Introduction to R

Anders Stockmarr
Course developers: Anders Stockmarr, Elisabeth Wreford Andersen

DTU Department of Applied Mathematics and Computer Science
Section for Statistics and Data Analysis
Technical University of Denmark
anst@dtu.dk

January 3rd, 2025

DTU Compute
Department of Applied Mathematics and Computer Science

Plan for today

- Introduction to R
- Data management
- Loops
- Graphics

Outline

Introduction to **R**

- **R** is a programming *language* and a programming *environment*.
- It is Free! Developed by users under a GNU license.
- Runs on a variety of platforms including Windows, Unix and MacOS. You can even get it for Android.
- Allows for fast implementation of new methods by user demand through packages.
- **R** has state-of-the-art graphics capabilities.

Advantages of **R**

Frank Harrel (my highlighting):

- "One point that hasn't been made very explicitly is one of the greatest advantages of **R**:

  Getting your work done better and in less time.

  Hundreds of companies hire a multitude of SAS programmers to write code in an archaic language, the SAS macro language. I believe there is a real cost savings from R because of its value as a **data analysis**, **data manipulation**, and **graphics** environment. Instead of programming using an indirect syntax manipulation environment (SAS macros), in R you can program in a dynamic data-sensitive framework".

That was more than 10 years ago. Things have progressed since...

Base **R**

- Base **R** and most **R** packages are available for download at the Comprehensive R Archive Network (CRAN).
- http://www.cran.r-project.org
- Base **R** includes basic data management, analysis and graphics tools.
- For non-specialized tasks, Base **R** is all you need.
- Specialized tasks may be handled by *packages*.
- We will download, install and use many packages during the course.
- Packages are not all very well-documented (depends on the contributor).
- Want to be sure about what your program does?
  - Use well-established packages only;
  - or write your own code.

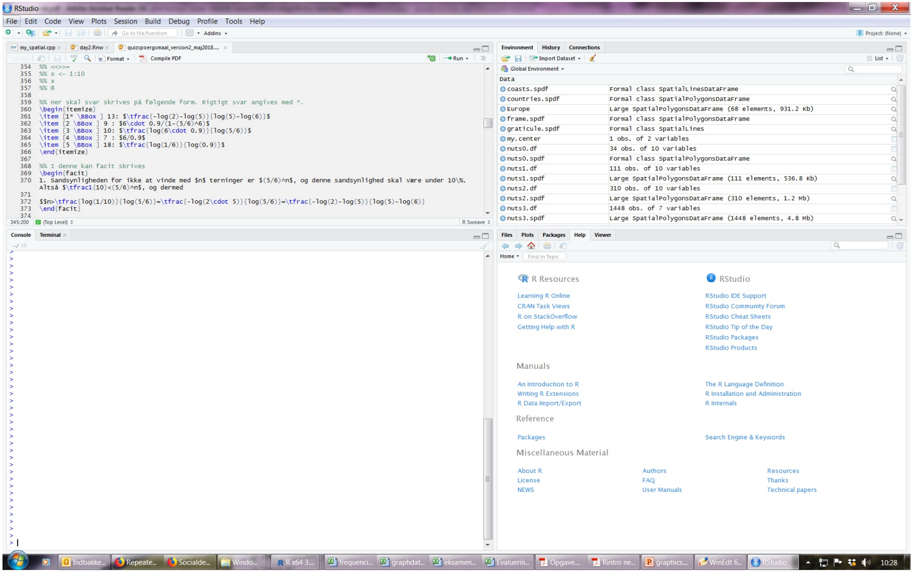
RStudio

- You can work directly in **R**.
- Many prefer another front end (GUI, **G**raphical **U**ser **I**nterface).
- We will use RStudio.
- Download from http://www.posit.co/

RStudio

- The GUI RStudio has 4 windows.
- One for writing the commands (the "script").
    - Use script for reproducibility.
- One for results and interactive use.
- One for plots, help and packages.
- One showing which objects are resident in the **R** memory.

RStudio



```
%% <<>>=
%% x <- 1:10
%% x
%% @

%% Her skal svar skrives på følgende form. Rigtigt svar angives med *.
\begin{itemize}
\item [1* \BBox ] 13: $\tfrac{-log(2)-log(5)}{log(5)-log(6)}$
\item [2 \BBox ] 9 : $6\cdot 0.9/(1-(5/6)^6)$
\item [3 \BBox ] 10: $\tfrac{log(6\cdot 0.9)}{log(5/6)}$
\item [4 \BBox ] 7 : $6/0.9$
\item [5 \BBox ] 18: $\tfrac{log(1/6)}{log(0.9)}$
\end{itemize}

%% I denne kan facit skrives
\begin{facit}
1. Sandsynligheden for ikke at vinde med $n$ terninger er $(5/6)^n$, og denne sandsynlighed skal være under 10\%.
Altså $\tfrac1{10}<(5/6)^n$, og dermed

$$n>\tfrac{log(1/10)}{log(5/6)}=\tfrac{-log(2\cdot 5)}{log(5/6)}=\tfrac{-log(2)-log(5)}{log(5)-log(6)}
\end{facit}
```

```
Data
coasts.spdf        Formal class SpatialLinesDataFrame
countries.spdf     Formal class SpatialPolygonsDataFrame
Europe             Large SpatialPolygonsDataFrame (68 elements, 931.2 Kb)
frame.spdf         Formal class SpatialPolygonsDataFrame
graticule.spdf     Formal class SpatialLines
my.center          1 obs. of 2 variables
nuts0.df           34 obs. of 10 variables
nuts0.spdf         Formal class SpatialPolygonsDataFrame
nuts1.df           111 obs. of 10 variables
nuts1.spdf         Large SpatialPolygonsDataFrame (111 elements, 536.8 Kb)
nuts2.df           310 obs. of 10 variables
nuts2.spdf         Large SpatialPolygonsDataFrame (310 elements, 1.2 Mb)
nuts3.df           1448 obs. of 7 variables
nuts3.spdf         Large SpatialPolygonsDataFrame (1448 elements, 4.8 Mb)
```

R Resources

Learning R Online
CRAN Task Views
R on StackOverflow
Getting Help with R

RStudio

RStudio IDE Support
RStudio Community Forum
RStudio Cheat Sheets
RStudio Tip of the Day
RStudio Packages
RStudio Products

Manuals

An Introduction to R
Writing R Extensions
R Data Import/Export
The R Language Definition
R Installation and Administration
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
License
NEWS
Authors
FAQ
User Manuals
Resources
Thanks
Technical papers

R as a calculator

```
 2+2

[1] 4

(2*5)+(12/3)-(2^3)

[1] 6

exp(log(1))

[1] 1

sqrt(25)

[1] 5

log(2*2)

[1] 1.3863

log(2)+log(2)

[1] 1.3863
```

Writing commands in **R**

- Commands are separated by either a new line or ;
- **R** is case sensitive: id is a different name than ID.
- The character # at the beginning of a line shows that the text in this line is a comment. I.e. the text is not executed.
- Help can be found on the internet or in **R** by writing ? followed by the function you want to help about:

```
?plot
```

- or, in RStudio, highlight the expression and press F1.

Objects in **R**

- Both data and output from analyses are stored as **objects** (if stored);
- Some times, output is just displayed on the screen, and you need to *assign* the object to an identifier to keep it (see below).
- In fact, everything in the **R** memory is stored in objects.
- An object could be a vector, a matrix or a data frame.
- Values are assigned to objects using the assignment operator <-
- The operator = also works, but it is **not recommended** as it may confuse assignments with default values and logical expressions.
- We can see the objects of the current **R** session memory in RStudio, or by using the function ls()

```
a <- 2+5
A <- 10
ls()

[1] "a" "A"
```

Generating a sequence

- Specify the first and last values separated by a colon.
- Otherwise use seq()

```
0:10

[1]  0  1  2  3  4  5  6  7  8  9 10

15:5

[1] 15 14 13 12 11 10  9  8  7  6  5

seq(from = 0, to = 1.2, by = 0.1)

[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2

x <- seq(from = 0, to = 1.5, length = 11); x

[1] 0.00 0.15 0.30 0.45 0.60 0.75 0.90 1.05 1.20 1.35 1.50
```

Generating repeats using rep()

```
rep(8, 5)

[1] 8 8 8 8 8

rep(1:4, each = 2)

[1] 1 1 2 2 3 3 4 4

rep(1:4, each = 2, times = 3)

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

Functions in R

We assign a simple function to the identifier f:

```
>f<-function(x){x^2}; f(2)
[1] 4
```

A function of two variables:

```
>f<-function(x,pow){x^pow}; f(2,2)
[1] 4
```

A function with a default value:

```
>f<-function(x,pow=2){x^pow}; f(2,2); f(2);f(2,3)
[1] 4
[1] 4
[1] 8
```

Functions in R

We have already used many functions with and without default values:

- "+"(2,2)
- sqrt(25)
- log(2)
- ls()
- ":"(0,10)
- seq(from=0.1,to=1.2,by=0.1)
- rep(1:4,each=2,time=3)

Many applications in **R** are built up as functions. You can see default arguments in the help files. Example: $\log$.

Data structures in R: Singles

- Logical, e.g:

```
> TRUE
[1] TRUE
> 1==2
[1] FALSE
```

- Single numbers, e.g:

```
> 1
[1] 1
> 1.2
[1] 1.2
```

- Character, e.g:

```
> "5"
[1] "5"
> "abc"
[1] "abc"
```

Data structures in R: Vectors

Constructed via the concatenate function $\mathrm{c}()$.

- Vector of numbers, e.g:

```
> c(1,1.2,pi,exp(1))
[1] 1.000000 1.200000 3.141593 2.718282
```

- We can have vectors of other things too, e.g:

```
> c(TRUE,1==2)
[1] TRUE FALSE
> c("a","ab","abc")
[1] "a" "ab" "abc"
```

- But not combinations, e.g:

```
> c("a",5,1==2)
[1] "a" "5" "FALSE"
```

Note that R just turned everything into characters!

Data structures in R: Matrices

- Columns of same type and same length:

```
> matrix(c(1,2,3,4,5,6)+pi,nrow=2)
[,1] [,2] [,3]
[1,] 4.141593 6.141593 8.141593
[2,] 5.141593 7.141593 9.141593

> matrix(c(1,2,3,4,5,6)+pi,nrow=2)<6
[,1] [,2] [,3]
[1,] TRUE FALSE FALSE
[2,] TRUE FALSE FALSE
```

Data structures in **R**: Data frames

- Same length of columns but different types; spread-sheet data.
- Created from reading in data from external files;
- or by using the function data.frame() on a set of vectors.

