Introduction to R: solutions

9 - 13 March 2026

1 Introduction & Basics

Solution 1

(a) Calculate how many US$ is 15 euro:

```
> # This is more or less the current exchange rate
> 15 * 1.13776
>
> # It is better to use a variable to store the exchange rate
> exrate <- 1.13776
> exrate
> 15 * exrate
```

(b) Round the number to the second decimal:

```
> curr <- round(15 * exrate, digits = 2)
> curr
```

(c) Inspect the object:

```
> mode(curr)
> str(curr)
> summary(curr)
```

Solution 2

(a) The most important windows in RStudio are: Script, Console, Environment, History, Files, Plots, Packages, Help, and Viewer.

(b) Calculate mean and standard deviation:

```
> vec <- seq(11, 30)
>
> # Don't call the variable 'mean' since the function 'mean' already exists
> xmean <- sum(vec) / length(vec)
> xmean
> mean(vec)
>
> # Calculate the standard deviation in three (baby) steps
> numerator <- sum((vec - xmean)^2)
> denominator <- (length(vec) - 1)
> xstd <- sqrt(numerator / denominator)
> xstd
> sd(vec)
```

(c) The Environment window shows all objects you have created (like `vec`, `xmean`, `xstd`). The History window shows all commands you have executed.

2 Data: vectors

Solution 3

(a) Generate a vector from 11 to 30:

```
> vec <- seq(11, 30)
> vec
```

(b) Select the 7th element:

```
> vec[7]
```

(c) Select all elements except the 15th:

```
> # Use '-' to exclude elements
> vec[-15]
```

(d) Select the 2nd and 5th element:

```
> # Use the concatenate function 'c'
> vec[c(2, 5)]
```

(e) Select only the odd valued elements:

```
> # Instead of "hard coding" the length of the vector by specifying the value 20,
> # you can calculate it using the function 'length'. This makes your code more
> # generic and less error prone
> index <- seq(1, length(vec), by = 2)
> vec[index]
```

3 Data: matrices

Solution 4

(a) Create a character matrix:

```
> letters
> M <- matrix(letters[1:18], nrow = 6, ncol = 3)
> M
```

(b) Letter on the fifth row and second column:

```
> M[5, 2]
```

(c) Replace the last column:

```
> M[, 3] <- letters[21:26]
> M
```

(d) Replace vowels:

```
> # Let's assume that "y" is also a vowel
> vowels <- c("a", "e", "i", "o", "u", "y")
> M[M %in% vowels] <- "vowel"
> M
```

(e) Replace consonants:

```
> # '!=': not equal to
> M[M != "vowel"] <- "consonant"
> M
```

(f) Count consonants and vowels:

```
> sum(M == "consonant")
> sum(M == "vowel")
```

4 Data: data frames, lists and booleans

Solution 5

(a) Data structure of mtcars:

```
> # See '?mtcars', it is a data frame. On the command line you can use 'class' or 'str'
> class(mtcars)
> str(mtcars)
```

(b) Car models with more than 25 mpg:

```
> sum(mtcars$mpg > 25)
>
> # Other solutions using different ways of extracting a column from a data frame
> sum(mtcars[, "mpg"] > 25)
> sum(mtcars["mpg"] > 25)
> sum(mtcars[, 1] > 25)
```

(c) Names of car models with more than 25 mpg:

```
> rownames(mtcars[mtcars$mpg > 25, ])
```

(d) Select car models with mpg < 22 and more than 6 cylinders:

```
> # Use the Boolean operator '&' (logical AND)
> mtcars[(mtcars$mpg < 22) & (mtcars$cyl > 6), ]
```

(e) Additional selection with carburators or weight:

```
> # Use the Boolean operator '|' (logical OR).
> # Note that the weight is specified per 1000 lbs (see ?mtcars)
> mtcars[(mtcars$mpg < 22) & (mtcars$cyl > 6) &
+          ((mtcars$carb > 3) | (mtcars$wt < 3.5)), ]
>
> # The number of car models is the number of rows, that is the first dimension
> dim(mtcars[(mtcars$mpg < 22) & (mtcars$cyl > 6) &
+              ((mtcars$carb > 3) | (mtcars$wt < 3.5)), ])
>
> # The number of rows can also be retrieved with 'nrow'
> nrow(mtcars[(mtcars$mpg < 22) & (mtcars$cyl > 6) &
+               ((mtcars$carb > 3) | (mtcars$wt < 3.5)), ])
```

(f) Recode the am variable:

```
> mtcars$am_char[mtcars$am == 0] <- "automatic"
> mtcars$am_char[mtcars$am == 1] <- "manual"
```

```
> mtcars[, c("am", "am_char")]
```

Solution 6

(a) Inspect the islands dataset:

```
> data(islands)
> head(islands)
> str(islands)
> help(islands)
```

(b) Number of landmasses:

```
> # From ?islands: "Description: The areas in thousands of square miles of
> # the landmasses which exceed 10,000 square miles." So all of them:
> length(islands)
```

(c) Boolean vector for landmasses $>$ 20,000 square miles:

```
> # Remember that area was given in thousands of square miles
> islands.more20 <- (islands > 20)
```

(d) Select islands with landmasses $>$ 20,000 square miles:

```
> # Use the Boolean vector islands.more20 to select the ones with value TRUE
> islands[islands.more20]
```

(e) Character vector with names:

```
> islands.names <- names(islands)
```

(f) Remove Moluccas:

```
> notMoluccas <- islands.names != "Moluccas"
> islands.withoutMoluccas <- islands[notMoluccas]
>
> # Another solution is:
> islands.withoutMoluccas <- subset(islands, islands.names != "Moluccas")
```

5 Importing, saving and managing data

Solution 7

(a) Import the titanic data:

```
> library(foreign)
> titanic3 <- read.dta("titanic3.dta", convert.underscore = TRUE)
```

(b) View the data set in spreadsheet format:

This can be done using `View` on the command line. You can also use the menu: in R use **Edit - Data editor**, in RStudio click on the name of the data set in the **Environment** window.

```
> View(titanic3)
```

(c) Inspect first and last records:

```
> head(titanic3, n = 4)
> tail(titanic3, n = 4)
```

(d) Study dim and str output:

```
> dim(titanic3)
> str(titanic3)
```

Solution 8

```
> # You will get different results depending on your configuration
> search()
> ls()
> getwd()
```

Solution 9

```
> ls("package:base", pattern = "mean")
```

6 Data manipulation

Solution 10

```
> # Here the summary is shown for four of the variables only
> summary(titanic3[, c("survived", "pclass", "home.dest", "dob")])
```

Solution 11

(a) Summarize the age variable:

```
> # Note that you have to convert percentages into probabilities
> quantile(titanic3$age, probs = c(0.05, 0.25, 0.5, 0.75, 0.95), na.rm = TRUE)
> IQR(titanic3$age, na.rm = TRUE)
```

(b) Tables for survived and sex:

```
> table(titanic3$survived)
> table(titanic3$sex, titanic3$survived)
```

Solution 12 (OPTIONAL)

Running the following line of code:

```
> titanic3.select <- read.table("titanic3select.txt", sep = "\t", header = TRUE)
```

throws an error message: `Error in scan: line 16 did not have 18 elements`

This error message can in principle be corrected by adding the argument `fill=TRUE`, which adds blank fields if rows have unequal length. However, running:

```
> titanic3.select <- read.table("titanic3select.txt", sep = "\t",
+                               header = TRUE, fill = TRUE)
```

throws a warning message: `EOF within quoted string`

It turns out that the data was not correctly imported, which is often easily diagnosed by inspecting the dimensions. Indeed the number of rows is 25 instead of the expected 30. This is caused by a trailing ' at row 25 for variable `home_dest`. This can be corrected by changing the default set of quoting characters:

```
> titanic3.select <- read.table("titanic3select.txt", sep = "\t",
+                               header = TRUE, fill = TRUE, quote = "\"")
```

A closer inspection of the data shows that the last column only contains NAs caused by trailing spaces. On the course website there is a corrected version of the file where the last column and the trailing ' at row 25 were deleted.

```
> titanic3.select <- read.table("titanic3select_corrected.txt",
+                               sep = "\t", header = TRUE)
```

Solution 13

(a) Add a factor variable for survival status:

```
> # Survival (0 = No; 1 = Yes)
> titanic3$status <- factor(titanic3$survived, labels = c("no", "yes"))
```

(b) Difference between pclass and home.dest:

```
> as.numeric(head(titanic3$pclass))
> as.numeric(head(titanic3$home.dest))
```

For `pclass`, which is a factor, `as.numeric` returns the underlying integer codes (1, 2, 3 corresponding to the factor levels). For `home.dest`, which is a character variable, `as.numeric` returns `NA` with a warning because character strings cannot be directly converted to numbers.

Solution 14

(a) Passengers older than 70:

```
> # Note that the function 'subset' also leaves out the passengers for which the
> # variable 'age' is missing
> subset(titanic3, age > 70)[, c("name", "home.dest")]
>
> # Another solution is
> subset(titanic3, age > 70, select = c(name, home.dest))
>
> # Yet another solution is
> titanic3[titanic3$age > 70 & !is.na(titanic3$age), c("name", "home.dest")]
```

(b) Person from Uruguay:

```
> subset(titanic3, name == "Artagaveytia, Mr. Ramon")
> # No he didn't travel with relatives, since the variable 'family' is 'no'
```

(c) Table for first class passengers:

```
> xtabs(~sex + status, data = titanic3, subset = (pclass == "1st"))
```

Solution 15

```
> # If not installed yet, first install the package dplyr that contains the function arrange
> # install.packages("dplyr")
> library(dplyr)
> data.sorted <- arrange(titanic3, age)
> head(data.sorted[, 1:5], 10)
>
> # Use is.na to exclude the passengers for whom 'age' is missing
> tail(subset(data.sorted, !is.na(age))[, 1:5], 10)
```

You will notice that the youngest passengers are mostly in 3rd class, while the oldest passengers are more often in 1st class.

Solution 16

```
> summary(subset(titanic3, pclass == "1st")$fare)
> summary(subset(titanic3, pclass == "2nd")$fare)
> summary(subset(titanic3, pclass == "3rd")$fare)
```

Instead of writing three lines of code, applying the `summary` function to each of the three subsets, we could write a `for` loop:

```
> for (i in 1:3) {
+   print(summary(subset(titanic3, pclass == levels(pclass)[i])$fare))
+ }
```

The function `aggregate` is yet another option:

```
> aggregate(fare ~ pclass, data = titanic3, FUN = summary)
```

7 Programming structures in R

Solution 17

```
> apply(subset(titanic3, select = c("age", "fare", "body")), 2, mean, na.rm = TRUE)
```

Solution 18

```
> data(chickwts)
```

1. Mean weight for each feed type:

```
> tapply(chickwts$weight, chickwts$feed, mean)
```

2. Number of chicks with weight over 300 grams:

```
> sum(chickwts$weight > 300)
```

Solution 19

(a) Write a function:

```
> greater300 <- function(x) {
+   sum(x > 300)
```

```
+ }
```

(b) Calculate per feed type:

```
> tapply(chickwts$weight, chickwts$feed, greater300)
```

8 Graphics

Solution 20

(a) Create the figure:

```
> par(mar = c(5, 5, 5, 5))
> plot(3, 3, main = "Main title", sub = "subtitle",
+      xlab = "x-label", ylab = "y-label")
> text(3.5, 3.5, "text at (3.5,3.5)")
> abline(h = 3.5, v = 3.5)
> for (side in 1:4) mtext(-1:4, side = side, at = 2.3, line = -1:4)
> mtext(paste("side", 1:4), side = 1:4, line = -1, font = 2)
```

(b) Save to png and pdf:

```
> png(file = "figure.png")
> # Add code here to plot the figure (including par(mar=c(5,5,5,5)))
> dev.off()
>
> pdf(file = "figure.pdf")
> # Add code here to plot the figure (including par(mar=c(5,5,5,5)))
> dev.off()
```

(c) Save to pdf with custom dimensions:

```
> pdf(file = "figureLarge.pdf", width = 21, height = 21)
> # Add code here to plot the figure (including par(mar=c(5,5,5,5)))
> dev.off()
```

Solution 21

```
> data(quakes)
```

(a) Scatterplot of latitude versus longitude:

```
> plot(quakes$long, quakes$lat)
```

(b) Categorize magnitude and depth:

```
> mag.cat <- with(quakes, cut(mag, breaks = c(4, 4.5, 5, 7),
+                             include.lowest = TRUE))
>
> # More generic using the functions 'min' and 'max' that return the
> # minimum and maximum, respectively, of a vector
> mag.cat <- with(quakes, cut(mag, breaks = c(min(mag), 4.5, 5, max(mag)),
+                             include.lowest = TRUE))
> levels(mag.cat) <- c("low", "medium", "high")
>
```

```
> depth.cat <- factor(ifelse(quakes$depth > 400, "deep", "shallow"))
```

(c) Plot with symbols and colors:

```
> # Note that for 'col' the factor 'depth.cat' is interpreted as numeric (deep=1 and
> # shallow=2) and that the first two colours of palette() are used. Strangely enough
> # for 'pch' this is not the case and the factor 'mag.cat' has to be converted to a
> # numeric vector using the function 'as.numeric'.
> plot(quakes$long, quakes$lat, pch = as.numeric(mag.cat), col = depth.cat)
```

(d) Calculate energy:

```
> energy <- (10^(3/2))^quakes$mag
```

(e) Calculate symbol size:

```
> symbolsize <- sqrt(energy) / median(sqrt(energy))
```

(f) Plot with symbol size by energy:

```
> plot(quakes$long, quakes$lat, cex = symbolsize, col = depth.cat)
```

Solution 22

```
> par(mfrow = c(3, 1))
> # Make the margins smaller than default
> par(mar = c(2, 4, 2, 2))
> hist(titanic3$age, breaks = 15, freq = FALSE)
> boxplot(titanic3$fare, ylim = c(0, 300), ylab = "fare")
> plot(fare ~ pclass, data = titanic3, ylim = c(0, 300), ylab = "fare")
```

9 Documentation and help

Solution 23

1. Create character vector:

```
> AMC <- c("Academic", "Medical", "Center")  # a character vector named AMC
> mode(AMC)
> is.character(AMC)
```

2. Abbreviate and combine:

```
> help.search("abbreviate")
> help.search("combine")
> help.search("quote")
>
> # help(abbreviate)
> abbreviate(AMC, 1)  # selects first character
>
> # help(paste)
> paste(abbreviate(AMC, 1), collapse = "")  # gives AMC
>
> noquote(paste(abbreviate(AMC, 1), collapse = ""))  # removes quotes
```

Solution 24

```
> help(crosstab)
> help.search("crosstab")
> RSiteSearch("crosstab")
>
> # For more advanced search:
> install.packages("sos")
> library(sos)
> findFn("crosstab")
```