Introduction to R

Roel Bakker

6-Apr-2022

Installing R

- download R from r-project.org [http://cran.at.r-project.org/]
- download RStudio Desktop from rstudio.org [http://www.rstudio.com/ide/download/]
- install both R and RStudio
- test successful installation with:

```
demo(graphics)
```

Your first R session ....

```
x = rnorm(12, mean = -5)
x
```

```
##  [1] -3.459797 -6.221416 -3.855667 -4.728501 -3.444439 -
##  [8] -4.623578 -4.639708 -4.667697 -5.462411 -4.856119
```

```
mean(x)
```

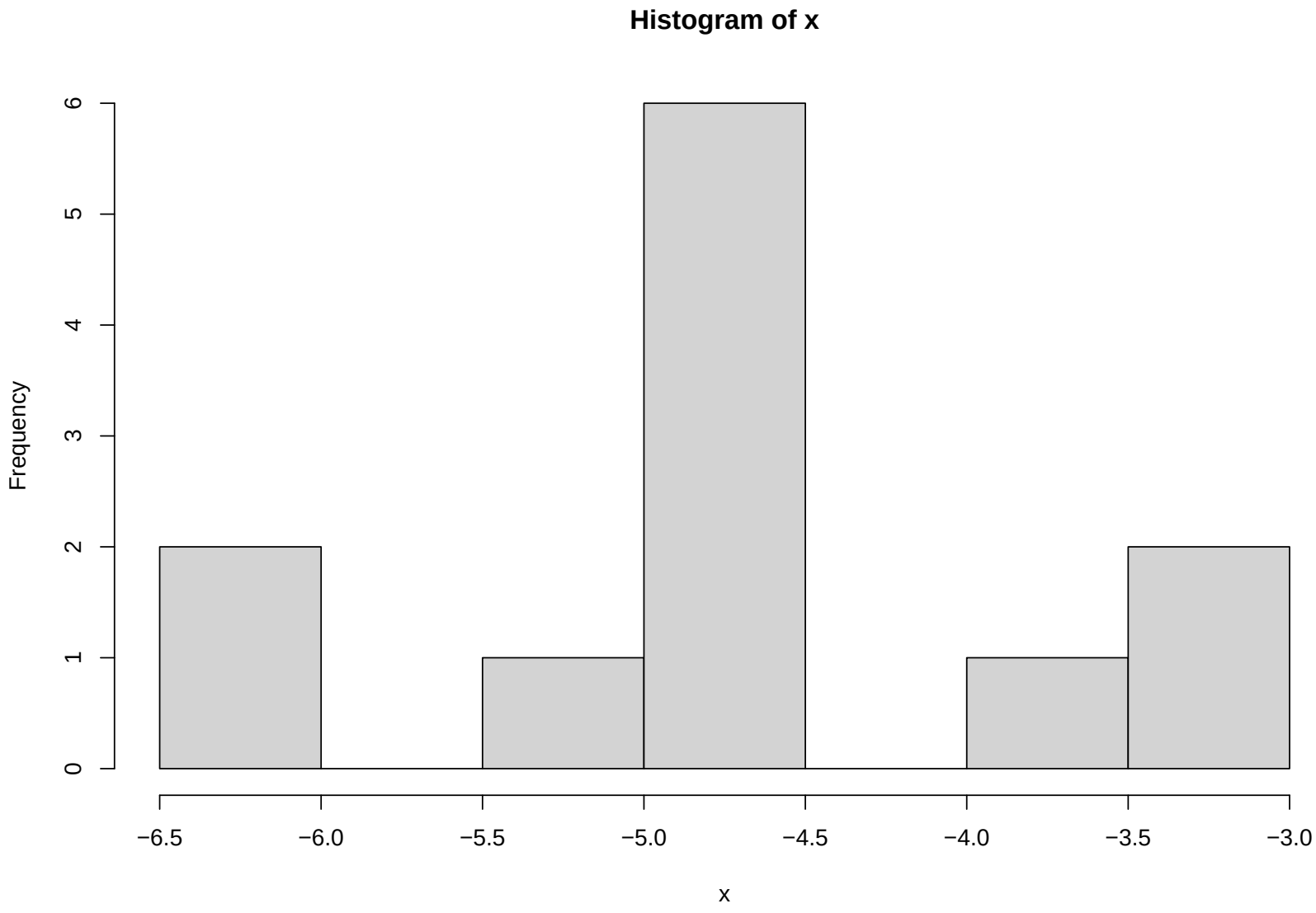
```
## [1] -4.748967
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -6.248  -5.008  -4.698  -4.749  -4.432  -3.444
```

