Introduction to R

Pachalo Chizala

2023-08-30

# Background

This is a living document… It gets updated as we move. You can add a section This document is developed in R with R Markdown, and hence it is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see. <http://rmarkdown.rstudio.com>. You combine code with text. No need to copy tables from one software such as STATA and paste them to MS Word. To demonstrate, check here.

# Introduction to R

In this journey, we seek to build our confidence working with R including data management and analysis using a statistical software called R- just R. Learning R by skill

Let’s get back to our business

# Dive straight in R

## Using R as a calculator

We can use R as a calculator. It follows the rules of arithmetic

#R works as calculator  
2+8

## [1] 10

2\*4+6-5/2.5

## [1] 12

(2\*4)+6-5/2.5

## [1] 12

5^2

## [1] 25

5\*\*2

## [1] 25

We can use R to compare expressions including numbers and letters, i.e.

#Logical operations  
4==2 #Is 4 equal to 2?

## [1] FALSE

4==4 #Is 4 equal to 4

## [1] TRUE

4>=3 #Is 4 greater than or equal to 3

## [1] TRUE

"a"=="b" #Is a equal to b

## [1] FALSE

# Objects, functions and Variables

From time to time we are faced with situations where we have to reference to some value. For instance, you might add two number 9 and 34, and keep the result for later calculations. In R, we can keep this result in an variable object named “a”. You can view the contents of this variable by just typing the variable name and place enter. Check the following examples

a = 9+34  
b = 4\*5+7  
a

## [1] 43

b

## [1] 27

a + b

## [1] 70

(a+b)/a

## [1] 1.627907

a=a^4-sqrt(a)  
a

## [1] 3418794

R is so fluid and very flexible. Just take caution that it overwrites an existing object (or variable) without issuing a warning. You might have noted this in the last commands.

b = sqrt(a\*runif(1)\*10)  
if (a > b) {print("a is greater than b")} else {print("a is less than b")}

## [1] "a is greater than b"

Suppose you are tasked to collect information such as name, age and sex of your colleagues at your place of work. You collect and write them on a paper. Probably, the next question would be “how do I feed then into R?”.

To this far, we need existing functions to help ease the work at hand, and (almost) all the work we will lay our hands on in R. To this far, we have already used three functions, namely, sqrt(), runif() and print(). One important but simply named function is the c() function. The c() function concatenates different entries, separated by commas, into a vector of the entries.

Now that we know the c() function, we can make use of it. Take note that you need to save your vector of values to an object of name of your choice. It is, however, important to name your objects or variables with meaningful names.

name <- c("Patrick", "Gregory", "Bernard", "Lesla", "Bridget", "Rico", "Temwa", "Andrew", "Cecil","Martha","Merriam", "William", "Martha","Mada","Sara",NA)  
age <- c(23,29,31,21,34,38,28,33,25,NA,30,35,33,30,29,NA)  
sex <- c(1,1,1,2,2,1,2,1,2,2,2,1,2,2,2,1)

If you just want to view the entries for each variable/object, you can print the contents using the function print() or just typing name of the object.

It would definitely be useful to check further if the list matches the number of colleagues you collected data for.

#Print values contained in each variable name, age, and sex  
print(name)

## [1] "Patrick" "Gregory" "Bernard" "Lesla" "Bridget" "Rico" "Temwa"   
## [8] "Andrew" "Cecil" "Martha" "Merriam" "William" "Martha" "Mada"   
## [15] "Sara" NA

age

## [1] 23 29 31 21 34 38 28 33 25 NA 30 35 33 30 29 NA

sex

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

#Returns number of items   
length(name)

## [1] 16

length(age)

## [1] 16

length(sex)

## [1] 16

## Data frames

Your thoughts are as good as mine if you ever wondered if it were possible to have variables, name, age and sex appear in a structured spreadsheet-like display. Possibly you would have guessed to use the same c() function to concatenate the variables… as in c(name, age, sex).. but this won’t give the required result. And since c() only stores values of the same type, you can guess what type of values you will now have.

There are, of course, a number of functions that combine vectors including cbind() and rbind(), but one you will likely come to work more with is the data.frame() function. Let’s dive into it…

cbind(name, age, sex) #Combines the variables column-wise

## name age sex  
## [1,] "Patrick" "23" "1"  
## [2,] "Gregory" "29" "1"  
## [3,] "Bernard" "31" "1"  
## [4,] "Lesla" "21" "2"  
## [5,] "Bridget" "34" "2"  
## [6,] "Rico" "38" "1"  
## [7,] "Temwa" "28" "2"  
## [8,] "Andrew" "33" "1"  
## [9,] "Cecil" "25" "2"  
## [10,] "Martha" NA "2"  
## [11,] "Merriam" "30" "2"  
## [12,] "William" "35" "1"  
## [13,] "Martha" "33" "2"  
## [14,] "Mada" "30" "2"  
## [15,] "Sara" "29" "2"  
## [16,] NA NA "1"

rbind(name, age, sex) #Combines the variables low-wise.

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]   
## name "Patrick" "Gregory" "Bernard" "Lesla" "Bridget" "Rico" "Temwa" "Andrew"  
## age "23" "29" "31" "21" "34" "38" "28" "33"   
## sex "1" "1" "1" "2" "2" "1" "2" "1"   
## [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]  
## name "Cecil" "Martha" "Merriam" "William" "Martha" "Mada" "Sara" NA   
## age "25" NA "30" "35" "33" "30" "29" NA   
## sex "2" "2" "2" "1" "2" "2" "2" "1"

data.frame(name, age, sex) #Combines the variables column-wise. This is the best.

## name age sex  
## 1 Patrick 23 1  
## 2 Gregory 29 1  
## 3 Bernard 31 1  
## 4 Lesla 21 2  
## 5 Bridget 34 2  
## 6 Rico 38 1  
## 7 Temwa 28 2  
## 8 Andrew 33 1  
## 9 Cecil 25 2  
## 10 Martha NA 2  
## 11 Merriam 30 2  
## 12 William 35 1  
## 13 Martha 33 2  
## 14 Mada 30 2  
## 15 Sara 29 2  
## 16 <NA> NA 1

#Saving the data frame to an object called mydata  
mydata <- data.frame(name, age, sex)  
mydata

## name age sex  
## 1 Patrick 23 1  
## 2 Gregory 29 1  
## 3 Bernard 31 1  
## 4 Lesla 21 2  
## 5 Bridget 34 2  
## 6 Rico 38 1  
## 7 Temwa 28 2  
## 8 Andrew 33 1  
## 9 Cecil 25 2  
## 10 Martha NA 2  
## 11 Merriam 30 2  
## 12 William 35 1  
## 13 Martha 33 2  
## 14 Mada 30 2  
## 15 Sara 29 2  
## 16 <NA> NA 1

Now we see better! In R, NA implies a missing value (Not applicable). Currently, we would say that we don’t yet have Martha’s age, and there is a certain male with no name and age. We will likely ask you to go back and fetch for these values!

Again, note now that we have two Marthas, and, obviously, it will not be easy to distinguish them using names. However, we can assign each person a unique identification number.

id <- 1:length(name)  
mydata <- data.frame(id,mydata)  
mydata

## id name age sex  
## 1 1 Patrick 23 1  
## 2 2 Gregory 29 1  
## 3 3 Bernard 31 1  
## 4 4 Lesla 21 2  
## 5 5 Bridget 34 2  
## 6 6 Rico 38 1  
## 7 7 Temwa 28 2  
## 8 8 Andrew 33 1  
## 9 9 Cecil 25 2  
## 10 10 Martha NA 2  
## 11 11 Merriam 30 2  
## 12 12 William 35 1  
## 13 13 Martha 33 2  
## 14 14 Mada 30 2  
## 15 15 Sara 29 2  
## 16 16 <NA> NA 1

I hope you are able to follow what’s happening here! Let me repeat the statement I once said above, “R is so fluid and very flexible”; you can achieve one task a million ways. Of course, this can also be a little confusing at times. We will soon look at how we can achieve the same task in a different way. For now, I would encourage that you experiment the use of a:b, seq(a,b,c) where a,b and c are numeric values.

But now, we have each person assigned a unique identification number.

## Having a feel of our data

Now we have our data, but how do we get a feel of it? If you were handed down a data set,you would want to know what type of data that it is, what variables are in it, and number of cases it contains.

The str() function re

str(mydata)

## 'data.frame': 16 obs. of 4 variables:  
## $ id : int 1 2 3 4 5 6 7 8 9 10 ...  
## $ name: chr "Patrick" "Gregory" "Bernard" "Lesla" ...  
## $ age : num 23 29 31 21 34 38 28 33 25 NA ...  
## $ sex : num 1 1 1 2 2 1 2 1 2 2 ...

dim(mydata)

## [1] 16 4

names(mydata)

## [1] "id" "name" "age" "sex"

head(mydata) #print the first default 6 rows

## id name age sex  
## 1 1 Patrick 23 1  
## 2 2 Gregory 29 1  
## 3 3 Bernard 31 1  
## 4 4 Lesla 21 2  
## 5 5 Bridget 34 2  
## 6 6 Rico 38 1

head(mydata,5) #print the first n=5 rows

## id name age sex  
## 1 1 Patrick 23 1  
## 2 2 Gregory 29 1  
## 3 3 Bernard 31 1  
## 4 4 Lesla 21 2  
## 5 5 Bridget 34 2

tail(mydata)

## id name age sex  
## 11 11 Merriam 30 2  
## 12 12 William 35 1  
## 13 13 Martha 33 2  
## 14 14 Mada 30 2  
## 15 15 Sara 29 2  
## 16 16 <NA> NA 1

tail(mydata,3)

## id name age sex  
## 14 14 Mada 30 2  
## 15 15 Sara 29 2  
## 16 16 <NA> NA 1

mydata <- data.frame(id = 1:length(name), mydata)  
mydata

## id id.1 name age sex  
## 1 1 1 Patrick 23 1  
## 2 2 2 Gregory 29 1  
## 3 3 3 Bernard 31 1  
## 4 4 4 Lesla 21 2  
## 5 5 5 Bridget 34 2  
## 6 6 6 Rico 38 1  
## 7 7 7 Temwa 28 2  
## 8 8 8 Andrew 33 1  
## 9 9 9 Cecil 25 2  
## 10 10 10 Martha NA 2  
## 11 11 11 Merriam 30 2  
## 12 12 12 William 35 1  
## 13 13 13 Martha 33 2  
## 14 14 14 Mada 30 2  
## 15 15 15 Sara 29 2  
## 16 16 16 <NA> NA 1

head(mydata) #print the first default 6 rows

## id id.1 name age sex  
## 1 1 1 Patrick 23 1  
## 2 2 2 Gregory 29 1  
## 3 3 3 Bernard 31 1  
## 4 4 4 Lesla 21 2  
## 5 5 5 Bridget 34 2  
## 6 6 6 Rico 38 1

head(mydata,5) #print the first n=5 rows

## id id.1 name age sex  
## 1 1 1 Patrick 23 1  
## 2 2 2 Gregory 29 1  
## 3 3 3 Bernard 31 1  
## 4 4 4 Lesla 21 2  
## 5 5 5 Bridget 34 2

tail(mydata)

## id id.1 name age sex  
## 11 11 11 Merriam 30 2  
## 12 12 12 William 35 1  
## 13 13 13 Martha 33 2  
## 14 14 14 Mada 30 2  
## 15 15 15 Sara 29 2  
## 16 16 16 <NA> NA 1

tail(mydata,3)

## id id.1 name age sex  
## 14 14 14 Mada 30 2  
## 15 15 15 Sara 29 2  
## 16 16 16 <NA> NA 1

### Work practice

Suppose that your friend asks you to assign the first 10 of your colleagues to department A and the rest to department B. Sure you can achieve this a number of ways, but your final output should use the function rep(). Without any explanation, you may compare with the following:

mydata <- data.frame(mydata,dept=rep(c("A","B"),c(10,6)))

# Indexing elememnts

There are times we want information for specific individuals

mydata[4,1]

## [1] 4

mydata[4,2]

## [1] 4

mydata[c(4),]

## id id.1 name age sex dept  
## 4 4 4 Lesla 21 2 A

#What id you wanted to infor for person on row number 1 and 2. What's your guess?  
mydata[c(5,4),c(2,4)]

## id.1 age  
## 5 5 34  
## 4 4 21

mydata[c(5,4,5),c(2,4)] #Everything is possible ;)

## id.1 age  
## 5 5 34  
## 4 4 21  
## 5.1 5 34

mydata[c(4),2]

## [1] 4

mydata[c(4,1),2]

## [1] 4 1

mydata[4,]

## id id.1 name age sex dept  
## 4 4 4 Lesla 21 2 A

mydata[c(4),] #If you don't specify any column after the comma, it selects all available columns

## id id.1 name age sex dept  
## 4 4 4 Lesla 21 2 A

mydata[,c(2,4)] #Similarly, if you don't specify any row, it selects all available rows

## id.1 age  
## 1 1 23  
## 2 2 29  
## 3 3 31  
## 4 4 21  
## 5 5 34  
## 6 6 38  
## 7 7 28  
## 8 8 33  
## 9 9 25  
## 10 10 NA  
## 11 11 30  
## 12 12 35  
## 13 13 33  
## 14 14 30  
## 15 15 29  
## 16 16 NA

And we can specify name of the column, we don’t always have to remember the column number of our variable of interest

mydata[1:3,"name"]

## [1] "Patrick" "Gregory" "Bernard"

mydata[1:3,c("name", "sex")]

## name sex  
## 1 Patrick 1  
## 2 Gregory 1  
## 3 Bernard 1

mydata[,"name"]

## [1] "Patrick" "Gregory" "Bernard" "Lesla" "Bridget" "Rico" "Temwa"   
## [8] "Andrew" "Cecil" "Martha" "Merriam" "William" "Martha" "Mada"   
## [15] "Sara" NA

Hopefully you get the idea. Suppose we only have the variables age, name, sex in the data frame

rm(age, name, sex, a, b)  
  
#mean(age) #This won't work now! Why? And what's your guess on this one below?  
mean(mydata[,"age"])

## [1] NA

mean(mydata[,"age"], na.rm = T) #The use of na.rm has been emphasized enough

## [1] 29.92857

mean(mydata["age"], na.rm = T) #Why won't this work?

## Warning in mean.default(mydata["age"], na.rm = T): argument is not numeric or  
## logical: returning NA

## [1] NA

An alternative (probably the the best) way to reference a variable in a data frame, is to use a dollar ($) sign as below

mydata$age

## [1] 23 29 31 21 34 38 28 33 25 NA 30 35 33 30 29 NA

head(mydata$age,7)

## [1] 23 29 31 21 34 38 28

mydata$age[6]

## [1] 38

mydata$age[c(6,3)]

## [1] 38 31

mean(mydata$age, na.rm = T)

## [1] 29.92857

table(mydata$sex)

##   
## 1 2   
## 7 9

#Table of proportions  
  
prop.table(1)

## [1] 1

prop.table(c(1,1))

## [1] 0.5 0.5

prop.table(c(1,1,2)) #I hope now you get the idea. Check this

## [1] 0.25 0.25 0.50

1/sum(c(1,1,2)) #First element

## [1] 0.25

1/sum(c(1,1,2)) #Second elemenet

## [1] 0.25

2/sum(c(1,1,2)) #Third element

## [1] 0.5

c(1,1,2)/4 #How cool!!

## [1] 0.25 0.25 0.50

c(1,1,2)/sum(c(1,1,2))

## [1] 0.25 0.25 0.50

a <- c(1,1,2)  
a/sum(a)

## [1] 0.25 0.25 0.50

Let’s get back to our problem

prop.table(c(1,1,2))

## [1] 0.25 0.25 0.50

prop.table(a)

## [1] 0.25 0.25 0.50

Just doing the same thing over and over. The idea is, if you pass a vector of numbers to prop.table, the function will calculate the the proportion of each value to the sum of all elements in the vector. Assume that 1 represents “male” and 2 represents “female”. This means, we have 2 males and 1 female. So, we can get the proportions as below.

prop.table(c(2,1)) #Does this make sense?

## [1] 0.6666667 0.3333333

We know we can get these counts of 2 males and 1 female using table function

table(c(1,1,2)) #Don't get confused with the output. Then,

##   
## 1 2   
## 2 1

prop.table(table(c(1,1,2))) #Does this make sense?

##   
## 1 2   
## 0.6666667 0.3333333

We know, if a = c(1,1,2), then

prop.table(table(a))

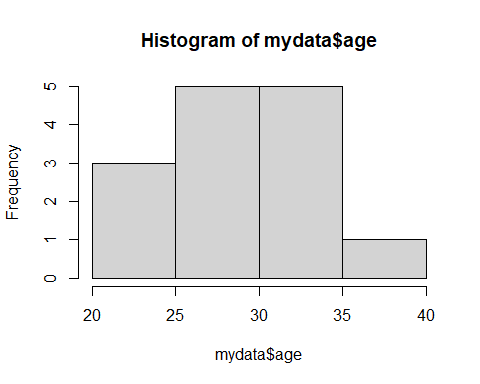
## a  
## 1 2   
## 0.6666667 0.3333333

Now, we get back to our data,

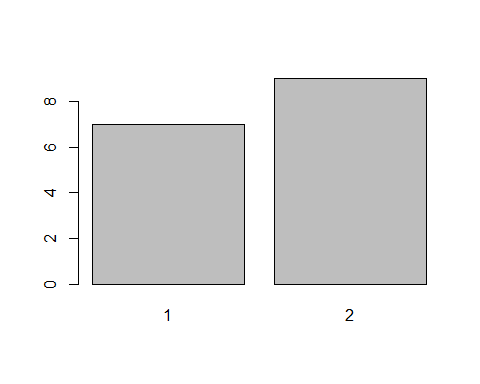
prop.table(table(mydata$sex))

##   
## 1 2   
## 0.4375 0.5625

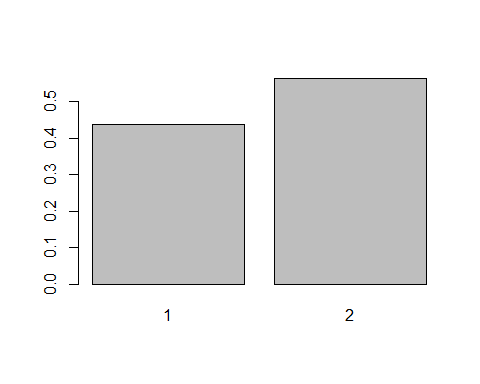
#Basic graphs  
hist(mydata$age)



barplot(table(mydata$sex)) #Barplot of frequencies of sex categories



barplot(prop.table(table(mydata$sex))) #Barplot of proportions of sex categories



#Note the chain.... function of a function of a function of a function.... ;)  
#Will get back to graphs later. Now, how would you select only males. Revisit   
#the indexing section.   
  
mydata[mydata$sex==1,]

## id id.1 name age sex dept  
## 1 1 1 Patrick 23 1 A  
## 2 2 2 Gregory 29 1 A  
## 3 3 3 Bernard 31 1 A  
## 6 6 6 Rico 38 1 A  
## 8 8 8 Andrew 33 1 A  
## 12 12 12 William 35 1 B  
## 16 16 16 <NA> NA 1 B

mydata[mydata$sex==2,]

## id id.1 name age sex dept  
## 4 4 4 Lesla 21 2 A  
## 5 5 5 Bridget 34 2 A  
## 7 7 7 Temwa 28 2 A  
## 9 9 9 Cecil 25 2 A  
## 10 10 10 Martha NA 2 A  
## 11 11 11 Merriam 30 2 B  
## 13 13 13 Martha 33 2 B  
## 14 14 14 Mada 30 2 B  
## 15 15 15 Sara 29 2 B

mydata[mydata$sex==2,c("name","sex","age")]

## name sex age  
## 4 Lesla 2 21  
## 5 Bridget 2 34  
## 7 Temwa 2 28  
## 9 Cecil 2 25  
## 10 Martha 2 NA  
## 11 Merriam 2 30  
## 13 Martha 2 33  
## 14 Mada 2 30  
## 15 Sara 2 29

mydata$age[mydata$sex==1] #Print ages for males

## [1] 23 29 31 38 33 35 NA

mean(mydata$age[mydata$sex==1], na.rm = T) #Avearge age for males

## [1] 31.5

mean(mydata$age[mydata$sex==2], na.rm = T) #Avearge age for females

## [1] 28.75

aggregate(age~sex, data = mydata, mean) #Print average for sex categories

## sex age  
## 1 1 31.50  
## 2 2 28.75

#Merging data #Lets create some data to illustrate the concept #########################################################################

id <- sample(1:20, 7, replace = F )  
ed <- factor(sample(1:3,7,replace = T), levels = c(1,2,3), labels = c("none","formal","informal"))  
  
edu.data <- data.frame(id, ed)  
  
#Merge using common variable "id"  
  
merge(mydata,edu.data, by = "id")

## id id.1 name age sex dept ed  
## 1 2 2 Gregory 29 1 A formal  
## 2 3 3 Bernard 31 1 A none  
## 3 5 5 Bridget 34 2 A informal  
## 4 8 8 Andrew 33 1 A none  
## 5 12 12 William 35 1 B none  
## 6 16 16 <NA> NA 1 B formal

merge(mydata,edu.data, by = "id", all.x = T)

## id id.1 name age sex dept ed  
## 1 1 1 Patrick 23 1 A <NA>  
## 2 2 2 Gregory 29 1 A formal  
## 3 3 3 Bernard 31 1 A none  
## 4 4 4 Lesla 21 2 A <NA>  
## 5 5 5 Bridget 34 2 A informal  
## 6 6 6 Rico 38 1 A <NA>  
## 7 7 7 Temwa 28 2 A <NA>  
## 8 8 8 Andrew 33 1 A none  
## 9 9 9 Cecil 25 2 A <NA>  
## 10 10 10 Martha NA 2 A <NA>  
## 11 11 11 Merriam 30 2 B <NA>  
## 12 12 12 William 35 1 B none  
## 13 13 13 Martha 33 2 B <NA>  
## 14 14 14 Mada 30 2 B <NA>  
## 15 15 15 Sara 29 2 B <NA>  
## 16 16 16 <NA> NA 1 B formal

merge(mydata,edu.data, by = "id", all.y = T)

## id id.1 name age sex dept ed  
## 1 2 2 Gregory 29 1 A formal  
## 2 3 3 Bernard 31 1 A none  
## 3 5 5 Bridget 34 2 A informal  
## 4 8 8 Andrew 33 1 A none  
## 5 12 12 William 35 1 B none  
## 6 16 16 <NA> NA 1 B formal  
## 7 18 NA <NA> NA NA <NA> none

merge(mydata,edu.data, by = "id", all = T)

## id id.1 name age sex dept ed  
## 1 1 1 Patrick 23 1 A <NA>  
## 2 2 2 Gregory 29 1 A formal  
## 3 3 3 Bernard 31 1 A none  
## 4 4 4 Lesla 21 2 A <NA>  
## 5 5 5 Bridget 34 2 A informal  
## 6 6 6 Rico 38 1 A <NA>  
## 7 7 7 Temwa 28 2 A <NA>  
## 8 8 8 Andrew 33 1 A none  
## 9 9 9 Cecil 25 2 A <NA>  
## 10 10 10 Martha NA 2 A <NA>  
## 11 11 11 Merriam 30 2 B <NA>  
## 12 12 12 William 35 1 B none  
## 13 13 13 Martha 33 2 B <NA>  
## 14 14 14 Mada 30 2 B <NA>  
## 15 15 15 Sara 29 2 B <NA>  
## 16 16 16 <NA> NA 1 B formal  
## 17 18 NA <NA> NA NA <NA> none

#Section: Sorting/Ordering data. Let’s look at the following functions. Take a closer look at the output

a <- c(7,4,9,1)  
sort(a) #This sorts individual elements in ascending order

## [1] 1 4 7 9

sort(a) #This sorts individual elements in ascending order

## [1] 1 4 7 9

order(a)

## [1] 4 2 1 3

#What is the smallest number in "a"; what position is it?  
#What is the second smallest number in "a"; what position is it?   
#Compare the output. Remember   
  
mydata[c(2,6,3),]

## id id.1 name age sex dept  
## 2 2 2 Gregory 29 1 A  
## 6 6 6 Rico 38 1 A  
## 3 3 3 Bernard 31 1 A

mydata[order(mydata$sex),] #Sort data by sex

## id id.1 name age sex dept  
## 1 1 1 Patrick 23 1 A  
## 2 2 2 Gregory 29 1 A  
## 3 3 3 Bernard 31 1 A  
## 6 6 6 Rico 38 1 A  
## 8 8 8 Andrew 33 1 A  
## 12 12 12 William 35 1 B  
## 16 16 16 <NA> NA 1 B  
## 4 4 4 Lesla 21 2 A  
## 5 5 5 Bridget 34 2 A  
## 7 7 7 Temwa 28 2 A  
## 9 9 9 Cecil 25 2 A  
## 10 10 10 Martha NA 2 A  
## 11 11 11 Merriam 30 2 B  
## 13 13 13 Martha 33 2 B  
## 14 14 14 Mada 30 2 B  
## 15 15 15 Sara 29 2 B

mydata[order(mydata$sex, mydata$age),] #Sort data by sex and age

## id id.1 name age sex dept  
## 1 1 1 Patrick 23 1 A  
## 2 2 2 Gregory 29 1 A  
## 3 3 3 Bernard 31 1 A  
## 8 8 8 Andrew 33 1 A  
## 12 12 12 William 35 1 B  
## 6 6 6 Rico 38 1 A  
## 16 16 16 <NA> NA 1 B  
## 4 4 4 Lesla 21 2 A  
## 9 9 9 Cecil 25 2 A  
## 7 7 7 Temwa 28 2 A  
## 15 15 15 Sara 29 2 B  
## 11 11 11 Merriam 30 2 B  
## 14 14 14 Mada 30 2 B  
## 13 13 13 Martha 33 2 B  
## 5 5 5 Bridget 34 2 A  
## 10 10 10 Martha NA 2 A

#Section: More useful functions #Section: More about graphs #Section: User-defined functions

# Tidyverse

The tidyverse package actually contains other packages (dplyr, ggplot2, etc.) and you’ll see that when you load the tidyverse package using library(). Remember the package must be installed to your device before it can be loaded into your libraries! For help on installing packages, refer to Section

library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.2 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.3 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

# pipes

The pipe operator, (%>%), feeds the results of one operation into the next operation. It is more handy when there is a sequence of operations on a data frame. The advantage of using the pipe operator is that it makes code extremely easy to read.

# mutate, group\_by, summarize, filter, select, arrange

set.seed(356863)  
region <- sample(1:3,1000, prob = c(0.2,0.5,0.3), replace = T)  
distr <- c(sample(1:7,200,replace = T),sample(1:10,504,replace = T),sample(1:15,296,replace = T))  
resid <- c(rep(c(1,2),c(39,161)),rep(c(1,2),c(203,301)),rep(c(1,2),c(75,221)))  
bdywgt <- rnorm(1000,33,9)  
  
#Tibble (a data frame version)  
sample.data <- tibble(region,distr, resid, bdywgt)  
rm(region,distr,resid,bdywgt)  
  
sample.data <- sample.data %>%   
 mutate(dist\_code = region\*100+distr)  
  
#Summarize  
sample.data %>%   
 group\_by(region) %>%   
 summarise(meanwgt = mean(bdywgt),   
 freqn = NROW(bdywgt))

## # A tibble: 3 × 3  
## region meanwgt freqn  
## <int> <dbl> <int>  
## 1 1 32.3 177  
## 2 2 32.3 519  
## 3 3 33.3 304

seed <- read.csv("seed.csv")  
str(seed)

## 'data.frame': 64 obs. of 4 variables:  
## $ Blocks : int 1 2 3 4 1 2 3 4 1 2 ...  
## $ cultivar: chr "vicland1" "vicland1" "vicland1" "vicland1" ...  
## $ seedchem: chr "control" "control" "control" "control" ...  
## $ response: num 42.9 41.6 28.9 30.8 53.3 69.6 45.4 35.1 62.3 58.3 ...

seed[sample(1:nrow(seed),10),]

## Blocks cultivar seedchem response  
## 60 4 clinton Agrox 51.8  
## 42 2 clinton panoge 46.1  
## 23 3 vicland2 ceresan 42.4  
## 29 1 Branch ceresan 70.3  
## 16 4 Branch control 52.7  
## 1 1 vicland1 control 42.9  
## 25 1 clinton ceresan 63.4  
## 8 4 vicland2 control 35.1  
## 32 4 Branch ceresan 58.5  
## 62 2 Branch Agrox 69.4

table(seed$Blocks)

##   
## 1 2 3 4   
## 16 16 16 16

table(seed$cultivar)

##   
## Branch clinton vicland1 vicland2   
## 16 16 16 16

table(seed$seedchem)

##   
## Agrox ceresan control panoge   
## 16 16 16 16

mean(seed$response)

## [1] 52.75312

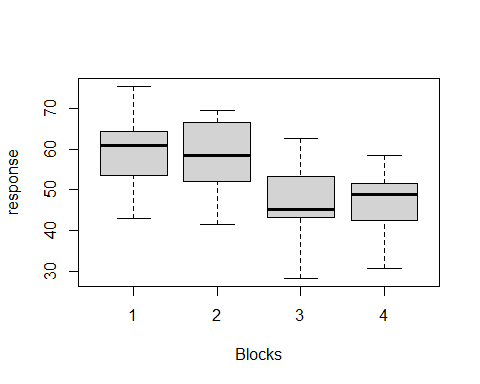
aggregate(response~Blocks,data=seed, mean)

## Blocks response  
## 1 1 59.80625  
## 2 2 58.53750  
## 3 3 46.17500  
## 4 4 46.49375

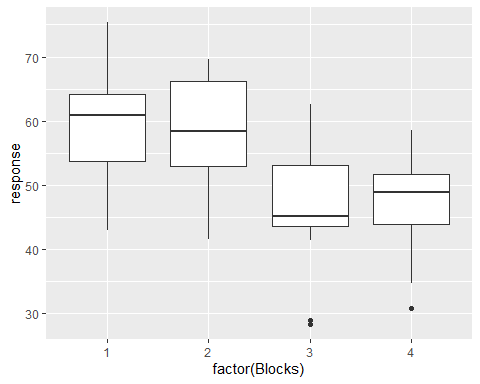
my.aov <- aov(response~Blocks,data=seed)  
summary(my.aov)

## Df Sum Sq Mean Sq F value Pr(>F)   
## Blocks 1 2188 2188.2 25.56 4.07e-06 \*\*\*  
## Residuals 62 5309 85.6   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

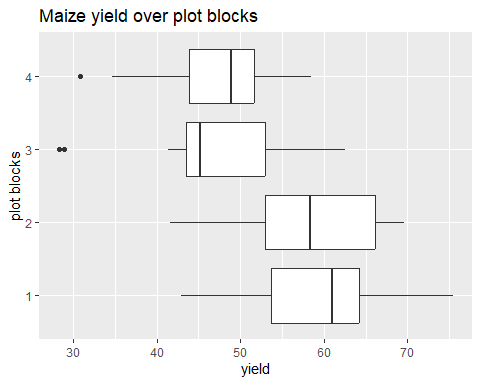
#Boxplot  
boxplot(response~Blocks, data=seed)



#ggplot version  
my.blocks.resp <- ggplot(seed, aes(x = factor(Blocks), y = response))   
my.blocks.resp + geom\_boxplot()



my.blocks.resp + geom\_boxplot() + coord\_flip() + labs(x = "plot blocks", y = "yield", title = "Maize yield over plot blocks")

 # ggplot2

set.seed(356863)  
region <- sample(1:3,1000, prob = c(0.2,0.5,0.3), replace = T)  
distr <- c(sample(1:7,200,replace = T),sample(1:10,504,replace = T),sample(1:15,296,replace = T))  
resid <- c(rep(c(1,2),c(39,161)),rep(c(1,2),c(203,301)),rep(c(1,2),c(75,221)))  
bdywgt <- rnorm(1000,33,9)  
  
#Tibble (a data frame version)  
sample.data <- tibble(region,distr, resid, bdywgt)  
rm(region,distr,resid,bdywgt)  
  
sample.data <- sample.data %>%   
 mutate(dist\_code = region\*100+distr)  
  
#Summarize  
sample.data %>%   
 group\_by(region) %>%   
 summarise(meanwgt = mean(bdywgt),   
 freqn = NROW(bdywgt))

## # A tibble: 3 × 3  
## region meanwgt freqn  
## <int> <dbl> <int>  
## 1 1 32.3 177  
## 2 2 32.3 519  
## 3 3 33.3 304

seed <- read.csv("seed.csv")  
str(seed)

## 'data.frame': 64 obs. of 4 variables:  
## $ Blocks : int 1 2 3 4 1 2 3 4 1 2 ...  
## $ cultivar: chr "vicland1" "vicland1" "vicland1" "vicland1" ...  
## $ seedchem: chr "control" "control" "control" "control" ...  
## $ response: num 42.9 41.6 28.9 30.8 53.3 69.6 45.4 35.1 62.3 58.3 ...

seed[sample(1:nrow(seed),10),]

## Blocks cultivar seedchem response  
## 60 4 clinton Agrox 51.8  
## 42 2 clinton panoge 46.1  
## 23 3 vicland2 ceresan 42.4  
## 29 1 Branch ceresan 70.3  
## 16 4 Branch control 52.7  
## 1 1 vicland1 control 42.9  
## 25 1 clinton ceresan 63.4  
## 8 4 vicland2 control 35.1  
## 32 4 Branch ceresan 58.5  
## 62 2 Branch Agrox 69.4

table(seed$Blocks)

##   
## 1 2 3 4   
## 16 16 16 16

table(seed$cultivar)

##   
## Branch clinton vicland1 vicland2   
## 16 16 16 16

table(seed$seedchem)

##   
## Agrox ceresan control panoge   
## 16 16 16 16

mean(seed$response)

## [1] 52.75312

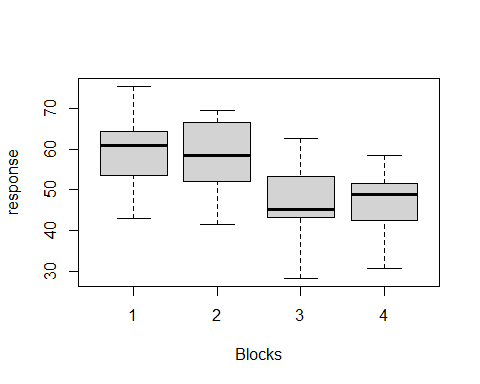
aggregate(response~Blocks,data=seed, mean)

## Blocks response  
## 1 1 59.80625  
## 2 2 58.53750  
## 3 3 46.17500  
## 4 4 46.49375

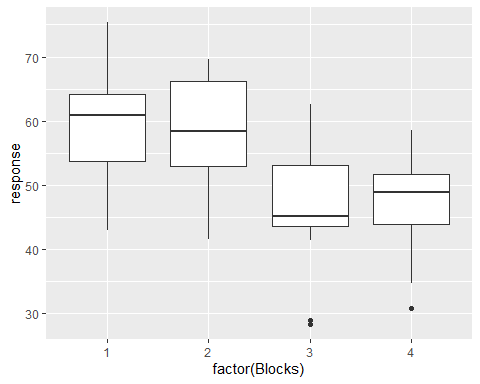
my.aov <- aov(response~Blocks,data=seed)  
summary(my.aov)

## Df Sum Sq Mean Sq F value Pr(>F)   
## Blocks 1 2188 2188.2 25.56 4.07e-06 \*\*\*  
## Residuals 62 5309 85.6   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

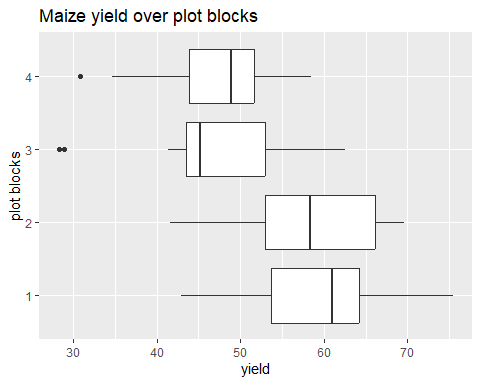
#Graphs  
boxplot(response~Blocks, data=seed)



my.blocks.resp <- ggplot(seed, aes(x = factor(Blocks), y = response))   
my.blocks.resp + geom\_boxplot()



my.blocks.resp + geom\_boxplot() + coord\_flip() + labs(x = "plot blocks", y = "yield", title = "Maize yield over plot blocks")



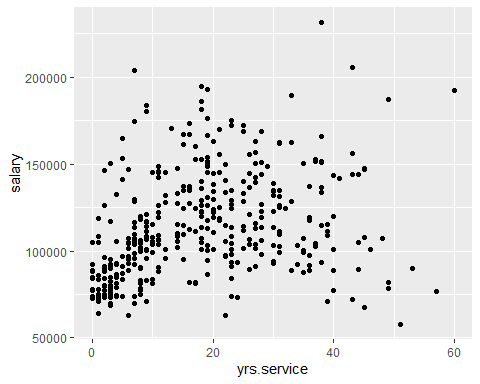
With ggplot2, data and aesthetic mappings are supplied in ggplot(), then layers are added on with +. This is an important pattern, and as you learn more about ggplot2 you’ll construct increasingly sophisticated plots by adding on more types of components.

Almost every plot maps a variable to x and y, so naming these aesthetics is tedious, so the first two unnamed arguments to aes() will be mapped to x and y. This means that the following code is identical to the example above:

#Linear regression

salaries <- read.csv("WorkSalaries.csv")  
View(salaries)

mylm.plot <- ggplot(salaries, aes(yrs.service, salary))  
mylm.plot + geom\_point()



my.lm <- lm(salary~yrs.service, data = salaries)  
summary(my.lm)

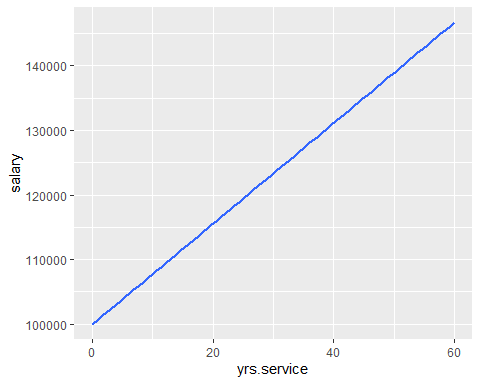
##   
## Call:  
## lm(formula = salary ~ yrs.service, data = salaries)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -81933 -20511 -3776 16417 101947   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 99974.7 2416.6 41.37 < 2e-16 \*\*\*  
## yrs.service 779.6 110.4 7.06 7.53e-12 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 28580 on 395 degrees of freedom  
## Multiple R-squared: 0.1121, Adjusted R-squared: 0.1098   
## F-statistic: 49.85 on 1 and 395 DF, p-value: 7.529e-12

ls(my.lm)

## [1] "assign" "call" "coefficients" "df.residual"   
## [5] "effects" "fitted.values" "model" "qr"   
## [9] "rank" "residuals" "terms" "xlevels"

mylm.plot +  
 geom\_smooth(method = lm, se = F, fullrange = T)

## `geom\_smooth()` using formula = 'y ~ x'



#APPLICATION

#Candidates, Diamonds Dataset #############################################################################

#setwd("C:/Users/Lenovo/Mirror/MDHS/2015-16")  
# library(haven)  
#   
 candidates <- read.csv("candidates.csv") #Candidates data  
# hhr.data <- read\_dta("MWHR7ADT/MWHR7AFL.DTA") #Household records  
# hhm.data <- read\_dta("MWIR7ADT/MWIR7AFL.DTA") #Individual records  
#   
# nrow(wra.data) #Number of rows/observations  
# dim(wra.data) #Dimension of the dataset  
# View(wra.data)  
#   
#   
# #attributes(wra.data)  
# selectedVar <- c("hhid", "hv001", "hv002", "hv005","hv009", "hv012","hv013","hv021","hv022","hv023","hv024","hv025")  
# hhrMerge <- hhr.data[,selectedVar]  
#   
# selectedVar <- c("caseid", "v001", "v002", "v003", "v005","v009","v010","v011", "v012","v013","v021","v022","v023","v024","v025")  
# hhmMerge <- hhm.data[,selectedVar]  
#   
#   
#   
# wra.data$anemia <- factor(wra.data$anemia, levels = c(0,1), labels = c("not anemic","anemic"))  
#   
# table(wra.data$anemia) #Tabulation of frequencies  
# prop.table(table(wra.data$anemia)) #Proportions  
# barplot(prop.table(table(wra.data$anemia)))  
#   
# wra.data$hv001 <- wra.data$mcluster  
# wra.data$hv002 <- wra.data$mnumber  
#   
# dim(merge(wra.data,hhrMerge, by=c("hv001", "hv002")))  
#   
#   
# wra.data <- merge(wra.data,hhrMerge, by=c("hv001", "hv002"))  
#   
#   
# wra.data$v001 <- wra.data$mcluster  
# wra.data$v002 <- wra.data$mnumber  
# wra.data$v003 <- wra.data$m01  
#   
# dim(merge(wra.data,hhmMerge, by=c("v001", "v002", "v003")))  
# wra.data <- merge(wra.data,hhmMerge, by=c("v001", "v002", "v003"))  
#   
# #Proportions  
#   
# prop.table(table(wra.data$anemia, wra.data$mtype),2) #Residence  
# prop.table(table(wra.data$anemia, wra.data$mregion),2) #Region  
# prop.table(table(wra.data$anemia, wra.data$m08),2)\*100 #Marital status  
#   
#   
#   
# #Test of associations  
#   
# chisq.test(table(wra.data$anemia, wra.data$mtype))  
# chisq.test(table(wra.data$anemia, wra.data$mregion))  
# chisq.test(table(wra.data$anemia, wra.data$m08))  
#   
# sum(table(wra.data$anemia)) #Total women  
#   
#   
# #Logistic regression  
# options(scipen = 999)  
# anemia.model <- glm(anemia ~ mregion + m08, family = "binomial", data = wra.data)  
# summary(anemia.model)  
#   
#   
# wave1 <- read\_dta("C:/Users/Lenovo/Documents/Wave 1/Wave 1/sect5\_Access\_r1.dta")  
# #str(wave1)  
#   
# table(wave1$s5q1a1)  
# prop.table(table(wave1$s5q1a1))\*100  
# sum(table(wave1$s5q1a1))  
#   
#   
#IHS Data

# Survival Tables

The cohort component projection method projects the population into the future by age (usually 5-year age groups) and sex. Survival rates are used to calculate the number of people that will be alive at a future date in time.

In many countries, life tables are based on an average of age-specific death rates for a 3-year time period, generally around a census taking. In many cases, the life tables are prepared every 10 years. For example, a country or state would collect age-specific death rates for 1999, 2000, and 2001. The census for year 2000 would be used for the base population.

# Projections

This chapter describes a variant of the cohort component method which can be used to make a projection either of the national population or of urban and rural populations. The method is capable of projecting the structure of the population by age and sex along with various indicators of population size, structure and change.

# Mathematical

The mathematical method is quick, simple, and requires little in the way of data. It is the approach of choice for many projections of the whole populations of countries.

## Cohort

The component method is much more cumbersome than the mathematical method, and has heavy data requirements. It is more time-consuming than the mathematical method, although the advent of computers has made it a great deal quicker than it used to be. It has the great advantage over the mathematical method that detailed aspects of the population structure can be forecast

The major strength of this technique is its ability to project a population in a straightforward and unambiguous manner. The technique does not embody restrictive or arbitrary assumptions and generates results which faithfully reflect the initial population structure and the fertility, mortality and migration conditions specified by the user. It yields projection results which are indispensable to any planning exercise seeking to take the future population change into account. These features make this technique fundamental for integrating population factors into development planning.

age\_int <- c(0,1,seq(5,95,5))  
nqx <- c(0.02592,0.0042,0.00232,0.00201,0.00443,0.00611,0.00632,0.00654,0.01098,0.01765,0.02765,0.04387,0.05987,0.09654,0.13654,0.18765,0.25439,0.37887,0.47898,0.57908,1)  
lx <- c(100000)  
  
for (i in 2:length(nqx))  
{  
 lx[i] <- round(lx[i-1] - lx[i-1]\*nqx[i-1])   
}  
  
ndx <- round(nqx \* lx)

Another example

x <- c(0,1,seq(5,75,5))  
n <- c(1,4,rep(5,(length(x)-2)))  
nMx <- c(0.1072,0.0034,0.0010,0.0007,0.0017,0.0030,0.0036,0.0054,0.0054,0.0146,0.0128,0.0269,0.0170,0.0433,0.0371,0.0785,0.0931)  
nkx <- c(0.33,1.56,rep(2.5,length(nMx)-2))  
  
nqx <- round((n\*nMx)/(1 + (n - nkx)\*nMx),4)  
  
lx <- c(100000)  
for (i in 2:length(nqx))  
{  
 lx[i] <- round(lx[i-1] - lx[i-1]\*nqx[i-1])   
 #Lx[i] <- (lx[i-1] + lx[i])\*2.5  
}  
  
ndx <- round(nqx\*lx)  
nLx <- n \* lx - ndx\*(n - nkx)  
  
Tx <- NA  
  
for (i in 1:length(nqx))  
{  
 Tx[i] <- sum(nLx[i:length(nqx)])  
 #Lx[i] <- (lx[i-1] + lx[i])\*2.5  
}  
  
ex <- Tx/lx  
as.data.frame(cbind(x,n,nMx,nkx, nqx,lx,ndx,nLx,Tx,ex))

## x n nMx nkx nqx lx ndx nLx Tx ex  
## 1 0 1 0.1072 0.33 0.1000 100000 10000 93300.0 5536915.4 55.369154  
## 2 1 4 0.0034 1.56 0.0135 90000 1215 357035.4 5443615.4 60.484616  
## 3 5 5 0.0010 2.50 0.0050 88785 444 442815.0 5086580.0 57.290984  
## 4 10 5 0.0007 2.50 0.0035 88341 309 440932.5 4643765.0 52.566362  
## 5 15 5 0.0017 2.50 0.0085 88032 748 438290.0 4202832.5 47.742099  
## 6 20 5 0.0030 2.50 0.0149 87284 1301 433167.5 3764542.5 43.129812  
## 7 25 5 0.0036 2.50 0.0178 85983 1530 426090.0 3331375.0 38.744577  
## 8 30 5 0.0054 2.50 0.0266 84453 2246 416650.0 2905285.0 34.401205  
## 9 35 5 0.0054 2.50 0.0266 82207 2187 405567.5 2488635.0 30.272787  
## 10 40 5 0.0146 2.50 0.0704 80020 5633 386017.5 2083067.5 26.031836  
## 11 45 5 0.0128 2.50 0.0620 74387 4612 360405.0 1697050.0 22.813798  
## 12 50 5 0.0269 2.50 0.1260 69775 8792 326895.0 1336645.0 19.156503  
## 13 55 5 0.0170 2.50 0.0815 60983 4970 292490.0 1009750.0 16.557893  
## 14 60 5 0.0433 2.50 0.1954 56013 10945 252702.5 717260.0 12.805242  
## 15 65 5 0.0371 2.50 0.1698 45068 7653 206207.5 464557.5 10.307924  
## 16 70 5 0.0785 2.50 0.3281 37415 12276 156385.0 258350.0 6.904985  
## 17 75 5 0.0931 2.50 0.3776 25139 9492 101965.0 101965.0 4.056048

#SIMILATIONS