Biostatistics using R

Dewey Brooke

2018-06-29

Contents

List of Tables

List of Figures

Preface

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

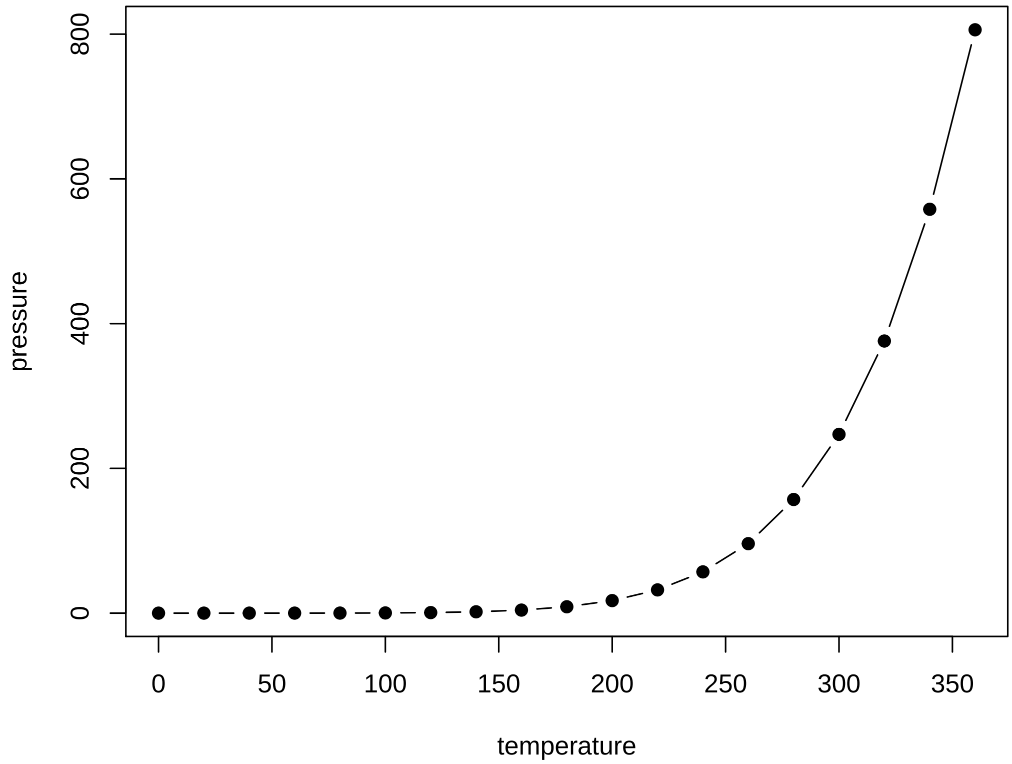


Figure 1: Here is a nice figure!

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

1 Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter 3.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Table 1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2018) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

Reading Data Files into R

The first step in every analysis requires data to be read into the environment, and learning how to do this is the first hurdle a person needs to overcome to begin learning to use R.

Data can exist in many different formats, either as the generic universal types (e.g. csv, tsv, .json, etc) or software specific types (e.g. `.xlsx`, “)

In this chapter, we will first discuss how to read data using functions in Base-R (when possible), and then we will discuss alternative packages, such as the multitude of packages in the Tidyverse, and highlight their advantages over Base-R functions.

1.1 Generic Formats

1.1.1 CSV- Comma Separated Values

The fields are separated by a comma , and are typically used for loading into spreadsheets.