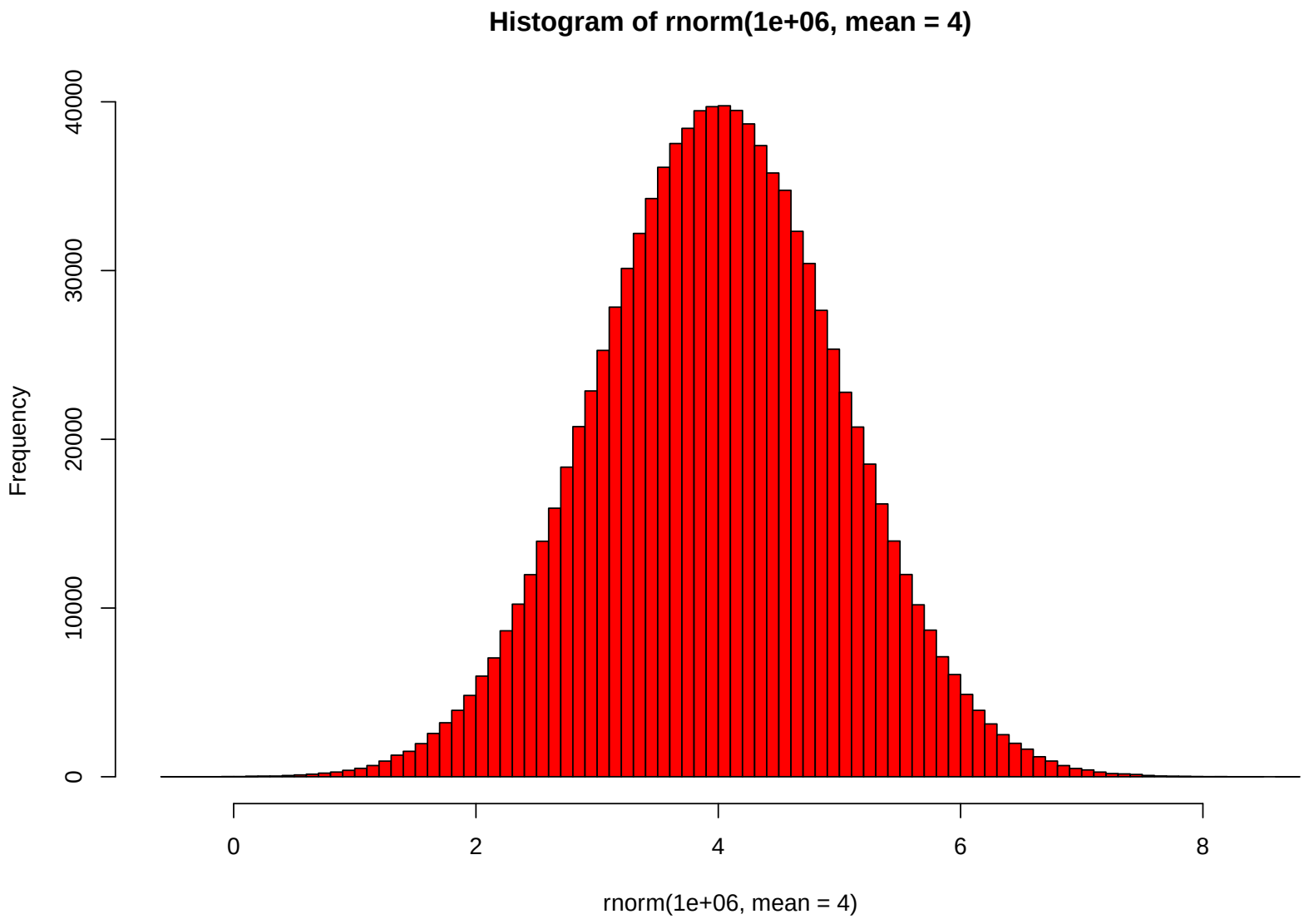
Draw a histogram

```
hist(x)
```



With more values

```
hist(rnorm(1000000,mean=4),breaks=100,col="red")
```



Working on the R command-line

- ĥ arrow keys to navigate your command history
- <- -> arrow keys to edit the command-line
- you can save the command history:

```
history()
```

- TAB for object/file completion

```
df = read.csv("data    ")
```

1.1 Expressions

```
2+3
```

```
## [1] 5
```

```
4*5
```

```
## [1] 20
```

```
3^(1/2)
```

```
## [1] 1.732051
```

```
2^8
```

```
## [1] 256
```

R **evaluates** these expressions, i.e. determines the **value** of these expressions

1.2 Expressions evaluating to logical values

```
2 < 3
```

```
## [1] TRUE
```

```
1 > 10
```

```
## [1] FALSE
```

```
5 == 5
```

```
## [1] TRUE
```

```
5 != 5
```

```
## [1] FALSE
```

Short for **TRUE** and **FALSE**

```
T
```

```
## [1] TRUE
```

```
F
```

```
## [1] FALSE
```

1.3 Variables

Every variable has a **name** and a **type**.

R has the following data types: **character**, **numeric**, **integer** and **logical**.

The name of a variable may contain

- letters (upper/lower case matters)
- digits
- periods
- underscores and
- should start with a letter

Variables do not need to be declared first.

A variable is assigned a value in an 'assignment' statement (in addition to = it is possible to use <- as assignment operator)

```
x = 200
half.x  =  x/2
half.x
```

```
## [1] 100
```

```
y <- 12.3
y
```

```
## [1] 12.3
```

1.3.1 character

The class() function is used to lookup the type of a variable.

```
firstName = "John"
class(firstName)
```

```
## [1] "character"
```

```
firstName
```

```
## [1] "John"
```

1.3.2 numeric

```
heightInCm = 178.2
class(heightInCm)
```

```
## [1] "numeric"
```

```
heightInCm
```

```
## [1] 178.2
```

1.3.3 integer

```
numberOfChildren = 2
class(numberOfChildren)
```

```
## [1] "numeric"
```

```
numberOfChildren = 2L # number of children should be integ
class(numberOfChildren)
```

```
## [1] "integer"
```

```
numberOfChildren
```

```
## [1] 2
```

1.3.4 logical

```
RisFun = T
class(RisFun)
```

```
## [1] "logical"
```

```
RisFun
```

```
## [1] TRUE
```

1.3.5 Casting (coercion)

"as" can be used to cast to a different type:

```
y = as.integer(3.9)
class(y)
```

```
## [1] "integer"
```

```
y # the value of y is truncated
```

```
## [1] 3
```

```
x = as.character(33)
class(x)
```

```
## [1] "character"
```

```
x
```

```
## [1] "33"
```

