A Very Short Intodution to R — as I like it

Per Møldrup-Dalum, Max Odsbjerg Pedersen

Introduction

This is an RMarkdown Notebook. It should be opened in RStudio. In RStudio you get different contexts for text like this and code that can be evaluated inline in the notebook.

The code can be evaluated either by pressing Ctrl-Shift-Enter or by pressing the little green arrow at the right margin of the code blocks.

The aim of this short R walk-through is to show the basics of R.

Prolouge: The "as I like it" part

Computer languages like R, that has been around for some decades, live through different styles and opinionated principles. One such principle is expressed by the Tidyverse which I like and advocate. Following this principle is the package tidytext, which we will be relying on through the datasprint

Tidyverse is entered by loading the `tidyverse` library.

If you try this in a newly created R installation, you need to install the libraries on your computer. This is done using the `install.packages` function in the following code part. Note that this is only nessecary once!

```r
#install.packages("tidyverse")
#install.packages("tidytext")
```

```r
library(tidyverse)
library(tidytext)
```

The very basics of R

Standard operators work as we expect

```r
2 + 2
```

```
## [1] 4
```

Naming values

values can be stored for later re-use by giving them a name. Naming values is done with an arrow and can be performed both to the left and to the right.

```r
2 + 2         -> a_name
another_name <-  2 + 3

a_name + another_name
```

```
## [1] 9
```

Pipelines

To me, one of the things that made me look at R seriously was the pipe operator: `%>%`. This operator take what's on the left and sends it to whatever is on the right.

Okay, so how does this pipe work?

Let's say we have a string of words that we want to count. Evaluating a string just gives us the string:

```r
"Gollum or Frodo And Sam"
```

```
## [1] "Gollum or Frodo And Sam"
```

To count elements in a list, we can use the `length` function:

```r
"Gollum or Frodo And Sam" %>% length()
```

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

Okay, so `length` recieves a list with one element: the string/sentence. Let's split that sentence into words (and it's okay to break pipes into multiple lines):

```r
"Gollum or Frodo And Sam" %>%
  str_split(" ", simplify = TRUE) %>%
  length()
```

```
## [1] 5
```

Now, the `simplify = TRUE` is needed because `str_split` can do a lot more that just split a single string of words, but we just want the simple stuff. However we are not really content yet. Two of our words "or" and "And" is what we consider as stopwords and not particularly interesting. We dont want them in our count! We will solve this issue in a moment. First we will shift our attention towards dataframes which is an integral part in R.

Data in tables

Most data come in tables in one form or another. Data could be in an Excel spreadsheet, a csv file, a database table, an HTML table and so on. R understands all these forms and can import them into an R data table, or data frame, as they are called in R-lingo.

A very easy way to create a data table or frame, is to use the `tibble` package, again part of the Tidyverse. The following function creates a data frame with two columns named letter_code and value:

```r
tribble(
  ~letter_code, ~value,
  "a", 2,
  "b", 3,
  "c", 4,
  " ", pi,
  "a", 9
)
```

```
## # A tibble: 5 x 2
##   letter_code value
##   <chr>       <dbl>
## 1 a            2
## 2 b            3
## 3 c            4
## 4             3.14
## 5 a            9
```

Let's do that again and now give the table a name

```r
tribble(
  ~letter_code, ~value,
  "a", 2,
  "b", 3,
  "c", 4,
  " ", pi,
  "a", 9
) -> some_data_frame
```

