Berkstats Day 2

Ulrich Matter

22 February 2017

Working with Data in R: Data Structures and Indeces

Vectors and Lists

```r
# A vector containing numeric (or integer) values
numeric_vector <- 10:20
numeric_vector[2]
```

```
## [1] 11
```

```r
numeric_vector[2:5]
```

```
## [1] 11 12 13 14
```

```r
# A string vector ('a vector containing text')
string_vector <- c("a", "b", "c")
string_vector[-3]
```

```
## [1] "a" "b"
```

```r
# Lists
# A list can contain different types of elements, for example a numeric vector and a string_vector
mylist <- list(numbers = numeric_vector, letters = string_vector)
mylist
```

```
## $numbers
##  [1] 10 11 12 13 14 15 16 17 18 19 20
##
## $letters
## [1] "a" "b" "c"
```

```r
# We can access the elements of a list in various ways
# with the element's name
mylist$numbers
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```r
mylist["numbers"]
```

```
## $numbers
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```r
# via the index
mylist[1]
```

```
## $numbers
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```r
# with [[]] we can access directly the content of the element
mylist[[1]]
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```r
# lists can also be nested (list of lists of lists....)
mynestedlist <- list(a = mylist, b = 1:5)
```

Matrices and Data Frames

```r
# matrices
mymatrix <- matrix(numeric_vector, nrow = 4)
```

```
## Warning in matrix(numeric_vector, nrow = 4): data length [11] is not a sub-
## multiple or multiple of the number of rows [4]
```

```r
# get the second row
mymatrix[2,]
```

```
## [1] 11 15 19
```

```r
# get the first two columns
mymatrix[, 1:2]
```

```
##      [,1] [,2]
## [1,]   10   14
## [2,]   11   15
## [3,]   12   16
## [4,]   13   17
```

```r
# data frames ("lists as columns")
mydf <- data.frame(Name = c("Alice", "Betty", "Claire"), Age = c(20, 30, 45))
mydf
```

```
##     Name Age
## 1  Alice  20
## 2  Betty  30
## 3 Claire  45
```

```r
# select the age column
mydf$Age
```

```
## [1] 20 30 45
```

```r
mydf[, "Age"]
```

```
## [1] 20 30 45
```

```r
mydf[, 2]
```

```
## [1] 20 30 45
```

```r
# select the second row
mydf[2,]
```

```
##    Name Age
## 2 Betty  30
```

Classes and Data Structure

```r
# have a look at what kind of object you are dealing with
class(mydf)
```

```
## [1] "data.frame"
```

```
class(mymatrix)
```

```
## [1] "matrix"
```

```
# have a closer look at the data structure
str(mydf)
```

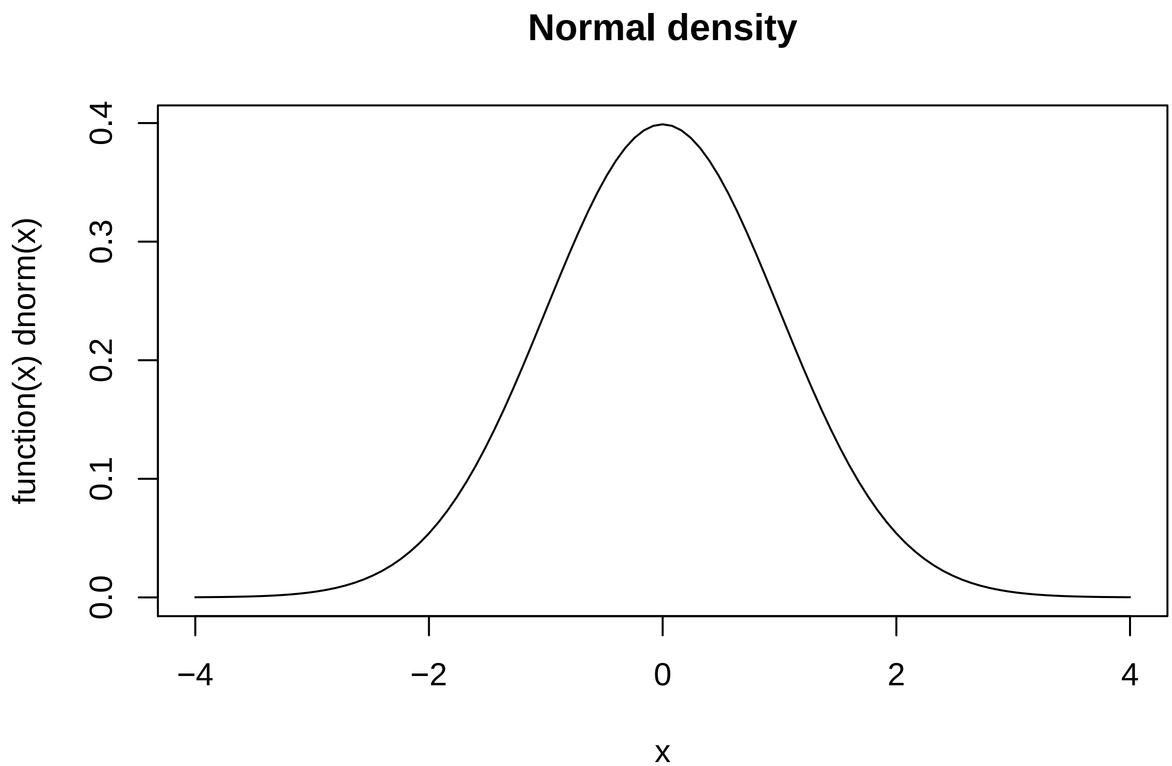
```
## 'data.frame':    3 obs. of  2 variables:
##  $ Name: Factor w/ 3 levels "Alice","Betty",..: 1 2 3
##  $ Age : num  20 30 45
```

