

TUTORIAL: SAMPLING, WEIGHTING AND ESTIMATION DAY 1

Stefan Zins, Matthias Sand
and Jan-Philipp Kolb

GESIS - Leibniz Institute
for the Social Sciences

GESIS Summer School

Cologne, Germany

August 24th, 2015

- Open Source
 - You can work with several datasets at the same time
 - You can create your own objects, functions and packages
 - Over 5,000 packages contributed by users available on CRAN
- Rapid implementation of new (scientific) developments
- Quick development of new tools that fit the user's demand

GETTING STARTED - DOWNLOAD R

The R Project for Statistical Computing

PCA 5 vars
prcomp(x = data, cor = var)

Clustering 4 groups

Factor 1 [41%]

Factor 3 [19%]

Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).
- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News:

- [The R Journal Vol.5/1](#) is available.
- [R version 3.0.1](#) (Good Sport) has been released on 2013-05-16.
- [R version 2.15.3](#) (Security Blanket) has been released on 2013-03-01.
- [useR! 2013](#) will take place at the University of Castilla-La Mancha, Albacete, Spain, July 10-12 2013. .

<https://www.r-project.org>



The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for data manipulation. Comments in German describe the steps: "1. alle auf 2-stellige", "2. alle auf 3-stellige", and "3. alle auf 4-stellige". The code uses functions like `unique`, `substr`, `read.table`, `write.csv`, `aggregate`, and `order` to process data from `Auswahl1` and `Untersuch2`.
- Environment:** Lists loaded data objects:
 - `Auswahl1`: 12811 obs. of 1 variable
 - `Auswahl2`: 121110 obs. of 1 variable
 - `tdta`: 48 obs. of 2 variables
 - `tdta2`: 105 obs. of 2 variables
 - `Tel.buch2`: 98587 obs. of 2 variables
 - `Untersuch`: 11869 obs. of 2 variables
 - `Untersuch2`: 4376 obs. of 3 variables
 - `Untersuch2`: 610 obs. of 3 variables
- Values:** Shows the structure of variables:
 - `ab`: int [1:610] 1 2 3 4 5 6 7 8 9 10...
 - `Auswahl1neu`: Large character [100310] elements...
 - `Auswahl2neu`: Large character [121110] elements...
 - `data2`: "J:/Work/Statistik/Sand/Mobilfun...
 - `drinne`: int [1:94213] 4 5 6 7 8 9 10 11...
 - `drinne2`: int [1:11259] 4 5 6 7 8 9 10 11...
 - `s`: int [1:10031] 22801 22802 22803...
- Console:** Shows the execution output of the code, including the creation of `tdta2` and `Untersuch2` objects.
- Markdown Quick Reference:** A sidebar panel providing a reference for creating documents and reports using R Markdown.

<https://www.rstudio.com>

- `<-` assignment operator
- `#` can be used to comment your script
- `x<-rnorm(10,0,1)` creates a vector with ten standardnormal-distributed values
- `mean(x)` calculates the mean of variable `x`; `length(x)` returns the number of observations in `x`

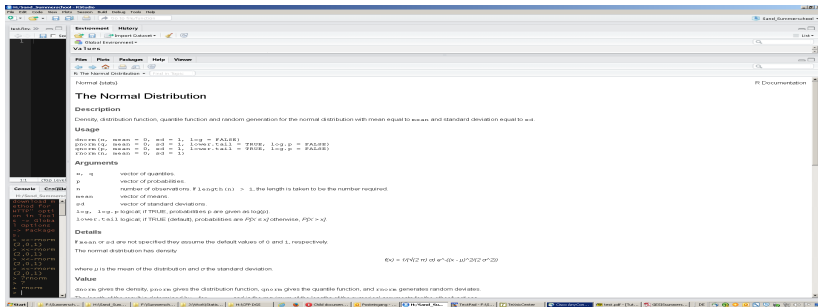
```
mean(x)
```

```
[1] -0.2370126
```

```
length(x)
```

```
[1] 10
```

- *?command*



- CRAN
- Quick-R
- stackoverflow.com

```
numeric    x<-c(1,2)
logical     x<-c(T,F)
character   x<-c("A","B")
factor      x<-as.factor(c("White","Black"))
```

`str()` returns the type of your data

```
x<-as.factor(c("White","Black"))
str(x)
```

```
Factor w/ 2 levels "Black","White": 2 1
```


Indexing a Vector:

```
A1<-c(1,2,3,4)
```

```
A1[1]
```

```
[1] 1
```

```
A1[1:3]
```

```
[1] 1 2 3
```

```
A1[-2]
```

```
[1] 1 3 4
```