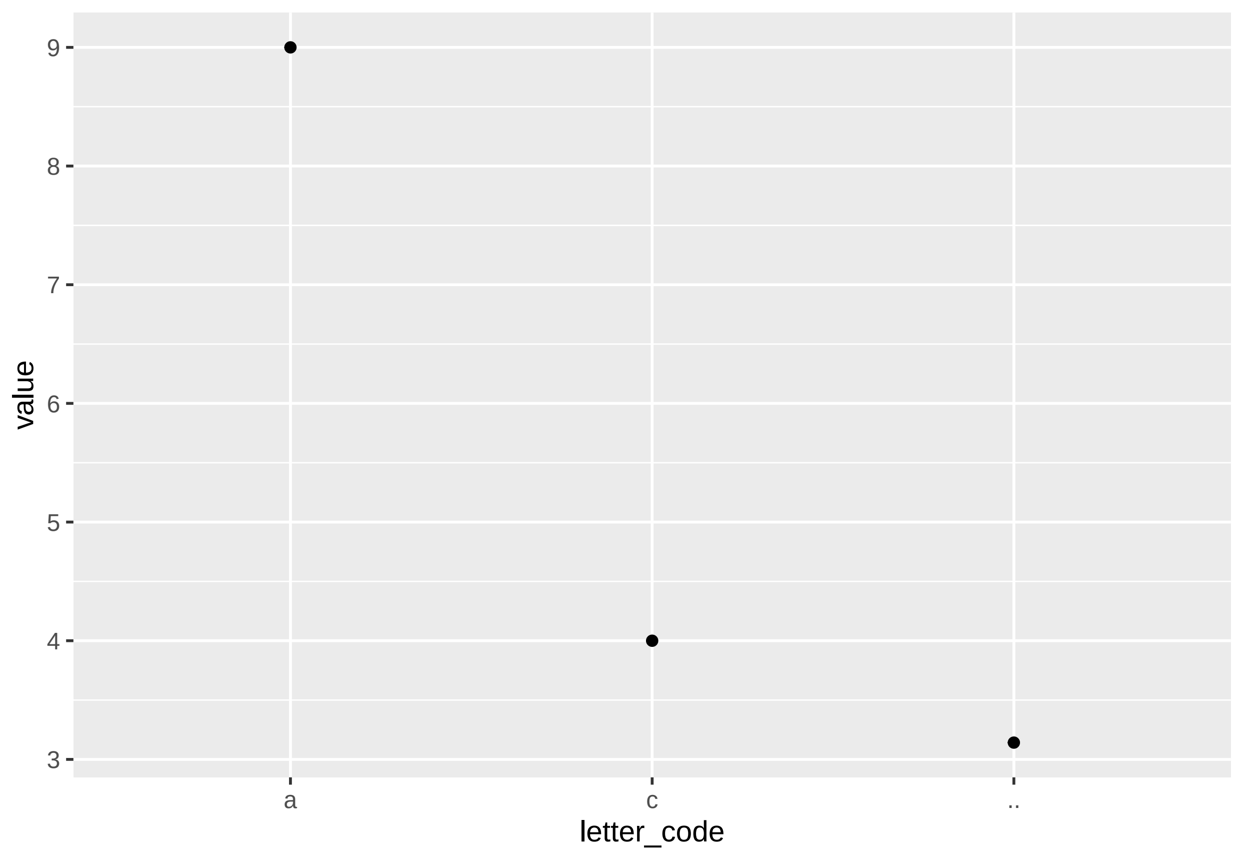
Data frames can now be mutated, filtered, grouped etc. Let's look at all the rows that have `value` greater than 3:

```r
some_data_frame %>%
  filter(value > 3)
```

```
## # A tibble: 3 x 2
##   letter_code value
##   <chr>       <dbl>
## 1 c            4
## 2             3.14
## 3 a            9
```

