Reporting in R – Word

Ryan Conrardy

## Quarto and R Markdown

R Markdown is the traditional method for generating code and output into a document. Uses Knitr to combine code and text to create documents or presentations. For more on R Markdown see <https://rmarkdown.rstudio.com/>.

Like R Markdown Quartro uses Knitr to execute the R code and can even run R Markdown documents. Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see <https://quarto.org>.

Both run on a similar framework. You have text interspersed with blocks of code (code blocks) that run to produce output that is inserted into the text (either tables, graphs, or raw text). For this example I will be using Quarto, to produce a word document (this file) and a HTML document (other file). This will be done to show how the same exact code does have some differences in the output, as well as to show the unique benefits and pros/cons of different output document types.

## Importing Data and Libraries

Just like when working in the R environment normally, you import libraries by calling them. Ideally this is done at the top of the document before any other code is run, but you can call the libraries anywhere in the code as long as it is **before** any functions requiring them.

# list of libraries, not all were used for this exact analysis, but are included to show importing and output  
library(ggplot2)  
library(plyr)  
library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:plyr':  
  
 arrange, count, desc, failwith, id, mutate, rename, summarise,  
 summarize

The following objects are masked from 'package:stats':  
  
 filter, lag

The following objects are masked from 'package:base':  
  
 intersect, setdiff, setequal, union

library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ forcats 1.0.0 ✔ stringr 1.5.0  
✔ lubridate 1.9.2 ✔ tibble 3.2.1  
✔ purrr 1.0.2 ✔ tidyr 1.3.0  
✔ readr 2.1.4

── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::arrange() masks plyr::arrange()  
✖ purrr::compact() masks plyr::compact()  
✖ dplyr::count() masks plyr::count()  
✖ dplyr::desc() masks plyr::desc()  
✖ dplyr::failwith() masks plyr::failwith()  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::id() masks plyr::id()  
✖ dplyr::lag() masks stats::lag()  
✖ dplyr::mutate() masks plyr::mutate()  
✖ dplyr::rename() masks plyr::rename()  
✖ dplyr::summarise() masks plyr::summarise()  
✖ dplyr::summarize() masks plyr::summarize()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(gtsummary)

Attaching package: 'gtsummary'  
  
The following object is masked from 'package:plyr':  
  
 mutate

library(readxl)  
library(table1)

Attaching package: 'table1'  
  
The following objects are masked from 'package:base':  
  
 units, units<-

library(kableExtra)

Attaching package: 'kableExtra'  
  
The following object is masked from 'package:dplyr':  
  
 group\_rows

Data can be read in and saved to the enviroment (usually as either a table or Dataframe) just like in Base R as well. Below is a sample of reading in a simple dataset on Food info. It consists of 8 columns (category, item name, amount, calories, protein, fat, carbs, and fiber).

# reading in the data via the readxl package  
Food <- read\_xlsx("C:/Users/rconrardy/Documents/Side Projects/Presentations/Data Peers/Producing Documents R/sampledatafoodinfo.xlsx", sheet = "FoodList", na = c("NA", "na", "N/A", "n/a", "", " "), col\_names = T)  
  
head(Food, n = 5)

# A tibble: 5 × 8  
 Category `Food Item` Measure Calories Protein Fat Carbs Fibre  
 <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
1 Baked Goods Angelfood, commercial … 1/12 73 2 NA 16 0.4  
2 Baked Goods Angelfood, from mix (2… 1/12 129 3 NA 29 0.1  
3 Baked Goods Animal crackers (arrow… 2 45 1 1 7 0.2  
4 Baked Goods Apple crisp, homemade 125mL 206 2 4 39 1.8  
5 Baked Goods Apple, commercial, 2 c… 1/8 296 2 14 43 2

# reading in a csv dataset with readcsv function  
Test <- read.csv("C:/Users/rconrardy/Documents/Side Projects/Presentations/Data Peers/Producing Documents R/Death\_rates\_for\_suicide.csv", na.strings = c("NA", "na", "N/A", "n/a", "", " "))  
  
head(Test, n = 5)

INDICATOR UNIT  
1 Death rates for suicide Deaths per 100,000 resident population, age-adjusted  
2 Death rates for suicide Deaths per 100,000 resident population, age-adjusted  
3 Death rates for suicide Deaths per 100,000 resident population, age-adjusted  
4 Death rates for suicide Deaths per 100,000 resident population, age-adjusted  
5 Death rates for suicide Deaths per 100,000 resident population, age-adjusted  
 UNIT\_NUM STUB\_NAME STUB\_NAME\_NUM STUB\_LABEL STUB\_LABEL\_NUM YEAR YEAR\_NUM  
1 1 Total 0 All persons 0 1950 1  
2 1 Total 0 All persons 0 1960 2  
3 1 Total 0 All persons 0 1970 3  
4 1 Total 0 All persons 0 1980 4  
5 1 Total 0 All persons 0 1981 5  
 AGE AGE\_NUM ESTIMATE FLAG  
1 All ages 0 13.2 <NA>  
2 All ages 0 12.5 <NA>  
3 All ages 0 13.1 <NA>  
4 All ages 0 12.2 <NA>  
5 All ages 0 12.3 <NA>

R can read other formats of data too with other functions. Depending it may require a special package, or packages may exist to make it easier than the function built into base R.

## Making Tables

There are plenty of packages for producing tables. Especially summary tables, such as table1, gtsummary, or even by hand using Kable. For this example I will show a quick table in gtsummary.

