Introduction to R Markdown

Contents

This document was used as part of a 3-hour tutorial on R Markdown I presented to my colleagues at work. You can find all the materials used in the class on Github including the R Markdown version of this document.

Outline

- What is R Markdown
- Why Use R Markdown
    - Use Cases
- Getting Started
    - Installation

    - Opening an R Markdown Document

- The Ingredients of an R Markdown Document
    - YAML Header
    - Formatted Text
    - Code Chunks
    - Inline Code
- Output
- Miscellaneous Information
- Additional Resources

What is R Markdown

R Markdown is a tool within RStudio that allows you to write documents, presentations, or webpages that **combine written text with analytical code.**

- The text in the document can be fully formatted in a report style (e.g., with headers, bolded text, hyperlinks, etc.)
- Your code runs when the document is created
- You can choose to make your code visible or not
- Documents can be output as PDFs, Word Documents, HTML, and other formats
- In new versions of RStudio, you can intermix R and Python code using the `reticulate` package

Why Use R Markdown

The strongest argument for using R Markdown is **reproducibility**; all of your analyses and text describing those analyses are in one place. In graduate school, I did most of my work in Excel and SPSS. Some of the analyses were scripted but not everything. I would be hard-pressed to perfectly recreate every ANOVA result, every mean, every p-value. Here's an excerpt (I bolded the calculations for emphasis):

> *Certainty.* Participants in the low distraction condition were more certain that their attitudes toward Wi-Fi networks were correct **(M = 5.20, SD = 1.66)** than were high distraction participants **(M = 4.53, SD 1.88), F(1, 141) = 5.11, p < .05**.

With R Markdown, you don't get a final report unless the code runs perfectly throughout the document.

A few more arguments in favor of R Markdown:

- The document will serve as a record for how you arrived at the results you include in your papers
- You can pass on your code to readers in addition to the report content
- Dynamic documents can change as data are updated
- Documents can also be used for future data releases and/or different subsets of data

Use Cases

There are dozens of different use cases listed in the R Markdown Definitive Guide. I highlight a few ideas below applicable to BLS:

- Reproducible Reports
- Situations Where You Have Frequently Updating Data
- Report Templates
  - Subsectors
  - Time Periods
- Customized Respondent Materials
- Tutorials (e.g., on Rpubs - Creating a Network Plot in R)
- Documentation
- Webpages/Websites

Getting Started

Installation

Before you start creating documents you'll need to do a few things first.

1. Download and install R
2. Download and install RStudio
3. Install `rmarkdown` package

```r
install.packages("rmarkdown")
```

4. Install some distribution of the LaTeX system (e.g., MikTex). This is necessary if you want to output to PDF or Word.
   - Rstudio suggests the `tinytex` package as an alternative to LaTeX software like MikTex. I haven't tried it myself.

Opening an R Markdown Document

1. Open RStudio

2. Select File > New File > R Markdown…
3. Enter Title, Author, and Output Format (this can be changed later)
4. Select OK
5. You will get an example that you can alter or delete

Hands-On: Open a blank R Markdown document and save it to a folder on your computer.

The Ingredients of an R Markdown Document

There are four key ingredients to an R Markdown document:

1. YAML Header
2. Formatted Text
3. Code Chunks
4. Inline Text

YAML Header

- YAML stands for "Yet Another Markup Language"
- This is where you set options for your overall document
- YAML can also be used to store document metadata (author, date published, etc.)
- Through the YAML header you can set:
  - font (size and style)
  - default figure options (height, width, etc.)
  - reference custom CSS (Cascading Style Sheets) Code

```
---
title: "My Title"
author: "Brandon Kopp"
date: "July 19, 2019"
output: html_document
css: customcss.css
---
```

Hands-On: Update the title and author in the YAML Header.

Formatted Text

R Markdown offers shorthand for formatting text. This shorthand is called **markdown**. It is important to note that not all markdown will be interpreted the same. What is shown below is fairly common and will display as intended in documents produced in R Markdown, but it may display differently if you load it onto Github.

