***Въведение в Статистиката с R - I***

1. **Въведение**
2. **Инсталиране**
   1. **Инсталиране на R**

R е безплатен софтуер за статистически изчисления и графики, open source и е наличен за Mac, Windows и Linux.

R може да бъде свален от [*www.r-project.org*](http://www.r-project.org).

* Download - CRAN(Comprehensive R Archive Network)
* Избирате Mirror
* Имате възможност да свалите R за Linux, (Mac) OS X, Windows
* (Windows) base – ако инсталираме R за първи път
* Download
* Install
  1. **Инсталиране на R Studio**

R Studio е професионално, мощно и продуктивно IDE за R. То е безплатно и е с отворен код, налично за Windows, Mac и Linux.

След инсталирането на R, може да бъде инсталирано R Studio.

R Studio може да бъде свален от *www.rstudio.com* .

* Download R Studio
* Имате възможност да избирате дали да изтеглите R Studio за десктоп или за сървър. За нашите цели ще използваме – R Studio за десктоп
* Download R Studio Desktop
* Избирате си операционната система на която искате да го инсталирате, например – R Studio 0.99.893 – Windows Vista/7/8/10
* Install

1. **Запознаване с R**
2. **Започване на R сесия**

R предлага много опции за зареждане на външни данни, включително от Excel, Minitab and SPSS сега обаче ще се научим да работим с конзолата в R.

След като стартирате R се появява конзола, която чака да въведете данни или команди. Символът (>) ви подсеща, че трабва да въвеждате след него.

**1.1 Можете да въведете числа или елементарни логически операции.**

> 3 + 2   
[1] 6

**1.2 Присвояване стойности на променливи**

С помощта на оператора "=" се присвояват стойности на променливи. Печатате името на променливата, после знакът "=" и след това печатате нейната стойност. Може да се използва и още един оператор за присвояване и той е "<-".

> x = 1   
> x   
[1] 1

> x <- 1   
> x   
[1] 1

**1.3 Принтиране на стойностите на променливи**

Можем да принтираме стойността на променлива с помощта на вградената функция "print()" или само чрез написване на името на променливата.

> print(x)   
[1] 1

> x   
[1] 1

**1.4 Различава големи и малки букви**

Трябва да споменем също, че R прави разлика между малки и главни букви.

> X  
Error: object 'X' not found

**1.5 Вградени функции**

Вече разгледахме една вградена функция “print()”, нека сега разгледаме още основни някой основни функции. R разполага с неизчерпаем ресурс от вградени функции. Те се извикват чрез тяхното име, следвано от параметрите на функцията заградени в скобки. Следната функция се използва за съднаване на вектор ред

> c(5, 2, 3)   
[1] 5 2 3

По нататък ще се запознаем с още много функции.

**Коментари**

Целият текст след знакът "#" в рамките на един ред се счита за коментар.

> 2\*3 + 1      # това е коментар   
[1] 7

**Добавяне на пакети**

Понякога имаме нужда от допълнителни функции, които ги няма в основната библиотека на R. За да инсталирате допълнителен пакет трябва да извикате функцията ***install.packages*** и да следвате инструкциите след това.

> install.packages(име на пакета)

**Четене на множества от данни с *library* и *data***

За да използваме данни или функция първо трябва да заредим пакета. За да разгледаме пакетите, които имаме на разположение можем да използваме командата library(). За да разгледаме множествата от данни можем да използваме командата data() без каквито и да е аргументи. За да разгледаме множествата от данни в даден пакет можем да използваме командата data, в която като параметър да зададем името на пакета. За да прочетем данните от дадено множество пишем името на множеството от данни като параметър на функцията data.

По-долу сме заредили пакета "ts", разгледали сме множествата от данни в него и сме заредили данните "lynx". Естествено преди това пакетът трябва да се инсталира.

> library("ts")

> data(package=ts).

> data("lynx")

Данните, които използваме са предоставени от CSI math department (http://www.math.csi.cuny.edu/Statistics/R/simpleR)

**Помощ**

R има обширна help документация. Например въвеждането на ***?c*** или на ***help(c)*** след стрелката извежда документацията за функцията ***c*** в R. Моля опитайте

> help(c)

Ако не сте сигурни за името на функцията, която търсите, можете да направите търсене по част от име на функция чрез функцията ***apropos***.

> apropos("sum")   
[1] ".\_\_C\_\_summary.table" ".\_\_C\_\_summaryDefault" ".\_\_T\_\_resume:rugarch"

[4] ".\_\_T\_\_Summary:base" ".colSums" ".rowSums"

….

Ако и това не помогне можете да търсите по ключова дума на интернет адрес [http://www.rseek.org](http://www.rseek.org/).

