A Data Scientist's Notebook in R

Gary Feng

July 8, 2017

Contents

Introduction

Who is the audiance you are writing for?

- R, because you want it to run your code. Sure.
- Yourself, because you know you won't recognize your own code in 2 weeks.
- Your professor, boss, or client, because they want to see the results.
- Science, because your work needs to be validated and replicated before accepted.

There you have it – a data science notebook is typically 3 parts for human consumption, 1 part for computer. Literate programming is your friend to this end.

- Literate programming: https://en.wikipedia.org/wiki/Literate_programming
- R Markdown http://rmarkdown.rstudio.com/ (check out the gallery) is the literate programming environment for R. I'd say it's one of the best and most useful LP implementations.

Anatomy of a data science notebook

Data science project reports often follow a basic pattern, which reflects the workflow of data analysis:

- *Report title*, authors, date, version, etc.
- *Introduction*, where you layout the basic problem and your approach
- *Setting up the environment* you will need for the rest of the projects; always load all your libraries upfront, so that whoever will replicate your analysis will be able to install them before going forward.

- *Data import and QC*: you need to document where your data come from, the filtering and transformations you did, whethere there are data quality issues and how you fixed them, missing and extreme values, etc.
- *Data analysis and visualization*: Personally I visualize the data first before serious modeling, but it depends on the problem.
- I recommend `ggplot2` for static visualization
- and something like `plotly (offline)` for interactive plots
- *Saving data*: think about what data you want to pass to others (including your future self) as the output of this work. RData is good if you know people will use R downstream; otherwise use common data exchange format such as CSV.
- *Conclusions*: summarize what you did, caveats, and to-dos.

You don't want your notebook to overgrow ... most often you want a notebook to cover a day's work, or shorter. It should be self-contained: it takes an input, does a thing, and generates a report (or saves new data). Plan your week-long project as a series of notebook.

Finally, be prepared (encouraged) to share your notebook (and data when appropriate) with others. It is your responsibility to ensure that "it just runs": - You document the requirements for setting up the environment, and the libraries needed to run the notebook - You specify up front the path to data, so that others can modify the path or URL if necessary - No manual interventions or magical leaps. Everything you did is in the R code chunk. If you have to do manually change the data in some ways, document precisely what the steps are so that others can replicate.

Think before you code

Before you jump to coding, sit down and write an overview of your notebook – what is the problem you are solving and what are potential solutions. This will be the most productive 15 minutes you have spent (compared to debugging your code).

Setting up the environment

Here you want to tell your audience how to replicate your work.

R and R Studio

Make sure you are using the latest - R (3.4.x) https://cran.r-project.org/bin/windows/base/ - RStudio https://www.rstudio.com/

- Install Latex on Windows: http://www.reed.edu/data-at-reed/software/R/r_studio_pc.html. You need this to "knit" to PDF. This is perhaps the most frustrating part with R on Windowns. The commended Miklatex did not work on my Windows system. I ended up installing TexLive (more than 1g download) and Pandoc to get it to work. But you may have better luck.

R libraries

You can install R libraries using RStudio (menu::Tools::install packages), or in R, using `install.packages("packageName")`

I recommend installing the following, in addition to what comes with RStudio, such as `knitr` etc. - tidyverse, which should include ggplot2, tidyr, dplyr and other Hadlay Wickham tools. See http://tidyverse.org/ - plotly https://plot.ly/r/getting-started/ - devtools (recommended for installing experimental libraries)

In the current notebook I only need to load `plotly` for graphing and `knitr` for generating a prettier table.

```
library(plotly)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(knitr)
```

Data import & QC

Reading data into R is not trival, because that's the first place where things can go wrong – some CSV files are actually tab-delimited; the header line is imported as a data row; a column of numbers are imported as strings becuase missing values are specified as a tring; you can add to this list.

Data quality control is absolutely necessary. Some of the things to check:

- inspect the data frame to see whether the columns make sense
- make sure the data type of each column is what you expected
- for string-based categorical variables, do a frequency table to make sure there are no extra trailing spaces in some labels or capitalization variations.
- check to see if missing data are properly defined; some people use 999 or -1 for missing values; make sure you set them as missing

Actually write down (and keep) your data QC code; never throw it away. It is the single most important element to ensure repeatable analysis.

In this notebook we use a data set built in to plotly, the `midwest` data frame. in RStudio, you can inspect the data frame with the following sonsole command `View(midwest)`.

Here we use the `kable` function in the `knitr` library to print summary stats of the variables in the data frame. `kable` generates a better looking table than the R `print` command. *Well, it's a little too wide, isn't it?*

```
kable(summary(midwest))
```

PID	county	state	area	poptotal	popdensity	popwh
Min. : 561	Length:437	Length:437	Min. :0.00500	Min. : 1701	Min. : 85.05	Min. :
1st Qu.: 670	Class :character	Class :character	1st Qu.:0.02400	1st Qu.: 18840	1st Qu.: 622.41	1st Qu.:
Median :1221	Mode :character	Mode :character	Median :0.03000	Median : 35324	Median : 1156.21	Median :
Mean :1437	NA	NA	Mean :0.03317	Mean : 96130	Mean : 3097.74	Mean : 8
3rd Qu.:2059	NA	NA	3rd Qu.:0.03800	3rd Qu.: 75651	3rd Qu.: 2330.00	3rd Qu.:
Max. :3052	NA	NA	Max. :0.11000	Max. :5105067	Max. :88018.40	Max. :32

