Literate Programming

301113 Programming for Data Science

WESTERN SYDNEY
UNIVERSITY

School of Computer, Data and Mathematical Sciences

Week 7 - R for Data Science, Chapter 27

Outline

Motivation

We are encouraged to write comments in our scripts to ensure that the code and the logic behind it is understandable.

Data scientists write programs to manipulate and analyse data, therefore it is crucial that the processes are documented so that they are reproducible and readable. So rather then writing scripts, data scientists should be writing documents that explain their reasoning for the choice of analytical methods, with interspersed code to perform the analysis.

In this section, we are following the book “R for Data Science”, Chapter 27.
`https://r4ds.had.co.nz/r-markdown.html`

Outline

What is Literate Programming?

Literate programming is the process of writing human readable programs. It is used in scientific computing and in data science where code is shared and verified, to provide reproducible research.

Literate programs contain the code required to run the program, and they also contain descriptions of the algorithm and usually the design choices for the code.

Having the design choices documented mean that programmers are more likely to think about the design choices, leading to better code.

History of Literate Programming

The concept of literate programming came from the Donald Knuth (considered one of the godfathers of computer science) in the early 1980s.

The original concept was to have:

- a compiler to create the executable program from the program code (a process called tangling) and
- have another program similar to a compiler that could generate the documentation from the program code (a process called weaving).

The popularity of scripted languages now means that the literate program can be used as the program with out any processing.

Giving up email

Donald Knuth has many achievements, but the one that I most remember him for is giving up on email in 1990. `https://www-cs-faculty.stanford.edu/~knuth/email.html`

Popular literate programming practices

There are many methods to write literate programs and they differ depending on the programming language used.

Current popular methods are:

- Jupyter Notebooks: for many languages but mainly used with Python.
- R Markdown: for many languages but mainly used with R.

Note that these lecture slides and notes (that you are reading) are written using R Markdown.

There has been a rise in popularity with Jupyter notebooks, but not due to literate programming. The popularity is mainly due to their interactivity with python, which can lead to poor programming practices for inexperienced programmers. See `https://conferences.oreilly.com/jupyter/jup-ny/public/schedule/detail/68282.html`

Literate programming in Data Science

A data scientist converts data into information. The information is used for answering questions or making predictions.

A general process followed by a data scientist consists of

1. Importing data
2. Cleaning data
3. Manipulate data into a required form
4. Visualise properties of the data
5. Build models for analysis using the data
6. Communicating the analytical results

Literate programming allows a data scientist create scripts that not only provide the analytical results, but also generate reports to communicate the results. One method of performing this is to use R Markdown.

Outline

R Markdown Overview

R Markdown is a combination of R code and Markdown (a simplified markup language) that provides us with a way to embed code into reports. R Markdown is written in text files with the name suffix `.Rmd`.

R Markdown files are use to:

- Provide results of analysis to those who want the results.
- Collaborate with others, providing them with your code and analysis.
- Record your work, containing your code and descriptions of your analysis.

Installing packages for R Markdown

When using R Markdown in RStudio, the required libraries are automatically installed. If not using RStudio, install the library `rmarkdown`.

A first R Markdown file

Create a new R Markdown file to see an example.

Let's examine the components and compile it.

R Notebook

RStudio also provides the option of creating an R Notebook. The text file created is an R Markdown file, but the interaction with RStudio is slightly different.

Parts of an R Markdown file

An R Markdown file contains three components:

- A YAML header.
- A number of code chunks containing R code.
- Markdown text surrounding the code chunks.

The YAML header describes the metadata for the document, such as the author, date, style used for the processed document, the type of output document wanted.

The code chunks contain all of the R code. Many chunks can be created. Variables from previous chunks can be used in later chunks.

Creating the compiled document

RStudio provides the button `knit` to convert an R Markdown document into HTML, PDF or Word formats.

In the background the process to create the document is:

- `.Rmd` to `.md`: `knitr` converts all of the code chunks into markdown, providing a document containing only markdown.
- `.md` to `.html`, `.docx` or `.pdf`: The program pandoc then takes the markdown file and converts it to the required format.

Conversion to PDF requires LaTeX to be installed on the machine.

Note that we can use pandoc to convert plain markdown files to many formats and also converts formats to other formats. I have used pandoc to convert between Word documents and LaTeX files.

Outline

Separation of Content and Style

Most of us are likely to have formatted text by selecting it and clicking on buttons in the word processor (such as Microsoft Word). This type of document formatting is easy to use and also easy to abuse.

