ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ | DEMOCRITUS UNIVERSITY OF THRACE

Τύποι δεδομένων, πλαίσια δεδομένων, παράγοντες και χειρισμοί συνόλων

Πέτρος Κολοβός

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΥΓΕΙΑΣ
ΤΜΗΜΑ ΜΟΡΙΑΚΗΣ ΒΙΟΛΟΓΙΑΣ ΚΑΙ ΓΕΝΕΤΙΚΗΣ

Τύποι δεδομένων στην R

Οι κύριες δομές των δεδομένων είναι:

- Τα Διανύσματα (**vectors**)
- Οι Πίνακες δύο Διαστάσεων (**matrices**)
- Τα Πλαίσια Δεδομένων (**data frames**)
- Οι Πολυδιάστατοι Πίνακες (**arrays/tables**)
- Λίστες (**lists**)

Τα ονόματά τους ενδέχεται να είναι παραπλανητικά σε σχέση με την πραγματική δομή τους. Έτσι π.χ. οι πίνακες τύπου **array**/**table** είναι πιο πολύπλοκοι από τους πίνακες τύπου matrix, ενώ οι λίστες (**lists**) είναι τα πιο σύνθετα αντικείμενα σε αντίθεση με αυτό που ίσως υποδηλώνει το όνομά τους.

Γενικά χαρακτηριστικά διανυσμάτων (vectors)

- Τα **διανύσματα** είναι απλά μονοδιάστατα αντικείμενα που περιέχουν στοιχεία του ίδιου είδους.
- Μπορούμε να δημιουργήσουμε ένα διάνυσμα συνδέοντας μεταβλητές του ίδιου είδους με τη χρήση μιας βασικής συνάρτησης που ονομάζεται **c()** και συμβολίζει τη συνένωση (concatenation).
- Η χρήση της c() γίνεται με την εισαγωγή των μεταβλητών ως ορισμάτων διαχωρισμένων με κόμμα:

Hide

```
vec<-c(1,2,5,3.14,8)
vec
```

```
## [1] 1.00 2.00 5.00 3.14 8.00
```

Γενικά χαρακτηριστικά διανυσμάτων (vectors)

- Στο παραπάνω παράδειγμα όλες οι μεταβλητες είναι αριθμητικές κι έτσι το είδος του αντικειμένου είναι numeric όπως προκύπτει τόσο από τον έλεγχο του είδους **class()** όσο και με τη διερεύνηση της δομής του με την **str()**:

Hide

```
class(vec)
```

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

Hide

```
str(vec)
```

```
##  num [1:5] 1 2 5 3.14 8
```

- Η **str()** δεν μας δίνει μόνο το είδος των μεταβλητών που περιέχει το διάνυσμα vec αλλά μας επιστρέφει και το μέγεθός του. Πρόκειται για ένα διάνυσμα 5 τιμών όπως φαίνεται από τις τιμές στην αγκύλη [1:5], που σημαίνει ότι τα στοιχεία ξεκινούν από το 1 και φτάνουν ως το 5.

Γενικά χαρακτηριστικά διανυσμάτων (vectors)

- Οι **αγκύλες** είναι ο τρόπος με τον οποίο μπορούμε να αναφερθούμε σε επιμέρους στοιχεία αντικειμένων, τόσο διανυσμάτων όσο και πινάκων και λιστών όπως θα δούμε στη συνέχεια.
- Γενικά για τα διανύσματα, μπορούμε να αντιμετωπίσουμε τις αγκύλες ως μια συνάρτηση της οποίας το όρισμα είναι ένα διάνυσμα που είναι υποχρεωτικά αριθμητικό ακέραιου είδους.
- Έτσι οι αγκύλες μπορούν να δεχτούν σύνολα τιμών όπως

Hide

```
vec[c(1,2,3)]
```

```
## [1] 1 2 5
```

Γενικά χαρακτηριστικά διανυσμάτων (vectors)

- Συνεχόμενες αριθμητικές τιμές μπορούν να δοθούν με τον τελεστή “:” που δημιουργεί μια σειρά ακέραιων αριθμών από a έως b αν γράψουμε a:b, και παίρνουμε τις τιμές που ανήκουν στο διάστημα a:b.

Hide

```
x<-1:4
vec[x]
```

```
## [1] 1.00 2.00 5.00 3.14
```

Γενικά χαρακτηριστικά διανυσμάτων (vectors)

- Οι αριθμοί που δίνονται σαν όρισμα στις αγκύλες δεν είναι υποχρεωτικό να δίνονται με αύξουσα σειρά και μπορούν να επαναλαμβάνονται.
- Ο μοναδικός περιορισμός είναι να μην ξεπερνούν το συνολικό μήκος του διανύσματος. Το τελευταίο δίνεται σε μια μεταβλητή με την εφαρμογή της συνάρτησης **length()**.

Hide

```
l<-length(vec)
i<-3:l
vec[i]
```

```
## [1] 5.00 3.14 8.00
```

Δημιουργία διανυσμάτων και “εξαναγκασμός” (coersion)

- Οι ίδιες αρχές διέπουν τόσο τα αριθμητικά όσο και τα αλφαριθμητικά και λογικά διανύσματα.
- Η δημιουργία διαφορετικών τύπων μπορεί να γίνει με τυπική δήλωση του είδους τους με τις συναρτήσεις **numeric(), character() και logical()** οι οποιες δημιουργούν κενά διανύσματα συγκεκριμένου μήκους που δίνεται ως όρισμά τους

Hide

```
numvec<-numeric(10) # default value is 0
chrvec<-character(10) # default value is ""
logvec<-logical(10) # default value is FALSE
```

Hide

```
numvec<-vector(mode="numeric", length=10)
chrvec<-vector(mode="character", length=10)
logvec<-vector(mode="logical", length=10)
```

Δημιουργία διανυσμάτων και “εξαναγκασμός” (coersion)

- Ένας τρίτος τρόπος δημιουργίας διανυσμάτων είναι με την απευθείας απόδοση τιμών όπως είδαμε παραπάνω.
- Η απευθείας απόδοση τιμών χρειάζεται προσοχή λόγω μιας βασικής λειτουργίας της R που λέγεται “εξαναγκασμός” (**coersion**) και που μετατρέπει το είδος της μεταβλητής ανάλογα με μια σειρά προτεραιότητας που είναι: χαρακτήρες, αριθμοί, λογικές τιμές.