Z-Scores and the Standard Normal Distribution

```
# The Z-Score formula in R
# define the parameter values
X <- 10
mu <- 12
sigma <- 2
# compute the z-score
z <- (X - mu) / sigma
z
```

```
## [1] -1
```

```
# Plot the Standard Normal Distribution
plot(function(x) dnorm(x), -4, 4, main = "Normal density")
```



```r
# Get the area under the curve (probability of observing a value of a certain size)
pnorm(-1)
```

```
## [1] 0.1586553
```

```r
pnorm(-2)
```

```
## [1] 0.02275013
```

```r
pnorm(-1) - pnorm(-2)
```

```
## [1] 0.1359051
```

Standard Errors

```r
# forumla for standard error
# define parameter values
s <- 20
n <- 100
# compute the standard error (of the mean)
se <- s / sqrt(n)
se
```

```
## [1] 2
```

```r
# write your own standard error (of the mean) function
se <- function(x) {
     s <- sd(x)
     n <- length(x)
     se <- s / sqrt(n)

     return(se)
}

# draw a random sample of size 100 and compute the mean and its estimated standard error
mysample <- rnorm(100)
mean(mysample)
```

```
## [1] -0.09034929
```

```r
se(mysample)
```

```
## [1] 0.09682044
```

```r
# repeat this but this time with a larger sample
mysample <- rnorm(1000)
mean(mysample)
```

```
## [1] -0.01860827
```

```r
se(mysample)
```

```
## [1] 0.03048619
```

Hypothesis Testing: the T-Statistic

Reproduce the example from the presentation

```r
# define parameters
mu <- 39000
sample_mean <- 37000
sample_sd <- 6150
n <- 100

# calculate the  standard error of the sample mean
se <- sample_sd / sqrt(n)

# compute the t-statistic (and compare it with the critical value)
t <- (sample_mean - mu) / se

# look up the p-value
# (the fraction of the mass under the standard normal distribution)
2*pnorm(-abs(t))
```

```
## [1] 0.001145829
```

Extended Example

```r
# I) Compute the t-value step by step with our own implementation while controlling the properties of t

# define size of sample
n <- 100
# draw the random sample from a normal distribution with mean 10 and sd 2
sample <- rnorm(n, mean = 10, sd = 2)

# compute the sample mean
sample_mean <- mean(sample)
# compute the sample sd
sample_sd <- sd(sample)
# estimated standard error of the mean
mean_se <- sample_sd/sqrt(length(sample))

# compute the t-statistic for the null hypothesis: H0: mu = 9
t <- (sample_mean - 10) / mean_se
t
```

```
## [1] 0.9884692
```

```r
# get the p value
2*pnorm(-abs(t))
```

```
## [1] 0.3229229
```

```r
# II) Apply the R-function t.test to do the same!
t.test(sample, mu = 10)
```

```
##
##  One Sample t-test
```

```
##
## data:  sample
## t = 0.98847, df = 99, p-value = 0.3253
## alternative hypothesis: true mean is not equal to 10
## 95 percent confidence interval:
##   9.80395 10.58528
## sample estimates:
## mean of x
##  10.19462
```

