Introduction to R

732A99/TDDE01, Lecture 1c

LINKÖPING UNIVERSITY

What is R?

- Programming language for statistical computing and graphics

→ Developed in 1993 by Ross Ihaka and Robert Gentleman at University of Auckland

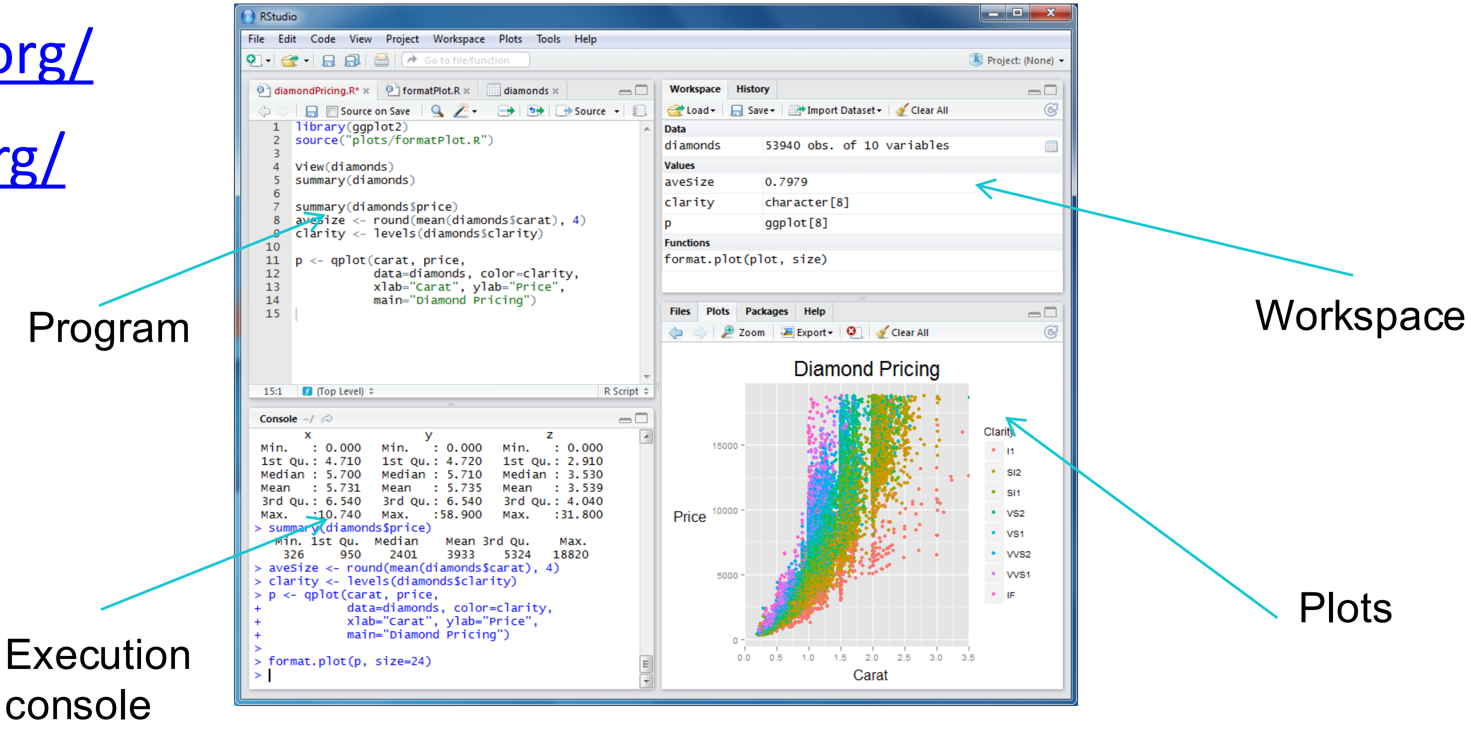
→ Managed by R Foundation and open-source

- Constantly increasing amount of software packages

→ Used in many fields and promoted by research

Software: RStudio

- Install R: http://www.r-project.org/
- Install RStudio: http://rstudio.org/



```r
library(ggplot2)
source("plots/formatPlot.R")

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")

```

```
       x               y               z         
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000  
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910  
 Median : 5.700   Median : 5.710   Median : 3.530  
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539  
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040  
 Max.   :10.740   Max.   :58.900   Max.   :31.800  
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    326     950    2401    3933    5324   18820 
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
> 
> format.plot(p, size=24)
> 
```

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979
clarity	character[8]
p	ggplot[8]
Functions	
format.plot(plot, size)	

LINKÖPING UNIVERSITY

RStudio: Basics

- Important to know:
  - Create a new file and save it (File menu)
  - Running one line or entire code (Edit menu)
  - Running one line in console
  - Workspace (Observe, Save, Clear)
  - Setting current directory (Tools)
  - Installing new package (Packages tabs)

RStudio: Basics

- Specific function
  - `help(`*`function`*`)`
- Help browser
  - `help.start()`
- Search for something in help
  - `help.search(“`*`expression”`*`)`
- Quick reminder of function arguments:
  - `args(`*`function`*`)`
- Examples of how to use function:
  - `example(`*`function`*`)`
- If some method is not installed on the computer:
  - `RSiteSearch(”expression")`

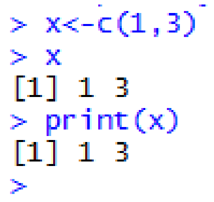
Getting started in R

- Each line of code forms a separate command
- Variables are initialized with -> or <- or =, e.g.
  - `a <- 3`
  - `3 -> a`
  - `a = 3`

  → R is case-sensitive (A and a!)
- Comments start with #, e.g.
  - `# R is a great programming language!`

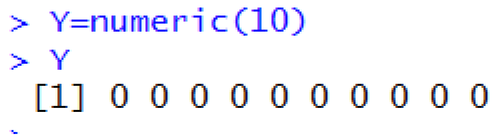
Vectors

- Creating a vector: `x <- c(1,3)`
- View the content of a vector: `x or print(x)`



```
> x<-c(1,3)
> x
[1] 1 3
> print(x)
[1] 1 3
>
```

