

EXPLORATORY DATA ANALYSIS with R



Gary Ernest Davis

Republic of Mathematics



First *Republic of Mathematics*[™] edition, 2016.

Copyright ©2016 by Gary Ernest Davis.

All rights reserved.

Published in the United States by *Republic of Mathematics*[™]
an imprint of Krackle LLC.

THIS IS A WORK IN PROGRESS:

PLEASE FEEL FREE TO USE, BUT DO NOT CITE

Contents

I	Becoming awesome with R	5
1	Getting R	7
1.1	Obtaining R	7
1.2	Running R	9
1.2.1	RStudio - an integrated development environment for R	9
1.2.2	Welcome to the world of the experts	9
2	Using R	15
2.1	Thinking like a beginner	15
2.2	The read.table function	17
2.3	Doing things with data	19
2.4	Extracting a subset of a data frame	21
2.5	Creating a histogram	23
2.6	Summary	25
2.6.1	What we did	25
2.6.2	R functions and assignments	26
2.7	More ways of getting data into R	27
2.7.1	Files stored on your computer	27
2.8	Basic data cleaning	29
II	Descriptive statistics	33
3	The purpose of descriptive statistics	35
3.1	Why we do descriptive statistics	35
3.2	The normal distribution	36
3.2.1	Testing for normality	36

4	Histograms	37
4.1	The basics	37
4.2	Comparing two, or more, histograms	41
4.2.1	Overlaying histograms	41
4.2.2	Back to back histograms	43
4.3	Smooth histograms	44
5	Summary statistics	45
5.1	Five-number summary	45
5.1.1	A single numeric data set	45
5.1.2	Two - or more - numeric data sets	46
5.1.3	The <code>fivenum()</code> command	49
5.2	Box plots	50
5.3	Violin plots	53
6	Standard scores & their applications	55
6.1	Mean & standard deviation	55
6.1.1	Mean	55
6.1.2	Standard deviation	55
6.2	Standard scores	57
6.2.1	The average value of z & z^2	57
6.3	Skewness	58
6.3.1	What is skewness telling us?	58
6.4	Kurtosis	60
6.4.1	Kurtosis of normally distributed data	60
6.4.2	What is kurtosis telling us?	60
7	Distance from the mean	61
7.1	Empirical estimates	61
7.1.1	The standard score function	61
7.1.2	Implementation in R	61
7.1.3	Examples	62
7.2	Theoretical inequalities & their application	62
7.2.1	Chebyshev's inequality	62
7.2.2	Graphical interpretation of Chebyshev's inequality	66
7.2.3	The Vysochanskij-Petunin inequality	67
7.2.4	The Laguerre-Samuelson inequality	71
8	Comparing data sets	73
8.1	Scatter plots	73
8.2	Quantile plots	74

<i>CONTENTS</i>	3
III More Exploratory Tools	75
IV The R community	77
9 Why is it called “R”?	79
10 Who uses R?	81
11 R package development	83
12 Major figures in R development	85

Part I

Becoming awesome with R

Chapter 1

Getting R



Our aim is to help you become awesome at using R for data analysis. Whether you are coming to R from Excel, or statistical software such as SPSS, or you are a novice at data analysis, our aim is to guide you in simple, clear steps through the process of learning to use R to analyze data, in a way that will blow the socks off your colleagues.

At the end of this book we give you some background on R, its development, and the wider R community. For now, let's get R loaded on your computer so you can be up and running.

1.1 Obtaining R

When you are connected to the internet, go to the R project site: <http://www.r-project.org/>.

There you will see a page that, toward the bottom, contains the following instructions:

Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).
- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Click on the [download R](#) link to see a list of Web addresses (URLs) of sites from which you can download R to your computer.

These sites are all part of the The Comprehensive R Archive Network (CRAN).

It does not matter much which site you choose: the major difference will be availability and speed of download. Try a site close to you geographically (although that may not always be the site that will download R fastest. Live on the wild side: try an exotic location!)

When you are connected to your choice of CRAN mirror site you will see a choice of download options for R dependent on your operating system (one each for Linux, Mac OS and Windows):

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for MacOS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Select the choice appropriate for your operating system, download the application to your computer, and install.

1.2 Running R

1.2.1 RStudio - an integrated development environment for R



To begin running R you simply have to open the application by double clicking on it.

However, to learn simple, clean and productive habits right from the beginning, install [RStudio](#) - an integrated development environment (= IDE) for R.

Click on the active [RStudio](#) link to go to the RStudio site.


Click on the [download link](#) to install RStudio on your

computer.

To activate an R session simply open the RStudio application.

1.2.2 Welcome to the world of the experts

Open the RStudio application and you will see the R Console, on the bottom left, that looks like the following:

```
Console ~/ 
R version 2.15.3 (2013-03-01) -- "Security Blanket"
Copyright (C) 2013 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

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

Natural language support but running in an English locale

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

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

> |
```

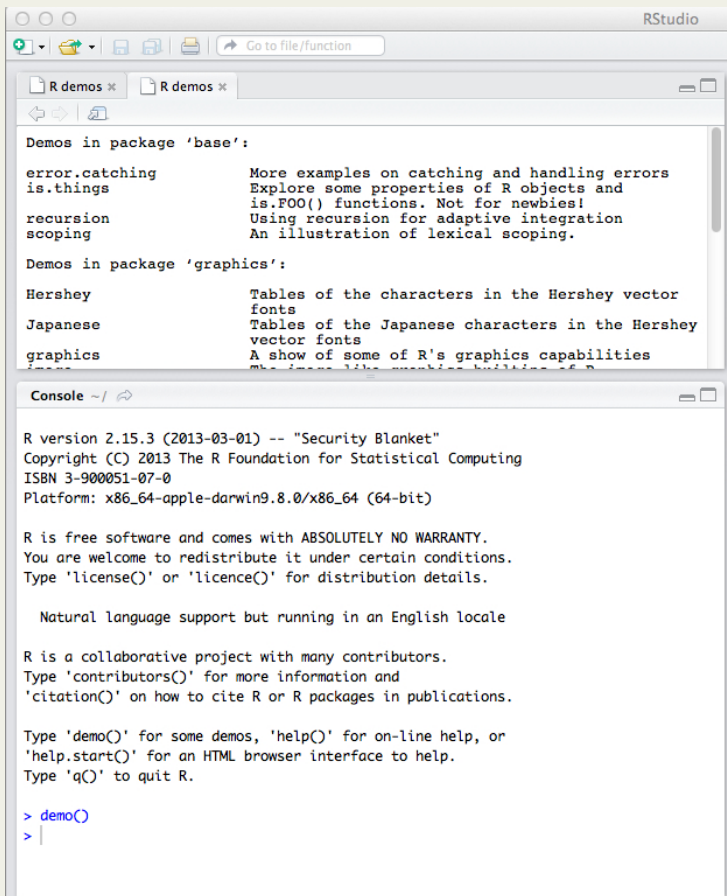
with a blinking cursor following the `>` sign. That blinking cursor looks like it wants you to do something, but what?

Perhaps the advice: “Type ‘demo()’ for some demos” will be helpful?

Typing `demo()` after the `>` prompt, as shown below:

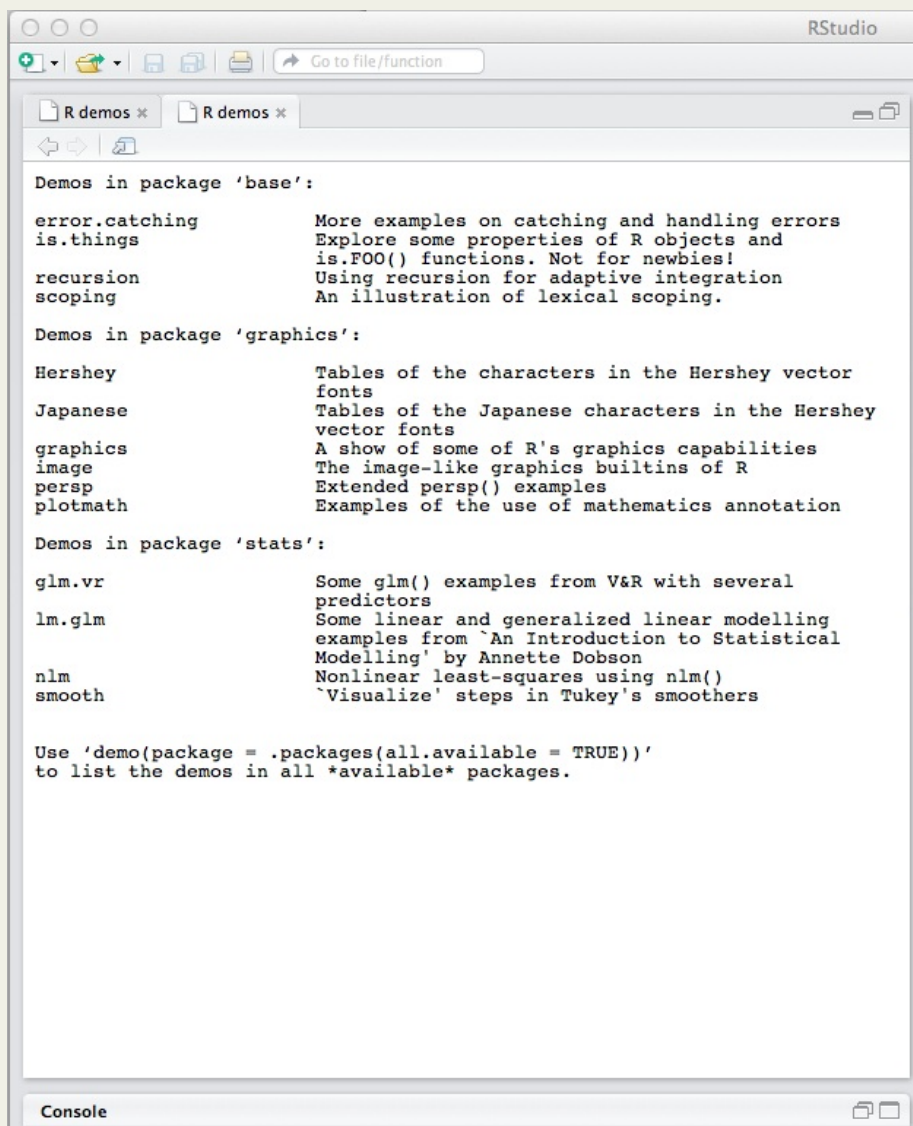
```
> demo()
```

yields a window, labelled ‘R demos’, containing the following information:



This window may already be open, above the R console, when you open RStudio.

