Statistical Audit Sampling with R

Koen Derks

12/19/22

Table of contents

Welcome

This book, **Statistical Audit Sampling with R**, is intended as a practical guide for auditors who wish to utilize state-of-the-art statistical methodology in their audit sampling activities. While the focus of this book is exclusively on audit sampling, it aims to examine 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.

Preface

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. In my opinion, 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 (one notable exception is Stewart (2012)). 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. 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 the topic of audit sampling from both a classical (frequentist) and a Bayesian perspective. By utilizing 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 organization of this book is as follows: Chapter 1 provides a foundational introduction to the R programming language. Chapter 2 addresses the fundamental statistical theory relevant to audit sampling. Chapters 3, 4, 5, and 5 delve more deeply into the use of statistical methods for the planning, selection, and evaluation of audit samples. Finally, Chapter 7 considers other software implementations of audit sampling utilizing R.

1 Introduction to R

R 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 and has a large and active community of users. As a result, there are many community resources available for learning and using R. This book will provide a short introduction to R but does not offer a full course on it. If you want to dive deeper into R, I recommend reading Chang (2022), Wickam and Grolemund (2017), Grolemund (2014) and Wilke (2022).

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 resource that discusses these applications of R in the audit can be found in Lin (2021).

We will first provide a basic introduction to R and the functionality that is required for reproducing the audit sampling examples in this book.

1.1 Basics

One of the basic features of R is its ability to manipulate data. 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.

```
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.

1.1.1 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 also has functions for performing mathematical operations on data. For example, we 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.

1.1.2 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.

1.1.3 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
```

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.

1.2 Packages

In addition to these basic features, R has many packages and libraries that extend its capabilities and provide additional functions and tools for data analysis and visualization. A package is a small bundle of code that an R user (or group of users) wrote and uploaded to a central server for everybody to access, download and use. Some popular packages include `dplyr` for data manipulation, `ggplot2` for data visualization, and `caret` for machine learning. With its versatility and robust community, R is a valuable tool for anyone working with data.

> If you want to know more about making your own package, read Wickam and Brian (2022).

1.2.1 Installing a Package

In many cases, you will download an R package from the CRAN server. This can be done via the `install.packages()` function by providing the package name in quotes. For example, an important package for data visualization is `ggplot2`. To install this package, you can simply run:

```
install.packages("ggplot2")
```

To illustrate its concepts and ideas, this book heavily draws from the **jfa** package, an R package for statistical auditing, which can also be downloaded from CRAN. Before running the examples in this book, you should install this package by running the following command in R:

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

1.2.2 Loading a Package

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 can load the **jfa** package with:

```
library(jfa)
```

> If you want a deeper understanding of how the **jfa** package works, or want to look at the source code of the package, see the package website.

1.2.3 Updating a Package

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")
```

1.3 Loading Data

1.3.1 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")
```

1.3.2 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:

```r
install.packages("readxl")
library(readxl)
read_excel("example.csv", sheet = 1)
```

2 Background Information

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 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 say about statistical sampling.

2.1 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.

2.2 Important Concepts

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

2.2.1 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 financical 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.

2.2.2 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.

2.2.3 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.

2.2.4 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.

$$\text{Audit risk} = \text{Inherent risk} \times \text{Control risk} \times \text{Detection risk} \tag{2.1}$$

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:

$$\text{Detection risk} = \frac{\text{Audit risk}}{\text{Inherent risk} \times \text{Control risk}} \tag{2.2}$$

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

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%
- Medium: 60%
- Low: 50%

If an auditor is working with an audit risk of 5%, and judges inherent and control risk to both be medium, the sampling risk can be calculated as:

$$\frac{0.05}{0.6 \times 0.6} = 0.139 \tag{2.3}$$

2.2.5 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.

2.2.6 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	θ	
Performance materiality	θ_{max}	`materiality`
Expected deviation rate	θ_{exp}	`expected`
Type-I error probability	α	`1 - conf.level`
Type-II error probability	β	
Population size	N	`N.units`
Population misstatements	K	
Sample size	n	`n`
Observed misstatements	k	`x`

2.3 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 statistical models are purely objective and that data is generated by a well-defined process, which can be described by a set of probabilistic assumptions. The philosophy behind frequentist statistics is that statistical estimates should be based on the frequency of events in a population, rather than on subjective or personal beliefs. This approach is particularly useful for making predictions or making 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 informed decisions based on data.

2.3.1 Parameter 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 a larger 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 precision, which can be quantified through statistical measures such as confidence bounds or intervals.

2.3.2 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 α, we reject the hypothesis that the population is materially misstated and conclude that it is not materially misstated.

2.3.3 Example

As an example, the `binom.test()` function in R can be used to test if a population contains less than 3 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 5%, 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 test if the proportion of misstatements in the sample is less than the hypothesized proportion.

```r
binom.test(x = 0, n = 100, p = 0.03, alternative = "less", conf.level = 0.95)
#>
#>  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 most likely misstatement in the population is displayed under `sample estimates` and is 0%. The 95% upper confidence bound for the estimate of the population misstatement is displayed under `95 percent confidence interval` and is 2.95%. The p value is shown to be 0.04755. Since the p value is lower than the specified sampling risk α, the auditor can reject the hypothesis of material misstatement.

2.4 Bayesian Inference

Bayesian inference is based on the idea that the parameters in a statistical model are not fixed but uncertain. In this approach, the parameter is onsidered 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).

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \tag{2.4}$$

Bayesian statistics is a more nuanced approach that allows for more efficiency in statistical audit sampling, 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.

2.4.1 Parameter 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:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{2.5}$$

where $p(\theta|y)$ is the posterior probability of the population parameter θ given the sample data y, $p(y|\theta)$ is the likelihood of the sample data given θ, $p(\theta)$ is the prior probability of θ, 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:

$$p(\theta|y) \propto p(y|\theta) \times p(\theta) \tag{2.6}$$

2.4.2 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.

The **jfa** package can help us to calculate Bayes factors for and against the hypothesis that the financial statements are represented fairly.

3 Planning a Sample

This chapter discusses the most commonly used approaches to plan a statistical audit sample.

One of the key considerations in audit sampling is determining the minimum sample size required to achieve a desired level of assurance or precision. In this chapter, we will discuss how to use three standard likelihoods to plan a minimum sample size for audit sampling: the hypergeometric likelihood, the binomial likelihood and the Poisson likelihood.

3.1 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 minimum 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.

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.

3.2 The Hypergeometric Likelihood

Let's consider how to use the hypergeometric likelihood to calculate the minimum sample size needed to achieve a desired level of assurance. 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. 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:

$$p(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}, \tag{3.1}$$

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 linear extrapolation of the assumed misstatement rate in the population θ_{max} to the total population:

$$K = \theta_{max} N. \tag{3.2}$$

3.2.1 Classical planning

Concretely, the following statistical model is assumed:

$$k \sim \text{Hypergeometric}(n, N, K) \tag{3.3}$$

Given a desired misstatement tolerance θ_{max}, we can solve for the minimum sample size n needed to achieve this assurance level. In **jfa**, this sample size can be calculated using the `planning()` function. For example, if we want to achieve an assurance level of 95% ($\alpha = 0.05$) 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 zero expected misstatements in the sample is $n = 94$.

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

The `dhyper()` function calculates the probability of observing k missatements in a sample of n items given an assumed misstatement probability. The sample size of 94 can be confirmed by checking that 94 is the minimum integer that results in less than 5% probability of finding 0 misstatements if the population misstatement is truly 3%.

```
K <- ceiling(0.03 * 1000)
dhyper(x = 0, m = K, n = 1000 - K, k = 93) < 0.05 # 93: Not sufficient
#> [1] FALSE
dhyper(x = 0, m = K, n = 1000 - K, k = 94) < 0.05 # 94: Sufficient
#> [1] TRUE
```

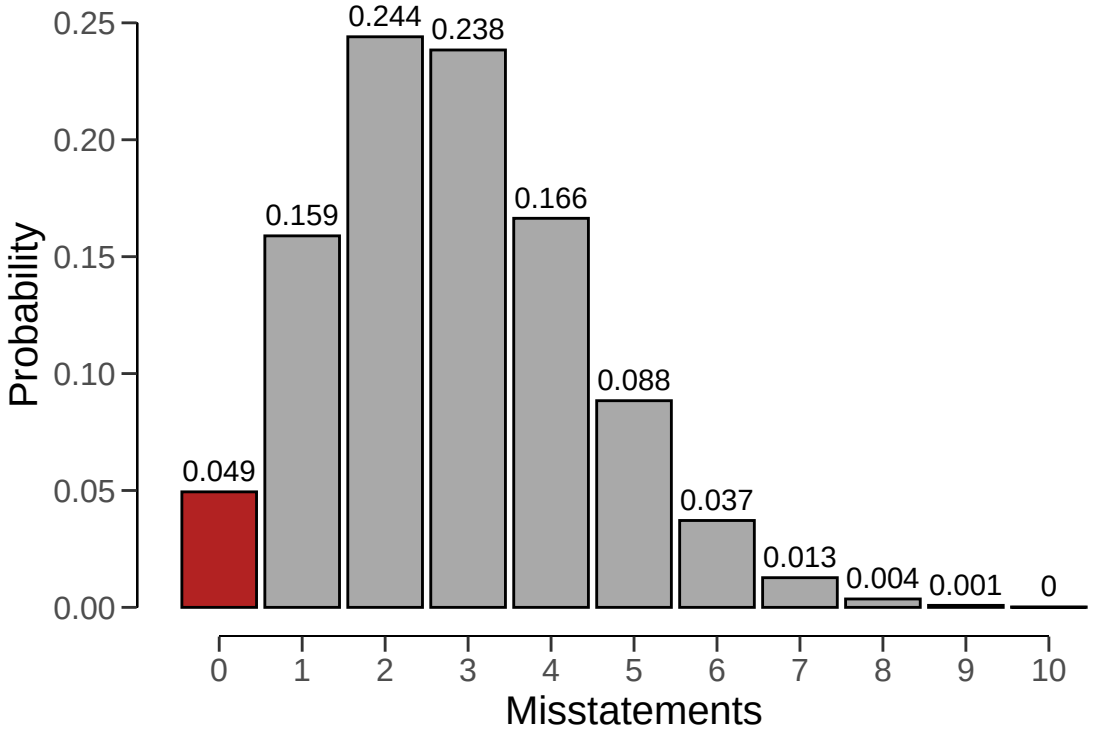
We can make this visually intuitive by showing the hypergeometric($k \mid 94, 1000, 30$) distribution and highlighting the probability for $k = 0$. This probability should be lower than the required sampling risk $\alpha = 0.05$.

As another example, if we want to achieve an assurance level of 95% ($\alpha = 0.05$) 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 $n = 147$.

```
planning(materiality = 0.03, expected = 1, conf.level = 0.95,
         likelihood = "hypergeometric", N.units = 1000)
#>
#>  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 5% probability of finding 0 or 1 misstatements if the population misstatement is truly 3%.

```
sum(dhyper(x = 0:1, m = K, n = 1000 - K, k = 146)) < 0.05 # 146: Not sufficient
#> [1] FALSE
sum(dhyper(x = 0:1, m = K, n = 1000 - K, k = 147)) < 0.05 # 147: Sufficient
```




```
#> [1] TRUE
```

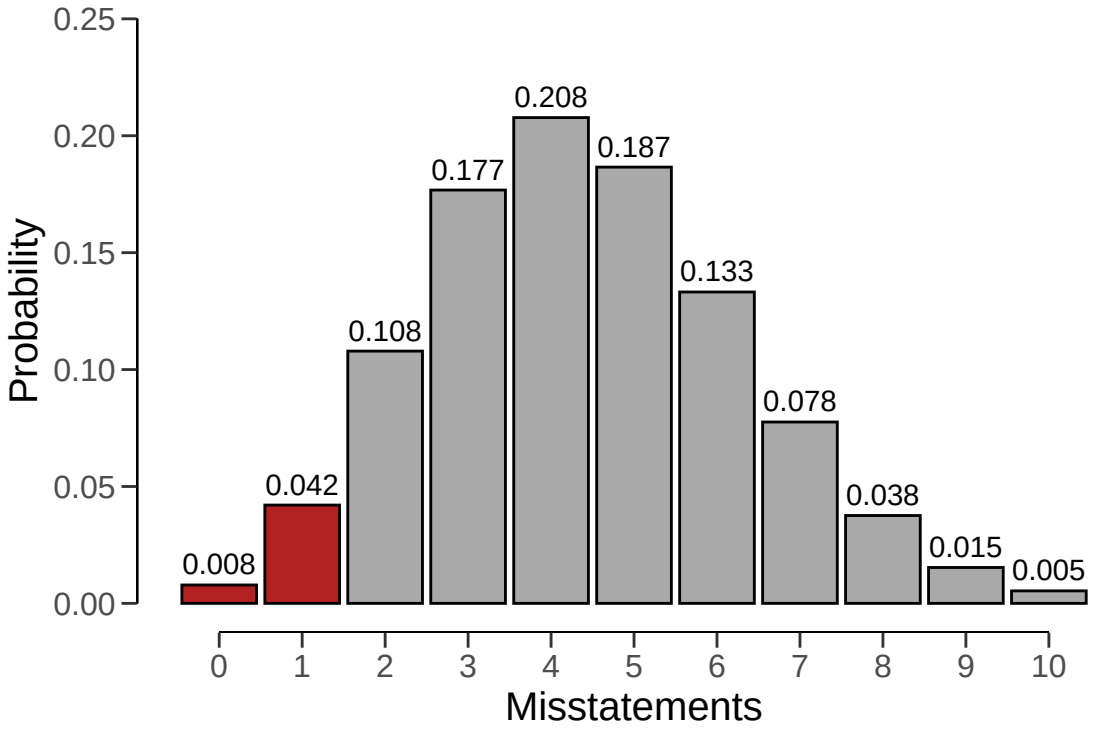
Like before, we can make this visually intuitive by showing the hypergeometric($k \mid 147, 1000, 30$) distribution and highlighting the probabilities for $k = 0$ and $k = 1$. The sum of these probabilities should be lower than the required sampling risk $\alpha = 0.05$.

3.2.2 Bayesian Planning

Performing Bayesian planning with the hypergeometric likelihood (Dyer and Pierce 1993) requires that you specify a prior distribution for the parameter θ. 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). Concretely, this means that the following statistical model is assumed:

$$k \sim \text{Hypergeometric}(n, N, K) \tag{3.4}$$
$$K \sim \text{Beta-binomial}(N, \alpha, \beta) \tag{3.5}$$



The beta-binomial prior distribution is conjugate to the hypergeometric likelihood (see this list of conjugate priors), which means that the posterior distribution of K can be determined analytically. For example, if the prior distribution for K is beta-binomial(N, α, β) and the auditor has observed a sample of n items containing k misstatements, the posterior distribution for K is beta-binomial($N - n$, $\alpha + k$, $\beta + n - k$).

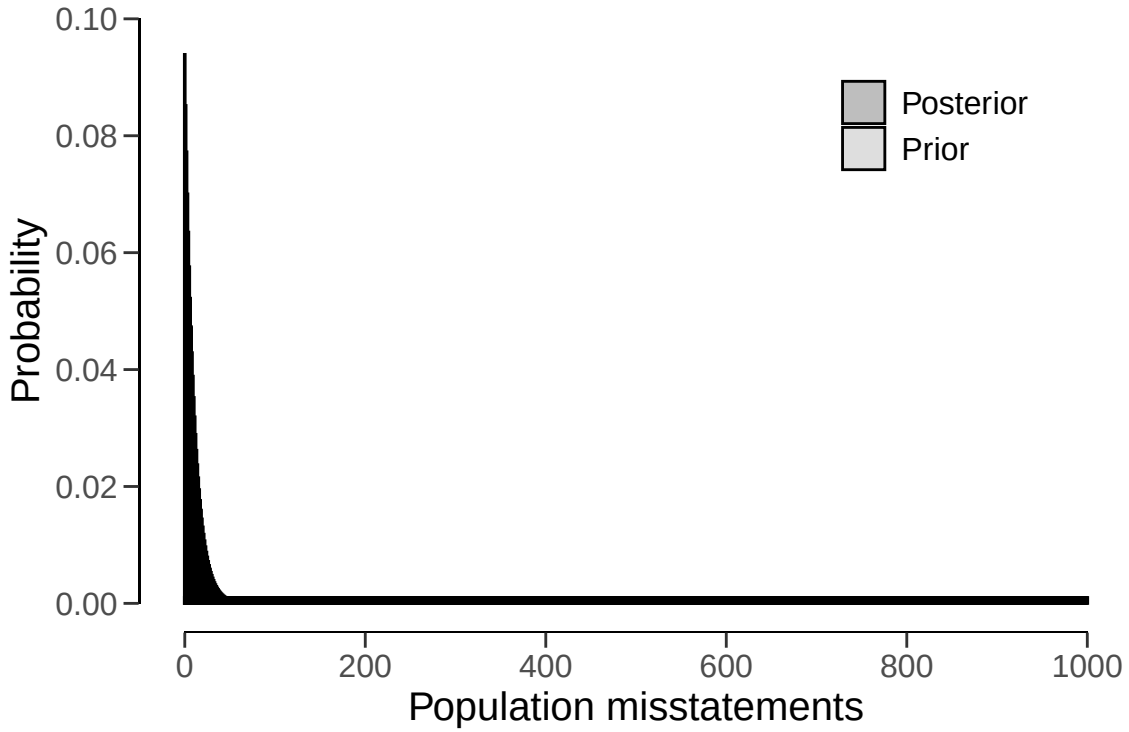
For example, the command below uses a default beta-binomial(N, 1, 1) prior distribution to plan the sample, since `planning()` is given the hypergeometric likelihood. If we want to achieve an assurance level of 95% ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$ in a population of $N = 1000$ items, then the required sample size under the assumption of zero expected misstatements in the sample is $n = 93$.

```
plan <- planning(materiality = 0.03, expected = 0, conf.level = 0.95,
                 likelihood = "hypergeometric", N.units = 1000, prior = TRUE)
summary(plan)
#>
#>  Bayesian Audit Sample Planning Summary
#>
```

```
#> Options:
#>   Confidence level:             0.95
#>   Population size:              1000
#>   Materiality:                  0.03
#>   Hypotheses:                   H : θ > 0.03 vs. H : θ < 0.03
#>   Expected:                     0
#>   Likelihood:                   hypergeometric
#>   Prior distribution:           beta-binomial(N = 1000,   = 1,   = 1)
#>
#> Results:
#>   Minimum sample size:          93
#>   Tolerable errors:             0
#>   Posterior distribution:       beta-binomial(N = 907,   = 1,   = 94)
#>   Expected most likely error:   0
#>   Expected upper bound:         0.029
#>   Expected precision:           0.029
#>   Expected BF :                620.58
```

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

```
plot(plan)
```



The hypergeometric likelihood does not allow for non-conjugate prior distributions to be used as a prior.