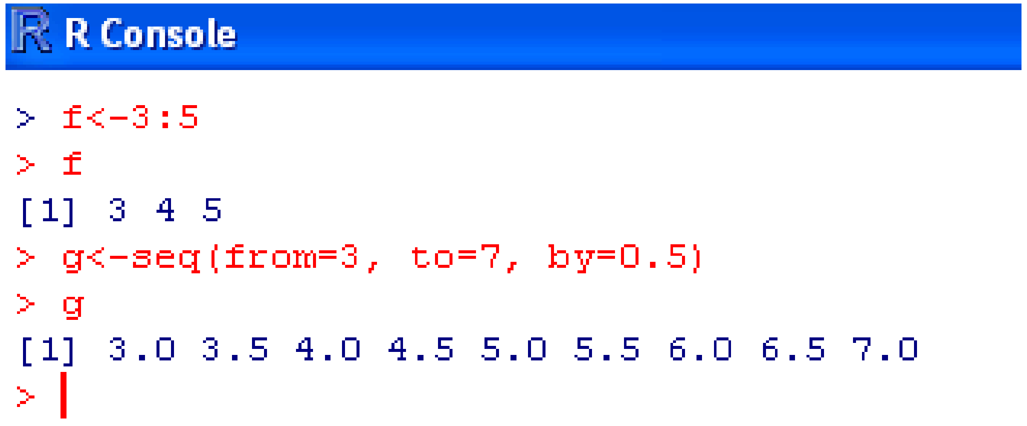
- Create an empty vector: `Y <- numeric(10)`



```
> Y=numeric(10)
> Y
 [1] 0 0 0 0 0 0 0 0 0 0
```

Sequences

- Sequences can be defined either by
  - `start:end`
  - `seq(from, to, by)`



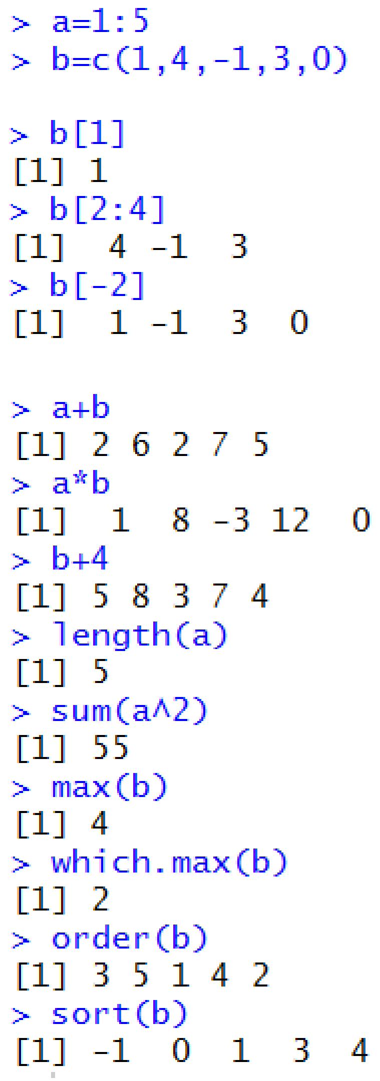
R Console

```
> f<-3:5
> f
[1] 3 4 5
> g<-seq(from=3, to=7, by=0.5)
> g
[1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
> 
```

LINKÖPING UNIVERSITY

Vector Operations

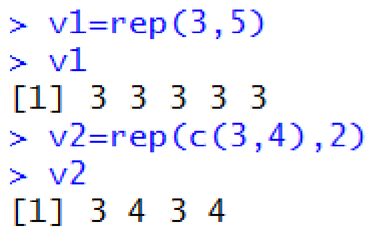
- Indexing: [] (Start at 1!)
- Element-wise operations:
  - +-*/^
  - Log, exp, sin, cos …
- Vector-wise operations:
  - Number of elements: length()
  - Sum of all elements: sum()
  - Max(), min(), which.min(), which.max()
  - Sort(), Order()
  - ….



```
> a=1:5
> b=c(1,4,-1,3,0)

> b[1]
[1] 1
> b[2:4]
[1]  4 -1  3
> b[-2]
[1]  1 -1  3  0

> a+b
[1] 2 6 2 7 5
> a*b
[1]  1  8 -3 12  0
> b+4
[1] 5 8 3 7 4
> length(a)
[1] 5
> sum(a^2)
[1] 55
> max(b)
[1] 4
> which.max(b)
[1] 2
> order(b)
[1] 3 5 1 4 2
> sort(b)
[1] -1  0  1  3  4
```

Vector operations

- Replication : `rep(vector, times)`



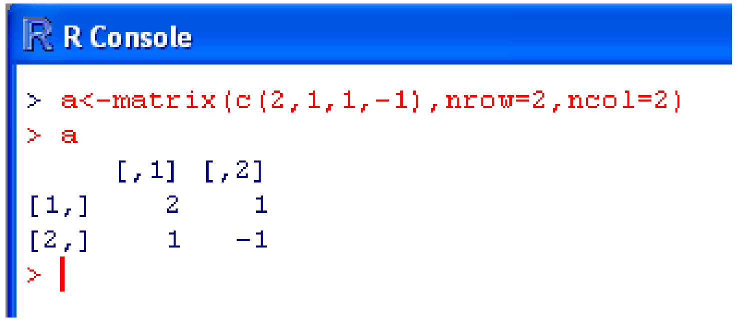
```
> v1=rep(3,5)
> v1
[1] 3 3 3 3 3
> v2=rep(c(3,4),2)
> v2
[1] 3 4 3 4
```

Matrices

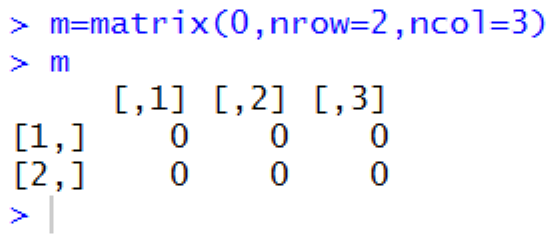
- Create a matrix:
  `a <- matrix(values, nrow=n, ncol=c)`

