BIOSTATS

Akira Terui

Last updated on 2023-06-17

Contents

Introduction

This textbook is designed to provide a comprehensive introduction to fundamental statistical techniques and their practical applications in the context of biological data analysis. What sets this book apart is its "flipped-order" approach to teaching. Unlike traditional statistics courses that often begin with theoretical concepts, this book takes a different approach by initially presenting real-world data. By grounding the material in practical scenarios, it aims to make the learning process more accessible, particularly for those who are new to statistics.

Through the "flipped-order" introduction, readers will be able to engage with concrete examples of statistical methods in use before delving into the underlying theories and concepts that support them. This pedagogical approach aims to provide a solid foundation for readers to comprehend and apply statistical techniques effectively in the biological sciences.

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)	$\text{Median}(\lvert x_i - \mu_{med} \rvert)$
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
```

```r
# 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
```

```r
print(mad_y)
```

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

```r
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)

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

```
## [1]  9.237455  9.171337 10.834474  9.032348  9.971185 10.232525
```

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

"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: Garden Plant Example

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 4
##     ...1 plant_id height unit
##    <dbl>    <dbl>  <dbl> <chr>
##  1     1        1   16.9 cm
##  2     2        2   20.9 cm
##  3     3        3   15.8 cm
##  4     4        4   28   cm
##  5     5        5   21.6 cm
##  6     6        6   15.9 cm
##  7     7        7   22.4 cm
##  8     8        8   23.7 cm
##  9     9        9   22.9 cm
## 10    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_h1
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 4
##     ...1 plant_id height unit
##    <dbl>    <dbl>  <dbl> <chr>
##  1    11       11   27.6 cm
##  2    12       12   21.9 cm
##  3    13       13   16.9 cm
##  4    14       14    8.9 cm
##  5    15       15   25.6 cm
##  6    16       16   19.8 cm
##  7    17       17   19.9 cm
##  8    18       18   24.7 cm
##  9    19       19   24.1 cm
## 10    20       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.

2.2 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 unknowns we are interested in, which the population mean and variance in this example, 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   16.9 cm
##  2     2        2   20.9 cm
##  3     3        3   15.8 cm
##  4     4        4   28   cm
##  5     5        5   21.6 cm
##  6     6        6   15.9 cm
##  7     7        7   22.4 cm
##  8     8        8   23.7 cm
##  9     9        9   22.9 cm
## 10    10       10   18.5 cm
## # i 990 more rows
```

Using this synthetic dataset (I generated through a random value generator), we can calculate the true mean and variance (reference values). It is important to note that in this case, we use the term "calculate" for the mean and variance because they represent the parameters of the entire population, which are known to us in this scenario.

```
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   308      308   22.8 cm
##  2   461      461   17.7 cm
##  3   602      602   27.5 cm
##  4   181      181   13.8 cm
##  5   488      488   14.8 cm
##  6   565      565   17   cm
##  7   302      302   14.8 cm
##  8    96       96   22.8 cm
##  9   853      853   14.8 cm
## 10   987      987   15.9 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   962      962   19.9 cm
##  2   618      618   11   cm
##  3   614      614   13   cm
##  4   735      735   22.1 cm
##  5   215      215   24.9 cm
##  6    32       32   19.5 cm
##  7   230      230   21.3 cm
##  8   367      367   23.4 cm
##  9   270      270   15.3 cm
## 10   524      524   21   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
```

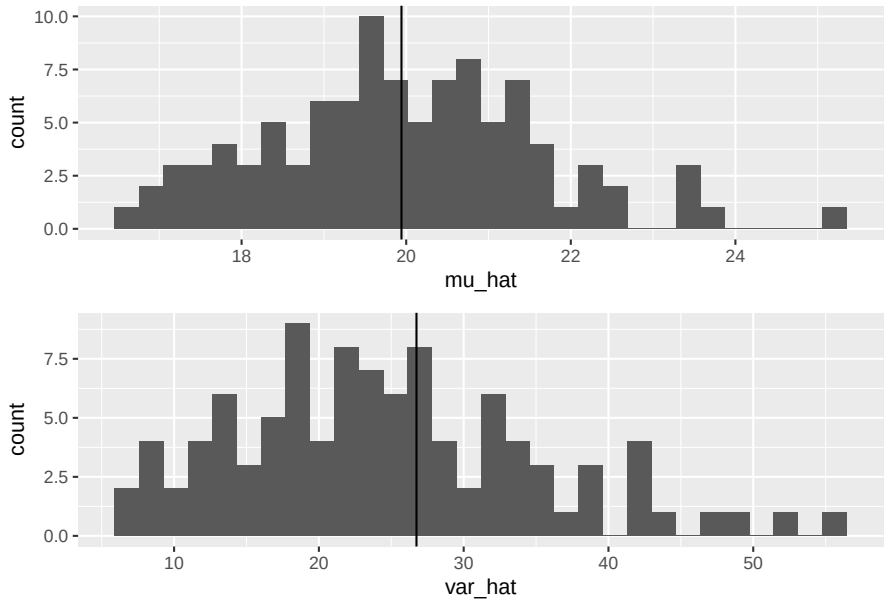
```
print(var_i)
```

```
##   [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)
```

```r
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

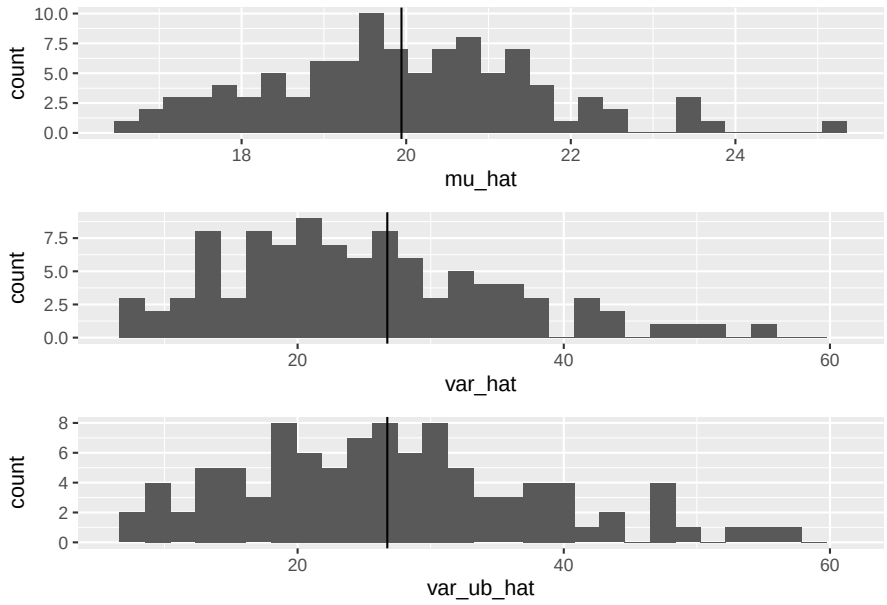
of variance exists. The formula for the unbiased estimator of variance is as follows:

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

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.

2.3 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_h0` and compare the results.

Chapter 3

Probabilistic View

Chapter 2 emphasized the concept of sampling and introduced the crucial aspect of statistics: randomness. Although the mean represents the central tendency of the data, it does not encompass all the data points. Inevitable deviations occur, and we need a way to express this "randomness." The concept of probability distributions aids in fully understanding the stochastic nature of observed data.

3.1 Probability Distribution of a Continuous Variable

3.1.1 Probability Density Function

Let me use the plant height example in Chapter 2:

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

While Chapter 2 primarily focused on exploring mean and variance, let's broaden our perspective to gain a comprehensive understanding of the entire distribution of height.

```
df_h0 %>%
  ggplot(aes(x = height)) +
  geom_histogram(binwidth = 1) + # specify binwidth
  geom_vline(aes(xintercept = mean(height))) # draw vertical line at the mean
```

This distribution comprises a thousand height measurements. However, there are certain patterns to consider. The data is centered around the mean and exhibits a symmetrical distribution. This characteristic implies that the distribution can be approximated by a simple formula that relies on only a few *key parameters*.

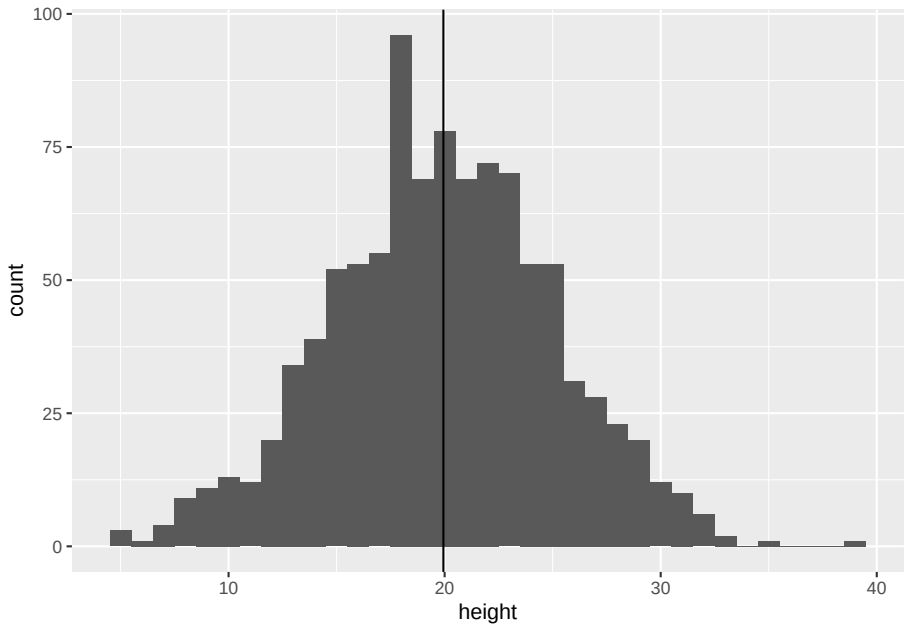


Figure 3.1: Distribution of plant height

In statistics, the symmetrical bell-shaped form is commonly approximated by a Normal distribution, often denoted as "*variable x is assumed to follow a Normal distribution.*" We express this using the following mathematical expression:

$$x \sim Normal(\mu, \sigma^2)$$

Unlike an "equation," we do not use the "=" sign because variable x is not equivalent to the Normal distribution. Rather, it represents a "stochastic" relationship that signifies the probability of x taking on a specific range of values.

A Normal distribution is characterized by two parameters: the mean and the variance. Once these parameters are determined, the formula that defines the Normal distribution will yield the probability density for a given range of values. This formula $f(x)$ is known as the **probability density function (PDF)** and is displayed below:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

3.1.2 PDF to frequency distribution

In R, the function `dnorm()` can be used to calculate the probability density for a given value. The first argument, `x`, should be a vector of values for which you want to calculate the probability density. The second argument, `mean`, and the third argument, `sd`, correspond to the mean and standard deviation of the distribution, respectively. Note that we must provide `mean` and `sd` to calculate the probability density.

To encompass the entire range of observed heights, we can use $\min(x)$ and $\max(x)$ as the lower and upper limits, respectively.

```
# vector of x values
# seq() generate min to max values with specified numbers of elements or interval
# the following produce 100 elements
x <- seq(min(df_h0$height), max(df_h0$height), length = 100)

# calculate probability density
mu <- mean(df_h0$height)
sigma <- sd(df_h0$height)
pd <- dnorm(x, mean = mu, sd = sigma)

# figure
tibble(y = pd, x = x) %>% # data frame
  ggplot(aes(x = x, y = y)) +
  geom_line() + # draw lines
  labs(y = "Probability density") # re-label
```

The shape of the curve appears quite similar to what we observed; however, the scale of the y-axis is different. This is because the y-axis represents "probability density." To convert it into actual "probability," we need to calculate the area under the curve. In R, we can utilize the `pnorm()` function for this purpose. It calculates the probability of a variable being less than the specified value, which is provided in the first argument `q`.

```
# probability of x < 10
p10 <- pnorm(q = 10, mean = mu, sd = sigma)
print(p10)
```

```
## [1] 0.02731905
```

```
# probability of x < 20
p20 <- pnorm(q = 20, mean = mu, sd = sigma)
print(p20)
```

```
## [1] 0.504426
```

```
# probability of 10 < x < 20
p20_10 <- p20 - p10
```

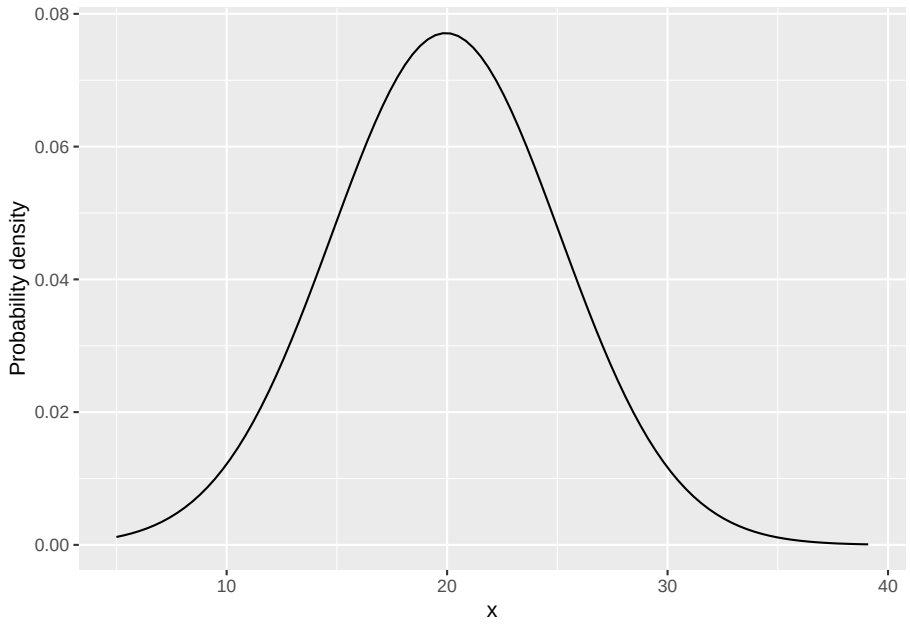


Figure 3.2: Probability density function of a Normal distribution

```
print(p20_10)
```

```
## [1] 0.4771069
```

To make the estimates comparable to the frequency data, you can calculate the probability for each 1 cm bin. The expected frequency can be obtained by multiplying the probability by the sample size, which in this case is 1000. This allows you to estimate the number of observations you would expect in each 1 cm bin based on the calculated probabilities.

```
x_min <- floor(min(df_h0$height)) # floor takes the integer part of the value
x_max <- ceiling(max(df_h0$height)) # ceiling takes the next closest integer
bin <- seq(x_min, x_max, by = 1) # each bin has 1cm

p <- NULL # empty object for probability
for (i in 1:(length(bin) - 1)) {
  p[i] <- pnorm(bin[i+1], mean = mu, sd = sigma) - pnorm(bin[i], mean = mu, sd = sigma)
}

# data frame for probability
# bin: last element [-length(bin)] was removed to match length
# expected frequency in each bin is "prob times sample size"
df_prob <- tibble(p, bin = bin[-length(bin)]) %>%
```

```
  mutate(freq = p * nrow(df_h0))
```

Overlay:

```
df_h0 %>%
  ggplot(aes(x = height)) +
  geom_histogram(binwidth = 1) +
  geom_point(data = df_prob,
             aes(y = freq,
                 x = bin),
             color = "salmon") +
  geom_line(data = df_prob,
            aes(y = freq,
                x = bin),
            color = "salmon")
```

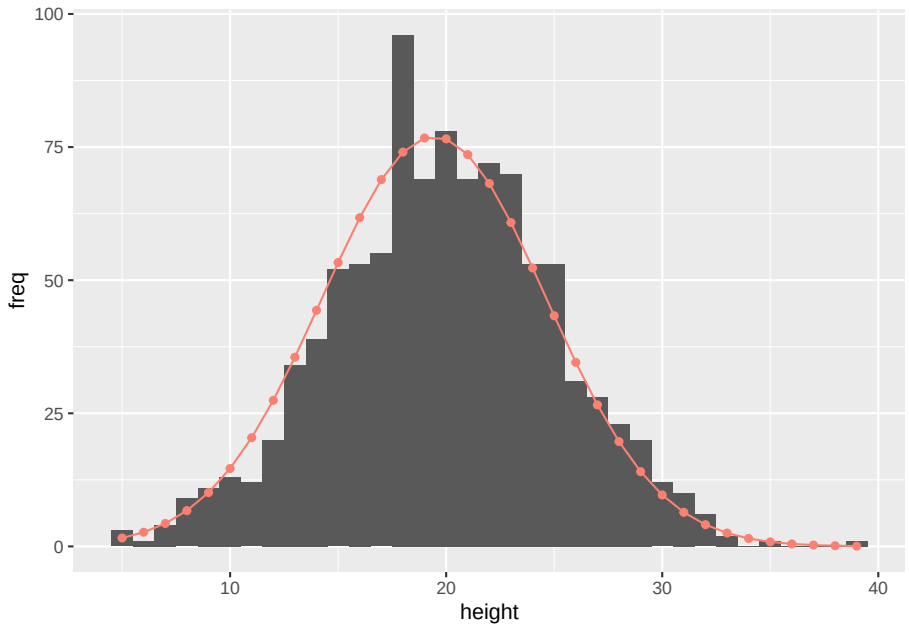


Figure 3.3: Histogram overlaid with predicted frequency (red dots and line)

It is remarkable that the probability distribution, characterized by just two parameters (mean and variance), can effectively reproduce the overall shape observed in the original data set of 1000 data points. This is great news because it means that once we determine these key parameters, we can infer the general pattern of the data.

3.2 Probability Distribution of a Discrete Variable

3.2.1 Probability Mass Function

Let's shift our perspective and examine the density of plants in the garden. For this analysis, we have counted the number of plant individuals within 30 plots, with each plot covering an area of one square meter (refer to Figure 3.4).

Download the data here and load it onto R:

```
df_count <- read_csv("data_raw/data_garden_count.csv")
print(df_count)
```

```
## # A tibble: 30 x 2
##     plot count
##    <dbl> <dbl>
##  1     1     3
##  2     2     2
##  3     3     3
##  4     4     1
##  5     5     2
##  6     6     1
##  7     7     3
##  8     8     3
##  9     9     3
## 10    10     1
## # i 20 more rows
```

Make a histogram:

```
df_count %>%
  ggplot(aes(x = count)) +
  geom_histogram(binwidth = 0.5, # define binwidth
                 center = 0) # relative position of each bin
```

There are several significant differences compared to the plant height example when considering the density of plants in the garden:

1. The possible values are discrete (count data) since we are counting the number of plant individuals in each plot.
2. The possible values are always positive, as counts cannot be negative.
3. The distribution appears to be non-symmetric around the sample mean.

Given these characteristics, a Normal distribution may not be an appropriate choice for representing such a variable. Instead, it would be more suitable to use a Poisson distribution to characterize the observed distribution of the discrete variable.

