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

Practice sheet 3 **Practice R, part 2** due on 2017-05-07

Preface

On this sheet, you will get to know some more functions for the data analysis with R, especially those for time-related operations and graphic illustration.

We continue working with the data sets `jikes.tsv`, `junit.tsv`, `zile.tsv` from the second practice sheet. Again, each step can be accomplished with only a few lines of R code, and yet again, helpful functions are mentioned in parentheses.

Task 3-1: Data manipulation

a) So far, the time stamp `tstamp` in your imported data sets (e.g. `junit`) is only available as a string. To enable date-related operations in R, it needs to be converted into a `POSIXct` object.[1] (Functions needed: `as.POSIXct`)

Add the converted variable as `tstamp2` to the data set (most easily done with `$`).

b) Now add the time stamp in the Unix format (i.e. the number of seconds since 1970-01-01) as `tstamp3`. Quite often, this is more convenient when doing calculations involving with dates and times. (Function needed: `as.numeric`)

c) Understand the ways in which these two representations differ.

- Look at the elements of `tstamp2` and understand how the objects' representation on the screen is realized. Use `mode` and `class` to learn more about the objects.
- Calculate the difference between two time stamps from `tstamp2` and store the result in a variable (e.g. using `diff` or just a simple `-`).
- Now do the same calculation, but use the corresponding values (= same index) from `tstamp3`.
- Understand and explain the difference in how the two results are displayed and how this works. (Functions: `print.default`, `class` and `?difftime`)

Since you now have written some functionality which enhances your data to make subsequent analyses easier, you should include the code for adding `tstamp2` and `tstamp3` to your data frame in your existing `myread.cvsdata` function, and include the new implementation in your submission.

d) Examine the time variations of the developers' activities during the course of the days and throughout the 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 from the result)
- Now examine a compilation of the sum of activities at each hour (0-23, disregarding the date) and on each weekday (0-6, disregarding the hour).
  Which are the minimum and maximum values? How much smaller or bigger are they compared to the average? What conclusion can you draw from the progression? (Helpful functions: `as.vector`, `summary`, `table`)

[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)

Task 3-2: Plotting data

a) Visualize the hourly and weekly activities calculated in Task **3-1 d)**. (Function needed: `plot`)

- `plot` is an object-oriented function (as is `print`, which is always 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.
- Assign real short names to the weekdays by using the `labels` argument of `factor`. You can also pre-define the `factor`'s levels (using the `levels` argument) if you want to "count" zero occurrences (e.g. as are the hours 3-14, or Tuesday in the JUnit20 data).

b) Get an overview of the distribution of the number of lines added or deleted 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`

c) 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`

The next task requires a few more lines of R code. If you try it and don't succeed, you may skip this one, but only if you come up with (and execute) another idea for a non-trivial analysis on the same three data sets which results in a (series of) plot(s).

d) Now, read at least the sections 1, 2.1, and 3.2.2 of the article *"Two Case Studies of Open Source Software Development: Apache and Mozilla"* by Mockus, Fielding, and Herbsleb (to be easily found on Google Scholar) 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 (one for each project) needs to contain three curves.
- Functions needed: `cumsum`, `length`, `lines`, `plot`, `sum`, `tapply`

Example outputs for `junit20.tsv`

Task 3-1

```
# Loading data
junit20 = myread.cvsdata("junit20.tsv")
```

```
# Inspecting new variable tstamp2
junit20$tstamp2

##  [1] "2004-11-17 23:07:28 CET"  "2002-09-01 00:29:52 CEST"
##  [3] "2002-08-31 18:44:09 CEST" "2002-08-23 20:43:51 CEST"
##  [5] "2001-05-21 21:50:15 CEST" "2001-04-08 02:18:42 CEST"
##  [7] "2001-04-01 23:22:15 CEST" "2001-01-17 01:02:18 CET"
##  [9] "2001-01-10 00:39:50 CET"  "2000-12-03 15:36:14 CET"
## [11] "2000-12-03 15:36:14 CET"  "2004-11-17 23:47:23 CET"
## [13] "2004-11-13 01:33:00 CET"  "2002-08-31 18:44:09 CEST"
## [15] "2002-08-23 20:43:51 CEST" "2002-02-14 21:58:36 CET"
## [17] "2002-02-07 00:43:07 CET"  "2002-02-06 22:12:01 CET"
## [19] "2002-02-06 21:42:19 CET"  "2001-05-21 21:50:15 CEST"

# Inspecting new variable tstamp3
junit20$tstamp3

##  [1] 1100729248 1030832992 1030812249 1030128231  990474615  986689122
##  [7]  986160135  979689738  979083590  975854174  975854174 1100731643
## [13] 1100305980 1030812249 1030128231 1013720316 1013038987 1013029921
## [19] 1013028139  990474615
```

Raw hourly activity for `junit20`:

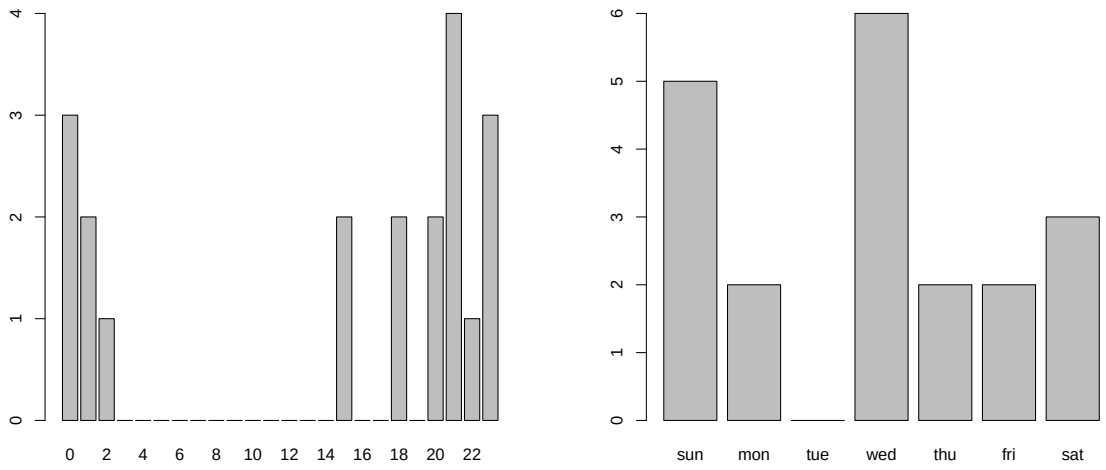
```
raw.hours(junit20)

##
##  0  1  2 15 18 20 21 22 23
##  3  2  1  2  2  2  4  1  3

raw.hours.summary(junit20)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   2.000   2.222   3.000   4.000
```

Raw activity per day of week for `junit20`:

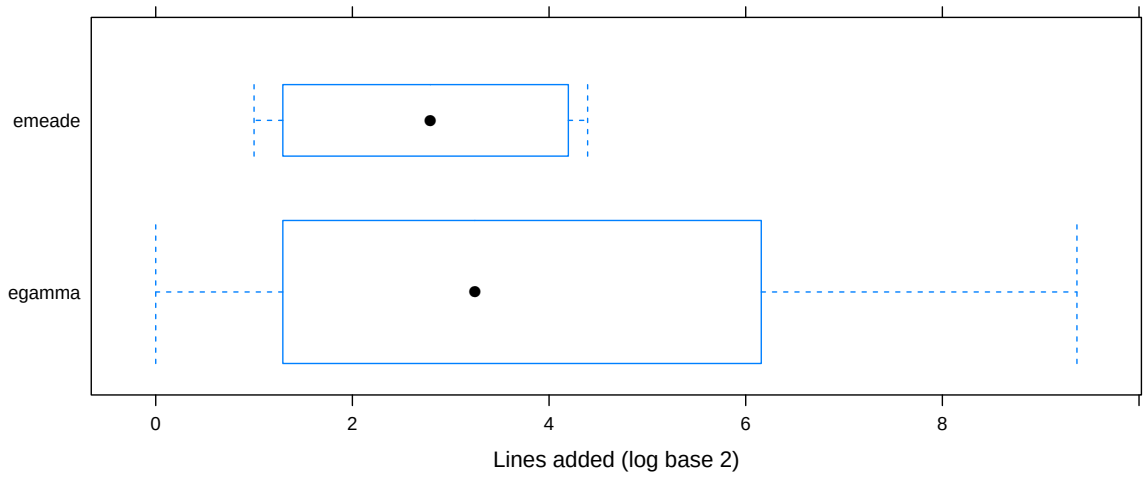
```
raw.weekdays(junit20)

##
## 0 1 3 4 5 6
## 5 2 6 2 2 3

raw.weekdays.summary(junit20)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.000   2.500   3.333   4.500   6.000
```

Task 3-2

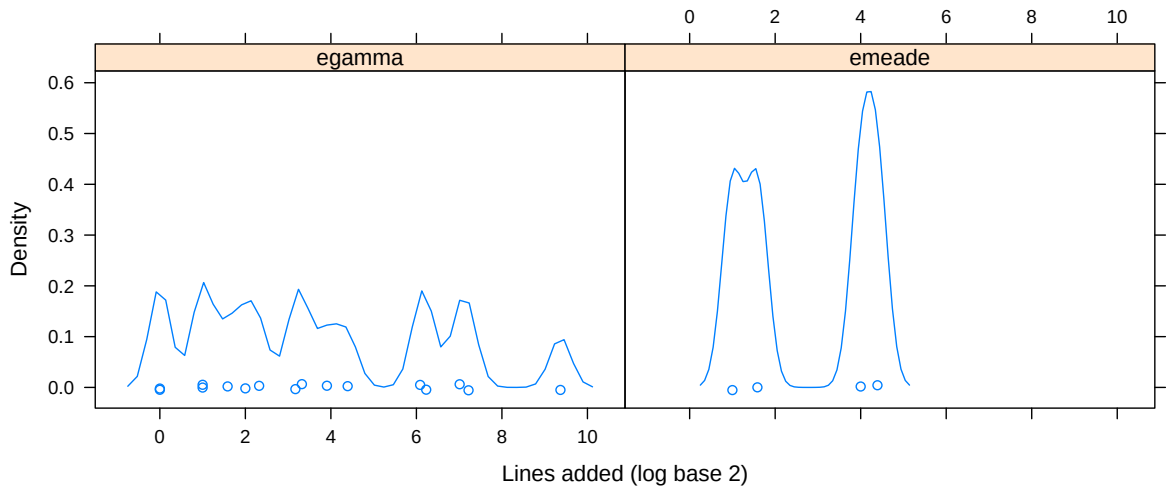
```
# Plots for junit20's hour and weekday distribution
myplot.hours.bars(junit20)
myplot.wdays.bars(junit20)
```




```
# Boxplot for lines_add per developer of junit20
myplot.lines_add.devs.boxplot(junit20)
```



```r
# Density plot for lines_add per developer of junit20
myplot.lines_add.devs.densityplot(junit20)
```




```r
# Plotting the developer participation for the junit20 subset
# (inspired by Mockus et al.)
myplot.participation(junit20)
```