For any markdown syntax, there is equivalent HTML syntax. You can use either version in R Markdown documents that you intend to output to HTML. HTML syntax will not be interpreted correctly if you output to PDF.

Format	Markdown	HTML	Formatted Text
italics	`*italics*` OR `_italics_`	`<i>italics</i>`	*italics*
bold	`**bold text**` OR `__bold text__`	`<b>bold text</b>`	**bold text**
strikethrough	`~~strikethrough~~`	`<strike>italics</strike>`	~~strikethrough~~
superscript	`superscript^text^`	`superscript<sup>text</sup>`	superscript^{text}

Format	Markdown	HTML	Formatted Text
subscript	`subscript~text~`	`subscript<sub>text</sub>`	subscript$_{\text{text}}$
hyperlink	`[BLS](www.bls.gov)`	`<a href="www.bls.gov">BLS</a>`	BLS
highlight code	`'highlighted code'` Note: ' is a backtick	`<code>highlighted code</code>`	`highlighted code`

Section Headings

Header 1

Markdown: `# Header 1`
HTML: `<h1>Header 1</h1>`

Header 2

Markdown: `## Header 2`
HTML: `<h2>Header 2</h2>`

Header 3

Markdown: `### Header 3`
HTML: `<h3>Header 3</h3>`

Header 4

Markdown: `#### Header 4`
HTML: `<h4>Header 4</h4>`

Lists

Unordered List

Markdown:
```
- Item
______ + Sub-item
- Item
- Item
```

- Item
    - Sub-item
- Item
- Item

Ordered List

Markdown:
```
1. Item Number 1
2. Item Number 2
______ + Sub-item
3. Item Number 3
```

1. Item Number 1
2. Item Number 2
    - Sub-item

3. Item Number 3

Tables

Markdown:
```
|  Header 1  |  Header 2  |
| ---------- | ---------- |
| Row1, Col1 | Row1, Col2 |
| Row2, Col1 | Row2, Col2 |
```

Header 1	Header 2
Row1, Col1	Row1, Col2
Row2, Col1	Row2, Col2

Note: The vertical pipes `|` don't have to line up for the table to display correctly.
Note: There are various options for cell alignment.

Markdown:
```
|  Center-aligned Header 1  |  Right-aligned Header 2  |
| :--------: | ---------: |
| Center-aligned Row1, Col1 | Right-aligned Row1, Col2 |
| Center-aligned Row2, Col1 | Right-aligned Row2, Col2 |
```

Center-aligned Header 1	Right-aligned Header 2
Center-aligned Row1, Col1	Right-aligned Row1, Col2
Center-aligned Row2, Col1	Right-aligned Row2, Col2

LaTeX Equations

Markdown: `$$x_{1,2} = {-b\pm\sqrt{b^2 - 4ac} \over 2a}.$$`

$$x_{1,2} = {-b\pm\sqrt{b^2 - 4ac} \over 2a}$$

Markdown: `$$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$`

$$P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)}$$

Images

Markdown: `![](./img/bls_emblem.png)`
HTML: `<img src="./img/bls_emblem.png">`

Line Breaks

- Simple line breaks can be somewhat confusing in R Markdown. In order to to create a line break, you have to **end a line with two spaces.**
- If you want more than one line between paragraphs, you need to use `<br>` which is HTML code for a manual line break.

For example:

Here are two lines separated by a hard return but no spaces at the end of the first sentence. They end up becoming one line.

You must include two spaces.
This will separate your lines.

Exercise: Copy and paste the unformatted text below into the R Markdown document you created earlier, then add markdown formatting to it so that it looks like the text in the final formatted text section.

```
Fisher's Iris Data
by [ENTER YOUR NAME]

The famous Fisher iris data set gives the measurements in centimeters of the variables sepal length and

iris is a data frame with 150 cases (rows) and 5 variables (columns) named:
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Species

For more information about this data set, see Fisher's Iris Data Set on Wikipedia
[USE THIS LINK]: https://en.wikipedia.org/wiki/Iris_flower_data_set
```

Final Formatted Text

Fisher's Iris Data

by Brandon Kopp

The famous **Fisher iris data set** gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris *setosa*, *versicolor*, and *virginica*.