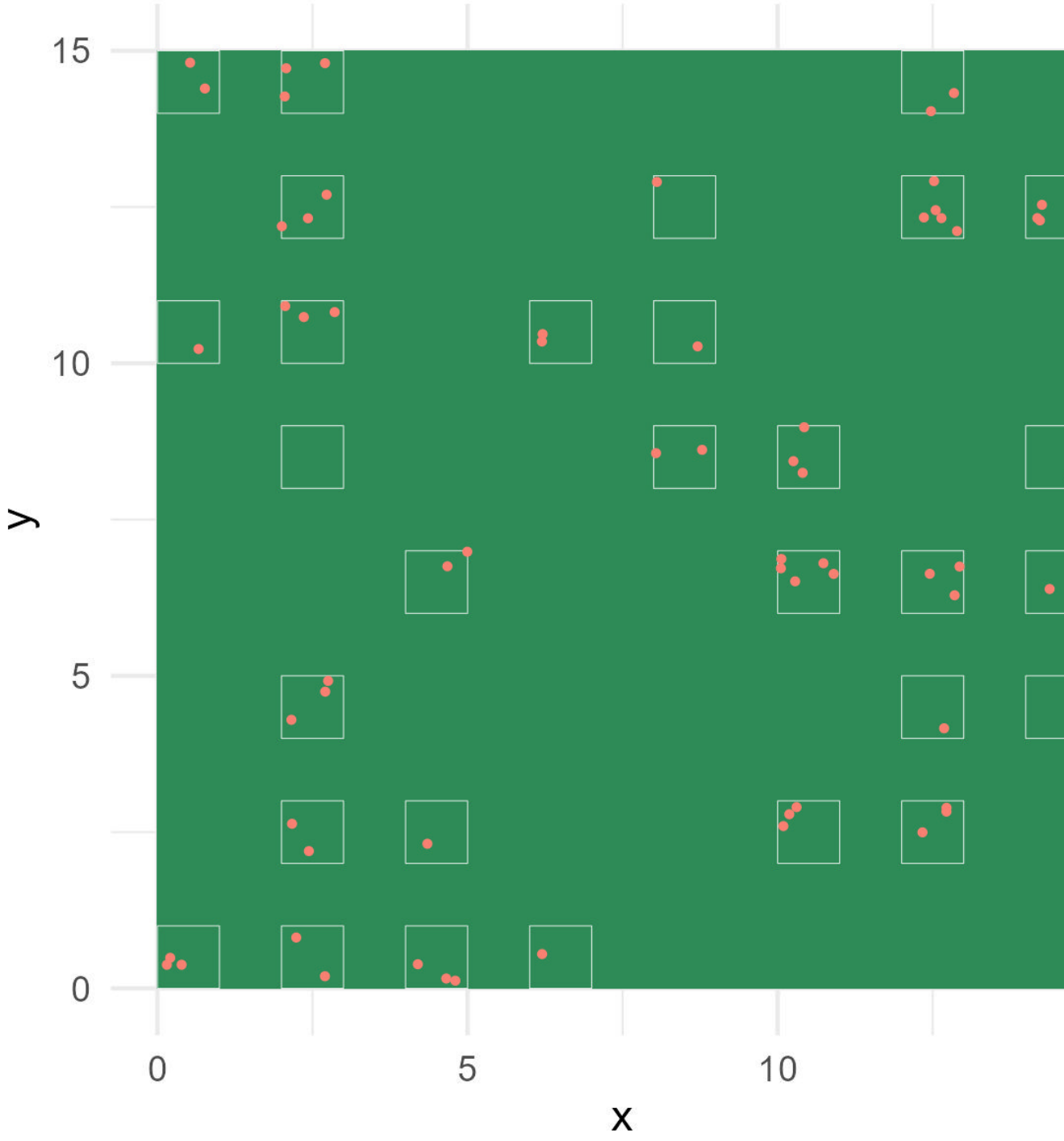


Figure 3.4: Garden view with sampling plots. White squares represent plots. Red dots indicate plant individuals counted.

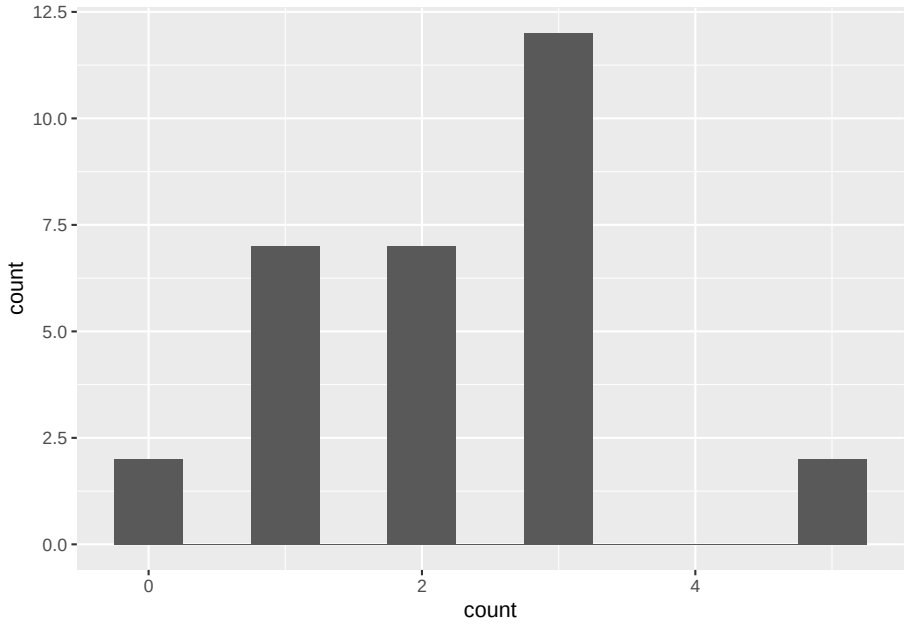


Figure 3.5: Histogram of plant individuals per plot

$$x \sim Poisson(\lambda)$$

In a Poisson distribution, the mean parameter (λ) serves as the sole parameter[1]. Probability distributions that describe discrete variables are expressed using a **probability mass function** (PMF):

$$g(x) = \frac{\lambda^x \exp(-\lambda)}{x!}$$

Unlike a probability density function (PDF), a probability mass function (PMF) represents the probability of a discrete variable x taking a specific value. For instance, the probability of $x = 2$, denoted as $\Pr(x = 2)$, can be calculated as follows:

$$\Pr(x = 2) = g(2) = \frac{\lambda^2 \exp(-\lambda)}{2!}$$

In a generalized form, we write:

[1] In a Poisson distribution, mean = variance

$$\Pr(x = k) = g(k) = \frac{\lambda^k \exp(-\lambda)}{k!}$$

3.2.2 PMF to frequency distribution

In R, you can use the `dpois()` function to visualize a Poisson distribution. This function allows you to plot the probability mass function (PMF) of the Poisson distribution and gain insights into the distribution of the discrete variable.

```
# vector of x values
# create a vector of 0 to 10 with an interval one
# must be integer of > 0
x <- seq(0, 10, by = 1)

# calculate probability mass
lambda_hat <- mean(df_count$count)
pm <- dpois(x, lambda = lambda_hat)

# figure
tibble(y = pm, x = x) %>% # data frame
  ggplot(aes(x = x, y = y)) +
  geom_line(linetype = "dashed") + # draw dashed lines
  geom_point() + # draw points
  labs(y = "Probability",
       x = "Count") # re-label
```

To convert the y-axis from probability to frequency, multiply the probabilities by the sample size (total number of observations) to obtain the expected frequency. As in the plant height example, you can plot the expected frequency on the histogram, providing a visual representation of the distribution of the discrete variable.

```
df_prob <- tibble(x = x, y = pm) %>%
  mutate(freq = y * nrow(df_count)) # prob x sample size

df_count %>%
  ggplot(aes(x = count)) +
  geom_histogram(binwidth = 0.5,
                 center = 0) +
  geom_line(data = df_prob,
            aes(x = x,
                y = freq),
            linetype = "dashed") +
  geom_point(data = df_prob,
             aes(x = x,
                 y = freq))
```

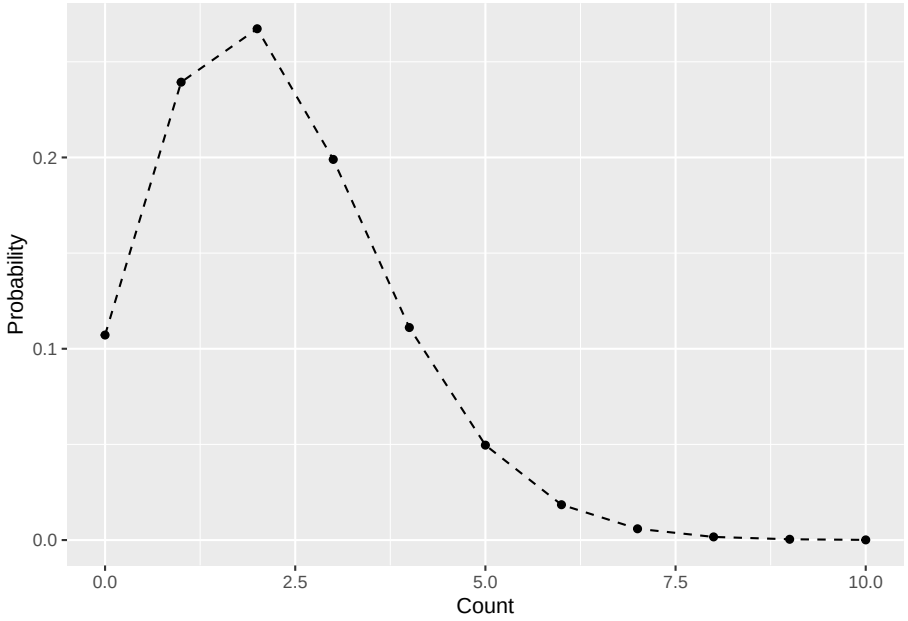


Figure 3.6: Example of a Poisson distribution

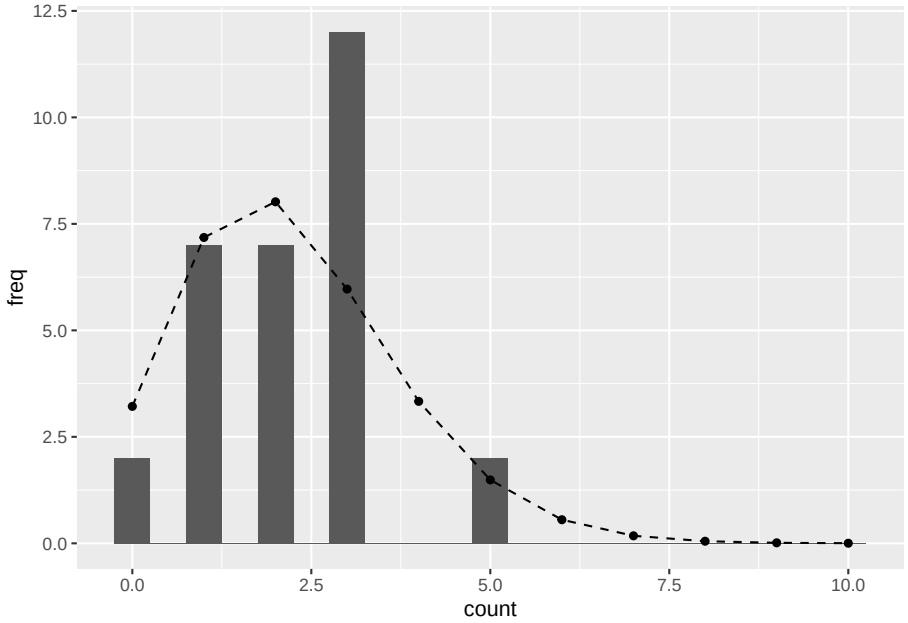


Figure 3.7: Observed histogram of the discrete variable overlaid with the Poisson expectation.

Cool, the Poisson distribution does an excellent job of characterizing the distribution of plant counts.

3.3 Why Probability Distributions?

As we have seen in the above examples, probability distributions provide a framework for describing the likelihood of different outcomes or events. By utilizing probability distributions, we can:

1. Make predictions and estimate probabilities of specific events or ranges of values.
2. Determine the most likely values or outcomes based on available information.
3. Quantify and analyze the uncertainty associated with random variables or processes.

Probability distributions serve as fundamental tools in statistics, enabling us to model, analyze, and make informed decisions in the face of uncertainty.

3.4 Laboratory

3.4.1 Normal Distribution

The function `rnorm()` produces a random variable that follows a Normal distribution with a specified mean and SD. Using this function,

1. Generate a variable with 50 observations.
2. Create a figure similar to Figure 3.3

3.4.2 Poisson Distribution

The function `rpois()` produces a random variable that follows a Poisson distribution with a specified mean. Using this function,

1. Generate a variable with 1000 observations.
2. Create a figure similar to Figure 3.7

Chapter 4

Two-Group Comparison

Data consists of "samples" extracted from the population of interest, but it's important to acknowledge that these samples are not flawless replicas of the entire population. Given this imperfect information, how can we effectively investigate and discern the distinction between two populations? In this Chapter, I will introduce one of the most basic statistical tests – "t-test." The t-test takes the following steps:

1. Define a **test statistic** to signify the difference between groups (t-statistic).
2. Define a probability distribution of t-statistic under the **null hypothesis**, i.e., a scenario that we assume no difference between groups.
3. Estimate the probability of yielding greater or less than the observed test statistic.

4.1 Explore Data Structure

Suppose that we study fish populations of the same species in two lakes (Lake `a` and `b`). These lakes are in stark contrast of productivity (Lake `b` looks more productive), and our interest is the difference in *mean* body size between the two lakes. We obtained 50 data points of fish length from each lake (download here). Save under `data_raw/` and use `read_csv()` to read data:

```
library(tidyverse) # call add-in packages everytime you open new R session
df_fl <- read_csv("data_raw/data_fish_length.csv")
print(df_fl)
```

```
## # A tibble: 100 x 3
##    lake  length unit
##    <chr>  <dbl> <chr>
##  1 a       10.8 cm
```

```
##  2 a       13.6 cm
##  3 a       10.1 cm
##  4 a       18.6 cm
##  5 a       14.2 cm
##  6 a       10.1 cm
##  7 a       14.7 cm
##  8 a       15.6 cm
##  9 a       15   cm
## 10 a       11.9 cm
## # i 90 more rows
```

In this data frame, fish length data from lake `a` and `b` are recorded. Confirm this with `unique()` or `distinct()` function:

```
# unique returns unique values as a vector
unique(df_fl$lake)
```

```
## [1] "a" "b"
```

```
# distinct returns unique values as a tibble
distinct(df_fl, lake)
```

```
## # A tibble: 2 x 1
##   lake
##   <chr>
## 1 a
## 2 b
```

Visualization provides a powerful tool for summarizing data effectively. By plotting individual data points overlaid with mean values and error bars, we can observe the distribution and patterns within the data, allowing us to identify trends, variations, and potential outliers:

```
# group mean and sd
df_fl_mu <- df_fl %>%
  group_by(lake) %>% # group operation
  summarize(mu_l = mean(length), # summarize by mean()
            sd_l = sd(length)) # summarize with sd()

# plot
# geom_jitter() plot data points with scatter
# geom_segment() draw lines
# geom_point() draw points
df_fl %>%
  ggplot(aes(x = lake,
             y = length)) +
  geom_jitter(width = 0.1, # scatter width
              height = 0, # scatter height (no scatter with zero)
              alpha = 0.25) + # transparency of data points
```

```
  geom_segment(data = df_fl_mu, # switch data frame
               aes(x = lake,
                   xend = lake,
                   y = mu_l - sd_l,
                   yend = mu_l + sd_l)) +
  geom_point(data = df_fl_mu, # switch data frame
             aes(x = lake,
                 y = mu_l),
             size = 3) +
  labs(x = "Lake", # x label
       y = "Fish body length") # y label
```

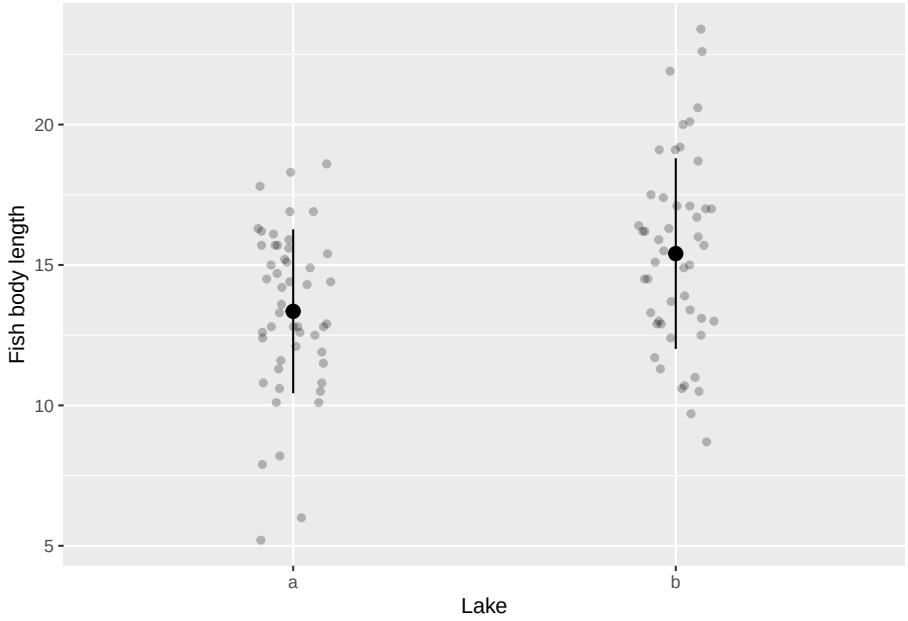


Figure 4.1: Example of mean and SD plot

Hmm, it is possible that there is a noticeable disparity in body size between Lake `a` and Lake `b`. However, how can we ascertain and provide evidence for this claim?

4.2 Test Statistic

4.2.1 t-statistic

Since our focus is on examining the difference in means, it is logical to estimate the disparity between the sample means in each lake. Let me denote the sample

means as $\hat{\mu}_a$ and $\hat{\mu}_b$ for Lake `a` and Lake `b`, respectively. We can estimate the difference between the means using the following approach:

```
# take another look at df_fl_mu
print(df_fl_mu)
```

```
## # A tibble: 2 x 3
##   lake   mu_l  sd_l
##   <chr> <dbl> <dbl>
## 1 a      13.4  2.92
## 2 b      15.4  3.39
```

```
# pull mu_l from tibble as vector
v_mu <- df_fl_mu %>%
  pull(mu_l)

# lake a
print(v_mu[1])
```

```
## [1] 13.35
```

```
# lake b
print(v_mu[2])
```

```
## [1] 15.406
```

```
# difference
v_mu[1] - v_mu[2]
```

```
## [1] -2.056
```

The average fish body size in Lake `b` is approximately 2 cm larger than that in Lake `a`. However, it is crucial to recognize that we are still lacking vital information, specifically, the uncertainty associated with this difference. As discussed in Chapter 2, we must acknowledge that sample means are not flawless representations of the entire population.

