Haim Bar, HaiYing Wang, and Jun Yan

Stumbling into Data Science

Haim: To my family ... ...

HaiYing: To my family ... ...

Jun: To Jo, Bo, and Jiafeng, without whom the book would have been finished much sooner, but life would be unimaginable.

Contents

Notes for ourselves

Styles

Basic template for R Markdown from Wenjie Wang at Stat Data Science Lab[1] with source[2].

- Keep line width under 80 characters for tidy source view.
- No labeling equations unless references.
- Cross-references labels
  - chapters: `ch:estimation`; Chapter~@ref{ch:estimation}
  - equations: `eq:area`
  - tables: `tab:simu`
  - figures: `fig:normhist`
- R chunk lables: include chapter index to avoid unwanted duplicates
- Index when appropriate for key words.

Reminders

- Bib entries for R packages are generated by a chunk at the end of this file; its output `packages.bib` are not tracked in the repo.
- The dedication page is controlled by `latex/before_body.tex`.
- Keep pdf/html consistent as much as possible (need to learn a lot)

[1] https://statds.org/template/

[2] https://github.com/statds/dslab-templates/

Preface

Who this book is for

This book is the product of our efforts to promote data science among high school students.

Many interesting points ...

How to read this book

Some chapters can be skipped ...

Computing language

We use R because ...

About the authors

Haim Bar is an Associate Professor in the Department of Statistics, University of Connecticut. He does a lof of interesting things ...

HaiYing Wang is an Associate Professor in the Department of Statistics, University of Connecticut. He does a lof of interesting things ...

Jun Yan is a Professor in the Department of Statistics, University of Connecticut. He does a lof of interesting things ...

All three authors have been members of the Computer Committee of the Department for years, during which time the idea of the book emerged.

Prerequisites

We hope the widest accessibility.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): `https://yihui.org/tinytex/`.

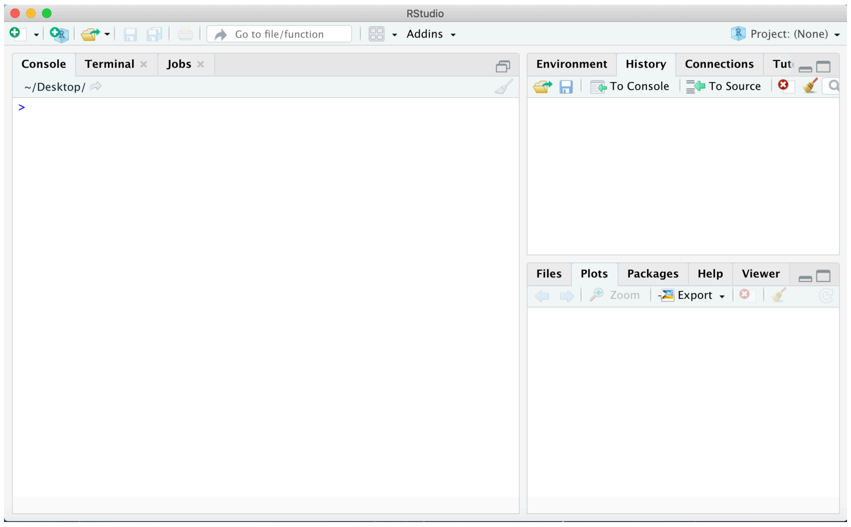
1 Introduction

In this (HB: class), we will use the R language, and the RStudio integrated development environment (IDE). Before the (HB: first meeting), install the necessary software on your personal computer by executing the following instructions.

1.1 Installing R and Rstudio

Go to the website of the Comprehensive R Archive Network (https://cran.r-project.org/) and download the latest version of R for your operating system (Windows, MacOS, or Linux). As of October 27, 2021, the latest version is 4.0.4. Follow the installation instructions.

Next, go the Rstudio download website (https://rstudio.com/products/rstudio/download/) and get the Desktop version (open source license). As of October 27, 2021, the version is 1.4.1106. Follow the installation instruction. An icon that looks like this: R will be on your computer's desktop. Double-click on this icon to start the R session. The Rstudio IDE will open, and look like this:



The left-hand side of the screen contains the Console tab. Notice the > sign (called the 'prompt'). When you see this character, it means that R is ready for the next command. Put the cursor there, and then type

```
2 + 2
```

and hit Enter on the keyboard. You should get the following on the Console:

```
[1] 4
```

Notice the top-right part of the Rstudio window. You should see an 'Environment' and a 'History' tab. Click on History. All your previous input appears there. Try entering another calculation or command and see that they appear in the session's history. For example, try entering the following

```
3 ^ 5
date()
```

Click on the Environment tab. It should be empty when you start R for the first time. In the Console, type

```
myFirstVariable <- factorial(5)
```

Notice that nothing was printed in the Console, but the Environment tab now contains a table with one row, with 'myFirstVariable' appearing in the cell on the left, and its value (120) on the right. Any object appearing in the Environment tab is available to you throughout your R session, and you don't have to redefine or recalculate it. For example, you can try the following:

```
myFirstVariable/6
```

The Console should now display 20.

The lower-right side of the IDE contains a file browser (the Files tab), information about installed packages (more about it later), and any plot generated during the R session. It also contains a Help tab, to obtain information about built-in functions.

Finally, before we move on to the next section, in the Rstudio top menu, click on File, then on New File, and then on R Script. Alternatively, you can click on little green '+' icon in the top-left part of the IDE. This will split the left side of the Rstudio IDE into two parts – the lower part will contain the Console, and the top part will contain a tab labeled 'Untitled1'. This is where you can enter R code which you will save to a permanent file, and re-use later. For example, enter the following in the blank space in the Untitled1 tab:

```
# This is my first R program
cat('Hello, World!\n')
```

Then, from the main menu in Rstudio, click on File, then on Save, and in the 'Save As' box enter FirstProgram.R and click the Save button. Notice that the tab name is now FirstProgram.R.

In that part of the window, there should now be a small button called Source. Click on it. The program will be executed and the output will be shown in the Console. You can also execute individual lines in the source code. Just put the cursor anywhere in that line, and click on the Run button (which is near the Source button) or click Ctrl+Enter (or Command+Enter on a Mac).

That's it. In the rest of these notes we will see more features of Rstudio, but you are now ready to start learning programming in R.

1.2 Basic Operations in R

1.2.1 Some Useful Functions

R has many built-in functions, and many more in external packages. We will introduce them as we go, but let's get started with some basic ones. The documentation on each function can be obtained by using `?func` or `help(func)` where `func` is some function. For example, to get the documentation about using the `help()` function, try the following:

```
help(help)
?help
```

When we start an R session, it's important to determine the 'working directory'. This is the folder on our computer which R will use to search for data or code files and save results. To find out which folder is currently used, we use the `getwd()` function

```
getwd()

[1] "/home/runner/work/sids/sids/book/Rnw"
```

If we want (and we often will), we may change the working directory by using `setwd()`.

```
setwd("~/Desktop")

Error in setwd("~/Desktop"): cannot change working directory
```

The `~/` notation is a shortcut to your home directory. Using this shortcut is convenient because if you are using different computers or you share code with others, the home directory may be different on each computer.

Data is stored in variables. We can get the data from a file (Excel, comma-separated values, etc.), the Internet, or we can generate it ourselves. Let's start by generating some data. The simplest function to create data is called `c()`, which stands for 'combine'.

```
(courseNames <- c("Data Science", "Statistics", "Probability"))

[1] "Data Science" "Statistics"   "Probability"
```

We created a variable called courseNames, and it contains three values. Note that we assigned the values into the variable by using the `<-` operator. The parentheses around the whole statement cause the content of the variable to printed immediately to the console. We can now do things with this variable. For example, we can check the variable type, using the `class()` function. We can also get a bit more detailed information by checking its structure, using the `str()` function.

```
class(courseNames)

[1] "character"

str(courseNames)

 chr [1:3] "Data Science" "Statistics" "Probability"
```

The variable is of class 'character', and it is a vector with three values. We can check the number of elements in a vector by using the `length()` function:

```
length(courseNames)

[1] 3
```

Pick good variable names. They should describe the meaning of the value, yet be short enough to type. Variable names can only contain letters, numbers, the dot character, or the underline character. Variable names can only begin with either a letter, or a dot as long as it is not followed by a number. They should not be the same as an existing function name or other reserved words in the language (like 'while', 'if', 'quit'.)

We can also generate variables which contain sequences of numbers. To do that, we use the `seq()` function. For example, we can create a variable which contains all the odd numbers between 1 and 20:

```
(oddLT20 <- seq(1,20,by=2))


 [1]  1  3  5  7  9 11 13 15 17 19


length(oddLT20)


[1] 10
```

To generate consecutive values, we can also use the colon operator:

```
(firstThirteen <- 1:13)


 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13
```

Another useful function to generate data is `rep()`, which replicates values. We often have to generate a vector of ones or zeros, and we can do it like in the following example. We use this opportunitiy to also introduce the functions `sum()` and `cumsum()` (cumulative sum).

```
(ones <- rep(1, 10))


 [1] 1 1 1 1 1 1 1 1 1 1


length(ones)


[1] 10


sum(ones)


[1] 10


cumsum(ones)


 [1]  1  2  3  4  5  6  7  8  9 10
```

There are also many functions to handle text. The `paste()` function attaches two strings of characters together. Note the usage of the collapse option (also called an 'argument' of the function) in the second example:

```
paste(courseNames)

[1] "Data Science" "Statistics"   "Probability"

paste(courseNames, collapse=", ")

[1] "Data Science, Statistics, Probability"
```

Let's combine what we've learned so far to create the 52 cards in a standard deck, which includes four suits: Club, Diamond, Heart, and Spade:

```
suits <- c(rep("C",13), rep("D",13), rep("H",13), rep("S",13))
(cards <- paste0(suits, rep(1:13,4)))

 [1] "C1"  "C2"  "C3"  "C4"  "C5"  "C6"  "C7"  "C8"  "C9"  "C10" "C11" "C12"
[13] "C13" "D1"  "D2"  "D3"  "D4"  "D5"  "D6"  "D7"  "D8"  "D9"  "D10" "D11"
[25] "D12" "D13" "H1"  "H2"  "H3"  "H4"  "H5"  "H6"  "H7"  "H8"  "H9"  "H10"
[37] "H11" "H12" "H13" "S1"  "S2"  "S3"  "S4"  "S5"  "S6"  "S7"  "S8"  "S9"
[49] "S10" "S11" "S12" "S13"
```

Note that `paste0()` is the same as `paste(..., collapse="")`.

Let's 'deal' five cards to each player for a game of poker. We will use the `sample()` function.

```
set.seed(5252)
(pokerHand <- matrix(sample(cards,20,replace=FALSE), nrow=5, ncol=4))

     [,1]  [,2]  [,3]  [,4] 
[1,] "S8"  "S13" "S7"  "C4" 
[2,] "D8"  "C1"  "H6"  "H11"
[3,] "D4"  "C11" "S6"  "S11"
[4,] "D12" "C13" "S4"  "S1" 
[5,] "C6"  "S9"  "H12" "H9" 
```

The `sample()` function in this example is used to draw 20 cards at random from the deck, without replacement. Then, to divide it into four hands, we use the `matrix()` function, and specify the number of rows and the number of columns.

Check the class and the structure of the variable pokerHand, using the functions we've mentioned earlier.

We can save variables to a file in order to use them in a later session.

```
save(pokerHand, file="pokerHand.RData")
```

If we don't specify the complete path, the file will be stored in the current working directory. Recall that you can find out which directory is used with the `getwd()` function, and set it to another directory with `setwd()`. Then, we may get the saved variables by using the following:

```
load("pokerHand.RData")
```

After you use the `load()` function, the variable pokerHand will be available to use.

Variables which we do not save, will not be available once we terminate the current R session. When we quit an R session, we have an option to save the entire session's information. However, if there are variables, datasets, or functions which we have created and want to save, it's better to save them explicitly.

There are a few constants in R , including the letters of the alphabet (upper- and lower-case), the month names, and the number π.

```
LETTERS

 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"

letters

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

month.abb

 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

month.name
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"
```

```
pi
```

```
[1] 3.141593
```

The base distribution of R is very comprehensive, but there are thousands of contributed packages which are written by R users. We will use several such packages in the book, so let us see how to do it. The package lattice provides 'elegant high-level data visualization system with an emphasis on multivariate data'. To install the package, we use the `install.packages()` function.

```
install.packages("lattice")
```

This has to be done just once. Occasionally, you may be prompted to install updates, which can also be done by using the `update.packages()` function. To use the package, we need to load it, using the `library()` function. In the following example we use a built-in dataset of opera singers, and we plot their heights by their vocal parts.

```
library("lattice")
bwplot(voice.part ~ height, data=singer, xlab="Height (inches)")
```

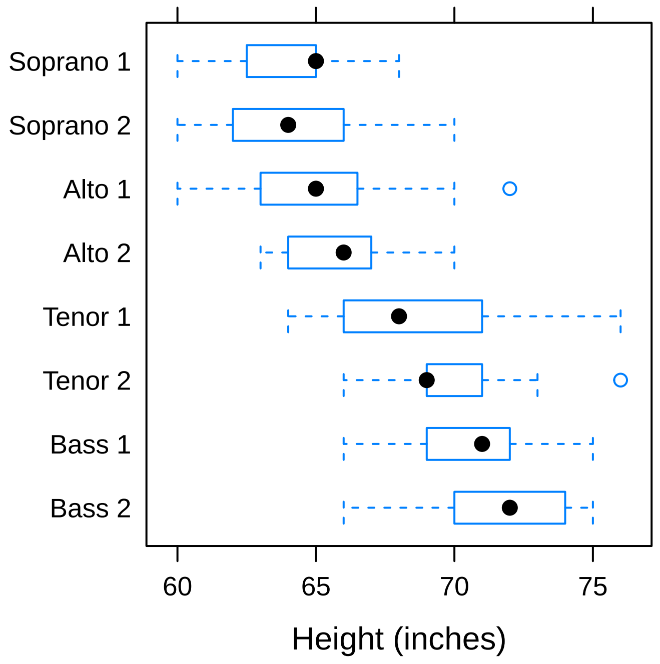


FIGURE 1.1: Using the lattice package to create boxplots

When you want to finish your session, just type `quit()`.

