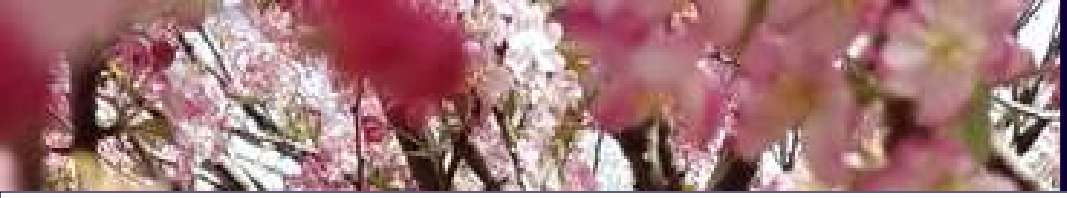

Appendix R:

Experimenting with R

Overview:

One of the keys to gaining a better understanding of statistics and randomness is to experiment with them. This chapter shows how to utilize the R Statistical Environment to perform simulations that allow you a glimpse into the wonder that is statistics.

Forsberg, Ole J. (10 DEC 2024). "Experimenting with R." In *Linear Models and Řuritâ Kràlovstvì*. Version 0.704442$\eta(\alpha)$.

Chapter Contents

❧❧❧

The purpose of this appendix is to

R.1: Installing R

The first step in using R is to install it on your computer. The process is relatively straight-forward. You download the installation program, then run the installation program.

You can find the installation program at the CRAN website:

```
https://cran.r-project.org/
```

Once there, download the appropriate installation program for your specific type of computer. Once it is downloaded, find the installation program and run it. Selecting all of the default options is fine. It's what I do.

The specifics are dependent on the type of computer operating system, so I will not get into the specifics. Let me point you in the direction of an Internet search engine and/or a site with a lot of videos. Learning how to access the information in these sites is a skill in itself.

So, enjoy the exploration and the learning.

Some prefer to install both R *and* and integrated developer environment (IDE) called R Studio. I do not. Except when doing certain niche projects, none of which we are doing in this book, R Studio just provides nothing (at best) or worse.

R.2: R Packages

While the base R that you installed has many statistical features, it is important to learn how to extend that base R installation. One way to do this is to install and load packages.

This book relies heavily on these packages:

- car
- lawstat
- lmtest
- MASS
- randtests
- snpar

Note that this book is not a part of the so-called tidyverse. Perhaps a future edition will be, but not now.

R.2.1 Installing Packages

The overall scheme for installing packages is the same for each operating system. The specifics are slightly different to take advantage of what the operating systems allow. That scheme is

1. Run the `install.packages` function
2. Select a place to download the package from
3. ????
4. Profit

For instance, if I want to install the car package, I would run the following code in the Console window

```
install.packages("car")
```

Running this will download the car package from the Internet (the mirror site you selected) and install it on your computer. [If you have already specified the mirror site, then step two will not happen.]

It will also download all packages that are needed to run the car functions (called "dependencies"). You can see what other packages are installed by watching the Console window.[1]

[1] The script window is where you type your analysis script... it is where you "show your work." The Console window is used for the quick work that is not a part of your analysis. Things like help queries and one-time-only work is done in the Console window.

R.2.2 Loading Packages

The installation needs to be done just once for your computer. However, if you need to use the package in a script, you will need to load it into the working memory. Since this is something tied to the script, you should have this line in your script if you are using the `car` package:

```
library("car")
```

Once that line is run, `R` is able to access all functions (and/or data) in the package. Usually, there are a lot of functions in a package. To see what is available in the package, run this in the Console window:

```
help(package="car")
```

After running this, a page will pop up showing all available functions and links to their help pages. This particular package has a lot of functions associated with it. Some other packages have just a couple. The number of functions depends on the purpose of the package.

R.3: R Functions

There are a plethora of functions available in R. The key is to use functions to become familiar with them. Also, because you will always need to use new functions, it is very important to be familiar with the R help files for the functions.

R.3.1 Basic Functions

The following are basic functions in R. You will use these frequently.

