# Generating Reports and Web Apps

http://datascience.tntlab.org

Module 10

# Today's Agenda

- Installing software to use Markdown on your own machine
- Walkthrough of Markdown and markup languages more generally
  - A bit of a preview of next week
- In-depth on *shiny*
  - How to create your own *shiny* apps
  - How to enable other people to access your *shiny* apps

# Necessary Installs Beyond *R*

- R Packages (not actually automatic)
  - *rmarkdown*
  - *shiny*
  - *knitr*
  - *ggvis*

- Software for Exports
  - PDF: LaTeX (http://latex-project.org/ftp.html; get MiKTeX [Win] or MacTeX [Mac]) and set it to automatically install packages during installation
  - HTML: pandoc v1.19.2.1 (https://github.com/jgm/pandoc/releases/; for now)
  - DOCX: MS Word

- Although you could theoretically use RMarkdown on the command line/directly to generate reports, we will only use R Studio.

# R Markdown utilizes a Markup Language

- Markup languages
  - Plain text document annotation systems intended to provide text meta-data directly on a manuscript
  - The term "markup" comes from traditional marking up by hand of documents

- Examples of markup languages
  - HTML
  - LaTeX
  - Markdown

- Advantages vs. disadvantages (comparing WYSIWYG vs. WYSIWYM)
  - Utilize plain text, so they are universally accessible and will not become obsolete
  - Utilize plain text, so they are hard to read and require computer interpretation

# Example of HTML Markup

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
</head>

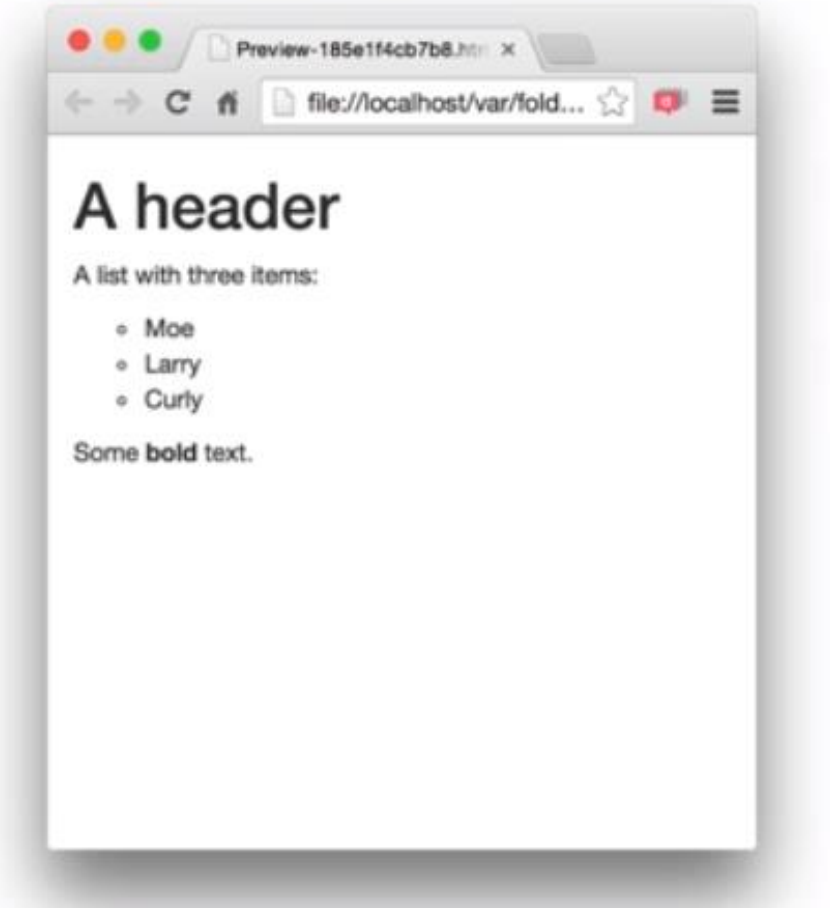
<body>

  <h1>A header</h1>

  <p>A list with three items:</p>
  <ul style="list-style-type:circle">
    <li>Moe</li>
    <li>Larry</li>
    <li>Curly</li>
  </ul>

  <p>Some <strong>bold</strong> text.</p>

</body>
```



# Example of Markdown Markup

- (from Wikipedia)

| Text using Markdown syntax | Corresponding HTML produced by a Markdown processor | Text viewed in a browser |
|---|---|---|

```
# Heading

## Sub-heading

### Another deeper heading

Paragraphs are separated
by a blank line.