1.2.2 Generating random numbers

In this (HB: course), much of the learning will be done by using simulated data, so let's start by learning how to generate data. The basic function to generate random data is `runif()` which is used to draw random numbers, uniformly between 0 and 1. In the following example we create 10,000 random draws from a uniform distribution, keep these numbers in a variable called simData, and plot a histogram to show the distribution of the data we have generated. The `set.seed()` function is used to ensure that every time we run this code, we will get the same set of random numbers. This is called reproducible code.

```
# Generate 10,000 points from a uniform distribution
set.seed(210313)
n <- 10000
simData <- runif(n)
hist(simData)
```

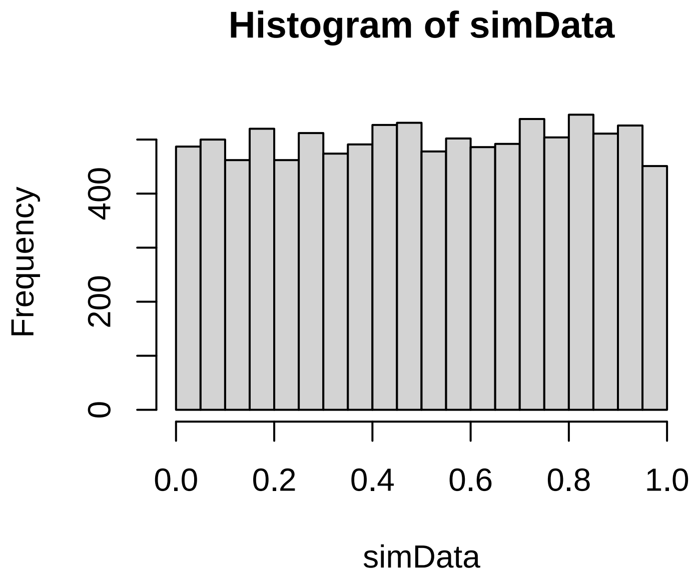


FIGURE 1.2: A histogram of 10,000 randomly drawn numbers from a standard uniform distribution.

From the range and the flatness of the histogram we can see that the generated data is indeed uniform on [0,1]. The `runif()` function can be used to draw random numbers uniformly on any finite interval. For example, if we want our random numbers to be in the interval [1,5] we will run the following code:

```r
n <- 10000
simData <- runif(n, min=1, max=5)
```

Try it, and draw the histogram as in the previous example. Try it with a fixed seed and verify that you get the same output each time. Then, run the code without a fixed seed and observe that you get a different distribution each time. Since the number of random draws is fairly large, the shape of the histogram will not change much.

From a random draw of a uniform distribution we can generate random numbers from other distributions. For example, we want to simulate (fair) coin flips, and count the number of Heads that we get. Let's say we want to simulate 200 coin tosses. We will draw 200 random numbers from a uniform distribution, and decide that we got Heads in the i-th toss if the i-the random number is less than 0.5, and Tails otherwise.

Try the following code multiple times (do not use `set.seed()`). What do you observe? We will discuss it further in a different (HB: lecture).

```
# Simulate 200 coin-flips, using the runif function:
n.trials <- 200
cat("Number of Heads is: ", sum(runif(n.trials) < 0.5), "\n")

Outout: Number of Heads is:  92
```

A few comments:

- A comment in R starts with #. Any text following the # sign is considered user-documentation and is not executed by R .
- We have used the `cat()` function to print (concatenate) text to the console. Fixed text appears in double (or single) quotes, but the content of variables or output from R functions should not be quoted. The `\n` symbol tells R to print a newline character at the end. Try to see what happens if you remove it.
- The expression within the `sum()` function produces TRUE/FALSE (Boolean) values. First, we draw n.trials random numbers from a uniform distribution. Then, each one is compared with 0.5. If the value is less than 0.5, the returned value is TRUE. Otherwise, it's FALSE. This demonstrates one of R 's greatest features - allowing to run 'vectorized' code. In one line, we generated 200 random numbers and compared each one to 0.5, and as a result, we got 200 Boolean values which we have added together (TRUE counts as 1, and FALSE counts as 0.) So, the result in the sum is the number of Heads in 200 tosses.

Statistical inference is based on a mental exercise in which we ask, if we could repeat the same experiment infinitely many times, what would we see? With simulations, we can get a good approximation. For example, the 200 coin-tosses experiment can be repeated, say, 100 times. One way is to use loops, like in the following example:

```
set.seed(442886)
n.trials <- 200  # the number of coin-tosses in each experiment
reps <- 100  # the number of experiments
Heads <- rep(0, reps)  # A vector to store the results (initialize with 0s)
for (i in 1:reps) {
  Heads[i] <- sum((runif(n.trials) < 0.5))
}
hist(Heads, breaks=20)
```

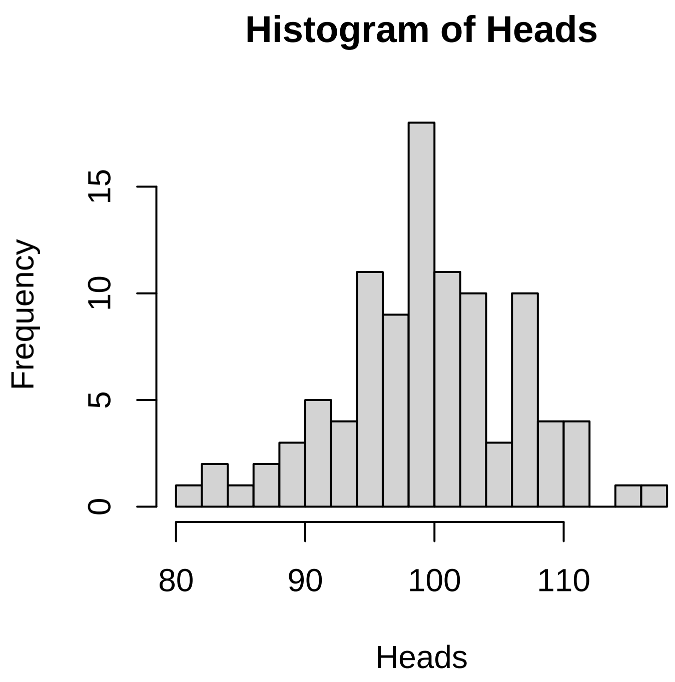


FIGURE 1.3: A histogram of 100 experiments, from each we get the total number of Heads in 200 fair coin tosses.

We've used a `for()` loop, with an index variable called `i` which runs from 1 to 100. In each iteration we simulated n.trials=200 coin tosses, as before. The result from each iteration is stored in the i-th position in the vector we called Heads.

Using the uniform distribution, we have created a random draw from a different distribution, called the 'binomial'. The mathematical notation is $Bin(N, p)$ where N is the number of trials such that in each trial there can be exactly two outcomes (e.g., coin tosses), and p is the probability of the first possible outcome (e.g., Head), and $1 - p$ is the probability of the second possible outcome (e.g., Tail). R has a built-in function to generate random numbers from many different distributions, so the code above can be replaced by the following, which uses the `rbinom()` function:

```
set.seed(442886)
reps <- 100
n.trials <- 200
Heads <- rbinom(reps, n.trials, 0.5)
hist(Heads, breaks=20)
```

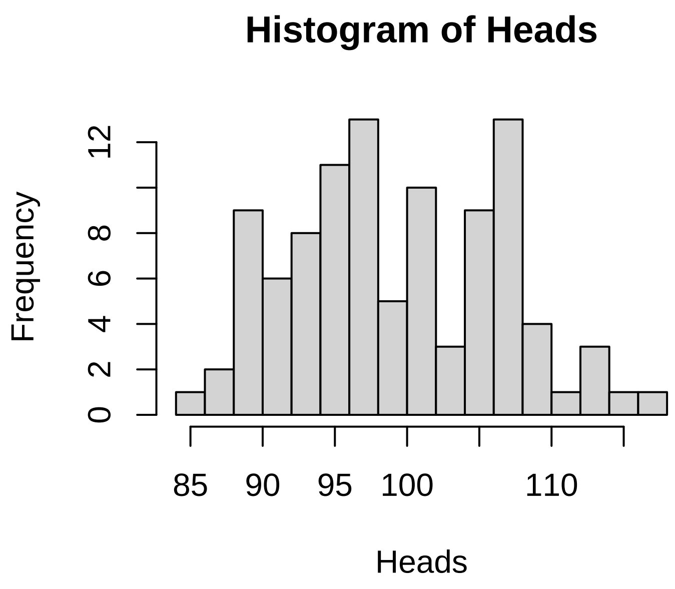


FIGURE 1.4: A histogram of 100 binomial experiments, using rbinom() to simulate 200 fair coin tosses in each experiment.

Exercises:

1. Change the number of simulated coin-toss datasets (`reps`) to 1,000 and rerun the code. Then change it to 10,000 and run it again. What do you notice?
2. Change the probability of Heads to 0.2 and run the code again. Then, change it to 0.8. What do you observe?
3. Try other values of the number of trials and the probability of Heads.

1.2.3 Summary statistics

We will introduce some statistical functions to summarize data. To demonstrate some of these functions, let's first generate some data:

```
n <- 10000
lambda <- 10
x <- -log(runif(n))/lambda
```

Note that in R , the `log` function uses the natural logarithm by default. To use base 10 or base 2, use `log10` and `log2`, respectively. To specify the base, use the `base` argument. For example, try `log(16, base=4)`.

The most commonly used one is the mean (a.k.a. the average), $\overline{x}$:

$$\overline{x} = \frac{x_1 + ... + x_n}{n} .$$

Other ways to estimate some sort of 'central tendency' of a distribution are:

- The trimmed mean, which is similar to the mean, except that the smallest and largest $p \cdot 100\%$ of the values are excluded from the computation. In our example, if we take $p = 0.1$ with the simulated data, only 8,000 data points will be used in the calculation of the trimmed mean;
- The median, which is a number (denoted here by $x_{0.5}$) such that half the data points are greater than $x_{0.5}$ and half are less than or equal to $x_{0.5}$.

Let's compute the sample's mean, trimmed mean, and median. Notice that in this example the mean is greater than trimmed mean, which is greater than the median. In general, the mean is not a great estimate of the 'center' of a non-symmetric distribution. We say that the mean is more 'sensitive to extreme values' than the median.

```
mean(x)


Output: [1] 0.1003771


mean(x, trim=0.1)
```

```
Output: [1] 0.08319167


median(x)


Output: [1] 0.06826259
```

The formula we used to generate the data is the probability function of the exponential distribution, with rate parameter $\lambda = 10$. We denote it by $X \sim exp(\lambda)$. The exponential distribution is often used to model random waiting times, like the time between incoming text messages. We would usually generate it by using the `rexp()` function in R . Mathematical analysis of the distribution leads to the fact that the expected value of an exponential random variable with rate λ, is $1/\lambda$. We see that the theoretical expected value of our example is 0.1, and the sample mean is very close to 0.1. This is no coincidence - we will discuss it in later chapter.

The distribution of `x` is shown below as a box-and-whisker plot (or simply, boxplot). This is a very simple representation of numeric data, which is constructed by summarizing the data using a few numeric characteristics. The boxplot below is drawn horizontally, and the vertical grey line inside the box is the median. Similar to the median, we find the first quartile – a point, $x_{0.25}$, such that 25% of the values are less than $x_{0.25}$ and 75% are greater than x_{25}; and the third quartile – a point, $x_{0.75}$, such that 75% of the values are less than $x_{0.75}$ and 25% are greater than $x_{0.75}$. The first and third quartiles are the vertical edges of the box, also called the lower and upper hinges. So, the box represents 50% of the data. The range between the first and third quartiles is called the Inter-Quartile Range, or IQR, which is sometimes used to estimate the dispersion or spread of the data. The 'whiskers', which are the dashed grey lines, are constructed by adding 1.5·IQR to each side of the box. If the result is smaller than the minimum value (or greater than the maximum), then the whisker only extends to the minimum (maximum). Points within the range between the two whiskers are not plotted individually, since their distribution is summarized succinctly by the box-and-whiskers plot. Points outside the range between the two whiskers are considered 'outliers', or extreme values, and are shown explicitly.

The plot was generated with the following code. Try it, and try changing some of the parameters to understand their role. We will cover the topic of visualization in a different chapter.

```
boxplot(x, cex=0.5, col=4,border = "grey66", horizontal = T, axes=F, at=0.25)
axis(1, pos = 0)
points(mean(x),0.25,col=2, pch=19, cex=0.7)
points(mean(x, trim=0.1),0.25, col="brown", pch=18)
```

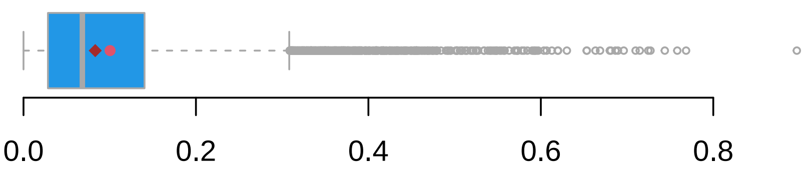


FIGURE 1.5: A boxplot.

A boxplot does not include the mean, or the trimmed mean, but we have added them here as a red circle and brown diamond, respectively, in order to show that they are different than the mean. The median is smaller than the mean in this case, so we say that the distribution is skewed to the left. The median does not depend on the scale of the data. It simply represents where half the data lies.

A more detailed summary of a sample can be obtained by calculating its quantiles. In the boxplot, only three quantiles (also called percentiles) are shown. We can show the quintiles (20, 40, 60, 80 percentiles), or deciles (10, 20,..., 90 percentiles) by using the `quantile()` function.

```
hist(x, breaks=50, border="white", col="lightblue", freq=FALSE, xlim=c(0,0.6))
deciles <- quantile(x, probs=seq(0.1,0.9,by=0.1))
abline(v=deciles, lty=2, col="purple", lwd=2)
text(deciles, dexp(deciles, 10), paste0(seq(10,90,by=10),"%"), cex=0.7, col="orange")
```

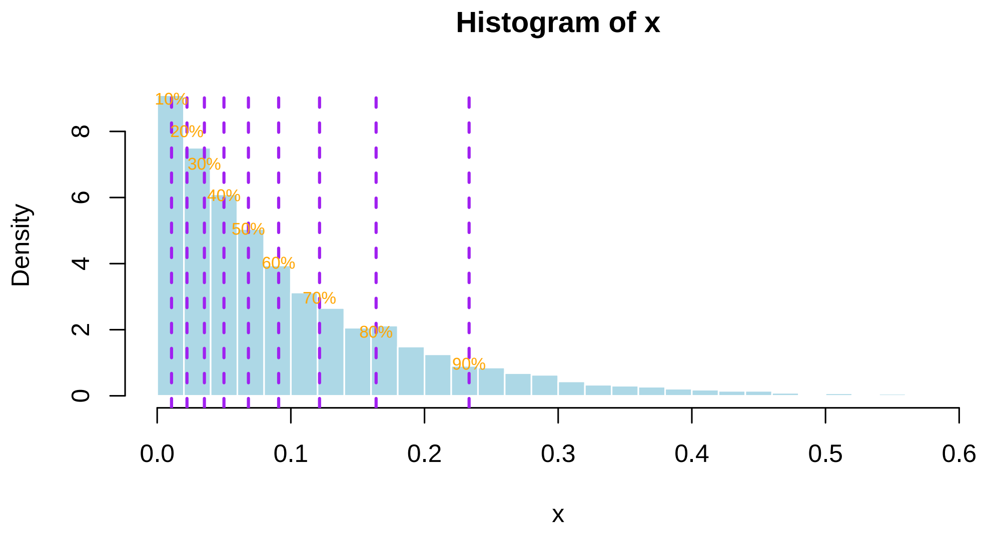


FIGURE 1.6: A histogram of a sample from an exponential distribution, with the 10, 20, 30,... percentiles

In addition to sample statistics which summarize some notion of the center of the distribution, we are often interested in estimating the dispersion of the data. The most commonly used measures of dispersion are the variance, and its square root - the standard deviation. The variance is defined as follows

$$Var(X) = \frac{(x_1 - \mu)^2 + ... + (x_n - \mu)^2}{n}$$

where μ is the mean of the distribution of X. In words, the variance is the average squared deviation from the mean. Notice that in R the variance is computed with n-1 in the denominator (with n-1 we get an 'unbiased estimator' for the true variance).

