

Empirical Reasoning Center

R Faculty Workshop (Summer 2016)

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.

It's always a good idea to start an R-script or session by clearing your workspace. Doing so clears all objects in memory but does not affect any files saved on your computer.

```
rm(list=ls(all=TRUE))      # clear all objects in memory
```

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
```

1.1 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
```

1.2 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.

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

A few examples...

```
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

```

1.3 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
```

1.4 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

The function `c()` allows you to concatenate multiple items into a vector. As you will learn, you can often use `c()` to pass vectors 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 1 2 3
## [44] 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 36 38 40 42
## [22] 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84
## [43] 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. Often, 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 Performing Basic Tasks

2.1 Setting up your work space

See the objects currently in memory

```
ls()

## [1] "a"      "ab"     "b"      "c"      "cd"     "d"      "data"   "e"      "matrix"
```

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

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

Working Directory

The working directory is 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.

```
getwd()

## [1] "/Users/patriciakirkland/Dropbox/Empirical Reasoning Center/R Workshop"

setwd("/Users/patriciakirkland/Dropbox/Empirical Reasoning Center/R Workshop")

getwd()          # check again

## [1] "/Users/patriciakirkland/Dropbox/Empirical Reasoning Center/R Workshop"
```

2.2 Installing and loading packages

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.

```
# 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)

# load packages

library(foreign)
library(xtable)
library(arm)

## Loading required package: Matrix
## Loading required package: lme4
##
## arm (Version 1.8-6, built: 2015-7-7)
## Working directory is /Users/patriciakirkland/Dropbox/Empirical Reasoning Center/R
Workshop
##
## Attaching package: 'arm'
## The following object is masked from 'package:xtable':
##
##   display

library(ggplot2)
library(dplyr)
library(stargazer)

##
## Please cite as:
## Hlavac, Marek (2015). stargazer: Well-Formatted Regression and Summary Statistics
Tables.
## R package version 5.2. http://CRAN.R-project.org/package=stargazer

library(modeest)

##
## This is package 'modeest' written by P. PONCET.
## For a complete list of functions, use 'library(help = "modeest")' or 'help.start()'.

library(lmtest)
```



```
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##   as.Date, as.Date.numeric

library(sandwich)
```

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
- more packages: <http://cran.r-project.org/web/packages/>

2.3 Reading in data

R can read data files in a variety of formats. Today, we will use a .csv file, but 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)

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

# .RData file
# load("fakedata1.RData")
# data
```

2.4 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" "course_eval"
## [7] "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 0.2899157 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 0.2899157 4.3 0 0
## 2 0 59 0 0 -0.7377322 4.5 0 0
## 3 0 51 0 0 -0.5719836 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 0.2899157         4.3      0          0
## 2         0  59      0         0 -0.7377322         4.5      0          0
## 3         0  51      0         0 -0.5719836         3.7      0          0
## 4         0  40      1         0 -0.6779634         4.3      0          0
## 5         0  31      1         0 1.5097940         4.4      0          0
## 6         0  62      0         0 0.5885687         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?

## Error in eval(expr, envir, enclos): object 'course_eval' not found
```

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"
```

2.5 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.

```
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).

```
# 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.

```
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
```

