

Lock5 with R
a companion to
Statistics: Unlocking the Power of Data

Randall Pruim and Lana Park

May 18, 2015

Contents

0	Introduction to R and Statistics	7
0.1	Getting Started With RStudio	7
0.2	Getting Help in RStudio	9
0.3	Four Things to Know About R	10
0.4	Data in R	11
0.5	The Most Important Template	16
0.6	Manipulating your data	18
0.7	Using R Markdown	21
0.8	Statistics: Answering Questions With Data	23
0.9	A First Example: The Lady Tasting Tea	25
0.10	Coins and Cups	26
1	Collecting Data	31
1.1	The Structure of Data	31
1.2	Sampling from a Population	35
1.3	Experiments and Observational Studies	37
2	Describing Data	39
2.1	Categorical Variables	39
2.2	One Quantitative Variable: Shape and Center	45
2.3	One Quantitative Variable: Measures of Spread	53
2.4	Outliers, Boxplots, and Quantitative/Categorical Relationships	59

2.5	Two Quantitative Variables: Scatterplot and Correlation	63
2.6	Two Quantitative Variables: Linear Regression	67
3	Confidence Intervals	73
3.1	Sampling Distributions	73
3.2	Understanding and Interpreting Confidence Intervals	77
3.3	Constructing Bootstrap Confidence Intervals	82
3.4	Bootstrap Confidence Intervals Using Percentiles	87
4	Hypothesis Tests	95
4.1	Introducing Hypothesis Tests	95
4.2	Measuring Evidence with P-values	96
4.3	Determining Statistical Significance	102
4.4	Creating Randomization Distributions	104
4.5	Confidence Intervals and Hypothesis Tests	110
5	Approximating with a Distribution	113
5.1	Normal Distributions	113
5.2	Confidence Intervals and P-values Using Normal Distributions	121
6	Inference for Means and Proportions	125
6.1	Distribution of a Sample Proportion	125
6.2	Confidence Interval for a Single Proportion	127
6.3	Test for a Single Proportion	130
6.4	Distribution of a Sample Mean	133
6.5	Confidence Interval for a Mean Using the t-Distribution	137
6.6	Test for a Single Mean	140
6.7	Distribution of Differences in Proportions	142
6.8	Confidence Interval for a Difference in Proportions	144
6.9	Test For a Difference in Proportions	145
6.10	Distribution of Differences in Means	146
6.11	Confidence Interval for a Difference in Means	147
6.12	Test for a Difference in Means	148
6.13	Paired Difference in Means	149

7	Chi-Squared Tests for Categorical Variables	153
7.1	Testing Goodness-of-Fit for a Single Categorical Variable	153
7.2	Testing for an Association Between Two Categorical Variables	157
8	ANOVA to Compare Means	163
8.1	Analysis of Variance	163
8.2	Pairwise Comparisons and Inference After ANOVA	171
9	Inference for Regression	177
9.1	Inference for Slope and Correlation	177
9.2	ANOVA for Regression	183
9.3	Confidence and Prediction Intervals	187
10	Multiple Regression	191
10.1	Multiple Predictors	191
10.2	Checking Conditions for a Regression Model	195
10.3	Using Multiple Regression	196



Introduction to R and Statistics

0.1 Getting Started With RStudio

RStudio

RStudio provides an integrated development environment (IDE) for R that makes R much easier to use. It is freely available from <http://rstudio.com> in versions for Macintosh, PC, or Linux. RStudio server provides access to RStudio via a web browser. We will generally assume that RStudio is being used throughout. Although most things can be done without RStudio as well, our descriptions may apply only to RStudio.

Loading packages

R is divided up into packages. A few of these are loaded every time you run R, but most have to be selected. This way you only have as much of R as you need.

In the Packages tab in RStudio, check the boxes next to the following packages to load them:

- Lock5withR (data sets and utilities to accompany the text)
- mosaic (a package from Project MOSAIC)
- mosaicData (Project MOSAIC data sets)

You can also load these packages with the following commands:

```
require(Lock5withR)
require(mosaic)
require(mosaicData)
```

require-packages

We will always assume that these three packages have been loaded.

Using R as a calculator

Notice that RStudio divides its world into four panels. Several of the panels are further subdivided into multiple tabs. The console panel is where we type commands that R will execute.

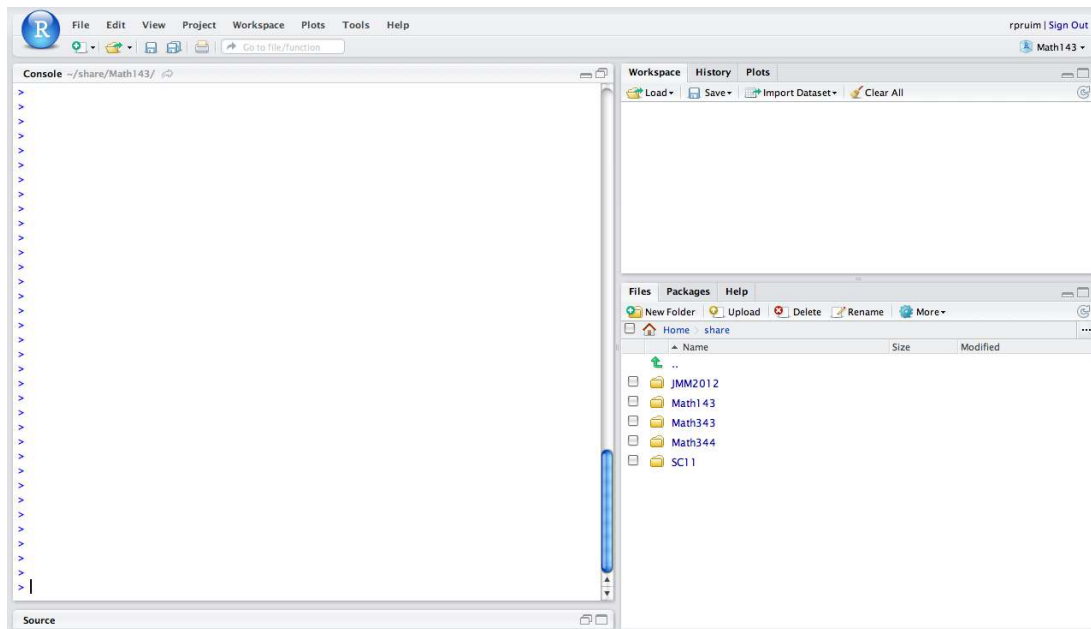


Figure 1: Welcome to RStudio.

R can be used as a calculator. Try typing the following commands in the console panel.

```
5 + 3
```

```
[1] 8
```

```
15.3 * 23.4
```

```
[1] 358.02
```

```
sqrt(16)
```

```
[1] 4
```

arithmetic2

You can save values to named variables for later reuse

```
product = 15.3 * 23.4      # save result  
product                    # show the result
```

```
[1] 358.02
```

```
product <- 15.3 * 23.4     # <- is assignment operator, same as =  
product
```

```
[1] 358.02
```

variables2


```
15.3 * 23.4 -> newproduct    # -> assigns to the right
newproduct

[1] 358.02

.5 * product                 # half of the product

[1] 179.01

log(product)                 # (natural) log of the product

[1] 5.880589

log10(product)               # base 10 log of the product

[1] 2.553907

log(product,base=2)          # base 2 log of the product

[1] 8.483896
```

The semi-colon can be used to place multiple commands on one line. One frequent use of this is to save and print a value all in one go:

```
15.3 * 23.4 -> product; product    # save result and show it

[1] 358.02
```

[variables-semi2](#)

0.2 Getting Help in RStudio

The RStudio help system

There are several ways to get RStudio to help you when you forget something. Most objects in packages have help files that you can access by typing something like:

```
?bargraph
?histogram
?HELPrct
```

[help-questionmark](#)

You can search the help system using

```
help.search("Grand Rapids")    # Does R know anything about Grand Rapids?
```

[help-GR](#)

This can be useful if you don't know the name of the function or data set you are looking for.

History

If you know you have done something before, but can't remember how, you can search your history. The history tab shows a list of recently executed commands. There is also a search bar to help you find things from longer ago.

Error messages

When things go wrong, R tries to help you out by providing an error message. If you can't make sense of the message, you can try copying and pasting your command and the error message and sending to me in an email. One common error message is illustrated below.

```
fred <- 23
frd

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

error-message

The object `frd` is not found because it was mistyped. It should have been `fred`. If you see an “object not found” message, check your typing and check to make sure that the necessary packages have been loaded.

0.3 Four Things to Know About R

Computers are great for doing complicated computations quickly, but you have to speak to them on their terms. Here are few things that will help you communicate with R.

1. R is case-sensitive

If you mis-capitalize something in R it won't do what you want.

2. Functions in R use the following syntax:

```
functionname(argument1, argument2, ...)
```

function-syntax

- The arguments are *always surrounded by (round) parentheses and separated by commas*. Some functions (like `data()`) have no required arguments, but you still need the parentheses.
- If you type a function name without the parentheses, you will see the *code* for that function – which probably isn't what you want at this point.

3. TAB completion and arrows can improve typing speed and accuracy.

If you begin a command and hit the TAB key, R will show you a list of possible ways to complete the command. If you hit TAB after the opening parenthesis of a function, it will show you the list of arguments it expects. The up and down arrows can be used to retrieve past commands.

4. If you get into some sort of mess typing (usually indicated by extra '+' signs along the left edge), you can hit the escape key to get back to a clean prompt.

0.4 Data in R

Data in Packages

Most often, data sets in R are stored in a structure called a **data frame**. There are a number of data sets built into R and many more that come in various add on packages. The **Lock5withR** package, for example, contains all the data sets from our text book. In the book, data set names are printed in bold text.

You can see a list of them using

```
data(package = "Lock5withR")
```

[datasets](#)

You can find a longer list of all data sets available in any loaded package using

```
data()
```

The HELPrct data set

The **HELPrct** data frame from the **mosaicData** package contains data from the Health Evaluation and Linkage to Primary Care randomized clinical trial. You can find out more about the study and the data in this data frame by typing

```
?HELPrct
```

[HELPrcthelp](#)

Among other things, this will tell us something about the subjects in this study:

Eligible subjects were adults, who spoke Spanish or English, reported alcohol, heroin or cocaine as their first or second drug of choice, resided in proximity to the primary care clinic to which they would be referred or were homeless. Patients with established primary care relationships they planned to continue, significant dementia, specific plans to leave the Boston area that would prevent research participation, failure to provide contact information for tracking purposes, or pregnancy were excluded.

Subjects were interviewed at baseline during their detoxification stay and follow-up interviews were undertaken every 6 months for 2 years.

It is often handy to look at the first few rows of a data frame. It will show you the names of the variables and the kind of data in them:

```
head(HELPrct)
```

[headHELP](#)

	age	anysubstatus	anysub	cesd	d1	daysanysub	dayslink	drugrisk	e2b	female	sex	g1b
1	37	1	yes	49	3	177	225	0	NA	0	male	yes
2	37	1	yes	30	22	2	NA	0	NA	0	male	yes
3	26	1	yes	39	0	3	365	20	NA	0	male	no
4	39	1	yes	15	2	189	343	0	1	1	female	no
5	32	1	yes	39	12	2	57	0	1	0	male	no

```

6 47      1 yes 6 1      31      365      0 NA      1 female no
  homeless i1 i2 id indtot linkstatus link      mcs      pcs pss_fr racegrp satreat
1  housed 13 26 1      39      1 yes 25.111990 58.41369      0 black      no
2 homeless 56 62 2      43      NA <NA> 26.670307 36.03694      1 white      no
3  housed 0 0 3      41      0 no 6.762923 74.80633      13 black      no
4  housed 5 5 4      28      0 no 43.967880 61.93168      11 white      yes
5 homeless 10 13 5      38      1 yes 21.675755 37.34558      10 black      no
6  housed 4 4 6      29      0 no 55.508991 46.47521      5 black      no
  sexrisk substance treat
1      4 cocaine yes
2      7 alcohol yes
3      2 heroin no
4      4 heroin no
5      6 cocaine no
6      5 cocaine yes

```

That's plenty of variables to get us started with exploration of data.

Using your own data

From Excel or Google to R

So far we have been using data that lives in R packages. This has allowed us to focus on things like how to make plots and create numerical summaries without worrying too much about the data themselves. But if you are going to do any of your own statistical analyses, then you will need to import your own data into R and have some tools for manipulating the data once it is there.

Excel or Google spreadsheets are reasonable tools for entering (small) data sets by hand and doing basic data tidying (organizing) and cleaning (correcting errors). This section describes how to get data from a spreadsheet into R.

While you are still in the spreadsheet

If you are creating your own data in a spreadsheet with the intent of bringing into R (or some other statistical package) for analysis, it is important that you design your spreadsheet appropriately. For most data sets this will mean

1. The first row should contain variables names.

These should be names that will work well in R. This usually means they will be relatively short and avoid spaces and punctuation.

2. Each additional row corresponds to a case/observational unit.

3. Each column corresponds to a variable.

4. There is **nothing** else in the spreadsheet.

Do not include notes to yourself, plots, numerical summaries, etc. These things can be kept in a separate worksheet, another file, your lab notebook, just not in the worksheet you are going to export.

Exporting to csv

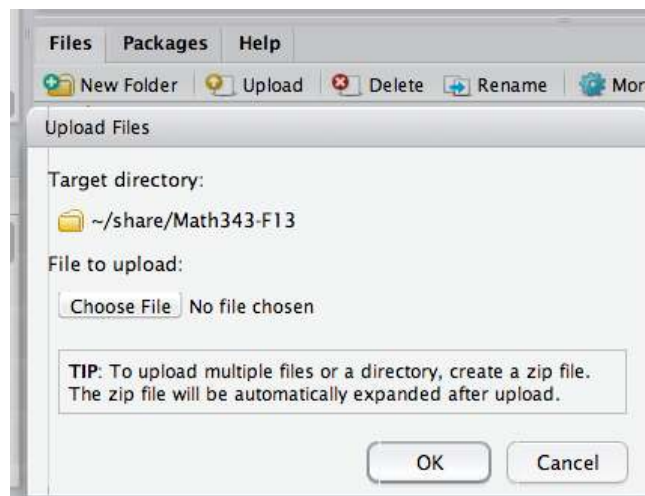
The comma separated values (csv) format has become a standard way of transferring data between programs. Both Google and Excel can export to this format, and R can import from this format. Once your data are ready

to go, export them to csv. Give the file a good name, and remember where you have put it.

Uploading the data (RStudio server only)

To get the data from your computer onto the server, you need to **upload** the data. (You can skip this step if you are working with a local copy of RStudio.) Uploading transfers a copy of your data from your computer onto the server (the “cloud”). This is like uploading pictures to Facebook so you can later use them in posts or as a cover photo or tag your friends or whatever else once the photo is on Facebook.

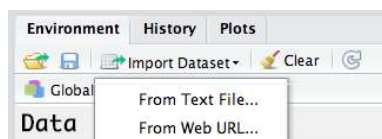
To upload the data, go to the **Files** tab and click on **Upload**:



A window will pop up prompting you to browse to the file's location on your computer. Choose the file and it will upload to the server. You should see it appear in your file menu.

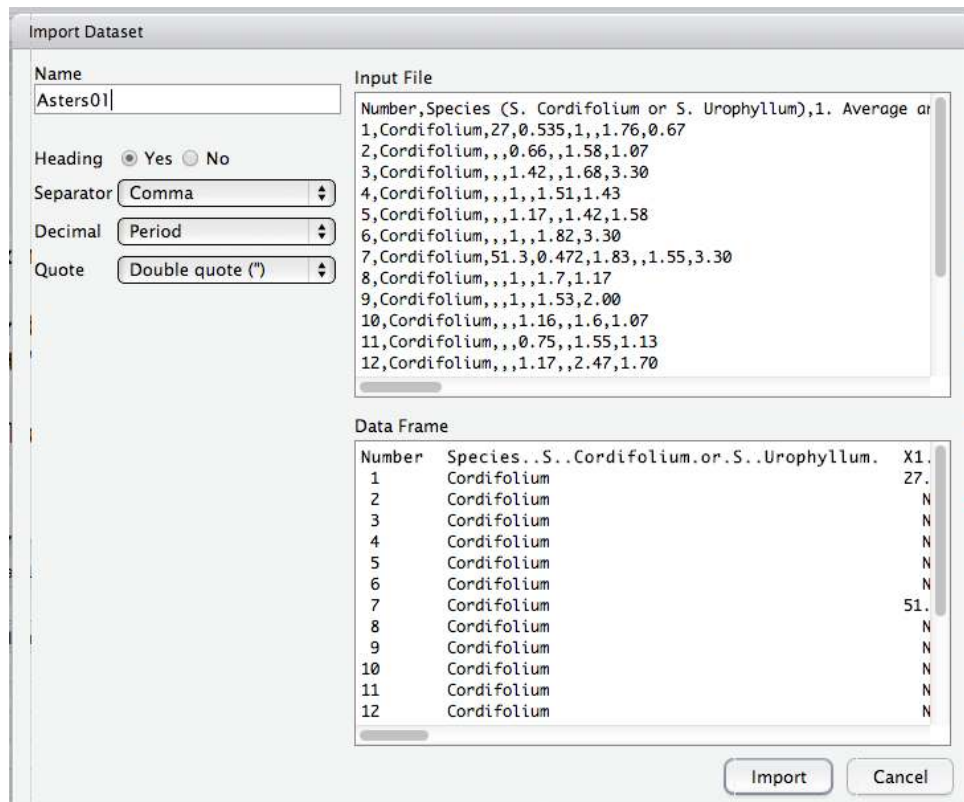
Importing the data into R

Now that the file is on the server, you can import it into R. This takes place in the **Environment** tab. Once there, choose **Import Dataset** and then **From Text File...**



The instructions are pretty clear from there, but here are some things to watch for:

- The default name for the data set is taken from the file name. If you used a very long file name, you will probably want to shorten this down. (But don't call it Data or something too generic either.) If the data are from the asters you have been tagging, perhaps call it Asters. If you are working with multiple data sets that deal with asters, add a bit more detail, perhaps Asters01 or some such thing.
- Be sure to select to use your first line as variable names (Heading = Yes).



The data set should now be ready for use in R.

A shortcut for Google Spreadsheets

The new [googlesheets](#) package provides a number of utilities for reading and writing between R and Google sheets. At the time of this writing, it is in active development and available via github.

Using R commands to read a data file

Even if you primarily use the RStudio interface to import data, it is good to know about the command line methods since these are required to import data into scripts, RMarkdown, and Rnw files. CSV files (and a few other types of files as well) can be read with

```
someData <- read.file("file.csv")
```

This can be used to read data directly from a URL as well. For example, here is some data from the US Census Bureau:

```
Population <- read.file(
  "https://www.census.gov/popest/data/national/totals/2012/files/NST_EST2012_ALLDATA.csv"
)
```

```
Loading required namespace: Rcurl
Reading data with read.csv()
```

```
dim(Population)
```

```
[1] 57 46
```

```
head(Population, 4)
```

	Sumlev	Region	Division	State	Name	CENSUS2010POP	ESTIMATESBASE2010
1	10	0	0	0	United States	308745538	308747508
2	20	1	0	0	Northeast Region	55317240	55317245
3	20	2	0	0	Midwest Region	66927001	66927489
4	20	3	0	0	South Region	114555744	114557147
	POPESTIMATE2010	POPESTIMATE2011	POPESTIMATE2012	NPOPCHG_2010	NPOPCHG_2011	NPOPCHG_2012	
1	309326225	311587816	313914040	578717	2261591	2326224	
2	55376926	55597646	55761091	59681	220720	163445	
3	66972135	67145089	67316297	44646	172954	171208	
4	114853800	116022230	117257221	296653	1168430	1234991	
	BIRTHS2010	BIRTHS2011	BIRTHS2012	DEATHS2010	DEATHS2011	DEATHS2012	NATURALINC2010
1	987836	3977039	3953593	598716	2490976	2513173	389120
2	160353	644224	631961	108678	465661	466970	51675
3	210660	839312	825776	141184	580842	580718	69476
4	373379	1503557	1510568	235600	956673	970844	137779
	NATURALINC2011	NATURALINC2012	INTERNATIONALMIG2010	INTERNATIONALMIG2011			
1	1486063	1440420		189597		775528	
2	178563	164991		48282		211281	
3	258470	245058		24794		100624	
4	546884	539724		71591		282102	
	INTERNATIONALMIG2012	DOMESTICMIG2010	DOMESTICMIG2011	DOMESTICMIG2012	NETMIG2010		
1		885804	0	0	0	189597	
2		221546	-38396	-161531	-220968	9886	
3		111790	-49082	-184696	-185118	-24288	
4		337769	86302	325546	353879	157893	
	NETMIG2011	NETMIG2012	RESIDUAL2010	RESIDUAL2011	RESIDUAL2012	Rbirth2011	Rbirth2012
1	775528	885804	0	0	0	12.81027	12.64135
2	49750	578	-1880	-7593	-2124	11.61030	11.35000
3	-84072	-73328	-542	-1444	-522	12.51610	12.28272
4	607648	691648	981	13898	3619	13.02480	12.95072
	Rdeath2011	Rdeath2012	RNATURALINC2011	RNATURALINC2012	RINTERNATIONALMIG2011		
1	8.023578	8.035701	4.786695	4.605646		2.498020	
2	8.392211	8.386769	3.218089	2.963234		3.807737	
3	8.661706	8.637692	3.854389	3.645032		1.500538	
4	8.287331	8.323442	4.737469	4.627274		2.443753	
	RINTERNATIONALMIG2012	RDOMESTICMIG2011	RDOMESTICMIG2012	Rnetmig2011	Rnetmig2012		
1		2.832299	0.000000	0.000000	2.4980205	2.83229855	
2		3.978960	-2.911135	-3.968579	0.8966018	0.01038086	
3		1.662782	-2.754247	-2.753475	-1.2537092	-1.09069231	
4		2.895832	2.820094	3.033949	5.2638466	5.92978076	

Many web sites provide data in csv format. Here some examples:

- <http://www.census.gov/> (Census Bureau data)
- <http://www.ncdc.noaa.gov/data-access> (NOAA Weather and climate data)
- <http://www.gapminder.org/data/> (Gapminder data)
- <http://introcs.cs.princeton.edu/java/data/> has a number of data sets, some in csv format, collected from other places on the internet.
- <http://www.exploredata.net/Downloads> has data from WHO, a genome expression study, and a microbiome study.

But be aware that some of these files might need to be cleaned up a bit before they are usable for statistics. Also, some internet files are very large and may take a while to download. Many sites will give an indication of the size of the data set so you know what you are in for. The better sites will include links to a code book (a description of all the variables, units used, how and when the data were collected, and any other information relevant to interpreting the data). Such a document is available for the population data loaded above. You can find it at <http://www.census.gov/popest/data/national/totals/2012/files/NST-EST2012-alldata.pdf>

Missing Data

The `na.strings` argument can be used to specify codes for missing values. The following can be useful, for example:

```
someData <- read.file('file.csv', na.strings='.')
someData <- read.file('file.csv', na.strings='-')
```

because SAS uses a period (.) to code missing data, and some csv exporters use '-'. By default R reads these as string data, which forces the entire variable to be of character type instead of numeric.

Importing Other Kinds of Data

Many R packages provide the ability to load data from special data files. If you have data in some other format, there may well be a package that makes it easy to load your data into R. For example, several packages (including `readxl`) provide the ability to read data directly from Excel files without first saving the data as a csv file. If you make frequent use of Excel spreadsheets, you may find this convenient. `rdrop2` provides the ability to manage data with Dropbox. And the `foreign` package provides functions to read data from a wide range of other statistical packages. But since these typically all know how to read and write csv files, learning a workflow that goes through CSV is a broadly applicable skill.

0.5 The Most Important Template

Most of what we will do in this chapter makes use of a single R template:

```
□ ( □ ~ □ , data = □ )
```

It is useful if we name the slots in this template:

```
goal ( [y] ~ [x] , data = [mydata] )
```

Actually, there are some variations on this template:

```
### Simpler version -- for just one variable
goal(~x, data = mydata)
### Fancier version:
goal(y ~ x | z, data = mydata)
### Unified version:
goal(formula, data = mydata)
```

To use the template (we'll call it the formula template because there is always a formula involved), you just need to know what goes in each slot. This can be determined by asking yourself two questions:

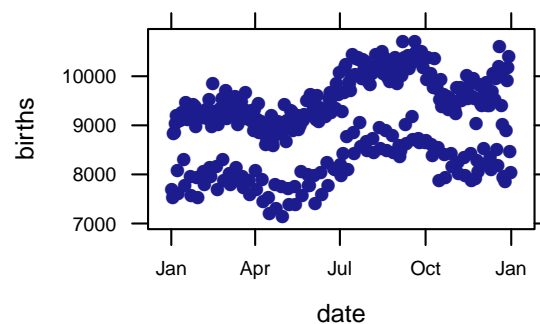
1. What do you want R to do?

- this determines what function to use (goal).

2. What must R know to do that?

- this determines the inputs to the function
- for describing data, must must identify *which data frame* and *which variable(s)*.

Let's try an example. Suppose we want to make this plot



1. What is our goal?

Our goal is to make a scatter plot. The function that does this is called `xypplot()`. That takes care of the first slot.

2. What does R need to know to do this?

It needs to know what data set to use, and which variables to use on the x and y axes. These data are in the `Births78` data set in the `mosaicData` package. Let's take a quick look at the data:

```
require(mosaicData) # load the package that contains our data set
head(Births78)
```

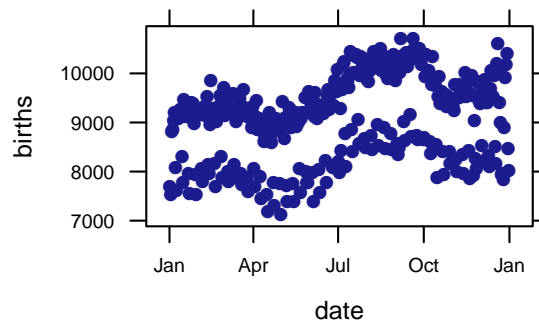
births-head

	date	births	dayofyear
1	1978-01-01	7701	1
2	1978-01-02	7527	2
3	1978-01-03	8825	3
4	1978-01-04	8859	4
5	1978-01-05	9043	5
6	1978-01-06	9208	6

We want the date on the x-axis and the number of births on the y axis, so the full command is

```
xypplot(births ~ date, data = Births78)
```

births-scatterplot



This same template can be used for a wide variety of graphical and numerical summaries. For example, to compute the mean number of births, we can change `xplot()` to `mean()` and provide `births` but not `date`:

```
mean(~births, data = Births78)

[1] 9132.162
```

Notice that when there is only one variable, it goes on the right side of the wiggle (code `~`). We'll see more examples of this template as we go along.

0.6 Manipulating your data

Creating a subset

The `filter()` command can be used to create subsets. The population data set we downloaded has population for states and various other regions. If we just want the states, we can select the items where the `State` variable is greater than 0. (Notice the double equals for testing equality.)

```
States <- filter(Population, State > 0)
dim(States)

[1] 52 46
```

filter

That two states too many. We can scan the list to see what else is in there.

```
States$name

NULL
```

extra-states

The two extras are Washington, DC and Puerto Rico.

Choosing specific columns

`filter()` chooses rows from a data frame. `select()` selects columns. This can be handy if you have a data set with many more variables than you are interested in. Let's pick just a handful from the [Population](#) data set.

```
States2 <- select(States, Name, POPESTIMATE2010, POPESTIMATE2011, POPESTIMATE2012)
```

select

Dropping Variables

Sometimes it is easier to think about dropping variables. We can use `select()` for this as well:

```
iris2 <- select(iris, -Sepal.Width, -Sepal.Length) # the minus sign means drop
head(iris2, 3)
```

	Petal.Length	Petal.Width	Species
1	1.4	0.2	setosa
2	1.4	0.2	setosa
3	1.3	0.2	setosa

Creating new variables

We can add a new variable to data set using `mutate()`:

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

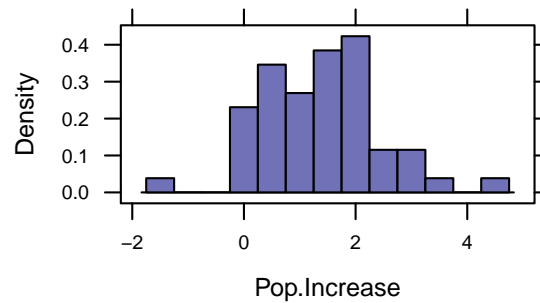
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa

```
iris3 <- mutate(iris,
                Sepal.Ratio = Sepal.Length / Sepal.Width,
                Petal.Ratio = Petal.Length / Petal.Width )
head(iris3,3)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.Ratio	Petal.Ratio
1	5.1	3.5	1.4	0.2	setosa	1.457143	7.0
2	4.9	3.0	1.4	0.2	setosa	1.633333	7.0
3	4.7	3.2	1.3	0.2	setosa	1.468750	6.5

```
States3 <- mutate(States2,
                  Pop.Increase = 100 * (POPESTIMATE2012 - POPESTIMATE2010)/POPESTIMATE2010 )
histogram( ~ Pop.Increase, data=States3, width=0.5,
           main="% Population increase (2010 to 2012)" )
```

% Population increase (2010 to 2012)



Generally, it is a good idea to keep raw data (like `Sepal.Length` and `Sepal.Width` in your data file, but let R do the computation of derived variables for you. Among other advantages, if you ever fix an error in a `Sepal.Length` measurement, you don't have to worry about remembering to also recompute the ratio. Furthermore, your R code documents how the derived value was computed.

Saving Data

`write.csv()` can be used to save data from R into csv formatted files. This can be useful for exporting to some other program.

```
write.csv(iris3, "iris3.csv")
```

writingData

Data can also be saved in native R format. Saving data sets (and other R objects) using `save()` has some advantages over other file formats:

- Complete information about the objects is saved, including attributes.
- Data saved this way takes less space and loads much more quickly.
- Multiple objects can be saved to and loaded from a single file.

The downside is that these files are only readable in R.

```
save(iris3, file = "iris3.rda") # the traditional file extension is rda for R native data.
load("iris3.rda") # loads previously saved data
```

savingData

For more on importing and exporting data, especially from other formats, see the *R Data Import/Export* manual available on CRAN.

Merging datasets

The `fusion1` data frame in the `fastR` package contains genotype information for a SNP (single nucleotide polymorphism) in the gene *TCF7L2*. The `pheno` data frame contains phenotypes (including type 2 diabetes case/control status) for an intersecting set of individuals. We can merge these together to explore the association between genotypes and phenotypes using `merge()`.

```
require(fastR)
head(fusion1, 3)
```

	id	marker	markerID	allele1	allele2	genotype	Adose	Cdose	Gdose	Tdose
1	9735	RS12255372	1	3	3	GG	0	0	2	0
2	10158	RS12255372	1	3	3	GG	0	0	2	0
3	9380	RS12255372	1	3	4	GT	0	0	1	1

```
head(pheno, 3)
```

	id	t2d	bmi	sex	age	smoker	chol	waist	weight	height	whr	sbp	dbp
1	1002	case	32.85994	F	70.76438	former	4.57	112.0	85.6	161.4	0.9867841	135	77
2	1009	case	27.39085	F	53.91896	never	7.32	93.5	77.4	168.1	0.9396985	158	88
3	1012	control	30.47048	M	53.86161	former	5.02	104.0	94.6	176.2	0.9327354	143	89

```
# merge fusion1 and pheno keeping only id's that are in both
fusion1m <- merge(fusion1, pheno, by.x = "id", by.y = "id", all.x = FALSE, all.y = FALSE)
head(fusion1m, 3)
```

	id	marker	markerID	allele1	allele2	genotype	Adose	Cdose	Gdose	Tdose	t2d
1	1002	RS12255372	1	3	3	GG	0	0	2	0	case
2	1009	RS12255372	1	3	3	GG	0	0	2	0	case
3	1012	RS12255372	1	3	3	GG	0	0	2	0	control

	bmi	sex	age	smoker	chol	waist	weight	height	whr	sbp	dbp
1	32.85994	F	70.76438	former	4.57	112.0	85.6	161.4	0.9867841	135	77
2	27.39085	F	53.91896	never	7.32	93.5	77.4	168.1	0.9396985	158	88
3	30.47048	M	53.86161	former	5.02	104.0	94.6	176.2	0.9327354	143	89

In this case, since the values are the same for each data frame, we could collapse `by.x` and `by.y` to `by` and collapse `all.x` and `all.y` to `all`. The first of these specifies which column(s) to use to identify matching cases. The second indicates whether cases in one data frame that do not appear in the other should be kept (`TRUE`) or dropped (filling in `NA` as needed) or dropped from the merged data frame.

Now we are ready to begin our analysis.

```
tally(~t2d + genotype, fusion1m)
```

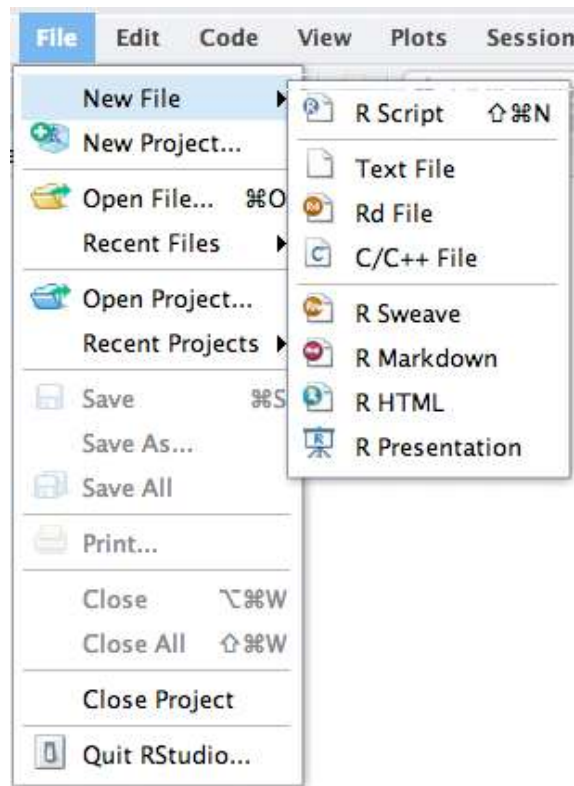
fusion1-xtabs

	genotype		
t2d	GG	GT	TT
case	737	375	48
control	835	309	27

0.7 Using R Markdown

Although you can export plots from RStudio for use in other applications, there is another way of preparing documents that has many advantages. RStudio provides several ways to create documents that include text, R code, R output, graphics, even mathematical notation all in one document. The simplest of these is R Markdown.

To create a new R Markdown document, go to “File”, “New”, then “R Markdown”:

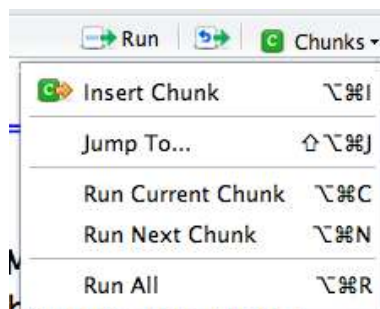


When you do this, a file editing pane will open with a template inserted. If you click on “Knit HTML”, RStudio will turn this into an HTML file and display it for you. Give it a try. You will be asked to name your file if you haven’t already done so. If you are using the RStudio server in a browser, then your file will live on the server (“in the cloud”) rather than on your computer.

If you look at the template file you will see that the file has two kinds of sections. Some of this file is just normal text (with some extra symbols to make things bold, add in headings, etc.) You can get a list of all of these mark up options by selecting the “Markdown Quick Reference” in the question mark menu.



The second type of section is an R code chunk. These are colored differently to make them easier to see. You can insert a new code chunk by selecting “Insert Chunk” from the “Chunks” menu:



(You can also type ````\{r\}` to begin and ````` to end the code chunk if you would rather type.) You can put any R code in these code chunks and the results (text output or graphics) as well as the R code will be displayed in your HTML file.

There are options to do things like (a) run R code without displaying it, (b) run R code without displaying the output, (c) controlling size of plots, etc., etc. But for starting out, this is really all you need to know.

R Markdown files must be self-contained

R Markdown files do not have access to things you have done in your console. (This is good, else your document would change based on things not in the file.) This means that you must explicitly load data, and require packages *in the R Markdown file* in order to use them. In this class, this means that most of your R Markdown files will have a chunk near the beginning that includes

```
require(mosaic) # load the mosaic package
require(Lock5withR) # get data sets from the book
```

The **mosaic** package provides some templates that are available if you choose “from template” when creating an RMarkdown file in RStudio. Among other things, this will insert the code required to load the **mosaic** package, change some default settings, and include a reminder to load any additional packages you will be using.

Output formats

RStudio makes it easy to generate HTML, PDF, or Word documents from your RMarkdown. Just remember that if you edit any of these files after you generate them with RMarkdown, then you will need to redo those edits if you ever go back and change the RMarkdown file, but if you change the RMarkdown file, one click will generate the new HTML, PDF, or Word document. (There are even ways to get it to generate all three in one go; see the `render()` function in the **rmarkdown** package.) So it is best to keep you editing to the RMarkdown document as much as possible.

0.8 Statistics: Answering Questions With Data

This is a course primarily about statistics, but what exactly is *statistics*? In other words, what is this course about?¹

Here are some definitions of statistics from other people:

- a collection of procedures and principles for gaining information in order to make decisions when faced with uncertainty (J. Utts [Utt05]),
- a way of taming uncertainty, of turning raw data into arguments that can resolve profound questions (T. Amabile [fMA89]),
- the science of drawing conclusions from data with the aid of the mathematics of probability (S. Garfunkel [fMA86]),
- the explanation of variation in the context of what remains unexplained (D. Kaplan [Kap09]),

¹ As we will see, the words *statistic* and *statistics* get used in more than one way. More on that later.

- the mathematics of the collection, organization, and interpretation of numerical data, especially the analysis of a population's characteristics by inference from sampling (American Heritage Dictionary [AmH82]).

Here's a simpler definition:

Statistics is the science of answering questions with data.

This definition gets at two important elements of the longer definitions above:

Data – the raw material

Data are the raw material for doing statistics. We will learn more about different types of data, how to collect data, and how to summarize data as we go along.

Information – the goal

The goal of doing statistics is to gain some information or to make a decision – that is, to answer some question.

Statistics is useful because it helps us answer questions like the following: ²

- Which of two treatment plans leads to the best clinical outcomes?
- Are men or women more successful at quitting smoking? And does it matter which smoking cessation program they use?
- Is my cereal company complying with regulations about the amount of cereal in its cereal boxes?

In this sense, statistics is a science – a method for obtaining new knowledge. Our simple definition is light on describing the context in which this takes place. So let's add two more important aspects of statistics.

Uncertainty – the context

The tricky thing about statistics is the uncertainty involved. If we measure one box of cereal, how do we know that all the others are similarly filled? If every box of cereal were identical and every measurement perfectly exact, then one measurement would suffice. But the boxes may differ from one another, and even if we measure the same box multiple times, we may get different answers to the question *How much cereal is in the box?*

So we need to answer questions like *How many boxes should we measure?* and *How many times should we measure each box?* Even so, there is no answer to these questions that will give us absolute certainty. So we need to answer questions like *How sure do we need to be?*

Probability – the tool

In order to answer a question like *How sure do we need to be?*, we need some way of measuring our level of certainty. This is where mathematics enters into statistics. Probability is the area of mathematics that deals with reasoning about uncertainty.

²The opening pages of each chapter of our book include many more questions.

0.9 A First Example: The Lady Tasting Tea

There is a famous story about a lady who claimed that tea with milk tasted different depending on whether the milk was added to the tea or the tea added to the milk. The story is famous because of the setting in which she made this claim. She was attending a party in Cambridge, England, in the 1920s. Also in attendance were a number of university dons and their wives. The scientists in attendance scoffed at the woman and her claim. What, after all, could be the difference?

All the scientists but one, that is. Rather than simply dismiss the woman's claim, he proposed that they decide how one should *test* the claim. The tenor of the conversation changed at this suggestion, and the scientists began to discuss how the claim should be tested. Within a few minutes cups of tea with milk had been prepared and presented to the woman for tasting.

Let's take this simple example as a prototype for a statistical study. What steps are involved?

1. Determine the question of interest.

Just what is it we want to know? It may take some effort to make a vague idea precise. The precise questions may not exactly correspond to our vague questions, and the very exercise of stating the question precisely may modify our question. Sometimes we cannot come up with any way to answer the question we really want to answer, so we have to live with some other question that is not exactly what we wanted but is something we can study and will (we hope) give us some information about our original question.

In our example this question seems fairly easy to state: Can the lady tell the difference between the two tea preparations? But we need to refine this question. For example, are we asking if she *always* correctly identifies cups of tea or merely if she does better than we could do ourselves (by guessing)?

2. Determine the **population**.

Just who or what do we want to know about? Are we only interested in this one woman or women in general or only women who claim to be able to distinguish tea preparations?

3. Select **measurements**.

We are going to need some data. We get our data by making some measurements. These might be physical measurements with some device (like a ruler or a scale). But there are other sorts of measurements too, like the answer to a question on a form. Sometimes it is tricky to figure out just what to measure. (How do we measure happiness or intelligence, for example?) Just how we do our measuring will have important consequences for the subsequent statistical analysis. The recorded values of these measurements are called **variables** (because the values vary from one individual to another).

In our example, a measurement may consist of recording for a given cup of tea whether the woman's claim is correct or incorrect.

4. Determine the **sample**.

Usually we cannot measure every individual in our population; we have to select some to measure. But how many and which ones? These are important questions that must be answered. Generally speaking, bigger is better, but it is also more expensive. Moreover, no size is large enough if the sample is selected inappropriately.

Suppose we gave the lady one cup of tea. If she correctly identifies the mixing procedure, will we be convinced of her claim? She might just be guessing; so we should probably have her taste more than one cup. Will we be convinced if she correctly identifies 5 cups? 10 cups? 50 cups?

What if she makes a mistake? If we present her with 10 cups and she correctly identifies 9 of the 10, what will we conclude? A success rate of 90% is, it seems, much better than just guessing, and anyone can make a mistake now and then. But what if she correctly identifies 8 out of 10? 80 out of 100?

And how should we prepare the cups? Should we make 5 each way? Does it matter if we tell the woman that there are 5 prepared each way? Should we flip a coin to decide even if that means we might end up with 3 prepared one way and 7 the other way? Do any of these differences matter?

5. Make and record the measurements.

Once we have the design figured out, we have to do the legwork of data collection. This can be a time-consuming and tedious process. In the case of the lady tasting tea, the scientists decided to present her with ten cups of tea which were quickly prepared. A study of public opinion may require many thousands of phone calls or personal interviews. In a laboratory setting, each measurement might be the result of a carefully performed laboratory experiment.