Indexing a data frame:

```
A2<-4:1
```

```
AA<-cbind(A1,A2)
```

```
AA[1,]
```

```
A1 A2
```

```
1  4
```

```
AA[,1]
```

```
[1] 1 2 3 4
```

```
AA[1:3,2]
```

```
[1] 4 3 2
```

```
AA[,-1]
```

```
[1] 4 3 2 1
```

Indexing an array

```
A3<-array(1:8,c(2,2,2))
```

```
A3
```

```
, , 1
```

	[,1]	[,2]
[1,]	1	3
[2,]	2	4

```
, , 2
```

	[,1]	[,2]
[1,]	5	7
[2,]	6	8

```
A3[, ,2]
```

	[,1]	[,2]
[1,]	5	7
[2,]	6	8

Indexing a list

```
A4<-list(A1,c("Summer","Winter"))
```

```
A4
```

```
[[1]]
```

```
[1] 1 2 3 4
```

```
[[2]]
```

```
[1] "Summer" "Winter"
```

```
A4[[1]]
```

```
[1] 1 2 3 4
```

```
1:5
```

```
[1] 1 2 3 4 5
```

```
rep("A",times=10)
```

```
[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

```
rep(1:3,times=2,each=3)
```

```
[1] 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
```

```
seq(-5,5,by=2.5)
```

```
[1] -5.0 -2.5  0.0  2.5  5.0
```

Function	Distribution	Important parameter
<code>runif()</code>	Uniform distribution	n, min, max
<code>rnorm()</code>	Normal distribution	n, mean, sd
<code>rpois()</code>	Poisson distribution	n, lambda
...

Function	Meaning	Example
<code>length()</code>	Length	<code>length(x)</code>
<code>max()</code>	Maximum	<code>max(x)</code>
<code>min()</code>	Minimum	<code>min(x)</code>
<code>sd()</code>	Standard deviation	<code>sd(x)</code>
<code>var()</code>	Variance	<code>var(x)</code>
<code>mean()</code>	Mean	<code>mean(x)</code>
<code>median()</code>	Median	<code>median(x)</code>

These functions do only need one argument
Other functions need to be specified by further arguments:

<code>quantile()</code>	90% Quantile	<code>quantile(x,.9)</code>
<code>sample()</code>	Draw a sample	<code>sample(x,1)</code>

Function	Meaning	Example
<code>length()</code>	Length	<code>length(x)</code>
<code>max()</code>	Maximum	<code>max(x)</code>
<code>min()</code>	Minimum	<code>min(x)</code>
<code>sd()</code>	Standard deviation	<code>sd(x)</code>
<code>var()</code>	Variance	<code>var(x)</code>
<code>mean()</code>	Mean	<code>mean(x)</code>
<code>median()</code>	Median	<code>median(x)</code>

These functions do only need one argument
Other functions need to be specified by further arguments:

<code>quantile()</code>	90% Quantile	<code>quantile(x,.9)</code>
<code>sample()</code>	Draw a sample	<code>sample(x,1)</code>

- R is a modular program with many functions included in basic R

Function	Meaning	Example
<code>length()</code>	Length	<code>length(x)</code>
<code>max()</code>	Maximum	<code>max(x)</code>
<code>min()</code>	Minimum	<code>min(x)</code>
<code>sd()</code>	Standard deviation	<code>sd(x)</code>
<code>var()</code>	Variance	<code>var(x)</code>
<code>mean()</code>	Mean	<code>mean(x)</code>
<code>median()</code>	Median	<code>median(x)</code>

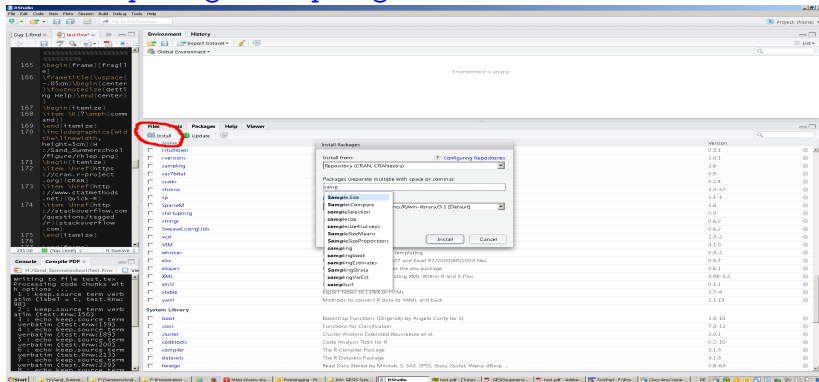
These functions do only need one argument
Other functions need to be specified by further arguments:

<code>quantile()</code>	90% Quantile	<code>quantile(x,.9)</code>
<code>sample()</code>	Draw a sample	<code>sample(x,1)</code>

- R is a modular program with many functions included in basic R
- more specific functions are embedded in further packages

INSTALLING AND LOADING PACKAGES

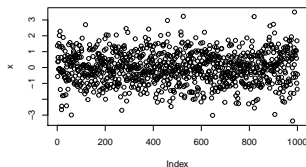
`install.package("sampling")`



`library(sampling)` or `require(sampling)`

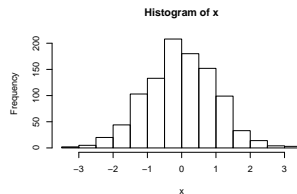
Library	Subject
<code>foreign</code>	reading and writing of data in numerous formats (e.g. <i>.dta</i> , <i>.sav</i>)
<code>sampling</code>	drawing and weighting samples
<code>survey</code>	analysis of complex survey samples
<code>xlsx</code>	read and write data in Excell-Format
<code>xtable</code>	export tables to LaTeX and HTML
<code>mice</code>	multiple imputation by chain equation
<code>reshape</code>	alter structure of datasets
<code>car</code>	applied regressions
<code>VIM</code>	visualization and imputation of Missing Values
<code>lattice</code>	high-level data visualization
<code>ggplot2</code>	grammar for graphics in R

