ESTP Course: The Use of R in Official Statistics: Examples

Alexander Kowarik, Bernhard Meindl
Vienna, April 2017

The Contractor is acting under a FWC concluded with the Commission.

Examples: Datatypes

Alexander Kowarik, Bernhard Meindl

Explanation

We have exercises on the following topics:

- Generation and replication of vectors
- Indexing and Subsetting
- Working with important data types
- dealing with missings values (**NA**)

Tasks / Exercises (1)

- 1a) Construct a vector of numbers from -3 to 2 (-3,-2,-1,0,1,2) using two different ways. Assign the vectors to objects *v1* and *v2*.
- 1b) Please check with the **identical()** function if *v1* and *v2* are exactly the same.
- 1c) Create the following sequence of data points and assign the result to an object *v*:
  - 2, 3, 4, 5, 6, 7, 8, 4.1, 4.2, 4.3, 4.4, 4.1, 4.2, 4.3, 4.4, 4.1, 4.2, 4.3, 4.4, -3, -2, -1, 0, 1
- 1d) Replicate *v* 2 times
- 1e) Replicate each element of *v* exactly 2 times
- 1f) Report the number of elements of *v*

Solution (1a, 1b)

- Constructing vectors and checking whether these vectors are equal

```
v1 <- -3:2; v1
```

```
[1] -3 -2 -1  0  1  2
```

```
v2 <- seq(from = -3, to = 2); v2
```

```
[1] -3 -2 -1  0  1  2
```

```
identical(v1, v2)
```

```
[1] TRUE
```

Solution (1c, 1d, 1e, 1f)

- Construction of sequencies and replications

```
v <- c(2:8, rep(seq(4.1,4.4, by=0.1), 3), -3:1); v
```

```
 [1]  2.0  3.0  4.0  5.0  6.0  7.0  8.0  4.1  4.2  4.3  4.4  4.1  4.2  4.3
[15]  4.4  4.1  4.2  4.3  4.4 -3.0 -2.0 -1.0  0.0  1.0
```

```
rep(v, times=2)
```

```
 [1]  2.0  3.0  4.0  5.0  6.0  7.0  8.0  4.1  4.2  4.3  4.4  4.1  4.2  4.3
[15]  4.4  4.1  4.2  4.3  4.4 -3.0 -2.0 -1.0  0.0  1.0  2.0  3.0  4.0  5.0
[29]  6.0  7.0  8.0  4.1  4.2  4.3  4.4  4.1  4.2  4.3  4.4  4.1  4.2  4.3
[43]  4.4 -3.0 -2.0 -1.0  0.0  1.0
```

```
rep(v, each=2)
```

```
 [1]  2.0  2.0  3.0  3.0  4.0  4.0  5.0  5.0  6.0  6.0  7.0  7.0  8.0  8.0
[15]  4.1  4.1  4.2  4.2  4.3  4.3  4.4  4.4  4.1  4.1  4.2  4.2  4.3  4.3
[29]  4.4  4.4  4.1  4.1  4.2  4.2  4.3  4.3  4.4  4.4 -3.0 -3.0 -2.0 -2.0
[43] -1.0 -1.0  0.0  0.0  1.0  1.0
```

```
length(v)
```

```
[1] 24
```

Tasks / Exercises (2)

- 2a) Create a character vector with the first 15 capital letters of the alphabet (Note: *? Constants*) and assign the result to a vector *s*
- 2b) Extract from *s* the letters in odd positions (1,3, …)
- 2c) Extract from *s* the letters at even positions (2,4, …)
- 2d) Extract from *s* all letters from (including) the letter *D*

Solution (2a, 2b, 2c, 2d)

- Indexing of vectors

```
s <- LETTERS[1:15]; s
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
```

- Vector of odd / even positions

```
s[seq(1, length(s), by=2)] # s set at odd
```

```
[1] "A" "C" "E" "G" "I" "K" "M" "O"
```

```
s[seq(2, length(s), by=2)] # s set at odd
```

```
[1] "B" "D" "F" "H" "J" "L" "N"
```

- negative indexing

```
s[-c(1:3)] # s from D onwards
```

```
 [1] "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"
```

Tasks / Exercises (3)

- 3a) Create a numeric vector *v1* of length 10 and values from -3 to 3. (Note: use **seq()** or **sample()**)
- 3b) Construct a logical vector, with a **TRUE** for all negative values of *v1*
- 3c) extract from *v1* all elements that are TRUE and interpret the result
- 3d) How many elements does the result of *3c* has?

Solution (3a, 3b, 3c, 3d)

- Creation of numeric vectors

```
v1 <- sample(-3:3, 10, replace=TRUE); v1
```

```
 [1] -2  0  2  3 -3 -2  3  0  0  0
```

- Create a logical vector

```
index <- v1 < 0; index
```

```
 [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

- Logical indexing

```
v2 <- v1[index]; v2 # only negative values!
```

```
[1] -2 -3 -2
```

```
length(v2)
```

```
[1] 3
```

Tasks / Exercises (4)

- 4a) Create a numeric vector *v1* with the integers from 0 to 10 and a numeric vector *v2* with the numbers 0 to 15 (step 1).
- 4b) Sum up the elements of *v1* and report the sum (**?sum**).
- 4c) calculate the element-wise differences between *v1 v2*. Interpret the result.
- 4d) Report the unique elements of the result of *4c*

Solution (4a, 4b)

- Generating the numerical vectors

```
v1 <- 0:10; v1
```

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

```
v2 <- 0:15; v2
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

- Working with vectors

```
sum(v1)
```

```
[1] 55
```

Solution (4c, 4d)

- differences (attention: recycling!)

```
res <- v1-v2; res # element-wise differences education, recycling v1
```

```
 [1]   0   0   0   0   0   0   0   0   0   0   0 -11 -11 -11 -11 -11
```

- Unique elements

```
unique(res)
```

```
[1]   0 -11
```

Tasks / Exercises (5)

- 5a) Create a matrix *mat* with 5 rows and 10 columns with the numbers of 1:50. Fill the matrix line by line.

  indexing is similar to data.frames –> more exercises later

Solution (5a, 5b)

construct matrix

```
mat <- matrix(1:50, nrow=5, ncol=10, byrow=TRUE); mat
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
[2,]   11   12   13   14   15   16   17   18   19    20
[3,]   21   22   23   24   25   26   27   28   29    30
[4,]   31   32   33   34   35   36   37   38   39    40
[5,]   41   42   43   44   45   46   47   48   49    50
```

Tasks / Exercises (6)

- 6a) Create a matrix *mat1* with 2 rows and 4 columns from the vector *mat1* as given by using the function **dim()**:

```
mat1 <- 1:8
```

- 6b) Create a matrix *mat2* with 2 rows and 3 columns from the vector *mat2* by using the function **dim()**:

```
mat2 <- 0:-5
```

- 6c) Combine the two matrices *mat1* and *mat2* (columns) and assign the result to the object *mat*.
- 6d) Calculate the dimension of *mat*

Solution (6a, 6b, 6c)

- Creating matrices from vectors by setting the Dimension Attributes

```
mat1 <- 1:8; dim(mat1) <- c(2,4)
mat2 <- -0:-5; dim(mat2) <- c(2,3)
```

- combinde

```
mat <- cbind(mat1, mat2); mat
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    3    5    7    0   -2   -4
[2,]    2    4    6    8   -1   -3   -5
```

- dimension

```
dim(mat)
```

```
[1] 2 7
```

Tasks / Exercises (7)

- 7a) Create a list *ll* with a total of 3 list items:
  - *V1*: numeric vector of length 10
  - *V2*: character vector of length 15
  - *V3*: logical vector of length 5
- 7b) Extract from *ll* element *v2*, and assign it to a vector
- 7c) Extract from *ll* element *v3* and assign it to a vector
- 7d) Extract from *ll* the first three elements of *v1* and assign it to a vector
- Give back the name of the list items

Solution (7a, 7b)

- Create a list

```
ll <- list(v1=1:10, v2=letters[1:15], v3=sample(c(TRUE,FALSE), 5, replace=TRUE))
ll
```

```
$v1
 [1]  1  2  3  4  5  6  7  8  9 10

$v2
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"

$v3
[1] FALSE  TRUE  TRUE  TRUE FALSE
```

- Extract of list items in a list

```
res <- ll[2]; res # analog: ll["v2"]
```

```
$v2
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
```

```
class(res)
```

```
[1] "list"
```

Solution (7c, 7d, 7e)

- Extract of list elements as vectors

```
res <- ll[[3]]; res # analog res <- ll[["v3"]]
```

```
[1] FALSE  TRUE  TRUE  TRUE FALSE
```

```
is.vector(res)
```

```
[1] TRUE
```

- Extract and index

```
res <- ll[["v1"]][1:3]; res
```

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

- Name of the list items

```
names(ll)
```

```
[1] "v1" "v2" "v3"
```

Tasks / Exercises (8)

- 8a) Create a character vector *v1* of length 15 with the forms *"bad"*, *"medium"* and *"good"*
- 8b) Convert *v1* into a unordered factor *ff*.
- 8c) Change the characteristics of the factor to *"G"* (instead of *"good"*), *"M"* (instead of *"medium"*) and *"B"* (instead of *"bad"*).
- 8d) Convert for *ff* the level *"M"* and *"B"* into a new category *medium-bad* together. In addition rename *"G"* again to *good*
- 8e) Create a logical vector which is exactly TRUE if the level *medium-bad* applies.
- 8f) report the number of elements of this category?

Solution (8a, 8b, 8c)

- Create a unordered factor

```
v1 <- sample(c("good","medium","bad"), 10, replace=TRUE); v1
```

```
 [1] "bad"    "medium" "medium" "medium" "bad"    "good"   "bad"
 [8] "medium" "medium" "bad"
```

```
ff <- factor(v1); ff
```

```
 [1] bad    medium medium medium bad    good   bad    medium medium bad
Levels: bad good medium
```

- Change Levels

```
levels(ff) <- c("B","G","M"); ff
```

```
 [1] B M M M B G B M M B
Levels: B G M
```

Solution (8d, 8e, 8f)

- Summarize and change levels

```
levels(ff) <- c("medium-bad", "good", "medium-bad"); ff
```

```
 [1] medium-bad medium-bad medium-bad medium-bad medium-bad good
 [7] medium-bad medium-bad medium-bad medium-bad
Levels: medium-bad good
```

Create a logical vector -

```
index <- ff == "medium-bad"; index
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

- How many elements of *ff* are *medium-bad*?

```
sum(index) # alternative: length (which (index == TRUE))
```

```
[1] 9
```

Tasks / Exercises (9)

- 9a) Create from the vector *v1* from Example 8a) an ordered factor *fo*. Pay attention to the order of the levels.
- 9b) Convert *fo* into a numeric vector.
- 9c) Convert *fo* into a character vector.

Solution (9a, 9b, 9c)

- Create a parent factor

```
v1 <- sample(c("good","medium","bad"), 10, replace = TRUE); v1
```

```
 [1] "medium" "good"   "medium" "bad"    "good"   "medium" "medium"
 [8] "medium" "good"   "bad"
```

```
fo <- factor(v1, ordered = TRUE, levels = c("bad","medium","good")); fo
```

```
 [1] medium good   medium bad    good   medium medium medium good   bad
Levels: bad < medium < good
```

- Generate numerical / character vector from factor

```
as.numeric(fo)
```

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

```
as.character(fo)
```

```
 [1] "medium" "good"   "medium" "bad"    "good"   "medium" "medium"
 [8] "medium" "good"   "bad"
```

Tasks / Exercises (10)

- 10a) Create a Data Frame *df* with:
  - Variable *"a"*: capital letters from *A* to *F*
  - Variable *"v1"*: Numbers 1:6
  - Variable *"v2"*: logical vector which is TRUE if *v1* is even and FALSE otherwise
- 10b) report the structure of *df*
- 10c) make sure that variable *a* is of type character
- 10d) calculate dimension of *df*
- 10e) extract the third and fourth lines of *df*
- 10f) extract in two different ways variable *v1* from *df*
- 10g) return the variable names *df*

Solution (10a, 10b)

Create a data frame -

```
df <- data.frame(a=LETTERS[1:6], v1=1:6, v2=FALSE); df # recycling!
```

```
  a v1    v2
1 A  1 FALSE
2 B  2 FALSE
3 C  3 FALSE
4 D  4 FALSE
5 E  5 FALSE
6 F  6 FALSE
```

```
df$v2[seq(2,nrow(df),2)] <- TRUE; str(df)
```

```
'data.frame':   6 obs. of  3 variables:
 $ a : Factor w/ 6 levels "A","B","C","D",..: 1 2 3 4 5 6
 $ v1: int  1 2 3 4 5 6
 $ v2: logi  FALSE TRUE FALSE TRUE FALSE TRUE
```

Solution (10c, 10d)

- No automatic conversion of character vectors

```
df <- data.frame(a=LETTERS[1:6], v1=1:6, v2=FALSE, stringsAsFactors=FALSE);
df$v2[seq(2,nrow(df),2)] <- TRUE; str(df)
```

```
'data.frame':   6 obs. of  3 variables:
 $ a : chr  "A" "B" "C" "D" ...
 $ v1: int  1 2 3 4 5 6
 $ v2: logi  FALSE TRUE FALSE TRUE FALSE TRUE
```

- Dimension of a data frame

```
dim(df) # num. lines: nrow (df); num. column: ncol (df)
```

```
[1] 6 3
```

Solution (10e, 10f, 10g)

- Extract lines

```
df[3:4,] # two lines, all columns
```

```
  a v1    v2
3 C  3 FALSE
4 D  4  TRUE
```

- Extracting variables

```
identical(df$v1, df[,2])
```

```
[1] TRUE
```

- column names

```
colnames(df)
```

```
[1] "a"  "v1" "v2"
```

Tasks / Exercises (11)

- 11a) Use the Cars93 data from package MASS. (use **install.packages("MASS")** if the package is not installed). Load the data via **data(Cars93, package="MASS")**. Inspect the data.
- 11b) Extract from *Cars93* the value in the second row and third column
- 11c) Extract from *Cars93* the first and fourth lines.
- 11d) Select from *Cars93* all vehicles (rows) that costs less than 7 (*Min.Price*)

Solution (11a) - Cars93

```
data(Cars93, package="MASS"); str(Cars93)
```

```
'data.frame':   93 obs. of  27 variables:
 $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
 $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
 $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
 $ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
 $ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
 $ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
 $ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
 $ EngineSize        : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
 $ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
 $ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
 $ Man.trans.avail   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
 $ Fuel.tank.capacity: num  13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
 $ Passengers        : int  5 5 5 6 4 6 6 6 5 6 ...
 $ Length            : int  177 195 180 193 186 189 200 216 198 206 ...
 $ Wheelbase         : int  102 115 102 106 109 105 111 116 108 114 ...
 $ Width             : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Turn.circle       : int  37 38 37 37 39 41 42 45 41 43 ...
 $ Rear.seat.room    : num  26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
 $ Luggage.room      : int  11 15 14 17 13 16 17 21 14 18 ...
 $ Weight            : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
 $ Origin            : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
 $ Make              : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

Solution (11a) - Cars93

```
colnames(Cars93)
```

```
 [1] "Manufacturer"       "Model"              "Type"
 [4] "Min.Price"          "Price"              "Max.Price"
 [7] "MPG.city"           "MPG.highway"        "AirBags"
[10] "DriveTrain"         "Cylinders"          "EngineSize"
[13] "Horsepower"         "RPM"                "Rev.per.mile"
[16] "Man.trans.avail"    "Fuel.tank.capacity" "Passengers"
[19] "Length"             "Wheelbase"          "Width"
[22] "Turn.circle"        "Rear.seat.room"     "Luggage.room"
[25] "Weight"             "Origin"             "Make"
```

```
head(Cars93,2)
```

```
  Manufacturer   Model    Type Min.Price Price Max.Price MPG.city
1        Acura Integra   Small      12.9  15.9      18.8       25
2        Acura  Legend Midsize      29.2  33.9      38.7       18
  MPG.highway            AirBags DriveTrain Cylinders EngineSize
1          31               None      Front         4        1.8
2          25 Driver & Passenger      Front         6        3.2
  Horsepower  RPM Rev.per.mile Man.trans.avail Fuel.tank.capacity
1        140 6300         2890             Yes                13.2
2        200 5500         2335             Yes                18.0
  Passengers Length Wheelbase Width Turn.circle Rear.seat.room
1          5    177       102    68          37           26.5
2          5    195       115    71          38           30.0
  Luggage.room Weight  Origin          Make
1           11   2705 non-USA Acura Integra
2           15   3560 non-USA  Acura Legend
```

