula = course_eval ~ female * beauty, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.83820 -0.37387  0.04551  0.39875  1.06764
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.10364    0.03359 122.158  < 2e-16 ***
## female        -0.20505    0.05103  -4.018 6.85e-05 ***
## beauty         0.20027    0.04333   4.622 4.95e-06 ***
## female:beauty -0.11266    0.06398  -1.761   0.0789 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5361 on 459 degrees of freedom
## Multiple R-squared:  0.07256,Adjusted R-squared:  0.0665
## F-statistic: 11.97 on 3 and 459 DF,  p-value: 1.47e-07
```

Include covariates by simply adding them to the specification

```
fit_3 <- lm(course_eval ~ female + beauty + age, data=data)
summary(fit_3)

##
## Call:
## lm(formula = course_eval ~ female + beauty + age, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.85612 -0.35831  0.04697  0.39308  1.04276
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.225242   0.142820  29.584  < 2e-16 ***
## female      -0.210792   0.052824  -3.990 7.68e-05 ***
## beauty       0.139978   0.033243   4.211 3.06e-05 ***
## age         -0.002602   0.002768  -0.940    0.348
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5374 on 459 degrees of freedom
## Multiple R-squared:  0.06809,Adjusted R-squared:  0.062
## F-statistic: 11.18 on 3 and 459 DF,  p-value: 4.305e-07
```

Add fixed effects by designating a factor variable, i.e., a categorical or group variable. Note that you can simply indicate that a variable should be treated as a factor variable. You do not have to generate dummy variables, and no variables will be added to your data frame.

```
fit_4 <-  lm(course_eval ~ factor(intro) + female + beauty + age, data=data)
summary(fit_4)

##
## Call:
## lm(formula = course_eval ~ factor(intro) + female + beauty +
##     age, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.82129 -0.36152  0.06073  0.41574  1.07880
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     4.153528   0.146316  28.387  < 2e-16 ***
## factor(intro)1  0.111469   0.052983   2.104  0.03593 *
## female         -0.201129   0.052828  -3.807  0.00016 ***
## beauty          0.139315   0.033121   4.206 3.12e-05 ***
## age            -0.001986   0.002773  -0.716  0.47436
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5354 on 458 degrees of freedom
## Multiple R-squared:  0.07701,Adjusted R-squared:  0.06895
## F-statistic: 9.553 on 4 and 458 DF,  p-value: 1.998e-07
```

Heteroskedasticity-robust standard errors
The `coeftest()` function in combination with the `vcovHC()` function can generate regression results with robust standard errors. The most commonly used types are `HC0` and `HC1`. Both options use White's estimator, but `HC1` includes a degree of freedom correction. On a practical note, using type `HC1` will produce the same standard errors as `, robust` in Stata.

For more information on various estimators, review the documentation for the `sandwich` package which supports the `vcovHC()` function (as well a functions to generate other types of standard errors).

NOTE: This method cannot generate cluster robust standard errors, but we will review a functional alternative in Session 3.

```
summary(fit_3)

##
## Call:
## lm(formula = course_eval ~ female + beauty + age, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.85612 -0.35831  0.04697  0.39308  1.04276
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.225242   0.142820  29.584  < 2e-16 ***
## female      -0.210792   0.052824  -3.990 7.68e-05 ***
## beauty       0.139978   0.033243   4.211 3.06e-05 ***
## age         -0.002602   0.002768  -0.940    0.348
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5374 on 459 degrees of freedom
## Multiple R-squared:  0.06809,Adjusted R-squared:  0.062
## F-statistic: 11.18 on 3 and 459 DF,  p-value: 4.305e-07

coeftest(fit_3, vcov=vcovHC(fit_3, type="HC0"))

##
## t test of coefficients:
##
```

```
##                Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  4.2252423  0.1392221 30.3489 < 2.2e-16 ***
## female      -0.2107917  0.0527683 -3.9947 7.544e-05 ***
## beauty       0.1399776  0.0314096  4.4565 1.048e-05 ***
## age         -0.0026016  0.0026565 -0.9793    0.3279
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

coeftest(fit_3, vcov=vcovHC(fit_3, type="HC1")) ## (like , robust in Stata)

##
## t test of coefficients:
##
##                Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  4.2252423  0.1398274 30.2175 < 2.2e-16 ***
## female      -0.2107917  0.0529977 -3.9774 8.094e-05 ***
## beauty       0.1399776  0.0315461  4.4372 1.142e-05 ***
## age         -0.0026016  0.0026681 -0.9751      0.33
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```