- `source` run external R script
- `head` show top 6 elements
- `tail` show last 6 elements
- `length` number of elements
- `seq` sequence of numbers
- `summary` six-number summary
- `mean` arithmetic average
- `median` median
- `sum` sum
- `sd` standard deviation
- `var` variance

Please know what each does. If necessary, use the help file for the given function. The more you familiarize yourself with the help files, the more they will tell you.

R.3.2 Matrix Functions

R can also do arithmetic on matrices. Since the internal computer calculations are all based on matrices, it is important to be familiar with matrix operations to make sure you know what R is doing (can check the output).

Element-wise Functions:

- `+` usual matrix addition
- `*` Hadamard product (element-wise multiplication)
- `^` element-wise exponentiation

Additional Matrix Functions:

- `%*%` usual matrix product
- `t` matrix transpose
- `solve` matrix inverse

R.3.3 Probability Functions

At the core of statistics is probability and probability distributions. These will be important in helping you better understand the effects of randomness on the estimates... and the effects of violating procedure requirements on those estimates.

- `set.seed` specify random number seed
- `sample` random sample from a vector

Probability Function Naming Logic: Each probability function in R can be parsed into two parts, the stem and the prefix. The **stem** specifies the probability distribution, whereas the **prefix** specifies what aspect of the distribution you wish to access.

This is an exhaustive list of the prefixes:

- `d`
  specifies the likelihood value. If the distribution is discrete, then this will also be the probability, otherwise it is the density.
- `p`
  specifies the *cumulative* probability, $\mathbb{P}[X \leq x]$. This prefix will rarely be available for multivariate functions.
- `q`
  specifies the quantile, the value of x that produces the probability. This prefix will rarely be available for multivariate functions.
- `r`
  specifies a random variate. In simulation, this is the most important prefix, as it produces a random sample of a given size from the specified distribution.

The second part is the stem. This specifies the distribution involved. Here is a list of some of the more interesting stems available:

- `binom`
  Binomial distribution. One needs to specify the number of trials, `size`, and the success probability, `prob`.
- `cauchy`
  Cauchy distribution. Optionally, one can specify the location and the spread. The default is the standard Cauchy with `location` of 0 and `scale` of 1.
- `exp`
  Exponential distribution. One needs to specify the `rate`, λ.
- `f`
  Snedecor's F distribution. One needs to specify both degrees of freedom, with the numerator preceding the denominator degrees of freedom, `df1` and `df2`.
- `gamma`
  Gamma distribution. One needs to specify the `shape` parameter, α. The `rate` parameter has a default value of $\sigma = 1$.
- `norm`
  Normal (Gaussian) distribution. Optionally, one can specify the mean `m` and standard deviation `s` of the Normal. The default is mean 0 and standard deviation 1.
- `pois`
  Poisson distribution. One needs to specify the expected value, `lambda`.

- `t`
  Student's t distribution. One needs to also specify the number of degrees of freedom, `df`.
- `unif`
  Continuous Uniform distribution. Optionally, one can specify the minimum and maximum value. The default is the standard Uniform with `min` of 0 and `max` of 1.

R.3.4 Testing Functions

Because R is a statistical program, it is able to perform all of the basic statistical tests and procedures. These are the related functions.

- `aov` Analysis of Variance procedure
- `lm` OLS regression
- `summary.lm` summary of a linear model
- `summary.aov` summary of an ANOVA
- `residuals` calculated residuals from a model
- `confint` confidence interval for estimated parameters
- `predict` estimation and prediction of a value
- `runs.test` the runs test
- `hetero.test` univariate test of heteroskedasticity
- `fligner.test` test of heteroskedasticity across groups
- `bptest` Breusch-Pagan test of heteroskedasticity

R.3.5 Control Functions

- `for` for-loops
- `if` if-then logic
- `numeric` creates a vector in memory

R.3.6 Graphical Functions

R is a full-fledged graphical system. In fact, this is what set R apart from its competitors (and still does!). Every pixel of a graphic can be modified in RṪhs book relies on the basic R graphic engine. There are two other graphical engines (metaphors): `grid` and `ggplot2`. Base-R graphics will always serve you. `ggplot2` is the modern graphics engine for Rİt serves as a wrapper for the basic graphics, making some graphics much easier to create.

