

Topic 2

R basics

Sergey Mastitsky ©

Klaipeda, 28-30 September 2011

2. R Basics

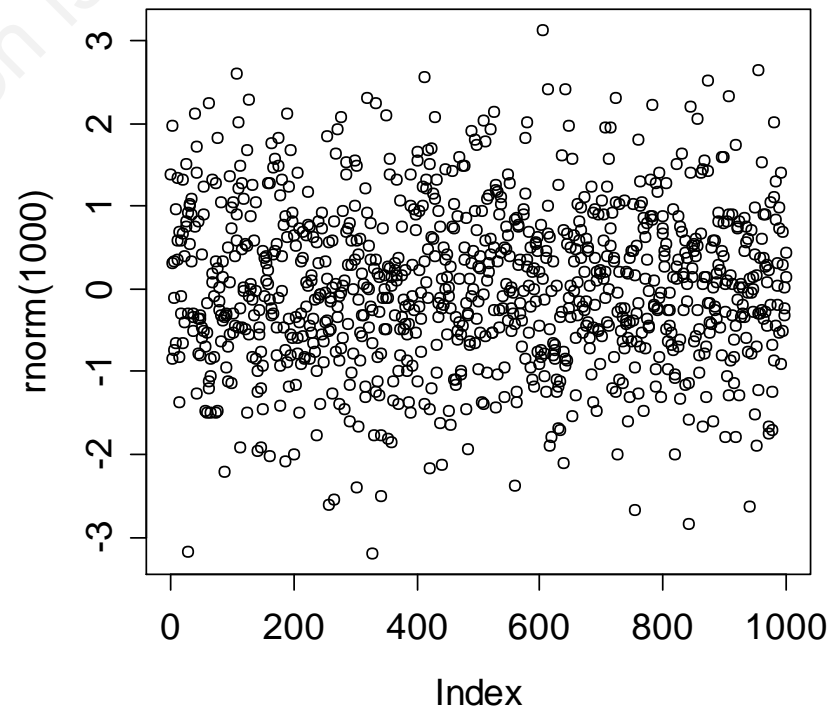
2.1. First steps

The command line

- R works by the question-and-answer model
- Commands are entered at the prompt sign in the command line, e.g.:

```
> plot(rnorm(1000))
```

1000 numbers drawn from a normal distribution



R is a “calculator on steroids”

```
> 2 + 2
[1] 4
> exp(-2)
[1] 0.1353353
> rnorm(10)
[1] -0.505 1.728 -0.323
[4] -1.900 1.808 -1.898
[7] -1.108 -1.215 1.092
[10] -0.759
```

The assignment operator <-

```
> x <- 2
```

```
> x
```

```
[1] 2
```

```
> x + x
```

```
[1] 4
```

```
> y <- 5
```

```
> x * y
```

```
[1] 10
```

Naming rules:

- Names can be built from any symbols, but:
- No variable names starting with dot “.” and no spaces in names
- No names starting from numbers
- Names are case sensitive, e.g. `WT` and `wt` would be treated as different
- Don't use system-reserved names, i.e.
 - `c`, `q`, `t`, `C`, `D`, `F`, `I`, and `T`
 - `diff`, `df`, `pt`, and some other names

Vectors – sets of values of the same type

- R can handle data vectors as single objects:

```
> weight <- c(60, 72, 57, 90, 95, 72)
> weight
[1] 60 72 57 90 95 72
```

`c(...)` – concatenation function

- One can do calculations with vectors just like ordinary numbers:

```
> height <- c(1.75, 1.80, 1.65, 1.90,
              1.74, 1.91)
> bmi <- weight/height^2
> bmi
[1] 19.592 22.2222 20.937 24.931 31.378 19.736
```

Using vectors, we can easily perform some typical statistical calculations

- Calculating average weight:

```
> xbar <- sum(weight) / length(weight)
> xbar
[1] 74.333
```

- Calculating the standard deviation:

$$SD = \sqrt{\sum (x_i - \bar{x})^2 / (n - 1)}$$

```
> SD <- sqrt(sum((weight - xbar)^2) /
              (length(weight) - 1))
> SD
[1] 15.422
```


Of course, we could've calculated the mean and SD much faster!

```
> mean(weight)
[1] 74.33333
```

```
> sd(weight)
[1] 15.42293
```

