An Introduction into Statistical Computing and Natural Language Processing with R

Andreas Schmidt\textsuperscript{1,2} and Steffen G. Scholz\textsuperscript{2}

\textsuperscript{(1)} Department of Informatics and Business Information Systems University of Applied Sciences Karlsruhe Germany

\textsuperscript{(2)} Institute for Applied Computer Sciences Karlsruhe Institute of Technology Germany
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 exercises
- 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 capabilities
- 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)
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
    ...
15. Create a file with the following content and store it somewhere on disk (with name "hands-on-1-batch.R"):
   ```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:
    ```r
    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 scalar values)
- Example:

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


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

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

  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:

```r
> 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 (additionally to index-position)
- Example:

```r
> 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

```r
> 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:

  ```r
  > sqrt(9)
  [1] 3
  > x <- 1:10
  > sqrt(x)
  [10] 3.162278
  >
  ```

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

• Example:

```r
> 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
```
set operations with vectors

```r
> 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,]  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

```r
> 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

```r
> 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)

```r
> cbind(m1, m2)
[1,]  11  21  31  31
[2,]  12  22  32  32
```

- Adding rows (rbind):

```r
> rb <- rbind(m1, m2)
> rb
[1,] 11 21 31 31
[2,] 12 22 32 32
```

```r
> nrow(rb)
[1] 4
> ncol(rb)
[1] 2
```
Example: Deleting rows/columns from a matrix

```r
> 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
```

Andreas Schmidt  DBKDA - 2016
Filtering matrices (row/columnwise)

```r
> m <- matrix(round(rnorm(24)), nrow=4)
> m
[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,]  1  1  -2  1  0  0
[2,] -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
> m
[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
```
Array

- Like matrix but can have more than 2 dimensions
- Syntax:
  \[
  \text{myarray} \leftarrow \text{array} (\text{vector}, \text{dimensions}, \text{dimnames})
  \]
- Examples:
  \[
  \text{myarray} \leftarrow \text{array}(1:27, \text{c}(3,3,3), \text{list}(\text{c}("a1","a2","a3"),
  \text{c}("b1","b2","b3"),
  \text{c}("c1","c2","c3")))
  \]
  >
  >
  > \text{myarray}
  , , c1 , , c2 , , c3
  |
  | 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:
  
  \[ \text{myframe} \leftarrow \text{data.frame}(\text{col1}, \text{col2}, \text{col3}, \ldots) \]

- Example:

  \[
  \begin{align*}
  &\text{PersonID} \leftarrow c(1,2,3) \\
  &\text{name} \leftarrow c("Klaus", "Ingo", "Tanja") \\
  &\text{age} \leftarrow c(31,27,29) \\
  &\text{dataset} \leftarrow \text{data.frame}(\text{PersonID, name, age}) \\
  &\text{dataset} \\
  \text{PersonID} & \text{name} \text{ age} \\
  1 & 1 \text{Klaus} & 31 \\
  2 & 2 \text{Ingo} & 27 \\
  3 & 3 \text{Tanja} & 29
  \end{align*}
  \]
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:

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

- **SQL**:
  
  ```sql
  select name
  from dataset_table
  where age < 30
  and age > 28
  ```
List-Datastructure

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

```r
> 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):

  ```r
  > 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

  > str(a_list[3])
  List of 1
  $ : int [1:2, 1:2] 1 2 3 4
  ```
Converting List to a vector (unlist)