- `qqnorm` Q-Q plot for a Normal target
- `qqline` plots the diagonal line in a Q-Q plot
- `barplot` bar chart
- `boxplot` box plot
- `hist` histogram
- `histogram` histogram that can be more easily modified
- `overlay` histogram with a overlaid density function
- `plot` basic scatter plot
- `par` specifies a graphical parameter
- `plot.new` starts a new plot
- `plot.window` specifies the viewing window
- `axis` draws an axis
- `title` writes a title on the graphic
- `lines` draws lines
- `points` draws points

R.4: Programming Practice

This section provides a series of practical examples to help you apply statistical concepts using the R Statistical Environment, one of the most versatile programming languages for statistical analysis and data visualization. R is particularly well-suited for solving a wide range of statistical problems, from basic descriptive statistics to advanced modeling techniques. As you progress through these examples, you will gain hands-on experience with R's powerful functions, libraries, and data structures, enabling you to approach statistical challenges with confidence.

I designed each example in this section to demonstrate the practical application of key statistical methods while also showcasing the capabilities of R. To help you fully understand the solutions, each example includes clear explanations, annotated code snippets, and discussions of the outputs. Even if you are new to R, the examples are structured to quickly build your proficiency with the language.

By working through these examples, you will not only deepen your understanding of statistical methods but also start developing a toolkit of practical R skills. Whether you aim to analyze data for research, business, or personal projects, these examples will equip you with the knowledge and techniques needed to leverage R effectively. Take your time to experiment with the code, explore variations, and see how the results change — this active learning approach will enhance both your statistical intuition and your programming expertise.

Note that there is a lot of white space in this section. It is there so that you can take notes directly on the examples.

Example 1

Produce a basic density plot of a Cauchy distribution between -3 and +3, where the Cauchy is centered at $x = 2$ and has an IQR of 3.

Solution: Since we are working with a probability distribution, let us refer to Section R.3.3. The function to calculate the density of the Cauchy centered at 2 with IQR 3 is `dcauchy(x,2,1.5)`. Thus, some code to produce a basic plot between -3 and +3 is

```
x = seq(-3,3,length=1000)
y = dcauchy(x, locataion=2, scale=1.5)
plot(x,y)
```

With a little bit of work, you can make a graphic like Figure R.1. The fill color is the green in a pallette of three colors designed to be safe for those with the usual color blindness, `#1b9e77`. When possible, it is best to accommodate those with color blindness and those who print out the graphic in shades of grey. ♦

The three safe colors are Green (`#1b9e77`), Orange (`#d95f02`), and Blue (`#7570b3`). These colors are from the `colorbrewer2.org` site.

Extension: Plot the pdf of a standard Normal distribution from -3 to +3 on the same graphic as a standard Cauchy. Looking at the two distributions, which has a higher variance?

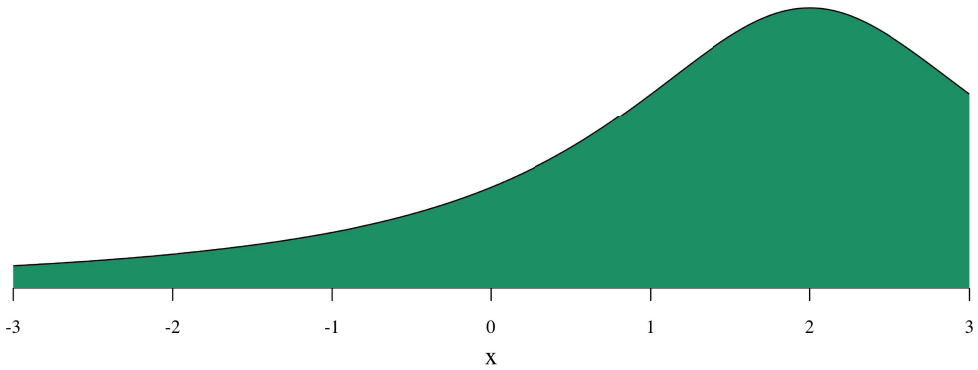

Figure R.1: *The probability density function (pdf) of a Cauchy(2,1.5) distribution.*

Example 2

Estimate a density graph of the volume of a cylinder with radius following a standard Uniform distribution and a height following a Normal distribution with mean 10 and standard deviation 0.1.