1.4 Functions

A function can be called with the name of the function followed by the arguments of the function:

```
max(9,3,4,2,11,2)
```

```
## [1] 11
```

```
sum(2,3,4)
```

```
## [1] 9
```

```
prod(5,6,7)
```

```
## [1] 210
```

1.5 Help

```
? sum
?? product
```

1.6 Files

- dir() shows the files in the working directory
- ls() shows the variables defined

```
dir()
ls()
```

2.1 Vectors: what is a vector?

A vector is an ordered collection of values of the same type

```
lengths = c (178.2, 180.2, 177.4) ## c for "combine"
lengths
```

```
## [1] 178.2 180.2 177.4
```

```
firstNames = c ("John","Peter","James","Mary")
firstNames
```

```
## [1] "John"  "Peter" "James" "Mary"
```

2.1 Vectors: vector of length 1

R works with vectors by default.
A variable with one value is a vector of length 1.

```
x = c(10,20,30)
x
```

```
## [1] 10 20 30
```

```
length(x)
```

```
## [1] 3
```

```
y = 33
length(y)
```

```
## [1] 1
```

```
y[1]
```

```
## [1] 33
```

2.2 Sequence vectors

```
z = 1:10
z
```

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

```
z = seq(1,5)
z = seq(from=1,to=5,by=1/3)
z
```

```
##  [1] 1.000000 1.333333 1.666667 2.000000 2.333333 2.666
##  [9] 3.666667 4.000000 4.333333 4.666667 5.000000
```

```
9:5
```

```
## [1] 9 8 7 6 5
```

2.3 Vectors: selecting elements

```
x = 100:120
x[1]            ## the first element is element nr 1 (not 0
```

```
## [1] 100
```

```
x[c(12,14)]     ## elements 12 and 14
```

```
## [1] 111 113
```

```
x[1:5]          ## elements 1 thru 5
```

```
## [1] 100 101 102 103 104
```

```
x[-c(1,3,5,7)] ## excludes elements 1, 3, 5 and 7
```

```
##  [1] 101 103 105 107 108 109 110 111 112 113 114 115 116
```

2.4 Vectors: assigning values to elements

```
x = rep(0,10)
x [1: 3] = 2
x [5: 6] = c(5,NA)
x [7:10] = c(1,9)
x
```

```
##  [1]  2  2  2  0  5 NA  1  9  1  9
```

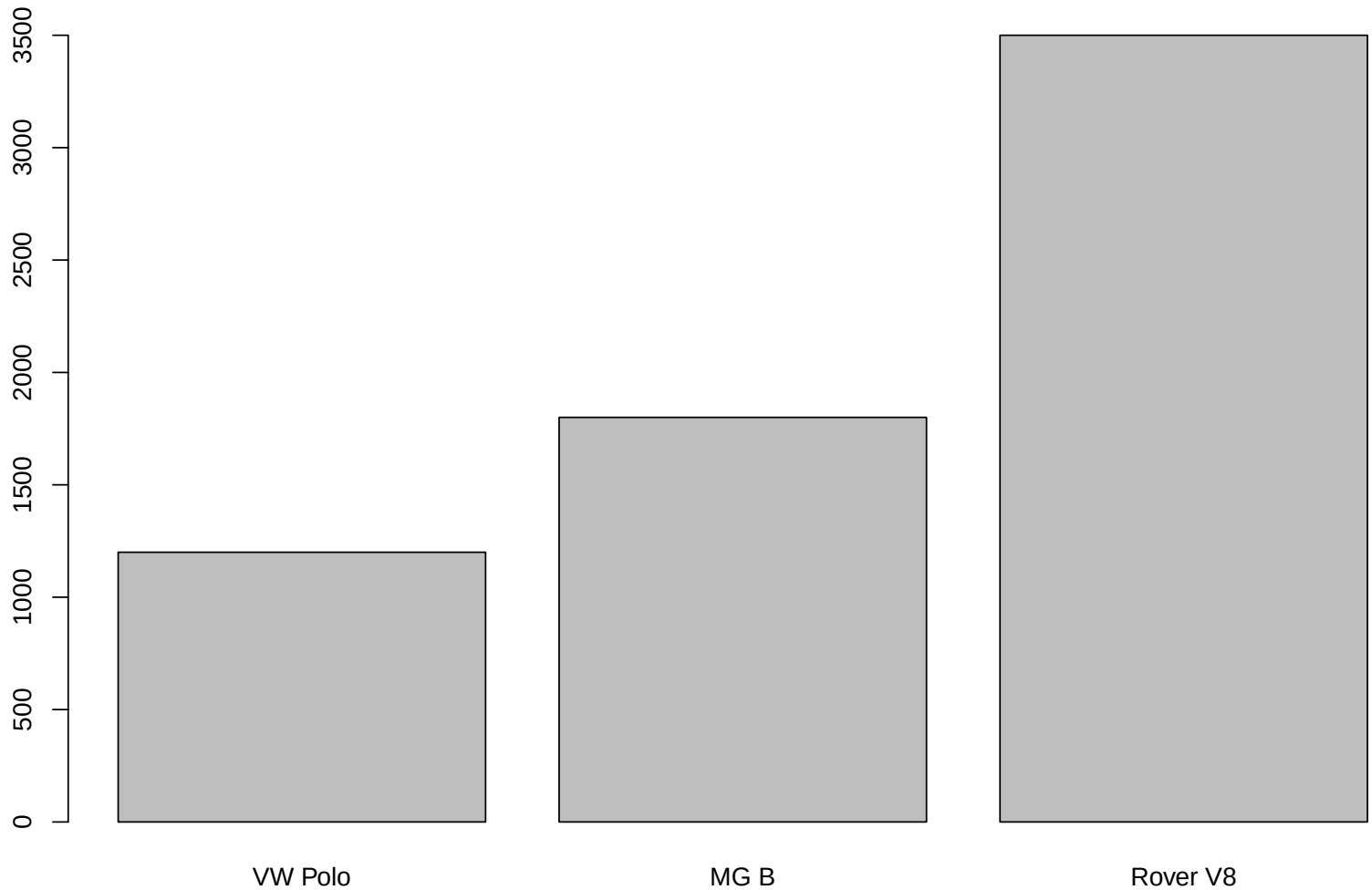
2.5 Vectors: indexing elements by name

```
cc = c(1200,1800,3500)
names(cc)=c("VW Polo","MG B","Rover V8")
cc["MG B"]
```

```
## MG B
## 1800
```