`iris` is a data frame with 150 cases (rows) and 5 variables (columns) named:

1. Sepal.Length
2. Sepal.Width
3. Petal.Length
4. Petal.Width
5. Species

For more information about this data set, see Fisher's Iris Data Set on Wikipedia.

Code Chunks

Code chunks are where the magic happens in R Markdown. Code chunks can contain any code that you would would use in R. The code you enter gets executed and the results are shown in the document.

The syntax for code chunks looks like this:

```
'''{r}
  SOME CODE GOES HERE
'''
```

NOTE: `'` is a backtick, *not* a single quote or apostrophe.

Let's start with something simple; assigning values to variables and doing a mathematical operation. The box below is an example of how a code chunk appears in a document. By default, you will see the grey box with code highlighting. You will not see the code chunk options.

```
x <- 7
y <- 6
x * y
```

```
## [1] 42
```

Once those values are assigned, we can use them in later code chunks.

```
x + y
```

```
## [1] 13
```

```
sum(c(x, y))
```

```
## [1] 13
```

```
round(x/y, 2)
```

```
## [1] 1.17
```

```
x > y
```

```
## [1] TRUE
```

You can run code chunks individually within R Studio or execute the code line-by-line as you would in a normal document.

Three important points:

1. Code chunks are run in order so you need to keep straight what variables need to be created in which order. This seems obvious, but can be problematic if you run code line-by-line and have variables loaded in the global environment.
2. You don't have to knit the document in order to see the output of the code chunks. You can run the code chunks individually.
3. You shouldn't have any issues with setting a working directory. When running individual lines within an R Markdown document or when you knit a document, it treats the Rmd file's directory as the working directory and you can use relative paths from there.

Using Code Chunks To Make Tables

We can use the code chunks to print tables

```
mycars <- mtcars[1:5,1:6]
mycars
```

```
##                    mpg cyl disp  hp drat    wt
## Mazda RX4         21.0   6  160 110 3.90 2.620
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875
## Datsun 710        22.8   4  108  93 3.85 2.320
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215
## Hornet Sportabout 18.7   8  360 175 3.15 3.440
```

We can also output more nicely formatted tables using `knitr`. There are *many* other packages that offer different formatting options (e.g., `xtable`).

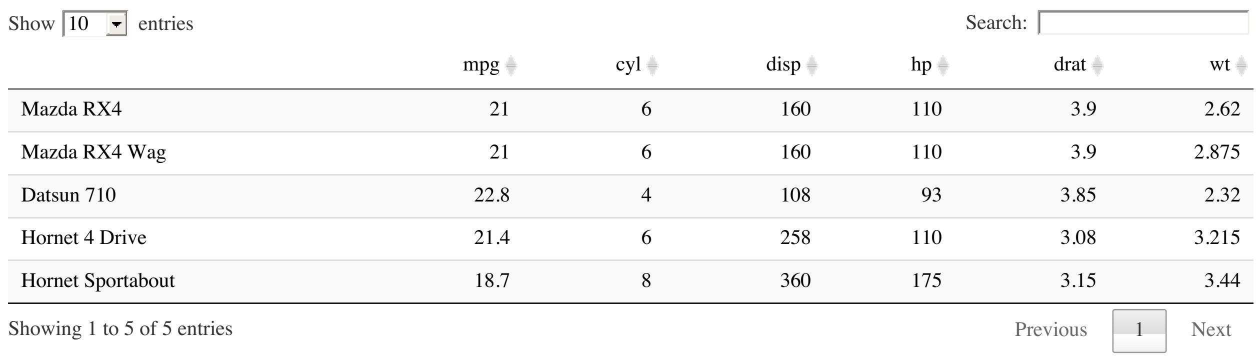
```
knitr::kable(mycars,caption = "Motor Trend Car Table")
```

