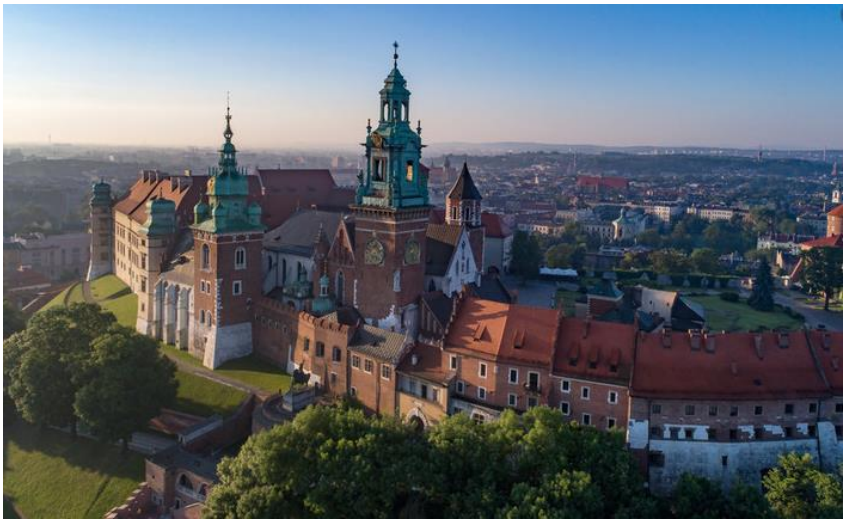
Introduction R language

International Doctoral Workshop

Paweł Lula, Cracow University of Economics, Poland
pawel.lula@uek.krakow.pl

1

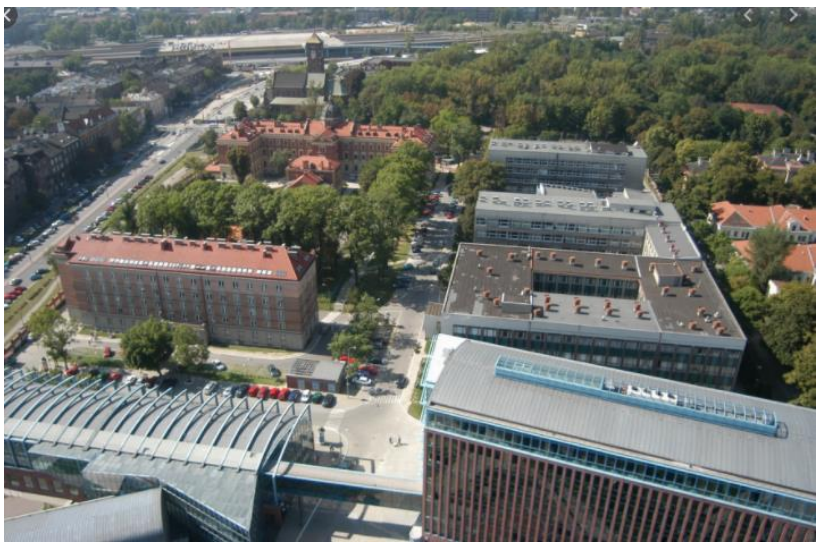
2



3

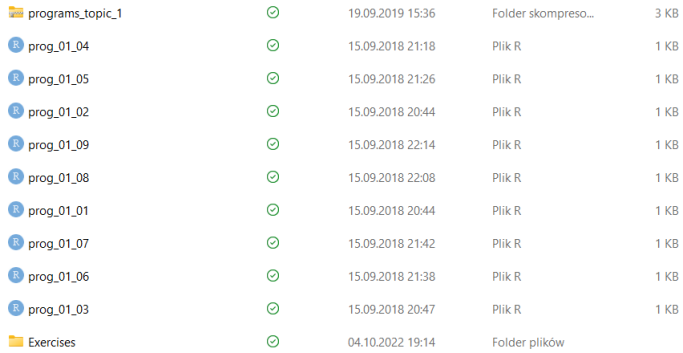
4

5



6

Materials



programs_topic_1	19.09.2019 15:36	Folder skompreso...	3 KB
prog_01_04	15.09.2018 21:18	Plik R	1 KB
prog_01_05	15.09.2018 21:26	Plik R	1 KB
prog_01_02	15.09.2018 20:44	Plik R	1 KB
prog_01_09	15.09.2018 22:14	Plik R	1 KB
prog_01_08	15.09.2018 22:08	Plik R	1 KB
prog_01_01	15.09.2018 20:44	Plik R	1 KB
prog_01_07	15.09.2018 21:42	Plik R	1 KB
prog_01_06	15.09.2018 21:38	Plik R	1 KB
prog_01_03	15.09.2018 20:47	Plik R	1 KB
Exercises	04.10.2022 19:14	Folder plików	

7

Outline

- Data mining concept
- Types of problems
- Types od data structures
- Fundamentals of R
- Vectors
- Sets
- Matrices
- Factors
- Data frames
- Lists
- Programs (scripts) in R language
- Control statements in R

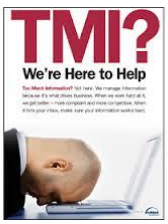
8

Data mining concept

9

Information overload

Information overload: a situation in which you get more information than you can deal with at one time and become tired and confused.

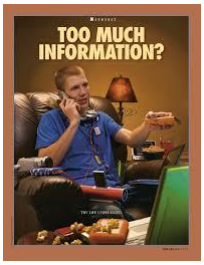


10

Flood of data

Computers have promised us a fountain of wisdom but delivered a flood of data.

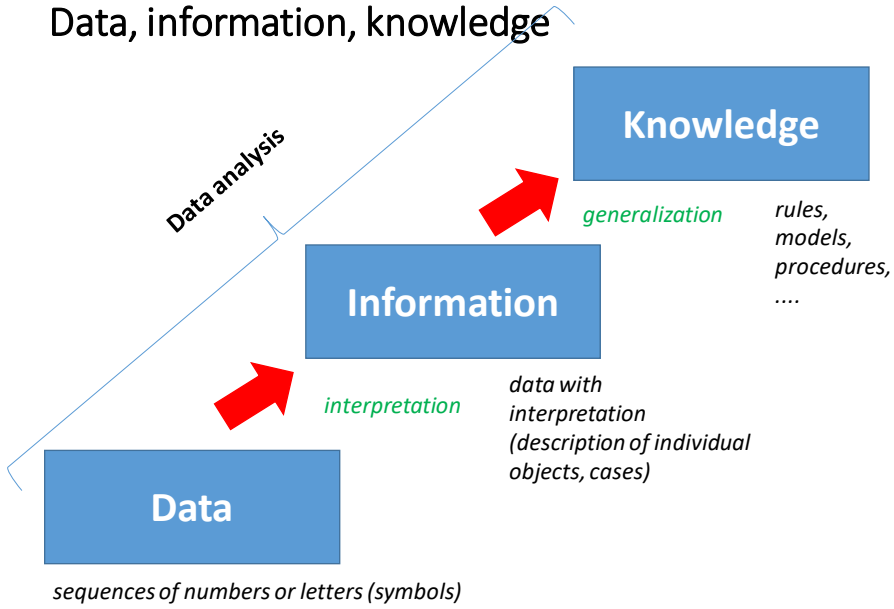
W. J. Frawley, G.Piatetsky-Shapiro, and C. J. Matheus, 1992



Data analysis

- **Data analysis** – process of data gathering, representation, cleaning, transforming and modelling realized to discover useful information, enrich knowledge and support decision-making processes.

Data, information, knowledge



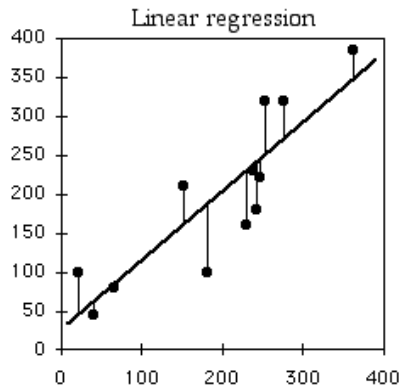
13

Three main approaches in data analysis

- descriptive statistics
  - calculation of statistical measures and coefficients to **present** and **describe** existing phenomena,
- exploratory data analysis (data mining)
  - analysis huge data sets to **discover** new (previously unknowns) patterns, regularities and relationships
- confirmatory data analysis
  - research process undertaken to **confirm** or **reject** previously formulated hypotheses.

14

Question about linear regression model



Exploratory or confirmatory approach?

15

Data mining definition

Data mining: the nontrivial extraction of implicit, previously unknown, and potentially useful information from data.

W. Frawley and G. Piatetsky-Shapiro and C. Matheus *Knowledge Discovery in Databases: An Overview*
AI Magazine, Fall 1992: pp. 213–228. ISSN 0738-4602.