The corresponding functions in R are `var()` and `sd()`. In the following code we also demonstrate the `IQR()` function.

```
var(x)


Output: [1] 0.01012885


sd(x) # note that sd(x) is sqrt(sd(x))
```

```
Output: [1] 0.1006422


IQR(x)


Output: [1] 0.1120348
```

We will see in subsequent chapters that the normal distribution plays a major role in statistics. It is defined by a probability density function, with two parameters – μ and σ^2:

$$\phi(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] .$$

It is symmetric, 'bell-shaped', and centered around the mean, μ, as shown in the following plot. The blue histogram is generated by the `rnorm()` function, using $\mu = 2.5$ and $\sigma^2 = 0.25$. The orange curve shows the function ϕ, by using the `dnorm()` function (to obtain the density of x).

```
x <- rnorm(10000, mean=2.5, sd=0.5)
hist(x, breaks=30, border="white", col="navyblue", freq=FALSE)
xs <- seq (0,5, length=500)
lines(xs, dnorm(xs,2.5, 0.5), col="orange", lwd=3)
```

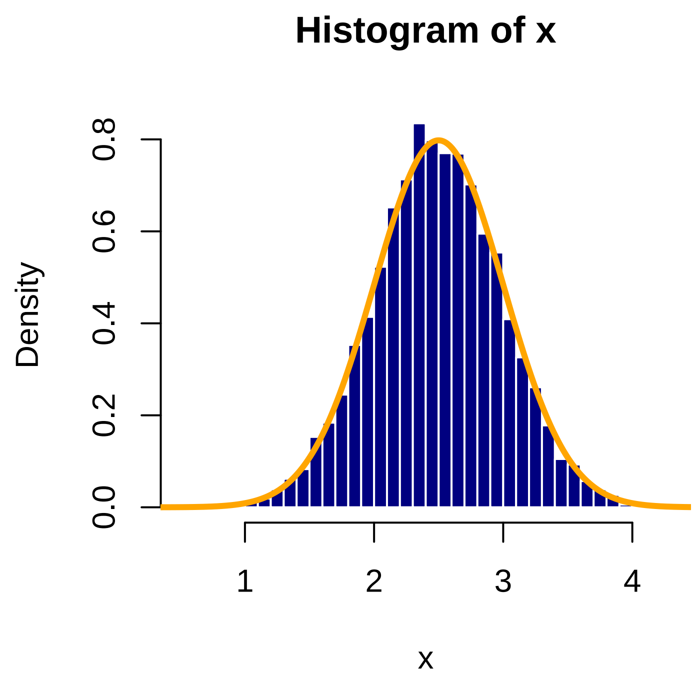


FIGURE 1.7: A histogram of a sample from an normal distribution, with mean=2.5 and variance=0.25

We can obtained more detailed summaries about the sample, by using the functions `summary()` and `describe()` (the latter is from the psych package, and provides more details.)

```
print(summary(x))


Output:    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
Output:  0.5987  2.1621  2.4882  2.4922  2.8268  4.2948


print(psych::describe(x))


Output:    vars     n mean  sd median trimmed  mad min  max range skew kurtosis se
Output: X1    1 10000 2.49 0.5   2.49    2.49 0.49 0.6 4.29   3.7    0     0.02  0
```

`summary()` gives the minimum, maximum, the 25th, 50th, and 75th percentiles, as well as the mean of the sample. `psych::describe()` also gives the sample size, the standard deviation, the trimmed mean, the mean absolute deviation (mad), the range (maximum minus minimum), the skewness, and the kurtosis (HB: will we use the kurtosis later? If so, we should add a ref.).

We conclude this chapter with some functions which can be used to summarize discrete (categorical) data, where the mean, variance, quantiles, etc. are not applicable. We will revisit the topic of summarizing data in the chapter on (HB: visualization).

To simulate categorical data we can use the `rmultinom()` function, which simulates putting N objects in K bins with given probabilities. Another possibility is to use `cut()` function which divides a range of number into discrete ranges and creates discrete categories in a factor variable. For example, suppose there is a town with three hotels (Motel 6, Best Western, and Hilton) with 400, 300, and 300 rooms, and one auto rental company with only two makes of cars (700 Honda, and 300 Teslas). In the following code we simulate the allocation of 100 visitors to hotels and cars. We use the `table()` function to show the counts by hotel, by car, and by both.

```
hotelrooms <- cut(runif(100),breaks = c(0,0.4,0.7,1), include.lowest = TRUE)
levels(hotelrooms) <- c("Motel 6", "Best Western", "Hilton")
autorental <- cut(runif(100),breaks = c(0,0.7,1), include.lowest = TRUE)
levels(autorental) <- c("Honda", "Tesla")
(hoteltbl <- table(hotelrooms))

Output: hotelrooms
Output:      Motel 6 Best Western       Hilton
Output:           41           27           32

(autotbl <- table(autorental))

Output: autorental
Output: Honda Tesla
Output:    66    34

table(hotelrooms, autorental)

Output:               autorental
Output: hotelrooms     Honda Tesla
Output:   Motel 6         29    12
Output:   Best Western    19     8
Output:   Hilton          18    14
```

We can use the `max()` and `which.max()` functions to find the mode of the data (the most frequent value).

```
cat(levels(hotelrooms)[which.max(hoteltbl)],":", max(hoteltbl),"\n")


Output: Motel 6 : 41


cat(levels(autorental)[which.max(autotbl)],":", max(autotbl),"\n")


Output: Honda : 66
```

2

Probability and Paradoxes

2.1 Probability

The world is random. Probability theory has been developed to quantify the randomness. We will go through examples to see how probability helps interpret phenomena in real life and how probability may counter intuition.

Some concepts:

- Experiment: a situations in which the outcome occur randomly.
- Sample Space: the set of all possible outcomes in an experiment.
- Event: a subset of the sample space is called an event; an event occur if the outcome from an experiment belong to the event.
- Probability: assuming the elements of the sample space all have equal chance to occur, the probability of event A is

  $$P(A) = \frac{n}{N} = \frac{\text{number of elements in A}}{\text{total number of outcomes}},$$

  where n is the number of elements in A and N is the number of elements in the sample space. Note that this formula holds only if all the outcomes are equally likely.

2.2 Two Child Problem

Example 1. ([Two Child Problem)] Consider the following two problems:

1. Mr. Jones has two children. The older child is a girl. What is the probability that both children are girls?

2. Mr. Smith has two children. At least one of them is a girl. What is the probability that both children are girls?

Here are the possibilities of {older child, younger child}: {Boy, Boy}, {Boy, Girl}, {Girl, Boy}, {Girl,Girl}, with equal probabilities to occur.

For the first question: two elements satisfy the situation (the sample space for the given situation) – {Girl, Boy}, {Girl,Girl}; one element corresponds to the event of two girls (number of elements in the event of interest). Thus the probability of two girls is $\frac{1}{2}$. This question can also be solved this way: Since the older child is a girl, the probability of two girls is the probability that the younger child is a girl which is $\frac{1}{2}$.

For the second question, three elements satisfy the situation (the sample space for the given situation): {Boy, Girl}, {Girl, Boy}, {Girl,Girl}. Thus the probability is $\frac{1}{3}$.

2.3 Elevator waiting time

Example 2. (Elevator waiting time) Mr. Smith works on the 13th floor of a 15 floor building. The only elevator moves continuously through floors 1, 2, . . . 14, 15, 14, . . . 2, 1, 2, . . . , except that it stops on a floor on which the button has been pressed. Assume that time spent loading and unloading passengers is very small compared to the travelling time. Mr. Smith complains that at 5pm, when he wants to go home, the elevator almost always goes up when it stops on his floor. What is the explanation?

When Mr. Smith gets to the elevator, it may be below the 13th floor or above it. The elevator will goes up if it is below the 13th floor and it will goes down if it is above the 13th floor. There are 12 floors below the 13th floor and 2 floors above it, so the probability that the elevator is below the 13th floor is $12/14 \approx 0.86 > 0.5$. Thus no matter when Mr. Smith wants to go home, it is more likely that the elevator is going up.

We can simulate this situation:

```r
set.seed(123) ## set the random seed so that results are reproducible
n <- 10000 ## number of days to simulate
## the sample space is the possible locations of the elevator [1, 15]
```

```
elevator <- runif(n, min=1, max=15) ## outcomes of experiments
up <- (elevator < 13)
## the elevator is below the 13th floor so that it will goes up
sum(up) / n

## [1] 0.8633
```

(JY: The location does not mean the moving direction)(WHY: See if it is clear now.)

2.4 Fair Division

Example 3. (Fair Division) Tom and Jerry, each put 30 dollar in a jackpot to start a game. Suppose that they have equal chance to win and the one who wins three times takes all the 60 dollars. Now, Tom has won twice and Jerry has won once, but then something happens and the game must be stopped. How should they split the 60 dollars according their probabilities of winning if the game was finished?

It seems Tom has won twice and Jerry has won once so the 60 dollars should be split as $40 : 20$. However, these are not proportional to their probabilities of winning. They need at most another two games to know the final winner. Here are the four possible results of the two games: {Tom, Tom}, {Tom, Jerry}, {Jerry, Tom}, {Jerry, Jerry}. The probabilities that Tom and Jerry would be the final winner are $\frac{3}{4}$ and $\frac{1}{4}$, respectively, so the fair division should be $45 : 15$.

Let's use simulation to solve this problem.

```
set.seed(2021)
players <- c("Tom", "Jerry")
res <- replicate(n=1000, sample(players, 5, replace=TRUE))
## Simulate the game n times
TomFirst = (res[1:3,] == "Tom")
## Whether Tom won in the first three rounds
TomTwice = colSums(TomFirst) == 2
## Whether Tom won twice in the first three rounds
TomFinal = (colSums(res == "Tom") >=3) & TomTwice
```

```
## Whether Tom would be the final winner
TomP = sum(TomFinal) / sum(TomTwice)
## The probability that Tom would be the final winner
c(Tom=TomP, Jerry=1-TomP)


##       Tom     Jerry
## 0.7647059 0.2352941
```

2.5 Birthday Problem

Example 4. (Birthday Problem) Suppose that a room contains 23 people. What is the probability that at least two of them have a common birthday? Assuming that each year has 365 days, this probability seems very small, but it is actually about 0.5. What is the probability that some one in that room has the same birthday as yours? This probability is quite small (≈ 0.061). In order to have the probability that someone's birthday is the same as yours to be 0.5, we need 253 random selected people to be in that room.

Let's use simulation to verify the aforementioned numbers.

```
set.seed(2021)
birthday <- function(n=23, yours="1980-05-01"){
    # n is the number of people
    # yours is your birthday with format "year-mm-dd"
    N <- 365 # number of days each year
    room <- sample(1:N, size=n, replace=TRUE) # randomly choose n people
    doy <- as.numeric(strftime(yours, format="%j"))
    # convert the convert your birthday to day of year
    share <- length(unique(room)) < n
    ## if there are people sharing a common birthday
    same <- doy %in% room # if someone's birthday the same as yours
    return (c(share, same))
}

res <- replicate(n=1000, birthday(23))
rowMeans(res)
```

```
## [1] 0.506 0.073
```

```r
res <- replicate(n=1000, birthday(253))
rowMeans(res)
```

```
## [1] 1.00 0.51
```

2.6 Henry's Choice

Henry has been caught stealing cattle, and is brought into town for justice. The judge is his ex-wife Gretchen, who wants to show him some sympathy, but the law clearly calls for two shots to be taken at Henry from close range. To make things a little better for Henry, Gretchen tells him she will place two bullets into a six-chambered revolver in successive order. She will spin the chamber, close it, and take one shot. If Henry is still alive, she will then either take another shot, or spin the chamber again before shooting.

Henry is a bit incredulous that his own ex-wife would carry out the punishment, and a bit sad that she was always such a rule follower. He steels himself as Gretchen loads the chambers, spins the revolver, and pulls the trigger. Whew! It was blank. Then Gretchen asks, "Do you want me to pull the trigger again, or should I spin the chamber a second time before pulling the trigger?" What should Henry choose?

We know that the first chamber Gretchen fired was one of the four empty chambers. Since the bullets were placed in consecutive order, one of the empty chambers is followed by a bullet, and the other three empty chambers are followed by another empty chamber. So if Henry has Gretchen pull the trigger again, the probability that a bullet will be fired is 1/4.

If Gretchen spins the chamber again, the probability that she shoots Henry would be 2/6, or 1/3, since there are two possible bullets that would be in firing position out of the six possible chambers that would be in position.

```r
set.seed(2021)
n <- 10000 # number of simulations
```

```
spin.shot <- replicate(n, sample(1:6, 2, replace=TRUE))
## Assume that the bullets are in chambers 1 and 2.
first.blank <- spin.shot[1,] > 2
prob1 <- sum(spin.shot[2,first.blank] <= 2) / sum(first.blank)
twoshots <- function() {
    first.shot  <-  sample(1:6, 1)
    second.shot  <-  ifelse(first.shot == 6, 1, first.shot+1)
    c(first.shot, second.shot)
}
shot.again <- replicate(n, twoshots())
first.blank <- shot.again[1,] > 2
prob2 <- sum(shot.again[2,first.blank] <= 2) / sum(first.blank)
c(prob1, prob2)

## [1] 0.3348978 0.2562649
```

2.7 Intransitive Dice

We know that in general if $A > B$ and $B > C$ then $A > C$. However, this may not be the case in the word of probability.

Consider three dice with different sides numbers:

- Die A has sides 2, 2, 4, 4, 9, 9.
- Die B has sides 1, 1, 6, 6, 8, 8.
- Die C has sides 3, 3, 5, 5, 7, 7.

To play a game using dice A and B, you can chose which die to roll and your opponent rolls the other. The one who tolls a larger number wins. Which die do you want to choose?

Let's table the possible results to see the better die.

			B	
		1	6	8
	2	$A > B$	$A < B$	$A < B$
A	4	$A > B$	$A < B$	$A < B$
	9	$A > B$	$A > B$	$A > B$

Since die A wins five out of the nine possible results and all possible results occur with equal probability, we know that die A has a higher winning probability ($\frac{5}{9}$) than die B. We should choose die A over die B.

Now consider dice B and C.

		B		
		1	6	8
	3	$C > B$	$C < B$	$C < B$
C	5	$C > B$	$C < B$	$C < B$
	7	$C > B$	$C > B$	$C < B$

We see that die B has a higher winning probability ($\frac{5}{9}$) than die C, so we should choose die B over die C.

Since we should choose die A over die B, and choose die B over die C, does this mean that we should choose die A over die C if these two dice are to be selected. Surprisingly, the answer is NO. Here is the table of the possible results.

		C		
		3	5	7
	2	$A < C$	$A < C$	$A < C$
A	4	$A > C$	$A < C$	$A < C$
	9	$A > C$	$A > C$	$A > C$

We should choose die C over die A!

