Module 3 – lesson 05: Building a Document Template – part 1

Script

In the previous lesson we laid out an R script with some commands for exploring the steak\_survey dataset from the fivethiryeight R package. We will use that R script to help us build a document template for an article that you will write about steak preferences in the United States. And we will make some further edits to the R code in this lesson as we develop the steak survey article.

For this lesson, we will continue working with the Module 3 repository. Go ahead and log into your Github account and bring up your Module 3 repository. Then start RStudio and open the R project for Module 3.

The goal of this lesson is to create a document template for writing a basic article to serve as a template for additional articles based on the steak survey dataset.

To begin your Steak Survey Article, create a new R Markdown document in HTML format – Click File/New File/R Markdown – Documents HTML.

Set the title to

title: "Steak Preferences in the United States"

Put your name in for author, but also type in the word author so it is printed on the final document.

author: "Author: Melinda K. Higgins, PhD."

For the date, we’re going to use some R code to insert the current system date. The R code to get the current date is Sys.Date(). We can also use the format() function with the Sys.Date command to apply some formatting to the date. The code I’m going to show you will print the day of the week, the month, day and year. To learn more about formatting dates and times in R, see the help pages for the strptime function in R. The help page for strptime explains the formatting of dates and times using the format and Sys.Date functions. To get the current date printed with the day of the week, month, day and year, we use the following R code,

format(Sys.Date(), '%A, %B %d, %Y')

which runs the Sys.Date() function to get the current date, and then uses the format function to take that date and format it using the conversion specifications given by letters after the percent sign contained within single quotation marks.

To get this entire line of R code to run and print within the text we define for the date in our YAML header, we put the code in between 2 backtick marks where the 1st backtick mark has an r immediately after it. This allows us to insert R code inline with our text. Here is the final YAML entry for the date – with the whole set of text placed between 2 double quotation marks

date: "Published on: `r format(Sys.Date(), '%A, %B %d, %Y')`"

To see the result, click Save – name your new R markdown HTML file “steakArticle.Rmd” – click KNIT to HTML to see the result. Check your title, author and date.

We’re going to make one more slight adjustment to the YAML header. It turns out there are additional options available for the YAML header beyond what is shown in the basic R markdown document formats reviewed on the Rmarkdown website. Remember that the R markdown files are actually processed through Pandoc to convert the document into their final formats of HTML, DOCX or PDF. So, there are additional options available that Pandoc understands and can process. [You can learn more about these additional Pandoc options at <http://pandoc.org/MANUAL.html#variables-set-by-pandoc> ] One of these additional options is a subtitle.

We’re going to add a subtitle to setup customizing the specific region in future reports. However, for right now, type in the following subtitle in the YAML header. We can add this after the title.

subtitle: "Summary Report For the \_\*\*Mountain\*\*\_ Region"

I have added 2 types of emphasis syntax to the word Mountain – the single underscore at the beginning and end will make the word italics and the double asterisks at the beginning and end will make the word bold. So, the word Mountain will show up with both bold and italics formatting.

Next let’s spend a few minutes setting up the first R code chunk. Typically, it is a good idea to use the first R code chunk to do a few key things for setting up the environment of your document. What is the environment of my document and why should I care?– good question. Let’s take a moment and discuss the R environment.

Go ahead and save your steakArticle R markdown document. Then open the R script that you developed in the last lesson. Take a moment and click on the ENVIRONMENT TAB in the top right window – notice that there are no objects currently listed.

In the R script, highlight the 1st 4 lines of code and click RUN. This code loads the R packages fivethirtyeight and tidyverse and then load the steak\_survey dataset into your R environment. Click the ENVIRONMENT TAB in the top right window. This shows that the steak\_survey dataset is now available in your local environment.

library(fivethirtyeight)

library(tidyverse)

data("steak\_survey", package="fivethirtyeight")

View(steak\_survey)

Now run the next 2 lines of R code

sdat <- na.omit(steak\_survey) %>%

filter(region=="Mountain")

This creates a new data frame object sdat which is a subset of the steak\_survey dataset. So, you now have 2 objects in your local R environment.

Let’s clear out all of the objects in your R environment – click on the little broom icon in the ENVIRONMENT TAB – or go to the Console window at the lower left and type this R command

rm(list = ls())

Or you can click on the RStudio menu option for Session/Clear Workspace. Now our local environment is empty.

Let’s also “restart” our R session – click on Session/Restart R. This unloads all of the current R packages and puts at back at the state the RStudio software is in when we first start RStudio.