Solution: Since we are working with a probability distribution, I again refer you to Section R.3.3.

```
radius = runif(n=1e6)
height = rnorm(n=1e6, m=10, s=0.1)
volume = pi * radius^2 * height

hist(volume)
```

This is an example where using simulation easily obtains the approximate distribution. After using my R skills, I obtained the histogram in Figure R.2. See how close you can come to this. ♦

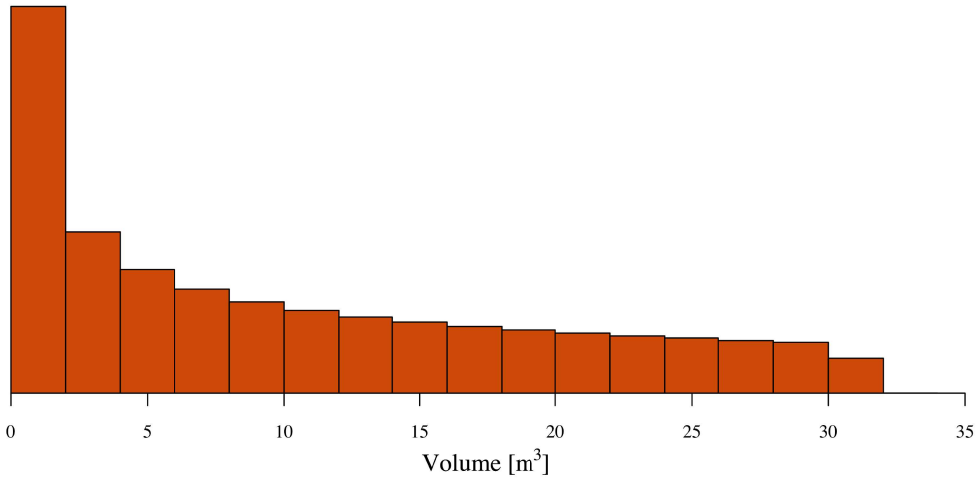

Figure R.2: *The estimated probability density function (pdf) of the volume of a cylinder.*

Example 3

Determine the effect of rounding on the appropriateness of the one-sample t-test.

Rounding can significantly affect statistical decisions because it alters the precision of numerical data (the inputs), which can lead to changes in calculated values such as means, variances, or p-values (the outputs). In hypothesis testing, for instance, rounding may cause a test statistic to fall on the border of a critical value, potentially shifting the conclusion about rejecting or failing to reject the null hypothesis. Similarly, in regression analysis, rounding predictor or response variables can affect the accuracy (and precision) of parameter estimates and the overall model fit.

These impacts are particularly pronounced in datasets with small sample sizes or values that are close to decision thresholds. Therefore, careful consideration of rounding practices is essential to ensure that statistical conclusions remain valid and reliable. This example explores the effects of rounding on the one-sample t-test.

Solution: This leaves a lot of decisions to us. The one-sample t-test requires data being generated from Normal distribution. So, let's generate data from a $\mathcal{N}(5,1)$ distribution. We need to round the data, so we will need to use the `round` function. Finally, we will need to determine if the distribution of the resulting p-values is standard Uniform (Section S.6.1).

The following code does this.

```
pval = numeric()

for(i in 1:1e4) {
 x = rnorm(10, m=5, s=1)
 y = round(x)
 pval[i] = t.test(y, mu=5)$p.value
}

ks.test(pval,"punif")
binom.test(sum(pval<0.05), n=length(pval), p=0.05)
```

The Kolmogorov-Smirnov test indicates that the distribution of the p-values is not standard Uniform. Thus, rounding in this situation breaks the t-test. Note that if we only care about $\alpha = 0.05$, we would use the Binomial test results. Given that p-value, I would still conclude that I should not use the t-test in this situation.