Fortunately, we can utilize sample variances to address such uncertainties. The **t-statistic** is a common indicator of the difference of means that takes into consideration the uncertainty associated with sample means.

$$t = \frac{\hat{\mu_a} - \hat{\mu_b}}{\sqrt{\hat{\sigma}_p^2 \left(\frac{1}{N_a} + \frac{1}{N_b}\right)}}$$

where N_a and N_b are sample sizes in Lake `a` and `b` (i.e., $N_a = N_b = 50$ in this specific example), and $\hat{\sigma}_p^2$ is the weighted mean of sample variances:

$$\hat{\sigma}_p^2 = \frac{N_a - 1}{N_a + N_b - 2}\hat{\sigma}_a^2 + \frac{N_b - 1}{N_a + N_b - 2}\hat{\sigma}_b^2$$

We can calculate this value in R manually:

```
# group mean, variance, and sample size
df_t <- df_fl %>%
  group_by(lake) %>% # group operation
  summarize(mu_l = mean(length), # summarize by mean()
            var_l = var(length), # summarize with sd()
            n = n()) # count number of rows per group

print(df_t)
```

```
## # A tibble: 2 x 4
##   lake   mu_l var_l     n
##   <chr> <dbl> <dbl> <int>
## 1 a      13.4  8.52    50
## 2 b      15.4 11.5     50
```

```
# pull values as a vector
v_mu <- pull(df_t, mu_l)
v_var <- pull(df_t, var_l)
v_n <- pull(df_t, n)

var_p <- ((v_n[1] - 1)/(sum(v_n) - 2)) * v_var[1] +
  ((v_n[2] - 1)/(sum(v_n) - 2)) * v_var[2]

t_value <- (v_mu[1] - v_mu[2]) / sqrt(var_p * ((1 / v_n[1]) + (1 / v_n[2])))

print(t_value)
```

```
## [1] -3.247322
```

The difference in sample means were -2.06; therefore, t-statistic further emphasizes the distinction between the study lakes. This occurrence can be attributed to the t-statistic formula.

In the denominator, we observe the inclusion of $\hat{\sigma}_p^2$ (estimated pooled variance), as well as the inverses of sample sizes N_a and N_b. Consequently, as $\hat{\sigma}_p^2$ decreases and/or the sample sizes increase, the t-statistic increases. This is a reasonable outcome since a decrease in variance and/or an increase in sample size enhance the certainty of the mean difference.

4.2.2 Null Hypothesis

The observed t-statistic serves a dual purpose in accounting for both the disparity of means and the accompanying uncertainty. However, the significance of this t-statistic remains unclear.

To address this, the concept of the **Null Hypothesis** is utilized to substantiate the observed t-statistic. The Null Hypothesis, no difference between groups

or $\mu_a = \mu_b$, allows us to draw the probability distribution of the test-statistic. Once we know the probability distribution, we can ***estimate the probability of observing the given t-statistic by random chance*** if there were no difference in mean body size between the lakes.

Let's begin by assuming that there is no difference in the mean body size of fish populations between the lakes, meaning the true difference in means is zero ($\mu_a - \mu_b = 0$; without hats in this formula!). Under this assumption, we can define the probability distribution of t-statistics, which represents how t-statistics are distributed across a range of possible values.

This probability distribution is known as the Student's t-distribution and is characterized by three parameters: the mean, the variance, and the degrees of freedom (d.f.). Considering that we are evaluating the difference in body size as the test statistic, the mean (of the difference) is assumed to be zero. The variance is estimated using $\hat{\sigma}_p^2$, and the degrees of freedom are determined by $N_a + N_b - 2$ (where d.f. can be considered as a measure related to the sample size[1]).

R has a function to draw the Student's t-distribution; let's try that out:

```
# produce 500 values from -5 to 5 with equal interval
x <- seq(-5, 5, length = 500)

# probability density of t-statistics with df = sum(v_n) - 2
y <- dt(x, df = sum(v_n) - 2)

# draw figure
tibble(x, y) %>%
  ggplot(aes(x = x,
             y = y)) +
  geom_line() +
  labs(y = "Probability density",
       x = "t-statistic")
```

The probability density in a distribution (Figure 4.2) determines the likelihood of observing a particular t-statistic. Higher probability density indicates a greater likelihood of the t-statistic occurring. Compare the observed t-statistic against this null distribution:

```
# draw entire range
tibble(x, y) %>%
  ggplot(aes(x = x,
             y = y)) +
  geom_line() +
  geom_vline(xintercept = t_value,
```

[1] Discussing the concept of degrees of freedom in depth would require a more comprehensive explanation, so I will refer to it here as a quantity relevant to the sample size.

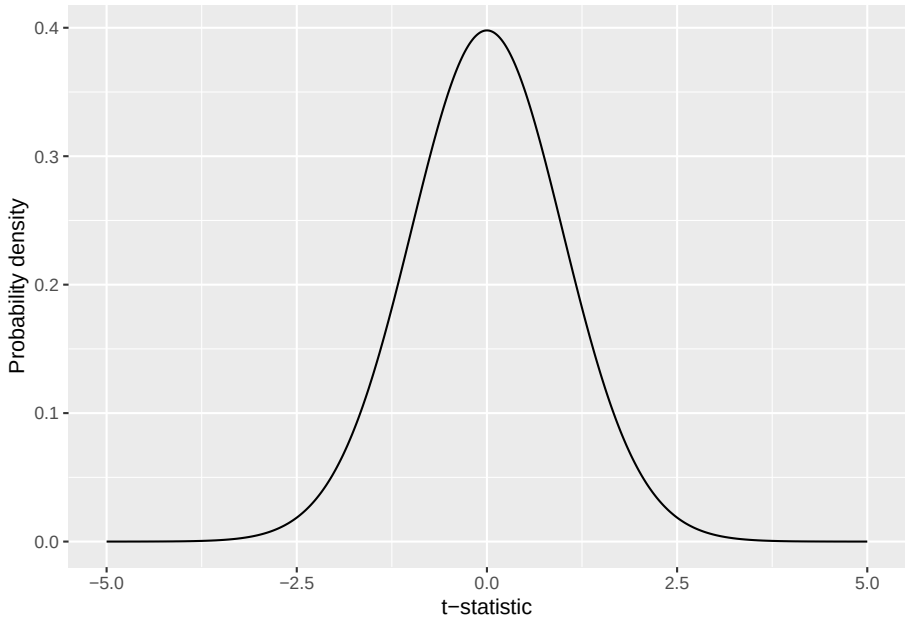


Figure 4.2: Distribution of t-statistics under null hypothesis. The probability density in a distribution determines the likelihood of observing a particular t-statistic. Higher probability density indicates a greater likelihood of the t-statistic occurring.

```
            color = "salmon") + # t_value is the observed t_value
  geom_vline(xintercept = abs(t_value),
            color = "salmon") + # t_value is the observed t_value
  labs(y = "Probability density",
       x = "t-statistic")
```

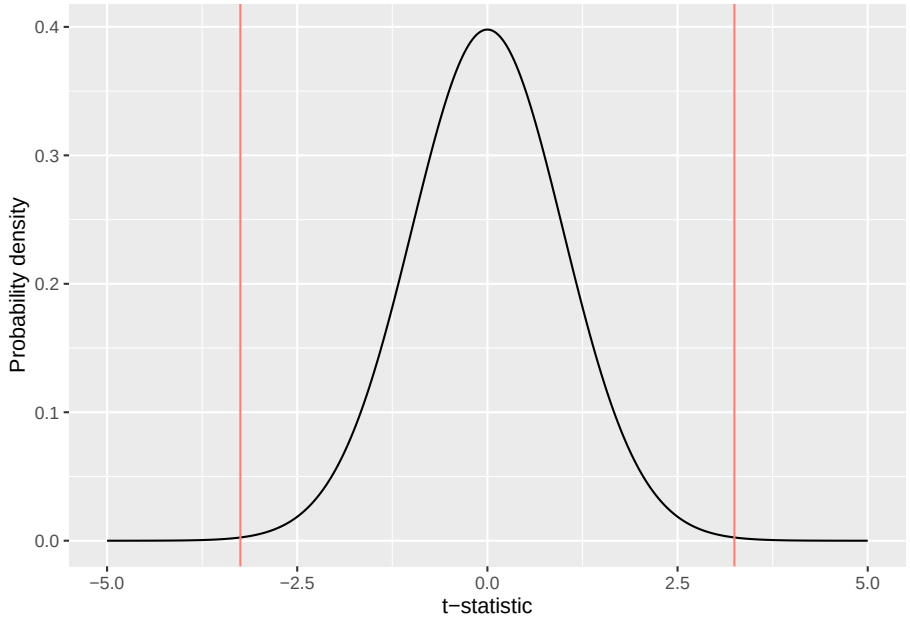


Figure 4.3: Observed t-statistic in comparison to the null distrubution

In a probability distribution, the area under the curve corresponds to the probability. Notably, **the area under the curve that falls below or above the observed t-statistic (indicated by vertical red lines) is very small (Figure** 4.3**)**, meaning that the observed difference in body size is very unlikely to occur under the null hypothesis (no difference in true means).

The function `pt()` allows us to calculate the area under the curve:

```
# calculate area under the curve from -infinity to t_value
pr_below <- pt(q = t_value, df = sum(v_n) - 2)

# calculate area under the curve from abs(t_value) to infinity
pr_above <- 1 - pt(q = abs(t_value), df = sum(v_n) - 2)
```

The **p-value,** i.e., *the probability of observing t-statistics* less than (`pr_below`) or greater than (`pr_above`) the observed t-statistic under the null hypothesis, can be estimated as the sum of `pr_below` and `pr_above`.

```
# p_value
p_value <- pr_below + pr_above
print(p_value)
```

```
## [1] 0.001595529
```

4.2.3 Interpretation

The above exercise is called **t-test** – perhaps, the most well-known hypothesis testing. To explain the interpretation of the results, let me use the following notations. t_{obs}, the observed t-statistic; t_0, possible t-statistics under the null hypothesis; $\Pr(\cdot)$, the probability of "$\cdot$" occurs – for example, $\Pr(x > 2)$ means that the probability of x exceeding 2.

We first calculated the observed t-statistic t_{obs} using the data of fish body size, which was -3.25. Then, we calculated p-value, i.e., $\Pr(t_0 < t_{obs}) + \Pr(t_0 > t_{obs})$ under the null hypothesis of $\mu_a = \mu_b$. We found p-value was very low, meaning that the observed t-statistic is very unlikely to occur if $\mu_a = \mu_b$ is the truth. Therefore, it is logical to conclude that the **Alternative Hypothesis** $\mu_a \neq \mu_b$ is very likely.

The same logic is used in many statistical analyses – define the null hypothesis, and estimate the probability of observing a given value under the null hypothesis. It is tricky, but I find this genius: we can substantiate the observation(s) objectively that are otherwise a subjective claim!

However, it is crucial to recognize that, **even if we found the observed t-statistic that results in very high p-value (i.e., very common under the null hypothesis), finding such a result does NOT support the null hypothesis – we just can't reject it.**

4.3 t-test in R

4.3.1 t-test with equal variance

In R, this t-test can be performed with `t.test()`. Let me examine if the function gives us identical results:

```
x <- df_fl %>%
  filter(lake == "a") %>%  # subset lake a
  pull(length)

y <- df_fl %>%
  filter(lake == "b") %>% # subset lake b
  pull(length)

t.test(x, y, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  x and y
## t = -3.2473, df = 98, p-value = 0.001596
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.3124399 -0.7995601
## sample estimates:
## mean of x mean of y
##    13.350    15.406
```

The reported t-statistic `t = -3.2473`, the p-value `p-value = 0.001596`, and the degrees of freedom all agreed (`df = 98`) with what we have estimated manually. Importantly, this t-test assumes relative similarity of variance between groups.

4.3.2 t-test with unequal variance

The relative similarity of variance between groups could be an unrealistic assumption. Luckily, there is a variant of t-test "Welch's t-test," in which we assume unequal variance between groups. The implementation is easy – set `var.equal = FALSE` in `t.test()`:

```
t.test(x, y, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -3.2473, df = 95.846, p-value = 0.001606
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.3127929 -0.7992071
## sample estimates:
## mean of x mean of y
##    13.350    15.406
```

In Welch's t-test, t-statistics are defined differently to account for unequal variance between groups[2]:

$$t = \frac{\hat{\mu_a} - \hat{\mu_b}}{\sqrt{\left(\frac{\hat{\sigma}_a^2}{N_a} + \frac{\hat{\sigma}_b^2}{N_b}\right)}}$$

[2]When $\hat{\sigma}_a^2 \approx \hat{\sigma}_b^2$, the t-statistic for Welch's t-test is reduced to the original t-statistic.

This t-statistic is known to follow the Student's t-distribution with the degrees of freedom:

$$d.f. = \frac{\frac{\hat{\sigma}_a^2}{N_a} + \frac{\hat{\sigma}_b^2}{N_b}}{\frac{(\hat{\sigma}_a^2/N_a)^2}{N_a - 1} + \frac{(\hat{\sigma}_b^2/N_b)^2}{N_b - 1}}$$

Therefore, the reported values of t-statistic and d.f. are different from what we estimated. The Welch's t-test covers the cases for equal variances; therefore, by default, we use the Welch's test.

4.4 Laboratory

Chapter 5

Multiple-Group Comparison

The t-test is used to compare two groups, but when there are more than two groups, ANOVA (Analysis of Variance) is employed. ANOVA allows for simultaneous comparison of means among multiple groups to determine if there are statistically significant differences. ANOVA uses the following steps:

1. Partition the total variability into **between-group** and **within-group** components.
2. Define a **test statistic** as a ratio of between-group variability to within-group variability (F statistic).
3. Define a probability distribution of F-statistic under the null hypothesis.
4. Estimate the probability of yielding greater the observed test statistic.

If appropriate, post-hoc tests can be conducted to identify specific differing groups.

5.1 Partition the Variability

The first step is to determine what aspect to examine. In the case of a t-test, we focus on the difference in sample means between groups. One might initially consider conducting t-tests for all possible combinations. However, this approach leads to a problem known as the multiple comparisons problem. Hence, this is not a viable option. Therefore, we need to explore the data from a different perspective.

To facilitate learning, let's once again utilize the lake fish data, but this time we have three lakes in our analysis:

```
df_anova <- read_csv("data_raw/data_fish_length_anova.csv")
distinct(df_anova, lake)
```

```
## # A tibble: 3 x 1
##   lake
##   <chr>
## 1 a
## 2 b
## 3 c
```

Visualization can be helpful in understanding how the data is distributed. While mean ± SD is perfectly fine here, let me use a violin plot to show a different way of visualization.

```
# geom_violin() - function for violin plots
# geom_jitter() - jittered points

df_anova %>%
  ggplot(aes(x = lake,
             y = length)) +
  geom_violin(draw_quantiles = 0.5, # draw median horizontal line
              alpha = 0.2) + # transparency
  geom_jitter(alpha = 0.2) # transparency
```

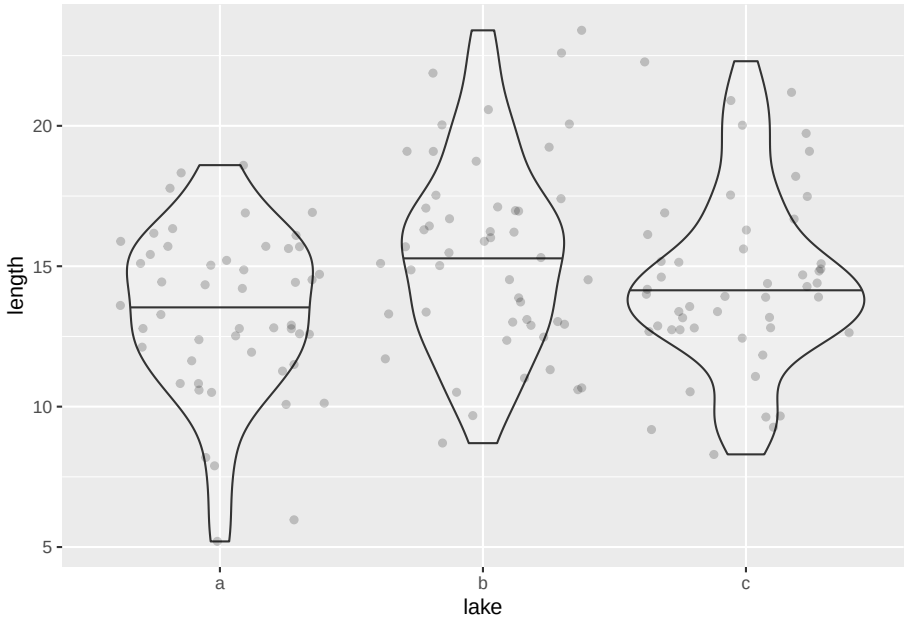


Figure 5.1: Violin plot for fish length in three lakes.

It appears that Lake `b` exhibits a larger average body size compared to the other lakes. One approach to quantifying this difference between groups is by examining the ratio of between-group variability to within-group variability. **If we observe a greater between-group variability relative to within-group variability, it suggests that the differences among the groups are substantial.** In other words, much of the observed variation is explained by the group structure (lake).

Let me denote between-group and within-group variability as S_w and S_a, respectively, which are defined as follows:

$$S_a = \sum_g \sum_i (\hat{\mu}_{g(i)} - \hat{\mu})^2$$
$$S_w = \sum_g \sum_i (x_i - \hat{\mu}_{g(i)})^2$$

The double summation $\sum_g \sum_i$ might scare you, but no worries. We can decompose this equation into two steps.

5.1.1 Between-group variability

Let me first consider $S_a = \sum_g \sum_i (\hat{\mu}_{g(i)} - \hat{\mu})^2$. In this equation, $\hat{\mu}$ is the overall mean of the fish length and $\hat{\mu}_{g(i)}$ is the group-mean in a given lake ($g = \{a, b, c\}$). Let's perform this estimation in R:

```
# estimate overall mean
mu <- mean(df_anova$length)

# estimate group means and sample size each
df_g <- df_anova %>%
  group_by(lake) %>%
  summarize(mu_g = mean(length), # mean for each group
            dev_g = (mu_g - mu)^2, # squared deviation for each group
            n = n()) # sample size for each group

print(df_g)
```

```
## # A tibble: 3 x 4
##   lake   mu_g   dev_g     n
##   <chr> <dbl>   <dbl> <int>
## 1 a      13.4 1.12       50
## 2 b      15.4 0.997      50
## 3 c      14.5 0.00344    50
```

In the column `dev_g`, we estimated $(\hat{\mu}_{g(i)} - \hat{\mu})^2$. We must sum over i (fish individual) to get the variability in each lake ($\sum_i (\hat{\mu}_{g(i)} - \hat{\mu})^2$). Since $\hat{\mu}_{g(i)}$ is

constant in each lake, we can simply multiply `dev_g` with sample size `n` in each lake:

```
df_g <- df_g %>%
  mutate(ss = dev_g * n)

print(df_g)
```

```
## # A tibble: 3 x 5
##   lake   mu_g   dev_g     n     ss
##   <chr> <dbl>   <dbl> <int>  <dbl>
## 1 a      13.4 1.12       50 55.9
## 2 b      15.4 0.997      50 49.9
## 3 c      14.5 0.00344    50  0.172
```

Sum over g (lake) to get S_a.

```
s_a <- sum(df_g$ss)
print(s_a)
```

```
## [1] 105.9365
```

5.1.2 Within-group variability

We can follow the same steps to estimate the within-group variability $S_w = \sum_g \sum_i (x_i - \hat{\mu}_{g(i)})^2$. Let's estimate $(x_i - \hat{\mu}_{g(i)})^2$ first:

```
df_i <- df_anova %>%
  group_by(lake) %>%
  mutate(mu_g = mean(length)) %>% # use mutate() to retain individual rows
  ungroup() %>%
  mutate(dev_i = (length - mu_g)^2) # deviation from group mean for each fish
```

You can take a look at group-level data with the following code; the column `mu_g` contains group-specific means of fish length, and `dev_i` contains $(x_{g(i)} - \hat{\mu}_{g(i)})^2$:

```
# filter() & slice(): show first 3 rows each group
print(df_i %>% filter(lake == "a") %>% slice(1:3))

print(df_i %>% filter(lake == "b") %>% slice(1:3))

print(df_i %>% filter(lake == "c") %>% slice(1:3))
```

Sum over i in each lake g ($\sum_i (x_{g(i)} - \hat{\mu}_{g(i)})^2$):

```
df_i_g <- df_i %>%
  group_by(lake) %>%
  summarize(ss = sum(dev_i))

print(df_i_g)
```

```
## # A tibble: 3 x 2
##   lake     ss
##   <chr> <dbl>
## 1 a      417.
## 2 b      565.
## 3 c      486.
```

Then sum over g to get S_w:

```
s_w <- sum(df_i_g$ss)
print(s_w)
```

```
## [1] 1467.645
```

5.1.3 Variability to Variance

I referred to S_a and S_w as "variability," which essentially represents the summation of squared deviations. To convert them into variances, we can divide them by appropriate numbers. In Chapter 2, I mentioned that the denominator for variance is the sample size minus one. The same principle applies here, but with caution.

