Scientific programming in R

Course material: http://izbifs.izbi.uni-leipzig.de/~wirth/R/Day4.pdf

Dr. Henry Löffler-Wirth
Maria Schmidt
Dr. Lena Sünke Mortensen

Interdisciplinary Center for Bioinformatics - IZBI,
University Leipzig
Organization


Day 1: R syntax, very basic programming techniques (vectors, loops, conditions)

Day 2: Data handling, data types and simple statistics

Day 3: Visualization basics and programming projects

Day 4: Apply-functions, graph handling, string functions

Day 5: Interactive Apps with Shiny
apply-family functions

apply, lapply, sapply, tapply are very handy functions. We start with **apply**: do some function for each row/column of a matrix

```r
> mat <- matrix( rnorm( 1000 ), nrow = 100 )

> apply( mat, 1, mean )
> rowMeans( mat )
```

```
[[1]]
[1] 0.3281474407 0.3258073136 -0.2801949620 -0.0303225630 -0.5268319537 -0.0907618633 0.1561192737 0.3025790446 -0.067637164 0.1528199955
[2] 0.3163809014 0.5947037489 -0.2571121778 0.4314409409 -0.1800393653 0.6935670923 0.2506651276 0.0031024802 0.2865546474 0.0435411111
[3] 0.3109472412 0.6288592566 0.5130210289 0.1034204362 -0.1679837183 0.1722670383 -0.0765389309 -0.4732221500 -0.1908464139 -0.2325302063
[4] -0.1846368202 0.1593712529 0.2321219619 -0.1565868388 -0.5592704717 -0.4078618108 -0.1396182160 -0.2692721473 -0.3524011183 0.1239756272
[5] -0.4221879790 0.2689086389 -0.0008226471 -0.0967608157 0.2852251431 -0.5600432729 -0.2120114761 -0.0125525481 -0.1795197745 0.1025574563
[6] -0.0183340363 0.2951102737 0.3138281526 -0.2769136864 0.0479505878 -0.2273378048 -0.0550463112 0.0005401879 -0.0736397542 0.2785794242
[7] 0.1799154871 -0.2732950451 0.2727882612 -0.2065996010 -0.4080205840 0.0362590946 -0.1254734597 0.0036140078 -0.0520611753 0.220402113
[8] -0.0852742581 0.1353503639 -0.5265431490 -0.5589758805 -0.4108971901 -0.0726597737 0.3362318913 0.4303352837 -0.2158480249 -0.0342293468
[9] 0.3152801693 -0.2896022884 -0.0559510452 -0.6124269126 -0.1262759514 -0.0858034004 -0.133119842 -0.5351418652 0.1785738355 0.1894726583
[10] -0.4912023808 0.3430814058 -0.0276810357 0.4184412656 0.2068321549 0.1020772172 0.1887167500 0.3587692342 -0.2858820803 0.4443388986
```

```r
> apply( mat, 2, sum )
> colSums( mat )
```

```
[1] -0.3875725 0.1606165 2.1200624 11.3178094 6.4623649 -0.4573779 -11.6417783 -7.9389483 2.3736285 -2.8495978
```

**very fast** built-in functions:

*colMeans, rowMeans, colSums, rowSums*
**apply-family functions**

*apply* can return a vector, a matrix, or a list; depending on content of *return(...)*:

```r
> apply(mat, 2, function(x) x[1] )
```

\[
\begin{bmatrix}
1.0893682 & 1.9950244 & 0.5048644 & -0.3934983 & 0.6426558 & -0.2890649 & -0.1323788 & 1.3712561 & 1.8578147 & 0.1794705
\end{bmatrix}
\]

\[return\text{(single value)} \rightarrow apply\text{ provides a vector}\]

```r
> apply(mat, 2, function(x) x[1:3] )
```

\[
\begin{bmatrix}
1.0893682 & 1.9950244 & 0.5048644 & -0.3934983 & 0.6426558 & -0.2890649 & -0.1323788 & 1.3712561 & 1.8578147 & 0.1794705 \\
0.1236050 & -1.7351955 & 1.5658201 & -1.1463340 & -1.5822586 & -0.7327124 & -0.9499925 & -0.5736847 & 0.1050991 & -0.4211171 \\
0.6294236 & -0.7869878 & 1.6373986 & -0.07240029 & -0.2312011 & -1.2577019 & -0.2702612 & -0.0807762 & 0.6843487 & -1.9705109
\end{bmatrix}
\]

\[return\text{(vector)} \rightarrow apply\text{ provides a matrix, if vector length is constant}\]

```r
> apply(mat, 2, function(x) list(x[1:3] ) )
```

\[
\begin{bmatrix}
1.0893682 & 0.1236050 & 0.6294236 \\
0.1236050 & -1.7351955 & -0.7869878 \\
0.6294236 & 1.5658201 & 1.6373986
\end{bmatrix}
\]

\[return\text{(otherwise)} \rightarrow apply\text{ provides a list, if return\text{(} is e.g. vector of changing length or a list}\]
apply-family functions

