# Statistical Audit Sampling with R

Koen Derks

*"The best things in life are free."*

# Contents

# Preface

This book, **Statistical Audit Sampling with R**, is intended as a practical guide for auditors who wish to employ probability theory in their audit sampling activities. While the focus of this book is exclusively on audit sampling, it aims to discuss the topic from both the classical (frequentist) perspective and the Bayesian perspective. By examining the subject through these two lenses, the book explains the statistical theory behind commonly used audit sampling procedures and demonstrates how to perform these procedures effectively and efficiently in accordance with international auditing standards using the **jfa** R package.

Audits can be time-consuming and require going through lots of paperwork. But when we find ways to make audits more efficient or understandable, we free up time for auditors to do other important things. They can focus on coming up with new ideas, foster innovation, and adressing the larger problems of our time. This not only helps them grow as individuals but, perhaps more important, sparks a chain reaction that will ripple through the audit. By clearing auditors' time, we pave the way for progress and a future where they can use their talents to make the world a better place.

This is a work-in-progress first edition of Statistical Auditing with R. The book is currently a dumping ground of ideas, and it is incomplete.

# Getting Started

This book is intended for auditors that want to obtain the knowledge and skill to utilize statistical audit sampling in their practice using the R programming language. It covers an array of traditional and innovative statistical tools that are available to auditors, explaining their function, the underlying assumptions, and when they are best utilized. In addition, it offers practical guidance on integrating advanced statistical sampling methodology into audit practice and demonstrates its value through real-world case studies. It is our hope that this book will serve as a valuable resource for auditors looking to effectively and efficiently utilize statistical methods in their practice.

The aim of this book is to address the need for a clear and transparent explanation of the use of statistical sampling methodology in audit practice. Most guidance about audit sampling (e.g., American Institute of Certified Public Accountants (AICPA) (2016a); American Institute of Certified Public Accountants (AICPA) (2016b)) lacks sufficient detail to allow for full transparency or a deep understanding. Additionally, the implementation of statistical sampling methodology in practice is often even less transparent, as theory and calculations are hidden from auditors in commercial closed-source tools or in Excel sheets from audit guides used internally by audit firms (one notable exception is Stewart (2012)). Thus, while attempting to comprehend the theoretical aspects of statistical audit sampling, auditors may encounter numerous relevant questions that are left unanswered by these tools. This book aims to clarify the statistical methodology utilized in practice, thereby empowering auditors through a comprehensive explanation.

This book discusses two philosphies to statistical audit sampling: the classical (frequentist) philosophy and the Bayesian philosophy. By contrasting these two approaches, the book elucidates the statistical theory that underlies commonly used audit sampling techniques and illustrates how to utilize these techniques in accordance with international auditing standards. Additionally, the book demonstrates the use of Bayesian statistical methods in auditing practice and highlights the practical advantages that these methods can offer for auditors.

The structure of this book is as follows: Chapter 2 introduces the basics of the R programming language. Chapter 3 discusses the fundamental statistical theory relevant to classical and Bayesian audit sampling. Subsequent chapters 4, 5, 6, 7 and 8 provide a more in-depth exploration of the use of these methods for statistical planning, selection, and evaluation of audit samples. These chapters illustrate the

practical differences and similarities between the classical and the Bayesian approach through code and examples, thus requiring minimal programming knowledge to follow along. Finally, Chapter 9 and 10 discuss other open-source software implementations of audit sampling using R.

In each chapter, we aim to adhere to a consistent structure: beginning with motivating examples to provide a broader context, and then delving into the specifics. Every section of the book is accompanied by exercises designed to reinforce the concepts learned. Although it may be tempting to skip the exercises, the most effective way to learn is by applying the concepts to actual problems through practice.

# Prerequisites

To run the code in this book, you will need three things: R, RStudio, and the **jfa** package.

## R

The **C**omprehensive **R** **A**rchive **N**etwork (CRAN) is a collection of mirror servers distributed globally that can be utilized to obtain both R and R packages. When seeking to download R, it is recommended to utilize the cloud mirror located at `https://cloud.r-project.org`, as it will automatically determine the most suitable mirror for your location. Please ensure that you have at least R `3.5.0` installed for the purposes of this book.

It is advisable to regularly update R, as new major versions are released annually and minor releases occur two to three times per year. While upgrading can be inconvenient, particularly for major versions which require the reinstallation of all packages, failing to do so will only exacerbate the issue.

## RStudio

RStudio is an Integrated Development Environment (IDE) specifically designed for the R programming language. It can be downloaded and installed from the official website (`https://posit.co/downloads/`). RStudio undergoes updates several times a year, and users will be notified when a new version becomes available. It is recommended to regularly upgrade RStudio to access the latest features. Please ensure that you have at least RStudio `1.0.0` installed for the purposes of this book.

When you start RStudio, you'll see three important regions in the interface: the console, the environment and the output panel.

The console in the left panel allows you to run R code. To execute some R code, type the code in the console (e.g., `1 + 1`) and press Enter. When you stored the results of a computation in a new variable (e.g., `x <- 1 + 1`), the variable is displayed in the environment in the top right panel. The environment allows you to keep track of the objects you have created during your R session. Finally, any plots that you request are displayed in the output panel in the bottom right.

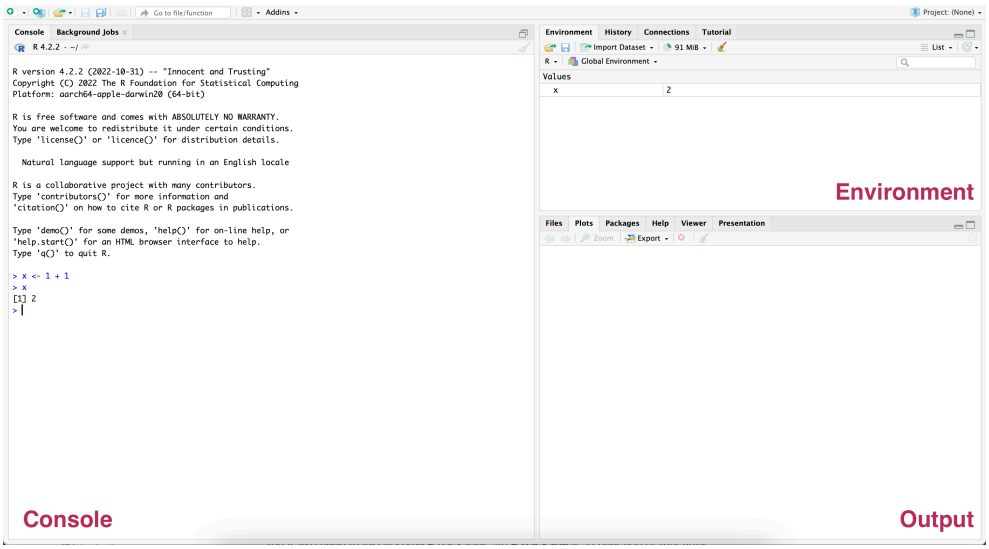



**Figure 1.** The RStudio IDE with the three important regions: the console, the environment, and the output panel. The console, located in the left panel, allows users to execute R code. The environment, displayed in the top right panel, keeps track of the variables and objects created in the session. The output panel, located in the bottom right, shows the generated plots.

## jfa

R has many packages and libraries that extend its capabilities and provide additional functions and tools for data analysis and visualization. R packages are the core building blocks of reproducible R code. They consist of reusable functions, the documentation describing how to utilize them, and sample data. Some popular packages include `dplyr` for data manipulation, `ggplot2` for data visualization, and `caret` for machine learning. If you want to learn more about creating your own R package, Wickam & Brian (2022) provide an excellent first-hand description of the principles and practices of creating R packages.

To illustrate its concepts and ideas, this book uses the **jfa** package, an R package for statistical auditing, which can also be downloaded from CRAN. This can be done via the `install.packages()` function by providing the package name in quotes. Thus, before running the examples in this book, you should install the **jfa** package by running the following command in R:

```
install.packages("jfa")
```

Once you have installed a package, you must load it into every R session. To load a package into your R session, call `library()` and provide the name of the package (without quotes) that you want to load. For example, before running the examples in this book, you should load the **jfa** package with:


(a) `dplyr`


(b) `ggplot2`


(c) `jfa`

**Figure 2.** R boasts an impressive amount of packages and libraries that enhance its capabilities by providing additional features. For example, the (a) `dplyr` package assists with data manipulation, the (b) `ggplot2` package supports data visualization, and the (c) `jfa` package facilitates statistical auditing.

```
library(jfa)
```

R packages are updated regularly. To update a package in your R library you should call `update.packages()` and provide the name of the package that you want to update in quotes. For example, each time there is a new release of the **jfa** package, you can update it by running:

```
update.packages("jfa")
```

If you want to look at the source code of the **jfa** package, see the package website at `https://koenderks.github.io/jfa`.

## Running R Code

The code in this book appears as follows:

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

When executing the same code within your local console, the resulting output will appear as follows:

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

There are two primary distinctions between the use of a console and the presentation of code in the book. Firstly, in a console, the user inputs their code after the prompt symbol (`>`) while the book does not display this symbol. Secondly, the output in the book is commented out with `#>`, while in a console it is displayed directly after the inputted code. Hence, those working with an electronic version of the book can directly copy code from the book to the console.

## Colophon

An online version of this book is available at `https://koenderks.github.io/sasr/`.
The source code of the book is available at `https://github.com/koenderks/sasr`.

# Part I

# Theory

# Chapter 1

# Audit Sampling

Auditors use audit sampling as a technique to assess a selection of transactions or items within a population in order to form conclusions about the population as a whole. It is a cost-efficient method for examining the accuracy of financial information as it allows auditors to test a representative sample of the population rather than the entire population.

In the field of auditing, sampling becomes necessary when the truth about a population is not readily accessible or discernible through other means. With the advent of modern technology, auditors often have access to an abundance of information about a population, which sometimes enables them to perform full population (i.e., integral) testing. Nonetheless, there are situations where a sample is still necessary due to the unavailability of complete data. For instance, an auditor may utilize analytical procedures to verify the consistency of payments with payment orders, but then must subsequently confirm the validity of these orders through detailed testing.

In auditing, there are two primary methods of sampling: statistical and nonstatistical. Statistical sampling involves using probability theory to select a sample from the population and draw conclusions about the population based on the sample. Nonstatistical sampling, on the other hand, is based on the auditor's professional judgment and does not use statistical inference to come to a conclusion. This book does not cover nonstatistical sampling.

International auditing standards prescribe the manner in which statistical sampling should be conducted in an audit. The following section discusses what these standards require from the auditor's approach to statistical sampling.

## 1.1 Auditing Standards

There are three auditing standards related to staistical sampling in the audit:

- ISA 530: Auditing standard for international firms published by the International Auditing and Assurance Standards Board (IAASB).

- AU-C 530: Auditing standard for private firms published by the American Institute of Certified Public Accountants (AICPA).
- AS 2315: Auditing standard for public firms published by the Public Company Accounting Oversign Board (PCAOB).

All three standards present a similar explanation of statistical sampling. For instance, ISA 530 (International Auditing and Assurance Standards Board (IAASB), 2018) defines statistical audit sampling as a method that at minimum exhibits the following two characteristics:

- Random selection of sample items,
- The use of an appropriate statistical technique to evaluate sample results, including measurement of sampling risk.

According to auditing standards, any sampling approach that lacks these two characteristics is considered nonstatistical sampling.

In order to effectively utilize statistical sampling during an audit, it is necessary to tailor the approach to the specific circumstances of the audit. This may involve considering factors such as the size and complexity of the population, the materiality of the items being tested, and the level of inherent risk in the audit area. It is also essential for the auditor to document the sampling process in order to demonstrate compliance with auditing standards. The following sections will delve further into these crucial concepts in the context of statistical audit sampling.

## 1.2 Important Concepts

This section aims to delve into several theoretical concepts that are integral to statistical audit sampling.

### 1.2.1 Materiality

In an audit, materiality is the maximum amount of misstatement that can be present in the financial statements of the auditee before the auditor concludes that the financial statements are materially misstated, meaning that they contains misstatements that would influence the decisions of stakeholders relying on those statements.

The term performance materiality refers to the maximum amount of misstatement that can be present in a given population that is part of the financial statements before the auditor concludes that the population is materially misstated. Performance materiality is used by auditors to determine the appropriate level of testing to be performed on a population. The performance materiality is usually defined to be lower than the materiality because an individual population that is subject to audit sampling is often only a (small) part of the financial statements.

For example, consider an audit of a company's financial statements for the year ended December 31, 2021. The auditor determines that the company's accounts receivable balance is a large part of to the financial statements and decides to test a sample of the accounts receivable transactions to assess the accuracy of the balance. The auditor calculates the performance materiality for the accounts receivable balance by

considering the materiality for the financial statements as a whole. If the auditor finds misstatements in the sample such that their estimate of the misstatement exceeds the performance materiality, the auditor would need to express an unqualified opinion on the population or would need to perform additional testing on the population. If the auditor finds misstatements in the sample such that their estimate of the misstatement does not exceed the performance materiality, the auditor would express a positive opinion on the financial statements.

### 1.2.2 Audit risk

After completing an audit and making any necessary corrections, an auditor will issue a written report stating whether the financial statements are accurate and free of material misstatement. The potential for this opinion to be incorrect is known as audit risk, and it is the auditor's job to minimize this risk as much as possible.

For example, during an audit of a company's financial statements, the auditor may carefully review documentation, perform tests of details via audit sampling, and speak with management in order to reduce the audit risk and provide a reliable opinion on the accuracy of the financial statements as a whole.

### 1.2.3 Population

In statistical inference, the term population refers to the entire group of individuals or items that have some common characteristic or interest, and about which we want to gather data or make inferences. A population can be as large as all the people in a country or, as is more sensible in auditing, as small as a group of employees in a specific department of a company.

For example, consider an audit of a company's payroll records. The population in this case would be all the employees of the company, and the goal of the audit would be to gather data on their salaries, benefits, and other payroll-related information. The audit team would collect this data from a a representative group of employees (i.e., a sample) of the population and use statistical methods to draw conclusions about the entire population.

### 1.2.4 Sampling risk

There is a possibility that the results of an audit based on a sample may differ from the results if the entire population were examined using the same procedures. This is known as sampling risk. Sampling risk can result in two types of incorrect conclusions:

1. The first type is when, in a test of controls, the controls are perceived to be more effective than they actually are, or in a test of details, a material misstatement is believed to not exist when it actually does. This type of erroneous conclusion is particularly concerning for auditors because it can compromise the effectiveness of the audit and may lead to an inappropriate audit opinion.
2. The second type of incorrect conclusion is when, in a test of controls, the controls are perceived to be less effective than they actually are, or in a test of details, a material misstatement is believed to exist when it actually does not. This type

of erroneous conclusion impacts the efficiency of the audit as it may require additional work to determine that the initial conclusions were incorrect.

Many audits are performed according to the audit risk model (ARM), which determines that the uncertainty about the auditor's statement as a whole is a factor of three terms: the inherent risk, the control risk, and the detection risk (i.e., the sampling risk). Inherent risk is the risk posed by a misstatement in the auditees financial statements that could be material, before consideration of any related control systems (e.g., computer systems). Control risk is the risk that a material misstatement is not prevented or detected by the auditee's internal control systems. Detection risk is the risk that the auditor will fail to find material misstatements that exist in the auditee's financial statements. The ARM is practically useful because, for a given level of audit risk, the tolerable detection risk bears an inverse relation to the other two risks.

$$\text{Audit risk} = \text{Inherent risk} \times \text{Control risk} \times \text{Detection risk} \tag{1.1}$$

Usually the auditor judges inherent risk and control risk on a three-point scale consisting of low, medium, and high. Different audit firms handle different standard percentages for these categories. Given an assessment of the inherent risk and the control risk, the detection risk can be calculated as:

$$\text{Detection risk} = \frac{\text{Audit risk}}{\text{Inherent risk} \times \text{Control risk}} \tag{1.2}$$

Let's consider an example. Suppose that, in their audit guide, an audit firm associates the following percentages with the categories high, medium and low:

- High: 100 percent
- Medium: 60 percent
- Low: 50 percent

If an auditor is working with an audit risk of 0.05, or five percent, and judges inherent and control risk to both be medium, the sampling risk can be calculated as:

$$\frac{0.05}{0.6 \times 0.6} = 0.139 \tag{1.3}$$

Note that the ARM is commonly used in practice but it is not a proper model of audit risk. For example, it is not possible to set one of the risks to zero, as that would result in an infinite detection risk (e.g., $\frac{0.05}{0 \times 1} = \infty$).

### 1.2.5 Sample Size

The sample size is an important consideration in the context of audit sampling, as it determines the number of items that will be selected for testing during the audit process. This factor has an impact on both effectiveness and efficiency. In general, a larger sample size can provide a higher level of assurance, but it requires more audit effort to obtain and inspect. On the other hand, a smaller sample size offers a lower level of assurance, but it is less costly.

### 1.2.6 Notation

The table below summarizes the notation used in this book (middle column) and in the **jfa** R package (right column).

| Meaning | Symbol | **jfa** |
|---|---|---|
| Probability of misstatement | $\theta$ | |
| Performance materiality | $\theta_{max}$ | `materiality` |
| Expected deviation rate | $\theta_{exp}$ | `expected` |
| Type-I error probability | $\alpha$ | `1 - conf.level` |
| Type-II error probability | $\beta$ | |
| Population size | $N$ | `N.units` |
| Population misstatements | $K$ | |
| Sample size | $n$ | `n` |
| Observed misstatements | $k$ | `x` |

# Chapter 2

# Statistical Inference

In the field of probability theory, two prominent schools of thought have emerged: frequentism and Bayesianism. These two approaches offer distinct perspectives on how to interpret and reason about uncertainty and probability.

## 2.1 Classical Inference

Frequentist statistics, also known as classical statistics, is a statistical framework that is based on the concept of probability as a long-term frequency of events. This approach assumes that parameters in a statistical models have a true but hidden value and that data is generated by a well-defined process, which can be described by a set of probabilistic assumptions about the model parameters. The philosophy behind frequentist statistics is that statistical estimates should be based on the frequency of events over a long term, rather than on subjective or personal information. This statistical approach is particularly useful for making predictions or decisions based on data, as it allows for the calculation of confidence intervals and statistical tests, which provide a measure of the reliability of the estimates. Overall, frequentist statistics is a rigorous and reliable approach that is widely used in the scientific community for making decisions based on data. However, as we will discuss, it also has some drawbacks that make it less suitable for use in audit practice.

### 2.1.1 Estimation

The philosophy behind frequentist parameter estimation is based on the idea that statistical parameters are fixed, but unknown, quantities that can be estimated through the process of repeated sampling. This approach assumes that the sample data represent a random sample from the population, and that the sample statistics (i.e., the sample proportion of misstatements) can be used to estimate the corresponding population parameters (i.e., the population misstatement). The key principle of frequentist estimation is that the estimated parameter values should be unbiased and have a certain level of uncertainty, which can be quantified through confidence bounds or intervals.

#### 2.1.1.1 Example

As an example, the `binom.test()` function in R can be used to estimate the rate of misstatement in a population given a data sample of $n$ items containing $k$ misstatements. Suppose an auditor audited a sample of $n = 60$ items containing $k = 0$ misstatements. To use the `binom.test()` function to perform estimation, the auditor must input the number of items in the sample `n = 60` and the number of misstatements in the sample `x = 0`. The sampling risk is set to 0.05, or five percent, which the auditor can provide to the function with `conf.level = 1 - 0.05`. Finally, the auditor can specify the alternative hypothesis as `alternative = "less"` to compute a one-sided confidence interval since they are interested in obtaining the upper confidence bound.

```
binom.test(x = 0, n = 60, alternative = "less", conf.level = 1 - 0.05)
#>
#>  Exact binomial test
#>
#> data:  0 and 60
#> number of successes = 0, number of trials = 60, p-value < 2.2e-16
#> alternative hypothesis: true probability of success is less than
 ↪  0.5
#> 95 percent confidence interval:
#>  0.00000000 0.04870291
#> sample estimates:
#> probability of success
#>                      0
```

The most likely misstatement in the population is displayed under `sample estimates` and is 0, or zero percent. The 95 percent upper confidence bound for the estimate of the population misstatement is displayed under `95 percent confidence interval` and is 4.87 percent.

### 2.1.2 Hypothesis Testing

Frequentist hypothesis testing is a statistical method that involves evaluating the probability of obtaining a certain sample outcome or more extreme, given a certain assumption or hypothesis. This probability, known as the $p$-value, is used to determine the likelihood of the hypothesis being true.

For example, in a typical audit sampling hypothesis test using the binomial distribution, we may be interested in testing the hypothesis that the misstatement is higher or lower than the performance materiality. We would inspect a sample and calculate the $p$-value based on the observed frequency of misstatements versus the expected frequency under the assumption of material misstatement. If the $p$-value is below the sampling risk $\alpha$, we reject the hypothesis that the population is materially misstated and conclude that it is not materially misstated.

**Figure 2.1.** Innovating statistical methods and shaping the field of genetics, Sir Ronald Fisher was a pioneer in the world of science. Image available under a CC-BY-NC 4.0 license.

#### 2.1.2.1 Example

Next to estimation, the `binom.test()` function in R can also be used to test if a population contains is free of material misstatement, which in this case means that the population contains less than three percent misstatements. Suppose an auditor obtained a sample of $n = 100$ items containing $k = 0$ misstatements. To use the `binom.test()` function, the auditor must input the number of items in the sample `n = 100`, the number of misstatements in the sample `x = 0`, and the hypothesized proportion of misstatement in the population (i.e., the performance materiality) `p = 0.03`. The sampling risk is set to 0.05, or five percent, which the auditor can provide to the function with `conf.level = 1 - 0.05`. This is the default value and therefore this argument is omitted in the code below. Finally, the auditor can specify the alternative hypothesis as `alternative = "less"` to test if the proportion of misstatements in the sample is less than the hypothesized proportion.

```
binom.test(x = 0, n = 100, p = 0.03, alternative = "less")
#>
#>  Exact binomial test
#>
#> data:  0 and 100
#> number of successes = 0, number of trials = 100, p-value = 0.04755
#> alternative hypothesis: true probability of success is less than
 ↪  0.03
#> 95 percent confidence interval:
#>  0.00000000 0.02951305
#> sample estimates:
#> probability of success
#>                      0
```

The $p$-value is shown under `p-value` and is 0.04755. Since the $p$-value is lower than the

specified sampling risk $\alpha$, the auditor can reject the hypothesis that the population contains material misstatement and should conclude that the population does not contain material misstatement.

## 2.2 Bayesian Inference

Bayesian inference (Jeffreys, 1939, 1948, 1961) is based on the idea that the parameters in a statistical model are not fixed but uncertain. In this approach, the misstatement in the population is considered to be a random variable with a certain distribution, and the goal is to use the data and any prior knowledge about the parameter to update our belief about its value. This is typically done using Bayes' theorem, which states that the posterior probability (i.e., the updated belief about the parameter after seeing the data) is equal to the prior probability (i.e., the belief about the parameter before seeing the data) times the likelihood (i.e., the probability of the data given the parameter).

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \tag{2.1}$$

Bayesian statistics is a more nuanced approach that allows for more efficiency in statistical audit sampling (Steele, 1992), but it requires the specification of prior distributions that can be difficult to quantify. That is because, especially in an audit, all information that is incorporated into the statistical analysis should be based on audit evidence and should be properly justified.

### 2.2.1 Estimation

One major difference between classical and Bayesian statistics is the way they handle uncertainty. In classical statistics, uncertainty is represented by the standard error of an estimate, which is a measure of the precision of an estimate. In Bayesian statistics, uncertainty is represented by the posterior distribution, which is a distribution of the possible values of the population parameter given the sample data and our prior beliefs. Bayesian inferences uses uses Bayes' theorem to update the prior beliefs about the population parameter with the new information from the sample data. Bayes' theorem is given by the following formula:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{2.2}$$

where $p(\theta|y)$ is the posterior probability of the population parameter $\theta$ given the sample data $y$, $p(y|\theta)$ is the likelihood of the sample data given $\theta$, $p(\theta)$ is the prior probability of $\theta$, and $p(y)$ is the total probability of the sample data occurring. Because with a fixed sample $p(y)$ is a constant, Bayes' theorem is often given as follows:

$$p(\theta|y) \propto p(y|\theta) \times p(\theta) \tag{2.3}$$

**Figure 2.2.** Thomas Bayes revolutionized the world of statistics and probability with his groundbreaking work on Bayes' Theorem. His contributions continue to shape the way we understand and analyze data today. Image available under a CC-BY-NC 4.0 license.

#### 2.2.1.1 Example

Bayesian inference involves first specifying a prior distribution that captures the available information about the probability of misstatement in the population. For illustrative purposes, we use a simple prior distribution that is indifferent about the possible values of the misstatement. Note that this prior distribution does not contain any information about the probability of misstatement, but yields statistical results that closely resemble classical outcomes. This prior distribution is shown in Figure 2.3. Chapters 3, 4, 5 and 6 dive deeper into the R functions from the **jfa** package used in this example, such as the `auditPrior()` function below.

```
prior <- auditPrior(method = "default", likelihood = "binomial")
plot(prior)
```

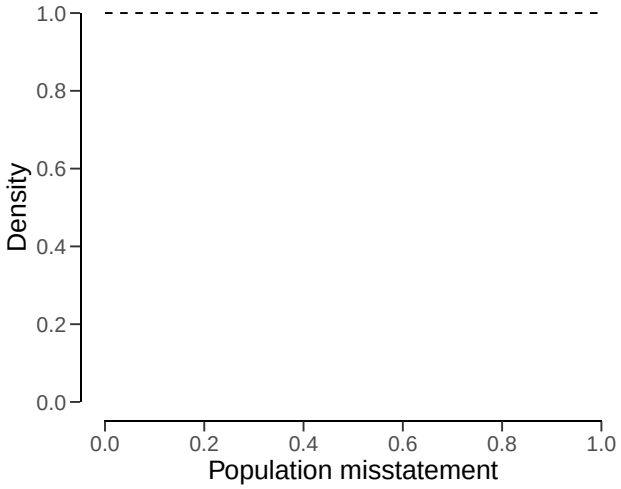



**Figure 2.3.** A uniform prior distribution assigning equal probability to every possible value of the population misstatement $\theta$.

After seeing an item from the population, the prior distribution is updated to the posterior distribution by means of Bayes' theorem. Next, the posterior is used as a prior distribution for the following item, which is once again updated to a posterior distribution after seeing another the item. This process of updating the prior distribution to a posterior distribution, and using the posterior as a prior can continue indefinately. For example, after seeing 30 items from the population, of which none contained a misstatement, the posterior distribution peaks at zero, reflecting the fact that no misstatements were found, and its mass has shifted towards zero when compared to the prior distribution, reflecting the fact that the data contained information that indicates a low probability of misstatement. Figure 2.4 displays the prior and posterior distribution.

```
eval <- evaluation(x = 0, n = 30, method = "binomial", prior = prior)
plot(eval, type = "posterior")
```

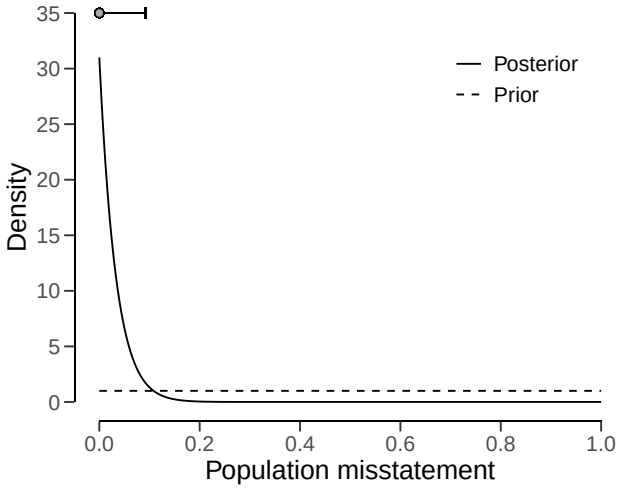



**Figure 2.4.** The prior and posterior distribution after seeing a sample of $n = 30$ items, of which none contained a misstatement. After seeing these data, lower values of the population misstatement now receive a relatively high probability.

The most likely value of the population misstatement based on these data is shown as a point above the highest point of the posterior distribution. The uncertainty about the population misstatement can be quantified using a credible interval (shown as a line above the posterior distribution). In this case, the figure above shows a 95 percent credible interval, which contains the true value of the population misstatement with a 95 percent probability.

After seeing 30 more observations, of which none contained a misstatement, the mass of the posterior distribution has shifted towards zero even more. The credible interval shown above the posterior distribution encompasses a smaller range of values, reflecting the fact that additional information has been observed and thus that there is less uncertainty about the population misstatement than before. This is displayed in Figure 2.5.

```
eval <- evaluation(x = 0, n = 30, prior = eval$posterior)
plot(eval, type = "posterior")
```

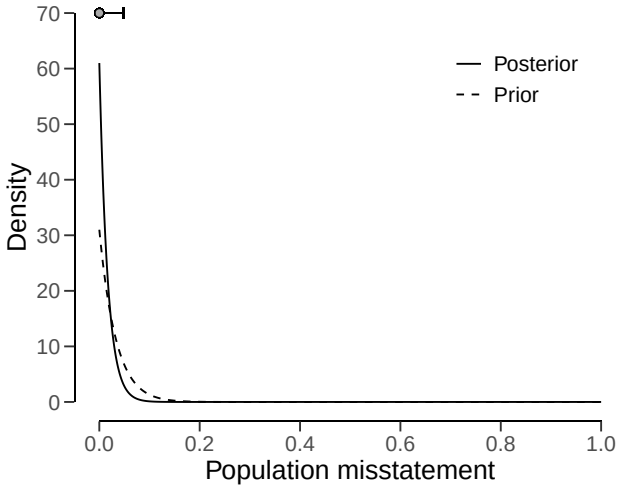



**Figure 2.5.** The prior and posterior distribution after seeing a second sample of $n = 30$ items, of which none contained a misstatement. The posterior distribution is relatively less wide than in the previous example.

The updating process works the same for the scenario in which the auditor finds misstatements in the sample. For example, the posterior distribution after finding a single misstatement is has its mass shifted away from zero, reflecting the fact that a probability of zero is unlikely given the sample data, see Figure 2.6.

```
eval <- evaluation(x = 1, n = 1, prior = eval$posterior)
plot(eval, type = "posterior")
```

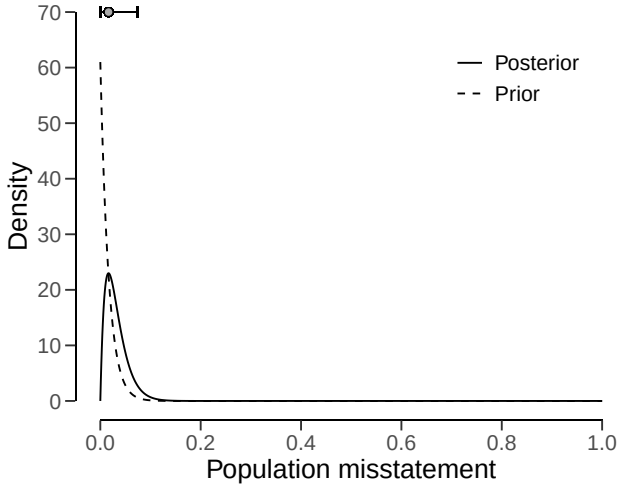



**Figure 2.6.** The prior and posterior distribution after seeing a third sample of a single item which contained a misstatement. Given this item on top of the other items in the sample, the posterior distribution does not peak at zero anymore.

### 2.2.2 Hypothesis Testing

The Bayes factor is a measure used in Bayesian inference to compare the relative strength of evidence between two competing hypotheses. The Bayes factor is calculated by comparing the probability of the observed data given each of the two competing hypotheses. This probability is known as the likelihood of the data. The Bayes factor is then the ratio of the likelihood of the data under one hypothesis to the likelihood of the data under the other hypothesis. The Bayes factor can be used in the context of an audit, where the auditor is trying to determine the likelihood that a particular financial statement is represented fairly or not.

For example, an auditor might be evaluating the fairness of a company's financial statements for the year. They have two hypotheses: the first is that the statements are accurate, and the second is that the statements are not accurate. The auditor gathers data from a statistical audit sample and uses this data to calculate the Bayes factor.

The Bayes factor is calculated by taking the ratio of the probability of the first hypothesis (that the statements are accurate) given the observed data, to the probability of the second hypothesis (that the statements are not accurate) given the observed data. The higher the Bayes factor, the more likely it is that the first hypothesis is true.

The Bayes factor can be used to assess the strength of evidence for one hypothesis over the other and to determine which hypothesis is more likely to be true given the observed data. It is often used in scientific research to help evaluate the validity of different hypotheses and to make informed decisions based on the available evidence. For auditors, the Bayes factor can be a useful tool to determine the likelihood of different hypotheses being true based on the data they have collected, and it can help

them make informed decisions about the fairness of the financial statements.

For example, if the Bayes factor is 5, this means that the probability of the statements being accurate given the observed data is 5 times higher than the probability of them being not accurate. In this case, the auditor would be more likely to conclude that the financial statements are accurate.

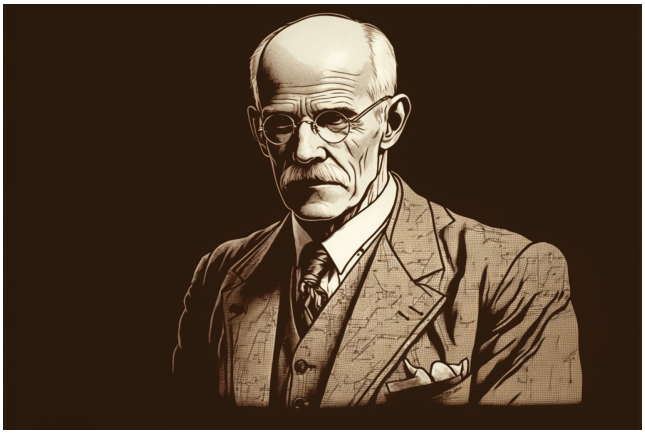


**Figure 2.7.** Sir Harold Jeffreys innovated statistical hypothesis testing with his Bayes factor approach, helping us make better decision in the face of uncertainty. Image available under a CC-BY-NC 4.0 license.