What about increasing the sample size from 10 to 50? The Kolmogorov-Smirnov test still indicates that the test is no longer acceptable. Note that if all we

care about is $\alpha = 0.05$, then the t-test *does* appear to be appropriate under these conditions.

What about increasing the variability in the data? Having a wider spread to the data may make the rounding less important. Let's change the standard deviation from 1 to 10 (and return the sample size to 10). From my run, the distribution of the p-value is still not standard Uniform, but the rejection rate for $\alpha = 0.05$ is close enough to 0.05.

What if we increase the sample size back to 50? In this situation, the distribution of the p-values is close enough to standard Uniform that the rounding does not affect the quality of the t-test conclusions. ♦

Example 4

If the inter-arrival time is Exponentially distributed with average time of 20 minutes, then what is the distribution of people who show up in an 8 hour period?

Algorithmic thinking is crucial in statistics as it enables a structured approach to solving complex problems by breaking them into smaller, logical steps. This mindset is particularly valuable when designing workflows for data analysis, from data preprocessing and visualization to modeling and interpretation. By thinking algorithmically, statisticians can systematically address challenges such as cleaning messy data, optimizing computational efficiency, or automating repetitive tasks.

Moreover, algorithmic thinking facilitates the translation of statistical concepts into code, allowing for reproducibility, scalability, and adaptability in analysis. In a field increasingly reliant on computational tools, developing this skill ensures that statisticians can efficiently tackle problems and adapt to new methods or technologies. This example illustrates algorithmic thinking to arrive at an interesting result.

Solution: This is a tough question but let's break it down into its parts, then simulate it.

The inter-arrival time is the time between arrivals.

```
iat = rexp(100, rate=3) ### 'iat' is in hours
```

The total time between the first and 20th arrival would be the sum of 20 of those inter-arrival times.

```
iat = rexp(100, rate=3)
sum(iat[1:20])
```

On the other hand, the number of arrivals in an hour would be

```
iat = rexp(100, rate=3)
iatCS = cumsum(iat)
max(which(iatCS <= 1))
```

So, the number of arrivals in eight hours would be

```
iat = rexp(100, rate=3)
iatCS = cumsum(iat)
```

```
max(which(iatCS <= 8))
```

To get the *distribution* of those "number of arrivals in eight hours," you just need to repeat the code many times, saving the number of arrivals each time:

```
arrNum = numeric()

for(i in 1:1e6) {
  iat = rexp(100, rate=3)
  iatCS = cumsum(iat)
  arrNum[i] = max(which(iatCS <= 8))
}
```

The histogram (Figure R.3) shows the estimated distribution of the number of customers arriving in those 8 hours.

Closely examining the histogram *and* deeply thinking about the distributions you should have already learned, it is clear that the number of arrivals in 8 hours follows this distribution

$$\text{Number of Arrivals} \sim \mathcal{P}(\lambda = 24) \tag{R.1}$$

To make it even more manifest, overlaying the histogram with a graph of that Poisson distribution illustrates this, Figure R.3.

```
hist(arrNum, freq=FALSE, breaks=seq(1,50)-0.5)
points(1:50, dpois(1:50, lambda=24), pch=16)
```

Again, I used some R programming skills to obtain the graphic at the bottom. ♦

Note that this is not *proof* of the relationship between the two distributions. It merely suggests the relationship. Your probability theory course will give you the tools to actually prove the relationship.

Extension 1: Change all of the time measurements in the previous example from hours to minutes. Make sure that the conclusions are the same.

Extension 2: Change the inter-arrival time to 10 minutes. What is the distribution of the number of arrivals in three hours? Show it using the histogram.