For the between-group variability, denoted as S_a, the realized sample size is the number of groups, N_g, which in this case is three (representing the number of lakes). Therefore, we divide by three minus one to obtain an unbiased estimate of the between-group variance, denoted as $\hat{\sigma}_a^2$":

$$\hat{\sigma}_a^2 = \frac{S_a}{N_g - 1}$$

```
# n_distinct() count the number of unique elements
n_g <- n_distinct(df_anova$lake)
s2_a <- s_a / (n_g - 1)
print(s2_a)
```

```
## [1] 52.96827
```

Meanwhile, we need to be careful when estimating the within-group variance. Since the within-group variance is measured at the individual level, the number of data used is equal to the number of fish individuals. Yet, we subtract the number of groups – while the rationale behind this is beyond the scope, we are essentially accounting for the fact that some of the degrees of freedom are "used up" in estimating the group means. As such, we estimate the within-group variance $\hat{\sigma}_w^2$ as follows:

$$\hat{\sigma}_w^2 = \frac{S_w}{N - N_g}$$

```
s2_w <- s_w / (nrow(df_anova) - n_g)
print(s2_w)
```

```
## [1] 9.983982
```

5.2 Test Statistic

5.2.1 F-statistic

In ANOVA, we use F-statistic – the ratio of between-group variability to within-group variability. The above exercise was essentially performed to yield this test statistic:

$$F = \frac{\text{between-group variance}}{\text{within-group variance}} = \frac{\hat{\sigma}_a^2}{\hat{\sigma}_w^2}$$

```
f_value <- s2_a / s2_w
print(f_value)
```

```
## [1] 5.305325
```

The F-statistic in our data was calculated to be 5.31. This indicates that the between-group variance is approximately five times higher than the within-group variance. While this difference appears significant, it is important to determine whether it is statistically substantial. To make such a claim, we can use the Null Hypothesis as a reference.

5.2.2 Null Hypothesis

The F-statistic follows an F-distribution when there is no difference in means among the groups. Therefore, the null hypothesis we are considering in our example is that the means of all groups are equal, represented as $\mu_a = \mu_b = \mu_c$. It's worth noting that the alternative hypotheses can take different forms, such as $\mu_a \neq \mu_b = \mu_c$, $\mu_a = \mu_b \neq \mu_c$, or $\mu_a \neq \mu_b \neq \mu_c$. ANOVA, however, is unable to distinguish between these alternative hypotheses.

The degrees of freedom in an F-distribution are determined by two parameters: $N_g - 1$ and $N - N_g$. To visualize the distribution, you can utilize the`df()` function. Similar to the t-test, we can plot the F-distribution and draw a vertical line to represent the observed F-statistic. This approach allows us to assess the position of the observed F-statistic within the distribution and determine the associated p-value.

```
x <- seq(0, 10, by = 0.1)
y <- df(x = x, df1 = n_g - 1, df2 = nrow(df_anova) - n_g)

tibble(x = x, y = y) %>%
  ggplot(aes(x = x,
             y = y)) +
  geom_line() + # F distribution
  geom_vline(xintercept = f_value,
             color = "salmon") # observed F-statistic
```

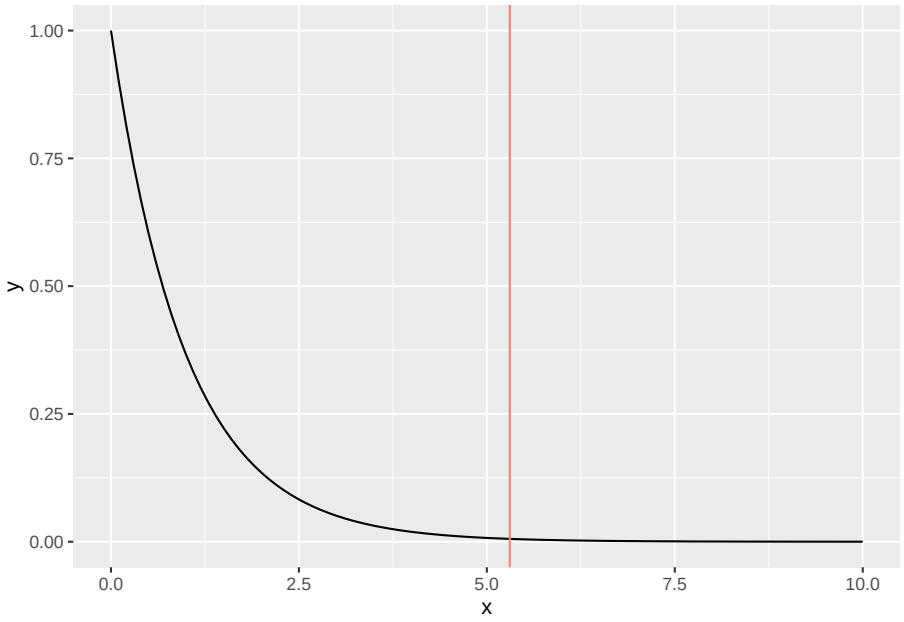


Figure 5.2: F-distribution. The vertical red line denotes the observed F-statistic.

Unlike t-statistics, F-statistics can take only positive values (because F-statistics are the ratio of positive values). The p-value here is $\Pr(F_0 > F)$, where F_0 is the possible F-statistics under the null hypothesis. Let's estimate this probability using `pf()`:

```
# pf() estimate the probability of less than q
# Pr(F0 > F) is 1 - Pr(F0 < F)
p_value <- 1 - pf(q = f_value, df1 = n_g - 1, df2 = nrow(df_anova) - n_g)
print(p_value)
```

```
## [1] 0.00596054
```

5.3 ANOVA in R

Like the t-test, R provides functions to perform ANOVA easily. In this case, we will utilize the `aov()` function. The first argument of this function is the "formula," which is used to describe the structure of the model. In our scenario, we aim to explain the fish body length by grouping them according to lakes. In the formula expression, we represent this relationship as `length ~ lake`, using the tilde symbol (~) to indicate the stochastic nature of the relationship.

```
# first argument is formula
# second argument is data frame for reference
# do not forget specify data = XXX! aov() refer to columns in the data frame
m <- aov(formula = length ~ lake,
         data = df_anova)

print(m)
```

```
## Call:
##    aov(formula = length ~ lake, data = df_anova)
##
## Terms:
##                       lake Residuals
## Sum of Squares   105.9365 1467.6454
## Deg. of Freedom         2       147
##
## Residual standard error: 3.159744
## Estimated effects may be unbalanced
```

The function returns `Sum of Squares` and `Deg. of Freedom` – these value matches what we have calculated. To get deeper insights, wrap the object with `summary()`:

```
summary(m)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## lake          2  105.9   52.97   5.305 0.00596 **
## Residuals   147 1467.6    9.98
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F-statistic and p-value are identical to our manual estimate.

5.4 Post-hoc Tests

When conducting an ANOVA and finding a significant difference among group means, a post-hoc test is often performed to determine which specific groups differ significantly from each other. Post-hoc tests help to identify pairwise comparisons that contribute to the observed overall difference.

There are several post-hoc tests available for ANOVA, including:

1. Tukey's Honestly Significant Difference (HSD): This test compares all possible pairs of group means and provides adjusted p-values to control for the family-wise error rate. It is a commonly used and reliable post-hoc test.
2. Bonferroni correction: This method adjusts the significance level for each individual comparison to control the overall Type I error rate. The adjusted p-value is obtained by dividing the desired significance level (e.g., 0.05) by the number of comparisons.
3. Scheffe's test: This test controls the family-wise error rate by considering all possible pairwise comparisons. It is more conservative than Tukey's HSD, but it is suitable for cases where specific comparisons are of particular interest.
4. Dunnett's test: This test is useful when comparing multiple groups against a control group. It controls the overall error rate by conducting multiple t-tests between the control group and each of the other groups.

The choice of post-hoc test depends on the specific research question, assumptions, and desired control of the Type I error rate. It is important to select an appropriate test and interpret the results accordingly to draw valid conclusions about group differences in an ANOVA analysis.

5.5 Laboratory

Chapter 6

Regression

Our previous examples focused on cases that involved a distinct group structure. However, it is not always the case that such a structure exists. Instead, we might examine the relationship between continuous variables. In this chapter, I will introduce a technique called linear regression.

6.1 Explore Data Structure

For this exercise, we will use algae biomass data from streams. Download the data here and locate it in `data_raw/`.

```
library(tidyverse)
df_algae <- read_csv("data_raw/data_algae.csv")
print(df_algae)
```

```
## # A tibble: 50 x 4
##    biomass unit_biomass conductivity unit_cond
##      <dbl> <chr>               <dbl> <chr>
##  1   19.8  mg_per_m2            30.2 ms
##  2   24.4  mg_per_m2            40.4 ms
##  3   27.4  mg_per_m2            59.4 ms
##  4   48.2  mg_per_m2            91.3 ms
##  5   19.2  mg_per_m2            24.2 ms
##  6   57.0  mg_per_m2            90.4 ms
##  7   51.9  mg_per_m2            94.7 ms
##  8   40.8  mg_per_m2            67.8 ms
##  9   37.1  mg_per_m2            64.8 ms
## 10    3.55 mg_per_m2            10.9 ms
## # i 40 more rows
```

This dataset comprises data collected from 50 sampling sites. The variable

`biomass` represents the standing biomass, indicating the dry mass of algae at the time of collection. On the other hand, `conductivity` serves as a proxy for water quality, with higher values usually indicating a higher nutrient content in the water. Let me now proceed to draw a scatter plot that will help visualize the relationship between these two variables.

```
df_algae %>%
  ggplot(aes(x = conductivity,
             y = biomass)) +
  geom_point()
```

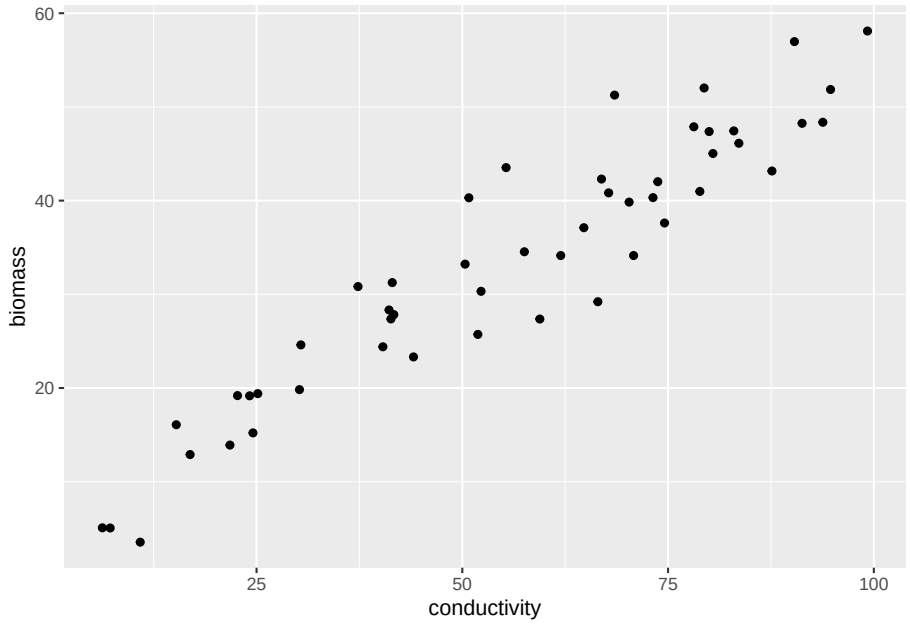


Figure 6.1: The relationship between algae biomass and conductivity.

It seems that there is a noticeable positive correlation where increased conductivity results in higher algal biomass. Nevertheless, how can we accurately represent this relationship with a line?

6.2 Drawing the "fittest" line

6.2.1 Linear formula

The first step is to represent the relationship as a formula; let me ASSUME that the relationship can be described by the following linear formula:

$$y_i = \alpha + \beta x_i$$

In this formula, the algal biomass, denoted as y_i, at site i, is expressed as a function of conductivity, represented by x_i. The variable being explained (y_i) is referred to as a **response (or dependent) variable**, while the variable used to predict the response variable (x_i) is referred to as **an explanatory (or independent) variable.** However, there are two additional constants, namely α and β. α is commonly known as the **intercept**, while β is referred to as the **slope** or **coefficient**. The intercept represents the value of y when x is equal to zero, while the slope indicates the change in y associated with a unit increase in x (refer to Figure 6.2).

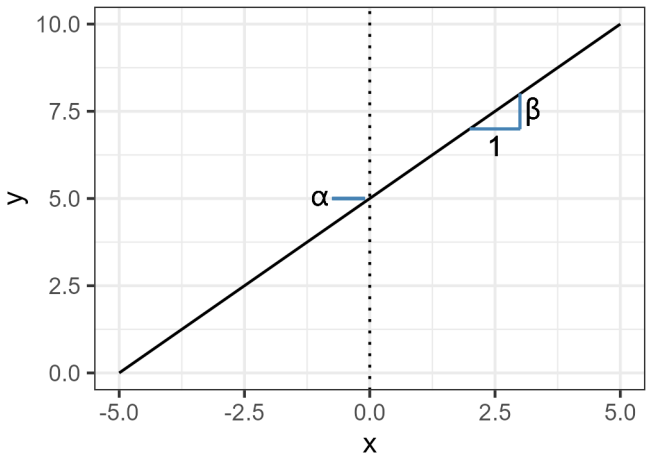


Figure 6.2: Intercept and slope.

However, this formula, or "model," is incomplete. We must add the error term ε_i (**residual**) to consider the uncertainty associated with the observation process.

$$y_i = \alpha + \beta x_i + \varepsilon_i \varepsilon_i \sim \text{Normal}(0, \sigma^2)$$

Notice that y_i, x_i, and ε_i have subscripts, while α and β do not. This means that y_i, x_i, and ε_i vary by data point i, and α and β are constants. Although $\alpha + \beta x_i$ cannot reproduce y_i perfectly, we "fill" the gaps with the error term ε_i, which is assumed to follow a Normal distribution.

In R, finding the best α and β – i.e., the parameters in this model – is very easy. Function `lm()` does everything for you. Let me apply this function to the example data set:

```
# lm() takes a formula as the first argument
# don't forget to supply your data
m <- lm(biomass ~ conductivity,
        data = df_algae)

summary(m)
```

```
##
## Call:
## lm(formula = biomass ~ conductivity, data = df_algae)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5578 -2.8729 -0.7307  2.5479 11.4700
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.29647    1.57178    3.37  0.00149 **
## conductivity  0.50355    0.02568   19.61  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.649 on 48 degrees of freedom
## Multiple R-squared:  0.889,  Adjusted R-squared:  0.8867
## F-statistic: 384.5 on 1 and 48 DF,  p-value: < 2.2e-16
```

In `Coefficients:`, it says `(Intercept)` is 5.3 and `conductivity` is 0.5. These values corresponds to α and β. Meanwhile, `Residual standard error:` indicates the SD of the error term σ. Thus, substituting these values into the formula yields:

$$y_i = 5.30 + 0.50x_i + \varepsilon_i\varepsilon_i \sim \text{Normal}(0, 4.6^2)$$

We can draw this "fittest" line on the figure:

```
# coef() extracts estimated coefficients
# e.g., coef(m)[1] is (Intercept)

alpha <- coef(m)[1]
beta <- coef(m)[2]

df_algae %>%
  ggplot(aes(x = conductivity,
             y = biomass)) +
  geom_point() +
```

```
  geom_abline(intercept = alpha,
              slope = beta) # draw the line
```

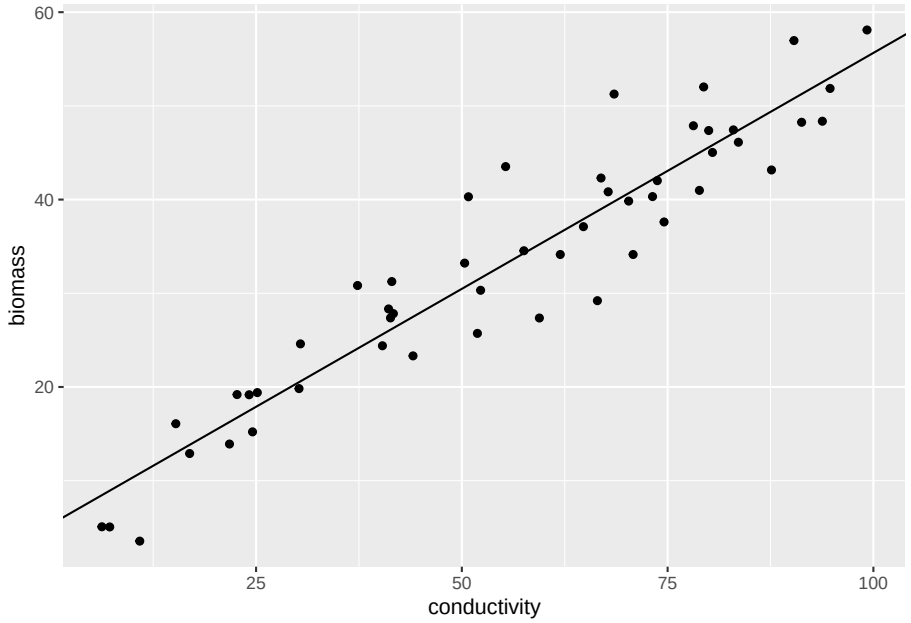


Figure 6.3: Drawing the fitted model prediction.

However, how did `lm()` find α and β?

6.2.2 Minimizing the errors

Notice that the error term is the difference between y_i and $\alpha + \beta x_i$:

$$\varepsilon_i = y_i - (\alpha + \beta x_i)$$

In other words, this term represents the portion of the observation y_i that cannot be explained by the conductivity x_i. Therefore, a logical approach would be to find values for α and β that minimize this unexplained portion across all data points i.

Least squares methods are a widely employed statistical approach to achieve this objective. These methods aim to minimize the sum of squared errors, denoted as $\sum_i \varepsilon_i^2$, across all data points. As the deviation from the expected value of $\alpha + \beta x_i$ increases, this quantity also increases. Consequently, determining parameter values (α and β) that minimize this sum of squared errors yields the best-fitting formula (see Section 6.2.3 for further details).

6.2.3 Least Squares

Introducing a matrix representation is indeed a helpful way to explain the least square method. Here, I represent the vector of observed values y_i as Y ($Y = \{y_1, y_2, ..., y_n\}^T$), the vector of parameters as Θ ($\Theta = \{\alpha, \beta\}^T$), and the matrix composed of 1s and x_i values as X. The matrix X can be written as:

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}$$

Using matrix notation, we can express the error vector $\boldsymbol{\varepsilon}$ as:

$$\boldsymbol{\varepsilon} = Y - X\Theta$$

The column of 1s is required to represent the intercept; $X\Theta$ reads:

$$X\Theta = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha + \beta x_1 \\ \alpha + \beta x_2 \\ \alpha + \beta x_3 \\ \vdots \\ \alpha + \beta x_n \end{pmatrix}$$

Minimizing the squared magnitude of $\vec{\varepsilon}$ (solve the partial derivative of $||\boldsymbol{\varepsilon}||^2$ about Θ) leads to the solution (for detailed derivation, refer to Wikipedia):

$$\hat{\Theta} = (X^T X)^{-1} X^T Y$$

Let me see if this reproduces the result of `lm()`:

```
# create matrix X
v_x <- df_algae %>% pull(conductivity)
X <- cbind(1, v_x)

# create a vector of y
Y <- df_algae %>% pull(biomass)

# %*%: matrix multiplication
# t(): transpose a matrix
# solve(): computes the inverse matrix
theta <- solve(t(X) %*% X) %*% t(X) %*% Y
print(theta)
```

```
##           [,1]
##     5.2964689
## v_x 0.5035548
```

`lm()` output for a reference:

```
m <- lm(biomass ~ conductivity,
        data = df_algae)

coef(m)
```

```
##  (Intercept) conductivity
##    5.2964689    0.5035548
```

6.2.4 Standard Errors and t-values

Ensuring the reliability of the estimated point estimates for α and β is crucial. In the `summary(m)` output, two statistical quantities, namely the `Std. Error` and `t-value`, play a significant role in assessing the uncertainty associated with these estimates.

The standard error (`Std. Error`) represents the estimated standard deviation of the parameter estimates ($\hat{\theta}$ is either $\hat{\alpha}$ or $\hat{\beta}$). A smaller value of the standard error indicates a higher degree of statistical reliability in the estimate. On the other hand, the `t-value` is a concept we have covered in Chapter 4. This value is defined as:

$$t = \frac{\hat{\theta} - \theta_0}{SE(\hat{\theta})}$$

The t-statistic is akin to the t-test; however, in the context of regression analysis, we do not have another group to compare with (i.e., θ_0). Typically, in regression analysis, we use zero as the reference ($\theta_0 = 0$). Therefore, higher t-values in `lm()` indicate a greater deviation from zero. Consequently, the **Null Hypothesis in regression analysis is** $\beta = 0$. This hypothesis is sensible in our specific example since we are interested in quantifying the effect of conductivity. If $\beta = 0$, it implies that conductivity has no effect on algal biomass. Since $\theta_0 = 0$, the following code reproduces the reported t-values:

```
# extract coefficients
theta <- coef(m)

# extract standard errors
se <- sqrt(diag(vcov(m)))

# t-value
t_value <- theta / se
print(t_value)
```

```
##  (Intercept) conductivity
##     3.369732    19.609446
```

After defining the Null Hypothesis and t-values, we can compute the probability of observing the estimated parameters under the Null Hypothesis. Similar to the t-test, `lm()` utilizes a Student's t-distribution for this purpose. However, the difference lies in how we estimate the degrees of freedom. In our case, we have 50 data points but need to subtract the number of parameters, which are α and β. Therefore, the degrees of freedom would be $50 - 2 = 48$. This value is employed when calculating the p-value:

```
# for intercept
# (1 - pt(t_value[1], df = 48)) calculates pr(t > t_value[1])
# pt(-t_value[1], df = 48) calculates pr(t < -t_value[1])
p_alpha <- (1 - pt(t_value[1], df = 48)) + pt(-t_value[1], df = 48)

# for slope
p_beta <- (1 - pt(t_value[2], df = 48)) + pt(-t_value[2], df = 48)

print(p_alpha)
```

```
## (Intercept)
## 0.001492023
```

```
print(p_beta)
```

```
## conductivity
## 7.377061e-25
```

6.3 Unexplained Variation

6.3.1 Retrieve Errors

The function `lm()` provides estimates for α and β, but it does not directly provide the values of ε_i. To gain a deeper understanding of the statistical model, it is necessary to examine the residuals ε_i. By using the `resid()` function, you can obtain the values of ε_i. This allows you to explore and extract insights from the developed statistical model.

```
# eps - stands for epsilon
eps <- resid(m)
head(eps)
```

```
##          1          2          3          4          5          6
## -0.6838937 -1.2149034 -7.8576927 -3.0109473  1.7076481  6.1773586
```

Each element retains the order of data point in the original data frame, so

`eps[1]` (ε_1) should be identical to $y_1 - (\alpha + \beta x_1)$. Let me confirm:

```
# pull vector data
v_x <- df_algae %>% pull(conductivity)
v_y <- df_algae %>% pull(biomass)

# theta[1] = alpha
# theta[2] = beta
error <- v_y - (theta[1] + theta[2] * v_x)

# cbind() combines vectors column-wise
# head() retrieves the first 6 rows
head(cbind(eps, error))
```

```
##          eps      error
## 1 -0.6838937 -0.6838937
## 2 -1.2149034 -1.2149034
## 3 -7.8576927 -7.8576927
## 4 -3.0109473 -3.0109473
## 5  1.7076481  1.7076481
## 6  6.1773586  6.1773586
```

To ensure that the two columns are indeed identical, a visual check may be prone to overlooking small differences. To perform a more precise comparison, you can use the `all()` function to check if the data aligns (here, evaluated at the five decimal points using `round()`). By comparing the two columns in this manner, you can confirm whether they are identical or not.

```
# round values at 5 decimal
eps <- round(eps, 5)
error <- round(error, 5)

# eps == error return "TRUE" or "FALSE"
# all() returns TRUE if all elements meet eps == error
all(eps == error)
```

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

6.3.2 Visualize Errors

To visualize the errors or residuals, which represent the distance from the best-fitted line to each data point, you can make use of the `geom_segment()` function:

```
# add error column
df_algae <- df_algae %>%
  mutate(eps = eps)

df_algae %>%
```

```
ggplot(aes(x = conductivity,
           y = biomass)) +
geom_point() +
geom_abline(intercept = alpha,
            slope = beta) +
geom_segment(aes(x = conductivity, # start-coord x
                 xend = conductivity, # end-coord x
                 y = biomass, # start-coord y
                 yend = biomass - eps), # end-coord y
             linetype = "dashed")
```

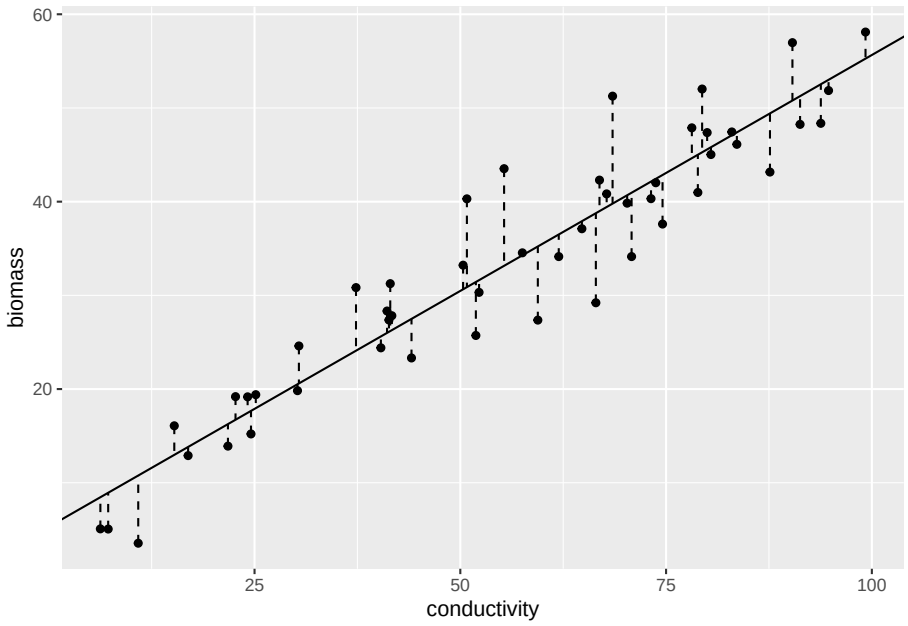


Figure 6.4: Vertical segments indicate errors.

6.3.3 Coefficient of Determination

One crucial motivation for developing a regression model, or any statistical model, is to assess the extent to which the explanatory variable (x_i) explains the variation in the response variable (y_i). By fitting a regression model, we can quantify the amount of variability in the response variable that can be attributed to the explanatory variable. This assessment provides valuable insights into the strength and significance of the relationship between the variables under consideration.

The **coefficient of determination**, denoted as R^2, is a statistical measure

that assesses the proportion of the variance in the response variable that can be explained by the explanatory variable(s) in a regression model. It provides an indication of how well the regression model fits the observed data. The formula for R^2 is:

$$R^2 = 1 - \frac{SS}{SS_0}$$

where SS is the summed squares of residuals ($\sum_i \varepsilon_i^2$), and SS_0 is the summed squares of the response variable ($SS_0 = \sum_i (y_i - \hat{\mu}_y)^2$). The term $\frac{SS}{SS_0}$ represents the proportion of variability "unexplained" – therefore, $1 - \frac{SS}{SS_0}$ gives the proportion of variability "explained." R^2 is a value between 0 and 1. An R^2 value of 0 indicates that the explanatory variable(s) cannot explain any of the variability in the response variable, while an R^2 value of 1 indicates that the explanatory variable(s) can fully explain the variability in the response variable.

While R^2 is provided as a default output of `lm()`, let's confirm if the above equation reproduces the reported value:

```
# residual variance
ss <- sum(resid(m)^2)

# null variance
ss_0 <- sum((v_y - mean(v_y))^2)

# coefficient of determination
r2 <- 1 - ss / ss_0

print(r2)
```

```
## [1] 0.8890251
```

Compare it with `lm()` output:

```
summary(m)
```

```
##
## Call:
## lm(formula = biomass ~ conductivity, data = df_algae)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5578 -2.8729 -0.7307  2.5479 11.4700
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.29647    1.57178    3.37  0.00149 **
```

```
## conductivity  0.50355    0.02568   19.61  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.649 on 48 degrees of freedom
## Multiple R-squared:  0.889,  Adjusted R-squared:  0.8867
## F-statistic: 384.5 on 1 and 48 DF,  p-value: < 2.2e-16
```

`Multiple R-squared:` corresponds to the calculated coefficient of determination.

6.4 Laboratory

Chapter 7

Appendix: Project Management

7.1 R Project

Throughout this book, I will use the 'R Project' as the core workspace unit. It serves as a centralized location where all relevant materials, such as R scripts (`.R`) and data files, are consolidated. There are various ways to organize your project, but I prefer creating a single 'R Project' for a set of scripts and data that contribute to a specific publication (see example here). To set up an 'R Project,' you'll need to have both *RStudio* and the base *R* software installed. While *R* can be used independently, I highly recommend using it in conjunction with *RStudio* due to the latter's numerous features that facilitate data analysis. You can download *R* and *RStudio* from the following websites:

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

Upon launching *RStudio*, you will encounter the interface depicted in Figure 7.1. Initially, the interface comprises three primary panels: the `Console`, `Environment`, and `Files`. The `Console` is where you write code and execute calculations, data manipulation, and analysis. The `Environment` panel lists the saved objects, while the `Files` panel displays files in a designated location on your computer.

After pasting the script into the console, you will notice that the variable `x` appears in the environment panel.

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

`x` is an object that stores information. In this particular case, we have stored a sequence of `1` and `2` in the object `x`. Once you have stored information in `x`,

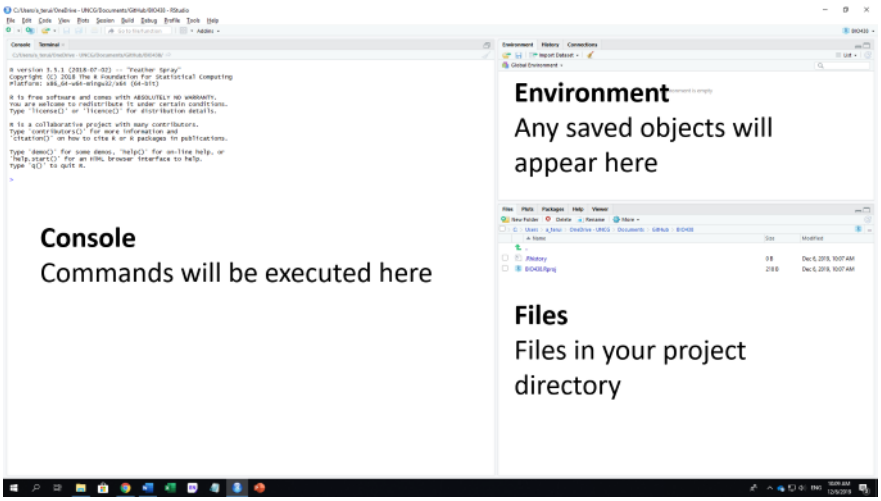


Figure 7.1: RStudio interface.

you can access it by simply typing `x`.

```
x
```

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

Excellent! While it is possible to work with your data using this approach, it is important to highlight that **it is generally regarded as a poor practice**. As your project evolves, you will accumulate a substantial amount of material, such as writing more than 2000 lines of code for a single project. Consequently, it becomes essential to implement effective code management strategies. How do you presently handle code management?

7.1.1 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].

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

7.1.2 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`.

7.1.3 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. 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 7.1, I present my recommended subdirectory structure.

Table 7.1: 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.

7.2 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.

Chapter 8

Appendix: Git & GitHub

8.1 Git & GitHub

In this section, I will explain how to integrate Git and GitHub into R Studio. While R Studio is already an excellent tool, it becomes even more powerful when combined with Git and GitHub. Git is a free and open-source distributed version control system that tracks changes in your code as you work on your project, ensuring you are aware of any modifications made to your script (and other) files. Tracking changes is crucial to avoid unintended errors in your code and helps prevent the creation of redundant files. Although Git is primarily a local system, it has an online counterpart called GitHub.

To set up this system, you'll need to follow a few steps. The first step is to install Git on your computer:

- For **Windows**: Install Git from here. During the installation process, you'll be prompted about "Adjusting your PATH environment." Choose "Git from the command line and also from 3rd-party software" if it's not already selected.
- For **Mac**: Follow the instructions provided here.

Once Git is installed, open R Studio and navigate to `Create Project` > `New Directory` > `New Project`. If you see the checkbox labeled "Create a git repository," make sure to select it before creating your new project (refer to Figure 8.1). This action will display the Git pane on the upper right panel of R Studio, indicating that Git integration is enabled.

If you are unable to find the options mentioned above, follow these steps:

1. Click on `Tools` in the menu bar of R Studio.
2. Select `Terminal` and then choose `New Terminal`.

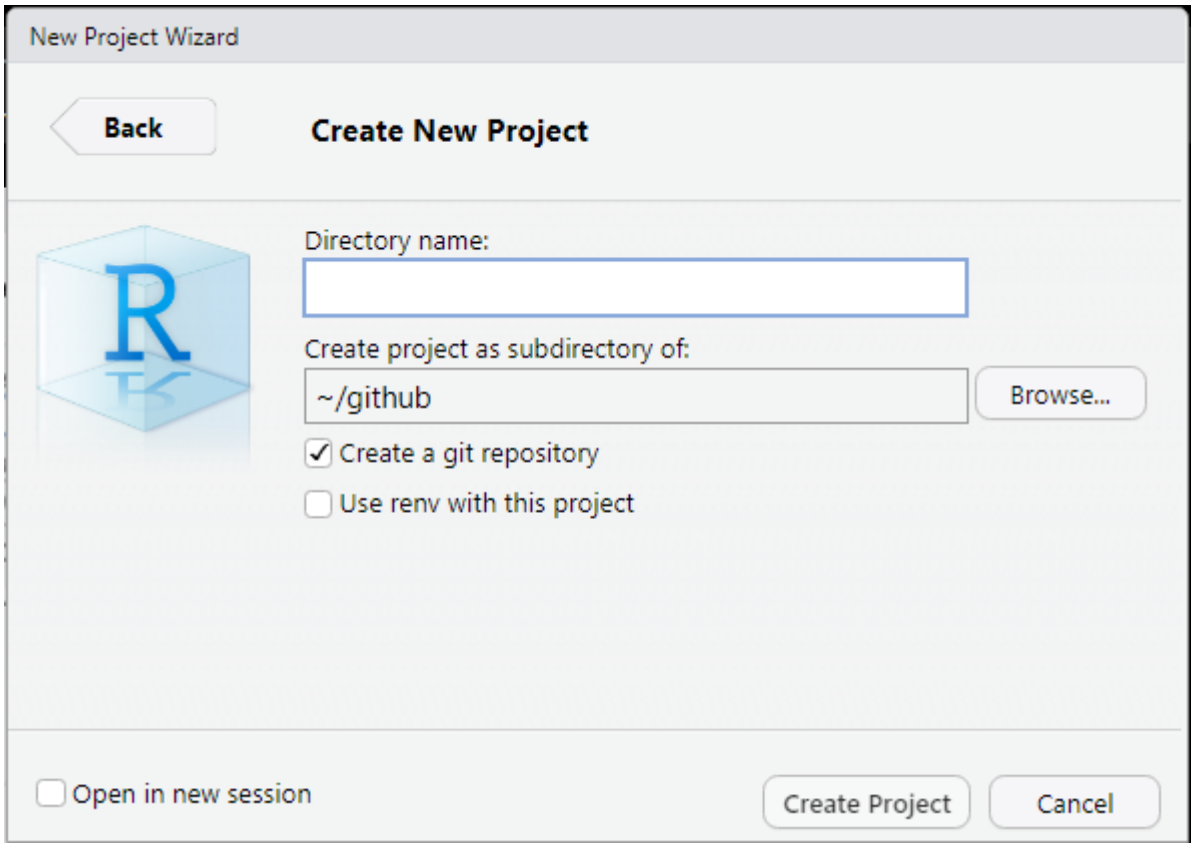


Figure 8.1: After installing Git, you should see 'Create a git repository'.

3. In the terminal window, type `where git` and press Enter. This command will display the location of the Git executable on your computer.
4. Next, go back to `Tools` in the menu bar and select `Global Options`.
5. In the options window, navigate to `Git/SVN`.
6. Look for the field labeled *Git executable* and specify the location of the Git executable that you obtained from the terminal.

After setting up Git, you can proceed to create an account on GitHub. It's a free platform, and when choosing a username, consider using lowercase letters and including your name to make it easier for others to find you.

While R Studio seamlessly integrates with Git and GitHub, using a Git client can provide additional visual aids. There are various options for Git clients (see choices here), but for this exercise, we will use GitHub Desktop. Install GitHub Desktop on your computer from here.

8.2 Commit & Push

8.2.1 Register Your Git repo

To open the `R Project` you've just created as a git repository, follow these steps:

1. Open R Studio.
2. Click on `File` in the menu bar.
3. Select `Open Project`.
4. Browse to the directory where you created the `R Project` and select the corresponding `.Rproj` file.
5. R Studio will open the project, and you will see the project name in the top-right corner of the window.

To create a sample `.R` file named `sample.R` within the project, you can use the shortcut `Ctrl + Shift + N` or follow these steps:

1. Click on `File` in the menu bar.
2. Select `New File`.
3. Choose `R Script`.
4. A new script editor will open, where you can write your R code.
5. Write your code in the editor and save it as `sample.R` by clicking on `File` and selecting `Save` or using the shortcut `Ctrl + S`.

Remember to save the file after writing your code.