#### 2.2.2.1 Example

Suppose an auditor obtained a sample of $n = 100$ items containing $k = 0$ misstatements. Given the number of items in the sample `n = 100`, the number of misstatements in the sample `x = 0`, and the hypothesized proportion of misstatement in the population (i.e., the performance materiality) `p = 0.03` the Bayes factor is displayed under `BF10` and is 668.65, meaning that it the data are about 668 times more likely to occur under the hypothesis of tolerable misstatement than under the hypothesis of intolerable misstatement.

```
evaluation(materiality = 0.03, x = 0, n = 100, method = "binomial",
 ↪  prior = TRUE)
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  0 and 100
#> number of errors = 0, number of samples = 100, taint = 0, BF  =
#> 668.65
#> alternative hypothesis: true misstatement rate is less than 0.03
#> 95 percent credible interval:
#>  0.00000000 0.02922515
#> most likely estimate:
#>  0
```

```
#> results obtained via method 'binomial' + 'prior'
```

# Part II

# Application

# Chapter 3

# Sampling Workflow

The audit sampling workflow involves a series of steps that help auditors select a representative sample from a population of transactions or items, and then use statistical analysis to draw conclusions about the entire population. It consists of four stages: planning, selection, execution and evaluation. Probability theory can be employed in three of these stages.

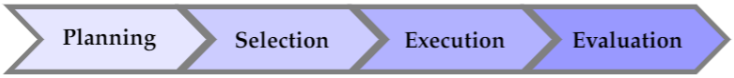



**Figure 3.1.** The standard audit sampling workflow consists of four stages: planning, selection, execution and evaluation.

In the planning stage, auditors lay the foundation for an effective and efficient audit sample. Careful planning allows auditors to determine the appropriate sample size, considering factors such as the desired level of assurance, the risk of material misstatement, and the available resources. This stage involves understanding the objectives of the audit, identifying relevant population characteristics, and considering any constraints or limitations. Probability theory can be applied in this stage to help auditors quantify the level of confidence they aim to achieve and make informed decisions about the sample size and selection methodology.

The selection stage focuses on the actual sampling process. Auditors employ various techniques to choose a representative sample that accurately reflects the characteristics of the population under investigation. This stage often involves random sampling methods, such as simple random sampling or stratified sampling, to ensure that each item in the population has an equal or known chance of being included in the sample. Probability theory plays a fundamental role in sample selection, as it provides the theoretical framework to ensure unbiased and statistically valid sampling methods.

Once the sample has been selected, the execution stage comes into play. During this stage, auditors carry out tests of details and inspections on the items in the sample. They examine financial information, scrutinize supporting documents, and perform

other procedures to assess the accuracy, completeness, and fairness of the selected items. The execution stage relies on auditing techniques and procedures specific to the nature of the audit engagement. While statistical inference is not typically employed during this stage, the execution phase provides the groundwork for the subsequent evaluation stage.

In this chapter of the book, we will explore the intricacies of the standard audit sampling workflow and demonstrate how it can be implemented using the powerful tools and techniques available in R. By understanding the various stages of the workflow and leveraging the capabilities of statistical analysis in R, auditors can enhance the efficiency and effectiveness of their audit procedures. Throughout this chapter, we will delve into the practical aspects of each stage, providing examples and step-by-step guidance to equip auditors with the necessary knowledge and skills to navigate the audit sampling process successfully.

## 3.1 Stage 1: Planning

The first stage in the audit sampling workflow is the planning stage. Proper planning of a sample plays a crucial role in enhancing audit efficiency. When auditors select a small sample, the audit effort required is relatively low. However, this approach may not provide a high level of assurance since the sample may not accurately represent the entire population of transactions or items under audit. On the other hand, auditing a larger sample increases the level of assurance but demands more audit effort and resources.


**Figure 3.2.** The planning stage is the first stage in the audit sampling workflow.

To strike a balance between efficiency and assurance, it is beneficial to determine the sample size in advance. By carefully considering various factors, auditors can optimize the sample size to meet the objectives of the audit engagement. Factors such as the desired level of confidence, acceptable precision, materiality thresholds, and the nature of the population being examined all influence the determination of the optimal sample size.

In Chapter 4, we will delve into the intricacies of planning an audit sample and explore the factors that should be taken into account. We will discuss the importance of understanding the characteristics of the population, identifying relevant risk factors, and applying statistical techniques to ensure an appropriate sample size. Furthermore, we will demonstrate how R can be leveraged to facilitate the sample planning process. R provides a wide range of tools and functions that aid in sample size calculations via the **jfa** package, allowing auditors to make informed decisions based on quantitative analysis.

By comprehensively addressing the considerations involved in sample planning and utilizing the capabilities of R, auditors can optimize the efficiency and effectiveness of their sampling procedures. This, in turn, contributes to the reliability and quality of

the audit results, enabling auditors to provide valuable insights and recommendations to stakeholders.

## 3.2 Stage 2: Selection

The second stage in the audit sampling workflow is the planning stage.

Selecting a sample is the second step in the audit sampling workflow and plays a crucial role in obtaining a representative subset of the population for examination. In Chapter 5, we will delve into the intricacies of selecting an audit sample and explore a range of approaches and techniques that can be employed using R.

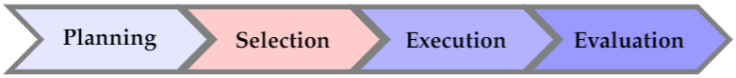



**Figure 3.3.** The selection stage is the second stage in the audit sampling workflow.

The selection of an audit sample requires careful consideration of factors such as the sampling method, sample size, and sampling frame. We will discuss popular sampling methods, including simple random sampling, fixed interval sampling, and cell sampling. Each method has its own advantages and considerations, and we will provide insights on when and how to effectively use them based on the characteristics of the population under audit.

Furthermore, we will explore the utilization of R to implement the sample selection techniques discussed. R provides a wide range of functions and packages specifically designed for sampling and randomization via the **jfa** package, making it an ideal tool for auditors seeking to streamline their sampling processes and enhance efficiency.

To ensure a comprehensive understanding of the material, we will present practical examples and demonstrate how these selection techniques can be applied in real-world audit scenarios. By examining these examples, you will gain practical insights into the application of different sampling methods, learn to navigate potential challenges, and develop a deeper understanding of the impact of sampling choices on the reliability and effectiveness of audit procedures.

By the end of Chapter 5, you will have a solid grasp of the various approaches and techniques available for selecting an audit sample, equipped with the knowledge and practical skills to implement them using R. This knowledge will enable you to make informed decisions regarding sample selection and enhance the overall quality and efficiency of your audit engagements.

## 3.3 Stage 3: Execution

The execution stage of the audit sampling process is a hands-on phase where the auditor meticulously examines a carefully chosen selection of items from the population under scrutiny. In this stage, the auditor focuses on assessing the accuracy, completeness, and fairness of the items in question. Through a combination of analytical

procedures, substantive testing, and detailed scrutiny of relevant documentation, the auditor seeks to gather concrete evidence and form objective judgments about the quality and integrity of the sample items.

Unlike the previous stages that heavily rely on statistical inference, the execution stage predominantly involves manual labor and detailed examination of the selected items. The auditor applies their expertise, professional judgment, and industry-specific knowledge to delve into the intricacies of each item, scrutinizing transactions, verifying supporting documentation, and assessing compliance with relevant regulations and accounting principles.

During this stage, auditors meticulously analyze financial statements, delve into underlying records, evaluate the authenticity of documents, and engage in interviews with key personnel. They perform detailed tests and inspections to identify potential discrepancies, irregularities, or areas of concern that may warrant further investigation. This rigorous examination enables the auditor to gain a comprehensive understanding of the accuracy, validity, and completeness of the sample items and helps them uncover potential red flags or areas where misstatements or irregularities may occur.

While the execution stage does not directly involve statistical inference, it plays a crucial role in gathering firsthand evidence, obtaining a deeper understanding of the audited items, and establishing a solid foundation for subsequent evaluation. The information and insights gathered during this stage form the basis for drawing meaningful conclusions about the entire population and provide a vital context for the subsequent statistical analysis performed in the evaluation stage.

In summary, the execution stage represents a meticulous and labor-intensive phase of the audit sampling process. Through detailed examinations and analyses, auditors ensure the accuracy, fairness, and compliance of the selected items. By conducting thorough manual inspections, auditors gather valuable evidence that serves as a cornerstone for the subsequent stages of the audit, ultimately contributing to the overall reliability and integrity of the audit findings.

## 3.4 Stage 4: Evaluation

The final stage in the audit sampling workflow is the evaluation stage, which plays a crucial role in drawing meaningful conclusions and making informed decisions based on the results obtained from the selected sample. During this stage, auditors employ various statistical tests, metrics, and estimation techniques to assess the findings and extrapolate them to the entire population.


**Figure 3.4.** The evaluation stage is the final stage in the audit sampling workflow.

In the evaluation stage, auditors aim to estimate the extent of misstatement or deviation from expected values within the population. They analyze the sample data and

apply statistical procedures to quantify the level of confidence or uncertainty associated with the findings. This enables auditors to provide a reliable assessment of the population's characteristics, identify potential risks, and evaluate the effectiveness of internal controls.

Statistical techniques commonly used in the evaluation stage include point estimation, interval estimation, and hypothesis testing. Point estimation involves estimating population parameters, such as the mean or proportion, based on the sample statistics. Interval estimation provides a range of plausible values for the population parameter, accompanied by a confidence level. Hypothesis testing allows auditors to test specific claims or hypotheses about the population based on the sample data.

In Chapter 6, we will examine the methods and techniques utilized to evaluate an audit sample from both a classical and Bayesian point of view. We will delve into the diverse range of statistical tests and metrics that can be applied to estimate the population misstatement and quantify the strength of evidence for or against a hypothesis based on the sample data.

Through detailed explanations and practical examples, you will gain a comprehensive understanding of how to effectively and efficiently evaluate an audit sample using R. You will learn how to interpret the results obtained from statistical tests and metrics, enabling you to make informed decisions in the context of the audit engagement. This chapter will equip you with valuable skills to assess the reliability of the audited information and provide meaningful insights to stakeholders.

Furthermore, Chapter 7 expands upon these evaluation methods and explores their generalization to stratified samples. Stratification allows auditors to partition the population into subgroups or strata based on specific characteristics. By evaluating samples within each stratum, auditors can obtain more detailed and targeted information about different segments of the population. We will discuss the application of evaluation techniques to stratified samples, enabling you to enhance the precision and accuracy of your audit conclusions.

By the end of Chapter 7, you will have a comprehensive toolkit to evaluate audit samples, whether they are obtained from simple random sampling or more complex stratified sampling designs. This knowledge will empower you to conduct thorough and rigorous audits, ensuring the reliability and integrity of the financial information under examination.

# Chapter 4

# Planning

One of the key considerations in audit sampling is determining the appropriate sample size to reduce the sampling risk to an appropriately low level, while minimizing audit effort. To quantify the sampling risk, it is necessary to specify the statistical model that connects the data to the population misstatement, referred to as the likelihood. This chapter will delve into three standard likelihoods commonly employed in audit sampling: the hypergeometric likelihood, the binomial likelihood, and the Poisson likelihood.

Note that planning is a game of optimization that could be avoided. If you are using the Bayesian approach to audit sampling, it is not required to plan a specific sample size in advance (Derks et al., 2022b). That is because in Bayesian inference, the posterior distribution after seeing each item is used as the prior distribution for the next item. That means that you can simply start sampling and monitor the evidence in the data over time. However, to get an idea of how many samples are required to reduce the sampling risk to an level, planning a sample using a Bayesian approach can still be good practice.

## 4.1 Required Information

First, planning a minimum sample requires knowledge of the conditions that lead to acceptance or rejection of the population (i.e., the sampling objectives). Typically, sampling objectives can be classified into one or both of the following:

- **Hypothesis testing**: The goal of the sample is to obtain evidence for or against the claim that the misstatement in the population is lower than a given value (i.e., the performance materiality).
- **Estimation**: The goal of the sample is to obtain an accurate estimate of the misstatement in the population with a certain precision.

Second, it is advised to specify the expected (or tolerable) misstatements in the sample. The expected misstatements are the misstatements that you allow in the sample, while still retaining the desired amount of assurance about the population. It is

strongly recommended to set the value for the expected misstatements in the sample conservatively to minimize the chance of the observed misstatements in the sample exceeding the expected misstatements, which would imply that insufficient work has been done in the end to reduce the sampling risk to an appropriately low level.

Finally, next to determining the sampling objective(s) and the expected misstatements, it is important to determine the statistical distribution linking the sample outcomes to the population misstatement. This distribution is called the likelihood (i.e., `poisson`, `binomial`, or `hypergeometric`). All three aforementioned likelihoods are commonly used in an audit sampling context, however, `poisson` is the default likelihood in **jfa** because it is the most conservative of the three. In the subsections below, we elaborate on the three standard likelihoods for audit sampling and demonstrate how they can be used to obtain a minimum sample size.

In **jfa**, determining an appropriate sample size is achieved via the `planning()` function.

**Figure 4.1.** When the sampling objectives, the expected misstatements and the likelihood are set, there exists only one sample size that minimizes the amount of audit effort. Image available under a CC-BY-NC 4.0 license.

## 4.2 The Hypergeometric Likelihood

The hypergeometric distribution is a discrete probability distribution that is commonly used to model the number of events occurring in a fixed number of trials when the population size is known. It assumes that samples are drawn from the population without replacement, and is therefore the likelihood that most closely resembles the audit practice. For our purpose, we can use the hypergeometric distribution as a likelihood to model the number of misstatements that are expected to be found in the sample.

The probability mass function (PMF) of the hypergeometric distribution is given by:

$$p(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}, \tag{4.1}$$

where $k$ is the number of misstatements in the sample, $n$ is the sample size, $N$ is the population size and $K$ is the total number of misstatements assumed in the population. The assumed misstatements $K$ is a whole number, that is, a linear extrapolation of the maximum tolerable misstatement rate (i.e., the performance materiality) $\theta_{max}$ to the total population of size $N$. In the equation below, $\lceil ... \rceil$ is the ceiling function, which means that $\lceil 1.2 \rceil = 2$.

$$K = \lceil \theta_{max} N \rceil. \tag{4.2}$$

Let's consider how to use the hypergeometric likelihood to calculate the minimum sample size needed to reduce the sampling risk to an appropriately low level.

### 4.2.1 Classical Planning

In classical planning using the hypergeometric likelihood, the following statistical model is specified:

$$k \sim \text{Hypergeometric}(n, N, K) \tag{4.3}$$

Given the performance materiality $\theta_{max}$, we can compute $K$ and solve for the minimum sample size $n$ needed to reduce the sampling risk to an appropriately low level. This sample size is also dependent on the number of misstatements that the auditor expects, or tolerates, in the sample.

#### 4.2.1.1 No Expected Misstatements

If the auditor does not expect any misstatements in the sample, they can set $k = 0$, which consequently determines how the sample size can be calculated. For example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$ in a population of $N = 1000$ items, then $K = \lceil 0.03 \cdot 1000 \rceil = 30$ and the minimum sample size under the assumption of no expected misstatements in the sample is $n = 94$.

```
plan <- planning(materiality = 0.03, expected = 0, conf.level = 0.95,
 ↪  likelihood = "hypergeometric", N.units = 1000)
plan
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 94
#> sample size obtained in 95 iterations via method 'hypergeometric'
```

The sample size of 94 can be confirmed by checking that 94 is the minimum integer that results in less than five percent probability of finding no misstatements, given

the assumption that the population misstatement is truly 0.03, or three percent. The `dhyper()` function calculates the probability of observing $k$ missatements in a sample of $n$ items given the assumed hypergeometric distribution with $N$ items and $K$ assumed misstatements in the population. By calculating this probability for $n = 93$, we can show that this sample size is insufficient as the relevant probability is higher than the sampling risk $\alpha$.

```
K <- ceiling(0.03 * 1000)
dhyper(x = 0, m = K, n = 1000 - K, k = 93) < 0.05
#> [1] FALSE
```

However, for $n = 94$ the relevant probability is lower than the sampling risk $\alpha$ and thus the sample size is considered to be sufficient.

```
dhyper(x = 0, m = K, n = 1000 - K, k = 94) < 0.05
#> [1] TRUE
```

We can make this sample size visually intuitive by showing the hypergeometric($k$ | 94, 1000, 30) distribution and highlighting the probability for $k = 0$, see Figure 4.2. This probability is lower than the required sampling risk $\alpha = 0.05$.

```
plot(plan)
```

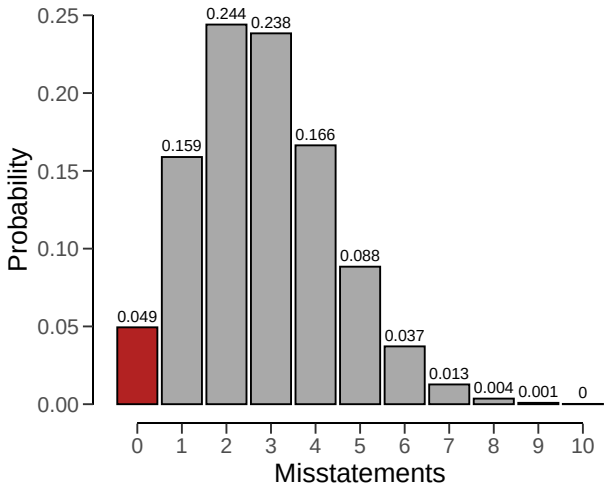



**Figure 4.2.** The hypergeometric($k$ | 94, 1000, 30) distribution showing the probability for $k = 0$ misstatements as a red bar.

The `planning()` function has two additional arguments that are not shown in the call above: `by` and `max`. The argument `by` sets the increment between possible sample sizes for consideration. For example, `by = 10` considers only samples of size 10, 20, 30, and so forth.

```
planning(materiality = 0.03, likelihood = "hypergeometric", N.units =
 ↪  1000, by = 10)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 100
#> sample size obtained in 11 iterations via method 'hypergeometric'
```

The argument `max` sets the sample size at which the algorithm terminates. This can be used to avoid too many iterations of the algorithm at very low values of the performance materiality. For instance, `max = 50` throws an error if more than 100 samples are required.

```
planning(materiality = 0.03, likelihood = "hypergeometric", N.units =
 ↪  1000, max = 50)
#> Error in planning(materiality = 0.03, likelihood =
 ↪  "hypergeometric", N.units = 1000, : the sample size is larger than
 ↪  'max'
```

#### 4.2.1.2 Expected Misstatements

If the auditor expects misstatements in the sample, they can set $k$ to any integer value, which consequently determines how the sample size can be calculated. As another example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$ in a population of $N = 1000$ items, then the required sample size under the assumption of one expected misstatement in the sample is $n = 147$.

```
plan <- planning(materiality = 0.03, expected = 1, likelihood =
 ↪  "hypergeometric", N.units = 1000)
plan
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 147
#> sample size obtained in 146 iterations via method 'hypergeometric'
```

Once again, the sample size of 147 can be confirmed by checking that 147 is the minimum integer that results in less than five percent probability of finding 0 or 1 misstatements, given the assumption that that the population misstatement is truly three percent. By calculating this probability for $n = 146$, we can show that this sample size is insufficient as the relevant probability is higher than the sampling risk $\alpha$.

```
sum(dhyper(x = 0:1, m = K, n = 1000 - K, k = 146)) < 0.05
#> [1] FALSE
```

However, for $n = 147$ the relevant probability is lower than the sampling risk $\alpha$ and thus the sample size is considered to be sufficient.

```
sum(dhyper(x = 0:1, m = K, n = 1000 - K, k = 147)) < 0.05
#> [1] TRUE
```

Like before, we can make this sample size visually intuitive by showing the hypergeometric($k \mid 147, 1000, 30$) distribution and highlighting the probabilities for $k = 0$ and $k = 1$, see Figure 4.3. The sum of these probabilities is lower than the required sampling risk $\alpha = 0.05$.

```
plot(plan)
```

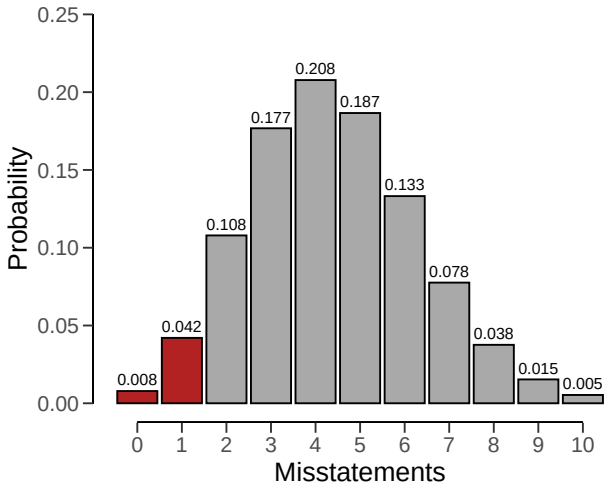



**Figure 4.3.** The hypergeometric($k \mid 147, 1000, 30$) distribution showing the probability for $k = 0$ and $k = 1$ misstatements as a red bar.

### 4.2.2 Bayesian Planning

Performing Bayesian planning with the hypergeometric likelihood (Dyer & Pierce, 1993) requires that you specify a prior distribution for the total misstatements $K$. Practically, this means that you should provide an input for the `prior` argument in the `planning()` function.

Setting `prior = TRUE` performs Bayesian planning using a default prior conjugate to the specified `likelihood` (i.e., a beta-binomial prior). Because this is a Bayesian analysis, the following statistical model is specified:

$$k \sim \text{Hypergeometric}(n, N, K) \tag{4.4}$$
$$K \sim \text{Beta-binomial}(N, \alpha, \beta) \tag{4.5}$$

The beta-binomial prior distribution is the conjugate prior for to the hypergeometric likelihood (see this list of conjugate priors), which means that the posterior distribu-

tion of $K$ can be determined analytically. For example, if the prior distribution for $K$ is:

$$K \sim \text{Beta-binomial}(N, \alpha, \beta) \qquad K = 0, \dots, N \tag{4.6}$$

and the auditor has observed a sample of $n$ items containing $k$ misstatements, then the posterior distribution for $K$ is:

$$K \sim \text{Beta-binomial}(N - n, \alpha + k, \beta + k - n) \qquad K = k, k + 1, \dots, N - n + k. \tag{4.7}$$

#### 4.2.2.1 No Expected Misstatements

Planning for no expected misstatements in the sample can be done by setting the value for the `expected` argument to zero. If we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.1$ in a population of $N = 20$ items, then the required sample size under the assumption of zero expected misstatements in the sample is $n = 15$. The command below uses a default beta-binomial($N$, 1, 1) prior distribution to plan this sample, since `planning()` is given the hypergeometric likelihood.

```
plan <- planning(materiality = 0.1, likelihood = "hypergeometric",
 ↪  N.units = 20, prior = TRUE)
```

The `summary()` function can be used to obatain relevant information about the planning.

```
summary(plan)
#>
#>  Bayesian Audit Sample Planning Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Population size:               20
#>   Materiality:                   0.1
#>   Hypotheses:                    H : θ > 0.1 vs. H : θ < 0.1
#>   Expected:                      0
#>   Likelihood:                    hypergeometric
#>   Prior distribution:            beta-binomial(N = 20, α = 1, β =
 ↪  1)
#>
#> Results:
#>   Minimum sample size:           15
#>   Tolerable errors:              0
#>   Posterior distribution:        beta-binomial(N = 5, α = 1, β =
 ↪  16)
#>   Expected most likely error:    0
#>   Expected upper bound:          0.05
```

```
#>   Expected precision:            0.05
#>   Expected BF :                190
```

You can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function, see Figure 4.4. The expected posterior distribution is the posterior distribution that would occur if you actually observed the planned sample containing the expected misstatements. Note that the posterior distribution is only defined in the range $[k; N - n + k]$, since a part of the population has already been seen.

```
plot(plan)
```

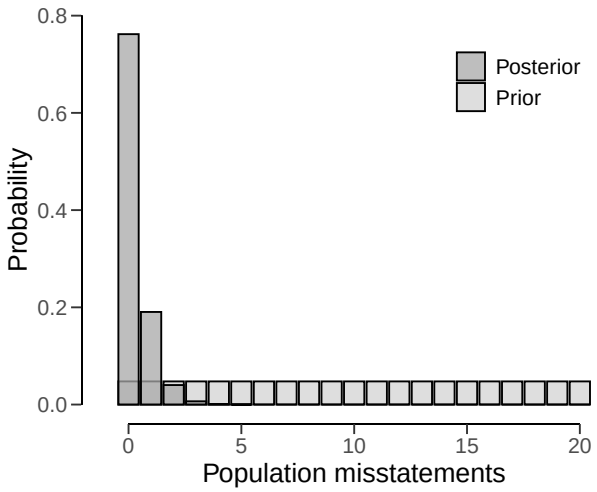



**Figure 4.4.** The beta-binomial prior and posterior distribution on the range $[k; N - n + k]$ after seeing no misstatements in the sample.

#### 4.2.2.2 Expected Misstatements

Planning for expected misstatements in the sample can be done by setting the value for the `expected` argument to a different value than zero. For example, the code below calculates the minimum sample size to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.1$ in a population of $N = 50$ items, given one expected misstatement in the sample. This sample size is $n = 32$.

```
plan <- planning(materiality = 0.1, expected = 1, likelihood =
 ↪  "hypergeometric", N.units = 50, prior = TRUE)
plan
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 32
#> sample size obtained in 31 iterations via method 'hypergeometric' +
 ↪  'prior'
```

Like before, you can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function, see Figure 4.5 below for the output of this call.

```
plot(plan)
```

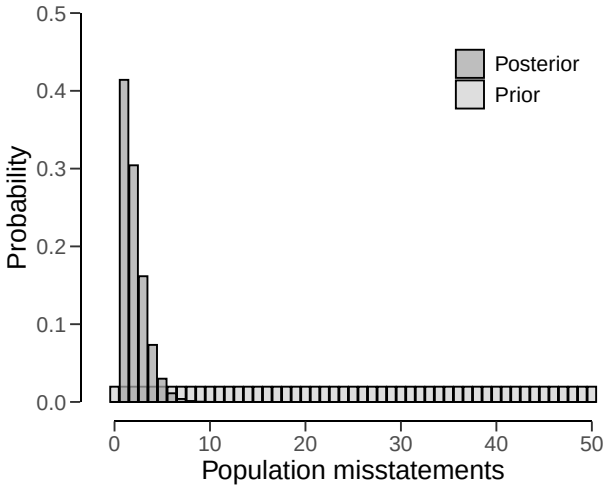



**Figure 4.5.** The beta-binomial prior and posterior distribution on the range $[k; N - n + k]$ after seeing one misstatement in the sample.

## 4.3 The Binomial Likelihood

The binomial distribution is a discrete probability distribution that is commonly used to model the number of events occurring in a fixed number of trials. It is similar to the hypergeometric distribution, however, it assumes that samples are drawn from the population with replacement. For our purpose, we can use the binomial distribution as a likelihood to model the number of misstatements that are expected to be found in the sample.

In audit sampling, the binomial likelihood is often used to approximate the hypergeometric likelihood since it is easier to work with (i.e., it only has two parameters: $\theta$ and $n$, while the hypergeometric has three: $n$, $N$, and $K$). However, the binomial likelihood is more conservative than the hypergeometric likelihood, meaning that resulting sample sizes will be higher.

The probability mass function (PMF) of the binomial distribution is given by:

$$p(k; n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}, \tag{4.8}$$

where $k$ is the number of misstatements in the sample, $n$ is the sample size and $\theta$ is the

probability of misstatement in the population. Let's consider how to use the binomial likelihood to calculate the minimum sample size needed to reduce the sampling risk to an appropriately low level.

### 4.3.1 Classical Planning

In classical planning using the binomial likelihood, the following statistical model is specified:

$$k \sim \text{Binomial}(n, \theta_{max}) \tag{4.9}$$

#### 4.3.1.1 No Expected Misstatements

If the auditor does not expect any misstatements in the sample, they can set $k = 0$, which consequently determines how the sample size can be calculated. Given a performance materiality $\theta_{max}$, we can solve for the minimum sample size $n$ needed to reduce the sampling risk to an appropriately low level. A useful trick to utilize is that, if we do not expect any misstatements in the sample, the formula for the minimum required sample size reduces to:

$$n = \lceil \frac{\ln(\alpha)}{\ln(1 - \theta_{max})} \rceil. \tag{4.10}$$

For example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of zero expected misstatements in the sample is $n = 99$.

```
ceiling(log(1 - 0.95) / log(1 - 0.03))
#> [1] 99
```

In **jfa**, this sample size can be replicated using the following code:

```
plan <- planning(materiality = 0.03, likelihood = "binomial")
plan
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 99
#> sample size obtained in 100 iterations via method 'binomial'
```

The sample size of 99 can be confirmed by checking that 99 is the minimum integer that results in less than five percent probability of finding 0 misstatements, given the assumption that the population misstatement is truly three percent. The `dbinom()` function calculates the probability of observing $k$ missatements in a sample of $n$ items given an assumed misstatement probability $\theta_{max}$. By calculating this probability for $n = 98$, we can show that this sample size is insufficient as the relevant probability is higher than the sampling risk $\alpha$.

```
dbinom(x = 0, size = 98, prob = 0.03) < 0.05
#> [1] FALSE
```

However, for $n = 99$ the relevant probability is lower than the sampling risk $\alpha$ and thus the sample size is considered to be sufficient.

```
dbinom(x = 0, size = 99, prob = 0.03) < 0.05
#> [1] TRUE
```

We can make this sample size visually intuitive by showing the binomial($k \mid 99, 0.03$) distribution and highlighting the probability for $k = 0$, see Figure 4.6. This probability is lower than the required sampling risk $\alpha = 0.05$.

```
plot(plan)
```

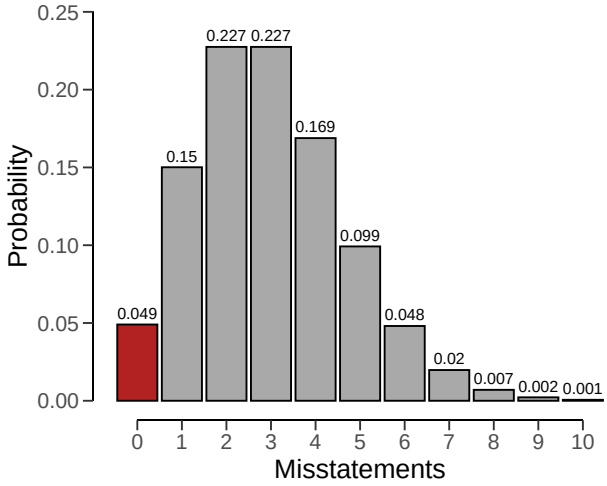



**Figure 4.6.** The binomial($k \mid 99, 0.03$) distribution showing the probability for $k = 0$ misstatement as a red bar.

#### 4.3.1.2 Expected Misstatementss

However, if the number of expected misstatements in the sample is non-zero, it becomes more difficult to solve the formula for $n$. Hence, they will need to set $k$ to a different integer value, which consequently determines how the sample size is calculated. Here, we can iteratively try every value of $n$ and return the smallest integer that satisfies the sampling objectives.

In **jfa**, this can be done by adjusting the `expected` argument in the `planning()` function. For example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of one expected misstatement in the sample is $n = 157$.

```
plan <- planning(materiality = 0.03, expected = 1, likelihood =
 ↪  "binomial")
```

Once again, the sample size of 157 can be confirmed by checking that 157 is the minimum integer that results in less than five percent probability of finding 0 or 1 misstatements, given the assumption that the population misstatement is truly three percent. By calculating this probability for $n = 156$, we can show that this sample size is insufficient as the relevant probability is higher than the sampling risk $\alpha$.

```
sum(dbinom(x = 0:1, size = 156, prob = 0.03)) < 0.05
#> [1] FALSE
```

However, for $n = 157$ the relevant probability is lower than the sampling risk $\alpha$ and thus the sample size is considered to be sufficient.

```
sum(dbinom(x = 0:1, size = 157, prob = 0.03)) < 0.05
#> [1] TRUE
```

Like before, we can make this sample size visually intuitive by showing the binomial($k$ | 157, 0.03) distribution and highlighting the probabilities for $k = 0$ and $k = 1$, see Figure 4.7. The sum of these probabilities is lower than the required sampling risk $\alpha$ = 0.05.

```
plot(plan)
```

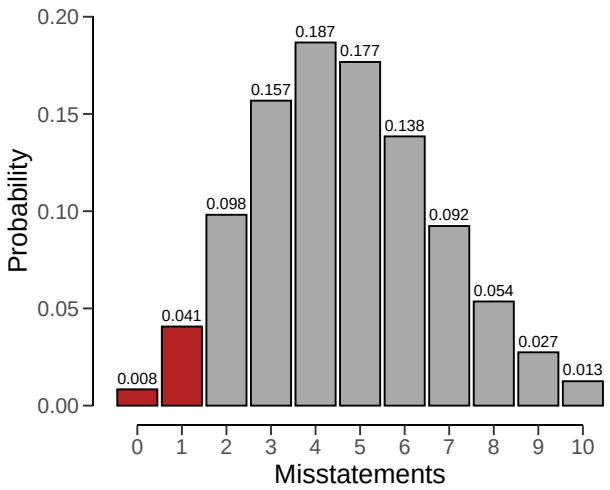



**Figure 4.7.** The binomial($k$ | 157, 0.03) distribution showing the probability for $k$ = 0 and $k = 1$ misstatements as a red bar.

#### 4.3.1.3 Expected Misstatement Rate

When the expected misstatement rate in the sample $\theta_{exp}$ is assessed, the value for $k$ can be determined as $k = n\theta_{exp}$, which consequently determines how the sample size

can be calculated.