3.3 The Binomial Likelihood

Let's consider how to use the binomial likelihood to calculate the minimum sample size needed to achieve a desired level of assurance. 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. 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: θ 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:

$$p(k; n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k}, \tag{3.6}$$

where k is the number of misstatements in the sample, n is the sample size and θ is the misstatement rate expected in the sample.

3.3.1 Classical Planning

Concretely, the following statistical model is assumed:

$$k \sim \text{Binomial}(n, \theta_{max}) \tag{3.7}$$

Given a desired misstatement tolerance θ_{max}, we can solve for the minimum sample size n needed to achieve the desired assurance 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:

$$n = \lceil \frac{\ln(\alpha)}{\ln(1-\theta_{max})} \rceil. \tag{3.8}$$

$\lceil ... \rceil$ is the ceiling function. Hence, $\lceil 1.2 \rceil = 2$.

For example, if we want to achieve an assurance level of 95% ($\alpha = 0.05$) 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 `planning()` function.

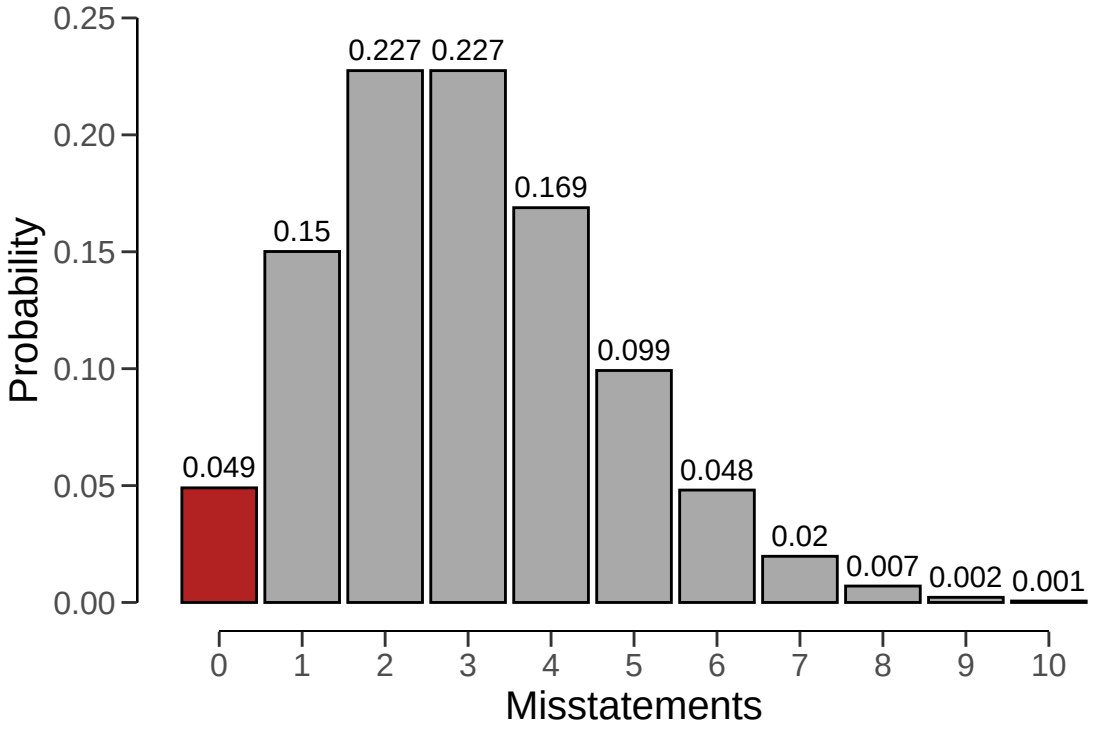
```
planning(materiality = 0.03, expected = 0, conf.level = 0.95,
         likelihood = "binomial")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 99
#> sample size obtained in 100 iterations via method 'binomial'
```

The `dbinom()` function calculates the probability of observing k missatements in a sample of n items given an assumed misstatement probability. The sample size of 99 can be confirmed

by checking that 99 is the minimum integer that results in less than 5% probability of finding 0 misstatements if the population misstatement is truly 3%.

```
dbinom(x = 0, size = 98, prob = 0.03) < 0.05 # 98: Not sufficient
#> [1] FALSE
dbinom(x = 0, size = 99, prob = 0.03) < 0.05 # 99: Sufficient
#> [1] TRUE
```

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



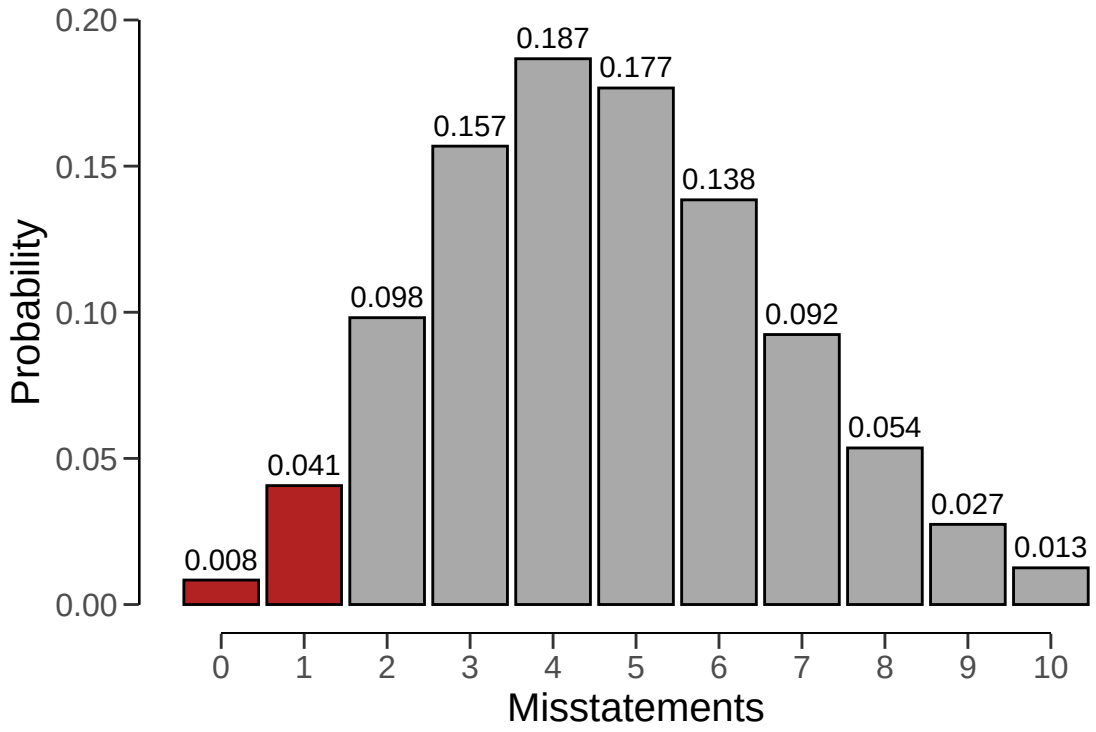
However, if the number of expected misstatements in the sample is non-zero, it becomes more difficult to solve the formula for n. Hence, we can iteratively try every value of n and return the smallest integer that satisfies the sampling objectives. In **jfa**, this can be done by adjusting the `expected` argument in the `planning()` function. For example, if we want to achieve an assurance level of 95% ($\alpha = 0.05$) 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 $n = 157$.

```
planning(materiality = 0.03, expected = 1, conf.level = 0.95,
         likelihood = "binomial")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 157
#> sample size obtained in 156 iterations via method 'binomial'
```

Once again, the sample size of 157 can be confirmed by checking that 157 is the minimum integer that results in less than 5% probability of finding 0 or 1 misstatements if the population misstatement is truly 3%.

```
sum(dbinom(x = 0:1, size = 156, prob = 0.03)) < 0.05 # 156: Not sufficient
#> [1] FALSE
sum(dbinom(x = 0:1, size = 157, prob = 0.03)) < 0.05 # 157: Sufficient
#> [1] TRUE
```

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



3.3.2 Bayesian Planning

Performing Bayesian planning with the binomial likelihood requires that you specify a prior distribution for the parameter θ. 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). Concretely, this means that the following statistical model is assumed:

$$k \sim \text{Binomial}(n, \theta) \tag{3.9}$$
$$\theta \sim \text{Beta}(\alpha, \beta) \tag{3.10}$$

> The beta prior distribution is conjugate to the binomial likelihood (see this list of conjugate priors), which means that the posterior distribution of θ can be determined analytically. For example, if the prior distribution is beta(α, β) and the auditor has observed a sample of n items containing k misstatements, the posterior distribution for θ is beta($\alpha + k$, $\beta + n - k$).

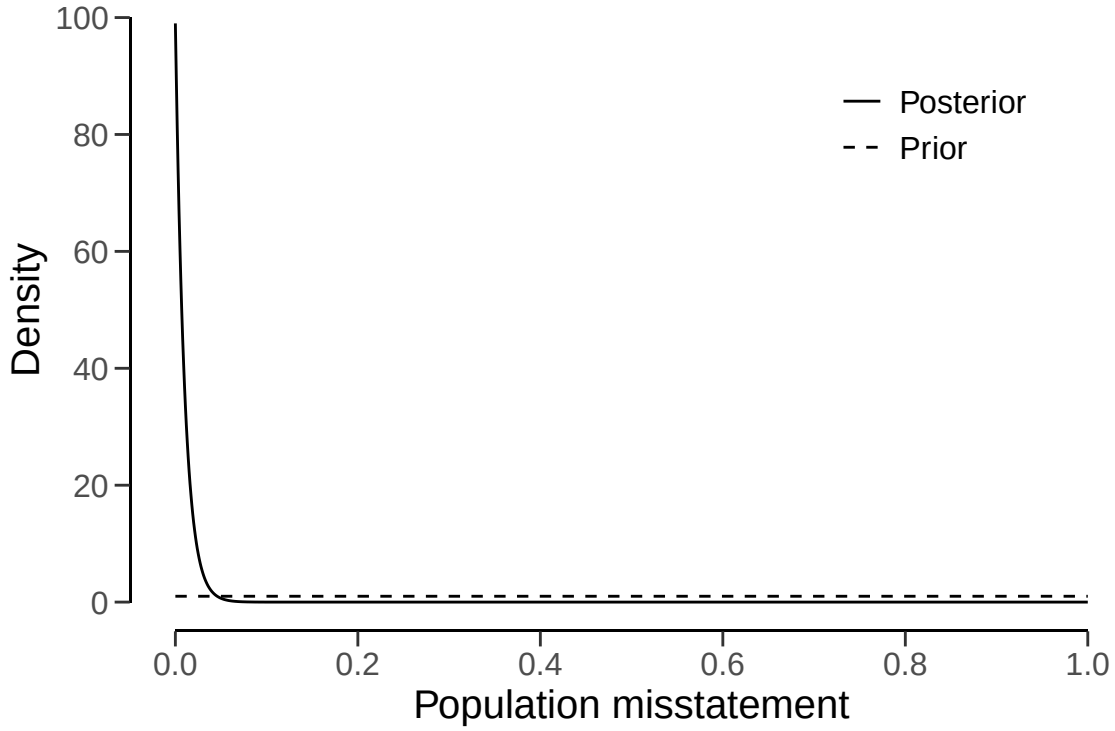
For example, the command below uses a default beta($\alpha = 1$, $\beta = 1$) prior distribution to plan the sample, since `planning()` is given the binomial likelihood. If we want to achieve an assurance level of 95% ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the 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)
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(  = 1,   = 1)
#>
#> Results:
#>   Minimum sample size:           98
#>   Tolerable errors:              0
```

```
#>   Posterior distribution:        beta(  = 1,   = 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. The expected posterior distribution is the posterior distribution that would occur if you actually observed the planned sample containing the expected misstatements.

```
plot(plan)
```



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. Concretely, this means that the following statistical model is assumed:

$$k \sim \text{Binomial}(n, \theta) \tag{3.11}$$
$$\theta \sim \text{Normal}(\mu = 0, \sigma = 0.05) \tag{3.12}$$

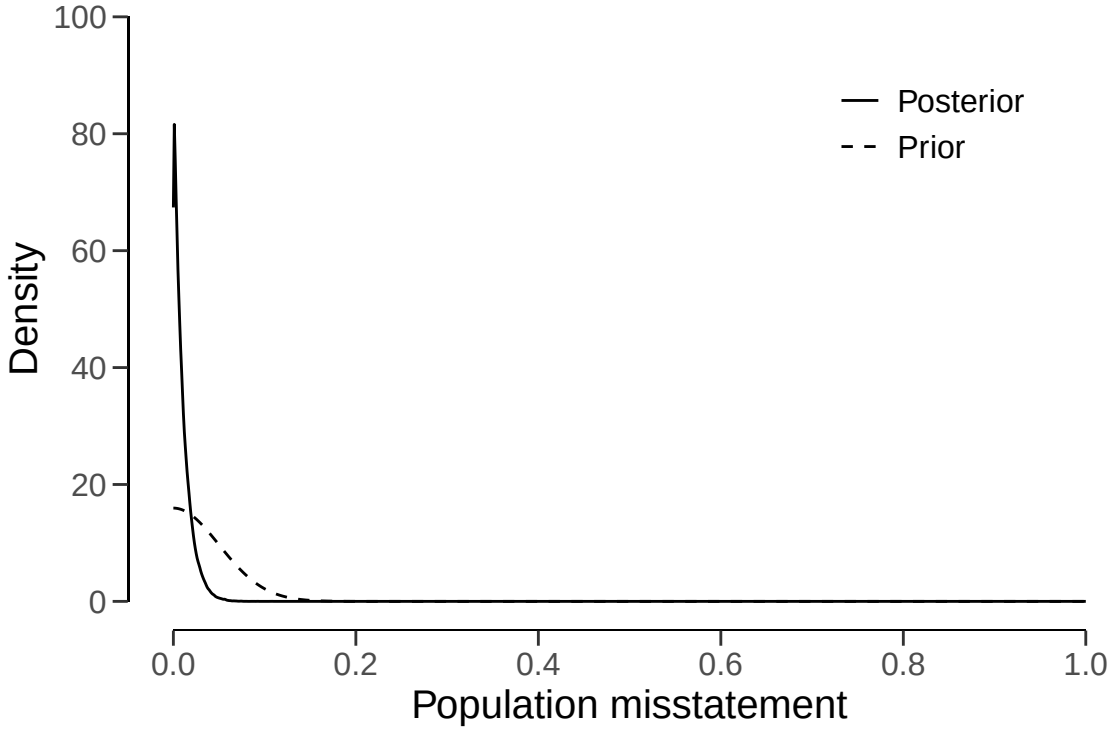
```
prior <- auditPrior(method = "param", likelihood = "normal",
                    alpha = 0, beta = 0.05)

plan <- planning(materiality = 0.03, expected = 0, conf.level = 0.95,
                 likelihood = "poisson", prior = prior)

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(  = 0,   = 0.05)T[0,1]
#>
#> Results:
#>   Minimum sample size:           90
#>   Tolerable errors:              0
#>   Posterior distribution:        Determined via MCMC sampling
#>   Expected most likely error:    0.0008648
#>   Expected upper bound:          0.029029
#>   Expected precision:            0.028164
#>   Expected BF :                 19.08
```

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.

```
plot(plan)
```



3.4 The Poisson Likelihood

Let's consider how to use the Poisson likelihood to calculate the minimum sample size needed to achieve a desired level of assurance. 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: λ, while the binomial has two parameters: θ and n). However, the Poisson likelihood is more conservative than the binomial likeliood, meaning that resulting sample sizes will be higher.