Examples I have received:

- Changes in font type or size throughout a document.
- Section, figure, table numbers not in order.
- Sections and subsection titles with inconsistent formatting.
- References to wrong sections or out of date bibliography.

Word provides the capability to define titles and sections to a document and define styles for consistency, but no one uses it.

Separation of Content and Style

Most of us are likely to have formatted text by selecting it and clicking on buttons in the word processor (such as Microsoft Word). This type of document formatting is easy to use and also easy to abuse.

Examples I have received:

- Changes in font type or size throughout a document.
- Section, figure, table numbers not in order.
- Sections and subsection titles with inconsistent formatting.
- References to wrong sections or out of date bibliography.

Word provides the capability to define titles and sections to a document and define styles for consistency, but no one uses it.

By separating the style and the content of a document, we instead:

- Write the content of the document and markup the document with tags describing each component, without distraction of formatting.
- Apply a style to the document describing how we want each component to look.

Examples of separating content and style: HTML and CSS, LaTeX and Class files.

Markdown

Markdown is a simple form of markup that describes the components of the document (but not the style of each component).

The markup is given by special sequences of characters, so a markdown file is a text file.

The goal of Markdown is to provide a markup language that is simple enough that the text is pleasant to read (unlike HTML).

Here is an example of a Markdown file describing Markdown:
`https://raw.githubusercontent.com/hadley/r4ds/master/rmarkdown.Rmd`

Markdown Formatting Syntax

Markdown uses the following syntax.

- The wanted text is written as text. Note that newline characters are ignored when converting to other formats.
- Paragraphs are defined by empty lines.
- Text can be italicised using single asterisks `*italic*` and made bold using double asterisks `**bold**`.
- Superscript uses carets `e^x^` and strikethrough uses tildes `~~wrong~~`.
- Code is presented encased in backticks.
- Hyperlinks can be presented as `<https://www.westernsydney.edu.au/>`, using the URL as the text or `[WSU](https://www.westernsydney.edu.au/)` replacing the URL with text.

Follow the Leader

Problem

Write the following text as markdown and compile it to check it.

Please don't submit work where ~~every~~ word has `markup` applied to it.
But make sure to use **paragraphs** when `changing` topic.

Markdown Structure Syntax

Document structure is provided by sections and sections in sections. Sections are provided using the hash symbol. The number of hashes provided defines the depth of the section.

- `# Section`
- `## Subsection`
- `### Subsubsection`
- `#### Subsubsubsection`

The section title lines must appear on a line of their own and the hash must be the first character of the line.

Markdown Lists

Lists can either be ordered or unordered. An unordered list requires all list items start on a new line and begin with a hyphen. Nested lists start with a few spaces.

```
- Item 1
- Item 2
  - Sub item 1
  - Sub item 2
- Item 3
```

If numbered lists are wanted, then replace the hyphens with a number and period.

```
1. Item 1
2. Item 2
  - Sub item 1
  - Sub item 2
3. Item 3
```

Follow the Leader Again

Problem

Write the following text as markdown and compile it to check it.

List of things to do:

1. Become proficient in R.
    - Attend all classes.
    - Ask lots of questions.
2. Don't forget to add +2 to dice rolls.
3. Teach friends R.

Maths in Markdown

Mathematics can be typeset in markdown using LaTeX notation. If you know LaTeX notation this is straightforward, otherwise there is a lot to memorise (or lots of Web searching).

Maths can be provided inline encased in dollar symbols `$y = ax^2 + bx + c$` or provided on its own line (equation style) using double dollar symbols

```
$$\hat{f}(\xi) = \int_\infty^\infty f(x)e^{-2\pi ix\xi}dx$$
```

produces

$$\hat{f}(\xi) = \int_{\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx$$

Maths in Markdown

Mathematics can be typeset in markdown using LaTeX notation. If you know LaTeX notation this is straightforward, otherwise there is a lot to memorise (or lots of Web searching).

Maths can be provided inline encased in dollar symbols `$y = ax^2 + bx + c$` or provided on its own line (equation style) using double dollar symbols

```
$$\hat{f}(\xi) = \int_\infty^\infty f(x)e^{-2\pi ix\xi}dx$$
```

produces

$$\hat{f}(\xi) = \int_{\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx$$

The Not So Short Introduction to LaTeX https://tobi.oetiker.ch/lshort/lshort.pdf was my main reference book when learning LaTeX. See section 3.10 for a list of symbols.

