Reproducible Templates *(Book in Development)*

Melinda K. Higgins

2018-02-18

Contents

List of Tables **7**

List of Figures **9**

Preface **11**

Why read this book . . . 11

Structure of the book . . . 12

Software information and conventions . . . 12

Acknowledgments . . . 12

Prerequisites . . . 12

Colophon . . . 13

About the Author **15**

I Part One **17**

1 History & Exemplars **19**

1.1 Reproducible Research . . . 19

1.2 Literate Programming . . . 20

1.3 Dynamic Documentation . . . 21

2 Why Reproducibility? **23**

2.1 Data . . . 23

2.2 Organization & Workflow . . . 23

2.3 Dissemination . . . 24

2.4 538.com . . . 26

2.5 Saving Lives . . . 26

3 Getting Started **27**
3.1 R . . . 27
3.2 RStudio . . . 28
3.3 Github . . . 28
3.4 GIT . . . 29
3.5 R Packages . . . 30
3.6 other . . . 30

4 Exercise with GIT & Github **31**
4.1 Using Git and Github . . . 31
4.2 Using the RStudio Interface . . . 33

5 First Project and Document **37**

6 Document Components **39**
6.1 Sections of a Document . . . 39
6.2 YAML . . . 39
6.3 BODY . . . 39
6.4 Document Metadata . . . 39

7 Document Formatting - R Markdown Syntax **43**
7.1 Script . . . 43
7.2 Body of text . . . 43

8 Document Elements **47**
8.1 Script Intro . . . 47
8.2 Figures . . . 47
8.3 Images . . . 49
8.4 Tables . . . 49
8.5 Videos and animations . . . 50
8.6 Equations . . . 51
8.7 Footnotes . . . 51

9 Presentation Formats **53**
9.1 Intro Script . . . 53
9.2 Ioslides . . . 53
9.3 Slidy . . . 54
9.4 Beamer . . . 55
9.5 RevealJS . . . 56
9.6 Backup . . . 57

10 Book Format **59**

10.1 Intro Script 59

10.2 Bookdown 59

11 Exercise - Air Quality Report **63**

11.1 Intro Script 63

II Appendix **65**

A First appendix section **67**

B another appendix section **69**

Bibliography **71**

Index **72**

List of Tables

List of Figures

Preface

This should be s short 2-3 paragraph summary of the book, what it is about and such.

Why read this book

This should cover the purpose of the book and what the read will gain after reading the book.

Include who this book is for - having some previous experience with R is helpful, but a raw beginner could read these book - but it is assumed that the reader knows how to install software on their computer, connect to the Internet, find, create, copy and move files and folders on their computer. some previous experience working with document and presentation software like Word, google docs, open office, powerpoint, others xxxxxx…

Include things like:

- coursera course
- filling in the gaps
- seemingly disconnected applications brought together with rmarkdown glue

Coursera Course

This book is based on the Coursera Course "Reproducible Templates for Analysis and Dissemination", https://www.coursera.org/learn/reproducible-templates-analysis

Coursera Course Description: *"This course will assist you with recreating work that a previous coworker completed, revisiting a project you abandoned some time ago, or simply reproducing a document with a consistent format and workflow. Incomplete information about how the work was done, where the files are, and which is the most recent version can give rise to many complications. This course focuses on the proper documentation creation process, allowing you and your colleagues to easily reproduce the components of your workflow. Throughout this course, you'll receive helpful demonstrations of RStudio and the R Markdown language and engage in active learning opportunities to help you build a professional online portfolio."*

The Course consists of 5 Modules:

1. Introduction to Reproducible Research and Dynamic Documentation
    - This module provides an introduction to the concepts surrounding reproducibility and the Open Science movement, RStudio and GitHub, and foundational cases and authors in the field.
    - 11 videos, 6 readings, 1 practice quiz
2. R Markdown: Syntax, Document, and Presentation Formats
    - This module explores the R Markdown syntax to format and customize the layout of presentations or reports and will also look at inserting and creating objects such as tables, images, or video within documents.

- 8 videos, 11 readings, 1 practice quiz

3. R Markdown Templates: Processing and Customizing
    - This module goes further with R Markdown to help turn documents, reports, and presentations into templates for easier automation, reproducibility, and customization.
    - 9 videos, 6 readings, 1 practice quiz
4. Leveraging Custom Templates from Leading Scientific Journals
    - This module delves into custom templates available for websites, books, and scientific publishers, such as Elsevier and the IEEE, with the chance to create your first R Package.
    - 6 videos, 3 readings, 1 practice quiz
5. Working in Teams and Disseminating Templates and Reports
    - This module focuses on helpful tips for sharing and using the templates you create, as well as methods for organizing content. We'll also look at a few web-publishing services.
    - 6 videos, 2 readings, 1 practice quiz

Structure of the book

A verbal summary of what is in the book, each capte or section, organization and workflow - does it have to be sequential or can the reader jump around as needed - have to reads versus optional reads…

Software information and conventions

Software assumptions, workflow and style conventions used in book. Maybe include stuff here on hints, warnings, and such.

Acknowledgments

People to thank:

- emory, cfde, tnt team
- coursera
- chester ismay - fivethirtyeight package and feedback
- yihui xie - online help and guidance with bookdown
- developers of R, Rstudio, rmarkdown, bookdown, knitr, …

Prerequisites

Placeholder for now - maybe come back and add details on getting started - what assumptions are made for getting setup to work thriugh the exercises in this book - to get the full experience…

See more in the "Getting Started" Chapter 3.

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need to install XeLaTeX.

Colophon

R Packages Used in This Book

This book will use the `R` programming language (R Core Team, 2017) with the following `R` packages:

1. `bookdown` (Xie, 2017a)
2. `rmarkdown` (Allaire et al., 2018)
3. `knitr` (Xie, 2017b)
4. `dplyr` (Wickham et al., 2017)
5. `ggplot2` (Wickham and Chang, 2016)
6. `printr` (Xie, 2017c)
7. `fivethirtyeight` (Ismay and Chunn, 2017)

Other external refs, book (Xie, 2015), and the FAD ref (Miller et al., 1985).

R Session Info as of 2018-02-18 07:30:13

This book was compiled using the `R` packages `bookdown`, `rmarkdown`, and `knitr` running under the following `sessionInfo()`:

```
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 15063)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] fivethirtyeight_0.3.0 printr_0.1            ggplot2_2.2.1
## [4] dplyr_0.7.4           knitr_1.18            rmarkdown_1.8.5
## [7] bookdown_0.5.10
##
```

```
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.13     rstudioapi_0.7   bindr_0.1        magrittr_1.5
##  [5] munsell_0.4.3    colorspace_1.3-2 R6_2.2.2         rlang_0.1.4
##  [9] plyr_1.8.4       stringr_1.2.0    tools_3.4.3      grid_3.4.3
## [13] gtable_0.2.0     htmltools_0.3.6  lazyeval_0.2.0   yaml_2.1.16
## [17] rprojroot_1.3-2  digest_0.6.14    assertthat_0.2.0 tibble_1.3.4
## [21] bindrcpp_0.2     glue_1.1.1       evaluate_0.10.1  stringi_1.1.5
## [25] compiler_3.4.3   scales_0.5.0     backports_1.1.1  pkgconfig_2.0.1
```

About the Author

Melinda Higgins has dual degrees in Chemometrics (PhD) and Statistics (MS) with 25 years experience in research, teaching, consulting, directing and managing projects. Her expertise includes programming/scripting languages (R, S, Pascal, Perl, Prolog) and statistical, mathematical, imaging, and geo-spatial processing software packages (R, SAS, SPSS, MATLAB, SYSTAT, ENVI, ESRI ArcView, IMAGINE). While at Georgia Tech Research Institute (1994-2011), she coordinated large team projects with rigorous timelines, milestone tracking and version control in the areas of remote sensing, geospatial information systems, sensor fusion and target recognition. In her current work at Emory (2007 –), she has over a decade of expertise mentoring students and faculty in nursing and public health science research and scholarship. Her health research experience includes pattern recognition, phenotype characterizations and longitudinal modeling in heart failure, diabetes, cognitive impairment, and HIV/AIDS chronic disease populations.

Part I

Part One

Chapter 1

History & Exemplars

1.1 Reproducible Research

So why is reproducible research important and incredibly beneficial? Let's take a look at its beginnings to find the answer.

In the early '90s, a geophysicist named Jon Claerbout revised his book Earth Soundings Analysis with a valid complaint. He claimed that few published results are reproducible in any practical sense. To verify them requires almost as much effort as it took to create them in the first place. After a time, even the authors are often unable to reproduce their own results! For these reasons, many people ignore most of the literature.

Then, in 1996, the Consolidated Standards of Reporting Trials (or CONSORT) published a set of guidelines to fix problems that developed from inadequate reporting of randomized controlled trials. Following their lead, in 2004, the International Committee of Medical Journal Editors stated they wouldn't publish a clinical trial that had not been registered, and that they would only endorse registries meeting several key criteria, including that they must be:

- free and publicly accessible,
- open to all prospective registrants,
- managed by a non-profit organization, and
- electronically searchable and validated

As a result of this turmoil in validity and research, the Food and Drug Administration got on board and required the registration of even more clinical trials. The Journal of Biostatistics also encouraged reproducible practices of author submissions. They began marking accepted papers based on the standards of reproducibility that were followed. For example, papers marked with a 'D' marking meant the data on which the study is based is freely available. A 'C' marking means the authors' code is freely available. And an 'R' marking is the gold standard, meaning that not only are the data and code freely available, but the associate editor for reproducibility was also able to reproduce the same results as the paper.

An example of an article given the highest designation for a fully reproducible article is one published in 2009 entitled "Air pollution and health in Scotland: a multicity study." To see the article's marking, you have to download the PDF and look for the marking letter in a bold box at the top right.

Most compelling, in the early 2000s, John Ioannidis published an article with the highest downloads in the history of the Public Library of Science. It was entitled "Why most published research findings are false." Despite multiple organizations attempting to fix this issue, the Open Science Collaboration revealed in 2015 that they were only able to reproduce or replicate between 30-50% of the results from more than 100 studies. Ziemann, et.al. in 2016 also found that 20% of papers published in leading genomics journals

have supplementary data files containing erroneous gene name conversions due to Microsoft Excel default settings. This 20% is an average, and some journals have even higher rates.