To see it more fully, as follows:



simply click on the larger of the two folder images at the top right of the R demos window.

To a beginner this all looks overwhelming - particularly the ominous “Not for newbies!”

What can we make of this esoteric gibberish?

Is this how we have to interact with R, struggling in confusion to understand even how to proceed? Worse, if you try running some of these demos you will, in all likelihood, become even more muddled: the level of gibberish increases massively!



The outlook, fortunately, is much brighter - very much brighter. It is simply that the smart people who wrote R, and R packages, are not using Beginner’s Mind. They cannot escape the use of Expert’s Mind, and their words seem unnecessarily complicated to Beginners. Experts are scared of looking stupid, so they write complicated examples first, in order to appear smart. Experts find it hard to recall, or empathize with, the mind of a

Beginner, full of openness, eagerness, and lacking preconceptions, a clear mind in which many things are possible.

Chapter 2

Using R

2.1 Thinking like a beginner

Let us start again, this time gently.

What is it we are trying to do? Because R is a data analysis program (and programming language) you probably came to R with data analysis in mind. To this end, let us import some data from a website into R.

The data we choose is from the [STAT LABS Data](http://www.stat.berkeley.edu/users/statlabs/labs.html) site:
<http://www.stat.berkeley.edu/users/statlabs/labs.html>.

Go to that site from within this document by clicking on the previous blue text “STAT LABS Data” and choose the [Birth weight I](#) link, under “Maternal Smoking and Infant Health I”. [In case that link is broken at the time you are reading this book, you can also find the data [here](#).]

On that data site you will see two columns of data that begin as follows:

bwt	smoke
120	0
113	0
128	1
123	0
108	1
136	0
138	0
132	0
120	0
143	1
140	0
144	1

141	1
110	1
114	0
115	0
92	1

The left hand column, labelled “bwt” gives a baby’s birth weight in ounces. The right hand column, labelled “smoke” contains a “0” if the mother of the baby, whose weight is recorded in the left hand column, did not smoke during pregnancy, and a “1” if she did smoke. Some of the right hand column entries are “9” and that indicates the data entry team could not determine if the mother did or did not smoke.

So, how to get this data into R, short of copying and pasting it?

2.2 The read.table function

We will use an importing function called *read.table*.

Because *read.table* is a function, it takes arguments as input in order to produce an output. That is what functions do: they produce a definite output from an appropriate input. The three arguments we will need to give to *read.table* are:

1. The Web address of the data:
`http://www.stat.berkeley.edu/users/statlabs/data/babiesI.data`
2. An indication of whether or not the data contains a header row (it does): `header = TRUE`
3. What we want to use to separate entries in a row (we will use blank space): `sep = " "`

This means our *read.table* function command will look as follows:

```
read.table("http://www.stat.berkeley.edu/users/statlabs/data/babiesI.data", header  
= TRUE, sep = " ")
```

Enter this into R, following the > sign (where you see the flashing cursor):

```
> read.table("http://www.stat.berkeley.edu/users/  
statlabs/data/babiesI.data",header = TRUE,sep=" ")
```

Press ENTER, or RETURN, and you will see all 1236 rows of data (excluding the header) laid out in columns. To get a summary of the structure of this data, use the structure function, *str*, as follows:

```
> str(data)
```

which produces the following output:

```
'data.frame': 1236 obs. of 2 variables:  
 $ bwt : int  120  113  128  123  108  136  138  132  120  143 ...  
 $ smoke: int   0   0   1   0   1   0   0   0   0   1 ...
```

This tells us the data frame 'data' consists of 1236 observations of two variables: 'bwt' and 'smoke', each of which is an integer value (= whole number), and then lists the first 10 of the numeric values for each of these variables, in order.

2.3 Doing things with data

What might we want to do with this data?

Let's say we want a [histogram](#) of the babies weight for the non-smoking mothers. A histogram gives us a visual estimate of the probability distribution of the birth weights by showing the frequencies of birth weights occurring in different ranges.

To do this we need to *refer* to the data in R, and to do that we need to assign a name to the data. Here is how we do that:

```
> data<-read.table("http://www.stat.berkeley.edu/users/
statlabs/data/babiesI.data",header = TRUE,sep="")
```

Here is a hint to save re-typing the *read.table* function information again: simply hit the UP arrow on your keyboard until you see:

```
> read.table("http://www.stat.berkeley.edu/users/statlabs/data/ ba-
biesI.data", header = TRUE, sep = "")
```

appear again at the active line of R (where the cursor is blinking). Place your cursor just after the > sign and type "data < -". Then hit ENTER or RETURN.

The expression "data" is just our chosen name for the data imported into R.

What does the sign "< -" mean? It means that the data, as we have imported it into R, has been assigned a name "data", and a location in memory. We can recall the data from memory by typing:

```
> data
```

Try it, and you will see the data re-appear. Anytime we want to see our data, or do something to it, we can refer to it by its given name: "data". [Note we can use any name at all for the data - we could have called it "Rodney" if we wished.]

Through using the *read.table* function we have read the data into R in a format that is called a *data frame*. You can think of a data frame as consisting of a collection of columns, where each column contains the values of a variable (in our case, baby weights in the first column, and a "0" or "1" depending on whether or not the mother smoked during pregnancy, in the second column.)

2.4 Extracting a subset of a data frame

Let us extract the list of birth weights of babies of non-smoking mothers from the data frame and then produce a histogram of that non-smoking data.

First, we form a subset of the data frame consisting of only those rows containing "0" under the header "smoke":

```
> nosmoke <- subset(data, smoke == 0)
```

When you do this R seems to have done nothing, yet, in fact, it has: it has formed the subset of birth weights of babies of non-smoking mothers, and has assigned it, in memory, to a variable called "nosmoke". [Of course, you can use any name you want for this variable].

To see this subset of the original data frame, we type:

```
> nosmoke
```

and, lo and behold, we see a subset of our original data frame that begins:

	bwt	smoke
1	120	0
2	110	0
4	123	0
6	136	0
7	138	0
8	132	0
9	120	0
11	140	0
15	114	0
16	115	0
19	144	0
21	105	0
23	137	0

```
24  122    0
25  131    0
27  146    0
29  125    0
31  122    0
```

Again we can see the structure of this data frame by using the structure function *str*:

```
> str(nosmoke)
```

which has the output:

```
'data.frame': 742 obs. of 2 variables:
 bwt : int  120 113 123 136 138 132 120 140 114 115...
 smoke : int  0 0 0 0 0 0 0 0 0 0...
```

What do you notice about this output? Yes, you're right: only some of the original rows are included - those for which the "smoke" entry was 0. So we have the original rows numbered 1, 2, 4, 6, 7, 8, 9, 11, 15, 16, 19, 21, 23, 24, 25, 27, 29, 31, ... but not 3, 5, 10, 12, 13, 14, 17, 18, 20, 22, 26, 28, 30, ...

2.5 Creating a histogram

The 'nosmoke' data is still a data frame, and to plot a histogram of the birth weights of babies born to non-smoking mothers, we need to extract the birth weights as a *vector* of numeric values (and not as a whole data frame, because the histogram function in R only takes column vectors of numeric values as arguments). Think of a vector as an ordered list of things - in our case numeric values.

To create a vector of the birth weights of the babies of non-smoking mothers we type:

```
> nosmokedata <- nosmoke[['bwt']]
```

Note the double brackets (*double square brackets* for non-U.S. readers) that tell R to take the data in the 'bwt' column of the data frame 'nosmoke' and convert it to a column vector which we have called 'nosmokedata'. To see this data, type:

```
> nosmokedata
```

You will see a vector of numeric values - the birth weights of babies born to non-smoking mothers.