16

Data mining definition

Data mining: the science of extracting useful information from large data sets or databases.

D. Hand, H. Mannila, P. Smyth
Principles of Data Mining. MIT Press, Cambridge
2001

17

Data mining definition

Data mining: the statistical and logical analysis of large sets of transaction data, looking for patterns that can aid decision making.

Ellen Monk, Bret Wagner (2006).
Concepts in Enterprise Resource Planning, Thomson Course Technology, Boston
2006

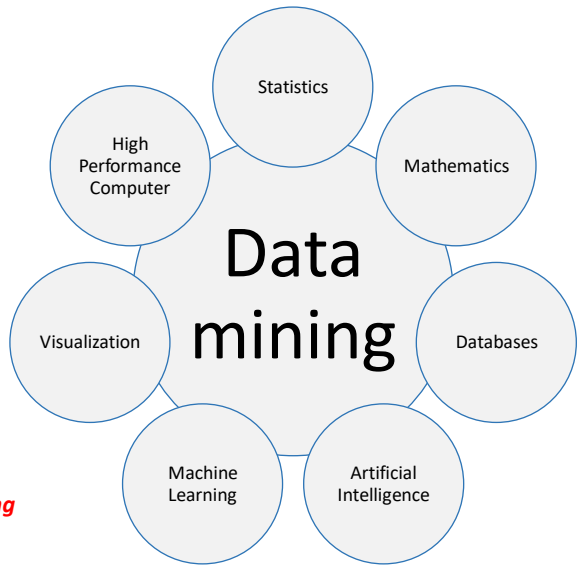
18

Key properties of data mining approach

- **data-based approach** (data-driven approach):
  - models are based on data, not on theory
  - huge databases and warehouses can be analyzed,
  - data mining methods belong to computational techniques
- outcomes: **easy-to-understand** and **easy-to-use**
- main field of application: **business**
- main goals: **decision support**

19

Data mining as an interdisciplinary field

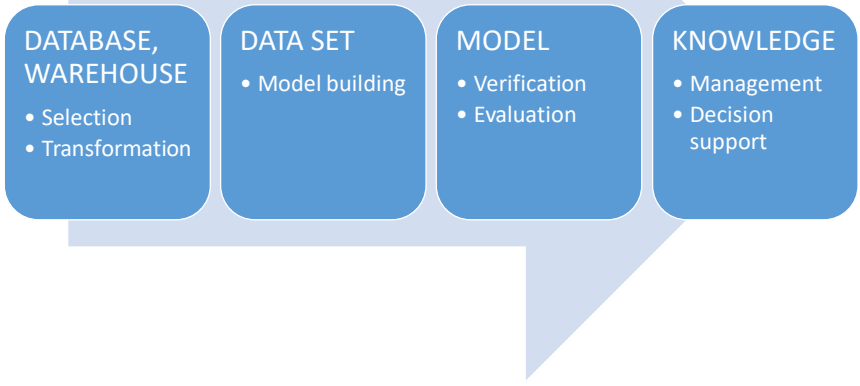


Pillars of Data Mining

20

Data mining process

Gain knowledge about the process!
Define the goal of analysis!

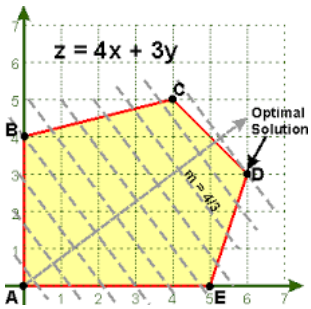


Types of problems

Types of problems

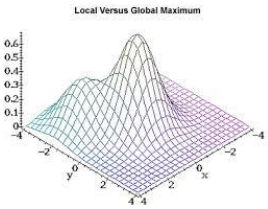
1. optimization problems,
2. regression
3. classification
4. cluster analysis
5. trends
6. network analysis
7. rules identification
8. sequence analysis
9. associate rules
10. text analysis

Types of problems: optimization problems



Main goal: finding optimal solution

the shortest path

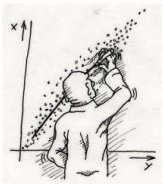


choice problem

Types of problems: regression

Regression analysis: techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables.

Independent variables → MODEL → *Dependent variables (numerical)*



Main goal: description of relationships between variables

25

Types of problems: regression

- *Location*
- *Size*
- *Number of bedrooms and bathrooms*
- *Age*
- *Garage size (number of cars)*
- *Central heating system*
- *Air conditioning system*
- *Swimming pool*

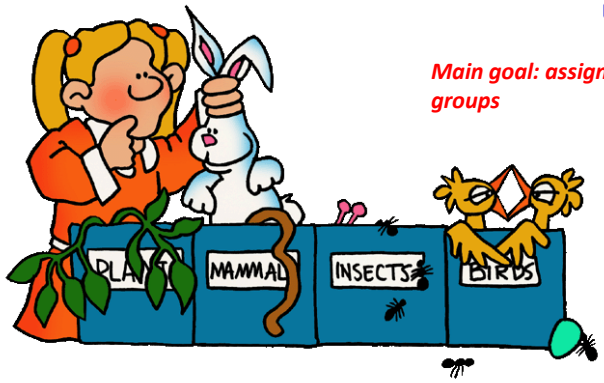
→ Regression model → ***Property value***

Main goal: description of relationships between variables

26

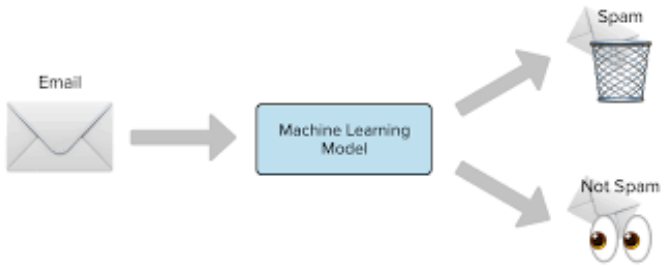
Types of problems: classification

Classification is the problem of identifying the group to which objects belong.



Main goal: assigning objects to groups

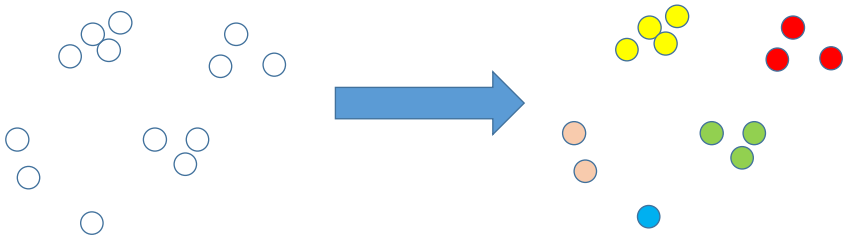
Spam detection as an example of classification



Types of problems: cluster analysis

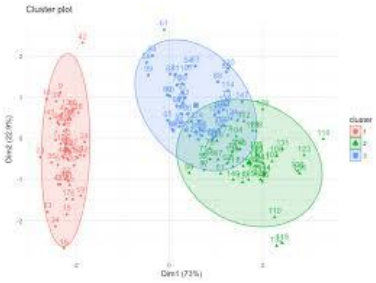
Main goal: analysis the structure of a given set of objects

- Cluster analysis or clustering is a process of grouping of a set of observations into subsets (called clusters).
- A cluster is a group of relatively homogeneous cases or observations. Objects in a cluster are similar to each other. They are also dissimilar to objects outside the cluster, particularly objects in other clusters.



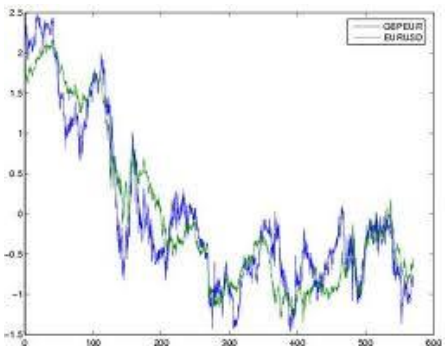
Iris clustering problem

Fig. 3.2 Three species of Iris flower



Types of problems: trends

x(t-1),
x(t-2) ,
...,
x(t-k)

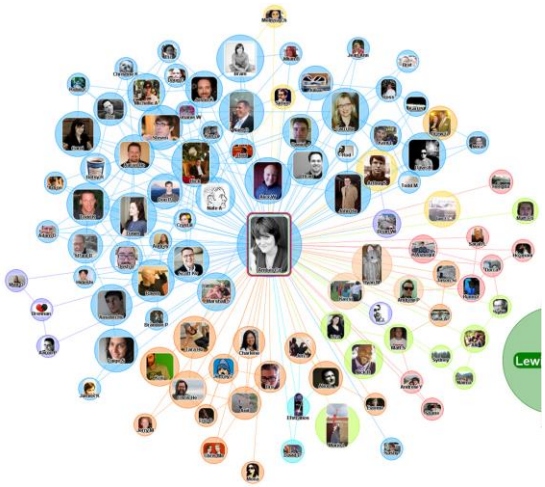


x(t)