But this isn't new. In 2011, Alsheikh-Ali [Al shake], et.al. assessed 500 research papers with unsettling results. Of these 500 papers sent to high-impact research journals, 30% were not subject to any data availability policy. The papers that adhered to the data availability instructions did so by publicly depositing only the specific data type as required, making a statement of willingness to share, or actually sharing all the primary data. Overall, only 47 papers (that's only 9%!) deposited full primary raw data online.

In the last several years, reproducibility errors have been at the center of some major controversies. In 2010, a published cancer clinical trial at Duke University was tested by two MD Anderson researchers, Keith Baggerly and Kevin Coombs. They found numerous spreadsheet errors leading to misalignment and incorrect assignment of cancer treatment therapies. Because of this, four papers published by the Duke team were retracted, the Duke lead scientist resigned, and Duke shut down three other trials using these results, and many patients have pursued legal action.

Another famous study, often called the "excel-error heard round the world" – was based on a paper by two well-known economists at Harvard, Kenneth Rogoff and Carmen Reinhart. In their paper "Growth in a Time of Debt", the authors claimed that countries whose debt exceeds 90 percent of their annual gross domestic product experience slower growth than countries with lower debt — a figure that's been cited by many people in order to justify slashing government spending. But when Thomas Herndon, a 28-year-old economics graduate student at the University of Massachusetts tried to reproduce the results, he discovered a major formula error in the excel data spreadsheet. The original paper had excluded key data from the countries of Canada, New Zealand, and Australia — all countries that experienced solid growth during periods of high debt and thus undercut the conclusion that high debt forestalls growth.

Due to these critical moments in the last few decades, reproducibility continues to grow as new policies are adopted and the practices are applied. But part of the benefit of being in this course is that you get to be part of the reproducible research movement!

1.2 Literate Programming

In 1991, around the same time Jon Claerbout coined the term "reproducible research", the computer scientist, Donald Knuth, introduced the concept of "literate programming." The idea of literate programming is that software/computer programs are written in a language humans can understand rather than a language only machines can understand. In literate programming, computer code is embedded within the program's documentation as opposed to the documentation embedded within computer code; the code follows the structure of the documentation.

The program that Donald Knuth used to implement his idea of "literate programming" was called WEB, which he introduced in 1981. WEB linked the TeX typesetting or formatting system for creating documents with the Pascal computer programming language. WEB was one of the first systems to directly link documentation creation and typesetting with computer programming. Donald Knuth chose the name WEB because it implied a program of ideas pieced together from simple materials.

Since WEB was introduced, many other programs implementing literate programming have emerged. Here are a few to give you an idea of the variety available.

- CWEB also created by Donald Knuth with Silvio Levy which was adapted for the C and C++ compute language instead of Pascal
- Axiom developed by IBM
- Noweb
- Literate
- Funnel WEB
- Molly

- Codnar
- Jupyter Notebook (formerly IPython Notebook) and
- R Notebooks

1.3 Dynamic Documentation

So literate programming is an approach that moves away from writing computer programs in a high-level machine language and instead combines programming language with documentation language so that the program reads almost like an essay or a piece of literature. But what about dynamic documentation? Dynamic documentation allows for constant change and is a tool that provides up-to-date reports if certain components, such as data or analysis, change.

In 2002, Friederich Leisch, a statistics professor from the University of Natural Resources and Life Sciences in Vienna, released the SWEAVE program for dynamic documentation generation. Notably, SWEAVE allows R code to be embedded within LaTeX documents. LaTeX is a more modern version of the TeX typesetting program used by Donald Knuth. The really exciting feature of literate programming and dynamic documentation is highlighted in what Friedrich Leisch says about SWEAVE: since the underlying computer code is wholly integrated within the document itself, anytime there are changes to the underlying data or analyses or code, the report itself is automatically updated ON THE FLY!

The next evolution of ideas for literate programming and dynamic documentation have emerged from the R programming and RStudio communities. In 2012, Yihui Xie (yeewhay she) released the R package called knitr. This package was inspired by SWEAVE, and thus combines R code with text typesetting for producing documents. Like SWEAVE, knitr works with LaTeX but it also works with rmarkdown, which uses simple text markup syntax based on the original "markdown" package. The primary "markdown" package was introduced by John Gruber in 2004 to make it easier to "markup" plain text files for generating HTML documents – ideally without having to learn HTML. The rmarkdown package you'll use throughout this course is built upon John Gruber's "markdown".

Rmarkdown itself was fully released in 2014 and its original objective was creating documents for the internet by creating HTML formatted documents. However, the "rmarkdown" package also leverages Pandoc for creating an even wider array of documentation formats – including:

- The DOC format, as used by Microsoft WORD or Google Docs
- The ODT format used by Libre Office
- The PDF format
- EPUB for electronic-books
- Slide shows using HTML5
- And the original TeX document formats and related TeX based slide formats like Beamer.

In this course, you won't interact directly with Pandoc, but it has been bundled with RStudio since 2015– so when you install RStudio, you'll also get the functionality of Pandoc. If you would like to learn more about Pandoc, you can visit their website at pandoc.org. Since Pandoc can convert many different document formats, it's often called the "swiss army knife" for document conversion. Pandoc is extremely versatile, allowing conversion between HTML web-based formats, word processor type formats, electronic publishing (or EPUB) formats, presentation slide-based formats, publication layout formats, TeX based formats, and many others.

Ultimately, the RStudio Interactive Development Environment becomes our central "HUB" for combining the capabilities of:

- The great packages of "knitr" and "rmarkdown"
- with the built-in functionality of Pandoc for document conversion
- plus the fantastic analysis and graphics capabilities of the R programming language.

From the RStudio interface, we can access all of this functionality and create documents on the fly in multiple formats for multiple end uses and products.

Chapter 2

Why Reproducibility?

This chapter will also cover thinking about what kinds of work products and projectscan benefit from reproducible workflow principles - how to get organized and xxx

2.1 Data

Data can be thought of as many different things. We often think of data as numbers or even short text in a spreadsheet. But more often than not, data is "unstructured." Unstructured data includes text, which could come from multiple sources, including not only reports and documents, but books, blogs, and websites. Other kinds of data could be:

- Images and artwork
- video and other media
- interview transcripts
- and any other "RAW" materials needed to complete your project.

Regardless of what kind of "data" you have, your data should be:

- high quality
- reviewed for completeness
- reviewed for mistakes and errors, and
- checked for changes or updates

2.2 Organization & Workflow

Because your projects may involve a variety of dynamic data, how do you ensure your reproducible workflow is always efficient? There are several principles to follow. The first starts with organization. Each project you work on should have its own file storage organization structure. Each document, code, script, and product should have a specific purpose, and the versions of these files should all be tracked with a version control system without creating multiple copies of the files.

Following this lecture, I've included a reading page with a helpful example of great organizational structure on Github.

File names should be:

- readable by the computer, easy to search, easy to sort (especially by date and author if needed)
- human readable with logical naming schemes and contain enough info so a human knows what is in the file and what the file is for
- and short enough to be reasonably manageable
- consider user-based access and security (partitioned by "need to know" [users with editing and write permissions versus users with read-only access]

Having an organizational structure for your project is a good idea even if your project only includes yourself, because:

- projects grow
- you may need to support numerous documents and files
- And relationships change and can become complex

No matter what kind of product you want to produce, there should also be instructions on how to use and combine the files in your project. Your documentation is another important component, and it should be clear and well-defined so it can be easily understood by team members at all levels. The documentation could also follow literate programming principles combining the code + text + figures in one document.

Ideally, your final workflow will allow any changes and updates to be automatically incorporated into your final product. You should write code/scripts to automate:

- raw data to processed output
- creating and removing temporary files
- creating tables, figures and other components
- assembling the components into final documents, products, and
- rendering documents into multiple-desired formats

Standardization is also a critical component. Your documentation, code, or templates might be used again in other projects and should be standardized for easier integration and efficiency. You don't want to reinvent any wheels if you can help it.

Finally, your files, documents, and code should be stored and shared in a centralized way. Cloud-based computing often provides centralized storage and sharing of your projects with your team members and external stakeholders.

2.3 Dissemination

Once your project is complete, you should disseminate your work. Why?

- To store and share your data and code. Odds are you will reuse something from this project in a future project.
- To fulfill expectations/requirements to disseminate your findings by the funding agency or publisher of your work
- To increase visibility - when you are listed as the source, you become, by default, THE subject matter expert!
- To increase the speed of collaboration for faster advancement of science and knowledge in your field, and finally
- To increase goodwill with the community and public

Some ways to disseminate your work using Cloud-based solutions are:

- Dropbox
- Google drive
- Github (better with version control and tracking)

Other ways to disseminate may be through:

- Journals - articles, manuscripts
- Books
- Blogs/Websites
- RSS (Rich Site Summary) feeds – like news feeds
- Rpubs – which we will discuss and try out in future lessons in this course
- Other online book platforms such as Gitbook and Bookdown

Some examples of data repositories are:

- GenBank
- PDB

In addition to Github, other data and code sharing repositories include:

- Bitbucket
- Dryad
- Figshare
- Zenodo

A helpful article was published in 2013 in the journal PLOS Computational Biology entitled "Ten Simple Rules for Reproducible Computational Research." While the article focused on applications in computational biology, the key principles they recommended still apply, and include:

- avoid manual steps
- use version control and tracking
- implement standardized formats
- store and track raw data
- organize your output – their list recommends a hierarchical organization
- link textual documentation to the results
- and make the work transparent by allowing public access to scripts, runs, and results

When considering standard practices, think about your own work:

- What do you want to automate?
- What could you re-use?
  - For example, code, files, formatting, graphics, logos, header, footer, boilerplate?
- What should you share with your team?
- What do you find yourself doing over and over?
  - correcting or reformatting?
- If you won the lottery today and left your job, what do you need to tell your replacement so that they can pick up where you left off and complete your current tasks?

The purpose of this course is to help you find the answers to these questions to improve your own workflow, teamwork, and efficiency!

2.4 538.com

A good example of an organization that follows reproducible principles is 538.com. They write and host stories and opinion pieces covering politics, economics, health, popular culture, and sports. The founder, Nate Silver, and the 538 team are best known for their political polling and forecasting during the United States Presidential and related elections since 2008.

Most of their articles provide references and links to their original data sources, and they also host their data, code, and details behind their analyses on their Github, which is available to the public. We're going to work with some of these datasets later in this course using the "fivethirtyeight" R package.

