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

Practice sheet 2 **Practice R, part 1** due on 2018-04-29

Preface (for sheets 2 to 4)

The purpose of the next three practice sheets is to perform a few data analyses and to get well acquainted with R's basic functions along the way. The data we will use are logs from source code repositories of three open-source projects. Mining software repositories is a common approach to learn something about the projects they belong to.

The **primary goal** is to formulate **human-readable conclusions** about those projects. R itself is merely the tool to perform the analyses to get to these answers. Raw **R output** and R implementation **code are only secondary**: They may be helpful to understand how you came to your conclusions, but they are *not* what is actually of interest.

None of the individual analysis steps is hard: It only takes 1–5 lines of code per step. For each step, the names of helpful R functions are listed. They are meant as hints to help you when stuck, not as requirements for your implementation (there is not *one* single implementation). To the same end, some demo output of R code using those function and based on a small subset of the data is printed at the end of each sheet. When having trouble with your implementation, you may use the same data subset and validate your implementation using those outputs.

In any case: remember the primary goal. R code and/or outputs are *no* analysis result.

The individual analysis steps are designed such that they take you to different areas of R's feature set. Whenever you are asked to explore or look something up, **write down your observations.**

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 contain logs from the CVS repositories of the projects *Jikes*, *JUnit*, and *Zile*. Each line represents a check-in of a single file in CVS (which, unlike SVN or Git, puts version numbers on individual files, instead of the whole repository).
- Given are the name of the affected file (`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 and the following sheets.

b) Write a function[1] called `myread.cvsdata`, which will be given a path to a file with such tab-separated CVS logs and returns a `data.frame` with the features described below.

- Create a data frame variable from a tab-separated file (function needed: `read.table` or `read.delim`).
- Prevent the changing into a factor[2] of the timestamp `tstamp`, the developers' names `developer`, and the files' names `file` (by using an `as.is` argument).

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

[2] 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.

- 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 frame (by assigning `NULL`, or via the function `subset`).
- The resulting data frame should only contain 8 variables.

Eventually, 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")
```

c) Verify your implementation by making sure the resulting data frame looks as expected (functions: `names`, `str`.)

d) Get a first overview of the three data sets (JUnit, Jikes, and Zile) by answering at least three simple questions of your choice, e.g.: *Does the number of data points correspond with the respective file sizes?* (functions: `nrow`, `summary`)

> Data validation (as in **d)**) should be done as early as possible, for a number of reasons, including:
>
> - You want to know right from the start whether your data set has some flaws (e.g., missing or corrupt data points).
> - You want to have a rough idea of what your data looks like to detect issues in your process, e.g., if you start out with 4,127 data points and end up with only 4,096 in your final analysis, your data processing might have serious problems.

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. Remember to include your implementations in your KVV submission.

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

c) Per developer: Which percentage of all files mentioned in the log file was checked-in by him/her at least once (their "reach")? (possible name: `developer.changedfiles`)

> **Hint for c)**: There are (at least) two implementation 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.

d) Characterize each of the three projects based on their respective developer profile you just created.

(If you have no idea what to look for, take JUnit as an example: What strikes you when you look at the developers' reach in the JUnit project?)

Example outputs for `junit20.tsv`

> **Important:** The outputs below are only a means for verifying your R implementation. These are not "model answers".

Task 2-1

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

```
# Check consistency
names(junit20)

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

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
```

```
# Raw data for first overview
nrow(junit20)

[1] 20
```

```
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

```
# counting developers
developer.count(junit20)

[1] 2

# most active developers
developer.busy(junit20)


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

# developers' reach
developer.changedfiles(junit20)

emeade egamma
    50    100
```

Note: The creation of the outputs above 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.