Main goal: description of relationships between consecutive observations

31

Types of problems: network analysis

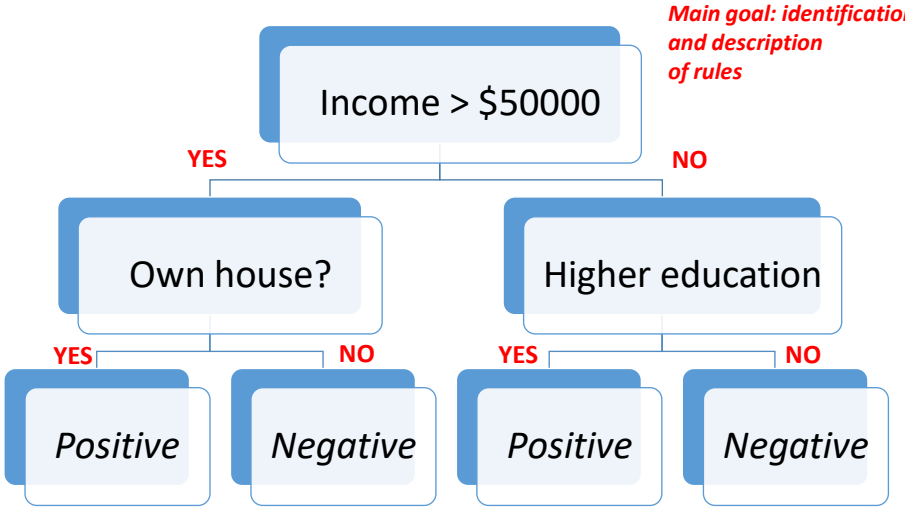


Main goal:
a) description of relationships between objects,
b) importance evaluation of objects and links,
c) analysis the structure of relationships.

32

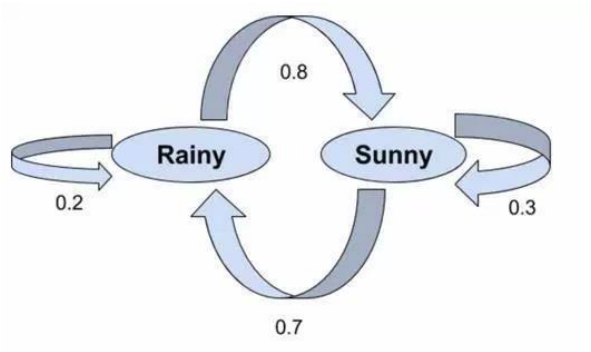
Types of problems: rules identification

Main goal: identification and description of rules



33

Types of problems: sequence analysis



Main goal: analysis, description and prediction of sequences of events

34

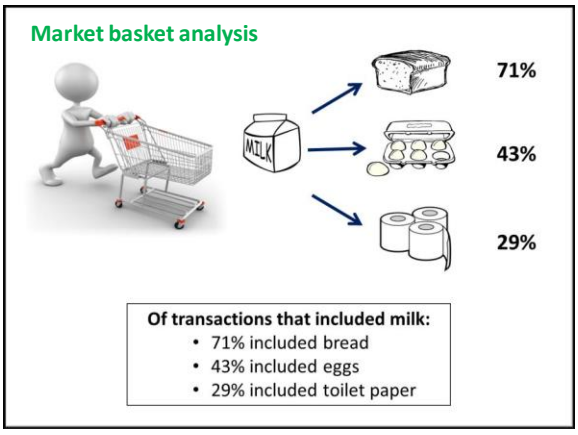
Types of problems: associate rules

Associate analysis: technique that allows to identify how the data items are associated each other

Main goal: analysis of co-occurrence of events

35

Types of problems: associate rules



Main goal: analysis of co-occurrence of events

36

Types of problems: text analysis

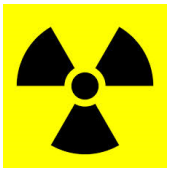
Main goal:

a) ***information retrieval,***
b) ***document classification,***
c) ***document clustering,***
d) ***identification of key-words,***
e) ***similarity evaluation.***

Types of data and the concept of similarity and distance

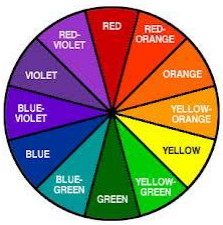
Datum and data

- Datum (*plural:* data):
  - something given,
  - a piece of information,
  - a single piece of information,
  - a fact or proposition used to draw a conclusion or make a decision.
- Data – a collection of facts.



Classification of data according to the type of values

- quantitative = numerical, number-based
  - discrete values (integer values),
  - continuous values (real values).
- qualitative = not numerical, word-based data
  - two-state data (logical data, True/False, Yes/No),
  - many-state data (color of eyes),
  - ordered values (small, medium, big)



Classification of data according to their structure

- Simple types of data (one object represents one value)
- Complex types of data (one objects represents many values)

Types of simple data

- numerical values,
  - integer,
  - real values,
- strings,
- logical values.

Types of complex data

- Matrices,
- Lists (sequence of elements),
- Records,
- Data frames (tables),
- Sets,
- Trees,
- Networks / Graphs,
- Texts (in natural languages).

Matrix

- a rectangular structure of elements,
- homogenous,
- elements are arranged in rows and columns,
- a position of the element is described by indices.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \leftarrow \text{second row}$$

$\uparrow$
third column

Objects representation in matrices

Features

Objects

	Column 1	Column 2	...	Column j	...	Column n
Row 1	a_{11}	a_{12}	...	a_{1j}	...	a_{1n}
Row 2	a_{21}	a_{22}	...	a_{2j}	...	a_{2n}
⋮	⋮	⋮		⋮		⋮
Row i	a_{i1}	a_{i2}	...	a_{ij}	...	a_{in}
⋮	⋮	⋮		⋮		⋮
Row m	a_{m1}	a_{m2}	...	a_{mj}	...	a_{mn}

Vector

- A matrix with one row (a 1 × m matrix) is called a row vector.
- A matrix with one column (an m × 1 matrix) is called a column vector.

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

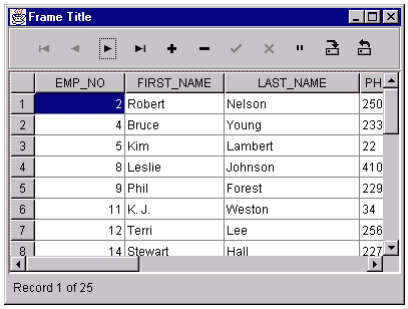
Record

- a complex structure with fields,
- fields store values,
- fields are identified by names,
- record is a heterogonous structure.

Data frame

- a table-based structure,
- row = record,
- column = field in the record,
- data frame = vector of records.

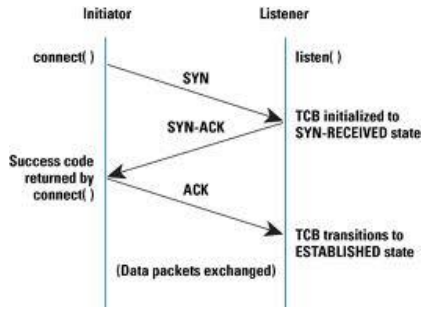
very popular
in data analysis problems!



List (sequence)

List – a **ordered** collection of:

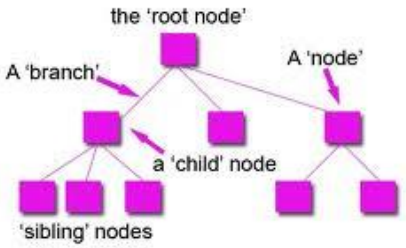
- values,
- events,
- tasks,
- goods,
- cities,
- ...



A sentence is a sequence of words.

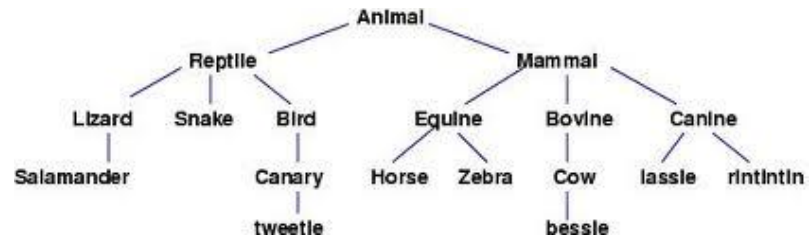
A word is a sequence of letters.