The structure of 'nosmokedata' shows us that this is simply a vector of numeric values:

```
> str(nosmokedata)
```

with output:

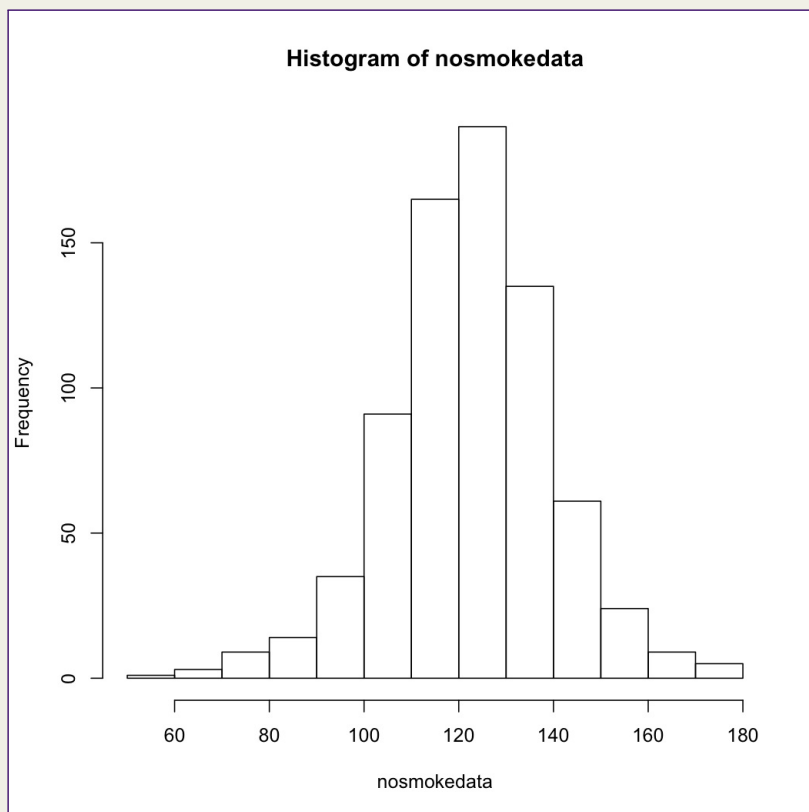
```
int [1:742] 120 113 123 136 138 132 120 140 114 115 ...
```

which tells us that 'nosmokedata' is a vector - an ordered list - of numeric values that are, in fact, integers (= whole numbers).

To see a histogram of this data, type:

```
> hist(nosmokedata)
```

A new window appears with a histogram of the birth weights:



A histogram gives a pictorial representation of the distribution of the data. If you are a little shaky on what a histogram does, or how one is constructed, see Chapter 2 on histograms.

2.6 Summary

2.6.1 What we did

How cool, and how easy was this! Starting from real data, situated on the web, we have:

1. Imported the data into R as a data frame, using the *read.table* function.
2. Created a new data frame consisting only of the weights of babies of mothers who did not smoke during pregnancy, using the *subset* function.
3. From that data frame, created a vector - an ordered list - of the numerical birth weights of the babies born to non-smoking mothers.
4. Used that vector to produce a histogram of the birth weights of babies of non-smoking mothers, using the *hist* function.

This is seeing R as a useful tool, and getting you rapidly and joyfully to a point where, with little effort, you can see R as a useful data analysis tool.

True, we are yet only taking first steps. However we avoided such silly exercises as:

```
> 1+1  
[1] 2
```

Why would we want to use a program as sophisticated as R to tell us that $1 + 1 = 2$? And this sort of childish exercise is not somehow getting us used to how R works. Let's face it: you came to this book because you have data to analyze and you want to take your first steps in using R to do that, and then rapidly become awesome at using R. Let's START with becoming awesome, and build on that!

In following chapters we will see how to produce spectacularly beautiful and informative histograms with ease and how to compare them on the same chart.

Before we do that we will look at other ways of getting data into R.

2.6.2 R functions and assignments

- The *read.table* function. This function reads data from a file and stores it in R as a data frame. In the example of this chapter we read the data from an internet file, and we used 3 arguments as input for this function:
 - (a) The URL of the data file.
 - (b) A logical value - TRUE or FALSE - for the header statement. In our case the data did have a header row, so we set *header=TRUE*.
 - (c) A field separator character that separates the entries in different columns, along a single row. The default is a blank space, and we explicitly specified this as *sep = ""*
- Assigning a name to data read into R. Typically this goes like *name - read.table(...)*. Here "name" is the name we choose for the imported data (read into R by the *read.table* function). We can see this data by typing *name* in the command line.

2.7 More ways of getting data into R

2.7.1 Files stored on your computer

Text files

Often you will want to import a text file stored on your computer into R. The procedure is almost identical to that for importing from a website - instead of specifying a Web URL, you simply have to tell R where the file is located on your computer.

To see what is the current directory R is using type the following command into the R console:

```
> getwd()
```

R responds with the current working directory.

The function *list.files()* will list all top level files in the working directory. These "files" may be folders - most likely, in fact, that's what they will be:

```
> list.files(" DIR")
```

provides a list of the top level "files" in the working directory. Here "DIR" is your working directory, which you know from using the function *getwd()*.

Suppose, as is often the case, that one of these "files" is the folder "Documents". To see the files in this folder, just tack /Documents to the end of the working directory information:

```
> list.files(" DIR/Documents")
```

This will provide a list of the files in the "Documents" folder. Suppose one of those documents is the text file "data.txt", which we want to read into R. We simply tell R the path to find this document, and specify, as in the previous chapter, whether or not there is a header, and what we want to have as column separators:

```
> read.table(" DIR/Documents/data.text", header=TRUE,  
sep = " ")
```

Spreadsheet files

2.8 Basic data cleaning

Commonly, government agencies put data on the web in - let's be honest - somewhat unusual formats. Reading oddly formatted data into R creates its own challenges. Let's look at an example.

At the U. S. [Bureau of Labor Statistics](#) there is [a page](#) giving unemployment rates for all major metropolitan areas for each month, over a range of years. The beginning of the data is laid out as follows:

Table 1. Civilian labor force and unemployment by metropolitan area, seasonally adjusted

LAUS Code	State FIPS Code	Area FIPS Code	Area	Year	Month	Civilian Labor Force	Employment	Unemployment	Unemployment Rate
MT011150	01	11500	Anniston-Oxford, AL MSA	2000	01	53,977	51,483	2,494	4.6
MT011222	01	12220	Auburn-Opelika, AL MSA	2000	01	50,731	56,657	2,074	3.5
MT011302	01	13020	Birmingham-Hoover, AL MSA	2000	01	532,631	514,172	18,459	3.5
MT011940	01	19400	Decatur, AL MSA	2000	01	73,261	70,443	2,818	3.8
MT012002	01	20020	Dothan, AL MSA	2000	01	65,723	63,200	2,523	3.8
MT012252	01	22520	Florence-Muscle Shoals, AL MSA	2000	01	70,208	66,953	3,255	4.6
MT012340	01	23400	Gadsden, AL MSA	2000	01	46,620	46,733	2,007	4.3

Let's try to import this data into R so we can analyze it in several ways. We presume that is why the Bureau of Labor Statistics made the data available.

R has difficulty importing this data for a number of reasons.

First, the descriptive explanation "Table 1. Civilian labor force and unemployment by metropolitan area, seasonally adjusted" at the top of the file, and a similar note:

: BLS, LAUS

March 22, 2013"

at the bottom, are not part of the data.

Second, the intended header "LAUS code" is spread out over 2 lines, apparently to be convenient for a human reader (otherwise the file would be too wide). As an aside, the word "code" is redundant for this header because ALL the headers are codes: the phrase "LAUS" would have sufficed.

These are relatively minor typographic matters that can be fixed by hand. If we fix them, for example by replacing all headers as single words on a single line, with the same spacing as the first row of the data, we will get the following error message in trying to load the data into R:

Error in read.table("/Users.....txt", : more columns than column names

There are serious issues with the layout of column 4, labelled "Area", and the separation of the columns. The columns have been separated by someone hitting the space bar several times, and not always the same number of times, even for a particular column. The reason someone does this is to align the data on a page so it looks readable to a person. We can see this in the separation between the "Area" column and the "Year" column: spaces have been inserted so as to have all the years line up vertically on the page, for ease of human readability.

Column 4, "Area" introduces special problems. Because someone entering the text wanted it to look nice, they have inserted spaces between words. For example, in the first data row the "Area" column entry is:

Anninston-Oxford, AL MSA

with a space between ", " and "AL" and a space between "AL" and "MSA".

Because R is interpreting spaces as column separators, it interprets the first row as having 12 columns, instead of 10. If that were consistent across all rows we could handle it, but that's not what happens - the spacing is not consistent throughout the "Area" column. And if we try to fix it by hand we get into the headache-inducing situation of having to format 59,660 rows! There has to be a better way, and there is.

We can easily fix this problem by first saving the Web data as a text file and then importing it into Excel as a text file. Once in Excel we can clean up the headers and the message at the bottom of the file, and save as a .csv (= comma separated values) file. This we can import into R using the *read.table* function.

Here's how:

- Under "File" in your browser choose "Save Page As..." or "Save As...". You get a suggested name with the extension .txt, or you have the option of saving as "Page Source". Both of these options will save the data as a text file.
- Open a blank document in Excel and under "File" choose "Import". You will see a set of choices that includes importing as a text file. Check that option, locate the file and open it in Excel.
- Clean up your headers and the bottom message in the file and you are almost done.