6. Organize the data.

Once the data have been collected, it is often necessary or useful to organize them. Data are typically stored in spreadsheets or in other formats that are convenient for processing with statistical packages. Very large data sets are often stored in databases.

Part of the organization of the data may involve producing graphical and numerical summaries of the data. These summaries may give us initial insights into our questions or help us detect errors that may have occurred to this point.

7. Draw conclusions from data.

Once the data have been collected, organized, and analyzed, we need to reach a conclusion. Do we believe the woman's claim? Or do we think she is merely guessing? How sure are we that this conclusion is correct?

Eventually we will learn a number of important and frequently used methods for drawing inferences from data. More importantly, we will learn the basic framework used for such procedures so that it should become easier and easier to learn new procedures as we become familiar with the framework.

8. Produce a report.

Typically the results of a statistical study are reported in some manner. This may be as a refereed article in an academic journal, as an internal report to a company, or as a solution to a problem on a homework assignment. These reports may themselves be further distilled into press releases, newspaper articles, advertisements, and the like. The mark of a good report is that it provides the essential information about each of the steps of the study.

As we go along, we will learn some of the standard terminology and procedures that you are likely to see in basic statistical reports and will gain a framework for learning more.

At this point, you may be wondering who the innovative scientist was and what the results of the experiment were. The scientist was R. A. Fisher, who first described this situation as a pedagogical example in his 1925 book on statistical methodology [Fis25]. Fisher developed statistical methods that are among the most important and widely used methods to this day, and most of his applications were biological.

0.10 Coins and Cups

You might also be curious about how the experiment came out. How many cups of tea were prepared? How many did the woman correctly identify? What was the conclusion?

Fisher never says. In his book he is interested in the method, not the particular results. But let's suppose we decide to test the lady with ten cups of tea. We'll flip a coin to decide which way to prepare the cups. If we flip a head, we will pour the milk in first; if tails, we put the tea in first. Then we present the ten cups to the lady and have her state which ones she thinks were prepared each way.

It is easy to give her a score (9 out of 10, or 7 out of 10, or whatever it happens to be). It is trickier to figure out what to do with her score. Even if she is just guessing and has no idea, she could get lucky and get quite a few correct – maybe even all 10. But how likely is that?

Let's try an experiment. I'll flip 10 coins. You guess which are heads and which are tails, and we'll see how you do.

⋮

Comparing with your classmates, we will undoubtedly see that some of you did better and others worse.

Now let's suppose the lady gets 9 out of 10 correct. That's not perfect, but it is better than we would expect for someone who was just guessing. On the other hand, it is not impossible to get 9 out of 10 just by guessing. So here is Fisher's great idea: Let's figure out how hard it is to get 9 out of 10 by guessing. If it's not so hard to do, then perhaps that's just what happened, so we won't be too impressed with the lady's tea tasting ability. On the other hand, if it is really unusual to get 9 out of 10 correct by guessing, then we will have some evidence that she must be able to tell something.

But how do we figure out how unusual it is to get 9 out of 10 just by guessing? We'll learn another method later, but for now, let's just flip a bunch of coins and keep track. If the lady is just guessing, she might as well be flipping a coin.

So here's the plan. We'll flip 10 coins. We'll call the heads correct guesses and the tails incorrect guesses. Then we'll flip 10 more coins, and 10 more, and 10 more, and That would get pretty tedious. Fortunately, computers are good at tedious things, so we'll let the computer do the flipping for us using a tool in the **mosaic** package. This package is already installed in our RStudio server. If you are running your own installation of R you can install **mosaic** using the following command:

```
install.packages("mosaic")
```

install-mosaic

The **rflip()** function can flip one coin

```
require(mosaic)
rflip()
```

flip1coin

```
Flipping 1 coin [ Prob(Heads) = 0.5 ] ...
```

```
H
```

```
Number of Heads: 1 [Proportion Heads: 1]
```

or a number of coins

```
rflip(10)
```

flip10coins

```
Flipping 10 coins [ Prob(Heads) = 0.5 ] ...
```

```
T H H H T T H H T T
```

```
Number of Heads: 5 [Proportion Heads: 0.5]
```

and show us the results.

Typing **rflip(10)** a bunch of times is almost as tedious as flipping all those coins. But it is not too hard to tell R to **do()** this a bunch of times.

```
do(2) * rflip(10)
```

flip2

	n	heads	tails	prop
1	10	6	4	0.6
2	10	6	4	0.6

Let's get R to `do()` it for us 10,000 times and make a table of the results.

```
results <- do(10000) * rflip(10)
table(results$heads)
```

flip4

0	1	2	3	4	5	6	7	8	9	10
5	102	467	1203	2048	2470	2035	1140	415	108	7

```
perctable(results$heads) # the table in percents
```

flip5

0	1	2	3	4	5	6	7	8	9	10
0.05	1.02	4.67	12.03	20.48	24.70	20.35	11.40	4.15	1.08	0.07

```
proptable(results$heads) # the table in proportions (i.e., decimals)
```

0	1	2	3	4	5	6	7	8	9	10
0.0005	0.0102	0.0467	0.1203	0.2048	0.2470	0.2035	0.1140	0.0415	0.0108	0.0007

We could also use `tally()` for this.

```
tally(~heads, data = results)
```

tally

0	1	2	3	4	5	6	7	8	9	10
5	102	467	1203	2048	2470	2035	1140	415	108	7

```
tally(~heads, data = results, format = "percent")
```

0	1	2	3	4	5	6	7	8	9	10
0.05	1.02	4.67	12.03	20.48	24.70	20.35	11.40	4.15	1.08	0.07

```
tally(~heads, data = results, format = "proportion")
```

0	1	2	3	4	5	6	7	8	9	10
0.0005	0.0102	0.0467	0.1203	0.2048	0.2470	0.2035	0.1140	0.0415	0.0108	0.0007

You might be surprised to see that the number of correct guesses is exactly 5 (half of the 10 tries) only 25% of the time. But most of the results are quite close to 5 correct. 67% of the results are 4, 5, or 6, for example. And 1% of the results are between 3 and 7 (inclusive). But getting 8 correct is a bit unusual, and getting 9 or 10 correct is even more unusual.

So what do we conclude? It is possible that the lady could get 9 or 10 correct just by guessing, but it is not very likely (it only happened in about 1.2% of our simulations). So *one of two things must be true*:

- The lady got unusually “lucky”, or
- The lady is not just guessing.

Although Fisher did not say how the experiment came out, others have reported that the lady correctly identified all 10 cups! [Sal01]

This same reasoning can be applied to answer a wide range of questions that have a similar form. For example, the question of whether dogs can smell cancer could be answered essentially the same way (although it would be a bit more involved than preparing tea and presenting cups to the Lady).

Collecting Data

1.1 The Structure of Data

Cases and Variables

Data sets in R are usually stored as **data frames** in a rectangular arrangement with rows corresponding to observational units and columns corresponding to variables. A number of data sets are built into R and its packages. The package for our text is **Lock5withR** which comes with a number of data sets.

```
require(Lock5withR) # Tell R to use the package for our text book
data(StudentSurvey) # load the StudentSurvey data set
```

Imagine data as a 2-dimensional structure (like a spreadsheet).

- Rows correspond to **observational units** (people, animals, plants, or other objects we are collecting data about).
- Columns correspond to **variables** (measurements collected on each observational unit).
- At the intersection of a row and a column is the **value** of the variable for a particular observational unit.

Observational units go by many names, depending on the kind of thing being studied. Popular names include subjects, individuals, and cases. Whatever you call them, it is important that you always understand what your observational units are.

Let's take a look at the data frame for the Student Survey example in the text. If we type the name of the data set, R will display it in its entirety for us. However, **StudentSurvey** is a larger data set, so it is more useful to look at some sort of summary or subset of the data.

Table 1.1

```
head(StudentSurvey) # first six cases of the data set
```

Table1.1

	Year	Gender	Smoke	Award	HigherSAT	Exercise	TV	Height	Weight	Siblings	BirthOrder
1	Senior	M	No	Olympic	Math	10	1	71	180	4	4
2	Sophomore	F	Yes	Academy	Math	4	7	66	120	2	2
3	FirstYear	M	No	Nobel	Math	14	5	72	208	2	1
4	Junior	M	No	Nobel	Math	3	1	63	110	1	1
5	Sophomore	F	No	Nobel	Verbal	3	3	65	150	1	1
6	Sophomore	F	No	Nobel	Verbal	5	4	65	114	2	2

	VerbalSAT	MathSAT	SAT	GPA	Pulse	Piercings	Sex
1	540	670	1210	3.13	54	0	Male
2	520	630	1150	2.50	66	3	Female
3	550	560	1110	2.55	130	0	Male
4	490	630	1120	3.10	78	0	Male
5	720	450	1170	2.70	40	6	Female
6	600	550	1150	3.20	80	4	Female

We can easily classify variables as either **categorical** or **quantitative** by studying the result of `head()`, but there are some summaries of the data set which reveal such information.

```
str(StudentSurvey) # structure of the data set
```

Data1.1

```
'data.frame': 362 obs. of 18 variables:
 $ Year      : Factor w/ 5 levels "", "FirstYear",...: 4 5 2 3 5 5 2 5 3 2 ...
 $ Gender    : Factor w/ 2 levels "F","M": 2 1 2 2 1 1 1 2 1 1 ...
 $ Smoke     : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 1 1 1 1 ...
 $ Award     : Factor w/ 3 levels "Academy","Nobel",...: 3 1 2 2 2 2 3 3 2 2 ...
 $ HigherSAT : Factor w/ 3 levels "", "Math", "Verbal": 2 2 2 2 3 3 2 2 3 2 ...
 $ Exercise  : num 10 4 14 3 3 5 10 13 3 12 ...
 $ TV        : int 1 7 5 1 3 4 10 8 6 1 ...
 $ Height    : int 71 66 72 63 65 65 66 74 61 60 ...
 $ Weight    : int 180 120 208 110 150 114 128 235 NA 115 ...
 $ Siblings  : int 4 2 2 1 1 2 1 1 2 7 ...
 $ BirthOrder: int 4 2 1 1 1 2 1 1 2 8 ...
 $ VerbalSAT : int 540 520 550 490 720 600 640 660 550 670 ...
 $ MathSAT   : int 670 630 560 630 450 550 680 710 550 700 ...
 $ SAT       : int 1210 1150 1110 1120 1170 1150 1320 1370 1100 1370 ...
 $ GPA       : num 3.13 2.5 2.55 3.1 2.7 3.2 2.77 3.3 2.8 3.7 ...
 $ Pulse     : int 54 66 130 78 40 80 94 77 60 94 ...
 $ Piercings : int 0 3 0 0 6 4 8 0 7 2 ...
 $ Sex       : Factor w/ 2 levels "Female","Male": 2 1 2 2 1 1 1 2 1 1 ...
```

```
summary(StudentSurvey) # summary of each variable
```

Year	Gender	Smoke	Award	HigherSAT	Exercise
: 2	F:169	No :319	Academy: 31	: 7	Min. : 0.000
FirstYear: 94	M:193	Yes: 43	Nobel :149	Math :205	1st Qu.: 5.000
Junior : 35			Olympic:182	Verbal:150	Median : 8.000
Senior : 36					Mean : 9.054
Sophomore:195					3rd Qu.:12.000
					Max. :40.000
					NA's :1

TV	Height	Weight	Siblings	BirthOrder
Min. : 0.000	Min. :59.00	Min. : 95.0	Min. :0.000	Min. :1.00

1st Qu.: 3.000	1st Qu.:65.00	1st Qu.:138.0	1st Qu.:1.000	1st Qu.:1.00
Median : 5.000	Median :68.00	Median :155.0	Median :1.000	Median :2.00
Mean : 6.504	Mean :68.42	Mean :159.8	Mean :1.727	Mean :1.83
3rd Qu.: 9.000	3rd Qu.:71.00	3rd Qu.:180.0	3rd Qu.:2.000	3rd Qu.:2.00
Max. :40.000	Max. :83.00	Max. :275.0	Max. :8.000	Max. :8.00
NA's :1	NA's :7	NA's :5		NA's :3
VerbalSAT	MathSAT	SAT	GPA	Pulse
Min. :390.0	Min. :400.0	Min. : 800	Min. :2.000	Min. : 35.00
1st Qu.:550.0	1st Qu.:560.0	1st Qu.:1130	1st Qu.:2.900	1st Qu.: 62.00
Median :600.0	Median :610.0	Median :1200	Median :3.200	Median : 70.00
Mean :594.2	Mean :609.4	Mean :1204	Mean :3.158	Mean : 69.57
3rd Qu.:640.0	3rd Qu.:650.0	3rd Qu.:1270	3rd Qu.:3.400	3rd Qu.: 77.75
Max. :800.0	Max. :800.0	Max. :1550	Max. :4.000	Max. :130.00
			NA's :17	
Piercings	Sex			
Min. : 0.000	Female:169			
1st Qu.: 0.000	Male :193			
Median : 0.000				
Mean : 1.673				
3rd Qu.: 3.000				
Max. :10.000				
NA's :1				

Here are some more summaries:

```
nrow(StudentSurvey) # number of rows

[1] 362

ncol(StudentSurvey) # number of columns

[1] 18

dim(StudentSurvey) # number of rows and columns

[1] 362 18
```

Data1.1b

Many of the datasets in R have useful help files that describe the data and explain how they were collected or give references to the original studies. You can access this information for the [AllCountries](#) data set by typing

```
?StudentSurvey
```

Data1.1c

We'll learn how to make more customized summaries (numerical and graphical) soon. For now, it is only important to observe how the organization of data in R reflects the observational units and variables in the data set.

This is important if you want to construct your own data set (in Excel or a google spreadsheet, for example) that you will later import into R. You want to be sure that the structure of your spread sheet uses rows and columns in this same way, and that you don't put any extra stuff into the spread sheet. It is a good idea to

include an extra row at the top which names the variables. Take a look at Chapter 0 to learn how to get the data from Excel into R.

Categorical and Quantitative Variables

categorical variable a variable that places observational units into one of two or more categories (examples: color, sex, case/control status, species, etc.)

These can be further sub-divided into ordinal and nominal variables. If the categories have a natural and meaningful order, we will call them **ordered** or **ordinal** variables. Otherwise, they are **nominal** variables.

quantitative variable a variable that records measurements along some scale (examples: weight, height, age, temperature) or counts something (examples: number of siblings, number of colonies of bacteria, etc.)

Quantitative variables can be **continuous** or **discrete**. Continuous variables can (in principle) take on any real-number value in some range. Values of discrete variables are limited to some list and “in-between values” are not possible. Counts are a good example of discrete variables.

Investigating Variables and Relationships between Variables

```
head(AllCountries)
```

Data1.2

	Country	Code	LandArea	Population	Energy	Rural	Military	Health	HIV	Internet
1	Afghanistan	AFG	652230	29.021	NA	76.0	4.4	3.7	NA	1.7
2	Albania	ALB	27400	3.143	2088	53.3	NA	8.2	NA	23.9
3	Algeria	ALG	2381740	34.373	37069	34.8	13.0	10.6	0.1	10.2
4	American Samoa	ASA	200	0.066	NA	7.7	NA	NA	NA	NA
5	Andorra	AND	470	0.084	NA	11.1	NA	21.3	NA	70.5
6	Angola	ANG	1246700	18.021	10972	43.3	NA	6.8	2.0	3.1

	Developed	BirthRate	ElderlyPop	LifeExpectancy	C02	GDP	Cell
1	NA	46.5	2.2	43.9	0.02503483	501.4709	37.80711
2	1	14.6	9.3	76.6	1.31285501	3678.2317	141.92896
3	1	20.8	4.6	72.4	3.23296040	4494.8867	92.42126
4	NA	NA	NA	NA	NA	NA	NA
5	NA	10.4	NA	NA	6.52783463	NA	77.17642
6	1	42.9	2.5	47.0	1.35108829	4422.5428	46.68924

	Electricity	kwhPerCap
1	NA	<NA>
2	1747.0980	Under 2500
3	970.9825	Under 2500
4	NA	<NA>
5	NA	<NA>
6	202.1545	Under 2500


```
summary(AllCountries)
```

	Country	Code	LandArea	Population
Afghanistan	: 1	: 3	Min. : 2	Min. : 0.0200
Albania	: 1	AFG : 1	1st Qu.: 10830	1st Qu.: 0.7728
Algeria	: 1	ALB : 1	Median : 94080	Median : 5.6135
American Samoa	: 1	ALG : 1	Mean : 608120	Mean : 31.4849
Andorra	: 1	AND : 1	3rd Qu.: 446300	3rd Qu.: 20.5835
Angola	: 1	ANG : 1	Max. : 16376870	Max. : 1324.6550

```

(Other)      :207  (Other):205      NA's      :1
  Energy      Rural      Military      Health      HIV
Min.   :    159  Min.   : 0.00  Min.   : 0.000  Min.   : 0.70  Min.   : 0.100
1st Qu.:   5252  1st Qu.:22.90  1st Qu.: 3.800  1st Qu.: 8.00  1st Qu.: 0.100
Median :  17478  Median :40.40  Median : 5.850  Median :11.30  Median : 0.400
Mean   :  86312  Mean   :42.13  Mean   : 8.277  Mean   :11.22  Mean   : 1.977
3rd Qu.:  52486  3rd Qu.:63.20  3rd Qu.:12.175  3rd Qu.:14.45  3rd Qu.: 1.300
Max.   :2283722  Max.   :89.60  Max.   :29.300  Max.   :26.10  Max.   :25.900
NA's   :   77      NA's   :115      NA's   :26      NA's   :68
  Internet    Developed    BirthRate    ElderlyPop    LifeExpectancy
Min.   : 0.20  Min.   :1.000  Min.   : 8.20  Min.   : 1.000  Min.   :43.90
1st Qu.: 5.65  1st Qu.:1.000  1st Qu.:12.10  1st Qu.: 3.400  1st Qu.:62.80
Median :22.80  Median :1.000  Median :19.40  Median : 5.400  Median :71.90
Mean   :28.96  Mean   :1.763  Mean   :22.02  Mean   : 7.473  Mean   :68.94
3rd Qu.:48.15  3rd Qu.:3.000  3rd Qu.:28.90  3rd Qu.:11.600  3rd Qu.:76.03
Max.   :90.50  Max.   :3.000  Max.   :53.50  Max.   :21.400  Max.   :82.80
NA's   :   14  NA's   :78      NA's   :16      NA's   :22      NA's   :17
    C02      GDP      Cell      Electricity
Min.   : 0.02262  Min.   : 192.1  Min.   : 1.238  Min.   : 35.68
1st Qu.: 0.61765  1st Qu.: 1252.7  1st Qu.: 59.206  1st Qu.: 800.32
Median : 2.73694  Median : 4408.8  Median : 93.696  Median : 2237.51
Mean   : 5.08557  Mean   :11298.4  Mean   : 91.093  Mean   : 4109.13
3rd Qu.: 7.01656  3rd Qu.:12431.0  3rd Qu.:121.160  3rd Qu.:5824.24
Max.   :49.05058  Max.   :105437.7  Max.   :206.429  Max.   :51259.19
NA's   :   15  NA's   :40      NA's   :12      NA's   :78
    kwhPerCap
Under 2500 :73
2500 - 5000:21
Over 5000  :41
NA's       :78

```

```
AllCountries[86, ]
```

```

  Country Code LandArea Population Energy Rural Military Health HIV Internet Developed
86 Iceland  ISL  100250      0.317  5255    7.7      0.1   13.1 0.3   90.5         3
  BirthRate ElderlyPop LifeExpectancy      C02      GDP      Cell Electricity kwhPerCap
86      15.2      11.7          81.3 7.024063 39616.84 109.662   51259.19 Over 5000

```

Using Data to Answer a Question

response variable a variable we are trying to predict or explain

explanatory variable a variable used to predict or explain a response variable

1.2 Sampling from a Population

Samples from Populations

population the collection of animals, plants, objects, etc. that we want to know about

sample the (smaller) set of animals, plants, objects, etc. about which we have data

parameter a number that describes a population or model.

statistic a number that describes a sample.

Much of statistics centers around this question:

What can we learn about a population from a sample?

Sampling Bias

Often we are interested in knowing (approximately) the value of some parameter. A statistic used for this purpose is called an **estimate**. For example, if you want to know the mean length of the tails of lemurs (that's a *parameter*), you might take a sample of lemurs and measure their tails. The mean length of the tails of the lemurs in your sample is a *statistic*. It is also an *estimate*, because we use it to estimate the parameter.

Statistical estimation methods attempt to

- reduce **bias**, and
- increase **precision**.

bias the systematic tendency of sample estimates to either overestimate or underestimate population parameters; that is, a *systematic tendency to be off in a particular direction*.

precision the measure of how close estimates are to the thing being estimated (called the **estimand**).

Simple Random Sample

Sampling is the process of selecting a sample. Statisticians use **random samples**

- to avoid (or at least reduce) **bias**, and
- so they can quantify **sampling variability** (the amount samples differ from each other), which in turn allows us to quantify precision.

The simplest kind of random sample is called a **simple random sample** (aren't statisticians clever about naming things?). A simple random sample is equivalent to putting all individuals in the population into a big hat, mixing thoroughly, and selecting some out of the hat to be in the sample. In particular, in a simple random sample, *every individual has an equal chance to be in the sample*, in fact, every subset of the population of a fixed size has an equal chance to be in the sample.

Other sampling methods include

convenience sampling using whatever individuals are easy to obtain

This is usually a terrible idea. If the convenient members of the population differ from the inconvenient members, then the sample will not be representative of the population.

volunteer sampling using people who volunteer to be in the sample

This is usually a terrible idea. Most likely the volunteers will differ in some ways from the non-volunteers, so again the sample will not be representative of the population.

systematic sampling sampling done in some systematic way (every tenth unit, for example).

This can sometimes be a reasonable approach.

stratified sampling sampling separately in distinct sub-populations (called *strata*)

This is more complicated (and sometimes necessary) but fine as long as the sampling methods in each stratum are good and the analysis takes the sampling method into account.

Example 1.15

```
sample(AllCountries, 5)
```

Example1.15

	Country	Code	LandArea	Population	Energy	Rural	Military	Health	HIV	Internet
211	Yemen, Rep.	YEM	527970	22.917	7478	69.4	NA	4.3	NA	1.6
168	Seychelles	SEY	460	0.087	NA	45.7	3.1	10.1	NA	39.0
2	Albania	ALB	27400	3.143	2088	53.3	NA	8.2	NA	23.9
123	Marshall Islands	MHL	180	0.060	NA	28.9	NA	14.6	NA	3.7
68	Gabon	GAB	257670	1.448	2073	15.0	NA	6.6	5.3	6.2
	Developed	BirthRate	ElderlyPop	LifeExpectancy	C02	GDP	Cell			
211	1	36.8	2.4	62.9	1.033494	NA	46.086659			
168	NA	17.8	NA	73.2	7.843760	10824.724	135.899451			
2	1	14.6	9.3	76.6	1.312855	3678.232	141.928961			
123	NA	NA	NA	NA	1.872334	3015.209	7.032089			
68	1	27.3	4.3	60.4	1.704158	8642.804	106.943844			
	Electricity	kwhPerCap	orig.ids							
211	218.8337	Under 2500	211							
168	NA	<NA>	168							
2	1747.0980	Under 2500	2							
123	NA	<NA>	123							
68	922.4955	Under 2500	68							

1.3 Experiments and Observational Studies

Confounding Variables

Table 1.2

```
head(LifeExpectancyVehicles, 10)
```

Table1.2

	Year	LifeExpectancy	Vehicles
1	1970	70.8	108.4
2	1971	71.1	113.0
3	1972	71.2	118.8
4	1973	71.4	125.7
5	1974	72.0	129.9
6	1975	72.6	132.9
7	1976	72.9	138.5
8	1977	73.3	142.1
9	1978	73.5	148.4
10	1979	73.9	151.9

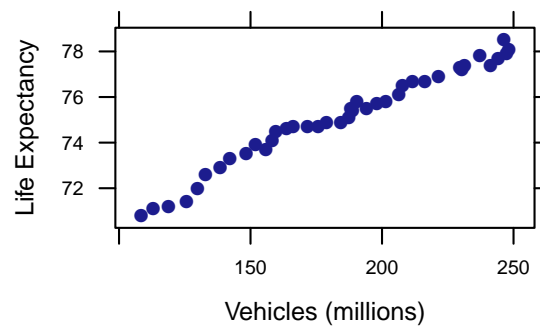
```
sub <- filter(LifeExpectancyVehicles, Year%%4 == 2)
sub
```

	Year	LifeExpectancy	Vehicles
1	1970	70.8	108.4
2	1974	72.0	129.9
3	1978	73.5	148.4
4	1982	74.5	159.6
5	1986	74.7	175.7
6	1990	75.4	188.8
7	1994	75.7	198.0
8	1998	76.7	211.6
9	2002	77.3	229.6
10	2006	77.7	244.2

Figure 1.2

```
xypplot(LifeExpectancy ~ Vehicles, xlab = "Vehicles (millions)", ylab = "Life Expectancy",
        data = LifeExpectancyVehicles)
```

Figure1.2



Observational Studies vs Experiments

Statisticians use the word experiment to mean something very specific. *In an **experiment**, the researcher determines the values of one or more (explanatory) variables, typically by random assignment.* If there is no such assignment by the researcher, the study is an **observational study**.

2

Describing Data

In this chapter we discuss graphical and numerical summaries of data.

2.1 Categorical Variables

Let us investigate categorical variables in R by taking a look at the data set for the One True Love survey. Notice that the data set is not readily available in our textbook's package. However, the authors do provide us with the necessary information to create our own data spreadsheet (in either Excel or Google) and import it into R. (See Chapter 0 for instructions.)

```
OneTrueLove <- read.file("OneTrueLove.csv")
```

Data2.1

Reading data with readr::read_csv()

Alternatively, we can read from a URL like this

```
OneTrueLove2 <- read.file("https://raw.githubusercontent.com/rpruim/Lock5withR/master/Book/OneTrueLove.csv")
```

Data2.1b

Reading data with read.csv()

One Categorical Variable

From the dataset we named as `OneTrueLove`, we can use the `prop()` function to quickly find **proportions**.

```
prop(~Response, data = OneTrueLove)
```

proportion

target level: Agree; other levels: Disagree, Don't know

```
Agree  
0.28
```

Table 2.1

We can also tabulate the categorical variable to display the *frequency* by using the `tally()` function. The default in tallying is to not include the row totals, or column totals when there are two variables. These are called marginal totals and if you want them, you can change the default.

```
tally(~Response, margin = TRUE, data = OneTrueLove)
```

Table2.1

Agree	Disagree	Don't know	Total
735	1812	78	2625

Example 2.3

To find the proportion of responders who *disagree* or *don't know*, we can use the `level=` argument in the function to find proportions.

```
prop(~Response, level = "Disagree", data = OneTrueLove)
```

Example2.3

target level: Disagree; other levels: Agree, Don't know

Disagree
0.6902857

```
prop(~Response, level = "Don't know", data = OneTrueLove)
```

target level: Don't know; other levels: Agree, Disagree

Don't know
0.02971429

Further, we can also display the *relative frequencies*, or **proportions** in a table.

```
tally(~Response, format = "proportion", margin = TRUE, data = OneTrueLove)
```

Example2.3b

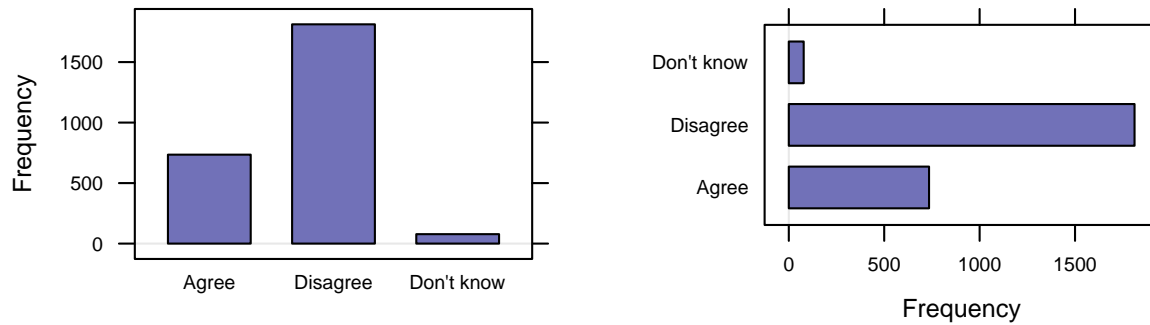
Agree	Disagree	Don't know	Total
0.28000000	0.69028571	0.02971429	1.00000000

Figure 2.1

R provides many different chart and plot functions, including *bar charts* and *pie charts*, to visualize counts or proportions. Bar charts, also known as bar graphs, are a way of displaying the distribution of a categorical variable.


```
bargraph(~Response, data = OneTrueLove)
bargraph(~Response, data = OneTrueLove, horizontal = TRUE)
```

Figure2.1



Two Categorical Variables: Two-Way Tables

Often, it is useful to compute cross tables for two (or more) variables. We can again use `tally()` for several ways to investigate a two-way table.

Table 2.3

```
tally(~Response + Gender, data = OneTrueLove)
```

Table2.3

Response	Gender	
	Female	Male
Agree	363	372
Disagree	1005	807
Don't know	44	34

Table 2.4

```
tally(~Response + Gender, margins = TRUE, data = OneTrueLove)
```

Table2.4

Response	Gender		Total
	Female	Male	
Agree	363	372	735
Disagree	1005	807	1812
Don't know	44	34	78
Total	1412	1213	2625

Example 2.5

Similar to one categorical variable, we can use the `prop()` function to find the proportion of two variables. The first line results in the proportion of females who agree and the proportion of males who agree. The second line shows the proportion who agree that are female and the proportion who disagree that are female. The third results in the proportion of all the survey responders that are female.

```
prop(Response ~ Gender, data = OneTrueLove)
```

target level: Agree; other levels: Disagree, Don't know

Agree.Female	Agree.Male
0.2570822	0.3066777

```
prop(Gender ~ Response, data = OneTrueLove)
```

target level: Female; other levels: Male

Female.Agree	Female.Disagree	Female.Don't know
0.4938776	0.5546358	0.5641026

```
prop(~Gender, data = OneTrueLove)
```

target level: Female; other levels: Male

Female
0.5379048

Example2.5

See though that because we have multiple levels of each variable, this process can become quite tedious if we want to find the proportions for all of the levels. Using the `tally` function a little differently will result in these proportions.

```
tally(Response ~ Gender, data = OneTrueLove)
```

	Gender	
Response	Female	Male
Agree	363	372
Disagree	1005	807
Don't know	44	34

```
tally(~Response | Gender, data = OneTrueLove)
```

	Gender	
Response	Female	Male
Agree	363	372
Disagree	1005	807
Don't know	44	34

```
tally(Gender ~ Response, data = OneTrueLove)
```

Example2.5b

```

      Response
Gender Agree Disagree Don't know
Female  363    1005         44
Male    372     807         34

```

```
tally(~Gender | Response, data = OneTrueLove)
```

```

      Response
Gender Agree Disagree Don't know
Female  363    1005         44
Male    372     807         34

```

Notice that (by default) some of these use counts and some use proportions. Again, we can change the format.

```
tally(~Gender, format = "percent", data = OneTrueLove)
```

Example2.5c

```

      Female      Male
53.79048 46.20952

```

Example 2.6

```
tally(~Gender + Award, margin = TRUE, data = StudentSurvey)
```

Example2.6

```

      Award
Gender Academy Nobel Olympic Total
F          20    76      73    169
M          11    73     109    193
Total       31   149     182    362

```

Also, we can arrange the table differently by converting it to a data frame.

```
as.data.frame(tally(~Gender + Award, data = StudentSurvey))
```

Example2.6b

```

  Gender Award Freq
1      F Academy   20
2      M Academy   11
3      F  Nobel   76
4      M  Nobel   73
5      F Olympic   73
6      M Olympic  109

```

```
prop(~Award, level = "Olympic", data = StudentSurvey)
```

Example2.6c

```
target level: Olympic; other levels: Academy, Nobel

Olympic
0.5027624
```

Example 2.7

To calculate the difference of certain statistics, we can use the `diff()` function. Here we use it to find the difference in proportions, but it can be used for means, medians, and etc.

```
diff(prop(Award ~ Gender, level = "Olympic", data = StudentSurvey))

target level: Olympic; other levels: Academy, Nobel

Olympic.M
0.1328142
```

Example2.7

We will continue more with proportions in Chapter 3.

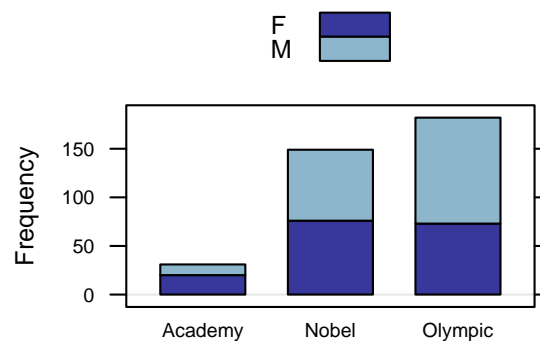
Figure 2.2

A way to look at multiple groups simultaneously is by using *comparative plots* such as a *segmented bar chart* or *side-by-side bar chart*. We use the `groups` argument for this. What `groups` depends a bit on the type of graph. Using `groups` with `histogram()` doesn't work so well because it is difficult to overlay histograms.¹ Density plots work better for this.

Notice the addition of `groups=` (to group), `stack=` (to segment the graph), and `auto.key=TRUE` (to build a simple legend so we can tell which groups are which).

```
bargraph(~Award, groups = Gender, stack = TRUE, auto.key = TRUE, data = StudentSurvey)
```

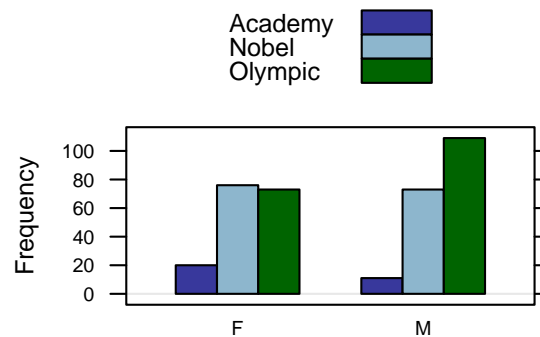
Figure2.2



¹The `mosaic` function `histogram()` does do something meaningful with `groups` in some situations.

```
bargraph(~Gender, groups = Award, auto.key = TRUE, data = StudentSurvey)
```

Figure2.2b



2.2 One Quantitative Variable: Shape and Center

The distribution of a variable answers two questions:

- *What values* can the variable have?
- *With what frequency* does each value occur?

Again, the frequency may be described in terms of counts, proportions (often called relative frequency), or densities (more on densities later).

A distribution may be described using a table (listing values and frequencies) or a graph (e.g., a histogram) or with words that describe general features of the distribution (e.g., symmetric, skewed).

The Shape of a Distribution

Table 2.14

MammalLongevity

Table2.14

	Animal	Gestation	Longevity
1	baboon	187	20
2	bear,black	219	18
3	bear,grizzly	225	25
4	bear,polar	240	20
5	beaver	122	5
6	buffalo	278	15
7	camel	406	12
8	cat	63	12
9	chimpanzee	231	20
10	chipmunk	31	6

11	cow	284	15
12	deer	201	8
13	dog	61	12
14	donkey	365	12
15	elephant	645	40
16	elk	250	15
17	fox	52	7
18	giraffe	425	10
19	goat	151	8
20	gorilla	257	20
21	guinea pig	68	4
22	hippopotamus	238	25
23	horse	330	20
24	kangaroo	42	7
25	leopard	98	12
26	lion	100	15
27	monkey	164	15
28	moose	240	12
29	mouse	21	3
30	opposum	15	1
31	pig	112	10
32	puma	90	12
33	rabbit	31	5
34	rhinoceros	450	15
35	sea lion	350	12
36	sheep	154	12
37	squirrel	44	10
38	tiger	105	16
39	wolf	63	5
40	zebra	365	15

Statisticians have devised a number of graphs to help us see distributions visually. The general syntax for making a graph of one variable in a data frame is

```
plotname(~variable, data = dataName)
```

In other words, there are three pieces of information we must provide to R in order to get the plot we want:

- The kind of plot (`histogram()`, `bargraph()`, `densityplot()`, `bwplot()`, etc.)
- The name of the variable
- The name of the data frame this variable is a part of.

This should look familiar from the previous section.

Figure 2.6

Let's make a *dot plot* of the variable Longevity in the `MammalLongevity` data set for a quick and simple look at the distribution. We use the syntax provided above with two additional arguments to make the figure look the way we want it to. The next few sections will explain a few of the different arguments available for plots in R.

```
dotPlot(~Longevity, width = 1, cex = 0.35, data = MammalLongevity)
```

Figure2.6

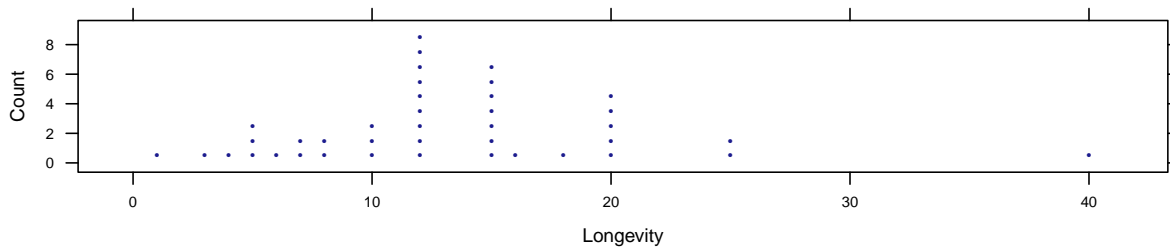


Table 2.15

Although `tally()` works with quantitative variables as well as categorical variables, this is only useful when there are not too many different values for the variable.

```
tally(~Longevity, margin = TRUE, data = MammalLongevity)
```

Table2.15

	1	3	4	5	6	7	8	10	12	15	16	18	20	25	40
	1	1	1	3	1	2	2	3	9	7	1	1	5	2	1
Total															
	40														

Sometimes, it is more convenient to group them into bins. We just have to tell R what the bins are. For example, suppose we wanted to group together by 5.

```
binmed.long <- cut(MammalLongevity$Longevity, breaks = c(0, 5, 10, 15, 20, 25, 30, 35, 40))
tally(~binmed.long) # no data frame given because it is not in a data frame
```

Table2.15b

(0,5]	(5,10]	(10,15]	(15,20]	(20,25]	(25,30]	(30,35]	(35,40]
6	8	16	7	2	0	0	1

Suppose we wanted to group the 1s, 10s, 20s, etc. together. We want to make sure then that 10 is with the 10s, so we should add another argument.

```
binmed.long2 <- cut(MammalLongevity$Longevity, breaks = c(0, 10, 20, 30, 40, 50), right = FALSE)
tally(~binmed.long2) # no data frame given because it is not in a data frame
```

Table2.15c

[0,10)	[10,20)	[20,30)	[30,40)	[40,50)
11	21	7	0	1

We won't use this very often however, since seeing this information in a histogram is typically more useful.

Figure 2.7

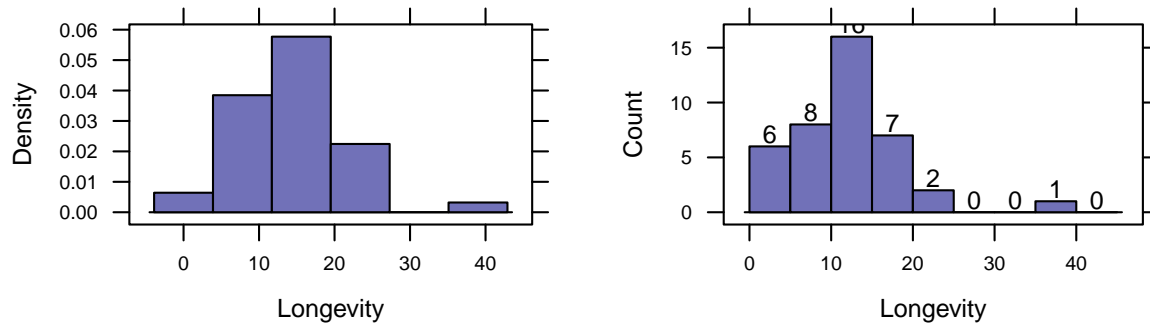
Histograms are a way of displaying the distribution of a quantitative variable.

Figure2.7

```

histogram(~Longevity, data = MammalLongevity)
histogram(~Longevity, width = 5, type = "count", center = 2.5, label = TRUE, data = MammalLongevity)

```



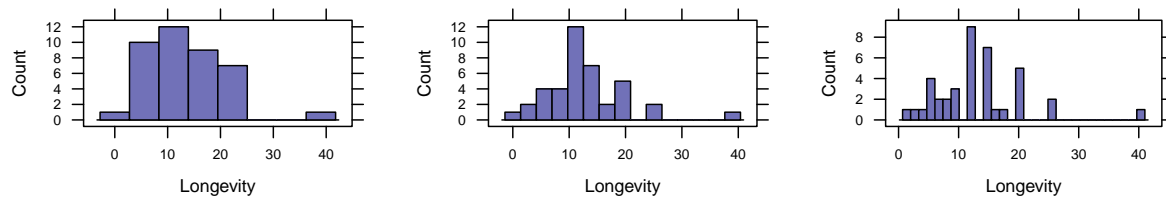
We can control the (approximate) number of bins using the `nint` argument, which may be abbreviated as `n`. The number of bins (and to a lesser extent the positions of the bins) can make a histogram look quite different.

Figure2.7b

```

histogram(~Longevity, type = "count", data = MammalLongevity, n = 8)
histogram(~Longevity, type = "count", data = MammalLongevity, n = 15)
histogram(~Longevity, type = "count", data = MammalLongevity, n = 30)

```



We can also describe the bins in terms of center and width instead of in terms of the number of bins. This is especially nice for count or other integer data.

