The Seventh International Conference on Internet Technologies & Applications - ITA17

Tuesday 12 - Friday 15 September 2017

Wrexham, North Wales, UK

An Introduction into Statistical Computing with R

Andreas Schmidt[1,2] and Steffen G. Scholz[2]

(1)
Department of Informatics and
Business Information Systems
University of Applied Sciences Karlsruhe
Germany

(2)
Institute for Applied Computer Sciences
Karlsruhe Institute of Technologie
Germany

Resources available

http://www.smiffy.de/ita-2017/ [1]

- Slideset
- Exercises
- Example datasets

1. all materials copyright, 2017 by andreas schmidt

Outlook

- What is R?
- Basic-Operations
- Basic Data Structures
  - Getting information about your data
  - Operations on data
  - Import and Export
  - Control Structures
  - User defined functions
- R and Big Data Applications

+ 3 hands on exercices

- Working with vectors
- Data frames and databases
- Simple NLP example

Timetable

• Part I - Introduction & vector datatype:	20 min.
• Hands-on-exercise:	10 min.
• Part II - Data-Frames & import from extrnal sources:	20 min.
• Hands-on-exercise:	10 min.
• Part III - Graphics & control structures:	20 min.
• Hands-on-exercise:	10 min.

Characteristics of R

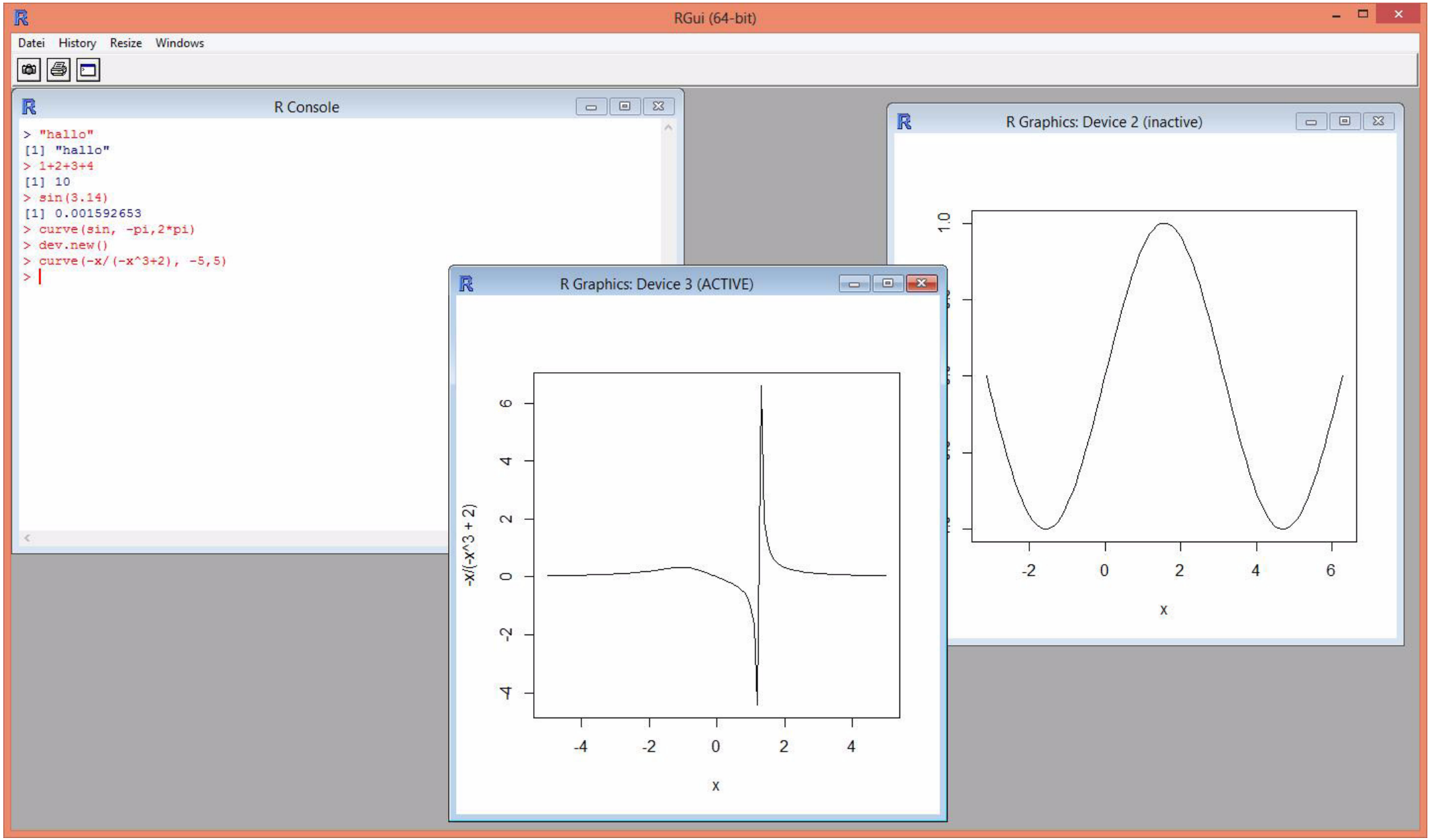
- Programming language/development environment for (statistical) data analysis
- Open Source project (gnu, cross plattform, high number of additional packages[1], huge development community)

- Interpreted language
- Main memory based
- Interface to C/C++, Fortran and Java
- Very good graphic cababilities
- Interactive and batch processing (see next slides ...)
- General programming language
- Easily extensible
- Leading edge algorithms

1. ~8500 packages (13.6.2016)

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES



R Basic Datastructures

- **Vector**
- Matrices
- Array
- (Lists)
- **Data Frames**

Vector

- Basic datatype (there exists no scalar values)
- One dimensional data structure
- All elements must be of same type
- Example:

```
x<-c(-0.1, 1.5, 2, 0)

sentence <- c("Data", "Scientist", "The", "Sexiest", "Job", "of",
              "the", "21st", "Century")

filter<-c(TRUE, FALSE, FALSE, TRUE)

x_values<-seq(-10, 10, by=0.1)
```