2.6 Vectors: plotting element values

```
cc = c(1200,1800,3500)
names(cc)=c("VW Polo","MG B","Rover V8")
barplot(cc)
```



2.8 Comparing vectors

```
a = c(1,2,3)
b = c(4,2,5)
a == b
```

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

The result is a vector of T/F values i.e. a logical vector *****
2.9 Logical vectors

If a vector is compared to a single value all elements of that vector are compared; the result is a logical vector:

```
x = 1:5
x>2
```

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

A logical vector can be used to select elements of the original vector:

```
x = 1:5
select = x>2
x[select]
```

```
## [1] 3 4 5
```

in SQL: SELECT * FROM NUMBERS WHERE x>2; ***** ### 2.9 Logical vectors: more examples

```
x = 1:10
gt5 = (x > 5)
lt8 = (x < 8)
x[gt5 & lt8]
```

```
## [1] 6 7
```

Use & (not &&) as 'short-circuit' boolean operators may have unexpected effects in R !! *****

```
lt5 = (x < 5)
gt8 = (x > 8)
```

2.10 Applying functions to vectors

```
z = c(9,49,16,36)
min(z)
```

```
## [1] 9
```

```
range(z)
```

```
## [1]  9 49
```

```
sqrt(z)
```

```
## [1] 3 7 4 6
```

```
sum(z)
```

```
## [1] 110
```

```
summary(z)
```

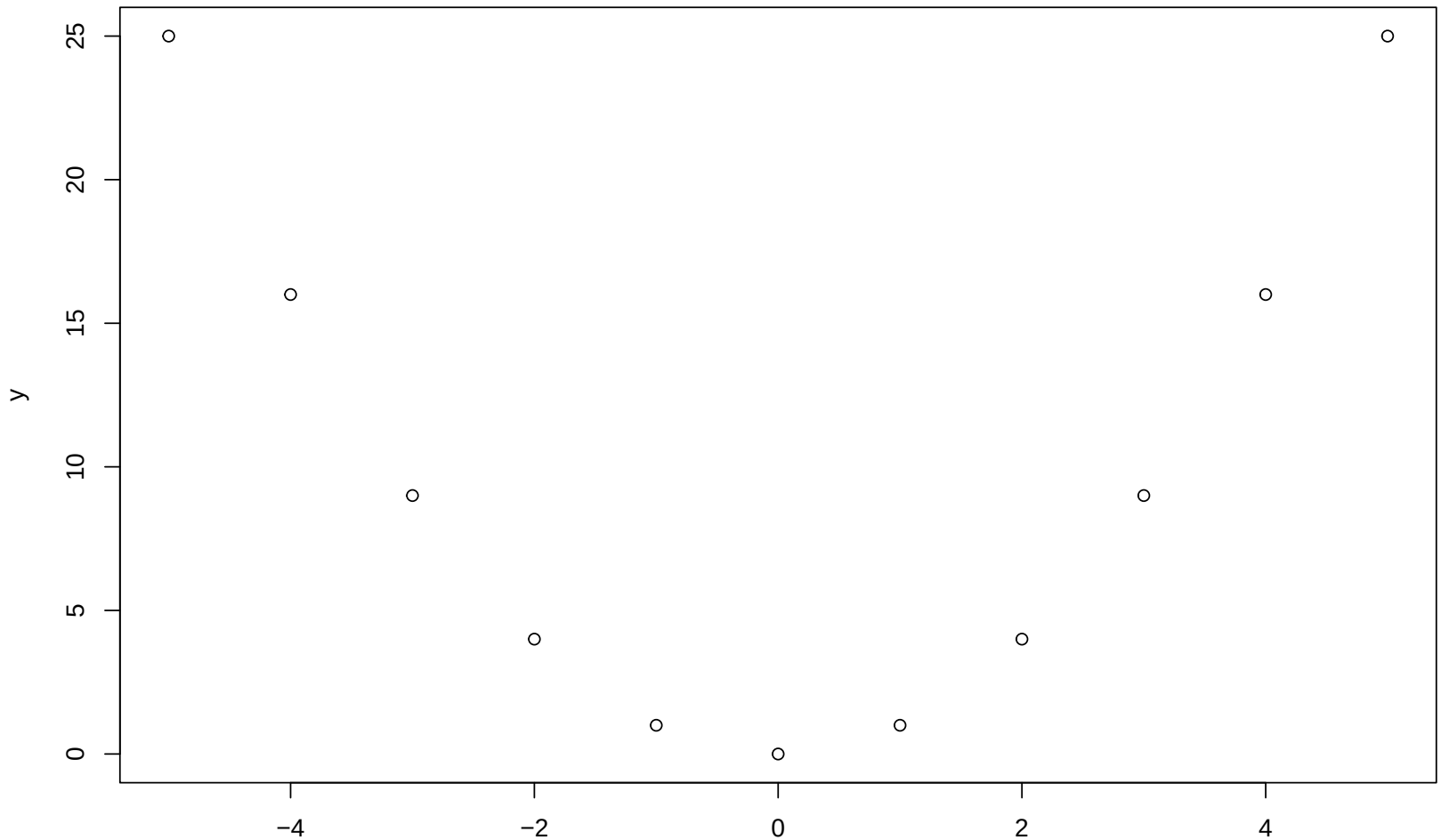
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   14.25   26.00   27.50   39.25   49.00
```

2.11 Often used functions with vectors