Here is an experiment to simulate the intransitive dice.

```r
set.seed(2021)
A <- c(2, 2, 4, 4, 9, 9)
B <- c(1, 1, 6, 6, 8, 8)
C <- c(3, 3, 5, 5, 7, 7)
n <- 1000
rollA <- sample(A, n, replace=TRUE)
rollB <- sample(B, n, replace=TRUE)
rollC <- sample(C, n, replace=TRUE)
mean(rollA > rollB)
```

```
## [1] 0.565
```

```r
mean(rollB > rollC)
```

```
## [1] 0.534
```

```r
mean(rollC > rollA)
```

```
## [1] 0.543
```

A set of dice is intransitive if it contains three dice, A, B, and C, with the property that A rolls higher than B more than half the time, and B rolls higher than C more than half the time, but it is not true that A rolls higher than C more than half the time. In other words, a set of dice is intransitive if the binary relation – X rolls a higher number than Y more than half the time – on its elements is not transitive.

2.8 Bertrand's Box

Bertrand's box paradox was first posed by Joseph Bertrand 1889. Here is the question: There are three boxes, one contains two gold coins, one contains two silver coins, and one contains a gold coin and a silver coin.

A box is selected at random and a coin is taken from that box at random. If the coin is a gold coin, what is the probability that the other coin in that box is also a gold coin.

```r
set.seed(2021)
boxs <- c("GG", "GS", "SS")

boxcoin <- function(boxs){
    box <- sample(boxs, 1)
    idx <- sample(1:2, 1)
    coin <- substr(box, start=idx, stop=idx)
    return(c(box, coin))
}

res <- replicate(n=1000, boxcoin(boxs))

mean(res[1, res[2,] == "G"] == "GG")
```

```
## [1] 0.6759443
```

2.9 Simpson's Paradox

It is a phenomenon in which a trend appears in several different groups of data but disappears or reverses when these groups are combined.

Example 5. (An urn example) A black urn contains 5 red and 6 green balls, and a white urn contains 3 red and 4 green balls. You are allowed to choose an urn and then choose a ball at random from the urn. If you choose a red ball, you get a prize. Which urn should you choose to draw from?

- If you draw from the black urn, the probability of choosing a red ball is $5/11 = .455$.
- If you choose to draw from the white urn, the probability of choosing a red ball is $3/7 = .429$.

You should choose to draw from the black urn.

Now consider another game in which a second black urn has 6 red and 3 green balls, and a second white urn has 9 red and 5 green balls.

- If you draw from the black urn, the probability of a red ball is $6/9 = .667$.
- If you choose to draw from the white urn, the probability if $9/14 = .643$.

Again you should choose to draw from the black urn.

In the final game, the contents of the second black urn are added to the first black urn, and the contents of the second white urn are added to the first white urn. Again, you can choose which urn to draw from. Which should you choose?

Intuition says choose the black urn, but let's calculate the probabilities.

- The black urn now contains 11 red and 9 green balls, so the probability of drawing a red ball from it is 11/20=0.55
- The white urn now contains 12 red and 9 green balls, so the probability of drawing a red ball from it is 12/21= .571.

You should choose the white urn!

Example 6. (UC Berkeley gender bias) A famous example of Simpson's paradox is a study of gender bias among graduate school admissions to University of California, Berkeley. In 1973 UC Berkeley was sued for sex-discrimination. Here are the overall numbers for the six largest departments in fall admission of 1973.

Men		Women	
Applicants	Admitted	Applicants	Admitted
2691	1198 (**44.5%**)	1835	557 (30.4%)

This table shows that men were more likely than women to be admitted, and the difference was so significant. Let's see which departments were mainly responsible for this gender bias. To do this we broke open the data according to each departments.

	Men		Women	
Department	Applicants	Admitted	Applicants	Admitted
A	825	512 (62%)	108	89 (82%)
B	560	353 (63%)	25	17 (68%)
C	325	120 (37%)	593	202 (34%)
D	417	138 (33%)	375	131 (35%)
E	191	53 (28%)	393	94 (24%)
F	373	22 (6%)	341	24 (7%)

Things get strange after we divide the data according different departments. For the six departments, four of them accepted women more than men. To explain this, **?** noticed that women tended to apply to more competitive departments with low admission rates even among qualified applicants, whereas men tended to apply to less competitive departments with high admission rates among the qualified applicants.

To use simulation to further illustrate this, we simulate a data set with four groups in the following.

```r
set.seed(2021)
g <- 4 # number of groups
n <- 40 # number of instances in each group
z <- rep(1:4, each=n) # grouping variable
x <- runif(n*g, 0, 2) + z # x variable that depends on z
y <- 2 * z - x + rnorm(n*g) # y variable that depends on x and z
plot(x, y) # plot the whole data
```

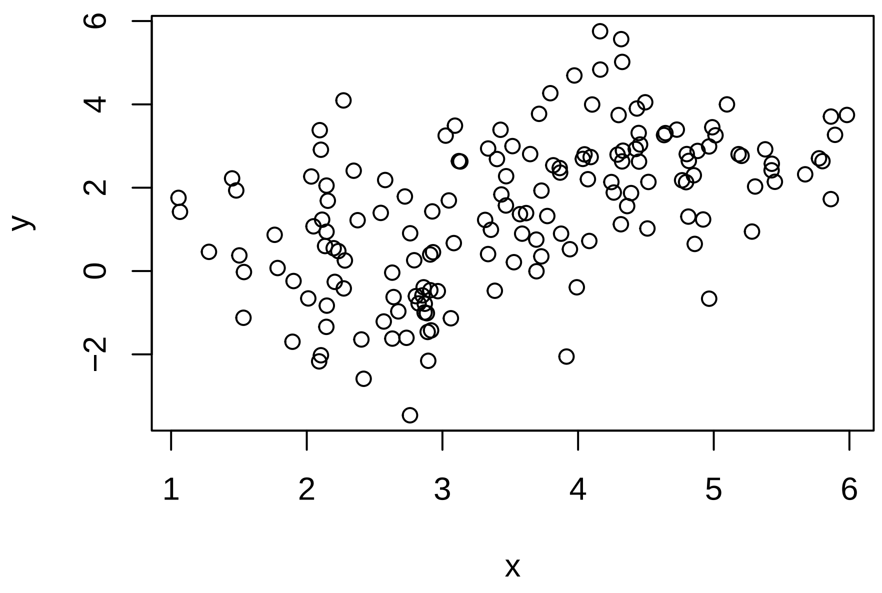


FIGURE 2.1: Scatter plot for the whole data

```
plot(x, y, pch=rep(1:g, each=n), col=rep(1:g, each=n))
```

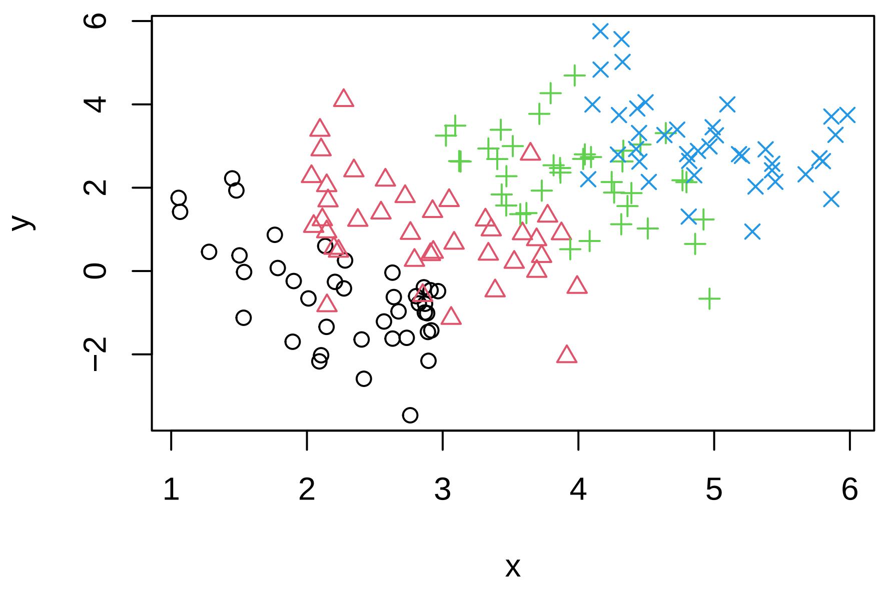


FIGURE 2.2: Scatter plot with different groups labeled

```
# label the points for different groups
```

We see that for the whole data, y has an increase pattern as x increases, but for each sub group y has a decrease pattern as x increases.

2.10 100 prisoners problem

The probability may help to obtain some changes in a seemingly hopeless situation. Consider the following example modified from **?**.

In a prison, there are 100 death row prisoners who are numbered from 1 to 100, and there is a room with 100 drawers labeled from 1 to 100. The director randomly puts one prisoner's number in each closed drawer and offers a last chance. The prisoners enter the room, one after another. Each prisoner may open and look into 50 drawers in any order. The drawers are closed again afterwards. If, during this search, every prisoner finds his number in one of the drawers, all prisoners are pardoned. If some prisoner does not find his number, all prisoners die. Before the first prisoner enters the room, the prisoners may discuss strategy, but they cannot communicate once the first prisoner enters the room.

The situation is hopeless if every prisoner selects 50 drawers at random. The probability that a single prisoner finds his number is 0.5, so the probability that all prisoners find their numbers is $0.5^{100} = 7.89 \times 10^{-31} \approx 0$. However, a better strategy gives the prisoners more than 0.30 probability to survive (**?**). The strategy is described below.

1. Each prisoner first opens the drawer with his own number.
2. If this drawer contains his number he is done and was successful.
3. Otherwise, the drawer contains the number of another prisoner and he next opens the drawer with this number.
4. The prisoner repeats steps 2 and 3 until he finds his own number or has opened 50 drawers.

In the following, we define two functions to simulate the method of randomly open 50 drawers and the better strategy, respectively.

```
set.seed(2021)
n <- 100 # number of prisoners
prisoners <- 1:n # prisoners' numbers
# simulate the procedure of randomly open 50 (n/2) drawers
open.random <- function(prisoners, n=length(prisoners)) {
    drawers <- sample(1:n, size=n, replace=FALSE)
    # randomly put prisoners' numbers in the drawers
    pardon <- TRUE # initialize pardon to be true
    for (i in prisoners) {
        opens <- sample(drawers, n/2)
        # randomly open n/2 drawers
        if (!(i %in% opens)) {
            # if any prisoner does not find his number
            # all prisoners die
            pardon <- FALSE
            break
        }
    }
    return (pardon)
}

open.smart <- function(prisoners, n=length(prisoners)) {
    drawers <- sample(1:n, size=n, replace=FALSE)
    pardon <- TRUE
    for (i in prisoners) {
        opens <- rep(NA, n/2)
        opens[1] <- drawers[i]
        for (j in 2:(n/2)){
            if (opens[j-1] == i) {
                break
            } else {
                opens[j] <- drawers[opens[j-1]]
            }
        }
        if (!(i %in% opens)) {
            pardon <- FALSE
            break
        }
    }
    return (pardon)
}
# survival probability of randomly open
mean(replicate(10000, open.random(prisoners)))

## [1] 0

# survival probability of using the better strategy
mean(replicate(10000, open.smart(prisoners)))

## [1] 0.2999
```

2.11 Monty Hall problem

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

```
set.seed(2021)
n <- 1000
first <- 1
prize <- rep(NA, n)
host <- rep(NA, n)
for (i in 1:n){
    prize[i] <- sample(1:3, 1)
    if (prize[i] == 1){
        host[i] <- sample(c(2, 3), 1)
    } else if (prize[i] == 2) {
        host[i] <- 3
    } else if (prize[i] == 3) {
        host[i] <- 2
    }
}
observed <- prize[host == 3]
sum(observed == 2) / length(observed)


## [1] 0.6707566
```

3

Towards the End of the Horizon

3.1 Introduction

Let's start with a couple of simple examples:

1. We get a random sample of 200 people and give them an IQ test. What is the distribution of the scores in this sample? How close is it to the true distribution of the whole population?
   How likely is it that we find a person with IQ greater than 150? In order to be accepted to the Mensa club a person has to be in the top 2% of the IQ distribution. What is the minimum score required in order to be a Mensa member?
2. The pockets of the roulette wheel are numbered from 0 to 36. In number ranges from 1 to 10 and 19 to 28, odd numbers are red and even are black. In ranges from 11 to 18 and 29 to 36, odd numbers are black and even are red. There is a green pocket numbered 0 (zero). Suppose you want to bet on a color, say, red, and if the ball lands in a red pocket your payoff is twice your bet, and otherwise you lose your bet. What is the probability you win? What is your expected payoff? Suppose you have $10,000, and you bet one dollar each time. How much money will you win or lose?.
3. Suppose that you get a text message every 6 minutes (a rate of 10 per hour), and suppose that they arrive independently of one another, according to a Poisson distribution. How many messages do you expect to get on an average day, between 8am and 8pm?

The main paradigm in statistical inference and prediction in order to answer such question is to take a sample from the population we're interested in, and choose a mathematical model which we believe represents the whole population. Then we check if our model appears to reasonable, in the sense that it fits the data well. If it does, we can use the mathematical properties of the model to draw conclusions about the population. By representing the sample using a mathematical (probabilistic) formula, we essentially augment a finite sample to an infinite one. Working with a formula/model for the data allows us to use the 'heavy artillery' of math, like finding maximum/minimum, computing areas under the curve of the function, and so on.

This chapter deals with inference which can be derived from the mean of a population, and we show the (arguably) two most important results in statistics – the law of large numbers (LLN), and the central limit theorem (CLT). Together, these two very general theorems make it possible to draw conclusions about the whole population, from a single sample! (as long as the sample size is large enough).

3.2 The Law of Large Numbers

3.2.1 Example #1 – the IQ score

We get a random sample of 200 people and give them an IQ test. From this sample, we want to infer the true distribution in the entire population.

Suppose that the IQ in the general population has a normal distribution with mean=100 and standard deviation =15, and from this population we draw 200 people, and calculate the sample mean and standard deviation:

```r
set.seed(95473)
n <- 200
samp <- rnorm(n, 100, 15)
cat("Mean=",mean(samp), ", SD=",sd(samp),"\n")
```

```
## Mean= 99.38613 , SD= 15.55667
```

We notice that the sample mean is 99.4 and the sample standard deviation is 15.6. Both are very close to the true values.

Let's pretend that just like in real life, we don't know the true distribution, so we have to check if the mathematical model we chose (normal distribution) is appropriate for the data from the finite sample. When it is assumed that the data come from a normal distribution, we can use the `qqnorm` function to check if the assumption is reasonable:

```
qqnorm(samp, cex=0.7, pch=18, col="purple")
abline(100,15,col="orange", lwd=3)
```

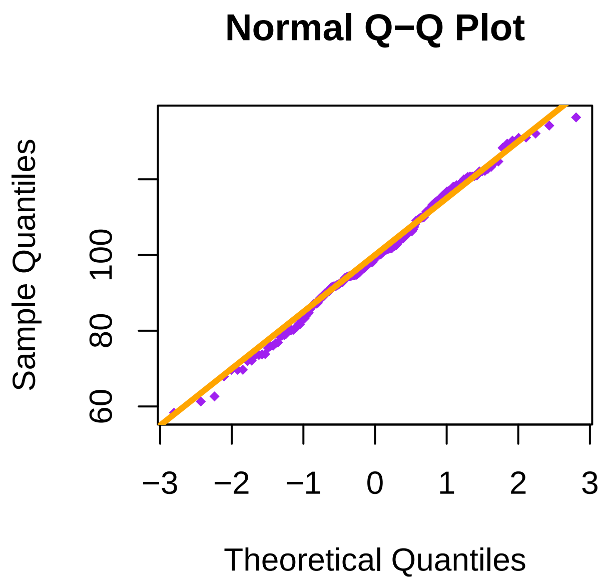


FIGURE 3.1: Q-Q plot for the simulated IQ scores.