Two spaces at the end of a line leave a
line break.

Text attributes _italic_, *italic*, __bold__, **bold**,
`monospace`.

Horizontal rule:

---

Bullet list:

  * apples
  * oranges
  * pears

Numbered list:

  1. apples
  2. oranges
  3. pears

A [link](http://example.com).
```

```
<h1>Heading</h1>

<h2>Sub-heading</h2>

<h3>Another deeper heading</h3>

<p>Paragraphs are separated
by a blank line.</p>

<p>Two spaces at the end of a line leave a<br />
line break.</p>

<p>Text attributes <em>italic</em>, <em>italic</em>,
<strong>bold</strong>, <strong>bold</strong>,
<code>monospace</code>.</p>

<p>Horizontal rule:</p>

<hr />

<p>Bullet list:</p>

<ul>
<li>apples</li>
<li>oranges</li>
<li>pears</li>
</ul>

<p>Numbered list:</p>

<ol>
<li>apples</li>
<li>oranges</li>
<li>pears</li>
</ol>

<p>A <a href="http://example.com">link</a>.</p>
```

Heading

**Sub-heading**

**Another deeper heading**

Paragraphs are separated by a blank line.

Two spaces at the end of a line leave a
line break.

Text attributes *italic*, *italic*, **bold**, **bold**, `monospace`.

Horizontal rule:

---

Bullet list:

- apples
- oranges
- pears

Numbered list:

1. apples
2. oranges
3. pears

A link.

# Markdown Signals

- See **Help/Markdown Quick Reference** in R Studio
- Or use this cheat sheet: https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf
- Or use this official guide:
- http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
superscript^2^
~~strikethrough~~
[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi*r^{2}$
image: ![](path/to/smallorb.png)

horizontal rule (or slide break):

***

> block quote

* unordered list
* item 2
    + sub-item 1
    + sub-item 2

1. ordered list
2. item 2
    + sub-item 1
    + sub-item 2

Table Header  | Second Header
------------- | -------------
Table Cell    | Cell 2
Cell 3        | Cell 4
```

Plain text
End a line with two spaces to start a new paragraph.
*italics* and *italics*
**bold** and **bold**
superscript$^2$
~~strikethrough~~
link

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: –
emdash: —
ellipsis: …
inline equation: $A = \pi * r^2$
image: 
horizontal rule (or slide break):

> block quote

- unordered list
- item 2
  - sub-item 1
  - sub-item 2

1. ordered list
2. item 2
   - sub-item 1
   - sub-item 2

| Table Header | Second Header |
|---|---|
| Table Cell | Cell 2 |
| Cell 3 | Cell 4 |

# Embedding Code

- `` `r codehere` `` to execute that code wherever you want to execute it.

- To execute code as a block (and display it as such):
  ```` ```{r} ````
  ```` ```codehere ```` 

- Options for the {r} call, separated by commas (also see cheat sheet):
  - **name**: name the code
  - **echo=TRUE**: display the code (does not affect results or messages)
  - **message=TRUE, warning=TRUE, error=TRUE:** display messages, warnings, and errors
  - **eval=TRUE**: run the code
  - **results='markup'**: other options: "asis", "hold", "hide"
  - Full list here: https://yihui.name/knitr/options/

- Example
  - ```` ```{r mycode, echo=F} ````
    ```` ```myvar <- 1 ````
    ````

# Basic YAML Headers

- Same format every time