Hide

```
unknown<-c("variable1", "variable2", FALSE, TRUE, TRUE)
str(unknown)
```

```
##  chr [1:5] "variable1" "variable2" "FALSE" "TRUE" "TRUE"
```

- Παρότι στην εκχώρηση των μεταβλητών περιέχονται δύο αλφαριθμητικά και τρεις λογικές τιμές, οι τελευταίες δεν λογίζονται ως τέτοιες, παρά μετατρέπονται σε σειρές χαρακτήρων.
- Αυτό συμβαίνει αφενός γιατί ένα διάνυσμα δεν μπορεί να περιέχει διαφορετικά είδη μεταβλητών και αφετέρου γιατί η προτεραιότητα εξαναγκασμού είναι προς τα αλφαριθμητικά.

Εξωτερικός “εξαναγκασμός” από τον χρήστη

- Σε κάποιες περιπτώσεις επιθυμούμε τη μετατροπή δεδομένων σε ένα συγκεκριμένο είδος αν και ως πρακτική είναι σχετικά επικίνδυνη και είναι καλό να γίνεται με προσοχή και πάντα με επαλήθευση.
- Ο επιβαλλόμενος εξαναγκασμός (**forced coersion**) γίνεται με τις συναρτήσεις τύπου **as.X()** όπου X=numeric, character, logical κλπ, οι οποίες μετατρέπουν διανύσματα ενός είδους στο είδος που ορίζει η συνάρτηση.

Hide

```
logvec<-c(T,T,F,F,T,T,F)
str(logvec)
```

```
##  logi [1:7] TRUE TRUE FALSE FALSE TRUE TRUE ...
```

Hide

```
as.numeric(logvec)
```

```
## [1] 1 1 0 0 1 1 0
```

Hide

```
as.character(logvec)
```

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

Εξωτερικός “εξαναγκασμός” από τον χρήστη

- Δεν είναι επιτρεπτές όλες οι μετατροπές. Έτσι είναι δυνατή η μετατροπή τόσο λογικών όσο και αριθμητικών σε αλφαριθμητικά με απλή χρήση εισαγωγικών.

Hide

```
chrvec<-c("steven","kenny","robbie","ian","fernando")
str(chrvec)
```

```
##  chr [1:5] "steven" "kenny" "robbie" "ian" "fernando"
```

Hide

```
as.numeric(chrvec)
```

```
## Warning: NAs introduced by coercion
```

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

Hide

```
as.logical(chrvec)
```

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

Συναρτήσεις Αριθμητικών Διανυσμάτων

Οι βασικές συναρτήσεις των αριθμητικών διανυσμάτων είναι οι εξής

- ✓ **length():** Μέγεθος, ή μήκος, δηλαδή το πλήθος των στοιχείων ενός διανύσματος:
- ✓ **max(),min():** Μέγιστη/Ελάχιστη τιμή.
- ✓ **rev():** Αντιστροφή σειράς των στοιχείων του διανύσματος:
- ✓ **sort():** Κατάταξη (με αύξουσα ή φθίνουσα σειρά)
- ✓ **order():** Θέση κατάταξης. Δίνει τη θέση στην κατάταξη κατά αύξουσα σειρά κρατώντας όμως τη σειρά των στοιχείων σταθερή.
- ✓ **rank():** Τιμή κατάταξης. Δίνει τη τιμή στην κατάταξη κατά αύξουσα σειρά.

Συναρτήσεις Αριθμητικών Διανυσμάτων

✓ **length():** Μέγεθος, ή μήκος, δηλαδή το πλήθος των στοιχείων ενός διανύσματος:

Hide

```
vec<-c(1, 8, 9, 10, -4, 2.8, 10)
length(vec)
```

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

Συναρτήσεις Αριθμητικών Διανυσμάτων

✓ **max(),min():** Μέγιστη/Ελάχιστη τιμή.

Hide

```
max(vec)
```

```
## [1] 10
```

Hide

```
min(vec)
```

```
## [1] -4
```


Συναρτήσεις Αριθμητικών Διανυσμάτων

✓ **rev():** Αντιστροφή σειράς των στοιχείων του διανύσματος:

Hide

```
vec
```

```
## [1]  1.0  8.0  9.0 10.0 -4.0  2.8 10.0
```

Hide

```
rev(vec)
```

```
## [1] 10.0  2.8 -4.0 10.0  9.0  8.0  1.0
```

Συναρτήσεις Αριθμητικών Διανυσμάτων

✓ **sort():** Κατάταξη (με αύξουσα ή φθίνουσα σειρά)

Hide

```
vec
```

```
## [1]  1.0  8.0  9.0 10.0 -4.0  2.8 10.0
```

Hide

```
sort(vec) # default αύξουσα σειρά
```

```
## [1] -4.0  1.0  2.8  8.0  9.0 10.0 10.0
```

Hide

```
sort(vec, decreasing = T) # φθίνουσα σειρά
```

```
## [1] 10.0 10.0  9.0  8.0  2.8  1.0 -4.0
```

Συναρτήσεις Αριθμητικών Διανυσμάτων

- ✓ **order():** Θέση κατάταξης. Δίνει τη θέση στην κατάταξη κατά αύξουσα σειρά κρατώντας όμως τη σειρά των στοιχείων σταθερή.

Hide

```
vec<-c(1, 8, 9, 10, -4, 2.8, 10)
order(vec)
```

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


Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

Οι βασικές συναρτήσεις των αλφαριθμητικών διανυσμάτων είναι οι εξής

- ✓ **length():** Μέγεθος, ή μήκος, δηλαδή το πλήθος των στοιχείων ενός διανύσματος:
- ✓ **rev():** Αντιστροφή σειράς των στοιχείων του διανύσματος:
- ✓ **sort():** Κατάταξη (με αύξουσα ή φθίνουσα σειρά)
- ✓ **toupper()** και **tolower()** για μετατροπή σε κεφαλαία ή πεζά αντίστοιχα
- ✓ Αποκοπή στοιχείων από μια σειρά χαρακτήρων με τη συνάρτηση **substr()**
- ✓ Μπορούμε να "κόψουμε" τους χαρακτήρες χρησιμοποιώντας συγκεκριμένα σύμβολα/γράμματα ως διαχωριστές με τη χρήση της **$strsplit()**

Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

✓ **length():** Μέγεθος, ή μήκος, δηλαδή το πλήθος των στοιχείων ενός διανύσματος:

Hide

```
vec<-c("nick", "mike", "eve", "laura", "george")
length(vec)
```

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

Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

✓ **rev():** Αντιστροφή σειράς των στοιχείων του διανύσματος:

Hide

```
vec
```

```
## [1] "nick"   "mike"   "eve"    "laura"  "george"
```

Hide

```
rev(vec)
```

```
## [1] "george" "laura"  "eve"    "mike"   "nick"
```

Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

✓ **sort():** Κατάταξη (με αύξουσα ή φθίνουσα σειρά)

Hide

```
vec
```

```
## [1] "nick"   "mike"   "eve"    "laura"  "george"
```

Hide

```
sort(vec) # default αύξουσα σειρά
```

```
## [1] "eve"    "george" "laura"  "mike"   "nick"
```

Hide

```
sort(vec, decreasing = T) # φθίνουσα σειρά
```

```
## [1] "nick"   "mike"   "laura"  "george" "eve"
```

Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

✓ **toupper()** και **tolower()** για μετατροπή σε κεφαλαία ή πεζά αντίστοιχα

Hide

```
toupper(vec)
```

```
## [1] "NICK"   "MIKE"   "EVE"    "LAURA"  "GEORGE"
```

Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

✓ Αποκοπή στοιχείων από μια σειρά χαρακτήρων με τη συνάρτηση **substr()**

Hide

```
substr(vec, 1, 2)
```

```
## [1] "ni" "mi" "ev" "la" "ge"
```


Συναρτήσεις Αλφαριθμητικών Διανυσμάτων

- ✓ Μπορούμε να “κόψουμε” τους χαρακτήρες χρησιμοποιώντας συγκεκριμένα σύμβολα/γράμματα ως διαχωριστές με τη χρήση της **$strsplit()**

Hide

```
strsplit(vec, "i")
```

```
## [[1]]
## [1] "n"  "ck"
##
## [[2]]
## [1] "m"  "ke"
##
## [[3]]
## [1] "eve"
##
## [[4]]
## [1] "laura"
##
## [[5]]
## [1] "george"
```

Πράξεις σε Διανύσματα

Η μεγάλη δύναμη της R συνίσταται στη διανυσματική εφαρμογή συναρτήσεων. Έστω το αριθμητικό διάνυσμα με την **seq()** που παίρνει τρία ορίσματα (**αρχή, τέλος, βήμα προόδου**) :

Hide

```
values<-seq(from=1, to=100, by=2)
```


Πράξεις σε Διανύσματα

Έστω τώρα ότι θέλουμε να υπολογίσουμε την τετραγωνική ρίζα όλων των τιμών του διανύσματος.

Hide

```
sr_values<-vector(mode="numeric", length=length(values))
for(i in 1:length(values)){
  sr_values[i]<-sqrt(values[i])
}
```

- Νέο διάνυσμα με τη συνάρτηση **vector()**, καθορίζοντας το είδος (**numeric**) και το μέγεθός του με τη χρήση του **length**.
- Επαναληπτική δομή με τη χρήση της εντολής **for**. Γυμνή μεταβλητη **i** που είναι ο δείκτης της επανάληψης και παίρνει τιμές από το **1 έως το μήκος του διανύσματος values**.
- Μέσα σε άγκιστρα παρατίθενται οι πράξεις/διαδικασίες που θα πραγματοποιηθούν σε κάθε βήμα της επανάληψης.
- Η **sqrt()** υπολογίζει την τετραγωνική ρίζα του κάθε στοιχείου του values και την αποδίδει στο αντίστοιχο στοιχείο του srvalues.
- Ο βρόχος (**loop**) επανάληψης θα ήταν απαραίτητος και ανάλογα με τον αριθμό των υπολογισμών, την πολυπλοκότητά τους και το μέγεθος των εμπλεκόμενων διανυσμάτων θα μπορούσε να είναι χρονοβόρος

Πράξεις σε Διανύσματα

Η παρακάτω διαδικασία είναι αυτό που θα χρειαζόταν να κάνουμε σε μια γλώσσα που δεν υποστηρίζει διανυσματική εφαρμογή. Ο βρόχος (loop) επανάληψης θα ήταν απαραίτητος και ανάλογα με τον αριθμό των υπολογισμών, την πολυπλοκότητά τους και το μέγεθος των εμπλεκόμενων διανυσμάτων θα μπορούσε να είναι και αρκετά χρονοβόρος

Hide

```
sr_values<-vector(mode="numeric", length=length(values))
for(i in 1:length(values)){
  sr_values[i]<-sqrt(values[i])
}
```

Ωστόσο στην R η ίδια διαδικασία μπορεί να γίνει με μια απλή εντολή όπως η παρακάτω

Hide

```
sr_values<-sqrt(values)
```

Πίνακες δύο διαστάσεων

- Οι πίνακες δύο διαστάσεων στην R ονομάζονται **matrices** και σχηματίζονται με δεδομένα σε σειρές και στήλες. Τα δεδομένα πρέπει να είναι του **ίδιου τύπου** σε όλες τις στήλες του πίνακα
- Οι πίνακες σχηματίζονται με τη χρήση της συνάρτησης **matrix()** που παίρνει σαν παραμέτρους τις τιμές του πίνακα σε ένα διάνυσμα, μαζί με τις διαστάσεις του πίνακα, τον τρόπο διάρθρωσης του διανύσματος και προαιρετικά τα ονόματα των γραμμών και στηλών του

Hide

```
values<-seq(1:30)
matrix(values, nrow=4, ncol=5, byrow=T)
```

```
## Warning in matrix(values, nrow = 4, ncol = 5, byrow = T): data length [30]
## is not a sub-multiple or multiple of the number of rows [4]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
```

Πρόσβαση σε στοιχεία πινάκων

- Τα στοιχεία των matrices καθορίζονται με δύο τιμές που αντιστοιχούν στις συντεταγμένες γραμμής, στήλης και που δίνονται μέσα σε αγκύλες με αντίστοιχη σειρά.