It's also worth mentioning Andrew Flowers, one of the contributors to the 'fivethirtyeight' R Package. He gave a great presentation at the 2017 RStudio conference on how to tell stories using data, and he highlighted the various aspects of "data journalism" and importance of workflow, data processing, and transparency in analysis and communication. These are all key aspects of reproducibility.

2.5 Saving Lives

To really see the power and importance of reproducible workflow principles, let's go back in history to 2001 where an outbreak of a deadly strain of e.coli bacteria killed 50 people in Europe. Researchers at the Beijing Genomics Institute worked in collaboration with the Medical Center in Hamburg-Eppendorf to rapidly sequence the genome of the e.coli pathogen. Given the severity of the outbreak, the team announced and released the genome via Twitter to the world-wide community of microbial genomicists. A Github repository was established to "crowdsource" analysis and research to find a treatment.

People started contributing their work in under 24 HOURS, and within 5 DAYS a bacterial agent was proposed to kill the pathogen. This case highlights the importance of these methods and work practices not only for speed and efficiency but also for rapidly addressing problems and developing solutions to save lives.

Chapter 3

Getting Started

This chapter will cover the software tools needed - installing R, RStudio, GIT and Github to get started. And will include installing the various packages needed for the exercises in this book.

3.1 R

So what is R? R is a language and environment for statistical computing and graphics. R is based on the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

R is Free - both in terms of no cost but also as FREELY distributed and shared under the GNU general public licensing.

To learn more about R, you should visit the R-project website. This site provides good information about what R is, who the key contributors are, and information about the development of the R language. Links are provided for the manuals, frequently asked questions, and other resources like books about R and "The R Journal". At the top of the page is a link to CRAN or C-RAN where you can download the R software.

Go ahead to the CRAN website to download R. The link from the R project website, takes you to the list of "mirrors" or servers around the world that host the code and files and installers for installing R. You should pick the mirror closest to your geographic location. For example, at the bottom is the list of mirror sites for the United States. The one hosted by Duke University is closest to my location.

You can also access this download page by directly going to https://cran.r-project.org/ At the top, there are links for the different operating systems for Windows, Mac or Linux. Choose the one for your operating system. For Windows, you'll want to click on the link for the "base" installer. This will take you to a page with a link to the executable (EXE) file that you'll need to download and run to install R on a Windows computer. When you click on the link for the Mac operating system, you are provided the link to the package (PKG) file needed to install R on your Mac.

Go ahead and take a few minutes to download the installer needed for your operating system. Run the installer, follow the instructions, and accept the defaults to install R on your computer.

Once R is installed, for example, on a windows computer, you will see R listed in your `/Start/Programs` list and may also have the R program icons shown on your desktop.

It is worth noting that this is the minimum software you need to use R. For example, we can run the R program and when it opens you get a simple command line interface. You can use this to submit and execute R commands. For example: you can do simple math like typing in `2+2`, or finding the mean of an array of numbers like `mean(c(1,2,3,4,5))`. Try this out on your computer to test and make sure R is up and running on your system before installing RStudio.

3.2 RStudio

Programming in R using the basic interface is not the best way. Let's also go ahead and download and install the RStudio software. RStudio is a fully integrated development environment (IDE) and is the key interface we'll use for the rest of the course. Not only does RStudio link directly to R and provide a much better programming interface, RStudio allows you to create great rmarkdown documents in multiple formats and then links everything to your Github repository with version control using Git. We'll cover Github and Git in the next lesson.

Go to https://www.rstudio.com/ I encourage you to explore the many other products and services available from the RStudio organization. Check out their resources, which include free webinars, videos, and online learning.

But let's go ahead and download and install RStudio. Go to products and click on RStudio desktop. We will be using the FREE Open Source edition. Click on Download RStudio Desktop. Click the download button for the FREE version – this scrolls down to a list of installers. You need to read the file names to find the one right for your operating system. The first link is for the Windows installer, next is Mac followed by various flavors of Linux. You'll want the "Installers" not the "TarBalls or "Source Code" – these are primarily for developers.

Go ahead and take a few minutes to download and install RStudio and get it up and running on your computer.

Once RStudio is up and running, you should see something that looks like this. We will explore this interface further in future lessons, but for now, let's look at a few basic things. The main window on the left is the same basic "console/command line" window that you saw when you ran the basic R software. Like before, we can type commands and R code here. Like `2+2` and `mean(c(1,2,3,4,5))`. But you'll notice there are more windows on the right side including information on your environment, history, files, plots, packages, help and viewer. To learn more about the RStudio interface, I've included several helpful links in a reading page after this video lecture. There are literally thousands of resources for learning more about both R and RStudio. Just pick your favorite search engine and search for tutorials on R and RStudio.

3.3 Github

So what is Github? It's a cloud repository, which hosts things like code, files and documents. It's very similar to Dropbox, Google drive and Microsoft's One Drive.

However, Github also includes version control and tracking using Git, which we'll get to shortly. Github has a web-based interface that includes support for desktop and mobile integration.

Github provides access control and collaboration features such as bug tracking, feature requests, task management, and wikis.

It also has native support and interpretation of markdown that's much easier to use and write than HTML. We're going to learn more about markdown at the end of this module.

So let's set up your Github account. Go to https://github.com.

1. Choose a Good Username for Your Github Account

    a. Pick something professional that represents you.
    b. This will be your identity on Github and will be viewable by everyone.
    c. NOTE: For this course, I assume that you are creating a PUBLIC Github account, which is FREE. You can create a PRIVATE Github account for a fee.

2. You can register one Github account per email.

3. Once you get logged into your Github Account, go to your account settings to customize your photo, bio, email, website URL, and more…
4. When you first get started you won't have any repositories, but we will be creating repositories for each project. [BEGIN computer demo]

[NOTES TO MYSELF - COMPUTER DEMO]

- Show create account and log in screen
- Once you are signed in, click on the icon on the top right – click the pull down arrow to see selection options…such as accessing your profile and settings
- Click on your settings – check your name and email. These are IMPORTANT – you need to know these to set up Git for version control and connectivity from your cloud account to your local computer
- Add your bio summary, a URL, your photo, and any other information you want to share with everyone
- View your profile page – this is your "home" on Github – your page will look different from mine. When you first create your account you won't have any repositories. However, we will be creating new repositories for this course shortly.

[END computer demo]

3.4 GIT

Details in installing GIT

Now that you have your Github account created and you are logged in, we're going to install Git. GIT is a source code management system for software development. It was designed and developed in 2005 by the Linux developers.

GIT is a distributed version control system with complete history & version-tracking capabilities. You may have heard of other version control systems, like Subversion, CVS, Perforce, and ClearCase.

Unlike some of these, GIT is FREE (cost) and freely distributed under the terms of the GNU General Public License.

[BEGIN computer demo]

[NOTES TO MYSELF - COMPUTER DEMO]

- Download and install Git from https://git-scm.com/ - click "Downloads" – at the lower left side of the web page
- This opens another web page. This page has the links for downloading the installer files for Mac, Linux and Windows operating systems. Choose the download link for your operating system – NOTE: Clicking these links starts the file download.
- Run the installer you just downloaded to install Git on your computer. Follow the instructions and accept the defaults.

For example on my windows computer, I can go to the start programs and see that Git was installed and has 3 options for running GIT:

- Git Bash – for this course we will use the Git Bash option
- Git CMD
- And Git GUI

[computer demo continued]

3.5 R Packages

Details on what R packages are, why you need them, and how to install them.

3.6 other

Latex optional… word, open office, google docs, powerpoint, Internet browser software (IE, Edge, Firefox, Chrome, Safari, xxx)

Chapter 4

Exercise with GIT & Github

4.1 Using Git and Github

Now that you have Git installed on your computer and you've created your Github account, let's test your setup.

1. Open your browser and log back into your Github account
2. Click on your Profile, and then Click on Repositories – now we're going to create a new repository
3. Click NEW to create a new repository.
    a. type in a name for your repository such as "MyFirstRepo"
    b. put in a short description like "My First Github Repository"
    c. this will be a PUBLIC repository, but as you can see if you have paid for a PRIVATE Github account you do have the option to create Private repositories
    d. Go ahead and click the box to select "Initialize this repository with a README"
    e. keep everything else the same (use the defaults)
    f. click "Create Repository"

It takes a moment for the repository to be created, but you'll notice that your repository now has 1 file in it. README.md, which is your readme for the repository.

Now we're going to connect everything back to your local drive using Git.

We need to create a place on your local drive where you want to save your work for this course. We're going to end up creating multiple repositories for this course, so I create a central folder on your computer like `C:\RepTemplates` where you'll keep everything organized.

You can see this folder created on my computer. It is this folder where I will store and link all of my Github repositories for this course.

Let's go ahead and run GIT. As I mentioned, we will use the Git Bash command window for running and executing GIT commands.

Once the GIT Bash window opens, you'll see some information and details in the window about what directory/folder it's currently in. On my system, GIT Bash defaults to my "users" directory.

However, we want to change out of this directory. Keep typing

```
cd ..
```

until you get to the main "C" drive. Then we're going to change to the RepTemplates folder we just created. Type

```
cd RepTemplates
```

You should see the directory folder change at the GIT Bash command line, but you can also type the command

```
pwd
```

To get the "path with directory" to verify that you ended in your `C:\RepTemplates` folder as intended.

We can also view the contents of this folder, by typing either

```
ls
```

To "list" the files in this directory or you can also type

```
dir
```

To get a "directory" listing of the contents. You'll notice at the moment there is nothing in this folder. That's fine. That's correct. In a minute we're going to link back up to our newly create Github repository "MyFirstRepo".

[END computer demo]

As we go through this course, I will refer many times to the book by Jenny Bryan entitled: Happy Git and Github for the useR

You can access this book for FREE online at http://happygitwithr.com/ There's a lot of good information on setting up R and RStudio and for getting setup using Git and Github.

[BEGIN computer demo]

Now to get started using GIT, you need to "introduce yourself." At this point, you should already be logged into your Github account. But we need to make sure that GIT understands how to talk to your Github account. So, we're going to type in 3 GIT commands in your GIT Bash window. Open your GIT Bash window.

This first command tells GIT your name – be sure to type in the same name you used when you set up your Github account. Your name goes between the 2 single quote marks.

```
git config --global user.name 'Jennifer Bryan'
```

Next we also have to tell GIT the email account you used when you set up your Github account. Again put your email in between the 2 single quote marks.

```
git config --global user.email 'jenny@stat.ubc.ca'
```

Finally, to check to make sure everything went in correctly, type in the following GIT command to list your global settings and you should see the user.name and user.email you just typed in.

```
git config --global -list
```

If you see these, CONGRATULATIONS you have successfully introduced yourself to GIT!!