Figure2.7c

```

histogram(~Longevity, type = "count", data = MammalLongevity, width = 10)
histogram(~Longevity, type = "count", data = MammalLongevity, width = 5)
histogram(~Longevity, type = "count", data = MammalLongevity, width = 2)

```

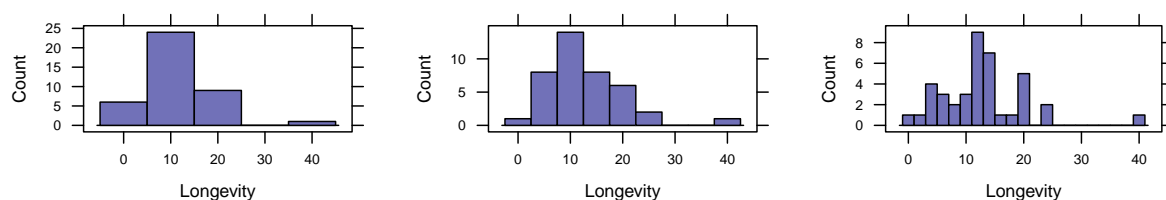
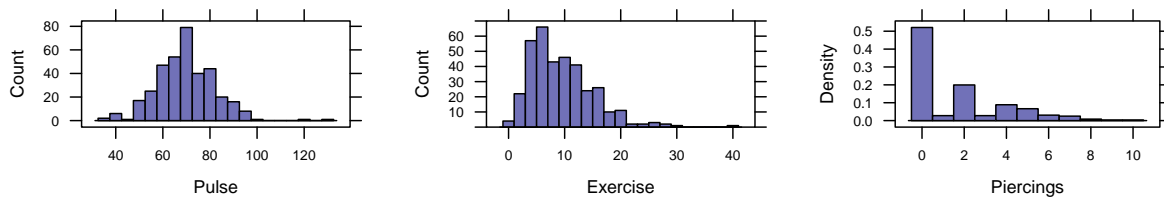


Figure 2.8

The various options available for the `histogram()` function enable us to replicate Figure 2.8, some including centering, adding counts, labels, and limit to the y-axis (similar for x-axis).

```
histogram(~ Pulse, type = "count", width = 5, data = StudentSurvey)
histogram(~ Exercise, type = "count", width = 2, center = 2,
           right = FALSE, ylim = c(0,70), data = StudentSurvey)
histogram(~ Piercings, width = 1, data = StudentSurvey)
```

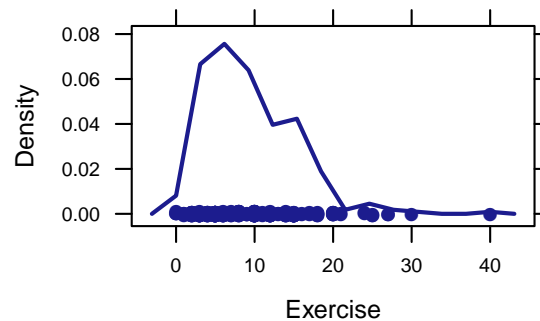
Figure2.8



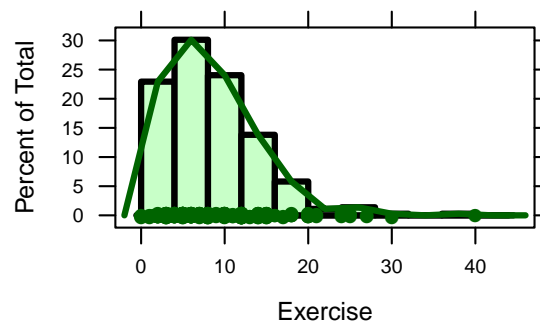
Sometimes a **frequency polygon** provides a more useful view. The only thing that changes is `histogram()` becomes `freqpolygon()`.

```
freqpolygon(~Exercise, width = 5, data = StudentSurvey)
```

freqpolygon



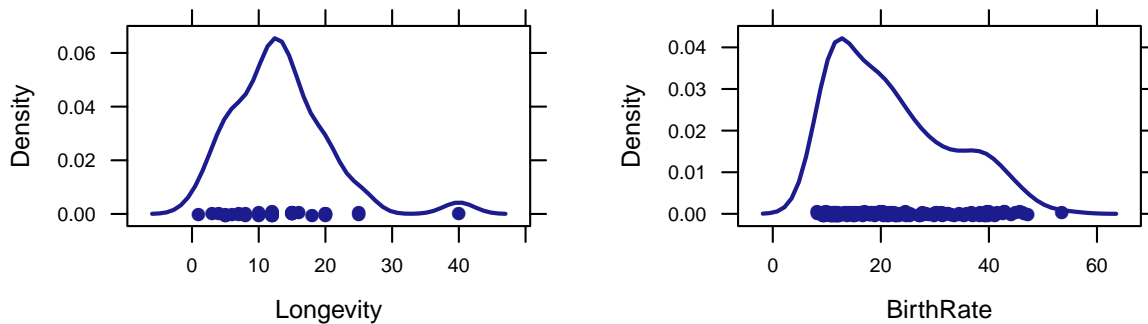
What is a frequency polygon? The picture below shows how it is related to a histogram. The frequency polygon is just a dot-to-dot drawing through the centers of the tops of the bars of the histogram.



R also provides a “smooth” version called a density plot; just change the function name from `histogram()` to `densityplot()`.

```
densityplot(~Longevity, data = MammalLongevity)
densityplot(~BirthRate, data = AllCountries)
```

densityplot



If we make a histogram (or any of these other plots) of our data, we can describe the overall shape of the distribution. Keep in mind that the shape of a particular histogram may depend on the choice of bins. Choosing too many or too few bins can hide the true shape of the distribution. (When in doubt, make more than one histogram.)

Here are some words we use to describe shapes of distributions.

symmetric The left and right sides are mirror images of each other.

skewed The distribution stretches out farther in one direction than in the other. (We say the distribution is skewed toward the long tail.)

uniform The heights of all the bars are (roughly) the same. (So the data are equally likely to be anywhere within some range.)

unimodal There is one major “bump” where there is a lot of data.

bimodal There are two “bumps”.

outlier An observation that does not fit the overall pattern of the rest of the data.

The Center of a Distribution

Recall that a statistic is a number computed from data. The **mean** and the **median** are key statistics which describe the center of a distribution. We can see through Example 2.11 that numerical summaries are computed using the same template as graphical summaries.

Note that the example asks about subsets of `ICUAdmissions`—specifically about 20-year-old and 55-year-old patients. In this case, we can manipulate the data (to name a new data set) with the `subset` command. Here are some examples.

1. Select only the males from the `ICUAdmissions` data set.

```
head(ICUAdmissions, 2)
```

subset

```
  ID Status Age Sex Race Service Cancer Renal Infection CPR Systolic HeartRate Previous
1  8      0  27  1  1      0      0      0      1  0      142      88      0
2 12      0  59  0  1      0      0      0      0  0      112      80      1
  Type Fracture P02 PH PC02 Bicarbonate Creatinine Consciousness status sex race
1  1      1      0  0  0      0      0      0      1 Lived Female White
2  1      1      0  0  0      0      0      0      1 Lived  Male White
  service cancer renal infection cpr previous      type p02low p02 pHlow pH pC02hi pC02
1 Medical      No      No      Yes No      No Emergency No Hi No Hi No Low
2 Medical      No      No      No No      Yes Emergency No Hi No Hi No Low
  bicarbonateLow bicarbonate creatinineHi creatinine consciousness
1              No      Hi      No      Low Conscious
2              No      Hi      No      Low Conscious
```

```
tally(~sex, data = ICUAdmissions)
```

```
Female  Male
    76   124
```

```
ICUMales <- subset(ICUAdmissions, sex == "Male") # notice the double =
tally(~sex, data = ICUMales)
```

```
Female  Male
     0   124
```

2. Select only the subjects over 50:

```
ICU0ld <- subset(ICUAdmissions, Age > 50)
```

subset2

The `subset()` function can use any condition that evaluates to TRUE or FALSE for each row (case) in the data set.

Example 2.11

```
ICU20 <- subset(ICUAdmissions, Age == "20")
mean(~HeartRate, data = ICU20)
```

Example2.11

```
[1] 82.2
```

```
median(~HeartRate, data = ICU20)
```

```
[1] 80
```

```
ICU55 = subset(ICUAdmissions, Age == "55")
mean(~HeartRate, data = ICU55)
```

```
[1] 108.5
```

```
median(~HeartRate, data = ICU55)
```

```
[1] 106
```

Resistance

Figure 2.10

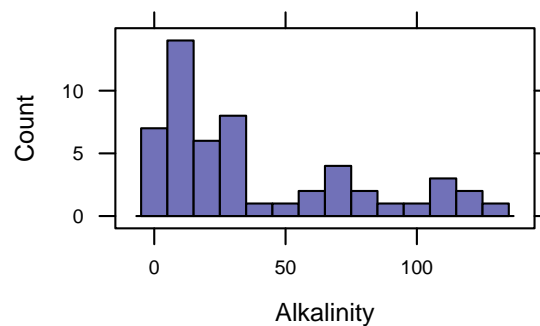
```
head(FloridaLakes)
```

Figure2.10

ID	Lake	Alkalinity	pH	Calcium	Chlorophyll	AvgMercury	NumSamples	MinMercury
1	1 Alligator	5.9	6.1	3.0	0.7	1.23	5	0.85
2	2 Annie	3.5	5.1	1.9	3.2	1.33	7	0.92
3	3 Apopka	116.0	9.1	44.1	128.3	0.04	6	0.04
4	4 Blue Cypress	39.4	6.9	16.4	3.5	0.44	12	0.13
5	5 Brick	2.5	4.6	2.9	1.8	1.20	12	0.69
6	6 Bryant	19.6	7.3	4.5	44.1	0.27	14	0.04

	MaxMercury	ThreeYrStdMercury	AgeData
1	1.43	1.53	1
2	1.90	1.33	0
3	0.06	0.04	0
4	0.84	0.44	0
5	1.50	1.33	1
6	0.48	0.25	1

```
histogram(~Alkalinity, width = 10, type = "count", data = FloridaLakes)
```



Example 2.14

```
mean(~Alkalinity, data = FloridaLakes)
```

Example2.14

```
[1] 37.53019
```

```
median(~Alkalinity, data = FloridaLakes)
```

```
[1] 19.6
```

2.3 One Quantitative Variable: Measures of Spread

In the previous section, we investigated center summary statistics. In this section, we will cover some other important statistics.

Example 2.15

```
summary(April14Temps)
```

Example2.15

Year	DesMoines	SanFrancisco
Min. :1995	Min. :37.20	Min. :48.70
1st Qu.:1999	1st Qu.:44.40	1st Qu.:51.30
Median :2002	Median :54.50	Median :54.00
Mean :2002	Mean :54.49	Mean :54.01
3rd Qu.:2006	3rd Qu.:61.27	3rd Qu.:55.90
Max. :2010	Max. :74.90	Max. :61.00

```
favstats(~DesMoines, data = April14Temps) # some favorite statistics
```

min	Q1	median	Q3	max	mean	sd	n	missing
37.2	44.4	54.5	61.275	74.9	54.4875	11.7303	16	0

```
favstats(~SanFrancisco, data = April14Temps)
```

min	Q1	median	Q3	max	mean	sd	n	missing
48.7	51.3	54	55.9	61	54.0125	3.376956	16	0

Standard Deviation

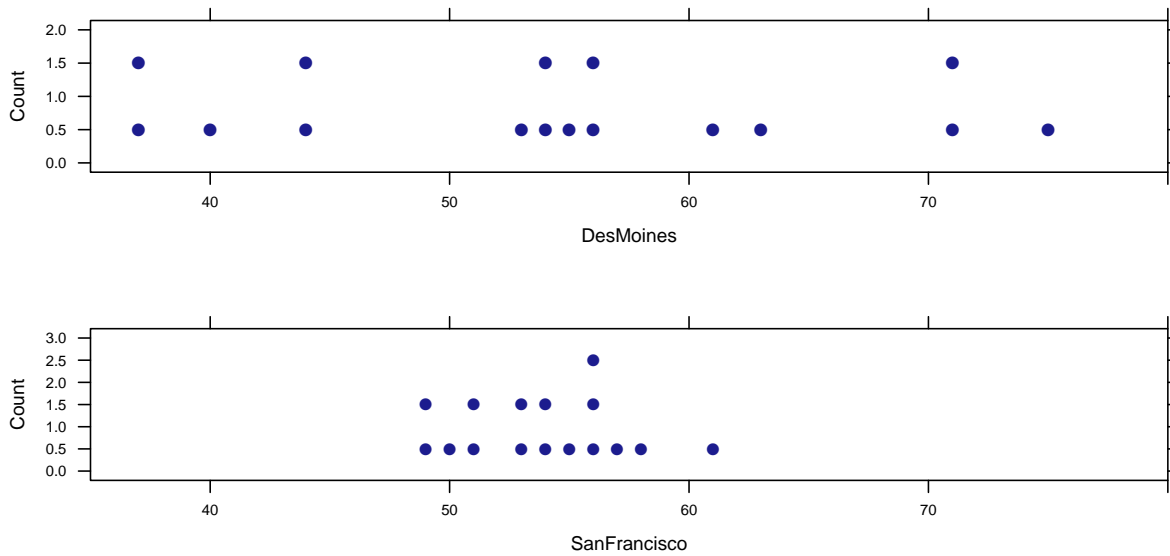
The density plots of the temperatures of Des Moines and San Francisco reveal that Des Moines has a greater *variability* or *spread*.

Figure 2.18

The `cex` argument controls “character expansion” and can be used to make the plotting “characters” larger or smaller by specifying the scaling ratio. `xlim` sets the limits for the x-axis.

```
dotPlot(~DesMoines, width = 1, cex = 0.25, xlim = c(35, 80), data = April14Temps)
dotPlot(~SanFrancisco, width = 1, cex = 0.35, xlim = c(35, 80), data = April14Temps)
```

Figure2.18



Example 2.16

Although both `summary()` and `favstats()` calculate the **standard deviation** of a variable, we can also use `sd()` to find just the standard deviation.

```
sd(~DesMoines, data = April14Temps)

[1] 11.7303

sd(~SanFrancisco, data = April14Temps)

[1] 3.376956

var(~DesMoines, data = April14Temps) # variance = sd^2

[1] 137.5998
```

standard-deviation

Example 2.17

To see that the distribution is indeed symmetric and approximately bell-shaped, you can use the argument `fit` to overlay a “normal” curve.

```
histogram(~Pulse, fit = "normal", data = StudentSurvey)
mean <- mean(~Pulse, data = StudentSurvey)
mean

[1] 69.57459
```

Example2.17

```
sd <- sd(~Pulse, data = StudentSurvey)
sd

[1] 12.20514

mean - 2 * sd

[1] 45.16431

mean + 2 * sd

[1] 93.98486
```

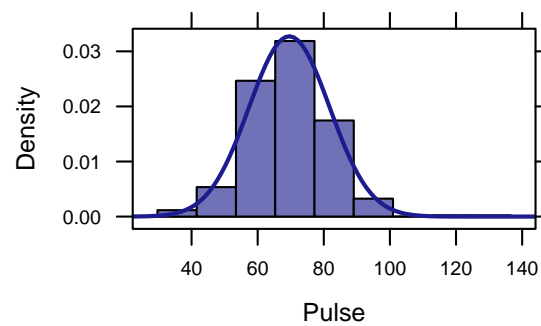
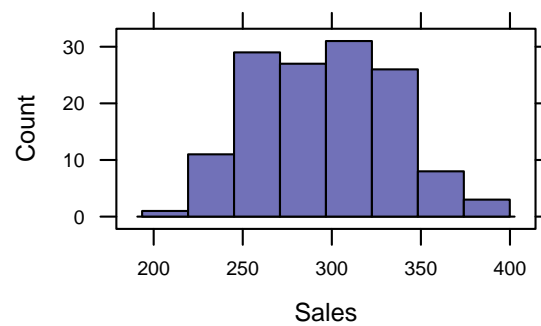


Figure 2.20

```
histogram(~Sales, type = "count", data = RetailSales)
```

Figure2.20



Example 2.18

```
mean <- mean(~Sales, data = RetailSales)
mean
```

Example2.18

```
[1] 296.4382
```

```
sd <- sd(~Sales, data = RetailSales)
sd
```

```
[1] 37.97074
```

```
mean - 2 * sd
```

```
[1] 220.4968
```

```
mean + 2 * sd
```

```
[1] 372.3797
```

Example 2.19

Z-scores can be computed as follows:

```
(204 - mean(~Systolic, data = ICUAdmissions))/sd(~Systolic, data = ICUAdmissions)
```

Example2.19

```
[1] 2.176493
```

```
(52 - mean(~HeartRate, data = ICUAdmissions))/sd(~HeartRate, data = ICUAdmissions)
```

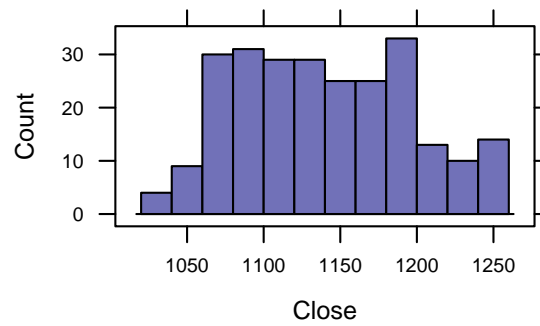
```
[1] -1.749
```

Percentiles

Figure 2.21

```
histogram(~Close, type = "count", width = 20, center = 10, data = SandP500)
```

Figure2.21



Example 2.20

The text uses a histogram to estimate the **percentile** of the daily closing price for the S&P 500 but we can also find the exact percentiles using the `quantile()` function.

```
quantile(SandP500$Close, probs = seq(0, 1, 0.25))
```

```
      0%      25%      50%      75%     100%
1022.580 1094.802 1136.985 1183.372 1259.780
```

```
quantile(SandP500$Close, probs = seq(0, 1, 0.9))
```

```
      0%      90%
1022.580 1216.906
```

Example2.20

Five Number Summary

We have already covered many different functions which results in the **five number summary** but `fivenum()` is most direct way to obtain in the five number summary.

Example 2.21

```
fivenum(~Exercise, data = StudentSurvey)
```

Example2.21

Example 2.22

```
fivenum(~Longevity, data = MammalLongevity)
```

```
[1]  1.0  8.0 12.0 15.5 40.0
```

Example2.22

```
min(~Longevity, data = MammalLongevity)

[1] 1

max(~Longevity, data = MammalLongevity)

[1] 40

range(~Longevity, data = MammalLongevity) # subtract to get the numerical range value

[1] 1 40

iqr(~Longevity, data = MammalLongevity) # interquartile range

[1] 7.25
```

Note the difference in the quartile and IQR from the textbook. This results because there are several different methods to determine the quartile.

Example 2.23

```
fivenum(~DesMoines, data = April14Temps)

[1] 37.20 44.40 54.50 61.95 74.90

fivenum(~SanFrancisco, data = April14Temps)

[1] 48.7 51.2 54.0 56.0 61.0

range(~DesMoines, data = April14Temps)

[1] 37.2 74.9

diff(range(~DesMoines, data = April14Temps))

[1] 37.7

range(~SanFrancisco, data = April14Temps)

[1] 48.7 61.0

diff(range(~SanFrancisco, data = April14Temps))

[1] 12.3
```

Example2.23

```
iqr(~DesMoines, data = April14Temps)

[1] 16.875

iqr(~SanFrancisco, data = April14Temps)

[1] 4.6
```

2.4 Outliers, Boxplots, and Quantitative/Categorical Relationships

Detection of Outliers

Generally, outliers are considered to be values

- less than $Q_1 - 1.5 \cdot (IQR)$, and
- greater than $Q_3 + 1.5 \cdot (IQR)$.

Example 2.25

```
fivenum(~Longevity, data = MammalLongevity)

[1] 1.0 8.0 12.0 15.5 40.0

iqr(~Longevity, data = MammalLongevity)

[1] 7.25

8 - 1.5 * 7.25

[1] -2.875

15.5 + 1.5 * 7.25

[1] 26.375

subset(MammalLongevity, Longevity > 26.375)

      Animal Gestation Longevity
15 elephant          645         40
```

Example2.25

There is no function in R that directly results in outliers because practically, there is no one specific formula for such a determination. However, a boxplot will indirectly reveal outliers.

Boxplots

A way to visualize the five number summary and outliers for a variable is to create a boxplot.

Example 2.26

```
favstats(~Longevity, data = MammalLongevity)
```

	min	Q1	median	Q3	max	mean	sd	n	missing
1	8	12	15.25	40	13.15	7.244981	40	0	

```
bwplot(~Longevity, data = MammalLongevity)
```

Example2.26

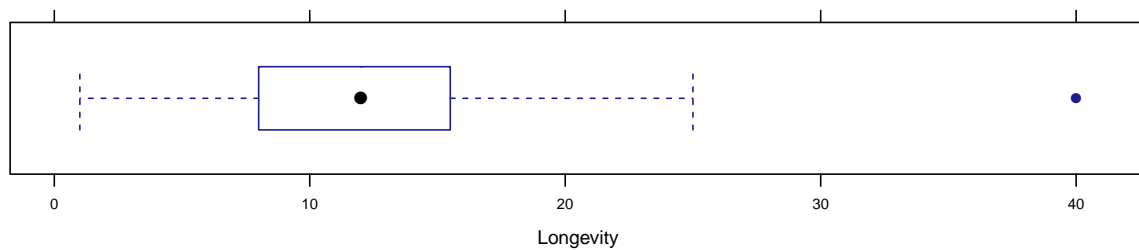
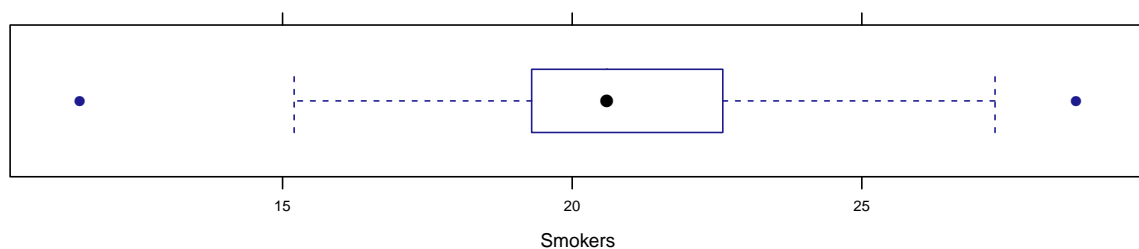


Figure 2.32

```
bwplot(~Smokers, data = USStates)
```

Figure2.32



Example 2.27

We can similarly investigate the *Smokers* variable in [USStates](#).

```
fivenum(~Smokers, data = USStates)
```

```
[1] 11.5 19.3 20.6 22.6 28.7
```

Example2.27

The boxplot reveals two outliers. To identify them, we can again use `subset()` for smokers greater or less than the *whiskers* of the boxplot.

```
subset(USStates, Smokers < 15)
```

Example2.27b

```

State HouseholdIncome  IQ McCainVote Region ObamaMcCain Population EighthGradeMath
44 Utah              55619 101.1      0.629      W           M    2.420708          279.16
HighSchool  GSP FiveVegetables Smokers PhysicalActivity Obese College NonWhite
44          91 36758          22.1    11.5          83.1  21.2      31      12.1
HeavyDrinkers Pres2008
44          2.9    McCain

```

```
subset(USStates, Smokers > 28)
```

```

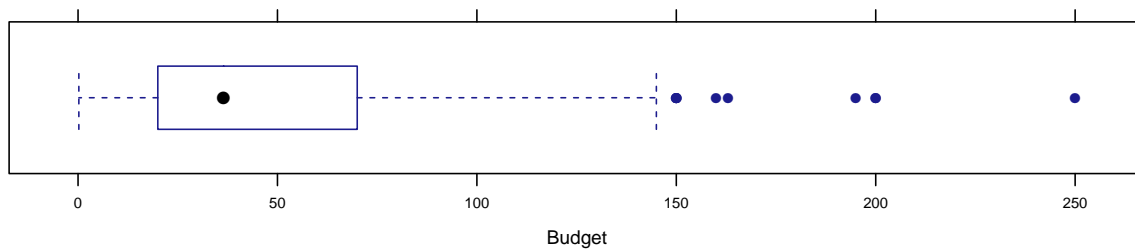
State HouseholdIncome  IQ McCainVote Region ObamaMcCain Population EighthGradeMath
17 Kentucky          38694 99.4      0.575      MW           M    4.141835          273.98
HighSchool  GSP FiveVegetables Smokers PhysicalActivity Obese College NonWhite
17          81.8 33666          16.8    28.7          70.1  28.6      22.6      9.4
HeavyDrinkers Pres2008
17          2.7    McCain

```

Figure 2.33

```
bwplot(~Budget, data = HollywoodMovies2011)
```

Figure2.33



Example 2.28

```
subset(HollywoodMovies2011, Budget > 225)
```

Example2.28

```

Movie LeadStudio RottenTomatoes AudienceScore
30 Pirates of the Caribbean:\nOn Stranger Tides Disney 34 61
Story Genre TheatersOpenWeek BOAverageOpenWeek DomesticGross ForeignGross WorldGross
30 Quest Action 4155 21697 241.07 802.8 1043.871
Budget Profitability OpeningWeekend
30 250 4.175484 90.15

```

```
head(HollywoodMovies2011)
```

				Movie	LeadStudio	RottenTomatoes		
1				Insidious	Sony	67		
2				Paranormal Activity 3	Independent	68		
3				Bad Teacher	Independent	44		
4				Harry Potter and the Deathly Hallows Part 2	Warner Bros	96		
5				Bridesmaids	Relativity Media	90		
6				Midnight in Paris	Sony	93		
	AudienceScore	Story	Genre	Theaters	OpenWeek	BOAverage	OpenWeek	DomesticGross
1	65	Monster Force	Horror		2408		5511	54.01
2	58	Monster Force	Horror		3321		15829	103.66
3	38	Comedy	Comedy		3049		10365	100.29
4	92	Rivalry	Fantasy		4375		38672	381.01
5	77	Rivalry	Comedy		2918		8995	169.11
6	84	Love	Romance		944		6177	56.18
	ForeignGross	WorldGross	Budget	Profitability	OpeningWeekend			
1	43.00	97.009	1.5	64.672667	13.27			
2	98.24	201.897	5.0	40.379400	52.57			
3	115.90	216.196	20.0	10.809800	31.60			
4	947.10	1328.111	125.0	10.624888	169.19			
5	119.28	288.382	32.5	8.873292	26.25			
6	83.00	139.177	17.0	8.186882	5.83			

One Quantitative and One Categorical Variable

The formula for a **lattice** plot can be extended to create multiple panels (sometimes called **facets**) based on a “condition”, often given by another variable. This is another way to look at multiple groups simultaneously. The general syntax for this becomes

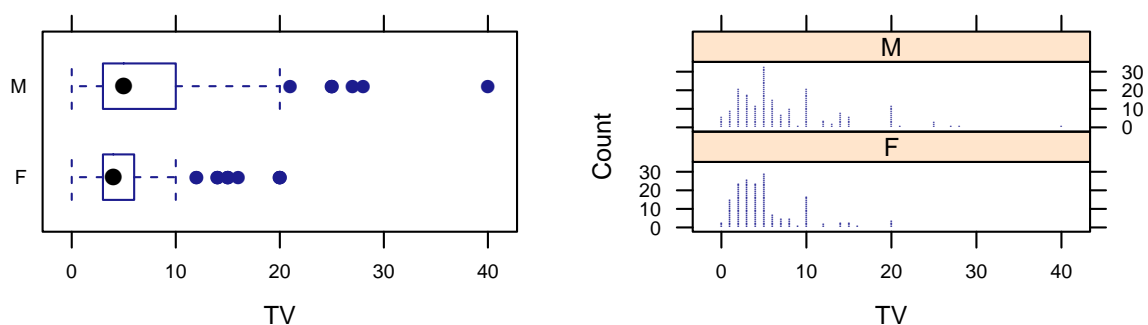
```
plotname(~variable | condition, data = dataName)
```

Figure 2.34

Depending on the type of plot, you will want to use conditioning.

```
bwplot(Gender ~ TV, data = StudentSurvey)
dotPlot(~TV | Gender, layout = c(1, 2), width = 1, cex = 1, data = StudentSurvey)
```

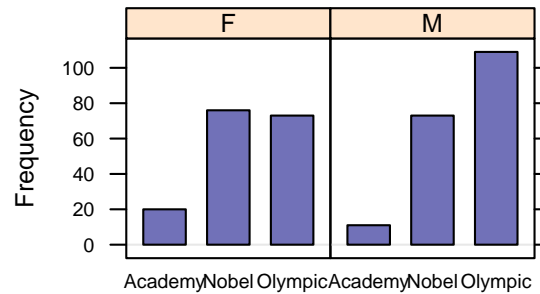
Figure2.34



We can do the same thing for bar graphs.

```
bargraph(~Award | Gender, data = StudentSurvey)
```

Figure 2.34b



This graph should be familiar as we have plotted these variables together previously. Here we used different panels, but before, in 2.1, we had used grouping. Note that we can combine grouping and conditioning in the same plot.

Example 2.31

```
favstats(~TV | Gender, data = StudentSurvey)
diff(mean(~TV | Gender, data = StudentSurvey))
```

Example 2.31

2.5 Two Quantitative Variables: Scatterplot and Correlation

Example 2.32

ElectionMargin

Example 2.32

	Year	Candidate	Approval	Margin	Result
1	1940	Roosevelt	62	10.0	Won
2	1948	Truman	50	4.5	Won
3	1956	Eisenhower	70	15.4	Won
4	1964	Johnson	67	22.6	Won
5	1972	Nixon	57	23.2	Won
6	1976	Ford	48	-2.1	Lost
7	1980	Carter	31	-9.7	Lost
8	1984	Reagan	57	18.2	Won
9	1992	G.H.W.Bush	39	-5.5	Lost
10	1996	Clinton	55	8.5	Won
11	2004	G.W.Bush	49	2.4	Won

Visualizing a Relationship between Two Quantitative Variables: Scatterplots

The most common way to look at two quantitative variables is with a scatterplot. The `lattice` function for this is `xyplot()`, and the basic syntax is

```
xyplot(yvar ~ xvar, data = dataName)
```

Notice that now we have something on both sides of the `~` since we need to tell R about two variables.

Example 2.33

```
xyplot(Margin ~ Approval, data = ElectionMargin)
```

Example2.33

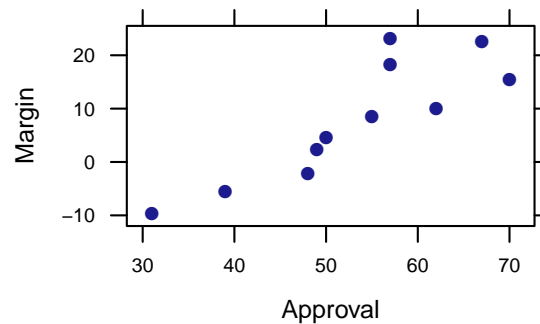
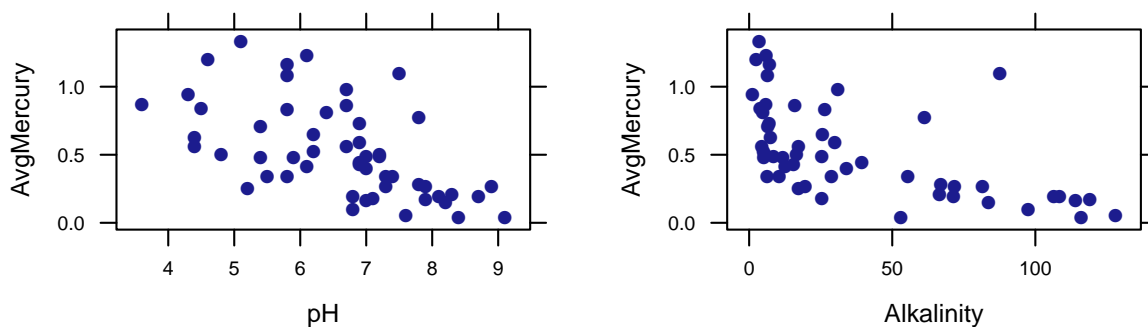
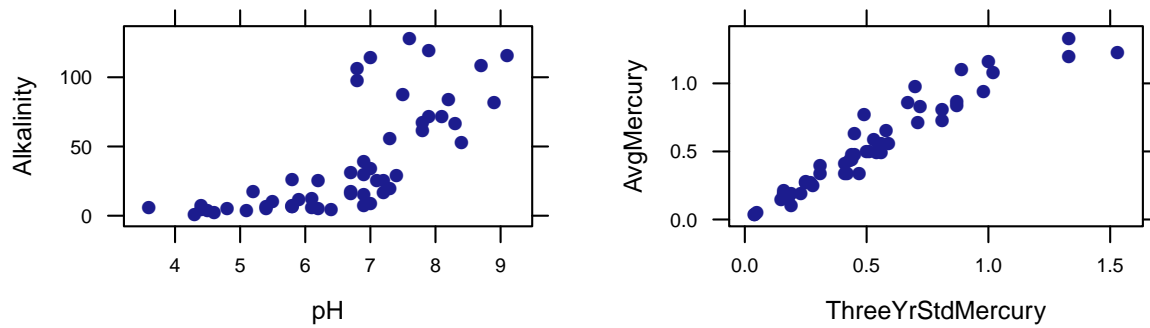


Figure 2.49

```
xyplot(AvgMercury ~ pH, data = FloridaLakes)
xyplot(AvgMercury ~ Alkalinity, data = FloridaLakes)
xyplot(Alkalinity ~ pH, data = FloridaLakes)
xyplot(AvgMercury ~ ThreeYrStdMercury, data = FloridaLakes)
```

Figure2.49





Summarizing a Relationship between Two Quantitative Variables: Correlation

Another key numerical statistic is the **correlation**—the correlation is a measure of the strength and direction of the relationship between two quantitative variables.

```
cor(Margin ~ Approval, data = ElectionMargin)
```

```
[1] 0.8629926
```

```
cor(AvgMercury ~ pH, data = FloridaLakes)
```

```
[1] -0.5754001
```

```
cor(AvgMercury ~ Alkalinity, data = FloridaLakes)
```

```
[1] -0.5938967
```

```
cor(Alkalinity ~ pH, data = FloridaLakes)
```

```
[1] 0.7191657
```

```
cor(AvgMercury ~ ThreeYrStdMercury, data = FloridaLakes)
```

```
[1] 0.9592148
```

Table2.30

Table 2.31

CricketChirps

Table2.31

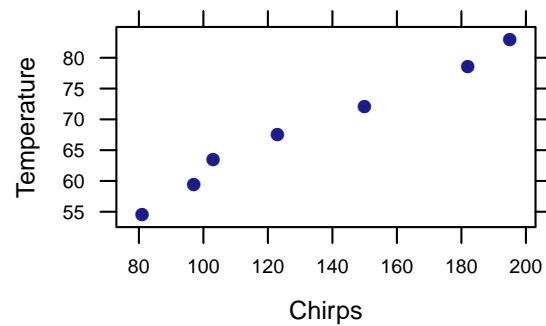
	Temperature	Chirps
1	54.5	81

2	59.5	97
3	63.5	103
4	67.5	123
5	72.0	150
6	78.5	182
7	83.0	195

Figure 2.50

```
xypLOT(Temperature ~ Chirps, data = CricketChirps)
```

Figure2.50



Example 2.35

```
cor(Temperature ~ Chirps, data = CricketChirps)
```

Example2.35

```
[1] 0.9906249
```

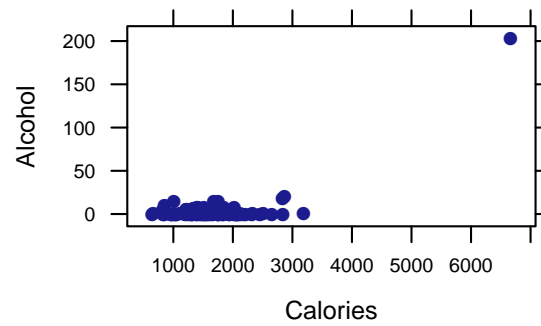
Example 2.38

Further, using the `subset()` function again, we can investigate the correlation between variables with some restrictions.

```
xypLOT(Alcohol ~ Calories, data = subset(NutritionStudy, Age > 59))
cor(Alcohol ~ Calories, data = subset(NutritionStudy, Age > 59))
```

Example2.38

```
[1] 0.7199945
```

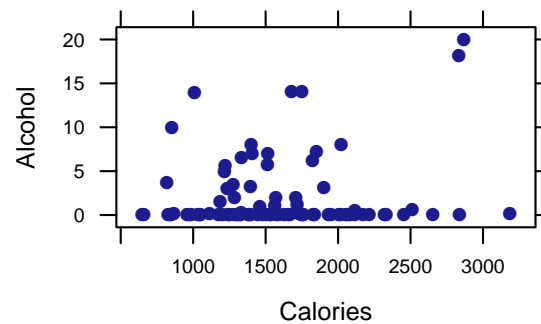


And now we omit the outlier

```
NutritionStudy60 = subset(NutritionStudy, Age > 59)
xyplot(Alcohol ~ Calories, data = subset(NutritionStudy60, Alcohol < 25))
cor(Alcohol ~ Calories, data = subset(NutritionStudy60, Alcohol < 25))
```

Example2.38b

```
[1] 0.1449633
```

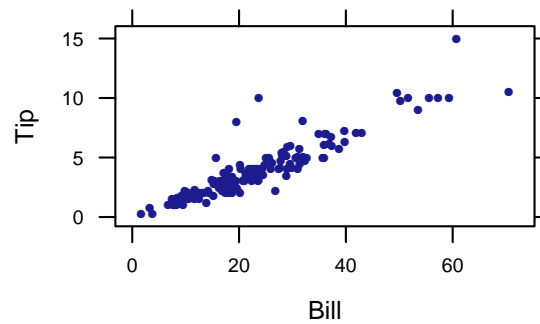


2.6 Two Quantitative Variables: Linear Regression

Figure 2.63

```
xyplot(Tip ~ Bill, cex = 0.5, data = RestaurantTips)
```

Figure2.63



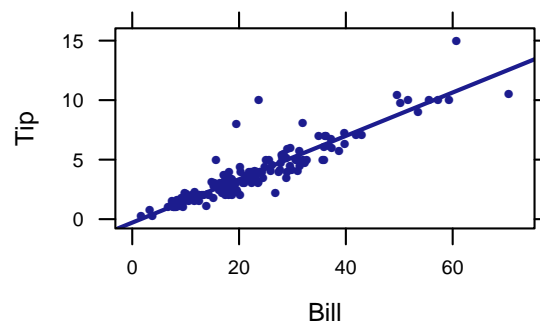
Example 2.39

When the relationship between variables is sufficiently *linear*, you may be able to predict the value of a variable using the other variable. This is possible by fitting a *regression line*. To plot this in R, all we need to do is add an additional argument, `type=c("p", "r")`, to the `xyplot`.

```
xyplot(Tip ~ Bill, cex = 0.5, type = c("p", "r"), data = RestaurantTips)
cor(Tip ~ Bill, data = RestaurantTips)
```

Example2.39

```
[1] 0.9150592
```



The equation for the regression line, or the *prediction equation* is

$$\widehat{\text{Response}} = a + b \cdot \text{Explanatory}$$

So now, we need to find the values for a , the intercept, and b , the slope using the function to fit linear models.

Example 2.41

```
lm(Tip ~ Bill, data = RestaurantTips)
```

Example2.41

```
Call:
lm(formula = Tip ~ Bill, data = RestaurantTips)

Coefficients:
(Intercept)      Bill
    -0.2923      0.1822

coef(lm(Tip ~ Bill, data = RestaurantTips)) # just show me the coefficients

(Intercept)      Bill
    -0.2922675    0.1822147
```

This results in the equation

$$\widehat{\text{Tip}} = -0.2922675 + 0.1822147 \cdot \text{Bill}$$

With this equation, one can predict the tip for different bill amounts.

```
Tip.Fun <- makeFun(lm(Tip ~ Bill, data = RestaurantTips)) # make a function of the linear model
Tip.Fun(Bill = 59.33) # predicted tip when bill is $59.33

      1
10.51853

Tip.Fun(Bill = 9.52)

      1
1.442417

Tip.Fun(Bill = 23.7)

      1
4.026222
```

Example2.41b

An important aspect of the linear regression is the difference between the prediction and actual observation. This is called the **residual**, defined

$$\text{residual} = \text{observed response} - \text{predicted response}$$

Example 2.42

```
Resid.a <- 10 - 10.51 # predicted tip from Example 2.41
Resid.a

[1] -0.51

Resid.b <- 1 - 1.44
Resid.b
```

Example2.42

```
[1] -0.44
```

```
Resid.c <- 10 - 4.02
Resid.c
```

```
[1] 5.98
```

Example 2.43

```
Elect.mod <- lm(Margin ~ Approval, data = ElectionMargin)
resid(lm(Margin ~ Approval, data = ElectionMargin))
```

Example2.43

```
      1      2      3      4      5      6      7      8
-5.3228564 -0.7958766 -6.6075096  3.0992354 12.0550519 -5.7247133  0.8801748  7.0550519
      9     10     11
-1.6044784 -0.9737848 -2.0602949
```

Example 2.45

```
lm(AvgMercury ~ pH, data = Floridalakes)
```

Example2.45

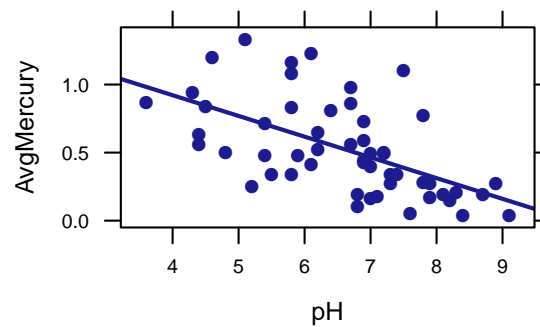
Call:

```
lm(formula = AvgMercury ~ pH, data = Floridalakes)
```

Coefficients:

```
(Intercept)      pH
    1.5309    -0.1523
```

```
xyplot(AvgMercury ~ pH, type = c("p", "r"), data = Floridalakes)
```



```
Mer.Fun <- makeFun(lm(AvgMercury ~ pH, data = Floridalakes))
Mer.Fun(pH = 7.5) # predicted mercury level at 7.5 pH
```

Example2.45b

```

      1
0.3886622

Resid <- 1.1 - 0.388 # residual at 7.5 pH
Resid

[1] 0.712

```

Example 2.46

```

Cal.Fun <- makeFun(lm(Calcium ~ pH, data = FloridaLakes))
Cal.Fun

function (pH, ..., transform = identity)
return(transform(predict(model, newdata = data.frame(pH = pH),
  ...)))
<environment: 0x7f8d2ee2a0a0>
attr(,"coefficients")
(Intercept)      pH
-51.40157      11.16800

```

Example2.46

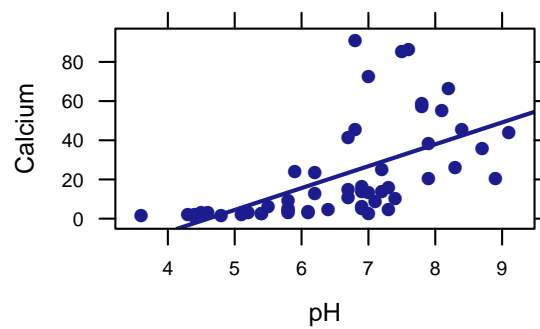
Figure 2.68

```

xyplot(Calcium ~ pH, type = c("p", "r"), data = FloridaLakes)

```

Figure2.68



3

Confidence Intervals

3.1 Sampling Distributions

The key idea in this chapter is the notion of a sampling distribution. Do not confuse it with the population (what we would like to know about) or the sample (what we actually have data about). If we could repeatedly sample from a population, and if we computed a statistic from each sample, the distribution of those statistics would be the sampling distribution. Sampling distributions tell us how things vary from sample to sample and are the key to interpreting data.