### Gtsummary

gtsummary is another package capable of making tables. This includes functions like tbl\_summary (shown below) for summary statistics, tbl\_regression for regression tables, and more. Generally it is more robust than Table1, but can often take longer to load/render. For more on gtsummary click here <https://www.danieldsjoberg.com/gtsummary/>.

tbl\_summary(data = trial,   
 by = trt,   
 include = c(age, marker, stage, grade, death))

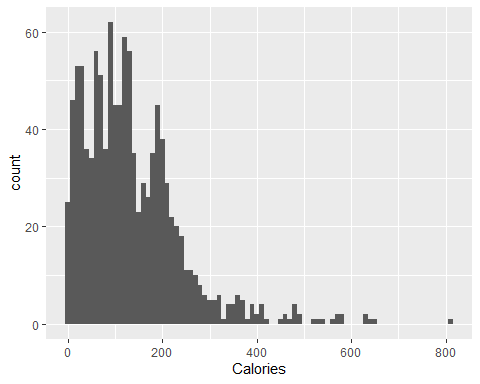
Table printed with `knitr::kable()`, not {gt}. Learn why at  
https://www.danieldsjoberg.com/gtsummary/articles/rmarkdown.html  
To suppress this message, include `message = FALSE` in the code chunk header.

| \*\*Characteristic\*\* | \*\*Drug A\*\*, N = 98 | \*\*Drug B\*\*, N = 102 |
| --- | --- | --- |
| Age | 46 (37, 59) | 48 (39, 56) |
| Unknown | 7 | 4 |
| Marker Level (ng/mL) | 0.84 (0.24, 1.57) | 0.52 (0.19, 1.20) |
| Unknown | 6 | 4 |
| T Stage |  |  |
| T1 | 28 (29%) | 25 (25%) |
| T2 | 25 (26%) | 29 (28%) |
| T3 | 22 (22%) | 21 (21%) |
| T4 | 23 (23%) | 27 (26%) |
| Grade |  |  |
| I | 35 (36%) | 33 (32%) |
| II | 32 (33%) | 36 (35%) |
| III | 31 (32%) | 33 (32%) |
| Patient Died | 52 (53%) | 60 (59%) |

## Making Figures

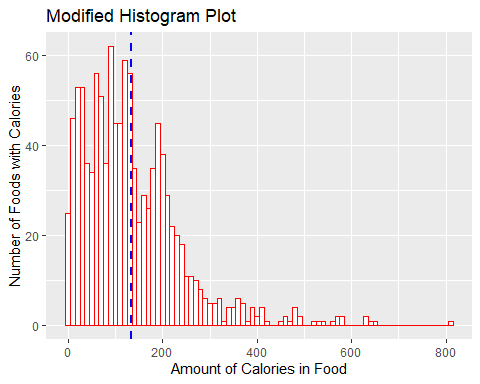
By far the best package used for making figures is ggplot2, however depending on the type of figure you need or want to make other packages exist. Much like how code blocks are used to generate tables, code blocks are also used to create figures.

ggplot(Food,   
 aes(x=Calories)) +   
 geom\_histogram(binwidth = 10)

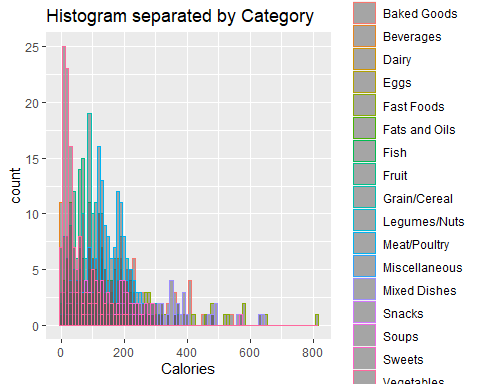


ggplot allows you to completely modify graphs, such as changing color, appearance, labels, titles, and more.

ggplot(Food,   
 aes(x=Calories)) +   
 geom\_histogram(binwidth = 10,  
 color = "red",  
 fill = "white") +  
 geom\_vline(aes(xintercept = mean(Calories)),  
 color = "blue",  
 linetype = "dashed",  
 linewidth = 1) +  
 labs(title = "Modified Histogram Plot",   
 y = "Number of Foods with Calories",   
 x = "Amount of Calories in Food")



# separate groups  
ggplot(Food,   
 aes(x=Calories, color = Category)) +   
 geom\_histogram(binwidth = 10,  
 alpha = 0.5,   
 position = "identity") +  
 labs(title = "Histogram separated by Category")



## Hiding Code and Code Output

Sometimes (or most of the time) you don’t want the client to see the raw code. Either cause it doesn’t look good, or because it’s not important. You can do this simply by adding #| echo: false to the top of the command arguments. For example: here is how the code would look:

# #| echo: false  
  
print("hello world")

and here is just the output of that code, with no visible code:

[1] "hello world"

You can also suppress warnings, messages, or even the output with other commands such as:

#| warning: false <-- suppresses warnings in the output

#| eval: false <-- just echos the code, but does not run the code (this can be done if you want to show code, but not actually run it)

#| output: false<--don’t include the results of executing the code in the output

#| error: false <-- suppresses errors in output (implies that errors executing code will not halt processing/rendering of document)

#| include: false <-- prevents any output (code or results) from being included in output

These and more execution options can be found here <https://quarto.org/docs/computations/execution-options.html>.