→ ”values” should be column-wise

→ if empty matrix, simply replace by 0



```
> a<-matrix(c(2,1,1,-1),nrow=2,ncol=2)
> a
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> 
```



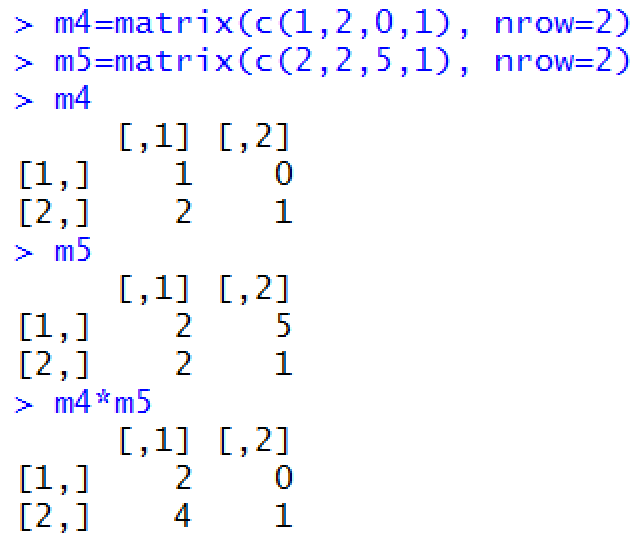
```
> m=matrix(0,nrow=2,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
> 
```

Matrix operations

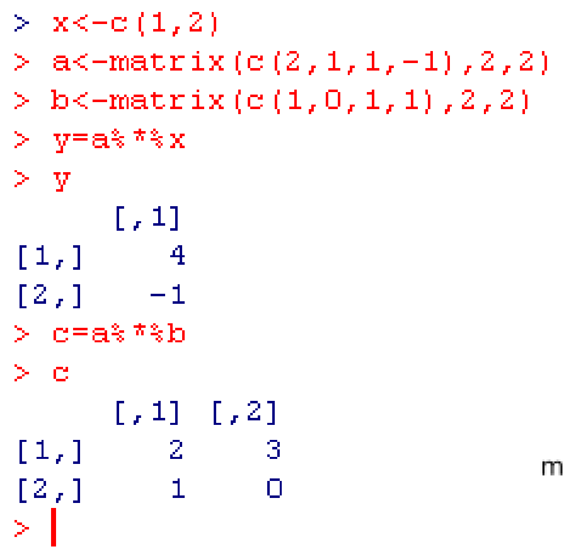
- Indexing for matrices:
  - Positive index: e.g., `x[1,6]` or `x[2:10,]`
  - Negative index: e.g., `x[2,-(1:5)]` (row 2 and all columns except 1:5)
  - Entire column or row: `y=x[2,]`
  - Extraction: `x[x>5]` (all values above 5)

Matrix operations

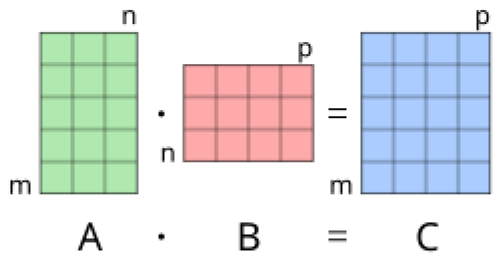
Usual vector operations can be applied, as well as matrix-specific ones:



```
> m4=matrix(c(1,2,0,1), nrow=2)
> m5=matrix(c(2,2,5,1), nrow=2)
> m4
     [,1] [,2]
[1,]    1    0
[2,]    2    1
> m5
     [,1] [,2]
[1,]    2    5
[2,]    2    1
> m4*m5
     [,1] [,2]
[1,]    2    0
[2,]    4    1
```

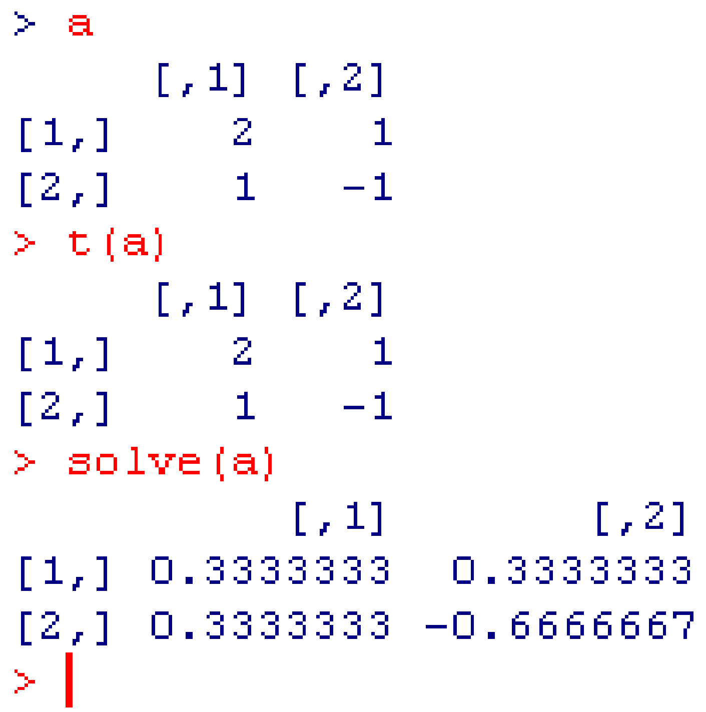


```
> x<-c(1,2)
> a<-matrix(c(2,1,1,-1),2,2)
> b<-matrix(c(1,0,1,1),2,2)
> y=a%*%x
> y
     [,1]
[1,]    4
[2,]   -1
> c=a%*%b
> c
     [,1] [,2]
[1,]    2    3
[2,]    1    0
> 
```



Matrix operations

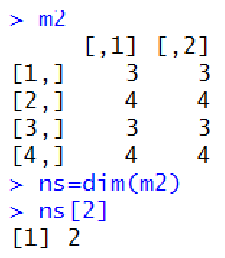
- Matrix operators/functions:
  - Transpose $b = a^T$ → `b=t(a)`
  - Inverse $b = a^{-1}$ → `b=solve(a)`
  - Solve $d = a^{-1}b$ → `d=solve(a,b)`