Tree



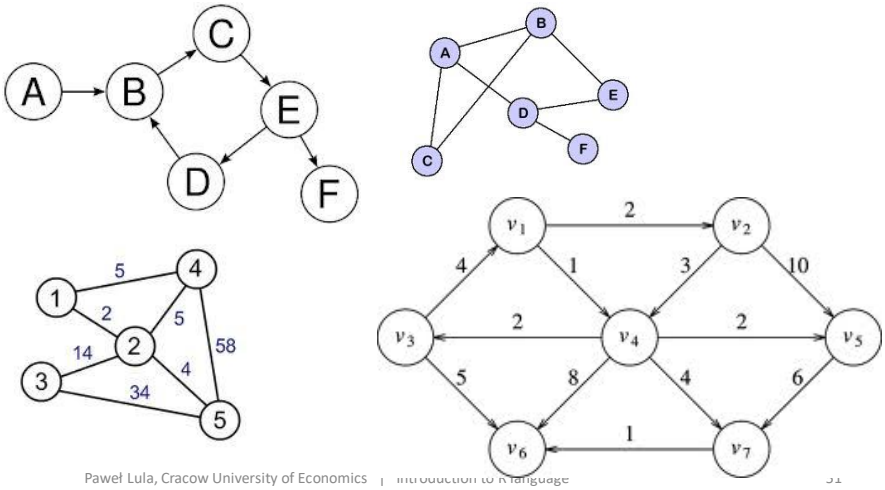
The best model for hierarchy representation

PARTS OF A TREE DATA STRUCTURE



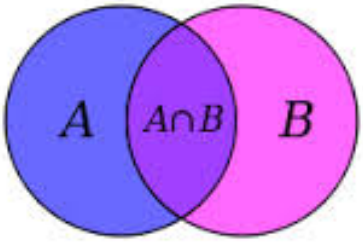
Graph / Network

- Graph – a set of nodes (vertices) connected by edges (links).



Set

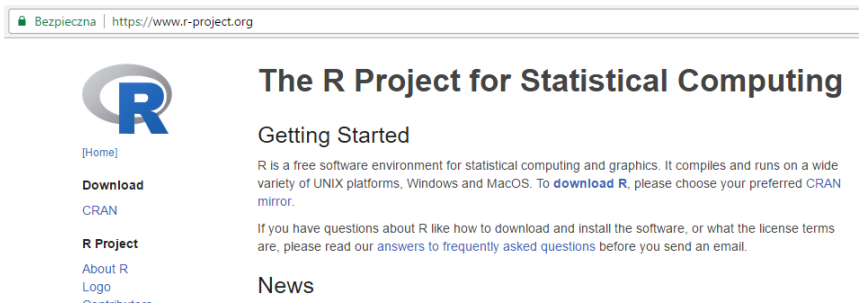
Set – a collection of objects without any particular order.



Fundamentals of R

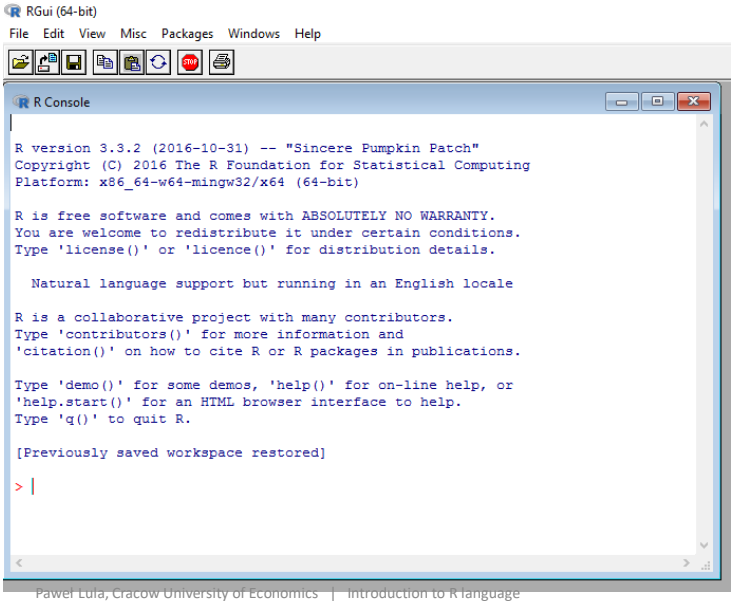
53

R - http://www.r-project.org/



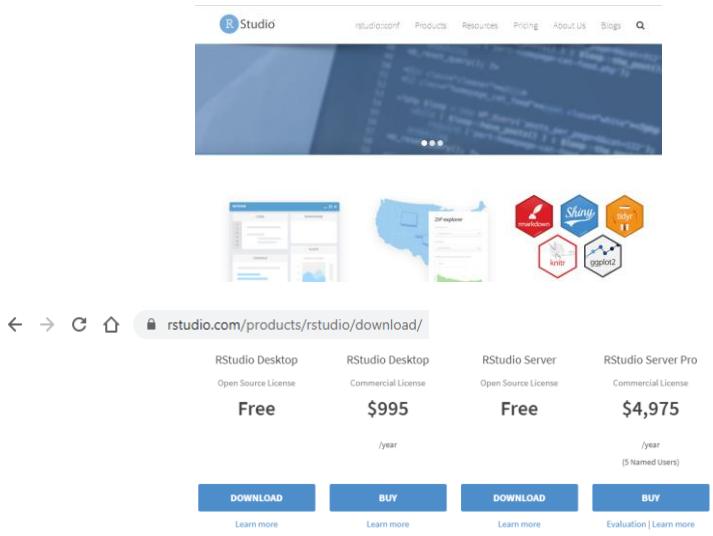
54

R Console



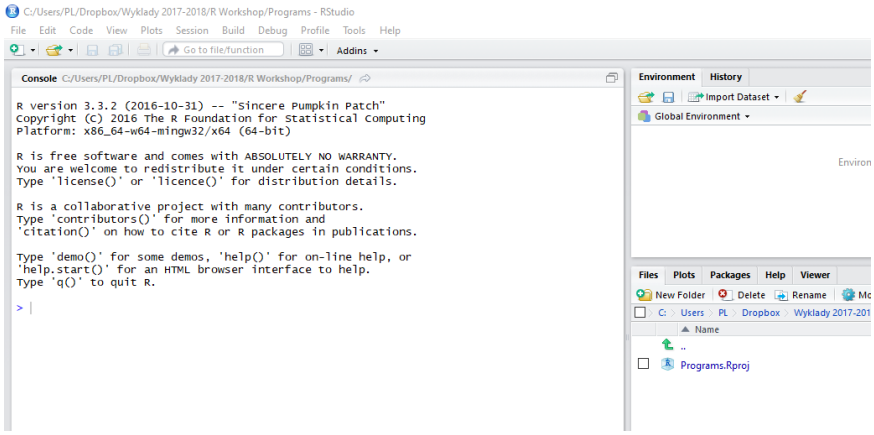
55

https://www.rstudio.com/



56

https://www.rstudio.com/



Simple calculations in R

```
> 2+7
[1] 9
> (1 + 3) / 8
[1] 0.5
> pi                    ← value π
[1] 3.141593
> abs(-2)
[1] 2
> 2^4                   ← exponentiation (2 to the power of 4)
[1] 16
> 2**4                  ← exponentiation (2 to the power of 4)
[1] 16
> 1/3
[1] 0.3333333
>
```

Simple calculations in R

```
> sin(30)                ← sine function, argument in radian
[1] -0.9880316
> sin(30*pi/180)
[1] 0.5
> log(2)                 ← natural logarithm
[1] 0.6931472
> log(25,5)              ← logarithm of 25 to base 5
[1] 2
> log(10)                ← natural logarithm
[1] 2.302585
> log(10,2)              ← logarithm of 10 to base 2
[1] 3.321928
```

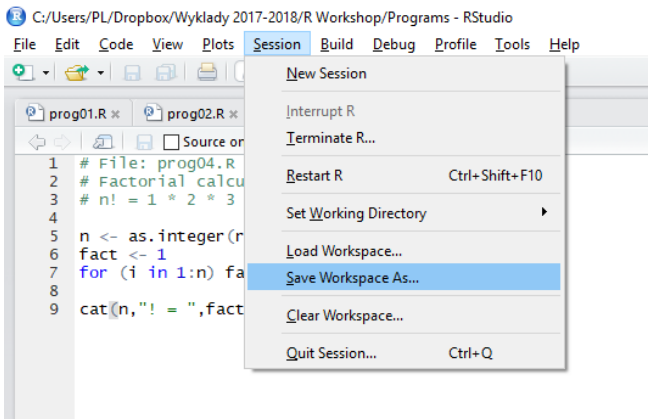

Variables and assignment statement

```
> x<-4          ← assignment statement
> x             ← displaying a object's value
[1] 4
> ls()          ← a list of objects stored in the workspace
[1] "x"
> rm(x)         ← removing of the x's object
> rm(list=ls())       ← removing all objects
> ls()
character(0)
>
```