@

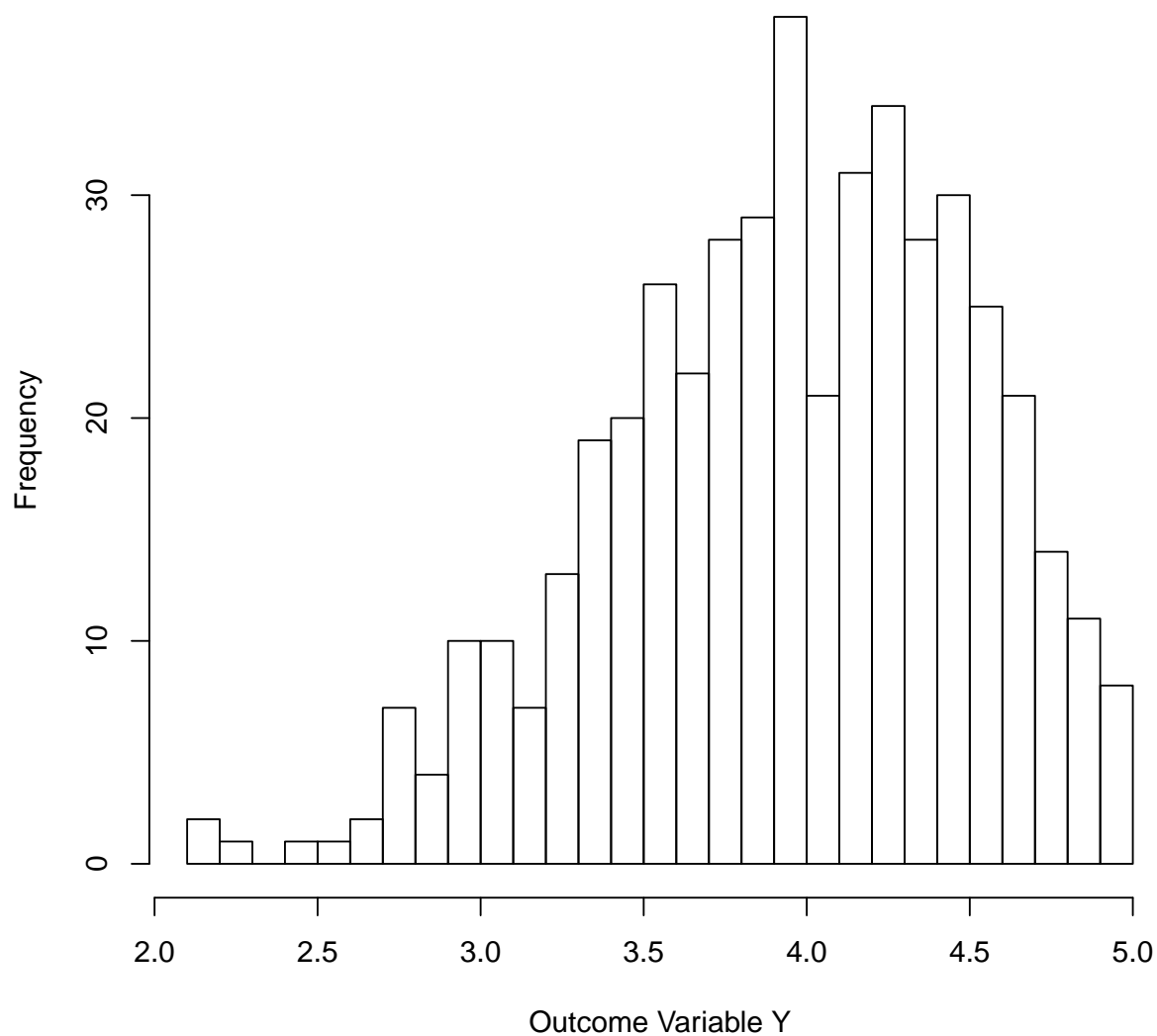
3.2 Basic histograms and scatterplots

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")
```

Histogram of Outcome Variable – Course Evaluation

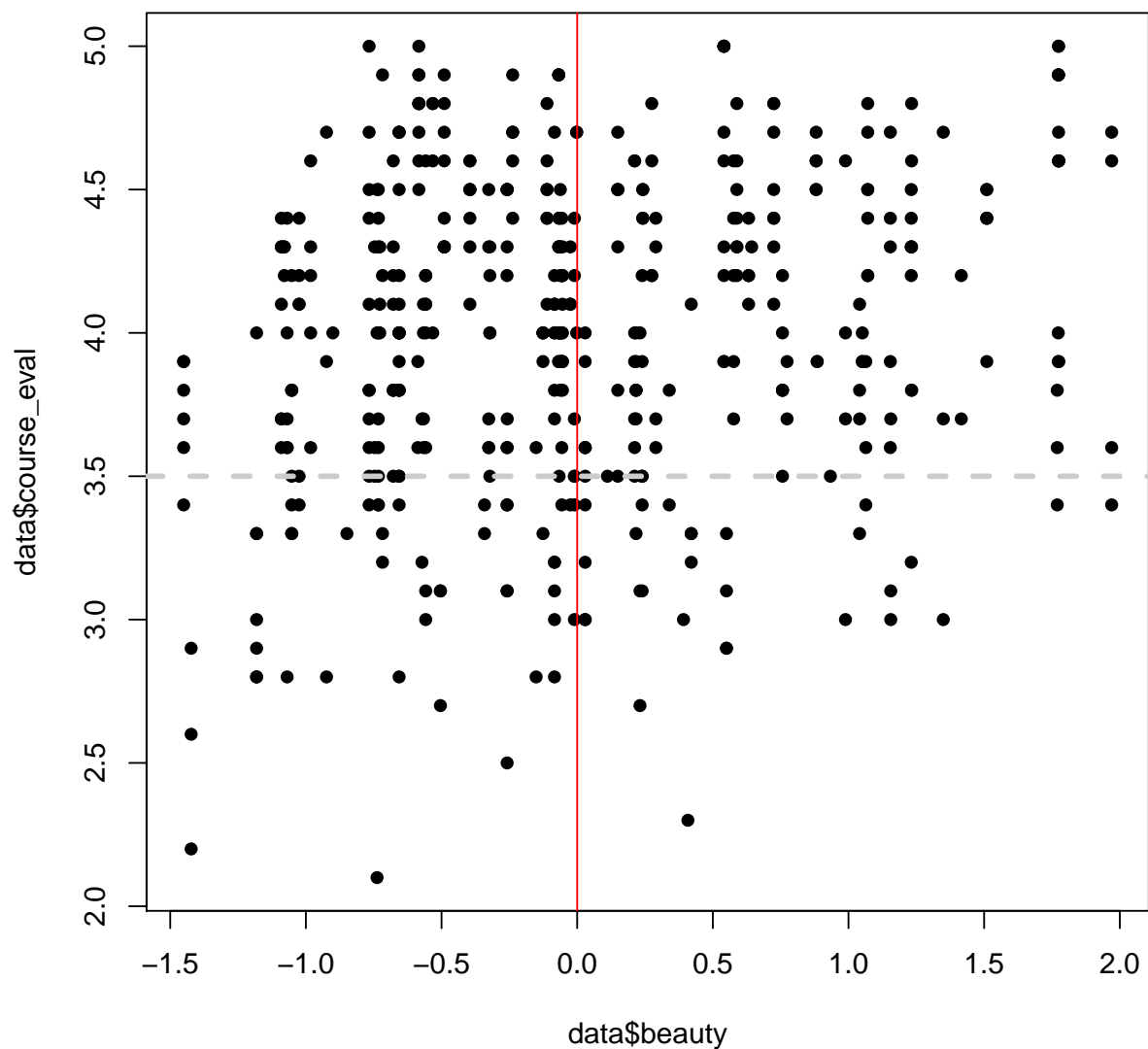


Scatterplot

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

```
abline(h=3.5, col="grey80", lty=2, lwd=3)
```

Scatterplot of Beauty and Course Evaluations



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()
```

```
## pdf
## 2
```

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	beauty
##	Min. :0.0000	Min. :29.00	Min. :0.0000	Min. :0.00000	Min. :-1.4504940
##	1st Qu.:0.0000	1st Qu.:42.00	1st Qu.:0.0000	1st Qu.:0.00000	1st Qu.: -0.6562689
##	Median :0.0000	Median :48.00	Median :0.0000	Median :0.00000	Median :-0.0680143
##	Mean :0.1382	Mean :48.37	Mean :0.4212	Mean :0.05832	Mean : 0.0000001
##	3rd Qu.:0.0000	3rd Qu.:57.00	3rd Qu.:1.0000	3rd Qu.:0.00000	3rd Qu.: 0.5456024
##	Max. :1.0000	Max. :73.00	Max. :1.0000	Max. :1.00000	Max. : 1.9700230

##	course_eval	intro	nnenglish
##	Min. :2.100	Min. :0.0000	Min. :0.00000
##	1st Qu.:3.600	1st Qu.:0.0000	1st Qu.:0.00000
##	Median :4.000	Median :0.0000	Median :0.00000
##	Mean :3.998	Mean :0.3391	Mean :0.06048
##	3rd Qu.:4.400	3rd Qu.:1.0000	3rd Qu.:0.00000
##	Max. :5.000	Max. :1.0000	Max. :1.00000


```
summary(data$beauty)
```

##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	-1.4500000	-0.6563000	-0.0680100	0.0000001	0.5456000	1.9700000

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
```

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:
##  0.09962509 0.27542456
## 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:
##  0.09962509 0.27542456
## 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:
## 0.06691754 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:
## 0.06691754 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" "null.value"
## [7] "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)  3.89085     0.03873  100.468 <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)   4.08595    0.03298 123.882 < 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  0.39876  1.06764
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.08595    0.03295 123.999 < 2e-16 ***
## female        -0.19510    0.05089  -3.834 0.000144 ***
## beauty         0.20027    0.04333   4.622 4.95e-06 ***
## female:beauty -0.11266    0.06398  -1.761 0.078910 .
## ---
## 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)  4.212875   0.143527  29.353 < 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)  4.141220   0.146994  28.173 < 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)  4.212875   0.143527  29.353  < 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)  4.2128754  0.1393425 30.2340 < 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)  4.2128754  0.1399483 30.1031 < 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

```