R: learn by the exercise

Myriam Luce

September 10, 2018

Contents

Chapter 1

Descriptive statistics

1.1 Possibly interesting extra tidbits

In the making of this tutorial, I used several tools that you might like to access as well. Being the tedium-averse programmer I am, I use a reference manager program, in my case Zotero. You can find the full bibliography for this project, including a few entries that did not make it into the references section here because I did not cite them, at the Zotero project page.

I also want to point out awesome-public-datasets, where I foraged for the examples in this tutorial. It has several interesting datasets in the public health domain.

1.2 Tables and figures

1.2.1 Frequency table (1D) or contingency table (2D)

If you feel the need to make a table with your data, use a spreadsheet software (Microsoft Excel, LibreOffice Calc, Google Sheets). ;) R is superior in statistics and (arguably) in figures, but spreadsheets definitely have their uses when it comes to tables.

1.2.2 Pie chart

A pie chart is a graph that can be used to visually represent proportions of a *discrete variable*[1]. Note that they have their critics, who recommend never using them, as our brain is bad at comparing the size of slices [4].

As an example data set, let's use ebola deaths by country [3]. An excerpt giving the source data is shown in figure 1.1. Enter the data in your favorite spreadsheet software and save it as a csv (thankfully for you English speakers,

[1] Words in italics are defined in the glossary.

there is no need to fiddle with decimal symbol (is it a dot or a comma?) and whether the data is really comma-separated). You should get the following:

```
Country,Deaths
Guinea,2543
Liberia,4809
Sierra Leone,3956
Mali,6
Nigeria,8
United States of America,1
```

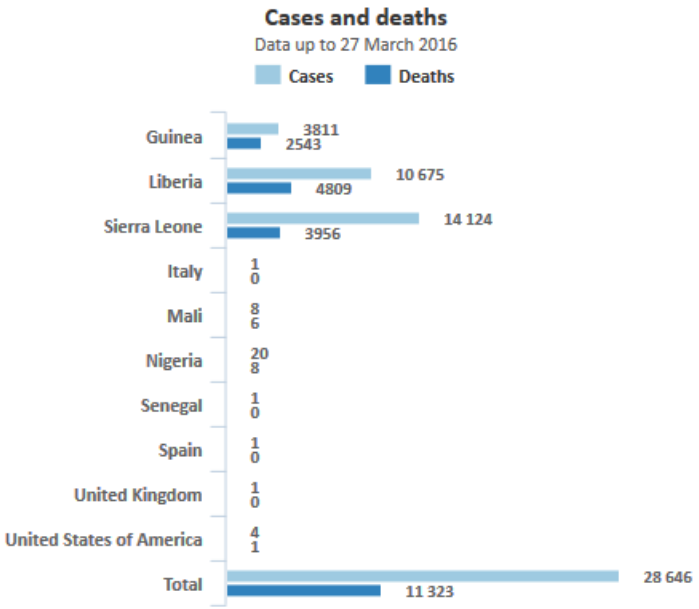

Figure 1.1: Excerpt from [3].

R offers various data import options. The most useful I have found were `read.csv`[2] to import csv data and `read.fwf` to import fixed-width data. To demonstrate, figure 1.2 shows what csv (delimited) and fixed-width data look like side by side.

Go ahead and load your small csv into R with `read.csv('C:/.....data.csv', header=TRUE)`. To avoid messing with default working folder in R settings, I recommend always using the full absolute file path (i.e. starting with C:). Note that you should use the *forward slash "/"* as a path separator, even on Windows. The second parameter, `header=TRUE`, tells R that the first line in your file

[2]Words in monospace font refer to R commands. The cheat sheet at the end of the tutorial lists most of those used in this document.