Save and Load statement

```
> ls()
character(0)
> x <- 1
> y <- 2
> z <- 3
> ls()
[1] "x" "y" "z"
> save(x,y,file="data1.RData")
> rm(list=ls())
> ls()
character(0)
> load(file="data1.RData")
> ls()
[1] "x" "y"
> x
[1] 1
> y
[1] 2
>
```


Workspace saving and loading



Logical values and operators

```
> x<-TRUE
> y<-FALSE
> !x                ← NOT (logical negation)
[1] FALSE
> !y
[1] TRUE
> x|y               ← OR (logical disjunction)
[1] TRUE
> x&y               ← AND (logical conjunction)
[1] FALSE
> x&!y
[1] TRUE
> isTRUE(x)
[1] TRUE
> isTRUE(y)
[1] FALSE
>
```


Operators with respect to their priority

`:: :::`	access variables in a namespace
`$ @`	component / slot extraction
`[ [[`	indexing
`^`	exponentiation (right to left)
`- +`	unary minus and plus
`:`	sequence operator
`%any%`	special operators (including `%%` and `%/%`)
`* /`	multiply, divide
`+ -`	(binary) add, subtract
`< > <= >= == !=`	ordering and comparison
`!`	negation
`& &&`	and
`| ||`	or
`~`	as in formulae
`-> ->>`	rightwards assignment
`<- <<-`	assignment (right to left)
`=`	assignment (right to left)
`?`	help (unary and binary)

Relational operators

```
> x<-3
> y<-8
> x>y
[1] FALSE
> x<y
[1] TRUE
> x>=y
[1] FALSE
> x<=y
[1] TRUE
> x==y          ← is x is equal to y? (x = y means x <- y)
[1] FALSE
> x!=y          ← is x is not equal to y?
[1] TRUE
>
```


Strings

```
> x<-"abcdef"
> x
[1] "abcdef"
> cat(x)
abcdef
> x<-"abc\ndef"          ← \n = new line
> x
[1] "abc\ndef"
> cat(x)
abc
def
> nchar(x)              ← number of characters
[1] 7
> x<-"a\tb\tc\nd\te\tf"  ← \t = tab char
> x
[1] "a\tb\tc\nd\te\tf"
> cat(x)
a       b       c
d       e       f
>
```

Strings

```
> paste("abc","def","ghi")      <- concatenation
[1] "abc def ghi"
> paste("abc","def","ghi",sep="*")
[1] "abc*def*ghi"
> paste("abc","def","ghi",sep="")
[1] "abcdefghi"
>
```


Strings

```
> x<-"Analiza danych"
> tolower(x)
[1] "analiza danych"
> toupper(x)
[1] "ANALIZA DANYCH"
> x<-"Łódź"
> tolower(x)
[1] "łódź"
> toupper(x)
[1] "ŁÓDŹ"
> toupper("работа")
[1] "РАБОТА"
```

Type checking

```
> x<-"abc"
> y<-5.7
> z<-TRUE
> typeof(x)                ← type checking
[1] "character"
> typeof(y)
[1] "double"
> typeof(z)
[1] "logical"
> typeof(4+4)
[1] "double"
> typeof(4>3)
[1] "logical"
> typeof(4=3)              ← it is not equality checking!
Error: unexpected '=' in "typeof(4="
> typeof(4==3)
[1] "logical"
>
```


Type changing

```
> x<-"abc"
> y<-5.7
> z<-TRUE
> is.numeric(x)
[1] FALSE
> is.numeric(y)
[1] TRUE
> is.character(x)
[1] TRUE
> is.character(y)
[1] FALSE
> as.character(y)
[1] "5.7"
>
```

Type changing

```
> as.numeric(TRUE)
[1] 1
> as.numeric(FALSE)
[1] 0
> as.logical(-1)
[1] TRUE
> as.logical(0)
[1] FALSE
> as.logical(1)
[1] TRUE
> as.logical(10)
[1] TRUE
>
```


Missing values

```
> x<-NA          ← NA – not available
> x
[1] NA
> 2*x
[1] NA
> cat(x)
NA
> is.na(x)
[1] TRUE
> x<-4
> is.na(x)
[1] FALSE
>
```

Vectors

Vector

- Vector – homogenous structure with elements identified by indices

```
> x<-c(1,2,3,4)
> x
[1] 1 2 3 4
> x[1]
[1] 1
> x[4]
[1] 4
> x[5]
[1] NA
> length(x)
[1] 4
>
```

1	2	3	4

Vectors

```
> x<-1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x<-seq(1,12)
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> x<-seq(from=1,to=5)
> x
[1] 1 2 3 4 5
> x<-seq(to=1,from=5)
> x
[1] 5 4 3 2 1
> x<-seq(from=1,to=5,by=1.2)
> x
[1] 1.0 2.2 3.4 4.6
> x<-seq(from=1,by=1.2,length=12)
> x
 [1]  1.0  2.2  3.4  4.6  5.8  7.0  8.2  9.4 10.6 11.8 13.0 14.2
>
```


Vectors

```
> rep(c(1,2,3),times=3)
[1] 1 2 3 1 2 3 1 2 3
> rep(c(1,2,3),each=3)
[1] 1 1 1 2 2 2 3 3 3
> x<-seq(from=10,to=60,by=10)
> x
[1] 10 20 30 40 50 60
> x[2]
[1] 20
> x[c(1,2,4)]
[1] 10 20 40
> x[c(1,2,4,4,4)]
[1] 10 20 40 40 40
```

Vectors

```
> x
[1] 10 20 30 40 50 60
> x >=20
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> x[x >= 20]
[1] 20 30 40 50 60
> x[x >= 20 & x <= 40]
[1] 20 30 40
> x
[1] 10 20 30 40 50 60
> x[-c(1,4)]
[1] 20 30 50 60
```


Vectors

```
> x<-seq(from=10,to=60,by=10)
> length(x)
[1] 6
> 1:length(x)
[1] 1 2 3 4 5 6
> 1:length(x)%%2
[1] 1 0 1 0 1 0
> 1:length(x)%%2==0   ← remainder (modulo operator)
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE
> x[1:length(x)%%2==0]     ← only elements on even positions
[1] 20 40 60
>
```

Vectors

```
> vecOfWords <- c("one","two","three")
> vecOfWords
[1] "one"   "two"   "three"
> vecOfWords[2]
[1] "two"
>
```


79

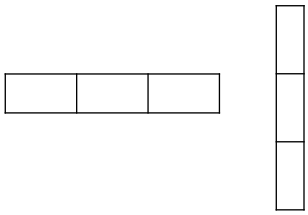
Vectors

```
> x<-c(1,2,3)
> y<-c(6,7,8)
> x+y
[1]  7  9 11
> x*y
[1]  6 14 24
```


80

Vectors

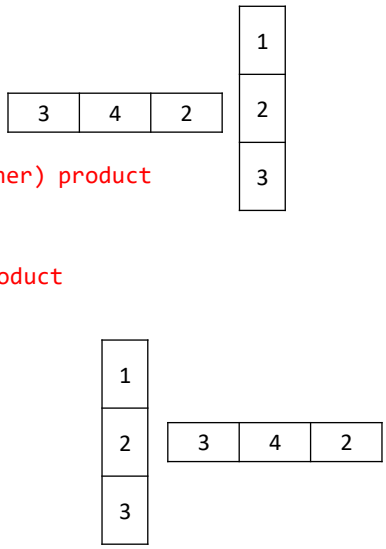
```
> x<-c(1,2,3)
> y<-(3,4,2)
> x
[1] 1 2 3
> y
[1] 3 4 2
> x*y
[1] 3 8 6
> x%*%y
     [,1]
[1,]   17
```



81

Vectors

```
> x<-c(1,2,3)
> y<-c(3,4,2)
> x%*%y      ← scalar (dot, inner) product
     [,1]
[1,]   17
> x%o%y            ← outer product
     [,1] [,2] [,3]
[1,]    3    4    2
[2,]    6    8    4
[3,]    9   12    6
> outer(x,y)  ← outer product
     [,1] [,2] [,3]
[1,]    3    4    2
[2,]    6    8    4
[3,]    9   12    6
```



82

Logical operations on vectors

```
> c(FALSE,FALSE,TRUE)&c(TRUE,FALSE,TRUE)
[1] FALSE FALSE  TRUE

> c(FALSE,FALSE,TRUE)&&c(TRUE,FALSE,TRUE)
[1] FALSE

> c(FALSE,FALSE,TRUE)|c(TRUE,FALSE,TRUE)
[1]  TRUE FALSE  TRUE

> c(FALSE,FALSE,TRUE)||c(TRUE,FALSE,TRUE)
[1] TRUE

> c(FALSE,FALSE,FALSE)|c(FALSE,FALSE,FALSE)
[1] FALSE FALSE FALSE

> c(FALSE,FALSE,FALSE)||c(FALSE,FALSE,FALSE)
[1] FALSE
>
```