To account for the fact that $k$ can have non-integer values in this case, we can use a well-known similarity between the binomial distribution and the beta distribution to plan the sample size. The upper bound for any binomial($k$; $n$, $\theta$) distributed variable can also be obtained via percentiles of the beta($1 + k$, $n - k$) distribution.

For example, the upper bound for a sample of $n = 10$ items containing $k = 2$ misstatements, when calculated via the traditional `binom.test()` is:

```
ub_binom <- binom.test(x = 2, n = 10, p = 0.03, conf.level = 0.95,
 ↪  alternative = "less")$conf.int[2]
ub_binom
#> [1] 0.5069013
```

When calculated via the beta relationship, the upper bound is:

```
ub_beta <- qbeta(p = 0.95, shape1 = 1 + 2, shape2 = 10 - 2)
ub_beta
#> [1] 0.5069013
```

It can be validated that the two approaches result in the same upper bound via:

```
ub_binom == ub_beta
#> [1] TRUE
```

This relationship between the binomial likelihood and the beta distribution is deliberately **not** used in **jfa**. That is because, in the case of the binomial distribution, the auditing standards round the tolerable misstatements upwards to a whole number. For example, if we try to call the `planning()` function with the argument `expected = 1.5`, **jfa** will internally convert this to `expected = 2` and base the sample size on this in order to stay compliant with American Institute of Certified Public Accountants (AICPA) (2016a). The resulting sample size is $n = 208$ in this case.

```
planning(materiality = 0.03, expected = 1.5, likelihood = "binomial")
#> Using 'expected = 2' since 'expected' must be a single integer >= 0
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 208
#> sample size obtained in 206 iterations via method 'binomial'
```

### 4.3.2 Bayesian Planning

Performing Bayesian planning using the binomial likelihood requires that you specify a prior distribution for the parameter $\theta$. Practically, this means that you should provide an input for the `prior` argument in the `planning()` function.

Setting `prior = TRUE` performs Bayesian planning using a default prior conjugate to the specified `likelihood` (i.e., a beta prior). Because this is a Bayesian analysis, the following statistical model is specified:

$$k \sim \text{Binomial}(n, \theta) \tag{4.11}$$
$$\theta \sim \text{Beta}(\alpha, \beta) \tag{4.12}$$

The beta prior distribution is the conjugate prior for the binomial likelihood (see this list of conjugate priors), which means that the posterior distribution of $\theta$ can be determined analytically. For example, if the prior distribution for $\theta$ is:

$$\theta \sim \text{Beta}(\alpha, \beta) \qquad \theta \in [0, 1] \tag{4.13}$$

and the auditor has observed a sample of $n$ items containing $k$ misstatements, then the posterior distribution for $\theta$ is:

$$\theta \sim \text{Beta}(\alpha + k, \beta + n - k) \qquad \theta \in [0, 1]. \tag{4.14}$$

For example, the command below uses a default beta($\alpha = 1$, $\beta = 1$) prior distribution to plan the sample, since `planning()` is given the binomial likelihood. If we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of zero expected misstatements in the sample is $n = 98$.

```
plan <- planning(materiality = 0.03, expected = 0, conf.level = 0.95,
 ↪  likelihood = "binomial", prior = TRUE)
```

The `summary()` function can be used to obatain relevant information about the planning.

```
summary(plan)
#>
#>  Bayesian Audit Sample Planning Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Materiality:                   0.03
#>   Hypotheses:                    H : θ > 0.03 vs. H : θ < 0.03
#>   Expected:                      0
#>   Likelihood:                    binomial
#>   Prior distribution:            beta(α = 1, β = 1)
#>
#> Results:
#>   Minimum sample size:           98
#>   Tolerable errors:              0
#>   Posterior distribution:        beta(α = 1, β = 99)
#>   Expected most likely error:    0
#>   Expected upper bound:          0.029807
#>   Expected precision:            0.029807
#>   Expected BF :                 627.22
```

You can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function, see Figure 4.8. The expected posterior distribution is the posterior distribution that would occur if you actually observed the planned sample containing the expected misstatements.

```
plot(plan)
```

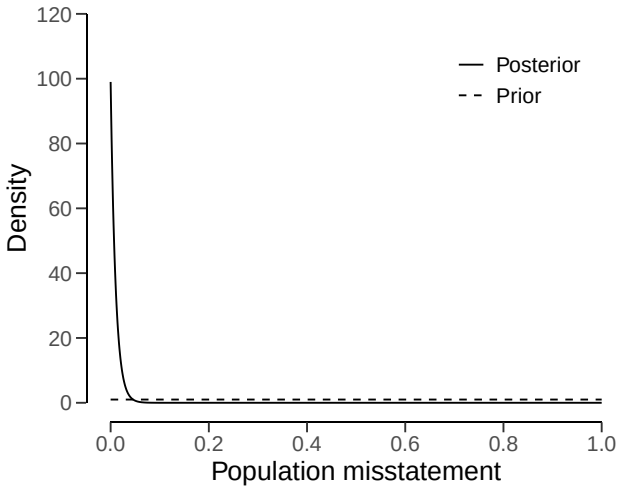



**Figure 4.8.** The beta prior and posterior distribution on the range [0; 1] after seeing no misstatements in the sample of 98 units.

The input for the `prior` argument can also be an object created by the `auditPrior` function. If `planning()` receives a prior for which there is no conjugate likelihood available, it will numerically derive the posterior distribution. For example, the command below uses a Normal(0, 0.05) prior distribution to plan the sample using the binomial likelihood. Because this is a Bayesian analysis, the following statistical model is specified:

$$k \sim \text{Binomial}(n, \theta) \tag{4.15}$$
$$\theta \sim \text{Normal}(\mu = 0, \sigma = 0.05) \tag{4.16}$$

```
prior <- auditPrior(method = "param", likelihood = "normal", alpha =
 ↪  0, beta = 0.05)
plan <- planning(materiality = 0.03, likelihood = "poisson", prior =
 ↪  prior)
```

The `summary()` function can be used to obatain relevant information about the planning.

```
summary(plan)
#>
#>  Bayesian Audit Sample Planning Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Materiality:                   0.03
#>   Hypotheses:                    H : θ > 0.03 vs. H : θ < 0.03
#>   Expected:                      0
#>   Likelihood:                    poisson
#>   Prior distribution:            normal(μ = 0, σ = 0.05)T[0,1]
#>
#> Results:
#>   Minimum sample size:           91
#>   Tolerable errors:              0
#>   Posterior distribution:        Nonparametric
#>   Expected most likely error:    0
#>   Expected upper bound:          0.029
#>   Expected precision:            0.029
#>   Expected BF :                 26.749
```

The resulting sample size under this prior is $n = 90$, a reduction of 8 samples when compared to the default beta(1, 1) prior distribution. Figure 4.9 shows this prior and posterior distribution.

```
plot(plan)
```

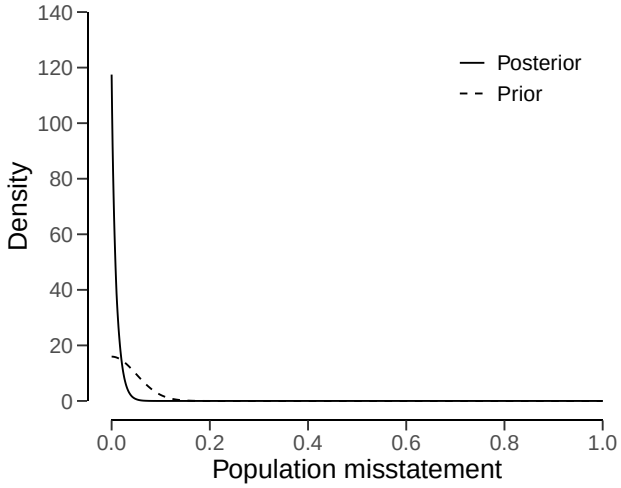



**Figure 4.9.** The normal prior distribution in this example contains risk-reducing information. The posterior distribution has roughly the same upper bound as the beta posterior in the previous example and occurs after seeing no misstatements in a sample of 90 units.

## 4.4 The Poisson Likelihood

The Poisson distribution is a discrete probability distribution that is commonly used to model the number of events occurring in a fixed time or space. We can use the Poisson distribution as a likelihood to model the number of misstatements that are expected to be found in the sample.

In audit sampling, the Poisson likelihood is often used to approximate the binomial likelihood since it is easier to work with (i.e., it only has one parameter: $\lambda$, while the binomial has two parameters: $\theta$ and $n$). However, the Poisson likelihood is more conservative than the binomial likelihood, meaning that resulting sample sizes will be higher.

The probability mass function (PMF) of the Poisson distribution is given by:

$$p(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{4.17}$$

where $k$ is the number of misstatements in the sample, and $\lambda$ is the average number of misstatements expected in the sample. The average number of misstatements is related to the misstatement rate in the population, denoted by $\theta$, and the sample size, $n$, by the following equation:

$$\lambda = n\theta. \tag{4.18}$$

Let's consider how to use the Poisson likelihood to calculate the minimum sample size needed to reduce the sampling risk to an appropriately low level.

### 4.4.1 Classical Planning

In classical planning using the Poisson likelihood, the following statistical model is specified:

$$k \sim \text{Poisson}(n\theta_{max}) \tag{4.19}$$

#### 4.4.1.1 No Expected Misstatements

Given the performance materiality $\theta_{max}$ and the Poisson likelihood, we can solve for the minimum sample size $n$ needed to reduce the sampling risk to an appropriately low level. A useful trick to utilize is that, if we do not expect any misstatements in the sample, the formula for the required sample size reduces to:

$$n = \lceil -\frac{\ln(\alpha)}{\theta_{max}} \rceil. \tag{4.20}$$

For example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of zero expected misstatements in the sample is $n = 100$.

```
ceiling(-log(1 - 0.95) / 0.03)
#> [1] 100
```

In **jfa**, this sample size can be replicated using the following code:

```
plan <- planning(materiality = 0.03, likelihood = "poisson")
```

The sample size of 100 can be confirmed by checking that 100 is the minimum integer that results in less than five percent probability of finding no misstatements, given the assumption that the population misstatement is truly three percent. The `dpois()` function calculates the probability of observing $k$ missatements in a sample of $n$ items given an assumed misstatement probability $\theta_{max}$. By calculating this probability for $n = 99$, we can show that this sample size is insufficient as the relevant probability is higher than the sampling risk $\alpha$.

```
dpois(x = 0, lambda = 99 * 0.03) < 0.05
#> [1] FALSE
```

However, for $n = 100$ the relevant probability is lower than the sampling risk $\alpha$ and thus the sample size is considered to be sufficient.

```
dpois(x = 0, lambda = 100 * 0.03) < 0.05
#> [1] TRUE
```

We can make this visually intuitive by showing the Poisson($k \mid 100 \cdot 0.03$) distribution and highlighting the probability for $k = 0$, see Figure 4.10. This probability is lower than the required sampling risk $\alpha = 0.05$.

```
plot(plan)
```

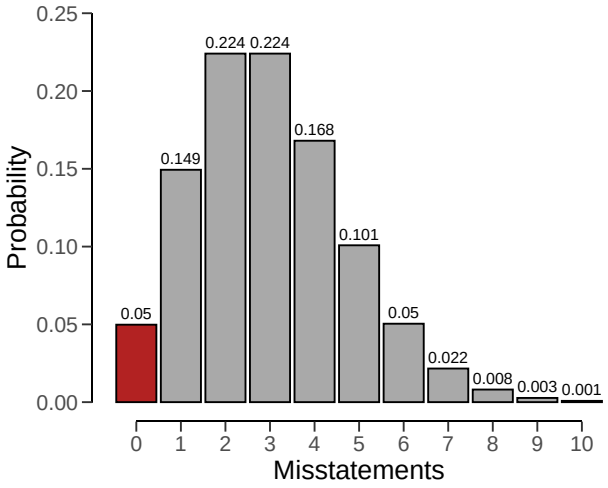



**Figure 4.10.** The Poisson($k \mid 100 \cdot 0.03$) distribution showing the probability for $k = 0$ misstatements as a red bar.

#### 4.4.1.2 Expected Misstatements

However, if the number of expected misstatements in the sample is non-zero, it becomes more difficult to solve the formula for $n$ algebraically. Hence, they will need to set $k$ to a different integer value. Next, we can iteratively try every value of $n$ and return the smallest integer that satisfies the sampling objectives.

For example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of one expected misstatement in the sample is $n = 159$.

```
plan <- planning(materiality = 0.03, expected = 1, likelihood =
 ↪  "poisson")
```

Once again, the sample size of 159 can be confirmed by checking that 159 is the minimum integer that results in less than five percent probability of finding $k = 0$ or $k = 1$ misstatements, given the assumption that the population misstatement is truly three percent. By calculating this probability for $n = 158$, we can show that this sample size is insufficient as the relevant probability is higher than the sampling risk $\alpha$.

```
sum(dpois(x = 0:1, lambda = 158 * 0.03)) < 0.05
#> [1] FALSE
```

However, for $n = 159$ the relevant probability is lower than the sampling risk $\alpha$ and thus the sample size is considered to be sufficient.

```
sum(dpois(x = 0:1, lambda = 159 * 0.03)) < 0.05
#> [1] TRUE
```

Like before, we can make this visually intuitive by showing the Poisson($k \mid 159 \cdot 0.03$) distribution and highlighting the probabilities for $k = 0$ and $k = 1$, see Figure 4.11. The sum of these probabilities is lower than the required sampling risk $\alpha = 0.05$.

```
plot(plan)
```

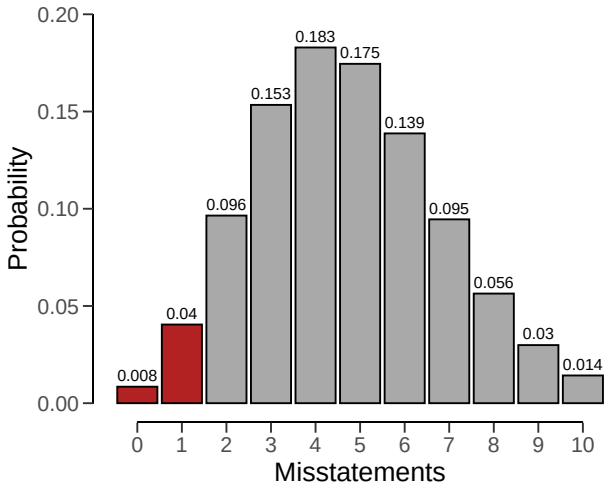



**Figure 4.11.** The Poisson($k \mid 159 \cdot 0.03$) distribution showing the probability for $k = 0$ and $k = 1$ misstatements as a red bar.

#### 4.4.1.3 Expected Misstatement Rate

When the expected misstatements in the sample $\theta_{exp}$ is assessed, the value for $k$ can be determined as $k = n\theta_{exp}$, which consequently determines how the sample size can be calculated.

To account for the fact that $k$ can have non-integer values in this case, we use a well-known similarity between the Poisson distribution and the gamma distribution to plan the sample size. The upper bound for any Poisson($k$; $n\theta$) distributed variable can also be obtained via percentiles of the gamma($1 + k$, $n$) distribution.

For example, the upper bound for a sample of $n = 10$ items containing $k = 2$ misstatements, when calculated via the traditional `poisson.test()` is:

```
ub_pois <- poisson.test(x = 2, T = 10, r = 0.03, alternative =
 ↪  "less")$conf.int[2]
ub_pois
#> [1] 0.6295794
```

When calculated via the relationship with the gamma distribution, the upper bound is:

```
ub_gamma <- qgamma(p = 0.95, shape = 1 + 2, rate = 10)
ub_gamma
#> [1] 0.6295794
```

It can be validated that the two approaches result in the same upper bound via:

```
ub_pois == ub_gamma
#> [1] TRUE
```

This relationship between the Poisson likelihood and the gamma distribution is used under the hood in **jfa**. For example, if we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of 1.5 expected misstatements in the sample is $n = 185$.

```
planning(materiality = 0.03, expected = 1.5, likelihood = "poisson")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 185
#> sample size obtained in 184 iterations via method 'poisson'
```

The sample size of 185 can be confirmed by checking that 185 is the minimum integer that results in less than five percent probability of finding a misstatement rate in the population equal to, or higher than, three percent. By calculating this probability for $n = 184$, we can show that this sample size is insufficient as the relevant upper bound is higher than the performance materiality $\theta_{max}$.

```
qgamma(p = 0.95, shape = 1 + 1.5, rate = 184) < 0.03
#> [1] FALSE
```

However, for $n = 185$ the relevant upper bound is lower than the performance materiality $\theta_{max}$ and thus the sample size is sufficient.

```
qgamma(p = 0.95, shape = 1 + 1.5, rate = 185) < 0.03
#> [1] TRUE
```

### 4.4.2 Bayesian Planning

Performing Bayesian planning with the Poisson likelihood requires that you specify a prior distribution for the parameter $\theta$. Practically, this means that you should provide an input for the `prior` argument in the `planning()` function.

Setting `prior = TRUE` performs Bayesian planning using a default prior conjugate to the specified `likelihood` (i.e., a gamma prior). Because this is a Bayesian analysis, the following statistical model is specified:

$$k \sim \text{Poisson}(n\theta) \tag{4.21}$$
$$\theta \sim \text{Gamma}(\alpha, \beta) \tag{4.22}$$

The gamma prior distribution is the conjugate prior for the Poisson likelihood (see this list of conjugate priors), which means that the posterior distribution of $\theta$ can be determined analytically. For example, if the prior distribution for $\theta$ is:

$$\theta \sim \text{Gamma}(\alpha, \beta) \qquad \theta \in [0, \infty] \tag{4.23}$$

and the auditor has observed a sample of $n$ items containing $k$ misstatements, then the posterior distribution for $\theta$ is:

$$\theta \sim \text{Gamma}(\alpha + k, \beta + n) \qquad \theta \in [0, \infty]. \tag{4.24}$$

For example, the command below uses a default gamma($\alpha = 1$, $\beta = 1$) prior distribution to plan the sample, since `planning()` is given the Poisson likelihood. If we want to achieve an assurance level of 95 percent ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the required sample size under the assumption of zero expected misstatements in the sample is $n = 99$.

```
plan <- planning(materiality = 0.03, likelihood = "poisson", prior =
 ↪  TRUE)
```

The `summary()` function can be used to obatain relevant information about the planning.

```
summary(plan)
#>
#>  Bayesian Audit Sample Planning Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Materiality:                   0.03
#>   Hypotheses:                    H : θ > 0.03 vs. H : θ < 0.03
#>   Expected:                      0
#>   Likelihood:                    poisson
#>   Prior distribution:            gamma(α = 1, β = 1)
#>
#> Results:
#>   Minimum sample size:           99
#>   Tolerable errors:              0
#>   Posterior distribution:        gamma(α = 1, β = 100)
#>   Expected most likely error:    0
#>   Expected upper bound:          0.029957
#>   Expected precision:            0.029957
#>   Expected BF :                 626.69
```

You can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function, see Figure 4.12. The expected posterior distribution is the posterior distribution that would occur if you actually observed the planned sample containing the expected misstatements.

```
plot(plan)
```

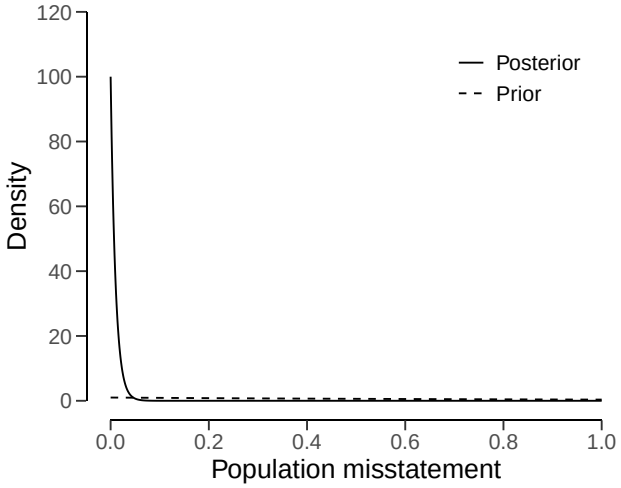



**Figure 4.12.** The gamma prior and posterior distribution on the range $[0; \infty]$ after seeing no misstatements in the sample of 99 units.

The input for the `prior` argument can also be an object created by the `auditPrior` function. If `planning()` receives a prior for which there is no conjugate likelihood available, it will numerically derive the posterior distribution. For example, the command below uses a Normal(0, 0.05) prior distribution to plan the sample using the Poisson likelihood. Concretely, this means that the following statistical model is specified:

$$k \sim \text{Poisson}(n\theta) \tag{4.25}$$
$$\theta \sim \text{Normal}(\mu = 0, \sigma = 0.05) \tag{4.26}$$

```
prior <- auditPrior(method = "param", likelihood = "normal", alpha =
 ↪  0, beta = 0.05)
plan <- planning(materiality = 0.03, likelihood = "poisson", prior =
 ↪  prior)
```

The `summary()` function can be used to obatain relevant information about the planning.

```
summary(plan)
#>
#>  Bayesian Audit Sample Planning Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Materiality:                   0.03
#>   Hypotheses:                    H : θ > 0.03 vs. H : θ < 0.03
#>   Expected:                      0
#>   Likelihood:                    poisson
#>   Prior distribution:            normal(μ = 0, σ = 0.05)T[0,1]
#>
#> Results:
#>   Minimum sample size:           91
#>   Tolerable errors:              0
#>   Posterior distribution:        Nonparametric
#>   Expected most likely error:    0
#>   Expected upper bound:          0.029
#>   Expected precision:            0.029
#>   Expected BF :                 26.063
```

The resulting sample size under this prior is $n = 91$, a reduction of 8 samples when compared to the default gamma(1, 1) prior. Figure 4.13 shows this prior and posterior distribution.

```
plot(plan)
```

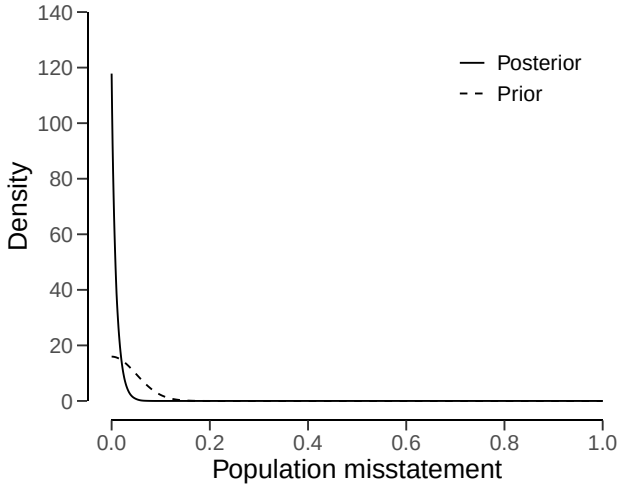



**Figure 4.13.** The normal prior distribution in this example contains risk-reducing information. The posterior distribution has roughly the same upper bound as the one in the previous example and occurs after seeing no misstatements in the sample of 91 units.

## 4.5 Multi-Stage Sampling

A multi-stage sampling plan, as opposed to a single-stage one, allows for an intermediate evaluation of the sample. In the first stage of a multi-stage sampling plan, the auditor selects an initial sample of size $n_1$. If this sample contains a tolerable number of misstatements (usually 0), the auditor can approve the population. However, if the sample contains more misstatements than the tolerable number, the auditor cannot approve the population. In such a scenario, the initial sample is supplemented with a second sample of $n_2$ items. If this additional sample contains a tolerable number of misstatements, the population can still be approved. If not, the population should either be rejected or a third sample of $n_3$ items should be added.

In the classical (i.e., frequentist) methodology, multi-stage sampling plans can be formulated by decomposing the sampling risk into multiple components. For example, if the auditor initially plans for $k = 0$ misstatements but considers extending the sample if $k = 1$ misstatement is discovered, then the probability of erroneously rejecting the hypothesis of material misstatement comprises:

- $p(k \leq 0|n_1, \theta_{max})$: The probability of finding $k = 0$ misstatements in the first sample of $n_1$ items; plus
- $p(k = 1|n_1, \theta_{max})$: The probability of finding $k = 1$ misstatement in the first sample of $n_1$ items; multiplied by
- $p(k \leq 0|n_2, \theta_{max})$: The probability finding $k = 0$ misstatements in the second sample of $n_2$ items.

The sum of these probabilities should be less than or equal to the sampling risk $\alpha$.

$$p(k \leq 0|n_1, \theta_{max}) + p(k = 1|n_1, \theta_{max}) \cdot p(k \leq 0|n_2, \theta_{max}) \leq \alpha \tag{4.27}$$

To constrain the problem, **jfa** sets the number of samples in the extension equal to the initial sample size ($n_1 = n_2 = n_s$). This implies that the sample size per stage in this two-stage sampling plan is the smallest integer $n_s$ that fulfills the following condition.

$$p(k \leq 0|n_s, \theta_{max}) + p(k = 1|n_s, \theta_{max}) \cdot p(k \leq 0|n_s, \theta_{max}) \leq \alpha \tag{4.28}$$

In **jfa**, multi-stage sampling plans can be calculated by supplying an integer vector of misstatements, after which each stage should be extended, to the `planning()` function via its `expected` argument. For example, the following code calculates the required sample size if the auditor initially plans for $k = 0$ misstatements but considers extending the sample if $k = 1$ misstatement is discovered. The required sample size per stage is $n_s = 103$, resulting in a total sample size (if both stages are necessary) of $n = 206$.

```
multiplan <- planning(materiality = 0.03, likelihood = "binomial",
 ↪  expected = c(1, 0))
print(multiplan)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 206 (103 per stage)
#> sample size obtained in 102 iterations via method 'binomial' +
 ↪  'sequential'
```

To confirm this calculation, we need to ensure that that the probability of incorrectly rejecting the null hypothesis of material misstatement, under the binomial distribution, is less than the sampling risk $\alpha = 0.05$.

```
p_k0_n1 <- pbinom(q = 0, size = 103, prob = 0.03)
p_k1_n1 <- dbinom(x = 1, size = 103, prob = 0.03)
p_k0_n2 <- pbinom(q = 0, size = 103, prob = 0.03)
p_k0_n1 + p_k1_n1 * p_k0_n2 < 0.05
#> [1] TRUE
```

The minimum sample size per stage, $n_2 = 103$, is only slighly larger than the minimum sample size for the first stage if the auditor opts for a single-stage sampling plan expecting $k = 0$, which is $n = 99$.

```
planning(materiality = 0.03, likelihood = "binomial", expected = 0)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 99
#> sample size obtained in 100 iterations via method 'binomial'
```

However, the total sample size, $n = 206$, is considerably larger than the minimum sample size if the auditor opts for a single-stage sampling plan expecting $k = 1$, which is $n = 157$. This illustrates the cost of allowing a sample size extension in the classical approach.

```
planning(materiality = 0.03, likelihood = "binomial", expected = 1)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 157
#> sample size obtained in 156 iterations via method 'binomial'
```

Note that the sample size per stage $n_s = 103$ is determined based on $n_s = n_1 = n_2$. However, it is important to note that this equality is not mandatory. If the auditor opts for an initial sample size larger than $n_s$, they can decrease the size of the follow-up sample. To illustrate this tradeoff, you can use the `plot()` function, see Figure 4.14. The figure includes textual information specifying the total sample size under the equality constrained multi-stage sampling plan (red) and under multiple choices for the initial sample size $n_1$.

```
plot(multiplan)
```

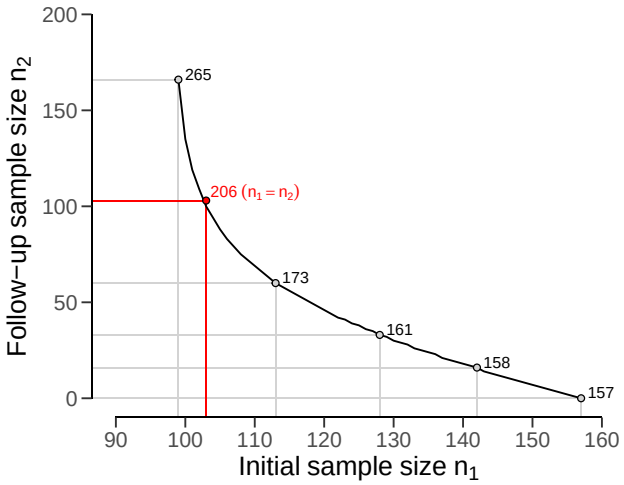



**Figure 4.14.** The preferred initial sample size $n_1$ versus the required follow-up sample size $n_2$.

As another example, consider a three-stage sampling plan where the auditor plans to extend the sample after finding $k = 1$ misstatement in the first stage, extend further if they find $k = 1$ misstatement in the second stage, and still be able to approve the population if they find $k = 0$ misstatements in the third stage. The required sample size in each of the three stages is $n_s = 208$.

```
planning(materiality = 0.03, likelihood = "binomial", expected = c(3,
 ↪  1, 0))
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 624 (208 per stage)
#> sample size obtained in 204 iterations via method 'binomial' +
 ↪  'sequential'
```

This can be confirmed by ensuring that the probability of incorrectly rejecting the null hypothesis of the population containing three percent misstatement is below the acceptable sampling risk $\alpha = 0.05$.

```
p_k2_n1 <- pbinom(q = 2, size = 208, prob = 0.03)
p_k3_n1 <- dbinom(x = 3, size = 208, prob = 0.03)
p_k0_n2 <- pbinom(q = 0, size = 208, prob = 0.03)
p_k1_n2 <- dbinom(x = 1, size = 208, prob = 0.03)
p_k0_n3 <- pbinom(q = 0, size = 208, prob = 0.03)
p_k2_n1 + p_k3_n1 * p_k0_n2 + p_k3_n1 * p_k1_n2 * p_k0_n3 < 0.05
#> [1] TRUE
```

Unlike the previous example, the minimum sample size per stage, $n_s = 208$, is not larger than the minimum sample size for the first stage if the auditor decides to use a single-stage sampling plan expecting $k = 2$ misstatements, which is also $n = 208$.

```
planning(materiality = 0.03, likelihood = "binomial", expected = 2)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 208
#> sample size obtained in 206 iterations via method 'binomial'
```

The Bayesian approach to audit sampling uses the posterior distribution from the observed sample as a prior distribution for a potential second sample in an iterative manner. For this reason, this approach does not affect the sampling risk based on the number of tests (Rouder, 2014). Hence, in the Bayesian framework, it is entirely appropriate to start sampling until there is enough evidence to make a decision (Edwards et al., 1963). This means that if you find $k = 1$ misstatement in the initial $n_1 = 99$ samples, calculating the sample size extension simply involves computing the single-stage sample size under the expectation of $k = 1$ tolerable misstatement. The total required sample size then becomes $n = 157$, and the sample size extension therefore amounts to $n_2 = 158 - 99 = 59$ items.

```
planning(materiality = 0.03, expected = 1, prior = TRUE)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 158
#> sample size obtained in 157 iterations via method 'poisson' +
 ↪  'prior'
```

## 4.6 Prior Distributions

In principle, any distribution that covers the range of $\theta$ can be used as a prior distribution. However, some distributions are more suitable than others. For instance, the beta-binomial, beta and gamma distributions are all commonly used because they are so-called conjugate distributions, that is, they stay in the same family when updated by the data. The **jfa** package provides the ability to construct a prior distribution for audit sampling. More specifically, the `auditPrior()` function is used to specify a prior distribution that can be used as input for the `prior` argument in the `planning()` (and `evaluation()`) function. Below is an enumeration of the several ways that a prior distribution can be constructed using the `auditPrior` function.

### 4.6.1 Default Prior

The default prior distributions are created using `method = 'default'`. There are no explicit rules for what constitutes a default prior. However, **jfa**'s default priors satisfy two criteria. First, they contain relatively little information about the population misstatement and, second, they are proper (i.e., they integrate to 1). For completeness, all default priors in **jfa** are provided in the following list.