Exercises

1. Got to http://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html. 2. Pick a data set that interests you, load it with the data() function.

```
# yes, I choose the swiss data set...
data(swiss)
# have a look at what we are dealing with here
str(swiss)
```

```
## 'data.frame':    47 obs. of  6 variables:
##  $ Fertility       : num  80.2 83.1 92.5 85.8 76.9 76.1 83.8 92.4 82.4 82.9 ...
##  $ Agriculture     : num  17 45.1 39.7 36.5 43.5 35.3 70.2 67.8 53.3 45.2 ...
##  $ Examination     : int  15 6 5 12 17 9 16 14 12 16 ...
##  $ Education       : int  12 9 5 7 15 7 7 8 7 13 ...
##  $ Catholic        : num  9.96 84.84 93.4 33.77 5.16 ...
##  $ Infant.Mortality: num  22.2 22.2 20.2 20.3 20.6 26.6 23.6 24.9 21 24.4 ...
```

```
# have a look at the first few lines
head(swiss)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality
## Courtelary               22.2
## Delemont                 22.2
## Franches-Mnt             20.2
## Moutier                  20.3
## Neuveville               20.6
## Porrentruy               26.6
```

```
# get more info about the variables in that data set
?swiss
```

3. Pose an empirical question you want to answer with that data set. In the case of the `swiss` data set, for example, is the average fertility in the entire population 85?

4. Formulate a null-hypothesis and test it with the techniques learned today. H0: mu = 85. This requires a t-statistic of the Fertility mean. We can compute this statistic either with the formulas and R-functions discussed on Day 2. Or we directly compute it with t.test() provided by R. The following code demonstrates both approaches step by step.

I. Own implementation (with a detailed explanation)

```
# a) compute sample mean and sample standard deviation, record how many observations we are having
# in our sample, define the population mean (that you want to test for)
sample_mean <- mean(swiss$Fertility)
sample_sd <- sd(swiss$Fertility)
n <- length(swiss$Fertility) # alternatively use nrow(swiss)
mu <- 85

# b) compute the (estimate of the) sample mean standard error
se <- sample_sd / sqrt(n)

# c) compute the t-statistic
t <- (sample_mean - mu) / se
t
```

```
## [1] -8.154018
```

```
# d) check what p-value is associated with that t-statistic
# i.e., check what fraction of the standard normal distribution has an at least as extreme value as
# the t value we computed.
pval <- 2*pnorm(-abs(t))
pval
```

```
## [1] 3.520284e-16
```

Note how we get from the t-value to the p-value here: `2*pnorm(-abs(t))`. This short line of code involves a lot. Let's have a look at specific aspects. Recall the standard normal distribution function which is implementd in `pnorm(x)`. This function returns the fraction of the mass under the standard normal distribution curve that is smaller than or equal to `x` (i.e., the probability of observing a value that is equal to or smaller than `x`). Lets have a closer look in R:

```
# the probability of observing a value at least as small as -1 in a standard normal distribution
pnorm(-1)
```

```
## [1] 0.1586553
```

```
# or in percent
pnorm(-1) * 100
```

```
## [1] 15.86553
```

Recall that we are interested in the probability of observing a value at least as *extreme* as `t`. Thus we need to consider the absolute value of `t`, `|t|`, and want to know at once what the probability of observing a value at least as small as `-t` *or* at least as big as `t` (therefore, the `abs(t)` which returns the absolute value).

```
# demonstration of concept of absolute value
abs(-1)
```

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

```
abs(1)
```

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

```
# in our example of t
abs(t)
```

```
## [1] 8.154018
```

Recall that `pnorm(x)` returns by default the probability of observing a value that is at least as small as `x` therefore the `-abs(t)`. Finally, we also know that the standard normal distribution is symmetric. Thus the fraction of mass under the curve between `-abs(t)` and -infinity (everything to the left of `-abs(t)` under the curve) is exactly the same as the one between `abs(t)` and infinity (everything to the left of `abs(t)` under the curve). Hence we can simply multiply the the value of `pnorm(-abs(t))` by two: `2*pnorm(-abs(t))` for simplicity's sake. Alternatively we could compute the probability for the 'lower tail' and the 'upper tail' separately and build the sum of the two.

```r
# compute the lower tail probability
lowerp <- pnorm(-abs(t))
# compute the upper tail probability
upperp <- pnorm(abs(t), lower.tail = FALSE)
# build the two-tail p-value
twotailp <- lowerp + upperp

# proof that this is the same as above
twotailp == 2*pnorm(-abs(t))
```

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

II. Apply the R function for t-tests (i.e., the tl;dr version...)

```r
# t-test for H0: mu = 85
t.test(swiss$Fertility, mu = 85)
```

```
##
##  One Sample t-test
##
## data:  swiss$Fertility
## t = -8.154, df = 46, p-value = 1.755e-10
## alternative hypothesis: true mean is not equal to 85
## 95 percent confidence interval:
##  66.47485 73.81025
## sample estimates:
## mean of x
##  70.14255
```

Given this result, we can quite confidently reject H0 and conclude that the population mean of this fertility measure is very unlikely 85.