KEEP your GIT Bash window open.

[END computer demo]

"pushmi-pullyu" SLIDE with PUSH / PULL GRAPHIC – insert here

We'll be using the terms PUSH and PULL to talk about moving files back and forth from our local computer to the Github cloud repository and from the cloud back to our local computer.

The "pushmi-pullyu" was a fictional animal in the Doctor Dolittle series of children's books by Hugh Lofting with two heads on opposite ends of its body, so you never knew if the animal was coming or going.

Hopefully, we won't have that confusion in this course, but we will be PUSH'ing and PULL'ing content in and out of your project repository between your local computer and your Github account using Git version control.

A PULL moves content from the cloud to your local computer.

A PUSH moves content from your local computer to the cloud.

[BEGIN computer demo]

Now let's CLONE your Github repository to copy the repository contents from your Github cloud repository down to your local computer.

Open your browser, and go to your "MyFirstRepo" repository. At the top right, there's a green button to "Clone or Download" your repository. Just below that green button, there's a little icon to the right to "copy to the clipboard" the long URL address you will need when we use GIT to clone your repository.

[END computer demo]

First PULL to Clone your repository SLIDE – insert graphic illustrating a PULL from the cloud

When you CLONE your repository, this is your first PULL. You will be PULLing the content down from your Github account to your local computer.

[BEGIN computer demo]

To execute a clone using GIT, open your GIT Bash window. Check to make sure you are in your `C:\RepTemplates` directory.

Go back to the "MyFirstRepo" repository and click "copy to clipboard" to get the Github repo URL. Make sure you have the option for "Clone with HTTPS" shown to get the correct URL.

Back in GIT Bash, Type git clone followed by the URL. Since the URL is now COPYied into your "clipboard", you can PASTE it into the GIT Bash window

```
git clone https://github.com/melindahiggins2001/MyFirstRepo.git
```

This will take a minute to run, but it should say that it is cloning your repository and you should not get any errors.

Now type in a ls or dir command to view the contents of your directory. You should now see a new folder created called "MyFirstRepo" in your `C:\RepTemplates` directory.

Then type in

```
cd MyFirstRepo
```

to change into this new directory and type ls or dir to view the contents. VIOLA!! You should now see the `README.md` file in this directory.

You can also see this file by viewing the directory contents in your file explorer. You may also be able to see a hidden folder called `/.git` which was created when you did the clone. If you can't see this folder, that's OK- it's usually hidden by default. I changed the settings on my computer so I can view these hidden folders.

[END computer demo]

TADA!! You have now successfully cloned your Github repository and have it linked from your local computer to Github using version control and tracking with GIT!!

We're going to do this again in the next part using the RStudio interface.

4.2 Using the RStudio Interface

Let's take a moment and look at some of the other content in the Happy Git and Github for the user book by Jenny Bryan. http://happygitwithr.com/.

There are many chapters in this book you may want to read and take a look at. For example, chapter 5 has information on setting up a Github account and chapter 6 has information on installing or upgrading both R and RStudio which you've already done. And chapter 7 covers installing Git which you've also already done.

Then in Chapter 8 there is information on introducing yourself to GIT which you've just completed.

If you would like to move beyond using just the Git Bash window and command line interface for using Git for version control, I recommend reading Chapter 9 on installing a more full-featured Git client. Jenny Bryan recommends either SourceTree or GitKracken.

Chapter 10 covers getting connected to Github which you just completed.

We're going to spend some time in this next part of the lesson, learning about setting up credentials on your computer using either HTTPS or SSH to securely connect to your Github account. These details are covered in chapters 11 and 12.

There is additional information in chapters 13, 14 and 15 on using RStudio with Git to connect to Github and manage your projects. I will be showing you how to use RStudio to connect to Github using Git shortly.

The later chapters 16, 17 and 18 provide examples of linking up projects with Github depending on whether the project is new or existing and whether you setup the project on Github first or last. For the projects we will be doing in this course, we will be creating new projects by setting up Github first.

The next section of the book provides some workflow examples. I point out Chapter 22 which covers Git commands some of which you've already learned. I also mention Chapter 26 entitled Burn it all down which is helpful to read when you have problems and Git stops communicating between Github and RStudio.

Now we're going to connect to your Github account using Git but from the RStudio interface instead of from the Git Bash window. Go ahead and start RStudio.

When you open RStudio you should see a screen similar to this but it won't look exactly like this and that is OK. Your layout should be similar. There are a few options we need to review and setup to make sure that RStudio known that you want to use Git.

In the tools menu, click on Global options. Click on the button for GIT/SVN. In this window we want to make sure that the box is checked for "enable version control interface for RStudio projects". Next we need to find where the GIT executable file is located on your computer. On my computer it is located on my program files folder for Git/bin. For example, if I click browse it shows where this is on my computer's hard drive. You'll notice that the file is named "git.exe" and is located in the "/bin" folder. You may also see an icon like the one shown here next to the filename. There is also a similar file under the "/cmd" folder, but this is NOT the one we want. We also DO NOT want the file for "git-bash.exe" NOR the one names "git-cmd.exe"

Also make sure you have the box checked for "Use Git Bash as shell for Git projects". This why I showed you earlier how to use the Git Bash shell window with your projects.

Since we're not using SVN you can ignore the line for SVN executable

[END COMPUTER DEMO]

[BACK TO SLIDES]

Now that we've got some of the options setup in RStudio for using Git, we next need to setup your Github account credentials on your computer so that each time you run a GIT command to connect and sync to your Github account you won't have to keep typing in your login name and password. You can setup your credentials by using either HTTPS (hyper text transfer protocol secure) or SSH (Secure Shell). These are two different approaches for setting up your credentials. I'm going to show you how to setup SSH from RStudio.

[BEGIN COMPUTER DEMO]

Back in RStudio in the Global options window for Git/SVN options, were going to setup your SSH RSA Key. This is for setting up a public key/private key cryptosystem. Click on the button to "Create RSA Key" and use the defaults. This is where you create the key. You can add a pass phrase or password, but this is optional. Note where on your hard drive it tells you where the security key will be created. Then click "create" to create your key. If you'd like to view your public key, click on the link to the right. Once you're done, click OK

Let's double check that GIT also now sees your SSH Key. Open your Git Bash window and type in this command

```
ls -al ~/.ssh
```

When you do this, you should see two files `id_rsa` (which is your private key) and `id_rsa.pub` (which is your public key). This is explained in more detail in the Happy Git book in chapter 12.2. You can also click the `[?]` Using Version Control with RStudio to get to the help webpages at RStudio.

Make sure you are in the local directory for your new repository `C:/RepTemplates/MyFirstRepo`. You should see this listed in your Git Bash window prompt or you can also type pwd to get the "path with directory"

You can double check your settings in the git bash window by typing

```
git config -global --list
```

You should be pretty much setup and ready to go at this point. If you are still getting errors, you might have a credentialing conflict. For example, if you have multiple Github accounts with different emails, you might have to remove one credential and add the other one instead. Search Stack Overflow https://stackoverflow.com/ or the Github help documentation https://help.github.com/ for help.

Let's go back to RStudio and create a New Project.

Chapter 5

First Project and Document

Pull from module 1 - lesson 7 - see slides and video...

Chapter 6

Document Components

6.1 Sections of a Document

In this lesson, you are going to learn a lot more about R markdown. I encourage you to read through the supplementary materials provided and spend some time reviewing the R markdown website by RStudio at http://rmarkdown.rstudio.com/index.html

[DEMO here of the website and quick overview of what is available – especially point out the steps and information provided in "Getting Started" – PLUS the Gallery, Formats and Articles]

However, let's also do a quick Overview of the components contained in an R Markdown Document:

6.2 YAML

At the top of an R markdown document you will typically have the "document metadata." This metadata is contained in a YAML header that has: o information about the document; o various parameters; and o formatting options; and o other options that can be customized

6.3 BODY

After the YAML header, comes the BODY of Document. The rest of the document will consist of: o Plain Text – this plain text will contain the content of your document along with Rmarkdown syntax for this course which is based on markdown o Most likely you'll also have some programming code included – these are called code chunks For this course you will be learning the R language, but in RStudio using the R markdown package there are options for including other computer languages like Python, Rcpp (R's version of C plus plus), the SQL database language and STAN (used for Bayesian statistical analyses). o You may also have other embedded "objects" such as: Figures; images; photos; pictures Tables Videos; animations Equations References; Or footnotes

6.4 Document Metadata

So, let's start at the beginning with the YAML document header which contains your document's "metadata". What is Document "Metadata"?

Every document (and file for that matter) you create has metadata. Metadata is often referred to as "data about data" – in other words, metadata is information about your document or file – not "data" within your document.

[DEMO on computer]

Let's see an example. On a Windows machine, if you open the file explorer and right click on a file (like a Word DOC file) and view "properties" and then click on "details" you get a lot of information about the file or document including: • The original author or user who created the file or document • Last date and time the file was saved and by who (which user) • The revision number • The program used to view/edit the file • The name of the company or manager for that file or software license (if available) • Depending on the file type sometimes there is a description of the content in the file • The file size (amount of hard drive space used to store the file) • And much more • Other file types (like computer programs or scripts) may have different information – for example, images and pictures will usually have information on the image size (e.g. width and height in pixels)

When we create an `R markdown` document we will have direct control over some of this metadata through information contained in the header of the document. This header information is provided and structured using YAML. So, what is YAML? It is pronounced to rhyme with camel.

YAML stands for "yet another markup language" or "YAML ain't markup language" depending on who you ask. You can learn a lot about YAML simply by searching the Internet using your favorite search engine. You can also visit the official YAML website at yaml.org but this website is aimed at programmers and is not very user friendly. However, the yaml.org website does offer some insight into how many programming languages and platforms YAML supports besides R markdown.

Technically, it is possible to create an R markdown file without a YAML header, but for this course we will always have some information contained and defined within the YAML header. We will be using the YAML header to define the parameters (or options) used by the Rmarkdown R package to "render" the final document. We'll talk more about the render function in future lessons, but for now, just understand that the information contained in the YAML header is used as instructions and input parameters for creating the final document.