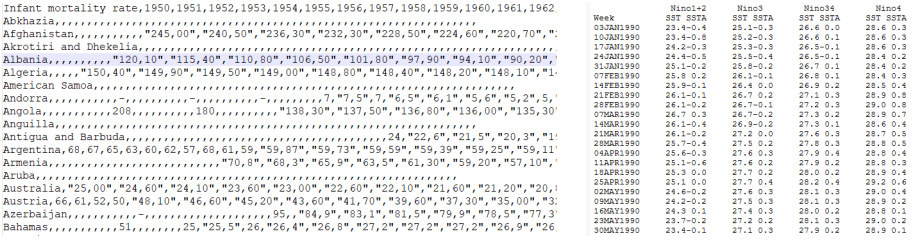

Figure 1.2: Delimited data (left) and fixed-width data (right).

corresponds to the column headers, not actual data. You can then use the function `pie(counts, labels)` to produce a pie chart. However, as shown below, a naive approach might displease.

```
> ebola = read.csv('D:/megha/Documents/r-tutorial/ebola.csv', header=TRUE)
> ebola
                    Country Deaths
1                    Guinea   2543
2                   Liberia   4809
3              Sierra Leone   3956
4                      Mali      6
5                   Nigeria      8
6 United States of America      1
> pie(ebola)
Error in pie(ebola) : 'x' values must be positive.
```

You might be scratching your head and wondering which part of 2543 or 6 is not positive, and you'd be justified to do so. Here, one must dive into computer programming concerns to understand what is going on. The "not positive" message hints at a problem with the format of the input data. Let's demonstrate:

```
> values = c(2543, 4809, 3956, 6, 8, 1
> labels = c('Guinea', 'Liberia', 'Sierra Leone', 'Mali',
↪  'Nigeria', 'United States of America')
> pie(values, labels)        # works! produces figure 1.3
> typeof(values)
[1] "double"
> typeof(ebola)
[1] "list"
> class(values)
[1] "numeric"
> class(ebola)
[1] "data.frame"
> class(ebola$Country)
```

```
[1] "factor"
> class(ebola$Deaths)
[1] "integer"
> pie(ebola$Deaths, ebola$Country)         # works too now!
```

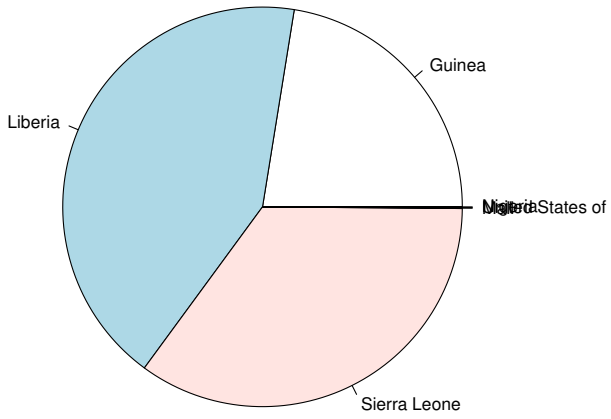

Figure 1.3: Ebola deaths in 2015-2016 by country.

Technically, `read.csv` returns a `data.frame`, while `pie` only accepts numbers. Let's take a painful tangent into R types that will hopefully help you later on.

R types

Logical:	TRUE or FALSE
Numeric:	real, by the math definition (ex. 12.3). Double is a numeric with better precision.
Integer:	integer, by the math definition (ex. 12).
Character:	text of any length
Factor:	a type that represents a discrete variable
Ordered:	a type that represents an ordinal variable
List:	a 1D collection of "things" (may be strings, numbers, or a mix of them)
Vector:	a 1D collection of things of *one type*
Matrix:	a 2D collection of things of *one type*
Array:	a nD collection of things of *one type*
Data Frame:	a (mostly) 2D collection of things, where each column can be of a different type

For future reference, Quick R gives an excellent introduction on the subject [5]. You can convert a variable to anything reasonable (R will turn "2" into an integer, but not "abc") using the host of `as.xyz` functions.