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

$$p(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{3.13}$$

where k is the number of misstatements in the sample, and λ 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 θ, and the sample size, n, by the following equation:

$$\lambda = n\theta. \tag{3.14}$$

3.4.1 Classical planning

Concretely, the following statistical model is assumed:

$$k \sim \text{Poisson}(n\theta_{max}) \tag{3.15}$$

Given a desired misstatement tolerance θ_{max} and the Poisson likelihood, we can solve for the minimum sample size n needed to achieve a assurance 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:

$$n = \lceil -\frac{\ln(\alpha)}{\theta_{max}} \rceil. \tag{3.16}$$

$\lceil ... \rceil$ is the ceiling function. Hence, $\lceil 1.2 \rceil = 2$.

For example, if we want to achieve an assurance level of 95% ($\alpha = 0.05$) 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 `planning()` function.

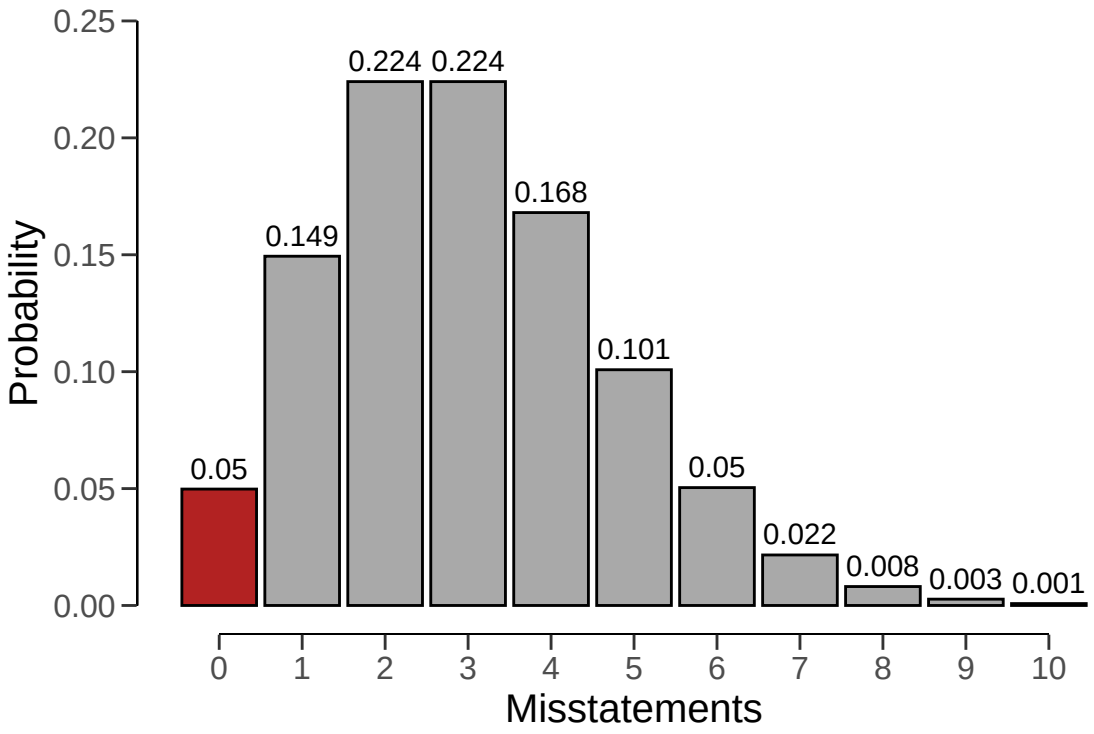
```
planning(materiality = 0.03, expected = 0, conf.level = 0.95,
         likelihood = "poisson")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 100
#> sample size obtained in 101 iterations via method 'poisson'
```

The `dpois()` function calculates the probability of observing k missatements in a sample of n items given an assumed misstatement probability. The sample size of 100 can be confirmed by

checking that 100 is the minimum integer that results in less than 5% probability of finding 0 misstatements if the population misstatement is truly 3%.

```
dpois(x = 0, lambda = 99 * 0.03) < 0.05  # 99:  Not sufficient
#> [1] FALSE
dpois(x = 0, lambda = 100 * 0.03) < 0.05 # 100: Sufficient
#> [1] TRUE
```

We can make this visually intuitive by showing the Poisson($k \mid 100 * 0.03$) distribution and highlighting the probability for $k = 0$. This probability should be lower than the required sampling risk $\alpha = 0.05$.



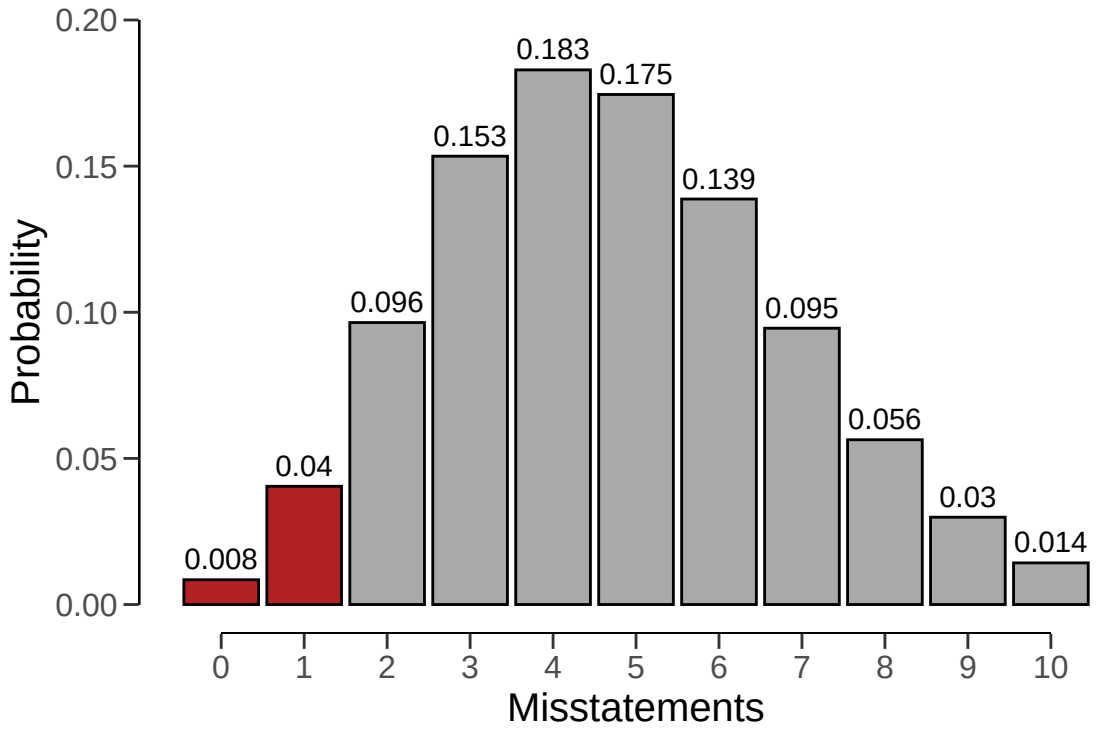
However, if the number of expected misstatements in the sample is non-zero, it becomes more difficult to solve the formula for n. Hence, we can iteratively try every value of n and return the smallest integer that satisfies the sampling objectives. In **jfa**, this can be done by adjusting the `expected` argument in the `planning()` function. For example, if we want to achieve an assurance level of 95% ($\alpha = 0.05$) 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 $n = 159$.

```
planning(materiality = 0.03, expected = 1, conf.level = 0.95,
         likelihood = "poisson")
#>
#>  Classical Audit Sample Planning
#>
#> minimum sample size = 159
#> sample size obtained in 158 iterations via method 'poisson'
```

Once again, the sample size of 159 can be confirmed by checking that 159 is the minimum integer that results in less than 5% probability of finding 0 or 1 misstatements if the population misstatement is truly 3%.

```
sum(dpois(x = 0:1, lambda = 158 * 0.03)) < 0.05 # 158: Not sufficient
#> [1] FALSE
sum(dpois(x = 0:1, lambda = 159 * 0.03)) < 0.05 # 159: Sufficient
#> [1] TRUE
```

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



3.4.2 Bayesian Planning

Performing Bayesian planning with the Poisson likelihood requires that you specify a prior distribution for the parameter θ. 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). Concretely, this means that the following statistical model is assumed:

$$k \sim \text{Poisson}(n\theta) \tag{3.17}$$
$$\theta \sim \text{Gamma}(\alpha, \beta) \tag{3.18}$$

> The gamma prior distribution is conjugate to the Poisson likelihood (see this list of conjugate priors), which means that the posterior distribution of θ can be determined analytically. For example, if the prior distribution is gamma(α, β) and the auditor has observed a sample of n items containing k misstatements, the posterior distribution for θ is gamma($\alpha + k$, $\beta + n$).

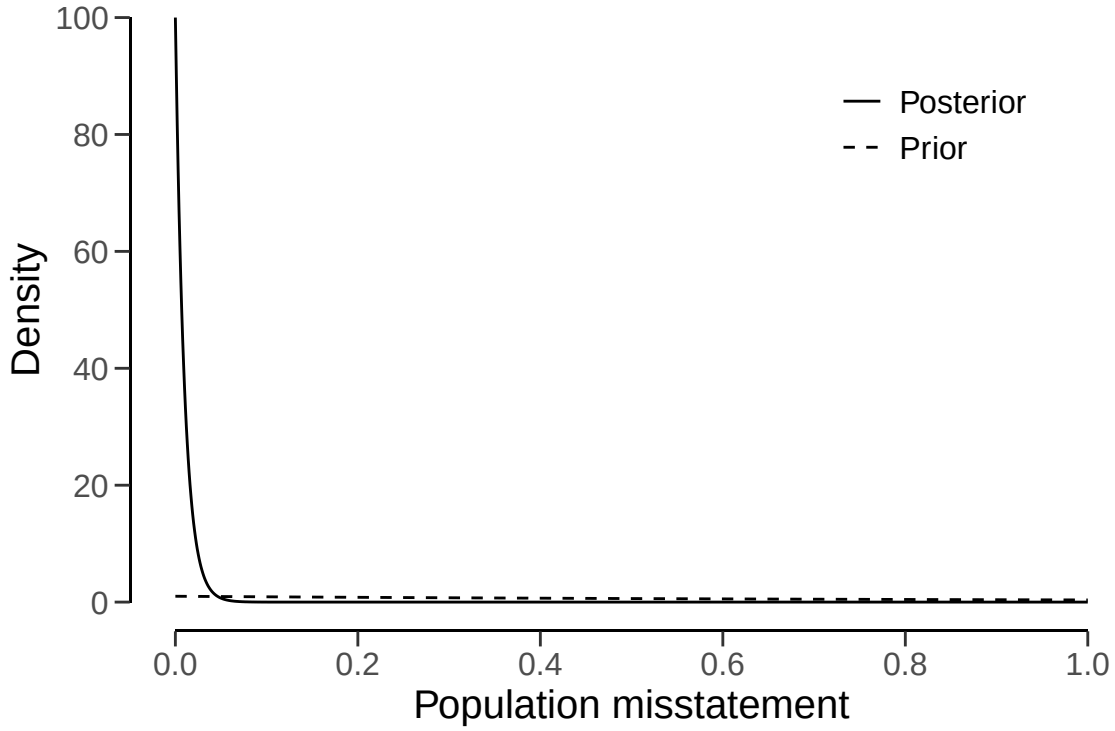
For example, the command below uses a default gamma($\alpha = 1$, $\beta = 1$) prior distribution to plan the sample, since `planning()` is given the Poisson likelihood. If we want to achieve an assurance level of 95% ($\alpha = 0.05$) for a performance materiality of $\theta_{max} = 0.03$, then the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

```
plan <- planning(materiality = 0.03, expected = 0, conf.level = 0.95,
                 likelihood = "poisson", prior = TRUE)
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(  = 1,   = 1)
#>
#> Results:
#>   Minimum sample size:           99
#>   Tolerable errors:              0
```

```
#>   Posterior distribution:        gamma(  = 1,   = 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. The expected posterior distribution is the posterior distribution that would occur if you actually observed the planned sample containing the expected misstatements.

```
plot(plan)
```



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 assumed:

$$k \sim \text{Poisson}(n\theta) \tag{3.19}$$
$$\theta \sim \text{Normal}(\mu = 0, \sigma = 0.05) \tag{3.20}$$

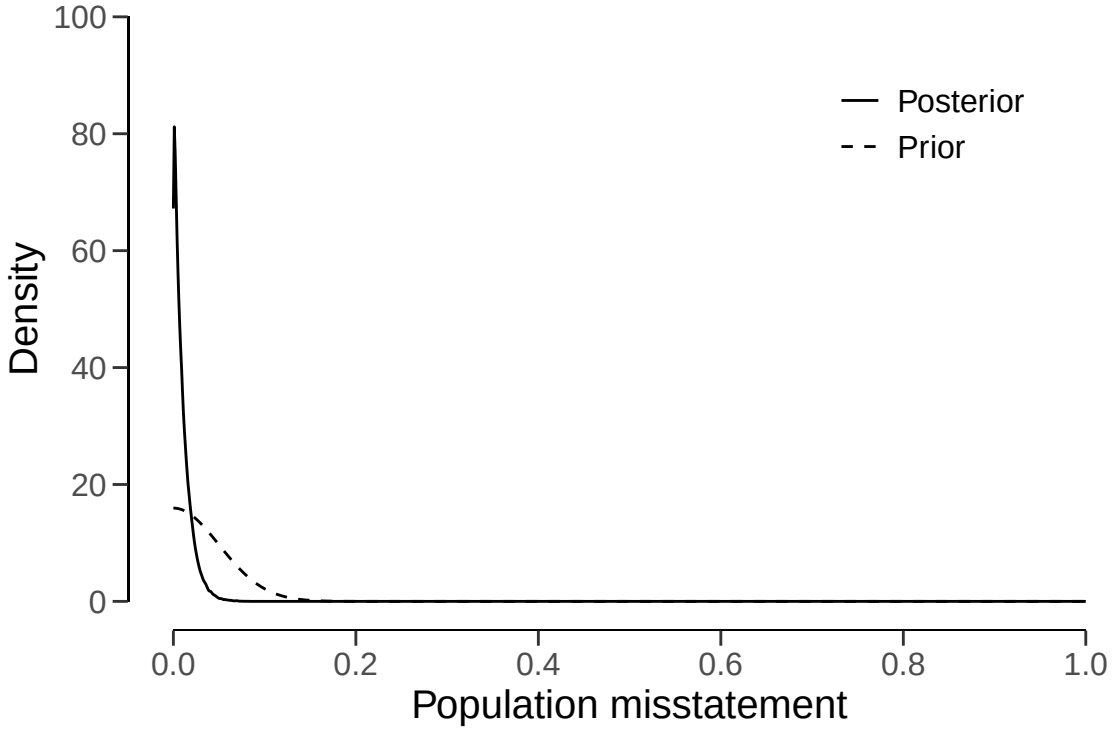
```
prior <- auditPrior(method = "param", likelihood = "normal",
                    alpha = 0, beta = 0.05)

plan <- planning(materiality = 0.03, expected = 0, conf.level = 0.95,
                 likelihood = "poisson", prior = prior)

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(  = 0,   = 0.05)T[0,1]
#>
#> Results:
#>   Minimum sample size:           91
#>   Tolerable errors:              0
#>   Posterior distribution:        Determined via MCMC sampling
#>   Expected most likely error:    0.0010232
#>   Expected upper bound:          0.029029
#>   Expected precision:            0.028006
#>   Expected BF :                 19.296
```

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

```
plot(plan)
```



3.5 Practical Examples

This section contains practical examples of how to construct a prior distribution based on audit information.

3.5.1 Audit Risk Model

In this 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 3%. Based on the results of last years audit, where the estimate of the maximum misstatement was 1%, the auditor wants to tolerate 1% 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%, 60%, and 100%.

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

By using the detection risk as the adjusted audit risk, the auditor can plan for a sample while taking into account the risk-reducing information from the assessments of inherent risk and control risk. 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'
```

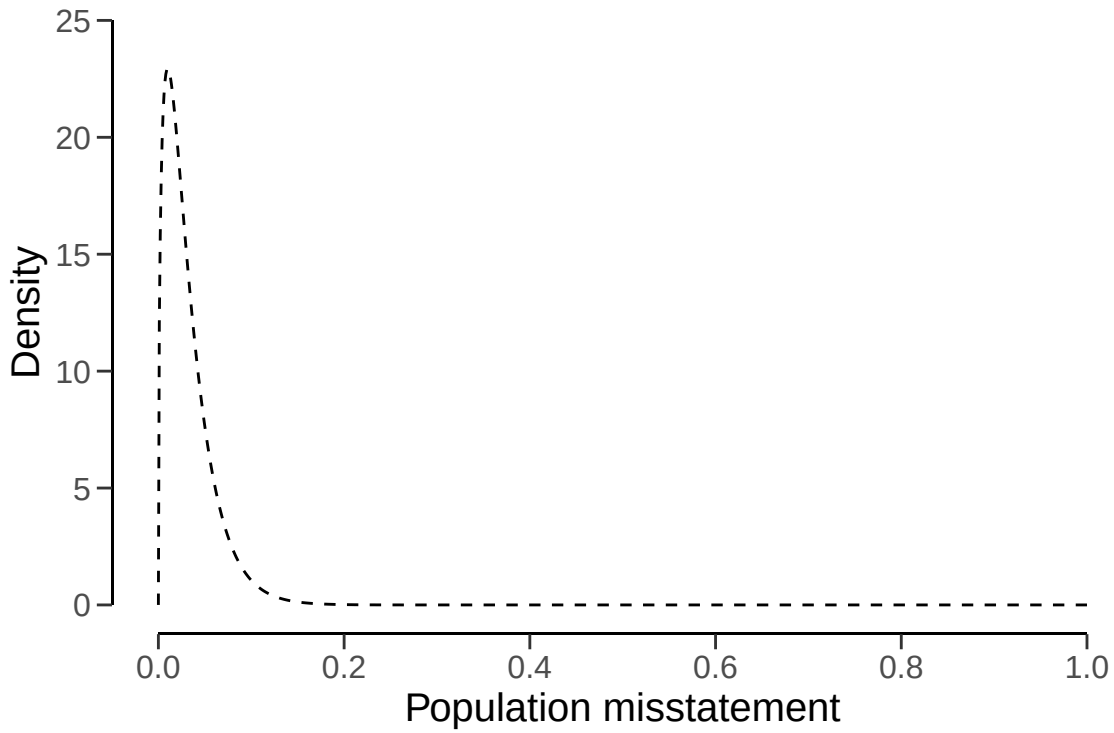
The auditor is free to apply a Bayesian philosophy in planning the sample. For example, the risk assessments from the ARM can be incorporated into a prior distribution. This can be done using `method = "arm"` in the `auditPrior()` function, which takes the values of the inherent risk probability `ir` 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)
summary(prior)
#>
#>  Prior Distribution Summary
#>
#> Options:
#>   Likelihood:                   poisson
#>   Specifics:                    ir = 1; cr = 0.6; dr = 0.0833333
```

```
#>
#> Results:
#>   Functional form:              gamma(  = 1.46,   = 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
```

The prior distribution can be visualized using the `plot()` function.