The points in the Q-Q plot lie very close to a straight line, indicating that the sample was likely drawn from a normal distribution (because we generated it this way, this is not surprising. However, when we get a random sample and we do not know the true distribution, this plot is useful to check whether our mathematical model is reasonable.)

To understand what the LLN, let's obtain samples of varying sizes and see what happens to the sample mean as we increase n.

```
set.seed(95473)
ns <- seq(10, 2000, by=10)
L <- length(ns)
allMeans <- rep(0, L)
for (i in 1:L) {
  samp <- rnorm(ns[i], 100, 15)
  allMeans[i] <- mean(samp)
}
plot(ns, allMeans, pch=19, cex=0.5, col=3, axes=FALSE)
```

```
axis(1); axis(2)
abline(h=100, lwd=3,col=2)
```

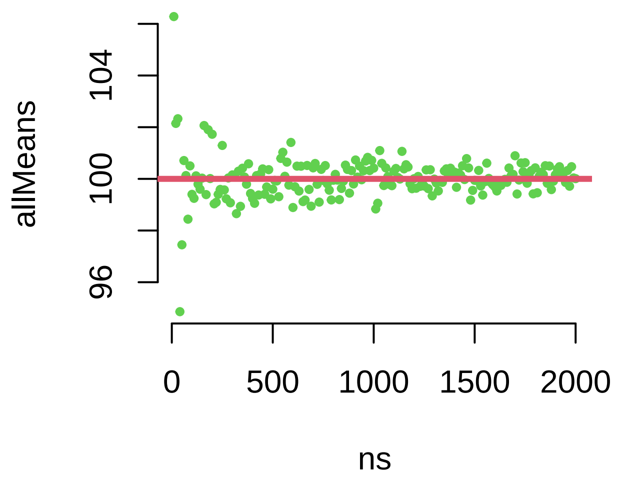


FIGURE 3.2: Q-Q plot for the simulated IQ scores.

We see that as n increases, the sample mean gets closer to the true mean (100).

3.2.2 Example #2 – the roulette wheel

Here, the outcome is binary – we either win or lose. The probability of winning if we choose the red color, is $18/37$. Let's simulate 40 bets at the roulette wheel, where, in each we put a dollar on red. Each bet is won with a Bernoulli distribution, $Ber(18/37)$ and we repeat it 40 times. Equivalently, we can look at it as a single binomial sample, with $n = 40$ and $p = 18/37$.

```
set.seed(75473)
n <- 40
samp <- rbinom(n, size=1, prob=18/37)
cat("Mean=",mean(samp), ", SD=",sd(samp),"\n")

## Mean= 0.575 , SD= 0.5006406
```

Notice that in those 40 bets, our probability of winning was 0.575 (greater than 0.5.) This seems great – if we keep going we might get rich. However, let's see what happens to the sample mean as we increase n.

```r
set.seed(95473)
ns <- seq(10, 2000, by=10)
L <- length(ns)
allMeans <- rep(0, L)
for (i in 1:L) {
  samp <- rbinom(ns[i], size=1, prob=18/37)
  allMeans[i] <- mean(samp)
}
plot(ns, allMeans, pch=19, cex=0.5, col=3, axes=FALSE)
axis(1); axis(2)
abline(h=18/37, lwd=3,col=2)
```

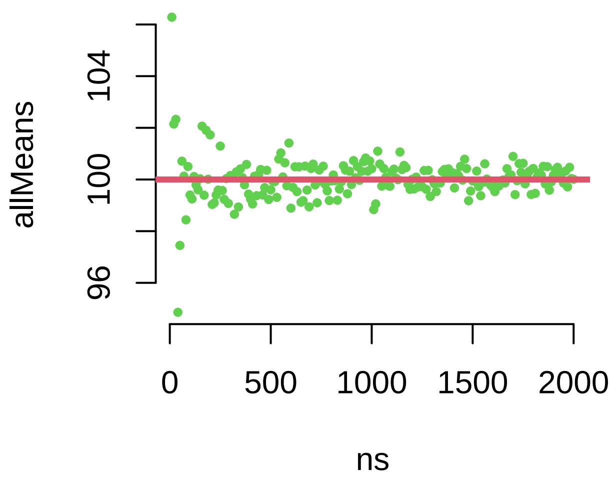


FIGURE 3.3: The sample mean from n bets at the roulette wheel.

As before, the sample mean gets closer to the true mean of the distribution, which is $p = 18/37$, which is less than 0.5, so if we keep placing bets we will end up losing money. In fact, we can check how much we will lose if we bet $1 each time, and do it 10,000 time

```
n <- 10000
roulette <- rbinom(n, 1, 18/37)
cat("Prob. win=", mean(roulette),"\n")

## Prob. win= 0.4898

cat("Paid: $", prettyNum(n, big.mark=","), ". Won: ", sum(roulette), "times.", "To-
tal    gain:",    prettyNum(2*sum(roulette),    big.mark    =    ","),    "dol-
lars. Net gain/loss:", prettyNum(2*sum(roulette)-n,big.mark = ","),"\n")

## Paid: $ 10,000 . Won:  4898 times. Total gain: 9,796 dollars. Net gain/loss: -
204
```

To the casino it doesn't matter if one person bets $1,000,000 or if 1,000 people each bets on $1,000 – in both cases the casino will win with probability 19/37.

3.2.3 Example #3 – receiving test messages

Let's simulate n days, and in each one count the number of messages, if they arrive independently from a Poisson distribution with rate=10. In each day we count messages over 12 hours, so notice that how we multiply by 12 in the code below.

```
ssize <- c(5,seq(10,1000,by=10))
myMsg <- rep(0,length(ssize))
set.seed((40001))
for (i in 1:length(ssize)) {
    myMsg[i] <- mean(12*rpois(ssize[i], 10))
}
plot(ssize, myMsg,pch=17,col="blue", axes=FALSE)
axis(1); axis(2)
abline(h=12*10, lwd=2,col=2)
```

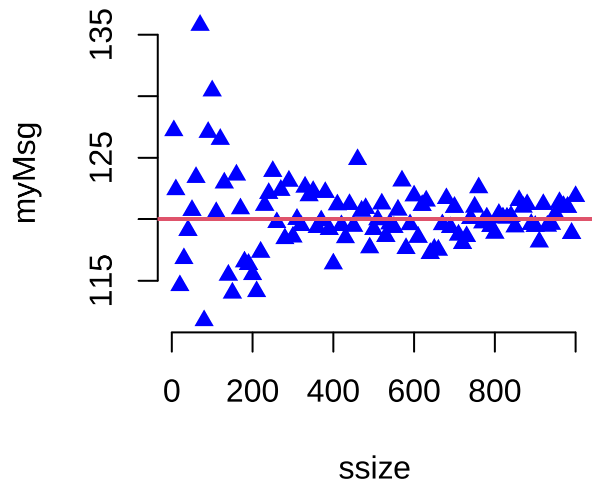


FIGURE 3.4: Simulated text messages.

If we continue to get text messages at this fixed rate over a long period of time, and we calculate the average daily messages (from 8am to 8pm) we see that that average converges to 12*10=120 messages per day.

3.2.4 The LLN – an intuitive statement of the theorem

Informally, the law of large number states that the results of an experiment tend to settle down to a fixed average when the experiment is repeated many times. In other words, as the sample size increases, the sample mean will get closer to the true population mean.

The LLN is very general – it doesn't matter if the question is about how many boy-births we expect to see ('binomial distribution') or if we measure IQ ('normal distribution'). The only requirements are that the samples are independent, and identically distributed, and that the true distribution actually has a mean. (We will see an example where this is not the case.)

It is important to emphasize that the LLN does not imply that the results of future experiments will balance out what already happened! If we bet 100 times at the roulette wheel and lost each time, this does not mean that we have a better chance of winning the next bet. All the LLN is saying is that if we continue to play many times, the overall probability that we win in each round will converge to 18/37.

3.2.5 Example #4 – when the LLN doesn't work

We have two groups of students who have to take a math test. Group A has a mean score of 70 and a standard deviation of 10, while group B has mean 60 and standard deviation 20. Both groups have a normal distribution. We take a sample of size n from each group and calculate the sample means. Then, we calculate the difference between the sample means, $d_{AB} = \bar{x}_A - \bar{x}_B$, and the ratio between the means, $r_{AB} = \bar{x}_A / \bar{x}_B$. As before, we increase the sample size and see how it affects d_{AB} and r_{AB}.

```r
n <- seq(50,5000,by=10)
set.seed(59112)
allDiffs <- rep(0,length(n))
allRatios <- rep(0,length(n))
for (i in 1:length(n)) {
  sA <- rnorm(n[i],70,10)
  sB <- rnorm(n[i],60,20)
  dAB <- sA - sB
  rAB <- sA / sB
  allDiffs[i] <- mean(dAB)
  allRatios[i] <- mean(rAB)
}
par(mfrow=c(1,2))
plot(n, allDiffs,pch=19,col=3, xlab="n", ylab="Diff.", cex=0.5)
abline(h=10,col=2,lwd=2)
plot(n, allRatios,pch=19,col="orange", xlab="n", ylab="Ratio", cex=0.5)
abline(h=70/60,col=2,lwd=2)
```

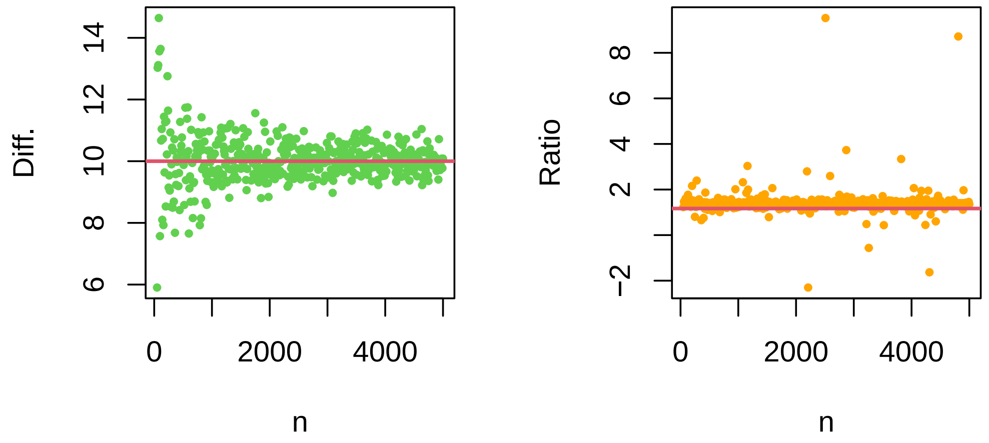


FIGURE 3.5: Simulated test scores - difference and ratio between two groups.

```
par(mfrow=c(1,1))
```

What we see here is that the mean difference between the groups, d_{AB}, converges to the true difference between the groups (10), but the ratio, r_{AB}, does not converge – n can be as large as we want, and we will still see ratios that are quite extreme. (Note that the mean of a ratio between two distributions is generally not equal to the ratio of the means of the distributions, so we don't expect r_{AB} to converge to exactly 70/60.)

What is happening here? The answer is that while a difference between two normal random variables is still normal (and hence, has a mean), the ratio between two normal distributions follows a Cauchy distribution which doesn't have a theoretical mean, so the LLN does not apply to r_{AB}.

Although the LLN applies to almost any distribution we will ever encounter, the lesson here is that taking the ratio between two perfectly well-behaved distributions may lead to a distribution to which the LLN does not apply. Other manipulations of random variables may lead to similar results.

3.3 The Central Limit Theorem

The LLN says that as we increase the sample size, the sample mean converges to a fixed value (the true population mean). It does not say anything, however, about how close the sample estimate is to the true value. In other words, it says nothing about our level of (un)certainty.

The central limit theorem (CLT) says that, in many (most) situations, when independent random variables are averaged, their normalized mean tends toward a normal distribution. This is true even if the original variables themselves are not normally distributed!

The CLT requires very little about the actual distribution of the data. The sample has to be i.i.d., and the distribution must have a finite variance.

3.3.1 CLT – Examples

We will draw a random sample of size n from the following distributions:

- Exponential with rate parameter 5.5 (use `rexp(n, 5.5)`).
- Binomial with probability 0.15.

In each case, we will plot a histogram of the sample, and overlay the probability density function on top of it. We will repeat it 10,000 times and each time calculate the sample mean. Then, we will plot a histogram of the 10,000 sample means.

3.3.1.1 Exponential distribution

```
# one iteration to show the actual distribution of the sample (not normal)
n <- 100
samp <- rexp(n, 5.5)
hist(samp, freq = F, breaks=20,main="", xlab="x", border="white") # show the em-
pirical distribution
xs <- seq(0, max(samp), length=1000)
lines(xs, dexp(xs,5.5), lwd=3, col=2) # show the true distribution
```

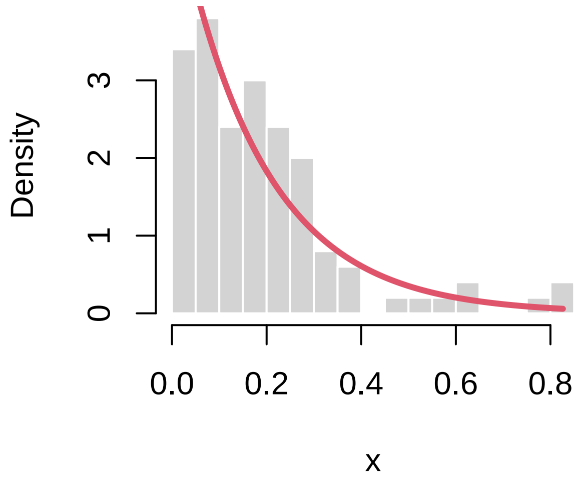


FIGURE 3.6: Simulated data from an exponential distribution.