Table 4: Motor Trend Car Table

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

You can also add interactive elements into HTML documents.

```
  DT::datatable(mycars)
```



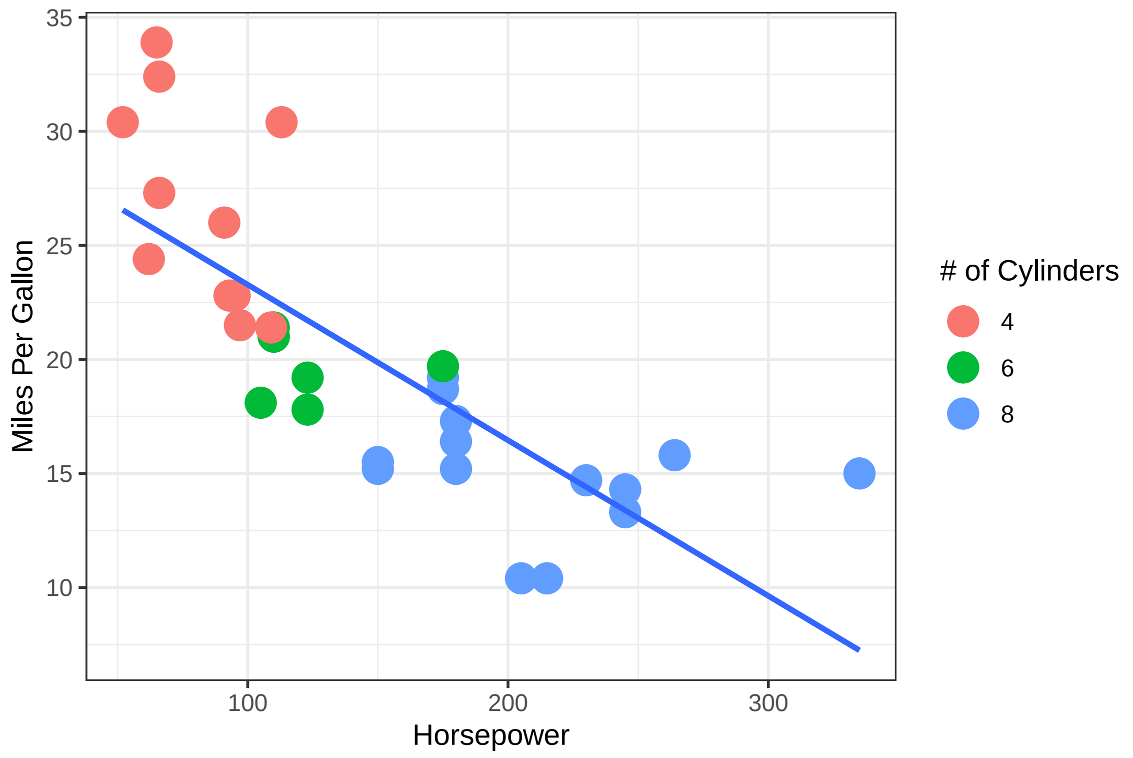
	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21	6	160	110	3.9	2.62
Mazda RX4 Wag	21	6	160	110	3.9	2.875
Datsun 710	22.8	4	108	93	3.85	2.32
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.44

