Literate Programming & Introduction to R Markdown

Eddie Kasner

June 09, 2025

Learning Objectives

Today we will:

- What literate programming is and why it's important
- How to use R Markdown for literate programming
- Use a built-in dataset to practice Rmarkdown

Reproducibility

For the purpose of this discussion, let's define:

- ***Replicate***: to repeat a study (with new samples and data)
- ***Reproduce***: to regenerate the results of an analysis (with the same data)

Our studies may be difficult to fully *replicate* . . . but our analysis should always be *reproducible*.

Or in other words:

We should always be able to go from raw data to final results at any time . . . *and always get the same results* from the same data and the same analysis*.

* Even if you are doing modeling with randomly-generated values, you can set the *seed* to produce the same values each time you run the analysis. There are situations where you may encounter small rounding differences, however.

Literate Programming

In the light of reproducibility, Literate Programming is:

Interspersing explanatory text with code to make a report that is executable and improves reproducibility.

It allows a recipient to re-render the report to review the code/analysis and verify the results (including plots and figures).

Literate Programming Tools

There are two main platforms for literate programming:

- ***Python*** and other languages: Jupyter Notebook (a future topic!)
- ***R***: R Markdown with *knitr* (today's focus) and R Notebooks

Both platforms can be used with other programming languages or a mix of them.

R Markdown

> "R Markdown was designed for easier reproducibility, since both the computing code and narratives are in the same document, and results are automatically generated from the source code. R Markdown supports dozens of static and dynamic/interactive output formats."

- See "basics" of R Markdown page

R Markdown and *knitr*

To use R Markdown and *knitr* for Literate Programming, all you need is a text editor, R, and a few packages such as *knitr*. These tools are well integrated into RStudio.

- R Markdown: The text notation you use for formatting, based on Markdown
- *knitr*: A package by Yihui Xie used to render R Markdown documents
- Check out Yihui Xie's *R Markdown: The Definitive Guide* e-book

Installing R Markdown and *knitr*

You need the R packages *htmltools*, *rmarkdown* and *knitr*.

The first time to try to "knit" in RStudio, you will likely be prompted to install these packages. If so, go ahead and install them as prompted.

Otherwise, you can install them manually:

```
#-----install new packages------

install.packages(c("htmltools", "rmarkdown", "knitr"), dependencies = TRUE)
```

R Markdown Syntax

R Markdown is a "wiki" syntax based on Markdown. Here is an example:

R Markdown Cheat Sheet

R Markdown cheat sheet

knitr

The R package which renders the R Markdown document is *knitr*. It uses other packages and utilities as needed to produce different document formats.

In RStudio, you would normally run *knitr* using the *Knit* button.

If your document has been configured in R Markdown to generate a PDF document, pressing the *Knit* button will create a `.pdf` file.

If you click the little black arrow button next to the *Knit* button, you can select alternative output options.

You can also use the *File -> Knit Document* or *File -> Compile Notebook* menu options.

Creating an `.Rmd` file

Let's make our first `.Rmd` file. This is similar to making a script:

```
File -> New File -> R Markdown...
```

Be sure to save it! The filename "example_markdown.Rmd" will work for now.

Features of the Markdown Document

- YAML Header
- Text
- Code chunks

Practice Rmarkdown with Example Dataset `airquality`

Let's practice using Rmarkdown with one of R's many built-in datasets - an air quality dataset from New York in 1973.

```
#-----load df------

# Load the "airquality" dataset into the environment.
airquality <- datasets::airquality
```

Show the structure of the dataset with `str()` or `head()`:

```
#-----inspect df------

# Show the head of the dataframe
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```r
# Show the structure of the dataset
str(airquality)
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Wrangle *airquality*

We've previously wrangled `airquality` wih `dplyr` - we'll repeat a few basic steps to practice with `Rmarkdown`. Using the pipe operator, `%>%`, will help us streamline operations so they are clear and succinct. The pipe operator feeds the output of one function into the input of the next, avoiding having to make multiple assignments.

```r
library(dplyr)
```

```r
#-----make may airquality-----

# Wrangle airquality with pipe operators starting with original dataframe.
airquality_may <- airquality %>%

  # Add year and date variables using mutate.
  mutate(Year = 1973,
         Date_char = paste(Year, Month, Day, sep = "-"),
         Date = as.Date(Date_char, format = "%Y-%m-%d")) %>%

  # Filter dataframe to the month of May.
  filter(Month == 5) %>%

  # Select the variables of interest.
  select(Ozone, Temp, Date)

# Show the dataframe.
head(airquality_may)
```

```
##   Ozone Temp       Date
## 1    41   67 1973-05-01
## 2    36   72 1973-05-02
## 3    12   74 1973-05-03
## 4    18   62 1973-05-04
## 5    NA   56 1973-05-05
## 6    28   66 1973-05-06
```

What if we're interested in the average ozone and temperature by month? Would we have to repeat this process for each month? *dplyr* has easy and powerful functions to perform data wrangling tasks on groups: `group_by()` and `summarise()`

```
#-----create df by month-------

# Average by month, using the argument `na.rm = TRUE` to ignore the NA values.
# If you omit `na.rm = TRUE`, then any NAs will cause the mean to be NA.
airquality_by_month <- airquality %>%

  # Group dataframe by month.
  group_by(Month) %>%

  # Calculate average ozone concentration and temperature.
  summarise(Ozone_avg = mean(Ozone, na.rm = TRUE),
            Temp_avg = mean(Temp, na.rm = TRUE),
            .groups = "keep")