```
plot(prior)
```



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'
```

3.5.2 Benchmark Analysis

The auditor may incorporate information obtained through analytical procedures (Derks, Swart, Batenburg, et al. 2021), such as a benchmark analysis, into the prior distribution for θ. 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.

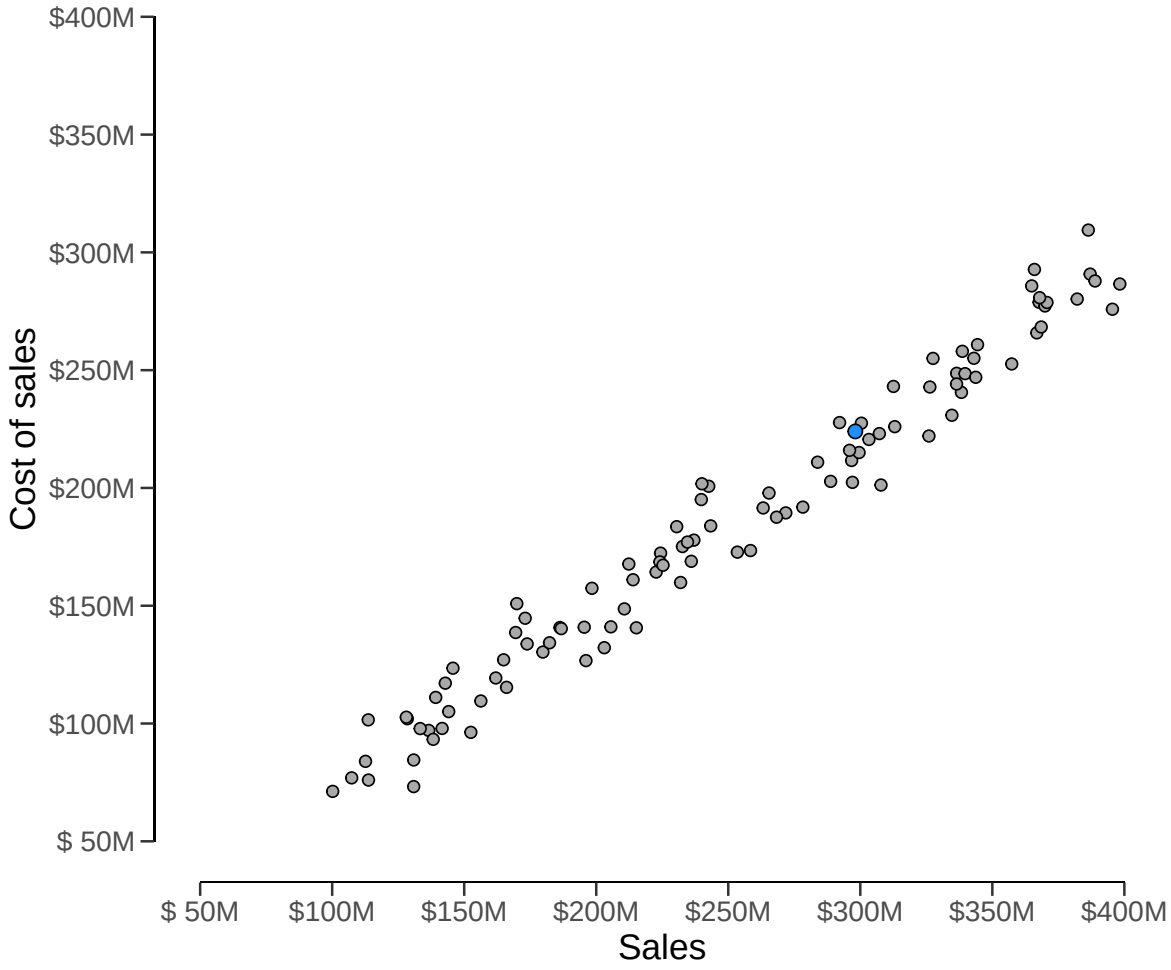
One way to construct a prior distribution on the basis of data is through the use of regression models, such as benchmarking the relationship between sales and costs of sales within the auditee's specific industry sector. The **jfa** package includes a data set `benchmark` that can be used for this example.

```
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 the figure below, which visualizes the industry sales versus the cost of sales.

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

$$C = \beta_0 + \beta_1 \cdot S + \epsilon. \tag{3.21}$$


$400M
$350M
$300M
$250M
$200M
$150M
$100M
$ 50M
Cost of sales
$ 50M
$100M
$150M
$200M
$250M
$300M
$350M
$400M
Sales

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

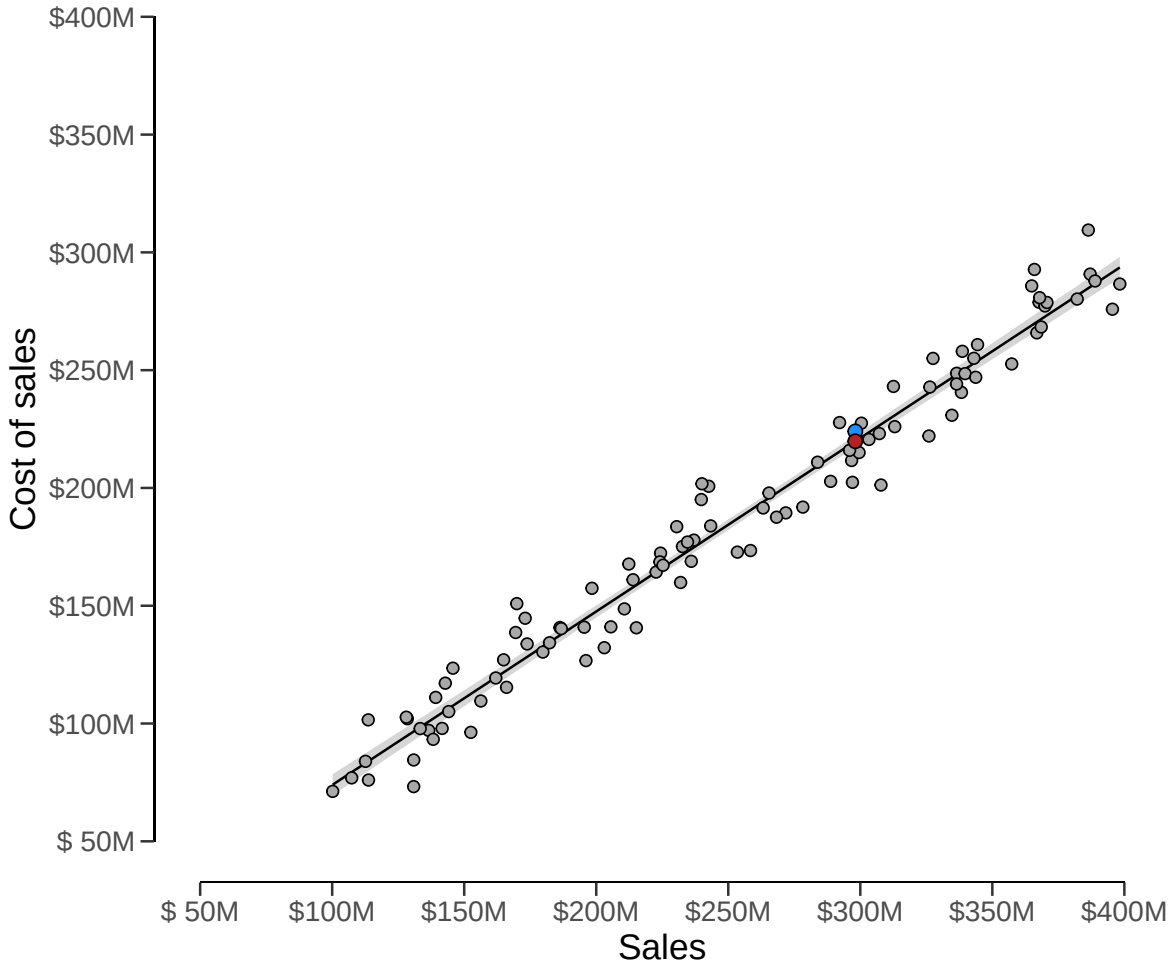
```
fit <- lm(costofsales ~ 1 + sales, data = benchmark)
summary(fit)
#>
#> Call:
#> lm(formula = costofsales ~ 1 + sales, data = benchmark)
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -25736696  -7052141   -226945   6857840  25498106
#>
#> Coefficients:
#>              Estimate 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_hat <- predict(fit, newdata = data.frame(sales = 298112312),
                 interval = "prediction", level = 0.90)[1]
C_hat
#> [1] 219817866
```

The fitted regression line and the predicted cost of sales (red dot) are visualized in the figure below:

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 θ can utilize the relative error distribution from the linear regression. This relative error distribution, which is a Normal(μ, σ) 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 μ of the prior distribution is determined by the relative deviation of the auditee's booked


$400M
$350M
$300M
$250M
$200M
$150M
$100M
$ 50M
Cost of sales
$ 50M
$100M
$150M
$200M
$250M
$300M
$350M
$400M
Sales

cost of sales when compared to the predicted cost of sales according to the benchmark data $\frac{C-\hat{C}}{C}$.

```
mu <- (223994405 - C_hat[1]) / 223994405
mu
#> [1] 0.01864573
```

The standard deviation of the prior distribution is expressed through the standard deviation of the distribution of ϵ:

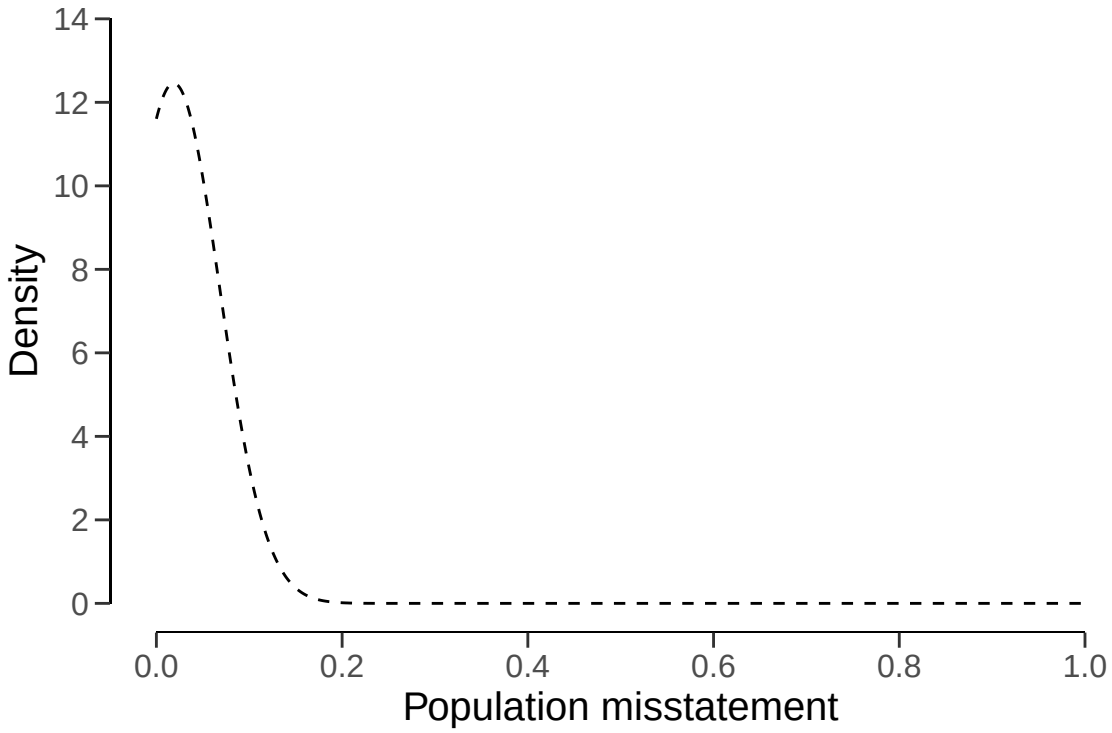
```
stdev <- sd(fit$residuals) / 223994405
stdev
#> [1] 0.04951199
```

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:                       = 0.0186457;   = 0.049512
#>
#> Results:
#>   Functional form:               normal(  = 0.019,   = 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.

```
plot(prior)
```

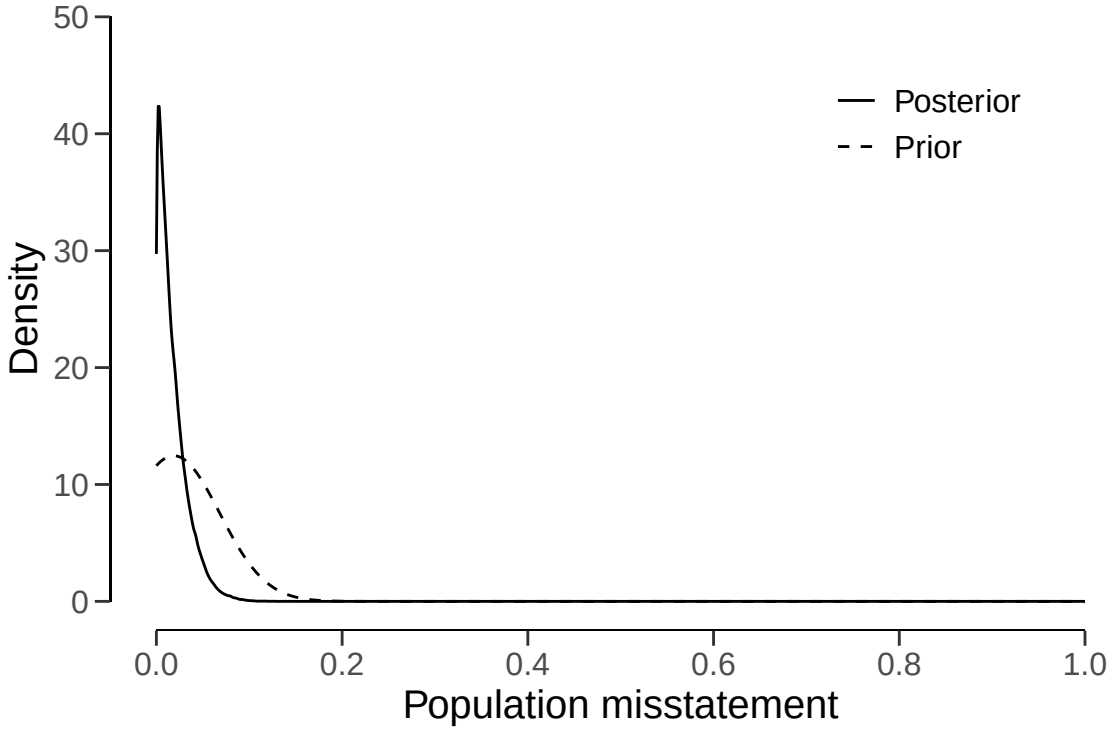


By using this prior distribution, the required minimum sample size is 50.