Follow the Leader Once More

Problem

Write the following text as markdown and compile it to check it.

The sum of integers 1 to n is $\sum_{x=1}^{n} x = (n+1)n/2$

Markdown Tables

There are methods of manually creating tables in Markdown (see the reference guide),

```
Age   | Weight | Eye Colour
------|------- | ----------
26    | 78.2   | Hazel
32    | 89.1   | Brown
```

But it is more likely that we will creating tables from data that we have obtained or from results we have computed.

Images in Markdown

Markdown is written in text, stored in a text file, so images cannot be placed in the file.

To include images, they must be referenced.

Any images must be stored in a file. The file is then referenced within the markdown.

```
![optional caption text](path/to/img.png)
```

Note that the file path can be written relative to the working directory, or the full path can be given. Also note that the path is not encased in quote marks.

Outline

Limitations of Markdown

Markdown is a simple markup language, but it is not designed for embedding code, and does not provide methods for defining sizes of images and tables when compiling to other formats.

Since Markdown is plain text, it does allow us to include any markup that can assist in generating HTML, PDF or Word documents. For example, we can include HTML in a Markdown document, that is later used if the document is converted to HTML.

R Markdown takes a step back and generalises Markdown documents to include R code chunks. Those chunks contain R code that is evaluated and the results placed in the markdown file.

If the evaluated R code provides HTML, then it can be used to further format the document if converted into HTML.

Code Chunks

Code chunks are defined regions of the R Markdown document that contains R code. Code chunks are the only difference between a Markdown and R Markdown document.

A code chunk begins with three backticks and `{r}` and ends with three backticks. These beginning and ending tags must be on a line of their own.

````
```{r}
z <- lm(y ~ x)
```
````

Code chunks can contain anything that could be placed in an R script and can contain as many lines as needed.

An R Markdown document can also contain as many chunks as needed.

Code Chunks

Code chunks are defined regions of the R Markdown document that contains R code. Code chunks are the only difference between a Markdown and R Markdown document.

A code chunk begins with three backticks and `{r}` and ends with three backticks. These beginning and ending tags must be on a line of their own.

````
```{r}
z <- lm(y ~ x)
```
````

Code chunks can contain anything that could be placed in an R script and can contain as many lines as needed.

An R Markdown document can also contain as many chunks as needed.

A single R session

All code chunks from an R Markdown document run in a single R session from the first chunk to the last, so any variables created in earlier chunks are available in later chunks.

Code Chunk Name

Code chunks can be named, but if not manually named, are provided a generated name. A name can be provided after the `r` in the opening braces.

Providing a name to a chunk helps in the following ways:

1. RStudio provides a chunk navigator that uses the names. Naming the chunks will allow easier navigation.
2. Files generated by chunks (such as plots) use the chunk name, so it helps in find the files.
3. It helps with results caching.

We will investigate this more later.

Code Chunk Options

Code chunks can have different purposes, for example, to run code, to display code, to provide the code and results. The options provided to each chunk control how it behaves. Commonly used options are:

- `eval = FALSE` to not evaluate the code, usually used to provide example code, or to stop code running when testing.
- `include = FALSE` evaluates the code, but does not show the code or the results.
- `echo = FALSE` evaluates the code and shows the results, but not the code.
- `message = FALSE` or `warning = FALSE` stops messages and warnings from R appearing in the document.
- `results = hide` stop results from printing.
- `fig.show = hide` prevents images such as plots from appearing.

The options are placed inside the braces that begin the chunk and separated with commas.

Hidden Code

Problem

Create a code chunk to compute the correlation between `mpg` and `disp` from the `mtcars` data, where the code is hidden, but the result is shown.
Call the code chunk `mtcarsCor`.

kable and kableExtra

kable is a library that converts tables in data frames into tables for the desired format (e.g. HTML or PDF).

kableExtra provide additional features to make tables prettier.

- To use kable the knitr library must be loaded using `library("knitr")`
- To use kableExtra the kableExtra library must be loaded using `library("kableExtra")`

A simple use of kable:

```
mts <- mtcars[1:5, 1:3]
kable(mts, comment = "Some of the cars data.")
```

kable and kableExtra

`kable` is a library that converts tables in data frames into tables for the desired format (e.g. HTML or PDF).

`kableExtra` provide additional features to make tables prettier.

