



Literate Programming with RMarkdown

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

June 15, 2017

CC BY-NC-SA 4.0, 2016 - 2017, James J Balamuta

Announcements

- HW1 released later tonight
- **Due Sunday, June 25th, 2017 @ 11:59 PM CDT**
- [Week 1 lecture feedback survey](#)
- Your chance to provide feedback and shape the course!

On the Agenda

1 Literate Programming

- Background
- Markdown
- R + Markdown

2 RMarkdown Structure

- YAML Header
- Code Chunks
- Writing Content

3 Advanced Features

- Global Options
- Caches
- Code Externalization
- Reusing Code
- Bibliographies
- Tables
- Code Engines

On the Agenda

1 Literate Programming

- Background
- Markdown
- R + Markdown

2 RMarkdown Structure

- YAML Header
- Code Chunks
- Writing Content

3 Advanced Features

- Global Options
- Caches
- Code Externalization
- Reusing Code
- Bibliographies
- Tables
- Code Engines

Motivation

“Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.”

— Donald Knuth in *Literate Programming* (1984) on pg. 1

Literate Programming

Definition: Literate Programming is the notion of encouraging *programmers* to interleaving code within narrative content that follows the natural logic and flow of human thought.

In essence, the object is to write an essay that serves as a means to provide documentation for the program simultaneously.

Literate Programming in R

The primary tools available to perform literate programming in *R* are:

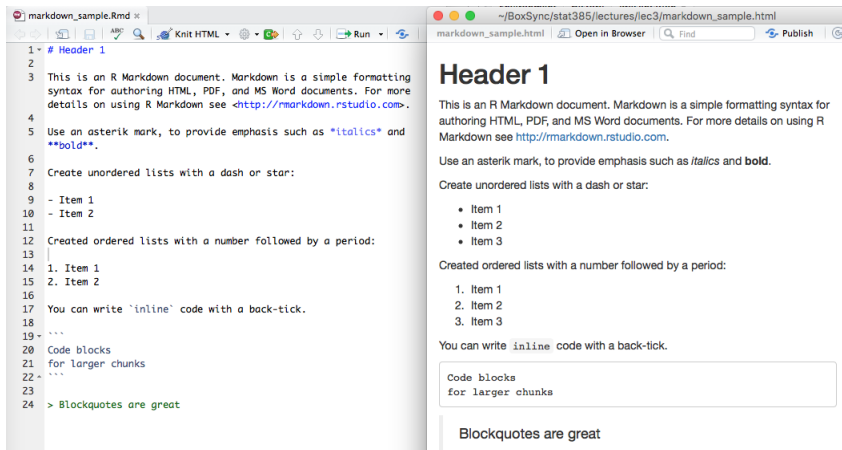
- `markdown`,
- `rmarkdown`, and
- `knitr`

We will be exploring these tools in depth within this lecture.

Markdown

- **Markdown** allows you to write a file format independent document using an easy-to-read and easy-to-write plain text format.
- Instead of **marking *up* text** so that is easy for a computer to read...
 - e.g. HTML: `<html><body>Name</body></html>`
- The goal is to **mark *down* text** so that is human readable:
 - e.g. `**Name**`

Example Markdown Document



The screenshot displays two side-by-side windows. The left window, titled 'markdown_sample.Rmd', shows the source R Markdown code. The right window, titled 'markdown_sample.html', shows the rendered HTML output.

Source Code (Left Window):

```
1 # Header 1
2
3 This is an R Markdown document. Markdown is a simple formatting
4 syntax for authoring HTML, PDF, and MS Word documents. For more
5 details on using R Markdown see <http://rmarkdown.rstudio.com>.
6
7 Use an asterik mark, to provide emphasis such as *italics* and
8 **bold**.
9
10 Create unordered lists with a dash or star:
11
12 - Item 1
13 - Item 2
14
15 Created ordered lists with a number followed by a period:
16
17 1. Item 1
18 2. Item 2
19
20 You can write `inline` code with a back-tick.
21
22 ```
23 Code blocks
24 for larger chunks
25 ```
26
27 > Blockquotes are great
```

Rendered Output (Right Window):

Header 1

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

Use an asterik mark, to provide emphasis such as *italics* and **bold**.

