BIOSTATS

Akira Terui

Last updated on 2023-05-29

Contents

Introduction

This textbook aims to introduce fundamental statistical techniques and their applications to biological data. A unique aspect of this book is the "flipped-order" introduction. Many statistics courses start with theory; yet, I found it difficult for those unfamiliar with statistics. I will start with a real example of the method, followed by the explanation for an underlying theory/concept. The author is an ecologist, so some methods in this book might not be popular in other fields.

Chapter 1

Descriptive Statistics

Descriptive statistics are a set of summary measures that provide a concise overview of a dataset. They help us understand the characteristics and properties of the data without delving into complex statistical analyses. Some commonly used descriptive statistics include the mean, standard deviation, and median.

To illustrate the concept, let's consider fish length measurement x. Each fish is identified by a subscript, and their lengths are denoted as follows:

$$x_1 = 15.9 x_2 = 15.1 x_3 = 21.9 x_4 = 13.3 x_5 = 24.4$$

Often times, we use subscript i (or any character you like) instead of actual number to indicate a given data point. For example, we write fish length x_i for individual i. Alternatively, instead of writing each individual data point, we can represent them as a vector using boldface, denoted as $\boldsymbol{x}$. In this case, the vector $\boldsymbol{x}$ can be expressed as:

$$\boldsymbol{x} = \{15.9, 15.1, 21.9, 13.3, 24.4\}$$

Now let's see how we represent summary statistics of vector $\boldsymbol{x}$.

1.1 Central Tendency

1.1.1 Measures of Central Tendency

Central tendency measures (Table 1.1) provide insights into the typical or central value of a dataset. There are three commonly used measures:

- **Arithmetic Mean**. This is the most commonly used measure of central tendency. It represents the *additive* average of the data. To calculate the

arithmetic mean, you sum up all the values and divide the total by the number of data points. It can be heavily influenced by extreme values, known as outliers.

- **Geometric Mean**. The geometric mean is a *multiplicative* average. It is always smaller than the arithmetic mean and less sensitive to unusually large values. However, it is not applicable when the data contain negative values.
- **Median**. The median is the value that separates the higher half from the lower half of the dataset. It represents the 50th percentile data point. The median is less affected by outliers compared to the arithmetic mean. To calculate the median, you arrange the data in ascending order and select the middle value if the dataset has an odd number of values. If the dataset has an even number of values, you take the average of the two middle values.

Table 1.1: Common measures of central tendency. N refers to the number of data points.

Measure	Equation	
Arithmetic mean μ	$\frac{\sum_i^N x_i}{N}$	
Geometric mean μ_{ge}	$(\prod_i^N x_i)^{\frac{1}{N}}$	
Median μ_{med}	$x_{(\frac{N+1}{2}^{\text{th}})}$	if N is odd number
	$\frac{1}{2}[x_{(\frac{N}{2}^{\text{th}})} + x_{(\frac{N}{2}+1^{\text{th}})}]$	if N is even number

1.1.2 R Exercise

To learn more about these measures, let's create vectors $\boldsymbol{x} = \{15.9, 15.1, 21.9, 13.3, 24.4\}$ and $\boldsymbol{y} = \{15.9, 15.1, 21.9, 53.3, 24.4\}$ – $\boldsymbol{y}$ is identical to $\boldsymbol{x}$ but contains one outlier value. How does this make difference? To construct vectors in R, we use `c()`, a function that stands for "construct." Below is the script:

```
# construct vectors x and y
x <- c(15.9, 15.1, 21.9, 13.3, 24.4)
y <- c(15.9, 15.1, 21.9, 53.3, 24.4)
```

Confirm you constructed them correctly:

```
x
```

```
## [1] 15.9 15.1 21.9 13.3 24.4
```

```
y
```

```
## [1] 15.9 15.1 21.9 53.3 24.4
```

Cool! Now we can calculate summary statistics of `x` and `y`.

Arithmetic mean

While R has a function for arithmetic `mean()`, let's try to calculate the value from scratch:

```
# for vector x
n_x <- length(x) # the number of elements in x = the number of data points
sum_x <- sum(x) # summation for x
mu_x <- sum_x / n_x # arithmetic mean
print(mu_x) # print calculated value
```

```
## [1] 18.12
```

```
# for vector y; we can calculate directly too
mu_y <- sum(y) / length(y)
print(mu_y) # print calculated value
```

```
## [1] 26.12
```

Compare with outputs from `mean()` :

```
print(mean(x))
```

```
## [1] 18.12
```

```
print(mean(y))
```

```
## [1] 26.12
```