1. **Основни типове данни**
   1. **Numeric**

**Това са числа та с десетична точка.** По подразбиране данните при пресмятанията в R са такива. Ако присвоим десетично число на променлива, тя автоматично става от този тип.

> x = 10.5       # присвоява десетична стойност   
> x              # отпечатва стойността на x   
[1] 10.5   
> class(x)       # отпечатва името на класа x   
[1] "numeric"

Дори **да присвоим целочислена стойност** на променливата k, то нейният стил също ще е numeric.

> k = 1   
> k              # отпечатва стойността на k   
[1] 1   
> class(k)       # отпечатва името на класа на k   
[1] "numeric"

**Проверка дали е Numeric**

Фактът, че k не е integer може да бъде потвърден с използването на функцията ***is.integer***.

> is.integer(k)  # проверява дали k е integer?   
[1] FALSE

* 1. **Integer**

**Дефиниране на променлива от този тип**

За да създадем променлива от тип **integer**в R,извикваме функцията ***as.integer***.

> y = as.integer(3)   
> y              # отпечатва стойността на y   
[1] 3   
> class(y)       # отпечатва името на класа на y   
[1] "integer"

**Проверка дали е Integer**

Можем да проверим дали променливата е от тип integer като използваме функцията ***is.integer***.

> is.integer(y)  # y integer ли е?   
[1] TRUE

**Транформиране в integer**

Можем да трансформираме десетично число в integer като махнем цифрите след десетичната точка, посредством функцията ***as.integer***.

> as.integer(5.09)       
[1] 5

> as.integer(5.69)

[1] 5

Можем да постигнем и същото с низ от десетично число по абсолютно същия начин

> as.integer("12.35")

[1] 12

Ако опитаме да направим същото със стринг, който не е десетично число, че получим NA – Not available и съобщение за грешка

> as.integer("MPP")

[1] NA   
Warning message:   
NAs introduced by coercion

* 1. **Complex**

Числата от тип **complex**в R се дефинират чрез имагенерното число *i*.

> z = 5 + 6i     # присвоява на z комплексна стойност 5 + 6i  
> z                 
[1] 5+6i   
> class(z)       # отпечатва името на класа на z   
[1] "complex"

Следващият израз дава неопределеност и съобщение за грешка когато −1 не е от тип complex.

> sqrt(−1)          
[1] NaN   
Warning message:   
In sqrt(−1) : NaNs produced

Вместо това ние би трябвало да напишем

> sqrt(−1+0i)    # квадратен корен от −1+0i   
[1] 0+1i

или да дефинираме −1 като complex-на стойност и тогава да определим квадратен корен от −1+0i.

> sqrt(as.complex(−1))   
[1] 0+1i

* 1. **Logical**

Логическите стойности често се създават като резултат от сравняване между променливи.

> x = 1; y = 2      
> z = x > y      # Присвоява на z логическата стойност на израза x > y?   
> z              # отпечатва стойността на z   
[1] FALSE   
> class(z)       # отпечатва името на класа на z   
[1] "logical"

Стандартните логически операции са "&" (и), "|" (или), и "!" (отрицание).

> u = TRUE; v = FALSE   
> u & v          # u и v   
[1] FALSE   
> u | v          # u или v   
[1] TRUE   
> !u             # отрицанието на u   
[1] FALSE

Повече информация за логическите оператори в R може да бъде намерена в помощната документация.

> help("&")

Често е полезно да извършим аритметични операции с логически стойности, TRUE има стайност 1, докато FALSE има стойност 0.

> as.integer(TRUE)    # връща числовата стойност на TRUE   
[1] 1   
> as.integer(FALSE)   # връща числовата стойност на FALSE   
[1] 0

* 1. **Character**

Стринговете в R се представят чрез обекти от тип **character.**

Превръщаме един обект в character чрез функцията as.***character()***:

> x = as.character(3.14)   
> x              # отпечатва стрингът от тип character   
[1] "3.14"   
> class(x)       # отпечатва името на класа на x   
[1] "character"

Два обекта могат да бъдат долепени с помощта на функцията ***paste***.

> l = "love"; m ="music"   
> paste(l, m)   
[1] "love music"

Често е по-удобно да се създаде лесен за четене стринг с помощта на функцията ***sprintf***, която има синтаксис както в езика C.

> sprintf("%s is %d years old", "Moni", 22)   
[1] "Moni is 22 years old"

Функцията ***substr*** извлича подстринг от символи, като започва от началната и свършва в спиращата позиция, включително. Интервалите също се броят.

> substr("Moni loves her students.", start=3, stop=12)   
[1] "ni loves h"

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

> sub("little", "big", "Mary has a little lamb.")   
[1] "Mary has a big lamb."