> apply( mat, 2, head )

form: `apply( matrix, margin, function, ...)`

... → optional arguments for `function`
e.g. `head(x, n)` → `x` provided automatically, `n` can be given

> apply( mat, 2, head, 3 )

```r
[1,] 1.0893682 1.9950244 0.5048644 -0.39349833 0.6426558 -0.2890649 -0.1323788 1.37125608 1.85781472 0.1794705
[2,] 0.1236050 -1.7351955 1.5658201 -1.14633430 -1.5822586 -0.7327124 -0.9499925 -0.57368471 0.10509909 -0.4211171
[3,] 0.6294236 -0.7869878 1.6373986 -0.07240029 -0.2312011 -1.2577019 -0.2702612 -0.08077620 0.68434868 -1.9705109
[4,] -1.4237321 -0.3047625 1.3269676 -0.5084513 -0.4549337 -0.2067314 -1.4021740 0.96935900 -0.51273922 -0.7488544
[5,] 1.1237304 -0.2651737 1.7613133 -0.55301124 1.3141597 -0.6094446 0.8635372 0.04296008 0.07419626 0.5263411
[6,] 0.4127814 0.005006386 1.1285000 0.1936617 -0.9434470 0.1260797 -2.1535970 -0.77144649 -1.29052846 1.7502699
```

```r
[1,] 1.0893682 1.9950244 0.5048644 -0.39349833 0.6426558 -0.2890649 -0.1323788 1.37125608 1.85781472 0.1794705
[2,] 0.1236050 -1.7351955 1.5658201 -1.14633430 -1.5822586 -0.7327124 -0.9499925 -0.57368471 0.10509909 -0.4211171
[3,] 0.6294236 -0.7869878 1.6373986 -0.07240029 -0.2312011 -1.2577019 -0.2702612 -0.08077620 0.68434868 -1.9705109
[4,] -1.4237321 -0.3047625 1.3269676 -0.5084513 -0.4549337 -0.2067314 -1.4021740 0.96935900 -0.51273922 -0.7488544
[5,] 1.1237304 -0.2651737 1.7613133 -0.55301124 1.3141597 -0.6094446 0.8635372 0.04296008 0.07419626 0.5263411
[6,] 0.4127814 0.005006386 1.1285000 0.1936617 -0.9434470 0.1260797 -2.1535970 -0.77144649 -1.29052846 1.7502699
```
apply-family functions

**lapply**: *do some function for each element of a list or vector, returns a list*

```r
> l <- lapply( c(1:5), function(x) return(x) )
build a list with 5 elements, each containing 1 number

> str(l)
List of 5
$ : int 1
$ : int 2
$ : int 3
$ : int 4
$ : int 5

> l <- lapply(l, function(x) return( c( 1:x ) )
create elements containing numbers 1:x

> l
[[1]]
 [1] 1

[[2]]
 [1] 1 2

[[3]]
 [1] 1 2 3

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

[[5]]
 [1] 1 2 3 4 5
```
apply-family functions

> lapply( l, length )

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

> lapply( l, tail, 2 )

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

form: \texttt{lapply( list, function, ...)}

additional parameter(s) for the functions can be given
apply-family functions

**sapply:** do some function for each element of a list or vector, returns a vector, matrix or list

```r
> sapply( l, length )
[1] 1 2 3 4 5
```

```r
> sapply( l, function(x) x[ c( 1:2 ) ] )
[1,]  1  1  1  1  1
[2,]  NA  2  2  2  2
```

```r
> sapply( l, function(x) x[ c( 1:length(x) ) ] )

```

length of return values differs  → list required

another way to generate square numbers,

```r
> sapply( c(1:100), function(x) x^2 )
```

!! **sapply** returns values directly, for stores them into predefined variables (e.g. vectors) !!
apply-family functions

`tapply`: do some function for each group of elements of a vector, returns a vector or list (no matrix)