For example:

```
csv_example_path <- "data/ASCII-comma/FEV.DAT.txt"

readLines(csv_example_path)[1:8]  # reads each line of the file
```

```
[1] "'Id','Age','FEV','Hgt','Sex','Smoke'"
[2] "301,9,1.708,57,0,0"
[3] "451,8,1.724,67.5,0,0"
[4] "501,7,1.72,54.5,0,0"
[5] "642,9,1.558,53,1,0"
[6] "901,9,1.895,57,1,0"
[7] "1701,8,2.336,61,0,0"
[8] "1752,6,1.919,58,0,0"
```

```
# Note: readLines(csv_example_path) is the same as
# readLines("data/ASCII-comma/FEV.DAT.txt")
```

In Base-R, CSV data can be read using the `read.csv()` function. The `read.csv2()` function is used in countires that use a comma as a decimal point and a semicolon as a field separator.

```
csv_example <- read.csv(csv_example_path)

head(csv_example)
```

```
  X.Id. X.Age. X.FEV. X.Hgt. X.Sex. X.Smoke.
1   301      9  1.708   57.0      0        0
2   451      8  1.724   67.5      0        0
3   501      7  1.720   54.5      0        0
4   642      9  1.558   53.0      1        0
5   901      9  1.895   57.0      1        0
6  1701      8  2.336   61.0      0        0
```

1.1.2 TSV- Tab Separeted Values

The fields are separated by a tabulation or ând are saved as `.txt` files. However, not all `.txt` files contain tab separated values.

For example:

```r
tsv_example_path <- "data/ASCII-tab/FEV.DAT.txt"

readLines(tsv_example_path)[1:8]
```

```
[1] "'Id'\t'Age'\t'FEV'\t'Hgt'\t'Sex'\t'Smoke'"
[2] "301\t9\t1.708\t57\t0\t0"
[3] "451\t8\t1.724\t67.5\t0\t0"
[4] "501\t7\t1.72\t54.5\t0\t0"
[5] "642\t9\t1.558\t53\t1\t0"
[6] "901\t9\t1.895\t57\t1\t0"
[7] "1701\t8\t2.336\t61\t0\t0"
[8] "1752\t6\t1.919\t58\t0\t0"
```

```r
tsv_example <- read.delim("data/ASCII-tab/FEV.DAT.txt")
head(tsv_example)
```

```
  X.Id. X.Age. X.FEV. X.Hgt. X.Sex. X.Smoke.
1   301      9  1.708   57.0      0        0
2   451      8  1.724   67.5      0        0
3   501      7  1.720   54.5      0        0
4   642      9  1.558   53.0      1        0
5   901      9  1.895   57.0      1        0
6  1701      8  2.336   61.0      0        0
```

1.2 Excel

```r
library(readxl)
```

1.3 Software Specific Formats

R is increasingly recognized as the gold standard for statistical computations, yet some of your future collaboraters will exclusively use Commercial Software (SAS, SPSS, Matlab, and Stata) for their statistical computations. Although these individuals are limited by the types of files they can read or write, the `haven` R-package can both read and write any of these file formats.

```r
library(haven)
```

1.3.1 SAS(`.sas7bdat`), SPSS(`.sav`,`.por`, `.xpt`), Stata (`.dta`)

```
sas <- read_sas("data/SAS/FEV.sas7bdat")

head(sas)
```

```
# A tibble: 6 x 6
     ID   AGE   FEV   HGT   SEX SMOKE
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   301     9  1.71  57       0     0
2   451     8  1.72  67.5     0     0
3   501     7  1.72  54.5     0     0
4   642     9  1.56  53       1     0
5   901     9  1.90  57       1     0
6  1701     8  2.34  61       0     0
```

```
spss <- read_spss("data/SPSS/FEV.DAT.sav")
head(spss)
```

```
# A tibble: 6 x 6
     Id   Age   FEV   Hgt   Sex Smoke
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   301     9  1.71  57       0     0
2   451     8  1.72  67.5     0     0
3   501     7  1.72  54.5     0     0
4   642     9  1.56  53       1     0
5   901     9  1.90  57       1     0
6  1701     8  2.34  61       0     0
```

```
stata <- read_stata("data/Stata/FEV.DAT.dta")
head(stata)
```

```
# A tibble: 6 x 6
     Id   Age   fev   Hgt   Sex Smoke
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   301     9  1.71  57       0     0
2   451     8  1.72  67.5     0     0
3   501     7  1.72  54.5     0     0
4   642     9  1.56  53       1     0
5   901     9  1.90  57       1     0
6  1701     8  2.34  61       0     0
```

The `foreign` package included in Base-R can also be used to Reading and writing data stored by some versions of 'Epi Info', 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka',and for reading and writing some 'dBase' files.