```
length()
rev()
sum(), cumsum(), prod(), cumprod()
mean(), sd(), var(), median()
min(), max(), range(), summary()
exp(), log(), sin(), cos(), tan()
round(), ceil(),
floor(), signif()
sort(), order(), rank()
which(), which.max()
any(), all()
```

2.12 Scatter plots

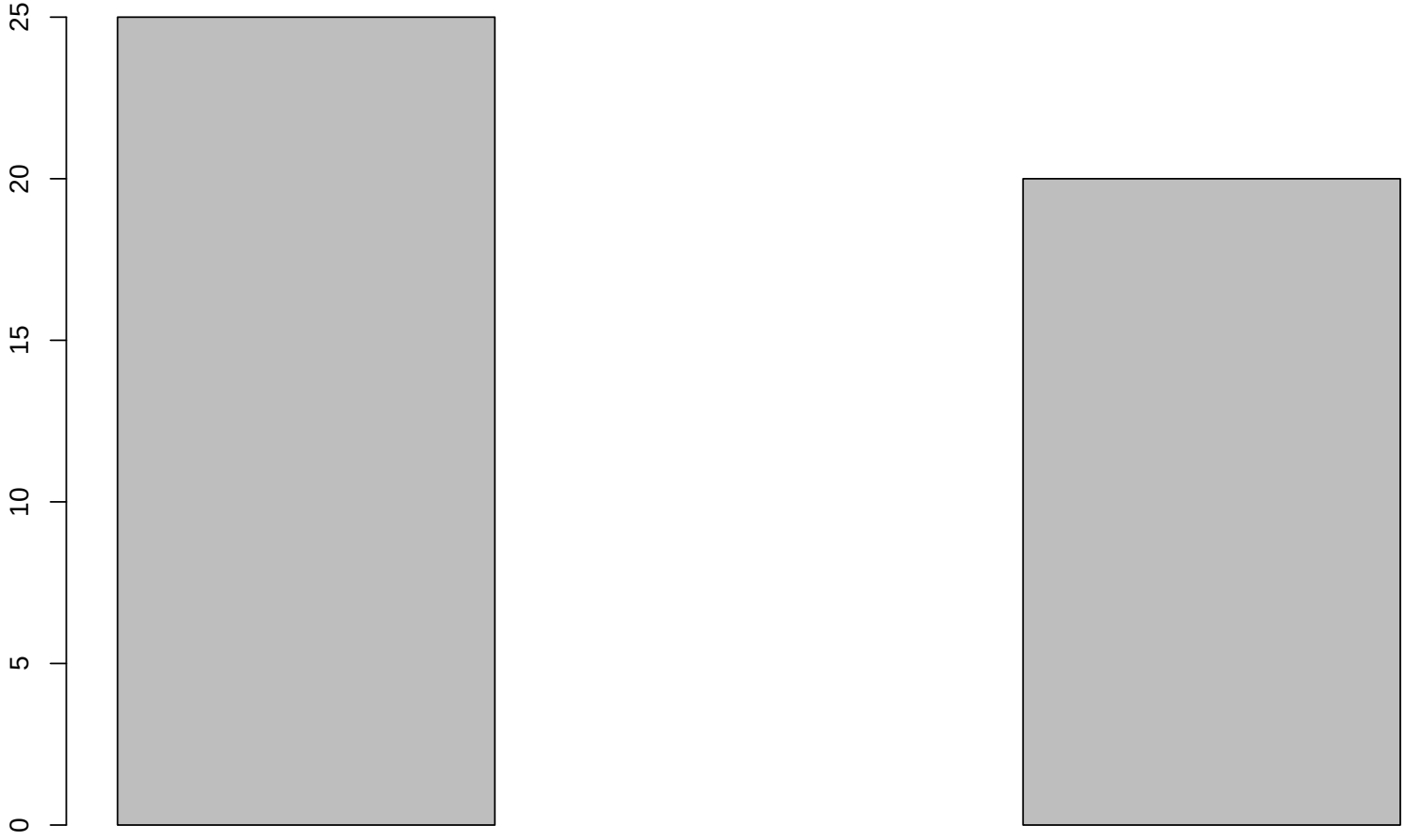
```
x = -5:5
y = x^2
plot(x,y)
```



2.13 NA (Not Available)

NA indicates that a value is not available (cf NULL in SQL)

```
barplot(c(25,NA,20))
```



'Missing values' are denoted with NA NA cannot be compared with other values

```
c(25,NA,20) == c(25,25,25)
```

```
## [1]  TRUE    NA FALSE
```

```
# NA == NA # neither with itself...
is.na(NA) # the function is.na() is used to test if a valu
```