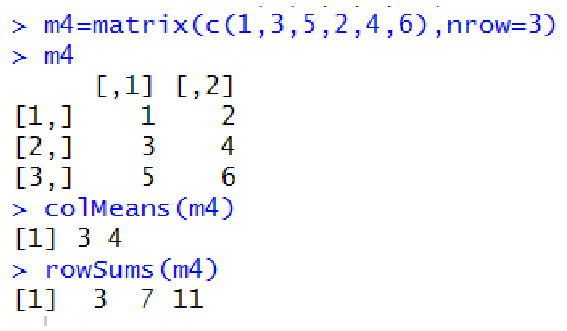
```
> a
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> t(a)
     [,1] [,2]
[1,]    2    1
[2,]    1   -1
> solve(a)
          [,1]       [,2]
[1,] 0.3333333  0.3333333
[2,] 0.3333333 -0.6666667
> 
```

Matrix operations

- Get Dimensions:
  - `dim(mat)`
  - `nrow(mat)`
  - `ncol(mat)`
- Row/column statistics:
  - `colMeans()`
  - `rowMeans()`
  - `colSums()`
  - `rowSums()`



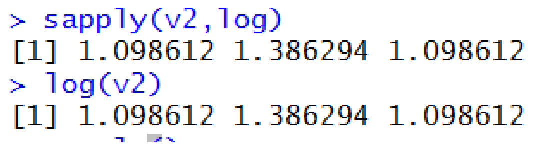
```
> m2
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    3    3
[4,]    4    4
> ns=dim(m2)
> ns[2]
[1] 2
```



```
> m4=matrix(c(1,3,5,2,4,6),nrow=3)
> m4
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> colMeans(m4)
[1] 3 4
> rowSums(m4)
[1]  3  7 11
```

- Apply a function over vector/matrix: `sapply()`, `apply()`, `lapply()`

→ Normally used when function works only element-wise

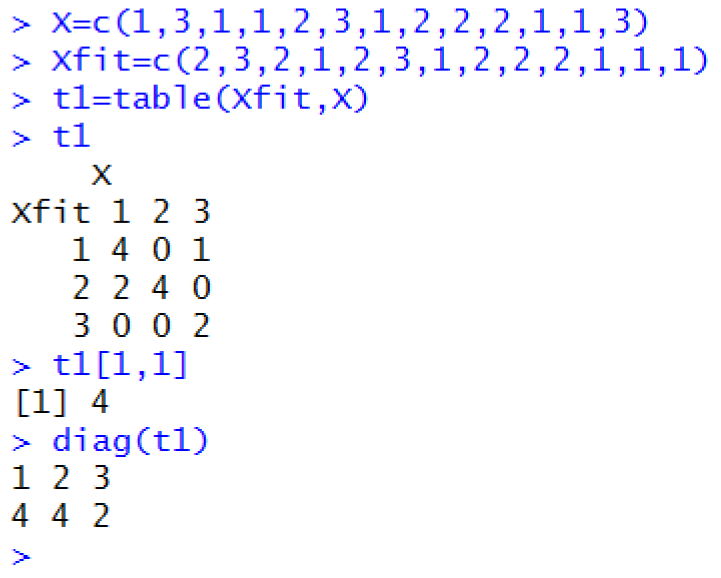


```
> sapply(v2,log)
[1] 1.098612 1.386294 1.098612
> log(v2)
[1] 1.098612 1.386294 1.098612
```

Matrix operations

- Create confusion matrix (classification):
  `table(X,Y)`

- Extract diagonal:
  `diag(X)`

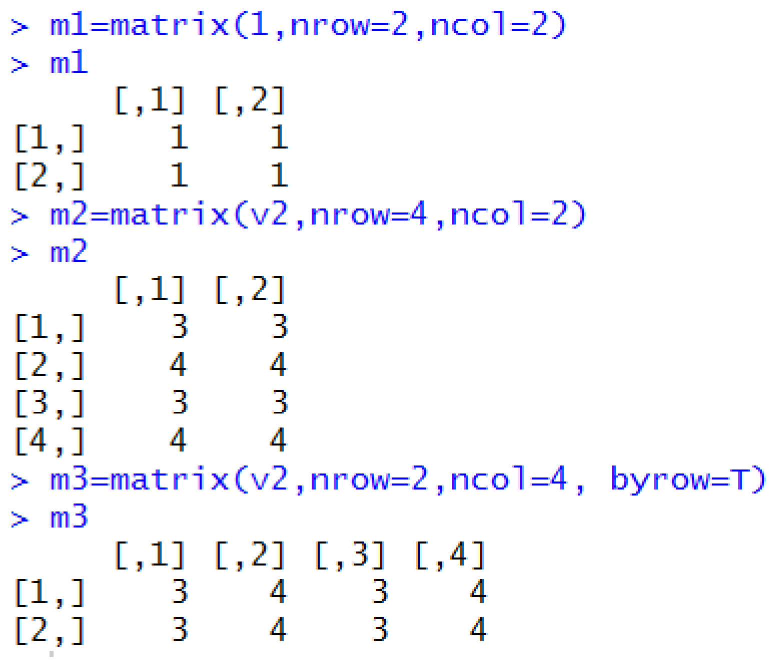


```
> X=c(1,3,1,1,2,3,1,2,2,2,1,1,3)
> Xfit=c(2,3,2,1,2,3,1,2,2,2,1,1,1)
> t1=table(Xfit,X)
> t1
    X
Xfit 1 2 3
   1 4 0 1
   2 2 4 0
   3 0 0 2
> t1[1,1]
[1] 4
> diag(t1)
1 2 3
4 4 2
>
```

LINKÖPING UNIVERSITY

Matrix operations

- Replication: `matrix()`



```
> m1=matrix(1,nrow=2,ncol=2)
> m1
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> m2=matrix(v2,nrow=4,ncol=2)
> m2
     [,1] [,2]
[1,]    3    3
[2,]    4    4
[3,]    3    3
[4,]    4    4
> m3=matrix(v2,nrow=2,ncol=4, byrow=T)
> m3
     [,1] [,2] [,3] [,4]
[1,]    3    4    3    4
[2,]    3    4    3    4
```