[DEMO on computer]

Let's create a Github repository for this next exercise. Log into your Github account and create a new repository. Name your repository "Module2_rmd1" for your first R markdown document exercise in Module 2. Type in a description and add a README file and click create repository.

Once you have your Github repository created – click on "Clone or Download" and copy the URL to the clipboard.

Open RStudio and Click on File/New Project. Choose Version Control using Git. Paste in the URL you just copied and MAKE SURE you are creating the repository in the folder you created for this course `"C:\RepResearchCourse"` – that way you'll have all of the content for this course organized in one central location on your local drive.

Now that we've got a new RStudio project created, let's go ahead and create a new R markdown document. Click on File/New File/R Markdown. We will be creating a "document" – type in a title for your document like "Module 2 - R Markdown Document 1". Type in your name as author – it may already be entered. And keep the default output format as HTML. Click OK.

Everything you just typed in – your title, author name and the output format – you will see again when we review the YAML header information next.

When the document opens, you'll notice that the TAB is called "untitled" and there is an icon next to it that looks like a document with a red circle over it. It is hard to see on my screen but the letters "RMD" are inside the red circle. Let's go ahead and save this file and name it module2_rmd1. Click Save. The file format defaults to the RMD format "module2_rmd1.Rmd" and the file name changes in the TAB at the top.

Now, let's look at the document. At the very top there are 3 dashes - - - these indicate the BEGINning of the YAML header content. You should see information for the: • Title • Author • Date • Output

After the output there are 3 more dashes - - - indicating the END of the YAML header.

The words "title" "author" "date" and "output" are all YAML "key words" or parameters or options used by the render function in the R markdown package which compiles and creates the final document.

After each key word there is a colon : followed by your input for each parameter. For example, your title input is contained within beginning and ending double quotation marks "Module 2 - R Markdown Document 1". Similarly on the next line you have the key word author followed by a colon : and then in between beginning and ending double quotation marks you have your name. Likewise for the date. In a future lesson, I will show you how to use r code to automatically change the date to the current date and time if you wish.

Currently, the last line in the YAML header has the key word output : html_document. This was defined when we first created the New File/R Markdown document.

We will be learning a lot more about this YAML key word for output since this is where we define the parameters for customizing the various output formats we want. For now, watch what happens if we select different KNIT options. Let's first KNIT to HTML. Your document should open in the VIEWER window to the right.

If your document does not appear in the VIEWER window, check your KNIT options – click the gear icon next to KNIT and see which options are selected. The preview in window opens a new separate window. Preview in Viewer Pan open in the VIEWER window at the bottom right.

Now, let's KNIT to WORD to create a DOCX formatted file. As soon as you select this option, watch the RMD file, a second output option is entered for word_document: default. This additional option is added to your YAML header automatically. This is one example of how changing the options in your YAML header directly affects the output produced from your R markdown document.

You'll notice that the DOCX file also opens in a new window to preview the WORD Document.

OPTIONALLY if you have LaTeX installed on your computer, you can also try KNIT to PDF and again this will open in a new window to preview the PDF document. And a 3rd line of text is added to the YAML header for pdf_document: default.

For now this completes the overview of the YAML header. But we will be adding and removing more YAML keywords as we work through this course.

Go ahead and keep this R markdown file open for the next lesson.

Chapter 7

Document Formatting - R Markdown Syntax

7.1 Script

If you need to, go ahead and log in to your Github account, open RStudio, and open the RStudio project for "Module2_rmd1". We're going to keep working with the R markdown file you just created "module2_rmd1.Rmd". Go ahead and open this R markdown file.

[DEMO – computer]

7.2 Body of text

Previously we discussed the information contained at the very top of the document. This is the YAML header which contains the title, author, date and output format. For this lesson we're going to concentrate on the TEXT in the main BODY of your document. You're going to learn some syntax for how to format parts of your text in different ways.

For now we'll ignore the part in between the 3 backticks `{r} xxxxxxxxxx` which may be highlighted in a light grey box. This is a chunk of R code that we'll be discussing later.

Let's start with the line that begins with 2 hashtags ## followed by the text R Markdown. The hashtags are markdown SYNTAX used to indicate that the text that follows is a HEADER. The number of hashtags indicates what level the header should be. For example, 2 hashtags ## indicate that the text R Markdown should be formatted and treated as a level 2 header.

Let's look at an overview of R Markdown syntax. In RStudio, go to HELP/R Markdown Quick Reference or look at the Markdown Cheatsheets – this opens a special HELP window at the lower right. Take a few moments to review the information and examples provided.

You can also learn more at the R Markdown website for "Markdown Basics" http://rmarkdown.rstudio.com/lesson-8.html which links you to the more detailed information on the markdown syntax for use by Pandoc http://rmarkdown.rstudio.com/authoring_pandoc_markdown.html

You will remember I mentioned Pandoc in Module 1 as the "universal document converter". When we process or compile the R Markdown document using the R Markdown package in R, it uses Pandoc to process the document into the final desired formats. As such, it is important to understand how the formatting syntax works so that Pandoc understands it.

Let's try out some of the basic markdown syntax to get you started formatting your text the way you want it.

Above the ## R Markdown, let's add a level 1 header and then below that we'll also add a level 3 header.

```
# This is a level 1 header
```

```
### This is a level 3 header
```

Click KNIT to HTML and see the result in the viewer.

If you scroll down a little further, you'll notice that there is a web-address URL shown between <> the less than, greater than symbols. This is one way to add in a web link. The use of the <> symbols is a short hand or abbreviated HTML approach, which is allowed. However, the markdown syntax for adding in a weblink is to put the word or words you want to highlight between square brackets [] followed immediately by the web-address URL in between parentheses (). Let's add this line to your document.

Here is a link to GOOGLE.

Click KNIT to HTML and see the results. From the viewer window test out the weblink – it should open a browser and take you to Google.

In the next paragraph the word KNIT has 2 asterisks ** both at the beginning and at the end of the word. These are used for emphasis. Two asterisks are used for strong emphasis, which results in the work being formatted in BOLD. If we use only 1 asterisk, the word is shown in italics. You can also use one or two underscores _ to emphasize a word.

Let's add two more lines of text to test this out.

Here is a word in **bold** and another word in **bold**.

Here is a word in *italics* and another word in *italics*.

Notice that the asterisk or underscore must come immediately before and after the word with no spaces in between for the syntax to work.

Go ahead and click KNIT to HTML to see the results.

Another kind of emphasis can also be done using backtick `marks before and after a word or series of words. This is usually done to highlight text that refers to computer code or a command. The text placed between two backtick marks` is usually formatted in a non-proportional font and is sometimes highlighted in a different color, like light grey.

Let's add this line of text and put the name of the R markdown package in between 2 backtick marks.

When we compile our document we are using the `rmarkdown` R package.

You can extend this concept by highlighting a whole block of text to be shown in non-proportional font between 3 backticks on separate lines at the beginning and end of the text block you want to highlight. For example, if we wanted to show an example of 2 lines of R code, we could type in the following.

Here are some example R commands:

```
2+2
mean(c(1,2,3,4,5))
```

Click KNIT to HTML and see the result

NOTE: Putting text between 3 backticks only changes the formatting of that text. This looks very similar to an R code chunk but doesn't have the {r} after the 1st 3 backticks. We'll cover R code chunks later.

Let's also make a bulleted list. We can make non-numbered bullets using asterisks and dashes and the plus symbol. Try the following list. To make a line indented, you must add 4 spaces. To indent twice, add 8 spaces and so on. This is the 4-space rule.

Here is an example of a non-numbered list:

- Breakfast
  - food
    - eggs
    - toast
    - bacon
  - drink
    - apple juice
- Lunch
  - taco
- Dinner
  - baked chicken
  - broccoli
  - rice

We can make this same list numbered, but simply using numbers or letters.

Here is an example of a numbered list:

1. Breakfast
   a. food
      i. eggs
      ii. toast
      iii. bacon
   b. drink
      i. apple juice
2. Lunch
   a. taco
3. Dinner
   a. baked chicken
   b. broccoli
   c. rice

Again, KNIT to HTML and view the results

You can also format quotes within R markdown using blockquotes which are highlighted by beginning each line in the quote with a > greater than symbol.

Here are some examples to try.

You can also do blockquotes:

> This is a block quote. This paragraph has two lines.
>
> 1. This is a list inside a block quote.
> 2. Second item.

You can also nest blockquotes - you need a space in between the 2 greater than symbols >'s.

> This is a block quote. This paragraph has two lines.

> This text is nested

To indent code within a blockquote, you need to add 5 spaces after the greater than symbol >.

```
2+2
mean(c(1,2,3,4,5))
```

KNIT to HTML and see how these come out.

Also try KNIT to WORD and (optionally) KNIT to PDF if you have LaTeX installed to see how each of these various text formatting syntaxes appear in the final document in each document format. You'll notice that each document format (HTML, DOC and PDF) render each type of text formatting slightly different – for example, notice how the blockquotes look different in HTML and DOC and PDF formats.

You can learn a lot more about Pandoc markdown at http://rmarkdown.rstudio.com/authoring_pandoc_markdown.html

Now that we've made a bunch of changes, let's remember to stage, commit and push our changes up to your Github account.

Open Git Bash and change to the directory for your Github repository created for "Module2_rmd1" – so go to:

```
C:\RepTemplates\Module2_rmd1
```

Once in that directory, type in the following 4 Git commands to check the status of your local files compared to your Github cloud repository; add or stage the modified files; commit your changes; and then push the changes to your Github cloud repository.

```
git status
git add .
git commit -m "changes to rmd file"
git push
```

Chapter 8

Document Elements

8.1 Script Intro

For this lesson we're going to continue working with the same R markdown document "module2_rmd1.Rmd". So, If you need to, go ahead and log in to your Github account, open RStudio, and open the RStudio project for "Module2_rmd1".

So far you've explored the YAML header and tried out several R markdown syntax markings to change the formatting of text in your document. Next we're going to see how to insert other objects and elements within your document. In this lesson you'll learn how to insert:

- Figures; images; photos; or pictures
- Tables
- Videos or animations
- Equations
- and footnotes

8.2 Figures

Take a look at the bottom of your current R markdown file. The last section of this current document contains a chunk of R code that makes a plot of the pressure dataset which is built into the base R software.