```
> data.frame(treatment=c("active","active","placebo"),
+ bp=c(80,85,90))
 treatment  bp
1   active  80
2   active  85
3  placebo  90
```

- Compare to a matrix created with the $\mathrm{cbind}()$ command:

```
> cbind(treatment=c("active","active","placebo"),bp=c(80,85,90))
     treatment bp
[1,] "active"  "80"
[2,] "active"  "85"
[3,] "placebo" "90"
```

Data structures in **R**: Lists

- Different length of columns and different types.
- Most general object type.

```
> list(a=1,b="abc",c=c(1,2,3),d=list(e=matrix(1:4,2), f=function(x){x^2}))
$a
[1] 1
$b
[1] "abc"
$c
[1] 1 2 3
$d
$d$e
     [,1] [,2]
[1,]    1    3
[2,]    2    4
$d$f
function (x)
{
    x^2
}
```

- The objects returned from many of the built-in functions in R are fairly complicated lists.

Importing Data to **R**

- The easiest is to use data saved as text files.
- Usually values in text files are separated, or delimited, by tabs or commas.
- First tell **R** where you want to find your data using the command setwd().
- Check that all went according to plan with getwd().

```
setwd("C:/Users/ANST/Teaching/02935 January 2025")
getwd()

[1] "C:/Users/ANST/Teaching/02935 January 2025"
```

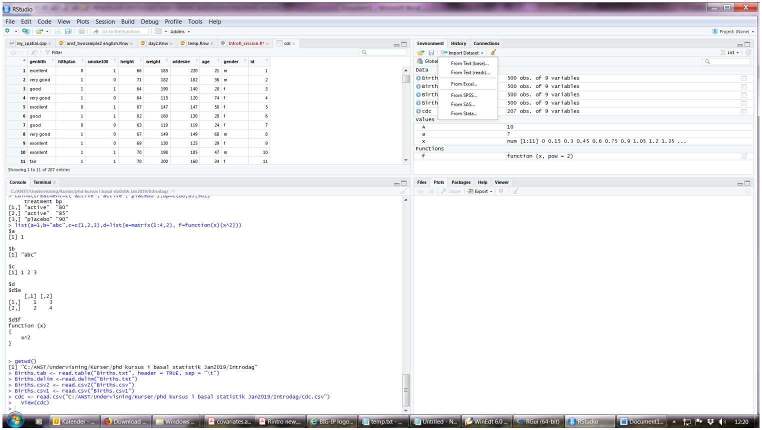
Importing Data to R

- The function read.table() can be used to read data saved as text.
- Wrappers: read.csv(), **read.csv2()** and read.delim().
- Notice the option sep = .
- We are assigning the loaded data to objects.
- If you have an Excel sheet, then save as text.

```
Births.tab <- read.table("Data/Births.txt", header = TRUE, sep = "\t")

# SHORT FORM TO READ TAB SEPARATED
Births.delim <- read.delim("Data/Births.txt")

# SHORT FORM TO READ ; SEPARATED
Births.csv2 <- read.csv2("Data/Births.csv")

# SHORT FORM TO READ , SEPARATED
Births.csv1 <- read.csv("Data/Births.csv1")
```

Importing Data using RStudio

- In the Objects Window, click "Import Dataset"



	genhlth	hlthplan	smoke100	height	weight	wtdesire	age	gender	id
1	excellent	0	1	66	185	220	21	m	1
2	very good	1	0	71	182	182	36	m	2
3	good	1	1	64	190	140	20	f	3
4	very good	1	0	64	115	120	74	f	4
5	excellent	0	1	67	147	147	50	f	5
6	good	1	1	62	160	130	20	f	6
7	good	0	0	63	119	119	24	f	7
8	very good	1	0	67	149	149	68	m	8
9	excellent	1	0	69	130	125	29	f	9
10	excellent	1	1	70	198	185	47	m	10
11	fair	1	1	70	200	160	34	f	11

Showing 1 to 11 of 207 entries

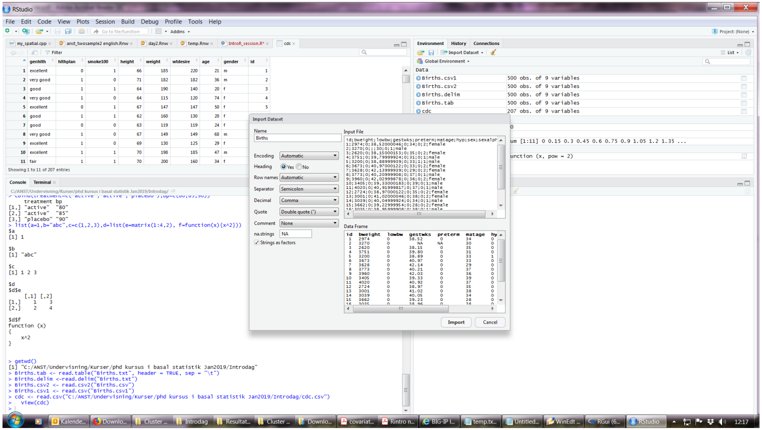
```
Data
Birth        500 obs. of 9 variables
Birth        500 obs. of 9 variables
Birth        500 obs. of 9 variables
Birth        500 obs. of 9 variables
cdc          207 obs. of 9 variables
Values
  A          10
  a          7
  x          num [1:11] 0 0.15 0.3 0.45 0.6 0.75 0.9 1.05 1.2 1.35 ...
Functions
  f          function (x, pow = 2)
```

C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag/

```
> cbind(treatment=c("active","active","placebo"),bp=c(80,85,90))
     treatment bp
[1,] "active"  "80"
[2,] "active"  "85"
[3,] "placebo" "90"
> list(a=1,b="abc",c=c(1,2,3),d=list(e=matrix(1:4,2), f=function(x){x^2}))
$a
[1] 1

$b
[1] "abc"

$c
[1] 1 2 3

$d
$d$e
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$d$f
function (x)
{
    x^2
}


> getwd()
[1] "C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag"
> Births.tab <- read.table("Births.txt", header = TRUE, sep = "\t")
> Births.delim <-read.delim("Births.txt")
> Births.csv2 <- read.csv2("Births.csv")
> Births.csv1 <- read.csv("Births.csv1")
> cdc <- read.csv("C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag/cdc.csv")
>   View(cdc)
> 
```

Importing Data using RStudio

- In the Objects Window, click "Import Dataset"



	genhlth	hlthplan	smoke100	height	weight	wtdesire	age	gender	id
1	excellent	0	1	66	185	220	21	m	1
2	very good	1	0	71	182	182	36	m	2
3	good	1	1	64	190	140	20	f	3
4	very good	1	0	64	115	120	74	f	4
5	excellent	0	1	67	147	147	50	f	5
6	good	1	1	62	160	130	20	f	
7	good	0	0	63	119	119	24	f	
8	very good	1	0	67	149	149	68	m	
9	excellent	1	0	69	130	125	29	f	
10	excellent	1	1	70	198	185	47	m	
11	fair	1	1	70	200	160	34	f	

Showing 1 to 11 of 207 entries

```
Data
Births.csv1     500 obs. of 9 variables
Births.csv2     500 obs. of 9 variables
Births.delim    500 obs. of 9 variables
Births.tab      500 obs. of 9 variables
cdc             207 obs. of 9 variables
0
um [1:11] 0 0.15 0.3 0.45 0.6 0.75 0.9 1.05 1.2 1.35 ...
unction (x, pow = 2)
```

Import Dataset

Name: Births

Encoding: Automatic
Heading: [✓ Yes] No
Row names: Automatic
Separator: Semicolon
Decimal: Comma
Quote: Double quote (")
Comment: None
na.strings: NA
✓ Strings as factors

Input File

```
id;bweight;lowbw;gestwks;preterm;matage;hyp;sex;sexalph
1;2974;0;38,52000046;0;34;0;2;female
2;3270;0;;;30;0;1;male
3;2620;0;38,15000153;0;35;0;2;female
4;3751;0;39,79999924;0;31;0;1;male
5;3200;0;38,88999939;0;33;1;1;male
6;3673;0;40,97000122;0;33;0;2;female
7;3628;0;42,13999939;0;29;0;2;female
8;3773;0;40,20999908;0;37;0;1;male
9;3960;0;42,02999878;0;36;0;2;female
10;3405;0;39,33000183;0;39;0;1;male
11;4020;0;40,91999817;0;37;0;1;male
12;2724;0;38,97000122;0;35;0;2;female
13;3001;0;41,02000046;0;38;0;2;female
14;3039;0;40,04999924;0;34;0;1;male
15;3662;0;39,22999954;0;28;0;2;female
```

Data Frame

```
id  bweight  lowbw  gestwks  preterm  matage  hy
 1  2974     0      38.52     0       34      0
 2  3270     0         NA    NA       30      0
 3  2620     0      38.15     0       35      0
 4  3751     0      39.80     0       31      0
 5  3200     0      38.89     0       33      1
 6  3673     0      40.97     0       33      0
 7  3628     0      42.14     0       29      0
 8  3773     0      40.21     0       37      0
 9  3960     0      42.03     0       36      0
10  3405     0      39.33     0       39      0
11  4020     0      40.92     0       37      0
12  2724     0      38.97     0       35      0
13  3001     0      41.02     0       38      0
14  3039     0      40.05     0       34      0
15  3662     0      39.23     0       28      0
```

C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag/

```
> cbind(treatment=c("active","active","placebo"),bp=c(80,85,90))
     treatment bp
[1,] "active"  "80"
[2,] "active"  "85"
[3,] "placebo" "90"
> list(a=1,b="abc",c=c(1,2,3),d=list(e=matrix(1:4,2), f=function(x){x^2}))
$a
[1] 1

$b
[1] "abc"

$c
[1] 1 2 3

$d
$d$e
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$d$f
function (x)
{
    x^2
}


> getwd()
[1] "C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag"
> Births.tab <- read.table("Births.txt", header = TRUE, sep = "\t")
> Births.delim <-read.delim("Births.txt")
> Births.csv2 <- read.csv2("Births.csv")
> Births.csv1 <- read.csv("Births.csv1")
> cdc <- read.csv("C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag/cdc.csv")
>   View(cdc)
> |
```

Importing Data from other stats programs

- We can read data from a series of other statistical software packages using the package foreign. A similar package is Hmisc, but we will stick to foreign.