Lists

- Collection of objects

```
> d<-15;
> a<-matrix(c(1,2,3,4),2,2);
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> b<-list(first=d, second=a, x="mary")
> b
$first
[1] 15

$second
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$x
[1] "mary"
```

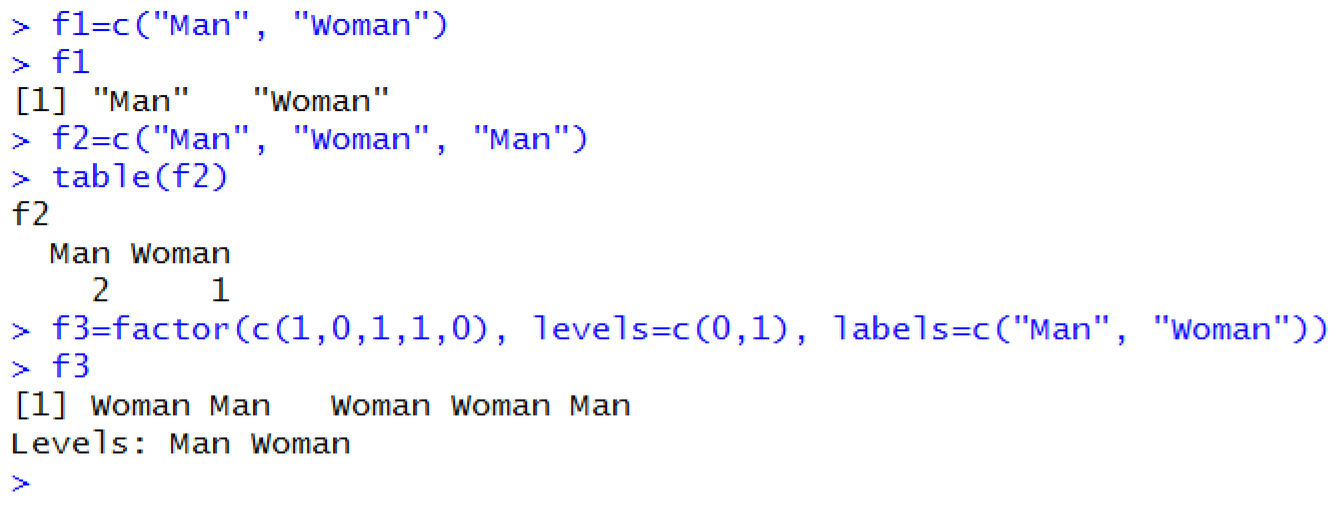
```
> b$x
[1] "mary"
> b[[3]]
[1] "mary"
```

LINKÖPING UNIVERSITY

Factors

- Categorize data, e.g.

→ Important for classification



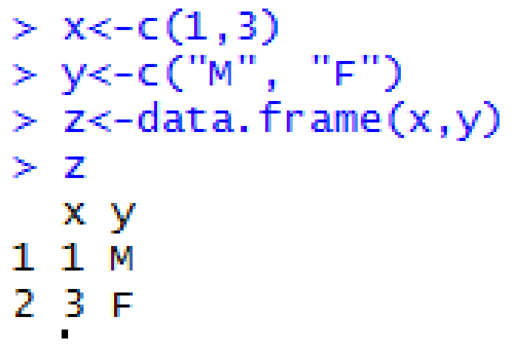
```
> f1=c("Man", "Woman")
> f1
[1] "Man"   "Woman"
> f2=c("Man", "Woman", "Man")
> table(f2)
f2
  Man Woman 
    2     1 
> f3=factor(c(1,0,1,1,0), levels=c(0,1), labels=c("Man", "Woman"))
> f3
[1] Woman Man   Woman Woman Man  
Levels: Man Woman
> 
```

Data frame

- Vectors and matrices of the same row length can be collected into a data frame

→ Used to store data of different types from same observation into a single table

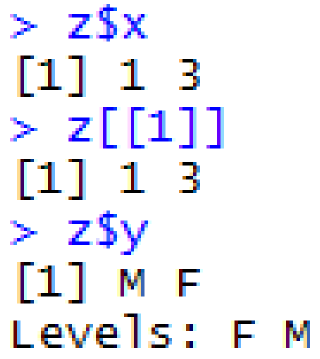
→ Use: `data.frame(object 1, object 2, … , object k)`



```
> x<-c(1,3)
> y<-c("M", "F")
> z<-data.frame(x,y)
> z
  x y
1 1 M
2 3 F
```

Data frame

- Any column in the data frame can be retrieved by `dataframe$object` or `dataframe[[col_nr]]`



```
> z$x
[1] 1 3
> z[[1]]
[1] 1 3
> z$y
[1] M F
Levels: F M
```

- Any row in the data frame can be extracted by using matrix notation, e.g., `z[1,]`

Read data from CSV files

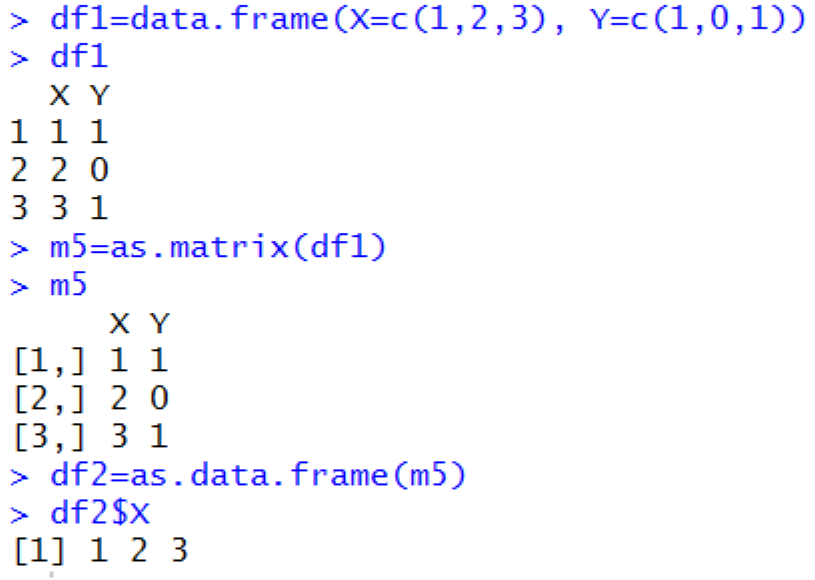
- 1. Specify exact file path or change current directory→ Set Working Directory or `setwd()`
- 2. Use:
  - `dataframe <- read.csv2(file_name)`
    → comma as decimals, semi-colon as separator
  - `dataframe <- read.csv(file_name)`