<table>
<thead>
<tr>
<th>vector</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

```r
> tapply( c(1:100), rep(c(1:4),25), mean )
```

form: `tapply(vector, indices, function, ... )`
- indices do not have to be ordered
- returns one value for each unique index

average expression values for the replicate groups

```r
> expression <- rnorm(1:100)
> replicates <- rep( c(1:5), each = 20 )

> tapply( expression, replicates, mean )
```

0.09992157 -0.11890472 0.23103137 0.41623656 -0.30262511
apply-family functions

**by**: do some function for each group of rows of a matrix, returns a list (->'matrix-tapply')

form: \( \text{by( matrix, row\_indices, function, ... )} \)

calls *function* for sub-matrices:
Scope and variable visibility

R is convenient regarding types (type definition, type changes) and visibility (global variables)

Some theory… there are three types of variables in R:

• Formal parameters: parameters of functions, locally visible

• Local variables: created within a function, locally visible

• Free variables: all other

```r
> z <- 3
> f <- function( x )
  {
    y <- 2 * x
    print( z )
  }
> f( 1 )
[1] 3
> f( 60 )
[1] 3
```

x: formal parameter
y: local variable
z: free variable
Scope and variable visibility

R is convenient regarding types (type definition, type changes) and visibility (global variables)

Some theory… there are three types of variables in R:

• Formal parameters: parameters of functions, locally visible
• Local variables: created within a function, locally visible
• Free variables: all other

> z <- 3
> f <- function( x )
  {
    y <- 2 * x
    z <- 5
    z <<- 99
    print( z )
  }
> z
[1] 3
> f( 1 )
[1] 5
> z
[1] 99

'Superassignment' operator '<<-' used to write into hidden variables

'free z' is hidden behind 'local z', see it as a completely other variable which just happen to have the same name → calling functions has no side effects, except:
Scope and variable visibility

R utilizes hierarchical organization of local variables:

```r
> f1 <- function()
{ 
  f2 <- function()
  { 
    l.f2 <- 4
    print( exists("gv.1") )  # [1] TRUE
    print( exists("l.f1") )  # [1] TRUE
    print( exists("l.f2") )  # [1] TRUE
  }
  l.f1 <- 3
  print( exists("gv.1") )  # [1] TRUE
  print( exists("l.f1") )  # [1] TRUE
  print( exists("l.f2") )  # [1] FALSE
  f2()
}
> gv.1 <- 1
> print( exists("gv.1") )  # [1] TRUE
> print( exists("l.f1") )  # [1] FALSE
> print( exists("l.f2") )  # [1] FALSE
> f1()
```
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to create, modify and analyze graph structures.

A graph is an **ordered pair** $G=(V, E)$ comprising of a set of vertices $V$ and a set of edges $E$, which are 2-element subsets of $V$. 

**undirected graph**

**directed graph**
A brief introduction to \textit{igraph} package

\textit{igraph} provides comprehensive functionalities to create, modify and analyze graph structures

Download of the package to your system and attach it to your open R session

> install.packages("igraph")

> library("igraph")
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to **create**, modify and analyze graph structures.

Option 1: Create graph manually and add nodes and edges:

```r
> g <- make_empty_graph( directed=TRUE )
> g <- add_vertices( g, 4 )
> plot( g )
> g <- add_edges( g, c(1,2, 3,4, 1,4) )
> plot( g )
> g   # = print(g) = print.igraph(g)
```

- Create an instance of *igraph* class
- Add vertices; additional attributes can be given
- *plot* method for *igraph* class
- Add edges; vertex indices are given as vector
**A brief introduction to igraph package**

*igraph* provides comprehensive functionalities to **create**, modify and analyze graph structures

Option 2: Create graph from adjacency matrix

\[
\begin{pmatrix}
0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{pmatrix}
\]

\[
> \text{adj}
\]

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

\[
[1, ] 0 1 0 1
[2, ] 0 0 0 0
[3, ] 0 0 0 1
[4, ] 0 0 0 0
\]

\[
> \text{g} <- \text{graph_from_adjacency_matrix}(\text{adj})
\]

{0, 1} determines if pair of nodes is connected; symmetrical for undirected graphs; weighted graphs can also be defined

Create an instance of *igraph* class from adj. matrix

\[
> \text{plot}(\text{g})
\]

![Graph visualization](image)
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to **create**, modify and analyze graph structures

Option 3: Use one of the numerous graph models provided by *igraph*:

> `g <- make_full_graph(40)`

> `g <- make_star(40)`
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to **create**, modify and analyze graph structures

Option 3: Use one of the numerous graph models provided by *igraph*:

```r
> g <- make_tree(40, children = 3)
> g <- sample_pa(n=100, directed=F) # scale-free
```
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to create, **modify** and analyze graph structures

```r
> g <- make_empty_graph( directed=TRUE )
> g <- add_vertices( g, 4, name=LETTERS[1:4] )
> g <- add_edges( g, c( 1,2, 3,4, 1,4 ) )

> V(g)
+ 4/4 vertices, named, from 5360dee:
  [1] A B C D
> E(g)
+ 3/3 edges from 5360dee (vertex names):
  [1] A->B C->D A->D

> g <- delete_vertices( g, 2 )
≡ g <- delete_vertices( g, "B" )

> g <- delete_edges( g, 2 )
≡ g <- delete_edges( g, "A|D" )
```

Adding nodes/edges

Inspecting nodes (vertices) and edges of g

Removing nodes/edges by index or by name
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to create, **modify** and analyze graph structures.

Set operations on graph level (teaser only):

- Union of graphs: \( g_1 \cup g_2 \)
- Intersection of graphs: \( g_1 \cap g_2 \)
- Difference of graphs: \( g_1 \setminus g_2 \)
igraph provides comprehensive functionalities to create, modify and **analyze** graph structures

```r
> ba <- sample_pa(n=100, directed=F)
> edge_density( ba )
[1] 0.02
> diameter( ba )
[1] 10
> get_diameter( ba )
+ 11/100 vertices, from 94f2d57:
[1] 53 36 33 15 8 3 1 9 18 51 65
> degree( ba )
[1]  9  2 11  3  8  3  3  9  4  6  1  3  1
[24]  1  2  1  3  1  3  2  2  1  4  1  1  2
```

**Proportion of present edges from all possible edges**

**Longest direct path between any two nodes**

**Returns node IDs of the diameter**

**Returns number of edges for each node**
A brief introduction to \textit{igraph} package

\textit{igraph} provides comprehensive functionalities to create, modify and \textbf{analyze} graph structures.

\begin{verbatim}
> rg <- graph_from_adjacency_matrix( random.adj, mode="undirected" )
> mean_distance( rg )
[1] 1.873684
> distances( rg )
> shortest_paths( rg, from = V(rg)[11], to = V(rg)[3], output = "both")
\end{verbatim}

- Mean of the shortest distance between each pair of nodes
- Length of all shortest paths
- Returns list of nodes and edges of the shortest path
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to create, modify and **analyze** graph structures

```r
> cliques( rg )
[[1]]
+ 3/20 vertices, from e72f11d:
  [1]  2  9 14

[[2]]
+ 2/20 vertices, from e72f11d:
  [1]  2 14

[[3]]
+ 2/20 vertices, from e72f11d:
  [1]  2 20

> largest_cliques( rg )
[[1]]
+ 4/20 vertices, from e72f11d:
  [1] 14  2  3  9
```

Detection of complete subgraphs

Teaser: definition of communities (densely connected nodes)
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to create, modify and analyze ... and **plot**!

Almost everything in the graph plot can be adjusted using node and edge parameters.

Major parameters (see `?igraph.plotting` for complete list):

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex.color</td>
<td>edge.color</td>
</tr>
<tr>
<td>vertex.frame.color</td>
<td>edge.width</td>
</tr>
<tr>
<td>vertex.size</td>
<td>edge.arrow.size</td>
</tr>
<tr>
<td>vertex.label</td>
<td>edge.arrow.width</td>
</tr>
<tr>
<td>vertex.label.cex</td>
<td>edge.label</td>
</tr>
<tr>
<td>....</td>
<td>edge.label.cex</td>
</tr>
</tbody>
</table>

...
A brief introduction to *igraph* package

*igraph* provides comprehensive functionalities to create, modify and analyze ... and **plot**!

Option 1: Provide parameters when calling `plot(igraph.instance)`

Option 2: Storing the parameters within the igraph object. Not today ;)

```R
> plot( rg )
> plot( rg, vertex.size=7.5, vertex.color="cornflowerblue", vertex.label=NA )
> plot( rg, edge.width=2, edge.color="gold" )
```
Programming task I

**Six degrees of separation** is the idea that each of us is six or fewer steps (friends) away from everybody else ("a friend of a friend" chain).