```
## produce 100 random numbers that follows a normal distribution
x <- rnorm(100, mean = 0, sd = 1)

## estimate mean
mean(x)
```

```
## estimate SD
sd(x)
```

To open the GitHub Desktop app, locate the application on your computer and launch it. Once opened, you will see a GUI similar to the one depicted in Figure 8.2.

Click on the "Current Repository" button located at the top left corner of the GitHub Desktop app interface. Then, select "Add" followed by "Add existing repository" from the dropdown menu, as shown in Figure 8.3.

8.2.2 Commit

GitHub Desktop will prompt you to enter a local path to your Git repository. Browse and select the directory where your `R Project` is located. Once you have selected the directory, the local Git repository will appear in the list of repositories on the left side bar of GitHub Desktop, as shown in Figure 8.3.

Now that your repository is added to GitHub Desktop, you are ready to start committing your files to Git. Committing is the process of recording the changes made to your files in Git. To proceed with committing, click on your Git repository in GitHub Desktop. You will see the following options and information displayed:

- The list of changed files: This section shows the files that have been modified since the last commit.
- A summary of the changes: This section provides an overview of the changes made to the selected files.
- Commit message: This is where you can enter a descriptive message explaining the changes made in the commit.

By providing a meaningful commit message, you can keep track of the changes made to your files and easily understand the purpose of each commit.

At this stage, your file (`.R` or any other file) is saved in your local directory but has not been recorded in Git as a change history. To commit the changes, follow these steps:

1. First, select the file(s) you want to commit. You can do this by checking the checkboxes next to the files listed in the "Changes" section of the GitHub Desktop app.

2. Once you have selected the files, go to the bottom left of the GitHub Desktop app window. You will see a small box that says `summary (required)` or `Create sample.R`.

3. In this box, enter a descriptive title that explains what you did in this commit. It is important to provide a meaningful summary, as you cannot proceed with the commit unless you enter this information. For example, for this exercise, you can enter `Initial commit` as the commit title. For

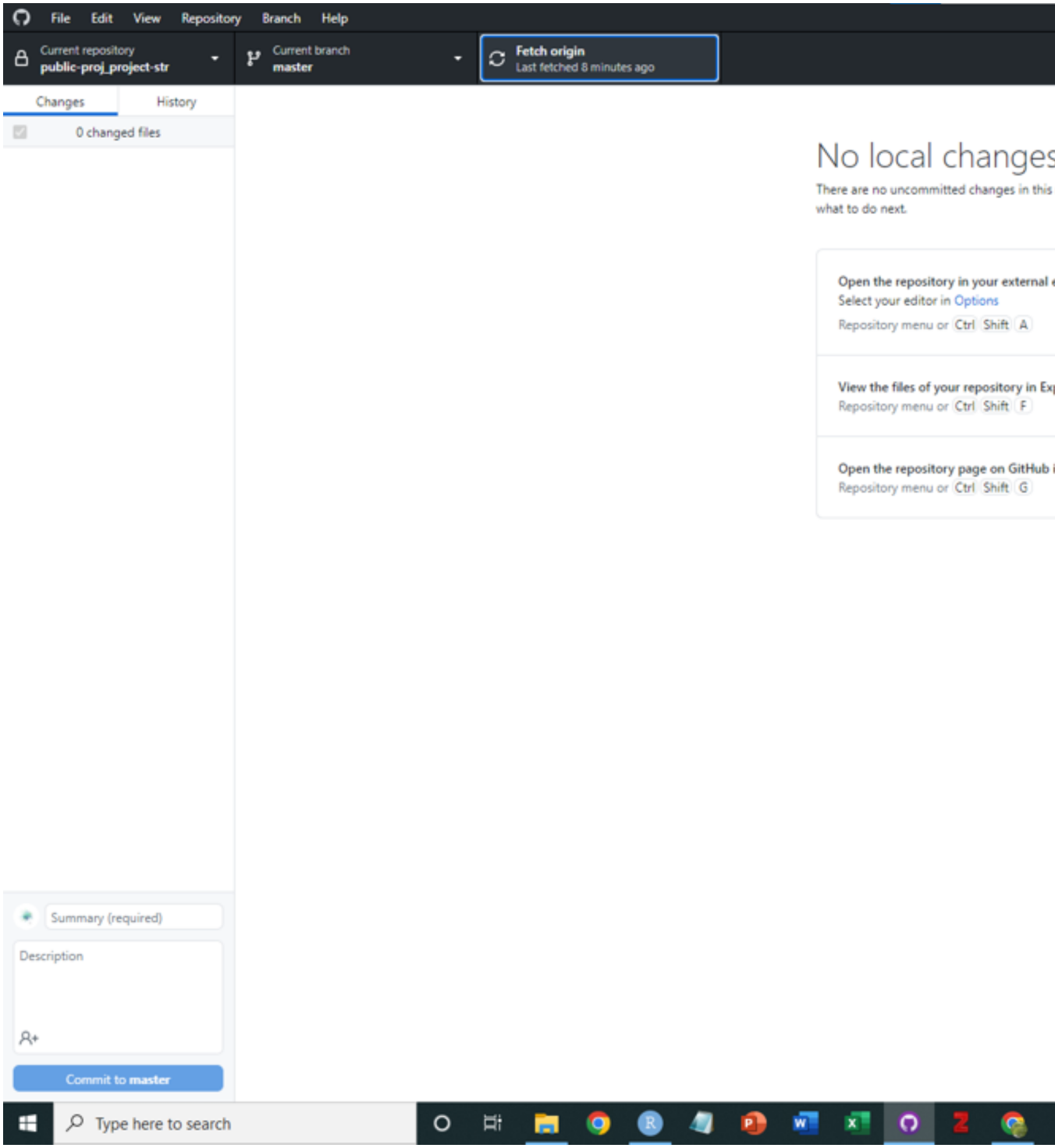


Figure 8.2: GUI for GitHub Desktop

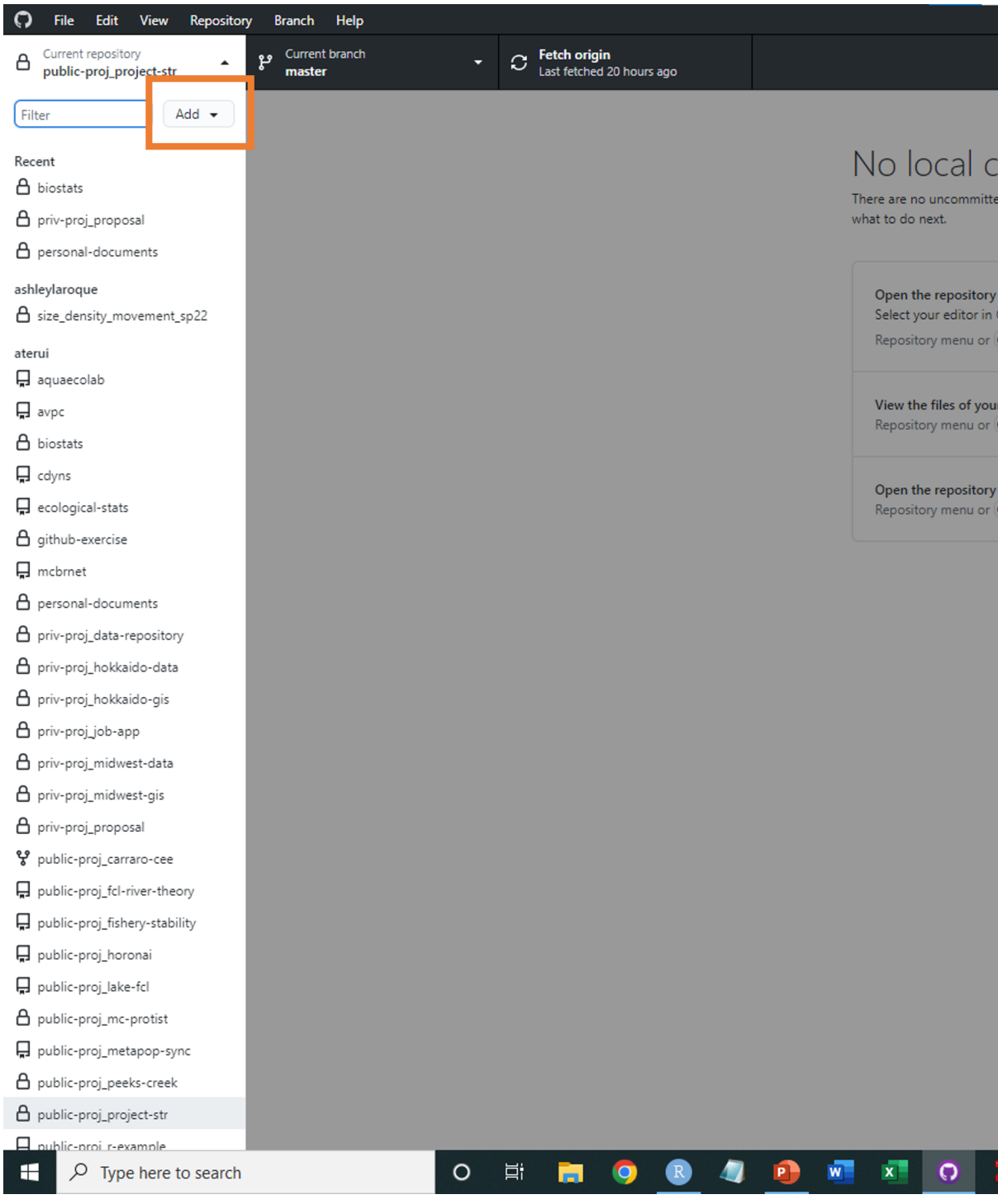


Figure 8.3: Add dropdown on the top left

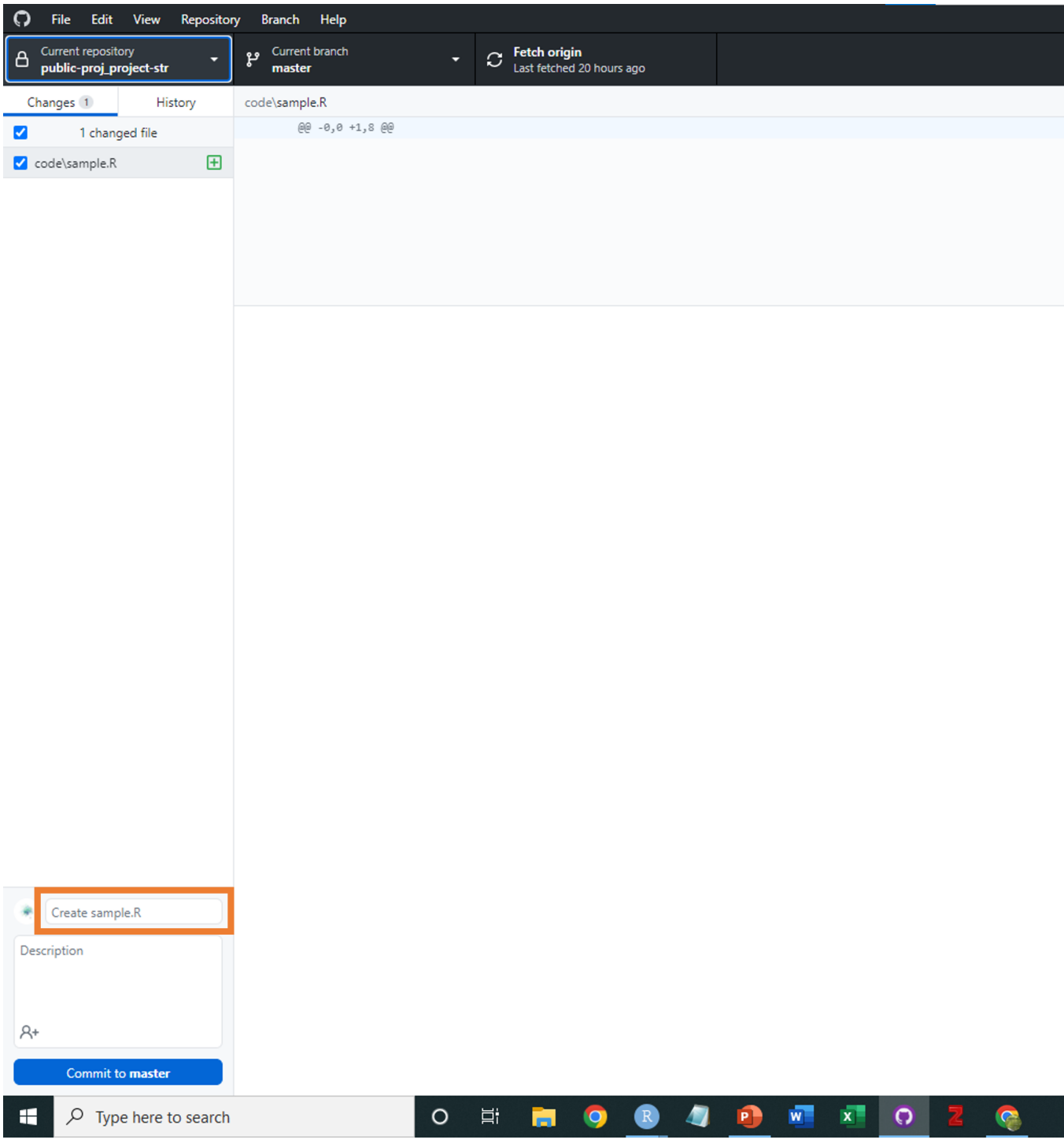


Figure 8.4: You will see ‘Create sample.R‘ or ‘summary (required)‘ on the bottom left

subsequent commits, it is recommended to provide more informative commit messages that accurately describe the changes made. You can refer to online resources for recommendations on how to write effective commit titles and descriptions.

4. After entering the commit title, click on the button that says `Commit to master`. This action will record the changes made to the selected files in Git.

Now, the changes to the selected files have been successfully recorded in Git as a commit. It's important to commit regularly and provide meaningful commit messages so that you can track changes and understand the evolution of your project when needed.

8.2.3 Push

It's important to note that your changes are currently recorded in your local computer's Git repository but have not been published to your online GitHub repository. To send the local changes to the online GitHub repository, you need to use the "Push" operation through GitHub Desktop.

After you make a commit in GitHub Desktop, you will be prompted with a dialog box asking if you want to push the commit to an online repository, as shown in Figure 8.5. If this is your first push, there won't be a corresponding repository on GitHub linked to your local repository. In this case, GitHub Desktop will ask if you want to publish it on GitHub. Please note that "publishing" here means making the repository available on GitHub, but it will remain private unless you explicitly choose to make it public.

If you are comfortable with the changes you made and want to send them to the online repository, click the "Push" button in GitHub Desktop. This action will push your commits to the remote repository on GitHub, effectively synchronizing your local repository with the online repository.

8.2.4 Edit

We have created a file named `sample.R` as per the previous instructions. Now, let's make a minor change to the content of the `sample.R` file. Open the `sample.R` file in your preferred text editor or in R Studio, and make a modification to the code. For example, you can update the code from:

```r
## produce 100 random numbers that follows a normal distribution
x <- rnorm(100, mean = 0, sd = 1)

## estimate mean
mean(x)
```

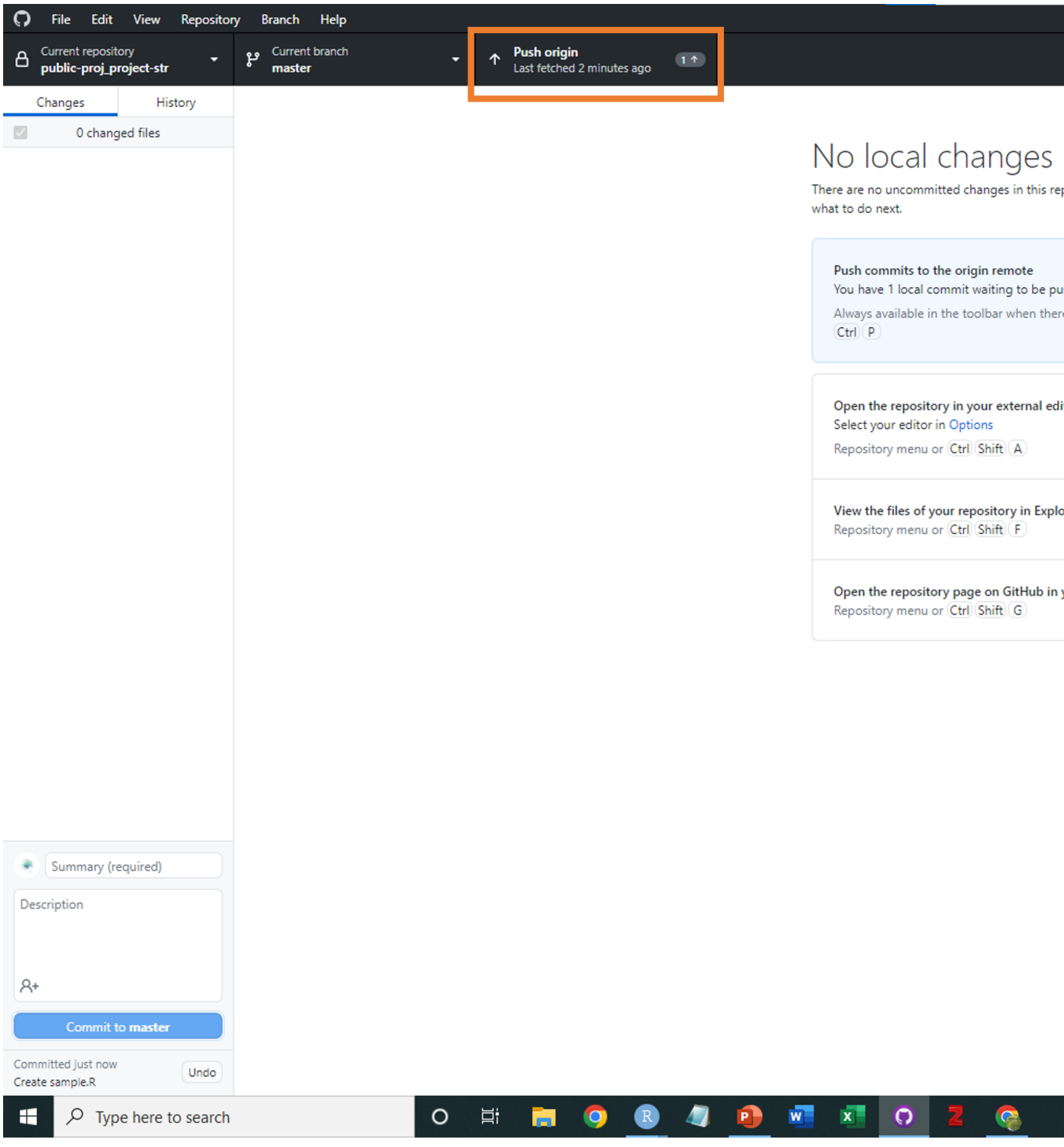


Figure 8.5: To Push your code, hit the highlighted menu button

