Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

The Eight International Conference on Advances in Databases, Knowledge, and Data Applications

June 26 - 30, 2016 - Lisbon, Portugal

An Introduction into Statistical Computing and Natural Language Processing 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

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Outlook

- What is R?
- Basic Data Structures
- Basic-Operations
  - Getting information about your data
  - Import- and Export
  - Operations on data
  - Some more useful build in functions
  - User defined functions
- NLP basics
- R and Big Data Applications

+ 3 hands on exercices

- First contact
- Querying data
- Simple NLP example

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)

A first example

see Hands-on Exercise I ...

(next page)

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hands-On Exercise 1 (Page 1)

1. Install R from https://www.r-project.org/
2. Start the R environment
3. Set the values of two variables

   ```
   x<-12
   y<-6
   ```
4. Calculate the sum:

   ```
   x+y
   ```
5. Define a vector with two elements:

   ```
   vec<-c(x,y)
   ```
6. Show the content of the vector

   ```
   vec
   ```
7. Calculate the mean of the values

   ```
   mean(vec)
   ```
8. Calculate the standard derivation

   ```
   sd(vec)
   ```
9. Generate 100 random numbers (normal distribution with mean=0, sd=1) and store them in a vector with name „n“

   ```
   n<-rnorm(100)
   ```
10. Show the generated vector

    ```
    n
    ```
11. Give some information about the „rnorm“ function:

    ```
    help(rnorm)
    ```
12. Print a histogram of the values

    ```
    hist(n)
    ```
13. Give some information about the function „hist“

    ```
    help(hist)
    ```
14. Your first job: change the number of intervals to 20

    ...

Hands-On Exercise 1 (Page 2)

15. Create a file with the following content and store it somewhere on disk (with name „hands-on-1-batch.R"):

```
x<-seq(from=-10, to=10, by=0.1)

my.first.func<-function(x) {
    return (sin(x) * 2)
}

y<-my.first.func(x)
plot(x,y)
```

16. Execute the script with the following command:

```
source("<path-to-script>/hands-on-1-batch.R")
```

R Basic Datastructures

- Vector
- Matrices
- Array
- Lists
- Data Frames

Vector

- One dimensional data structure
- All elements must be of same type
- Basic datatype (there exists no skalar values)
- 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
  - „:“-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 Operations

- Example:

```
> vector <- 10:1
> vector
 [1] 10  9  8  7  6  5  4  3  2  1
> pos <- seq(1,10, by=2)
> pos
[1] 1 3 5 7 9
> vector[pos]
[1] 10  8  6  4  2
> vector[-pos]
[1] 9 7 5 3 1
>
> a_filter<-c(TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE)
> vector[a_filter]
[1] 10  8  6  4  2

> x1 <- seq(1,10, by=2)
> x2 <- seq(2,10, by=2)
> c(x1,x2)
 [1]  1  3  5  7  9  2  4  6  8 10
```

Vector Element Names

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

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

Adding/deleting elements to a vector

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

Vectorized functions

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

```
> 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
>
```

- Rule of thumb: if a function uses vectorized operations, it is also vectorized

Vector filtering

- Example:

```
> x <- rnorm(10)
> x
 [1] -0.7491410 -0.8740810 -0.9511798  0.2182755  1.0107457  0.5258976 -0.9350032
 [8] 0.3756790 -1.3494970 -0.4172580
>
> x > 0
 [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
>
> positive_values <- x[x > 0]
>
> positive_values
[1] 0.2182755 1.0107457 0.5258976 0.3756790
>
> y <- ifelse(x>=0,'Pos.','Neg.')
> y
 [1] "Neg." "Neg." "Neg." "Pos." "Pos." "Pos." "Neg." "Pos." "Neg." "Neg."
>
```

set operations with vectors

```
> union(c(1,2), c(2,3))
[1] 1 2 3
> intersect(c(1,2), c(2,3))
[1] 2
> setdiff(c(1,2), c(2,3))
[1] 1
> setequal(c(1,2), c(2,3))
[1] FALSE
> setequal(c(1,2), c(2,1))
[1] TRUE
> 2 %in% c(1,4,6)
[1] FALSE
> 4 %in% c(1,4,6)
[1] TRUE
>
```