- Generate a graph of 100 individuals (=nodes).
- For every individual, add a 'friendship relation' (=edge) to a randomly selected other individual.
- Visualize the network.
- Extend this approach to 10 friends per individual.
- Make the world bigger (1,000 individuals). Is this a valid model to investigate the six-degree assumption?
- Are there cliques in the network? What is the average and what the longest "friend of a friend" chain length? *Hint: handy commands were shown on the slides.*
- How are these parameters scaling? Check for a network of 2,000 individuals.
- Generate a small-world network of using *igraph* function *sample_smallworld()*. *Hint: Use example call from the function's help, and increase size from 100 to 2,000 individuals.*
- Check the abovementioned parameters for this model. Is it more appropriate?
Programming task I

Generate a graph of 100 individuals (=nodes).

```r
> library(igraph)
> g <- make_empty_graph( directed=FALSE )
> g <- add_vertices( g, 100 )
```

For every individual, add a 'friendship relation' (=edge) to a randomly selected other individual.

```r
> for( i in 1:vcount(g) )
> {
>     g <- add_edges( g, c( i, sample( setdiff( c(1:vcount(g)), i ), 1 ) ) )
> }
```

Visualize the network.

```r
> plot( g )
```
Programming task I

Extend this approach to 10 friends per individual.

```
> for( i in 1:vcount(g) )
  {
    for( j in 1:10 )
    {
      g <- add_edges( g, c( i, sample( setdiff( c(1:vcount(g)), i ), 1 ) ) )
    }
  }
```

Make the world bigger (1,000 individuals).

```
> g <- make_empty_graph( directed=FALSE )
> g <- add_vertices( g, 1000 )

> for( i in 1:vcount(g) )
  {
    for( j in 1:10 )
    {
      g <- add_edges( g, c( i, sample( setdiff( c(1:vcount(g)), i ), 1 ) ) )
    }
  }
```
Programming task I

Are there cliques in the network? What is the average and what the longest "friend of a friend" chain length?

```r
> head(largest_cliques(g))
[[1]]
+ 4/1000 vertices, from de2d647:
[1] 942 471 55 698
[[2]]
+ 4/1000 vertices, from de2d647:
[1] 895 311 204 255
[[3]]
+ 4/1000 vertices, from de2d647:
[1] 777 259 617 696
[[4]]
+ 4/1000 vertices, from de2d647:
[1] 640 204 733 764

> mean_distance(g)
[1] 2.646589

> diameter(g)
[1] 4
```
Programming task I

How are these parameters scaling? Check for a network of 2,000 individuals.

```r
> g <- make_empty_graph( directed=FALSE )
> g <- add_vertices( g, 10000 )

> for( i in 1:vcount(g) ) {
    friends <- sample( setdiff( c(1:vcount(g)), i ), 10 )
    for( j in friends ) {
        g <- add_edges( g, c( i, j ) )
    }
}

> mean_distance( g )
[1] 3.130415
> diameter( g )
[1] 4
```
String handling & manipulation

Remember the 'Loading a data matrix' lesson?

```r
> mat <- read.csv2( "bitcoin.csv", row.names = 1, stringsAsFactors = FALSE )
> mat <- apply( mat, 2, function(x) gsub("."","",x,fixed=TRUE) )
> mat <- apply( mat, 2, function(x) sub("",".",x,fixed=TRUE) )
> mat <- apply( mat, 2, as.numeric )

> mat

<table>
<thead>
<tr>
<th></th>
<th>Offen</th>
<th>Hoch</th>
<th>Tief</th>
<th>Geschlossen</th>
<th>Volumen</th>
</tr>
</thead>
<tbody>
<tr>
<td>21. Nov 18</td>
<td>4465.54</td>
<td>4675.73</td>
<td>4343.98</td>
<td>4602.17</td>
<td>6120120000</td>
</tr>
<tr>
<td>20. Nov 18</td>
<td>4863.93</td>
<td>4951.61</td>
<td>4272.11</td>
<td>4451.87</td>
<td>8428290000</td>
</tr>
<tr>
<td>19. Nov 18</td>
<td>5620.78</td>
<td>5620.78</td>
<td>4842.91</td>
<td>4871.49</td>
<td>7039560000</td>
</tr>
</tbody>
</table>
```