Geometric Mean

Unfortunately, there is no build-in function for geometric mean μ_{ge} in R (as far as I know; there are packages though). But, we can calculate the value from scratch again:

```
# for vector x
prod_x <- prod(x) # product of vector x; x1 * x2 * x3...
n_x <- length(x)
mug_x <- prod_x^(1 / n_x) # ^ means power
print(mug_x)
```

```
## [1] 17.63648
```

```
# for vector y
mug_y <- prod(y)^(1 / length(y))
print(mug_y)
```

```
## [1] 23.28022
```

Median

Lastly, let's do the same for median:

```
# for vector x
x <- sort(x) # sort x from small to large
index <- (length(x) + 1) / 2 # (N + 1)/2 th index as length(x) is an odd number
med_x <- x[index]
print(med_x)
```

```
## [1] 15.9
```

```
# for vector y
y <- sort(y) # sort y from small to large
med_y <- y[(length(y) + 1) / 2]
print(med_y)
```

```
## [1] 21.9
```

Compare with outputs from `median()`

```
print(median(x))
```

```
## [1] 15.9
```

```
print(median(y))
```

```
## [1] 21.9
```

1.2 Variation

1.2.1 Measures of Variation

Variation measures (Table 1.2) provide information about the spread of data points.

- **Variance.** Variance is a statistical measure that quantifies the spread or dispersion of a dataset. It provides a numerical value that indicates how far individual data points in a dataset deviate from the mean or average value. In other words, variance measures the average squared difference between each data point and the mean. **Standard deviation (SD) is the square root of variance**.
- **Inter-Quantile Range.** The interquartile range (IQR) is a statistical measure that provides information about the spread or dispersion of a dataset, specifically focusing on the middle 50% of the data. It is a robust measure of variability that is less affected by outliers compared to variance.
- **Median Absolute Deviation.** Median Absolute Deviation (MAD) is similar to variance, but provides a robust estimation of variability that is less affected by outliers compared to variance. MAD is defined as the median of the absolute deviations from the data's median.

- **Coefficient of Variation.** The coefficient of variation (CV) is a statistical measure that expresses the relative variability of a dataset in relation to its mean. It is particularly useful when comparing the variability of different datasets that have different scales or units of measurement.
- **MAD/Median.** MAD/Median is a statistical measure used to assess the relative variability of a dataset without assuming any specific distribution or parametric model. CV is sensitive to outliers because of its reliance on the arithmetic mean. However, MAD/Median is robust to this issue.

Table 1.2: Common measures of variation. N refers to the number of data points.

Measure	Equation
Variance σ^2 (standard deviation σ)	$\frac{\sum_i^N (x_i-\mu)^2}{N}$
Inter-Quantile Range IQR (x_l and x_h are l^{th} and h^{th} percentiles, respectively)	$\lvert x_l - x_h \rvert$
Median Absolute Deviation (MAD)	Median($\lvert x_i - \mu_{med} \rvert$)
Coefficient of Variation (CV)	$\frac{\sigma}{\mu}$
MAD/Median	$\frac{\text{MAD}}{\mu_{med}}$

1.2.2 R Exercise

Variance, SD, and CV

Let's try variance, SD, and CV:

```
# for x
sqd_x <- (x - mean(x))^2 # sqared deviance
sum_sqd_x <- sum(sqd_x)

var_x <- sum_sqd_x / length(x)
sd_x <- sqrt(var_x) # sqrt(): square root
cv_x <- sd_x / mean(x)

print(var_x)
```

```
## [1] 18.2016
```

```
print(sd_x)
```

```
## [1] 4.266333
```

```
print(cv_x)
```

```
## [1] 0.2354489
```

```
# for y
var_y <- sum((y - mean(y))^2) / length(y)
sd_y <- sqrt(var_y)
cv_y <- sd_y / mean(y)

print(var_y)
```

```
## [1] 197.0816
```

```
print(sd_y)
```

```
## [1] 14.03858
```

```
print(cv_y)
```

```
## [1] 0.5374646
```

IQR, MAD, and MAD/Median

Let's try IQR, MAD, and MAD/Median. IQR can be defined for given percentiles. Here, let me use 25 and 75 percentiles as x_l and x_h.

```
# for x
## IQR
x_l <- quantile(x, 0.25) # quantile(): return quantile values, 25 percentile
x_h <- quantile(x, 0.75) # quantile(): return quantile values, 75 percentile
iqr_x <- abs(x_l - x_h) # abs(): absolute value

## MAD
ad_x <- abs(x - mean(x))
mad_x <- median(ad_x)

## MAD/median
mad_m_x <- mad_x / median(x)

print(iqr_x)
```

```
## 25%
## 6.8
```

```
print(mad_x)
```

```
## [1] 3.78
```

```
print(mad_m_x)
```

```
## [1] 0.2377358
```

```
# for y
## IQR
y_q <- quantile(y, c(0.25, 0.75)) # return as a vector
```

```
iqr_y <- abs(y_q[1] - y_q[2]) # y_q[1] = 25 percentile; y_q[2] = 75 percentile

## MAD
mad_y <- median(abs(y - mean(y)))

## MAD/median
mad_m_y <- mad_y / median(y)

print(iqr_y)
```

```
## 25%
## 8.5
```

```
print(mad_y)
```

```
## [1] 10.22
```

```
print(mad_m_y)
```

```
## [1] 0.4666667
```