- You have the choice of saving the file as a Comma Separated Values (.csv) file or as a Tab Delimited Text (.txt) file. Either choice will work.
- Now all that's left to do is to import the file into R.

Part II

Descriptive statistics

Chapter 3

The purpose of descriptive statistics

3.1 Why we do descriptive statistics

The point of descriptive statistics is to describe data sets in a variety of useful ways that add to our understanding of how the data is distributed.

Techniques of descriptive statistics should be the first things that are used on any and every data set, to get a feel for what the data looks like:

- Is the data highly skewed or not?
- Is the data distribution very peaky around the mean?
- Are there long tails, with many data points a long way from the mean?
- Is the data distribution [unimodal](#), [bimodal](#) or [multimodal](#), or something else altogether?
- Is the data approximately normally distributed?
- Do two - or more - related data sets have approximately the same distribution?

3.2 The normal distribution

3.2.1 Testing for normality

Quantile plots

Anderson-Darling test for normality

Pre-trest & post-test: the differences between the "before" and "after" data have to be normal.

Chapter 4

Histograms

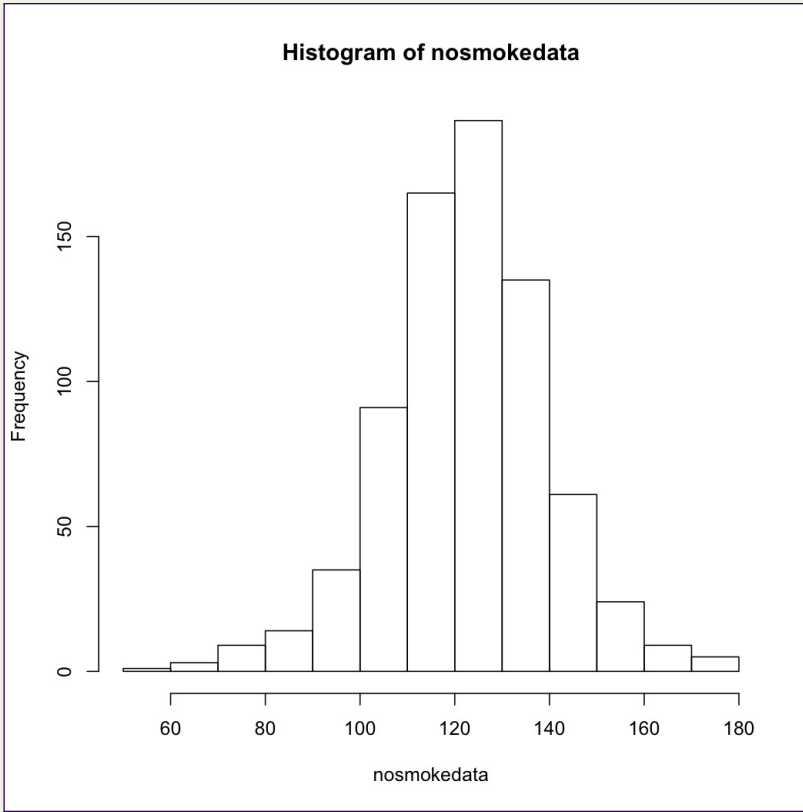
4.1 The basics

A histogram graphically represents data: we have seen an example of this in 1.3.2, where we used R to create a histogram of the birthweight data for babies of non-smoking mothers. In that section we assumed you already knew about histograms. Here, for the record, we will go into histogram construction in a little more detail.

Histograms were introduced by the statistician [Karl Pearson](#) in 1895, in a paper on the mathematical theory of evolution:

Pearson, K. (1895). [Contributions to the mathematical theory of evolution. II. Skew variation in homogeneous material.](#) *Philosophical Transactions of the Royal Society of London*. A, 343-414.

The idea of a histogram is to show the frequency of occurrence of data points in specified “data bins”. For example, for the birthweight data for babies of non-smoking mothers, we know that data ranges from 55 ounces through 176 ounces. If we divide that range of $176 - 55 = 121$ ounces into 13 bins: [50,59], [60,69], [70,79], [80,89], [90,99], [100,109], [110,119], [120,129], [130,139], [140,149], [150,159], [160,169], [170,179], then each data point lies in exactly one of these bins. Counting the frequency of data in each bin gives us a histogram:



The bins are made so as to be non-overlapping at the end points: this means there is no ambiguity as to which bin a data point belongs.

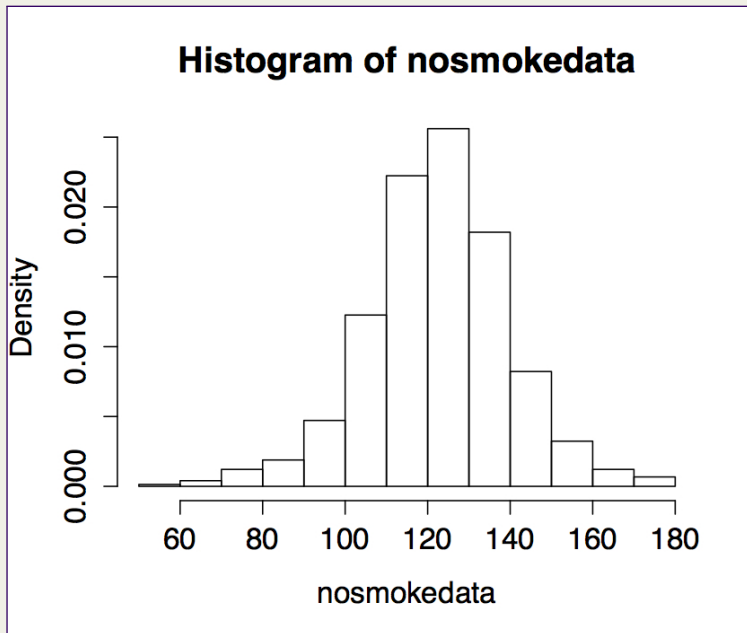
The bins are also chosen to be all the same size - that is, width. But why this number of bins? Would the histogram look much different if we chose a different number of bins? The idea of the histogram is to give a visual representation of the distribution of the data. This means that we are attempting to visually approximate the empirical probability density function for the data. We will come back to the question of the number of bins shortly and, for now, leave it to R to determine the number of bins: we just note that histograms can be poor visual representations of the empirical probability distribution because their shape depends critically on the number of bins used.

What we notice about the vertical axis of the histogram is that it records

the total number of data points in each bin. This means that the histogram, as drawn, is **not** a representation of the empirical probability density function, because the *total area* under the bars of the histogram is not 1. We can rectify this by using the following R command:

```
> hist(nosmokedata, probability=TRUE)
```

which outputs the following graphic:



Notice the change in the vertical axis: we now have the *relative frequency* of occurrence, or *density*, of data points in the bins, so we can interpret the vertical axis as the probability that a data point will lie in that bin.

Why do we care whether the vertical axis of our histogram shows frequencies or densities? Under what circumstances would it matter? The shape is still the same after all.

We answer this question in the next section when we look at how R can approximate the probability density function by a smooth curve: this is especially helpful in:

- remedying to a considerable extent the issue of choosing a bin size for the histogram;
- assisting in comparison of one histogram with another.

4.2 Comparing two, or more, histograms

4.2.1 Overlaying histograms

Often we want to look at the distributions of two sets of data to see how they might differ. For example, we might want to compare the distributions of birthweights of babies for smoking and non-smoking mothers. So what is useful is to get R to plot both histograms on the one plot, so that we can compare their shapes and differences.

To do this for the data for birthweights of non-smoking and smoking mothers we first extract the birthweight data for the smoking mothers as we did for the non-smoking mothers:

```
> smoke <- subset(data, smoke == 0)
> smokedata <- smoke[["bwt"]]
```

We then *name* the histogram plots of both data (because we will, in a moment, call these names in a new plot command):

```
> p1 <- hist(nosmokedata)
> p2 <- hist(smokedata)
```

Now we will plot both histograms together, on the one plot. But in order to do that we will:

- ensure that both histograms are plotted over the same range
- color the histograms in different colors
- make one of the histograms transparent, so we can see both histograms on the same plot.
- remove all labels from the histogram plots
- add in a new set of labels at the end.