Повече информация за операциите със стрингове може да бъде намерена в помощната документация в R. Например

> help("sub")

1. **Вектори**

Вектор е наредена съвкупност от елементи от един и същ основен тип. Членовете на вектора се наричат компоненти.

* 1. **Създаване на вектори**

Вече видяхме, че с помощта на функцията с може да се създаде вектор ред

Той може да има три числови стойности(в този случай можем да ги наречем координати) 1, 5 и 9.

> c(1, 5, 9)   
[1] 1 5 9

Може да имаме вектор, чиито компоненти са логически стойности.

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

Може да имаме вектор, чиито компоненти са символни стойности.

> c("MM", "oo", "nn", "ii")   
[1] "MM" "oo" "nn" "ii"

Можем да създадем вектор чрез задаване на координати, които образуват аритметична прогресия, чрез функцията ***seq***.

seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),

length.out = NULL, ...)

seq(17)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

seq(0, 1, length.out = 11) # достига последното

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

seq(0, 1.05, length.out = 11) # не достига последното

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

seq(1, 9, by = 2) # достига последното

[1] 1 3 5 7 9

seq(1, 9.1, by = 2) # не достига последното

[1] 1 3 5 7 9

Можем да създадем вектор чрез задаване на повтарящи се координати, чрез функцията ***rep***.

rep(x, each = How many times to repeat consecutively?,

times = How many times to repeat the previous?, len = max length)

each и times по подразбиране са единици

> rep(1:4, 2) # повторенията са последователно два пъти.

[1] 1 2 3 4 1 2 3 4

> rep(1:4, each = 2) # повторенията са всяко по два пъти.

[1] 1 1 2 2 3 3 4 4

> rep(1:4, c(2,2,2,2)) # същото както предното.

[1] 1 1 2 2 3 3 4 4

> rep(1:4, c(2,1,2,1)) # повторенията са по различен брой пъти и този брой е зададен чрез компоненти на вектор.

[1] 1 1 2 3 3 4

> rep(1:4, each = 2, len = 4) # повторенията са всяко по два пъти но докато станат не повече от 4 броя.

[1] 1 1 2 2

> rep(1:4, each = 2, len = 10) # повторенията са всяко по два пъти но докато станат не повече от 10 броя.

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

> rep(1:4, each = 2, times = 3) # Всяко число от 1 до 4 се повтаря по два пъти и имаме три пълни повторения на предходното

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

* 1. **Дължина на вектор**

Броят на компонентите на вектора може да бъде определен с помощта на функцията ***length***.

> length(c("MM", "oo", "nn", "ii"))   
[1] 4

* 1. **Трансформиране на вектори**

Два и повече вектори могат да бъдат долепени с помощта на функцията ***c***. В този случай числовите стойности се трансформират в стрингове за да се постигне един и същ тип на елементите на новия вектор.

> n = c(1, 5, 9)   
> s = c("MM", "oo", "nn", "ii")

> c(n, s)   
[1] "1"  "5"  "9"  "MM" "oo" "nn" "ii"

* 1. **Аритметични операции с вектори**

Аритметичните операции с вектори се извършват „покоординатно“. Например ако имаме

> a = c(1, 2, 3, 4)   
> b = c(5, 6, 7, 8)

то

> 5 \* a   
[1]  5 10 15 20

> a + b   
[1]  6  8  10 12

> a - b   
[1]  -4  -4  -4 -4   
> a \* b   
[1]  5  12 21 32   
> a / b   
[1] 0.2000000 0.3333333 0.4285714 0.5000000

С помощта на функциите ***options*** или ***round*** можем да определим до колко знака след десетичната запетая да се закръгля резултата

options(digits=1)

> a/b

[1] 0.2 0.3 0.4 0.5

> round(a/b,2)

[1] 0.20 0.33 0.43 0.50

Функцията ***оptions*** определя начинът на закръгляне през цялата ви работна сесия.

Циклично правило

Ако два вектора не са с еднаква дължина при аритметичните операции по-късият от тях ще бъде направен, чрез циклично попълване на празните елементи, равен по дължина на по-дългия. В същото време ще получите и съобщение за да внимавате

> a = c(1, 2, 3)   
> b = c(1, 2, 3, 4, 5, 6, 7)   
> a + b   
[1] 2 4 6 5 7 9 8

Warning message:

In a + b : longer object length is not a multiple of shorter object length

* 1. **Достигане до елементите на вектор**

Достигаме до елементите на вектор като посочваме индексът им, в оператор единични квадранти скоби"[ ]". Номерата на индексите започват от 1.

> s = c("aa", "bb", "cc", "dd", "ee")   
> s[3]   
[1] "cc"

Резултатът винаги е отново вектор.