```
## [1] TRUE
```

3.1 Matrices

Matrices have 2 dimensions

```
m1 = matrix(9,nrow=3,ncol=3)
m2 = matrix(c(11:14,21:24),nrow=2,ncol=4,byrow=T)
m2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   12   13   14
## [2,]   21   22   23   24
```

3.2 Selecting elements of a matrix

```
# a single element
m2[1,3]
```

```
## [1] 13
```

```
# selecting a column
m2[,1]
```

```
## [1] 11 21
```

```
# selecting a row
m2[2,]
```

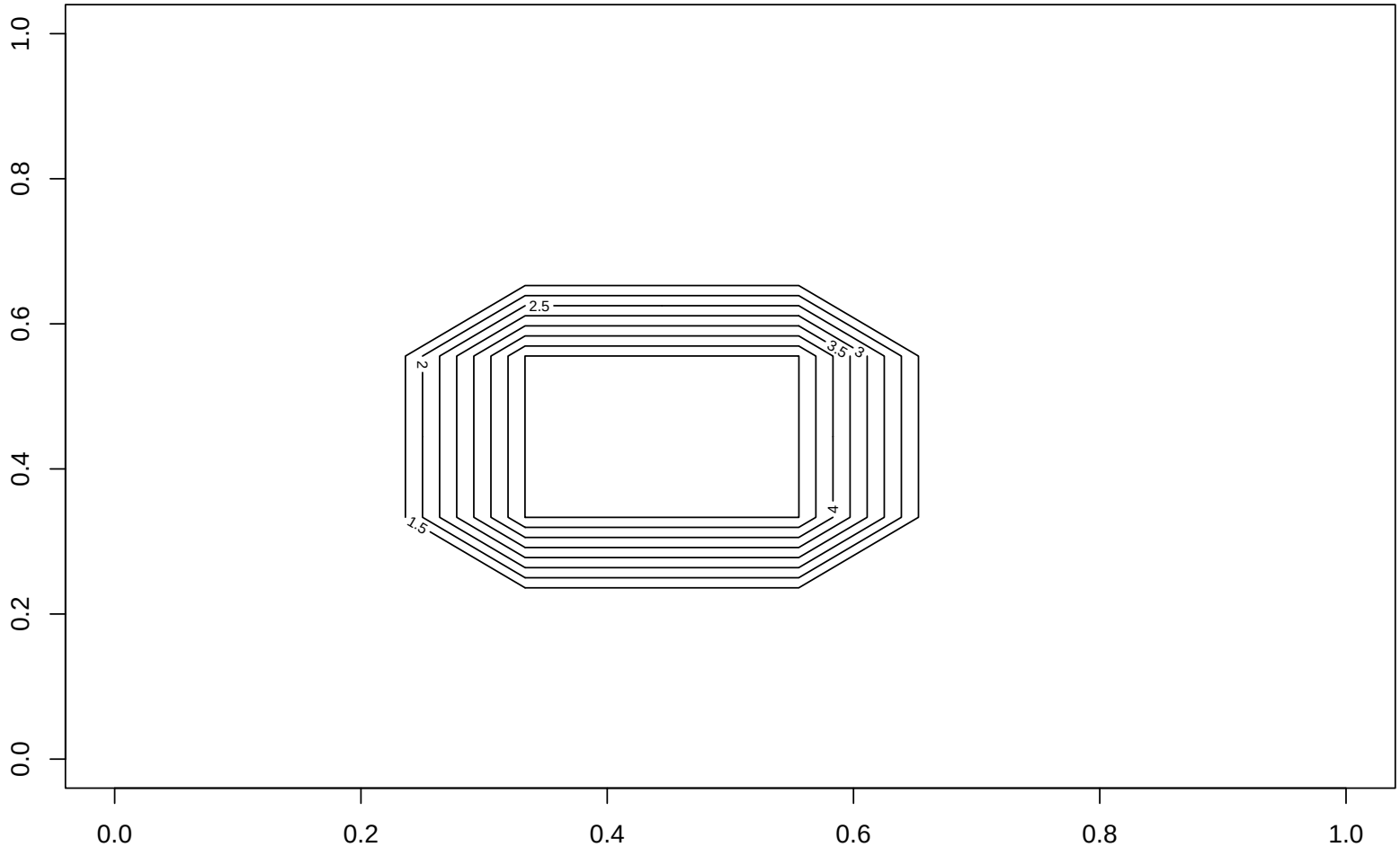
```
## [1] 21 22 23 24
```

3.3 Plotting matrix values

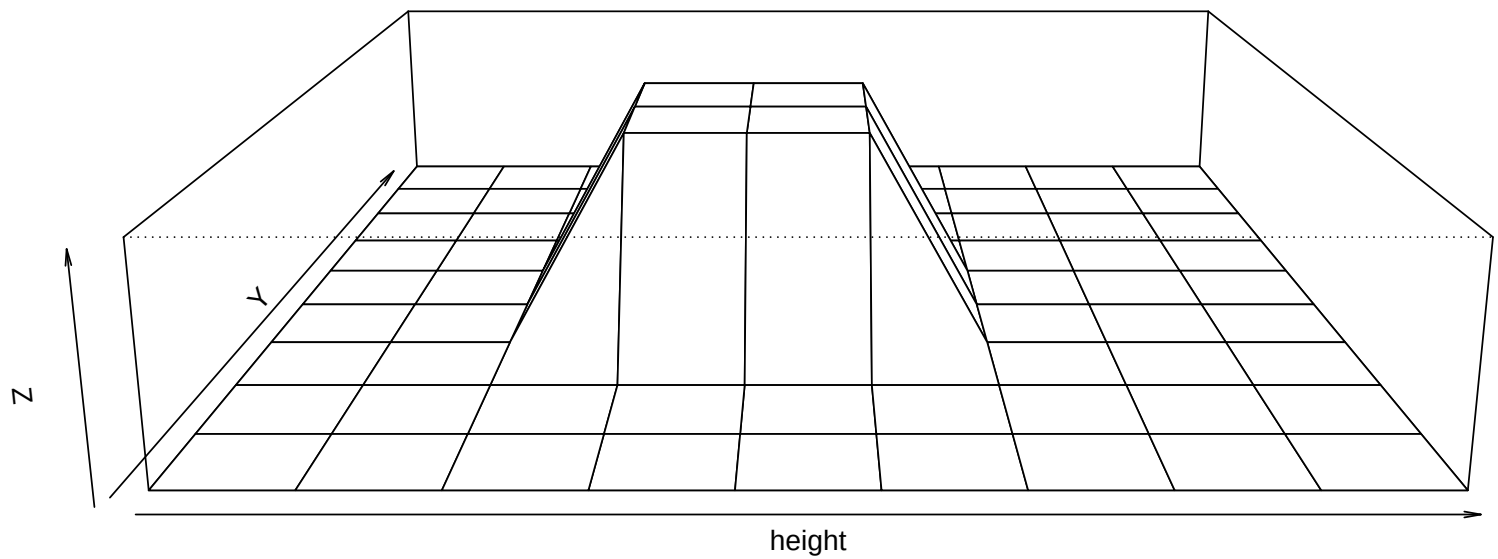
```
height=matrix(1,10,10)
height[4:6,4:6]=5
height
```

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