# Show dataframe
airquality_by_month
```

```
## # A tibble: 5 x 3
## # Groups:   Month [5]
##   Month Ozone_avg Temp_avg
##   <int>     <dbl>    <dbl>
## 1     5      23.6     65.5
## 2     6      29.4     79.1
## 3     7      59.1     83.9
## 4     8      60.0     84.0
## 5     9      31.4     76.9
```

The last argument of `summarise()`, `.groups` specifies how to leave the grouping when the operation is finished. In this case, we "keep" the grouping as it is. (This is an "experimental" feature as of *dplyr* 1.0.2.)

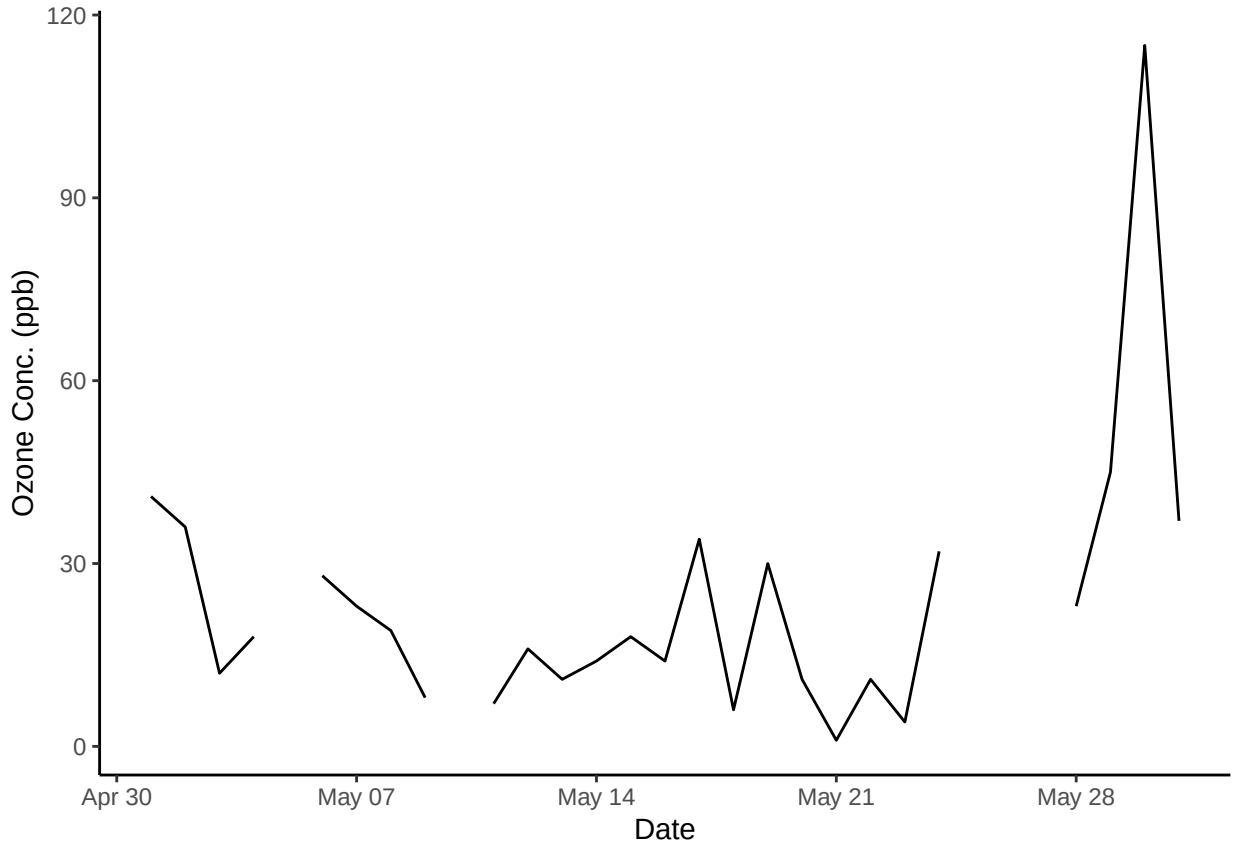
Make a plot using *ggplot2*

```
library(ggplot2)
```

Now let's make a simple plot so we have an example for our rendered document.

```
#-----make plot------

# Create a ggplot object and save it as "p".
p <- ggplot(data = airquality_may) +

  # Specify the type of geometry and the aesthetic features.
  geom_line(aes(x = Date, y = Ozone)) +

  # Specify the labels.
  labs(y = "Ozone Conc. (ppb)") +

  # Choose the theme.
  theme_classic()
```

```
# show plot
p
```



Session Information

It's a good idea to include the session information in your report.

```
## R version 4.5.0 (2025-04-11 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 26100)
##
## Matrix products: default
##   LAPACK version 3.12.1
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/Los_Angeles
## tzcode source: internal
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.5.2 dplyr_1.1.4   pacman_0.5.1  knitr_1.50
##
## loaded via a namespace (and not attached):
##  [1] vctrs_0.6.5        cli_3.6.5          rlang_1.1.6        xfun_0.52
##  [5] generics_0.1.4     textshaping_1.0.1  labeling_0.4.3     glue_1.8.0
##  [9] htmltools_0.5.8.1  ragg_1.4.0         scales_1.4.0       rmarkdown_2.29
## [13] grid_4.5.0         evaluate_1.0.3     tibble_3.2.1       fastmap_1.2.0
## [17] yaml_2.3.10        lifecycle_1.0.4    compiler_4.5.0     RColorBrewer_1.1-3
## [21] pkgconfig_2.0.3    rstudioapi_0.17.1  systemfonts_1.2.3  farver_2.1.2
## [25] digest_0.6.37      R6_2.6.1           tidyselect_1.2.1   pillar_1.10.2
## [29] magrittr_2.0.3     withr_3.0.2        tools_4.5.0        gtable_0.3.6
```

Render Markdown Document

Next, let's render the document using the "Knit" button!