1.3 Laboratory

1.3.1 Comparing Central Tendency Measures

What are the differences of the three measures of central tendency? To investigate this further, let's perform the following exercise.

1. Create a new vector `z` with length 1000 as `exp(rnorm(n = 1000, mean = 0, sd = 0.1))`, and calculate the arithmetic mean, geometric mean, and median.
2. Draw a histogram of `z` using functions `tibble()`, `ggplot()`, and `geom_histogram()`.
3. Draw vertical lines of arithmetic mean, geometric mean, and median on the histogram with different colors using a function `geom_vline()` .
4. Compare the values of the central tendency measures.
5. Create a new vector `z_rev` as `-z + max(z) + 0.1`, and repeat step 1 – 4.

1.3.2 Comparing Variation Measures

Why do we have absolute (variance, SD, MAD, IQR) and relative measures (CV, MAD/Median) of variation? To understand this, suppose we have 100 measurements of fish weight in unit "gram." (`w` in the following script)

```r
w <- rnorm(100, mean = 10, sd = 1)
head(w) # show first 10 elements in w
```

```
## [1]  7.693534 12.056249  9.954456 10.489594 12.082495 12.152664
```

Using this data, perform the following exercise:

1. Convert the unit of `w` to "milligram" and create a new vector `m`.
2. Calculate SD and MAD for `w` and `m`.
3. Calculate CV and MAD/Median for `w` and `m`.

Chapter 2

Sampling Concept

"Why do I need statistics in the first place?" This was the initial question that arose when I entered the field of ecology. Initially, I assumed it would be a straightforward query with an immediate response. However, I soon realized that it is a profound question with a complex answer. In short, "we need statistics because we often possess only partial information about what we seek to understand." Now, let's explore the more elaborate explanation below.

2.1 The Unknown

2.1.1 Garden Plant Height

Consider a scenario where we are conducting a study on plant height in a garden. In this garden, there exists a thousand of individual plants, making it impractical for a single researcher to measure all of them. Instead, due to resource limitations, a sample of 10 plants was selected to ***calculate*** the average height and the extent of variation among these plant individuals:

```
## # A tibble: 10 x 3
##    plant_id height unit
##       <int>  <dbl> <chr>
##  1        1   16.9 cm
##  2        2   20.9 cm
##  3        3   15.8 cm
##  4        4   28   cm
##  5        5   21.6 cm
##  6        6   15.9 cm
##  7        7   22.4 cm
##  8        8   23.7 cm
##  9        9   22.9 cm
```

```
## 10       10   18.5 cm
```

Cool. Let's use this data set to learn about the pitfall behind this. Create a vector of plant height `h` and put it in a `tibble()` to analyze it:

```
h <- c(16.9, 20.9, 15.8, 28, 21.6, 15.9, 22.4, 23.7, 22.9, 18.5)

df_h1 <- tibble(plant_id = 1:10, # a vector from 1 to 10 by 1
                height = h, # height
                unit = "cm") # unit
```

This format (`tibble()`) is better than a raw vector of height because it allows more flexible analysis. Let's add columns of `mu_height` and `var_height` using `mutate()`, a function that adds new column(s) to an existing `tibble()` (or `data.frame()`):

```
# nrow() returns the number of rows
# while piping, "." refers to the dataframe inherited
# i.e., nrow(.) counts the number of rows in df_height
df_h1 <- df_h1 %>%
  mutate(mu_height = mean(height),
         var_height = sum((height - mu_height)^2) / nrow(.))
```

Awesome, we were able to get the average height and the variance! – **however, how confident are you?** We obtained plant height only from 10...out of 1000. Are they different if we measure another set of 10 plant individuals? Let's see:

```
## # A tibble: 10 x 3
##    plant_id height unit
##       <int>  <dbl> <chr>
##  1       11   27.6 cm
##  2       12   21.9 cm
##  3       13   16.9 cm
##  4       14    8.9 cm
##  5       15   25.6 cm
##  6       16   19.8 cm
##  7       17   19.9 cm
##  8       18   24.7 cm
##  9       19   24.1 cm
## 10       20   23   cm
```