← for every element

← for the whole vector

Functions for operating on vectors

- min
- max
- mean
- median
- sd
- var
- length
- sum
- prod
- diff
- *and others ...*

Sets

85

Sets

Sets are stored in vectors. All elements of a set must be unique!

```
> x<-c(1,4,12)
> y<-4:15
> union(x,y)        ← set union
 [1]  1  4 12  5  6  7  8  9 10 11 13 14 15
> intersect(x,y)  ← set intersection
[1]  4 12
> setdiff(x,y)      ← set difference
[1] 1
> setdiff(y,x)      ← set difference
 [1]  5  6  7  8  9 10 11 13 14 15
>
```


86

Sets

```
> x<-c(1,2,3,4)
> y<-c(3,4,1,2)
> x==y                    ← corresponding elements are compered
[1] FALSE FALSE FALSE FALSE
> setequal(x,y)           ← comparison of sets
[1] TRUE
> is.element(3,x)         ← membership checking
[1] TRUE
> is.element(8,x)
[1] FALSE
> 3 %in% x                ← membership checking
[1] TRUE
> 8 %in% x
[1] FALSE
>
```


Sets

```
> x<-c(2,5,3,1,6)
> y<-c(3,6,2,3,3)
> duplicated(x)           ← are unique?
[1] FALSE FALSE FALSE FALSE FALSE
> duplicated(y)
[1] FALSE FALSE FALSE  TRUE  TRUE
> which(duplicated(y))    ← which are not unique?
[1] 4 5
> unique(y)               ← set of unique values
[1] 3 6 2
>
```

Matrices

89

Matrices

- matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
- Parameters:
  - data – vector of elements
  - nrow – number of rows
  - ncol – number of columns
  - byrow – arrangement of elements in a matrix
  - dimnames – names of columns

90

Matrices

```
> x<-matrix(c(1,2,3,4,5,6,7,8,9),3,3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x<-matrix(c(1,2,3,4,5,6,7,8,9),3,3,byrow=TRUE)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
>
```


91

Matrices

```
> x<-matrix(c(1,2,3,4,5,6,7,8,9),3,3,byrow=TRUE)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> x[1,]
[1] 1 2 3
> x[1:2,]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>
```


92

Matrices

```
> x[,2]
[1] 2 5 8
> x[,2:3]
     [,1] [,2]
[1,]    2    3
[2,]    5    6
[3,]    8    9
> x[,c(1,3)]
     [,1] [,2]
[1,]    1    3
[2,]    4    6
[3,]    7    9
>
```


Matrices

```
a<-matrix(c(2,4,6,3),2,2)

> a
     [,1] [,2]
[1,]    2    6
[2,]    4    3

> b<-matrix(c(4,3,2,7),2,2)

> b
     [,1] [,2]
[1,]    4    2
[2,]    3    7
```

Matrices

```
> a-b
     [,1] [,2]
[1,]   -2    4
[2,]    1   -4
> a*b
     [,1] [,2]
[1,]    8   12
[2,]   12   21
> a %*% b
     [,1] [,2]
[1,]   26   46
[2,]   25   29
```

← multiplication of corresponding elements

← matrix multiplication

Matrices

```
> b %*% a
     [,1] [,2]
[1,]   16   30
[2,]   34   39
> t(a)
     [,1] [,2]
[1,]    2    4
[2,]    6    3
```

← matrix multiplication

← matrix transposition

Matrices

```
> solve(a)            ← matrix inversion
           [,1]       [,2]
[1,] -0.1666667  0.3333333
[2,]  0.2222222 -0.1111111

> a %*% solve(a)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
>
```


System of linear equations

System of linear equations:

$$3x_1 + x_2 + 4x_3 = 2$$
$$-x_1 + 6x_2 + x_3 = 1$$
$$x_1 + 2x_2 + 7x_3 = 2$$

```
> a <- matrix(c(3,1,4,-1,6,1,1,2,7),nrow=3,ncol=3,byrow=TRUE)
> a             ← matrix of equations coefficients
     [,1] [,2] [,3]
[1,]    3    1    4
[2,]   -1    6    1
[3,]    1    2    7
> b <- c(2,1,2) ← vector of constant terms
> solve(a,b)
[1] 0.3645833 0.1979167 0.1770833
>
```

Matrix determinant

```
> a
     [,1] [,2] [,3]
[1,]    3    1    4
[2,]   -1    6    1
[3,]    1    2    7
> det(a)              ← matrix determinant
[1] 96
>
```


99

Matrix eigenvalues and eigenvectors

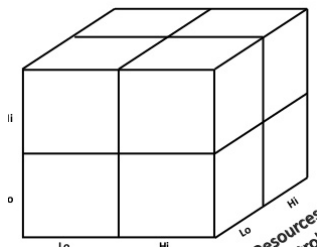
```
> a <- matrix(c(1,-2,-1,3,1,3,0,2,2),3,3)
> a
     [,1] [,2] [,3]
[1,]    1    3    0
[2,]   -2    1    2
[3,]   -1    3    2
> eigen(a)
$values
[1] 2 1 1

$vectors
           [,1]          [,2]          [,3]
[1,] -0.6359987 -7.071068e-01  7.071068e-01
[2,] -0.2119996 -1.148786e-09 -1.148786e-09
[3,] -0.7419985 -7.071068e-01  7.071068e-01
```


100

Multidimensional matrices

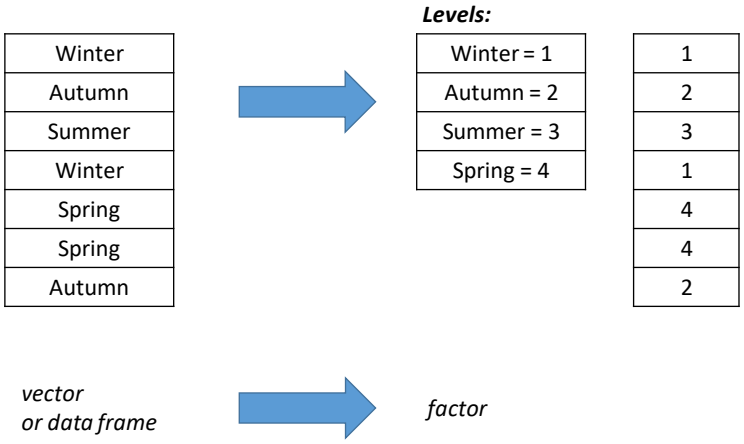
array(data = NA, dim = length(data), dimnames = NULL)

```
> x<-array(c(1,2,3,4,5,6,7,8),dim=c(2,2,2))
> x
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

>
```



Factors

Factors as a form of representation for nominal and ordinal variables



Factors (representation for nominal variables)

```
> seasons<-c("Winter","Autumn","Summer","Winter","Spring","Spring","Autumn")
> fseasons<-factor(seasons)
> fseasons
[1] Winter Autumn Summer Winter Spring Spring Autumn
Levels: Autumn Spring Summer Winter
> sort(fseasons)
[1] Autumn Autumn Spring Spring Summer Winter Winter
Levels: Autumn Spring Summer Winter
> levels(fseasons)[fseasons]
[1] "Winter" "Autumn" "Summer" "Winter" "Spring" "Spring" "Autumn"
>
```

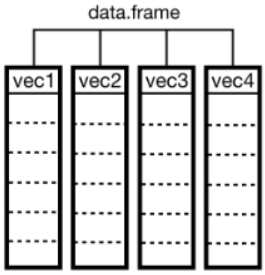
Factors (representation for ordinal variables)

```
> seasons<-c("Winter","Autumn","Summer","Winter","Spring","Spring","Autumn")
> fseasons<-factor(seasons,levels=c("Spring","Summer","Autumn","Winter"),ordered=TRUE)
> fseasons
[1] Winter Autumn Summer Winter Spring Spring Autumn
Levels: Spring < Summer < Autumn < Winter
> sort(fseasons)
[1] Spring Spring Summer Autumn Autumn Winter Winter
Levels: Spring < Summer < Autumn < Winter
> levels(fseasons)[fseasons]
[1] "Winter" "Autumn" "Summer" "Winter" "Spring" "Spring" "Autumn"
>
```


Data frames

Data frame objects

- A data frame – a matrix-like structure whose columns may be of differing types (numeric, logical, factor, character and so on).
- all values in the one column must be of the same type.