```
set.seed(42)  
x <- rnorm(1000,0,1)  
plot(x)
```



`set.seed()` is used to specify a starting point

```
hist(x)
```



⇒ we will use 42 as seed-value for future exercises to obtain comparable results

`sample (base)`

R Documentation

Random Samples and Permutations

Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)
```

`x`: From what do we want
to sample ?



```
sample(x=1:10, n=1, replace=T)
```

`sample (base)`

R Documentation

Random Samples and Permutations


Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)
```

`n`: How many elements
do we want to draw?



```
sample(x=1:10, n=1, replace=T)
```

`sample {base}`

R Documentation

Random Samples and Permutations


Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)
```

Do we want to draw with
or without replacement?



```
sample(x=1:10, n=1, replace=T)
```

```
id <- 1:10000
set.seed(42)
education <- sample(c("none", "low", "average", "high"), 10000,
+                   replace = T, prob = c(.072, .356, .289, .283))
gender <- sample(c("male", "female"), 10000,
+               replace = T, prob = c(.488, .512))
iq <- rnorm(10000, 100, 20)
my.pop <- data.frame(id, gender, education, iq)
head(my.pop)
```

	id	gender	education	iq
1	1	male	high	123.26218
2	2	male	none	96.19531
3	3	male	low	94.21088
4	4	female	high	92.02308
5	5	male	average	114.18485
6	6	male	average	67.54705


```
summary(my.pop)
```

id	gender	education	iq
Min. : 1	female:5125	average:2851	Min. : 30.93
1st Qu.: 2501	male :4875	high :2820	1st Qu.: 86.50
Median : 5000		low :3588	Median :100.08
Mean : 5000		none : 741	Mean :100.02
3rd Qu.: 7500			3rd Qu.:113.60
Max. :10000			Max. :173.26

```
prop.table(table(my.pop$gender,my.pop$education))
```

	average	high	low	none
female	0.1449	0.1465	0.1844	0.0367
male	0.1402	0.1355	0.1744	0.0374

```
var(my.pop$iq)*(nrow(my.pop)-1)/nrow(my.pop)
```

```
[1] 406.1684
```

⇒ σ^2

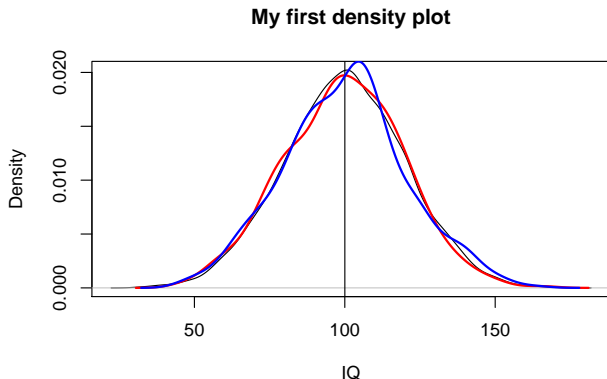
```
set.seed(42)
s.SRS <- sample(1:nrow(my.pop),500,replace=T)
s.SRSWOR <- sample(1:nrow(my.pop),500,replace=F)
my.samp.SRS <- my.pop[s.SRS,]
my.samp.SRSWOR <- my.pop[s.SRSWOR,]
summary(my.samp.SRS)
```

	id	gender	education	iq
Min.	: 3	female:257	average:132	Min. : 45.95
1st Qu.:	2322	male :243	high :134	1st Qu.: 85.38
Median	:4804		low :192	Median :100.00
Mean	:4896		none : 42	Mean : 99.60
3rd Qu.:	7434			3rd Qu.:113.20
Max.	:9966			Max. :165.63

```
nrow(unique(my.samp.SRS))
```