Variability of Sample Statistics

Example 3.4

```
head(StatisticsPhD)
```

Example 3.4

	University	Department	FTGradEnrollment
1	Baylor University	Statistics	26
2	Boston University	Biostatistics	39
3	Brown University	Biostatistics	21
4	Carnegie Mellon University	Statistics	39
5	Case Western Reserve University	Statistics	11
6	Colorado State University	Statistics	14

```
mean(~FTGradEnrollment, data = StatisticsPhD) # mean enrollment in original population
```

```
[1] 53.53659
```

Example 3.5

To select a random sample of a certain size in R, we can use the `sample()` function.

```
sample10 <- sample(StatisticsPhD, 10)
sample10
```

	University	Department	FTGradEnrollment	orig.ids
23	Michigan State University	Statistics	81	23
28	Ohio State University	Statistics	101	28
11	Emory University	Biostatistics	58	11
21	Medical College of Wisconsin	Biostatistics	7	21
15	Harvard University	Biostatistics	70	15
29	Oklahoma State University	Statistics	22	29
13	George Mason University	Statistics	10	13
5	Case Western Reserve University	Statistics	11	5
9	Cornell University	Statistics	78	9
81	Western Michigan University	Statistics	31	81

```
x.bar <- mean(~FTGradEnrollment, data = sample10)
x.bar # mean enrollment in sample10

[1] 46.9
```

Example3.5

Note that this sample has been assigned a name to which we can refer back to find the mean of that particular sample.

```
mean(~FTGradEnrollment, data = sample(StatisticsPhD, 10)) # mean enrollment in another sample

[1] 65.3
```

Example3.5b

Figure 3.1

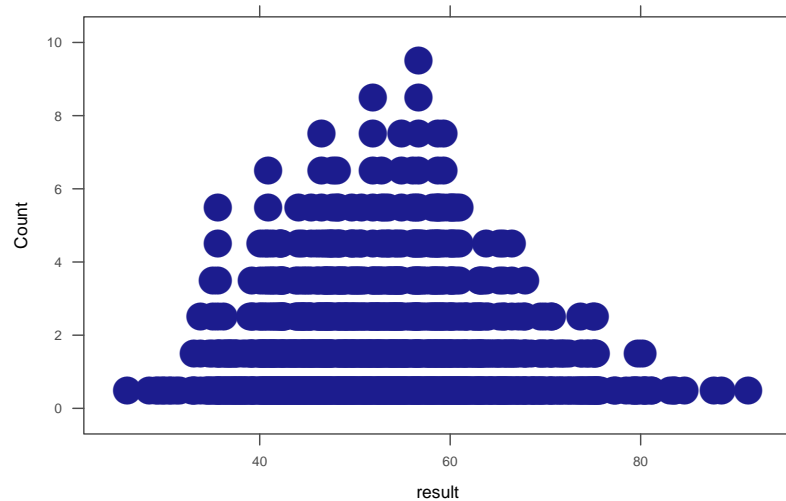
We should check that that our sample distribution has an appropriate shape:

```
# Now we'll do it 1000 times
sampledist <- do(1000) * mean(~FTGradEnrollment, data = sample(StatisticsPhD, 10))
head(samedist, 3)
```

	result
1	44.6
2	58.3
3	61.0

```
dotPlot(~result, width = 0.005, data = samedist)
```

Figure3.1



In many (but not all) situations, the sampling distribution is

- unimodal,
- symmetric, and
- bell-shaped (The technical phrase is “approximately normal”).

Example 3.6

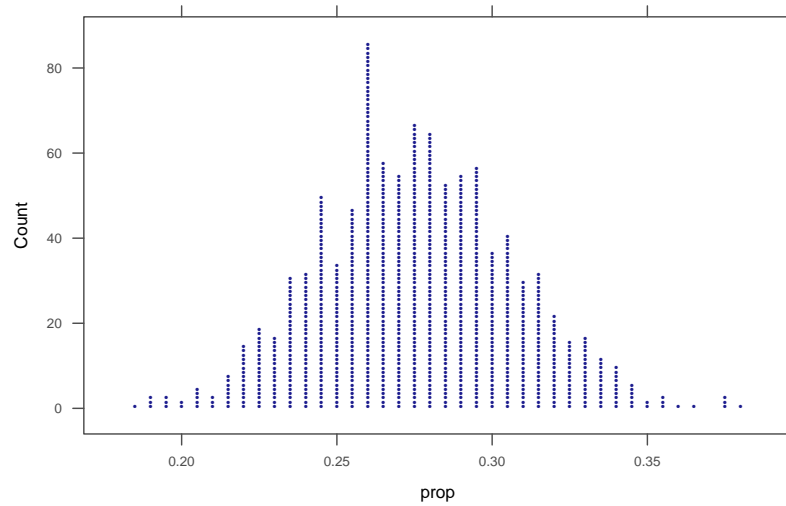
This time we don’t have data, but instead we have a summary of the data. We can however, still simulate the sample distribution by using the `rflip()` function.

Example 3.6

```
sampldist.deg <- do(1000) * rflip(200, 0.275) # 1000 samples, each of size 200 and proportion 0.275
head(sampldist.deg, 3)
```

	n	heads	tails	prop
1	200	63	137	0.315
2	200	58	142	0.290
3	200	60	140	0.300

```
dotPlot(~prop, width = 0.005, data = sampldist.deg)
```



Measuring Sampling Variability: The Standard Error

The standard deviation of a sampling distribution is called the **standard error**, denoted *SE*.

The standard error is our primary way of measuring how much variability there is from sample statistic to sample statistic, and therefore how precise our estimates are.

Example 3.7

Calculating the SE is the same as calculating the standard deviation of a sampling distribution, so we use `sd()`.

```
SE <- sd(~result, data = sampledist)
SE # sample from Example 3.5

[1] 10.86339

SE2 <- sd(~prop, data = sampledist.deg)
SE2 # sample from Example 3.6

[1] 0.03161997
```

Example3.7

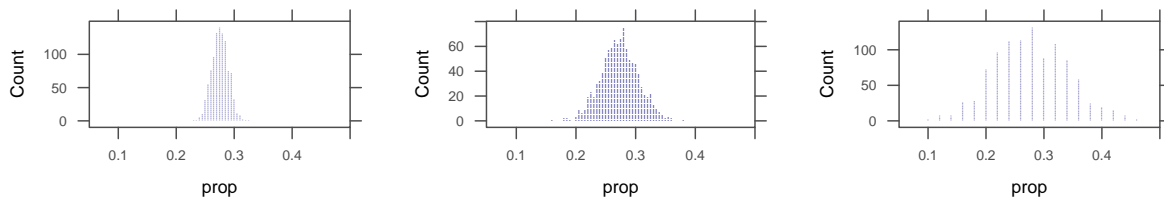
The Importance of Sample Size

Example 3.9

```
sampldist.1000 <- do(1000) * rflip(1000, 0.275) # 1000 samples, each of size 1000 and proportion 0.275
sampldist.200 <- do(1000) * rflip(200, 0.275) # 1000 samples, each of size 200 and proportion 0.275
sampldist.50 <- do(1000) * rflip(50, 0.275) # 1000 samples, each of size 50 and proportion 0.275
```

Figure 3.3

```
dotPlot(~prop, width = 0.005, xlim = c(0.05, 0.5), data = sampldist.1000)
dotPlot(~prop, width = 0.005, xlim = c(0.05, 0.5), data = sampldist.200)
dotPlot(~prop, width = 0.005, xlim = c(0.05, 0.5), data = sampldist.50)
```



3.2 Understanding and Interpreting Confidence Intervals

Interval Estimates and Margin of Error

An **interval estimate** gives a range of plausible values for a population parameter.

This is better than a single number (also called a point estimate) because it gives some indication of the precision of the estimate.

One way to express an interval estimate is with a point estimate and a **margin of error**.

We can convert margin of error into an interval by adding and subtracting the margin of error to/from the statistic.

Example 3.12

```
p.hat <- 0.42 # sample proportion
MoE <- 0.03 # margin of error
p.hat - MoE # lower limit of interval estimate

[1] 0.39

p.hat + MoE # upper limit of interval estimate

[1] 0.45
```

Example 3.13

```

p.hat <- 0.54          # sample proportion
MoE <- 0.02           # margin of error
p.hat - MoE            # lower limit of interval estimate

[1] 0.52

p.hat + MoE           # upper limit of interval estimate

[1] 0.56

```

Example3.13

```

p.hat <- 0.54
MoE <- 0.1
p.hat - MoE

[1] 0.44

p.hat + MoE

[1] 0.64

```

Example3.13b

Confidence Intervals

A confidence interval for a parameter is an interval computed from sample data by a method that will capture the parameter for a specified proportion of all samples

1. The probability of correctly containing the parameter is called the coverage rate or **confidence level**.
2. So 95% of 95% confidence intervals contain the parameter being estimated.
3. The margins of error in the tables above were designed to produce 95% confidence intervals.

Example 3.14

```

x.bar <- 61.5          # given sample mean
SE <- 11              # given estimated standard error
MoE <- 2 * SE; MoE    # margin of error for 95% CI

[1] 22

x.bar - MoE           # lower limit of 95% CI

```

Example3.14

```
[1] 39.5

x.bar + MoE           # upper limit of 95% CI

[1] 83.5
```

Understanding Confidence Intervals

Example 3.15

```
SE <- 0.03
p1 <- 0.26
p2 <- 0.32
p3 <- 0.2
MoE <- 2 * SE
```

Example3.15

```
p1 - MoE
```

```
[1] 0.2
```

```
p1 + MoE
```

```
[1] 0.32
```

```
p2 - MoE
```

```
[1] 0.26
```

```
p2 + MoE
```

```
[1] 0.38
```

```
p3 - MoE
```

```
[1] 0.14
```

```
p3 + MoE
```

```
[1] 0.26
```

Example3.15b

Figure 3.12

```
p <- 0.275
SE <- 0.03
MoE <- 2 * SE
p - MoE

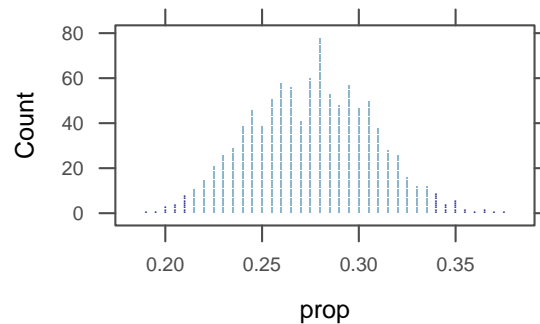
[1] 0.215

p + MoE

[1] 0.335

dotPlot(~prop, width = 0.005, groups = (0.215 <= prop & prop <= 0.335), data = sampledist.deg)
```

Figure3.12



Notice how we defined groups in this dotplot. We are grouping proportions that less than 0.215 and more than 0.335.

Figure 3.13

We can create the data needed for plots like Figure 3.13 using `CIsim()`. The plot itself uses `xYplot()` from the `Hmisc` package.

```
results <- CIsim(200, samples = 3, rdist = rbinom, args = list(size = 1, prob = 0.275), method = binom.test,
  method.args = list(success = 1), verbose = FALSE, estimand = 0.275)
require(Hmisc)
xYplot(Cbind(estimate, lower, upper) ~ sample, data = results, par.settings = col.mosaic(),
  groups = cover)
```

Figure3.13

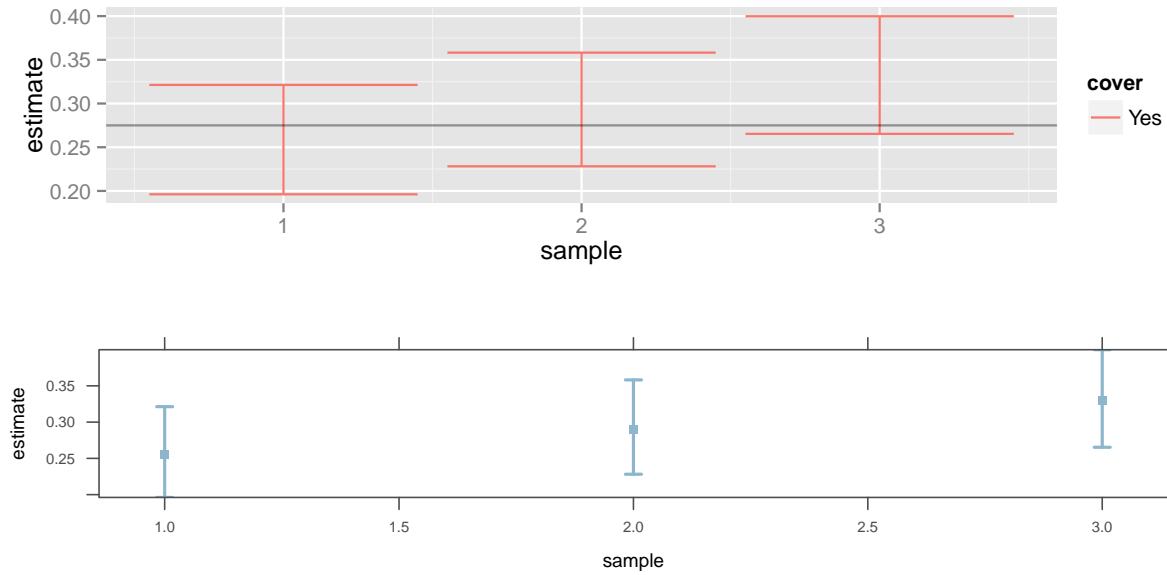
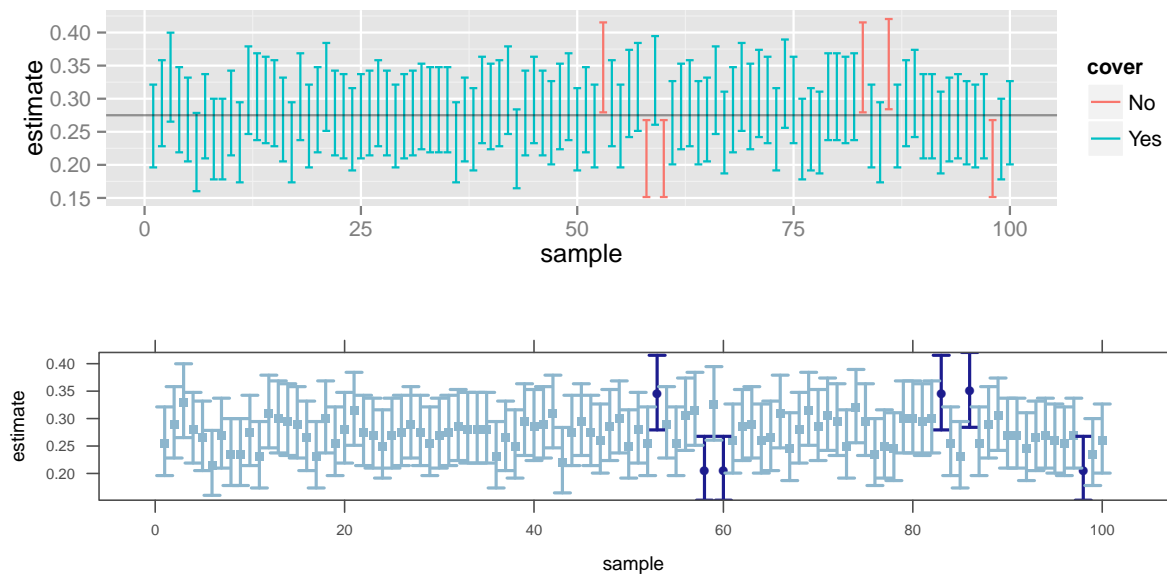


Figure 3.13b

```
results <- CIsim(200, samples = 100, rdist = rbinom, args = list(size = 1, prob = 0.275), method = binom.test,
  method.args = list(success = 1), verbose = FALSE, estimand = 0.275)
require(Hmisc)
xyplot(Cbind(estimate, lower, upper) ~ sample, data = results, par.settings = col.mosaic(),
  groups = cover)
```



Interpreting Confidence Intervals

Example 3.16

```
x.bar <- 27.655
SE <- 0.009
MoE <- 2 * SE
x.bar - MoE
```

```
[1] 27.637
```

```
x.bar + MoE
```

```
[1] 27.673
```

Example3.16

Example 3.17

```
diff.x <- -1.915
SE <- 0.016
MoE <- 2 * SE
diff.x - MoE
```

```
[1] -1.947
```

```
diff.x + MoE
```

```
[1] -1.883
```

Example3.17

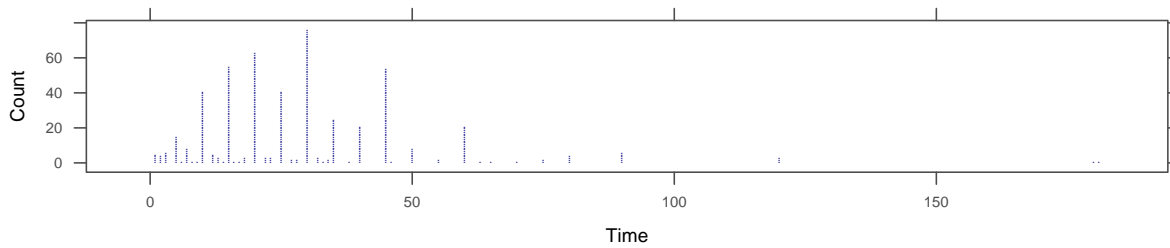
3.3 Constructing Bootstrap Confidence Intervals

Here's the clever idea: We don't have the population, but we have a sample. Probably the sample is similar to the population in many ways. So let's sample from our sample. We'll call it **resampling** (also called **bootstrapping**). We want samples the same size as our original sample, so we will need to sample with replacement. This means that we may pick some members of the population more than once and others not at all. We'll do this many times, however, so each member of our sample will get its fair share. (Notice the similarity to and difference from sampling from populations in the previous sections.)

Figure 3.14

```
dotPlot(~Time, width = 1, data = CommuteAtlanta)
```

Figure3.14



Bootstrap Samples

Table 3.7

The computer can easily do all of the resampling by using the `resample()`.

```
mean(~Time, data = resample(CommuteAtlanta)) # mean commute time in one resample

[1] 30.078

mean(~Time, data = resample(CommuteAtlanta)) # mean commute time in another resample

[1] 30.854

mean(~Time, data = resample(CommuteAtlanta))

[1] 28.324
```

Table3.7

Bootstrap Distribution

Figure 3.16

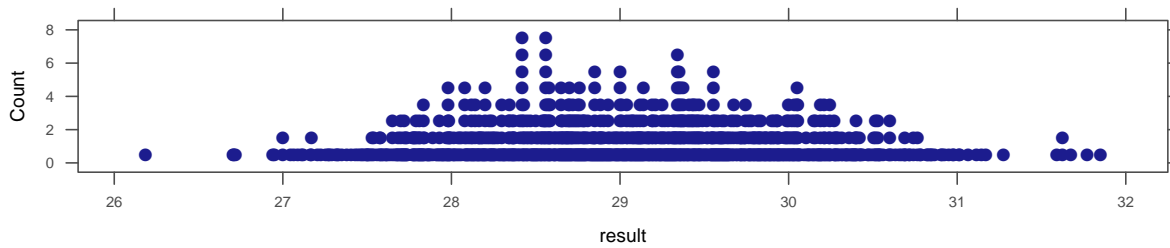
The example below uses data from 500 Atlanta commuters.

```
# Now we'll do it 1000 times
Bootstrap <- do(1000) * mean(~Time, data = resample(CommuteAtlanta))
head(Bootstrap, 3)

  result
1 30.84
2 30.76
3 28.64

# We should check that that our bootstrap distribution has an appropriate shape:
dotPlot(~result, width = 0.005, data = Bootstrap)
```

Figure3.16



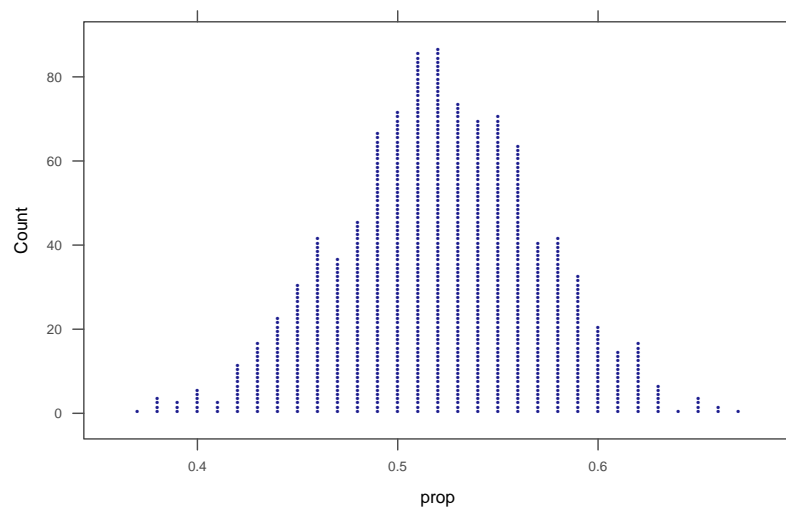
Example 3.19

```
BootP <- do(1000) * rflip(100, 0.52)
head(BootP, 3)
```

Example3.19

```
      n heads tails prop
1 100   51    49 0.51
2 100   48    52 0.48
3 100   52    48 0.52
```

```
dotPlot(~prop, width = 0.01, data = BootP)
```



Example 3.20

Variables can be created in R using the `c()` function then collected into a data frame using the `data.frame()` function.

```
Laughter <- data.frame(NumLaughs = c(16, 22, 9, 31, 6, 42))
mean(~NumLaughs, data = Laughter)
```

Example3.20

```
[1] 21
```

```
mean(~NumLaughs, data = resample(Laughter))

[1] 24

mean(~NumLaughs, data = resample(Laughter))

[1] 23.83333

mean(~NumLaughs, data = resample(Laughter))

[1] 14.5
```

Example3.20b

Estimating Standard Error Based on a Bootstrap Distribution

Example 3.21

Since the shape of the bootstrap distribution from Example 3.19 looks good, we can estimate the standard error.

```
SE <- sd(~prop, data = BootP)
SE

[1] 0.05021711
```

Example3.21

95 % Confidence Interval Based on a Bootstrap Standard Error

Example 3.22

We can again use the standard error to compute a 95% confidence interval.

```
x.bar <- mean(~Time, data = CommuteAtlanta); x.bar

[1] 29.11

SE <- sd(~result, data = Bootstrap ); SE      # standard error

[1] 0.9170349

MoE <- 2 * SE; MoE                          # margin of error for 95% CI

[1] 1.83407
```

Example3.22

```
x.bar - MoE                                     # lower limit of 95% CI

[1] 27.27593

x.bar + MoE                                     # upper limit of 95% CI

[1] 30.94407
```

```
p.hat <- 0.52
SE <- sd(~prop, data = BootP)
SE

[1] 0.05021711

MoE <- 2 * SE
MoE

[1] 0.1004342

p.hat - MoE

[1] 0.4195658

p.hat + MoE

[1] 0.6204342
```

Example 3.22b

The steps used in this example get used in a wide variety of confidence interval situations.

1. Compute the statistic from the original sample.
2. Create a bootstrap distribution by resampling from the sample.
 - (a) same size samples as the original sample
 - (b) with replacement
 - (c) compute the statistic for each sample

The distribution of these statistics is the bootstrap distribution

3. Estimate the standard error SE by computing the standard deviation of the bootstrap distribution.
4. 95% CI is

$$\text{statistic} \pm 2SE$$

3.4 Bootstrap Confidence Intervals Using Percentiles

Confidence Intervals Based on Bootstrap Percentiles

Example 3.23

Another way to create a 95% confidence interval is to use the middle 95% of the bootstrap distribution. The `cdata()` function can compute this for us as follows:

```
cdata(0.95, result, data = Bootstrap)
```

Example3.23

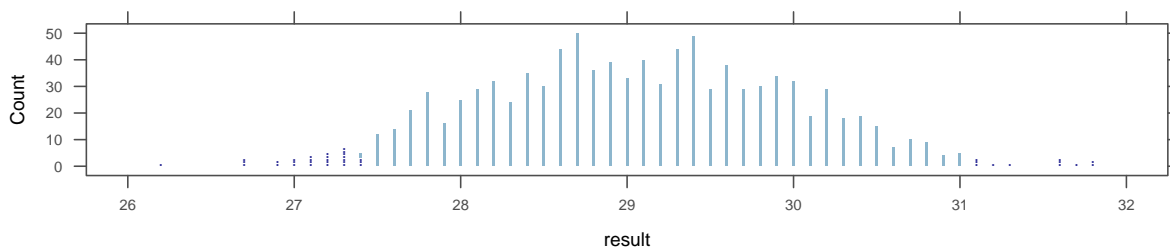
low	hi	central.p
27.35140	30.78685	0.95000

This is not exactly the same as the interval of the original sample, but it is pretty close.

Figure 3.22

```
dotPlot(~result, width = 0.1, groups = (27.43 <= result & result <= 31.05), data = Bootstrap)
```

Figure3.22



Notice the `groups=` for marking the confidence interval.

Example 3.24

One advantage of this method is that it is easy to change the confidence level.

To make a 90% and 99% confidence interval, we use the middle 90% and 99% of the sample distribution instead.

```
cdata(0.9, result, data = Bootstrap)
```

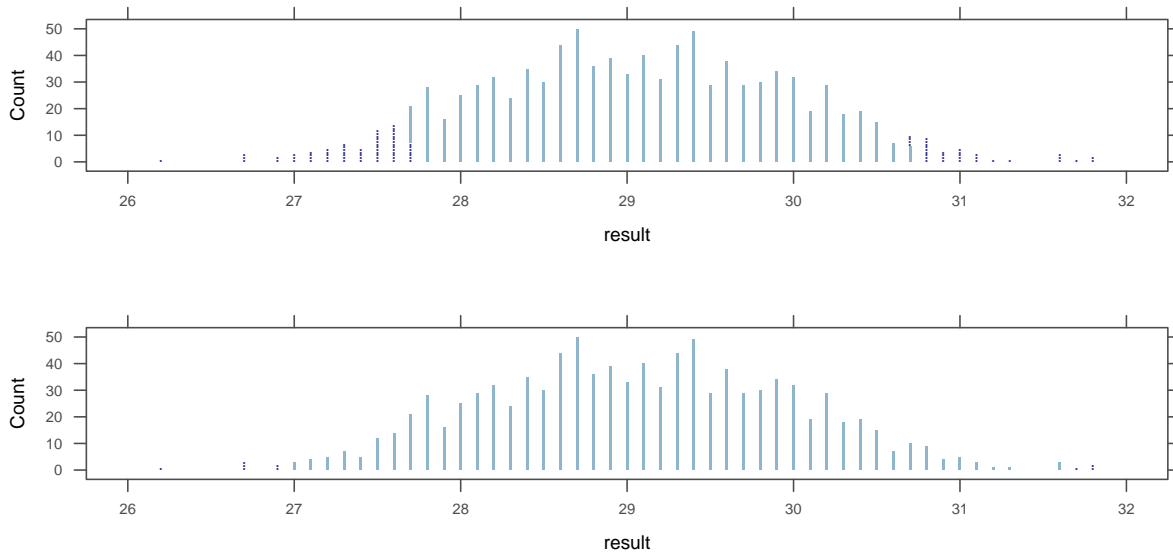
Example3.24

low	hi	central.p
27.6139	30.5281	0.9000

```
dotPlot(~result, width = 0.1, groups = (27.7 <= result & result <= 30.71), data = Bootstrap)
cdata(0.99, result, data = Bootstrap)
```

```
low      hi central.p
26.94797 31.58818  0.99000
```

```
dotPlot(~result, width = 0.1, groups = (26.98 <= result & result <= 31.63), data = Bootstrap)
```

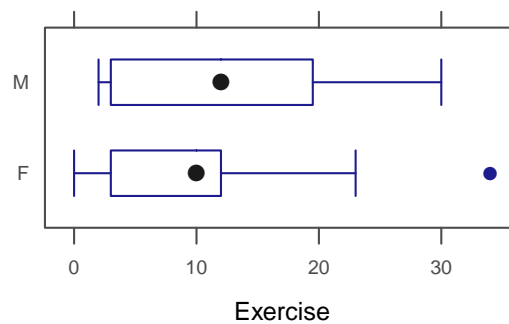


Finding Confidence Intervals for Many Different Parameters

Figure 3.24

```
bwplot(Gender ~ Exercise, data = ExerciseHours)
```

Figure3.24



Example 3.25

```
head(ExerciseHours)
```

Example3.25

	Year	Gender	Hand	Exercise	TV	Pulse	Pierces
1	4	M	l	15	5	57	0
2	2	M	l	20	14	70	0
3	3	F	r	2	3	70	2
4	1	F	l	10	5	66	3
5	1	M	r	8	2	62	0
6	1	M	r	14	14	62	0

```
favstats(~Exercise | Gender, data = ExerciseHours)
```

	Gender	min	Q1	median	Q3	max	mean	sd	n	missing
1	F	0	3	10	12.00	34	9.4	7.407359	30	0
2	M	2	3	12	19.25	30	12.4	8.798325	20	0

```
stat <- diff(mean(Exercise ~ Gender, data = ExerciseHours))
stat
```

```
M
3
```

Example3.25b

```
BootE <- do(3000) * diff(mean(Exercise ~ Gender, data = resample(ExerciseHours)))
head(BootE, 3)
```

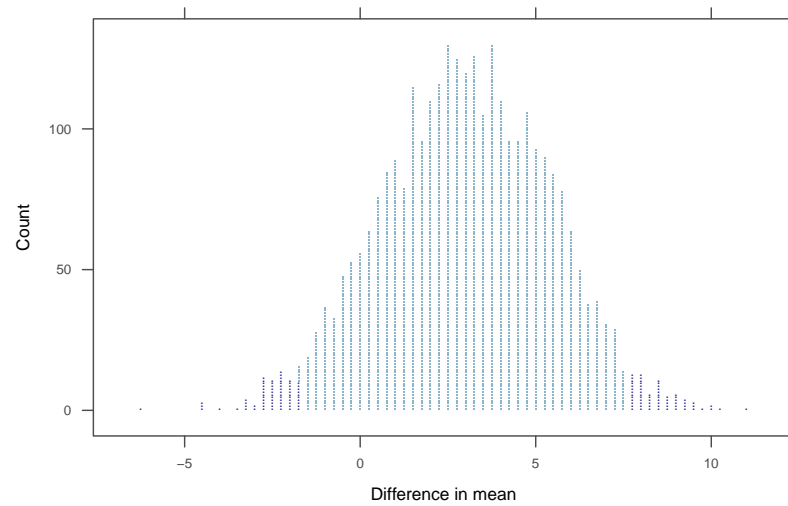
```
      M
1 -0.2372
2  5.2394
3  3.1461
```

Example3.25c

```
cdata(0.95, M, data = BootE)
```

	low	hi	central.p
	-1.642548	7.489907	0.950000

```
dotPlot(~M, width = 0.25, cex = 0.75, groups = (-1.717 <= M & M <= 7.633), xlab = "Difference in mean",
data = BootE)
```



```
SE <- sd(~M, data = BootE)
SE
```

```
[1] 2.369424
```

```
stat - 2 * SE
```

```
      M
-1.738847
```

```
stat + 2 * SE
```

```
      M
7.738847
```

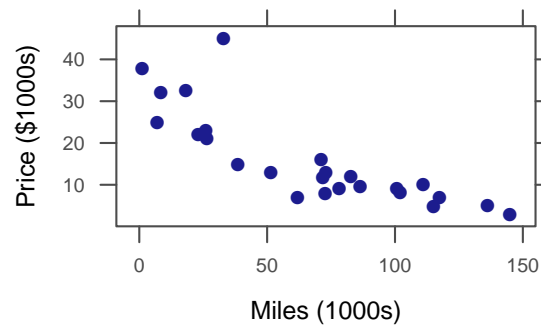
Example3.25d

Figure 3.26

```
xyplot(Price ~ Miles, ylab = "Price ($1000s)", xlab = "Miles (1000s)", data = MustangPrice)
cor(Price ~ Miles, data = MustangPrice)
```

```
[1] -0.8246164
```

Figure3.26



Example 3.26

```
BootM <- do(5000) * cor(Price ~ Miles, data = resample((MustangPrice)))
head(BootM, 3)
```

Example3.26

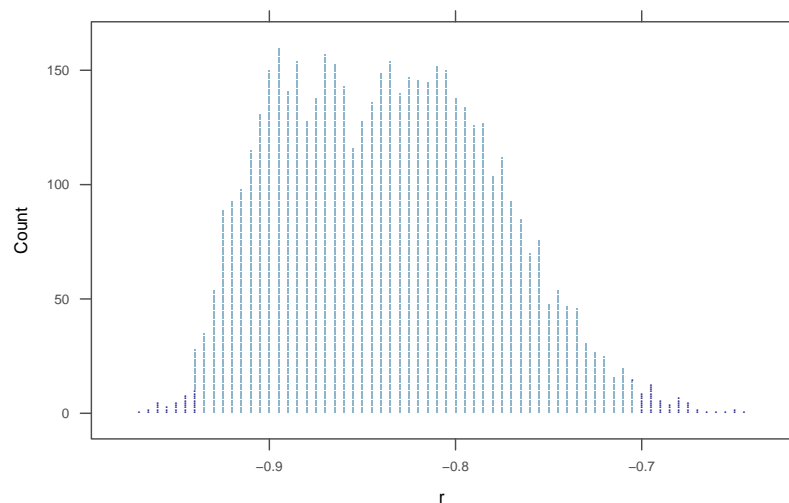
```
      result
1 -0.7513
2 -0.9226
3 -0.8620
```

```
cdata(0.98, result, data = BootM)
```

Example3.26b

```
      low      hi central.p
-0.9376978 -0.7020222 0.9800000
```

```
dotPlot(~result, width = 0.005, groups = (-0.94 <= result & result <= -0.705), xlab = "r",
        data = BootM)
```



Another Look at the Effect of Sample Size

Example 3.27

```
BootP400 <- do(1000) * rflip(400, 0.52)
head(BootP400, 3)
```

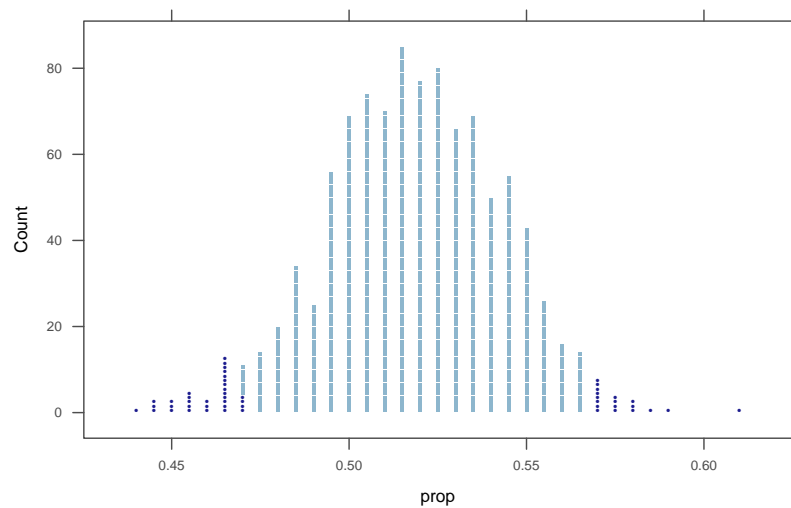
Example3.27

	n	heads	tails	prop
1	400	189	211	0.4725
2	400	212	188	0.5300
3	400	210	190	0.5250

```
cdata(0.95, prop, data = BootP400)
```

low	hi	central.p
0.4675	0.5651	0.9500

```
dotPlot(~prop, width = 0.005, groups = (0.472 <= prop & prop <= 0.568), data = BootP400)
```



One Caution on Constructing Bootstrap Confidence Intervals

Example 3.28

```
median(~Price, data = MustangPrice)
```

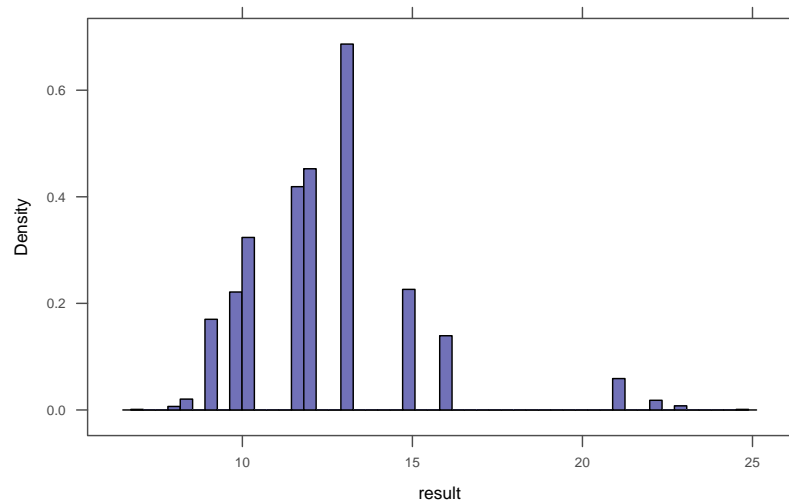
Example3.28

```
[1] 11.9
```

```
Boot.Mustang <- do(5000) * median(~Price, data = resample(MustangPrice))
head(Boot.Mustang, 3)
```

```
result
1  11.9
2  13.0
3  11.9

histogram(~result, n = 50, data = Boot.Mustang)
```



This time the histogram does not have the desired shape. There are two problems:

1. The distribution is not symmetric. (It is right skewed.)
2. The distribution has spikes and gaps.

Since the median must be an element of the sample when the sample size is 25, there are only 25 possible values for the median (and some of these are *very* unlikely).

Since the bootstrap distribution does not look like a normal distribution (bell-shaped, symmetric), we cannot safely use our methods for creating a confidence interval.

4

Hypothesis Tests

4.1 Introducing Hypothesis Tests

The 4-step outline

The following 4-step outline is a useful way to organize the ideas of hypothesis testing.

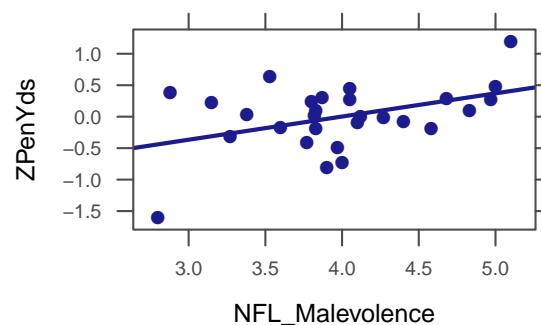
1. State the Null and Alternative Hypotheses
2. Compute the Test Statistic
The test statistic is a number that summarizes the evidence
3. Determine the p-value (from the Randomization Distribution)
4. Draw a conclusion

Null and Alternative Hypotheses

Figure 4.1

```
xyplot(ZPenYds ~ NFL_Malevolence, type = c("p", "r"), data = MalevolentUniformsNFL)
```

Figure4.1



4.2 Measuring Evidence with P-values

Randomization distributions are a bit like bootstrap distributions except that instead of resampling from our sample (in an attempt to approximate resampling from the population), we need to sample from a situation in which our null hypothesis is true.

P-values from Randomization Distributions

Example 4.13

Testing one proportion.

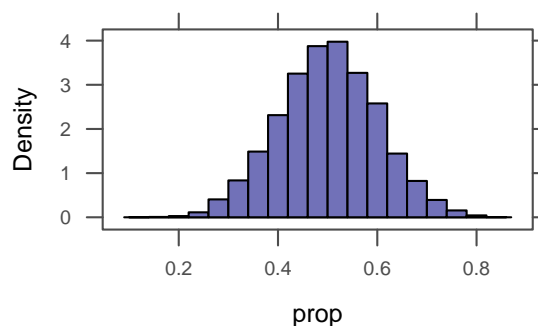
1. $H_0: p = 0.5$
 $H_a: p > 0.5$
2. Test statistic: $\hat{p} = 16/25$ (the sample proportion)
3. We can simulate a world in which $p = 0.5$ using `rflip()`:

```
Randomization.Match <- do(10000) * rflip(25, 0.5) # 25 because n=25
head(Randomization.Match)
```

Example4.13

	n	heads	tails	prop
1	25	13	12	0.52
2	25	9	16	0.36
3	25	14	11	0.56
4	25	11	14	0.44
5	25	14	11	0.56
6	25	14	11	0.56

```
histogram(~prop, width = 0.04, data = Randomization.Match)
```



Here we find the proportion of the simulations which resulted in 16 or more matches out of 25, or 0.64 or greater, for the p-value.