Create another `tibble()` :

```
h <- c(27.6, 21.9, 16.9, 8.9, 25.6, 19.8, 19.9, 24.7, 24.1, 23)

df_h2 <- tibble(plant_id = 11:20, # a vector from 11 to 20 by 1
                height = h,
                unit = "cm") %>%
  mutate(mu_height = mean(height),
```

```
        var_height = sum((height - mu_height)^2) / nrow(.))

print(df_h2)
```

```
## # A tibble: 10 x 5
##    plant_id height unit  mu_height var_height
##       <int>  <dbl> <chr>     <dbl>      <dbl>
##  1       11   27.6 cm         21.2       25.8
##  2       12   21.9 cm         21.2       25.8
##  3       13   16.9 cm         21.2       25.8
##  4       14    8.9 cm         21.2       25.8
##  5       15   25.6 cm         21.2       25.8
##  6       16   19.8 cm         21.2       25.8
##  7       17   19.9 cm         21.2       25.8
##  8       18   24.7 cm         21.2       25.8
##  9       19   24.1 cm         21.2       25.8
## 10       20   23   cm         21.2       25.8
```

Wow, that's totally different.

2.1.2 Linking Part to the Whole

The exercise highlights an important takeaway: what we can determine from the above data is the average and variance of the sample, **which may not perfectly represent the characteristics of the entire garden.**

In the field of biological research, it is often impractical or impossible to sample the entire population, so we must rely on estimating the unknowns (in this case, the *mean* and *variance*) from the available samples. This is where statistics comes into play, offering a tool to infer information about the entire population based on partial information obtained from the samples.

The population mean and variance in the garden, which are the unknowns we are interested in, are referred to as "**parameters.**" These parameters *cannot be directly measured* but can be *estimated* from samples through statistical inference.

Provided that certain assumptions are met, the *sample mean* is the unbiased point estimate of the *population mean*. The **"*unbiased*"** means that the sample means – if we repeat the sampling process – are centered around the population mean. In the meantime, the sample variance – if we use the formula in Chapter 1 – is "***biased***." It tends to be smaller than the population variance.

Let's explore this concept further through simple simulations. Suppose we have data on a thousand plant individuals, although this scenario may be unrealistic in practice. However, by conducting these simulations, we can examine how different sample means and variances can deviate from the true values.

Download the data here containing height measurements of thousand individuals, and place this file under `data_raw/` in your project directory. You can load this `csv` file in R as follows:

```
# load csv data on R
df_h0 <- read_csv("data_raw/data_plant_height.csv")

# show the first 10 rows
print(df_h0)
```

```
## # A tibble: 1,000 x 4
##     ...1 plant_id height unit
##    <dbl>    <dbl>  <dbl> <chr>
##  1     1        1      1   16.9 cm
##  2     2        2      2   20.9 cm
##  3     3        3      3   15.8 cm
##  4     4        4      4   28   cm
##  5     5        5      5   21.6 cm
##  6     6        6      6   15.9 cm
##  7     7        7      7   22.4 cm
##  8     8        8      8   23.7 cm
##  9     9        9      9   22.9 cm
## 10    10       10     10   18.5 cm
## # i 990 more rows
```

Using this "virtual data" (I created this through a random value generator), we first calculate the true mean and variance (we "calculate" it because it it the entire population; the whole is known) as follows:

```
mu <- mean(df_h0$height)
sigma2 <- sum((df_h0$height - mu)^2) / nrow(df_h0)

print(mu)
```

```
## [1] 19.9426
```

```
print(sigma2)
```

```
## [1] 26.74083
```

We can simulate the sampling of 10 plant individuals by randomly selecting 10 rows from `df_h0`:

```
df_i <- df_h0 %>%
  sample_n(size = 10) # size specifies the number of rows to be selected randomly

print(df_i)
```

```
## # A tibble: 10 x 4
##     ...1 plant_id height unit
```

```
##    <dbl>    <dbl>  <dbl> <chr>
##  1   815      815   18.8 cm
##  2   923      923   21.7 cm
##  3   495      495   39.1 cm
##  4    15       15   25.6 cm
##  5   291      291   18   cm
##  6   274      274   33.2 cm
##  7   350      350   29.4 cm
##  8   516      516   21.2 cm
##  9   789      789   23.4 cm
## 10   232      232    5.6 cm
```