Other data source, same pain:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DLBCL</td>
</tr>
<tr>
<td>2</td>
<td>DLBCL, B-Zell-Lymphom, hochmaligne</td>
</tr>
<tr>
<td>3</td>
<td>DLBCL, sekretorisch differenzierter DLBCL</td>
</tr>
<tr>
<td>4</td>
<td>DLBCL, sekundäres DLBCL NOS</td>
</tr>
<tr>
<td>5</td>
<td>DLBCL, DLBCL, B-Zell Lymphom, plasmablastisch</td>
</tr>
<tr>
<td>6</td>
<td>DLBCL, DH-Lymphom, plasmablastisch</td>
</tr>
<tr>
<td>7</td>
<td>DLBCL, atypischer Burkitt oder DLBCL NOS</td>
</tr>
<tr>
<td>8</td>
<td>BL, BLoder oder DLBCL</td>
</tr>
<tr>
<td>9</td>
<td>BL, BL(?Kryo)</td>
</tr>
<tr>
<td>10</td>
<td>BL, atypischer Burkitt oder DLBCL NOS</td>
</tr>
</tbody>
</table>

[4.465, 54]
String handling & manipulation

> load(url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData"))  # !not! names

→ contains 50 most popular baby names for girls and boys, resp.

→ needs some revision
String handling & manipulation

> load( url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData") )

Basic functions:

> nchar( nmes )
[1]  8  8  9  8  9  8  9  9 11 10 10  9  8
[32] 10 11  8 11 10  9 11  9 10  8  8 13 11 15
[63] 11 12  9 11  8 10 10 10  9  9 10  9 10  9 11
[94]  9  9  9 11 14 12 10

Count letters in character objects: `nchar(...)`

> tolower( nmes )
[1] "1_m_lobo"   "2_m_levi"   "3_m_elias"
[6] "6_m_theo"   "7_m_emil"   "8_m_jonas"
[11] "11_m_markus" "12_m_lukas" "13_m_felix"
[16] "16_m_julian" "17_m_noah" "18_m_matteo"

Convert all letters to lower case: `tolower(...)`

> toupper( nmes )
[1] "1_M_LOBO"   "2_M_LEVI"   "3_M_ELIAS"
[6] "6_M_THEO"   "7_M_EMIL"   "8_M_JONAS"
[16] "16_M_JULIAN" "17_M_NOAH" "18_M_MATTEO"

Convert all letters to upper case: `toupper(...)`
String handling & manipulation

> load( url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData") )

Functions for character assembly/concatenation:

> paste( "xyz", nmes[1:8], 8:1 )
[1] "xyz 1_m_Lobo 8" "xyz 2_m_Levi 7"
[3] "xyz 3_m_Elias 6" "xyz 4_m_Liam 5"
[5] "xyz 5_m_Anton 4" "xyz 6_m_Theo 3"
[7] "xyz 7_m_Emil 2" "xyz 8_m_Jonas 1"

> paste( "xyz", nmes[1:8], 8:1, sep="/" )
[1] "xyz/1_m_Lobo/8" "xyz/2_m_Levi/7"
[3] "xyz/3_m_Elias/6" "xyz/4_m_Liam/5"
[5] "xyz/5_m_Anton/4" "xyz/6_m_Theo/3"
[7] "xyz/7_m_Emil/2" "xyz/8_m_Jonas/1"

> paste0( "xyz", nmes[1:8], 8:1 )
[1] "xyz1_m_Lobo8" "xyz2_m_Levi7"
[3] "xyz3_m_Elias6" "xyz4_m_Liam5"
[5] "xyz5_m_Anton4" "xyz6_m_Theo3"
[7] "xyz7_m_Emil2" "xyz8_m_Jonas1"

> paste( nmes[1:8], collapse="|" )
[1] "1_m_Lobo|2_m_Levi|3_m_Elias|4_m_Liam|5_m_Anton|6_m_Theo|7_m_Emil|8_m_Jonas"

Assembling strings using `paste(...)`:
- works for vectors
- recycling of single-value objects
- content taken from variables
- numeric variables converted to character
- standard separator: ‘ ’ (space); can be adjusted

Wrapper for `paste(..., sep="" )`

Pasting elements of a vector to one long string
`collapse` determines separator between elements
String handling & manipulation

> load(url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData"))

Functions for character disassembly:

> substr(nmes, 4, 7)
[1] "_Lob" "_Lev" "_Eli" "_Lia" "_Ant" "_The"
[12] "m_Lu" "m_Fe" "m_Mi" "m_Le" "m_Ju" "m_No"
[23] "m_Fi" "m_Ja" "m_Mo" "m_Ph" "m_Sa" "m_Aa"
[34] "m_Be" "m_Da" "m_Le" "m_Lu" "m_Ja" "m_Ki"

