Week 00 Module 04: Working with R Notebooks

CSCI E-5a

In CSCI S-5a, we always start every R notebook by clearing the global computing environment.

We'll explore what this means in Lecture 2, but for right now just run this code:

```
rm(list = ls())
```

Module 4 Overview and Learning Objectives

In this module, we'll explore how to work with RStudio and R notebooks.

- In Section 1, we'll discuss what R and RStudio are, and how they work together.
- In Section 2, we'll learn some of the basics of working with the RStudio interface.
- In Section 3, we'll explore the R markdown system of formatting text and structuring documents.
- In Section 4, we'll learn about code chunks.
- In Section 5, we'll learn about LaTeX and pandoc.

When you've finished this module, you should be able to:

- Explain what R is, what RStudio is, and the relationship between the two.
- Use R markdown to perform simple text formatting, including bold, italic, verbatim, block quotes, lists and sublists, and basic tables.
- Use R markdown to specify the logical structure of an R notebook.
- Use the document outline feature of RStudio to efficiently navigate through the logical structure of an R notebook.
- Explain what LaTeX code is, and what to do with it.
- Explain what a code chunk is, and how it allows RStudio to pass expressions to R to for evaluation, as well as to display any results.
- Explain how to insert a code chunk into an R notebook.
- Explain how to control the execution of code chunks within an R notebook.
- Explain what LaTeX is, and what to do with LaTeX code.
- Explain strategies for dealing with pandoc errors.

For this module, if you're watching the video you'll need to open the R notebook – of course, if you're reading this, then you've already done that!

OK, we're all set up, so let's get an overview of R and RStudio.

Section 1: R and RStudio Overview

> **Main Concept:** *R is a computational engine, and RStudio provides an interface for R.*

In this section, we'll discuss what R and RStudio are, and how they work together.

R is a computer programming language that takes expressions, evaluates them, and returns an output value, and when you installed R you were installing this computing system.

To work directly with this programming language, you type expressions into a command line, R evaluates these expressions, and finally returns the result.

You did this already when you installed the `rmarkdown` package in Module 3.

While it's possible to work with R this way, it isn't very pleasant, and it's desirable to have some sort of interface to make the process easier for humans.

One such interface is RStudio, which is an *integrated development environment* (IDE), which means that it provides a complete system for writing, testing, and executing R code.

In CSCI E-5a, we'll use a special type of document called an "R notebook", which can integrate text, code, output, and graphics.

R notebooks feature a simple but effective system called "R markdown", which allows us to control the formatting of text as well as specify the logical structure of the notebook.

We'll discuss R markdown in Section 3 of this module.

It's important to understand that RStudio doesn't "know" anything about the R programming language.

RStudio by itself can't interpret R statements, and it can't execute R commands.

Instead, RStudio takes any code that we write and passes this to R for evaluation.

RStudio also displays any results from R.

So this is what an IDE is – it provides us with a powerful system for writing and executing code, but the actual computational work is done by R.

We'll discuss how this process works in greater detail in Section 4, on code chunks.

So that's the basic relationship between R and RStudio.

You'll become more comfortable with these concepts once you've gained a little experience.

Now let's move on to learning some of the basics of the RStudio interface.

Section 2: RStudio interface

> **Main Idea:** *Setting up the RStudio interface.*

Before we get rolling, let's first get familiar with the basics of the RStudio interface.

The main window that contains the text of this R notebook is called the "Source window", and this is where we'll spend most of our time.

Underneath the Source window is the Console window.

The Console window is usually not all that useful for our purposes in CSCI S-5a, so I'm going to suggest that you minimize it by clicking on the small icon in the upper right-hand corner of the Console window.

Likewise, you may see the Environment and Help panes displayed on the right-hand side of the screen.

These panels contain important information and we'll often want to consult them, but we don't need them on a consistent basis so I'm going to suggest that you minimize these as well.

You can resize these panes by moving your mouse pointer over the frame until it turns into a four-directional arrow, then click and drag the panes all the way to the right.

You can retrieve the panes by moving your mouse over the right-hand frame, when it will again turn into a four-direction arrow, and then click and drag.

At the left-hand side of the top of RStudio is a horizontal menu.

- The "File" drop-down menu allows you to create, save, and open files in RStudio, including R markdown documents.
- The "Edit" drop-down menu provides Undo / Redo, Cut, Copy, and Paste, and Find and Replace operations.

So that's the basics of working with the RStudio interface.

Now let's explore the R markdown system.

Section 3: R Markdown

> **Main idea:** *R notebooks use a simple system to format and organize text called R markdown.*

In this section, we'll explore the R markdown system of formatting text and structuring documents.