  → dot as decimals, comma as separator

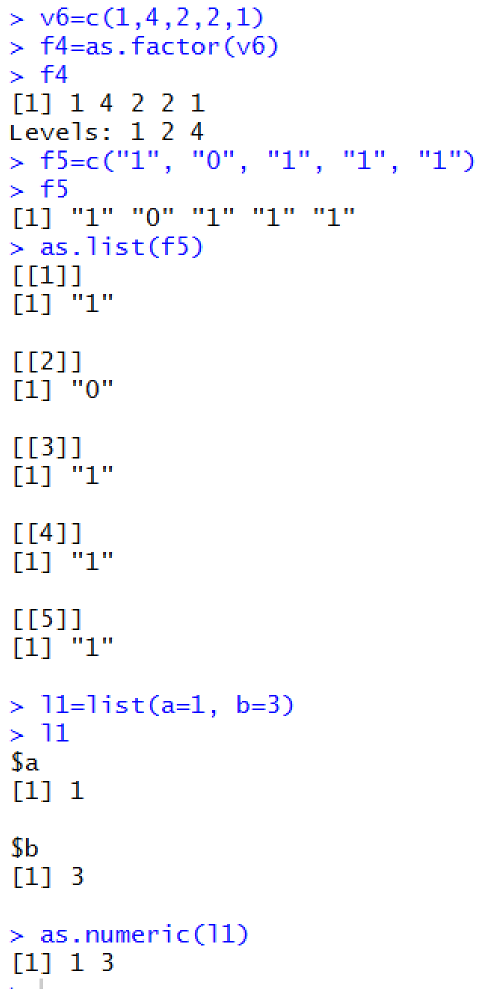
- Similar workflow for other file types , e.g. read from Excel files with packages xlsx, readxl

Conversion between data types

- E.g.,
  - Data frame to matrix
  - Matrix to data frame
  - Numeric to factor
  - Factor to numeric
  - List to vector
  - Vector to list



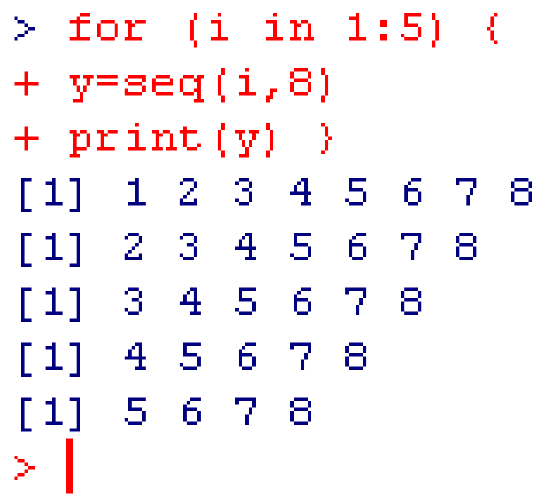
```
> df1=data.frame(X=c(1,2,3), Y=c(1,0,1))
> df1
  X Y
1 1 1
2 2 0
3 3 1
> m5=as.matrix(df1)
> m5
     X Y
[1,] 1 1
[2,] 2 0
[3,] 3 1
> df2=as.data.frame(m5)
> df2$X
[1] 1 2 3
```



```
> v6=c(1,4,2,2,1)
> f4=as.factor(v6)
> f4
[1] 1 4 2 2 1
Levels: 1 2 4
> f5=c("1", "0", "1", "1", "1")
> f5
[1] "1" "0" "1" "1" "1"
> as.list(f5)
[[1]]
[1] "1"

[[2]]
[1] "0"

[[3]]
[1] "1"

[[4]]
[1] "1"

[[5]]
[1] "1"

> l1=list(a=1, b=3)
> l1
$a
[1] 1

$b
[1] 3

> as.numeric(l1)
[1] 1 3
```

Loops

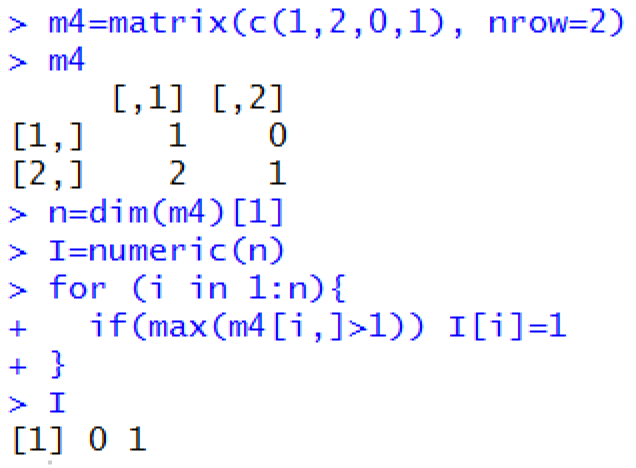
```
for (name  in expr1 )
{
…
}


while (condition)
{
…
}
```



```
> for (i in 1:5) {
+ y=seq(i,8)
+ print(y) }
[1] 1 2 3 4 5 6 7 8
[1] 2 3 4 5 6 7 8
[1] 3 4 5 6 7 8
[1] 4 5 6 7 8
[1] 5 6 7 8
> 
```

Loops with conditions

```
if (x==3) {

…

…

} else {

…

}
```



```
> m4=matrix(c(1,2,0,1), nrow=2)
> m4
     [,1] [,2]
[1,]    1    0
[2,]    2    1
> n=dim(m4)[1]
> I=numeric(n)
> for (i in 1:n){
+   if(max(m4[i,]>1)) I[i]=1
+ }
> I
[1] 0 1
```