Notice that the distribution is very skewed, and not at all like a normal distribution. To understand the CLT, we will compare the distribution of the 10,000 means for two different sample sizes, 50 and 500.

```
# 10000 iterations - show the distribution of the sample means:
n <- 50
allMeans <- rep(0, 10000)
for (i in 1:10000) {
  samp <- rexp(n,5.5)
  allMeans[i] <- mean(samp)
}
par(mfrow=c(1,2))
m <- min(allMeans)
M <- max(allMeans)
hist(allMeans, freq = F, breaks=40, xlim=c(m, M), main="", xlab="Sample mean", bor-
der="white", col="orchid")
abline(v=1/5.5, lwd=3, col="green")

# repeat, this time with a larger sample size:
# 10000 iterations - show the distribution of the sample means:
n <- 500
allMeans <- rep(0, 10000)
```

```
for (i in 1:10000) {
  samp <- rexp(n,5.5)
  allMeans[i] <- mean(samp)
}
hist(allMeans, freq = F, breaks=40, xlim=c(m, M), main="", xlab="Sample mean", bor-
der="white", col="orange")
abline(v=1/5.5, lwd=3, col="green")
```

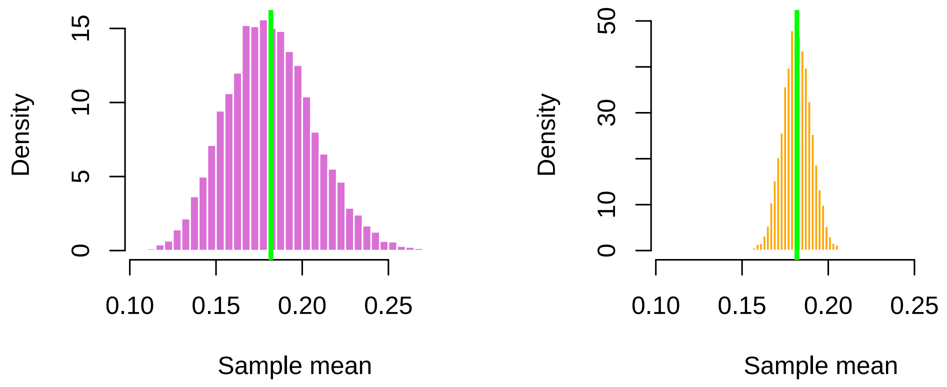


FIGURE 3.7: The distribution of the sample mean from 10,000 simulated data from an exponential distribution. Left: $n = 50$, right: $n = 500$

```
par(mfrow=c(1,1))
```

The mean is estimated quite accurately in both cases, and the overall shape of the 10,000 sample means has the shape of a normal distribution. However, the spread (variance) of the distribution decreases as the sample size increases.

3.3.1.2 Binomial distribution

In this example, our data come from a discrete and skewed distribution.

```
# 10000 iterations - show the distribution of the sample means:
n <- 50
allMeans <- rep(0, 10000)
```

```
for (i in 1:10000) {
  samp <- rbinom(n,1,0.15)
  allMeans[i] <- mean(samp)
}
par(mfrow=c(1,2))
m <- min(allMeans)
M <- max(allMeans)
hist(allMeans, freq = F, breaks=40, xlim=c(m, M), main="", xlab="Sample mean", bor-
der="white", col="orchid")
abline(v=0.15, lwd=3, col="green")

# repeat, this time with a larger sample size:
# 10000 iterations - show the distribution of the sample means:
n <- 500
allMeans <- rep(0, 10000)
for (i in 1:10000) {
  samp <- rbinom(n,1,0.15)
  allMeans[i] <- mean(samp)
}
hist(allMeans, freq = F, breaks=40, xlim=c(m, M), main="", xlab="Sample mean", bor-
der="white", col="orange")
abline(v=0.15, lwd=3, col="green")
```

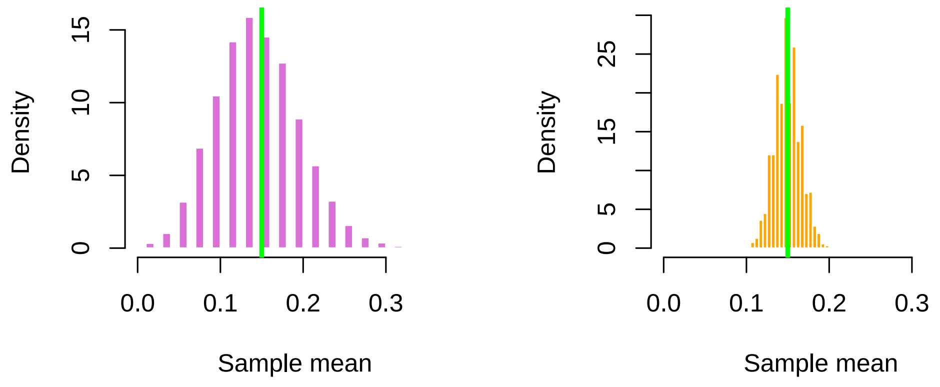


FIGURE 3.8: The distribution of the sample mean from 10,000 simulated data from a binomial distribution. Left: $n = 50$, right: $n = 500$

```
par(mfrow=c(1,1))
```

Notice that because p=0.15 is quite small, when n=50 the shape of the distribution of sample means is still a bit skewed, but when n is large the bell shape appears!

3.3.1.3 Application to child births – boy/girl ratio

In 2021 the boy to girl birth ratio in the USA is estimated to be 1.05. If we took a random sample of 300 newborns across the USA, how many do you expect to be boys?

```
b2g <- 1.05 # B/G
# We need B/(B+G)
# Since B/G=1.05, B=1.05G, so:
pboy <- 1.05/(1+1.05)
n <- 300
nsim <- 1
set.seed(100091)
boysInSample <- rbinom(nsim, n, pboy)
cat("Prob.  boy:",  pboy,  ".  Simulated  number  of  boys  in  a  sam-
ple of 300 is:", mean(boysInSample),"\n")

## Prob. boy: 0.5121951 . Simulated number of boys in a sample of 300 is: 151
```

Would it be very surprising if we actually see 148 in the random sample? Suppose we took a sample of 3,000, instead. Would is be surprising if we observed 1,480 boys?
How about if we took a sample of 30,000, instead. Would is be surprising if we observed 14,800 boys?

The probability of a boy birth does not depend on the sample size, and we calculated it to be 0.512, so the expected number of boys when the sample size is 300, 3000, and 30000 is 151, 1510, and 15100, respectively. So, it may appear that actually observing 148 instead of 151 is as likely as observing 1480 instead of 1510, or 14800 instead of 15100, but that is not true! The CLT tells us that when the sample size increases, the dispersion of the sample means around the true mean gets smaller and smaller. This is demonstrated in the code below. The green vertical line represents the expected number of boys, and the red one represents the observed ones. Notice that as we increase the sample size, the likelihood of the observed number of boys (148, 1480, 14800) gets smaller. When $n = 30000$ the red line is well outside the range of the simulated sample means.

```
n <- 300
nsim <- 10000
par(mfrow=c(1,3))
set.seed(100091)
boysInSample <- rbinom(nsim, n, pboy)
hist(boysInSample, breaks=30,border="white", xlim=c(130,190), freq=FALSE, main="n=300")
abline(v=pboy*n, col=3, lwd=3)
abline(v=148, col=2, lwd=2)

# sample size is 10 times larger:
boysInSample2 <- rbinom(nsim, 10*n, pboy)
hist(boysInSample2, breaks=30, border="white", xlim=c(1400,1700), freq=FALSE, main="n=3,000")
abline(v=pboy*10*n, col=3, lwd=3)
abline(v=1480, col=2, lwd=2)

# sample size is 100 times larger than the original example:
boysInSample3 <- rbinom(nsim, 100*n, pboy)
hist(boysInSample3, breaks=30,border="white", xlim=c(14500,16000), freq=FALSE, main="n=30,000")
abline(v=pboy*100*n, col=3, lwd=3)
abline(v=14800, col=2, lwd=2)
```

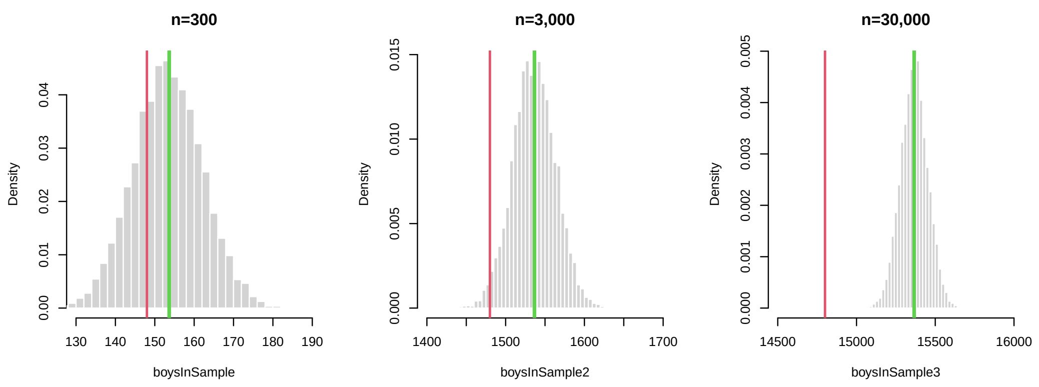


FIGURE 3.9: Number of boy births.

```
par(mfrow=c(1,1))
```

3.3.2 Summary

The LLN and CLT are very powerful results. Keep in mind that in real-life when we do not know the true distribution, we can usually get just one sample, not 10,000 like we did in our simulations. However, these theorems tell us that as long as this one sample is large enough, we can get a very good approximation for the distribution of the sample mean, if we could get multiple samples. Not only that, but regardless of the distribution of the data, the sample mean will have a normal distribution with mean which is equal to the true mean of the original distribution (even if it's skewed, or discrete), and the variance of the sample mean will shrink as we increase the sample size. This is very convenient for testing hypotheses and constructing confidence intervals (next chapter).

To conclude, let us revisit the IQ example, where we assumed that in the general population IQ scores have a normal distribution with mean=100 and standard deviation =15. We drew a sample of 200 people, and calculated the sample mean and standard deviation, which turned out to be 99.4, and 15.6, respectively. The red curve shows the true density function, and the dashed green one shows the one estimated from the sample of 200 people. They are very close.

```r
set.seed(95473)
n <- 200
samp <- rnorm(n, 100, 15)
hist(samp,     freq=FALSE,     main="",     xlab="IQ     score",     bor-
der="white", col="lightblue")
xs <- seq(40,180, length=1000)
lines(xs, dnorm(xs, 100, 15), col=2, lwd=3)
lines(xs, dnorm(xs, mean(samp), sd(samp)), col=3, lwd=3, lty=2)
```

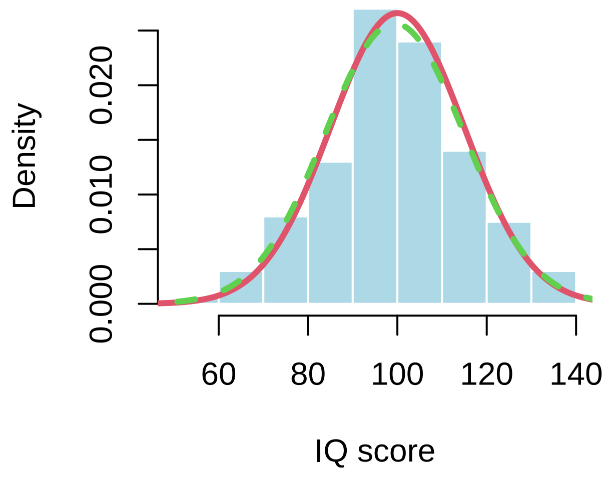


FIGURE 3.10: Simulated IQ scores.

To answer the question how likely it is to find a person with IQ greater than 150, we can calculate the area of the right tail of the distribution:

```
cat(1-pnorm(150, mean=mean(samp), sd=sd(samp)),"\n")

## 0.0005699322
```

A very select group, indeed.

In order to be accepted to the Mensa club a person has to be in the top 2% of the IQ distribution. In order to find the minimum score required in order to be a Mensa member, we use the `quantile` function, if we want to use the data, or the `qnorm` function to get the threshold from distribution of the entire population:

```
mensacutoff <- round(quantile(samp,probs = 0.98))
cat("Sample quantile:", mensacutoff,"\n")

## Sample quantile: 131

cat("Population quantile:", qnorm(0.98, mean=mean(samp), sd=sd(samp)),"\n")

## Population quantile: 131.3356
```

We can show it graphically:

```
xs <- seq(40,180, length=1000)
plot(xs, dnorm(xs, 100, 15), col=2, lwd=2, type='l', axes=F, ylab="", xlab="IQ")
axis(1); axis(2)
xx <- seq(qnorm(0.98, mean=mean(samp), sd=sd(samp)), 200, length=20)
yy <- dnorm(xx, 100, 15)
polygon(c(xx, rev(xx)), c(rep(0,length(xx)), rev(yy)), col="green", bor-
der="green", lwd=2)
```

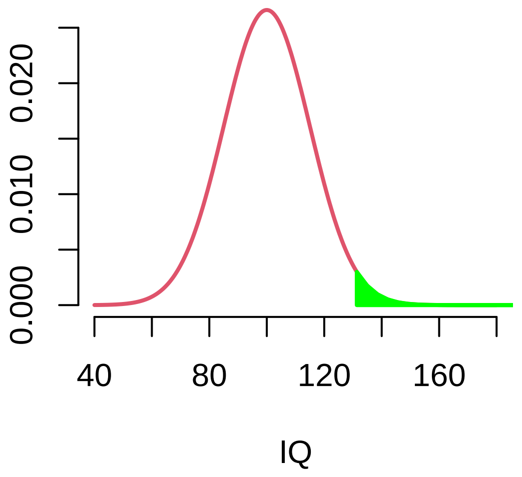


FIGURE 3.11: The distribution of IQ's and the acceptance range to the Mensa club.

In chapter 6 we will see how the CLT can be use to form hypotheses and construct confident intervals. For example, we will be able to answer questions such as: "You are given the IQ test results of 13 people and they are: 139, 104, 115, 151, 116, 141, 117, 105, 134, 155, 130, 139, 121. Is this group different than the overall population?"

4 Estimation: A Hide-and-Seek Game with the Nature

4.1 Introduction

Consider this game setup. We have a random sample of size n from a normal distribution $N(\mu, 1)$, where μ is unknown, and we want to estimate μ with this sample. An estimator is a quantity constructed from the observed sample, that is, it is a statistic. What estimators can we construct? How do we assess which is better?

Let's set up a true μ that no one knows and generate a random sample of size $n = 20$.

```
set.seed(123) # each student can contribute a digit to make mu really unknown
mu <- runif(1, min =- 10, max = 10)
n <- 20
x <- rnorm(n, mean = mu, sd = 1)
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -5.976  -4.520  -3.877  -3.946  -3.046  -1.720
```

4.1.0.0.1 Estimators

Let us consider a few candidate estimators

- Sample mean
- Sample median
- Midrange

The three estimates based on the observed sample are

```
(estimates <- c(mu1 = mean(x), mu2 = median(x), mu3 = mean(range(x))))


##       mu1       mu2       mu3
## -3.945533 -3.877437 -3.847917
```

4.1.0.0.2 Mean Squared Error

To assess which estimator is better, we compare them with the true value of μ. The squared error of an estimator $\hat{\mu}$ of μ is $(\hat{\mu} - \mu)^2$. For the three estimates based on the given sample, we have:

```
(estimates - mu)^2


##        mu1        mu2        mu3
## 0.09175876 0.13765041 0.16042665
```

Note that this comparison is for only the given sample. A good estimator may by chance behaves worse than a bad estimator. A real assessment of the quality of the estimator should be based on a large number of replicates, where we can check which estimator performs the best on average. To do so, we need to play this game repeatedly, collect the squared errors of the estimators, and compare their means, that is, mean squared error.