Creating of a data frame object

Data frame creating:

```
data.frame(vec1, vec2, ..., vecN, row.names = NULL,
check.rows = FALSE, check.names = TRUE,
stringsAsFactors = default.stringsAsFactors())
```

```
> a<-1:5
> b<-6:10
> c<-11:15
> df<-data.frame(a,b,c)
> df
  a  b  c
1 1  6 11
2 2  7 12
3 3  8 13
4 4  9 14
5 5 10 15
>
```

Data frame object

```
> df[2,3]
[1] 12
> df[1,]
  a b  c
1 1 6 11
> df[,1]
[1] 1 2 3 4 5
> df[c(1,4),2]
[1] 6 9
>
```


Creating of a data frame object

```
data.frame(..., row.names = NULL, check.rows = FALSE,
check.names = TRUE, stringsAsFactors =
default.stringsAsFactors())
```

defining names for rows

```
> df<-data.frame(a,b,row.names=c)
> df
   a  b
11 1  6
12 2  7
13 3  8
14 4  9
15 5 10
```

Creating of a data frame object

```
data.frame(..., row.names = NULL, check.rows = FALSE,
check.names = TRUE, stringsAsFactors =
default.stringsAsFactors())
```

defining names for columns

```
> df<-data.frame(kol1=a,kol2=b,kol3=c)
> df
  kol1 kol2 kol3
1    1    6   11
2    2    7   12
3    3    8   13
4    4    9   14
5    5   10   15
```


Reffering to columns of data frame object

```
> df=data.frame(kol1=a,kol2=b,kol3=c)
> df
  kol1 kol2 kol3
1    1    6   11
2    2    7   12
3    3    8   13
4    4    9   14
5    5   10   15
> df$kol1
[1] 1 2 3 4 5
> df$kol3
[1] 11 12 13 14 15
```

Data frames with strings

```
> d<-paste("s",1:5,sep="-")
> d
[1] "s-1" "s-2" "s-3" "s-4" "s-5"
> df<-data.frame(a,b,c,d)
> df
  a  b  c   d
1 1  6 11 s-1
2 2  7 12 s-2
3 3  8 13 s-3
4 4  9 14 s-4
5 5 10 15 s-5
> str(df)
'data.frame':     5 obs. of  4 variables:
 $ a: int  1 2 3 4 5
 $ b: int  6 7 8 9 10
 $ c: int  11 12 13 14 15
 $ d: Factor w/ 5 levels "s-1","s-2","s-3",..: 1 2 3 4 5
>
```


Data frames with strings

```
> df<-data.frame(a,b,c,d,stringsAsFactors = FALSE)
> df
  a  b  c   d
1 1  6 11 s-1
2 2  7 12 s-2
3 3  8 13 s-3
4 4  9 14 s-4
5 5 10 15 s-5
> str(df)
'data.frame':     5 obs. of  4 variables:
 $ a: int  1 2 3 4 5
 $ b: int  6 7 8 9 10
 $ c: int  11 12 13 14 15
 $ d: chr  "s-1" "s-2" "s-3" "s-4" ...
>
```

Lists

115

Lists

List = a ordered sequence of elements (heterogenous)

List creation

```
> ls <- list(5,4,3,2)
```

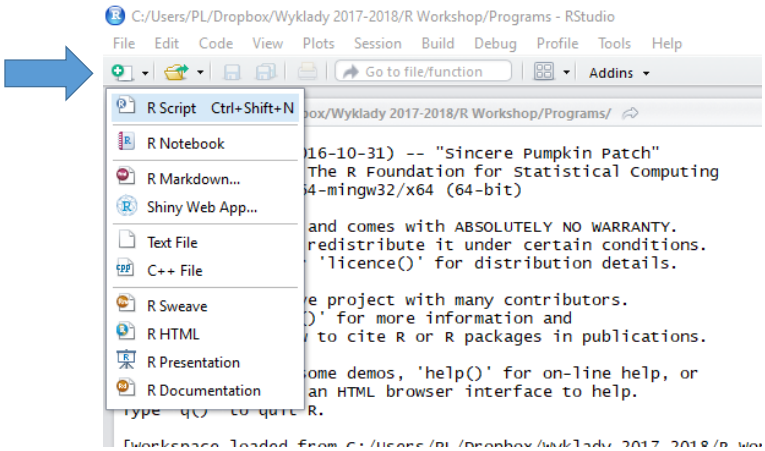
Two methods of reffering to list's elements

```
> ls[3]
[[1]]
[1] 3

> ls[[3]]
[1] 3

> ls[c(1,2)]
[[1]]
[1] 5

[[2]]
[1] 4

> ls[[c(1,2)]]
Error in ls[[c(1, 2)]] : subscript out of bounds
>
```

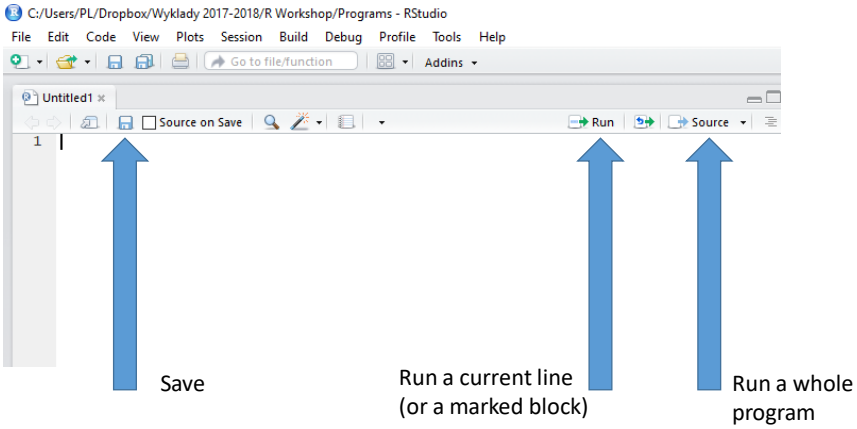

116

Programs (scripts) in R language

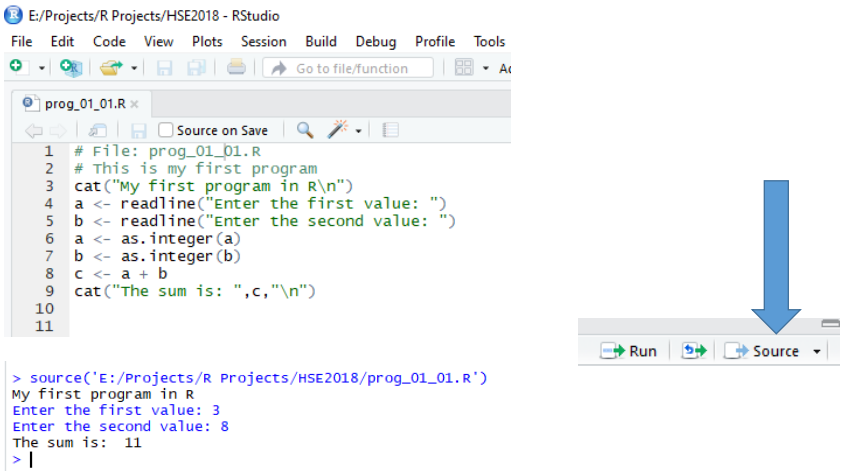
New script creation



New script creation



My first program



Functions – definition and calling

- Function definition:

```
object <- function(parameters) function-definition
```

- Function calling:

```
object(parameters)
```

- Showing function definition:

```
object
```

- Example:

```
> multiplyByTwo<-function(x) 2*x
> multiplyByTwo(44)
[1] 88
> multiplyByTwo
function(x) 2*x
>
```


Two forms of function definition

a) one-line definition

```
object <- function (parameters) expression
```

- b) multi-line definition:

```
object <- function(parameters) {
       function definition (one or more lines)
}
```

Two forms of return value defining

Returning the value of the last expression

prog_01_01.R | prog_01_02.R

Source on Save

```
# File: prog_01_02.R

f <- function(a, b) {
  cat("The function 'f' was called!\n")
  min(a,b)
}

cat("Function calling example\n")

x <- as.integer(readline("Enter the first value: "))
y <- as.integer(readline("Enter the second value: "))

z <- f(x,y)
cat("The minimum value is: ",z)

```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_02.R')
Function calling example
Enter the first value: 3
Enter the second value: 4
The function 'f' was called!
The minimum value is:  3
>
```


Two forms of return value defining

Using the return instruction for defining a return value

prog_01_01.R | prog_01_02.R | prog_01_03.R

Source on Save

```
# Program: prog_01_03.R

test <- function(x) {
  print("step 1")
  return (2*x)
  print("step 2")
}

```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_03.R')
> test(25)
[1] "step 1"
[1] 50
>
```

Actual and formal parameters

a) the order of formal and actual parameters must be the same

```
> s <- function(a,b) 3*a+b
> s(2,3)
[1] 9
```

b) we can change the order of parameters when we use parameters names

```
> s(a=2,b=3)
[1] 9
> s(b=2,a=3)
[1] 11
>
```