> substr(nmes, 4, 100000000)
> substr(nmes, 4, nchar(nmes))
[1] "_Lobo" "_Levi" "_Elias"
[11] "m_Markus" "m_Lukas" "m_Felix"
[16] "m_Julian" "m_Noah" "m_Matteo"

→ need for more flexible tools to find & manipulate substrings

→ grep(), strsplit(), sub() using regular expressions
String handling & manipulation

> load(url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData"))

Functions for pattern matching:

> grep("N", nmes)
[1] 17 74 79 92 93 94

> grepl("N", nmes)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[2] TRUE

> nmes[ grep("N", nmes) ]
[1] "17_m_Noah" "24_f_Nora" "29_f_Noah" "42_f_Nele" "43_f_Nina" "44_f_Nane"

- `grep(pattern, x)` provides indices of x matching
- `grepl(pattern, x)` provides logical matching vector
  - `value=TRUE`: give values instead of indices
  - `ignore.case=TRUE`: matches lower & upper case
String handling & manipulation

> load(url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData"))

Functions for character disassembly:

> strsplit(nmes, "_")

```
[[1]]
[1] "1"  "m"  "Lobo"

[[2]]
[1] "2"  "m"  "Levi"

[[3]]
[1] "3"  "m"  "Elias"

[[4]]
[1] "4"  "m"  "Liam"

[[5]]
[1] "5"  "m"  "Anton"

[[6]]
[1] "6"  "m"  "Theo"
```

> sapply(strsplit(nmes, "_"), function(x) x[3])

```
[1]  "Lobo"  "Levi"  "Elias"  "Liam"  "Linus"  "Paul"  "Markus"  "Lukas"  "Noah"  "Matteo"  "Michael"  "Leon"  "Moritz"  "Philipp"  "Samuel"  "Aaron"
```

`strsplit(x, pattern)`:  
- creates a list, one element per element in x  
- divides each element of x at each `pattern`  
- `pattern` character(s) are removed
String handling & manipulation

```r
> load(url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData"))
```

Functions for character manipulation:

```r
> sub("_", "%", nmes)
[1] "1%m_Lobo"       "2%m_Levi"   "8%m_Jonas"   "13%m_Felix"   "18%m_Matteo"
[6] "6%m_Theo"       "7%m_Emil"    "15%m_Markus"
[11] "11%m_Markus"    "12%m_Lukas"
[16] "16%m_Julian"    "17%m_Noah"
```

Substitution of substrings: `sub(pattern, replace, x)`
→ replaces first occurrence of `pattern` in `x`

```r
> gsub("_", "%", nmes)
[1] "1%m$m$Lobo"       "2%m$m$Levi"   "8%m$m$Jonas"   "13%m$m$Felix"   "18%m$m$Matteo"
[5] "5%m$m$m$Anton"    "6%m$m$m$Theo"  "7%m$m$m$Emil"   "11%m$m$m$Markus"
[9] "9%m$m$m$m$Linus"  "10%m$m$m$m$Paul" "15%m$m$m$m$Leo"
[13] "13%m$m$m$m$Felix" "14%m$m$m$m$Milo"
```

`gsub(pattern, replace, x)`
→ replaces all occurrences of `pattern` in `x`

```r
> sub("^[0-9]+_[mf]_", "", nmes)
[1] "Lobo"       "Levi"       "Elias"       "Lukas"       "Leon"
[9] "Linus"      "Paul"       "Markus"      "Leon"
[17] "Noah"       "Matteo"     "Michael"     "Leon"
[25] "Moritz"     "Philipp"    "Samuel"      "Aaron"
```

`pattern` provided as regular expression
String handling & manipulation

Regular expressions: special grammar rules describing patterns

```
> grep( "^5", nmes, value=TRUE )
[1] "5_m_Anton" "50_m_Louis" "5_f_Emma" "50_f_Elina"

> grep( "l$", nmes, value=TRUE )
[1] "7_m_Emili" "10_m_Paul" "19_m_Michael" "27_m_Samuel" "35_m_Daniel"

> grep( "[xy]", nmes, value=TRUE )
[1] "13_m_Felix" "31_m_Alexander" "42_m_Max" "46_m_Maximilian" "39_f_Elisa"

> sub( "[0-9]", "", nmes )
[1] "_m_Lobo" "_m_Levi" "_m_Emili" "_m_Jonas"
[13] "3_m_Felix" "4_m_Milo" "5_m_Leo"
[19] "9_m_Michael" "0_m_Leon" "1_m_Levi"

> sub( "[0-9]++", "", nmes )
[1] "_m_Lobo" "_m_Levi" "_m_Elias"
[7] "_m_Emili" "_m_Jonas" "_m_Linus"
[13] "_m_Felix" "_m_Milo" "_m_Leo"
[19] "_m_Michael" "_m_Leon" "_m_Levin"
```