Create unordered lists with a dash or star:

- Item 1
- Item 2
- Item 3

Created ordered lists with a number followed by a period:

1. Item 1
2. Item 2
3. Item 3

You can write `inline` code with a back-tick.

```
Code blocks
for larger chunks
```

Blockquotes are great

Popular Use Cases of Markdown

You may not realize it, but if you ...

- wrote a README.md on [GitHub](#)
- asked or answered a question on [StackOverflow](#)
- created a thread or wrote a comment on [Reddit](#)

... , then you *used* Markdown!

Supported Output Files

As a result of Markdown being structured so loosely, any file format can be generated using [pandoc](#).

That is to say, from one Markdown document you can generate any of the following:

- docx
- PDF
- HTML
- ODT
- RTF

The downside is that there is less control over formatting. This translates over to being unable to effectively set new pages or specify color.

RMarkdown

RMarkdown developed by RStudio takes what Markdown has established and extends it significantly by:

- Allowing *R* code and its results to be merged with Markdown;
- Ensuring that RMarkdown documents are fully reproducible;
- Enabling extra modifications to original markdown specification.

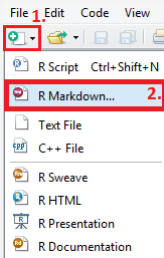
Why learn RMarkdown?

- It's a highly flexible format that allows for easy report generation.
- Provides a way to perform *literate programming*.
- There are countless applications that grow each day:
 - `blogdown`: Create a blog
 - `bookdown`: Write a book
 - `rticles`: Write a LaTeX journal articles
 - `flexdashboard`: Interactive dashboards
 - `uiucthemes`: The theme of this slide deck!

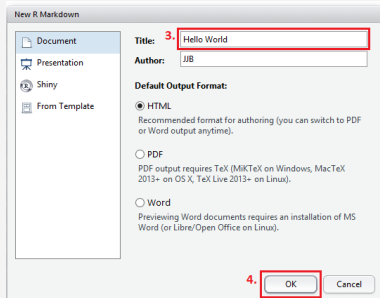
Creating an RMarkdown Document

To create an RMarkdown or .Rmd Document within RStudio:

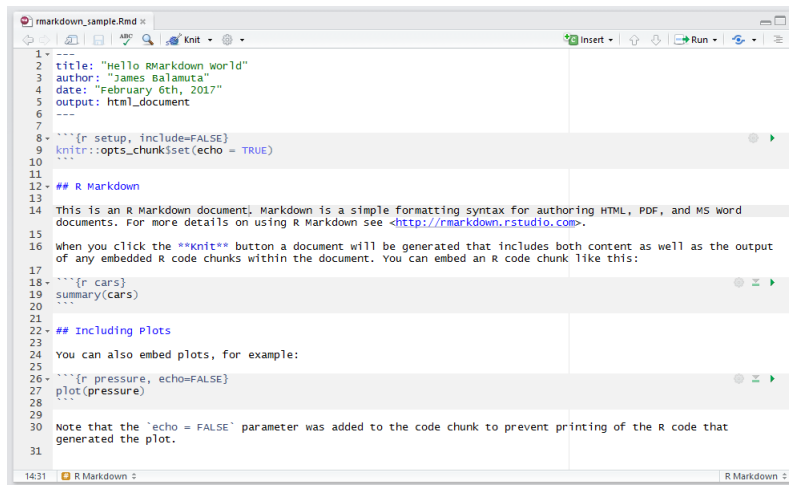
Click the White Plus
Select 'R Markdown'



Enter Document Title



Initial RMarkdown View



```
1 ---
2 title: "Hello RMarkdown world"
3 author: "James Balamuta"
4 date: "February 6th, 2017"
5 output: html_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS word
15 documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
16
17 When you click the Knit button a document will be generated that includes both content as well as the output
18 of any embedded R code chunks within the document. You can embed an R code chunk like this:
19
20 ```{r cars}
21 summary(cars)
22 ```
23
24 ## Including Plots
25
26 You can also embed plots, for example:
27
28 ```{r pressure, echo=FALSE}
29 plot(pressure)
30 ```
31
32 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that
33 generated the plot.
```

Compiling RMarkdown Document

Use either:

- An RStudio shortcut
 - Windows: `Ctrl+Shift+K`
 - macOS: `Command+Shift+K`
- The “Knit ..” button on the source editor window



- Or, compile the document via `rmarkdown::render()`
 - more on this later...