Accessing collection elements

Let's say you have a data frame, for example your ebola deaths by country data:

```
> ebola
                   Country Deaths
1                   Guinea   2543
2                  Liberia   4809
3             Sierra Leone   3956
4                     Mali      6
5                  Nigeria      8
6 United States of America      1
```

Notice how the line with "Country" and "Deaths" is not numbered in the output? It means R is aware it's a header and not data. Data frame columns can be accessed by their name using the `$` operator, like so:

```
> ebola$Country
[1] Guinea                    Liberia
↪  Sierra Leone
[4] Mali                      Nigeria
↪  United States of America
6 Levels: Guinea Liberia Mali Nigeria ... United States of
↪  America
```

If you want to access lines, you can use the `[row, column]` operator, like so:

```
> ebola[1,]
  Country Deaths
1  Guinea   2543
> ebola[1,2]
[1] 2543
```

While the `$` operator is exclusive to data frames, the `[]` is used for all collections. Vectors, lists, matrices and arrays can be accessed with the `[index]` operator for 1D structure, `[row, column]` operator for 2D structures, and `[i, j, k...]` for nD structures.
In the case you want to access several items at once, you can use a colon inside the brackets, i.e. `[begin:end]` like so:

```
> ebola[1:3,]
       Country Deaths
1       Guinea   2543
2      Liberia   4809
3 Sierra Leone   3956
```

Now that we have our basic pie chart, you might be thinking, "That squiggle on the right with the tiny pie slices is quite unseemly". In addition, you might want to tweak other aspects of the graph, like adding a title or choosing colors. We will discuss common graph properties in a following section, to keep it all in the same place. Let's just deal with the pie-chart specific problem of small slices here (I reiterate, you should run away, run away into the arms of a bar chart.), and add a percent annotation, as that is a common occurence. R does not offer an option to deal with small slices out of the box (probably because it tells you in its own manual to use bar charts instead), so let's just manually tweak the labels:

```
> labels = as.character(ebola$Country)
> labels[4]='Others'
> labels[5:6]=''
> labels
[1] "Guinea"       "Liberia"      "Sierra Leone" "Others"
↪  ""
[6] ""
> percents = ebola\$Deaths/sum(ebola\$Deaths)*100
> percents
[1] 22.458712355 42.471076570 34.937737349  0.052989490
↪  0.070652654
[6]  0.008831582
> percents[4] = sum(percents[4:6])
> percents
[1] 22.458712355 42.471076570 34.937737349  0.132473726
↪  0.070652654
[6]  0.008831582
> percents = round(percents, 2)
> percents
[1] 22.46 42.47 34.94  0.13  0.07  0.01
> labels[1:4] = paste(labels[1:4], percents[1:4], '\%')
> labels
[1] "Guinea 22.46 \%"       "Liberia 42.47 \%"      "Sierra Leone
↪  34.94 \%"
[4] "Others 0.13 \%"
> pie(ebola\$Deaths, labels)
```

Hacky, but it works, and no more time should be dedicated to pie charts, so let's move on.

1.2.3 Bar chart

A bar chart, sometimes called a line graph, is used to represent a discrete variable, and the bars *do not touch*. As an example, data on infant mortality by country can be found at Gapminder [2]. If you simply `read.csv` the file you just downloaded and converted to csv using your favorite Spreadsheet software,

and then examine the structure of your data inside R, you might notice the somewhat strange following occurence:

```
> infant = read.csv('D:/megha/Documents/r-tutorial/infant.csv',
↪  header=TRUE)
> class(infant)
[1] "data.frame"
> str(infant)
'data.frame':   260 obs. of  217 variables:
 $ Infant.mortality.rate: Factor w/ 260 levels
 ↪  "Abkhazia","Afghanistan",..: 1 2 3 5 6 7 8 9 10 11 ...
 $ X1800                : int  NA NA NA NA NA NA NA NA NA NA ...
 $ X1801                : int  NA NA NA NA NA NA NA NA NA NA ...
# ...
 $ X1861                : int  NA NA NA NA NA NA NA NA NA NA ...
 $ X1862                : Factor w/ 11 levels
 ↪  "","."","110","131",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ X1863                : Factor w/ 13 levels
 ↪  "","."","106","113",..: 1 1 1 1 1 1 1 1 1 1 ...
# ...
```

Wait, what? The country is of type factor, we all agree on that, but infant mortality rate for 1862 is a factor? Using good ol' Notepad or a better text editor, open your csv and scroll around, you should spot the problem illustrated in figure 1.4.