```r
# INSTALL AN EXTRA PACKAGE
install.packages("foreign")

# ACTIVATE THE PACKAGE
library("foreign")

SPSS_Data <- read.spss("Data/SPSS_Data.sav", to.data.frame = TRUE)

Stata_Data <- read.dta("Data/string.dta")
```

- For SAS data not in XPORT format, use the sas7bdat package.

Look at Your Data

There are several ways to look at the data (or parts of the data).

```
# FIRST FEW OBSERVATIONS
head(Births.tab)

  id bweight lowbw gestwks preterm matage hyp sex sexalph
1  1    2974     0   38.52       0     34   0   2  female
2  2    3270     0      NA      NA     30   0   1    male
3  3    2620     0   38.15       0     35   0   2  female
4  4    3751     0   39.80       0     31   0   1    male
5  5    3200     0   38.89       0     33   1   1    male
6  6    3673     0   40.97       0     33   0   2  female
```

Look at Your Data

```
# LAST FEW OBSERVATIONS
tail(Births.tab)

      id bweight lowbw gestwks preterm matage hyp sex sexalph
 495 495    2968     0   41.01       0     34   0   1    male
 496 496    2852     0   38.45       0     28   0   2  female
 497 497    3187     0   38.03       0     38   1   1    male
 498 498    3054     0   38.50       0     26   0   2  female
 499 499    3178     0   39.92       0     31   0   2  female
 500 500    2918     0   37.97       0     31   0   1    male

# VARIABLE NAMES
names(Births.tab)

 [1] "id"      "bweight" "lowbw"   "gestwks" "preterm" "matage"  "hyp"
 [8] "sex"     "sexalph"

# VIEW THE DATA IN A NEW WINDOW; OFTEN THE HEAD WILL DO THOUGH
View(Births.tab)
```

Missing values

- In real life examples it is very common to have missing values.
- In **R** missing values are coded as NA (not available).
- In your Excel file leave missing values blank, do not set them to 99 or 999.

```
  id bweight lowbw gestwks preterm matage hyp sex sexalph
1  1    2974     0   38.52       0     34   0   2  female
2  2    3270     0      NA      NA     30   0   1    male
```

Accessing Observations

- Data are (usually) stored in a data frame object.
- Observations are the rows.
- Variables, either numerical or categorical, are the columns.
- We can access individual rows, columns and cells in the data frame.
- For this, we use the bracket operator: object[row, column].

Accessing Observations

```
# A SINGLE CELL
Births.tab[345, 4]

 [1] 38.55

# LEAVING OUT A COLUMN NUMBER INDICATES THAT ALL COLUMNS
# ARE CHOSEN. HERE ALL COLUMNS IN ROW 224
Births.tab[224 , ]

     id bweight lowbw gestwks preterm matage hyp sex sexalph
 224 224    3216     0   39.94       0     38   1   1    male
```

Accessing Observations

```
# LEAVING OUT A ROW NUMBER INDICATES THAT ALL ROWS ARE CHOSEN
# HERE ALL ROWS IN COLUMN 5
Births.tab[ ,5]


   [1]  0 NA  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
  [24]  0  0  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
  [47]  0  1  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0
  [70]  0  0  0  1  1  0  0  0  0  0 NA  0  0  0  0  0  0  0  0  0  0 NA  0
  [93]  1  0  1  0  0  0  0  0  0  0  0  0  0  0 NA  0  0  0  0  0  0  0  1
 [116]  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
 [139]  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 [162]  0  0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0
 [185]  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0
 [208]  0  1  1  0  0  0  1  0  0  1  0  0  1  1  0  0  0  0  1  0  0  1  0
 [231]  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  1
 [254]  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1
 [277]  0  0  0  0  1 NA  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 [300]  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  1  0
 [323]  0  0  0  0  0  1  0  0  0  0  1  0  0  0  1  1  0  0  0  0  0  0  0
 [346]  1  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0 NA  0  1  0  1  0  0
 [369]  0  1  0  0  0  0  0  0  1  0  0  0 NA  0  0  0  0  0  0  0  0  0  0
 [392]  0  0  0  1 NA  0  0 NA NA  0  0  0  0  0  0  0  0  0  0  1  0  0  1
 [415]  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0  0  0
 [438]  0  1  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
 [461]  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0  0
 [484]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Accessing Observations

```
# USE RANGES, ROWS 15 TO 18 COLUMNS 1 TO 4
Births.tab[15:18, 1:4]

   id bweight lowbw gestwks
15 15    3662     0   39.23
16 16    3035     0   38.96
17 17    3351     0   39.35
18 18    3804     0   38.99
```

Accessing Observations

Variables can be accessed directly using their name, either with the $ operator (object$variable), the name (object[,"variable"]), or the column number (object[,k]).

```
# GET THE BIRTH WEIGHT FOR CHILD 26 TO 36
Births.tab$bweight[26:36]

  [1] 3585 3798 3164 3739 1780 4022 3942 2887 2391 3911 3509

Births.tab[26:36, "bweight"]

  [1] 3585 3798 3164 3739 1780 4022 3942 2887 2391 3911 3509

Births.tab[26:36,2]

  [1] 3585 3798 3164 3739 1780 4022 3942 2887 2391 3911 3509
```

Subsetting using the c() function

- The concatenate function c() concatenates the arguments into a vector. It can be used for many things; one is to access non-sequential rows and columns from a data frame.

```
# GET COLUMNS 2, 5, 7, 8, 9 FOR ROW 33
Births.tab[33, c(2, 5, 7:9)]

   bweight preterm hyp sex sexalph
 33    2887       0   0   1    male

# GET bweight, preterm and sexalph FOR ROW 71
Births.tab[71, c("bweight", "preterm", "sexalph")]

   bweight preterm sexalph
 71    3189       0    male
```

Variable Names

If we want to change the variable names we can use names().

```
# NEW VARIABLE NAMES
names(Births.tab) <- c("ID", "Bweight", "LowBW", "GestWks",
                      "Preterm", "Matage", "Hyp", "Sex", "Sexalph")

# JUST THE FIRST VARIABLE NAME
names(Births.tab)[1] <- c("ID_new")

# CHECK HOW IT WENT
names(Births.tab)

 [1] "ID_new"  "Bweight" "LowBW"   "GestWks" "Preterm" "Matage"  "Hyp"
 [8] "Sex"     "Sexalph"

# RESETTING
names(Births.tab)[1] <- c("ID")
```

Saving/Exporting data

- We can save the data to a textfile using either write.table() for a tab separated file, or write.csv()/write.csv2() for a comma/semicolon separated file (with "."and ","as decimal point, respectively).

```r
write.table(Births.tab, file = "Birth_new.txt",
              sep = "\t", na = ".", row.names= FALSE)

write.csv2(Births.tab, file = "Birth_new.csv")
```

Exercise: Protein Consumption

- Open RStudio and set the working directory to where you want to keep the data for the course today.
- Import the data Protein.xlsx to **R**.
- Look at the data.
- What is the protein consumption in Denmark?
- Look at the protein consumption from red meat alone.
- Look at the protein consumption in Denmark, Norway, Sweden from red meat, white meat and eggs.
- Rename the variables “RedMeat“ and “WhiteMeat“ to “Red“ and “White“.
- Save the new version of the protein data as a tab delimited text file “Protein2.txt“.

Description of Data

We are still looking at the data set with birth weights for 500 children. Using the function str() we can see a description of what our data frame contains (the structure).

```
str(Births.tab)

 'data.frame': 500 obs. of  9 variables:
  $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
  $ bweight: int  2974 3270 2620 3751 3200 3673 3628 3773 3960 3405 ...
  $ lowbw   : int  0 0 0 0 0 0 0 0 0 0 ...
  $ gestwks: num  38.5 NA 38.2 39.8 38.9 ...
  $ preterm: int  0 NA 0 0 0 0 0 0 0 0 ...
  $ matage : int  34 30 35 31 33 33 29 37 36 39 ...
  $ hyp     : int  0 0 0 0 1 0 0 0 0 0 ...
  $ sex     : int  2 1 2 1 1 2 2 1 2 1 ...
  $ sexalph: chr "female" "male" "female" "male" ...
```

Description of Data: Birth weights

- str() supplies all information about an object. **Much is redundant**, do not use str() for report writing but for your own overview only.
- We see that we have a data frame with 500 observations and 9 variables.
- Some are integers but “gestwks“ is numeric.
- The variable “sexalph“ is a character variable.
- Note that “sexalph“ and “sex“ describes the same thing. But R does not interpret the character values as group labels.
- Factor: Grouping with informative labels. We can convert “sexalph“ to a factor using as.factor().