1.3.1.1 RDS

```
rds_example <- readRDS("data/RDS/BETACAR.DAT.rds")
head(rds_example)
```

```
# A tibble: 6 x 8
  `'Prepar'` `'Id'` `'Base1lvl'` `'Base2lvl'`
       <int>  <int>        <int>        <int>
1          1     71          298          116
2          1     73          124          146
3          1     80          176          200
4          1     83          116          180
5          1     90          152          142
6          1     92          106          106
# ... with 4 more variables: `'Wk6lvl'` <int>,
#   `'Wk8lvl'` <int>, `'Wk10lvl'` <int>,
#   `'Wk12lvl'` <int>
```

1.3.1.2 `rdata`

The `.rdata` format is R's specific format. Instead of using a `read.{something}` function, `.rdata` is read into the environment using `load(filename.rdata)` and retains the original name it had when it was last saved.

```
load("data/R/BETACAR.DAT.rdata")  #named betacar when it was last saved
head(betacar)
```

```
  Prepar Id Base1lvl Base2lvl Wk6lvl Wk8lvl Wk10lvl
1      1 71      298      116    174    178     218
2      1 73      124      146    294    278     244
3      1 80      176      200    276    286     308
4      1 83      116      180    164    238     308
5      1 90      152      142    290    300     270
6      1 92      106      106    246    206     304
  Wk12lvl
1     190
2     262
3     334
4     226
5     268
6     356
```

2 Chapter 2: Descriptive Statistics

2.1 Introduction

```
PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, please make s
```

2.2 Measures of Location using Base R

```r
head(ChickWeight)
```

```
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1
```