```
## estimate SD
sd(x)
```

to:

```
## produce 100 random numbers that follows a normal distribution
x <- rnorm(100, mean = 0, sd = 1)

## estimate mean
median(x)

## estimate SD
var(x)
```

Save the changes to the `sample.R` file. Now, let's go back to GitHub Desktop and see how we can handle this change.

After making the change to the `sample.R` file, open GitHub Desktop again. You will notice that GitHub Desktop automatically detects the difference between the new and old versions of the file. It highlights the specific parts of the script that have been edited, which can be extremely helpful when reviewing and comparing changes in your code.

This feature provided by GitHub Desktop makes it easier to track modifications and understand the specific updates made to your files. It helps streamline the coding process by providing a clear visual representation of the differences between different versions of your code.

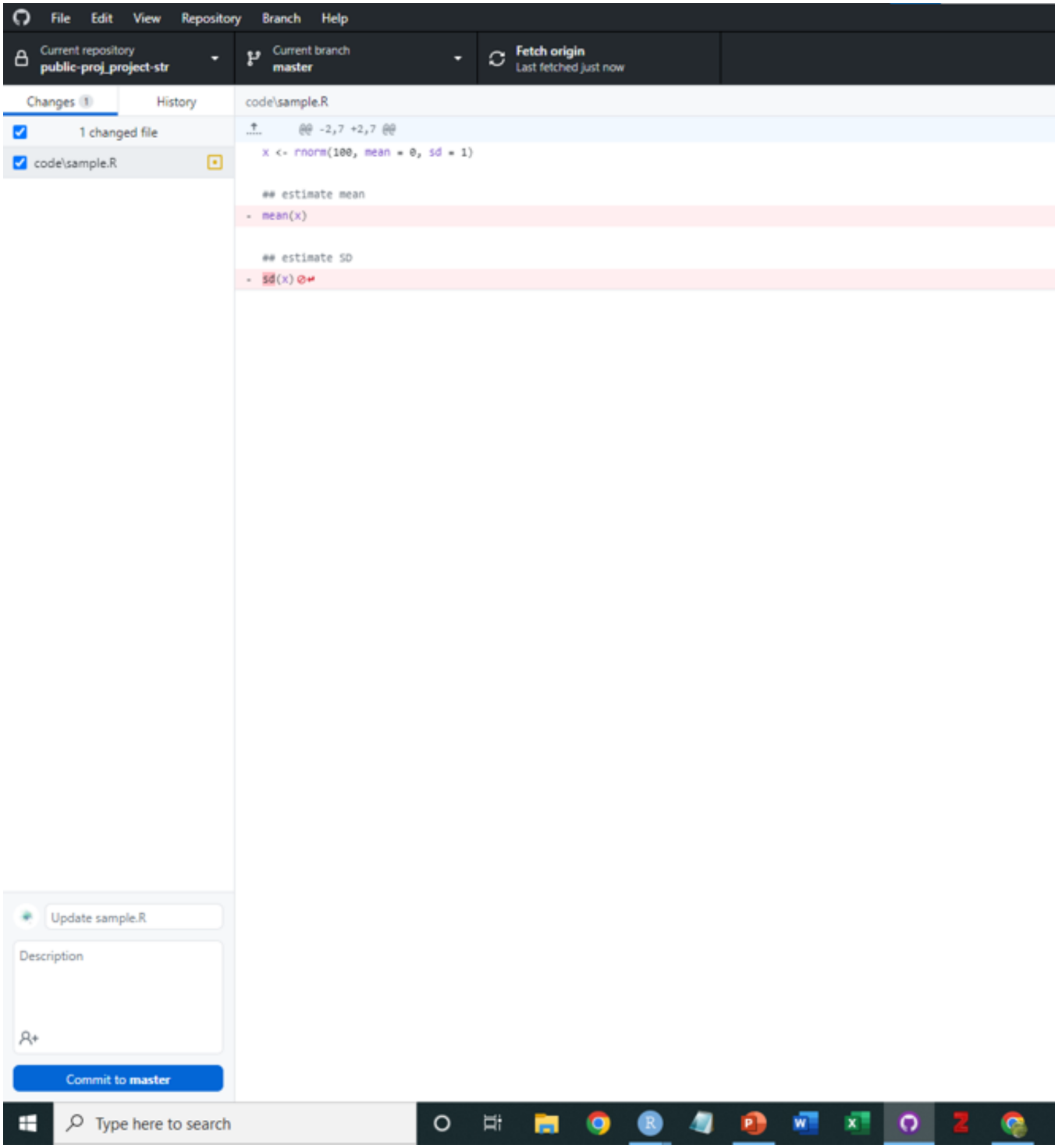


Figure 8.6: Git detects edits to your codes

Chapter 9

Appendix: Data Structure

9.1 Overview

R has 6 basic **data types**.

- character: `"aquatic"`, `"ecology"` (no order)
- factor: similar to character, but has *levels* (alphabetically ordered by default)
- numeric: `20.0` , `15.5`
- integer: `3`, `7`
- logical: `TRUE` , `FALSE`
- complex: `1+2i` (complex numbers with real and imaginary parts)

These elements form one of the following **data structures**.

- **vector**: a series of elements. A single data type is allowed in a single vector
- **matrix**: elements organized into rows and columns. A single data type is allowed in a single matrix
- **data frame**: looks similar to a matrix, but allows different data types in different columns

9.2 Vector

9.2.1 Create Vector

Below are examples of atomic character vectors, numeric vectors, integer vectors, etc. There are many ways to create vector data. The following examples use `c()`, `:`, `seq()`, `rep()`:

```r
#ex.1a manually create a vector using c()
x <- c(1,3,4,8)
x
```

```
## [1] 1 3 4 8
```

```r
#ex.1b character
x <- c("a", "b", "c")
x
```

```
## [1] "a" "b" "c"
```

```r
#ex.1c logical
x <- c(TRUE, FALSE, FALSE)
x
```

```
## [1]  TRUE FALSE FALSE
```

```r
#ex.2 sequence of numbers
x <- 1:5
x
```

```
## [1] 1 2 3 4 5
```

```r
#ex.3a replicate same numbers or characters
x <- rep(2, 5) # replicate 2 five times
x
```

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

```r
#ex.3b replicate same numbers or characters
x <- rep("a", 5) # replicate "a" five times
x
```

```
## [1] "a" "a" "a" "a" "a"
```

```r
#ex.4a use seq() function
x <- seq(1, 5, by = 1)
x
```

```
## [1] 1 2 3 4 5
```

```r
#ex.4b use seq() function
x <- seq(1, 5, by = 0.1)
x
```

```
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8
## [20] 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7
## [39] 4.8 4.9 5.0
```

9.2.2 Check Features

R provides many functions to examine features of vectors and other objects, for example:

- `class()` - return high-level data structure of the object
- `typeof()` - return low-level data structure of the object
- `attributes()` - metadata of the object
- `length()` - number of elements in the object
- `sum()` - sum of object's elements
- `mean()` - mean of object's elements

Numeric Vector

```
x <- c(1.2, 3.1, 4.0, 8.2)
x
```

```
## [1] 1.2 3.1 4.0 8.2
```

```
class(x)
```

```
## [1] "numeric"
```

```
typeof(x)
```

```
## [1] "double"
```

```
length(x)
```

```
## [1] 4
```

```
sum(x)
```

```
## [1] 16.5
```

```
mean(x)
```

```
## [1] 4.125
```

Character Vector

```
y <- c("a", "b", "c")
class(y)
```

```
## [1] "character"
```

```
length(y)
```

```
## [1] 3
```

9.2.3 Access

Element ID

Use brackets `[]` when accessing specific elements in an object. For example, if

you want to access element #2 in the vector `x`, you may specify as `x[2]`:

```
x <- c(2,2,3,2,5)
x[2] # access element #2
```

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

```
x[c(2,4)] # access elements #2 and 4
```

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

```
x[2:4] # access elements #2-4
```

```
## [1] 2 3 2
```

Equation
R provides many ways to access elements that suffice specific conditions. You can use mathematical symbols to specify what you need, for example:

- `==` equal
- `>` larger than
- `>=` equal & larger than
- `<` smaller than
- `<=` equal & smaller than
- `which()` a function that returns element # that suffices the specified condition

The following examples return a logical vector indicating whether each element in x suffices the specified condition:

```
# creating a vector
x <- c(2,2,3,2,5)

# ex.1a equal
x == 2
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE
```

```
# ex.1b larger than
x > 2
```

```
## [1] FALSE FALSE  TRUE FALSE  TRUE
```

You can access elements that suffice the specified condition using brackets, for example:

```
# ex.2a equal
x[x == 2]
```

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

```
# ex.2b larger than
x[x > 2]
```

```
## [1] 3 5
```

Using `which()`, you can see which elements (i.e., #) matches what you need:

```
# ex.3a equal
which(x == 2) # returns which elements are equal to 2
```

```
## [1] 1 2 4
```

```
# ex.3b larger than
which(x > 2)
```

```
## [1] 3 5
```

9.3 Matrix

9.3.1 Create Matrix

Matrix is a set of elements (*single data type*) that are organized into rows and columns:

```
#ex.1 cbind: combine objects by column
x <- cbind(c(1,2,3), c(4,5,6))
x
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
#ex.2 rbind: combine objects by row
x <- rbind(c(1,2,3), c(4,5,6))
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
#ex.3 matrix: specify elements and the number of rows (nrow) and columns (ncol)
x <- matrix(1:9, nrow = 3, ncol = 3)
x
```

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

9.3.2 Check Features

R provides many functions to examine features of matrix data, for example:

- `dim()` number of rows and columns
- `rowSums()` row sums
- `colSums()` column sums

Integer Matrix

```
x <- matrix(1:9, nrow = 3, ncol = 3)
x
```

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

```
class(x)
```

```
## [1] "matrix" "array"
```

```
typeof(x)
```

```
## [1] "integer"
```

```
dim(x)
```

```
## [1] 3 3
```

Character Matrix

```
y <- matrix(c("a","b", "c", "d", "e", "f"), nrow = 3, ncol = 2)
y
```

```
##      [,1] [,2]
## [1,] "a"  "d"
## [2,] "b"  "e"
## [3,] "c"  "f"
```

```
class(y)
```

```
## [1] "matrix" "array"
```

```
typeof(y)
```

```
## [1] "character"
```

```
dim(y)
```

```
## [1] 3 2
```

9.3.3 Access

When accessing matrix elements, you need to pick row(s) and/or column(s), for example:

```
x <- matrix(1:9, nrow = 3, ncol = 3)
x
```

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

```
x[2,3] # access an element in row #2 and colum #3
```

```
## [1] 8
```

```
x[2,] # access elements in row #2
```

```
## [1] 2 5 8
```

```
x[c(2,3),] # access elements in rows #2 and 3
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    3    6    9
```

```
x[,c(2,3)] # access elements in columns #2 and 3
```

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

You can assess each element with mathematical expressions just like vectors:

```
x == 2 # equal
```

```
##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,]  TRUE FALSE FALSE
## [3,] FALSE FALSE FALSE
```

```
x > 2 # larger than
```

```
##       [,1] [,2] [,3]
## [1,] FALSE TRUE TRUE
## [2,] FALSE TRUE TRUE
## [3,]  TRUE TRUE TRUE
```

However, care must be taken when accessing elements, as it will be automatically converted to vector data:

```
x[x == 2] # equal
```

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

```
x[x > 2] # larger than
```

```
## [1] 3 4 5 6 7 8 9
```

`which()` needs an additional argument to return both row and column #:

```
which(x == 2, arr.ind = TRUE)
```

```
##      row col
## [1,]   2   1
```

```
which(x > 2, arr.ind = TRUE)
```

```
##      row col
## [1,]   3   1
## [2,]   1   2
## [3,]   2   2
## [4,]   3   2
## [5,]   1   3
## [6,]   2   3
## [7,]   3   3
```

9.4 Data Frame

A data frame is a collection of elements organized into rows and columns, but it differs from a matrix in several ways.

- It allows for the inclusion of *multiple data types* in different columns.
- Each column in a data frame has a *name* associated with it.
- You can access columns in a data frame by their respective names using the `$` operator.

The data frame is the most commonly used data structure when manipulating ecological data. When loading a dataset from a spreadsheet (which we will discuss later), it is automatically recognized as a data frame. Let's consider an example:

Creating a data frame
In the following example, the variables `x` and `y` are organized into a single data frame named `df0`. The variables are renamed as part of the process of creating the data frame.

```
# Create data frame
x <- c("Pristine", "Pristine", "Disturbed", "Disturbed", "Pristine") # Lake type
y <- c(1.2, 2.2, 10.9, 50.0, 3.0) # TSS: total suspended solids (mg/L)
df0 <- data.frame(LakeType = x, TSS = y) # x is named as "LakeType" while y is named a
df0
```

```
##    LakeType  TSS
```

```
## 1  Pristine  1.2
## 2  Pristine  2.2
## 3 Disturbed 10.9
## 4 Disturbed 50.0
## 5  Pristine  3.0
```

Call column names

```
colnames(df0) # call column names
```

```
## [1] "LakeType" "TSS"
```

Access by columns

```
df0$LakeType # access LakeType
```

```
## [1] "Pristine"  "Pristine"  "Disturbed" "Disturbed" "Pristine"
```

```
df0$TSS # access TSS
```

```
## [1]  1.2  2.2 10.9 50.0  3.0
```

You can access elements like a matrix as well:

```
df0[,1] # access column #1
```

```
## [1] "Pristine"  "Pristine"  "Disturbed" "Disturbed" "Pristine"
```

```
df0[1,] # access row #1
```

```
##   LakeType TSS
## 1 Pristine 1.2
```

```
df0[c(2,4),] # access row #2 and 4
```

```
##    LakeType  TSS
## 2  Pristine  2.2
## 4 Disturbed 50.0
```

9.5 Exercise

9.5.1 Vector

a. Create three numeric vectors with length 3, 6 and 20, respectively. Each vector must be created using different functions in R.
b. Create three character vectors with length 3, 6 and 20, respectively. Each vector must be created using different functions in R.
c. Copy the following script to your R script and perform the following analysis:

- Identify element IDs of `x` that are greater than `2.0`
- Identify element values of `x` that are greater than `2.0`

```
set.seed(1)
x <- rnorm(100)
```

9.5.2 Matrix

a. Create a numeric matrix with 4 rows and 4 columns. Each column must contain identical elements.
b. Create a numeric matrix with 4 rows and 4 columns. Each row must contain identical elements.
c. Create a character matrix with 4 rows and 4 columns. Each column must contain identical elements.
d. Create a character matrix with 4 rows and 4 columns. Each row must contain identical elements.
e. Copy the following script to your R script and perform the following analysis:

- Identify element IDs of `x` that are greater than `2.0` (**specify row and column IDs**)
- Identify element values of `x` that are greater than `2.0` and calculate the mean.

```
set.seed(1)
x <- matrix(rnorm(100), nrow = 10, ncol = 10)
```

9.5.3 Data Frame

a. Create a data frame of 3 variables with 10 elements (name variables as `x`, `y` and `z`. `x` must be `character` while `y` and `z` must be `numeric`.
b. Check the data structure (higher-level) of `x`, `y` and `z`
c. Copy the following script to your R script and perform the following analysis:

- Calculate the means of `temperature` and `abundance` for states `VA` and `NC` separately.

```
set.seed(1)
x <- rnorm(100, mean = 10, sd = 3)
y <- rpois(100, lambda = 10)
z <- rep(c("VA", "NC"), 50)
df0 <- data.frame(temperature = x, abundance = y, state = z)
```

Chapter 10

Appendix: Tidyverse

10.1 Overview

R packages are collections of code, data, and documentation that extend the functionality of the R programming language. They provide a convenient way for users to access and utilize specialized tools and functions for data analysis, visualization, statistical modeling, and more. Among them, `tidyverse` provides very useful functions for data manipulation and visualization.

`tidyverse` is a collection of R packages designed to make data manipulation, exploration, and visualization more efficient and intuitive. Developed by Hadley Wickham and other contributors, the Tidyverse packages share a common philosophy and syntax, emphasizing a consistent and tidy data format. The core packages, such as `dplyr`, `tidyr`, and `ggplot2`, provide powerful tools for data wrangling, reshaping, and creating visualizations.

10.2 Data Manipulation

10.2.1 Data Format

10.2.2 Select Rows

10.2.3 Select Columns

10.2.4 Group Operation

10.2.5 Reshape

10.2.6 Combination

10.3 Visualization

10.3.1 Point

10.3.2 Histogram

10.3.3 Boxplot

10.3.4 Violin Plot

10.3.5 Styling

10.3.6 More

Chapter 11

Appendix: Base Plot

11.1 Overview

R offers a variety of functions that aid in visualizing data. The `graphics` package in R provides a set of functions for basic graphics (list of functions). To demonstrate the functionality of these `graphics` functions, I will utilize the built-in `iris` dataset in R.

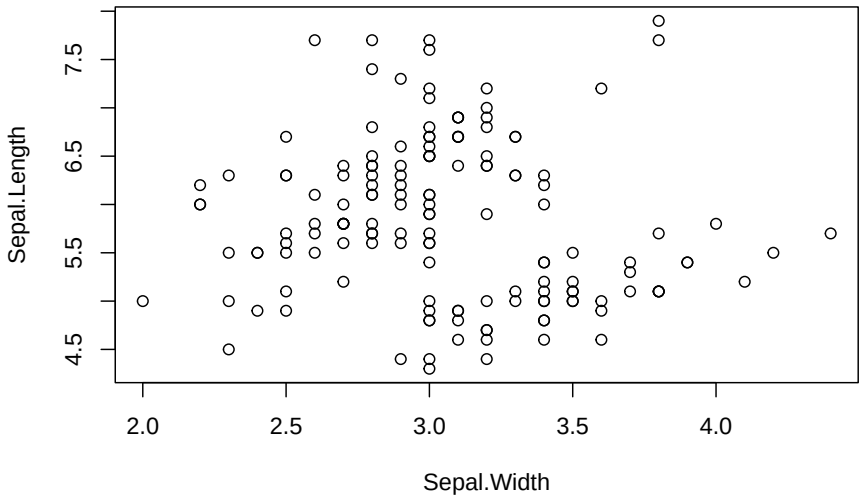
```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

11.2 Plot

When creating a plot, you typically need to specify the **formula** to define the relationship between variables. For instance, if you wish to visualize the association between `x` and `y` (with `y` on the vertical axis and `x` on the horizontal axis), the formula would be `y ~ x` (where the left side of the formula represents the vertical axis). In the `iris` dataset, you have access to the following columns: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`, and `Species`. In the subsequent example, we will plot the relationship between `Sepal.Length` and `Sepal.Width`:

```
plot(Sepal.Length ~ Sepal.Width, data = iris)
```

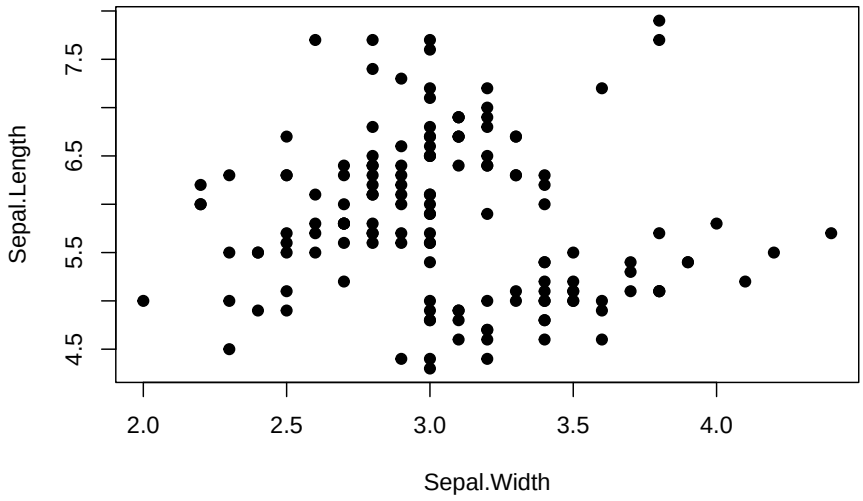


The `data =` argument informs the function about the dataset from which the variables (`Sepal.Length` and `Sepal.Width`) should be extracted.