Let's take a quick look at this built-in dataset. Go to the Console windows (bottom left) and type in

```
data1 <- pressure
```

which creates an object called "data1" that is assigned a copy of the built-in pressure dataset. Then click on the Environment TAB in the upper right window – click on the little table icon on the right – this opens the dataset in a viewer window at the top left. As you can see there are 2 columns of data and 19 rows for 19 data points of temperature and pressure. You can get more detailed information on this built-in dataset by looking it up in the HELP window. This brings up a help page on the pressure dataset which says that the pressure datasets is "Data on the relation between temperature in degrees Celsius and vapor pressure of mercury in millimeters (of mercury)."

So, in the R chunk shown here, there is only 1 line of code between backticks `{r}` ending with 3 more backticks

This line of code makes a scatterplot of the pressure (along the Y vertical axis) against temperature (along the X horizontal axis).

```
plot(pressure)
```

By running this R code chunk, a scatterplot is created and then inserted in the document where this code was specified.

By the way, we can control the size of the figure using code "chunk options". You can learn more about code chunk options at Yihui Xie's website for the knitr package at https://yihui.name/knitr/options/

Let's change the figure width and height using the fig.width and fig.height chunk options. These options come after the r in the {r} curly brackets. The current R code chunk has the following

```
{r pressure, echo=FALSE}
```

Let's explore each piece.

The curly brackets indicate that this is a code chunk and the r indicates that the code will be R code.

The word pressure is a label of the code chunk. This is helpful for debugging later. When you get an error the R Markdown log will show in which code chunk the error occurred. Giving them descriptive names will help you keep track.

Let's look at the R Markdown TAB at the bottom left and review the log of when we last rendered or compiled the document – when we last ran KNIT.

After the chunk label and a comma, there is already 1 chunk option saying echo=FALSE. This tells R markdown to hide this code chunk and not "echo" it or show it in the final document. So, if you look back at the documents you've made so far – all you see is the plot of the pressure dataset – you do not see the R code.

At the very end of the document, let's add another code chunk and alter this next figure slightly. Click the button in the editor window with a little green C with a + plus, click the down arrow for insert an R code chunk. When you click this, it automatically enters the 3 backticks with the r in curly brackets `{r}` `followed by a blank line and then 3 more backticks` ending the code chunk.

Go to the blank line and type in

```
plot(pressure)
```

Now go back to the 1st line, let's add the following options. We'll add a new label "pressure2" since each code chunk has to have a unique chunk name. We'll also add fig.width=5 and fig.height=5 which does 2 things – it will make the horizontal and vertical dimensions of the figure to be the same and should render a figure approximately 5 inches by 5 inches – see the chunk options described on Yihui Xie's website.

```
{r pressure2, fig.width=5, fig.height=5}
```

Save the RMD file and KNIT to HTML as see the differences between the 2 plots. Not only will the plot sizes be different, in the 2nd one we left out the echo=FALSE so the R code was shown right before the 2nd plot. If you want to hide this code, go back and add echo=FALSE to the chunk options.

```
{r pressure2, fig.width=5, fig.height=5, echo=FALSE}
```

In RStudio 1.1.x, you'll also notice that in the editor window, within the code chunk at the far right there are a couple of little icons. Click on the one that looks like a gear and another window pops up that shows various code chunk options available including

- The name of the code chunk
- What kind of output you want to show (with or without the code – setting echo-TRUE or FALSE)
- You can also turn on or off warnings and messages
- At the bottom you can also set the figure width and height – try changing this to 4 inches and watch the changes in your R markdown file.

There is also a green arrow > that you can click which will "run" the R code as if you were running R from the command line or from a standalone R script. If you click this, the plot(pressure) command will be executed and the plot will be generated in the Plots window.

8.3 Images

We do not have to use R code to add figures in your document. You can also bring in external images and pictures. Let's use the picture called sunstar.png – the read ahead materials had instructions on how to download this picture. For now, let's put this picture in the same directory as your R markdown document. If you want to see this in your document, you can use the simple R markdown syntax `![alt text](filename)`

Let's create a new header in our document

```
## Insert Images
```

Here is an image inserted

```
![sunstar](sunstar.png)
```

You can also insert images off the web by linking directly to them via their web-address URL

Here is the R logo

```
![Rlogo](https://www.r-project.org/logo/Rlogo.svg)
```

NOTE!! The image with the web-address URL will NOT compile correctly for the KNIT to WORD and KNIT to PDF since the image is not stored locally. If you want this image for a DOC or PDF formatted file, you will need to download the image and store it locally when you "KNIT" the final document. Then use the same syntax as you just did for sunstar.png. So, delete this section before you KNIT to WORD or KNIT to PDF.

8.4 Tables

Doing tables in with R markdown or rather using Pandoc's markdown can be done but is a tedious process for even simple tables. There are numerous examples provided at http://rmarkdown.rstudio.com/authoring_pandoc_markdown.html#tables

Typically, it is easier to use R code to generate a table. The best function for making tables using R markdown is the kable function from the knitr package, see https://yihui.name/knitr/ It may also help to install and learn more about the printr package also which improves the formatting of knitr output, see https://yihui.name/printr/

Let's try a simple table of another built-in dataset, cars. Look at the help pages for cars. The cars dataset has 50 observations or rows and 2 columns with the 1st column having data on the speed of the cars and the 2nd column having data on the stopping distance. We can use the head function from base R to look at the top 6 rows of the cars dataset. The cars dataset is in an R object called a "data.frame" which the kable function handles like a table. So the following R chunk will make a table of the top 6 rows of the cars dataset.

Let's create another new header in our document

```
## Insert Tables
```

```
\begin{tabular}{r|r}
\hline
speed & dist\\
\hline
4 & 2\\
\hline
4 & 10\\
\hline
```

```
7 & 4\\
\hline
7 & 22\\
\hline
8 & 16\\
\hline
9 & 10\\
\hline
\end{tabular}
```

Let me explain the R code a little bit. Since the kable function comes from the knitr R package, it is good practice to list the package followed by 2 colons followed by the function so it is easy for you or anyone else reading your R code to know which package the function came from. There are literally tens of thousands of R packages and some have the same function name which do entirely different things. So, to avoid confusion it is always a good idea to list use the `package::function()` when using a function in R.

So this code says to extract the "head" or top 6 rows of the cars dataset and use the kable function from the knitr package to print out a table in the final document.

Go ahead and save your document and click KNIT to HTML to see the result. Feel free to also try KNIT to WORD and KNIT to PDF to see how the table appears in each of those formats.

We can add other options to the kable function – like adding a table caption. Let's add the option caption = "Top 6 Rows of Cars Dataset"

```
knitr::kable(head(cars),
             caption = "Top 6 Rows of Cars Dataset")
```

You'll notice this added a caption or title at the top of the table in each of the formats (HTML, DOC and PDF).

8.5 Videos and animations

Suppose you have a video you want to include in your document. This doesn't make sense for a printed final document, but these will work for a HTML document, you can view an embedded video. Let's work again with the sunstar graphic as an animated GIF and as a MP4 video. The read ahead materials had instructions on how to download both the animated GIF "sunstar.gif" and video "sunstar.mp4" into a subfolder in your project called "sunstar". Since these are in a directory called "sunstar" in your project folder, you need to include the folder name when you list the filename. The syntax for embedding videos and animated GIFs is similar to how images are inserted.

```
## Insert an Animated GIF and Video

![sunstar](sunstar/sunstar.gif)

![sunstar](sunstar/sunstar.mp4)
```

KNIT to HTML to see the results. You may need to open the saved html file in a separate browser window to see the embedded MP4 video. These videos will not work if you KNIT to WORD – the document will compile but these sections will be blank. If you try running KNIT to PDF you will get an error. So, Videos and Animated GIFs only work in HTML format – at least using this approach. New functions and methods are created daily, so there is probably a way to embed videos and animated GIFs in other formats if you look for it.

In the final HTML document, you'll notice that the animated GIF plays over and over again. But the MP4 file is embedded with a video viewer that you can click play to see the video. This is only a simple introduction to videos. This is only scratching the surface. There is a R package called vebmedr which allows embedding of videos like YouTube in your HTML documents. Learn more at http://ijlyttle.github.io/vembedr/

8.6 Equations

For those of you not interested in typing math equations in your documents, you can skip this section. But if you are interested, know that you can embed equations using LaTeX syntax. For example, suppose we wanted to write out the equation for a simple linear regression model. We would embed the LaTeX formatting inside a block beginning and ending with 2 dollar signs `$$`

```
## Insert an equation

$$ Y = \beta_0 + \beta_1x $$
```

There are hundreds of websites with information, tutorials and help online for formatting equations using LaTeX. One example is https://www.sharelatex.com/learn/Mathematical_expressions if you want to learn more.

8.7 Footnotes

Finally, you might want to include a footnote in your document. The syntax for inserting a footnote is square brackets with an up arrow ^ inserted. You can add footnotes in one of two ways.

First – you can simply add the notation where you want to add a footnote in the place in the text where you want the index number. And then at the END of the document you have to provide the content you want displayed with the footnote index.

Second, you can use what is called an inline note. With an inline note you don't have to remember to add the footnote references at the end. Let's add the following to your document using both methods.

```
## Insert text with some footnotes

Here is a footnote reference,[^1] and another.[^longnote]

Here is an inline note.^[Inlines notes are easier to write, since you don't have to pick an identifier a

[^1]: Here is the footnote.
[^longnote]: Here's one with multiple blocks.
```

Save your document and KNIT to HTML to view your document.

Finally, let's make sure to back everything up and save your changes to your Github repository.

Open Git Bash and change to the directory for your Github repository created for "Module2_rmd1" – so go to:

```
C:\RepTemplates\Module2_rmd1
```

Once in that directory, type in the following 4 Git commands to check the status of your local files compared to your Github cloud repository; add or stage the modified files; commit your changes; and then push the changes to your Github cloud repository.

```
git status
git add .
git commit -m "inserting multiple elements to my RMD file"
git push
```

Chapter 9

Presentation Formats

9.1 Intro Script

For this lesson we're going to create some presentation formats. We'll make slides using either ioslides or slidy formats which make HTML based slides. If you have LaTeX installed you can also try making PDF slides in Beamer format. We'll also check out a newer slide template format called revealjs which can be accessed by installing a new R package.

For now, let's keep using the same Github repository and the same RStudio project for "Module2_rmd1". Go ahead and open this project in RStudio. We will create your new R markdown files for the different presentation formats in this repository. This repository should be saved on your local drive at

`C:\RepTemplates\Module2_rmd1`

In RStudio, click File/Open Project and find the .Rproj file and open it.