One of the options of the *plot* command is `xlim=c(a,b)`, where “a” and “b” are limits on the data ranges that includes the min and max of both data sets (so we can see all of both histograms on the same scale). We make “a” the smaller of the minimum of the nosmoke data and the smoke data, and similarly we make “b” the larger of the maximum of the nosmoke data and the smoke data:

```
> a <- min(min(nosmokedata), min(smokedata))
> b <- -max(max(nosmokedata), max(smokedata))
```

Now we can plot the two histograms one over the other, with the first light blue, and the second light red and transparent - the "`add=T`" option - to allow the first histogram to show through:

```
> plot( p1, col=rgb(0,0,1,1/4), main="", sub = "", xlab="",
xlim=c(a,b)) # first histogram
```

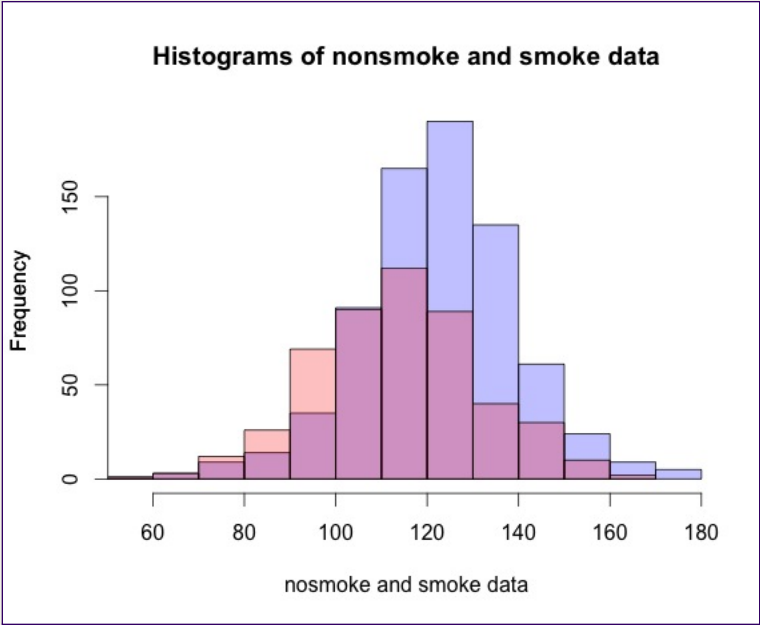
```
> plot( p2, col=rgb(1,0,0,1/4), main="", sub="", xlab="",
xlim=c(a,b), add=T) # second histogram
```

```
< -title(main= "Histograms of nonsmoke and smoke data",
xlab= "nosmoke and smoke data", ylab="Frequency")
```

The "`main`", "`sub`" and "`xlab`" are set to empty in the two histogram plots so that there is no main title, sub-title, or x-axis label on either of those plots. Then the "`label`" command adds in the labels we want on the joint plots.

The `# first histogram`, `# second histogram` statements are just non-functional comments to remind us which histogram is which.

This gives the following output, with the overlap colored light purple:



4.2.2 Back to back histograms

A function `histbackback()`, in the package `Hmisc`, allows us to plot two histograms back to back.

4.3 Smooth histograms

The term “smooth histogram” is something of a misnomer: what we are really going to do is to find a smooth approximation to the histogram of a given data set. The background to constructing a smooth histogram is a little complicated: it involves critters known as *kernel density estimations*. However, the implementation in R is very simple.

Chapter 5

Summary statistics

5.1 Five-number summary

5.1.1 A single numeric data set

A five-number summary of a list of numeric data values gives:

- The [median](#) of the data
- The upper and lower [quartiles](#) of the data
- The minimum value of the data
- The maximum value of the data

The *median* of a numerical data set is a number m such that the number of data points less than m is the same as the number of data points greater than m .

For an odd number of data points, the median is chosen to be the middle data point, while for an even number of data points, the median is chosen to be the average of the two data points nearest the middle of the data.

The *lower quartile* splits the data below the median in the same way as the median splits the whole data set, and the *upper quartile* splits the data above the median in the same way.

A more formal definition of the median of a set of numerical data (x_1, \dots, x_n) is that it is the number \tilde{x} that minimizes

$$|x_1 - c| + \dots + |x_n - c|$$

as a function of c . In other words, for all c we have:

$$|x_1 - \tilde{x}| + \dots + |x_n - \tilde{x}| \leq |x_1 - c| + \dots + |x_n - c|$$

The five-number summary is implemented in R as follows:

```
> summary(data)
```

where “data” is the data of numerical values. For example, for the birth-weight data for the non-smoking mothers we have:

```
> summary(nosmokedata)
```

with output:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
55	113	123	123	134	176

Note that the “five-number summary” contains the mean as well as the other information (so it’s really a six-number summary!)

5.1.2 Two - or more - numeric data sets

When we have two data vectors - such as the baby birthweights for non-smoking and smoking mothers - it is helpful to see their five-number summaries side by side. To achieve this we can make a *data frame* by combining the two data vectors.

However ... when the data vectors are of different lengths - as they are in the case of the baby birthweights for non-smoking and smoking mothers - we need something extra to combine these data vectors. That something extra is the `rbind.fill()` command from Hadley Wickham’s *plyer package*. The name “plyer” is pronounced “PLIER” as in the following tool (a pair of pliers):



That is because, as Hadley Wickham writes:

“plyr is a set of tools for a common set of problems: you need to split up a big data structure into homogeneous pieces, apply a function to each piece and then combine all the results back together.”

So we need some “pliers” to join data vectors of different lengths!

To get these tools we need to install the plyr package:

```
> install.packages('plyr')
```

Now we can load the `plyr` package using the `library()` command, and join the non-smoking and smoking data vectors in a single data frame by using the `rbind.fill()` command:

```
> library(plyr)
> combined<-rbind.fill(nosmokedata,smokedata)
```

This has the effect of creating a data frame with two columns, each of which is as long as both data vectors combined. The first column contains all the non-smoking data, followed by a string of NA. The second column contains a string of NA as long as the non-smoking data, followed by the smoking data. We can see the beginning of this by looking at the structure of the data frame:

```
> str(combined)
```

with output:

```
'data.frame': 1226 obs. of 2 variables:
 $ nosmokedata: int 120 113 123 136 138 132 120 140 114 115 ...
 $ smokedata : int NA NA NA NA NA NA NA NA NA NA NA ...
```

Now we can apply the `summary()` command to this data frame to get a side by side summary of the non-smoking and smoking data:

```
> summary(combined)
```

nosmokedata		smokedata	
Min.:	55	Min.:	58.0
1st Qu.:	113	1st Qu.:	102.0
Median:	123	Median:	115.0
Mean:	123	Mean:	114.1
3rd Qu.:	134	3rd Qu.:	126.0
Max.:	176	Max.:	163.0
NA's:	484	NA's:	742

5.1.3 The `fivenum()` command

A five-number summary can also be calculated using the following R command:

```
> fivenum(data)
```

with output:

```
[1] 55 113 123 134 176
```

Note that this is indeed a five-number summary, but unlike the `summary()` command `fivenum()` does not include headers for the numbers.

Occasionally, these two different commands for calculating five-number summaries will give slightly different results. The reason is the different ways that the two commands calculate the 1st and 3rd quartiles (for which there is no entirely agreed-upon definition by data analysts). For a fuller discussion of this issue see:

Exploratory Data Analysis: The 5-Number Summary - Two Different Methods in R by Eric Cai.

The five-number summary was first used by John Wilder Tukey:

Tukey, J. W. (1977). *Exploratory data analysis*. Reading, MA.

so it is often called *Tukey's five-number summary*.

Prior to Tukey, Arthur Lyon Bowley used a seven number summary, consisting of the `deciles` in addition to the median, upper and lower quartiles, and the minimum and maximum:

Bowley, A. L. (1915). *An elementary manual of statistics*. PS King son, Limited.

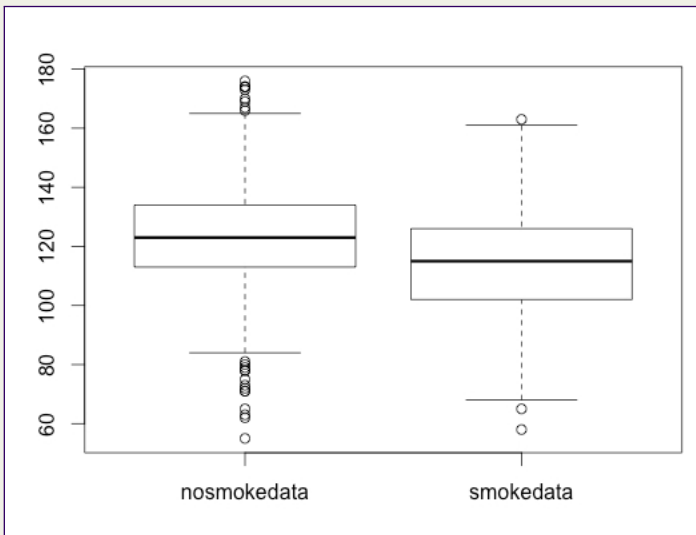
5.2 Box plots

Box plots give more information than a five-number summary, but less information than a histogram. In a box plot, the bottom of the box indicates the lower quartile, the top of the box indicates the upper quartile, and the median is shown by a line drawn through the box;

The R command `boxplot()` will create a simple boxplot graphic from a data vector or a data frame. We apply it to the combined non-smoking/smoking data frame:

```
> boxplot(combined)
```

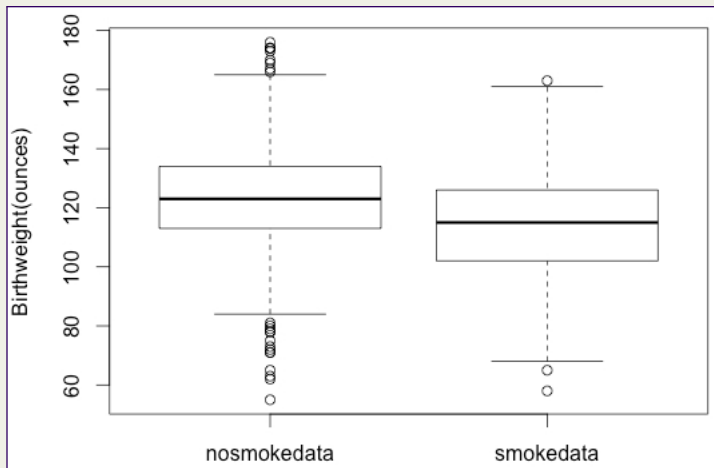
to get the output:



This is as fine as a simple black and white box plot with no labeling on the vertical axis. To include a label simply use the "ylab" option in `boxplot()`:

```
> boxplot(combined, ylab="Birthweight (ounces) ")
```

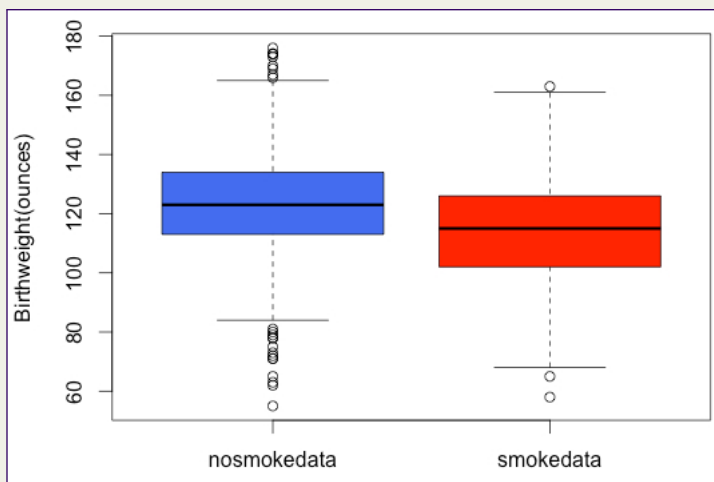
to get the following output with the label on the vertical axis:



We can add color to our boxplots by using the "col" option in `boxplot()`:

```
> boxplot(combined, ylab="Birthweight(ounces)",  
col=c("royalblue2","red"))
```

which gives us the colored boxplots below:



Earl F. Glynn of the Stowers Institute for Medical Research posted [a PDF of color names](#) and RGB values for colors in R (24 May 2005). It's a very useful tool to use when experimenting with color in plots.

24	black	#000000	0	0	0
25	blanchedalmond	#FFEB3D	255	235	205
26	blue	#0000FF	0	0	255
27	blue1	#0000FF	0	0	255
28	blue2	#0000EE	0	0	238
29	blue3	#0000CD	0	0	205
30	blue4	#00008B	0	0	139
31	blueviolet	#8A2BE2	138	43	226
32	brown	#A52A2A	165	42	42
33	brown1	#FF4040	255	64	64
34	brown2	#E33B3B	238	59	59
35	brown3	#CD3333	205	51	51
36	brown4	#8B2323	139	35	35
37	burlywood	#DEB887	222	184	135
38	burlywood1	#FFD39B	255	211	155
39	burlywood2	#EEC951	238	197	145
40	burlywood3	#CDA47D	205	170	125
41	burlywood4	#8B7355	139	115	85
42	cadetblue	#5F9EA0	95	158	160
43	cadetblue1	#98F5FF	152	245	255
44	cadetblue2	#8EE5EE	142	229	238
45	cadetblue3	#7AC5CD	122	197	205
46	cadetblue4	#53868B	83	134	139
47	chartreuse	#7FFF00	127	255	0
48	chartreuse1	#7FFF00	127	255	0
49	chartreuse2	#76EE00	118	238	0
50	chartreuse3	#66CD00	102	205	0
74	darkcyan	#008B8B	0	139	139
75	darkgoldenrod	#8B860B	184	134	11
76	darkgoldenrod1	#FFB90F	255	185	15
77	darkgoldenrod2	#EAD0E	238	173	14
78	darkgoldenrod3	#CD950C	205	149	12
79	darkgoldenrod4	#8B6508	139	101	8
80	darkgray	#A9A9A9	169	169	169
81	darkgreen	#006400	0	100	0
82	darkgrey	#A9A9A9	169	169	169
83	darkkhaki	#BDB76B	189	183	107
84	darkmagenta	#8B008B	139	0	139
85	darkolivegreen	#556B2F	85	107	47
86	darkolivegreen1	#CAFF70	202	255	112
87	darkolivegreen2	#BCEE68	188	238	104
88	darkolivegreen3	#A2CD5A	162	205	90
89	darkolivegreen4	#6E8B3D	110	139	61
90	darkorange	#FF8C00	255	140	0
91	darkorange1	#FF7F00	255	127	0
92	darkorange2	#EE7600	238	118	0
93	darkorange3	#CD6600	205	102	0
94	darkorange4	#8B4500	139	69	0
95	darkorchid	#9932CC	153	50	204
96	darkorchid1	#BF3EFF	191	62	255
97	darkorchid2	#B23AEE	178	58	238
98	darkorchid3	#9A32CD	154	50	205
99	darkorchid4	#68228B	104	34	139
100	darkred	#8B0000	139	0	0

You can find more information on creating differing styles of box plots at [Quick R](#) and [R-bloggers](#).

5.3 Violin plots

Chapter 6

Standard scores & their applications

6.1 Mean & standard deviation

6.1.1 Mean

6.1.2 Standard deviation

The *standard deviation* σ of a data set $X:=\{x_1, \dots, x_n\}$ is the square root of the average value of $(X - \mu)^2$:

$$\sigma = \sqrt{\mathbb{E}((X - \mu)^2)} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

Note: many statisticians will make a distinction between a sample, X , from a larger population, and the case when the data X *is* the entire population. In the former case they will replace the factor $\frac{1}{n}$ with $\frac{1}{n-1}$. This has to do with the somewhat esoteric notion of [degrees of freedom](#). For a data analyst, particularly those for whom the data sets are large (i.e. n is a large number), the difference is not particularly important. This issue has been discussed by some of the best applied statisticians on the planet (see, for example, [Terence Speed, IMS Bulletin](#)) and the

sensible approach seems to be: it doesn't much matter! As a result we will always calculate the standard deviation (sample or whole population) from the definition in the highlighted box above.

6.2 Standard scores

The *standard score* (also known as the *z-score*) of a data point x_i from a data set $X := \{x_1, \dots, x_n\}$ is $z := \frac{x_i - \mu}{\sigma}$.

Standard scores are a useful way to rank data when it is important for us to see how far individual data points are from the mean (as measured in units of standard deviation). For example,...

We can calculate the standard scores for each of the points in this data set using the `scale()` function:

6.2.1 The average value of z & z^2

Given a data set $X = \{x_1, \dots, x_n\}$, the average value $\mathbb{E}(z)$ of the standard scores $\frac{x_i - \mu}{\sigma}$ is 0:

$$\mathbb{E}(z) = \mathbb{E}\left(\frac{X - \mu}{\sigma}\right) = \frac{\mathbb{E}(X) - \mu}{\sigma} = \frac{\mu - \mu}{\sigma} = 0$$

and, similarly, the average value of z^2 is 1:

$$\mathbb{E}(z^2) = \mathbb{E}\left(\frac{(X - \mu)^2}{\sigma^2}\right) = \frac{\mathbb{E}((X - \mu)^2)}{\sigma^2} = \frac{\sigma^2}{\sigma^2} = 1$$

We might expect that the average values of z^3, z^4, \dots are equally uninspiring. However, as we shall see in the next two sections, $\mathbb{E}(z^3)$ and $\mathbb{E}(z^4)$ tell us a lot about the shape of the distribution of a data set.

6.3 Skewness

The average value $\mathbb{E}(z^3)$ of the cube z^3 of the standard scores $z = \frac{x_i - \mu}{\sigma}$ is called the *skewness* of the data set $X = \{x_1, \dots, x_n\}$.

The skewness, as we shall see, is a measure of the symmetry, or lack of it, of the distribution of the data. It is not the only such measure of symmetry, but it is used widely.

To get R to calculate the skewness of a data vector we need to install the "moments" package:

```
> install.packages('moments')
```

With the moments package loaded, we can calculate the skewness of our non-smoking data, for example:

```
> library(moments)
> skewness(nosmokedata)
```

with output:

```
nosmokedata
-0.1869841
```

6.3.1 What is skewness telling us?

How can we see that skewness is related to the symmetry of data?

The skewness of a data set $X = \{x_1, \dots, x_n\}$ is the average value $\mathbb{E}(z^3)$ of the cube z^3 of the standard scores $z = \frac{x_i - \mu}{\sigma}$. What do the values of z^3 have to do with symmetry?

The standard score (= z-score), $z = \frac{x_i - \mu}{\sigma}$, of a data point x_i tells us how far away x_i is from the mean, μ , of the data, as measured in units of standard deviation, σ .