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
```

Hide

```
mat[1,3]
```

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

Συναρτήσεις πινάκων

- Διαστάσεις ενός πίνακα με την **dim()**

Hide

```
dim(mat)
```

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


Συναρτήσεις πινάκων

- **dimnames()** η οποία χρησιμοποιείται για να αποδοθούν ονόματα στις γραμμές και τις στήλες του πίνακα

Hide

```
dimnames(mat)<-list(c("R1","R2","R3","R4"), c("A","B","C","D","E"))
mat
```

```
##     A  B  C  D  E
## R1  1  2  3  4  5
## R2  6  7  8  9 10
## R3 11 12 13 14 15
## R4 16 17 18 19 20
```

Συναρτήσεις πινάκων

- Η τιτλοδότηση γραμμών και στηλών μπορεί να γίνει και σε δύο βήματα με τη χρήση των συναρτήσεων **rownames()** και **colnames()** οι οποίες δέχονται διανύσματα

Hide

```
colnames(mat)<-c("C1", "C2", "C3", "C4", "C5")
mat
```

```
##    C1 C2 C3 C4 C5
## R1  1  2  3  4  5
## R2  6  7  8  9 10
## R3 11 12 13 14 15
## R4 16 17 18 19 20
```


Συναρτήσεις πινάκων

- Οι πίνακες μπορούν να αυξηθούν, να συνενωθούν ή να αλλάξουν διαστάσεις. Διανύσματα ή και πίνακες μπορούν να ενωθούν με πίνακες ανά γραμμή ή ανά στήλη με τις **rbind()** και **cbind()** αντίστοιχα

Hide

```
mat1<-matrix(seq(1:10), nrow=2, ncol=5, byrow=T, dimnames=list(c("R1","R2"), c("C1","C2","C3","C4","
C5")))
mat2<-matrix(seq(21:40), nrow=4, ncol=5, byrow=T, dimnames=list(c("L1","L2","L3","L4"), c("S1","S2",
"S3","S4","S5")))
rbind(mat1,mat2)
```

```
##    C1 C2 C3 C4 C5
## R1  1  2  3  4  5
## R2  6  7  8  9 10
## L1  1  2  3  4  5
## L2  6  7  8  9 10
## L3 11 12 13 14 15
## L4 16 17 18 19 20
```

Πλαίσια Δεδομένων (data frames)

- Τα πλαίσια δεδομένων (**dataframes**) είναι η πιο κοινή και πιο διαδεδομένη κατηγορία αντικειμένων στην R.
- Η **ευελιξία** τους και ο **μικρός αριθμός περιορισμών** που έχουν σε ό,τι αφορά την ενσωμάτωση διαφορετικών τύπων δεδομένων τα κάνουν εξαιρετικά εύχρηστα.
- Ένα μεγάλο εύρος **συναρτήσεων** επενεργούν πάνω σε πλαίσια δεδομένα και για το λόγο αυτό αφιερώνουμε εξ ολοκλήρου σε αυτά το επόμενο κεφάλαιο.
- Τα πλαίσια δεδομένων λοιπόν μπορούν να **περιέχουν διαφορετικούς τύπους δεδομένων σε διαφορετικές στήλες**, με την προϋπόθεση ότι το **μήκος της κάθε στήλης είναι το ίδιο**.
- Η δημιουργία τους μπορεί να γίνει με τη συνάρτηση **data.frame()**

Πλαίσια Δεδομένων (data frames)

- Η **data.frame()** δίνει έτσι ταυτόχρονα τους τίτλους των διανυσμάτων καθώς και τις τιμές τους.

Hide

```
legends<-data.frame(names=c("Kenny","Robbie","Steven"), numbers=c(7,9,8), years=c(1959, 1970, 1974))
legends
```

```
##    names numbers years
## 1  Kenny       7  1959
## 2 Robbie       9  1970
## 3 Steven       8  1974
```


Πλαίσια Δεδομένων (data frames)

- Η πρόσβαση στις τιμές ενός πλαισίου δεδομένων μπορεί να γίνει είτε μέσω των διαστάσεών του είτε μέσω των ονομάτων των επιμέρους διανυσμάτων του

Hide

```
legends[,2]
```

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

Hide

```
legends$names
```

```
## [1] Kenny  Robbie Steven
## Levels: Kenny Robbie Steven
```

Εφαρμογή συναρτήσεων σε πίνακες και πλαίσια δεδομενων (apply)

- Οι πίνακες (**matrices**) και τα πλασια δεδομένων (**dataframes**) είναι σύνολα διανυσμάτων ίσου μήκους.
- Στην περίπτωση που θέλουμε να εφαρμόσουμε μια συνάρτηση σε καθένα από τα συστατικά διανύσματα ενός πίνακα μπορούμε να το κάνουμε χωρίς να καλέσουμε μια επαναληπτική διαδικασία με τη χρήση μιας ειδικής συναρτήσης που ονομάζεται **apply().**
- Η apply() είναι ουσιαστικά μια “συνάρτηση συναρτήσεων” που καλεί άλλες συναρτήσεις πάνω σε πίνακες είτε ανα γραμμή, είτε ανα στήλη.
- Έστω για παράδειγμα ότι θέλουμε να βρούμε το άθροισμα των τιμών κάθε γραμμής ενός matrix. Μπορούμε να εφαρμόσουμε την αντίστοιχη συνάρτηση sum() μέσω της apply() απευθείας πάνω στον πίνακα. Αρχικά θα δημιουργήσουμε έναν πίνακα 4χ5 τον οποίον θα γεμίσουμε με 20 τυχαίους αριθμούς μεταξύ 0 και 1 που επιλέγονται από μια ομοιόμορφη κατανομή. Τη συνάρτηση που κάνει αυτήν την μοντελοποίηση και που λέγεται runif() θα την δούμε αναλυτικά σε επόμενο κεφάλαιο. Στη συνέχεια οι 20 αυτές τιμές οργανώνονται σε έναν πίνακα με την εντολή matrix():

Εφαρμογή συναρτήσεων σε πίνακες και πλαίσια δεδομενων (apply)

- Έστω για παράδειγμα ότι θέλουμε να βρούμε το άθροισμα των τιμών κάθε γραμμής ενός matrix. Μπορούμε να εφαρμόσουμε την αντίστοιχη συνάρτηση **sum()** μέσω της **apply()** απευθείας πάνω στον πίνακα.
- Αρχικά θα δημιουργήσουμε έναν πίνακα 4χ5 τον οποίον θα γεμίσουμε με 20 τυχαίους αριθμούς μεταξύ 0 και 1 που επιλέγονται από μια ομοιόμορφη κατανομή.
- Στη συνέχεια οι 20 αυτές τιμές οργανώνονται σε έναν πίνακα με την εντολή matrix():

Hide

```
values<-runif(20)
mat<-matrix(values, nrow=4, ncol=5, byrow=T, dimnames=list(c("L1","L2","L3","L4"), c("S1","S2","S3",
"S4","S5")))
mat
```