- To use `kable` the `knitr` library must be loaded using `library("knitr")`
- To use `kableExtra` the `kableExtra` library must be loaded using `library("kableExtra")`

A simple use of `kable`:

```
mts <- mtcars[1:5, 1:3]
kable(mts, comment = "Some of the cars data.")
```

Details for kableExtra
https://haozhu233.github.io/kableExtra/awesome_table_in_html.html

Caching Chunk Evaluation

Each time the R Markdown document is compiled, all of the code chunks are evaluated from the first to the last. If a code chunk happens to include code that takes a long time to run, then we will have to wait a long time each time the document is compiled.

To avoid this repeated waiting time, code chunks can be cached. Caching saves the chunk and its evaluation to the hard drive as it is run. The next time the document is compiled, R will check if the code chunk has changed, and if not, it will use the cached results rather than recomputing them.

Code chunks can be cached using the chunk option `cache = TRUE`

Caching Chunk Evaluation

Each time the R Markdown document is compiled, all of the code chunks are evaluated from the first to the last. If a code chunk happens to include code that takes a long time to run, then we will have to wait a long time each time the document is compiled.

To avoid this repeated waiting time, code chunks can be cached. Caching saves the chunk and its evaluation to the hard drive as it is run. The next time the document is compiled, R will check if the code chunk has changed, and if not, it will use the cached results rather than recomputing them.

Code chunks can be cached using the chunk option `cache = TRUE`

Example

Finding the inverse of large matrices is time consuming. Let's create a large random matrix and compute its inverse and examine the effect of caching.

Chunk Dependencies and Caching

We stated that R checks if a chunk has been modified to determine if it can use the chunk cache. We also stated that all chunks use the same R session, so variables from previous chunks are available to use in later chunks.

If a chunk uses a variable from a previous chunk, and the previous chunk has been changed, then the cache will be stale, but R will not know.

R provides the chunk option `dependson`, which should be assigned a character vector of the names of all chunks that the current chunk depends on. If one of those change then, the cache will not be used.

Chunk Dependencies and Caching

We stated that R checks if a chunk has been modified to determine if it can use the chunk cache. We also stated that all chunks use the same R session, so variables from previous chunks are available to use in later chunks.

If a chunk uses a variable from a previous chunk, and the previous chunk has been changed, then the cache will be stale, but R will not know.

R provides the chunk option `dependson`, which should be assigned a character vector of the names of all chunks that the current chunk depends on. If one of those change then, the cache will not be used.

R also provides the chunk option `cache.extra` which can be assigned any value or data structure. If that value changes, then the cache is recomputed.

Global Chunk Options

Rather than inserting the same options into every chunk, the default options can be changed by setting the global chunk options.

The global chunk options are usually set at the top of the R Markdown file in their own chunk with option `include = FALSE`.

The global options are set by passing them to the knitr function `opts_chunk$set()`. For example, the following sets all chunks to not show their code in the compiled document be default:

```
knitr::opts_chunk$set(
  echo = FALSE
)
```

Multiple options can be provided and separated by commas.

Inline Code

There are many times where we need to refer to results or variables from the analysis in the text that is written. For example, a regression has been run and the mean squared error is reported in the main text.

We can include variables and even the results of function calls, in the text using inline code. To include the result, a backtick followed by `r` opens the inline code statement, then next backtick closes it. For example:

```
The average miles per gallon is `r mean(mtcars$mpg)`.
```

The function `format()` can be used to ensure that the number of decimal places is sensible.

Inline Code

There are many times where we need to refer to results or variables from the analysis in the text that is written. For example, a regression has been run and the mean squared error is reported in the main text.

We can include variables and even the results of function calls, in the text using inline code. To include the result, a backtick followed by `r` opens the inline code statement, then next backtick closes it. For example:

```
The average miles per gallon is `r mean(mtcars$mpg)`.
```

The function `format()` can be used to ensure that the number of decimal places is sensible.

Problem

Use the earlier correlation result between `mpg` and `disp` from the `mtcars` data. Write a statement stating this correlation with inline code.

YAML Header

The YAML header allows us to set many metadata items, such as `title`, `author`, `date`. It also allows us to set variables for the document, that can be used for the analysis and remain constant for the analysis.

For example, we can set the parameters:

```
params:
  mpgLowerLimit: 20
  transmission: 1
```

Then use these parameters to control the analysis:

```
mtcars2 <- subset(mtcars, subset = ((mpg > params$mpgLowerLimit) &
    (am == params$transmission)))
```

These variables could be set later in the document, but having them in the header clearly shows the parameters.