> s = c("aa", "bb", "cc", "dd", "ee")   
> s[c(2, 3)]   
[1] "bb" "cc"

Допуска се дублиране на индекси

> s = c("aa", "bb", "cc", "dd", "ee")   
> s[c(2, 3, 3)]   
[1] "bb" "cc" "cc"

Изброяването на индексите може да не е подредено, резултатът ще следва подредбата на индексите

> s = c("aa", "bb", "cc", "dd", "ee")   
> s[c(2, 1, 3)]   
[1] "bb" "aa" "cc"

Изброяването на индексите може да стане чрез техния обхват

Това става чрез оператора ":". Удобно е при дълги вектори.

> s[2:4]   
[1] "bb" "cc" "dd"

Повече информация за оператора ":" може да бъде намерена в помощната документация на R.

> help(":")

Може да се достигне до компонентите на вектор чрез логически стойности, които могат да бъдат наречени „индикатори“ и имат същата дължина както изходният вектор. Новият вектор ще има компоненти, на чиито места има логическа стойност TRUE във вектора от логически индекси. За целта трябва да дефинираме вектор с логически стойности. По-долу това е L.

> s = c("aa", "bb", "cc", "dd", "ee")

> L = c(FALSE, TRUE, FALSE, TRUE, FALSE)

> s[L]   
[1] "bb" "dd"

Същата операция може да бъде записана на един ред

> s[c(FALSE, TRUE, FALSE, TRUE, FALSE)]   
[1] "bb" "dd"

Или

> s[s!="aa"&s!="cc"&s!="ee"]

[1] "bb" "dd"

* 1. **Триене на компоненти чрез отрицателни индекси**

Ако посоченият в единичните квадратни скоби индекс е отрицателен, в резултата ще бъде пропуснат компонентът на изходния вектор, който има индекс с абсолютна стойност равна на отрицателния индекс.

> s[-3]   
[1] "aa" "bb" "dd" "ee"

* 1. **Индекси извън обхвата** водят до резултат символът NA – not availablе. Това не променя типа на останалите компоненти на вектора.

> s[10]   
[1] NA

* 1. **Именуване на компонентите на вектор**

Когато имаме зададен вектор можем да именуваме компонентите му.

> v = c("I", "You", "He", "She", "It")   
> v   
[1] "I" "You" "He" "She" "It"

Сега ще именуваме първият член като I, вторият член като II и т.н. петият член като V

> names(v) = c("I", "II", "III", "IV", "V")   
> v   
  I II III IV V

"I" "You" "He" "She" "It"

После можем да използваме имената на компонентите вместо индекси.

> v["II"]   
[1] "You"

Можем да разместим компонентите чрез разместване на имената в индексиращия вектор.

> v[c("V", "II")]   
  V   II   
 "It"  "You"

1. **Матрици**
   1. **Създаване на матрици**

matrix(data, nrow, ncol, byrow)

Можем да създадем матрица чрез функцията ***matrix.*** Елементите трябва да са от един и същ основен тип.

> A = matrix(   
+   c(2, 4, 3, 1, 5, 7), # задаване на елементите   
+   nrow = 2,            # брой редове   
+   ncol = 3,            # брой колони   
+   byrow = TRUE)        # начин на попълване – в случая - по редове   
> A                      # печатна матрицата   
     [,1] [,2] [,3]   
[1,]    2    4    3   
[2,]    1    5    7

По подразбиране попълването е по колони

> B = matrix(   
+   c(2, 4, 3, 1, 5, 7),   
+   nrow = 3,   
+   ncol = 2)   
> B             # B има 3 реда и 2 колони   
     [,1] [,2]   
[1,]    2    1   
[2,]    4    5   
[3,]    3    7

Когато броят на компонентите е кратен на размерността на матрицата можем да зададем само едната размерност. Например при 20 елемента ако броят на редовете е 4 R автоматично oпределя броя на колоните – 5.

> F = matrix(   
+  seq(1:20),   
+  nrow = 4)   
F

[,1] [,2] [,3] [,4] [,5]

[1,] 1 5 9 13 17

[2,] 2 6 10 14 18

[3,] 3 7 11 15 19

[4,] 4 8 12 16 20

* 1. **Определяне на размерността на матрица**

Броят редовете и броят на колоните на матрицата може да бъде определен с помощта на функцията ***dim***.

> dim(B)

[1] 3 2 # B има 3 реда и 2 колони

* 1. **Извеждане на елементи на матрица**

Елементът, който е на *m-ти*ред и в *n-ти* стълб се достига чрез A[m, n].

> A[2, 3]      # елементът на 2-ри ред, 3-ти стълб   
[1] 7

*m-ти*ред на A може да бъде изведен чрез A[m, ].

