An Awesome Book

Gaston Sanchez
gastonsanchez.com

About this ebook

Abstract

This book aims to show you different tools.

About the reader

I am assuming two things about you: 1) You acknowledge the importance of data exploration and visualization; 2) you have some knowledge of the statistical software R.

Citation

Sanchez, G. (2015) **An Awesome Book**
URL `http://gastonsanchez.com/about`

Source

Github Repository:
`https://github.com/gastonstat/ujat-dynadocs-workshop`

Revision

March 7, 2015
Version 1.0

Contents

Preface

I didn't enjoy my years as an undergraduate college student. Mostly because I was a total misfit. But as years have gone by, I've been able to identify a crucial missing link in my early formal education: *data visualization*. Yes, I know it sounds crazy but I'm absolutely positive that things would have been radically different had I only been exposed early on my data crunching career to the fascinating world of charts, plots, graphics, and diagrams.

During my undergrad years I took courses such as geometry, algebra, calculus, probability, and statistics, but I never had the chance to see a lot of graphics. There were many equations for sure. But almost no graphics. One of the rare charts were some population pyramids in my Demography course, and some basic charts during my "advanced" *Excel* course. The last time I checked the program offered by my college (not very long ago), I discoverd some changes. But not the changes I was expecting. There are more courses on economics, management, and finance. Sadly, there's still no dedicated data visualization material.

Acknowledgements

I want to thank my sensei Tomas Aluja for showing me the landscape of exploratory methods and for introducing me to the fascinating world of data analysis techniques for dimension reduction.

Gaston Sanchez
Berkeley, California
April 2015

Chapter 1

Introduction

First some definitions. We'll consider a data set to be integrated by a group of *individuals* or observations on which we observe or measure one or more characteristics called *variables*.

Individuals can be people, animals, objects, planets, plants, countries, etc. Terminology is abundant and you'll find synonym terms such as cases, objects, items?, samples?

A **variable** is any characteristic of an individual. Sometimes we refer to variables as features, aspects, indicators, or descriptors. The idea is that an individual is describes by one or more variables.

1.1 Categorical Variables

A **categorical** variable is a variable that takes values representing one or more categories. We also call them *qualitative* variables because they describe qualities of the observed individuals. This is in contrast with *quantitative* variables that takes numerical values representing quantities, and for which arithmetic operations such as adding or calculating differences make sense.

Typical examples of categorical variables are things like gender (female, male), hair color (e.g. black, brown, blod, red), icecream flavors (e.g. chocolate, vanilla, lemon, coffee), cloth sizes (e.g. small, medium, large),

or college years (e.g. freshman, sophomore, junior, senior). Notice that in all these cases we are using values that do not reflect quantification.

1.1.1 Levels of Measurement

Categorical variables can be divided in two main groups according to their scale. On one hand we have **nominal** variables, on the other hand we have **ordinal** variables.

Nominal variables refer to categories that have no natural order. The term *nominal* according the dictionary means "existing in name only". Thus, nominal values are just that: names. Icecream flavors is one example of a nominal variable. There is no reason why chocolate is better or greater than vanilla. You could say that you prefer chocolate over vanilla but that's a different variable: personal preference.

We can find categorical data under a wide range of formats. I've seen categorical data codified in different ways, and sometimes people are very creative in the way they store it.

The main types of formats can be classified in three main groups:

- text or characters
- numbers (ideally integers)
- logical (TRUE / FALSE), typically for binary variables

Here's an example with a "gender" variable:

- as text: `"F"` (female), `"M"` (male)
- as numbers: 1 (female), 0 (male)
- as logical: `TRUE` (female), `FALSE` (male)

Chapter 2

Categorical Data in R

I'm one of those with the humble opinion that great software for data science and analytics should have a data structure dedicated to handle categorical data. Lucky for us, R is one of the greatest. In case you're not aware, one of the nicest features about R is that it provides a data structure exclusively designed to handle categorical data: **factors**.