Solution (11b, 11c)

- Extracting cells

```
Cars93[2,3]
```

```
[1] Midsize
Levels: Compact Large Midsize Small Sporty Van
```

- Extract lines

```
Cars93[c(1,4),]
```

```
  Manufacturer   Model    Type Min.Price Price Max.Price MPG.city
1        Acura Integra   Small      12.9  15.9      18.8       25
4         Audi     100 Midsize      30.8  37.7      44.6       19
  MPG.highway            AirBags DriveTrain Cylinders EngineSize
1          31               None      Front         4        1.8
4          26 Driver & Passenger      Front         6        2.8
  Horsepower  RPM Rev.per.mile Man.trans.avail Fuel.tank.capacity
1        140 6300         2890             Yes               13.2
4        172 5500         2535             Yes               21.1
  Passengers Length Wheelbase Width Turn.circle Rear.seat.room
1          5    177       102    68          37           26.5
4          6    193       106    70          37           31.0
  Luggage.room Weight  Origin          Make
1           11   2705 non-USA Acura Integra
4           17   3405 non-USA      Audi 100
```

Solution (11d)

- Subsetting a data.frame

```
Cars93[Cars93$Min.Price < 7,]
```

```
   Manufacturer   Model  Type Min.Price Price Max.Price MPG.city
31         Ford Festiva Small       6.9   7.4       7.9       31
39          Geo   Metro Small       6.7   8.4      10.0       46
44      Hyundai   Excel Small       6.8   8.0       9.2       29
   MPG.highway AirBags DriveTrain Cylinders EngineSize Horsepower  RPM
31          33    None      Front         4        1.3         63 5000
39          50    None      Front         3        1.0         55 5700
44          33    None      Front         4        1.5         81 5500
   Rev.per.mile Man.trans.avail Fuel.tank.capacity Passengers Length
31         3150             Yes                10.0          4    141
39         3755             Yes                10.6          4    151
44         2710             Yes                11.9          5    168
   Wheelbase Width Turn.circle Rear.seat.room Luggage.room Weight  Origin
31        90    63          33           26.0           12   1845     USA
39        93    63          34           27.5           10   1695 non-USA
44        94    63          35           26.0           11   2345 non-USA
           Make
31  Ford Festiva
39     Geo Metro
44 Hyundai Excel
```

Tasks / Exercises (12)

- 12a) install package **VIM** and load the package via **library(VIM)**
- 12b) load data set **sleep** using **data(sleep)**
- 12c) how many missing values are in the data set *sleep*?
- 12d) report the percentage of missing values in variable *Dream*
- 12e) set all NA-values in variable *Dream* to the mean of observed values in *Dream*

Solution (12a, 12b, 12c)

- install the package

```
install.packages("VIM")
```

- small look at the data

```
library(VIM)
data(sleep)
str(sleep)
```

```
'data.frame':   62 obs. of  10 variables:
 $ BodyWgt : num  6654 1 3.38 0.92 2547 ...
 $ BrainWgt: num  5712 6.6 44.5 5.7 4603 ...
 $ NonD    : num  NA 6.3 NA NA 2.1 9.1 15.8 5.2 10.9 8.3 ...
 $ Dream   : num  NA 2 NA NA 1.8 0.7 3.9 1 3.6 1.4 ...
 $ Sleep   : num  3.3 8.3 12.5 16.5 3.9 9.8 19.7 6.2 14.5 9.7 ...
 $ Span    : num  38.6 4.5 14 NA 69 27 19 30.4 28 50 ...
 $ Gest    : num  645 42 60 25 624 180 35 392 63 230 ...
 $ Pred    : int  3 3 1 5 3 4 1 4 1 1 ...
 $ Exp     : int  5 1 1 2 5 4 1 5 2 1 ...
 $ Danger  : int  3 3 1 3 4 4 1 4 1 1 ...
```

- abount of missings

```
sum(is.na(sleep))
```

```
[1] 38
```

Solution (12d, 12e)

- percentage of missings in Dream

```
nm <- sum(is.na(sleep$Dream))
100 * nm / length(sleep$Dream)
```

```
[1] 19.35484
```

- mean imputation of Dream

```
sleep$Dream
```

```
 [1]  NA 2.0  NA  NA 1.8 0.7 3.9 1.0 3.6 1.4 1.5 0.7 2.7  NA 2.1 0.0 4.1
[18] 1.2 1.3 6.1 0.3 0.5 3.4  NA 1.5  NA 3.4 0.8 0.8  NA  NA 1.4 2.0 1.9
[35] 2.4 2.8 1.3 2.0 5.6 3.1 1.0 1.8 0.9 1.8 1.9 0.9  NA 2.6 2.4 1.2 0.9
[52] 0.5  NA 0.6  NA 2.2 2.3 0.5 2.6 0.6 6.6  NA
```

```
m <- mean(sleep$Dream, na.rm=TRUE)
sleep[is.na(sleep$Dream), "Dream"] <- m
sleep$Dream
```

```
 [1] 1.972 2.000 1.972 1.972 1.800 0.700 3.900 1.000 3.600 1.400 1.500
[12] 0.700 2.700 1.972 2.100 0.000 4.100 1.200 1.300 6.100 0.300 0.500
[23] 3.400 1.972 1.500 1.972 3.400 0.800 0.800 1.972 1.972 1.400 2.000
[34] 1.900 2.400 2.800 1.300 2.000 5.600 3.100 1.000 1.800 0.900 1.800
[45] 1.900 0.900 1.972 2.600 2.400 1.200 0.900 0.500 1.972 0.600 1.972
[56] 2.200 2.300 0.500 2.600 0.600 6.600 1.972
```

Examples: Functions

Alexander Kowarik, Bernhard Meindl

Explanation

We have exercises on the following topics:

- Mathematical and statistical functions
- Functions for probability distributions
- Functions handling character vectors
- Own functions and control structures

Tasks / Exercises (1)

- 1a) Create a vector *x* which contains 8 random numbers from a normal distribution with mean 10 and standard deviation 20. (?**rnorm**)
- 1b) Create a vector *y* which contains 8 random numbers from a uniform distribution from -10 to 10 (?**runif**)
- 1c) Round *x* to the next (larger) integer
- 1d) Round *y* to the next (smaller) integer

Solution (1a, 1b, 1c, 1d)

- Generating random numbers

```
x <- rnorm(8, mean=10, sd=20); x
```

```
[1]  -4.306490  -6.585802  -3.168930  31.669076 -24.891672 -26.920540
[7] -11.670708   5.408974
```

```
y <- runif(8, min=-10, max=10); y
```

```
[1]  4.627881 -3.405145 -6.336393 -6.589241  4.233122 -2.032591 -8.214601
[8]  9.358393
```

- Rounding

```
ceiling(x) # next (larger) integer
```

```
[1]  -4  -6  -3  32 -24 -26 -11   6
```

```
floor(y) # next (smaller) integer
```

```
[1]  4 -4 -7 -7  4 -3 -9  9
```

Tasks / Exercises (2)

- 2a) Calculate the base-10 logarithm of the absolute values of the vector *x*
- 2b) Sum up the values of *x* and *y*, and compute the absolute value of the difference
- 2c) Calculate the remainder of the vector *1:100* after dividing by 5

Solution (2a, 2b, 2c)

- Calculate the logarithm of the absolute values

```
log10(abs(x))
```

```
[1] 0.6341234 0.8186087 0.5009126 1.5006354 1.3960541 1.4300838 1.0670972
[8] 0.7331149
```

- mathematical operations

```
abs(sum(x) - sum(y))
```

```
[1] 32.10751
```

- rest after division

```
1:100 %% 5
```

```
  [1] 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0
 [36] 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0
 [71] 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0
```

Tasks / Exercises (3)

- 3) Load the data set *Cars93* with **data(Cars93, package="MASS")**
- 3b) Set randomly any 5 observations in the variables *Horsepower* and *Weight* to *NA* (missings)
- 3c) Calculate the arithmetic mean and the median of the variables *Horsepower* and *Weight*
- 3d) Calculate the standard deviation and the interquartile range of the variable *Price*

Solution (3a, 3b, 3c, 3d)

- Load data and setting missings

```
data(Cars93,package="MASS")
Cars93[sample(1:nrow(Cars93),5), c("Horsepower","Weight")] <- NA
```

- Calculate the mean and the median

```
c(mean(Cars93$Horsepower, na.rm=TRUE), median(Cars93$Horsepower, na.rm=TRUE))
```

```
[1] 144.0909 140.0000
```

```
c(mean(Cars93$Weight, na.rm=TRUE), median(Cars93$Weight, na.rm=TRUE))
```

```
[1] 3059.659 3035.000
```

- Calculation of the standard deviation and IQR

```
c(sd(Cars93$Price),IQR(Cars93$Price)) # no missing values
```

```
[1]  9.65943 11.10000
```

Tasks / Exercises (4)

- 4a) Calculate the correlation matrix (Pearson) of the variables *Horsepower*, *Weight* and *Price*. Use only the complete observations.
- 4b) Calculate the empirical covariance matrix of the variables *Horsepower*, *Weight* and *Price*. Use only the complete observations.
- 4c) Calculate the range of the variable *Price*.
- 4d) Provide a summary of the variables *Weight* and *Price*.

Solution (4a, 4b)

- Calculate the correlation matrix

```
cor(Cars93[,c("Horsepower","Weight","Price")], use ="complete.obs")
```

```
           Horsepower    Weight     Price
Horsepower  1.0000000 0.7513704 0.7918655
Weight      0.7513704 1.0000000 0.6589055
Price       0.7918655 0.6589055 1.0000000
```

- Calculate the covariance matrix

```
cov(Cars93[,c("Horsepower","Weight","Price")], use ="complete.obs")
```

```
           Horsepower     Weight      Price
Horsepower  2883.4859  24127.560  421.11735
Weight     24127.5601 357602.756 3902.26110
Price        421.1173   3902.261   98.08115
```

Solution (4c, 4d)

- Range

```
range(Cars93[,"Price"])
```

```
[1]  7.4 61.9
```

- Summary of numerical vectors

```
summary(Cars93[,"Weight"])
summary(Cars93[,"Price"])
```

Tasks / Exercises (5)

- 5a) The function **date()** returns the current time as a string. Assign the result to an object *s*.
- 5b) How many characters does *s* have?
- 5c) Convert *s* to uppercase letters and assign the result back to *s*.
- 5d) Delete the first 4 characters of *s* and assign the result back to *s*.
- 5e) Check if the pattern *"2015"* appears in *s*.
- 5f) Check if the pattern *"2016"* appears in *s*.

Solution (5a, 5b, 5c)

- Generate character vector

```
s <- date(); s
```

```
[1] "Wed Feb 15 07:57:42 2017"
```

- Calculate number of characters

```
nchar(s)
```

```
[1] 24
```

- Manipulate strings

```
s <- toupper(s); s
```

```
[1] "WED FEB 15 07:57:42 2017"
```

Solution (5d)

- Manipulate strings

```
s <- substring(s, first=5, last=nchar(s)); s
```

```
[1] "FEB 15 07:57:42 2017"
```

- Matching

```
grep("2015",s)
```

```
integer(0)
```

```
grep("2016",s)
```

```
integer(0)
```

Tasks / Exercises (6)

- 6a) Write a function **myrange()** which takes as an argument a numeric vector *x*.

  The function should

- return the difference between the maximum and minimum value of *x*
- ignore any missing values

- return *NA* if *x* is not numeric
  - 6b) and then test your function.

Solution (6a, 6b)

- Write your own function

```
myrange <- function(x) {
  if ( !is.numeric(x) ) {
    return(NA)
  }
  return(max(x, na.rm=TRUE) - min(x, na.rm=TRUE))
}

myrange(x=1:5)
```

```
[1] 4
```

```
myrange(x=c("a","b"))
```

```
[1] NA
```

Tasks / Exercises (7)

- 7a) Write a function **myMat(n, m, minVal, maxVal)**. The function should return a matrix with *n* rows and columns *m*. The matrix should be filled with random integers between *minVal* and *maxVal*.

myMat() should result in *NA* if one of the following conditions is met:

- *maxVal* is smaller than *minVal*
- *n* or *m* are smaller than 1

- the length of *n* or *m* is different from 1
  - 7b) Test the function.
  - 7c) How could we improve the function?

Solution (7a)

- Write your own function:

```
myMat <- function(n, m, minVal, maxVal) {
  if( n < 1 | m < 1) {
    return(NA)
  }
  if (minVal > maxVal) {
    return(NA)
  }
  if ( length(n)!= 1 | length(m) != 1 ) {
    return(NA)
  }
  m <- matrix(nrow=n,ncol=m,sample(round(minVal):round(maxVal), n*m, replace=TRUE))
  return(m)
}
```

Solution (7b, 7c)

- Testing the function:

```
myMat(n=3, m=5, minVal=4, maxVal=2)
```

```
[1] NA
```

```
myMat(n=c(3,5), m=5, minVal=4, maxVal=2)
```

```
[1] NA
```

```
myMat(n=3, m=5, minVal=2, maxVal=7)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    6    6    3    6    7
[2,]    6    3    5    6    5
[3,]    3    4    3    7    5
```

- possible improvement: Are all inputs numeric?

```
myMat(n="a", m=5, minVal=4, maxVal=2) # implicit :)
```

```
[1] NA
```

Examples: Tables

Alexander Kowarik, Bernhard Meindl

Data set Cars93

- Use the data set **Cars93** from the package **MASS**
- We are interested in the variables:
  - *Type*: type of the car
  - *Origin*: produces in the US or outside US
  - *AirBags*: is an airbag included, and where
- Use the functions **table()**, **prop.table()**, **margin.table()**, and **addmargins()** for the following exercises.

Tasks / Exercises (1)

- 1a) Load the data set **Cars93** from **library(MASS)**, and look at the available variables using **names(Cars93)**
- 1b) Look at the values of the first few observations of *Type*, *Origin* and *AirBags*, using **head()**
- 1c) Use **table()** to look at the frequencies of these single variables
- 1d) Are any missing values in our data?
- 1e) Generate a contingency table for *Type* versus *AirBags*
- 1f) Compute all marginal sums for this table.