Showing 1 to 5 of 5 entries

Using Code Chunks To Make Figures

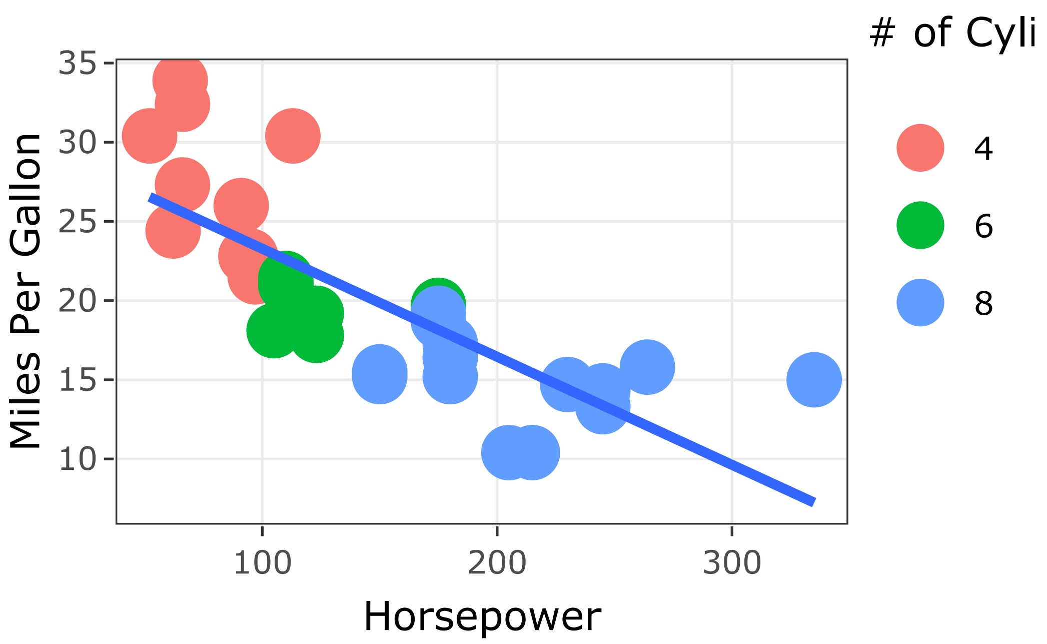
We can also use code chunks to print figures

```
library(ggplot2)
gg <- ggplot(mtcars, aes(hp, mpg)) +
        geom_point(aes(color=as.factor(cyl)), size=5) +
        geom_smooth(method="lm", se=FALSE) +
        labs(x = "Horsepower",y= "Miles Per Gallon",
             color= "# of Cylinders") +
        theme_bw()
gg
```



You can also make interactive graphics in HTML documents.

```
  plotly::ggplotly(gg)
```



Code Chunk Options

Code chunks accept optional arguments

```
'''{r name, eval=FALSE, warning=FALSE, message=FALSE}
  # SOME CODE GOES HERE
'''
```

- **name** - This is not necessary, but it is good practice to label your code chunks. Two figures cannot have the same name
- **echo** - Whether to display the code chunk or just show the results. If you want the code embedded in your document but don't want the reader of the document to see it, you can set echo=FALSE (Default: `echo=True`)
- **eval** - Whether to run the code in the code chunk. 'eval=FALSE' can be used if you want to display the code but not have it run (Default: `eval=TRUE`)
- **warning** - Whether to display warning messages in the document (Default: `warning=TRUE`)
- **message** - Whether to display code messages in the document (Default: `message=TRUE`)
- **results** - Whether and how to display the computation of the results
- **cache** - Saves output of the code chunk for use when you run the R Markdown file the next time (within the same session). This is useful if you have a code chunk that takes a long time to run. (Default: `cache=FALSE`)

In your first code chunk, you can set your own defaults.

```
'''{r include=FALSE}
knitr::opts_chunk$set(echo = FALSE, warning=FALSE, message=FALSE)
'''
```

Code Chunk Examples

Here is a code chunk with `warning = TRUE` and `message = TRUE` (i.e., the defaults for code chunks):

```
library(dplyr)
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

These are things you probably don't want in your report, but luckily you can turn them off.

And with `warning = FALSE` and `message = FALSE`:

```
library(dplyr)
```

Hands-on: Create a new code chunk and use it to load the `dplyr` and `ggplot2` libraries. Knit the document with and without `warning=FALSE` and `message=FALSE`.

Here is a code chunk with `echo = TRUE`:

```
n <- 7*6
n
```

```
## [1] 42
```

Here is the same code chunk with `echo = FALSE`:

```
## [1] 42
```

Here is the same code chunk with `eval = FALSE`. This just shows the code chunk but the code is never run so 'n' is not available for use in later code chunks/inline code.