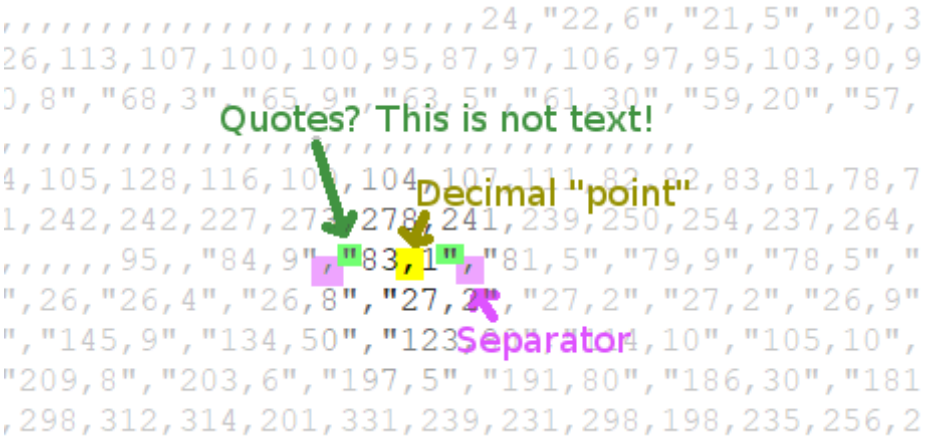

Figure 1.4: Welcome to the real world. CSV sucks internationally.

In other languages, like French for instance, the decimal "point" is a comma, and therefore the "comma-separated" part of comma-separated value leads to some issues. Other gems in this data include "-" for I assume missing data, and

some lonely dots, assuming again, for missing data. Gods forbid the authors hit a snag while importing data and their software didn't warn them something foul was afoot and they just pasted it in the global csv without noticing. (This is why science should be in databases. Real databases. They don't let you put a dash in a number field, they just don't.)

Thankfully, `read.csv` comes with some handy options. You can see them all in the manual by typing `?read.csv` in a R console. So, we need to tell R that our `dec` point is a comma, and dash and dot stand for missing data, i.e. `na.strings`. Technically, R also thinks that things surrounded by quotes are text, but is smart enough to automatically check if the "text" looks like a number and, if it does, it converts it automatically to a number. You therefore get the appropriate following:

```
> infant = read.csv('D:/megha/Documents/r-tutorial/infant.csv',
↪  header=TRUE, dec=",", na.strings=c('-', '.'))
> str(infant)
'data.frame':   260 obs. of  217 variables:
 $ Infant.mortality.rate: Factor w/ 260 levels
 ↪  "Abkhazia","Afghanistan",..: 1 2 3 5 6 7 8 9 10 11 ...
 $ X1800                : int  NA NA NA NA NA NA NA NA NA NA ...
 $ X1801                : int  NA NA NA NA NA NA NA NA NA NA ...
 # ...
```

A barplot is relatively straightforward to produce with R, but we will see all "common" (imho) plot options here, so tie your winter hat down with wire, you'll be sitting here a while. Let's start by simply plotting infant mortality rate by country. To keep the plot readable, let's choose a subset of G8 countries: Canada, France, Germany, Italy, Japan, Russia, United Kingdom and United States of America. Let's also start by studying the mortality rate in 2000. First, we will select each of the countries by its row number, then we will stitch the G8 back together with a function called `rbind`, which binds data frames together by row, as long as all data frames have the same columns.

```
> canada = infant[38,]
> france = infant[77,]
> germany = infant[83,]
> italy = infant[109,]
> japan = infant[111,]
> russia = infant[186,]
> uk = infant[240,]
> usa = infant[241,]
> g8 = rbind(canada, france, germany, italy, japan, russia, uk,
↪  usa)
```