Sample Render of Default RMarkdown View

~/GitHub/stat355p17/lectures/lec5/support/rmarkdown_sample.html
rmarkdown_sample.html Open in Browser Find Publish

Hello RMarkdown World

James Balamuta
February 6th, 2017

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

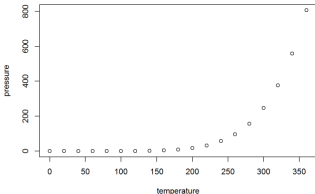
When you click the Knit button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
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
```

Including Plots

You can also embed plots, for example:



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

On the Agenda

1 Literate Programming

- Background
- Markdown
- R + Markdown

2 RMarkdown Structure

- YAML Header
- Code Chunks
- Writing Content

3 Advanced Features

- Global Options
- Caches
- Code Externalization
- Reusing Code
- Bibliographies
- Tables
- Code Engines

Sections of an RMarkdown Document

There are principally *three* sections to an RMarkdown document.

- YAML header
- Code chunks
- Copious amounts of text!

YAML Header and the RMarkdown Document Properties

- *YAML*

- stands for *YAML Ain't Markup Language*, which is a recursive acronym!
 - provides standardized syntax for storing data configurations.
 - properties are set via `key: value`
- YAML is used to specify RMarkdown document properties like:
 - Name of the document
 - Date
 - Output format (pdf, docx, et cetera)
 - And many more.

Example YAML Document Header

```
---
title: "Hello RMarkdown World" # Title of the document
author: "James Balamuta"       # Set the author
date: "June 15th, 2017"        # Set today's date
output:
  html_document:               # Custom format options
    toc: true                  # Display a table of contents
---
```

Once upon a time...

Outputting multiple files from a single RMarkdown

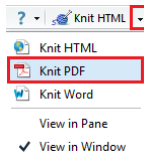
```
---  
title: "Sample Document"  
author: "James Balamuta"  
date: "June 15th, 2017"  
output:  
  html_document:  
    toc: true           # Table of contents  
    theme: cerulean     # Bootstrap theme  
  pdf_document:  
    toc: true           # Table of contents  
    keep_tex: true      # Retain .tex file used for .pdf  
  word_document:  
    fig_width: 5        # Set figure width  
    fig_height: 5       # Set figure height  
    fig_caption: true   # Output captions with figures  
---
```

Compiling .Rmd to .*

Compile via RStudio to .html document:



Switching to a different output format (PDF¹ in this case):



¹Requires a LaTeX distribution installed on ones system.

Compiling .Rmd to .* via R CLI

Single line output declarations

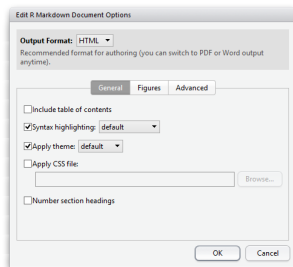
```
library("knitr")  
knit2html("input.Rmd") # Creates .html file  
knit2pdf("input.Rmd")  # Creates .pdf file  
knit("input.Rmd")      # Creates .md file
```

Or use for multiline output declarations

```
library("rmarkdown")  
render("input.Rmd", "pdf_document")  
render("input.Rmd", "word_document")  
render("input.Rmd", "md_document")
```


Options... Options... Options...

Some of RMarkdown's output options can be configured via a GUI in RStudio²



²To see all the options granted by RMarkdown, check out the package website at: <http://rmarkdown.rstudio.com/>.

Dynamic Code Chunks

To initiate a code chunk within RMarkdown, all one needs to do is use:

```
```{r chunk_label}  
Code here
```
```

Example:

Here we will embed the output of obtaining 10 random numbers.