9.2 Ioslides

Once you have your project open in RStudio, go ahead and create a new R Markdown file. But this time instead of creating a new document, select Presentation and the ioslides format. Type in a title like "Module 2 – ioslides" and click OK to create the new ioslides document.

Take a quick look at the YAML header. The keywords title, author and date are all very similar to the YAML header for the HTML document. But notice that the output has changed to ioslides_presentation. This is the line that tells R markdown that this document will be rendered as an ioslides HTML formatted presentation.

You'll notice that this template for the HTML ioslides format has many similar elements and formatting as the basic R markdown HTML document we created in an earlier lesson.

However, this time in ioslides the 2 hashtags ## for level 2 headers are the titles for each new slide. So, this basic ioslides template contains 4 slides. The 1st slide has some basic text with 2 paragraphs. The 2nd slide has a list with 3 bullets. The 3rd slide shows a simple summary of the cars dataset. The 4th slide shows a slide containing the plot of the pressure dataset.

Go ahead and KNIT to HTML (ioslides) to see the final slides. Save the file to your local drive with a filename like "module2_ioslides.Rmd"

You'll notice that there are actually 5 total slides. The 1st slide is your title slide and was generated using the information contained in the YAML header – the other 4 slides created using the level 2 headers with

the 2 hashtags ## come after this title slide. In your YAML header, try changing your title, author or date and KNIT to HTML (ioslides) the slides again to see how it changes your title slide.

Open the saved HTML slides in a browser so you can view these outside of the RStudio viewer. A nice feature of the ioslides format is the ability to view the slides in a normal width/height ratio or in a wide format. While viewing the slides in a browser, type the letter w to switch from a normal to a wide view and back again.

You can also view the slides in a full screen mode by typing the letter f to toggle from full screen to browser window view.

9.3 Slidy

Let's also try making these slides in the Slidy HTML format as well. We can do this one of two ways. We can create a new R Markdown file and choose Presentations and the Slidy format. Or since we have the ioslides formatted file open already, we can simply KNIT to HTML (Slidy) and the output format will update automatically in the YAML header and your Slidy formatted slides will be created also. BEFORE we do this, go ahead and save the file as another filename to keep them separate from the ioslides format. Click File/SaveAs and save as "Module2_slidy.Rmd". After you save your file as "Module2_slidy", click KNIT to HTML (Slidy) and see these slides in the different format for Slidy HTML slides.

Take a few moments now and choose one of these 2 R Markdown files – either for ioslides or Slidy – and spend some time adding a few new slides inserting an image or video or an equation and view the results. You can use the text and R Markdown syntax we created earlier for the R markdown HTML document. Everything should work as expected in HTML except for the footnotes which are not supported in ioslides or Slidy but do work in the Beamer PDF format. You always have to check the documentation for each format as some options are not supported across formats – like animations or videos not working in PDF or DOC formats.

It is a good idea to read through the documentation for each R Markdown format at http://rmarkdown.rstudio.com/formats.html

For now, let's try adding 3 slides to our ioslides (or Slidy) presentation here based on what we did earlier in the R Markdown HTML document. [NOTE TO SELF – demonstrate this on the video in both ioslides and Slidy format for the viewers]

```
## A slide with an inserted image

Here is an image inserted

![sunstar](sunstar.png)

## A slide with a table

\begin{table}

\caption{\label{tab:unnamed-chunk-4}Top 6 Rows of Cars Dataset}
\centering
\begin{tabular}[t]{r|r}
\hline
speed & dist\\
\hline
4 & 2\\
\hline
4 & 10\\
```

```
\hline
7 & 4\\
\hline
7 & 22\\
\hline
8 & 16\\
\hline
9 & 10\\
\hline
\end{tabular}
\end{table}

## A slide with an equation

A simple linear regression equation

$$ Y = \beta_0 + \beta_1x $$

Notice that the slide with the sunstar image shows the word sunstar below the image. This is because we

## A slide with an inserted image

Here is an image inserted

![](sunstar.png)
```

KNIT to HTML (ioslides) or (Slidy) whichever you prefer and view the changes. Save your file.

Suppose we want to center the image on this slide. To the header (title) for this slide we can add the following layout option as follows.

```
## A slide with an inserted image {.flexbox .vcenter}
```

You can learn more about these options for the ioslides format at http://rmarkdown.rstudio.com/ioslides_presentation_format.html

You can also make 2 columns using an ioslide option {.columns-2} in the slide header. Try adding this slide to your ioslides R markdown file.

```
## A slide with 2 columns an image and a bulleted list {.columns-2}

![](sunstar.png)

- bullet 1
- bullet 2
- bullet 3
```

KNIT to HTML (ioslides) to see the results. NOTE: This 2-column option shown here will NOT work for Slidy NOR for the Beamer PDF formats.

9.4 Beamer

Optionally, if you have LaTeX installed, save your file as "Module2_beamer" and try KNIT to PDF (Beamer) to see the Beamer PDF slide format. Again some options and syntax may work differently in the Beamer

format. Read through the Beamer documentation to learn more about what is possible. http://rmarkdown.rstudio.com/beamer_presentation_format.html

For the most part you can switch between the different slide formats without too much trouble but you'll notice that the 2-columns option only worked for the ioslides format.

9.5 RevealJS

Another HTML presentation format gaining popularity is the revealjs format. To use this format, we first have to install the R package for this slide presentation format.

See the details at http://rmarkdown.rstudio.com/revealjs_presentation_format.html

Also learn more at http://lab.hakim.se/reveal-js/#/ and

https://github.com/hakimel/reveal.js/

In RStudio, click on Tools/Install Package, make sure the CRAN repository is selected and type in revealjs. Alternatively, in the Console window you can type in the R command install.packages with the name of the package revealjs put in quotes between the parentheses

```
install.packages("revealjs")
```

Once the revealjs package is installed, we can now access a revealjs slide template by going to File/New File/R Markdown file – choose From Templates and look for "Reveal.js Presentation (HTML)".

By the way, any other R packages you have installed that have R markdown templates available will be listed here in the Templates window. Later in this course you will learn about more R packages and templates that are available and you will learn how to create an R package with your own template!

For now, let's go ahead and create a new R markdown file using the revealjs presentation template. This will create a set of slides similar to those we just did using ioslides or slidy.

You will notice that when the new R markdown file was created, this time the YAML header is slightly different – only the title and output were defined. Let's go ahead and add back author and date to your YAML header.

Click KNIT to revealjs_presentation to see the resulting HTML slides. These slides are produced with a default slide transition animation.

Let's try changing the default slide transition to "zoom" (see list at http://rmarkdown.rstudio.com/revealjs_presentation_format.html#slide_transitions)

To change a parameter in the YAML header we will move the revealjs::revealjs_presentation after output: to a new line and indent it 2 spaces. Then we add a colon: and on a 3rd new line also indented another 2 spaces (4 space total) we add the transition option we want as follows.

```
output:
  revealjs::revealjs_presentation:
    transition: zoom
```

Save your file and KNIT to revealjs_presentation to see the transition changes.

The revealjs format also utilizes themes to apply some styling to the slides using color. See the example provided at http://rmarkdown.rstudio.com/revealjs_presentation_format.html#appearance_and_style

Let's add a few more lines to your YAML header and give your slides some color. We'll set the theme to solarized, the highlight to kate and the center option to true to give us this YAML header

```
output:
  revealjs::revealjs_presentation:
    transition: zoom
    theme: solarized
    highlight: kate
    center: true
```

All of the keywords after revealjs::revealjs_presentation: are specific parameters or options available in the revealjs_presentation format from the revealjs package.

Save your file again. KNIT to revealjs_presentation to see your updated slides with a new color and format.

Take a few moments and try different options and settings by modifying the YAML header to experiment with making changes that affect your final presentation. Refer back to the revealjs format to see what options are available, http://rmarkdown.rstudio.com/revealjs_presentation_format.html

9.6 Backup

Finally, be sure to back everything up and save your changes to your Github repository.

Open Git Bash and change to the directory for your Github repository created for "Module2_rmd1" – go to:

```
C:\RepTemplates\Module2_rmd1
```

Once in that directory, type in the following 4 Git commands to check the status of your local files compared to your Github cloud repository; add or stage the modified files; commit your changes; and then push the changes to your Github cloud repository.

```
git status
git add .
git commit -m "created multiple slide presentation formats"
git push
```

Chapter 10

Book Format

10.1 Intro Script

In this lesson, I'm going to show you how to create a book using R markdown.

10.2 Bookdown

For this lesson we will be working with the bookdown demo repository. Go to https://github.com/rstudio/bookdown-demo. To get started, we're going to download this whole repository as a ZIP file. Click on the green Clone or Download button and Download as a ZIP file into your local directory for this course

`C:\RepResearchCourse`

Then unzip this file into this folder. When you get done you should have all of the files from the bookdown-demo repository in this directory

`C:\RepResearchCourse\bookdown-demo-master`

In this directory there is ALREADY an .Rproj file – so this repository already comes with an RStudio project ready to go.

Start RStudio and Click Open Project and select bookdown-demo.Rproj in this directory

`C:\RepResearchCourse\bookdown-demo-master`

The first thing to note is that in the upper right window, there is now a TAB that says Build. This TAB will be used to "Build" the final book. However, you will notice that there is NO GIT TAB – this is because we need to re-establish a link using GIT back to your Github account. To do this you may want to refer to Jenny Bryan's Happy Git and Github for user book – Chapter 18 on how to connect to Github when you have an Existing Project and you are connecting to Github last

http://happygitwithr.com/existing-github-last.html

To turn GIT on in RStudio, click on Tools/Project options – click on Git/SVN. At the moment, the Version Control system says "none" – click on the arrow and choose Git. Another window will pop open asking if you want to initialize a new GIT repository for this project – click YES. Then it will say you need to restart RStudio – click YES you want to do this now.

After RStudio restarts, you'll notice that you now have the GIT TAB in the window at the upper right next to the Build TAB. But we're not done – we still need to establish a connection between your local "repository" (the directory with all of the files and RStudio project for the bookdown-demo) and your Github cloud account.

Before we connect to the cloud, you need to create a new repository in your Github account. Log into your Github account and Click New Repository. To avoid confusion, go ahead and name your new repository the same as your RStudio project "bookdown_demo". However, DO NOT initialize it with a README. Click Create Repository. Your repository is created but there are NO FILES in it – yet.