Producing a barplot now is easy:

```
> barplot(g8$X2000, names.arg=g8$Infant.mortality.rate)
```

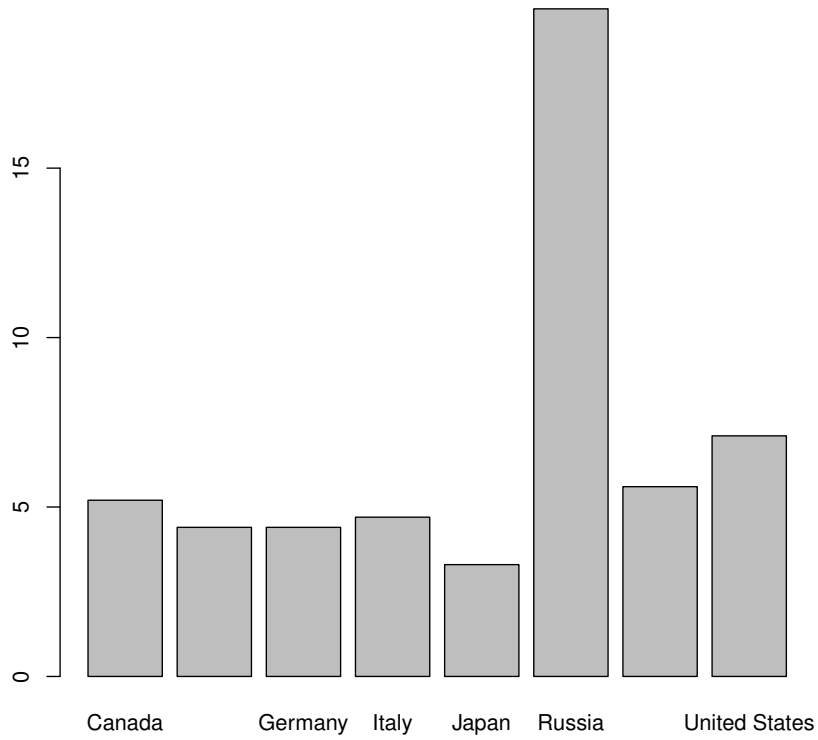

Figure 1.5: Simplest bar plot: infant mortality rate per country.

Several things are wrong with this graph. Glaringly, a bar should not extend beyond its axis. Axes are set as plot options with `xlim` and `ylim`. Also, should you want a box around the graph, `bty` takes care of that. Usually. Bar plots are special and you need to all an extra function after your plot appears. See all graph options with `?par`, which we will use a lot more as we customize our graphs.

```
barplot(g8$X2000, names.arg=g8$Infant.mortality.rate,
↪  ylim=c(0,20), bty='o')        # why oh why won't bty work
↪  like everywhere else!
box()
```