```
height=matrix(1,10,10)
height[4:6,4:6]=5
contour(height)
```



```
height=matrix(1,10,10)
height[4:6,4:6]=5
persp(height,expand=0.2)
```



4.1 Calling R functions

```
? read.csv
read.csv(file, header = TRUE, sep = ",", quote="\"", dec="
# valid ways to call a function:
read.csv("data.csv",F)                  # correct
read.csv(file="data.csv",header=F)# clear
read.csv("data.csv",F,";")              # correct
read.csv(file="data.csv",header=F,sep=";") # clear
read.csv("data.csv",";")                # error!
read.csv("data.csv",sep=";")            # correct
```

4.2 Writing R functions

```r
mult = function(a,b){
  a*b # the same as return(a*b)
}
mult(5,6)
```

```
## [1] 30
```

4.3 Scope in functions

```
a = 1
f = function(x){
  a = 2*a
  x+a
}
f(3)
```

```
## [1] 5
```

```
a
```

```
## [1] 1
```

```r
g = function(x){
  x = x+3
  x
}
a=1:3
g(a)
```

```
## [1] 4 5 6
```

```r
a
```

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

4.3 Scope in functions

- Functions in R don't have unexpected 'side-effects'.
- When you write a function: pretend to be living in the body of the function, where only data is available that is passed through the function arguments and where you can only pass a value to the outside world through the return value...
- This means that any result of a function should be passed as a return value...
- This is not a problem as a combination of values can be passed as a list e.g. list(a,b,x)

5.1 Data frames

Multiple (column) vectors possibly of different types, all of the same lengths

```
vector1 = c(178.2, 180.2, 177.4, 166.2)
vector2 =  c("John","Richard","Peter","Sophie")
myDataFrame = data.frame(heights=vector1,firstNames=vector2
myDataFrame
```

```
##   heights firstNames
## 1   178.2       John
## 2   180.2    Richard
## 3   177.4      Peter
## 4   166.2     Sophie
```

5.2 Data frames - subsetting

```
myDataFrame[1,1:2]
```

```
##   heights firstNames
## 1   178.2       John
```

```
myDataFrame$firstNames
```

```
## [1] "John"    "Richard" "Peter"   "Sophie"
```

5.3 Data frames - logical subsetting

```
myDataFrame[myDataFrame$firstNames=="John",]
```

```
##   heights firstNames
## 1   178.2        John
```

```
myDataFrame[myDataFrame$heights < 170,]
```

```
##   heights firstNames
## 4   166.2     Sophie
```

6.1 Lists

A vector with values of different types

```
p = list(temp=-5,precipitation="snow",sunny=T)
p
```

```
## $temp
## [1] -5
##
## $precipitation
## [1] "snow"
##
## $sunny
## [1] TRUE
```

```
p[[1]]
```

```
## [1] -5
```

```
p[["precipitation"]]
```

```
## [1] "snow"
```

7. Factors

Qualitative variables used in models

```
smoker = c("yes","no","yes","yes")
smokerFactor = as.factor(smoker)
smokerFactor
```