• List to vector

```r
> linel=list("the", "adventures", "of", "tom", "sawyer")
> str(linel)
List of 5
$ : chr "the"
$ : chr "adventures"
$ : chr "of"
$ : chr "tom"
$ : chr "sawyer"
> str(unlist(linel))
chr [1:5] "the" "adventures" "of" "tom" "sawyer"
```
Converting Nested Lists (unlist)

```r
> 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
<table>
<thead>
<tr>
<th>Name</th>
<th>Country</th>
<th>Province</th>
<th>Population</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aachen</td>
<td>D</td>
<td>&quot;Nordrhein Westfalen&quot;</td>
<td>247113</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aalborg</td>
<td>DK</td>
<td>Denmark</td>
<td>113865</td>
<td>10</td>
<td>57</td>
</tr>
<tr>
<td>Aarau</td>
<td>CH</td>
<td>AG</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aarhus</td>
<td>DK</td>
<td>Denmark</td>
<td>194345</td>
<td>10.1</td>
<td>56.1</td>
</tr>
<tr>
<td>Aarri</td>
<td>WAN</td>
<td>Nigeria</td>
<td>111000</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aba</td>
<td>WAN</td>
<td>Nigeria</td>
<td>264000</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Abakan</td>
<td>R</td>
<td>&quot;Rep. of Khakassiya&quot;</td>
<td>161000</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Abancay</td>
<td>PE</td>
<td>Apurimac</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Abeokuta</td>
<td>WAN</td>
<td>Nigeria</td>
<td>377000</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aberdeen</td>
<td>GB</td>
<td>Grampian</td>
<td>219100</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aberystwyth</td>
<td>GB</td>
<td>Ceredigion</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Abidjan</td>
<td>CI</td>
<td>&quot;Cote d'Ivoire&quot;</td>
<td>NULL</td>
<td>-3.6</td>
<td>5.3</td>
</tr>
<tr>
<td>Abilene</td>
<td>USA</td>
<td>Texas</td>
<td>108476</td>
<td>-99.6833</td>
<td>32.4167</td>
</tr>
<tr>
<td>&quot;Abu Dhabi&quot;</td>
<td>UAE</td>
<td>&quot;United Arab Emirates&quot;</td>
<td>363432</td>
<td>54.36</td>
<td>24.27</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Import from file

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Country</th>
<th>Province</th>
<th>Population</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aachen</td>
<td>D</td>
<td>Nordrhein Westfalen</td>
<td>247113</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aalborg</td>
<td>DK</td>
<td>Denmark</td>
<td>113865</td>
<td>10</td>
<td>57</td>
</tr>
<tr>
<td>Aarau</td>
<td>CH</td>
<td>AG</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aarhus</td>
<td>DK</td>
<td>Denmark</td>
<td>194345</td>
<td>10.1</td>
<td>56.1</td>
</tr>
<tr>
<td>Aarri</td>
<td>WAN</td>
<td>Nigeria</td>
<td>111000</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Aba</td>
<td>WAN</td>
<td>Nigeria</td>
<td>264000</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Getting information about a data frame

\begin{verbatim}
> 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
\end{verbatim}
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:
    ```r
city.frame$Name
    ```
  - return name and population from cities in switzerland:
    ```r
city.frame[city.frame$Country=="CH",c('Name','Population')]
    ```
  - Replace NULL values in column Population with NA (not available)
    ```r
city.frame$Population[city.frame$Population=="NULL"]<-NA
    ```
  - Change datatype of column Population to numeric
    ```r
city.frame<-transform(city.frame, Population=as.numeric(Population))
    ```
  - return city names, ordered by name
    ```r
    sort(city.frame$Name)
    ```
  - Adding a dataset to a data frame
    ```r
city.frame<-rbind(city.frame, c('Richterswil','CH','ZH',21654,NA,NA))
    ```
Accessing a data-frame (2)

- Return all Cities with name and population
  
  ```r
  city.frame[,c('Country','Population')]
  ```

- Return all cities with coordinates
  
  ```r
  city.frame[!is.na(city.frame$Longitude) & !is.na(city.frame$Latitude),]
  ```

- City with most inhabitants
  
  ```r
  max.population<-max(city.frame$Population, na.rm=TRUE)
  city.frame[!is.na(city.frame$Population) &
              city.frame$Population==max.population,]
  ```

<table>
<thead>
<tr>
<th>Name</th>
<th>Country</th>
<th>Province</th>
<th>Population</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>2410</td>
<td>Seoul</td>
<td>ROK South Korea</td>
<td>10229262</td>
<td>126.967</td>
<td>37.5667</td>
</tr>
</tbody>
</table>
Change Ordering/Sorting

• Example:

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

dutch.cities
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aalborg</td>
<td>113865</td>
</tr>
<tr>
<td>Aarhus</td>
<td>194345</td>
</tr>
<tr>
<td>Copenhagen</td>
<td>1358540</td>
</tr>
<tr>
<td>Esbjerg</td>
<td>70975</td>
</tr>
<tr>
<td>Odense</td>
<td>136803</td>
</tr>
<tr>
<td>Randers</td>
<td>55780</td>
</tr>
</tbody>
</table>

• Change order:

```r
dutch.cities[, c(6, 5, 4, 3, 2, 1)]
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randers</td>
<td>55780</td>
</tr>
<tr>
<td>Odense</td>
<td>136803</td>
</tr>
<tr>
<td>Esbjerg</td>
<td>70975</td>
</tr>
<tr>
<td>Copenhagen</td>
<td>1358540</td>
</tr>
<tr>
<td>Aarhus</td>
<td>194345</td>
</tr>
<tr>
<td>Aalborg</td>
<td>113865</td>
</tr>
</tbody>
</table>
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$Population, decreasing=TRUE)
[1] 3 2 5 1 4 6
dutch.cities[order(dutch.cities$Population, decreasing=T),]