Solution (1a)

- Load and view the data

```
data(Cars93,package="MASS")
names(Cars93)
```

```
 [1] "Manufacturer"       "Model"              "Type"
 [4] "Min.Price"          "Price"              "Max.Price"
 [7] "MPG.city"           "MPG.highway"        "AirBags"
[10] "DriveTrain"         "Cylinders"          "EngineSize"
[13] "Horsepower"         "RPM"                "Rev.per.mile"
[16] "Man.trans.avail"    "Fuel.tank.capacity" "Passengers"
[19] "Length"             "Wheelbase"          "Width"
[22] "Turn.circle"        "Rear.seat.room"     "Luggage.room"
[25] "Weight"             "Origin"             "Make"
```

```
dim(Cars93)
```

```
[1] 93 27
```

Solution (1b)

- The part interesting for us:

```
head(Cars93[,c("Type","Origin","AirBags")])
```

```
     Type  Origin             AirBags
1   Small non-USA                None
2 Midsize non-USA Driver & Passenger
3 Compact non-USA         Driver only
4 Midsize non-USA Driver & Passenger
5 Midsize non-USA         Driver only
6 Midsize     USA         Driver only
```

Solution (1c)

- Absolute frequencies:

```
table(Cars93$Type)
```

```
Compact   Large Midsize   Small  Sporty     Van
     16      11      22      21      14       9
```

```
table(Cars93$Origin)
```

```
    USA non-USA
     48      45
```

```
table(Cars93$AirBags)
```

```
Driver & Passenger        Driver only               None
                16                 43                 34
```

Solution (1d)

- Are there missing values?

```
table(Cars93$Type, useNA="always")
```

```
Compact   Large Midsize   Small  Sporty     Van    <NA>
     16      11      22      21      14       9       0
```

```
table(Cars93$Origin, useNA="always")
```

```
    USA non-USA    <NA>
     48      45       0
```

```
table(Cars93$AirBags, useNA="always")
```

```
Driver & Passenger        Driver only               None
                16                  43                 34
              <NA>
                 0
```

Solution (1e)

- Contingency table for *Type* x *AirBags*

```
table(Cars93[,c("Type","AirBags")])
```

```
         AirBags
Type      Driver & Passenger Driver only None
  Compact                  2           9    5
  Large                    4           7    0
  Midsize                  7          11    4
  Small                    0           5   16
  Sporty                   3           8    3
  Van                      0           3    6
```

Solution (1f)

- Marginal sums

```
tab <- table(Cars93[,c("Type","AirBags")])
margin.table(tab) # total
```

```
[1] 93
```

```
margin.table(tab,1) # by Type (rows)
```

```
Type
Compact   Large Midsize   Small  Sporty     Van
     16      11      22      21      14       9
```

```
margin.table(tab,2) # by AirBags (columns)
```

```
AirBags
Driver & Passenger        Driver only               None
                16                 43                 34
```

Tasks / Exercises (2)

- 2a) Generate a contingency table for *Type* versus *AirBags* with relative frequencies using **prop.tab()**
- 2b) Use **addmargins()** to add marginal sums to the above table.
- 2c) Use in addition to **Type** and **AirBags** also the variable **Origin** to create a contingency table with relative frequencies, including marginal sums.

Solution (2a)

- Contingency table for *Type* x *AirBags*

```
tab <- table(Cars93[,c("Type","AirBags")])
prop.table(tab)
```

```
         AirBags
Type      Driver & Passenger Driver only       None
  Compact          0.02150538  0.09677419 0.05376344
  Large            0.04301075  0.07526882 0.00000000
  Midsize          0.07526882  0.11827957 0.04301075
  Small            0.00000000  0.05376344 0.17204301
  Sporty           0.03225806  0.08602151 0.03225806
  Van              0.00000000  0.03225806 0.06451613
```

Solution (2b)

- Add marginal sums:

```
addmargins(prop.table(tab))
```

```
         AirBags
Type      Driver & Passenger Driver only       None        Sum
  Compact         0.02150538  0.09677419 0.05376344 0.17204301
  Large           0.04301075  0.07526882 0.00000000 0.11827957
  Midsize         0.07526882  0.11827957 0.04301075 0.23655914
  Small           0.00000000  0.05376344 0.17204301 0.22580645
  Sporty          0.03225806  0.08602151 0.03225806 0.15053763
  Van             0.00000000  0.03225806 0.06451613 0.09677419
  Sum             0.17204301  0.46236559 0.36559140 1.00000000
```

Solution (2c)

- Add marginal sums:

```
tt <- addmargins(prop.table(table(Cars93[,c("Type","AirBags","Origin")])))
tt[,,1] # non-usa
```

```
         AirBags
Type      Driver & Passenger Driver only       None        Sum
  Compact          0.01075269  0.02150538 0.04301075 0.07526882
  Large            0.04301075  0.07526882 0.00000000 0.11827957
  Midsize          0.02150538  0.05376344 0.03225806 0.10752688
  Small            0.00000000  0.02150538 0.05376344 0.07526882
  Sporty           0.02150538  0.05376344 0.01075269 0.08602151
  Van              0.00000000  0.02150538 0.03225806 0.05376344
  Sum              0.09677419  0.24731183 0.17204301 0.51612903
```

```
#tt[,,2] # usa
#tt[,,3] # total
```

Examples: Import / Export

Alexander Kowarik, Bernhard Meindl

Show object contents

- Show contents of simple R objects with cat()

```
x <- 100
cat(x)
```

```
100
```

```
cat("Iteration",x," was completed \n", sep=" ")
```

```
Iteration 100  was completed
```

Tasks / Exercises (1)

- 1a) Calculate the mean of the numbers 5,7,9,5
- 1b) Write the result into a text file with a sentence like “The resulting value is 6.5”
- 1c) Round the value before it is written to the text file.

Solution (1a, 1b, 1c)

- Generate the input

```
x <- c (5,7,9,5)
```

- Write text with result of calculation to a file

```
cat("The resulting value is",mean(x),"\n", file="test.txt")
```

- Additional calculation

```
cat ("The rounded value is",round(mean(x)),"\n", file="test.txt")
```

Tasks / Exercises (2)

- 2a) At *http://osrp01/estpr/* you can find several files with "Cars" in the file name. Read the file **CarsA.csv** using **read.csv2()**, and save it to an object **CarsA**.
- 2b) Look at the object, and apply **str()** to this object. Are all variables correctly represented?
- 2c) Save the object locally on your computer as CSV file.
- 2d) Reload the previously saved file. Do you get the correct data structure?

Solution (2a, 2b)

- Read *CarsA.csv* into an object **CarsA** and inspect the result

```
CarsA <- read.csv2("http://osrp01/estpr/CarsA.csv")
head(CarsA)
```

```
  Manufacturer   Model Price            AirBags Horsepower Width Weight
1        Acura Integra  15.9               None        140    68   2705
2        Acura  Legend  33.9 Driver & Passenger        200    71   3560
3         Audi      90  29.1        Driver only        172    67   3375
4         Audi     100  37.7 Driver & Passenger        172    70   3405
5          BMW    535i  30.0        Driver only        208    69   3640
6        Buick Century  15.7        Driver only        110    69   2880
```

```
str(CarsA)
```

```
'data.frame':   20 obs. of  7 variables:
 $ Manufacturer: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model       : Factor w/ 20 levels "100","535i","90",..: 13 14 3 1 2 8 15 19 18 12 ...
 $ Price       : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ AirBags     : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ Horsepower  : int  140 200 172 172 208 110 170 180 170 200 ...
 $ Width       : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Weight      : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Solution (2c, 2d)

- Save **CarsA** locally as csv

```
write.csv2(CarsA, file="myCarsA.csv", row.names=FALSE)
```

- Reload and inspect the previously saved file

```
myCarsA <- read.csv2("myCarsA.csv")
str(myCarsA)
```

```
'data.frame':   20 obs. of  7 variables:
 $ Manufacturer: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model       : Factor w/ 20 levels "100","535i","90",..: 13 14 3 1 2 8 15 19 18 12 ...
 $ Price       : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ AirBags     : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ Horsepower  : int  140 200 172 172 208 110 170 180 170 200 ...
 $ Width       : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Weight      : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Tasks / Exercises (3)

- 3a) Load the file **CarsB.txt** from *http://osrp01/estpr/* with **read.table()**, and save it to an object **CarsB**. What are the correct parameter settings?
- 3b) Look at the object, and apply **str()** to this object. Are all variables correctly represented?
- 3c) Save the object locally on your computer as CSV file.
- 3d) Reload the previously saved file. Do you get the correct data structure?

Solution (3a, 3b)

- Read *CarsB.txt* into an object **CarsB** and inspect the result

```
url <- "http://osrp01/estpr/CarsB.txt"
CarsB <- read.table(url, sep="\t",dec=",", header=TRUE)
head(CarsB)
```

```
  Manufacturer   Model Price            AirBags Horsepower Width Weight
1        Acura Integra  15.9               None        140    68   2705
2        Acura  Legend  33.9 Driver & Passenger        200    71   3560
3         Audi      90  29.1        Driver only        172    67   3375
4         Audi     100  37.7 Driver & Passenger        172    70   3405
5          BMW    535i  30.0        Driver only        208    69   3640
6        Buick Century  15.7        Driver only        110    69   2880
```

```
str(CarsB)
```

```
'data.frame':   20 obs. of  7 variables:
 $ Manufacturer: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model       : Factor w/ 20 levels "100","535i","90",..: 13 14 3 1 2 8 15 19 18 12 ...
 $ Price       : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ AirBags     : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ Horsepower  : int  140 200 172 172 208 110 170 180 170 200 ...
 $ Width       : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Weight      : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Solution (3c, 3d)

- Save **CarsB** locally as csv

```
write.csv2(CarsB, file="myCarsB.csv", row.names=FALSE)
```

- Reload and inspect the previously saved file

```
myCarsB <- read.csv2("myCarsB.csv")
str(myCarsB)
```

```
'data.frame':   20 obs. of  7 variables:
 $ Manufacturer: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model       : Factor w/ 20 levels "100","535i","90",..: 13 14 3 1 2 8 15 19 18 12 ...
 $ Price       : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ AirBags     : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ Horsepower  : int  140 200 172 172 208 110 170 180 170 200 ...
 $ Width       : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Weight      : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Tasks / Exercises (4)

- 4a) Load the file **CarsC.txt** from *http://osrp01/estpr/* with **read.table()**, and save it to an object **CarsC**. What are the correct parameter settings?
- 4b) Look at the object, and apply **str()** to this object. Are all variables correctly represented?
- 4c) Save the object locally on your computer as CSV file.
- 4d) Reload the previously saved file. Do you get the correct data structure?

Solution (4a, 4b)

- Read *CarsC.txt* into an object **CarsC** and inspect the result

```
url <- "http://osrp01/estpr/CarsC.txt"
CarsC <- read.table(url, sep=";", dec=".", header=FALSE)
head(CarsC)
```

```
     V1      V2   V3                V4  V5 V6   V7
1 Acura Integra 15.9              None 140 68 2705
2 Acura  Legend 33.9 Driver & Passenger 200 71 3560
3  Audi      90 29.1        Driver only 172 67 3375
4  Audi     100 37.7 Driver & Passenger 172 70 3405
5   BMW    535i 30.0        Driver only 208 69 3640
6 Buick Century 15.7        Driver only 110 69 2880
```

```
str(CarsC)
```

```
'data.frame':   20 obs. of  7 variables:
 $ V1: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ V2: Factor w/ 20 levels "100","535i","90",..: 13 14 3 1 2 8 15 19 18 12 ...
 $ V3: num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ V4: Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ V5: int  140 200 172 172 208 110 170 180 170 200 ...
 $ V6: int  68 71 67 70 69 69 74 78 73 73 ...
 $ V7: int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Solution (4c, 4d)

- Save **CarsC** locally as csv

```
write.csv2(CarsC, file="myCarsC.csv", row.names=FALSE)
```

- Reload and inspect the previously saved file

```
myCarsC <- read.csv2("myCarsC.csv")
str(myCarsC)
```

```
'data.frame':   20 obs. of  7 variables:
 $ V1: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ V2: Factor w/ 20 levels "100","535i","90",..: 13 14 3 1 2 8 15 19 18 12 ...
 $ V3: num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ V4: Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ V5: int  140 200 172 172 208 110 170 180 170 200 ...
 $ V6: int  68 71 67 70 69 69 74 78 73 73 ...
 $ V7: int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Tasks / Exercises (5)

- 5a) Load the file **CarsNA.csv** from *http://osrp01/estpr/* with **read.table()**, and save it to an object **CarsNA**. What are the correct parameter settings? *Hint:* The object contains missings, coded by "-".
- 5b) Look at the object, and apply **str()** to this object. Are all variables correctly represented?
- 5c) Save the object locally on your computer as CSV file.
- 5d) Reload the previously saved file. Do you get the correct data structure?

Solution (5a, 5b)

- Read *CarsNA.csv* into an object **CarsNA** and inspect the result

```
url <- "http://osrp01/estpr/CarsNA.csv"
CarsNA <- read.table(url, sep=";", dec=",",header=TRUE, na.strings="-")
head(CarsNA)
```

```
  Manufacturer   Model Price            AirBags Horsepower Width Weight
1        Acura Integra  15.9               None        140    68   2705
2        Acura  Legend  33.9               <NA>        200    71   3560
3         Audi      90  29.1        Driver only        172    67   3375
4         Audi     100  37.7 Driver & Passenger        172    70   3405
5          BMW    535i  30.0        Driver only        208    69   3640
6        Buick Century  15.7        Driver only        110    69   2880
```

```
str(CarsNA)
```

```
'data.frame':   20 obs. of  7 variables:
 $ Manufacturer: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 NA 4 4 5 ...
 $ Model       : Factor w/ 19 levels "100","535i","90",..: 12 13 3 1 2 8 14 18 17 NA ...
 $ Price       : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ AirBags     : Factor w/ 3 levels "Driver & Passenger",..: 3 NA 2 1 2 2 2 2 2 ...
 $ Horsepower  : int  140 200 172 172 208 110 170 180 170 200 ...
 $ Width       : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Weight      : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Solution (5c, 5d)

- Save **CarsC** locally as csv

```
write.csv2(CarsNA,file="myCarsNA.csv",row.names = FALSE)
```

- Reload and inspect the previously saved file

```
myCarsNA <- read.csv2("myCarsNA.csv")
str(myCarsNA)
```

```
'data.frame':   20 obs. of  7 variables:
 $ Manufacturer: Factor w/ 7 levels "Acura","Audi",..: 1 1 2 2 3 4 NA 4 4 5 ...
 $ Model       : Factor w/ 19 levels "100","535i","90",..: 12 13 3 1 2 8 14 18 17 NA ...
 $ Price       : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ AirBags     : Factor w/ 3 levels "Driver & Passenger",..: 3 NA 2 1 2 2 2 2 2 2 ...
 $ Horsepower  : int  140 200 172 172 208 110 170 180 170 200 ...
 $ Width       : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Weight      : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
```

Exercises: Graphics

Alexander Kowarik, Bernhard Meindl

Tasks / Exercises (1)

- 1a) Apply the function **plot()** on different columns of the data set **Cars93**.