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

   b1 b2 b3
a1  1  4  7
a2  2  5  8
a3  3  6  9

, , c2

   b1 b2 b3
a1 10 13 16
a2 11 14 17
a3 12 15 18

, , c3

   b1 b2 b3
a1 19 22 25
a2 20 23 26
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(1,2,3)
> name<-c("Klaus", "Ingo", "Tanja")
> age<-c(31,27,29)
> dataset<-data.frame(PersonID, name, age)
> dataset
PersonID  name age
1         1 Klaus  31
2         2  Ingo  27
3         3 Tanja  29
```

Data-Frame: Acess methods

```
> dataset[2,]
PersonID name age
2        2 Ingo  27

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

> dataset[2,2]
[1] Ingo

> dataset$age
[1] 31 27 29
>
```

```
> dataset[c("name","age")]
   name age
1 Klaus  31
2  Ingo  27
3 Tanja  29

> dataset[3,c("name","age")]
   name age
3 Tanja  29
```

Filtering

- Examples:

```
> dataset
  PersonID  name age
1        1 Klaus  31
2        2  Ingo  27
3        3 Tanja  29
> dataset$age < 30
[1] FALSE  TRUE  TRUE
> dataset[dataset$age < 30,]
   name age
2  Ingo  27
3 Tanja  29
>
> dataset[dataset$age < 30 & dataset$age > 28, c("name")]
[1] Tanja
```

Data Frame filtering: Comparison to SQL

- R

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

- SQL:

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

projection

selection

List-Datastructure

- Ordered collection of objects
- Example (named list):

```
> dozent<-list(firstname="Steffen", surname="Scholz")>
> dozent
$firstname
[1] "Steffen"
$surname
[1] "Scholz"
> dozent$surname
[1] "Scholz"
> dozent$gender<-"male"
> dozent
$firstname
[1] "Steffen"
$surname
[1] "Scholz"
$gender
[1] "male"
```

Operations on lists

- Example (unnamed list):

```
> a_list<-list(c(1,2),
               "example",
               matrix(1:4, nrow=2))
> a_list
[[1]]
[1] 1 2

[[2]]
[1] "example"