In the table below the negative z-scores are colored yellow and the positive z-scores are colored blue.

z	z^3	<i>Relation of z^3 to z</i>
-2	-8	$\leftarrow z^3$ more negative than z
-1.75	-5.35938	$\leftarrow z^3$ more negative than z
-1.5	-3.375	$\leftarrow z^3$ more negative than z
-1.25	-1.95313	$\leftarrow z^3$ more negative than z
-1	-1	$\leftarrow z^3 = z$
-0.75	-0.421875	$\leftarrow z^3$ less negative than z
-0.5	-0.125	$\leftarrow z^3$ less negative than z
-0.25	-0.015625	$\leftarrow z^3$ less negative than z
0	0	$\leftarrow z^3 = z$
0.25	0.015625	$\leftarrow z^3$ less positive than z
0.5	0.125	$\leftarrow z^3$ less positive than z
0.75	0.421875	$\leftarrow z^3$ less positive than z
1	1	$\leftarrow z^3 = z$
1.25	1.95313	$\leftarrow z^3$ more positive than z
1.5	3.375	$\leftarrow z^3$ more positive than z
1.75	5.35938	$\leftarrow z^3$ more positive than z
2	8	$\leftarrow z^3$ more positive than z

Note that the z-scores that lie between -1 and 1 give rise to cubes that are much smaller in size.

As the z-score gets much bigger than 1, or much less than -1, so the cube gets very much more positive or negative respectively.

So, if the data is distributed asymmetrically about the mean then the skewness, as the average of the z^3 values, will be either positive or negative, the more so as the distribution is more skewed one way or the other.

6.4 Kurtosis

The average value $\mathbb{E}(z^4)$ of the 4th power z^4 of the standard scores $z = \frac{x_i - \mu}{\sigma}$ is called the *kurtosis* of the data set $X = \{x_1, \dots, x_n\}$.

With the moments package loaded, we can calculate the kurtosis of our non-smoking data, for example:

```
> library(moments)
> kurtosis(nosmokedata)
```

with output:

```
nosmokedata
4.03706
```

6.4.1 Kurtosis of normally distributed data

6.4.2 What is kurtosis telling us?

Chapter 7

Distance from the mean

7.1 Empirical estimates

7.1.1 The standard score function

For a given data set $X = \{x_1, \dots, x_n\}$, with mean μ and standard deviation σ , we can compute and tabulate, or plot, what fraction of the data lies within k standard deviations from the mean. we do this by utilizing the following function of k :

$$\Phi(k) := \frac{\#\{x_i : |x_i - \mu| \leq k\sigma\}}{n}$$

We call Φ the *standard-score function* (also referred to as the *z-score function*). This is because, for each $k \geq 0$, $\Phi(k)$ tells us what proportion of data points have the *size* of their standard-score (= z-score) $\leq k$.

7.1.2 Implementation in R

We will see how to implement the standard score function Φ in R.

To compute $\Phi(k)$ we need to know both the data and k . So, strictly speaking, Φ is a function of *two* variables: the *data* and k .

$\Phi \leftarrow$ -function(data,k)

7.1.3 Examples

Here are some examples of what the standard score function looks like for different data sets:

7.2 Theoretical inequalities & their application

We discuss Chebyshev's inequality, and the Vysochanskij-Petunin inequality which is a refinement of Chebyshev's inequality for data with only one mode (otherwise known as *unimodal data*). It is more or less obligatory to mention these inequalities when discussing exploratory data analysis, because they give some broad estimates of the proportion of the data within a given number of standard deviations from the mean. However these inequalities are probably of more theoretical than practical usefulness: we discuss them here in the context of actual data sets so you can get a feel for what the inequalities say in practice.

7.2.1 Chebyshev's inequality

We will phrase Chebyshev's inequality for the distribution of a data set $X = \{x_1, \dots, x_n\}$ (although it can, and probably should, be stated as a more general probability statement about random variables)

Let μ and σ denote the mean and standard deviation of the data set $X = \{x_1, \dots, x_n\}$. Then, for all real numbers $k \geq 1$:

$$\frac{\#\{x_i : |x_i - \mu| \geq k\sigma\}}{n} \leq \frac{1}{k^2}$$

When $k = \sqrt{2}$, for example, Chebyshev's inequality states that

$$\frac{\#\{x_i : |x_i - \mu| \geq \sqrt{2}\sigma\}}{n} \leq \frac{1}{2}$$

That is, no more than half the data points can be more than $\sqrt{2} \approx 1.414$ standard deviations from the mean.

When $k = 4.5$, Chebyshev's inequality states that

$$\frac{\#\{x_i : |x_i - \mu| < 4.5\sigma\}}{n} \geq 1 - \frac{1}{4.5^2} \approx 0.95$$

which says that approximately 95% of the data points are within 4.5 standard deviations of the mean, no matter what the distribution of the data.

Chebyshev's inequality applies to any data set whatsoever, but is generally more informative for highly skewed data. An example of a highly skewed data set is the collection of large Danish fire insurance claims, January 3, 1980 through December 31, 1990. This data is available - as a data vector - in the R package [evir](#).



We install 'evir', load the library, and examine the structure of the data named 'danish':

```
> install.packages('evir')  
> library(evir)  
> str(danish)
```

with output:

```
atomic [1:2167] 1.68 2.09 1.73 1.78 4.61 ...
- attr(*, "times")= POSIXt[1:2167], format: "1980-01-02 19:00:00"
"1980-01-03 19:00:00" "1980-01-04 19:00:00" ...
```

The phrase “atomic [1:2167]” means simply that this is a data vector (otherwise known as an atomic vector, as distinct from a data-frame). The second line refers to the fact that the data was collected in time and that those times are also recorded, along with the data. This does *not* mean that the data isn’t a simple data vector whose descriptive statistics we can compute:

```
> summary(danish)
```

with output:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	1.321	1.778	3.385	2.967	263.300

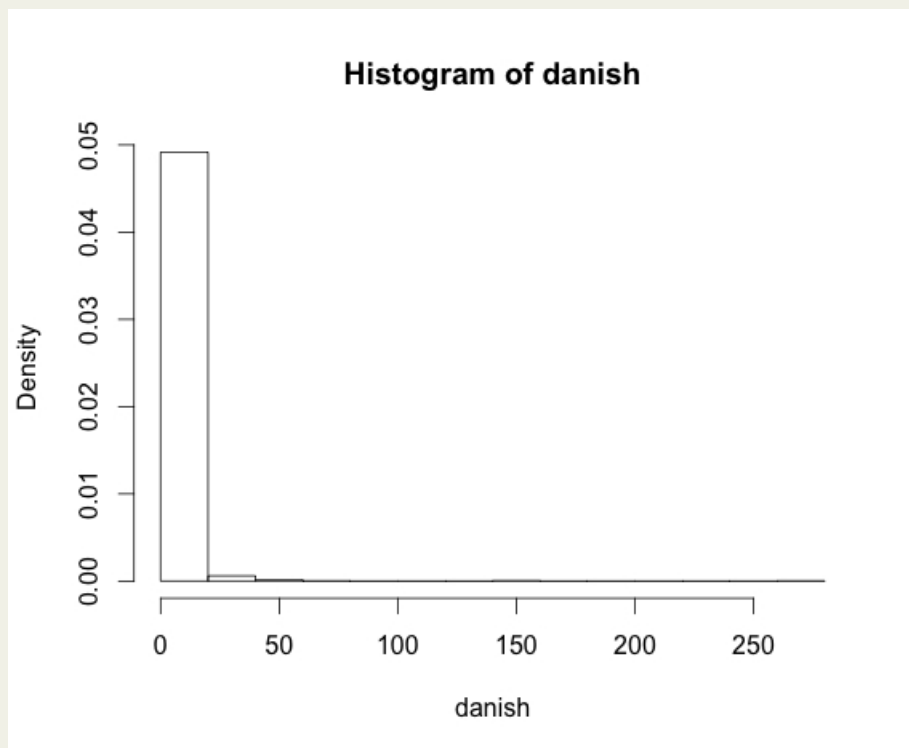
and

```
> mean(danish); sd(danish); skewness(danish); kurtosis(danish)
```

with output:

```
[1] 3.385088
[1] 8.507452
[1] 18.73685
[1] 482.198
attr(,"method")
[1] "excess"
```

This gives a mean $\mu = 3.385088$, a standard deviation $\sigma = 8.507452$, a very high skewness of 18.73685, and a whopping kurtosis of $482.198 + 3 = 485.198$. The reason for the “+3” in the kurtosis calculation is that R is telling us it has used a method that calculates the *excess* kurtosis: this is the kurtosis above and beyond the value of 3 for the normal distribution. This extraordinarily high value for the kurtosis is not entirely



unexpected, because the maximum data point is over 30 standard deviations from the mean, as a histogram of the data indicates visually. The histogram also shows clearly the high skewness of the data distribution.

What does Chebyshev's inequality tell us about this data?

We know, from above, that approximately 95% of the data points lie within 4.5 standard deviations of the mean. This tells us in particular, that approximately 95% of the data points are less than

$$\mu + 4.5\sigma = 3.385088 + 4.5 \times 8.507452 = 41.6686 < 42$$

which is substantially less than the maximum value of the data at 482.198. Chebyshev's inequality tells us that even though there are quite high values for the data, because of the high skewness of the distribution, 95% of the data points are less than 42.

A similar calculation tells us that 99% of the data points are within 10 standard deviations from the mean, and so are less than 89:

$$\mu + 10\sigma = 3.385088 + 10 \times 8.507452 = 88.4596 < 89$$

7.2.2 Graphical interpretation of Chebyshev's inequality

We can graphically interpret Chebyshev's inequality in terms of the function

$$\Phi(k) := \frac{\#\{x_i : |x_i - \mu| \leq k\sigma\}}{n}$$

which returns the proportion of the data set lying within k standard deviations of the mean. Chebyshev's inequality says that

$$\Phi(k) \geq 1 - \frac{1}{k^2}$$

which, of course, is only informative for $k \geq 1$.

