Assignment #3: Coding Basics

6/21/2024

90/100 Points

Offline Score: **90/100**

Add Comment

Anonymous Grading: **no**

Details

This project asks you to write code to do a simple data analysis. This also involves group work, and of course GitHub.

This exercise has staggered due dates, see below.

As a reminder, you are allowed to use AI tools for this - or any other - task in this course. If you do, I suggest you add comments to your R/Quarto files to indicate where and how you used AI.

Group setup

As in the previous exercise, assign each member in your group an (arbitrary) number (I'm calling them M1, M2, ...). Make sure you are teamed up with a different person this time. For this exercise, everyone will first work on their own and finish this part by Wednesday. Then M1 will contribute to M2's repository, M2 will work on M3, etc. The last person (M3/M4/M5, based on the number of people in your group), will work on M1's repository. This way, everyone will work on their own and one group member's repository. Details are given below.

Part 1

Part 1 is due by **Tuesday**.

Getting started

Open **your portfolio website project** in RStudio. Then open the file for the coding exercise, `coding-exercise.qmd`.

Documenting well is very important! Add lots of comments to your code/file. I suggest that your code should be more than half comments. For every block of code, you have a few lines of comments at the beginning explaining what the code block does, and then each line of code gets its separate line of comment with more details. Comment on both the how and why of your code. This much commentary might seem overkill initially. But as your code gets more complex, it will be very useful. Both your collaborators, and your future self looking at the code you wrote several weeks ago will be incredibly thankful for your comments!

If you write R code, your comments will be lines that start with `#`. For Quarto or R Markdown files, you can either add comments as Markdown text above/below your code, and/or add comments inside your R code chunks. Both is ideal. R Studio allows you to quickly turn sections of a document into comments or un-comment them (In the `Code` section of the R Studio menu). That can be useful for turning on/off code during testing, or hiding some parts of text that's just meant for you but not for the reader.

Loading and checking data

We'll look at and play with some data from the `dslabs` package (https://cran.r-project.org/web/packages/dslabs/index.html). Write a code chunk using the `library()` function that loads the package (install the `dslabs` package first if you don't have it yet).

It is good practice to load all packages at the beginning of your code. So if you are using some R package, instead of loading it with `library` just before you use it, place all your `library` commands at the beginning of your R script or Quarto file. Also, add a short comment explaining why you are loading a certain package. For a complex project, it might even make sense to list all packages you use in a `readme` file. Even better is to use something like the `renv` package (https://rstudio.github.io/renv/) which keeps track of all your packages and makes sure someone running your project at a later time gets exactly the same packages you use. While I'm not requiring `renv` for the course, I encourage you to check it out and if you are up for it, use it for your portfolio or the project.

All the code you write for this (and any other) project should be written into an R or Quarto file, not in the R console. The reason for that is that you want a permanent record of what you did, and the ability to modify and re-run your analysis easily. For this exercise, you can either write the code directly into the Quarto document, or if you prefer the setup of a separate R script and pulling code into the Quarto document (see the data analysis template for an example), you can do it that way too.

We'll look at the `gapminder` dataset from `dslabs`. Once you have installed and loaded the `dslabs` package, the dataset is available. I.e., different than datasets you get from external sources, those that come with R packages are available right after you load the package. Write a code chunk using the `help()` function that pulls up the help page for the data to see what it contains. Then use the `str()` and `summary()` functions to take a look at the data. Use the `class()` function to check what type of object `gapminder` is.