```
## [1] yes no  yes yes
## Levels: no yes
```

A factor contains data indexed by its 'levels' i.e. an enumeration of the values

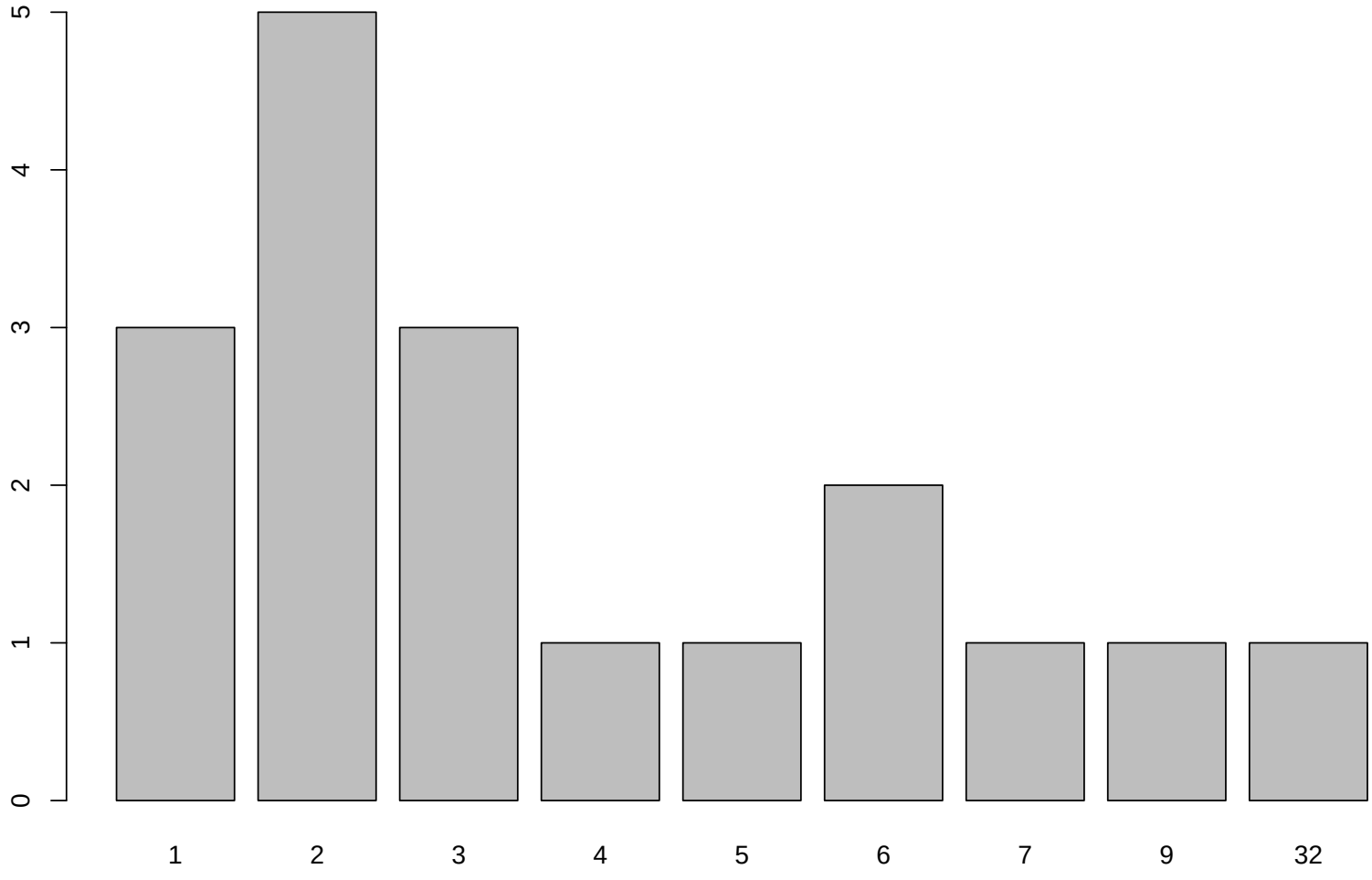
```
delay = c(1,2,2,7,3,2,3,9,1,1,2,3,6,6,5,4,32,2)
factor(delay)
```

```
##  [1] 1  2  2  7  3  2  3  9  1  1  2  3  6  6  5  4  32
## Levels: 1 2 3 4 5 6 7 9 32
```

```
table(factor(delay))
```

```
##
```

```r
delay = c(1,2,2,7,3,2,3,9,1,1,2,3,6,6,5,4,32,2)
barplot(table(factor(delay)))
```



8. Summary Statistics

8.1 Mean

8.2 Median

8.3 Standard Deviation

8.4 Variance

8.5 Summary

8.1 Mean

```r
x = c(178.2, 180.2, 177.4)
mean(x)
```

```
## [1] 178.6
```

Mean is defined as: $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$

8.2 Median

```
a = c(1,2,3,4,5)
mean(a); median(a)
```

```
## [1] 3
```

```
## [1] 3
```

```
a[5]=20
mean(a); median(a)
```

```
## [1] 6
```

```
## [1] 3
```

8.3 Standard Deviation

Standard deviation is defined as

$$\sigma = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n}}$$

```
a = c(1,2,3,4,5)
sd(a)
```

```
## [1] 1.581139
```

8.4 Variance

Variance is defined as

$$\sigma^2 = \frac{\sum(x_i - \bar{x})^2}{n}$$

```
a = c(1,2,3,4,5)
var(a)
```

```
## [1] 2.5
```

```
sd(a)==sqrt(var(a))
```

```
## [1] TRUE
```

9. Naming conventions

Variable names should be short and speficic. Examples of often used styles:

Camel caps

```
myHeightCM = 188
```

Underscore

```
my_height_cm = 188
```

Dot separated

```
my.height.cm = 188
```

10. Style guides

- http://4dpiecharts.com/r-code-style-guide/
- http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html
- http://wiki.fhcrc.org/bioc/Coding_Standards

11. References

A free R course at Cognitive Class AI (IBM)

More advanced: my favorite book on Statistical Learning with R (free pdf ; Labs in R with Introduction to R in the 1st chapter)

Videos and slides of an excellent course accompanying Introduction to Statistical Learning (1st Ed) by Hastie and Tibshirani

Free course at edX of of the 2nd Ed of Introduction to Statistical Learning (2nd Ed) by Hastie and Tibshirani

Slides of the course Introduction to Data Science with R by Garrett Grolemund (RStudio)

Coursera Data Science Specialization

Data Science Specialization Community Site

O'Reilly Data

Try R at Code School