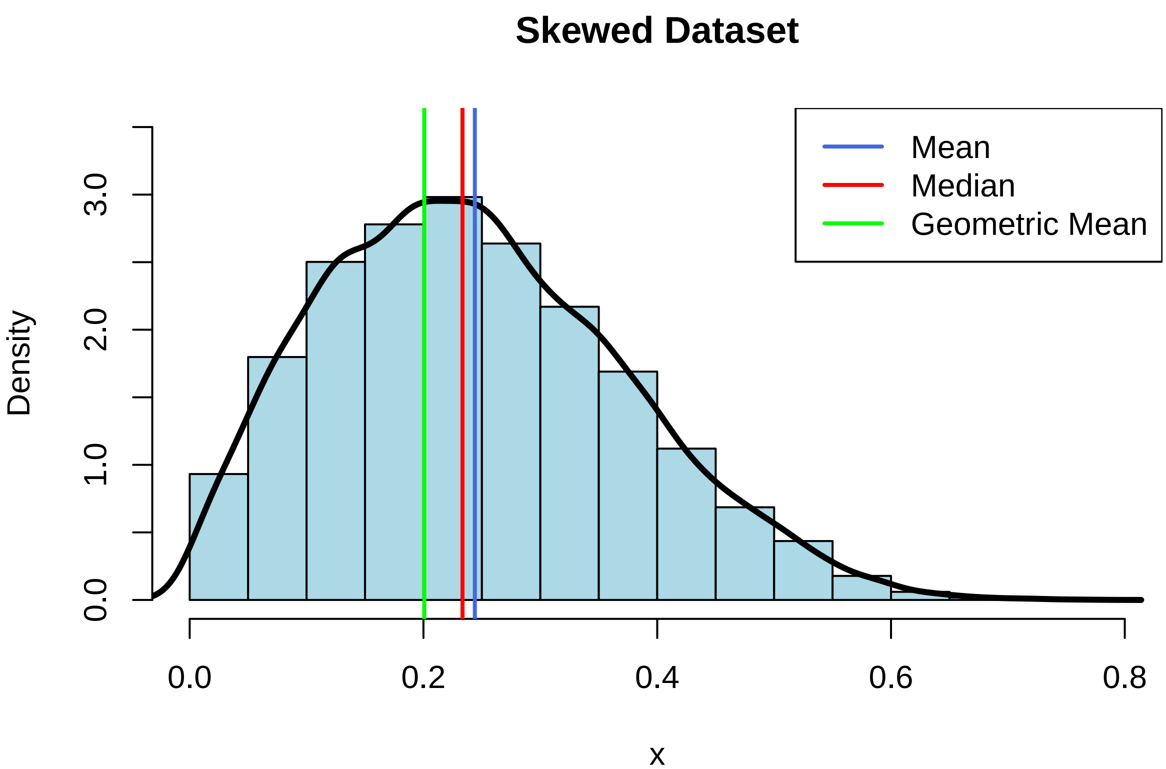
2.2.1 The Arithmetic Mean

The arithmetic mean is the sum of all the observations divided by the number of observations. It is written in statistical terms as

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```r
y= rbeta(10000,1,12,6)
hist(y, # histogram
 col = "lightblue", # column color
 border = "black",
 prob = TRUE, # show densities instead of frequencies
 xlab = "x",
 ylim = c(0,3.5),
 main = "Skewed Dataset"
 )

lines(density(y), col='black', lwd=3)
abline(v = mean(y),
 col = "royalblue",
 lwd = 2)

abline(v = median(y),
```

```r
 col = "red",
 lwd = 2)

abline(v = exp(mean(log(y))),
 col = "green",
 lwd = 2)



legend(x = "topright", # location of legend within plot area
 c("Mean", "Median", "Geometric Mean"),
 col = c( "royalblue", "red", "green"),
 lwd = c(2, 2, 2))
```




```r
mean(ChickWeight$weight)
```

```
[1] 121.8
```

2.2.2 The Median

```r
median(ChickWeight$weight)
```

```
[1] 103
```

2.2.3 The Mode

The mode is the most frequently occuring value among all observations in the sample. Although it is infrequently used, it is very useful for categorical and discrete data.

Since there isn't a built in R-function for mode, we learn how to write a function to return the mode through a few examples.

2.2.3.1 Functions

2.2.3.1.1 Base R Example

The most simple function begins by assigning the output of `function()` to some character string (e.g. `simple_fun`)

All statements after the `function()` are referred as the body of the function.

```
function_name <- function(arg1, arg2,...) {
  #statements

  return("some output")
}
function_name() # returns NULL
```

```
[1] "some output"
```

Use `return()` to output the result of the function.

```
return_value <- function(x,y) {
  z=x-y
  z=x+y
  return(z)
}
return_value(4,5)
```

```
[1] 9
```

Since our goal is to find the most frequently occuring value in our dataset (`ChickWeight`), we need to deside the sequence of functions that we need to accomplish this. As you continue to add various R functions to your R toolbelt, you will find many possible combinations for the same solution.

First, let's assign the weight column from ChickWeight to x to simplify things. When `x` is called, the weight column from ChickWeight is returned as a vector.

```
x<-ChickWeight$weight
head(x)
```

```
[1] 42 51 59 64 76 93
```

We can return the size of `x` using the `length` function. 578

```r
length(x)
```

```
[1] 578
```

We can reduce x to return only the unique values by using the `unique` function. We'll assign it to y so we can use it later.

```r
y <- unique(x)
length(y)
```

```
[1] 212
```

To more easily watch how the functions are working, we will create two dataframes to watch how we are manipulating both x and y.

```r
df.x <- data.frame(x)
df.y <- data.frame(y)
```

Using the unique values from the `x` vector we defined as `y`, we can use the `match` function to return a vector that replaces each value in x with their position in the y vector (1-212).