You probably also want all country names to show up. Easiest way to do

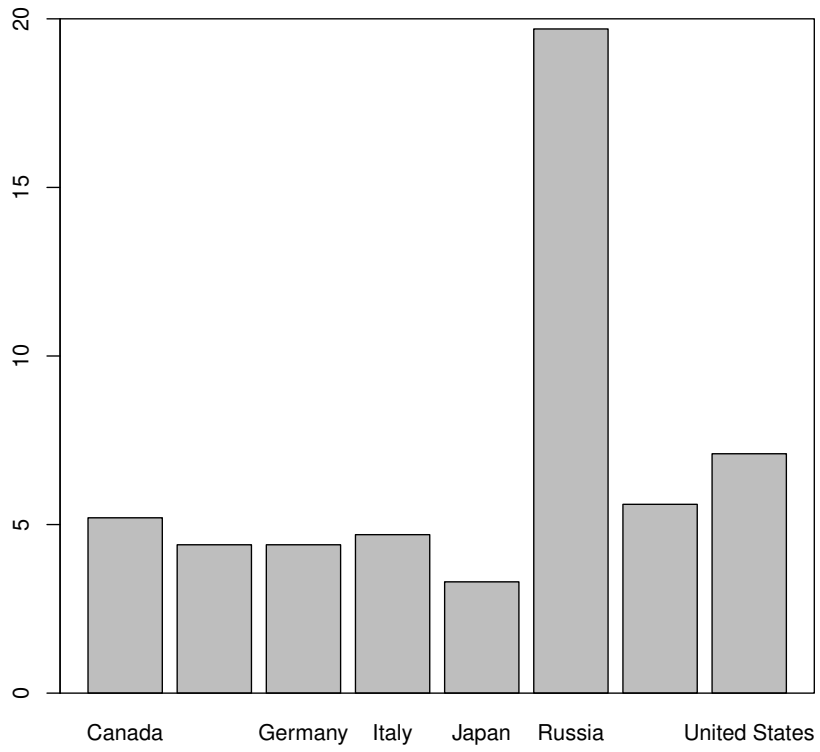

Figure 1.6: Simple bar plot: infant mortality rate per country, axis set.

that is to tilt the axis label text. Enter `par`, used *before* your graph function to set general settings. For this next iteration, let's do a few things at once. First, let's make all labels perpendicular to their axis with `par` and `las`. Let's also demonstrate color manipulation by making each country's bar the dominant color on their flag (this might lead to arbitrary choices) with `col`.

```
> colors = c('red', 'blue', 'black', 'green', 'white', 'snow',
↪  'purple', 'purple4')
> par(las=2)                # axis labels: perpendicular
> barplot(g8$X2000, names.arg=g8$Infant.mortality.rate,
↪  ylim=c(0,20), col=colors)
> box()
```

Colors in R

Colors in R can be specified by their names, if they are among R's list of approved colors, which you can see by calling `colors()`.

A more visually helpful version can be found at Color Chart [1] which, incidentally, has other fascinating references about the use of color in science (good vs. bad color ramps, color blindness considerations).

Additionally, colors can be specified in other formats like `#RRGGBB` where $00 \leq \text{color} \leq \text{FF}$. These values can be found with graphics software or off a color generator on the internet.

Finally, if color space is a factor, additional functions exist: `rgb`, `hsv`, `hcl`, `gray` and `rainbow`.

With the country names printed at the vertical, they are running out of space at the bottom of the graph. More margin is needed there. Figures have two types of margins in R: outer and inner. The inner margin is used to draw the figure title and the axis ticks and labels and can be set in inches with `mai=c(bottom, left, top, right)` or in lines with `mar=c(bottom, left, top, right)`. The outer margin is outside the figure; it makes more sense when several plots are displayed together, as we will do a few exercises down the line. The outer margin as well can be set in inches with `omi=c(bottom, left, top, right)` or in lines with `oma=c(bottom, left, top, right)`. As for the appropriate margin necessary to display the full country name, that's a matter of trial and error. Starting with the current parameters' value of inner margin, I found that a value of 8 worked well.

```
par()$mar
[1] 5.1 4.1 4.1 2.1
> par(mar=c(8, 4.1, 4.1, 2.1))
> barplot(g8$X2000, names.arg=g8$Infant.mortality.rate,
↪  ylim=c(0,20), col=colors)
> box()
```