[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
>
```

```
> a_list[3]
[[1]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

> a_list[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
>
> str(a_list[3])
List of 1
 $ : int [1:2, 1:2] 1 2 3 4
>
> str(a_list[[3]])
 int [1:2, 1:2] 1 2 3 4
```

Converting List to a vector (unlist)

- List to vector

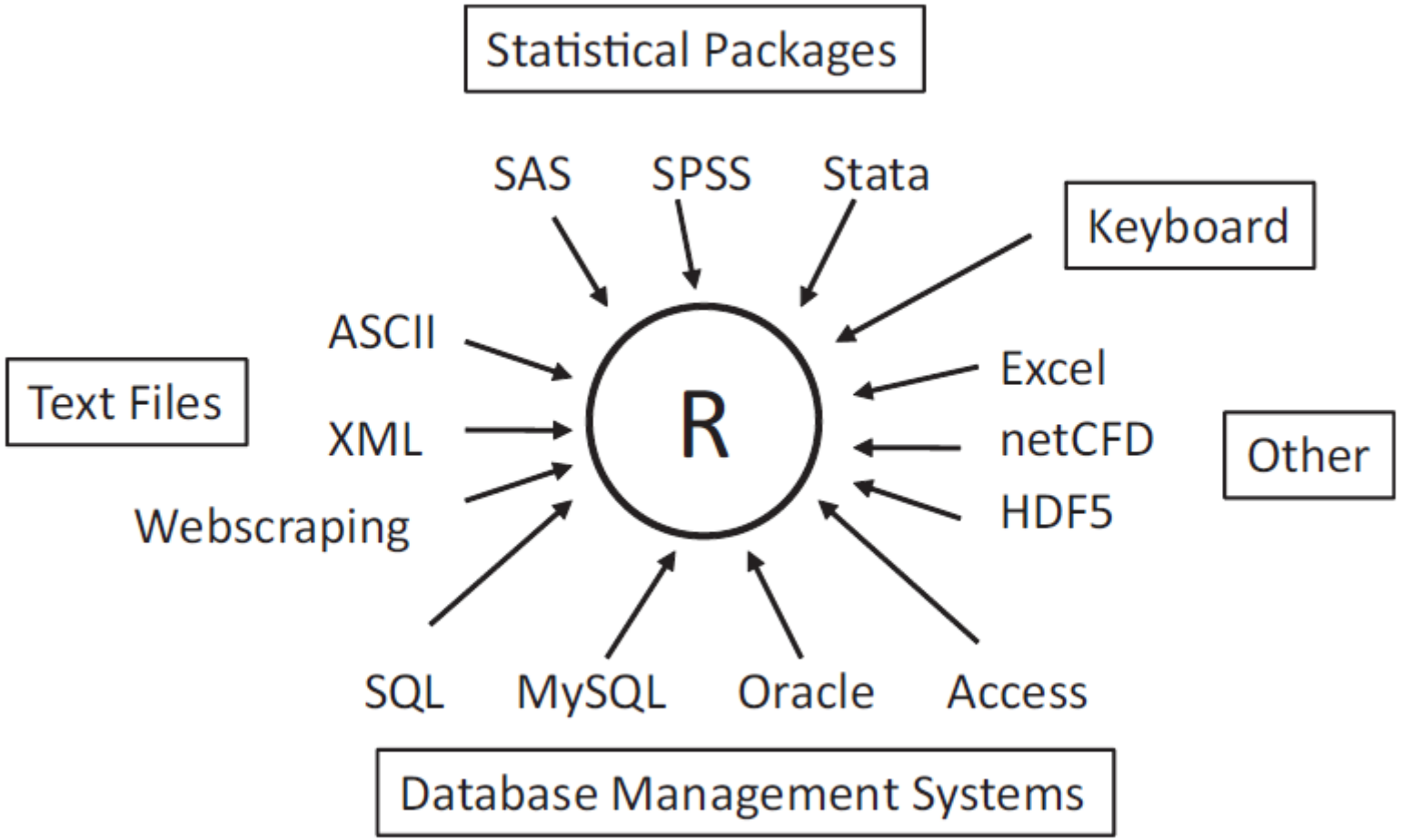
```
> line1=list("the", "adventures", "of", "tom", "sawyer")
> str(line1)
List of 5
 $ : chr "the"
 $ : chr "adventures"
 $ : chr "of"
 $ : chr "tom"
 $ : chr "sawyer"
> str(unlist(line1))
 chr [1:5] "the" "adventures" "of" "tom" "sawyer"
>
```

Converting Nested Lists (unlist)

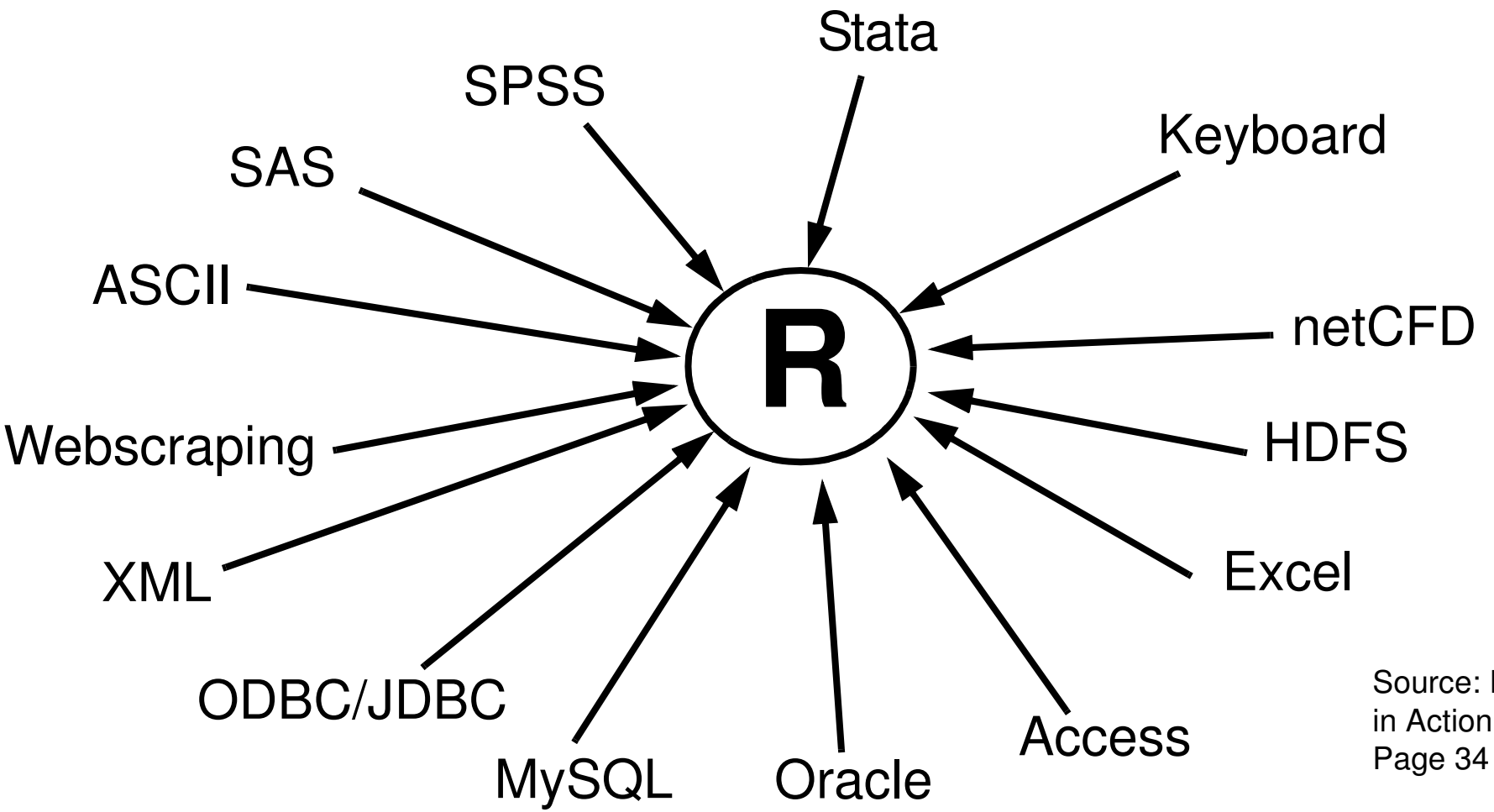
```
> text<-list(list("the", "adventures", "of", "tom", "sawyer"),
+            list("by", "mark","twain"))
> str(text)
List of 2
 $ :List of 5
  ..$ : chr "the"
  ..$ : chr "adventures"
  ..$ : chr "of"
  ..$ : chr "tom"
  ..$ : chr "sawyer"
 $ :List of 3
  ..$ : chr "by"
  ..$ : chr "mark"
  ..$ : chr "twain"
> str(unlist(text))
 chr [1:8] "the" "adventures" "of" "tom" "sawyer" "by" "mark" "twain"
>
```

Data Import in R

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



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/dbkda-2016/tutorial"
city.frame <- read.table(
      paste(path,"/","city.tsv",sep=""),
      header=TRUE,
      stringsAsFactors=FALSE,
      sep="\t")

city.frame
```

```
   Name     Country  Province              Population LongitudeLatitude
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

- 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
```