7.2.3 The Vysochanskij-Petunin inequality

Application to unimodal data

The Vysochanskij-Petunin inequality is a substantive improvement on the Chebyshev inequality for data that has a *unimodal* distribution. “Unimodal” means that the distribution - the empirical probability function - has only one local maximum.

The Vysochanskij-Petunin inequality for a *unimodal* discrete data set $X = \{x_1, \dots, x_n\}$ with mean μ and standard deviation σ , says that, for all real numbers $k > \sqrt{8/3} \approx 1.633$:

$$\frac{\#\{x_i : |x_i - \mu| \geq k\sigma\}}{n} \leq \frac{4}{9k^2}$$

Equivalently, for all real numbers $k > \sqrt{8/3} \approx 1.633$:

$$\Phi(k) := \frac{\#\{x_i : |x_i - \mu| < k\sigma\}}{n} > 1 - \frac{4}{9k^2}$$

When $k = 3$ we have $1 - \frac{4}{9k^2} \approx 0.95$ so the Vysochanskij-Petunin inequality tells us that for unimodal data, 95% of the data is within 3 standard deviations of the mean.

Similarly, when $k = 7$ we have $1 - \frac{4}{9k^2} \approx 0.99$ so for unimodal data, 99% of the data is within 7 standard deviations of the mean.

This is so even if the data is highly skewed, so long as it is unimodal. For the Danish fire insurance claim data, which is unimodal, the Vysochanskij-Petunin inequality assures us that 99% of the data is less than 63:

$$\mu + 7\sigma = 3.385088 + 7 \times 8.507452 = 62.9373 < 63$$

which is a substantial improvement on the estimate obtained from Chebyshev's inequality.

Testing for unimodality

Unimodality is easy to determine for a continuous distribution: the graph of the probability density function will, for some point x_0 , be ... for $x \leq x_0$ and

... for $x \geq x_0$:

However, for a discrete data set, which is what we get in practice, the question of determining unimodality is not so straightforward.

(1) The dip test

The *dip test* for unimodality of a discrete data set was formulated by J.A. & P.M. Hartigan in 1985:

Hartigan, J.A. & Hartigan, P.M.. (1985). *The dip test for unimodality*. *The Annals of Statistics*, 13(1), 70-84.

The dip test is implemented in R through the 'dipTest' package, which we load into R:

```
>install.packages(dipTest)
```

The 'dipTest' package utilizes a function *dip()* that produces a dip statistic: a number in the range $[0, 1]$. According to the [Wikipedia article](#) on the dip statistic:

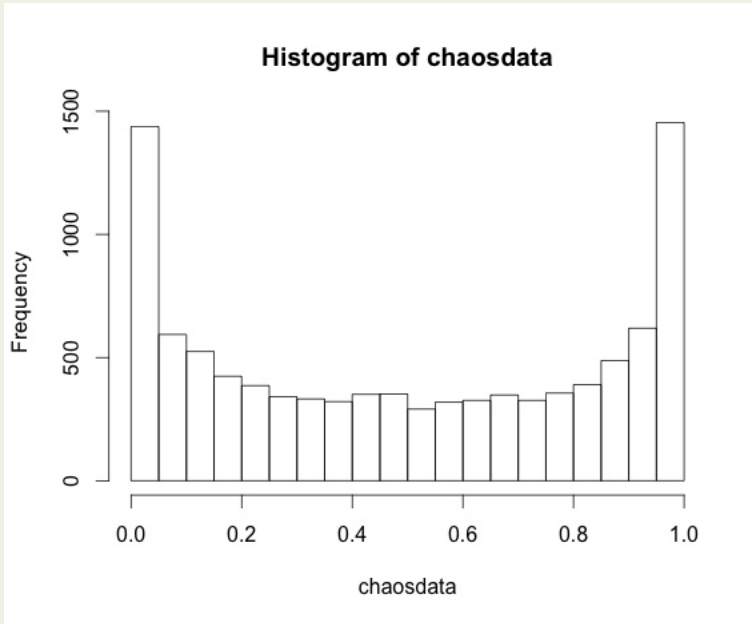
“Values less than 0.05 indicate significant bimodality and values greater than 0.05 but less than 0.10 suggest bimodality with marginal significance.”

The dip test is *not* especially good at detecting obvious bi-modality when the modes occur toward the minimum and maximum of the data set.

For example, the function $Q(x) := 4x(1 - x)$ is known to be [chaotic](#) on the interval $[0, 1]$, so if we begin with a point such as 0.1 and apply Q to get $Q(0.1) = 0.36$, then take this output of 0.36 and apply Q to get $Q(0.36) = 0.9216$, and so on, over and over, 10,000 times, we get a set of points in the interval $[0, 1]$ that bunches up more toward 0 and 1 than it does in the middle:

Technically, these points follow a $\beta(1/2, 1/2)$ distribution (= [arcsine distribution](#)):

Jakobson, M.V. (1981) [Absolutely continuous invariant measures for one parameter families of one-dimensional map](#). *Communications in Mathematical Physics*, 81, 39-88.



The data is clearly bi-modal with distinct modes near 0 and 1, yet the dip test returns a value of approximately 0.063, which is *above* the recommended value of 0.05 at which we convincingly reject unimodality:

```
>library(diptest)
>diptest(chaosdata)
```

with output:

```
[1] 0.06293824
```

The dip test, in other words, is not telling us to reject unimodality, though it is a close call and the fact that the dip statistic is below 0.1 gives us reason to be suspicious that there is more than one mode. The data, on the other hand, patently - and provably - has a bi-modal distribution.

(2) Smooth Histograms

A smooth histogram (= kernel density estimate) of the chaos data indicates two clear modes, one near 0 and the other near 1:

A technique for using smooth histograms to detect multiple modes in a discrete data distribution was introduced by B. W. Silverman in 1980:

Silverman, B. W. (1981). [Using kernel density estimates to investigate multimodality](#). Journal of the Royal Statistical Society. Series B (Methodological), 97-99.

7.2.4 The Laguerre-Samuelson inequality

The Laguerre-Samuelson inequality applies to the distribution of a data set $X = \{x_1, \dots, x_n\}$ with mean μ and standard deviation σ . It says, simply, that:

$$|x_i - \mu| \leq \sqrt{n-1}\sigma \text{ for all } 1 \leq i \leq n$$

In other words, all the data points x_1, \dots, x_n lie within $\sqrt{n-1}$ standard deviations of the mean.

Let's examine this inequality for the non-smoking data we have previously used. Recall from the 5-number summary of that data that the mean is 123 ounces and the standard deviation is 17.4 ounces. There are 742 data points and $\sqrt{742-1} \approx 27.2$, so the Laguerre-Samuelson inequality says that every data point lies within $27.2 \times 17.4 = 473.6$ of the mean 123. In other words, since we know all the data points are positive - in fact all are 56 ounces or greater - all the data is no greater than $123 + 473.6 = 596.6$. This is a grossly inaccurate estimate, because the maximum value of the non-smoking data is 176.

So how, you may well ask, is the Laguerre-Samuelson inequality useful?

As for Chebyshev's inequality, the Laguerre-Samuelson inequality is more useful for *highly skewed* distributions - those data distributions that are highly assymmetric in relation to the mean of the data.

The Laguerre-Samuelson inequality tells us that for the Danish fire insurance claim data, all the data lies within $\sqrt{2167-1} \approx 46.5403$ standard deviations from the mean. In other words the maximum data point, 263.3, has to be less than

$$3.385088 + 46.5403 \times 8.507452 = 399.324$$

which is not too bad a bound.

The Laguerre-Samuelson inequality can be expressed as saying that if *min* denotes the minimum value of the Danish fire insurance claims, and *max* denotes the maximum, then:

$$\mu - \sqrt{2166} \times \sigma \leq \min < \max \leq \mu + \sqrt{2166} \times \sigma$$

Substituting values for μ , \min and \max - but not σ - we see that

$$3.385088 - 46.5403 \times \sigma \leq 1.0 < 263.3 \leq 3.385088 + 46.5403 \times \sigma$$

and the last inequality says that

$$\sigma \geq (263.3 - 3.385088)/46.5403 = 5.58473$$

which is not a bad lower bound for the actual value $\sigma = 8.507452$

The first inequality says that

$$\sigma \geq (3.385088 - 1.0)/46.5403 = 0.0512478$$

which, of course, is a much worse lower bound for the standard deviation.

For a general data set, the Laguerre-Samuelson inequality will always yield the lower bounds:

$$\sigma \geq (\max - \mu)/\sqrt{n-1}$$

and

$$\sigma \geq (\mu - \min)/\sqrt{n-1}$$

for the standard deviation, where n is the number of data points. Unless the data values grow commensurate with the number of data points these estimates rapidly become useless, because the numerator $1/\sqrt{n-1}$ will dominate the size of the data and tend to 0.

Chapter 8

Comparing data sets

8.1 Scatter plots

8.2 Quantile plots

Part III

More Exploratory Tools

Part IV

The R community

Chapter 9

Why is it called “R”?

Chapter 10

Who uses R?

Chapter 11

R package development

Chapter 12

Major figures in R development