<table>
<thead>
<tr>
<th>Name</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copenhagen</td>
<td>1358540</td>
</tr>
<tr>
<td>Aarhus</td>
<td>194345</td>
</tr>
<tr>
<td>Odense</td>
<td>136803</td>
</tr>
<tr>
<td>Aalborg</td>
<td>113865</td>
</tr>
<tr>
<td>Esbjerg</td>
<td>70975</td>
</tr>
<tr>
<td>Randers</td>
<td>55780</td>
</tr>
</tbody>
</table>

>
Aggregate (Some simple statistic)

- Example:

```r
> 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

```r
> aggregate(city.frame$Population, by=list(city.frame$Country), FUN=sum, na.rm=TRUE)
```

<table>
<thead>
<tr>
<th>Group.1</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>AFG</td>
</tr>
<tr>
<td>3</td>
<td>AG</td>
</tr>
<tr>
<td>4</td>
<td>AL</td>
</tr>
<tr>
<td>5</td>
<td>AND</td>
</tr>
<tr>
<td>6</td>
<td>ANG</td>
</tr>
<tr>
<td>7</td>
<td>ARM</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>2434525</td>
</tr>
<tr>
<td></td>
<td>AFG</td>
</tr>
<tr>
<td></td>
<td>892000</td>
</tr>
<tr>
<td></td>
<td>AG</td>
</tr>
<tr>
<td></td>
<td>36000</td>
</tr>
<tr>
<td></td>
<td>AL</td>
</tr>
<tr>
<td></td>
<td>475000</td>
</tr>
<tr>
<td></td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td>15600</td>
</tr>
<tr>
<td></td>
<td>ANG</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ARM</td>
</tr>
<tr>
<td></td>
<td>1200000</td>
</tr>
</tbody>
</table>
Frequency tables

```r
> 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"
```
> colors<-c("blue","red","green", "green", "yellow", "green", +           "blue", "black", "black", "white")
> table(colors)
> sort(table(colors))
String handling

- General String handling
  
  \texttt{paste(..., sep=" ", collapse=NULL), cat(...), tolower(str), toupper(str), chartr(pattern, replace, str), substr(x, start, stop), nchar(str), ...}

- Regular expression functions:
  
  \texttt{sub(pattern=..., replacement=..., x=...), gsub(pattern=..., replacement=..., x=...), grep(pattern, text), strsplit(x, split), ...}
x <- seq(-pi, pi, by=0.01)
y <- sin(2*x)

plot(x, y)

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

```r
> 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

<table>
<thead>
<tr>
<th></th>
<th>the</th>
<th>and</th>
<th>a</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moby Dick</td>
<td>0.051</td>
<td>0.041</td>
<td>0.025</td>
<td>0.023</td>
</tr>
<tr>
<td>Tom Sawyer</td>
<td>0.052</td>
<td>0.037</td>
<td>0.022</td>
<td>0.024</td>
</tr>
</tbody>
</table>

> barplot(word.freq,
  + col=c('red', 'blue'),
  + legend=rownames(word.freq),
  + beside=TRUE)
```
Graphics 101 - histogram

- Histogram for (numeric values only)
- Example:

```r
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

```r
> 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 following data frame:

   ```r
   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 coordinates (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

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

• while loop

```r
> 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 smaller than ",b,"\n")
  1.661342 is smaller than  1.043969
  ```
User defined functions

• General syntax:

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

• Example:

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

```r
x <- seq(-2,5, by=0.1)
y <- my.polynom(x)
plot(x,y, type="l")
```
Writing to a file

• Example:

```r
> 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 occurring words (i.e. stop-words)
• To identify an author of an 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-exercise.
• 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)
1. Goto the webpage http://www.smiffy.de/dbkda-2016
2. Evaluate the function `readBookFromFile(...)` (Appendix A) in your R-environment.
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 evaluated function and examine the result.
5. Count the number of occurrence 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 barplots 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)
- Parallelization (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

- Data Camp (very good online courses), Overview: https://www.datacamp.com/getting-started?step=2&track=r
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.character`, `unlist`, `as.matrix`, `as.data.frame`, `as.Date`

- Statistic
  - `rnorm`, `runif`, `rbinom`, ...