- `likelihood = 'poisson'`: gamma($\alpha = 1$, $\beta = 1$)
- `likelihood = 'binomial'`: beta($\alpha = 1$, $\beta = 1$)
- `likelihood = 'hypergeometric'`: beta-binomial($N$, $\alpha = 1$, $\beta = 1$)
- `likelihood = 'normal'`: normal($\mu = 0$, $\sigma = 1000$)
- `likelihood = 'uniform'`: uniform($min = 0$, $max = 1$)
- `likelihood = 'cauchy'`: Cauchy($x_0 = 0$, $\gamma = 1000$)
- `likelihood = 't'`: Student-t($df = 1$)
- `likelihood = 'chisq'`: chi-squared($df = 1$)
- `likelihood = 'exponential'`: exponential($\lambda = 1$)

For instance, to create a default prior distribution using the binomial likelihood (i.e., a beta(1, 1) prior), you can use the following code that creates a prior distribution and stores it in the `prior` object. You can then use the `summary()` function to obtain relevant information about the prior distribution.

```
prior <- auditPrior(method = "default", likelihood = "binomial")
summary(prior)
#>
#>  Prior Distribution Summary
#>
#> Options:
#>   Likelihood:                    binomial
#>   Specifics:                     default prior
#>
#> Results:
#>   Functional form:               beta(α = 1, β = 1)
#>   Mode:                          NaN
```

```
#>   Mean:                          0.5
#>   Median:                        0.5
#>   Variance:                      0.083333
#>   Skewness:                      0
#>   Information entropy (nat):     0
#>   95 percent upper bound:        0.95
#>   Precision:                     NaN
```

All prior distributions can be visually inspected via the `plot()` function, see Figure 4.15.

```
plot(prior)
```

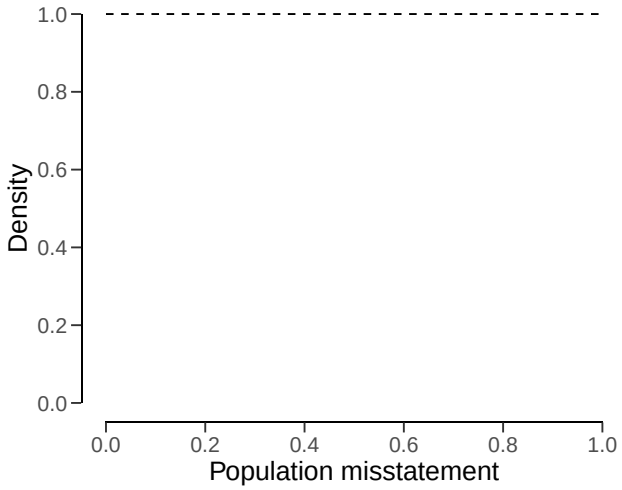



**Figure 4.15.** The default beta(1, 1) prior distribution.

Furthermore, the `predict()` function produces the predictions of the prior distribution on the data level for a sample of `n` items. For example, the command below requests the prediction of the default beta(1, 1) prior for a hypothetical sample of 6 items.

```
predict(prior, n = 6)
#>       x=0       x=1       x=2       x=3       x=4       x=5
 ↪  x=6
#> 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571
 ↪  0.1428571
```

The predictions of the prior distribution can be visualized using the `plot()` function, see Figure 4.16.

```
plot(predict(prior, n = 10))
```

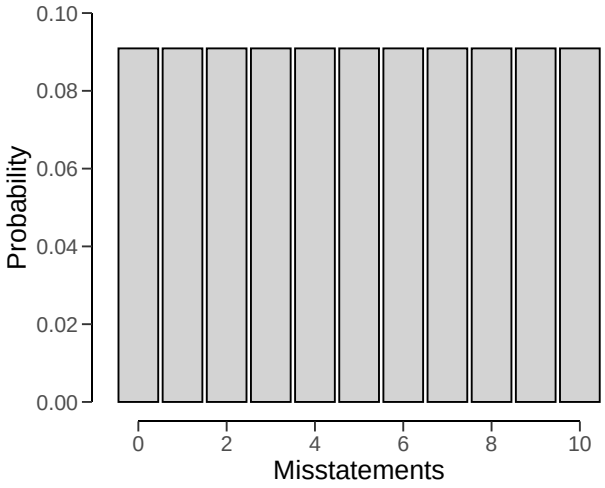



**Figure 4.16.** The predictions of the beta(1, 1) prior distribution concerming the possible misstatements in an an intended sample of $n = 6$.

### 4.6.2 Parametric Prior

You can manually specify the parameters of the prior distribution with `method = 'param'` and the `alpha` and `beta` arguments, which correspond to the first and (optionally) second parameter of the prior as described above. For example, the commands below create a beta(2, 10) prior distribution, a normal(0.025, 0.05) prior distribution and a Student-t(0.01) prior distribution.

```
auditPrior(method = "param", likelihood = "binomial", alpha = 2, beta
 ↪  = 10)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 2, β = 10)
#> parameters obtained via method 'param'
auditPrior(method = "param", likelihood = "normal", alpha = 0.025,
 ↪  beta = 0.05)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: normal(μ = 0.025, σ = 0.05)T[0,1]
#> parameters obtained via method 'param'
auditPrior(method = "param", likelihood = "t", alpha = 0.01)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: Student-t(df = 0.01)T[0,1]
#> parameters obtained via method 'param'
```

### 4.6.3 Improper Prior

You can construct an improper prior distribution with classical properties using `method = 'strict'`. The posterior distribution of from this prior yields the same results as the classical methodology with respect to sample sizes and upper limits, but is only proper once a single non-misstated unit is present in the sample (Derks et al., 2022a). For example, the command below creates an improper beta(1, 0) prior distribution. This method requires the `poisson`, `binomial` or `hypergeometric` likelihood.

```
auditPrior(method = "strict", likelihood = "binomial")
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 0)
#> parameters obtained via method 'strict'
```

### 4.6.4 Impartial Prior

You can incorporate the assumption that tolerable misstatement is equally likely as intolerable misstatement (Derks et al., 2022a) using `method = 'impartial'`. For example, the command below creates an impartial beta prior distribution for a performance materiality of five percent. This method requires that you specify a value for the `materiality`.

```
auditPrior(method = "impartial", likelihood = "binomial", materiality
 ↪  = 0.05)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 13.513)
#> parameters obtained via method 'impartial'
```

### 4.6.5 Probability of Tolerable Misstatement

You can manually assign prior probabilities to the hypothesis of tolerable misstatement and the hypotheses of intolerable misstatement (Derks, de Swart, van Batenburg, et al., 2021) with `method = 'hyp'` in combination with `p.hmin`. For example, the command below incorporates the information that the hypothesis of tolerable misstatement has a pror probability of 60 percent into a beta distribution. Naturally, this method requires that you specify a value for the `materiality`.

```
auditPrior(method = "hyp", likelihood = "binomial", materiality =
 ↪  0.05, p.hmin = 0.6)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 17.864)
#> parameters obtained via method 'hyp'
```

### 4.6.6 Audit Risk Model

You can translate risk assessments from the Audit Risk Model (inherent risk and internal control risk) into a prior distribution (Derks, de Swart, van Batenburg, et al., 2021) using `method = 'arm'` in combination with the `ir` and `cr` arguments. For example, the command below incorporates the information that the inherent risk is equal to 90 percent and internal control risk is equal to 60 percent into a beta prior distribution. This method requires the `poisson`, `binomial` or `hypergeometric` likelihood.

```
auditPrior(method = "arm", likelihood = "binomial", materiality =
 ↪  0.05, ir = 0.9, cr = 0.6)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 12)
#> parameters obtained via method 'arm'
```

### 4.6.7 Bayesian Risk Assessment Model

You can incorporate information about the mode and the upper bound of the prior distribution using `method = 'bram'`. For example, the code below incorporates the information that the mode of the prior distribution is one percent and the upper bound is 60 percent into a beta prior distribution. This method requires the `poisson`, `binomial` or `hypergeometric` likelihood.

```
auditPrior(method = "bram", likelihood = "binomial", materiality =
 ↪  0.05, expected = 0.01, ub = 0.6)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1.023, β = 3.317)
#> parameters obtained via method 'bram'
```

### 4.6.8 Earlier Sample

You can incorporate information from an earlier sample into the prior distribution (Derks, de Swart, van Batenburg, et al., 2021) using `method = 'sample'` in combination with `x` and `n`. For example, the command below incorporates the information from an earlier sample of 30 items in which 0 misstatements were found into a beta prior distribution. This method requires the `poisson`, `binomial` or `hypergeometric` likelihood.

```
auditPrior(method = "sample", likelihood = "binomial", x = 0, n = 30)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 30)
#> parameters obtained via method 'sample'
```

### 4.6.9 Weighted Earlier Sample

You can incorporate information from last years results, weighted by a factor (Derks, de Swart, van Batenburg, et al., 2021), into the prior distribution using `method = 'factor'` in combination with `x` and `n`. For example, the command below incorporates the information from a last years results (a sample of 58 items in which 0 misstatements were found), weighted by a factor 0.7, into a beta prior distribution. This method requires the `poisson`, `binomial` or `hypergeometric` likelihood.

```
auditPrior(method = "factor", likelihood = "binomial", x = 0, n = 58,
 ↪  factor = 0.7)
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 40.6)
#> parameters obtained via method 'factor'
```

### 4.6.10 Nonparametric Prior

You can base the prior on samples of the prior distribution using `method = 'nonparam'` in combination with `samples`. For example, the command below creates a prior on 1000 samples of a beta(1, 10) distribution. The `likelihood` argument is not required and will be ignored in this method.

```
auditPrior(method = "nonparam", samples = stats::rbeta(1000, 1, 10))
#>
#>  Prior Distribution for Audit Sampling
#>
#> functional form: Nonparametric
#> parameters obtained via method 'nonparam'
```

## 4.7 Practical Examples

This section contains practical examples of how to conduct the planning of statistical audit samples and demonstrates how to set up a prior distribution based on various types of relevant audit information.

### 4.7.1 Audit Risk Model

In our first example, an auditor is performing tests of details on a population of the auditee. For instance, let's say an auditor is performing an audit on a company's accounts payable transactions. The company has a total of $N = 1000$ accounts payable transactions for the year. Rather than testing all 1000 transactions, the auditor can choose to test a sample of the transactions. The performance materiality for the payable transactions account is set to $\theta_{max} = 0.03$ (3 percent), and the audit risk is set to $\alpha = 0.05$, or 5 percent. Based on the results of last years audit, where the most likely estimate of the misstatement was one percent, the auditor wants to tolerate

one percent misstatements in the sample before giving an unqualified opinion on the population.

```
ar <- 0.05 # Audit risk
materiality <- 0.03 # Performance materiality
expected <- 0.01 # Tolerable deviation rate
```

Before tests of details, the auditor has assessed risk of material misstatement via the audit risk model. In this example, the auditor has assessed the effectiveness of the company's internal controls, such as its segregation of duties and its risk management processes, and has determined that they are sufficient to prevent or detect material misstatements. Because the internal control systems were effective, the auditor assesses the control risk as medium. The auditor's firm defines the risk categories low, medium, and high respectively as 50 percent, 60 percent, and 100 percent. According to the Audit Risk Model, the detection risk can be calculated as a function of the audit risk, the inherent risk and the control risk.

```
ir <- 1 # Inherent risk
cr <- 0.6 # Control risk
dr <- ar / (ir * cr) # Detection risk
dr
#> [1] 0.08333333
```

By using the detection risk of 8.33 percent as the sampling risk for this population, the auditor can plan for a sample while taking into account the risk-reducing information. The required minimum sample size is 174 in this case.

```
planning(materiality = 0.03, expected = expected, conf.level = 1 - dr)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 174
#> sample size obtained in 175 iterations via method 'poisson'
```

The example above is a frequentist one. However, the auditor is free to apply a Bayesian philosophy in planning the sample. For example, the risk assessments from the ARM can be incorporated into a prior distribution. This can be done using `method = "arm"` in the `auditPrior()` function, which takes the values of the inherent risk probability `ir` and the control risk probability `cr`. Hence, the prior distribution in this example can be constructed using the following command:

```
prior <- auditPrior(method = "arm", materiality = 0.03, expected =
 ↪  expected, ir = ir, cr = cr)
```

The `summary()` function can be used to obtain relevant information about the prior distribution.

```
summary(prior)
#>
#>  Prior Distribution Summary
```

```
#>
#> Options:
#>   Likelihood:                    poisson
#>   Specifics:                     ir = 1; cr = 0.6; dr = 0.0833333
#>
#> Results:
#>   Functional form:               gamma(α = 1.46, β = 46)
#>   Mode:                          0.01
#>   Mean:                          0.031739
#>   Median:                        0.024859
#>   Variance:                      0.00069
#>   Skewness:                      1.6552
#>   Information entropy (nat):     -2.4894
#>   95 percent upper bound:        0.08343
#>   Precision:                     0.07343
```

Furthermore, the prior distribution can be visualized with a call to the `plot()` function, see Figure 4.17.

```
plot(prior)
```

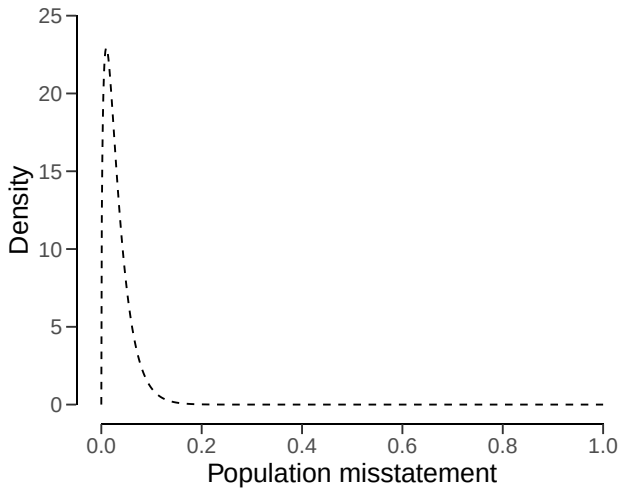



**Figure 4.17.** The prior distribution constructed on the basis of the assessments of inherent risk and control risk.

By using the prior distribution to incorporate the assessments of the inherent risk and the control risk, the auditor can plan a sample while taking into account the risk-reducing information. The required minimum sample size is also 174 in this case.

```
planning(materiality = 0.03, expected = expected, conf.level = 1 - ar,
 ↪  prior = prior)
```

```
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 174
#> sample size obtained in 175 iterations via method 'poisson' +
 ↪  'prior'
```

### 4.7.2 Benchmark Analysis

The auditor may incorporate information obtained through analytical procedures (Derks, de Swart, van Batenburg, et al., 2021), such as a benchmark analysis, into the prior distribution for $\theta$. While we have previously discussed methods for constructing a prior distribution based on existing knowledge, there is no set procedure for incorporating information obtained through analytical procedures, as these procedures can vary significantly depending on the type of information being incorporated into the prior distribution. Therefore, it is important to thoroughly substantiate the data and assumptions used in this approach and to carefully consider how these assumptions are incorporated into the prior distribution.

One way to construct a prior distribution on the basis of data is through the use of regression models, such as benchmarking the relationship between sales and costs of sales within the auditee's specific industry sector. The **jfa** package includes a data set `benchmark` that can be used for this example.

```
data(benchmark)
head(benchmark)
#>       sales costofsales
#> 1 186273256   140755372
#> 2 336491541   248675452
#> 3 222693077   164299866
#> 4 364905221   285768790
#> 5 382140185   280187371
#> 6 113666950   101552955
```

The auditee's the sum of the sales is $298,112,312 and the sum of the booked costs of sales is $223,994,405, respectively. This is indicated by a blue dot in Figure 4.18 below, which visualizes the industry sales versus the cost of sales.

```
C_real <- 223994405
```

The relationship between the sales $S$ and the cost of sales $C$ can be modelled by a linear equation:

$$C = \beta_0 + \beta_1 \cdot S + \epsilon. \tag{4.29}$$

In practice, this relationship is often more complex than is presented above, and the auditor must carefully construct and evaluate the applied regression model. However,

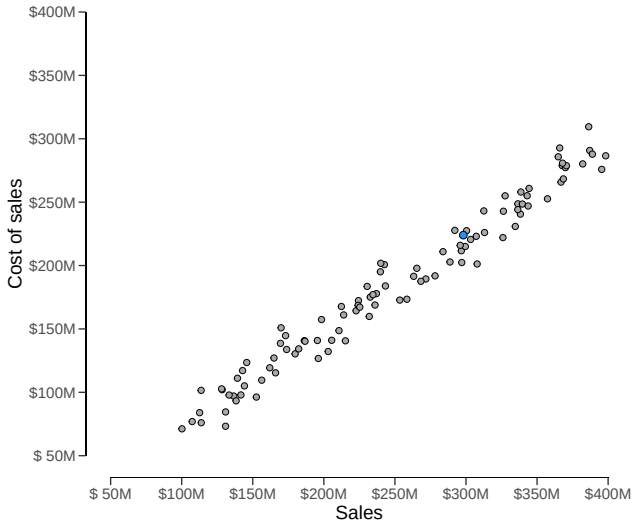



**Figure 4.18.** Scatter plot of the industry sales versus the cost of sales.

for ease of understanding we will continue our example with this toy model. The auditor can estimate the regression model using the following command:

```
fit <- lm(costofsales ~ 1 + sales, data = benchmark)
summary(fit)
#>
#> Call:
#> lm(formula = costofsales ~ 1 + sales, data = benchmark)
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -25736696  -7052141   -226945   6857840  25498106
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 2.413e+05  3.455e+06    0.07    0.944
#> sales       7.366e-01  1.310e-02   56.21   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 11150000 on 98 degrees of freedom
#> Multiple R-squared:  0.9699, Adjusted R-squared:  0.9696
#> F-statistic:  3160 on 1 and 98 DF,  p-value: < 2.2e-16
```

The predicted cost of sales for the auditee, based on the industry benchmark, can be computed as follows:

```
C_pred <- predict(fit, newdata = data.frame(sales = 298112312),
 ↪  interval = "prediction", level = 0.90)[1]
C_pred
#> [1] 219817866
```

The fitted regression line and the predicted cost of sales (red dot) are visualized in Figure 4.19 below.

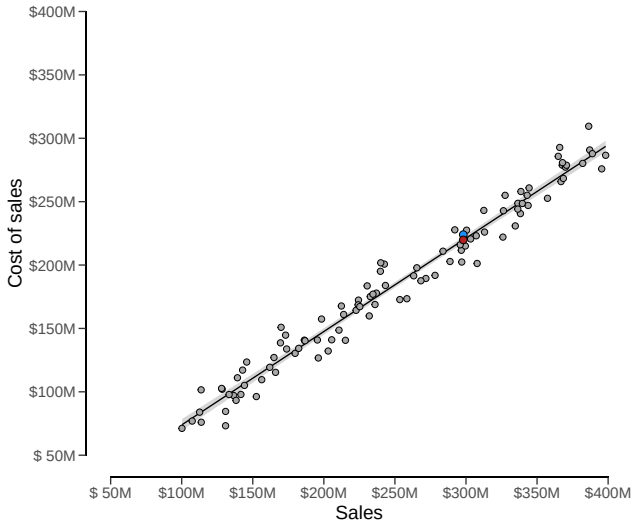



**Figure 4.19.** Scatter plot of the industry sales versus the cost of sales including the regression line and the auditee's (predicted) cost of sales.

The prior distribution can be justified by the data and the auditee's numerical prediction of the cost of sales. In this analytical procedure, the prior distribution on $\theta$ can utilize the relative error distribution from the linear regression. This relative error distribution, which is a Normal($\mu$, $\sigma$) distribution, captures the uncertainty of the prediction of the cost of sales through the use of linear regression, scaled to be a percentage of the total cost of sales. The mean $\mu$ of the prior distribution is determined by the relative deviation of the auditee's booked cost of sales when compared to the predicted cost of sales according to the benchmark data $\frac{C-\hat{C}}{C}$.

```
mu <- (C_real - C_pred) / C_real
mu
#> [1] 0.01864573
```

The standard deviation of the prior distribution is expressed through the standard deviation of the distribution of $\epsilon$:

```
stdev <- sd(fit$residuals) / C_real
stdev
#> [1] 0.04951199
```

The Normal(0.019, 0.05) prior distribution can be constructed through a call to `auditPrior()`, where the likelihood of the prior is specified as `normal`. We call the function with `method = "param"` to manually specify the parameters of the prior distribution.

```
prior <- auditPrior(method = "param", likelihood = "normal", alpha =
 ↪  mu, beta = stdev)
summary(prior)
#>
#>  Prior Distribution Summary
#>
#> Options:
#>   Likelihood:                    normal
#>   Specifics:                     α = 0.0186457; β = 0.049512
#>
#> Results:
#>   Functional form:               normal(μ = 0.019, σ = 0.05)T[0,1]
#>   Mode:                          0.018646
#>   Mean:                          0.047096
#>   Median:                        0.041335
#>   Variance:                      0.0011116
#>   Skewness:                      NA
#>   Information entropy (nat):     -2.1306
#>   95 percent upper bound:        0.11012
#>   Precision:                     0.091473
```

The specified prior distribution can be visualized using the `plot()` function, see Figure 4.20.

```
plot(prior)
```

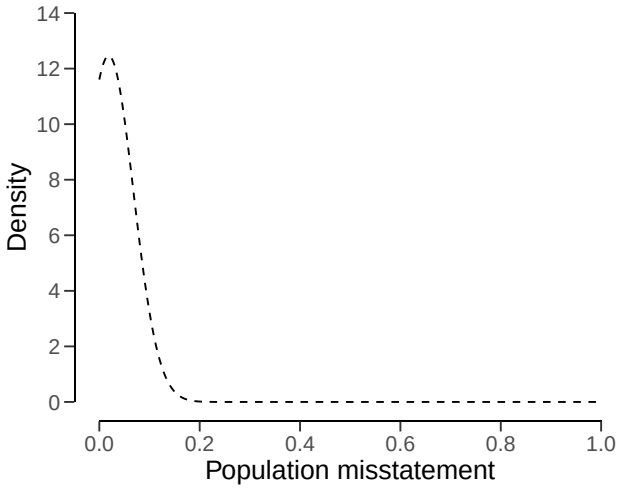



**Figure 4.20.** The prior distribution constructed on the basis of the benchmark analysis.

The performance materiality for this example is set to $\theta_{max} = 0.05$, or five percent, and the audit risk is set to $\alpha = 0.05$, or five percent. By using this prior distribution, the required minimum sample size is $n = 50$.

```
plan <- planning(materiality = 0.05, likelihood = "binomial", prior =
 ↪  prior)
plan
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 50
#> sample size obtained in 51 iterations via method 'binomial' +
 ↪  'prior'
```

You can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function, see Figure 4.21.

```
plot(plan)
```

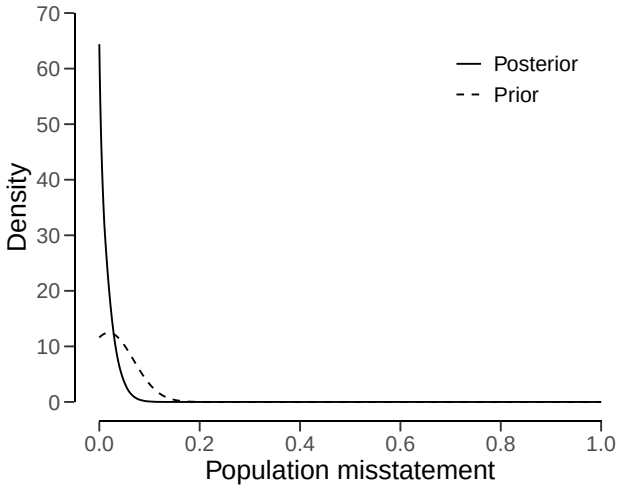



**Figure 4.21.** The prior and expected posterior distribution for this example after seeing a sample of $n = 50$ containing no misstatements.

By using a frequentist approach, the required minimum sample size is $n = 59$. Thus, by performing the analytical procedure and incorporating this information into the prior distribution, the auditor has achieved a reduction in sample size of 9 items.

```
plan <- planning(materiality = 0.05, likelihood = "binomial")
plan
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 59
#> sample size obtained in 60 iterations via method 'binomial'
```

### 4.7.3 Predictive Modeling

As a final example, we consider an example wehre the auditor incorporates information about the probability of misstatement obtained through a predictive analysis into the prior distribution for $\theta$. In this example, the auditor is conducting an audit on a long-term client that they have been working with for fifteen years. Hence, the auditor has access to a historycal data set called `history`, which contains the samples from all of the previous audits that have been conducted on this auditee over the past fifteen years.

```
history <-
 ↪  read.csv("https://github.com/koenderks/sasr/raw/master/data/ch3_history.csv",
 ↪  colClasses = c("factor", "numeric", "numeric"))
head(history)
#>   k  ftes days
#> 1 0  2.09 1.83
```

```
#> 2 0 17.51 5.02
#> 3 0 17.29 8.47
#> 4 0 23.88 8.31
#> 5 0 15.31 8.25
#> 6 0 18.28 4.47
```

It should be noted that for all historical sample items, there are three known characteristics: whether they contained a misstatement (designated as `k`), the number of full-time equivalent employees (FTEs) who had access to that item within the internal computer systems of the auditee (designated as `ftes`), and the number of days that the item was outstanding (designated as `days`). Additionally, the `ftes` and `days` characteristics are also known for all items in the current year's population. Of course, it is unknown if any misstatements exist within the population of the current year.

```
population <-
 ↪  read.csv("https://github.com/koenderks/sasr/raw/master/data/ch3_population.csv")
head(population)
#>      ID bookValue ftes days
#> 1 82884    242.61   14    4
#> 2 25064    642.99   11    4
#> 3 81235    628.53    8    3
#> 4 71769    431.87   11    3
#> 5 55080    620.88   12    3
#> 6 93224    501.76   12    5
```

The objective of this analytical procedure is to forecast potential misstatements within the population of the current year. In order for the information obtained through this procedure to serve as prior knowledge in a Bayesian analysis, the procedure must yield a distribution of the probability of misstatement. Therefore, the auditor employs a machine learning technique known as Random Forest (Hastie et al., 2009) to learn the relationship between misstatements, the number of full-time equivalent employees, and the number of outstanding days in the historical data set.

```
set.seed(1)
fit <- randomForest::randomForest(formula = k ~ ftes + days, data =
 ↪  history)
```

The auditor specifically uses the random forest technique due to its ability to provide a distribution of the misstatement probabilities. The probabilistic predictions for the unseen misstatements in the `population` data can be obtained by calling the `predict()` function with the argument `type = "prob"`.

```
predictions <- predict(object = fit, newdata = population, type =
 ↪  "prob")
```

These predictions come in a probabilistic format, which means that for each item in the population of the current year there is a predicted probability that that item is misstated. These probabilities are stored in the second column of the `predictions` data frame.

```
head(predictions)
#>       0     1
#> 1 0.974 0.026
#> 2 0.966 0.034
#> 3 0.992 0.008
#> 4 0.928 0.072
#> 5 0.776 0.224
#> 6 0.982 0.018
```

The prior distribution for $\theta$ will be based on the distribution of probabilities that each item in the population is misstated. In contrast to the previous example, this distribution is not a parametric distribution, which means we are unable to utilize any parametric priors from **jfa**. However, by providing samples of the prior distribution, **jfa** is able to construct a nonparametric prior distribution internally via the density of the samples.

The nonparametric prior distribution can be constructed through a call to `auditPrior()`, where the method to construct of the prior is specified as `nonparam`. The samples of the prior distribution can be provided through the `samples` argument. We use the second column of the the `predictions` object for this.

```
prior <- auditPrior(method = "nonparam", samples = predictions[, 2])
```

The nonparametric prior distribution can be visualized using the `plot()` function, see Figure 4.22.

```
plot(prior)
```

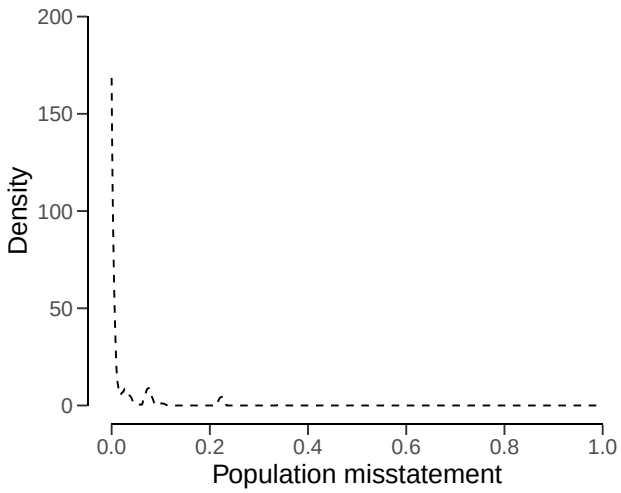



**Figure 4.22.** The prior distribution constructed on the basis of the predictive model.

The performance materiality for this example is set to $\theta_{max} = 0.01$, or one percent,

and the audit risk is set to $\alpha = 0.05$, or five percent. The minimum sample size can be calculated with the command below and is $n = 108$.

```
plan <- planning(materiality = 0.01, likelihood = "binomial", prior =
 ↪  prior)
plan
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 108
#> sample size obtained in 109 iterations via method 'binomial' +
 ↪  'prior'
```

You can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function, see Figure 4.23.

```
plot(plan)
```

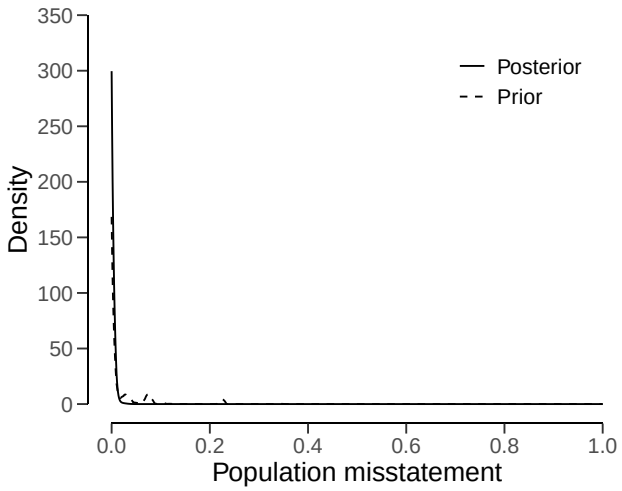



**Figure 4.23.** The prior and expected posterior distribution for this example after seeing a sample of $n = 108$ units containing no misstatements.

By using a frequentist approach, the required minimum sample size is $n = 299$. Thus, by performing the analytical procedure and incorporating this information into the prior distribution, the auditor has achieved a reduction in sample size of 299 - 108 = 191 items.

```
plan <- planning(materiality = 0.01, likelihood = "binomial")
plan
#>
#>  Classical Audit Sample Planning
#>
```

```
#> minimum sample size = 299
#> sample size obtained in 300 iterations via method 'binomial'
```

## 4.8 Practical Exercises

1. Use the classical approach with the hypergeometric likelihood to compute the minimum required sample size for a population of $N = 100$ when applying a performance materiality of three percent and a sampling risk of five percent. Tolerate no misstatements in the sample.
2. Recompute the previous sample size with the Bayesian approach using a default prior.
3. Use the classical approach with the binomial likelihood to compute the minimum required sample size for a performance materiality of 4.4 percent and a sampling risk of five percent. Tolerate one misstatement in the sample.
4. Recompute the previous sample size with the Bayesian approach using a default prior.
5. Use the classical approach with the Poisson likelihood to compute the minimum required sample size for a performance materiality of two percent and a sampling risk of five percent. Use an expected misstatement rate of 0.5 percent.
6. Recompute the previous sample size with the Bayesian approach using a default prior.
7. Compute the sampling risk according to the Audit Risk Model for an audit risk percentage of five percent, an inherent risk percentage of 50 percent and an internal control risk percentage of 80 percent. Next, use the classical approach with the binomial likelihood to compute the minimum sample size for a performance materiality of five percent and the new sampling risk. Tolerate no misstatements in the sample.
8. Construct a prior distribution on the basis of the information in exercise 7. Use this prior distribution to recompute the previous sample size with the Bayesian approach.
9. Construct a prior distribution on the basis of the assumption that tolerable misstatement is equally likely as intolerable misstatement before seeing the sample data. Use the binomial likelihood and assume a performance materiality of 2.5 percent.

## 4.9 Answers to the Exercises

1. This sample size can be computed using the `planning()` function with the arguments `likelihood = "hypergeometric"` and `N.units = 100`.

```
planning(materiality = 0.03, likelihood = "hypergeometric", N.units =
 ↪  100)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 63
#> sample size obtained in 64 iterations via method 'hypergeometric'
```

2. The previous sample size can be recomputed using a default prior by adding `prior = TRUE` to the call.

```
planning(materiality = 0.03, likelihood = "hypergeometric", N.units =
 ↪  100, prior = TRUE)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 63
#> sample size obtained in 64 iterations via method 'hypergeometric' +
 ↪  'prior'
```

3. This sample size can be computed using the `planning()` function with the arguments `likelihood = "binomial"` and `expected = 1`.

```
planning(materiality = 0.044, likelihood = "binomial", expected = 1)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 106
#> sample size obtained in 105 iterations via method 'binomial'
```

4. The previous sample size can be recomputed using a default prior by adding `prior = TRUE` to the call.

```
planning(materiality = 0.044, likelihood = "binomial", expected = 1,
 ↪  prior = TRUE)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 105
#> sample size obtained in 104 iterations via method 'binomial' +
 ↪  'prior'
```

5. This sample size can be computed using the `planning()` function with the argument `expected = 0.005`.

```
planning(materiality = 0.02, expected = 0.005)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 262
#> sample size obtained in 263 iterations via method 'poisson'
```

6. The previous sample size can be recomputed using a default prior by adding `prior = TRUE` to the call.

```
planning(materiality = 0.02, expected = 0.005, prior = TRUE)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 261
#> sample size obtained in 262 iterations via method 'poisson' +
 ↪  'prior'
```

7. This sample size can be computed by using an adjusted confidence level via the `conf.level` argument.

```
dr <- 0.05 / (0.5 * 0.8)
planning(materiality = 0.05, conf.level = 1 - dr)
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 42
#> sample size obtained in 43 iterations via method 'poisson'
```

8. The information in exercise 7 can be incorporated into the prior distribution by using the `auditPrior()` function in combination with `method = "arm"`.

```
prior <- auditPrior(method = "arm", likelihood = "binomial",
 ↪  materiality = 0.05, ir = 0.5, cr = 0.8)
planning(materiality = 0.05, prior = prior)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 41
#> sample size obtained in 42 iterations via method 'binomial' +
 ↪  'prior'
```

9. This prior distribution can be constructed using the `auditPrior()` function in combination with `method = "impartial"`, while providing a value for the `materiality`.

```
auditPrior(method = "impartial", likelihood = "binomial", materiality
 ↪  = 0.05)
#>
```

```
#>  Prior Distribution for Audit Sampling
#>
#> functional form: beta(α = 1, β = 13.513)
#> parameters obtained via method 'impartial'
```

# Chapter 5

# Selection

Auditors must often evaluate balances or populations that include a large quantity of items. As it is not possible to individually examine all of these items, they must select a subset, or sample, from the total population to make a statement about a specific characteristic of the population. Several selection methodologies, which are widely accepted in the audit context, are available for this purpose. This chapter discusses the most frequently used sampling methodology for audit sampling and demonstrates how to select a sample using these methods in R.

## 5.1 Sampling Units

Selecting a subset from the population requires knowledge of the sampling units; physical representations of the population that needs to be audited. Generally, the auditor has to choose between two types of sampling units: individual items in the population or individual monetary units in the population. In order to perform statistical selection, the population must be divided into individual sampling units that can be assigned a probability to be included in the sample. The total collection of all sampling units which have been assigned a selection probability is called the sampling frame.

### 5.1.1 Items

A sampling unit for record (i.e., attributes) sampling is generally a characteristic of an item in the population. For example, suppose that you inspect a population of receipts. A possible sampling unit for record sampling can be the date of payment of the receipt. When a sampling unit (e.g., date of payment) is selected by the sampling method, the population item that corresponds to the sampled unit is included in the sample.