When we're working in an R notebook, we are really just typing text.

However, we can use a simple system to indicate text formatting to RStudio. RStudio will use this information when knitting the document.

Simple formatting

R markdown allows us to format text.

The text won't appear to be formatted when we view it in RStudio, but R will apply the formatting when we knit the document.

To put text in italics, surround it with single asterisks: *This is italic text.*

To put text in boldface, surround it with double asterisks: **This is bold text.**

To put text in verbatim, surround it with backticks: `This is verbatim text`.

To put text in a block quote, use a '>' character at the beginning of a line:

> This is a block quote.

Notice that the RStudio text editor can parse these simple markdown commands, and uses different colors for the text.

To create a bullet list, start the 1st item with an asterisk in the 1st position, and use indenting to indicate the 2nd level:

- Here is the 1st level of the bullet list.
  - Here is the 2nd level of the bullet list; notice the text indenting.
  - We're still in the 2nd level of the bullet list.
- Now we're back to the 1st level of the bullet list, because there is no text indenting.
  - We're back in the 2nd level of the bullet list, but now with a different bullet point symbol.

A nice example of how we can use R markdown is the Main Idea statement at the beginning of each section:

- It starts with a block quote character '>'.
- The text "Main Idea:" is in boldface, because it's surrounded by double asterisks.
- Finally the text of the main idea is in italics, because it's surrounded by single asterisks in the R notebook.

Let's knit this R notebook, and examine it to see how the formatting is rendered.

You can see that now in both the HTML and PDF files that the text is displayed using the R markdown formatting.

Specifying the logical structure of the document

Many different types of files use some sort of system to indicate the logical structure of the document. This is implemented using a system of headers with different levels. For instance, you can use headers with level 1 to indicate the main divisions within a document, and then use headers with level 2 to indicate subdivisions within each main division. This is a common method that is used in many different applications such as HTML or Microsoft Word.

To specify the logical structure of the document using R markdown, we use hashtag characters in the 1st position of the line.

A line starting with a single hashtag character is a level 1 header.

Here's an example of a level 1 header:

Example Level 1 Header

A line starting with two hashtag characters is a level 2 header.

Here's an example of a level 2 header:

Example Level 2 Header

R markdown supports up to 6 levels of headers, although I think that once you get beyond level 2 that it starts to become difficult to work with.

RStudio will automatically apply special formatting to the headers when it knits the document, increasing the font size and applying bold face.

You might have noticed that we've been using this system of headers throughout this document. For instance, at the top of the document the "Module Overview and Learning Objectives" section is a level 1 header. Likewise, this section started with a level 1 header for "Section 3: R Markdown", and then we had a level 2 header for "Specifying the logical structure of the document".

Let's knit this R notebook, and examine it to see how the header formatting is rendered.

You can see that now in both the HTML and PDF files that the header text is displayed using a larger font and bold face.

Navigating the R notebook using the logical structure

We can use the header system to actually visualize the logical structure of the document, as well as to navigate through the document. To do this, go to the upper right-hand corner of the editor pane, and find the "Show Document Outline" icon. When you click on this, the Document Outline pane slides out. Now this Document Outline displays the header structure of the document. For instance, you can see that the 1st entry is the Module 4 Overview and Learning Objectives, followed by Section 1: R and RStudio Overview. Notice that the level 2 headings are indented, so the Document Outline display is actually a visual representation of the header structure of the document.

We can also use the entries in the Document Outline to navigate to other sections of the document. For instance, let's jump up to the Module Overview and Learning Objectives section. To do this, we'll click on the corresponding entry in the Document Overview window. Notice that when we do this, RStudio automatically jumps to the Module Overview and Learning Objectives.

Let's try this again.

This time, we'll jump back to where we were before. The closest link is the Level 2 header titled "Navigating the R notebook using the logical structure". When we click on the entry in the Document Outline window, RStudio jumps to this level 2 header.

You can navigate through a document very easily using the Document Outline window, and I recommend incorporating this technique into your personal RStudio workflow.

So that's how to use R markdown to format text and structure documents.

Now let's find out about code chunks.

Exercise 1: Using R markdown

In CSCI S-5a, the lecture notebooks will often contain short exercises for you to do.

These short exercises usually consist of a simple variation of some method that has just been demonstrated, so they provide an opportunity for you to get some basic experience working with that method. They're not intended to be particularly challenging, and you should find them straightforward to do if you've understood the preceding material.

The exercises are for your benefit, and we don't collect them for review and assessment. However, they are an important part of the course, and whenever you encounter an exercise you should stop and do it immediately.