Outline

Bibliographies

Data science is a field where it is necessary to stand on the shoulders of giants. Therefore, any analysis will require citing others' work.

RStudio uses `pandoc` to convert markdown files to other formats. `pandoc` is also able to use bibliography files from different formats and include the references in your document when it is cited.

The process is:

1. Maintain a bibliography file containing details of publications.
2. Include the name of the file in the YAML header.
3. If an article in cited in the markdown, the reference will be placed at the end of the document and a citation to the reference at the desired place.

Creating a Bibliography File

`pandoc` can read BibLaTeX, BibTeX, Endnote and Medline bibliography files. If you have not created a bibliography file, the easiest is a BibTeX file. A bibfile is a text file, with bibtex entries for each reference. The entries can be hand written, but it is easier to get them from Google Scholar.

1. Go to `https://scholar.google.com.au/`
2. Click on the hamburger button and choose Settings.
3. Under "Bibliography Manager" select "Show links to import citations into BibTeX"

When searching Google Scholar, you will find "Import into BibTeX" beneath each result. Clicking on that will provide the text to insert into your bib file.

Citing an Article

Once you have included `bibliography: nameoffile.bib` in the YAML header, articles can be cited from the bib file using the items tag.

```
@inproceedings{park2018blended,
  title={A blended metric for multi-label optimisation and evaluation},
  author={Park, Laurence AF and Read, Jesse},
  booktitle={Joint European Conference on Machine Learning and Knowledge Discovery in Datab
  pages={719--734},
  year={2018},
  organization={Springer}
}
```

The tag for the above article is `park2018blended`. To cite the article in the markdown file, we use `[@park2018blended]`. Additional cites can be placed in the brackets, separated with a semicolon.

Citing an Article

Once you have included `bibliography: nameoffile.bib` in the YAML header, articles can be cited from the bib file using the items tag.

```
@inproceedings{park2018blended,
  title={A blended metric for multi-label optimisation and evaluation},
  author={Park, Laurence AF and Read, Jesse},
  booktitle={Joint European Conference on Machine Learning and Knowledge Discovery in Datab
  pages={719--734},
  year={2018},
  organization={Springer}
}
```

The tag for the above article is `park2018blended`. To cite the article in the markdown file, we use `[@park2018blended]`. Additional cites can be placed in the brackets, separated with a semicolon.

The references will be placed the end of the generated file, so add the section heading `# References` or `# Bibliography` to the last line of the R Markdown document.

Adding a Reference

Problem

In your R Markdown document, add a reference to the paper “Least angle regression” by B Efron et al.

R Markdown Formats

Once we have written the R Markdown document, it can be compiled into different formats, depending on the YAML header. For:

- PDF, use `output: pdf_document`
- HTML, use `output: html_document`
- Microsoft Word, use `output: word_document`

Note that compiling the document to PDF required LaTeX to be installed on the machine. To generate the PDF, the process is R Markdown -> [knitr] -> Markdown -> [pandoc] -> LaTeX -> [pdflatex] -> PDF. Pandoc is able to convert Markdown to HTML or Word, so no additional software is needed.

Outline

Interactive Programming

Programming using an interpreted (scripted) language (such as R) allows us to test code as we write it to ensure it is doing what we want it to do. Using an R Markdown file as a notebook, allows us to break our code into code chunks and test them one at a time.

R Markdown files allow us to develop code and record our reasoning and thoughts. This parallels with a traditional Laboratory notebook used in physical sciences. Using an R Markdown file as a notebook, we can:

- Record what was done and why it was done.
- Allows us to reflect on our work.
- Helps others to read our work.

Each of these points are useful when working in a team or on our own.

Using R Markdown as a Notebook

Tips for using R Markdown files as notebooks.

- Name your files appropriately, provide good titles and a summary paragraph describing the analysis.
- Record start data in the YAML header.
- Don't delete failed analysis. Keep a record to make sure the same path is not taken in the future.
- Don't modify data files, write code to convert it to a usable format.
- When finishing for the day, make sure that you can compile your file.
- To make sure that your code is reusable in the long term, take into account the versions of any external libraries used.
- Over the years, the number of notebooks will grow; make sure to organise them so they will be useful.

Summary

- Literate programming explains our code and help us to think about the algorithms.
- R Markdown files provide way for us to write literate programs.
- R Markdown files are Markdown text files with embedded R code chunks.
- R Markdown can be compiled to HTML, Word and PDF files.
- R Markdown files can be used as notebooks to record analysis and thoughts.