A New PPB Toolkit

Sergio Castro and Matteo Petitti

2021-06-04

Contents

Chapter 1

Foreword

This is a very important foreword. .

Chapter 2

Introduction

#A beautiful introduction goes here.

Chapter 3

First steps in R

If you already know the basics of R, you can skip this section!

3.1 Why R?

- Open Source.
  Constantly uptaed by users around the world.
- Polifunctional.
  Data Management, Statistica Analysis and Data Visualization.
- Reproducibile.
  You can share your methods and results.

3.2 Let's start

Rstudio is like a calculator (but very powerful) Try to write on the console, and then click the Enter button.:

```
1+1
```

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

Try to do the same on the text editor (and then type Ctrl+Enter)

```
1+1
```

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

3.3 Other Arithmetic functions.

6x9

```
6*9
```

```
## [1] 54
```

3/2

```
3/2
```

```
## [1] 1.5
```

10+20/(2)

```
(10+20)/2
```

```
## [1] 15
```

Square root of 2

```
sqrt(2)
```

```
## [1] 1.414214
```

10 power of 2

```
10^2
```

```
## [1] 100
```

Logarithm of 10

```
log(10)
```

```
## [1] 2.302585
```

3.4 Functions

Functions are the base of, they are always written followed by a parenthesis. What is between parenthesis is called "arguments".

Three examples:

```
Mean = mean()
Standard deviation =  sd()
Concatenate =  c()
```

How to use them?

```
#Put together four elements.
c(1,2,3,4)
```

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

```
#Estimate the mean of these four elements.
mean(c(1,2,3,4))
```

```
## [1] 2.5
```

```
#Estimate the Standard Deviation of these four elements.
sd (c(1,2,3,4))
```

```
## [1] 1.290994
```

3.5 Objects

Mostly everything in R are "Objects". They can be numbers, characters, databases and other things.

You can see your objects in the upper right window of RStudio, called environment.

You can create objects by using ' = ' o ' <- '

Example:

```
#Create an object called x
x = c(1,2,3)
x
```

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

```
#Creae and object called myvector, with all numbers from 1 to 5
myvector= c(1:5)
myvector
```

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

Important!

- To actually see the object you created, you have to "call it" by writing it's name on the screen and typing enter.
- Attention to ortography!! MyVector is not myvector.

3.6 Saving your script.

You can save your script to work on it later.

It's very useful to create a folder in your computer for every project you do in R.

We invite you to create a folder on your documents and call it "Learning R"

Once created, you can save your script by clicking.

File > Save As > Choose your folder and name it "myscript_yourname.R".

(It's important to finish with ".R")

3.7 Setting up your Working Directory.

The Working Directory is the folder in which you are working.

1. You can know which folder is that by typing getwd()

```
getwd()
```

```
## [1] "C:/Users/Usuario1/Dropbox/RSR/PPB Toolkit"
```

*This is my working directory =)

2. You can fix your Working Directory by writing setwd("Folder Location")

```
setwd("C:/Users/Usuario1/Dropbox/RSR/PPB Toolkit")
```

*This is tricky and works differently in PC and Mac.

3. You can also fix your working Directory by clicking.

Session > Set Working Directory > Choose Directory > Manually locate your folder.

*After this, you will see on the console that a code was automatically written, using the setwd(function)

3.8 Preparing your data

To be done.

3.9 Uploading the data.

Once your data file is on csv format and in the folder you choose for Working Directory, you can upload it this way.

```
mydata <- read.csv("tomatodata.csv")
```

*Here, you are at the same time reading the data, and assigning it to an object called "mydata"

Another way to do this would be be clicking

Import Database > From text(base) > Manually choose and select your file.