Change Ordering/Sorting

- Example:

```
dutch.cities<-city.frame[city.frame$Country=="DK",
                         c('Name','Population')]
dutch.cities
          Name Population
2       Aalborg     113865
4        Aarhus     194345
602  Copenhagen    1358540
779     Esbjerg      70975
1915     Odense     136803
2178    Randers      55780
```

  - Change order:

```
dutch.cities[c(6,5,4,3,2,1),]
          Name Population
2178    Randers      55780
1915     Odense     136803
779     Esbjerg      70975
602  Copenhagen    1358540
4        Aarhus     194345
2       Aalborg     113865
```

sort vs. order

```
dutch.cities$Population
[1]  113865  194345 1358540   70975  136803   55780
```

- sort(...): Sorts the elements in a vector

```
sort(dutch.cities$Population)
[1]   55780   70975  113865  136803  194345 1358540
```

- order(...): returns the permutation which rearranges the arguments into increasing or decreasing order

```
> order(dutch.cities$Population)
[1] 6 4 1 5 2 3

> order(dutch.cities$Population, decreasing=TRUE)
[1] 3 2 5 1 4 6
```

Sorting datasets

```
dutch.cities[order(dutch.cities$Population, decreasing=T),]
           Name Population
602  Copenhagen    1358540
4        Aarhus     194345
1915     Odense     136803
2       Aalborg     113865
779     Esbjerg      70975
2178    Randers      55780
>
```

Aggregate (Some simple statistic)

- Example:

```
> num.cities.per.country<-
          aggregate(city.frame$Country, by=list(city.frame$Country), FUN=length)
> num.cities.per.country
Group.1   x
1         A   9
2       AFG   1
3        AG   1
4        AL   6
5       AND   1
6       ANG  18
7       ARM   1
8       AUS  18
9        AZ   1
> str(num.cities.per.country)
'data.frame':   195 obs. of  2 variables:
 $ Group.1: chr  "A" "AFG" "AG" "AL" ...
 $ x      : int  9 1 1 6 1 18 1 18 1 12 ...
```