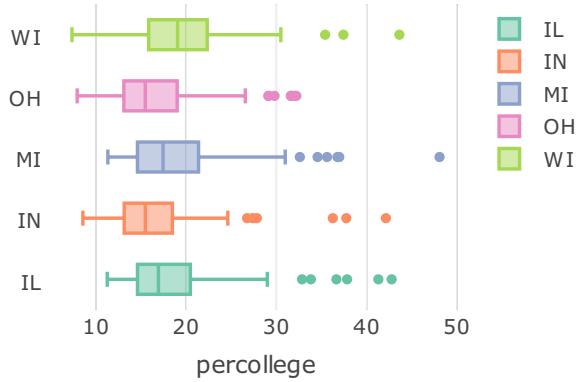
Data analysis & visualization

We will illustrate how to generate an interactive R visualization using plotly offline.

Generating a single interactive plotly offline plot

you will have an interacive plotly graph when knitting to HTML. If you knit to PDF or Word, you end up with a screenshot of the graph.

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.



Multiple interactive plots (HTML output only)

If you want to generate the same plot repeatedly for different variables or different dataset, the following examples generates multiple interactive HTML figures in an one-liner. The downside is that it only works in the HTML output, not PDF or Word.

```
htmltools::tagList(lapply(1:3, function(x) { plot_ly(x = rnorm(10)) }))
```

```
## No trace type specified:
##   Based on info supplied, a 'histogram' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#histogram
## No trace type specified:
##   Based on info supplied, a 'histogram' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#histogram
## No trace type specified:
##   Based on info supplied, a 'histogram' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#histogram
```

R output and plots

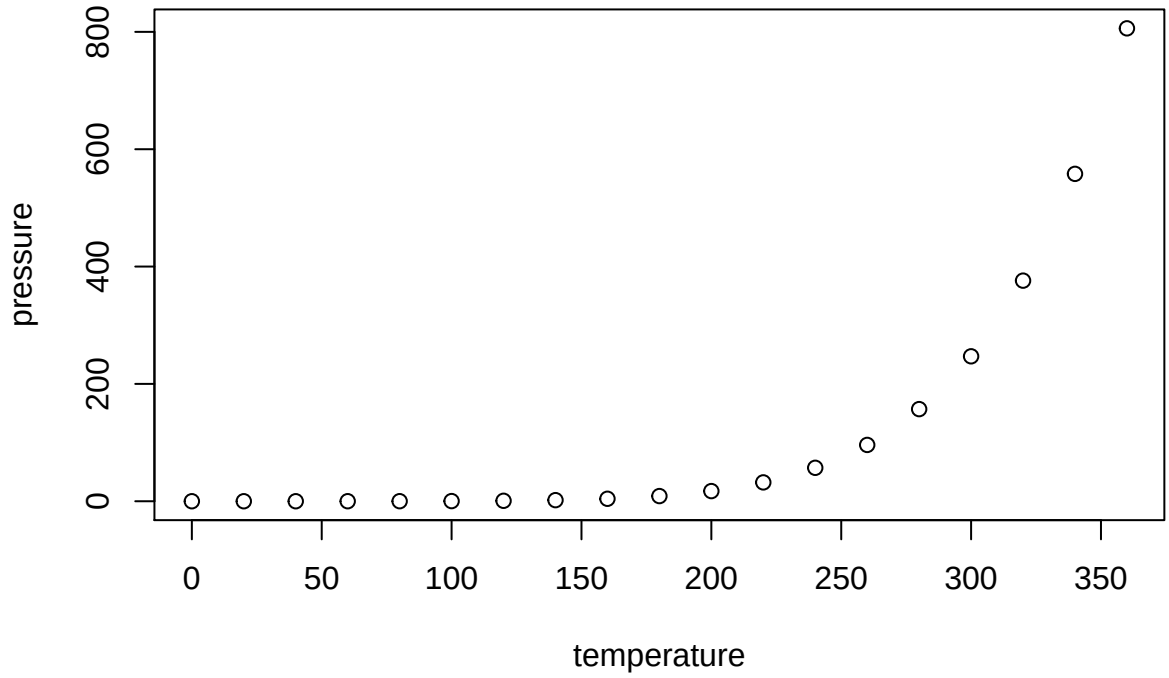
The old fashioned R output is of course also supported. Note the `{r cars}` line: `{r}` says this code block is in R. `cars` is the **optional** name of the code block.

Because with using named blocks – the name must be unique in the notebook. If you copy-and-paste a code block, make sure you change the name, or else you end up with cryptic error messages. You either use it conscientiously, or not at all.

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

You can also embed plots, for example:



Saving data

Before you save your data (and notebook), go back and clean up your code. Can you simplify it by refactoring repeated lines into functions? Can you reduce the number of data frames or junk variables? You will update your thought process the prose sections, starting from the intro.

Now you are happy with the logic and code. Time to save everything.

Conclusions

Always keep your audience in mind when you write. In this case, the audiance is 1 part R, 3 part human.

Humans are pretty bad at remembering or following instructions. The data scientist's notebook is a way to keep us disciplined (and less frustrated).

To-dos:

Additional topics to explore:

- More RMarkdown tools and tricks

- How to refine and refactor your R code
- Visualization: why ggplot is not intuitive until it is