```
plan <- planning(materiality = 0.05, conf.level = 0.95,
                 likelihood = "binomial", prior = prior)
plan
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 50
#> sample size obtained in 51 iterations via method 'binomial' + 'prior'
```

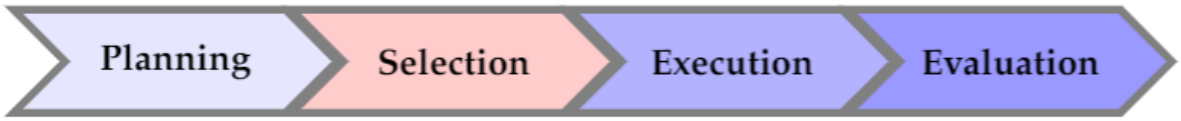
You can inspect how the prior distribution compares to the expected posterior distribution by using the `plot()` function.

```
plot(plan)
```


50
40
30
20
10
0
Density
0.0
0.2
0.4
0.6
0.8
1.0
Population misstatement
Posterior
Prior

4 Selecting a Sample

This chapter outlines the most commonly used sampling methodology for auditing and shows how to select a sample using these methods with the **jfa** package.



Auditors are often required to assess balances or processes that involve a large number of items. Since they cannot inspect all of these items individually, they need to select a subset (i.e., a sample) from the total population to make a statement about a certain characteristic of the population. For this purpose, various selection methodologies are available that have become standard in an audit context.

4.1 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.

4.1.1 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.

4.1.2 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^{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.

4.2 Sampling Methods

This section discusses the four sampling methods implemented in **jfa**. First, for notation, let the the population N be defined as the total set of individual sampling units x_i.

$$N = \{x_1, x_2, \dots, x_N\}. \tag{4.1}$$

In statistical sampling, every sampling unit x_i in the population must 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.

The next section discusses which sampling methods are available in **jfa**. To illustrate the outcomes for different sampling methods, we will use the `BuildIt` data set that can be loaded using the code below. For simplicity, we will use a sample size of 10 for all examples.

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

4.2.1 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:

$$p(x) = \frac{1}{N}, \tag{4.2}$$

where N is the number of units in the population. To clarify this procedure, Figure 3 provides an illustration of the random sampling method.

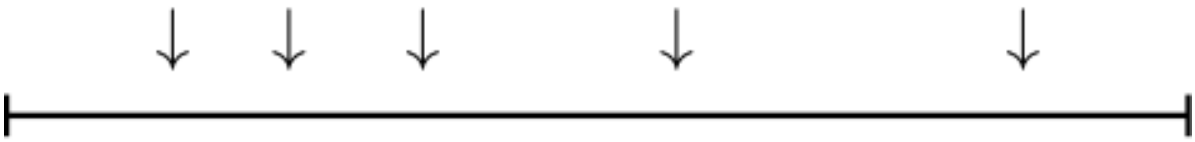


Figure 4.1: Figure 4.1: Illustration of random sampling

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.

4.2.1.1 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(1: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 following command:

```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "items",
                    method = "random")
result$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
```

4.2.1.2 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(1:nrow(BuildIt), size = n, replace = FALSE,
                prob = BuildIt$bookValue)
items
#>  [1] 2174 2928 1627  700  147 3056 3118 2045 1311  716
```

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

```
BuildIt[items, ]
#>         ID bookValue auditValue
#> 2174 90260    625.98     625.98
```

```
#> 2928 68595    548.21     548.21
#> 1627 98301    429.07     429.07
#> 700  29683    239.26     239.26
#> 147  72906    677.62     677.62
#> 3056 86317    246.22     246.22
#> 3118 14548    204.63     204.63
#> 2045 45416    381.05     381.05
#> 1311 91955    398.96     398.96
#> 716  12815    873.43     873.43
```

The sample can be reproduced in **jfa** via the following command:

```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "values",
                    method = "random", values = "bookValue")
result$sample
#>     row times    ID bookValue auditValue
#> 1  2174     1 90260    625.98     625.98
#> 2  2928     1 68595    548.21     548.21
#> 3  1627     1 98301    429.07     429.07
#> 4   700     1 29683    239.26     239.26
#> 5   147     1 72906    677.62     677.62
#> 6  3056     1 86317    246.22     246.22
#> 7  3118     1 14548    204.63     204.63
#> 8  2045     1 45416    381.05     381.05
#> 9  1311     1 91955    398.96     398.96
#> 10  716     1 12815    873.43     873.43
```

4.2.2 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:

$$I = \frac{N}{n}, \tag{4.3}$$

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:

$$p(x) = \frac{1}{I}, \tag{4.4}$$

with the property that the space between selected units i is the same as the interval I, see Figure 1. However, in practice the selection is deterministic and completely depends on the chosen starting points (using `start`).

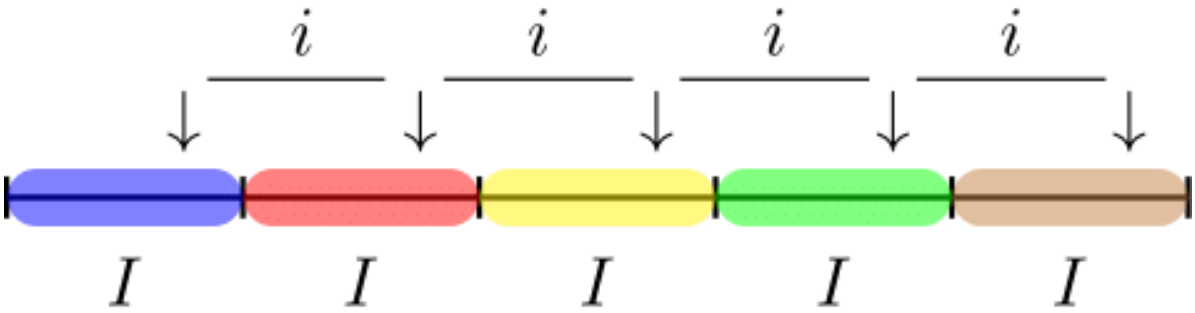


Figure 4.2: Figure 4.2: Illustration of fixed interval sampling

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.

4.2.2.1 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 <- floor(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 following command. Note that, by default, the first sampling unit from each interval is selected. However, this can be changed by setting the argument `start = 1` to a different value.

```
result <- selection(data = BuildIt, size = n, units = "items",
                    method = "interval", start = start)
result$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
```

4.2.2.2 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 <- floor(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 following command:

```
result <- selection(data = BuildIt, size = n, units = "values",
                    method = "interval", values = "bookValue", start = start)
result$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
```

4.2.3 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.

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

$$p(x) = \frac{1}{I}. \tag{4.5}$$

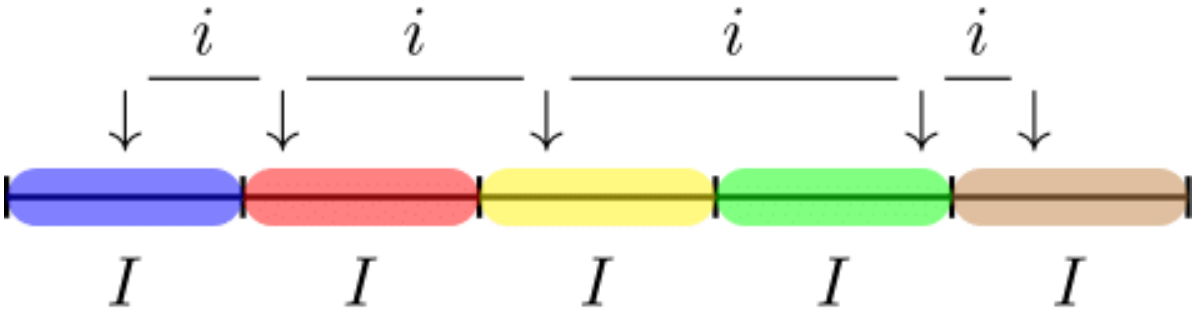


Figure 4.3: Figure 4.3: Illustration of cell sampling

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.

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.

4.2.3.1 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 following command. Note that, by default, the first sampling unit from each interval is selected. However, this can be changed by setting the argument `start = 1` to a different value.

```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "items",
                    method = "cell")
result$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
```

4.2.3.2 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 following command:

```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "values",
                    method = "cell", values = "bookValue")
result$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
```

4.2.4 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:

$$S_i = \frac{Y_i}{R_i} \tag{4.6}$$

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.

4.2.4.1 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 following command:

```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "values",
                    method = "sieve", values = "bookValue")
result$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
```

4.3 Ordering or Randomizing the Population

The `selection()` function has additional arguments (`order`, `decreasing`, and `randomize`) to preprocess your population before selection. 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 selection. In this case, you should use the `decreasing = FALSE` (default) argument.

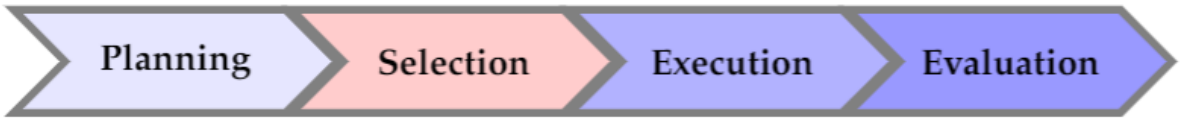
```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "values",
                    values = "bookValue", order = "bookValue")
result$sample
#>     row times    ID bookValue auditValue
#> 1  2662     1 30568     14.47      14.47
#> 2  2307     1 95785    244.49     244.49
#> 3  1639     1 39570    307.54     307.54
#> 4  2318     1 11857    360.57     360.57
```

```
#> 5  3409     1 18796    414.75     414.75
#> 6  1158     1 28985    468.57     468.57
#> 7  2892     1 27448    530.73     530.73
#> 8  2759     1 16442    596.34     596.34
#> 9  3112     1 62440    678.36     678.36
#> 10 1820     1 82309    796.31     796.31
```

The `randomize` argument can be used to randomly shuffle the items in the population before selection.

```
set.seed(1)
result <- selection(data = BuildIt, size = n, units = "values",
                    values = "bookValue", randomize = TRUE)
result$sample
#>     row times    ID bookValue auditValue
#> 1  1017     1 50755    618.24     618.24
#> 2  2097     1 53186    642.34     642.34
#> 3  2508     1 55666    426.33     426.33
#> 4   779     1 82046    569.21     227.68
#> 5  2255     1 63601    234.12     234.12
#> 6  2072     1 92569    517.14     517.14
#> 7  1938     1 35525    366.94     366.94
#> 8   595     1 30750    338.36     338.36
#> 9  3207     1 74858    884.51     884.51
#> 10 2981     1 24682    302.23     302.23
```

5 Evaluating a Sample



A non-stratified audit sample does not involve dividing the population into subgroups. Here, the auditor selects a random sample from the entire population without considering any specific characteristics of the population.

For example, in an audit of a company's inventory, the auditor may simply select a random sample of items from the entire inventory without dividing it into subgroups based on characteristics such as location or type of item. Another example of such a situation would be where the auditor is auditing the general ledger of a small business.

Naturally, non-stratified sampling is simpler than stratified sampling and can be used when the population is considered homogenous or the auditor does not need to consider differences between subgroups (e.g., strata).

5.1 Classical Evaluation

Classical hypothesis testing uses the p-value to make a decision about whether to reject the hypothesis H_0 or not. As an example, consider that an auditor wants to verify whether the population contains less than 5 percent misstatement, implying the hypotheses $H_1 : \theta < 0.05$ and $H_0 : \theta \geq 0.05$. They have taken a sample of 100 items, of which 1 contained an error. They set the significance level for the p-value to 0.05, implying that $p < 0.05$ will be enough to reject the hypothesis H_0. The call below evaluates the sample using a classical non-stratified evaluation procedure.

```
evaluation(materiality = 0.05, x = 1, n = 100)
#>
#>  Classical Audit Sample Evaluation
#>
#> data:  1 and 100
#> number of errors = 1, number of samples = 100, taint = 1, p-value =
```

```
#> 0.040428
#> alternative hypothesis: true misstatement rate is less than 0.05
#> 95 percent confidence interval:
#>  0.00000000 0.04743865
#> most likely estimate:
#>  0.01
#> results obtained via method 'poisson'
```

The output shows that the most likely error in the population is estimated to be 1 / 100 = 1% and that the 95% (one-sided) confidence interval ranges from 0% to 4.74%. The output also shows that the p-value is lower than 0.05 implying that the hypothesis H_0 can be rejected. Hence, the auditor is able to conclude that the sample provides sufficient evidence to state with reasonable assurance that the population does not contain material misstatement.

5.2 Bayesian Evaluation

Bayesian hypothesis testing uses the Bayes factor, BF_{10} or BF_{01}, to make a statement about the evidence provided by the sample in support for one of the two hypotheses H_1 or H_0. As an example of how to interpret the Bayes factor, the value of $BF_{10} = 10$ (provided by the `evaluation()` function) can be interpreted as: *the data are 10 times more likely to have occurred underthe hypothesis H_1 than under the hypothesis H_0*. $BF_{10} > 1$ indicates evidence in favor of H_1 and against H_0, while $BF_{10} < 1$ indicates evidence in favor of H_0 and against H_1. The `evaluation()` function returns the value for BF_{10}, but BF_{01} can be computed as $\frac{1}{BF_{10}}$.

Consider the previous example of an auditor who wants to verify whether the population contains less than 5 percent misstatement, implying the hypotheses $H_1 : \theta < 0.05$ and $H_0 : \theta \geq 0.05$. They have taken a sample of 100 items, of which 1 was found to contain a misstatement. The prior distribution is assumed to be a default *beta(1,1)* prior. The call below evaluates the sample using a Bayesian non-stratified evaluation procedure.

```
prior <- auditPrior(materiality = 0.05, method = "default", likelihood = "binomial")
evaluation(materiality = 0.05, 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  =
#> 515.86
#> alternative hypothesis: true misstatement rate is less than 0.05
#> 95 percent credible interval:
```

```
#>  0.00000000 0.04610735
#> most likely estimate:
#>  0.01
#> results obtained via method 'binomial' + 'prior'
```

The output shows that the most likely error in the population is estimated to be 1 / 100 = 1% and that the 95% (one-sided) credible interval ranges from 0% to 4.61%. The small difference between the classical and default Bayesian results is due to the prior distribution, which must be proper in order to calculate a Bayes factor (classical results can be emulated by constructing a prior with `method = "strict"` in the `auditPrior()` function). The Bayes factor in this case is shown to be $BF_{10} = 515$, meaning that the data from the sample are about 515 times more likely to occur under the hypothesis of tolerable misstatement than under the hypothesis of material misstatement.

Note that this is a very high Bayes factor for the little data that is observed. That is because the Bayes factor is dependent on the prior distribution for θ. As a rule of thumb, when the prior distribution is highly conservative (as with `method = 'default'`) with respect to the hypothesis of tolerable misstatement, the Bayes factor tends to over quantify the evidence in favor of this hypothesis. You can mitigate this dependency by using a prior distribution that is impartial with respect to the hypotheses via `method = "impartial"` in the `auditPrior()` function (Derks et al., 2022).

```
prior <- auditPrior(materiality = 0.05, method = "impartial", likelihood = "binomial")
evaluation(materiality = 0.05, 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  =
#> 47.435
#> alternative hypothesis: true misstatement rate is less than 0.05
#> 95 percent credible interval:
#>  0.00000000 0.04110834
#> most likely estimate:
#>  0.0088878
#> results obtained via method 'binomial' + 'prior'
```

The output shows that $BF_{10} = 47$, implying that under the assumption of impartiality there is strong evidence for H_1, the hypothesis that the population contains misstatements lower than 5 percent of the population (tolerable misstatement). Since the two prior distributions both resulted in convincing Bayes factors, the results can be considered robust to the choice of prior distribution. Hence, the auditor is able to conclude that the sample provides convincing evidence to state that the population does not contain material misstatement.

5.3 Evaluation using Data

For this example, we take the `allowances` that set that comes with the package. This data set contains 3500 financial statement line items, each with a booked value `bookValue` and, for illustrative purposes, and audited (true) value `auditValue`. Since the focus of this vignette 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 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
```

Evaluating a non-stratified sample using data requires specification of the `data`, `values` and `values.audit` arguments. The input for these arguments is the name of the specific column in `data`.

5.3.1 Classical Evaluation

The call below evaluates the `allowances` sample using a classical non-stratified evaluation procedure.

```
x <- evaluation(
  materiality = 0.05, data = allowances,
  values = "bookValue", values.audit = "auditValue", times = "times"
)
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
```

In this case, the output shows that the estimate of the misstatement in the population is 15.77%, with the 95% (one-sided) confidence interval ranging from 0% to 17.5%.

5.3.2 Bayesian evaluation

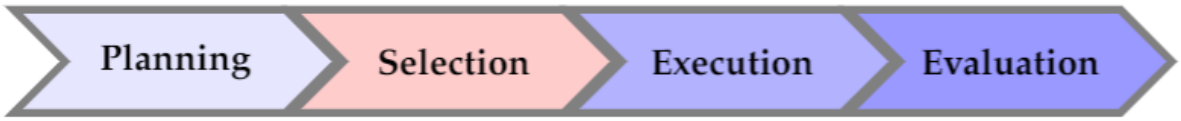
The call below evaluates the `allowances` sample using a Bayesian non-stratified evaluation procedure.