- Number of inhabitants of a country who live in a city

```
> aggregate(city.frame$Population, by=list(city.frame$Country), FUN=sum, na.rm=TRUE)
Group.1         x
1         A   2434525
2       AFG    892000
3        AG     36000
4        AL    475000
5       AND     15600
6       ANG         0
7       ARM   1200000
```

Frequency tables

```
> colors<-c("blue","red","green", "green", "yellow", "green",
+           "blue", "black", "black", "white")

> counts<-aggregate(colors, by=list(colors), FUN=length)
> counts
  Group.1 x
1   black 2
2    blue 2
3   green 3
4     red 1
5   white 1
6  yellow 1
> str(counts)
'data.frame':   6 obs. of  2 variables:
 $ Group.1: chr  "black" "blue" "green" "red" ...
 $ x      : int  2 2 3 1 1 1
> counts[,'x']
[1] 2 2 3 1 1 1
> counts[,'Group.1']
[1] "black"  "blue"   "green"  "red"    "white"  "yellow"
>
```

table-operator

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

String handling

- General String handling

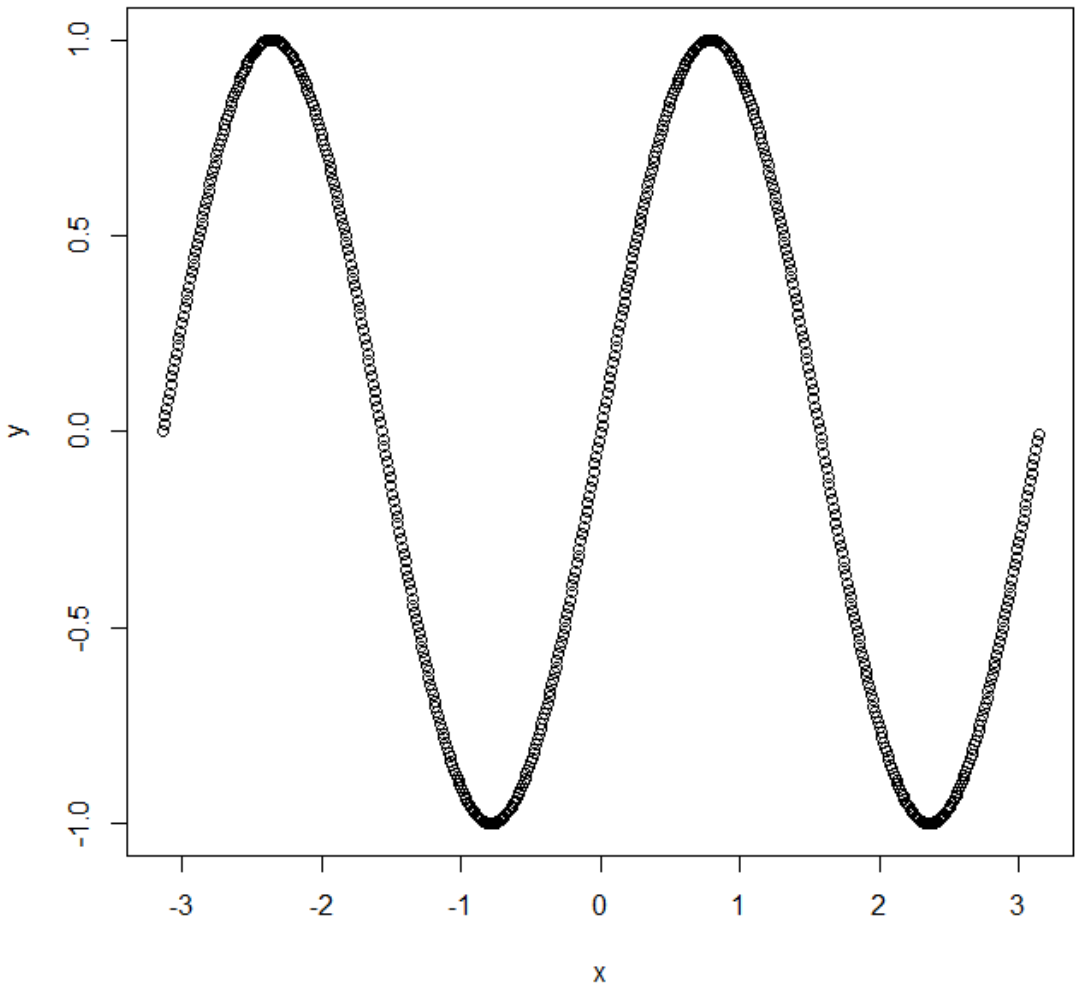
```
paste(..., sep=" ", collapse=NULL), cat(...), tolower(str), toupper(str), chartr(pat-
tern, replace, str), substr(x, start, stop), ncar(str), ...
```