To illustrate how that should look, you should have something like these lines of code and R output so far for this exercise.

```
#load dslabs package
library("dslabs")
#look at help file for gapminder data
help(gapminder)
```

```
starting httpd help server ... done
```

```
#get an overview of data structure
str(gapminder)
```

```
'data.frame':   10545 obs. of  9 variables:
 $ country         : Factor w/ 185 levels "Albania","Algeria",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ year            : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
 $ infant_mortality: num  115.4 148.2 208 NA 59.9 ...
 $ life_expectancy : num  62.9 47.5 36 63 65.4 ...
 $ fertility       : num  6.19 7.65 7.32 4.43 3.11 4.55 4.82 3.45 2.7 5.57 ...
 $ population      : num  1636054 11124892 5270844 54681 20619075 ...
 $ gdp             : num  NA 1.38e+10 NA NA 1.08e+11 ...
 $ continent       : Factor w/ 5 levels "Africa","Americas",..: 4 1 1 2 2 3 2 5 4 3 ...
 $ region          : Factor w/ 22 levels "Australia and New Zealand",..: 19 11 10 2 15 21 2 1 22 21 ...
```

```
#get a summary of data
summary(gapminder)
```

```
                country           year      infant_mortality life_expectancy
 Albania             :   57   Min.   :1960   Min.   :  1.50   Min.   :13.20  
 Algeria             :   57   1st Qu.:1974   1st Qu.: 16.00   1st Qu.:57.50  
 Angola              :   57   Median :1988   Median : 41.50   Median :67.54  
 Antigua and Barbuda:   57   Mean   :1988   Mean   : 55.31   Mean   :64.81  
 Argentina           :   57   3rd Qu.:2002   3rd Qu.: 85.10   3rd Qu.:73.00  
```

```
 Armenia           :   57   Max.   :2016   Max.   :276.90   Max.   :83.90
 (Other)           :10203                  NA's   :1453
   fertility       population             gdp               continent
 Min.   :0.840   Min.   :3.124e+04   Min.   :4.040e+07   Africa  :2907
 1st Qu.:2.200   1st Qu.:1.333e+06   1st Qu.:1.846e+09   Americas:2052
 Median :3.750   Median :5.009e+06   Median :7.794e+09   Asia    :2679
 Mean   :4.084   Mean   :2.701e+07   Mean   :1.480e+11   Europe  :2223
 3rd Qu.:6.000   3rd Qu.:1.523e+07   3rd Qu.:5.540e+10   Oceania : 684
 Max.   :9.220   Max.   :1.376e+09   Max.   :1.174e+13
 NA's   :187     NA's   :185         NA's   :2972
           region
 Western Asia   :1026
 Eastern Africa : 912
 Western Africa : 912
 Caribbean      : 741
 South America  : 684
 Southern Europe: 684
 (Other)        :5586
```

```
#determine the type of object gapminder is
class(gapminder)
```

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

Processing data

You can accomplish the next steps (and pretty much anything) with just basic R commands and not using extra functionality from packages. However, things are often easier with packages. For data processing tasks, the packages from the `tidyverse` (https://www.tidyverse.org/) are very useful. You can do the following tasks with any commands/packages you like.

Write code that assigns only the African countries to a new object/variable called `africadata`.
Run `str` and `summary` on the new object you created. You should now have 2907 observations, down from 10545. Depending on how you do this, you might also notice that all the different categories are still kept in the `continent` (and other) variables, but show 0. R does not automatically remove categories of what in R is called a factor variable (a categorical variable) even if they are empty. We don't have to worry about that just now, but something to keep in mind, it can sometimes lead to strange behavior.

Take the `africadata` object and create two new objects (name them whatever you want), one that contains only `infant_mortality` and `life_expectancy` and one that contains only `population` and `life_expectancy`. You should have two new objects/variables with 2907 rows and two columns. Use the `str`, and `summary` commands to take a look at both. Make sure you add comments into your code to explain what each line of code is doing, and as needed, also add additional explanatory Markdown text to your Quarto file.

I find it the least confusing to call *things which store values in R* **objects** (e.g., `x` is an object here: `x <- 2 + 2`) and reserve the word **variable** for a data variable, i.e., usually a column. However, it is common in programming to also refer to an **object** as a **variable**. Because of that, I sometimes use that terminology (inadvertently) too. So if I talk about a variable, you need to determine from the context if I mean a certain variable in the data (e.g. height or weight), or a variable in R (e.g. `x` or `result`) that stores some content.

Plotting

Using the new variables you created, plot life expectancy as a function of infant mortality and as a function of population size. Make two separate plots. Plot the data as points. For the plot with population size on the x-axis, set the x-axis to a log scale.

You should see a negative correlation between infant mortality and life expectancy, which makes sense. You should also see a positive correlation between population size and life expectancy. In both plots, especially the second one, you will see 'streaks' of data that seem to go together. Can you figure out what is going on here? Take another look at the `africadata` data we generated, which should give you a hint of what's happening. Add descriptive text into your Quarto file to explain what you see and why.