Let's connect using GIT. Open Git Bash and change directory to

```
C:\RepResearchCourse\bookdown-demo-master
```

Let's stage – add all of the files

```
git add .
```

Next let's commit these files

```
git commit -m "add files for bookdown demo"
```

BUT BEFORE we can PUSH these up to the cloud, we need to connect this directory to your cloud account. To do this – go back to the Github repository you just created and click the "copy to clipboard" button to get the URL for the repository from the 1st line you can see in your browser for the "Quick Setup"

Now go back to GIT BASH and type in "git remote add origin" followed by the URL you just copied to your clipboard.

git remote add origin https://github.com/melindahiggins2000/bookdown_demo.git

The next step is to PUSH your content. However, the syntax is slightly different since this is our first time connecting the Github account for this repository to your local drive. According to Jenny Bryan we need to "cement the tracking relationship between your GitHub repository and the local repo by pushing and setting the "upstream" remote". To so this type in the following GIT command.

```
git push -u origin master
```

This will take a minute or to two to run. When it finishes, go back and refresh your Github repository and you should now see all of the bookdown_demo files in your cloud account.

Now that you have the basic files and have everything backed up to your Github account and synched up, let's dive into all of these files and see how everything works together.

Go back to RStudio, let's look at the files. The .gitignore file "specifies intentionally untracked files that Git should ignore." See the GIT documentation at https://git-scm.com/docs/gitignore

RStudio created a .Rhistory file (which is empty at the moment which is ok).

Go ahead and click on the bookdown-demo.Rproj. This opens a window showing the various Project Settings for this RStudio project. This is the same window that pops up when you click on Tools/Project Options. The key elements to notice in this window are the settings in the Git/SVN tab which should show that YES you are using Version Control and the URL of your Github repository is now listed. Also see the settings under the Build Tools. Here the Project Build tools should show Website and we'll keep all of the other default settings for (Project Root) for the Site Directory, all formats for book output formats, and the box is checked for preview book after building and re-knit current preview when supporting files change.

There are 3 YAML files. YAML header information can be kept in a separate file instead of at the top of an R Markdown file which is what has been done here.

`.travis.yml` – this YAML file provides some settings if you choose to set-up TRAVIS CI for "continuous integration" – learn more about TRAVIS CI at https://travis-ci.org/ - this is beyond the scope of this course, but may be worthwhile learning more if you plan to write computer programs and scripts for data analysis.

`_bookdown.yml` – this YAML file contains information about the book, like the filename for the book and chapter names.

`_output.yml` – this file has various options for compiling the book in multiple formats like gitbook (which is an HTML format), PDF book and even an EPUB book format.

There are 2 "SHELL" script files (_build.sh and _deploy.sh) we will ignore for now

There are 2 CSS (cascading style sheets) files (style.css and toc.css) which contain codes for formatting the resulting HTML format for the book. We will not change these files for now.

There is a DESCRIPTION and LICENSE file we will ignore for now. Feel free to view these files and learn more about them in the Bookdown book https://bookdown.org/yihui/bookdown/ You may want to update the LICENSE file in the future if you plan on distributing your book online under a different LICENSE.

There is also a README.md file which you can edit later if you wish to do so.

There is a file called preamble.tex which has some important formatting and setup information for compiling the book to PDF format using LaTeX. We can ignore this for now also.

The remaining 7 files are all R Markdown RMD files. The introductory chapter is defined by index.Rmd. Go ahead and open this file.

At the beginning of this index.Rmd file there is another YAML header. For now, go ahead and change the author to your name.

Notice the YAML keyword for bibliography – this line sets up the references used or created when this book is compiled or rendered. There are 2 BIB files listed book.bib which is a file in your repository and packages.bib which does not exist yet. Packages.bib will be created when the book is built. These BIB files are BibTeX formatted files – learn more at http://www.bibtex.org/

After this introductory chapter there are 6 chapters each named beginning with a number and a short description. Take a few minutes and open and look at each of these RMD files. When you look at these RMD files you will notice that there is NO YAML at the top of these files. And each RMD file starts with a level 1 header indicated by 1 hastag #.

Let's try building the book into an HTML book. Go to the Build TAB at the top right. Click the down arrow for Build Book and choose bookdown::gitbook. You may be prompted to install a package as needed. Just say yes.

If all goes well, your book will appear in the Viewer window. Let's open a File Explorer window to see the new files created.

You'll notice that the file packages.bib was created. This file now has the BibTeX for all of the citations for each R package used to build this book.

There are also 2 new folders that were created

`_book` – this directory has all of the HTML files to see the book in HTML format. If you go to this folder and double click on index.html, you will be able to see the book in a browser window.

`_bookdown_files` – this folder has a few more subfolders and an image that was produced by one of R code chunks that made a plot in the book.

If you have LaTeX installed, try making a PDF copy of the book – go to Build Book and click bookdown::pdf_book

When this finishes compiling, there will now be a new file in the _book directory called bookdown-demo.pdf.

Let's try one more change. Open the _bookdown.yml YAML file. After the chapter_name line, add a line with this option for the output directory

```
output_dir: docs
```

Click Build Book to bookdown::gitbook to rebuild the book into an HTML format. This time instead of all of the HTML files being placed in the _book directory, a new directory is created called docs and all of the HTML and supporting files are placed here. There is a reason we created the docs directory - I'll show you why we want to save these HTML files into the docs directory in just a minute.

Let's go ahead and back up all of these changes and sync everything up to your Github repository.

Open Git Bash and change to the directory for your Github repository created for "bookdown-demo" – go to:

`C:\RepResearchCourse\bookdown-demo-master`

Once in that directory, type in the following 4 Git commands to check the status of your local files compared to your Github cloud repository; add or stage the modified files; commit your changes; and then push the changes to your Github cloud repository.

```
git status
git add .
git commit -m "compiled bookdown demo"
git push
```

Open your browser and go to your bookdown_demo repository. Check to make sure your changes were pushed to your Github cloud repository. Now at the top of your repository at the far right there is a gear-shaped icon for the settings of your repository. Click on this icon.

Scroll about half way down the page to the section titled Github Pages. We're going to set up a "webpage" for your book. Click on the button for Source – change it from "None" to "master branch/docs folder" and click Save. After you click save, the page will refresh and if you scroll back down to the Github Pages section of the setting page, you'll see the URL for your book's website. It should look something like

https://USERNAME.github.io/bookdown_demo/

https://melindahiggins2000.github.io/bookdown_demo/

Click on this URL and you should now see your book online!! Congratulations you have now created a book and served it online. Plus, since we have everything connected using Git from your local drive to your Github cloud account, AND we have it setup to publish to your /docs directory - then any time you make changes to your book and sync the files up using Git, your online book will be updated also!!

Take a few moments and try making changes to the different RMD files for each chapter by adding text, inserting some R code, adding a plot or an image and recompile the book, resync the files to Github and view the changes.

We have only barely scratched the surface on creating a book using the bookdown package. But I encourage you to learn more because a BOOK format is a wonderful way to organize multiple related documents together in one nice neat package even if your intention was not to write a "book" per se, but to simply organize a multi-component report.

I highly recommend Yihui Xie's bookdown book at https://bookdown.org/yihui/bookdown/

I also recommend checking out other books created using the bookdown package at https://bookdown.org/

You may also be interested to learn more about Gitbook as well at https://www.gitbook.com/

Chapter 11

Exercise - Air Quality Report

11.1 Intro Script

For your graded assignment, you will need to create an R Markdown file with each of the following 8 elements and then answer questions about the resulting document.

Create an R Markdown file with the following 8 elements:

1. Title "The Air Quality Dataset"
2. Author: "your name"
3. A level 2 header "Summary of Air Quality Dataset" followed by
4. this paragraph. In this paragraph provide the syntax to:
    - show "airquality" in non-proportional font using backticks;
    - create a non-numbered bulleted list for the 6 variables in the dataset
    - put each variable name in bold using double asterisks
    - put everything in parentheses in italics by placing a single underscore immediately before and after the opening and closing parentheses
    - notice that the 1st sentence contains an inline footnote which should appear at the bottom of your document when compiled

This exercise will be working with the built-in airquality dataset.[1] This dataset contains 154 daily air quality measurements in New York from May 1, 1973 (a Tuesday) to September 30, 1973. The dataset contains 6 variables:

`Ozone`: Mean ozone in parts per billion (ppb) from 1300 to 1500 hours at Roosevelt Island;

`Solar.R`: Solar radiation in Langleys (lang) in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park;

`Wind`: Average wind speed in miles per hour (mph) at 0700 and 1000 hours at LaGuardia Airport;

`Temp`: Maximum daily temperature in degrees Fahrenheit (oF) at La Guardia Airport;

`Month`: numeric month (1-12)

`Day`: numeric Day of the month (1-31)

[1] Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983) Graphical Methods for Data Analysis. Belmont, CA: Wadsworth.

5. a second level 3 header “Table of Top of the Air Quality Dataset” followed by
6. a table of the head of the airquality dataset – insert an R code chunk with the following code

```
knitr::kable(head(airquality))
```

7. a third level 3 header “Plot of Ozone by Temperature – Air Quality Dataset” followed by
8. a plot of the temperature and ozone from the built-in airquality dataset – insert an R code chunk with the following code

```
plot(airquality$Temp, airquality$Ozone,
        main="Airquality: Ozone by Temperature")
```

KNIT the document to HTML and KNIT to WORD

Part II

Appendix

Appendix A

First appendix section

We have finished a nice book .

some random text

Appendix B

another appendix section

We have finished a nice book .

some random text

Bibliography

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., and Chang, W. (2018). *rmarkdown: Dynamic Documents for R*. http://rmarkdown.rstudio.com, https://github.com/rstudio/rmarkdown.

Ismay, C. and Chunn, J. (2017). *fivethirtyeight: Data and Code Behind the Stories and Interactives at 'FiveThirtyEight'*. R package version 0.3.0.

Miller, I. W., Epstein, N. B., Bishop, D. S., and Keitner, G. I. (1985). The mcmaster family assessment device: Reliability and validity*. *Journal of Marital and Family Therapy*, 11(4):345–356.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Wickham, H. and Chang, W. (2016). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 2.2.1.

Wickham, H., Francois, R., Henry, L., and Müller, K. (2017). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.4.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2017a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.5.10.

Xie, Y. (2017b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.18.

Xie, Y. (2017c). *printr: Automatically Print R Objects to Appropriate Formats According to the 'knitr' Output Format*. R package version 0.1.

Index

Nice Book, 65, 67

random, 65, 67