```
##           S1        S2        S3        S4        S5
## L1 0.9602964 0.7352544 0.9139848 0.6774785 0.3400456
## L2 0.8106948 0.3572961 0.4517536 0.6759379 0.7712844
## L3 0.2125459 0.2344822 0.5377259 0.4109099 0.2776608
## L4 0.2221093 0.4251656 0.6455789 0.3703142 0.1553387
```

Εφαρμογή συναρτήσεων σε πίνακες και πλαίσια δεδομενων (apply)

- Μπορούμε τώρα να αθροισουμε τις τιμές του mat σε μία εντολή ανά γραμμή με τη χρήση της **apply()** και την εφαρμογή της συνάρτησης που υπολογίζει το άθροισμα ενός διανύσματος και που λέγεται **sum()**
- Η κλήση της **apply()** περιλαμβάνει τον **πίνακα**, την **εφαρμοζόμενη συνάρτηση** (εδώ είναι η sum()) και τη **διάσταση που θα εφαρμοστεί**, η οποία δίνεται ενδιάμεσα και είναι 1 για τις γραμμές και 2 για τις στήλες

Hide

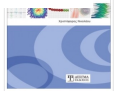
```
apply(mat, 2, FUN=sum)
```

```
##       S1       S2       S3       S4       S5
## 2.205646 1.752198 2.549043 2.134640 1.544329
```

Hide

```
apply(mat, 1, FUN=sum)
```

```
##       L1       L2       L3       L4
## 3.627060 3.066967 1.673325 1.818507
```

Πίνακες περισσότερων από δύο διαστάσεων (arrays)

- Τα **arrays** οργανώνονται σε πολλές διαστάσεις με ιδιαίτερη σειρά που ακολουθεί τη λογική [γραμμή, στήλη, μπλόκ1, μπλοκ2…, κοκ].

Hide

```
x[1,3,2]<-5
x
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    5    0    0    0
## [2,]    0    0    0    0    0    0
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
```

Πίνακες περισσότερων από δύο διαστάσεων (arrays)

- Η R υποστηρίζει τη δημιουργία πινάκων περισσότερων από δύο διαστάσεων, με την απλή χρήση της παραμέτρου **dim**
- Διάνυσμα από 36 μηδενικά με τη συνάρτηση **rep(a,n)**, η οποία δημιουργεί μια επανάληψη του a για n στοιχεία
- Πίνακας 2x6x3

Hide

```
x<-rep(0,36)
dim(x)<-c(2,6,3)
```

Hide

```
class(x)
```

```
## [1] "array"
```

Δημιουργία Λιστών (lists)

- Οι λίστες (**lists**) είναι η πιο πολύπλοκη κατηγορία δεδομένων στην R.
- Αποτελούνται από σύνολα διανυσμάτων που δεν είναι υποχρεωτικό να είναι του ίδιου τύπου αλλά ούτε και του ίδιου μήκους.
- Στην πιο διευρυμένη μορφή τους μπορούν να περιέχουν ακόμα και πίνακες ή και συναρτήσεις ως στοιχεία.

Hide

```
l<-list(c(1,3,5), c("Steven", "Robbie", "Kenny", "Ian"), TRUE, c(0.9, 0.8))
l
```

```
## [[1]]
## [1] 1 3 5
##
## [[2]]
## [1] "Steven" "Robbie" "Kenny"  "Ian"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 0.9 0.8
```


Δημιουργία Λιστών (lists)

- Για να δημιουργήσουμε μια πιο καλά οργανωμένη εκδοχή της θα δώσουμε ονόματα στα στοιχεία της **l** με τη χρήση της **names**.

Hide

```
names(l)<-c("odds", "names", "logical", "decimals")
l
```

```
## $odds
## [1] 1 3 5
##
## $names
## [1] "Steven" "Robbie" "Kenny"  "Ian"
##
## $logical
## [1] TRUE
##
## $decimals
## [1] 0.9 0.8
```

Δημιουργία Λιστών (lists)

- Η διαφορά στη χρήση μονών και διπλών αγκυλών είναι αρκετά λεπτή και μπορεί να διαφύγει

Hide

```
l[1]
```

```
## $odds
## [1] 1 3 5
```

```
l[[1]]
```

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

```
class(l[1])
```

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

```
class(l[[1]])
```

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

- Μονές αγκυλες -> αντικείμενο τύπου λίστας
- Διπλές αγκύλες -> διάνυσμα του πρώτου στοιχείου

Δημιουργία Λιστών (lists)

- Στην περίπτωση που θέλουμε να πάρουμε πίσω όλη τη λίστα σε ένα διάνυσμα μπορούμε να την "καταρρεύσουμε" σε διάνυσμα με την συνάρτηση **unlist()**.

Hide

```
unlist(l)
```

```
##     odds1     odds2     odds3    names1    names2    names3    names4
##       "1"       "3"       "5"  "Steven"  "Robbie"  "Kenny"     "Ian"
##   logical decimals1 decimals2
##    "TRUE"     "0.9"     "0.8"
```

Χειρισμός Λιστών με συναρτήσεις τύπου apply()

- Μπορούμε να ενώσουμε δύο ή περισσότερες λίστες με την συνάρτηση **c()**

Hide

```
l1<-list(c("A","B"), c(1,2,3))
l2<-list(c("C","D","E"), c(T,F,F,F))
l3<-c(l1,l2)
l3
```

```
## [[1]]
## [1] "A" "B"
##
## [[2]]
## [1] 1 2 3
##
## [[3]]
## [1] "C" "D" "E"
##
## [[4]]
## [1]  TRUE FALSE FALSE FALSE
```


Χειρισμός Λιστών με συναρτήσεις τύπου apply()

- Η **βασικότερη χρησιμότητα των λιστών** είναι η δυνατότητα εφαρμογής συναρτήσεων στα στοιχεία τους με τη χρήση των συναρτήσεων **apply()**.
- Οι συναρτήσεις τύπου apply() λειτουργούν καλώντας άλλες συναρτήσεις και εφαρμόζοντάς τις απευθείας, με αποτέλεσμα την ταχύτατη εκτέλεσή τους.
- Στην περίπτωση των λιστών, το **πλεονέκτημα** που έχουμε σε σχέση με τους πίνακες είναι ότι τα διανύσματα που μπορούμε να αναλύσουμε δεν είναι υποχρεωτικά ίδιου μήκους.
- Έστω για παράδειγμα ότι έχουμε μια σειρά από διανύσματα των οποίων θέλουμε να υπολογίσουμε τις μέγιστες τιμές. Μπορούμε να αποφύγουμε μια επαναληπτική κλήση της αντίστοιχης συνάρτησης max() ενσωματώνοντας τα διανύσματα σε μια **λίστα** και καλώντας την **max()** σε μία κλήση της **sapply()**

Hide