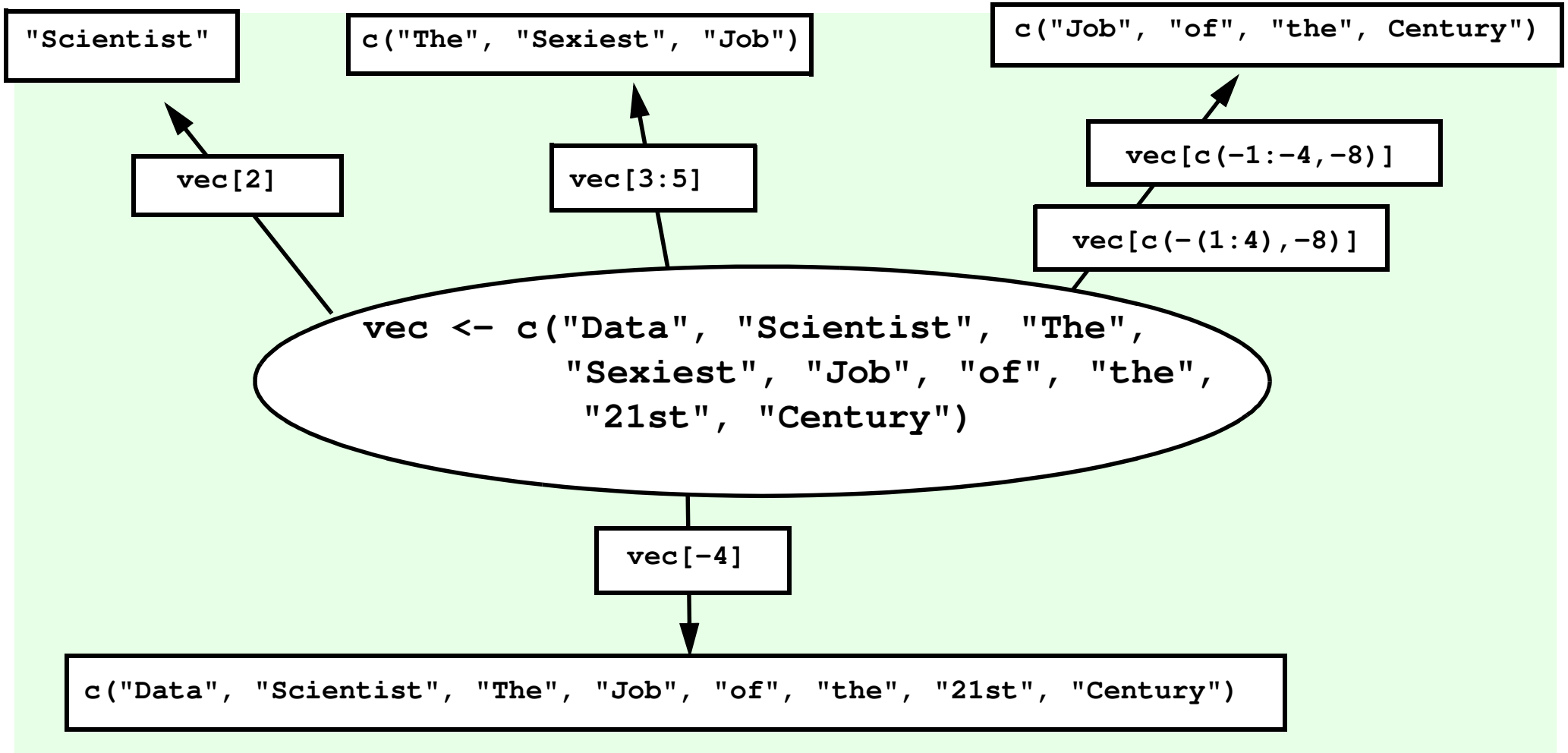
- Access (first element has index 1):

```
print(x[1])                    # result:-0.1
a_cool_job<-sentence[2]        # "Scientist"
a_statement<-sentence[3:5]     # "The" "Sexiest" "Job"
x[2] = 3                       # change second value from 1.5 to 3
```

Vector Operations

- Generating vectors
  - c(...) - Constructor
  - „:“-operator (i.e. 1:10)
  - seq(from=1, to=1, by=.., length=...), rep(x, time)
- misc operations
  - length(vec)
  - +, -, *, /, ^, %/%, %%, >, < operator (operates on each element)
- Retrieving subranges:
  - vector[index]: Retrieve element at position ’index’]
  - vector[vector_indices]: Retrieve the elements at positions ’vector_indices’
  - vector[-index]: Retrieve all elements except at index ’index’
  - vector[-vector_indices]: Retrieve all elements except at positions ’vector_indices’

Accessing Vectors



Filter

```
> numbers<-1:5
> numbers
[1] 1 2 3 4 5
```

- Boolean Filter

```
> numbers[c(TRUE, FALSE, FALSE, FALSE, TRUE)]
[1] 1 5

> numbers[c(T, F)]     # same as numbers[c(TRUE, FALSE, TRUE, FALSE, TRUE)]
[1] 1 3 5
```

- Vectorized Comparision operator

```
> numbers < 4
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

- Combination of both Techniques

```
> numbers[numbers < 4]
[1] 1 2 3
```

Filtering with grep

```
> strings <- c('CCAA','CCAA','AAGT','CAGT','TCCT','CGCT',
               'ATGT','AACA','CACA','TCTT','GGCT')
> grep('^(\\w)\\1', strings)
[1]  1  2  3  8  9 11
>

> strings[grep('^(\\w)\\1', strings)]
[1] "CCAA" "CCAA" "AAGT" "AACA" "GGCT"
```

Vector Element Names

- Vector elements can have names (additionaly to index-position)
- Examples:

```
> named_vector <- 1:5
> named_vector
[1] 1 2 3 4 5
> names(named_vector) <- c("one", "two", "three", "four", "five")
> named_vector
  one   two three  four  five
    1     2     3     4     5
> names(named_vector)
[1] "one"   "two"   "three" "four"  "five"
> named_vector["two"]
two
  2
> named_vector[c("three", "five")]
three  five
    3     5
> unname(named_vector)
[1] 1 2 3 4 5
```

Adding/deleting elements to a vector

```
> a<-10:5
> a
[1] 10 9 8 7 6 5
```

- Append a scalar

```
> a<-c(a,4)
> a
[[1] 10  9  8  7  6  5  4
>
```

- Append a vector

```
> a<-c(a, 3:1)
> a
[1] 10  9  8  7  6  5  4  3  2  1
```

- Delete an element at position:

```
> a<-a[-2]
>
> a
[1] 10  8  7  6  5  4  3  2  1
```

Vectorized functions

- Remember: Basic datatype is a vector
- Most functions also accept a vector as input:
- Examples:

```
> sqrt(9)
[1] 3
> x <- 1:10
> sqrt(x)
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000
[10] 3.162278
>

> x*-1
 [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
```

Set Operations with Vectors

- Union

```
> union(c(1,2), c(2,3))
[1] 1 2 3
```

- Intersection

```
> intersect(c(1,2), c(2,3))
[1] 2
```

- Difference

```
> setdiff(c(1,2), c(2,3))
[1] 1
```

- Test equality

```
> setequal(c(1,2), c(2,3))
[1] FALSE
> setequal(c(1,2), c(2,1))
[1] TRUE
```

- Test if value is in Set