The answers to the exercises are at the end of the module, so once you've completed your solution to the exercise you should jump down to the bottom and check your work. You'll find that the Document Outline window makes this very simple, even for a long document.

So let's do our first exercise!

Write R markdown to display:

- Italic text.
- Bold text.
- Verbatim text.
- Block quote text.

Finally, create a bullet list consisting of 3 things that you like.

When you've finished this, use the Document Outline window to jump to the bottom of the file to see how I did it.

dark-teal-coder

dark-teal-coder

`dark-teal-coder`

> dark-teal-coder

3 Things I Like:

- Mathematics
  - Calculus
- Programming
  - Python
  - Java
- YouTube

Section 4: Code Chunks

> **Main Idea:** *RStudio uses code chunks to pass expressions to the R computational engine for evaluation.*

In this section, we'll learn about code chunks.

Defining code chunks

We saw previously that RStudio itself cannot execute R code. Instead, when RStudio encounters R code, it passes these statements to the R computational engine, which does the actual computing. When R performs these computations, it generates output, which is then passed back to RStudio. Finally, RStudio displays the output to the user.

This is why we can say that RStudio doesn't "know" anything about R.

But how does R know that some text is supposed to represent R code?

RStudio uses the concept of "code chunks" to indicate when text should be considered as R code.

Code chunks are indicated by two delimiters:

- The start of the code chunk consists of 3 backticks, followed by "{r}".
- The end of the code chunk consists of just 3 backticks.

Here's a simple code chunk:

```r
1 + 1
```

```
## [1] 2
```

Let's review what happens with this code:

- First, RStudio sees the start delimiter for a code chunk, so it takes all the text before the end delimiter and passes this directly to R.
- R then performs the computation, in this case adding 1 and 1, and returns the value 2 to RStudio.
- RStudio then displays the value of 2.

Creating Code Chunks

There are 3 ways to create a code chunk in an R markdown document.

1. The first way is to use the "Insert a new R chunk" icon in the upper right-hand corner of the window.
2. The second way is to use a keystroke combination:
   - On Windows, use Ctrl-Alt-I
   - On Macs, use Cmd + Option + I
3. You can even type in the characters for the delimiters by hand.

In general, I find that using keystroke combinations is the most useful of these three methods, and I encourage you to incorporate these into your workflow.

Running code chunks

Once you've created a code chunk, there are 3 ways to execute the code inside of it.

1. First, there is a menu in the upper right-hand corner of the window for running code chunks, and one of the entries is "Run Current Chunk". To use this, place the cursor inside the code chunk (that's what the "current chunk" is), and then execute the "Run Current Chunk" menu item.
2. The second method is to place the cursor inside the code chunk, and then use a keystroke combination:
   - On Windows, use Ctrl + Shift + Enter or Ctrl + Alt + C
   - On Macs, use Cmd + Shift + Enter or Cmd + Option + C
3. The third method is to notice that in the upper right-hand corner of the code chunk, there is a tiny rectangle pointing to the right, and clicking on that will execute the code in the chunk.

Try out these methods for yourself with this code chunk:

```r
4 + 7
```

```
## [1] 11
```

Another useful keystroke to know is "Run All Chunks":

- In Windows, this is Ctrl + Alt + R
- On Macs, this is Cmd + Option + R

The Run menu at the upper right-hand corner of the window has a number of special options for running code chunks, and it's worth checking out.

Again, I strongly recommend learning to use keystrokes, and you'll find that using Create Code Chunk, Run Current Chunk, and Run All Chunks will improve your productivity.

```r
1 + 1
```

```
## [1] 2
```

Knitting and Code Chunks

When you knit an R markdown document to a PDF, RStudio starts at the top of the document and sweeps down, evaluating each code chunk in turn.

If a code chunk generates an error, RStudio will halt the knitting process. Thus, in order to knit to a PDF, you can't have any code chunks that generate errors. That means that you can't have things like incorrect syntax or attempting to perform an illegal operation.

This is important for CSCI S-5a, because you have to submit a knitted version of the problem set for grading, so if your code is really broken then you won't be able to even generate the document.

This sounds scary; but in fact it's very rare that this happens. However, you should be aware of this issue.

So that's how to create and execute code chunks.

Now let's explore some practical issues with LaTeX and pandoc.

Section 5: LaTeX and pandoc

> **Main Idea:** You should be aware of practical issues with LaTeX and pandoc.

In this section, we're going to talk about some practical issues with LaTeX and pandoc.

LaTeX

What's up with this??!?

$$x \ = \ \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

This doesn't look like anything that we've discussed so far.