### 5.1.2 Monetary Units

A sampling unit for monetary unit sampling is different than a sampling unit for record sampling in that it is an individual monetary unit within an item or transaction, like an individual dollar. For example, a single sampling unit can be the $10^{\text{th}}$ dollar from a specific receipt in the population. When a sampling unit (e.g., individual dollar) is selected by the sampling method, the population item that includes the sampling unit is included in the sample.

## 5.2 Sampling Methods

This section discusses four sampling methods that are commonly used in audit sampling. The methods that will be discussed are:

- Random sampling
- Fixed interval sampling
- Cell sampling
- Modified sieve sampling

First, let's get some notation out of the way. As discussed in Chapter 2, the population size $N$ is defined as the total set of individual sampling units (denoted by $x_i$).

$$N = \{x_1, x_2, ..., x_N\}. \tag{5.1}$$

In statistical sampling, every sampling unit $x_i$ in the population should receive a selection probability $p(x_i)$. The purpose of the sampling method is to provide a framework to assign selection probabilities to each of the sampling units, and subsequently draw sampling units from the population until a set of size $n$ has been created.

To illustrate how the resulting sample differs for various sampling methods, we will use the `BuildIt` data set included in the **jfa** package. These data can be loaded into R using the code below. For simplicity, we will use a sample size of $n = 10$ for all examples.

```
data(BuildIt)
n <- 10
```

### 5.2.1 Random Sampling

Random sampling is the most simple and straight-forward selection method. The random sampling method provides a method that allows every sampling unit in the population an equal chance of being selected, meaning that every combination of sampling units has the same probability of being selected as every other combination of the same number of sampling units. Simply put, the algorithm draws a random selection of size $n$ of the sampling units. Therefore, the selection probability for each sampling unit is defined as:

$$p(x) = \frac{1}{N}. \tag{5.2}$$

To make this procedure visually intuitive, Figure 5.1 below provides an illustration of the random sampling method.

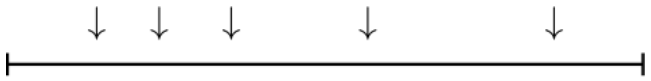


**Figure 5.1.** Illustration of random sampling, which involves selecting a subset of items from a population in such a way that every sampling unit in the population has an equal chance of being included in the sample.

- **Advantage(s):** The random sampling method yields an optimal random selection, with the additional advantage that the sample can be easily extended by applying the same method again.
- **Disadvantages:** Because the selection probabilities are equal for all sampling units there is no guarantee that items with a large monetary value in the population will be included in the sample.

#### 5.2.1.1 Record Sampling

Random sampling can easily be coded in base R. First, we have to get a vector of of the possible items (rows) in the population that can be selected. When we are performing record sampling, we can simply use R's build in `sample()` function to draw a random sample from a vector `1:nrow(BuildIt)` representing the row indices of the items and store the result in a variable `items`.

```
set.seed(1)
items <- sample.int(nrow(BuildIt), size = n, replace = FALSE)
items
#>  [1] 1017  679 2177  930 1533  471 2347  270 1211 3379
```

You can then select the sample from the population using the selected indices stored in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 1017 50755    618.24     618.24
#> 679  20237    669.75     669.75
#> 2177  9517    454.02     454.02
#> 930  85674    257.82     257.82
#> 1533 31051    308.53     308.53
#> 471  84375    824.66     824.66
#> 2347 75616    623.70     623.70
#> 270  82033    352.75     352.75
#> 1211 12877     52.89      52.89
#> 3379 85322    330.24     330.24
```

The sample can be reproduced in **jfa** via the `selection()` function. This function takes as input the population data, the sample size, and the characteristics of the sampling method. The argument `units` allows you to specify that you want to use record

sampling (`units = "items"`), while the `method` argument enables you to specify that you are performing random sampling (`method = 'random'`).

```
set.seed(1)
selection(BuildIt, size = n, units = "items", method =
 ↪  "random")$sample
#>     row times    ID bookValue auditValue
#> 1  1017     1 50755    618.24     618.24
#> 2   679     1 20237    669.75     669.75
#> 3  2177     1  9517    454.02     454.02
#> 4   930     1 85674    257.82     257.82
#> 5  1533     1 31051    308.53     308.53
#> 6   471     1 84375    824.66     824.66
#> 7  2347     1 75616    623.70     623.70
#> 8   270     1 82033    352.75     352.75
#> 9  1211     1 12877     52.89      52.89
#> 10 3379     1 85322    330.24     330.24
```

An alternative to specifying the desired sample size through the `size` argument is to provide an object generated by the `planning()` function to the `selection()` function. For instance, the following code utilizes the `planning()` function to plan a sample size based on a performance materiality of 0.03, or three percent, and a sampling risk of 0.05, or five percent, which can be passed directly to `selection()` to select the sample from the `BuildIt` population.

```
selection(BuildIt, size = planning(materiality = 0.03), units =
 ↪  "items", method = "random")
#>
#>  Audit Sample Selection
#>
#> data:  BuildIt
#> number of sampling units = 100, number of items = 100
#> sample selected via method 'items' + 'random'
```

The ability of one function to accept input from another function allows for the implementation of a workflow in which the `planning()` function and the `selection()` function are sequentially linked. Additionally, the use of R's native pipe operator `|>` further simplifies this process.

```
planning(materiality = 0.03) |>
  selection(data = BuildIt, units = "items", method = "random")
#>
#>  Audit Sample Selection
#>
#> data:  BuildIt
#> number of sampling units = 100, number of items = 100
#> sample selected via method 'items' + 'random'
```

The `selection()` function has three additional arguments which you can use to

preprocess your population before selection. These arguments are `order`, `decreasing` and `randomize`.

The `order` argument takes as input a column name in `data` which determines the order of the population. For example, you can order the population from lowest book value to highest book value before engaging in the selection. In this case, you should use the `decreasing = FALSE` (its default value) argument.

```
set.seed(1)
selection(BuildIt, size = n, order = "bookValue", units = "items",
 ↪  method = "random")$sample
#>     row times    ID bookValue auditValue
#> 1    29     1 58849    274.26     274.26
#> 2  3297     1 24279    229.95     229.95
#> 3   931     1 28025    429.14     429.14
#> 4   756     1 11563    263.08     105.23
#> 5  3375     1 58981    335.39     335.39
#> 6  1624     1 97783    197.19     197.19
#> 7  2534     1 95715    457.42     457.42
#> 8  1798     1 95520    157.54     157.54
#> 9  3448     1 12959    296.82     296.82
#> 10  450     1 39908    831.31     831.31
```

The `randomize` argument can be used to randomly shuffle the items in the population before selection. For example, you can randomly shuffle the population before engaging in the selection using `randomize = TRUE`.

```
set.seed(1)
selection(BuildIt, size = n, randomize = TRUE, units = "items", method
 ↪  = "random")$sample
#>     row times    ID bookValue auditValue
#> 1  1264     1 85424    406.81     406.81
#> 2   923     1 12566    287.61     287.61
#> 3   776     1 92923    247.89     247.89
#> 4   127     1 71325    306.78     306.78
#> 5  1611     1 10019    191.18     191.18
#> 6  3087     1 87887    666.13     666.13
#> 7  1729     1 78779    608.02     608.02
#> 8  2037     1 74155    347.18     347.18
#> 9  2769     1 26010    240.10     240.10
#> 10 2276     1 80154    282.91     282.91
```

#### 5.2.1.2 Monetary Unit Sampling

When we are performing record sampling, we have to consider that each item in the population consists of multiple smaller items (i.e., the monetary units), which means that items with a higher book value should get a higher probability of being selected. The `sample()` function faciliates weighted selection via the `prob` argument, which takes a vector of values and, using normalization, computes the weights for

selection. The call below is similar to before, but in this case we use the book values in the column `bookValues` of the data set to weigh the items and store the result in a variable `items`.

```
set.seed(1)
items <- sample.int(nrow(BuildIt), size = n, replace = TRUE, prob =
 ↪  BuildIt[["bookValue"]])
items
#>   [1] 1017  679 2856  471  270 2642  597 2208  330 1615
```

You can then select the sample from the population using the selected indices stored in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 1017 50755    618.24     618.24
#> 679  20237    669.75     669.75
#> 2856 21836    820.86     820.86
#> 471  84375    824.66     824.66
#> 270  82033    352.75     352.75
#> 2642 49666    601.71     601.71
#> 597  93226    376.55     376.55
#> 2208 61540    963.53     963.53
#> 330  30774    508.26     508.26
#> 1615 42859    598.07     598.07
```

The sample can be reproduced in **jfa** via the `selection()` function. The argument `units` allows you to specify that you want to use monetary unit sampling (`units = "values"`), while the `method` argument enables you to specify that you are performing random sampling (`method = 'random'`). Note that you should provide the name of the column in the data that contains the monetary units via the `values` argument.

```
set.seed(1)
selection(BuildIt, size = n, units = "values", method = "random",
 ↪  values = "bookValue")$sample
#>     row times    ID bookValue auditValue
#> 1  1017     1 50755    618.24     618.24
#> 2   679     1 20237    669.75     669.75
#> 3  2856     1 21836    820.86     820.86
#> 4   471     1 84375    824.66     824.66
#> 5   270     1 82033    352.75     352.75
#> 6  2642     1 49666    601.71     601.71
#> 7   597     1 93226    376.55     376.55
#> 8  2208     1 61540    963.53     963.53
#> 9   330     1 30774    508.26     508.26
#> 10 1615     1 42859    598.07     598.07
```

### 5.2.2 Fixed Interval Sampling

Fixed interval sampling is a method designed for yielding representative samples from monetary populations. The algorithm determines a uniform interval on the (optionally ranked) sampling units. Next, a starting point is handpicked or randomly selected in the first interval and a sampling unit is selected throughout the population at each of the uniform intervals from the starting point. For example, if the interval has a width of 10 sampling units and sampling unit number 5 is chosen as the starting point, the sampling units 5, 15, 25, etc. are selected to be included in the sample.

The number of required intervals $I$ can be determined by dividing the number of sampling units in the population by the required sample size:

$$I = \frac{N}{n}, \tag{5.3}$$

in which $n$ is the required sample size and $N$ is the total number of sampling units in the population.

If the space between the selected sampling units is equal, the selection probability for each sampling unit is theoretically defined as:

$$p(x) = \frac{1}{I}, \tag{5.4}$$

with the property that the space between selected units $i$ (of which the first one is the starting point) is the same as the interval $I$, see @#fig-selection-interval below. However, in practice the selection is deterministic and completely depends on the chosen starting points (using `start`).

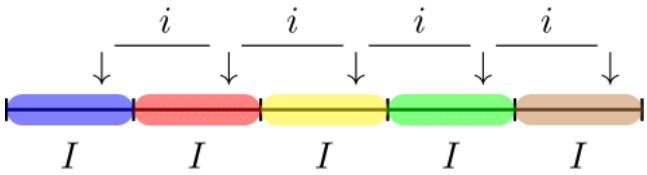



**Figure 5.2.** Illustration of fixed interval sampling. The population is represented by the horizontal line, and the vertical lines indicate the intervals of size $I$ at which samples units are selected. By using fixed interval sampling, equal spacing between sampling units is ensures, which means that every $i^{th}$ unit in the population is included in the sample.

The fixed interval method yields a sample that allows every sampling unit in the population an equal chance of being selected. However, the fixed interval method has the property that all items in the population with a monetary value larger than the interval $I$ have an selection probability of one because one of these items' sampling units are always selected from the interval. Note that, if the population is arranged randomly with respect to its deviation pattern, fixed interval sampling is equivalent to random selection.

- **Advantage(s):** The advantage of the fixed interval sampling method is that it is often simple to understand and fast to perform. Another advantage is that, in monetary unit sampling, all items that are greater than the calculated interval will be included in the sample. In record sampling, since units can be ranked on the basis of value, there is also a guarantee that some large items will be in the sample.
- **Disadvantage(s):** A pattern in the population can coincide with the selected interval, rendering the sample less representative. What is sometimes seen as an added complication for this method is that the sample is hard to extend after drawing the initial sample. This is due to the chance of selecting the same sampling unit. However, by removing the already selected sampling units from the population and redrawing the intervals this problem can be efficiently solved.

#### 5.2.2.1 Record Sampling

To code fixed interval sampling in a record sampling context, we first have to compute the size of the interval we are working with. This is computed by dividing the number of items in the population by the desired sample size $n$. Suppose the auditor wants to select a sample of 10 items, then the interval is computed by:

```
interval <- nrow(BuildIt) / n
```

Next, we have to determine the starting point. We are going to take the fifth unit in each interval in this case.

```
start <- 5
```

To find which rows are part of the sample, we execute the following code:

```
items <- ceiling(start + interval * 0:(n - 1))
```

You can then select the sample from the population using the selected indices stored in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 5    55080    620.88     620.88
#> 355  27934    749.38     749.38
#> 705  21900    919.00     919.00
#> 1055 66675    384.27     384.27
#> 1405 13472    360.05     360.05
#> 1755 61607    389.75     389.75
#> 2105 68519    354.71     354.71
#> 2455 91983    467.72     467.72
#> 2805 25646    420.80     420.80
#> 3155 94955    248.77     248.77
```

The sample can be reproduced in **jfa** via the `selection()` function. The argument `units` allows you to specify that you want to use record sampling (`units = "items"`),

while the `method` argument enables you to specify that you are performing fixed interval sampling (`method = 'interval'`). Note that, by default, the first sampling unit from each interval is selected. However, this can be changed by setting the argument `start` to a different value.

```
selection(BuildIt, size = n, units = "items", method = "interval",
 ↪  start = start)$sample
#>     row times    ID bookValue auditValue
#> 1     5     1 55080    620.88     620.88
#> 2   355     1 27934    749.38     749.38
#> 3   705     1 21900    919.00     919.00
#> 4  1055     1 66675    384.27     384.27
#> 5  1405     1 13472    360.05     360.05
#> 6  1755     1 61607    389.75     389.75
#> 7  2105     1 68519    354.71     354.71
#> 8  2455     1 91983    467.72     467.72
#> 9  2805     1 25646    420.80     420.80
#> 10 3155     1 94955    248.77     248.77
```

#### 5.2.2.2 Monetary Unit Sampling

In monetary unit sampling, the only difference is that we are computing the interval on the basis of the booked values in the column `bookValue` of the data set. In this case, the starting point `start = 5` determines which monetary unit from each interval is selected.

```
interval <- sum(BuildIt[["bookValue"]]) / n
```

To find which units are part of the sample, we execute the following code:

```
units <- start + interval * 0:(n - 1)
```

To obtain which items are part of the sample, we can run the following for loop. Note that this does not take into account whether the book values contain negative values, which should not be included in the cumulative sum below.

```
all_units <- ifelse(BuildIt[["bookValue"]] < 0, 0,
 ↪  BuildIt[["bookValue"]])
all_items <- 1:nrow(BuildIt)
items <- numeric(n)
for (i in 1:n) {
  item <- which(units[i] <= cumsum(all_units))[1]
  items[i] <- all_items[item]
}
```

You can then select the sample from the population using the selected indices stored in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
```

```
#> 1    82884    242.61     242.61
#> 358  20711    610.88     610.88
#> 715  99012    313.75     313.75
#> 1081 65319    502.54     201.02
#> 1421 88454    856.28     856.28
#> 1774 87258    157.68     157.68
#> 2103 48652    497.21     497.21
#> 2435 37248   1041.44    1041.44
#> 2787 10925    377.10     377.10
#> 3152 71832   1001.82    1001.82
```

The sample can be reproduced in **jfa** via the `selection()` function. The argument `units` allows you to specify that you want to use monetary unit sampling (`units = "values"`), while the `method` argument enables you to specify that you are performing fixed interval sampling (`method = 'interval'`). Note that you should provide the name of the column in the data that contains the monetary units via the `values` argument.

```
selection(BuildIt, size = n, units = "values", method = "interval",
 ↪  values = "bookValue", start = start)$sample
#>     row times    ID bookValue auditValue
#> 1     1     1 82884    242.61     242.61
#> 2   358     1 20711    610.88     610.88
#> 3   715     1 99012    313.75     313.75
#> 4  1081     1 65319    502.54     201.02
#> 5  1421     1 88454    856.28     856.28
#> 6  1774     1 87258    157.68     157.68
#> 7  2103     1 48652    497.21     497.21
#> 8  2435     1 37248   1041.44    1041.44
#> 9  2787     1 10925    377.10     377.10
#> 10 3152     1 71832   1001.82    1001.82
```

### 5.2.3 Cell Sampling

The cell sampling method divides the (optionally ranked) population into a set of intervals $I$ that are computed through the previously given equations. Within each interval, a sampling unit is selected by randomly drawing a number between 1 and the interval range $I$. This causes the space $i$ between the sampling units to vary. The procedure is displayed in @#fig-selection-cell.

Like in the fixed interval sampling method, the selection probability for each sampling unit is defined as:

$$p(x) = \frac{1}{I}. \tag{5.5}$$

The cell sampling method has the property that all items in the population with a monetary value larger than twice the interval $I$ have a selection probability of one.

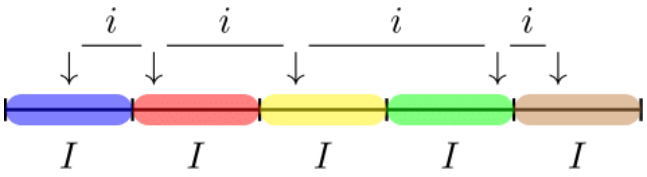



**Figure 5.3.** Illustration of cell sampling. In this illustration, the population is fist divided into distinct cells of size $I$ and subsequently a random sampling unit is selected within each cell such that the space between units $i$ varies.

- **Advantage(s):** More sets of samples are possible than in fixed interval sampling, as there is no systematic interval $i$ to determine the selections. It is argued that the cell sampling algorithm offers a solution to the pattern problem in fixed interval sampling.
- **Disadvantage(s):** A disadvantage of this sampling method is that not all items in the population with a monetary value larger than the interval have a selection probability of one. Besides, population items can be in two adjacent cells, thereby creating the possibility that an items is included in the sample twice.

#### 5.2.3.1 Record Sampling

To code cell sampling in a record sampling context, we again have to compute the size of the interval we are working with:

```
interval <- nrow(BuildIt) / n
```

Next, we have to randomly determine which items are going to be selected in each interval.

```
set.seed(1)
starts <- floor(runif(n, 0, interval))
```

To find which rows are part of the sample, we execute the following code:

```
items <- floor(starts + interval * 0:(n - 1))
```

You can then select the sample from the population using the selected indices stored in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 92   75133    355.16     355.16
#> 480  81037    456.27     456.27
#> 900   1730    449.87     449.87
#> 1367 36587    282.32     282.32
#> 1470 10305    648.70     648.70
#> 2064 96344    268.94     268.94
#> 2430 60885    493.77     493.77
```

```
#> 2681 60935    312.98     312.98
#> 3020  8716    450.76     450.76
#> 3171 61036    387.67     387.67
```

The sample can be reproduced in **jfa** via the `selection()` function. The argument `units` allows you to specify that you want to use record sampling (`units = "items"`), while the `method` argument enables you to specify that you are performing cell sampling (`method = 'cell'`).

```
set.seed(1)
selection(BuildIt, size = n, units = "items", method = "cell")$sample
#>     row times    ID bookValue auditValue
#> 1    92     1 75133    355.16     355.16
#> 2   480     1 81037    456.27     456.27
#> 3   900     1  1730    449.87     449.87
#> 4  1367     1 36587    282.32     282.32
#> 5  1470     1 10305    648.70     648.70
#> 6  2064     1 96344    268.94     268.94
#> 7  2430     1 60885    493.77     493.77
#> 8  2681     1 60935    312.98     312.98
#> 9  3020     1  8716    450.76     450.76
#> 10 3171     1 61036    387.67     387.67
```

#### 5.2.3.2 Monetary Unit Sampling

In monetary unit sampling, the only difference is that we are computing the interval on the basis of the booked values in the column `bookValue` of the data set. In this case, the starting points `start` determines which monetary unit from each interval is selected.

```
interval <- sum(BuildIt[["bookValue"]]) / n
```

To obtain which items are part of the sample, we can run the following for loop. Note that this does not take into account whether the book values contain negative values, which should not be included in the cumulative sum below.

```
set.seed(1)
all_units <- ifelse(BuildIt[["bookValue"]] < 0, 0,
 ↪  BuildIt[["bookValue"]])
all_items <- 1:nrow(BuildIt)
intervals <- 0:n * interval
items <- numeric(n)
for (i in 1:n) {
  unit <- stats::runif(1, intervals[i], intervals[i + 1])
  item <- which(unit <= cumsum(all_units))[1]
  items[i] <- all_items[item]
}
```

You can then select the sample from the population using the selected indices stored

in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 95   15009    415.60     415.60
#> 486  79093    635.85     635.85
#> 931  28025    429.14     429.14
#> 1387 56444    296.37     296.37
#> 1492 81443    543.80     543.80
#> 2074 14196    270.45     270.45
#> 2418 87743    347.99     347.99
#> 2660 23927    454.81     454.81
#> 3024 78925    251.44     251.44
#> 3172 18286    450.57     450.57
```

The sample can be reproduced in **jfa** via the `selection()` function. The argument `units` allows you to specify that you want to use monetary unit sampling (`units = "values"`), while the `method` argument enables you to specify that you are performing cell sampling (`method = 'cell'`). Note that you should provide the name of the column in the data that contains the monetary units via the `values` argument.

```
set.seed(1)
selection(BuildIt, size = n, units = "values", method = "cell", values
 ↪  = "bookValue")$sample
#>     row times    ID bookValue auditValue
#> 1    95     1 15009    415.60     415.60
#> 2   486     1 79093    635.85     635.85
#> 3   931     1 28025    429.14     429.14
#> 4  1387     1 56444    296.37     296.37
#> 5  1492     1 81443    543.80     543.80
#> 6  2074     1 14196    270.45     270.45
#> 7  2418     1 87743    347.99     347.99
#> 8  2660     1 23927    454.81     454.81
#> 9  3024     1 78925    251.44     251.44
#> 10 3172     1 18286    450.57     450.57
```

### 5.2.4 Modified Sieve Sampling

The fourth option for the sampling method is modified sieve sampling (Hoogduin, Hall, & Tsay, 2010). The algorithm starts by selecting a standard uniform random number $R_i$ between 0 and 1 for each item in the population. Next, the sieve ratio:

$$S_i = \frac{Y_i}{R_i} \tag{5.6}$$

is computed for each item by dividing the book value of that item by the random number. Lastly, the items in the population are sorted by their sieve ratio $S$ (in decreasing order) and the top $n$ items are selected for inspection. In contrast to the

classical sieve sampling method (Rietveld, 1978), the modified sieve sampling method provides precise control over sample sizes.

#### 5.2.4.1 Monetary Unit Sampling

```
set.seed(1)
all_units <- ifelse(BuildIt[["bookValue"]] < 0, 0,
 ↪  BuildIt[["bookValue"]])
all_items <- 1:nrow(BuildIt)
ri <- all_units / stats::runif(length(all_items), 0, 1)
items <- all_items[order(-ri)]
items <- items[1:n]
```

You can then select the sample from the population using the selected indices stored in `items`.

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 2329 29919    681.10     681.10
#> 2883 59402    279.29     279.29
#> 1949 56012    581.22     581.22
#> 3065 47482    621.73     621.73
#> 1072 79901    789.97     789.97
#> 488  50811    651.35     651.35
#> 1916 53565    266.37     266.37
#> 463  65768    480.89     480.89
#> 1311 91955    398.96     398.96
#> 2895  8688    492.02     492.02
```

The sample can be reproduced in **jfa** via the `selection()` function. The argument `units` allows you to specify that you want to use monetary unit sampling (`units = "values"`), while the `method` argument enables you to specify that you are performing modified sieve sampling (`method = 'sieve'`). Note that you should provide the name of the column in the data that contains the monetary units via the `values` argument.

```
set.seed(1)
selection(BuildIt, size = n, units = "values", method = "sieve",
 ↪  values = "bookValue")$sample
#>     row times    ID bookValue auditValue
#> 1  2329     1 29919    681.10     681.10
#> 2  2883     1 59402    279.29     279.29
#> 3  1949     1 56012    581.22     581.22
#> 4  3065     1 47482    621.73     621.73
#> 5  1072     1 79901    789.97     789.97
#> 6   488     1 50811    651.35     651.35
#> 7  1916     1 53565    266.37     266.37
#> 8   463     1 65768    480.89     480.89
#> 9  1311     1 91955    398.96     398.96
```

```
#> 10 2895     1  8688    492.02     492.02
```

## 5.3 Practical Exercises

1. Select a random sample of 120 items from the `BuildIt` data set.
2. Select a sample of 240 monetary units from the `BuildIt` data set using a fixed interval selection method. Use a starting point of 12.

## 5.4 Answers to the Exercises

1. Selecting a random sample of 120 items can be done using the `selection()` function with the additional arguments `size = 120`, `method = "random"` and `units = "items"`.

```
selec <- selection(data = BuildIt, size = 120, method = "random",
 ↪  units = "items")
head(selec[["sample"]], 5)
#>    row times    ID bookValue auditValue
#> 1 1827     1 38946    517.88     517.88
#> 2 1372     1 59757    177.25     177.25
#> 3  940     1  9621    429.21     429.21
#> 4  949     1 48821    363.40     363.40
#> 5 2307     1 95785    244.49     244.49
```

2. Selecting a random sample of 240 monetary units can be done using the `selection()` function with the arguments `size = 240`, `method = "interval"` and `units = "values"`. Additionally, for fixed interval monetary unit sampling, the book values must be given in via argument `values = "bookValue"`. The starting point is indicated using `start = 12`.

```
selec <- selection(data = BuildIt, size = 240, method = "interval",
 ↪  units = "values", values = "bookValue", start = 12)
head(selec[["sample"]], 5)
#>   row times    ID bookValue auditValue
#> 1   1     1 82884    242.61     242.61
#> 2  15     1 76073    469.93     469.93
#> 3  31     1 83557    507.34     507.34
#> 4  47     1 53784    325.19     325.19
#> 5  63     1 51272    248.40     248.40
```

# Chapter 6

# Evaluation

In statistical audit sampling, the evaluation of the sample is the crucial last step in the process. Here, the auditor assesses the audit evidence collected from the sample and blends it with the evidence gathered in earlier stages of the audit. The outcome of this evaluation forms the auditor's overall opinion of the population being audited.

Just as with planning a sample, evaluating a sample requires knowledge of the circumstances that determine whether the population should be accepted or rejected, which are referred to as sampling objectives. Sampling objectives can be divided into two main categories:

- **Hypothesis testing**: The goal of the sample is to obtain evidence for or against the claim that the misstatement in the population is lower than a given value (i.e., the performance materiality).
- **Estimation**: The goal of the sample is to obtain an accurate estimate of the misstatement in the population with a certain precision.

When an auditor needs to perform audit sampling, they typically have two options: non-stratified or stratified sampling. Non-stratified sampling involves selecting a sample of units without considering any categorical characteristics, such as product type or store location, of these items. Stratified sampling involves dividing the population into subgroups based on specific characteristics and selecting a sample from each subgroup. In comparison to non-stratified sampling, stratified sampling can improve efficiency. However, in this chapter, we will focus on non-stratified audit sampling, while stratified audit sampling will be covered in the next chapter.

Non-stratified sampling is typically used when the population is considered homogenous, meaning there are no significant differences between subgroups. This approach is also suitable when the auditor does not need to consider differences between subgroups. For example, when an auditor reviews a company's inventory using non-stratified sampling, they may choose a random sample of items from the entire inventory without dividing it into subgroups. Similarly, when conducting an audit of a small business's general ledger, an auditor may select a sample of entries without dividing them based on categorical features such as payment method.

**Figure 6.1.** When evaluating a sample with respect to a specific hypothesis, the auditor must consider the evidence in favour as well as the evidence against that hypothesis. Image available under a CC-BY-NC 4.0 license.

To get started with non-stratified evaluation of audit samples, the auditor must first decide if they want to evaluate based on summary statistics derived from a sample (i.e., 1 misstatement in 100 items) or if they want to calculate the misstatements from a data set. To make the principle of evaluation easier to understand, we will first explain how to evaluate a sample using summary statistics from a sample. Next, we will explain how to evaluate a sample using the raw data.

## 6.1 Binary Misstatements

Complete misstatements arise when there is an absolute discrepancy between the actual value of an item in the sample and its recorded value. From a data perspective, this signifies that the misstatements in the sample are strictly binary (i.e., 0 or 1) and do not encompass values within the intermediary range of 0 to 1. For example, a receipt for goods received can be fully misstated if the goods have not been received at all.

### 6.1.1 Classical Evaluation

In classical evaluation, confidence intervals and $p$-values are used to measure the uncertainty and the evidence against the hypothesis of intolerable misstatement, respectively.

Confidence intervals play a crucial role in classical inference by helping to determine the uncertainty in a sample estimate. For example, if an auditor needs to estimate the misstatement in a tax return, they can calculate a confidence interval for the misstatement using classical inference. This confidence interval represents a range of possible values in which the true misstatement of the population is likely to fall. This range helps auditors make informed decisions about the misstatement and determine the potential impact of the misstatement on the (loss of) taxes.

To illustrate, suppose an auditor wants to estimate the misstatement in a population based on a sample of 100 items containing one misstatement. Using the `evaluation()` function in **jfa** and specifying `x = 1` and `n = 100`, the output shows that the estimated most likely misstatement in the population is one percent, and the 95 percent (one-sided) confidence interval ranges from 0 percent to 4.74 percent. It is important to note that the correct interpretation of a 95 percent confidence interval is: *"If we were to repeat the experiment over and over, then 95 percent of the time the confidence interval contains the true misstatement rate"* (Hoekstra et al., 2014).

```
evaluation(x = 1, n = 100, method = "binomial")
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  1 and 100
#> number of errors = 1, number of samples = 100, taint = 1
#> 95 percent confidence interval:
#>  0.00000000 0.04655981
#> most likely estimate:
#>  0.01
#> results obtained via method 'binomial'
```

Classical hypothesis testing relies on the $p$-value to determine whether to accept or reject a certain hypothesis about a population. For example, suppose an auditor wishes to test whether the population contains misstatements of less than three percent (they formulate the performance materiality based on existing rules and regulations). They would create the hypotheses $H_1$: $\theta < 0.03$ and $H_0$: $\theta \geq 0.03$. The significance level is set to 0.05, equivalent to an audit risk of 5 percent. This means that a $p$-value below 0.05 is sufficient to reject the hypothesis of intolerable misstatement $H_0$.

In **jfa**, a classical hypothesis test using the $p$-value can be conducted by specifying the `materiality` argument in the `evaluation()` function. For example, to indicate a performance materiality of three percent, the auditor can specify `materiality = 0.03`. Along with the confidence interval, the output displays a $p$-value of 0.19462, which is greater than 0.05. Therefore, the hypothesis $H_0$ cannot be rejected at a significance level of five percent. As a result, the auditor cannot conclude that the sample provides sufficient evidence to reduce the audit risk to an appropriate level and cannot state that the population does not have misstatements of three percent or more.

```
eval <- evaluation(materiality = 0.03, x = 1, n = 100, method =
 ↪  "binomial")
eval
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  1 and 100
#> number of errors = 1, number of samples = 100, taint = 1, p-value =
#> 0.19462
#> alternative hypothesis: true misstatement rate is less than 0.03
```

```
#> 95 percent confidence interval:
#>  0.00000000 0.04655981
#> most likely estimate:
#>  0.01
#> results obtained via method 'binomial'
```

The exact definition of the $p$-value is *"the probability of observing the data, or more extreme data, given the truth of the hypothesis of intolerable misstatement"*. The $p$-value of 0.19462 can be visualized via the `plot()` function, see Figure 6.2.

```
plot(eval, type = "posterior")
```

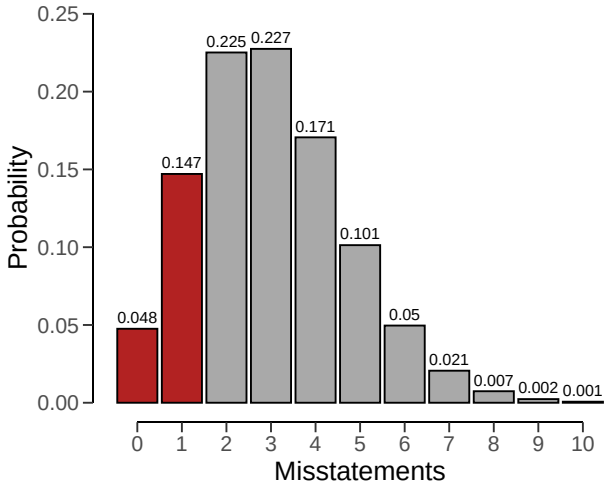



**Figure 6.2.** The $p$-value is the sum of the observed and more extreme (but unobserved) outcomes, which in this case is the sum of $k = 0$ and $k = 1$ and equals 0.048 + 0.147 = 0.195.

### 6.1.2 Bayesian Evaluation

In addition to classical evaluation methods, Bayesian inference offers an alternative approach to assessing audit samples. Unlike classical methods that use confidence intervals, Bayesian methods use credible intervals to measure the uncertainty in estimates.

Bayesian inference begins by specifying a prior distribution, which reflects prior knowledge about the misstatement in the population before any data is collected. This prior distribution is then combined with the information obtained from the sample to derive a posterior distribution. From the posterior distribution, credible intervals can be calculated to estimate the most likely misstatement in the population and the range of values within which the true value is likely to fall.

A Bayesian credible interval is intuitively interpreted as follows: *There is a 95 percent*

*probability that the misstatement falls within the credible interval.* This is in contrast to the interpretation of a classical confidence interval, which is often misinterpreted for its Bayesian counterpart.

For instance, consider a scenario where a uniform beta(1, 1) prior distribution is used, along with a sample of 100 units, one of which contains a misstatement. Using the posterior distribution, it can be estimated that the most likely misstatement in the population is 1 percent. Furthermore, a Bayesian credible interval can be calculated to show that there is a 95 percent probability that the true misstatement rate lies between 0 percent and 4.61 percent. The small difference between the classical and default Bayesian results arises from the use of the uniform beta(1, 1) prior distribution. To achieve classical results, we can create a prior with `method = "strict"` using the `auditPrior()` function. Remember that any call to `evaluation()` can be done in a Bayesian way by specifying a prior distribution. Therefore, the sole difference between the call for a classical analysis and the call for a Bayesian analysis is the use of the `prior` constructed through a call to `auditPrior()`.