```
n <- 7*6
n
```

Here is that code chunk again with `results = 'hide'`. Similar to `eval=FALSE`, this just shows the code chunk, however, now 'n' is available for later.

```
n <- 7*6
n
```

Hands-on: Create a new code chunk and use it to perform a calculation that will be used later.

```
iris_df <- iris %>%
  mutate(sepal_area = Sepal.Length * Sepal.Width,
         petal_area = Petal.Length * Petal.Width)
```

Hands-on: Create a new code chunk and use it to output a table of the first few lines of the newly transformed data set.

```
knitr::kable(head(iris_df))
```

Code Chunk Figure Options

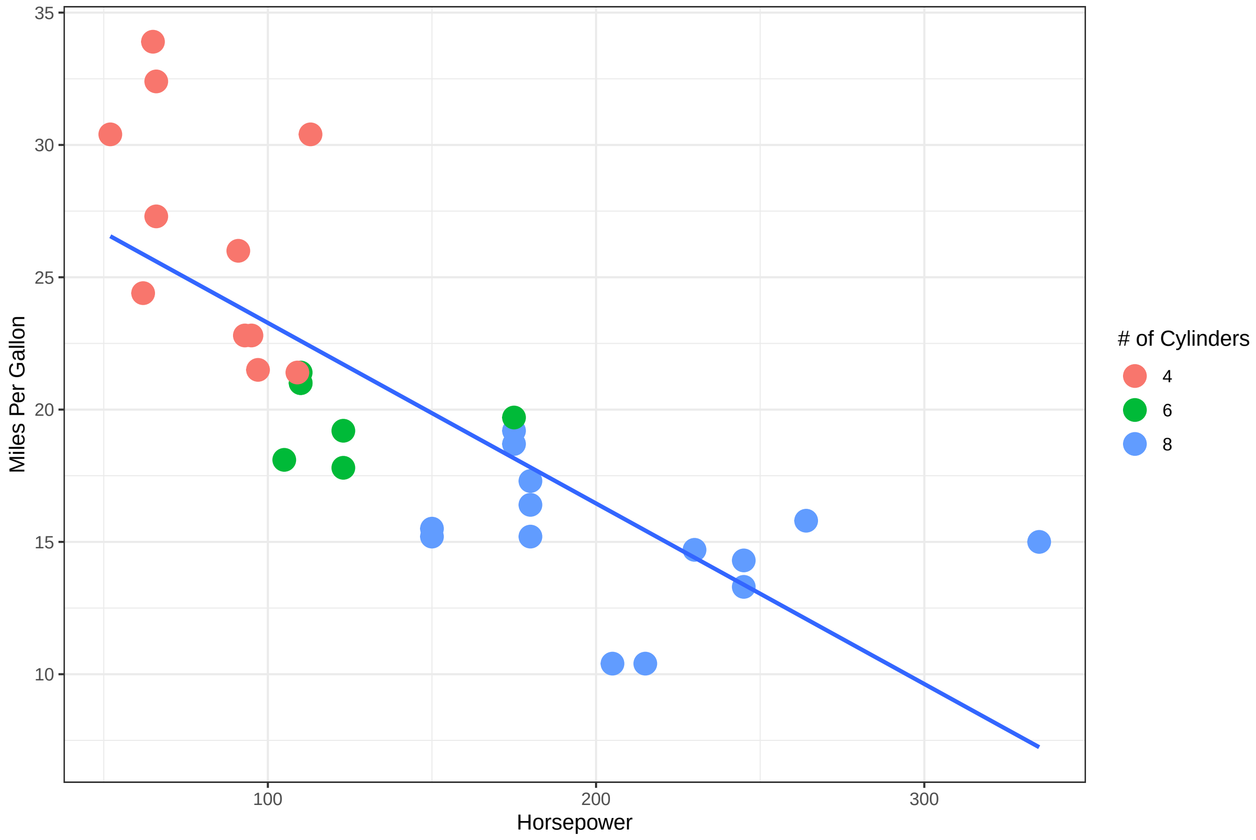
There are a whole set of optional arguments just for displaying figures

```
'''{r name, fig.height=6, fig.width=4, dpi=300, fig.align='center'}
  SOME CODE GOES HERE
'''
```

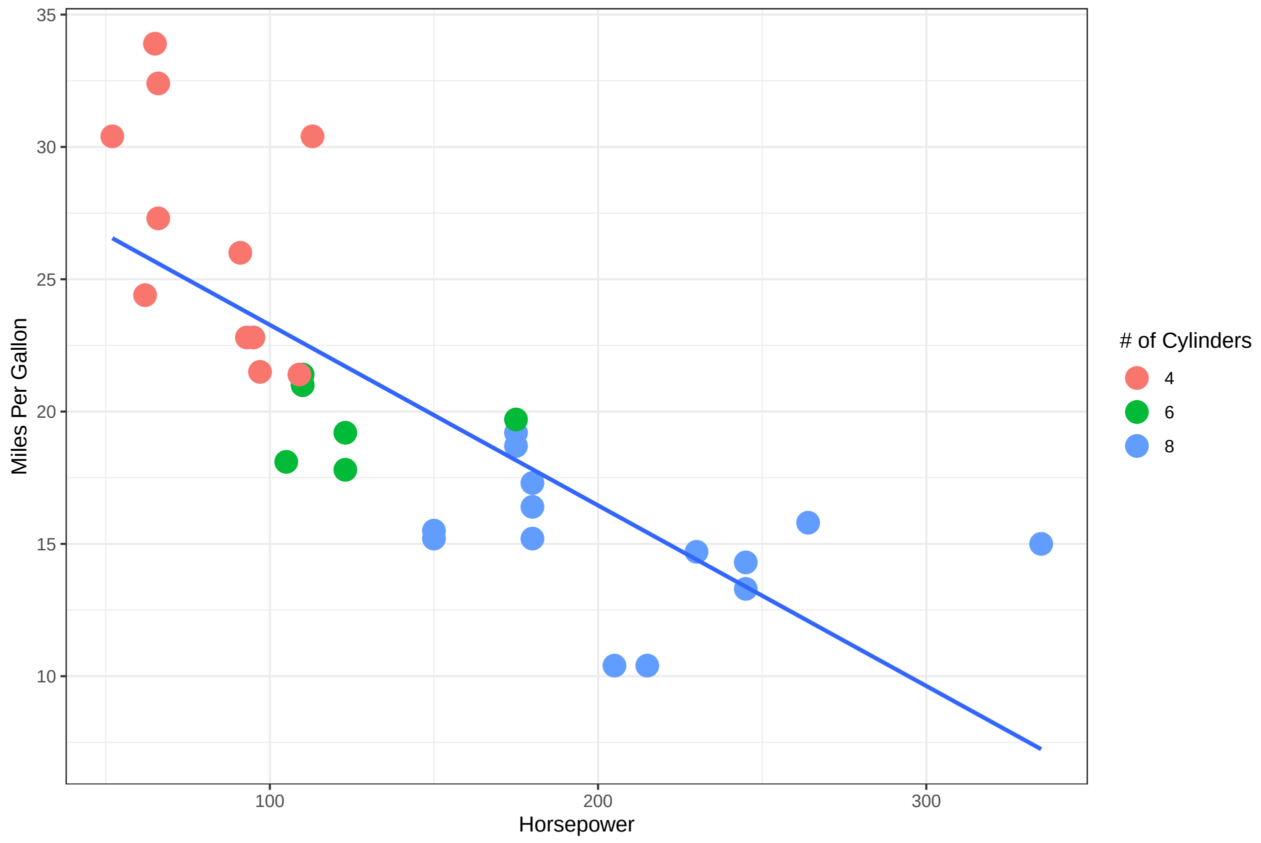
- **fig.height, fig.width** - Specify the height and width of the figure to make it fit into the space you have available.

- **dpi** - Specifies the pixels per inch. This effectively controls the size of the objects (text, lines, etc.) in your figure. Too low and your figure can look pixelated.
- **fig.align** - Specify whether your figure appears right, left, or center aligned.
- **fig.cap** - Enter text that will be included underneath your plot as a caption.

The plot from earlier with `fig.height=6`, `fig.width=9`, `dpi=75`, and `fig.align='center'`.



Below is the same plot with `fig.height=6`, `fig.width=9`, `dpi=300`, and `fig.align='center'`. Notice, the only thing I changed was the `dpi`. See how that affects the size of plot, the fonts, the points, etc. You can play around with `fig.height`, `fig.width`, and `dpi` until you find a combination that suits your preferences.



Hands-on: Create a new code chunk and use it to output a scatterplot that displays the Sepal Area and Petal Area variables calculated earlier.