Since `sample_n()` selects rows randomly, you will (very likely) get different set of 10 individuals/rows every single time. Below is another set of 10 rows (notice that `df_i` is overwritten with the new data set):

```
df_i <- df_h0 %>%
  sample_n(size = 10)

print(df_i)
```

```
## # A tibble: 10 x 4
##     ...1 plant_id height unit
##    <dbl>    <dbl>  <dbl> <chr>
##  1    87       87   25.3 cm
##  2    52       52   16.9 cm
##  3   657      657   20   cm
##  4   336      336   23.7 cm
##  5   806      806   19.1 cm
##  6   335      335   18.3 cm
##  7   819      819   11.7 cm
##  8   557      557    8.3 cm
##  9   838      838   21.5 cm
## 10   911      911   20.5 cm
```

Let's obtain 100 sets of 10 plant individuals (randomly selected) and *estimate* the mean and variance in each. While we can perform random sampling one by one, this is cumbersome – at least, I do not want to do it. Instead, we can leverage a technique of `for` loop:

```
# for reproducibility
set.seed(3)

mu_i <- var_i <- NULL # create empty objects

# repeat the work in {} from i = 1 to i = 100
for (i in 1:100) {
```

```
  df_i <- df_h0 %>%
    sample_n(size = 10) # random samples of 10 individuals

  # save mean for sample set i
  mu_i[i] <- mean(df_i$height)

  # save variance for sample set i
  var_i[i] <- sum((df_i$height - mean(df_i$height))^2) / nrow(df_i)

}
```

Take a look at `mu_i` and `var_i` :

```
print(mu_i)
```

```
##   [1] 19.97 17.86 22.55 22.03 17.00 23.28 21.33 21.23 18.55 19.29 22.14 19.84
##  [13] 22.43 22.13 21.13 19.40 19.39 20.43 19.12 20.66 21.01 19.48 21.73 19.63
##  [25] 21.03 20.60 21.11 20.42 18.76 23.70 20.31 22.22 21.34 20.70 20.96 20.03
##  [37] 21.77 19.19 19.87 21.38 19.64 23.31 19.89 19.21 19.68 19.54 17.54 19.05
##  [49] 18.91 20.57 18.33 18.07 19.48 17.70 20.24 17.74 20.45 16.48 18.93 17.60
##  [61] 17.23 20.75 18.06 20.06 20.80 21.72 19.02 25.08 18.90 20.69 23.28 20.87
##  [73] 18.65 19.74 21.47 17.95 16.98 18.30 19.77 17.25 19.60 21.27 19.28 20.42
##  [85] 19.60 18.41 20.15 21.24 19.70 21.56 20.75 19.54 17.54 18.52 19.85 18.40
##  [97] 20.39 17.07 17.84 20.66
```

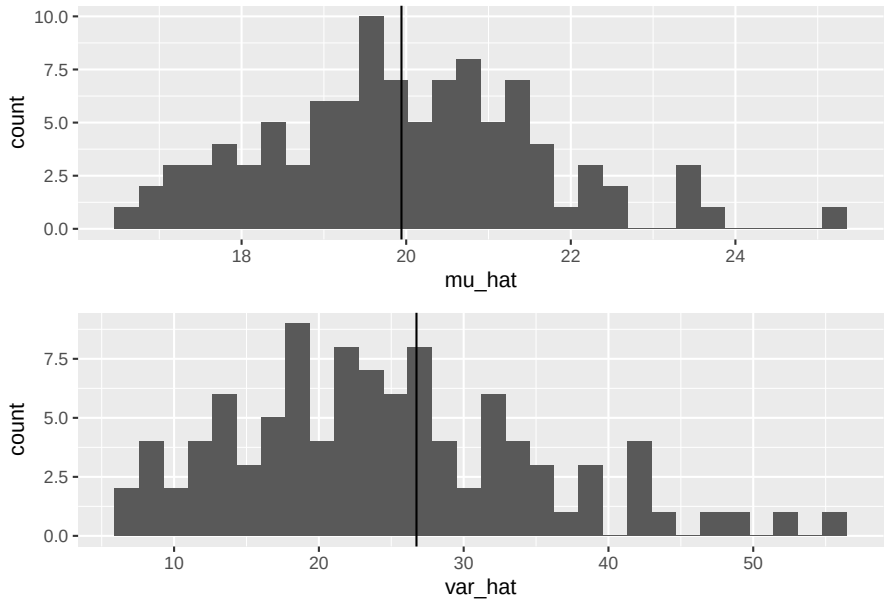
```
print(var_i)
```

```
##   [1] 14.1961 21.6884 28.2685 21.6341  6.5700 24.5996 23.6941 24.8681 21.6485
##  [10] 36.3689 17.8844 31.5784 21.0521 34.2341 20.6281 10.4420 20.0469 31.2781
##  [19] 32.0316 23.0964  9.8789 31.8916 28.3061 19.6861  8.2641 23.3640 26.8769
##  [28] 48.9416 27.7644 42.4220 38.4989 12.7076 15.9004  8.6740 11.7284 26.2061
##  [37] 21.7461 25.9269 28.0421  6.2616 16.5584 27.7489 17.3609 23.1349 38.6936
##  [46] 25.0984 13.5864 25.2625  7.7149 17.5341 11.4681 38.1561 12.6376 42.5480
##  [55] 31.9224 13.9824 14.9725 25.8296 33.6781  8.2440 31.4741 29.7805 26.7324
##  [64] 47.3704 26.8660 24.0536 29.5076 23.7376 14.8000 51.8449 13.3536 18.5581
##  [73] 18.9185 12.4544 36.2201 16.3225 17.9236 42.8760 18.5421 22.0205 34.1680
##  [82] 28.8841 26.6976 22.7356 35.2900 19.2229 34.3905 17.3784 23.1880 42.4264
##  [91] 18.5365 18.5424 21.5204 13.1276 17.7185 21.2940 27.1649 35.3141 43.5624
## [100] 55.1484
```