Description of Data: Birth weights

```
# TELL R THAT sexalph IS A FACTOR
Births.tab$sexalph <- as.factor(Births.tab$sexalph)
levels(Births.tab$sexalph)

 [1] "female" "male"

str(Births.tab)

 'data.frame': 500 obs. of  9 variables:
  $ id     : int  1 2 3 4 5 6 7 8 9 10 ...
  $ bweight: int  2974 3270 2620 3751 3200 3673 3628 3773 3960 3405 ...
  $ lowbw  : int  0 0 0 0 0 0 0 0 0 0 ...
  $ gestwks: num  38.5 NA 38.2 39.8 38.9 ...
  $ preterm: int  0 NA 0 0 0 0 0 0 0 0 ...
  $ matage : int  34 30 35 31 33 33 29 37 36 39 ...
  $ hyp    : int  0 0 0 0 1 0 0 0 0 0 ...
  $ sex    : int 2 1 2 1 1 2 2 1 2 1 ...
  $ sexalph: Factor w/ 2 levels "female","male": 1 2 1 2 2 1 1 2 1 2 ...
```

Description of Data: Birth weights

```
# TELL R THAT sex IS A FACTOR WITH SPECIFIC LEVELS
Births.tab$sex <- factor(Births.tab$sex,labels =c("Male","Female"))
levels(Births.tab$sex)

 [1] "Male" "Female"

str(Births.tab)

 'data.frame': 500 obs. of  9 variables:
  $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
  $ bweight: int  2974 3270 2620 3751 3200 3673 3628 3773 3960 3405 ...
  $ lowbw   : int  0 0 0 0 0 0 0 0 0 0 ...
  $ gestwks: num  38.5 NA 38.2 39.8 38.9 ...
  $ preterm: int  0 NA 0 0 0 0 0 0 0 0 ...
  $ matage : int  34 30 35 31 33 33 29 37 36 39 ...
  $ hyp     : int  0 0 0 0 1 0 0 0 0 0 ...
  $ sex     : Factor w/ 2 levels "Male","Female": 2 1 2 1 1 2 2 1 2 1 ...
  $ sexalph: Factor w/ 2 levels "female","male": 1 2 1 2 2 1 1 2 1 2 ...
```

Descriptive Statistics

- There are some simple functions for summary statistics in **R**.
- Very common extractor functions are mean(), sd(), median(), max() and min().

```
mean(Births.tab$bweight)

 [1] 3136.9

sd(Births.tab$bweight)

 [1] 637.45

median(Births.tab$bweight)

 [1] 3188.5

max(Births.tab$bweight)

 [1] 4553

min(Births.tab[ , 2])

 [1] 628
```

The Summary Function

- The function summary() can be used with many objects in **R**.
- When used on a data frame we get all the main summary statistics.

```
# SUMMARY OF THE DATA FRAME
summary(Births.tab)

       id          bweight          lowbw           gestwks
 Min.   :  1    Min.   : 628    Min.   :0.00    Min.   :24.7
 1st Qu.:126    1st Qu.:2862    1st Qu.:0.00    1st Qu.:37.9
 Median :250    Median :3188    Median :0.00    Median :39.1
 Mean   :250    Mean   :3137    Mean   :0.12    Mean   :38.7
 3rd Qu.:375    3rd Qu.:3551    3rd Qu.:0.00    3rd Qu.:40.1
 Max.   :500    Max.   :4553    Max.   :1.00    Max.   :43.2
                                                NA's   :10
    preterm           matage          hyp          sex       sexalph
 Min.   :0.000   Min.   :23   Min.   :0.000   1:264   female:236
 1st Qu.:0.000   1st Qu.:31   1st Qu.:0.000   2:236   male  :264
 Median :0.000   Median :34   Median :0.000
 Mean   :0.129   Mean   :34   Mean   :0.144
 3rd Qu.:0.000   3rd Qu.:37   3rd Qu.:0.000
 Max.   :1.000   Max.   :43   Max.   :1.000
 NA's   :10
```

Summaries

- We may only want summaries for some of the data, e.g. babies with birth weight < 2900g.
- We subset the data and then summarize as before:

```
summary(Births.tab[Births.tab$bweight<2900,])


       id           bweight          lowbw           gestwks
  Min.   :  3   Min.   : 628   Min.   :0.000   Min.   :24.7
  1st Qu.:146   1st Qu.:2120   1st Qu.:0.000   1st Qu.:35.5
  Median :254   Median :2580   Median :0.000   Median :37.4
  Mean   :250   Mean   :2355   Mean   :0.441   Mean   :36.6
  3rd Qu.:359   3rd Qu.:2741   3rd Qu.:1.000   3rd Qu.:38.5
  Max.   :496   Max.   :2894   Max.   :1.000   Max.   :41.4
                                               NA's   :2
     preterm           matage          hyp          sex      sexalph
  Min.   :0.000   Min.   :24   Min.   :0.000   1:63   female:73
  1st Qu.:0.000   1st Qu.:31   1st Qu.:0.000   2:73   male  :63
  Median :0.000   Median :34   Median :0.000
  Mean   :0.403   Mean   :34   Mean   :0.243
  3rd Qu.:1.000   3rd Qu.:37   3rd Qu.:0.000
  Max.   :1.000   Max.   :43   Max.   :1.000
  NA's   :2
```

Group Summaries

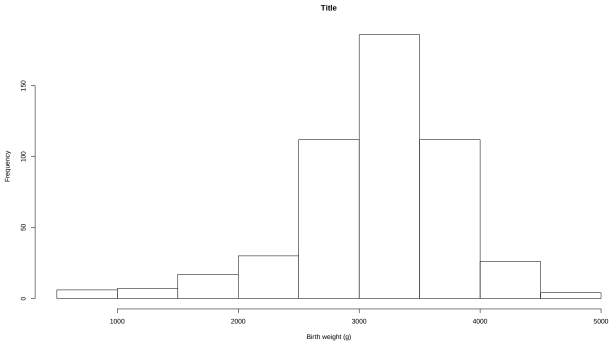
- Data may be separated by groups.
- Suppose that we want to calculate the mean birth weight for boys and girls (many ways to do this).
- We will use the tapply() function to apply the mean function to the two levels of “sexalph“.
- tapply(<variable to summarize>, <variable to group by>, <function to use>).

```
# MEAN BIRTH WEIGHT FOR BOYS AND GIRLS
tapply(Births.tab$bweight, Births.tab$sexalph, mean)

  female     male
3032.831 3229.902
```

Histogram

Often it is easier to get an impression of a distribution using plots. Histograms are typically used for continuous variables.

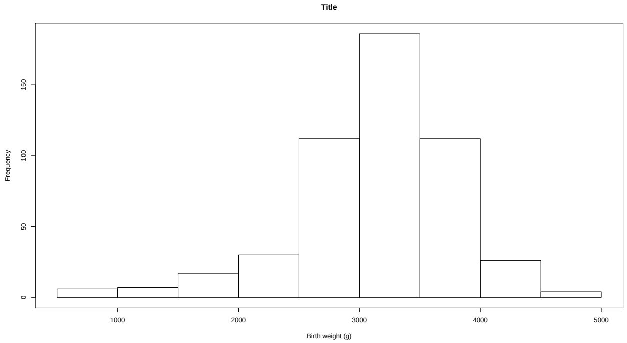
```
hist(Births.tab$bweight, main = "Title", xlab = "Birth weight (g)"
```



Histogram

Often it is easier to get an impression of a distribution using plots. Histograms are typically used for continuous variables. Here with a box on.

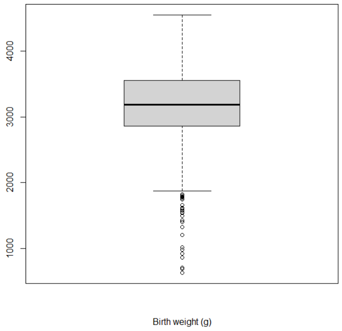
```
hist(Births.tab$bweight, main = "Title",
     xlab = "Birth weight (g)")
box()
```



Boxplot

Boxplots show the median, upper, lower quartiles and potentially extreme values.

```
boxplot(Births.tab$bweight, xlab = "Birth weight (g)")
```



Exercise: Descriptive Statistics 1

- Import the dataset cdc.csv into **R**.
- Look at the data.
- Describe the structure of the data.
- Change the variable “smoke100“ to a factor.
- Summarize the variables in the dataset.
- Calculate the mean age by general health.
- Draw a histogram of age.
- Draw a boxplot of age.

Modifying Data

We will concentrate on how to modify and rearrange our data.

- Data can be sorted with the order function.
- order can sort the Birth.tab data by “sex“, and then by “bweight“.
- The order function returns a vector of sorted indices, which we apply to the rows of the unsorted data frame to get a sorted version.

```
Birth_sort <- Births.tab[order(Births.tab$sex, Births.tab$bweight), ]

head(Birth_sort)

      id bweight lowbw gestwks preterm matage hyp sex sexalph
 253 253     693     1   30.71       1     34   1   1    male
 226 226     981     1   27.99       1     29   1   1    male
 181 181    1019     1   28.04       1     31   1   1    male
 22   22    1203     1   32.80       1     39   0   1    male
 312 312    1500     1   35.27       1     34   0   1    male
 313 313    1595     1   30.52       1     33   1   1    male
```

Creating new variables and deleting old

New variables can be added to a data frame.

```
# ADD A VARIABLE TO DATA FRAME
Births.tab$log_bweight <- log(Births.tab$bweight)
```

Columns can be dropped from a data frame (log birth weight is column 10):

```
Births.tab <- Births.tab[ , -10]
```

```
# CREATE A VARIABLE AS A SEPARATE OBJECT
log_bweight <- log(Births.tab$bweight)
```

Objects can be removed from the R memory (cleaning up):

```
rm(log_bweight)
```

Grouping the values of a variable using cut

You might want to group a continuous variable e.g. mother's age (matage) into the groups:]20-30],]30-35],]35-40],]40-45]:

```
 Births.tab$agegrp <- cut(Births.tab$matage,
                          breaks = c(20, 30, 35, 40, 45))
summary(Births.tab[ , c("matage", "agegrp")])

     matage        agegrp
  Min.   :23   (20,30]: 99
  1st Qu.:31   (30,35]:215
  Median :34   (35,40]:174
  Mean   :34   (40,45]: 12
  3rd Qu.:37
  Max.   :43
```

Creating new variables: RowSums

- Often we want to form new variables from other variables.
- For example, we might want to calculate a total score from sub scores.
- We can sum variables using rowSums. Related functions are: rowMeans, colSums, colMeans.
- Notice the option na.rm.
- If we take a row sum where one of the values is missing then the row sum is set to missing na.rm= FALSE.
- If we want to ignore missing values and calculate a sum of the non missing then na.rm= TRUE.
- rowSums, rowMeans, colSums and colMeans are wrappers of sapply, ie. t.ex. colMeans(x) is the same as sapply(x,mean). sapply can be used with many other functions.