Illustration 4.1.1 (Hide-and-seek with a normal population)
Now the game becomes a racing game. We do it through a simulation study. Let us construct a function to do replicate of such game.

```
do1rep <- function(n, mu) {
    ## generate data
    x <- rnorm(n, mean = mu, sd = 1)
    ## collect estimates
    est <-  c(mu1 = mean(x), mu2 = median(x), mu3 = mean(range(x)))
    ## return squared error
    return((est - mu)^2)
}
```

Each time we call this function, the experiment is done and the squared errors of the three estimators are returned.

```
do1rep(n, mu)


##        mu1        mu2        mu3
## 0.17323711 0.07180305 1.16285198


do1rep(n, mu)


##        mu1        mu2        mu3
## 0.02543683 0.02323603 0.03439818
```

Which estimator is winning the game? Let's repeat the experiment 1000 times and compare the MSE.

```
nrep <- 1000
sim <- replicate(nrep, do1rep(n, mu))
rowMeans(sim)


##        mu1        mu2        mu3
## 0.05115881 0.07603630 0.13703344
```

The first estimator, the sample mean, is a clear winner! □

Illustration 4.1.2 (Hide-and-seek with a Cauchy population)
Now let's change the distribution of the population from normal to Cauchy.

```
set.seed(1979)
mu <- runif(1, min =- 10, max = 10)
x <- rcauchy(n, location = mu)
summary(x)


##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.871   7.999   8.884  10.031   9.821  25.848


do1rep <- function(n, mu) {
    ## generate data
    x <- rcauchy(n, location = mu)
    ## collect estimates
    est <-  c(mu1 = mean(x), mu2 = median(x), mu3 = mean(range(x)))
    ## return squared error
    return((est - mu)^2)
}

sim <- replicate(nrep, do1rep(n, mu))
rowMeans(sim)

##          mu1          mu2          mu3
## 3.133503e+04 1.430156e-01 3.128879e+06
```

Which estimator is the winner this time? □

Illustration 4.1.3 (Hide-and-seek with a uniform population)
Now let's change the distribution of the population to one with light tails.

```
set.seed(1975)
mu <- runif(1, min =- 10, max = 10)
x <- runif(n, mu - 1, mu + 1)
summary(x)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3267  0.8011  1.4551  1.2868  1.6794  2.1342

do1rep <- function(n, mu) {
    ## generate data
    x <- runif(n, mu - 1, mu + 1)
    ## collect estimates
    est <-  c(mu1 = mean(x), mu2 = median(x), mu3 = mean(range(x)))
    ## return squared error
    return((est - mu)^2)
}

sim <- replicate(nrep, do1rep(n, mu))
rowMeans(sim)

##         mu1         mu2         mu3
## 0.015830990 0.041249277 0.004328254
```

Which estimator is the winner this time? □

Can we find an estimator that performs the best in all scenarios?

Now that the performance depends on the true population which is unknown, can we make some good guesses and choose one? This is known as adaptive estimation.

The location parameter of a distribution controls the location of the distribution.

4.2 Seeking strategies

To play the hide-and-seek game better, we need to select a good strategy. From the examples in the last section, it seems that which strategy is better depends on the underlying truth, which is unknown.

Let's start with an example.

Illustration 4.2.1 (Likelihood in guessing jars)
There are 9 jars. The first jar has 9 red and 1 green balls; the second jar has 8 red and 2 green balls; $\cdots\cdots$; the 9th jar has 1 red and 9 green balls. Now, one jar is randomly selected, from which one ball is randomly selected, and the ball is red. Which jar was the one that was selected?

There are nine possibilities. If the selected jar were the first one, the probability that a randomly selected ball is red is 9/10. If the selected jar were the second one, the probability that a randomly selected ball is red is 8/10. Continue until the 9th jar. With this calculation, which jar should we guess?

The first one! $\square$

The principle we use here is the so-called maximum likelihood estimation. The unknown true has 9 possibilities. We choose the one that gives the highest likelihood of observing a randomly selected ball being red.

4.3 Uncertainty

Now let's play a game to appreciate the importance of uncertainty in estimation. We will use the asymptotic theory to construct the 95% confidence intervals and check their actural coverage rates.

Illustration 4.3.1
Let's generate some θ that we do not know.

```
set.seed(20210623)
theta  <- runif(1, 1, 10)
n <- 50
x <- rgamma(n, shape = theta/2, scale = 2)
ciclt <- function(x, alpha = .05) {
    xbar <- mean(x)
    ss <- sd(x)
    z <- qnorm(1 - alpha / 2)
    dd <- z * ss / sqrt(length(x))
    c(xbar - dd, xbar + dd)
}
ciclt(x)

## [1] 2.089077 3.249442

x <- rgamma(n, shape = theta/2, scale = 2)
ciclt(x)

## [1] 2.130422 3.616720

x <- rgamma(n, shape = theta/2, scale = 2)
ciclt(x)

## [1] 1.985616 3.233137
```

Does a 95% onfidence interval constructed this way really give 95% probability of covering the truth? By theory, it should when the sample size is large. Is $n = 50$ large enough for this to be good? We can check the actual coverage in a simulation study.

```
nrep <- 1000
do1rep <- function(n, theta, alpha = .05) {
    ## generate data
    x <- rgamma(n, shape = theta / 2, scale = 2)
    ## return the confidence interval
    ciclt(x, alpha)
}
```

```
sim <- replicate(nrep, do1rep(n, theta))
mean(sim[1, ] < theta & sim[2,] > theta)

## [1] 0.945

## let's try a smaller sample size
sim <- replicate(nrep, do1rep(20, theta))
mean(sim[1, ] < theta & sim[2,] > theta)

## [1] 0.895
```

The simulation shows that the confidence intervals constructed from the large sample theory works well for sample size $n = 50$ but not so well for $n = 20$. □

Here we construct bootstrap confidence interval for estimating the population median of a gamma distribution.

```
set.seed(2021)
theta <- runif(1, 1, 10)
myBootCI <- function(x, alpha = .05, B = 400, fun = mean) {
    ## mb <- replicate(B, mean(sample(x, size = length(x), replace = TRUE)))
    mb <- rep(0, B)
    for (i in 1:B) {
        xb <- sample(x, size = length(x), replace = TRUE)
        mb[i] <- fun(xb)
    }
    quantile(mb, c(alpha / 2, 1 - alpha / 2))
}

n <- 50
x <- rgamma(n, shape = theta / 2, scale = 2)
myBootCI(x, fun = mean)

##     2.5%    97.5%
## 3.603285 5.368040

nrep <- 1000
do1rep <- function(n, theta, alpha = .05, B = 400, fun = mean) {
    ## generate data
    x <- rgamma(n, shape = theta / 2, scale = 2)
    ## return the confidence interval
    myBootCI(x, alpha, B, fun)
}

sim <- replicate(nrep, do1rep(n, theta, fun = mean))
mean(sim[1, ] < theta & sim[2,] > theta)


## [1] 0.933


## let's try a smaller sample size
sim <- replicate(nrep, do1rep(20, theta))
mean(sim[1, ] < theta & sim[2,] > theta)


## [1] 0.899
```

The same code could be used to estimate other target, say the population median.

```
## Let's estimate the population median with sample median
sim <- replicate(nrep, do1rep(50, theta, fun = median))
pmed <- qgamma(0.5, shape = theta / 2, scale = 2)
mean(sim[1, ] < pmed & sim[2,] > pmed)


## [1] 0.947
```

5

Data Collection and Selection Bias

Example 7. (The 1936 Literary Digest Poll)

- For the 1936 presidential election, the Literary Digest predicted that Alfred Landon would get 57% of the vote against Franklin D. Roosevelt's 43%, while the actual results were 62% for Roosevelt against 38% for Landon.
- The Literary Digest poll was one of the largest and most expensive polls with a sample size of around 2.4 million people!
- At the same time, George Gallup was able to predict a victory for Roosevelt using a much smaller sample of about 50,000 people.
- Literary Digest's sampling method: Based on every telephone directory in the United States, lists of magazine subscribers, rosters of clubs and associations, and other sources, a mailing list of about 10 million names was created. Every name on this lest was mailed a mock ballot and asked to return the marked ballot to the magazine.

Discuss the two problems of Literry Digest Poll.

5.1 Berkson's bias

The following impressions are common, but are then really true?

- Hollywood ruins good books.
  - Popular books that were made into terrible movies[1]

[1] https://www.yardbarker.com/entertainment/articles/popular_books_that_were_made_into_terrible_movies/s1__29971472

  - Bad Books That Made Great Films[2]

- Handsome men are such jerks.
- More talented people are less attractive.
- Smarter students spend less time on studying.
- The list can go on forever ...

The impressions listed above are obtained based biased data related to Berkson's bias. Let's start with an example to explain.

Example 8. (Stumps display) Suppose you have 1000 postage stamps: 300 are pretty, 100 are rare, and 30 are both pretty and rare. Is a rare stamp more likely to be pretty? Now you want to show some of your stamps to your friends, you probably want to show them the pretty ones and the rare ones. Lets say you only show the 370 stamps which are either pretty or rare to your friends. Based on what your friends observed, will they conclude that a rare stamp more likely to be pretty?

Let's calculate the numbers now: For all the stamps, the probability that a stamp is pretty is 300/1000=0.3; for rare stamps, the probability that a stamp is pretty is 30/100=0.3. A rare stamp is not more likely to be pretty.

For the stamps your friends saw, the probability that a stamp is pretty is 300/370=0.81; for rare stamps, the probability that a stamp is pretty is 30/100=0.3. Your friends' conclusion: a rare stamp is much less likely to be pretty!

The reason for their incorrect conclusion is that the 370 stumps are not a representative sample of all the stamps you have. The sample is biased.

Now let's simulate some data to further illustrate.

```r
set.seed(2021)
n  <- 1000
book  <- rnorm(n)
movie <- rnorm(n)
plot(book, movie) # plot all the books and movies
abline(lm(movie~book))
```

[2] https://www.cinelinx.com/movie-news/movie-stuff/bad-books-that-made-great-films/

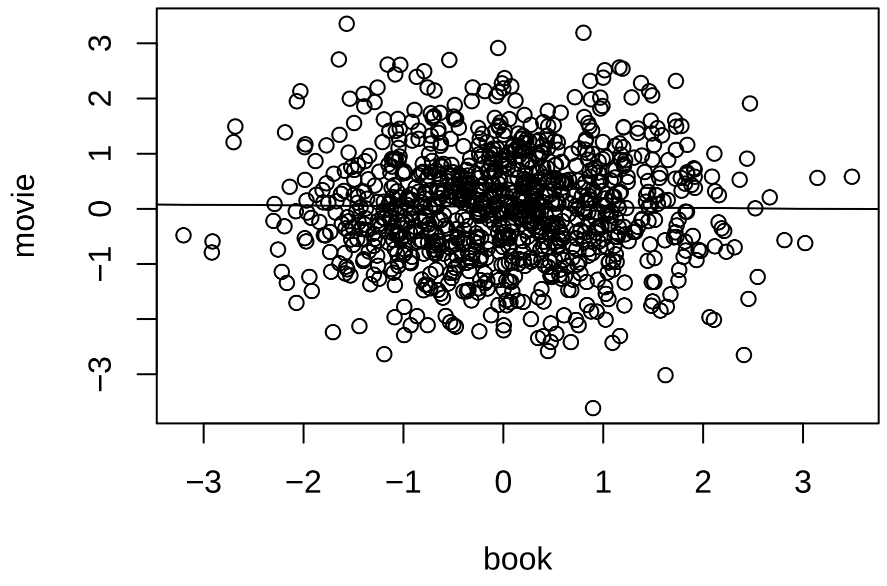


FIGURE 5.1: No linear association between goodness of movie and goodness of book

```
cor(book, movie)


[1] -0.01178458
```

```
good.b <- book > quantile(book, 0.9)
good.m <- movie > quantile(movie, 0.9)
good  <- good.b | good.m
plot(book[good], movie[good]) # plot only the good books or good movies
abline(lm(movie[good]~book[good]))
```

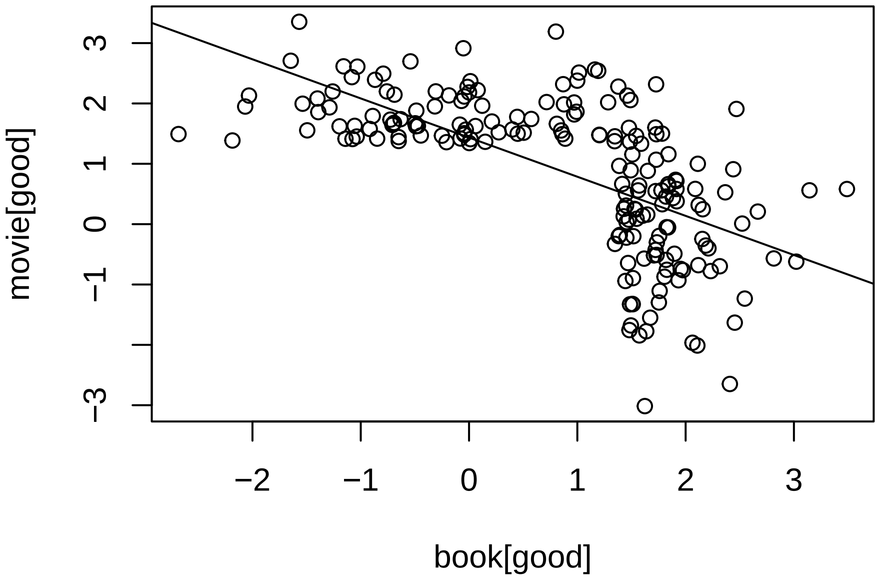


FIGURE 5.2: Negative linear association for top 10 percent books and/or movie

```
cor(book[good], movie[good])

[1] -0.6394144
```

5.2 Survival Bias

During World War II, aircraft that had returned from missions were examined and the most-hit areas of the plane are given in Figure 5.3. Which areas should the army add armor to increase the survivorship of the planes? A direct intuition may suggest to reinforce areas with the most bullet holes. However, this is not a wise decision. Why? The data were collected from the aircraft that had survived their missions so they are biased. Planes that had been shot down or otherwise lost had not been counted. The bullet holes in the returning aircraft represented areas where a plane could take damage and still fly well enough to return safely. Thus, it is the areas where the returning aircraft were unscathed that need additional armor.