```
prior <- auditPrior(method = "default", likelihood = "binomial")
eval <- evaluation(x = 1, n = 100, prior = prior)
eval
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  1 and 100
#> number of errors = 1, number of samples = 100, taint = 1
#> 95 percent credible interval:
#>  0.00000000 0.04610735
#> most likely estimate:
#>  0.01
#> results obtained via method 'binomial' + 'prior'
```

You can use the `plot()` function to visualize the posterior distribution along the most likely misstatement and the credible interval for the population misstatement. Figure 6.3 shows this posterior distribution.

```
plot(eval, type = "posterior")
```

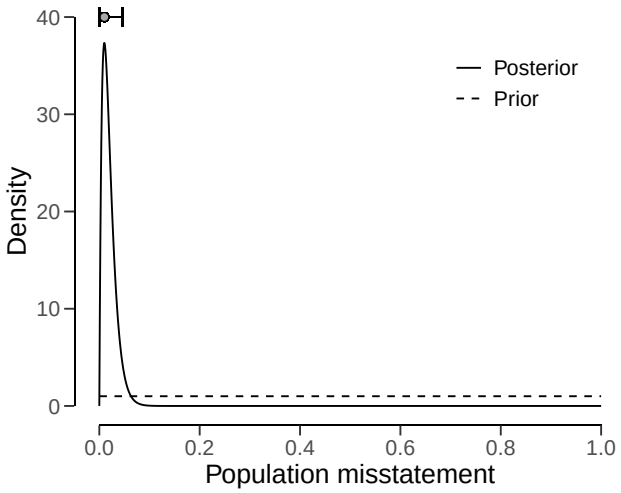



**Figure 6.3.** The posterior distribution showing the most likely misstatement in the population as a gray dot and the 95 percent credible interval for the population misstatement as the black bars.

Bayesian hypothesis testing also involves the use of evidence measures, but instead of $p$-values, Bayesian inference employs the Bayes factor, either $BF_{10}$ or $BF_{01}$, to arrive at conclusions regarding the evidence furnished by the sample in favor of one of two hypotheses, $H_1$ or $H_0$. The Bayes factor quantifies the strength of evidence in favor of one hypothesis over another.

The Bayes factor provides an intuitive measure of statistical evidence, allowing auditors to interpret the probability of the data occurring under either hypothesis. For instance, if the `evaluation()` function outputs a value of 10 for $BF_{10}$, it means that the data are ten times more likely to have arisen under $H_1$ than under $H_0$. A Bayes factor $BF_{10}$ greater than 1 suggests evidence for $H_1$ and against $H_0$, while a Bayes factor $BF_{10}$ less than 1 suggests evidence for $H_0$ and against $H_1$. Although the `evaluation()` function returns $BF_{10}$ by default, one can compute $BF_{01}$ as the inverse of $BF_{10}$ (i.e., $BF_{01} = \frac{1}{BF_{10}}$).

To illustrate, suppose an auditor wishes to verify whether a population contains less than three percent misstatement. Like before, this corresponds to the hypotheses $H_1$: $\theta < 0.03$ and $H_0$: $\theta \geq 0.03$. The auditor has taken a sample of $n = 100$ items, with only $k = 1$ item containing a misstatement. By assuming a default beta(1, 1) prior distribution, the following code evaluates the sample using a Bayesian hypothesis test and the Bayes factor. The `materiality = 0.03` argument specifies the materiality for this population.

```
evaluation(materiality = 0.03, x = 1, n = 100, prior = prior)
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  1 and 100
```

```
#> number of errors = 1, number of samples = 100, taint = 1, BF  =
#> 137.65
#> alternative hypothesis: true misstatement rate is less than 0.03
#> 95 percent credible interval:
#>  0.00000000 0.04610735
#> most likely estimate:
#>  0.01
#> results obtained via method 'binomial' + 'prior'
```

In this case, the Bayes factor is $BF_{10} = 137.65$, which means that the sample data is 137.65 times more likely to occur under the hypothesis of tolerable misstatement than the hypothesis of material misstatement. We arrived at this value by considering both the prior distribution and the posterior distribution. Specifically, we first used the beta(1, 1) prior distribution to calculate the prior probability of the hypothesis of tolerable misstatement.

```
prior.prob.h1 <- pbeta(0.03, shape1 = 1, shape2 = 1)
prior.prob.h1
#> [1] 0.03
```

The probability of the hypothesis of intolerable misstatement is essentially the opposite of the probability of the hypothesis of tolerable misstatement. To clarify, it is just one minus the prior probability of the hypothesis of tolerable misstatement.

```
prior.prob.h0 <- 1 - prior.prob.h1
prior.prob.h0
#> [1] 0.97
```

We use the prior probabilities to calculate the prior odds, which is the ratio of the prior probabilities.

```
prior.odds.h1 <- prior.prob.h1 / prior.prob.h0
prior.odds.h1
#> [1] 0.03092784
```

To compute the posterior probability of the hypothesis of tolerable misstatement, we can use the posterior distribution and essentially follow the same steps. Hence, we calculate the posterior probability for the hypothesis of tolerable misstatement, then obtain the posterior probability of the hypothesis of intolerable misstatement by subtracting this probability from one. Finally, the posterior odds are calculated as the ratio of the posterior probabilities

```
post.prob.h1 <- pbeta(0.03, shape1 = 1 + 1, shape2 = 1 + 100 - 1)
post.prob.h0 <- 1 - post.prob.h1
post.odds.h1 <- post.prob.h1 / post.prob.h0
post.odds.h1
#> [1] 4.257346
```

Finally, the Bayes factor can be computed as the ratio of the posterior odds and the

prior odds.

```
bf10 <- post.odds.h1 / prior.odds.h1
bf10
#> [1] 137.6542
```

It is worth noting that this Bayes factor of 137.65 is remarkably high, considering the data that has been observed. However, this high value is not unexpected since the Bayes factor depends on the prior distribution for $\theta$. Typically, when the prior distribution expresses a very conservative opinion on the population misstatement, as is the case with the beta(1, 1) prior, the Bayes factor tends to overestimate the evidence in favor of the hypothesis of tolerable misstatement. To mitigate this, you can use a prior distribution that is impartial towards the hypotheses by using `method = "impartial"` in the `auditPrior()` function (Derks et al., 2022a).

```
prior <- auditPrior(materiality = 0.03, method = "impartial",
 ↪  likelihood = "binomial")
evaluation(materiality = 0.03, x = 1, n = 100, prior = prior)
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  1 and 100
#> number of errors = 1, number of samples = 100, taint = 1, BF  =
#> 7.7685
#> alternative hypothesis: true misstatement rate is less than 0.03
#> 95 percent credible interval:
#>  0.00000000 0.03806016
#> most likely estimate:
#>  0.0082131
#> results obtained via method 'binomial' + 'prior'
```

The analysis above was conducted using an impartial prior. The resulting output indicates that $BF_{10} = 7.77$, which moderately supports $H_1$. This outcome suggests that the population contains misstatements lower than five percent (tolerable misstatement), assuming impartiality. Both prior distributions resulted in persuasive Bayes factors, making the results reliable regardless of the prior distribution selected. As a result, the auditor can confidently assert that the sample data provides evidence that the population does not contain a material misstatement.

### 6.1.3 Using Data

Previously, we relied on summary statistics obtained from a sample to carry out evaluations. However, it is also possible to supply the `evaluation()` function with a data set. Doing so allows the function to calculate misstatements based on the booked and audited values of individual items.

To demonstrate how this works, we will use the `allowances` data set that comes with the **jfa** package. This data set includes $N = 4076$ financial statement line items, each with a booked value (`bookValue`) and an audited (true) value (`auditValue`) for

illustrative purposes. The total value of the population is $16,772,249. Since this example focuses on the evaluation stage of an audit, the sample is already identified within the data set. For this example, the performance materiality has been set at five percent, or $838,612.5.

```
data(allowances)
head(allowances)
#>   item branch bookValue auditValue times
#> 1    1     12      1600       1600     1
#> 2    2     12      1625         NA     0
#> 3    3     12      1775         NA     0
#> 4    4     12      1250       1250     1
#> 5    5     12      1400         NA     0
#> 6    6     12      1190         NA     0
```

When evaluating an audit sample using a data set, it is necessary to specify the `data`, `values`, and `values.audit` arguments in the `evaluation()` function. The input for these arguments should be the name of the relevant column in `data`. For example, the call below evaluates the `allowances` sample using a classical evaluation procedure. In this case, the output shows that the estimate of the misstatement in the population is 15.77 percent, with the 95 percent (one-sided) confidence interval ranging from 0 percent to 17.5 percent.

```
x <- evaluation(
  materiality = 0.05, data = allowances, times = "times",
  values = "bookValue", values.audit = "auditValue"
)
summary(x)
#>
#>  Classical Audit Sample Evaluation Summary
#>
#> Options:
#>   Confidence level:               0.95
#>   Materiality:                    0.05
#>   Hypotheses:                     H : θ >= 0.05 vs. H : θ < 0.05
#>   Method:                         poisson
#>
#> Data:
#>   Sample size:                    1604
#>   Number of errors:               401
#>   Sum of taints:                  252.9281046
#>
#> Results:
#>   Most likely error:              0.15769
#>   95 percent confidence interval: [0, 0.175]
#>   Precision:                      0.017311
#>   p-value:                        1
```

## 6.2 Partial Misstatements

Partial misstatements occur when there is only a partial discrepancy between the true value of an item in the sample and its recorded value. From a data perspective, this implies that the misstatements in the sample are not just binary (i.e., 0 or 1) but also often lie between 0 and 1. For instance, a receipt for goods received can be partially misstated if only part of the goods have been received. In practice, this means that the usual methods for evaluating binary misstatements do not suffice.

To illustrate how auditors can adequately deal with this type of misstatements, we examine a realistic sample from a financial audit using the `allowances` data set included in the **jfa** package. For clarity, we perform some data pre-processing to arrive at the population for our example. In this case, the performance materiality (i.e., the maximum tolerable misstatement) for the population is set to $\theta_{max} = 0.1$, or ten percent.

```
population <- allowances[, c(1, 3, 4)]
population <- population[population[["bookValue"]] > 0, ]
population <- population[!is.na(population[["auditValue"]]), ]
head(population)
#>    item bookValue auditValue
#> 1     1      1600       1600
#> 4     4      1250       1250
#> 7     7      1150       1150
#> 10   10      1250       1250
#> 12   12      6700       6700
#> 13   13      1450       1450
```

To decide whether the misstatement in the population is lower than the performance materiality of ten percent, a sample of $n = 60$ monetary units distributed across the same number of items is selected from the population of $N = 1177$ items. In this case, the sample is selected using a fixed interval sampling method in which the items in the population are randomized before selection.

```
set.seed(21)
sample <- selection(
  population, size = 60, randomize = TRUE,
  method = "interval", units = "values", values = "bookValue"
)$sample
```

After inspecting the items in the sample, each item is annotated with its recorded (i.e., book) value and its true (i.e., audit) value. When evaluating partial misstatements, auditors often calculate the 'taint' $t_i$ of each item $i$, which is defined as the proportional misstatement in the item:

$$t_i = \frac{\text{Book value}_i - \text{Audit value}_i}{\text{Book value}_i}. \tag{6.1}$$

For instance, if item $i$ is booked for \$1,000 but has an audit value of \$500 it has a

taint of $t_i = 0.5$. On the other hand, if the item does not contain any misstatement, it has a taint of $t_i = 0$. Lastly, if the item is fully misstated (i.e., it has a book value of \$0), its taint is $t_i = 1$.

As is often the case in practice, this audit sample contains many items that are correct (34), some items that are fully misstated (4), and some items that are partially misstated (22). The table below shows the distribution of these misstatements.

```
sample[["diff"]] <- sample[["bookValue"]] - sample[["auditValue"]]
sample[["taint"]] <- sample[["diff"]] / sample[["bookValue"]]
table(round(sample[["taint"]], 2))
#>
#>    0 0.02  0.1  0.2 0.25 0.33 0.35 0.38  0.5 0.53  0.6 0.75  0.8
 ↪  1
#>   34    1    6    1    3    2    2    1    2    1    1    1    1
 ↪  4
```

Typically, techniques for evaluating audit samples overlook the prevalence of many zeros in the data. Yet, considering the abundance of zeros while modeling the misstatement in the population can enhance the auditor's efficiency. In the following sections, we demonstrate several approaches that can be used to deal with partial misstatements and show how they improve the auditor's efficiency.

### 6.2.1 Classical Evaluation

Classical (i.e., frequentist) methods solely look to the data for estimating the parameters in a statistical model via maximum likelihood estimation. This methodology is widely adopted in auditing, and therefore we will discuss it first in this vignette.

#### 6.2.1.1 Binomial Likelihood

The easiest way to analyze partial misstatements is to aggregate them and simply extrapolate the sum of proportional misstatements (i.e., the total taint) to the population. Under the hood, this approach employs a binomial likelihood to estimate the misstatement: $k \sim \text{Binomial}(n, \theta)$. In this model, $\theta$ represents the misstatement in the population. This method uses the sum of the taints, denoted as $k = \sum_{i=0}^{n} t_i$ as the number of misstatements, together with the sample size $n$.

While aggregating the taints in this manner may not adhere to a strictly 'clean' modeling approach, as the binomial likelihood is only defined for binary data representing complete misstatements, it proves to be effective in estimating the misstatement (Broeze, 2006, Chapter 4.3). Despite its somewhat unconventional nature, the analytical feasibility of this approach makes it easy to use and therefore it is commonly applied in practice.

For the classical binomial likelihood, aggregating the taints can be done using `method = "binomial"` in the `evaluation()` function. Using maximum likelihood estimation, $\theta = \frac{\sum t}{n} = \frac{11.003}{60} = 0.18338$.

```
evaluation(
  method = "binomial", data = sample, materiality = 0.1,
  values = "bookValue", values.audit = "auditValue"
)
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  sample
#> number of errors = 26, number of samples = 60, taint = 11.003,
#> p-value = 0.98545
#> alternative hypothesis: true misstatement rate is less than 0.1
#> 95 percent confidence interval:
#>  0.0000000 0.2852153
#> most likely estimate:
#>  0.18338
#> results obtained via method 'binomial'
```

As shown in the output, the most likely misstatement is estimated to be 18.34 percent, with a 95 percent upper bound of 28.52 percent. Note that this approach might be effective (i.e., estimate the misstatement well) but not not very efficient (i.e., has a relatively high upper bound). That is because it does not take into account the information in the distribution of the taints.

#### 6.2.1.2 Stringer Bound

The Stringer bound is a commonly used method to evaluate audit samples. It is attractive because it takes into account the magnitude of the taints, thereby resulting in a smaller confidence interval (i.e., a lower upper bound). Note that, because it takes into account the magnitude of the taints (Bickel, 1992; Stringer, 1963), the Stringer bound only works if the actual data are present to calculate the taints.

Here we describe te calculation of the typical Stringer bound using the binomial distribution. The quantity $p(0; 1 - \alpha)$ is the Clopper-Pearson one-sided upper confidence bound for a binomial parameter with 0 successes in $n$ trials which, for zero errors, can be calculated as $1 - \alpha^{\frac{1}{n}}$. The more general $p(j; 1 - \alpha)$ is the Clopper-Pearson one-sided upper confidence bound for binomial parameter with $j$ successes in $n$ trials (Clopper & Pearson, 1934). In other words, it is the proportion corresponding to a binomial distribution with $\alpha$ percent chance that $j$ or less errors are observed in $n$ observations. That means that $p(j; 1 - \alpha)$ is the unique solution of:

$$\sum_{k=j+1}^{n} \binom{n}{k} p^k (1-p)^{n-k} = 1 - \alpha. \tag{6.2}$$

The outcome of this equation is equal to the $1 - \alpha$ percentile of a beta$(1 + i, n - i)$ distribution (Pearson, 1948). The Stringer bound is calculated using the Clopper-Pearson bounds, the number of overstatements $m_+$ and the overstatement taints $z_+$. When calculating the bound, the taints are placed in descending order in the formula as a form of conservatism.

$$p(0; 1-\alpha) + \sum_{j=1}^{m_+} [p(j; 1-\alpha) - p(j-1; 1-\alpha)] \cdot z_{+_j} \tag{6.3}$$

In `jfa`, the Stringer bound can be applied using `method = "stringer"` in the `evaluation()` function. Note that the Stringer bound can also be calculated using the Poisson or hypergeometric distributions. The **jfa** package supports the Stringer bound in the `evaluation()` function using `method = "stringer.poisson"`, `stringer.binomial` or `stringer.hypergeometric`, depending on the preferred distribution. The maximum likelihood estimate for the Stringer bound is the same as that of the binomial likelihood.

```
evaluation(
  method = "stringer", data = sample, materiality = 0.1,
  values = "bookValue", values.audit = "auditValue"
)
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  sample
#> number of errors = 26, number of samples = 60, taint = 11.003
#> 95 percent confidence interval:
#>  0.0000000 0.2799705
#> most likely estimate:
#>  0.18338
#> results obtained via method 'stringer.binomial'
```

As shown in the output, the most likely misstatement is estimated to be 18.34 percent, with a 95 percent upper bound of 27.99 percent. Since the upper bound of the Stringer method is (slightly) lower than that of the binomial likelihood, the Stringer bound is more efficient.

#### 6.2.1.3 Hurdle Beta Model

Efficiency can be further improved by explicitly modeling the probability of a taint being zero (i.e., zero-inflation). This is a more realistic method because, in practice, most taints are zero. More formally, if there are explicit zeros present in the data, a model that incorporates a separate probability of being zero should be unequivocally favored over a model that does not. That is because the probability mass associated with the zeros in the adjusted model are infinitely larger than their probability densities under the traditional model.

In **jfa**, there are two types of models that incorporate a probability of the taint being zero. These are the zero-inflated Poisson model and the hurdle beta model. In the zero-inflated Poisson model, the idea is to incorporate extrazeros into the zeros already present in the Poisson distribution. However, this idea does not directly apply to a "zero-inflated beta" model. Since the beta likelihood cannot accommodate zeros, there are no zeros to inflate. A more appropriate name for this model is therefore a hurdle (i.e., two-component) beta model. This terminology reflects the idea that the

model has to overcome the hurdle of zero values which are not inherently part of the beta likelihood.

The formal specification of the hurdle beta model is given below. Here, $p_{(0,1]}$ is the probability of a misstatement occurring, regardless of its size. Additionally, $p_{1|(0,1]}$ is the conditional probability of a misstatement being a full misstatement. Furthermore, $\phi$ is the average (non-zero-one) taint and $\nu$ is the concentration of these taints. The total misstatement in the population is $\theta = p_{(0,1)} \cdot \phi + p_1$.

$$t = \begin{cases} 0 & \text{with probability } p_0 = 1 - p_{(0,1]} \\ (0,1] & \begin{cases} \text{Beta}(\phi\nu, (1-\phi)\nu) & \text{with probability } p_{(0,1)} = p_{(0,1]} \cdot (1 - p_{1|(0,1]}) \\ 1 & \text{with probability } p_1 = p_{(0,1]} \cdot p_{1|(0,1]} \end{cases} \end{cases}$$

In `jfa`, this hurdle beta model can be fitted using `method = "hurdle.beta"` in the `evaluation()` function. To estimate the parameters in this model using the maximum likelihood, we use the default option `prior = FALSE`. Using maximum likelihood estimation, $p_{(0,1]} = \frac{n_t}{n} = \frac{26}{60} = 0.433$, where $n_t$ is the number of misstatements, and $p_{1|(0,1]} = \frac{n_1}{n_t} = \frac{4}{26} = 0.1539$, where $n_1$ is the number of full misstatements. Furthermore, $\phi = \frac{\sum t - n_1}{n_t - n_1} = \frac{11.003 - 4}{26 - 4} = 0.3183$. Hence, the most likely misstatement is estimated to be $\theta = (p_{(0,1]} \cdot (1 - p_{1|(0,1]}) \cdot \phi) + (p_{(0,1]} \cdot p_{1|(0,1]}) = (0.433 \cdot (1 - 0.1539) \cdot 0.3183) + (0.433 \cdot 0.1539) = 0.18338$. The upper bound cannot be derived analytically and has to be determined by drawing samples from the fitted model. Hence, it is recommended to call `set.seed()` before executing the following command to make the results reproducible.

```
set.seed(1)
evaluation(
  method = "hurdle.beta", data = sample, materiality = 0.1,
  values = "bookValue", values.audit = "auditValue"
)
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  sample
#> number of errors = 26, number of samples = 60, taint = 11.003
#> alternative hypothesis: true misstatement rate is less than 0.1
#> 95 percent confidence interval:
#>  0.0000000 0.2577749
#> most likely estimate:
#>  0.18338
#> results obtained via method 'hurdle.beta'
```

As shown in the output, the most likely misstatement is estimated to be 18.34 percent, with a 95 percent upper bound of around 26.28 percent. Since the upper bound of the hurdle beta model is (slightly) lower than that of the Stringer bound, the hurdle beta model is more efficient.

We leave it to the interested reader to apply the zero-inflated Poisson model to the data from this sample using `method = "inflated.poisson"`. Please note that this requires specification of the number of items in the population (`N.items`) and the number of monetary units in the population (`N.units`).

### 6.2.2 Bayesian Evaluation

In the Bayesian framework, the auditor defines a prior distribution for the parameters in the model. This approach offers a notable advantage for models dealing with partial misstatement, as the prior distribution has a regularizing effect, reducing the amount of data required for reliable inferences. More specifically, frequentist models cannot be optimized when there are no non-zero taints in the sample. Additionally, the prior distribution enables auditors to integrate pre-existing information about the misstatement into the statistical analysis, thereby increasing efficiency when the auditor has pre-existing information about the misstatement.

#### 6.2.2.1 Beta Distribution

The prior distribution we typically assign in the binomial model is the beta distribution. Just as we handle taints in the binomial likelihood, they can also be aggregated given the prior distribution. The statistical model is $k \sim \text{Binomial}(n,\ \theta)$ and the prior distribution that is used is $\theta \sim \text{Beta}(\alpha,\ \beta)$. Given the data $n$ and $k = \sum t$, the posterior distribution for the default beta(1, 1) prior distribution (`prior = TRUE`) can be derived analytically and is $\theta \sim \text{Beta}(\alpha = 1 + \sum t,\ \beta = 1 + n - \sum t)$.

For the binomial likelihood, this Bayesian model can be applied in the `evaluation()` function using `method = "binomial"` in comination with `prior = TRUE`. However, the input for the `prior` argument can also be an object created by the `auditPrior()` function.

```
eval_bayes <- evaluation(
  method = "binomial", data = sample, materiality = 0.1,
  values = "bookValue", values.audit = "auditValue", prior = TRUE
)
eval_bayes
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  sample
#> number of errors = 26, number of samples = 60, taint = 11.003, BF
#> = 0.15107
#> alternative hypothesis: true misstatement rate is less than 0.1
#> 95 percent credible interval:
#>  0.0000000 0.2808365
#> most likely estimate:
#>  0.18338
#> results obtained via method 'binomial' + 'prior'
```

As shown in the output, the most likely misstatement is estimated to be 18.34 percent,

with a 95 percent upper bound of 28.1 percent. The prior and posterior distribution can be visualized via the `plot(..., type = "posterior")` command in Figure 6.4.

```
plot(eval_bayes, type = "posterior")
```

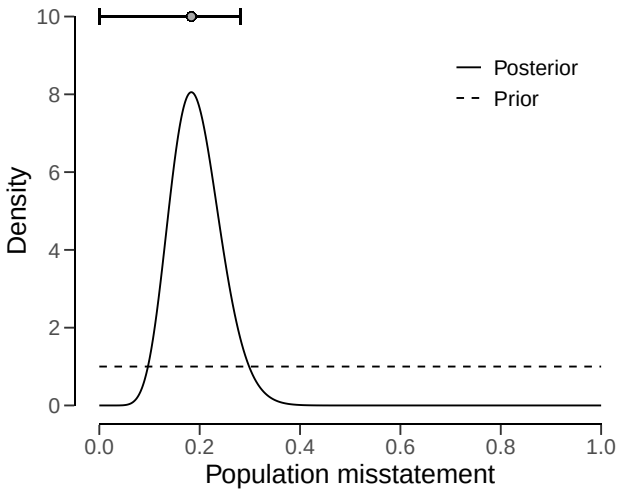



**Figure 6.4.** The beta prior and posterior distribution on the range [0; 1] after seeing a total taint of 11.003 in the sample of 60 units. The total taint is made up of 26 individual full or partial misstatements.

#### 6.2.2.2 Hurdle Beta Model

It is also possible to fit a Bayesian variant of the hurdle beta model. This entails applying prior distributions for the parameters $p$, $\phi$, and $\nu$. The prior distributions used by **jfa** for the Bayesian hurdle beta model are $p_{(0,1]} \sim \text{Beta}(\alpha, \beta)$ (to be specified by the user), $p_{1|(0,1]} \sim \text{Beta}(1, 1)$, $\phi \sim \text{Beta}(1, 1)$, and $\nu \sim \text{Pareto}(1, \frac{3}{2})$.

This Bayesian hurdle beta model can be applied using `method = "hurdle.beta"` in combination with `prior = TRUE` (or an object created by the `auditPrior()` function) in the `evaluation()` function. In this case, the posterior distribution cannot be determined analytically and must be determined using MCMC sampling, which is why it is recommended to call `set.seed()` before computing the results.

```
set.seed(2)
eval_hurdle_bayes <- evaluation(
  method = "hurdle.beta", data = sample, materiality = 0.1,
  values = "bookValue", values.audit = "auditValue", prior = TRUE
)
eval_hurdle_bayes
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  sample
```

```
#> number of errors = 26, number of samples = 60, taint = 11.003, BF
#> = 0.007913
#> alternative hypothesis: true misstatement rate is less than 0.1
#> 95 percent credible interval:
#>  0.000000 0.259837
#> most likely estimate:
#>  0.179
#> results obtained via method 'hurdle.beta' + 'prior'
```

As shown in the output, the most likely misstatement is estimated to be 17.73 percent, with a 95 percent upper bound of around 26.51 percent. The prior and posterior distribution can be visualized using the `plot()` function in Figure 6.5.

```
plot(eval_hurdle_bayes, type = "posterior")
```

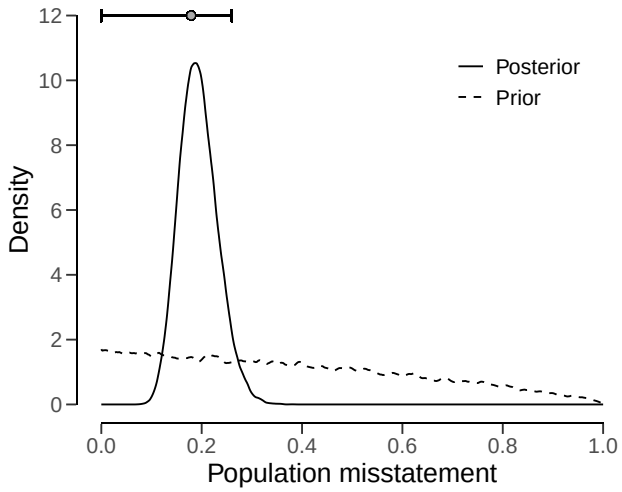



**Figure 6.5.** The hurdle beta prior and posterior distribution on the range [0; 1] after seeing a total taint of 11.003 in the sample of 60 units. The total taint is made up of 26 individual full or partial misstatements.

## 6.3 Practical Exercises

1. Evaluate a sample of $n = 30$ items containing $k = 2$ misstatements. Use the classical approach.

## 6.4 Answers to the Exercises

1. The evaluation can be performed using the `evaluation()` function with the default arguments.

```
evaluation(n = 30, x = 2, method = "binomial")
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  2 and 30
#> number of errors = 2, number of samples = 30, taint = 2
#> 95 percent confidence interval:
#>  0.000000 0.195326
#> most likely estimate:
#>  0.066667
#> results obtained via method 'binomial'
```

# Chapter 7

# Stratified Evaluation

In the realm of audit sampling, the practice of stratified sampling is a powerful technique that enables auditors to enhance the accuracy and representativeness of their samples. Stratified sampling is a strategy employed in audit sampling where the population is divided into distinct subgroups or strata based on relevant characteristics. These characteristics could include geographical location, department, business unit, or any other factor that might influence the distribution of errors or misstatements. By segregating the population into strata, auditors can ensure that their sample captures the diversity of the entire population. This chapter delves into the intricacies of stratified evaluation, exploring three distinct approaches—no pooling, complete pooling, and partial pooling—each with its own advantages and trade-offs. Knowledge of these three approaches is crucial for auditors aiming to optimize their statistical evaluation and generate robust population estimates.

Consider an example of auditing expense claims in a large organization. Instead of treating all expense claims as a uniform entity, auditors can stratify the claims based on the departments they belong to. This approach ensures that the audit sample includes a representative mix of claims from different departments, thereby reducing the risk of overlooking specific areas of concern. Another example of such a situation would be a group audit where the audited organization consists of different components or branches. Stratification is relevant for the group auditor if they must form an opinion on the group as a whole because they must aggregate the samples taken by the component auditors.

In general, there are three approaches to evaluating a stratified sample: no pooling, complete pooling, and partial pooling. No pooling assumes no similarities between strata, which means that all strata are analyzed independently. Complete pooling assumes no difference between strata, which means that all data is aggregated and analyzed as a whole. Finally, partial pooling assumes differences and similarities between strata, which means that information can be shared between strata. Partial pooling (i.e., hierarchical modeling) is a powerful technique that can result in more efficient population and stratum estimates.

As a data example, consider the `retailer` data set that comes with the package. The

**Figure 7.1.** There are three approaches to evaluating a stratified audit sample: no pooling, complete pooling and partial pooling. Image available under a CC-BY-NC 4.0 license.

organization in question consists of 20 branches across the country. In each of the 20 strata, a component auditor has taken a statistical sample and reported the outcomes to the group auditor.

```
data(retailer)
head(retailer)
#>   stratum items samples errors
#> 1       1  5000     300     21
#> 2       2  5000     300     16
#> 3       3  5000     300     15
#> 4       4  5000     300     14
#> 5       5  5000     300     16
#> 6       6  5000     150      5
```

The number of units per stratum in the population can be provided with `N.units` to weigh the stratum estimates to determine population estimate. This is called poststratification. If `N.units` is not specified, each stratum is assumed to be equally represented in the population.

## 7.1 No pooling

No pooling (`pooling = "none"`, default) assumes no similarities between strata. This means that the prior distribution specified through `prior` is applied independently for each stratum. This allows for independent estimates for the misstatement in each stratum but also results in a relatively high uncertainty in the population estimate. Assuming a binomial likelihood and a beta($\alpha$, $\beta$) prior on $\theta$ (these choices may differ among analysts), the statistical model applied in the no pooling approach is the following:

$$k_s \sim \text{Binomial}(n_s, \theta_s) \tag{7.1}$$
$$\theta_s \sim \text{Beta}(\alpha, \beta) \tag{7.2}$$
$$\theta \leftarrow \frac{\sum \theta_s N_s}{N} \tag{7.3}$$

The call below evaluates the sample using a Bayesian stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate. Since the posterior distribution is determined via sampling it is important to use `set.seed()` to make the results reproducible.

```
set.seed(1)
result_np <- evaluation(
  materiality = 0.05,
  method = "binomial",
  n = retailer[["samples"]],
  x = retailer[["errors"]],
  N.units = retailer[["items"]],
  alternative = "two.sided",
  pooling = "none",
  prior = TRUE
)
result_np
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  115 and 2575
#> number of errors = 115, number of samples = 2575, taint = 115, BF
#> = 0
#> alternative hypothesis: true misstatement rate is not equal to 0.05
#> 95 percent credible interval:
#>  0.04276326 0.08220109
#> most likely estimate:
#>  0.0598
#> results obtained via method 'binomial' + 'no-pooling' + 'prior'
```

In this case, the output of the `summary()` function shows that the estimate of the misstatement in the population is 5.98 percent, with the 95 percent credible interval ranging from 4.28 percent to 8.22 percent. The stratum estimates can be visualized using the `plot()` function in combination with `type = "estimates`, see Figure 7.2. Estimation plots display stratum estimates and their uncertainties, revealing the differences and overlaps between strata. As the figure shows, the stratum estimates differ substantially from each other but are relatively uncertain.

```
plot(result_np, type = "estimates")
```

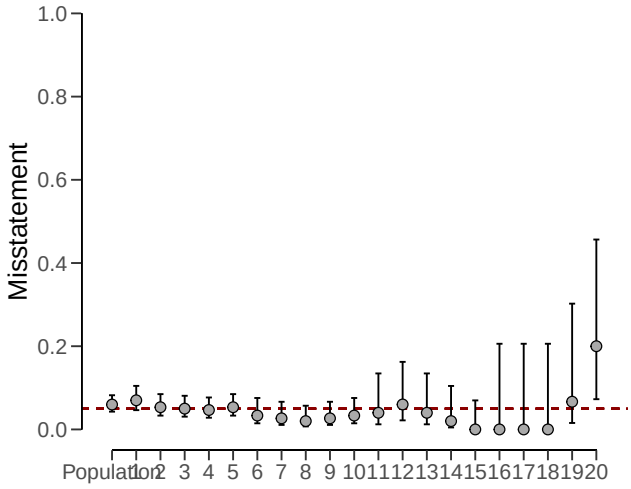



**Figure 7.2.** Estimates of the population and stratum misstatement under the no pooling model.

Posterior distribution plots provide insights into how the prior beliefs evolve after considering the data, showcasing the gradual convergence of information. The prior and posterior distribution for the population misstatement can be requested via the `plot()` function, see Figure 7.3.