In each element of `mu_i` and `var_i`, we have saved estimated mean ($\hat{\mu}$; reads mu hat) and variance ($\widehat{\sigma^2}$) of 10 plant height measures for dataset `i`. By drawing a histogram of these values, we can examine the distributions of mean and variance estimates. I use R package `patchwork` to make a better figure:

```
#install.packages("patchwork") # install only once
library(patchwork)
```

```
df_sample <- tibble(mu_hat = mu_i, var_hat = var_i)

# histogram for mean
g_mu <- df_sample %>%
  ggplot(aes(x = mu_hat)) +
  geom_histogram() +
  geom_vline(xintercept = mu)

# histogram for variance
g_var <- df_sample %>%
  ggplot(aes(x = var_hat)) +
  geom_histogram() +
  geom_vline(xintercept = sigma2)

# layout vertically
# possible only if "patchwork" is loaded
g_mu / g_var
```



While sample means are indeed symmetrically distributed around the true mean, sample variances tend to be biased and skewed to the right, often underestimating the true variance.

The bias in estimating the variance arises due to inferring the parameter from a small number of samples. However, there is good news: an unbiased estimator

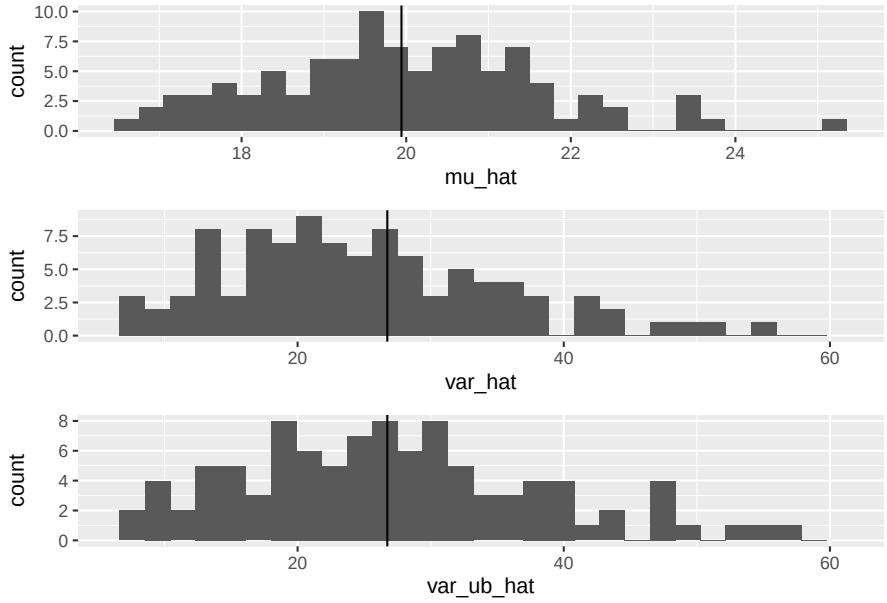
of variance exists. The formula for the unbiased estimator of variance is as follows:

$$\frac{\sum_i^N (x_i - \mu)^2}{N - 1}$$

The correction in the denominator (N replaced with $N - 1$) compensates for the bias, providing an estimate of the true variance without systematic underestimation (although this seems a simple correction, a deep math underlies the derivation of $N - 1$). This is the default formula in `var()` in R, a function used to estimate *unbiased* variance (and *unbiased* SD `sd()`). Comparison reveals how this works:

```
# for reproducibility
set.seed(3)

# redo simulations ----
mu_i <- var_i <- var_ub_i <- NULL # create empty objects

# repeat the work in {} from i = 1 to i = 100
for (i in 1:100) {

  df_i <- df_h0 %>%
    sample_n(size = 10) # random samples of 10 individuals

  # save mean for sample set i
  mu_i[i] <- mean(df_i$height)

  # save variance for sample set i
  var_i[i] <- sum((df_i$height - mean(df_i$height))^2) / nrow(df_i)

  var_ub_i[i] <- var(df_i$height)
}

# draw histograms ----
df_sample <- tibble(mu_hat = mu_i,
                    var_hat = var_i,
                    var_ub_hat = var_ub_i)

# histogram for mu
g_mu <- df_sample %>%
  ggplot(aes(x = mu_hat)) +
  geom_histogram() +
  geom_vline(xintercept = mu)

# histogram for variance
```

```
# scale_x_continuous() adjusts scale in x-axis
g_var <- df_sample %>%
  ggplot(aes(x = var_hat)) +
  geom_histogram() +
  geom_vline(xintercept = sigma2) +
  scale_x_continuous(limits= c(min(var_i, var_ub_i),
                               max(var_i, var_ub_i)))

# histogram for unbiased variance
g_var_ub <- df_sample %>%
  ggplot(aes(x = var_ub_hat)) +
  geom_histogram() +
  geom_vline(xintercept = sigma2) +
  scale_x_continuous(limits= c(min(var_i, var_ub_i),
                               max(var_i, var_ub_i)))

g_mu / g_var / g_var_ub
```



In summary, samples can only provide information about a part of the whole population. The complete picture of the entire population is often unknown, and we rely on estimating key parameters from the available samples. This concept applies to a wide range of parametric analyses in statistics, where we use sample data to make inferences about the population parameters.

Recognizing the limitations and uncertainties associated with working with samples is essential for proper statistical analysis and interpretation of results in various fields of study.

2.2 Laboratory

1. We used 10 plants to estimate sample means and variances. Obtain 100 sub-datasets with 50 and 100 measures each, and draw histograms of sample means and unbiased variances (use `var()`).
2. Sample means and unbiased variances are unbiased if samples are randomly selected. What happens if samples are non-random? Suppose the investigator was unable to find plants less than 10 cm in height – the following code excludes those less than 10 cm in height:

   ```
   df_h10 <- df_h0 %>%
     filter(height >= 10)
   ```

   Repeat step 1 with `df_h10` instead of `df_h` and compare the results.

Chapter 3

Appendix: Project Management

3.1 R Project

For the entirety of this book, I will utilize the 'R Project' as the fundamental unit of workspace, where all relevant materials such as R scripts (`.R`) and data files are gathered together. There are multiple ways to organize your project, but my preferred approach is to create a single 'R Project' for a collection of scripts and data that will contribute to a single publication (see example here). To set up an 'R Project,' you will need to have both *RStudio* and the base *R* software installed. Although *R* can be used as a standalone software, I highly recommend using it in conjunction with *RStudio* due to the latter's many features that facilitate data analysis. You can download *R* and *RStudio* from the following websites:

- R (you can select any CRAN mirror for downloading)
- RStudio

Once you launch *RStudio*, you will be greeted by the interface shown in Figure 3.1. The interface consists of three primary panels upon first opening it: the `Console`, `Environment`, and `Files`. The `Console` is where you write your code and execute calculations, data manipulation, and analysis. The `Environment` panel lists the objects you have saved, and the `Files` panel displays any files in a designated location on your computer.

```
knitr::include_graphics(here::here("image/r_image01.png"))
```

After pasting the script into the console, you should see the variable `x` appear in the environment panel.

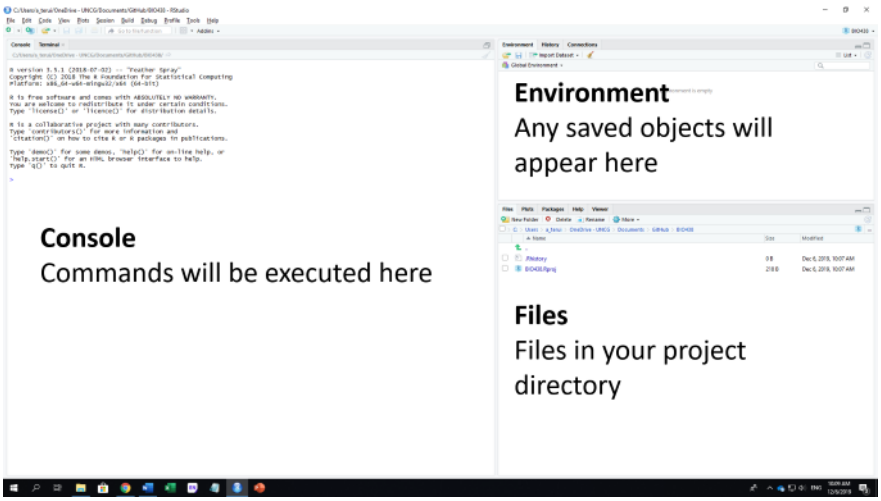


Figure 3.1: RStudio interface.

```
x <- c(1, 2)
```

`x` is an **object** where information is stored. In this case, we have stored a sequence of `1` and `2` in the object `x`. Once you have stored information in `x`, you can access it by simply typing `x`.

```
x
```

```
## [1] 1 2
```