```
data(Cars93, package = "MASS")
colnames(Cars93)
```

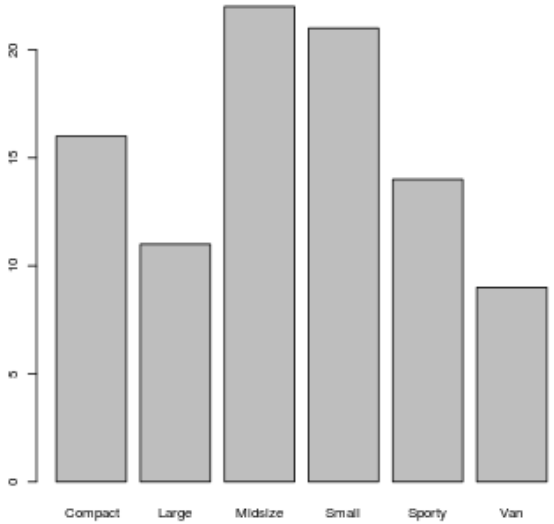
```
 [1] "Manufacturer"       "Model"              "Type"
 [4] "Min.Price"          "Price"              "Max.Price"
 [7] "MPG.city"           "MPG.highway"        "AirBags"
[10] "DriveTrain"         "Cylinders"          "EngineSize"
[13] "Horsepower"         "RPM"                "Rev.per.mile"
[16] "Man.trans.avail"    "Fuel.tank.capacity" "Passengers"
[19] "Length"             "Wheelbase"          "Width"
[22] "Turn.circle"        "Rear.seat.room"     "Luggage.room"
[25] "Weight"             "Origin"             "Make"
```

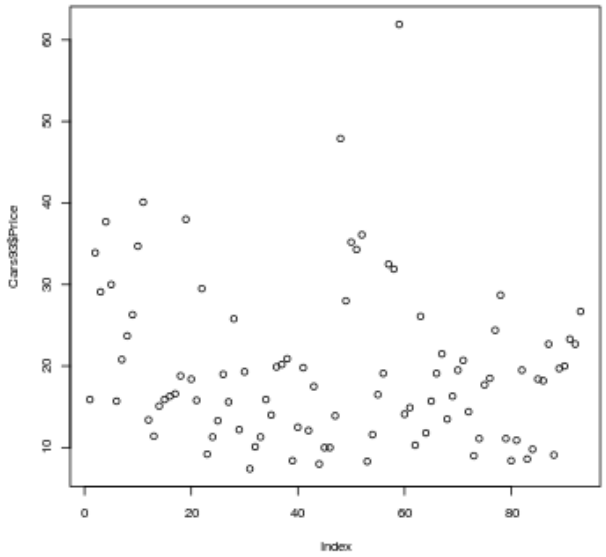
- 1b) Create a histogram of a numerical variable.

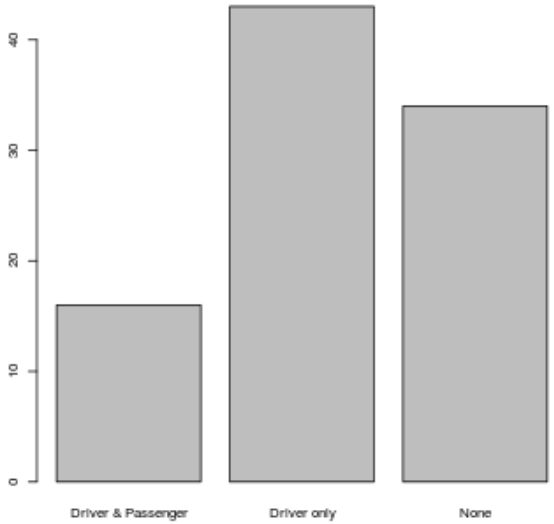
Solution (1a)

- Plot multiple variables

```
data(Cars93, package="MASS")
par(mfrow = c(1, 3))
plot(Cars93$Type)
plot(Cars93$Price)
plot(Cars93$AirBags)
```



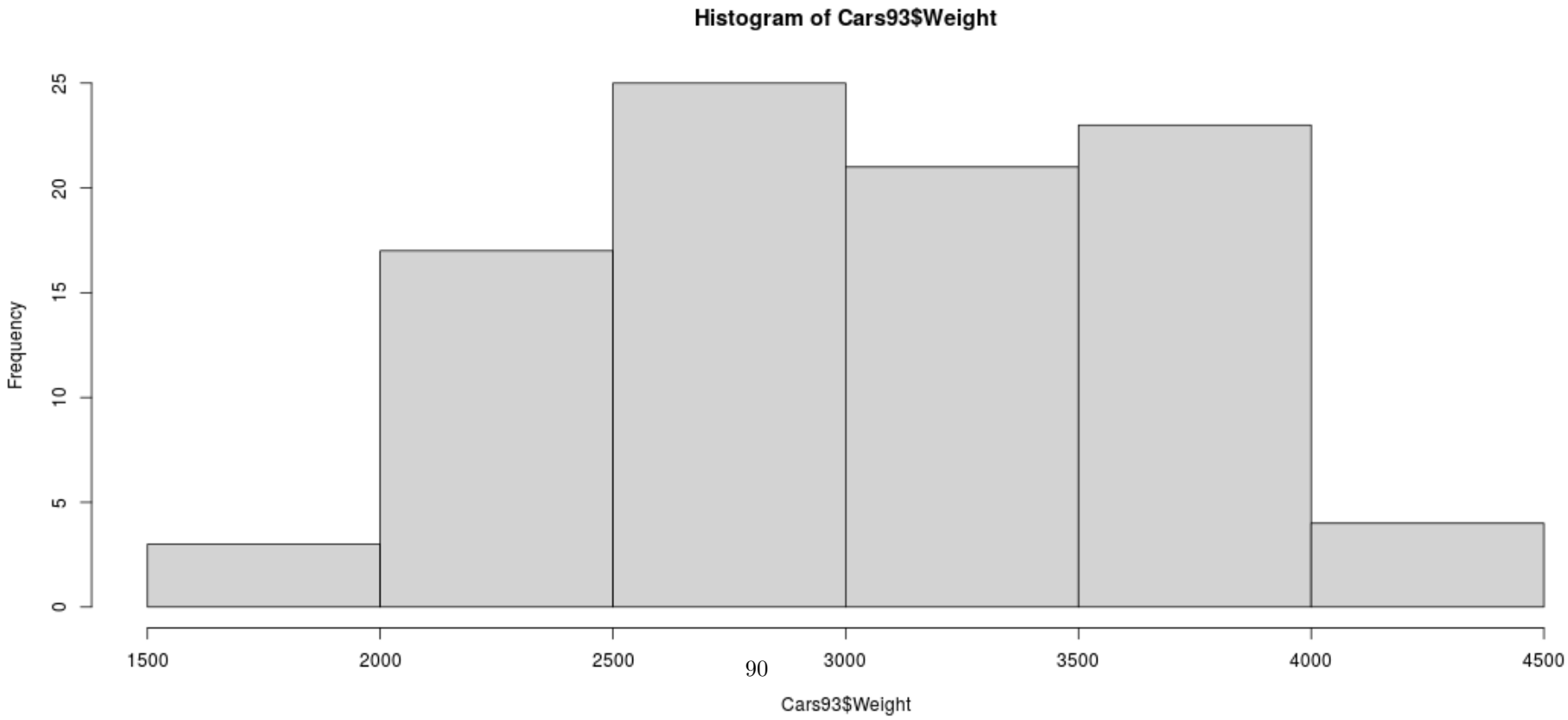




Solution (1b)

- Create a Histogram

```
hist(Cars93$Weight, col="lightgrey")
```



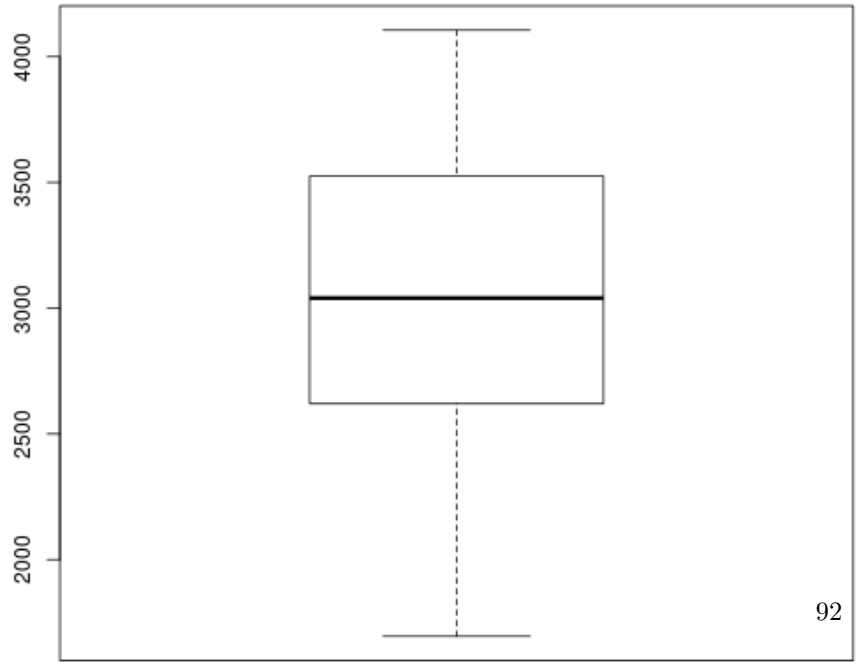
Tasks / Exercises (2)

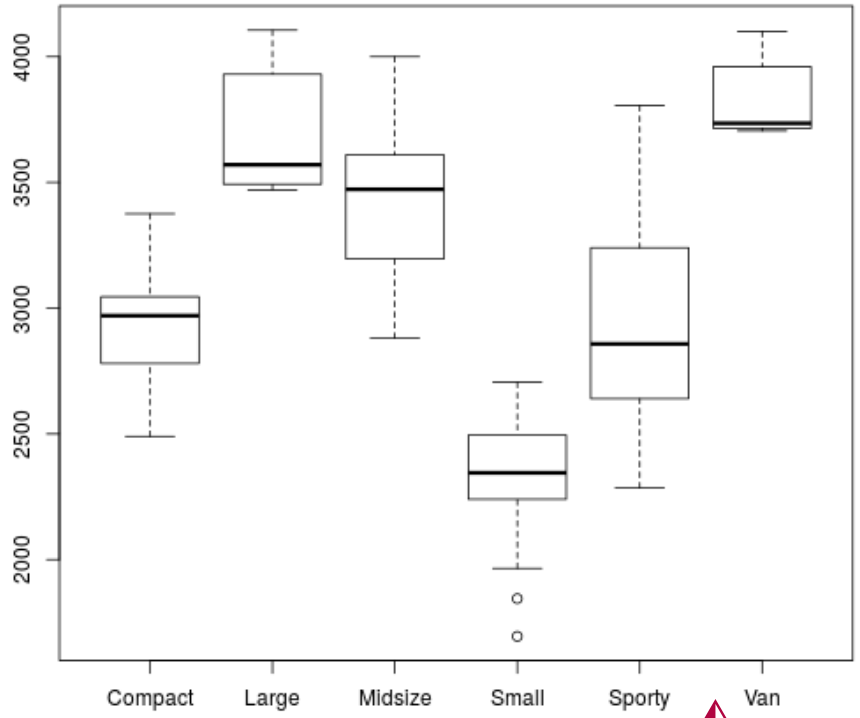
- 2a) Create a boxplot of the metric variables **Weight**.
- 2b) Create a boxplot of the variable **Weight** grouped according to the Variable **Type**

Solution (2a, 2b)

- Create boxplots

```
data(Cars93)
par(mfrow = c(1, 2))
boxplot(Cars93$Weight)
boxplot(Cars93$Weight~Cars93$Type)
```





Exercises: graphics package

Alexander Kowarik, Bernhard Meindl

data

- Use the record *mtcars*

```
data(mtcars)
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- The following exercises should be solved with traditional R graphics (base **graphics** package). Have fun!

Tasks / Exercises (1)

- 1a) Load the data set **mtcars** and familiarize yourself with the variables in the data set (**?mtcars**)
- 1b) encode categorical variables into factors (class **factor**)

Solution (1a)

```
data(mtcars) #? mtcars
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Solution (1b)

- Recoding categorical variables to factors

```
w <- which(colnames(mtcars) %in% c("cyl","vs","am","gear","carb"))
mtcars[,w] <- lapply(mtcars[,w], factor)
str(mtcars)
```

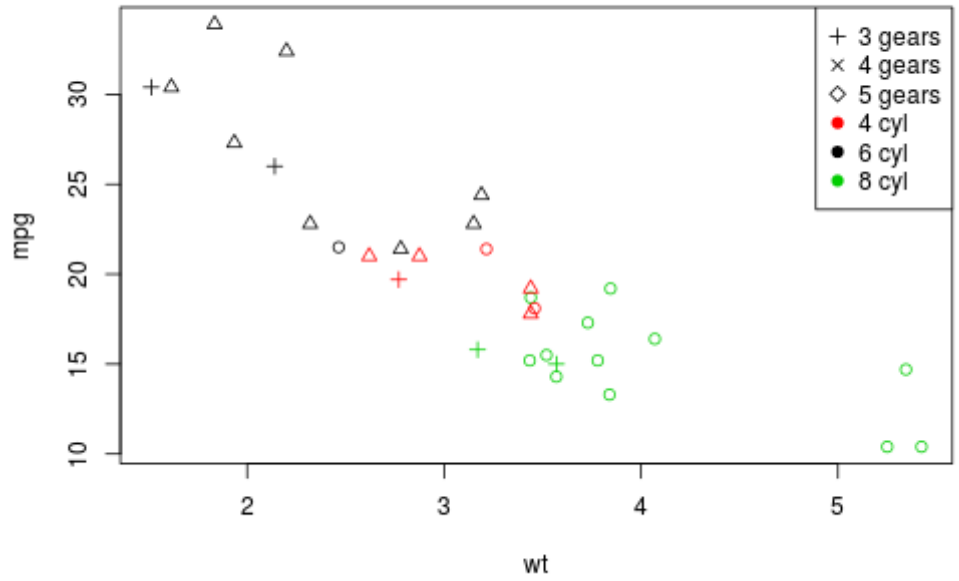
```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 1 2 2 2 ...
 $ am  : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
 $ gear: Factor w/ 3 levels "3","4","5": 2 2 2 1 1 1 1 2 2 2 ...
 $ carb: Factor w/ 6 levels "1","2","3","4",..: 4 4 1 1 2 1 4 2 2 4 ...
```

Tasks / Exercises (2)

- 2a) Create a scatter plot of the variable *mpg* depending on the weight (*wt*).
- 2b) Color the points corresponding to the variable a *cyl*.
- 2c) Use for the different number of gears (*gear*) different symbols.
- 2d) Add a legend displaying the different number of gears (and possibly cylinders).

Solution (2a, 2b, 2c, 2d)

```
plot(mpg ~ wt, col=cyl, pch=as.numeric(gear), data=mtcars)
legend("topright",
  legend=c(paste(levels(mtcars$gear), "gears"), paste(levels(mtcars$cyl), "cyl")),
  pch=c(as.numeric(levels(mtcars$gear)), rep(19,length(levels(mtcars$cyl)))),
  col=c(rep("black", length(levels(mtcars$gear))), unique(as.numeric(mtcars$cyl))))
```



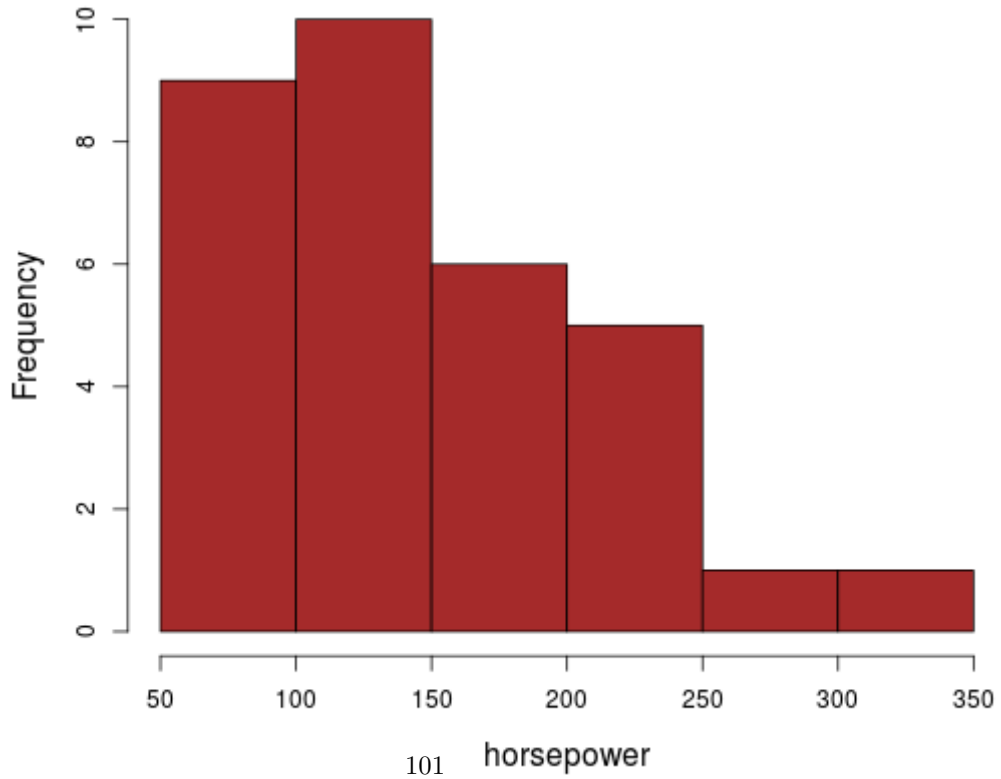
Tasks / Exercises (3)

- 3a) Use the function **hist()** a histogram of the Variable horsepower (*hp*) to create. Colorize the histogram and label the histogram. Enlarge the axis label.
- 3b) Use the function **density()** to add a density estimate to the histogramm.

Solution (3a)

- Plot a histogram

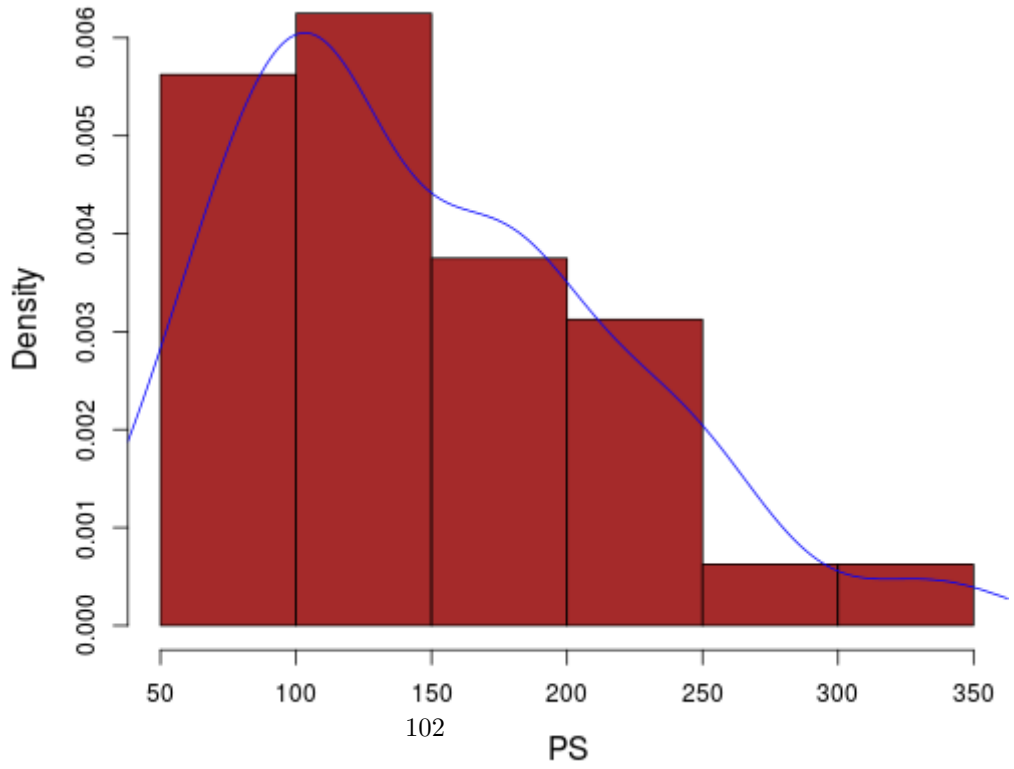
```
par(mar=c(4.5,4.5,0.1,0.1), cex.lab=1.3)
hist(mtcars$hp, col="brown", main="", xlab="horsepower")
```



Solution (3b)

- add a line showing density estimator