Creating new variables: RowSums

```
# NEW VARIABLE SCORE SUMMING PRETERM, LOWBW AND HYP
Births.tab$score <- rowSums(Births.tab[ ,c(3,5,7)], na.rm = FALSE)

#REMOVE MISSING
Births.tab$scoreRM <- rowSums(Births.tab[ ,c(3,5,7)], na.rm = TRUE)

head(Births.tab)

  id bweight lowbw gestwks preterm matage hyp sex sexalph score scoreRM
1  1    2974     0   38.52       0     34   0   2  female     0       0
2  2    3270     0      NA      NA     30   0   1    male    NA       0
3  3    2620     0   38.15       0     35   0   2  female     0       0
4  4    3751     0   39.80       0     31   0   1    male     0       0
5  5    3200     0   38.89       0     33   1   1    male     1       1
6  6    3673     0   40.97       0     33   0   2  female     0       0
```

Exercise

- Import the data cdc.
- Height is in inches, weight and wtdesire are in pounds. Generate new variables in cm and kg (use Google to find conversion factors).
- Make a factor with 4 *approximately* equally sized groups from the weight variable.
- Sort the data by gender and age.
- Calculate the average of the weight and desired weight for each subject.
- Calculate the mean weight and wtdesire (in kg) for each level of "genhlth"with the by function.

Split Data: Subset

- Sometimes we may need to split our data.
- In the Births data we may need to split the data into boys and girls.
- We can use the subset() function and assign the new data sets to separate **R** objects.
- Notice == (logical expression). We are not assigning a value to "sex", but asking whether "sex is equal to 1".

```
Births.Male <- subset(Births.tab, sex == 1)
Births.Female <- subset(Births.tab, sex == 2)
```

- Alternatively the bracket operator:

```
Births.Male <- Births.tab[Births.tab$sex == 1,]
Births.Female <- Births.tab[Births.tab$sex == 2,]
```

Subset

- Often data sets come with a lot of variables and we only want to use a few.
- Similarly to the bracket operator [the function subset() can be used to select the variables we want.
- Notice the select option. This is needed to say that we want a subset of columns (on the previous slide it was rows).
- Notice that we do not need quotes in select.

```
# SELECT 3 VARIABLES
Births.new <- subset(Births.tab, select = c(id, bweight, sex))
```

Aggregating data

- Sometimes we want to make a new dataframe as a summary of the original dataframe on the basis of factor levels.
- Below we want to make a new dataframe with the mean birthweight for combinations of preterm and sex.

```
PreSex <- aggregate(Births.tab$bweight,
                    by = list(Preterm = Births.tab$preterm,
                              Sex = Births.tab$sexalph), mean)
PreSex

  Preterm    Sex      x
1       0 female 3191.6
2       1 female 2020.8
3       0   male 3361.2
4       1   male 2321.6
```

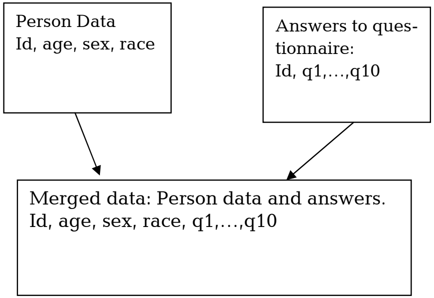
Add rows: rbind()

- Data are collected for subgroups of subjects and saved in separate objects.
- The separate objects are appended (stacked) to create a single object.
- This will give an error message if the number of columns differs.

```
# APPEND
Births.Both <- rbind(Births.Male, Births.Female)
dim(Births.Both)

 [1] 500  11

dim(Births.Male)

 [1] 264  11

dim(Births.Female)

 [1] 236  11
```

Add variables: merge()

Often you have data in several data sets and want to combine the data sets by merging using one or more variables as *key variables*. Adding variables to a master data set.


Person Data
Id, age, sex, race
Answers to questionnaire:
Id, q1,...,q10
Merged data: Person data and answers.
Id, age, sex, race, q1,...,q10

Merge

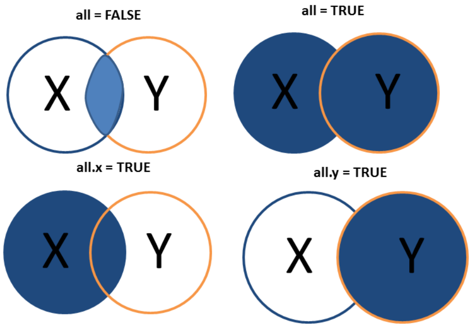
We have two data sets with a key variable "id". One with background information and one set with blood pressure measurements.

```
agesex <- read.delim("agesex.txt")
agesex

   id age sex
1  99  43   m
2 100  47   f
3 101  NA   f
4 102  67   m
```

```
bp <- read.delim("bp.txt")
bp

   id visit  bp
1 100     1 180
2 100     2 160
3 100     3 155
4 101     1 160
5 102     1 120
6 102     2 140
7 103     1 135
```

4 Different Merges

- In the merge function we will look at 4 of the options.
- We have merge(x, y, by = "key variable", all = TRUE, $<$ all = FALSE, all.x = TRUE, all.y =FALSE $>$).
- Here x and y are data frames



Merging all=FALSE

```
merge_small <- merge(agesex, bp, by = "id", all = FALSE)
merge_small

   id age sex visit  bp
1 100  47   f     1 180
2 100  47   f     2 160
3 100  47   f     3 155
4 101  NA   f     1 160
5 102  67   m     1 120
6 102  67   m     2 140
```

Merging all=TRUE

```
merge_large <- merge(agesex, bp, by = "id", all = TRUE)
merge_large

   id age  sex visit  bp
1  99  43    m    NA  NA
2 100  47    f     1 180
3 100  47    f     2 160
4 100  47    f     3 155
5 101  NA    f     1 160
6 102  67    m     1 120
7 102  67    m     2 140
8 103  NA <NA>     1 135
```

Merging all.x=TRUE

```
merge_x <- merge(agesex, bp, by = "id", all.x = TRUE)
merge_x

   id age sex visit  bp
1  99  43   m    NA  NA
2 100  47   f     1 180
3 100  47   f     2 160
4 100  47   f     3 155
5 101  NA   f     1 160
6 102  67   m     1 120
7 102  67   m     2 140
```

Merging all.y=TRUE

```
merge_y <- merge(agesex, bp, by = "id", all.y = TRUE)
merge_y

   id age  sex visit  bp
1 100  47    f     1 180
2 100  47    f     2 160
3 100  47    f     3 155
4 101  NA    f     1 160
5 102  67    m     1 120
6 102  67    m     2 140
7 103  NA <NA>     1 135
```

Counting the Missing Observations: The is.na() and sum() functions

- Suppose that we want to count the number of missing observations.
- The function is.na returns a logical vector that is TRUE when a value is missing and FALSE otherwise.

```
is.na(merge_y$sex)

 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE

#COUNT MISSING FOR ONE VARIABLE
sum(is.na(merge_y$sex))

 [1] 1

#COUNT FOR DATA FRAME
colSums(is.na(merge_y))

    id   age   sex visit    bp
     0     2     1     0     0
```

Loops in R

- In **R**, the **for loop** is used perform a repetitive task for each element in a set.
- Example:
  - Given a set of integers 1:3:
  - Let a variable i run through the set and print $i + i$:

```
for(i in 1:3) {
  cat(i, "+", i, "=", i+i, "\n")
  }
```

- Output:

```
1 + 1 = 2
2 + 2 = 4
3 + 3 = 6
```

Flow Control: if and if else Statements

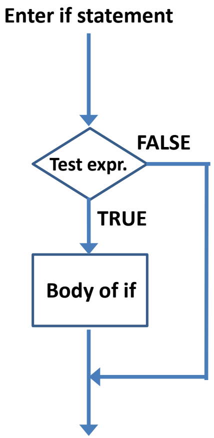
- if statement:

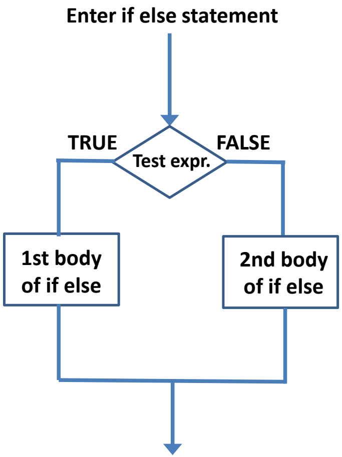
```
for(i in 1:3){
  if (i==2) cat("This index is even:","\n")
  cat(i,"\n")
  }
1
This index is even:
2
3
```

- if else statement:

```
for(i in 1:3){
  if (i==2) cat("The index is 2","\n") else
            cat("The index is not 2","\n")
  }
The index is not 2
The index is 2
The index is not 2
```

Flow Charts for if and if else Statements





while and repeat Loops

The **while** loop:

- while(condition) expression

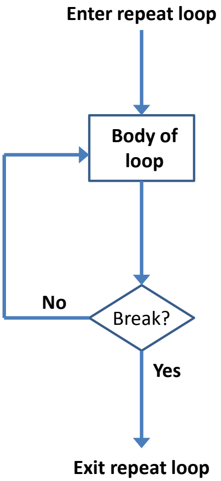
The **repeat** loop:

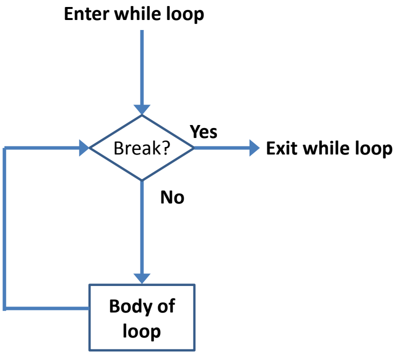
- repeat expr

The repeat loop has to be exited manually. Flow controls:

- **next:** Halts the the current iteration and advances to the next immediately;
- **break:** Exits the loop.