```
x <- evaluation(
  materiality = 0.05, data = allowances, prior = TRUE,
  values = "bookValue", values.audit = "auditValue", times = "times"
)
summary(x)
#>
#>  Bayesian Audit Sample Evaluation Summary
#>
#> Options:
#>   Confidence level:               0.95
#>   Materiality:                    0.05
#>   Hypotheses:                     H : θ > 0.05 vs. H : θ < 0.05
#>   Method:                         poisson
#>   Prior distribution:             gamma(  = 1,   = 1)
#>
#> Data:
#>   Sample size:                    1604
#>   Number of errors:               401
#>   Sum of taints:                  252.9281046
#>
#> Results:
#>   Posterior distribution:         gamma(  = 253.928,   = 1605)
```

```
#>   Most likely error:              0.15759
#>   95 percent credible interval:   [0, 0.17489]
#>   Precision:                      0.0173
#>   BF :                            0
```

The output shows that the estimate of the misstatement in the population is 15.76%, with the 95% (one-sided) credible interval ranging from 0% to 17.49%.

6 Evaluating a Stratified Sample



In an audit context, stratified sampling can be used to select a sample of transactions from different departments, locations, or business units to ensure that the sample is representative of the population.

For example, if you want to audit the expense claims of a large organization, you can stratify the population based on the department, location, or business unit to ensure that all departments are represented accordingly in the sample. 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.

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

In general, there are three approaches to evaluating a stratified sample: no pooling, complete pooling, and partial pooling (see Derks et al., 2022). When using `evaluation()`, you must to indicate which type of pooling to use via the `pooling` argument. 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., multilevel/hierarchical modeling) is a powerful technique that can result in more efficient population and stratum estimates but is currently only feasible when performing a Bayesian analysis. For this reason, this vignette only describes the Bayesian approach to stratified evaluation but going from this approach to a classical approach only requires setting `prior = FALSE`.

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.

6.1 No pooling

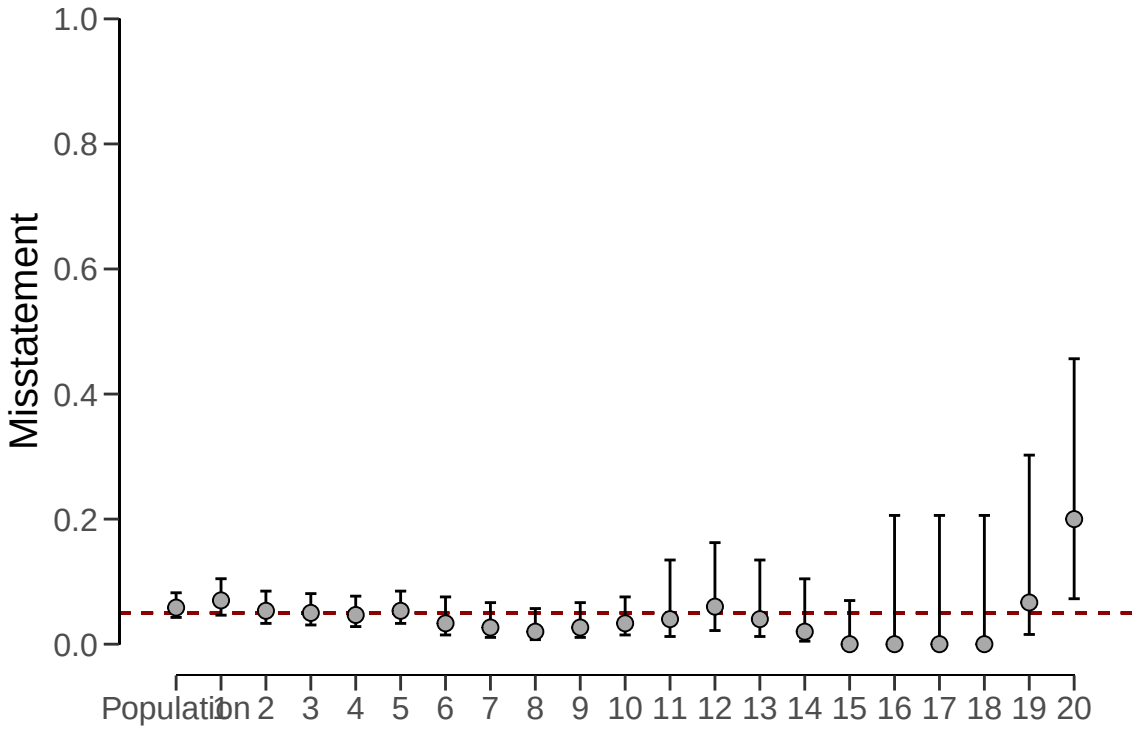
No pooling (`pooling = "none"`, default) assumes no similarities between strata. This means that the prior distribution specified through `prior` is applied independently for each stratum. This allows for independent estimates for the misstatement in each stratum but also results in a relatively high uncertainty in the population estimate. 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) # Important because the posterior distribution is determined via sampling
result_np <- evaluation(
  materiality = 0.05, method = "binomial", prior = TRUE,
  n = retailer$samples, x = retailer$errors, N.units = retailer$items,
  alternative = "two.sided", pooling = "none"
)
summary(result_np)
#>
#>  Bayesian Audit Sample Evaluation Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Population size:               144000
#>   Materiality:                   0.05
#>   Hypotheses:                    H : θ = 0.05 vs. H : θ   0.05
#>   Method:                        binomial
#>   Prior distribution:            Determined via MCMC sampling
#>
#> Data:
```

```
#>   Sample size:                   2575
#>   Number of errors:              115
#>   Sum of taints:                 115
#>
#> Results:
#>   Posterior distribution:        Determined via MCMC sampling
#>   Most likely error:             0.058487
#>   95 percent credible interval:  [0.042763, 0.082201]
#>   Precision:                     0.023714
#>   BF :                           0
#>
#> Strata (20):
#>        N   n  x  t     mle      lb      ub precision
#> 1   5000 300 21 21 0.07000 0.04637 0.10467   0.03467
#> 2   5000 300 16 16 0.05333 0.03324 0.08489   0.03156
#> 3   5000 300 15 15 0.05000 0.03069 0.08086   0.03086
#> 4   5000 300 14 14 0.04667 0.02816 0.07681   0.03014
#> 5   5000 300 16 16 0.05333 0.03324 0.08489   0.03156
#> 6   5000 150  5  5 0.03333 0.01472 0.07558   0.04225
#> 7   5000 150  4  4 0.02667 0.01084 0.06643   0.03977
#> 8   5000 150  3  3 0.02000 0.00726 0.05696   0.03696
#> 9   5000 150  4  4 0.02667 0.01084 0.06643   0.03977
#> 10  5000 150  5  5 0.03333 0.01472 0.07558   0.04225
#> 11 10000  50  2  2 0.04000 0.01230 0.13459   0.09459
#> 12 10000  50  3  3 0.06000 0.02178 0.16242   0.10242
#> 13 10000  50  2  2 0.04000 0.01230 0.13459   0.09459
#> 14 10000  50  1  1 0.02000 0.00478 0.10447   0.08447
#> 15 10000  50  0  0 0.00000 0.00050 0.06978   0.06978
#> 16 10000  15  0  0 0.00000 0.00158 0.20591   0.20591
#> 17 10000  15  0  0 0.00000 0.00158 0.20591   0.20591
#> 18 10000  15  0  0 0.00000 0.00158 0.20591   0.20591
#> 19 10000  15  1  1 0.06667 0.01551 0.30232   0.23565
#> 20  4000  15  3  3 0.20000 0.07266 0.45646   0.25646
```

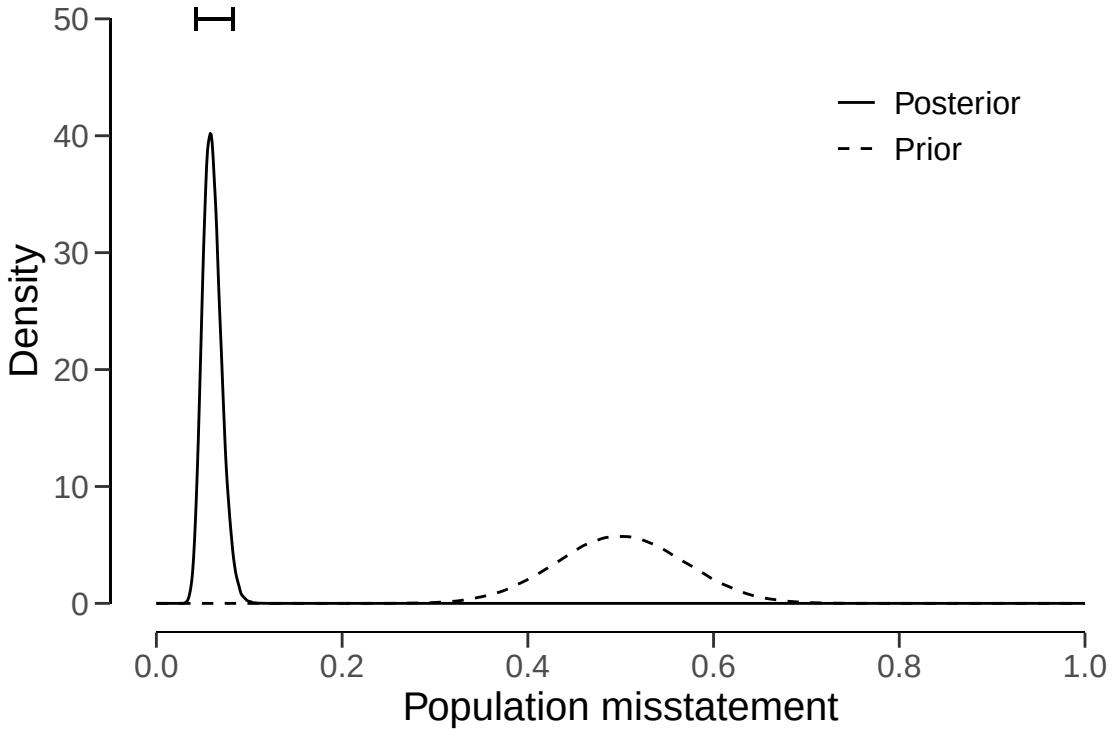
In this case, the output of the `summary()` function shows that the estimate of the misstatement in the population is 5.85%, with the 95% credible interval ranging from 4.28% to 8.22%. The stratum estimates differ substantially from each other but are relatively uncertain.

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



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

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



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

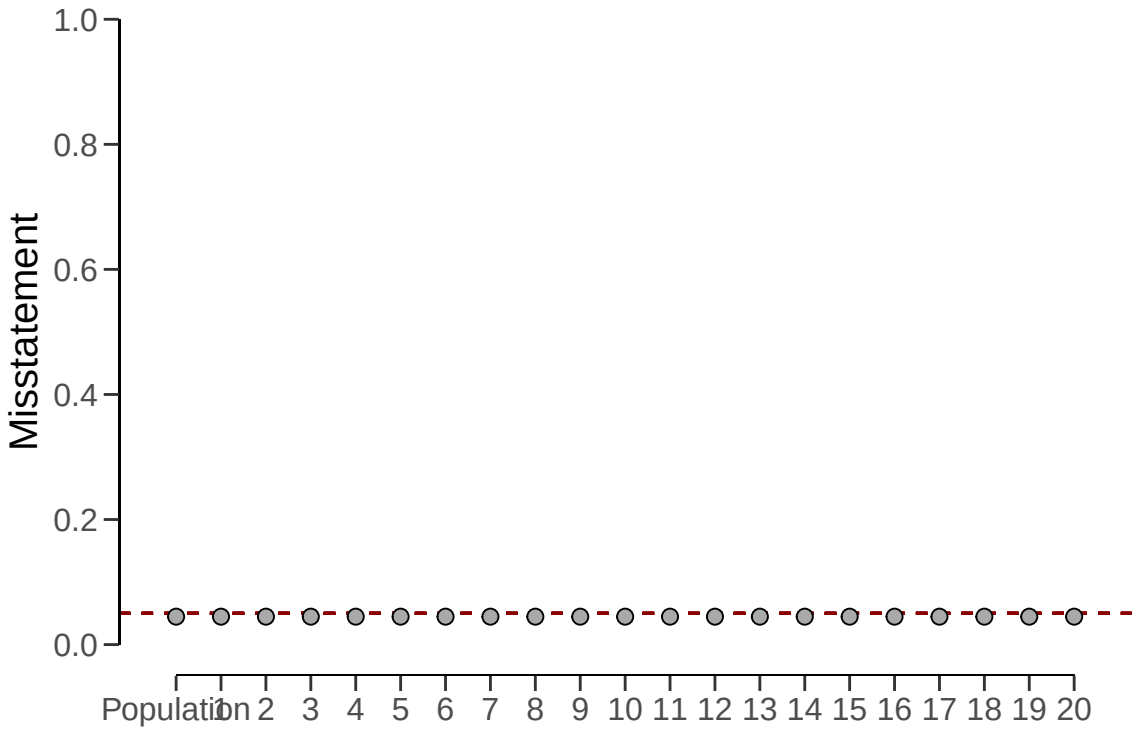
```
result_cp <- evaluation(
  materiality = 0.05, method = "binomial", prior = TRUE,
  n = retailer$samples, x = retailer$errors, N.units = retailer$items,
  alternative = "two.sided", pooling = "complete"
)
summary(result_cp)
#>
#>  Bayesian Audit Sample Evaluation Summary
```

```
#>
#> Options:
#>   Confidence level:               0.95
#>   Population size:                144000
#>   Materiality:                    0.05
#>   Hypotheses:                     H : θ = 0.05 vs. H : θ   0.05
#>   Method:                         binomial
#>   Prior distribution:             beta(  = 1,   = 1)
#>
#> Data:
#>   Sample size:                    2575
#>   Number of errors:               115
#>   Sum of taints:                  115
#>
#> Results:
#>   Posterior distribution:         beta(  = 116,   = 2461)
#>   Most likely error:              0.04466
#>   95 percent credible interval:   [0.03735, 0.053345]
#>   Precision:                      0.0086852
#>   BF :                            0.022725
#>
#> Strata (20):
#>        N   n  x  t     mle      lb      ub precision    bf10
#> 1   5000 300 21 21 0.04466 0.03735 0.05335   0.00869 0.02273
#> 2   5000 300 16 16 0.04466 0.03735 0.05335   0.00869 0.02273
#> 3   5000 300 15 15 0.04466 0.03735 0.05335   0.00869 0.02273
#> 4   5000 300 14 14 0.04466 0.03735 0.05335   0.00869 0.02273
#> 5   5000 300 16 16 0.04466 0.03735 0.05335   0.00869 0.02273
#> 6   5000 150  5  5 0.04466 0.03735 0.05335   0.00869 0.02273
#> 7   5000 150  4  4 0.04466 0.03735 0.05335   0.00869 0.02273
#> 8   5000 150  3  3 0.04466 0.03735 0.05335   0.00869 0.02273
#> 9   5000 150  4  4 0.04466 0.03735 0.05335   0.00869 0.02273
#> 10  5000 150  5  5 0.04466 0.03735 0.05335   0.00869 0.02273
#> 11 10000  50  2  2 0.04466 0.03735 0.05335   0.00869 0.02273
#> 12 10000  50  3  3 0.04466 0.03735 0.05335   0.00869 0.02273
#> 13 10000  50  2  2 0.04466 0.03735 0.05335   0.00869 0.02273
#> 14 10000  50  1  1 0.04466 0.03735 0.05335   0.00869 0.02273
#> 15 10000  50  0  0 0.04466 0.03735 0.05335   0.00869 0.02273
#> 16 10000  15  0  0 0.04466 0.03735 0.05335   0.00869 0.02273
#> 17 10000  15  0  0 0.04466 0.03735 0.05335   0.00869 0.02273
#> 18 10000  15  0  0 0.04466 0.03735 0.05335   0.00869 0.02273
#> 19 10000  15  1  1 0.04466 0.03735 0.05335   0.00869 0.02273
```

```
#> 20  4000  15  3  3 0.04466 0.03735 0.05335   0.00869 0.02273
```

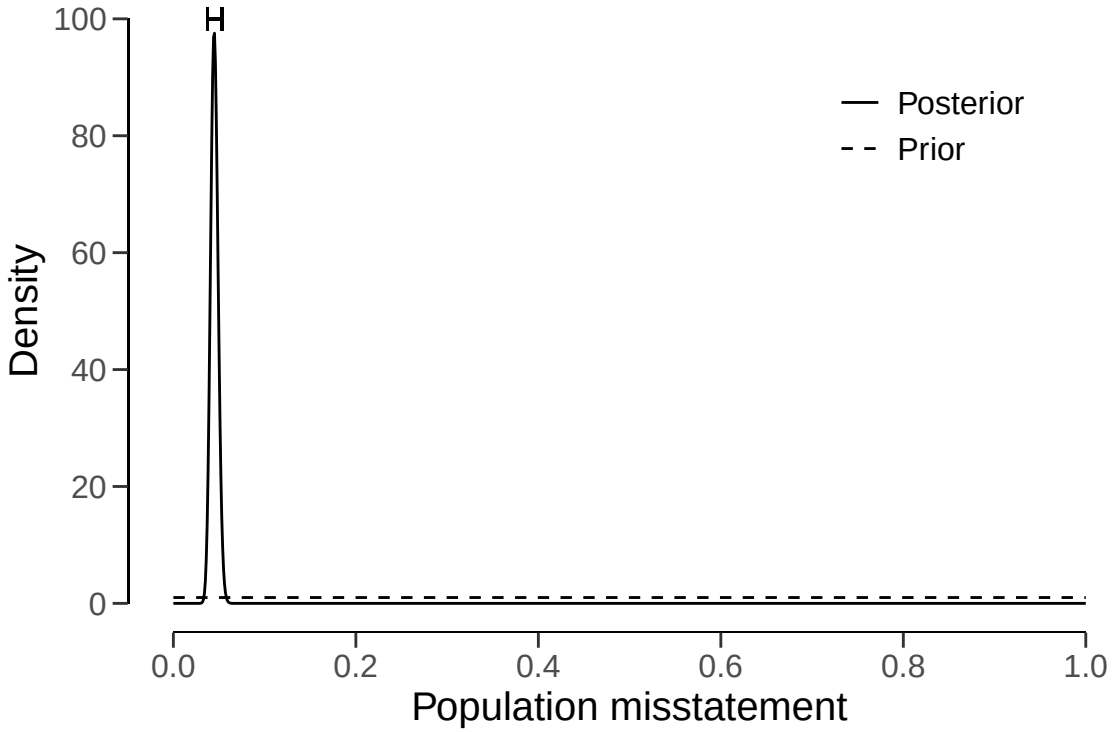
For example, the output of the `summary()` function shows that the estimate of the misstatement in the population is 4.47%, with the 95% credible interval ranging from 3.74% to 5.33%. Since the data is aggregated, the stratum estimates contain relatively little uncertainty. However, the probability of misstatement in stratum 20 (many misstatements) under this assumption is the same as that of stratum 15 (few misstatements).