```
par(mar=c(4.5,4.5,0.1,0.1), cex.lab=1.3)
hist(mtcars$hp, col="brown", main="", xlab="PS", prob=TRUE)
lines(density(mtcars$hp),col="blue",type="l")
```



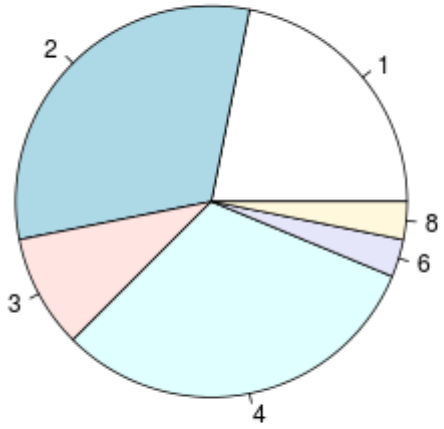
Tasks / Exercises (4)

- 4a) Visualise the amount of carburetors (*carb*) in a piechart (**pie()**) and
- 4b) a barchart (**barplot()**)
- 4c) Where do you assess the the distribution of counts better?

Solution (4a)

- Create a piechart

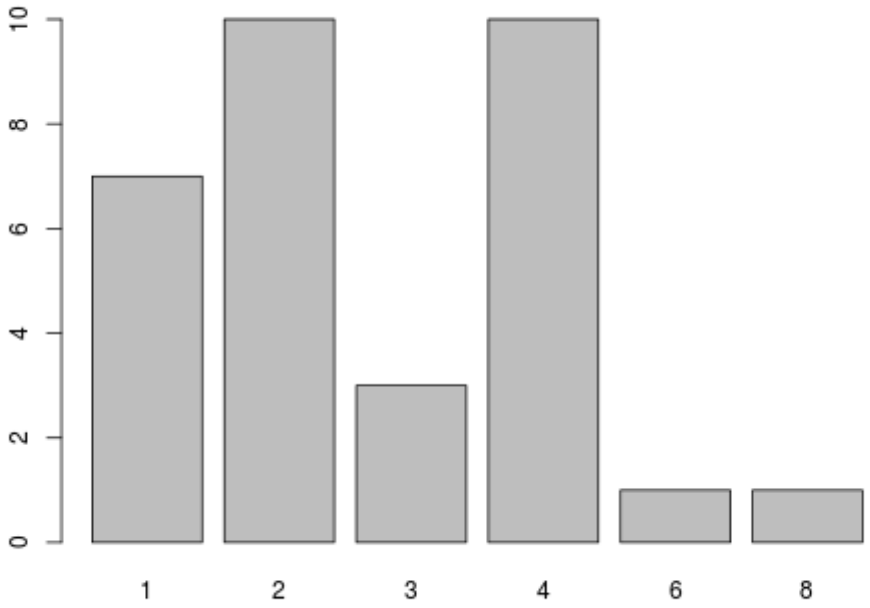
```
pie(table(mtcars$carb))
```



Solution (4b)

- Create a barchart

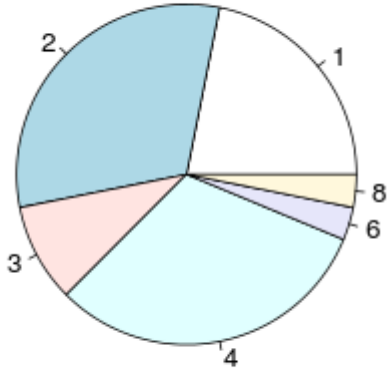
```
barplot(table(mtcars$carb))
```

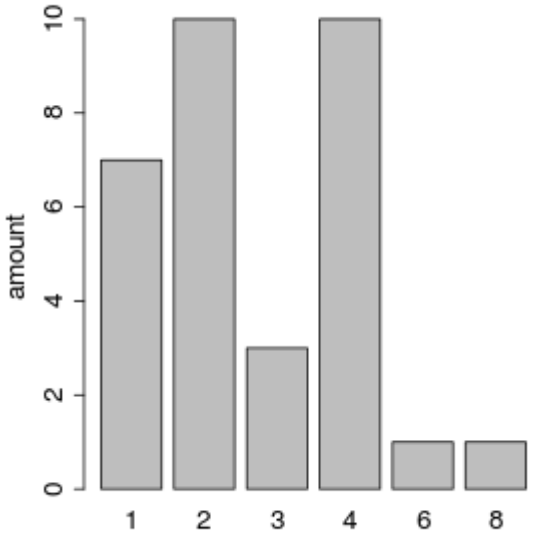


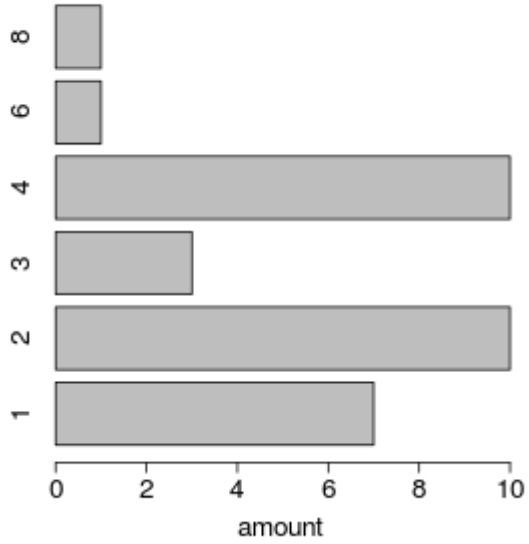
Solution (4c)

- Where can we assess the counts best?

```
par(mfrow=c(1,3))
par(cex.axis=1.7, cex.lab=1.7)
pie(table(mtcars$carb), cex=1.7)
barplot(table(mtcars$carb), ylab="amount")
barplot(table(mtcars$carb), horiz=TRUE, xlab="amount")
```







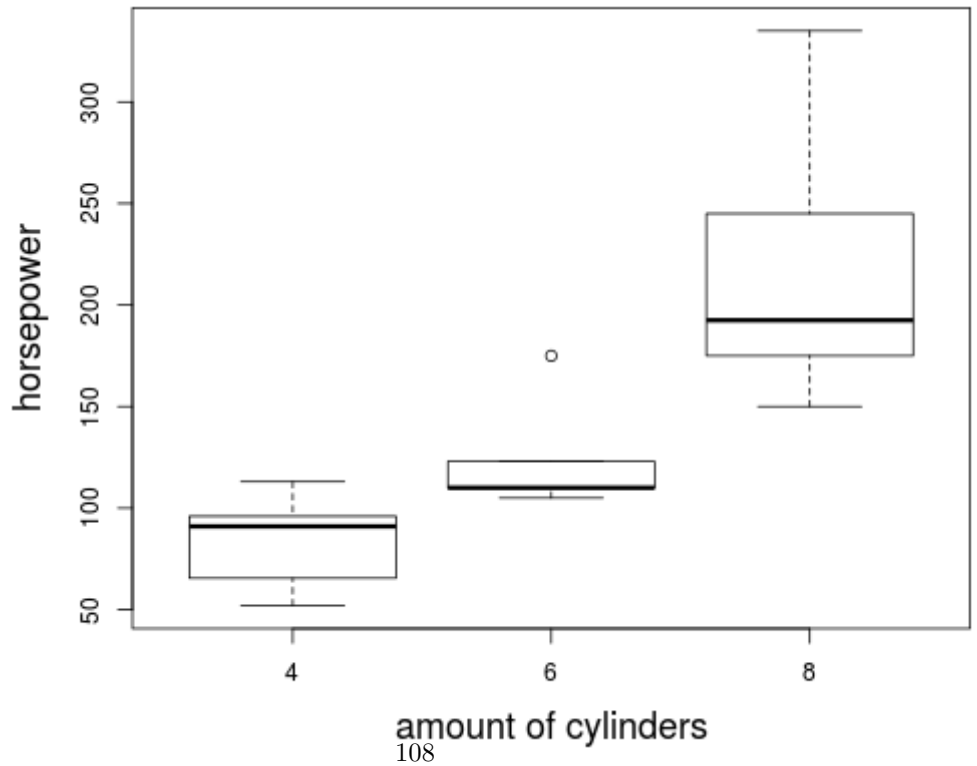
Tasks / Exercises (5)

- 5a) Create separate boxplots (**boxplot()**) of horsepower (*hp*) for different number of cylinders (*cyl*). Label the axes.

Solution (5a)

- Boxplot of *hp* by *cyl*

```
par(mar=c(5,5,1,1))
boxplot(mtcars$hp ~ mtcars$cyl, xlab="amount of cylinders", ylab="horsepower", cex.lab=1.5)
```



Exercises: ggplot2

Alexander Kowarik, Bernhard Meindl

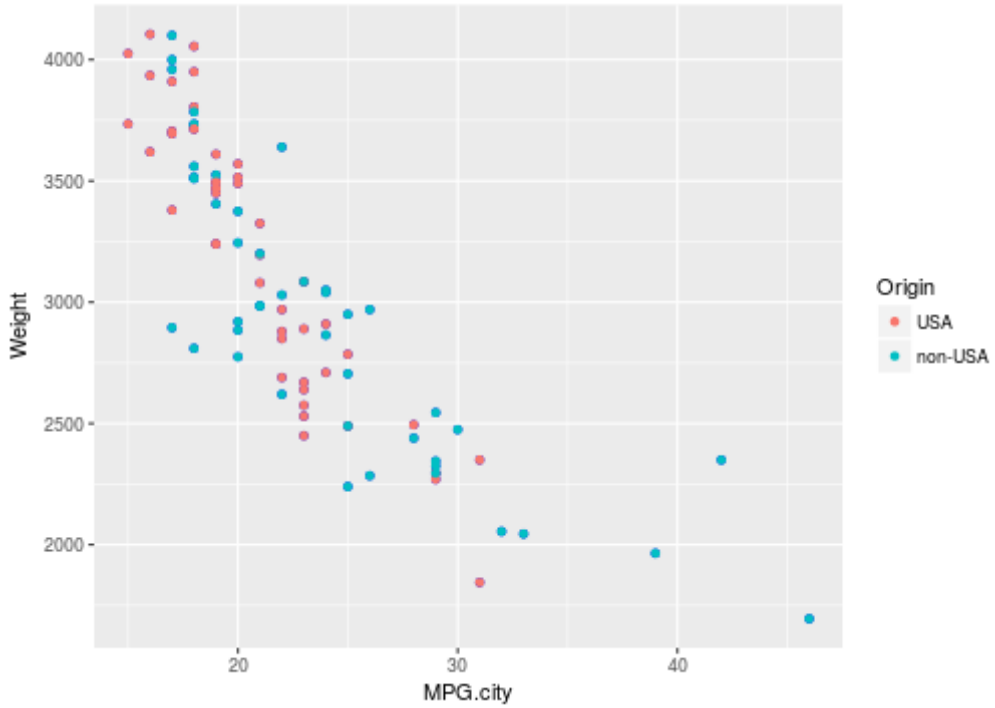
Tasks / Exercises (1)

In the following examples, some ggplot2 functions are to be tested and some parameters to be adjusted. Use the data set *Cars93*. More detailed information on the data you obtained with the open the help (**?Cars93**).

- 1a) Create a scatter plot of the variable *MPG.city* dependend of *Weight*.
- 1b) Color the points in blue.
- 1c) Color the points according to variable *Origin*.

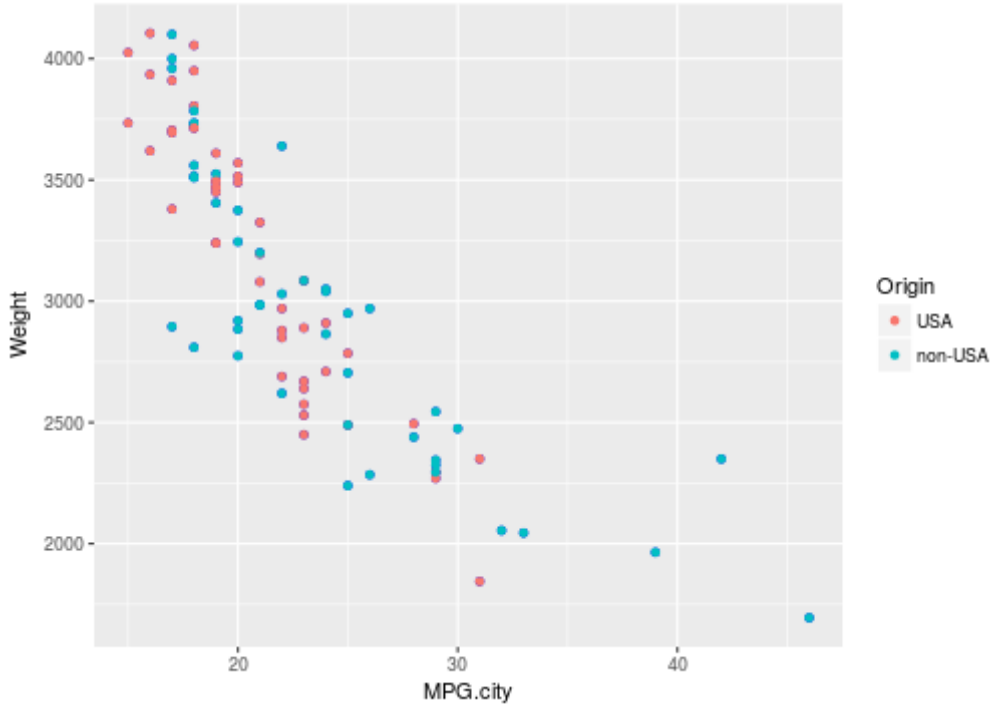
Solution (1a, 1b)