1.2.4 Histogram

1.2.5 Line graph

1.2.6 Scatter graph

1.2.7 Box and whiskers graph

1.3 Numbers

1.3.1 Center

Mean

Median

Mode

1.3.2 Dispersion

Range

Variance

Standard deviation

Coefficient of variation

Quartiles and percentiles

1.3.3 Shape

Skewness

Kurtosis

L-moments

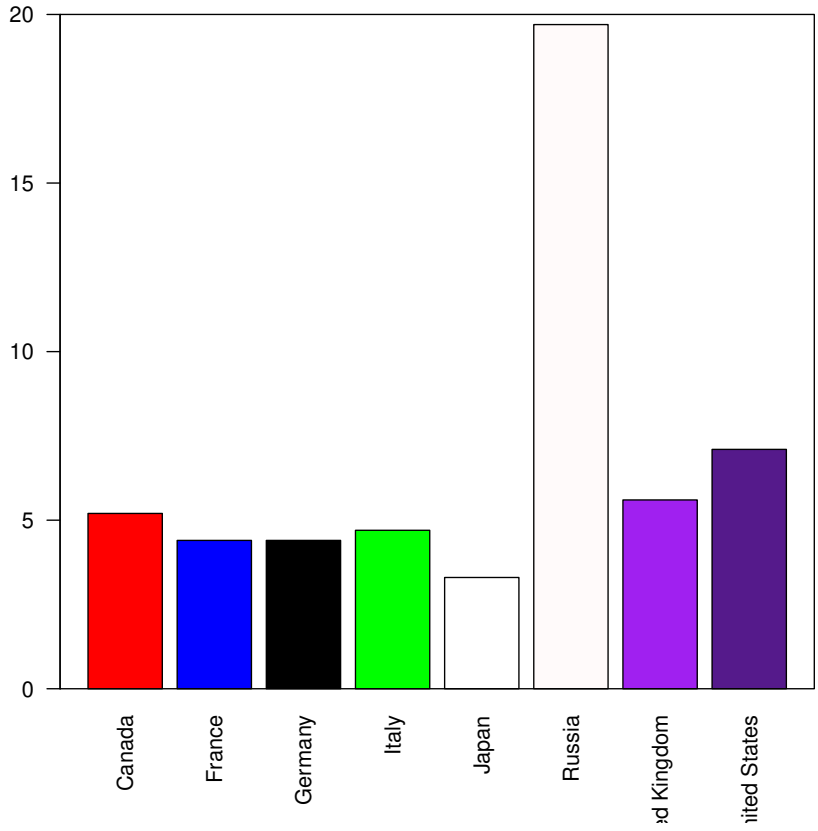

Figure 1.7: Psychedelic bar plot: infant mortality rate per country, axis set, labels perpendicular, colors.

Chapter 2

Probabilities

2.1 Factorial

2.2 Combinations

2.3 Permutations

2.4 Probability Mass/Density Function

Chapter 3

Statistics

Chapter 4

Inferential statistics

Chapter 5

Cheat sheet

5.1 Plumbing

?	?exact_function_name
??	??keyword
typeof	typeof(R_variable)
class	class(R_variable)
str	str(R_variable)
colnames	colnames(R_variable)
as.integer	as.integer(R_variable)

5.2 Data import and export

read.csv	read.csv('delimited_data.csv', header=TRUE, sep=",", dec=".")
read.fwf	read.fwf('fixed_width_data.txt', widths=c(10, 5, 4), header=TRUE, skip=2)
write.csv	write.csv(R_variable, file='desired_file_name.csv', append=FALSE)

Bibliography

[1] Earl F. Glynn. *Chart of R Colors*. Apr. 2005. URL: http://research.stowers.org/mcm/efg/R/Color/Chart/ (visited on 09/10/2018).

[2] Klara Johannson, Mattias Lindgren, and Ola Rosling. *Infant mortality rate (per 1,000 live birth)*. Oct. 2015. URL: https://docs.google.com/spreadsheets/d/1OHMMuHbSFKDolNHXsmgHYlkjSKfAZyyY1P-ddMu_Fz0/pub# (visited on 09/10/2018).

[3] World Health Organization. *Ebola Situation Reports | Ebola*. URL: http://apps.who.int/ebola/ebola-situation-reports (visited on 09/10/2018).

[4] *Pie chart*. en. Page Version ID: 856409948. Aug. 2018. URL: https://en.wikipedia.org/w/index.php?title=Pie_chart&oldid=856409948 (visited on 09/09/2018).

[5] *Quick-R: Data Types*. URL: https://www.statmethods.net/input/datatypes.html (visited on 09/10/2018).

Glossary

discrete variable A variable that refers to categorical data (ex. color of eyes), as opposed to continuous data (ex. height in mm). 3, 7, 9

ordinal variable A variable that refers to categorical data, but where the categories can be ordered (ex. small, medium, large). 7