2. R Basics

2.2. R language essentials

For your personal use only.
Public presentation is not allowed

Expressions

- What we enter into the command line has a more general name – **expression**
- Expressions typically include variable names, operators, and function calls:

```
xbar <- sum(weight) / length(weight)
```

function



operator

variable name

Objects

- Expressions work on **objects**
- Basically, everything we create in R can be called “object”

`x`, `y`, `weight`, `height`, `SD` – all are objects

R is one of the object-oriented languages

Types of R objects

- **Data objects** – contain data (i.e. vectors, matrices, lists, and data frames)
- **Function objects** - functions

2.2. R language essentials

Functions

For your personal use only.
Public presentation is not allowed

Functions and arguments

- **Function** – a named set of commands that perform specific operation(s) with other objects

`plot()`, `c()`, `sum()`, `mean()` ...

- **Actual function arguments** – objects that a function is applied to

`plot(height, weight)`

- **Formal function arguments** – specific parameters that control function's behavior

`plot(height, weight, pch = 2)`

Parentheses in a function call

- Must be used at all times, even though there may be no arguments in the function call

Typical example: `ls ()`

2.2. R language essentials

Vectors

For your personal use only.
Public presentation is not allowed

Vectors – sets of objects of the same type

■ Numeric

```
> c(2, 7, 10)
[1] 2 7 10
```

■ Character

```
> c("Treatment 1", "Treatment 2")
[1] "Treatment 1" "Treatment 2"
```

■ Logical: take the value TRUE or FALSE

```
> b <- bmi > 25
> b
[1] FALSE FALSE FALSE FALSE TRUE FALSE
```

Missing values

- Often occur in practical data analysis
- In R such values are denoted as NA

```
> c(2, NA, 10)
[1] 2   NA  10
```

- Operations on NA yield NA, and thus in many cases NAs have to be excluded from calculations (...discussed later)

```
> mean(c(2, NA, 10))
[1] NA
```

Functions that create vectors: `c()`

- Examples of concatenation:

```
> c(42, 57, 12, 39, 1, 3, 4)
```

```
[1] 42 57 12 39 1 3 4
```

```
> x <- c(1, 2, 3)
```

```
> y <- c(10, 20)
```

```
> c(x, y)
```

```
[1] 42 57 12 39 1 3 4 10 20
```

```
> c("abc", 10, 20)
```

```
[1] "abc" "10" "20"
```

Functions that create vectors: `seq()`

- A numeric sequence:

```
> seq(from = 4, to = 10)
```

```
[1] 4 5 6 7 8 9 10
```

- Command with the same outcome:

```
> 4:10
```

- Sequence with step 2:

```
> seq(from = 4, to = 10, by = 2)
```

```
[1] 4 6 8 10
```

Functions that create vectors: `rep()`

- Repeated pattern:

```
> z <- c(2, 4, 6)
```

```
> rep(z, 2)
```

```
[1] 2 4 6 2 4 6
```

- Try also these commands:

```
> rep(z, 1:3)
```

```
> rep(1:2, c(3, 6))
```

```
> rep(1:2, each = 10)
```

2.1. R language essentials

Matrices

For your personal use only.
Public presentation is not allowed

Matrices

- **Matrix** – two-dimensional array of numbers
- Widely used in theoretical and practical statistics
- Can be used to store data

Creating matrices with `matrix()`

■ Creating a matrix

```
> h <- matrix(1:12, nrow=3, byrow=TRUE)
> h
```

■ Assigning names to rows and columns:

```
> rownames(h) <- c("A", "B", "C")
> colnames(h) <- c("V1", "V2", "V3", "V4")
> h
```

■ Matrix transposition:

```
> t(h)
```

Creating matrices by combining vectors

- Binding several vectors by columns:

```
> cbind(A = 1:4, B = 5:8, C = 9:12)
```

- Binding several vectors by rows:

```
> rbind(A = 1:4, B = 5:8, C = 9:12)
```

2.2. R language essentials

Factors

Factors

- **Factor** – categorical variable indicating some subdivision of data (e.g., by sex, sampling location, treatment group, etc.)
- **Factor levels** – specific categories of a factor (e.g., male and female)