```
require(ggplot2)
data(Cars93, package="MASS")
g <- ggplot(Cars93, aes(x=MPG.city, y=Weight))
g <- g + geom_point(color="blue")
g + geom_point(aes(color=Origin))
```



Solution (1c)

```
g + geom_point(aes(color=Origin))
```

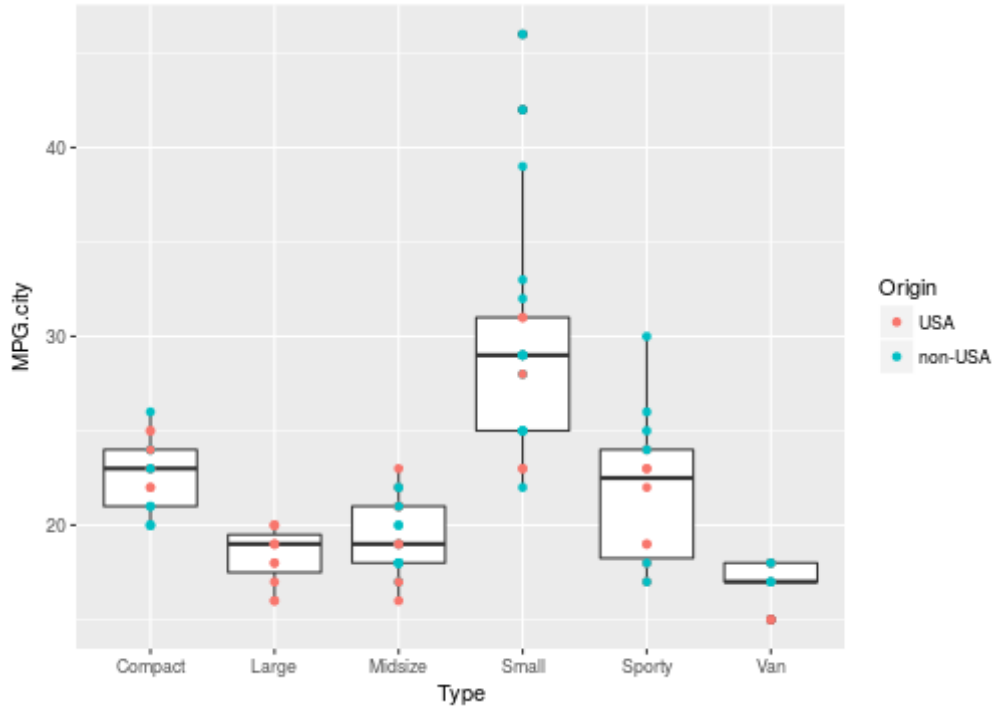


Tasks / Exercises (2)

- 2a) Create a boxplot of *MPG.city* by *Type*.
- 2b) Add points with colouring according to *Origin*.
- 2c) Make a scatter plot of variable *Weight* dependend on *Horsepower*.
- 2d) Add a regression line.
- 2e) Apply it for all groups of variable *Origin*

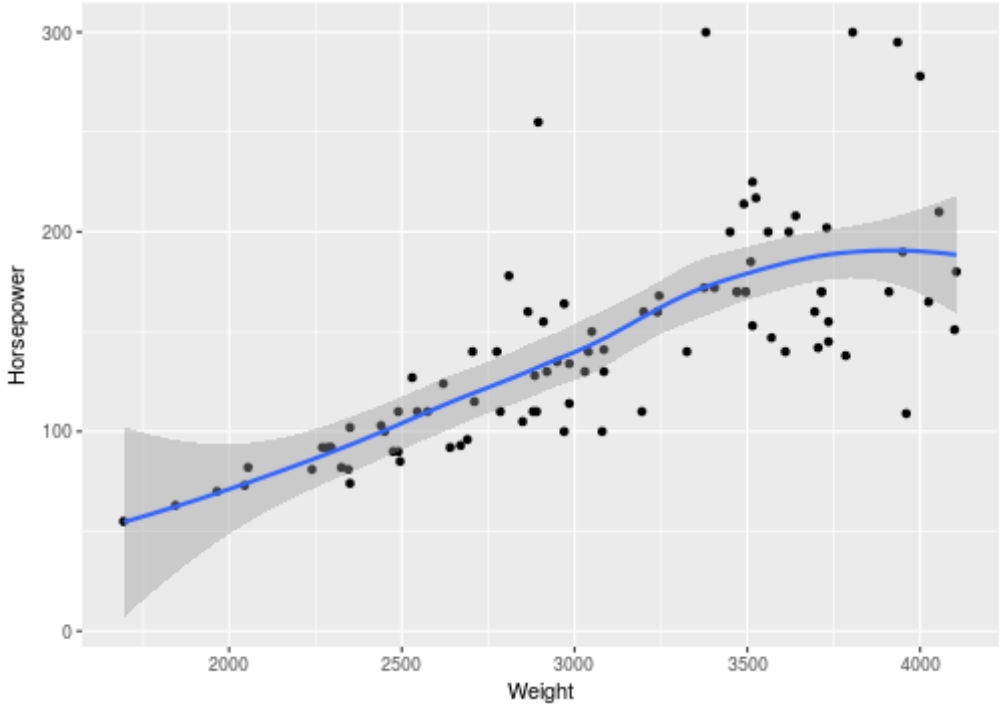
Solution 2 (2a, 2b)

```
g <- ggplot(Cars93,aes(x=Type,y=MPG.city))
g <- g + geom_boxplot()
g + geom_point(aes(color=Origin))
```



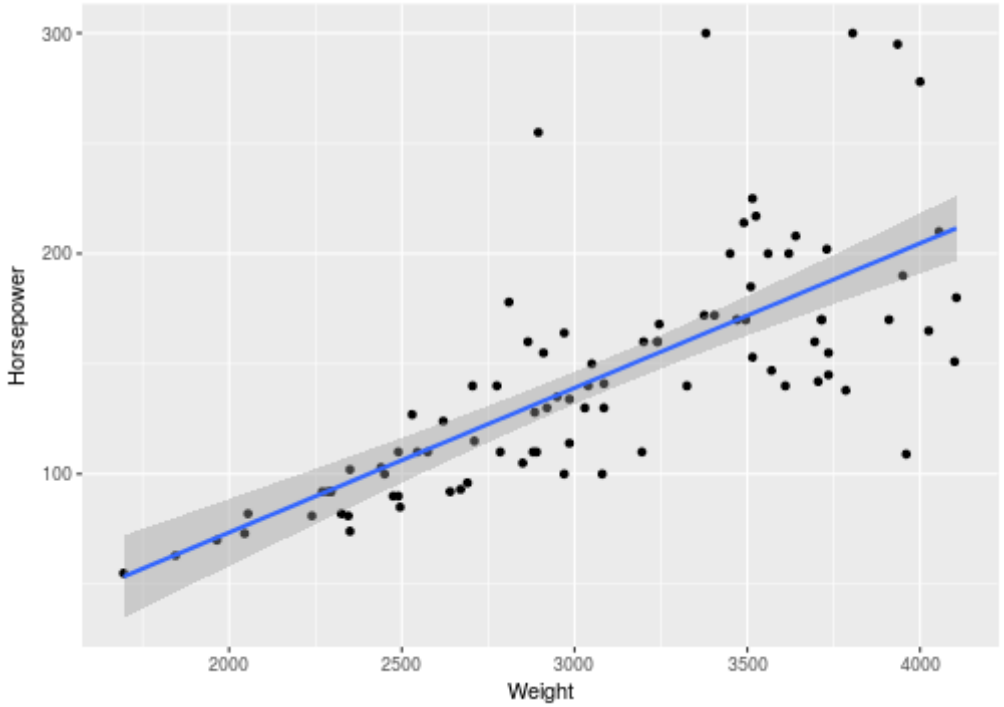
Solution 2 (2c, 2d)

```
g <- ggplot(Cars93,aes(x=Weight,y=Horsepower))
g <- g + geom_point()
g + geom_smooth()
```



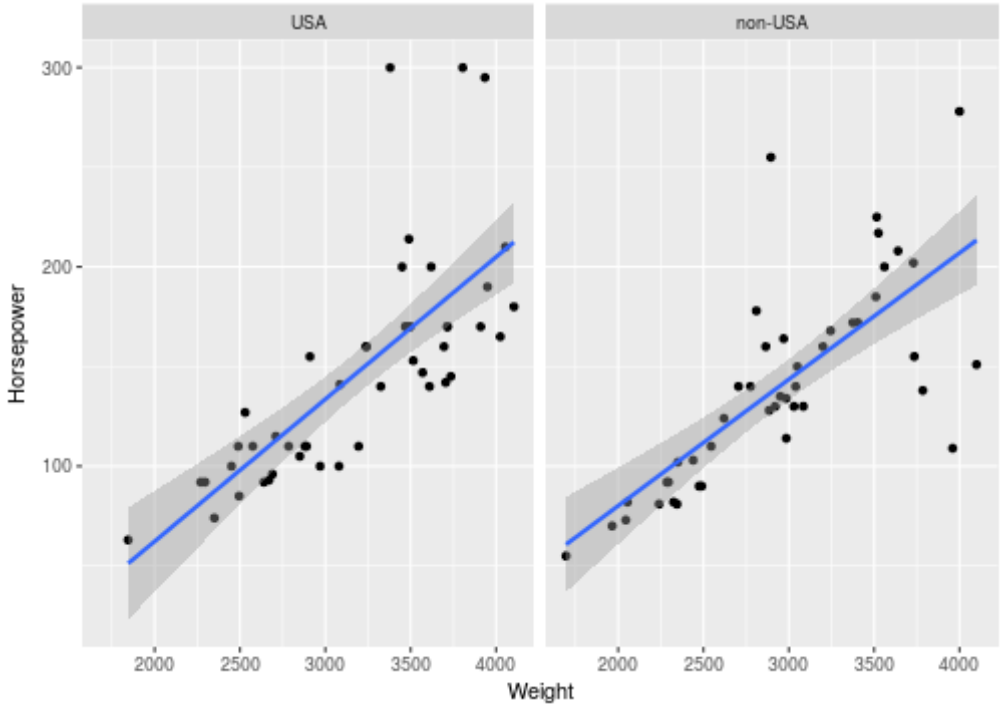
Solution 2 (2d)

```
g <- g + geom_smooth(method="lm") # changing default parameter
g
```



Solution 2 (2e)

```
g + facet_wrap(~ Origin)
```



Exercises: Dynamic and automated reports with R

Alexander Kowarik, Bernhard Meindl

Tasks / Exercises (1)

- 1a) Create a new **rmarkdown** file in **RStudio**
- 1b) Adapt the file and create HTML output, open it in a browser
- 1c) Experiment with automated reporting and include for example using markdown
  - headlines
  - tables
  - text (bold, italic, ..)
  - a graphic

You can use any data (e.g. *Cars93*) or create yourself a data set.

Solution (1a)

let's use the *Cars93* data and tabulate

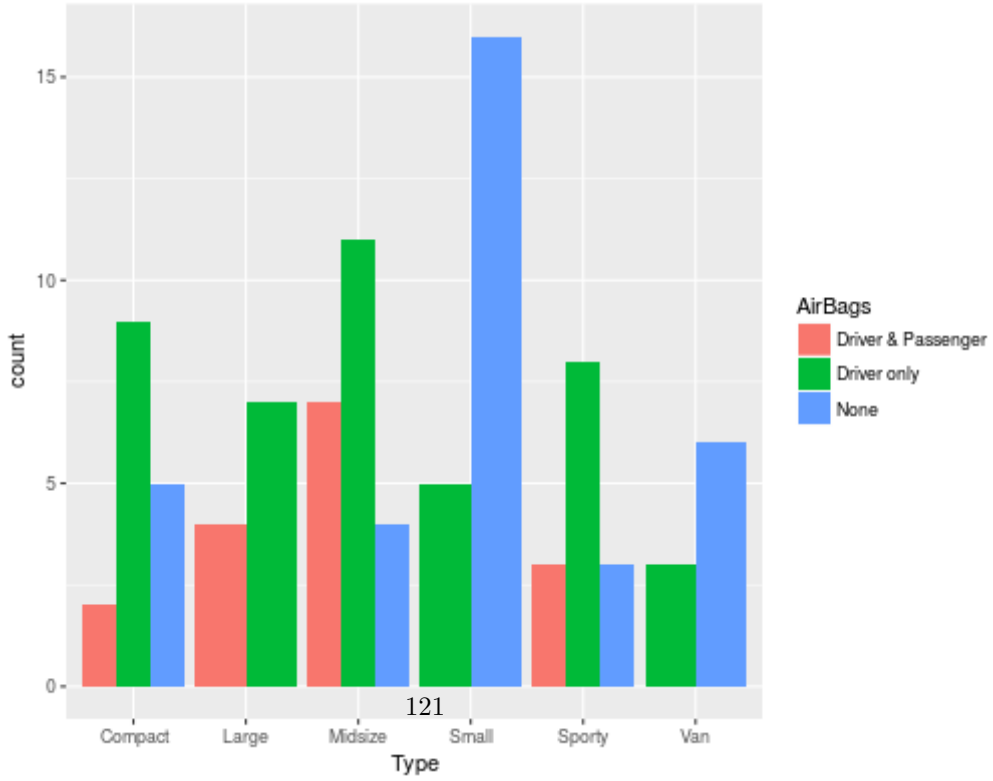
```
data(Cars93, package="MASS")
library(dplyr)
Cars93 %>% select(Type, AirBags) %>% group_by(Type, AirBags) %>% summarise(N=n())
```

```
Source: local data frame [15 x 3]
Groups: Type [?]

      Type            AirBags     N
    <fctr>             <fctr> <int>
1  Compact Driver & Passenger     2
2  Compact        Driver only     9
3  Compact               None     5
4    Large Driver & Passenger     4
5    Large        Driver only     7
6  Midsize Driver & Passenger     7
7  Midsize        Driver only    11
8  Midsize               None     4
9    Small        Driver only     5
10   Small               None    16
11  Sporty Driver & Passenger     3
12  Sporty        Driver only     8
13  Sporty               None     3
14     Van        Driver only     3
15     Van               None     6
```

Solution (1b)

do the plot

```
require(ggplot2); ggplot(Cars93, aes(Type, fill=AirBags)) + geom_bar(position="dodge")
```



Exercises: data manipulation package dplyr

Alexander Kowarik, Bernhard Meindl

Data

- Use the data set *mtcars*

```
data(mtcars)
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

- Use the Package **dplyr** for the following exercises. Have Fun!