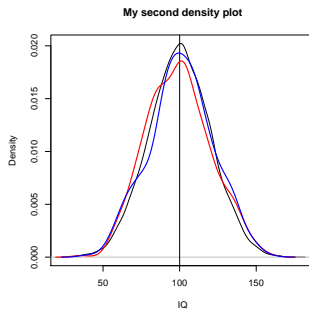
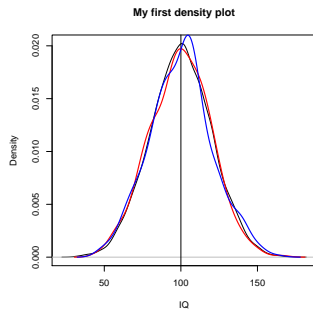
```
[1] 487
```

```
plot(density(my.pop$iq),main = "My first density plot"  
+      , xlab = "IQ")  
abline(v=mean(my.pop$iq), col = "black")  
lines(density(my.samp.SRS$iq),col = "red",lwd=2)  
lines(density(my.samp.SRSWOR$iq),col = "blue",lwd=2)
```



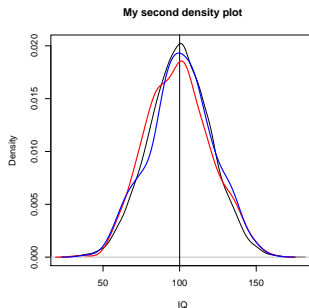
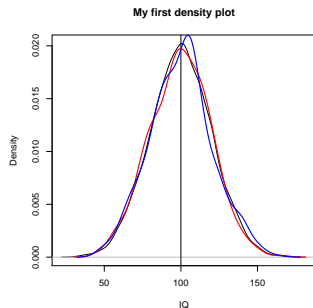
```
library(sampling)
set.seed(42)
s.SRS1 <- srswr(500,nrow(my.pop))
s.SRSWOR1 <- srswor(500,nrow(my.pop))
my.samp.SRS1 <- rbind(my.pop[s.SRS1!=0,]
+                      ,my.pop[s.SRS1>1,])
my.samp.SRSWOR1 <- my.pop[s.SRSWOR1==1,]
```

```
par(mfrow=c(1,2))
```



```
dev.off()
```

```
par(mfrow=c(1,2))
```



```
dev.off()
```

- should yield same results
- ⇒ routine differs in "starting point"

Declaring a working directory

```
path<-"H:/Sand_ Summerschool/Data Day1/"  
setwd(path)
```

- It is always useful to define and set your working directory at the beginning of each script
- `getwd()` displays you your current working directory
- `dir()` shows you all objects in a specific directory
- `ls()` lists all objects in your workspace
- `rm()` removes a object from your workspace

Example:

```
rm(list = ls())
```

Writing/ saving data and results

```
write.table(my.pop,"Synthetic Data Day1.csv",  
row.names = F, quote = F, dec = ".",sep = ",")
```

OR:

```
save(my.samp.SRS,s.SRS,my.samp.SRSWOR1,file = "Day1.Rdata")
```

⇒ See also: `write.csv` and `write.csv2` (`sep = ";"`)

Reading/ loading data and results

```
d1 <- read.table("Synthetic Data Day1.csv",  
header = F, dec = ".",sep = ",")
```

OR:

```
load("Day1.Rdata")
```


SAMPLE SIZES

- 1 Generate 1000 numbers from a exponential distribution
- 2 Draw three samples($n_1=2, n_2=10, n_3=100$)
- 3 Plot the density and add the means of the three samples as vertical lines

BELGIAN MUNICIPALITIES/ VARIANCE DECOMPOSITION

- 1 Load the dataset "belgianmunicipalities" from the `sample`-package using the `data()`-command
- 2 Inspect the structure of the dataset
- 3 Calculate mean and variance of the variable `averageincome`
- 4 Calculate the mean of each province for that variable, plot your results and add the mean of 3
- 5 Recalculate the mean of `averageincome` based on the means by province and compare your results
- 6 Make a boxplot of the variable `averageincome` for each province
- 7 Calculate the variance of `averageincome` using variance decomposition and compare it with 3
 - ⇒ Advice: consider the dataset as the aggregates for the whole population and use the formula for the population variance