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

Practice sheet 3 **Practice R (part 2)** due on 04.05.2015

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 3-1: Learning elementary data analysis in R, part 2

In this task, you will get to know some more functions for the data analysis with R, especially those for time specification and graphic illustration.

We continue working with the data sets `jikes.tsv`, `junit.tsv`, `zile.tsv` from the second practice sheet and its subset `junit200.tsv`. Again, each step is possible with only a few lines of R code.

1. The time stamp `tstamp` in your imported data sets (e.g. `junit`) is so far available as string. To enable operations in R, it needs to be converted into a POSIXct object[1].
   - Add the converted variable as `tstamp2` to the data set (most easily done with `junit$tstamp2 = ...`).
   - Look at the elements of `tstamp2` and understand how the objects' illustration on the screen is realized. Use `mode` and `class`.
   - Functions needed: `as.POSIXct`, (`strptime`)
2. Now add the time stamp as `tstamp3` in the Unix format (i.e. the number of seconds since 1970-01-01). Often, this is more convenient when calculating with times.
   - Function needed: `as.numeric`
   - Background: Compare the results of `diff(junit200$tstamp2[1:2])` and `diff(junt200$tstamp3[1:2])`
   - Use `print.default`, `class` and `?difftime` to understand the difference.
3. Examine the time variations of the developers' activities during the course of the day and week.
   - Extract the respective weekday from `tstamp2` and add it to your data.frame as the variable `wday`.
   - Do the same for the hour and add it your data.frame as the variable `hour`.
   - Functions needed: `as.POSIXlt` (you extract one element each from the result)
   - Now examine a compilation of the sum of activities at each hour (irrespective of the date) and on each weekday (irrespective of the date).
   - Which are the minimum and maximum values? How much smaller or bigger are they as the average? What do you gather from the progression? Functions needed: `as.vector`, `summary`, `table`
4. Visualize the hourly and weekly activities just calculated.
   - Function needed: `plot`

[1] For a better understanding you may use the following sources: `?DateTimeClasses` and the article on date-time classes by Brian D. Ripley and Kurt Hornik in the R-News Volume 1/2, June 2001 (http://cran.r-project.org/doc/Rnews/Rnews_2001-2.pdf#chapter*.12)

5. `plot` is an object-oriented function (as is `print`, which always is implicitly applied to each command's result). There are separate versions of `plot` for many different types of objects.
   - Get an overview with the help of `methods(plot)` and read up on the documentation of three `plot` functions you consider interesting.
   - We have just implicitly used `plot.table` to illustrate weekdays. `plot.factor` would lead to similar results. Try it.
   - Functions needed: `factor`, `plot`
   - Assign real short names to the weekdays by using the `labels` argument of `factor`.
6. Get an overview of the distribution of the number of lines added or deleted in one step per developer (variables `lines_add` and `lines_del`)
   - Boxplots: Use `bwplot` with a formula of the type `developer~log(lines_add+1,2)`. Also use the arguments `varwidth` and `box.ratio` to improve the illustration.
   - Are there developers who often add or delete particularly many lines, or spread the sizes particularly much or little?
   - Compare the size of the boxes to the illustration of the number of jobs with `plot(table(df$developer))`
   - Functions needed: `library(lattice)`, `bwplot`, `log`, `plot`, `table`, `?panel.bwplot`
7. Repeat the same analysis with a distribution of the data according to data type instead of developer.
   - Functions needed: `bwplot`, `factor`, `log`
8. Reconstruct the results for developers with the help of a density plot. It illustrates the frequency distribution with a curve.
   - Here, the formula could be for instance `log(lines_add+1,2)|developer`; also use the argument `width=1`. Read up on it and test its effect.
   - Functions needed: `densityplot`, `log`
9. Now read at least the sections 1, 2.1 and 3.2.2 in the article *"Two Case Studies of Open Source Software Development: Apache and Mozilla"* (to be found on the course's website as "MocFieHer02") to get an idea of its investigations.
   - Understand the meaning of the values on the x-axis in Fig. 1 in section 3.2.2.
   - Carry out an analysis with our data along the lines to the one in section 3.2.2 in Fig. 1. Compile corresponding images for our data sets and interpret the results.
   - Use a linear scale on the x-axis.
   - Each of your three images needs to contain three curves.
   - Functions needed: `cumsum`, `length`, `lines`, `plot`, `sum`, `tapply`