11.2.1 Symbol

`pch` argument. Choose from `1` to `25` (google **r plot pch** for details)

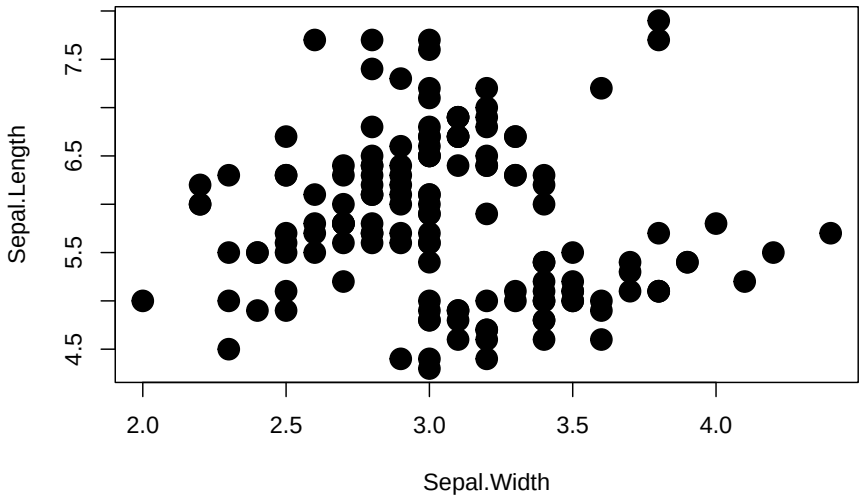
```
plot(Sepal.Length ~ Sepal.Width, data = iris,
     pch = 19)
```



11.2.2 Symbol size

`cex` argument. `cex = 1` is the default value. `cex = 2` is as twice large as default value.

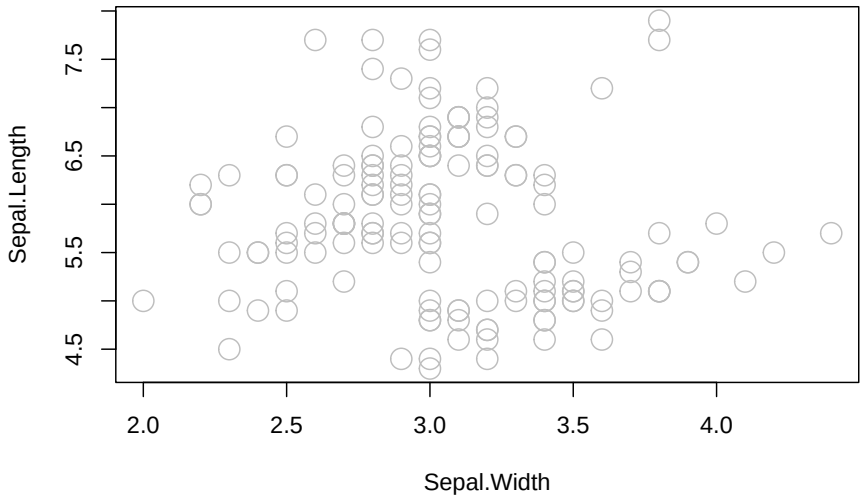
```
plot(Sepal.Length ~ Sepal.Width, data = iris,
     pch = 19, cex = 2)
```



11.2.3 Symbol color (border)

`col` argument (quote `"color name"` when specifying). Google **r color name** for color options.

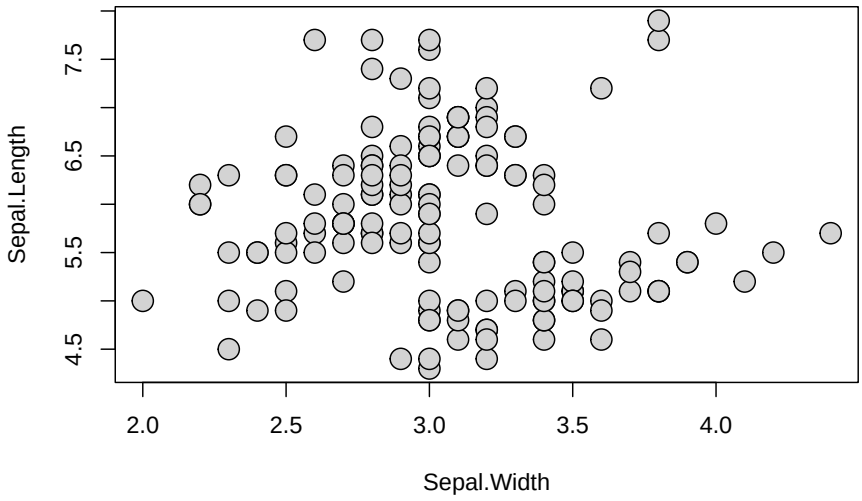
```
plot(Sepal.Length ~ Sepal.Width, data = iris,
     pch = 21, cex = 2, col = "gray")
```



11.2.4 Symbol color (fill)

`bg` argument (quote `"color name"` when specifying). Available for a subset of symbol options (some symbols have pre-defined filled color).

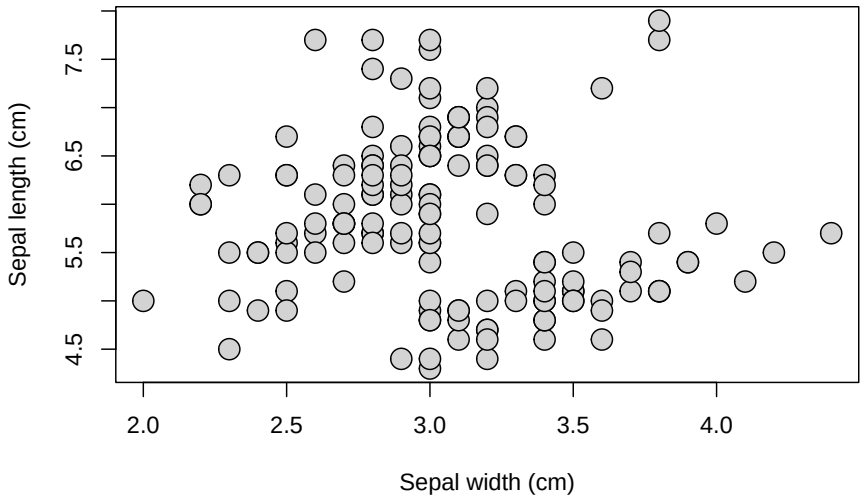
```
plot(Sepal.Length ~ Sepal.Width, data = iris,
     pch = 21, cex = 2, bg = "lightgray")
```



11.2.5 Label

`ylab` or `xlab` arguments. Provide `"quoted text"`.

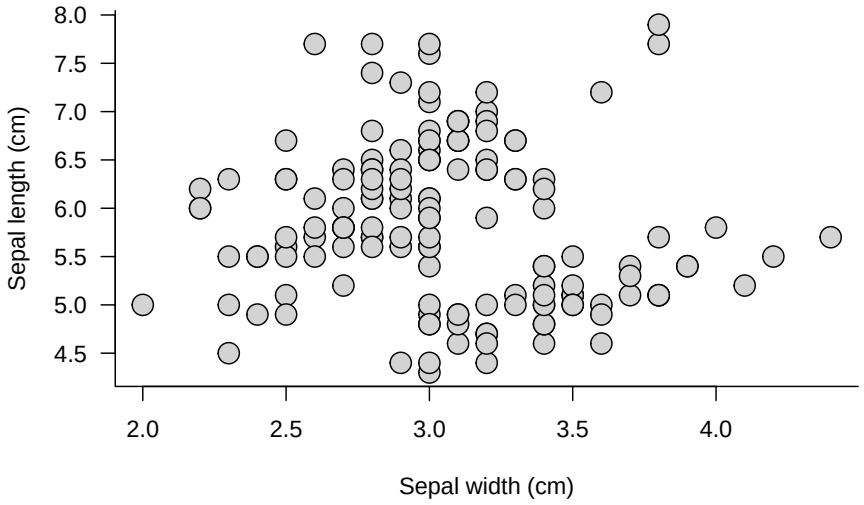
```
plot(Sepal.Length ~ Sepal.Width, data = iris,
     pch = 21, cex = 2, bg = "lightgray",
     xlab = "Sepal width (cm)", ylab = "Sepal length (cm)")
```



11.2.6 Axis

Delete axes with `axes = F` and re-draw with `box()` and `axis()` functions.

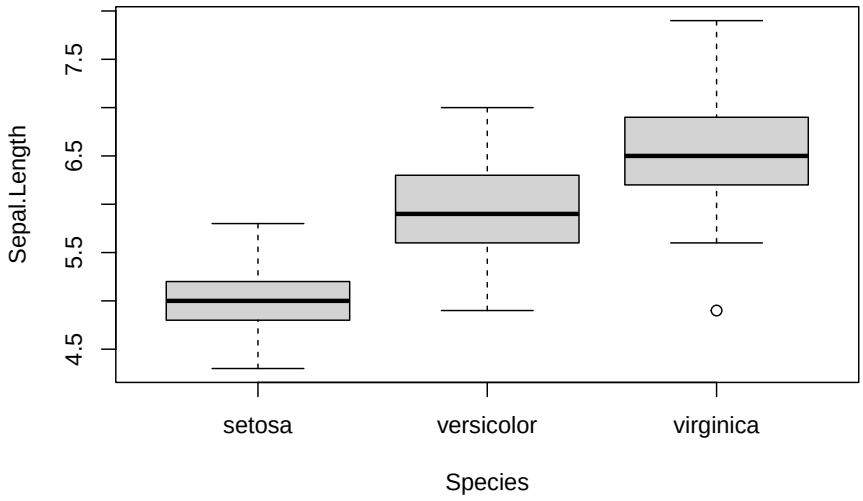
```
plot(Sepal.Length ~ Sepal.Width, data = iris,
     pch = 21, cex = 2, bg = "lightgray",
     xlab = "Sepal width (cm)", ylab = "Sepal length (cm)",
     axes = F)
box(bty = "l") # L-shaped border lines
axis(1) # 1: draw x-axis
axis(2, las = 2) # 2: draw y-axis, las = 2: make axis lables horizontal
```



11.3 Boxplot

`boxplot()` is used when the x-axis is factor-type data (by default, `plot()` will produce a boxplot when x-axis is a factor variable). In the `iris` dataset, the column `Species` is a factor variable. Compare `Sepal.Length` among species using `boxplot()`.

```
boxplot(Sepal.Length ~ Species, data = iris)
```

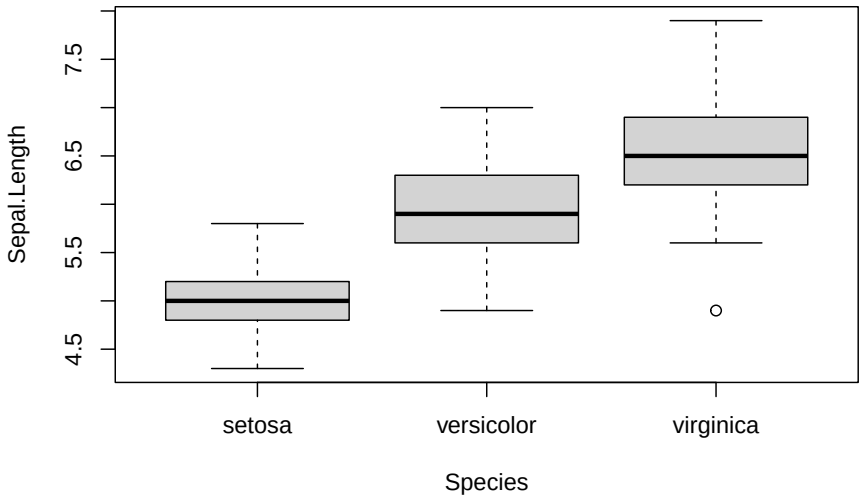


You can customize as in `plot()`, but slighlty different.

11.3.1 Box color

`col` argument.

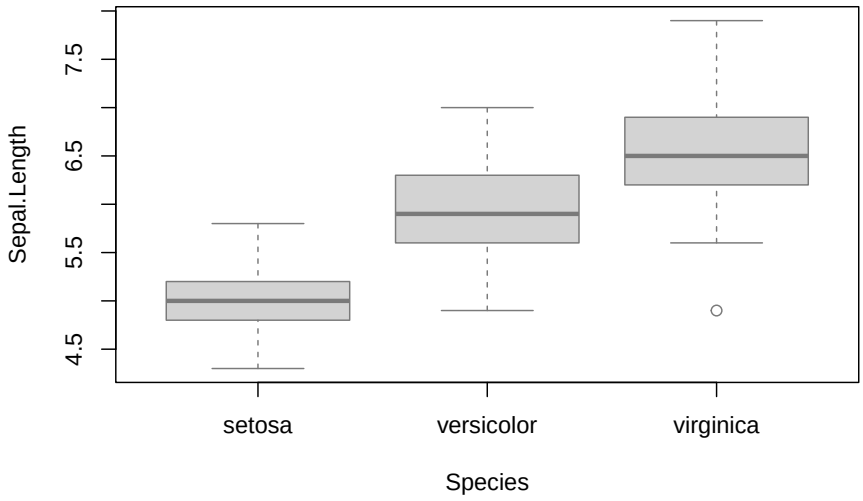
```
boxplot(Sepal.Length ~ Species, data = iris,
        col = "lightgray")
```



11.3.2 Border color

`border` argument.

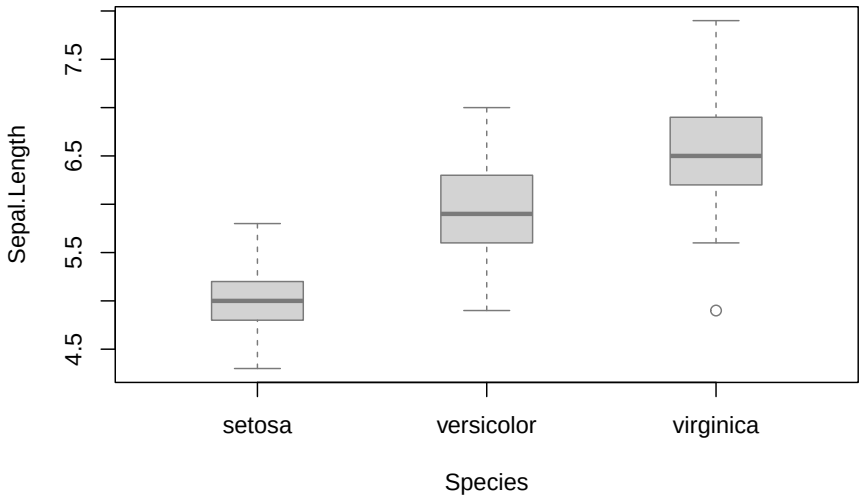
```
boxplot(Sepal.Length ~ Species, data = iris,
        col = "lightgray", border = "grey48")
```



11.3.3 Box width

`boxwex` argument.

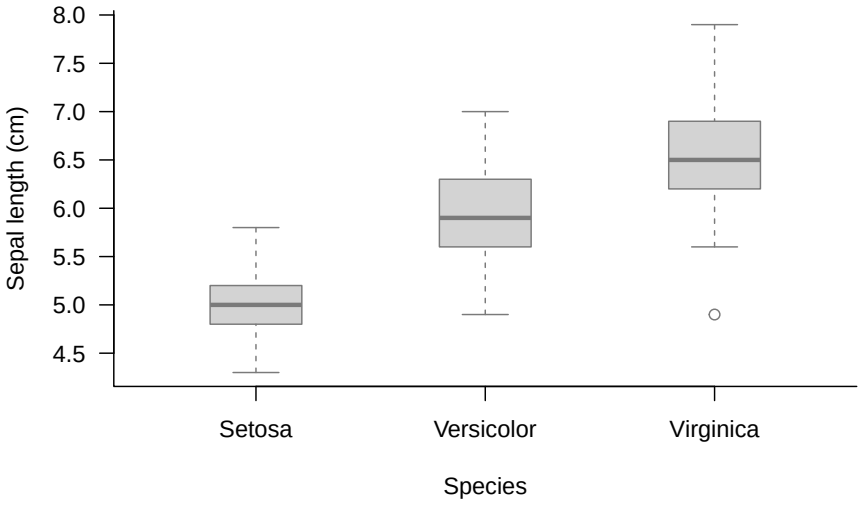
```
boxplot(Sepal.Length ~ Species, data = iris,
        col = "lightgray", border = "grey48",
        boxwex = 0.4 )
```



11.3.4 Axis

Delete axes with `axes = F` and re-draw with `box()` and `axis()` functions.

```
boxplot(Sepal.Length ~ Species, data = iris,
        col = "lightgray", border = "grey48",
        boxwex = 0.4, ylab = "Sepal length (cm)",
        axes = F)
box(bty = "l")
axis(1, at = c(1, 2, 3), labels = c("Setosa", "Versicolor", "Virginica") )
axis(2, las = 2)
```


8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
Sepal length (cm)
Setosa
Versicolor
Virginica
Species