```
> 2 %in% c(1,4,6)
[1] FALSE
> 4 %in% c(1,4,6)
[1] TRUE
```

Sort & Order

```
> unordered<-c(-3,7,5,9,-2,8,2)
> unordered
[1] -3  7  5  9 -2  8  2
```

- Sort Elements in Vector

```
> sort(unordered)
[1] -3 -2  2  5  7  8  9
```

- Sort in reverse order

```
> sort(unordered, decreasing=TRUE)
[1]  9  8  7  5  2 -2 -3

# short version of above
sort(unordered, dec=T)
```

- Change the ordering in a vector

```
unordered[c(1,5,7,3,2,6,4)]
[1] -3 -2  2  5  7  8  9
```

- Return the ordering index

```
> orderedIdx<-order(unordered)
> orderedIdx
[1] 1 5 7 3 2 6 4
```

Frequency tables

```
> colors<-c('red','green','red','red','green','yellow')
```

- Calculate the frequency of different colors

```
> freq<-table(colors)
> freq
colors
 green    red yellow
     2      3      1
```

- Datatype table

```
> str(freq)
 'table' int [1:3(1d)] 2 3 1
 - attr(*, "dimnames")=List of 1
  ..$ colors: chr [1:3] "green" "red" "yellow"
```

- Conversion to vector

```
> as.vector(freq)
[1] 2 3 1
> names(freq)
[1] "green"  "red"    "yellow"
```

Some useful functions

- Generate random numbers

```
rnorm(1000)
```

- calculate the sum of the vector elements

```
sum(vector)
```

- calculate the mean

```
mean(vector)
```

- Calculate the median

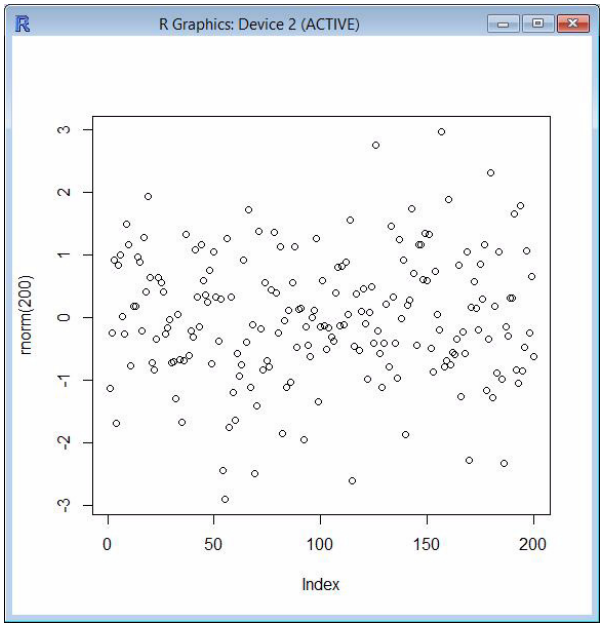
```
median(vector)
```

- standard deviation

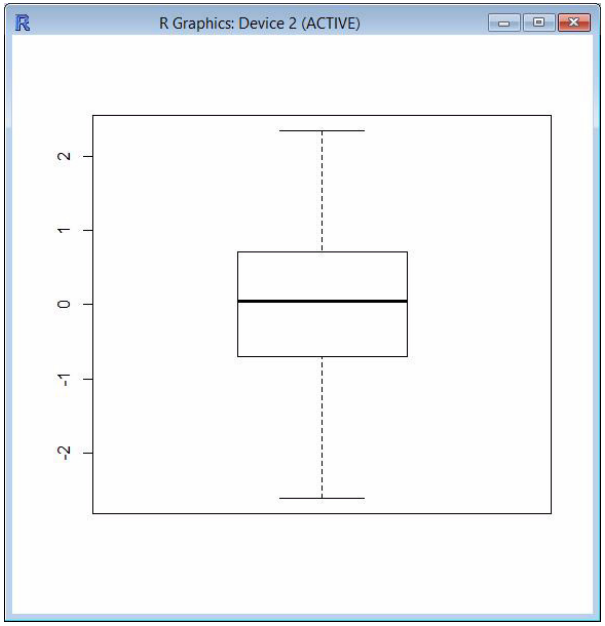
```
sd(vector)
```

Visualisation

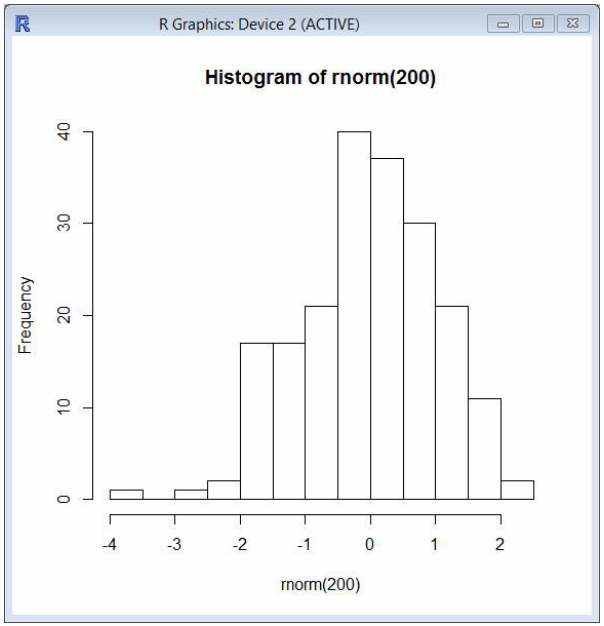
- plot(rnorm(200))



- boxplot(rnorm(200))



- hist(rnorm(200))



Hands-on-Exercise I - Vectors

Matrices

- Two dimensional array
- numeric/character/logical data (alle elments must be from the same type)
- Syntax:

```
a_matrix<-matrix(vector, nrow=..., ncol=..., byrow=FALSE/TRUE,
                 dimnames=list(rowname_vec, colname_vec))
```

- Examples:

```
> m1<-matrix(1:8, nrow=2)
> m1
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

```
> m2<-matrix(1:8, nrow=4)
> m2
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Matrices

```
> ticTacToe<-matrix(rep(0,9), nrow=3)
> ticTacToe
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
>
> ticTacToe[2,2] = 1
> ticTacToe
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    1    0
[3,]    0    0    0
```

```
> ticTacToe[3,] = 2
> ticTacToe
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    1    0
[3,]    2    2    2

> ticTacToe[,1] = 3
> ticTacToe
     [,1] [,2] [,3]
[1,]    3    0    0
[2,]    3    1    0
[3,]    3    2    2
```

Matrix Row and Column Names