Creating a factor

```
> sex <- c("male", "male",  
  "female", "female", "female")  
> sex <- factor(sex)  
> sex  
[1] male male female female female  
Levels: female male  
> levels(sex)  
[1] "female" "male"  
> as.numeric(sex)  
[1] 2 2 1 1 1
```

Creating a factor

Also works, but better not to use:

```
> sex <- c(0, 0, 1, 1, 1)
```

```
> sex <- factor(sex)
```

```
> sex
```

```
[1] 0 0 1 1 1
```

```
Levels: 0 1
```

2.2. R language essentials

Lists

For your personal use only.
Public presentation is not allowed

Lists

- **List** – collection of R objects of any type

```
> sex <- c("male", "male",  
  "female")  
> weight <- c(80, 85, 60)  
> height <- c(1.75, 1.80, 1.68)  
> data <- list(Sex = sex,  
  Weight = weight,  
  Height = height)
```


Lists

```
> data
```

```
$Sex
```

```
[1] "male" "male" "female"
```

```
$Weight
```

```
[1] 80 85 60
```

```
$Height
```

```
[1] 1.75 1.80 1.68
```

Extraction of list components

```
> data$Sex  
[1] "male" "male" "female"
```

```
> data$Weight  
[1] 80 85 60
```

and so on...

2.2. R language essentials

Data frames

For your personal use only.
Public presentation is not allowed

Data frames

- **Data frame** – a list of vectors and/or factors of the same length that are related “across” such that data in the same position come from the same experimental unit (subject, animal, etc.). In addition, it can have a unique set of row names

Creating a data frame

```
> data <- data.frame(  
  Sex = sex,  
  Weight = weight,  
  Height = height)  
> data
```

	Sex	Weight	Height
1	male	80	1.75
2	male	85	1.80
3	female	60	1.68

Extracting data frame components

```
> names(data)
[1] "Sex" "Weight" "Height"
> data$Sex
[1] male male female
Levels: female male
> data$Height
[1] 1.75 1.80 1.68
```

2.2. R language essentials

Indexing

For your personal use only.
Public presentation not allowed

Indexing of vectors

- Extracting particular values from a vector:

```
> weight[2]
```

```
[1] 85
```

```
> height[c(1, 3)]
```

```
[1] 1.75 1.68
```

```
> i <- c(2, 3)
```

```
> weight[i]
```

```
[1] 85 60
```

```
> height[-i]
```

```
[1] 1.75
```


Indexing of vectors

■ Conditional selection

```
> height[height < 1.80]
```

```
[1] 1.75 1.68
```

```
> weight[weight>=80 & weight<=85]
```

```
[1] 80 85
```

Other logical operators to use:

| (logical “or”)

! (logical “not”)

Indexing of data frames

- Value from the 2nd row and 3rd column:

```
> data[2, 3]  
[1] 1.8
```

- Entire 2nd row:

```
> data[2, ]  
  Sex  Weight Height  
2 male   85    1.8
```

- Entire 3rd column:

```
> data[, 3]  
[1] 1.75 1.80 1.68
```

Indexing of data frames

- All data for males:

```
> data[data$Sex == "male", ]
```

	Sex	Weight	Height
1	male	80	1.75
2	male	85	1.80

```
> data$Sex == "male"
```

```
[1] TRUE TRUE FALSE
```

Indexing of data frames

- Data from the 1st and 3rd columns:

```
> data[, c(1, 3)]
```

	Sex	Height
1	male	1.75
2	male	1.80
3	female	1.68

Indexing of data frames

- Very useful for large data frames:

```
> head(data, n = 2)
```

	Sex	Weight	Height
1	male	80	1.75
2	male	85	1.80

```
> tail(data, n = 2)
```

```
...
```

2. R Basics

2.3. The R environment

For your personal use only.
Public presentation is not allowed

2.3. The R environment

The workspace

The workspace

- **Workspace** – part of the computer memory where all the objects created during a R session are stored (temporarily, until saved!)
- Checking what's in the workspace:

```
> ls ()
```

```
...
```

In **RStudio**, check the tab “**Workspace**”

The workspace

- When it gets messy, tidy up!

```
> rm(i)
```

- To remove everything (! careful):

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

In **RStudio**:

tab “**Workspace**”, button “**Clear All**”

The workspace

- To save the current workspace for later usage (creates an `.Rdata` file in the current *working directory*):

```
> save.image()
```