> A[2, ]       # 2-ри ред   
[1] 1 5 7

*n-ти* стълб на А се достига чрез  A[ ,n].

> A[ ,3]       # 3-ти стълб   
[1] 3 7

Можем да изведем повече от една колона в едно и също време.

> A[ ,c(1,3)]  #  1-ва и 3-та колона на А   
     [,1] [,2]   
[1,]    2    3   
[2,]    1    7

* 1. **Трансформиране на матрици**

Транспониранетостава с помощта на функцията t .

> t(B)          # Транспониране на B   
     [,1] [,2] [,3]   
[1,]    2    4    3   
[2,]    1    5    7

Долепяне на матрици с един и същ брой редове една до друга

Ако редовете имат един и същ брой, то колоните на матриците могат да бъдат долепени с помощта на функцията ***cbind*** и да бъде получена по-голяма матрица.

> C = matrix(   
+   c(7, 4, 2),   
+   nrow=3,   
+   ncol=1)   
> C             # C има 3 реда   
     [,1]   
[1,]    7   
[2,]    4   
[3,]    2

> cbind(B, C)   
     [,1] [,2] [,3]   
[1,]    2    1    7   
[2,]    4    5    4   
[3,]    3    7    2

Долепяне на матрици с един и същ брой колони една под друга

По аналогичен начин ако колоните имат един и същ брой, то редовете на матриците могат да бъдат долепени с помощта на функцията ***rbind*** и да бъде получена по-голяма матрица.

> D = matrix(   
+   c(6, 2),   
+   nrow=1,   
+   ncol=2)   
> D             # D има 2 колони  
     [,1] [,2]   
[1,]    6    2   
> rbind(B, D)   
     [,1] [,2]   
[1,]    2    1   
[2,]    4    5   
[3,]    3    7   
[4,]    6    2

Превръщане на матрица във вектор-ред

Можем да превърнем матрица във вектор ред чрез последователно обхождане на компонентите по колони и с помощта на функцията ***с***.

> c(B)   
[1] 2 4 3 1 5 7

* 1. **Именуване на компонентите на матрица**

Когато имаме зададена матрица можем да именуваме редовете и колоните й.

> dimnames(A) = list(   
+   c("I row", "II row"),      #имена на редовете    
+   c("I col", "II col", "III col")) #имена на колоните   
После можем да достигнем до съответните редове и колони чрез техните имена  
> A                 #  Печат на A   
      I col II col III col   
I row     2     4     3   
II row    1     5     7   
   
> A["I row", "II col"] # Елементът във 2-ри ред, 3-та колона   
[1] 7

Премахването на имената на редовете и колоните става като им се зададат нулеви стойности

dimnames(A) = list(NULL, NULL)

A

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

[1,] 2 4 3

[2,] 1 5 7

* 1. **Аритметични операции с матрици**

Прибавяне на число към всеки елемент на матрица

A = matrix(

c(2, 4, 3, 1, 5, 7), # задаване на елементите

nrow = 2, # брой редове

ncol = 3, # брой колони

byrow = TRUE) # начин на попълване – в случая - по редове

A

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

[1,] 2 4 3

[2,] 1 5 7

A+3

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

[1,] 5 7 6

[2,] 4 8 10

По аналогичен начин умножение на всеки елемент на матрица с число (деление, изваждане).

Покомпонентно събиране на матрици **–** матриците трябва да са с еднаква размерност

G = matrix(c(rep(1:2, 3)), nrow=2)

> G

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

[1,] 1 1 1

[2,] 2 2 2

A + G

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

[1,] 3 5 4

[2,] 3 7 9

Покомпонентно изваждане на матрици **–** матриците трябва да са с еднаква размерност

A - G

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

[1,] 1 3 2

[2,] -1 3 5

Покомпонентно умножение на матрици **–** матриците трябва да са с еднаква размерност

A \* G

A \* G

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

[1,] 2 4 3

[2,] 2 10 14

По аналогичен начин покомпонентно деление на матрици.

Умножение на матрици **–** матриците трябва да са с размерност съответно mxn и nxr. Резултатът е mxr матрица.

G = matrix(c(rep(1:4, 2)), ncol=2)

G

[,1] [,2]

[1,] 1 1

[2,] 2 2

[3,] 3 3

[4,] 4 4

G%\*%A

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

[1,] 3 9 10

[2,] 6 18 20

[3,] 9 27 30

[4,] 12 36 40

Намиране на обратна матрица **–** изходната матрица трябва да е квадратна

|  |
| --- |
| G = matrix(c(2,5,9,3,4,7,9,10,2), ncol=3)  > G  [,1] [,2] [,3]  [1,] 2 3 9  [2,] 5 4 10  [3,] 9 7 2  > solve(G)  [,1] [,2] [,3]  [1,] -0.579 0.5 -0.06  [2,] 0.748 -0.7 0.23  [3,] -0.009 0.1 -0.07  > H=solve(G)  > H%\*%G  [,1] [,2] [,3]  [1,] 1e+00 1e-15 2e-15  [2,] -4e-16 1e+00 -5e-16  [3,] -1e-16 0e+00 1e+00 |

1. **Списъци(List) - list**e общ вектор, съдържащ други обекти.
   1. **Създаване на списъци** с функцията ***list***

В следващия пример променливата x е list, съдържащ копия на три вектора n, s, b, и числовата стойност 3.