```
plot(result_np, type = "posterior")
```

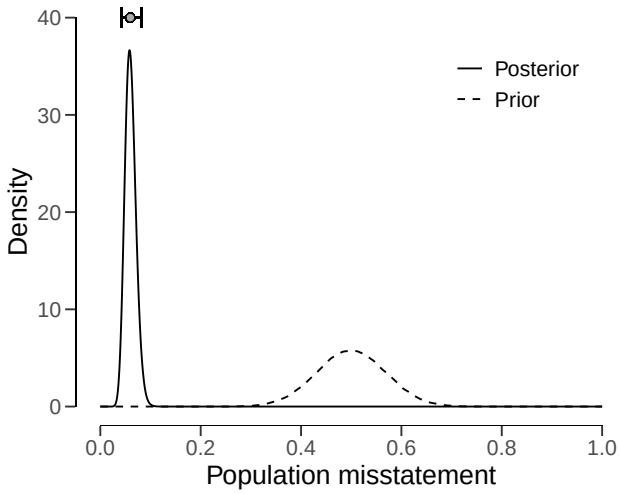



**Figure 7.3.** Prior and posterior distribution for the population misstatement under the no pooling model.

## 7.2 Complete pooling

Complete pooling (`pooling = "complete"`) assumes no differences between strata. This has the advantages that data from all strata can be aggregated, which decreases the uncertainty in the population estimate compared to the no pooling approach. However, the disadvantage of this approach is that it does not facilitate the distinction between between strata, as every stratum receives the same estimate equal to that of the population, see Figure 7.4. Assuming a binomial likelihood and a beta($\alpha$, $\beta$) prior on $\theta$, the statistical model applied in the complete pooling approach is the following:

$$k \sim \text{Binomial}(n, \theta) \tag{7.4}$$
$$\theta \sim \text{Beta}(\alpha, \beta) \tag{7.5}$$

The call below evaluates the sample using a Bayesian stratified evaluation procedure, in which the strata are assumed to be the same.

```
result_cp <- evaluation(
  materiality = 0.05,
  method = "binomial",
  n = retailer[["samples"]],
  x = retailer[["errors"]],
  N.units = retailer[["items"]],
  alternative = "two.sided",
  pooling = "complete",
  prior = TRUE
)
result_cp
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  115 and 2575
#> number of errors = 115, number of samples = 2575, taint = 115, BF
#> = 0.022725
#> alternative hypothesis: true misstatement rate is not equal to 0.05
#> 95 percent credible interval:
#>  0.03735031 0.05334542
#> most likely estimate:
#>  0.04466
#> results obtained via method 'binomial' + 'complete-pooling' +
 ↪  'prior'
```

For example, the output of the `summary()` function shows that the estimate of the misstatement in the population is 4.47 percent, with the 95 percent credible interval ranging from 3.74 percent to 5.34 percent. Since the data is aggregated, the stratum estimates contain relatively little uncertainty. However, the probability of misstatement in stratum 20 (many misstatements) under this assumption is the same as that of stratum 15 (few misstatements).

```
plot(result_cp, type = "estimates")
```

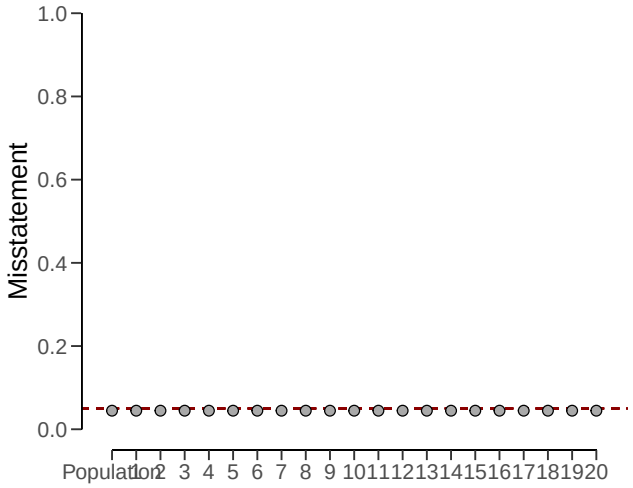



**Figure 7.4.** Estimates of the population and stratum misstatement under the complete pooling model.

The prior and posterior distribution for the population misstatement can be requested via the `plot()` function, see Figure 7.5.

```
plot(result_cp, type = "posterior")
```

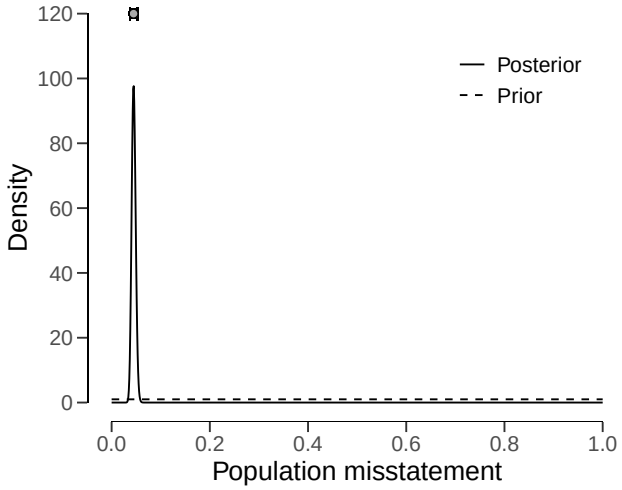



**Figure 7.5.** Prior and posterior distribution for the population misstatement under the complete pooling model.

## 7.3 Partial pooling

Finally, partial pooling (`pooling = "partial"`) assumes differences and similarities between strata. This allows the auditor to differentiate between strata, while also sharing information between the strata to reduce uncertainty in the population estimate. Assuming a binomial likelihood and a beta prior for $\theta$, the statistical model applied in the partial pooling approach is the following:

$$k_s \sim \text{Binomial}(n_s, \theta_s) \tag{7.6}$$
$$\theta_s \sim \text{Beta}(\phi\nu, (1-\phi)\nu) \tag{7.7}$$
$$\phi \sim \text{Beta}(\alpha, \beta) \tag{7.8}$$
$$\nu \sim \text{Pareto}(50, \frac{3}{2}) \tag{7.9}$$
$$\theta \leftarrow \frac{\sum \theta_s N_s}{N} \tag{7.10}$$

**Figure 7.6.** Partial pooling takes into account the hierarchical structure in the data. Image available under a CC-BY-NC 4.0 license.

The call below evaluates the sample using a Bayesian stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate.

```
set.seed(1)
result_pp <- evaluation(
  materiality = 0.05,
  method = "binomial",
  n = retailer[["samples"]],
  x = retailer[["errors"]],
  N.units = retailer[["items"]],
  alternative = "two.sided",
  pooling = "partial",
  prior = TRUE
```

```
)
result_pp
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  115 and 2575
#> number of errors = 115, number of samples = 2575, taint = 115, BF
#> = 0.033676
#> alternative hypothesis: true misstatement rate is not equal to 0.05
#> 95 percent credible interval:
#>  0.03268710 0.05325719
#> most likely estimate:
#>  0.0436
#> results obtained via method 'binomial' + 'partial-pooling' +
 ↪  'prior'
```

In this case, the output shows that the estimate of the misstatement in the population is 4.17 percent, with the 95 percent credible interval ranging from 3.26 percent to 5.35 percent. Note that this population estimate is substantially less uncertain than that of the no pooling approach. Figure 7.7 visualizes the population and stratum estimates. Note that, like in the no pooling approach, the stratum estimates are different from each other but lie closer together and are less uncertain.

```
plot(result_pp, type = "estimates")
```

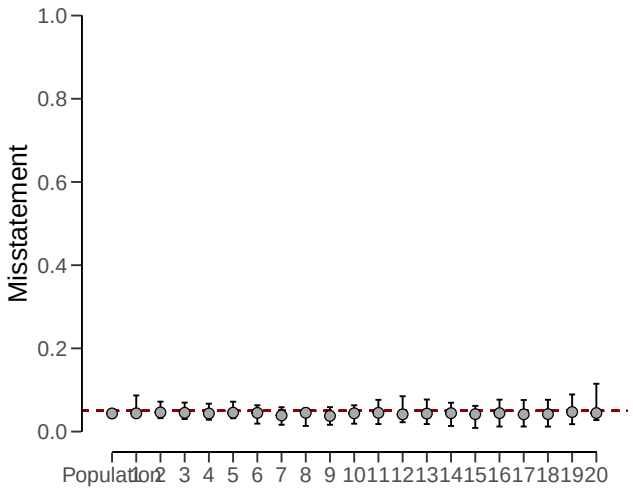



**Figure 7.7.** Estimates of the population and stratum misstatement under the partial pooling model.

The prior and posterior distribution for the population misstatement can be requested via the `plot()` function, see Figure 7.8.

```
plot(result_pp, type = "posterior")
```

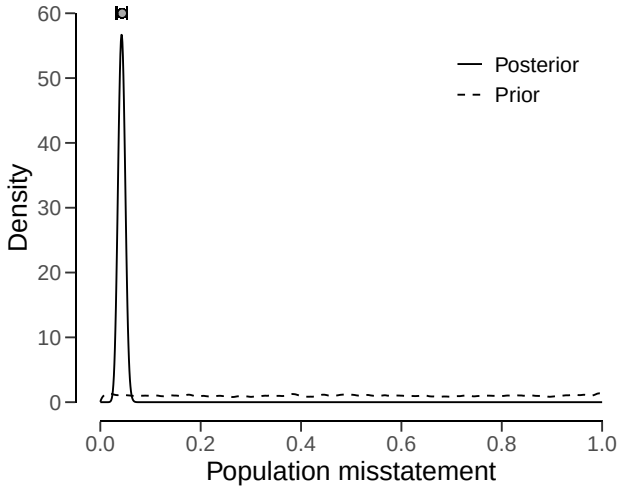



**Figure 7.8.** Prior and posterior distribution for the population misstatement under the partial pooling model.

## 7.4 Evaluation using data

To illustrate these concepts using data, let's consider the `allowances` dataset included in the package, which contains 3500 financial statement line items with book values (`bookValue`) and, for illustrative purposes, audited (true) values (`auditValue`) across different branches. Since the focus of this chapter is the evaluation stage in the audit, the sample is already indicated in the data set. The performance materiality in this example is set to five percent.

```
data(allowances)
head(allowances)
#>   item branch bookValue auditValue times
#> 1    1     12      1600       1600     1
#> 2    2     12      1625         NA     0
#> 3    3     12      1775         NA     0
#> 4    4     12      1250       1250     1
#> 5    5     12      1400         NA     0
#> 6    6     12      1190         NA     0
```

Evaluating a stratified sample using data requires specification of the `data`, `values`, `values.audit` and `strata` arguments in the `evaluation()` function. In this case, the units are monetary and calculated by aggregating the book values of the items in each stratum.

```
N.units <- aggregate(allowances[["bookValue"]],
 ↪  list(allowances[["branch"]]), sum)$x
```

### 7.4.1 Classical Evaluation

Using classical evaluation, auditors can apply stratified evaluation to assess the population misstatement rate. The estimates obtained under this approach reflect independent evaluation of each stratum, potentially leading to a relatively high uncertainty in the overall population estimate. The statistical model for this evaluation using the Poisson likelihood is relatively simple:

$$t_s \sim \text{Poisson}(n_s \theta_s) \tag{7.11}$$

$$\theta \leftarrow \frac{\sum \theta_s N_s}{N} \tag{7.12}$$

The call below evaluates the `allowances` sample using a classical stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate.

```
set.seed(1)
result_dnpc <- evaluation(
  materiality = 0.05,
  data = allowances,
  N.units = N.units,
  values = "bookValue",
  values.audit = "auditValue",
  strata = "branch",
  times = "times",
  alternative = "two.sided",
  pooling = "none"
)
result_dnpc
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  allowances
#> number of errors = 401, number of samples = 1604, taint = 252.93,
#> p-value = NA
#> alternative hypothesis: true misstatement rate is not equal to 0.05
#> 95 percent confidence interval:
#>  0.1254576 0.1827606
#> most likely estimate:
#>  0.14723
#> results obtained via method 'poisson' + 'no-pooling'
```

In this case, the output shows that the estimate of the misstatement in the population is 14.72 percent, with the 95 percent confidence interval ranging from 12.55 percent to

18.28 percent. The precision of the population estimate is 5.73 percent. The stratum estimates can be seen in the output of the `summary()` function and are visualized in Figure 7.9 below.

```
plot(result_dnpc, type = "estimates")
```

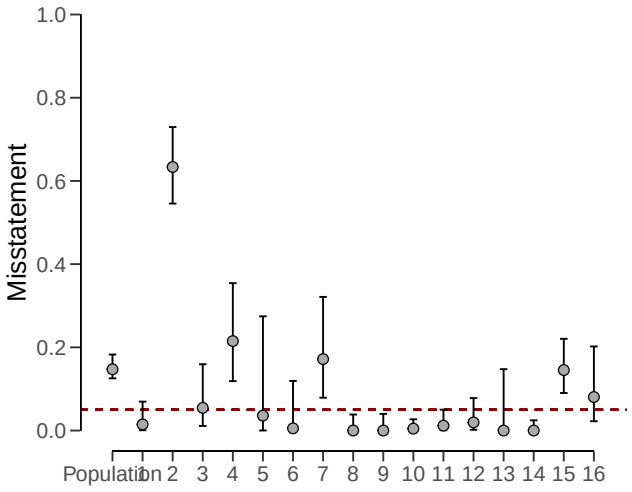



**Figure 7.9.** Estimates of the population and stratum misstatement under the no pooling model.

### 7.4.2 Bayesian Evaluation

Bayesian inference can improve upon the estimates of the classical approach by pooling information between strata where possible. The statistical model for this evaluation using the multilevel model with the use of taints is relatively complex:

$$t_i, s \sim \text{Beta}(\theta_s \kappa_s, (1 - \theta_s)\kappa_s) \tag{7.13}$$
$$\theta_s \sim \text{Beta}(\phi\nu, (1 - \phi)\nu) \tag{7.14}$$
$$\kappa_s \sim \text{Normal}(\mu, \sigma)^+ \tag{7.15}$$
$$\phi \sim \text{Beta}(\alpha, \beta) \tag{7.16}$$
$$\nu \sim \text{Pareto}(50, \frac{3}{2}) \tag{7.17}$$
$$\mu \sim \text{Normal}(1, 100)^+ \tag{7.18}$$
$$\sigma \sim \text{Normal}(0, 10)^+ \tag{7.19}$$
$$\theta \leftarrow \frac{\sum \theta_s N_s}{N} \tag{7.20}$$

The call below evaluates the `allowances` sample using a Bayesian multilevel stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate.

```
set.seed(1)
result_dnpb <- evaluation(
  materiality = 0.05,
  method = "binomial",
  data = allowances,
  N.units = N.units,
  values = "bookValue",
  values.audit = "auditValue",
  strata = "branch",
  times = "times",
  alternative = "two.sided",
  pooling = "partial",
  prior = TRUE
)
result_dnpb
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  allowances
#> number of errors = 401, number of samples = 1350, taint = 224.66,
#> BF  = Inf
#> alternative hypothesis: true misstatement rate is not equal to 0.05
#> 95 percent credible interval:
#>  0.1466362 0.1627476
#> most likely estimate:
#>  0.1534
#> results obtained via method 'binomial' + 'partial-pooling' +
 ↪  'prior'
```

The output shows that the estimate of the misstatement in the population is 15.36 percent, with the 95 percent credible interval ranging from 14.66 percent to 16.31 percent. The precision of the population estimate is 1.65 percent, which is substantially lower than that of the classical approach. The stratum estimates can be seen in the output of the `summary()` function and are visualized in Figure 7.10 below.

```
plot(result_dnpb, type = "estimates")
```

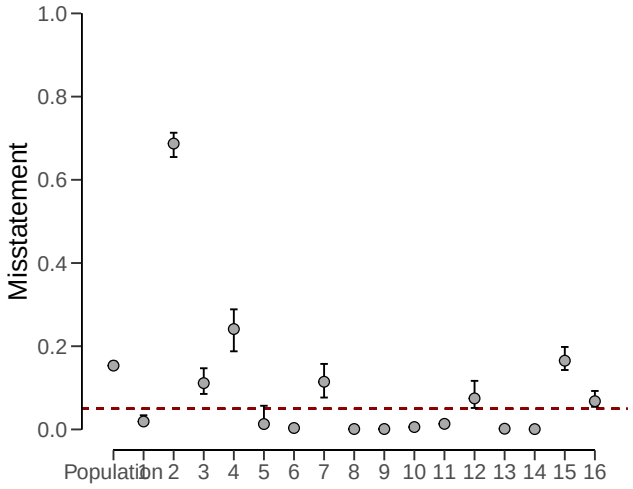



**Figure 7.10.** Estimates of the population and stratum misstatement under the partial pooling model.

The prior and posterior distribution for the population misstatement can be requested via the `plot()` function, see Figure 7.11.

```
plot(result_dnpb, type = "posterior")
```

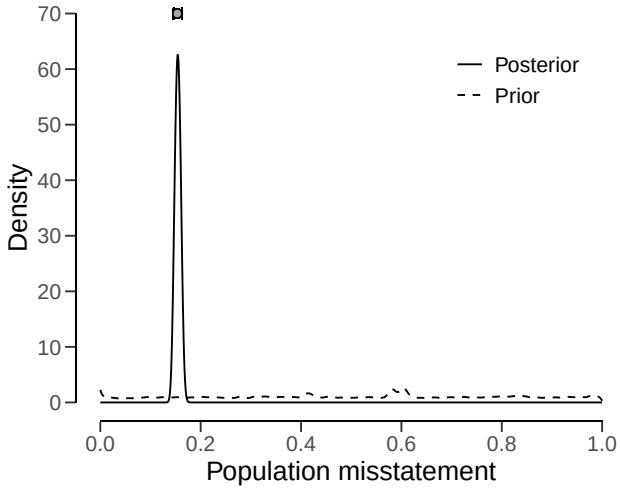



**Figure 7.11.** Prior and posterior distribution for the population misstatement under the partial pooling model.

Stratified evaluation is a pivotal tool in an auditor's arsenal, allowing for the analysis of diverse populations with varying characteristics. By embracing the principles of no

pooling, complete pooling, and partial pooling, auditors can tailor their evaluation strategies to balance independence and shared information, resulting in more accurate and reliable population estimates. The combination of these approaches with real-world data offers auditors a comprehensive toolkit to enhance the quality and efficiency of their evaluations.

## 7.5 Practical Exercises

1. Evaluate a stratified sample of $n_s = [30, 40, 50]$ items containing $k_s = [0, 1, 2]$ misstatements. Use the classical approach.

## 7.6 Answers to the Exercises

1. To evaluate a stratified sample using the classical approach, the `evaluation()` function can be used with the default arguments.

```
evaluation(n = c(30, 40, 50), x = c(0, 1, 3), method = "binomial")
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  4 and 120
#> number of errors = 4, number of samples = 120, taint = 4
#> 95 percent confidence interval:
#>  0.00000000 0.08888266
#> most likely estimate:
#>  0.028333
#> results obtained via method 'binomial' + 'no-pooling'
```

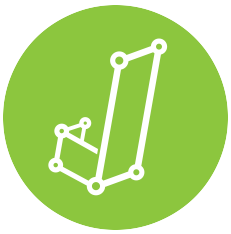
# Part III

# Software

# Chapter 8

# JASP for Audit

JASP (JASP Team, 2023; Love et al., 2019), an acronym for Jeffreys's Amazing Statistics Program, is a free and open-source software for statistical analysis developed at the University of Amsterdam. It is intended to be user-friendly and familiar to those who have experience with SPSS. A significant feature of JASP is that it provides most standard statistical analysis procedures in both their classical and Bayesian forms. Furthermore, the software is actively being translated into various languages, including Dutch.



**(a)** JASP


**(b)** JASP for Audit

**Figure 8.1.** JASP is an open-source statistical software program that provides free and flexible access to classical and Bayesian statistics. JASP for Audit is an add-on module for JASP that facilitates statistal auditing in both manifestations.

## 8.1 Downloading JASP

You can freely download JASP from their website `https://jasp-stats.org`. Simply click on the 'Download JASP' button on the homepage, and you will be taken to the download page. There, you can select your preferred installation option. JASP is compatible with Windows, MacOS, Linux, and Chrome OS. The installation process is fairly straightforward and familiar. Once installed, open the software to see the welcoming screen displayed below.

JASP for Audit (Derks, de Swart, Wagenmakers, et al., 2021) is an add-on module


**Figure 8.2.** Upon launching JASP, you are presented with the following screen. The icons in the ribbon display a range of statistical analyses. The menu icon located in the top left corner conceals the most crucial user options. The plus icon situated in the top right corner conceals the additional modules.

for JASP, based on the **jfa** package, that facilitates statistical audit sampling. The module provides graphical a user interface (GUI) for calculating sample sizes, selecting items according to standard audit sampling techniques, and performing inference about the population misstatement on the basis of a data sample or summary statistics of a sample. The module also features Bayesian equivalents of these analyses that enable the user to easily incorporate prior information into the statistical procedure. In all analyses, the Audit module offers explanatory text that helps the auditor in interpreting, explaining, and reporting the analysis. Since JASP for Audit is an R-based GUI around **jfa**, its functionality can be mapped almost one-on-one to that of the package.

The Audit module in JASP, labeled as "Audit" in the module list, is included by default in the software, but is not initially visible upon starting the program. To access the Audit module, click on the + icon in the top right corner of the JASP welcome screen, and select the module from the list of available options. The Audit module will then be displayed with a blue icon in the ribbon at the top of the screen.

Upon selecting the Audit module icon, the user can view all of the analyses that the module contains. It is important to note that some of these analyses are grayed out by default and can only be activated once a data set has been loaded into JASP. This means that the user must first import a data set in order to access and make use of these specific analyses. For a quick explanation on how to achieve this, see this student manual on the JASP website.


**Figure 8.3.** The + icon displays all modules that are currently available in JASP. To activate a module and add it to the ribbon at the top of the screen, click on the checkbox next to it.


**Figure 8.4.** By clicking the Audit module icon in the ribbon, you can view the various analyses that are included in the audit module.

## 8.2 Planning

To plan a sample for an audit using JASP, the procedure is comparable to using **jfa**. This means that both programs involve similar steps and considerations in order to effectively plan an audit sample. Like **jfa**, JASP for Audit offers a classical and a Bayesian approach to planning a sample.

The figure below showcases a snapshot of the classical planning analysis in JASP for Audit. The graphical user interface is displayed on the left side of the screen, while the audit report containing statistical results is displayed on the right side. In the user interface, the auditor can input the known parameters for the sample planning, after which JASP calculates and directly displays the statistical results.

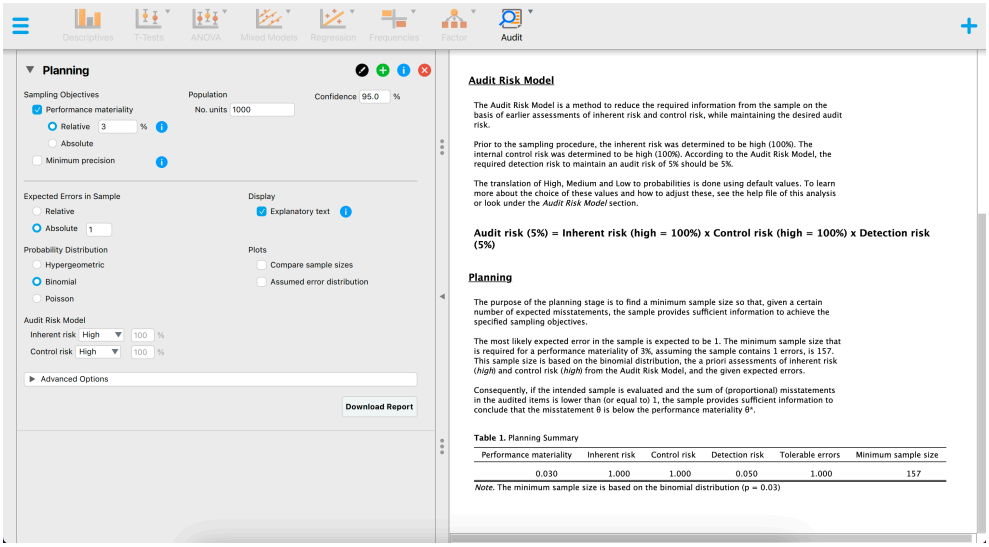



**Figure 8.5.** A snapshot of the classical planning analysis in JASP for Audit. In this analysis, the auditor is using the binomial likelihood, a sampling risk of five percent, a performance materiality of three percent and one expected misstatement in the sample. The resulting sample size is 157.

The above screenshot shows an analysis where the auditor is using the binomial distribution, a sampling risk of five percent, a performance materiality of three percent and one expected misstatement in the sample. The resulting sample size is $n = 157$.

Using **jfa**, these statistical results can be reproduced by executing the following code:

```
planning(materiality = 0.03, expected = 1, likelihood = "binomial")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 157
#> sample size obtained in 156 iterations via method 'binomial'
```

The figure below showcases a snapshot of the Bayesian planning analysis in JASP. The

graphical user interface is largely the same as the interface of the classical planning analysis, with the exception that we can specify a prior distribution with the options under “Prior”.

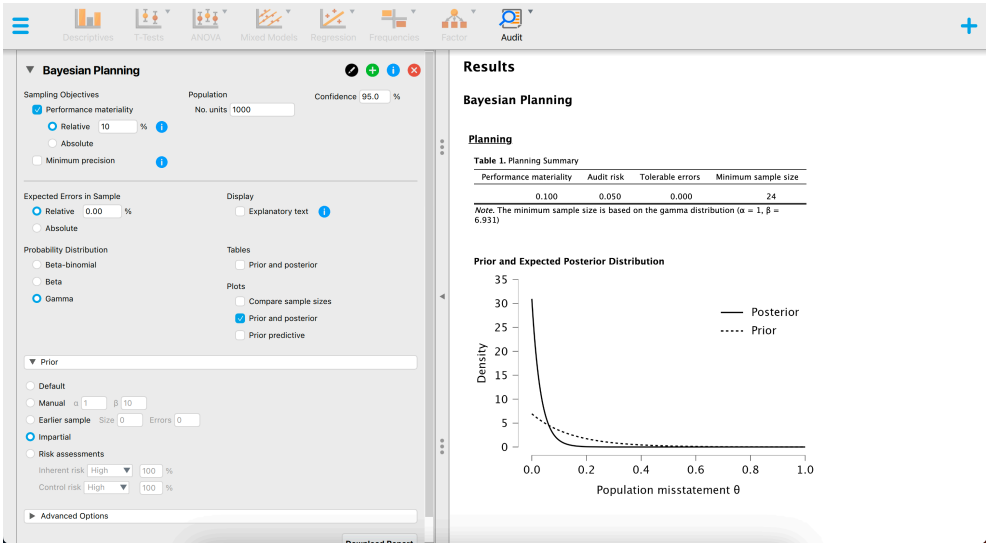



**Figure 8.6.** Snapshot of the Bayesian planning analysis in JASP for Audit. In this analysis, the auditor is using the Poisson likelihood with an impartial gamma prior, a sampling risk of five percent, a performance materiality of ten percent and no expected misstatements in the sample. The resulting sample size is 24.

The above screenshot shows an analysis where the auditor is using the Poisson likelihood together with an impartial gamma prior, a sampling risk of five percent, a performance materiality of ten percent and no expected misstatements in the sample. The resulting sample size is $n = 24$.

Using **jfa**, these statistical results can be reproduced by executing the following code:

```
prior <- auditPrior(method = "impartial", materiality = 0.1)
planning(materiality = 0.1, likelihood = "poisson", prior = prior)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 24
#> sample size obtained in 25 iterations via method 'poisson' +
 ↪  'prior'
```

## 8.3 Selection

Selecting a sample in JASP for Audit works similar to how you would do it in **jfa**. The figure below showcases a snapshot of the selection analysis in JASP for Audit. In the user interface, the auditor can input the known parameters for the sample selection,

after which JASP calculates and directly displays the statistical results.

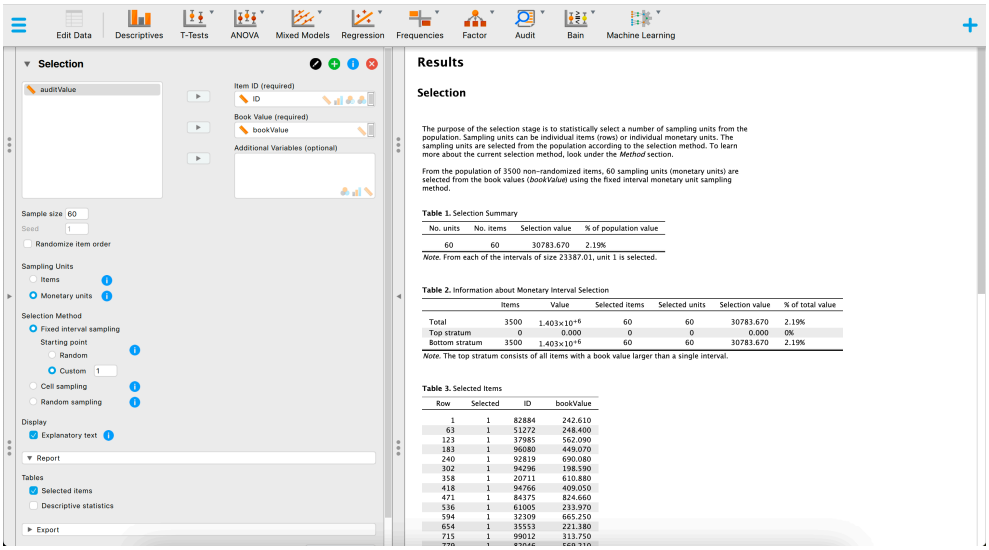



**Figure 8.7.** A snapshot of the selection analysis in JASP for Audit. In this analysis, the auditor is using a fixed interval monetary unit sampling method to select a sample of 60 monetary units from a population.

The above screenshot shows an analysis where the auditor is using a fixed interval monetary unit sampling method to select a sample of 60 monetary units from the `BuildIt` population. They use a starting point of 1.

Using **jfa**, these statistical results can be reproduced by executing the following code:

```
set.seed(1)
data(BuildIt)
result <- selection(
  data = BuildIt, size = 60,
  method = "interval", start = 1, values = "bookValue"
)
head(result[["sample"]])
#>   row times    ID bookValue auditValue
#> 1   1     1 82884    242.61     242.61
#> 2  60     1 70084    377.41     377.41
#> 3 118     1 59254    353.26     353.26
#> 4 176     1 27801    314.65     314.65
#> 5 235     1 98624    340.60     340.60
#> 6 293     1 38060    403.92     403.92
```

## 8.4 Evaluation

Finally, evaluating a sample in JASP for Audit works similar to how you would do it in **jfa**. The figure below showcases a snapshot of the evaluation analysis in JASP for Audit. In the user interface, the auditor can input the known parameters for the sample evaluation, after which JASP calculates and directly displays the statistical results.

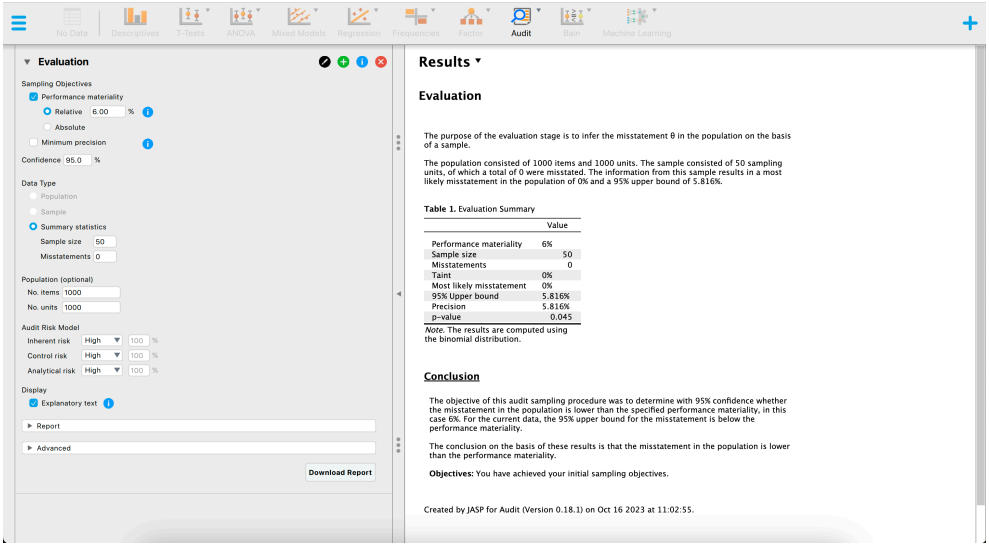



**Figure 8.8.** A snapshot of the classical evaluation analysis in JASP for Audit. In this analysis, the auditor is using the binomial likelihood, a sampling risk of five percent, a performance materiality of six percent and evaluates a sample of 50 items of which none contained a misstatement. The resulting upper bound is 5.8 percent and the $p$-value is 0.045.

The above screenshot shows an analysis where the auditor is using the binomial likelihood, a sampling risk of five percent, a performance materiality of six percent and evaluates a sample of $n = 50$ items of which $k = 0$ contained a misstatement. The resulting 95 percent upper confidence bound is 5.8 percent and the $p$-value is 0.045, which is lower than the sampling risk of five percent.

Using **jfa**, these statistical results can be reproduced by executing the following code:

```
evaluation(materiality = 0.06, method = "binomial", x = 0, n = 50)
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  0 and 50
#> number of errors = 0, number of samples = 50, taint = 0, p-value =
#> 0.045331
#> alternative hypothesis: true misstatement rate is less than 0.06
#> 95 percent confidence interval:
```

```
#>  0.00000000 0.05815508
#> most likely estimate:
#>  0
#> results obtained via method 'binomial'
```

The figure below showcases a snapshot of the Bayesian evaluation analysis in JASP. The graphical user interface is fairly similar to that of the classical evaluation analysis, with the exception that we can specify a prior distribution with the options under "Prior".

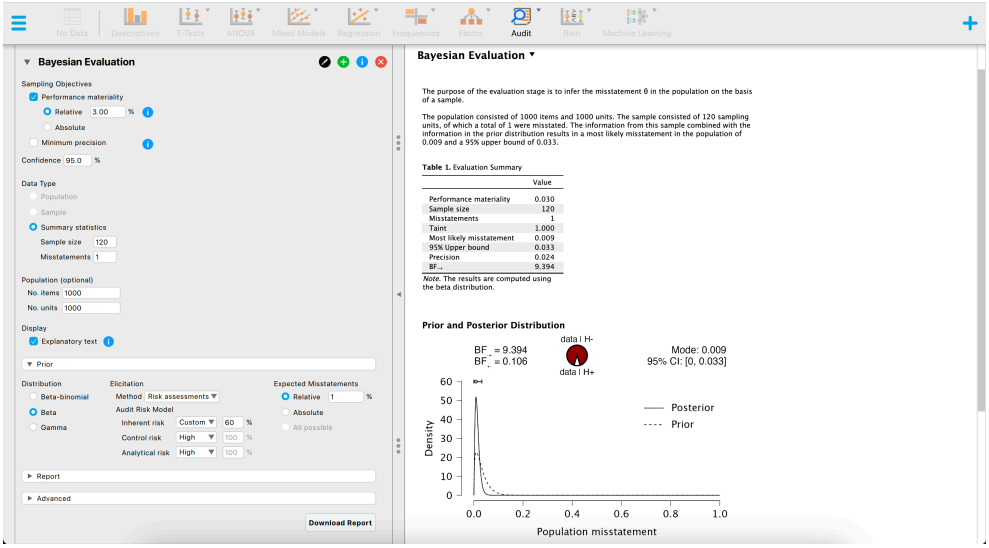



**Figure 8.9.** Snapshot of the Bayesian evaluation analysis in JASP for Audit. In this analysis, the auditor is using the binomial likelihood together with a beta prior distribution, a sampling risk of five percent, a performance materiality of three percent and evaluates a sample of 120 items of which one contained a misstatement. The resulting upper bound is 3.27 percent and the Bayes factor in favor of tolerable misstatement is 9.394.

The above screenshot shows an analysis where the auditor is using the binomial likelihood together with a beta prior distribution based on an expected error rate of three percent constructed using the audit risk model, a sampling risk of five percent, a performance materiality of three percent and evaluates a sample of $n = 120$ items of which $k = 1$ contained a misstatement. The resulting 95 percent upper credible bound is 3.27 percent and the Bayes factor in favor of tolerable misstatement is 9.394, indicating that the sample data are about 9 times more likely to occur under the hypothesis of tolerable misstatement than under the hypothesis of intolerable misstatement.

Using **jfa**, these statistical results can be reproduced by executing the following code:

```
prior <- auditPrior(
  method = "arm", likelihood = "binomial", materiality = 0.03,
  expected = 0.01, ir = 0.6, cr = 1
```

```
)
evaluation(
  materiality = 0.03, method = "binomial",
  x = 1, n = 120, prior = prior
)
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  1 and 120
#> number of errors = 1, number of samples = 120, taint = 1, BF  =
#> 9.3941
#> alternative hypothesis: true misstatement rate is less than 0.03
#> 95 percent credible interval:
#>  0.00000000 0.03267759
#> most likely estimate:
#>  0.0088415
#> results obtained via method 'binomial' + 'prior'
```

## 8.5 Sampling Workflow

This example demonstrates how JASP for Audit simplifies the standard audit sampling workflow, also known as the "audit workflow." Let's consider an instance of the Classical audit workflow involving a fictional construction company called BuildIt, which is being audited by an external auditor from a fictional audit firm.

BuildIt diligently maintains a record of every transaction made in their general ledger throughout the year. The auditor's primary responsibility is to assess the fairness of these general ledger items, specifically to determine whether this population of general ledger items is free of material misstatement. Material misstatement indicates the presence of large enough to potentially impact decisions made based by someone relying on the financial statements. Given that BuildIt is a small company, its general ledger population comprise only 3500 items, each accompanied by a corresponding recorded book value. Before scrutinizing the population in detail, the auditor must evaluate the reliability of BuildIt's internal control systems, which processed these general ledger items, and deems them to be reasonably dependable. Therefore, the auditor determines the control risk to be "medium".

To arrive at a conclusion regarding the accuracy of BuildIt's recorded items, the auditor divides the audit workflow into the four (by now) well-known stages. Firstly, they plans the size of the sample that needs to be examined to make well-founded inferences about the entire population. Secondly, the auditor selects the necessary sampling units from the population. Thirdly, the auditor inspects the selected sample and determines the audit value (true value) of the items it contains. Lastly, the auditor employs the information gathered from the audited sample to draw inferences about the financial statements as a whole.

To initiate this workflow, the auditor begins by importing BuildIt's financial statements into JASP. The dataset containing this information is accessible in JASP

through the path: “Open” -> “Data Library” -> “7. Audit” -> “Testing for Overstatements”.

### 8.5.1 Audit Risk Model

In statistical terms, the auditor aims to make a statement with 95 percent confidence about the misstatement in the population is lower than the materiality threshold (i.e., the performance materiality). In this example, the performance materiality is set at one percent. Drawing from the previous year’s audit at BuildIt, where no misstatements were found, the auditor expects zero misstatements in the sample that will be audited. Consequently, the statistical statement can be rephrased as follows: When zero misstatements are detected in the sample, the auditor can conclude with 95 percent confidence that the misstatement in the entire population is below the one percent materiality threshold.

Typically, auditors evaluate inherent risk and control risk (sometimes analytical risk is also taken into acocunt as a fourth constituent of audit risk) using a three-point scale consisting of “Low”, “Medium”, and “High”. Different audit firms employ varying standard percentages for these risk categories. JASP for Audit defines the probabilities associated with low, medium, and high as shown in the table below. Since the auditor has conducted testing on BuildIt’s computer systems, the control risk assessment is determined to be medium (52 percent).

| Level | Inherent Risk (IR) | Control Risk (CR) | Analytical Risk (CAR) |
|---|---|---|---|
| High | 1 | 1 | 1 |
| Medium | 0.63 | 0.52 | 0.5 |
| Low | 0.40 | 0.34 | 0.25 |

### 8.5.2 Planning

The Sampling Workflow begins with the Planning stage. The auditor enters a performance materiality value of one percent and keeps the confidence level at 95 percent. To ensure the creation of an annotated report, the Explanatory text option is enabled. Moving on, the auditor selects the variables ID and bookValues and assigns them to their respective fields in the interface. Finally, the control risk assessment is adjusted from High to Medium, and the distribution used (e.g., likelihood) is set to the binomial distribution.

The default output provides information that if no misstatements are encountered in the sample, the auditor needs to audit 234 items from the population of 3500 items. This sample size will sufficiently reduce the audit risk to conclude that the population does not contain misstatements larger than one percent. The auditor proceeds to the Selection stage by clicking the “To Selection” button located in the bottom-right corner of the interface.

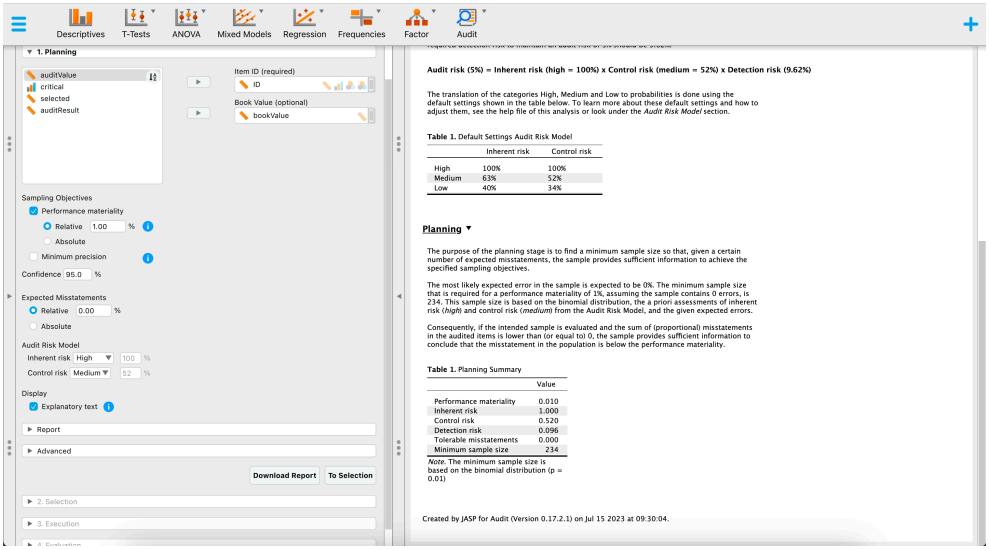



**Figure 8.10.** A snapshot of the planning stage for this example in the sampling workflow analysis in JASP for Audit.

### 8.5.3 Selection

The auditor has a variable in the "Book Value" field in the previous stage, which automatically selects the "monetary units" option for sampling units in the Selection stage. By default, the chosen sampling method is "Fixed interval sampling," and a random starting point is applied. However, these settings can be modified in the corresponding section if needed. The default output shows that 234 euros have been selected, distributed over 234 items. The selection has a total value of 114,896.23 euros, which is 8.19 percent of the total value of the population.

Upon reviewing the default output, the auditor determines that no adjustments are required in this stage. To proceed to the Execution stage, the auditor clicks the "To Execution" button located in the bottom-right corner of the interface.

### 8.5.4 Execution

During the execution phase, the auditor is prompted to provide two column names. The first column, referred to as the "Column name selection result," records the frequency of monetary units selected within a transaction for the sample. The second column, known as the "Column name audit values," allows the auditor to manually enter the audited amounts for the transactions. These column names are automatically populated, but they can be modified based on preference. To input the values for these variables in the dataset, the auditor clicks the "Fill Variables" button. Once clciked, the "Data Entry" section opens, allowing the auditor to input the audit values for the sample. However, if using an example file where the audited amounts are already known, the auditor can proceed directly to the evaluation step without entering the audit values.

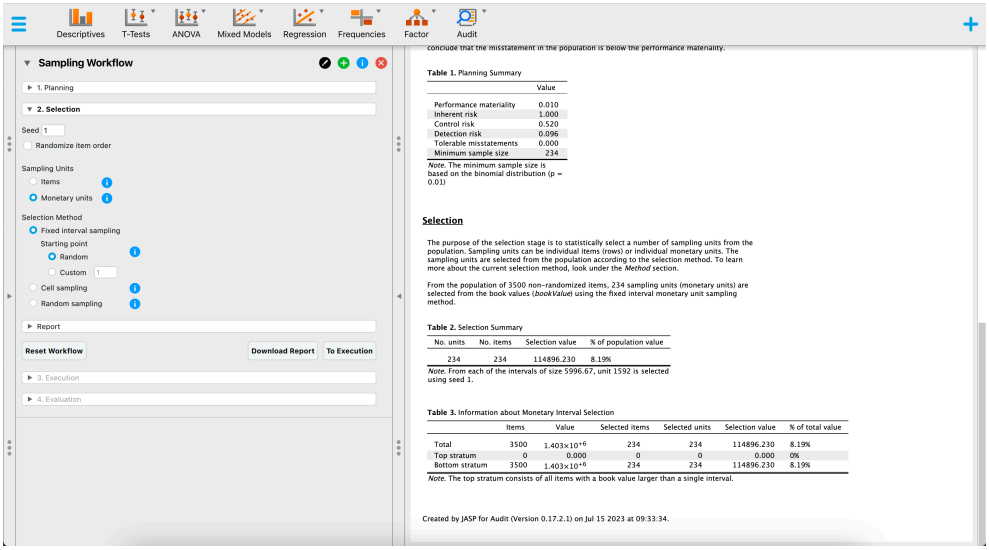



**Figure 8.11.** A snapshot of the selection stage for this example in the sampling workflow analysis in JASP for Audit.

Suppose the auditor finds a single misstatement in the sample. Specifically, item four has a recorded value of 431.87 euros and a true value of 200 euros. Hence, it is overstated by 231.87 euros.

### 8.5.5 Evaluation

In the Evaluation stage, the column created and filled with the audit values is placed in the Audit Values field within the interface. The inference is automatically conducted based on the selected options from the previous stages. The auditor has the ability to adjust the evaluation method, if applicable, and modify the numerical format of the results using the advanced options. Additionally, tables and plots that provide clear visualization of the statistical outcomes can be requested.

Based on the default output, the auditor discovers that out of the 234 items in the sample, one item contained a partial misstatement. This information yields a most likely error of 0.002 (0.23 percent), with an upper bound of 0.014 (1.37 percent) at a confidence level of 90.38 percent. The precision is calculated to be 0.011 (1.1 percent).

Since the 90.38 percent upper bound on misstatement in BuildIt's financial statements exceeds the performance materiality threshold of one percent, the auditor is unable to conclude that the population as a whole is free from misstatements below one percent. Consequently, the auditor cannot determine that the sampling risk has been sufficiently reduced to conclude that BuildIt's financial statements are free of material misstatement.

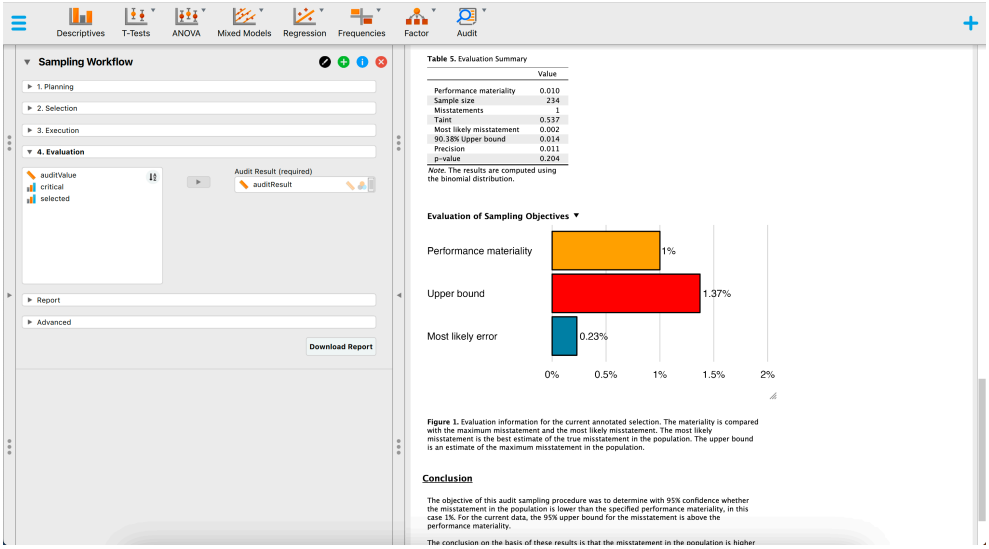



**Figure 8.12.** A snapshot of the evaluation stage for this example in the sampling workflow analysis in JASP for Audit.

# Chapter 9

# R Packages

This chapter discusses other R packages that implement statistical techniques for audit sampling. Such information may be useful for auditors who wish to explore alternative approaches or simply want to confirm the results obtained through the use of the **jfa** package. By exploring these other packages, auditors can gain a greater understanding of the various options available for audit sample planning, selection and evaluation.

## 9.1 MUS

**MUS** (Prömpers & Guimarães, 2019) is an R package providing sampling and evaluation methods to apply Monetary Unit Sampling during an audit of financial statements. The package is available via CRAN and can be downloaded by running the code below. Unlike **jfa**, the **MUS** package provides no functionality for Bayesian audit sampling.

```
install.packages("MUS")
```

To show the diffences and similarities between the use of the **MUS** package and the **jfa** package, consider a scenario in which an auditor wants to plan a monetary unit sample such that the sampling risk can be reduced below five percent. The population in this example consists of $N = 1000$ monetary units and the performance materiality is defined as ten percent (or 100 monetary units). The auditor plans the sample using an expected misstatement rate of one percent (or 10 monetary units).

To compute this sample size, the **MUS** package provides the `MUS.calc.n.conservative()` function, which takes the performance materiality in monetary units as the `tolerable.error` argument, the expected misstatements in monetary units as the `expected.error` argument, and the total number of unit in the population as the `book.value` argument. The resulting sample size is 37.

```
MUS::MUS.calc.n.conservative(tolerable.error = 100, expected.error =
 ↪  10, book.value = 1000, confidence.level = 0.95)
```

```
#> [1] 37
```

These results can be reproduced in **jfa** using the following command:

```
planning(materiality = 0.1, expected = 0.01, likelihood = "poisson")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 37
#> sample size obtained in 38 iterations via method 'poisson'
```

## 9.2 samplingbook

**samplingbook** (Manitz et al., 2021) is an R package based on the book **Stichproben: Methoden und praktische Umsetzung mit R** (Kauermann & Kuechenhoff, 2010) that focuses on survey sampling and statistical analysis of these samples. The package is available via CRAN and can be downloaded by running the code below. Unlike **jfa**, the **samplingbook** package provides limited functionality for auditing as it mainly focuses on survey sampling.

```
install.packages("samplingbook")
```

To show the diffences and similarities between the use of the **samplingbook** package and the **jfa** package, consider a scenario in which an auditor wants to evaluate a sample of items and is interested in estimating the 95 percent upper confidence bound. The population in this example consists of $N = 300$ items and the auditor has inspected a sample of $n = 100$ items, of which $k = 3$ contained a misstatement.

To compute the upper bound, the **samplingbook** package provides the `Sprop()` function, which takes the number of found misstatements in the sample as the `m` argument, the sample size as the `n` argument and the population size as the `N` argument. Note that because this function solely computes a two-sided interval for the population misstatement, the 95 percent upper bound can be obtained by using an interval of 90 percent with `level = 0.9`. The resulting upper bound is 0.07.

```
samplingbook::Sprop(m = 3, n = 100, N = 300, level = 0.90)
#>
#> Sprop object: Sample proportion estimate
#> With finite population correction: N = 300
#>
#> Proportion estimate:  0.03
#> Standard error:  0.014
#>
#> 90% approximate confidence interval:
#>  proportion: [0.007,0.053]
#>  number in population: [3,15]
#> 90% exact hypergeometric confidence interval:
#>  proportion: [0.01,0.07]
```

```
#>  number in population: [3,21]
```

These results can be reproduced in **jfa** using the following command:

```
evaluation(x = 3, n = 100, N.units = 300, method = "hypergeometric",
 ↪  alternative = "less", conf.level = 0.95)
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  3 and 100
#> number of errors = 3, number of samples = 100, taint = 3
#> 95 percent confidence interval:
#>  0.00000000 0.06666667
#> most likely estimate:
#>  0.03
#> results obtained via method 'hypergeometric'
```

## 9.3 audit

**audit** (Meeden, 2021) is an R package based on the paper by Meeden & Sargent (2007) that can be used to find an upper bound for the total amount of overstatement of assets in a set of accounts and estimating the amount of sales tax owed on a collection of transactions. The package is available via CRAN and can be downloaded by running the code below. Unlike **jfa**, The **audit** package provides limited functionality for evaluating audit samples and has no functionality for planning and selection.

```
install.packages("audit")
```

# References


American Institute of Certified Public Accountants (AICPA). (2016a). Appendix A: Attributes statistical sampling tables. In *Audit guide: Audit sampling*. https://doi.org/10.1002/9781119448617.app1

American Institute of Certified Public Accountants (AICPA). (2016b). Appendix C: Monetary unit sampling tables. In *Audit guide: Audit sampling*. https://doi.org/10.1002/9781119448617.app3

Bickel, P. J. (1992). Inference and auditing: The Stringer bound. *International Statistical Review/Revue Internationale de Statistique*, 197–209. https://doi.org/10.2307/1403650

Broeze, G. B. (2006). *Validation of risk assessment in auditing* [PhD thesis, Vrije Universiteit]. https://research.vu.nl/ws/portalfiles/portal/42174612/complete+dissertation.pdf

Chang, W. (2022). *R Graphics Cookbook*. O'Reilly. https://r-graphics.org/

Clopper, C. J., & Pearson, E. S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, *26*(4), 404–413. https://doi.org/10.2307/2331986

Derks, K., de Swart, J., van Batenburg, P., Wagenmakers, E., & Wetzels, R. (2021). Priors in a Bayesian audit: How integration of existing information into the prior distribution can improve audit transparency and efficiency. *International Journal of Auditing*, *25*(3), 621–636. https://doi.org/10.1111/ijau.12240

Derks, K., de Swart, J., Wagenmakers, E., & Wetzels, R. (2022a). An impartial Bayesian hypothesis test for audit sampling. *PsyArXiv*. https://doi.org/10.31234/osf.io/8nf3e

Derks, K., de Swart, J., Wagenmakers, E., & Wetzels, R. (2022b). The Bayesian approach to audit evidence: Quantifying statistical evidence using the Bayes factor. *PsyArXiv*. https://doi.org/10.31234/osf.io/kzqp5

Derks, K., de Swart, J., Wagenmakers, E., Wille, J., & Wetzels, R. (2021). JASP for Audit: Bayesian tools for the auditing practice. *Journal of Open Source Software*, *6*(68), 2733.

Dyer, D., & Pierce, R. L. (1993). On the choice of the prior distribution in hypergeometric sampling. *Communications in Statistics - Theory and Methods*, *22*(8), 2125–2146. https://doi.org/10.1080/03610929308831139

Edwards, W., Lindman, H., & Savage, L. J. (1963). Bayesian statistical inference for psychological research. *Psychological Review*, *70*(3), 193. https://doi.org/10.1037/h0044139

Grolemund, G. (2014). *Hands-On Programming with R*. https://rstudio-education.

github.io/hopr/

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (Vol. 2). Springer.

Hoekstra, R., Morey, R. D., Rouder, J. N., & Wagenmakers, E. (2014). Robust misinterpretation of confidence intervals. *Psychonomic Bulletin & Review*, *21*, 1157–1164.

International Auditing and Assurance Standards Board (IAASB). (2018). ISA 530: Audit sampling. In *International standards on auditing (ISA)*.

JASP Team. (2023). *JASP (Version 0.18.1)[Computer software]*. https://jasp-stats.org/

Jeffreys, H. (1939). *Theory of probability*. Clarendon Press.

Jeffreys, H. (1948). *Theory of probability*. Clarendon Press.

Jeffreys, H. (1961). *Theory of probability*. Clarendon Press.

Kauermann, G., & Kuechenhoff, H. (2010). *Stichproben: Methoden und praktische umsetzung mit r*. Springer-Verlag.

Lin, J. (2021). *Audit Analytics with R*. https://auditanalytics.jonlin.ca/

Love, J., Selker, R., Marsman, M., Jamil, T., Dropmann, D., Verhagen, J., Ly, A., Gronau, Q. F., Šmíra, M., Epskamp, S., Matze, D., Wild, A., Knight, P., Rouder, J. N., Morey, R. D., & Wagenmakers, E. (2019). JASP: Graphical statistical software for common statistical designs. *Journal of Statistical Software*, *88*, 1–17.

Manitz, J., Hempelmann, M., Kauermann, G., Kuechenhoff, H., Shao, S., Oberhauser, C., Westerheide, N., & Wiesenfarth, M. (2021). *Samplingbook: Survey sampling procedures*. https://CRAN.R-project.org/package=samplingbook

Meeden, G. (2021). *Audit: Bounds for accounting populations*. https://CRAN.R-project.org/package=audit

Meeden, G., & Sargent, D. (2007). Some Bayesian methods for two auditing problems. *Communications in Statistics —- Theory and Methods*, *36*(15), 2727–2740. https://doi.org/10.1080/03610920701386802

Pearson, K. (1948). *Tables of the incomplete beta-function*. Biometrika.

Prömpers, H., & Guimarães, A. (2019). *MUS: Monetary unit sampling and estimation methods, widely used in auditing*. https://CRAN.R-project.org/package=MUS

R Core Team. (2022). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rouder, J. N. (2014). Optional stopping: No problem for Bayesians. *Psychonomic Bulletin & Review*, *21*, 301–308. https://doi.org/10.3758/s13423-014-0595-4

Steele, A. (1992). *Audit risk and audit evidence: The Bayesian approach to statistical auditing*. London Academic Press.

Stewart, T. (2012). *Technical notes on the AICPA audit guide audit sampling*. 5–8.

Stringer, K. W. (1963). Practical aspects of statistical sampling in auditing. *Proceedings of the Business and Economic Statistics Section*, 405–411.

Wickam, H., & Brian, J. (2022). *R Packages*. https://r-pkgs.org/

Wickam, H., & Grolemund, G. (2017). *R for Data Science*. https://r4ds.had.co.nz/

Wilke, C. O. (2022). *Fundamentals of Data Visualization*. https://clauswilke.com/dataviz/

# Appendix: R Tutorial

R (R Core Team, 2022) is a programming language and software environment for statistical computing and graphics. It is widely used among statisticians and data scientists for data analysis and data visualization. R has a large and active community of users and, as a result, there are many community-made resources available for learning and using R.

In an audit context, R can be used to analyze and visualize large datasets, allowing auditors to identify trends and anomalies in the data. R is particularly useful for performing statistical analysis and testing hypotheses, which can be employed in verifying the accuracy and reliability of financial statements. R can also be used to automate certain audit procedures, reducing the time and effort required to manually review and analyze large amounts of data. Additionally, R allows auditors to easily share their work with others through the use of code and reproducible reports, enabling more efficient and collaborative audit processes. A useful resource that discusses relevant applications of R in the audit can be found in Lin (2021).

This chapter provides a short introduction to working with R but it is not a full programming course in R. Furthermore, if you want to dive deeper into the nooks and crannies of this language, the author recommends reading Chang (2022), Wickam & Grolemund (2017), Grolemund (2014) and Wilke (2022). These books are free to read online and discuss various best practices for using R, including examples in code that can be easily reproduced.

## Calculations

One of the basic features of R is its ability to perform calculations. In R, basic calculations work by using the standard arithmetic operators such as `+` for addition, `-` for subtraction, `*` for multiplication, and `/` for division. For example, if you want to calculate $2 + 3$, you would type in `2 + 3` and R will return the result of 5.

```
2 + 3
#> [1] 5
```

R also allows for more advanced calculations such as exponentiation using the `^` operator, and square roots using the `sqrt()` function. For example, to calculate the square root of 9, you would type in `sqrt(9)` and R will return the result of 3.


**Figure 9.1.** R can be a powerful tool in a business context. Image available under a CC-BY-NC 4.0 license.

```
sqrt(9)
#> [1] 3
```

You can also use parentheses to specify the order of operations in your calculations. For example, if you want to calculate (2 + 3) * 4, you would type in `(2 + 3) * 4` to get the result of 20.

```
(2 + 3) * 4
#> [1] 20
```

Overall, basic calculations in R are similar to those in other programming languages and follow the standard order of operations.

## Vectors

In R, vectors are one-dimensional arrays of data that can hold numeric, character, or logical values. Vectors can be created using the `c()` function, which stands for concatenate. For example, to create a numeric vector, you can use the following code:

```
x <- c(1, 2, 3, 4, 5)
```

To create a character vector, you can use quotes around the values:

```
y <- c("apple", "banana", "orange")
```

To create a logical vector, you can use the logical values `TRUE` and `FALSE`:

```
z <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

Vectors can be indexed using square brackets and a numeric value. For example, to access the second element of the vector `x`, you can use the following code:

```
x[2]
#> [1] 2
```

Vectors can also be subsetted using a logical vector. For example, to get all elements of the vector `x` that are greater than 3, you can use the following code:

```
x[x > 3]
#> [1] 4 5
```

Vectors can also be modified using indexing and assignment. For example, to change the third element of the vector `x` to 6, you can use the following code:

```
x[3]
#> [1] 3
x[3] <- 6
x[3]
#> [1] 6
```

R has many built-in functions for performing mathematical operations on vectors. For example, you can use the `mean()` function to calculate the average of a vector of numbers, or we can use the `length()` function to calculate the number of elements in a vector:

```
mean(x)
#> [1] 3.6
length(y)
#> [1] 3
```

Overall, vectors are a useful data structure in R for storing and manipulating data.

## Matrices

In R, a matrix is a two-dimensional collection of values that are arranged in rows and columns. You can create a matrix using the `matrix()` function. For example:

```
m <- matrix(1:9, nrow = 3, ncol = 3)
m
#>      [,1] [,2] [,3]
#> [1,]    1    4    7
#> [2,]    2    5    8
#> [3,]    3    6    9
```

This creates a 3x3 matrix with the values 1, 2, 3 in the first column, 4, 5, 6 in the second column, and 7, 8, 9 in the third column.

You can also create a matrix by combining several vectors using the `cbind()` or `rbind()` functions. For example:

```
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
```

```
v3 <- c(7, 8, 9)
m <- cbind(v1, v2, v3)
m
#>      v1 v2 v3
#> [1,]  1  4  7
#> [2,]  2  5  8
#> [3,]  3  6  9
```

This creates a matrix with the same values as before, but the columns are created by binding the vectors together.

You can access the elements of a matrix using the square bracket notation. For example, to access the element in the second row and third column of `m`, you would use the following code:

```
m[2, 3]
#> v3
#>  8
```

You can also use the `dim()` function to get the dimensions of a matrix, and the `colnames()` and `rownames()` functions to get the names of the columns and rows, respectively.

There are many other functions and operations available for working with matrices in R, including mathematical operations such as matrix multiplication and inversion.

## Data Frames

In R, a data frame is a two-dimensional table of data with rows and columns. Each row represents a single observation or record, and each column represents a particular variable or attribute. Data frames are similar to a spreadsheet in Excel or a table in a database. Each column in a data frame can have a different data type, such as numerical, character, or logical. The data in each row must match the data type of the corresponding column.

To create a data frame in R, you can use the `data.frame()` function and pass in the data you want to include in the data frame as arguments. For example:

```
df <- data.frame(x = c(1, 2, 3), y = c(4, 5, 6))
```

This will create a data frame with two columns, `x` and `y`, and three rows of data. You can access the data in a data frame using indexing and subsetting. For example, to access the first row of the data frame, you can use the following command:

```
df[1, ]
#>   x y
#> 1 1 4
```

To access a specific column, you can use the `$` operator (or the index):

```
df$x
#> [1] 1 2 3
df[, 1]
#> [1] 1 2 3
df[, "x"]
#> [1] 1 2 3
df[["x"]]
#> [1] 1 2 3
```

You can also use functions like `head()` and `tail()` to view the first or last few rows of a data frame. Data frames also have several built-in functions that allow you to manipulate and analyze the data. For example, you can use the `summarize()` function to calculate summary statistics for each column, or the `group_by()` function to group the data by a specific variable and apply a function to each group.

# Data Sets

When working with data, you will need to load the data file into your R session. How this is done depends on the type of data file that you want to read.

## Built-in Data

Data that is included in an R package can be loaded via the `data()` function. For example, to load the `BuildIt` data set that is included in the **jfa** package, you can run the following R code. Note that this requires that the package is loaded in the R session via a call to `library()`.

```
data(BuildIt)
```

## Loading Data from a CSV File

A commonly used data type is a `.csv` file. You can load this type of files via the `read.csv()` function. For example, if the file `example.csv` is in the current working directory, you can load it by running:

```
read.csv("example.csv")
```

## Loading Data from an Excel File

Another commonly used data type are Excel files. You can load this type of files via the `read_excel()` function from the `readxl` package. For this to work, you should first install this package using the `install.packages()` command and load it into the R session using a call to `library()`. For example, if the file `example.xlsx` is in the current working directory, and the data you want to load is on the first worksheet, you can load it by running:

```
install.packages("readxl")
library(readxl)
read_excel("example.csv", sheet = 1)
```

## Practical Exercises

1. Compute the square root of 81 and store the result in a variable called `t1`.
2. Compute 81 to the power a half and store the result in a variable called `t2`.
3. Use the `==` operator to check whether the content of `t1` and `t2` is the same.
4. Use the `c()` function (or `:`) to create the following vector: `-2 -1 0 1 2 3 4 5 6 7 8`.
5. Find out the length of the vector created in exercise 4.
6. Find out the mean of the vector created in exercise 4.

## Answers to the Exercises

1. Assigning a variable can be achieved via the `<-` operator, while the square root is computed via the `sqrt()` function.

```
t1 <- sqrt(81)
```

2. The square root can also be computed using the power operator `^`.

```
t2 <- 81^0.5
```

3. The `==` operator can be used to check if the contents of `t1` and `t2` are the same.

```
t1 == t2
#> [1] TRUE
```

4. There are two main (and many more other) ways of creating this vector:

```
c(-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8)
#>  [1] -2 -1  0  1  2  3  4  5  6  7  8
```

or:

```
-2:8
#>  [1] -2 -1  0  1  2  3  4  5  6  7  8
```

5. The length of any vector can be obtained using the `length()` function.

```
length(-2:8)
#> [1] 11
```

6. The average of any vector can be obtained using the `mean()` function.

```
mean(-2:8)
#> [1] 3
```

This book, **Statistical Audit Sampling with R**, is intended as a practical guide for auditors who wish to employ probability theory in their audit sampling activities. While the focus of this book is exclusively on audit sampling, it aims to discuss the topic from both the classical (frequentist) perspective and the Bayesian perspective. By examining the subject through these two lenses, the book explains the statistical theory behind commonly used audit sampling procedures and demonstrates how to perform these procedures in accordance with international auditing standards, using the **jfa** R package, in a statistically sound manner.

Furthermore, the book serves as a user manual for **jfa** and **JASP for Audit**, which is a module for the free and open-source statistical software program **JASP** that integrates the functionality of the **jfa** package and offers a user-friendly graphical interface that caters specifically to statistical auditing (**https://jasp-stats.org**).