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

Practice sheet 2 **Practice R (part 1)** due on 27.04.2015

Please make sure to always prepare your solutions in a way that you are able to present them to your classmates and discuss your solution process effectively.

Task 2-1: Learning elementary data analysis in R, part 1

In this task, you will become familiar with some of R's basic functions for data manipulation and data analysis.

Please solve each sub-step in such a way that the solution may be read off directly after execution without further ado. It only takes 1-5 lines of R code for each task. The entire solution is much shorter than the task description. To gain a solid basic knowledge in R, you should read more in the documentation than is needed for the given task.

In most cases, the functions needed for the solution are mentioned explicitly. However, you are not obligated to use them; other approaches may work just as well.

The task's focus is mainly on R, not on the data dealt with. Do not expect spectacular analysis results, but enjoy the elegance of the approach. Clarify questions concerning the approach and style via the lecture's forum in the KVV.

1. Get the files `jikes.tsv`, `junit.tsv`, `zile.tsv`, and `junit200.tsv` from the lecture's website (they are bundled in `data.zip`).
- These are CVS data sections of the projects Jikes[1], JUnit[2], and Zile[3].
- The files' data are tab-separated.
- 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 `junit200.tsv` only contains the first 200 lines of `junit.tsv`. To save time when working at the solution, first concentrate on this sub-set until you know how to do it, and only then turn to the complete data sets. In the tutorial you should then be able to makes statements concerning the complete data sets.

2. Read the file into a `data.frame` variable.
- Function needed: `read.table` or `read.delim`.
- Prevent the changing into a *factor* of the time stamp `tstamp`, the developers' names `developer` and the files' names `file` by using an `as.is` argument.
  - Read up *thoroughly* on factors: `?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, *factor*s 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 set (by assigning `NULL` or the function `subset`).

[1] Java compiler, http://jikes.sourceforge.net
[2] Framework for unit tests in Java, http://junit.org
[3] Emacs clone, http://www.gnu.org/software/zile

- The resulting data set should only contain 8 variables. Get a first overview.
  - Or do you already have a general idea because you first looked closely at the text file? Good!
  - Functions needed: `names`, `nrow`, `str`, `summary`
- Now define your own function `myread.cvsdata`, which will be given a path to a file with such tab-separated cvs data and returns a data set with the features described above.

  Read up on functions: `?"function"`

3. Answer questions concerning the developers involved:

- How many different developers have actually committed into the CVS overall? (functions: `levels`, `length`)
- How many changes have been made by the five most active developers respectively? (functions: `sort`, `table`)
- How many percent of the files did each developer commit at least once? There are two approaches here:

  (1) You count the number of existing files for each developer (`tapply`, `levels`, `length`, `factor`, `sort`), or

  (2) you draw a frequency table (developer by file) and count for each developer the entries that are non-zero (`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.
- What strikes you when looking at the result concerning the JUnit data?

4. Understand that the prior as well as all following analysis steps need to be performed several times, the analyses differing only in the data set used.

- You should therefore save the final versions of all commands you have developed during analysis in a text file to be able to repeat the analysis easily, if needed. In this way, it will not be necessary to save the workspace; you may safely ignore the question "Save workspace image?".
- It is helpful to collect all steps in a function `myanalyze.cvsdata`, which is parameterized with the data set.
- Additionally, read up on `with`, `attach`, `detach` to simplify your code considerably.

Task 2-2: Quality criteria for research

Listed below you find a list of quality criteria for “good” research. Make sure, you understand the meaning of each criterion.

Visualize or describe the relations between the listed quality criteria (how they depend, influence each other, etc.) and explain (give reasons for) the relation.

- Validity
- Objectivity
- Reliability
- Internal Validity
- External Validity
- Relevance
- Credibility
- Replicability