3.10 Check your data

How many columns do I have?

```
ncol(mydata)
```

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

How many rows?

```
nrow(mydata)
```

```
## [1] 112
```

What are the names of my variables?

```
names(mydata)
```

```
## [1] "location"          "genotype"          "repetition"
## [4] "farmers_eval"      "plant_number"      "total_harvest"
## [7] "perc_mark_fruits"  "mean_fruit_weight"
```

What is the structure of my data?

```
str(mydata)
```

```
## 'data.frame':    112 obs. of  8 variables:
##  $ location         : chr  "Molise" "Molise" "Molise" "Molise" ...
##  $ genotype         : chr  "Molise Random" "Sestola Random" "Local Molise" "Modern
##  $ repetition       : int  1 1 2 2 1 1 2 2 1 1 ...
##  $ farmers_eval     : num  3.24 2.71 3.35 3.24 3.24 ...
##  $ plant_number     : int  20 20 20 20 20 20 20 20 18 20 ...
##  $ total_harvest    : num  15.79 9.96 16.44 16.93 21.66 ...
##  $ perc_mark_fruits : num  57.5 55.8 76 55.3 81.2 ...
##  $ mean_fruit_weight: num  0.01743 0.00737 0.03554 0.01419 0.06885 ...
```

*This function is particularly useful, as you can see the data type of your columns. They might be characters, numerics, integrers (numbers without decimals) and factors, among other,

See your data in spreadsheet format.

```
View(mydata)
```

See only the first 10 rows

```
head(mydata)
```

See only the last 10 rows

```
tail(mydata)
```

Useful tips about your data:

- Always check your data when uploading.
- Especifically, check the data type of each variable, as this might lead to problems.
- Choose simple names for your columns and avoid using spaces.

3.11 Accesing your data

3.11.1 Using $ per le collonne (ie. mydata$columnname)

```
mydata$total_harvest
```

```
##   [1] 15.78500  9.95500 16.43800 16.93000 21.66457 15.00973 14.38000 23.26000
##   [9] 15.38500 15.32500 15.82500 20.46357 20.08227 17.23000 15.78500 19.07035
##  [17] 18.12500 16.62500 13.97518 19.54500 13.42000 19.99500 21.11300 12.41000
##  [25] 16.29770 20.32452 24.46040 15.70000 78.15500 73.58000 49.76000 59.19600
##  [33] 47.02000 63.98500 70.20500 72.34000 68.54500 67.45500 72.39600 68.95500
##  [41] 48.06500 53.21500 61.98000 31.75000 64.30000 53.43500 55.78000 52.92000
##  [49] 54.30600 26.43500 68.98000 63.89000 66.30500 43.70000 55.08500 57.54000
##  [57]  8.06500  9.64500  8.92000 11.21500  7.48500 10.19500  8.13000 10.31200
##  [65]  5.67000  3.46000 19.70500 12.78500  4.82500 18.89500  3.61500  6.39500
##  [73]  8.37500  8.82600  6.56500  9.09500 14.23500  4.43500  3.16500 14.38500
##  [81]  5.50000  8.37000  8.81000  4.96500 16.44300 13.77100 14.43500 22.26200
##  [89] 14.29300 16.35600 15.57800 14.95300 19.15600 13.97200 15.29900 22.71800
##  [97] 11.14300  9.73900 13.84300 17.43500 21.79200 13.15200 15.63700 17.03200
## [105] 26.90100 14.46900  8.79000 15.96100 23.88700  8.10700 11.61100  9.64700
```

3.11.2 Using the form data[row,column]

For example, try to see only the third row, third column.

```
mydata[3,3]
```

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

See the first row, all columns.

```
mydata[1,]
```

```
##   location      genotype repetition farmers_eval plant_number total_harvest
## 1   Molise Molise Random          1     3.235294           20        15.785
##   perc_mark_fruits mean_fruit_weight
## 1         57.45961        0.01742664
```

See rows 1, 2 and 3, and only the first column.

```
mydata[c(1,2,3),1]
```

See 2 first rows, and only columns 4 to 8

```
mydata[c(1,2),c(4:8)]
```

```
##   farmers_eval plant_number total_harvest perc_mark_fruits mean_fruit_weight
## 1     3.235294           20        15.785         57.45961        0.01742664
## 2     2.705882           20         9.955         55.75088        0.00736670
```

3.12 Some basic statistics

Estimate the mean of a variable.

```
mean(mydata$total_harvest)
```

```
## [1] 25.16323
```

Estimate the median.

```
median(mydata$total_harvest)
```

```
## [1] 16.12935
```

Estimate the standard deviation.

```
sd(mydata$total_harvest)
```

```
## [1] 21.03161
```

3.13 Vector arithmetics

In the same way in which you can do arithmetics with simple numbers, you can do it with vectors.

```
# Define a v vector
v <- c(11, 12, 13, 14, 15)
# Define a w vector
w <- c(1, 2, 3, 4, 5)
# Add them up to create a t vector
t= v + w

# t is the sum of v and w
t
```

```
## [1] 12 14 16 18 20
```