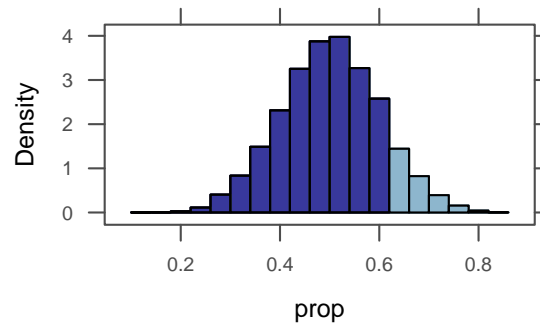
```
prop(~(prop >= 0.64), data = Randomization.Match) # 16/25
```

Example4.13b

```
target level: TRUE; other levels: FALSE
```


TRUE
0.1144

```
histogram(~prop, width = 0.04, groups = (prop >= 0.64), data = Randomization.Match)
```



Example 4.15

```
prop(~(prop >= 0.6), data = Randomization.Match) # 15/25
```

Example4.15

target level: TRUE; other levels: FALSE

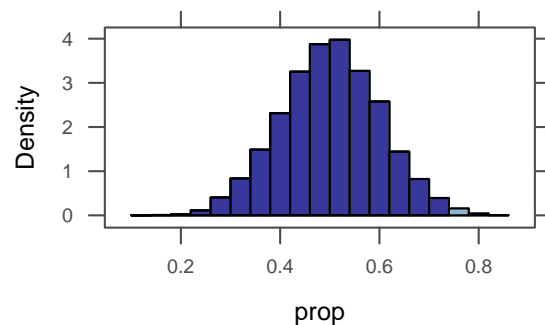
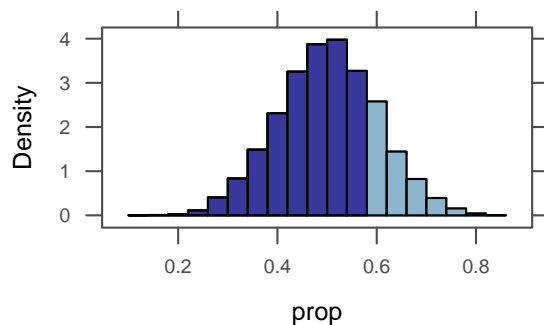
TRUE
0.2175

```
prop(~(prop >= 0.76), data = Randomization.Match) # 19/25
```

target level: TRUE; other levels: FALSE

TRUE
0.0081

```
histogram(~prop, width = 0.04, groups = (prop >= 0.6), data = Randomization.Match)
histogram(~prop, width = 0.04, groups = (prop >= 0.76), data = Randomization.Match)
```



Example 4.16

```
prop(~(prop >= 0.88), data = Randomization.Match) # 22/25
```

Example4.16

target level: TRUE; other levels: FALSE

TRUE
0

```
histogram(~prop, width = 0.04, v = c(0.88), data = Randomization.Match)
```

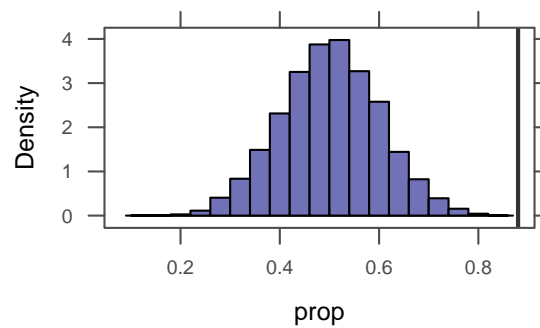
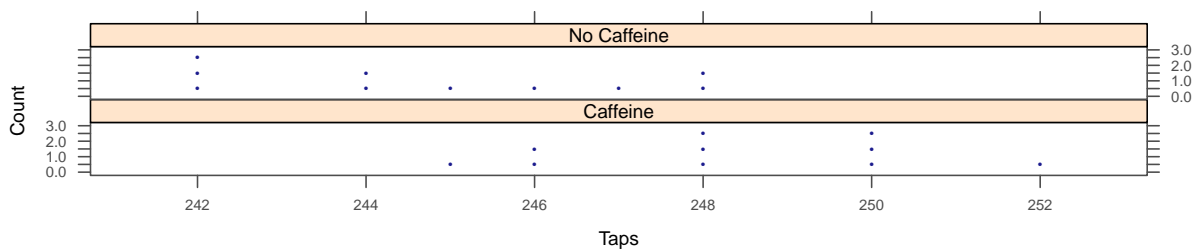


Figure 4.10

```
dotPlot(~Taps | Group, layout = c(1, 2), width = 1, cex = 0.3, data = CaffeineTaps)
```

Figure4.10



Example 4.18

Testing two means.

```
mean(Taps ~ Group, data = CaffeineTaps)
```

Example4.18

Caffeine	No Caffeine
248.3	244.8

```
diff(mean(Taps ~ Group, data = CaffeineTaps))
```

```
No Caffeine
-3.5
```

1. $H_0: \mu_1 = \mu_2$

$H_a: \mu_1 > \mu_2$

2. Test statistic: $\bar{x}_1 - \bar{x}_2 = 3.5$ (the difference in sample means)

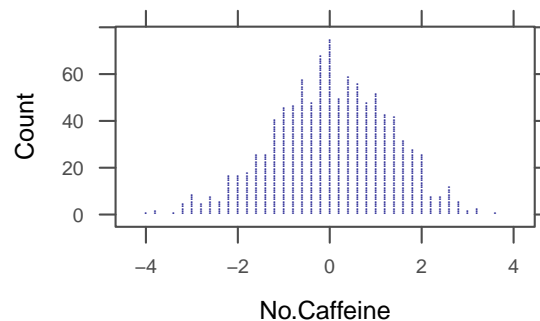
3. We simulate a world in which $\mu_1 = \mu_2$ or $\mu_1 - \mu_2 = 0$:

```
Randomization.Caff <- do(1000) * ediff(mean(Taps ~ shuffle(Group), data = CaffeineTaps))
head(Randomization.Caff, 3)
```

Example4.18b

```
V1 No.Caffeine
1 NA 2.1
2 NA 0.1
3 NA 3.7
```

```
dotPlot(~No.Caffeine, width = 0.2, data = Randomization.Caff)
```



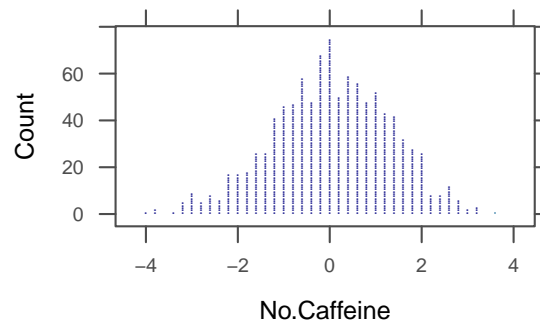
```
prop(~(No.Caffeine >= 3.5), data = Randomization.Caff)
```

Example4.18c

```
target level: TRUE; other levels: FALSE
```

```
TRUE
0.001
```

```
dotPlot(~No.Caffeine, width = 0.2, groups = (No.Caffeine >= 3.5), data = Randomization.Caff)
```



P-values and the Alternative Hypothesis

Example 4.19

Testing one proportion.

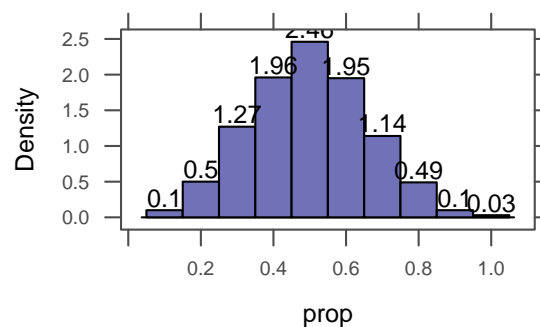
1. $H_0: p = 0.5$
 $H_a: p > 0.5$
2. Test statistic: $\hat{p} = 0.8, 0.6, 0.4$ (the sample proportion of 8/10, 6/10, 4/10 heads)
3. We simulate a world in which $p = 0.5$:

```
RandomizationDist <- do(1000) * rflip(10, 0.5) # 10 because n=10
head(RandomizationDist)
```

	n	heads	tails	prop
1	10	6	4	0.6
2	10	5	5	0.5
3	10	5	5	0.5
4	10	7	3	0.7
5	10	5	5	0.5
6	10	5	5	0.5

```
histogram(~prop, label = TRUE, width = 1/10, data = RandomizationDist)
```

Example4.19



```
prop(~(prop >= 0.8), data = RandomizationDist)

target level: TRUE; other levels: FALSE

TRUE
0.064

prop(~(prop >= 0.6), data = RandomizationDist)

target level: TRUE; other levels: FALSE

TRUE
0.386

prop(~(prop >= 0.4), data = RandomizationDist)

target level: TRUE; other levels: FALSE

TRUE
0.836
```

Example4.19b

Example 4.20

Testing one proportion.

1. $H_0: p = 0.5$
 $H_a: p \neq 0.5$
2. Test statistic: $\hat{p} = 0.8$ (the sample proportion of 8/10 heads)
3. We use the simulated world in which $p = 0.5$:

```
prop(~ (prop >= 0.8), data = RandomizationDist)

target level: TRUE; other levels: FALSE

TRUE
0.064

prop(~ (prop <= 0.2), data = RandomizationDist)

target level: TRUE; other levels: FALSE

TRUE
0.055
```

Example4.20b

```
# a 2-sided p-value is the sum of the values above
prop(~(prop <= 0.2 | prop >= 0.8), data = RandomizationDist)

target level: TRUE; other levels: FALSE
```

Example4.20b

```

TRUE
0.119

# We can also approximate the p-value by doubling one side
2 * prop(~prop >= 0.8, data = RandomizationDist)

target level: TRUE; other levels: FALSE

TRUE
0.128

```

4.3 Determining Statistical Significance

Less Formal Statistical Decisions

Example 4.27

Testing two means.

```

head(Smiles)

  Leniency Group
1      7.0 smile
2      3.0 smile
3      6.0 smile
4      4.5 smile
5      3.5 smile
6      4.0 smile

mean(Leniency ~ Group, data = Smiles)

  neutral    smile
4.117647 4.911765

diff(mean(Leniency ~ Group, data = Smiles))

  smile
0.7941176

```

Example4.27

1. $H_0: \mu_1 = \mu_2$
 $H_a: \mu_1 \neq \mu_2$
2. Test statistic: $\bar{x}_1 - \bar{x}_2 = 0.79$ (the difference in sample means)
3. We simulate a world in which $\mu_1 = \mu_2$:

```
Randomization.Smiles <- do(1000) * diff(mean(Leniency ~ shuffle(Group), data = Smiles))
head(Randomization.Smiles, 3)
```

Example4.27b

```
smile
1 0.29412
2 -0.05882
3 0.08824
```

```
prop(~ (smile <= -0.79 | smile >= 0.79), data = Randomization.Smiles)
```

Example4.27c

```
target level: TRUE; other levels: FALSE
```

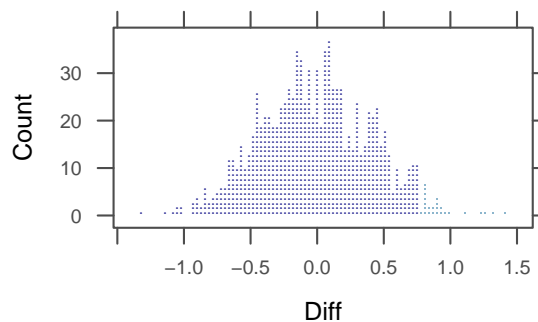
```
TRUE
0.054
```

```
2 * prop(~ smile >= 0.79, data = Randomization.Smiles)
```

```
target level: TRUE; other levels: FALSE
```

```
TRUE
0.05
```

```
dotPlot(~ smile, width = 0.03, cex = 0.5, groups = (smile >= 0.79),
        xlab = "Diff", data = Randomization.Smiles)
```



Now we find the p-value to test a difference of 0.76:

```
prop(~(smile <= -0.76 | smile >= 0.76), data = Randomization.Smiles)
```

Example4.27d

```
target level: TRUE; other levels: FALSE
```

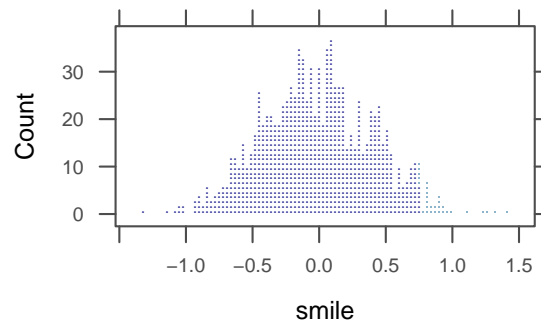
```
TRUE
0.062
```

```
2 * prop(~smile >= 0.76, data = Randomization.Smiles)
```

```
target level: TRUE; other levels: FALSE
```

```
TRUE
0.06
```

```
dotPlot(~smile, width = 0.03, cex = 0.5, groups = (smile >= 0.76), data = Randomization.Smiles)
```



4.4 Creating Randomization Distributions

In order to use these methods to estimate a p-value, we must be able to generate a randomization distribution. In the case of a test with null hypothesis claiming that a proportion has a particular value (e.g, $H_0: p = 0.5$), this is pretty easy. If the population has proportion 0.50, we can simulate sampling from that proportion by flipping a fair coin. If the proportion is some value other than 0.50, we simply flip a coin that has the appropriate probability of resulting in heads. So the general template for creating such a randomization distribution is

```
do(1000) * rflip(n, hypothesized_proportion)
```

where n is the size of the original sample.

In other situations, it can be more challenging to create a randomization distribution because the null hypothesis does not directly specify all of the information needed to simulate samples.

- $H_0: p_1 = p_2$
This would be simple if we knew the value of p_1 and p_2 (we could use `rflip()` twice, once for each group),
- $H_0: \mu = \text{some number}$
Just knowing the mean does not tell us enough about the distribution. We need to know about its shape. (We might need to know the standard deviation, for example, or whether the distribution is skewed.)
- $H_0: \mu_1 \neq \mu_2$ some number.
Now we don't know the common mean and we don't know the things mentioned in the previous example either.

So how do we come up with randomization distribution?

The main criteria to consider when creating randomization samples for a statistical test are:

- Be consistent with the null hypothesis.
If we don't do this, we won't be testing our null hypothesis.
- Use the data in the original sample.
With luck, the original data will shed light on some aspects of the distribution that are not determined by null hypothesis.
- Reflect the way the original data were collected.

Randomization Test for a Difference in Proportions: Cocaine Addiction

Data 4.7

Data 4.7 in the text describes some data that are not in a data frame. This often happens when a data set has only categorical variables because a simple table completely describes the distributions involved. Here's the table from the book:¹

	Relapse	No Relapse
Lithium	18	6
Placebo	20	4

Here's one way to create the data in R:

```
Cocaine <- rbind(
  do(18) * data.frame( treatment = "Lithium",   response="Relapse"),
  do(6)  * data.frame( treatment = "Lithium",   response="No Relapse"),
  do(20) * data.frame( treatment = "Placebo",   response="Relapse"),
  do(4)  * data.frame( treatment = "Placebo",   response="No Relapse")
)
```

Section4.4b

Example 4.29

Testing two proportions.

```
tally(response ~ treatment, data = Cocaine)
```

```
      treatment
response  Lithium Placebo
Relapse      18      20
No Relapse    6       4
```

```
prop(response ~ treatment, data = Cocaine)
```

```
target level: Relapse; other levels: No Relapse
```

```
Relapse.Lithium Relapse.Placebo
0.7500000      0.8333333
```

```
diff(prop(response ~ treatment, data = Cocaine))
```

```
target level: Relapse; other levels: No Relapse
```

```
Relapse.Placebo
0.08333333
```

Example4.29

¹The book includes data on an additional treatment group which we are omitting here.

1. $H_0: p_1 = p_2$
 $H_a: p_1 < p_2$
2. Test statistic: $\hat{p}_1 = \hat{p}_2$ (the difference in sample proportions)
3. We simulate a world in which $p_1 = p_2$ or $p_1 - p_2 = 0$:

```
Randomization.Coc <- do(5000) * diff(prop(response ~ shuffle(treatment), data = Cocaine))
head(Randomization.Coc)
```

Example4.29b

```
Relapse.Placebo
1      0.00000
2     -0.16667
3     -0.08333
4      0.25000
5     -0.16667
6      0.00000
```

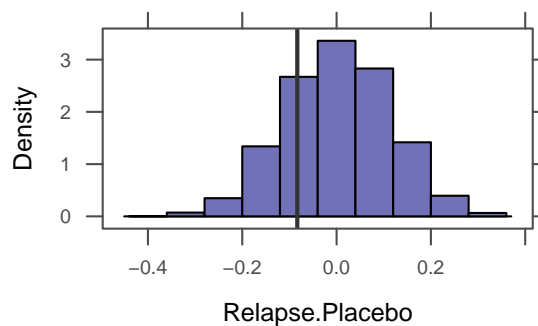
```
prop(~(Relapse.Placebo < -0.0833), data = Randomization.Coc)
```

Example4.29c

```
target level: TRUE; other levels: FALSE
```

```
TRUE
0.3546
```

```
histogram(~Relapse.Placebo, data = Randomization.Coc, v = c(-0.0833), width = 0.08)
```



Randomization Test for a Correlation: Malevolent Uniforms and Penalties

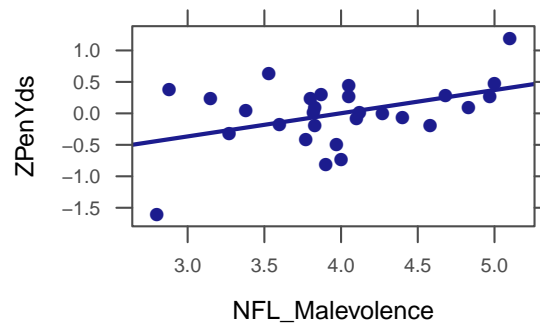
Example 4.31

Testing correlation.

```
xypplot(ZPenYds ~ NFL_Malevolence, type = c("p", "r"), data = MalevolentUniformsNFL)
cor(ZPenYds ~ NFL_Malevolence, data = MalevolentUniformsNFL)
```

Example4.31

```
[1] 0.429796
```



1. $H_0: \rho = 0$
 $H_a: \rho > 0$
2. Test statistic: $r = 0.43$ (the sample correlation)
3. We simulate a world in which $\rho = 0$:

```
Randomization.Mal <- do(10000) * cor(NFL_Malevolence ~ shuffle(ZPenYds),
                                     data = MalevolentUniformsNFL)
head(Randomization.Mal)
```

Example4.31b

```
      result
1  0.004128
2 -0.154235
3 -0.194265
4  0.147729
5 -0.102568
6 -0.047919
```

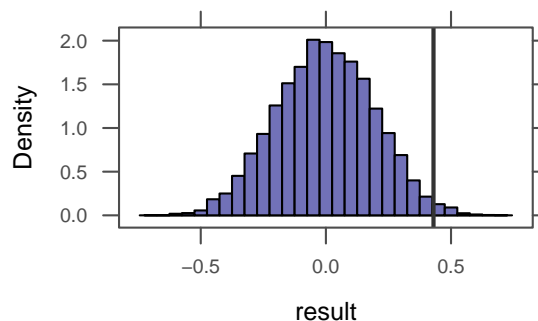
```
prop(~(result > 0.43), data = Randomization.Mal)
```

Example4.32c

```
target level: TRUE; other levels: FALSE
```

```
TRUE
0.0121
```

```
histogram(~result, v = c(0.43), width = 0.05, data = Randomization.Mal)
```



Randomization Test for a Mean: Body Temperature

Example 4.33

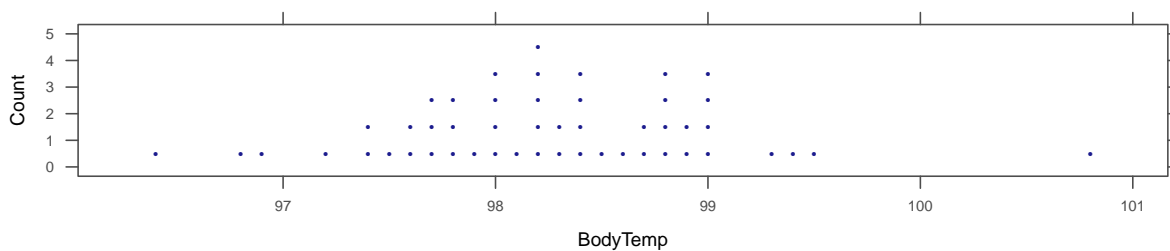
Testing one mean.

```
mean(~BodyTemp, data = BodyTemp50)
```

```
[1] 98.26
```

```
dotPlot(~BodyTemp, v = c(98.26), width = 0.1, cex = 0.2, data = BodyTemp50)
```

Example4.33



1. $H_0: \mu = 98.6$

$H_a: \mu \neq 98.6$

2. Test statistic: $\bar{x} = 98.26$ (the sample mean)

Notice that the test statistic differs a bit from 98.6

```
98.6 - mean(~BodyTemp, data = BodyTemp50)
```

```
[1] 0.34
```

Example4.33c

But might this just be random variation? We need a randomization distribution to compare against.

3. If we resample, the mean will not be 98.6. But we shift the distribution a bit, then we will have the desired mean while preserving the shape of the distribution indicated by our sample. We simulate a world in which $\mu = 98.6$:

```
Randomization.Temp <- do(10000) * (mean(~BodyTemp, data = resample(BodyTemp50)) + 0.34)
head(Randomization.Temp, 3)
```

Example4.33c

```
result
1 98.61
2 98.52
3 98.61
```

```
mean(~result, data = Randomization.Temp)
```

```
[1] 98.6
```

```
cdata(0.95, result, data = Randomization.Temp)
```

```
low      hi central.p
98.39    98.81    0.95
```

From this we can estimate the p-value:

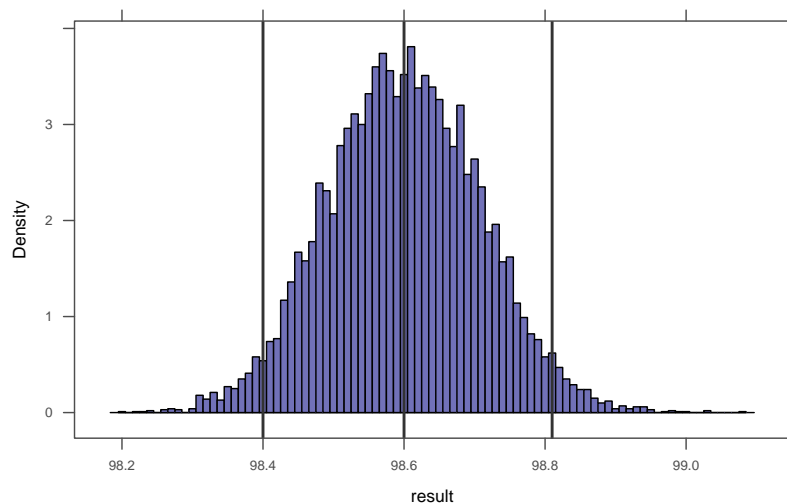
```
prop(~abs(result - 98.6) > 0.34, data = Randomization.Temp)

target level: TRUE; other levels: FALSE

TRUE
0.0019

histogram(~result, width = 0.01, v = c(98.4, 98.6, 98.81), data = Randomization.Temp)
```

Example4.33d



How do we interpret this (estimated) p-value of 0? Is it impossible to have a sample mean so far from 98.6 if the true population mean is 98.6? No. This merely means that we didn't see any such cases *in our 10000 randomization samples*. We might estimate the p-value as $p < 0.001$. Generally, to more accurately estimate small p-values, we must use many more randomization samples.

Example 4.33: A different approach

An equivalent way to do the preceding test is based on a different way of expressing our hypotheses.

1. $H_0: \mu - 98.6 = 0$
 $H_a: \mu - 98.6 \neq 0$
2. Test statistic: $\bar{x} - 98.6 = -0.34$
3. We create a randomization distribution centered at $\mu - 98.6 = 0$:

```
Randomization.Temp2 <- do(5000) * (mean(~BodyTemp, data = resample(BodyTemp50)) - 98.26)
head(Randomization.Temp2, 3)

result
1 -0.194
2 0.050
3 -0.094
```

Example4.33e

```
mean(~result, data = Randomization.Temp2)

[1] -0.0006556
```

From this we can estimate the p-value:

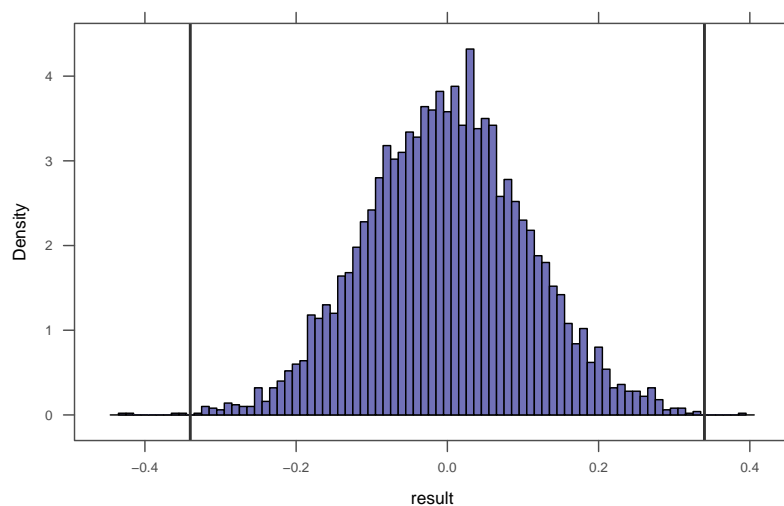
```
prop(~abs(result) > 0.34, data = Randomization.Temp2)

target level: TRUE; other levels: FALSE

TRUE
0.001

histogram(~result, width = 0.01, v = c(0.34, -0.34), data = Randomization.Temp2)
```

Example4.33f,



Often there are multiple ways to express the same hypothesis test.

4.5 Confidence Intervals and Hypothesis Tests

If your randomization distribution is centered at the wrong value, then it isn't simulating a world in which the null hypothesis is true. This would happen, for example, if we got confused about randomization vs. bootstrapping.

Randomization and Bootstrap Distributions

Figure 4.32

```
Boot.Temp <- do(5000) * mean(~BodyTemp, data = resample(BodyTemp50))
head(Boot.Temp, 3)
```

Figure4.32

```

  result
1  98.18
2  98.25
3  98.08

mean(~result, data = Boot.Temp)

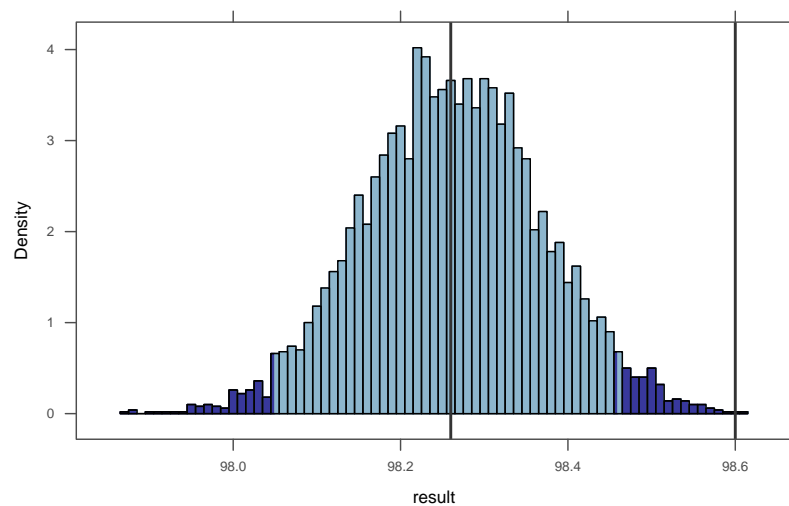
[1] 98.26

cdata(0.95, result, data = Boot.Temp)

      low      hi central.p
98.05    98.47      0.95

histogram(~result, width = 0.01, v = c(98.26, 98.6), groups = (98.05 <= result & result <=
  98.46), data = Boot.Temp)

```



Notice that the distribution is now centered at our test statistic instead of at the value from the null hypothesis.

Example 4.35

1. $H_0: \mu = 98.4$
 $H_a: \mu \neq 98.4$
2. Test statistic: $\bar{x} = 98.26$ (the sample mean)
3. We simulate a world in which $\mu = 98.4$:

```

Randomization.Temp3 <- do(5000) * (mean(~BodyTemp, data = resample(BodyTemp50)) + 0.14)
head(Randomization.Temp3, 3)

```

Example4.35

```

  result
1  98.36

```

```

2  98.45
3  98.53

mean(~result, data = Randomization.Temp3)

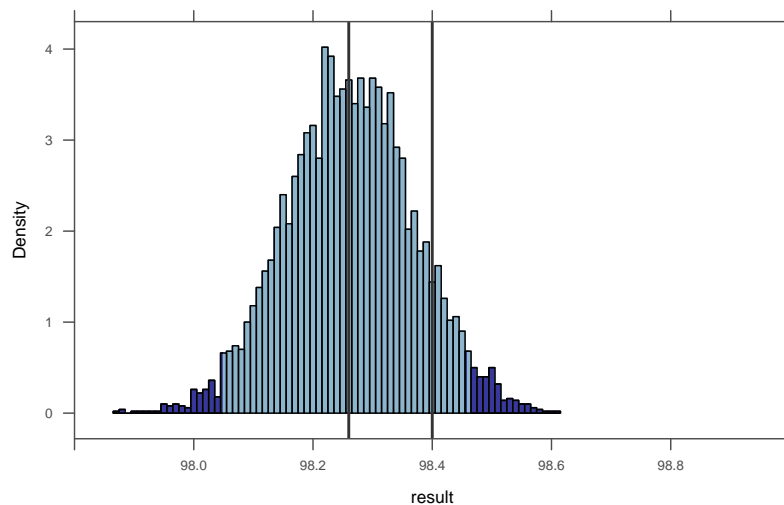
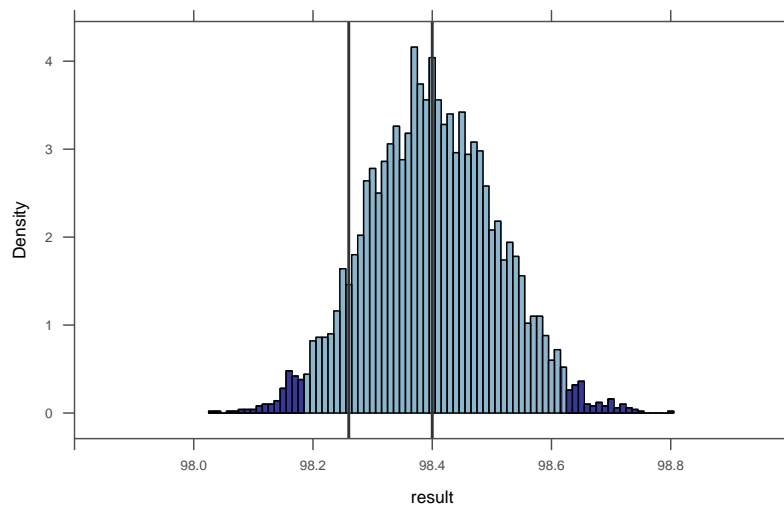
[1] 98.4

cdata(0.95, result, data = Randomization.Temp3)

      low      hi central.p
98.19    98.61      0.95

histogram(~result, width = 0.01, v = c(98.26, 98.4), groups = (98.19 <= result & result <=
  98.62), xlim = c(97.8, 99), data = Randomization.Temp3) # randomization
histogram(~result, width = 0.01, v = c(98.26, 98.4), groups = (98.05 <= result & result <=
  98.46), xlim = c(97.8, 99), data = Boot.Temp) # bootstrap

```



5

Approximating with a Distribution

5.1 Normal Distributions

Density Curves

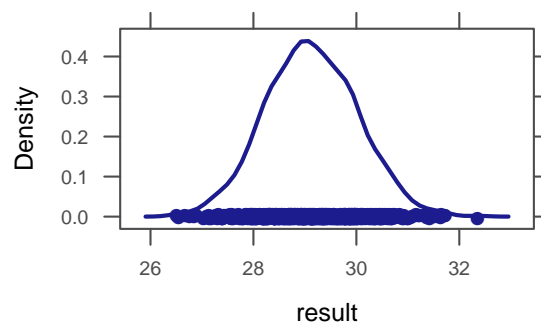
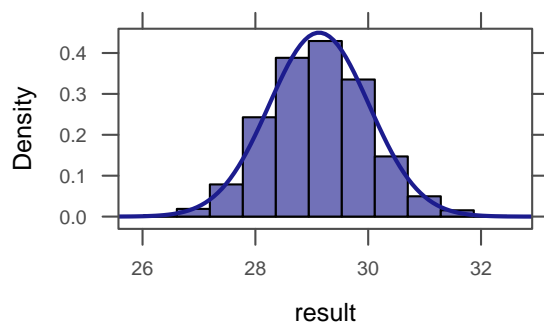
Example 5.1

```
Bootstrap <- do(1000) * mean(~Time, data = resample(CommuteAtlanta))  
head(Bootstrap, 3)
```

Example5.1

```
  result  
1  28.52  
2  29.24  
3  29.16
```

```
histogram(~result, density = TRUE, data = Bootstrap)  
densityplot(~result, data = Bootstrap)
```



```
prop(~(result <= 30), data = Bootstrap) # proportion less than 30 min
```

Example5.1b

```

    target level:  TRUE;  other levels:  FALSE

TRUE
0.844

prop(~(result >= 31), data = Bootstrap) # proportion greater than 31 min

    target level:  TRUE;  other levels:  FALSE

TRUE
0.022

prop(~(result >= 30 & result <= 31), data = Bootstrap) # proportion between 30 and 31 min

    target level:  TRUE;  other levels:  FALSE

TRUE
0.134

```

Normal Distributions

Normal distributions

- are symmetric, unimodal, and bell-shaped
- can have any combination of mean and standard deviation (as long as the standard deviation is positive)
- satisfy the 68–95–99.7 rule:

Approximately 68% of any normal distribution lies within 1 standard deviation of the mean.

Approximately 95% of any normal distribution lies within 2 standard deviations of the mean.

Approximately 99.7% of any normal distribution lies within 3 standard deviations of the mean.

Many naturally occurring distributions are approximately normally distributed. Normal distributions are also an important part of statistical inference. The `plotDist()` function can be used to plot many common distributions.

Figure 5.5

```

plotDist("norm", mean = 0, sd = 1, xlim = c(-5, 6))
plotDist("norm", mean = 2, sd = 1, col = "red", add = TRUE)

```

Figure5.5

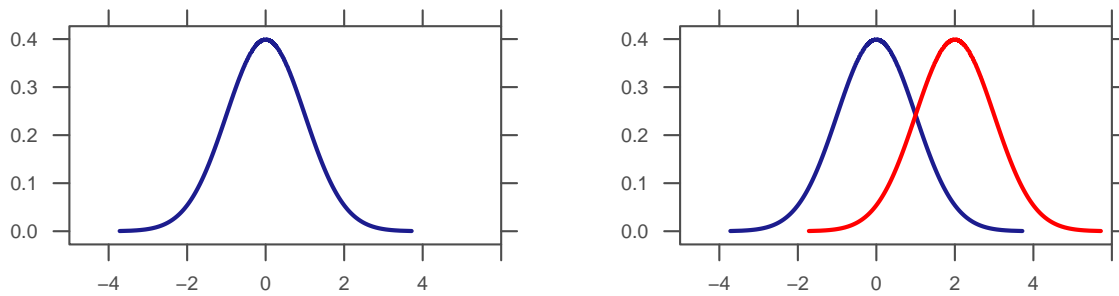
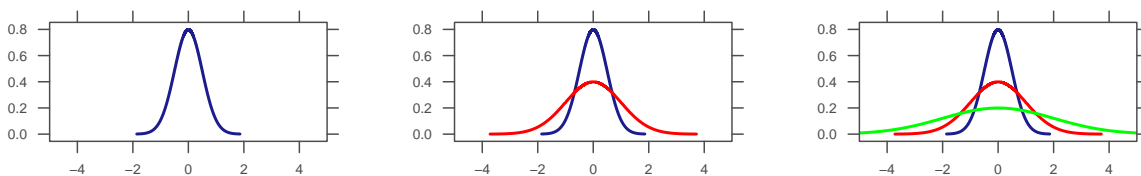


Figure 5.5b

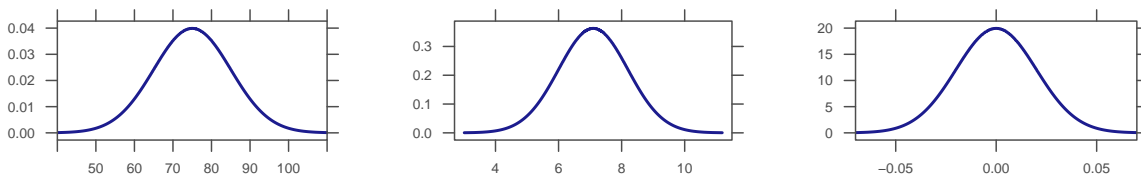
```
plotDist("norm", mean = 0, sd = 0.5, xlim = c(-5, 5))
plotDist("norm", mean = 0, sd = 1, add = TRUE, col = "red")
plotDist("norm", mean = 0, sd = 2, add = TRUE, col = "green")
```



Example 5.2

```
plotDist("norm", mean = 75, sd = 10, xlim = c(40, 110))
plotDist("norm", mean = 7.1, sd = 1.1, xlim = c(2.7, 11.5))
plotDist("norm", mean = 0, sd = 0.02, xlim = c(-0.07, 0.07))
```

Example 5.2



Finding Normal Probabilities and Percentiles

The two main functions we need for working with normal distributions are `pnorm()` and `qnorm()`. `pnorm()` computes the proportion of a normal distribution below a specified value:

$$\text{pnorm}(x, \text{mean}=\mu, \text{sd}=\sigma) = \Pr(X \leq x)$$

when $X \sim \text{Norm}(\mu, \sigma)$.

We can obtain arbitrary probabilities using `pnorm()`

Example 5.3

```
pnorm(90, 75, 10, lower.tail = FALSE) # proportion of scores above 90
```

Example5.3

```
[1] 0.0668072
```

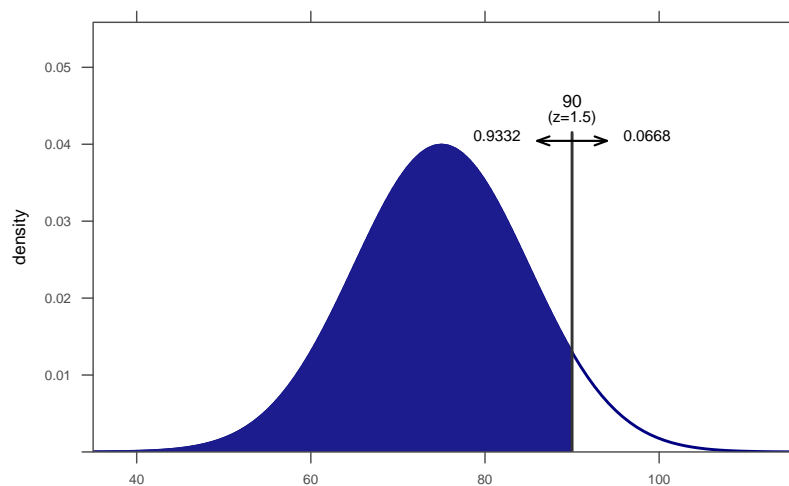
```
xpnorm(90, 75, 10, lower.tail = FALSE)
```

If $X \sim N(75, 10)$, then

$P(X \leq 90) = P(Z \leq 1.5) = 0.9332$

$P(X > 90) = P(Z > 1.5) = 0.0668$

```
[1] 0.0668072
```



The `xpnorm()` function gives a bit more verbose output and also gives you a picture. Notice the `lower.tail=FALSE`. This is added because the default for `pnorm()` and `xpnorm()` finds the lower tail, not the upper tail. However, we can also subtract the proportion of the lower tail from 1 to find the the proportion of the upper tail.

Example 5.4

`qnorm()` goes the other direction: You provide the quantile (percentile expressed as a decimal) and R gives you the value.

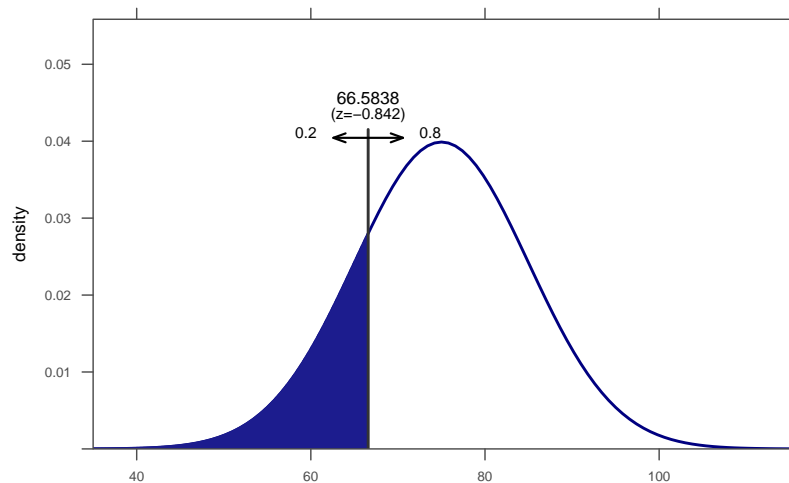
```
qnorm(0.2, 75, 10) # 20th percentile in Norm(75, 10)
```

Example5.4

```
[1] 66.58379
```

```
xqnorm(0.2, 75, 10)
```

```
P(X <= 66.5837876642709) = 0.2
P(X > 66.5837876642709) = 0.8
[1] 66.58379
```



Standard Normal N(0,1)

Because probabilities in a normal distribution depend only on the number of standard deviations above and below the mean, it is useful to define Z-scores (also called standardized scores) as follows:

$$Z\text{-score} = \frac{\text{value} - \text{mean}}{\text{standard deviation}}$$

If we know the population mean and standard deviation, we can plug those in. When we do not, we will use the mean and standard deviation of a random sample as an estimate.

Z-scores provide a second way to compute normal probabilities.

