CompStat/R - Paper 1

Group 2: Carlo Michaelis, Patrick Molligo, Lukas Ruff

11 May 2016

Part I

1. What are the atomic vector types in R? Explain which value they can take and give an example!

There are six *atomic* (or *basic*) vector types in R:

- **character**: Text, i.e. string variables.
- **numeric**: Real numbers, i.e. float variables.
- **integer**: Integers, i.e. values in $\mathbb{Z}$.
- **complex**: Complex numbers, i.e. a pair of values with a real and imaginary part.
- **logical**: Boolean variables, i.e. either 1 (`TRUE`) or 0 (`FALSE`).
- **raw**: A raw vector contains fixed-length sequences of bytes.

Examples

```
a <- c("blue", "red", "yellow")     ## character
b <- c(pi, exp(1), 0, 1)            ## numeric
c <- 1:10                           ## integer
d <- c(0+1i, 1+1i)                  ## complex
e <- c(TRUE, FALSE)                 ## logical
f <- raw(length = 3L)               ## raw
```

It is important to note, that a vector can only contain elements of the same type. We can check the type of an object using the `class`-function.

```
# verify types by using class function
lapply(list(a,b,c,d,e,f), class)
```

```
## [[1]]
## [1] "character"
##
## [[2]]
## [1] "numeric"
##
## [[3]]
## [1] "integer"
##
## [[4]]
## [1] "complex"
##
## [[5]]
## [1] "logical"
##
## [[6]]
## [1] "raw"
```

2. What is the difference between generic and atomic vectors?

- An *atomic vector* can only contains objects of the same class. An example would be a vector which contains only integers.
- A *generic vector* (in R representend as a `list`) can conatain objects of different classes. An example would be a vector which contains characters and numbers.

3. Explain the following statement: "A data frame is a list, but not every list is a data frame."

- A `list` is an object containing collections of objects. The types of the elements of the list can be different. It is for example allowed that a `list` contains a vector of real values (doubles) and a vector of characters. The length of the containing vectors can be *different*.
- A `data frame` is also an object containing colletions of objects. The types of the elements of the list can also be different. But the length of the containing vectors have to be *the same*. We can think of a `data frame` as a table or matrix, where each row is an observation and each column a different variable. The length of each element or column are the number of rows or observations.

In conclusion `list` and `data frame` are very similar, but the `data frame` has one more restriction (same length of all vectors). That is why a `data frame` is always a `list`, but a `list` is not always a `data frame`.

Part II

The following code will perform a simulation of 100'000'000 samples from a $\mathcal{N}(5, 10)$ distribution, i.e. a normal distribution with mean $\mu = 5$ and standard deviation $\sigma = 10$. For reproducibility, we set a seed for the random number generator. In a second step, the cumulative sums of the first 100 samples are computed in two different ways, where the function `cumsum` returns a vector where element i is the cumulative sum up to sample i. Finally, we check if the two ways of computing the cumulative sums up to sample 100 result in exactly equal vectors.

The code with comments in detail:

```
# Set the state of the random number generator (RNG) to 1
set.seed(1)

# Perform simulation of 1e8 samples from a normal distribution with mean 5
# and standard deviation 10
largeVector <- rnorm(1e6, mean=5, sd=10)

# Compute the cumulative sums for the whole "largeVector" and subset the
# first 100 elements
a <- cumsum(largeVector)[1:100]

# Compute the cumulative sums only for the first 100 elements of
# "largeVector"
b <- cumsum(largeVector[1:100])

# Check, whether both ways of computation are exactly identical
identical(a, b)
```

```
## [1] TRUE
```

Of course, both ways of computing the cumulative sums for the first 100 samples above have the same result and hence `identical(a, b)` returns `TRUE`, but computation `a` is very inefficient compared to computation `b` since we first apply `cumsum` to the whole `largeVector`, i.e. we compute the cumulative sums for 100'000'000 elements and then only look at the first 100 elements. Computation `b` instead only computes the cumulative sums for the subset of the first 100 elements of `largeVector`. In the following code, we stop the time for each of the two ways of computation:

```
# Computation method a
system.time(cumsum(largeVector)[1:100])
```

```
##    user  system elapsed
##   0.004   0.004   0.008
```

```
# Computation method b
system.time(cumsum(largeVector[1:100]))
```

```
##    user  system elapsed
##       0       0       0
```

The results prove our reasoning above, since the first computation takes much longer than the second.

Part III

We consider dataset from "Munchner Mietspiegel 2003" which contains 13 variables about 2053 flats in Munich. In the dataset the logical variables have following encoding: 'yes' is 1 and 'no' is 0. The variables are:

- **nm**: rent in EUR
- **nmqm**: rent per m^2 in EUR
- **wfl**: living space in m^2
- **rooms**: number of rooms
- **bj**: year of construction
- **bez**: district
- **wohngut**: good residential area (yes/no)
- **wohnbest**: good residential area (yes/no)
- **ww0**: water heating (yes/no)
- **zh0**: central heating (yes/no)
- **badkach0**: tiles in bathroom (yes/no)
- **badextra**: optional extras in bathroom (yes/no)
- **kueche**: luxury kitchen (yes/no)

Data import and descriptive statistics

First we read the data into our environment using `load` function. We will have a look to the raw data using `head` and we will get some first descriptive statistic information of the interval scaled variables using `summary` function.

<<>>= load('miete.Rdata') head(miete) summary(miete$nm) summary(miete$nmqm) summary(miete$wfl) summary(miete$roo summary(miete$bj) @

We get the `min`, the `max`, the first quantile, the third quantile, the `mean` and the `median`. If we also want to get some extra information like the standard deviation and maybe skew and kurtosis, we can use the `psych` library and the containing function `describe`. In this case we include all variables.

<<>>= library(psych) describe(miete) @

With the above results we can also do a quick validation. The `min` and `max` of the logical (yes/no) variables should be 0 and 1 respectively, which is the case. To get the amount of missing values we can calulcate the `sum` of `is.na()`.

<<>>= sum(is.na(miete)) @

There are no missing values in the whole dataset.

%% TODO: More validation? %%

Identify relevant regressors and fit regression model

To idenfity relevant regressors we can apply `lm()`, which calculates a linear model, to all variables. The first argument of the function is the formular. In our case we want to do a regression of the rent in EUR (`miete$nm`) on all other variables (we can use the `.` to include all variables). In the second argument we set our dataset.

<<>>= regrel <- lm(miete$nm ~ ., data = miete) @

We can omit all variables which have no significant slope. To get the slope we can have a look to the `summary` of the result of the linear regression.

<<>>= summary(regrel) @

We would suggest to include all variables which are significant on a 99% level (* or **). With the relevant variables we can fit the regression.

<<>>= summary(lm(miete$nm ~ miete$nmqm + miete$wfl + miete$wohnbest + miete$ww0 + miete$kueche, data = miete)) @

Discussion of model fit and interpretation