Data Analytics and Visualizations in R - Exercises

Contents

1 Basic R Data Structures

1.1 Types

What are the scalar types in R?

```
# Scalars are just vectors of length one
```

1.2 Weirdness of R

What is the major difference between atomic vectors and lists? How can you turn a list into an atomic vector?

In order to check if an object is of a certain type you can use `is.[type](object)` ,e.g. `is.integer(object)`

Can you use the `is.vector()` function to understand whether a data structure is a vector? If not, what are the functions that you can use for this purpose?

```
# Atomic vectors can contain elements of the same type. The elements of a list
# can have different types.

# Yes, we can use the `is.vector()` function to understand is the structure is a
# vector. It will return TRUE or FALSE.

a <- list(c(1,2,3), "bla", TRUE)
print(a)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "bla"
##
## [[3]]
## [1] TRUE
```

```
is.vector(a)
```

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

```
# Best response: If possible you should use type specific coercions like
# `as.numeric()` or `as.character()`. But since lists are heterogenous, this
# might not work. A more general function is `unlist()`,  which returns the list
# into a vector of the most general type. Notice this difference:

a <- list(c(1,2,3), "bla", TRUE)
unlist(a)
```

```
## [1] "1"    "2"    "3"    "bla"  "TRUE"
```

```
as.character(a)
```

```
## [1] "c(1, 2, 3)" "bla"        "TRUE"
```

```
# Generally you can, but here comes the weird part: `is.vector()` will only return `TRUE` if the vector
# no attributes `names`. Therefore more specific functions like `is.atomic()` or `is.list()` functions
# be used to test if an object is actually atomic vectoror a list
```

1.3 Atomic Vector Concatenation

What happens when you try to generate an atomic vector with `c()`which is composed of different types of elements? What is the `mean()` of a logical vector?

```
# When we attempt to combine different types they will be coerced to the most
# flexible type. Types from least to most flexible are: logical, integer, double
#  and character.

str(c("a", 1)) # 1 corced to char
```

```
##  chr [1:2] "a" "1"
```

```
# As TRUE is encoded as 1 and FALSE as 0, the mean is the number of TRUEs
# devided by the vector length.
mean(c(TRUE, FALSE,FALSE))
```

```
## [1] 0.3333333
```

1.4 Vector Concatenation

Compare X and Y where X and Y are defined as follows. What is the difference?

```
x <- list(list(1,2), c(3,4))
y <- c(list(1,2), c(3,4))
```

```
# Answer
# X will combine seveal lists into one. Given a combination of atomic vectors
# and lists, y will coerce the vectors to lists before combining them.
str(x)
```

```
## List of 2
##  $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
##  $ : num [1:2] 3 4
```

```
str(y)
```

```
## List of 4
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : num 4
```

1.5 From Vectors to `data.frames`

First, create three named numeric vectors of size 10, 11 and 12 respectively in the following manner:

- One vector with the "colon" approach: *from:to*
- One vector with the `seq()` function: *seq(from, to)*

- And one vector with the `seq()` function and the by argument: *seq(from, to, by)*

For easier naming you can use the vector `letters` or `LETTERS` which contain the latin alphabet in small and capital, respectively. In order to select specific letters just use e.g. `letters[1:4]` to get the first four letters. Check their types. What is the outcome? Where do you think does the difference come from?

Then combine all three vectors in a list. Check the attributes of the vectors and the list. What is the difference and why?

Finally coerce the list to a `data.frame` with `as.data.frame()`. Why does it fail and how can we fix it? What happend to the names?

Hint: If list elements have no names, we can access them with the double brackets and an index, e.g. `my_list[[1]]`

```
# Answer

# A. create vectors
aa <- 1:10
names(aa) <- letters[aa]
aa
```

```
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
```

```
bb <- seq(1, 11)
names(bb) <- letters[bb]
bb
```

```
##  a  b  c  d  e  f  g  h  i  j  k
##  1  2  3  4  5  6  7  8  9 10 11
```

```
cc <- seq(1, 12, by=1)
names(cc) <- letters[cc]

typeof(aa)
```

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

```
typeof(bb)
```

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

```
typeof(cc)
```

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

```
# B. Combine all three vectors in a list

my_list <- list(aa, bb, cc)
my_list
```

```
## [[1]]
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##  a  b  c  d  e  f  g  h  i  j  k
##  1  2  3  4  5  6  7  8  9 10 11
##
## [[3]]
```

```
##  a  b  c  d  e  f  g  h  i  j  k  l
##  1  2  3  4  5  6  7  8  9 10 11 12
```

```
attributes(aa)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
attributes(bb)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k"
```

```
attributes(cc)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
attributes(my_list)
```

```
## NULL
```

```
# C. Coerce to data.frames

# my_df <- as.data.frame(my_list)# fails

# Fixing the length
my_list[[1]] <- c(my_list[[1]], NA, NA)
my_list[[2]] <- c(my_list[[2]], NA)

my_df <- as.data.frame(my_list)
names(my_df) <- LETTERS[1:3]
my_df
```

```
##    A  B  C
## a  1  1  1
## b  2  2  2
## c  3  3  3
## d  4  4  4
## e  5  5  5
## f  6  6  6
## g  7  7  7
## h  8  8  8
## i  9  9  9
## j 10 10 10
## k NA 11 11
##   NA NA 12
```

1.6 Attributes

Take again our `data.frame` from Question 5.

- Change the row names and the column names of the `data.frame` to capital letters (or small letters, if they are already capital.
- Change the `class` attribute to *list*. What happens?
- Change it now to any name you like. What happens now? What happens if you remove the class attribute

```r
# Answer
# A. One possible way through attributes

attributes(my_df)
```

```
## $names
## [1] "A" "B" "C"
##
## $row.names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" ""
##
## $class
## [1] "data.frame"
```

```r
attr(my_df, "names") <- letters[1:3]
attr(my_df, "row.names") <- LETTERS[1:12]
my_df
```

```
##    a  b  c
## A  1  1  1
## B  2  2  2
## C  3  3  3
## D  4  4  4
## E  5  5  5
## F  6  6  6
## G  7  7  7
## H  8  8  8
## I  9  9  9
## J 10 10 10
## K NA 11 11
## L NA NA 12
```

```r
# Or through accessor functions

names(my_df) <- LETTERS[1:3]
row.names(my_df) <- letters[1:12]
my_df
```

```
##    A  B  C
## a  1  1  1
## b  2  2  2
## c  3  3  3
## d  4  4  4
## e  5  5  5
## f  6  6  6
## g  7  7  7
## h  8  8  8
## i  9  9  9
## j 10 10 10
## k NA 11 11
## l NA NA 12
```

```r
# B.

attr(my_df, "class") <- "list"
my_df
```

```
## $A
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA
##
## $B
##  [1]  1  2  3  4  5  6  7  8  9 10 11 NA
##
## $C
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## attr(,"row.names")
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
## attr(,"class")
## [1] "list"
```

```
# Answer - the data.frame coerced to a list

# C
attr(my_df, "class") <- "Batman"
my_df
```

```
## $A
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA
##
## $B
##  [1]  1  2  3  4  5  6  7  8  9 10 11 NA
##
## $C
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## attr(,"row.names")
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
## attr(,"class")
## [1] "Batman"
```

```
# Answer - Nothing changes
```

1.7 Factors

- What is the difference between a Factor and a Vector?
- Create a vector of length 30 with three levels *Rita Repulsa, Lord Zedd* and *Rito Revolto* and equal length for each level
- What happens if you replace the second element of the vector with *Shredder*

```
# Answer
# A. A factor is a vector that can contain only predefined values, and is used
# to store categorical data. It is stored as an integer with a character string
# associated with each integer value

# B.

x <- gl(n=3, k=10, length=30, labels=c("Rita Repulsa", "Lord Zedd", "Rito Revolto"))
str(x)
```

```
##  Factor w/ 3 levels "Rita Repulsa",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
levels(x)
```

```
## [1] "Rita Repulsa" "Lord Zedd"    "Rito Revolto"
```

```r
attributes(x)
```

```
## $levels
## [1] "Rita Repulsa" "Lord Zedd"    "Rito Revolto"
##
## $class
## [1] "factor"
```

```r
# C
x[2] <- "Shredder"
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "Shredder"): invalid factor
## level, NA generated
```

```r
# It doesn't work. We get the error 'NA generated'
```

1.8 More fun with factors

```r
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))
f2 <- rev(factor(letters))
f3 <- factor(letters, levels = rev(letters))
```

The function `rev` reverses the order of an orderable object. What is the difference between f1, f2 and f3? Why?

```r
# Answer
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))

# f1 goes from a to z and when we apply the levels(f1), z will become 1 and a=26

f2 <- rev(factor(letters))

# f2 goes from z to a. but the levels are not changed.

f3 <- factor(letters, levels = rev(letters))

# f3 goes from a - z, but the underlying encoding goes from z = 1 to a = 26.
# We create the vector with the letters a to z BUT the mapped integer structure
# 26 to 1. Hence the levels but not the vector are reversed

f3
```

```
##  [1] a b c d e f g h i j k l m n o p q r s t u v w x y z
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

```r
# Reversing f3 will give f1

rev(f3)
```

```
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

1.9 Creating data.frames

Create a data.frame with 26 rows like this: Only the first and the last six rows are shown. Hint: Instead of the workaround with list you can also use simply `data.frame(column_name = column_vector, ...)`

```
aa <- seq(1:26)
bb <- seq(from=4, to=4*26, by=4)
cc <- rep(seq(1, 26, 2), each=2)
df <- data.frame(V1 = aa, V2 = bb, V3 = letters[cc])
head(df)
```

```
##   V1 V2 V3
## 1  1  4  a
## 2  2  8  a
## 3  3 12  c
## 4  4 16  c
## 5  5 20  e
## 6  6 24  e
```

```
tail(df)
```

```
##    V1  V2 V3
## 21 21  84  u
## 22 22  88  u
## 23 23  92  w
## 24 24  96  w
## 25 25 100  y
## 26 26 104  y
```

1.10 Combining `data.frames`

Now take the previous data.frame from Question 10 and reproduce the following `data.frame`. Only the first and the last six rows are shown **Hint:** In order to combine to data.frames by column you can use `cbind(df1, df2, ...)`

help(cbind)

```
df[,1] <- NULL
dd <- rev(rep(seq(1, 26, 2), each = 2))
ee <- seq(0, 1.6, length.out = 26)
df2 <- data.frame(V4 = dd, V5 = ee)
binded_df <- cbind(df2, df)
head(binded_df)
```

```
##   V4    V5 V2 V3
## 1 25 0.000  4  a
## 2 25 0.064  8  a
## 3 23 0.128 12  c
## 4 23 0.192 16  c
## 5 21 0.256 20  e
## 6 21 0.320 24  e
```

```
tail(binded_df)
```

```
##    V4    V5  V2 V3
## 21  5 1.280  84  u
## 22  5 1.344  88  u
```

```
## 23  3 1.408  92  w
## 24  3 1.472  96  w
## 25  1 1.536 100  y
## 26  1 1.600 104  y
```

1.11 Computation on data.frames

Create the data.frame *df* with `df <- as.data.frame(matrix(runif(9e6), 3e3, 3e3))` This will create a data.frame with 3000 columns and rows and a total of 9mil values.

Now compute the sum of any row, then compute the sum of any column. Measure the time for both operations. Why are the times different, although the size is the same?

- **Hint1:** The time is measured with the function `system.time(my_function_call())`, e.g: `system.time(mean(my_vector))`
- **Hint2:** The sum can be computed with the sum function `sum(my_vector)`
- **Hint2:** Columns and rows are selected by single brackets. Rows: `df[row_number,]`, Columns: `df[,col_number]`

```
# Answer

df <- as.data.frame(matrix(runif(9e6), 3e3, 3e3))

# rows
system.time(res <- sum(df[1,]))
```

```
##    user  system elapsed
##   0.023   0.000   0.023
```

```
res
```

```
## [1] 1511.687
```

```
# columngs
system.time(res2 <- sum(df[,1]))
```

```
##    user  system elapsed
##       0       0       0
```

```
res2
```

```
## [1] 1502.534
```

```
# Look at the structure of the objects over which we are computing the sum
# Column
str(df[1,])
```

```
## 'data.frame':    1 obs. of  3000 variables:
##  $ V1   : num 0.537
##  $ V2   : num 0.295
##  $ V3   : num 0.0878
##  $ V4   : num 0.887
##  $ V5   : num 0.916
##  $ V6   : num 0.969
##  $ V7   : num 0.961
##  $ V8   : num 0.657
##  $ V9   : num 0.929
##  $ V10  : num 0.429
##  $ V11  : num 0.209
```

```
##  $ V12  : num 0.4
##  $ V13  : num 0.192
##  $ V14  : num 0.649
##  $ V15  : num 0.652
##  $ V16  : num 0.0282
##  $ V17  : num 0.912
##  $ V18  : num 0.0502
##  $ V19  : num 0.762
##  $ V20  : num 0.0821
##  $ V21  : num 0.675
##  $ V22  : num 0.33
##  $ V23  : num 0.334
##  $ V24  : num 0.295
##  $ V25  : num 0.902
##  $ V26  : num 0.545
##  $ V27  : num 0.8
##  $ V28  : num 0.101
##  $ V29  : num 0.767
##  $ V30  : num 0.0634
##  $ V31  : num 0.747
##  $ V32  : num 0.0142
##  $ V33  : num 0.593
##  $ V34  : num 0.635
##  $ V35  : num 0.883
##  $ V36  : num 0.872
##  $ V37  : num 0.214
##  $ V38  : num 0.968
##  $ V39  : num 0.42
##  $ V40  : num 0.0649
##  $ V41  : num 0.465
##  $ V42  : num 0.229
##  $ V43  : num 0.534
##  $ V44  : num 0.806
##  $ V45  : num 0.68
##  $ V46  : num 0.102
##  $ V47  : num 0.919
##  $ V48  : num 0.192
##  $ V49  : num 0.503
##  $ V50  : num 0.19
##  $ V51  : num 0.737
##  $ V52  : num 0.272
##  $ V53  : num 0.0513
##  $ V54  : num 0.387
##  $ V55  : num 0.347
##  $ V56  : num 0.266
##  $ V57  : num 0.87
##  $ V58  : num 0.736
##  $ V59  : num 0.833
##  $ V60  : num 0.657
##  $ V61  : num 0.602
##  $ V62  : num 0.545
##  $ V63  : num 0.707
##  $ V64  : num 0.296
##  $ V65  : num 0.881
```

```
##  $ V66  : num 0.239
##  $ V67  : num 0.921
##  $ V68  : num 0.287
##  $ V69  : num 0.182
##  $ V70  : num 0.818
##  $ V71  : num 0.873
##  $ V72  : num 0.722
##  $ V73  : num 0.321
##  $ V74  : num 1
##  $ V75  : num 0.629
##  $ V76  : num 0.474
##  $ V77  : num 0.799
##  $ V78  : num 0.525
##  $ V79  : num 0.763
##  $ V80  : num 0.886
##  $ V81  : num 0.899
##  $ V82  : num 0.724
##  $ V83  : num 0.93
##  $ V84  : num 0.646
##  $ V85  : num 0.204
##  $ V86  : num 0.635
##  $ V87  : num 0.195
##  $ V88  : num 0.741
##  $ V89  : num 0.164
##  $ V90  : num 0.982
##  $ V91  : num 0.157
##  $ V92  : num 0.877
##  $ V93  : num 0.75
##  $ V94  : num 0.384
##  $ V95  : num 0.145
##  $ V96  : num 0.41
##  $ V97  : num 0.098
##  $ V98  : num 0.436
##  $ V99  : num 0.434
##   [list output truncated]
```

```
# Row
str(df[,1])
```

```
##  num [1:3000] 0.537 0.625 0.981 0.871 0.847 ...
```

```
# As we can see the extracted column is a numeric vector. But the extracted
# row is a list. Under the hood the sum function is iterating in C/Fortran
# over the specific structure. Iterating over a native array of doubles is
# faster, than iterating over a structure, where at each position, the value
# has to be retrieved from an object possibly strored somewhere further away
# in memory.
```

1.12 Missing Values

- If NA is just a placeholder for a missing value of the same type and Infinity is of type double, why is Infinity plus NA not Infinity?

Hint:

```r
paste(paste(rep((Inf - Inf), 20), collapse = ""), "Batman!")
```

```
## [1] "NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!"
```

```r
# Answer:
# Because infinity is not a well defined number, but a concept with
# the type 'double' in R. Adding or subtracting any number from infinity
# will give infinity. However NA is placeholder for any value of the same
# type and therefore also for infinity. As infinity plus/minus infinity
# is not defined, adding NA to infinity can theoretically lead to
# Nan (not a number). Therefore Inf + NA will produce NA.

Inf + NA
```

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

```r
Inf + 1
```

```
## [1] Inf
```

```r
Inf - Inf
```

```
## [1] NaN
```

---

2 Advanced R Data Structures and Mathematical Operations

2.1 Basic data structures

How would you create a 3 by 4 matrix that contains the numbers 1 to 12 and then convert it into a data frame?

```r
# Answer

x <- matrix(1:12, 3,4)
x <- as.data.frame(x)
x
```

```
##   V1 V2 V3 V4
## 1  1  4  7 10
## 2  2  5  8 11
## 3  3  6  9 12
```

2.2

Please use the data frame you created in the first question for the next 5 questions. How would you select the second row elements at second and fourth column ?

```r
x <- data.frame(matrix(1:12, 3, 4))
x[2, c(2,4)]
```

```
##   X2 X4
## 2  5 11
```

2.3

How would you assign zero to the elements at row 2 which are greater than 4?

```
x <- data.frame(matrix(1:12, 3, 4))
x[2, x[2,]>4] <- 0
x
```

```
##   X1 X2 X3 X4
## 1  1  4  7 10
## 2  2  0  0  0
## 3  3  6  9 12
```

2.4

How do you set the rownames to "row1", "row2", "row3" and column names to "col1", "col2", "col3" and "col4" ? (hint:use function "paste0")

```
x <- data.frame(matrix(1:12, 3, 4))
rownames(x) <- paste0("row", 1:3)
colnames(x) <- paste0("col", 1:4)
x
```

```
##      col1 col2 col3 col4
## row1    1    4    7   10
## row2    2    5    8   11
## row3    3    6    9   12
```

2.5

How do you assign 0 to all elements in columns "col3" and "col4" by using paste0 function ?

```
x <- data.frame(matrix(1:12, 3, 4))
colnames(x) <- paste0("col", 1:4)
x[, paste0(0, 3:4)] <- 0
x
```

```
##   col1 col2 col3 col4 03 04
## 1    1    4    7   10  0  0
## 2    2    5    8   11  0  0
## 3    3    6    9   12  0  0
```

2.6 `lapply()`

How do you get the numbers whose mod 2 is 0

- by using lapply() function
- by subsetting the data frame directly?

```
x <- data.frame(matrix(1:12), 3, 4)

lapply(x, function(a) a[(a %% 2) == 0])
```

```
## $matrix.1.12.
## [1]  2  4  6  8 10 12
```

```
##
## $X3
## numeric(0)
##
## $X4
##  [1] 4 4 4 4 4 4 4 4 4 4 4 4 4
```

```
x[x %% 2 == 0]
```

```
##  [1]  2  4  6  8 10 12  4  4  4  4  4  4  4  4  4  4  4  4
```

2.7

Considering `x <- c("a"=1, "b"=2, "c"=3, "d"=4, "e"=5)`, show how to select the third and fifth elements of x by using positive integers, negative integers, a logical vector, and a character vector.

```
x <- c("a"=1, "b"=2, "c"=3, "d"=4, "e"=5)
x[c(3,5)]
```

```
## c e
## 3 5
```

```
x[-c(1,2,4)]
```

```
## c e
## 3 5
```

```
x[c(F,F,T,F,T)]
```

```
## c e
## 3 5
```

```
x[c("c", "e")]
```

```
## c e
## 3 5
```

2.8

Why are `vals[c(2, 5)]` and `vals[2, 5]` different where `vals <- outer(1:5, 1:5, FUN = "/")`? How would you select fifth and nineth elements of vals by the use of a matrix?

```
vals <- outer(1:5, 1:5, FUN = "/")

# Because when you subset matrix with a vector, the 2d matrix behaves
# like a vector and vals[c(2, 5)] returns the elements at indices 2
# and 5 in column-major order. vals[2, 5] returns the element at row 2, column 5.

select <- matrix(ncol=2, byrow=TRUE, c(5,1,4,2))

vals[select]
```

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

2.9

Consider df <- data.frame(a=paste("Point_", 1:20), b=rep(1:4, each = 2, len = 20), c= seq(1,40,length.out = 20), stringsAsFactors = F). Assign "Point_undefined" to column a of all rows of df where column b > 1 and column c > 21 ? What is the reason of the different result that you get if you do the same operation with df being created with option stringsAsFactors = T ?

2.10

Assume x <- matrix(1:20, ncol=2). What is the difference between x[1, , drop = T] and x[1, , drop = F]? Now let y <- as.data.frame(x). What is the difference between y[,1] , y[[1]] and y[1]

2.11

What is the difference between x["b"] <- list(NULL) and x["b"] <- NULL where x <- list(a = c(1:5), b = c(12:15))?

2.12

Assume you have a lookup table as lookup <- c(a = "sun", b = "rain", c="wind", u = NA). How would you generate the weekly weather predictions c("sun","sun","rain", NA, "rain", "rain", "wind") out of this lookup table?

2.13

Now assume the weather in winter lookup table is a data frame as below and we have the predictions for the next week as stored in weeklyCast. How would you create "weeklyTable" by the use of rownames function? How would you create it by the use of match function? How would you order the rows of lookup table by desc column?

2.14

Consider the bigDF data frame which has 1500 columns and rows. How would you select the even numbered columns named such as "Column_2", "Column_4", etc.? How would you select all the columns other than column 76? How would you assign 1 to 500 randomly selected diagonal indices? How can you retrieve the row and column indices of the elements which has been assigned 1? How would you select rows where columns Column_1 or Column_2 are 1 by using the subset() function?

2.15

Assume x <- 1:20 %% 2 == 0 and y <- 1:20 %% 5 == 0 . What are the indices of the elements that are True for both x and y? What are the indices of the elements that are True for either x or y, or both?

3 Data Import

3.1 Flat File - Q1

A csv file has numbers as column names in the first row, i.e. IDs to randomize persons. Which parameter of read.table() needs to be adjusted to read the column names as they are in the csv?

```
tmp_tidy_table <- "1_colname, 2_colname, 3_colname
3,4,5
a,b,c"
tmp_tidy_table
```

```
## [1] "1_colname, 2_colname, 3_colname\n3,4,5\na,b,c"
```

```
read.csv(text=tmp_tidy_table)
```

```
##   X1_colname X2_colname X3_colname
## 1          3          4          5
## 2          a          b          c
```

```
# Parameter `check.names`: a logical, tests for synatactically valid varaible
# names

tidy_text_df <- read.csv(text=tmp_tidy_table, check.names = FALSE)
tidy_text_df
```

```
##   1_colname 2_colname 3_colname
## 1         3         4         5
## 2         a         b         c
```

3.2 Flat File - Q2

How to read the following table to have the `identical()` information as in `tidy_txt_df` from question above?

```
tmp_messy_table <- "# This line is just useless info

1_colname,2_colname,3_colname
3,4,5

a,b,c"
```

```
# To have the identical information as in the previous table we have to check
# which lines are comments. We can do this with `comment.char` parameter.

messy_text_df <- read.csv(text = tmp_messy_table, comment.char = '#', check.names = F)
identical(messy_text_df, tidy_text_df)
```

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

3.3 Flat File - Q3

Read the `hollywood.tsv` (not *.csv) file into a `data.table` R object. What is the problem with `fread()`?

```
library(data.table)
file_holly_tab <- "extdata/hollywood.tsv"
holly <- as.data.table(read.delim(file_holly_tab), keep_row_names=T)
head(holly)
```

```
##                    Film   Genre Lead.Studio Audience..score..
## 1:           27 Dresses  Comedy         Fox                71
## 2: (500) Days of Summer  Comedy         Fox                81
## 3:   A Dangerous Method   Drama Independent                89
## 4:        A Serious Man   Drama   Universal                64
## 5:  Across the Universe Romance Independent                84
## 6:            Beginners  Comedy Independent                80
##    Profitability Rotten.Tomatoes.. Worldwide.Gross Year
## 1:     5.3436218                40      160.308654 2008
## 2:     8.0960000                87       60.720000 2009
## 3:     0.4486447                79        8.972895 2011
## 4:     4.3828571                89       30.680000 2009
## 5:     0.6526032                54       29.367143 2007
## 6:     4.4718750                84       14.310000 2011
```

```
class(holly)
```

```
## [1] "data.table" "data.frame"
```

```
holly_data_table <- fread(file_holly_tab, skip=1)
class(holly_data_table)
```

```
## [1] "data.table" "data.frame"
```

```
head(holly_data_table)
```

```
##    V1                   V2      V3          V4 V5        V6 V7         V8
## 1:  1           27 Dresses  Comedy         Fox 71 5.3436218 40 160.308654
## 2:  2 (500) Days of Summer  Comedy         Fox 81 8.0960000 87  60.720000
## 3:  3   A Dangerous Method   Drama Independent 89 0.4486447 79   8.972895
## 4:  4        A Serious Man   Drama   Universal 64 4.3828571 89  30.680000
## 5:  5  Across the Universe Romance Independent 84 0.6526032 54  29.367143
## 6:  6            Beginners  Comedy Independent 80 4.4718750 84  14.310000
##      V9
## 1: 2008
## 2: 2009
## 3: 2011
## 4: 2009
## 5: 2007
## 6: 2011
```

```
holly_cn <- c("ID", colnames(read.delim(file_holly_tab, nrows= 1)))
holly_cn
```

```
## [1] "ID"                "Film"              "Genre"
## [4] "Lead.Studio"       "Audience..score.." "Profitability"
## [7] "Rotten.Tomatoes.." "Worldwide.Gross"   "Year"
```

```
# setnames(holly, holly_cn)
# head(holly)
# Difference between them: We can keep row_names. data.frame has no row_names.
```

3.4 Flat File - Q4

Who was the oldest surviving passenger of the titanic accident (titanic.csv)? Tipp: `?subset`

```
tit_df <- read.csv("extdata/titanic.csv")
head(tit_df)
```

```
##   pclass survived                                             name    sex
## 1      1        1                    Allen, Miss. Elisabeth Walton female
## 2      1        1                   Allison, Master. Hudson Trevor   male
## 3      1        0                     Allison, Miss. Helen Loraine female
## 4      1        0             Allison, Mr. Hudson Joshua Creighton   male
## 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6      1        1                                  Anderson, Mr. Harry   male
##     age sibsp parch ticket     fare   cabin embarked boat body
## 1 29.00     0     0  24160 211.3375      B5        S    2   NA
## 2  0.92     1     2 113781 151.5500 C22 C26        S   11   NA
## 3  2.00     1     2 113781 151.5500 C22 C26        S        NA
## 4 30.00     1     2 113781 151.5500 C22 C26        S       135
## 5 25.00     1     2 113781 151.5500 C22 C26        S        NA
## 6 48.00     0     0  19952  26.5500     E12        S    3   NA
##                         home.dest
## 1                     St Louis, MO
## 2 Montreal, PQ / Chesterville, ON
## 3 Montreal, PQ / Chesterville, ON
## 4 Montreal, PQ / Chesterville, ON
## 5 Montreal, PQ / Chesterville, ON
## 6                     New York, NY
```

```
survivor_name <- subset(tit_df, survived==1 & age==max(age, na.rm = T), name)
survivor_age <- subset(tit_df, survived==1 & age==max(age, na.rm = T), age)

survivor_name
```

```
##                                   name
## 15 Barkworth, Mr. Algernon Henry Wilson
```

```
survivor_age
```

```
##    age
## 15  80
```

3.5 Excel Questions - Q1

Read only Name, Type and Total columns for only the first 10 pokemons of the pokemon.xlsx file.

3.6 Excel Question - Q2

Which athlete won most bronze medals?

3.7 Excel Question - Q3

Are the columns Gender and Event_gender consistent?

3.8 Excel Question - Q4

Which country won most medals? Which country has the highest ratio of silver medals? Use the data in the country summary sheet starting at row 147.

3.9 Excel Questions

Which countries did participate, but without winning medals?

3.10 XML - Q1

Load the XML document plant_catalog.xml. Use XPath and DOM functions to find out all unique element names in the document.

Get all plants of zone 4 and transform the data into an R list.

3.11 XML - Q2

Read the tables HTML tables from the TUM website of dates for the winter term https://www.tum.de/en/studies/application-and-acceptance/dates-and-deadlines/ dates-and-deadlines-17/ into your workspace. When are the Christmas holidays?

3.12 JSON - Q1

Read the countries.json file. Which countries have common border with Jordan? Which country has the most neighbors?

3.13 JSON - Q2

Read this JSON file about projects funded by the world bank: world_bank.json.zip. Be aware, you might need to add syntax elements like "[" and "," to convert the file into textbook JSON format, i.e. readable by R. What was the most expensive project?

3.14 SQL - Q1

Use the extdata/Northwind.sl3 SQLite data base and retrieve a table that lists for all customers (name of the company, name of the contact person and city) all the products (name of the product) that they ordered. How many rows does this table have? Display the first 5 rows.