Course “Empirical Evaluation in Informatics” SoSe 2017

Freie Universität Berlin, Software Engineering Research Group
Prof. Dr. Lutz Prechelt, Franz Zieris

Practice sheet 2 **Practice R, part 1** due on 2017-04-30

Preface

The purpose of the next three practice sheets is to get well acquainted with R’s basic functions for data manipulation and data analysis. The tasks will build upon each either and they all use the same data.

None of the individual steps is hard: It only takes 1–5 lines of code, the necessary R functions are listed, and putting them together does not require much ingenuity, just you paying attention to what you are doing. Additionally, some demo output based on a small subset of the data is printed at the end of this sheet. This should give you an idea whether what you are doing for each of the tasks below is going in the right direction: When working on your solutions, you may use that data subset and validate your implementation using those outputs. Then you can turn to the complete data sets.

Finally, do not just stop by answering the questions below. Feel encouraged to perform additional analyses on the data using the R functions you just learned.

Task 2-1: Importing data with R

a) First, get the files `jikes.tsv`, `junit.tsv`, `zile.tsv`, and `junit20.tsv` from the KVV (they are bundled in `data.zip`).

They represent tab-separated CVS[1] logs of the projects *Jikes*,[2] *JUnit*,[3] and *Zile*.[4] Each line represents a check-in process (commit) in CVS. Given are the file name `file`, the time `tstamp`, the caller `developer`, the file’s version number `version`, number of lines added or removed with respect to the prior version (`lines_add`, `lines_del`), and the comment of the check-in `description`.

The file `junit20.tsv` contains the first 20 lines of `junit.tsv` so it can be analyzed by hand, if need be. This subset is used for the demo output at the end of this sheet.

b) Read one of the data files into a `data.frame` variable (function needed: `read.table` or `read.delim`).

- Prevent the changing into a factor[5] of the timestamp `tstamp`, the developers’ names `developer`, and the files’ names `file` (by using an `as.is` argument).
- Explicitly add `developer` and `file` to your data frame again as factors `developerf` and `filef` (via `cbind` plus assignment to names, or simply via `$`).
- Delete the column `description` from your data set (by assigning `NULL`, or via the function `subset`).
- The resulting data frame should only contain 8 variables. Get a first overview to make sure your pre-processing was successful. (functions: `names`, `nrow`, `str`, `summary`)

[1] A version-control system

[2] A Java compiler, http://jikes.sourceforge.net

[3] Framework for unit tests in Java, http://junit.org

[4] An Emacs clone, http://www.gnu.org/software/zile

[5] In case you haven’t done it already, read up *thoroughly* on factors in R (`?factor`). These are used very often. Creating a factor is a comfortable way to establish the number of different values that exist in a vector without counting their frequency as `table` would do it. Apart from that, factors are represented as numbers (as opposed to strings), which can save much storage space and CPU time when dealing with them.

Now define your own function[6] called `myread.cvsdata`, which will be given a path to a file with such tab-separated CVS data and returns a data frame with the features described above.

Include your implementation of `myread.cvsdata` in your KVV submission. It should be possible to use your function like this:

```
junit = myread.cvsdata("junit.tsv")
jikes = myread.cvsdata("jikes.tsv")
ziles = myread.cvsdata("zile.tsv")
```

Task 2-2: Elementary data analysis

Answer the following developer-related questions for each of the three projects *Jikes*, *JUnit*, and *Zile*.

You should start by writing functions that take a data frame of the above form as an argument so they can be reused for every project of interest.

a) How many different developers have actually committed into the CVS repository overall?
(possible name for the function: `developer.count`, functions to use: `levels`, `length`)

b) How many changes have been made by the five most active developers respectively?
(possible name: `developer.busy`, functions to use: `sort`, `table`)

c) How many percent of the files did each developer commit at least once?
(possible name: `developer.changedfiles`)

Hint: There are two approaches here.

1. Count the number of existing files for each developer (`tapply`, `levels`, `length`, `factor`, `sort`), or
2. Create a frequency table (developer by file) and count the non-zero entries for each developer (`table`, `apply`, `length`, `levels`, `sum`, `sort`).

Method 1 is slightly shorter to write down and scales better (it requires less memory for bigger data). A frequency table (method 2), however, allows many other queries.

Please answer these questions in *natural language*, i.e. do not simply paste R's output, but at least comment it. (The example outputs at the end of this practice sheet are only for validating your R implementation. Those are not model "answers".)

What strikes you when looking at the result concerning question **c)** for the JUnit data?

Include your implementations of the three functions in your KVV submission.

Example outputs for `junit20.tsv`

Note: The creation of the outputs below went hand in hand with the compilation of the LaTeX file this document is based on. Google for ***knitr*** *if you want to know more.*

Task 2-1 b)

Inspection of the final data frame created using `junit20.tsv`:

```
junit20 = myread.cvsdata("junit20.tsv")
names(junit20)

[1] "file"      "tstamp"     "developer"  "version"     "lines_add"
[6] "lines_del" "developerf" "filef"

nrow(junit20)

[1] 20
```

[6] Read up on functions (`?"function"`) if necessary.

```
str(junit20)

'data.frame':	20 obs. of  8 variables:
 $ file      : chr  "/cvsroot/junit/junit/README.html" "/cvsroot/junit/junit/README.html
 $ tstamp    : chr  "2004-11-17 23:07:28.0" "2002-09-01 00:29:52.0" "2002-08-31 18:44:09
 $ developer : chr  "egamma" "egamma" "egamma" "egamma" ...
 $ version   : Factor w/ 14 levels "1.1","1.1.1.1",..: 3 14 13 12 11 10 9 8 7 1 ...
 $ lines_add : int  660 8 14 128 67 74 20 9 148 0 ...
 $ lines_del : int  527 2 3 8 5 248 6 7 10 0 ...
 $ developerf: Factor w/ 2 levels "egamma","emeade": 1 1 1 1 1 1 1 1 1 1 ...
 $ filef     : Factor w/ 2 levels "/cvsroot/junit/junit/build.xml",..: 2 2 2 2 2 2 2 2 2

summary(junit20)

     file              tstamp            developer            version  
 Length:20          Length:20          Length:20          1.10   :2  
 Class :character   Class :character   Class :character   1.5    :2  
 Mode  :character   Mode  :character   Mode  :character   1.6    :2  
                                                          1.7    :2  
                                                          1.8    :2  
                                                          1.9    :2  
                                                          (Other):8  
   lines_add        lines_del       developerf
 Min.   :  0.00   Min.   :  0.00   egamma:16  
 1st Qu.:  1.75   1st Qu.:  0.75   emeade: 4  
 Median :  8.50   Median :  3.00              
 Mean   : 58.85   Mean   : 42.50              
 3rd Qu.: 31.75   3rd Qu.:  7.25              
 Max.   :660.00   Max.   :527.00              

                              filef   
 /cvsroot/junit/junit/build.xml  : 9  
 /cvsroot/junit/junit/README.html:11  
```

Task 2-2

Expected raw outputs of the developer-related functions for the JUnit20 data set:

```
developer.count(junit20)

[1] 2

developer.busy(junit20)


egamma emeade   <NA>   <NA>   <NA> 
    16      4                      

developer.changedfiles(junit20)

emeade egamma 
    50    100 
```