- Regular expression functions:

```
sub(pattern=..., replacement=..., x=...), gsub(pattern=..., replacement=..., x=...),
grep(pattern, text), strsplit(x, split), ...
```

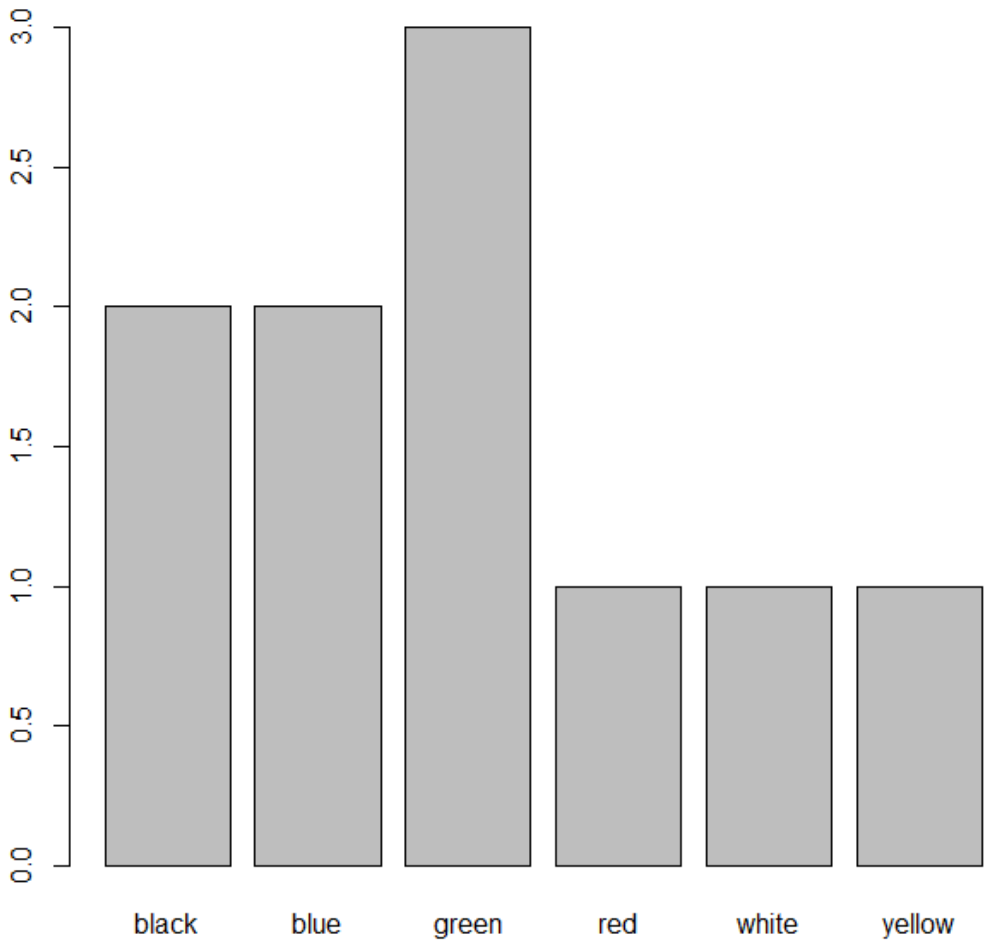
Graphics 101 - plot

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

plot(x,y)