```
> rownames(ticTacToe) <- c('A','B','C')
> colnames(ticTacToe) <- c('I','II','III')
> ticTacToe
  I II III
A 3  0   0
B 3  1   0
C 3  2   2
>
> ticTacToe["A","II"]
[1] 0
> ticTacToe["A",]
  I  II III
  3   0   0
> ticTacToe[,"II"]
A B C
0 1 2
> rownames(ticTacToe)
[1] "A" "B" "C"
> colnames(ticTacToe)
[1] "I"   "II"  "III"
```

Combining Matrices/Vectors

```
> m1 <- matrix(c(11,12,21,22), nrow=2)
> m2 <- matrix(c(31,32,31,32), nrow=2)
```

- Adding columns (cbind)

```
> cbind(m1, m2)                    # number of rows must match
     [,1] [,2] [,3] [,4]
[1,]   11   21   31   31
[2,]   12   22   32   32
```

- Adding rows (rbind):

```
> rb <- rbind(m1, m2)              # number of columns must match
> rb
     [,1] [,2]
[1,]   11   21
[2,]   12   22
[3,]   31   31
[4,]   32   32
> nrow(rb)
[1] 4
> ncol(rb)
[1] 2
```

Example: Deleting rows/columns from a matrix

```
> rb
     [,1] [,2]
[1,]   11   21
[2,]   12   22
[3,]   31   31
[4,]   32   32

> # remove first two rows:
> rb <- rb[3:4,drop=FALSE]          # drop=FALSE to prevent dimension reduction
> rb
     [,1] [,2]
[1,]   31   31
[2,]   32   32
>
> # remove last column
> rb <- rb[,1,drop=FALSE]
> rb
     [,1]
[1,]   31
[2,]   32
>
```

Filtering matrices (row/columnwise)

```
> m <-matrix(round(rnorm(24)), nrow=4)
>
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1   -2    1    0    0
[2,]   -1   -1   -1   -1    1    1
[3,]    0    1    2    0    0   -2
[4,]   -1    1    0    1   -1    1
>
> #    rowwise
> m[,4] == 1
[1]  TRUE FALSE FALSE  TRUE
>
> m[m[,4] == 1,]
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1   -2    1    0    0
[2,]   -1    1    0    1   -1    1
```

```
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1   -2    1    0    0
[2,]   -1   -1   -1   -1    1    1
[3,]    0    1    2    0    0   -2
[4,]   -1    1    0    1   -1    1
>
> #    columnwise
> m[1,] == 0
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
>
> m[, m[1,] == 0]
     [,1] [,2]
[1,]    0    0
[2,]    1    1
[3,]    0   -2
[4,]   -1    1
>
```

Array

- Like matrix but can have more than 2 dimensions
- Syntax:

```
myarray<-array(vector, dimensions, dimnames)
```

- Examples:

```
myarray < -array(1:27, c(3,3,3), list(c("a1","a2","a3"),
                                      c("b1", "b2", "b3"),
                                      c("c1", "c2", "c3")))
>
>
> myarray
, , c1                    , , c2                    , , c3

   b1 b2 b3                  b1 b2 b3                  b1 b2 b3
a1  1  4  7               a1 10 13 16               a1 19 22 25
a2  2  5  8               a2 11 14 17               a2 20 23 26
a3  3  6  9               a3 12 15 18               a3 21 24 27
```

Data-Frame

- Like a matrix, but can contain different types (numeric, character, ...) of data
- Most common datastructure in R
- Syntax:

```
myframe <- data.frame(col1, col2, col3, ...)
```

- Example:

```
> PersonID<-c(101,102,103)
> name<-c("Klaus", "Ingo", "Tanja")
> age<-c(31,27,29)
> dataset<-data.frame(PersonID, name, age)
> dataset
    PersonID  name age
1        101 Klaus  31
2        102  Ingo  27
3        103 Tanja  29
```

Data-Frame: Acess methods

```
> str(dataset)
'data.frame':   3 obs. of  3 variables:
 $ PersonID: num  101 102 103
 $ name    : Factor w/ 3 levels
             "Ingo","Klaus",..: 2 1 3
 $ age     : num  31 27 29

> dataset[2,c(1,2,3)]
    PersonID name age
2        102 Ingo  27
> dataset[2,]
    PersonID name age
2        102 Ingo  27

> dataset[2,2]
[1] Ingo
```

```
> dataset[c(1,2,3),2]
[1] Klaus Ingo  Tanja

> dataset[,2]
[1] Klaus Ingo  Tanja

> dataset$age
[1] 31 27 29
>

> dataset[c("name","age")]
   name age
1 Klaus  31
2  Ingo  27
```

Filtering

- Examples:

```
> dataset
  PersonID  name age
1      101 Klaus  31
2      102 Ingo  27
3      103 Tanja  29
> dataset$age < 30
[1] FALSE  TRUE  TRUE
> dataset[dataset$age < 30,c('PersonID','name','age')]
  PersonID  name age
2      102  Ingo  27
3      103 Tanja  29
>
> dataset[dataset$age < 30,]
  PersonID  name age
2      102  Ingo  27
3      103 Tanja  29

> dataset[dataset$age < 30 & dataset$age > 28, c("name")]
[1] Tanja
```

Data Frame filtering: Comparison to SQL

- SQL:

```
select name, age
  from dataset_table
 where age < 30
   and age > 28
```

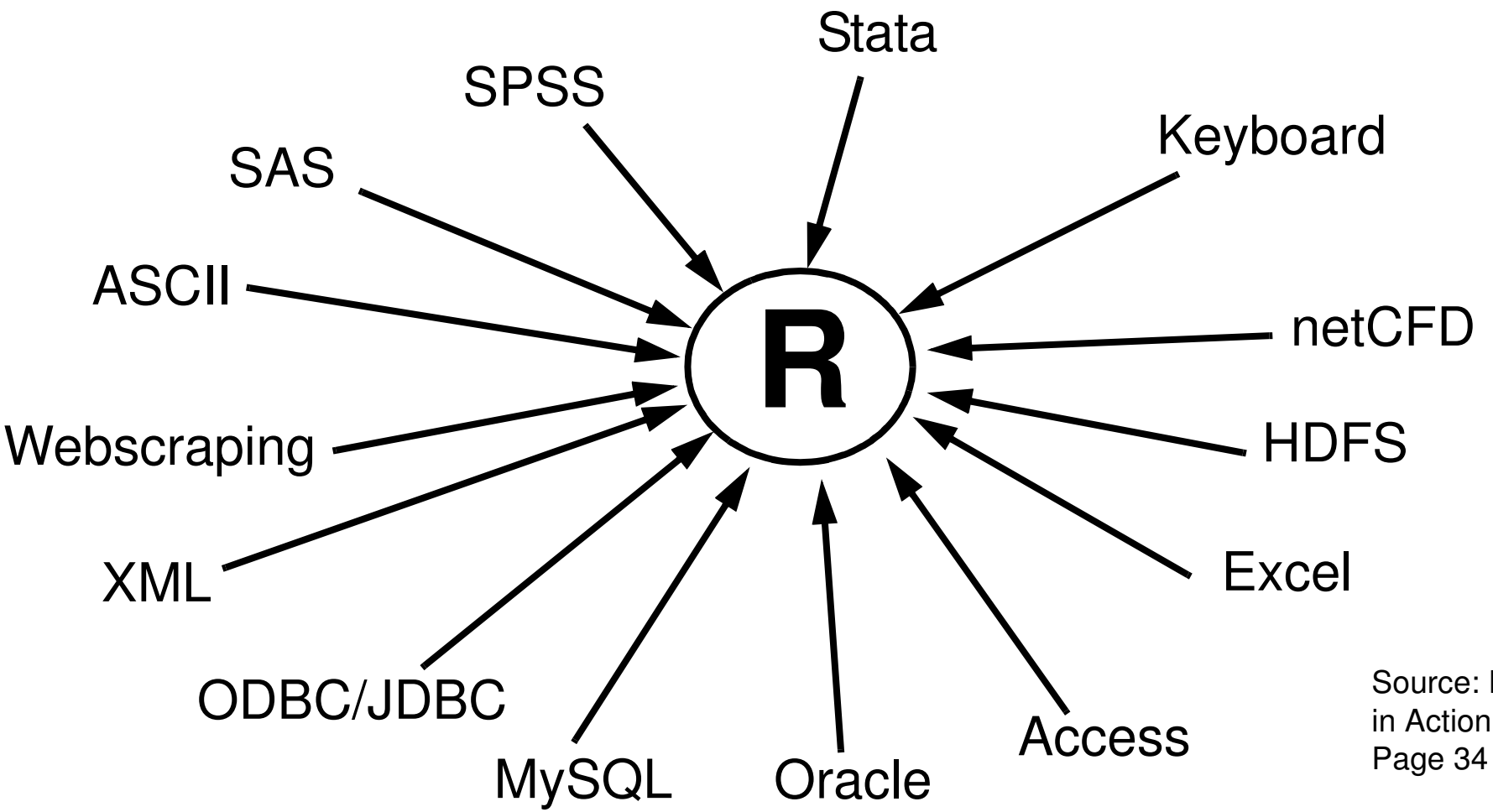
- R

```
dataset[dataset$age < 30 &
        dataset$age > 28,
        c("name","age")]
```

selection

projection

Data Import in R



Source: Robert Kabacoff, R in Action, Manning, 2011, Page 34

city.tsv

Name	Country	Province	Population	Longitude	Latitude
Aachen	D	"Nordrhein Westfalen"	247113	NULL	NULL
Aalborg	DK	Denmark	113865	10	57
Aarau	CH	AG	NULL	NULL	NULL
Aarhus	DK	Denmark	194345	10.1	56.1
Aarri	WAN	Nigeria	111000	NULL	NULL
Aba	WAN	Nigeria	264000	NULL	NULL
Abakan	R	"Rep. of Khakassiya"	161000	NULL	NULL
Abancay	PE	Apurimac	NULL	NULL	NULL
Abeokuta	WAN	Nigeria	377000	NULL	NULL
Aberdeen	GB	Grampian	219100	NULL	NULL
Aberystwyth	GB	Ceredigion	NULL	NULL	NULL
Abidjan	CI	"Cote dIvoire"	NULL	-3.6	5.3
Abilene	USA	Texas	108476	-99.6833	32.4167
"Abu Dhabi"	UAE	"United Arab Emirates"	363432	54.36	24.27

...

Import from file

```
path <- "d:/Dropbox/ita-2017/tutorial/city.tsv"
city.frame <- read.table(
      path,
      header=TRUE,
      stringsAsFactors=FALSE,
      sep="\t")

city.frame

   Name     Country  Province              Population Longitude Latitude
1  Aachen   D        Nordrhein Westfalen   247113     NULL      NULL
2  Aalborg  DK       Denmark               113865     10        57
3  Aarau    CH       AG                    NULL       NULL      NULL
4  Aarhus   DK       Denmark               194345     10.1      56.1
5  Aarri    WAN      Nigeria               111000     NULL      NULL
6  Aba      WAN      Nigeria               264000     NULL      NULL
```

Getting information about a data frame

```
> names(city.frame)
[1] "Name"       "Country"    "Province"   "Population" "Longitude"
[6] "Latitude"
>
> str(city.frame)
'data.frame':   3053 obs. of  6 variables:
 $ Name      : chr  "Aachen" "Aalborg" "Aarau" "Aarhus" ...
 $ Country   : chr  "D" "DK" "CH" "DK" ...
 $ Province  : chr  "Nordrhein Westfalen" "Denmark" "AG" "Denmark" ...
 $ Population: chr  "247113" "113865" "NULL" "194345" ...
 $ Longitude : chr  "NULL" "10" "NULL" "10.1" ...
 $ Latitude  : chr  "NULL" "57" "NULL" "56.1" ...

> nrow(city.frame)
[1] 3053
> ncol(city.frame)
[1] 6
> dim(city.frame)
[1] 3053    6
>
```

Getting information about a data frame

```
> head(city.frame)
     Name Country            Province Population Longitude Latitude
1  Aachen       D Nordrhein Westfalen     247113      NULL     NULL
2 Aalborg      DK             Denmark     113865        10       57
3   Aarau      CH                  AG       NULL      NULL     NULL
4  Aarhus      DK             Denmark     194345      10.1     56.1
5   Aarri     WAN             Nigeria     111000      NULL     NULL
6     Aba     WAN             Nigeria     264000      NULL     NULL

> tail(city.frame)
          Name Country   Province Population Longitude Latitude
3048 Zonguldak      TR  Zonguldak     115900      NULL     NULL
3049       Zug      CH         ZG       NULL      NULL     NULL
3050     Zunyi      TJ    Guizhou     261862      NULL     NULL
3051    Zurich      CH         ZH     343106      NULL     NULL
3052   Zwickau       D    Sachsen     104921      NULL     NULL
3053    Zwolle      NL Overijssel       NULL      NULL     NULL
```

Accessing a data-frame (1)

- Examples:
  - return all city names:

```
city.frame$Name
```

  - return name and population from cities in switzerland:

```
city.frame[city.frame$Country=="CH",c('Name','Population')]
```

  - Replace NULL values in column Population with NA (not available)

```
city.frame$Population[city.frame$Population=="NULL"]<-NA
```

  - Change datatype of column Population to numeric

```
city.frame<-transform(city.frame, Population=as.numeric(Population))
```

  - return city names, ordered by name

```
sort(city.frame$Name)
```

  - Adding a dataset to a data frame

```
city.frame<-rbind(city.frame, c('Richterswil','CH','ZH',21654,NA,NA))
```

Accessing a data-frame (2)

- Return all Cities with name and population

```
city.frame[,c('Country','Population')]
```

- Return all cities with coordinates

```
city.frame[! is.na(city.frame$Longitude) &
            ! is.na(city.frame$Latitude),]
```

- City with most inhabitants

```
max.population<-max(city.frame$Population, na.rm=TRUE)
city.frame[!is.na(city.frame$Population) &
              city.frame$Population==max.population,]
      Name Country    Province Population Longitude Latitude
2410 Seoul     ROK South Korea   10229262   126.967  37.5667
```

grep

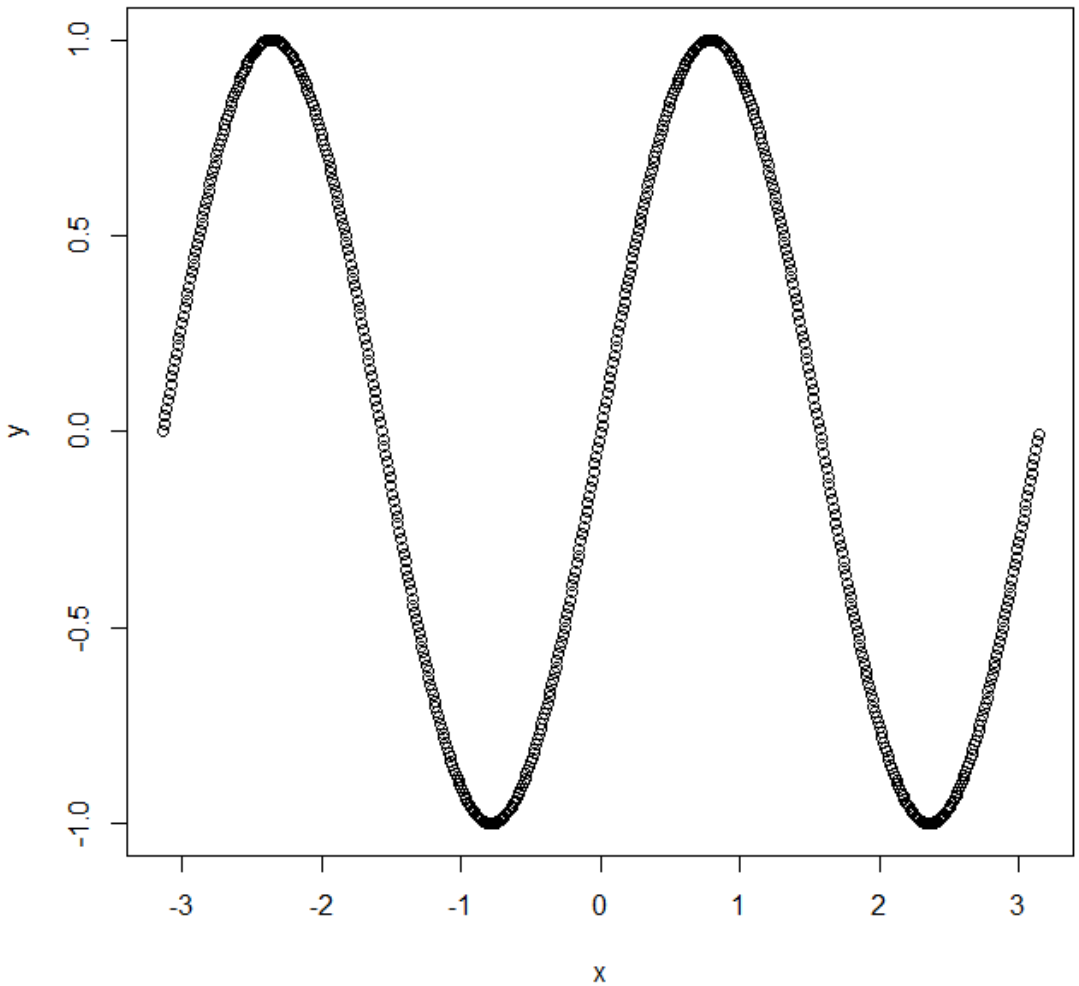
- Searching for regular expressions

```
> grep('^St.*', city.frame$Name)

[1] 2530 2531 2532 2533 2534 2535 2536

> city.frame[grep('^St.*', city.frame$Name),c('Name','Country')]

                  Name Country
2530          St. Louis     USA
2531           St. Paul     USA
2532     St. Petersburg     USA
2533         St. Polten       A
```

Hands-on-Exercise II - data.frame

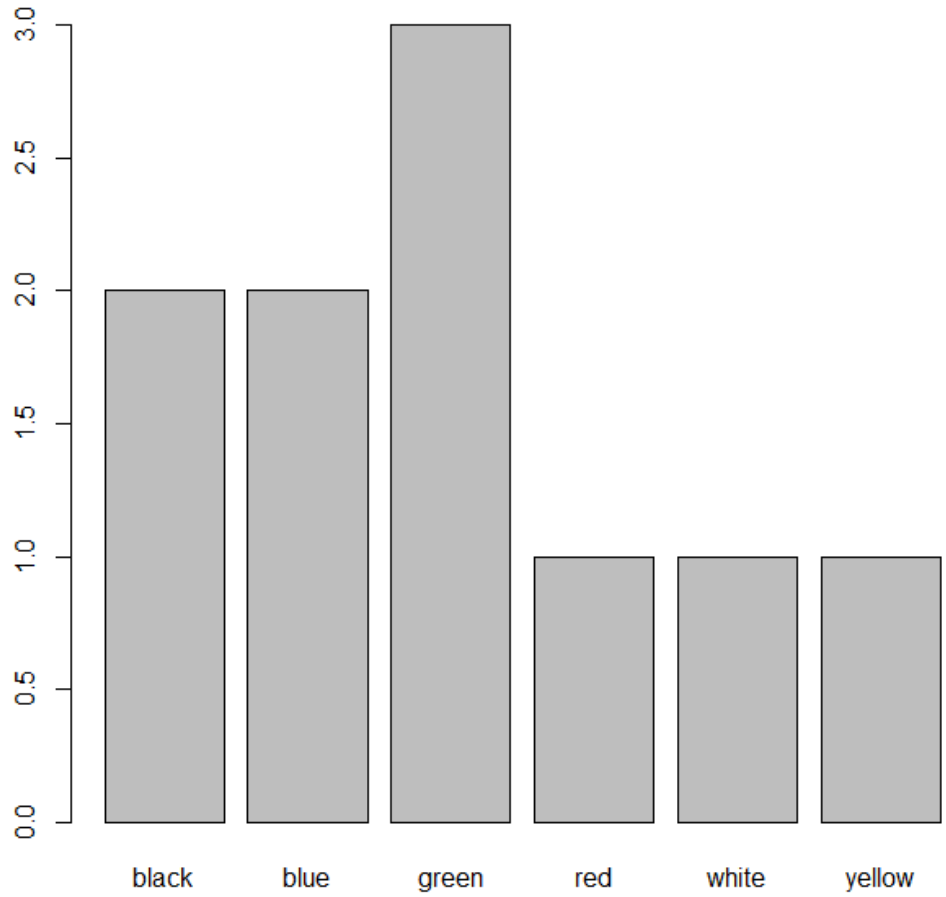
Graphics 101 - plot

```
x<-seq(-pi, pi, by=0.01)
y<-sin(2*x)

plot(x,y)


# try help(plot) for more
# information and options
#
# i.e. plot(x,y, type="l")
```



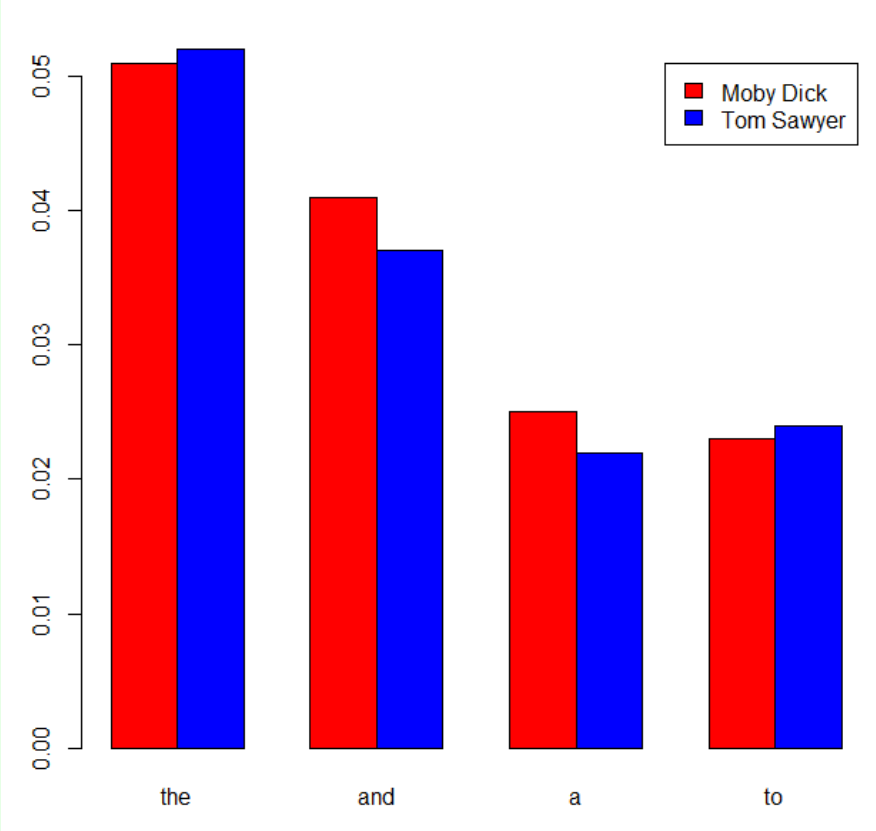
Graphics 101 - barplot

```
> colors<-c("blue","red","green",
    "green", "yellow", "green", "blue",
    "black", "black", "white")
>
> counts<-table(colors)
> counts
colors
 black   blue  green    red  white yellow
     2      2      3      1      1      1

> barplot(counts)
```



Graphics 101 - stacked barplot

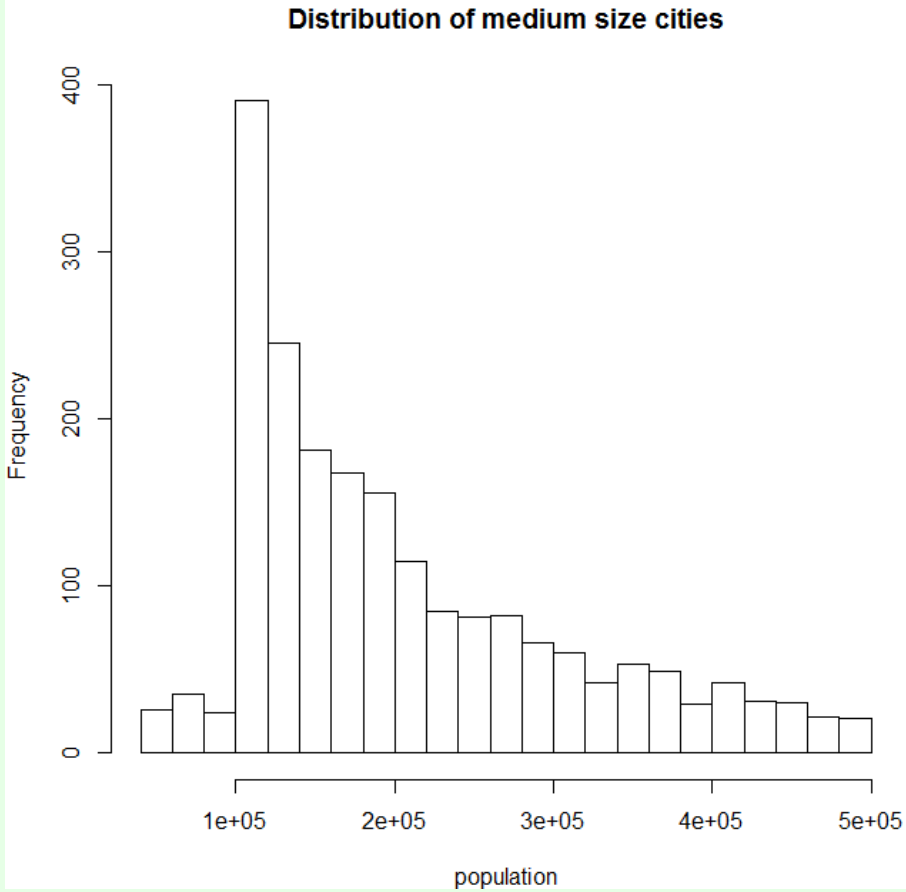
```
> word.freq<-matrix(rep(0, 8), nrow=2)
> rownames(word.freq)<-c('Moby Dick','Tom Sawyer')
> colnames(word.freq)<-c('the','and','a','to')
>
> word.freq['Moby Dick',]<-c(0.051, 0.041,
                             0.025, 0.023)
> word.freq['Tom Sawyer',]<-c(0.052, 0.037,
                              0.022, 0.024)
>
> word.freq
             the   and     a    to
Moby Dick  0.051 0.041 0.025 0.023
Tom Sawyer 0.052 0.037 0.022 0.024
>
> barplot(word.freq,
+         col=c('red','blue'),
+         legend=rownames(word.freq),
+         beside=TRUE)
>
```



Graphics 101 - histogram

- Histogram for (numeric values only)
- Example:

```
medium.size.cities<-
   city.frame[city.frame$Population > 50000 &
              city.frame$Population < 500000,
              'Population']

hist(medium.size.cities,
     breaks=20,
     xlab="population",
     main="Distribution of medium size cities")
```



saving a plot to disk

```
> pdf("c:/temp/figures/color-frequency.pdf")

> barplot(counts$x, names=counts$Group.1,
+         main="Frequency of different colors")

> dev.off()
```

Control flow elements

- R is a complete programming language (turing complete)
- Loops
- Conditional elements
- Definition of user defined functions

Loops

- for - Loop

```
> x<-1
> fak<-5
> for (i in 2:fak)
+     x<-x*i
> cat(fak,"! = ", x,"\n")
5 ! =  120
```

- while loop

```
> eps<-0.00003;
> a<-1000
> steps<-0
> while (a > eps) {
+     a <- a/2.0
+     steps<-steps + 1
+ }
> cat("steps:", steps,"\n")
steps: 25
```

type help(cat) to get more info about

Loops - next & break

```
i <- 0
while (T) {
    cat(i,"\n")
    i<-i+1
    if (i==5)
       break
}
cat("loop ended\n")
```

```
0
1
2
3
4
loop ended
```

```
for (i in 0:5) {
    if (i==3)
       next
    cat(i,"\n")
}
cat("loop ended\n")
```

```
0
1
2
4
5
loop ended
>
```

Conditional Statements

- if - else

```
> a<-rnorm(1)
> b<-rnorm(1)
> if (a > b) {
+     tmp<-a
+     a<-b
+     b<-tmp
+     cat("exchange ", a, " with ", b,"\n")
+ } else
+     cat("nothing to do\n")
exchange  -1.165896  with  1.043969
> cat(a," is smaller than ",b,"\n")
-1.165896  is smaller than  1.043969
```

- ifelse

```
> a<-rnorm(1)
> str<-ifelse(a>0, "positive", "negative")
> cat(a, "is", str,"\n")
1.661342 is positive
>
```

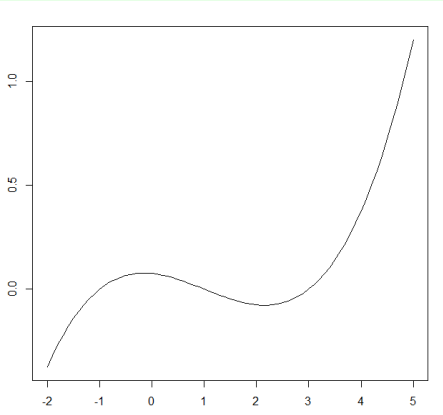
User defined functions

- General syntax:

```
funcname <- function(arg1, arg2, ...) {
                statements
            }
```

- Example:

```
my.polynom<-function(x) {
                y <- 1/4*(x-3) * 1/2*(x-1) * 1/5*(x+1)
                return (y)
            }

x <- seq(-2,5, by=0.1)
y <- my.polynom(x)
plot(x,y, type="l")

# or

curve(my.polynom, -2,5)
```



Writing to a file

- Example:

```
> file<-"c:/temp/test.txt"
> file.create(file)
> write("An Introduction into Statistical Computing with R", file=file)
> for (i in 100:500) {
+     write(paste("line", i), file=file, append=T)
+ }
```

- File content (c:/temp/test.txt)

```
An Introduction into Statistical Computing with R
line 100
line 101
line 102
line 103
line 104
line 105
line 106
line 107
...
```

Execution of external scripts

- File myScript.R

```
f<-function(x) {
      y <- 1/4*(x-3) *
            1/2*(x-1) *
            1/5*(x+1)
      return (y)
}

x <- -2
while (x < 5) {
    cat("f(",x,")=", f(x),"\n")
    x<- x + 0.1
}

/*
   in script mode you must write
   print(x) or cat(x)
   instead of only x,
   to output the content of x
*/
```

- Inside R IDE:

```
> source('myScript.R')
f( -2 )= -0.375
f( -1.9 )= -0.319725
f( -1.8 )= -0.2688
f( -1.7 )= -0.222075
...
```

- From Operating System:

```
$R_HOME/bin/x64/R.exe -f myScript.R
f( -2 )= -0.375
f( -1.9 )= -0.319725
f( -1.8 )= -0.2688
f( -1.7 )= -0.222075
...
```

Outlook - Big Data and R

- Easy Integration of C/C++ Code (package Rcpp)
- Memory mapped file-access (package bigmemory)
- Parallelistation (package parallel)
  - Multithreading
    - Communication via shared memory or
    - sockets
  - Cluster
    - Communication via sockets
    - Use of R inside a Hadoop cluster (package rmr2, rhdfs, rhbase)

Hands-on-Exercise III - text & graphic

Resources

- Norman Matloff, The Art of R Programming, http://heather.cs.ucdavis.edu/~matloff/132/NSPpart.pdf
- Rob Kabacoff. R in Action, Second Edition - Data analysis and graphics with R, Manning, 2015.
- Matthias Kohl, Introduction to statistical data analysis with R. bookboon.com http://bookboon.com/en/introduction-to-statistical-data-analysis-with-r-ebook
- Data Camp (very good online courses), Overview: https://www.datacamp.com/getting-started?step=2&track=r
- G. Ryan Spain, R Essentials, DZone. https://dzone.com/asset/download/88835

Appendix Function overview (selection)

- Mathematical

  `abs, sqrt, ceiling, floor, trunc, round, signif, cos, sin, tan, acos, asin, atan, log, log, log10, exp, mean, median, sd, var, quantile, range, sum, diff, min, max, scale, ...`
- Character

  `nchar, substr, grep, strsplit, paste, toupper, tolower, sub, gsub, ...`
- Misc

  `length, seq, rep, cut, pretty, cat`
- Conversion

  `as.vector, as.numeric, as.charcter, unlist, as.matrix, as.data.frame, as.Date`
- Statistic

  `rnorm, runif, rbinom, ...`