- A better approach (in **RStudio**):
 - Tab “**Workspace**”
 - Option “**Save Workspace As...**”

The working directory

- **Working directory** – path where R automatically goes to load files from and to save files to

- Checking the current working directory:

```
> getwd()
```

```
[1] "D:/DOCS"
```

- Setting up a working directory:

```
> setwd("D:/DOCS/Analysis")
```

or in RStudio:

Tools/Set Working Directory/Choose Directory

2.3. The R environment

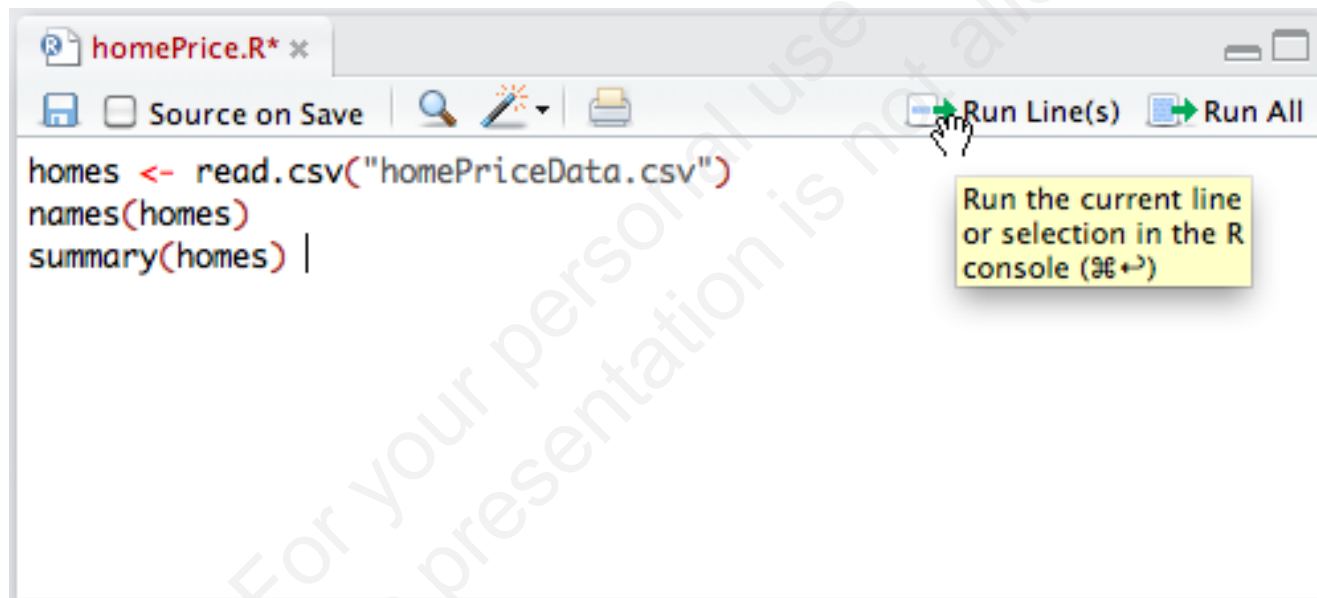
Scripting

For your personal use only.
Public presentation is not allowed

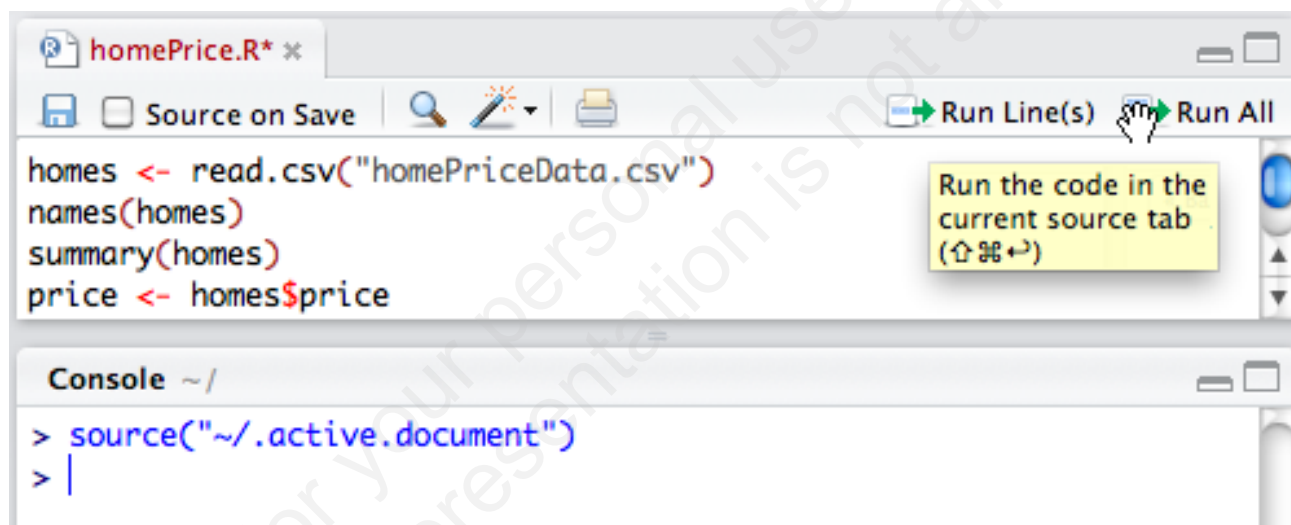
Scripting

- After a certain level of complexity, it's better to have R working line by line from a **script** (i.e., a collection of commands)
- Option 1: function **source()** – takes a file with commands as its input and runs them
- Option 2: **script editor** window – allows users to submit one or several code lines to R at a time

Executing a single line from the script editor in RStudio



Executing all lines from the script editor in RStudio



2.3. The R environment

Getting help

For your personal use only.
Public presentation is not allowed

Getting help

- R has extensive online help:

```
> help.start()
```

or go to **Help** tab in **RStudio**

- Getting help on a specific function:

```
> help(plot)    # or equivalently
```

```
> ?plot
```

- Getting help on a specific topic:

```
> help.search("correlation")
```

2.3. The R environment

Packages

Packages

- **Package** – a collection of functions designed for specific operations, or collection of data sets. Go to **Packages** tab in **RStudio** to check what's available
- To load a (already installed) package for work:
> `library(package)`
- To install a new package (needs live Internet connection):
> `install.packages("package")`

We will use a few extra packages

```
> install.packages("ISwR")  
> install.packages("HSAUR2")
```

For your personal use only.
Public presentation is not allowed.