Flow Charts for the while and repeat loops





while Loop Example

Storing of machine parts

```
k<-0 # number of big parts (>2)
y<-abs(rnorm(1000)) # simulated part size
i<-0 # index of parts
# loop:
while(k<3 & i<1000){
 i<-i+1
 temp<-y[i]
 k<-k+(temp>2)
  }
i
```

```
[1] 42
```

repeat Loop Example

Selecting persons without blue or yellow eyes

```
eye.colors<-c("brown","blue","green","yellow","grey")
eyecolor<-data.frame(personId=1:100,color=
                     sample(eye.colors,100,rep=T))
i<-0
list.of.ids<-numeric(0) # patient ID list
#loop:
repeat {
  i<-i+1
  if(eyecolor$color[i]=="yellow" |
     eyecolor$color[i]=="blue") next
  list.of.ids<-c(list.of.ids,eyecolor$personId[i])
  if(i==100 | length(list.of.ids)==20) break
  }
list.of.ids
```

```
[1]  5  6  7  9 10 11 12 14 15 18 19 20 21 22 23 24 25 28 29 30
```

Saving your work

- Saving your script
- Saving your workspace

Always save your script - do it often if you work in Rstudio.

- Reasons for saving your workspace:
  - Extensive data creations will be there next time you open your workspace.
  - Objects created 'on the fly' (not in your script) will be there.
- Reasons for not saving your workspace:
  - With a well-written script, you can recreate your analysis in seconds, unless you work with huge amounts of data.
  - Edited and saved data where editions have been forgotten may cause havoc on your results.
  - Left-over objects created for various purposes may enter your calculations unintentionally due to the structure of **R**'s search path.

Saving your work

How to save your work:

- Script: Click on the script and press 'save' in Rstudio and the plain R GUI.
- Workspace: Click on the command prompt and press 'save'. Alternatively, use the save.image() function
- Both: Accept when asked after terminating Rstudio or the plain R GUI.

Exercise

- Data on exercise habits for the subjects in the cdc data set are found in the data cdc_exer.csv. Load and describe these data.
- Merge the cdc data set with the cdc_exer data set using id as the key variable. How many subjects in the study did not have exercise information?
- List the identifiers for the subjects without exercise information.
- Make a new dataframe with average age for combinations of gender and general health.

Visualizing Data is Important

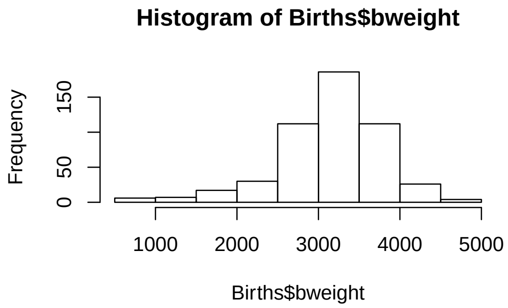
- Whenever we want to analyze data the first thing we should do is to have a look at it.
- How are the observations spread out?
- What are the most common values?
- Are there any unusual observations?
- Are there any relationships between variables?

This session will not tell you all about graphics in **R** but get you going.

A Basic Histogram

- Common way to examine the distribution of a continuous variable.
- The range of the variable is by default divided into equal-width intervals (bins). Plots the number of observations in each bin (unless you specify otherwise).

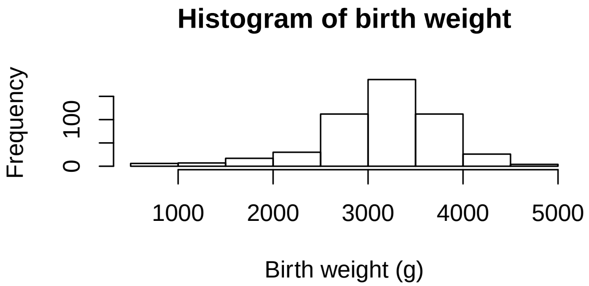
```r
hist(Births$bweight)
```



Histogram with a few options

- Note that **R** automatically has created axis labels and a heading.
- To modify axis labels we set the options xlab and ylab.
- The heading is set in the option main.

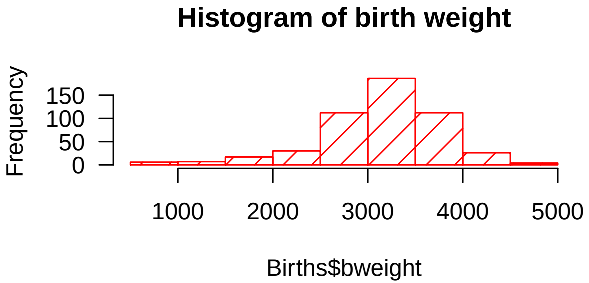
```
hist(Births$bweight, xlab = "Birth weight (g)",
    main = "Histogram of birth weight")
```



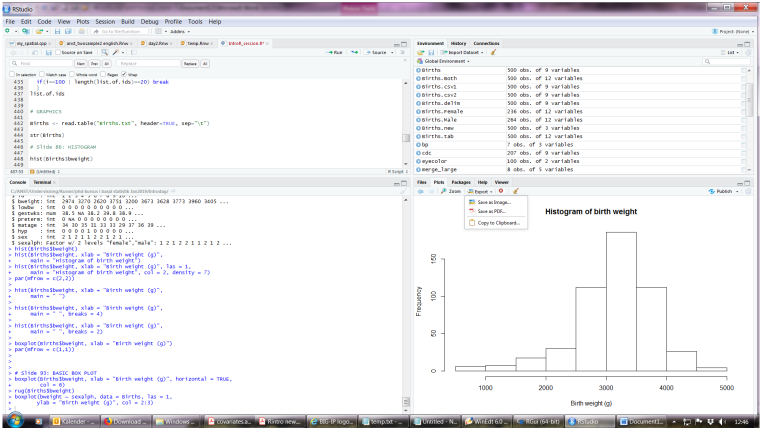
Histogram with more options

- We could type ?hist to find more options to customize the histogram.
- The available colours are coded as numbers or one can write col = “red“
- If we want shading we can try the density function.
- The angle of the numbers on the axes is set by the option las.

```r
hist(Births$bweight,las = 1, main = "Histogram of birth weight",
     col = 2, density = 7)
```



How to get your plot from RStudio



```
  if(i==100 | length(list.of.ids)==20) break
  }
list.of.ids


# GRAPHICS

Births <- read.table("Births.txt", header=TRUE, sep="\t")

str(Births)

# Slide 86: HISTOGRAM

hist(Births$bweight)

```

Births	500 obs. of 9 variables
Births.Both	500 obs. of 12 variables
Births.csv1	500 obs. of 9 variables
Births.csv2	500 obs. of 9 variables
Births.delim	500 obs. of 9 variables
Births.Female	236 obs. of 12 variables
Births.Male	264 obs. of 12 variables
Births.new	500 obs. of 3 variables
Births.tab	500 obs. of 12 variables
bp	7 obs. of 3 variables
cdc	207 obs. of 9 variables
eyecolor	100 obs. of 2 variables
merge_large	8 obs. of 5 variables

C:/ANST/Undervisning/Kurser/phd kursus i basal statistik Jan2019/Introdag/

```
 $ bweight: int  2974 3270 2620 3751 3200 3673 3628 3773 3960 3405 ...
 $ lowbw  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ gestwks: num  38.5 NA 38.2 39.8 38.9 ...
 $ preterm: int  0 NA 0 0 0 0 0 0 0 0 ...
 $ matage : int  34 30 35 31 33 33 29 37 36 39 ...
 $ hyp    : int  0 0 0 0 1 0 0 0 0 0 ...
 $ sex    : int  2 1 2 1 1 2 2 1 2 1 ...
 $ sexalph: Factor w/ 2 levels "female","male": 1 2 1 2 2 1 1 2 1 2 ...
> hist(Births$bweight)
> hist(Births$bweight, xlab = "Birth weight (g)",
+      main = "Histogram of birth weight")
> hist(Births$bweight, xlab = "Birth weight (g)", las = 1,
+      main = "Histogram of birth weight", col = 2, density = 7)
> par(mfrow = c(2,2))
>
> hist(Births$bweight, xlab = "Birth weight (g)",
+      main = " ")
>
> hist(Births$bweight, xlab = "Birth weight (g)",
+      main = " ", breaks = 4)
>
> hist(Births$bweight, xlab = "Birth weight (g)",
+      main = " ", breaks = 2)
>
> boxplot(Births$bweight, xlab = "Birth weight (g)")
> par(mfrow = c(1,1))
>
>
>
> # Slide 93: BASIC BOX PLOT
> boxplot(Births$bweight, xlab = "Birth weight (g)", horizontal = TRUE,
+         col = 6)
> rug(Births$bweight)
> boxplot(bweight ~ sexalph, data = Births, las = 1,
+        ylab = "Birth weight (g)", col = 2:3)
>
```

Writing to a graphics device

```
pdf("my.histogram.pdf")
hist(Births$bweight,las = 1, main = "Histogram of birth weight",
     col = 2, density = 7)
dev.off()
```

```
svg("my.histogram.svg")
hist(Births$bweight,las = 1, main = "Histogram of birth weight",
     col = 2, density = 7)
dev.off()
```

```
png("my.histogram.png")
hist(Births$bweight,las = 1, main = "Histogram of birth weight",
     col = 2, density = 7)
dev.off()
```

- Options can be specified; see the help files.