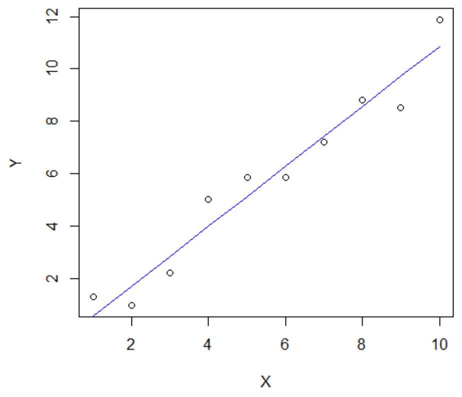
Using functions

→ We have been using many functions already throughout this lecture

- There are some obligatory parameters and optional parameters
- The optional parameters can be specified in different order

→ Use `?name_of_function` to see function parameters, e.g. `?lm`

```
X=1:10
Y=1:10+rnorm(10)
W=c(rep(1,5), rep(2,5))
mydata=data.frame(X,Y)

result=lm(Y~X,  weights=W,data=mydata)
?predict.lm
Fit=predict(result)

plot(X,Y)
points(X,Fit, type="l", col="blue")
```

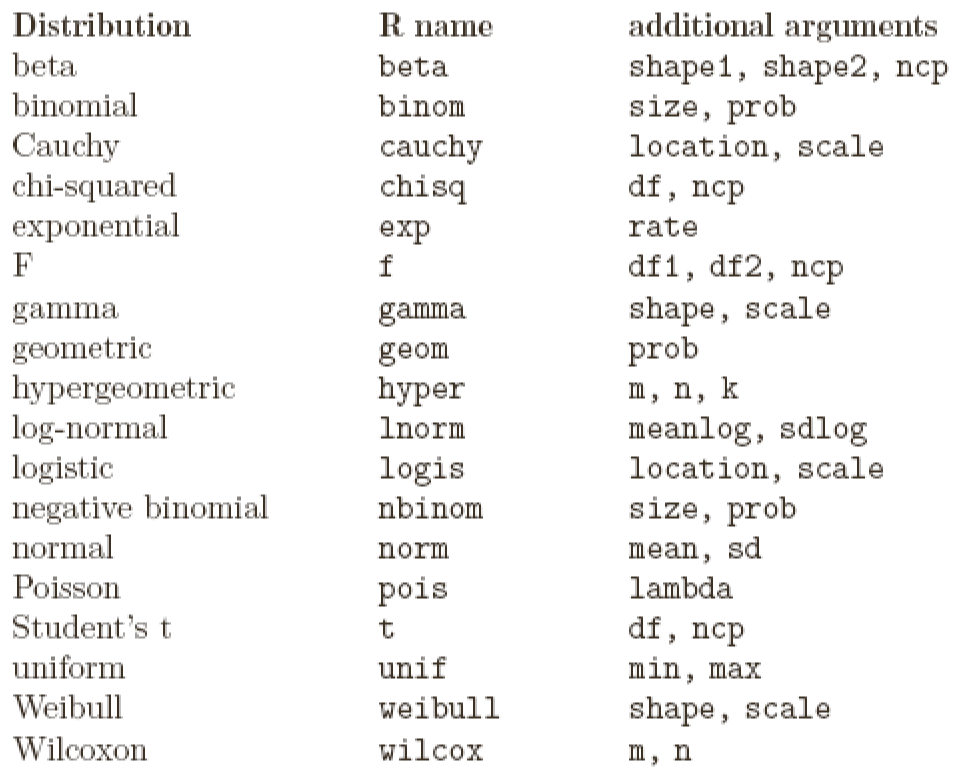


Random number generation

- Random is not random

→ **Use set.seed(12345) to get identical results**

- Plenty of random number generators available
  - Rnorm
  - Runif
  - …
- Use d for density, p for CDF, q for quantiles, and r for simulation: (ex: rnorm pnorm dnorm qnorm)



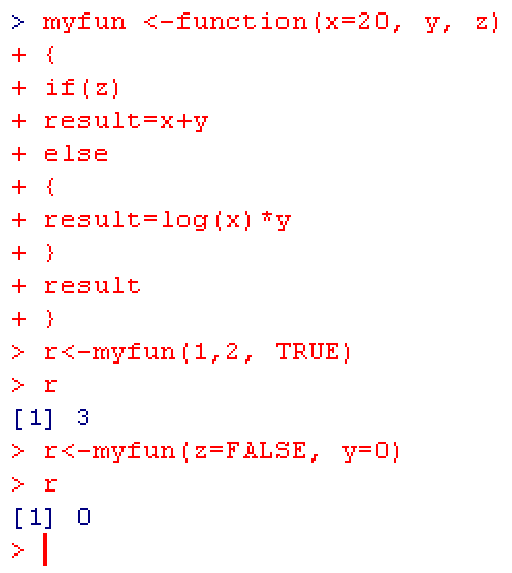
Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df2, ncp
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
log-normal	lnorm	meanlog, sdlog
logistic	logis	location, scale
negative binomial	nbinom	size, prob
normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

Writing your own functions

- Define a function with `x <- function() {}`

→ {} define function scope, i.e., what the function will do

- Specify which parameters a functions takes
- Specify what values a functions returns with `return(value)`

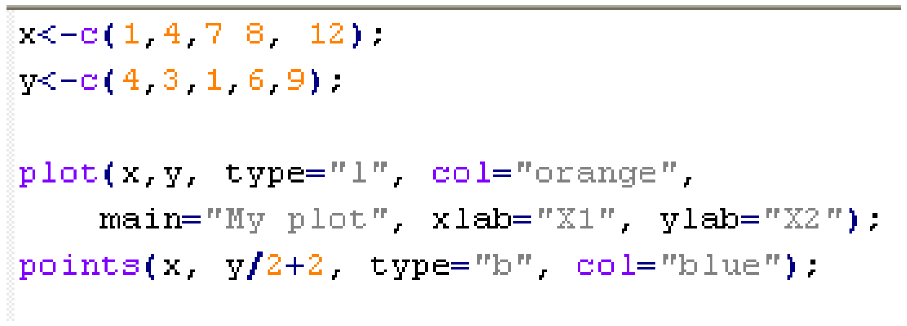


```
> myfun <-function(x=20, y, z)
+ {
+ if(z)
+ result=x+y
+ else
+ {
+ result=log(x)*y
+ }
+ result
+ }
> r<-myfun(1,2, TRUE)
> r
[1] 3
> r<-myfun(z=FALSE, y=0)
> r
[1] 0
> 
```