Example 5.5

```
z30 <- (30 - 29.11) / 0.93; z30 # z-score for 30 min
```

Example 5.5

```
[1] 0.9569892
```

```
z31 <- (31 - 29.11) / 0.93; z31 # z-score for 31 min
```

```
[1] 2.032258
```

```
xpnorm(c(30, 31), 29.11, 0.93) # original normal distribution proportion between 30 and 31 min
```

If $X \sim N(29.11, 0.93)$, then

```
P(X <= 30) = P(Z <= 0.957) = 0.8307
P(X <= 31) = P(Z <= 2.032) = 0.9789
P(X > 30) = P(Z > 0.957) = 0.1693
P(X > 31) = P(Z > 2.032) = 0.0211
[1] 0.8307137 0.9789362
```

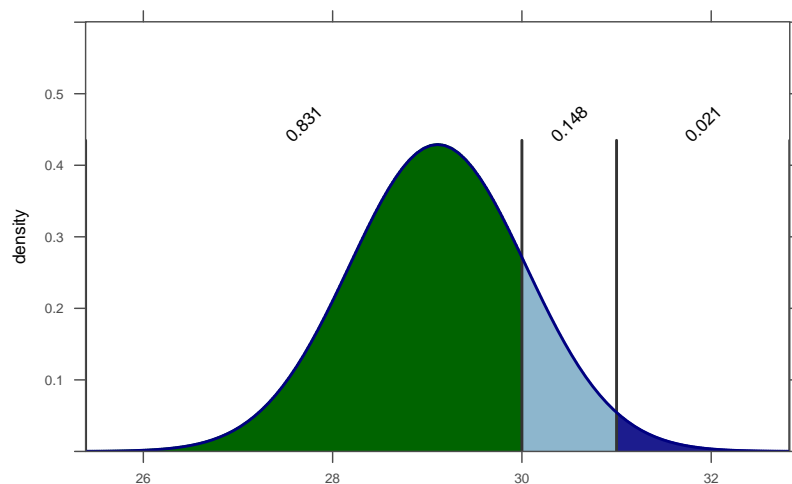
```
xpnorm(c(z30, z31))           # standardized distribution proportion between 30 and 31 min
```

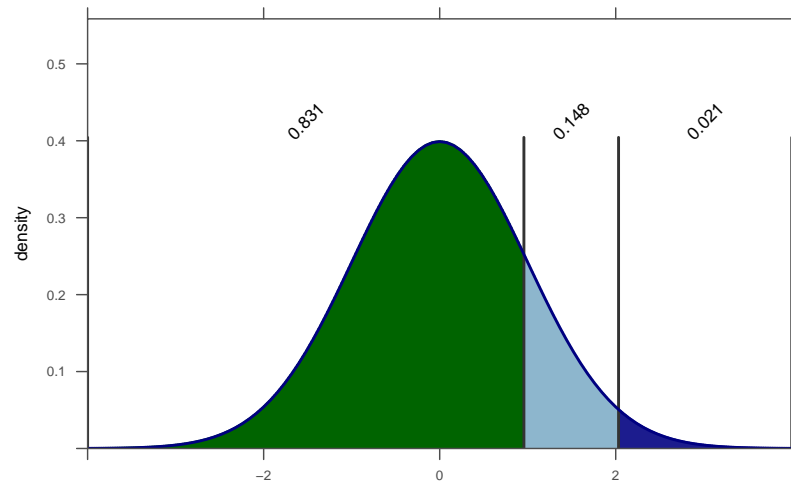
If $X \sim N(0, 1)$, then

```
P(X <= 0.956989247311829) = P(Z <= 0.957) = 0.8307
P(X <= 2.03225806451613) = P(Z <= 2.032) = 0.9789
P(X > 0.956989247311829) = P(Z > 0.957) = 0.1693
P(X > 2.03225806451613) = P(Z > 2.032) = 0.0211
[1] 0.8307137 0.9789362
```

```
pnorm(z31) - pnorm(z30)
```

```
[1] 0.1482226
```





```
xpnorm(0.957) # proportion with z-score below 0.957
```

Example5.5b

If $X \sim N(0,1)$, then

```
P(X <= 0.957) = P(Z <= 0.957) = 0.8307
P(X > 0.957) = P(Z > 0.957) = 0.1693
[1] 0.8307164
```

```
xpnorm(2.032, lower.tail = FALSE) # proportion with z-score above 2.032
```

If $X \sim N(0,1)$, then

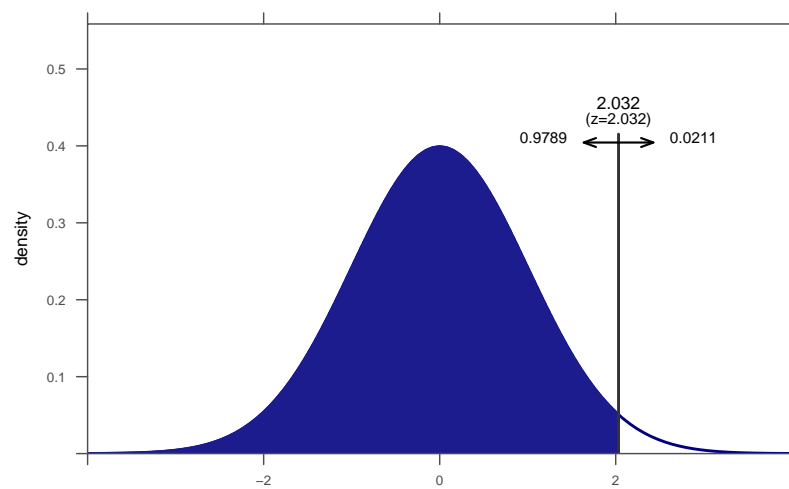
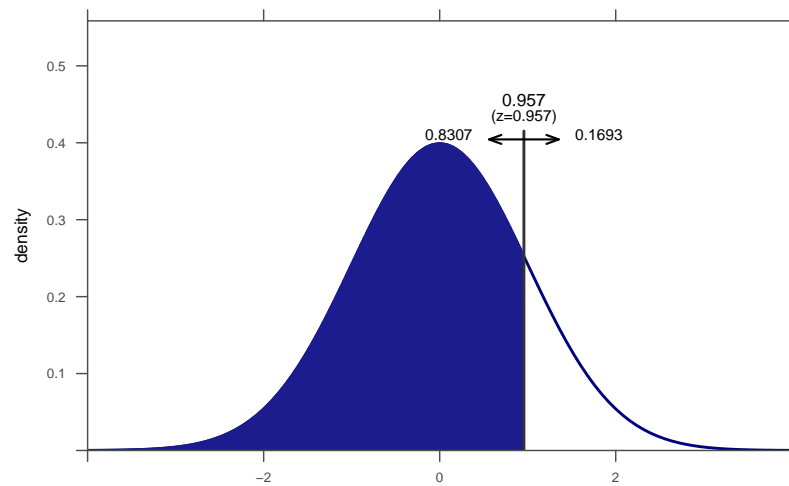
```
P(X <= 2.032) = P(Z <= 2.032) = 0.9789
P(X > 2.032) = P(Z > 2.032) = 0.0211
[1] 0.02107683
```

```
pnorm(30, 29.11, 0.93)
```

```
[1] 0.8307137
```

```
pnorm(31, 29.11, 0.93, lower.tail = FALSE)
```

```
[1] 0.02106377
```



Example 5.6

```
z <- qnorm(0.2)
z
```

```
[1] -0.8416212
```

```
75 + z * 10
```

```
[1] 66.58379
```

Example 5.6

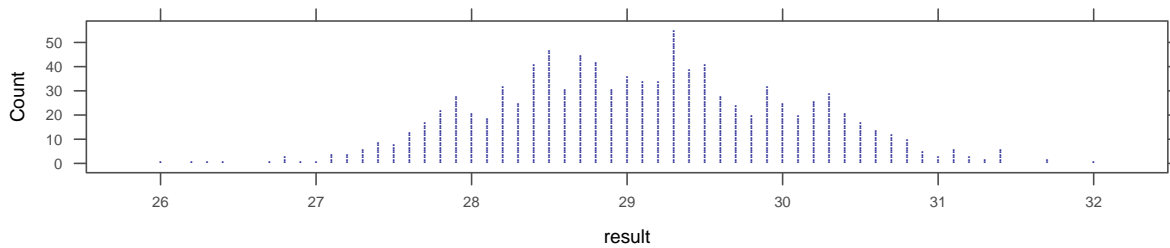
5.2 Confidence Intervals and P-values Using Normal Distributions

Confidence Intervals Based on a Normal Distribution

Example 5.7

```
Bootstrap <- do(1000) * mean(~Time, data = resample(CommuteAtlanta))
dotPlot(~result, width = 0.1, data = Bootstrap)
```

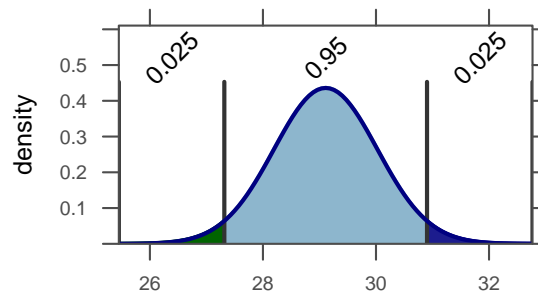
Example5.7



```
xqnorm(c(0.025, 0.975), 29.11, 0.915) # 95% confidence interval for the normal distribution
```

Example5.7b

```
P(X <= 27.3166329541458) = 0.025
P(X <= 30.9033670458542) = 0.975
P(X > 27.3166329541458) = 0.975
P(X > 30.9033670458542) = 0.025
[1] 27.31663 30.90337
```



```
qnorm(0.005, 29.11, 0.915) # lower endpoint for 99% confidence interval
```

```
[1] 26.75312
```

```
qnorm(0.995, 29.11, 0.915) # upper endpoint for 99% confidence interval
```

```
[1] 31.46688
```

Example5.7c

```
qnorm(0.05, 29.11, 0.915) # lower endpoint for 90% confidence interval

[1] 27.60496

qnorm(0.95, 29.11, 0.915) # upper endpoint for 90% confidence interval

[1] 30.61504
```

Example 5.8

```
qnorm(0.005, 13.1, 0.2) # lower endpoint for 99% confidence interval

[1] 12.58483

qnorm(0.995, 13.1, 0.2) # upper endpoint for 99% confidence interval

[1] 13.61517
```

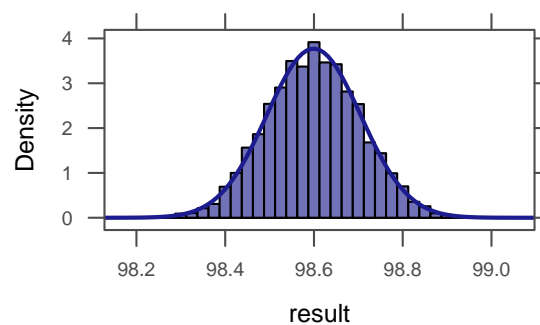
Example5.8

P-values Based on a Normal Distribution

Example 5.9

```
Randomization.Temp <- do(10000) * (mean(~BodyTemp, data = resample(BodyTemp50)) + 0.34)
histogram(~result, width = 0.025, fit = "normal", data = Randomization.Temp)
```

Example5.9



```
pnorm(98.26, 98.6, 0.1066)

[1] 0.0007126119
```

Example5.9b

```
2 * pnorm(98.26, 98.6, 0.1066)
```

```
[1] 0.001425224
```

```
z <- (98.26 - 98.6)/0.1066  
z
```

Example5.9c

```
[1] -3.189493
```

```
pnorm(z)
```

```
[1] 0.0007126119
```

```
2 * pnorm(z)
```

```
[1] 0.001425224
```

Example 5.10

```
pnorm(0.66, 0.65, 0.013, lower.tail = FALSE)
```

Example5.10

```
[1] 0.2208782
```


6

Inference for Means and Proportions

6.1 Distribution of a Sample Proportion

When sampling distributions, bootstrap distributions, and randomization distributions are well approximated by normal distributions, and when we have a way of computing the standard error, we can use normal distributions to compute confidence intervals and p-values using the following general templates:

- confidence interval:

$$\text{statistic} \pm \text{critical value} \cdot SE$$

- hypothesis testing:

$$\text{test statistic} = \frac{\text{statistic} - \text{null parameter}}{SE}$$

Example 6.1

```
SE <- sqrt(0.25 * (1 - 0.25)/50)
SE
```

```
[1] 0.06123724
```

```
SE <- sqrt(0.25 * (1 - 0.25)/200)
SE
```

```
[1] 0.03061862
```

```
SE <- sqrt(0.4 * (1 - 0.4)/50)
SE
```

```
[1] 0.06928203
```

Example6.1

How Large a Sample Size is Needed?

Figure 6.2

```
P.05 <- do(2000) * rflip(50, 0.05)
dotPlot(~prop, width = 0.02, cex = 25, data = P.05)
P.10 <- do(2000) * rflip(50, 0.1)
dotPlot(~prop, width = 0.02, cex = 15, data = P.10)
P.25 <- do(2000) * rflip(50, 0.25)
dotPlot(~prop, width = 0.02, cex = 10, data = P.25)
P.50 <- do(2000) * rflip(50, 0.5)
dotPlot(~prop, width = 0.02, cex = 5, data = P.50)
P.90 <- do(2000) * rflip(50, 0.9)
dotPlot(~prop, width = 0.02, cex = 10, data = P.90)
P.99 <- do(2000) * rflip(50, 0.99)
dotPlot(~prop, width = 0.02, cex = 25, data = P.99)
```

Figure6.02

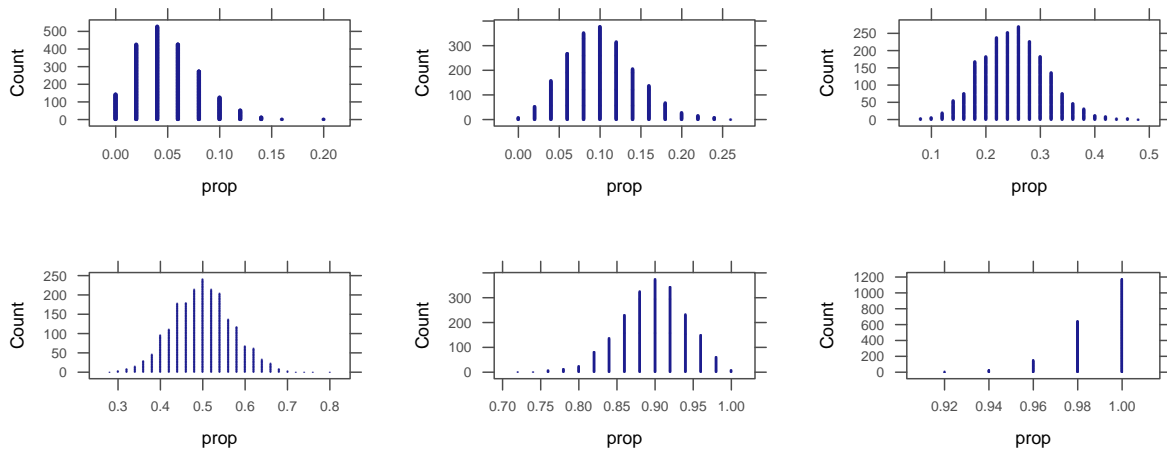
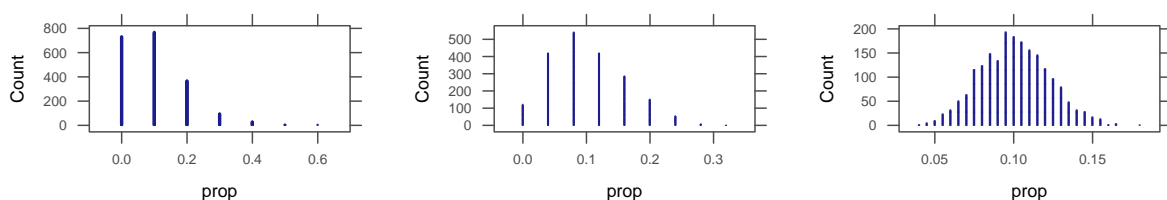


Figure 6.3

```
n10 <- do(2000) * rflip(10, 0.1)
dotPlot(~prop, width = 0.1, cex = 25, data = n10)
n25 <- do(2000) * rflip(25, 0.1)
dotPlot(~prop, width = 0.04, cex = 10, data = n25)
n200 <- do(2000) * rflip(200, 0.1)
dotPlot(~prop, width = 0.005, cex = 5, data = n200)
```

Figure6.03



Example 6.2

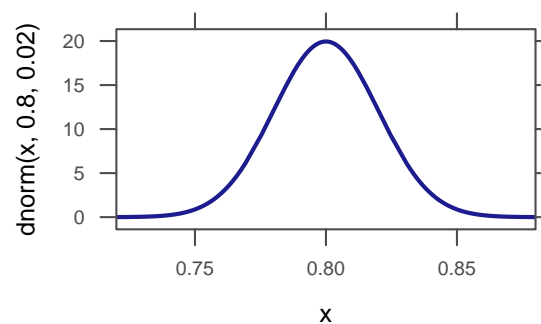
```
p.hat <- 0.80; p.hat  
[1] 0.8  
  
p.hat * 400          # check >= 10  
[1] 320  
  
(1 - p.hat) * 400    # check >= 10  
[1] 80  
  
SE <- sqrt( .80 * .20 / 400 ); SE  
[1] 0.02
```

Example6.2

Figure 6.4

```
plotFun(dnorm(x, 0.8, 0.02) ~ x, x.lim = c(0.72, 0.88))
```

Figure6.4



6.2 Confidence Interval for a Single Proportion

Confidence Interval for a Single Proportion

Example 6.3

```
p.hat <- 52/100; p.hat
```

Example6.3

```
[1] 0.52

SE <- sqrt( p.hat * (1 - p.hat) / 100 ); SE      # est. SE

[1] 0.04995998

p.hat - 1.96 * SE                                # lower end of CI

[1] 0.4220784

p.hat + 1.96 * SE                                # upper end of CI

[1] 0.6179216
```

R can automate finding the confidence interval. Notice the `correct = FALSE` in the second line. The default for the proportion test includes a continuity correction for more accurate results. You can perform the test without the correction for answers closer to the ones in the textbook.

```
confint(prop.test(52, 100))
```

p	lower	upper	level
0.5200000	0.4183183	0.6201278	0.9500000

```
confint(prop.test(52, 100, correct = FALSE))
```

p	lower	upper	level
0.5200000	0.4231658	0.6153545	0.9500000

Example6.3b

Example 6.4

```
p.hat <- 0.28; p.hat

[1] 0.28

SE <- sqrt( p.hat * (1 - p.hat) / 800 ); SE      # est. SE

[1] 0.01587451

p.hat - 1.96 * SE                                # lower end of CI

[1] 0.248886

p.hat + 1.96 * SE                                # upper end of CI
```

Example6.4


```
[1] 0.311114

confint(prop.test(224, 800))           # 224 = 0.28 * 800

      p      lower      upper      level
0.2800000 0.2493936 0.3127512 0.9500000
```

Example6.4b

```
p.hat <- 0.82; p.hat

[1] 0.82

SE <- sqrt( p.hat * (1 - p.hat) / 800 ); SE   # est. SE

[1] 0.01358308

p.hat - 1.96 * SE                        # lower end of CI

[1] 0.7933772

p.hat + 1.96 * SE                        # upper end of CI

[1] 0.8466228

confint(prop.test(656, 800))           # 656 = 0.82 * 800

      p      lower      upper      level
0.8200000 0.7912104 0.8456596 0.9500000
```

Determining Sample Size for Estimating a Proportion

Example 6.5

Example6.5

```
z.star <- qnorm(0.995)
z.star # critical value for 99% confidence

[1] 2.575829

p.hat <- 0.28
p.hat

[1] 0.28

n <- ((z.star/0.01)^2) * p.hat * (1 - p.hat)
n
```

```
[1] 13375.95
```

Example 6.6

```
z.star <- qnorm(0.975)
z.star # critical value for 95% confidence

[1] 1.959964

p.hat <- 0.5
p.hat

[1] 0.5

n <- ((z.star/0.03)^2) * p.hat * (1 - p.hat)
n

[1] 1067.072
```

Example6.6

6.3 Test for a Single Proportion

Example 6.7

1. $H_0: p = 0.20$
 $H_a: p < 0.20$
2. Test statistic: $\hat{p} = 0.19$ (the sample approval rating)
3. Test for a single proportion:

```
p.hat <- 0.19
p.hat

[1] 0.19

p <- 0.2
p

[1] 0.2

p * 1013 # check >= 10

[1] 202.6

(1 - p) * 1013 # check >= 10

[1] 810.4
```

Example6.7

```
SE <- sqrt(p * (1 - p)/1013)
SE

[1] 0.01256768

z <- (p.hat - p)/SE
z

[1] -0.7956915

pnorm(z)

[1] 0.2131057
```

Again, R can automate the test for us.

```
prop.test(192, 1013, alt = "less", p = 0.2) # 192 = 0.19 * 1013
```

1-sample proportions test with continuity correction

data: 192 out of 1013
X-squared = 0.62938, df = 1, p-value = 0.2138
alternative hypothesis: true p is less than 0.2
95 percent confidence interval:
0.0000000 0.2111204
sample estimates:
p
0.189536

Example6.7b

Notice the “less” for the alternative hypothesis because this is a lower tail alternative.

Example 6.8

```
p.hat <- 66/119; p.hat

[1] 0.5546218

p <- 1/3; p

[1] 0.3333333

p * 119 # check >= 10

[1] 39.66667

(1 - p) * 119 # check >= 10

[1] 79.33333
```

Example6.8

```
SE <- sqrt(p * (1 - p) / 119); SE

[1] 0.04321358

z <- (p.hat - p) / SE; z

[1] 5.120809

pnorm(z)                # large side (rounded)

[1] 0.9999998

1 - pnorm(z)            # small side (less rounding)

[1] 1.521135e-07

2 * (1 - pnorm(z))      # p-value = 2 * small side

[1] 3.04227e-07

prop.test(66, 119, p=1/3)

1-sample proportions test with continuity correction

data: 66 out of 119
X-squared = 25.236, df = 1, p-value = 5.072e-07
alternative hypothesis: true p is not equal to 0.3333333
95 percent confidence interval:
 0.4608610 0.6448083
sample estimates:
      p
0.5546218
```

Example 6.9

```
p.hat <- 8/9
p.hat

[1] 0.8888889

p <- 0.5
p

[1] 0.5

p * 9 # check >= 10
```

Example6.9

[1] 4.5

```
Randomization <- do(1000) * rflip(9, 0.5)
head(Randomization, 3)

  n heads tails  prop
1 9     6     3 0.6667
2 9     5     4 0.5556
3 9     6     3 0.6667

prop(~(prop >= p.hat), data = Randomization)

TRUE
0.022
```

Example6.9b

6.4 Distribution of a Sample Mean

Computing the Standard Error

Example 6.10

```
SE <- 32000/sqrt(100)
SE

[1] 3200

SE <- 32000/sqrt(400)
SE

[1] 1600
```

Example6.10

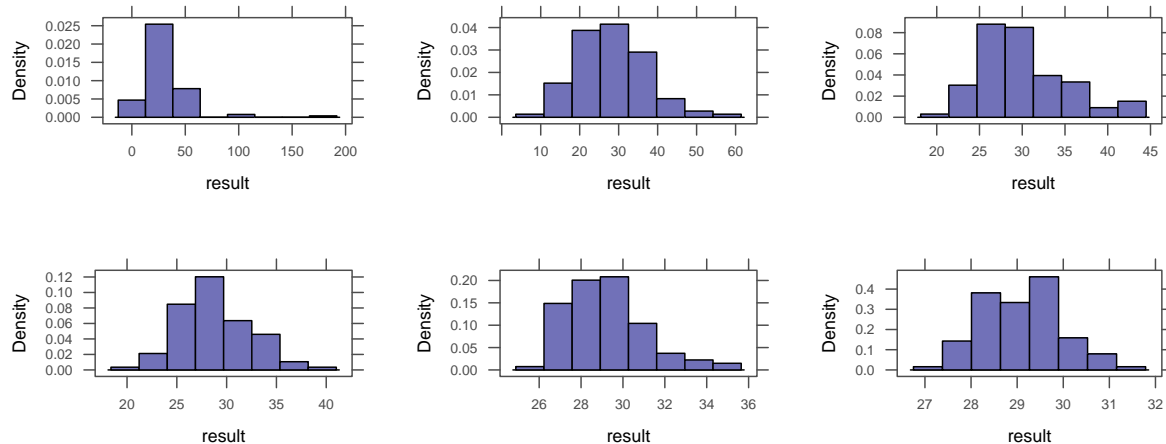
How Large a Sample Size is Needed?

Figure 6.6

```
n1 <- do(100) * mean(~Time, data = resample(CommuteAtlanta, 1))
histogram(~result, data = n1)
n5 <- do(100) * mean(~Time, data = resample(CommuteAtlanta, 5))
histogram(~result, data = n5)
n15 <- do(100) * mean(~Time, data = resample(CommuteAtlanta, 15))
histogram(~result, data = n15)
n30 <- do(100) * mean(~Time, data = resample(CommuteAtlanta, 30))
histogram(~result, data = n30)
```

Figure6.06

```
n125 <- do(100) * mean(~Time, data = resample(CommuteAtlanta, 125))
histogram(~result, data = n125)
n500 <- do(100) * mean(~Time, data = resample(CommuteAtlanta, 500))
histogram(~result, data = n500)
```



The t-Distribution

If we are working with one quantitative variable, we can compute confidence intervals and p-values using the following standard error formula:

$$SE = \frac{\sigma}{\sqrt{n}}$$

Once again, there is a small problem: we won't know σ . So we will estimate σ using our data:

$$SE \approx \frac{s}{\sqrt{n}}$$

Unfortunately, the distribution of

$$\frac{\bar{x} - \mu}{s/\sqrt{n}}$$

does not have a normal distribution. Instead the distribution is a bit “shorter and fatter” than the normal distribution. The correct distribution is called the t-distribution with $n-1$ degrees of freedom. All t-distributions are symmetric and centered at zero. The smaller the degrees of freedom, the shorter and fatter the t-distribution.

Example 6.11

```
df <- 50 - 1
df

[1] 49

SE <- 10.5/sqrt(50)
SE

[1] 1.484924
```

Example6.11

```
df <- 8 - 1
df

[1] 7

SE <- 1.25/sqrt(8)
SE

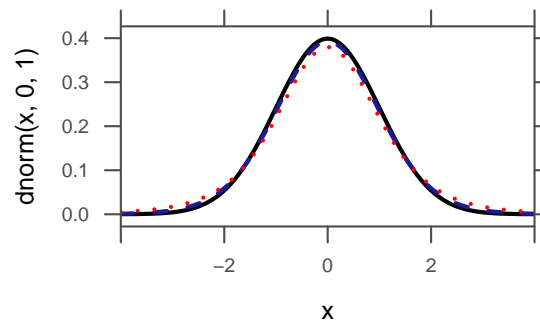
[1] 0.4419417
```

Example6.11b

Figure 6.8

```
plotFun(dnorm(x, 0, 1) ~ x, x.lim = c(-4, 4), col = "black")
plotFun(dt(x, df = 15) ~ x, add = TRUE, lty = 2)
plotFun(dt(x, df = 5) ~ x, add = TRUE, lty = 3, col = "red")
```

Figure6.08



Example 6.12

```
qt(0.975, df = 15)

[1] 2.13145

pt(1.5, df = 15, lower.tail = FALSE)

[1] 0.07718333
```

Example6.12

Similar to the normal distribution, the function for t-distribution is set to find probability of the lower tail.

```
qnorm(0.975)
```

Example6.12b

```
[1] 1.959964
```

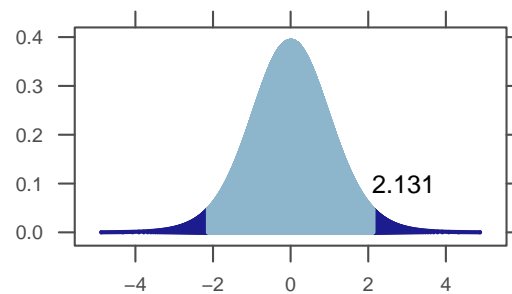
```
pnorm(1.5, lower.tail = FALSE)
```

```
[1] 0.0668072
```

Figure 6.9

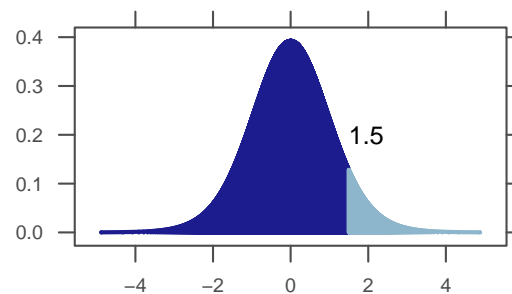
```
plotFun(dt(x, df = 15) ~ x, x.lim = c(-4, 4))
plotDist("t", params = list(df = 15), type = c("h", "l"), groups = (-2.131 < x & x < 2.131),
  lty = 1)
ladd(grid.text("2.131", 2.1, 0.1, default.units = "native", hjust = 0))
```

Figure6.09



```
plotFun(dt(x, df = 15) ~ x, x.lim = c(-4, 4))
plotDist("t", params = list(df = 15), type = c("h", "l"), groups = x > 1.5, lty = 1)
ladd(grid.text("1.5", 1.5, 0.2, default.units = "native", hjust = 0))
```

Figure6.09b



6.5 Confidence Interval for a Mean Using the t-Distribution

Confidence Interval for a Mean Using the t-Distribution

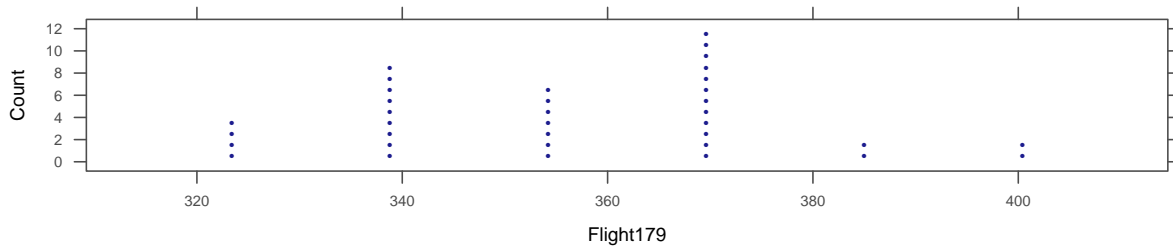
Example 6.13

```
head(Flight179, 3)
```

Example6.13

	Date	Flight179	Flight180	MDY
1	01/05/2010	368	308	2010-01-05
2	01/15/2010	370	292	2010-01-15
3	01/25/2010	354	290	2010-01-25

```
dotPlot(~Flight179, cex = 0.5, data = Flight179) # to check for normality
```



RStudio can do all of the calculations for you if you give it the raw data:

```
favstats(~Flight179, data = Flight179)
```

Example6.13b

min	Q1	median	Q3	max	mean	sd	n	missing
330	341.5	358.5	370.25	407	357.8611	20.178	36	0

```
t.test(~Flight179, data = Flight179)
```

One Sample t-test

```
data: data$Flight179
t = 106.41, df = 35, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 351.0339 364.6884
sample estimates:
mean of x
 357.8611
```

You can also zoom in just the information you want:

```
confint(t.test(~Flight179, data = Flight179))
```

Example6.13c

mean of x	lower	upper	level
357.8611	351.0339	364.6884	0.9500

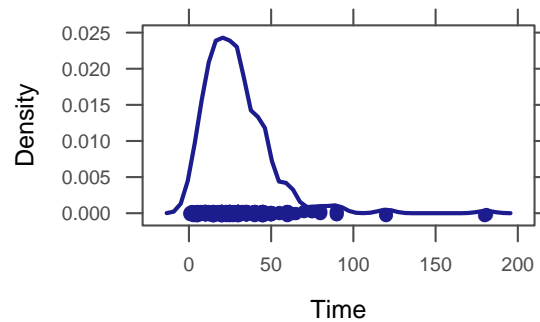
Example 6.14

```
head(CommuteAtlanta, 3)
```

Example6.14

	City	Age	Distance	Time	Sex
1	Atlanta	19	10	15	M
2	Atlanta	55	45	60	M
3	Atlanta	48	12	45	M

```
densityplot(~Time, data = CommuteAtlanta) # to check for normality
```



```
favstats(~Time, data = CommuteAtlanta)
```

Example6.14b

	min	Q1	median	Q3	max	mean	sd	n	missing
1	15	25	40	181	29.11	20.71831	500	0	

```
confint(t.test(~Time, conf.level = 0.99, data = CommuteAtlanta))
```

mean of x	lower	upper	level
29.1100	26.7142	31.5058	0.9900

```
confint(t.test(~Time, conf.level = 0.95, data = CommuteAtlanta))
```

mean of x	lower	upper	level
29.11000	27.28958	30.93042	0.95000

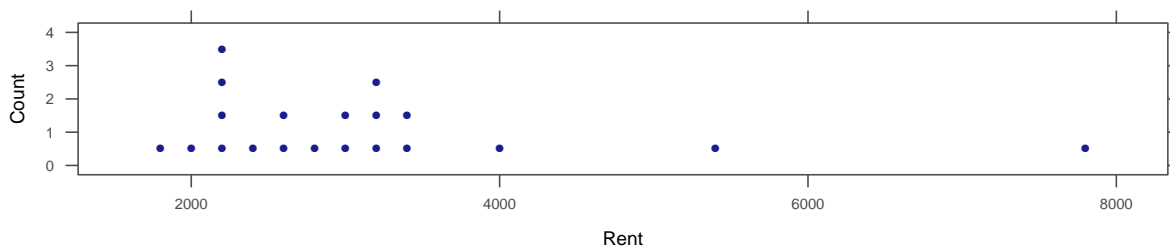
Example 6.15

```
head(ManhattanApartments, 3)
```

Example6.15

```
Rent
1 2275
2 5495
3 2250
```

```
dotPlot(~Rent, width = 200, cex = 0.3, data = ManhattanApartments) # to check for normality
```



```
Boot.Rent <- do(1000) * mean(~Rent, data = resample(ManhattanApartments))
```

Example6.15b

```
head(Boot.Rent, 3)
```

```
result
1 2492
2 2778
3 2833
```

```
favstats(~result, data = Boot.Rent)
```

```
min  Q1 median  Q3  max mean   sd   n missing
2378 2937   3136 3361 4216 3162 303.8 1000      0
```

```
cdata(0.95, result, data = Boot.Rent)
```

```
low      hi central.p
2642.23  3835.90      0.95
```

Determining Sample Size for Estimating a Mean

Example 6.16

```
n <- (1.96 * 20.18/2)^2
n
```

Example6.16

```
[1] 391.106
```

6.6 Test for a Single Mean

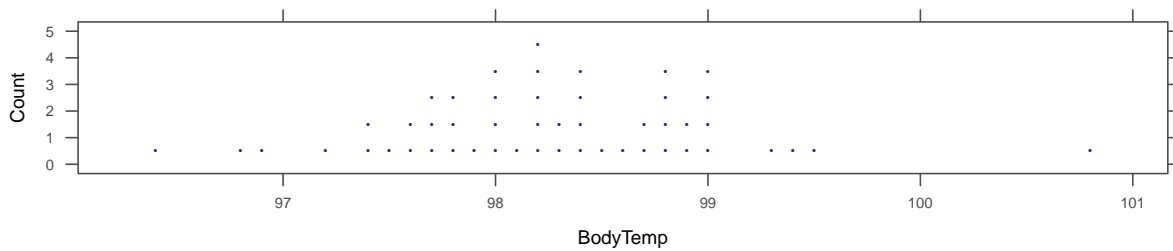
Example 6.17

```
head(BodyTemp50)
```

Example6.17

	BodyTemp	Pulse	Gender	Sex
1	97.6	69	0	Female
2	99.4	77	1	Male
3	99.0	75	0	Female
4	98.8	84	1	Male
5	98.0	71	0	Female
6	98.9	76	1	Male

```
dotPlot(~BodyTemp, cex = 0.15, width = 0.1, data = BodyTemp50) # to check for normality
```



```
favstats(~BodyTemp, data = BodyTemp50)
```

Example6.17b

min	Q1	median	Q3	max	mean	sd	n	missing
96.4	97.8	98.2	98.8	100.8	98.26	0.7653197	50	0

```
t.test(~BodyTemp, mu = 98.6, data = BodyTemp50)
```

One Sample t-test

```
data: data$BodyTemp
t = -3.1414, df = 49, p-value = 0.002851
alternative hypothesis: true mean is not equal to 98.6
95 percent confidence interval:
 98.0425 98.4775
sample estimates:
mean of x
 98.26
```

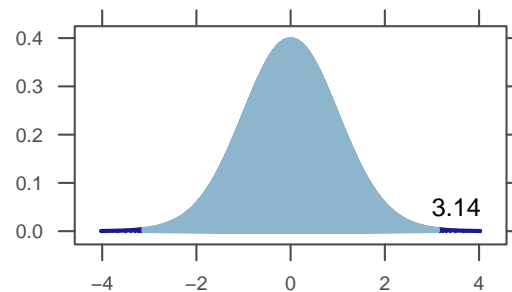
```
pval(t.test(~BodyTemp, mu = 98.6, data = BodyTemp50)) # to find the p-value directly

p.value
0.002850509
```

Figure 6.17

```
plotFun(dt(x, df = 49) ~ x, x.lim = c(-4, 4))
plotDist("t", params = list(df = 49), type = c("h", "l"), groups = (-3.14 < x & x < 3.14),
        lty = 1)
ladd(grid.text("3.14", 3, 0.05, default.units = "native", hjust = 0))
```

Figure6.17



Example 6.18

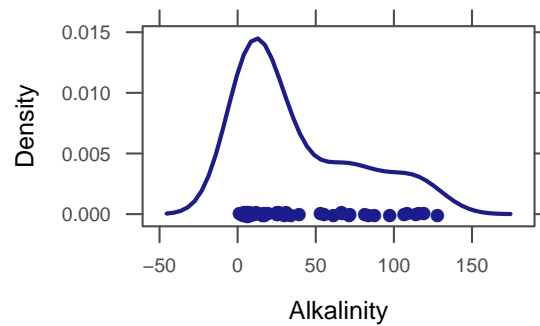
```
head(FloridaLakes, 3)
```

Example6.18

ID	Lake	Alkalinity	pH	Calcium	Chlorophyll	AvgMercury	NumSamples	MinMercury
1	1 Alligator	5.9	6.1	3.0	0.7	1.23	5	0.85
2	2 Annie	3.5	5.1	1.9	3.2	1.33	7	0.92
3	3 Apopka	116.0	9.1	44.1	128.3	0.04	6	0.04

	MaxMercury	ThreeYrStdMercury	AgeData
1	1.43	1.53	1
2	1.90	1.33	0
3	0.06	0.04	0

```
densityplot(~Alkalinity, data = FloridaLakes) # to check for normality
```



Example6.18b

```
favstats(~Alkalinity, data = FloridaLakes)
```

```
min  Q1 median  Q3 max    mean      sd  n missing
1.2  6.6   19.6 66.5 128 37.53019 38.20353 53      0
```

```
t.test(~Alkalinity, alt = "greater", mu = 35, data = FloridaLakes)
```

One Sample t-test

```
data: data$Alkalinity
t = 0.48216, df = 52, p-value = 0.3159
alternative hypothesis: true mean is greater than 35
95 percent confidence interval:
 28.74199      Inf
sample estimates:
mean of x
 37.53019
```

Notice the “greater” for the alternative hypothesis.

6.7 Distribution of Differences in Proportions

Example 6.19

```
OneTrueLove <- read.file("OneTrueLove.csv")
```

Example6.19

Reading data with readr::read_csv()

```
head(OneTrueLove)
```

```
  Gender Response
1  Male   Agree
2  Male   Agree
3  Male   Agree
```

```
4 Male Agree
5 Male Agree
6 Male Agree
```

```
tally(Response ~ Gender, format = "count", margins = TRUE, data = OneTrueLove)
```

	Gender	
Response	Female	Male
Agree	363	372
Disagree	1005	807
Don't know	44	34
Total	1412	1213

```
prop(Response ~ Gender, data = OneTrueLove)
```

```
target level: Agree; other levels: Disagree, Don't know
```

Agree.Female	Agree.Male
0.2570822	0.3066777

```
diff(prop(Response ~ Gender, data = OneTrueLove))
```

```
target level: Agree; other levels: Disagree, Don't know
```

Agree.Male
0.04959551

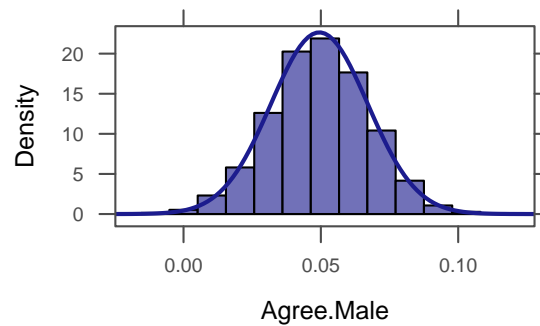
Figure 6.20

```
Boot.Love <- do(5000) * diff(prop(Response ~ Gender, data = resample(OneTrueLove)))
head(Boot.Love, 3)
```

Figure6.20

	Agree.Male
1	0.04922
2	0.02343
3	0.04922

```
histogram(~Agree.Male, fit = "normal", data = Boot.Love)
```



Example 6.20

```
SE <- sqrt(0.257 * (1 - 0.257)/1412 + 0.307 * (1 - 0.307)/1213)
SE
```

Example6.20

```
[1] 0.01762461
```

6.8 Confidence Interval for a Difference in Proportions

Data 6.3

```
success <- c(158, 109)
n <- c(444, 922)
```

Data6.3

Example 6.21

```
success <- c(158, 109)
n <- c(444, 922)
prop.test(success, n, conf.level = 0.9)
```

Example6.21

2-sample test for equality of proportions with continuity correction

```
data: success out of n
X-squared = 106.11, df = 1, p-value < 2.2e-16
alternative hypothesis: two.sided
90 percent confidence interval:
 0.1947026 0.2805666
sample estimates:
 prop 1    prop 2 
0.3558559 0.1182213
```


6.9 Test For a Difference in Proportions

Data 6.4

```
SplitSteal <- rbind(
  do(187) * data.frame( agegroup = "Under40", decision = "Split"),
  do(195) * data.frame( agegroup = "Under40", decision = "Steal"),
  do(116) * data.frame( agegroup = "Over40", decision = "Split"),
  do(76) * data.frame( agegroup = "Over40", decision = "Steal")
)
```

Data6.4

Example 6.22

```
prop(decision ~ agegroup, data = SplitSteal) # sample prop within each group
```

Example6.22

```
target level: Split; other levels: Steal
```

```
Split.Under40 Split.Over40
0.4895288      0.6041667
```

```
prop(~decision, data = SplitSteal) # pooled proportion
```

```
target level: Split; other levels: Steal
```

```
Split
0.5278746
```

Example 6.23

```
diff <- diff(prop(decision ~ agegroup, data = SplitSteal))
```

Example6.23

```
target level: Split; other levels: Steal
```

```
diff
```

```
Split.Over40
0.1146379
```

```
prop.test(decision ~ agegroup, data = SplitSteal)
```

```
2-sample test for equality of proportions with continuity correction
```

```
data: tally(decision ~ agegroup)
```

```

X-squared = 6.2857, df = 1, p-value = 0.01217
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.20397777 -0.02529797
sample estimates:
 prop 1      prop 2
0.4895288 0.6041667

```

6.10 Distribution of Differences in Means

Figure 6.21

```

BootE <- do(2000) * diff(mean(Exercise ~ Gender, data = resample(ExerciseHours)))
head(BootE, 3)

```

Figure6.21

```

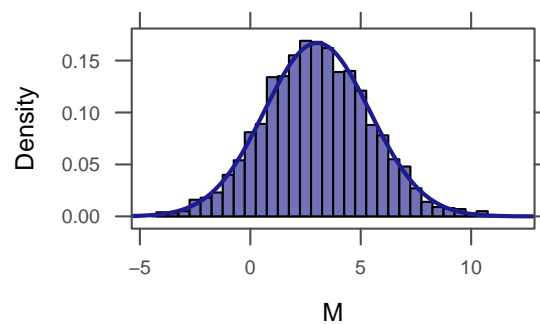
      M
1 4.3167
2 3.7793
3 0.7435

```

```

histogram(~M, width = 0.5, fit = "normal", data = BootE)

```



```

Random.Smiles <- do(2000) * diff(mean(Leniency ~ shuffle(Group), data = Smiles))
head(Random.Smiles, 3)

```

Figure6.21b

```

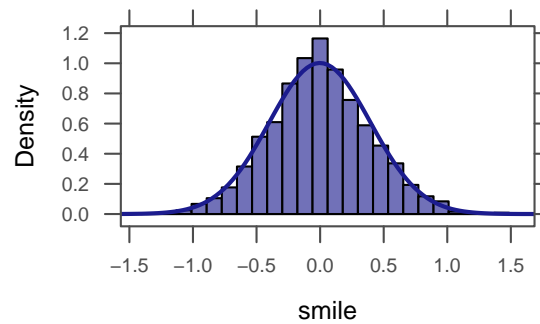
      smile
1 -0.08824
2  0.11765
3  0.44118

```

```

histogram(~smile, n = 24, , fit = "normal", data = Random.Smiles)

```



The t-Distribution

Example 6.24

```
favstats(Exercise ~ Gender, data = ExerciseHours)
```

Example6.24

	Gender	min	Q1	median	Q3	max	mean	sd	n	missing
1	F	0	3	10	12.00	34	9.4	7.407359	30	0
2	M	2	3	12	19.25	30	12.4	8.798325	20	0

```
SE <- sqrt(8.8^2/20 + 7.41^2/30)
SE
```

```
[1] 2.387943
```

```
favstats(Leniency ~ Group, data = Smiles)
```

	Group	min	Q1	median	Q3	max	mean	sd	n	missing
1	neutral	2.0	3.0	4.00	4.875	8	4.117647	1.522850	34	0
2	smile	2.5	3.5	4.75	5.875	9	4.911765	1.680866	34	0

```
SE <- sqrt(1.68^2/34 + 1.52^2/34)
SE
```

```
[1] 0.3885418
```

6.11 Confidence Interval for a Difference in Means

Example 6.26

```
head(CommuteStLouis)
```

Example6.26

	City	Age	Distance	Time	Sex
1	St. Louis	52	10	20	M
2	St. Louis	21	35	40	F
3	St. Louis	23	40	45	F
4	St. Louis	38	0	2	M
5	St. Louis	26	15	25	M
6	St. Louis	46	7	12	M

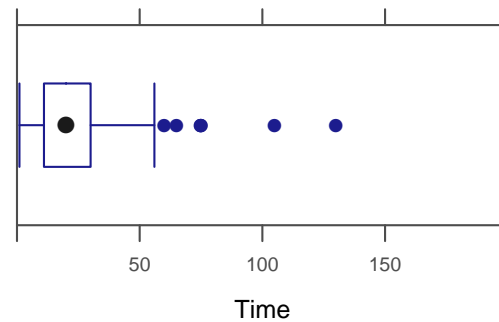
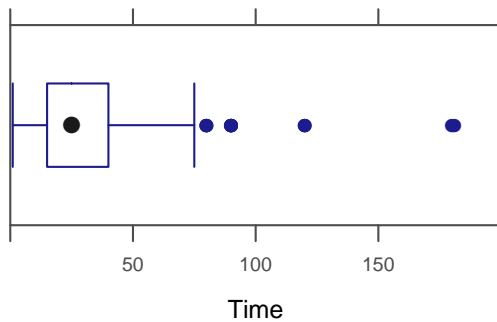
```
favstats(~Time, data = CommuteStLouis)
```

	min	Q1	median	Q3	max	mean	sd	n	missing
1	11.5	20	30	130	21.97	14.23244	500	0	

```
favstats(~Time, data = CommuteAtlanta)
```

	min	Q1	median	Q3	max	mean	sd	n	missing
1	15	25	40	181	29.11	20.71831	500	0	

```
bwplot(~Time, xlim = c(0, 200), data = CommuteAtlanta) # to check for normality
bwplot(~Time, xlim = c(0, 200), data = CommuteStLouis) # to check for normality
```



```
confint(t.test(CommuteAtlanta$Time, CommuteStLouis$Time, conf.level = 0.9))
```

Example6.26b

	mean of x	mean of y	lower	upper	level
	29.110000	21.970000	5.289066	8.990934	0.900000

6.12 Test for a Difference in Means

Example 6.27

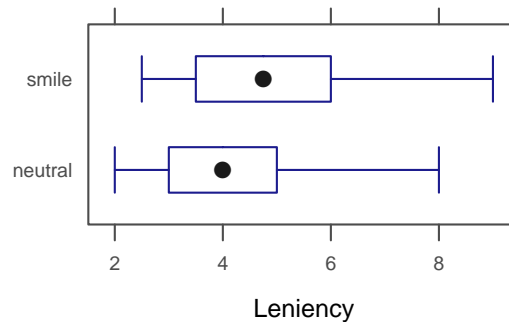
```
head(Smiles, 3)
```

Example6.27

Leniency Group

```
1      7 smile
2      3 smile
3      6 smile
```

```
bwplot(Group ~ Leniency, data = Smiles) # to check for normality
```



```
t.test(Leniency ~ Group, alt = "less", data = Smiles)
```

Example6.27b

Welch Two Sample t-test

```
data: Leniency by Group
t = -2.0415, df = 65.367, p-value = 0.02262
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
 -Inf -0.1451043
sample estimates:
mean in group neutral    mean in group smile
      4.117647           4.911765
```

6.13 Paired Difference in Means

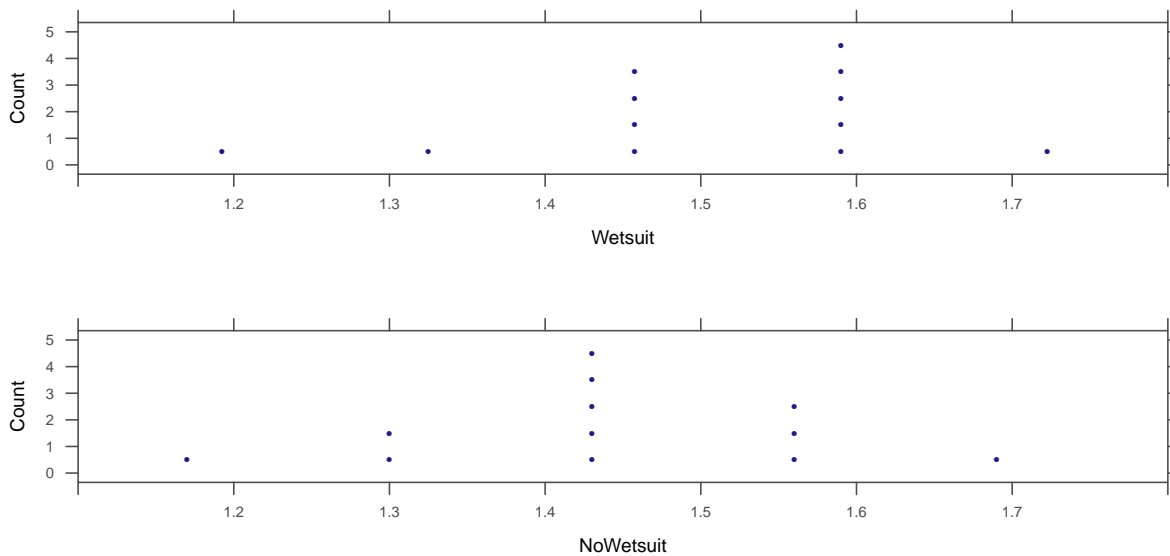
Example 6.28

```
head(Wetsuits, 3)
```

Example6.28

	Wetsuit	NoWetsuit	Gender	Type	Sex
1	1.57	1.49	F	swimmer	Female
2	1.47	1.37	F	triathlete	Female
3	1.42	1.35	F	swimmer	Female

```
dotPlot(~Wetsuit, xlim = c(1.1, 1.8), cex = 0.25, data = Wetsuits) # to check for normality
dotPlot(~NoWetsuit, xlim = c(1.1, 1.8), cex = 0.25, data = Wetsuits) # to check for normality
```



```
t.test(Wetsuits$Wetsuit, Wetsuits$NoWetsuit)
```

Example6.28b

Welch Two Sample t-test

```
data: Wetsuits$Wetsuit and Wetsuits$NoWetsuit
t = 1.3688, df = 21.974, p-value = 0.1849
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.03992937  0.19492937
sample estimates:
mean of x mean of y
 1.506667  1.429167
```

Example 6.29

```
head(Wetsuits, 3)
```

Example6.29

	Wetsuit	NoWetsuit	Gender	Type	Sex
1	1.57	1.49	F	swimmer	Female
2	1.47	1.37	F	triathlete	Female
3	1.42	1.35	F	swimmer	Female

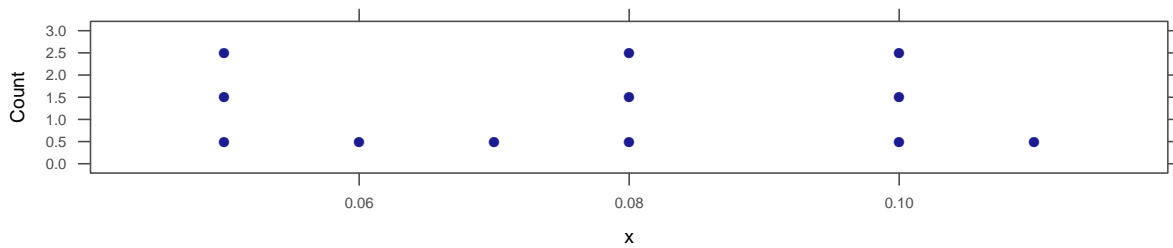
```
t.test(Wetsuits$Wetsuit, Wetsuits$NoWetsuit, paired = TRUE)
```

Paired t-test

```
data: Wetsuits$Wetsuit and Wetsuits$NoWetsuit
t = 12.318, df = 11, p-value = 8.885e-08
```

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.06365244 0.09134756
sample estimates:
mean of the differences
      0.0775
```

```
dotPlot(Wetsuits$Wetsuit - Wetsuits$NoWetsuit, width = 0.01, cex = 0.3)
```



Example 6.30

```
confint(t.test(Wetsuits$Wetsuit, Wetsuits$NoWetsuit, paired = TRUE))
```

Example6.30

```
mean of the differences      lower      upper
      0.07750000      0.06365244      0.09134756
level
      0.95000000
```

```
confint(t.test(~(Wetsuit - NoWetsuit), data = Wetsuits))
```

```
mean of x      lower      upper      level
0.07750000 0.06365244 0.09134756 0.95000000
```


7

Chi-Squared Tests for Categorical Variables

Goodness of fit tests test how well a distribution fits some hypothesis.

7.1 Testing Goodness-of-Fit for a Single Categorical Variable

Example 7.1

```
tally(~Answer, format = "proportion", data = APMultipleChoice)
```

Example 7.1

A	B	C	D	E
0.2125	0.2250	0.1975	0.1950	0.1700

Chi-square Statistic

The **Chi-squared test statistic**:

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

There is one term in this sum *for each cell in our data table*, and

- observed = the tally in that cell (a count from our raw data)
- expected = the number we would “expect” if the percentages followed our null hypothesis exactly. (Note: the expected counts might not be whole numbers.)

Example 7.5

You could calculate the chi-square statistic manually but of course, R can automate this whole process for us if we provide the data table and the null hypothesis. Notice that to use `chisq.test()`, you must enter the data

like `answer <- c(85, 90, 79, 78, 68)`. The default null hypothesis is that all the probabilities are equal.

```
head(APMultipleChoice)
```

Example 7.5

```
Answer
1      B
2      B
3      D
4      A
5      E
6      D
```

```
answer <- c(85, 90, 79, 78, 68)
chisq.test(answer)
```

Chi-squared test for given probabilities

```
data:  answer
X-squared = 3.425, df = 4, p-value = 0.4894
```

Chi-square Distribution

Figure 7.2

```
chisq.sample <- do(1000) * chisq.test(tally(~resample(toupper(letters[1:5]), 400)))$statistic
histogram(~X.squared, data = chisq.sample)
```

Figure 7.02

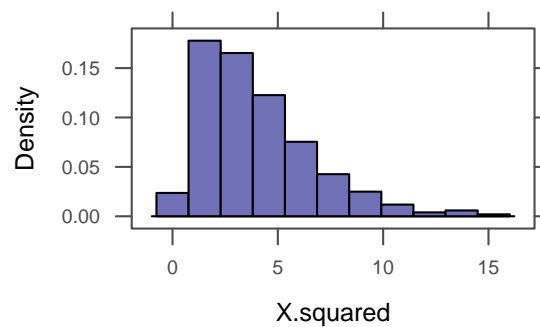
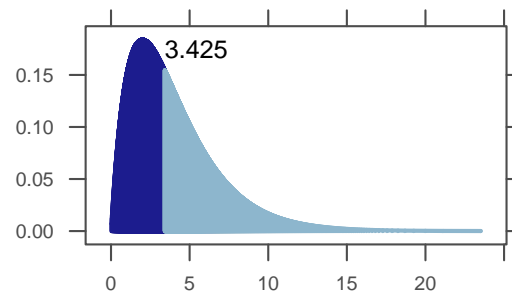


Figure 7.3

```
plotDist("chisq", params = list(df = 4), type = c("h", "l"), groups = x > 3.425, lty = 1)
ladd(grid.text("3.425", 3.425, 0.175, default.units = "native", hjust = 0))
```

Figure 7.03



Our test statistic will be large when the observed counts and expected counts are quite different. It will be small when the observed counts and expected counts are quite close. So we will reject when the test statistic is large. To know how large is large enough, we need to know the sampling distribution.

If H_0 is true and the sample is large enough, then the sampling distribution for the Chi-squared test statistic will be approximately a Chi-squared distribution.

- The **degrees of freedom** for this type of goodness of fit test is one less than the number of cells.
- The approximation gets better and better as the sample size gets larger.

The mean of a Chi-squared distribution is equal to its degrees of freedom. This can help us get a rough idea about whether our test statistic is unusually large or not.

Example 7.6

1. $H_0: p_w = 0.54, p_b = 0.18, p_h = 0.12, p_a = 0.15, p_o = 0.01$;
 H_a : At least one p_i is not as specified.
2. Observed count: $w = 780, b = 117, h = 114, a = 384, o = 58$
3. Chi-squared test:

```
jury <- c(780, 117, 114, 384, 58)
chisq.test(jury, p = c(0.54, 0.18, 0.12, 0.15, 0.01))
```

Example 7.6

Chi-squared test for given probabilities

```
data: jury
X-squared = 357.36, df = 4, p-value < 2.2e-16
```

```
xchisq.test(jury, p = c(0.54, 0.18, 0.12, 0.15, 0.01)) # to list expected counts
```

Chi-squared test for given probabilities

```
data: x
```

```
X-squared = 357.36, df = 4, p-value < 2.2e-16

 780.00  117.00  114.00  384.00   58.00
(784.62) (261.54) (174.36) (217.95) ( 14.53)
[  0.027] [ 79.880] [ 20.895] [126.509] [130.051]
<-0.16>  <-8.94>  <-4.57>  <11.25>  <11.40>

key:
observed
(expected)
[contribution to X-squared]
<residual>
```

Notice in this example, we need to tell R what the null hypothesis is.

How unusual is it to get a test statistic at least as large as ours? We compare to a Chi-squared distribution with 4 degrees of freedom. The mean value of such a statistic is 4, and our test statistic is much larger, so we anticipate that our value is extremely unusual.

Goodness-of-Fit for Two Categories

When there are only two categories, the Chi-squared goodness of fit test is equivalent to the 1-proportion test. Notice that `prop.test()` uses the count in one category and total but that `chisq.test()` uses cell counts.

Example 7.8

```
prop.test(84, 200)
```

Example 7.8

1-sample proportions test with continuity correction

```
data: 84 out of 200
X-squared = 4.805, df = 1, p-value = 0.02838
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3513385 0.4917858
sample estimates:
 p
0.42
```

```
chisq.test(c(84, 116), p = c(0.5, 0.5))
```

Chi-squared test for given probabilities

```
data: c(84, 116)
X-squared = 5.12, df = 1, p-value = 0.02365
```

```
binom.test(84, 200)
```

Exact binomial test (with Score CI)

```
data: 84 out of 200
number of successes = 84, number of trials = 200, p-value = 0.02813
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.3507439 0.4916638
sample estimates:
probability of success
          0.42
```

Although all three tests test the same hypotheses and give similar p-values (in this example), the binomial test is generally used because

- The binomial test is exact for all sample sizes while the Chi-squared test and 1-proportion test are only approximate, and the approximation is poor when sample sizes are small.
- The binomial test and 1-proportion test also produce confidence intervals.

7.2 Testing for an Association Between Two Categorical Variables

Example 7.9

```
OneTrueLove <- read.file("OneTrueLove.csv")
```

Example 7.9

Reading data with readr::read_csv()

```
tally(~Response, format = "proportion", data = OneTrueLove)
```

```
      Agree  Disagree Don't know
0.28000000 0.69028571 0.02971429
```

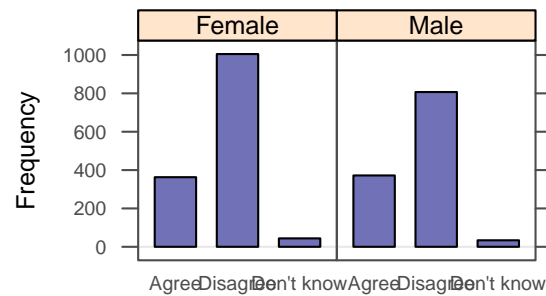
```
tally(~Response + Gender, format = "proportion", margin = TRUE, data = OneTrueLove)
```

	Gender		
Response	Female	Male	Total
Agree	0.13828571	0.14171429	0.28000000
Disagree	0.38285714	0.30742857	0.69028571
Don't know	0.01676190	0.01295238	0.02971429
Total	0.53790476	0.46209524	1.00000000

Figure 7.4

```
bargraph(~Response | Gender, type = "count", data = OneTrueLove)
```

Figure 7.04



Chi-square Test for Association

Example 7.10

```
head(WaterTaste, 3)
```

Example7.10

```

  Gender Age Class UsuallyDrink FavBotWatBrand Preference First Second Third
1      F  18     F    Filtered    DEER PARK    CABD  Fiji SamsChoice Aquafina
2      F  18     F      Tap      NONE    CABD  Fiji SamsChoice Aquafina
3      F  18     F      Tap    DEER PARK    CADB  Fiji SamsChoice    Tap
  Fourth Sex
1    Tap Female
2    Tap Female
3 Aquafina Female

```

```
water <- tally(~UsuallyDrink + First, data = WaterTaste)
water
```

```

      First
UsuallyDrink Aquafina Fiji SamsChoice Tap
  Bottled      14   15      8   4
  Filtered      4   10      9   3
  Tap           7   16      7   3

```

```
water <- rbind(c(14, 15, 8, 4), c(11, 26, 16, 6)) # to combine Tap and Filtered
water
```

Example7.10b

```

      [,1] [,2] [,3] [,4]
[1,]   14   15    8    4
[2,]   11   26   16    6

```

```

colnames(water) <- c("Aquafina", "Fiji", "SamsChoice", "Tap") # add column names
rownames(water) <- c("Bottled", "Tap/Filtered") # add row names
water

```

```

      Aquafina Fiji SamsChoice Tap

```

Bottled	14	15	8	4
Tap/Filtered	11	26	16	6

```
xchisq.test(water)
```

Example7.10c

Pearson's Chi-squared test

```
data: x
X-squared = 3.243, df = 3, p-value = 0.3557
```

```
 14.00   15.00    8.00    4.00
(10.25) (16.81) ( 9.84) ( 4.10)
[1.3720] [0.1949] [0.3441] [0.0024]
< 1.171> <-0.441> <-0.587> <-0.049>
```

```
 11.00   26.00   16.00    6.00
(14.75) (24.19) (14.16) ( 5.90)
[0.9534] [0.1354] [0.2391] [0.0017]
<-0.976> < 0.368> < 0.489> < 0.041>
```

```
key:
observed
(expected)
[contribution to X-squared]
<residual>
```

Special Case for a 2 x 2 Table

There is also an exact test that works only in the case of a 2×2 table (much like the binomial test can be used instead of a goodness of fit test if there are only two categories). The test is called **Fisher's Exact Test**.

In this case we see that the simulated p-value from the Chi-squared Test is nearly the same as the exact p-value from Fisher's Exact Test. This is because Fisher's test is using mathematical formulas to compute probabilities of *all* randomizations – it is essentially the same as doing infinitely many randomizations!

Note: For a 2×2 table, we could also use the method of 2-proportions (`prop.test()`, manual resampling, or formula-based). The approximations based on the normal distribution will be poor in the same situations where the Chi-squared test gives a poor approximation.

Example 7.11

```
SplitStealTable <- rbind(c(187, 195), c(116, 76))
SplitStealTable
```

Example7.11

```
      [,1] [,2]
[1,]  187  195
[2,]  116   76
```

```
colnames(SplitStealTable) <- c("Split", "Steal")
```

```
rownames(SplitStealTable) <- c("Younger", "Older")
SplitStealTable
```

	Split	Steal
Younger	187	195
Older	116	76

Example7.11b

```
fisher.test(SplitStealTable)
```

Fisher's Exact Test for Count Data

```
data: SplitStealTable
p-value = 0.01023
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.4346122 0.9065598
sample estimates:
odds ratio
 0.6288154
```

```
xchisq.test(SplitStealTable)
```

Pearson's Chi-squared test with Yates' continuity correction

```
data: x
X-squared = 6.2857, df = 1, p-value = 0.01217
```

```
 187.00   195.00
(201.65) (180.35)
 [0.99]  [1.11]
<-1.03>  < 1.09>
```

```
 116.00    76.00
(101.35) ( 90.65)
 [1.97]  [2.21]
< 1.46>  <-1.54>
```

```
key:
observed
(expected)
[contribution to X-squared]
<residual>
```

To use the test for proportions as done in Example 6.23,

Example7.11c

```
SplitStealData <- rbind(
  do(187) * data.frame( agegroup = "Under40", decision="Split"),
  do(195) * data.frame( agegroup = "Under40", decision="Steal"),
  do(116) * data.frame( agegroup = "Over40",  decision="Split"),
  do(76)  * data.frame( agegroup = "Over40",  decision="Steal")
)
```



```
prop.test(decision ~ agegroup, data = SplitStealData)
```

[Example7.11d](#)

2-sample test for equality of proportions with continuity correction

```
data:  tally(decision ~ agegroup)
X-squared = 6.2857, df = 1, p-value = 0.01217
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.20397777 -0.02529797
sample estimates:
  prop 1    prop 2 
0.4895288 0.6041667
```


8

ANOVA to Compare Means

8.1 Analysis of Variance

- Two variables: categorical explanatory and quantitative response
 - Can be used in either experimental or observational designs.
- Main Question: Does the population mean response depend on the (treatment) group?
 - H_0 : the population group means are all the equal ($\mu_1 = \mu_2 = \dots \mu_k$)
 - H_a : the population group means are not all equal
- If categorical variable has only 2 values, we already have a method: 2-sample t -test
 - ANOVA allows for 3 or more groups (sub-populations)
- F statistic compares within group variation (how different are individuals in the same group?) to between group variation (how different are the different group means?)
- ANOVA assumes that each group is normally distributed with the same (population) standard deviation.
 - Check normality with normal quantile plots (of residuals)
 - Check equal standard deviation using 2:1 ratio rule (largest standard deviation at most twice the smallest standard deviation).

Null and Alternative Hypotheses

Example 8.1

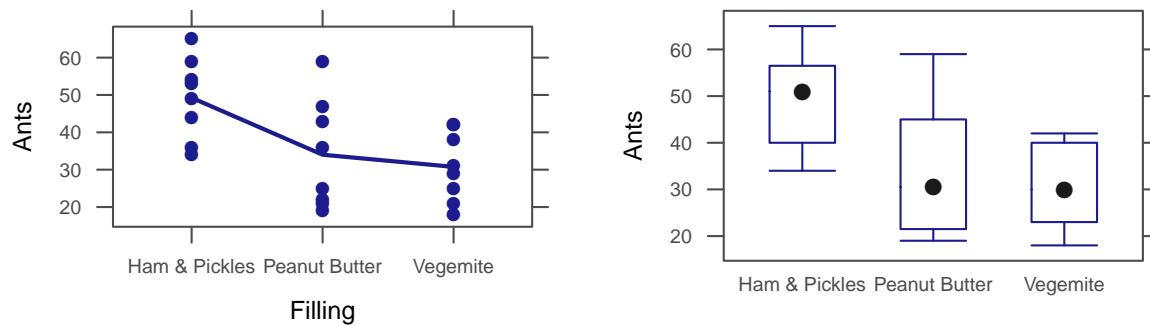
```
favstats(Ants ~ Filling, data = SandwichAnts)
```

Example 8.1

	Filling	min	Q1	median	Q3	max	mean	sd	n	missing
1	Ham & Pickles	34	42.00	51.0	55.25	65	49.25	10.793517	8	0
2	Peanut Butter	19	21.75	30.5	44.00	59	34.00	14.628739	8	0
3	Vegemite	18	24.00	30.0	39.00	42	30.75	9.254343	8	0

```
xyplot(Ants ~ Filling, SandwichAnts, type = c("p", "a"))
bwplot(Ants ~ Filling, SandwichAnts)
```

Example8.1b



Partitioning Variability

Example 8.3

```
Ants.Model <- lm(Ants ~ Filling, data = SandwichAnts)
anova(Ants.Model)
```

Example8.3

Analysis of Variance Table

Response: Ants

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Filling	2	1561	780.50	5.6267	0.01105 *
Residuals	21	2913	138.71		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

The p-value listed in this output is the p-value for our null hypothesis that the mean population response is the same in each treatment group. In this case we would reject the null hypothesis at the $\alpha = 0.05$ level.

In the next section we'll look at this test in more detail, but notice that if you know the assumptions of a test, the null hypothesis being tested, and the p-value, you can generally interpret the results even if you don't know all the details of how the test statistic is computed.

The F-Statistic

The ANOVA test statistic (called F) is based on three ingredients:

1. how different the group means are (between group differences)
2. the amount of variability within each group (within group differences)
3. sample size

Each of these will be involved in the calculation of F .

Figure 8.3

```
Rand.Ants <- do(1000) * anova(lm(Ants ~ shuffle(Filling), data = SandwichAnts))
tally(~(F >= 5.63), data = Rand.Ants)
```

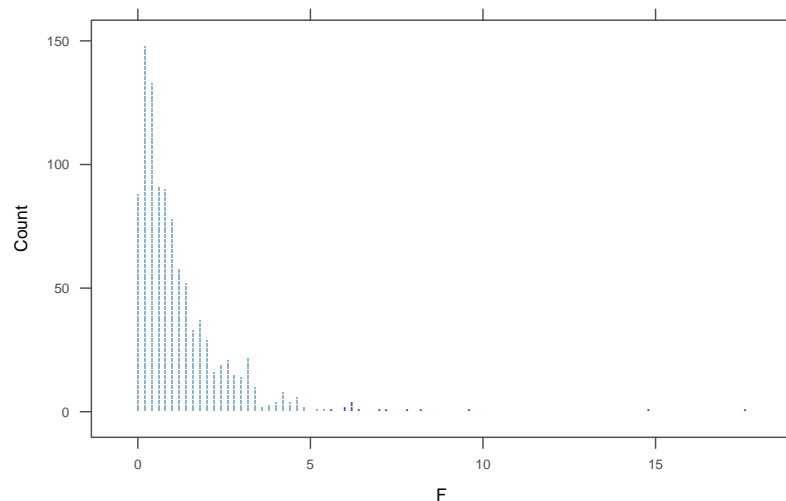
Figure8.3

```
TRUE FALSE <NA>
  15   985  1000
```

```
prop(~(F >= 5.63), data = Rand.Ants)
```

```
TRUE
0.0075
```

```
dotPlot(~F, width = 0.2, groups = (F <= 5.63), data = Rand.Ants)
```



The F-distribution

Under certain conditions, the F statistic has a known distribution (called the F distribution). Those conditions are

1. The null hypothesis is true (i.e., each group has the same mean)
2. Each group is sampled from a normal population
3. Each population group has the same standard deviation

When these conditions are met, we can use the F -distribution to compute the p-value without generating the randomization distribution.

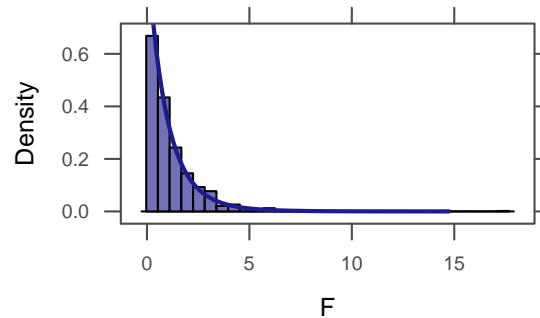
- F distributions have two parameters – the degrees of freedom for the numerator and for the denominator. In our example, this is 2 for the numerator and 7 for the denominator.

- When H_0 is true, the numerator and denominator both have a mean of 1, so F will tend to be close to 1.
- When H_0 is false, there is more difference between the groups, so the numerator tends to be larger.
This means we will reject the null hypothesis when F gets large enough.
- The p-value is computed using `pf()`.

Figure 8.4

```
histogram(~F, width = 4/7, center = 0.25, data = Rand.Ants)
plotDist("f", df1 = 2, df2 = 21, add = TRUE)
```

Figure8.4



More Examples of ANOVA

Example 8.5

```
head(StudentSurvey, 3)
```

Example8.5

	Year	Gender	Smoke	Award	HigherSAT	Exercise	TV	Height	Weight	Siblings	BirthOrder
1	Senior	M	No	Olympic	Math	10	1	71	180	4	4
2	Sophomore	F	Yes	Academy	Math	4	7	66	120	2	2
3	FirstYear	M	No	Nobel	Math	14	5	72	208	2	1

	VerbalSAT	MathSAT	SAT	GPA	Pulse	Piercings	Sex
1	540	670	1210	3.13	54	0	Male
2	520	630	1150	2.50	66	3	Female
3	550	560	1110	2.55	130	0	Male

```
favstats(~Pulse, data = StudentSurvey)
```

	min	Q1	median	Q3	max	mean	sd	n	missing
	35	62	70	77.75	130	69.57459	12.20514	362	0

```
favstats(Pulse ~ Award, data = StudentSurvey)
```

	Award	min	Q1	median	Q3	max	mean	sd	n	missing
1	Academy	42	64.5	71	76	95	70.51613	12.35818	31	0
2	Nobel	40	65.0	72	80	130	72.21477	13.09093	149	0
3	Olympic	35	60.0	68	74	96	67.25275	10.97067	182	0

```
anova(lm(Pulse ~ Award, StudentSurvey))
```

Analysis of Variance Table

Response: Pulse

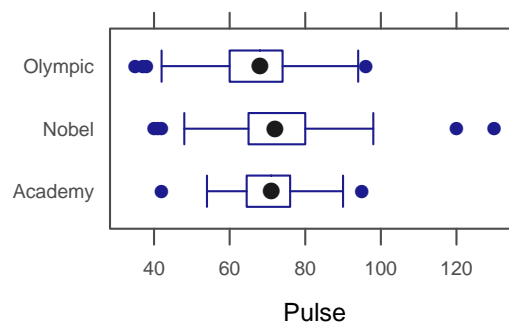
	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Award	2	2047	1023.62	7.1039	0.0009425 ***
Residuals	359	51729	144.09		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Figure 8.5

```
bwplot(Award ~ Pulse, data = StudentSurvey)
```

Figure8.5



ANOVA Calculations

- Between group variability: $G = \text{groupMean} - \text{grandMean}$

This measures how different a group is from the overall average.

- Within group variability: $E = \text{response} - \text{groupMean}$

This measures how different and individual is from its group average. E stands for “error”, but just as in “standard error” it is not a “mistake”. It is simply measure how different an individual response is from the model prediction (in this case, the group mean).

The individual values of E are called **residuals**.

Example 8.6

Let's first compute the grand mean and group means.

Example8.6

SandwichAnts

	Butter	Filling	Bread	Ants	Order
1	no	Vegemite	Rye	18	10
2	no	Peanut Butter	Rye	43	26
3	no	Ham & Pickles	Rye	44	39
4	no	Vegemite	Wholemeal	29	25
5	no	Peanut Butter	Wholemeal	59	35
6	no	Ham & Pickles	Wholemeal	34	1
7	no	Vegemite	Multigrain	42	44
8	no	Peanut Butter	Multigrain	22	36
9	no	Ham & Pickles	Multigrain	36	32
10	no	Vegemite	White	42	33
11	no	Peanut Butter	White	25	34
12	no	Ham & Pickles	White	49	13
13	no	Vegemite	Rye	31	14
14	no	Peanut Butter	Rye	36	31
15	no	Ham & Pickles	Rye	54	20
16	no	Vegemite	Wholemeal	21	19
17	no	Peanut Butter	Wholemeal	47	38
18	no	Ham & Pickles	Wholemeal	65	5
19	no	Vegemite	Multigrain	38	21
20	no	Peanut Butter	Multigrain	19	22
21	no	Ham & Pickles	Multigrain	59	8
22	no	Vegemite	White	25	41
23	no	Peanut Butter	White	21	16
24	no	Ham & Pickles	White	53	23

```
mean(Ants, data = SandwichAnts) # grand mean
```

```
[1] 38
```

```
mean(Ants ~ Filling, data = SandwichAnts) # group means
```

Ham & Pickles	Peanut Butter	Vegemite
49.25	34.00	30.75

And add those to our data frame

Example8.6b

```
SA <- transform(SandwichAnts, groupMean = c(30.75, 34, 49.25, 30.75, 34, 49.25, 30.75, 34,
49.25, 30.75, 34, 49.25, 30.75, 34, 49.25, 30.75, 34, 49.25, 30.75, 34, 49.25, 30.75, 34,
49.25))
```

```
SA <- transform(SA, grandMean = rep(38, 24))
```

```
SA
```

	Butter	Filling	Bread	Ants	Order	groupMean	grandMean
1	no	Vegemite	Rye	18	10	30.75	38
2	no	Peanut Butter	Rye	43	26	34.00	38
3	no	Ham & Pickles	Rye	44	39	49.25	38
4	no	Vegemite	Wholemeal	29	25	30.75	38
5	no	Peanut Butter	Wholemeal	59	35	34.00	38
6	no	Ham & Pickles	Wholemeal	34	1	49.25	38

7	no	Vegemite	Multigrain	42	44	30.75	38
8	no	Peanut Butter	Multigrain	22	36	34.00	38
9	no	Ham & Pickles	Multigrain	36	32	49.25	38
10	no	Vegemite	White	42	33	30.75	38
11	no	Peanut Butter	White	25	34	34.00	38
12	no	Ham & Pickles	White	49	13	49.25	38
13	no	Vegemite	Rye	31	14	30.75	38
14	no	Peanut Butter	Rye	36	31	34.00	38
15	no	Ham & Pickles	Rye	54	20	49.25	38
16	no	Vegemite	Wholemeal	21	19	30.75	38
17	no	Peanut Butter	Wholemeal	47	38	34.00	38
18	no	Ham & Pickles	Wholemeal	65	5	49.25	38
19	no	Vegemite	Multigrain	38	21	30.75	38
20	no	Peanut Butter	Multigrain	19	22	34.00	38
21	no	Ham & Pickles	Multigrain	59	8	49.25	38
22	no	Vegemite	White	25	41	30.75	38
23	no	Peanut Butter	White	21	16	34.00	38
24	no	Ham & Pickles	White	53	23	49.25	38

```
SA <- transform(SA, M = groupMean - grandMean)
SA <- transform(SA, E = Ants - groupMean)
SA
```

Example8.6c

	Butter	Filling	Bread	Ants	Order	groupMean	grandMean	M	E
1	no	Vegemite	Rye	18	10	30.75	38	-7.25	-12.75
2	no	Peanut Butter	Rye	43	26	34.00	38	-4.00	9.00
3	no	Ham & Pickles	Rye	44	39	49.25	38	11.25	-5.25
4	no	Vegemite	Wholemeal	29	25	30.75	38	-7.25	-1.75
5	no	Peanut Butter	Wholemeal	59	35	34.00	38	-4.00	25.00
6	no	Ham & Pickles	Wholemeal	34	1	49.25	38	11.25	-15.25
7	no	Vegemite	Multigrain	42	44	30.75	38	-7.25	11.25
8	no	Peanut Butter	Multigrain	22	36	34.00	38	-4.00	-12.00
9	no	Ham & Pickles	Multigrain	36	32	49.25	38	11.25	-13.25
10	no	Vegemite	White	42	33	30.75	38	-7.25	11.25
11	no	Peanut Butter	White	25	34	34.00	38	-4.00	-9.00
12	no	Ham & Pickles	White	49	13	49.25	38	11.25	-0.25
13	no	Vegemite	Rye	31	14	30.75	38	-7.25	0.25
14	no	Peanut Butter	Rye	36	31	34.00	38	-4.00	2.00
15	no	Ham & Pickles	Rye	54	20	49.25	38	11.25	4.75
16	no	Vegemite	Wholemeal	21	19	30.75	38	-7.25	-9.75
17	no	Peanut Butter	Wholemeal	47	38	34.00	38	-4.00	13.00
18	no	Ham & Pickles	Wholemeal	65	5	49.25	38	11.25	15.75
19	no	Vegemite	Multigrain	38	21	30.75	38	-7.25	7.25
20	no	Peanut Butter	Multigrain	19	22	34.00	38	-4.00	-15.00
21	no	Ham & Pickles	Multigrain	59	8	49.25	38	11.25	9.75
22	no	Vegemite	White	25	41	30.75	38	-7.25	-5.75
23	no	Peanut Butter	White	21	16	34.00	38	-4.00	-13.00
24	no	Ham & Pickles	White	53	23	49.25	38	11.25	3.75

As we did with variance, we will square these differences:

```
SA <- transform(SA, M2 = (groupMean - grandMean)^2)
SA <- transform(SA, E2 = (Ants - groupMean)^2)
SA
```

Example8.6d

	Butter	Filling	Bread	Ants	Order	groupMean	grandMean	M	E	M2
1	no	Vegemite	Rye	18	10	30.75	38	-7.25	-12.75	52.5625
2	no	Peanut Butter	Rye	43	26	34.00	38	-4.00	9.00	16.0000
3	no	Ham & Pickles	Rye	44	39	49.25	38	11.25	-5.25	126.5625
4	no	Vegemite	Wholemeal	29	25	30.75	38	-7.25	-1.75	52.5625
5	no	Peanut Butter	Wholemeal	59	35	34.00	38	-4.00	25.00	16.0000
6	no	Ham & Pickles	Wholemeal	34	1	49.25	38	11.25	-15.25	126.5625
7	no	Vegemite	Multigrain	42	44	30.75	38	-7.25	11.25	52.5625
8	no	Peanut Butter	Multigrain	22	36	34.00	38	-4.00	-12.00	16.0000
9	no	Ham & Pickles	Multigrain	36	32	49.25	38	11.25	-13.25	126.5625
10	no	Vegemite	White	42	33	30.75	38	-7.25	11.25	52.5625
11	no	Peanut Butter	White	25	34	34.00	38	-4.00	-9.00	16.0000
12	no	Ham & Pickles	White	49	13	49.25	38	11.25	-0.25	126.5625
13	no	Vegemite	Rye	31	14	30.75	38	-7.25	0.25	52.5625
14	no	Peanut Butter	Rye	36	31	34.00	38	-4.00	2.00	16.0000
15	no	Ham & Pickles	Rye	54	20	49.25	38	11.25	4.75	126.5625
16	no	Vegemite	Wholemeal	21	19	30.75	38	-7.25	-9.75	52.5625
17	no	Peanut Butter	Wholemeal	47	38	34.00	38	-4.00	13.00	16.0000
18	no	Ham & Pickles	Wholemeal	65	5	49.25	38	11.25	15.75	126.5625
19	no	Vegemite	Multigrain	38	21	30.75	38	-7.25	7.25	52.5625
20	no	Peanut Butter	Multigrain	19	22	34.00	38	-4.00	-15.00	16.0000
21	no	Ham & Pickles	Multigrain	59	8	49.25	38	11.25	9.75	126.5625
22	no	Vegemite	White	25	41	30.75	38	-7.25	-5.75	52.5625
23	no	Peanut Butter	White	21	16	34.00	38	-4.00	-13.00	16.0000
24	no	Ham & Pickles	White	53	23	49.25	38	11.25	3.75	126.5625
E2										
1	162.5625									
2	81.0000									
3	27.5625									
4	3.0625									
5	625.0000									
6	232.5625									
7	126.5625									
8	144.0000									
9	175.5625									
10	126.5625									
11	81.0000									
12	0.0625									
13	0.0625									
14	4.0000									
15	22.5625									
16	95.0625									
17	169.0000									
18	248.0625									
19	52.5625									
20	225.0000									
21	95.0625									
22	33.0625									
23	169.0000									
24	14.0625									

And then add them up (SS stands for “sum of squares”)

```
SST <- sum(~((Ants - grandMean)^2), data = SA)
SST
```

```
[1] 4474
```

Example8.6e

```
SSM <- sum(~M2, data = SA)
SSM # also called SSG
```

```
[1] 1561
```

```
SSE <- sum(~E2, data = SA)
SSE
```

```
[1] 2913
```

8.2 Pairwise Comparisons and Inference After ANOVA

Using ANOVA for Inferences about Group Means

We can construct a confidence interval for any of the means by just taking a subset of the data and using `t.test()`, but there are some problems with this approach. Most importantly,

We were primarily interested in comparing the means across the groups. Often people will display confidence intervals for each group and look for “overlapping” intervals. But this is not the best way to look for differences.

Nevertheless, you will sometimes see graphs showing multiple confidence intervals and labeling them to indicate which means appear to be different from which. (See the solution to problem 15.3 for an example.)

Example 8.7

```
anova(Ants.Model)
```

Example8.7

Analysis of Variance Table

Response: Ants

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Filling	2	1561	780.50	5.6267	0.01105 *
Residuals	21	2913	138.71		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
MSE <- 138.7
```

```
mean(Ants ~ Filling, data = SandwichAnts)
```

Ham & Pickles	Peanut Butter	Vegemite
49.25	34.00	30.75

```
mean <- 34
```

```
t.star <- qt(0.975, df = 21)
```

```
t.star
```

```
[1] 2.079614

mean - t.star * (sqrt(MSE)/sqrt(8))

[1] 25.34084

mean + t.star * (sqrt(MSE)/sqrt(8))

[1] 42.65916
```

Example8.7b

```
TukeyHSD(Ants.Model)
```

```
Tukey multiple comparisons of means
95% family-wise confidence level
```

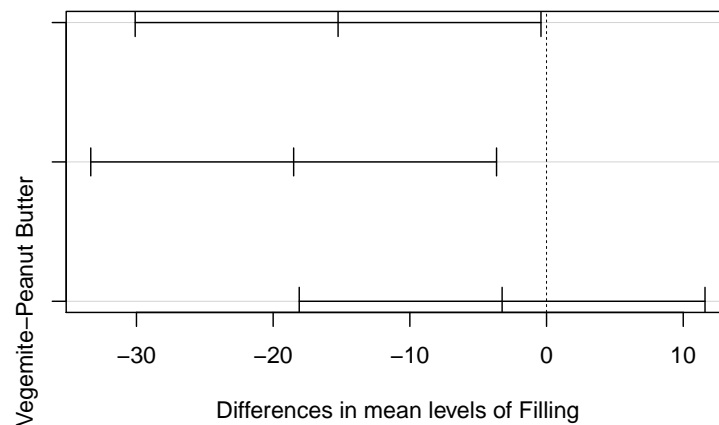
```
Fit: aov(formula = x)
```

```
$Filling
```

	diff	lwr	upr	p adj
Peanut Butter-Ham & Pickles	-15.25	-30.09326	-0.4067363	0.0433492
Vegemite-Ham & Pickles	-18.50	-33.34326	-3.6567363	0.0131006
Vegemite-Peanut Butter	-3.25	-18.09326	11.5932637	0.8466376

```
plot(TukeyHSD(Ants.Model))
```

95% family-wise confidence level



Example 8.8

```
MSE <- 138.7
mean(Ants ~ Filling, data = SandwichAnts)
```

Example8.8

Ham & Pickles	Peanut Butter	Vegemite
49.25	34.00	30.75

```
diff.mean <- (30.75 - 49.25)
t.star <- qt(0.975, df = 21)
t.star
```

```
[1] 2.079614
```

```
diff.mean - t.star * (sqrt(MSE * (1/8 + 1/8)))
```

```
[1] -30.74591
```

```
diff.mean + t.star * (sqrt(MSE * (1/8 + 1/8)))
```

```
[1] -6.254094
```

Example8.8b

Example 8.9

```
MSE <- 138.7
mean(Ants ~ Filling, data = SandwichAnts)
```

Ham & Pickles	Peanut Butter	Vegemite
49.25	34.00	30.75

```
diff.mean <- (30.75 - 34)
```

Example8.9

```
t <- diff.mean/sqrt(MSE * (1/8 + 1/8))
t
```

```
[1] -0.5519187
```

```
pt(t, df = 21) * 2
```

```
[1] 0.5868302
```

Example8.9b

Lots of Pairwise Comparisons

Example 8.10

```
head(TextbookCosts)
```

Example8.10

```
      Field Books Cost
1 SocialScience    3   77
2 NaturalScience    2  231
3 NaturalScience    1  189
4 SocialScience    6   85
5 NaturalScience    1  113
6 Humanities       9  132
```

```
Books.Model <- lm(Cost ~ Field, data = TextbookCosts)
anova(Books.Model)
```

Analysis of Variance Table

Response: Cost

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Field	3	30848	10283	4.0547	0.01397 *
Residuals	36	91294	2536		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
summary(Books.Model)
```

Call:

```
lm(formula = Cost ~ Field, data = TextbookCosts)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-77.60	-35.30	-4.95	36.90	102.70

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	94.60	15.92	5.940	8.35e-07 ***
FieldHumanities	25.70	22.52	1.141	0.26134
FieldNaturalScience	76.20	22.52	3.384	0.00174 **
FieldSocialScience	23.70	22.52	1.052	0.29965

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 50.36 on 36 degrees of freedom

Multiple R-squared: 0.2526, Adjusted R-squared: 0.1903

F-statistic: 4.055 on 3 and 36 DF, p-value: 0.01397

```
TukeyHSD(Books.Model)
```

Example8.10b

Tukey multiple comparisons of means
95% family-wise confidence level

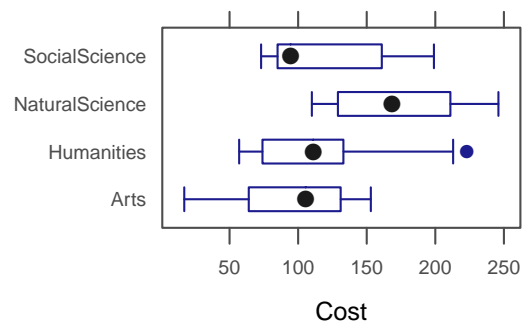
```
Fit: aov(formula = x)
```

\$Field	diff	lwr	upr	p adj
Humanities-Arts	25.7	-34.95384	86.353844	0.6669143
NaturalScience-Arts	76.2	15.54616	136.853844	0.0090147
SocialScience-Arts	23.7	-36.95384	84.353844	0.7201024
NaturalScience-Humanities	50.5	-10.15384	111.153844	0.1312366
SocialScience-Humanities	-2.0	-62.65384	58.653844	0.9997441
SocialScience-NaturalScience	-52.5	-113.15384	8.153844	0.1097759

Figure 8.8

```
bwplot(Field ~ Cost, data = TextbookCosts)
```

Figure8.8



9

Inference for Regression

9.1 Inference for Slope and Correlation

Simple Linear Model

$$Y = \beta_0 + \beta_1 x + \epsilon \quad \text{where } \epsilon \sim \text{Norm}(0, \sigma).$$

In other words:

- The mean response for a given predictor value x is given by a linear formula

$$\text{mean response} = \beta_0 + \beta_1 x$$

- The distribution of all responses for a given predictor value x is normal.
- The standard deviation of the responses is the same for each predictor value.

One of the goals in simple linear regression is to estimate this linear relationship – that is to estimate the intercept and the slope.

Of course, there are lots of lines. We want to determine the line that fits the data best. But what does that mean?

The usual method is called the **method of least squares** and chooses the line that has the *smallest possible sum of squares of residuals*, where residuals are defined by

$$\text{residual} = \text{observed response} - \text{predicted response}$$

For a line with equation $y = b_0 + b_1 x$, this would be

$$e_i = y_i - (b_0 + b_1 x)$$

Simple calculus (that you don't need to know) allows us to compute the best b_0 and b_1 possible. These best values define the least squares regression line. Fortunately, statistical software packages do all this work for us. In R, the command that does this is `lm()`.

Example 9.1

```
lm(Price ~ PPM, data = InkjetPrinters)
```

Example9.1

Call:

```
lm(formula = Price ~ PPM, data = InkjetPrinters)
```

Coefficients:

```
(Intercept)      PPM
    -94.22      90.88
```

You can get terser output with

```
coef(lm(Price ~ PPM, data = InkjetPrinters))
```

Example9.1b

```
(Intercept)      PPM
    -94.22176    90.87807
```

You can also get more information with

```
summary(lm(Price ~ PPM, data = InkjetPrinters))
```

Example9.1c

Call:

```
lm(formula = Price ~ PPM, data = InkjetPrinters)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-79.38 -51.40  -3.49   43.85   87.76
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -94.22      56.40   -1.671 0.112086
PPM             90.88      19.49    4.663 0.000193 ***
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 58.55 on 18 degrees of freedom

Multiple R-squared: 0.5471, Adjusted R-squared: 0.522

F-statistic: 21.75 on 1 and 18 DF, p-value: 0.0001934

So our regression equation is

$$\widehat{\text{Price}} = -94.2217606 + 90.8780677 \cdot \text{PPM}$$

For example, this suggests that the average price for inkjet printers that print 3 pages per minute is

$$\widehat{\text{Price}} = -94.2217606 + 90.8780677 \cdot 3.0 = 178.4124425$$

Inference for Slope

Figure 9.1

```
xypLOT(Price ~ PPM, data = InkjetPrinters, type = c("p", "r"))
```

Figure9.1

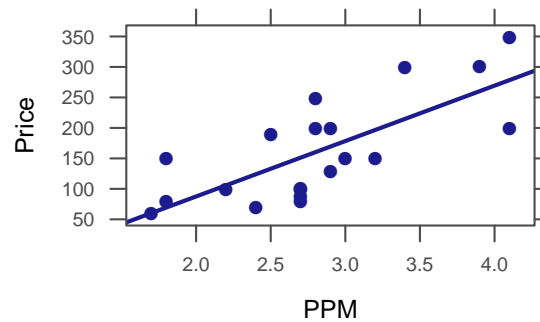


Figure 9.2

```
Boot.Ink <- do(1000) * lm(Price ~ PPM, data = resample(InkjetPrinters))
favstats(~PPM, data = Boot.Ink)
```

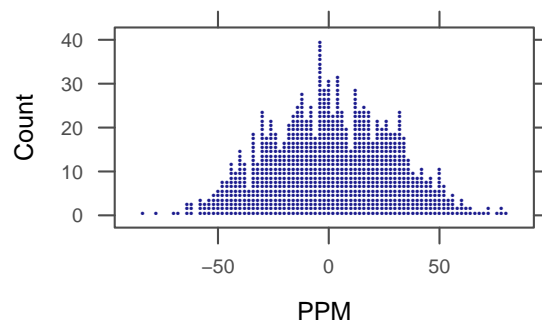
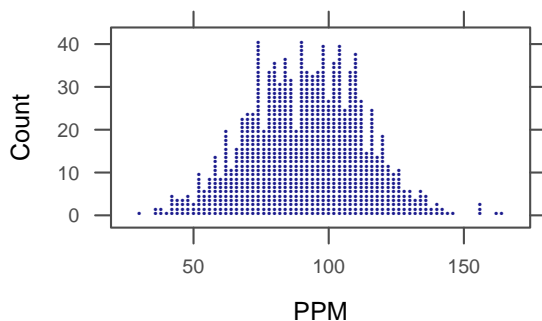
Figure9.2

min	Q1	median	Q3	max	mean	sd	n	missing
29.3309	76.49386	91.76079	106.6797	164.1968	91.50961	21.29004	1000	0

```
dotPlot(~PPM, width = 2, data = Boot.Ink)
Rand.Ink <- do(1000) * lm(Price ~ shuffle(PPM), data = InkjetPrinters)
favstats(~PPM, data = Rand.Ink)
```

min	Q1	median	Q3	max	mean	sd	n	missing
-84.79087	-18.95518	0.23489	21.73508	80.24154	0.943161	28.13176	1000	0

```
dotPlot(~PPM, width = 2, data = Rand.Ink)
```



Example 9.2

```
summary(lm(Price ~ PPM, data = InkjetPrinters))
```

Example9.2

Call:

```
lm(formula = Price ~ PPM, data = InkjetPrinters)
```

Residuals:

Min	1Q	Median	3Q	Max
-79.38	-51.40	-3.49	43.85	87.76

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-94.22	56.40	-1.671	0.112086
PPM	90.88	19.49	4.663	0.000193 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 58.55 on 18 degrees of freedom

Multiple R-squared: 0.5471, Adjusted R-squared: 0.522

F-statistic: 21.75 on 1 and 18 DF, p-value: 0.0001934

```
confint(lm(Price ~ PPM, data = InkjetPrinters), "PPM")
```

	2.5 %	97.5 %
PPM	49.93604	131.8201

Example 9.3

```
head(RestaurantTips)
```

Example9.3

	Bill	Tip	Credit	Guests	Day	Server	PctTip	CreditCard
1	23.70	10.00	n	2	Fri	A	42.2	No
2	36.11	7.00	n	3	Fri	B	19.4	No
3	31.99	5.01	y	2	Fri	A	15.7	Yes
4	17.39	3.61	y	2	Fri	B	20.8	Yes
5	15.41	3.00	n	2	Fri	B	19.5	No
6	18.62	2.50	n	2	Fri	A	13.4	No

```
summary(lm(Tip ~ Bill, data = RestaurantTips))
```

Call:

```
lm(formula = Tip ~ Bill, data = RestaurantTips)
```

Residuals:

Min	1Q	Median	3Q	Max
-2.3911	-0.4891	-0.1108	0.2839	5.9738

Coefficients:

```

      Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.292267   0.166160  -1.759   0.0806 .
Bill         0.182215   0.006451  28.247  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9795 on 155 degrees of freedom
Multiple R-squared:  0.8373, Adjusted R-squared:  0.8363
F-statistic: 797.9 on 1 and 155 DF,  p-value: < 2.2e-16

confint(lm(Tip ~ Bill, data = RestaurantTips), "Bill", level = 0.9)

      5 %      95 %
Bill 0.1715402 0.1928893

```

Example 9.4

1. $H_0: \beta_1 = 0; H_a: \beta_1 \neq 0$
2. Test statistic: $b_1 = 0.0488$ (sample slope)
3. t-test for slope:

```

summary(lm(PctTip ~ Bill, data = RestaurantTips))

```

Example9.4

```

Call:
lm(formula = PctTip ~ Bill, data = RestaurantTips)

Residuals:
    Min       1Q   Median       3Q      Max
-8.9927 -2.3096 -0.6455  1.4679 25.5335

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 15.50965    0.73956   20.97  <2e-16 ***
Bill         0.04881    0.02871    1.70  0.0911 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.36 on 155 degrees of freedom
Multiple R-squared:  0.01831, Adjusted R-squared:  0.01197
F-statistic: 2.89 on 1 and 155 DF,  p-value: 0.09112

```

t-Test for Correlation

Example 9.5

```
summary(lm(CostBW ~ PPM, data = InkjetPrinters))
```

Example9.5

```
Call:
lm(formula = CostBW ~ PPM, data = InkjetPrinters)

Residuals:
    Min       1Q   Median       3Q      Max
-2.1383 -0.7288 -0.3365  0.5315  3.8066

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   8.6829     1.2839   6.763 2.46e-06 ***
PPM          -1.5517     0.4436  -3.498 0.00257 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.333 on 18 degrees of freedom
Multiple R-squared:  0.4046, Adjusted R-squared:  0.3715
F-statistic: 12.23 on 1 and 18 DF, p-value: 0.002571
```

Example 9.6

```
summary(lm(PctTip ~ Bill, data = RestaurantTips))
```

Example9.6

```
Call:
lm(formula = PctTip ~ Bill, data = RestaurantTips)

Residuals:
    Min       1Q   Median       3Q      Max
-8.9927 -2.3096 -0.6455  1.4679 25.5335

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 15.50965     0.73956  20.97  <2e-16 ***
Bill         0.04881     0.02871   1.70  0.0911 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.36 on 155 degrees of freedom
Multiple R-squared:  0.01831, Adjusted R-squared:  0.01197
F-statistic:  2.89 on 1 and 155 DF, p-value: 0.09112
```

Coefficient of Determination: R-squared

Example 9.7

```
summary(lm(Price ~ PPM, data = InkjetPrinters))
```

Example9.7

```
Call:
lm(formula = Price ~ PPM, data = InkjetPrinters)
```

```

Residuals:
    Min       1Q   Median       3Q      Max
-79.38 -51.40  -3.49   43.85   87.76

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -94.22     56.40  -1.671  0.112086
PPM             90.88     19.49   4.663  0.000193 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 58.55 on 18 degrees of freedom
Multiple R-squared:  0.5471, Adjusted R-squared:  0.522
F-statistic: 21.75 on 1 and 18 DF, p-value: 0.0001934

```

Checking Conditions for a Simple Linear Model

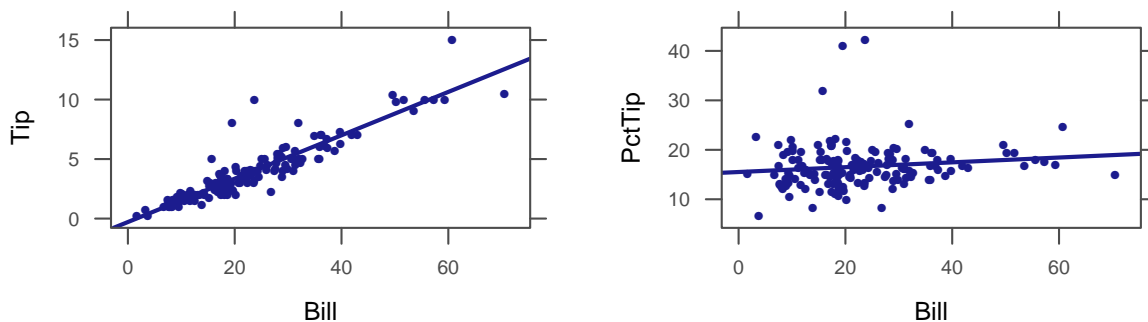
Example 9.9

```

xyplot(Tip ~ Bill, data = RestaurantTips, type = c("p", "r"), cex = 0.5)
xyplot(PctTip ~ Bill, data = RestaurantTips, type = c("p", "r"), cex = 0.5)

```

Example 9.9



9.2 ANOVA for Regression

Partitioning Variability

We can also think about regression as a way to analyze the variability in the response. This is a lot like the ANOVA tables we have seen before. This time:

$$\begin{aligned}
 SST &= \sum (y - \bar{y})^2 \\
 SSE &= \sum (y - \hat{y})^2 \\
 SSM &= \sum (\hat{y} - \bar{y})^2 \\
 SST &= SSM + SSE
 \end{aligned}$$

As before, when SSM is large and SSE is small, then the model ($\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$) explains a lot of the variability and little is left unexplained (SSE). On the other hand, if SSM is small and SSE is large, then the model explains only a little of the variability and most of it is due to things not explained by the model.

Example 9.10

```
summary(lm(Calories ~ Sugars, Cereal))
```

Example9.10

Call:
lm(formula = Calories ~ Sugars, data = Cereal)

Residuals:

	Min	1Q	Median	3Q	Max
	-36.574	-25.282	-2.549	17.796	51.805

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	88.9204	10.8120	8.224	5.96e-09 ***
Sugars	4.3103	0.9269	4.650	7.22e-05 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 26.61 on 28 degrees of freedom
Multiple R-squared: 0.4357, Adjusted R-squared: 0.4156
F-statistic: 21.62 on 1 and 28 DF, p-value: 7.217e-05

```
anova(lm(Calories ~ Sugars, Cereal))
```

Analysis of Variance Table

Response: Calories

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Sugars	1	15316	15316.5	21.623	7.217e-05 ***
Residuals	28	19834	708.3		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

F-Statistic

- $MSM = SSM/DFM = SSM/(\text{number of groups} - 1)$
- $MSE = SSE/DFE = SSE/(n - \text{number of groups})$

MS stands for “mean square”

Our test statistic is

$$F = \frac{MSM}{MSE}$$

Example 9.11

```
SSM <- 15317
MSM <- SSM/(2 - 1)
MSM

[1] 15317

SSE <- 19834
MSE <- SSE/(30 - 2)
MSE

[1] 708.3571
```

Example9.11

```
F <- MSM/MSE
F

[1] 21.62327

pf(F, 1, 28, lower.tail = FALSE)

[1] 7.216549e-05
```

Example9.11b

Example 9.12

```
summary(lm(Calories ~ Sodium, Cereal))

Call:
lm(formula = Calories ~ Sodium, data = Cereal)

Residuals:
    Min       1Q   Median       3Q      Max
-47.387 -22.917  -8.006  18.747  76.233

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 103.75873   18.86779    5.499  7.1e-06 ***
Sodium       0.13658    0.08098    1.686   0.103
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 33.76 on 28 degrees of freedom
Multiple R-squared:  0.09221, Adjusted R-squared:  0.05979
F-statistic: 2.844 on 1 and 28 DF, p-value: 0.1028

anova(lm(Calories ~ Sodium, Cereal))
```

Example9.12

Analysis of Variance Table

Response: Calories

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Sodium	1	3241	3241.3	2.8442	0.1028
Residuals	28	31909	1139.6		

The percentage of explained variability is denoted r^2 or R^2 :

$$R^2 = \frac{SSM}{SST} = \frac{SSM}{SSM + SSE}$$

Example 9.13

The summary of the linear model shows us the **coefficient of determination** but we can also find it manually.

```
SSM <- 15317
SST <- SSM + 19834
R2 <- SSM/SST
R2

[1] 0.4357486

rsquared(lm(Calories ~ Sugars, data = Cereal))

[1] 0.4357451
```

Example9.13

```
SSM <- 3241
SST <- SSM + 31909
R2 <- SSM/SST
R2

[1] 0.09220484

rsquared(lm(Calories ~ Sodium, data = Cereal))

[1] 0.09221243
```

Example9.13b

Computational Details

Example 9.15

Again, the summary of the linear model gives us the standard deviation of the error but we can calculate it manually.

```
SSE <- 31909
SD <- sqrt(SSE/(30 - 2))
SD
```

Example9.15

```
[1] 33.75807
```

Example 9.16

```
favstats(~Sodium, data = Cereal)
```

Example9.16

```
min      Q1 median      Q3 max  mean      sd  n missing
5 183.75    217 251.25 408 220.2 77.40641 30      0
```

```
SE <- SD/(77.4 * sqrt(30 - 1)) # SD from Example 9.15
SE
```

```
[1] 0.08099116
```

9.3 Confidence and Prediction Intervals

Interpreting Confidence and Prediction Intervals

It may be very interesting to make predictions when the explanatory variable has some other value, however. There are two ways to do this in R. One uses the `predict()` function. It is simpler, however, to use the `makeFun()` function in the `mosaic` package, so that's the approach we will use here.

Prediction intervals

1. are much wider than confidence intervals
2. are very sensitive to the assumption that the population normal for each value of the predictor.
3. are (for a 95% confidence level) a little bit wider than

$$\hat{y} \pm 2SE$$

where SE is the “residual standard error” reported in the summary output.

The prediction interval is a little wider because it takes into account the uncertainty in our estimated slope and intercept as well as the variability of responses around the true regression line.

Example 9.18

First, let's build our linear model and store it.

```
ink.model <- lm(Price ~ PPM, data = InkjetPrinters)
summary(ink.model)
```

Call:
lm(formula = Price ~ PPM, data = InkjetPrinters)

Residuals:

Min	1Q	Median	3Q	Max
-79.38	-51.40	-3.49	43.85	87.76

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-94.22	56.40	-1.671	0.112086
PPM	90.88	19.49	4.663	0.000193 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 58.55 on 18 degrees of freedom
Multiple R-squared: 0.5471, Adjusted R-squared: 0.522
F-statistic: 21.75 on 1 and 18 DF, p-value: 0.0001934

Example9.18

Now let's create a function that will estimate values of **Price** for a given value of **PPM**:

```
Ink.Price <- makeFun(ink.model)
```

Example9.18b

We can now input a PPM and see what our least squares regression line predicts for the price:

```
Ink.Price(PPM = 3) # estimate Price when PPM is 3.0
```

Example9.18c

```
1
178.4124
```

R can compute two kinds of confidence intervals for the response for a given value

1. A confidence interval for the *mean response* for a *given explanatory value* can be computed by adding `interval='confidence'`.

```
Ink.Price(PPM = 3, interval = "confidence")
```

Example9.18d

```
      fit      lwr      upr
1 178.4124 149.8849 206.94
```

2. An interval for an *individual response* (called a prediction interval to avoid confusion with the confidence interval above) can be computed by adding `interval='prediction'` instead.

```
Ink.Price(PPM = 3, interval = "prediction")
```

Example9.18e

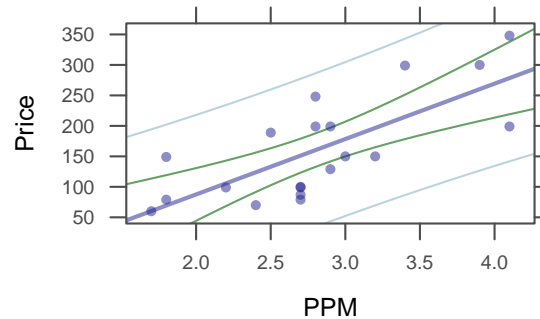
```
      fit      lwr      upr
1 178.4124  52.1476 304.6773
```

Figure 9.13

The figure below shows the confidence (dotted) and prediction (dashed) intervals as bands around the regression line.

```
xyplot(Price ~ PPM, data = InkjetPrinters, panel = panel.lmbands, cex = 0.6, alpha = 0.5)
```

Figure9.13



As the graph illustrates, the intervals are narrow near the center of the data and wider near the edges of the data. It is not safe to extrapolate beyond the data (without additional information), since there is no data to let us know whether the pattern of the data extends.

10

Multiple Regression

10.1 Multiple Predictors

Multiple Regression Model

Example 10.1

```
lm(Price ~ PPM + CostBW, InkjetPrinters)
```

Example10.1

Call:

```
lm(formula = Price ~ PPM + CostBW, data = InkjetPrinters)
```

Coefficients:

(Intercept)	PPM	CostBW
89.20	58.10	-21.13

```
Ink.Price <- makeFun(lm(Price ~ PPM + CostBW, data = InkjetPrinters))
```

```
Ink.Price(PPM = 3, CostBW = 3.7)
```

```
1  
185.3402
```

Testing Individual Terms in a Model

Example 10.2

```
summary(lm(Price ~ PPM + CostBW, data = InkjetPrinters))
```

Example10.2

Call:

```
lm(formula = Price ~ PPM + CostBW, data = InkjetPrinters)
```

```

Residuals:
    Min       1Q   Median       3Q      Max
-80.910 -35.605  -6.985   38.906   82.730

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    89.205     95.744   0.932  0.3645
PPM             58.099     22.785   2.550  0.0207 *
CostBW         -21.125      9.341  -2.262  0.0371 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 52.82 on 17 degrees of freedom
Multiple R-squared:  0.6519, Adjusted R-squared:  0.6109
F-statistic: 15.92 on 2 and 17 DF,  p-value: 0.0001273

```

Example 10.3

```

summary(lm(Bodyfat ~ Weight + Height, data = BodyFat))

```

Call:
lm(formula = Bodyfat ~ Weight + Height, data = BodyFat)

```

Residuals:
    Min       1Q   Median       3Q      Max
-12.7697  -3.9527  -0.5364   4.0473  13.2829

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  71.48247    16.20086   4.412 2.65e-05 ***
Weight        0.23156     0.02382   9.721 5.36e-16 ***
Height       -1.33568     0.25891  -5.159 1.32e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.754 on 97 degrees of freedom
Multiple R-squared:  0.494, Adjusted R-squared:  0.4836
F-statistic: 47.35 on 2 and 97 DF,  p-value: 4.48e-15

```

Example10.3

Example 10.4

```

summary(lm(Bodyfat ~ Weight + Height + Abdomen, data = BodyFat))

```

Call:
lm(formula = Bodyfat ~ Weight + Height + Abdomen, data = BodyFat)

```

Residuals:
    Min       1Q   Median       3Q      Max

```

Example10.4


```

-9.5219 -2.9969  0.0378  2.8933  9.2859

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -56.1329    18.1372  -3.095  0.002580 **
Weight      -0.1756     0.0472  -3.720  0.000335 ***
Height       0.1018     0.2444   0.417  0.677750
Abdomen       1.0747     0.1158   9.279  5.27e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.199 on 96 degrees of freedom
Multiple R-squared:  0.7332, Adjusted R-squared:  0.7249
F-statistic: 87.96 on 3 and 96 DF,  p-value: < 2.2e-16

```

ANOVA for a Multiple Regression Model

Example 10.6

```

Mod0 <- lm(Price ~ 1, data = InkjetPrinters)
Mod1 <- lm(Price ~ PPM, data = InkjetPrinters)
Mod2 <- lm(Price ~ PPM + CostBW, data = InkjetPrinters)
anova(Mod0, Mod1)

```

Analysis of Variance Table

```

Model 1: Price ~ 1
Model 2: Price ~ PPM
  Res.Df    RSS Df Sum of Sq    F    Pr(>F)
1      19 136237
2      18  61697  1    74540 21.747 0.0001934 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(Mod0, Mod2)

```

Example10.6

```

Analysis of Variance Table

Model 1: Price ~ 1
Model 2: Price ~ PPM + CostBW
  Res.Df    RSS Df Sum of Sq    F    Pr(>F)
1      19 136237
2      17  47427  2    88809 15.917 0.0001273 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Example10.7

```
Mod0 <- lm(Price ~ 1, data = InkjetPrinters)
```

```
Mod1 <- lm(Price ~ PhotoTime + CostColor, data = InkjetPrinters)
summary(Mod1)
```

Call:
lm(formula = Price ~ PhotoTime + CostColor, data = InkjetPrinters)

Residuals:

	Min	1Q	Median	3Q	Max
	-128.765	-55.548	-1.611	53.632	109.251

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	371.8920	66.8920	5.560	3.46e-05 ***
PhotoTime	0.1038	0.3663	0.283	0.78036
CostColor	-18.7323	5.2821	-3.546	0.00248 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 67.85 on 17 degrees of freedom
Multiple R-squared: 0.4255, Adjusted R-squared: 0.3579
F-statistic: 6.296 on 2 and 17 DF, p-value: 0.00899

```
anova(Mod0, Mod1)
```

Analysis of Variance Table

Model 1: Price ~ 1
Model 2: Price ~ PhotoTime + CostColor

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	19	136237				
2	17	78264	2	57973	6.2963	0.00899 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Example 10.8

```
rsquared(lm(Price ~ PPM + CostBW, data = InkjetPrinters))
```

[1] 0.6518759

```
rsquared(lm(Price ~ PhotoTime + CostColor, data = InkjetPrinters))
```

[1] 0.4255317

Example10.8

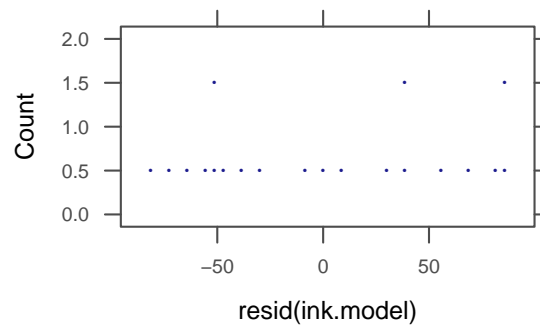
10.2 Checking Conditions for a Regression Model

Histogram/Dotplot/Boxplot of Residuals

Example 10.12

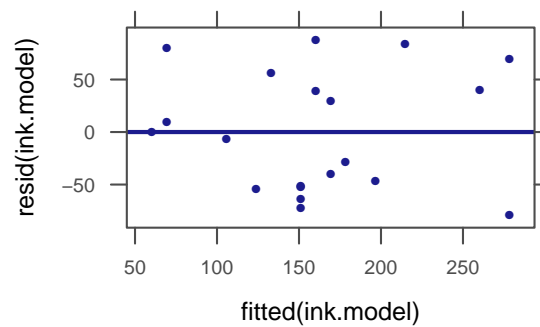
```
ink.model <- lm(Price ~ PPM, data = InkjetPrinters)
dotPlot(~resid(ink.model), cex = 0.05, nint = 40)
```

Example10.12



```
xyplot(resid(ink.model) ~ fitted(ink.model), type = c("p", "r"), cex = 0.5)
```

Example10.12b



Checking Conditions for a Multiple Regression Model

Example 10.13

```
body.model <- lm(Bodyfat ~ Weight + Abdomen, data = BodyFat)
summary(body.model)
```

Example10.13

Call:

```
lm(formula = Bodyfat ~ Weight + Abdomen, data = BodyFat)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-9.5953	-2.9776	-0.0177	2.8974	9.1920

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-48.77854	4.18098	-11.667	< 2e-16 ***
Weight	-0.16082	0.03101	-5.185	1.18e-06 ***
Abdomen	1.04408	0.08918	11.707	< 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

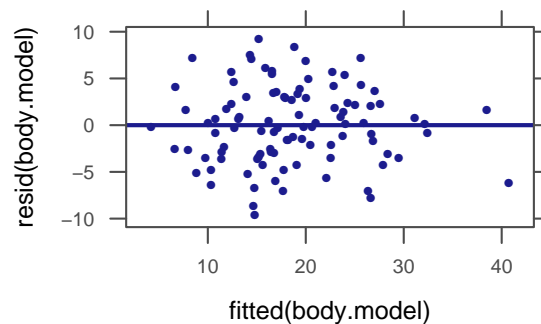
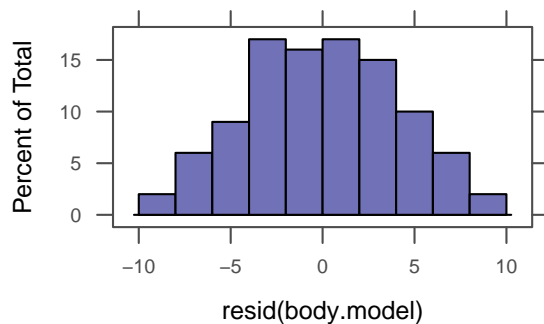
Residual standard error: 4.181 on 97 degrees of freedom

Multiple R-squared: 0.7328, Adjusted R-squared: 0.7273

F-statistic: 133 on 2 and 97 DF, p-value: < 2.2e-16

```
histogram(~resid(body.model), breaks = 10)
```

```
xyplot(resid(body.model) ~ fitted(body.model), type = c("p", "r"), cex = 0.5)
```



10.3 Using Multiple Regression

Choosing a Model

Example 10.14

```
summary(lm(Bodyfat ~ Weight + Height + Abdomen + Age + Wrist, data = BodyFat))
```

Example10.14

Call:

```
lm(formula = Bodyfat ~ Weight + Height + Abdomen + Age + Wrist,
    data = BodyFat)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-10.7319	-2.4789	-0.2068	2.7668	9.6343

```

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -24.94157   20.77414  -1.201   0.2329
Weight       -0.08434    0.05891  -1.432   0.1555
Height        0.05177    0.23849   0.217   0.8286
Abdomen       0.96762    0.13040   7.421 5.15e-11 ***
Age           0.07740    0.04868   1.590   0.1152
Wrist        -2.05797    0.72893  -2.823   0.0058 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.074 on 94 degrees of freedom
Multiple R-squared:  0.7542, Adjusted R-squared:  0.7411
F-statistic: 57.67 on 5 and 94 DF, p-value: < 2.2e-16

summary(lm(Bodyfat ~ Weight + Abdomen + Age + Wrist, data = BodyFat))

Call:
lm(formula = Bodyfat ~ Weight + Abdomen + Age + Wrist, data = BodyFat)

Residuals:
    Min       1Q   Median       3Q      Max
-10.7803  -2.4434  -0.2681   2.8294   9.5902

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -21.06107   10.52814  -2.000   0.04831 *
Weight       -0.07608    0.04474  -1.700   0.09231 .
Abdomen       0.95069    0.10399   9.142 1.13e-14 ***
Age           0.07854    0.04815   1.631   0.10620
Wrist        -2.06898    0.72350  -2.860   0.00521 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.054 on 95 degrees of freedom
Multiple R-squared:  0.754, Adjusted R-squared:  0.7437
F-statistic: 72.81 on 4 and 95 DF, p-value: < 2.2e-16

```

Example 10.15

```

summary(lm(Bodyfat ~ Weight + Abdomen + Wrist, data = BodyFat))

Call:
lm(formula = Bodyfat ~ Weight + Abdomen + Wrist, data = BodyFat)

Residuals:
    Min       1Q   Median       3Q      Max
-10.0671  -3.1182  -0.2414   2.4271   9.3611

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -28.75313    9.49382  -3.029 0.003156 **

```

Example10.15

```

Weight      -0.12360    0.03425   -3.609 0.000491 ***
Abdomen      1.04495    0.08720   11.983 < 2e-16 ***
Wrist       -1.46586    0.62722   -2.337 0.021513 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.089 on 96 degrees of freedom
Multiple R-squared:  0.7471, Adjusted R-squared:  0.7392
F-statistic: 94.56 on 3 and 96 DF,  p-value: < 2.2e-16

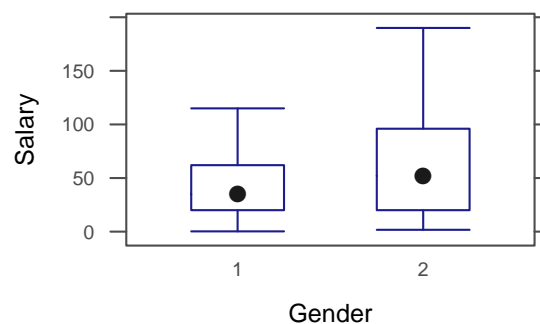
```

Categorical Variables

Figure 10.9

```
bwplot(Salary ~ Gender, horizontal = FALSE, data = SalaryGender)
```

Figure10.9



Example 10.16

```
summary(lm(Salary ~ Gender, data = SalaryGender))
```

Example10.16

Call:

```
lm(formula = Salary ~ Gender, data = SalaryGender)
```

Residuals:

```

      Min       1Q   Median       3Q      Max
-61.718 -30.131  -9.025  25.582 126.582

```

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept)   41.631      5.796    7.183 1.34e-10 ***
Gender         21.787      8.197    2.658  0.00918 **
---

```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 40.98 on 98 degrees of freedom
```

Multiple R-squared: 0.06724, Adjusted R-squared: 0.05772
F-statistic: 7.065 on 1 and 98 DF, p-value: 0.009181

Example 10.17

```
summary(lm(Salary ~ PhD, data = SalaryGender))
```

Example10.17

Call:

```
lm(formula = Salary ~ PhD, data = SalaryGender)
```

Residuals:

Min	1Q	Median	3Q	Max
-66.513	-24.488	-5.788	14.174	108.287

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	33.863	4.518	7.496	2.97e-11 ***
PhD	47.850	7.234	6.614	1.98e-09 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 35.28 on 98 degrees of freedom

Multiple R-squared: 0.3086, Adjusted R-squared: 0.3016

F-statistic: 43.75 on 1 and 98 DF, p-value: 1.979e-09

```
confint(lm(Salary ~ PhD, data = SalaryGender))
```

	2.5 %	97.5 %
(Intercept)	24.89789	42.82834
PhD	33.49385	62.20556

Accounting for Confounding Variables

Example 10.18

```
summary(lm(Salary ~ Gender + PhD + Age, data = SalaryGender))
```

Example10.18

Call:

```
lm(formula = Salary ~ Gender + PhD + Age, data = SalaryGender)
```

Residuals:

Min	1Q	Median	3Q	Max
-81.296	-18.889	-0.802	14.716	93.504

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-6.9549	10.8364	-0.642	0.52253

```

Gender      11.0944      6.7070      1.654  0.10136
PhD         36.4305      7.2534      5.023 2.35e-06 ***
Age          0.8474      0.2318      3.655 0.00042 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 32.78 on 96 degrees of freedom
Multiple R-squared:  0.4154, Adjusted R-squared:  0.3972
F-statistic: 22.74 on 3 and 96 DF,  p-value: 3.308e-11

```

Association between Explanatory Variables

Example 10.19

```

summary(lm(Final ~ Exam1 + Exam2, data = StatGrades))

```

Call:
lm(formula = Final ~ Exam1 + Exam2, data = StatGrades)

Residuals:

Min	1Q	Median	3Q	Max
-19.323	-2.550	0.613	2.963	11.443

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	30.8952	7.9973	3.863	0.000342 ***
Exam1	0.4468	0.1606	2.783	0.007733 **
Exam2	0.2212	0.1760	1.257	0.215086

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.377 on 47 degrees of freedom
Multiple R-squared: 0.5251, Adjusted R-squared: 0.5049
F-statistic: 25.98 on 2 and 47 DF, p-value: 2.515e-08

Example10.19

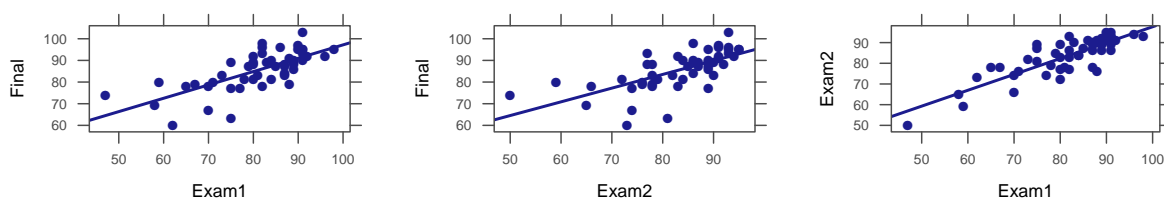
Figure 10.10

```

xyplot(Final ~ Exam1, type = c("p", "r"), data = StatGrades)
xyplot(Final ~ Exam2, type = c("p", "r"), data = StatGrades)
xyplot(Exam2 ~ Exam1, type = c("p", "r"), data = StatGrades)

```

Figure10.10



Bibliography

- [AmH82] *American Heritage Dictionary*, Houghton Mifflin Company, 1982.
- [Fis25] R. A. Fisher, *Statistical methods for research workers*, Oliver & Boyd, 1925.
- [Fis70] ———, *Statistical methods for research workers*, 14th ed., Oliver & Boyd, 1970.
- [fMA86] Consortium for Mathematics and Its Applications, *For all practical purposes: Statistics*, Intellimation, 1985–1986.
- [fMA89] ———, *Against all odds: Inside statistics*, Annenberg Media, 1989.
- [Kap09] Daniel T. Kaplan, *Statistical modeling: A fresh approach*, 2009.
- [Sal01] D. Salsburg, *The lady tasting tea: How statistics revolutionized science in the twentieth century*, W.H. Freeman, New York, 2001.
- [Utt05] Jessica M. Utts, *Seeing through statistics*, 3rd ed., Thompson Learning, 2005.

AllCountries, 33
 Births78, 17
 CIsim(), 80
 HELPrct, 11
 Hmisc, 80
 ICUAdmissions, 50
 Lock5withR, 11, 31
 MammalLongevity, 46
 OneTrueLove, 39
 Population, 19
 StudentSurvey, 31
 USStates, 60
 bargraph(), 46
 bwplot(), 46
 c(), 84
 cdata(), 87
 chisq.test(), 156
 chisq.test, 153
 cut(), 47
 data(), 10
 data.frame(), 84
 densityplot(), 46, 50
 diff(), 44
 do(), 27, 28
 fastR, 20
 favstats(), 54
 filter(), 18, 19
 fivenum(), 57
 foreign, 16
 freqpolygon(), 49
 fusion1, 20
 googlesheets, 14
 head(), 32
 histogram(), 44, 46, 49, 50
 lattice, 62, 64
 lm(), 177
 log(), 8
 log10(), 8
 makeFun(), 187
 mean(), 18
 mosaicData, 11, 17
 mosaic, 23, 27, 44, 187
 mutate(), 19
 pf(), 166
 pheno, 20
 plotDist(), 114
 pnorm, 115, 116
 predict(), 187
 prop(), 39, 42
 prop.test(), 156, 159
 qnorm(), 116
 qnorm, 115
 quantile(), 57
 rdrop2, 16
 readxl, 16
 render(), 23
 resample(), 83
 rflip(), 27, 75, 96, 104
 rmarkdown, 23
 sample(), 73
 save(), 20
 sd(), 54, 76
 select(), 19
 sqrt(), 8
 subset(), 51, 61, 66
 summary(), 54
 t.test(), 171
 tally(), 28, 40, 41, 47
 write.csv(), 20
 xYplot(), 80
 xpnorm(), 116
 xpnorm, 116
 xyplot(), 17, 18, 64
 data, 24
 Fisher, R. A., 25, 26

lady tasting tea, 25

population, 25

randomness, 24

sample, 25