> n = c(2, 3, 5)   
> s = c("aa", "bb", "cc", "dd", "ee")   
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)   
> x = list(n, s, b, 3)

> x

[[1]]

[1] 2 3 5

[[2]]

[1] "aa" "bb" "cc" "dd" "ee"

[[3]]

[1] TRUE FALSE TRUE FALSE FALSE

[[4]]

[1] 3

* 1. **Достигане до вектори от един и същ list(списък) и трансформиране на list**

За да достигнем до компонентите на list, които са вектори и образуват част от списък(т.е. резултатът е списък) използваме единични квадратни скоби"[ ]".

> x[2] # списък, който съдържа втория вектор от списъка х  
[[1]]   
[1] "aa" "bb" "cc" "dd" "ee"

Можем да направим list, който съдържа едновременно повече от един вектор от един и същи list.

> y = x[c(2, 4)] # списък, който съдържа втория и четвъртия вектор от списъка х  
> y

[[1]]   
[1] "aa" "bb" "cc" "dd" "ee"   
[[2]]   
[1] 3

За да достигнем до компонентите на list, които са вектори използваме двойни квадратни скоби"[[ ]]". Следващият обект е вектор, който съвпада с втората компонента на списъка х.

> x[[2]]   
[1] "aa" "bb" "cc" "dd" "ee"

Можем да достигнем и до съдържанието на векторите от списъка чрез редуване на двойни и единични квадратни скоби и да му присвоим стойност или да използваме неговата стойност.

> x[[2]][1] = "ta" # присвоява стойност “ta” на първата компонента на втория вектор на списъка х  
> x[[2]]   
[1] "ta" "bb" "cc" "dd" "ee"   
> s   
[1] "aa" "bb" "cc" "dd" "ee"   # s не е променен

* 1. **Именуване на елементи на списъци**

Можем да поставим имена членовете на list, и да се обръщаме към тях по име вместо по индекс.

Например в следващия списък имената са "Education" и "Gender".

> v = list("Education"=c(2, 3, 5), Gender=c("m", "f"))   
> v   
$Education   
[1] 2 3 5   
$Gender   
[1] "m" "f"

За да достигнем до компонентите на list, които са вектори и образуват част от списък(т.е. резултатът е списък) използваме единични квадратни скоби"[ ]", в които можем да поставим името на вектора.

> v["Education"]   
$Education   
[1] 2 3 5

Можем да направим list, който съдържа едновременно повече от един вектор от един и същи list като използваме единични квадратни скоби"[ ]", в които можем да поставим имената на векторите, до които искаме да достигнем. Могат да са в произволен ред. Резултатът е списък.

> v[c("Gender", "Education")]   
$Gender   
[1] "m" "f"   
   
$Education   
[1] 2 3 5

За да достигнем до компонентите на list, които са вектори използваме двойни квадратни скоби"[[ ]]". В тях можем да сложим името на вектора от листа. Следващият обект е вектор, който съвпада с първата компонента на списъка х.

> v[["Education"]]   
[1] 2 3 5

До неговите елементи можем да достигнем чрез

> v[["Education"]] [3]  
[1] 5

Същият ефект ще постигнем и ако използваме оператора "$"

> v$Education   
[1] 2 3 5

До неговите елементи можем да достигнем чрез

> v$Education[3]   
[1] 5

* 1. **Търсене на път - Attachment**

Можем да прикачим пътя до векторите от списъка за да не споменаваме винаги името на списъка, когато достигаме компонентите му. Това става с функцията *attach.* За да се върнем в предишното състояние използваме функцията *detach*.

> attach(v)   
> Education  
[1] 2 3 5   
> detach(v)

1. **Таблици(Data Frame)**
   1. **Създаване на data frame**

Фреймове с данни(**data frame)**се използват за съхраняване на таблични данни. По тази причина те се наричат често просто таблици. Те представляват list(списък) от вектори с еднаква дължина. Например.