```
x<-c(1, 8, 9, 12, 5)
y<-c(6, 7, 0.8, 2, 5.1)
z<-c(-10, 5, 2, 3.8)
my_list<-list(x,y,z)
sapply(my_list, FUN=max)
```

```
## [1] 12  7  5
```


Χειρισμός Λιστών με συναρτήσεις τύπου apply()

- Η **lapply()** μας δινει το ιδιο αποτελεσμα αλλα σε λιστα

Hide

```
lapply(my_list, FUN=max)
```

```
## [[1]]
## [1] 12
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] 5
```

Ύλη

Ανάλυση δεδομένων με την R

Κεφάλαιο 3

Χειρισμοί Πλαίσιων Δεδομένων

Ας θυμηθούμε καταρχάς τη δομή ενός dataframe φορτώνοντας απευθείας από την R ένα σετ δεδομένων που περιέχει μετεωρολογικές μετρήσεις (airquality)

Hide

```
str(airquality)
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```


Κενές τιμές (missing values)

Hide

```
airquality$Ozone
```

```
##   [1]  41  36  12  18  NA  28  23  19   8  NA   7  16  11  14  18  14  34
##  [18]   6  30  11   1  11   4  32  NA  NA  NA  23  45 115  37  NA  NA  NA
##  [35]  NA  NA  NA  29  NA  71  39  NA  NA  23  NA  NA  21  37  20  12  13
##  [52]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 135  49  32  NA  64  40  77
##  [69]  97  97  85  NA  10  27  NA   7  48  35  61  79  63  16  NA  NA  80
##  [86] 108  20  52  82  50  64  59  39   9  16  78  35  66 122  89 110  NA
## [103]  NA  44  28  65  NA  22  59  23  31  44  21   9  NA  45 168  73  NA
## [120]  76 118  84  85  96  78  73  91  47  32  20  23  21  24  44  21  28
## [137]   9  13  46  18  13  24  16  13  23  36   7  14  30  NA  14  18  20
```

Κενές τιμές (missing values)

- **NA** που στην R αντιστοιχεί στο αρκτικόλεξο “**Non-assigned**”.
- Η τιμή NA είναι ο τρόπος με τον οποίο η R αποδίδει τις τιμές που λείπουν από ένα σύνολο δεδομένων, έτσι ώστε να διατηρείται η δομή των πινάκων και των διανυσμάτων ακόμα κι αν στη θέση κάποιων στοιχείων δεν έχει αποδοθεί τιμή.
- Η ύπαρξη της NA είναι πολύ χρήσιμη για τη δομή των αντικειμένων, ωστόσο μπορεί να δημιουργήσει προβλήματα στην εφαρμογή ακόμα και απλών συναρτήσεων.

Hide

```
sum(airquality$Ozone)
```

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

Hide

```
sum(airquality$Ozone, na.rm=T)
```

```
## [1] 4887
```

Κενές τιμές (missing values)

Ένας εύκολος τρόπος να δούμε ποιες τιμές σε ένα διάνυσμα είναι κενές είναι η συνάρτηση **is.na()**

Hide

```
is.na(airquality$Ozone)
```

```
##   [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE
##  [45]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

Συναρτήσεις Ελέγχου Τιμών

Συνάρτηση	Έλεγχος
is.na	Ύπαρξη τιμής
is.infinite	Προσδιορισμένη τιμή
is.nan	Τιμή που απειρίζεται
is.integer	Η τιμή είναι ακέραιος
is.character	Η τιμή είναι χαρακτήρας
is.numeric	Η τιμή είναι αριθμητικό
is.factor	Η τιμή είναι παράγοντας
is.element	Σύγκριση συνόλων

Συναρτήσεις Ελέγχου Τιμών

- **is.na()**: όπως είδαμε παραπάνω ελέγχει την ύπαρξη τιμής
- **is.finite(), is.nan()**: ελέγχουν κατά πόσο σε ένα διάνυσμα μια τιμή έχει αποδοθεί ως άπειρο, πραγματικός αριθμός ή αλφαριθμητικό
- **is.numeric(), is.integer(), is.character()**: είναι συναρτήσεις που ελέγχουν τον τύπο της μεταβλητής που εξετάζεται.
- **is.factor()** ελέγχει μια ειδική κατηγορία δεδομένων που ονομάζονται παράγοντες (factors). Οι παράγοντες είναι κατηγορικές μεταβλητές που η R χρησιμοποιεί για να οργανώσει πιο πολύπλοκα αντικείμενα.

Hide

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Hide

```
is.factor(iris$Species)
```

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

Παραγοντές (factors)

Το πλήθος των μοναδικών τιμών μπορεί κανείς να δει με τη χρήση της συνάρτησης **levels()** η οποία τις επιστρέφει σε ένα διάνυσμα χαρακτήρων

Hide

```
levels(iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

Hide

```
a_vector<-c(rep("X",5),rep("Y",3), rep("Z",1))
a_factor<-factor(a_vector)
levels(a_factor)
```

```
## [1] "X" "Y" "Z"
```

Δημιουργία και χειρισμός επιπέδων (levels) σε παράγοντες

Όταν παίρνουμε τα επίπεδα ενός παράγοντα, αυτά οργανώνονται πάντοτε με αλφαβητική σειρά.

Hide

```
# Data in vectors
income <- c(32,51,12,23,26,27,24,26,25,18,30,22,24,21,25,27,18)
age <- c(48,59,26,23,37,32,20,25,45,55,44,42,51,30,35,29,19)
education <- c("middle","high","basic","high","high","middle","high","high","middle","basic", "high"
, "basic", "basic", "middle", "middle", "middle", "basic")
# Create the dataframe.
income_data <- data.frame(income,age,education)
levels(income_data$education)
```

```
## [1] "basic"  "high"   "middle"
```

	income	age	education
1	32	48	middle
2	51	59	high
3	12	26	basic
4	23	23	high
5	26	37	high
6	27	32	middle
7	24	20	high
8	26	25	high
9	25	45	middle
10	18	55	basic
11	30	44	high
12	22	42	basic
13	24	51	basic
14	21	30	middle
15	25	35	middle
16	27	29	middle
17	18	19	basic

Hide

```
income_data$education<-factor(income_data$education, levels=c("basic", "middle", "high"))
levels(income_data$education)
```

```
## [1] "basic"  "middle" "high"
```

Χρήση Παραγόντων σε Πλαίσια Δεδομένων

- Η βασική χρησιμότητα των παραγόντων είναι η κατηγοριοποίηση των δεδομένων σε ένα dataframe.
- Η R αναγνωρίζει τις μεταβλητές που μπορούν να χρησιμοποιηθούν ως παράγοντες και κατατάσσει τις υπόλοιπες μεταβλητές με βάση τα επίπεδά τους πριν κάνει υπολογισμούς σε διανυσματικό επίπεδο.
- Χαρακτηριστικές συναρτήσεις που επιτρέπουν υπολογισμούς αυτού του είδους είναι η **by()** και η **aggregate()**.