Anchors:
^ matches start position, $ matches string ending

Sets & ranges: [characters]
[ae] matches a OR e, [0-9] [a-z] [A-Z] match ranges

Repetitions: [characters]* or [characters]+
* matches zero or more repeats, + at least one
String handling & manipulation

Regular expressions: special grammar rules describing patterns

> sub("^[0-9]+_[mf]_", "", nmes)
  [1] "Lobo"      "Levi"      "Elias"
  [9] "Linus"     "Paul"      "Markus"
 [17] "Noah"      "Matteo"    "Michael"
 [25] "Moritz"    "Philipp"   "Samuel"

→ ^[0-9]+ matches preceding numbers (one or two digits)

→ [mf] matches m or f gender labels

Further expressions comprise alterations, groups, backreferences, non-printables and other
String handling & manipulation

Examples:

Matching a date in yyyy-mm-dd format, with variable separator:

```r
\d\d[-/.](0[1-9]1[012])[-/.](0[1-9]1[2]0-9]3[01])$```

RFC standard matches/describes valid email addresses:

```r
```
Programming task II

Load the name list: `load( url("http://izbifs.izbi.uni-leipzig.de/~wirth/R/nmes.RData") )`

- Extract the first letters of the names. How are they distributed? (use descriptive statistics)
- How many girl’s names end with an ‘a’?
- Are there boy’s and girl’s names with same terminal letter? If so, list them!
- optional A*: List all pairs of names differing in only one letter!
  Hint: Iterate over all names. For each name iterate over the letters. ‘.’ matches any letter in `grep()`.
- optional B**: Generate a transition matrix for the letters in the names. It contains frequencies of letter combinations, e.g. after ‘E’ there is 12 times ‘L’.
  Hint: Convert letters to upper case. Initialize a matrix representing all unique letters in the names as rows (from-letter) and columns (to-letter). Add a column $, which is ‘to-letter’ for terminal letters. Iterate over all names. For each name iterate over the letters to fill the matrix accordingly.
- optional B***: Implement a name generator based on the transition matrix (finite automaton):
  Chose a starting letter from overall letter probability distribution. Successively sample subsequent letters based on the transition frequencies/probabilities.
Programming task II

- Extract the first letters of the names. How are they distributed? (use descriptive statistics)

```r
> L1 <- substr( sub( "^\d+_[mf]_", "", nmes), 1, 1 )
> sort( prop.table( table(L1) ), decreasing=T )
L1
   A   B   D   E   F   I   J   K   L   M   N   O   P   S   T   V
0.27 0.18 0.10 0.09 0.08 0.06 0.05 0.03 0.03 0.02 0.02 0.02 0.02 0.01 0.01 0.01
> barplot( table(L1) )
```
Programming task II

- How many girl’s names end with an ‘a’?

```r
> fnames <- grep("_f_", nmes, value=TRUE)
> fnamesLast <- substr(fnames, nchar(fnames), nchar(fnames))
> sum(fnamesLast == "a")
[1] 37
> barplot(table(fnamesLast))
```

```r
> fnames
[1] "1_f_Emilia"  "2_f_Ella"   "3_f_Lena"
[7] "7_f_Leonie"  "8_f_Lea"    "9_f_Lina"
[13] "13_f_Sophie" "14_f_Johanna" "15_f_Lea"
[19] "19_f_Aлина" "20_f_Maria"  "21_f_Leon"
[25] "25_f_Luisa"  "26_f_Leni"   "27_f_Lina"
```
Programming task II

• Are there boy’s and girl’s names with same terminal letter? If so, list them!

```r
> fNmes <- grep( "_f_", nmes, value=TRUE)
> fNmesLast <- substr( fNmes, nchar(fNmes), nchar(fNmes) )
> mNmes <- grep( "_m_", nmes, value=TRUE)
> mNmesLast <- substr( mNmes, nchar(mNmes), nchar(mNmes) )
> intersect( fNmesLast, mNmesLast )
[1] "a" "i" "h" "n"
```