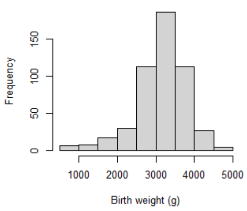
Exercise

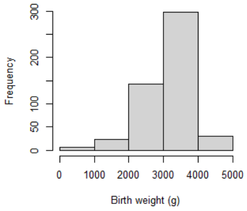
- Load the cdc data set.
- Make a histogram of the weight in kg.
- Add your own title and x-axis label.
- Try different colours and shadings.
- Copy your favorite histogram into a document in e.g. Word.

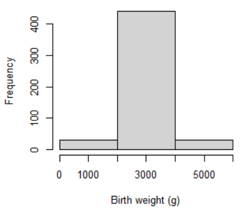
A Basic Box Plot

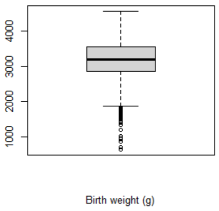
- Box plots show some distributional properties very clearly.
- Box plots show a measure of the location (the median line).
- The spread of the distribution (the length of the box and whiskers).
- Skewness as asymmetry in the upper and lower parts of the box and whisker length.
- We use the function boxplot(variable). Adding labels to the axes and colours is done as for hist.

Histograms and a Box Plot





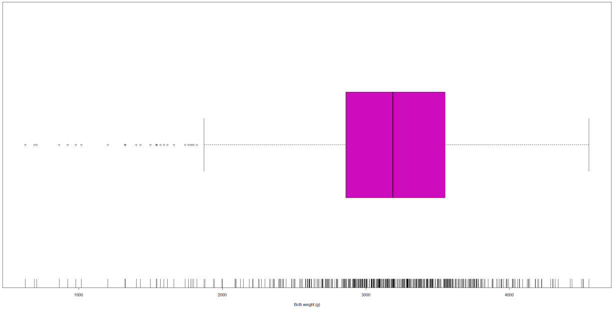




A Basic Box Plot

- When describing data we can even add the observations to the plot.
- Notice the function rug shows the observations.

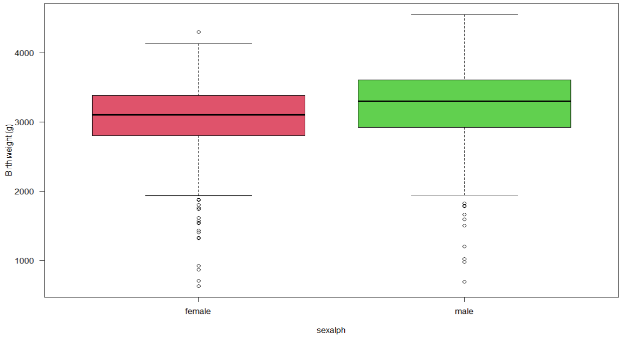
```
boxplot(Births$bweight, xlab = "Birth weight (g)", horizontal = TRUE,
        col = 6)
rug(Births$bweight)
```



Box Plot for Groups

A useful feature is that we can make box plots for different groups next to each other for comparison. Notice the option data = Births.

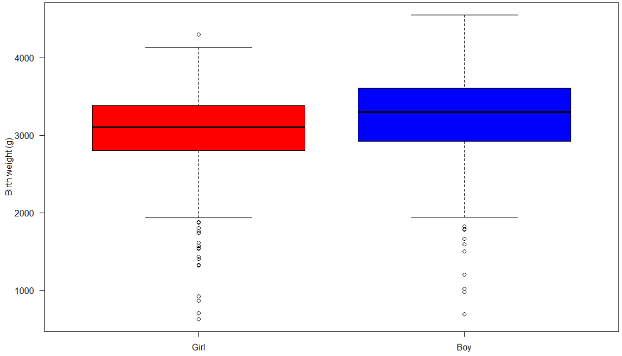
```
# BOX PLOT FOR BOYS AND GIRLS
boxplot(bweight ~ sexalph, data = Births, las = 1,
        ylab = "Birth weight (g)", col = 2:3)
```



Box Plot for Groups

Set our own axis. Notice xaxt = “n“.

```
# BOX PLOT WHERE WE WANT TO MAKE OUR OWN AXIS
boxplot(bweight ~ sexalph, data = Births, las = 1,
        ylab = "Birth weight (g)", col = c("red", "blue"), xaxt = "n")
axis(1 ,at = c(1,2), labels = c('Girl', 'Boy'))
```



Exercise

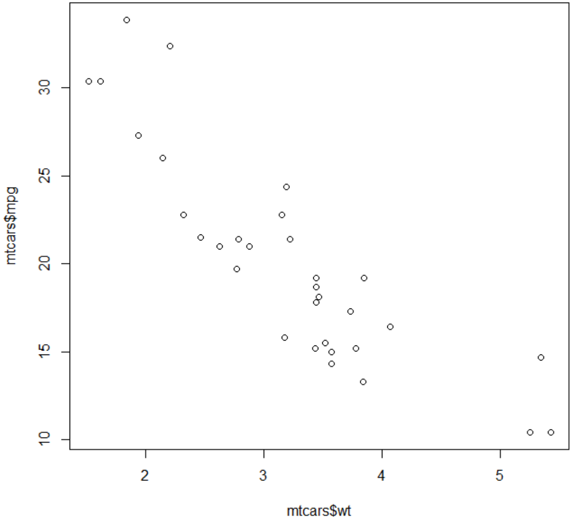
- Load the cdc data set.
- Make a box plot of the weight in kg showing individual observations. Add your own titles and colours.
- Make a box plot for each level of "genhlth", where each box is a different colour. How were the plots sorted?
- Make a new variable genhlth.num with 5 levels where 1 is "poor", 2 is "fair", 3 is "good", 4 is "very good"and 5 is "excellent".
- Make a new box plot for each level of genhlth.num with labels on the x-axis.
- Does this box plot show any pattern?

The Basic Scatter Plot

- The scatter plot is the standard graph for examining the relationship between two continuous variables.
- The plot(x,y) function is used to create scatter plots. Where (x,y) are the points we want to plot.
- We will look at the relationship between car weight (lbs/1000) and miles per gallon for 32 cars.

```
plot(mtcars$wt, mtcars$mpg)
```

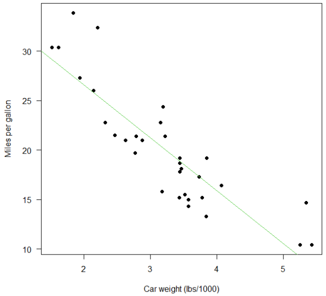
The Basic Scatter Plot



The Scatter Plot

- We can customize the scatter plot similar to before.
- The function abline adds a straight line to the plot.
- When we write abline(lm(mpg $\sim$ wt)) we get the best fitting line.

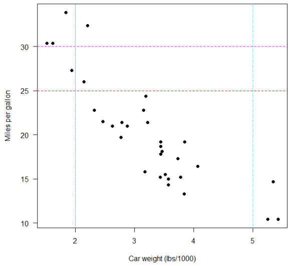
```
plot(mtcars$wt, mtcars$mpg, xlab = "Car weight (lbs/1000)",
      ylab = "Miles per gallon", las = 1, pch = 19)
abline(lm(mtcars$mpg ~ mtcars$wt), lty = 1, col = 3)
```



abline

- The function abline can also add reference lines to a plot.
- A horizontal line, e.g. at 25 and 30 abline(h = c(25, 30))
- A vertical line, e.g. at 2 and 5 abline(v = c(2, 5))

```
plot(mtcars$wt, mtcars$mpg, xlab = "Car weight (lbs/1000)",
       ylab = "Miles per gallon", las = 1, pch = 19)
abline(h = c(25, 30), col = c("red", "magenta"), lty = 2)
abline(v = c(2, 5), col = 4:5, lty = 3:4)
```

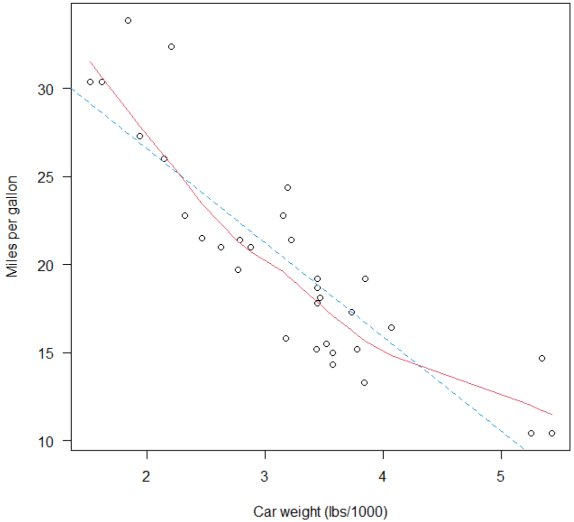


Add a smoothed line

Perhaps we do not think the association is linear and try a nonparametric smoothed line.

```
plot(mtcars$wt, mtcars$mpg, xlab = "Car weight (lbs/1000)",
     ylab = "Miles per gallon", las = 1)
abline(lm(mtcars$mpg ~ mtcars$wt), lty = 2, col = 4)
lines(lowess(mtcars$wt, mtcars$mpg), lty = 1, col = 2)
```