In fact, it isn't R code at all, and if you try to put this into a code chunk and execute it you'll get an error.

Instead, this is code in the LaTeX typesetting language, which is used for formatting and displaying mathematical expressions.

Do you remember how code chunks use three backticks as delimiters?

LaTeX does the same thing, except it uses dollar signs as the delimiters.

You can see in this example that double dollar signs are used to display the mathematics as an equation block, that is, with all the math in a separate dedicated display.

LaTeX uses single dollar signs to display *in-line* expressions, such as $x = 2$.

LaTeX is a powerful system, and if you're interested in developing your skills in statistics, data science, or machine learning, it can be a very useful tool to know.

However, for CSCI S-5a, you don't need to understand LaTeX at all. In fact, the official course policy on LaTeX is to leave it alone when you encounter it.

LaTeX is very sensitive to small deviations from the official syntax, so it's easy to break LaTeX code by typing in something that seems innocuous.

I only use LaTeX occasionally, when I need to display a mathematical expression, and it's not something that you are expected to learn in the course.

Again, it's a great system to learn, but you don't need it for CSCI S-5a, and in general you should leave it alone when you encounter it.

Remember – LaTeX uses dollar signs as delimiters, so if you see some weird code that doesn't look like R you should search for the dollar signs, and if you find them then that's an indication that you're dealing with LaTeX code.

pandoc

Another possible source of confusion is a system known as "pandoc".

Pandoc is a software tool that can convert document formats, and RStudio uses it in the knitting process.

Occasionally, you'll get random pandoc errors when attempting to knit to a PDF, and they often don't seem to make much sense. Sometimes these pandoc errors are the result of something incorrect in your file, such as broken LaTeX, and you resolve these by figuring out what you did wrong and correcting it. But often, pandoc will generate errors even when you're pretty sure that there is no problem with the R notebook. For instance, it can happen that you knit a document, and then knit it immediately again without making any changes, yet pandoc generates an error the second time.

Why does pandoc do this?

I have no idea. It's just pandoc being pandoc. I also cannot find any documentation on the specific pandoc errors.

For instance, sometimes when I try to knit an R notebook, I get the error message "pandoc error 99".

What is pandoc error 99? I've tried to find out, with little success. It would be nice if there were some sort of reference explaining the numeric error codes, but I've never been able to discover this.

The simple answer when you get a pandoc error is – try again. Sometimes it will work properly the second time around. But usually, once pandoc starts generating errors, it won't revert to normal. In that case, you should save all your work and completely exit from RStudio. Then start up RStudio again, and usually this will resolve the problem.

Remember, sometimes a pandoc error is the result of a genuine problem in the R notebook, and to resolve that you really do have to go in and fix the issue.

But often, pandoc will generate an error even when the original notebook is fine, and for that the best remedy is usually to just close down RStudio and start it up again.

So those are some practical issues when working with LaTeX and pandoc.

Now let's review what we've learned in this module.

Module 4 Review

In this module, we explored how to work with RStudio and R notebooks.

- In Section 1, we discussed what R and RStudio are, and how they work together.
- In Section 2, we learned some of the basics of working with the RStudio interface.
- In Section 3, we explored the R markdown system of formatting text and structuring documents.
- In Section 4, we learned about code chunks.
- In Section 5, we learned about LaTeX and pandoc.

Now that you've completed this module, you should be able to:

- Explain what R is, what RStudio is, and the relationship between the two.
- Use R markdown to perform simple text formatting, including bold, italic, verbatim, block quotes, lists and sublists, and basic tables.

- Use R markdown to specify the logical structure of an R notebook.
- Use the document outline feature of RStudio to efficiently navigate through the logical structure of an R notebook.
- Explain what LaTeX code is, and what to do with it.
- Explain what a code chunk is, and how it allows RStudio to pass expressions to R for evaluation, as well as to display any results.
- Explain how to insert a code chunk into an R notebook.
- Explain how to control the execution of code chunks within an R notebook.
- Explain what LaTeX is, and what to do with LaTeX code.
- Explain strategies for dealing with pandoc errors.

So that's how to work with RStudio and R notebooks.

Now let's move on to Module 5, where we'll fill out Problem Set 0 together!

Solutions to the Exercise

Exercise 1: Using R markdown

Write R markdown to display:

- Italic text.
- Bold text.
- Verbatim text.
- Block quote text.

Finally, create a bullet list consisting of 3 things that you like.

Solution

Here is some *italic* text.

Here is some **bold** text.

Here is some `verbatim` text.

> Here is some block quote text.

Here are three things that I like:

- Jazz
- Mathematics
- Chess