```r
df.x$position_in_y<-match(x, y)
head(df.x, n = 30)
```

```
     x position_in_y
1   42             1
2   51             2
3   59             3
4   64             4
5   76             5
6   93             6
7  106             7
8  125             8
9  149             9
10 171            10
11 199            11
12 205            12
13  40            13
14  49            14
15  58            15
16  72            16
17  84            17
18 103            18
19 122            19
20 138            20
21 162            21
```

```
22 187             22
23 209             23
24 215             24
25  43             25
26  39             26
27  55             27
28  67             28
29  84             17
30  99             29
```

The output from match can then be simplified using the tabulate function

```r
df.y$frequency <- tabulate(df.x$position_in_y)
head(df.y)
```

```
   y frequency
1 42        15
2 51         8
3 59         5
4 64         5
5 76         3
6 93         4
```

`which.max` returns the position of the maximum value.

```r
which.max(df.y$frequency)
```

```
[1] 43
```

```r
df.y[43,]  #df.y[row,column]
```

```
    y frequency
43 41        20
```

Putting it all together, we can do this in one line.

```r
df.y[which.max(tabulate(match(x,y))),]
```

```
    y frequency
43 41        20
```

```r
y[which.max(tabulate(match(x,y)))]
```

```
[1] 41
```

Writing this as a function

```r
mode <- function(x){
  unique_x <- unique(x)
  result<-unique_x[which.max(tabulate(match(x,unique_x)))]
  return(result)
}
```

```
mode(x)
```

```
[1] 41
```

2.2.3.1.2 Tidyverse Example

As with most problems in R, we can also find a solution using packages from the Tidyverse. We will therefore use this as an opportunity to introduce some of the basic tenants of Tidyverse functions.

In the `dplyr` package, a typical workflow will combine observations into a single dataframe, aggregate them into groups, manipulate values into new columns, and summarise the dataframe into more simple terms.

The piping operator `%>%` allows for this to be done seamlessly by literally pipping the result of one function into arguments of another function.

```
print("non-piped text")
```

```
[1] "non-piped text"
```

```
library(dplyr)
"piped text" %>% print()
```

```
[1] "piped text"
```

To show how this works, we will start with a simple example where we first want to divided the sum of three and some other number (e.g. 2) by seven.

Because of the order of operations, the sum of two and three would need to be placed with parenthesis to indicate it happens before dividing by seven.

```
(4+3)/7 # correct
```

```
[1] 1
```

```
4 + 3 / 7 # incorrect
```

```
[1] 4.429
```

The piping operator allows the order of operations be explicated dictated with manipulations of starting value reading from the left to right.

```
# pipes use the (.) as a placeholder
4 %>% + 3 %>% {./7} # removing the { } returns an error
```

```
[1] 1
```

Using pipes increases readability of your R-code and it can easily be reused for different starting values. In RStudio, the pipe character can be easily inserted using a keyboard shortcut (Windows:Ctrl+Shift+M, Mac:Cmd+Shift+M).

```
11 %>% + 3 %>% {./7}
```

```
[1] 2
```

Plus, the piped workflow can easily be defined by a function by assigning it to some string with a `.` in the beginning.

```
op_order <- . %>% +3 %>% {./7}
op_order(4)
```

```
[1] 1
```

```
op_order(11)
```

```
[1] 2
```

Determining Mode with `dplyr`

Using the `ChickWeight` dataset as before, we start by outlining the order of operations.

1. Group the data by weights `group_by()`
2. Tally the number of members within each group and sort by frequency. `tally()`
3. Select the row with the largest n. `slice()`
4. Return the corresponding weight. `.$weight`

```
ChickWeight %>% group_by(weight) %>% tally(sort = TRUE) %>% slice(1) %>% .$weight
```

```
[1] 41
```

As before, this workflow can be written as a function by placing `.` between the assignment opperator `<-` and piping operator `%>%`.

```
mode_cw<-. %>% group_by(weight) %>% tally(sort = TRUE) %>% slice(1) %>% .$weight


mode_cw(ChickWeight)
```

```
[1] 41
```

However, this function will only work on the `ChickWeight` dataset.

```
mode_cw(mtcars)
```

```
Error in grouped_df_impl(data, unname(vars), drop): Column `weight` is unknown
```

3 Methods

We describe our methods in this chapter.

4 Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

5 Final Words

We have finished a nice book.

References

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.