```
---
title: "My Title"
author: "My Name"
date: "Today"
output: output_type
---
```

- *output_type* specifies what you're going to generate
  - html_document
  - pdf_document
  - word_document
  - beamer_presentation # PDF slideshow
  - slidy_presentation # HTML slideshow using slidy template
  - ioslides_presentation # HTML slideshow using ioslides template
  - md_document # a .rmd markdown file – so when is this useful?

# Extended YAML Options

- Indention and tabs matter, which do not follow clear patterns except re:output options
- Output options vary by output format, but HTML is by far the most flexible
- Again, see http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

```
---
title: "My Title"
author: "My Name"
date: "Today"
output:
        html_document:
                theme: cosmo
                number_sections: true
                toc: true
                toc_float: true
---
```

- Most flexibility with display comes from creating CSS files

# CSS: Cascading Style Sheets

- HTML is a markup language intended to provide webpage content
- CSS is a style language intended to provide stylistic/formatting data for markup

- HTML+CSS is the backbone of the web
  - We will talk about CSS more later when we get into web scraping but will focus more on how-to-read than how-to-write

- If you want to learn to write CSS, I suggest Codecademy
  - https://www.codecademy.com/learn/learn-css

# HTML + CSS

```
<html>
<head>
<title>My Webpage</title>
<link href="mycss.css"
type="text/css" rel="stylesheet">
</head>
<body>
<h1>My Wepage</h1>
<p>This is my webpage.</p>
</body>
</html>
```

```
body {
        background-color: #EBEBEB;
}
h1 {
        font-weight: bold;
        font-size: 24pt;
}
p {
        font-style: italic;
}
```

# Creating PDFs

- Just like you saw in Data Camp

- Things that are different from what we normally do:
  - All files should be in a *markdown* directory together
  - Don't run *rstudioapi* code

# Easiest Way to Create a Shiny App

- The simplest way to write a *shiny* app using Markdown:
  1. Use **write_csv()** on your final ready-for-figure dataset in *markdown* folder
  2. Have your original figure generation code ready
  3. Create a new *shiny* markdown document using R Studio (**not** a new *shiny* app)
  4. Delete everything below the YAML (unless you want an example)
  5. Within a code block (```) add *shiny* functions for anything to be changeable
     - Examples: **numericInput(), radioButtons(), selectInput(), sliderInput()**
  6. Test your *shiny* input code by copying each function's results into a .html file and viewing that file in a web browser to verify it looks right
  7. Use **read_csv()** to import your final ready-for-analysis dataset
  8. Copy existing figure generation code and modify to use *shiny* variables (remember to use **input$** to refer to those variables)
  9. Surround entire figure generation code with **renderPlot({ })**

# *ggplot2* vs *ggvis*

- A summary is available here: http://ggvis.rstudio.com/ggplot2.html

- *ggvis* is a more advanced tidyverse plotting system that differs from *ggplot2* in that *ggvis*:
  - Is more experimental
  - Combines elements using %>% instead of +
  - Does not support faceting
  - Distinguishes between static **geoms** (which are now **emit_**) and dynamic **geoms** (layers)
  - Creates an HTML object when exported (vs. *ggplot2*'s static image file)
  - Enables animation using *shiny* and unique *ggvis* functions
  - Is much faster
  - Has less graphical complexity by default
  - Has fewer emits than *ggplot2* has **geoms**

- You generally want to use *ggvis* to render animations on webpages and either of the two (but probably *ggplot2*) for everything else

# Easiest Way to Put a *Shiny* App Online

- Easiest: https://shinyapps.io

- First time setup only
  - Sign up for a free account (something.shinyapps.io)
  - Follow the instructions on the webpage, shorthanded here
    1. **install.packages("rsconnect")**
    2. **library(rsconnect)**
    3. On the website, click "Show Secret" then "Copy to Clipboard" – paste that into R and run it
    4. **deployApp("nameofappfile.Rmd")**

- After initial setup
  - **library(rsconnect)**
  - **deployApp("nameofappfile.Rmd")**

# Common Problems with Shiny and shinyapps.io

- Forgot to set up access tokens in R Studio
- Forgot **runtime:shiny**
- Didn't put skinny datafile in *markdown* directory
- Didn't define *shiny* input functions correctly, especially related to named vectors
- Put too much re-processing within **renderPlot()**