# try help(plot) for more
# information and options
```



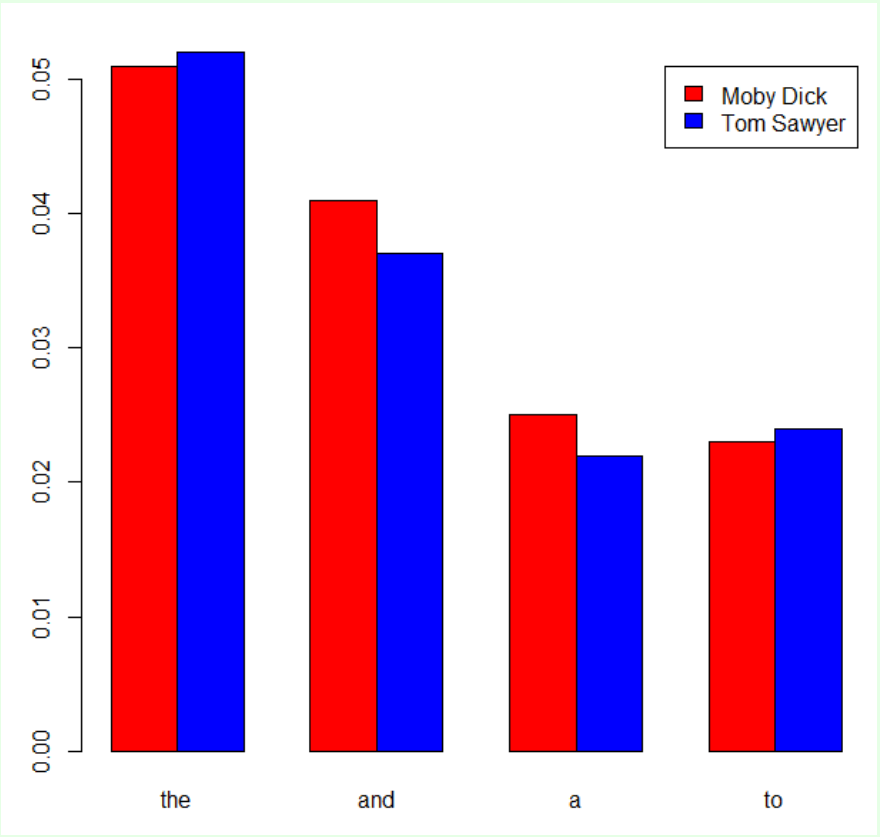
Graphics 101 - barplot

```
> colors<-c("blue","red","green",
  "green", "yellow", "green", "blue",
  "black", "black", "white")
>
> counts<-aggregate(colors,
                    by=list(colors),
                    FUN=length)
> counts
  Group.1 x
1   black 2
2    blue 2
3   green 3
4     red 1
5   white 1
6  yellow 1

> barplot(counts$x,
          names=counts$Group.1)
```



Graphics 101 - stacked barplot

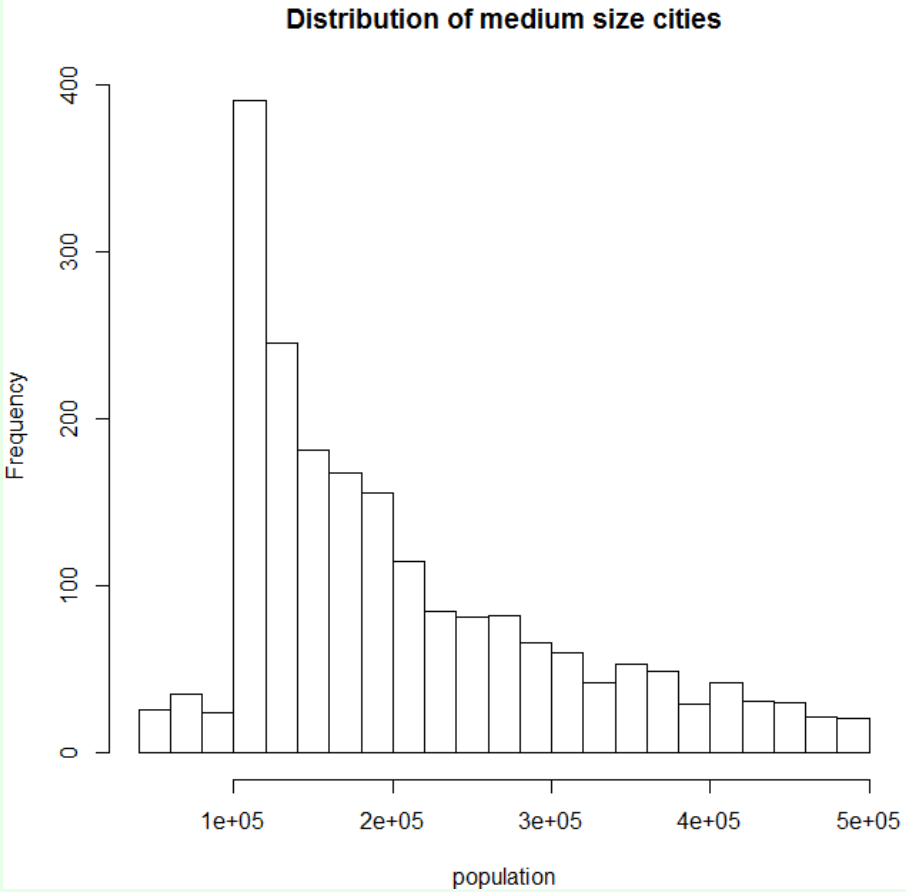
```
> word.freq<-matrix(rep(0, 8), nrow=2)
> rownames(word.freq)<-c('Moby Dick','Tom Sawyer')
> colnames(word.freq)<-c('the','and','a','to')
>
> word.freq['Moby Dick',]<-c(0.051, 0.041,
                             0.025, 0.023)
> word.freq['Tom Sawyer',]<-c(0.052, 0.037,
                              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()
```

Hands-On Exercise 2

1. Download the file city.tsv from http://www.smiffy.de/dbkda-2016/city.tsv and store it on your disk.
2. Import the file into a data frame with name 'city.all'.
3. Substitute the „NULL"-values for the concerned columns by NA values.
4. Change the datatype of the columns Longitude, Latitude and Population to numeric.
5. Formulate the following queries:
   - How many cities are in the data frame?
   - Return the cities which reside on the equator.
   - Return all cities from Brasilia with coordinates, ordered by their population.
   - How many cities can be found in the USA
   - In which states (Province) can cities with the name 'Springfield' be found?

6. Add the city of Roskilde (population: 46701; coordinates: 55.65, 12.083333) to the data frame.
7. Formulate a query, which returns Roskilde from the data frame.

8. Datastructure conversion:
   Generate the follwing data frame:

```
to.convert<-city.frame[city.frame$Country=="DK",
                       c('Name','Population')]
```

   and convert it into a named vector (population as vector values, cityname as vector element names)

9. Graphics:
   - Create a plot, which shows the koordinates (longitude, latitude) of all cities.

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
```

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
>
```

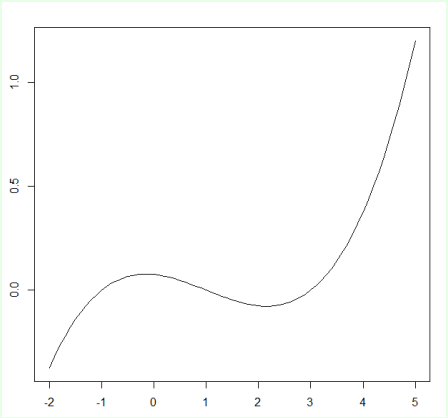
User defined functions

- 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")
```



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
...
```

A NLP Example

General idea:

- The use of words (and the frequency) vary from author to author
- This is mainly relevant for frequently occuring words (i.e. stop-words)
- To identify an author of a unknown book, compare (visually) the similarity of the histograms of frequent words

A simple NLP Example

- Download the 2 books (Moby Dick and Tom Sawyer) from the tutorial page at http://www.smiffy.de/dbkda-2016.
- Download also the additional book „Book from an Unknown Author“ from this page.
- Evaluate the function readBookFromFile, given in Appendix A of the third hands-on-exercice.
- Use the function readBookFromFile(path, ...) to read the books from your local disk.
- Build an appropriate datastructure to represent each book as a histogram of the 10 most frequent terms.
- Compare the histograms and decide from which author the third book was written.
- Modify your plots, so that the histogram of the book from the unknown author is additionally shown in the histograms of Tom Sawyer and Moby Dick (hint: Stacked Barplot)

Karlsruhe Institute of Technology

Hochschule Karlsruhe Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hands-On Exercise 3

1. Goto the webpage http://www.smiffy.de/dbkda-2016
2. Evaluate the function 'readBookFromFile(...) (Appendix A) in your R-environmant.
3. Download the books „Moby Dick", „Tom Sawyer", and „Ulysses" to your local disk. Remove the Gutenberg specific header and footer information.
4. Load the first 20 lines from the book „Moby Dick" with the previously evaluarted function and examine the result.
5. Count the number of occurence for each word (using R functionality of course ;-)).
6. Draw a graph which shows the frequency of the most popular 20 words of the book Moby Dick (barplot).
7. Combine the bartplots from different books in one chart (StackedBarPlot)

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)

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, ...`