> n = c(2, 3, 5)   
> s = c("aa", "bb", "cc")   
> b = c(TRUE, FALSE, TRUE)   
> df = data.frame(n, s, b)       # df е data frame

* 1. **Разглеждане на данни от data frame**

За да разгледаме данните от таблица с данни можем да напишем името на таблицата. Често обаче тези данни са прекалено много и по тази причина е желателно да видим началото на таблицата. Това става с помощта на функцията ***head***. Библиотеките в R съдържат много готови фреймове с данни, но обикновено те се построяват в резултат от експеримент. Един такъв data frame в R, се нарича **mtcars**.

> mtcars   
               mpg cyl disp  hp drat   wt ...   
Mazda RX4     21.0   6  160 110 3.90 2.62 ...   
Mazda RX4 Wag 21.0   6  160 110 3.90 2.88 ...   
Datsun 710    22.8   4  108  93 3.85 2.32 ...   
               ............

По-подробна информация за множеството от данни mtcars data set може да бъде намерена в поможната документация в R.

> help(mtcars)

На първия ред в таблицата са разположениимената на колоните те се наричат **header**. Всеки хоризонтален ред с данни също започва с име. Всяка данна в таблицата се нарича клетка (**cell)**.

> head(mtcars)   
               mpg cyl disp  hp drat   wt ...   
Mazda RX4     21.0   6  160 110 3.90 2.62 ...   
               ............

* 1. **Извличане на елементи на data frame**

За да достигнем до клетките в таблицата трябва в операторът "[ ]" единични квадратни скоби да посочим първо номерът на реда и след това номерът на колоната, в която се намира клетката. Двете координати трябва да са разделени със запетая.

> mtcars[1, 2]   
[1] 6

Нещо повече, можем да използваме имената на реда и колоната вместо числовите координати.

> mtcars["Mazda RX4", "cyl"]   
[1] 6

* 1. **Определяне на броят на редовете и броят на колоните в data frame**

> nrow(mtcars)    # брой редове   
[1] 32

> ncol(mtcars)    # брой колони   
[1] 11

* 1. **Вектор колона в data frame(резултатът няма имена на ледовете)**

Вектор-колона от data frame се достига с името на таблицата, последвано от операторът "[[ ]]" двойни скоби, в който се поставя името или номера на колоната .

> mtcars[[9]]   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

> mtcars[["am"]]   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

Може вместо това да използваме името на таблицата последвано от оператора "$" и след това името на колоната.

> mtcars$am   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

Може да се достигне до същия вектор-колона по още един начин, чрез операторът "[ ]" единични квадратни скоби. На мястото след запетаята мажем да напишем име или номер на колона

> mtcars[,"am"]   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

* 1. **Вектор колона от data frame като нов data frame, със същите имена на редовете. (резултатът има имена на ледовете)**

За да достигнем до вектор-колона от data frame, който да се разглежда отново като data frame се пише името на таблицата, последвано от операторът "[ ]" единични скоби, в който се поставя името или номера на колоната .

> mtcars[1]   
                   mpg   
Mazda RX4         21.0   
Mazda RX4 Wag     21.0   
Datsun 710        22.8   
                   ............

Същото ще постигнем с

> mtcars["mpg"]   
                   mpg   
Mazda RX4         21.0   
Mazda RX4 Wag     21.0   
Datsun 710        22.8   
                   ............

Ако искаме да отрежем от таблицата няколко колони, в квадратните скоби пишем вектор с имената или номерата им.

> mtcars[c("mpg", "hp")]   
                   mpg  hp   
Mazda RX4         21.0 110   
Mazda RX4 Wag     21.0 110   
Datsun 710        22.8  93   
                   ............

* 1. **Вектор колона от data frame като нов data frame, със същите имена на редовете. (резултатът има имена на ледовете)**

До ред в data frame се достига с името на таблицата последвано от единични квадратни скоби, в които има номера или името на реда последвани от запетая, която отбелязва, че става дума за ред.

> mtcars[24,]   
            mpg cyl disp  hp drat   wt  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

> mtcars["Camaro Z28",]   
            mpg cyl disp  hp drat   wt  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

Това показва, че през 1974 Camaro Z28 има разход на горива 13.3 miles на gallon, 8 цилиндъра, 245 конски сили мощност на двигателя, и т.н.

За да се достигне до повече от един ред след първата квадратна скоба и преди запетаята се задава вектор с имената или номерата на редовете, които искаме да отделим в друга таблица.

> mtcars[c(3, 24),]   
            mpg cyl disp  hp drat   wt  ...   
Datsun 710 22.8   4  108  93 3.85 2.32  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

> mtcars[c("Datsun 710", "Camaro Z28"),]   
            mpg cyl disp  hp drat   wt  ...   
Datsun 710 22.8   4  108  93 3.85 2.32  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

* 1. **Използване на логическо индексиране на редовете**

Можем да използваме индикатори, за да зададем кои редове да се включат в новият ни data frame. Включват се тези редове, които във вектора с логически индекси имат индикатор TRUE. Обикновено векторът с логически индекси се получава в резултат от някаква логическа операция. В слудващия ред във вектора L имаме TRUE ако колата има автоматична трансмисия и FALSE иначе.