```
```{r chunk_demo}  
x = runif(10)
print(x)
```
```

- What may be slightly problematic about a discussion that follows 10 random numbers?

Code Chunks Options

Each code chunk can receive further customization via chunk options. The values supplied by the code chunk options must be valid *R* expressions (e.g. TRUE or “10px”).

```
```{r chunk_label, chunk_opt1 = value1, chunk_opt2 = value2}  
Code here
```
```

As an example, we can prevent a code chunk from running by setting the chunk option `eval = FALSE`.

```
```{r chunk_label, eval = FALSE}  
x = letters[1:10]
print(x)
```
```

Top Code Chunk Options³

| Chunk Option | Description |
|-----------------------------------|-----------------------------------|
| <code>cache = TRUE</code> | Stores to file code chunk results |
| <code>eval = FALSE</code> | Doesn't evaluate/run the code |
| <code>eval = 2:3</code> | Evaluate only lines 2 and 3 |
| <code>echo = FALSE</code> | Doesn't display the code |
| <code>echo = -c(1,3,4)</code> | Hide only lines 1, 3 and 4 |
| <code>results = 'hide'</code> | Doesn't display code output |
| <code>include = FALSE</code> | Doesn't display code or output |
| <code>fig.path = 'img/'</code> | File path to store images |
| <code>fig.align = 'center'</code> | Align image in the center |
| <code>fig.width = #</code> | Sets width of figure |
| <code>fig.height = #</code> | Sets height of figure |
| <code>warning = FALSE</code> | Hides R's warning messages |
| <code>message = FALSE</code> | Hides R's note messages |

³More options: <http://yihui.name/knitr/options>

In-line R code for RMarkdown

RMarkdown also has the ability to microembed code by using an inline operator that refers to an R expression:

```
`r expression_here`
```

Example:

Did you know that there are ``r nrow(apples)`` observations and ``r ncol(apples)`` variables contained within the apples data set?

A couple of remarks:

- All inline commands must be on the same line (no returns)!
- Helpful to have code chunk hidden before paragraph and use inline features to control output.

Examples of Markdown syntax

Writing text with emphasis in **italics**, ****bold**** and ``code style``.

Line breaks create a new paragraph.

Links can be hidden e.g. `[illinois](www.illinois.edu)` or not `<http://illinois.edu>` .

Sample Image:

```
![Illinois](img/wordmark_vertical.png)
```

Rendered Example of Markdown syntax

Writing text with emphasis in *italics*, **bold** and code style.

Line breaks create a new paragraph.

Links can be hidden e.g. [illinois](#) or not <http://illinois.edu> .

Sample Image:



Examples of Markdown syntax (Con't)

```
> "Never gonna give you up, never gonna let you down..."
```

```
>
```

```
> --- Rick Astley
```

```
- generic
```

```
- bullet point
```

```
- listing
```

```
1. Ordered number list
```

```
1. is numbered
```

```
42. correctly
```

```
Inline math  $a^2 + b^2 = c^2$ 
```

```
Display math (centered math)  $1 - x = y$ 
```


Rendered Example of Markdown syntax (Con't)

“Never gonna give you up, never gonna let you down. . .”
— Rick Astley

- generic
- bullet point
- listing

- 1 Ordered number list
- 2 is numbered
- 3 correctly

Inline math $a^2 + b^2 = c^2$

Display math (centered math)

$$1 - x = y$$

Example of a Table Style in Markdown

There are many [variants of tables](#) available for Markdown. The `pipe_tables` is the preferred tabular format.

```
Left	Center	Right
Hey, check it out	Colons provide	873
its Markdown	alignment thus	1000
right in the table	*centered* text	
```

Renders as:

| Left | Center | Right |
|---------------------|----------------------|-------|
| Hey, check it out | Colons provide | 873 |
| its Markdown | alignment thus | 1000 |
| right in the table | <i>centered</i> text | |

On the Agenda

1 Literate Programming

- Background
- Markdown
- R + Markdown

2 RMarkdown Structure

- YAML Header
- Code Chunks
- Writing Content

3 Advanced Features

- Global Options
- Caches
- Code Externalization
- Reusing Code
- Bibliographies
- Tables
- Code Engines

Setting Global Chunk Options

Instead of declaring options repetitively across multiple code chunks, it is better to create a global declaration in a “setup” chunk at the start of the document.

Even with a global declaration, values are able to be changed locally on code chunks on an as needed basis. The local values will not affect future code chunks.

To set global chunk settings use:

```
library("knitr") # Make sure this is loaded!

opts_chunk$set(eval = FALSE, comment = NA,
                fig.width = 6, fig.height = 6,
                fig.align = 'center')
```

Caching

Caching refers to storing data locally in order to speed up subsequent retrievals.

In essence, the code chunk will be run once, the resulting objects are then stored within a file, and the stored data is then displayed on subsequent runs.

This feature is very handy when embedding analysis on large data sets or time intensive computations.

Be Careful When Caching...

Simple yet problematic cache:

```
{r cache_demo, cache = TRUE}  
x = rnorm(5)  

```

```
## [1]  1.3887859  0.7123659 -1.4047637  0.6126488  
## [5]  0.9406619
```

The Problematic Cache

Suppose you have three code chunks, say *A*, *B*, and *C*, that each use a random number generation (RNG) routine and have been evaluated with the option `cache = TRUE`.

If the code chunk *C* is inserted between *A* and *B*, then the order updates to *A*, *C*, and *B*. However, the values have been *cached* and a subsequent, re-running of the document will **not** change generated values.

This is problematic as the *order* in which the RNG modifies `.Random.seed` is no longer consistent. As a result, if the cache becomes *invalidated* or deleted, then the reproducibility of the results comes into question.

That is to say: **The reproducibility of *B* and *C* is now bogus.**

Solutions for the Problematic Cache

There have been many solutions put forward to address the problematic caching behavior described above. A few of them are as follows:

- Associate the `.Random.seed` with the cache for each chunk
- If a domino block computation, use `dependson = <chunk-name>` chunk option.
- Cache by Session or File information.

Associating `.Random.seed` with Code Chunks

To guarantee reproducibility with RNG, `.Random.seed` needs to be associated with the cache for each chunk using:

```
# Set global chunk options  
opts_chunk$set(cache.extra = rand_seed)
```

Using dependson

If results depend on a previous chunk's work, then it make sense to apply a dependency via the `dependson='chunk-name'` option.

Origin Chunk

```
```{r chunk_A, cache = TRUE}  
set.seed(12354)
x = runif(10)
```
```

Add-on Dependency

```
```{r chunk_B, cache = TRUE, dependson="chunk_A"}  
y = x + 100
```
```

Global dependson option

To use the global dependson option we need to set the autodep = TRUE global chunk option.⁴

```
library("knitr")  
  
# Set global chunk options  
opts_chunk$set(cache = TRUE, autodep = TRUE)
```

⁴This option is not ideal since if variables share the same name anywhere and are slightly changed, then an update is forced automatically.

The Limiting Cache Scope

Sometimes the cache needs to be limited by R version, Session Info, or a time stamp.

```
# Set global chunk options
opts_chunk$set(cache.extra =
  list(R.version,      # R version info w/ OS
        rand_seed,     # .Random.seed
        sessionInfo(), # Current Configuration
        # Month timestamp
        format(Sys.Date(), '%Y-%m'))
```

Limit Cache by File Info

Limit cache using the file stamp associated with data files being read into R for the analysis

```
##' @param files is a character vector containing filenames  
##' @return time stamp  
mtime = function(files){  
  lapply(Sys.glob(files),function(x) file.info(x)$mtime)  
}
```

```
```{r data_read_in, cache.extra = mtime("apple.csv")}  
data = read.csv("apple.csv")
```
```

Returning to In-line R code for .Rmd

Previously, we executed code inline with an actual function call:

```
`r dim(apples)[1]`
```

In this case, it is highly preferred to move those calls to a code chunk *before* the text as this calculation can then be cached.

Code Chunk Example:

```
```{r data_inline_calc, cache = TRUE}  
data("apples") # Loads the apple data set.
obs = nrow(apples) # Number of observations in the dataset
vars = ncol(apples) # Number of variables in the dataset
```
```

Revisiting Inline Code

In document:

```
Did you know that there are `r obs` observations  
and `r vars` variables contained within the apples  
data set?
```

Compared to the previous approach, we now have:

- Cached computations
- Clearer function calls
- Less obtrusive explanations.

Separation of Concerns

When we first began using `rmarkdown`, the R code was directly embedded into the `.Rmd` file.

This is not a good practice in general.

Why? The elements of the analysis are then merged with elements of the presentation.

If the code is externalized:

- 1 The code can be run without compiling the document.
- 2 Sharing the code with others is easier.
- 3 Switch between presentation slides and the code during the actual presentation.

Code Externalization in Action

To externalize code, the `r` code file must contain comments of the form:

```
## ---- label or ## @knitr label
```

, where `label` denotes a code chunk name.

Example Code File: **analysis_code.R**

```
# Label code as:  
  
## @knitr external_code  
(x = "Rawr?")  
  
# or  
  
## ---- external_code  
(x = "Rawr?")
```

Sample External Code Read In

Example .Rmd File: **analysis_writeup.Rmd**

```
read_chunk('analysis_code.R') # Setups code chunks
```

Activate Code Chunk in Document:

```
```{r external_code, echo = TRUE}  
```
```

Result:

```
(x = "Rawr?")
```

```
## [1] "Rawr?"
```

Reusing Code Chunks

By labeling code chunks, `knitr` is able to call them in the future or embed them within other code chunks.

To reuse the code from `<<chunk-name>>`, call it from within:

```
`r chunk_embedded_var`.
```

Reusing Code Chunks Example

Initial chunk:

```
```{r chunk-name}  
test = "hello"
```
```

Embedded Chunk:

```
```{r chunk-embedded}  
<<chunk-name>>
```
```

Result:

```
"hello"
```

```
## [1] "hello"
```

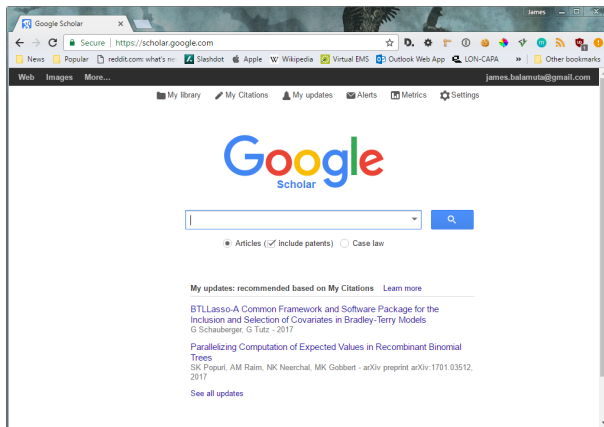
BibTex

In RStudio, we are able to use BibTex to generate bibliographies.

BibTex is a way to structure output of paper citations you can obtain from [Google Scholar](#)

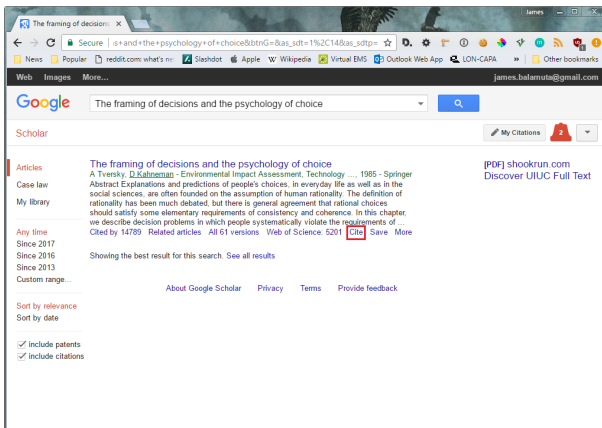
Google Scholar Search Portal

Just like with regular ol' Google, type in to the search bar either the full paper names or just key terms that you are interested in.



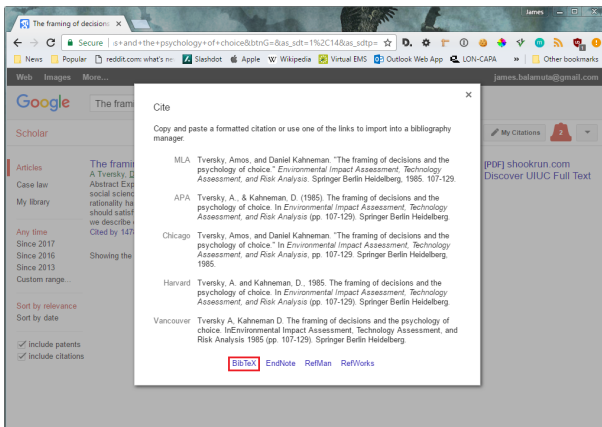
Obtaining a citation from Google Scholar

Once you found the paper you were looking for, press the Cite button.



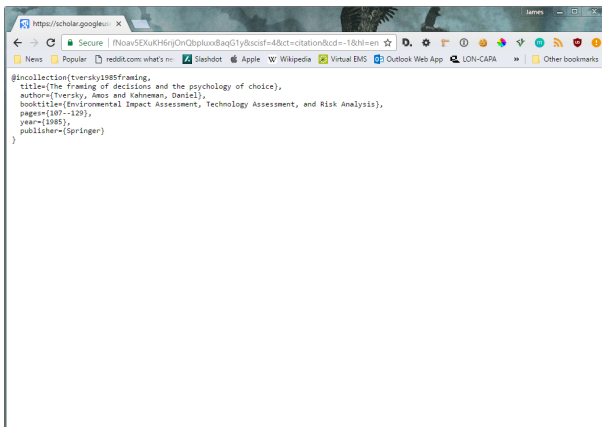
Navigating Citation Options

From the citation menu, there are many different export options. In our case, we'll opt for the BibTeX option.



Copying the BibTeX Citation.

From here, use either CNTRL + C (Windows/Linux) or CMD + C (macOS) to copy the citation.



Structuring a .bib file

Bibliography information is stored in a .bib file, which is just a regular text file.

Here is the mybiblib.bib with the citation used in the previous example:

```
@incollection{tversky1985framing,  
  title={The framing of decisions and the psychology of choice},  
  author={Tversky, Amos and Kahneman, Daniel},  
  booktitle={Environmental Impact Assessment, Technology Assessment,  
            and Risk Analysis},  
  pages={107--129},  
  year={1985},  
  publisher={Springer}  
}# Additional entries...  
@book{ho1987,  
  title={Rational choice: Contrast between economics and psychology.},  
  author={Hogarth, Robin M and Reder, Melvin W},  
  year={1987},  
  publisher={University of Chicago Press}  
}
```

YAML for including a bibliography

```
---  
title: "Example Biblography heading"  
output: html_document  
bibliography: bibliography.bib  
---
```

Reference in the document using:

- Authors + Year e.g. `[@tversky1985framing]`
 - “Choice is important (Tversky and Kahneman 1985)”.
- Suppress author information and only have a year e.g. `[-@tversky1985framing]`
 - Tversky says, “Choice is important (1985)”.

knitr's `kable()`

The main benefit to `kable()` is not having to worry about the mode `knitr` is currently in (e.g. `html`, `latex`, or `markdown`) and its ability to split tables over pages.⁵

⁵Make sure to set the chunk option `results = 'asis'`.

Demo of kable()

```
kable(head(iris,3),           # Data to be converted to table
      row.names = FALSE,     # Show rownames
      align = c('l', 'c', 'r', 'l', 'r'), # Sets column alignment
      digits = 1             # Restrict amount of digits after decimal
    )
```

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|--------------|-------------|--------------|-------------|---------|
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |

Different Language Engines

RMarkdown supports a large variety of different languages.⁶ This allows for you to embed code in an RMarkdown document and receive results from the desired language. Some supported languages include:

- Rcpp, bash, python, sas, sql, c, fortran, and stata.

To do so, simply replace the `r` at the start of the chunk with the correct engine name.⁷

```
```{bash chunk_engine_demo}  
We're in bash now!
echo $HOME
```
```

⁶Assuming each language is installed and available on your system's PATH variable

⁷Unlike R, the results from each code chunk may not persist for a given language engine.