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



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

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



6.3 Partial pooling

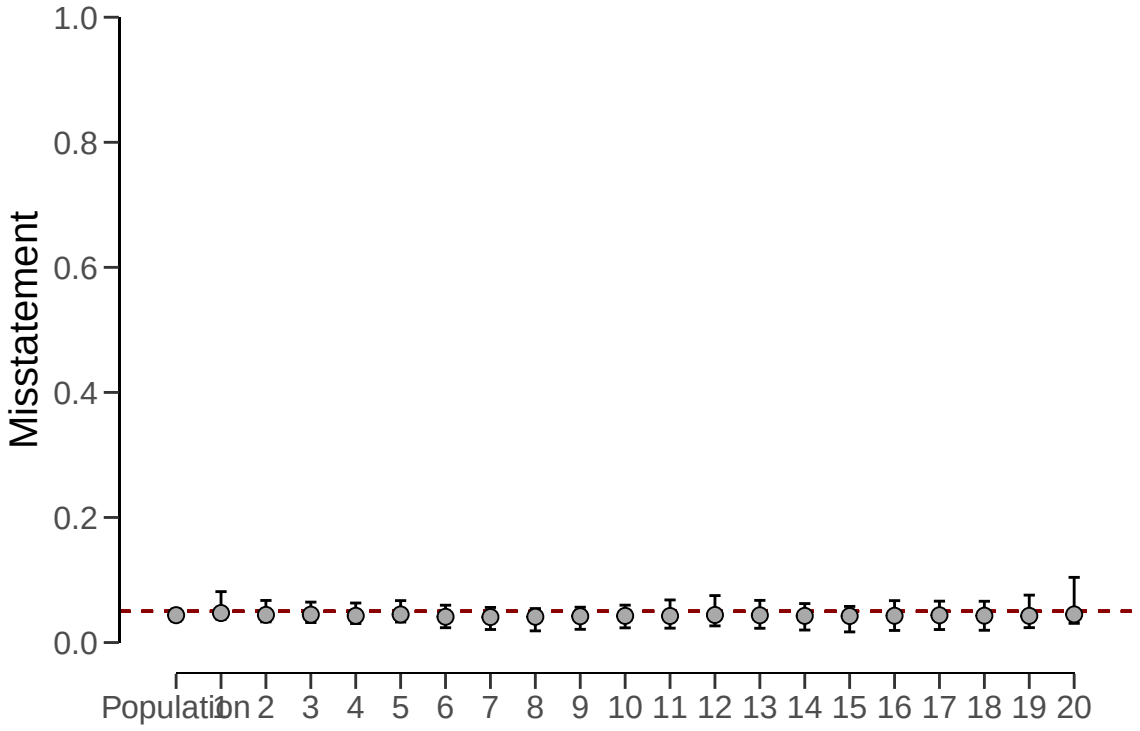
Finally, partial pooling (`pooling = "partial"`) assumes differences and similarities between strata. This allows the auditor to differentiate between strata, while also sharing information between the strata to reduce uncertainty in the population estimate. The call below evaluates the sample using a Bayesian stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate.

```
set.seed(1) # Important because the posterior distribution is determined via sampling
result_pp <- evaluation(
  materiality = 0.05, method = "binomial", prior = TRUE,
  n = retailer$samples, x = retailer$errors, N.units = retailer$items,
  alternative = "two.sided", pooling = "partial"
)
summary(result_pp)
#>
#>  Bayesian Audit Sample Evaluation Summary
#>
```

```
#> Options:
#>   Confidence level:               0.95
#>   Population size:                144000
#>   Materiality:                    0.05
#>   Hypotheses:                     H : Θ = 0.05 vs. H : Θ   0.05
#>   Method:                         binomial
#>   Prior distribution:             Determined via MCMC sampling
#>
#> Data:
#>   Sample size:                    2575
#>   Number of errors:               115
#>   Sum of taints:                  115
#>
#> Results:
#>   Posterior distribution:         Determined via MCMC sampling
#>   Most likely error:              0.043733
#>   95 percent credible interval:   [0.034497, 0.053341]
#>   Precision:                      0.0096074
#>   BF :                            0.020115
#>
#> Strata (20):
#>        N   n  x  t     mle      lb      ub precision
#> 1   5000 300 21 21 0.04717 0.03750 0.08142   0.03425
#> 2   5000 300 16 16 0.04422 0.03307 0.06730   0.02309
#> 3   5000 300 15 15 0.04460 0.03200 0.06465   0.02006
#> 4   5000 300 14 14 0.04279 0.03042 0.06322   0.02043
#> 5   5000 300 16 16 0.04490 0.03269 0.06705   0.02215
#> 6   5000 150  5  5 0.04119 0.02371 0.05970   0.01851
#> 7   5000 150  4  4 0.04075 0.02085 0.05610   0.01535
#> 8   5000 150  3  3 0.04122 0.01868 0.05434   0.01312
#> 9   5000 150  4  4 0.04203 0.02113 0.05652   0.01448
#> 10  5000 150  5  5 0.04263 0.02348 0.05977   0.01714
#> 11 10000  50  2  2 0.04246 0.02299 0.06812   0.02565
#> 12 10000  50  3  3 0.04404 0.02664 0.07512   0.03108
#> 13 10000  50  2  2 0.04360 0.02280 0.06741   0.02381
#> 14 10000  50  1  1 0.04264 0.01998 0.06230   0.01966
#> 15 10000  50  0  0 0.04242 0.01700 0.05773   0.01530
#> 16 10000  15  0  0 0.04287 0.01932 0.06700   0.02413
#> 17 10000  15  0  0 0.04361 0.02080 0.06621   0.02260
#> 18 10000  15  0  0 0.04300 0.01962 0.06601   0.02301
#> 19 10000  15  1  1 0.04270 0.02384 0.07583   0.03313
#> 20  4000  15  3  3 0.04498 0.03069 0.10421   0.05923
```

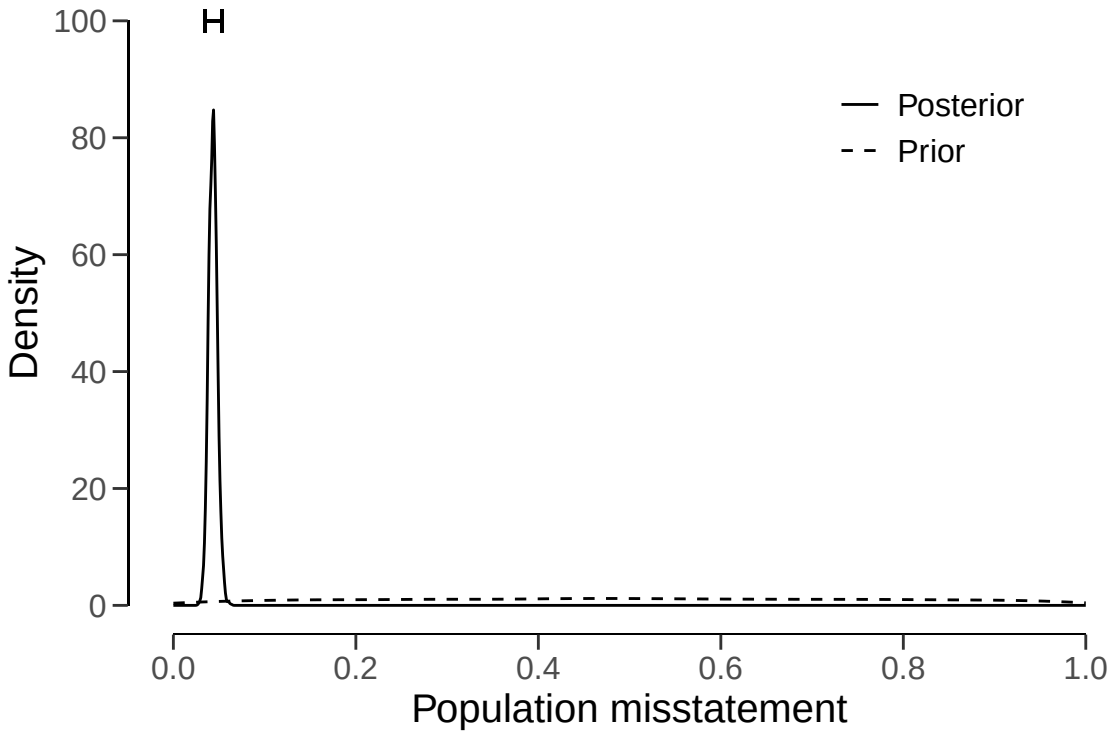
In this case, the output shows that the estimate of the misstatement in the population is 4.34%, with the 95% credible interval ranging from 3.45% to 5.33%. Note that this population estimate is substantially less uncertain than that of the no pooling approach. Note that, like in the no pooling approach, the stratum estimates are different from each other but lie closer together and are less uncertain.

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



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

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



6.4 Evaluation using data

For this example, we take the `allowances` that set that comes with the package. This data set contains 3500 financial statement line items, each with a booked value `bookValue` and, for illustrative purposes, and audited (true) value `auditValue`. Since the focus of this vignette 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 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
```

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
```

6.4.1 Classical Evaluation

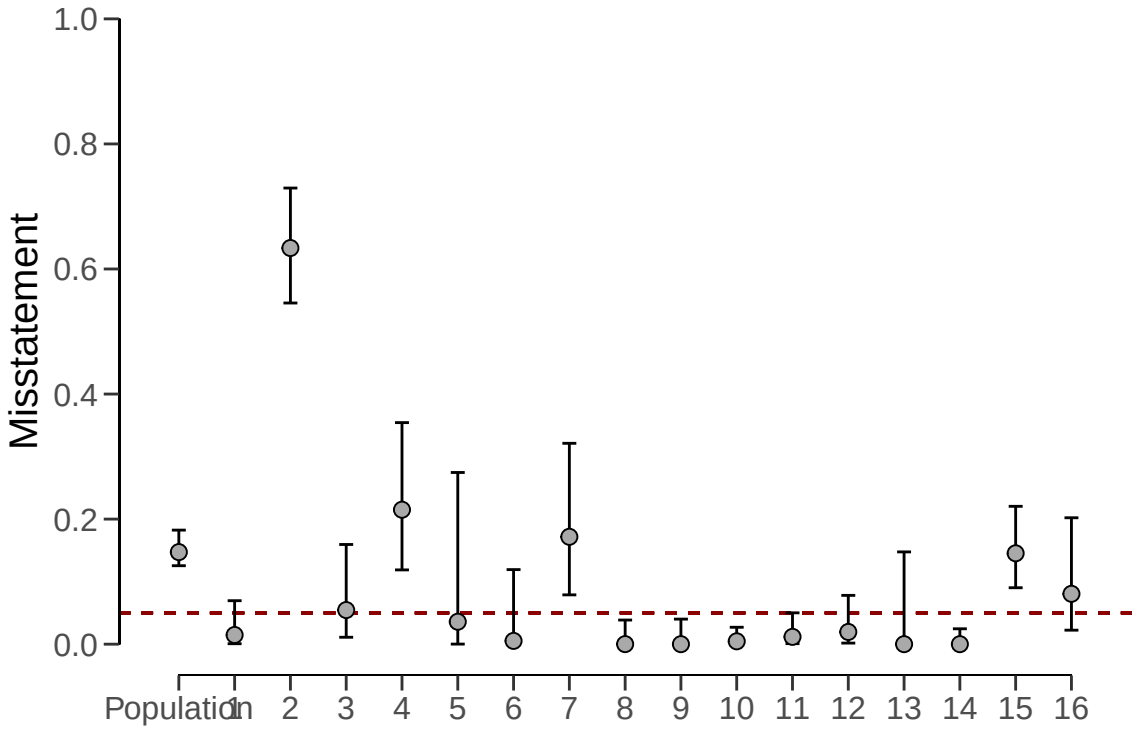
The call below evaluates the `allowances` sample using a classical stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate.

```
x <- evaluation(
  materiality = 0.05, data = allowances,
  values = "bookValue", values.audit = "auditValue", strata = "branch", times = "times",
  alternative = "two.sided", N.units = N.units
)
summary(x)
#>
#>  Classical Audit Sample Evaluation Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Population size:               16772249
#>   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:             0.14723
#>   95 percent confidence interval: [0.12549, 0.18239]
#>   Precision:                     0.03516
#>   p-value:                       NA
#>
#> Strata (16):
#>             N   n   x         t     mle      lb      ub precision p.value
#> 1   317200.09  87   6   1.27814 0.01469 0.00073 0.06950   0.05481 0.46285
```

```
#> 2  2792814.33 305 233 193.23313 0.63355 0.54558 0.72945   0.09590 0.00000
#> 3  1144231.69  55   3   3.00000 0.05455 0.01105 0.15940   0.10486 0.75827
#> 4   414202.89  70  45  15.05094 0.21501 0.11878 0.35434   0.13933 0.00000
#> 5    96660.53  18   1   0.64537 0.03585 0.00015 0.27456   0.23871 0.59343
#> 6   348006.13  34   1   0.17866 0.00525 0.00000 0.11926   0.11401 1.00000
#> 7  2384079.33  55  14   9.44448 0.17172 0.07885 0.32122   0.14950 0.00058
#> 8  1840399.33  96   1   0.00813 0.00008 0.00000 0.03860   0.03852 0.10355
#> 9   563957.70  92   0   0.00000 0.00000 0.00000 0.04010   0.04010 0.01783
#> 10 3198877.73 201   7   0.92023 0.00458 0.00009 0.02703   0.02245 0.00122
#> 11 1983299.06 128   7   1.50034 0.01172 0.00084 0.05013   0.03841 0.10773
#> 12  319144.13  86   5   1.68141 0.01955 0.00174 0.07806   0.05851 0.46069
#> 13  148905.79  25   0   0.00000 0.00000 0.00000 0.14756   0.14756 0.64187
#> 14  513058.76 150   0   0.00000 0.00000 0.00000 0.02459   0.02459 0.00134
#> 15  432007.61 150  39  21.80000 0.14533 0.09026 0.22045   0.07511 0.00001
#> 16  275403.70  52  39   4.18726 0.08052 0.02237 0.20215   0.12163 0.12258
```

In this case, the output shows that the estimate of the misstatement in the population is 14.72%, with the 95% confidence interval ranging from 12.55% to 18.26%. The precision of the population estimate is 3.54%. The stratum estimates can be seen in the output of the `summary()` function and are visualized below.