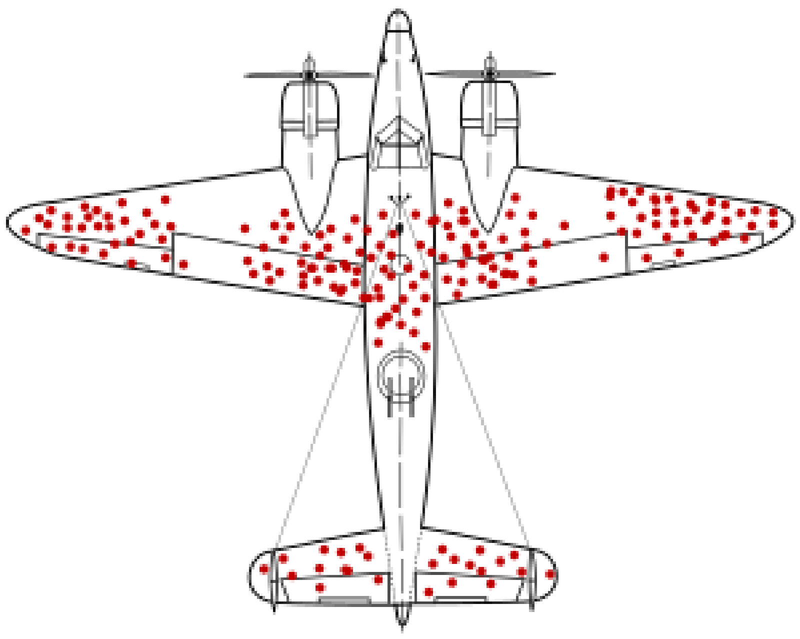


FIGURE 5.3: Most-hit areas of the returned aircraft

5.3 Common sources of biases

- Observation time interval: Early termination of a trial at a time when its results support the desired conclusion.
- Data partition: divide data with knowledge of the contents of the partitions.
- Confirmation bias: to search for, interpret, favor, and recall information in a way that confirms or supports one's prior beliefs or values. A famous is cherry picking. The picker would be expected to select only the ripest and healthiest fruits. An observer who sees only the selected fruit may thus wrongly conclude that most, or even all, of the tree's fruit is in a likewise good condition.
- Rejection of bad data on (1) arbitrary grounds, instead of according to previously stated or generally agreed criteria or (2) discarding "out-

liers" on statistical grounds that fail to take into account important information that could be derived from "wild" observations.

- Self-selection bias or a volunteer bias.
- Nonresponse bias: It is often related to sensitive questions.
- Rejection of bad data on (1) arbitrary grounds, instead of according to previously stated criteria or (2) discarding "outliers" on statistical grounds that fail to take into account important information that could be derived from "wild" observations.
- Length time bias: It often occurs in cancer screening studies. Compared with fast-growing tumors, slow-growing tumors has a longer period of time during which the cancer is in the body but not large enough to cause symptoms. Thus if the same number of slow-growing and fast-growing tumors appear in a year, the screening test detects more slow-growers than fast-growers. If the slow growing tumors are less likely to be fatal than the fast growers, the people whose cancer is detected by screening do better, on average, than the people whose tumors are detected from symptoms. This may give the impression that detecting cancers by screening causes cancers to be less dangerous.

5.4 Some basic sampling designs

- Simple Random Sampling: Every elements in the target population have equal chances to be selection. Samples are selected strictly by chance. It is probably the most effective method to prevent sampling bias. A disadvantage is the uncertainty may be large.
- Stratified Sampling: Divide members of the population into homogeneous subgroups and then use simple random sampling for each subgroups. It helps to reduce the uncertainty. Need to be careful of potential biases.
- Weighted Sampling: Let more informative elements have higher chances to be selected. It may increase the estimation efficiency, but the resulting sample may be biased so need special way of estimation.

5.5 Different sampling approaches

- Sampling with replacement: An element may be included in a sample more than once. It is possible to have replicates in the sample.
- Sampling without replacement: An element can be included in a sample at most once. There is no replicates in the sample.
- Poisson sampling: determines if each element of the population is selected in the sample independently. The sample size is random.
- Between Sampling with replacement and without replacement, which is more efficient?

Example 9. (Numerical comparisons)

- To simulation hosehold incomes, generate a population P of size N from a $P \sim \chi^2$ distribution with degrees of freedom one.
- Take samples of size n from P to estimator the population mean, say μ, using simple random sampling both with and without replacement. Compare their efficiency.
- Assume another variable $A = P + Unif(0,2)$ is available to define informative sampling weights. Evaluate the efficiency of weighted sampling both with and without replacement.

6 Hypothesis Testing

Hypothesis testing can be thought of as a stochastic analog of proof by contradiction.

"To understand the essence of statistical hypothesis testing properly, it is necessary to compare intentionally hypothesis testing with proof by contradiction and characterize the former as not the same as the latter." (**?**).

It should be relatively clear, then, why a failure to reject H , does not constitute a verification of its truth. It simply means that the researcher has failed to provide evidence suflcient to cast serious doubt on the truth of H_0. (**?**)

Misconception (**?**).

Consider testing whether two sample have the same mean.

```r
set.seed(20210628)
delta <- 0
n1 <- n2 <- n <- 30
x1 <- rnorm(n)
x2 <- rnorm(n2) + delta
t.test(x1, x2)
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = 0.6515, df = 57.364, p-value = 0.5173
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3095775  0.6082215
## sample estimates:
##  mean of x  mean of y
## 0.24627636 0.09695435
```

```r
## understand the output
wilcox.test(x1, x2) # rank-based


##
##  Wilcoxon rank sum exact test
##
## data:  x1 and x2
## W = 488, p-value = 0.5819
## alternative hypothesis: true location shift is not equal to 0
```

Let's investigate the properties of the two tests. The validity of a test is affected by the sample size n, the data-generating distribution, the deviation δ from the null.

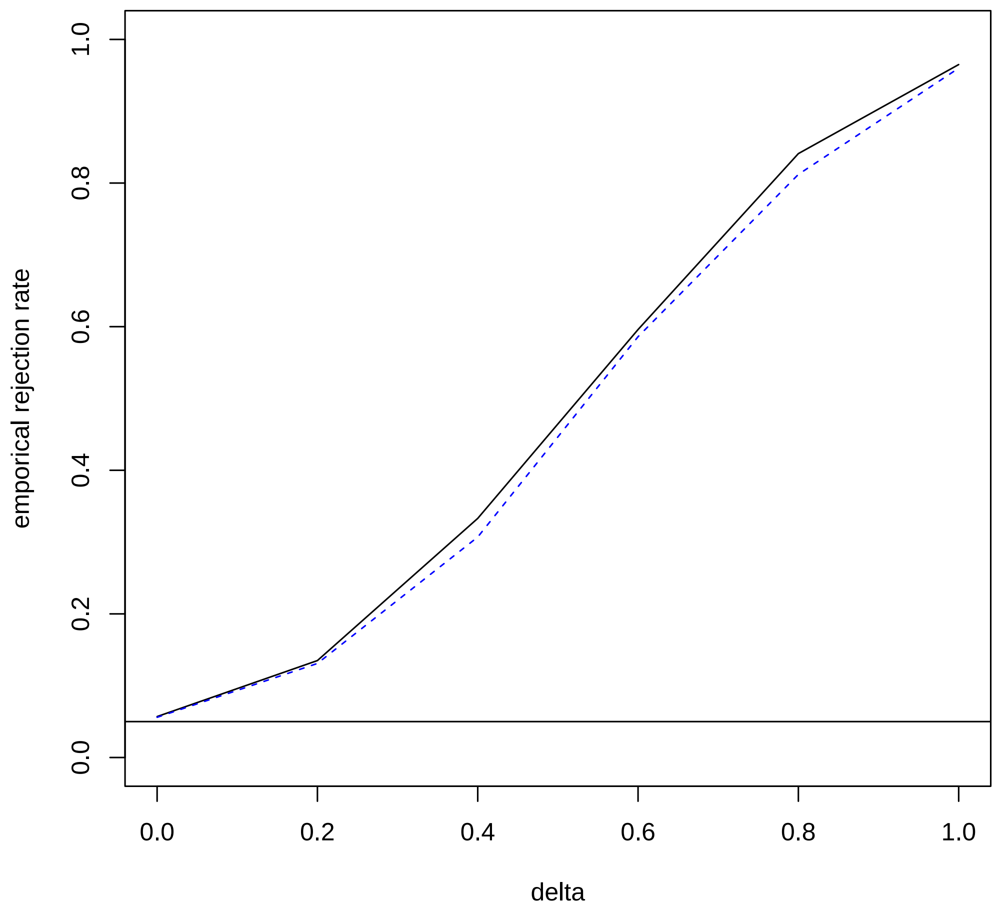
```r
do1rep <- function(n, datagen, delta = 0) {
    x1 <- datagen(n)
    x2 <- datagen(n) + delta
    p1 <- t.test(x1, x2)$p.value
    p2 <- wilcox.test(x1, x2)$p.value
    c(t = p1, wilcox=p2)
}

do1rep(30, rnorm, 0)


##         t    wilcox
## 0.4054286 0.5133481
```

Now we can check the empirical rejection rates of the tests in a simulation study.

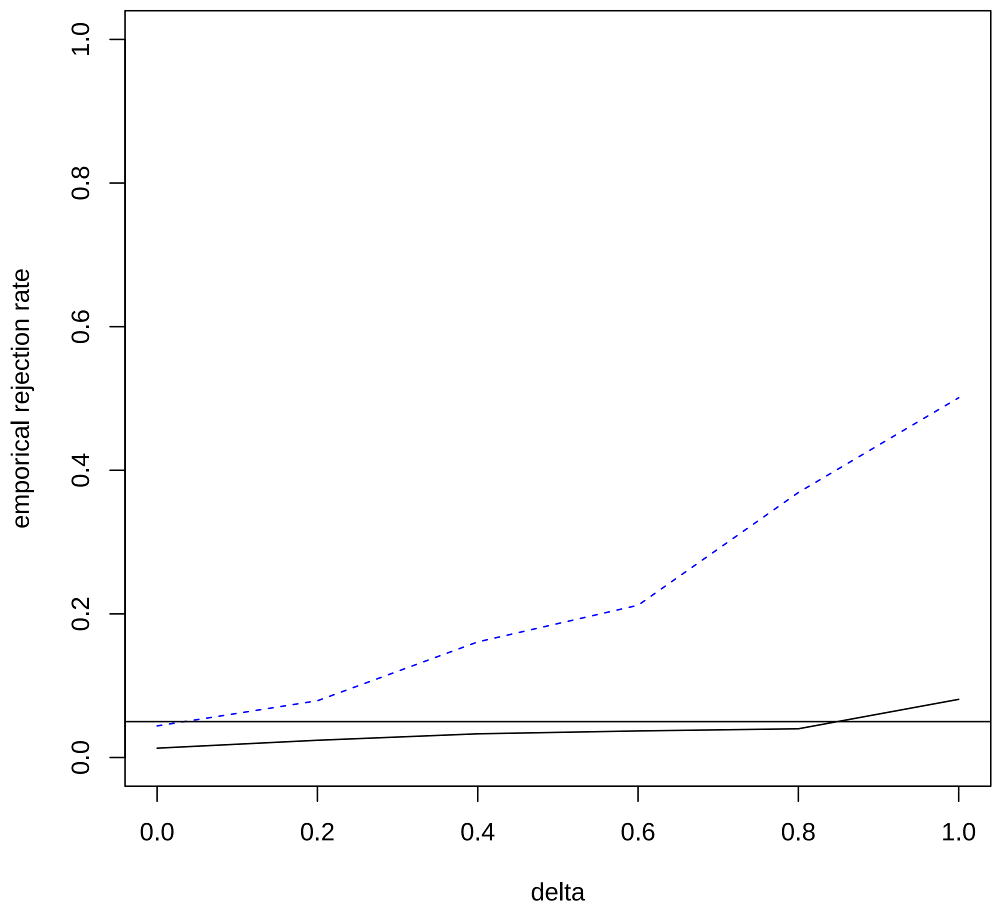
```r
nrep <- 1000
sim <- replicate(nrep, do1rep(n, rnorm, 0))
rowMeans(sim < .05)


##      t wilcox
##  0.055  0.058


## put them into a function for ease of accessing
empRejRate <- function(nrep, n, datagen, delta = 0, alpha = .05) {
    sim <- replicate(nrep, do1rep(n, datagen, delta))
    rowMeans(sim < alpha)
}

## normal population
empRejRate(nrep, n, rnorm, 0)


##      t wilcox
##  0.056  0.051


empRejRate(nrep, n, rnorm, 0.5)


##      t wilcox
##  0.501  0.489


## Cauchy population
empRejRate(nrep, n, rcauchy, 0)


##      t wilcox
##  0.020  0.047


empRejRate(nrep, n, rcauchy, 0.5)


##      t wilcox
##  0.032  0.171
```

We can draw power curves to compare two tests.

```
delta <- seq(0, 1, by = .2)
## normal distribuion
rejrate <- sapply(delta, function(x) empRejRate(nrep, n, rnorm, x))
plot(delta,  rejrate["t",  ],  type  =  "l",  ylab  =  "emporical  rejec-
tion rate", ylim = c(0, 1))
lines(delta, rejrate["wilcox", ], lty = 2, col = "blue")
abline(.05, 0)
```



```
## Cauchy distribuion
rejrate <- sapply(delta, function(x) empRejRate(nrep, n, rcauchy, x))
plot(delta,   rejrate["t",   ],   type   =   "l",   ylab   =   "emporical   rejec-
tion rate", ylim = c(0, 1))
lines(delta, rejrate["wilcox", ], lty = 2, col = "blue")
abline(.05, 0)
```



Sometimes, we don't need to resort to tests based on large sample theory. Consider the setting where we compare the two arms of treatment in a clinical trial: treatment versus placebo. Suppose that we want to show that the treatment leads to an increase in an outcome of interest. Let μ_0 and μ_1 denote the mean of the outcome under placebo and treatment, respectively. The hypotheses to be tested are

$$H_0 : \mu_1 = \mu_0, \quad \text{vs} \quad H_1 : \mu_1 > \mu_0.$$

Under H_0, the labels of the treatment arms are exchangeable. Consider test statistics T, the difference in the two sample means.

```r
xpooled <- c(x1, x2)
xperm <- sample(xpooled, size = length(xpooled), replace = FALSE)
xd <-  mean(xperm[(n1 + 1):(n1 + n2)]) - mean(xperm[1:n1])
## put in a function
myPermTest <- function(x1, x2, nperm = 1000) {
    n1 <- length(x1)
    n2 <- length(x2)
    stat <- mean(x2) - mean(x1)
    xpooled <- c(x1, x2)
    stat.sim <- replicate(nperm, {
        xperm <- sample(xpooled, size = length(xpooled), replace = FALSE)
        xd <- mean(xperm[(n1 + 1):(n1 + n2)]) - mean(xperm[1:n1])
    })
    p.value  <- mean(stat.sim >= stat)
    p.value
}

delta <- 1
x1 <- rgamma(n, shape = 2, scale = 2)
x2 <- rgamma(n, shape = 2, scale = 2) + delta
myPermTest(x1, x2)

## [1] 0.12

## try a simulation study
do1rep <- function(n, datagen, delta = 0) {
    x1 <- datagen(n)
    x2 <- datagen(n) + delta
    myPermTest(x1, x2)
}

sim <- replicate(200, do1rep(n, rnorm, delta = .5))
mean(sim < .05)

## [1] 0.645

sim <- replicate(200, do1rep(n, rcauchy, delta = 1))
mean(sim < .05)

## [1] 0.16
```

We can test on data generated from a distribution with flexible shapes, the generalized Tukey's lambda family, which is defined in terms of the inverse of the distribution function.

```r
rgtl <- function(n, lambda1 = 0, lambda2 = 1, lambda3 = 0, lambda4 = 1) {
    u <- runif(n)
    lambda1 + (u ^ lambda3 - (1 - u) ^ lambda4) / lambda2
}

sim <- replicate(200,
                 do1rep(n,
                        function(n) rgtl(n, lambda3 = 1.5, lambda4 = 5.8),
                        delta = .2))
```

Index