Hide

```
by(income_data$income, income_data$education, mean)
```

```
## income_data$education: basic
## [1] 18.8
## ------------------------------------------------------------
## income_data$education: middle
## [1] 26.16667
## ------------------------------------------------------------
## income_data$education: high
## [1] 30
```

Η **by(a,b,c)** ορίζει αρχικά το **διάνυσμα** στο οποίο θέλει να πραγματοποιήσει τον υπολογισμό, στη δεύτερη θέση το **διάνυσμα παραγόντων** με το οποίο θα κάνει την κατηγοριοποίηση και στην τρίτη θέση τη **συνάρτηση** που θα εφαρμοστεί για τους υπολογισμούς.

Χρήση Παραγόντων σε Πλαίσια Δεδομένων

Αντίστοιχη λειτουργία πραγματοποιεί η **aggregate()** με τη βασική διαφορά ότι τα διανύσματα παραγόντων που θα χρησιμοποιηθούν δίνονται με τη μορφή λίστας

Hide

```
aggregate(income_data$age, by=list(Education=education), mean)
```

```
##   Education        x
## 1     basic 38.60000
## 2      high 34.66667
## 3    middle 36.50000
```

Ένα σημαντικό πλεονέκτημα της **aggregate()** είναι ότι μπορεί να εφαρμοστεί σε συνδυασμούς παραγόντων που οδηγούν σε πιο πολύπλοκες κατηγοριοποιήσεις.
Η R επιτρέπει την κατηγοριοποίηση μέσω πολλαπλών παραγόντων με τη χρήση της συνάρτησης **table()**.
Η συγκεκριμένη συνάρτηση δέχεται ως είσοδο δύο ή περισσότερα διανύσματα παραγόντων και δημιουργεί έναν πίνακα “σύμπτωσης” (contingency table) που περιέχει τον αριθμό των στοιχείων των συνδυασμών τους.

Χρήση Παραγόντων σε Πλαίσια Δεδομένων

Το dataframe περιέχει τώρα δύο διανύσματα παραγόντων τα **education** και **AgeClass**. Η **table()** μας βοηθάει να απαντήσουμε σε ερωτήματα του τύπου “πόσοι συνδυασμοί ηλικακών ομάδων και μορφωτικών επιπέδων υπάρχουν;” ή “πόσα άτομα ανήκουν στο υψηλότερο μορφωτικό επίπεδο ενώ είναι νέοι;

Hide

```
ageclass<-c("higher","higher","lower","lower","higher","lower","lower", "lower","higher","higher","h
igher","higher","higher","lower","higher","lower",  "lower")
income_data<-data.frame(income_data, AgeClass=ageclass)
str(income_data)
```

```
## 'data.frame':    17 obs. of  4 variables:
##  $ income   : num  32 51 12 23 26 27 24 26 25 18 ...
##  $ age      : num  48 59 26 23 37 32 20 25 45 55 ...
##  $ education: Factor w/ 3 levels "basic","middle",..: 2 3 1 3 3 2 3 3 2 1 ...
##  $ AgeClass : Factor w/ 2 levels "higher","lower": 1 1 2 2 1 2 2 2 1 1 ...
```

	income	age	education	AgeClass
1	32	48	middle	higher
2	51	59	high	higher
3	12	26	basic	lower
4	23	23	high	lower
5	26	37	high	higher
6	27	32	middle	lower
7	24	20	high	lower
8	26	25	high	lower
9	25	45	middle	higher
10	18	55	basic	higher
11	30	44	high	higher
12	22	42	basic	higher
13	24	51	basic	higher
14	21	30	middle	lower
15	25	35	middle	higher
16	27	29	middle	lower
17	18	19	basic	lower

Hide

```
table(income_data$education, income_data$AgeClass)
```

```
##
##          higher lower
##   basic       3     2
##   middle      3     3
##   high        3     3
```

Χρήση Παραγόντων σε Πλαίσια Δεδομένων

Μπορούμε να χρησιμοποιήσουμε την **aggregate()** για να υπολογίσουμε χαρακτηριστικές τιμές για αυτούς τους συνδυασμούς, κάτι που δεν μπορούμε να κάνουμε με την **by().**

Hide

```
aggregate(income_data$income, by=list(Education=education, Age=ageclass), mean)
```

```
##   Education    Age        x
## 1     basic higher 21.33333
## 2      high higher 35.66667
## 3    middle higher 27.33333
## 4     basic  lower 15.00000
## 5      high  lower 24.33333
## 6    middle  lower 25.00000
```

Δημιουργία Υποσυνόλων (subsetting)

Η συνάρτηση **summary()** μας δινει κάποια βασικά στατιστικά χαρακτηριστικά των δεδομένων που περιέχει

Hide

```
summary(income_data)
```

```
##      income           age          education   AgeClass
##  Min.   :12.00   Min.   :19.00   basic :5   higher:9
##  1st Qu.:22.00   1st Qu.:26.00   middle:6   lower :8
##  Median :25.00   Median :35.00   high  :6
##  Mean   :25.35   Mean   :36.47
##  3rd Qu.:27.00   3rd Qu.:45.00
##  Max.   :51.00   Max.   :59.00
```

Δημιουργία υποσυνόλων με απλούς ελέγχους

Hide

```
income_data$age<=30
```

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

Hide

```
income_data$age<=30->x
income_data$age[x]
```

```
## [1] 26 23 20 25 30 29 19
```

Hide

```
income_data$age[income_data$age<=30]
```

```
## [1] 26 23 20 25 30 29 19
```

Υποσύνολα με συνδυασμούς ελέγχων

Τελεστής	Πράξη
==	Έλεγχος Ισότητας
!=	Έλεγχος Διαφοράς
>	Μεγαλύτερο από
<	Μικρότερο από
>=	Μεγαλύτερο ή ίσο
<=	Μικρότερο ή ίσο
&	Σύνδεση ενδεχομένων (ΚΑΙ)
|	Σύνδεση ενδεχομένων (Ή)
!	Συμπλήρωμα ενδεχομένου (άρνηση)