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



6.4.2 Bayesian Evaluation

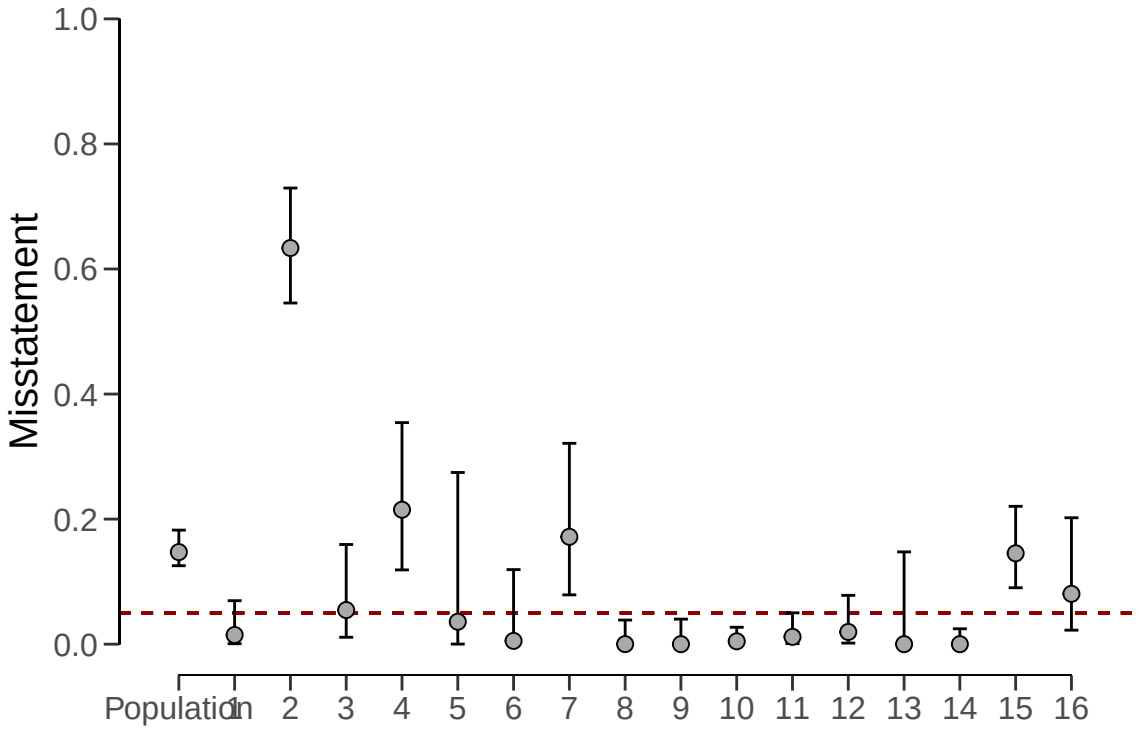
Bayesian inference can improve upon the estimates of the classical approach by pooling information between strata where possible. The call below evaluates the `allowances` sample using a Bayesian multilevel stratified evaluation procedure, in which the stratum estimates are poststratified to arrive at the population estimate.

```
# x <- evaluation(
#   materiality = 0.05, data = allowances, prior = TRUE,
#   values = "bookValue", values.audit = "auditValue", strata = "branch", times = "times",
#   alternative = "two.sided", N.units = N.units, pooling = "partial"
# )
summary(x)
#>
#>  Classical Audit Sample Evaluation Summary
#>
#> Options:
#>   Confidence level:              0.95
#>   Population size:               16772249
```

```
#>   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:              0.14723
#>   95 percent confidence interval: [0.12549, 0.18239]
#>   Precision:                      0.03516
#>   p-value:                        NA
#>
#> Strata (16):
#>             N   n   x         t     mle      lb      ub precision p.value
#> 1   317200.09  87   6   1.27814 0.01469 0.00073 0.06950   0.05481 0.46285
#> 2  2792814.33 305 233 193.23313 0.63355 0.54558 0.72945   0.09590 0.00000
#> 3  1144231.69  55   3   3.00000 0.05455 0.01105 0.15940   0.10486 0.75827
#> 4   414202.89  70  45  15.05094 0.21501 0.11878 0.35434   0.13933 0.00000
#> 5    96660.53  18   1   0.64537 0.03585 0.00015 0.27456   0.23871 0.59343
#> 6   348006.13  34   1   0.17866 0.00525 0.00000 0.11926   0.11401 1.00000
#> 7  2384079.33  55  14   9.44448 0.17172 0.07885 0.32122   0.14950 0.00058
#> 8  1840399.33  96   1   0.00813 0.00008 0.00000 0.03860   0.03852 0.10355
#> 9   563957.70  92   0   0.00000 0.00000 0.00000 0.04010   0.04010 0.01783
#> 10 3198877.73 201   7   0.92023 0.00458 0.00009 0.02703   0.02245 0.00122
#> 11 1983299.06 128   7   1.50034 0.01172 0.00084 0.05013   0.03841 0.10773
#> 12  319144.13  86   5   1.68141 0.01955 0.00174 0.07806   0.05851 0.46069
#> 13  148905.79  25   0   0.00000 0.00000 0.00000 0.14756   0.14756 0.64187
#> 14  513058.76 150   0   0.00000 0.00000 0.00000 0.02459   0.02459 0.00134
#> 15  432007.61 150  39  21.80000 0.14533 0.09026 0.22045   0.07511 0.00001
#> 16  275403.70  52  39   4.18726 0.08052 0.02237 0.20215   0.12163 0.12258
```

The output shows that the estimate of the misstatement in the population is 15.66%, with the 95% credible interval ranging from 14.59% to 17%. The precision of the population estimate is 1.34%, 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 below.

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



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

```
# plot(x, type = "posterior")
```

7 Other Software

This chapter discusses other R-related open-source software implementing statistical techniques for audit sampling.

7.1 JASP for Audit (GUI)

JASP for Audit (Derks, Swart, Wagenmakers, et al. 2021) is an add-on module for JASP (JASP Team 2022), based on the **jfa** package, that facilitates statistical audit sampling. Concretely, it contains graphical user interfaces (GUI's) 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.

7.1.1 Planning

Planning a sample in JASP for Audit works similar to how you would do it in **jfa**. For example, the screenshot below shows how to plan a minimum sample size for a performance materiality of 5% using a beta(1, 1) prior distribution, while expecting zero misstatements in the sample.

In **jfa**, these results can be reproduced via the following command:

```
planning(materiality = 0.05, likelihood = "binomial", prior = TRUE)
#>
#>  Bayesian Audit Sample Planning
#>
#> minimum sample size = 58
#> sample size obtained in 59 iterations via method 'binomial' + 'prior'
```

Figure 7.1: Figure 7.1: The JASP welcome screen.

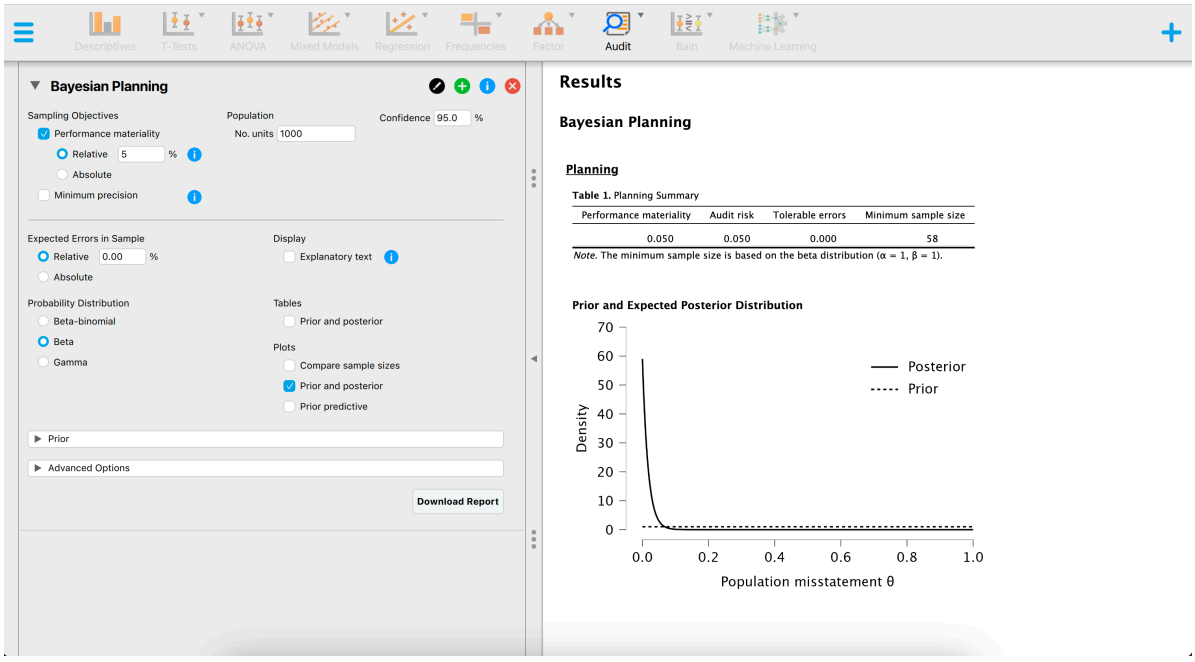


Figure 7.2: Figure 7.2: The JASP interface for planning a sample.

7.1.2 Selection

Selecting a sample in JASP for Audit works similar to how you would do it in **jfa**. For example, the screenshot below shows how to select a sample of 60 monetary units from the `BuildIt` data set that is included in the package using a fixed interval sampling method with a starting point of 1.

```
set.seed(1)
data(BuildIt)
result <- selection(data = BuildIt, size = 60, method = "interval", start = 1,
                    units = "values", values = "bookValue")
head(result$sample)
#>   row times    ID bookValue auditValue
#> 1   1     1 82884    242.61     242.61
#> 2  63     1 51272    248.40     248.40
#> 3 123     1 37985    562.09     562.09
#> 4 183     1 96080    449.07     449.07
#> 5 240     1 92819    690.08     690.08
#> 6 302     1 94296    198.59     198.59
```

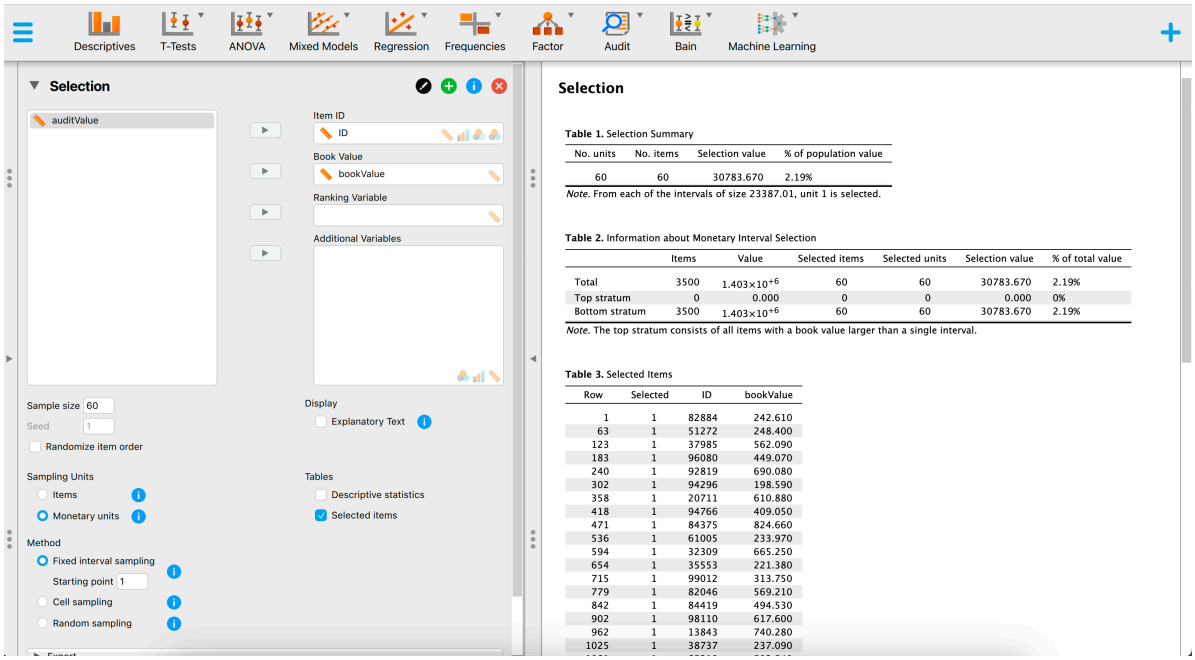


Figure 7.3: Figure 7.3: The JASP interface for selecting a sample.

7.1.3 Evaluation

Evaluating a sample in JASP for Audit works similar to how you would do it in **jfa**. For example, the screenshot below shows how to evaluating a sample of $n = 60$ containing $x = 0$ misstatements against a performance materiality of 5% using a beta(1, 1) prior distribution.

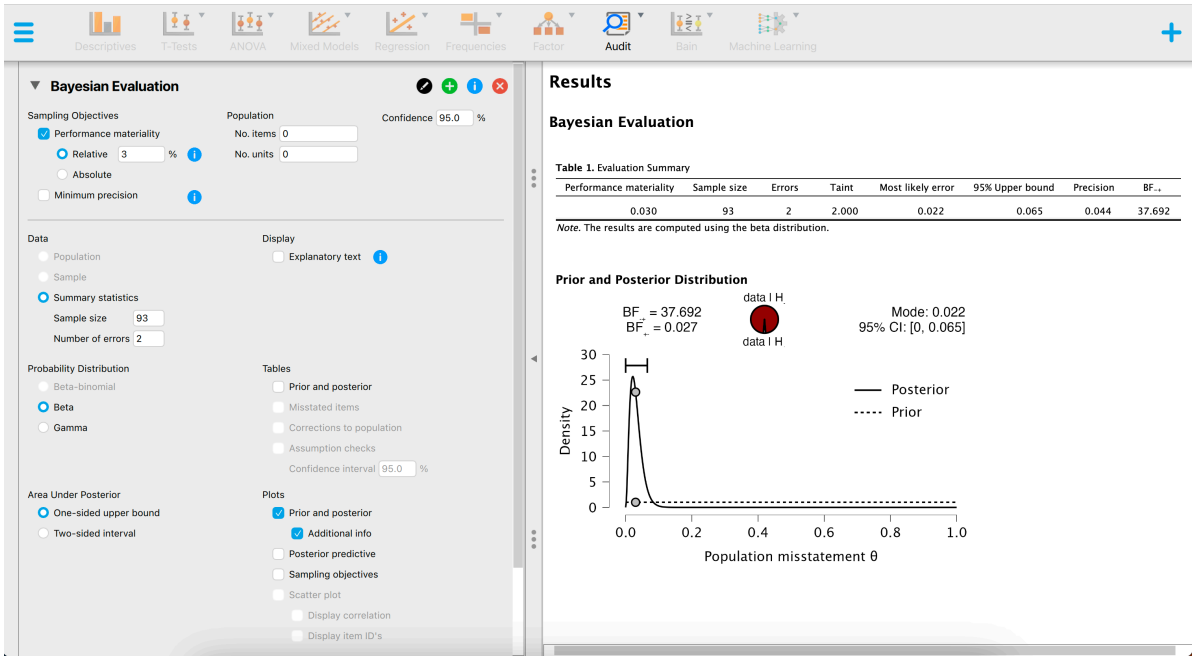


Figure 7.4: Figure 7.4: The JASP interface for evaluating a sample.

In **jfa**, these results can be reproduced via the following command:

```
evaluation(materiality = 0.05, method = "binomial", x = 2, n = 93, prior = TRUE)
#>
#>  Bayesian Audit Sample Evaluation
#>
#> data:  2 and 93
#> number of errors = 2, number of samples = 93, taint = 2, BF  =
#> 111.66
#> alternative hypothesis: true misstatement rate is less than 0.05
#> 95 percent credible interval:
#>  0.0000000 0.0654624
#> most likely estimate:
#>  0.021505
#> results obtained via method 'binomial' + 'prior'
```

7.2 MUS (R)

MUS (Prömpers and Guimarães 2019) is an R package proving 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 with:

```
install.packages("MUS")
```

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.

———. 2016b. "Appendix C: Monetary Unit Sampling Tables." In *Audit Guide: Audit Sampling*. https://doi.org/10.1002/9781119448617.app3.

Chang, Winston. 2022. *R Graphics Cookbook*. O'Reilly. https://r-graphics.org/.

Derks, Koen, Jacques de Swart, Paul van Batenburg, Eric-Jan Wagenmakers, and Ruud Wetzels. 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): 621–36. https://doi.org/10.1111/ijau.12240.

Derks, Koen, Jacques de Swart, Eric–Jan Wagenmakers, Jan Wille, and Ruud Wetzels. 2021. "JASP for Audit: Bayesian Tools for the Auditing Practice." *Journal of Open Source Software* 6 (68): 2733.

Dyer, Danny, and Rebecca L Pierce. 1993. "On the Choice of the Prior Distribution in Hypergeometric Sampling." *Communications in Statistics - Theory and Methods* 22 (8): 2125–46. https://doi.org/10.1080/03610929308831139.

Grolemund, Garrett. 2014. *Hands-On Programming with R*. https://rstudio-education.github.io/hopr/.

International Auditing and Assurance Standards Board (IAASB). 2018. "ISA 530: Audit Sampling." In *International Standards on Auditing (ISA)*.

JASP Team. 2022. "JASP (Version 0.16.4)[Computer software]." https://jasp-stats.org/.

Lin, Jonathan. 2021. *Audit Analytics with R*. https://auditanalytics.jonlin.ca/.

Prömpers, Henning, and André Guimarães. 2019. *MUS: Monetary Unit Sampling and Estimation Methods, Widely Used in Auditing*. https://CRAN.R-project.org/package=MUS.

Stewart, Trevor. 2012. "Technical Notes on the AICPA Audit Guide Audit Sampling," 5–8.

Wickam, Hadley, and Jenny Brian. 2022. *R Packages*. https://r-pkgs.org/.

Wickam, Hadley, and Garrett Grolemund. 2017. *R for Data Science*. https://r4ds.had.co.nz/.

Wilke, Claus O. 2022. *Fundamentals of Data Visualization*. https://clauswilke.com/dataviz/.