Now that our environment is empty and we have restarted our R session, try running the last 2 lines of R code again

sdat <- na.omit(steak\_survey) %>%

filter(region=="Mountain")

You will get an error – this is due to the fact that the R packages we need are no longer loaded and because the dataset we needed to make the subset from is also no longer loaded in the local environment.

It turns out, when you KNIT a document, the software reverts to a clean state without the packages loaded and with a completely empty environment. This seems like a major hassle when you first start trying to create R markdown documents – especially if you are working back and forth between writing and testing R code in a script or from the command line in the Console window and then putting that code in an code chunk in an R markdown document. However, there is a method to the madness. R markdown documents are considered to be “self-contained” in that it doesn’t matter what you are doing in your local R environment, when you KNIT an R markdown document you begin with a clean slate which does avoid other problems – mainly that anyone can KNIT your R markdown document and not be worried that something in their local environment will cause a problem compiling the document.

The point of this exercise was to point out that when we create an R markdown document, we need to make sure to load everything we need for the R code to run inside the R code chunks within the document. That means we need to set all of the document options, load the R packages we need, load the datasets we need and create all of the objects we need inside the document.

So, let’s go back to the first R code chunk. In this code chunk we are going to do 3 things

1. set up the knitr options we want for the whole document

2. load all of the R packages we want for the whole document

3. and load the main steak\_survey dataset and create the subset data object sdat we will be working with for this document

KNITR OPTIONS

# define knitr options

knitr::opts\_chunk$set(echo = FALSE)

knitr::opts\_chunk$set(warning = FALSE)

knitr::opts\_chunk$set(message = FALSE)

In this first R code chunk, we will set the KNITR options for ECHO, WARNING and MESSAGE all to FALSE. This basically hides the code as well as any warnings or messages to make the final document cleaner. It is useful to have these set to TRUE while you are still debugging, but in a final document, it is good to set all of these to FALSE. The very first line has a comment inserted with the R code which is designated by beginning with a single hashtag #. Inside R code chunks, hashtags are used to insert comments. These are NOT headers – that is only for hashtags in the document itself outside of the R code chunks.

There is one more KNITR option we will add which tells KNITR to print a blank space when there is missing data in a table made using the kable function rather than printing “NA” which is what R uses for not available when there is missing data.

# set any missing NAs in tables to blank

options(knitr.kable.NA='')

The next section of the code, loads all of the R packages that we will use to build this document. Take a moment and install any of these packages. It is worth noting here that you must have the packages installed first – outside of the R markdown document. If the package is not installed, then you will get an error when the document is compiled using KNITR. This is another reason to list all of the R packages that your document needs right at the beginning. Not only does this first code chunk set up the environment for your document, but by having the packages listed right at the beginning then anyone else using your document template will know they need to install these packages before compiling the document.

Here is the code to load 5 packages

# load R packages

library(fivethirtyeight)

library(tidyverse)

library(knitr)

library(kableExtra)

library(ggthemes)

Previously, we’ve discussed the fivethirtyeight and tidyverse packages. In this document we will also be creating tables for data summaries – to make the tables we will be using the knitr package function kable so to use the kable function we have to load the knitr package. The knitr package should already be installed. There is another useful package called kableExtra which helps make nicer tables out of the basic knitr kable function. So, take a minute and install the kableExtra package from the CRAN repository by clicking on Tools/Install Packages. You will also need to install the ggthemes package which will use to make our ggplot graphic a little nicer in the final document.

To complete this first R code chunk, go ahead and add the next 3 lines of R code we had in the R script earlier and add a few descriptive comments

# load steak\_survey dataset

data("steak\_survey", package="fivethirtyeight")

# create subset, no missing NAs

# pick a specific region

sdat <- na.omit(steak\_survey) %>%

filter(region=="Mountain")

This completes the first R code chunk. Click Save and KNIT to HTML to test that everything works and you do not get any errors.

We’ll stop here for this lesson. In the next lesson we will add the rest of the document template for our article on steak preferences in the United States.

But before we stop this lesson, go ahead and back everything up to your Github account.

Open Git Bash and make sure you are in the correct directory:

C:\RepTemplates\Module3

Once in that directory, type in the following 4 Git commands to check the status of your local files compared to your Github cloud repository; add or stage the modified files; commit your changes; and then push the changes to your Github cloud repository.

git status

git add .

git commit –m “add steakArticle document template”

git push

Now go to your Github repository, refresh to see your newly committed files.