Default values for parameters

```
> fun <- function(a=1,b) a * b
> fun(3,4)
[1] 12
> fun(3)
Error in fun(3) : argument "b" is missing, with no default
> fun(b=3)
[1] 3
>
```

Usually we use parameters with default values at the end of the list of parameters

Optional parameters

- Optional = not obligatory

```
> s <- function(...) {
+ sum(...)
+ }
> s(3,4,5,5,5,9)
[1] 31
```

- Optional parameters must be used only at the end of parameter list

Control statements in R

Control statements

- if
- switch
- iterative instructions (loops):
  - for
  - while
  - repeat

If statement

```
if (condition) statement-1 else statement-2
```

or:

```
if (condition) {
       statement-1
} else {
       statement-2
}
```

If statement

Example:

```
> x<-5
> if (x==5) cat("yes\n") else cat("no\n")
yes
```


131

if statement in assigning statement

```
> x <- 6
> y <- if (x%%2==0) 0 else 1
> y
[1] 0
> y <- if (x%%2==0) "even number" else "odd number"
> y
[1] "even number"
>
```


132

Quadratic equation

prog_01_01.R | prog_01_02.R | prog_01_03.R | prog_01_04.R

Source on Save

```
# Program prog_01_04.R
# Quadratic equation

# Coefficient passing
a <- as.numeric(readline("Enter the coefficient a: "))
b <- as.numeric(readline("Enter the coefficient b: "))
c <- as.numeric(readline("Enter the coefficient c: "))
delta <- b^2 - 4*a*c
if (delta > 0) {
  # if delta greater then zero
  # two solutions exist
  x1 <- (-b - sqrt(delta))/(2*a)
  x2 <- (-b + sqrt(delta))/(2*a)
  cat("x1 = ",x1,"\n")
  cat("x2 = ",x2,"\n")
} else {
  if (delta ==0) {
    # if delta is equal to 0
    # one solution exists
    x0 <- -b/(2*a)
    cat("x0 = ",x0,"\n")
  }
  else {
    # if delta less then zero
    # no solution exists
    cat("No solutions!\n")
  }
}
```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_04.R')
Enter the coefficient a: 2
Enter the coefficient b: -7
Enter the coefficient c: 3
x1 =  0.5
x2 =  3
>
```


133

Switch statement

Statement switch:

```
switch (statement, list)
```

Examples:

```
> switch(2,"red","green","blue")
[1] "green"

> switch(1,"red","green","blue")
[1] "red"

> switch(3,"a","b","c","d","e")
[1] "c"

> switch(2,5,6,7,8,9)
[1] 6

> switch(8,5,6,7,8,9)
NULL
```


134

Quadratic equation

$$\operatorname{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

prog_01_01.R | prog_01_02.R | prog_01_03.R | prog_01_04.R | prog_01_05.R

Source on Save

```
1  # Program prog_01_05.R
2  # Quadratic equation
3
4  # Coefficient passing
5  a <- as.numeric(readline("Enter the coefficient a: "))
6  b <- as.numeric(readline("Enter the coefficient b: "))
7  c <- as.numeric(readline("Enter the coefficient c: "))
8  delta <- b^2 - 4*a*c
9  x <- switch(sign(delta)+2,
10             "No solutions!",
11             -b/(2*a),
12             c((-b-sqrt(delta))/(2*a),(-b+sqrt(delta))/(2*a)))
13 print(x)
14
```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_05.R')
Enter the coefficient a: 2
Enter the coefficient b: -7
Enter the coefficient c: 3
[1] 0.5 3.0
>
```


For statement

For statement:

```
for (variable in list-or-vector) statement
```

or:

```
for (variable in list-or-vector) {
      sequence of statements
}
```

For statement

```
for (variable in list-or-vector) {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

137

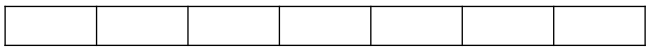
For statement

```
for (variable in list-or-vector) {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

number of repetitions (iterations) is equal to the number of elements of a list or a vector



138

For statement

```
for (variable in list-or-vector) {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

number of repetitions (iterations) is equal to the number of elements of a list or a vector

during consecutive iterations succeeding elements of a list/vector are assigned to a control variable of a for statement

4	2	3	7	5	5	1

For statement

```
for (variable in list-or-vector)
      statement
```

Examples:

```
> for(i in 1:5) cat(i)
12345> for (i in 1:5) cat(i,"\n")
1
2
3
4
5
>
```

For statement

```
> s<-0
> for (i in 1:5) s <- s + i
> s
[1] 15
> f <- 1
> for (i in 1:6) f <- f * i
> f
[1] 720
>
```

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 3 \cdot 2 \cdot 1$$

For statement

prog_01_06.R

Source on Save

```
1  # File: prog_01_06.R
2  # Factorial calculation
3  # n! = 1 * 2 * 3 * ... * n
4
5  n <- as.integer(readline("Enter n: "))
6  fact <- 1
7  for (i in 1:n) fact <- fact * i
8
9  cat(n,"! = ",fact)
```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_06.R')
Enter n: 6
6 ! =  720
>
```

For statement

prog_01_06.R | prog_01_07.R

Source on Save

```
1  # File: prog_01_07.R
2  # Factorial calculation
3  # n! = 1 * 2 * 3 * ... * n
4
5
6  factorial<-function(x){
7    f<-1
8    for(i in 1:x) f <- f*i
9    return(f)
10 }
11
12 n <- as.integer(readline("Enter a number: "))
13 cat(n,"! =",factorial(n),"\n")
14
15
```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_07.R')
Enter a number: 6
6 ! = 720
>
```


For statement

```
> l <- list("Spring","Summer","Autumn","Winter")
> for (i in l) cat (i,"\n")
Spring
Summer
Autumn
Winter
>
```

While statement

```
While statement:

while (condition) statement

or:

while (condition) {
       sequence of statements
}
```


145

While statement

```
while (condition) {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

146

While statement

```
while (condition) {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

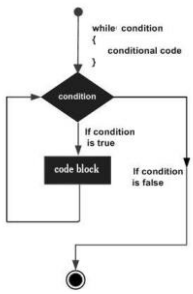
while the condition is fulfilled

147

While statement

```
while (condition) {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times



IMPORTANT:
statement execution should have an impact on the condition!
Otherwise the while statement will not finish (infinite loop)!

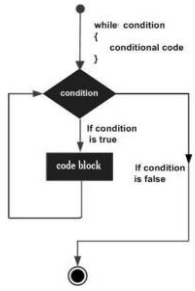
how many times?

while the condition is fulfilled

148

While statement

```
> x <- 5
> while(x > 0) {
+ cat(x,"\n")
+ x <- x - 1
+ }
5
4
3
2
1
```



While statement

prog_01_06.R | prog_01_07.R | prog_01_08.R

Source on Save

```
1  # File: prog_01_08.R
2  # Factorial calculation
3  # n! = 1 * 2 * 3 * ... * n
4
5
6  factorial<-function(x){
7    f <- 1
8    while (x > 0) {
9      f <- f * x
10     x <- x - 1
11   }
12   return(f)
13 }
14
15 n <- as.integer(readline("Enter a number: "))
16 cat(n,"! =",factorial(n),"\n")
17
```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_08.R')
Enter a number: 6
6 ! = 720
>
```

Repeat statement

Repeat statement:

```
repeat statement
```

or:

```
repeat {
        sequence of statements
}
```


151

Repeat statement

```
repeat {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

152

Repeat statement

```
repeat {
        statement-1
        statement-2
        statement-3
        ...
        statement-N
}
```

this part will be repeated many times

how many times?

until "break" statement is executed

Repeat statement

prog_01_06.R | prog_01_07.R | prog_01_08.R | prog_01_09.R

Source on Save

```
1  # File: prog_01_09.R
2  # Factorial calculation
3  # n! = 1 * 2 * 3 * ... * n
4
5
6  factorial<-function(x) {
7    f <- 1
8    repeat {
9      f <- f * x
10     x <- x - 1
11     if (x == 0) break
12   }
13   return(f)
14 }
15
16 n <- as.integer(readline("Enter a number: "))
17 cat(n,"! =",factorial(n),"\n")
18
```

```
> source('E:/Projects/R Projects/HSE2018/prog_01_09.R')
Enter a number: 6
6 ! = 720
>
```

Repeat statement

```
> repeat cat("test","\n")
```
← infinite loop !!!

```
> x <- 0
> repeat {
+ cat("test","\n")
+ x <- x + 1
+ if (x==5) break
+ }
test
test
test
test
test
>
```


155

Next statement

next = starting a new iteration of a loop

```
> for(i in 1:10) {
+ if (i%%2 == 0) next
+ cat(i,"\n")
+ }
1
3
5
7
9
>
```


156

Thank you for your attention!

Paweł Lula, Cracow University of Economics, Poland
pawel.lula@uek.krakow.pl

157