3.14 Create a new column.

In this particular data set, we have many plots with missing plants. There are many ways to adress this, but one simple way is to divide the plot harvest by the number of plants, so that we have yield per plant, as a measure of the genotype's productivity

```
newvector <- mydata$total_harvest /
                      mydata$plant_number

#We see this new vector has yield per plant
newvector[1:10]
```

```
##  [1] 0.7892500 0.4977500 0.8219000 0.8465000 1.0832286 0.7504866 0.7190000
##  [8] 1.1630000 0.8547222 0.7662500
```

Once you created the vector, you can add it to the data frame. You can also do it directly, but we chose it this way to make it simpler

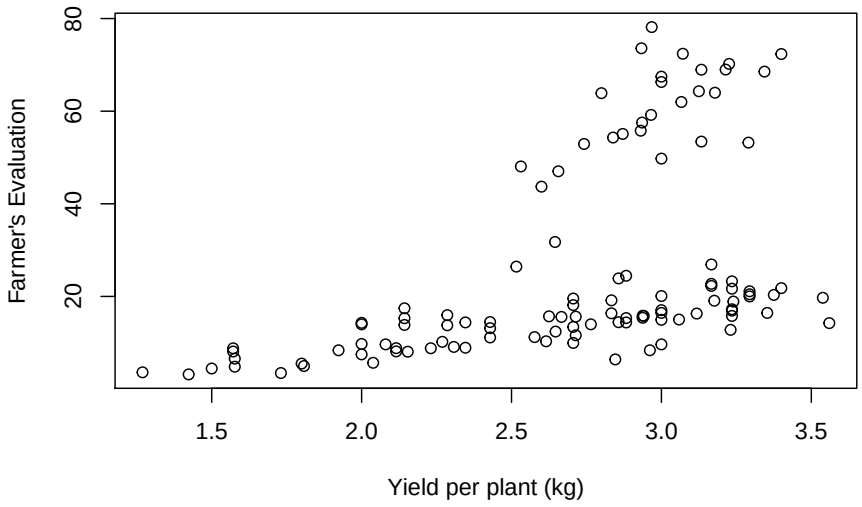
```
mydata$harvest_perplant <- newvector
```

*Notice that the column "harvest_perplant" was named and created at the same moment.

3.15 A simple plot

Using the function plot() we can create simple exploratory graphs.

```
plot( x= mydata$farmers_eval, #My x axis
      y= mydata$total_harvest, # My y axis
      xlab= "Yield per plant (kg)",  # My x label
      ylab= "Farmer's Evaluation" ) # My y label
```



3.16 Another simple plot

We could, for example plot the harvest data for each location. But first, we will check if the data types are right.

```
str(mydata)
```

```
## 'data.frame':    112 obs. of  9 variables:
##  $ location         : chr  "Molise" "Molise" "Molise" "Molise" ...
##  $ genotype         : chr  "Molise Random" "Sestola Random" "Local Molise" "Modern
##  $ repetition       : int  1 1 2 2 1 1 2 2 1 1 ...
##  $ farmers_eval     : num  3.24 2.71 3.35 3.24 3.24 ...
##  $ plant_number     : int  20 20 20 20 20 20 20 20 18 20 ...
```

```
##  $ total_harvest    : num  15.79 9.96 16.44 16.93 21.66 ...
##  $ perc_mark_fruits : num  57.5 55.8 76 55.3 81.2 ...
##  $ mean_fruit_weight: num  0.01743 0.00737 0.03554 0.01419 0.06885 ...
##  $ harvest_perplant : num  0.789 0.498 0.822 0.847 1.083 ...
```

Looks like the location is a character variable, meaning it is just a loose string of text. We rather want it to be a factor, so that all observations with the same location can be grouped in plots.

```
mydata$location = as.factor(mydata$location)
class(mydata$location)
```

```
## [1] "factor"
```

And now we can do the plot.

```
plot( x= mydata$location,
      y=mydata$total_harvest,
      ylab= "Total Harvestper plot(kg)",
      xlab= "Location")
```

3.17 Subsetting your data.

Suppose you only want to work with a section of your data frame, for example, only the data for one location.