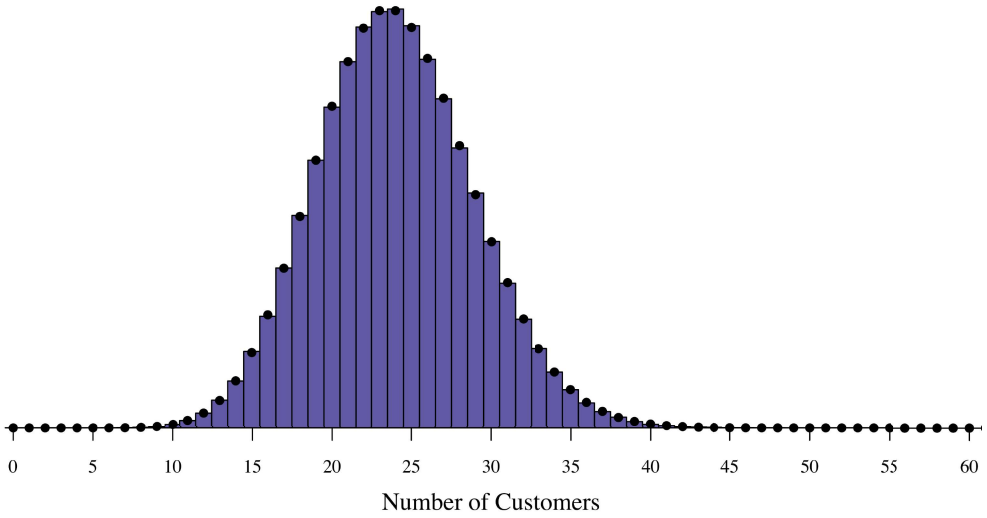

Figure R.3: *The estimated probability mass function (pmf) of the number of customers arriving in eight hours. Note that it closely follows the $\mathcal{P}(\lambda = 24)$ distribution. Using the techniques of probability theory, one can prove this relationship.*

Example 5

A flashlight uses five batteries. The lifetime of each battery is independent and follows a Gamma distribution with mean 50 days and standard deviation 5 days (`shape` = 100, `scale` = 0.5). The flashlight will show light until the first battery dies.

What is the expected time the flashlight will work after receiving five new batteries?

If we know the distribution of the lifetimes of each component, does this mean we know the distribution of the minimum lifetime? Sometimes. Note that sometimes we may be curious about the distribution of a *function* of random variables... which is also a random variable.

But, it goes beyond simple curiosity. Calculating the distribution of functions of random variables is essential because it allows us to understand how transformations or combinations of random variables behave and how their uncertainty propagates. This knowledge is fundamental in many applications, such as deriving the sampling distribution of an estimator, which forms the basis for hypothesis testing and constructing confidence intervals. Additionally, understanding the distribution of these functions enables probabilistic modeling in scenarios where direct distributions are unavailable, such as in operations research or risk management. It also helps in determining the likelihood of complex events, optimizing decision-making, and evaluating reliability in systems involving random inputs. By studying these distributions, statisticians can make more accurate predictions and draw meaningful inferences in both theoretical and applied contexts.

Solution: This is a great place for you to think through the problem (algorithmic thinking). What is the first step modeling this physical event? What is the second? Etc.? Use the next page to write out the steps... and the code. If done correctly, you will obtain the graphic at the bottom of the page, Figure R.4.

The question actually asked for the expected lifetime of the flashlight. The expected lifetime (`mean`) of the flashlight is 44.3 days, with a standard deviation of 3.1 days (`sd`). It is nice to know that 90% of the flashlights survive between 39.1 and 49.2 days (`quantile`).

♦

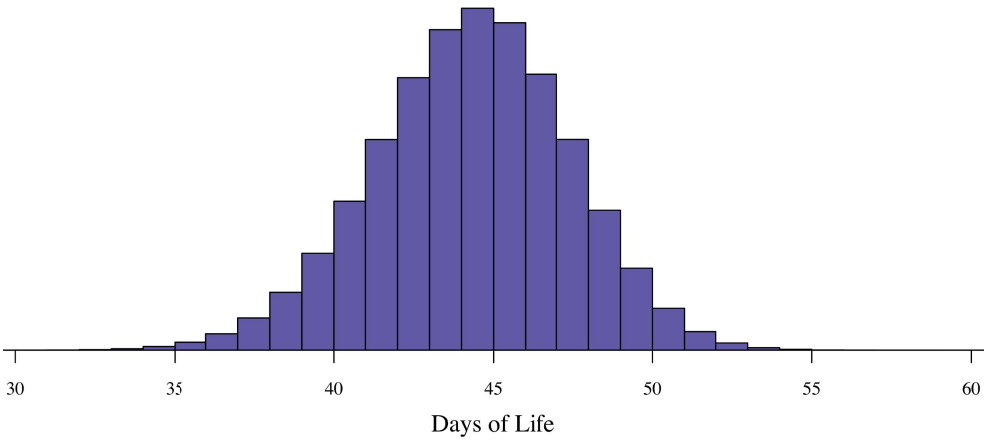

Figure R.4: *The estimated probability density function (pdf) of the lifetime of a flashlight.*

Example 6

A sample $n = 10$ counts is drawn from a population with unknown distributional characteristics. From the collected data, estimate a 95% confidence interval for the population mean.

$$4, 3, 4, 5, 4, 4, 1, 4, 9, 8, 3$$

The **bootstrap** is a powerful and versatile tool in statistics because it provides a non-parametric method for estimating the sampling distribution of a statistic without requiring strong assumptions about the underlying population distribution. By repeatedly resampling with replacement from the observed data, the bootstrap allows statisticians to approximate the variability of estimators, construct confidence intervals, and conduct hypothesis tests.

This approach is especially valuable when theoretical distributions are difficult to derive, such as with complex estimators or small sample sizes. Furthermore, the bootstrap is widely applicable across diverse statistical methods, making it a critical tool for robustness and flexibility in real-world data analysis. Its ability to leverage computational power to provide insights where traditional methods may falter has made the bootstrap indispensable in modern statistics.

Solution: Note that the sample size is small in this example. As such, we cannot rely on the Central Limit Theorem to assume the sample mean is Normally distributed. We do not even know if the population has a finite variance. The only thing we know is that we collected that particular sample. Thus, we will use the bootstrap.

That is, we will treat the sample as being representative of the population. Without this assumption, there is absolutely nothing we can do. With that assumption, we can effectively (or essentially) "recreate" the population as being an infinite repetition of this data. Then, to estimate the variability in the population, we simply redraw a sample of the same size from that population.

This is called the **non-parametric** bootstrap because it does not assume a specific (named) distribution of the data.

Here is the code to accomplish this once.

```
theData = c(4, 3, 4, 5, 4, 4, 1, 4, 9, 8, 3)

newData = sample(theData, replace=TRUE)
mean(newData)
```

This will give us one more sample mean. It takes thousands of them to understand the distribution (centers and variabilities). Thus, the non-parametric bootstrapping code will be

```
theData = c(4, 3, 4, 5, 4, 4, 1, 4, 9, 8, 3)
newMeans = numeric()

for(i in 1:1e6) {
  newData = sample(theData, replace=TRUE)
  newMeans[i] = mean(newData)
}

mean(newMeans)
sd(newMeans)
quantile(newMeans, c(0.025,0.975))
```

From this code, I estimate the population mean is 4.45, with an estimated 95% confidence interval from 3.27 to 5.82. ♦

The non-parametric bootstrap is used when all you know about the population is that you randomly selected the sample. It is a *very* flexible procedure that should be a part of your statistical toolbox.

However, statistically, it tends to be of low power (confidence intervals are too wide; p-values are too high). The reason for this lower power is that you are making fewer assumptions on the population. If you know more about the population and are able to use it, then that procedure will have higher power (all things being equal).

Figure R.5 shows the histogram of the sample means from the simulation. The triangle on the x-axis represents the mean of these sample means. The thick bar along the axis represents the estimated 95% confidence interval.

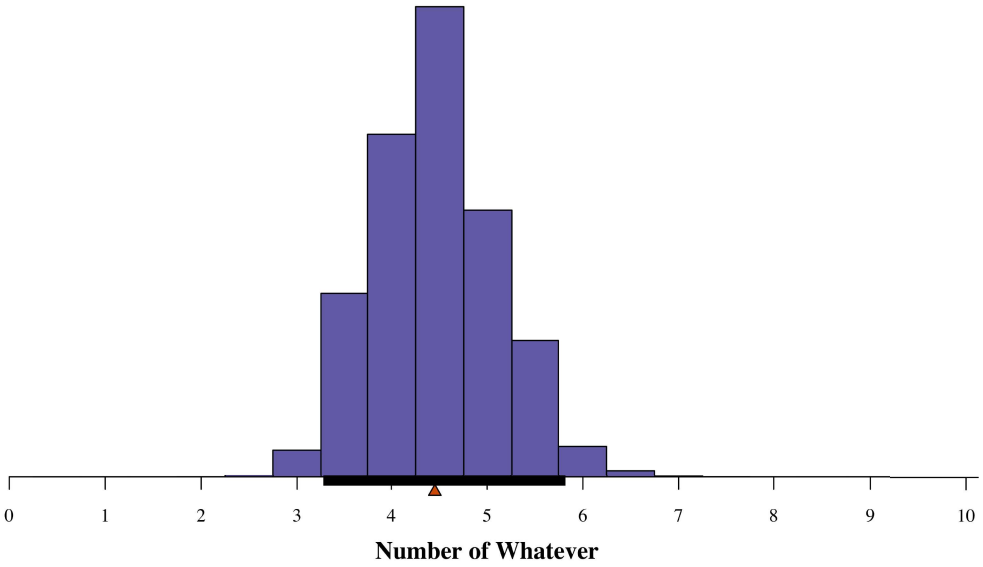

Figure R.5: *The estimated probability density function (pdf) of the sample means from the unknown population. The mean of the sample means is denoted by the triangle on the axis; the 95% confidence intervals, thick bar on the axis.*

Example 7

How good is the non-parametric bootstrap in terms of covering the population mean? For this investigation, let's assume the population really is standard Normal.

Solution: The first step is to draw a sample of size $n = 10$ from that population, then do the non-parametric bootstrap, then determine how frequently the population mean is in the confidence interval. It should happen 95% of the time.

Here is the code:

```
estLCL = numeric()
estUCL = numeric()

for(j in 1:1e6) {  # The loop testing the NPB

 theData = rnorm(10)
 newMeans = numeric()

 for(i in 1:1e6) {
  newData = sample(theData, replace=TRUE)
  newMeans[i] = mean(newData)
 }

 estLCL[j] = quantile(newMeans,0.025)
 estUCL[j] = quantile(newMeans,0.975)
}
```

Note that this will take quite some time to run. There are a total of 1,000,000,000,000 iterations taking place. On a new laptop, one should probably expect it to run overnight. On an older one, it may take an entire day. However, the results will be rather precise.

Let me take this opportunity to (re-) introduce you to a measure of the quality of a confidence interval: **coverage**. Coverage is defined as the proportion of the time that the confidence interval contains the true mean.

For our example, the true mean is 0. So, the coverage will be the proportion of the time that 0 is between the two estimated confidence bounds.

```
mean( 0>estLCL & 0<estUCL )
```

When I run this, I obtain 0.896, which is quite different from the hoped-for 0.95. It means I am rejecting at a rate of 10.4% instead of the claimed 5%. This is not good.
♦

Having too low of a coverage (as here) is problematic because it increases the risk that the true parameter value lies outside the confidence interval or prediction range. Coverage probability reflects the proportion of intervals that, over repeated sampling, are expected to contain the true value. If the coverage is too low, it means that the interval is overly narrow or poorly calibrated, leading to an underestimation of uncertainty. This can result in overconfidence in statistical conclusions, which may cause critical errors in decision-making or policy formulation, particularly in fields like medicine, finance, or engineering where risks are high.