More data processing

I'm sure you realized that the pattern we see in the data is due to the fact that we have different years for individual countries, and that over time these countries increase in population size and also life expectancy. Let's pick only one year and see what patterns we find. We want a year for which we have the most data. You might have noticed that in `africadata`, there are 226 NA (i.e., missing values) for infant mortality. Write code that figures out which years have missing data for infant mortality. You should find that there is missing up to 1981 and then again for 2016. So we'll avoid those years and go with 2000 instead. Create a new object by extracting only the data for the year 2000 from the `africadata` object. You should end up with 51 observations and 9 variables. Check it with `str` and `summary`.

More plotting

Let's make the same plots as above again, this time only for the year 2000. Based on those plots, there seems to still be a negative correlation between infant mortality and life expectancy, and no noticeable correlation between population size and life expectancy. Let's apply some statistical model to this data.

Simple model fits

Use the `lm` function and fit life expectancy as the outcome, and infant mortality as the predictor. Then repeat, now with the population size as the predictor variable. (Use the data from 2000 only.) Save the result from the two fits into two objects (e.g. `fit1` and `fit2`) and apply the `summary` command to both, which will print various fit results to the screen. Based on the p-values for each fit, what do you conclude? Add comments into your quarto file to explain what you did and found.

Sending updates to Github

Once you are done with your exercise, re-build your portfolio website. Make sure no error messages show up. A preview should show up, check that the page for this exercise looks the way you want it to. Once you are happy with how everything looks, close RStudio. Go to GitKraken, commit your changes, and push to the remote repository. Check your portfolio website online to make sure you can now see the newly created R exercise document (in addition to your previously created `About` page). Of course, at any point, feel free to make further enhancements to your portfolio website.

Part 2

Part 2 is due by **Thursday**.

Adding to each other's work

Once you've done your first part, you'll contribute to another group member's project. As previously, find your classmates' portfolio repository and fork it, then clone to your local computer.

Open the repository. Remember to always do that by clicking on the `.Rproj` file in the main folder, which should open the project/repository in RStudio. Once you have the repository open, find the file for this exercise `coding-exercise.qmd`. Open it, make sure it runs/renders. Then add your part at the bottom of the file.

Start off by adding a comment that says something like **This section contributed by YOURNAME**. This needs to be there for me to be able to grade your contribution.

More data exploration

Pick another dataset from `dslabs`, whichever one you want. Unfortunately, the `dslabs` package doesn't have a nice website. But you can go to **its offical CRAN entry** (https://cran.r-project.org/web/packages/dslabs/index.html) and click on *Reference Manual*. The pdf lists the various other datasets and for each provides a brief explanation.

Once you chose one of the datasets, write R code to go through similar steps as above. Specifically, do the following:

- Explore the dataset.
- Do any processing/cleaning you want to do.
- Make a few exploratory figures. Optionally, also some tables.
- Run some simple statistical model(s). Your choice.
- Report the results from the model(s).
- For each step, add plenty comments to the code and explanatory Markdown text to the Quarto file.

Sending a pull request (PR)

Once you are done with your additions, make sure the whole website renders. Then push your updates to your fork on GitHub. Then follow the instructions from last week and issue a pull request to the original owner of the repository. They should get a notification, but you can also let them know that your updates are ready.

Part 3

Part 3 is due by **Friday**.

Merging the PR

Once you received the pull request, check it. Hopefully all is ok and merge it. If something isn't right, tell the person who sent the PR to fix what needs fixing and send another one.

Once you merged the PR, pull the updates to your local computer. Make sure everything looks fine and everything runs. Make any further needed changes/updates. Once everything is good, push your updated portfolio repository to GitHub. Check the website URL to make sure everything looks good. I'll be looking at the content shown on the website for assessment/grading purposes.

Since this will be part of your portfolio site, and you already posted a link to that previously, you don't need to post anything, I know where to find it.

(https://utsa.instructure.com/courses/42465/modules/items/2150470)

(https://utsa.instructure.com/courses/42465/n