The term "factor" as used in R for handling categorical variables, comes from the terminology used in *Analysis of Variance*, commonly referred to as ANOVA. In this statistical method, a categorical variable is commonly referred to as *factor* and its categories are known as *levels*. Perhaps this is not the best terminology but it is the one R uses, which reflects its distinctive statistical origins. Especially for those users without a brackground in statistics, this is one of R's idiosyncracies that seems disconcerning at the beginning. But as long as you keep in mind that a factor is just the object that allows you to handle a qualitative variable you'll be fine. In case you need it, here's a short mantra to remember: *"factors have levels"*.

2.1 Creating Factors

To create a factor in R you use the homonym function `factor()`, which takes a vector as input. The vector can be either numeric, character or logical. Let's see our first example:

```
# numeric vector
num_vector <- c(1, 2, 3, 1, 2, 3, 2)

# creating a factor from num_vector
first_factor <- factor(num_vector)

first_factor
```

```
## [1] 1 2 3 1 2 3 2
## Levels: 1 2 3
```

As you can tell from the previous code snippet, `factor()` converts the numeric vector `num_vector` into a factor (i.e. a categorical variable) with 3 categories —the so called `levels`.

You can also obtain a factor from a string vector:

```
# string vector
str_vector <- c('a', 'b', 'c', 'b', 'c', 'a', 'c', 'b')

str_vector
```

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

```
# creating a factor from str_vector
second_factor <- factor(str_vector)

second_factor
```

```
## [1] a b c b c a c b
## Levels: a b c
```

Notice how `str_vector` and `second_factor` are displayed. Even though the elements are the same in both the vector and the factor, they are printed in different formats. The letters in the string vector are displayed with quotes, while the letters in the factor are printed without quotes.

And of course, you can use a logical vector to generate a factor as well:

```
# logical vector
log_vector <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

# creating a factor from log_vector
third_factor <- factor(log_vector)

third_factor
```

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

```
## Levels: FALSE TRUE
```

2.1.1 How R treats factors?

If you're curious and check the technical *R Language Definition*, available online https://cran.r-project.org/manuals.html, you'll find that R factors are referred to as *compound objects*. According to the manual: "Factors are currently implemented using an integer array to specify the actual levels and a second array of names that are mapped to the integers." Essentially, a factor is internally stored using two arrays: one is an integer array containing the values of categories, the other array is the "levels" which has the names of categories which are mapped to the integers.

Under the hood, the way R stores factors is as vectors of integer values. One way to confirm this is using the function `storage.mode()`

```
# storage of factor
storage.mode(first_factor)

## [1] "integer"
```

This means that we can manipulate factors just like we manipulate vectors. In addition, many functions for vectors can be applied to factors. For instance, we can use the function `length()` to get the number of elements in a factor:

```
# factors have length
length(first_factor)

## [1] 7
```

We can also use the square brackets `[ ]` to extract or select elements of a factor. Inside the brackets we specify vectors of indices such as numeric vectors, logical vectors, and sometimes even character vectors.

```
# first element
first_factor[1]

## [1] 1
## Levels: 1 2 3

# third element
first_factor[3]

## [1] 3
## Levels: 1 2 3
```

```
# second to fourth elements
first_factor[2:4]

## [1] 2 3 1
## Levels: 1 2 3

# last element
first_factor[length(first_factor)]

## [1] 2
## Levels: 1 2 3

# logical subsetting
first_factor[rep(c(TRUE, FALSE), length.out = 7)]

## [1] 1 3 2 2
## Levels: 1 2 3
```

If you have a factor with named elements, you can also specify the names of the elements within the brackets:

```
names(first_factor) <- letters[1:length(first_factor)]
first_factor

## a b c d e f g
## 1 2 3 1 2 3 2
## Levels: 1 2 3

first_factor[c('b', 'd', 'f')]

## b d f
## 2 1 3
## Levels: 1 2 3
```

However, you should know that factors are NOT really vectors. To see this you can check the behavior of the functions `is.factor()` and `is.vector()` on a factor:

```
# factors are not vectors
is.vector(first_factor)

## [1] FALSE

# factors are factors
is.factor(first_factor)

## [1] TRUE
```