There are many ways to do this, a rather simple one is using the function subset()

```
soloRotonda <- subset(mydata, mydata$location == "Rotonda")
nrow(soloRotonda)
```

```
## [1] 28
```

Notice how we created a new subset, only with data from Rotonda. After checking, we see it has only 28 rows, as supposed.

Notice also how we used the "==" sign, to express a logical function stating the location had to be equal to "Rotonda"

In R, the other characters to indicate logical expressions are:
* '==' for equal
* '!=' for different

* '<,>' less than, more than
* '<=' less or equal to
* '>=' more or equal to
* '&' if we want one condition AND another. * "|" if we Want one condition OR another.

For example, we can select only the data for Rotonda, in which the evaluation is higher than 3.

```
soloRotondaSuperiori <- subset(mydata,
                  mydata$localita== "Rotonda"
                  &
                  mydata$valutassione_agricultori >= 3)
nrow(soloRotondaSuperiori)
```

```
## [1] 0
```

3.18 Packages in R.

Virtually anyone can create and upload a package in R.

Some of them are very useful to treat and analyze data. We will see many examples later.

Download and installing them is very easy, you have mostly two options./

1. The simplest is to click on Tools > Install Packages > and write the package name.
2. You could also write

```
install.packages("yourpackagename")
#Don't try this! It's only an example!
```

Once installed, you have to call it so that it is active on R. This is true for every time you open R.

```
library(yourpackagename)
#Again, don't try this.
```

For example, we will try to instally DPLYR, which gives us an easy and intuitive grammatic for data manipulation.

```
install.packages("dplyr")
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Some of the most useful functions in Dplyr R are

```
filter() # To filter rows according to criteria
select() # TO select only certain columns
rename() # To rename columns.
mutate() # TO create a new column from an old one.
```

For example, you can create a new dataframe if you select only location, genotype, repetition and mean fruit weight.

```
newdataframe <- select(mydata, location, genotype, repetition, mean_fruit_weight)
```

Within this new data frame, I can create a new column, which converts mean fruit weight (now in kg) to grams (g)

```
newdataframe2 <- mutate(newdataframe, meanfruitweight_grams = mean_fruit_weight * 1000)
```

3.19 The pipeline %>% operator

With the pipeline operator you can chain more than one operation together, instead of using temporary objects and changing them. It also allows to call my first object ('mydata') only once

```
newdataframe3 <- mydata %>%
                 select(location, genotype, repetition, mean_fruit_weight) %>%
                 mutate( meanfruitweight_grams = mean_fruit_weight * 1000)
```

3.20 Aggregate data

Finally, DPLYR helps us us to calculate, for example, the mean perfomance of each genotype per environment, given that there are two repetitions.

For this, you have to use the group_by() and summarize (). Under group_by, we put the aggrupating facors. Under summarize, we put the columns we want, in this case, we want the means and the standard deviations.

```
mydata_means_genoxenv <- mydata %>%
              group_by(genotype, location) %>%
              summarize(mean(farmers_eval),
                          sd(farmers_eval),
                          mean(harvest_perplant),
                         sd(harvest_perplant))
```

3.21 Save your data

Once you have created and modified and object, you can save it on the same folder in your .csv format.

```
#We can save the tomato data with our modifications.
write.csv(mydata, "tomatodata_modified.csv")

#And we can save the new data frame we created with the means.
write.csv(mydata_means_genoxenv, "tomatodata_means_genoxenv.csv")
```

3.21.1 Ask questions!

It's very hard to know (and remember!) how everything is done.

There will always be things that you want to do, you've never done befora and are out of this very small manual.

Essempi: 1. When wanting to understand a function, you can type ?functionname (and changing "functionname" for your actual function.) 2. A great approach is just to google: "How to _____ in R". This can get you out of many errors. 3. When you receive errors (red letters on the console) try to read them and see what they mean. Another good approach is to copy-paste them in google.

3.22 To learn more

This are just some sources to learn more about R:

The basic book is the R Cookbook

This is a cool blog to ask questions and learn more: Stack Overflow

Chapter 4

Methods

We describe our methods in this chapter.

Chapter 5

Applications

Some *significant* applications are demonstrated in this chapter.

5.1 Example one

5.2 Example two

Chapter 6

Final Words

We have finished a nice book.