> L = mtcars$am == 0   
> L   
 [1]   FALSE FALSE FALSE  TRUE ...

Ето я и таблицата с автомобилите, които ИМАТ автоматична трансмисия

> mtcars[L,]   
                     mpg cyl  disp  hp drat    wt  ...   
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215  ...   
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440  ...   
                 ............

Същото можем да постигнем с една команда

mtcars[mtcars$am == 0,]   
                     mpg cyl  disp  hp drat    wt  ...   
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215  ...   
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440  ...   
                 ............

* 1. **Достигане до стойностите на колона, които удовлетворяват определено условие в друга колона**

Следващият вектор връща разходът на газ за колите, които имат автоматична трансмисия.

> mtcars[L,]$mpg   
 [1] 21.4 18.7 18.1 14.3 24.4 ...

Същото можем да постигнем с една команда

> mtcars[mtcars$am == 0,]$mpg   
 [1] 21.4 18.7 18.1 14.3 24.4 ...

* 1. **Импортиране на data frame**

**Работа с Excel File**

Често се налага да се импортират данни, например от Excel File в R преди началото на нашата работа. За целта може да бъде използвана функцията ***read.xls*** от пакета ***gdata***. Тя превръща данните от таблица в Еxcel в [data frame](http://www.r-tutor.com/r-introduction/data-frame). Следните команди показват как да заредим данни от таблица в Excel с име "mydata.xls". Този метод изисква method Perl runtime да присъства в системата.

> library(gdata)                   # зареждане на пакета gdata   
> help(read.xls)                   # докемунтация   
> mydata = read.xls("mydata.xls")  # четене от първата работна таблица

За същата цел можем да използваме функцията loadWorkbook от пакетът XLConnect за да заредим работната таблица и после с readWorksheet да я прочетем. Пакетът XLConnect изисква предварително на компютърът да бъде инсталирана Java.

> library(XLConnect)               # Зарежда пакетът XLConnect   
> wk = loadWorkbook("mydata.xls")   
> df = readWorksheet(wk, sheet="Sheet1")

#### Работа с текстов фаил

Могат да бъдат импортират данни от текстов файл в R преди началото на нашата работа. Да предположим, че името му е "mydata.txt" и е записан в текстов редактор при това данните в един ред нека да бъдат разделени с интервал. Ето пример на една такава текстова таблица.

100   a1   b1   
200   a2   b2   
300   a3   b3   
400   a4   b4

За да заредим данните в работното пространство на R можем да използваме функцията ***read.table.***

> mydata = read.table("mydata.txt")  # чете текстов файл   
> mydata                             # отпечатва таблицата като автоматично слага имена    
   V1 V2 V3   
1 100 a1 b1   
2 200 a2 b2   
3 300 a3 b3   
4 400 a4 b4

За по-подробна информация относно употребата на функцията read.table, можете да разгледате помощната документация в R.

> help(read.table)

#### Работа с CSV фаил

Същите данни могат да бъдат съхранени в **comma separated values**(CSV) формат. Всяка клетка в такъв файл е отделена със специален символ от съседните в реда. Обикновено се използва запетая.

Първият ред би трябвало да съдържа имената на редовете, а не данни. Ето очакваният формат на една такава таблица.

Col1,Col2,Col3   
100,a1,b1   
200,a2,b2   
300,a3,b3

Записваме файла, например с име "mydata.csv" в текстов редактор или в Excel. Можем да прочетем този фаил с функцията ***read.csv***.

> mydata = read.csv("mydata.csv")  # четене на csv файл   
> mydata   
  Col1 Col2 Col3   
1  100   a1   b1   
2  200   a2   b2   
3  300   a3   b3

В много случаи запетаята и десетичната точка могат да бъдат сменени. В този случай е по-добре да се използва функцията ***read.csv2***.

За по-подробна информация относно употребата на функцията read.csv, можете да разгледате помощната документация в R.

> help(read.csv)

1. **Работна директория**

По-горните примери предполагат, че всички файлове са разположени в работната директория на R. Тя може да бъде открита с помощта на функцията getwd.

> getwd()

Работната директория може да бъде сменена. Това става с помощта на функцията setwd().

> setwd("<new path>")   # set working directory

Дори в пратформата на Windows като разделител на директориите се използват една или две наклонени черти

> setwd("[1] "C:/Users/My")

Това ще предотврати постоянното въвеждане на дълги имена на пътища, които се налага да пишем за да достигнем с файловете с данни.