```
ggplot(iris_df, aes(sepal_area, petal_area)) +
  geom_point(aes(color=Species), alpha=0.5, size=2) +
  labs(x="Sepal Area (in sq cm)", y="Petal Area (in sq cm)") +
  theme_bw()
```

Inline Code

Inline code allows you to create dynamic fills in your documents that update as the data are updated. If you have values that could potentially change when you add new data or make changes, you should consider inline code. Some examples:

- Numeric values or percentages
- The outcome of a statistical test (e.g., F-value, Degrees of Freedom, p-value)
- Text references to numeric values (e.g., whether a value increased, decreased, or stayed the same).
- Time-related variables (e.g., month that the report refers to)

The syntax for inline code starts with a `'r` and ends with `'` (backticks).

```
Some text 'r CODE GOES HERE' some more text.
```

Let's say, for example, you want to write a sentence that updates when new data are collected each month. You first provide the data:

```
lastmon <- 4.7
lmon <- "May"
thismon <- 4.9
tmon <- "June"
```

And then type the inline code:

- The unemployment rate in **'r tmon'** was **'r paste0(thismon, "%")'**, 'r ifelse(thismon < lastmon, paste0(" down ",paste0(abs(thismon-lastmon),"% from ")) ,ifelse(thismon > lastmon, paste0(" up ",paste0(abs(thismon-lastmon),"% from "))," unchanged from "))' **'r paste0(lastmon, "%")'** in **'r lmon'**

And it would look like this:

- The unemployment rate in June was 4.9%, up 0.2% from 4.7% in May.

Exercise: Update the sentence below from the earlier text formatting exercise. Replace '150' and '5' with inline code calculations. **Hint:** Calculate the number of rows and number of columns respectively.

```
iris is a data frame with 150 cases (rows) and 5 variables (columns) named:
```

Output

When you are ready, you can "knit" the document to some format. HTML is available right away, but you need to install a LaTeX package in order to knit to PDF or Word. Other options are available (e.g., kindle) if you download other packages.

Hands-on: Practice knitting your document to different formats. **NOTE:** You may not be able to output to PDF or Word if you did not install MikTex or some other LaTeX software.

Miscellaneous Information

- You don't have to write all of your code into the R Markdown document. If you have custom functions, for example, you can write those in a `.R` file and then run `source('functions_file.R')` in one of your early code chunks
- Similarly, you can use the source file to import your libraries
- You can use R Markdown documents to show bad, even non-functional, code. Just be sure to add `eval=FALSE` to the code chunk options so it doesn't run.
- When you knit to HTML, you end up with a single HTML file. Your images get converted to some code gibberish. This can be changed by adding `self_contained: false` to the YAML file under `html_document`.

```
---
output:
  html_document:
    self_contained: false
---
```

Additional Resources

These websites can help fill in some of the gaps left by this document.

- R Markdown by RStudio
- R Markdown for Data Science
- R Markdown Definitive Guide
- Reproducible Reserach Class on Coursera
- R Markdown Cheat Sheet (PDF)

Questions & Comments

If you have any questions or comments, you can contact me at kopp.brandon@bls.gov.