Tasks / Exercises (1)

- 1a) Load the data set **mtcars** and familiarize yourself with the variables in the data set (**?mtcars**)
- 1b) Convert the object to a *local* to data.frame
- 1c) Select all observations with less than 100 horsepower. How many are they?
- 1d) Remove variables *gear* and *carb*
- 1e) Sort the data in descending order by variable *cyl* and within *cyl* in ascending order by horsepower.
- 1f) Rename variable *hp* into *power*

Solution (1a)

- Download and view the Data

```
library(dplyr)
data(mtcars)
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Solution (1b)

- Create a *local* data frames

```
mtcars <- tbl_df(mtcars)
head(mtcars, 15)
```

```
# A tibble: 15 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
11  17.8     6 167.6   123  3.92 3.440 18.90     1     0     4     4
12  16.4     8 275.8   180  3.07 4.070 17.40     0     0     3     3
13  17.3     8 275.8   180  3.07 3.730 17.60     0     0     3     3
14  15.2     8 275.8   180  3.07 3.780 18.00     0     0     3     3
15  10.4     8 472.0   205  2.93 5.250 17.98     0     0     3     4
```

Solution (1c)

Filter out cars with less than 100hp -

```
lowhp <- filter(mtcars, hp < 100)
lowhp
```

```
# A tibble: 9 × 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
2  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
3  22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
4  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
5  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
6  33.9     4  71.1    65  4.22 1.835 19.90     1     1     4     1
7  21.5     4 120.1    97  3.70 2.465 20.01     1     0     3     1
8  27.3     4  79.0    66  4.08 1.935 18.90     1     1     4     1
9  26.0     4 120.3    91  4.43 2.140 16.70     0     1     5     2
```

Solution (1d)

- Selection of variables

```
dat1 <- select(mtcars, mpg:am); head(dat1, 3)
```

```
# A tibble: 3 × 9
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  21.0     6   160   110  3.90 2.620 16.46     0     1
2  21.0     6   160   110  3.90 2.875 17.02     0     1
3  22.8     4   108    93  3.85 2.320 18.61     1     1
```

```
dat2 <- select(mtcars, -gear, -carb); head(dat2, 3)
```

```
# A tibble: 3 × 9
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  21.0     6   160   110  3.90 2.620 16.46     0     1
2  21.0     6   160   110  3.90 2.875 17.02     0     1
3  22.8     4   108    93  3.85 2.320 18.61     1     1
```

```
identical(dat1, dat2)
```

```
[1] TRUE
```

Solution (1e)

- Assignment

```
mtcars <- arrange(mtcars, desc(cyl), hp)
head(mtcars, 15)
```

```
# A tibble: 15 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   15.5     8 318.0   150  2.76 3.520 16.87     0     0     3     2
2   15.2     8 304.0   150  3.15 3.435 17.30     0     0     3     2
3   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
4   19.2     8 400.0   175  3.08 3.845 17.05     0     0     3     2
5   16.4     8 275.8   180  3.07 4.070 17.40     0     0     3     3
6   17.3     8 275.8   180  3.07 3.730 17.60     0     0     3     3
7   15.2     8 275.8   180  3.07 3.780 18.00     0     0     3     3
8   10.4     8 472.0   205  2.93 5.250 17.98     0     0     3     4
9   10.4     8 460.0   215  3.00 5.424 17.82     0     0     3     4
10  14.7     8 440.0   230  3.23 5.345 17.42     0     0     3     4
11  14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
12  13.3     8 350.0   245  3.73 3.840 15.41     0     0     3     4
13  15.8     8 351.0   264  4.22 3.170 14.50     0     1     5     4
14  15.0     8 301.0   335  3.54 3.570 14.60     0     1     5     8
15  18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
```

Solution (1f)

- Renaming

```
mtcars <- select(mtcars, power=hp, matches("."))
# mtcars <- rename(mtcars, power=hp) # alternative way
head(mtcars, 15)
```

```
# A tibble: 15 × 11
   power   mpg   cyl  disp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1    150  15.5     8 318.0  2.76 3.520 16.87     0     0     3     2
2    150  15.2     8 304.0  3.15 3.435 17.30     0     0     3     2
3    175  18.7     8 360.0  3.15 3.440 17.02     0     0     3     2
4    175  19.2     8 400.0  3.08 3.845 17.05     0     0     3     2
5    180  16.4     8 275.8  3.07 4.070 17.40     0     0     3     3
6    180  17.3     8 275.8  3.07 3.730 17.60     0     0     3     3
7    180  15.2     8 275.8  3.07 3.780 18.00     0     0     3     3
8    205  10.4     8 472.0  2.93 5.250 17.98     0     0     3     4
9    215  10.4     8 460.0  3.00 5.424 17.82     0     0     3     4
10   230  14.7     8 440.0  3.23 5.345 17.42     0     0     3     4
11   245  14.3     8 360.0  3.21 3.570 15.84     0     0     3     4
12   245  13.3     8 350.0  3.73 3.840 15.41     0     0     3     4
13   264  15.8     8 351.0  4.22 3.170 14.50     0     1     5     4
14   335  15.0     8 301.0  3.54 3.570 14.60     0     1     5     8
15   105  18.1     6 225.0  2.76 3.460 20.22     1     0     3     1
```

Tasks / Exercises (2)

- 2a) Compute from variable *mpg* a new variable *litre_km* and then remove the variable *mpg* from the data.
  - Where: litre_km ~ 235.2146 / mpg
- 2b) Create a variable *type* where the values should be *fast* when variable *qsec < 17.5* and *slow* otherwise (**?cut**)
- 2c) Group the data according to the variable *type*
- 2d) Compute for fast and slow cars the following values:
  - the group size
  - the average horse power (variable *ps*)
  - the average consumption of fuel (variable *litre_km*)

Solution (2a)

- Computing a new variable

```
mtcars <- mtcars %>% mutate(litre_km=235.2146/mpg) %>% select(-mpg)
head(mtcars, 15)
```

```
# A tibble: 15 × 11
   power   cyl  disp  drat    wt  qsec    vs    am  gear  carb litre_km
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
1    150     8 318.0  2.76 3.520 16.87     0     0     3     2 15.17514
2    150     8 304.0  3.15 3.435 17.30     0     0     3     2 15.47464
3    175     8 360.0  3.15 3.440 17.02     0     0     3     2 12.57832
4    175     8 400.0  3.08 3.845 17.05     0     0     3     2 12.25076
5    180     8 275.8  3.07 4.070 17.40     0     0     3     3 14.34235
6    180     8 275.8  3.07 3.730 17.60     0     0     3     3 13.59622
7    180     8 275.8  3.07 3.780 18.00     0     0     3     3 15.47464
8    205     8 472.0  2.93 5.250 17.98     0     0     3     4 22.61679
9    215     8 460.0  3.00 5.424 17.82     0     0     3     4 22.61679
10   230     8 440.0  3.23 5.345 17.42     0     0     3     4 16.00099
11   245     8 360.0  3.21 3.570 15.84     0     0     3     4 16.44857
12   245     8 350.0  3.73 3.840 15.41     0     0     3     4 17.68531
13   264     8 351.0  4.22 3.170 14.50     0     1     5     4 14.88700
14   335     8 301.0  3.54 3.570 14.60     0     1     5     8 15.68097
15   105     6 225.0  2.76 3.460 20.22     1     0     3     1 12.99528
```

Solution (2b)

- Computing a new variable

```
mtcars <- mtcars %>% mutate(typ=cut(qsec, breaks=c(-Inf, 17.5, Inf), labels=c("fast","slow")))
head(mtcars,15)
```

```
# A tibble: 15 × 12
   power   cyl  disp  drat    wt  qsec    vs    am  gear  carb litre_km
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
1    150     8 318.0  2.76 3.520 16.87     0     0     3     2 15.17514
2    150     8 304.0  3.15 3.435 17.30     0     0     3     2 15.47464
3    175     8 360.0  3.15 3.440 17.02     0     0     3     2 12.57832
4    175     8 400.0  3.08 3.845 17.05     0     0     3     2 12.25076
5    180     8 275.8  3.07 4.070 17.40     0     0     3     3 14.34235
6    180     8 275.8  3.07 3.730 17.60     0     0     3     3 13.59622
7    180     8 275.8  3.07 3.780 18.00     0     0     3     3 15.47464
8    205     8 472.0  2.93 5.250 17.98     0     0     3     4 22.61679
9    215     8 460.0  3.00 5.424 17.82     0     0     3     4 22.61679
10   230     8 440.0  3.23 5.345 17.42     0     0     3     4 16.00099
11   245     8 360.0  3.21 3.570 15.84     0     0     3     4 16.44857
12   245     8 350.0  3.73 3.840 15.41     0     0     3     4 17.68531
13   264     8 351.0  4.22 3.170 14.50     0     1     5     4 14.88700
14   335     8 301.0  3.54 3.570 14.60     0     1     5     8 15.68097
15   105     6 225.0  2.76 3.460 20.22     1     0     3     1 12.99528
# ... with 1 more variables: typ <fctr>
```

Solution (2c, 2d)

- Aggregating within groups, using the pipe-operator

```
mtcars %>% group_by(typ) %>% summarise(N=n(), mean(power), mean(litre_km))
```

```
# A tibble: 2 × 4
     typ     N `mean(power)` `mean(litre_km)`
  <fctr> <int>         <dbl>            <dbl>
1   fast    15      183.2000         13.44329
2   slow    17      114.4706         12.14776
```

Exercises to Basic Statistics in R

Alexander Kowarik, Bernhard Meindl

Tasks / Exercises (1)

Load the **Cars93** data, and focus on the variables **Weight** (weight of car) and **Origin** ("USA" and "non-USA").

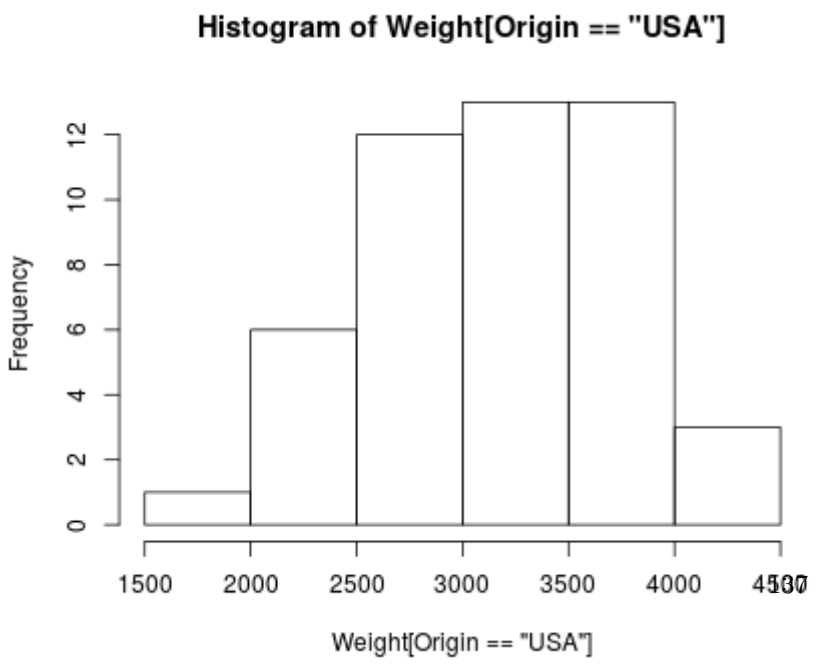
Normal distribution:

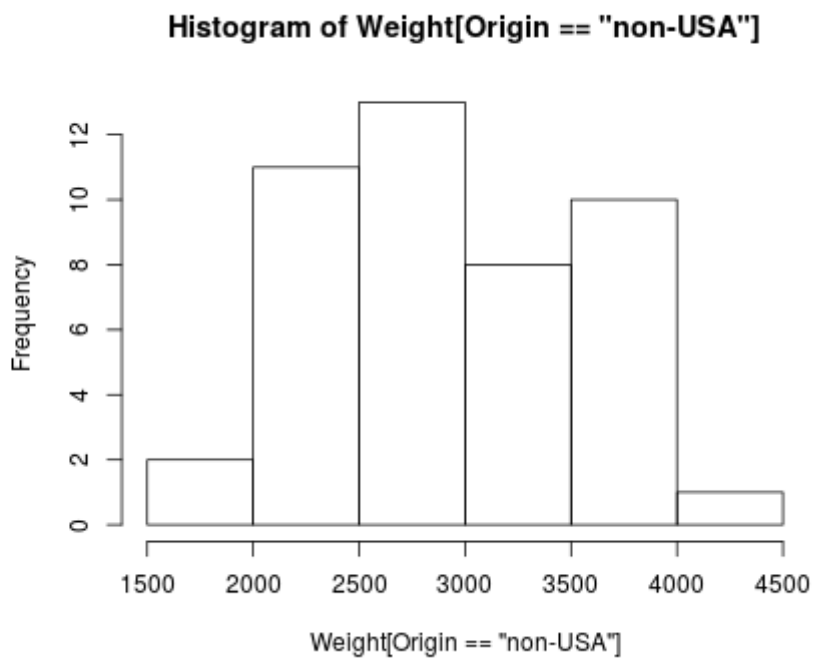
- 1a) Show histograms of **Weight**, separately for each **Origin**
- 1b) Show QQ-plots of **Weight**, separately for each **Origin**. Can you assume normal distribution?
- 1c) Test for normality (Shapiro-Wilk test) of **Weight**, separately for each **Origin**

Solution (1a)

Look at histograms:

```
data(Cars93,package="MASS")
attach(Cars93)
par(mfrow=c(1,2))
hist(Weight[Origin=="USA"]); hist(Weight[Origin=="non-USA"])
```

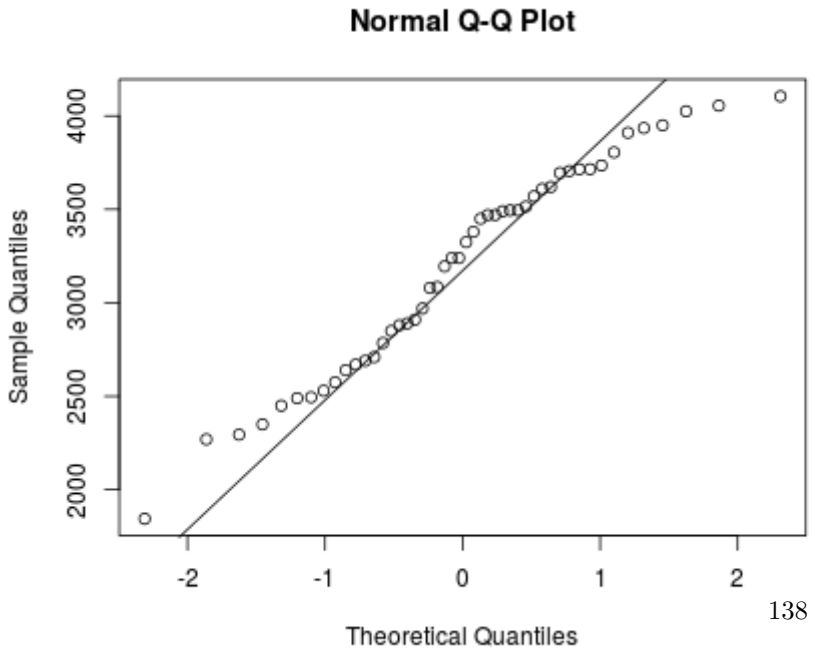


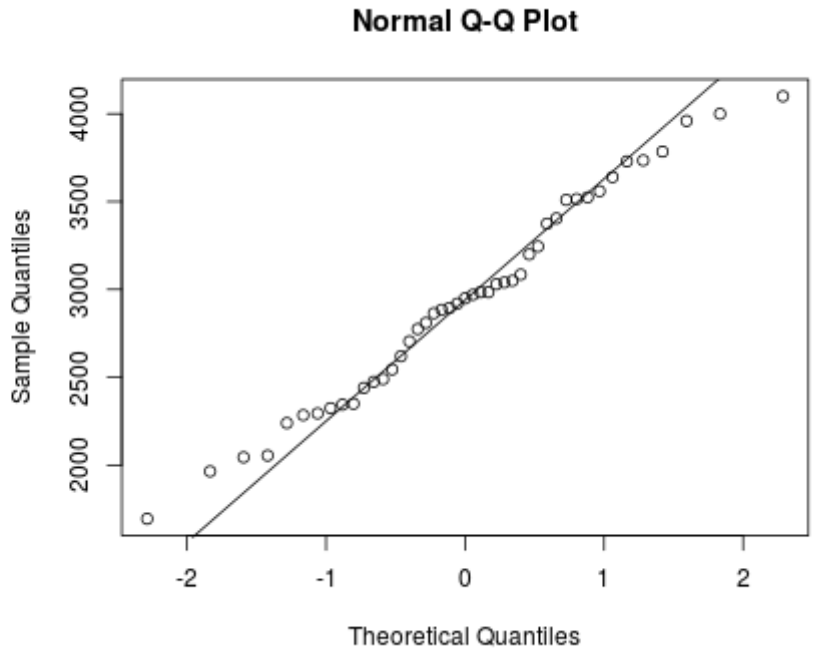


Solution (1b)

Look at QQ-plots:

```
par(mfrow=c(1,2))
qqnorm(Weight[Origin=="USA"]); qqline(Weight[Origin=="USA"])
qqnorm(Weight[Origin=="non-USA"]); qqline(Weight[Origin=="non-USA"])
```





Solution (1c)

Testing normality with Shapiro-Wilk test:

```
shapiro.test(Weight[Origin=="USA"])
```

```
    Shapiro-Wilk normality test

data:  Weight[Origin == "USA"]
W = 0.96082, p-value = 0.1089
```

```
shapiro.test(Weight[Origin=="non-USA"])
```

```
    Shapiro-Wilk normality test

data:  Weight[Origin == "non-USA"]
W = 0.98083, p-value = 0.6534
```

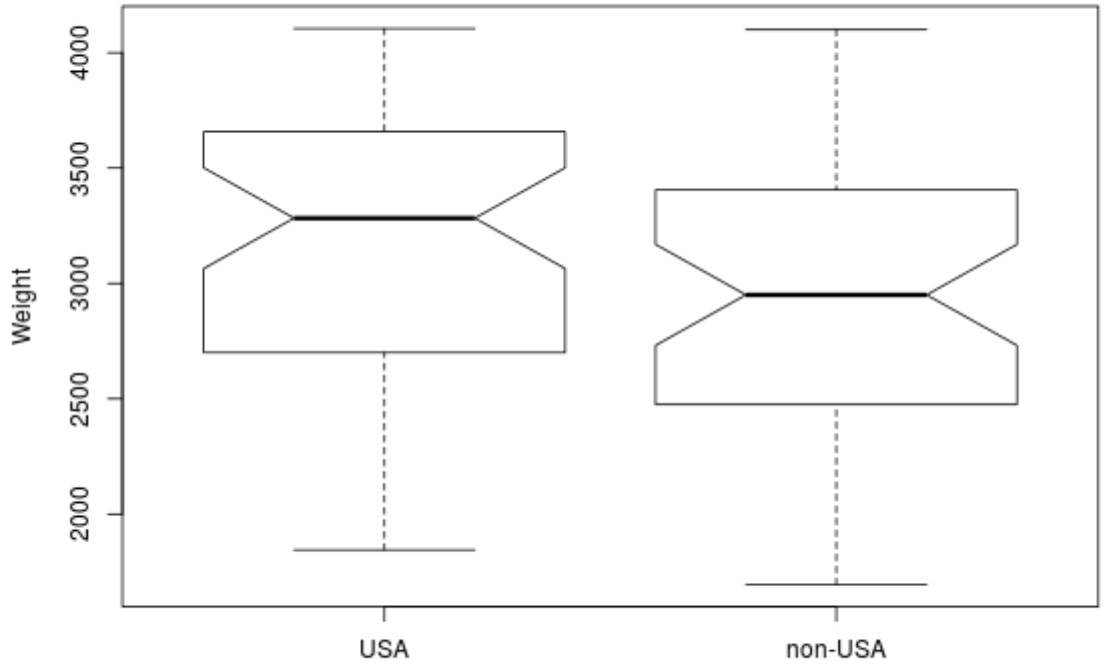
Tasks / Exercises (2)

- 2a) Show parallel boxplots (with notches) of **Weight**, separately for each **Origin**
- 2b) Test for equality of variances of **Weight** for the two different labels of **Origin** using **var.test()**
- 2c) Test for equality of means of **Weight** for the two different labels of **Origin** using **t.test()**. Which parameters do you need to select for the function **t.test()**?

Solution (2a)

- Boxplot comparisons:

```
boxplot(Weight~Origin,ylab="Weight",notch=TRUE)
```



Solution (2b, 2c)

- Tests

```
var.test(Weight~Origin) # F-test for equality of variances
```

```
    F test to compare two variances

data:  Weight by Origin
F = 0.90622, num df = 47, denom df = 44, p-value = 0.7388
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.501495 1.628160
sample estimates:
ratio of variances
         0.9062231
```

```
t.test(Weight~Origin,var.equal=TRUE) # two-sample t-test (equal variances)
```

```
    Two Sample t-test

data:  Weight by Origin
t = 2.105, df = 91, p-value = 0.03805
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  14.25214 491.70620
sample estimates:
    mean in group USA mean in group non-USA
             3195.312              2942.333
```

Tasks / Exercises (3)

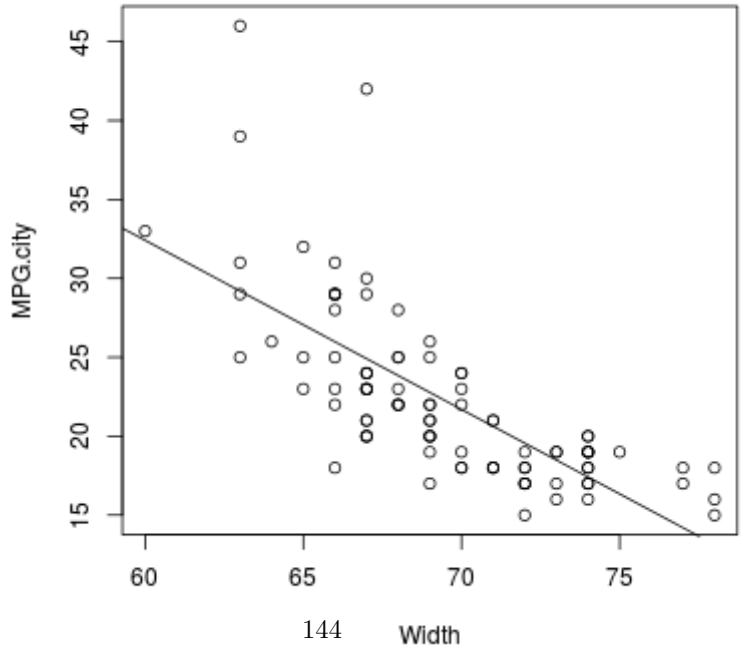
Linear regression:

- 3a) Use **MPG.city** as response variable, and **Width** as explanatory variable. Show them in a plot.
- 3b) Add the LS-regression line to the plot.
- 3c) Show and interpret the inference statistics.
- 3d) Show and interpret the diagnostic plots.

Solution (3a, 3b)

- Plot data, add LS-regression line:

```
plot(MPG.city~Width,data=Cars93)
res <- lm(MPG.city~Width,data=Cars93)
abline(res)
```



Solution (3c)

- LS-regression inference statistics:

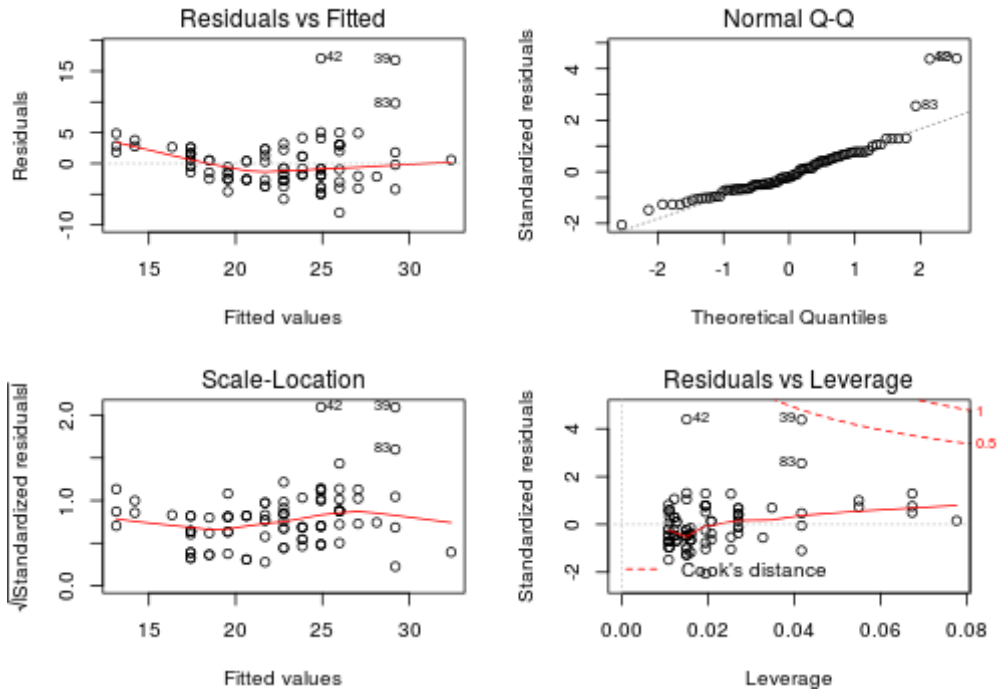
```
summary(res)
```

```
Call:
lm(formula = MPG.city ~ Width, data = Cars93)

Residuals:
    Min      1Q  Median      3Q     Max
-7.9834 -2.5543 -0.7689  2.0166 17.0881

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  96.7039     7.5105  12.876  < 2e-16 ***
Width        -1.0715     0.1081  -9.912 3.89e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.918 on 91 degrees of freedom
Multiple R-squared:  0.5192,    Adjusted R-squared:  0.5139
F-statistic: 98.26 on 1 and 91 DF,  p-value: 3.89e-16
```

Solution (3d)

- LS-regression diagnostic plots:

```
par(mfrow=c(2,2),mar=c(4,4,3,2))
plot(res)
```



Tasks / Exercises (4)

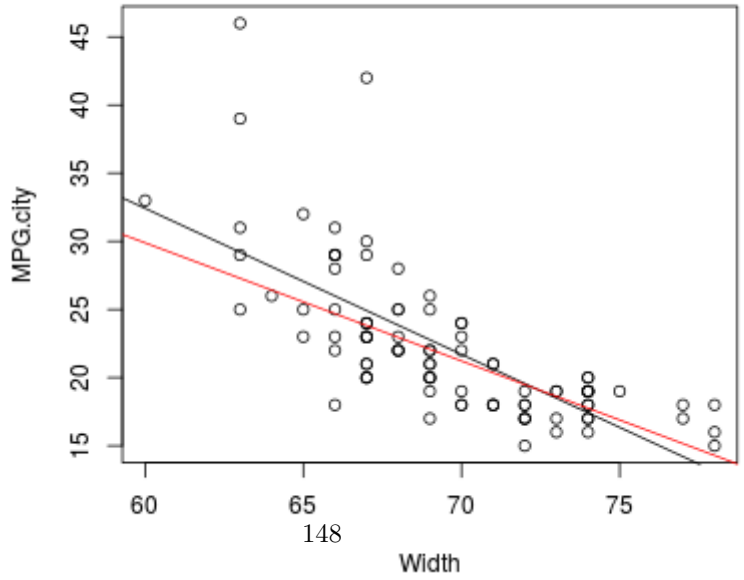
Linear regression:

- 4a) Use **MPG.city** as response variable, and **Width** as explanatory variable. Show them in a plot.
- 4b) Add the LS- and the robust regression line to the plot.
- 4c) Show and interpret the inference statistics from robust regression.
- 4d) Show and interpret the diagnostic plots from robust regression.

Solution (4a, 4b)

- Plot data, add LS- and robust regression line:

```
plot(MPG.city~Width,data=Cars93)
res <- lm(MPG.city~Width,data=Cars93)
abline(res)
library(robustbase)
res1 <- lmrob(MPG.city~Width,data=Cars93)
abline(res1,col="red")
```



Solution (4c)

- Robust regression inference statistics:

```
ss <- summary(res1)
ss$call
```

```
lmrob(formula = MPG.city ~ Width, data = Cars93)
```

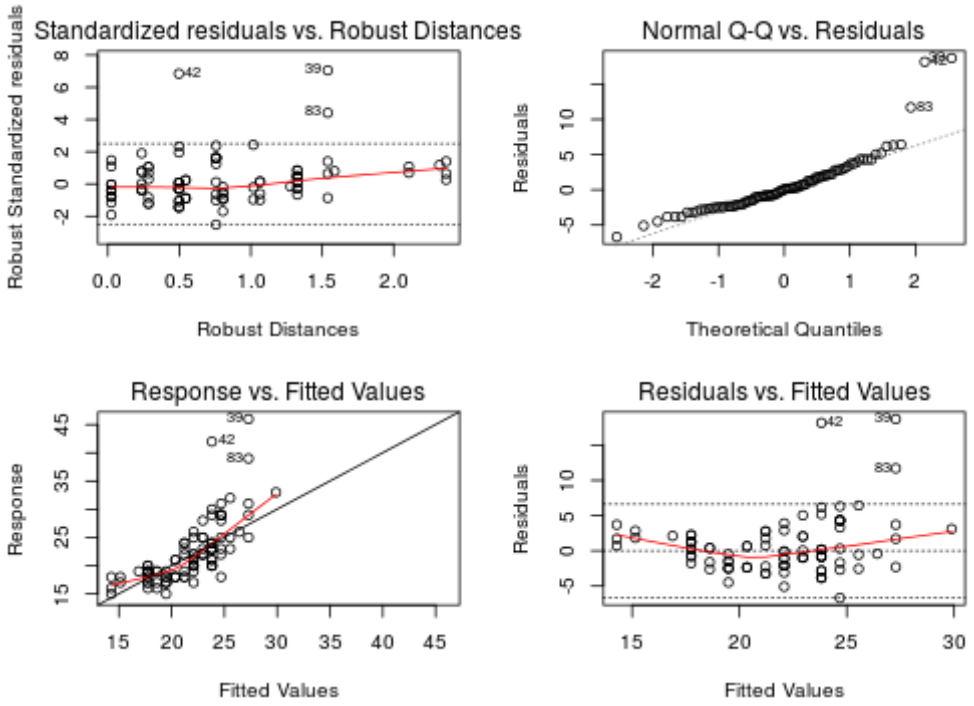
```
names(ss)
```

```
 [1] "call"          "terms"         "residuals"     "scale"
 [5] "rweights"      "converged"     "iter"          "control"
 [9] "df"            "coefficients"  "r.squared"     "adj.r.squared"
[13] "cov"           "aliased"       "sigma"
```

Solution (4d)

- Robust regression diagnostic plots:

```
par(mfrow=c(2,2),mar=c(4,4,3,2))
plot(res1,which=1:4)
```



Exercises: Classes

Alexander Kowarik, Bernhard Meindl

Tasks / Exercises (1)

- 1a) write a function **myMoment()** with parameters *x* and *n*, where *x* should be a numeric vector and *n* should be a postive integer number. The output of **myMoment()** should contain the result of \(\frac{\sum x^n}{obs} \) (with obs being the number of elements in *x*) and *n* in a list.
- 1b) modify **myMoment()** so that it returns an object of class *moment*
- 1c) implement a print function (using S3) for objects of class *moment* (returned by **myMoment()**)
- 1d) advanced: Implement 1c) using S4-classes and methods

Solution (1a, 1b)

- create a custom function **myMoment()**

```
## simple S3
myMoment <- function(x, n=2) {
  list(result=sum(x^n)/length(x), n=n)
}
myMoment(4)
```

```
$result
[1] 16

$n
[1] 2
```

- Let **myMoment()** return an object of class *moment*

```
## S3, more sophisticated
myMoment <- function(x, n=2) {
  out <- list(result=sum(x^n)/length(x), n=n)
  class(out) <- "moment"
  out
}
myMoment(4)
```

```
$result
[1] 16

$n
[1] 2

attr(,"class")
[1] "moment"
```

Solution (1c)

implement a print-method for class *moment*

```
## print method for S3 methods
print.moment <- function(x, ...){
  cat(paste0("Moment=", x$result, " (parameter n=",x$n,")"))
}
res <- myMoment(4)
print(res)
```

```
Moment=16 (parameter n=2)
```

Solution (1d)

- for experienced programmers:

```
## S4 class style
setClass("momentS4", representation(result="numeric", n="numeric"))
momentS4 <- function(x, n=2) {
  res <- sum(x^n)/length(x)
  new("momentS4", result=as(res, "numeric"), n=as(n, "numeric"))
}
momentS4(x=4, n=2)
```

```
An object of class "momentS4"
Slot "result":
[1] 16

Slot "n":
[1] 2
```

Solution (1d)

```
setMethod("show", signature(object="momentS4"),
function(object) {
  cat(paste0("Moment=", object@result, " (parameter n=",object@n,")"))
  invisible(object)
})
```

```
[1] "show"
```

```
resS4 <- momentS4(x=4, n=2)
print(resS4)
```

```
Moment=16 (parameter n=2)
```