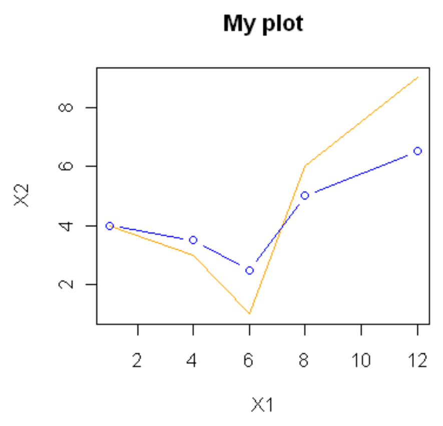
LINKÖPING UNIVERSITY

Graphical procedures

- Some common procedures:
  - plot(x,..) plots time series
  - plot(x,y) scatter plot
  - plot(x,y) followed by points(x,y) plots several scatterplots in one coordinate system
  - hist(x,..) plots a hitogram
  - persp(x,y,z,…) creates surface plots
  - cloud(formula,data..) creates 3D scatter plot



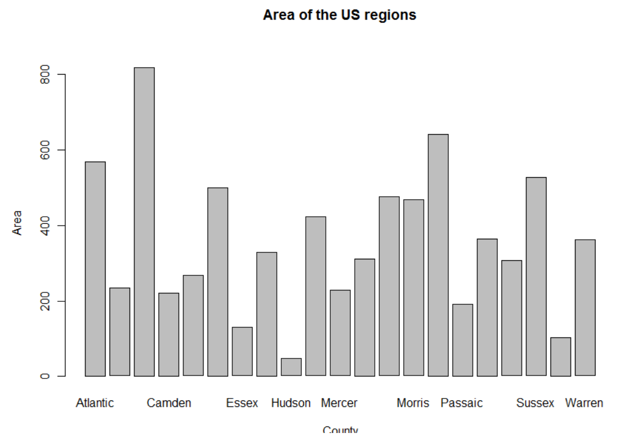
```
x<-c(1,4,7 8, 12);
y<-c(4,3,1,6,9);

plot(x,y, type="l", col="orange",
    main="My plot", xlab="X1", ylab="X2");
points(x, y/2+2, type="b", col="blue");
```



Graphical procedures

- Parameters:
  - Color: `col=”color”`
  - Title: `main=”text”`
  - Footnote: `sub=”text”`
  - X-axis label: `xlab=”text”`
  - Y-axis label: `ylab=”text”`

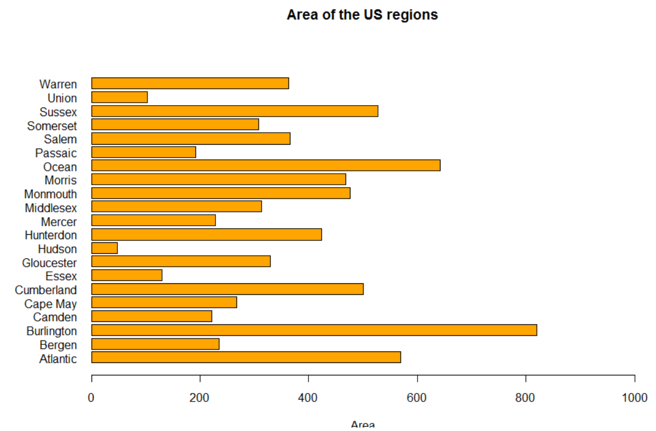
```
mydata<-read.csv2("Counties.csv");
barplot(mydata$Area,
names.arg=mydata$County, main=”Area of
the US regions”, xlab=”County”,
ylab=”Area”);
```



Graphical procedures

- Some parameters need to be specified either in the plotting function or inside `par(…)`
  - Symbol that is plotted: `pch=number`
  - Linetype: `lty=number`
  - Direction of axis values : `las=0, 1 or 2`
  - Margins (inch): `mai=c(bottom, left, top, right)`
  - Horizontal justification: `adj=between 0 and 1`

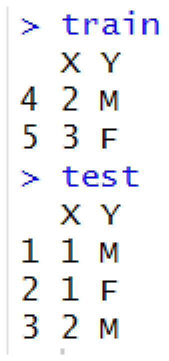
```
barplot(mydata$Area,
names.arg=mydata$County,
horiz=TRUE, las=1,
xlim=c(0,1000), col="orange",
main="Area of the US
regions", xlab="Area");
```



Useful code snippets

- Dividing a dataset into training/test sets:

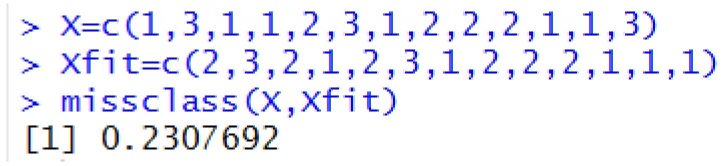
```r
data=data.frame(X=c(1,1,2,2,3), Y=c("M","F","M","M","F"))
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```



```
> train
  X Y
4 2 M
5 3 F
> test
  X Y
1 1 M
2 1 F
3 2 M
```

- Computing misclassification rate

```r
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
```



```
> X=c(1,3,1,1,2,3,1,2,2,2,1,1,3)
> Xfit=c(2,3,2,1,2,3,1,2,2,2,1,1,1)
> missclass(X,Xfit)
[1] 0.2307692
```

Additional resources

- R manuals of dplyr and tidyr packages (data manipulation) → many useful functions to apply on datasets
- R language manual https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf