R for Data Science - Final exam

University of Milano - Bicocca, Department of Economics, Management and Statistics

2023-04-05

Instructions

The first and second exercises of this exam are related to the key concepts we reviewed during Units 1 and 2 of this course (i.e. Tidyverse and Debugging). In particular, you are required to develop a small exploratory data analysis (Exercise 1) and debug an existing R function (Exercise 2). The third and last exercise is related to the remaining two Units (i.e. Git and R packages). You are going to work together as a group of R developers to extend the package we created in class, adding more functionalities.

A few remarks:

1. The answers for each question included in the first two exercises must be submitted as a single PDF/HTML file created using an R Markdown or Quarto document. If you are not familiar with those file formats, see here or here. Each group must email me both files, i.e. the PDF/HTML document that includes your solutions AND the corresponding `.Rmd`/`.qmd` file used to generate it.
2. I will check the enhanced version of our package directly from the Github page of one member of each team. Therefore, you are supposed to email me just a link to the shared repository and no additional document. Please notice that I will ignore any commit pushed after the deadline.
3. The extended version of the R package must pass the R CMD checks without any error or warning. You are not allowed to delete or change any of the test we implemented together in class. The new functionalities must be properly documented and you must add a few basic tests.

If you face any problem, feel free to send me an email or just come by my office!

Deadline to email/push the solutions: April 30th, 2023 at 23:59.

Exercise 1 - Exploratory Data Analysis (12pt)

The `covfefe_chat.txt` file (that I sent to each one of you by email) includes a copy of the text messages exchanged in the Covfefe Whatsapp group during the last few months. The file was saved using the "Whatsapp Text Data" format and can be conveniently read into R as follows:

```r
library(rwhatsapp)
covfefe_chat <- rwa_read(
  here::here("exam", "covfefe_chat.txt"),
  encoding = "UTF-8"
)
```

Whatsapp text data analysis in R

The vignette of the R package `rwhatsapp` provides a great introduction to the analysis of whatsapp text data. You might want to read it to get inspiration on how to solve the following exercises.

If we check the structure of the dataset

```
library(dplyr, warn.conflicts = FALSE)
glimpse(covfefe_chat)
## Rows: 1,610
## Columns: 6
## $ time       <dttm> 2022-10-25 17:50:03, 2022-10-25 17:51:03, 2022-10-25 17:52~
## $ author     <fct> Tommaso Rigon, Luca Presicce, Roberto Ascari, Tommaso Rigon~
## $ text       <chr> "Hi Valentina! Nice to meet you!", "Welcome Valentina! <U+0001F601>",~
## $ source     <chr> "D:/git/R4DS-PhD-Unimib/exam/covfefe_chat.txt", "D:/git/R4D~
## $ emoji      <list> <NULL>, "<U+0001F601>", "<U+0001F601>", <NULL>, <NULL>, <NULL>, <NULL>,
## $ emoji_name <list> <NULL>, "beaming face with smiling eyes", "beaming face wi~
```

we can see that there are 1610 rows and 6 column with a quite descriptive name.

Using the Covfefe dataset, answer the following questions:

1. Who sent the highest number of messages?
2. How many messages were exchanged during December 2022? **Tip:** Check the functions exported by `lubridate` package if you need to extract the "month" from the `time` field
3. Who sent the first message of the current year? At which time?
4. On which day did we exchange the highest number of messages? After filtering the corresponding text messages, check their content and try to explain the anomalous behaviour.
5. How many messages are sent on average per day?
6. Who sent the highest number of messages which included at least one emoji? **Tip:** As we can see from the output of `glimpse()`, the `emoji` column is a `<list>` column. The following code can be used to select only the not-`NULL` values from a list-column named `col` in a dataset named `data`: `data |> filter(!vapply(col, is.null, logical(1)))`.
7. **(Difficult)** Determine the most common emoji for each author. In case of ties, you can select any of the equally-used emojies. **Tip:** The `unnest()` function (which is defined in the R package `tidyr`) can be used to "unnest" a list column. See the corresponding help page and the vignette of `rwhatsapp` for more details. In any case, you don't need to "parse" the UTF-8 codes.
8. **(More difficult)** Compute and display the total number of messages exchanged in the whatsapp chat after dividing the observations according to the hour of the day AND the day of the week. **Tip:** The function `tidyr::complete` can be used to fill the "implicit" missing values (i.e. those combinations of day and hour where no message was sent). Filling the 0 counts might help for the development of the visualisation.

Please notice that the previous questions can be answered using the `tidyverse` tools that we briefly reviewed during the first two classes of this course, but you can also use any other R package (or even no extra package apart from the basic ones). The only important part is that you provide the right answers with reproducible code.

Exercise 2 - Debugging techniques (8pt)

According to the official R documentation, the `termplot()` function can be used to plot the regression terms included in a linear model against their predictors (i.e. the product of $\hat{\beta}_j$ and x_j). So, for example, if we define a small simulation study such as

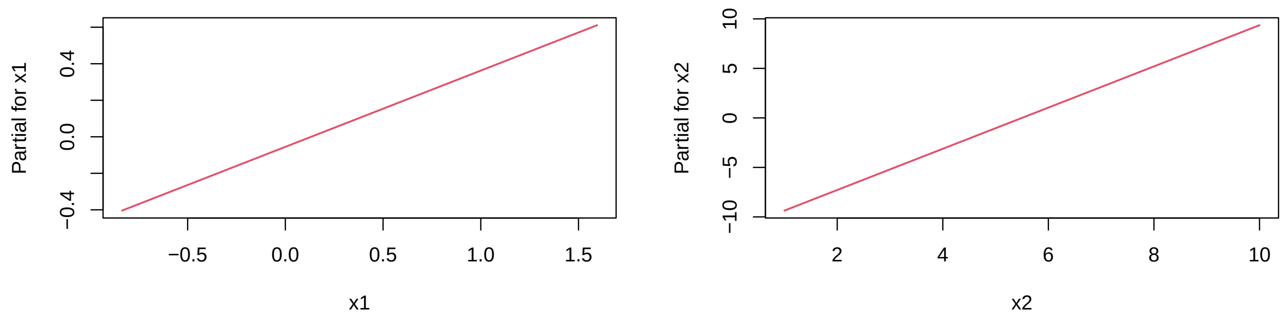
```r
set.seed(1)
n <- 10L
x1 <- rnorm(n)
x2 <- seq.int(n)
beta0 <- 0; beta1 <- 1; beta2 <- 2
y <- beta0 + beta1 * x1 + beta2 * x2 + rnorm(n)
```

then we can obtain a least square estimate of β_0, β_1, and β_2 as follows

```r
(mod1 <- lm(y ~ x1 + x2))
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Coefficients:
## (Intercept)           x1           x2
##     -0.1165       0.4176       2.0804
```

and the following command plots x_j vs $\hat{\beta}_j x_j$, $j = 1, 2$:

```r
termplot(mod1, ask = FALSE, ylim = "free")
```



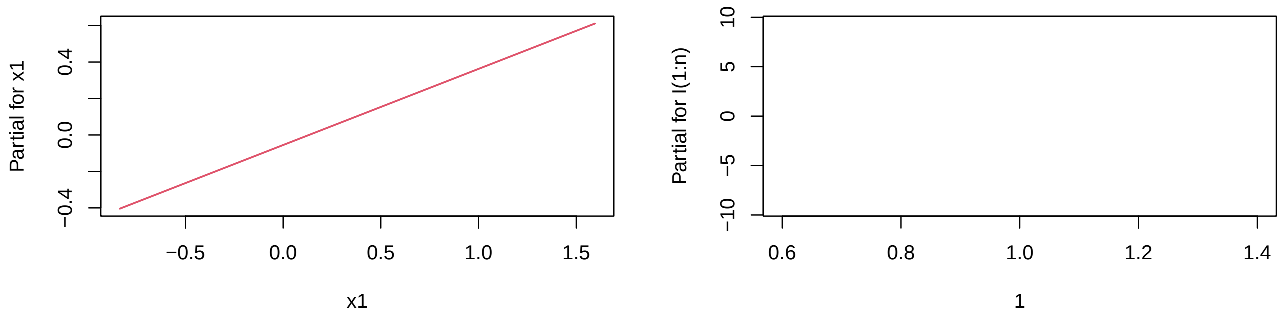
The `I()` function can be used inside a formula to stop the interpretation of its argument, indicating that it should be treated "as is". See also its help page for more details. Therefore, `mod1` can also be defined as follows:

```r
(mod2 <- lm(y ~ x1 + I(1:n)))
##
## Call:
## lm(formula = y ~ x1 + I(1:n))
```

```
##
## Coefficients:
## (Intercept)           x1       I(1:n)
##     -0.1165       0.4176       2.0804
```

Nevertheless, the following chunk of code shows that, in this second case, the `termplot` function returns an empty plot and a suspicious warning which is clearly misleading:

```
termplot(mod2, ask = FALSE, ylim = "free")
## Warning in termplot(mod2, ask = FALSE, ylim = "free"): 'model' appears to
## involve interactions: see the help page
```



Questions:

1. Explain why `termplot()` raises a warning message when we pass `mod2` instead of `mod1` and the steps you took / the techniques you used to tackle this problem.
2. **(Difficult)** Explain why `termplot()` creates an empty plot when displaying the relationship between `x2 := 1:n` and its predicted values.

In both cases, you are not required to actually solve the bugs. You just have to show me you can describe the problems behind the current implementation.

Exercise 3 - R packages (16pt)

During the last two classes of our course we developed a small R package named `statsAndBooze` that can be used to perform the following task:

```
library(statsAndBooze)

# Which days are you available to have a beer?
beer_dates_string <- list(
  andrea = c("2023-04-04", "2023-04-05"),
  federico = "2023-04-04"
)
```

```r
# Convert strings to Date(s)
beer_dates <- parse_dates(beer_dates_string)
decide_happy_hour(beer_dates)
## [1] "2023-04-04"
```

Now you are required to extend the currently-existing functionalities in the following ways!

1. As you may already know, I love beer and, more importantly, I'm a lazy guy… So, I'm (almost) always available to have a beer but I really don't want to manually enter all the single dates into the R script :(Therefore, now you have to extend the `parse_dates` function to allow the specification of time intervals instead of single dates. So, for example, given the following input

```r
beer_dates_string <- list(
  # I'm free on the 1st of April and from the 3rd to the 5th of April
  andrea = c("2023-04-01", "2023-04-03 / 2023-04-05"),
  federico = "2023-04-04"
)
```

our package should return something along these lines

```r
parse_dates(beer_dates_string)
# $andrea
# [1] "2023-04-01", "2023-04-03", "2023-04-04", "2023-04-05"

# $federico
# [1] "2023-04-04"
```

Please notice that each element of the list should be a vector of Dates!

Date Intervals in R

The R package `lubridate` implements a class of objects named Interval(s). So, for example, the following code creates and prints a time interval that starts on April 3rd, 2023 and finishes on April 5th, 2023.

```r
library(lubridate, warn.conflicts = FALSE)
(my_interval <- interval("2023-04-03 / 2023-04-05"))
## [1] 2023-04-03 UTC--2023-04-05 UTC
```

Given an `Interval` object you can also access the boundary points:

```r
int_start(my_interval)
## [1] "2023-04-03 UTC"
int_end(my_interval)
## [1] "2023-04-05 UTC"
```

From that point, you may also create a **seq**uence of Days and then…

2. **(Difficult)** Unfortunately, I'm much more lazy than that and I'm also getting old, so I can barely link dates and weekdays… Therefore, you need to further help me extending the `statsAndBooze`

package to allow the specification of weekdays instead of numerical dates! For example, suppose that today is April 5th, 2023 (i.e. the last day of classes together) and we know it's Wednesday. Then, if we assume that I will be up for a beer onThursday and Friday, the `parse_dates()` function should automatically convert those strings into the corresponding Dates:

```
beer_dates_string <- list(
  andrea = c("thursday", "friday"),
  federico = "2023-04-05"
)
parse_dates(beer_dates_string)
# $andrea
# [1] "2023-04-06", "2023-04-07"

# $federico
# [1] "2023-04-05"
```

! Restriction

If you want, you can assume that the strings indicating the names of the weekdays refer to the subsequent 7 days with respect to the current date when we run the `parse_date()` function.

3. At the end, you need to showcase the functionalities we developed together and the new ones creating a **beautiful** README file. If you need inspiration, see also here. Please notice that you just need to explain the installation process of your package and present its basic functionalities (i.e. the ones we coded together and the new ones).