Υποσύνολα με συνδυασμούς ελέγχων

Τελεστής	Πράξη
==	Έλεγχος Ισότητας
!=	Έλεγχος Διαφοράς
>	Μεγαλύτερο από
<	Μικρότερο από
>=	Μεγαλύτερο ή ίσο
<=	Μικρότερο ή ίσο
&	Σύνδεση ενδεχομένων (ΚΑΙ)
|	Σύνδεση ενδεχομένων (Ή)
!	Συμπλήρωμα ενδεχομένου (άρνηση)

Hide

```
(income_data$age<=30 & income_data$education=="basic")->x
income_data$income[x]
```

```
## [1] 12 18
```

Υποσύνολα με συνδυασμούς ελέγχων

Ένας εύκολος και πολύ ευέλικτος τρόπος να συνδυάσουμε λογικούς/αριθμητικούς ελέγχους με ταυτόχρονη λήψη του υποσυνόλου είναι η συνάρτηση **subset()** η οποία δρα πάνω σε ένα dataframe με ταυτόχρονο "πέρασμα" των ελέγχων σαν δεύτερη παράμετρο.

Hide

```
subset(income_data, age<=30)
```

```
##    income age education AgeClass
## 3      12  26     basic    lower
## 4      23  23      high    lower
## 7      24  20      high    lower
## 8      26  25      high    lower
## 14     21  30    middle    lower
## 16     27  29    middle    lower
## 17     18  19     basic    lower
```


Υποσύνολα με συνδυασμούς ελέγχων

Ένας εύκολος και πολύ ευέλικτος τρόπος να συνδυάσουμε λογικούς/αριθμητικούς ελέγχους με ταυτόχρονη λήψη του υποσυνόλου είναι η συνάρτηση **subset()** η οποία δρα πάνω σε ένα dataframe με ταυτόχρονο "πέρασμα" των ελέγχων σαν δεύτερη παράμετρο.

Hide

```
subset(income_data, age<=30, c(income, education))
```

```
##    income education
## 3      12     basic
## 4      23      high
## 7      24      high
## 8      26      high
## 14     21    middle
## 16     27    middle
## 17     18     basic
```

Υποσύνολα με συνδυασμούς ελέγχων

Ένας εύκολος και πολύ ευέλικτος τρόπος να συνδυάσουμε λογικούς/αριθμητικούς ελέγχους με ταυτόχρονη λήψη του υποσυνόλου είναι η συνάρτηση **subset()** η οποία δρα πάνω σε ένα dataframe με ταυτόχρονο "πέρασμα" των ελέγχων σαν δεύτερη παράμετρο.

Hide

```
subset(income_data, age<=30, -c(age, AgeClass))
```

```
##    income education
## 3      12     basic
## 4      23      high
## 7      24      high
## 8      26      high
## 14     21    middle
## 16     27    middle
## 17     18     basic
```

Υποσύνολα με συνδυασμούς ελέγχων

Ένας εύκολος και πολύ ευέλικτος τρόπος να συνδυάσουμε λογικούς/αριθμητικούς ελέγχους με ταυτόχρονη λήψη του υποσυνόλου είναι η συνάρτηση **subset()** η οποία δρα πάνω σε ένα dataframe με ταυτόχρονο "πέρασμα" των ελέγχων σαν δεύτερη παράμετρο.

Hide

```
subset(income_data, age<=30 & education=="high", c(AgeClass, income, education))
```

```
##   AgeClass income education
## 4    lower     23      high
## 7    lower     24      high
## 8    lower     26      high
```

Λήψη θέσεων στοιχείων υποσυνόλων με τη συνάρτηση which()

Η χρήση των ελέγχων για τη δημιουργία διανυσμάτων λογικών τιμών (TRUE/FALSE) όπως αναλύθηκε στις προηγούμενες ενότητες έχει έναν βασικό περιορισμό. Αυτός είναι ότι δεν επιτρέπει τον εύκολο χειρισμό των υποσυνόλων για περαιτέρω υπολογισμούς πάνω στο ίδιο dataframe.

Για να καταλάβετε καλύτερα το πρόβλημα φανταστείτε το εξής ερώτημα: Θέλουμε να εντοπίσουμε το υποσύνολο των ατόμων με εισόδημα κάτω από 20 και να προσθέσουμε στις τιμές εισοδήματος ένα bonus (+2). Πώς θα κάνουμε κάτι τέτοιο;

Με την προσέγγιση λογικού ελέγχου, το διάνυσμα που επιστρέφεται μας επιτρέπει να πάρουμε τις τιμές που επιβεβαιώνουν τον έλεγχο αλλά δεν μπορούμε, (τουλάχιστον όχι με κάποιον απευθείας τρόπο) να χειριστούμε τις τιμές αυτές μέσα στο dataframe.

Για να κάνουμε κάτι τέτοιο θα χρειαστούμε τις σχετικές θέσεις των στοιχείων μέσα στο πλαίσιο δεδομένων. Αυτό επιτυγχάνεται με το συνδυασμό λογικών ελέγχων στο πλαίσιο της συνάρτησης **which().**

Λήψη θέσεων στοιχείων υποσυνόλων με τη συνάρτηση which()

Hide

```
which(income_data$income<=20)->i
i
```

```
## [1]  3 10 17
```

Hide

```
income_data$income;
```

```
##  [1] 32 51 12 23 26 27 24 26 25 18 30 22 24 21 25 27 18
```

Hide

```
income_data$income[i]<-income_data$income[i]+2
income_data$income;
```

```
##  [1] 32 51 14 23 26 27 24 26 25 20 30 22 24 21 25 27 20
```

Λήψη θέσεων στοιχείων υποσυνόλων με τη συνάρτηση which()

Ας οργανώσουμε δηλαδή το dataframe με έναν ηλικιακό παράγοντα δύο επιπέδων (higher, lower) με όριο όμως τα 30, αντί για τα 35 χρόνια.
Πριν την αλλαγή

Hide

```
table(income_data$AgeClass)
```

```
##
## higher  lower
##      9      8
```

Λήψη θέσεων στοιχείων υποσυνόλων με τη συνάρτηση which()

Hide

```
which(income_data$age<=30)->lowage
which(income_data$age>30)->highage
```

Hide

```
income_data$AgeClass[lowage]<-"lower"
income_data$AgeClass[highage]<-"higher"
```

Hide

```
table(income_data$AgeClass)
```

```
##
## higher  lower
##     10      7
```

Ύλη

Ανάλυση δεδομένων με την R

Κεφάλαιο 4