Look at the same data in a visual way

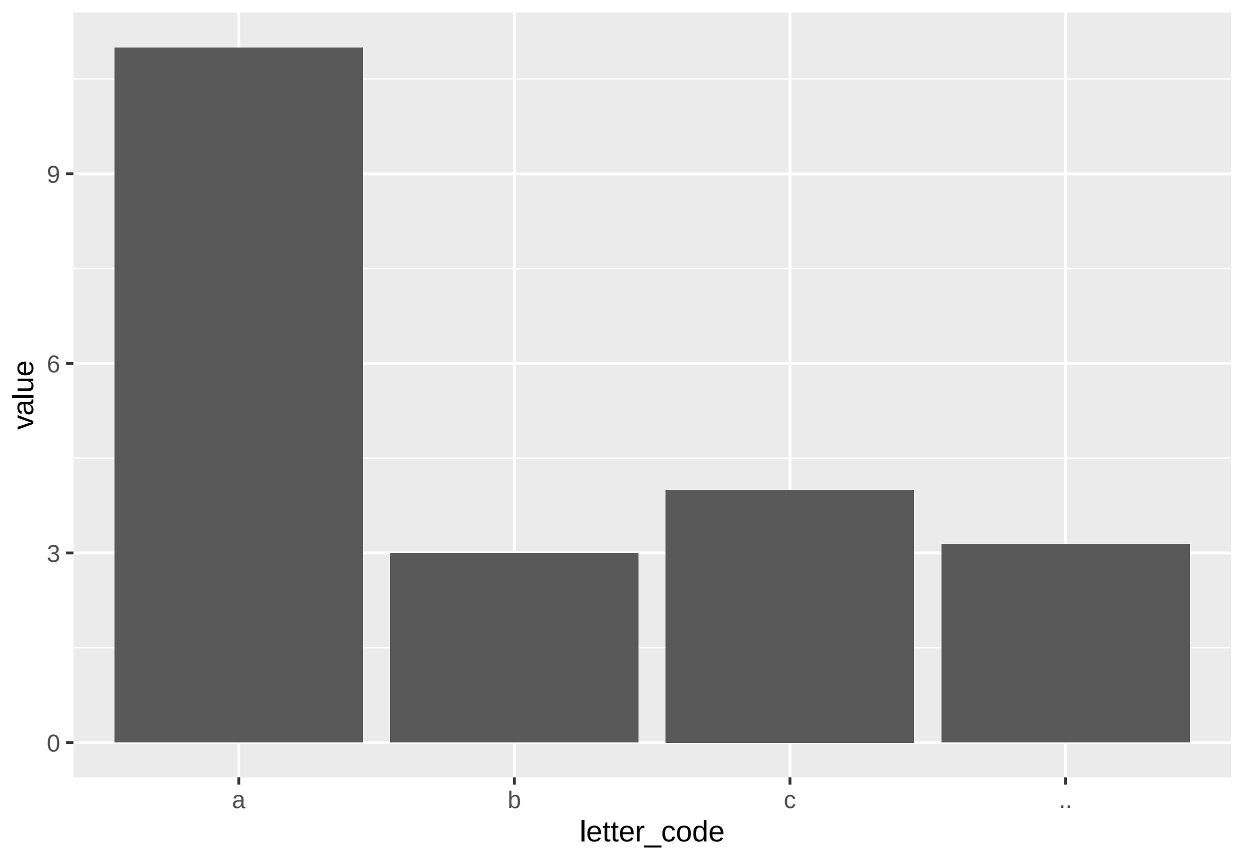
```r
some_data_frame %>%
  filter(value > 3) %>%
  ggplot() +
    geom_point(aes(x = letter_code, y = value))
```



As mentioned ggplot2 is the best plotting library for R. It can produce everything from siple plots to animations and high quality plots ready for publication. It's also a part of the Tidyverse.

Here is a plot aggregating the `values` into `letter_codes`:

```
some_data_frame %>%
  ggplot() +
    geom_col(aes(x = letter_code, y = value))
```



Analysing text with tidytext

Remeber or issue from before? We didn't want to count the stopword ("or" and "And") in the following little text:

```
"Gollum or Frodo And Sam"
```

```
## [1] "Gollum or Frodo And Sam"
```

Let store this in a dataframe instead and save it to an element that we name example_text:

```
tibble(text = "Gollum or Frodo And Sam") -> example_text
```

When you just write the name of an element it will print it out beneath the code chunk:

```
example_text
```

```
## # A tibble: 1 x 1
##   text
##   <chr>
## 1 Gollum or Frodo And Sam
```

The next step is to filter out the stop words of this little text. FOr this we will be using the tidytext package, which splits the text up, so as each word is in its own row. Let's see that in action:

```
example_text %>%
  unnest_tokens(word, text)
```

```
## # A tibble: 5 x 1
##   word
##   <chr>
## 1 gollum
## 2 or
## 3 frodo
## 4 and
## 5 sam
```

The function we used for this is `unnest_tokens`, and within this function we define that it shoud split into words and it should be working on the column text. The result is as we see that each word has been put on its own row, but we also observe that all capital letters has been lowered. But what about the stopwords?

First we need to define our list of stopwords in a data frame:

```
stopwords <- tibble(word = c("or", "and"))
```

Now we have a way to tell R what the stopwords are next step is to remove them from the data. We do this with the function `anti_join`, which we add with a pipe:

```
example_text %>%
  unnest_tokens(word, text) %>%
  anti_join(stopwords)
```

```
## Joining, by = "word"
```

```
## # A tibble: 3 x 1
##   word
##   <chr>
## 1 gollum
## 2 frodo
## 3 sam
```

Thus we have removed the stopwords from our little example. This is of course not very fun with just a small text like we did here. But this method is scaleable and can be applied to huge textual dataset!

(Most of this Rmarkdown was written by Per Møldrup-Dalum and has been published at https://kulturarvscluster.kb.dk/arkiver/87)