Great! You can indeed work on your data in this manner, but it is important to note that **it is generally considered a poor practice**. As you progress through your project, you will generate a significant amount of materials, such as writing over 2000 lines of code for a single project. Therefore, it becomes crucial to implement effective code management strategies. How do you currently manage your code?

3.1.1 Script Editor

It is highly recommended to manage your scripts in the `Editor` instead. The `Editor` is where you draft and fine-tune your code before executing it in the `Console`. To create space for the `Editor`, press `Ctrl + Shift + N`. A new panel will appear in the top left corner. Let's type the following script in the `Editor`. Please note that the key combination `Ctrl + Shift + N` assumes a Windows or Linux operating system. If you're using a Mac, you can use `Command + Shift + N` instead.

```
y <- c(3, 4)
```

Then, hit `Ctr + S` to save the `Editor` file. *RStudio* will prompt you to enter the file name of the `Editor`[1].

3.1.2 File Name

It is also crucial to establish consistent naming **rules** for your files. As your project progresses, the number of files within each sub-directory may increase significantly. Without clear and consistent naming rules for your files, navigating through the project can become challenging, not only for yourself but also for others involved. To alleviate this issue, consider implementing the following recommendations for file naming:

- **NO SPACE.** Use underscore.
  - Do: `script_week1.R`
  - Don't: `script week1.R`
- **NO UPPERCASE.** Use lowercase for file names.
  - Do: `script_week1.R`
  - Don't: `Script_week1.R`
- **BE CONSISTENT.** Apply consistent naming rules within a project.
  - Do: R scripts for figures always start with a common prefix, e.g., `figure_XXX.R` `figure_YYY.R`(`XXX` and `YYY` specifies further details).
  - Don't: R scripts for figures start with random text, e.g., `XXX_fig.R` , `Figure_Y2.R` , `plotB.R`.

3.1.3 Structure Your Project

If you fail to save or haphazardly store your code files on your computer, the risk of losing essential items becomes inevitable sooner or later. To mitigate this risk, I highly recommend gathering all the relevant materials within a single `R Project`. To create a new `R Project`, follow the procedure outlined below:

a. Go to `File > New Project` on the top menu
b. Select `New Directory`
c. Select `New Project`

A new window will appear, prompting you to name a directory and select a location on your computer. To choose a location for the directory, click on the 'Browse' button. When organizing your project directories on your computer, I highly recommend creating a dedicated space. For instance, on my computer, I have a folder named `/github` where I store all my `R Project` directories.

The internal structure of an `R Project` is crucial for effective navigation and ensures clarity for both yourself and others when it is published. An `R Project` typically consists of various file types, such as `.R`, `.csv`, `.rds`, `.Rmd`, and others.

[1] In *R*, an editor file has an extension of `.R`.

Without an organized arrangement of these files, there is a high probability of encountering significant coding errors. Therefore, I place great importance on maintaining a well-structured project. In Table 3.1, I present my recommended subdirectory structure.

Table 3.1: Suggested internal structure of `R Project`

Name	Content
`README.md`	Markdown file explaining contents in the `R Project`. Can be derived from `README.Rmd`.
`/code`	Sub-directory for R scripts (`.R`).
`/data_raw`	Sub-directory for raw data before data manipulation (`.csv` or other formats). Files in this sub-directory MUST NOT be modified unless there are changes to raw data entries.
`/data_fmt`	Sub-directory for formatted data (`.csv`, `.rds`, or other formats).
`/output`	Sub-directory for result outputs (`.csv`, `.rds`, or other formats). This may include statistical estimates from linear regression models etc.
`/rmd`	(Optional) Sub-directory for Rmarkdown files (`.Rmd`). Rmarkdown allows seamless integration of R scripts and text.

3.2 Robust coding

While it is not mandatory, I highly recommend using *RStudio* in conjunction with *Git* and *GitHub*. Coding is inherently prone to errors, and even the most skilled programmers make mistakes—without exception. However, the crucial difference between beginner and advanced programmers lies in their ability to develop robust coding practices accompanied by a self-error-detection system. *Git* plays a vital role in this process. Throughout this book, I will occasionally delve into the importance of *Git* and its usage.