CRAN Task Views



The Comprehensive R Arch x +

← → ↺ ⬆️ cran.r-project.org ☆ 🔍



CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
gR	gRaphical Models in R
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods

CRAN

- [Mirrors](#)
- [What's new?](#)
- [Task Views](#)
- [Search](#)

About R

- [R Homepage](#)
- [The R Journal](#)

Software

- [R Sources](#)
- [R Binaries](#)
- [Packages](#)
- [Other](#)

Documentation

- [Manuals](#)
- [FAQs](#)
- [Contributed](#)

Author: Sergey Mastitsky

http://www.inside-r.org/packages

Package reference: A | insid x

www.inside-r.org/packages

log in rss

inside-R Blogs R Language **R Packages** How to Questions Pretty R Get R R Consultants

A Community Site for R – Sponsored by Revolution Analytics

Search... GO

Home » Package Reference

Package reference: A

[View all packages on one page](#)

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

Package	Description
aaMI	Mutual information for protein sequence alignments
abc	Tools for Approximate Bayesian Computation (ABC)
abd	The Analysis of Biological Data
abind	Combine multi-dimensional arrays
AcceptanceSampling	Creation and evaluation of Acceptance Sampling Plans
ACCLMA	ACC & LMA Graph Plotting
accuracy	Tools for testing and improving accuracy of statistical results

Search Packages

GO

About Package Reference

This is an index of user-contributed packages for R, from [cranastic.org](#). Follow the links within each package to rate and review packages you use.

To install and use a package, use the [install.packages](#) function or the Packages menu within R. You can also download packages directly from the [CRAN](#) network of mirrors.

About inside-R.org

inside-R.org is a collection of resources about the open-source [R Project](#) for the R Statistics Community. This site is sponsored by Revolution Analytics.

Author: Sergey Mastitsky

Exercises

- Go to the [CRAN Task Views](#) and explore what's available
- Go to the [Inside-R](#) website and explore the list of packages