R Exercises

Noah Cheruiyot Mutai

October 1, 2016

Contents

Vectors

Exercises

Exercise 1

Using the seq() function, generate the sequence 2, 5, 8, 11.

Exercise 2

Use the seq() function to generate the sequence 9, 18, 27, 36, 45.

Exercise 3

Generate the sequence 9, 18, 27, 36, 45, 54, 63, 72, 81, 90 using the length.out parameter.

Exercise 4

For this exercise, first write down your answer, without using R. Then, check your answer using R.

What is the output for the code:

seq(from = -10, to = 10, length.out = 5)

Exercise 5

Assign value 5 to variable x. Write code 1:x-1 you should get 0, 1, 2, 3, 4. Write code 1 : (x-1) you will get 1, 2, 3, 4. Explain the discrepancy in the output.

Exercise 6

For this exercise, first write down your answer, without using R. Then, check your answer using R. Create a vector a with values 1, 2, 3, 4 For the code seq(along.with = a), what will be the output?

Exercise 7 For this exercise, first write down your answer, without using R. Then, check your answer using R. Generate a sequence using the below code. seq(from=1, to=4, by=1) What other ways can you generate the same sequence?

Exercise 8 Generate a backward sequence from 5, 4, 3, 2, 1

Exercise 9 Assign x <- c(1, 2, 3, 4) Using the function rep() , create the below sequence 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4

Exercise 10 Assign x <- c(1, 2, 3, 4) Using the rep() function generate the sequence: 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4

Solutions

```r
# Excercise 1
# Using the seq() function generate the sequence 2,5,8,11
seq(from = 2,to = 13,by = 3)
```

```
## [1]  2  5  8 11
```

```r
# Exercise 2:
# Use the seq() function to generate the sequence 9,18,27,36,45
seq(from = 9, to = 45, by = 9)
```

```
## [1]  9 18 27 36 45
```

```r
# Exercise 3:
# Generate the sequence 9,18,27,36,45,54,63,72,81,90 using the length.out parameter
seq(from = 9, to = 90, length.out = 10)
```

```
##  [1]  9 18 27 36 45 54 63 72 81 90
```

```r
# Exercise 4:
# What is the output for the code
seq(from = -10, to = 10, length.out = 5) # The ouput will be -10,-5,0,5,10
```

```
## [1] -10  -5   0   5  10
```

```r
# Exercise 5:
# Assign value 5 to variable x.
# Write code 1:x-1 you should get 0,1,2,3,4
# Write code 1 : (x-1) you will get 1,2,3,4.
# Explain the discrepancy in the output
x <- 5
1:x-1
```

```
## [1] 0 1 2 3 4
```

```r
1:(x-1)
```

```
## [1] 1 2 3 4
```

```r
# Exercise 6:
# Create a vector a with values 1,2,3,4
# For the code seq(along.with = a)
# What will be the output?
a <- c(1,2,3,4) # Creates vector a with values 1,2,3,4
seq(along.with = a) # ouput is 1,2,3,4
```

```
## [1] 1 2 3 4
```

```r
# Exercise 7:
# Generate a sequence using the below code.
seq(from=1, to=4, by=1)
```

```
## [1] 1 2 3 4
```

```r
# What other ways can you generate the same sequence?
# a) X <- 1:4
# b) X <- seq(4)
# c) X <- c(1,2,3,4)
# d) All the above
x <- 1:4
x <- seq(4)
x <- c(1,2,3,4)
# Solution d) All the above
# Exercise 8:
# Generate a backward sequence from  5,4,3,2,1
seq(from =5, to =1, by = -1)
```

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

```r
# Exercise 9:
# Assign x <- c(1,2,3,4)
# Using the function rep() create the below sequence
# 1,2,3,4,1,2,3,4,1,2,3,4
x <- c(1,2,3,4)
rep(x, times = 3)
```

```
##  [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

```r
# Exercise 10:
# Assign x<- c(1,2,3,4)
# Using the rep() function generate the sequence 1,1,1,2,2,2,3,3,3,4,4,4
x <- c(1,2,3,4)
rep(x, each = 3)
```

```
##  [1] 1 1 1 2 2 2 3 3 3 4 4 4
```

```
##  [1] 1 1 1 2 2 2 3 3 3 4 4 4
```

Regular sequences

Exercises

Excercise 1

Using the seq() function, generate the sequence 2, 5, 8, 11.

Exercise 2

Use the seq() function to generate the sequence 9, 18, 27, 36, 45.

Exercise 3

Generate the sequence 9, 18, 27, 36, 45, 54, 63, 72, 81, 90 using the length.out parameter.

Exercise 4

For this exercise, first write down your answer, without using R. Then, check your answer using R.

What is the output for the code:

```
seq(from = -10, to = 10, length.out = 5)
```

```
## [1] -10  -5   0   5  10
```

Exercise 5

Assign value 5 to variable x.

Write code 1:x-1 you should get 0, 1, 2, 3, 4.

Write code 1 : (x-1) you will get 1, 2, 3, 4.

Explain the discrepancy in the output.

Exercise 6

For this exercise, first write down your answer, without using R. Then, check your answer using R.

Create a vector a with values 1, 2, 3, 4

For the code seq(along.with = a), what will be the output?

Exercise 7

For this exercise, first write down your answer, without using R. Then, check your answer using R.

Generate a sequence using the below code.

seq(from=1, to=4, by=1)

What other ways can you generate the same sequence?

Exercise 8

Generate a backward sequence from 5, 4, 3, 2, 1

Exercise 9

Assign x <- c(1, 2, 3, 4)

Using the function rep() , create the below sequence

1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4 Exercise 10

Assign x <- c(1, 2, 3, 4)

Using the rep() function generate the sequence:

1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4

Solutions

Excercise 1 Using the seq() function generate the sequence 2,5,8,11

```
seq(from = 2,to = 13,by = 3)
```

```
## [1]  2  5  8 11
```

Exercise 2: Use the seq() function to generate the sequence 9,18,27,36,45

```
seq(from = 9, to = 45, by = 9)
```

```
## [1]  9 18 27 36 45
```

Exercise 3: Generate the sequence 9,18,27,36,45,54,63,72,81,90 using the length.out parameter

```
seq(from = 9, to = 90, length.out = 10)
```

```
##  [1]  9 18 27 36 45 54 63 72 81 90
```

Exercise 4: What is the output for the code

```
seq(from = -10, to = 10, length.out = 5) # The ouput will be -10,-5,0,5,10
```

```
## [1] -10  -5   0   5  10
```

Exercise 5: Assign value 5 to variable x. Write code 1:x-1 you should get 0,1,2,3,4 Write code 1 : (x-1) you will get 1,2,3,4. Explain the discrepancy in the output

```
x <- 5
1:x-1
```

```
## [1] 0 1 2 3 4
```

```
1:(x-1)
```

```
## [1] 1 2 3 4
```

Exercise 6: Create a vector a with values 1,2,3,4 For the code seq(along.with = a) What will be the output?

```
a <- c(1,2,3,4) # Creates vector a with values 1,2,3,4
```

```
seq(along.with = a) # ouput is 1,2,3,4
```

```
## [1] 1 2 3 4
```

Exercise 7: Generate a sequence using the below code.

```r
seq(from=1, to=4, by=1)
```

```
## [1] 1 2 3 4
```

What other ways can you generate the same sequence?

```r
x <- 1:4
x <- seq(4)
x <- c(1,2,3,4)
```

Exercise 8: Generate a backward sequence from 5,4,3,2,1

```r
seq(from =5, to =1, by = -1)
```

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

Exercise 9: Assign x <- c(1,2,3,4) Using the function rep() create the below sequence 1,2,3,4,1,2,3,4,1,2,3,4

```r
x <- c(1,2,3,4)
rep(x, times = 3)
```

```
##  [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

Exercise 10: Assign x<- c(1,2,3,4) Using the rep() function generate the sequence 1,1,1,2,2,2,3,3,3,4,4,4

```r
x <- c(1,2,3,4)
rep(x, each = 3)
```

```
##  [1] 1 1 1 2 2 2 3 3 3 4 4 4
```

Logical vectors and operators

Exercises

Before you start, enter the following code:

```
data <- mtcars
```

Exercise 1

Use logical operators to output only those rows of data where column mpg is between 15 and 20 (excluding 15 and 20).

Exercise 2

Use logical operators to output only those rows of data where column cyl is equal to 6 and column am is not 0.

Exercise 3

Use logical operators to output only those rows of data where column gear or carb has the value 4.

Exercise 4

Use logical operators to output only the even rows of data.

Exercise 5

Use logical operators and change every fourth element in column mpg to 0.

Exercise 6

Output only those rows of data where columns vs and am have the same value 1, solve this without using == operator.

Exercise 7 (TRUE + TRUE) * FALSE , what does this expression evaluate to and why?

Exercise 8

Output only those rows of data where at least vs or am have the value 1, solve this without using == or !=.

Exercise 9

Explain the difference between | , || , & and &&.

Exercise 10 Change all values that are 0 in the column am in data to 2.

Exercise 11

Add 2 to every element in the column vs without using numbers.

Exercise 12

Output only those rows of data where vs and am have different values, solve this without using == or !=.

Solutions

```
data <- mtcars
##Q1
```

```
data[data$mpg > 15 & data$mpg < 20,]
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
## Q2
data[data$cyl == 6 & data$am != 0,]
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Ferrari Dino  19.7   6  145 175 3.62 2.770 15.50  0  1    5    6
```

```
## Q3
data[data$gear == 4 | data$carb == 4,]
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
## Q4
data[c(F,T),]
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
## Q5
data$mpg[c(F,F,F,T)] <- 0
```

```r
## Q6
data[data$vs & data$am,]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla  0.0   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Lotus Europa    0.0   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E      0.0   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
## Q7
#the answer is 0 because when R encounters a numeric operator "+", it coerces
#TRUE to 1 and FALSE to 0
```

```r
## Q8
data[data$vs | data$am,]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive  0.0   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 240D       0.0   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

```
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla  0.0   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa    0.0   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E      0.0   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
## Q9
#&& evaluates only one of the condition and only if it's TRUE will it
#evaluate the second condition. & on the other hand evaluates both expressions
#and compares them. | and || have the same difference as the above
#so if there is a condition as 3<4 || 7/"b" it wont flag an error as 3<4 is TRUE so
#7/"b" is never evaluated. However 3<4 | 7/"b" will flag an error as you cannot
#divide by a character. && is generally used in control flow (ifelse) and & is used
#in vectorization
```

```
## Q10
data$am[data$am == 0] <- 2
```

```
## Q11
data$vs <- data$vs + 2*(TRUE)
```

```
## Q12
data[xor(data$vs,data$am),]
```

```
##  [1] mpg  cyl  disp hp   drat wt   qsec vs   am   gear carb
## <0 rows> (or 0-length row.names)
```

Missing values

Exercises

Exercise 1

If X <- c (22,3,7,NA,NA,67) what will be the output for the R statement length(X)

Exercise 2

If X = c(NA,3,14,NA,33,17,NA,41) write some R code that will remove all occurrences of NA in X.

a. X[!is.na(X)]
b. X[is.na(X)]
c. X[X==NA]= 0

Exercise 3

If Y = c(1,3,12,NA,33,7,NA,21) what R statement will replace all occurrences of NA with 11?

a. Y[Y==NA]= 11
b. Y[is.na(Y)]= 11
c. Y[Y==11] = NA

Exercise 4

If X = c(34,33,65,37,89,NA,43,NA,11,NA,23,NA) then what will count the number of occurrences of NA in X?

a. sum(X==NA)
b. sum(X == NA, is.na(X))
c. sum(is.na(X))

Exercise 5

Consider the following vector W <- c (11, 3, 5, NA, 6) Write some R code that will return TRUE for value of W missing in the vector.

Exercise 6

Load 'Orange' dataset from R using the command data(Orange) . Replace all values of age=118 to NA.

Exercise 7

Consider the following vector A <- c (33, 21, 12, NA, 7, 8) . Write some R code that will calculate the mean of A without the missing value.

Exercise 8

Let:

c1 <- c(1,2,3,NA) ;

c2 <- c(2,4,6,89) ;

c3 <- c(45,NA,66,101).

If X <- rbind (c1,c2,c3, deparse.level=1) , write a code that will display all rows with missing values.

Exercise 9

Consider the following data obtained from df <- data.frame (Name = c(NA, “Joseph”, “Martin”, NA, “Andrea”), Sales = c(15, 18, 21, 56, 60), Price = c(34, 52, 21, 44, 20), stringsAsFactors = FALSE) Write some R code that will return a data frame which removes all rows with NA values in Name column

Exercise 10

Consider the following data obtained from df <- data.frame(Name = c(NA, “Joseph”, “Martin”, NA, “Andrea”), Sales = c(15, 18, 21, NA, 60), Price = c(34, 52, 33, 44, NA), stringsAsFactors = FALSE) Write some R code that will remove all rows with NA values and give the following output

Name Sales Price 2 Joseph 18 52 3 Martin 21 33

Solutions

Exercise 1

Answer: 6

Exercise 2

Answer: a

Exercise 3

Answer: b

Exercise 4

Answer: C

Exercise 5

W <- c (11, 3, 5, NA, 6) is.na(W)

Exercise 6

data(Orange) Orange*age*[*Orange*age == 118] <- NA

Exercise 7

A <- c (33, 21, 12, NA, 7, 8) mean(A, na.rm = TRUE)

Exercise 8

c1 <- c(1,2,3,NA) c2 <- c(2,4,6,89) c3 <- c(45,NA,66,101) X <- rbind (c1,c2,c3, deparse.level=1) X[!complete.cases(X),]

Exercise 9

df <- data.frame(Name = c(NA, “Joseph”, “Martin”, NA, “Andrea”), Sales = c(15, 18, 21, 56, 60), Price = c(34, 52, 21, 44, 20), stringsAsFactors = FALSE) df[!is.na(df$Name),]

Exercise 10

df <- data.frame(Name = c(NA, “Joseph”, “Martin”, NA, “Andrea”), Sales = c(15, 18, 21, NA, 60), Price = c(34, 52, 33, 44, NA), stringsAsFactors = FALSE) df[!(is.na(df*Name*)|*is.na*(*df*Sales)| is.na(df$Price)),]

Character vector

Exercises

Exercise 1

If x <- “Good Morning!”, find out the number of characters in X

Exercise 2

Consider the character vector x <- c (“Nature’s”, “Best”), how many characters are there in x?

Exercise 3

If x <- c(“Nature’s”," At its best “) , how many characters are there in x?

Exercise 4

If fname <- “James” and lname <- “Bond”, write some R code that will produce the output “James Bond”.

Exercise 5

If m <- “Capital of America is Washington” then extract the string “Capital of America” from the character vector m.

Exercise 6

Write some R code to replace the first occurrence of the word “failed” with “failure” in the string “Success is not final, failed is not fatal”.

Exercise 7

Consider two character vectors: Names <- c(“John”, “Andrew”, “Thomas”) and Designation <- c(“Manager”, “Project Head”, “Marketing Head”). Write some R code to obtain the following output. Names Designation 1 John Manager 2 Andrew Project Head 3 Thomas Marketing Head

Exercise 8

Write some R code that will initialise a character vector with fixed length of 10.

Exercise 9

Write some R code that will generate a vector with the following elements, without using loops. “aa” “ba” “ca” “da” “ea” “ab” “bb” “cb” “db” “eb” “ac” “bc” “cc” “dc” “ec” “ad” “bd” “cd” “dd” “ed” “ae” “be” “ce” “de” “ee”

Exercise 10

Let df <- data.frame(Date = c(“12/12/2000 12:11:10”)) . Write some R code that will convert the given date to character values and gives the following output: “2000-12-12 12:11:10 GMT”

Solutions

Exercise 1

```r
x <- "Good Morning!"
nchar(x)
```

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

Exercise 2

```r
x <- c ("Nature's", "Best ")
nchar(x)
```

```
## [1] 8 5
```

Exercise 3

```r
x <- c("Nature's"," At   its best ")
nchar(x)
```

```
## [1]  8 15
```

Exercise 4

```r
fname <- "James"
lname <- "Bond"
paste(fname, lname)
```

```
## [1] "James Bond"
```

Exercise 5

```r
m <- "Capital of America is Washington"
substr(m, start=1, stop=18)
```

```
## [1] "Capital of America"
```

Exercise 6

```r
x <- "Success is not final, failed is not fatal"
sub("failed", "failure", x)
```

```
## [1] "Success is not final, failure is not fatal"
```

Exercise 7

```r
Names <- c("John", "Andrew", "Thomas")
Designation <- c("Manager", "Project Head", "Marketing Head")
data.frame(Names, Designation)
```

```
##    Names    Designation
## 1   John        Manager
## 2 Andrew   Project Head
## 3 Thomas Marketing Head
```

Exercise 8

```r
vector(mode="character", length=10)
```

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

Exercise 9

```r
c(outer(letters[1:5], letters[1:5], FUN=paste, sep=""))
```

```
##  [1] "aa" "ba" "ca" "da" "ea" "ab" "bb" "cb" "db" "eb" "ac" "bc" "cc" "dc"
## [15] "ec" "ad" "bd" "cd" "dd" "ed" "ae" "be" "ce" "de" "ee"
```

Exercise 10

```r
df <- data.frame(Date = c("12/12/2000 12:11:10"))
strptime(df$Date, "%m/%d/%Y %H:%M:%S")
```

```
## [1] "2000-12-12 12:11:10 EAT"
```

Index Vectors

Exercises

Exercise 1 If x <- c("ww", "ee", "ff", "uu", "kk"), what will be the output for x[c(2,3)]?

a. "ee", "ff"
b. "ee"
c. "ff"

Exercise 2 If x <- c("ss", "aa", "ff", "kk", "bb"), what will be the third value in the index vector operation x[c(2, 4, 4)]?

a. "uu"
b. NA
c. "kk"

Exercise 3 If x <- c("pp", "aa", "gg", "kk", "bb"), what will be the fourth value in the index vector operation x[-2]? a. "aa" b. "gg" c. "bb"

Exercise 4 Let a <- c(2, 4, 6, 8) and b <- c(TRUE, FALSE, TRUE, FALSE), what will be the output for the R expression max(a[b])?

Exercise 5 Let a <- c (3, 4, 7, 8) and b <- c(TRUE, TRUE, FALSE, FALSE), what will be the output for the R expression sum(a[b])?

Exercise 6 Write an R expression that will return the sum value of 10 for the vector x <- c(2, 1, 4, 2, 1, NA)

Exercise 7 If x <- c(1, 3, 5, 7, NA) write an r expression that will return the output 1, 3, 5, 7.

Exercise 8 Consider the data frame s <- data.frame(first= as.factor(c("x", "y", "a", "b", "x", "z")), second=c(2, 4, 6, 8, 10, 12)). Write an R statement that will return the output 2, 4, 10, by using the variable first as an index vector.

Exercise 9 What will be the output for the R expression (c(FALSE, TRUE)) || (c(TRUE, TRUE))?

Exercise 10 Write an R expression that will return the positions of 3 and 7 in the vector x <- c(1, 3, 6, 7, 3, 7, 8, 9, 3, 7, 2).

Solutions

```r
#Exercise 1
x <- c("ww", "ee", "ff", "uu", "kk")
x[c(2, 3)]
```

```
## [1] "ee" "ff"
```

```r
#(Answer: a)

#Exercise 2
x <- c("ss", "aa", "ff", "kk", "bb")
y <- x[c(2, 4, 4)]
y[3]
```

```
## [1] "kk"
```

```r
#(Answer: c)

#Exercise 3
x <- c("pp", "aa", "gg", "kk", "bb")
y <- x[-2]
y[4]
```

```
## [1] "bb"
```

```r
#(Answer:  c)

#Exercise 4
a <- c(2, 4, 6, 8)
b <- c(TRUE, FALSE, TRUE, FALSE)
max(a[b])
```

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

```r
#(Answer: 6)

#Exercise 5
a <- c(3, 4, 7, 8)
b <- c(TRUE, TRUE, FALSE, FALSE)
sum(a[b])
```

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

```r
#(Answer: 7)

#Exercise 6
x <- c(2, 1, 4, 2, 1, NA)
sum(x, na.rm=TRUE)
```

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

```r
sum(x[-6])  # alternative solution
```

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

```r
#Exercise 7
x <- c(1, 3, 5, 7, NA)
x[!is.na(x)]
```

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

```r
x[-5]     # alternative solution
```

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

```r
#Exercise 8
s <- data.frame(first= as.factor(c("x", "y", "a", "b", "x", "z")), second=c(2, 4, 6, 8, 10, 12))
s$second[(s$first=='x') | (s$first=='y')]
```

```
## [1]  2  4 10
```

```r
s$second[s$first %in% c('x', 'y')]   # alternative solution
```

```
## [1]  2  4 10
```

```r
#Exercise 9
(c(FALSE, TRUE)) || (c(TRUE, TRUE))
```

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

```r
#Exercise 10
x <- c(1, 3, 6, 7, 3, 7, 8, 9, 3, 7, 2)
which(x %in% c(3, 7))
```

```
## [1]  2  4  5  6  9 10
```

Object modes and attributes

Exercises

Exercise 1 What is the mode of the following objects? First write down the mode, without using R. Then confirm using an approriate R command.

a. c('a', 'b', 'c')
b. 3.32e16
c. 1/3
d. sqrt(-2i)

Exercise 2

What is the mode of the following objects? First, enter the name of the object at the prompt (R will show its contents), and try to infer the mode from what you see. Then enter an R command, such that R will print the mode on the screen. a. pressure b. lm c. rivers

Exercise 3

Consider the following list: x <- list(LETTERS, TRUE, print(1:10), print, 1:10) What is the mode of x, and each of its elements? First write down the mode, without using R. Then confirm using the appropriate R commands.

Exercise 4

Show whether the vector x <- 1:100 is of mode numeric (TRUE) or not (FALSE).

Exercise 5

Change the mode of the vector x <- 1:100 to character, with and without using the mode function. Write down the first 5 elements of the vector, after the mode conversion. Check your answer by printing the first 5 characters on the screen.

Exercise 6

Change the mode of the character vector you created in the previous exercise, back to numeric. Again, with and without using the mode function.

Exercise 7

Change the mode of the vector x <- c('1', '2', 'three') to numeric. First write down the new vector x, without using R, then check your answer using R.

Exercise 8

Change the mode of the vector x <- c(TRUE, TRUE, FALSE, TRUE) to numeric. First write down the new vector x, without using R, then check your answer using R.

Exercise 9

Consider the vector x <- c('1', '2', 'three'). What is the mode of y <- x + 1. First write down your answer without using R, then check using R.

Exercise 10

Create a vector y <- c('2', '4', '6') from the vector x <- c('1', '2', '3').

Exercise 11

Try to create some exercises yourself, on the mode topic. This is the best way to really master the subject… Feel free to share as a comment below, so we can all learn from it!

Solutions

```
# Exercise 1
mode(c('a', 'b', 'c'))
```

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

```
## [1] "character"
mode(3.32e16)
```

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

```
## [1] "numeric"
mode(1/3)
```

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

```
## [1] "numeric"
mode(sqrt(-2i))
```

```
## [1] "complex"
```

```
## [1] "complex"
# Exercise 2
mode(pressure)
```

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

```
## [1] "list"
mode(lm)
```

```
## [1] "function"
```

```
## [1] "function"
mode(rivers)
```

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

```
## [1] "numeric"
# Exercise 3
x <- list(LETTERS, TRUE, print(1:10), print, 1:10)
```

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

```
##  [1]  1  2  3  4  5  6  7  8  9 10
mode(x)
```

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

```
## [1] "list"
mode(x[[1]])
```

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

```
## [1] "character"
mode(x[[2]])
```

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

```
## [1] "logical"
mode(x[[3]])
```

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

```
## [1] "numeric"
mode(x[[4]])
```

```
## [1] "function"
```

```
## [1] "function"
mode(x[[5]])
```

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

```
## [1] "numeric"
sapply(x, mode)  # alternative to previous 5 statements
```

```
## [1] "character" "logical"   "numeric"   "function"  "numeric"
```

```
## [1] "character" "logical"   "numeric"   "function"  "numeric"
# Exercise 4
x <- 1:100
is.numeric(x)
```

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

```
## [1] TRUE
# Exercise 5
x <- 1:100
mode(x) <- 'character'  # using the mode function
x <- 1:100
x <- as.character(x)    # without using the mode function
x[1:5]                  # to check answer
```

```
## [1] "1" "2" "3" "4" "5"
```

```
## [1] "1" "2" "3" "4" "5"
# Exercise 6
mode(x) <- 'numeric'    # using the mode function
x <- as.numeric(x)      # without using the mode function

# Exercise 7
x <- c('1', '2', 'three')
as.numeric(x)
```

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

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

```
## Warning: NAs introduced by coercion
## [1]  1  2 NA
# Exercise 8
x <- c(TRUE, TRUE, FALSE, TRUE)
as.numeric(x)
```

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

```
## [1] 1 1 0 1
# Exercise 9
# The mode of y does not exist, because y is not defined,
# because the '+' operator does not accept a vector of
# mode character.

# Exercise 10
x <- c('1', '2', '3')
y <- as.numeric(x) * 2
mode(y) <- 'character'
y
```

```
## [1] "2" "4" "6"
```

```
## [1] "2" "4" "6"
```

Factors

Exercises

Exercise 1

If x = c(1, 2, 3, 3, 5, 3, 2, 4, NA), what are the levels of factor(x)?

a. 1, 2, 3, 4, 5
b. NA
c. 1, 2, 3, 4, 5, NA

Exercise 2

Let x <- c(11, 22, 47, 47, 11, 47, 11). If an R expression factor(x, levels=c(11, 22, 47), ordered=TRUE) is executed, what will be the 4th element in the output?

a. 11
b. 22
c. 47

Exercise 3

If z <- c("p", "a" , "g", "t", "b"), then which of the following R expressions will replace the third element in z with "b".

a. factor(z[3]) <- "b"
b. levels(z[3]) <- "b"
c. z[3] <- "b"

Exercise 4

If z <- factor(c("p", "q", "p", "r", "q")) and levels of z are "p", "q" ,"r", write an R expression that will change the level "p" to "w" so that z is equal to: "w", "q" , "w", "r" , "q".

Exercise 5

If:

s1 <- factor(sample(letters, size=5, replace=TRUE)) and s2 <- factor(sample(letters, size=5, replace=TRUE)), write an R expression that will concatenate s1 and s2 in a single factor with 10 elements.

Exercise 6

Consider the iris data set in R. Write an R expression that will 'cut' the Sepal.Length variable and create the following factor with five levels.

(4.3, 5.02] (5.02, 5.74] (5.74, 6.46] (6.46, 7.18] (7.18, 7.9] 32 41 42 24 11

Exercise 7

Consider again the iris data set. Write an R expression that will generate a two-way frequency table with two rows and three colums. The rows should relate to Sepal.length (less than 5: TRUE or FALSE) and columns to Species, with the following output:

setosa versicolor virginica FALSE 30 49 49 TRUE 20 1 1

Exercise 8

Consider the factor responses <- factor(c("Agree", "Agree", "Strongly Agree", "Disagree", "Agree")), with the following output:

[1] Agree Agree Strongly Agree Disagree Agree Levels: Agree Disagree Strongly Agree

Later it was found that new a level "Strongly Disagree" exists. Write an R expression that will include "strongly disagree" as new level attribute of the factor and returns the following output:

[1] Agree Agree Strongly Agree Disagree Agree Levels: Strongly Agree Agree Disagree Strongly Disagree

Exercise 9

Let x <- data.frame(q=c(2, 4, 6), p=c("a", "b", "c")). Write an R statement that will replace levels a, b, c with labels "fertiliser1", "fertliser2", "fertiliser3".

Exercise 10

If x <- factor(c("high", "low", "medium", "high", "high", "low", "medium")), write an R expression that will provide unique numeric values for various levels of x with the following output:

levels value 1 high 1 2 low 2 3 medium 3

Solutions

```r
# Exercise 1
x = c(1, 2, 3, 3, 5, 3, 2, 4, NA)
levels(factor(x))
```

```
## [1] "1" "2" "3" "4" "5"
```

```r
## [1] "1" "2" "3" "4" "5"
# (Answer: a)

# Exercise 2
x <- c(11, 22, 47, 47, 11, 47, 11)
factor(x, levels=c(11, 22, 47), ordered=TRUE)
```

```
## [1] 11 22 47 47 11 47 11
## Levels: 11 < 22 < 47
```

```r
## [1] 11 22 47 47 11 47 11
## Levels: 11 < 22 < 47
# (Answer: c)

# Exercise 3
z <- c("p", "a" , "g", "t", "b")
z[3] <- "b"
z
```

```
## [1] "p" "a" "b" "t" "b"
```

```r
## [1] "p" "a" "b" "t" "b"
# (Answer: c)

# Exercise 4
z <- factor(c("p", "q", "p", "r", "q"))
levels(z)[1] <- "w"
z
```

```
## [1] w q w r q
## Levels: w q r
```

```r
## [1] w q w r q
## Levels: w q r
# Exercise 5
s1 <- factor(sample(letters, size=5, replace=TRUE))
s2 <- factor(sample(letters, size=5, replace=TRUE))
factor(c(levels(s1)[s1], levels(s2)[s2]))
```

```
##  [1] b h v f t q x g k g
## Levels: b f g h k q t v x
```

```
## Levels: d f g h j s u v x
# Exercise 6
table(cut(iris$Sepal.Length, 5))
```

```
##
##  (4.3,5.02] (5.02,5.74] (5.74,6.46] (6.46,7.18]  (7.18,7.9]
##          32          41          42          24          11
```

```
# Exercise 7
table(iris$Sepal.Length < 5, factor(iris$Species))
```

```
##
##         setosa versicolor virginica
##   FALSE     30         49        49
##   TRUE      20          1         1
```

```
##
# Exercise 8
responses <- factor(c("Agree", "Agree", "Strongly Agree", "Disagree", "Agree"))
responses
```

```
## [1] Agree          Agree          Strongly Agree Disagree
## [5] Agree
## Levels: Agree Disagree Strongly Agree
```

```
## Levels: Agree Disagree Strongly Agree
factor(responses, levels=c("Strongly Agree", "Agree", "Disagree", "Strongly Disagree"))
```

```
## [1] Agree          Agree          Strongly Agree Disagree
## [5] Agree
## Levels: Strongly Agree Agree Disagree Strongly Disagree
```

```
# Exercise 9
x <- data.frame(q=c(2, 4, 6), p=c("a", "b", "c"))
x$p <- factor(x$p, levels=c("a", "b", "c"), labels=c("fertiliser1", "fertiliser2", "fertiliser3"))
x
```

```
##   q           p
## 1 2 fertiliser1
## 2 4 fertiliser2
## 3 6 fertiliser3
```

```
# Exercise 10
x <- factor(c("high", "low", "medium", "high", "high", "low", "medium"))
data.frame(levels = unique(x), value = as.numeric(unique(x)))
```

```
##   levels value
## 1   high     1
## 2    low     2
## 3 medium     3
```

Matrices

Exercises

Exercise 1

Create three vectors x,y,z with integers and each vector has 3 elements. Combine the three vectors to become a 3×3 matrix A where each column represents a vector. Change the row names to a,b,c. Think: How about each row represents a vector, can you modify your code to implement it?

Exercise 2

Please check your result from Exercise 1, using is.matrix(A). It should return TRUE, if your answer is correct. Otherwise, please correct your answer. Hint: Note that is.matrix() will return FALSE on a non-matrix type of input. Eg: a vector and so on.

Exercise 3

Create a vector with 12 integers. Convert the vector to a 4*3 matrix B using matrix(). Please change the column names to x, y, z and row names to a, b, c, d. The argument byrow in matrix() is set to be FALSE by default. Please change it to TRUE and print B to see the differences.

Exercise 4

Please obtain the transpose matrix of B named tB .

Exercise 5

Now tB is a 3×4 matrix. By the rule of matrix multiplication in algebra, can we perform tB*tB in R language? (Is a 3×4 matrix multiplied by a 3×4 allowed?) What result would we get?

Exercise 6

As we can see from Exercise 5, we were expecting that tB*tB would not be allowed because it disobeys the algebra rules. But it actually went through the computation in R. However, as we check the output result , we notice the multiplication with a single* operator is performing the componentwise multiplication. It is not the conventional matrix multiplication. How to perform the conventional matrix multiplication in R? Can you compute matrix A multiplies tB ?

Exercise 7

If we convert A to a data.frame type instead of a matrix , can we still compute a conventional matrix multiplication for matrix A multiplies matrix A ? Is there any way we could still perform the matrix multiplication for two data.frame type variables? (Assuming proper dimension)

Exercise 8 Extract a sub-matrix from B named subB . It should be a 3×3 matrix which includes the last three rows of matrix B and their corresponding columns.

Exercise 9 Compute 3*A , A+subB , A-subB . Can we compute A+B? Why?

Exercise 10

Generate a n * n matrix (square matrix) A1 with proper number of random numbers, then generate another n * m matrix A2. If we have A1*M=A2 (Here* represents the conventional multiplication), please solve for M. Hint: use the runif() and solve() functions. E.g., runif(9) should give you 9 random numbers.

Solutions

```
#1.
 x<-c(1,2,3)
 y<-c(4,5,6)
 z<-c(7,8,9)
 A<-cbind(x,y,z)
 rownames(A)<-c("a","b","c")
####if combined by rows A<-rbind(x,y,z)

#2.
is.matrix(A)
```

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

```
#if A is a data.frame then this should return false. So please note the
#different usages between data.frame and matrix.

#3.
b<-c(1:12)
B<-matrix(b, 4, 3, dimnames = list(c("a","b","c","d"),c("x", "y", "z")))

#4.
tB<-t(B)

#5.
tB*tB  #although expecting this would return error, it actually passes the computation
```

```
##    a   b   c   d
## x  1   4   9  16
## y 25  36  49  64
## z 81 100 121 144
```

```
#this is due to that * give component-wise multiplication, but not the real matrix #multiplication defi

#6. #use %*% for conventional matrix multiplication
# so matrix A (3x3) multiplies tB(3x4) should be

A%*%tB
```

```
##     a   b   c   d
## a  84  96 108 120
## b  99 114 129 144
## c 114 132 150 168
```

```
#7.
A%*%A #this is allowed
```

```
##    x  y   z
## a 30 66 102
## b 36 81 126
## c 42 96 150
```

```
#if we do
datA<-data.frame(A)
#datA%*%datA #this will return error

## Error in datA %*% datA: requires numeric/complex matrix/vector arguments
#although data.frame type looks the same as matrix
# we could do
as.matrix(datA)%*%as.matrix(datA)
```

```
##    x  y   z
## a 30 66 102
## b 36 81 126
## c 42 96 150
```

```
#8.
subB<-B[2:dim(B)[1],1:3]
subB
```

```
##   x y  z
## b 2 6 10
## c 3 7 11
## d 4 8 12
```

```
#9.
3*A
```

```
##   x  y  z
## a 3 12 21
## b 6 15 24
## c 9 18 27
```

```
A+subB
```

```
##   x  y  z
## a 3 10 17
## b 5 12 19
## c 7 14 21
```

```
A-subB
```

```
##    x  y  z
## a -1 -2 -3
## b -1 -2 -3
## c -1 -2 -3
```

```
#A+B #not allowed due to improper dimension
## Error in A + B: non-conformable arrays
#10.
A1<-matrix(runif(16),4,4)
A1
```

```
##           [,1]      [,2]       [,3]      [,4]
## [1,] 0.8154828 0.5365791 0.51875004 0.1175676
## [2,] 0.8128234 0.6462754 0.01446487 0.4971514
## [3,] 0.7046687 0.5552448 0.34951704 0.2554621
## [4,] 0.6984671 0.2552108 0.51724565 0.6313133
```

```
A2<-matrix(runif(8),4,2)
A2
```

```
##            [,1]      [,2]
## [1,] 0.72487118 0.5367465
## [2,] 0.40450982 0.1026095
## [3,] 0.06425064 0.3362053
## [4,] 0.50015831 0.8730614
```

```
M<-solve(A1,A2)
M
```

```
##           [,1]       [,2]
## [1,]  5.678450  0.6834276
## [2,] -5.014578 -0.9467410
## [3,] -1.912637  0.8729047
## [4,] -1.896007  0.2943420
```

Arrays

Exercises

Exercise 1

Create an array (3 dimensional) of 24 elements using the dim() function.

Exercise 2

Create an array (3 dimensional) of 24 elements using the array() function.

Exercise 3

Assign some dimnames of your choice to the array using the dimnames() function.

Exercise 4

Assign some dimnames of your choice to the array using the arguments of the array() function.

Exercise 5

Instead of column-major array, make a row-major array (transpose).

Exercise 6

For this exercise, and all that follow, download this file, and read it into R using the read.csv() function, e.g.: temp Copy the column named N into a new variable arr.

Exercise 7

Set dimensions of this variable and convert it into a 3 * 2 * 4 array. Add dimnames.

Exercise 8

Print the whole array on the screen.

Exercise 9

Print only elements of height 2, assuming the first dimension represents rows, the second columns and the third heigth.

Exercise 10

Print elements of height 1 and columns 3 and 1.

Exercise 11

Print element of height 2, column 4 and row 2.

Exercise 12

Repeat the exercises 9-11, but instead of using numbers to reference row, column and height, use dimnames.

Solution

Exercises

Exercise 1

Try to create matrices from the vectors below, by binding them column-wise. First, without using R, write down whether binding the vectors to a matrix is actually possible; then the resulting matrix and its mode (e.g., character, numeric etc.). Finally check your answer using R.

a. a <- 1:5 ; b <- 1:5
b. a <- 1:5 ; b <- c(‘1’, ‘2’, ‘3’, ‘4’, ‘5’)
c. a <- 1:5 ; b <- 1:4; c <- 1:3

Exercise 2 Repeat exercise 1, binding vectors row-wise instead of column-wise while avoiding any row names.

Exercise 3 Bind the following matrices column-wise. First, without using R, write down whether binding the matrices is actually possible; then the resulting matrix and its mode (e.g., character, numeric etc.). Finally check your answer using R.

a. a <- matrix(1:12, ncol=4); b <- matrix(21:35, ncol=5)
b. a <- matrix(1:12, ncol=4); b <- matrix(21:35, ncol=3)
c. a <- matrix(1:39, ncol=3); b <- matrix(LETTERS, ncol=2)

Exercise 4 Bind the matrix a <- matrix(1:1089, ncol=33) to itself, column-wise, 20 times (i.e., resulting in a new matrix with 21*33 columns). Hint: Avoid using cbind() to obtain an efficient solution. Various solutions are possible. If yours is different from those shown on the solutions page, please post yours on that page as comment, so we can all benefit.

Exercise 5 Try to create new data frames from the data frames below, by binding them column-wise. First, without using R, write down whether binding the data frames is actually possible; then the resulting data frame and the class of each column (e.g., integer, character, factor etc.). Finally check your answer using R. a. a <- data.frame(v1=1:5, v2=LETTERS[1:5]) ; b <- data.frame(var1=6:10, var2=LETTERS[6:10]) b. a <- data.frame(v1=1:6, v2=LETTERS[1:6]) ; b <- data.frame(var1=6:10, var2=LETTERS[6:10])

Exercise 6 Try to create new data frames from the data frames below, by binding them row-wise. First, without using R, write down whether binding the data frames is actually possible; then the resulting data frame and the class of each column (e.g., integer, character, factor etc.). Finally check your answer using R, and explain any unexpected output. a. a <- data.frame(v1=1:5, v2=LETTERS[1:5]) ; b <- data.frame(v1=6:10, v2=LETTERS[6:10]) b. a <- data.frame(v1=1:6, v2=LETTERS[1:6]) ; b <- data.frame(v2=6:10, v1=LETTERS[6:10])

Exercise 7 a. Use cbind() to add vector v3 <- 1:5 as a new variable to the data frame created in exercise 6b. b. Reorder the columns of this data frame, as follows: v1, v3, v2.

Exercise 8 Consider again the matrices of exercise 3b. Use both cbind() and rbind() to bind both matrices column-wise, adding NA for empty cells.

Exercise 9 Consider again the data frames of exercise 5b. Use both cbind() and rbind() to bind both matrices column-wise, adding NA for empty cells.

Solutions

```
###################
#                 #
#    Exercise 1   #
#                 #
####################

a <- 1:5; b <- 1:5
m <- cbind(a, b)
m
```

```
##      a b
## [1,] 1 1
## [2,] 2 2
## [3,] 3 3
## [4,] 4 4
## [5,] 5 5
```

```
is.matrix(m)
```

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

```
## [1] TRUE
mode(m)
```

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

```
## [1] "numeric"
a <- 1:5; b <- c('1', '2', '3', '4', '5')
m <- cbind(a, b)
m
```

```
##      a   b
## [1,] "1" "1"
## [2,] "2" "2"
## [3,] "3" "3"
## [4,] "4" "4"
## [5,] "5" "5"
```

```
is.matrix(m)
```

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

```
## [1] TRUE
mode(m)
```

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

```
## [1] "character"
a <- 1:5; b <- 1:4; c <- 1:3
m <- cbind(a, b)
```

```
## Warning in cbind(a, b): number of rows of result is not a multiple of
## vector length (arg 2)
```

```
## Warning in cbind(a, b): number of rows of result is not a multiple of
## vector length (arg 2)
m
```

```
##      a b
## [1,] 1 1
## [2,] 2 2
## [3,] 3 3
## [4,] 4 4
## [5,] 5 1
```

```
is.matrix(m)
```

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

```
mode(m)
```

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

```
####################
#                  #
#    Exercise 2    #
#                  #
####################

a <- 1:5; b <- 1:5
m <- rbind(a, b, deparse.level=0)
m
```

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

```
is.matrix(m)
```

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

```
mode(m)
```

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

```r
a <- 1:5; b <- c('1', '2', '3', '4', '5')
m <- rbind(a, b, deparse.level=0)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] "1"  "2"  "3"  "4"  "5"
## [2,] "1"  "2"  "3"  "4"  "5"
```

```r
is.matrix(m)
```

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

```r
mode(m)
```

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

```r
a <- 1:5; b <- 1:4; c <- 1:3
m <- rbind(a, b, deparse.level=0)
```

```
## Warning in rbind(a, b, deparse.level = 0): number of columns of result is
## not a multiple of vector length (arg 2)
```


```r
## Warning in rbind(a, b, deparse.level = 0): number of columns of result is
## not a multiple of vector length (arg 2)
```


```r
m
```

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

```r
is.matrix(m)
```

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


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


```r
mode(m)
```

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

```r
####################
#                  #
#    Exercise 3    #
#                  #
####################

a <- matrix(1:12, ncol=4) ; b <- matrix(21:35, ncol=5)
nrow(a) == nrow(b)  # to check if cbind is possible
```

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

```
## [1] TRUE
m <- cbind(a, b)
m
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    1    4    7   10   21   24   27   30   33
## [2,]    2    5    8   11   22   25   28   31   34
## [3,]    3    6    9   12   23   26   29   32   35
```

```
mode(m)
```

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

```
a <- matrix(1:12, ncol=4) ; b <- matrix(21:35, ncol=3)
nrow(a) == nrow(b)  # cbind not possible due to different number of rows
```

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

```
a <- matrix(1:39, ncol=3) ; b <- matrix(LETTERS, ncol=2)
nrow(a) == nrow(b)  # to check if cbind is possible
```

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

```
m <- cbind(a, b)
m
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,] "1"  "14" "27" "A"  "N"
##  [2,] "2"  "15" "28" "B"  "O"
##  [3,] "3"  "16" "29" "C"  "P"
##  [4,] "4"  "17" "30" "D"  "Q"
##  [5,] "5"  "18" "31" "E"  "R"
##  [6,] "6"  "19" "32" "F"  "S"
##  [7,] "7"  "20" "33" "G"  "T"
##  [8,] "8"  "21" "34" "H"  "U"
##  [9,] "9"  "22" "35" "I"  "V"
## [10,] "10" "23" "36" "J"  "W"
## [11,] "11" "24" "37" "K"  "X"
## [12,] "12" "25" "38" "L"  "Y"
## [13,] "13" "26" "39" "M"  "Z"
```

```
mode(m)
```

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

```
####################
#                  #
#    Exercise 4    #
#                  #
```

```r
####################

a <- matrix(1:1089, ncol=33)
a1 <- a[,rep(1:33, 21)]    # possible solution
a2 <- matrix(a, ncol=21*33, nrow=33)   # another solution
all.equal(a1, a2)
```

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

```r
## [1] TRUE
####################
#                  #
#    Exercise 5    #
#                  #
####################

a <- data.frame(v1=1:5, v2=LETTERS[1:5]); b <- data.frame(var1=6:10, var2=LETTERS[6:10])
nrow(a) == nrow(b)  # to check if cbind is possible
```

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

```r
## [1] TRUE
m <- cbind(a, b)
m
```

```
##   v1 v2 var1 var2
## 1  1  A    6    F
## 2  2  B    7    G
## 3  3  C    8    H
## 4  4  D    9    I
## 5  5  E   10    J
```

```r
str(m)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ v1  : int  1 2 3 4 5
##  $ v2  : Factor w/ 5 levels "A","B","C","D",..: 1 2 3 4 5
##  $ var1: int  6 7 8 9 10
##  $ var2: Factor w/ 5 levels "F","G","H","I",..: 1 2 3 4 5
```

```r
## 'data.frame':    5 obs. of  4 variables:
##  $ v1  : int  1 2 3 4 5
##  $ v2  : Factor w/ 5 levels "A","B","C","D",..: 1 2 3 4 5
##  $ var1: int  6 7 8 9 10
##  $ var2: Factor w/ 5 levels "F","G","H","I",..: 1 2 3 4 5
a <- data.frame(v1=1:6, v2=LETTERS[1:6]); b <- data.frame(var1=6:10, var2=LETTERS[6:10])
nrow(a) == nrow(b)  # cbind not possible due to different number of rows
```

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

```
####################
#                  #
#    Exercise 6    #
#                  #
####################

a <- data.frame(v1=1:5, v2=LETTERS[1:5]); b <- data.frame(v1=6:10, v2=LETTERS[6:10])
m <- rbind(a, b)
m
```

```
##    v1 v2
## 1   1  A
## 2   2  B
## 3   3  C
## 4   4  D
## 5   5  E
## 6   6  F
## 7   7  G
## 8   8  H
## 9   9  I
## 10 10  J
```

```
str(m)
```

```
## 'data.frame':    10 obs. of  2 variables:
##  $ v1: int  1 2 3 4 5 6 7 8 9 10
##  $ v2: Factor w/ 10 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 9 10
```

```
## 'data.frame':    10 obs. of  2 variables:
##  $ v1: int  1 2 3 4 5 6 7 8 9 10
##  $ v2: Factor w/ 10 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 9 10
a <- data.frame(v1=1:6, v2=LETTERS[1:6]); b <- data.frame(v2=6:10, v1=LETTERS[6:10])
m <- rbind(a, b)
```

```
## Warning in `[<-.factor`(`*tmp*`, ri, value = 6:10): invalid factor level,
## NA generated
```

```
## Warning in `[<-.factor`(`*tmp*`, ri, value = 6:10): invalid factor level,
## NA generated
m
```

```
##    v1   v2
## 1   1    A
## 2   2    B
## 3   3    C
## 4   4    D
## 5   5    E
## 6   6    F
## 7   F <NA>
## 8   G <NA>
## 9   H <NA>
## 10  I <NA>
## 11  J <NA>
```

```r
str(m)
```

```
## 'data.frame':    11 obs. of  2 variables:
##  $ v1: chr  "1" "2" "3" "4" ...
##  $ v2: Factor w/ 6 levels "A","B","C","D",..: 1 2 3 4 5 6 NA NA NA NA ...
```

```r
## 'data.frame':    11 obs. of  2 variables:
##  $ v1: chr  "1" "2" "3" "4" ...
##  $ v2: Factor w/ 6 levels "A","B","C","D",..: 1 2 3 4 5 6 NA NA NA NA ...
# If the NAs came as a surprise: Note that a$v2 if of class factor, while b$v2 is of class integer.

####################
#                  #
#    Exercise 7    #
#                  #
####################

v3 <- 11:21
m <- cbind(m, v3)
m <- m[c('v1', 'v3', 'v2')]


####################
#                  #
#    Exercise 8    #
#                  #
####################

a <- matrix(1:12, ncol=4) ; b <- matrix(21:35, ncol=3)
a <- rbind(a, matrix(NA, ncol=4, nrow=2))
m <- cbind(a, b)


####################
#                  #
#    Exercise 9    #
#                  #
####################

a <- data.frame(v1=1:6, v2=LETTERS[1:6]); b <- data.frame(var1=6:10, var2=LETTERS[6:10])
b <- rbind(b, c(NA, NA))
m <- cbind(a, b)
```

Matrix Operations

Exercises

Exercise 1

Consider A=matrix(c(2,0,1,3), ncol=2) and B=matrix(c(5,2,4,-1), ncol=2).

a) Find A + B
b) Find A - B

Exercise 2

Scalar multiplication. Find the solution for aA where a=3 and A is the same as in the previous question.

Exercise 3

Using the the diag function build a diagonal matrix of size 4 with the following values in the diagonal 4,1,2,3.

Exercise 4

Find the solution for Ab, where A is the same as in the previous question and b=c(7,4).

Exercise 5

Find the solution for AB, where B is the same as in question 1.

Exercise 6

Find the transpose matrix of A.

Exercise 7

Find the inverse matrix of A.

Exercise 8

Find the value of x on Ax=b.

Exercise 9

Using the function eigen find the eigenvalue for A.

Exercise 10

Find the eigenvalues and eigenvectors of A'A . Hint: Use crossprod to compute A'A .

Solutions

```r
####################
#                  #
#    Exercise 1    #
#                  #
####################

A <- matrix(c(2,0,1,3), ncol=2)
B <- matrix(c(5,2,4,-1),ncol=2)

A+B
```

```
##      [,1] [,2]
## [1,]    7    5
## [2,]    2    2
```

```r
A-B
```

```
##      [,1] [,2]
## [1,]   -3   -3
## [2,]   -2    4
```

```r
####################
#                  #
#    Exercise 2    #
#                  #
####################

a <- 3
a*A
```

```
##      [,1] [,2]
## [1,]    6    3
## [2,]    0    9
```

```r
####################
#                  #
#    Exercise 3    #
#                  #
####################

diag(4)*c(4,1,2,3)
```

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

```
####################
#                  #
#    Exercise 4    #
#                  #
####################
b <- c(7,4)
b%*%A
```

```
##      [,1] [,2]
## [1,]   14   19
```

```
####################
#                  #
#    Exercise 5    #
#                  #
####################
A%*%B
```

```
##      [,1] [,2]
## [1,]   12    7
## [2,]    6   -3
```

```
####################
#                  #
#    Exercise 6    #
#                  #
####################

t(A)
```

```
##      [,1] [,2]
## [1,]    2    0
## [2,]    1    3
```

```
####################
#                  #
#    Exercise 7    #
#                  #
####################


solve(A)
```

```
##      [,1]       [,2]
## [1,]  0.5 -0.1666667
## [2,]  0.0  0.3333333
```

```
####################
#                  #
#    Exercise 8    #
#                  #
```

```r
####################

solve(A,b)
```

```
## [1] 2.833333 1.333333
```

```r
####################
#                  #
#    Exercise 9    #
#                  #
####################

eigen(A)$values
```

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

```r
####################
#                  #
#    Exercise 10   #
#                  #
####################

eigen(crossprod(A))
```

```
## $values
## [1] 10.605551  3.394449
##
## $vectors
##           [,1]       [,2]
## [1,] 0.2897841 -0.9570920
## [2,] 0.9570920  0.2897841
```

Lists

Exercises

Exercise 1

If: p <- c(2,7,8), q <- c("A", "B", "C") and x <- list(p, q), then what is the value of x[2]? a. NULL b. "A" "B" "C" c. "7"

Exercise 2

If: w <- c(2, 7, 8) v <- c("A", "B", "C") x <- list(w, v), then which R statement will replace "A" in x with "K". a. x[[2]] <- "K" b. x[[2]][1] <- "K" c. x[[1]][2] <- "K"

Exercise 3

If a <- list ("x"=5, "y"=10, "z"=15), which R statement will give the sum of all elements in a? a. sum(a) b. sum(list(a)) c. sum(unlist(a))

Exercise 4

If Newlist <- list(a=1:10, b="Good morning", c="Hi"), write an R statement that will add 1 to each element of the first vector in Newlist.

Exercise 5

If b <- list(a=1:10, c="Hello", d="AA"), write an R expression that will give all elements, except the second, of the first vector of b.

Exercise 6

Let x <- list(a=5:10, c="Hello", d="AA"), write an R statement to add a new item z = "NewItem" to the list x.

Exercise 7

Consider y <- list("a", "b", "c"), write an R statement that will assign new names "one", "two" and "three" to the elements of y.

Exercise 8

If x <- list(y=1:10, t="Hello", f="TT", r=5:20), write an R statement that will give the length of vector r of x.

Exercise 9

Let string <- "Grand Opening", write an R statement to split this string into two and return the following output:

[[1]][1] "Grand" [[2]][1] "Opening"

Exercise 10

Let: y <- list("a", "b", "c") and q <- list("A", "B", "C", "a", "b", "c"). Write an R statement that will return all elements of q that are not in y, with the following result:

[[1]][1] "A" [[2]][1] "B" [[3]][1] "C"

Solutions

```r
# Exercise 1
p <- c(2,7,8)
q <- c("A", "B", "C")
x <- list(p, q)
x[2]
```

```
## [[1]]
## [1] "A" "B" "C"
```

```r
## [[1]]
## [1] "A" "B" "C"

# (Answer: b)

# Exercise 2
w <- c(2, 7, 8)
v <- c("A", "B", "C")
x <- list(w, v)
x[[2]][1] <- "K"
x
```

```
## [[1]]
## [1] 2 7 8
##
## [[2]]
## [1] "K" "B" "C"
```

```r
# (Answer: b)
# Exercise 3
a <- list ("x"=5, "y"=10, "z"=15)
sum(unlist(a))
```

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

```r
## [1] 30

# (Answer: c)

# Exercise 4
Newlist <- list(a=1:10, b="Good morning", c="Hi")
Newlist$a <- Newlist$a + 1
Newlist
```

```
## $a
##  [1]  2  3  4  5  6  7  8  9 10 11
##
## $b
## [1] "Good morning"
##
## $c
## [1] "Hi"
```

```r
# Exercise 5
b <- list(a=1:10, c="Hello", d="AA")
b$a[-2]
```

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

```r
# Exercise 6
x <- list(a=5:10, c="Hello", d="AA")
x$z <-"New Item"
x
```

```
## $a
## [1]  5  6  7  8  9 10
##
## $c
## [1] "Hello"
##
## $d
## [1] "AA"
##
## $z
## [1] "New Item"
```

```r
# Exercise 7
y <- list("a", "b", "c")
names(y) <- c("one", "two", "three")
y
```

```
## $one
## [1] "a"
##
## $two
## [1] "b"
##
## $three
## [1] "c"
```

```r
# Exercise 8
x <- list(y=1:10, t="Hello", f="TT", r=5:20)
length(x$r)
```

```
## [1] 16
```

```r
## [1] 16

# Exercise 9
string <- "Grand Opening"
a <- strsplit(string," ")
list(a[[1]][1], a[[1]][2])
```

```
## [[1]]
## [1] "Grand"
##
## [[2]]
## [1] "Opening"
```

```r
# Exercise 10
y <- list("a", "b", "c")
q <- list("A", "B", "C", "a", "b", "c")
setdiff(q, y)
```

```
## [[1]]
## [1] "A"
##
## [[2]]
## [1] "B"
##
## [[3]]
## [1] "C"
```

Data Frame

Exercises

Exercise 1

Create the following data frame, afterwards invert Sex for all individuals.

Exercise 2

Create this data frame (make sure you import the variable Working as character and not factor).

Add this data frame column-wise to the previous one. a) How many rows and columns does the new data frame have? b) What class of data is in each column?

Exercise 3

Check what class of data is the (built-in data set) state.center and convert it to data frame.

Exercise 4

Create a simple data frame from 3 vectors. Order the entire data frame by the first column.

Exercise 5

Create a data frame from a matrix of your choice, change the row names so every row says id_i (where i is the row number) and change the column names to variable_i (where i is the column number). I.e., for column 1 it will say variable_1, and for row 2 will say id_2 and so on.

Exercise 6

For this exercise, we'll use the (built-in) dataset VADeaths. a) Make sure the object is a data frame, if not change it to a data frame. b) Create a new variable, named Total, which is the sum of each row. c) Change the order of the columns so total is the first variable.

Exercise 7

For this exercise we'll use the (built-in) dataset state.x77. a) Make sure the object is a data frame, if not change it to a data frame. b) Find out how many states have an income of less than 4300. c) Find out which is the state with the highest income.

Exercise 8

With the dataset swiss, create a data frame of only the rows 1, 2, 3, 10, 11, 12 and 13, and only the variables Examination, Education and Infant.Mortality. a) The infant mortality of Sarine is wrong, it should be a NA, change it. b) Create a row that will be the total sum of the column, name it Total. c) Create a new variable that will be the proportion of Examination (Examination / Total)

Exercise 9

Create a data frame with the datasets state.abb, state.area, state.division, state.name, state.region. The row names should be the names of the states. a) Rename the column names so only the first 3 letters after the full stop appear (e.g. States.abb will be abb).

Exercise 10

Add the previous data frame column-wise to state.x77 a) Remove the variable div. b) Also remove the variables Life Exp, HS Grad, Frost, abb, and are. c) Add a variable to the data frame which should categorize the level of illiteracy: [0,1) is low, [1,2) is some, [2, inf) is high. d) Find out which state from the west, with low illiteracy, has the highest income, and what that income is.

Solutions

```
# Exercise 1
Name <- c("Alex", "Lilly", "Mark", "Oliver", "Martha", "Lucas", "Caroline")
Age <- c(25, 31, 23, 52, 76, 49, 26)
Height <- c(177, 163, 190, 179, 163, 183, 164)
Weight <- c(57, 69, 83, 75, 70, 83, 53)
Sex <- as.factor(c("F", "F", "M", "M", "F", "M", "F"))
df <- data.frame (row.names = Name, Age, Height, Weight, Sex)
levels(df$Sex) <- c("M", "F")
df
```

```
##          Age Height Weight Sex
## Alex      25    177     57   M
## Lilly     31    163     69   M
## Mark      23    190     83   F
## Oliver    52    179     75   F
## Martha    76    163     70   M
## Lucas     49    183     83   F
## Caroline  26    164     53   M
```

```
# Exercise 2

Name <- c("Alex", "Lilly", "Mark", "Oliver", "Martha", "Lucas", "Caroline")
Working <- c("Yes", "No", "No", "Yes", "Yes", "No", "Yes")

dfa <- data.frame(row.names = Name, Working)

# a)
dfa <- cbind (df,dfa)

dim(dfa)
```

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

```
## [1] 7 5
# or:
nrow(dfa)
```

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

```
## [1] 7
ncol(dfa)
```

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

```
## [1] 5
# b)
sapply(dfa, class)
```

```
##       Age    Height    Weight       Sex   Working
## "numeric" "numeric" "numeric"  "factor"  "factor"
```

```
##       Age    Height    Weight       Sex   Working
## "numeric" "numeric" "numeric"  "factor"  "factor"
str(dfa)   # alternative solution
```

```
## 'data.frame':    7 obs. of  5 variables:
##  $ Age    : num  25 31 23 52 76 49 26
##  $ Height : num  177 163 190 179 163 183 164
##  $ Weight : num  57 69 83 75 70 83 53
##  $ Sex    : Factor w/ 2 levels "M","F": 1 1 2 2 1 2 1
##  $ Working: Factor w/ 2 levels "No","Yes": 2 1 1 2 2 1 2
```

```
# Exercise 3
class (state.center)
```

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

```
## [1] "list"
df <- as.data.frame(state.center)


# Exercise 4

# Example vectors
v <- c(45:41, 30:33)
b <- LETTERS[rep(1:3, 3)]
n <- round(rnorm(9, 65, 5))

df <- data.frame(Age = v, Class = b, Grade = n)

df[with (df, order(Age)),]
```

```
##   Age Class Grade
## 6  30     C    69
## 7  31     A    66
## 8  32     B    69
## 9  33     C    67
## 5  41     B    70
## 4  42     A    73
## 3  43     C    67
## 2  44     B    61
## 1  45     A    60
```

```
df[order(df$Age), ]  # alternative solution
```

```
##   Age Class Grade
## 6  30     C    69
## 7  31     A    66
## 8  32     B    69
## 9  33     C    67
## 5  41     B    70
## 4  42     A    73
```

```
## 3  43     C    67
## 2  44     B    61
## 1  45     A    60
```

```
# Exercise 5
matr <- matrix(1:20, ncol = 5) # Example matrix
df <- as.data.frame(matr)
colnames(df) <- paste("variable_", 1:ncol(df))
rownames(df) <- paste("id_", 1:nrow(df))
df
```

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

```
# Exercise 6
#a)
class(VADeaths)
```

```
## [1] "matrix"
```

```
## [1] "matrix"
df <- as.data.frame(VADeaths)

#b)
df$Total <- df[, 1] + df[, 2] + df[, 3] + df[, 4]
df$Total <- rowSums(df[1:4])   # alternative solution

#c)
df <- df[, c(5, 1:4)]

# Exercise 7

#a)
class (state.x77)
```

```
## [1] "matrix"
```

```
## [1] "matrix"
df <- as.data.frame(state.x77)

#b)
nrow(subset(df, df$Income < 4300))
```

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

```
## [1] 20
#c)
row.names(df)[(which(max(df$Income) == df$Income))]
```

```
## [1] "Alaska"
```

```
## [1] "Alaska"
# Exercise 8
df <- swiss[c(1:3, 10:13), c("Examination", "Education", "Infant.Mortality")]

#a)
df[4,3] <- NA

#b)
df["Total",] <- c(sum(df$Examination), sum(df$Education), sum(df$Infant.Mortality, na.rm = TRUE))

#c)
df$proportion <- round(df$Examination / df["Total", "Examination"], 3)

# Exercise 9
df <- data.frame(state.abb, state.area, state.division, state.region, row.names = state.name)

#a)
names(df) <- substr(names(df), 7, 9)

# Exercise 10
dfa <- cbind(state.x77, df)

#a)
dfa$div <- NULL

#b)
dfa <- subset(dfa, ,-c(4, 6, 7, 9, 10))

# c)
dfa$illi <- ifelse(dfa$Illiteracy < 1,
                   "Low Illiteracy",
                   ifelse(dfa$Illiteracy >= 1 & dfa$Illiteracy < 2,
                          "Some Illiteracy",
                          "High Illiteracy"))
# Or:
dfa$illi <- cut(dfa$Illiteracy,
                c(0, 1, 2, 3),
                include.lowest = TRUE,
                right = FALSE,
                labels = c("Low Illiteracy", "Some Illiteracy", "High Illliteracy"))

# d)
sub <- subset(dfa, illi == "Low Illiteracy" & reg == "West")
max <- max(sub$Income)
stat <- row.names(sub)[which (sub$Income == max)]
cat("Highest income from the West is", max , "the state where it's from is", stat, "\n")
```

```
## Highest income from the West is 5149 the state where it's from is Nevada
```

```
## Highest income from the West is 5149 the state
```

Merging data frames

Exercises

Exercise 1

Create the dataframes to merge: buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3")) data <- data.frame(survey=c(1,1,1,2,2,2), location=c(1,2,3,2,3,1), efficiency=c(51,64,70,71,80,58)) The dataframes, buildings and data have a common key variable called, "location". Use the merge() function to merge the two dataframes by "location", into a new dataframe, "buildingStats".

Exercise 2

Give the dataframes different key variable names: buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3")) data <- data.frame(survey=c(1,1,1,2,2,2), LocationID=c(1,2,3,2,3,1), efficiency=c(51,64,70,71,80,58)) The dataframes, buildings and data now have corresponding variables called, location, and LocationID. Use the merge() function to merge the columns of the two dataframes by the corresponding variables.

Exercise 3

Inner Join: The R merge() function automatically joins the frames by common variable names. In that case, demonstrate how you would perform the merge in Exercise 1 without specifying the key variable.

Exercise 4

Outer Join: Merge the two dataframes from Exercise 1. Use the "all=" parameter in the merge() function to return all records from both tables. Also, merge with the key variable, "location".

Exercise 5

Left Join: Merge the two dataframes from Exercise 1, and return all rows from the left table. Specify the matching key from Exercise 1.

Exercise 6

Right Join: Merge the two dataframes from Exercise 1, and return all rows from the right table. Use the matching key from Exercise 1 to return matching rows from the left table.

Exercise 7

Cross Join: Merge the two dataframes from Exercise 1, into a "Cross Join" with each row of "buildings" matched to each row of "data". What new column names are created in "buildingStats"?

Exercise 8

Merging Dataframe rows: To join two data frames (datasets) vertically, use the rbind function. The two data frames must have the same variables, but they do not have to be in the same order. Merge the rows of the following two dataframes:

buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3")) buildings2 <- data.frame(location=c(5, 4, 6), name=c("building5", "building4", "building6")) Also, specify a new dataframe, "allBuidings".

Exercise 9

A new dataframe, buildings3, has variables not found in the previous dataframes

buildings3 <- data.frame(location=c(7, 8, 9), name=c("building7", "building8", "building9"), startEfficiency=c(75,87,91)) Create a new buildings3 without the extra variables.

Exercise 10

Instead of deleting the extra variables from buildings3. append the buildings, and buildings2 with the new variable in buildings3, (from Exercise 9). Set the new data in buildings and buildings2 , (from Exercise 8), to NA.

Solutions

```
####################
#                  #
#    Exercise 1    #
#                  #
####################

# Create the dataframes to merge:
buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3"))

data <- data.frame(survey=c(1,1,1,2,2,2), location=c(1,2,3,2,3,1),
efficiency=c(51,64,70,71,80,58))

# Solution
buildingStats <- merge(buildings, data, by="location")

####################
#                  #
#    Exercise 2    #
#                  #
####################

# Give the dataframes different key variable names:
buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3"))
data <- data.frame(survey=c(1,1,1,2,2,2), LocationID=c(1,2,3,2,3,1), efficiency=c(51,64,70,71,80,58))

# Solution
buildingStats <- merge(buildings, data, by.x="location",
                       by.y="LocationID")

####################
#                  #
#    Exercise 3    #
#                  #
####################

# Create the dataframes to merge:
buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3"))

data <- data.frame(survey=c(1,1,1,2,2,2), location=c(1,2,3,2,3,1),
efficiency=c(51,64,70,71,80,58))

# Solution
buildingStats <- merge(buildings, data)

####################
#                  #
#    Exercise 4    #
#                  #
####################

buildingStats <- merge(buildings, data, by="location", all=TRUE)
```

```
####################
#                  #
#    Exercise 5    #
#                  #
####################

 buildingStats <- merge(buildings, data, by="location", all.x=TRUE)

####################
#                  #
#    Exercise 6    #
#                  #
####################

 buildingStats <- merge(buildings, data, by="location", all.y=TRUE)

####################
#                  #
#    Exercise 7    #
#                  #
####################

buildingStats <- merge(buildings, data, by=NULL)

####################
#                  #
#    Exercise 8    #
#                  #
####################

# Required dataframes
buildings <- data.frame(location=c(1, 2, 3), name=c("building1", "building2", "building3"))
buildings2 <- data.frame(location=c(5, 4, 6), name=c("building5", "building4", "building6"))

# Solution
allBuidlings <- rbind(buildings, buildings2)

####################
#                  #
#    Exercise 9    #
#                  #
####################

# New dataframe
buildings3 <- data.frame(location=c(7, 8, 9),
name=c("building7", "building8", "building9"),
startEfficiency=c(75,87,91))

# Solution
buildings3 <- buildings3[,-3]

####################
#                  #
```

```
#    Exercise 10   #
#                  #
####################

buildings[ , "startEfficiency"] <- NA
buildings2[ , "startEfficiency"] <- NA
```

Accessing Dataframe Objects

Exercises

Exercise 1

attach() - Attach a set of R Objects to Search Path Required Dataframe:

```
buildingSurvey <- data.frame(name=c("bldg1", "bldg2", "bldg3",
"bldg4", "bldg5", "bldg6"),
survey=c(1,1,1,2,2,2),
location=c(1,2,3,2,3,1),
floors=c(5, 10, 10, 11, 8, 12),
efficiency=c(51,64,70,71,80,58))
```

Use the attach() function to make the variables in "buildingSurvey" independently searchable. Then, use "summary()" to create a summary of the "floors" variable.

Exercise 2

Using the "summary()" function, find the median "efficiency" value of "buildingSurvey", using objects in the R environment search path.

Exercise 3

Once attached, in order to change the dataframe variable, use the assignment operator "<<-". For example: variable1 <<- log(variable1) Use "<<-" to divide the "efficiency" category by 100.

Exercise 4

detach() - Detach Objects from the Search Path After detaching, modified attach() dataframes are restored to their pre-attach() values. and the R environment search path is restored. detach() is needed to prevent symantec errors in programming. Therefore, use the detach() function to restore the search paths of the dataframe, "buildingSurvey".

Exercise 5

The "transform()" function performs a transformation on a dataframe object. Use transform() to replace the "efficiency" column's values with the starting values divided by 100.

Exercise 6

First, re-attach the dataframe, "buildingSurvey". Then, use transform() to evaluate the log of the "efficiency" variable. Set the result to the dataframe, "efficiencyL". The column names of the dataframe "efficiencyL" should be "X_data", and "efficiencyLog".

Exercise 7

Next, use transform() to round the "efficiencyLog" variable of "efficiencyL" to one decimal place.

Exercise 8

The within() function creates a modified copy of a dataframe. For this exercise, use within() to append the "buildingSurvey" dataframe with a variable called, "efficiency10". The new variable contains "efficiency" multiplied by 10.

Exercise 9

Use the within() function to set efficiency[4] to "85". This will also create a copy of "buildingSurvey". Setting a new dataframe isn't required for this exercise.

Exercise 10

For the final exercise, restore the R environment search path.

Solutions

```
####################
#                  #
#    Exercise 1    #
#                  #
####################

attach(buildingSurvey)
summary(floors)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.000   8.500  10.000   9.333  10.750  12.000
```

```
####################
#                  #
#    Exercise 2    #
#                  #
####################

summary(efficiency)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   51.00   59.50   67.00   65.67   70.75   80.00
```

```
summary(efficiency)[3]
```

```
## Median
##     67
```

```
####################
#                  #
#    Exercise 3    #
#                  #
####################

efficiency <<- efficiency/100

####################
#                  #
#    Exercise 4    #
#                  #
####################

detach(buildingSurvey)

####################
#                  #
#    Exercise 5    #
#                  #
####################

transform(buildingSurvey, efficiency = efficiency/100)
```

```
##    name survey location floors efficiency
## 1 bldg1      1        1      5       0.51
## 2 bldg2      1        2     10       0.64
## 3 bldg3      1        3     10       0.70
## 4 bldg4      2        2     11       0.71
## 5 bldg5      2        3      8       0.80
## 6 bldg6      2        1     12       0.58
```

```
####################
#                  #
#    Exercise 6    #
#                  #
####################

attach(buildingSurvey)
efficiencyL <- transform(efficiency, efficiencyLog = log(efficiency))

####################
#                  #
#    Exercise 7    #
#                  #
####################

efficiencyL <- transform(efficiencyL,
efficiencyLog = round(efficiencyLog, 1))

####################
#                  #
#    Exercise 8    #
#                  #
####################

within(buildingSurvey, efficiency10 <- efficiency * 10)
```

```
##    name survey location floors efficiency efficiency10
## 1 bldg1      1        1      5         51          510
## 2 bldg2      1        2     10         64          640
## 3 bldg3      1        3     10         70          700
## 4 bldg4      2        2     11         71          710
## 5 bldg5      2        3      8         80          800
## 6 bldg6      2        1     12         58          580
```

```
####################
#                  #
#    Exercise 9    #
#                  #
####################

within(buildingSurvey, efficiency[4] <- 85)
```

```
##    name survey location floors efficiency
## 1 bldg1      1        1      5         51
## 2 bldg2      1        2     10         64
```

```
## 3 bldg3      1         3      10          70
## 4 bldg4      2         2      11          85
## 5 bldg5      2         3       8          80
## 6 bldg6      2         1      12          58
```

```r
####################
#                  #
#    Exercise 10   #
#                  #
####################

detach(buildingSurvey)
```

Applying Functions To Lists

Exercises

The lapply() function applies a function to individual values of a list, and is a faster alternative to writing loops. Structure of the lapply() function: lapply(LIST, FUNCTION, …) The list variable used for these exercises: list1 <- list(observationA = c(1:5, 7:3), observationB=matrix(1:6, nrow=2)) Answers to the exercises are available here.

Exercise 1

Using lapply(), find the length of list1's observations.

Exercise 2

Using lapply(), find the sums of list1's observations.

Exercise 3

Use lapply() to find the quantiles of list1.

Exercise 4

Find the classes of list1's sub-variables, with lapply().

Exercise 5

Required function: DerivativeFunction <- function(x) { log10(x) + 1 } Apply the "DerivativeFunction" to list1.

Exercise 6

Script the "DerivativeFunction" within lapply(). The dataset is list1. Exercise 7

Find the unique values in list1.

Exercise 8

Find the range of list1.

Exercise 9

Print list1 with the lapply() function.

Exercise 10

Convert the output of Exercise 9 to a vector, using the unlist(), and lapply(), functions.

Solutions

```
####################
#                  #
#    Exercise 1    #
#                  #
####################

list1 <- list(observationA = c(1:5, 7:3), observationB=matrix(1:6, nrow=2))

lapply(list1, length)
```

```
## $observationA
## [1] 10
##
## $observationB
## [1] 6
```

```
####################
#                  #
#    Exercise 2    #
#                  #
####################

lapply(list1, sum)
```

```
## $observationA
## [1] 40
##
## $observationB
## [1] 21
```

```
####################
#                  #
#    Exercise 3    #
#                  #
####################

lapply(list1, quantile)
```

```
## $observationA
##   0%  25%  50%  75% 100%
##    1    3    4    5    7
##
## $observationB
##   0%  25%  50%  75% 100%
## 1.00 2.25 3.50 4.75 6.00
```

```
####################
#                  #
#    Exercise 4    #
```

```
#                  #
####################

lapply(list1, class)
```

```
## $observationA
## [1] "integer"
##
## $observationB
## [1] "matrix"
```

```
####################
#                  #
#    Exercise 5    #
#                  #
####################

DerivativeFunction <- function(x) { log10(x) + 1 }

lapply(list1, DerivativeFunction)
```

```
## $observationA
##  [1] 1.000000 1.301030 1.477121 1.602060 1.698970 1.845098 1.778151
##  [8] 1.698970 1.602060 1.477121
##
## $observationB
##         [,1]     [,2]     [,3]
## [1,] 1.00000 1.477121 1.698970
## [2,] 1.30103 1.602060 1.778151
```

```
####################
#                  #
#    Exercise 6    #
#                  #
####################

lapply(list1, function(x) log10(x) + 1)
```

```
## $observationA
##  [1] 1.000000 1.301030 1.477121 1.602060 1.698970 1.845098 1.778151
##  [8] 1.698970 1.602060 1.477121
##
## $observationB
##         [,1]     [,2]     [,3]
## [1,] 1.00000 1.477121 1.698970
## [2,] 1.30103 1.602060 1.778151
```

```
####################
#                  #
#    Exercise 7    #
#                  #
```

```
####################

lapply(list1, unique)
```

```
## $observationA
## [1] 1 2 3 4 5 7 6
##
## $observationB
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
####################
#                  #
#    Exercise 8    #
#                  #
####################

lapply(list1, range)
```

```
## $observationA
## [1] 1 7
##
## $observationB
## [1] 1 6
```

```
####################
#                  #
#    Exercise 9    #
#                  #
####################

lapply(list1, print)
```

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

```
## $observationA
##  [1] 1 2 3 4 5 7 6 5 4 3
##
## $observationB
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
####################
#                  #
#    Exercise 10   #
#                  #
```

```r
####################

unlist(lapply(list1, print))
```

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

##  observationA1  observationA2  observationA3  observationA4  observationA5
##              1              2              3              4              5
##  observationA6  observationA7  observationA8  observationA9 observationA10
##              7              6              5              4              3
##  observationB1  observationB2  observationB3  observationB4  observationB5
##              1              2              3              4              5
##  observationB6
##              6
```

Reading delimited data

Exercises

Exercise 1

Read the file Table0.txt. a) Change the names of the columns to Name, Age, Height, Weight and Sex. b) Change the row names so that they are the same as Name, and remove the variable Name.

Exercise 2

Read the file Table1.txt, how many rows and columns does it have? a) Reread the file and make the variable Name be the row names. Make sure you read the variable as characters and not as factors.

Exercise 3

Read the file Table2.txt, watch out for the first line.

Exercise 4

Read the file Table3.txt, watch out for the first line and the missing values.

Exercise 5

Read the file Table4.txt, watch out for the missing values and the decimal separator.

Exercise 6

Read the file Table5.txt, watch out for the missing values and the decimal separator and the separator.

Exercise 7

Read the file states1.csv, the names of the states should be the row names.

Exercise 8

Read the file states2.csv, the names of the states should be the row names, watch out for the decimal separator and the separator.

Exercise 9

Read the file states2.csv, watch out for the same as the last exercise plus the missing values. Add to the previous dataset, column-wise.

Exercise 10 Read the file Table6.txt, check out the file first. Notice that the information is repeated, we only want the first non-repeated ones. Make sure to create only characters not factors this time around. Lastly, we don't want the comments.

Solutions

```
# Exercise 1
df<-read.table("D:/R/Exercises_and_Solutions/Table0.txt",header=T)
df
```

```
##       Alex X25 X177 X57 F
## 1    Lilly  31  163  69 F
## 2     Mark  23  190  83 M
## 3   Oliver  52  179  75 M
## 4   Martha  76  163  70 F
## 5    Lucas  49  183  83 M
## 6 Caroline  26  164  53 F
```

```
# a)
names(df) <- c('Name', 'Age', 'Height', 'Weight','Sex')

# b)
row.names(df)<-df$Name
df$Name <- NULL
df
```

```
##          Age Height Weight Sex
## Lilly     31    163     69   F
## Mark      23    190     83   M
## Oliver    52    179     75   M
## Martha    76    163     70   F
## Lucas     49    183     83   M
## Caroline  26    164     53   F
```

```
# Exercise 2
df<-read.table("D:/R/Exercises_and_Solutions/Table1.txt",header=T)
dim (df)
```

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

```
# a)
df <- read.table("D:/R/Exercises_and_Solutions/Table1.txt",header=T,
                 row.names = "Name",
                 stringsAsFactors = FALSE)
lapply(df, class)
```

```
## $Age
## [1] "integer"
##
## $Height
## [1] "integer"
##
## $Weight
## [1] "integer"
##
## $Sex
## [1] "character"
```

```r
# Exercise 3
df <- read.table("D:/R/Exercises_and_Solutions/Table2.txt",
                 header = T,
                 skip = 1,
                 quote ="/")
df
```

```
##       Name Age Height Weight Sex
## 1     Alex  25    177     57   F
## 2    Lilly  31    163     69   F
## 3     Mark  23    190     83   M
## 4   Oliver  52    179     75   M
## 5   Martha  76    163     70   F
## 6    Lucas  49    183     83   M
## 7 Caroline  26    164     53   F
```

```r
# Exercise 4
df <- read.table("D:/R/Exercises_and_Solutions/Table3.txt",
                 header = T,
                 skip = 1,
                 na.strings = c("NA", "*", "**", "--"))
df
```

```
##       Name Age Height Weight Sex
## 1     Alex  25    177     57   F
## 2    Lilly  31     NA     69   F
## 3     Mark  NA    190     83   M
## 4   Oliver  52    179     75   M
## 5   Martha  76     NA     70   F
## 6    Lucas  49    183     NA   M
## 7 Caroline  26    164     53   F
```

```r
# Exercise 5
df <- read.table("D:/R/Exercises_and_Solutions/Table4.txt",
                 header = T,
                 na.strings = c("NA", "*", "**", "--"),
                 dec = ",")
df
```

```
##       Name Age Height Weight Sex
## 1     Alex  25   1.77     57   F
## 2    Lilly  31     NA     69   F
## 3     Mark  NA   1.90     83   M
## 4   Oliver  52   1.79     75   M
## 5   Martha  76     NA     70   F
## 6    Lucas  49   1.83     NA   M
## 7 Caroline  26   1.64     53   F
```

```r
# Exercise 6
df <- read.table("D:/R/Exercises_and_Solutions/Table5.txt",
                 header = T,
                 na.strings = c(NA, "**", "--"),
```

```r
                 dec = ",",
                 sep = ";")
df
```

```
##       Name Age Height Weight Sex
## 1     Alex  25   1.77     57   F
## 2    Lilly  31     NA     69   F
## 3     Mark  NA   1.90     83   M
## 4   Oliver  52   1.79     75   M
## 5   Martha  76     NA     70   F
## 6    Lucas  49   1.83     NA   M
## 7 Caroline  26   1.64     53   F
```

```r
# Exercise 7
df <- read.csv("D:/R/Exercises_and_Solutions/states1.csv",
               row.names = 1)
df
```

```
##                Population Income Illiteracy Life.Exp Murder HS.Grad Frost
## Alabama              3615   3624        2.1    69.05   15.1    41.3    20
## Alaska                365   6315        1.5    69.31   11.3    66.7   152
## Arizona              2212   4530        1.8    70.55    7.8    58.1    15
## Arkansas             2110   3378        1.9    70.66   10.1    39.9    65
## California          21198   5114        1.1    71.71   10.3    62.6    20
## Colorado             2541   4884        0.7    72.06    6.8    63.9   166
## Connecticut          3100   5348        1.1    72.48    3.1    56.0   139
## Delaware              579   4809        0.9    70.06    6.2    54.6   103
## Florida              8277   4815        1.3    70.66   10.7    52.6    11
## Georgia              4931   4091        2.0    68.54   13.9    40.6    60
## Hawaii                868   4963        1.9    73.60    6.2    61.9     0
## Idaho                 813   4119        0.6    71.87    5.3    59.5   126
## Illinois            11197   5107        0.9    70.14   10.3    52.6   127
## Indiana              5313   4458        0.7    70.88    7.1    52.9   122
## Iowa                 2861   4628        0.5    72.56    2.3    59.0   140
## Kansas               2280   4669        0.6    72.58    4.5    59.9   114
## Kentucky             3387   3712        1.6    70.10   10.6    38.5    95
## Louisiana            3806   3545        2.8    68.76   13.2    42.2    12
## Maine                1058   3694        0.7    70.39    2.7    54.7   161
## Maryland             4122   5299        0.9    70.22    8.5    52.3   101
## Massachusetts        5814   4755        1.1    71.83    3.3    58.5   103
## Michigan             9111   4751        0.9    70.63   11.1    52.8   125
## Minnesota            3921   4675        0.6    72.96    2.3    57.6   160
## Mississippi          2341   3098        2.4    68.09   12.5    41.0    50
## Missouri             4767   4254        0.8    70.69    9.3    48.8   108
## Montana               746   4347        0.6    70.56    5.0    59.2   155
## Nebraska             1544   4508        0.6    72.60    2.9    59.3   139
## Nevada                590   5149        0.5    69.03   11.5    65.2   188
## New Hampshire         812   4281        0.7    71.23    3.3    57.6   174
## New Jersey           7333   5237        1.1    70.93    5.2    52.5   115
## New Mexico           1144   3601        2.2    70.32    9.7    55.2   120
## New York            18076   4903        1.4    70.55   10.9    52.7    82
## North Carolina       5441   3875        1.8    69.21   11.1    38.5    80
## North Dakota          637   5087        0.8    72.78    1.4    50.3   186
```

```
## Ohio                10735   4561         0.8    70.82    7.4    53.2  124
## Oklahoma             2715   3983         1.1    71.42    6.4    51.6   82
## Oregon               2284   4660         0.6    72.13    4.2    60.0   44
## Pennsylvania        11860   4449         1.0    70.43    6.1    50.2  126
## Rhode Island          931   4558         1.3    71.90    2.4    46.4  127
## South Carolina       2816   3635         2.3    67.96   11.6    37.8   65
## South Dakota          681   4167         0.5    72.08    1.7    53.3  172
## Tennessee            4173   3821         1.7    70.11   11.0    41.8   70
## Texas               12237   4188         2.2    70.90   12.2    47.4   35
## Utah                 1203   4022         0.6    72.90    4.5    67.3  137
## Vermont               472   3907         0.6    71.64    5.5    57.1  168
## Virginia             4981   4701         1.4    70.08    9.5    47.8   85
## Washington           3559   4864         0.6    71.72    4.3    63.5   32
## West Virginia        1799   3617         1.4    69.48    6.7    41.6  100
## Wisconsin            4589   4468         0.7    72.48    3.0    54.5  149
## Wyoming               376   4566         0.6    70.29    6.9    62.9  173
##                  Area
## Alabama         50708
## Alaska         566432
## Arizona        113417
## Arkansas        51945
## California     156361
## Colorado       103766
## Connecticut      4862
## Delaware         1982
## Florida         54090
## Georgia         58073
## Hawaii           6425
## Idaho           82677
## Illinois        55748
## Indiana         36097
## Iowa            55941
## Kansas          81787
## Kentucky        39650
## Louisiana       44930
## Maine           30920
## Maryland         9891
## Massachusetts    7826
## Michigan        56817
## Minnesota       79289
## Mississippi     47296
## Missouri        68995
## Montana        145587
## Nebraska        76483
## Nevada         109889
## New Hampshire    9027
## New Jersey       7521
## New Mexico     121412
## New York        47831
## North Carolina  48798
## North Dakota    69273
## Ohio            40975
## Oklahoma        68782
## Oregon          96184
```

```
## Pennsylvania    44966
## Rhode Island     1049
## South Carolina  30225
## South Dakota    75955
## Tennessee       41328
## Texas          262134
## Utah            82096
## Vermont          9267
## Virginia        39780
## Washington      66570
## West Virginia   24070
## Wisconsin       54464
## Wyoming         97203
```

```
# Exercise 8
dfa <- read.csv("http://r-exercises.com/wp-content/uploads/2015/12/states3.csv",
                row.names = 1,
                sep = ";",
                dec = ",")
dfa
```

```
##                    state.division state.area
## Alabama        East South Central      51609
## Alaska                       <NA>     589757
## Arizona                  Mountain     113909
## Arkansas       West South Central      53104
## California                   <NA>     158693
## Colorado                 Mountain     104247
## Connecticut           New England       5009
## Delaware           South Atlantic       2057
## Florida            South Atlantic      58560
## Georgia            South Atlantic         NA
## Hawaii                    Pacific       6450
## Idaho                    Mountain      83557
## Illinois       East North Central      56400
## Indiana        East North Central      36291
## Iowa           West North Central      56290
## Kansas         West North Central      82264
## Kentucky       East South Central      40395
## Louisiana      West South Central      48523
## Maine                 New England      33215
## Maryland           South Atlantic      10577
## Massachusetts         New England       8257
## Michigan       East North Central      58216
## Minnesota      West North Central      84068
## Mississippi    East South Central      47716
## Missouri       West North Central      69686
## Montana                  Mountain     147138
## Nebraska       West North Central      77227
## Nevada                   Mountain     110540
## New Hampshire         New England       9304
## New Jersey        Middle Atlantic         NA
## New Mexico               Mountain     121666
## New York          Middle Atlantic      49576
```

```
## North Carolina     South Atlantic      52586
## North Dakota   West North Central      70665
## Ohio           East North Central      41222
## Oklahoma       West South Central      69919
## Oregon                    Pacific      96981
## Pennsylvania      Middle Atlantic      45333
## Rhode Island           New England       1214
## South Carolina     South Atlantic      31055
## South Dakota   West North Central      77047
## Tennessee      East South Central      42244
## Texas          West South Central     267339
## Utah                         <NA>      84916
## Vermont                New England       9609
## Virginia           South Atlantic      40815
## Washington                Pacific      68192
## West Virginia      South Atlantic      24181
## Wisconsin      East North Central      56154
## Wyoming                  Mountain      97914
```

```
# Exercise 9
dfb <- read.csv("http://r-exercises.com/wp-content/uploads/2015/12/states2.csv",row.names = 1,sep = ";"
cbind(dfa, dfb)
```

```
##                    state.division state.area Population Income Illiteracy
## Alabama        East South Central      51609       3615   3624        2.1
## Alaska                       <NA>     589757        365   6315        1.5
## Arizona                  Mountain     113909       2212   4530        1.8
## Arkansas       West South Central      53104       2110   3378        1.9
## California                   <NA>     158693      21198   5114        1.1
## Colorado                 Mountain     104247       2541   4884        0.7
## Connecticut           New England       5009       3100   5348        1.1
## Delaware           South Atlantic       2057        579   4809        0.9
## Florida            South Atlantic      58560       8277   4815        1.3
## Georgia            South Atlantic         NA       4931   4091        2.0
## Hawaii                    Pacific       6450        868   4963        1.9
## Idaho                    Mountain      83557        813   4119        0.6
## Illinois       East North Central      56400      11197   5107        0.9
## Indiana        East North Central      36291       5313   4458        0.7
## Iowa           West North Central      56290       2861   4628        0.5
## Kansas         West North Central      82264       2280   4669        0.6
## Kentucky       East South Central      40395       3387   3712        1.6
## Louisiana      West South Central      48523       3806   3545        2.8
## Maine                 New England      33215       1058   3694        0.7
## Maryland           South Atlantic      10577       4122   5299        0.9
## Massachusetts         New England       8257       5814   4755        1.1
## Michigan       East North Central      58216       9111   4751        0.9
## Minnesota      West North Central      84068       3921   4675        0.6
## Mississippi    East South Central      47716       2341   3098        2.4
## Missouri       West North Central      69686       4767   4254        0.8
## Montana                  Mountain     147138        746   4347        0.6
## Nebraska       West North Central      77227       1544   4508        0.6
## Nevada                   Mountain     110540        590   5149        0.5
## New Hampshire         New England       9304        812   4281        0.7
## New Jersey        Middle Atlantic         NA       7333   5237        1.1
```

```
## New Mexico              Mountain     121666       1144   3601          2.2
## New York          Middle Atlantic      49576      18076   4903          1.4
## North Carolina     South Atlantic      52586       5441   3875          1.8
## North Dakota   West North Central      70665        637   5087          0.8
## Ohio           East North Central      41222      10735   4561          0.8
## Oklahoma       West South Central      69919       2715   3983          1.1
## Oregon                    Pacific      96981       2284   4660          0.6
## Pennsylvania      Middle Atlantic      45333      11860   4449          1.0
## Rhode Island          New England       1214        931   4558          1.3
## South Carolina     South Atlantic      31055       2816   3635          2.3
## South Dakota   West North Central      77047        681   4167          0.5
## Tennessee      East South Central      42244       4173   3821          1.7
## Texas          West South Central     267339      12237   4188          2.2
## Utah                         <NA>      84916       1203   4022          0.6
## Vermont               New England       9609        472   3907          0.6
## Virginia           South Atlantic      40815       4981   4701          1.4
## Washington                Pacific      68192       3559   4864          0.6
## West Virginia      South Atlantic      24181       1799   3617          1.4
## Wisconsin      East North Central      56154       4589   4468          0.7
## Wyoming                  Mountain      97914        376   4566          0.6
##                Life.Exp Murder HS.Grad Frost   Area
## Alabama           69.05   15.1    41.3    20  50708
## Alaska            69.31   11.3    66.7   152 566432
## Arizona           70.55    7.8    58.1    15 113417
## Arkansas          70.66   10.1    39.9    65  51945
## California        71.71   10.3    62.6    20 156361
## Colorado          72.06    6.8    63.9   166 103766
## Connecticut       72.48    3.1    56.0   139   4862
## Delaware          70.06    6.2    54.6   103   1982
## Florida           70.66   10.7    52.6    11  54090
## Georgia           68.54   13.9    40.6    60  58073
## Hawaii            73.60    6.2    61.9     0   6425
## Idaho             71.87    5.3    59.5   126  82677
## Illinois          70.14   10.3    52.6   127  55748
## Indiana           70.88    7.1    52.9   122  36097
## Iowa              72.56    2.3    59.0   140  55941
## Kansas            72.58    4.5    59.9   114  81787
## Kentucky          70.10   10.6    38.5    95  39650
## Louisiana         68.76   13.2    42.2    12  44930
## Maine             70.39    2.7    54.7   161  30920
## Maryland          70.22    8.5    52.3   101   9891
## Massachusetts     71.83    3.3    58.5   103   7826
## Michigan          70.63   11.1    52.8   125  56817
## Minnesota         72.96    2.3    57.6   160  79289
## Mississippi       68.09   12.5    41.0    50  47296
## Missouri          70.69    9.3    48.8   108  68995
## Montana           70.56    5.0    59.2   155 145587
## Nebraska          72.60    2.9    59.3   139  76483
## Nevada            69.03   11.5    65.2   188 109889
## New Hampshire     71.23    3.3    57.6   174   9027
## New Jersey        70.93    5.2    52.5   115   7521
## New Mexico        70.32    9.7    55.2   120 121412
## New York          70.55   10.9    52.7    82  47831
## North Carolina    69.21   11.1    38.5    80  48798
```

```
## North Dakota      72.78    1.4    50.3   186  69273
## Ohio              70.82    7.4    53.2   124  40975
## Oklahoma          71.42    6.4    51.6    82  68782
## Oregon            72.13    4.2    60.0    44  96184
## Pennsylvania      70.43    6.1    50.2   126  44966
## Rhode Island      71.90    2.4    46.4   127   1049
## South Carolina    67.96   11.6    37.8    65  30225
## South Dakota      72.08    1.7    53.3   172  75955
## Tennessee         70.11   11.0    41.8    70  41328
## Texas             70.90   12.2    47.4    35 262134
## Utah              72.90    4.5    67.3   137  82096
## Vermont           71.64    5.5    57.1   168   9267
## Virginia          70.08    9.5    47.8    85  39780
## Washington        71.72    4.3    63.5    32  66570
## West Virginia     69.48    6.7    41.6   100  24070
## Wisconsin         72.48    3.0    54.5   149  54464
## Wyoming           70.29    6.9    62.9   173  97203
```

```r
# Exercise 10
df <- read.table("http://r-exercises.com/wp-content/uploads/2015/12/Table6.txt",
                 skip = 1,
                 header = TRUE,
                 row.names = "Name",
                 nrow = 7,
                 flush = TRUE,
                 comment.char = "@",
                 stringsAsFactors = FALSE)
df
```

```
##          Age Height Weight Sex
## Alex      25    177     57   F
## Lilly     31    163     69   F
## Mark      23    190     83   M
## Oliver    52    179     75   M
## Martha    76    163     70   F
## Lucas     49    183     83   M
## Caroline  26    164     53   F
```

Scan

Exercises

Exercise 1

Read the file scan01.txt as a vector.

Exercise 2

Read the file scan02.txt: a) As a vector b) As a matrix with 10 rows.

Exercise 3

Read the file scan03.txt as a vector.

Exercise 4

Read the file scan04.txt as a matrix of 2 columns. Read only the 5 first rows. Finally, convert it to a data frame.

Exercise 5

Read the file scan05.txt as a list.

Exercise 6

Read the file scan05.txt as a list, read only the first 50 rows.

Exercise 7

Read the file scan.csv as a list.

Exercise 8

Read the file scan06.txt as a list. Read 10 rows of data.

Exercise 9

Read the file scan07.txt as a list.

Exercise 10

Read the file scan2.csv as a list.

Solutions

```r
# Exercise 1
v <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan01.txt")

# Exercise 2
# a)
vec <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan02.txt")
# b)
matrix <- matrix(scan("http://r-exercises.com/wp-content/uploads/2015/12/scan02.txt"),
                 nrow=10)

# Exercise 3
v <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan03.txt",
          what="character")

# Exercise 4
mat <- matrix(scan("http://r-exercises.com/wp-content/uploads/2015/12/scan04.txt",
                   sep="\t",
                   nlines=5),
              ncol=2)
df <- as.data.frame(mat)

# Exercise 5
list <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan05.txt",
             list(name="", x="character", y=1))

# Exercise 6
list <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan05.txt",
             list(name="", x="character", y=1), nlines = 50)

# Exercise 7
list <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan.csv",
             what=list("character", 1, "character"), sep=";", dec = ",")

# Exercise 8
list <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan06.txt",
             list(name="", x="character", y=1),
             nlines= 10,
             skip=2)

# Exercise 9
list <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan07.txt",
             list("character", 1, "character"),
             skip=1,
             quote="/",
             sep=";")

# Exercise 10
list <- scan("http://r-exercises.com/wp-content/uploads/2015/12/scan2.csv",
             list("character", 1, "character"),
             sep=",",
             quote="@",
```

```
                 skip=1)
```

Data Exploration with table()

Exercises

Exercise 1

Basic tabulation of categorical data:

This is the first dataset to explore:

```
Gender <- c("Female","Female","Male","Male")
Restaurant <- c("Yes","No","Yes","No")
Count <- c(220, 780, 400, 600)
DiningSurvey <- data.frame(Gender, Restaurant, Count)
DiningSurvey
```

```
##   Gender Restaurant Count
## 1 Female        Yes   220
## 2 Female         No   780
## 3   Male        Yes   400
## 4   Male         No   600
```

Using the above data, and the table() function, compare the Gender and Restaurant variables in the above dataset.

Exercise 2

The table() function modified with a logical vector:

Use the table() function, and logical vector of "Count > 650", to summarize the DiningSurvey data.

Exercise 3

The useNA & is.na arguments find missing values:

First append the "DiningSurvey" dataset with missing values:

```
DiningSurvey$Restaurant <- c("Yes", "No", "Yes", NA)
```

Apply the "useNA" argument to find missing Restaurant data.

Next, apply the "is.na()" argument to find missing Restaurant data by Gender.

Exercise 4

The "exclude =" parameter excludes columns of data:

Exclude one of the dataset's Genders with the "exclude" argument.

Exercise 5

The "margin.table()" function requires data in array form, and generates tables of marginal frequencies. The margin.table() function summarizes arrays within a given index:

First, generate array format data:

```
RentalUnits <- matrix(c(45,37,34,10,15,12,24,18,19),ncol=3,byrow=TRUE)
RentalUnits
```

```
##      [,1] [,2] [,3]
## [1,]   45   37   34
## [2,]   10   15   12
## [3,]   24   18   19
```

```
colnames(RentalUnits) <- c("Section1","Section2","Section3")
rownames(RentalUnits) <- c("Rented","Vacant","Reserved")
RentalUnits <- as.table(RentalUnits)
RentalUnits
```

```
##          Section1 Section2 Section3
## Rented         45       37       34
## Vacant         10       15       12
## Reserved       24       18       19
```

Using the above dataset, and the margin.table() function, find the amount of Occupancy summed over Sections.

Next, find the amount of Units summed by Section.

Exercise 6

The prop.table() function creates tables of proportions within the dataset:

With the "RentalUnits" data table, use the "prop.table()" function to create a basic table of proportions.

Next, find row percentages, and column percentages.

Exercise 7

The ftable() function generates multidimensional n-way tables, or "flat" contingency tables:

Use the ftable() function to summarize the dataset, "RentalUnits".

Exercise 8

The "summary()" function performs an independence test of factors:

Use "summary()" to perform a Chi-Square Test of Independence, of the "RentalUnits" variables.

Exercise 9

"as.data.frame()" summarizes frequencies of data arrays.

Use "as.data.frame()" to list frequencies within the "RentalUnits" array.

Exercise 10

The "addmargins()" function creates arbitrary margins on multivariate arrays:

Use "addmargins()" to append "RentalUnits" with sums.

Next, summarize columns with "RentalUnits".

Next, summarize rows with "RentalUnits".

Finally, combine "addmargins()" and "prop.table()" to summarize proportions within "RentalUnits". What is statistically inferred about sales of rental units by section?

Solutions

```
####################
# #
# Exercise 1 #
# #
####################

table(DiningSurvey$Gender)
```

```
##
## Female   Male
##      2      2
```

```
##
## Female Male
## 2 2

####################
# #
# Exercise 2 #
# #
####################

table(DiningSurvey$Count > 650)
```

```
##
## FALSE  TRUE
##     3     1
```

```
##
## FALSE TRUE
## 3 1

####################
# #
# Exercise 3 #
# #
####################

# Solution 1
table(DiningSurvey$Restaurant, useNA="always")
```

```
##
##   No  Yes <NA>
##    1    2    1
```

```
##
## No Yes
## 1 2 1

# Solution 2
table(DiningSurvey$Gender, is.na(DiningSurvey$Restaurant))
```

```
##
##           FALSE TRUE
##   Female     2    0
##   Male       1    1
```

```
##
## FALSE TRUE
## Female 2 0
## Male 1 1

####################
# #
# Exercise 4 #
# #
####################

table(DiningSurvey$Gender, exclude = "Male")
```

```
##
## Female
##      2
```

```
##
## Female
## 2

####################
# #
# Exercise 5 #
# #
####################

# Solution 1
margin.table(RentalUnits,1)
```

```
##   Rented   Vacant Reserved
##      116       37       61
```

```
## Rented Vacant Reserved
## 116 37 61

# Solution 2
margin.table(RentalUnits, 2)
```

```
## Section1 Section2 Section3
##       79       70       65
```

```
## Section1 Section2 Section3
## 79 70 65

####################
# #
```

```r
# Exercise 6 #
# #
####################

# Solution 1
prop.table(RentalUnits)
```

```
##             Section1   Section2   Section3
## Rented   0.21028037 0.17289720 0.15887850
## Vacant   0.04672897 0.07009346 0.05607477
## Reserved 0.11214953 0.08411215 0.08878505
```

```r
# Solution 2
prop.table(RentalUnits, 1)
```

```
##           Section1  Section2  Section3
## Rented   0.3879310 0.3189655 0.2931034
## Vacant   0.2702703 0.4054054 0.3243243
## Reserved 0.3934426 0.2950820 0.3114754
```

```r
prop.table(RentalUnits, 2)
```

```
##           Section1  Section2  Section3
## Rented   0.5696203 0.5285714 0.5230769
## Vacant   0.1265823 0.2142857 0.1846154
## Reserved 0.3037975 0.2571429 0.2923077
```

```r
####################
# #
# Exercise 7 #
# #
####################

ftable(RentalUnits)
```

```
##           Section1 Section2 Section3
##
## Rented          45       37       34
## Vacant          10       15       12
## Reserved        24       18       19
```

```r
####################
# #
# Exercise 8 #
# #
####################

summary(RentalUnits)
```

```
## Number of cases in table: 214
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 2.2034, df = 4, p-value = 0.6984
```

```r
####################
# #
# Exercise 9 #
# #
####################

as.data.frame(RentalUnits)
```

```
##       Var1     Var2 Freq
## 1   Rented Section1   45
## 2   Vacant Section1   10
## 3 Reserved Section1   24
## 4   Rented Section2   37
## 5   Vacant Section2   15
## 6 Reserved Section2   18
## 7   Rented Section3   34
## 8   Vacant Section3   12
## 9 Reserved Section3   19
```

```r
####################
# #
# Exercise 10 #
# #
####################

# Solution 1
addmargins(RentalUnits)
```

```
##          Section1 Section2 Section3 Sum
## Rented         45       37       34 116
## Vacant         10       15       12  37
## Reserved       24       18       19  61
## Sum            79       70       65 214
```

```r
# Solution 2
addmargins(RentalUnits,1)
```

```
##          Section1 Section2 Section3
## Rented         45       37       34
## Vacant         10       15       12
## Reserved       24       18       19
## Sum            79       70       65
```

```r
# Solution 3
addmargins(RentalUnits,2)
```

```
##          Section1 Section2 Section3 Sum
## Rented         45       37       34 116
## Vacant         10       15       12  37
## Reserved       24       18       19  61
```

```
## Section1 Section2 Section3 Sum
## Rented 45 37 34 116
## Vacant 10 15 12 37
## Reserved 24 18 19 61

# Solution 4
addmargins(prop.table(RentalUnits, 2))
```

```
##           Section1  Section2  Section3       Sum
## Rented   0.5696203 0.5285714 0.5230769 1.6212686
## Vacant   0.1265823 0.2142857 0.1846154 0.5254834
## Reserved 0.3037975 0.2571429 0.2923077 0.8532480
## Sum      1.0000000 1.0000000 1.0000000 3.0000000
```

Complex tables

Exercise 1

In order to demonstrate the ftable() function's capabilities, input the Titanic data from R:

```
data(Titanic)
```

For the first exercise, create a basic flat contingency table from the Titanic data, using the ftable() function.

Exercise 2

The row.vars argument specifies the table variables that will format as table rows. row.vars= is definable with variable numbers, or the variable names.

Therefore, use row.vars= to specify the variable, Class, as the row variables.

Exercise 3

Combine the row.vars and col.vars arguments to specify Class & Sex as the row variables, and Survived as the column variable.

row.vars and col.vars are definable with variable numbers, or the variable names.

Exercise 4

With the parameters from the ftable used in Exercise 3, reverse the order of the Class and Sex columns.

Exercise 5

Next, using the ftable() code from Exercise 4, specify Age as the column variable.

As you can see from this exercise, ftable() allows for the formatting of data for different areas of inquiry.

Exercise 6

Using the data.frame() function will coerce ftable columns into rows. To demonstrate this, place the ftable() from Exercise 5, within the data.frame() function.

Exercise 7

Function ftable.formula provides a formula interface, (a data = . argument), for creating flat contingency tables.

For example:

```
ftable(Survived ~ ., data = Titanic)
```

```
##                    Survived  No Yes
## Class Sex    Age
## 1st   Male   Child            0   5
##              Adult          118  57
##       Female Child            0   1
##              Adult            4 140
## 2nd   Male   Child            0  11
##              Adult          154  14
##       Female Child            0  13
##              Adult           13  80
## 3rd   Male   Child           35  13
##              Adult          387  75
##       Female Child           17  14
##              Adult           89  76
```

```
## Crew  Male   Child            0   0
##             Adult          670 192
##      Female Child            0   0
##             Adult            3  20
```

Use the formula interface for ftable() to display the quantities in the Titanic data for Male/Female passengers, by Class and Age.

Exercise 8

The ftable() function creates an object of class ftable. In order to demonstrate this, save the results of the ftable formula from Exercise 5 as an ftable variable called titanicStats.

Exercise 9

Using the write.table() function, write the ftable, titanicStats, to a file. Make sure your working directory is set to a folder where you can find the resulting file. Name the file, “table1”.

Exercise 10

read.ftable() reads in a flat-like contingency table from a file.

Using read.ftable(), read the file, “table1”, into an R language environment variable called data1.

Solutions

```r
####################
#                  #
#    Exercise 1    #
#                  #
####################
ftable(Titanic)
```

```
##                    Survived  No Yes
## Class Sex    Age
## 1st   Male   Child             0   5
##              Adult           118  57
##       Female Child             0   1
##              Adult             4 140
## 2nd   Male   Child             0  11
##              Adult           154  14
##       Female Child             0  13
##              Adult            13  80
## 3rd   Male   Child            35  13
##              Adult           387  75
##       Female Child            17  14
##              Adult            89  76
## Crew  Male   Child             0   0
##              Adult           670 192
##       Female Child             0   0
##              Adult             3  20
```

```r
####################
#                  #
#    Exercise 2    #
#                  #
####################

ftable(Titanic, row.vars = 1)
```

```
##       Sex       Male                Female
##       Age       Child     Adult     Child     Adult
##       Survived     No Yes    No Yes    No Yes    No Yes
## Class
## 1st                 0   5   118  57     0   1     4 140
## 2nd                 0  11   154  14     0  13    13  80
## 3rd                35  13   387  75    17  14    89  76
## Crew                0   0   670 192     0   0     3  20
```

```r
ftable(Titanic, row.vars = "Class")
```

```
##       Sex       Male                Female
##       Age       Child     Adult     Child     Adult
##       Survived     No Yes    No Yes    No Yes    No Yes
## Class
## 1st                 0   5   118  57     0   1     4 140
```

```
## 2nd                  0  11   154  14      0  13     13  80
## 3rd                 35  13   387  75     17  14     89  76
## Crew                 0   0   670 192      0   0      3  20
```

```
####################
#                  #
#    Exercise 3    #
#                  #
####################

ftable(Titanic, row.vars = 1:2, col.vars = 4)
```

```
##              Survived  No Yes
## Class Sex
## 1st   Male             118  62
##       Female             4 141
## 2nd   Male             154  25
##       Female            13  93
## 3rd   Male             422  88
##       Female           106  90
## Crew  Male             670 192
##       Female             3  20
```

```
ftable(Titanic, row.vars = c("Class", "Sex"),
col.vars = "Survived")
```

```
##              Survived  No Yes
## Class Sex
## 1st   Male             118  62
##       Female             4 141
## 2nd   Male             154  25
##       Female            13  93
## 3rd   Male             422  88
##       Female           106  90
## Crew  Male             670 192
##       Female             3  20
```

```
####################
#                  #
#    Exercise 4    #
#                  #
####################

ftable(Titanic, row.vars = 2:1, col.vars = 4)
```

```
##              Survived  No Yes
## Sex    Class
## Male   1st             118  62
##        2nd             154  25
##        3rd             422  88
##        Crew            670 192
## Female 1st               4 141
```

```
##        2nd              13  93
##        3rd             106  90
##        Crew              3  20
```

```r
ftable(Titanic, row.vars = c("Sex", "Class"),
col.vars = "Survived")
```

```
##              Survived  No Yes
## Sex    Class
## Male   1st            118  62
##        2nd            154  25
##        3rd            422  88
##        Crew           670 192
## Female 1st              4 141
##        2nd             13  93
##        3rd            106  90
##        Crew             3  20
```

```r
####################
#                  #
#    Exercise 5    #
#                  #
####################

ftable(Titanic, row.vars = 1:2, col.vars = 3)
```

```
##              Age Child Adult
## Class Sex
## 1st   Male           5   175
##       Female         1   144
## 2nd   Male          11   168
##       Female        13    93
## 3rd   Male          48   462
##       Female        31   165
## Crew  Male           0   862
##       Female         0    23
```

```r
ftable(Titanic, row.vars = c("Sex", "Class"),
col.vars = "Age")
```

```
##              Age Child Adult
## Sex    Class
## Male   1st           5   175
##        2nd          11   168
##        3rd          48   462
##        Crew          0   862
## Female 1st           1   144
##        2nd          13    93
##        3rd          31   165
##        Crew          0    23
```

```
####################
#                  #
#    Exercise 6    #
#                  #
####################
data.frame(ftable(Titanic, row.vars = 1:2, col.vars = 3))
```

```
##    Class    Sex   Age Freq
## 1    1st   Male Child    5
## 2    2nd   Male Child   11
## 3    3rd   Male Child   48
## 4   Crew   Male Child    0
## 5    1st Female Child    1
## 6    2nd Female Child   13
## 7    3rd Female Child   31
## 8   Crew Female Child    0
## 9    1st   Male Adult  175
## 10   2nd   Male Adult  168
## 11   3rd   Male Adult  462
## 12  Crew   Male Adult  862
## 13   1st Female Adult  144
## 14   2nd Female Adult   93
## 15   3rd Female Adult  165
## 16  Crew Female Adult   23
```

```
data.frame(ftable(Titanic, row.vars = c("Sex", "Class"),
col.vars = "Age"))
```

```
##       Sex Class   Age Freq
## 1    Male   1st Child    5
## 2  Female   1st Child    1
## 3    Male   2nd Child   11
## 4  Female   2nd Child   13
## 5    Male   3rd Child   48
## 6  Female   3rd Child   31
## 7    Male  Crew Child    0
## 8  Female  Crew Child    0
## 9    Male   1st Adult  175
## 10 Female   1st Adult  144
## 11   Male   2nd Adult  168
## 12 Female   2nd Adult   93
## 13   Male   3rd Adult  462
## 14 Female   3rd Adult  165
## 15   Male  Crew Adult  862
## 16 Female  Crew Adult   23
```

```
####################
#                  #
#    Exercise 7    #
#                  #
####################
ftable(Sex ~ Class + Age, data = Titanic)
```

```
##              Sex Male Female
## Class Age
## 1st   Child        5      1
##       Adult      175    144
## 2nd   Child       11     13
##       Adult      168     93
## 3rd   Child       48     31
##       Adult      462    165
## Crew  Child        0      0
##       Adult      862     23
```

```
####################
#                  #
#    Exercise 8    #
#                  #
####################
titanicStats <- ftable(Sex ~ Class + Age, data = Titanic)
####################
#                  #
#    Exercise 9    #
#                  #
####################
write.ftable(titanicStats, "table1")
####################
#                  #
#    Exercise 10   #
#                  #
####################
data1 <- read.ftable("table1")
```

Crosstabulation with Xtab

The xtabs() function creates contingency tables in frequency-weighted format. Use xtabs() when you want to numerically study the distribution of one categorical variable, or the relationship between two categorical variables. Categorical variables are also called "factor" variables in R.

Using a formula interface, xtabs() can create a contingency table, (also a "sparse matrix"), from cross-classifying factors, usually contained in a data frame.

Answers to the exercises are available here.

Exercise 1 xtabs() with One Categorical Variable

Input the following required Data Frame:

```
Data1 <- data.frame(Reference = c("KRXH", "KRPT", "FHRA", "CZKK", "CQTN", "PZXW", "SZRZ", "RMZE", "STNX
```

The xtabs() function can display the frequency, or count, of the levels of categorical variables. For the first exercise, use the xtabs() function to find the count of levels in the variable, "Status", within the above dataframe, "Data1".

Exercise 2

Two Categorical Variables - Discoving relationships within a dataset

Next, using the xtabs() function, apply two variables from "Data1", to create a table delineating the relationship between the "Reference" category, and the "Status" category.

Exercise 3

Three Categorical Variables - Creating a Multi-Dimensional Table

Apply three variables from "Data1" to create a Multi-Dimensional Cross-Tabulation of "Status", "Gender", and "Test".

Exercise 4

Creating Two Dimensional Tables from Multi-Dimensional Cross-Tabulations

Enclose the xtabs() formula from Exercise 3 within the "ftable()" function, to display a Multi-Dimensional Cross-Tabulation in two dimensions.

Exercise 5

Row Percentages

The R package "tigerstats" is required for the next two exercises.

```
if(!require(tigerstats)) {install.packages("tigerstats"); require(tigerstats)}
```

```
## Loading required package: tigerstats

## Loading required package: abd

## Loading required package: nlme

## Loading required package: lattice

## Loading required package: grid
```

```
## Loading required package: mosaic

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:nlme':
##
##     collapse

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Loading required package: mosaicData

## Loading required package: Matrix

##
## The 'mosaic' package masks several functions from core packages in order to add additional features.
## The original behavior of these functions should not be affected by this.

##
## Attaching package: 'mosaic'

## The following object is masked from 'package:Matrix':
##
##     mean

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cov, D, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```r
library(tigerstats)
```

1) Create an xtabs() formula that cross-tabulates "Status", and "Test".
2) Enclose the xtabs() formula in the tigerstats function, "rowPerc()" to display row percentages for "Status" by "Test".

Exercise 6

Column Percentages

1) Create an xtab() formula that cross-tabulates "Reference", and "Status".
2) Use "colPerc()" to display column percentages for "Reference" by "Status".

Exercise 7

Plotting Cross-Tabulations

Use the "plot()" function, and the "xtabs()" function to plot "Status" by "Gender".

Exercise 8

xtabs() - Explanatory and Response Variables

In order to examine whether the explanatory variable "Gender" affects the response variable " Status", create a two factor xtabs() formula with the Response variable as the first condition, and the Explanatory variable as the second condition.

Exercise 9

Using cbind() with xtabs()

Using the "cbind()" function within an xtabs() formula can define the last two columns of a flat table of your dataset. The variable after ~ (tilde) will display as the row data. For example, ftable(xtabs(cbind(variable1, variable2) ~ variable3, data=" ")).

For this exercise, create a flat table with columns for "Gender" and "Test". The row variables are "Reference".

Exercise 10

Testing Correlation with xtabs()

When processed through the "summary()" function, an xtabs() formula can test for independence of variables. Therefore, use summary() and xtabs() to test for a "Reference" affecting "Status" correlation.

##Solutions

```r
#Exercise 1
Data1 <- data.frame(Reference = c("KRXH", "KRPT", "FHRA", "CZKK", "CQTN", "PZXW", "SZRZ", "RMZE", "STNX
xtabs(~Status, data=Data1)
```

```
## Status
## Accepted Rejected
##        6        4
```

```r
#Exercise 2
xtabs(~Reference + Status, data=Data1)
```

```
##          Status
## Reference Accepted Rejected
##      CQTN        0        1
##      CZKK        1        0
##      FHRA        0        1
##      KRPT        1        0
##      KRXH        1        0
##      PZXW        1        0
##      RMZE        0        1
##      STNX        1        0
##      SZRZ        0        1
##      TMDW        1        0
```