Add a smoothed line

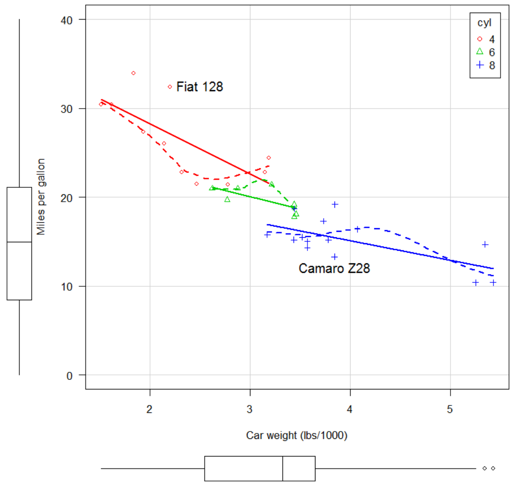


Enhanced graph procedures: Scatter plot example from the "car"package

```
scatterplot(mpg ~ wt | cyl, data = mtcars, ylim = c(0,40),
            xlab = "Car weight (lbs/1000)",
            ylab = "Miles per gallon", las = 1,
            legend = list(coords="topright"),
            id = list(method="identify"),
            boxplots = "xy",col=2:4)
```

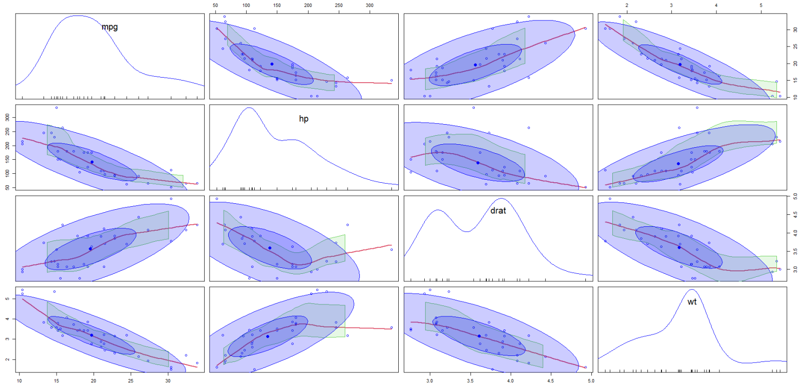
- Here we want to plot miles per gallon versus weight for cars that have 4, 6 or 8 cylinders. We write this as mpg ∼ wt | cyl.
- By default we get different colours for groups and both a linear and a smoothed line.
- A legend is included in the top right corner of the plot.
- The option id = list(method="identify") means that points can be identified by mouse clicks.
- Box plots of miles per gallon and weight included ("xy"option for both axes).
- More possibilities: ?scatterplot.

The resulting scatter plot



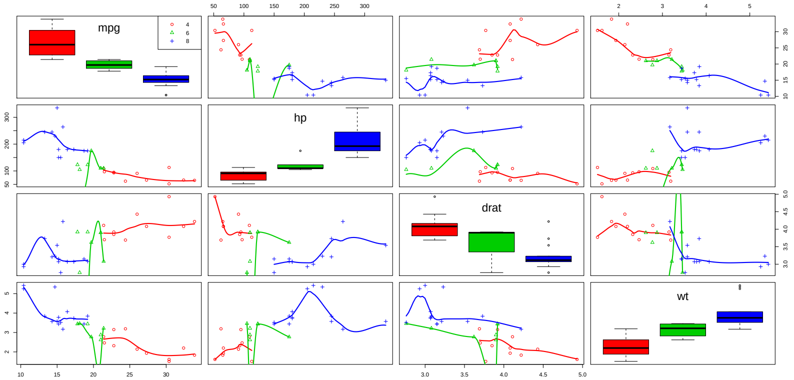
A scatter plot matrix from the "car"package

```
scatterplotMatrix( ~ mpg + hp + drat + wt, data = mtcars,
                   smooth = list(col.smooth=2,col.spread=3,
                                 lty.smooth=1),
                   regLine = FALSE,ellipse = TRUE)
```



A scatter plot matrix with boxplots

```
scatterplotMatrix( ~ mpg + hp + drat + wt | cyl,
                   diagonal = list(method="boxplot"),
                   smooth = list(lty.smooth=1),
                   regLine = FALSE, col = 2:6,data = mtcars )
```



Exercise

- Load the cdc data set.
- Make a basic scatter plot of weight and desired weight with “nice“ axes and labels.
- Add a smoothed lined to the plot.
- Make a scatter plot of weight and desired weight for each level of “genhlth“ in separate plots, with straight and smoothed lines.
- Install the package “car“.
- Make a scatter plot of weight and desired weight for each level of “genhlth“ in one plot.

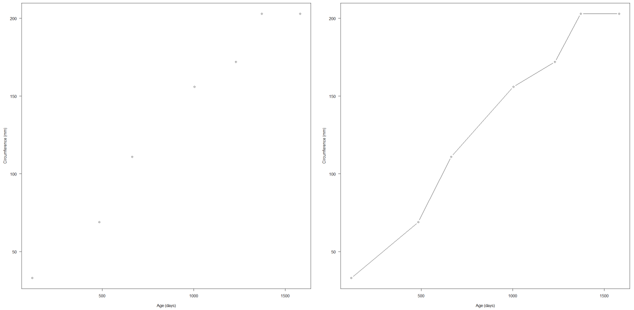
Exercise

- Load the Protein data set.
- Make a scatter plot matrix of “RedMeat“, “WhiteMeat“, “Eggs“, “Milk“ and “Fish“. Can you see any patterns in the protein intake?
- Also make a scatter plot matrix with box plots in the diagonal.

A Line Plot

Connecting points in a scatter plot from left to right. Here the growth of a tree. Notice the option type = “b“ meaning points joined by lines.

```
par(mfrow=c(1,2)),
plot(TreeA$age, TreeA$circumference, xlab = "Age (days)",
     ylab = "Circumference (mm)", las = 1)
plot(TreeA$age, TreeA$circumference, type = "b", xlab = "Age (days)",
     ylab = "Circumference (mm)", las = 1)
```

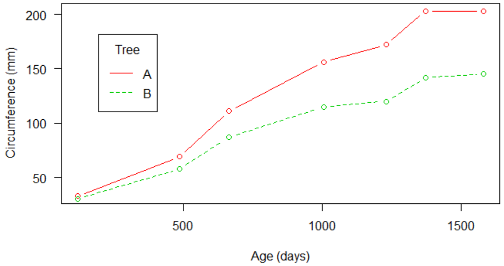


Difference between plot() and lines() functions

- We have seen both the plot and the lines functions.
- The plot function creates a new graph. It is a **high**-**level** plotting function.
- The lines function adds information to an existing graph but it cannot produce it's own graph. It is a **low**-**level** plotting function.
- A **high**-**level** plotting function can (often) be converted to a **low**-**level** plotting function with the option ADD=TRUE.
- Usually lines will be used after a **high**-**level** plotting function (such as plot) has produced a graph.

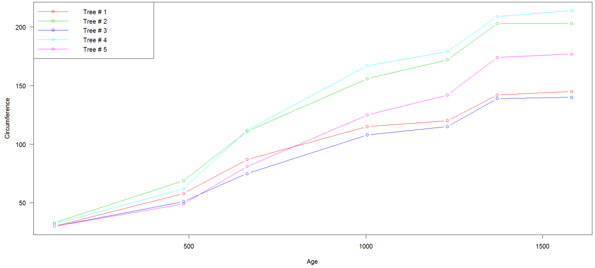
A line plot and a legend

```
plot(TreeA$age, TreeA$circumference, type = "b", lty = 1,
     xlab = "Age (days)",ylab = "Circumference (mm)", las = 1, col= 2)
lines(TreeB$age, TreeB$circumference, type = "b", col = 3, lty = 2)
legend(locator(1),          # we will place it with a mouse click
       legend = c("A","B"), title = "Tree",
       lty = 1:2, col= 2:3)
```



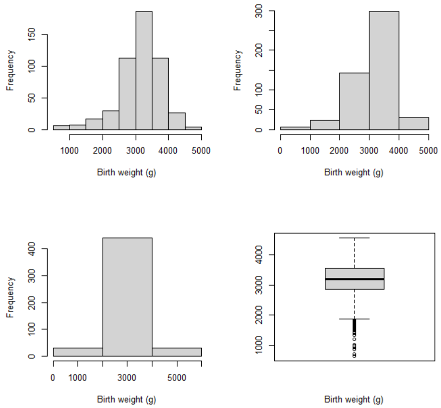
Plot of growth of 5 trees

```
treedata<-Orange
plot(treedata$age,treedata$circumference,pch=" ",xlab = 'Age',
     ylab='Circumference',las=1)
for(i in 1:5){lines(treedata$age[treedata$Tree==i],
                treedata$circumference[treedata$Tree==i],type="b",col=i+1)}
legend(x="topleft",paste("Tree #",1:5),pch=1,lty=1,col=2:6)
```



Layout of several plots on one graph

Several plots on one graph:



Use the option par(mfrow = c(2, 2)), and back to one plot par(mfrow = c(1, 1)).

Layout of several plots on one graph

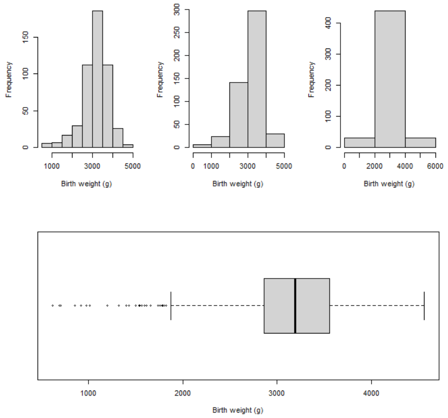
The layout() function:

- indicate in matrix form which part of the plot area that you wish to belong to which graph

```
> layout.matrix<-matrix(c(1:3,rep(4,3)),nrow=2,ncol=3,byrow=T)
> layout.matrix
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    4    4
```

Layout of several plots on one graph

```
>layout(layout.matrix)
```


Frequency
Birth weight (g)
Frequency
Birth weight (g)
Frequency
Birth weight (g)
Birth weight (g)